Semi-Dynamic Scheduling of Synchronization-Mechanisms

Wolfgang Ecker
Siemens AG
Corporate Research & Development
Munich, 81730 Germany

Abstract

This paper presents a novel approach to scheduling of hardware supported synchronization operations. The optimization goal is to minimize the interation time of processes and thus the overall computation time of a system composed of a set of interacting processes.

To minimize computation time, minimum timing constraints of synchronization mechanisms have to be satisfied. In order to meet these requirements the control flow oriented scheduling algorithm allows to schedule synchronization operations into loops with unknown iteration count. To achieve this, a set of control steps into which each synchronization operation may be scheduled is computed and afterwards, a controller is synthesized, which determines the final schedule dynamically during execution.

1 Introduction

System-level synthesis of digital systems[1] deals with multiple communicating processes. Tasks of system-level synthesis (see e.g.[1, 2]) are e.g. partitioning (see e.g. [3]), including clustering and constraint partitioning as well as behavioral transformations like process generation from procedures or functional pipelining.

In an early design phase, only causal relations inside processes and between processes are specified. Later on detailed timing information is considered in the design process. Assuming a synchronous design style, first clock cycles and later clock-period(s) or propagation delays respectively are introduced in the design. The introduction of detailed timing can be split up into two tasks: protocol- or interface synthesis and high-level synthesis. The former deals with synthesis of synchronization and communication mechanisms between the processes, and the latter (for an overview see [1, 3, 4]) with synthesis of operations inside single processes.

Tasks of protocol synthesis are control step scheduling [5], which inserts or removes time slots for correct data exchange, selection of protocols including serial-parallel transmission trade-off [6], or the implementation of hardware-supported synchronization which adds hardware to allow for correct data exchange, control of subunits or resource sharing between concurrent processes.

This paper presents semi-dynamic scheduling a novel approach to scheduling of hardware supported synchronization. The paper is organized as follows: Existing approaches and related work are presented in the next section. Properties of protocols and synchronization mechanisms are discussed afterwards. The algorithm and an example concludes the paper.

2 Previous Work

Methods as implemented in high-level synthesis tools (see e.g. [1, 3, 7, 4]) typically do not attack the freedom of scheduling synchronization operations. Either the number of time slots (= number of required control steps or clock cycles respectively) is minimized, or area and propagation delay are minimized subject to pre-scheduled IO-operations. In the first case the optimization of concrete synchronization mechanisms is inhibited, in the second case adjustments to the control steps of I/O operations can not be made.

A special approach in high-level synthesis, the data-flow oriented scheduling under consideration of relative and incomplete time constraints, as presented in [8] or [9], may allow for scheduling of synchronization operations. However, this approach treats level sensitive synchronization mechanisms equally to edge sensitive synchronization mechanisms due to data-flow oriented schedule. Therefore it wastes one clock cycle per level-sensitive synchronization mechanism. Moreover, this approach does not allow scheduling of loop variant operations into loops with unknown iterations count.

Scheduling approaches which do allow scheduling into loops and over loops are described in [10] and [11]. These approaches, however, do not allow scheduling of loop variant operations into loops with an unknown iteration count.

Approaches for protocol and synchronization synthesis, respectively can be found in [6] and [5]. Both approaches, however, try to minimize area only. The first approach is based on the evaluation of a serial-parallel data transmission trade-off and the second approach focuses on detection and removal of unnecessary synchronization lines and optimization of the required control logic.

[1] System-level synthesis includes also the synthesis of heterogeneous, e.g. electrical analog/digital systems or mechatronic systems. This paper, however, focuses on the synthesis of synchronous digital systems only. Hardware/software co-design is not considered, too.

3 Protocols

Protocols are used to synchronize the execution of concurrent processes and/or to allow for data exchange between concurrent processes. *Protocols* can either consist of merely a synchronization mechanism or merely a data exchange mechanism or both a synchronization mechanism and a data exchange mechanism. This paper mainly focuses on the synchronization aspect of protocols.

Synchronization mechanisms are used to guarantee a required temporal relation-ship of a set of concurrent processes. A synchronization specification consists of a physical specification and a behavioral specification. The physical specification describes the connectivity required for synchronization. The behavioral specification includes temporal requirements and describes the temporal sequence of events. It is composed of a set of synchronization operations. The physical specification is implemented by wires, whereas the behavioral specification is implemented by a controller, a sub-controller or a part of a controller[2].

