International Journal of Electrical and Computer Engineering (IJECE)
Vol. 7, No. 4, August 2017, pp. 1867~1873
ISSN: 2088-8708, DOI: 10.11591/ijece.v7i4.pp1867-1873

Security Enhancement in Networked Embedded System

Pradip Ram Selokar[1], P. T. Karule[2]
[1]Departement of Electronics and Communication, RCOEM, Nagpur, Maharashtra, India
[2]Departement Electronics Engineering, YCCE, Nagpur, Maharashtra, India

Article Info

Article history:

Received Feb 7, 2017
Revised April 24, 2017
Accepted Apr 10, 2017

Keyword:

Bootloader
CAN – controller area network
Flash Write
Intel Hex file format
Signature Authenticity
UART – universal asynchronous receiver transmitter

ABSTRACT

In the developed system ARM9 is a master and Two ARM7s are slaves. The peripherals are being controlled by two ARM7 boards. The Peripherals are connected to the ARM7 through Complex Programmable Logic Device (CPLD). The CPLD is in turn connected to the ARM7 using Serial Peripheral Interface (SPI). The ARM7 boards collect the information from the peripherals and send it to the ARM9 board. The communication between ARM7 and ARM9 is via UART (Universal Asynchronous Receiver Transmitter) over CAN (Controller Area Network). The ARM9 board has got the software intelligence. The ARM9 behaves as a master and two ARM7 boards behave as slaves. Being master ARM9 passes tokens to ARM7 which in turn returns (Acknowledges) the token. The ARM9 is further connected to Proxy via Ethernet. The proxy is further connected to the service platform (server) via Ethernet. So subsequently any decisions at any stage can be changed at server level. Further these commands can be passed on to ARM9 which in turn controls the peripherals through ARM7. (a) The system which we have developed consists of ARM9 as a master, Two ARM7 as Slaves. The communication between ARM9-ARM7 is via UART over a CAN, (b) Each ARM7 further communicates serially (RS232) with the two 8051 Microcontroller nodes, (c)Thus a networked Embedded System is developed wherein the serial data is brought over Ethernet. The ARM7 board, which is directly linked with the peripherals, can be modified of its functionality as and when required. The functionality of ARM7 can be modified by upgrading its firmware. To upgrade the firmware same communication link has been used. ARM7 receives the new firmware via same ARM9-ARM7 communication link. The Flash Write operation is performed using the source code to write the new firmware. Bootloader application for the ARM7 has been developed. The signature has been incorporated to assure authenticity of the new Firmware. Intel Hex File Format is used to parse the hex file.

Corresponding Author:

Pradip Ram Selokar,
Department of Electronics and Communication Engineering
Shri Ramdeobaba College of Enginering and Management,
Nagpur, India.
Email: pradip.selokar@gmail.com

1. INTRODUCTION

Security is a measure of how difficult it is to break into the object which the system protects. Embedded systems present significant security challenges due to their limited resources and power constraints. In addition to standard concerns regarding system performance and power consumption, security has become a leading issue for many embedded applications [1].

In the developed system firmware upgrade is carried out using the data communication link. For the development purpose the ARM7 board has got the J-link interface. It is very easy to burn the hex file to the

Flash memory of ARM7. But finally when the entire firmware development is completed and the product is ready to be sold in market, such J-link interfaces will be removed. The ARM7 board will be mounted in a control panel. This control panel would be located at remote locations inside the building. If the system works fine, it is well and good.

Think of a situation, in future the customers complain about the bugs in the system. Will it be possible for the manufacturer to dismantle the entire assembly, take it back to the developer's site and modify the functionality? Obviously not! And also it would not be cost wise affordable to the manufacturer.

Hence it was a necessary requirement while developing the firmware for ARM7, to have the capability to receive the new firmware via same ARM7-ARM9 communication link and overwrite the old one.

When old firmware is to be overwritten, it means that the Flash write operation has to be performed using the source code (Which otherwise would have been performed by J-link). The key task in Firmware upgrade was Flash Write.

