Design Representation And Transformation For High Level Synthesis In Resource Constrained System

M. Sangeetha[1] C. Tharini[2] J. Raja Paul Perinbam[3]
[1&3] School of Electronics and Communication Engineering , College of Engineering , Anna University, India
[2] Electronics and Communication Engineering Department, Crescent Engineering College, Chennai, India.

Introduction

High-Level Synthesis is the process of mapping a behavioral description at the algorithmic level to a structural description in terms of functional units, memory elements and interconnections (e.g. multiplexers and buses). The intermediate format for High-level synthesis can be represented in Control Data Flow Graph (CDFG), Data Flow Graph (DFG), Control Flow Graph (CFG), Finite State Machine (FSM), Finite State Machine using Data path (FSMD) or Codesign Finite State Machine (CFSM). **The primary goal is to suggest a new theoretical approach to Hardware/Software codesign partitioning and scheduling for a Resource-Constrained System.** The approach is based on data path for CDFG model that capture the design information from the source file specified by VHDL language from its equivalent separate Control Flow Graph and Data Flow Graph. This paper discusses helpful optimization method for HW/SW partitioning and High-Level Synthesis tools.

Design Methodology For High Level Synthesis

- Main steps involved in the high level synthesis of a digital system are:
  - **Description of the behavior of the system.**
  - **Translation of the description into a graph (eg. CDFG)**
  - **Partitioning the system behavior into Hardware and Software Module**
  - **Operation scheduling. Here each operation in the Graph is assigned to a control step.**
  - **Allocate the resources for the digital system. Here the resources can be function units assigned to execute operation derived from the graph CDFG.**
- In most methodologies, this RTL network is then submitted to logic synthesis for gate level optimization that attempts to produce a design satisfying certain area and delay constraints. Clearly, the quality of the final result depends on the quality of the two tools.

- In order to produce an efficient RTL network, HLS has to estimate or compute the effect that a given high-level algorithmic decision will have on the final gate level network. This effect is translated into costs, which are used in most HLS algorithms, such as scheduling, allocation, and resource sharing.
- In most Scheduling/allocation algorithms, the costs are usually based on the number of states and number of resources. These metrics give a rough indication of the complexity and performance of the finite state machine (FSM) and data path area of the final design.
- A system is described in VHDL and transformed using CDFG. The CDFG is represented using FSMD to calculate the delay constraints for the system design. The next level involves the partitioning of the system into Hardware (HW) and Software model.
- **Often used model can be partitioned as Hardware model and control unit can be designed as Software model and verified for its functionality and specification after scheduling.** However, they almost completely ignore important aspects such as the size and delay of the control logic.
- The scheduling step and the hardware allocation are two important subtasks. These subtasks are not independent. To obtain an optimal design a system should perform both subtasks simultaneously. But many systems perform them separately or apply iteration

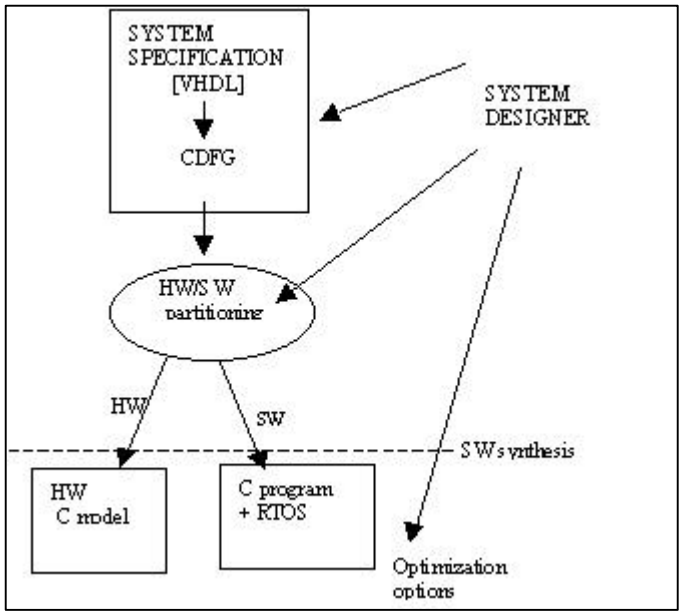

Figure 1 HW/SW Synthesis model

Control Data Flow Graph

- Represents the specification of the design at a very different level than the final hardware implementation.
- Nodes representing hardware operators such as adders and subtractors
- It usually does not contain any explicit specification of the multiplexers and

control logic required by the implementation.

