Scope International Journal of Science, Humanities, Management and Technology. ISSN : 2455-068X
Vol.1 Issue 2 (2015) 36-40. Submitted 24/12/2015. Published 31/12/2015

BEHAVIORAL SYNTHESIS OF ASYNCHRONOUS CIRUIT

Radhika.D, Priyanka.V, V.R.S College of Engineering and Technology,
Sangeetha priya.S, Asst prof, V.R.S College of Engineering and Technology,

Abstract:

A novel methodology and algorithm for the design of large low-power asynchronous systems are described. The system is synthesized by a commercial tool as a synchronous circuit, and subsequently converted into an asynchronous one. The conversion algorithm consists of extracting input and output sets, replacing the storage elements, identifying fork and join sets, and constructing request and acknowledge networks. A DLAP (Doubly Latched Asynchronous Pipeline) architecture is employed. The resulting asynchronous circuit can adapt its effective operating frequency to the supply voltage, facilitating flexible and efficient power management.

Keywords- DLAP, Asynchronous, Latched.

1. INTRODUCTION

Asynchronous logic has been advocated as a means of reducing power consumption in a number of situations. Such circuits typically switch only when required or when their inputs change. The power dissipated by the clock tree of a synchronous circuit is eliminated in asynchronous ones. The clock is replaced by local handshake signals, which typically require less power than the clock tree. Since switching power is proportional to the operating frequency, the circuit dissipates less power when the required throughput is reduced. Adaptive supply voltage can be lowered when speed is not required. Since power depends quadratically on voltage, the combination of slow-down and adaptive supply yields a cubic power saving with the reduction of speed. In addition, leakage power, which becomes more significant in newer process technology, can also be managed by reducing the supply voltage. It is easier to vary supply voltage in an asynchronous circuit, since there is no need to coordinate simultaneous variation of the clock frequency. Unfortunately, achieving such ambitious power savings by asynchronous design has proven to be extremely difficult for designers that are not experts in asynchronous design, because the methodology for the design of large asynchronous logic systems lags substantially behind that of synchronous circuits. To assist a designer in his/her attempts to convert a behavior level description of a compute function to be im- plemented in digital hardware, we have developed an toolbox, which is capable of scheduling and mapping operations on hardware resources. These operations are currently limited to ALU functions like multiplication, subtraction, comparison and addition. However, since many computational and signal processing func- tions consist of only these functions, many of them can be implemented.

2. PROBLEM STATEMENT

Behavioral synthesis is widely explored in the past, mostly targeting synchronous circuits. Scheduling, the process of allocating operations to time slots, is a well- known method for behavioral synthesis. A large number of scheduling algorithms are available, as well as control network topologies. In this paper, standard scheduling algorithms for synchronous circuits are used, but a new control network is created, targeting asynchronous circuits. For behavioral synthesis of asynchronous circuits, a number of methods for scheduling and resource allocation are published. However, these publications do not include the syn- thesis of the control network. Also, a number of behavioral synthesis methods for asynchronous circuits including the control network synthesis are published. Distributed controllers for asynchronous scheduled

data flow graphs are proposed, similar to our method, but each distributed controller is specified in a separate Signal Transition Graph (STG). STG's are hard to synthesize because they should operate hazard free. In our method, only a few small STG's have to be synthesized, which can then be reused to create the larger distributed controller. De-synchronization is the process of converting a (synthesized) synchronous circuit to an asynchronous circuit. Although this method does not target high-level synthesis and prevents resource sharing, the theory of de-synchronization is used in this paper since our method uses scheduling results for synchronous circuits.

3. BACKGROUD

A Data Flow Graph (DFG) is a graph of operations, represented by nodes, and data-dependencies represented by directed edges. Additionally, there are two extra nodes, the source and sink node. Those are used to represent data- dependencies from and to the environment.

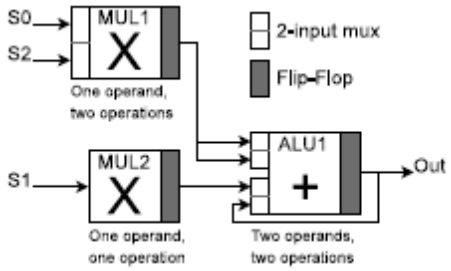

Fig.1. Datapath of FIR3 filter with flip-flop

When an operation processes input data, it depends on the source node, and when the output of an operation is used by the environment, the sink node depends on this operations. An example is shown in Figure 1-a, where the DFG of a 3rd order Finite Impulse Response (FIR) filter is depicted. When the data-dependencies are identified using the DFG, a scheduling algo- rithm can map each operation to a time slot. Then, operations can be allocated to resources like Multipliers and ALU's. Each resource can execute a number of operations from the DFG, but it can only execute one operation per time slot. Each operation is scheduled on a resource that is able to execute the operation. Data can be saved for more than one cycle in the flip-flop of a resource, but when the data needs to be saved after a new operation is executed, a register is used which is also considered a resource. This paper will not go into detail about scheduling and resource allocation, since well-known algorithms for synchronous circuits are used.Since the second input to the multiplier is a constant in the FIR, these are hardcoded in the multiplier and not shown in the datapath. There are a number of requirements which have to be satisfied in order to create a valid scheduling for the intended synchronous datapath.

