Lab 4: Greatest Common Divisor (GCD)

Acknowledgement: This lab is based on the course material provided by Prof. Arvind of MIT.

1 GCD Module using Euclidean Algorithm

The Euclidean Algorithm for finding the greatest common divisor (GCD) of two positive integers `X` and `Y` `GCD(X,Y)` uses only subtractions:

1. `GCD(0,Y)=Y`.
2. `GCD(X,0)=X`.
3. Otherwsie repeat the next two steps as often as necessary
4. If `X>Y` then `GCD(X,Y)=GCD(X-Y,Y)`.
5. If `Y>=X` then `GCD(X,Y)=GCD(X,Y-X)`.

Here is an example of calculating GCD(15,9) using the above algorithm:

```
GCD(15,9)
=GCD(15-9,9)=GCD(6,9)
=GCD(6,9-6)=GCD(6,3)
=GCD(6-3,3)=GCD(3,3)
=GCD(3,3-3)=GCD(3,0)
=3
```

Here is another example - GCD(7,12)

```
GCD(7,12)
=GCD(7,12-7)=GCD(7,5)
=GCD(7-5,5)=GCD(2,5)
=GCD(2,5-2)=GCD(2,3)
=GCD(2,3-2)=GCD(2,1)
=GCD(2-1,1)=GCD(1,1)
=GCD(1,1-1)=GCD(1,0)
=1
```

You can see that calculating `GCD(7,12)` and `GCD(15,9)` take different number of steps. This indicates that different number of clock cycles will be needed for calculating `GCD(7,12)` and `GCD(15,9)`. After inputting two positive integers (e.g., 7 and 12) to the GCD module, we should not input another two numbers (e.g., 15 and 9) until the first computation is complete.

2 Lab Assignment

In this lab, you will implement a GCD module in BSV using the Euclidean algorithm.
The interface for GCD module with lock, or LGCD, is given below (`gcd.bsv`):

```
interface LGCD;
    method Action start(int a, int b);
    method Bool busy();
    method int result();
endinterface
```

Method start(int a, int b) initiates a new computation when module is not busy. Method busy returns true if the module is computing GCD/busy otherwise false. Method busy is used by TestBench.bsv to ensure that the module is not busy before calling the start method. Method result returns the GCD for input a and b. If either of the input is non positive, then no computation is done (module will not go busy) and ERROR is printed (Use $display("ERROR") at proper place in your code). In order to implement the `busy` method, we will use a register `Reg#(Bool) bz` and initialize it to `False`.

```
module mkLGCD (LGCD);
    Reg#(int) x <- mkRegU;
    Reg#(int) y <- mkReg(0);
    Reg#(Bool) bz <- mkReg(False);
     ...
endmodule
```

Exercise

In `gcd.bsv`, complete `mkLGCD` module which implements `LGCD` interface. You must explicitly use `Reg#(Bool) bz` for implementation of `method Bool busy`.

You have to test your implementations using the following testbenches.

```
$ make gcdsimple
$ ./simGCDSimple
```

This calculates the GCD for 423 and 142.

```
$ make gcdseq
$ ./simGCDSeq
```

This calculates GCD(a,b) for a=1,...,7 and b=1,...,62.

```
$ make gcdzero
$ ./simGCDZero
```

This checks if your GCD module prints "ERROR" if zero input is used.

```
$ make lgcd
$ ./simLCGD
```

This checks if your GCD module correctly implements `method Bool busy`. We first call `start(32,7)` method, and we continuously check if `busy` is `True`. Until `busy` is false, we call `result`, which prints out the result.