International Journal of Information and Education Technology, Vol. 1, No. 1, April 2011
ISSN: 2010-3689

Table Memory and Controller Implementation Scheme in FPGA for Hard Real Time Control Systems

Gillani Ghayoor Abbas, Yian Zhu, Amjad Hafiz Muhammad, Ahmad Waqar and Jianfeng An

Abstract—**High level of reliability is needed by the backplane bus for the Aircraft Information Management System (AIMS) that can ensure the robust and fault tolerant communication between its Line Replaceable Modules (LRM), which in turns ensures the safety critical hard real time control system operation. As the time driven protocol is more reliable than the event driven protocols, this backplane bus needs to be implemented with time division control protocol. For that matter, more attention is needed to ensure the synchronization issues between LRMs. This paper is the extension of our previous work and addresses Table memory and Controller implementation scheme for the Bus Interface Onit (BIU). The aforesaid system is developed for ARINC 659 backplane bus as an example, which controls all the BIU operations. The table memory introduces the cabin wide harmony by providing the same command sequence for all the LRMs. This command sequence also includes the source and destination addresses that avoids extra load on the backplane bus. The controller needs to fetch the commands from the table memory, decode them and then execute them to drive the bus for the BIU operations including message operations and synchronization handling. We have designed the Instruction Set Architecture (ISA) for table commands and implemented three Finite State Machines (FSM) for designing this controller along with some glue logic. First FSM is meant for managing commands, second for managing the BIU current state for synchronization needs, and the third for controlling the BIU operations. The aforesaid design has been modeled by using Verilog, Hardware Descriptive Language (HDL) and implemented in Altera Cyclone II board. Results of Modelsim and Quartus proved the cycle accurate implementation of controller in compliance with ARINC 659 specifications**.

Keywords-Backplane bus controller;Table Memory; FPGA; Hard Real Time control system; ARINC 659; AIMS

I. INTRODUCTION

Aircraft Information Management System (AIMS) gathers the information from the different sensors and process them for displaying and controlling different aspects of flight data. These include flight management, display, central maintenance, communication management, airplane condition monitoring and etc [1]. AIMS consists of two redundant cabinets implementing seven primary aircraft functions, each cabinet has four processing modules and four I/O modules that have to share same back plane bus [2]. The control systems for passenger aircrafts are considered to be safety critical hard real time systems. These systems need intra-cabinet communication system to share the information among the different Line Replaceable Units (LRU) within the cabinet. Time Triggered Protocol (TTP) is a preferred choice for such systems over the event triggered protocol for the aforesaid communication because the former has known and constant bus loading, latency and jitter, and also it supports composability [3]. As these control systems work on 25 Hz to 100 Hz, this backplane communication needs not to be very fast but the reliability must be high to ensure the safety of the aircraft. In this paper, we proposed a scheme to implement the backplane bus controller in the FPGA for such kind of safety critical hard real time control systems. We used ARINC 659 backplane bus protocol as an example and implemented the controller of its Bus Interface Unit (BIU). This protocol ensures the safety critical requirements as it has robust portioning in time and space, fault detection and tolerance; and the availability of redundant buses with cross-checking capability [4].

Moreover, the use of FPGAs has been proven for the experimental as well as industrial purposes especially for developing the real time systems in a single chip [5, 6]. We used Altera's NIOS development board, Cyclone II edition for our implementation platform [7]. This board has been used because it also has soft processor core NIOS, which has to be used in our related work of Host design for the BIU to complete our full system on chip design. It provides a hardware platform for developing embedded systems based on Altera Cyclone II devices with 50 MHz oscillator and Power-on reset circuitry.

This paper represents the extension of our previous work in [8] that includes Table Memory implementation scheme for the referred system. Section II discusses the architecture overview of ARINC 659 back plane bus; section III discusses the design and implementation of Table Memory and controller; section IV shows the test bench and the results of Modelsim and Quartus on our proposed Verilog model that proves the cycle accurate implementation of controller and the Table Memory in compliance with ARINC 659 specifications; and section V is the conclusion.

II. ARINC 659 ARCHITECTURE

The detailed architecture of ARINC 659 is explained in [4], some important aspects are highlighted in this section from the aforesaid reference to introduce the overall architecture who's Table Memory and Controller implementation will be discussed in the later sections.

