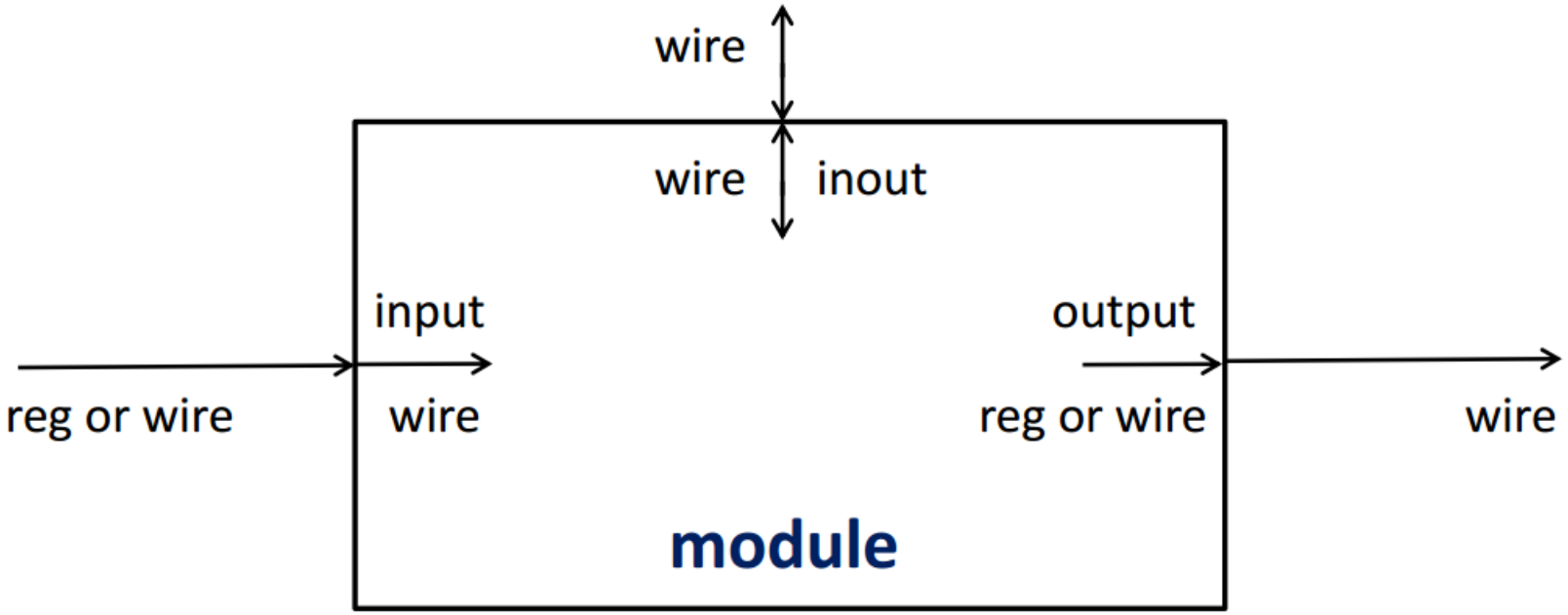
Verilog Syntax

Oct 26, 2016

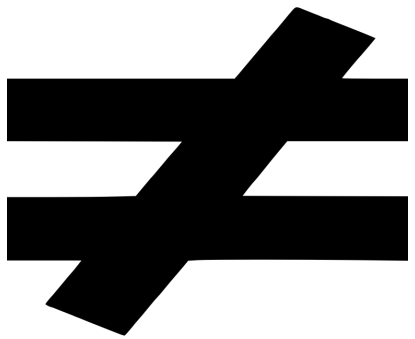
Port declaration

```
module postfix ( a,b);
Input[2:0]a;                                    //1995 syntax
Output[2:0]b;
```

```
module postfix (input[2:0] a, Output[2:0] b);      //2005 syntax
```

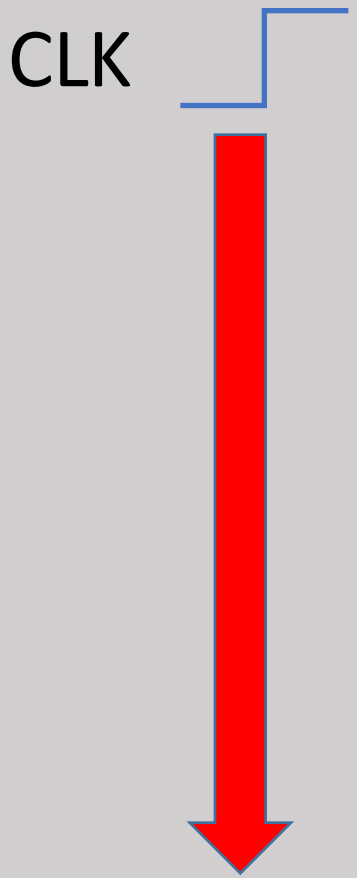


The above figure is the rule to design your types of inputs and outputs

if  always@

Sensitive List

```
always@(posedge CLK)
begin
    if(IN_VALID)
     begin
          if(OP_MODE)
          begin
                first<=first-1;
          end
          else
          begin
                stack[first]<= IN;
                first<=first+1;
          end
       end
end
```

