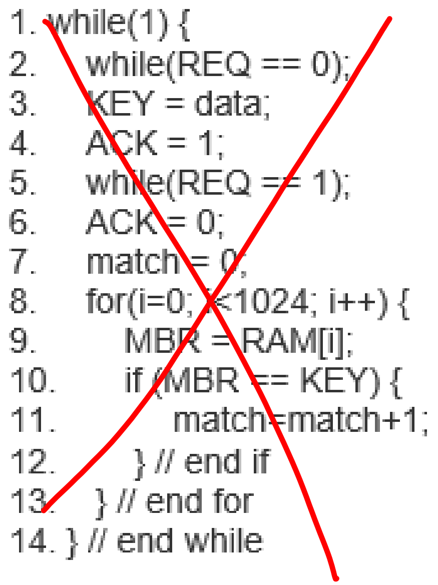
L18 now L19

# ECE 383 - Embedded Computer Systems II Lecture 18 - Soft Core (MicroBlaze) + Custom IP

UNITED STATES AIR FORCE ACADEMY

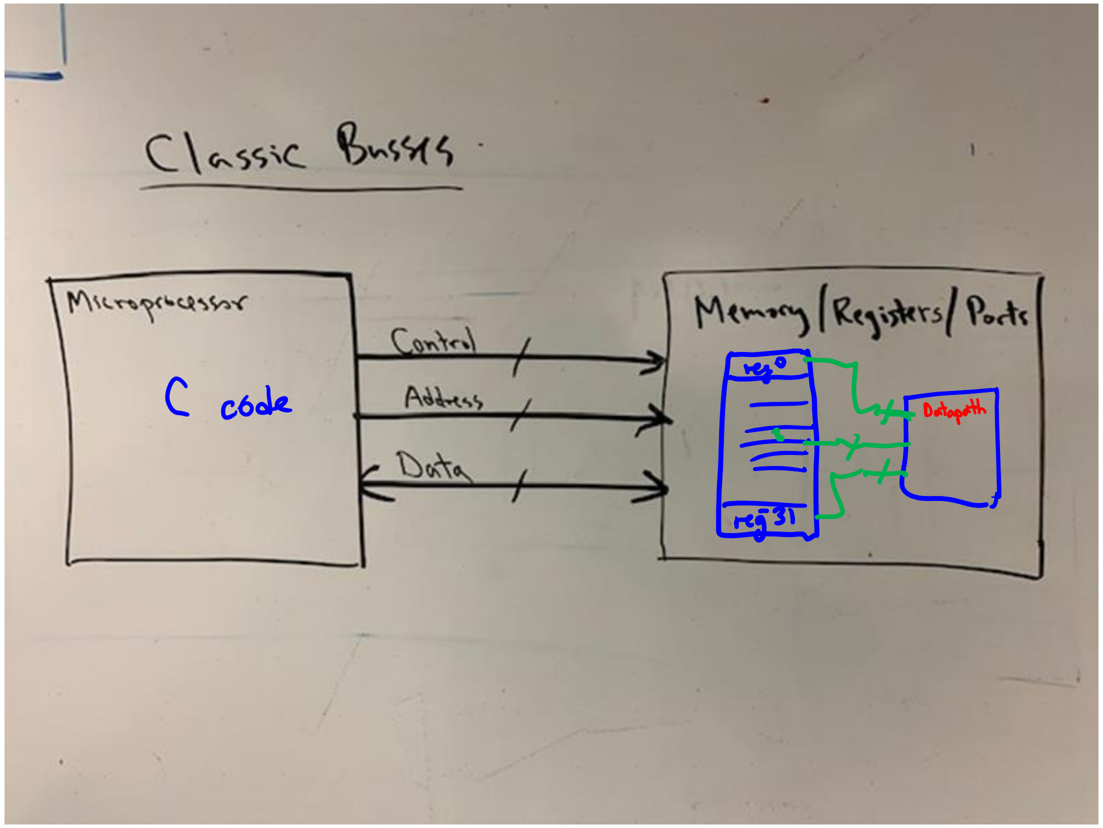
# Similarity of Lab2 FSM with HW8b?



```
while(1) {
    while(REQ == 0);
    KEY = data;
    ACK = 1;
    while(REQ == 1);
    ACK = 0;
    match = 0;
    for(i=0; i<1024; i++) {
        MBR = RAM[i];
        if (MBR == KEY) {
            match=match+1;
        } // end if
    } // end for
} // end while
```

