A VHDL-based design methodology for asynchronous circuits

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 blocksorter. 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. In summary, a design environment for asynchronous circuits has been established based upon the micropipeline style and VHDL, a standard hardware description language.

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

1 Introduction

Asynchronous design has potential advantages over synchronous design [23][24][25], 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. 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 [6]. It converts the behavioural VHDL into structural VHDL and Verilog based on micropipelines had been published [6]. 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 blocksorter using the sytnthesizer and evaluate the experimental results.

Section 2 introduces some asynchronous logic techniques. Section 3 describes the synthesis flow which is used by the synthesizer. Section 4 introduces 4-phase event-driven Logic modules and 4-phase control circuits. Examples of the "while loop" control circuits is given in Section 5. The blocksorter design will be presented in Section 6. Section 7 will present experimental results. Finally, Section 8 will give a short conclusion and suggestions for further research.

2 Asynchronous logic

Asynchronous design does not have a clock to govern the timing of state changes. Storage components for holding internal states and data within asynchronous systems work at different times depending on the preceding and successor circuits. For reliable operation, an asynchronous circuit must be free from critical races and unstable states and liveness checking must be undertaken. If a system contains n storage elements there are 2^n possible states. One approach is to arrange that all storage elements work together to capture data at the same time. Such a system could have a central clock and storage elements that capture internal state signals and data on the rising edge of the clock, the new states being derived from combinational logical circuits that read the old state and inputs. Furber gave a more clear definition and relation between asynchronous and synchronous designs as follows:

> It is a common misconception to view asynchronous design as a single alternative to synchronous design. It is more accurate to view synchronousdesign as a special case representing a single point in a multi-dimensional asynchronous design space [15].

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 [16]. Asynchronous logic may be used to implement systems with lower power dissipation.

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 [17].

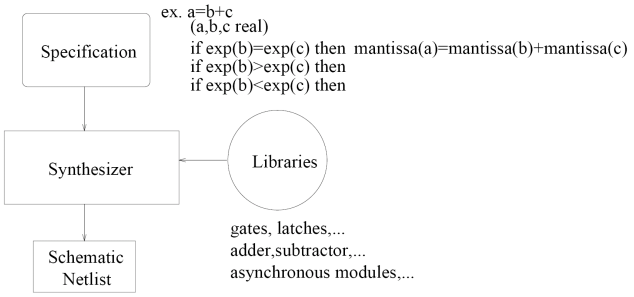

Fig. 1. A block diagram of a synthesiser.

3 The synthesis

Fig. 1 shows a block diagram of a synthesiser. A synthesizer performs translations or compilations from specifications into the implementations of circuits.The specification describes the computation function of synthesized circuits. The libraries contain the standard gates and modules which are used to construct the synthesized circuits by the synthesizer. The specification may be denoted by languages, graphics or mathematics. Language based CSP [5][7] and Occam [4], graphic based Petri nets [8] and STGs [9], and mathematic based FSMs and Algebras [10] have been used previously for synthesizing asynchronous circuits. An alternative, IEEE standard language, VHDL [14], is used for the specifications in this work.

The creation of component libraries is another key issue for synthesis. The libraries used here contain VHDL models and various views of 2-phase and 4-phase modules and standard gates which were created either for this research or were used to develop AMULET1 [1] and AMULET2e [2]. Structural VHDL and Verilog are chosen to be the output of the synthesizer.

The synthesis procedure begins by partitioning descriptions into several pipeline stages, the number depending on the concurrency and other properties of the description. Then we produce the circuits for each stage. Appropriate control circuits are automatically added into the stages. Finally, we produce the interconnections between the stages.

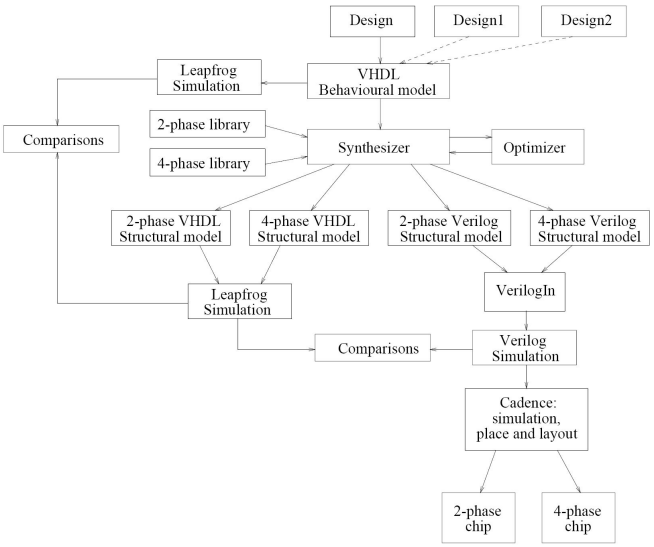

Fig. 2. Design flow.

Design flow

Fig. 2. shows a potential design flow. The designs are described in behavioural VHDL descriptions. This VHDL descriptions may be simulated using a VHDL simulator (Leapfrog). Then the descriptions can be synthesized into two-phase and four-phase VHDL structural models and Verilog structural models respectively. The structural VHDL models may also be simulated using a VHDL simulator. The simulation results may be compared with the previous behavioural simulation results to verify the implementation against its specification.

Fig. 3 shows the interface between the synthesizer and other tools, LARD [18], Yellow [19] and Balsa [20][21]. The design may be also described and simulated in LARD. The simulation results may be compared with the VHDL results to check the correctness of the input descriptions and the output implementation.

