EM MICROELECTRONIC - MARIN SA

AN431

SWATCH GROUP ELECTRONIC SYSTEMS

Application Note 431

Title:	**EMDB410 RFID Reader Firmware Description**
Product Family:	**RFID**
Part Number:	EMDB410 – EM4294 RFID Reader
Keywords:	ISO15693 - ISO14443 Type A,B – EM4294
Date:	March 29, 2007

www.emmicroelectronic.com

1. Introduction

EMDB410 Reader is a base station for communication with a selected set of 13.56MHz transponders. The reader supports the EM Microelectronic-Marin ISO15693 transponder ICs and some ISO14443 ICs. Application note AN431 describes programmer notes concerning the EMDB410 Reader firmware code and should be treated as extending information to the "EMDB410 RFID Reader – Description of Firmware protocol" document which chapters are referred to.

The source files can be ported to another microcontroller family that provides sufficient performance, three hardware counters, UART interface and several independent I/O pins.

EMDB410 Reader firmware source files are written in C programming language and targeted for ATMega16 microcontroller family. The firmware communicates with the respective software application whose source files are also available.

EMDB410 Reader was derived from the EMDB410 Reader. EMDB410 firmware is close to the EMDB408 firmware in terms of the firmware code functions and source architecture. Most of the EMDB408 commands work in the same way as the EMDB410 commands.

EMDB410 versus EMDB408 difference

EMDB410 Reader differs from EMDB408 Reader in the following features;

- EMDB410 is USB powered only while EMDB408 requires the power supply adapter.

- EMDB410 is based on EM4294 multi crypto HF reader chip while the EMDB408 uses separate EM4094 IC (RF front-end) and external SIM card crypto.

- EMDB410 uses an ATMega16 microcontroller while EMDB408 uses ATMega64 microcontroller. Because of the smaller code and SRAM memory size provided by the ATMega16 microcontroller, the EMDB410 firmware is split into two different firmware binary streams compiled from the unified source files (one for ISO15693+EM4006 communication, one for ISO14443 communication), EMDB408 firmware is a single unified binary stream only. ATMega16 firmware memory usage had to be optimized too.

2. Abbreviations

ACK – ACKnowledge status byte
AFE – Analogue Front End
ASK – Amplitude Shift Keying
ATR – Answer To Reset
BPSK – Binary Phase Shift Keying
CRC – Cyclic Redundancy Check
CTC – Clear Timer on Compare
DES – Data Encryption Standard
EAS – Emergency Alert System
EGT – Extra Guard Time
EOF – End Of Frame
ETU – Elementary Time Unit
FSK – Frequency Shift Keying
PPS – Protocol and Parameter Selection
SOF – Start Of Frame
UART – Universal Asynchronous Receiver/Transmitter
USB – Universal Serial Bus
EMDB408 – EM4094 RFID Reader
EMDB410 – EM4294 RFID Reader

3. Environment setup

Following tools were used:
1. Compilation - the whole code can be compiled via make/gcc port for ATMega16 chip family. See Readme.txt included in source package file for actual compiler release used.
2. Programming[1] - the ATMega16 chip is equipped with standard serial programming interface. It can be programmed via Xilinx Parallel Cable III+ (and compatible) and several programming utilities. For example, see http://www.elm-chan.org for avrsp utility, or uisp utility (included in the WinAVR package). When appropriate firmware is already present in the ATMega16 chip, an update of application part firmware can be performed by the bootloader part of the firmware via USB port until the boot_ld.c file content is modified.

---

1 ATMega16 chip configuration (fuse bits) needs to be set (e.g., see fuse.bat in the source package) before first programming. FTDI FT232BL USB<->Serial Bus converter IC device descriptor needs to be configured (MProg utility) to allow the requested SUB power consumption (typically 250mA), before the first switching the RF field ON.

www.emmicroelectronic.com

4. Firmware Description

Firmware architecture is split into following levels, each containing specific functions.

