FPGA 101 - WORKSHOP

HACKADAY BELGRADE 26th MAY 2018

MIODRAG MILANOVIĆ

INTRODUCTION

- Engineer
- Hardware and emulation enthusiast

- Software developer since year 2000
- Software architect at “Levi Nine” Serbia
- Worked in C,C++,C# and Java

- Working on MAME emulation project since 2006

- Gave few talks related to FPGA targeting software developers.

THANKS

- Jan Dolinaj – for helping making these boards
- David Shah – for great help in designing boards
- Grant Jennings and Lattice – for Upduino 2 board and support
- OSH Park and Drew Fustini – for sponsoring PCB production
- IcoTC and Clifford Wolf – for excellent tools and logistic support for project
- Jesús Arroyo Torrens and Juan González for APIO and IceStudio
- Risc-V and MicroPython community
- Hackaday for creating this great event

WHAT WE ARE GOING TO LEARN ?

- What is FPGA and how it works
- How to use visual design tools to create hardware solutions
- Verilog basics
- How to create audio and video
- Create CPU
- Use existing Risc-V CPU core and run C programs on it
- Run MicroPython on FPGA and program directly on it
- And much more …

APPROACH

How to actually learn any new programming concept

Essential

Changing Stuff and Seeing What Happens

O RLY?

@ThePracticalDev

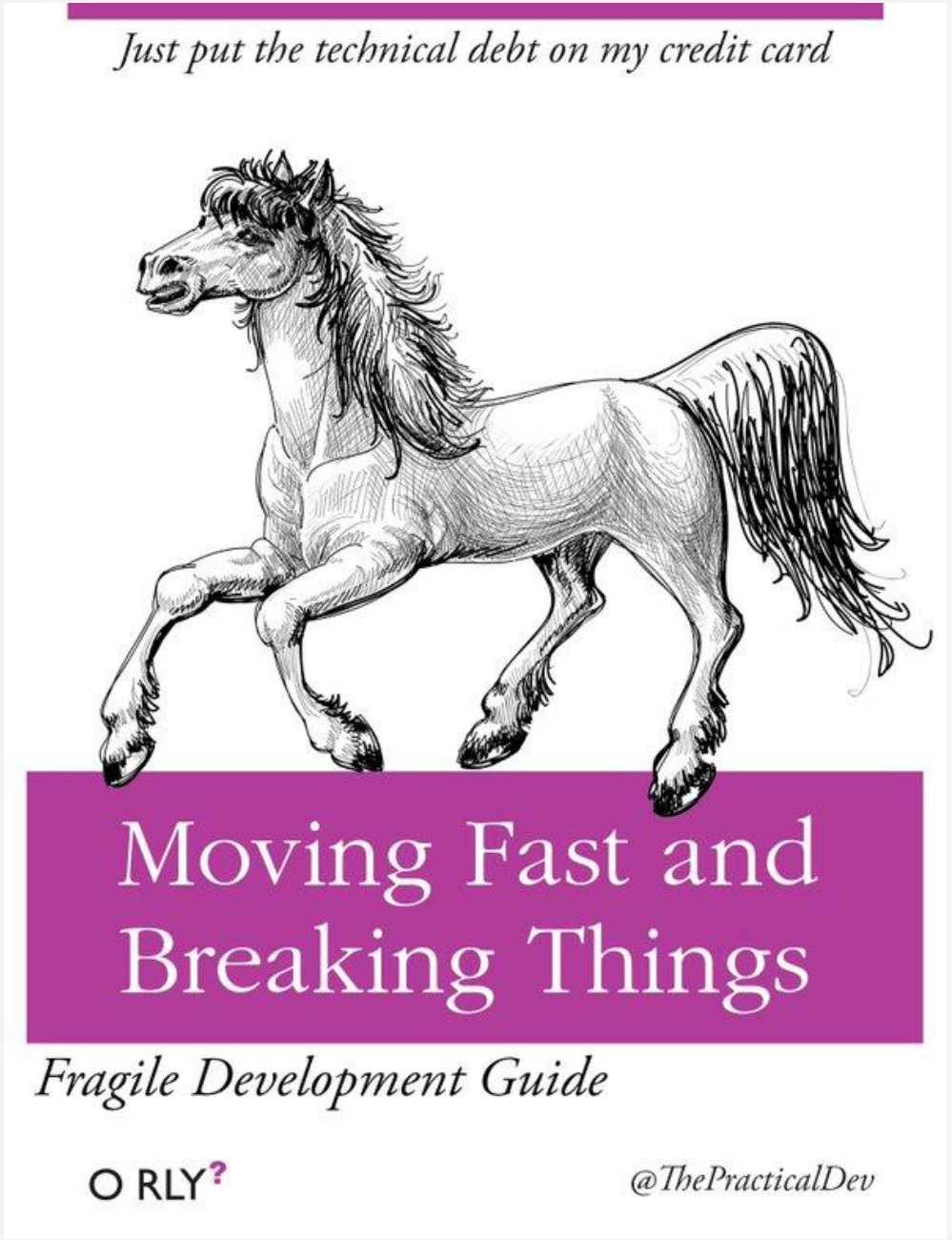
Software can be chaotic, but we make it work

Expert

Trying Stuff Until it Works

O RLY?

The Practical Developer
@ThePracticalDev


Just put the technical debt on my credit card
Moving Fast and Breaking Things
Fragile Development Guide
O RLY?
@ThePracticalDev

LET'S START

LOGIC LEVELS

1 0 X Z

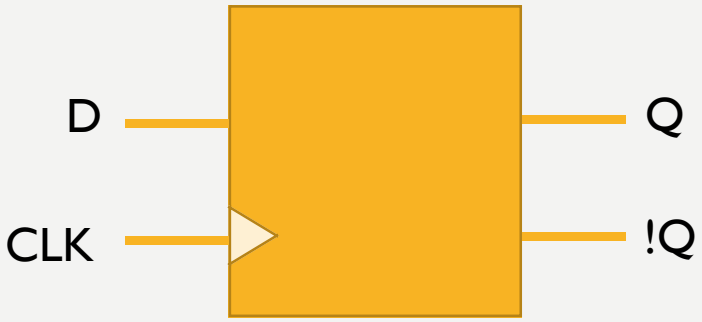
BOOLEAN OPERATIONS

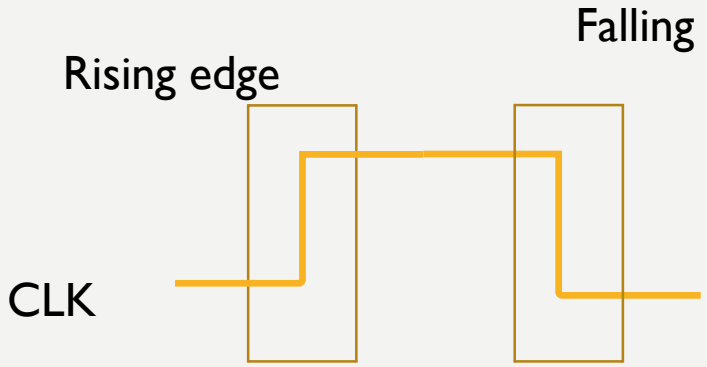
- AND
- OR
- XOR
- NOT
- NAND
- NOR
- XNOR

INPUT		OUTPUT					
A	B	A AND B	A NAND B	A OR B	A NOR B	A NOR B	A XNOR B
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

D FLIP-FLOP

CLK	D	R	S	Q	!Q
rising edge	0	0	0	0	1
rising edge	1	0	0	1	0
falling edge	x	0	0	Q	!Q
x	x	1	0	0	1
x	x	0	1	1	0
x	x	1	1	1	1

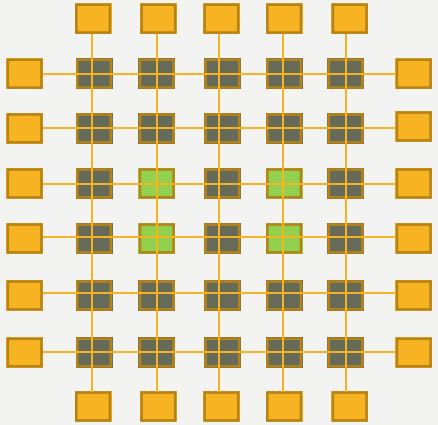




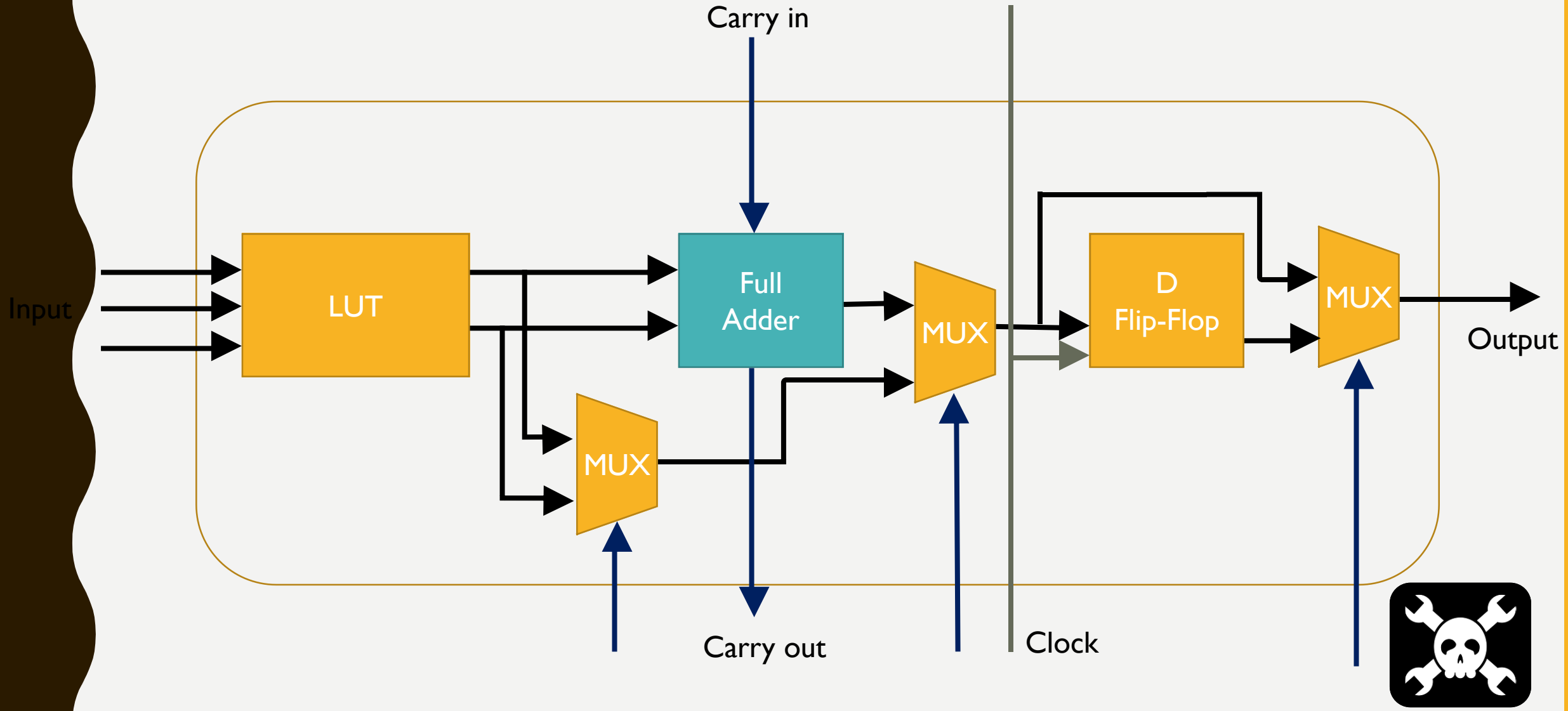
WHAT IS FPGA ?