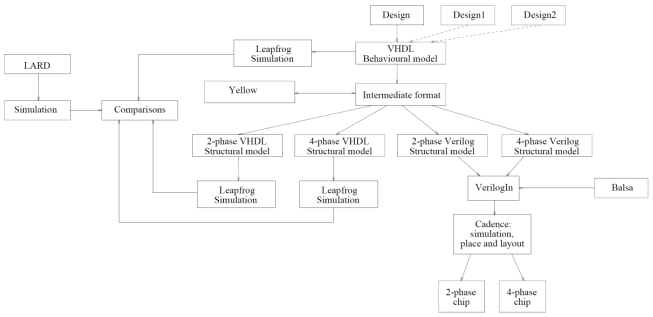

Fig. 3. Interfacing to other asynchronous description languages.

4 Four-phase Micropipelines

A "four-phase bundled data convention" is a communication system where a 4-phase handshaking protocol is used and an arbitrary number of data wires are treated as a bundle together with the request signal wire. 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. 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-**.

The data signals can use a traditional data representation which is similar to that used in synchronous circuits, such as the ***8-4-2-1 code ...*** etc. Fig. 4 illustrates two kinds of four phase signalling, the **'early'** mode and the **'broad'** mode [11]. The **'early'** mode (Fig. 4(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. 4(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. 5. 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 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.

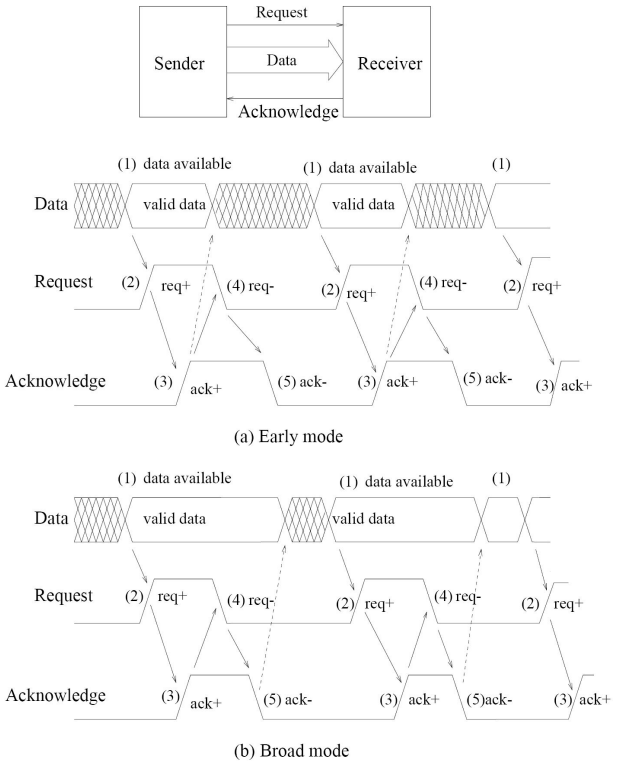

Fig. 4. 4-phase bundled data convention

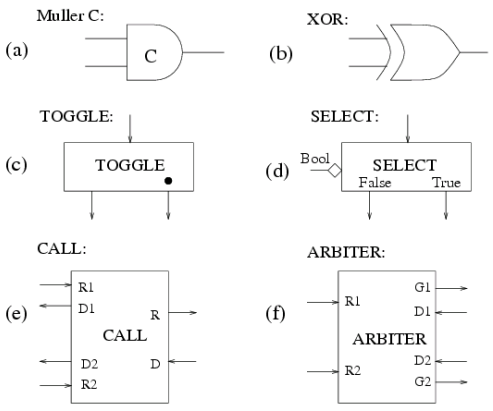

Fig. 5. Various event-driven logic modules.

AMULET1 was implemented using a 2-phase micropipeline design style. However, AMULET2 and AMULET2e [2][12] use a 4-phase micropipeline design style to improve their performance. 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 [13]. They use the 4-phase bundled data convention.

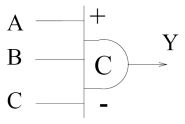

Fig. 6. Asymmetric C-gate notation.

The asymmetric C-gate notation shown in Fig. 6 indicates that an input controls both edges of the output when it is connected to the main body of the gate; it controls only the rising edge when connected to the extension marked **'+'**, and it controls only the falling edge when connected to the extension marked '-'.

As shown in Fig. 7 **Rin+** must wait for **Aout-** and **Rin-** must wait for **Aout+** to proceed to **lt+** and **lt-** when the simple 4-phase latch control circuit is used, this may lead to poor performance. However, the cost is very low.

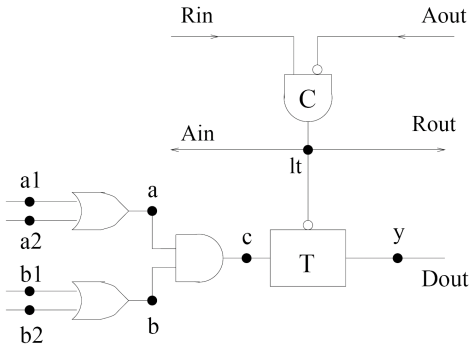

Fig. 7. A simple 4-phase latch control circuit.