- An edge (or a node) in the CDFG is used to represent a value, but in hardware this value may become a simple net or a register, depending on the schedule.

Adder node in the CDFG

- may be mapped onto an adder or functional unit in the RTL network, which in turn may be expanded into gates by logic synthesis and optimized with the surrounding logic. Hence, it might be inaccurate to consider simply the area and delay of an adder.

- Final implementation (and cost) of a given CDFG node/edge is not really known after HLS or even after logic synthesis, it is very difficult to measure hardware costs accurately during HLS.

- Main reason is that these costs are computed on a representation that is closer to the language level than it is to the hardware level that it is trying to measure.

HLS and logic synthesis operate on different representations

- makes it very inefficient for the two domains to interact.
- In today's systems this would require HLS to finish synthesis completely, and then logic synthesis would process the controller in the RTL network. This is a time consuming and inefficient approach.

If the two tools could operate on the same internal representation this problem would be resolved.

- DFGs exist in many different forms.

- Scheduling, it would help to know the exact size and delay of the resulting optimized control logic.

Graphical Representation of System

- CDFG generated for a simple system in VHDL as shown in the Figure 2, which consists of separate control-flow and data-flow graphs.
- The control-flow graph (CFG) in the Figure 3 represents the sequencing of operations as described in the language specification.
- The data-flow graph (DFG) in the Figure 4 represents the data-dependencies among the operations and values.
  - The numbers within round in Figure 4 represents the lines of execution in the source file and the number with # represents the value assignment in the particular node.
  - Node A and Node B deals with data operations and value assignment.

- The data path for CDFG model is designed using adders, multiplexers and demultiplexer from separate Control Flow Graph and Data Flow Graph as shown in figure 5.
- To optimize the logic it is further partitioned and scheduled using algorithms.

```
library ieee; use ieee.std_logic_1164.all;
Entity simple is
Port ( clock : in std_logic;
in1, in2, m1 : in integer range 0 to 3;
m2 : in Boolean;
out1, out2, out3, out4 : out integer range 0 to 3 ) ;
end simple;

Architecture behavior of simple is
Begin
Process (clock)
Variable A,B : integer range 0 to 3;
  Begin
if (clock'event and clock = '1') then
B := 0;                                  -- 00
    Case (m1) is                         -- 01
    When 0 =>  B:= in1 + in2;              -- 02
     A := A + B ;                          -- 03
    When 1 => B:= 3;                       -- 04
    When 2 => A:= in2;                   -- 05
    When 3 => A := 3;                    -- 06
          B := 2;                        -- 07
   End case;                             -- 08
Out1 <= A;                               -- 09
     if  (m2) then                           -- 10
          b := in1;                      -- 11
     Else out2 <= B;                     -- 12
Out3 <= A + in1;                           -- 13
     End if;                             -- 14
     Out4 <= A+B;                        -- 15
end if;
End process;                                -- 16
End behavior;
```

