

Take the Leap: From MCUs to FPGAs

Spend a year with me as I use my MCU and software background as a base from which to explore the strange land of programmable logic

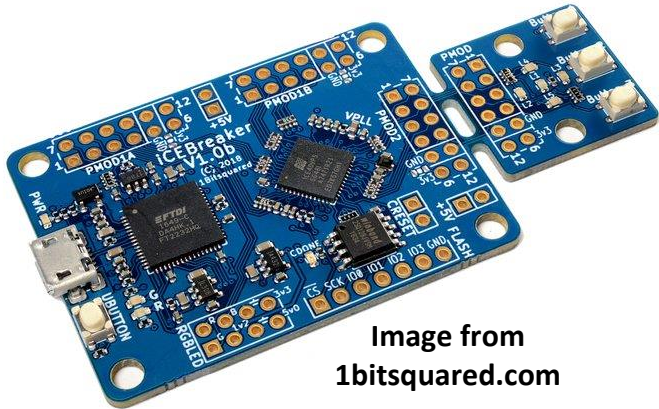


Image from
1bitsquared.com

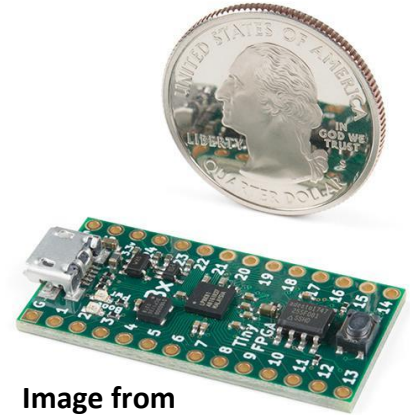


Image from
tinyFPGA.com

Field Programmable Gate Array

Don't
~~Fear~~

Programmable

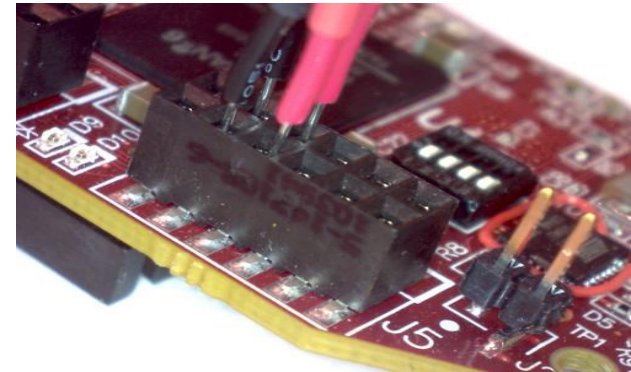
Gate

Arrays



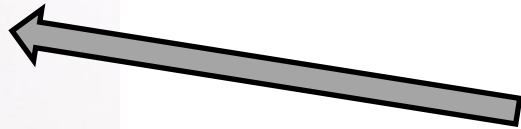
Key Take Aways

- Why you might want to use an FPGA
- Exposure to some of the most common traps that trip up FPGA beginners



Where I came from

- First soldering iron burn on finger: 1975
- Digital electronics 30 years ago
- Software off and on for a long time
 - The '80s were a weird: I wrote software for spare change
 - MCUs for a decade +
 - Picked up robots in the 21st century because they're modern

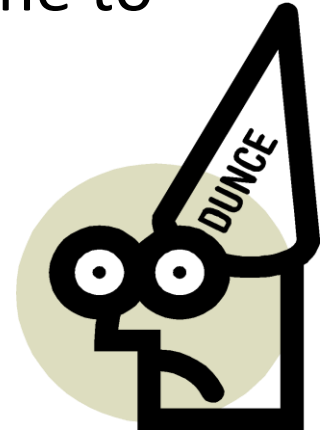
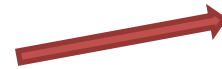


Me

My Role in All of This

- Ever asked a question where the answer was a smug “RTFM”?
- I spent a year asking those dumb questions so, in theory, you won’t have to. You’re welcome to anyway, but it won’t be mandatory.

Bad clip-art that came with this presentation software representing me



Take the Leap: from MCUs to FPGAs.

