A Scheduling Method for Asynchronous VLSI System Design

2R－2 Rafael K. Morizawa Yoichiro Ueno Hiroshi Nakamura† Takashi Nanya†

Tokyo Institute of Technology, Graduate School of Information Science and Engineering
†University of Tokyo, Research Center for Advanced Science and Technology

1 Introduction

With the revival of interest in asynchronous systems there is a need for methods and tools for the high-level synthesis tailored for them. Although there are various methodologies already published that deal with synchronous design, there is not much work developed for the scheduling of asynchronous VLSI systems. In particular, we are interested in a methodology to schedule asynchronous pipelines.

In this note we propose a method for scheduling asynchronous VLSI systems, and then show how this method, together with existing synchronous methodologies can be used to synthesize asynchronous pipelines.

2 Asynchronous scheduling

Given a data flow graph (DFG), a set of constraints (performance/area), and a set of libraries of functional modules, the task of scheduling an asynchronous system is to decide how to place the functional modules so that they satisfy the constraints, and at the same time trying to minimize the necessary resources.

In the method we propose here we assume that a DFG, performance constraints (throughput and latency), and a set of libraries of functional modules are given. (We will not consider in this note area/resource constraints.) An example of a data flow graph is shown in figure 1a.

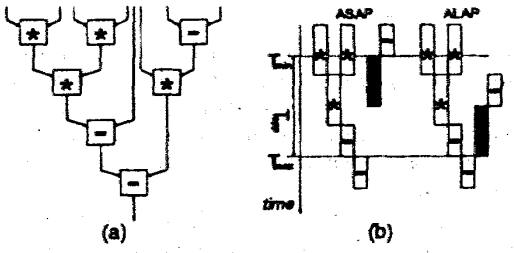

Figure 1: Data flow graph example (a), and the corresponding ASAP and ALAP schedules to find the time frames (b).

In general, the problem of pipeline scheduling is to find where to insert stage latches. An additional difficulty of asynchronous pipeline scheduling is, besides this, the fact that time is not discrete, but continuous. This makes difficult to apply synchronous pipeline scheduling methods to asynchronous systems. However, we can think of an asynchronous system as a synchronous system driven by an extremely high frequency clock. Under this condition, it is possible to make modifications in a few synchronous pipeline scheduling methodologies, so that they can be used in the synthesis of asynchronous pipelines.

The method we propose here takes an "asynchronous" extension of the Force-Directed scheduling [1] as its base. We use the concept of *time frames* introduced in the Force-Directed scheduling to guide the division of the data flow graph into pipeline stages, then use the adapted force-directed scheduling to schedule. If the resultant schedule does satisfy the given constraints, then the algorithm is over. In the next sections we explain in detail each of these parts.

3 Definitions

We will first present a few definitions necessary to explain the adaptations we made in the force-directed scheduling algorithm. It is important to remember that the adaptation is based on the idea that an asynchronous system can be seen as a synchronous system driven by an extremely high frequency clock. We will call the period of this high frequency clock as a *slot*.

Time frames A time frame is determined by finding the ASAP (as soon as possible) and ALAP (as late as possible) schedules. It represents the time interval in which an operation associated to a node of the DFG within which can be scheduled. In the synchronous version of the ASAP and ALAP schedules, we assume that all the operators' delay is bounded, so that we can consider that all "fit" within a clock period. Here, each operator's processing delay is its average delay.

Figure 1b illustrates the ASAP and ALAP schedules of figure 1a. The time interval T_{app} shown in figure 1b is the time frame of the shaded multiply operation. If we take the delay values of table 1, $T_{app} = 13.4$ ns.

Distribution graph The distribution graph dg of an operation op is the summation of the probabilities of each operation of the same type for each slot i of the DFG and is defined as $dg[i] = \sum_{\text{same type } op} prob[i]$. The probability $prob[i]$ is calculated as follows: $prob[i] = \frac{d_{av}}{T_{app}}$, $T_{min} \leq i \leq T_{max}$, and $prob[i] = 0$, $T_{min} > i > T_{max}$. d_{av} is the average delay of the functional unit that performs the operation and T_{app} is the length of the time frame. The distribution graph represents the concurrency of an operation of type op.

Self force In the original algorithm, the self force is calculated using *distribution graphs*. In our modified algorithm the self force sf associated with the assignment of an operation op within the time interval $[t_i, t_f]$, is calculated as $sf[t_i, t_f] = \sum_{i=t_i}^{t_f} [dg[i] \times x[i]]$, where $x[i]$ is the difference between the value of $dg[i]$ before and after assigning an operation in the time interval. The *self force* reflects the effect of assigning a functional module to a time interval on the overall operation concurrency.

4 Algorithm

The force-directed scheduling is a scheduling algorithm that uses the concept of *self force* as a cost function in order to perform scheduling. The scheduling consists of systematically (in an top down approach) allocating a resource in the given DFG so that to leverage its dg. The self force provides the information necessary to decide when to schedule an operation. This part of the algorithm has not been modified in our adapted version.

