HEADEND PROJECT REPORT

1. Introduction

The system description appearing in this document has been generated in a process of reverse engineering of device memory contents. Since the original naming conventions are not retained within the machine code, we had to devise our own conventions. As a result, names of subroutines, files, commands etc. used in this document are not necessarily these used by the original system implementers.

The present document describes:

- The card's hardware and its security implications
- The memory map and its subdivisions
- The file system
- The system commands

2. Hardware

The SGS Thomson ST16CF54 CPU has a standard Motorola 6805 architecture, with the following differences:

- Two additional instruction: Opcode 9EH, (mnemonic TSA) which stores the contents of the stack pointer (SP) to the Accumulator, and Opcode 42H, (mnemonic MUL) which multiplies the X register by the Accumulator.
- Inclusion of a Modulo Arithmetic Processor unit, addressed through @@@@@
- Inclusion of hardware devices intended to verify and protect the integrity of the system, such as memory access matrix and intrusion sensor. However, these are either not enabled by the device software, or did not have an effect on the reverse engineering effort.

HEADEND PROJECT REPORT

1. Introduction

The system description appearing in this document has been generated in a process of reverse engineering of device memory contents. Since the original naming conventions are not retained within the machine code, we had to devise our own conventions. As a result, names of subroutines, files, commands etc. used in this document are not necessarily these used by the original system implementers.

The present document describes:

- The card's hardware and its security implications
- The memory map and its subdivisions
- The file system
- The system commands

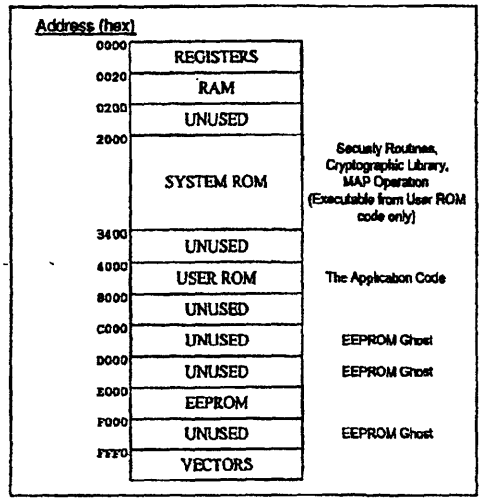
2. Hardware

The SGS Thomson ST16CF54 CPU has a standard Motorola 6805 architecture, with the following differences:

- Two additional instruction: Opcode 9EH, (mnemonic TSA) which stores the contents of the stack pointer (SP) to the Accumulator, and Opcode 42H, (mnemonic MUL) which multiplies the X register by the Accumulator.
- Inclusion of a Modulo Arithmetic Processor unit, addressed through @@@@@
- Inclusion of hardware devices intended to verify and protect the integrity of the system, such as memory access matrix and intrusion sensor. However, these are either not enabled by the device software, or did not have an effect on the reverse engineering effort.

EXHIBIT
98
No. SA CV 03-950 C 02-1178
DOC (JTLx)

The CPU memory map is shown below.



1.1. EEPROM

The CPU has 4 KB of EEPROM. Since the ROM is fixed and the contents of RAM are lost each time the card is removed from the IRD, the EEPROM must be used for permanent storage of all significant data, including personalization and smart card application upgrades (patches):

Address (Hex)	Contents.
E000 to E071	General Data.
E072 to <E030> - 1	Patch Code area.
<E030> to EFFF	Heap (dynamically allocated data).

Please note that different ROM software versions of the card may use different storage areas etc. The following material describes Version 3. Appendix @ includes some notes regarding the earlier Version 2.

1.1.1 General Data

This area contains fixed ID numbers, file records length data and pointers that determine the location of the other areas. See Appendix A for the General Data storage area structure. This is the only EEPROM area whose location is fixed. Conversely, it contains some essential pointers for making the dynamically allocated file system work.

1.1.2. Patches

Patches are accessed through a table starting at fixed location 0xE072. The maximum number of table entry is specified in the General data area. The current design is for 10 entries (patches), but just 5 entries are currently implemented.

Each patch table entry contains three bytes: one byte of patch ID used by the application to invoke the particular patch, and a two-byte pointer to the patch code associated with that ID. The table entries are arranged in two vectors: patches ID vector and patches code pointer vector. The patch code itself is stored in the memory area between 0xE072 + (Max_TblEntries) * 3, and the start of the Heap.

In order to invoke a patch code the application calls a patch switch subroutine that takes the specified patch ID as a parameter. The routine searches the patch table ID vector, and invokes the patch code indicated by the corresponding pointer. If the specified ID is not present, no patch is invoked.

Usually the application invokes patches before some sensitive action is performed. This device lets the broadcaster fix bugs and change the application behavior in the future. Patches are added or their code modified by small routines that arrive "from the air". Revision number is increased each time a change is made in the patches table or code.

1.1.3. The Heap

The Heap is a formatted memory area that is intended for data storage similarly to a file system. Each file can have one or more records that are dynamically allocated on the Heap. There are mechanisms to create, delete, retrieve and update records.

The Heap consists of memory blocks. Each block has two header bytes used to store the block relating data. The first byte is a "block control byte" (BCB) specifying the block type. It defines what kind of data the block contains. There are four possible block types:

BCB value (Hex)	Block type
0x0F	A free bock.
0x07	A valid file record.
0x03	A file record whose update is not completed yet.
0x87	A transaction.

The second header byte is the length of the block data (not counting the two header bytes).

The first byte of each record identifies the type of file to which it belongs. There are 14 file types: 12 files (0x01 to 0x0C) whose records have fixed length and two (0x10 and 0x11) whose records have variable length. Appendix B lists the record structure of the relevant file types.

The data in the files is not encrypted. File data is passed from the smartcard to the IRD, mainly following card insertion / reset. Some of the file data fields placed at the end of the file, however, are "hidden" or "protected", i.e. the standard ROM file handling routines do not transmit these fields. The entitlement information stored in the files is identical for all cards having the same entitlement.

File 01 stores subscriber and IRD related data such as ZIP code, time zone, IRD software version information, pairing information etc. It has just one record, and is the first record in the Heap. File 01 exists on unauthorized card, but most of its fields are zeroed. These fields get their contents during an authorization process. The pairing key is a protected field.

File 02 is important for a decision on providing the Video Key to the IRD. . It has only one record, and probably signals that the subscriber is permitted to receive services from the given broadcaster. It exists on unauthorized and authorized cards.

File 06 consists of data relating to connection to the broadcaster's Head office, such as the card number, phone number of the Head office, date and time when to call it and send purchases report (feedback), black-out (spot) pattern and so on. It also contains two 64-bit keys to decrypt ECM when it is sent to request a Video Key. These two keys are the protected fields of the record. The file exists on an unauthorized card, but all fields apart of the card number have no data. They are written during authorization process. The broadcaster retransmits a command to update this keys pair used to decrypt ECM again and again approximately 10 times an hour (as seen the keys remained the same for more than two months).

File 07 is a hidden file. That means IRD never asks for its content. All its informative fields are protected. Actually this is a set of keys used in the RSA (?) cryptographic routine, excepting a 64-bit number used to generate and check digital signatures in some messages. This file exists on an unauthorized card as is.

File 08 consists of important entitlement information about subscribed channels. This information is the base on which the card makes decision whether to provide or not the Video Key that was obtained from the passed ECM. This file doesn't exist on an unauthorized card and it is created and filled in when the user subscribes for a channel or a package of channels. The data in file 08 is card irrelevant that means two different cards with the same entitlement will have identical files 08.

File 0B contains the important entitlement information about purchased PPV program or event. Like file 08 its data is a base for decision about entitlement for a particular service, but in this case, the service is limited for watching by a short period of time. The data in this file is also card irrelevant.

File 0C is responsible for storage purchasing information, such a subtotal (debit) of all purchased PPV programs, maximal amount of credit, threshold for debit and so on. File 0C is created and filled in during authorization process.

File 11 is an informative file. It contains supplemental information about the purchased PPV program, such as the program title, its start date and time and PPV channel name. All this data is used by IRD together with file 0B to display the purchasing list and probably to remind to the customer about a purchased service start or perform an automatic switch to the program.

2.1. Messages

IRD communicates with the smart card by sending a set of consequent bytes, called a message. The smart card analyzes these incoming bytes and performs actions accordingly. The smart card sends to IRD as a reply a set of consequent bytes, called an answer. Only IRD can initiate the communication and each its message should be responded by the smart card with an answer. The protocol T=1 is used in the system to perform IRD-SC communication (see ISO 7816 for details). This protocol distinguishes between three different block types: information blocks (I-blocks), reception acknowledgment blocks (R-blocks) and system blocks (S-blocks). This document deals only with I-blocks, or more exactly, with data in the information field (APDU) of such blocks. The Node Address field (NAD) is 0x21 (from source node 1 to target node 1). The Protocol Control Byte field (PCB) has a sequence flag that toggles in each new incoming message (once 0, once 0x40). Maximal valid information field (APDU) length is up to 0x64 bytes. APDU actually consists of a command (or set of commands) to card.

2.1.1. Memory Access Messages

Actually this group of messages is not supposed to be sent by IRD, because of two main reasons. First of all, the data in these messages and answers is very sensitive. It can provide free access to the card memory to read, write and erase and software upload. Second, the data is passed unscrambled (plain text), so it can be easily monitored. The access is based on knowledge of an 128-bit long access key that is located at address 0xE020 in EEPROM. The key differs from card to card. This key is an arbitrary binary string, but with the restriction that the number of 1-bits must be between 32 to 96. When the key does not meet this condition, it is overwritten by the default key (located in ROM at 0x7AFC) during the card initialization process. Most likely the reason for this restriction is that highly asymmetrical key is very improbable. So an appearance of such key may be a sign of its corruption.