- Field-programmable gate array
- Logic block – LUT + D Flip-Flop + full adder
- I/O block – one per each pin
- Interconnections, clock tree
- Hard block – Block RAM, multipliers, DSP, CPU,…

- Vendors: Xilinx, Intel/Altera, Lattice, Microsemi …



LOGIC BLOCK


Carry in
Input
LUT
Full
Adder
MUX
D
Flip-Flop
MUX
Output
MUX
Carry out
Clock

HOW DO WE PROGRAM FPGA ?

- HDL - Hardware description language
- VHDL and Verilog

- Analysis: parsing and validation of HDL
- Synthesis: HDL -> netlist
- Place-and-route: netlist -> specific FPGA technology
- Assembler: specific FPGA technology -> bitstream
- Programming: deploying bitstream on device (serial flash memory or directly)
- Timing analysis
- Simulation

OPEN SOURCE TOOLS

- Yosys - Verilog synthesis tool by Clifford Wolf
- Arachne PnR - Place and route tool by Cotton Seed
- Project IceStorm – Assembler, time analysis and FPGA programming tool
  by Clifford Wolf and Mathias Lasser
- APIO – micro-ecosystem for open FPGAs by Jesús Arroyo Torrens and Juan González (Obijuan)
- Icestudio - graphic editor for open FPGAs by Jesús Arroyo Torrens
- Lattice iCE40 FPGA only

AND MORE...

- Icarus Verilog by Stephen Williams
- Verilator by Wilson Snyder with Duane Galbi and Paul Wasson
- FuseSoC - package manager and a set of build tools for FPGA/ASIC development by Olof Kindgren

LET'S SETUP

INSTALL DRIVERS

- apio drivers --ftdi-enable
- apio system –lsftdi

  Number of FTDI devices found: 1

  Checking device: 0

  Manufacturer: FTDI, Description: Single RS232-HS
- apio system –lsserial

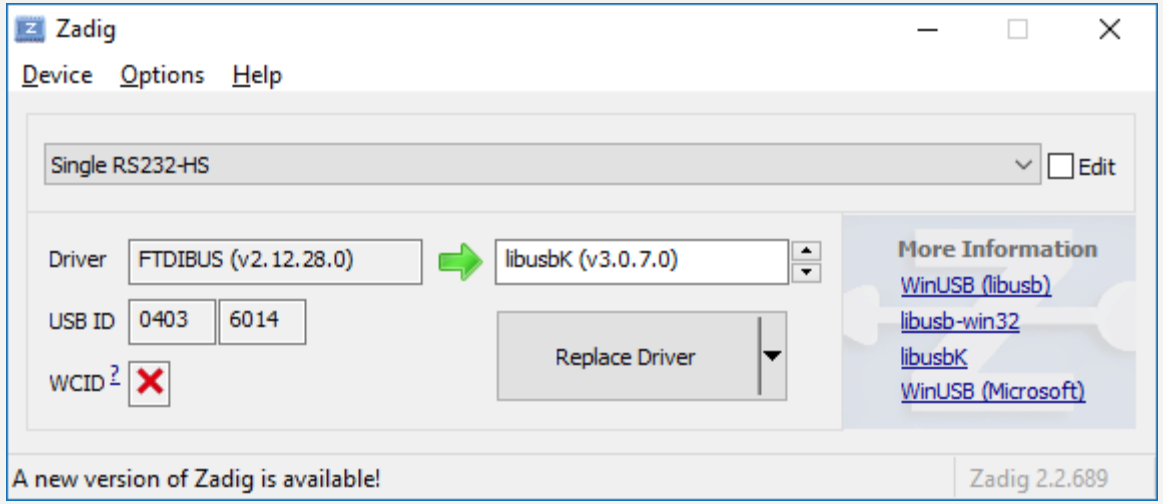
  Number of Serial devices found: 1

  COM3

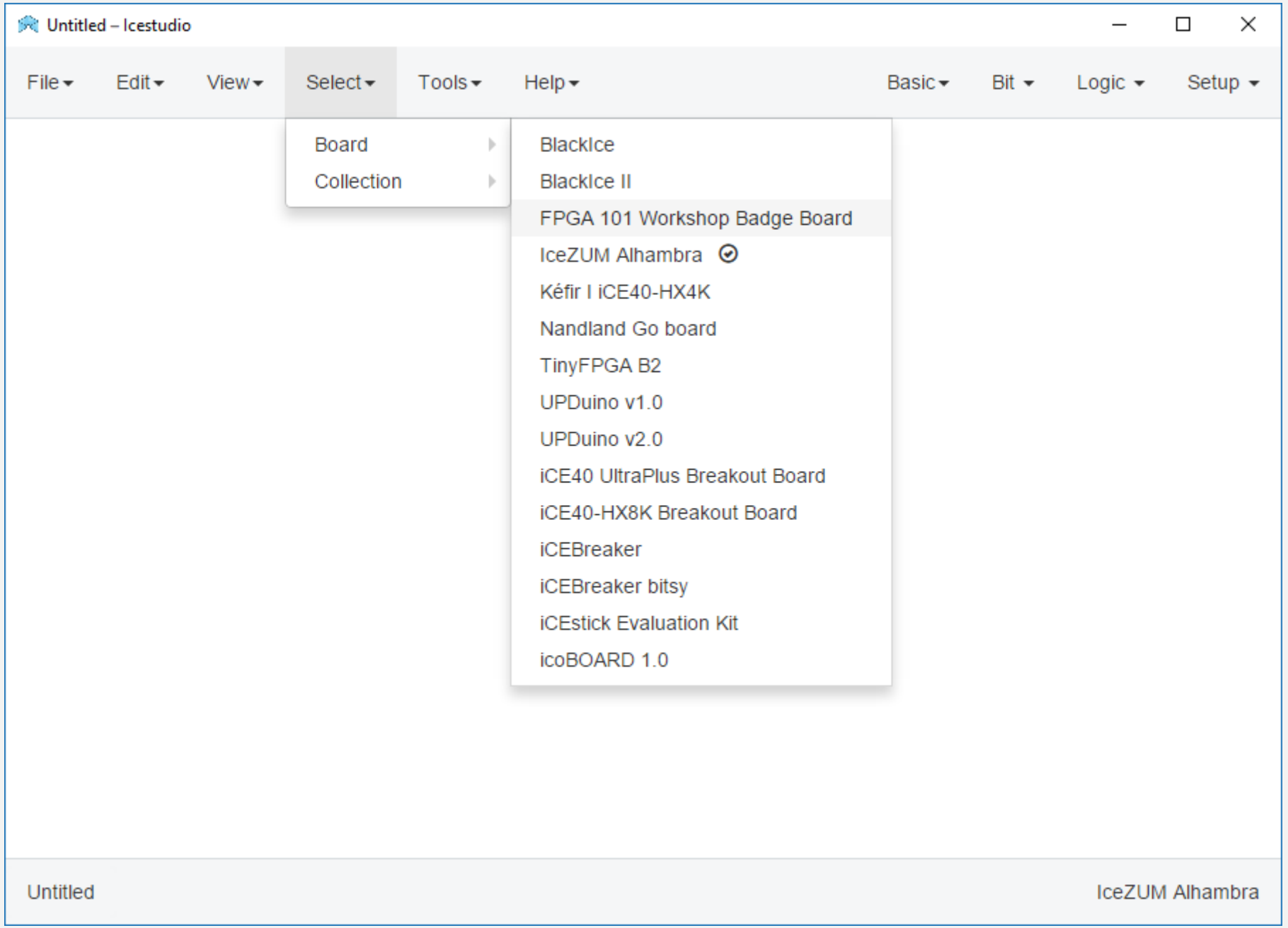
  Description: USB-SERIAL CH340 (COM3)