The token based communication protocol between one ARM9 (AT91SAM9260) board [2] and two ARM7 (AT91SAM7S256) boards has been developed. In this communication link the ARM9 acts as Master and two ARM7 act as Slaves. The main constrain for setting up this communication link was TIME. i.e. peripheral should get the quicker response from the server. Often the performance in embedded devices is limited by the firmware which runs over it. Traditionally systems on devices have been designed using ATMEL 89C51 Microcontroller, PIC Microcontroller etc. These microcontrollers support the communication using RS-232 or RS-485. Here the ARM Microcontrollers are used. The ARM microcontrollers are having fast execution speed (MIPS – million instructions per second), thereby meeting the basic constrain (TIME) of this control system.

In the simplest form it can be explained as, half duplex communication has been established between server and peripherals. In one direction the data collected by peripherals is transmitted to server. In other direction the command from server is transmitted to the peripherals. Figure 1 gives the block diagrammatic view of the System.

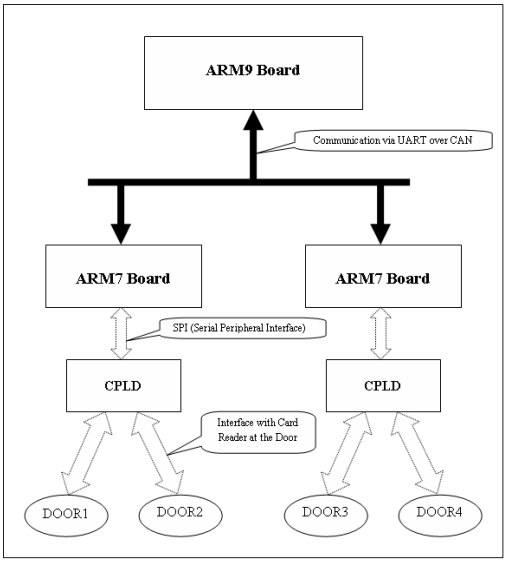

Figure 1. Block Diagram

2. COMMUNICATION LINK

2.1. UART (Universal Asynchronous Receiver Transmitter)

The Universal Asynchronous Receiver Transceiver (UART) provides one full duplex universal asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver timeout enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission. The UART features three test modes: remote loopback, local loopback and automatic echo.

2.2. Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the Control Register (US_CR). However, the receiver registers can be programmed before the receiver clock is enabled. After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in the Control Register (US_CR). However, the transmitter registers can be programmed before being enabled. The Receiver and the Transmitter can be enabled together or independently. At any time, the software can perform a reset on the receiver or the transmitter of the UART by setting the corresponding bit, RSTRX and RSTTX respectively, in the Control Register (US_CR).

2.3. Transmitter Timeguard

The timeguard feature enables the USART interface with slow remote devices. The timeguard function enables the transmitter to insert an idle state on the TXD line between two characters. This idle state actually acts as a long stop bit.

2.4. Receiver Timeout

The Receiver Time-out provides support in handling variable-length frames. This feature detects an idle condition on the RXD line. When a time-out is detected, the bit TIMEOUT in the Channel Status Register (US_CSR) rises and can generate an interrupt, thus indicating to the driver an end of frame.

2.5. CAN (Controller Area Network)

CAN is a broadcast serial bus standard for connecting electronic control units (ECUs). Each node is able to send and receive messages, but not simultaneously: a message (consisting primarily of an ID – usually chosen to identify the message-type/sender – and up to eight message bytes) is transmitted serially onto the bus, one bit after another – this signal-pattern codes the message (in NRZ) and is sensed by all nodes.

The devices that are connected by a CAN network are typically sensors, actuators and control devices. A CAN message never reaches these devices directly, but instead a host-processor and a CAN controller is needed between these devices and the bus.

If the bus is free, any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits i.e. bit 0) will overwrite other nodes' less dominant IDs, so that eventually (after this arbitration on the ID) only the dominant message remains and is received by all nodes.

3. COMMUNICATION PROTOCOL

The communication between ARM7 and ARM9 is via UART (Universal Asynchronous Receiver Transmitter) over CAN (Controller Area Network). The ARM9 behaves as a master and two ARM7 boards behave as slaves. Being master ARM9 passes tokens to ARM7 which in turn returns (Acknowledges) the token. The ARM9 is further connected to Proxy via Ethernet. The proxy is further connected to the service platform (server) via Ethernet. In the simplest form it can be explained as, half duplex communication has been established between server and peripherals. In one direction the data collected by peripherals (User swipes the card at the reader connected to door and reader reads (extracts) the card number) is transmitted to server [3]. In other direction the command from server is transmitted to the peripherals (Based on the card number received server sends access granted/denied command to ARM7, which in turn takes the corresponding action (turn ON/OFF) on the Relay connected at the door to ultimately open/close the door).

