ECE 474a/574a Assignment 3

Assignment Points: 100; Up to 20 points extra credit.

DUE DATE: Tuesday, December 4, 11:59PM (Submit by Thursday, November 29 for 10% Extra Credit)

Overview

Create a program named ***hlsyn*** (high-level synthesizer) that will convert a C-like behavioral description into a scheduled high-level statement machine implemented in Verilog.

Command-line Arguments

Your program must be capable of utilizing a command-line argument to specify the input C-like file, a latency constraint specified in cycles, and the name of the output Verilog file.

```
hlsyn cFile latency verilogFile
```

Your program must ensure the user has correctly provided the required command-line arguments and display a usage statement if the provided arguments are incorrect.

C-like Behavioral Specification

A behavioral netlist file provides an acyclic connection of components. where:

- All empty lines should be ignored
- All lines beginning with "//" are considered comments and should be ignored
- The netlist file can be assumed to be fully space/tab delimited, i.e. one or more space or tab characters should appear between each token that needs to be parsed, including colons.
- Circuit inputs and outputs can be declared on a line using the formats:

```
input dataType inputName1, inputName2
output dataType outputName1, outputName2
```

- Valid data types include:
    - Signed Integers Types: `Int1`, `Int2`, `Int8`, `Int16`, `Int32`, and `Int64`
    - Unsigned Integer Types: `UInt1`, `UInt2`, `UInt8`, `UInt16`, `UInt32`, and `UInt64`
    - The number after `Int` and `UInt` specifies the width of the data input, output, register

- Variables must be explicitly declared using the format:

```
variable dataType regName1, regName2
```

- Statements are declared on a single line using the following formats for specific components.

```
o = a + b
o = a - b
o = a * b
o = a > b
o = a < b
o = a == b
```

```
o = sel ? i1 : i0
o = a >> sh
o = a << sh
o = a / b          // (ECE 574a only)
o = a % b          // (ECE 574a only)
o = a + 1          // (ECE 574a and for loop extra credit)
o = a - 1          // (ECE 574a and for loop extra credit)
```

- All names for inputs, outputs, and internal variables should be unique.
- All names for inputs, outputs, and internal variables are case sensitive and can consists of any number of letters or digits.
- Input, output, variable declarations should come before component instantiations.

if statements

Your program must provide support if and if-else constructs. The if statement will execute the sequential statements within the if statement if the conditional is true. The conditional is assumed to be true if the value is non-zero and false if the value is zero. The condition is a single value (e.g., reg2, a, o2) and not a comparison (e.g., a < temp). Your program may assume that braces (e.g., { }) will always be used for the if and/or else parts. The else part is optional.

```
if (condition) {
   // do something
}
else {
   // do something else
}
```

for loops (10% extra credit)

The for loop executes the `initialExpression` once at the beginning of the for loop. The for loop continues to execute as long as the `conditionExpression` is true, where the condition is checked at the beginning of each loop iteration. At the end of each iteration, the `updateExpression` is executed.

```
for (initialExpression; conditionExpression; updateExpression) {
   // do something
}
```

The condition is assumed to be true if the value is non-zero, and false if the value is zero. **For this assignment, the condition is one of the following: <, >, or ==.** Your program may assume that braces (e.g., { }) will always be used for the for loop. The `initialExpression` and `updateExpression` can be any of the statements defined above. For both 474a/574a students, the for loop must support the increment and decrement statements.

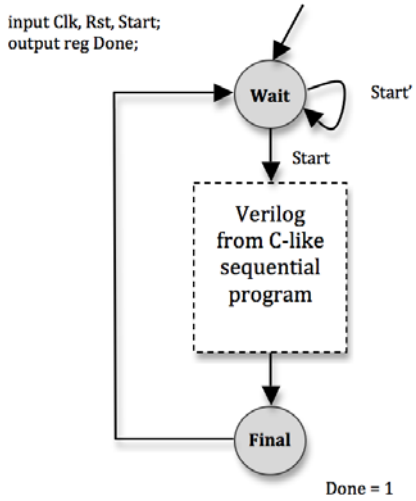
Verilog Code Generation

- The generated Verilog module should be named `HLSM` with `Clk`, `Rst`, `Start`, and `Done` as the first set of inputs/outputs as shown here:

```
module HLSM (Clk, Rst, Start, Done);
   input Clk, Rst, Start;
   output reg Done;

endmodule
```

- The resulting high-level state machine should include an initial `Wait` state and a final `Final` state used for controlling the execution of the resulting hardware description.