A *synchronization operation* is seen in this paper as a part of a synchronization mechanism that can be mapped onto one control step. A *control step* is an atomic operation performed by a controller. The time required for a control step is one clock cycle in synchronous designs.

A *time constraint* specifies the distance of two events or two operations, respectively. The distance may be specified by an exact value, a lower bound, an upper bound or an interval consisting of a lower bound and an upper bound. A synchronous design style requires that the values are specified in terms of clock cycles and an additional specification of the clock period. A *minimum time constraint* is the exact value or the lower bound of a time constraint.

A *synchronization point* specifies one point in the synchronization mechanism between two synchronization operations. Intended time relations are anchored at a synchronization point.

The need for a separate specification of the synchronization point is illustrated by the following example. Assume a synchronization mechanism as shown in Figure 1. It ensures that two processes run in parallel after its execution.

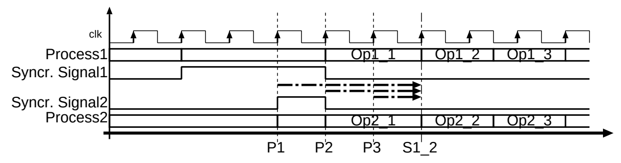

Figure 1: *Need for Synchronization Point*

Three points in the protocols P1, P2 and P3 can be used to specify a time constraint for the beginning of the subsequent operations Op1_2 and Op2_2 respectively. Hence, one of the three possible points must be specified as the point at which synchronization is performed to allow an unambiguous time constraint specification of the starting point S1_2 of operation Op1_2 and Op2_2.

[2]It should be remarked that a controller is a piece of hardware which implements the control-flow of a description. The implementation of a controller, however, is not restricted to a special target architecture, e.g. a finite state machine

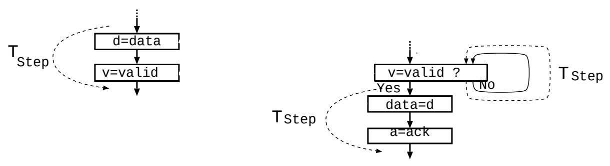

Figure 2: *CDFG of a Hand-Shake Mechanism*

4 Properties of Synchronization Mechanisms

The behavioral specification of synchronization mechanisms shows some important features listed below. They must be considered during scheduling but also can be used for optimization purposes.

4.1 Data Types of Synchronization Mechanisms

Data types for synchronization consist of bit values or enumeration types which may be mapped on bit values. Thus, only bit manipulations but no arithmetic operations have to be performed for synchronization. The hardware required for their implementation is relatively small compared to datapath elements like a multiplier.

4.2 Control-Flow Structure of Synchronization Mechanisms

Synchronization ensures that one or more processes perform actions concurrently, exclusively or sequentially. To allow this, synchronization operations include operations signalling that an action has been performed or that an action waits to be performed, and operations awaiting a signal of other processes. The latter task can be implemented in hardware via polling only. This implies that a loop with an unknown iteration count has to be executed for this task.

An example containing an unconstrained loop is shown in Figure 2. It represents pieces of a handshake protocol's control flow graph. Dataflow arcs have been shadowed and arcs for time constraints have been dashed in the example. T_{Step} specifies the exact time constraint of one clock cycle.

4.3 Timing Constraints of Synchronization Mechanisms

Scheduling synchronization operations must generate an implementation which satisfies time constraints. To allow for fast response of a system composed of interacting processes, synchronization operations must be scheduled as fast as possible This requires, that minimum time constraints have to be exactly met.