3.1. UART Initialization

a. The ttyS4 port (UART3) of ARM9 is opened using the file operation. The file descriptor is used to access this port.
b. While initializing UART following settings are done through the source code,
   1) Baud Rate: 115200
   2) Data bits: 8 bits
   3) Parity: None
   4) Stop Bit: 1 bit
   5) Hardware flow Control: No
c. The UART0 of ARM7 is accessed using its base address (AT91_BASE_US0) and similar settings are done through the source code.
d. A communication link between these two ports (ttyS4 of ARM9 and UART0 of ARM7 is set up. The physical medium is CAN bus.

This is how ARM7–ARM9 physical communication link was setup. Initially there wasn't any prescribed communication packet format. The very first attempt to send data over this communication link was done by transmitting just the single character (1 byte) from ARM7 to ARM9 using 'putchar' function in C. This byte was received on the ARM9 side using 'getchar' function in C. This was successful. Later on communication with single byte but both transmitting and receiving was tried successfully over this communication link. At this stage a functional communication link was established. Now the next task was to decide the format of the communication packet. Table 1 lists the prescribed packet format for the ARM7–ARM9 communication.

Table 1. ARM7–ARM9 Communication – Prescribed Packet Format

2 Bytes	1 Byte	2 Bytes	1 Byte	1 Byte		1 Byte	1 Byte	1 Byte	1 Byte	2 Bytes
Start of packet	Address of board	Sequence Number	Number of Bytes	Packet Type	DATA	Sensor Status	Relay Status	Checksum	ACK/NACK	End of Packet
0,1	2	3,4	5	6	Max 256 Bytes	N-6	N-5	N-4	N-3	N-2, N-1

3.2. Checksum

For both, the data received in the packet and the data to be transmitted in the packet the 8 bits checksum is calculated on both the sides ARM7 and ARM9. The logic used for calculating the checksum is XORing the data in the packet byte by byte.

Based on the checksum criterion the integrity of the data received in the packet is checked i.e. the data is said to be error free if the received checksum (Checksum field in the packet) and the calculated checksum are matching. The data is added to the receive queue only if its integrity is validated [4].

3.3. ACK/NACK

If the integrity of the received data is validated on the basis of checksum, the ACKNOWLEDGEMENT (ACK) is sent back by piggybacking in the next packet, otherwise NO_ ACKNOWLEDGEMENT (NACK) is sent. Whenever NACK is received the packet is resend.

The packet resending is performed for maximum of three times. If the problem still persists an event is send to the proxy letting it know about the problem in ARM7–ARM9 communication link. In turn the server will come to know about the communication problem.

For resending of the data, whenever packet is sent its copy is written in the BACKUP SEND QUEUE. If NACK is received for the last sent packet, the same packet is read from the BACKUP SEND QUEUE and resend. However if the ACK is received for the last sent packet, the BACKUP SEND QUEUE is read and it is freed [5].

4. FIRMWARE UPGRADE OF ARM7

The ARM7 board, which is directly linked with the peripherals at the door, can be modified of its functionality as and when required. And to modify the functionality, the board need not be dismantled from the control panel assembly mounted in a remote location within the building.

How is this possible? The answer to this question lies in this paper. This paper explains in detail the Firmware Upgrade Procedure of ARM7 board using the same ARM7-ARM9 communication link. The main emphasis is given on the Flash write operation, Intel hex file format and signature for the new firmware [6].

4.1. Bootloader

Bootloader is a piece of code that runs before any main application is running. Since it is usually the first software to run after power up or reset, it is highly processor and board specific. The bootloader performs the necessary initializations. The bootloader application uses the same token based communication protocol to communicate with ARM9. It is different from the main application in the sense that it does not perform any other functionality except for Flash write operation. Following steps explain in detail the development of bootloader application. Bootloader is located at Flash address 0x0000. The 18.943 KB of Flash memory has been assigned for Bootloader (0x0000 – 0x4FFF). Rest of the Flash memory has been divided into two Regions, each of size 120.832 KB. The Region a (Master Region) is between 0x5000 – 0x227FF. The Region B (Secondary Region) is between 0x22800 – 0x3FFFF.

