Diagnostic Application for Development of Custom ATCA Carrier Board for LLRF

Jan Wychowaniak, Paweł Prędki, Dariusz Makowski, Bartosz Sakowicz, Andrzej Napieralski
Department of Microelectronics and Computer Science
Technical University of Lodz
Lodz, Poland
jan.wychowaniak@gmail.com

***Abstract*—The Advanced Telecommunications Computing Architecture (ATCA) standard describes a powerful and efficient platform. With multiple integrated solutions like redundancies and intelligent control mechanisms this technology is characterized with reliability estimated at the level of 99.99999 percent. These features make the standard perfect for use in projects like the Free Electron Laser in Hamburg (FLASH) and the X-ray Free Electron Laser (X-FEL) in order to help them meet the requirements of high availability and reliability. The ATCA standard incorporates advanced control systems defined in the Intelligent Platform Management Interface (IPMI) specification as one of the key elements. The entire ATCA implementation retains its functionality as long as the IPMI remains operational. The complexity level of the application increases, which results in preparing it to run and debugging being more difficult to perform. At the same time, only scrupulous elimination of any kind of possible deficiencies can enable the ATCA implementation to offer the desired level of reliability. Thus, diagnostics become crucial, which creates a need for additional tools performing these tasks during the preparations of both hardware and software for the ATCA application.**

The paper presents application aiding in development of the prototype Carrier Board by enabling the user of external PC station to perform diagnostic and control activities over the Board. It helps in examining all its components at the stage of running the Board, as well as in further operation analysis.

***Index Terms*—Advanced Telecommunications Computing Architecture; Intelligent Platform Management Interface; Carrier Board; EIA RS-232; Java; X-ray Free Electron Laser**

I. Introduction

The Advanced Telecommunications Computing Architecture (ATCA) standard includes Intelligent Platform Management Interface (IPMI) as the platform control system [1]. Thus, each Carrier Board (CB) used in the ATCA implementation is required to be equipped with an IPMI control unit. This unit is referred to as the Intelligent Platform Management Controller (IPMC) [2-4]. Primarily, IPMC communicates with ATCA Shelf Manager over a redundant I2C bus [2, 5]. Apart from using this route, the prototype CB by the Department of Microelectronics and Computer Science (DMCS) [3, 5] is also able to exchange data with an external PC station using the EIA RS-232 communication standard as its secondary interface (Fig. 1).

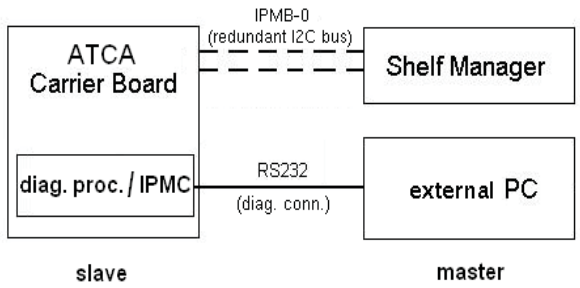

Figure 1. The IPMI and serial connection of the ATCA Carrier Board

The IPMC on the DMCS prototype CB is implemented on Atmel Atmega1281 [6] microcontroller cooperating with Xilinx Spartan 3 Field Programmable Gate Array (FPGA) [3, 7]. The microcontroller can operate in two modes. In its IPMC mode this serial connection is used only to transmit the most important status messages (e.g. Field Replaceable Unit (FRU) state changes after insertion). These messages do not provide system operators with sufficient information in case of failure during the development stage. Therefore, the microcontroller firmware can be replaced, changing its role from IPMC to a diagnostic processor. With the microcontroller reprogrammed this way the Shelf Manager connection turns inactive and the serial connection with a PC becomes fully employed, giving more detailed diagnostics and possibility to perform manual control over the CB. The CB can be then detached from the ATCA system and, separately from it, every CB device can be examined and manually configured. Such possibility offers substantial aid at the stage of preparing the system to run. With the serial connection it was possible to create a software tool allowing the external PC user controlling the CB and monitoring its status and activities. This system enables the user to easily view the overall state of the operating CB, read numerous of its parameters, contents of registers, values of sensors and to perform direct control over many of the CB aspects, like power management and devices settings. This constitutes a tool that substantially simplifies the process of diagnosing, controlling and debugging the CB operation. The system is composed of two major pieces of software. One operates on the ATCA Carrier Board microcontroller setting its role to diagnostic processor and the other is a PC-side application.

