FPGA learning resources

Introduction to FPGAs

Getting Started with Xilinx

SCIPP
SANTA CRUZ INSTITUTE FOR PARTICLE PHYSICS

Digital Design

- Everything is represented in two discrete values: “0” and “1”
- We use low and high voltages to represent these values
- Use binary arithmetic and boolean math

Binary Numbers

- Converting decimal to binary:

  11 / 2 = 5 remainder 1

  5 / 2 = 2 remainder 1

  2 / 2 = 1 remainder 0

  1 / 2 = 0 remainder 1

- Read it from bottom to top: 1011

Binary Numbers

- Converting from binary to decimal
- Every digit represents a power of two
- Note that we start with 2^0

$$(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 11$$

What is an FPGA?

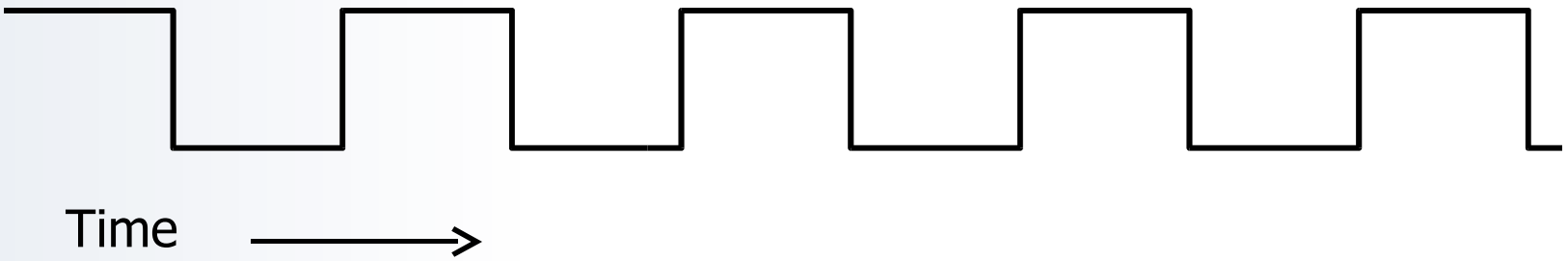
- Field Programmable Gate Array
- Hardware that can be customized
- Can be programmed to do just about anything you want them to do.

Verilog

- HDL (Hardware Descriptive Language)
- Allows one to create designs using a text language
- One “lays out” the design

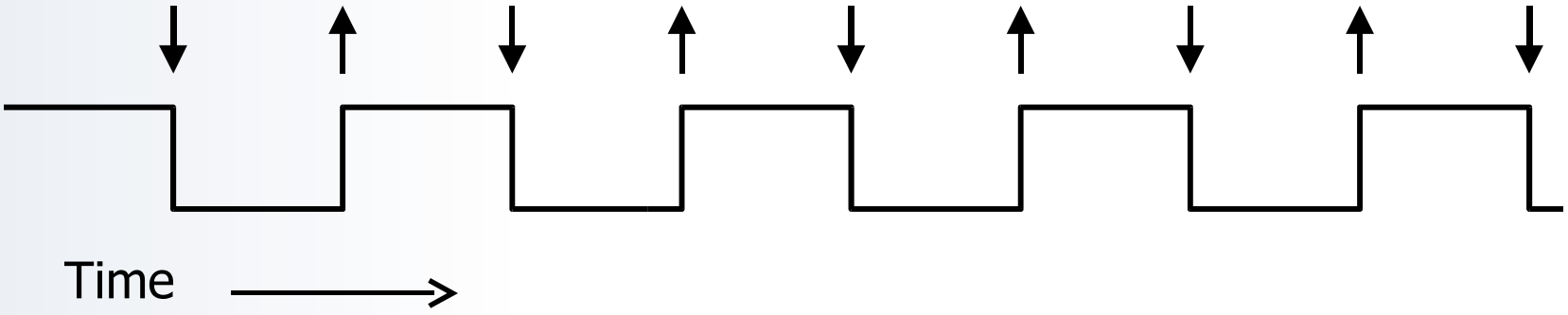
Clocks

- Clocks define the time element of a design
- Clocks alternate between high and low in a square wave



Clock Edges

- We are only concerned with clock edges usually
- Edges are simply where a transition takes place