1. Level1 - defines decoding routines

2. Level2 - defines low level data send and data extraction routines

3. Level3 - defines high level data transformations and main loop body, bootloader, and simcard functions

4. Level4 - defines UART communication routines

4.1. Firmware philosophy

Main loop (level3) periodically invokes an analysis of the UART receive buffer (level4) and performs particular actions on valid messages. All performed actions or detected errors result to at least one response message. UART data reception is performed asynchronously. No next message analysis is invoked until the complete response on previous action is sent out.

Actions of regular commands communicating with the tag are controlled by means of hardware counters (counters T0, T1, T2) that are incremented by microcontroller clock signal. Some routines are triggered by interrupt, routines requiring higher performance are coded a polling way. Send (level2) and capture (level1) actions are expected to run mutually exclusive as same as other heavy load operations.

Uplink (send) routines usually expect the command bytes to be prepared by the application software. All the routines usually prepare the appropriate bit stream into the **data_buffer** array.

To separate high performance capture routines from off-line data extraction, the capture actions transform each captured information item to the pair [data bit, validity bit]. Each pair emitted by capture routine is stored by the level1/store_bit function into the **capture** array indexed by **capture_cnt** and **capture_bit_count** variables. **Capture** array is initialised before each capture routine execution; the data bit part of the array is zeroed, the valid bit part is set to '1', i.e. all bits are invalid. The received data is then searched off-line.

When even higher performance is required, the capture actions just store the stream of raw captured data into the **raw_data** array. Such raw data is then processed later to produce the [data bit, validity bit] arrays.

Such philosophy gives a quite serial and deterministic behaviour without need of asynchronous process communication or priority re-entrant interrupt handlers.

4.2. Source files

- Makefile
- Batches
  - o gcc.bat – invokes the compilation of both firmware binary streams
  - o fuse.bat – initialises the microcontroller fuses

  - o progX.bat – uploads the firmware into the microcontroller (prog5.bat for ISO15693 firmware family, prog4.bat for ISO14443 firmware family)

- Level 1
  - o level1.h – declares common decoding variables and functions
  - o level1_14443.h – declares common variables and functions related to ISO14443
  - o level1_14443.c – defines data capture functions related to ISO14443
  - o level1_15693.h – declares common variables and functions related to ISO15693
  - o level1_15693.c – defines data capture functions related to ISO15693
  - o main.c – contains main entry point and microcontroller resources initialisation

- Level 2
  - o level2.h – declares common uplink variables, functions and microcontroller port initialisation
  - o level2_14443.h – declares common variables and functions related to ISO14443 uplink
  - o level2_14443.c – defines ISO14443 related uplink routines and response data extraction
  - o level2_15693.h – declares common variables and functions related to ISO15693 uplink
  - o level2_15693.c – defines ISO15693 related uplink routines and response data extraction
  - o level2.c – defines EM4294 SPI routine, CRC functions and EM4006 data extraction routine

- Level 3
  - o level3.h – declares common main loop variables and routine
  - o level3.c – defines main loop and main execution functions
  - o simcard.h – declares common variables and routines related to SIM card
  - o simcard.c – defines SIM card related routines
  - o boot_ld.c – implements a bootloader feature

- Level 4
    - level4.h – declares common variables and routines for UART handler
    - level4.c – defines UART handler code
- Custom level
    - custom_level.h – declares custom level variables and routines
    - custom_level.c – defines custom level code

Two separate firmware streams can be created by the compilation via <make> command in the shell window.

The corresponding value of RF_SUPPORT environment variable must be set before the compilation;

- Use <set RF_SUPPORT=V15693> command for compiling the ISO15693 and EM4006 family 161 firmware. The result is main_V15693.hex file, generated if no error occurs only. ISO14443 related code is excluded from the compilation.
- Use <set RF_SUPPORT=V14443> command for compiling the ISO14443 family 162 firmware. The result is main_V14443.hex, generated if no error occurs only. ISO15693 and EM4006 related code is excluded from the compilation.

