Lab 4: Greatest Common Divisor (GCD)

Due: 11:59:59pm, Sunday May 14th, 2017

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. Otherwise 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.

First, make a copy of the source code using the following command

```
$ cp -r  /home/linux/ieng6/cs140g/public/lab4 .
```

The interface for GCD module with lock, or LGCD, is given below (`gcd.bsv`):

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

1. Method busy returns true if the module is computing GCD/busy otherwise false. In order to implement the `busy` method, we will use a register `Reg#(Bool) bz` and initialize it to `False`.
2. Method start(int a, int b) initiates a new computation when module is not busy. Method busy is used by TestBench.bsv to ensure that the module is not busy before calling the start method
3. Method result returns the GCD for input a and b
4. If either of the input is zero, then no computation is done (module will not go busy) and method isError will return True. It will return True until a new computation is started using the start method.

Busy method is called repeatedly (every cycle after start) to check if GCD module is busy or not. If the module is not busy, then next GCD computation is started using the start method. All the above methods are called by the Testbench. Note that GCD can take a variable number of clock cycles. Following is an illustration of how calls are made from the Testbench.

```
gcd.start(a1,b1)
gcd.busy()
gcd.busy()
gcd.busy()
gcd.busy()
gcd.result()
gcd.start(a2,b2)
gcd.busy()
gcd.busy()
gcd.result()
gcd.start(a3,b3)
gcd.busy()
gcd.busy()
gcd.busy()
gcd.busy()
gcd.busy()
gcd.result()
```

Exercise 1: 100 points

In `GCD.bsv`, complete `mkLGCD` module which implements `LGCD` interface. You can test your implementations using the following testbenches.

```
$ make gcd
$ ./simGCD
```

This calculates GCD(a,b) for a=0,...,32 and b=0,...,8.

3 Submission

For Exercise 1, please write your answers in `GCD.bsv`.Write your answers in `Lab4.txt`. Submission instructions are the same as Lab 1 (Piazza @ 68). Write down your name, PID and e-mail address in `Lab4.txt` located in `lab4/` directory. If you worked in a group, write down your partner's information as well. While your solutions can be identical, each group member must make their own submission. You will be scored based on your own submission. To submit your assignment, simply run the following command from `lab4/` directory:

To submit your assignment, simply run the following command from `lab4/` directory:

```
$ bundleP4 <your-email>
```

Submission verification

Refer to Lab 1.