But watch out for alligators

Duane Benson

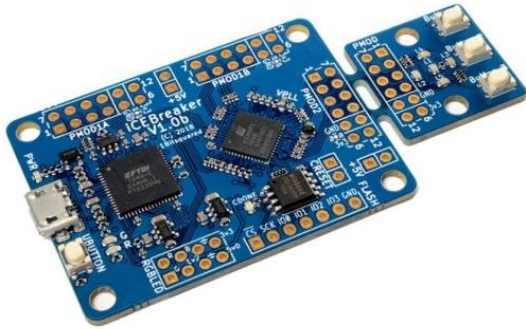


Image from
1bitsquared.com

```

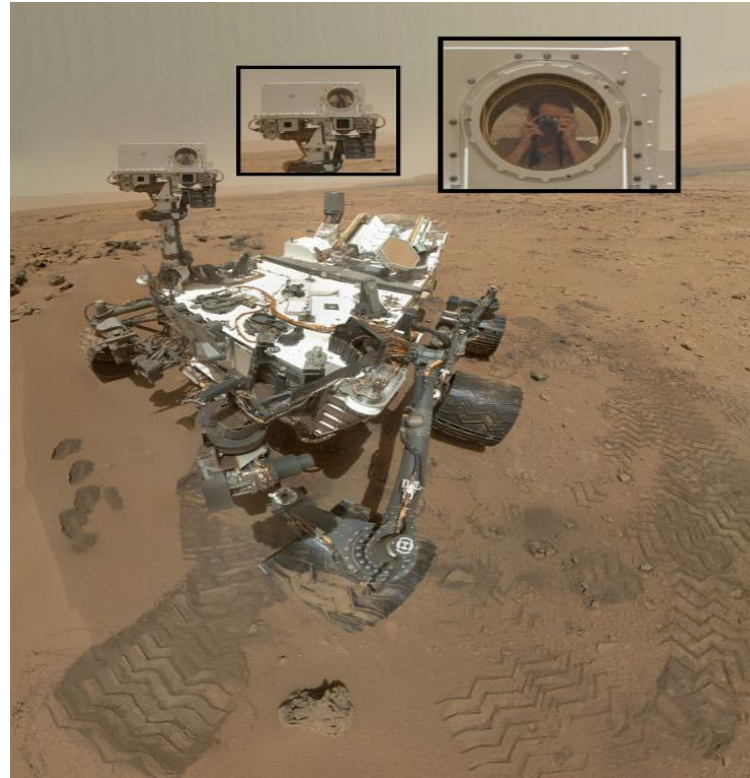
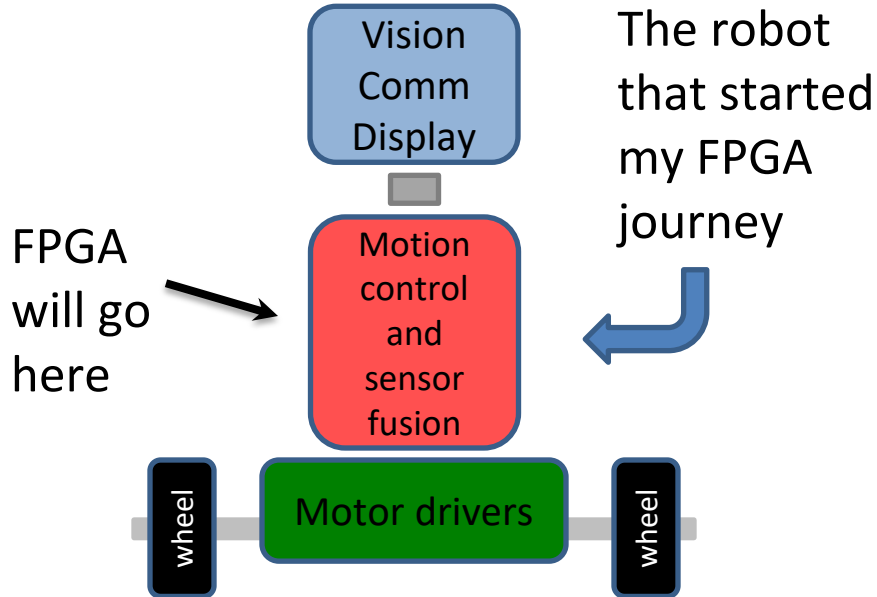
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  Library UNISIM;
5  use UNISIM.vcomponents.all;
6
7  entity ledflash is
8  Port ( CLK_66MHZ   : in  STD_LOGIC;
9         USER_RESET : in  STD_LOGIC;
10        LED         : out STD_LOGIC_VECTOR (3 downto 0));
11 end ledflash;
12
13 architecture Behavioral of ledflash is
14 signal clk           : std_logic;
15 signal clk_enable    : std_logic;
16 signal led_count     : std_logic_vector(26 downto 0);
17 begin
18     clk_enable <= not USER_RESET;
19     BUFGCE_inst : BUFGCE
20     port map (
21         O => clk,
22         CE => clk_enable,
23         I => CLK_66MHZ
24     );
25
26     process (clk)
27     begin
28         if clk='1' and clk'event then
29             led_count <= led_count + 1;
30         end if;
31     end process;
32     LED <= led_count(26 downto 23);
33 end Behavioral;
    
```



Image from
tinyFPGA.com

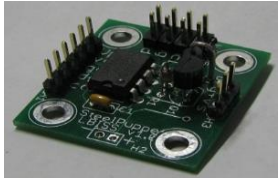
Why FPGAs?

- Extreme customization
- Parallel work

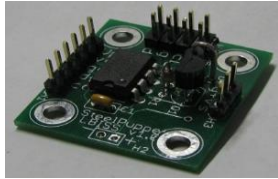


Why FPGAs?

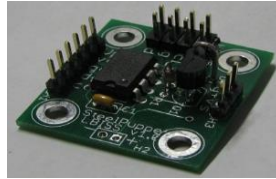
- And this: Parallel work



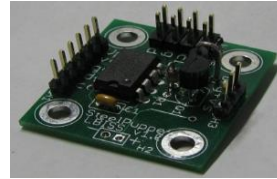
I2C
↓



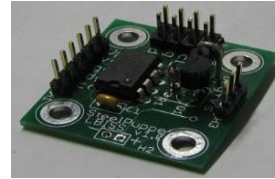
I2C
↓



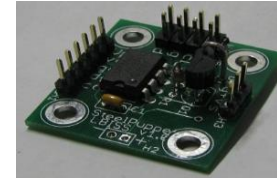
I2C
↓



I2C
↓



I2C
↓



I2C
↓

All loaded into the FPGA and processed at the same time.

