FPGA Implementation of Quasi-Delay Insensitive Microprocessor

Sufian Sudeng and Arthit Thongtak

***Abstract*- This paper proposes the self-timed circuits FPGA based design. Quasi-delay insensitive circuit is introduced as asynchronous prototype. The designed focuses on asynchronous processor design. The processor employs asynchronous system bus, asynchronous cache, asynchronous DMA controller and synchronous interfaces. The system almost completely asynchrony operation except I/O devices and memory interfaces, it's a limitation on the present time devices, the designed has been implemented on spartan-3E FPGA no. 3S500EFG320 by partitioning each module to prevent place and routing conflict, 100-Mhz memory frequency connected, and consumed 141,063 equivalent gate counts. Finally, the timing details of each instruction execution are shown.**

Keywords: Asynchronous processor, Asynchronous system bus, Asynchronous DMA controller, QDI-delay model, Dual-rail encoded, Structural encoding.

I. Introduction

The rapid developments of micro electronics technology, cause the designed circuits are small and high-speed communication. The old design methods are synchronous design. It cannot serve the ability to correct an operation on the high frequency clock. Clock skew is introduced which is cannot distribute high frequency signals to the system. Synchronous circuits also generate worse case delay propagation of global clock to control each part of circuit, it makes much of waiting time to process all parts of circuits, and more energy consumption. Asynchronous design method is restudied. The asynchronous design does not need a global clock to control the combinational circuits to behave on each other. It's depending on event-driven behavior, speed of circuits do not reduced by clock; furthermore, clock skew is avoided. More outstanding potentials of asynchronous circuits over synchronous circuits are average case delay, less energy consumed, robustness and etc [1].

Manuscript received July 22, 2007; revised August 10, 2007. This work was supported in part by TJTTP-OECF (Thailand - Japan Technology Transfer Project – Japanese Overseas Economic Cooperation Fund)

S. Sudeng is a researcher from Digital System Engineering Laboratory (DSEL), Department of computer engineering, faculty of engineering, Chulalongkorn University, Thailand (phone: +66813818331; email: sovanyy@msn.com)

A. Athongtak is now with the department of Computer engineering, Chulalongkorn University, Thailand (Tel: +66-2-218-6956, Fax: +66-2-218-6955 email: Arthit@cp.eng.chula.ac.th)

Asynchronous circuits do not use a global clock, circuits operation response to signal transition, output can be sensed as an operation is completed, it's also harder design and verify, no tools supported, made some elements are limited to clock operation. To enhance the design, need to compromise such a hybrid system, both synchronous and asynchronous are working together, Finding synchronization method to synchronize to be well form, and good circuit operation between synchronous and asynchronous one [2].

This work proposes an implementation of asynchronous quasi-delay insensitive circuits on FPGA, and also introduced the interfacing method between synchronous and asynchronous circuits.

II. Asynchronous circuit design

To understanding asynchronous design is familiarity with the assumptions commonly made regarding the delays in the gates and wires within a circuit and the mode in which the circuit operates. The two common delay models, bounded delay model and unbounded-delay model. The bounded delay model was commonly use in asynchronous circuits design, and still used in some backplane level interconnection scheme, current VLSI designs and research efforts use the unbounded delay model for the implementation of state – machines and controllers since it leads to circuits that will always operate correctly whatever the distribution of delay. It separates delay management for the correctness issue allowing the functionality of the circuits to be more easily verified. The bounded-delay model still commonly used for data path components, however, since in this area it can lead to smaller implementations

A. *Circuit classification*

Within the unbounded delay model, there is various different design styles in common use, each with its own problems. In order to increasing number of timing assumptions

<u>Speed – independent (SI) circuits</u>

If wire delay in a circuits are assumed to be zero and the circuits exhibits correct operation regardless of the delay in any circuit elements, then the circuits is said to be speed-independent. The assumption of zero wire delay is valid for small circuits.

<u>Delay-insensitive (DI) circuits</u>

A circuit whose operation is independent of the delays in both circuits' elements (gates) and wires is to be delay insensitive.

<u>*Quasi delay-insensitive (QDI) circuits*</u>

If the difference between signal propagation delays in the branches of a set of interconnecting wires is negligible compared to the delays of the gates connected these branches when the wires are said to from an *Isochronic fork*. Circuits created using the DI design style augmented with the *Isochronic fork* assumption are said to be Quasi-delay assumption as shown in figure 1.

