Desynchronization of a Processor

Vikas Vij, University of Utah

***Abstract*--This paper presents the novel concept of desynchronization which deals with deriving an asynchronous circuit from an optimized synchronous circuit by replacing the clock distribution tree by a handshaking network. A new controller for implementing desynchronization is also proposed which achieves more concurrency. This concept is implemented on a MIPS processor and a comparison of the synchronous and the desynchronized version is also presented.**

***Index Terms*--Asynchronous circuits, desynchronization, handshake protocol, synthesis.**

I. Introduction

As the system on chip (SOC) becomes faster and larger, it is becoming increasingly difficult to distribute the clock to the entire chip because of clock skew. For achieving low skew clock distribution, low skew drivers are used which are extremely power hungry.

Asynchronous circuits can be seen as solution for this problem as they switch only to do useful work and skew is also not an issue for them. Asynchronous circuits are also modular and exhibit average case performance instead of worst case which is the case for synchronous circuits. They exhibit dramatic improvements in terms of EMI (Electro-magnetic Interference). This is due to the fact that flip flops no longer switch in phase and thus reduces noise power. Asynchronous circuits are inherently closed loop and hence more robust in the presence of PVT (Process, voltage and temperature) variations.

Asynchronous design methodology is not widely used due to the following reasons:

i) Lack of good CAD (Computer Aided Design) tools that completely cover the design flow.

ii) Changing the mentality of the designers from synchronous to asynchronous.

Desynchronization basically helps in solving these problems. It deals with generating the asynchronous circuits from an optimized synchronous circuit using the standard EDA (Electronic Design & Automation) tools and flow with slight modification. The essential idea is to start with a synchronous synthesized circuit and then replacing the global clock network with a set of handshaking circuits.

Desynchornization is based on the basic concept of Micropipelines. Micropipelines concept was introduced in [9]. It was based on 2 phase communication protocol. This concept was explained on 4 phase communication protocol by [5], [7] and [8]. In these various kinds of handshake circuits were presented.

The implementation of desynchronization in [1] - [4] is different from this implementation, as individual control logic for each latch of each bit of the databus and not a common control logic for the entire bus. Due to this, if there is an 8 bit databus, then there will be 8 copies of the handshake circuit.

The implementation in [6] is also interesting as it implements a common handshake circuit for a bus and also gives a comparison of desynchronization performed on various circuit blocks like datapath, traffic light controller, FIR filter, etc. and uses the Doubly Latched Asynchronous Pipeline ([5]) concept for latches.

II. Desynchronization

This paper considers two kinds of implementation of Desynchronization, one with master-slave latches and other with D-flipflops.

A. Steps in Desynchronization

The desynchronization method proceeds in three steps.

1) This is an optional step as handshake circuits for both D-flipflops as well as Master slave latches will be developed. The latter implementation requires the conversion of the flip-flop-based synchronous circuit into a latch-based one [M and S latches in Fig. 1(b)]. D flipflops are conceptually composed of master–slave latches. To perform desynchronization, this internal structure is explicitly revealed [see Fig. 1(b)] to:

   a) Decouple local clocks for master and slave latches (in a D flip-flop, they are both derived from the same clock);

   b) Optionally improve performance through retiming, i.e., by moving latches across combinational logic.

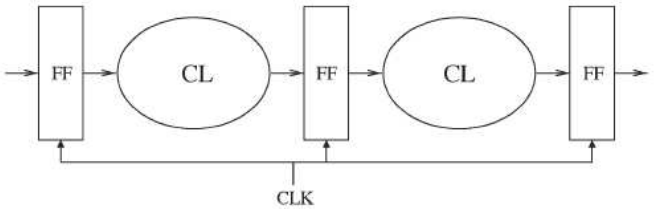

Fig. 1(a) - Synchronous circuit

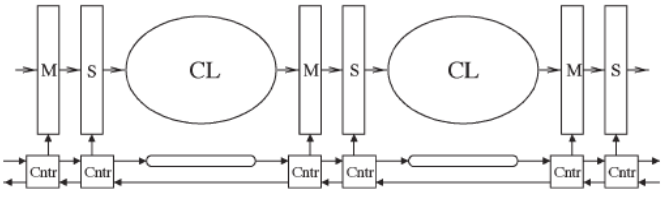

Fig1. (b) - Desynchronized circuit.

