L08 – Hardware Synthesis

For the recitation session for L08, instead of having tutorial problems, we will use a handout to go through various aspects of synthesizing circuits from Bluespec modules.

There are intentional blank pages left for you to practice drawing these synthesized circuits.

Modulo-4 counter Circuit

```verilog
module moduloCounter(Counter);
    Reg#(Bit#(2)) cnt <- mkReg(0);
    method Action inc;
        cnt <= {cnt[1]^cnt[0],~cnt[0]};
    endmethod
    method Bit#(2) read;
        return cnt;
    endmethod
endmodule
```

- I/O: Interface
- Instantiate state
- Compile inc
- Compile read

Systematically generate the hardware circuit
Streaming a function: Circuit

```
rule stream;
    outQ.enq(f(inQ.first));
    inQ.deq;
endrule
```

This is a sequential machine too!

- Connect the datapath
- Collect the ready signals
- Enable the called methods

Notice that `enq.en` cannot be True unless `enq.rdy` is true; `deq.en` cannot be True unless `deq.rdy` is true

Hierarchical sequential circuits
sequential circuits containing modules

Each module represents a sequential machine

Register inputs and outputs are replaced by method inputs and outputs
module mkGCD (GCD);
  Reg#(Bit#(32)) x <- mkReg(0); Reg#(Bit#(32)) y <- mkReg(0);
  Reg#(Bool) busy <- mkReg(False);

rule gcd;
  if (x >= y) begin x <= x - y; end // subtract
  else if (x != 0) begin x <= y; y <= x; end // swap
endrule

method Action start(Bit#(32) a, Bit#(32) b) if (!busy);
  x <= a; y <= b; busy <= True;
endmethod

method ActionValue#(Bit#(32)) getResult if (busy && (x==0));
  busy <= False; return y;
endmethod
endmodule
module mkMultiGCD (GCD);
    GCD gcd1 <- mkGCD();
    GCD gcd2 <- mkGCD();
    Reg#(Bool) turnI <- mkReg(False);
    Reg#(Bool) turnO <- mkReg(False);
    method Action start(Bit#(32) a, Bit#(32) b);
        if (turnI) gcd1.start(a,b); else gcd2.start(a,b);
        turnI <= !turnI;
    endmethod
    method ActionValue #((Bit#(32)) getResult);
        Bit#(32) y;
        if (turnO) y <- gcd1.getResult
        else y <- gcd2.getResult;
        turnO <= !turnO
        return y;
    endmethod
endmodule

interface GCD;
    method Action start (Bit#(32) a, Bit#(32) b);
    method ActionValue#((Bit#(32)) getResult);
endinterface
Streaming the GCD module

1. Write the code
2. Systematically generate the hardware circuit

Your drawing: