



# Lab 4: Push Button and LED Control

Chun-Jen Tsai and Lan-Da Van  
Department of Computer Science  
National Chiao Tung University  
Taiwan, R.O.C.  
*Fall, 2019*



# Lab 4: Push Buttons and LED Control

Lab 4

- ◆ In this lab, you will use the FPGA development board “Arty” to implement a simple I/O control circuit
  - There are 4 push-buttons and 4 yellow LED lights on the board
  - You must design a synchronous circuit that reads each of the push-button inputs and display different light patterns
  
- ◆ The lab file submission deadline is on 10/28 by 6:00pm.

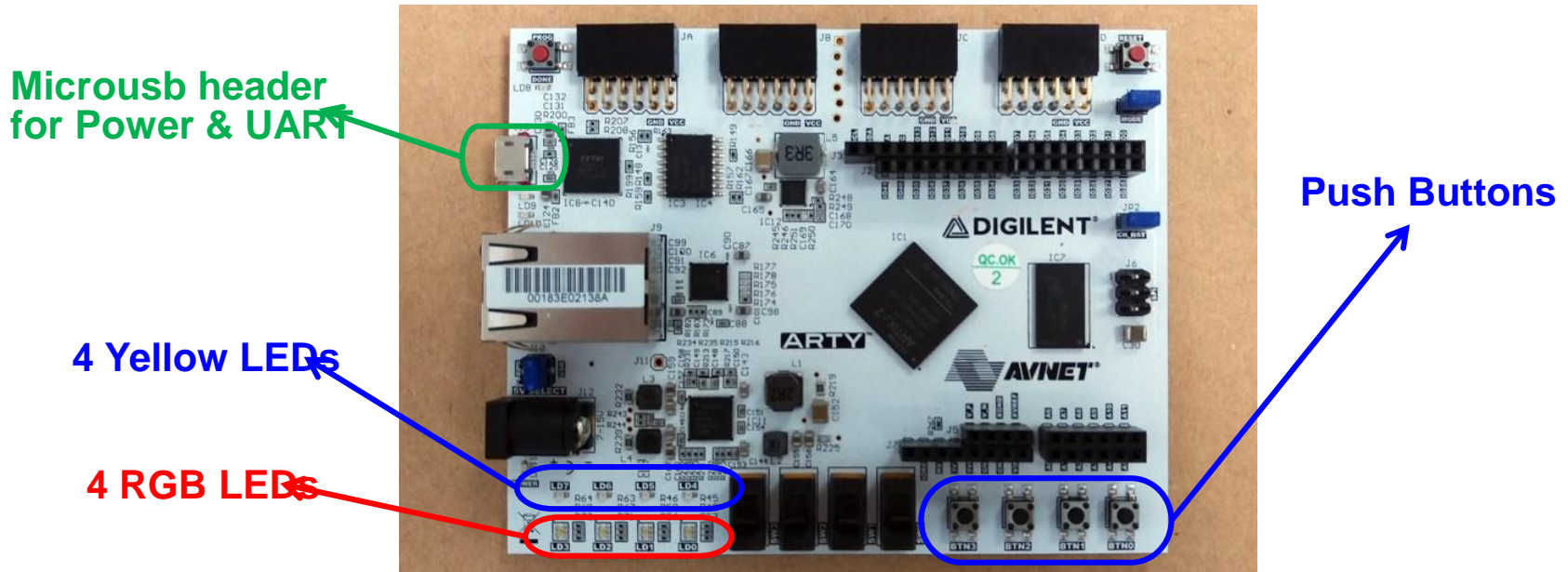




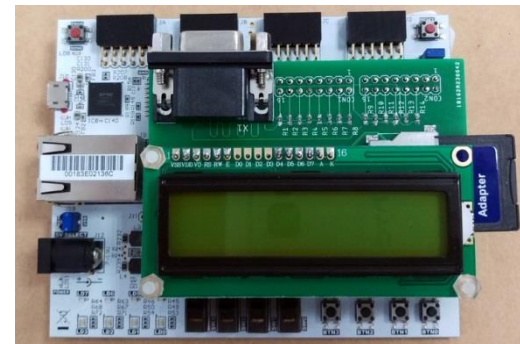
# Buttons and LEDs on the Arty Board

Lab 4

◆ The Arty FPGA development board:



◆ We have designed an I/O daughter board for Arty:

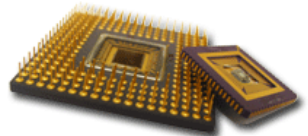




# The System Behavior of Lab 4

Lab 4

- ◆ Your circuit should have a 4-bit counter register
  - The counter value is set to zero upon reset
  - The counter value is a signed value in 2's complement format
  - The 4 LEDs display the 4 counter bits at all time
- ◆ Push-buttons #0 and #1 are used to decrease/increase the counter value:
  - Push the BTN1/BTN0 increases/decreases the counter by 1
  - If the counter value becomes greater than 7, it is truncated to 7; if the value is smaller than -8, it is set to -8
- ◆ Push-buttons #2 and #3 are used to control the brightness of the LEDs
  - BTN3 makes the LED brighter and BTN2 makes it darker