2) Generation of matched delays for the combinational logic [denoted by rounded rectangles in Fig. 1(b)]. Each matched delay must be greater than or equal to the delay of the critical path of the corresponding combinational block. Each matched delay serves as a completion detector for the corresponding combinational block and ensures that the data reaches the latch/flip-flop before the control signal (bundled data constraint).
3) Implementation of the local controllers.

B. Other Implementation

This paper also considers the implementation of non-linear pipelining structures for forks and joins too (Fig. 2).

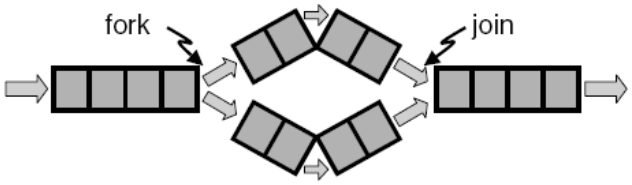

Fig. 2 - Fork and Join structure

Forks and joins are common structures in a processor and needs to be addressed in order achieve desynchronization. There are two ways forks and join can be approached:

a) They are individually implemented outside the asynchronous handshake circuit.
b) They are implemented inside the asynchronous handshake circuits, as part of it.

III. Protocols For Desynchronization

The most important part of desynchronization is the design of the asynchronous handshake circuit, as the latency and throughput of the data is directly dependant on it. Design of this circuit requires tradeoff between data concurrency and circuit complexity, which also affects the speed of operation. As the data concurrency is increased, the complexity of the handshake circuit also increases, which makes it slower. Similarly if the handshake circuit is kept simple and hence fast, not much data concurrency can be achieved.

The handshake circuits were designed using the single rail encoding, with 4 phase protocol (Return to Zero protocol) and complying with bundled data constraints. Two types of asynchronous handshake circuits were explored and their state transition graph and synthesized circuit description are shown in Fig. 3 and Fig. 4.

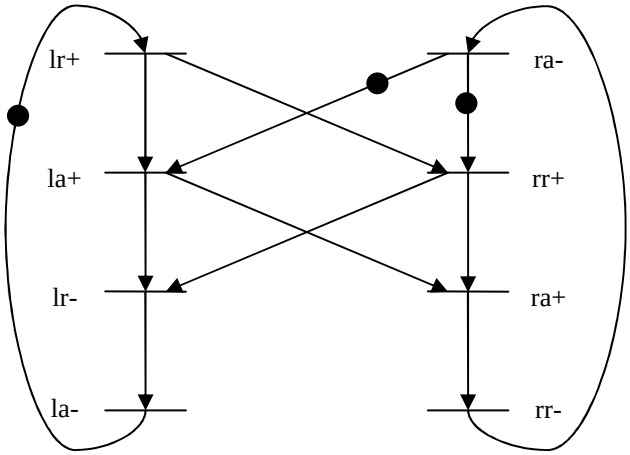

Fig. 3(a) – State graph for asynchronous block 1

la = lr (ra’ csc0’ + la)
rr = ra’ rr + la csc0’
csc0’ = rr’ la’

Fig. 3(b) – Synthesized circuit description for asynchronous block 1

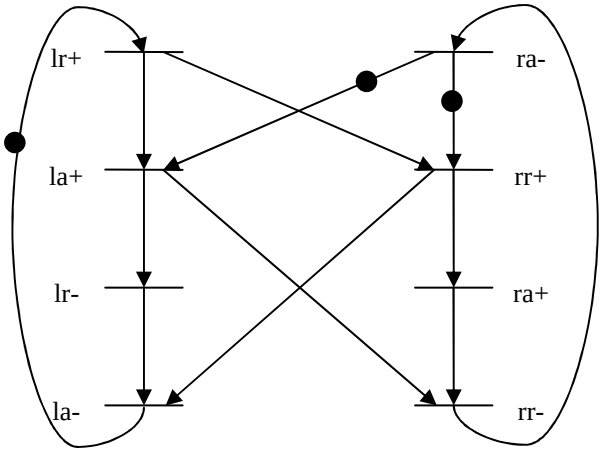

Fig. 4(a) – State graph for asynchronous block 2

la’ = la’ (lr’ + csc1’ + ra) + csc1’ lr’
rr’ = (rr’ + ra) (la’ + csc1’)
csc0 = rr
csc1 = rr’ (la’ + csc1)

