An optimization for the design of a simple asynchronous processor

SUN-YEN TAN [1], WEN-TZENG HUANG [2]

[1] Department of Electronic Engineering
National Taipei University of Technology
No. 1, Sec. 3, Chung-hsiao E. Rd., Taipei,10608, Taiwan, R.O.C.
sytan@ntut.edu.tw

[2] Department of Computer Science and Information Engineering
Mingsin University of Science and Technology
No.1, Xinxing Rd., Xinfeng Hsinchu 30401, Taiwan, R.O.C.
wthuang@must.edu.tw

Abstract: - The asynchronous circuit style is based on micropipelines, a style used to develop asynchronous microprocessors at Manchester University. This paper has presented some engineering work on developing a micropipeline Stump processor. The work presented in this paper demonstrates that VHDL can be used to describe the behaviour of micropipelined systems. It also shows a comparison of 2-phase and 4-phase implementations in transistor count, speed, and energy. Though the nature of the work is mainly engineering, there are some significant new insights gained in the course of the work. The 2-phase circuits have good performance in speed. This is due to the rising and falling transitions of the 4-phase circuits following the same routes. Asymmetric delays with fast reset circuit can be applied to improve the performance. The fastest speed is 1.55 MIPS for the two-phase synthesized processor and the lowest power consumption is 362.33 fj for the synthesized four-phase long hold processor.

Key-Words: - Asynchronous design, Micropipelines, Processor, VHDL, Synthesis

1 Introduction

The design of asynchronous circuits is more difficult than that of synchronous circuits. Hazards must be removed from the circuits to ensure that there are no unexpected transitions. Well structured asynchronous design styles such as micropipelines reduce the difficulty. Event-driven logic modules may be designed by electronic experts. Then designers with less experience can easily build micropipelined circuits using such modules. An automatic synthesis tool is available [1][4]. It converts the behavioural VHDL into structural VHDL and Verilog based on micropipelines had been published [1][7][11][14][15]. 2-phase and 4-phase VHDL models of event-drive logic modules and standard logic function elements were created.

In this paper we demonstrate the design of an asynchronous processor using the sytnthesizer and evaluate the experimental results. We also discuss the return to zero problems for the 4-phase designs. An optimization method is applied to improve the performance of the synthesized processors.

Section 2 introduces some asynchronous design techniques. Section 3 briefly describes nicropipelines and introduces 2-phase and 4-phase event-driven Logic modules. The Stump processor design will be presented in Section 4. Section 5 will present experimental results. Section 6 will discuss the return to zero problems. Section 7 will demonstrate an optimization method. Finally, Section 8 will give a short conclusion.

2 Asynchronous design

Asynchronous design has potential advantages over synchronous design [8][9][10], such as no clock skew problem, low power, average case performance and good Electro-Magnetic Compatibility (EMC). The benefits may be most apparent in mobile communication applications and other portable systems which use advanced VLSI technologies.

Asynchronous logic circuits have several important advantages over their counterparts in clocked logic. An asynchronous logic function is potentially faster because it works at the average-case delay rather than the worst-case delay. There is no global clock on asynchronous circuits so they will not unnecessarily dissipate power when there is no useful work to do. Asynchronous logic has the potential for low power [5]. Asynchronous logic may be used to implement systems with lower power dissipation.

3 Micropipelines

The design of asynchronous circuits generally follows a modular approach, where a system is designed as an interconnection of modules. In the 1988 Turing Award Lecture, Sutherland expounded a modular approach to building hardware systems based on data-driven asynchronous self-timed logic elements called micropipelines [6]. In the 4-phase handshaking protocol, only rising transitions or only falling transitions of either control wire have the meaning; they represent request events or acknowledge events.

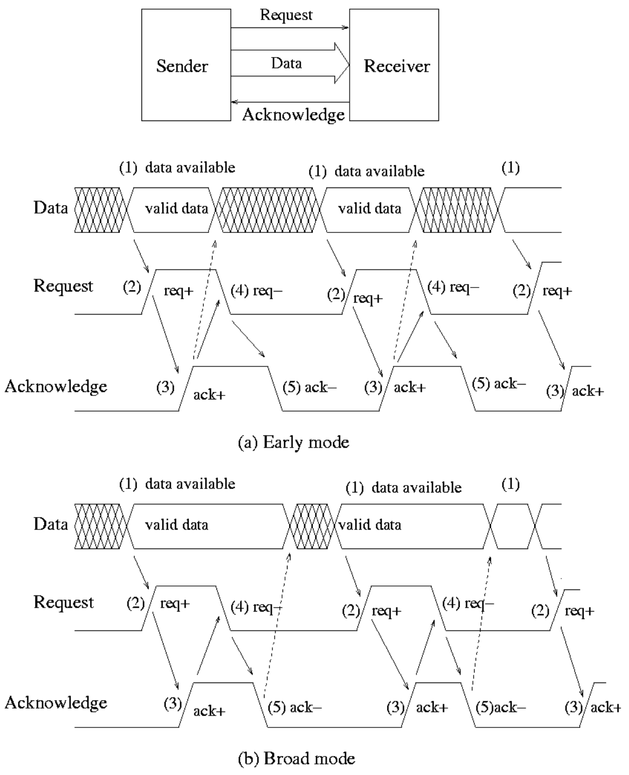

Fig. 1. 4-phase bundled data convention

In this signalling scheme, the operating cycle is (1) data available (2) change request to active state, (3) change acknowledge to active state, (4) return request to inactive state, and (5) return acknowledge to inactive state. If the active state is **logic "1"** the the operating cycle is (1) data available (2) **request+**, (3) **acknowledge+**, (4) **request-**, and (5) **acknowledge-**. Fig. 1 illustrates two kinds of four phase signalling, the **'early'** mode and the **'broad'** mode [2]. The **'early'** mode (Fig. 1(a)) uses the rising edge of the Request line to indicate **'data available'** and the rising edge of the Acknowledge line to indicate **'data latched'**. The falling edges are return to zero actions that carry no meaning. The **'broad'** mode (Fig. 1(b)) uses the rising edge of the Request line to indicate **'data available'** and the falling edge of the Acknowledge line to indicate **'data latched'**. Another possible protocol is **'late'** mode which uses the falling edges as active.

Various event-driven logic modules for controlling transition signals are shown in Fig. 2. They were devised for composing to 2-phase control circuits. Muller C-elements and XOR gates are the same whether they are used in 2- or 4-phase designs. However, 4-phase Toggle, Select, Call and Arbiter modules are different from their 2-phase counterparts.

A Toggle is used to alternately deliver events on its input to one of two outputs. In the 2-phase protocol each transition denotes an event. Therefore, the odd number transitions on the input of a Toggle will be sent to the dotted output and the even number transitions on the input of a Toggle will be sent to the non-dotted output. In the 4-phase protocol each event consists of a rising transition and a following falling transition. A rising transition and the following falling transition must be sent to the same output. Therefore, the odd number rising and falling transitions on the input of a Toggle will be sent to the dotted output and the even number rising and falling transitions on the input of a Toggle will be sent to the non-dotted output.

