Synthesis Tool for Asynchronous Circuits Based on PCFB and PCHB

[1]K. Saleh, [1]H. Pedram, [1]M. Naderi, [1]M. H. Shafiaabadi, [1]H. Kalantari, [2]A. Farhoodfar
[1]Department of Computer Eng. and IT, Amirkabir University of Technology
Hafez Ave., Tehran 15875, Iran
{k.saleh, pedram, naderi, shafiaabadi, hkalantari}@ce.aut.ac.ir
[2]Applied Micro Circuit Corporation, Kanata, ON, Canada
afarhood@amcc.com

Abstract

This paper introduces a synthesis tool for template-based synthesis of asynchronous circuits. The tool, named Template Synthesizer (TSYN) and it is a part of a complete asynchronous design flow named Persia. This tool transforms a behavioral description of a circuit to a sized transistor net-list. The input behavioral description must fit into some previously known templates. These templates are general enough to allow implementation of almost all circuit blocks. Finally, an asynchronous Reed-Solomon Decoder is synthesized using this tool.

Keywords: Asynchronous, Synthesis

1. Introduction

One of the most important problems in popularity of asynchronous circuits is complex design and synthesis methods and lack of automatic tools. Many different asynchronous circuit design methods are proposed [1,2,3] but the synthesis method developed in Caltech [4] has shown better capability to gain higher performance, lower power and more robustness in different environmental conditions [5,6]. This synthesis method is based on QDI timing model.

Because of the complexity of the Caltech's synthesis method, using templates for reducing the design cycle is considered as a solution [7,8]. TSYN, a part of ***Persia*** [9], is proposed to automate synthesizing specific templates. Because Verilog is used as the HDL in all levels of abstraction, it is not required to design customized simulation tool. The standard Verilog simulators powered by a library of macros and PLI routines can be used to simulate asynchronous circuits [10,11]. Verilog is chosen because of availability of *fork* and *join* constructs, which are critical for describing asynchronous circuits, and very good support for switch-level simulation.

In Caltech synthesis method, the circuit is modeled at the behavioral level using a CSP [12] derived language. Then the initial design is decomposed to smaller CSP blocks and this procedure is continued to get the simple elements. The elements generated by this method are modeled as processes communicating to each other using input and output channels. Communications between different processes are via handshaking and in most cases include data transfer as well. This can be viewed as *read* or *write* operations.

The circuit should be decomposed to basic elements complying PCFB (Pre-Charge logic Full-Buffer) or PCHB (Pre-Charge logic Half-Buffer) templates before using this tool. These templates imply that all *read* operations must precede *write* operations. It normally includes evaluation of functions, where some input data is read and the computed value is put on outputs.

The Caltech synthesis method is described in section 2. Section 3 explains PCFB and PCHB templates. The synthesis tool behavior is presented in sections 4 through 6. A sample circuit and its synthesized equivalent are presented in section 7 and finally, experimental results are shown in section 8.

2. Synthesis Method

After decomposition, each block is fed into synthesis tool. In the first step, this behavioral description is transformed into a set of production Rules (PR). Each PR describes pull-up or pull-down network of a CMOS circuit.

In the second step, Operator Reduction, some PRs are replaced by standard gates such as AND, OR, NAND, etc. Also state-holding elements are added to the outputs where needed. The output of this step composed of a set of gates and PRs is fed into the third step of the synthesis tool, which should produce the final circuit. In the last step, PRs are converted into a network of NMOS and PMOS switches and transistor sizes are determined. The overview of the synthesis flow is depicted in figure 1.

In the following sections we describe the structure of PCFB and PCHB templates and then three steps of TSYN are described.

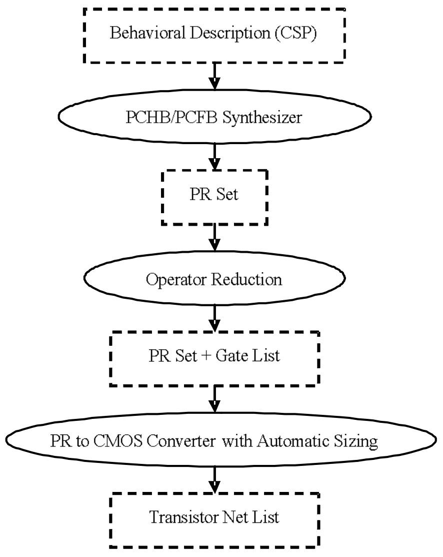

Figure 1: The synthesis flow

3. PCFB and PCHB templates

Structure of PCFB and PCHB templates are shown in figure 2 and figure 3. Input and outputs are dual-rail coded [13], so the circuit can determine the validity of the input or output value.