Remember to check the compilation output for errors. Run <prog.bat> to program main.hex firmware into the microcontroller device.

4.5. Custom level

Custom level has been added to make the customer firmware modification easy.

EMDB410 firmware expects the custom level defines/declares the following constructs;

- INITIALISE_CUSTOM_PORTS macro that controls the customer I/O
- customer_init() function that is called once after the basic microcontroller initialisation is done
- customer_main() function that is called every time before the check for incoming UART message is called
- customer_acquire() function that is called before the standard firmware command is executed so that it can be delayed, for example due to custom critical timings
- customer_release() function that is called after the standard firmware command is executed
- exec_customer_command() function that is called with custom data when EFh command has been received

Customer_acquire() and customer_release() functions are implemented so that the custom level can intercept the standard firmware command execution that requires the maximum of performance. Thus custom level can either delay the standard command execution or disable all the conflicting custom resources (e.g.; customer implemented interrupts pending during command transmission/response capture).

4.3. Microcontroller Start-up

After power-up, the microcontroller enters in the bootloader section (see "EMDB410 RFID Reader – Description of Firmware protocol" document for bootloader description). The bootloader is bypassed if entered because of watchdog. Then, main.c/main() initialises the microcontroller resources including the microcontroller port settings and directions and passes the control to level3/main_receiver().

4.4. Level 3

Level 3 defines **main_receiver loop** body (see Figure 1 Main_receiver loop) and main execution routines.

Main loop periodically calls the level4/CheckIncomingMessage() routine to check incoming UART data. If any valid message is parsed well, the main loop executes appropriate action block otherwise it invokes an error response generation. In general, all the executive routines are coded there, see "EMDB410 RFID Reader – Description of Firmware protocol" document for each execution command description. Each action should generate at least one response in bounded time.
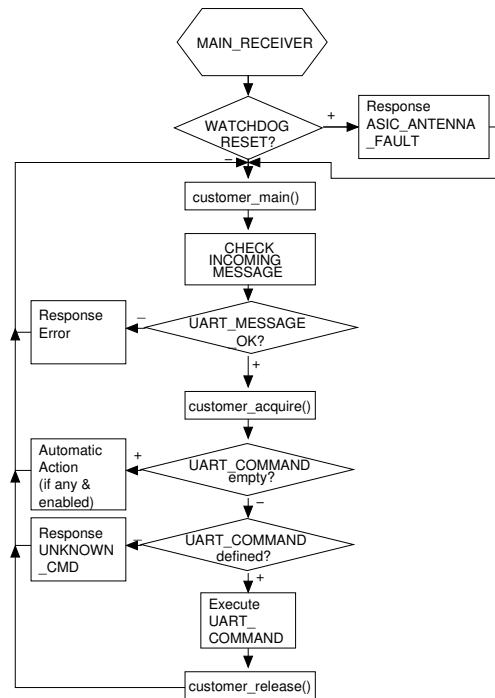

Figure 1 Main_receiver loop

4.6. Level 4

Level 4 defines UART communication routines.

4.6.1. PC -> Reader communication

Incoming bytes are stored in a circular buffer. Function CheckIncommingMessage (see Figure 2 Check Incoming Message) analyses the content of incoming circular buffer which contains incoming bytes.