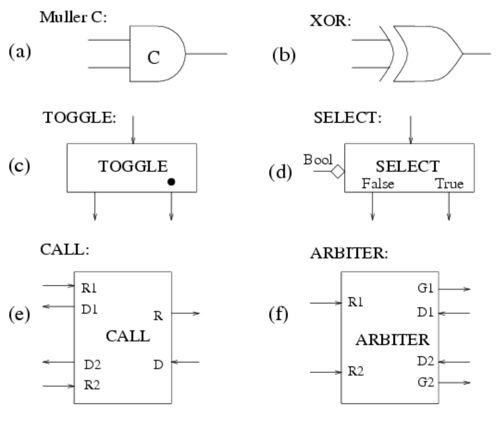

Fig. 2. Various event-driven logic modules.

Furber and Day developed four kinds of 4-phase latch control circuits. They are the simple, semi-decoupled, fully decoupled and long hold 4-phase latch control circuits [3][11]. They use the 4-phase bundled data convention.

4 Stump processor

A simple processor example, Stump [12] is a 16-bit mini-risc processor which has 8 registers and a 4-bit condition code register. R0 is always zero and can be used as a source operand allowing **Move** instructions to be synthesized from an **Add** instruction. R0 may be written to, but the result is always discarded allowing **Compare** instruction to be synthesized from **Subtract** instructions. R7 is the program counter. The 4-bit condition code register contains Sign Flag(N), Zero Flag(Z), Overflow Flag(V) and Carry-Out Flag(C) shown in Table 1.

Table 1 The Condition code

Bit 3	Bit 2	Bit 1	Bit 0
Sign Flag (N)	Zero Flag (Z)	Overflow Flag (V)	Carry-Out Flag (C)

The condition code register is updated to reflect the result of the ALU operation when arithmetic and logical instructions are executed. The condition code register is not changed for Load, Store, and Branch instructions. The instruction set and its 3 formats are shown as Tables 2, 3, 4 and 5.

Table 2 Instruction set

Opcode	Instruction	Explanation
000	ADD	2's complement addition
001	ADC	2's complement addition with carry-in
010	SUB	2's complement subtract
011	SBC	2's complement subtract with borrow
100	AND	Bitwise AND of two 16-bit words
101	OR	Bitwise OR of two 16-bit words
110	LD/ST	Load Reg. from memory or Store Reg. to memory
111	Bcc	Branch if condition cc is satisfied

Table 3 Instruction format:
Type 1: 2 Source registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INSTR			0	LD CC	DST			SRC A			SRC B			SHIFT	

Table 4 Instruction format:
Type 2: 1 Source register, 1 immediate value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INSTR			1	LD CC	DST			SRC A			Immediate				

Table 5 Instruction format:
Type 3: Conditional Branch

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Condition				Offset							

Table 6 shows shift operations. Table 7 shows conditional branch instructions. The behavioural VHDL description can be used to describe the asynchronous operation of the processor. The following programs are used to test how the model works.

Table 6 Shift operations

Operation	**Instr Bit 1**	**Instr Bit 0**	**SRC A, bit 15:=**	**Carry-out:=**
No shift	**0**	**0**	**A15**	**0**
ASR	**0**	**1**	**A15**	**A0**
ROR	**1**	**0**	**A0**	**A0**
RRC	**1**	**1**	C_{in}	**A0**

Table 7 Conditional Branch Instructions

Mnemonic	**Bits 11:8**	**Condition**	**Explanation**
BAL	**0000**	**Always**	
BNV	**0001**	**Never**	
BHI	**0010**	$C + Z = 0$	**comparison: unsigned arithmetic**
BLS	**0011**	$C + Z = 1$	
BCC	**0100**	$C = 0$	**overflow test: unsigned arithmetic**
BCS	**0101**	$C = 1$	
BNE	**0110**	$Z = 0$	**zero test**
BEQ	**0111**	$Z = 1$	
BVC	**1000**	$V = 0$	**overflow test: signed arithmetic**
BVS	**1001**	$V = 1$	
BPL	**1010**	$N = 0$	**comparison: signed arithmetic**
BMI	**1011**	$N = 1$	
BGE	**1100**	$N \oplus V = 0$	
BLT	**1101**	$N \oplus V = 1$	
BGT	**1110**	$(N \oplus V) + Z = 0$	
BLE	**1111**	$(N \oplus V) + Z = 1$	

```
          LD  R4,  0020H
          LD  R1,  ( 0021H )
          LD  R2,  ( 0022H )
          ADD  R3,  R1,  R2
          LD  ( 0023H ),  R3
LOOP:     BAL LOOP
```

(A)

```
LOOP:     LD  R4,  0020H
          LD  R1,  ( 0021H )
          LD  R2,  ( 0022H )
          ADD  R3,  R1,  R2
          LD  ( 0023H ),  R3
          BAL  LOOP
```

(B)

A test program which contains the machine code of the above program (A) and some initial memory data and the behavioural memory model for the program simulation. The simulation result of the behavioural asynchronous Stump description is shown in Fig. 3.

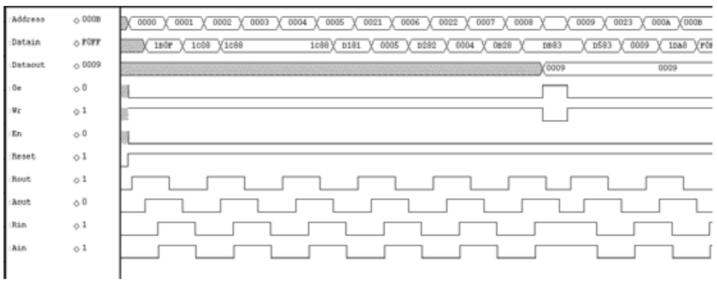

Fig. 3 The waveform of the simulation of a behavioural 2-phase Stump processor

The synthesizer read the behavioural VHDL description and generated two-phase and 4-phase Stump processor circuits. The synthesized two-phase structural Stump processor is shown in Fig. 4. When the test program was used to simulate the synthesized two-phase structural Stump processor the correct result was obtained. Different 4-phase latch control circuits were used to generate various 4-phase implementations of the Stump processor from the same description. Two changes are required for the 4-phase implementations.

The first one is for the start circuit of the processor. This is shown in Fig. 5. Because a rising transition and a falling transition have to flow into the r1 input of the **Call** module at the top of the circuit in Fig. 4, an **XOR** and a two-phase **Toggle** are required to enable the rising transition of Reset to pass through the **XOR** and into the **Call** module. Then a corresponding rising transition is sent to the **Rin** of the stage labelled ss4. After the **ss4** stage holds data the stage sends a rising transition to the **d** input of the **Call** module. Immediately there is a rising transition on the output labelled **d1**.