ARINC 659 transfers half duplex serial data. The use of serial lines increases reliability by reducing the hardware. A

Manuscript received March 21, 2011.
School of Computer Science and Technology Northwestern, Polytechnical University,710072, Xi'an, China (email: itsghayoor@yahoo.com)

single BIU interfaces to two buses for availability. Through the use of cross-compared dual BIUs in each Line replaceable module (LRM), a total of four buses provide dual self-checking capability. Further cross-checking of the four paths increases data availability. A block diagram showing bus interface architecture and bus line composition and connection is shown in Fig. 1. Robust partitioning of time and space are provided by the Table Driven Proportional Access (TDPA) protocol. This is controlled by commands stored in the nonvolatile Table Memory attached to each BIU. The clock speed of the bus is 30 MHz. Data is transferred two bits at a time for a maximum throughput approaching 60 Megabits per second (Mbps). Bus time is divided into a series of windows, each window containing a single message from 32 to 8192 bits in length or a resync pulse (approximately five bit times). The Tables define the length of each window and which LRMs transmit, receive, or ignore the bus during the time assigned to the window. The bus transfer schedule is organized into cyclic loops, or frames, of constant length set by the sum of the individual window lengths. In particular, the immediate source and destination addresses for each message are contained in the Table Memories rather than being transferred across the bus. This saves the bus bandwidth normally consumed by address fields. Resync pulses are transmitted periodically to establish and maintain synchronization among all BIUs on the backplane. The bus supports module-to-module (point-to-point) transfers, a single module to a group of modules (broadcast) communication, and also alternative (candidate) modules to a group of modules. Accordingly, two types of messages exist: Basic and Master/Shadow (M/S) Messages. Frame Description Language (FDL) is included as a means to compare Tables from different vendors to ensure their compatibility.

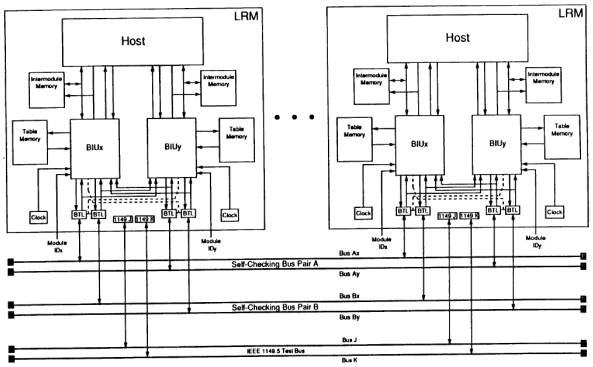

Figure 1. ARINC 659 Architecture showing LRMs and Backplane Bus [4]

III. Implementation Scheme of Table Memroy and Controller for ARINC 659 BIU

The Table Memroy and Control block of ARINC 659 BIU are shown as dark colored boxes in Fig. 2 [8]. Table memory is for storing the table commands just like ROM. BIU accesses 32 bit command words by providing specific address to the table memory. BIU control block can be regarded as an application specific instruction processor (ASIP) that can fetch the command words from the Table memory by generating appropriate address, decode them, and execute them to control all the synchronization and window operations of BIU.

Fig. 3 shows the internal design of control block [8]. Modular approach has been used to simplify the design; which includes Table Memory Interface Unit (TMIU), Command Decoder Unit (CDU), Command Management Unit (CMU), Command Buffer Unit (CBU), BIU State Management Unit (BSMU) and BIU Operations Control Unit (BOCU). Three separate finite state machines have been implemented to design the control block. This enables the partitioning of control block into three subdivisions. First state machine has been implemented in CMU. It is responsible for controlling TMIU to fetch the command words from table memory. It also enables the CDU to decode the received command word. Moreover, some FDL commands consist of two or three table memory command words, so it is also responsible to get the full information of these commands and convert them to comprehensive single access commands also called Comprehensive Command Information (CCI) in this paper. The CCIs are then stored in the CBU that can be accessed by the BOCU in a single system cycle.