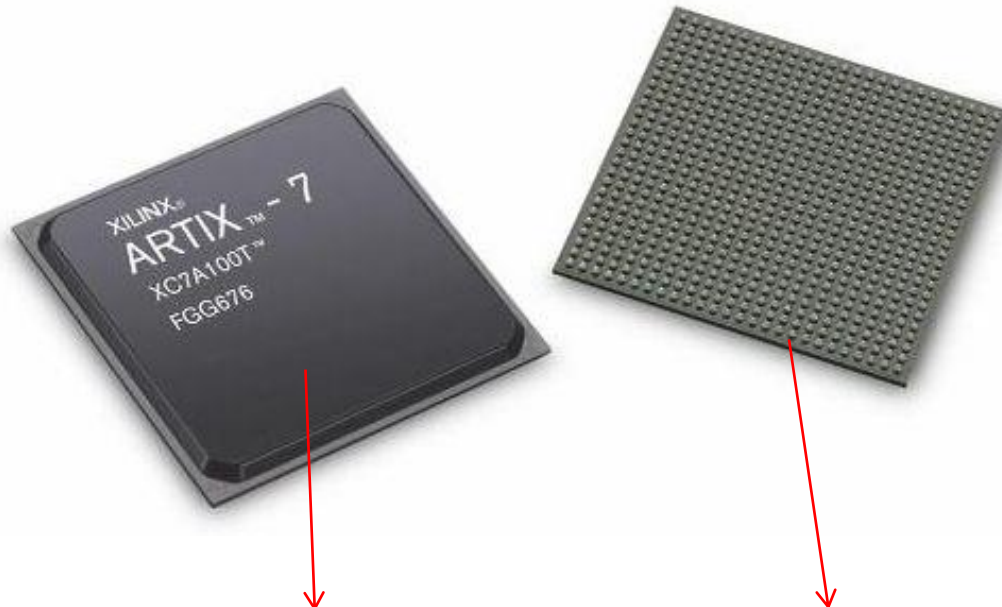




# User I/O Pins of an FPGA IC

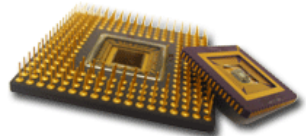
Lab 4

- ◆ There are many “FPGA” pins that are used as user I/O pins: each pin connects to an I/O device such as the push-buttons or the LEDs:



Top-view of an FPGA IC  
(shows IC markings)

Bottom-view of an FPGA IC  
(shows IC pins)