In order to authenticate access to the card memory first the access key is transmitted to the card. When done, two possible actions are available to perform: code upload or the card memory access. Code to be uploaded to the card and executed can be up to 0x5F bytes long and must be terminated with a RTS instruction (0x81). In order to perform the card memory access, the memory area should be chosen first. There are 5 memory areas: Registers (location 0), area at location 0x8000 (?), RAM (location 0x20), User ROM (location 0x4000) and EEPROM (location 0xE000). System ROM can not be accessed by these messages. After the memory area is chosen, it can be accessed for read (all areas) and write or erase (EEPROM only). See Appendix C for list and structure of memory access messages.

2.1.2. Status Inquiry Messages

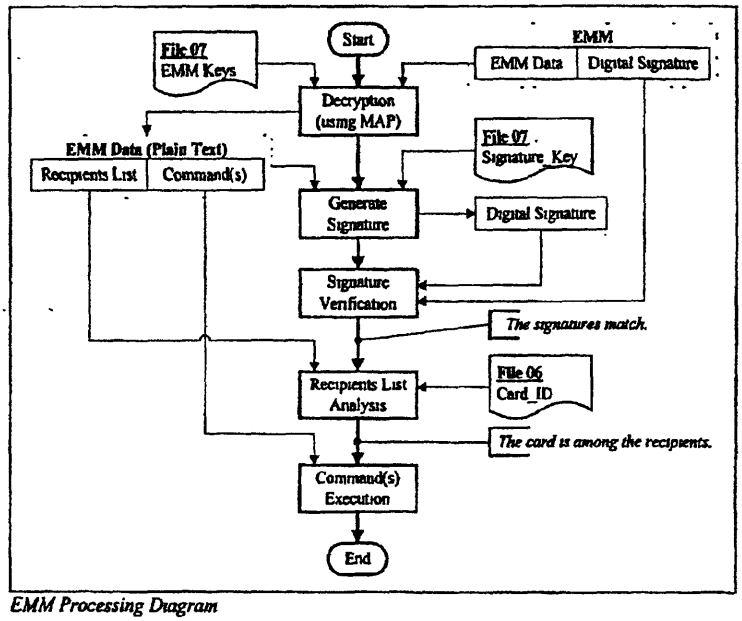
The smart card has several bytes in RAM that are bit-mapped and intended for storage of flags. These are the application states flags used for semaphoring, Boolean results of some important internal actions, files update indication and more. IRD systematically requests some of those registers to get various indications from the smart card application. There are two status inquiry messages: Process Status inquiry (Command C0) and Files Status inquiry (Command C1). See Appendix D for status inquiry messages details.

2.1.3. Files Inquiry Messages

IRD can inquire of the contents of files stored in the Heap and volume of free space in the Heap. Only public fields are available to be inquired of. There are three messages can come under this group of messages: Records Number inquiry (Command 20), Record Contents inquiry (Command 21) and Free Space inquiry (Command 40). Appendix D details these messages.

2.1.4. Secure Messages: EMM and ECM

Secure messages are encrypted and usually signed by a 64-bit long digital signature. EMM consists of an encrypted data block and a 64-bit digital signature. The cryptographic function used for EMM is based, like RSA, on heavy mathematical calculations. In order to perform these calculations the application uses the MAP, and the EMM data decryption takes about 400 milliseconds. After the decryption and the signature verification, the data block is interpreted. It contains a list of recipients (cards to which the EMM is addressed) and one or more commands which aim is usually to update sensitive data such as an entitlement, keys, PPV purchases and so on. One of commands is intended for upload code "on the fly", that is used, as observed, to upgrade the card application (patches). The recipients list may be either a sign "to all the cards", or a particular card number, or a group number with a flags pattern for each card of the group whether the EMM is addressed to it. Three most significant bytes of the card number are a group number, so each group merges 256 cards. See on the scheme below an EMM processing flow.

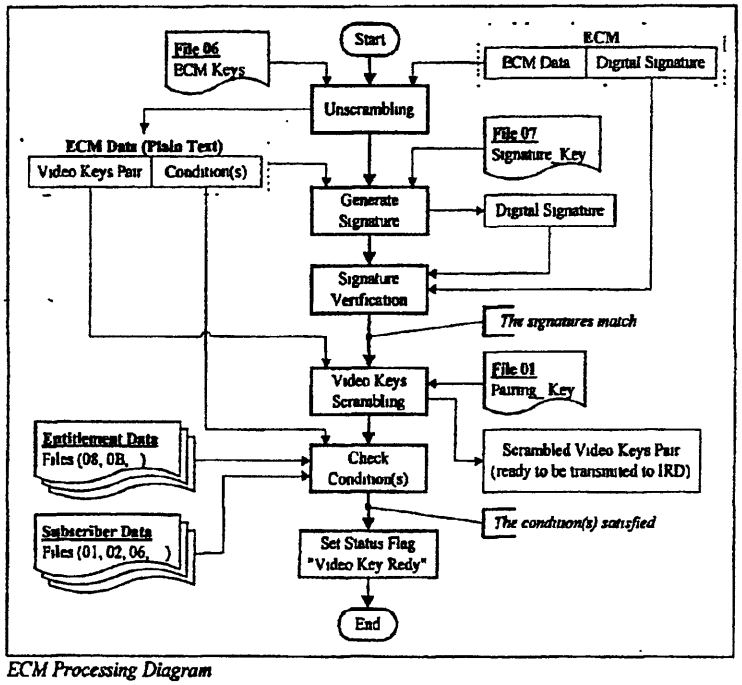


EMM Processing Diagram

EMM can be passed by command 00 or 01 or 02. Different commands are used to distinguish between different EMM data block length, signature position and the ways the EMM data block is interpreted.

ECM consists of a data block encrypted by the scrambling function, which is similar to the digital signature generating function, and a 64-bit long digital signature. This scrambling function is much simpler that the one used to decrypt EMM data block. It does not use MAP and works much faster. The reason could be that the EMM commands could perform very sensitive actions (entitlement data or keys update, code upload etc.), so it must be protected better. However ECM does not affect any of those. The ECM data block consists of a Video Keys pair and one or more entitlement conditions, which should be satisfied in order to provide these keys to IRD. After the ECM data block is unscrambled and the signature is verified, the Video Keys pair is prepared first. The pair includes, as it seems, the current and the future Video Keys. IRD sends ECM once approximately every 6-seconds period. Only one Video Key in the pair is changed each time.

ECM includes one or more conditions at least one of which has to be satisfied in order to set the status flag "Video Key is Ready". That flag is inspected by IRD when the card status is requested by Command C0. Once the flag is set. IRD requests the prepared Video Key by sending Command 13. The flag is cleared whenever the card receives a new ECM. That means IRD can request the same Video Key several times without re-sending ECM. It is used when a communication error occurred during the request. See on the scheme below an ECM processing flow.



ECM Processing Diagram

Each condition is a data block that has a byte containing condition's ID. There are 5 different conditions supported in the application (20 - 24), but only two of them (20 and 21) were observed in practice. Condition 20 is following the Video Keys in ECM that sent for the both usual services (subscribed channels) and PPV programs. It contains the channel information, the current program start date and time, and the current date and time used to update the appropriate fields in the card's EEPROM. Condition 21 presents only in ECM sent for PPV programs. It contains the PPV program ID used to verify whether the program was purchased or not. Note that in the case of a PPV program, the firstly checked Condition 20 normally is not satisfied, because the PPV channel cannot be subscribed. Thus the decision on the entitlement for a particular PPV program is made upon the fact whether Condition 21 is satisfied or not.

Other conditions are probably intended for blackout function implementation (Condition 22 and 23) and free (but scrambled) services supplying (Condition 24).

2.1.5. Other Messages

The application also supports a number of messages that are less relevant, and most of them have not been inspected in practice. The following messages can be counted as examples (were observed in practice): Command 12 (Card ID Request), Command 14 (Software ID (?) Request), Command 61 (Update IRD Software Information), Command 41 (Create PPV Info Slot) and Command 42 (Link PPV Info Slot). These command details can be found in the Appendix E.

2.2. Processing

The application implementation should work in a classic Master-Slave environment, where the IRD plays the role of Master and the card that of Slave. That means the card performs various actions upon the IRD's request, can never initiate conversation to IRD, but must always respond messages from the IRD. Such a concept is implemented by two major principles: Main Application Loop and IO Interrupt Service Routine (ISR). The Main Application Loop is invoked after the application initialization.

2.2.1. Application Initialization

When the card chip is reset, the starting point address is read from the predefined location 0xFFF0. It points to location 0x4000. Starting at 0x4000 the application firstly verifies whether the Issuer Fuse Status bit of the Security Register is set. If it is not, the card will work in Issuer mode, and an appropriate subroutine, located in the System ROM area, will be invoked. When a card works in User mode (i.e. the Issuer Fuse was blown) the card application routine is invoked. It performs the following actions:

- resets the stack pointer (SP);
- clears RAM (pads it out with zeroes);
- initializes internal variables and flags located in RAM;
- sends out ATR, reporting to IRD the application version and the EEPROM revision;
- checks for validity the EEPROM part of the Memory Access key, located at 0xE020;
- checks the Heap integrity;
- looks for an uncompleted Heap transaction and completes it if found;
- looks for Heap blocks with invalid header bytes and deletes them if found;
- merges and formats (if required) the free Heap blocks;
- finally, enters the application Main Loop.

2.2.2. Main Application Loop

In the Main Loop the application checks if there is any task pending to do. If it is, the application will perform the task and return to the loop. There are two kinds of tasks: additional processing for incoming messages and ECM/EMM data processing. Incoming messages are served by the IO ISR, which handles external interrupts caused by the falling-edge of a start bit on the IO line. These interrupts are maskable and the IO ISR is invoked only when the interrupt mask bit (I-bit) is clear. The IO ISR has a fixed location at 0x4018 in the User ROM. When a start bit of the first byte in a message causes the interrupt, the currently executed instruction (usually in the Main Loop) is completed, the current CPU state is pushed into the stack, I-bit is set to prevent reentry and the IO ISR code is invoked. The ISR collects the message according to the protocol, stores it in the input buffer, and verifies that the communication was free of errors. It can answer immediately messages not requiring any additional processing, for example, Process Status Inquiry message. When a message needs additional processing, the ISR sets an internal flag that will signal a task pending and masks external interrupts (I-bit is set) to prevent tasks overlapping. After the ISR ends, CPU resumes the Main Loop processing. No external interrupts are permitted until the pending task has been processed and answered. The Main Loop detects by the flag that a message is pending for additional processing and invokes a message dispatcher that switches to a particular subroutine according to the message type. After that subroutine finishes processing the message, it sends an answer to the IRD, the pending message flag and I-bit are cleared, and the application returns to the Main Loop.

