Application Note 430

Title: **EM4095 RFID Reader Firmware Description**

Product Family: **RFID**

Part Number: EMDB409

Keywords: EM4095, EMDB409, ISO 11784/ 11785, Read Only, EM4005, EM4105, EM4100, EM4102, EM4150, EM4350, EM4450, EM4550, EM4469, EM4569, EM4026, EM4205, EM4305

Date: January 24, 2008

1. Introduction

EMDB409 Reader is a base station for communication with a selected set of 125 kHz transponders.

This document describes programmer notes concerning the EMDB409 Reader firmware structure and function and it should be treated as extending information to the AN428 document (EM4095 RFID Reader – Description of Firmware protocol) which chapters are referred to.

In spite of comprising many Atmega64 dependent features, 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. EMDB409 Reader firmware source files are written in C programming language and targeted for ATMega64 microcontroller family. The firmware communicates with the respective software application whose source files are also available.

2. Environment setup

Following tools were used:

1. Compilation - the whole code can be compiled via make/gcc port for ATMega64 chip family. Make/gcc for AT64Mega is free under OSI approved licence and can be obtained at http://sourceforge.net/projects/winavr/. See Readme.txt included in source package file for actual compiler release used.
2. Programming[1] - the ATMega64 chip is equipped with standard serial programming interface. It can be programmed via Xilinx Parallel Cable III+ and uisp programming utility that is also included in the WinAVR gcc package. When appropriate firmware is already present in the ATMega64 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 Atmega64 chip configuration (fuse bits) needs to be set (e.g., see fuse.bat in the source package) before first programming

3. Firmware Description

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

1. Level1 - defines decoding routines
2. Level2 - defines low level uplink (send) and data extraction routines
3. Level3 - defines high level data transformations and **main loop** body, and bootloader
4. Level4 - defines UART communication routines

3.1. 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, T3 for reception, counter T1 for transmission) that are incremented by RF clock signal (RDY_CLK signal of EM4095). 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_data** and **capture_valid** arrays indexed by **capture_cnt** and **capture_bit_count** variables. Capture arrays are 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 in the capture arrays.

Such philosophy gives a quite serial and deterministic behaviour without need of asynchronous process communication (except EM4026 slow down/switch off scan code) or priority re-entrant interrupt handlers.

EMDB409 (125kHz reader) firmware structure is similar to the EMDB408 (13.56MHz reader) firmware structure. Many information items are common to both firmware implementations.

3.2. Source files

- Makefile
- Batches
  - gcc.bat – invokes the compilation
  - fuse.bat – initialises the uC fuses
  - prog.bat – uploads the firmware into the uC
- Level 1
  - level1.h – declares common decoding variables and functions
  - level1_4026.c – defines EM4026 data capture functions
  - level1_41xx.c – defines EM4x50 data capture functions
  - main.c – contains main entry point , uC resources initialisation, and general capture functions
- Level 2
  - level2.h – declares common uplink variables, functions, and uC port initialisation
  - level2_4026.c – defines EM4026 uplink and data extraction functions
  - level2_41xx.c – defines EM4x50 uplink and data functions
  - level2.c – defines general uplink routine and data extraction routines (EM4x69, EM4205/EM4305)
- Level 3
  - level3.h – declares common main loop routine, execution functions, and variables

- level3_4026 – defines EM4026 execution functions
- level3_41xx – defines EM4x50 execution functions
- level3.c – defines main loop and main execution functions
- 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

To compile the source files, run <make> command in the shell window. Check the compilation output for errors. The main.hex bitstream file is generated if no error occurs. Run <prog.bat> or use a bootloader to program main.hex firmware into the microcontroller device.

3.3. Microcontroller Start-up

After power-up, the microcontroller enters the bootloader section (see AN428 document for bootloader description). The bootloader is bypassed if entered because of watchdog. Then, main.c/main() initialises the uC resources including the uC port settings and directions and passes the control to level3/main_receiver().

3.4. Level 3

Level3 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 AN428 for each execution command description. Each action should generate at least one response at bounded time.