Fig. 4(b) – Synthesized circuit description for asynchronous block 2

Both these implementation were synthesized using Petrify ([10]), with automatic timing optimization and complex gate

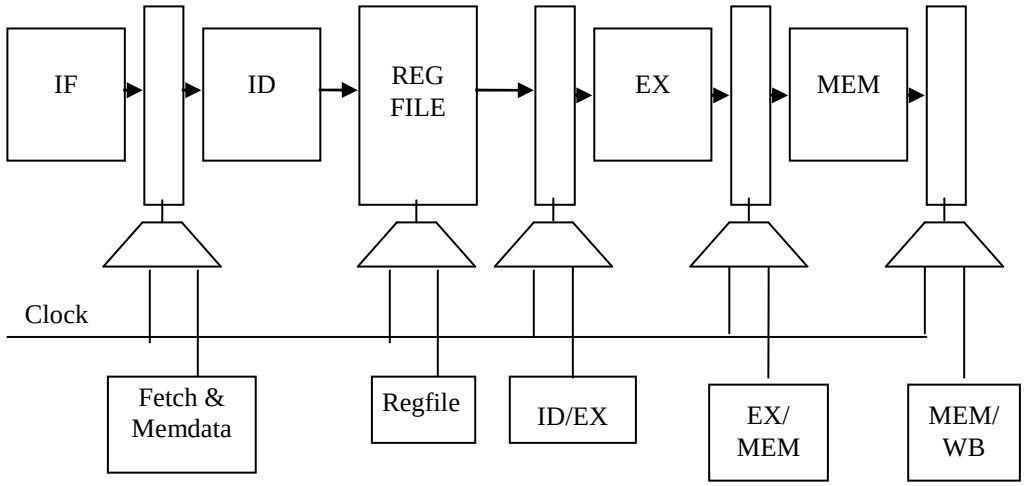

Fig. 5 – Block Diagram of Desynchronized Pipelined MIPS

implementation. The first implementation seems to be better because of easier control circuit. (Need to test the asynchronous blocks and write about the tests with waveforms. Also need to synthesize the asynchronous circuit and put its Figure instead of the description).

IV. Synthesis & SOC of MIPS Microprocessor

This section deals with the MIPS Microprocessor and also with its synthesis and SOC results.

A. MIPS

MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC (Reduced Instruction Set Computer) microprocessor architecture. The MIPS microprocessor with 4 pipeline stages was used for implementing the desynchronized system. Fig. 5 shows the block diagram for implementing desynchonization of the pipelined MIPS microprocessor.

To achieve this, the verilog code for the MIPS was divided into 3 parts (i.e. controller, alucontrol and datapath) and a MUX was inserted into the verilog code for each pipeline stages, for selection between clock and asynchronous input coming from the asynchronous block. The select pin of each MUX is used to set the microprocessor in synchronous or desynchronized mode.

B. Synthesis Results

The processor blocks were synthesized with Synopsis Design Compiler using the Artisan 130 nm library.

Table 1 shows the comparison between the area and power values given by Design Compiler for synchronous and desynchronized implementation. The values represented are without the asynchronous block.

It is indeed interesting to note the increase in the area of the desynchronized implementation because of the extra logic for MUX's, but surprisingly the power usage of the desynchronized implementation is much less than that of the synchronous implementation.