Figure 2 A simple system in VHDL

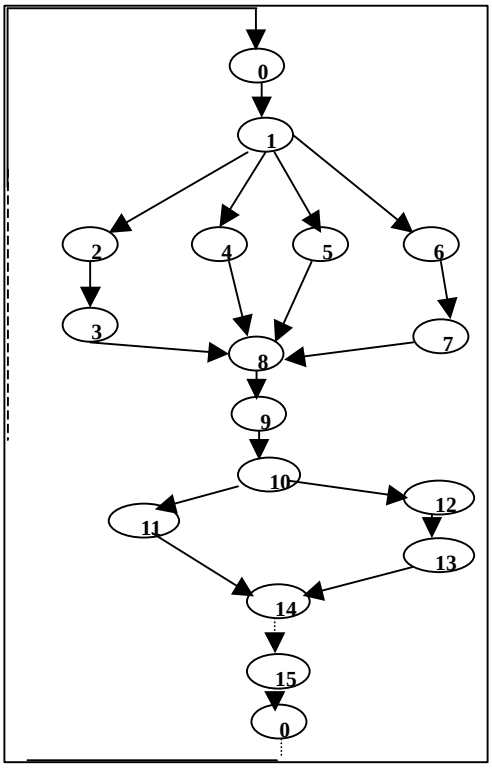

Figure 3 Control Flow Graph

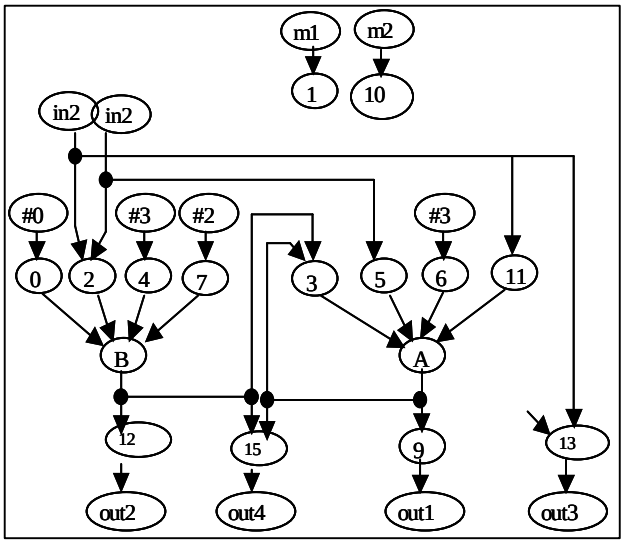

Figure 4 Data Flow Graph representation

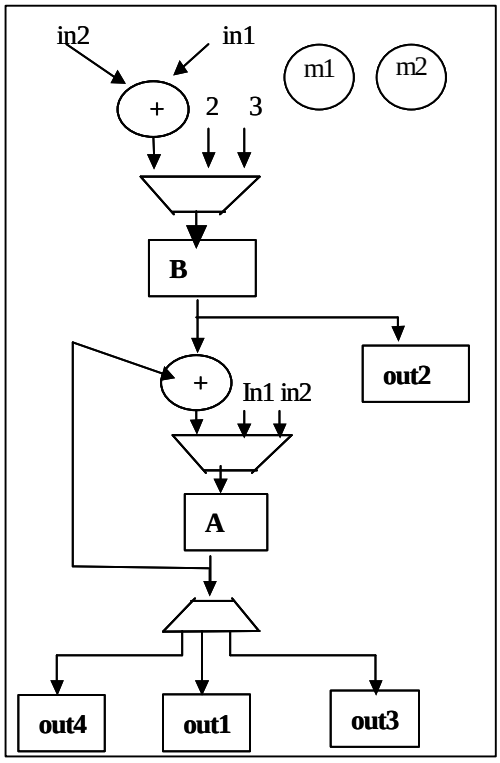

Figure 5 Data path for Control Data

```
k ← 0;
G_c^k(V_c^k, E_c^k) ← G_c(V_c, E_c);
While (E_c^k ≠ φ) {
" find (v_i, v_j) ∈ E_c^k with largest set of common neighbors";
N ← set of common neighbors of v_i and v_j;
s ← i U j;
V_c^{k+1} ← V_c^k U (vs) \ (v_i, v_j);
E_c^{k+1} ← φ;
for each (v_m, v_n) ∈ E_c^k
if (v_m ≠ v_i ≠ v_m ≠ v_j ≠ v_n ≠ v_i ≠ v_n ≠ v_j)
E_c^{k+1} ← E_c^{k+1} U {(v_m, v_n)};
for each v_n ∈ N
E_c^{k+1} ← E_c^{k+1} U {(v_m, v_s)};
k ← k + 1;
}
```

Figure 6 A heuristic to compute the clique partitioning of a graph

Clique Partitioning

- This partitioning can be applied to almost any subproblem of the task assignment task.
- The probem of finding the clique partitioning of a graph is NP complete.
- **Super Vertex**
  - It is based on combining vertices in the compatibility graph step by step. The vertices are called super vertices.
  - The index *i* of a supervertex v_i represents the set of indices of the vertices from which the supervertex was formed. For example, combining vertices v_1 ,v_2 and v_3 gives a supervertex v. The intermediate graph after k steps of the algorithm is called $G_c^k(V_c^k, E_c^k)$, the initial graph $G_c^0(V_c^0,E_c^0)$, is equal to the compatibility graph.
- **Role of Super Vertex**
  - The algorithm looks for a pair of supervertices with the largest number of common neighbors. A supervertex v_n ? V_c^k is a common neighbor of the supervertices v_i , v_j ? V_c^k, if both edges (v_i , v_n) and (v_j , v_n).
  - The two supervertices are combined into a new supervertex as described in the pseudocode of figure 6. The new supervertex remains connected to the common neighbors only.
  - The algorithm goes on to combine are included in E_c^k supervertices until the graph $G_c^k(V_c^k, E_c^k)$ has an empty edge set.

- Clique partitioning can be used for both nonoverlapped and overlapped scheduling.

Mobility based Scheduling

- In ASAP ("as soon as possible ") scheduling computes the earliest time at which an operation can be scheduled
- ALAP ("as late as possible") can also be computed by adapting the longest path algorithm to work from the outputs backwards.
- Combining the information obtained in bothways of scheduling algorithm gives rise to more powerful heuristics.
- ASAP scheduling time of node v_i is denoted by $s_S(v_i)$ and the ALAP time by $s_L(v_i)$, the interval $[s_S(v_i), s_L(v_i)]$ contains all possible time instants at which v_i can be scheduled. This interval is called the **time frame or the scheduling range of operation.**
- The length of the interval, i.e. $s_L(v_i)$ $s_S(v_i)$, is called the operation's mobility. A simple pseudocode for mobility based scheduling is described in the figure 7.

```
"determine? (0) by computing sS and sL";
k ⟵ 0;

while ( " there are unscheduled operations")
{
"schedule v at some time that optimizes the current
resource utilization";
"determine ? (k+1) by updating the scheduling ranges of
the unscheduled nodes";
k ⟵ k + 1
}
```

Figure 7 A simple mobility based scheduling algorithm

Conclusion

A modified data path for CDFG model is designed for HW/SW codesign, which represents the RTL network in graphical level representation. It is partitioned and scheduled to obtain optimized logic. The pseudocode described for the clique partitioning and scheduling algorithm elaborated leads to a more exact approach to the optimization of any resource-constrained system. Future work includes researches on generating CDFG model from C source file. Partitioning and Scheduling of the system is applied to the Control Data Flow Graph to generate optimized Hardware and Software model.

References:

1. Ohm, S.Y, Blough, D.M. Kurdahi, F.J 1996. High-level synthesis of recoverable microarchitectures. In Proceedings of the European Design and Test Conference (ED&TC 96), 11-14.

2. Qingsheng Wang, Hongxi Xue, Ming Su, Jinian Bian 1995. Behavioral description in VisualVHDL and its implementation.In proceedings of the International Conference on Oct. 1995, 361 -363 .

3. Qiang Wu, Yunfeng Wang, Jinian Bian, Weimin Wu, Hongxi Xue 2002. A hierarchical CDFG as intermediate representation for hardware/software codesign. In Proceedings of the International 1666 -1669 conference on Communications, Circuits and Systems and West Sino Expositions, 1429 -1432 vol.2.

4. Euiseok Kim, Jeong-Gun Lee, Dong-Ik Lee 1993. Building a distributed asynchronous control unit through automatic derivation of hierarchically decomposed AFSMs from a CDFG. In Proceeding of the International Conference on Advanced research in VLSI (ARVLSI 2001), 2 –15.

5. Said Amellal and Bozena Kaminska 1993. Scheduling of a Control and Data Flow Graph. In Proceeding of the International Symposium on Circuits and systems, (ISCAS'93), 1666 –1669.

6. Potkonjak, M. Dey, S. Roy, R.K 1995. Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , Volume: 14 Issue:9,1141 –1154.

7. Sabih.H. Gerez, eds. 1998. Algorithms for VLSI design Automation. Reading, . John Wiley & Sons.

8. Knudsen, P.V., Madsen. J. 1999. Graph based communication analysis for hardware/software codesign. In the (CODES '99) Proceedings of the Seventh International Workshop on Hardware/software Codesign, 3-5 May 1999, 131 –135.

9. Lei Tang, Shaojun Wei, Yunlin Qiu. 2001. Sub-graph matching based HW/SW co-design In the Proceedings of fourth International Conference on ASIC, 75 –78.

10. Raje,S.;Sarrafzadeh,M.1993.GEM:A geometric Algorithm for scheduling. In the IEEE International Symposium(ISCAS '93) on Circuits and Systems , 3-6 .

11. Wayne Wolf , eds. 2001. Computers as Components: Principles of Embedded Computing System Design. Reading,.Harcourt India Private Limited.