This function loop implements a final state machine with following states: UART_EMPTY - no bytes are pending, UART_READ_SIZE - analysing incoming message size from pending bytes, UART_READ_BYTES - analysing body and ETX of message from pending bytes, UART_WAIT_ERROR_SENT - error state, UART_VALID - valid message format is found.
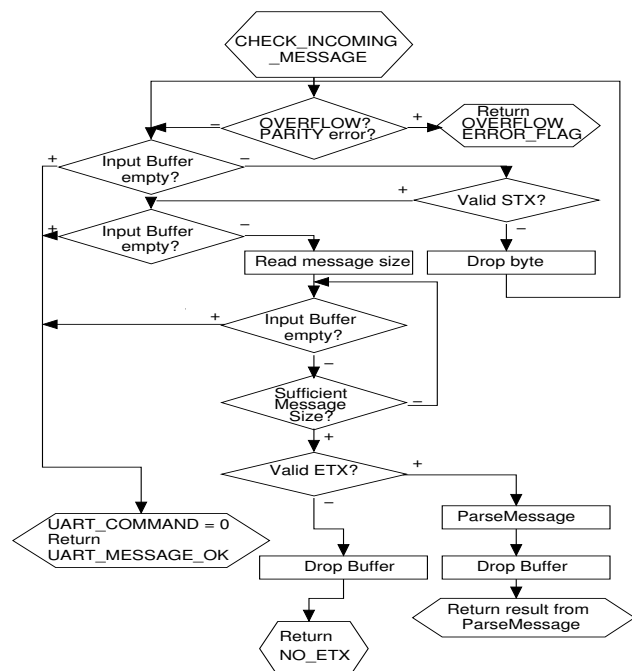

Figure 2 Check Incoming Message

Wrong start message symbol (byte STX = 02h) causes immediate error response. Zero bytes received prior the valid STX byte are ignored. Thus, if no response occurs by defined time-out, by means of sending zero bytes to the reader the internal buffer can be forced to overflow to detect any possible firmware lock-up.

Valid message is parsed in function level4/ParseMessage(), refer to

Figure 3 Parse Message (level4.c). Valid message length is checked and message useful information is copied into appropriate variables. Specific data is copied into the dedicated variables, uplink command data is stored into the **cmd_message** array (**cmd_message_len** variable contains the number of valid bytes in this array).
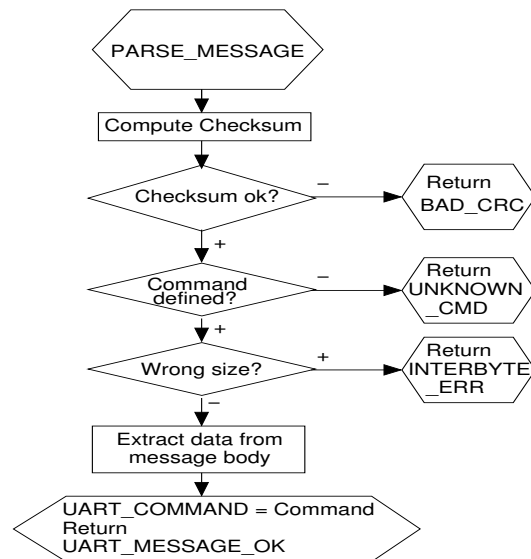

Figure 3 Parse Message (level4.c)

4.6.2. Reader -> PC communication

Response can be sent using one of the response forming routines (FormatResponseShort, FormatResponseData, SendCaptureData, SendDebugData, SendRawData, and FormatResponse_2Blocks) according to the transported data contents. All these routines are blocking, i.e. they do not return until the whole response is transmitted.

FormatResponse_Short routine that generates short response is used when the error occurs during the command parsing or the execution of the last command. It contains no data part.

FormatResponse_Data is used to send non-zero data streams successfully received by the reader.

SendCaptureData sends out current **capture** buffer comprising the captured bits and valid bits array. The size of each array is defined by **capture_cnt** variable.

SendRawData sends out current **raw_data** capture buffer of length defined in **raw_cnt** variable.

SendDebugData sends out current content of **debug_buffer** of length defined in **debug_cnt** variable.

FormatResponse_2Blocks sends out two arbitrary arrays.

5.   ISO15693 communication routines

5.1.   Uplink routine (reader to tag)

ISO15693 uplink final state machine is implemented in level2_15693/SIG _OUTPUT_COMPARE1A interrupt routine.

