Asynchronous Scheduling and Allocation

Anatoliy Prihodzhy
University of Informatics and Radioelectronics, 6 P.Brovka, Minsk 220027, Belarus

Abstract

This paper presents an approach to generating asynchronous schedules of various concurrency levels and describes novel net-based scheduling and allocation optimization techniques for asynchronous high-level synthesis. The asynchronous schedules are optimized through the sets of concurrent variable and statement pairs. Experimental results and a comparison of the net-based techniques with the best sequential scheduling and allocation techniques are presented.

1: Introduction

Asynchronous circuits inherently data driven, are active only when they do useful work and allow low power consumption, timing fault tolerance, and high-speed operation with an average delay instead of a worst-case delay [1,2,6]. Techniques for synthesis of asynchronous circuits have been proposed at logic, high, and system levels. The Petri Net is the most universal model to specify an asynchronous behavior. The Signal Transition Graph model is a Petri Net the transitions of which are interpreted as value changes on circuit signals. The Predicate/Transition Nets are a system level specification model that deals with concurrency and covers both control and calculation. Currently the majority of high-level synthesis tools target at the synchronous RTL-structures [3-4] and perform scheduling that introduce control steps and finite state machine states. The Tangram high-level VLSI programming language supports the entire automatic compilation into asynchronous circuits.

2: Asynchronous schedules

The asynchronous schedule is a four-tuple $AS=(V,S,F,M_0)$ where $V$ is a set of variables, $S$ is a set of statements, $N=S\cup V$ is a set of nodes, $F\subseteq N^2$ is a flow relation, and $M_0\subseteq F$ is the initial marking. The edges in $F$ may be labeled with Boolean variables that regulate the token flow. The node $i$ input-edges set and output-edges set are denoted by $i^\prime$ and $i^\prime\prime$. Statement-node $i$ is enabled if $i\subseteq M$ and variable-node $j$ is enabled if $j\cap M\neq\emptyset$. An enabled node fires, removing a token from edges in $i^\prime$ and adding a token to edges in $i^\prime\prime$. Two nodes $i$ and $j$ are concurrent if marking $M$ exists for which $i^\prime\cap M\neq\emptyset$ and $j^\prime\cap M\neq\emptyset$, otherwise the nodes are sequential. The sequential nodes may share the same resources, while the concurrent nodes may not.

The asynchronous pipelined schedule of maximum concurrency is derived from CDFG. A pair of request/acknowledge signals (Figure 2) is introduced for each pair statement/output-variable and input-variable /statement (Figure 1). All the request edges except the incoming edges for input variables and the acknowledge outgoing edges for the input variables belong to the initial marking. Asynchronous schedules of less concurrency are derived from the maximum con-cur-rency schedule by adding and/or removing edges and tokens. This mechanism allows generating a less pipelined schedule, reducing a pipelined schedule to a nonpipelined one (Figure 3), and decreasing the concurrency level of the nonpipelined schedule. Concurrency relations $C_r$ and $C_i$ for the variables and statements define the maximum sets $D_r^M$ and $D_i^M$ of concurrent variable and statement pairs. A less concurrent schedule is defined by subsets $D_r^M \subseteq D_r^M$ and $D_i^M \subseteq D_i^M$. The schedule execution time is characterized by value

$$T_D=(1|U,D|)*\sum_{u\in U, D} t_{f(u)}$$

where $D=D_r^M|U,D|$ is the cardinality of the clique set of graph $G_D=(N,D)$ and $t_{f(u)}$ is the average execution time of the functional, storage, and interconnection units associated with statement $u$. The schedule cost is

$$S_D^{fu} = \sum_{j=1}^{N_{fu}} \sum_{v \in V_D} s_j \cdot \max_{m_v} m_v$$

where $N_{fu}$ is the number of functional unit types, $V_D$ is the set of cliques of graph $G_D=(N,D)$, and $m_v$ is the number of functional units of type $j$ needed to execute the clique $v$ statements concurrently. The number of storage units is estimated through the concurrency relation $C_r$, and the number of interconnection units is estimated through the maximum number of different variables in a set $V_D$ clique.

```
loop
  R := (X < A); --1
  exit when not R; --2
  C := X + (2* X); --3
  B := U * DX; --4
  D := B * C; --5
  G := U * D; --6
  E := Y * DX; --7
  H := E * (2* E); --8
  U := G * H; --9
  X := X + DX; --10
  Y := Y + B; --11
end loop;
```

Figure 1: VHDL-behavioral description

```
Figure 2: Maximum time concurrency schedule
```
3: Optimizing an asynchronous schedule

I decompose the high-level synthesis problem into four subproblems: to optimize the concurrency level, to solve the existence problem, to generate a schedule, and to map the schedule onto an asynchronous RTL-structure. To find set $D$, two optimization tasks are considered: P1: $\min \{ T_D \} S_D \leq S_0$ and P2: $\min \{ T_D \} T_D \leq T_0$ where $S_D$ and $T_D$ are the bounding cost and time. The pairs are consecutively added to $D$ while solving P1 and the pairs are consecutively removed from $D$ while solving P2. The selection of the pair to be added or removed depends on the order of pairs in maximum set $D_M$ and on the contents of current sets $D$, $U_{-D}$, and $V_D$. The pairs are ordered on the freedom for a statement to execute concurrently with other statements.