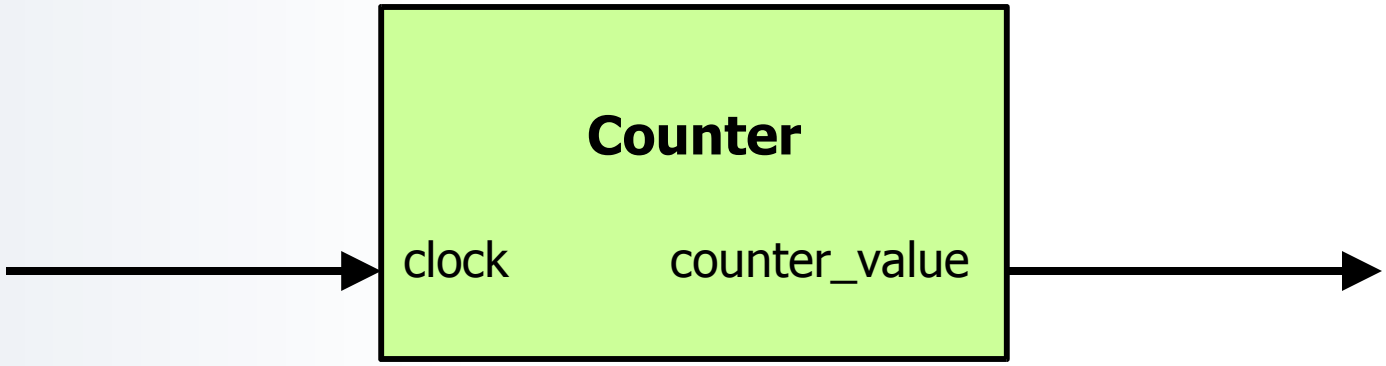


A Simple Counter

- We will create a design that counts up on every clock edge
- Will store the value in a piece of hardware called a “register” (think memory)
- We will use an 8-bit number, representing $2^8 = 256$ different values

A Simple Counter

- "Black box" – should have inputs and outputs
- Focus on "what" not "how"



The Code

```
module counter( clock, counter_value );

        input clock;
        output [3:0] counter_value;
        reg [3:0] counter;
        assign counter_value = counter;

        always @(posedge clock)
        begin
                counter = counter + 1;
        end

endmodule
```

The Code

- "module" declaration specified what the inputs and outputs are

```
module counter( clock, counter_value );

        input clock;
        output [3:0] counter_value;
        …

endmodule
```

The Code

- "reg" keyword makes a register. We use the bracket notation to specify how big the number is
- Since we're using 4 bits, we can represent 2^4 numbers, or 16 total numbers
  - 0000, 0001, 0010, 0011, 0100, etc.

```
reg [3:0] counter;
```

The Code

- "always" specifies that we want to do something whenever a change occurs
- In our case, anytime the clock goes from low to high, increment the counter

```
always @(posedge clock)
begin
    counter = counter + 1;
end
```

Visualizing The Code

- Concurrency – Everything happens at once
- Verilog code turns into real hardware