ISO15693 1 out of 4 forward link encoding signal can be generated by toggling the DIN input signal, on the EM4294, in intervals that are multiple of 9.44us long. Calculating all the possible interval durations, there are 15 possible values found only. See level2_15693.c file for the **fwd_time** array actual values. The values of this array are measured by the microcontroller clock minus the interrupt latency. The values were obtained empirically.

Before the transmission, the command contents are transformed into the array so that it contains the sequence of indices into the predefined uplink timings array **fwd_time** according to the 1 out of 4 forward link encoding. This array is computed by Prepare_timings_15693 routine in level2_15693.c.

Actual data transmission interrupt routine implements a simple final state machine for reprogramming the timer T1. Timer T1 is set to CTC mode where the interrupt is generated by counter T1 compare event. The new value of DIN pin is toggled in bounded time.

Timings of single modulated pulse is fixed at **fwd_time[1]** item (i.e. 9.44us minus the interrupt latency).

5.2.   Capture routines

In current versions, both ASK and FSK decoding routines are supported. Decode routines are empirical algorithms that trade off among performance, reliability, and robustness.

5.2.1.   ASK decode routine

The ASK decoding routine implements a small final state machine that decodes the response Manchester data, see level1_15693/SIG_INPUT_CAPTURE1 ASK decoding routine. The routine emits one pair [decoded bit, decode valid bit for each bit (not necessarily on each call) decoded. If a decoded bit flow is considered to be broken (the routine encounters/detects bad DOUT signal properties), only one pair having demodulation invalid bit set to 1 can be emitted. Thus, wrong or noisy sequence can be reduced to 1 pair only since no useful information is received.
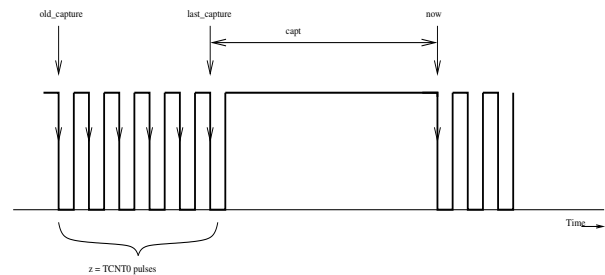

Figure 4 ASK Capture

During the reception, T1 counter counts the number of microcontroller clocks between two falling edges on DOUT output (EM4294 data demodulation output). Each edge detection causes timer counter value capture event that invokes T1 capture interrupt.

Counter T0 counts the number of DOUT pulses. Manchester encoded bit is resolved only if the **capt** value is higher than half bit data rate interval, i.e. in **now** time (see Figure 4 ASK Capture).

Number of pulses **x** and **capt** delay determine the Manchester data envelope.

Before starting the capture routine execution, the **sof** value is zeroed. If the ASK Start of Frame (SOF) is captured, the **sof** value is set to 1, if the ASK End of Frame (EOF) is captured, the **sof** value is set to 2.

Note: Keep the code latency at minimum as much as possible (T1 capture interrupt plus T2 overflow interrupt routines latency has to be strictly less than *halfDataRate* variable value) otherwise the decoding routine fails.

5.2.2.   FSK capture routine

FSK decode routine is located in level1_15693/dual_subcarrier_polling function. Analysis of FSK DOUT signal consumes the full performance of the microcontroller because of the signal bit frequency. Since one FSK pulse length is 28 to 32 RF clocks, there is no time to involve interrupt style capture routine. Therefore, FSK decode routine is a polling type routine.

The FSK response signal consists of a sequence of pairs comprising 9 and 8 (or 8 and 9) pulses of the two different frequencies. Having set the counter T2 to count the rising edge of the subcarrier DOUT signal, the polling routine can measure the time duration of fixed sequence pairs 9, 8; 9, 8, etc. pulses assuming the measurement error of inverted pair (8 and 9 pulses) can be neglected. Each pair, i.e. time duration of adjacent 9 and 8 pulses, is evaluated as one decoded bit as the Manchester envelope is determined by the comparison of the measured duration time between 9 pulses and 8 pulses.