4. PROPOSED METHOD

The controllers are based on the fall-decoupled model from. This model is live and flow-equivalent to synchronous circuits. However, this model does not allow hardware reuse, so a new model is created which allows hardware reuse, but is still live and flow-equivalent to the synchronous scheduling results. The A+ transition will make latch A transparent, while A− will make latch A opaque. In this model, even and odd latches alternate. To be able to implement the scheduling results, the Fall- decoupled model has to be extended to implement resource sharing. For each operand, a separate handshake signal is introduced

unless the data is an input from the environment or a constant, which are available during the entire operation of the circuit. The communication with the environment should also contain a form of handshaking to indicate that new input data is available and that the output data is ready. it indicates that all input data is valid, and when the done signal goes high, it indicates that the output data is available. The signals have to go low in the same order to reset the handshake signals to the initial state, i.e. it is a 4-phase handshake. Combining the Fall-decouple model with resource sharing results in the controller model .

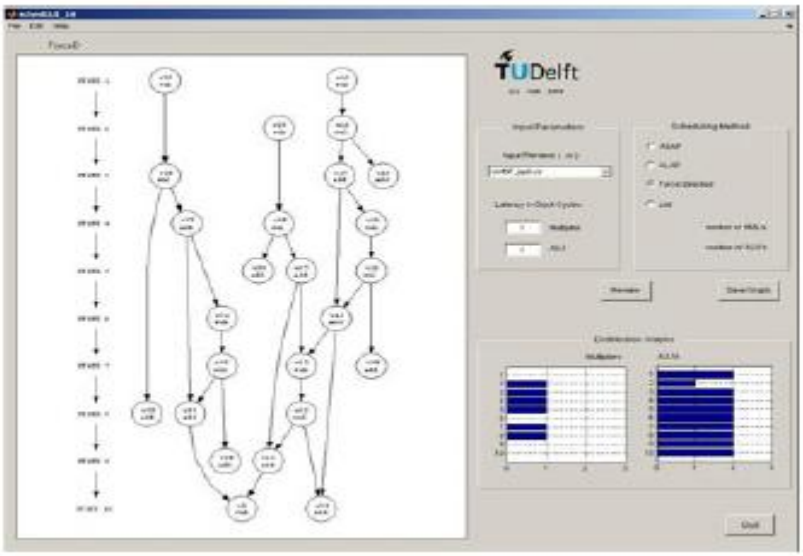

Fig.2. The user interface of the scheduling and mapping tool

In this marked graph, each transition is a latch control signal except Start and Done. The letter A, B and C indicate the input latch control signals for three different resources, while X represents the output latch control signal for the re- source with input latch A. The numbers associated with the latch control signal represent the time slot in which the operation is scheduled. If a resource has no operation scheduled for a certain time slot, the numbers will not be subsequent, but the numbers are always strictly increasing, e.g. no two operations can be scheduled on one resource at the same time and the order in time is honored by the marked graph.

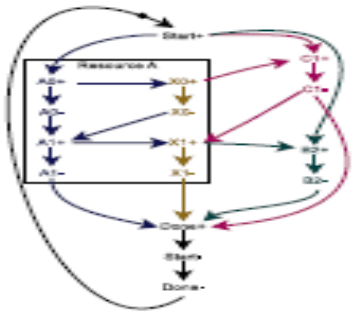

Fig.3. Fall decoupled latch controllers

In the rest of this section, we focus on the implementation of the Control Network. Initially, there is only a token at the positive event of the start signal. To prove that the model is Live, we have to prove that any directed circuit includes the start signal. The positive event of an odd latch (Xn+) is always fired by an event of an preceding (An+) or succeeding (Bn-m- where m>= 0) even latch at the same cycle or a lower cycle; The even latch from the same resource belongs to the same operation, and thus the same cycle. The negative event from the succeeding even latch (Bn-m-) has to be from the same cycle or a lower cycle, because the negative event indicates that the data in the odd latch from the previous cycle can be overwritten, because it is saved in the succeeding even latch. In the synchronous scheduling results, it is also assumed that previous output data is also available until the end of the next operation.