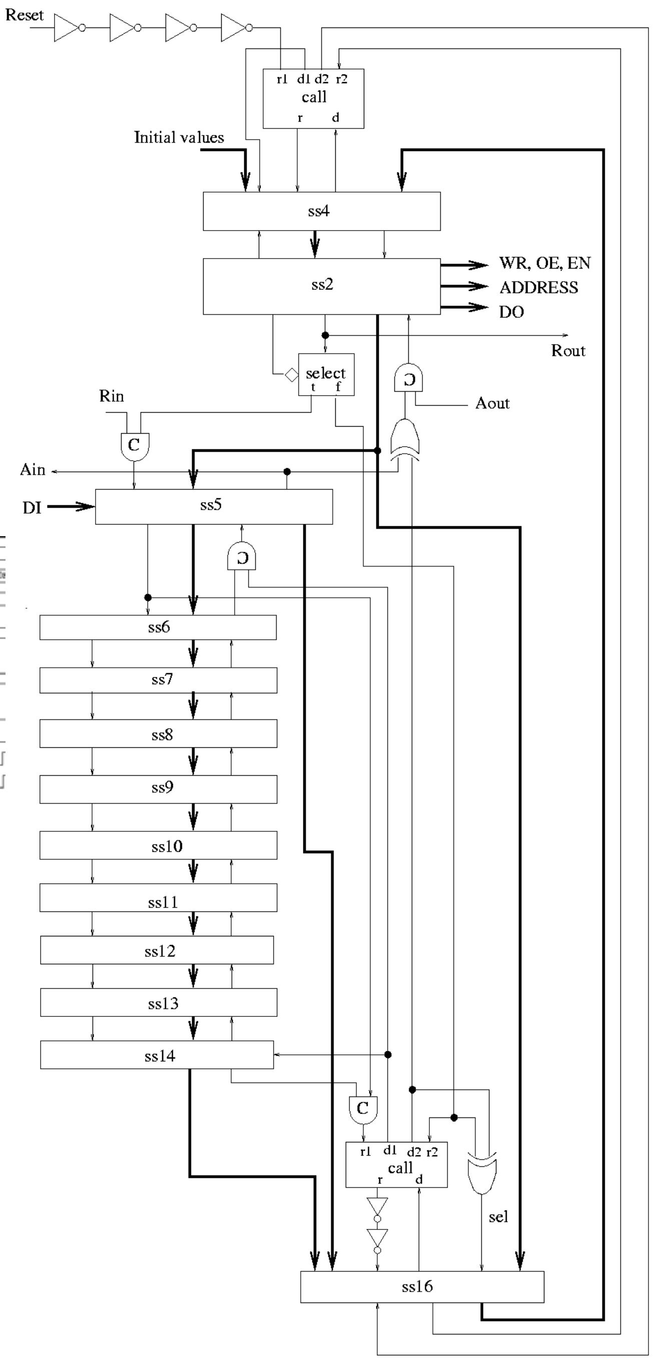

Fig. 4 A two-phase structural Stump processor

This rising transition goes through the Toggle and shows on its dotted output. Then the rising transition goes through the **XOR** and becomes a falling transition. This falling transition goes through the ss4 stage and appears on the output labelled **d1**. Then the falling transition flows into the **Toggle**. A rising transition is present on the non-dotted output. This rising transition is used as a control signal of the multiplexer inside **ss4** stage, which is used to switch the latches connected to the initial values or the connections from the stage labelled **ss16**. The non-dotted output of the **Toggle** will remain at logic **Hi** except when Reset goes **Lo**.

The second change is to remove the **XOR** whose inputs are connected to the signals labelled **r2** and **d2** of the **Call** module at the bottom of the circuit shown in Fig. 4. A wire must be connected between the signal labelled **r2** and the control signal of the multiplexer inside stage **ss16**. This change is due to the 4-phase protocol being used and a rising transition and a falling transition are present on the **r2** input of the **Call** module. If this change is not made there will be two 4-phase events on the output of the **XOR**. A 4-phase structural Stump processor circuit is shown in Fig. 6.

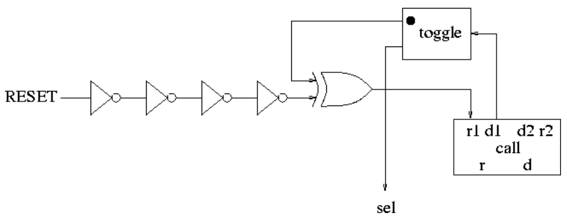

Fig. 5 A change is required for the 4-phase handshaking protocol

Table 8 shows the number of the transistors and the run time of the 240 instructions, the throughput, the latency and the energy of the synthesized Stump processors using different control latch circuits.

Table 8 The performance of the Stump processors

Circuit Name	Transis-tors (piece)	Run Time (μs)	Through-put (KIPS)	Latency (ns)	Energy (fj)
2-p	171166	220.0	1090.7	687.2	1530.57
4-p Simple	170428	378.9	633.4	1148.4	1329.71
4-p Semi	170922	411.8	582.8	1249.1	1514.44
4-p Fully	171028	237.3	1011.3	738.0	1109.13
4-p Long	171350	240.9	996.1	748.1	1067.50

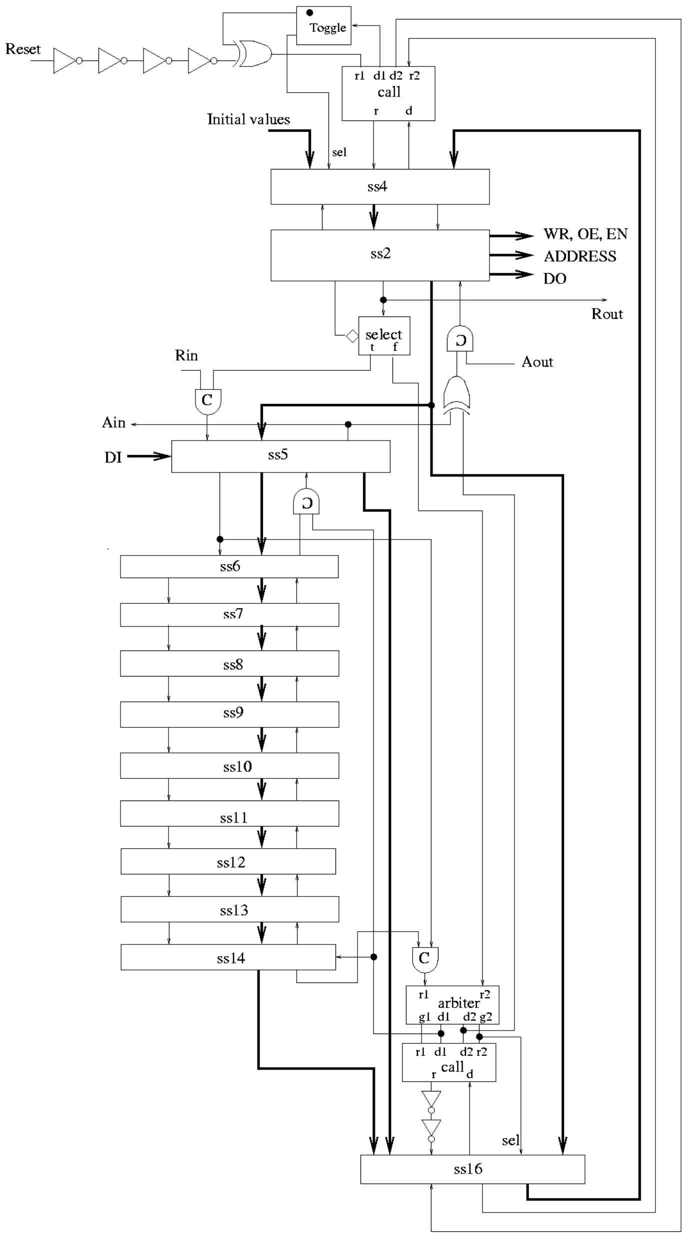

Fig. 6 A 4-phase structural Stump processor

5 Experimental results

A loop program with twelve instructions were used to test the synthesized Stump processors shown Figures 4 and 6. **Rout** of the synthesized Stump processor is connected to the **Aout** input directly. Memory data and a request or acknowledge was sent to the Stump processor by the test program when the Stump processor fetches instructions or reads data. The run time is the time difference between the 321th **request-out** signal on **Rout** and the first **request-out** signal on **Rout**. This is the time for 240 instructions to be executed. The energy information was obtained from PowerMill simulation for 12 instructions.