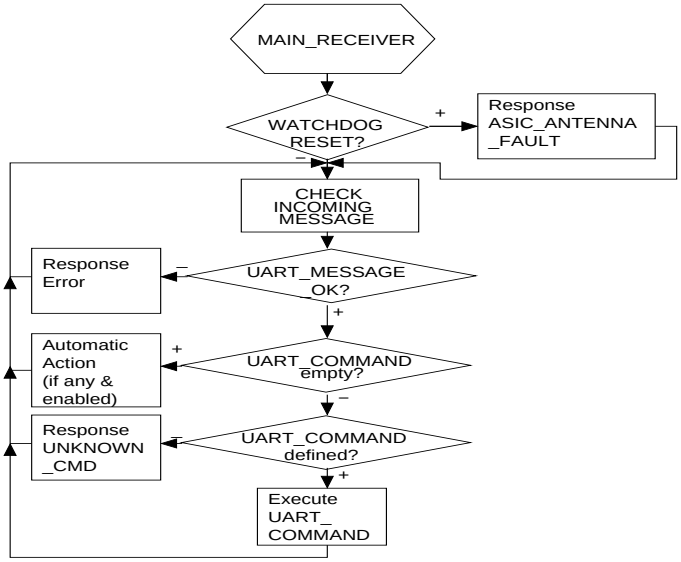

Figure 1 Main_receiver loop

3.5. Level 4

Level4 defines UART communication routines.

3.5.1. PC -> Reader

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.

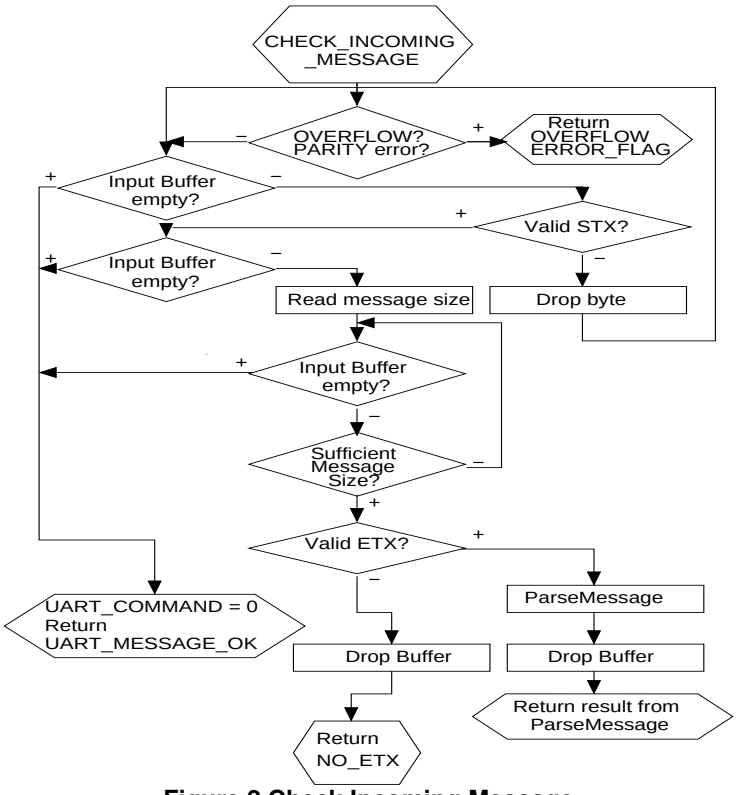

Figure 2 Check Incoming Message

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() (see Figure 3 Parse Message (level4.c)). Valid message length is checked and message useful information is copied into appropriate variables.

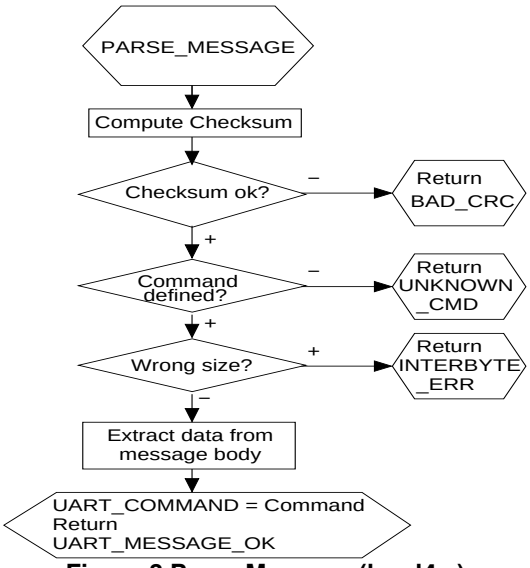

Figure 3 Parse Message (level4.c)

3.5.2. Reader -> PC

Response can be sent using one of the response forming routines according to the transported data contents. All these routines are blocking, i.e. they do not return until the whole response is transmitted.