FPGAs:

- Not as big a jump as I had thought
- Not as expensive as I had thought
- There are quite a number of inexpensive and (relatively) easy to use tools available



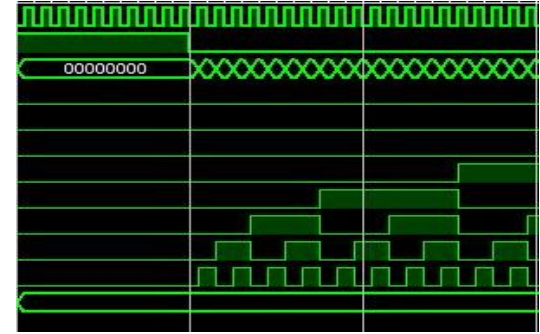
But – There are a few dangerous traps – especially if you come with an MCU mindset

What's the Same vs. Not the Same

- Use an IDE
- Multiple files in the development set
- Multiple language options
- You code, then implement
- Often familiar syntax
- No instant start at power-up
- The FPGA isn't "running" your code*
- You are designing a piece of hardware
- You must wire up the chip pins to the innards
- Most pins are not-mapped to a function

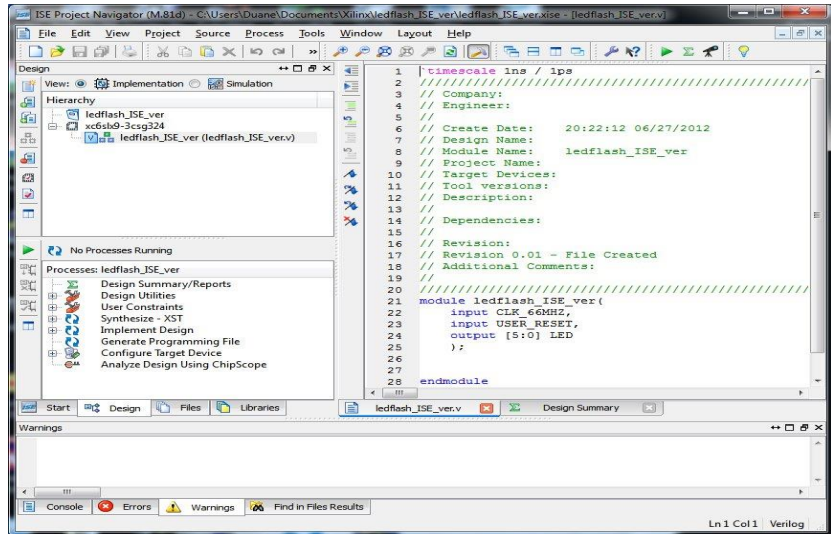
Time to Dig In

- What do you need?
- Digital logic knowledge
- Programming experience will both help and hurt
- Familiarity with IDEs
- MCU programming experience may be more relevant than OS or applications programming



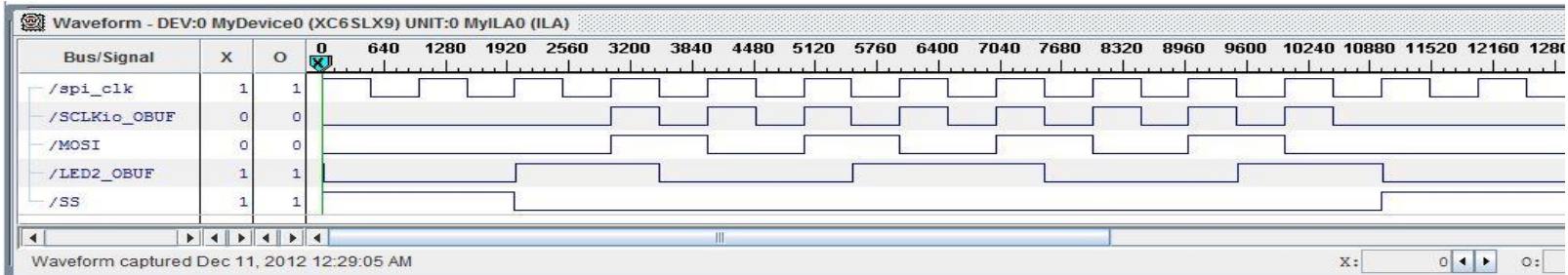
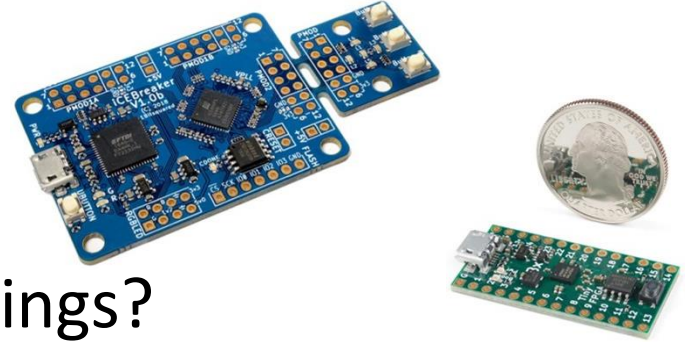
Development environment