Circuit	2-p	4-p simple	4-p semi	4-p fully	4-p long
Energy	1530.57	1329.71	1514.44	1109.13	1067.50

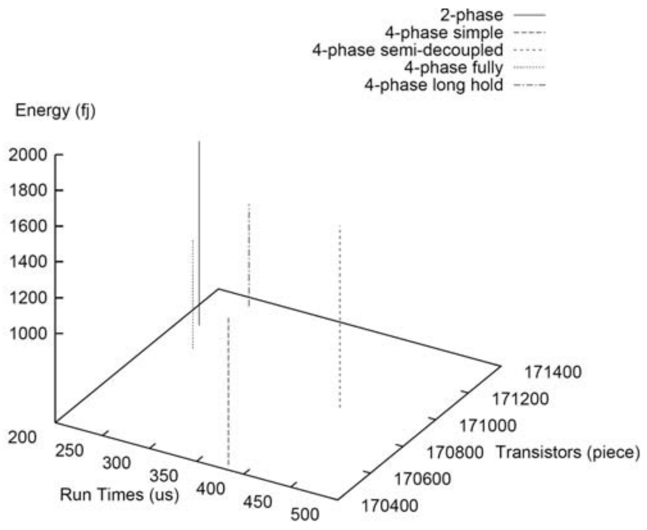

Fig. 7 The performance of the synthesized Stump processors

As shown in Figures 7, 8 and 9 the fast circuit is still the 2-phase implementation. The two-phase circuit also has high throughput and low latency. The 4-phase fully decoupled circuit is the fastest of the different 4-phase circuits. It also has high throughput. The 4-phase simple circuit is the cheapest circuit but it is slow. The synthesized Stump processor using the 4-phase semi-decoupled circuit is slowest and has low throughput as well as high latency. The 4-phase long hold circuit has low power consumption but high cost. However, its speed is not slow.

6 Return to zero problems

There is no **Arbiter** in the circuit shown in Figure 4 and the simulations ran correctly on the leapfrog simulator. When 4-phase control circuits are used within each stage and 4-phase control modules are also used for the connections between stages it is necessary to put an **Arbiter** at the front of the **Call** module which is connect to the **ss16** stage as shown in Figure 6.

Because the **ss5** stage is connected to both **ss6** and **ss16** stages, when the rising transition flows into the **Call** module a rising transition is sent to **ss6** stage. When stage **ss6** holds data the acknowledge rising transition is through **d** and **d1** to **ss14** and the **C** gate which is connected to the **Aout** of the **ss5** stage. The acknowledge rising transition goes through **ss14**, **ss13**, ... etc. When the **ss5** stage receives the transition the falling transition is sent out to the **ss6** stage and **C** gate which is connected to the **r1** of the **Call** module. At this moment another rising transition may be sent to the **r2** input of the **Call** module.

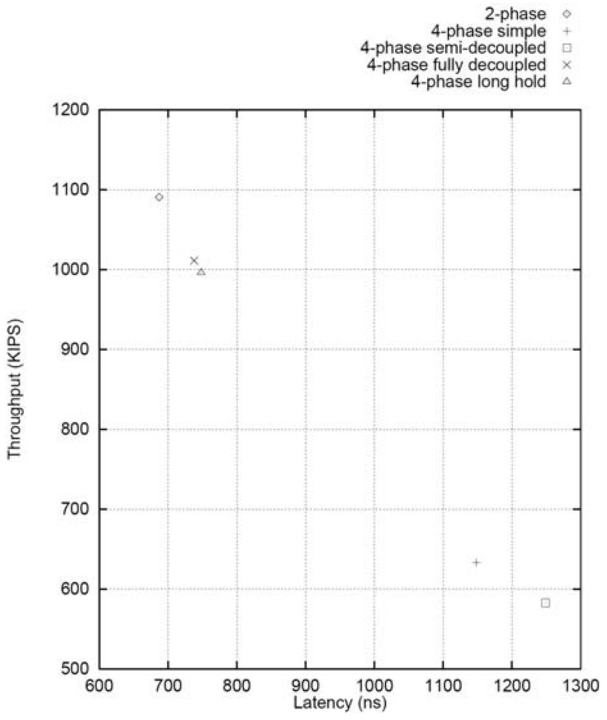

Fig. 8 The latency and throughput of the synthesized Stump processors

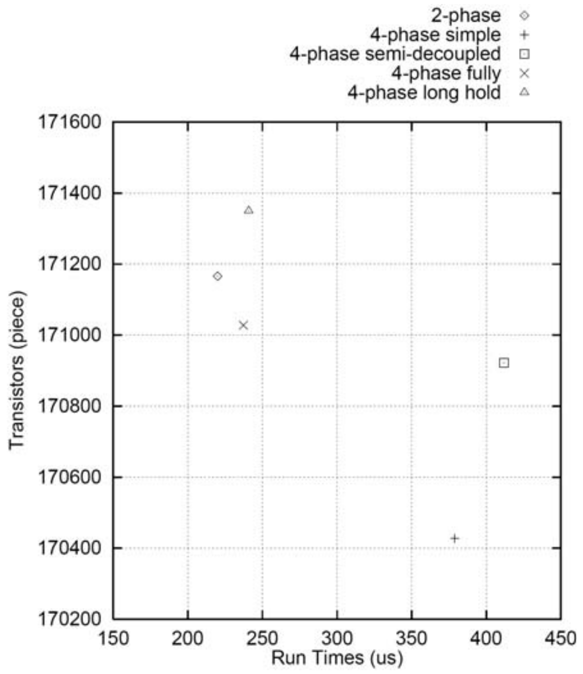

Fig. 9 The performance of the synthesized Stump processors

The previous calling sequence for **r1** is not finished yet. The rising transition appears on the **r2** input will cause the **Call** module to go wrong. Each 4-phase event consists of a rising transition and a falling transition. Both transitions must flow through the control path. The time from **r1** going **Hi** until **d1** going **Hi** and going **Lo** may be longer than that in two-phase counterpart. To avoid such circuit errors an **Arbiter** is required to connected to the **Call** module at the front of the **ss16** stage.

Because only the initial event needs to go through **r1** on the **Call** module at the front of the **ss4** stage, at the other times only the event sent from the **ss16** stage will go to **r2** on that **Call** module. An Arbiter is not necessary for that **Call** module.

