School of Electrical, Electronic & Computer Engineering

UNIVERSITY OF
NEWCASTLE UPON TYNE

Transition Sequence Encoder

A. Mokhov and A. Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2006-117

September 2006

Contact:

Andrey.Mokhov@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant EP/C512812

NCL-EECE-MSD-TR-2006-117

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

`http://async.org.uk/`

Transition Sequence Encoder

A. Mokhov and A. Yakovlev

September 2006

Abstract

This paper introduces concept of a programmable event scheduler able to dynamically schedule a set of events given their order specification and presents Transition Sequence Encoder as a possible solution for a simplified event scheduling problem. The solution is derived in form of analytical Boolean equation and is translated futher to gate-level implementation.

1 Introduction

The problem of event scheduling is usually solved during the logic synthesis phase of design flow as the order of events is known from system specification. The approach allows synthesis tools like Petrify [2] or Optimist [4] to be used to generate circuits that will schedule the events in some preset order. Typical design flow of such an approach is shown in Figure 1. The system specification is given as a *Signal Transition Graph* (STG). It describes the circuit behavior and all the event order constraints are defined strictly without a possibility of real-time reordering. STG can be directly mapped into a Verilog netlist by tools like Optimist. Another way is to generate logic equations using Petrify and to map them into netlist using a particular gate library. The obtained implementation is static in sense that it is impossible to introduce new or change the existing order relations within the system without the its complete redesign and resynthesis.

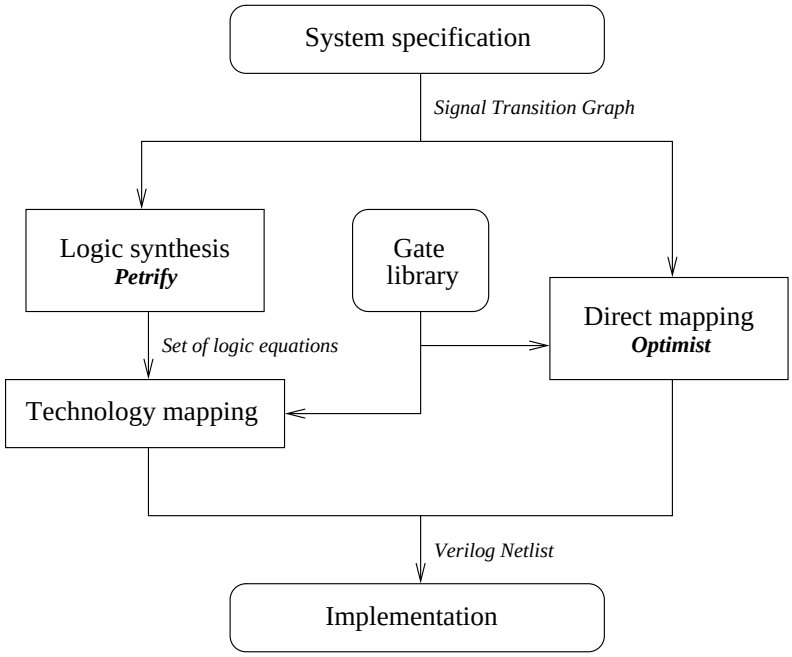


Figure 1: Design flow

But sometimes the order of events is not known beforehand and is constantly changing throughout the circuit lifetime. This calls for a dynamically programmable event schedulers. The paper presents a family of circuits able to schedule an arbitrary number of events given their order specification. The behaviour of such circuits

is described in Figure 2: given an event order specification and starting signal *go* the circuit initiates a series of request-acknowledgement handshakes in the specified order. The completion of the series is flagged by signal *done*. Note, that some of the handshakes can occur concurrently if the order specification does not restrict so. In such case the events are scheduled using *as-soon-as-possible* (ASAP) scheduling strategy i.e. an event is generated as soon as all of its order requirements are satisfied without waiting for the other, not important events for completion. This strategy minimises the overall execution time of all the events under the given constraints and maximises concurrency.

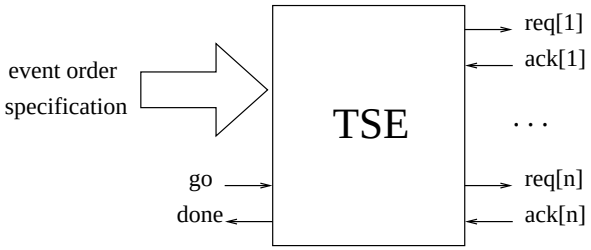


Figure 2: Transition Sequence Encoder

The general event scheduling problem can involve multiple occurences of the same event and cyclic order dependencies. In the paper we concentrate on a simpler case where an event can occur only once and event order specification does not contain cyclic dependencies.