4: The schedule existence problem

To solve the existence problem is to prove that an asynchronous schedule exists which realizes the given behavior correctly and has the concurrency level defined by sets $D'$ and $D'$. The cyclic schedule existence problem is formulated in a matrix equation form to prove matrices $F$ and $M_D$ exist and define a correct live and safe net. The maximum concurrency noncyclic nonpipelined schedule is described by a statement precedence relation $Q$ [5]. Set $D$ is defined by matrix $Q_D$ in which Boolean variable $x_{ij}$ for $(i,j) \in D$ defines whether $i$ precedes $j$ ($x_{ij} = 1$) or $j$ precedes $i$ ($x_{ij} = 0$). A noncyclic schedule exists if the following combined logical equation has a solution: For $i,j,k \in S$

$$L_1 = + (x_{ij} \oplus x_{jk}) + (x_{ij} \equiv x_{jk}) + (x_{ij} \oplus x_{jk}) = 0 \quad \text{for } (i,j) \in D$$

$$L_2 = + (x_{ij} \oplus x_{ik} \oplus x_{ij}) + (x_{ij} \equiv x_{ik} \equiv x_{ij}) = 0$$

$$i < j \land k < j \land (i,j) \in D \land (i,k) \in D \land (j,k) \notin D$$

If $L_1$ and $L_2$ have no solution, $D$ must be modified. In [5] $L_1$ is represented as labeled graph $G_D$ which nodes are variables labeled $\theta$ and $1$ and edges are pairs of variables connect-ed $\oplus$ and $\equiv$. Whether a solution exists or not, depends on features of graphs $G_D$ and $G_D'$. If $L_1$ and $L_2$ have no solution, $D$ must be modified. In [5] $L_1$ is represented as labeled graph $G_D$ which nodes are variables labeled $\theta$ and $1$ and edges are pairs of variables connect-ed $\oplus$ and $\equiv$. Whether a solution exists or not, depends on features of graphs $G_D$ and $G_D'$.

5: Generating an asynchronous schedule

Two types of conflicts are possible for $L_1$ and $L_2$ [5]. To find a graph $G_D'$ optimal labeling and use target function $f = \alpha \varphi + \beta \varphi' + \gamma \varphi + \delta \varphi''$ where $\varphi$ ($\varphi'$) is the number of conflict pairs or nodes that increase (do not increase) the execution time or cost, $\varphi''$ ($\varphi'''$) is the number of conflicts associated with the pairs and nodes, and $\alpha$, $\beta$, $\gamma$, and $\delta$ are factors. Depending on the factor values, the number of conflicts, conflict pairs, conflict nodes, and the schedule critical path are minimized. To generate the statement direct precedence relation $H_0$, value $1$ in position $(i,j)$ of the precedence matrix $Q_D$ is replaced with $0$, if the Boolean multiplication of row $i$ and column $j$ equals $1$. The resulted schedule includes all the variable- and statement-nodes and for each pair $(i,j) \in H_0$, the edge that connects the statement $i$ output variable to the statement $j$ (Figure 4).

6: Results

Experimental results obtained on a PC 486/50 for the fifth-order wave filter [4] are presented in Table 1. Some of the asynchronous schedules have the critical path shorter than the critical path of feasible sequential schedules generated by the ALPS system (ILPF). The average path length is 18% less than the critical path length.

Acknowledgments

Submitting this paper for DATE98 has been encouraged by professors Bernard Courtois, Wolfgang Nebel, Jean Mermet, and Franz Ramming. The author is grateful to them.

References


Table 1. Schedules generated for the fifth-order filter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adders / Multipliers</td>
<td>1 / 1</td>
</tr>
<tr>
<td></td>
<td>2 / 1</td>
</tr>
<tr>
<td></td>
<td>2 / 2</td>
</tr>
<tr>
<td></td>
<td>3 / 2</td>
</tr>
<tr>
<td></td>
<td>3 / 3</td>
</tr>
<tr>
<td>ALPS: cycles</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>17</td>
</tr>
<tr>
<td>AHILES: Set D</td>
<td>39</td>
</tr>
<tr>
<td>Critical path</td>
<td>28</td>
</tr>
<tr>
<td>Average path</td>
<td>22.6</td>
</tr>
<tr>
<td></td>
<td>17.9</td>
</tr>
<tr>
<td></td>
<td>17.5</td>
</tr>
</tbody>
</table>