- Lattice, Xilinx, Altera/Intel all have their own IDEs
- That's nice of them
- Open source toolchains are around now too



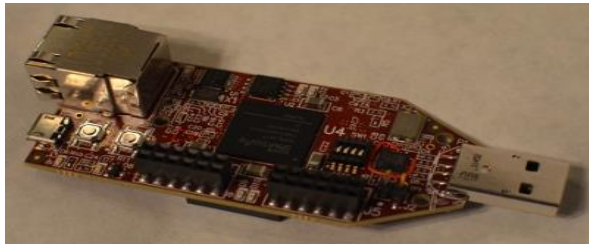
Pick Your Board

- Now What?
- How do you “program” these things?
- Some will say: “Technically, it’s not a program.” But what is it? How do you make an FPGA do something?



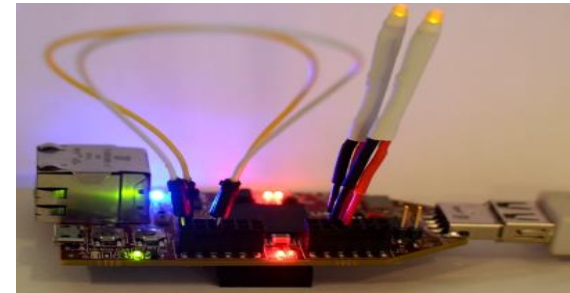
Order of Work

1. Define “constraints” in UCF/LPF
2. Code in your HDL
3. Simulate
4. Synthesize
5. Crunch to bit file
6. Load bit file to device



Empty hardware

Configured and
working thing



Three primary file Types



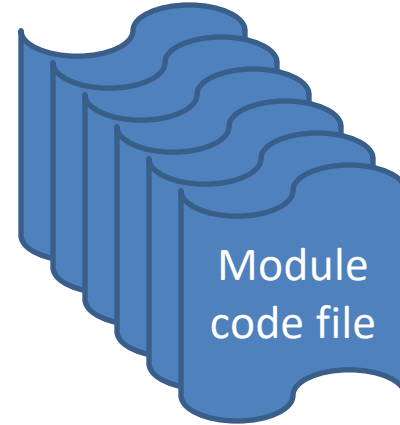
UCF/LPF file

Defines pins,
port names and
locations



Verilog code
file

Core of your HDL
code



Module
code file

Library/Code
module snippets
(includes)

UCF/LPF: Mapping/defining pins

```
1 NET "CLK_66MHZ"      LOC = "K15" | IOSTANDARD = LVCMOS33;
2
3 NET LED<0>           LOC = P4 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
4 NET LED<1>           LOC = L6 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
5 NET LED<2>           LOC = F5 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
6 NET LED<3>           LOC = C2 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
7
8 NET USER_RESET      LOC = V4 | IOSTANDARD = LVCMOS33 | PULLDOWN; # "USER_RESET"
9
10 CONFIG VCCAUX = "3.3" ;
```

User Constraints File / logical Preferences File, or equivalent

Anatomy Of The UCF/LPF

```
1 NET "CLK_66MHZ"          LOC = "K15" | IOSTANDARD = LVCMOS33;
2
3 NET LED<0>                LOC = P4 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
4 NET LED<1>                LOC = L6 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
5 NET LED<2>                LOC = F5 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
6 NET LED<3>                LOC = C2 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
7
8 NET USER_RESET           LOC = V4 | IOSTANDARD = LVCMOS33 | PULLDOWN;      # "USER_RESET"
9
10 CONFIG VCCAUX = "3.3" ;
```

My labels

Physical chip pins

User Constraints File / logical Preferences File, or equivalent

UCF/LPF: Mapping/defining pins

```
1 NET "CLK_66MHZ"      LOC = "K15" | IOSTANDARD = LVCMOS33;
2
3 NET LED<0>           LOC = P4 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
4 NET LED<1>           LOC = L6 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
5 NET LED<2>           LOC = F5 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
6 NET LED<3>           LOC = C2 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
7
8 NET USER_RESET      LOC = V4 | IOSTANDARD = LVCMOS33 | PULLDOWN; # "USER_RESET"
9
10 CONFIG CCAUX = "3.3" ;
```

My label for use in HDL

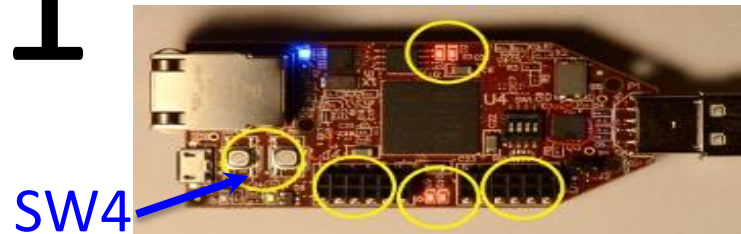
Physical chip pin

User Constraints File / logical Preferences File, or equivalent

First – Define Connections

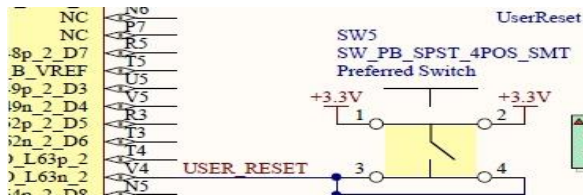
```
NET USER_RESET LOC = V4 | IOSTANDARD = LVCMOS33 | PULLDOWN; # "USER_RESET"
```