As the intention of the entire diagnostic system is to make the task of controlling and debugging the IPMC and the entire CB operation as easy and simple as possible, the PC-side software was designed and built in a form of application providing a Graphical User Interface (GUI), thus constituting a user-friendly front-end to the system.

The GUI application has been created with Java programming language due to convenient methods of creating graphical elements [8] that the language offers, suitable serial communication routines implemented [9] and, what is vital from the perspective of usability, the application operating system independence [8]. The Java Development Kit (JDK) used for the development process was in version 1.6.0 Update 11 (1.6.0_11).

II. Communication Principle of Diagnostic System

The communication between the GUI application running on external PC and the CB diagnostic processor is based on simple question-answer model. The former is the party that generates queries to the diagnostic processor and waits until a response comes. Therefore it can be described as master during the communication process (see Fig. 1). The diagnostic processor in turn, gathers the information needed from CB or carries out an appropriate command and sends the requested information or command execution confirmation back. It can hence be described as slave (Fig. 1). The slave module never transmits any data without prior request from master.

The architecture of protocol for communication between master and slave is of custom design. It allows to exchange substantial amount of information in a form of compact frames (table I), without vast transmission overhead.

TABLE I. Communication Frames Format – Request (left), Response (right)

Byte 1	Start signature
Byte 2	Command
Byte 3	Data byte 1
Byte 4	Data byte 2
Byte 5	Data byte 3
Byte 6	control sum

Byte 1	Start signature
Byte 2	Acknowledge
Byte 3	Data byte 1
Byte 4	Data byte 2
Byte 5	Data byte 3
Byte 6	control sum

Each frame consists of six bytes. The first one is always the same, it contains a fixed number (0x55) indicating the beginning of a frame. Each pack of six bytes received by either of the sides is tested for the first byte being equal 0x55 in order to ensure that both sides maintain synchronization.

There are two types of frames defined. The first one is a request frame (table I – left). It is generated by the master. The second byte of the frame contains two pieces of information. The first piece concerns the frame being either a query for some CB value (a 'read' frame) or an activity to be performed or a value to be stored on CB (a 'write' frame). This feature is coded in the least significant bit of this byte (0 for write, 1 for read). The other piece of information signifies particular command, indicating, which CB feature or device to examine or to control. Every such command is coded using the seven higher bits of this byte. The complete list and details of these commands (including mnemonics used within the application) are presented in table II. The subsequent three bytes (data bytes) of the frame can carry more precise request information, e.g. address of particular device register to read from or to write to, a value to store. The last byte contains control sum to ensure the validity of content of the rest of the frame.

TABLE II. Request Frame Commands Details

Command Code (7 bits)	Command Mnemonic	Command Description
"0011000"	AMC_DIODES	Read/set AMC diodes state
"0011001"	AMC_SWITCHES	Read AMC switches state
"0011010"	POWER_GOOD	Read management and payload power status for each AMC bay
"0011011"	POWER_FLAGS	Read power flags state for each AMC bay and the entire Board
"0011100"	AMC_PRESENCE	Read AMC presence for each AMC bay
"0011101"	AMC_POWERON	Set management and payload power status for each AMC bay
"0011110"	AMC_ENABLE	Set AMC visibility for each AMC bay
"0011111"	ADC	Read ADC values
"0100000"	FPGA_POWER	Read FPGA power status
"0100001"	I2C_ENABLE	Set AMC I2C connection status
"0100010"	EXTRA_SWITCH	Read Extra Switch state
"0100011"	TEMP_MAX_1	MAX6626#1: Read/write register
"0100100"	TEMP_MAX_2	MAX6626#2: Read/write register
"0100101"	TEMP_MAX_3	MAX6683: Read/write register
"0100111"	ATC210	ATC210: Read/write register
"0101000"	READ_GA	Read Carrier Board Hardware Address

TABLE III. Response Frame Completion Codes

Binary	Character	Mnemonic	Description
"00110000"	'0'	FR_OK_INVALID	Command Acknowledgement - ignore data bytes content
"00110001"	'1'	FR_OK_VALID	Command Acknowledgement - data bytes valid
"00110010"	'2'	FR_ERROR	Error