Second state machine has been implemented in BSMU. It controls the BIU current state with respect to synchronization. Third state machine can be regarded as master controller of the control block that has been implemented in BOCU to control all the BIU operations. The detailed design scheme is discussed as under;

A. Table Memory

Table memory is a ROM used to store the table command sequence. The top level design is shown in Fig. 4. The command sequence is programmed by the user according to the backplane activity requirements. The BIU starts fetching commands from the table memory as it gets In-sync state by using its TMIU. The table memory has been designed for the following features; Single port ROM to be accessed by BIU, 8K words ROM, Word accessible 32-bit register output, 13 bit address field required to address 8K addresses of the respective words. As ARINC 659 supports the Entry Resynchronous and Frame Change operations, the indirect addressing is needed to store the real address of the jump—next frame address or the next address in a command sequence. As the code width is 8 bit wide so the lower 256 addresses are supported for the jump table. In this way, the next jump address is stored within first 256 addresses of the table memory. Therefore, the general command sequence is mapped starting from the 257th location of the ROM till the 8192th location.

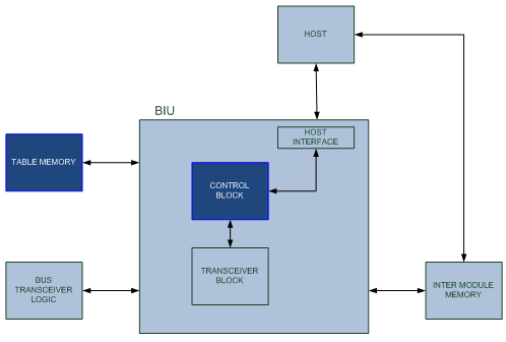

Figure 2. A portion of LRM showing BIU, Table memory, IMM, BTL and Host

B. ISA Design

ISA has been designed to translate the FDL commands into the binary format that can be stored in the table memory. All the commands necessary for the possible BIU operations have been included as discussed in [4]. Fig. 5 shows the 32 bit instruction fields for every command. Some commands like BOW, VER, ERU, ERV, FCU and FCV take more than one word to store the required information.

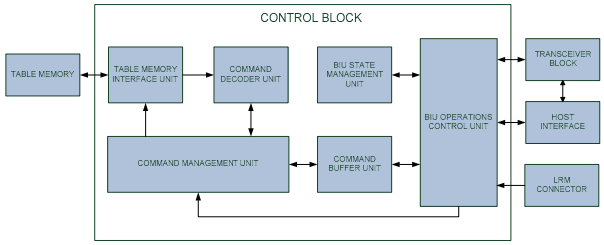

Figure 3. Internal Design of Control Block

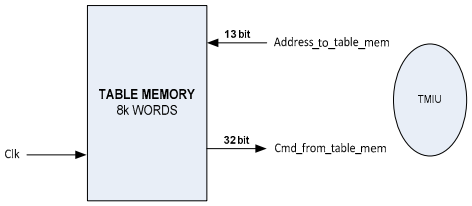

Figure 4. Table Memory Top Level Design

C. Command Management Scheme

TMIU, CDU, CBU and CMU in Fig. 3 show the command management scheme of our controller design. The aforesaid design units are responsible for fetching command words from the table memory, decoding them, generating and storing CCI in the CBU. Functionality of TMIU may be regarded as Address Generation Unit (AGU). It generates 16-bit address for the table memory to map 64K words.

Sr. no.	Command	No. of words for instruction	Op-code (5 bits) [4:0] of 1st word	Other fields	Remarks
1	BOW	3	00000	Window size (8 bits [12:5]); data_item (1 bit [13]); IMM Address[17:0] (18 bits [31:14])	Word #1
				Tx_Master (5 bits [4:0]); Tx_Shadow1 (5 bits [9:5]); Tx_Shadow2 (5 bits [14:10]); Tx_Shadow3 (5 bits [19:15]); valid (3 bits [22:20]); IMM Address[26:18] (7 bits [31:23])	Word #2
				Rx Mask (32 bits [31:0])	Word #3
2	GAP	1	00001	Gap size (4 bits [8:5]); don't care (23 bits)	
3	DELTA	1	00010	Delta size (4 bits [8:5]); don't care (23 bits)	
4	FREE	1	00011	Free bit time value (27 bits [31:5])	
5	RET	1	00101	don't care (27 bits [31:5])	
6	RETI	1	00110	don't care (27 bits [31:5])	
7	SSYNC	1	00111	don't care (27 bits [31:5])	
8	SUB	1	01000	don't care (27 bits [31:5])	
9	CALL	1	01010	Address (27 bits [31:5])	Supporting 2 Mb ROM
10	CALLI	1	01011	Address (27 bits [31:5])	
11	JUMP	1	01100	Address (27 bits [31:5])	
12	JUMPI	1	01101	Address (27 bits [31:5])	
13	ERU, ERV, FCU, FCV	2	ERU: 01110 ERV: 01111 FCU: 10000 FCV: 10001	Tx_Master (5 bits [9:5]); Tx_Shadow1 (5 bits [14:10]); Tx_Shadow2 (5 bits [19:15]); Tx_Shadow3 (5 bits [24:20]); valid (3 bits [27:25]); don't care (4 bits [31:28])	Word #1
				Code (8 bits [7:0]); don't care (24 bits [31:8])	Word #2
14	VER	2	10010	Version minor value (8 bits [12:5]); cabinet# (4 bits [16:13]); don't care (15 bits [31:17]).	Word #1
				Version value (32 bit [31:0])	Word #2