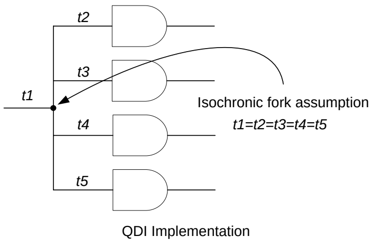

Figure 1: Quasi-delay Insensitive implementation

B. Signaling protocol

The transfer of information across a channel is negotiated between the sender and receiver using a signaling protocol. Every transfer features a request action where the initiator starts a transfer, and an acknowledge action allowing the target to respond. These may occur on dedicated signaling wires, or may be implicit in the data-encoding used, but in either case, one event indicates data validity, and the other signals its acceptance and the readiness of the receiver to accept further data.

The flow of Information relative to the request event determines whether the channel is classified as a push channel (information flows in the same direction as the request) or pull channel (information flows in the same direction as the acknowledge)

<u>*4- Phase signaling*</u>

The 4-phase signaling protocol uses the level of the signaling wires to indicate the validity of data and its acceptance by the receiver. When this signaling scheme is used to pass the request and acknowledge timing information on a channel, a return-to-zero phase is necessary so that the channel signaling system ends up in the same state after a transfer as it was in before transfer. This scheme thus uses twice as many signaling edges per transfer than its 2-phase.

4-phase control circuits are often simpler than those of the equivalent 2-phase system because the signaling lines can be used to drive level – controlled latches and the like directly as shown in figure 2.

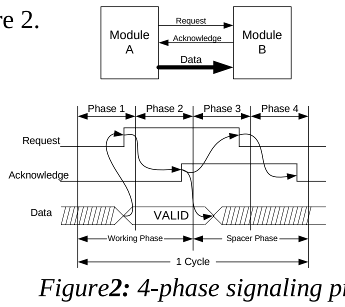

Figure2: 4-phase signaling protocol

C. Dual-rail encoding

Dual-rail circuits use two wires to represent each bit of information. Each transfer will involve activity on only one of the two wires for each bit, and dual-rail circuits has uses 2n signals to represent n bits of information. Timing information is also implicit in the code, in that it is possible to determine when the entire data word is valid by detecting a level for 4-phase signaling on one of the two rails for every bit in the word.

4-phase dual-rail coding is popular for QDI design style but as with dual rail techniques its carries a significant area overhead in both excess wiring and the large fan-in network that it require to detect an event of pair of wires to determine when the word is complete and the next stage of processing can begin.

D. The Muller C-element

The Muller c-element is commonly used in asynchronous VLSI design where it is used both for synchronizing events and as state holding element. Its represent outputs to 0 when all inputs are 0 and represent outputs 1 when all inputs are 1, otherwise it hold previous state.

F .Asynchronous Synthesis from STG

Signal transition graph (STG) is a graph-based method used to describe the circuit's behavior. It composes of three types of signals. Input, output and internal signal Inputs are known as underline or bold as the input specification with the transition labeled with signal names. In the STG notation a transition is labeled with either a '+' (represents a rising signal), '-' (represent a falling signal).for any STG it's a single cycle if only one rising and falling transition, otherwise multi cycle. Any STG there must follow on its rules to provide well form and healthy circuits design

STG Rules

(a) - Input free – choice: only Mutex inputs may control the choice

(b) - 1- Bounded – Maximum 1 token per phase

(c) - Liveness – No deadlocks occurred

STG for Speed independent circuits:

(a) - Consistent state assignment – signal strictly alternate between + and –

(b) - Persistency – Excited signal fire, namely they cannot disable by other transition