The original force directed scheduling as defined in [1] is targeted at synthesizing non-pipelined designs. However, with an extension to the algorithm presented in the same paper it can be easily adapted to the scheduling of pipelines. The extension proposed in [1] consists of divi-

ding the distribution graph of the given DFG in pieces of the same length of the stage's latency and placing them horizontally. Then, perform the force-directed scheduling of the resultant graph.

In our adapted version, we try to divide the time frame into pieces of the same length of the data initiation interval (this value can be obtained from the required throughput). The cut pieces correspond to pipeline stages and latches are inserted between them. However, differently from the original algorithm, where there was no need to recalculate the distribution graph after dividing it, there are cases when this is necessary in the asynchronous version. Figure 2 shows an example of a cut.

The reason for recalculation is that, when dividing the original distribution graph, there may be an operation's distribution graph that will not entirely fit into one division, i.e., into one stage. When this happens, the solution is to cut the distribution graph to divide into two (or more) parts, and to distribute these parts over the other parts of the distribution graph. We follow the next guidelines to make the cuts.

1. If the over extended part of the time frame is smaller than the average delay of the correspondent operator, then this part is *cut off*.
2. If the over extended part of the time frame is larger than the average delay of the correspondent operator, then it is left without modification.

Summarizing, below are the steps of the proposed pipeline scheduling.

1. Find the time frames.
2. Find the critical path of the data flow graph.
3. Divide the time frame found in step 1 guided by the critical path and according to the given constraints. Calculate the distribution graphs from the new time frame.
4. Perform the force-directed scheduling.
5. If the resultant schedule satisfies the given constraints, then end the algorithm.

5 Example

Here we perform the pipeline scheduling of the DFG shown in figure 1a. The available functional modules are shown in table 1. The performance constraints are a minimum data initiation interval of 14 ns. The time frames of the DFG are shown in figure 1b. The critical path of the DFG is the chain of the following operations: multiply, multiply, subtract, subtract. Step 3 is shown in figure 2. Figure 3a shows the *dg* of the multiply operation, and figure 3b shows the schedule. The box's length in the figure corresponds to the average delay of the operator indicated inside the box. Boxes with the same color indicate that the same functional module was used.

module	average delay (ns)
subtracter	4.0
multiplier	6.7

Table 1: Average delay of functional modules used to synthesize the pipelined design of figure 1.

Stage 1 has average latency of 13.4 ns and stage 2 has average latency of 8 ns. Although this satisfies the given constraints, we note that the "average" latency of the individual stages differ in almost 25%. In complex pipelined data paths, a great dispersion in the latencies

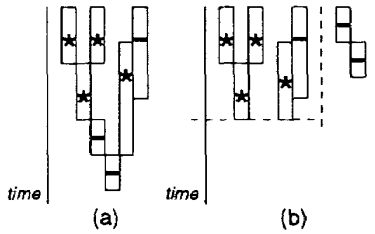

Figure 2: Time frame of the DFG of figure 1a before (a), and after (b) the time frame's cut.

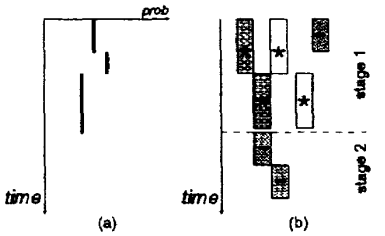

Figure 3: The *dg* of the multiply operation (a), and the obtained schedule (b).

of the stages may have negative effects in its overall performance [2].

Also, when actually simulating the data path of figure 3b, we have noticed that the resultant average latency of each stage is lower than the predicted. Further simulations indicated that the sum of the average delay of functional modules composing a data path's critical path is always greater than the actual data path's critical path average latency. This discrepancy may also prevent us finding a better design.

6 Conclusion

We have presented a methodology that uses existing synchronous scheduling methodologies, adapted to asynchronous VLSI system synthesis. The principle used to adapt the algorithm is that an asynchronous system can be seen as a synchronous system driven by a high frequency clock.

This approach allow the designer the possibility of using existing methodologies to synthesize asynchronous VLSI systems. However, as shown in section 5, it may not always produce a good asynchronous design. Further study of the two points raised in section 5 is necessary. Also it is necessary to develop methods to intelligently choose functional modules from the libraries, since an unfortunate selection of operators can make the design impossible to satisfy the given constraints.

This work was supported in part by the Ministry of ESSC under Grant-in-aid for Scientific research No. (B)09480049 and by STARC.

References

[1] Pierre G. Paulin and John P. Knight. Force-directed scheduling for the behavioral synthesis of ASIC's. *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 6, pp. 661–679, June 1989.

[2] David Kearney and Neil W. Bermann. Performance evaluation of asynchronous logic pipelines with data dependant processing delays. In *Asynchronous Design Methodologies*, pp. 4–13. IEEE Computer Society Press, May 1995.