The semi-decoupled and fully decoupled latch control circuits may be employed to solve this problem. In the semi-decoupled latch control circuit (see Fig. 8) **Rin+** does not need to wait for **Aout-** and can proceed to **lt+** after **Rout-**. But Rin- still needs to wait for **Aout+** to proceed to **lt-.** The fully decoupled latch control circuit (see Fig. 10) is much faster than simple and semi-decoupled control circuits. When it is used **Rin+** can proceed to **lt+** after **Rout-** and **Rin-** can through to **Ain-** after **Ain+** shortly. For some applications, it may be necessary that the latch holds the data stable until **Aout** goes low. The long hold 4-phase control circuit shown in Fig. 10 can serve this purpose.

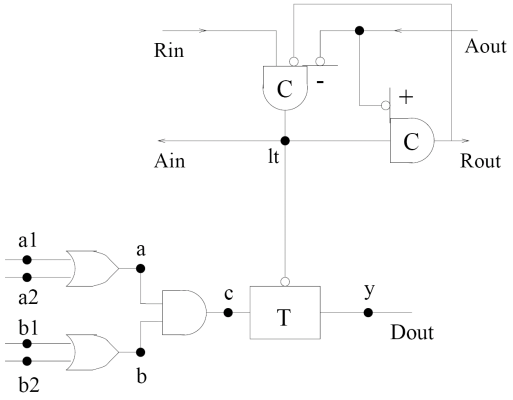

Fig. 8. A semi-decoupled 4-phase control circuit .

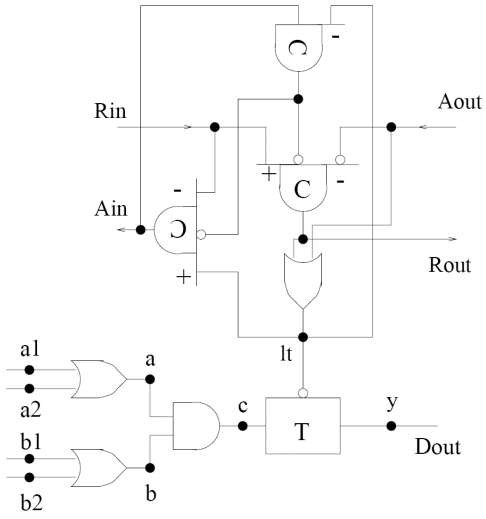

Fig. 9. A fully decoupled 4-phase control circuit.

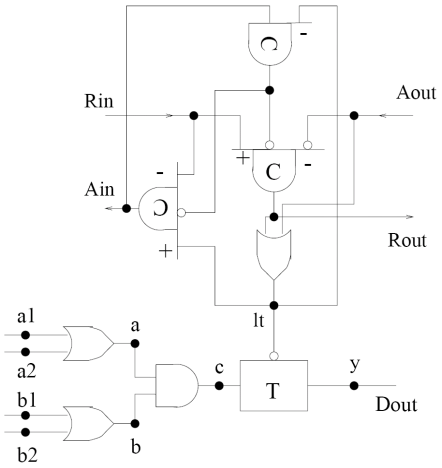

Fig. 10. A long hold 4-phase control circuit.

5 while loop examples

Fig. 11 shows the schematic of the output of the synthesizer for the following input description.

```
y := a; w := "00101";
while ( w >= "00001" )  loop
     y := y + "00001";
     w := w - "00001";
end loop;
e <= y;
```

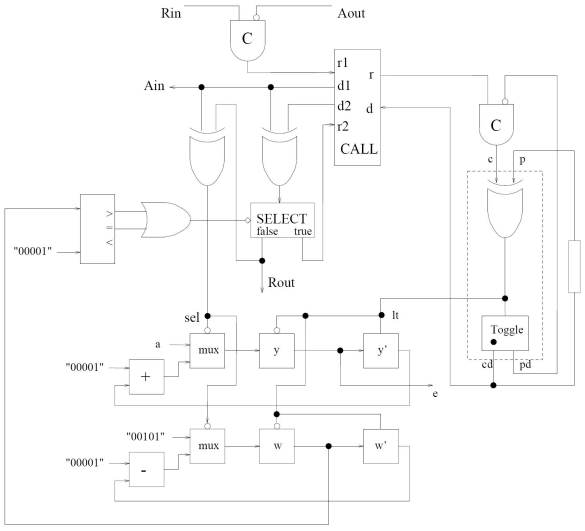

Fig. 11. A schematic for a while loop example.

Fig. 12 shows a 4-phase while loop control circuit which can perform the following loop operation. The circuit continuously performs the loop operation if the greater output of the comparator remains true.

```
w = A;
while ( w > "00101" ) loop
     w = w - "00001";
end loop;
```

In the 4-phase handshaking protocol the request and acknowledge signals will return to their inactive state to start a new cycle. Activating the reset of related components is a simple and fast way to deliver a falling transition from the request input to the request output. Fig. 13 shows a 4-phase while loop control circuit where the reset is activated if the **Rin** is **Low** and the **Aout** is **High**. The circuit can also perform the same loop operation as shown in Fig. 12. It will terminate the loop operation if the **'equal'** output of the comparator becomes true. The number of the inverters between the output of the asymmetric C-gate and the junction **p1** must be odd.