The other kind of frame is a response frame (table I – right). It is generated by the slave. The second byte contains information named Completion Code, indicating whether the requested operation succeeded or an error occurred. It also determines if the following three data bytes carry valid information (which is the case when e.g. a value of some register or reading from a sensor was requested) or can be disregarded (e.g. when the command was to enable or disable

some feature and only the completion information from the second frame byte matters). The Completion Code is a single piece of information occupying an entire byte, so in table III each Code is represented twofold: as a character and as a binary number, which may be treated as an ASCII code corresponding to that character. The last byte contains control sum similarly to the request frame.

III. Slave Controller

The CB microcontroller software can be configured to work in two modes. One is the standard IPMC mode, where the IPMI commands are processed and all the activities dealing with the IPMI standard are taken care of. The other one is the diagnostic mode. In this mode the IPMI processing is turned off altogether and the messages of interest are the serial messages coming from the PC GUI application.

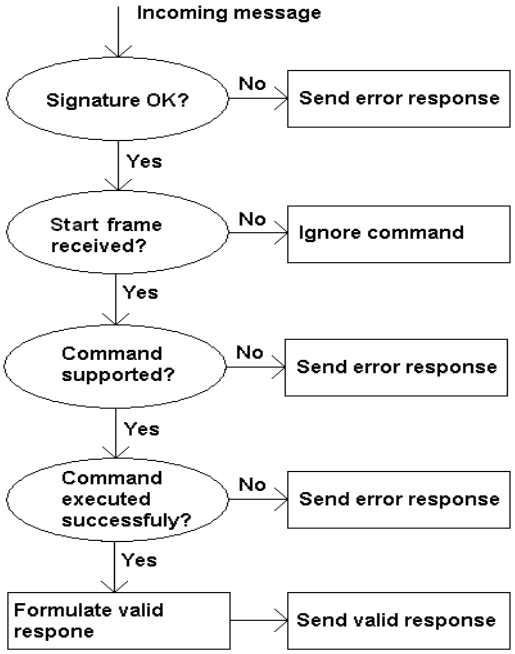

Figure 2. Frame processing algorithm of diagnostic processor

The communication between master and slave is initialized by exchanging a special block of data – a start frame. Slave will not pay attention to any incoming messages until the start frame is received. After this event all the bytes arriving on the EIA RS-232 interface are treated as frames of six and analyzed according to the protocol. If the frame starting signature is confirmed and the control value is verified to be correct the frame second byte (Command) is taken into consideration. The software compares its value against a set of predefined values representing all the supported functions. If the command is recognized an appropriate procedure that implements the given functionality is called.

The procedure may analyze the data bytes of the received frame if they are necessary to complete the action it is supposed to carry out. Then, the function interacts with the hardware gathering information about its state or changing it according to the command received. Upon completion a response frame is created. Depending on the original command the data bytes may be optionally filled with appropriate values e.g. raw sensor reading, state of LEDs. The signature byte is attached to the beginning of the message and the control sum is calculated and suffixed to the frame. In such a way a six byte array is constructed and forwarded to the functions dealing with EIA RS-232 communication where they are sent back to master.

The algorithm of the diagnostic processor software operation is illustrated in Fig. 2.

IV. Master Application Structure in Outline

The GUI application is divided into three modules as presented in Fig. 3. Two of them, Main GUI and GUI Components, are visible to the user and together they compose the high-level GUI layer of the application. The third module is responsible for communication.

The first module, Main GUI, is responsible for creation of the main application window, which is visible after the program startup. This module contains routines for determining settings of the connection with diagnostic processor, establishing and ending the connection and monitoring the connection status during the program operation. It also receives internal status messages and alerts from routines across itself and other modules and displays them in its Status Console in an easily readable manner. This enables user to control and monitor the connection together with its state and to supervise behavior of the entire application from one place.