EMM and ECM are processed in two stages. Firstly, like usual messages, they are collected and verified by the IO ISR. The ISR sets the message pending flag and the Main Loop, when resumed, invokes the dispatcher. The message processing subroutines for EMM/ECM, invoked by the dispatcher, do not decrypt and process ECM/EMM data, because this action takes too much time and could possibly cause time-out error on the IRD. They simply check the message format, scan some files to obtain the data needed to decrypt and process the message, copy data from an input buffer to a processing buffer, and finally answer to the IRD that the message has been successfully received. Note that no processing of EMM/ECM data buffer has yet been done. As opposed to other messages, ECM and EMM have an additional stage of processing. Before the application returns to the Main Loop, some internal flags are set signaling that EMM/ECM data buffer is pending, i.e., to be decrypted and processed. The Main Loop recognizes these flags and begins the required processing. Whilst the EMM/ECM data is being processed the I-bit remains clear and the card can receive messages and respond to them. Note that every message, except the Process Status Inquiry, will abort the EMM/ECM data processing. After the processing is completed, the appropriate completion flags are set and pending flags are cleared in the Process Status. The IRD requests the Process Status by Command C0 approximately once a second and therefore can recognize the processing completion.

2.2.3. Initial Data Exchange

When the card is inserted into the IRD, or when the IRD is reset, the IRD and the card exchange some information. Firstly, the card sends ATR to the IRD, where it specifies parameters for future communications and informs the IRD about its ROM version and EEPROM revision number. Next the IRD sends to the card Command C0 (Process Status Inquiry) twice, Command 12 (Card ID Request), Command 14 ((?) Software ID Request) and Command C1 (Files Status Inquiry). Upon the application initialization the Files Status is set as though all files have been updated since the last inquiry. Accordingly, the Data Synchronization mechanism (see below) causes the IRD to request all files in their entirety. Note that File 07 is never requested because it is supposedly hidden, however even if it were requested, only public fields (i.e. the first 4 bytes) could be reported. Finally, the IRD requests the volume of free space in the Heap by Command 40.

2.2.4. Data Synchronization

The IRD retains its own image of the data stored in the card. However, some commands, such as EMM, can affect that data. The IRD should synchronize its data in accordance with the changes that were made. As said above, the IRD sends Command C0 approximately once a second. When the bit of the Process Status that indicates EEPROM update is set, IRD finds out by Command C1 which files were updated. Each updated file is inquired of in its entirety, regardless of which of its records were really changed. Thus the IRD can keep up to date its own image of cards' files, although it does not decrypt EMM data block in order to check up on what EMM commands it includes.

2.3. Entitlement

The entitlement for a particular service is the most important part of the Headend Conditional Access system provided both by the IRD and by the card. Firstly, a newly purchased IRD should be authorized by the Head office in order to involve it into the network. After the IRD is authorized, the system offers a possibility to subscribe to a particular channel or a number of channels (Subscription), or to purchase a particular service for one-time watching (Pay-Per-View or PPV). Information about both of them (Subscription and PPV) is stored on the card in several files.

2.3.1. Authorization

In order to involve a new IRD into the network, the IRD owner (not yet a subscriber) should dial up the Head office and state numbers of the IRD and the card. The Head office transmits via

satellite several EMM that affect both the IRD and the card. Upon authorization, the IRD and the card are paired ("married") to each other, so the IRD will work only with the card that was authorized with it. Although the changes that the authorization makes in the card data are well known, the authorization process within the IRD has not been discovered. After the authorization is completed, the IRD owner can subscribe to channels and therefore becomes a Subscriber. The Subscriber can change his/her subscription several times, but the authorization for the IRD and the card can be made only once.

2.3.2. IRD Role

The IRD plays an active role in the Headend Conditional Access system. It denies attempts by the Subscriber to access any services he/she is not entitled to view. These unauthorized services are painted in a red background in the Electronic Program Guide (EPG), when displayed to the Subscriber. The IRD obtains the information about the Subscriber's entitlement from the image of the card data kept in the IRD. As described above this image is always synchronized with the card data. Therefore, even if we could bypass the Check Conditions block during the ECM processing and assuming a positive decision on its output, it is useless, because the IRD would deny any attempt to switch to an unauthorized service.

Another aspect of the IRD function is PPV order and a report to the Head office about those purchases (Feedback). In order to send a Feedback to the Head office, the IRD must be connected to a telephone line. The card retains in file 06 the information required to establish a telephone connection to the Head office.

2.3.3. Subscription

A Subscriber can subscribe to a particular channel or a package of channels only by dialing up the Head office, which will transmit an appropriate EMM to the IRD via satellite (and the IRD subsequently sends it to the card). Subscription information is stored in the file 08. EMM commands 20, 21, 22, 23 (and possibly others) affect records of this file. Therefore the Head office only may begin, expire, cancel or renew a subscription. The application in theory invalidates any service after the subscription expiry by verifying a service starting date and a subscription expiry date. However, in practice the subscription expiry date fields in all records are set to 31/12/2030.

2.3.4. Pay-Per-View

A Subscriber can purchase a PPV service by dialing up the Head office, which in turn sends an appropriate EMM to the IRD via satellite (and the IRD will send it to the card). Another way to purchase a PPV service is "on-the-fly", i.e. by choosing it in the EPG. In this case, the IRD generates an appropriate EMM by itself and sends it to the card. In either option, this EMM creates a new record of the file 0B and writes into it the entitlement information regarding the purchased PPV service. Next the PPV Info Slots are created (by Command 41) and linked (by Command 42) to this new record. The card retains in the file 0C a debit, a credit and a threshold for PPV purchasing. When a PPV service is purchased, its price is added to the debit. No further purchases are allowed when the credit limit is reached. As results from the Headend software analysis show, the debit can be reset only by an EMM command, sent e.g. from the Head office. Most likely, after the Head office receives a Feedback from the IRD, it broadcasts EMM(s) in order to reset the PPV purchases debit. However, such EMM(s) may be generated by the IRD as well, after it has successfully transmitted a Feedback to the Head office. Note that when the Subscriber starts watching a purchased PPV service, he receives a short free preview, while the service has not yet been marked as viewed. Probably the Head office in future will not charge the Subscriber for unviewed services, but the PPV services' price has already been added to his/her debit on the card at the time of the purchase.

2.3.5. ECM and Video Keys

Only when the Subscriber switches to an authorized service, the IRD begins to send ECMs, which are associated with this service, to the card. The frequency of ECM arrival is approximately every 6 seconds. That means that in the worst case scenario the subscriber will wait for 6 seconds before viewing the chosen service. On reception of the ECM the application prepares the Video Key pair, updates the current date and time according to the received ECM, and checks the conditions on which it provides the Video Key pair. Appropriate flags are set in the Process Status according to results of the checking. Upon recognizing the positive result of the check (by requesting the Process Status), the IRD sends Command 13 to acquire the Video Key pair.

3. Attack Tactic

The chip penetration involved two stages: firstly the memory contents (User ROM, System ROM and EEPROM) were extracted. Further, reverse engineering of the chip software was done in order to understand the logic of the system. Understanding of the software logic enables finding "holes" that may be further applied to manipulate the chip. This section will describe the initial code extraction, the discovered hole in the security logic, the utilization of this hole to access the chip's memory and the possibilities of 3M hack.

3.1. Attack on Chip's Hardware

In a normal program flow the CPU issues an address reference to the memory. The content of the memory is interpreted as an instruction, which is then loaded to the Instruction Latch. The output of the Latch is routed to the micro code memory that, apart from controlling the CPU, loads the Latch with a new instruction at the right moment. The program counter (PC) output is incremented either linearly, when the current instruction is an arithmetic or logic one, or irregularly, when the current instruction is a flow control (JMP/BRA/JSR etc.). If one inhibits the Load signal when the Instruction Latch is loaded with an arithmetic instruction, the program counter will be incremented linearly and the entire memory will be scanned thorough in a linear form. An instruction fetch mechanism is shown on the Figure 3.1 below.

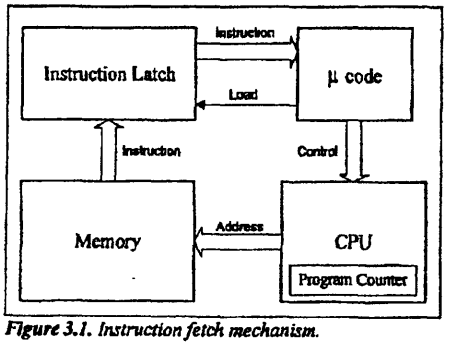


Figure 3.1. Instruction fetch mechanism.

Figure 3.2 below depicts the chip Instruction Latch and the Load input signal. Strapping this point to the Ground (GND) (with a probing needle) disables refreshing of the latch. Then the data bus is probed (one bit at a time) with a second needle clock by clock. The rising edge of the Reset signal (RST) triggers the start of the probing, so results of all 8 bits probation can be easily synchronized to obtain the data bus contents. It consists of repeated executed patterns, each consisting of a byte

fetched linearly from the memory and irrelevant data that were passed through the bus during the instruction execution.

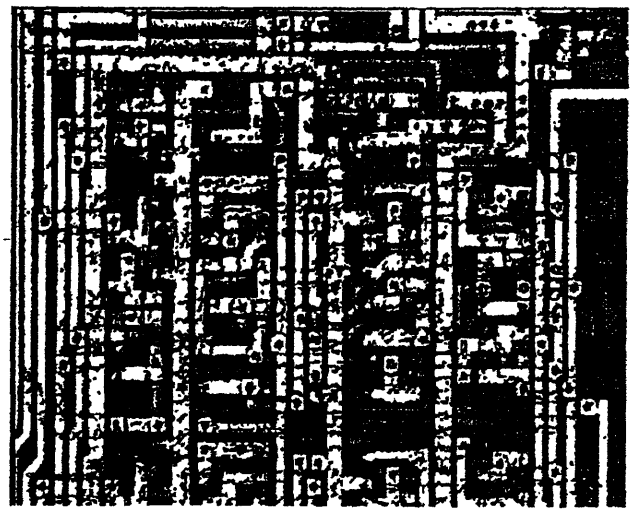


Figure 3.2. The Load input signal and the Instruction Latch.

