CIS 4930 001/CIS 6930 010: Digital Circuit Synthesis

Spring 2017 Final Project

Implementation of a Datapath Synthesis Tool

Dr. Srinivas Katkoori

Assigned on 22nd March 2017 (Wednesday).

Demo Date: 3rd May 2017 (Wednesday) 1pm – 5pm

This project carries 40 pts towards the final grade. Start early!

This project requires you to develop a datapath synthesis tool. You are provided with code of the clique partitioning algorithm implementation in C. You must develop several modules which can call the clique partionining code to build the datapath synthesis tool. If you are an undergraduate student, you can team up with upto **two more undergraduate students** in the class. If you are graduate student, you can team up with atmost **one more graduate student** in the class.

Figure 1 shows the overall system. The input to the synthesis tool must be a data flow graph (DFG) in the AUDI intermediate format (AIF). The following tasks must be executed in that order:

1. Operation Scheduling
2. Functional Unit Allocation and Operation Binding
3. Register Allocation and Carrier Binding
4. Multiplexor Generation
5. Datapath Generation

During the last step, you will generate a datapath in VHDL that can be validated by simulation. You will also generate a controller as a microprogram that can be used to set the control signals in the testbench. Note that you do not need to generate a controller design in VHDL.

We will outline in detail the work involved in each of the above steps.

1. **Operation Scheduling:** You must implement a resource constrained or a latency constrained scheduler. You can choose to implement one of the following scheduling algorithms:
   (1) List Scheduling.
   (2) Integer Linear Programming Based Scheduling.
   In case of (2), you need to *automatically* generate the constraints file and call an LP solver. The results of the scheduler must be then read by the synthesis tool. Note that the user will provide resource or latency constraints depending on the scheduling algorithm you have implemented.

2. **Functional Unit Allocation and Binding:** The number of functional units must have been determined either by the latency-constrained scheduler or is provided by the user. Given this information, you need to bind each operation to a functional unit.

   The pseudo-code for this task is given below:

```
for each type of operation T do
   G(V) <- NULL  // G(V,E) is a compatibility graph
   G(E) <- NULL
   // Build the compatibility graph
   for each operation op1 of type T do
        if(op1 is not in G) then
          G(V) <- G(V) U {op1}
        end if
        for each operation op2 of type T do
```

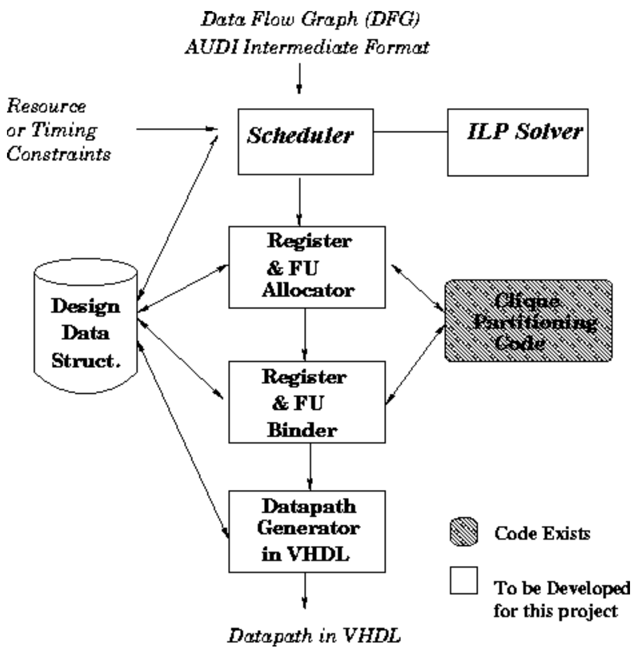

Figure 1: Synthesis Flow

```
          if(op2 is not in G) then
            G(V) <- G(V) U {op2}
           end if
          if( op1 != op2) then
               if( op1 and op2 are not scheduled in same tstep) then
                       G(E) <- G(E) U {(op1, op2)}
               end if
          end if
       end for
  end for
  C <- clique partition(G)    // Call Clique Partitioner
  for each clique c in C  do
     Bind all the operations in c to
       an unbound resource of type T
  end for
end for
```

3. **Register Allocation and Binding**

We need to bind a set of **compatible** edges in the dataflow graph. Two edges are said to be compatible if they have *non-overlapping lifetimes*. For each edge in the scheduled dataflow graph, we can determine its life time as follows: the lifetime begins with the first *write* into the edge and ends with the last *read* into the edge.

The pseudo-code for this task is given below:

```
  // Determine lifetimes of edges
  for each edge E in the DFG do
     determine its life time
  end for
   // Build the edge compatibility graph
```

```
G(V) <- NULL  // G(V,E) is a compatibility graph
G(E) <- NULL
for each edge e1 do
     if(e1 is not in G) then
       G(V) <- G(V) U {e1}
     end if
     for each edge e2 do
        if(e2 is not in G) then
          G(V) <- G(V) U {e2}
         end if
        if( e1 != e2) then
             if( lifetime(e1) intersection lifetime(e2) == NULL) then
                     G(E) <- G(E) U {(e1, e2)}
             end if
        end if
     end for
end for
C <- clique partition(G)    // Call Clique Partitioner
for each clique c in C do
   Allocate a new register R
   Bind all the edges in c to R
end for
```

4. **Multiplexor Generation:** Resource sharing gives rise to multiplexors in the design. There are two kinds of resource sharing: (a) register sharing; and (b) functional unit sharing. We need to examine this sharing and generate multiplexors accordingly. The pseudo-code is as follows:

```
// Register Sharing
  for each register R do
     Let n be the number of carriers mapped to R
     Generate a multiplexor M with n inputs
   end for

// Functional Unit Sharing
   for each FU do
      Let n be the number of operations bound to FU
      for each input of FU do
        Create a multiplexor with n inputs
      end for
   end for
```

5. **Datapath Generation in VHDL:** During this phase, you need to generate the datapath in VHDL. Examples of datapath, controller, will be discussed in the class later.

   The datapath has three parts:

   (a) **entity:** Besides the primary input and primary outputs, `reset`, `start`, and `clock` ports.
   (b) **component declaration:** You can either use the AUDI component library (available on the course homepage) or use the components you have developed in earlier assignments.
   (c) **architecture:** structural netlist of registers, adders, multipliers, and multiplexors.

Design Validation: The synthesized datapath must be validated by simulation in VHDL. The appropriate control signals in each clock cycle must be provided in the test bench. The controller generated as a microprogram must be used for this purpose. You need to demo your synthesis tool for atleast three DFGs.

Deliverables:

1. **C Code:** You need to submit your entire C code as a tar file on Cavnas after your demo.

2. **Code Demo:** Your team should demo your system on 3rd May 2017 in C4 Lab. A signup sheet will be posted on my door.

3. **Report:** A 2 page user document for your system and 1 page report of your experience in this project, work distribution etc. The report is due at the beginning of the demo.

Hints:

1. The Register allocation and binding task is similar to functional unit allocation and binding task. You can share code between these two tasks, if you organize your software intelligently.

2. Print the results of each task onto the screen. This way even if you do not finish the project, you will get credit for the working modules.

3. Follow the general coding guidelines:

- No function is more than 100 lines of code.
- Once you write a function test it immediately.
- Write comments liberally.
- Think of the algorithm of each task before you start coding. Preferably, have a hand worked example so that you can debug your code.
- Use `assert` statements in your code.
- Decide on the data structures completely before you start coding. This facilitates simultaneous development of the modules.

♢ Good Luck ♢