1 Switch on PCB: SW4



SW4

2 SW4 wired to chip pin V4 with PCB trace



3 Write UCF to map chip pins to labels:
 NET USER_RESET LOC = V4

*“NET” and “LOC” are reserved words,
 “V4” is the chip pin,
 “USER_RESET” is my label*

4 HDL code will later reference the label:
 Input USER_RESET

Anatomy Of a Verilog File

```
1  `timescale 1 ns / 1 ps
2
3  module ledflash
4  (
5  input  wire      CLK_66MHZ,
6  input  wire      USER_RESET,
7  output wire [5:0] LED
8  );
9
10 wire      clk;
11 wire      clk_enable;
12
13 assign clk_enable = ~USER_RESET;
14
15 BUFGCE BG (.O(clk), .CE(clk_enable), .I(CLK_66MHZ));
16 reg [26:0] led_count;
17
18 always @(posedge clk)
19     led_count <= led_count + 1;
20
21 assign LED[5:0] = led_count[26:21];
22 endmodule
```

Module / Ports
(connections to the outside world)

Declarations
(Internal use only)

Clock triggered circuitry

Wire / Register / Assign

```
1  `timescale 1 ns / 1 ps
2
3  module ledflash
4  (
5    input wire      CLK_66MHZ,
6    input wire      USER_RESET,
7    output wire [5:0] LED
8  );
9
10 wire      clk;
11 wire      clk_enable;
12
13 assign clk_enable = ~USER_RESET;
14
15 BUFGCE BG (.O(clk), .CE(clk_enable), .I(CLK_66MHZ));
16 reg [26:0] led_count;
17
18 always @(posedge clk)
19     led_count <= led_count + 1;
20
21 assign LED[5:0] = led_count[26:21];
22 endmodule
```

Module / Ports

(connections to the outside world)

Declarations

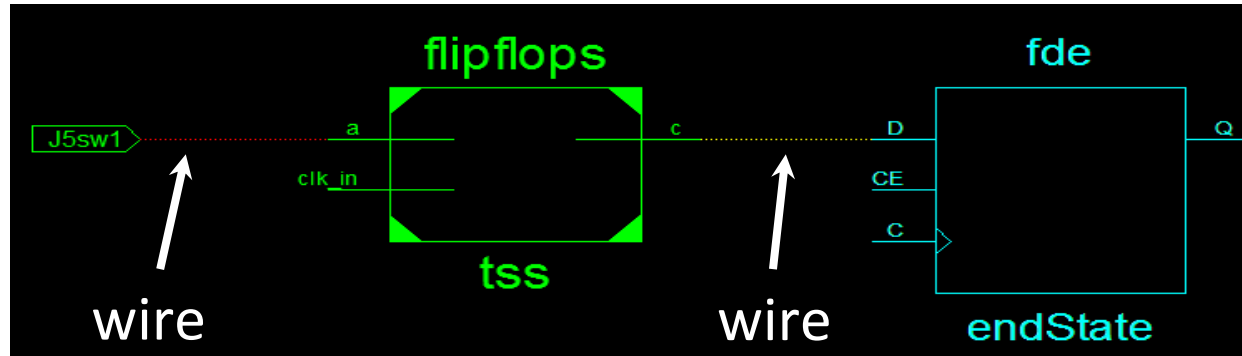
(Internal use only)

Clock triggered circuitry

Wire / Register / Assign

Blinding flash of the obvious here: Wires just go between two things. Obvious, yes. But it needs to be stated in the “new to FPGA” world.

It’s not a register like a hardware register in your MCU. It stores value or state logically to combine with another register value – more like a RAM location or variable, although some people don’t like that comparison.



“Assign” creates a permanent connection

Wire / Register / Assign

A few important rules

`assign` `awire` = `aregister_or_awire`

Only a “wire” can be on the left of the = sign in an `assign` statement.

An `assign` cannot be used within an `always` block

“`assign`” means to wire something up at configuration time.