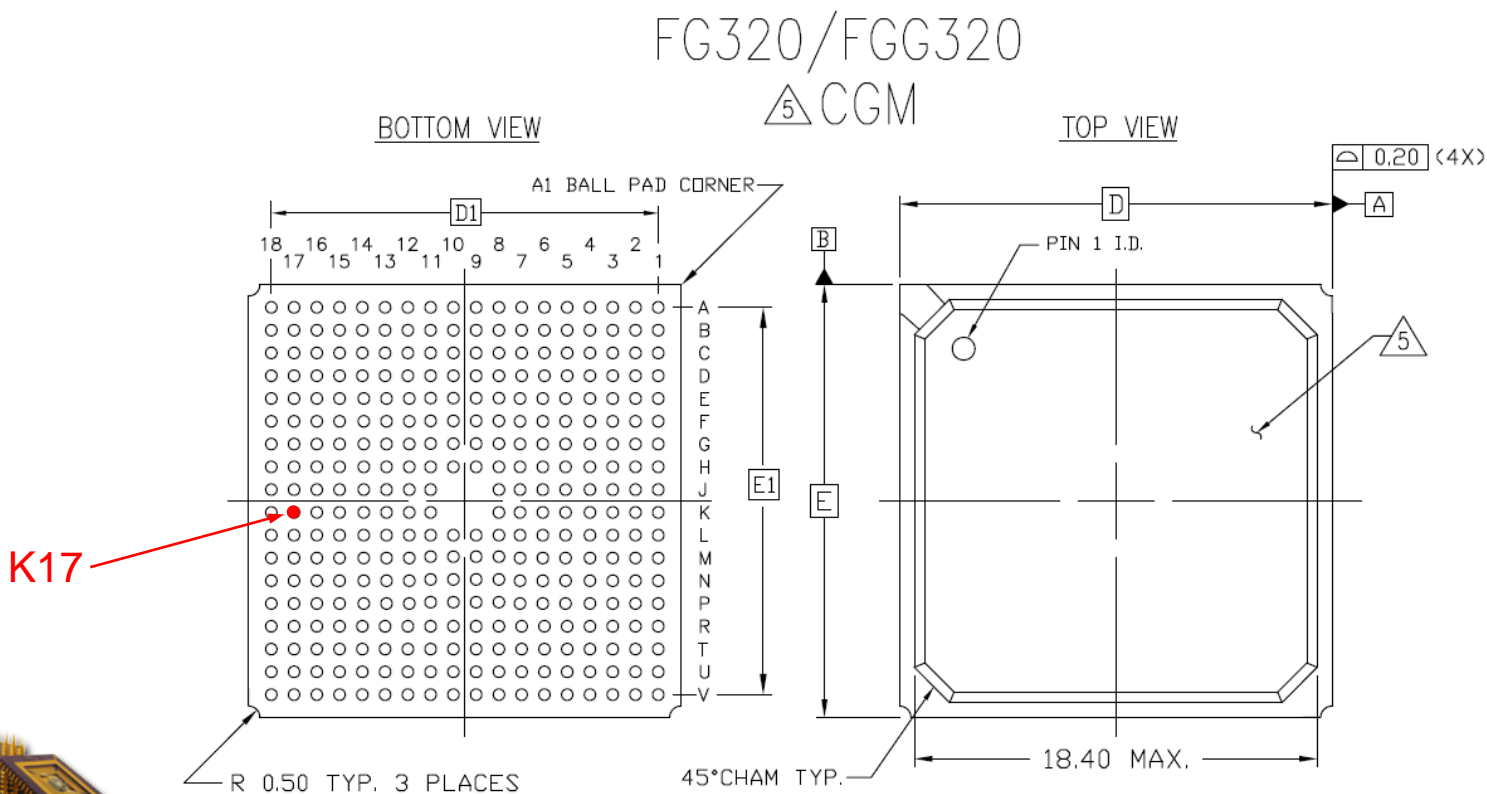




# FPGA Pin Coordinates

Lab 4

- Each pin at the bottom of the FPGA has a coordinate. For example, “K17” is the coordinate of the red pin of the Xilinx FPGA IC in the “FG320” package:





# Use the I/O Pin Signal in Verilog

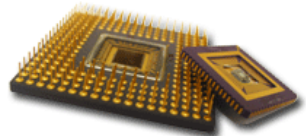
Lab 4

- ◆ To read/write the I/O pins, we must map the pin coordinates to Verilog signals in our code
  - A user constraint file, \*.xdc, is used to do the job
- ◆ A user constraint is a text command that specifies the physical property in an HDL code. For example, for the four push-buttons, their mapping to Verilog signals can be as follows:

```
set_property -dict { PACKAGE_PIN D9 IOSTANDARD LVCMOS33 } [get_ports { usr_btn[0] }];
set_property -dict { PACKAGE_PIN C9 IOSTANDARD LVCMOS33 } [get_ports { usr_btn[1] }];
set_property -dict { PACKAGE_PIN B9 IOSTANDARD LVCMOS33 } [get_ports { usr_btn[2] }];
set_property -dict { PACKAGE_PIN B8 IOSTANDARD LVCMOS33 } [get_ports { usr_btn[3] }];
```

IC pin coordinates      Signal type

Signal names to be used  
in your Verilog code!



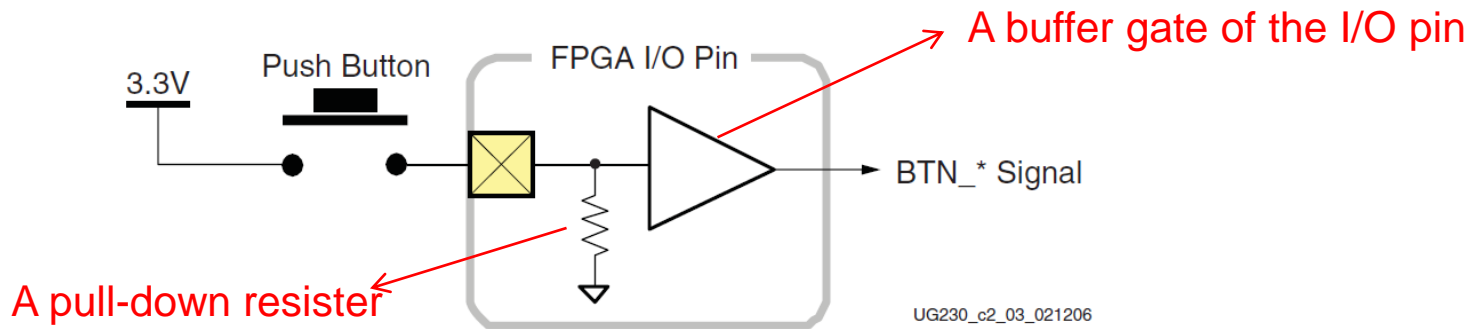




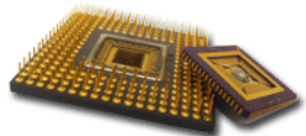
# How to Read the Input Push-Button

Lab 4

- ◆ The physical connection from an FPGA I/O pin to a push-button is as follows:



- ◆ Ideally, when a push-button is pushed (the circuit is closed), the FPGA pin that connects to the button becomes high voltage and the corresponding signal in Verilog reads “1”, otherwise it reads “0”.