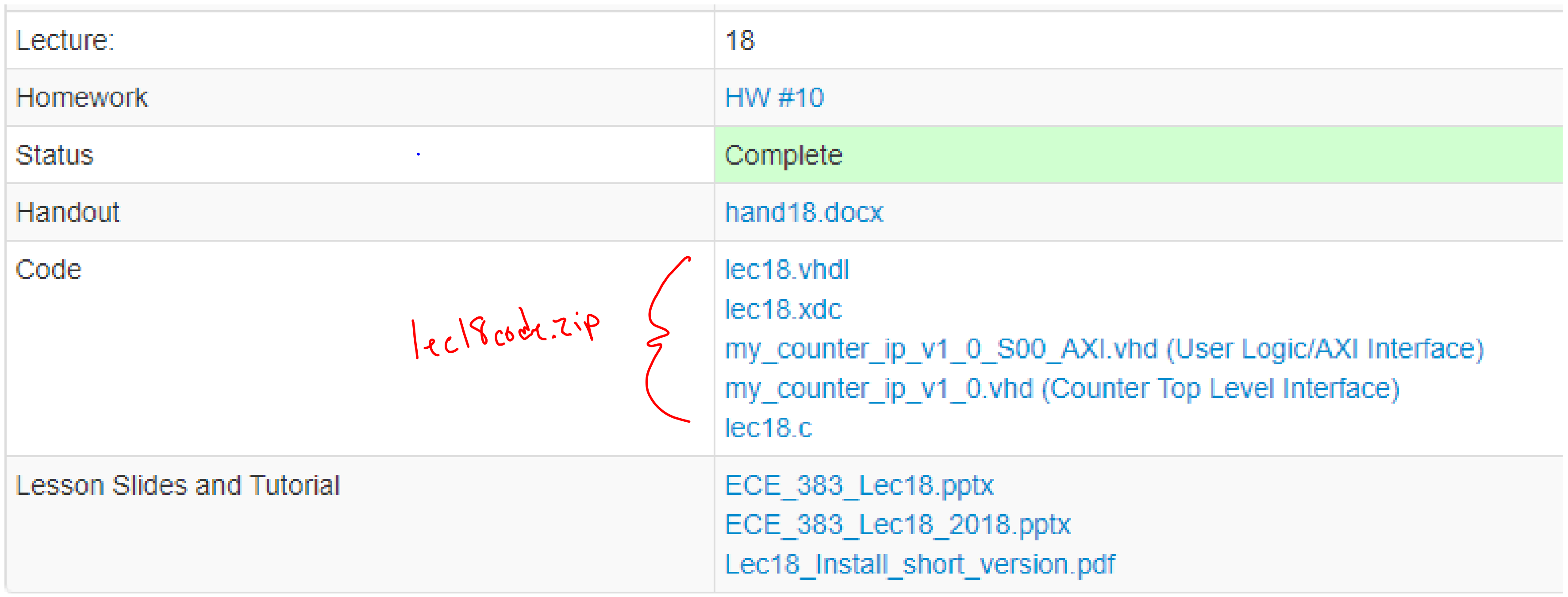
# Lesson Outline

- ~~Time Logs!~~
- MicroBlaze + Custom IP



Write Cycle?
Read Cycle?