2 Events order specification

2.1 Partial order graph and matrix

Let $E = \{1, ..., n\}$ be a set of n events. Their order can be specified with a directed graph $G = (E,\ R)$ where E is a set of vertices (which represent the events) and R is the set of ordered pairs of vertices or *arcs* (which represent the order relations between pairs of events). An example of such a graph is shown in Figure 3. Event 2 can occur only after event 1 has occured what is stated with an arc between them; event 5 can occur only when both of its precedeing events (4 and 7) have occured and so on. If two events are not connected by an arc then they can occur independently of each other. However there can exist indirect dependency between events like, for example, between events 3 and 6. They are not directly dependent but clearly event 6 can occur only after event 3.

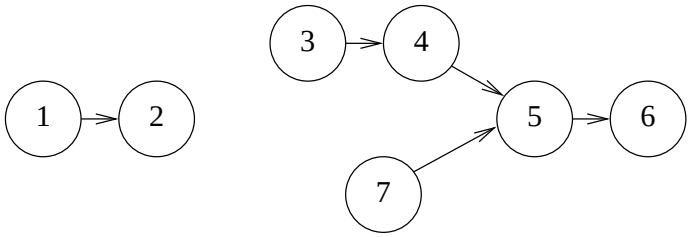


Figure 3: Partial order graph

A partial order graph can be represented with a boolean matrix $R:\ E \times E \rightarrow \{1,\ 0\}$ such that $R[i, j] = 1$ if there is an arc in the graph directed from i to j i.e. if event i should occur before event j; otherwise $R[i, j] = 0$.

Let $R*$ denote transitive closure [1] of matrix R, i.e. it should satisfy the following two conditions:

1. $\forall i, j \in E,\ R[i, j] \Rightarrow R*\,[i, j]$;
2. $\forall i, j, k \in E,\ R*\,[i, j] \wedge R*\,[j, k] \Rightarrow R*\,[i, k]$ - transitivity condition.

In other words $R*\,[i, j] = 1$ if there is a direct and/or indirect dependency between events i and j and event i should occur before event j. Note, that R itself is not a proper partial order relation as it does not necessary satisfy the transitivity condition while $R*$ is a partial order relation [3].

A partial order matrix R is *consistent* if it satisfies the following two conditions:

1. $\forall i \in E,\ R[i,i] = 0$ - no event can precede itself;
2. $\forall i \in E,\ R*[i,i] = 0$ - there are no cyclic dependencies in the partial order matrix.

Partial order matrix for the graph in Figure 3 and its transitive closure are shown below:

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \quad R* = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

The consistency conditions ensure that there is a possible event schedule that satisfies the the order requirements.

2.2 Total order

Total order is such a partial order that the following *totality* condition is held: $\forall i, j \in E, i \neq j,\ R[i,j] \oplus R[j,i] = 1$. In other words there is a direct order dependecy between any pair of events. Total order naturally arises in some applications and knowing the fact that a given partial order is total can simplify the TSE logic.

So a total order matrix R is consistent if it satisfies the following three conditions:

1. $\forall i \in E,\ R[i,i] = 0$ - no event can precede itself;
2. $\forall i \in E,\ R*[i,i] = 0$ - there are no cyclic dependencies in the partial order matrix;
3. $\forall i, j \in E, i \neq j,\ R[i,j] = \overline{R[j,i]}$ - totality condition.

3 Transition Sequence Encoder

Given a consistent partial order matrix $R:\ E \times E \rightarrow \{1,\ 0\}$ specifying the events order, the *Transition Sequence Encoder* (TSE) circuit is to generate request signals $req[k],\ k \in E$ in the specified order as shown in Figure 2. The following two chapters derive TSE analytically in form of a Boolean equation and then translate it to gate-level implementation.

3.1 TSE for partial order

Request signal $req[k]$ can only be generated when the acknowledgement signals $ack[j]$ have been received for all the preceding events j (such that $R[j,k] = 1$). This leads to the following equation for signal $req[k]$ generation:

$$req[k] = \bigvee_{P \subseteq E} \left(\bigwedge_{j \in P} R[j,k] \cdot \bigwedge_{j \in E \setminus P} \overline{R[j,k]} \cdot \bigwedge_{j \in P} ack[j] \right)$$

Here set $P \subseteq E$ is the set of preceding events for event k (this is guaranteed by terms $\bigwedge_{j \in P} R[j,k] \cdot \bigwedge_{j \in E \setminus P} \overline{R[j,k]}$). Term $\bigwedge_{j \in P} ack[j]$ guarantees that the acknowledgment signals for events in P have been received.