Table 1 – Synthesis results for synchronous and desynchronized implementation

	Synchronous		**Desynchronous**	
	Area (um2)	Power (uW)	Area (um2)	Power (uW)
Controller	537.12	30.76	550.08	24.66
Alucontrol	84.96	4.87	84.96	5.84
Datapath	6995.52	658.20	8369.28	437.37
Async. Block	--	--	1933.92	90.21
Total	7617.60	693.83	10938.24	558.08

C. Synthesis Issues

In order to add a MUX for selection between the handshake circuit and clock, a few problems occurred in synthesis as shown in Fig. 6. Though the data required time is greater than data arrival, then also a negative slack is shown in the constraints file. But in the paths file, none of the paths show a negative slack as shown in Fig. 7.

```
Startpoint: memdata[3] (input port clocked by clk)
Endpoint: ir0_q_reg_3_0
          (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min

Point                                   Incr       Path
-----------------------------------------------------------
clock clk (rise edge)                   0.00       0.00
clock network delay (propagated)        0.00       0.00
input external delay                    0.05       0.05 r
memdata[3] (in)                         0.00       0.05 r
U203/Y (CLKBUFX2TS)                     0.09       0.14 r
U202/Y (CLKBUFX2TS)                     0.12       0.26 r
ir0_q_reg_3_0/D (EDFFXLTS)              0.00       0.26 r
data arrival time                                  0.26

clock clk (rise edge)                   0.00       0.00
clock network delay (propagated)        0.42       0.42
clock uncertainty                       0.05       0.47
ir0_q_reg_3_0/CK (EDFFXLTS)             0.00       0.47 r
library hold time                      -0.10       0.37
data required time                                 0.37
-----------------------------------------------------------
data required time                                 0.37
data arrival time                                 -0.26
-----------------------------------------------------------
slack (VIOLATED)                                  -0.10
```

Fig. 6 – Screenshot of datapath.constraints file

```
Operating Conditions: typical   Library: typical
Wire Load Model Mode: segmented

Endpoint                        Path Delay     Path Required     Slack
-------------------------------------------------------------------------
zero (out)                        5.37 r           5.90          0.53
zero (out)                        5.25 r           5.90          0.65
zero (out)                        5.24 r           5.90          0.66
pcreg_q_reg_7_0/D (EDFFTRX1TS)    5.30 f           5.97          0.67
zero (out)                        5.14 r           5.90          0.76
pcreg_q_reg_7_0/D (EDFFTRX1TS)    5.18 f           5.97          0.80
pcreg_q_reg_7_0/D (EDFFTRX1TS)    5.17 f           5.97          0.81
zero (out)                        5.04 r           5.90          0.86
pcreg_q_reg_7_0/D (EDFFTRX1TS)    5.06 f           5.97          0.91
zero (out)                        4.97 r           5.90          0.93
zero (out)                        4.95 r           5.90          0.95
zero (out)                        4.94 r           5.90          0.96
zero (out)                        4.93 r           5.90          0.97
zero (out)                        4.92 r           5.90          0.98
zero (out)                        4.92 r           5.90          0.98
zero (out)                        4.91 r           5.90          0.99
zero (out)                        4.91 r           5.90          0.99
zero (out)                        4.89 r           5.90          1.01
pcreg_q_reg_0_0/D (EDFFTRX1TS)    4.93 f           5.97          1.05
zero (out)                        4.84 f           5.90          1.06

1
```

Fig. 7 – Screenshot of datapath.paths file

D. SOC Results and Issues

SOC was done on each block individually and passed with no connectivity or geometry errors.

Without specifying the pin placements, there were few metal pieces left unconnected and also a few pins overlapped, but no connectivity or geometry errors were reported by the tool. This occurred for the alucontrol block which is the same for both kinds of implementations.

FABRICATION

The chip was fabricated in the AMI 0.6 micron technology using the MOSIS circuit-fabrication service. It's a free service which helps students to fabricate designs and receive a physical chip which can be tested for functionality. In return MOSIS requires students to provide a report of the testing results of the chip.

The fabricated chip has a synchronous and a desynchronous 8-bit processor which can be selected using a select pin. Also delay selection pins were kept in order to select the delay between various asynchronous handshake controllers. Over and above this there were also some output pins to keep track of the state of the processor as well as the levels of individual asynchronous inputs given to clock pins of each pipeline stages.

The unit used to measure the size of a chip is called TCU (Tiny Chip Unit). This chip had a size of four TCU.

TESTING

The chip was tested using the Tektronix LV-500 tester, which is a very old tester. It can store up to 64000 test vectors which can be fed to the chip.

Testing on the LV500 tester can be done in 2 ways.