In the chip it was found that this technique does not violate the Security Access Matrix logic, thus it does not cause the non-maskable interrupt (NMI). Therefore no further chip modification is necessary.

3.2. RAM Ghost Effect

The memory structure of the chip is standard one for the ST16 family. The first 20h locations are allocated for special purpose registers (SPR). The address decoder for the SPR uses the full address bus (the importance of this fact will become clearer later).

The RAM is 1E0h bytes starting from location 20h. The RAM address decoder uses only 5 lines of address bus: A15 to A11. The RAM itself uses address lines A8 to A0. So when decoding the RAM address the decoder ignores lines A9 and A10. Therefore, any memory reference under mask b' 00000xx????????? will access the RAM. For example, a reference to either the address 0x020 or 0x220 or 0x420 or 0x620 will access the same RAM location 0x020. This is the RAM Ghost effect. Note, however, that a reference to the address 0x200 will not access SPR location 0x00, because the address of SPR is fully decoded.

3.3. Stack Overwrite

The RAM Ghost effect can be used to upload a code to the card and cause the card to execute that code. An incoming message is collected into the input buffer starting at RAM location 0x19C. The designed length of the buffer is 0x64 bytes up to and including the location 0x1FF, i.e. the last memory location of the RAM. The application designers did not check the maximal possible length of an incoming message while it is being collected, as to determine if it exceeds the buffer. Probably they believed that such verification was superfluous (indeed, there is no physical memory allocated from the location 0x200 to 0x1FFF). However, due to the RAM Ghost effect, an incoming message

of maximal possible length, i.e.255 bytes, will affect RAM locations from 0x19C to 0x1FF and from 0x20 to 0x9A. The affected area includes the stack, so such a message can overwrite it. By composing an appropriate message an attacker can upload its written code to the card's RAM and then cause a communication error, for example, by sending a wrong CRC byte or by not sending the CRC byte at all (it cause a time-out error). In the example in Appendix F overwriting the variable Flags0 (location 0x30), so that the bit signaling the parity error is set to 1, causes the error. When such an error occurs, the application ignores the just received message, sends an error status to the IRD and ends the IO ISR by executing a RTI instruction. On that instruction the CPU pops a return address from the stack to resume the interrupted program. As said, the stack can be overwritten by an incoming message so that the return address will point to the just uploaded code. It causes the chip to execute that code when the CPU resumes the program flow on returning from the interrupt. The stack overwriting message will contain the code to be executed, correct values for some internal variables and flags being overwritten (in order to keep the application running), and the stack overwriting data that causes the chip to execute the given code. The structure of a typical message, that uses the RAM Ghost effect to upload and execute miscellaneous code, is shown below.

21 00 A8 ; The message header: NAD, PCB and LEN fields.

Offset in message	Size	Description	Value	Location in RAM
0x00	100	User's code to be uploaded.	?	19Ch
0x64	32	Irrelevant data (skip to adjust the location).	Zeroes	200h (not mapped)
0x84	6	Irrelevant application internal variables.	Zeroes	20h (220h)
0x8A	1	Opcode of RTS instruction.	0x81	26h (226h)
0x8B	3	Irrelevant application internal variables.	Zeroes	27h (227h)
0x8E	1	Opcode of RTS instruction.	0x81	2Ah (22Ah)
0x8F	5	Irrelevant application internal variables.	Zeroes	2Bh (22Bh)
0x94	1	The Flags0 variable. Bit0 must be set to 1 specifying the Inverse convention for IO. Bit2 = 1 signals the parity error to invoke an error treatment.	0x05	30h (230h)
0x95	1	The Flags1 variable. Bits 0 and 2 must be set to 1 specifying the message reception stage.	0x05	31h (231h)
0x96	2	Irrelevant data (skip to adjust the location).	Zeroes	32h (232h)
0x98	1	The NAD field of the current message.	0x21	34h (234h)
0x99	1	The PCB field of the current message.	0x00	35h (235h)
0x9A	1	The LEN field of the current message.	0xA8	36h (236h)
0x9B	6	Irrelevant data (skip to adjust the location).	Zeroes	37h (237h)
0xA1	1	The RcvCntr variable. It counts the number of message bytes received so far. The application increments it after overwriting.	0xA1	3Dh (23Dh)
0xA2	1	Irrelevant data (skip to adjust the location).	0x00	3Eh (23Eh)
0xA3	1	The Indx variable. It retains an index in the input buffer where to store a next received byte. Like RcvCntr it is post-incremented. By overwriting this byte to 0xDF, the index to store the next message byte is moved directly to 0x7C (Top_Of_Stack – 4): 0x7C=(0x19C+(0xDF+1))%0x200.	0xDF	3Fh (23Fh)
0xA4	2	The first possible location of the return address in the stack. It is overwritten to the starting address of the uploaded code.	0x01 0x9C	7Ch (27Ch)
0xA6	2	The second possible location of the return address in the stack. It is overwritten to the starting address of the uploaded code.	0x01 0x9C	7Eh (27Eh)

CRC ; Any value.

An example of such a message can be found in Appendix F. Note that instead of the code used in this example to download the card's EEPROM contents, any code can be designed and written into.

3.4. ROM Functions Utilization

The chip software analysis discovered several useful functions that can be utilized in the uploaded code in order to efficient and optimize it. Some of these functions have one or more parameters passed explicitly in the code. The parameters passing mechanism in the function supposes them to follow the JSR instruction that involves the function. The parameter bytes are then skipped (PC is appropriately incremented on return from the function). In the function call convention (see Appendix G) these parameters appear as Prm1–4. Some other functions receive their parameters through CPU registers (A and X) and the Common Pointer variable (W24) – 2 bytes at the RAM location 0x24. Note that none of these functions affect A or X registers, unless it is notified.

3.4.1. Random Write/Erase

Random write and erase functions allow writing or erasing any EEPROM location excluding OTP area that can not be erased. Before any of these functions can be invoked, a value 0x96 must be assigned to the byte at RAM location 0x4E called E2ProgKey. The invoked write/erase function verifies this byte as one of security conditions before the operation (possibly to avoid glitches). Then it shifts the value of this byte one bit right, so that on the function return it becomes 0x4B. When several these functions are called consequently, the E2ProgKey must be set to 0x96 before the first function is called and simply shifted left just before to call a next one. See Appendix G for details of these functions usage. Note that an error occurrence during writing/erasing the EEPROM counts as fatal and causes the reset to the card (jump to 0x4000).

3.4.2. File Operations

Several functions allow easily operating with files in the Heap. These are Create, Write, Close, Read, Find, and Delete functions. Files are implicitly pointed by the W24 variable that points to the current record after Create, Read, or Find function is called. Consequent calls to Read or Find functions will read/find the next file record from the currently pointed by W24 one. If W24 does not point to a valid Heap location, it will get the pointer to the start of the Heap (as stored at the location 0xE030). All these functions clear the Carry bit to signal a success completion. See Appendix G for call convention of these functions.

3.4.3. Other Functions

Apart of the described above functions, there are several functions that do not affect EEPROM, but they are also useful to efficient uploaded code. Such are the ISO_Write function, memory-manipulating functions (see from location 0x6838) and some cryptographic function (see from location 0x7DBE), etc.

3.5. 3M Hack Possibilities

3M hack or a "blue card" composing seems to be implemented easier than others due to the active IRD's role in the system. The IRD has to be authorized by the Head office, which in turn requests the IRD owner to identify himself. On the other hand, when the IRD is disconnected from the telephone line, the Head office has no feedback from it. Thus the Head office can not inspect the data stored on the card to verify it with the actual entitlement information. The card can be removed from the IRD and reinserted again freely, although it is "not recommended" by the broadcaster. There are several possibilities to modify the data on the card in order to expand the actual entitlement.

3.5.1. Reset PPV Debit

This method of 3M hack is the simplest. The IRD is simply disconnected from the telephone line. Any desired PPV services can be still purchased until a credit limit is reached. When it is, the debit field in the file 0C must be zeroed and all files 0B and 11, created at the time when PPV services were purchased, must be deleted.

3.5.2. Cloned and Universal Subscription

This is a classic 3M hack. The Subscriber, subscribing to a basic package of services for a minimal possible charge, can view any services (excluding PPV) even if he/she is not authorized to view them. The principle of this method is that an attacker writes the appropriate entitlement information regarding the subscription on the card. Because that information is stored on the card as plain text and, moreover, is identical for cards with the same entitlement, such information can be taken from file 08 of the card having the maximal possible subscription and written to another card having the minimal. Another possibility to expand the actual subscription is to modify an existing file 08, so that it will universally satisfy ECM conditions for any service except PPV. This can not be applicable for PPV services, because the IRD requires an appropriate file 0B to exist in order to switch to the chosen PPV service. If it does not exists, the IRD will offer to purchase the PPV service on attempt to choose it in the EPG.

3.5.3. Cloned and Universal PPV Entitlement

As said above, the IRD allows the Subscriber to switch to a PPV service only when an appropriate file 0B, containing correct information regarding this service, exists on the card. The difficulty in "faking" file 0B for a particular PPV service is that the ID number is different for each service (even if they have the same content and are broadcast on the same channel), and such ID can not be easily discovered. Certainly, as in the case of the subscription cloning (see above), the appropriate file 0B may be copied from another card, where it was properly created by the IRD. Another way to allow the Subscriber to switch to an unordered PPV service is to create a universal file 0B that will satisfy ECM conditions for any PPV service and never expire.