Figure 5. ISA Design for ARINC 659 FDL Commands

Address generation logic include the incremental address for the commands like BOW, GAP, DELTA, SSYNC etc; it also includes the jump and return for supporting JUMP, CALL and RETURN commands—a level 8 stack has been implemented for storing return addresses. Moreover, it includes the indirect addressing for supporting Frame Change (FC) and Entry Resynchronous (ER) commands. The CDU decodes the command words that are fed by the TMIU. The decoded information is given to CMU for further management of commands. The decoded information include the command type, gap value, delta value, version value, BIU behavior in the current BOW command that may be transmit or receive or skip, basic or M/S and etc. One implementation problem is to keep record of previous command words to decode the multi-word commands. For that matter, the specific command counters and flags have been used that are set when the first word of command is encountered; and the command counters are incremented on every next command word received. When the counter value equals to the number of command words for the specific command, the decoded information is completed and the flags are cleared. The CBU is a FIFO designed to the depth of 8 49-bit levels. The CMU stores the 49-bit comprehensive command information into the CBU that can be accessed in a single cycle by the BOCU. Therefore, we implemented the CBU to cater dual access problems. Full and Empty signals have been provided to let the writing and reading units know about the status of the FIFO. Moreover, simultaneous read and write has also been resolved in the HDL design. A state machine has been implemented in the CMU that is responsible for managing commands by controlling different operations during its different states; Get next command word state: enables TMIU to get the next command word and also provides it information to generate the address of next command word, Decode command word state: enables CDU to decode the current command word, Execute command word state: generates the comprehensive command information (CCI), Push command words state: writes the CCI into the CBU when it is not Full.

The need of this command management scheme arise from the stringent requirement of executing transmit command (may include BOW, ERU, ERV, FCU and FCV) within two bus cycles of the last window—minimum Gap time [4]. The aforesaid requirement needs the next command to be fetched and decoded in one bus cycle and executed in the second cycle to enable the transmission within the minimum Gap time. As shown in Fig. 4, some FDL commands that exhibit transmission on the bus are two (ER/FC) and some are three (BOW) words in width. To fetch a command consisting of three command words, and decode those three words needs six system clock cycles. Therefore, the severity of this problem depends upon the comparative speed of system clock and the bus clock.

To overcome the abovementioned problem, three schemes can be analyzed. First is to support comparatively high system clock with respect to bus clock. Secondly, ISA can be designed to support encoding of a FDL command into a single access register. Thirdly, the command words can be fetched and decoded in advance to the execution time, and saved as a single cycle access entity, then BOCU may get the comprehensive command in one bus cycle and execute that in the next to comply with the requirement. We adopted the third scheme because it has no hazards of additive

complexity or synthesis/implementation problems as introduced by the first two schemes.

D. BIU Synchronization Management Scheme

BIU synchronization management unit (BSMU) is shown in Fig. 3. A state machine has been implemented in this unit to control the current state of BIU and its transitions. Fig. 6 shows the BIU state transition requirements as described in [4] that have been implemented in this unit. This state machine requires four states: Initializing, out of sync, in sync and disconnected. The states are changed on receiving different status signals from the BOCU. These signals include sync loss, detection of initial sync pulse or long resync pulse, bad cabinet position and version mismatch. All these signals are generated during the different bus operations that include entry resynchronization and frame change processes and the sync pulses receive operations. Hence, this unit decides the current BIU state and assists the BOCU unit to control BIU operations as they depend on it.