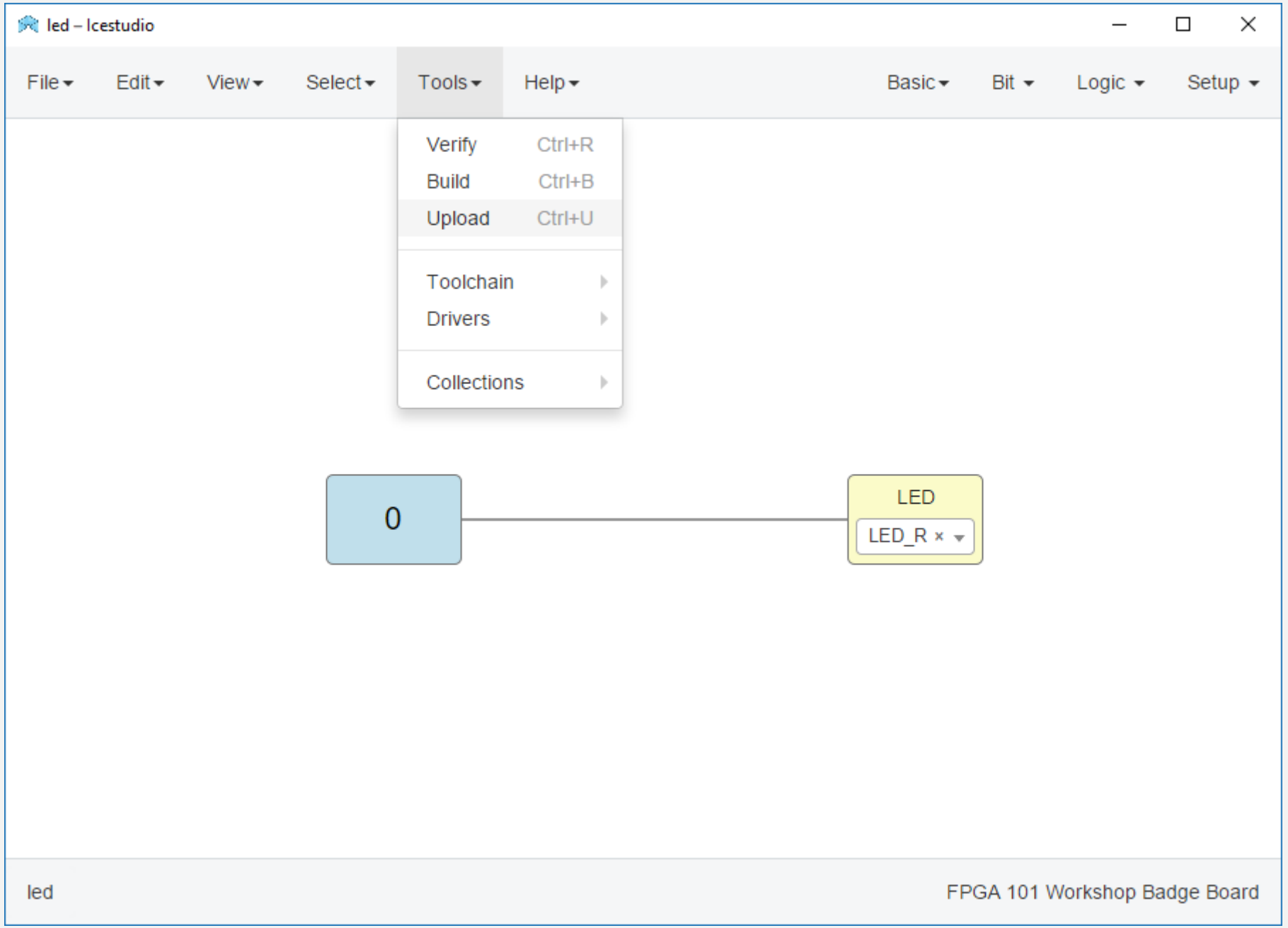
  Hardware info: USB VID:PID=1A86:7523 SER=5 LOCATION=1-13