UNITED STATES
AIR FORCE
ACADEMY

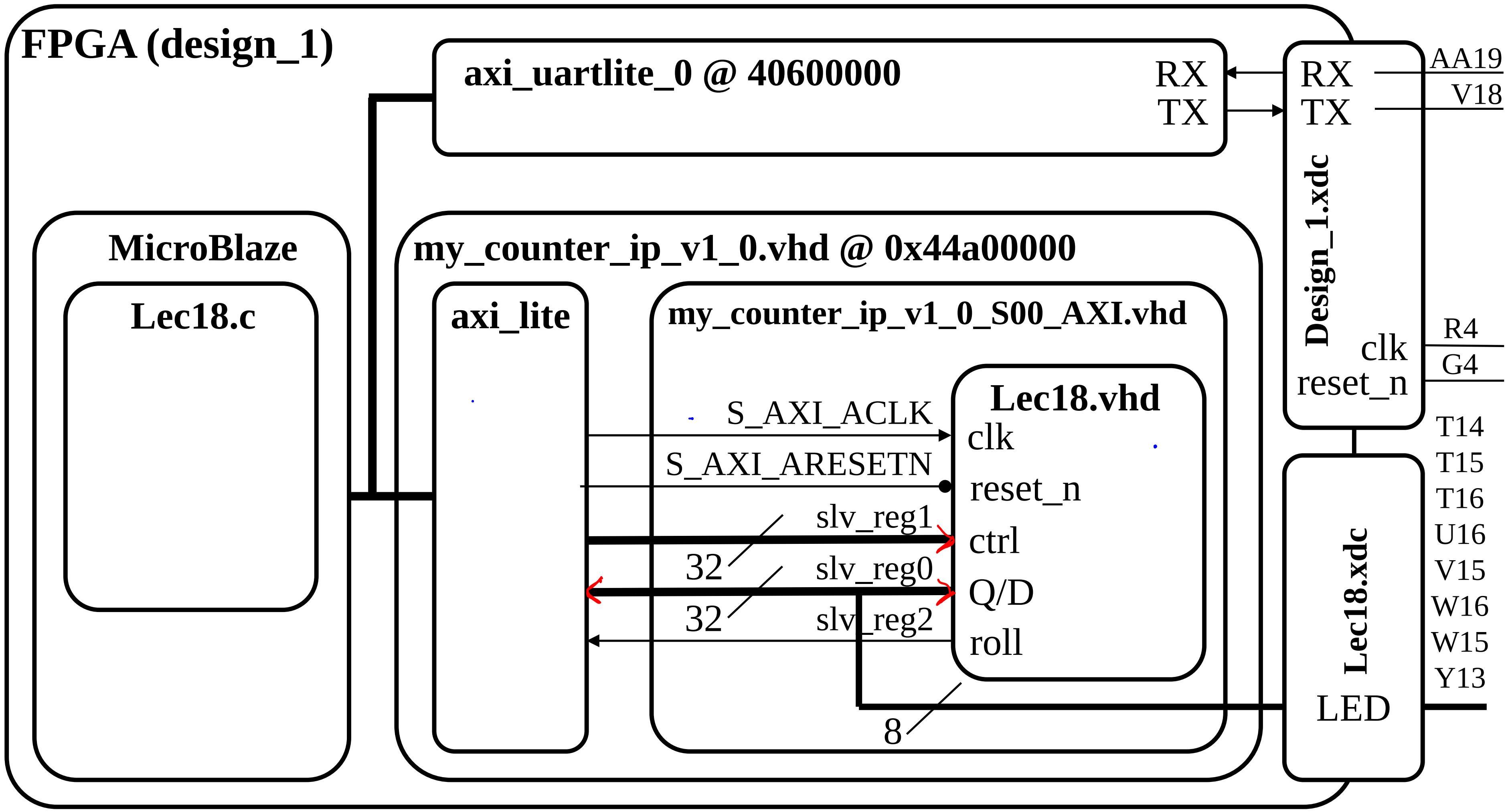


| | |
|---|---|
| Lecture: | 18 |
| Homework | HW #10 |
| Status | Complete |
| Handout | hand18.docx |
| Code<br>lec18code.zip | lec18.vhdl<br>lec18.xdc<br>my_counter_ip_v1_0_S00_AXI.vhd (User Logic/AXI Interface)<br>my_counter_ip_v1_0.vhd (Counter Top Level Interface)<br>lec18.c |
| Lesson Slides and Tutorial | ECE_383_Lec18.pptx<br>ECE_383_Lec18_2018.pptx<br>Lec18_Install_short_version.pdf |