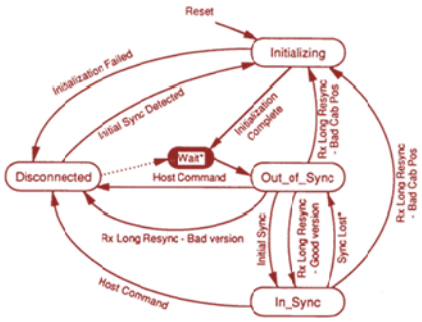

Figure 6. BIU States and their transitions [4]

E. BIU Operations Control Scheme

This is the core module of BIU controller design that is responsible for all the bus activity and controlling other modules within BIU for their desired operations shown in Fig. 3 as BIU operations control unit (BOCU). It holds the special function registers needed by BIU, executes the table memory commands, manages the synchronization pulses operations and provides services to the Host along with controlling the smaller modules within control block. It also communicates outside the control block with transceiver block and host interface. Following is the brief description of BOCU functionality;

Special Function Registers (SFRs) include Full Resolution Timer Register (FRTR) that is a 43 bit wide counter used to count every bit time of the backplane bus; two 32-bit Version Registers contain table major version, table minor version and cabinet position used to ensure cabinet-wide operational consistency; two 4-bit registers for GAP size and Delta size are used to store the current Gap value and Delta value supported in the command table; 20-bit Wait Limit register used as a reference to wait time for the long resync pulse detection before generating initial sync message. The details of these registers can be seen from [4].

A state machine has been implemented in the BOCU that have the states named Get command state: to fetch the next CCI from the CBU, Execute command state: to execute the command operations, Wait for initial pulse state: for waiting the long resync pulse before generating the initial pulse, Initial pulse operation state: for generating the initial pulse when the Wait limit is over, Out of sync operations state: this includes waiting for the long resync message and executing its receive operations to get the jump address, and the synchronization operation.

When the BOCU encounters the constant value commands like GAP, DELTA and VER it simply stores their values in their respective SFRs. The BOW commands are executed depending upon the message type—Transmission_info. It may be Master transmit, shadow1 transmit, shadow2 transmit, shadow3 transmit, M/S skip, Basic transmit, Basic receive and Basic skip. When BOCU encounters the BOW command, it transmits/receives or skips the required number of words as shown in Fig. 7.