For synthesizable STG

(a) - Complete state coding – Different marking must represent different states

III. Asynchronous Processor Design

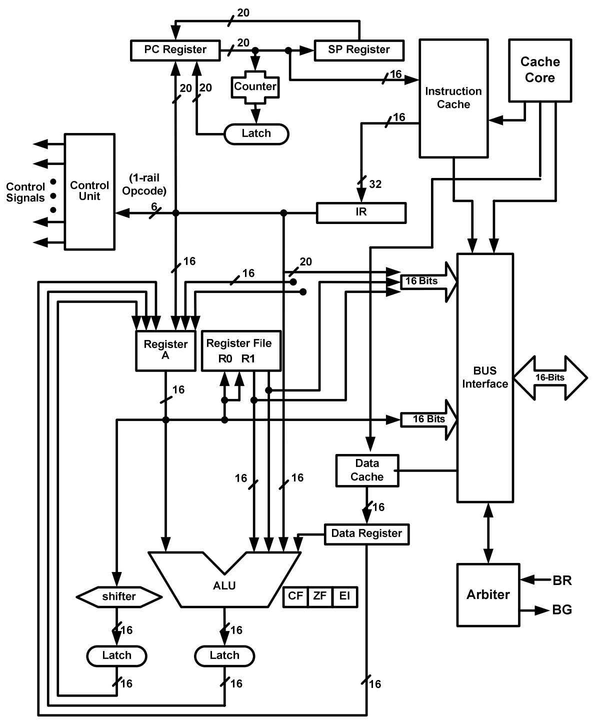

Figure 3: Processor Organization

The processor architecture[5,7] composes of 8-bit accumulator register A, general purpose register R1 and R2, program counter PC, SP register for CALL instruction , IR register for instruction loading and 2 status flags, carry and zero flag as shown in figure 3.

The processing section composes of shifter, uses to shift register A left or right direction, accumulate based computing (Register A and operand processing, store result to A). The asynchronous processor addressing mode divided into 4 modes, Immediate Mode, Register Mode, Direct Memory Addressing Mode and Indirect Memory Addressing with Register Indexing Mode.

because of accumulate based processing that write the completion result to register A. if A value has changed, it causes to produce a new calculation, need some latching circuit to prevent double writing problem.

The processor has 16-bit instruction word with 16K bits location accessing, 1-k location for 8-bit data memory. For dual-rail encoding, need additional circuits to convert the circuit from 1-to-2 signals and from 2-to-1 signal.

The internal architecture composes of 2 processing logic circuits, arithmetic logic unit and shifting circuit. The main task of arithmetic logic unit is Add, subtract, or, xor. After finished an operation, it passes the result through the latching circuit. The shifting circuit is the circuit that shifts register A left or right direction 1-bit precisely.

Control unit processes following 4-phase signal protocol which returns to zero on each computation by using auto sweep module as shown in figure 4.

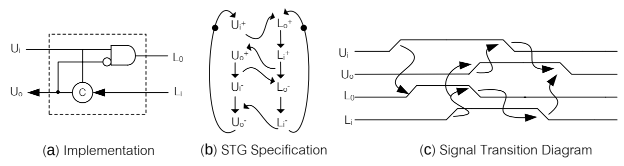

Figure 4: Auto Sweep Module

From figure 4, when previous request signal (U_i) changes from 0 to 1, ASM changes current request signal (L_o) from 0 to 1 to enter working phase, after current work has done, the acknowledge signal (L_i) is asserted back to ASM module, provided previous acknowledge signal (U_o) changes from 0 to 1 immediately with current request signal changes from 1 to 0 for idle phase, when working phase has done, current request signal is changes from 1 to 0 automatically, hence ASM was used to design control circuit while it can parallel flip working on previous idle phase as shown in figure 4 and 5.

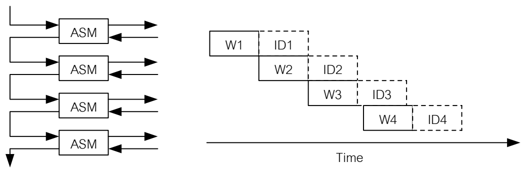

Figure 5: Auto Sweep on action

The instruction unit has 16-bit wide length, separated to *opcode* and *operand* as shown in figure 6.

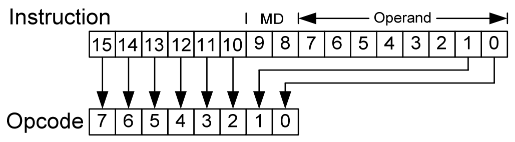

Figure 6: 16-bit instruction design

Where MD is memory addressing mode

00 → Immediate modes, 01 → register mode, 10 → direct addressing, 11 → indirect addressing

The speed of memory is very low in comparison with the speed of processor. The processor cannot spend more waiting time to access an instruction and data in memory. Cache is a fast memory will use on our design. The correspondence of cache and memory specified by mapping function, the design is apply simplest way to determine cache location which store in memory call direct mapping technique. All of asynchronous cache design is shown in Figure 7.