Figure 3 illustrates the impact of delayed synchronization operations which consist in the example only of the valid signals valid_in, valid_x and valid_out. It shows waveforms of the synchronization and data exchange of a system composed of two sub-systems. The overall computation time of the system depicted

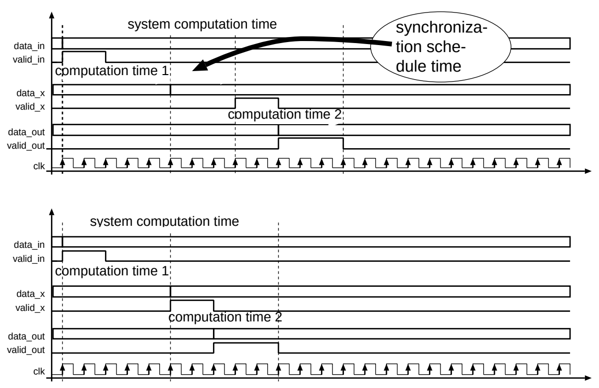

Figure 3: *The Influence of the Schedule of Synchronization Operations on the Overall Computation Time*

in the upper part of the figure is greater than the computation time of the system depicted in the lower part due to the internal synchronization operation (`valid_x`) being performed too late.

4.4 Level- and Edge-Sensitive Synchronization Mechanisms

A *level-sensitive synchronization mechanism* is a synchronization mechanism which requires a defined value for a defined time interval only. It is aminly used for closely coupled processes.

A level-sensitive synchronization mechanism can be scheduled into a single control step. An example for a level-sensitive synchronization mechanism is the increment control of an up-counter. If a signal of the synchronization mechanism has a specified value, the counter will increment its value during each control step.

An *edge sensitive synchronization mechanism* is a synchronization mechanism which requires the change of a value to a specified value. It is used for the synchronization of completely independent processes.

Hence, an edge sensitive synchronization mechanism requires at least two control steps. Hand shake synchronization mechanisms belong to this type of synchronization mechanisms.

It is important to note, that edge- and level-sensitive synchronization mechanisms must be distinguished. Otherwise one control step is wasted when a level-sensitive synchronization mechanism is used like an edge-sensitive synchronization mechanism.

4.5 The Synchronization Point

If operations for data exchange exist then they occure befor the synchronization point. These and all synchronization operations before the synchronization point main not be moved to keep the time anchor associated with the synchronization point fixed.

Additionally, it is important to note that all dataflow dependecies of the synchronization operations are fixed by control flow dependencies and time constraints.

5 Semi-Dynamic Scheduling

5.1 The Basic Idea

The basic idea is to perform control flow oriented time constrained scheduling of the synchronization operations after the synchronization point. The operation before the synchonization point must remain in their place in the data flow graph. These should be considered only in an additional data flow oriented scheduling step.

Dealing in this way, all dataflow dependencies inside and between synchronization mechanisms are fulfilled due to 4.5.

5.2 Level- and Edge Sensitive Protocols

Level and edge sensitive protocols must be be handled differently. This can be achived by setting minimum time constraints (see 4.4) to all synchonization operations but the tail operation of a level sensitive protocol. Thus, this synchronization operation can be both executed or overwritten in control flow oriented scheduling.

5.3 Solving the Scheduling Problem

The main problem to be solved in scheduling of synchronization mechanisms is to allow scheduling of loop variant operations into loops with unknown iteration count due to 4.2 and 4.3. The basic idea is to schedule the synchronization mechanisms *dynamically*, i.e. the schedule of the synchronization operations is determined during execution by the synthesized controller. This can be achieved by synthesizing a controller which executes synchronization operations depending on their earlier execution in both conditionally executed branches[3] again as well as in unconditionally executed branches.

To store the state of execution of synchronization mechanisms, i.e. the synchronization operations which have already been executed, a flag is introduced in the controller. The controller will only execute a synchronization mechanism if its corresponding flag is not set; in turn it will set the flag upon execution of the operation.

To avoid errors, it must be also considered that a synchronization mechanism may not have been finished before beeing initiated again. This implies that the synthesized controller has to finish execution of a synchronization operation of a synchronization mechanism, before it starts a subsequent instance of the synchronization mechanism.

The expected overhead should be acceptable due to the considerations of 4.1.

5.4 Scheduling Algorithm

The semi-dynamic scheduling algorithm consists of four parts (see Listing 1): The first part removes the synchronization operations which can be scheduled, stores them related to their synchronization mechanism for later use, and analyzes their properties. The second part determines the control steps for the operations between the synchronization operations. The implementation of this point is discussed in Section 5.5 in more detail.

