ENSC 350: Digital Systems Design

Introduction to Lab 3

Instructor: Dr. Ameer Abdelhadi

Spring 2018

Lab 3

In this lab, you will create a slightly more complicated datapath.

- Unlike Lab 2, we won't give you the details of the datapath
- But this slide set will give you some hints…

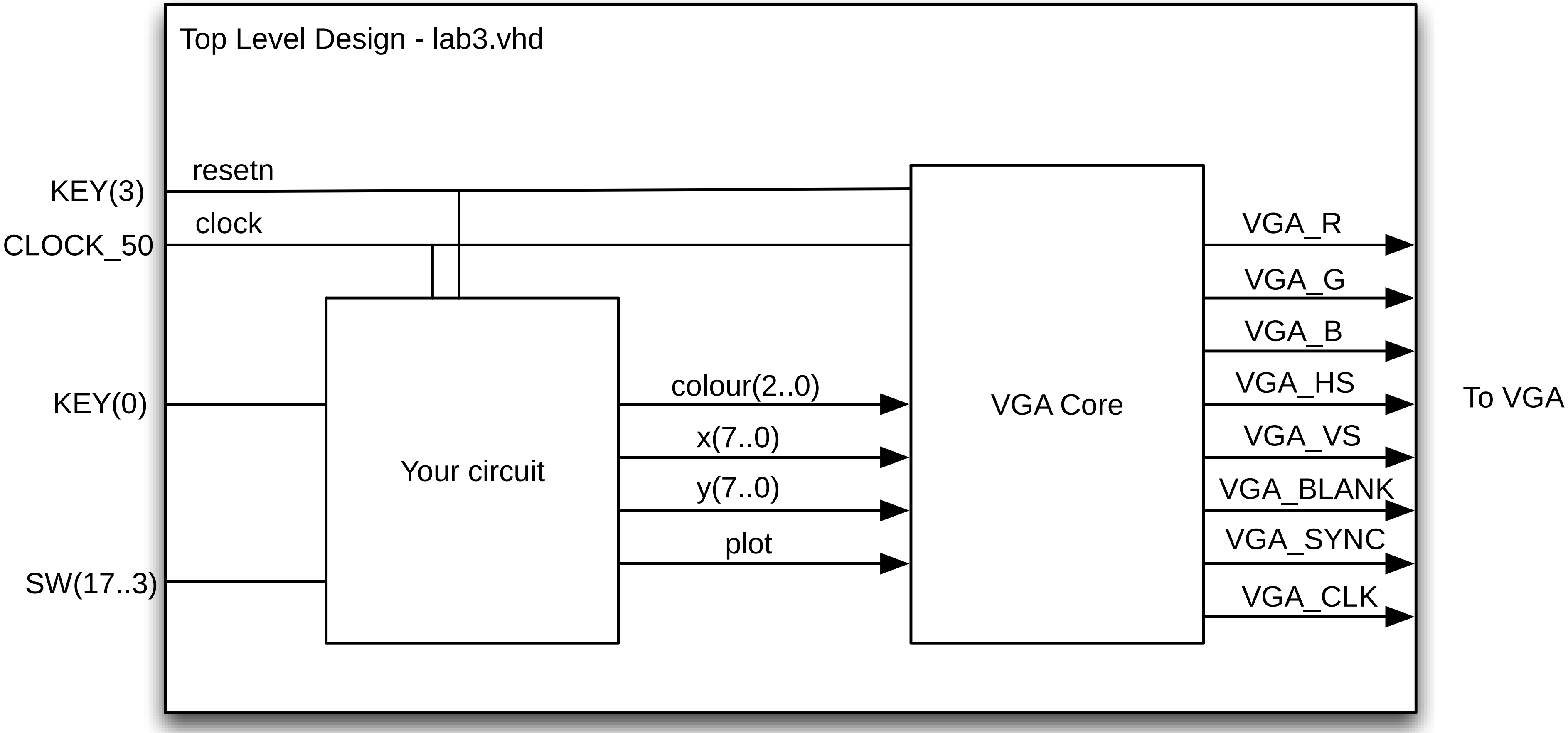
We will also use a VGA core to allow you to draw pixels to a VGA screen.

Step One: a circuit that draws lines connecting two arbitrary points on the screen.

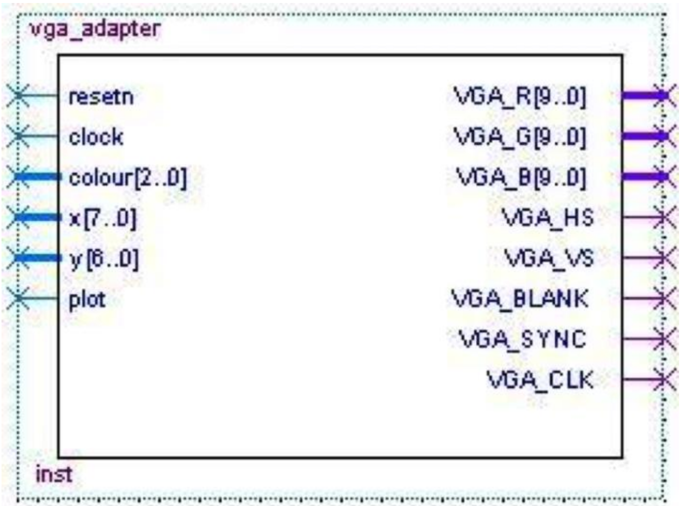
Step Two: a circuit that displays multiple lines connecting two arbitrary points on the screen at the same time and then sequentially.