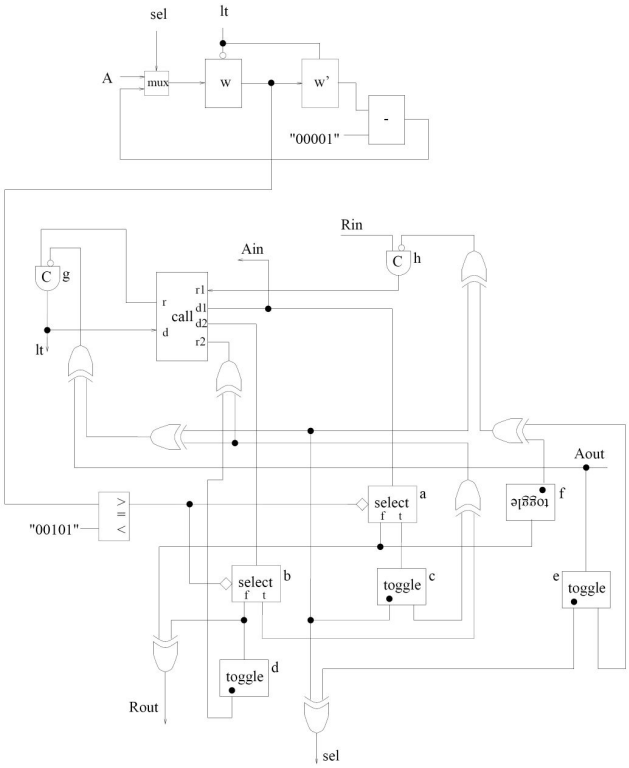

Fig. 12. A 4-phase while loop control circuit.

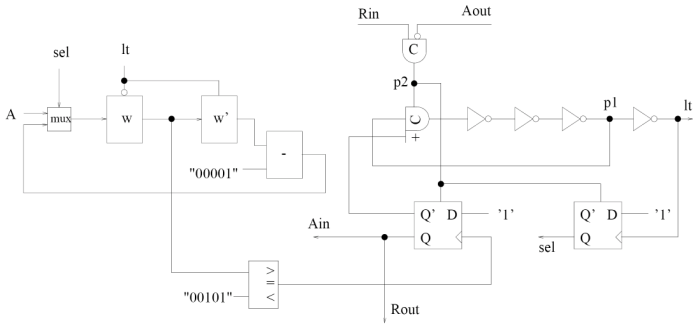

Fig. 13. Another 4-phase while loop control circuit.

6 The blocksorter

The following description shows a simple handshaking control with a simple computation. When the Reset signal is logic '0' it is in a reset loop to clear the **AIN** and **Rout** signals. While the Reset signal is logic '1' the operating cycle is (1) waiting for an **Rin** transition; (2) computing; (3) sending **AIN** and **Rout** signals; (4) waiting for an **Aout** transition.

```
while ( RESET= '0' ) loop Qrin := '0'; Qaout := '0'; Sro := '0';
      ROUT <= sro;  AIN <='0';
end loop;
while ( Rin = Qrin ) loop  end loop;  Qrin := Rin;
if a > "00101" then   b := a - "00010";  else  b := a + "00011";
end if;
E <= b; AIN <= RIN; sro := not sro;  Rout <= sro;
while ( Aout = Qaout ) loop  end loop;  Qaout := Aout;
```

A different synthesis procedure is applied to generate the circuits. Therefore, checking whether the loop statement contains handshaking controls is required before the circuit synthesis. In this case only the computation part of the description is required to convert into circuits and then a suitable control circuit is added. In the previous paper we discussed the blocksorter [22]. In the blocksorter [3] example a computation loop contains handshaking controls. A 2-phase blocksorter can be described as follows:

```
while ( RESET = '0' ) loop
      Qrin := '0';  Qaout := '0';   Sro := '0';
      ROUT <= sro;  AIN <='0';
end loop;
while ( Rin = Qrin ) loop   end loop;   Qrin := Rin;
y := a;    w := "00101";
AIN <= RIN;
while ( w >= "00001" ) loop
      while ( Rin = Qrin ) loop end loop;  Qrin := Rin;
      x := a;   AIN <= RIN;
      if x > y then  b := y;  else  b := x;  end if;   E <= b;
      sro := not sro;   Rout <= sro;
    if x > y then  y := x;  else  y := y;  end if;
      while ( Aout = Qaout ) loop  end loop;
Qaout := Aout;
      w := w - "00001";
end loop;
b := y;   E <= b;
sro := not sro;  Rout <= sro;
while ( Aout=Qaout ) loop end loop; Qaout := Aout;
```

To get a correct simulation the description contains some handshaking control loop and initial assignments. However, this causes the synthesis work to be more difficult. If the reset loop, the handshaking control loop and initial assignments are put into procedures there is no effect on the simulation. But it is easier to remove these procedures and to convert the description into the circuits. The previous blocksorter description can be re-written as follows:

```
ENTITY blocksort1 IS
  PORT ( RIN, AOUT : IN  MVL;
AIN, ROUT : OUT  MVL;
       A : IN MVL_VECTOR ( 4 DOWNTO 0 );
       E : OUT MVL_VECTOR ( 4 DOWNTO 0 );
       RESET : IN MVL );
END blocksort1;
architecture BEHAVIORAL of blocksort1 is
begin
 U1: process
  variable Qrin, Qaout : MVL;
  variable sro : MVL;
  variable X,Y,B,W:MVL_VECTOR (4 DOWNTO 0);
PROCEDURE  req_in (Qrin: INOUT MVL) IS
   BEGIN  while ( Rin = Qrin ) loop wait for 1 ns;
end loop;   Qrin := Rin;   END req_in;
PROCEDURE  req_out (sro: INOUT MVL) IS
   BEGIN  sro := not sro;   Rout <= sro;   END req_out;
PROCEDURE  ack_out (Qaout: INOUT MVL) IS
   BEGIN  while ( Aout = Qaout ) loop   wait for 1 ns;
   end loop;   Qaout := Aout;  END ack_out;
PROCEDURE  ack_in IS
   BEGIN  AIN <= RIN;  END ack_in;
PROCEDURE  reset_q (Qrin, Qaout, sro: INOUT MVL) IS
   BEGIN  while ( RESET = '0' ) loop   wait for 1 ns;
      Qrin := '0';  Qaout := '0';  sro := '0';   wait for 1 ns;
      ROUT <= sro;   AIN <='0';     end loop;   END reset_q;
begin
  wait for 1 ns;   reset_q(Qrin, Qaout, sro);    wait for 1 ns;
  req_in(Qrin);  y := a;  w := "00101";  wait for 1 ns;   ack_in;
  while ( w >= "00001" ) loop
    wait for 1 ns;  req_in(Qrin);  x := a;  wait for 1 ns;  ack_in;
    if x > y then  b := y;  else  b := x;  end if;
    E <= b;   wait for 1 ns;  req_out(sro);   wait for 1 ns;
    if x > y then  y := x;   else   y := y;   end if;
  ack_out(Qaout);   w := w - "00001";
end loop;
 wait for 1 ns;   b := y;  E <= b;  wait for 1 ns;
req_out(sro);    ack_out(Qaout);
end process U1;
end BEHAVIORAL;
```