[3]including loops with unknown iteration count

```
algorithm semi-dynamic scheduling
    $r$ : list of synchronization operations;
    $Lb$, $Le$ : array of labels;
    $S$ : array of integer;
    $R$ : array of lists of synchronization operations;
    $T$ : array of integer;
    $CSmin$, $CSmax$, $csmin$, $csmin$ : integer;
begin
Part 1: for all synchronization mechanisms $s$
        label begin and tal of synchronization protocols in CFG with labels $Lb(s)$ and $Le(s)$
        $r$ = cut synchronization operations of $s$ after the synchronization point;
        $S(s)$ = number of synchronization operations in $r$;
        $R(s, \mathrm{i})$ = i-th synchronization operation of $r$ $\forall$ $1 \leq \mathrm{i} \leq S(s)$;
        $T(s)$ = maximum number of control steps required for $r$;
    end for;
Part 2: (see Section 5.5) determine other control steps in controller
Part 3: for all synchronization mechanisms $s$
        / / Block oriented Timing Analysis
        $CSmin$ = minimum number of control steps between synchronization point and start of $s$
        if $T( s ) > CSmin$ then / / Enough control steps available for synchronization operation
            for i = 1 ... $S(s)$ / / Determine control steps which may not be executed
                csmin = minimum number of control steps required for $R(s, \mathrm{j})$ $\forall 1 \leq j \leq i$;
                if $csmin > CSmin$ then
                    / / Introduce flag controlling dynamic schedule
                    insert flag($s$) in controller, if not already done
                    / / Schedule synchronization operation dynamically
                    insert R($s$, i) before Lb($s$) under condition of flag($s$) in control flow graph;
                end if;
            end for;
        end if;
    end for;
Part 4: for all synchronization mechanisms $s$
        for all control steps $c$
            if $c \in R(s, \mathrm{i})$ $\forall$ $1 \leq i \leq S(s)$ then continue
            / / Block oriented Timing Analysis
            $CSmin$ = minimum number of control steps between synchronization point of $s$ and $c$;
            $CSmax$ = maximum number of control steps between synchronization point of $s$ and $c$;
            for i = 1 ... $S(s)$
                $csmin$ = minimum number of control steps required for $R(s, \mathrm{j})$ $\forall 1 \leq j \leq i$;
                $csmax$ = maximum number of control steps required for $R(s, \mathrm{j})$ $\forall 1 \leq j \leq i$;
                if $csmin = csmax = CSmin = CSmax$ or / / interval exactly determined?
                        i = S($s$) and kind( $R(s, \mathrm{i})$ ) = invariant and / / loop invariant final syncronization operation?
                            $csmin \leq CSmax$ and $CSmin \leq csmax$ then
                    insert $R(s, \mathrm{i})$ in $c$ / / static schedule
                else
                    if $csmin \leq CSmax$ and $CSmin \leq csmax$ then / / syncronization operation possibly in interval?
                        / / semi-dynamic schedule
                        insert flag($s$) in controller, if not already done
                        insert R($s$, i) in $c$ under condition of flag($s$) in control flow graph;
                    end if;
                end if;
            end for;
        end for;
    end for;
end algorithm;
```

Listing 1: Semi-Dynamic Scheduling

The third part performs a static analysis of the control steps and inserts synchronization operations depending on the minimum number of control steps, to ensure that a synchronization mechanism has finished before it is being started again. The final part schedules, depending on the number of control steps between the synchronization point of each synchronization mechanism and any control step in the process, the synchronization operations either statically, or dynamically without using a flag or dynamically using a flag.

The complexity $\mathcal{C}$ of the semi-dynamic scheduling algorithm can be formulated as

$$\mathcal{C} = \underbrace{|\mathcal{S}| * |\mathcal{P}|}_{\text{part1}} + \underbrace{|\mathcal{S}| * |\mathcal{S}|}_{\text{part3}} + \underbrace{|\mathcal{S}| * |\mathcal{C}| * |\mathcal{P}|}_{\text{part4}} = O(|\mathcal{S}|^3)$$

without considering part 2: determination of control steps in the process. Here, $|\mathcal{S}|$ is the number of nodes in the control-flow graph before scheduling, $|\mathcal{P}|$ is the number of nodes in the synchronization mechanism and $|\mathcal{C}|$ is the number of control steps. The formula shows that the complexity of the semi-dynamic scheduling algorithm is in the worst case cubic in the number of nodes of the control-flow graph.