1) Write all the pin names and all the details manually using the keyboard of the tester
2) Make a setup file having all the details about the pins and the values forced and compared at the pins, which can be fed to the tester as a .msa file using FTP.

The synchronous version of MIPS was tested by forcing the input pins based on the opcodes of the instructions and then the output pin levels were compared to the voltage levels obtained from simulation results. Also the memread and the memwrite pins were checked to see if their levels were as per the execution of the instruction, like in the 4 stage fetch cycle, the memread pin needs to be high for all the four cycles and then based on the type of instruction the next few cycles will have the memread pin low till the instruction execution was completed except in case of the load instruction wherein the value needs to be loaded from the address generated.

The desynchronous version of MIPS was tested in 2 stages. In the first stage the desynchronous version was run and checked if the address pins got incremented or not. This was done because these circuits have a peculiar property of self testability, which means that if the circuit doesn't work then it stalls. In the second stage, the various test pin levels were checked on the oscilloscope and their functionality checked.

RESULTS

1) Synchronous version

The synchronous version of the chip worked fine with a clock period of 100 ns and was tested for the Load and Arithmetic instructions.

2) Desynchronous version

The self testing of the chip resulted into no changes on the address bits, which means that there was stall in the transfer of any one request or acknowledge signal from or to one of the blocks. To know the source of the problem, the second step was run, but the resulting signals couldn't be isolated to find the source of the problem. So a different testing methodology was formulated to know which signals were generated properly and whether the pulse width was enough or not.

The chip was run in synchronous mode and the waveforms on each test pins for asynchronous inputs given to each pipeline stage were seen on an oscilloscope.

Below are the waveforms obtained from these tests:

Fig. 8 shows the comparison of the async input 1 and 2, which are fed to the registers which store the instructions bits 7 to 0 and 15 to 8 respectively. Fig. 9 shows the comparison of the async input 2 and 3, which are fed to the registers which store the instructions bits 15 to 8 and 23 to 16 respectively. Fig. 10 shows the comparison of the async input 2 and 4, which are fed to the registers which store the instructions bits 15 to 8 and 31 to 24 respectively. These waveforms show the proper traversal of the request and acknowledge handshake signals between the asynchronous blocks involved with instruction fetch.