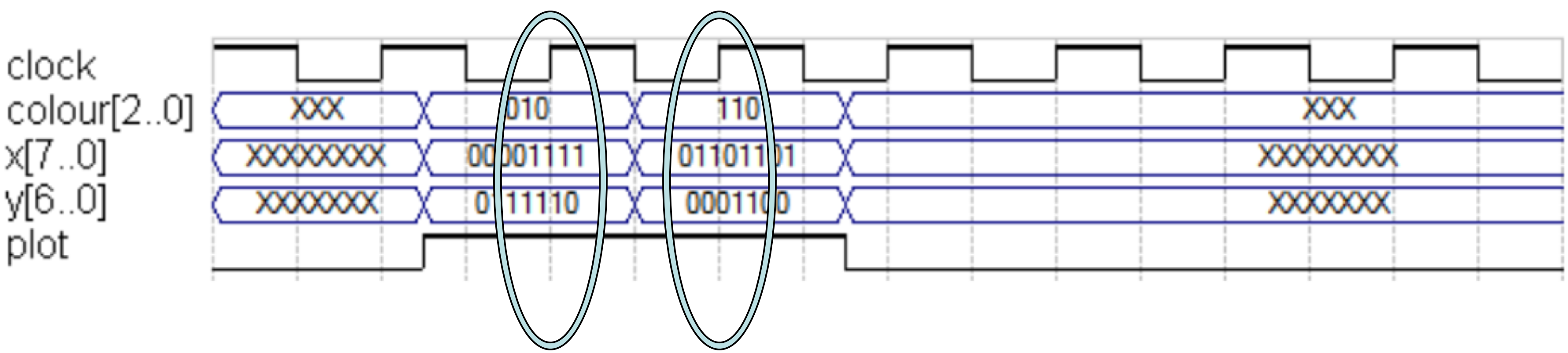
Step Three: a circuit that draws right angle triangles sequentially on the screen. (Challenge Task)