Unfortunately, there is no exact way how to detect the collisions by means of microcontroller resources hence current decode routine is not able to emit invalid bits.

FSK capture routine does not allow any processing of **SOF** (Start of Frame) status.

www.emmicroelectronic.com

5.3. Data Extraction

The result of capture routines is processed by level2/ExtractData_15693 function to extract the response data from the **capture** array.

The function utilises SearchValidBit_15693 function that searches for a first valid data bit. In ASK case, the first valid '0' bit is searched and its next bit position is returned. In FSK case, the response SOF comprises 9,9,9,8,8,8,9,8 pulses sequence of the constant pattern with some invalid bits at two possible positions. The position next to the pattern is returned.

If the valid position is obtained from SearchValidBit_15693 function, ExtractData_15693 function shifts consequent valid bits into the **data_capture** array until the invalid bit is found or end is reached.

Response length check, CRC check, and other response proprieties check are left on the upper level routines or directly on the application software.

6. Watchdog (handling EAS)

Interrupt driven capture routines have one disadvantage – interrupt priority. For example, in case some tag with EAS feature enabled is in the RF field, the EAS signal invokes the capture interrupt continuously. And, if the capture timeout interrupt has lower priority than the capture interrupt, the capture phase cannot be stopped.

Therefore, the watchdog is involved to stop the capture phase in case of EAS. Watchdog reset is handled in different way unlike the standard reset. The watchdog generates the watchdog response (0x01 - ERR_ASIC_ANTENNA_FAULT, see EMDB410 Firmware Protocol Description application note) after ~2.1s.

Application software has to set its communication timeout to wait at least for the watchdog response. Receiving the watchdog response, the application software should restore the EMDB410 Reader state completely. Anyway, EM4294 configuration word firmware internal value is not reset by the watchdog and remains corresponding to the value actually configured to EM4294 AFE.

7. EM4006 data reception

7.1. Capture Routine

EM4006 capture routine is located in level1_15693/EM4006_polling function.

EM4006 reception routine is decoding a Miller encoded response signal from DOUT EM4294 output. Such data is decoded by a small final state algorithm using DOUT pulses measured with internal timers. The decoded pair [data bit, validity] is stored into the **capture** array. EM4006 datarate that specifies the threshold period is defined by the **4006_scale** parameter by formula DataRate = $2 \char94 4006\_scale$.

7.2. Data Extraction

EM4006 data extraction routine is located in level2/Extract_EM4006 function.

EM4006 data extraction routine searches for a sequence of 80 valid bits (including the CRC check) according to the EM4006 telegram within the **capture** array. Since the EM4006 telegram is asynchronous to the capture phase, the telegram can be located at arbitrary position. If any invalid bit is found, the search is restarted starting from next valid bit until the end of the array is reached.

8. ISO14443 communication routines

8.1. Uplink routines (reader to tag)

8.1.1. ISO14443 Type A Uplink routine

Type A Uplink code is located in level2_14443/SIG_OVERFLOW1 routine together with the Type B Uplink code.

Type A Uplink code is interrupt driven final state machine that generates Modified Miller encoded pulses on DIN signal. Each interrupt is raised so that the DIN pulse falling edge can be generated, the DIN pulse in 'low' value duration is hard coded by a loop and then the DIN is set to 'high' value waiting for the next interrupt.

Duration between two interrupts is defined in **fwd_A_timings** array.

8.1.2. ISO14443 Type B Uplink routine

Type B Uplink code is located in level2_14443/SIG_OVERFLOW1 routine together with the Type A Uplink code.

Type B Uplink code is interrupt driven shifter of the data bits to DIN signal because of NRZ encoding.

Duration between two interrupts is defined by **fwd_B_timings[0]** value, see level3 initialisation code. SOF and EOF symbols are generated in level2_14443/SendForward_14443 routine according to the **fwd_B_timings[1]** (SOF) and **fwd_B_timings[2]** (EOF) data items.