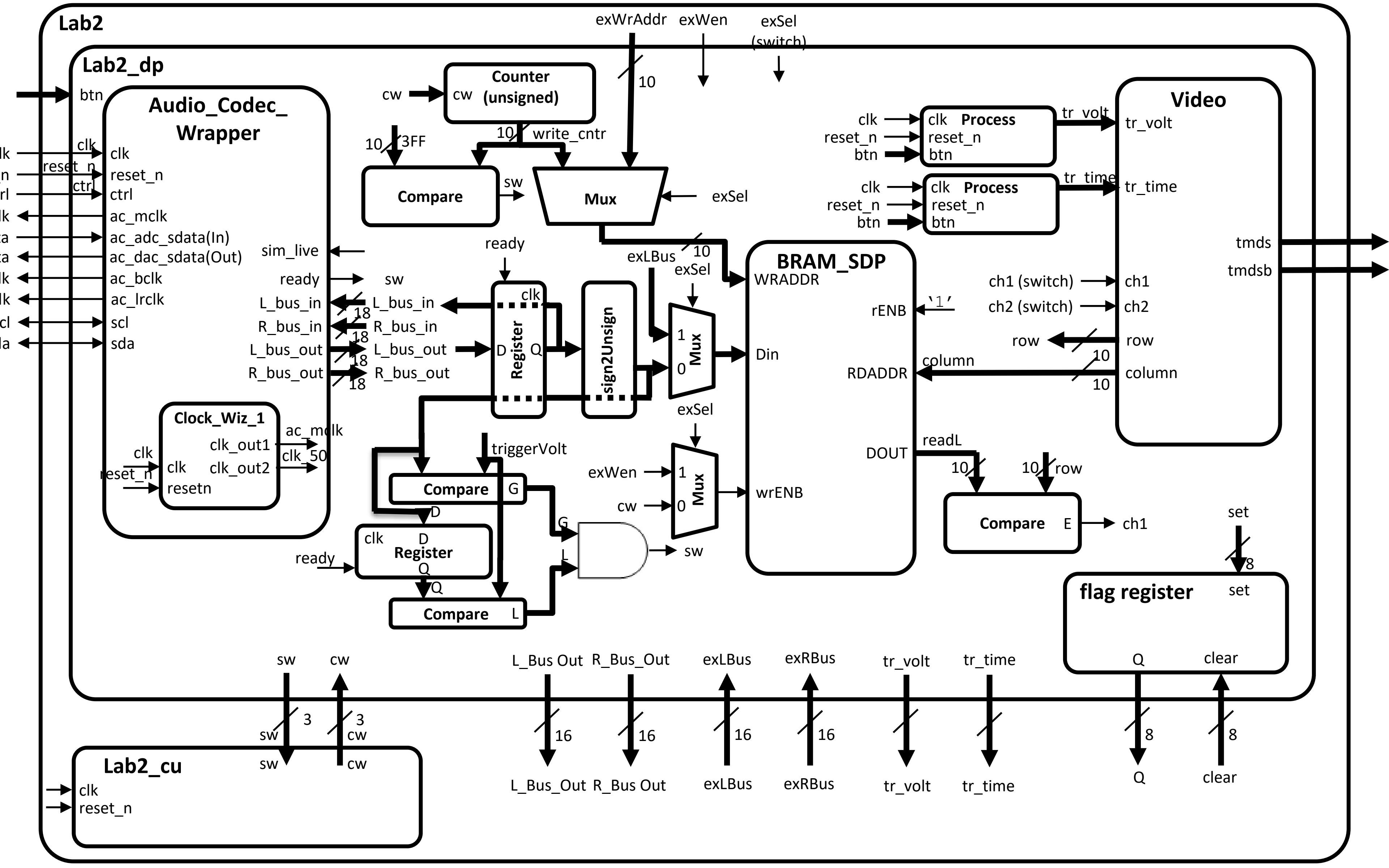
## MicroBlaze + Custom IP

# MicroBlaze + Custom IP what you are building today

(1st without the "roll"; then add "roll" in HW#10)



What does this have to do with Lab3???