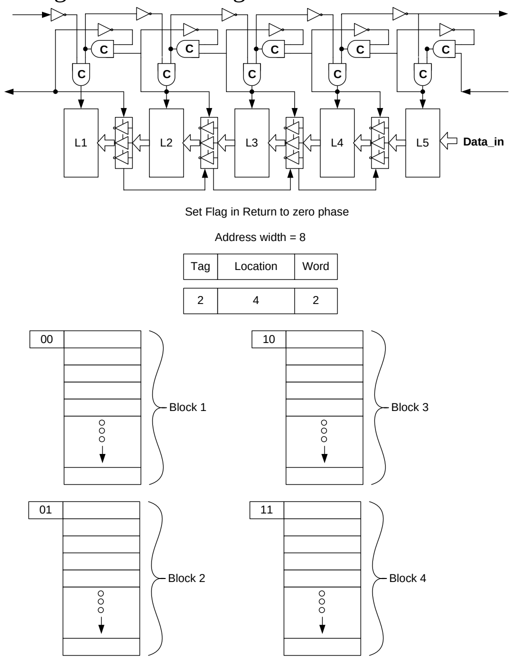

Figure 7: Asynchronous Cache Design

The processor instruction that used in this system has 16-bit instruction set; the instruction is loaded from cache to IR register. When 16-bit instruction has loaded into IR register, IR register responses to separate instruction in to two types, *opcode* and *operand*, the *opcode* is loaded to control unit section while operand is loaded to general purpose register or accumulator base on instruction type [7]. The modification benefit to generate a ***virtual instruction*** that occurred during interrupts even.

When interrupt instruction occurred, a virtual instruction has generated by external signal, IR register is loaded, separate to *opcode* and operand that point to interrupt service routine.

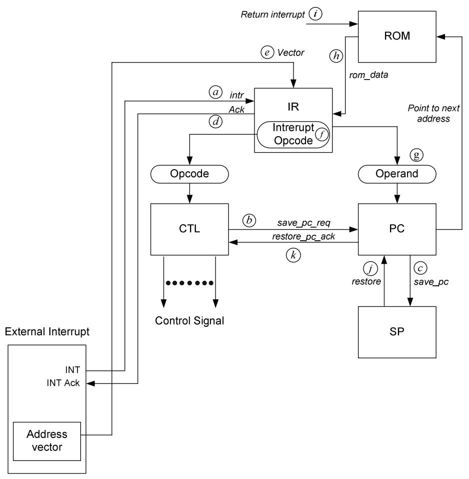

Figure 8: Interrupt design

Transaction sequence of an interrupt operation is shown respectively.

a. Interrupt signal from external device is asserted.
b. Control unit tell pc register to save current pointer.
c. Save PC register to SP register.
d. Interrupt acknowledges signal is asserted.
e. Address vector is loaded to IR register.
f. Interrupt *opcode* is generated.
g. Load new PC value.
h. PC point to interrupt service routine, load data to IR register.
i. Instruction meets RETI command (return interrupt).
j. RETI command tells SP register to restore value to PC register.
k. PC restore acknowledge to control unit.
l. Processing normal instruction.

The processor works following sequences, instruction is loaded from IR register to PC register and processes base on type of instruction. When EI flag is disabled, the interrupt signal does not respond until finish an execution of previous instruction. EI is enabled, if interrupt signal occur by assert *intr* signal by external devices , the IR register releases cache connection and produces itself instruction, *virtual instruction* , this instruction is valid for interrupt instruction and separate instruction to two pieces, *opcode* and operand. *opcode* is responsible to save / restore signal to pc register , operand is first address that kept an interrupt service routine, when PC register receives save signal from control unit, It save current pointer to SP register and jump to next address that produced by virtual *opcode*.

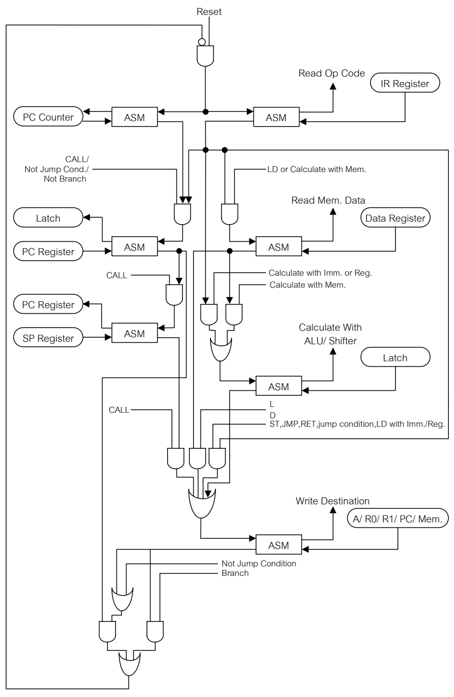

Figure 9: Control Unit Design

IV. Asynchronous System Bus Design

The bus design is multiplex (The data and address transition on the same lines). 8-bit with dual-rail encoded, transition by 4-phase signaling protocol [4]. The designed bus is started with design each component and group them together.