A 4-phase **Call** circuit [13] is shown in Figure 10. Two 3-input asymmetric **C**-gates are used to send the rising transitions and falling transitions of **D** to **D1** or **D2** depending whether the event on the **R** output was generated from **R1** or **R2**. A 2-input asymmetric **C**-gate is used to send the rising transition and falling transition of **R1** or **R2** to the **R** output. Initially **R1** and **R2** are **low**. When a rising transition arrives on the **R1** input a rising transition is sent out from the **R** output. After a rising transition is received on the **D** input this causes a rising transition to be sent out from the output of the upper 3-input asymmetric **C**-gate on **D1**. If a rising transition arrives on the **R2** input before **R1** falls, **R2**, **R** and **D** will be at logical **'1'** and therefore a rising transition is sent out from the **D2** output in error. No transition corresponding to that on **R2** is sent out from the **R** output, and a deadlock may happen. A 4-phase Arbiter costs about 40 transistors. The easy way to remove this problem is to connect an Arbiter at the front of the **Call** circuit. Alternatively, a more complex **Call** circuit can be used.

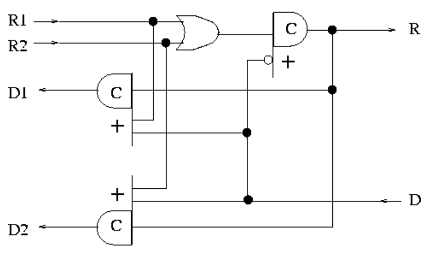

Fig. 10 A 4-phase call circuit

When the fully decoupled 4-phase control circuit is applied the simulation of the processor stopped due to the **Select** module. The reason is that the **Rout** of the **ss2** stage sends out a rising transition to the Select module and the **Select** module will deliver this rising transition to one of its two outputs depending on the boolean value which is produced from the **ss2** stage.

If the boolean value is a logic **'0'** the rising transition is sent to the output labelled **f**. Then the rising transition goes to the **r2** input of the Arbiter. The **Arbiter** sends the rising transition out from **g2** and the rising transition flows into the **r2** input of the **Call** module. Finally the rising transition arrives at the **Rin** input of the **ss16** stage.

When the **ss16** stage holds data the acknowledge rising transition will be sent out. It will go through the **d** and **d2** of the **Call** module as well as an **XOR** and a **C** gate and arrive on **Aout** of the **ss2** stage.

After the **ss2** stage receives a rising transition from **Aout** the data in the **ss2** stage may be changed. If the data is changed the boolean value of the **Select** module is also changed. Therefore, the falling transition sent from **Rout** of the **ss2** stage may be delivered to the wrong place. The simulation shows that the falling transition which should be sent to the **false** path was sent to the **true** path. This caused the simulation to fail.

A 4-phase **Select** module which remembers the boolean value at the time of sending the rising transition is required. After using this **Select** module the simulations ran successfully.

Due to the fact that there is no return to zero in a two-phase protocol, a two-phase protocol may be a better choice than a 4-phase protocol for communication between stages.

7 Optimization

If two stages only contain assignments, the first is the only input stage to the second, and the second is the only output stage from the first, then these two stages can be merged. If the first one is called **stage A** and the second is called **stage B**, the procedure to merge the two stages is as follows:

1. Connect the outputs of each latch to the sources of its inputs inside **stage A** and remove these latches as well as redundant connections.

2. Remove the outputs of **stage A** and the inputs of **stage B** from the device list.

3. Remove the control circuits of **stage A**.

4. Remove the interconnections between **stage A** and **stage B** from the network device list.

However, one more condition must be satisfied if the simple 4-phase or the semi-decoupled 4-phase control circuits are applied. This is that if the first stage is an output stage of the second then the two stages cannot be merged. The 2-phase Stump processor shown in Figure 4 can be merged as shown in Figure 11. The performance of the Stump processors after merge is shown in Table 9.

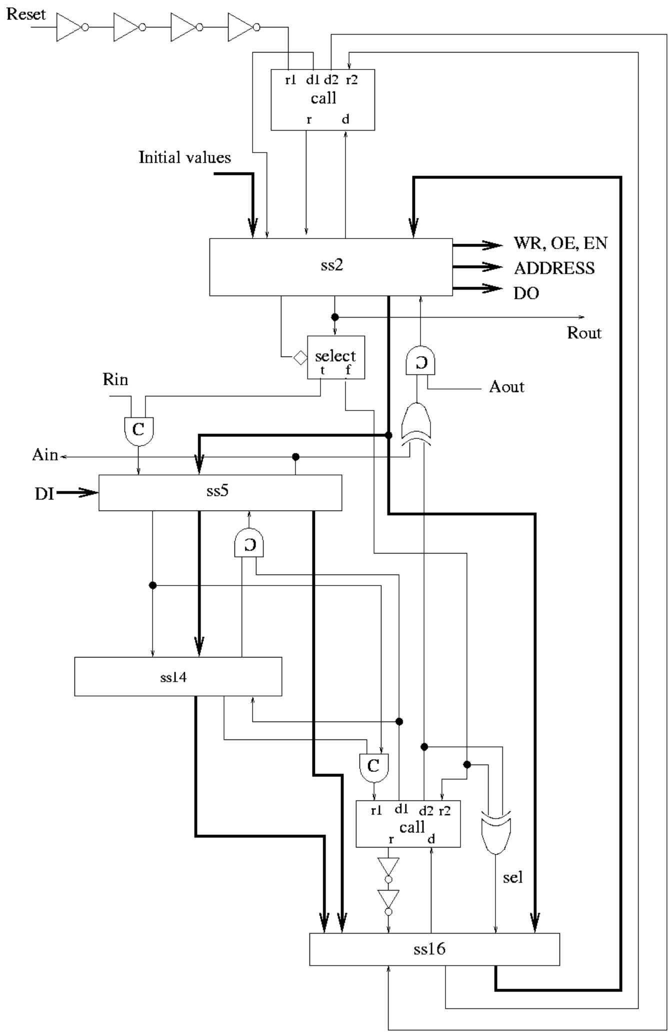

Fig. 11 A merged two-phase structural Stump processor

As shown in Figures 12, 13 and 14 the performance of the synthesized circuits is improved. However, the speed of the merged 4-phase fully decoupled and long hold Stump processors are decreased. The long hold Stump processors has low power consumption. The two-phase one is the fastest.