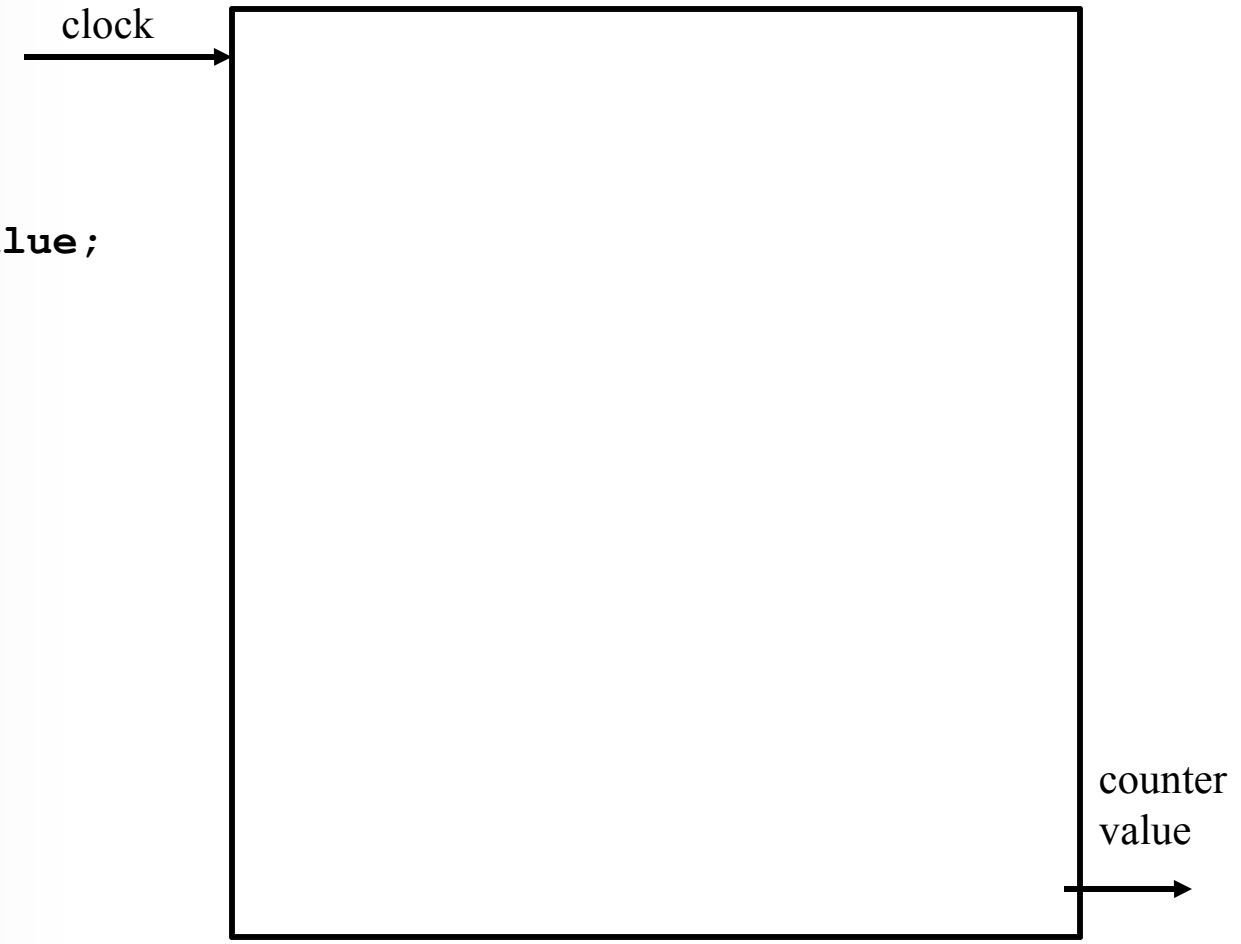
Visualizing The Code

```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;




endmodule
```



Visualizing The Code

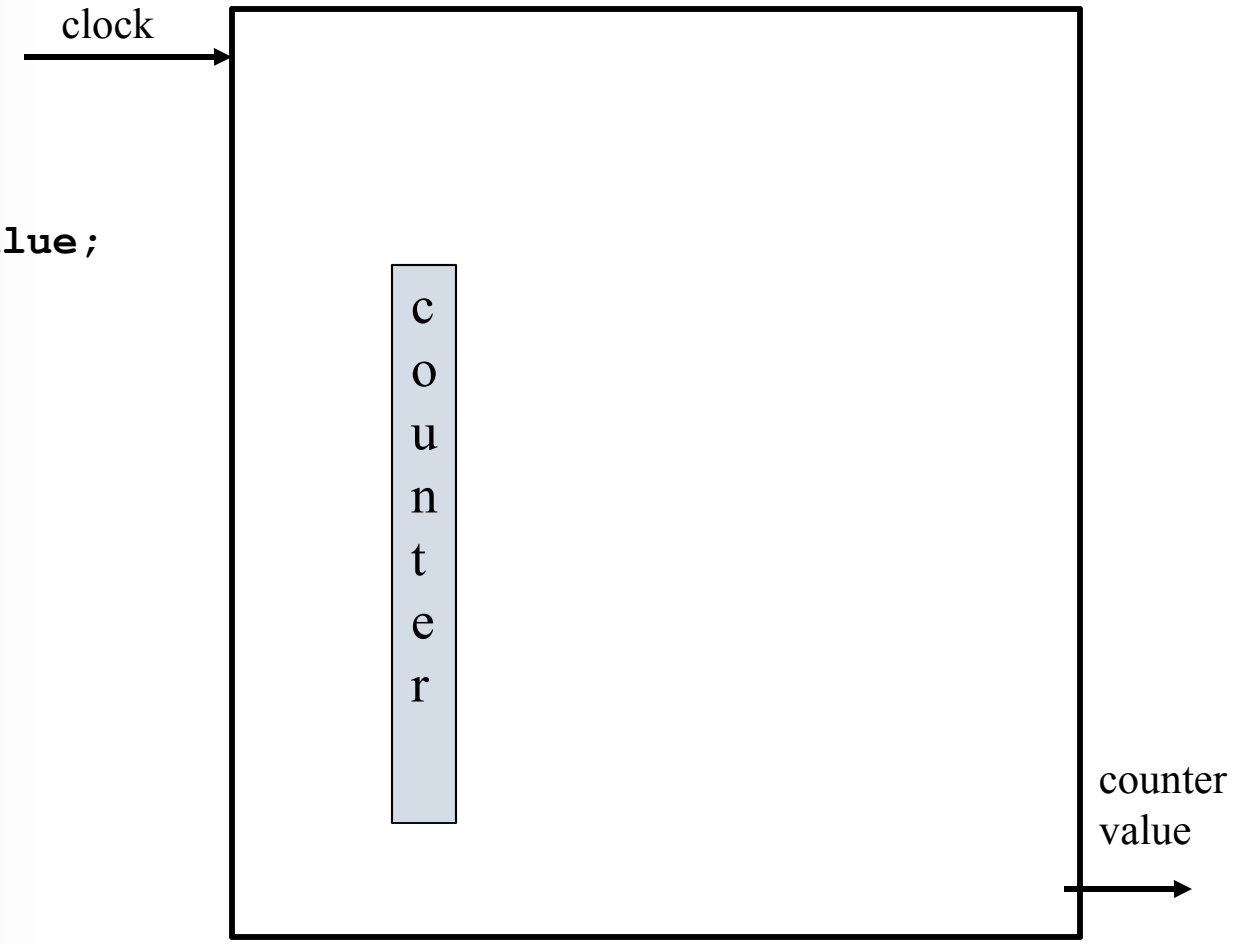
```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;

 reg [3:0] counter;




endmodule
```



