Advanced Modeling -- Spring 2003 – Test 2 Solutions

Questions 1, 2 are non-optional. You may skip one question in questions 2-10 (cross out the question that you do not want graded).

1. (15 pts) Explain what role the Synopsys DesignWare libraries fulfill in the synthesis process. This is a DISCUSSION question, the more you can tell me, the better. Be sure to explain the general need for these type of library support in the synthesis process.

Arithmetic operators do not fit general Boolean synthesis methods very well because the design space in terms of area versus speed is too large. Design Ware components are structured implementations of arithmetic operators that are optimized for a particular implementation technology (FPGA, ASIC) that offer different area/speed tradeoffs. During the synthesis process, a particular implementation is chosen from those available based upon the area/speed constraints.

2. (15 pts) What is the difference between a high level synthesis tool (as represented by Synopsys Behavioral Compiler) versus a logic synthesis tool (as represented by Synopsys Design Compiler). This is a DISCUSSION question, the more you can tell me the better. You might want to use the 'compare and constrast' method, where you describe how they are the same, and how they are different.

A logic synthesis tool uses RTL as input which specifies the number of execution units, number of registers, and FSM control. Synthesis optimization choices are at a gate level and the choices are area/speed. The output is a gate level netlist.

A high level synthesis tool reads a behavioral description that only describes the operations to be performed. Based on external constraints such at latency and initiation period (both measured in clock cycles), RTL code is synthesized to meet these constraints. The HLL tool chooses the number of execution units, number of registers, and generates the FSM to perform the datapath control. The output of a HLL tool is RTL code.

3. (10 pts) Explain what it meant for Synopsys DesignWare component to be 'inferred' by a synthesis tool? BE VERY SPECIFIC.

Designware chooses a particular implementation (i.e ripple vs. carry-lookahead) for an arithmetic operation based upon the area/speed/technology constraints.

4. (10 pts) What is the purpose of the numeric_std (or std_logic_arith) library? What are some important type definitions in this package and why are they needed?

It defines interfaces for arithmetic operators such as +, - , * , / for the signed and unsigned types. These types are necessary because some operators required different internal implementations for signed vs. unsigned operations.

5. (10 pts) IEEE standard 1076.6-1999 defines the VHDL synthesizeable subset. What does this mean? Why is important to have such a standard? (give an example to back up your statement).

A standard synthesizeable subset is needed so that RTL code is portable across different vendors. Also, it is important to define how to write RTL code in such a way as to infer the correct hardware component. In the notes were shown two RTL examples of a process that simulated a DFF, but only one of them would produce a DFF if synthesized.

6. (10 pts) Give the general structure of the coding style required by Synopsys Behavioral Compiler. Explain why this coding structure was used.

The coding style used two loops – an outer 'reset' loop, and an inner computation loop. The operations in the synchronous reset loop were operations performed one time, before the system began the computations in the compute loop.

7. (10 pts) Synopsys Behavioral Compiler had two parameters that controlled synthesis -- latency and initiation interval. Clearly explain what each parameter means in terms of the hardware that is generated. If latency was set equal to initiation interval, what effect this did have on the hardware? If latency was not equal to initiation interval, what was the constraint on these values and what was the effect on the generated hardware?

Latency defined the number of clocks required from the start of the computation loop to the end of the computation loop body.

Initiation interval was defined as the number of clocks between the start of a new set of computations for the computation loop body.

If Latency = initiation interval, then there was only one computation loop body instantiated in hardware.

If Latency /= inititiation interval, then the number of parallel computation loop bodies representing overlapped computations was equal to Latency/ initiation interval (this had to be evenly divisible).

8. (10 pts) A constraint called ‘extend-latency’ could be used for Synopsys Behavioral Compiler – what did this constraint do? EXPLAIN.

This told BC that no constraints were placed on the number of clock cycles for the computation loop body resulting in a minimum hardware configuration.

9. (10 pts) What was a *superstate* in code used by Synopsys Behavioral Compiler? How were *superstates* specified? How did *superstates* map to clock cycles?

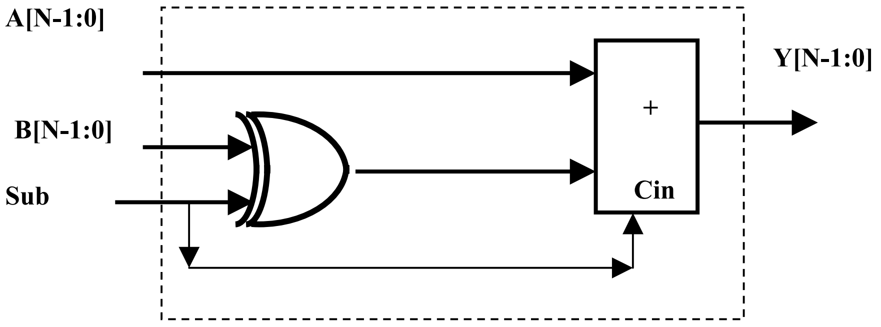
Each superstate was represented by a set of VHDL operations between ‘wait’ statements. A superstate could be mapped to 1 or more clock cycles, depending on the input constraints.

10. (10 pts) Use VHDL Generate statements to produce a gate level structural model of a adder/subtractor. The entity declaration is :

```
Entity  addsub is
   Generic map (N :  integer;
                       ADDSUB:  Boolean := FALSE);
  Port map (  A,B: in std_logic_vector( N-1 downto 0);
                  SUB: in std_logic;
                  Y: out std_logic_vector(N-1 downto 0));
```

If ADDSUB is false, just an adder should be generated (use a ripple carry architecture). If ADDSUB is TRUE, then an adder/subtractor should be generated. Assume you have the following two cells available:

XOR2 -- inputs are A,B, output is Y.
FA (full adder)- inputs are A,B, CI (carryin); outputs are S (sum), CO (carryout)



```
c(0) <= sub;
L1:  for  I in 0 to N-1 loop generate
          if (ADDSUB = TRUE) then
                u1: xor (A=> b(i), B=> sub, Y=>  b_tmp(i));

         else
                b_tmp(i) <= b(i);
        end if;
        u2: fa (A=> a(i), B=> b_tmp(i), ci=>c(i), co=>c(I+1),S=>Y(i))
     end generate;
```