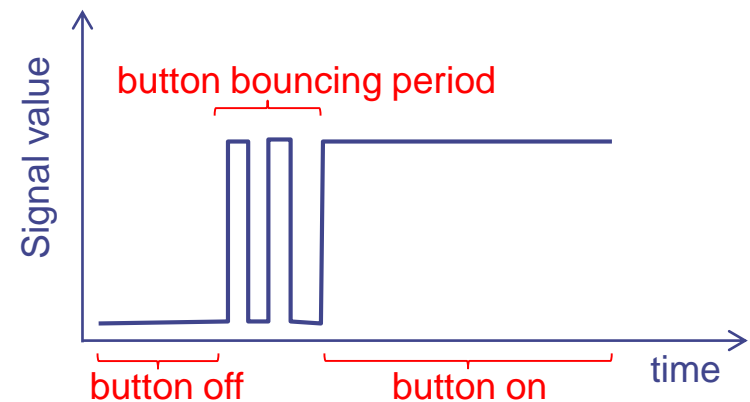
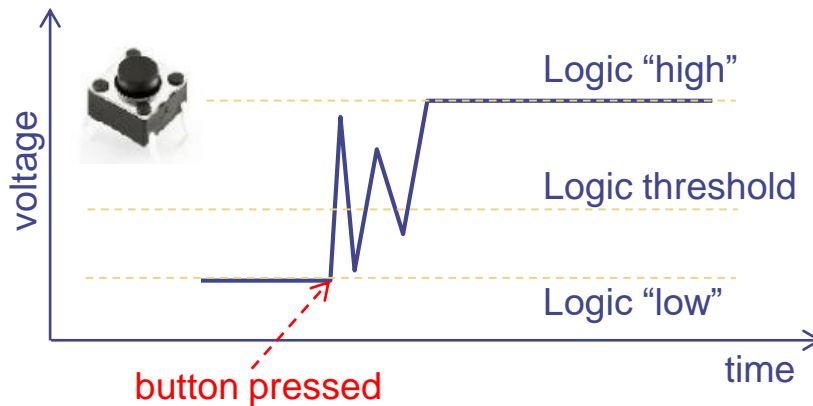


# The Bouncing Problem

Lab 4

- ◆ In reality, however, the signal value oscillates between 0 and 1 several times before it stabilizes. This is called the bouncing behavior of a hardware button.

**The physical voltage values** **The actual digital signal:**





# The De-bouncing Circuit

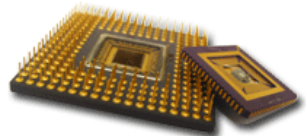
Lab 4

- ◆ To detect whether the button has been pressed, you cannot simply check the button signals:

```
always @(posedge clk, posedge reset)
  if (reset == 1)
    BTN0_is_pressed = 0;
  else
    BTN0_is_pressed = (usr_btn[0]) ? 1 : 0;
```



- This circuit will catch all the state changes during the bouncing period → a single button click will be treated as multiple clicks!
- ◆ You must find a way to average-out the noises of the push-button signal during the bouncing period
  - Hint: you can use a shift register to accumulate the input signal; or a timer to wait out the bouncing period





# Turn On/Off the LEDs

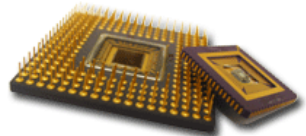
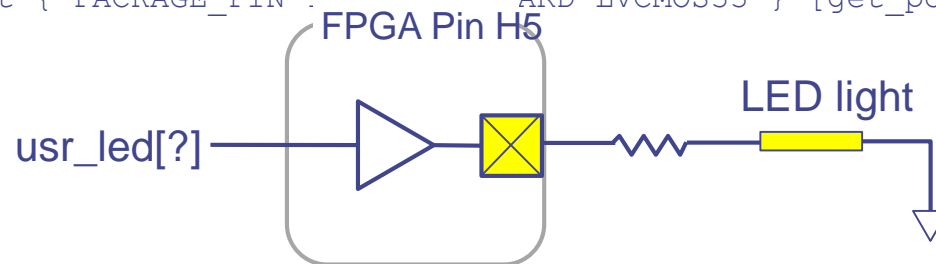
Lab 4

- ◆ The LEDs can be turned on/off by writing 1/0 to the corresponding Verilog signals, `reg [3:0]`

`usr_led`

- The LED constraint definitions:

```
set_property -dict { PACKAGE_PIN H5  IOSTANDARD LVCMOS33 } [get_ports { usr_led[0]
}];
set_property -dict { PACKAGE_PIN J5  IOSTANDARD LVCMOS33 } [get_ports { usr_led[1]
}];
set_property -dict { PACKAGE_PIN T9  IOSTANDARD LVCMOS33 } [get_ports { usr_led[2]
}];
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { usr_led[3]
}];
```





# Clock and Reset Pins

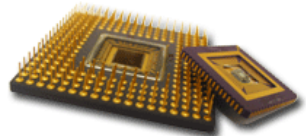
Lab 4

- ◆ For synchronous design, you need a clock signal for your circuit
  - The clock signal usually comes from an on-board oscillator
  - There is an FPGA pin that connects to the oscillator

```
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports { clk }];
```