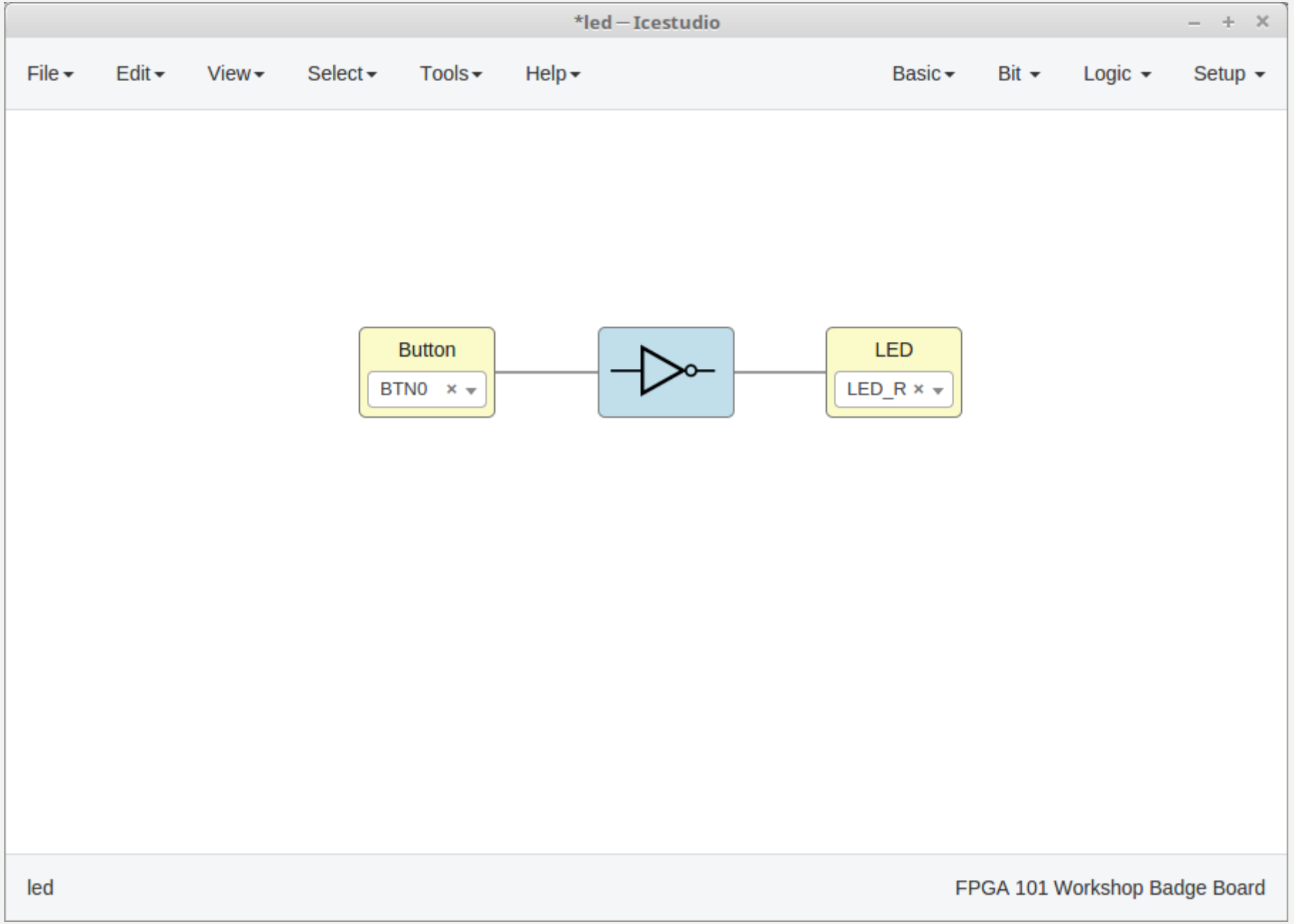
WINDOWS UPDATE DRIVER

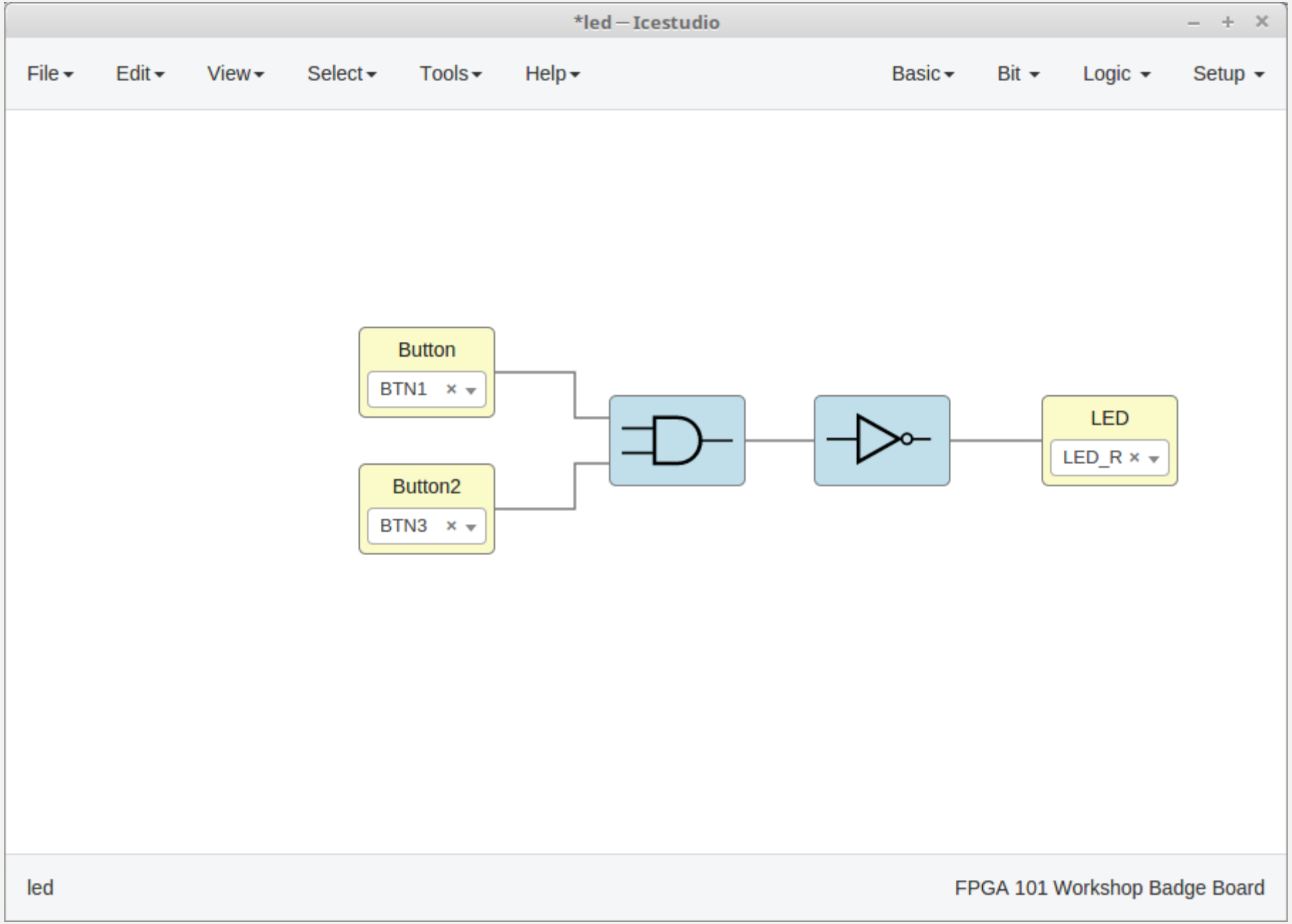


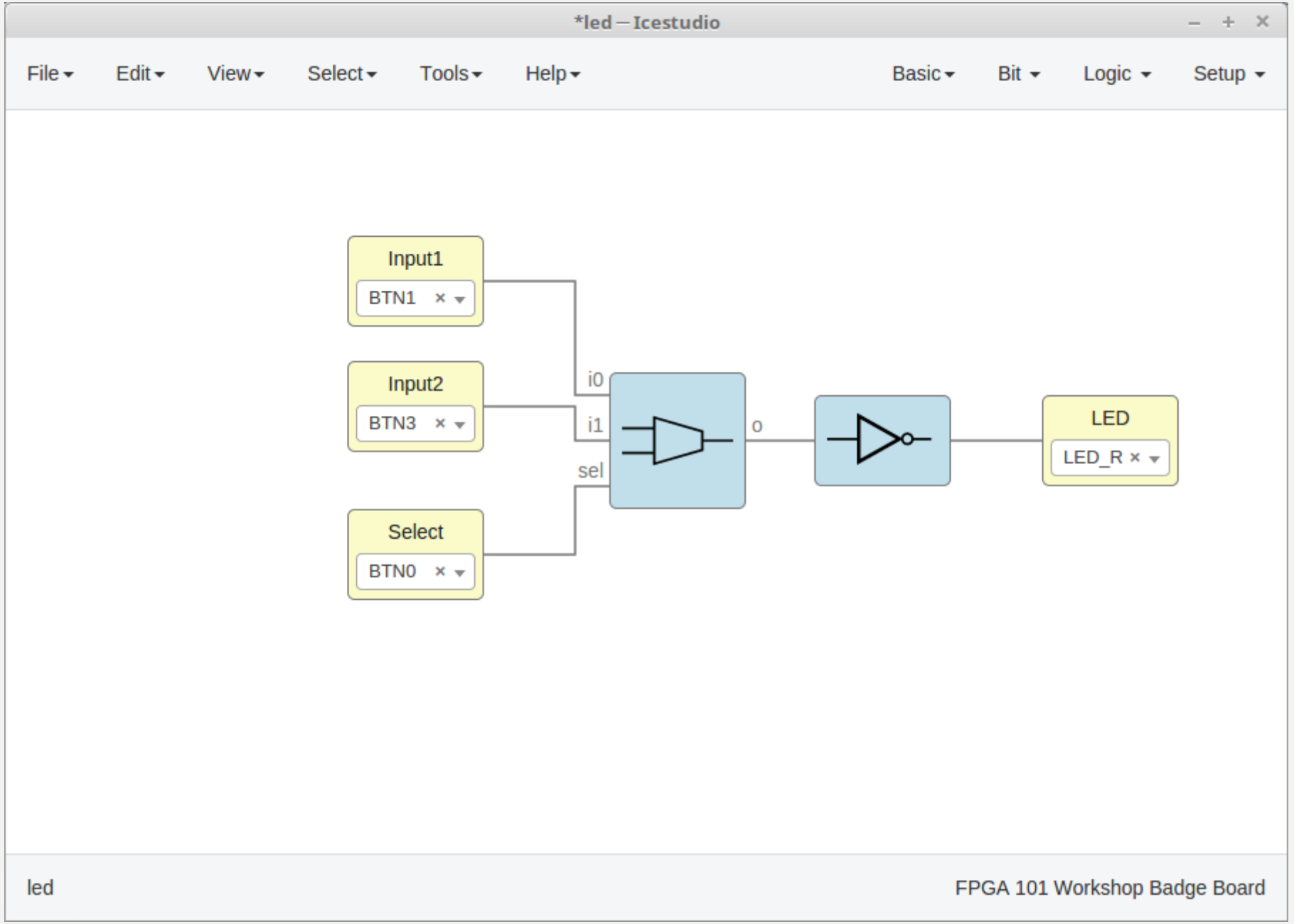
ICESTUDIO

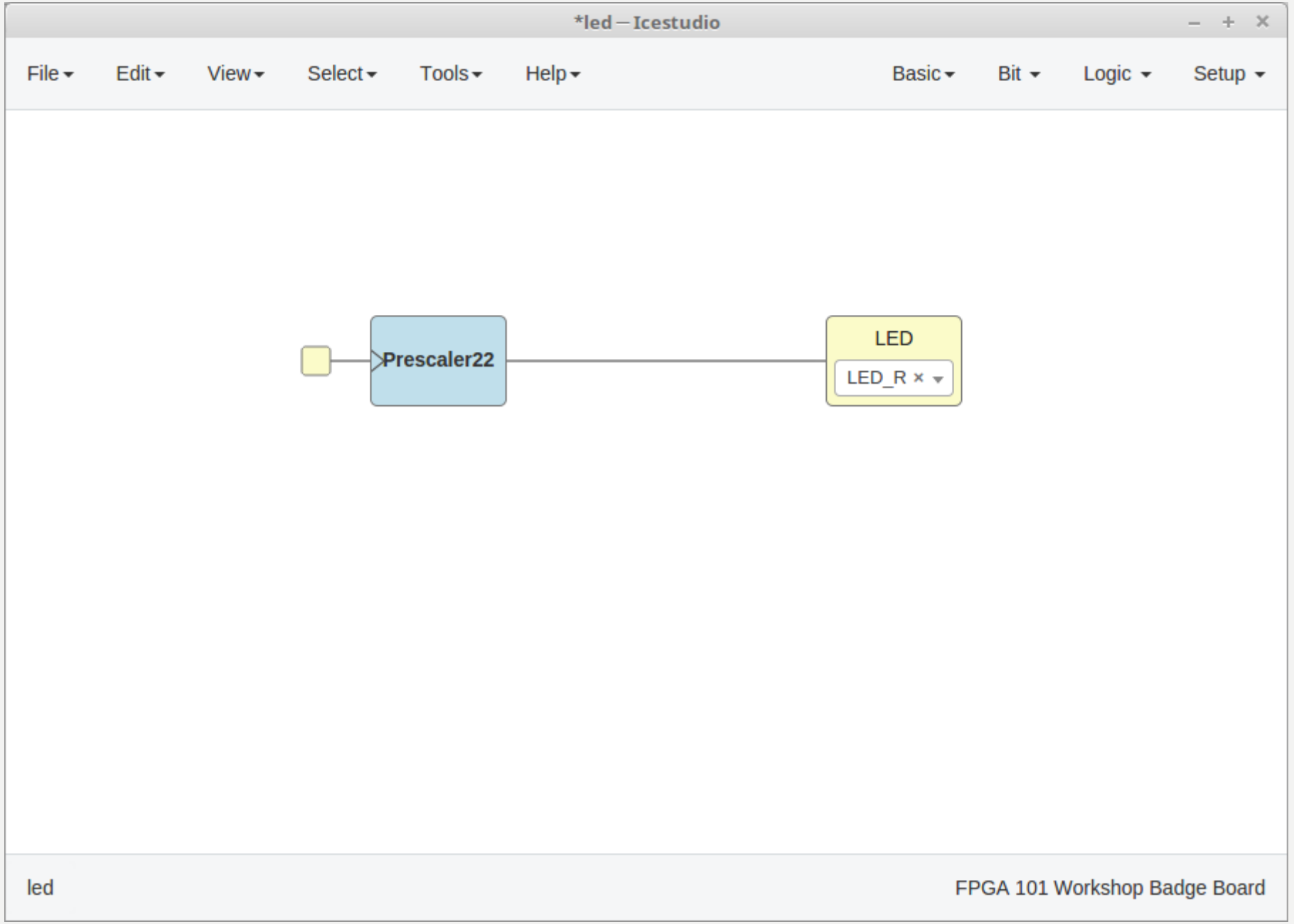

Untitled – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Board
Collection
BlackIce
BlackIce II
FPGA 101 Workshop Badge Board
IceZUM Alhambra
Kéfir I iCE40-HX4K
Nandland Go board
TinyFPGA B2
UPDuino v1.0
UPDuino v2.0
iCE40 UltraPlus Breakout Board
iCE40-HX8K Breakout Board
iCEBreaker
iCEBreaker bitsy
iCEstick Evaluation Kit
icoBOARD 1.0
Untitled
IceZUM Alhambra

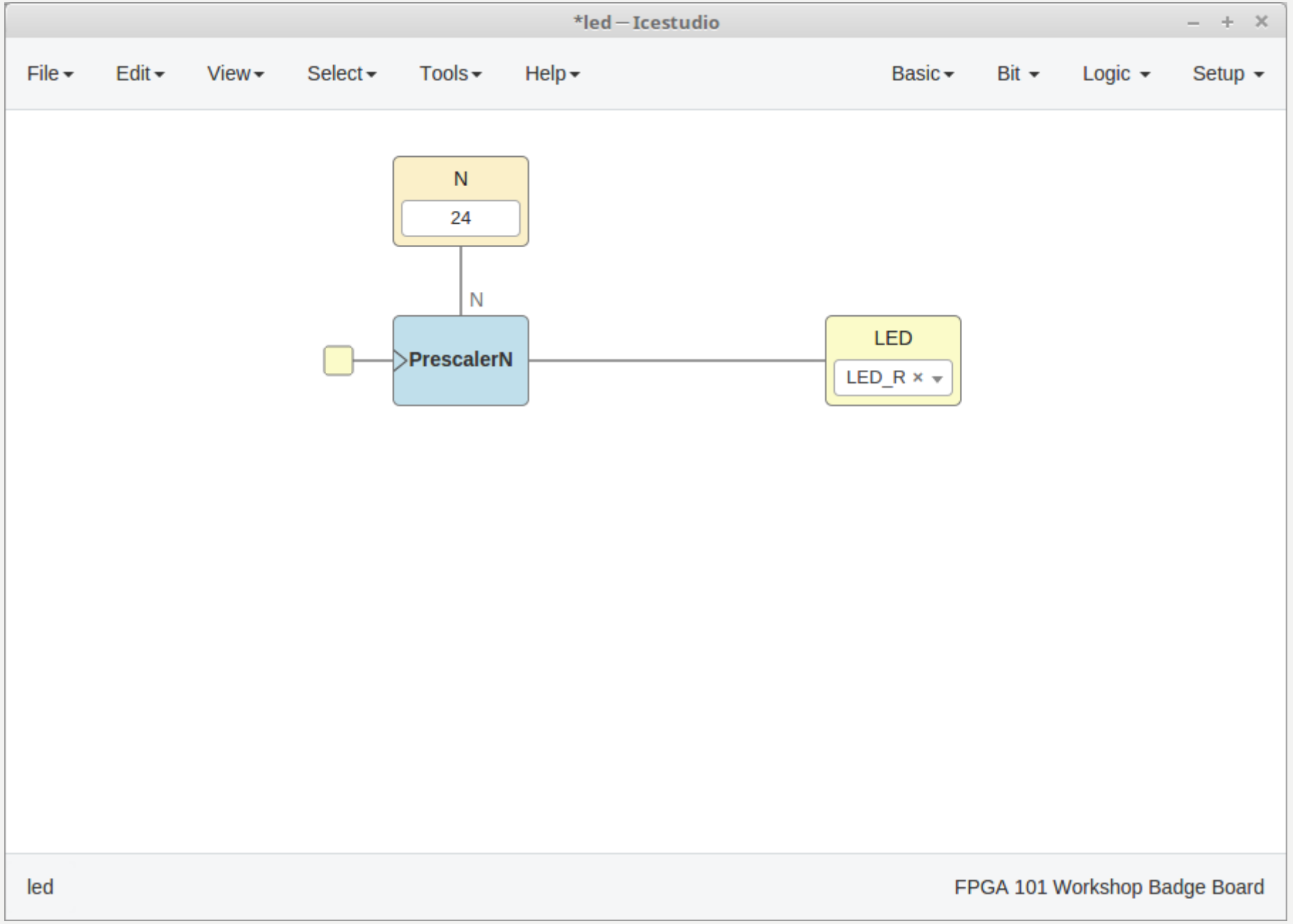

led – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Verify Ctrl+R
Build Ctrl+B
Upload Ctrl+U
Toolchain
Drivers
Collections
0
LED
LED_R
led
FPGA 101 Workshop Badge Board