Lab2
Lab2_dp
exWrAddr
exWen
exSel (switch)
10
btn
Audio_Codec_ Wrapper
clk
reset_n
ctrl
ac_mclk
ac_adc_sdata(In)
ac_dac_sdata(Out)
ac_bclk
ac_lrclk
scl
sda
sim_live
ready
sw
L_bus_in
R_bus_in
L_bus_out
R_bus_out
18
Clock_Wiz_1
clk
resetn
clk_out1
clk_out2
ac_mclk
clk_50
Counter (unsigned)
cw
10
write_cntr
3FF
Compare
sw
Mux
exSel
ready
Register
clk
D
Q
sign2Unsign
exLBus
exSel
1
0
Mux
triggerVolt
Compare
G
L
Register
ready
exWen
cw
BRAM_SDP
WRADDR
rENB
'1'
Din
RDADDR
DOUT
wrENB
readL
column
row
Compare
E
ch1
Video
tr_volt
tr_time
tmds
tmdsb
ch1 (switch)
ch2 (switch)
ch1
ch2
row
column
Process
clk
reset_n
btn
flag register
set
Q
clear
8
L_Bus Out
R_Bus_Out
exLBus
exRBus
tr_volt
tr_time
16
3
Lab2_cu
clk
reset_n
sw
cw
L_Bus_Out
R_Bus Out

- **Hook up 32 32-bit registers**

  **How many address bits do we need?**

  $2^x = 32$

  $x = \_\_\_$

  **Default code setup to read/write**

  **slv_reg0 to slv_reg31**

  **Base address: $\boxed{\text{0x44a00000}}$ → slv_reg0**

  **What address is slv_reg3?**

Byte Addressable,
not word addressable

32-bit word = __ bytes

__ bits to address

| | Hex |
|---|---|
| reg0 | 0x44a000 00 |
| reg1 | 0x44a000 __ |
| reg2 | 0x44a000 __ |
| reg3 | 0x44a000 __ |
| reg4 | 0x44a000 __ |