- ◆ For the Arty board, the reset pin is the red push button defined as follows:

```
set_property -dict {PACKAGE_PIN C2 IOSTANDARD LVCMOS33} [get_ports { reset_n }];
```





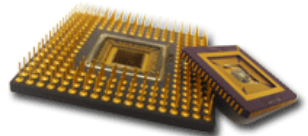
# Sample Project of Lab 4

Lab 4

- ◆ A sample project, lab3.zip, is available on E3
  - The constraint file for Arty is provided to you in this project
- ◆ The project has a circuit that lights up LED 0 ~ 3 when you press BTN 0 ~ 3, respectively

```
module lab3(  
    input  clk,                // System clock at 100 MHz  
    input  reset_n,           // System negative reset signal  
    input  [3:0] usr_btn,     // Four user pushbuttons  
    output [3:0] usr_led      // Four yellow LEDs  
);  
  
assign usr_led = usr_btn;  
endmodule
```

- ◆ There is no de-bouncing circuit for the button inputs so you have to add this part by yourself





# Generating the Programming File

Lab 4

- ◆ To test the design on Arty, you must generate the programming file “lab3.bit” for the FPGA:

The screenshot shows the Vivado 2017.2.1 IDE interface. The top menu bar includes File, Edit, Flow, Tools, Window, Layout, View, and Help. The top toolbar contains icons for file operations and simulation. The left sidebar shows the Project Manager with sections for Settings, Add Sources, Language Templates, IP Catalog, IP INTEGRATOR, SIMULATION, RTL ANALYSIS, SYNTHESIS, IMPLEMENTATION, and PROGRAM AND DEBUG. The main workspace is divided into several panes: Sources (showing Design Sources, Constraints, and Hierarchy), Source File Property (for arty\_base.xdc), Project Summary (showing the module lab3.v), and Design Runs (showing synthesis and implementation results). A red dashed arrow points from the 'Generate Bitstream' button in the PROGRAM AND DEBUG section to the 'write\_bitstream Complete' status in the top right corner. Another red dashed arrow points from the 'Generate Bitstream' button to the 'lab3.v' file in the Project Summary pane.

Click this to generate the lab3.bit file

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
✓ synth_1	constrs_1	synth_design Complete!								0	0
✓ impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA	NA	0.065	0	0	0



# Downloading Your Circuit to the Board

Lab 4

- ❖ To download your programming file into the FPGA, you must use the “Hardware Manager” :

The screenshot shows the Vivado 2017.2.1 IDE interface. The **HARDWARE MANAGER - unconnected** tab is active, displaying a message: "No hardware target is open. Open target". Below this, the **Hardware** section is empty. The **Source File Properties** section shows the **General** tab for **arty\_base.xdc**. The **PROGRAM AND DEBUG** tab is also visible, with the **Open Hardware Manager** section expanded. The **Open Target** button is highlighted with a red dashed circle and a red arrow. The **Tcl Console** at the bottom shows a list of instructions for connecting the hardware target.

lab3 - [D:\xilinx\vivado\arty\2017\lab3\lab3\lab3.xpr] - Vivado 2017.2.1

File Edit Flow Tools Window Layout View Help Quick Access

write\_bitstream Complete

Default Layout

Flow Navigator

- PROJECT MANAGER
- IP INTEGRATOR
- SIMULATION
- RTL ANALYSIS
- SYNTHESIS
- IMPLEMENTATION
- PROGRAM AND DEBUG
  - Generate Bitstream
  - Open Hardware Manager
    - Open Target
    - Program Device
    - Add Configuration Memory Device

HARDWARE MANAGER - unconnected

No hardware target is open. Open target

Hardware

No content

Source File Properties

arty\_base.xdc

General Properties

lab3.v

D:\xilinx\vivado\arty\2017\lab3\lab3\srcs/sources\_1/new\lab3.v

```
22
23 module lab3(
24     input  clk,           // System clock at 100
25     input  reset_n,       // System reset signal
26     input  [3:0] usr_btn, // Four user pushbutton
27     output [3:0] usr_led  // Four yellow LEDs
28 )
```

Tcl Console

1. Make sure the clock connected to the debug hub (dbg\_hub) core is a free running clock and i/
2. Make sure the BSCAN\_SWITCH\_USER\_MASK device property in Vivado Hardware Manager reflects the
For more details on setting the scan chain property, consult the Vivado Debug and Programming U
close\_hw
open\_hw

Type a Tcl command here

打开硬件程序和调试管理

Plug the Arty USB cable into a PC, click “Open Target” then “Auto Connect”