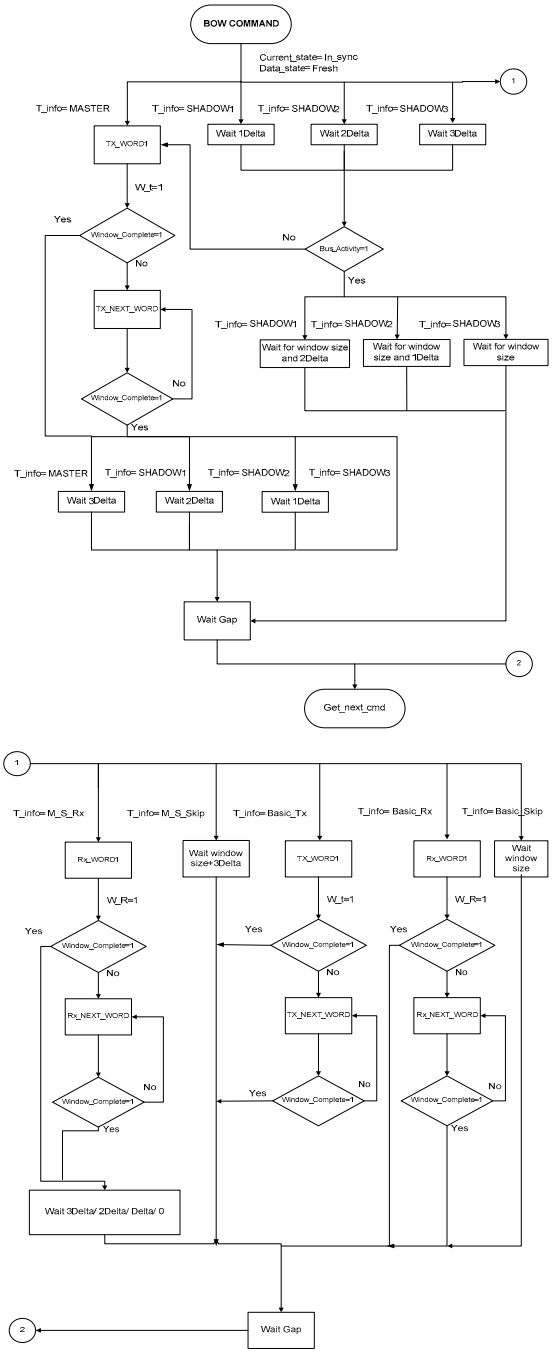

Figure 7. Flow Diagram of BOW Command Operations

When the BOCU encounters the SSYNC command it enables the transceiver block shown in Fig. 2 to generate the short sync pulse. Entry resync and Frame change commands (ERU/ERV/FCU and FCV) operations have also been implemented in this unit. These commands need the BIU operations to receive or transmit the three word message

along with the long resync pulse. The flow diagrams of these commands can be seen in [4]. JUMP, CALL and RETURN commands are managed by CMU to avoid any delay in the back plane activity. FREE, JUMPI, CALLI and RETI commands just need a counter to wait for the number of bus cycles as described in the FREE command and implicit idle respectively before proceeding to next command execution. Moreover, the receive operations of the long resync and initial sync messages for BIU not In sync state have also been implemented in the BOCU. This enables a Disconnected or Out of sync BIU to re-enter the In sync state.

IV. Experimental Results

Modelsim and Quartus have been used for the verification of controller implementation. Fig. 8 shows the command program test bench and its hex code translation—to be saved into the table memory—for verification purposes. This command program includes five windows that generate basic and M/S messages. Also, FC and ER commands are included to check the frame change and entry resync operations. Two LRMs and backplane bus were modeled in

Verilog HDL. Each LRM contained two BIUs. Transceiver block was also modeled to check the functionality of controller—whole system design explanation is beyond the scope of this paper. Fig. 9 shows the bus activity (Modelsim simulation). From left to right is initial sync pulse, long resync pulse, SSYNC, 4 message windows separated by SSYNC, ERU, ERV, 5th window, FCV and then again jump to the start of program. Fig. 10 shows the RTL view of synthesized finite state machines; Fig. 11 shows the Table Memory implementation by using Mega Wizard Plug-In Manager of Quartus; and Fig. 12 shows the synthesized Table Memory and Controller implementation as generated by Quartus. We used 60 MHz as a system clock to verify the controller design as previously used in [8].

V. Conclusion

This work shows the successful implementation of backplane bus Controller and Table Memory for ARINC 659 specifications. This implementation technique addresses the real time problems of fetching and decoding commands, managing synchronization mechanisms, and controlling the bus operations. It also addresses the design of instruction set for translating commands into binary format that can be stored in Table Memory. The Controller and Table Memory have been modeled in Verilog HDL and tested by using CAD tools. These CAD tools include Quartus and ModelSim. The results show the cycle accurate implementation of the aforesaid design in compliance with ARINC 659 specifications. Same technique can be used for the implementation of other reliable backplane bus protocols for hard real time control systems in FPGA.

Acknowledgment

The authors acknowledge Chinese Govt. for their Scholarship award of Masters Degree in such a prestigious university of china. Moreover, this research is supported by NSFC grants (NO. 61003037) and (NO. 60736012). Finally, special thanks to Mr. Zhang Sheng Bing, Mr. Wang Deli, Mr. Najam and Mr. Aamir for their guidance to complete this work.