- The `Wait` state will wait for the `Start` input to be 1 before proceeding to execute the synthesized statements from the C-like sequential program.
- After the HLSM has executed the synthesized statements from the C-like sequential program, the `Final` state will assert the `Done` output to 1 for one cycle before returning to the `Wait` state.
- All inputs and outputs should be included in the model declaration in the order specified within the C-like sequential program.
- All outputs and variables should be declared as reg variables/outputs in the Verilog description.
- A single `State` register should be declared as a reg vector using the fewest number of bits possible to implement the resulting HLSM.
- The HLSM description should consist of a single procedure sensitive only to `posedge Clk`.
- Upon a reset indicated by the `Rst` input, all outputs and internal registers should be set to 0 and the `State` register should be assigned to the `Wait` state.
- The generated Verilog code must be synthesizable using Xilinx ISE or Vivado tools.

Scheduling Algorithm

- Available resource types include adder/subtractor, multiplier, logic/logical, and divider/modulo.
- The multiplier resource only supports multiplication.
- The adder/subtractor resource supports addition and subtraction.
- The divider/modulo resource supports division and modulo.
- The logic/logical resource type supports all other operations, including comparisons, shift, and ternary operators.
- Multipliers have a 2 cycle delay, divider/modulo have a 3 cycle delay, all other resources have a 1 cycle delay.
- The operations implemented within each cycle (determined by the scheduling algorithm) should be implemented within a separate state in the HLSM.

List_R (ECE 474a)

- Your program should create a one procedure high-level state machine description (HLSM) such that all sequential statements are scheduled using the List_R scheduling algorithm.

Force Directed (ECE 574a):

- Your program should create a one procedure high-level state machine description (HLSM) such that all sequential statements are scheduled using the Force Directed scheduling algorithm.

Submission Requirements

- All programming assignments must be implemented with C or C++ using the CMake cross platform make tools. CMake is a set of tools that provides an easy way to utilize different platforms and development environments (e.g. Mac OSX with XCode, Linux using Makefiles, Windows using Visual Studio, etc.). Using CMake, you are encouraged to utilize whatever development tools and platform you are most comfortable with.
- All programs will be compiled and graded using ece3.ece.arizona.edu: All assignments will be compiled and tested using the ECE department serv*er ece3 (ece3.ece.arizona.edu)*, on which the CMake tools are already available. You can access the ece3 server using any secure shell (SSH) client. You are strongly encouraged to test your programs using the ece3 server before submission.

Note: If you use your own computer for development, you will have to download and install the CMake tools.

- All assignments must be submitted as a single TAR/GZIP (.tgz) or ZIP (.zip) using the naming convention `NetID1_NetID2_hlsyn.tgz` or `NetID1_NetID2_hlsyn.zip`, where *`NetID1`* and *`NetID1`* are the UA NetIDs for each group member. The submitted archive should use the following directory and file structure:

```
NetID1_NetID2_hlsyn
   |
   ---> CMakeLists.txt
   |
   ---> README.txt
   |
   ---> src
         |
         ---> CMakeLists.txt
         |
         ---> .c/.h/.cpp files
```

Note: You can create a TAR/GZIP archive for your assignment using the following commands on the ece3 server.

```
tar cvzf NetID1_NetID2_hlsyn.tgz NetID1_NetID2_hlsyn
```

- All submitted assignments will be compiled using the following commands:

```
cd NetID1_NetID2_hlsyn
mkdir build
cd build
cmake ..
```

```
make
./src/hlsyn
```

- Do NOT include build files, executables, temporary files, etc. with your assignment submission.
- All assignments must be submitted as a single TAR/GZIP (.tgz) or ZIP (.zip) using the naming convention `NetID1_NetID2_hlsyn.tgz` or `NetID1_NetID2_hlsyn.zip`, where *`NetID1`* and *`NetID2`* are the UA NetIDs for each group member.
- Include a README.txt in your submission that specifies: 1) name and NETID of group members, 2) course you are enrolled in, 3) brief description of your program, and 4) contribution of each group member to the assignment.