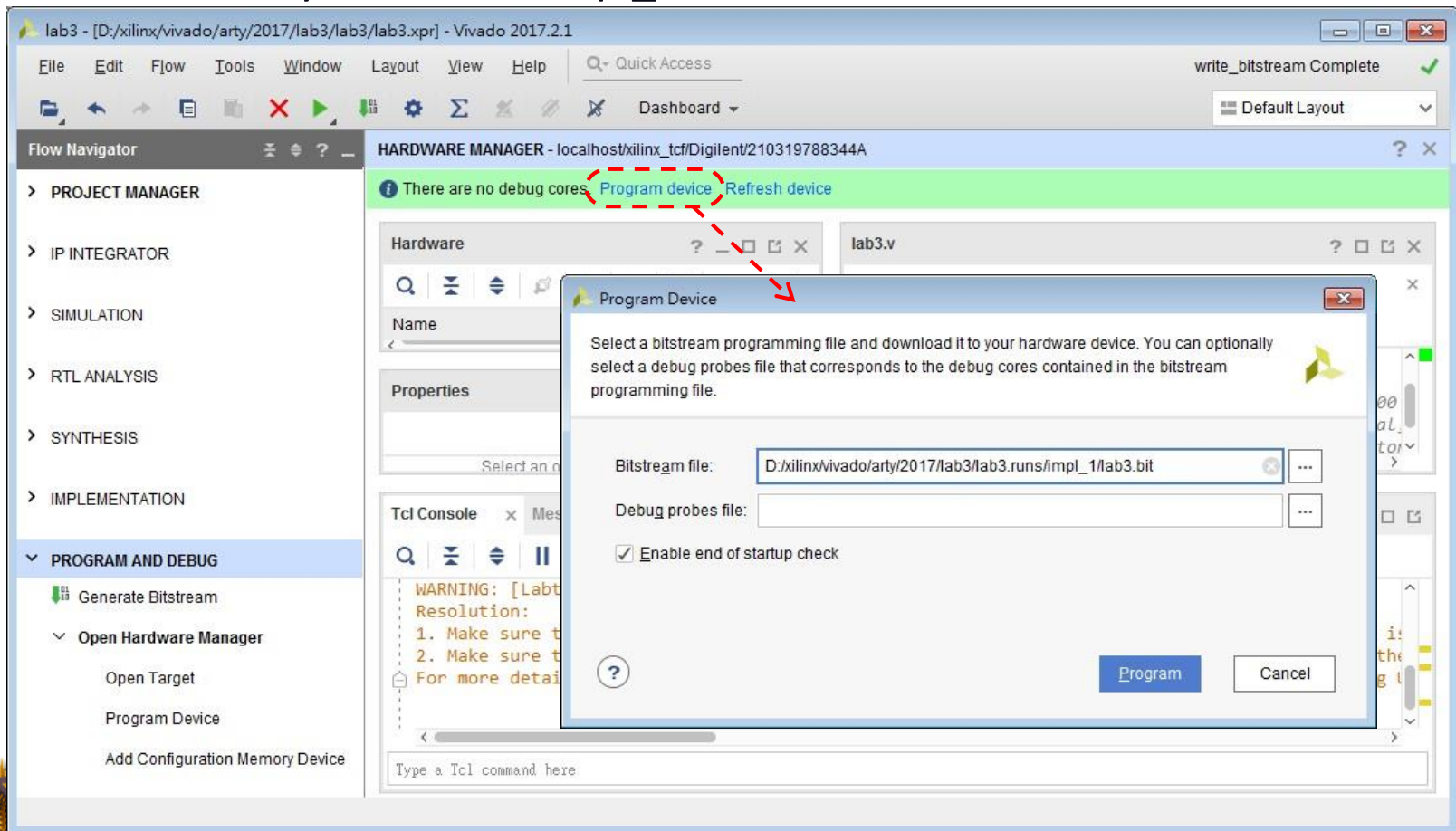




# Program the FPGA

Lab 4

- ◆ Hit “Program device” then browse to the \*.bit file:
  - The bit file is under <project directory>/lab3.run/impl\_1/lab3.bit