Visualizing The Code

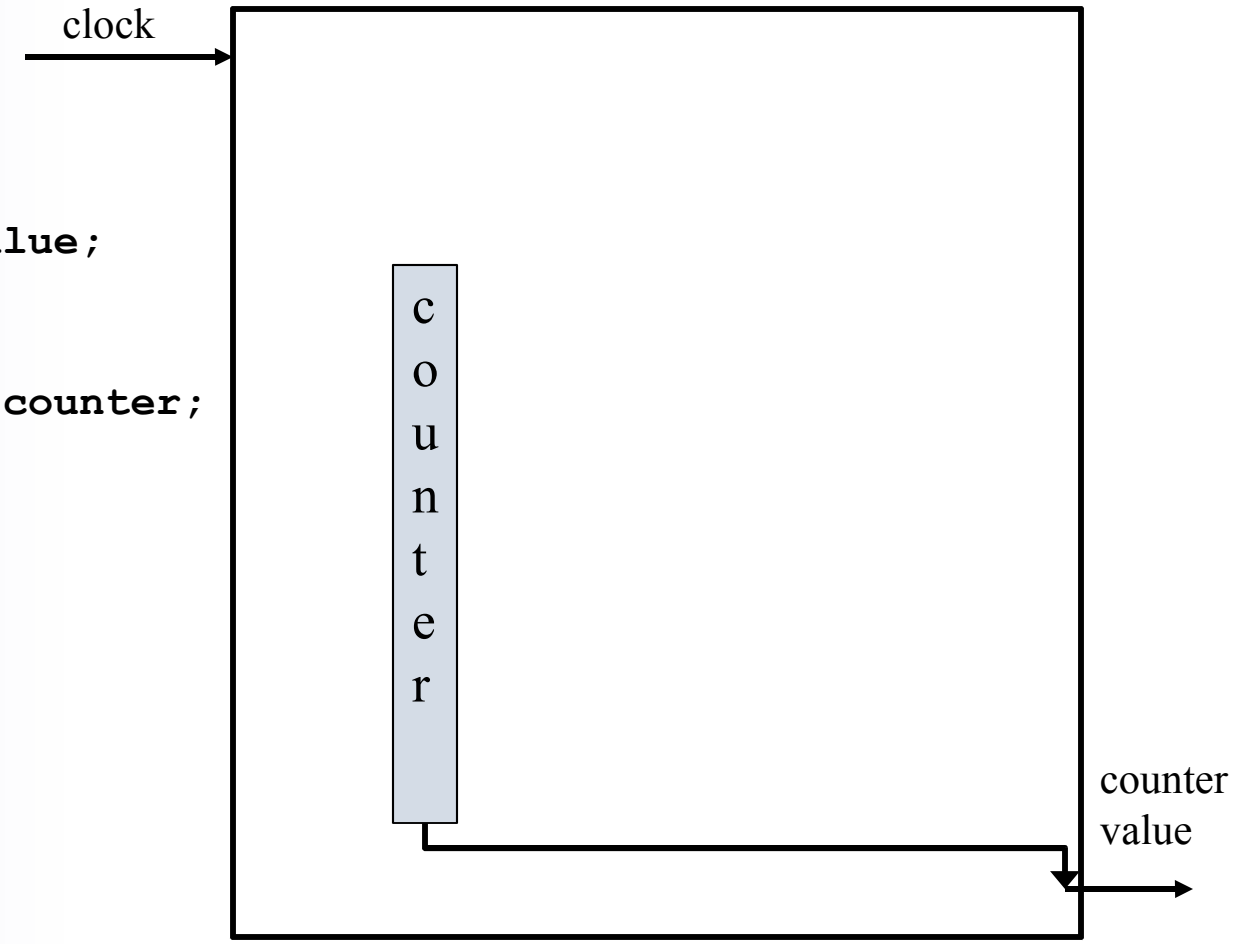
```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;

 reg [3:0] counter;
 assign counter_value = counter;




endmodule
```