4. 3M Hack in Practice

In order to try to implement 3M hack in practice, an authorized IRD and its "married" card of ROM version 003 were taken. The IRD (and the card) were intended for Dish Network USA. Once the card EEPROM contents were downloaded, the card was not used anymore. Its EEPROM image was burnt up to another card, and this card was utilized in all 3M hack attempts. The IRD, of course, was disconnected from the telephone line.

4.1. Dish Network USA

The card initially had a minimal subscription and an option to purchase PPV services. Using the RAM Ghost effect, the EEPROM contents of the card were downloaded. Further using knowledge of the 128-bit memory access key, the card's EEPROM could be easily modified by the memory access messages. Apart of such an approach, there is another way to perform each one of described above 3M hack possibilities, i.e. by composing an appropriate stack-overwriting message.

4.1.1. Reset PPV Debit

The given card had some PPV purchased and viewed, but not charged yet. The IRD displayed them in the PPV purchasing history window. After a "manual" deletion of all files 0B and 11 and

clearing the PPV debit field in file 0C, the IRD displayed an empty window. The same action could be done automatically by sending a stack-overwriting message shown in Appendix H.

4.1.2. Universal Subscription

Files 08 on the card were modified so that they would satisfy entitlement conditions for any service excluding PPV. That was done "manually" by setting the field Services_List to {00, 01, 7F, FF}, the field Service_Active – to {FF, FF} and, finally, the field Parental_Rate(?) – to {FF, FF, FF, FF}. After that the IRD displayed every service in the EPG with blue background and every service was available for watching.

4.1.3. Universal PPV Entitlement

In order to allow watching of every PPV service without purchasing it first, an existing record of the file 0B for the PPV service that had been already marked as viewed was modified. The modifications caused the IRD to recognize this record as suitable for every PPV service and the IRD did not offer to purchase the service on attempting to switch to it. Firstly, a currently screened PPV service was purchased. Next the following fields of the newly created record of the file 0B were "manually" set to:

- Record_Flags (offset 0x03): {30} (valid, viewed).
- Valid_By (offset 0x0D): {4C, 21} (01/01/2030)
- Channel_List (offset 0x10): {00, 01, 7F, FF} (all possible channels ID range)
- Service_Active (offset 0x14): {FF, FF, FF, FF}
- Services_List (offset 0x16): {00, 00, 01, 7F, FF, FF, FF} (all possible services ID)

Appendix A. General Data Storage Area Structure.

Address	Type	ID	Description
0xE000	Array [32]	OTP	One Time Programmable (OTP). Used only by System ROM.
0xE020	Array [16]	E2AccessKey	128-bit number used as the second part of the key provided to access smart card memory.
0xE030	Word	Heap_Ptr	Pointer (High-Low) to start of the Heap.
0xE032	String [6]	RevNum_Str	ASCII representation of the EEPROM software revision number ("Rev###"). It is output with other historical characters in ATR.
0xE038	Long Integer	Card_ID	4 bytes (High-Low) giving the ID number as printed on the card, without the check digit. They are sent to IRD as an answer to the message 0x12.
0xE03C	Date Packed[1]	Current_Date	Updated only when a Video Key preparation requested.
0xE03E	Array [14]	RecLen_Arr	Holds record length for E2 file types with fixed record length (01 – 0C). The two last bytes are zeroes indicating that files 10 & 11 have variable record length.
0xE04C	Array [4]	Software_ID	(?) IRD requests these 4 bytes by Command 14 during an initial data exchange. These bytes are identical for all cards with the same ROM application version. Current value is 0F 4C 54 60.
0xE050	Byte	RevNumber	Patches revision number. This byte plays an important role in update patches mechanism. The downloaded routine checks this bytes first and modifies it accordingly after the update completed.
0xE051	Byte	Max_TblEntries	Defines the number of entries in the patch table. When zero, the patch invoking routine returns immediately. So the patch invoke mechanism may be bypassed by setting this byte to zero.
0xE052	String [24]	NipperIs_Str	(?) String ASCII "NipPEr Is a buTt liCkeR!"
0xE06A	Array [8]	Unknown_8	(?) Used in message 0x99 processing with NipperIs_Str.

[1] Date packed into two bit mapped bytes: bits 0-4: Day, bits 5-8: Month, bits 9-15: Years since 1992.
[2] Time packed into two bit mapped bytes: bits 0-4: Seconds / 2, bits 5-10: Minutes, bits 11-15: Hours.

Appendix B. File Structures.

File 0x01. Record Length = 0x27.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte == 0x01.
0x01	Byte	Network_ID	(?) Network ID: 00 for USA, 08 for Canada.
0x02	Array [2]	Unknown_02	(?) Correct value: 01 00 for both USA and Canada.
0x03	Byte	Record_Flags	Bit 7 == 1 means that the record is logically deleted (it keeps existing in the Heap, but the application can't use it). Correct value is 01.
0x05	Long Integer	Zip_Code	For USA is the user's ZIP code. For Canada – unknown.
0x09	Byte	Time_Zone	Difference of the current time zone (hours * 4) from the World Time.
0x0A	Byte	Unknown_0A	(?) Not used. Correct value: 00.
0x0B	Array [4]	Unknown_0B	(?) Different for each authorized card.
0x0F	String [16]	IRD_Info	An ASCII string representing IRD software and system information written during authorization.
0x1F	Array [8]	Pairing_Key	64-bit key used to encrypt the Video Key before it is provided to IRD.

File 0x02. Record Length = 5.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte == 0x02.
0x01	Byte	Network_ID	(?) Broadcaster ID: 01 for USA, 09 for Canada.
0x02	Array [2]	Unknown_03	(?) Correct value for both USA & Canada: 01 00 00.
0x03	Byte	Record_Flags	Bit 7 == 1 means that the record is logically deleted (it keeps existing in the Heap, but the application can't use it). Correct value is 01.

File 0x06. Record Length = 0x38.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte = 0x06.
0x01	Byte	Network_ID	(?). 00 or 01 for USA, 08 or 09 for Canada.
0x02	Byte	Unknown_02	(?) Correct value is 00.
0x03	Byte	Record_Flags	Bit 7 = 1 means that the record is logically deleted (it keeps existing in the Heap, but the application can't use it). Correct value is 01.
0x04	Long Integer	Card_ID	4 bytes (High-Low) giving the card ID number without the check digit
0x08	Array [10]	FB_PhoneNum	Value: 18 00 45 42 50 4F FF FF FF FF for USA authorized card. Probably it is a phone number (?) of the Head-office to send Feedback (1-800-4542504).
0x12	Byte	FeedBack_Set	(?) This byte specifies whether date & time for the Feedback is set. Value is 04 if they are set or FC if not.
0x13	Date Packed, Time Packed.	FeedBack_Day	(?) Date and time when IRD should call the Head-office to report about purchases (PPV, Adult TV and more). If it is not set, the value will be FF FF 00 00.
0x17	Array [5]	Unknown_17	(?)
0x1C	Array[12]	BlackOut_Map	(?) Bit mapped. Verified in some criteria (if passed to) when Video Key is requested.
0x28 🔑	Array[8]	ECM_Key1	This 64-bit key is used to decrypt the passed ECM, when Video Key is requested (its first part).
0x30 🔑	Array[8]	ECM_Key2	This 64-bit key is used to decrypt the passed ECM, when Video Key is requested (its second part).

File 0x07. Record Length = 0x79.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte = 0x07.
0x01	Byte	Network_ID	(?). 00 or 01 for USA, 08 or 09 for Canada.
0x02	Byte	Unknown_02	(?) Correct value is 3F.
0x03	Byte	KeySet_Num	(?) The keys set number. Correct value is 00 or 01.
0x04	Array[15]	Group_Key	(?) This array of bytes is used in RSA (?) cryptographic routine, when the passed encrypted message is addressed to the particular group of cards only.
0x13	Array[15]	Private_Key	(?) This array of bytes is used in RSA (?) cryptographic routine, when the passed encrypted message is addressed to this particular card only.
0x22	Array [15]	Common_Key	This array of bytes is used in RSA (?) cryptographic routine, when the passed encrypted message is addressed to all cards.
0x31	Array [8]	Signature_Key	A 64-bit number used to generate and check digital signature in messages 00, 01, 02, 03 & 30.
0x39	Array [64]	RSA_Key	Used as 512-bit number in RSA (?) cryptographic routine (also as two 256-bit numbers).

File 0x08. Record Length = 0x1C.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte == 0x08.
0x01	Byte	Network_ID	(?). 01 for USA, 09 for Canada.
0x02	Byte	Unknown_02	(?) Correct value is 01 for both USA and Canada.
0x03	Byte	Record_Flags	Bit 7 == 1 means that the record is logically deleted (it keeps existing in the Heap, but the application can't use it). Correct value is 00.
0x04	Array [3]	Package_ID	(?) Non-relevant data, perhaps it is a service package ID.
0x07	Array [6]	Unknown_07	(?) Not used.
0x0D	Date Packed	Starting_Date	The date when the entitlement is started.
0x0F	Date Packed	Expiry_Date1	The date when the entitlement will expire. The value is usually 4C 21 (January 1, 2030).
0x11	Date Packed	Expiry_Date2	The same as above. Both are checked (what for ?).
0x13	Word Range[1]	Services_List	When a Video Key is requested, the passed ECM contains a number of the service scrambled by that key. The service number is checked to be in the range as one of conditions to provide the Video Key.
0x17	Word	Service_Active	Two bit-mapped (?) bytes. When a Video Key is requested, the passed ECM contains two bytes next to the service number. One of conditions to provide the Video Key is that logical AND of these two bytes with the Service_Active field doesn't give zero. The value is usually 80 00.
0x19	Array [4]	Parental_Rate	(?) The purpose of this field is not clear. Its value is usually FF 00 FF 00. It seems that this field is used to specify a rating needed to watch at a particular service.

[1] Range is a special type used to define a list. One possibility of a list definition is by the low and the high boundaries. In this case the list definition includes two parts with the same length, up to 4 bytes each one. The first part is the low boundary; the second is the high one. The most significant bit (MSB) of the high bound must be zero. Every value between the boundaries belongs to the list. Another possibility of a list definition is when the MSB of the high boundary is set to 1. In this case, the first part is the list base and the second part is bit-mapped include/exclude pattern (it can also be longer then the base). Its MSB is always set to 1 meaning that the list consists of at least one value – the base itself. If the (MSB-1) bit is set to 1, the list will include also the base value +1 and so on.