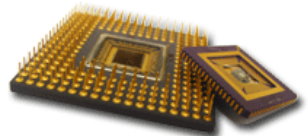
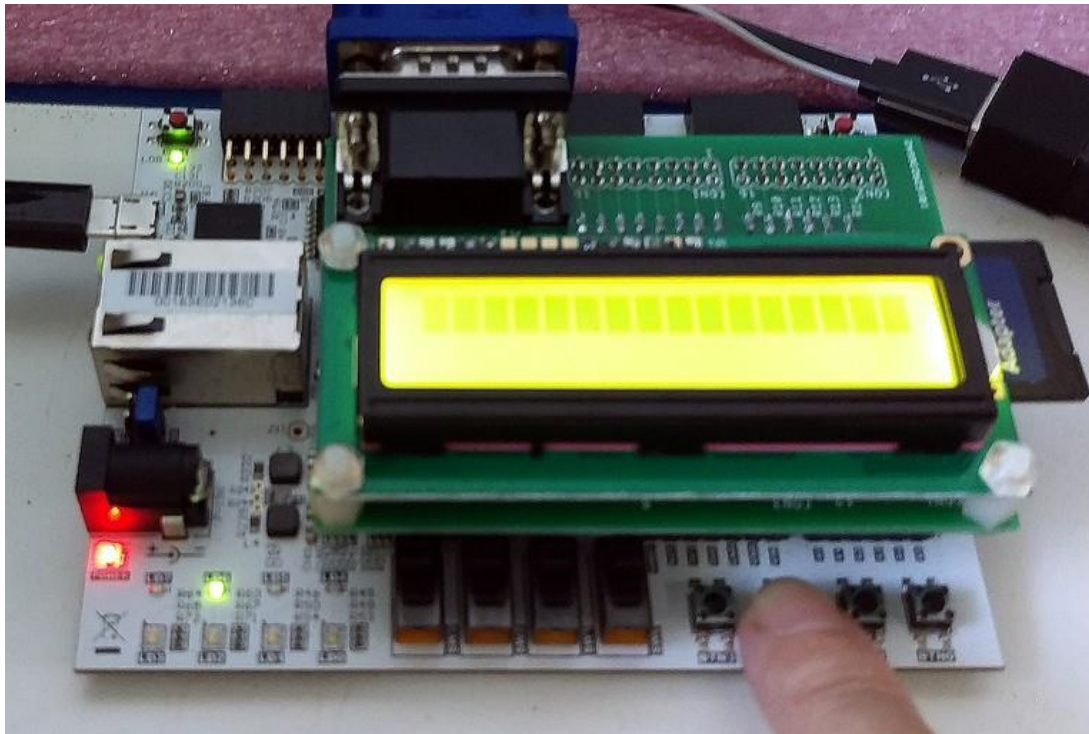




# Test Your Design

Lab 4

- ◆ You can now test your circuit by clicking the buttons on the Arty board and see how the LEDs lights up!

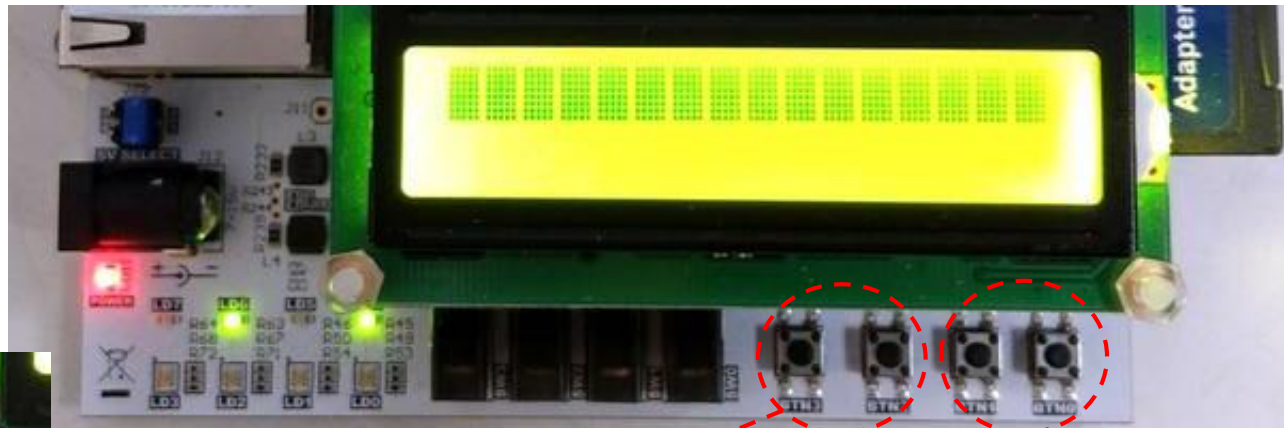




# What You Need to Do for Lab 4

Lab 4

- ◆ Design a circuit to display the value of a 4-bit signed counter on the LEDs with different brightness
  - BTN1/BTN0 increases/decreases the counter value
  - BTN2/BTN1 increases/decreases the brightness of the LEDs (all four LEDs should have the same brightness)



Increase/decrease the brightness of LEDs with five different brightness levels

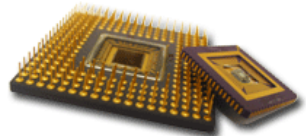
Increase/decrease the 4-bit counter value between  $-8 \sim 7$



# Control of the LED Brightness

Lab 4

- ◆ The LED devices on the Arty board can only be fully lit (full power) or turned off (zero power), you can not set them to show different levels of brightness
- ◆ To trick your eyes to see different levels of brightness, you can send a PWM signal to its power input
- ◆ A PWM input to the LED turns it on-an-off quickly
  - The persistence of human visions will not see flickering but only different levels of brightness, as long as your PWM frequency is high enough