Visualizing The Code

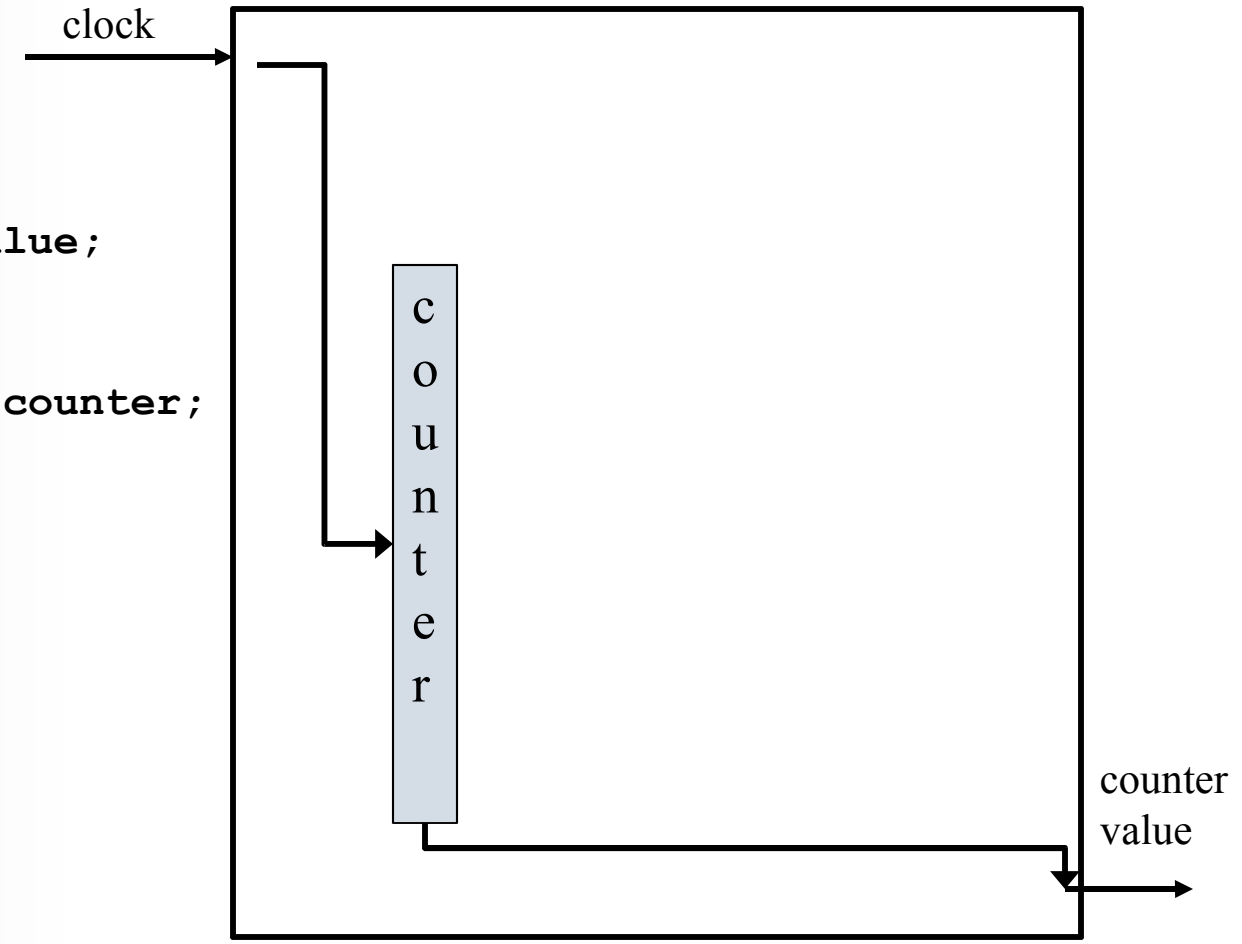
```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;

 reg [3:0] counter;
 assign counter_value = counter;

 always @(posedge clock)
 begin
  counter
 end

endmodule
```