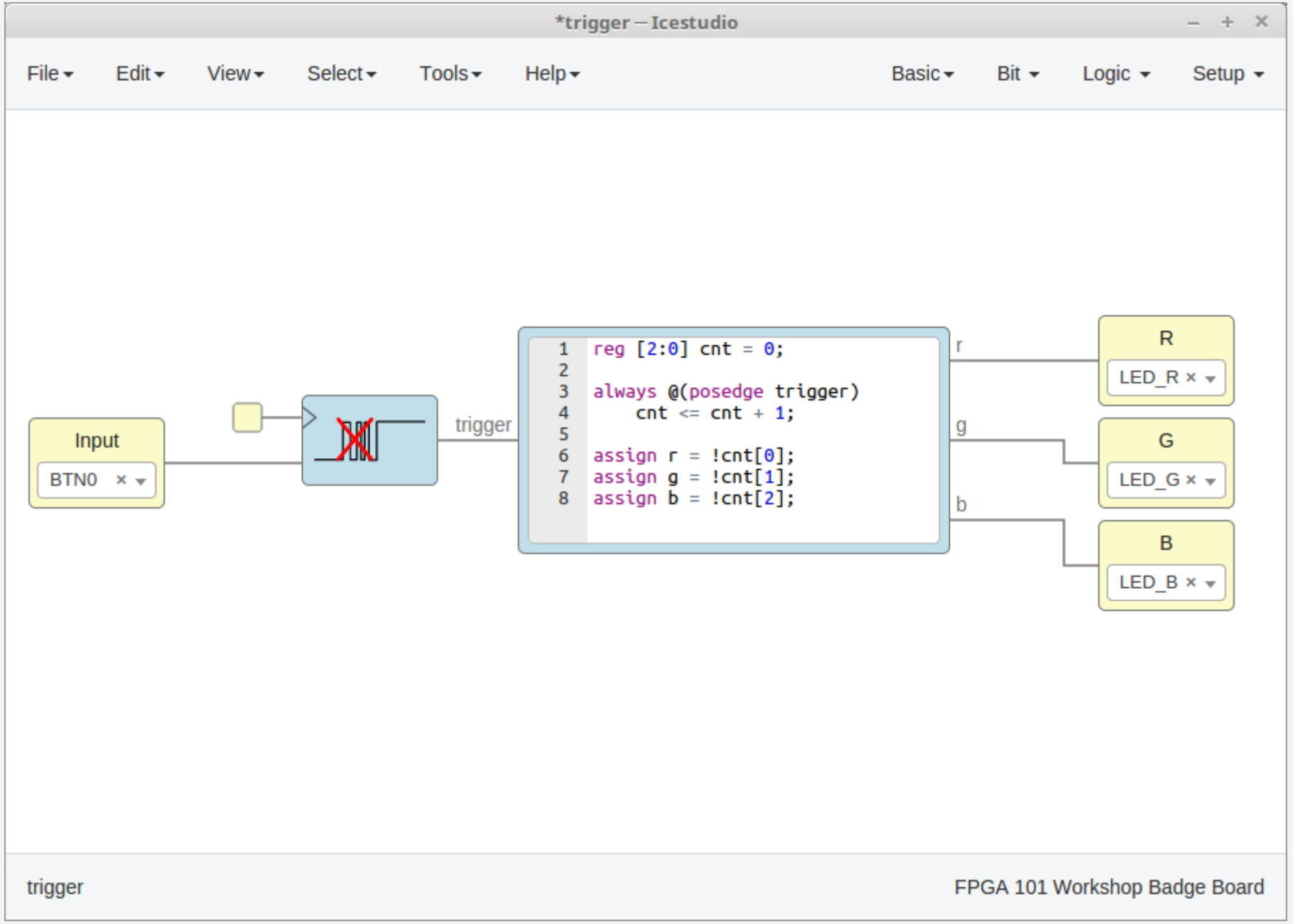

*led – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Button
BTN0
LED
LED_R
led
FPGA 101 Workshop Badge Board

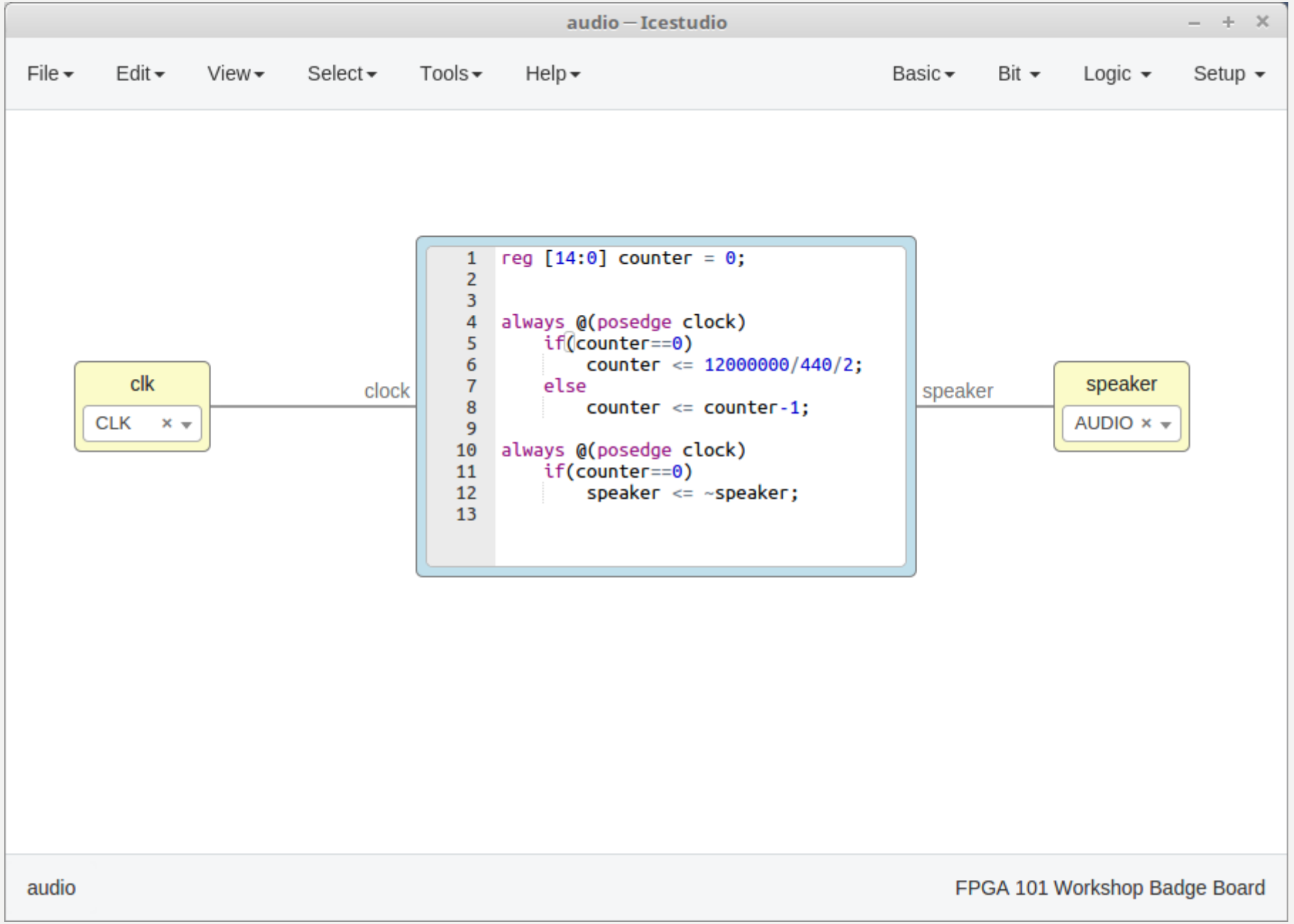

*led – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Button
BTN1
Button2
BTN3
LED
LED_R
led
FPGA 101 Workshop Badge Board


*led – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Input1
BTN1
Input2
BTN3
Select
BTN0
i0
i1
sel
o
LED
LED_R
led
FPGA 101 Workshop Badge Board


*led – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Prescaler22
LED
LED_R
led
FPGA 101 Workshop Badge Board


*led – Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
N
24
N
PrescalerN
LED
LED_R
led
FPGA 101 Workshop Badge Board


*trigger — Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
Input
BTN0
trigger
1 reg [2:0] cnt = 0;
2
3 always @(posedge trigger)
4 cnt <= cnt + 1;
5
6 assign r = !cnt[0];
7 assign g = !cnt[1];
8 assign b = !cnt[2];
r
g
b
R
LED_R
G
LED_G
B
LED_B
trigger
FPGA 101 Workshop Badge Board


audio — Icestudio
File
Edit
View
Select
Tools
Help
Basic
Bit
Logic
Setup
clk
CLK
clock
1 reg [14:0] counter = 0;
2
3
4 always @(posedge clock)
5 if(counter==0)
6 counter <= 12000000/440/2;
7 else
8 counter <= counter-1;
9
10 always @(posedge clock)
11 if(counter==0)
12 speaker <= ~speaker;
13
speaker
speaker
AUDIO
audio
FPGA 101 Workshop Badge Board

BASICS

CREATE APIO PROJECT

```
apio init --board fpga101

Creating apio.ini file ...
File 'apio.ini' has been successfully created!
```

```
[env]
board = fpga101
```

PCF FILE

```
# 12 MHz clock
set_io -nowarn CLK 35 # input

# LCD connector
set_io -nowarn LCD_CLK 18 # output
set_io -nowarn LCD_DEN 19 # output
set_io -nowarn LCD_VS  11 # output
set_io -nowarn LCD_HS  9  # output
set_io -nowarn LCD_RST 2  # output
set_io -nowarn LCD_D7  6  # output
set_io -nowarn LCD_D6  44 # output
set_io -nowarn LCD_D5  4  # output
set_io -nowarn LCD_D4  3  # output
set_io -nowarn LCD_D3  48 # output
set_io -nowarn LCD_D2  45 # output
set_io -nowarn LCD_D1  47 # output
set_io -nowarn LCD_D0  46 # output
set_io -nowarn LCD_PWM 13 # output

# Buttons
set_io -nowarn B0 23 # input
set_io -nowarn B1 25 # input
set_io -nowarn B2 26 # input
set_io -nowarn B3 27 # input
set_io -nowarn B4 21 # input
set_io -nowarn B5 12 # input

# Audio jack
set_io -nowarn AUDIO 37 # output
```

```
# PMOD connector
set_io -nowarn PMOD0 32
set_io -nowarn PMOD1 31
set_io -nowarn PMOD2 34
set_io -nowarn PMOD3 43
set_io -nowarn PMOD4 36
set_io -nowarn PMOD5 42
set_io -nowarn PMOD6 38
set_io -nowarn PMOD7 28

# RGB LED Driver
set_io -nowarn LED_G 39 # output
set_io -nowarn LED_B 40 # output
set_io -nowarn LED_R 41 # output
```