Address	32-bit HEX Value	Comments
0	0000 0104	COLD address is stored in code 0
1	0000 011F	Code 1 contains the next command address after 285: ERU 1, 1 0 3 4
2	0000 0121	Code 2 contains the next command address after 287: ERV 2, 2 0 3 4
3	0000 0122	Code 3 contains the next frame address (WARM)
4 to 255	FFFF FFFF	Unused code addresses;
256	0000 0101	GAP 8
257	0000 0102	DELTA 8
258	0000 40F2	VER; WORD1
259	0000 111A	VER; WORD2
260	0000 0007	COLD: SSYNC
261	0000 0080	BOW, Window1; WORD1
262	0000 0001	BOW, Window1; WORD2
263	0000 0004	BOW, Window1; WORD3
264	0000 0007	SSYNC
265	0000 0080	BOW, Window2; WORD1
266	0072 0C01	BOW, Window2; WORD2
267	0000 0004	BOW, Window2; WORD3
268	0000 0007	SSYNC
269	0000 0080	BOW, Window3; WORD1
270	0072 0C20	BOW, Window3; WORD2
271	0000 0004	BOW, Window3; WORD3
272	0000 0007	SSYNC
273	0000 0080	BOW, Window4; WORD1
274	0072 0460	BOW, Window4; WORD2
275	0000 0004	BOW, Window4; WORD3
276	0000 0007	SSYNC
277	0000 23AA	CALL HOME
278	0000 0080	BOW, Window5; WORD1
279	0070 9060	BOW, Window5; WORD2
280	0000 0004	BOW, Window5; WORD3
281	0000 0007	SSYNC
282	0000 0050	FCV 3 WARM 2; WORD1
283	0000 0003	FCV 3 WARM 2; WORD2
284	0000 208C	JUMP COLD
285	0E41 802E	HOME ERU 1, 1 0 3 4; WORD1
286	0000 0001	ERU 1, 1 0 3 4; WORD2
287	0E41 804F	ERV 2, 2 0 3 4; WORD1
288	0000 0002	ERV 2, 2 0 3 4; WORD2
289	0000 0005	RET
290	0000 0007	WARM SSYNC
291	0000 208C	JUMP COLD
292	0000 0009	END
293 to 65535	FFFFFFFF	Unused Table Memory

Figure 8. Command Program Test Bench & its Hex code for Controller Verification

References

[1] Michael J. Morgan, “29. Boeing B-777”. The Avionics Handbook, Ed. Cary R. Spitzer. Boca Ratonby by CRC Press LLC 2001.

[2] T. Carpenter, K. Driscoll, Hoyme and J. Carciofini “ARINC 659 scheduling: Problem definition” Real-Time Systems Symposium, 1994 Proceedings, 1052-8725/94, IEEE 1994.

[3] D.A Gwaltney and J.M. Briscoe “Comparison of communication architectures for spacecraft modular avionics systems” NASA/TM—2006 -214431 available at http://naca.larc.nasa.gov

[4] ARINC Specification 659 “Backplane Data bus” Aeronautical Radio, Inc. Annapolis Maryland, December 27, 1993.

[5] Zeyad Assi Obaid, Nasri Sulaiman and M. N. Hamidon “FPGA-based Implementation of Digital Logic Design using Altera DE2 Board” IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.8, July 2009.

[6] A.R.M. Khan, A.P. Thakare, S.M. Gulhane “FPGA-based design of controller for sound fetching from codec using Altera DE2 Board” IJSER International journal of scientific and engineering research, volume 1, issue 2, novembe-2010.

[7] NIOS Development Board, Cyclone II edition reference manual, available at http://www.altera.com

[8] Gillani Ghayoor Abbas, Yian Zhu, Amjad Hafiz Muhammad, Ahmad Waqar, Jianfeng An ”Backplane Bus Controller Implementation in FPGA for Hard Real Time Control Systems” 2011 IEEE International Conference on Information and Education Technology (ICIET)—January, 2011.

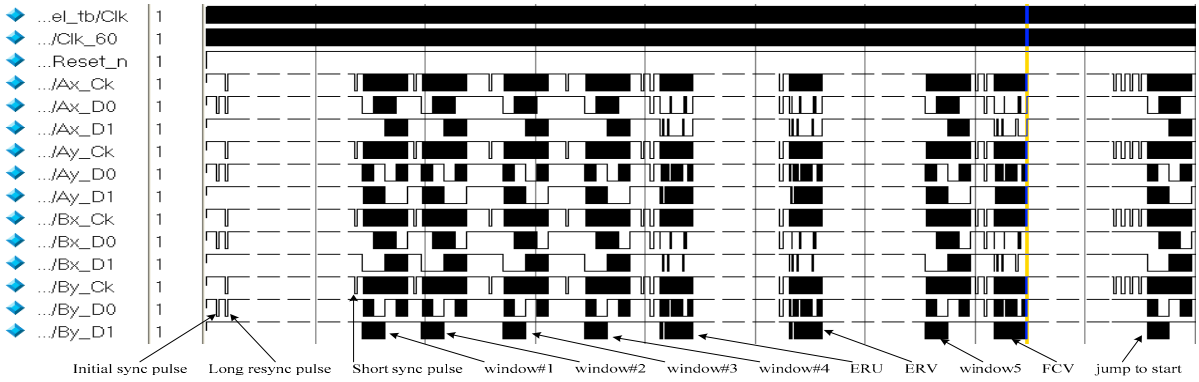

Figure 9. Backplane Activity Validating the Controller Design Scheme (Modelsim waveform)