Overall Block Diagram

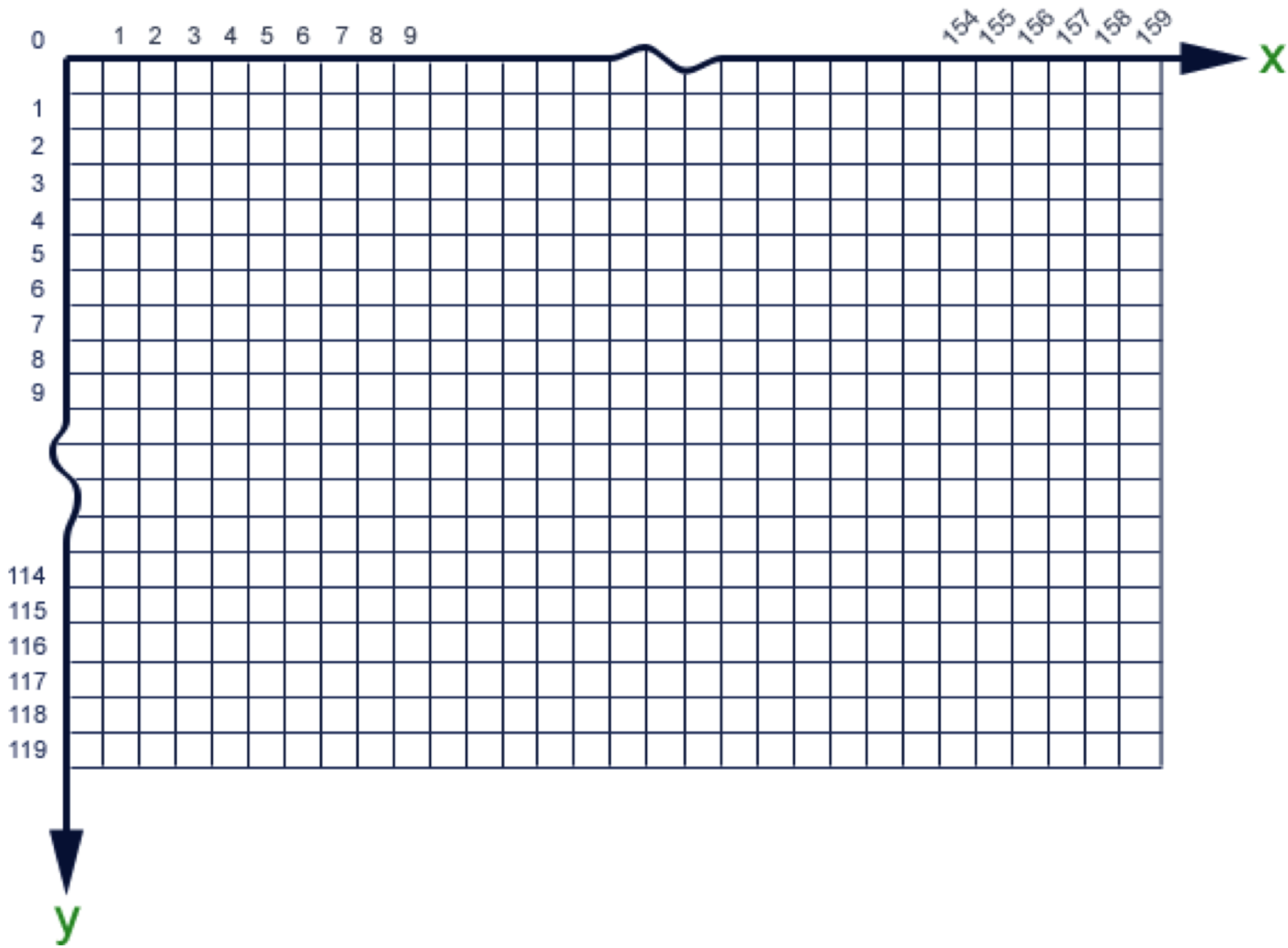


VGA Core – We will give this to you





This would turn on two pixels


0
1
2
3
4
5
6
7
8
9
154
155
156
157
158
159
x
1
2
3
4
5
6
7
8
9
114
115
116
117
118
119
y

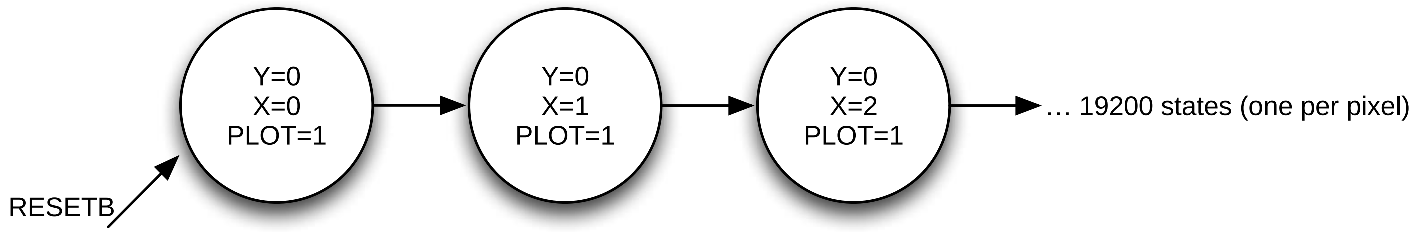
Task 2: Fill the Screen

Task 2: Fill the Screen

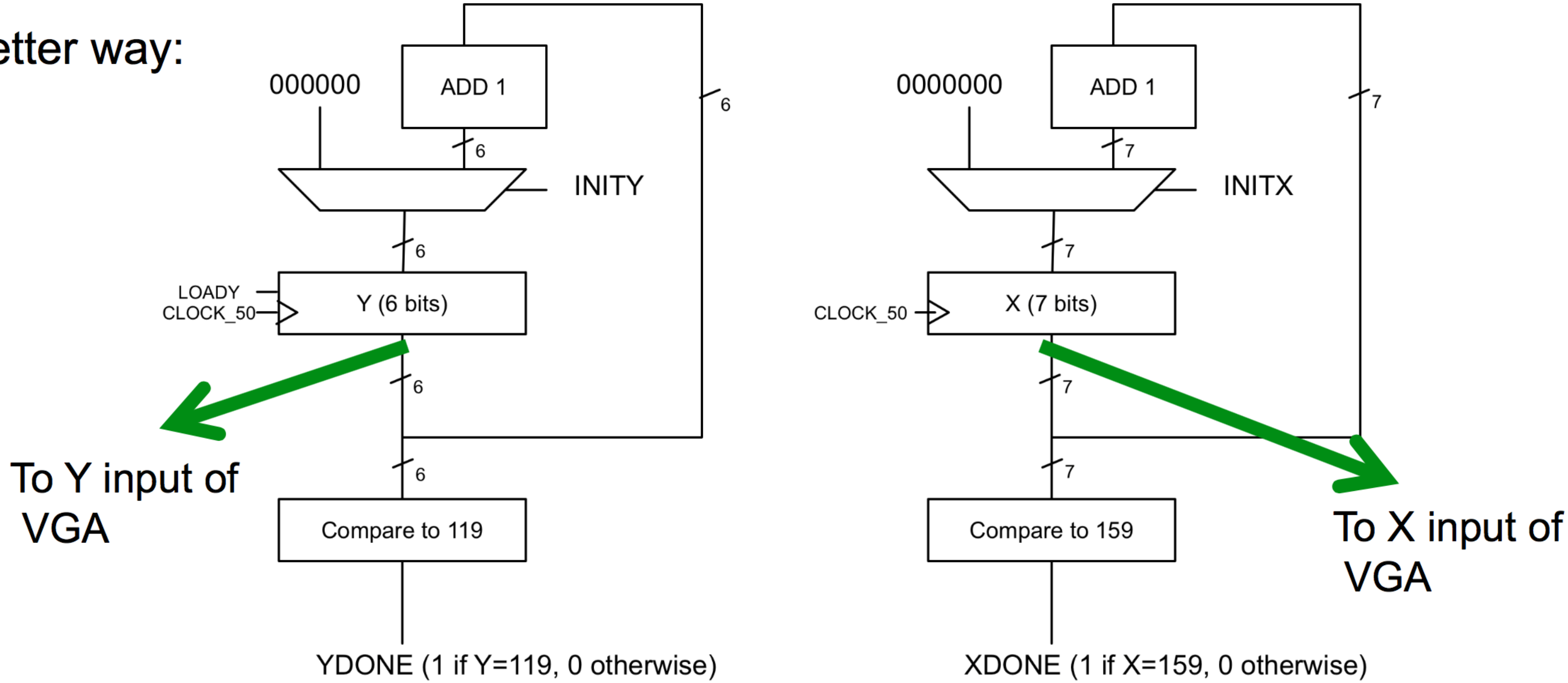
Turn on each pixel one at a time:

```
for y = 0 to 119 {
        for x = 0 to 159 {
                turn on pixel (x, y) with colour ( x mod 8)
    }
}
```