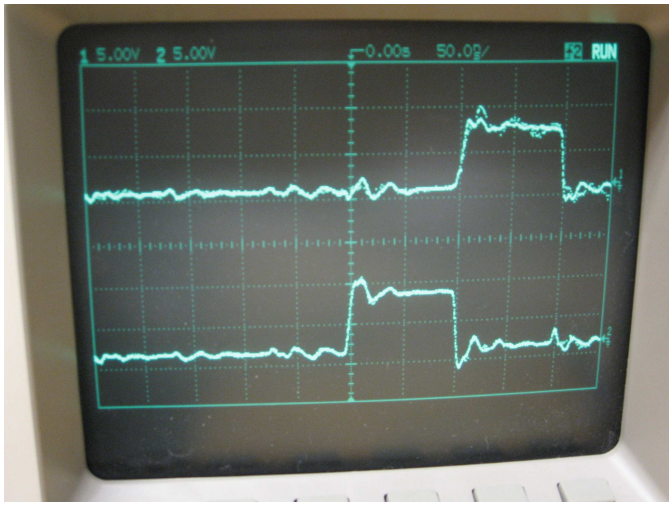

Fig 8 – Async input 1 (bottom) and 2 (top) comparison

Fig 9- Async input 2 (top) and 3 (bottom) comparison

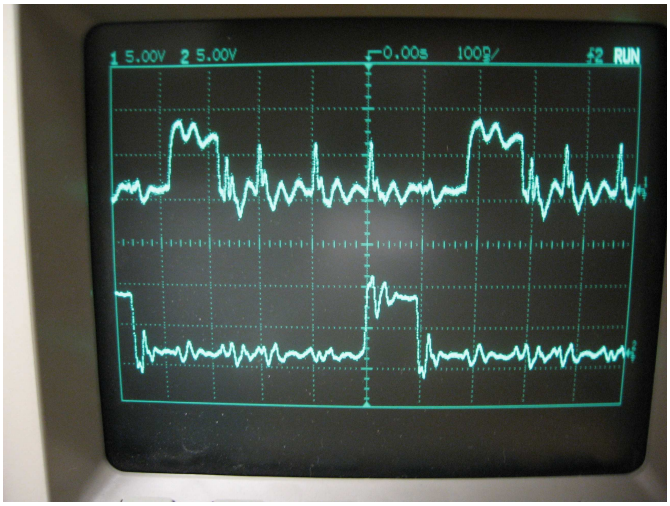

Fig 10 – Async input 2 (bottom) and 4 (top) comparison

Similarly Fig 11, 12 and 13 shows the comparison of async input 1 with respect to address bits. These waveforms give a comparison between the sync pulse width and the async pulse width and also shows that the address bits getting incremented properly for the synchronous MIPS.

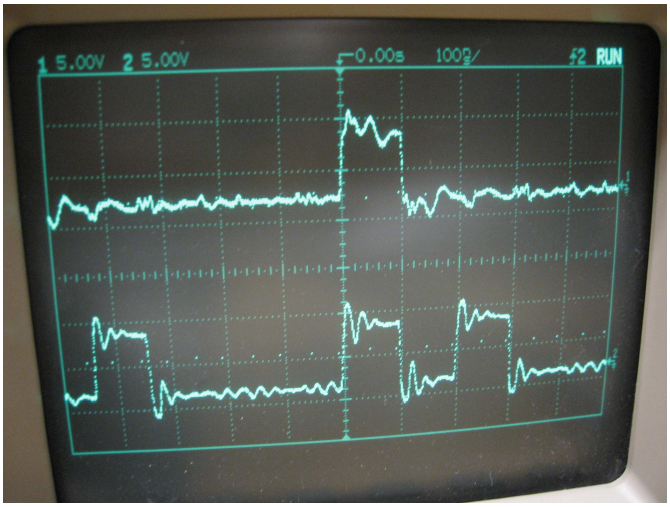

Fig 11 – Address pin 1 (bottom) and async input 1 (top) comparison

Fig 12 – Address bit 0 (top) and 1 (bottom) comparison

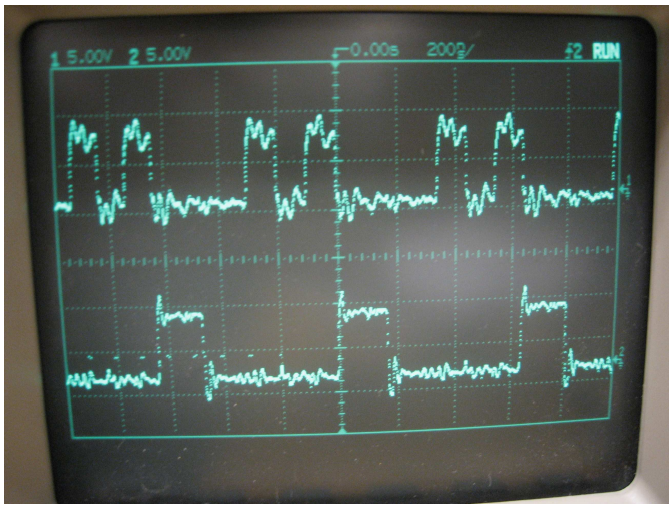

Fig 13 – Address bit 0 (top) and 2 (bottom) comparison

Fig 14 shows a comparison between the async input generated for the fetch block and for the MEM/WB pipeline stage. It helps to know one of the problems in the desynchronous version. The async input 5 is not able to reach the pipeline registers; this can be due to the fact that the wire lengths driven by the gate generating the input is much larger and so the signal strength is not enough. But the request and acknowledge signals are produced properly and that is why async input 1, 2, 3, and 4 were generated (Fig. 8, 9, 10). Over and above this it was also observed that the asynchronous inputs with a very small pulse width, i.e. less than 15 gate delays were not generated properly and hence stalling the circuit.

Another problem found was that the central state variable needs to be incremented on startup by async input 5, but this never gets generated, which is the main reason for stalling of the chip. If this would have been working fine then the delays set between various pipeline stages could be checked accurately.