Table 2 gives the details of memory partitions. Now there are two cases for bootloader to handle

Table 2. ARM7 – Flash Memory Partition

Bootloader Start Address	0x0000
Bootloader End Address	0x4FFF
Region 'A' (Master Reg) Start Address	0x5000
Region 'A' End Address	0x227FF
Region 'B' (Secondary Reg) Start Address	0x22800
Region 'B' End Address	0x3FFFF
Main Application Start Address	Reg A Start Address
Region Size	0x1D800 Bytes (120.832 KB)

The Firmware Upgrade command is received from ARM9. On receiving this command ARM7 is reset using software reset. On reset, execution goes to Flash start address 0x0000 where bootloader is located.Bootloader waits for 10 Seconds to receive new firmware bytes, otherwise jumps back to Region A. The message 0x01 is displayed on 7-Seg Display on ARM7 indicating availability of application firmware in Region A.

If it receives the new firmware bytes, the receiving are line by line (one line is of 16 bytes in parsed hex file). The memory page of these bytes is formed according to the addresses received (Each page is of 256 Bytes = Flash Page Size).

After completion of one page ARM9 sends the WRITE command to ARM7. On receiving this command that page is written to flash memory in Region B. This procedure goes on till the entire firmware is received and written to Region B.

Once complete new firmware is available in Region B, its signature is verified. If signature verification validates then new firmware is copied to Region A from Region B, replacing the old firmware. If signature verification fails, message is sent to ARM9, and Old firmware in Region A is not replaced by this new firmware in Region B.

After finishing the copying of new firmware from Region B to Region A, ARM7 sends back FW UPGRADE SUCCESS message to ARM9 and jumps to main application from bootloader (to Region A where new firmware is available now).

4.2. Flash Write Operation

The Flash memory is written in a page-by-page fashion [7]. The write is carried out by storing data for an entire page into a temporary page buffer prior to writing the Flash. Which Flash address to write to is decided by the content of the parsed Intel hex file. A flash page has to be erased before it can be programmed with the data stored in the temporary buffer. The write operation uses the following procedure when writing a Flash page:

a. Fill temporary page buffer
b. Erase Flash page
c. Write Flash page

As one can see of this sequence there is a possibility for loss of data if a reset or power failure should occur immediately after a page erase. Loss of data can be avoided by taking necessary precautions in software, involving buffering in nonvolatile memory [8]. Figure 2 gives the logic of flash write operation.

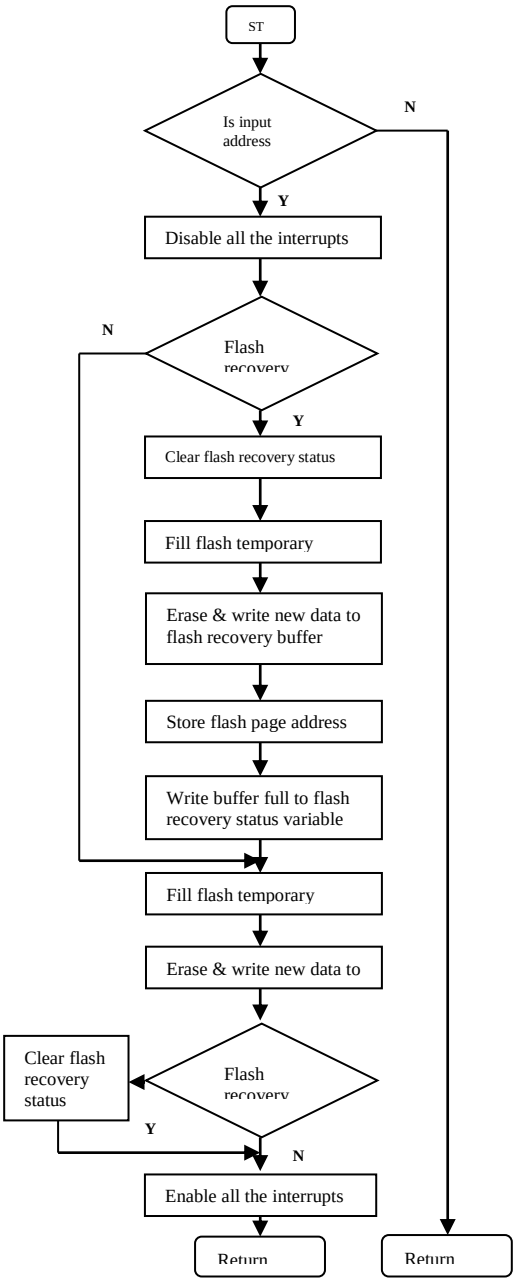

Figure 2. Flowchart – Flash Write Procedure

4.3. Signature Authensity for Security Enhancement

The signature [9] has been incorporated to assure that the firmware is issued from the authentic manufacturer. While developing the main application the predefined signature has been written in a predefined memory address at a compile time itself. To write at compile time following are the steps.