5. RESULT ANALYSIS

To test the asynchronous control flow, a number of high-level descriptions were synthesized. The implemented circuits include a 5th order LWDF low-pass filter and an 18-point IMDCT. The circuits were scheduled using the List scheduling algorithm. It is assumed that an ALU with two latches and a MUX has 70% of the delay of an MUL with two latches and a MUX, so during scheduling the ALU was assigned 7 cycles and the MUL was assigned 10 cycles.

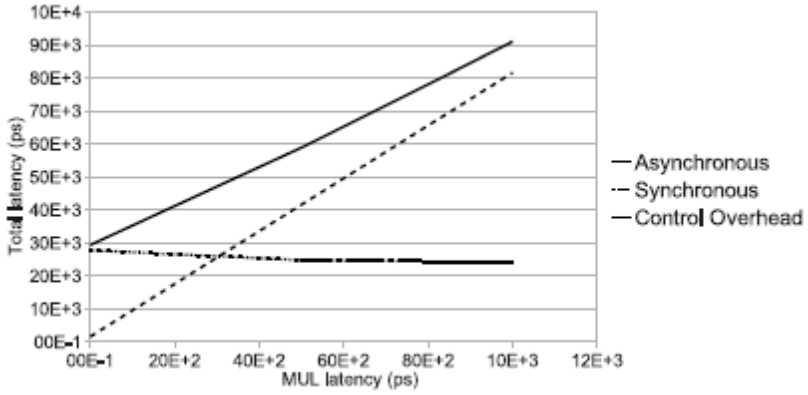

Fig.4. Latency of asynchronous and synchronous LWDF filter with different multiplier latencies

During synthesis, the delay constraints for the ALU was set to 3.5 ns and the delay for the MUL was set to 5 ns. The circuits were implemented in UMC90 with a gate library produced by the Faraday corporation. The netlist of the IP-blocks was created using the Technology Mapping function in Petrify and the layout of the IP-blocks is designed using Cadence Encounter. The IP- blocks are then used to implement the control network for the scheduling results using our scheduling toolbox. Synopsys Design Compiler is used to compile the datapath and select delay elements to match the datapath latency. Then, the datapath, control network and delays are combined in Cadence Encounter and the placement and routing of the IP-blocks and datapath completes the layout. The delay of the data operations were distributed over the latch delay elements instead of using an extra delay element.

CONCLUSION

In this paper we have presented a toolbox for the automatic generation of asyn- chronous circuits starting from van data flow graph description. The toolbox con- sists of a scheduling and code generation tool.

We use traditional scheduling al- gorithms as for synchronous circuits, but have replaced the implied synchronous controller for an asynchronous distributed control network. We have also pre- sented an asynchronous distributed control network based which based upon a number of pre-designed and optimized IP-blocks. More importantly, the design asynchronous digital circuits has become a lot easier, since our high level synthesis toolbox automatically generates asynchronous circuit implementations of a given set of data flow graphs. But also, our toolbox is capable to synthesize large and very large complex circuits. To our knowledge, this was not possible before.

REFERENCES

1. Bachman, B., Zheng, H., Myers, C.: Architectural synthesis of timed asynchronous systems. In: International Conference on Computer Design, ICCD 1999, pp. 354–363 (1999)
2. Blunno, I., Cortadella, J., Kondratyev, A., Lavagno, L., Lwin, K., Sotiriou, C.: Handshake protocols for de-synchronization. In: Proceedings of 10th International Symposium on Asynchronous Circuits and Systems, pp. 149–158 (April 2004)
3. Cortadella, J., Badia, R.: An asynchronous architecture model for behavioral synthesis. In: Proceedings of 3rd European Conference on Design Automation, pp. 307–311 (March 1992)
4. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers (1996)
5. Hamada, N., Shiga, Y., Saito, H., Yoneda, T., Myers, C., Nanya, T.: A behavioral synthesis method for asynchronous circuits with bundled-data implementation (tool paper). In: 8th International Conference on Application of Concurrency to System Design, ACSD 2008, pp. 50–55 (June 2008)
6. Imai, M., Nanya, T.: A novel design method for asynchronous bundled-data transfer circuits considering characteristics of delay variations. In: 12th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 10–77 (2006)
7. de Micheli, G.: Synthesis and Optimization of Digital Circuits. McGraw-Hill Higher Education, New York (1994)
8. Saito, H., Hamada, N., Jindapetch, N., Yoneda, T., Myers, C., Nanya, T.: Scheduling methods for asynchronous circuits with bundled-data implementations based on the approximation of start times. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E90-A, 2790–2799 (2007), http://portal.acm.org/citation.cfm?id=1521680.1521697
9. Sparsø, J., Furber, S.: Principles of Asynchronous Circuit Design. Kluwer Academic Publishers, Dordrecht (2001).