File 0x0B. Record Length = 0x23.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte = 0x0B.
0x01	Byte	Network_ID	(?). 01 for USA, 09 for Canada.
0x02	Byte	Unknown_02	(?) Correct value is 01 for both USA and Canada.
0x03	Byte	Record_Flags	Bit 7 = 1 means that the record is logically deleted (it keeps existing in the Heap, but the application can't use it). Bit 5 is set when the record is created or the PPV purchasing. Bit 4 is set when the PPV program is getting being watched (probably means that this PPV purchasing can not be deleted).
0x04	Array [3]	Service_ID	PPV service ID.
0x07	Byte	PPV_ProgInfo	The PPV program information record number. This slot contains the program name and its start date and time.
0x08	Amount[1]	PPV_Price	Price of the PPV program.
0x0B	Date Packed	Valid_Since	The purchased PPV program is available to watch since this date.
0x0D	Date Packed	Valid_By	The purchased PPV program is available to watch up to this date.
0x0F	Byte	PPV_ChnInfo	The PPV channel information record number. This slot contains the PPV channel name.
0x10	Word Range	Channel_List	The list of channels that can be watched due to this PPV purchasing.
0x14	Word	Service_Active	A field with similar purposes as the same named field in the file 08.
0x16	3-byte Integer Range	Services_List	The first 3 bytes are PPV service number is in the field Service_ID. The next 4 bytes are used as the include/ exclude list in the range checking.
0x1D	Word	Preview_Count	Counter of free preview before the PPV program is marked as watched and is charged.
0x1F	Packed Date. Packed Time	Start_Watching	These date and time are written when the PPV program was started to watch.

[1] Amount is a 3-byte long numeric type. The first word is a number of dollars. The third byte is a number of cents in BCD format.

File 0x0C. Record Length = 0x12.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte == 0x0C.
0x01	Byte	Network_ID	(?). 01 for USA, 09 for Canada.
0x02	Byte	Unknown_02	(?) Correct value is 01 for both USA and Canada.
0x03	Byte	Record_Flags	Bit 7 == 1 means that the record is logically deleted (it keeps existing in the Heap, but the application can't use it). Correct value is 00.
0x04	Array [3]	Unknown_04	(?) Usage isn't clear. Correct value is 01 86 9F for USA.
0x07	Date Packed	Unknown_07	(?) Usage isn't clear.
0x09	Amount	PPV_Debit	Total debit of PPV purchasing.
0x0C	Byte	Unknown_0C	(?) Not used.
0x0D	Amount	PPV_Credit	(?) Maximal allowed credit for PPV purchasing.
0x10	Word	PPV_Threshold	(?) Threshold for debit of PPV purchasing (in dollars).

File 0x11. Record length is variable. Such records store a PPV program information.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte == 0x11.
0x01	Byte	InfoSlot_Num	This byte is used to retrieve records of file 11.
0x03	Byte	Data_Length	The record data length: the length of PPV program name + 4 (for the program's start date & time).
0x04	Date Packed, Time Packed	Start_DateTime	PPV program's start date & time.
0x07	String []	Program_Name	ASCII string of (Data_Length - 4) bytes long. It contains the name of PPV program.

File 0x11. Record length is variable. Such records store a PPV channel name.

Offset	Type	ID	Description
0x00	Byte	File_Type	File type byte == 0x11.
0x01	Byte	InfoSlot_Num	This byte is used to retrieve records of file 11.
0x03	Byte	Data_Length	The record data length: the length of PPV channel name.
0x04	String []	PPV_ChnName	ASCII string of Data_Length bytes long. It contains the name of PPV channel.

Appendix C. Memory Access Messages.

Command A0 20 – Access Authentication.

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
25 ; LEN = 5 bytes of Command header + 10h bytes ROM key + 10h bytes EEPROM key.
A0 20 00 00 ; Command Header
20 ; Command Data Length: 10h bytes of RAM key part + 10h bytes of EEPROM key part.
8F AB C2 64 44 9A FE 70 1D E7 62 FA B1 4C 31 06 ; ROM key part (version 3).
... ; 10h bytes of the EEPROM key part (card depended).
CRC ; XOR of all the message bytes.

Command A0 B0 – Random Memory Read.

Choose memory area to access:

21 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
05 ; LEN = 5 bytes.
A0 B0 ; Command Header.
?? ; Memory area ID: 90 for Registers (starting at location 0), 91 for area starting at 8000 (?), 92 for RAM (starting at 20), 93 for User ROM (starting at 4000) and 94 for EEPROM (starting at location E000).
00 : Low byte of the block offset.
01 : Number of bytes to read.
CRC ; XOR of all the message bytes.

Read memory block (the block length is specified by 2 bytes):

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
07 ; LEN = 7 bytes.
A0 B0 ; Command Header.
?? ; High byte of the block offset relatively to the starting location of the memory area.
?? ; Low byte of the block offset.
00 : Not used.
?? ?? : 2 bytes specifying the number of bytes to read (the block length).
CRC ; XOR of all the message bytes.

Read memory block (the block length is specified by 1 byte).

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
05 ; LEN = 5 bytes.
A0 B0 ; Command Header.
?? ; High byte of the block offset relatively to the starting location of the memory area.
?? ; Low byte of the block offset.
?? ; 1 byte specifying the number of bytes to read (the block length).
CRC ; XOR of all the message bytes.

Command A0 D6 – Write EEPROM or User Code Upload.

Write memory block (EEPROM must be chosen for access first):

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
?? ; LEN = 5 bytes + length of the block to be written.
A0 D6 ; Command Header.
?? ; High byte of the block offset relatively to the starting location of the EEPROM (E000).
?? ; Low byte of the block offset.
?? ; 1 byte specifying the number of bytes to write (the block length).
... ; The data bytes to write (the block).
CRC ; XOR of all the message bytes.

Upload & Execute user defined code.

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
?? : LEN = 5 bytes + length of the user code to upload & execute.
A0 D6 98 00 ; Command Header.
00 ; Not used.
... : The user code to upload & execute, terminated by a RTS instruction (81). After execution of the code the card will answer this message with SW1=X and SW2=Acc.
CRC ; XOR of all the message bytes.

Appendix D. Inquiry Messages.

Command C0 – Process Status Inquiry.

Message:

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
08 ; LEN = 8 bytes
A0 CA 00 00 02 ; Command Header
C0 ; Command ID.
00 ; (?) Not used (must be zero).
06 ; (?) Expected answer length.
CRC : XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
08 ; LEN = 8 bytes
B0 04 ; Answer Header
?? ; Security Register value (location 0001) is usually 08 (no hardware violation occurred).
?? ; Byte from location 0049. It is bit-mapped. There are meanings of particular bits.
- Bit 0 = 1 means that EEPROM update (either write or erase) was performed since the last files status inquiry.
- Bit 1 = 1 means that software reset was made since the last common status inquiry message (this flag is cleared after the first status inquiry message).
- Bit 2 = 1 means that an error occurred during an EEPROM cell erase / write.
- Bits 3 – 7 are not used (zeroes).

?? ; Byte from location 004A. It is bit-mapped. There are meanings of particular bits.
- Bit 0 = 1 (?) means that Video Key can not be prepared from the passed ECM due to invalid data (?) that was preset by EMM Command 02.
- Bit 1 = 1 means that an EMM Command (either 00 or 01) was received and pending to be decrypted and executed.
- Bit 2 = 1 means that the last EMM Command was decrypted and executed completely.
- Bit 3 = 1 means that IRD requested the smart card to collect and encrypt information to be sent as a feed back to the Head office and the request is still pending.
- Bit 4 = 1 means that the information for the feedback was collected and encrypted or this request can not be completed due to absence of the required key (?).
- Bit 5 is not relevant. (?) Bit 5 = 1 means that EMM Command 64, to encrypt 64 bytes with IRD pairing key (64-bit number stored in file 01 at offset 0x1F), was successfully completed and the IRD can send Command 60 to take the result.
- Bits 6 and 7 are not used.

?? ; Byte from location 004B. It is bit-mapped. There are meanings of particular bits.
- Bit 0 = 1 means that an ECM Command was received, thus a Video Key request is pending.
- Bit 1 = 1 means that the Video Key was successfully prepared from the received ECM and subscriber's entitlement meets all ECM conditions for the given service.
- Bit 2 is set together with the Bit 1 when Video Key preparation is completed, but in opposition to the Bit 1 it is never checked.
- Bit 3 = 1 means that the Video Key preparation was canceled due to the required files absence, invalid ECM data, bad digital signature, insufficient entitlement for the given service and more. Note that if there is all the data needed for the preparation, the Video Key will be prepared before any examinations of entitlement are made.

Bit 4 is not relevant. Bit 4 = 1 means that ECM Condition 24 (?) is true and the Video Key can be provided. It is set together with Bits 1 and 2. That condition has never observed in ECM Commands and its purpose is not clear.
Bit 5 = 1 means that EMM Command 02 was received and pending to be decrypted and executed.
Bit 6 = 1 means that EMM Command 02 was decrypted and executed completely.
Bit 7 is not used (zeroed).

90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Command C1 – Files Status Inquiry.