We may count the number of appearances of the ***req_in***, ***ack_in***, ***req_out*** and ***ack_out*** signals in the description. A ***req_in*** and an ***ack_in*** appear before the while loop. A ***req_in***, an ***ack_in***, a ***req_out*** and an ***ack_out*** appear inside the while loop. A ***req_out*** and an ***ack_out*** appear after the while loop. The while loop contains four handshaking signals. In the previous simple while loop example the loop contains no handshaking signal. The handshaking signal number inside a while loop can be used to recognize for generating different while loop circuits. The handshaking signal numbers of the blocksorter description are shown as follows:

```
req_in = 1
        y := a;     w := "00101";
ack_in = 1
        while ( w >= "00001" ) loop
req_in = 2
           x := a;
ack_in = 2
           if x > y then  b := y;  else  b := x;  end if;
           E <= b;
req_out = 1
           if x > y then  y := x;  else   y := y;  end if;
ack_out = 1
           w := w - "00001";
        end loop;
        b := y;
        E <= b;
req_out = 2
ack_out = 2
```

The synthesizer read the blocksorter behavioural description and produced the structural VHDL files of the circuit shown in Fig. 14. The synthesized 2-phase blocksorter was simulated using the test program and the correct result was obtained. 6-stage blocksorters connected together were also simulated using the test program and the expected results was obtained. A 4-phase blocksorter circuit is shown in Fig. 15. "**Return to zero**" is required in the 4-phase handshaking protocol. The circuit enclosed in the dashed line and labelled *M3* is used to ensure that the latches which are connected to the control signal **lt y** still hold data and that the signal connected to the input labelled **d** of the Call module becomes **logical '0'** when the output **r** of the Call module becomes **logical '0'**. The latches will be clear when the signal labelled **CX** becomes **logical '1'** and wait for holding the next data. The circuit enclosed in the dashed line and labelled *M1* is used to ensure that the control signal connected to the input **r1** of the Arbiter module becomes **logical '0'** when the signal labelled **RX** becomes **logical '1'**. This ensures that the complete cycle signal of the input **d** of the Call module is sent to the output **d1** and then the request of **RX** can be sent into the Call module.

The circuit also ensures that the signal labelled **A1** stays **logical '1'** if both signals labelled **RX** and *RA* are **logical '1'** or the output **d1** of the Call module is **logical '1'**. When the data is held and the output **d2** of the Call module becomes **logical '1'** it ensures that *Rin-* goes through and the signal *RA* becomes **logical '0'** as well as the signal labelled **A1** becomes **logical '0'**. The circuit enclosed in the dashed line and labelled *M2* is used to ensure that the request of the signal **RX** can be held until the corresponding completion signal is received.

The circuits labelled *M1* and *M3* are required if the simple latch control circuit is applied. The circuits labelled *M1*, *M2* and *M3* are required if the semi-decoupled control circuit is applied. The circuits labelled *M2* and *M3* are required if the fully-decoupled control circuit and the long-hold control circuit are applied. The circuit enclosed in the dashed line and labelled *M4* is to produce the select signal of the multiplexer at the front of the low active transparent latch **w**. The signal **sel w** goes **high** when the signal labelled **A** becomes **high** and it goes **low** when the signal labelled **B** becomes **high**.

Table 1 shows the number of the transistors and the run time of the 120 sets of data, the throughput, the latency and the energy of the synthesized blocksorter circuit. Table 2 shows the number of the transistors and the run time of the 120 sets of data, the throughput and the latency of the 6-stage synthesized blocksorter circuits were connected in series.

Table 1. The performance of the synthesized blocksorter.

Circuit Name	Transistors (piece)	Run Time (μs)	Throughput (MHz)	Latency (ns)	Energy (fj)
2-phase	2878	15.53	7.80	261.3	10.78
4-p Simple	2598	19.74	6.19	228.0	12.13
4-p Semi	2766	22.33	5.41	264.4	14.23
4-p Fully	2782	19.29	6.27	220.0	14.72
4-p Long	2878	20.78	5.82	226.3	13.65

7 Experimental results

A blocksorter was used to test the synthesizer. The properties of these synthesized circuits are shown in Tables 1 and 2. 120 sets of test data were sent to the synthesized blocksorter for the Leapfrog simulation. **Rout** of the blocksorter is connected to the **Aout** input directly. A new request was sent to the blocksorter when the test program received a transition from the **Ain** output.