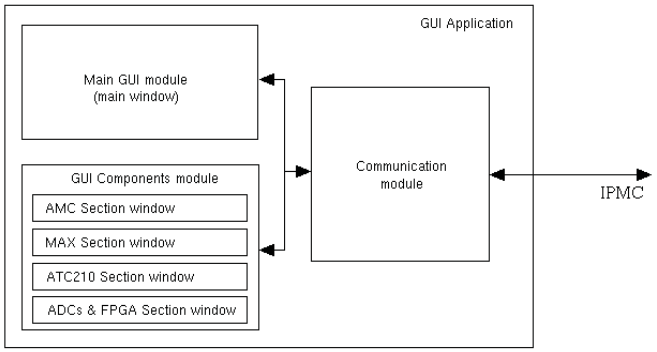

Figure 3. GUI application block diagram

The second module, GUI Components, contains all the application high-level routines for triggering the creation of requests that are later packed into communication frames as well as presenting information from the response frames in a clear and comprehensible way. The main window created by the Main GUI module contains block named Control Section, which after the connection has been established gives the user access to four further areas, which are responsible for displaying CB state information and switching its features. These four windows are created by the GUI Components.

The third program unit, Communication module, is separated from the GUI part and hidden from the user. It includes all the routines for handling serial port and performing low-level connection control and supervision. It

also collects request details from the GUI modules, packs them into frames, then transmits. And in the opposite way, after receiving data it performs frame validity check, then extracts and processes its content and passes the results obtained to the GUI modules.

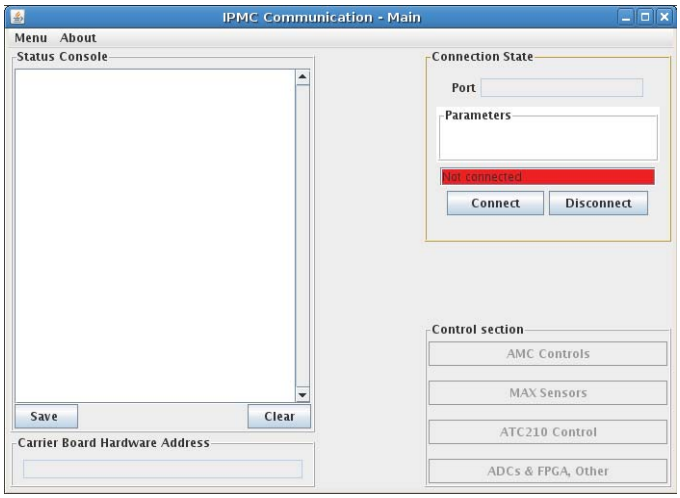

Figure 4. GUI example - main window

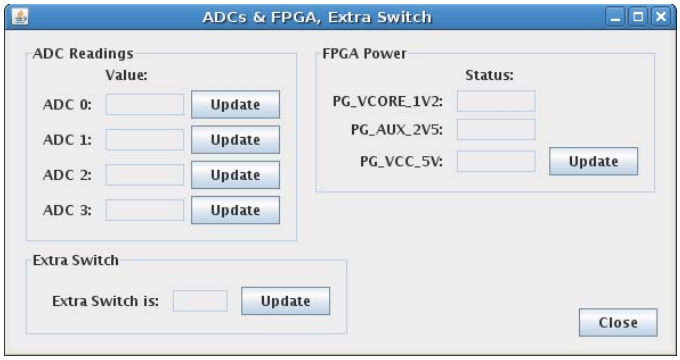

Figure 5. GUI example - ADCs and FPGA section

Fig. 4 and Fig. 5 present exemplary snapshots of the GUI, depicting general flavor of the system from the user's perspective. The first represents the program main window, and the other the area where the CB Analog-to-Digital Converter (ADC) values and FPGA power status is read.

A. The request frame transmission process overview

The Communication module itself is a hierarchical entity with several levels of communication routines. The top-most level of the module is an interface for communicating with the GUI parts. This interface contains routines, which trigger the process of request frame formation and transmission. Invocations of these routines are intended to be placed within appropriate GUI event processors, which are activated after a button on some GUI element has been pressed.