`always @(posedge clk) begin`

`areg = areg` `or` `areg <= areg`

`areg = awire` `or` `areg <= awire`

A wire can't be on the left

when inside of an `always` block

Both wires and registers can be on the right side anywhere



It's not an array, it's a ribbon cable – sort of

In the UCF: →

```
NET LED<0> LOC = P4 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;  
NET LED<1> LOC = L6 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;  
NET LED<2> LOC = F5 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;  
NET LED<3> LOC = C2 | IOSTANDARD = LVCMOS18 | DRIVE = 8 | SLEW = SLOW ;
```

```
output wire [5:0] LED  
...  
reg [15:0] ledCount = 16'hFFFF;
```

```
always @(posedge clk) begin  
    ledCount <= ledCount + 1;  
End
```

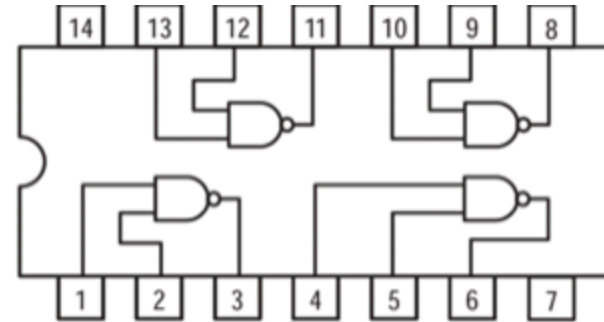
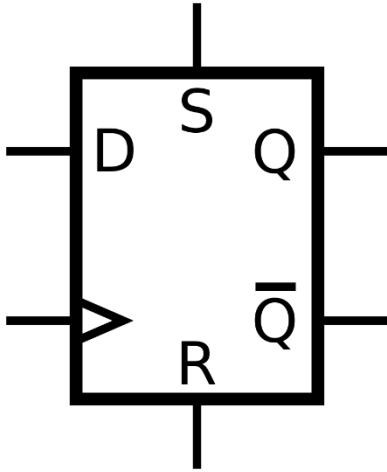
```
assign LED[3:0] = ledCount [15:12];
```

← In Verilog

You can read or load* the
value of register "ledCount"
at wire/NET "LED"

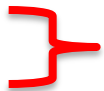
Two-types of Logic in your FPGA

Clocked (triggered) vs. Combinatorial (AKA combinational)



Clocked logic: Always block

```
1  `timescale 1 ns / 1 ps
2
3  module ledflash
4  (
5      input  wire          CLK_66MHZ,
6      input  wire          USER_RESET,
7      output wire [5:0]    LED
8  );
9
10     wire          clk;
11     wire          clk_enable;
12
13     assign clk_enable = ~USER_RESET;
14
15     BUFGCE BG (.O(clk), .CE(clk_enable), .I(CLK_66MHZ));
16     reg  [26:0] led_count;
17
18     always @(posedge clk)
19         led_count <= led_count + 1;
20
21     assign LED[5:0] = led_count[26:21];
22 endmodule
```



Clock triggered circuitry

Parallel Activity with Always Blocks

Common clock

```
always @(posedge clk_1) begin  
    flashValueR = ~flashValueR;  
end
```

```
always @(posedge clk_1) begin  
    flashValueG = ~flashValueG;  
end
```

```
always @(posedge clk_1) begin  
    flashValueB = ~flashValueB;  
end
```

Independent clocks

```
always @(posedge clk_1) begin  
    flashValueR = ~flashValueR;  
end
```

```
always @(posedge clk_2) begin  
    flashValueG = ~flashValueG;  
end
```

```
always @(posedge clk_3) begin  
    flashValueB = ~flashValueB;  
end
```

You can have many on the same clock or on different clocks*

Combanatorial Logic: assign statement

```
➔ assign clk_enable = ~USER_RESET;

    BUFGCE BG (.O(clk), .CE(clk_enable), .I(CLK_66MHZ));
    reg [26:0] led_count;

    always @(posedge clk)
        led_count <= led_count + 1;

➔➔ assign LED[5:0] = led_count[26:21];
➔➔ assign GATE_OUT_Y = GATE_IN_A & GATE_IN_B;
endmodule
```

Connected, even outside the “loop”

Results happen instantly (less propagation delay) without need for a clock

Combinatorial: Connected at start

MCU world:

```
main() {  
int flashValue = 0;
```

```
while (1) {  
    if (flashValue == 0) {  
        flashValue = 1;  
    } else {  
        flashValue = 0;  
    }  
}
```

```
PORTA.0 = flashValue;  
}
```

Nothing happens 😞

FPGA world:

```
module flashStuff  
(  
    input clk,  
    output flash  
);
```

```
reg flashValue;
```

```
always @(posedge clk) begin  
    flashValue = ~flashValue;  
end
```

```
assign flash = flashValue;  
endmodule
```

Hey – It flashes! 😊

loop

Connected, even
outside the "loop"

MCU vs. FPGA: Parallel vs. serial

```
main() {  
  - some C code  
  
  funct_1();  
  
  funct_2();  
  
  funct_3();  
  
  - some C code  
}
```

MCU world:
In sequence

```
module flashStuff  
  - some Verilog stuff  
  
  always @(posedge clk) begin  
    Some stuff 1;  
  end  
  
  always @(posedge clk) begin  
    Some stuff 2;  
  end  
  
  always @(posedge clk) begin  
    Some stuff 3;  
  end  
  
  - some Verilog stuff  
endmodule
```

FPGA world:
Simultaneously

“Registering”

Experts talked about “registering” a signal

Brain thinks:

“Registering”... Maybe like registering a Windows .dll or something...



Major case of overthinking.