Table 2. The performance of the 6-stage synthesized blocksorters.

Circuit Name	Transistors (piece)	Run Time (μs)	Throughput (MHz)	Latency (ns)
2-phase	17268	16.70	7.71	1273.1
4-p Simple	15576	21.84	5.81	1328.1
4-p Semi	16596	25.0	5.09	1575.7
4-p Fully	16692	20.48	6.25	1353.2
4-p Long	17268	24.24	5.24	1534.9

The run time is the time difference between the **request-out** signal of the 120th data on **Rout** and the **request-in** signal of the first data on **Rin**. The energy information was obtained from PowerMill simulation for twelve sets of test data. As shown in Figures 16, 17 and 18 the 2-phase blocksorter is the fastest and its power consumption is the lowest.

The 2-phase blocksorter also has high throughput and high latency. But the cost is high. The blocksorter using the 4-phase simple latch control circuit has a small number of transistors. Therefore, its power consumption is low. The blocksorter using the 4-phase fully decoupled latch control circuit is the fastest of the different 4-phase circuits. It also has low latency.

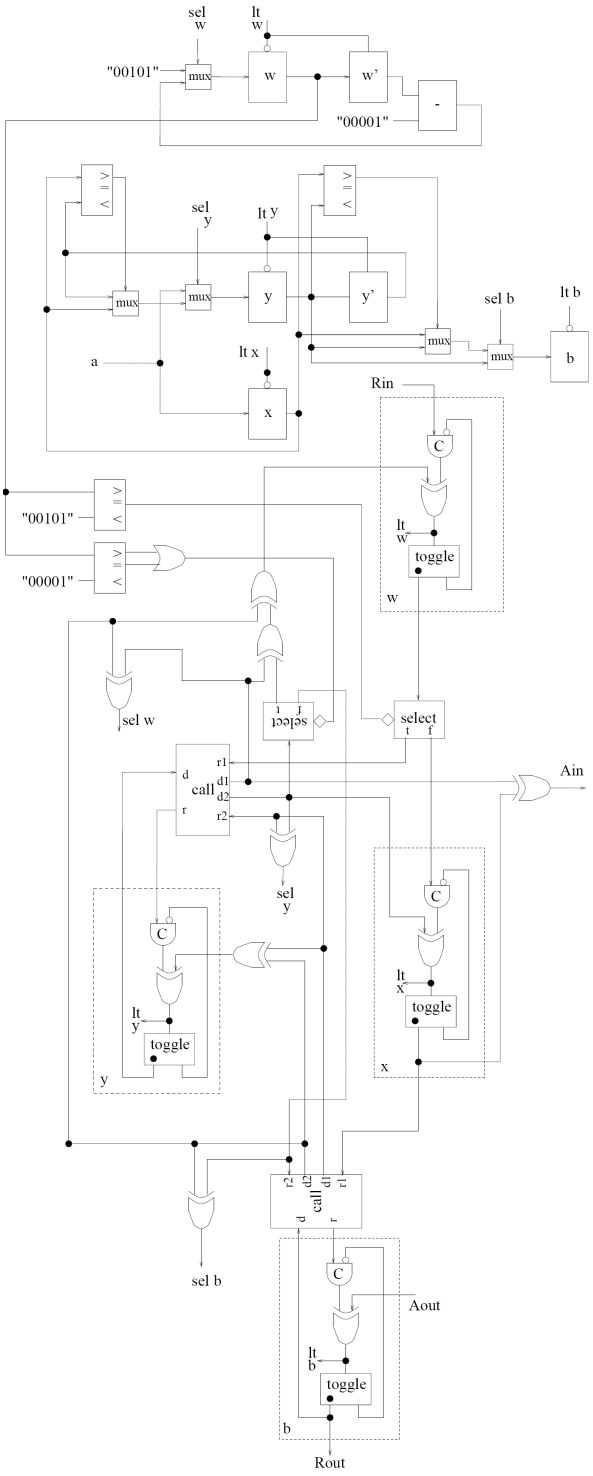

Fig. 14. The synthesized 2-phase Blocksorter.

This means that the time between the **request-in** signal on **Rin** and the **request-out** signal on **Rout** is short. However, a waiting time is required in asynchronous design if the circuit is busy. The blocksorter using the 4-phase semi-decoupled control circuit does not have good performance here. The only advantage of the blocksorter using the 4-phase long hold control circuit is that it is fast.