Visualizing The Code

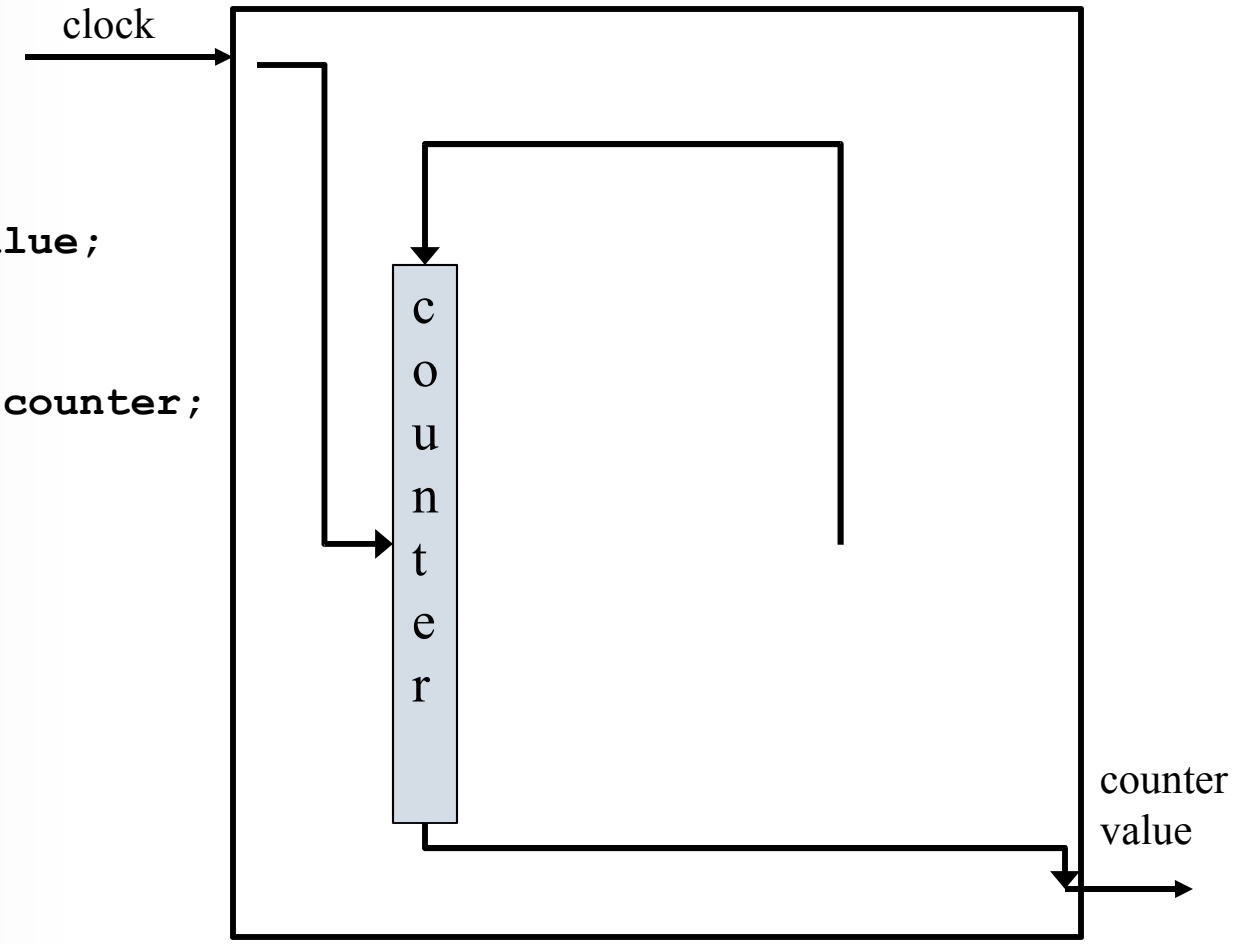
```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;

 reg [3:0] counter;
 assign counter_value = counter;

 always @(posedge clock)
 begin
  counter =
 end

endmodule
```