A. *Bus components*[9]

.

a. Bus Interface

Bus interface is designed to interface with asynchronous processor. Due to the multiplex bus design, bus interface also handshakes with bus controller to control the sequence of data, address or spacer transition as shown in figure 10.

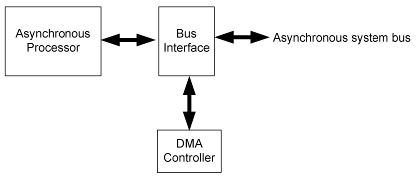

Figure 10: Bus interface

b. Bus Driver

Bus driver acts like an on/off switch to enable/disable transition on the bus. Designed with tri-state buffer and C-element as shown in figure 11 (the element that produces output 1 when all of inputs are 1, produce output 0 when all of inputs are 0, Otherwise its hold previous state).

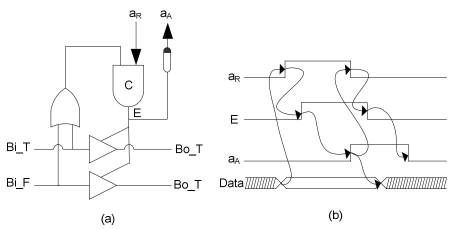

Figure 11: Bus Driver

a_R is request line from bus controller, when data arrive at Bi_T and Bi_F , C-element is enabling tri-state buffer to transfer signal to Bo_T and Bo_F , a_A is the acknowledge signal to the bus controller.

c. Bus lines

Due to high impedance state of Tri-state buffer, there were making some problem for asynchronous circuit. Need to improve to be zero value when it’s high impedance by added weak inverter to each line on the bus [4] as shown in figure 12.

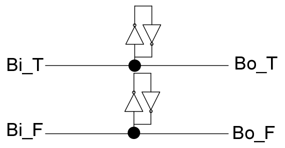

Figure 12: Bus lines

d. Bus Receiver

This component is used to receive data from bus signal and catches all signals before transfer to the destination, designed with C-element as shown in figure 13.

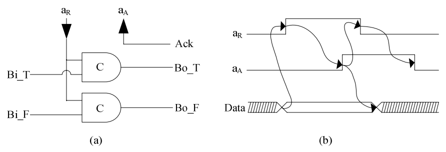

Figure 13: Bus Receiver

a_R is request line from bus controller, when data arrive at Bi_T and Bi_F , a_R is transition from 0 ➔1 C-element enables signal transition to Bo_T and Bo_F, a_A is the acknowledge signal from synchronous interface to bus controller.

e. Bus Controller

Designed with STG (Signal Transition Graph), to control the synchronization on the bus, it’s responsible to control the sequence of data, address, and spacer transition.

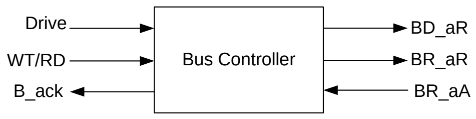

Figure 14: Bus Controller

B. *Transaction*[9]

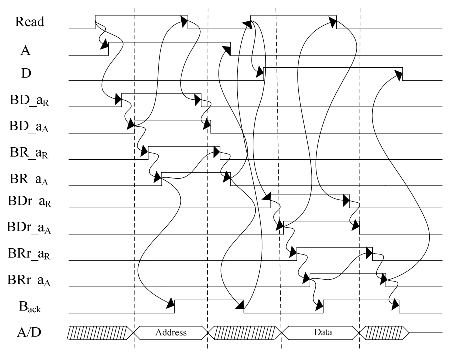

Figure 15: Read Transaction

For read transaction, read signal changes from 0→1 then signal A is the address signal changes from 0→1 indicate the address phase as shown in figure 15.

BD_a_R, BD_a_A, BR_a_R, BR_a_A are the control signals from bus controller change from 0→1 respectively. Indicate the flowing through of data form bus driver to bus receiver components. B_{ack} signal changes from 0→1 indicate the completion of data transfer on address phase. Then all signals change from 1→0 for resetting. After that signal D changes from 0→1 for data phase. *BDr_a_R, BDr_a_A, BRr_a_R, BRr_a_A* signals are the control signal for read data change from 0→1 respectively. B_{ack} changes from 0→1 indicates the completion of data transfer. After that all signals change from 1→0 for signals resetting on return to zero phase.