Table 9 The performance of the Stump processors after merge(1)

Circuit Name	Transistors (piece)	Run Time (μs)	Throughput (KIPS)	Latency (ns)	Energy (fj)
2-phase	137226	159.8	1502.3	499.2	522.37
4-p Simple	140016	289.8	828.1	878.7	558.77
4-p Semi	140206	295.9	811.2	898.2	613.95
4-p Fully	137088	262.2	915.3	801.2	417.46
4-p Long	137292	266.8	899.5	811.5	404.62

Circuit	2-phase	4-p simple	4-p semi	4-p fully	4-p long
Energy	522.37	558.77	613.95	417.46	404.62

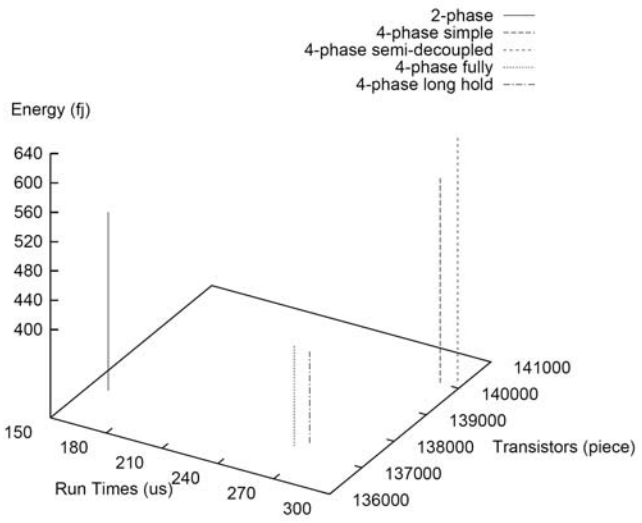

Fig. 12 The power information of the merged Stump processors

The synthesized Stump processors using the 4-phase simple control latch circuit and the 4-phase semi-decoupled control circuit have 5 stages. The synthesized Stump processors using the 4-phase fully decoupled control circuit and the 4-phase long hold control circuit only have 4 stages. For the comparison Table 10 shows the performance of the 5-stage fully decoupled Stump processor and the 5-stage long hold Stump processor.

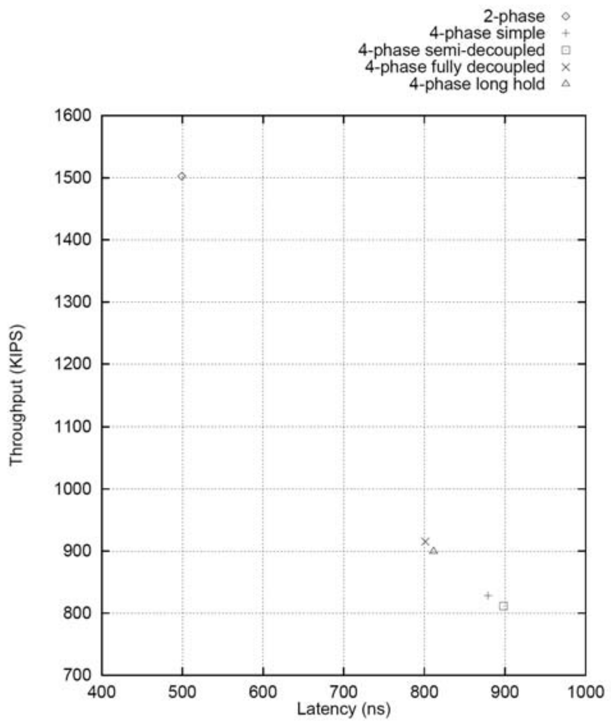

Fig. 13 The latency and throughput of the merged Stump processors

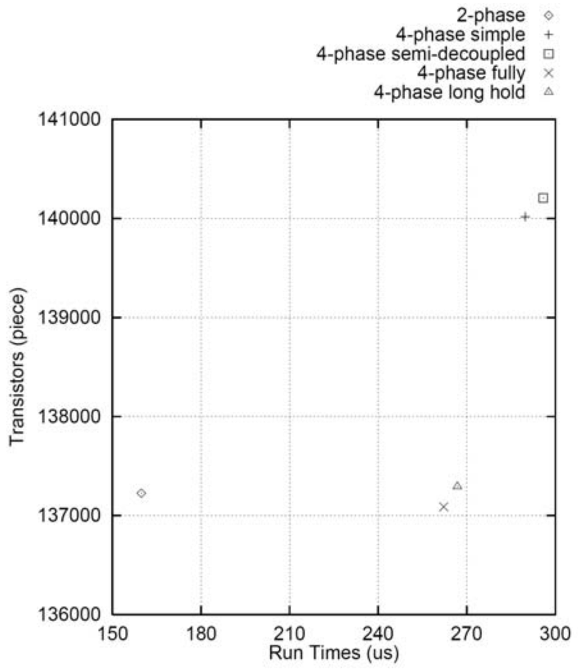

Fig. 14 The run time of the merged Stump processor

As shown in Figures 15 and 16 the speed of the merged 4-phase Stump processors using the simple and semi-decoupled control circuits is faster than the original synthesized Stump processors.

Circuit	4-p simple	4-p semi	4-p fully	4-p long	4-p fully 5 stages	4-p long 5 staages
Energy	558.7	613.95	417.46	404.62	481.38	442.08

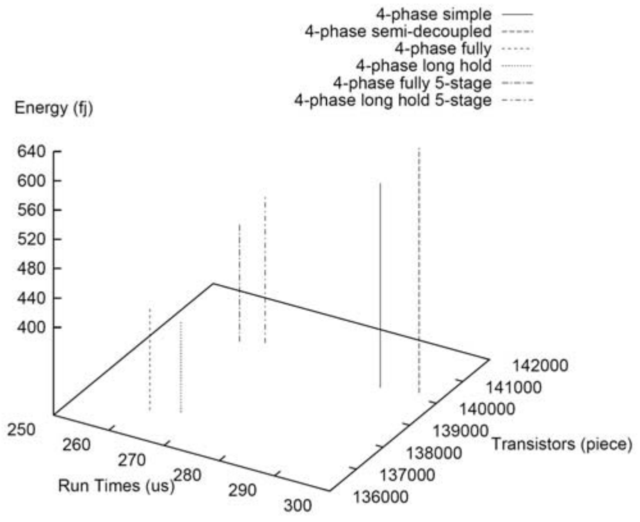

Fig. 15 The power information of the 4-phase merged Stump processors

Table 10 The performance of the Stump processors after merge(2)

Circuit Name	Transistors (piece)	Run Time (μs)	Throughput (KIPS)	Latency (ns)	Energy (fj)
2-p	137226	159.8	1502.3	499.2	522.37
4-p Simple	140016	289.8	828.1	878.7	558.77
4-p Semi	140206	295.9	811.2	898.2	613.95
4-p Fully	137088	262.2	915.3	801.2	417.46
4-p Long	137292	266.8	899.5	811.5	404.62
4-p Fully 5	140252	263.2	911.8	801.8	481.38
4-p Long 5	140466	266.8	899.5	812.0	442.08

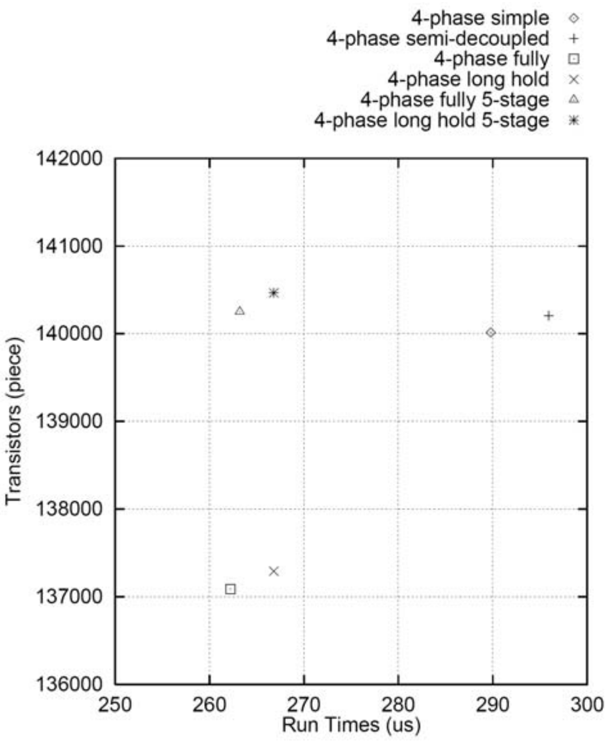

Fig. 16 The performance of the 4-phase merged Stump processors

However, the speed of the merged 4-phase Stump processors using the fullly decoupled and long hold control circuits is slower than the speed of the original synthesized Stump processors. There is no much difference between the 5-stage and 4-stage Stump processors using the fully decoupled and long hold control circuits in speed.