8.2. Capture Routines

8.2.1. ISO14443 Type A Capture routine

Type A capture routine is located in level1_14443/type_A_polling function.

Manchester 106kbps encoded data requires high performance capture routine. Therefore, the capture phase is split into two parts; raw data capture phase and off-line decoding phase.

The raw data capture is polling type loop that measures the total duration of the group of subcarrier manchester modulated pulses and the duration of no modulation of DOUT signal (same principle as in ISO15693 ASK capture routine). In the worst case, there is no time to perform the data decoding, thus this phase only stores

the measured values into the **raw_data** array, nothing else.

The off-line decoding phase searches the Manchester envelope in the **raw_data** array, and decodes it into the **capture** array.

8.2.2. ISO14443 Type B Capture routine

Type B capture routine is located in level1_14443/type_B_polling function.

BPSK decoded data is already received on DOUT/DOUT1 signals from EM4294 AFE. Type B capture routine just samples the DOUT signal on each DOUT1 rising edge and stores them into the **capture** buffer. The validity bits are always set to '0'.

8.3. Data Extraction

8.3.1. ISO14443 Type A Data extraction

Type A data extraction routine is located in level2_14443/ExtractTypeAData function.

At first, Type A Data Extraction function calls SearchValidBit_14443 function to search for a first valid bit within the first 7 bits of **capture** array. If the valid bit position is found, the SOF bit is checked.

Then, the response data bytes are shifted out into the **data_buffer** array starting from the next position. Every data byte parity is compared to every 9th bit. The data extraction loop is terminated when any invalid bit is encountered or the parity check fails or the end of the buffer is reached.

Response length check, CRC check, and other response proprieties check are left on the upper level routines or directly on the application software.

8.3.2. ISO14443 Type B Data extraction

Type B Data Extraction routine is located in level2_14443/ExtractTypeBData function (short version used by 65h, 6Ch, and 6Dh command) and level2_14443/ExtractTypeBDataL functions (long version used by 63h command).

Response length check, CRC check, and other response proprieties check are left on the upper level routines or directly on the application software.

Short version

At first, Type B Data Extraction function calls SearchValidBit_14443 function to search for a first valid bit within the first 7 bits of **capture** array. If the valid bit position is found, the SOF sequence is checked.

Then, the response data bytes are shifted out into the **data_buffer** array starting from the next position. Each byte starts with '0' start bit. The stop bit '1' is merged with consequent EGT '1' bits and is checked together. The data extraction loop is terminated when any invalid bit is encountered or the stop bit is missing or the end of the buffer is reached.

The EOF check is omitted.

Long version

At first, Type B Data Extraction function calls SearchValidBit_14443 function to search for a first valid bit within the first 7 bits of **capture** array. If the valid bit position is found, the data extraction is started.

At the beginning of the data extraction, the first 10 data bits are extracted assuming it is the SOF. If non-zero bit is found within these 10 bits, the bits are treated as normal data bits and SOF is assumed missing.

Then, the response data bytes are shifted out into the **data_buffer** array starting from the next position. Each byte starts with '0' start bit. The stop bit '1' is merged with consequent EGT '1' bits and is checked together. The data extraction loop is terminated when any invalid bit is encountered or the stop bit is missing or the end of the buffer is reached.

The EOF check is accepted to terminate the data extraction.

9. EM4294 Crypto engine communication

Dedicated crypto engine implements the encryption algorithms. Currently, the EM4294 reader chip implements the EM4035 crypto algorithm (for the communication with EM4035 tags in secure mode) and standard 3 DES ciphering algorithm.

The EM4294 crypto communication routines implement a minimal set of operations defined in ISO 7816-3 document;

- Warm reset
- PPS command
- T=0 protocol – single block write and single block read transaction
- Clock stop

Crypto engine clock signal (SIM_CLK) is generated by the counter T0 in PWM mode, the 2MHz square signal. As the counter T0 is utilised also for data capture, the T0 SIM_CLK output has to be disabled for the necessary period (i.e. Clock stop). Therefore, block write and block read operations automatically restart the counter T0 in appropriate mode before the next SIM card operation is processed.