LED.V (VERILOG FILE)

```
module led (
    output LED_B
);
    assign LED_B = 1'b0;
endmodule
```

LED - COUNTER

COUNTER

```
module rgb (
     input CLK,
     output LED_R,
     output LED_G,
     output LED_B,
     output LCD_PWM
);
assign LCD_PWM = 1'b0;
reg [32:0] clk_cnt;

initial
begin
     clk_cnt = 0;
end

always @(posedge CLK)
begin
     clk_cnt <= clk_cnt + 1;
end

assign LED_R = !clk_cnt[2];
assign LED_G = !clk_cnt[3];
assign LED_B = !clk_cnt[4];
endmodule
```

TEST BENCH

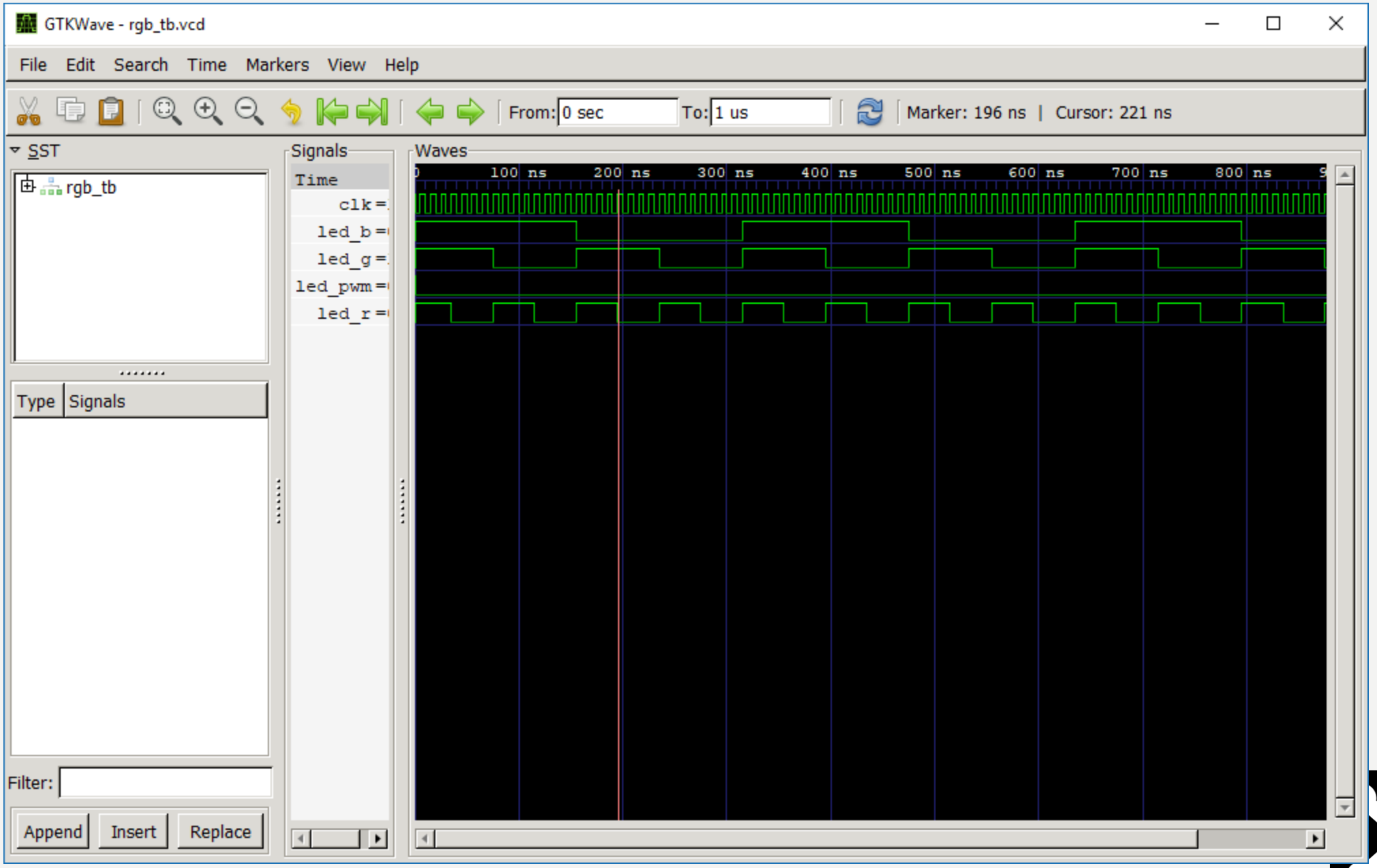
```
`timescale 1ns/1ps
module rgb_tb();
    reg clk = 0;
     wire led_r;
     wire led_g;
     wire led_b;
     wire led_pwm;

    rgb leds(.CLK(clk),.LED_R(led_r),.LED_G(led_g),
          .LED_B(led_b),.LCD_PWM(led_pwm));

    always
        #(5) clk <= !clk;

    initial
    begin
        $dumpfile("rgb_tb.vcd");
        $dumpvars(0,rgb_tb);
        #1000
        $finish;
    end
endmodule
```

APIO TOOLS

```
apio verify
iverilog -o hardware.out -D VCD_OUTPUT= "C:\msys64\opt\homedir\.apio\packages\toolchain-iverilog\vlib\cells_sim.v" rgb.v
========================== [SUCCESS] Took 0.30 seconds==============================
```

```
apio lint
verilator --lint-only -IC:\msys64\opt\homedir\.apio\packages\toolchain-verilator\share rgb.v
========================== [SUCCESS] Took 0.21 seconds==============================
```

```
apio sim
iverilog -o rgb_tb.out -D VCD_OUTPUT=rgb_tb "C:\msys64\opt\homedir\.apio\packages\toolchain-iverilog\vlib\cells_sim.v" rgb.v rgb_tb.v
vvp rgb_tb.out
VCD info: dumpfile rgb_tb.vcd opened for output.
gtkwave rgb_tb.vcd rgb_tb.gtkw
```


GTKWave - rgb_tb.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 1 us
Marker: 196 ns | Cursor: 221 ns
SST
rgb_tb
Type Signals
Filter:
Append Insert Replace
Signals
Time
clk
led_b
led_g
led_pwm
led_r
Waves
100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns 800 ns

BUTTONS

DIRECT CONNECTION

```
module button_rgb (
    input CLK,
    output LED_R,
    output LED_G,
    output LED_B,
    output LCD_PWM,
    input B0,
    input B1,
    input B2
);
assign LCD_PWM = 1'b0;
assign LED_R = !B0;
assign LED_G = !B1;
assign LED_B = !B2;
endmodule
```

BUTTON AS TRIGGER

```
module button_cnt (
    input CLK,
    output LED_R,
    output LED_G,
    output LED_B,
    output LCD_PWM,
    input B0
);

reg [2:0] cnt = 0;

always @(posedge B0)
begin
    cnt <= cnt + 1;
end

assign LCD_PWM = 1'b0;
assign LED_R = !cnt[0];
assign LED_G = !cnt[1];
assign LED_B = !cnt[2];
endmodule
```

BUTTON DEBOUNCER

```
module button_rgb (
     input CLK, output LED_R, output LED_G, output LED_B,
     output LCD_PWM, input B0
);

reg [24:0] clk_cnt;
reg [2:0] cnt = 0;
reg prev = 0;

always @(posedge CLK)
begin
     clk_cnt <= clk_cnt + 1;
end

always @(posedge clk_cnt[20])
begin
     prev <= B0;
     if (prev == 1'b0 && B0 == 1'b1)
          cnt <= cnt + 1;
end

assign LCD_PWM = 1'b0;
assign LED_R = !cnt[0];
assign LED_G = !cnt[1];
assign LED_B = !cnt[2];
endmodule
```

EXERCISE

- Map different buttons to different RGB colors

AUDIO

THEORY

- Digital – Analog Converter
- PWM
- We will change frequency of change, but keep 50% duty cycle

PLAY SOUNDS

```
module audio(
    input CLK,
    output LCD_PWM,
    input B0,
    input B1,
    input B2,
    input B3,
    input B4,
    input B5,
    output reg AUDIO);

parameter TONE_A4 = 12000000/440/2;
parameter TONE_B4 = 12000000/494/2;
parameter TONE_C5 = 12000000/523/2;
parameter TONE_D5 = 12000000/587/2;
parameter TONE_E5 = 12000000/659/2;
parameter TONE_F5 = 12000000/698/2;
parameter TONE_G5 = 12000000/783/2;

reg [14:0] counter;
```