The 5-stage Stump processors using the fully decoupled and long hold control circuits require more transistors than the merged 4-phase Stump processors using the simple and semi-decoupled control circuits. Anyway the merged 4-phase Stump processors are cheaper than the original synthesized Stump processors and have lower power consumption.

If a modification is made in the behavioural description the processor can be further merged as shown in Figure 17. If some statements are added after the corresponding **ELSE** of the **false** of **Select** module, the 4-phase Stump processors using the simple latch control circuit can also be further merged. The 4-phase Stump processors using the simple and semi-decoupled latch control circuits can be further merged as shown in Figure 18.

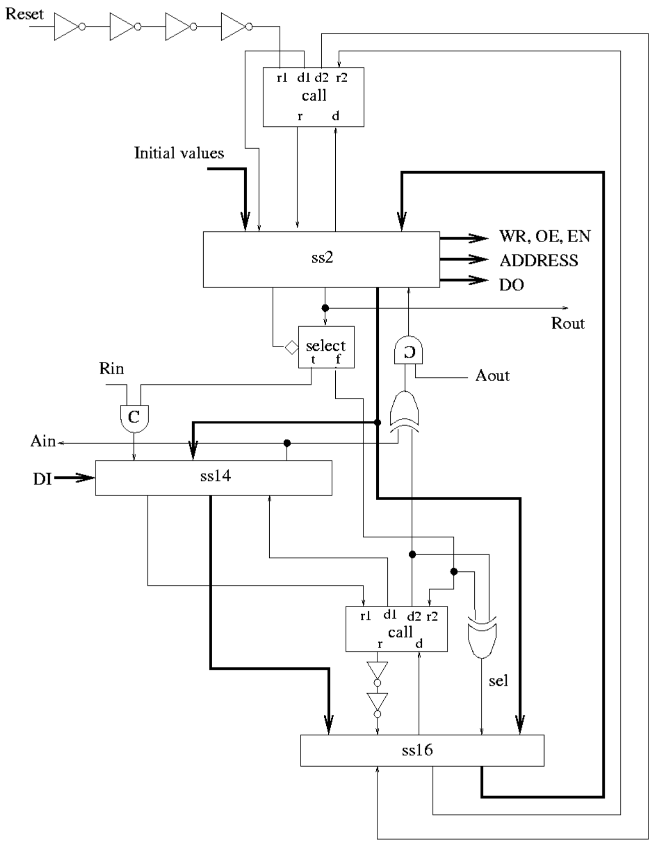

Fig. 17 A further merged two-phase structural Stump processor

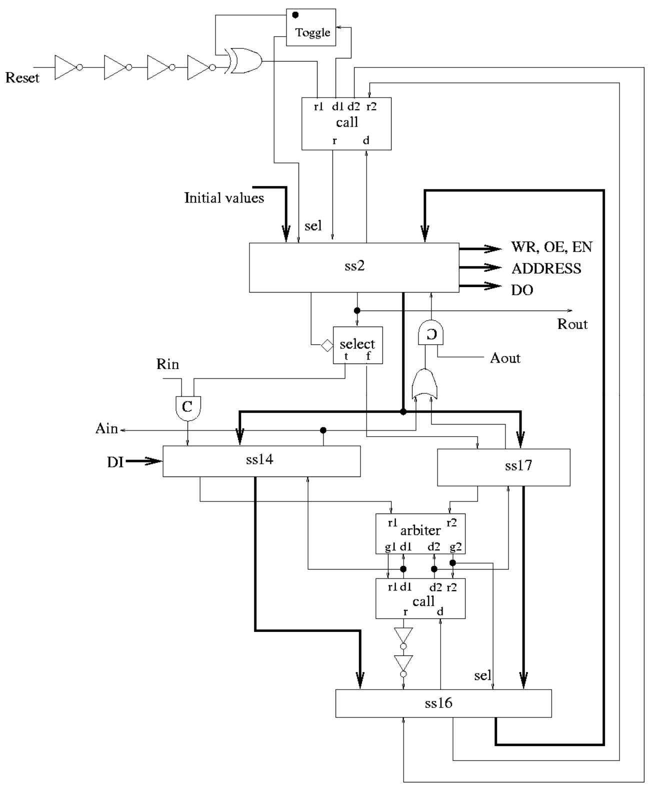

Fig. 18 A merged 4-phase structural Stump processor

The run time of the 2-phase Stump processor is about 155.1 μs. The 4-phase long hold circuit gives low power consumption.

Table 11 The performance of the Stump processors after optimization

Circuit Name	Transistors (piece)	Run Time (μs)	Throughput (KIPS)	Latency (ns)	Energy (fj)
2-phase before	137226	159.8	1502.3	499.2	522.37
4-p Simple before	140016	289.8	828.1	878.7	558.77
4-p Fully before	137088	262.2	915.3	801.2	417.46
4-p Long before	137292	266.8	899.5	811.5	404.62
2-phase after	134286	155.1	1547.9	484.5	461.59
4-p Simple after	137128	240.4	998.1	751.4	434.85
4-p Fully after	134152	244.6	981.1	764.5	365.19
4-p Long after	134342	248.5	965.6	776.7	362.33

The performance of these further merged Stump processors are shown in Table 11, Figures 19, 20 and 21.

Circuit	2-p merged	4-p fully merged	2-p modified	4-p simple modified	4-p fully Modified	4-p long modified
Energy	522.37	417.46	461.59	434.85	365.19	362.33