4. EM4x69

Generally, level2 defines transmission interrupt routine and low level data preparation routines to send the command and capture the response.

Low level data preparation routines initialise proper values into the timer counters and enable their respective flags. Data transmission interrupt is generated by counter T2 overflow event. The counter T2 is always reprogrammed according to the pulse or bit period lengths. The new value of MOD pin is set in bounded time (not necessarily constant time).

4.1. 4050 Uplink routine

Since EM4x69, EM4205/EM4305, and EM4x50 use the same uplink encoding (4050 encoding), the uplink routine is unified. The 4050 encoding uplink routine is located in level2/SIG_OVERFLOW2 routine.

4050 encoding uplink routine is interrupt driven final state machine that generates appropriate 4050 encoding pulses on MOD signal. The 4050 encoding final state machine has 4 states (‘0’ bit-modulation, ‘0’ bit -no modulation, ‘1’ bit – modulation, ‘1’ bit – no modulation phase) which timings is defined by **field_stop**, **fwd_01_stop**, **fwd_01_zero**, and **fwd_01_one** variables.

Uplink command bits processed by 4050 encoding uplink routine are fed from **forwardLink_data** array. The contents of this array is prepared be level3 routines using level2 Prepare_Cmd, Prepare_Data, and Prepare_Addr functions, or by their modified versions in level2_41xx level.

4.2. EM4x69 data capture

EM4x69 response capture and decoding operation is performed at once by following routines;

- main/manchester_capture
- main/biphase_capture
- main/miller_capture

Any decoding routine is called from T1 capture interrupt handler via pointer to function. During the reception, the T1 counter counts the number of RF clocks between two consecutive edges of DEMOD signal. Each edge detection causes timer counter value capture and invokes capture interrupt. The flow of consequent DEMOD pulses' lengths is processed by selected decoding routine. Decoding routines are implemented as small final state machines with more or less accurate decode detection.

Each routine emits the pair (decoded bit, decode invalid flag = 0) for each bit (not necessarily on each call). If decoded bit flow is considered to be broken (final state machine encounters the state of bad decoding), only one pair having decoding invalid flag bit set to 1 is emitted. Thus, wrong or noisy sequence can be reduced to 1 pair only since no useful information is received. Emitted data pairs are stored in **capture_data** and **capture_valid** arrays. Decoding routines are empirical algorithms that trade off among performance, reliability and robustness. For example, see Figure 3 describing Bi-phase type decoding routine flow.

If necessary, the capture routines are responsible of triggering the capture even sensitive edge of DEMOD_OUT input signal.

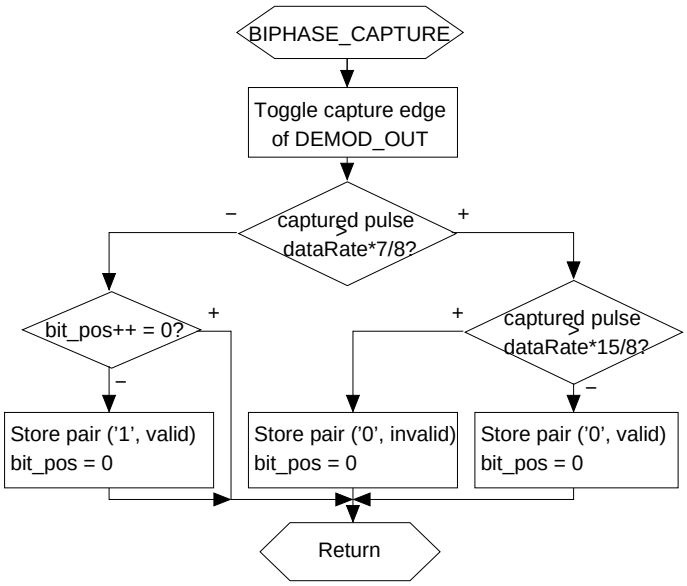

Figure 3. Biphase Capture function

4.3. EM4x69 data extraction

The EM4x69 data extraction routine is located in level2/Extract_data function. Data extraction of desired format is performed off-line.

At first, the level3 high level routines (e.g.; level3/WriteWord, level3/ReadWord) search the ACK and NACK pattern within the first 10 captured valid bits using level2/SearchPattern function. Level2/ExtractData function then extracts the EM4x69 block data starting from the position supplied from SearchPattern function. When the EM4x69 data block is extracted successfully, the parity is checked by forming the data-block-to-be-sent structure and compared to the received structure. The extracted data is stored into the **read_tag_memory_word_low** and **read_tag_memory_word_hi** variables.