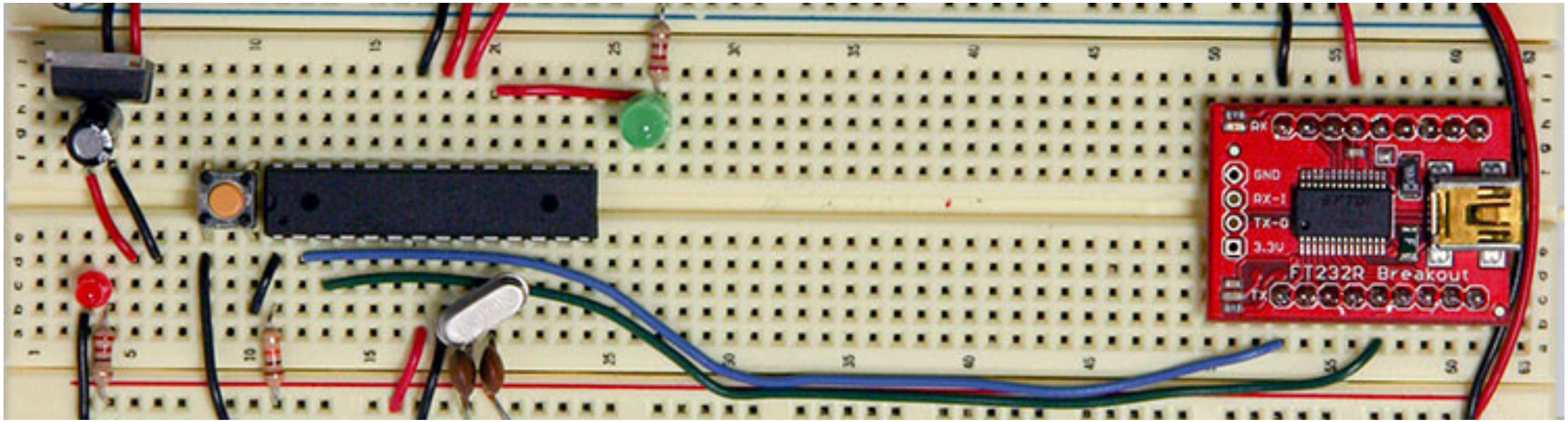


Multiple source

```
@always(posedge clk) begin
If(reset)
      A <= b;
end

@always(posedge clk) begin
If(flag)
      A <= c;
end
```

One always block for one signal

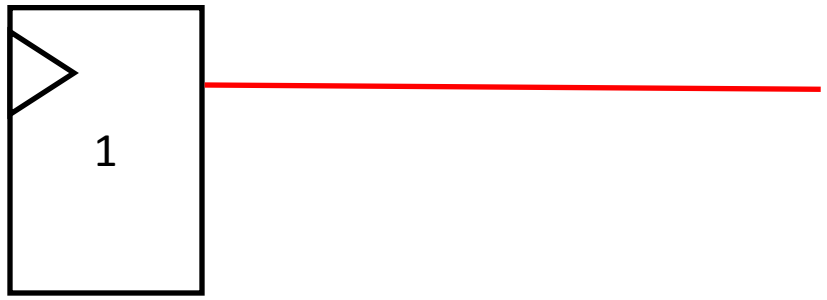
Wires



Are NOT controlled by
the clock

```
Wire a;
Always @(postedge clk) begin
If(flag)A=1;
end
```