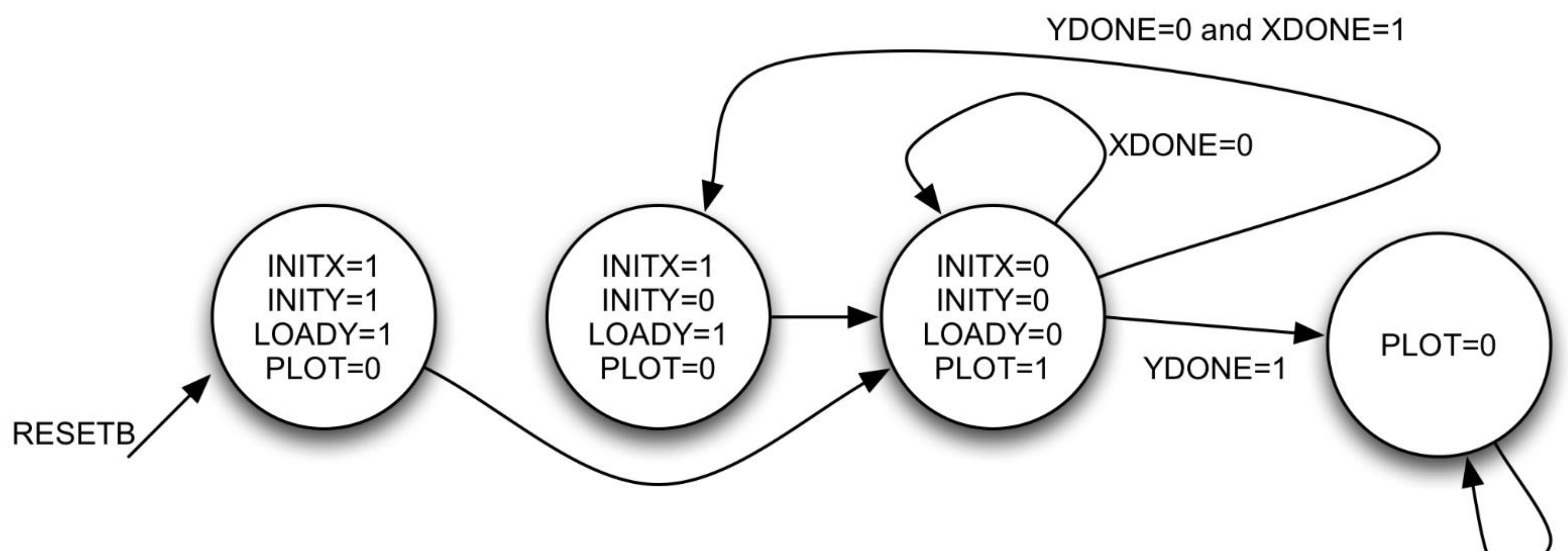
Naïve way to do it:



Better way:



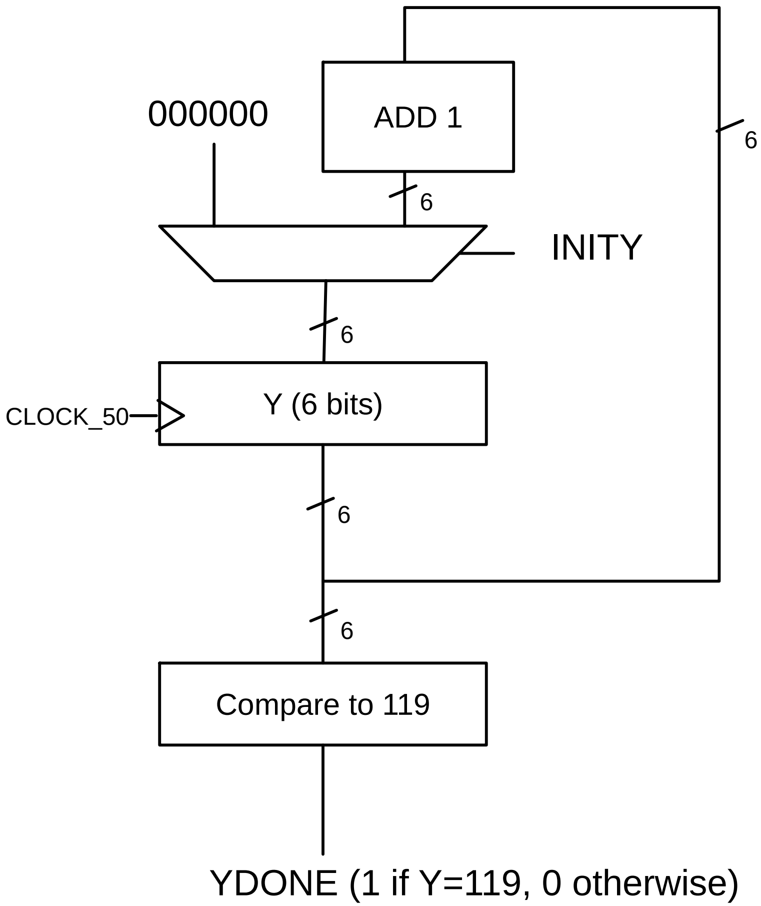
000000
ADD 1
6
INITY
LOADY
CLOCK_50
Y (6 bits)
To Y input of VGA
Compare to 119
YDONE (1 if Y=119, 0 otherwise)
0000000
ADD 1
7
INITX
CLOCK_50
X (7 bits)
To X input of VGA
Compare to 159
XDONE (1 if X=159, 0 otherwise)



YDONE=0 and XDONE=1
XDONE=0
INITX=1
INITY=1
LOADY=1
PLOT=0
INITX=1
INITY=0
LOADY=1
PLOT=0
INITX=0
INITY=0
LOADY=0
PLOT=1
PLOT=0
YDONE=1
RESETB

You know how to write that datapath and state machine in VHDL.

Short-cut: don't need to do everything structurally:



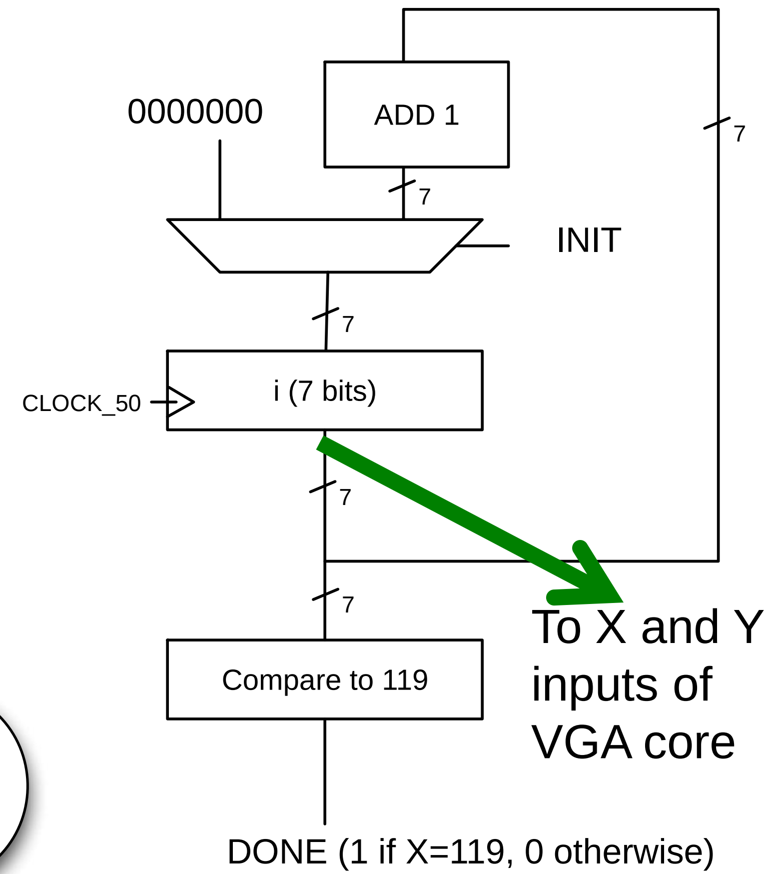

```
process(CLOCK_50)
variable Y : unsigned(5 downto 0);
begin
   if rising_edge(CLOCK_50) then
      if (INITY = '1') then
          Y := "000000";
      elsif (LOADY = '1') then
          Y := Y + 1;
      end if;
      YDONE <= '0';
      if (Y = 119) then
         YDONE <= '1';
      end if;
   end if;
end process;
```