To simplify the above equation it is possible to fold it into conjunction of simple clauses:

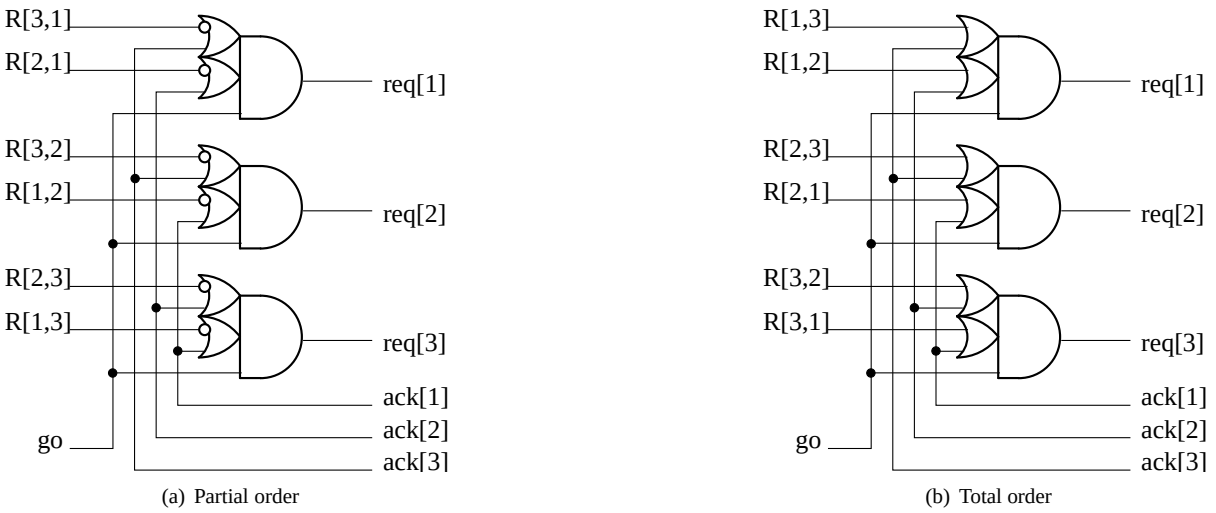
$$req[k] = \bigwedge_{j \in E} \left(R[j,k] \cdot ack[j] + \overline{R[j,k]} \right)$$

This can be further simplified using Boolean algebra to:

$$req[k] = \bigwedge_{j \in E} \left(ack[j] + \overline{R[j,k]} \right)$$

Note that this equation contains a redundant term when j equals k: $ack[k] + \overline{R[k,k]}$ that is equal to 1 because $\overline{R[k,k]} = 1$. It doesn't affect the correctness of the equation but of course in physical circuit implementation it will be omitted.

The obtained solution can be mapped to gate-level implementation. An example of TSE circuit for 3 events is shown in Figure 4(a). Signal go was added which is a general "ready" signal that prompts the circuit to start generating requests.



(a) Partial order (b) Total order

Figure 4: TSE gate-level implementation

3.2 TSE for total order

The final equation obtained for the strict partial order can be slightly modified taking into account totality condition ($\forall i, j \in E, i \neq j,\ R[i,j] = \overline{R[j,i]}$) and removing the redundant term when j equals k:

$$req[k] = \bigwedge_{j \in E} \left(ack[j] + \overline{R[j,k]} \right) = \bigwedge_{j \in E,\, j \neq k} (ack[j] + R[k,j])$$

Using this modification it is possible to get rid of inverters and come up with a circuit shown in Figure 4(b).

4 Conclusions

The paper introduced the concept of a programmable event scheduler able to dynamically schedule a set of events given their order specification. Transition Sequence Encoder is presented as a solution for a simplified version of a programmable event scheduler which can handle only acyclic order dependencies and single occurrences of a particular event. Analytical solution in Boolean equations as well as gate-level implementations are derived. The solution of the general event scheduling problem is the point of the future research.

Acknowledgement

This work is supported by EPSRC project NEGUS at the University of Newcastle upon Tyne.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[2] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *IEICE Trans. Inf. & Syst.*, volume E80-D, pages 315–325, March 1997.

[3] Art Lew. *Computer Science: A Mathematical Introduction*. Prentice-Hall, 1985.

[4] Danil Sokolov. *Automated synthesis of asynchronous circuits using direct mapping for control and data paths*. PhD thesis, University of Newcastle upon Tyne, 2005.