5. EM4205/EM4305

As the EM4205/EM4305 tag is the successor of EM4x69 tags, the most of the EM4x69 code is used for EM4205/EM4305 communication without change.
The difference is in;

- Level2.c/SendForward() – longer uplink timings constants and small synchronization delay
- Level3.c/CheckConfiguration() – EM4205/EM4305 dedicated command to response time-out constants
- Level3/level4 – EM4205/EM4305 dedicated UART command set (mirroring the EM4x69 command set) to select the above mentioned uplink variables and time-out constants, and the new EM4205/EM4305 Protect command

6. Animal mode tags

The main Animal mode tags capture code is located in level3/AnimalCapture function.

The animal capture function performs a dummy read data capture in bi-phase RF/32. The **capture_data** and **capture_valid** array are examined off-line to find the animal mode data header at arbitrary position. If the animal mode data header is found, the animal data is extracted into the **Animal_data** array using the level2/AnimalTestValidRange function.

Then, all the 7bit parts of animal data are fed into the animal mode CRC state routine. The stuff-bits are not checked.

7. Read Only tags

The main read only (RO) capture code is located in level3/ROCapture_lowLevel function.

The ROCapture_lowLevel routine is called 4 times with all possible data capture settings (i.e. Mn/32, Mn/64, Bi/32, Bi/64). The ROCapture_lowLevel routine performs a dummy read data capture in the specified mode. Then, the **capture_data** and **capture_valid** arrays are examined to find the RO data header (9x'1' + 1x'0' seqeunce) at arbitrary position. If the valid RO header is found, the validity of adjacent expected range of capture response bits is checked by level2/TestValidRange function which also stores the data into the **RO_value_hi** and **RO_value_low** variables. If all the capture response bits are valid, the RO data formatting is checked by level3/CheckRODataData function.

CheckRODataData function compares the parity in the capture data arrays to the parity generated from already extracted RO data value.

8. EM4x50

EM4x50 related specific code is referred as 41xx.

8.1. EM41xx Uplink routine

EM4x50 LIW uplink routine is shared with EM4x69 routine, see chapter 4.1.

Before the 4050 uplink routine is called, the firmware has to synchronise to EM4x50 LIW.

Level2_41xx/Prepare_Data_4150 function is used to format the uplink message according to the input parameters.

8.2. LIW, ACK, NACK capture routine

The LIW synchronization routines are located in level1_41xx/LIW_capture (EM4x50).

LIW capture routines determine the ratio of the measured length of the pulse recently received and the expected half data rate (**halfDataRate** variable) including the tolerance. The sequence of the ratios (**cpt_bits** and **cpt_valid** values) is compared to the expected sequence of LIW ratios as same as the ACK and NACK ratios. **Capture_result** global variable contains the result code.

Generally, the routine behaviour is controlled by **enable_capture** variable. When synchronizing to the LIW, two cases can occur. The first case is the synchronisation to LIW to transmit the command - as soon as the routine identifies the first part of LIW, it disables the capture immediately so that the first uplink modulation can be transmitted on specified time. The second case is a almost complete capture of LIW so that the data capture routine can be start with defined phase.

To determine the actual EM4x50 tag data rate, the LIW capture routine has to be called twice with the different expected data rate (16 for RF/32, 32 for RF/64) settings.

8.3. EM41xx data capture

EM41xx data capture routine is not a standalone function, see level3_41xx/ReadWord_4150 and level3_41xx/ EM4150_Write functions instead.

According to the current command, each EM41xx response capture routine captures synchronizes to the appropriate number of ACK/NACK/LIW using the LIW synchronization routine. If the data block is expected, the main/ Manchester_capture_4150_4026 function is invoked to capture the data word bits.

8.4. EM41xx data extraction

EM41xx data extraction routine is located in level2_41xx/ExtractData_4150 function.

Single word data is extracted from the **capture_data** and **capture_valid** arrays, the parity check is perfomed. The extracted data is stored into the **read_tag_memory_word_low** and **read_tag_memory_word_hi** variables. As the data capture is synchronised, the data extraction starts always from the bit no.1.