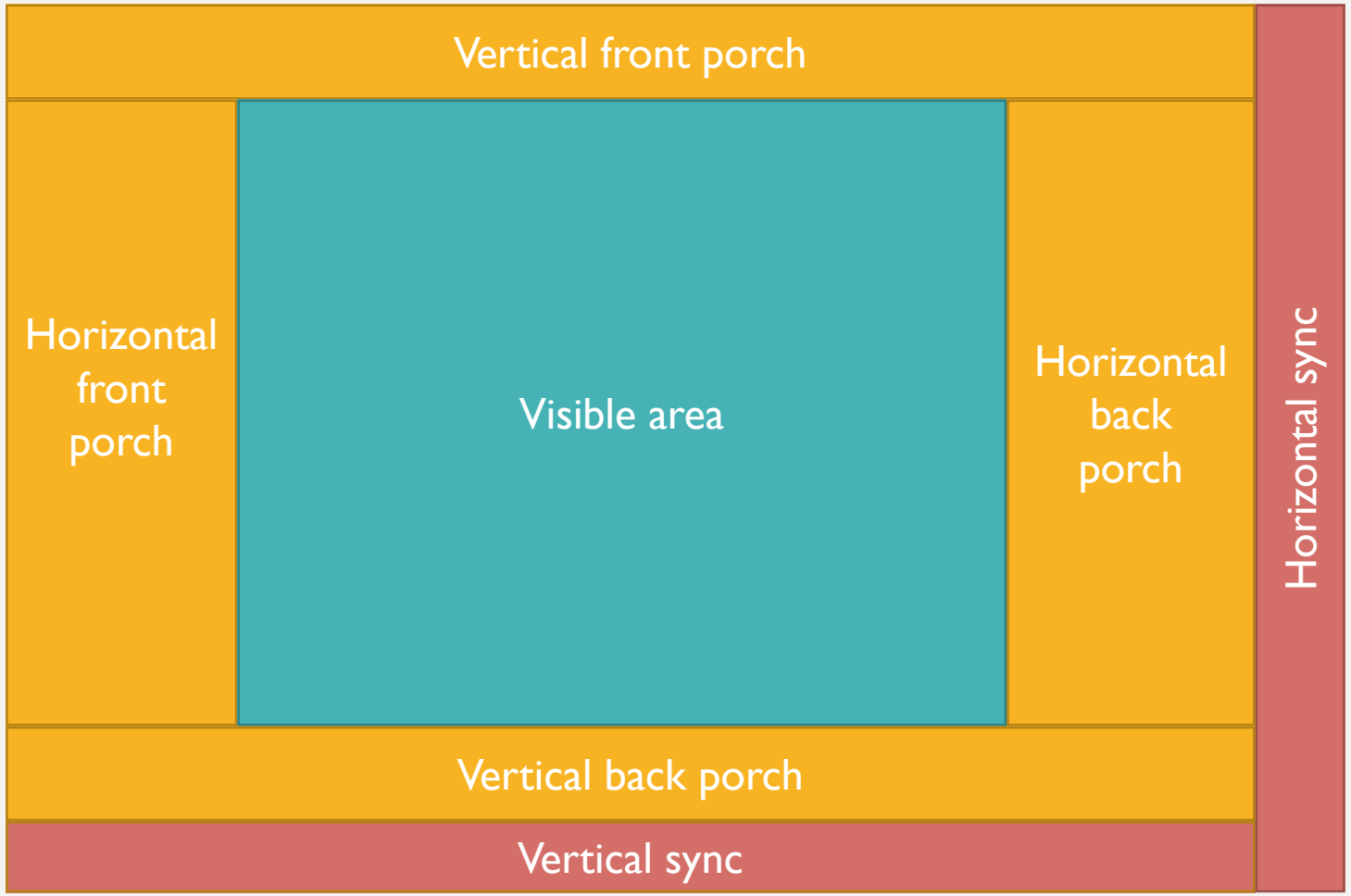
```
always @(posedge CLK)
if(counter==0)
counter <= (
    B0 ? TONE_A4-1 :
    B1 ? TONE_B4-1 :
    B2 ? TONE_C5-1 :
    B3 ? TONE_D5-1 :
    B4 ? TONE_E5-1 :
    B5 ? TONE_F5-1 : 0);
else
    counter <= counter-1;

always @(posedge CLK)
    if(counter==0)
        AUDIO <= ~AUDIO;

assign LCD_PWM = 1'b0;
endmodule
```

EXERCISE

- Make police car siren sound
- Play full octave round and round

VIDEO

THEORY


Vertical front porch
Horizontal front porch
Visible area
Horizontal back porch
Horizontal sync
Vertical back porch
Vertical sync

CLOCK GENERATOR

```
/* -------------------------
Clock generator
-------------------------*/

wire clkhf;
wire locked;
wire pixclk;

(* ROUTE_THROUGH_FABRIC=1 *)
SB_HFOSC #(.CLKHF_DIV("0b01")) hfosc_i (
.CLKHFEN(1'b1),
.CLKHFPU(1'b1),
.CLKHF(clkhf)
);

pll pll_i(.clock_in(clkhf), .clock_out(pixclk), .locked(locked))
```

PLL

```
/*
* Given input frequency: 24.000 MHz
* Requested output frequency: 19.200 MHz
* Achieved output frequency: 19.125 MHz
*/
module pll(
    input clock_in,
    output clock_out,
    output locked
);

SB_PLL40_CORE #(
        .FEEDBACK_PATH("SIMPLE"),
        .DIVR(4'b0001),     // DIVR = 1
        .DIVF(7'b0110010),  // DIVF = 50
        .DIVQ(3'b101),      // DIVQ = 5
        .FILTER_RANGE(3'b001)   // FILTER_RANGE = 1
    ) uut (
        .LOCK(locked),
        .RESETB(1'b1),
        .BYPASS(1'b0),
        .REFERENCECLK(clock_in),
        .PLLOUTGLOBAL(clock_out)
        );

endmodule
```

VIDEO GENERATOR MODULE

```
module video (
    input clk, //19.2MHz pixel clock in
    input resetn,
    output reg [7:0] lcd_dat,
    output reg lcd_hsync,
    output reg lcd_vsync,
    output reg lcd_den,
    output reg [9:0] h_pos,
    output reg [9:0] v_pos,
    input [23:0] rgb_data
);
```

ONE COLOR SCREEN

```
assign rgb_data = 24'hff0000;
```

- Exercise:
  - Try different colors
  - Try to change color depending of button 0

CHECKERS

```
assign rgb_data = (h_pos < 320) ?
       h_pos[4] ^ v_pos[4] ? 24'h000000 : 24'hffffff :
       24'h000000;
```

STRIPES

```
assign rgb_data = (h_pos < 64) ? {h_pos[5:0], 18'd0} :
    (h_pos < 128) ? {8'd0, h_pos[5:0], 10'd0} :
    (h_pos < 192) ? {16'd0, h_pos[5:0], 2'd0} :
    (h_pos < 256) ? {h_pos[5:0], 2'd0, h_pos[5:0], 2'd0, h_pos[5:0], 2'd0} :
    (h_pos < 320) ? {h_pos[5:0], 2'd0, 8'd0, h_pos[5:0], 2'd0} :
    24'd0;
```

KEY TEST

```
assign rgb_data =
(h_pos > 50 && h_pos < 100 && v_pos > 100 && v_pos < 150 && B0) ? 24'hffffff :
(h_pos > 100 && h_pos < 150 && v_pos > 50 && v_pos < 100 && B1) ? 24'hffffff :
(h_pos > 150 && h_pos < 200 && v_pos > 100 && v_pos < 150 && B2) ? 24'hffffff :
(h_pos > 100 && h_pos < 150 && v_pos > 150 && v_pos < 200 && B3) ? 24'hffffff :
(h_pos > 250 && h_pos < 300 && v_pos > 50 && v_pos < 100 && B5) ? 24'hff6633 :
(h_pos > 250 && h_pos < 300 && v_pos > 150 && v_pos < 200 && B4) ? 24'hff6633 :
24'h000000;
```

EXERCISE

- Update key-test example according to your board layout
- Try generating flag image
- Experiment with idea of your own ☺

STATE MACHINES

SIMPLE STATE MACHINE

```
reg [32:0] clk_cnt;

reg [1:0] state = 2'b00;

initial
begin
     rgb_data <= 24'h0000ff;
end

always @(posedge pixclk)
begin
     clk_cnt <= clk_cnt + 1;

     case(state)
          2'b00 : begin
                        if (clk_cnt > 32'h02ffffff)
                        begin
                             rgb_data <= 24'hffffff;
                             clk_cnt <= 32'd0;
                             state <= 2'b01;
                        end
                  end
          2'b01 : begin
```

EXERCISE

- Add more states
- Make buttons change state

MEMORY

SYNCED MEMORY

```
module font_rom(
    input clk,
    input [11:0] addr,
    output [7:0] data_out
);

reg [7:0] store[0:4095];

initial
begin
    $readmemh("font_vga.mem", store);
end

always @(posedge clk)
    data_out <= store[addr];
endmodule
```

EXERCISE

- Change font/background color in example
- Use your of custom text displayed
- Try making text to be double width or double height when displayed

SPRITES

4 BIT WIDE INDEXED MEMORY

```
module sprite_rom(
    input clk,
    input [11:0] addr,
    output [23:0] data_out
);

wire [23:0] palette[0:15];

assign palette[0] = 24'h000000;
assign palette[1] = 24'h180002;
assign palette[2] = 24'h240001;
assign palette[3] = 24'h390100;
assign palette[4] = 24'h530000;
assign palette[5] = 24'h710000;
assign palette[6] = 24'h8a0000;
assign palette[7] = 24'ha20000;
assign palette[8] = 24'hba0000;
assign palette[9] = 24'hdd0002;
assign palette[10] = 24'hff0000;
assign palette[11] = 24'hff3031;
assign palette[12] = 24'hff5f62;
assign palette[13] = 24'hff8d8b;
assign palette[14] = 24'hffcccb;
assign palette[15] = 24'hffffff;
```

```
reg [3:0] store[0:4095];

initial
begin
    $readmemh("sprite.mem", store);
end

always @(posedge clk)
   data_out <= palette[store[addr]];
endmodule
```

DISPLAY SPRITE

```
wire [23:0] sprite_rgb;

reg [10:0] pos_x = 100;
reg [10:0] pos_y = 100;

wire [10:0] sprite_x;
wire [10:0] sprite_y;

assign sprite_x = h_pos - pos_x;
assign sprite_y = v_pos - pos_y;

sprite_rom sprite(
     .clk(pixclk),
     .addr({ sprite_y[5:0], sprite_x[5:0] }),
     .data_out(sprite_rgb)
);

assign rgb_data = (h_pos > pos_x && h_pos < pos_x + 64
&& v_pos > pos_y && v_pos < pos_y + 64) ? sprite_rgb :
24'hffffff;
```

MOVE SPRITE

```
localparam SCREEN_WIDTH = 320;
localparam SCREEN_HEIGHT = 240;
reg [10:0] sprite_vel_x = 1;
reg [10:0] sprite_vel_y = -1;
always @(posedge cnt[17])
begin
    if (pos_x == 0)
    begin
        sprite_vel_x <= 1;
    end
    if (pos_x + 64 == SCREEN_WIDTH)
    begin
        sprite_vel_x <= -1;
    end
    if (pos_y == 0)
    begin
        sprite_vel_y <= 1;
    end
    if (pos_y + 64 == SCREEN_HEIGHT)
    begin
        sprite_vel_y <= -1;
    end
    pos_x <= pos_x + sprite_vel_x;
    pos_y <= pos_y + sprite_vel_y;
end
```