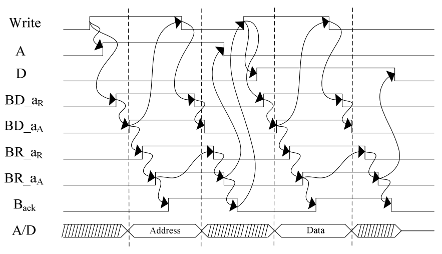

Figure 16: Write Transaction

For write transaction, write signal changes from 0→1 then signal A is the address signal changes from 0→1 indicate the address phase as shown in figure 16.

BD_a$_R$, *BD_a*$_A$, *BR_a*$_R$, *BR_a*$_A$ are the control signals from bus controller change from 0→1 respectively. Indicate the flowing through of data form bus driver to bus receiver components. B$_{ack}$ changes from 0→1 indicates the completion of data transfer on address phase. Then all signals change from 1→0 for resetting. After that signal D changes from 0→1 for data phase.
BD_a$_R$, *BD_a*$_A$, *BR_a*$_R$, *BR_a*$_A$ signals change from 0→1 respectively. B$_{ack}$ changes from 0→1 indicates the completion of data transfer. After that all signal transition from 1→0 for signals resetting return to zero phase.

V. ASYNCHRONOUS DMA CONTROLLER DESIGN

The asynchronous processor is a master device on the asynchronous system bus, that it is able to initiate read and write transfers to all devices on the bus. The DMA controller is slave device, which is only able to response to read or write requests. The DMA controller is required to initiate read and write transfers to memory and also other I/O on the bus, DMA controller also required to be a master device [10].

As there will be more than on asynchronous bus master in the system, the asynchronous system bus specification required the presence of a bus arbiter. The bus arbiter selects which master is to have right to the bus at any one instant in time. An arbiter also exits for the asynchronous processor. Initially the DMA controller is programmed by asynchronous processor. This information is stored in internal DMA registers. Include following information: base address for transfer source, base address in memory to where the data is transferred and size of data transfer. The DMA controller also be slave on the asynchronous system bus in order the asynchronous processor access these registers

A. DMA Function

The DMA controller waits for *dma_req* line to assert and take over the asynchronous system bus from asynchronous processor. It checks its internal registers to obtained details of the transfer. The DMA controller read data from the source, store it in internal buffer, and then write it out to the memory, until the transfer is complete. The DMA controller then release usage of asynchronous system bus and activate the IRQ line in order to indicate the asynchronous processor that transfer was complete. If any time the asynchronous processor requires usage of asynchronous system bus it is able to take priority over the DMA controller. The DMA controller check it has control of the bus at each stage, if it does not the controller wait for the bus become available, the asynchronous processor has finished using the bus [10,11].

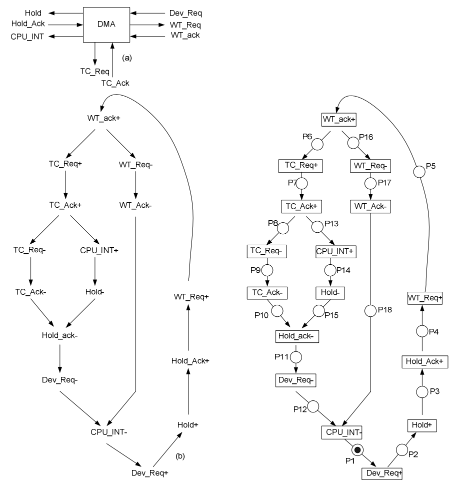

Figure 17: Asynchronous DMA controller Specification, STG and structural encoded

B. DMA Architecture

The internal architecture is very closely based on the classic DMA controller. The architecture proposed of the DMA controller split into above functional units. The most complex of these units is control unit, which consist of a large state transition graph.

C. Reduced STG and CSC support

Several method of synthesis speed independent circuits. One can classify them by the way the synthesis is performed:

State base methods: perform synthesis of the state space of the specification. They can derive optimal implementation, but suffer from state explosion problem and they can only synthesize small size of specification.

Structural methods, working at level of petri-net can synthesize a big size of specification. State base are use in the final state of this method, when specification has been decompose to smaller ones [8].

The synthesis flow is given a consistent STG, encode for all signals resulting an STG contains a new set of signals that ensure unique state and complete state coding property. Depending encoding technique applied, since many of encoding signal may unnecessary to guarantee unique and complete state coding, they are iteratively removed each signal using greedy heuristics until no more signal can be removed without violating unique and complete state coding property. The reduced STG next projected onto different sets of signals to implement each individual output signal. One the reduced STG is reached, it must be computed the complete state support for each non-input signal, applying *csc* support algorithm. Afterward the projection of the STG into the CSC support for each non-input signal is performed, finally speed independent synthesis of each projection is performed, provide the small size of the projection, state based techniques can be applied for performing the synthesis. When synthesizing the projection for each non-input signal a, every signal is the projection but a is considered as an input signal. These prevent the logic synthesis to try to solve possible conflicts for the rest of signals.