3 March 2021

UNITED STATES
AIR FORCE
ACADEMY

# MicroBlaze + Custom IP – Workflow

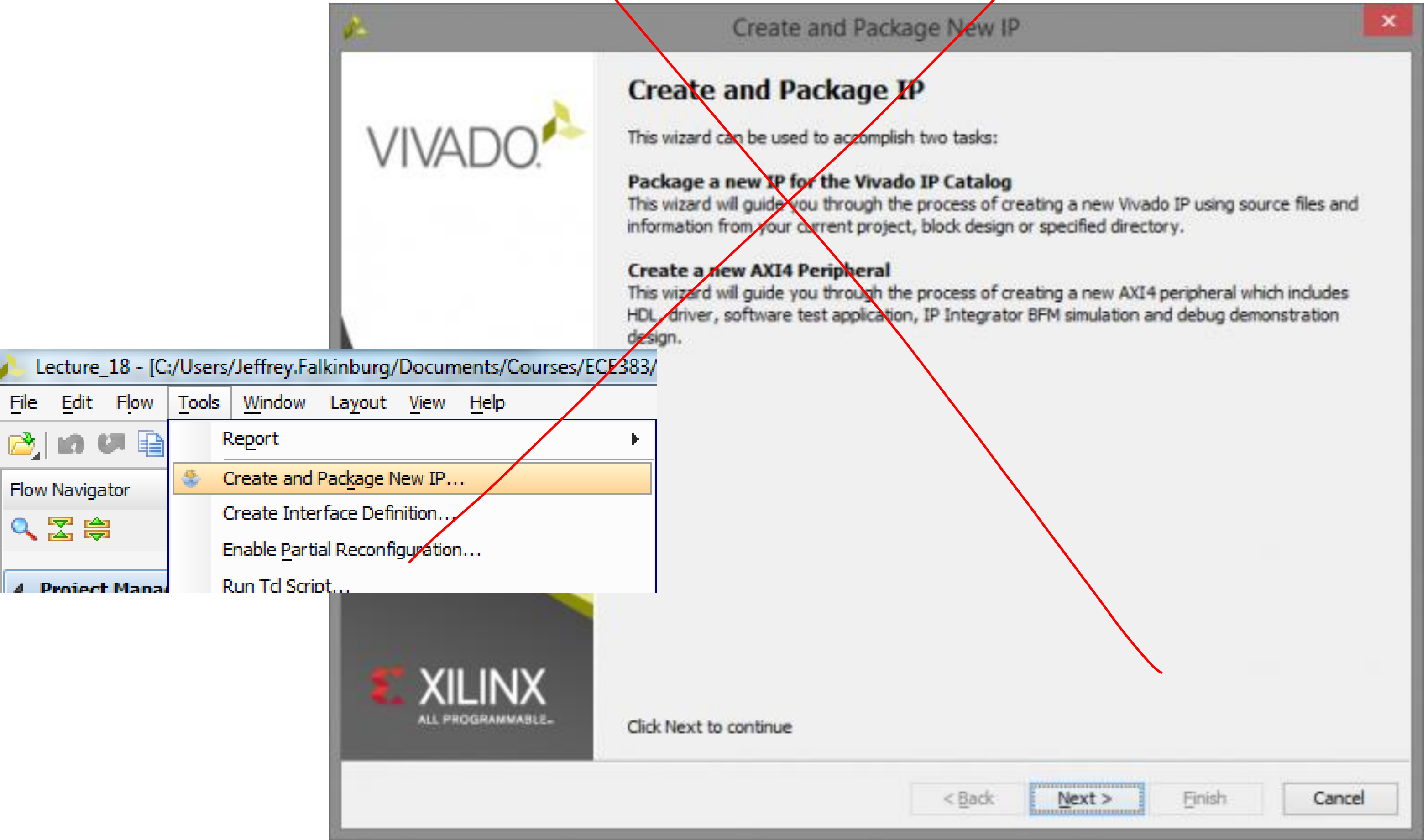
- The work flow has three main steps.
  1. Define a new hardware design (MicroBlaze + axi_uartlite) in Vivado IP Integrator ← **HW#9!**
  2. Create and package new custom IP (your custom hardware, which is a **my_counter**) and import it into your Vivado design
  3. Program the resulting hardware in the SDK environment.
- We will assume you did the last part of HW#9, and called the project **Lecture_18** (at least to step #11, validate design)

Follow next 70 screen shot slides? or

Follow 5 page Lec18_Install_short_version.pdf instead?

## ~~Xilinx Vivado – Create and Package Custom IP (my_counter)~~

- ~~**1. Open your Lecture_18 Vivado project**~~
  - ~~Go to **Tools→Create and package IP**~~


Create and Package New IP
VIVADO
XILINX
ALL PROGRAMMABLE
< Back
Next >
Finish
Cancel
Lecture_18 - [C:/Users/Jeffrey.Falkinburg/Documents/Courses/ECE383/
File
Edit
Flow
Tools
Window
Layout
View
Help
Flow Navigator
Project Mana
Report
Create and Package New IP...
Create Interface Definition...
Enable Partial Reconfiguration...
Run Tcl Script...