EXERCISE

- Change speed of moving in example
- Make sprite react on commands

PONG

DISPLAY PADS AND NET

```
localparam SCREEN_WIDTH = 320;
localparam SCREEN_HEIGHT = 240;
localparam PADDLE_SIZE = 60;
localparam PADDLE_WIDTH = 10;
localparam NET_WIDTH = 2;

reg [8:0] paddle1_y = (SCREEN_HEIGHT-PADDLE_SIZE)/2;
reg [8:0] paddle2_y = (SCREEN_HEIGHT-PADDLE_SIZE)/2;

assign rgb_data =
(h_pos > 0 && h_pos < PADDLE_WIDTH && v_pos > paddle1_y && v_pos < paddle1_y +
PADDLE_SIZE) ? 24'hffffff :
(h_pos > (SCREEN_WIDTH-PADDLE_WIDTH) && h_pos < SCREEN_WIDTH && v_pos > paddle2_y
&& v_pos < paddle2_y + PADDLE_SIZE) ? 24'hffffff :
(h_pos > (SCREEN_WIDTH/2) - NET_WIDTH && h_pos < (SCREEN_WIDTH/2) + NET_WIDTH &&
v_pos[4]==0) ? 24'hffffff :
24'h000000;
```

MOVING PADS

```
reg [31:0] cnt;
always @(posedge clk)
begin
cnt <= cnt + 1;
end

always @(posedge cnt[16])
begin
if (B1==1'b1)
paddle1_y <= (paddle1_y > 0) ? paddle1_y - 1 : paddle1_y;
if (B3==1'b1)
paddle1_y <= (paddle1_y < (SCREEN_HEIGHT-PADDLE_SIZE)) ? paddle1_y +1 : paddle1_y;
if (B5==1'b1)
paddle2_y <= (paddle2_y > 0) ? paddle2_y - 1 : paddle2_y;
if (B4==1'b1)
paddle2_y <= (paddle2_y < (SCREEN_HEIGHT-PADDLE_SIZE)) ? paddle2_y +1 : paddle2_y;
end
```

EXERCISE

- Change color of elements on screen
- Add obstacles on field
- Try expanding "The Wall" game code

CPU

INSTRUCTION SET (1/2)

IR			2nd	Instruction	Info
0000	dst	src		MOV dst, src	
0001	00	reg		ADD reg	r0 = r0 + reg
0001	01	reg		SUB reg	r0 = r0 - reg
0001	10	reg		ADC reg	r0 = r0 + reg + C
0001	11	reg		SBC reg	r0 = r0 - reg - C
0010	00	reg		AND reg	r0 = r0 and reg
0010	01	reg		OR reg	r0 = r0 or reg
0010	10	reg		NOT reg	r0 = not reg
0010	11	reg		XOR reg	r0 = r0 xor reg
0011	00	reg		INC reg	reg = reg + 1
0011	01	reg		DEC reg	reg = reg - 1
0011	10	reg		CMP reg	flags of r0 - reg
0011	11	reg		TST reg	flags of r0 and reg
0100	00	00		SHL	
0100	00	01		SHR	
0100	00	10		SAL	
0100	00	11		SAR	
0100	01	00		ROL	
0100	01	01		ROR	
0100	01	10		RCL	rotate with carry
0100	01	11		RCR	rotate with carry
0100	10	reg		PUSH reg	
0100	11	reg		POP reg	
0101	dst	src		LOAD dst, [src]	
0110	dst	src		STORE [dst], src	
0111	00	reg		MOV CS, reg	
0111	01	reg		MOV DS, reg	
0111	10	00		PUSH CS	
0111	10	01		PUSH DS	
0111	10	10		???	
0111	10	11		???	
0111	11	00		???	
0111	11	01		???	
0111	11	10		RET	
0111	11	11		HLT	

INSTRUCTION SET (2/2)

1000	00	00	val	JMP val	(unconditionally jump)
1000	00	01	val	JC val	(carry=1 jump)
1000	00	10	val	JNC val	(carry=0 jump)
1000	00	11	val	JM val	(sign=1 jump)
1000	01	00	val	JP val	(sign=0 jump)
1000	01	01	val	JZ val	(zero=1 jump)
1000	01	10	val	JNZ val	(zero=0 jump)
1000	01	11	val	???	
1000	10	00	val	JR val	(unconditionally jump)
1000	10	01	val	JRC val	(carry=1 jump)
1000	10	10	val	JRNC val	(carry=0 jump)
1000	10	11	val	JRM val	(sign=1 jump)
1000	11	00	val	JRP val	(sign=0 jump)
1000	11	01	val	JRZ val	(zero=1 jump)
1000	11	10	val	JRNZ val	(zero=0 jump)
1000	11	11	val	???	

1001	high		low	JUMP addr	
1010	high		low	CALL addr	
1011	high		low	MOV SP,addr	
1100	xx	reg	val	IN reg,[val]	
1101	xx	reg	val	OUT [val],reg	
1110	xx	00	val	MOV CS,val	
1110	xx	01	val	MOV DS,val	
1110	xx	10	val	???	
1110	xx	11	val	???	
1111	00	reg	val	MOV reg, val	
1111	01	reg	val	LOAD reg, [val]	
1111	10	reg	val	STORE [val], reg	
1111	11	xx		???	

ALU

```
ALU_OP_ADD :
    tmp = A + B;
ALU_OP_SUB :
    tmp = A - B;
ALU_OP_ADC :
    tmp = A + B + { 7'b0000000, CF };
ALU_OP_SBC :
    tmp = A - B - { 7'b0000000, CF };
ALU_OP_AND :
    tmp = {1'b0, A & B };
ALU_OP_OR :
    tmp = {1'b0, A | B };
ALU_OP_NOT :
    tmp = {1'b0, ~B };
ALU_OP_XOR :
    tmp = {1'b0, A ^ B};
```

```
ALU_OP_SHL :
    tmp = { A[7], A[6:0], 1'b0};
ALU_OP_SHR :
    tmp = { A[0], 1'b0, A[7:1]};
ALU_OP_SAL :
    // Same as SHL
    tmp = { A[7], A[6:0], 1'b0};
ALU_OP_SAR :
    tmp = { A[0], A[7], A[7:1]};
ALU_OP_ROL :
    tmp = { A[7], A[6:0], A[7]};
ALU_OP_ROR :
    tmp = { A[0], A[0], A[7:1]};
ALU_OP_RCL :
    tmp = { A[7], A[6:0], CF};
ALU_OP_RCR :
    tmp = { A[0], CF, A[7:1]};
```

EXERCISE

- Change executed binary
- Add new instruction of your choice

COMPUTER

MAPPING MEMORY READS

```
casex ({boot,sysctl[6],addr[15:8]})
   // Turn-key BOOT
    {2'b10,8'bxxxxxxxx}: begin idata = boot_out; rd_boot = rd; end // any address
    // MEM MAP
    {2'b00,8'b000xxxxx}: begin idata = rammain_out; rd_rammain = rd; end // 0x0000-0x1fff
    {2'b00,8'b11111011}: begin idata = ram_out; rd_ram = rd; end // 0xfb00-0xfbff
    {2'b00,8'b11111101}: begin idata = rom_out; rd_rom = rd; end // 0xfd00-0xfdff
    // I/O MAP - addr[15:8] == addr[7:0] for this section
    {2'b01,8'b000x000x}: begin idata = sio_out; rd_sio = rd; end // 0x00-0x01 0x10-0x11
endcase
```

MAPPING MEMORY WRITES

```
casex ({sysctl[4],addr[15:8]})
    // MEM MAP
    {1'b0,8'b000xxxxx}: wr_rammain = ~wr_n; // 0x0000-0x1fff
    {1'b0,8'b11111011}: wr_ram = ~wr_n; // 0xfb00-0xfbff
                                        // 0xfd00-0xfdff read-only
    // I/O MAP - addr[15:8] == addr[7:0] for this section
    {1'b1,8'b000x000x}: wr_sio = ~wr_n; // 0x00-0x01 0x10-0x11
endcase
```

MAPPING DEVICES

```
I8080 cpu
(.clk(clk),.ce(ce),.reset(reset),.intr(intr),.idata(idata),.addr(addr),.sync(sync),.rd(rd),.wr_n(wr_n),.i
nta_n(inta_n),.odata(odata),.inte_o(inte_o));

jmp_boot boot_ff(.clk(clk),.reset(reset),.rd(rd_boot),.data_out(boot_out),.valid(boot));

rom_memory #(.ADDR_WIDTH(8),.FILENAME("turnmon.bin.mem"))
    rom(.clk(clk),.addr(addr[7:0]),.rd(rd_rom),.data_out(rom_out));

ram_memory #(.ADDR_WIDTH(8))
    stack(.clk(clk),.addr(addr[7:0]),.data_in(odata),.rd(rd_ram),.we(wr_ram),.data_out(ram_out));

ram_memory #(.ADDR_WIDTH(13),.FILENAME("basic4k32.bin.mem"))
    mainmem(.clk(clk),.addr(addr[12:0]),.data_in(odata),.rd(rd_rammain),.we(wr_rammain),.data_out(rammai
n_out));

mc6850
    sio(.clk(clk),.reset(reset),.addr(addr[0]),.data_in(odata),.rd(rd_sio),.we(wr_sio),.data_out(sio_out
),.ce(ce),.rx(rx),.tx(tx));
```

EXERCISE

- Write your first Altair code in BASIC
- Display text on LCD

RISC-V

MEMORY MAPPED I/O

```
#define reg_uart_clkdiv (*(volatile uint32_t*)0x02000004)
#define reg_uart_data (*(volatile uint32_t*)0x02000008)

#define reg_io ((volatile uint32_t*)0x03000000)
#define reg_text ((volatile uint32_t*)0x04000000)
#define reg_attr ((volatile uint32_t*)0x05000000)
#define reg_tone ((volatile uint32_t*)0x06000000)
```

SEND DATA OVER UART

```
void putchar(char c)
{
    if (c == '\n')
        putchar('\r');
    if (c != '\r')
        lcd_putch(c);
    reg_uart_data = c;
}

void print(const char *p)
{
    while (*p)
        putchar(*(p++));
}
```

EXERCISE

- Display your own text by programming in C
- Make terminal emulation in C
- Write a small Q&A game

MICROPYTHON

CLASSIC PYTHON CODING

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ‘)
        a, b = b, a+b
        print()

fib(1000)
```

ACCESSING LEDS

```
import pyb
def blinky():
    for i in range(30):
        pyb.LED(i % 3 + 1).on()
        pyb.delay(100)
        pyb.LED(i % 3 + 1).off()
```

ACCESSING BUTTONS

```
import pyb

but0 = pyb.Switch(1)
but1 = pyb.Switch(2)

but0()
```

LET'S PLAY SOUND

```
import pyb

pyb.tone(440, 100)
pyb.delay(100)
pyb.tone(440, 100)
```

DISPLAY ON LCD

```
import pyb

lcd = pyb.LCD()
lcd.clear()
lcd.move(5,5)
lcd.write("Test")
lcd.move(10,10)
lcd.write("Test")
```

EXERCISE

- Show me your Python skills ☺

DEMOSTRATION

MIKROELEKTRONIKA

- Click boards with FPGA
- Order any Click board with provided 20% discount code (valid one month from now)