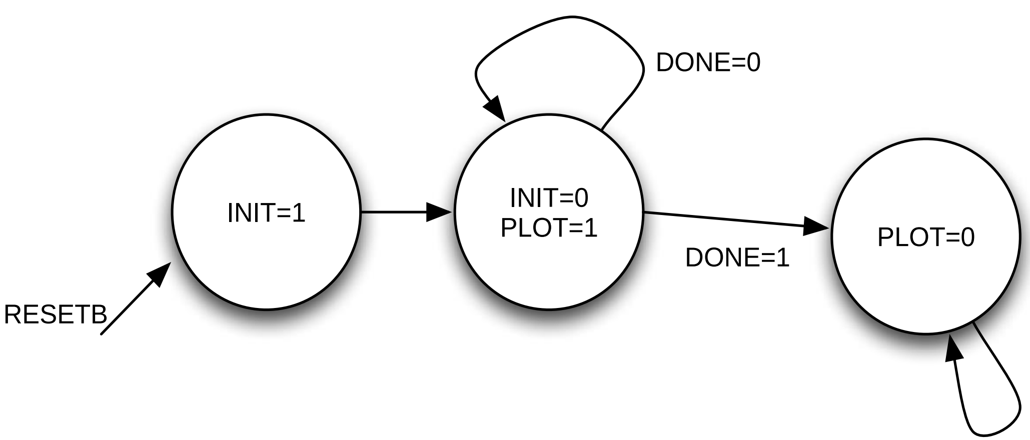
```
process(CLOCK_50)
variable Y : unsigned(5 downto 0);
variable X : unsigned(6 downto 0);
begin
   if rising_edge(CLOCK_50) then
      if (INITY = '1') then
          Y := "000000";
      elsif (LOADY = '1') then
          Y := Y + 1;
      end if;
      if (INITX = '1') then
          X := "0000000";
      else
          X := X + 1;
      end if;
      XDONE <= '0';
      YDONE <= '0';
      if (Y = 119) then
         YDONE <= '1';
      end if;
      if (X = 159) then
         XDONE <= '1';
      end if;
   end if;
end process;
```

Diagonal Lines

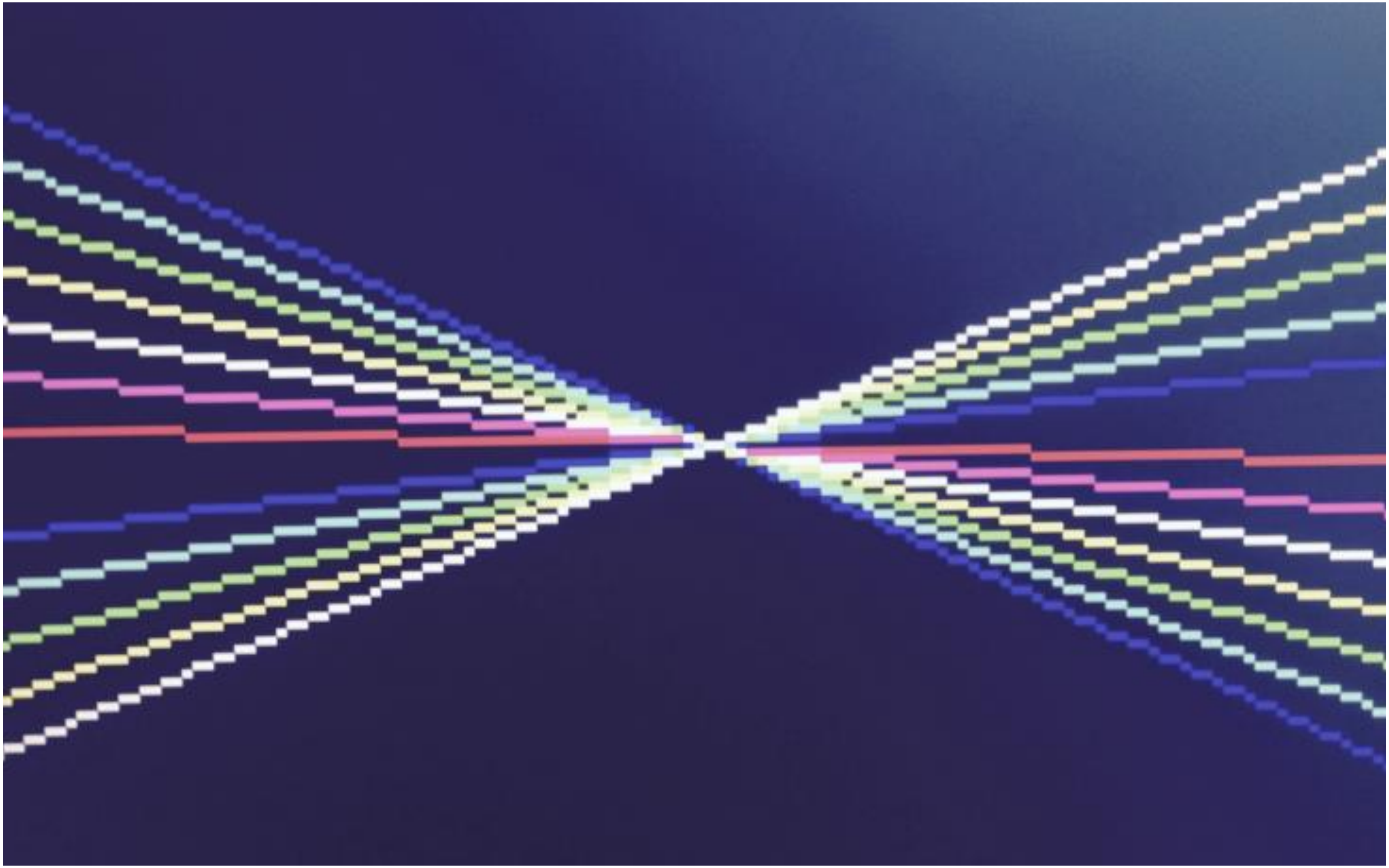
Not directly part of this lab, but say we want to draw a diagonal line from (0,0) to (119, 119):

```
for i = 0 to 119 {
    turn on pixel (i, i)
}
```