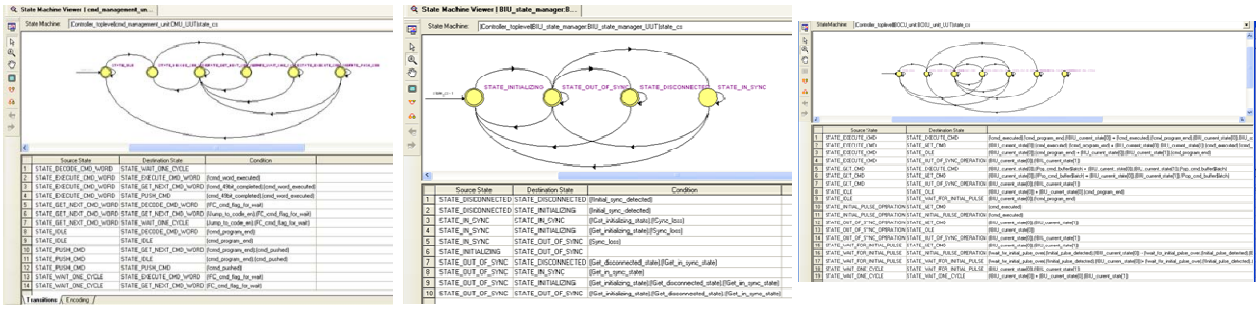

Figure 10. Command Management (Left), BIU State Management (Middle) and BIU Operations Control (Right) FSMs Implementation in Quartus

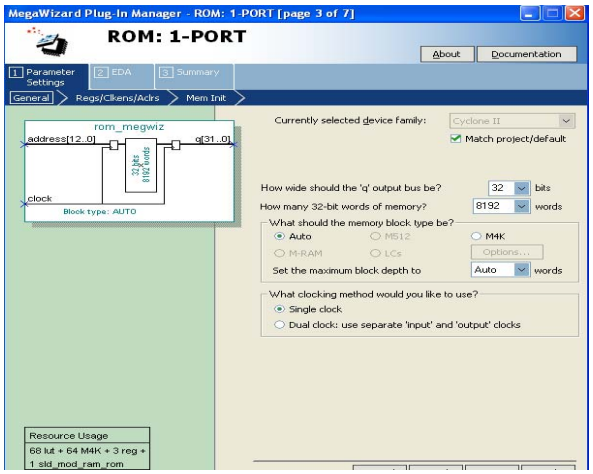

Figure 11. Table Memory Implementation in Quartus

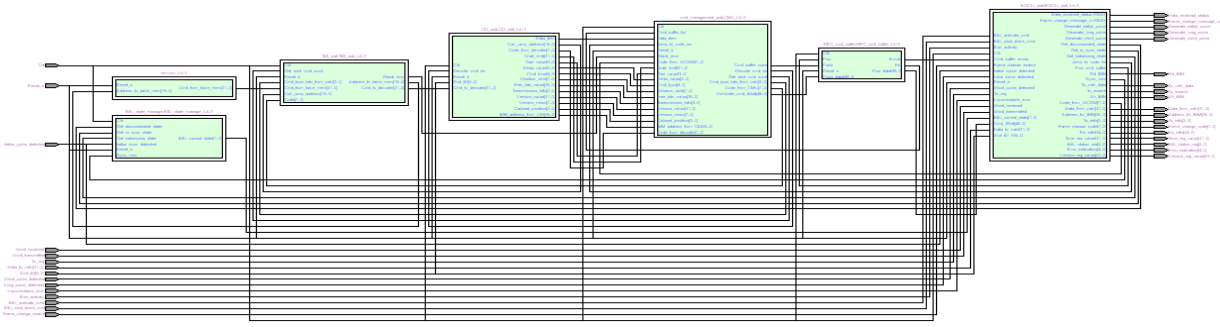

Figure 12. Synthesized Controller Design (RTL view, Quartus)