Each button on the GUI has a special item associated with it. This item, called button listener, is activated every time the button is pressed. Among the instructions executed by a button listener there is an invocation of an appropriate routine from the Communication module interface. Once a button has been pressed, this routine calls lower-level mechanisms that resolve the kind of request (i.e. whether it is a 'read' or 'write' one), then determine the precise command code (indicating, which CB device or feature this frame will refer to). These two pieces of information are then coded into a single byte, so called command byte. The button listener may also provide the high-level communication routine with some additional information (the case of some of 'write' frames, where it represent data to be transmitted, and some of 'read' frames, where e.g. a specified device register address in needed). This information is to be later coded into three data bytes for the frame. The lower-level mechanisms provide a generic frame from a blueprint with its first field already written with the frame signature. Then the command byte is placed at the second position, subsequently go the three data bytes and finally these five bytes undergo control sum calculation, which is placed as the sixth byte in the frame. With this process accomplished, routines dealing directly with serial port are called to transmit the formed frame.

B. The response frame reception and processing overview

When a request frame is sent, master waits for the response frame from slave. Routines for handling the serial port listen and expect six bytes to arrive. After receiving the sixth byte the frame content is processed.

Initially the frame validity check is performed. It goes in three steps:

- The frame signature is read. If it is invalid, that may mean that master and slave could have lost synchronization between each other and resetting the connection may be needed. In this case the rest of the frame is disregarded and an appropriate message is sent to the Status Console in the main application window.
- If the frame signature is correct, then all but the last frame byte undergo control sum calculation. The computed value is compared to the received control value from the last frame byte. If not equal, that may mean a transmission error has occurred and repeating the request is advised. The frame is disregarded and an appropriate message is sent to the Status Console in the main application window.
- Passing the control sum test enables the application to consider the received frame valid. The second frame byte, also referred to as Completion Code, is then examined. It can have one of three values (table III). The value equal binary "00110010" causes the frame to be ignored and an appropriate message to be sent to the Status Console in the main application window. Repeating the request is then advisable.

V. Diagnostic System Capabilities

The range of capabilities of the diagnostic system is strictly related to hardware installed on a particular CB. The equipment of the DMCS CB is discussed in [3] and [5]. The functionality of the GUI application dedicated to this CB is divided into four groups.

A. AMC related

This set gathers all the options concerning AMC modules placed in the CB AMC bays. These include:

- AMC presence - for retrieving from the slave information indicating whether AMC module presence is detected in any of the bays
- AMC power control - allowing to query for or to set the AMC modules power status. The management power and the payload power for each of the AMC bays are allowed to be treated independently from each other or both together, subject to needs. For testing purposes power can be supplied to the bays independently of the presence of AMC modules. Also AMC power flags state can be read. For each AMC bay these flags inform on whether the management power and the payload power after being turned on are properly supplied to the bays. At this point it is also possible to read status flags of each of the redundant voltages supplied to the entire CB. If any of the two voltages fail, these flags denote this fact.
- AMC enabling - when AMC module is present in bay and powered, the user has an additional possibility to toggle the module visibility in the system. Therefore, the user can decide on AMC module to be detectable for the diagnostic processor without changing its power status or physically moving the device.
- I2C enabling - allowing to enable or disable communication with the diagnostic processor via I2C bus for each of the available AMC modules individually
- AMC diodes control - allowing to read and set the state of a cluster of LED diodes present near the AMC bays. These diodes are installed on this CB for the diode operation test purposes only, but in further designs they are intended to be employed as AMC or CB state or signal indicators.
- AMC switches control – allowing to read the status, and therefore to test the operation of a group of switches present near the AMC bays. In further designs they are intended to be given the ability to trigger or alter certain CB aspects.

B. MAX sensors related

The DMCS CB is equipped with two Maxim MAX6626 temperature sensors [10] and one Maxim MAX6683 temperature/voltage monitor [11]. The diagnostic application enables reading any of the registers of these devices and storing data to any of their writable registers [10, 11]. Therefore these sensors can be conveniently configured and their readings immediately fetched.

C. ATC210 converter related

The prototype CB is also provided with Artesyn ATC210 Dual Input Bus Converter [12]. Similarly, as with the MAX sensors, the diagnostic application grants reading and, where it is possible, writing access to the device registers [12], thus simplifying configuration and readings fetching.