5.5 Integration in the Design-Flow

All control steps of a process must be known to allow semi-dynamic scheduling. Three alternatives for the determination of the control steps are possible. These possibilities also show the interaction of semi-dynamic scheduling with high-level synthesis.

1. If the time constraints between the synchronization mechanisms are not specified exactly, high-level synthesis will be performed as a second part of the presented semi-dynamic scheduling algorithm. In this case the control steps between the synchronization mechanisms are determined by high-level synthesis.

   This alternative facilitates the application of pure resource constraint scheduling in combination with fixed synchronization timing of the synchronization operations.

2. If the time constraints between the synchronization points are fixed and feasible two alternatives exist.

   (a) As above, high level synthesis is performed as second part of the scheduling algorithm.

   (b) The second part consists of the introduction of control steps only. High-level synthesis is performed on the pre-scheduled control steps after part four of the semi-dynamic scheduling algorithm.

   For both alternatives, however, high-level synthesis must be able to satisfy exact time constraints. Alternative 2(b) moreover requires that high-level synthesis operates on pre-scheduled control steps.

Note that the determination of the control steps between synchronization mechanisms may be NP-hard depending on the accuracy of the implemented algorithm.

6 Scheduling Example

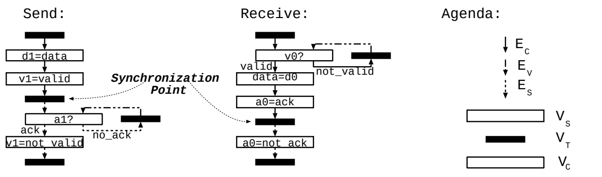

Figure 4: *CFG of a Hand Shake Mechanism*

Figure 4 shows the control-flows graphs for an independent[4] send and receive operation based on edge-sensitive hand-shake synchronization. The send operation consists of one control step before the synchronization point (operations `d1=data` and `v1=valid`) which can not be scheduled and two control steps after the synchronization point, which can be scheduled. The first control step after the synchronization point is required to wait for the acknowledge signal of the communication partner and the second control step is required to ensure that the valid signal `v1` changes its value to `not_valid`. Hence, the following properties of the statements to be scheduled of the send routine can be evaluated:

1. The maximum number of clock cycles required for executing the synchronization operations of the send routine after the synchronization point is infinite.

2. The size of the flag storing the state of the send routine is three[5].

3. The first control step which can be scheduled is loop variant whereas the second control step which can be scheduled is loop invariant. This property of the second control step, however, can not be used for optimization as the number of iterations of the first control step is unknown.

The receive operation consists of two control steps, which can not be scheduled. The first is conditionally executed. It is required to wait for the valid signal of the communication partner. The second one is necessary to take data from the communication medium. One control step however can be scheduled. It consists of the statement `a0=not_ack`. It is required to ensure that the acknowledge signal `a0` changes its value to `not_ack` due to the fact that a hand-shake protocol is edge-sensitive. The receive operation has the following properties, to be considered for scheduling:

[4]The postfix 1 respectively 0 is introduced to show that the operations may operate on different signals. Hence, the same postfix is used for the example in Figure 5

[5]Later in the paper, the values of the flag will be called `waiting`, `pulse`, and `ready`.

1. The maximum number of clock cycles required for the executing the synchronization operations of the receive routine after the synchronization point is one.

2. The size of the flag storing the state of the receive routine is two.

3. The synchronization operation after the synchronization point is loop invariant.

Figure 5 shows the semi-dynamically scheduled control-flow graph of a process, which consists of a receive and send operation only. First, the synchronization operations of the send and receive operation (`v0?`, `data=d0`, `a0=ack`, `d1=data` and `v1=valid`), before the synchronization points of the synchronization mechanisms, are scheduled. No further process-internal scheduling must be performed in this example.