As it can be seen in these figures, after validation of the input, the output is computed and after validation of the output, *InputAck* signal is activated. The output value is neutralized upon receiving the *OutputAck* signal, and the *InputAck* returns to zero after neutralization of inputs. Therefore, in all ports a four-phase handshake takes place [4]. The PCFB/PCHB circuits could communicate to each other. In other words, output of any circuit could be connected to any input port of any other blocks and there is no need for any extra signal lines such as clock line. Each block is activated upon receiving a valid input. Input validity detection circuitry is similar to figure 4.

In these circuits there is special gate indicated by a ‘c’ letter and called C-element with inverted output. This gate is not used in normal synchronous circuits and operates as follows: when all of the inputs of the gate are zero, the output raises and when all the inputs of the gate are one, the output goes down. In other cases the gate *holds* its previous output value. This gate can be used as a small memory element.

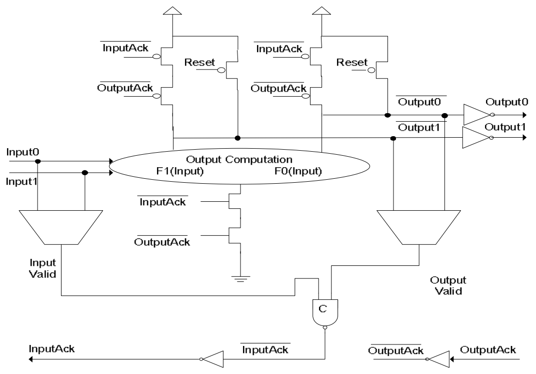

Figure 2: PCHB block diagram

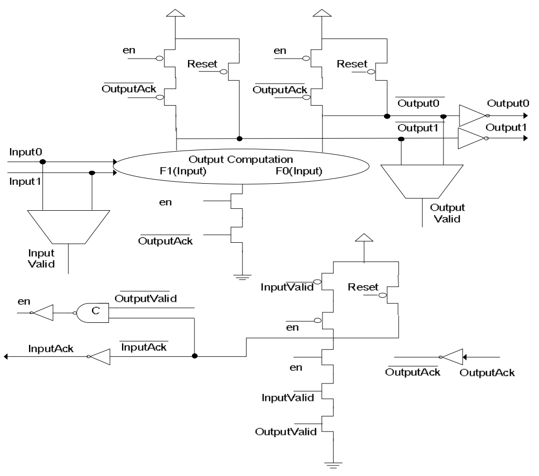

Figure 3: PCFB block diagram

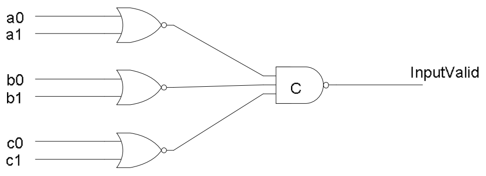

Figure 4: Validity checker circuit for a 3-bit dual-rail input

The only difference between PCFB and PCHB templates is that in the PCFB, neutralization of the output and falling of *InputAck* could happen together and independent of each other. On the other hand, in PCHB blocks always *InputAck* goes down after neutralization of the output.

In Figures 2 and 3, the circuits only have normal (Unconditional) input/outputs and perform simple computations on the data. PCFB and PCHB templates can perform more complicated operations such as conditional input and output actions and also can contain internal state variables [7, 8]. TSYN can synthesize circuits containing all of these types of operations. These templates are versatile enough to allow implementation of almost all circuit blocks [7].

The most important part to be mentioned during decomposition of the circuits is the contribution of smaller blocks to higher operation speed. This is due to more *fine-grained pipeline* that is achieved. The fine-grain pipelines increase the area because of completion detection circuits. This is the same as the synchronous pipeline stages in spite of the fact that these circuits do not need any extra registers.

4. PCHB/PCFB Synthesizer

This program receives the behavioral description of the circuit including *read*, *write* and function evaluation in Verilog [10,11] format. For example the description of a 4-input fully asynchronous NOR gate is described as follows:

```
always begin
       `READ(a)
       `READ(b)
       `READ(c)
       `READ(d)
       x = ~ ( a | b | c | d );
       `WRITE(x)
end
```

The PCHB/PCFB synthesizer is able to synthesize any logical condition for *read*, *write* or function evaluation. The general format of the behavioral description is as follows:

```
        `READ(a)

        if ( Condition1 )
                `READ(b)

        x = function1
        `WRITE(x)

        if ( condition2 ) begin
                y = function2;
                `WRITE(y)
        end
```