Visualizing The Code

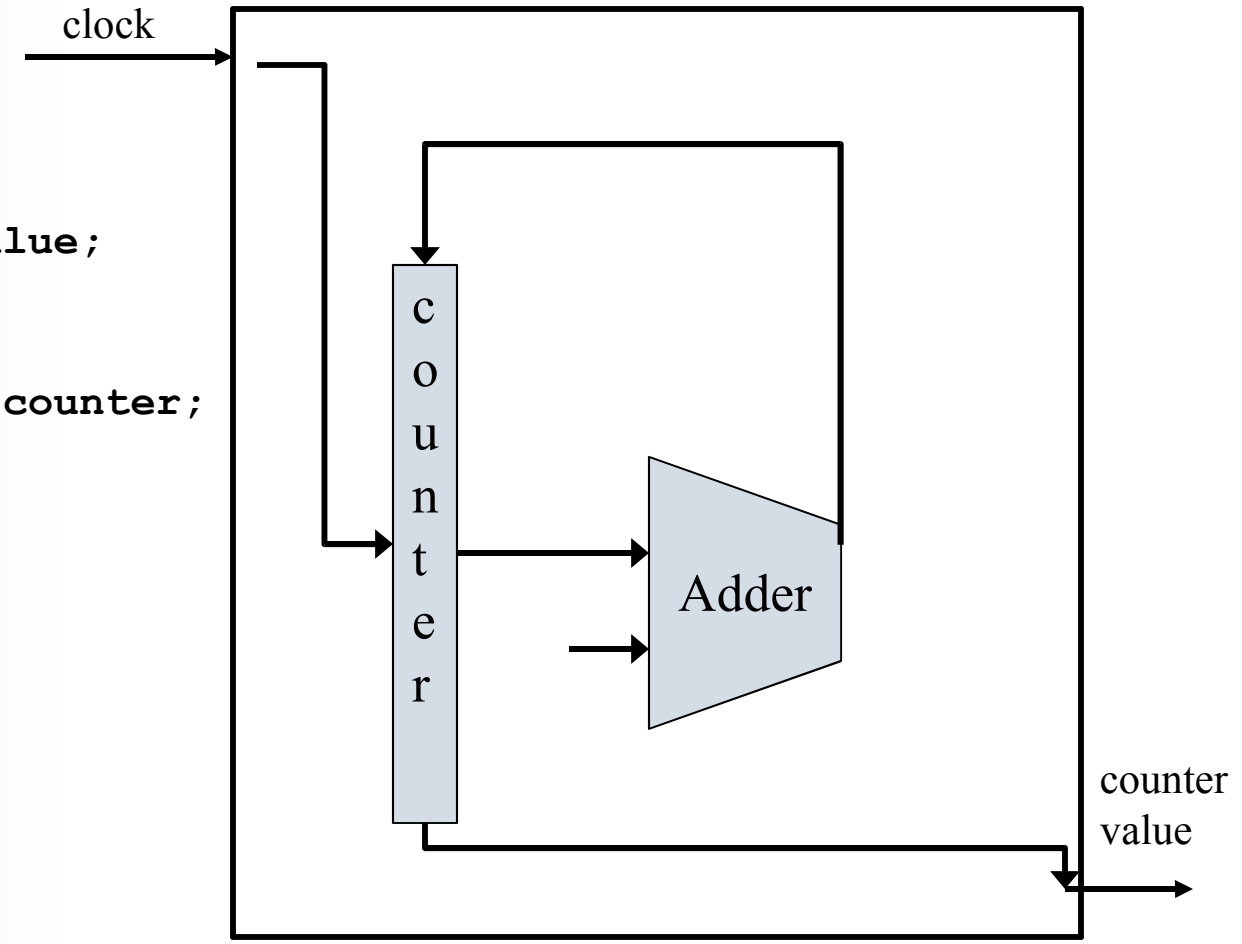
```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;

 reg [3:0] counter;
 assign counter_value = counter;

 always @(posedge clock)
 begin
  counter = counter +
 end

endmodule
```