D. ADCs and FPGA related

This group relates to features chosen not to be assigned to any of the previous groups. The GUI window corresponding to this group is depicted in Fig. 5. The options include:

- ADC readings - for displaying values of voltage measured by four channels of ADC integrated in the Atmel microcontroller [6].
- FPGA power - for retrieving information on power status of the Xilinx Spartan FPGA [7]
- Extra Switch - for detecting and displaying the state of the Extra Switch, a small entity for testing the on-board switch functionality for the purpose of further applications

Apart from aforementioned functionality, the diagnostic application is also able to determine the Hardware Address of the examined CB. This is done at the stage of setting up the diagnostic connection and the address is displayed beneath the Status Console in the GUI main window (see Fig. 4).

VI. Conclusions

Currently the diagnostic application exists in a version dedicated to the prototype Carrier Board, as the program shape and functionality strictly depends on the collection of hardware installed on a particular CB. This is particularly important in the context of a fact, that the design of successive versions of the CB undergoes constant development and their hardware layer changes. The Atmel microcontroller and the Xilinx FPGA are planned to be abandoned in fulfilling the functionality of the IPMC and replaced with Renesas microcontroller. The EIA RS-232 communication standard is also intended to be replaced with faster and more reliable USB. These factors, together with successive devices being incorporated in CB design during the development process, have substantial impact on the diagnostic system shape. The design of the application, however, enables it to be easily customized for the use with future Carrier Boards.

Several acceptance tests proved the functionality of the diagnostic system. The use of the application during the prototype CB development process confirmed its usefulness, simplified debugging procedures and reduced the time costs of running the device.

There are plans to implement the functionality of both IPMC and diagnostic processor in a single software entity. The new microcontroller programmed this way would not need to be reprogrammed in order to switch from one role to another. Instead, there might be an additional toggle switch installed on CB, which would trigger the microcontroller functionality transition. This combined with customizable architecture of the presented application makes it a handy tool for diagnostics and debugging all the Carrier Boards intended to be used in this ATCA implementation.

Acknowledgment

The research leading to these results has received funding from the European Commission under the EuCARD FP7 Research Infrastructures grant agreement no. 227579 and Polish National Science Council Grant 642/N-TESLA-XFEL/09/2010/0. The authors are scholarship holders of project entitled "Innovative education ..." supported by European Social Fund.

References

[1] AdvancedTCA PICMG 3.0 Revision 2.0, “AdvancedTCA Base Specification”, 28 October 2005

[2] Intel Hewlett-Packard NEC DELL, “Intelligent Platform Management Specification version 1.5”, Document Revision 1.1, 20 February 2002

[3] Zawada, A., Makowski, D., Jezynski, T., Simrock, S., Napieralski, A., "ATCA Carrier Board with IPMI supervisory circuit", Mixed Design of Integrated Circuits and Systems, 2008. MIXDES 2008. 15th International Conference on 19-21 June 2008 Page(s):101 – 105

[4] Makowski, D., Koprek, W., Jezynski, T., Piotrowski, A., Jablonski, G. Jalmuzna, W., Simrock, S. "Interfaces and communication protocols in ATCA-based LLRF control systems", Nuclear Science Symposium Conference Record, 2008. NSS '08. IEEE, 19-25 Oct. 2008 Page(s):32–37

[5] Zawada, A.; Makowski, D.; Jezynski, T.; Simrock, S.; Napieralski, A. "Prototype AdvancedTCA Carrier Board with three AMC bays", Nuclear Science Symposium Conference Record, 2008. NSS '08. IEEE, 19-25 Oct. 2008 Page(s):53 – 57

[6] Atmel Corporation, “ATmega640/1280/1281/2560/2561”, Rev. 2549L, August 2007, http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf

[7] Xilinx Incorporation, “Spartan-3 FPGA Family Data Sheet”, June 2008, http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[8] B. Eckel, "Thinking in Java", 4th Edition, Prentice Hall, 2006

[9] I. Darwin, "Java Cookbook" O'Reilly, 2001

[10] Maxim Integrated Products, “MAX6625/MAX6626”, Rev. 4, October 2006, http://datasheets.maxim-ic.com/en/ds/MAX6625-MAX6626.pdf

[11] Maxim Integrated Products, “MAX6683”, Rev. 1, July 2004, http://datasheets.maxim-ic.com/en/ds/MAX6683.pdf

[12] Artesyn Technologies, “ATC210 Dual Input Bus Converter I2C Serial Bus Interface Application Note 206”, 16 August 2006, www.artesyn.com/assets/an_atc210_software_1199485474_techref.pdf