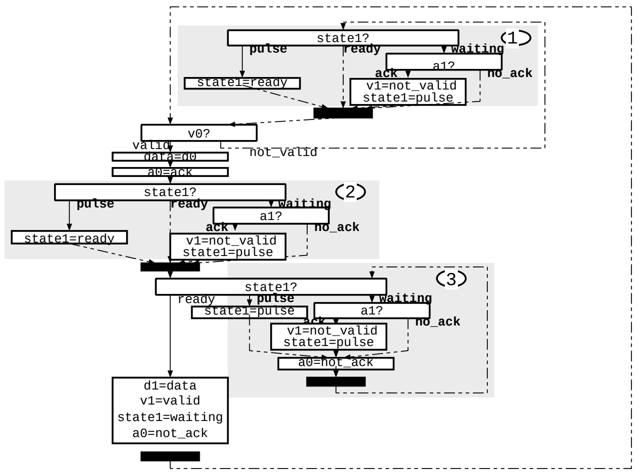

Figure 5: *Semi-Dynamically Scheduled CFG*

The next step, the static analysis of control steps, returns two as the minimum number of executed control steps. This implies that it can be ensured for the receive routine only, that it will be finished correctly. Due to the fact that the tail synchronization operation of the receive routine is loop invariant, no flag is required to store the state of the receive routine. The number of control steps required for the send routine is infinite and hence not smaller than the minimum number of executed control steps. Thus flag `state1` has to be introduced to store the current execution state of the send routine, and block **(3)** has to be introduced in the control flow graph to ensure, that the send operation can be finished before it will be started again.

The last step consists of scheduling the remaining control steps of the synchronization mechanisms. The operation `a0=not_valid` is scheduled statically in block **(3)** and after the operation `v1=valid` and the operations of the send routine are scheduled dynamically by inserting the blocks **(1)** and **(2)** in the control-flow graph.

7 Results

Several tests were made to analyze the effect of semi-dynamic scheduling. Semi-dynamic scheduling reduced the number of required clock-cycles in every test. The minimum clock period and the required area of the resulting circuits, however, increased in some cases.

8 Summary and Conclusion

A new method for clock cycle minimization of interacting processes by control flow oriented scheduling of synchronization mechanisms with cubic complexity was presented. The method allows for optimizing edge-sensitive as well as level sensitive synchronization mechanisms, which can not be handled by scheduling algorithms presented in the past. Moreover, the scheduling algorithm is able to schedule operations into branches and loops with unknown iteration count. The method was only developed for cycle based protocol optimization. Currently, we are working to apply semi-dynamic scheduling also for resource minimization and to reduce hardware overhead by generating concurrent controller for flag control.

Acknowledgments

I would like to thank Michael Münch and Norbert Wehn for their support and for helpful discussions.

References

[1] P. Michel, U. Lauther, and P. Duzy. *The Synthesis Approach to Digital System Design.* Kluwer Academic Publishers, 1992.

[2] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems.* P T R Prentice Hall, 1994.

[3] D.D. Gajski, A. Wu, N. Dutt, and S. Lin. *High-Level Synthesis.* Kluwer Academic Publishers, 1992.

[4] R.A. Walker and R. Camposano. *A Survey of High-Level VLSI Synthesis Systems.* Kluwer Academic Publishers, 1991.

[5] D.C. Ku, C. Coelho, and G. De Micheli. Interface Optimization for Concurrent Systems under Timing Constraints using Interface Matching. In *High Level Synthesis Workshop 92*, pages 202–213, 1992.

[6] S. Narayan and D.D. Gajski. Synthesis of System-Level Bus Interfaces. In *Proceedings of the European Design Automation Conference (EDAC)*, 1994.

[7] R. Camposano and W. Wolf. *High-Level VLSI Synthesis.* Kluwer Academic Publishers, 1991.

[8] J.A. Nestor. *Specification and Synthesis of Digital Systems with Interfaces.* PhD thesis, Carnegie Mellon University, Pittsburgh, 1987.

[9] D. Ku and G. De Micheli. Relative Scheduling under Timing Constraints. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC)*, pages 59–64, 1990.

[10] J. Fisher. Trace Scheduling: A technique for Global Microcode Compaction. *IEEE Transactions on Computers*, July 1981.

[11] R. Potasman, J. Lis, A. Nicolau, and D. Gajski. Percolation Based Synthesis. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC)*, pages 444–449, 1990.