Visualizing The Code

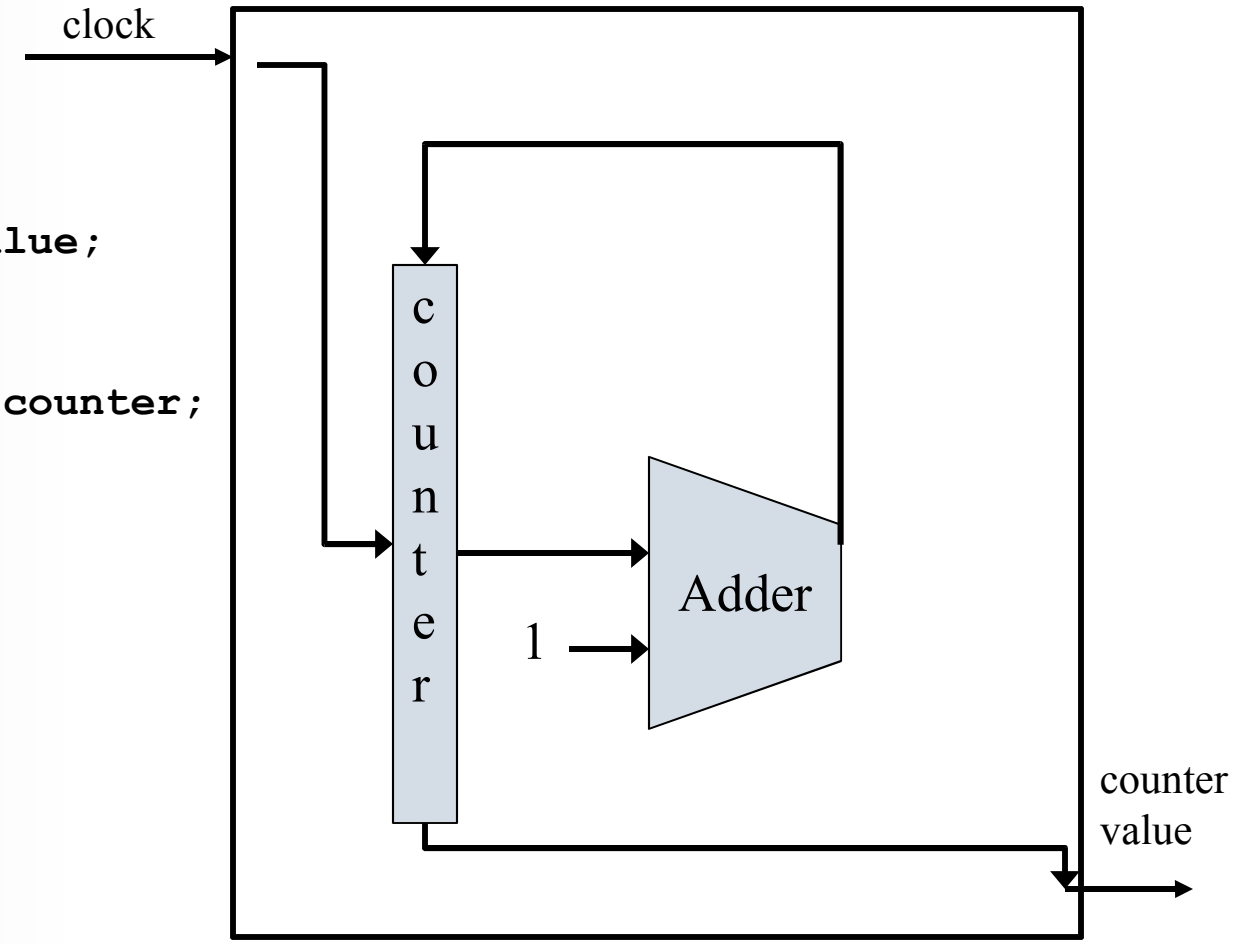
```
module counter( clock,
 counter_value );

 input clock;
 output [3:0] counter_value;

 reg [3:0] counter;
 assign counter_value = counter;

 always @(posedge clock)
 begin
  counter = counter + 1;
 end

endmodule
```



Tools Of The Trade

- Xilinx Software – ISE (Integrated Software Environment)
- Allows us to create designs
- Does all the grunt work of turning our code into physical hardware
- We program the chip through ISE

Hands On

- Next session will be a walkthrough of the hardware
- We'll get to program an actual counter