“Registering” just means put the signal in a register

Blocking / Non-blocking

= blocking (better described as “immediate” – real time game)

<= non-blocking (clock based – like a turn based game. You do stuff and read it all at the end of the turn)

```
assign GATE_OUT_Y = GATE_IN_A & GATE_IN_B;
```

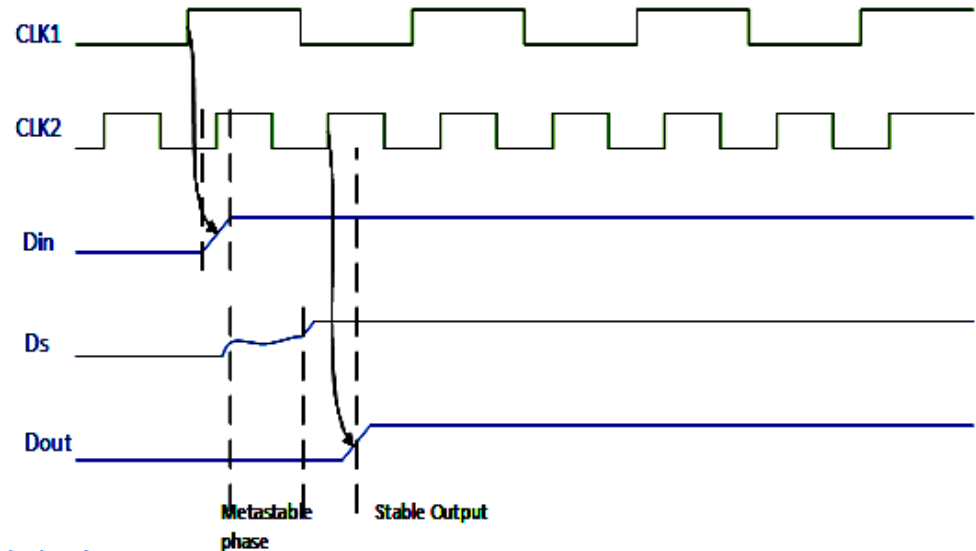
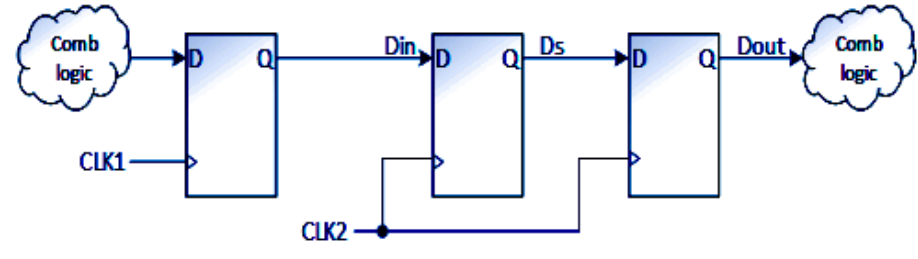
Blocking: “GATE_OUT_Y” will always, immediately reflect the results of “GATE_OUT_A” AND “GATE_OUT_B”

```
always @(posedge clk)  
    led_count <= led_count + 1;
```

Non-blocking: “led_count” will only be accurate after clock cycle

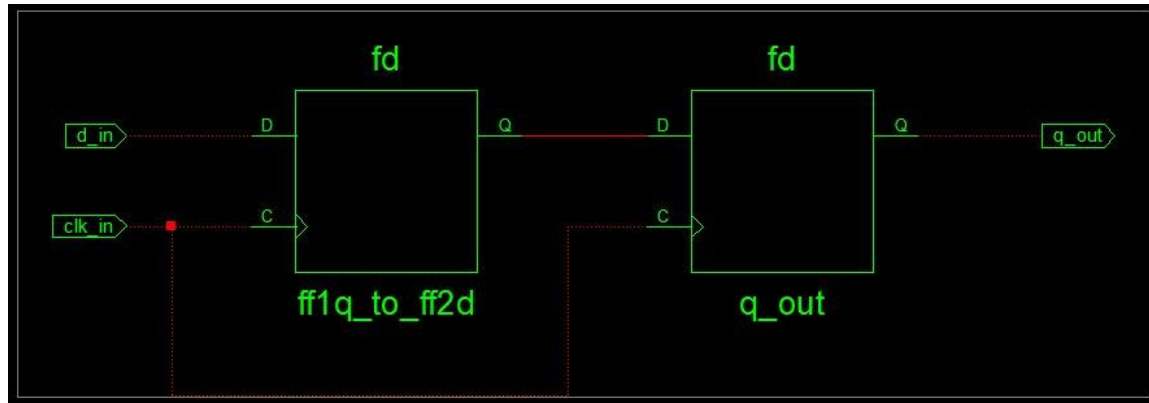
Metastability - Warning

- Clock sampling too soon
- Transition from 0 to 1 or 1 to 0 not complete yet
- Results in an unknown output
- Often caused by asynchronous inputs or using multiple clock domains



Metastability - Warning

- From the MCU world, think about key bounce, but worse and easier to solve (not an exact analogy, but close enough)
- Better to use once clock source
- Mitigated by using two or three flip flops in series