This step of the synthesis generates a set of PRs. Each PR describes a pull-up or a pull-down network of a CMOS gate and all PRs together implement the same functionality as the original CSP code. For example the circuit depicted in figure 5 can be described by PRs in the following manner:

```
`PR_SET(~a | ~b | ~c , z)
`PR_RESET(a & b & c , z)
```

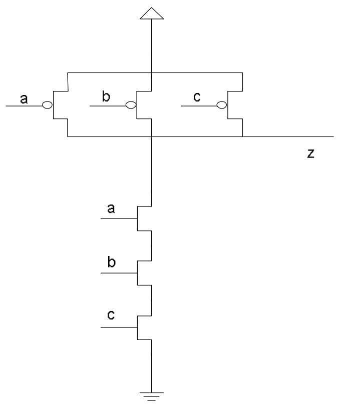

Figure 5: A sample 3-input NAND

By defining the *PR_SET* and *PR_RESET* macros, this description can be simulated by a normal Verilog simulator.

5. Operator Reduction

In this step PRs which can be mapped into standard gates are extracted and replaced by the equivalent gates. This operation is performed using evaluation of the logical expression in the PRs for different input values and comparing the results to the available gates in the library and corresponding truth tables.

The second functionality of this step is adding *state-holding* [4] elements to CMOS circuits that are not complementary. The result of this step consists of a set of PRs that could not be mapped into standard gates plus a net-list of standard gates and state-holders.

6. PR to CMOS transformation

In this step each PR is converted into a transistor network. For example the following PRs are converted to circuit in figure 6.

```
`PR_SET((~a | ~b) & ~c , z)
`PR_RESET((a & b) | c , z)
```

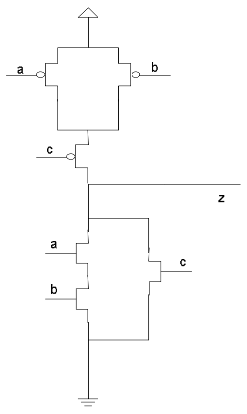

Figure 6: A sample transformation of PR to CMOS

This program can implement any logical function as a transistor network and finally generate Verilog or Spice output.

The other task of this step is transistor sizing for generating final layout. Transistor sizing of an inverter is used as a base to calculate transistor sizes in a more complicated network. Finally, we get a transistor netlist suitable for simulation and layout generation.

7. Synthesis of a sample circuit

For a clear view of the operation of the synthesis tool, consider the synthesis of a PCFB De-Multiplexer as depicted in figure 7. The circuit has an input port named *a* and a control port named *sel* and two output ports named *x* and *y*. when *sel*=1, the value of the port *a* is sent to *x* and when *sel*=0 the value of *a* is sent to port *y*. All the input and output ports are dual-rail coded.

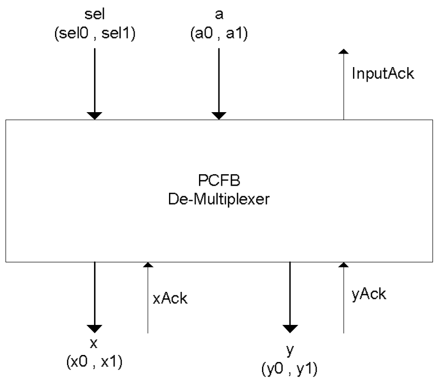

Figure 7: DeMux Circuit

The CSP description of the circuit is as follows:

```
	`READ(sel)
	`READ(a)


	if ( sel==1 )
		begin
			x = a;
			`WRITE(x)
		end
	if( sel==0 )
		begin
			y = a;
			`WRITE(y)
		end
```

The final circuit produced by the synthesis tool is shown in figures 8 to 11. In these figures state-holding elements are not shown and circuits for generating *InputAck* and *en* are the same as in Figure 3. Note that this circuit is not a pure combinational circuit and it could store data.