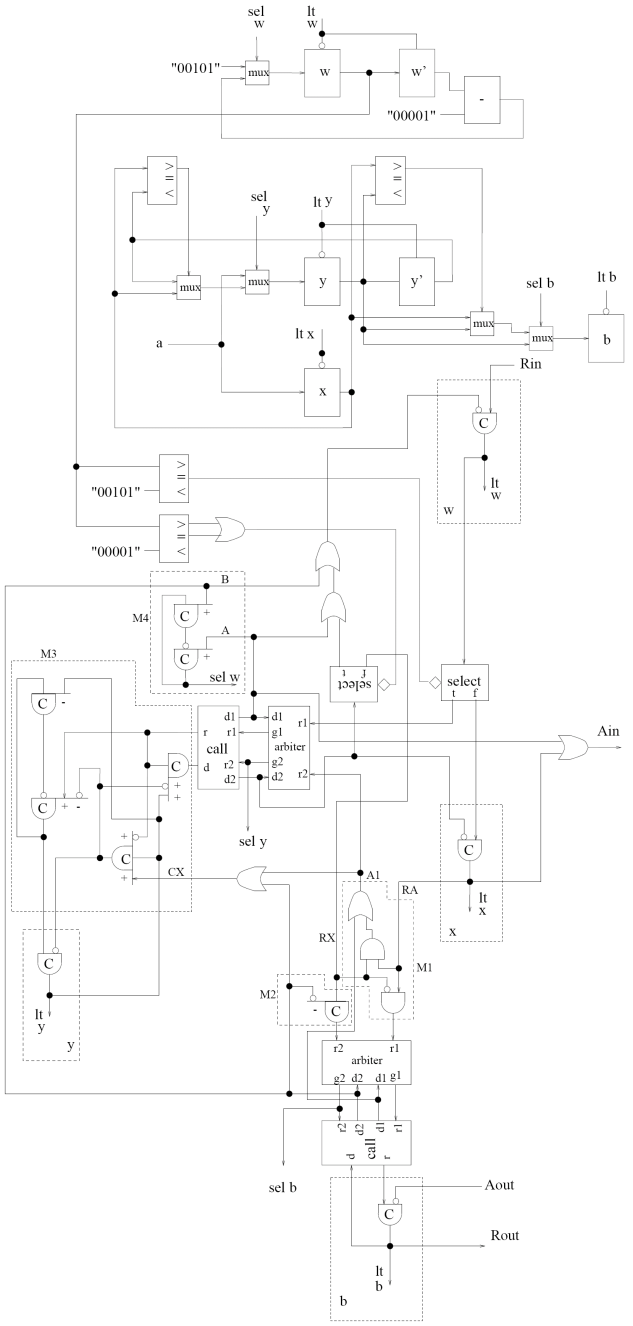

Fig. 15.The synthesized 4-phase Blocksorter.

150 sets of test data were sent to the synthesized floating point adder/subtractor for the Leapfrog simulation. **Rout** of the floating point adder/subtractor is connected to the **Aout** input directly. A new request was sent to the floating point adder/subtractor when the test program received a transition from the **Ain** output. The run time is the time difference between the **request-out** signal of the 150th data on **Rout** and the **request-in** signal of the first data on **Rin**. The energy information was obtained from PowerMill simulation for fifteen sets of test data.

Circuit	2-phase	4-p simple	4-p semi	4-p fully	4-p long
Energy	10.78	12.13	14.23	14.72	13.65

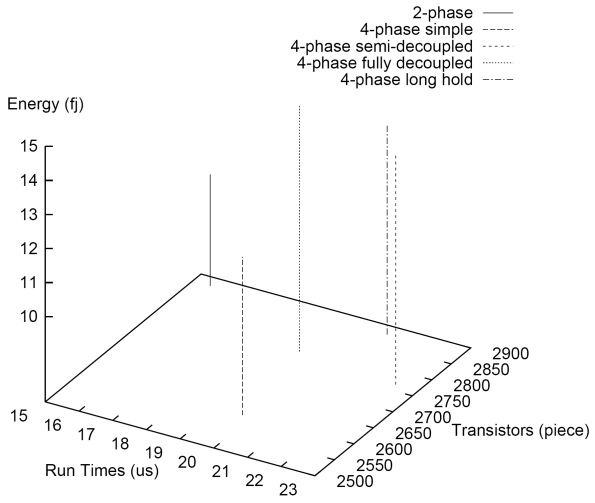

Fig. 16. The performance of the synthesized blocksorter circuits.

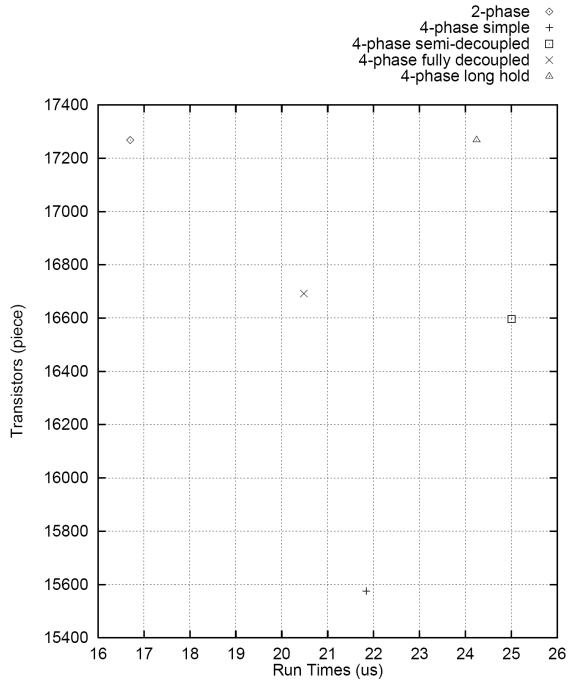

Fig. 17. The run time and the transistor number of the synthesized blocksorter circuits.

The fast circuit is the 2-phase implementation. The two-phase circuit also has high throughput, high latency and low power consumption. For this circuit the 2-phase design is not especially expensive, unlike the blocksorter. The floating point adder/subtractor using the 4-phase simple latch control circuit is the cheapest and has low latency.

The 4-phase semi-decoupled circuit is slowest and has low throughput. Again the 4-phase fully decoupled circuit is the fastest of the different 4-phase circuits. It also has high throughput. The 4-phase long hold circuit has high cost.