Message:

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
08 ; LEN = 8 bytes
A0 CA 00 00 02 ; Command Header
C1 ; Command ID.
00 ; (?) Not used (must be zero).
04 ; (?) Expected answer length.
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
06 ; LEN = 6 bytes
B1 02 ; Answer Header.
?? ; Byte from location 004C. After the inquiry it is cleared. The byte is bit-mapped. There are meanings of particular bits.
Bit 0 = 1 means that file 09 was updated.
Bit 1 = 1 means that file 0A was updated.
Bit 2 = 1 means that file 0B was updated.
Bit 3 = 1 means that file 0C was updated.
Bits 4 and 5 are not used.
Bit 6 = 1 means that file 11 was updated.
Bit 7 = 1 means that file 10 was updated.
?? ; Byte from location 004D. After the inquiry it is cleared. The byte is bit-mapped. There are meanings of particular bits.
Bit 0 = 1 means that file 01 was updated.
Bit 1 = 1 means that file 02 was updated.
Bit 2 = 1 means that file 03 was updated.
Bit 3 = 1 means that file 04 was updated.
Bit 4 = 1 means that file 05 was updated.
Bit 5 = 1 means that file 06 was updated.
Bit 6 = 1 means that file 07 was updated.
Bit 7 = 1 means that file 08 was updated.
90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Command 40 – Heap Free Space Inquiry.

Message.

21 : NAD (Node Address)
00 : or 40 - PCB (Protocol Control Byte) according to sequence
08 ; LEN = 8 bytes
A0 CA 00 00 02 ; Command Header
40 ; Command ID.
00 ; (?) Not used (must be zero).
04 ; (?) Expected answer length.
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
06 : LEN = 6 bytes
70 02 : Answer Header.
?? ?? ; The total number of bytes in all free (unused) blocks in the Heap.
90 00 ; SW1, SW2 – Protocol Status word.
CRC : XOR of all the message bytes.

Command 20 – Number of Records Inquiry.

Message:

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
0C ; LEN = 12 bytes
A0 CA 00 00 06 ; Command Header
20 ; Command ID.
04 ; (?) Not used.
?? ; Type number of the inquired file (01 – 0C, 10, 11).
02 FF FF ; (?) Not used.
03 ; (?) Expected answer length.
CRC : XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
05 : LEN = 5 bytes
A0 01 : Answer Header.
?? : The number of records in the inquired file.
90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Command 21 – Record Contents Inquiry.

Message.

21 ; NAD (Node Address)
00 : or 40 - PCB (Protocol Control Byte) according to sequence
0D : LEN = 13 bytes

A0 CA 00 00 07 ; Command Header
21 ; Command ID.
05 : (?) Not used.
?? : Type number of the inquired file (01 – 0C, 10, 11).
03 FF FF ; (?) Not used.
?? ; Required record number (starting at 0).
?? : Expected answer length. Usually it is the record length (not including the private fields if any) – 1 (the file type byte is skipped) + 2 (for SW1, SW2).
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
?? ; LEN = as the expected answer length + 2 (for the answer header).
A0 ?? ; Answer Header. ?? is the record length – 1 (for the byte of the file type that is skipped).
... ; The required record contents starting at offset 0x01 in the record. Note, if the specified in the message expected length of the answer is greater than the public part of the record, only public fields will be reported, and the rest of the answer will padded with zeroes.
90 00 ; SW1, SW2 – Protocol Status word.
CRC : XOR of all the message bytes.

Appendix E. Other Messages.

Command 12 – Card ID Request.

Message:

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
08 ; LEN = 8 bytes
A0 CA 00 00 02 ; Command Header
12 ; Command ID.
00 ; (?) Not used (must be zero).
06 ; (?) Expected answer length.
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
08 ; LEN = 8 bytes
92 04 ; Answer Header.
?? ?? ?? ?? ; 4-byte Card ID number.
90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Command 14 – Software ID (?) Request.

Message.

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
08 ; LEN = 8 bytes
A0 CA 00 00 02 ; Command Header
14 ; Command ID.
00 ; (?) Not used (must be zero).
06 ; (?) Expected answer length.
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
08 ; LEN = 8 bytes
94 04 ; Answer Header.
?? ?? ?? ?? ; 4 bytes from the location E04C (probably a software or system ID).
90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Command 41 – Create PPV Info Slot.

Message:

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
?? ; LEN = 8 + PPV Info data length.

A0 CA 00 00 ?? ; Command Header. ?? is PPV Info data length + 2.
41 ; Command ID.
?? ; PPV Info data length.
... ; PPV Info data. It can have two different structures according to the PPV Info Slot purpose: either the PPV program information or the PPV channel information storage. PPV program information consists of 4 bytes holding the program starting date and time, and the program name in ASCII that is variable length. PPV channel information includes only the PPV channel name in ASCII that usually is 5 bytes long.
03 ; (?) Expected answer length.
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
05 ; LEN = 5 bytes
71 01 ; Answer Header.
?? ; ID number of the created PPV Info Slot.
90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Command 42 – Link PPV Info Slot.

Message:

21 ; NAD (Node Address)
00 ; or 40 - PCB (Protocol Control Byte) according to sequence
0F ; LEN = 15 bytes.
A0 CA 00 00 09 ; Command Header. ?? is PPV Info data length + 2.
42 : Command ID.
07 ; Linkage information length = 7 bytes. Those are the following fields up to and excluding the expected answer length field.
?? ?? ?? ?? ?? ; These 5 bytes are used to retrieve File 0B to whom the slot will be linked. First 2 bytes must be identical to 2 bytes at offset 01 in the file (Network ID). Next 3 bytes are PPV service ID. They have to match the appropriate field in the file at offset 04.
?? ; PPV Info Slot ID number as answered to the Command 41 (Create PPV Info Slot).
?? ; The position in the File 0B where the given Slot is linked to. It must be either 00 or 01 according to the data the given Slot contains: 00 is for PPV program information and 01 is for PPV channel information.
03 ; (?) Expected answer length.
CRC ; XOR of all the message bytes.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.
00 ; or 40 - PCB is the same as in the message.
05 : LEN = 5 bytes
72 01 ; Answer Header.
?? ; ID number of the linked PPV Info Slot (the same as the given one).
90 00 ; SW1, SW2 – Protocol Status word.
CRC ; XOR of all the message bytes.

Appendix F. Stack Overwrite Example.

EEPROM Contents Download.

Message:

```
21 ; NAD (Node Address).
00 ; or 40 - PCB (Protocol Control Byte) according to sequence.
A8 ; LEN = 0xA8 bytes.

; EEPROM (E000 - EFFF) download routine.
; Code      Address   Mnemonic   Notes
9D          ;019C     NOP
9D          ;019D     NOP
9D          ;019E     NOP
9D          ;019F     NOP
C6 E0 00    ;01A0     LDA   0xE000   ; Load the current location contents to Acc.
CD 01 C8    ;01A3     JSR   0x01C8   ; Send out the value of Acc.
A6 FF       ;01A6     LDA   #0xFF    ; Waste some time between bytes.
B7 21       ;01A8     STA   0x21
42          ;01AA     MUL
3A 21       ;01AB     DEC   0x21
26 FB       ;01AD     BNE   0x01AA
AE FF       ;01AF     LDX   #0xFF    ; Load 0xFF to Xreg to efficient addressing.
6C A3       ;01B1     INC   0xA3, X1 ; Increment the Low Byte of the current location.
26 0A       ;01B3     BNE   0x01BF   ; Loop if the High byte shouldn't be incremented.
6C A2       ;01B5     INC   0xA2, X1 ; Increment the High Byte of the current location.
E6 A2       ;01B7     LDA   0xA2, X1 ; Load the High Byte.
A1 F0       ;01B9     CMP   #0xF0    ; Check the EEPROM boundary.
26 02       ;01BB     BNE   0x01BF   ; Loop if it has not been reached yet
20 FE       ;01BD     BRA   0x01BD   ; When it has, stop execution.
CC 01 A0    ;01BF     JMP   0x01A0   ; Loop for the next byte.
9D 9D 9D 9D 9D 9D ; Location 0x1C2 – skip 6 bytes (6 NOP instructions) to adjust location.
; Byte writing routine (ETU = 32 / f, where f is an external frequency applied to the card).
; Code      Address   Mnemonic   Notes
11 00       ;01C8     BCLR0 0x00     ; Set IO low for the Start Bit.
AE 08       ;01CA     LDX   #8
BF 20       ;01CC     STX   0x21     ; Set bits counter for 8 Data bits.
AE 01       ;01CE     LDX   #1       ; Set initial Parity Bit value.
9D          ;01D0     NOP
9D          ;01D1     NOP
9D          ;01D2     NOP
9D          ;01D3     NOP            ; Waste some time to hold IO in the given state.
9D          ;01D4     NOP
9D          ;01D5     NOP
20 00       ;01D6     BRA   0x01D8
48          ;01D8     LSL   A
25 05       ;01D9     BCS   0x01E0   ; Branch if the current bit is zero.
10 00       ;01DB     BSET0 0x00     ; Set IO high for the current bit is one.
5C          ;01DD     INC   X        ; Toggle the Parity Bit.
20 04       ;01DE     BRA   0x01E4   ; Go to check loop conditions.
11 00       ;01E0     BCLR0 0x00     ; Set IO low for the current bit.
```

```
30 21      ;01E2   NEG   0x21      ; Compensate time.
3A 20      ;01E4   DEC   0x20      ; Decrement bits counter.
26 EB      ;01E6   BNE   0x01D3    ; Loop for the next bit.
AD 11      ;01E8   BSR   0x01FB    ; Waste some time.
9D         ;01EA   NOP
57         ;01EB   ASR   X         ; Obtain the Parity Bit.
39 00      ;01EC   ROL   0x00      ; Set IO according to the Parity Bit.
42         ;01EE   MUL
42         ;01EF   MUL
30 21      ;01F0   NEG   0x21      ; Waste time of the Parity Bit durance.
9D         ;01F2   NOP
10 00      ;01F3   BSET0 0x00      ; Set IO high for Stop Bits.
42         ;01F5   MUL
42         ;01F6   MUL
42         ;01F7   MUL
42         ;01F8   MUL
42         ;01F9   MUL
42         ;01FA   MUL             ; Waste time of the Stop Bits durance.
81         ;01FB   RTS
```

00 00 00 00 ; Location 0x1FC – skip 4 bytes to adjust location.

; Location 0x00 – Skip 0x20 bytes to adjust location (contents is immaterial).
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

; Location 0x20 – Note that some of these bytes must be overwritten with 0x81 (RTS opcode).
00 00 00 00 00 00 81 00 00 00 81 00 00 00 00 00

05 ; Location 0x30 – this is Flags0. Bit0 = 1 specifies the Inverse convention for communication. Bit2 = 1 signals the parity error to exit IO ISR without message processing.