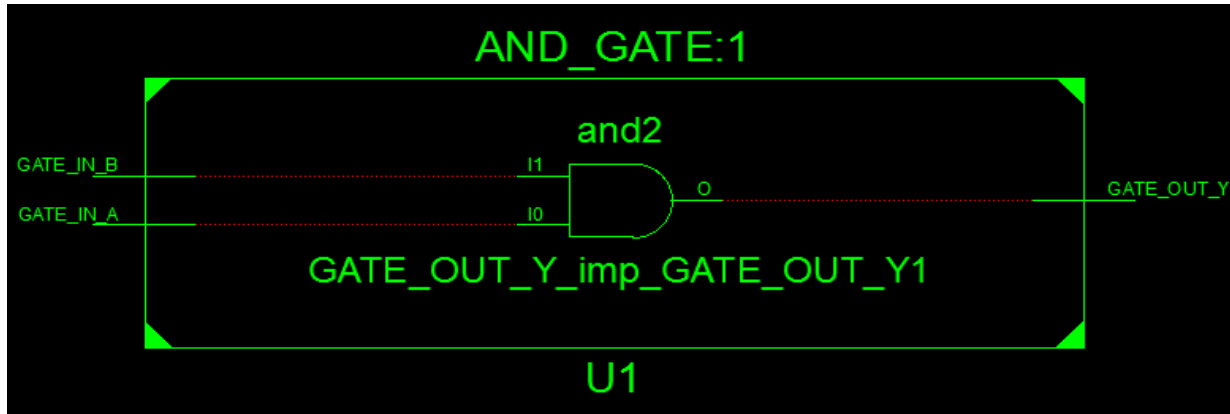
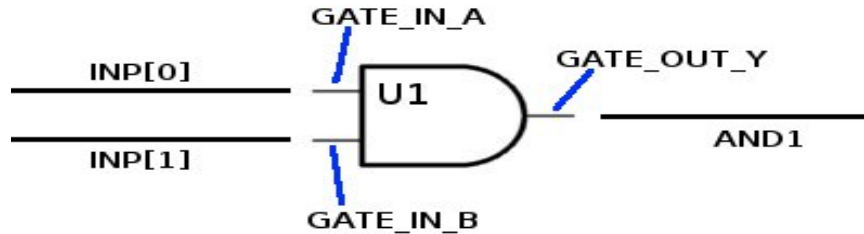


Anatomy of Libraries & Modules

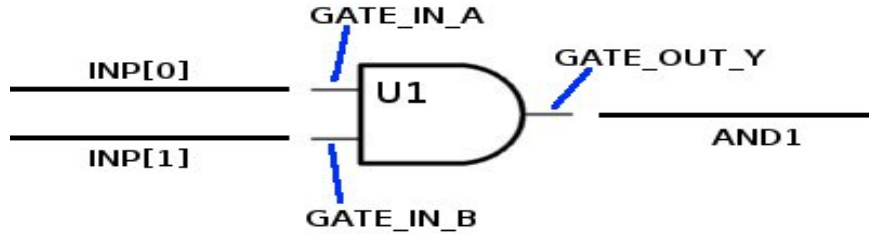
- VHDL has standard and user developed libraries
- Verilog code can be separated out as a module
- They look like typical software libraries and functions

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.numeric_std.all;  
  
Library UNISIM;  
use UNISIM.vcomponents.all;
```

Simple Example



Simple Example



Logic Symbol

The module In Verilog:

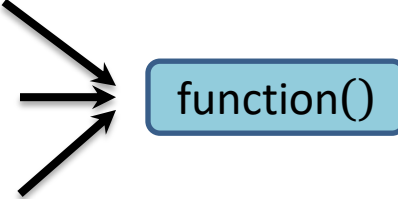
```
module AND_GATE (  
    input GATE_IN_A,  
    input GATE_IN_B,  
    output GATE_OUT_Y  
);  
    assign GATE_OUT_Y = GATE_IN_A & GATE_IN_B;  
endmodule
```

Used in another Verilog file
Rather than “call”, the
term “Instantiate” is used.

Not the Same as Calling a Function

In the MCU software world

```
main() {  
  - some C code  
  
  function();  
  function();  
  function();  
  
  - some C code  
}
```



The diagram illustrates that multiple function calls in the code point to a single, shared physical code block. Three arrows originate from the three 'function();' lines and converge on a light blue rounded rectangle containing the text 'function()'.

Calls the same physical code each time
(unless multi-threaded, but that's not the analogy)

Not the Same as Calling a Function

Uses the same template when synthesizing from the HDL code

In FPGA world

```
// Instantiate flip flop
flipflops input_0 (
    .d_in(signal_in[0]),
    .clk_in(clock_sel),
    .q_out(status[0])
);

flipflops input_1 (
    .d_in(signal_in[1]),
    .clk_in(clock_sel),
    .q_out(status[1])
);

flipflops input_2 (
    .d_in(signal_in[2]),
    .clk_in(clock_sel),
    .q_out(status[2])
);
```

template

Flip flop

Flip flop

Flip flop

Creates independent flip flop module in three different physical locations



Conclusions

- I've only touched the surface
- The barriers to FPGA entry have dropped
- A lot of options
- A lot of opportunities for confusion and MCU-derived traps
- But, they are amazing tools once you get to know them.

Download this presentation at: positiveedge.today/teardown-2023-mcu-to-fpga