9. EM4026

There are three types of EM4026 communication routines currently implemented in the EMDB409 firmware;

- Free running scan of EM4026 tags
- Slow down/switch off scan of EM4026 tags
- SendCodeID command for single EM4026 tag

9.1. Low level routines

9.1.1. EM4025 Uplink routine

EM4026 mute pulse and send IDCode command uplink function are located in level2/SIG_OVERFLOW2 routine.

Both commands consist of just a few of pulses which respective timings is defined in **fwd_tgap**, **fwd_ack**, and **fwd_tsendcode** variables.

9.1.2. EM4026 Data capture

The EM4026 data capture routine is located in main/manchester_capture_4150_4026 function while the setup code is located in level1_4026/Capture_4026 function.

Manchester_capture_4150_4026 function is interrupt driven routine that process the pulse lengths similar way as other capture routines. The capture routine decodes the Manchester data according to the **halfDataRate** variable and emits the bit pairs into the **capture_data** and **capture_valid** arrays.

9.1.3. EM4026 Data extraction

EM4026 Data extraction is performed by level2_4026/FindEM4026UID function.

FindEM4026UID function searches all the 64 valid bits sequences in the **capture_data** and **capture_valid** arrays content starting from position specified by **ptr** input parameter. As soon as such valid sequence is found, the CRC check is performed. If the CRC check passes, the EM4026 UID sequence content in the **data_buffer_4026** array is valid.

FindEM4026UID function returns the position of last bit examined so that the resumption of the next search at this position is allowed.

9.2. SendCodeID

SendCodeID code main body is located in level3_4026/SendCodeID function.

SendCodeID function invokes the EM4026 uplink routine to generate the SendCodeID command. Then, it calls level1_4026/Capture_4026 function to capture the response. Finally, the capture data arrays are examined off-line using FindEM4026UID function. EM4026 low level routines described above are used without any special feature.

9.3. Free running scan

Free running scan code is located in the level3_4026/FreeEM4026Scan function.

At first, the EM4026 capture function is executed whereas the EM4026 UIDs are captured asynchronously. Then, the capture data arrays are examined off-line using FindEM4026UID function. Every EM4026 UID found is stored into the **found_uids** array. The returned **found_uids** array may contain more than one instance of each UID. EM4026 low level routines described above are used without any special feature.

9.4. Slow down/switch off scan

Slow down/switch off scan main body is located in level3/SlowOrSwitchEM4026Scan function.

SlowOrSwitchEM4026Scan function violates the overall philosophy of the firmware. The violation has been forced because of the short period of time (Tackstart, see EM4026 datasheet) to respond with ACK command a after the last bit of the UID has been received successfully. Current uC performance and off-line UID extraction exceed this period and hence cannot be used.

While the off-line EM4026 extraction time exceeds the Tackstart period, the ACK command can be sent in time only if the EM4026 extraction routine is running in parallel to the EM4026 data capture routine.

Therefore, Capture_4026 function is invoked in the mode, when manchester_capture_4150_4026 interrupt driven function runs with high priority and FindEM4026UID function is run within the wait loop. While the capture routine adds new bits into the **capture_data** and **capture_valid** arrays, FindEM4026UID function is restarted over these arrays incrementally, i.e. starting position is incremented according to the result of previous calls. As soon as FindEM4026UID function finds a valid EM4026 UID, the capture process is stopped and SlowOrSwitchEM4026Scan function generates the ACK command.

After transmission of ACK command, each EM4026 UID is passed to the level2_4026/CheckAndStoreUID function that checks whether the currently found EM4026 UID is unique. In such case the EM4026 UID is stored into the **found_uids** array. Finally, capture process is restarted until the full scan time elapses.

The full scan time is specified by **maxCaptureTimeHi** and **maxCaptureTimeLow** variables as same as for the rest of capture routines. When the Capture_4026 routine is terminated before the full scan time elapses, these values are saved and reused in the next Capture_4026 routine invocation.

The capture process is also stopped when the **capture_data** and **capture_valid** arrays reach full capacity because of invalid bits. In this case the SlowOrSwitchEM4026Scan restarts the capture process with these arrays flushed. Any potential UID that is currently received is lost.

10. Debug functions

EMDB409 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 uC 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 a sub-set of communication routines to be implemented, i.e. EM4x69 command set or EM41xx command set.

ATMega64 microcontroller was chosen because of program memory capacity and additional timer T3 available. After the firmware implementation, the firmware code length is about 21kB and the performance is still sufficient.

The complete firmware can be further optimised for ATMega32 that does not posses T3 counter and has enough code memory.