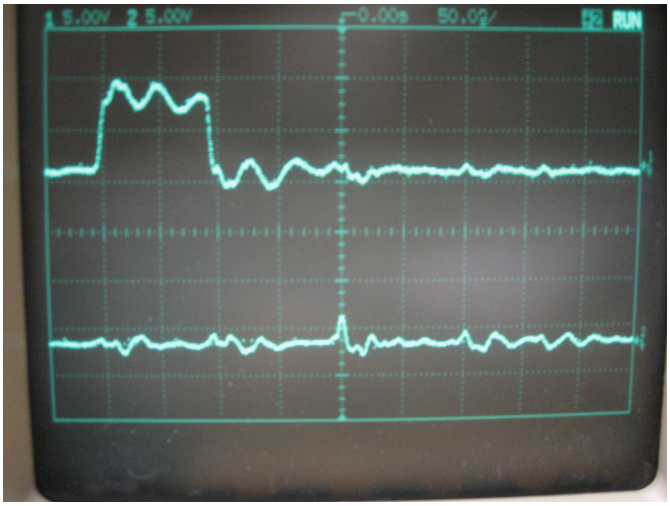

Fig 14 – Async input 1 (top) and 5 (bottom) comparison

The schmoo plots (Fig. 15) generated for the circuit showed that the circuit will operate normal if the voltage was above 4.5V and the clock period was above 92 ns. This range is only for the synchronous version, but for the asynchronous version the circuit needs to be checked for timing errors.

Fig. 15 – Schmoo plot for the chip

All the problems described above can be resolved by firstly generating the constraints and sdf file for synthesis and place and route. The constraints help to automatically generate the delay blocks between the asynchronous controllers. It will also be useful to drive the optimum wire load and help in additions of repeaters wherever necessary. The sdf file can be back

annotated to get a more accurate simulation result, by using the timings written for various gates and wires. Tools like Primetime can be used to do a much better job for generating timing numbers. In order to get better timing numbers for this chip further detailed tests need to be conducted which will help in understanding the issues that resulted into problems, like during testing some pins it was found that due to design error the state status pin and the pcsource test pins were short circuited, this did not affect the functionality of the chip, but increased the difficulty to test the chip. Similarly the consideration that the external RAM didn't send any request signal to the chip when its data gets ready. This leads to problems in feeding the correct test vector in the desynchronous version.

CONCLUSION

The chip was fully functional for the synchronous mode, but not for the desynchronous version. But it gave a very good insight on the development of desynchronous circuits and also on the parameters to be considered while developing such kind of circuits. A much better job needs to be done in terms of timing verification of the chip which couldn't be done because of the time available to test the chip, which was less than 15 days.

GLOSSARY

lr – Left Request
la – Left Acknowledge
rr – Right Request
ra – Right Acknowledge
async - Asynchronous

REFERENCES

[1] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. A concurrent model for De-synchronization. In Proceedings of the International Workshop on Logic Synthesis, pager 294-301, 2003.

[2] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. From synchronous to asynchronous: an automatic approach. DATE-2004.

[3] J.Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications. IEEE Trans. on CAD of Integrated Circuits and Systems 25(10): 1904-1921 (2006)

[4] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin and C. Sotiriou. Handshake Protocols for De-Synchronization. ASYNC 2004: 149-158

[5] R. Kol and R. Ginosar. A doubly-latched asynchronous pipeline. In Proc. International Conf. Computer Design (ICCD), pages 706-711, Oct. 1996.

[6] A. Branover, R. Kol and R. Ginosar. Asynchronous design by conversion: Converting synchronous circuits into asynchronous ones. DATE-2004.

[7] P. Day and J. V. Woods. Investigation into Micropipeline Latch Design Styles. IEEE Transactions on VLSI Systems, June 1995.

[8] S. B. Furber and P. Day. Four Phase Micropipeline Latch Control Circuits. IEEE Transactions on VLSI Systems, June 1996.

[9] I. E. Sutherland. Micropipelines. Communications of the ACM, June 1989.

[10] Cortadella *et al.*, "Petrify," *IEICE Trans. Information and Systems*, E80-D, 315—325, 1997.

[11] CMOS VLSI Design, Third Edition – Neil H. E. Weste and David Harris, Addison Wesley.