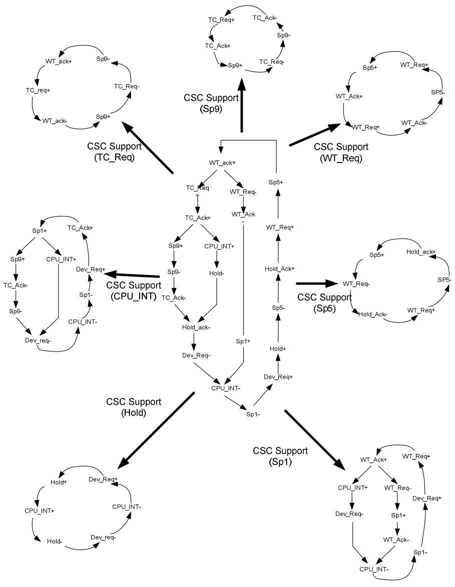

Figure 18: Reduced STG and CSC Support of DMA controller

D. Transaction flow

The DMA transaction following on this sequence

a. I/O Module asserts *DMA_req* signal.
b. DMA asserts Hold signal to processor.
c. Processor asserts *Hold_ack* signal to DMA (processor program *Base_addr* to DMA base register and Count register [ST command]).
d. DMA asserts *DMA_ack* signal to I/O Module.
e. I/O Module asserts *Device_ready* to DMA (transfer type is set by I/O Module (Read or Write) .
f. DMA transfer (read or writes transaction is occurred).
g. DMA asserts TC indicates zero value of count register.
h. DMA interrupts to processor for complete transaction.

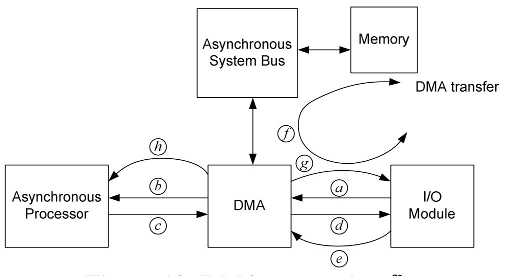

Figure 19: DMA transaction flow

VI. Synchronous And Asynchronous Interfacing

This part is designed for communicating between Synchronous and Asynchronous section, stretchable clock is introduced [6].

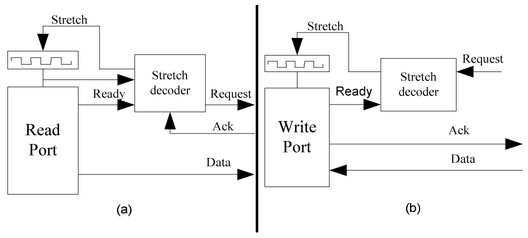

Figure 20: Stretchable clock for read (a) and write (b) interfacing

The clock control is generated using the handshaking signals and the internal state of the synchronous region, a block has either a read port or a write port. Read port provides data for an external element while a write port receive data into the module. In a 4-phase protocol, each module must receive two asynchronous transitions to complete a cycle. Each synchronous module must go through to two clock cycles to allow for two possible stretch signals.

VII. Full System Integration

The full system integration is shown in figure 21, consider; only RAM and I/O section are synchronous section, otherwise it’s fully asynchrony operation. The designed works properly under FPGA delays, timing summary and device utilization are shown on next section.

Components arrangement are shown, each component has its subcomponent. From figure 21 it’s clearly separating between synchronous and asynchronous section.