Sol 1

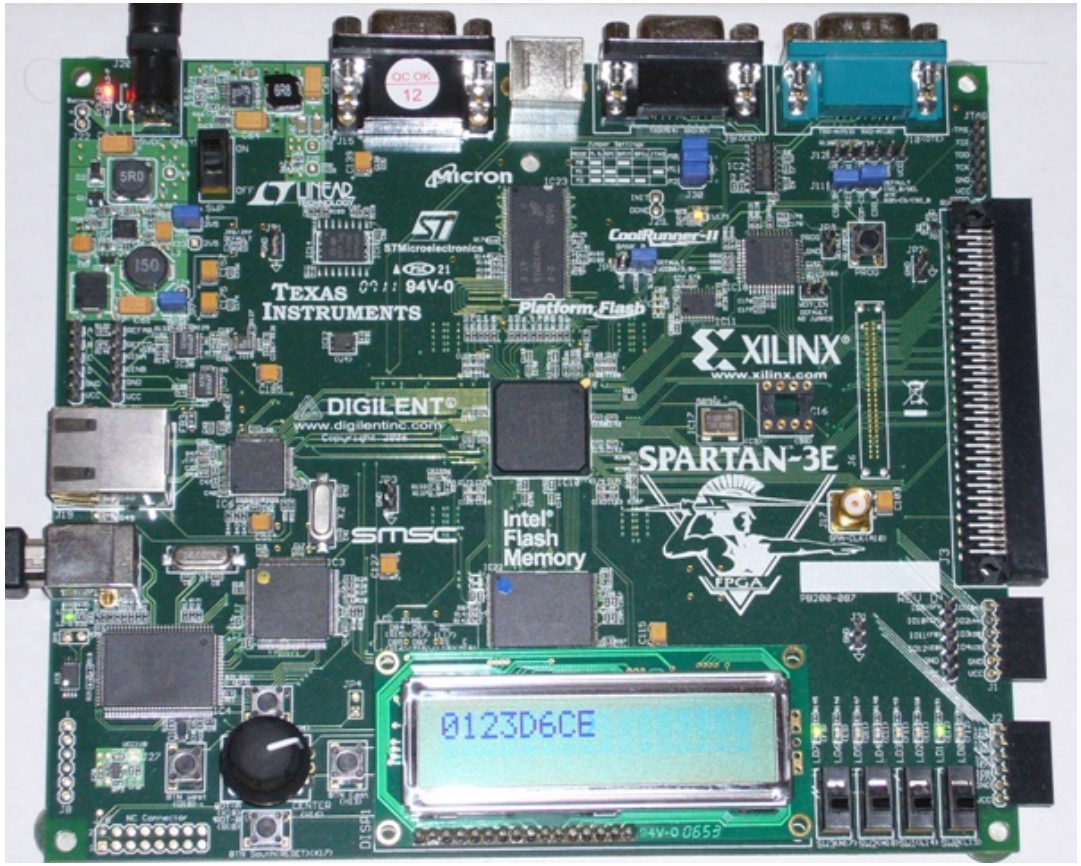



```
assign (wire) = (register);
```

Sol 2

```
assign (wire) =(flag)? 0: 1;
```

Find the component called "integer” on the board



Data are stored in registers, not integer

Suggestion

```
@always(posedge clk) begin
If(flag) begin
      A <= B;
      B = C;
end
end
```

Do NOT mix blocking & Non-blocking assignment

```
@always(posedge clk)
begin
if(flag)
begin
A<=1;
end
end
end
```

Coding-Style Indention

```
If
      (do something)
else
      (do something)

case(reg)
default:
      (do something)
endcase
```

```
if(RESET==0)begin
    OUT<=0;
    OUT_VALID=0;
end
```

When and How to use blocking and non-blocking assignments?

1. Use non-blocking assignments in ‘always’ block.
2. Use blocking assignments in combinational circuit.
3. Cannot mix non-blocking and blocking assignments in one ‘always’ block.

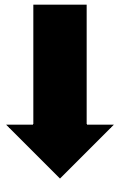
```
case(IN)
    4'b0001 :
        begin

        end
    4'b0010:
        begin

        end
    4'b0100:
        begin

        end
    //default:
    //  begin
    //
    //  end
endcase
```

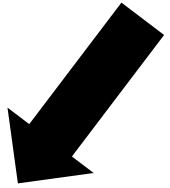
```
always@(posedge CLK or RESET)begin
    if(!RESET)begin
        OUT<=0;
    end
    else if(IN_VALID)begin
        /*
        *
        */
    end

end
```

1. When we're writing 'case' instruction, we have to add 'default'.
2. When we're writing 'if' instruction, we have to add 'else'.
3. You can put A<=A;
4. Otherwise, it will synthesize the code with Latches(uncontrollable)

```
always@(posedge CLK or RESET)begin
    if(RESET)begin
        A<=0;
    end
    else if(IN_VALID)begin
        A<=A+A;
    end
    else A<=A;
end
```

```
always@(posedge CLK or RESET)begin
    if(RESET)begin
        B<=0;
    end
    else if(IN_VALID)begin
        B<=B+B;
    end
    else B<=B;
end
```





```
always@(posedge CLK or RESET)begin
    if(RESET)begin
        OUT<=0;
    end
    else if(IN_VALID)begin
        OUT<=A+B;
    end
    else OUT<=OUT;
end
```

1. One module, one always
2. One always, one register