Task 3: Lines connecting arbitrary points



Task 3: Lines connecting arbitrary points

You can imagine it would be simple enough to work out the slope of the line (*m*) and, each iteration, increment y by *m* and x by 1.

Problem: this involves fractional arithmetic

- We will talk about fractional arithmetic later, however, for now, understand it would be more complex than it needs to be

Solution: Bresenham Line Algorithm:

- Uses only integer arithmetic

We are connecting (x0, y0) to (x1, y1):

```
dx := abs(x1-x0)
dy := abs(y1-y0)
if x0 < x1 then sx := 1 else sx := -1
if y0 < y1 then sy := 1 else sy := -1
err := dx-dy
loop
        setPixel(x0,y0)
        if x0 = x1 and y0 = y1 exit loop
        e2 := 2*err
        if e2 > -dy then
                err := err – dy
                x0 := x0 + sx
        end if
        if e2 < dx then
                err := err + dx
                y0 := y0 + sy
        end if
end loop
```

We are connecting (x0, y0) to (x1, y1):

```
dx := abs(x1-x0)
dy := abs(y1-y0)
if x0 < x1 then sx := 1 else sx := -1
if y0 < y1 then sy := 1 else sy := -1
err := dx-dy
loop
        setPixel(x0,y0)
        if x0 = x1 and y0 = y1 exit loop
        e2 := 2*err
        if e2 > -dy then
                err := err – dy
                x0 := x0 + sx
        end if
        if e2 < dx then
                err := err + dx
                y0 := y0 + sy
        end if
end loop
```

To start:

Straightforward implementation seems to need 5 registers, plus registers for x0, y0, x1, and y1, so draw those

We are connecting (x0, y0) to (x1, y1):

```
dx := abs(x1-x0)
dy := abs(y1-y0)
if x0 < x1 then sx := 1 else sx := -1
if y0 < y1 then sy := 1 else sy := -1
err := dx-dy
loop
        setPixel(x0,y0)
        if x0 = x1 and y0 = y1 exit loop
        e2 := 2*err
        if e2 > -dy then
                err := err – dy
                x0 := x0 + sx
        end if
        if e2 < dx then
                err := err + dx
                y0 := y0 + sy
        end if
end loop
```

To start:

Straightforward implementation seems to need 5 registers, plus registers for x0, y0, x1, and y1, so draw those

We are connecting (x0, y0) to (x1, y1):

```
dx := abs(x1-x0)
dy := abs(y1-y0)
if x0 < x1 then sx := 1 else sx := -1
if y0 < y1 then sy := 1 else sy := -1
err := dx-dy
loop
        setPixel(x0,y0)
        if x0 = x1 and y0 = y1 exit loop
        e2 := 2*err
        if e2 > -dy then
                err := err – dy
                x0 := x0 + sx
        end if
        if e2 < dx then
                err := err + dx
                y0 := y0 + sy
        end if
end loop
```

Next step:

Identify operations between those registers. These operations will be implemented by combinational blocks.

We are connecting (x0, y0) to (x1, y1):

```
dx := abs(x1-x0)                              Cycle 1
dy := abs(y1-y0)                              Cycle 2
if x0 < x1 then sx := 1 else sx := -1         Cycle 3
if y0 < y1 then sy := 1 else sy := -1         Cycle 4
err := dx-dy                                  Cycle 5
loop                                          etc…
      setPixel(x0,y0)
      if x0 = x1 and y0 = y1 exit loop
      e2 := 2*err
      if e2 > -dy then
            err := err – dy
            x0 := x0 + sx
      end if
      if e2 < dx then
            err := err + dx
            y0 := y0 + sy
      end if
end loop
```

Next step:

Figure out when each operation happens (draw state machine)

We are connecting (x0, y0) to (x1, y1):

```
dx := abs(x1-x0)
dy := abs(y1-y0)
if x0 < x1 then sx := 1 else sx := -1
if y0 < y1 then sy := 1 else sy := -1
err := dx-dy
loop
	setPixel(x0,y0)
	if x0 = x1 and y0 = y1 exit loop
	e2 := 2*err
	if e2 > -dy then
		err := err – dy
		x0 := x0 + sx
	end if
	if e2 < dx then
		err := err + dx
		y0 := y0 + sy
	end if
end loop
```

Cycle 1

Cycle 2

etc…

We can do better… (fewer states = faster)

Look for operations we can do in the same cycle

To draw 14 lines:

```
for i = 1 to 14
{
        draw line from ( 0, i*8) to
            ( 159, 120-(i*8) ) using
             colour gray(i mod 8)
}
```

Will all of these lines be drawn at the same time? Why/Why not?