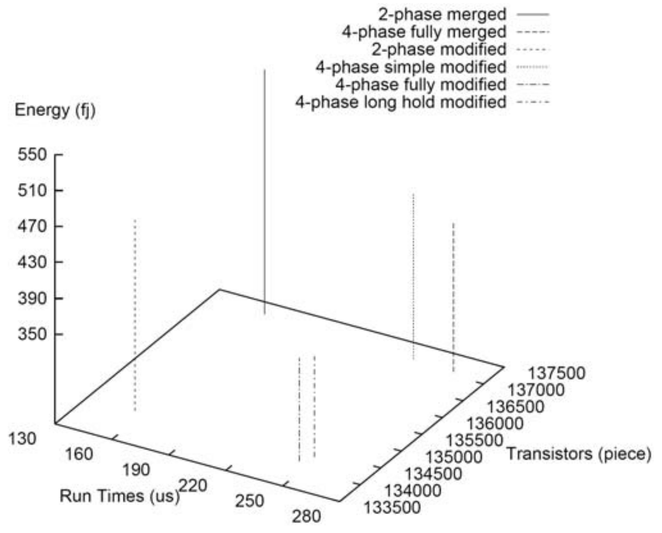

Fig. 19 The power information of the further merged Stump processors

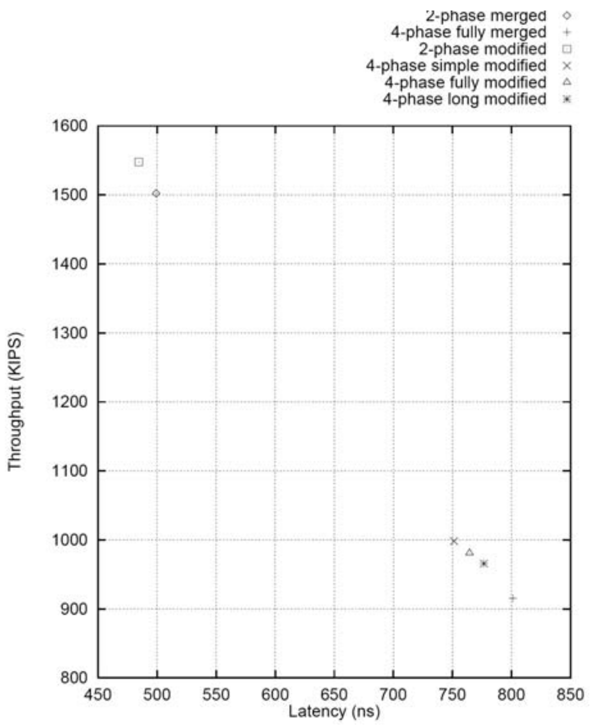

Fig. 20 The latency and throughput of the further merged Stump processors

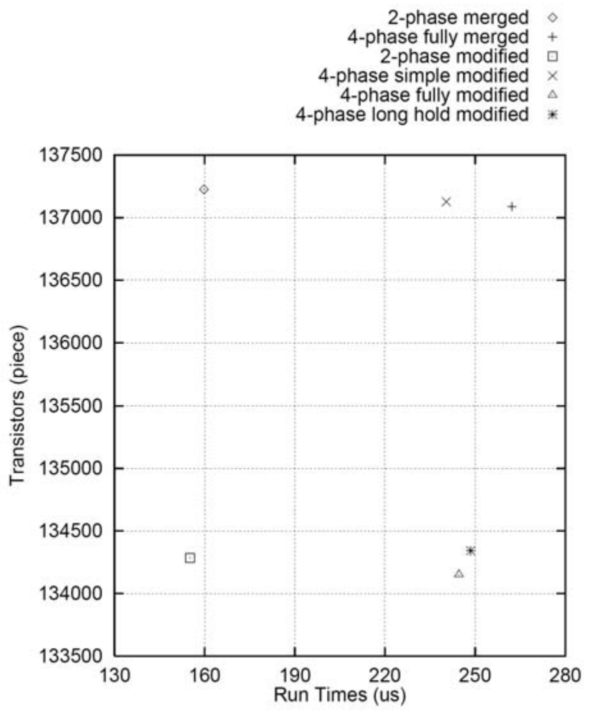

Fig. 21 The run time of the simple Stump processor

8 Conclusion

This paper has presented some engineering work on developing a micropipeline Stump processor. The experimental results show that the fastest speed is 1090.7 KIPS for the 2-phase synthesized Stump processor. The lowest power consumption is 1067.5 fj for the Stump processor with the long hold 4-phase latch control circuits. The Stump processor using the 4-phase simple latch control circuit has the lowest the transistor count.

The 2-phase circuits have good performance in speed. This is due to the rising and falling transitions of the 4-phase circuits following the same routes. Asymmetric delays with fast reset circuit can be applied to improve the performance.

The fastest speed is 1.55 MIPS for the two-phase synthesized processor and
the lowest power consumption is 362.33 fj for the synthesized 4-phase long hold processor.

References:

[1] Tan, S.-Y., Furber, S.B., Yen, W.-F., "The Design of an Asynchronous VHDL Synthesizer", Proceedings of the Design, Automation and Test in Europe Conference 1998 (DATE98), Paris, Feb. 1998, pp. 44-51.

[2] Furber, S.B., and and Liu, J., "Dynamic Logic in Four-Phase Micropipelines", Async'96, Aizu-Wakamatsu, Japan, Mar 18-21 1996.

[3] Furber, S.B., Day, P., "Four-Phase Micropipeline Latch Control Circuits", IEEE Trans. on VLSI Systems, vol. 4 no. 2, Jun. 1996 pp. 247-253.

[4] Sacker, M., Brown, A.D., Rushton, A.J., Wilson, P.R., "A Behavioral Synthesis System for Asynchronous Circuits", IEEE Trans. on VLSI Systems, vol. 12 no. 9, Sep. 2004, pp. 978-994.

[5] Furber, S.B., "Computing without Clocks: Micropipelining the ARM Processor", in "Asynchronous Digital Circuit Design" edited by G. Birtwistle and A. Davis, Springer Verlag, pp.211-262.

[6] Sutherland, I. E., "Micropipelines", The 1988 Turing Award Lecture, Communications of the ACM, Vol. 32, No. 6, January 1989, pp. 720-738.

[7] S.-Y. Tan, W.-T. Huang, "The Design of an Asynchronous Blocksorter", Proceedings of the 12th International Conference on Networking, VLSI and Signal Processing (ICNVS '10) (WSEAS Cooperating Conference), University of Cambridge, UK, 20-22 February 2010, pp. 73-78.

[8] J. Carlsson, K. Palmkvist, and L. Wanhammar, "Synchronous Design Flow for Globally Asynchronous Locally Synchronous Systems", Proceedings of the 10th WSEAS International Conference on CIRCUITS, Vouliagmeni, Athens, Greece, July 10-12, 2006, pp. 64-69.

[9]A. N. Ismailoglu, M. Askar, "Verification of Delay Insensitivity in Bit-Level Pipelined Dual-Rail Threshold Logic Adders", 7th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Cambridge, UK, February 20-22, 2008

[10]A.Vasilescu, "Algebraic model for the intercommunicating hardware components behaviour", 12th WSEAS International Conference on COMPUTERS, Heraklion, Greece, July 23-25, 2008, pp. 241-246.

[11] S.-Y. Tan, W.-T. Huang, "A VHDL-based design methodology for asynchronous circuits", WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS, Vol. 9, Issue 5, May 2010, pp. 315-324.

[12] P.B. Endecott, S.B. Furber, "Behavioural Modelling of Asynchronous Systems for Power and Performance Analysis", Proceedings of PATMOS'98, Lyngby, Denmark, 6-9 October 1998.

[13] J. Liu, "Arithmetic and Control Components for an Asynchronous System", PhD Thesis, Dept. of Computer Science, University of Manchester, 1997.

[14] S.-Y. Tan and W.-T. Huang, "The Design of a simple asynchronous processor", Proceedings of the 12th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering (MMACTEE '10), Timisoara, Romania, October 21-23, 2010, pp. 165-170.

[15] S.-Y. Tan and W.-T. Huang, "The Design of sharing resources for asynchronous systems", Proceedings of the 12th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering (MMACTEE '10), Timisoara, Romania, October 21-23, 2010, pp. 171-176.