Coding Styles for the FPGA

Display Statements

- **Do not use Display statements!**

These statements help when debugging verilog code on a simulator however, the FPGA has no means to display them.

Initial Statements

- **Do not use initial statements!**

These are statements that allow you to initialize values in a simulator. However, they do not work on an FPGA.

So how do I initialize values?

- When the FPGA is programmed everything is initialized to zero by default

What if I need my value to be initialized to something other than zero?

This can be done in a couple of ways

- Set the value at the reset state

```
if (~reset) counter = 0;
        else if (~interrupt)
                counter = input ;
```

Reset

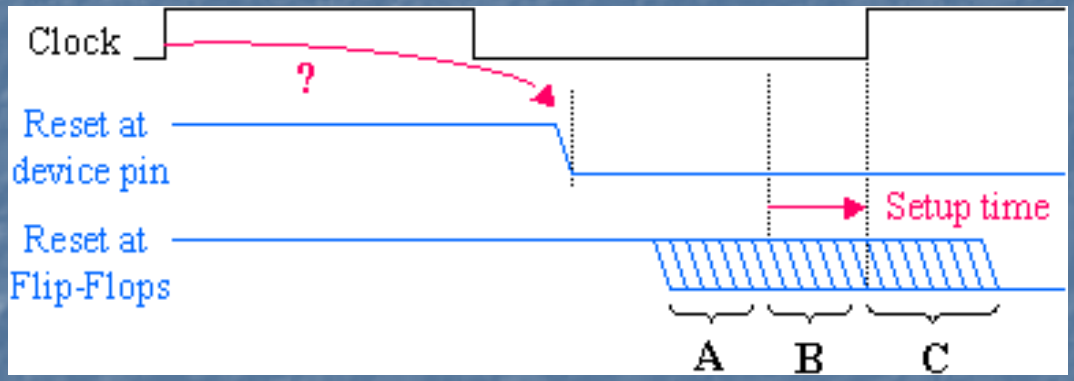
Good programming practice has always told you to “ALWAYS USE A RESET”. However, it is possible to get away without one just reprogram the FPGA each time you need to reset it since you can initialize it at program time. If you do use one you **must** be careful how you use it.

Do you use a Asynchronous or Synchronous Reset?

It is not recommended that you use a asynchronous reset unless it is absolutely necessary.

- It slows down your code
- If your design is large you can not guarantee that all parts of it will end up in the correct state afterwards

But what if the our clock was faster?



If the reset is released asynchronously to the clock (often the case), then there is no way to guarantee that all flip-flops will be released on the same clock edge, even if the distribution time is less than a clock period.

If-Else Vs. Case Statements

Both are possible how ever there are drawback to both.

For a FSM using if statements can get messy because every subsequent if must be inside of the previous else clause unless you “and” every condition with the state value. It is very easy to overlook two if statements in parallel

Examples

```
If ((Data_Ready == 1) && (High == 1) && (Low == 0))
begin
      ...//block 1
end
else if ((Data_Ready == 1) && (High == 0) && (Low == 1))
begin
      ...//block 2
end
else if ((Data_Ready == 0)
begin
      ...//block 3
end
else
begin
      ...//block 4
end
```

Case Statements

Case Statements are much more readable and normal case statements only have one drawback; only one condition clause can be executed per clock. However this can be solved by using a casex statement. A casex statement is exactly the same as a case statement except it can have “Don’t cares” in the conditions.

Examples

```
Case({Data_Ready,High,Low})
    3'b110 :
                    Begin
                            ...//block 1
                    End
    3'b101 :        Begin
                            ...//block 2
                    End
    3'b000 :        Begin
                            ...//block 3
                    End
    3'b001 :        Begin
                            ...//block 3
                    End
    Default         Begin
                            ...//block 4
                    End
```

Examples

```
Casex({Data_Ready,High,Low})
    3'b110        :        Begin
                           ...//block 1
                  End
    3'b101        :        Begin
                           ...//block 2
                  End
    3'b0xx        :        Begin
                           ...//block 3
                  End
    Default                Begin
                           ...//block 4
                  End
```

For Loops

It is strongly recommended that you avoid for loops. They do not occur sequentially so you can not use them for timing, counter, or state transitions.

Why?

The synthesis tool tries to figure out how to make every thing happen in one clock cycle. This is known as unrolling the loop.