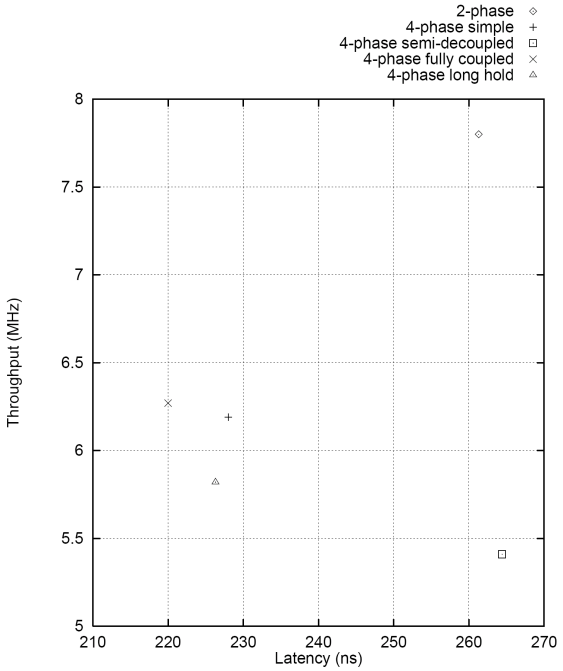

Fig. 18. The throughput and latency of the synthesized blocksorter circuits.

8 Conclusion

This paper has presented some engineering work on developing a micropipeline blocksorter. In order to synthesize correct circuits the input description must be correct first. The simulation mechanism of VHDL is able to assist in the discovery of potential design errors at an early stage. Though the nature of the work is mainly engineering, there are some significant new insights gained in the course of the work.

The experimental results show that the fastest speed is 7.80 MHz and the lowest power consumption is 10.78 fj for the 2-phase synthesized Blocksorter. Blocksorter 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.

A difficult engineering problem with the synthesis method presented in this paper is the insertion of delays in the control path. In some cases the delay may reduce the performance of the circuits. For example, the 4-phase fully decoupled circuits have high speed performance, but if unsuitable delay

is added on the control path the performance of the circuits becomes poor. A dual-rail technique could be added as an option in the data path implementation to synthesize fully delay-insensitive pipelines, thereby avoiding this problem.

References:

[1] Furber, S. B., Day, P., Garside, J.D., Paver, N.C. and Woods, J.V., "A Micropipelined ARM", 1993 International Conference on VLSI, Grenoble, France, September 6-10, 1993.

[2] Furber, S.B., Day, P., Garside, J.D., Paver, N.C. and Temple, S., "AMULET2e", EMSYS'96 - OMI Sixth Annual Conference, Berlin, 23-25 September 1993, IOS Press ISBN 90 5199 300 5.

[3] van Berkel, K., Handshake Circuits -- An Asynchronous Architecture for VLSI programming, 1993, Cambridge Univ. Press.

[4] Inmos, Occam Programming Manual, 1983.

[5] Hoare, C. A. R., "Communicating Sequential Processes", Prentice-Hall, 1985.

[6] 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.

[7] Hoare, C. A. R., "Communicating Sequential Processes", Communications of the ACM", Vol. 21, No. 8, August 1978, pp. 666-677.

[8] Peterson, J. L., "Petri Nets", Computing Surveys", Vol. 9, No. 3, September 1977, pp. 223-253.

[9] Chu, T. A., "Synthesis of Self-timed VLSI Circuits from Graph-Theoretic Specifications", PhD Thesis, MIT/LCS/TR-393, MIT Laboratory for Computer Science, June 1987.

[10] Josephs, M. B., Udding, J. T., "Delay-Insensitive Circuits: An Algebraic Approach to their Design", Lecture Notes in Computer Science, edited by J. C. M. Baeten and J. W. Klop, Vol. 458, Springer-Verlag, August 1990, pp. 342-366.

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

[12] Furber, S.B., Garside, J.D., Temple, S., Liu, J., Day, P., and Paver, N.C., "AMULET2e: An Asynchronous Embedded Controller", Proc. of Async '97, Apr. 1997, pp. 290-299.

[13] 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.

[14] 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.

[15] Furber, S.B., "An Introduction to Asynchronous Design: The Return of Asynchronous Logic", Department of Computer Science, University of Manchester, UK, 1995.

[16] 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.

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

[18] Endecott, P.B. and Furber, S.B., "Modelling and Simulation of Asynchronous Systems using the LARD Hardware Description Language", Proceedings of the 12th European Simulation Multiconference, Manchester, June 1998, pp. 39-43.

[19] Barringer, H. and Fellows, D., Gough, G., Williams, A., "Abstract Modelling of Asynchronous Micropipeline Systems using Rainbow", International Conference on Hardware Description Languages and their Applications, edited by C. D. Kloos and E. Cerny, IFIP, 20-25 April 1997, Spain, pp. 285-304.

[20] Bardsley, A. and Edwards, D., "Compiling the language Balsa to Delay Insensitive Hardware", International Conference on Hardware Description Languages and their Applications, edited by C. D. Kloos and E. Cerny, IFIP, 20-25 April 1997, Spain, pp. 89-91.

[21] Bardsley, A., "Implementing Balsa Handshake Circuits", PhD Thesis, Dept. of Computer Science, University of Manchester, 2000.

[22] S.-Y. Tan, W.-T. Huang, "The Design of an Asynchronous Blocksorter", VLSI and Signal Processing (ICNVS '10), Feb. 2010, pp.73-78.

[23] 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.

[24]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

[25]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.