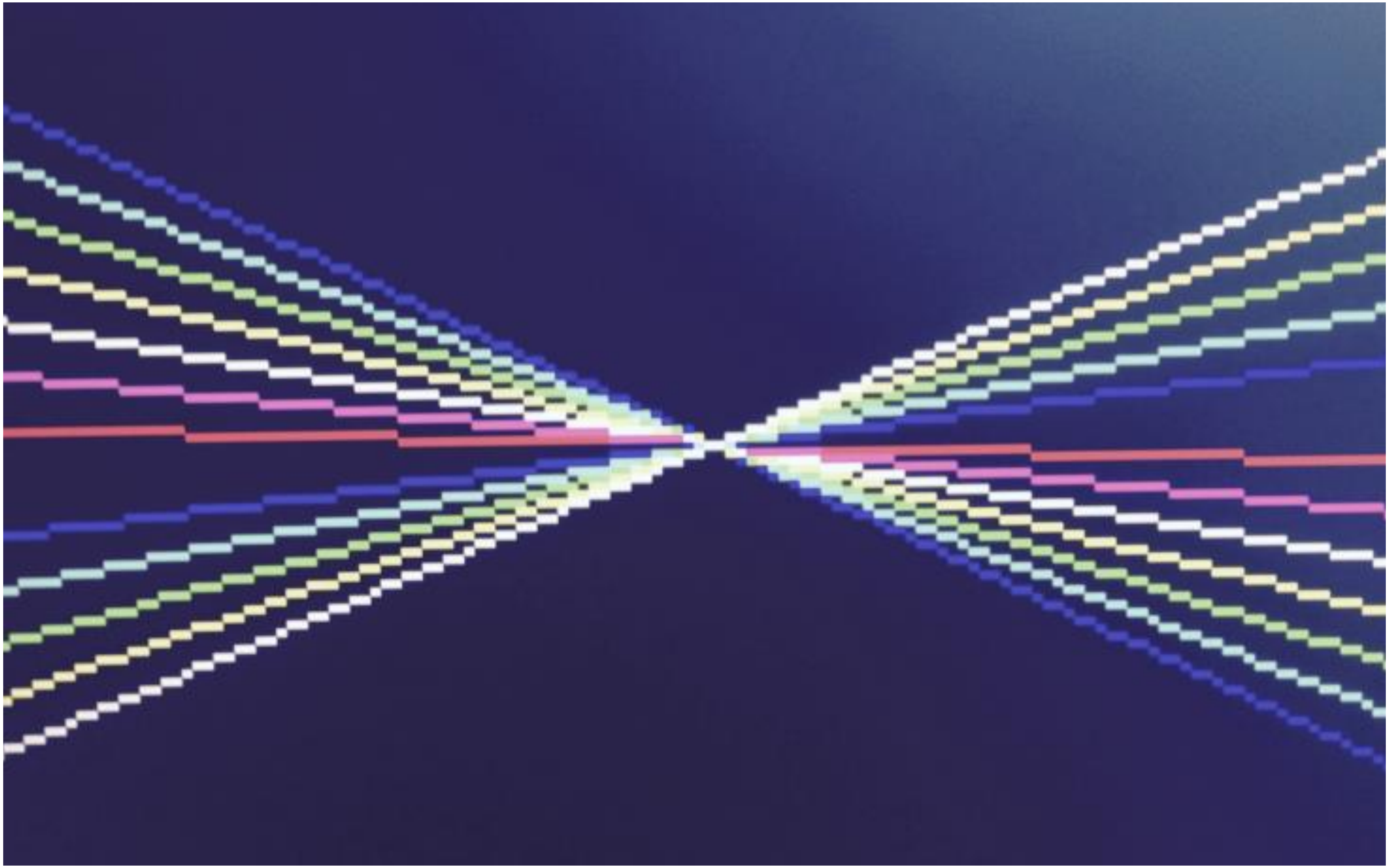
To draw 14 lines:

```
for i = 1 to 14
{
        draw line from ( 0, i*8) to
            ( 159, 120-(i*8) ) using
             colour gray(i mod 8)
        wait 1 second
        erase the line that was just drawn
}
```

How do you “wait 1 second”?

How do you “erase” a line?

Task 4: Lines connecting arbitrary points (Sequential)



Challenge Task: Triangles

A few points:

1. You will have a datapath and an FSM. You can put them together or build them separately depending on your comfort levels.

2. The inputs and outputs of each datapath and controller will depend on your design. Everyone's might be different and still work.

3. The x0 and y0 registers of the datapath should be connected directly to the x and y inputs of the VGA controller. The FSM will generate the PLOT signal.

4. Do not specify x and y coordinates outside of the screen dimensions. The VGA core does not clip as you might expect.

5. Remember you do not need to describe each datapath component as a separate file (as you might have done in Lab 2). I can imagine your entire datapath as one entity.

6. In all the arithmetic operations, you are using *signed* arithmetic now (just use type "signed" instead of "unsigned").