05 ; Location 0x31 – this is Flags1. Bits 0 and 2 set specify the stage of the message collection.

00 00 ; Location 0x32 – Skip 2 bytes.

21 00 A8 ; Location 0x34 – these 3 bytes contain NAD, PCB and LEN of the current message.

00 00 00 00 00 00; Location 0x37 – Skip 6 bytes to adjust location.

A1 ; Location 0x3D – this byte contains a number of bytes received so far. It is incremented after the currently received message byte is stored in the input buffer. Its value must be returned back correctly after overwriting to let the application complete reception of the message.

00 : Location 0x3E – Skip 1 byte.

DF ; Location 0x3F – this byte retains the index in the input buffer where to store a next received byte. It is incremented after the currently received message byte is stored in the appropriate position of the input buffer. By overwriting its value to DF we set the actual location to store next message bytes to 0x7C (Top-Of-Stack – 4):
(0x19C + (0xDF + 1)) mod 0x200.

01 9C ; Location 0x7C – overwrite the currently stored in the stack return address to 0x01A0.

01 9C ; Location 0x7E – overwrite the currently stored in the stack return address to 0x01A0.

CRC ; Any value.

Answer:

12 ; NAD is the same as in the message, nibbles are swapped.

81 ; The error code of invalid CRC or incorrect parity.

00 ; LEN

93 ; CRC.

... ; 4096 bytes of EEPROM contents from the location 0xE000 to 0xEFFF.

Appendix G. Useful ROM Functions.

E2Prog_1byte – Random write a byte to EEPROM.

Address: 0x7801.

Input: X – an address of the source byte in the RAM.
W24 – an address in the EEPROM where to write the source byte.

Output: A = 1. X and W24 are unchanged.

E2Prog – Random write a vector to EEPROM.

Address: 0x7803.

Input: X – an address of the source vector in the RAM.
A – a number of bytes in the source vector to write.
W24 – an address in the EEPROM where to write the source byte.

Output: None. A, X and W24 are unchanged.

E2Erase_1byte – Random erase a byte in EEPROM.

Address: 0x7986.

Input: W24 – an address of the byte in the EEPROM to erase.

Output: A = 1. X and W24 are unchanged.

E2Prog – Random erase a vector in EEPROM.

Address: 0x7988.

Input: W24 – an address of the vector in the EEPROM to erase.
A – a number of bytes in the vector to erase.

Output: None. A, X and W24 are unchanged.

Create_E2File – Create a new file record in the Heap.

Address: 0x4740.

Input: Prm1 – a file number, a new record of which is to be created.
A – a length of the record to be created. This input is relevant only if Prm1 is 0x10 or 0x11. For other files the record length is fixed and taken from the table at location 0xE03E.

Output: CF = 1 – there is not enough free memory to create a new record of the given length.
CF = 0 – the record has been successfully created; W24 points to the new record. Note that the new record must be closed further by the Close function to make it a valid file record.

Write_E2File – Write data into a file record.

Address: 0x47C0.

Input: W24 – pointer to the record.
Prm1 – an address of the source buffer.
Prm2 – an offset in the record, where the data from the source buffer is written. Note that the file number (the first byte of the record) can not be overwritten by this function. If this parameter is zero, the first byte in the source buffer will be skipped.
Prm3 – a number of bytes to be written.

Output: None.

Close_E2File – Validate (close) a just created record.

When a new record has been just created, the block that contains it is still marked as invalid (BCB=3). This function usually called when data is completely written into the new record in order to validate the block (BCB=7).

Address: 0x4871.

Input: W24 – pointer to the record.

Output: None.

Read_E2File – Find (next) and read a record of the given file.

Address: 0x45FD.

Input: W24 – pointer to the current record (if it points outside the Heap, the function sets it to point to the start of the Heap).
Prm1 – the given file number.
Prm2 – an address of the target buffer to read the record into (irrelevant if Prm3 = 0).
Prm3 – a number of bytes to read.

Output: CF = 1 – no more records of the given file found in the Heap (next the current record if any).
CF = 0 – the record has been found and read into the buffer; W24 points to this record.

Find_E2File – Find (under mask) and read a record of the given file.

Address: 0x45FD.

Input: W24 – pointer to the current record (if it points outside the Heap, the function sets it to point to the start of the Heap).
V8_F8 – up to 8 bytes at location 0xF8 – a search mask for the record to find (see Prm4).
Prm1 – the given file number.
Prm2 – an address of the target buffer to read the record into (irrelevant if Prm3 = 0).
Prm3 – a number of bytes to read.
Prm4 – a search pattern. When a particular bit of it is set to 1, a particular byte of the record is checked to match a particular byte of the mask. MSB is responsible for checking the second byte of the record against the byte at location 0xF8, the Bit6 – for the third byte against the byte at 0xF9 and so forth.

Output: CF = 1 – no more records of the given file found under the given mask in the Heap (next the current record if any).
CF = 0 – the record has been found and read into the buffer; W24 points to this record.

Delete_E2File – Delete the currently pointed file record.

Address: 0x45BF.

Input: W24 – pointer to the current record.

Output: None. Note that W24 is not changed. Therefore Read_E2File or Find_E2File function can be called after the current record was deleted to switch to a next record.

ISO_Write – Send a byte to the host.

Address: 0x42D7.

Input: A – a byte to send.

Output: None. Note that the interrupt mask bit is set on return. A and X are affected.

Appendix H. 3M Hack in Practice.

Reset PPV Debit.

Message.

21 ; NAD (Node Address).
00 ; or 40 - PCB (Protocol Control Byte) according to sequence.
A8 ; LEN = 0xA8 bytes.

; EEPROM (E000 - EFFF) download routine.

; Code	Address	Mnemonic		Notes
CD 4B 8B	;019C	jsr	0x4B8B	; Set W24 to point to the start of the Heap.
9D	;019F	nop		
A6 4B	;01A0	lda	#0x4B	; Set E2ProgKey = Stand By.
B7 4E	;01A2	sta	0x4E	
9D	;01A4	nop		
9D	;01A5	nop		
CD 45 FD	;01A6	jsr	0x45FD	; Find the next record of the file 0B.
0B	;01A9	.db	0x0B	; File number: 0B
80	;01AA	.db	0x80	; Target buffer: irrelevant:
00	;01AB	.db	0	; Number of bytes (simply find the record).
25 05	;01AC	bcs	.+7	; No more records of the file 0B.
CD 45 BF	;01AE	jsr	0x45BF	; Delete the current record.
20 F3	;01B1	bra	.-11	; Loop for the file 0B
3F 24	;01B3	clr	0x24	; Reset the W24 to point to the start of the Heap.
CD 45 FD	;01B5	jsr	0x45FD	; Find the next record of the file 11.
0B	;01B8	.db	0x11	; File number: 11
80	;01B9	.db	0x80	; Target buffer: irrelevant.
00	;01BA	.db	0	; Number of bytes (simply find the record).
25 05	;01BB	bcs	.+7	; No more records of the file 11.
CD 45 BF	;01BD	jsr	0x45BF	; Delete the current record.
20 F3	;01C0	bra	.-11	; Loop for the file 11.
3F 24	;01C2	clr	0x24	; Reset W24 to point to the start of the Heap.
CD 45 FD	;01C4	jsr	0x45FD	; Find the next record of the file 0C.
0C	;01C7	.db	0x0C	; File number: 0C
80	;01C8	.db	0x80	; Target buffer: irrelevant.
00	;01C9	.db	0	; Number of bytes (simply find the record).
25 0D	;01CA	bcs	.+15	; No more records of the file 0C.
CD 6E 0F	;01CC	jsr	0x6E0F	; Clear (zero) a vector.
80	;01CF	.db	0x80	; Vector address: 0x80
04	;01D0	.db	4	; Clear 4 bytes
CD 47 C0	;01D1	jsr	0x47C0	; Write to the current record.
80	;01D4	.db	0x80	; From buffer at location 0x80.
09	;01D5	.db	9	; To the record at offset 0x09.
03	;01D6	.db	3	; Write 3 bytes.
20 EB	;01D7	bra	.-19	; Loop for the file 0C.
9D	;01D9	nop		
CC 40 00	;01DA	jmp	0x4000	; Reset the card.
9D	;01DD	nop		
9D	;01DE	nop		
9D	;01DF	nop		

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 01E0
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; 01F0
```

; Location 0x00 – Skip 0x20 bytes to adjust location (contents is immaterial).

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

; Location 0x20 – Note that some of these bytes must be overwritten with 0x81 (RTS opcode).

```
00 00 00 00 00 00 81 00 00 00 81 00 00 00 00 00
```

`05` ; Location 0x30 – this is Flags0. Bit0 = 1 specifies the Inverse convention for communication. Bit2 = 1 signals the parity error to exit IO ISR without message processing.

`05` ; Location 0x31 – this is Flags1. Bits 0 and 2 set specify the stage of the message collection.

`00 00` ; Location 0x32 – Skip 2 bytes.

`21 00 A8` ; Location 0x34 – these 3 bytes contain NAD, PCB and LEN of the current message.

`00 00 00 00 00 00` ; Location 0x37 – Skip 6 bytes to adjust location.

`A1` ; Location 0x3D – this byte contains a number of bytes received so far. It is incremented after the currently received message byte is stored in the input buffer. Its value must be returned back correctly after overwriting to let the application complete reception of the message.

`00` ; Location 0x3E – Skip 1 byte.

`DF` ; Location 0x3F – this byte retains the index in the input buffer where to store a next received byte. It is incremented after the currently received message byte is stored in the appropriate position of the input buffer. By overwriting its value to `DF` we set the actual location to store next message bytes to 0x7C (Top-Of-Stack – 4):

`(0x19C + (0xDF + 1)) mod 0x200.`

`01 9C` ; Location 0x7C – overwrite the currently stored in the stack return address to 0x01A0.

`01 9C` ; Location 0x7E – overwrite the currently stored in the stack return address to 0x01A0.

`CRC` ; Any value.