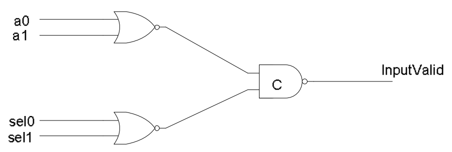

Figure 8: Input validity checker

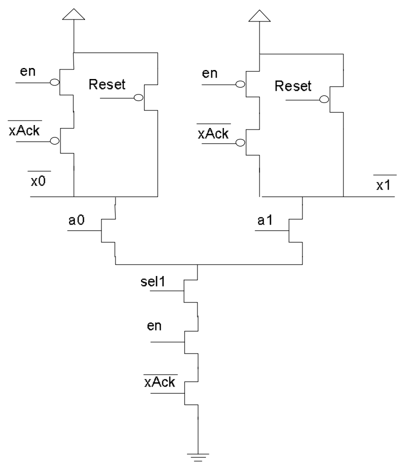

Figure 9: x output computation circuit

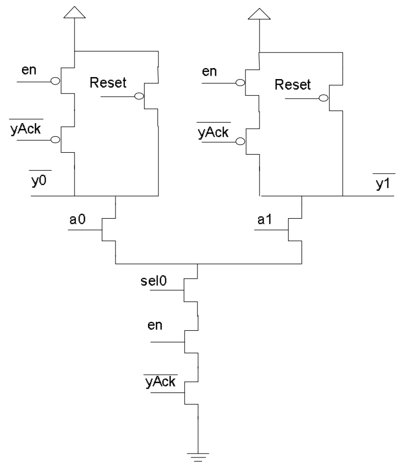

Figure 10: y output computation circuit

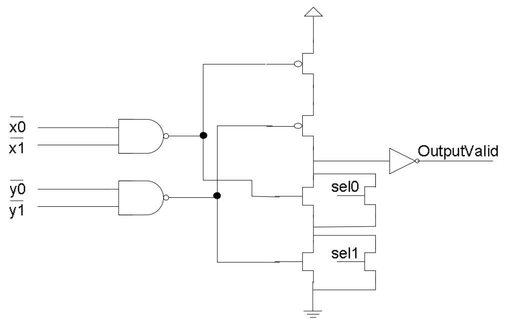

Figure 11: DeMux output validity checker

8. Results and Conclusion

TSYN is used to synthesize an Error Detection and Correction circuit using the Reed-Solomon codes. The circuit includes about 40,000 transistors. The synthesis of an 8-bit Microprocessor is in its final steps.

In summary, our synthesis tool is able to receive a behavioral description of a circuit and generate a sized transistor level output. Due to unavailability of high-level synthesis tools for asynchronous circuits, our approach, at the time, is focused on the practicability of automated asynchronous synthesis.

References

[1] Al Davis, Steven M. Nowick, “An Introduction to Asynchronous Circuit Design”, *Technical Report UUCS-97-013,* Department of computer Science, University of Utah, September 1997

[2] Jens Sparso, Steve Furber, *Principles of Asynchronous Circuit Design – A System Perspective*, Kluwer Academic Publishers, 2002

[3] C.H.(Kees) van Berkel, Mark B. Josephs and Steven M. Nowick, “Scanning the Technology : Applications of Asynchronous Circuits“, *Proceedings of the IEEE, 87(2):223-233*, 1999

[4] Alain J. Martin, “Synthesis of Asynchronous VLSI Circuits”, *Caltech, CS-TR-93-28*, 1991.

[5] A. J. Martim, M. Nystrom, K. Papadantonakis, P. I. Penzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E. V. Talvala, J. T. Tong, A. Tura, “The Lutonium: A Sun-Nanojoule Asynchronous 8051 Microcontroller”, *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC'03),* 2003

[6] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings and T. Lee. “The Design of an Asynchronous MIPS R3000 Microprocessor”, *Proceedings of the 17th Conference on Advanced Research in VLSI. Los Alamitos, Calif.:IEEE Computer Society Press, pp. 164–181,* 1997.

[7] A.M. Lines. Pipelined Asynchronous Circuits. *M.Sc. Thesis, California Institute of Technology,* June 1995, revised 1998.

[8] Kamran Saleh, “Asynchronous Design Using Pre-Synthesized Templates”, *Technical Report,* AmirKabir University of Technology, September 2003.

[9] Arash Seifhashemi, Mohsen Naderi, Kamran Saleh, Mostafa Salehi, Hossein Pedram, "PERSIA: An Asynchronous Synthesis Tool Based on Alain Martin's Method", *CAD Tutorial, 9th IEEE International Symposium on Asynchronous Systems & Circuits,* Vancouver, Canada, May 2003

[10] Arash Seifhashemi, Hossein Pedram, "Verilog HDL, Powered by PLI: a Suitable Framework for Describing and Modeling Asynchronous Circuits at All Levels of Abstraction", *Proc. Of 40th DAC, Anneheim, CA, USA,* June 2003

[11] Arash Seifhashemi, H. Pedram, "Verilog HDL, a Replacement for CSP", *3rd ACiD-WG Workshop FP5,* FORTH, Heraklion, Crete, Greece, Jan. 2003

[12] C. A. R. Hoare, “Communicating Sequential Processes”, *Communication of ACM 21, 8, pp 666-667,* 1978

[13] Alain J. Martin, “Asynchronous Datapaths and the Design of an Asynchronous Adder”, *Formal Methods in System Design, Kluwer, 117-137*, 1992