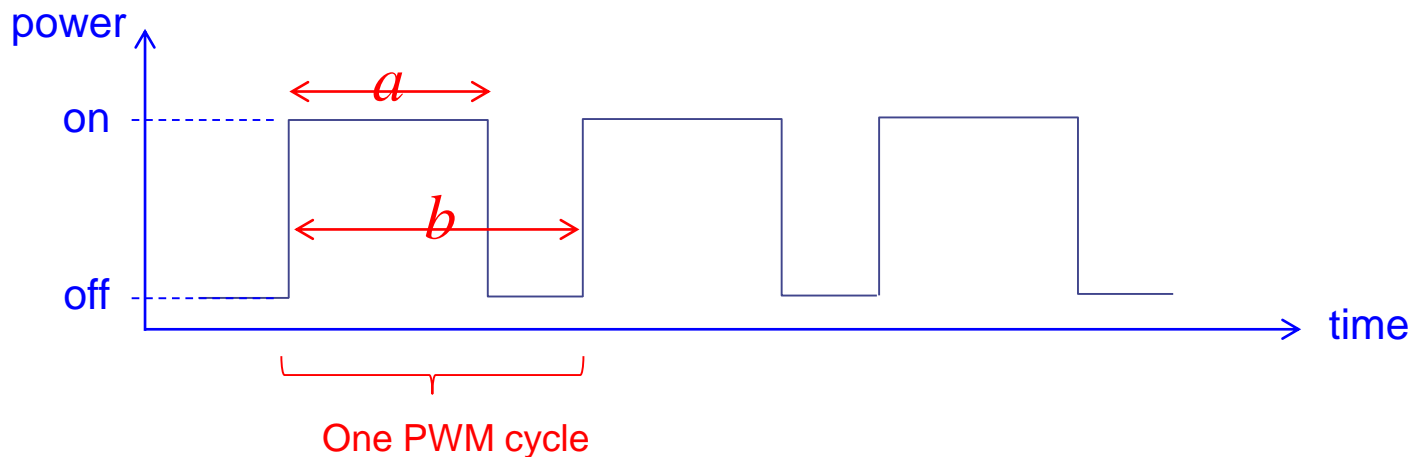




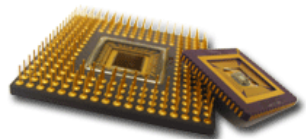
# A PWM Signal

Lab 4

- ◆ A PWM signal is simply a square wave signal:



- ◆ Duty-cycle: the percentage of one cycle of PWM that is in “on” state (i.e.,  $(a/b) \times 100\%$  in the figure)
  - 50% duty-cycle means the signal is “on” half of the time

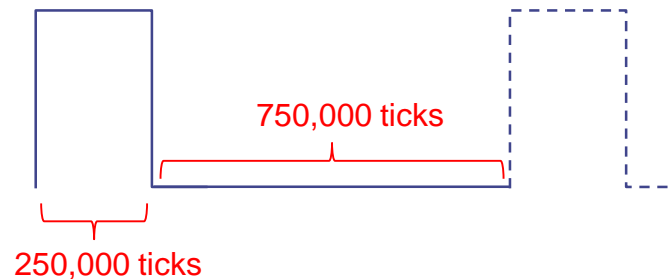




# Generation of a PWM Signal

Lab 4

- ◆ The system clock of our boards is 100MHz
  - Each second has 100,000,000 clock ticks
- ◆ To generate a 100 Hz PWM signal, the full cycle period would be equal to 1,000,000 clock ticks
  - The clock ticks for a 25% duty cycle PWM signal @ 100Hz would be 250,000 clock ticks for “on” period and 750,000 clock ticks for “off” period



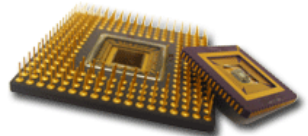
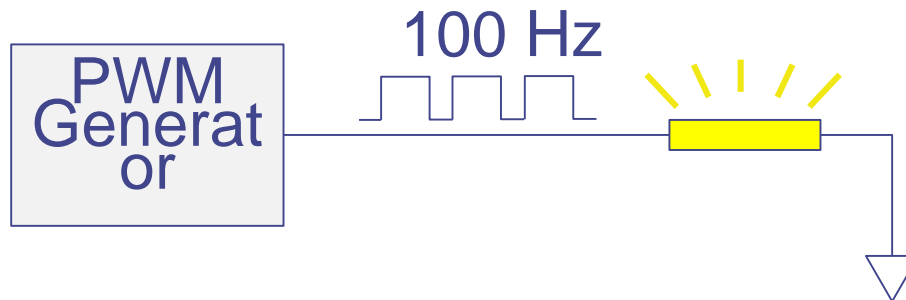




# PWM Control of Brightness

Lab 4

- ◆ Persistence of visions make most people do not see flickering when the LED is switching faster than 60 Hz
- ◆ We can use a PWM signal higher than 60Hz to control the brightness of an LED
- ◆ The PWM duty cycle determines the brightness







# Brightness Control for Lab 4

Lab 4

- ◆ In Lab3, you must design a PWM signal generator circuit
  - The PWM signal must have a frequency of 100 Hz and five different duty cycles: 5%, 25%, 50%, 75%, and 100%
  - If LED #n should be on, the PWM signal will be assign to `usr_led[n]`
  - If LED #n should be off, 0 will be assigned to `usr_led[n]`
  - BTN3 increases the current duty cycle, and BTN2 decreases the current duty cycle