9.1. Warm reset

Crypto Engine warm reset is located in the simcard/SIM_Detect function.

SIM Card reset signal (SIM_RST) is asserted for ~450 SIM card clock cycles (SIM_CLK signal) during the warm reset operation. Then, 5 bytes of the Answer To Reset (ATR) response are expected to arrive on SIM_IO signal, the 1st byte (0x3B) has to be received within ~4700 SIM_CLK cycles.

If the ATR is received successfully, the SIM_CLK clock is left running and **sim_detected** variable is set. Otherwise the SIM_CLK is stopped and **sim_detected** variable is reset.

The SIM card type and version detection is left on the application software.

9.2. PPS command

Crypto Engine PPS command implementation is located in the simcard/ SIM_RSTPPS function. The goal of this function is to increase the communication speed from the default 372 SIM clocks per 1 ETU to the 32 SIM clocks per 1 ETU.
At first, Warm reset is invoked. If the SIM card is detected successfully and the SIM card ATR reports TA(1) = 0x95, the PPS command (0xFF, 0x10, 0x95, 0x7A) is sent to the SIM card. If correct PPS response is returned, the firmware timings constants are altered to 32 SIM clocks per 1 ETU until the next Warm reset is executed.

9.3. Single block write and block read

Single block write and single block read code is located in simcard/SIM_Command function.
The command sequence comprises;
- If the **sim_detected** variable is reset, quit the processing

- If the SIM_CLK is stopped, resume it

- Send the command header (5 bytes)

- Receive the ACK byte

- If incorrect or none ACK is received, quit the processing

- Send or receive data bytes, if any

- Receive the status bytes

Response length check, status bytes check, and other response proprieties check is left on the upper level routines or directly on the application software.

10. Debug functions

EMDB410 firmware contains lot of debugging code. Although the debugging code should be removed in the final release, the debugging code allows easier portion to other microcontrollers. Hence, the debugging code will be deleted only if insufficient code memory occurs.

Within most of the data capture routines, the debugging code parts can be found. Usually, the debugging code is located in the branch executed conditionally depending on the **debug_mode** variable. The purpose of the debugging code is the only one; to get the captured data of whatever form (i.e. raw envelope pulses array, decoded data in **capture** array, etc.) into the host PC so that the data can be checked and processed by the application software functions. Such feature allows the development/debugging/tuning of firmware routines in the PC instead of in the microcontroller simulator.

Warning: Any debugging code functionality is not assured.

11. Resource utilisation

Historically, the firmware source was developed for ATMega8 microcontroller, however, its code memory size allowed only one set of communication routines to be implemented, i.e. either ISO15693 or ISO14443.

EMDB408 ATMega64 microcontroller was chosen because of program memory capacity and additional timer T3 available. After the firmware compilation, the resulting firmware code length is about 21k Bytes and the performance is still sufficient.

It can be said that since the routines requiring high performance are the polling type, the interrupts may be shared for the routines requiring low performance. Currently, the timer T3 is not used by the EM4094 RFID Reader (part number: EMDB408) firmware, thus the complete firmware can be further optimised for ATMega32 or similar.

EMDB410 ATMega16 microcontroller is enough for single protocol communication (either ISO15693+EM4006 or ISO14443). ISO15693+EM4006 firmware code length is about 12kB, ISO14443 firmware code length is about 10kB.

However, ATMega16 microcontroller code memory capacity appears not sufficient for creating stand-alone readers (e.g.; door access systems) because the sophisticated high level "application" functions. Also, the SRAM capacity (1kB) becomes insufficient in the applications where the microcontroller large data structure internal operations are expected.

Therefore, it can be stated that the ATMega32 microcontroller is the optimal microcontroller with balanced code/SRAM/features parameters for the usage with EM4294 reader chip.

© EM Microelectronic-Marin SA, 04/07, version A