Use of compiler directive, "# pragma location = 'SECTION_NAME' " has been made to reserve the predefined memory location for signature.

The signature data is written in the above section by declaring the signature data array immediately following the "# pragma location" directive.

The location address for the section 'SECTION_NAME' has been defined in the at91SAM7S256_FLASH.icf file. This predefined memory location is locked permanently with the predefined signature characters written on it. This region is never overwritten.

Whenever there is a need of firmware upgrade, it is first received to the secondary region of the memory. Then the signature bytes are compared with the predefined signature. If it matches then only the old

firmware is overwritten with the new one otherwise it is discarded. This ensures the security of the firmware preventing it from getting corrupted from external sources.

5. CONCLUSION AND COMPARISON

The new firmware is overwritten to the ARM7 through source code using the same communication link through which the data communication is carried out. This eliminates the need of dismantling the hardware from the remote location for the purpose of firmware up gradation.

In the past people have people have suggested firmware upgrate to an embedded system through J-Link/J-Trace. This method of upgradation requires dismantling of the hardware from remote location and taking it to the developer's site. This would be a costlier option.

Also Signature takes care of Authenticity. So due care has been taken to overwrite the old firmware with the new authentic one thereby enhancing the security of the networked embedded system developed. The technique of Firmware upgrade presented in this paper will make the hardware multifunctional against the application specific. Also the same hardware can be modified of its functionality in future as per the need and new peripherals.

In the past people have people have suggested security based on error control codes. In this paper we have suggested signature based security which is more efficient as the security code is known only to the authentic manufacturer.

REFERENCES

[1] Norfadzlia Mohd Yusof, Aiman Zakwan Jidin, Lim Mei Sze, "Web Based Home Security, Automation System", *International Journal of Reconfigurable and Embedded Systems IJRES*, 2016; 5(2); 92-98.

[2] Vinayak Pandit K., Sanket Dessai, Shilpa Chaudhari, "Development of BSP for ARM9 Evaluation Board", *International Journal of Reconfigurable and Embedded Systems IJRES*. 2015; 4(3); 161-172.

[3] Mike Meyerstein, Inhyok Cha, Yogendra Shah, "*Security Aspects of Smart Cards vs. Embedded Security in Machine-to-Machine (M2M) Advanced Mobile Network Application",* Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, Revised Selected Papers – Springerlink. (June 3-5, 2009)

[4] Ou Qingvu, Huang Kai, Wu Xiaoping, "*Research on the Embedded Security Architecture Based on the Control Flow Security",* IEEE Computer Society: Second International Workshop on Computer Science and Engineering (2009).

[5] Chandrasekaran, S. Rajendran, J. Annamalai, "*Data Driven Security Alarm Model for Embedded Applications",* Computing, Communication and Networking, ICCCn 2008, International Conference – IEEE xplore digital library (2008).

[6] Guy Gogniat, Tilman Wolf, et al., "Reconfigurable Hardware for High-Security/high-Performance Embedded Systems: The SAFES Perspective", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, February 2008; 16(2).

[7] Ted Huffmire, Brett Botherton and others. Managing Security in FPGA based embedded Systems. *IEEE CS digital Library*. November-December 2008; 25(6).

[8] R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, W. Burleson, "A Security Approach for off-chip Memory in Embedded Microprocessor Systems", *Elsevier Journal of Microelectronics and Microprocessors*, 2000.

[9] T. Eisenbarth, T. Guneysu, C. Paar, A.-R. Sadeghi, D. Schellekens, M. Wolf, "*Reconfigurable Trusted Computing in Hardware*", In Proceedings of the ACM Workshop on Scalable Trusted Computing. (November 2007)

[10] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, M. Bardouillet, A. Martinez, "*A Parallelized way to Provide data Encryption and Integrity Checking on a Processor-Memory bus",* In Proceedings of theIEEE/ ACMInternational Design Automation Conference (July 2006).