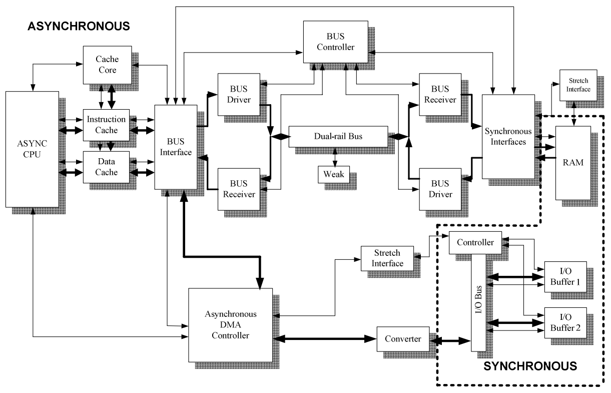

Figure 21: Full System Integration

VIII. Experimental Result

TIMING SUMMARY @- Memory 100 MHz			
INSTRUCTION TYPE	TIME (ns)	INSTRUCTION TYPE	TIME (ns)
Setup Time	32	AND (immediate)	55
Interrupt Response Time	42	AND (Register)	60
Bus Read Cycle	16	AND (Direct)	80
Bus Write Cycle	15	AND (Indirect)	80
DMA Read Cycle	28	OR (Immediate)	80
DMA Write Cycle	30	OR (Register)	80
LD (Immediate)	42	OR (Direct)	120
LD (Register)	40	OR (Indirect)	120
LD (Direct)	65	SUB (immediate)	90
LD (Indirect)	58	SUB (Register)	100
ST (Register)	39	SUB (Direct)	144
ST (Direct)	60	SUB (Indirect)	130
ST (Indirect)	57	ADD (Immediate)	90
JMP	45	ADD (Register)	110
CALL	50	ADD (Direct)	130
RET	45	ADD (Indirect)	130
RETI	43		

Table 1: Timing Summary

Device Utilization Summary			
	Used	Available	Utilization
Slice Flip Flops	2	9312	1%
4-inputs LUTs	1,662	9312	17%
Occupied Slices	880	4,656	18%
Bonded IOBs	82	232	35%
Block RAMs	2	20	10%
GCLKs	1	24	4%
Gate count	141,063		

Table 2: Device Utilization Summary

IX. Concluding Remark

The paper presents FPGA implementation of 8-bit almost completely asynchronous computer. The processor employs an asynchronous system bus, asynchronous cache and asynchronous DMA controller. All designed circuits are high level specification, and it operates correctly under FPGA combinational logic propagation delays, the system uses much of function generator on FPGA, Only 2 flip-flop slices are consumed to prepare an interfacing with synchronous one. The designed has been implemented on Xilinx Spartan-3E FPGA no. 3S500EFG320, partitioning each module to prevent place and routing conflicts.

Acknowledgment

The research is supported by TJTTP-OECF (Thailand - Japan Technology Transfer Project – Japanese Overseas Economic Cooperation Fund)

References

[1] Davis. A, Nowick.S.M "An Introduction to Asynchronous Circuit Design" , 1997
[2] S.Hauck. "Asynchronous Design Methhodologies : An Overview". Proceedings of the IEEE. vol.83. No.1. pp.69-93, January 1995.
[3] J.Bhasker. "*A VHDL Primer*". Prentice-Hall, Inc., 1992. ISBN 0-13-952987-X.
[4] Molina, P.A.; Cheung, P.Y.K ,"A Quasi Delay-Insensitive Bus Proposal for Asynchronous Systems" Proceedings Third International Symposium on
Advanced Research in Asynchronous Circuits and Systms, 1997 Pages:129-139
[5] T.Nanya., Y.Ueno., H.Kagotani., M.Kuwako., A.Takamura. TITAC : Design of a Quasi-Delay-Insensitive Microprocessor. IEEE Design & Test of Computers., Vol.11., No.2, pp.50-63, Summer 1994.
[6] Bormann, D.S.; Molina, P.A.; Cheung, P.Y.K, "Combining asynchronous and synchronous circuits using stretchable clocks" IEEE Colloquium on Design and Test of Asynchronous Systems, 28 Feb 1996,pp.4/1-4/8
[7] P.Ruangsilsap. "Design of 8-bit scalable-delay-insensitive microprocessor using FPGA". Master thesis, Chlalongkorn University, Bangkok, Thailand, 2001
[8] J. Carmona and J. Cortadella. State encoding of large asynchronous controllers. *In* Proc. ACM/IEEE Design Automation Conference, *pages 939-944*, July 2006.
[9] S.Sudeng, A.Thongtak:" A Design of System Bus for Asynchronous Circuits" International Technical Conference in Circuits/Systems Computer and Communication (ITC-CSCC2006), Pang Suan Kaew hotel, Chiangmai, Thailand on July 10-13, 2006.
[10] S.Sudeng, A.Thongtak:" Asynchronous System Bus Enhancement by Interrupt and DMA Technique" International Annual Conference in Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI-CON2007), Mae Fah Luang University, Chiangrai, Thailand on May 9-12, 2007.
[11] S.Sudeng, A.Thongtak:" Signal Transition Graph Based Logic Synthesis for Asynchronous Control Circuits Using Template Based Method" IEEE Tencon 2007, IEEE Region 10 conference (IEEE Tencon2007), Taipei International Convention center, Taipei, Taiwan on Oct 30- Nov 2, 2007.