Create and Package IP

This wizard can be used to accomplish two tasks:

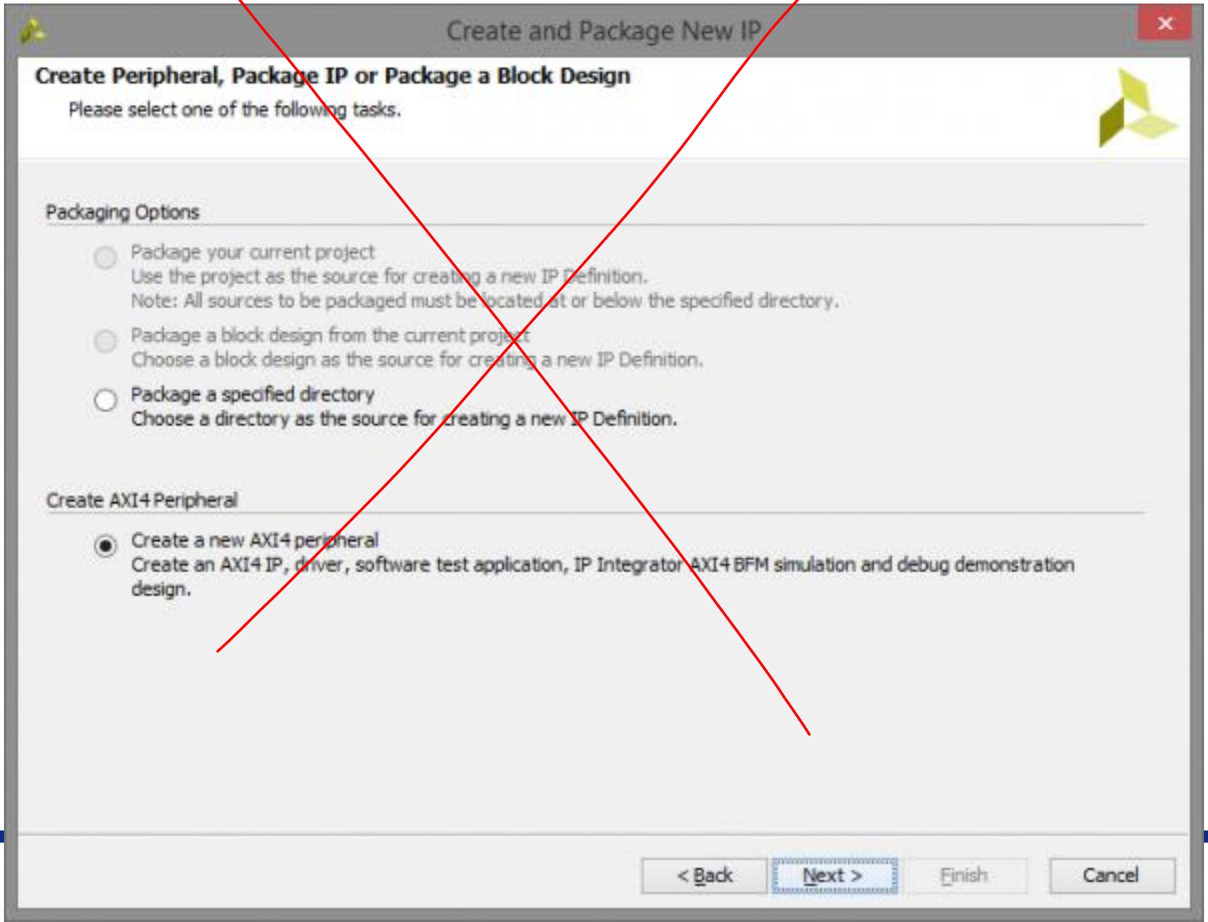
**Package a new IP for the Vivado IP Catalog**
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

**Create a new AXI4 Peripheral**
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator BFM simulation and debug demonstration design.

Click Next to continue

# Xilinx Vivado – Create and Package Custom IP

- **2. Create your custom IP project**
  - 2.1) Select **Create a new AXI4 peripheral** and click **Next**



Create and Package New IP

**Create Peripheral, Package IP or Package a Block Design**

Please select one of the following tasks.

Packaging Options

○ Package your current project
Use the project as the source for creating a new IP Definition.
Note: All sources to be packaged must be located at or below the specified directory.

○ Package a block design from the current project
Choose a block design as the source for creating a new IP Definition.

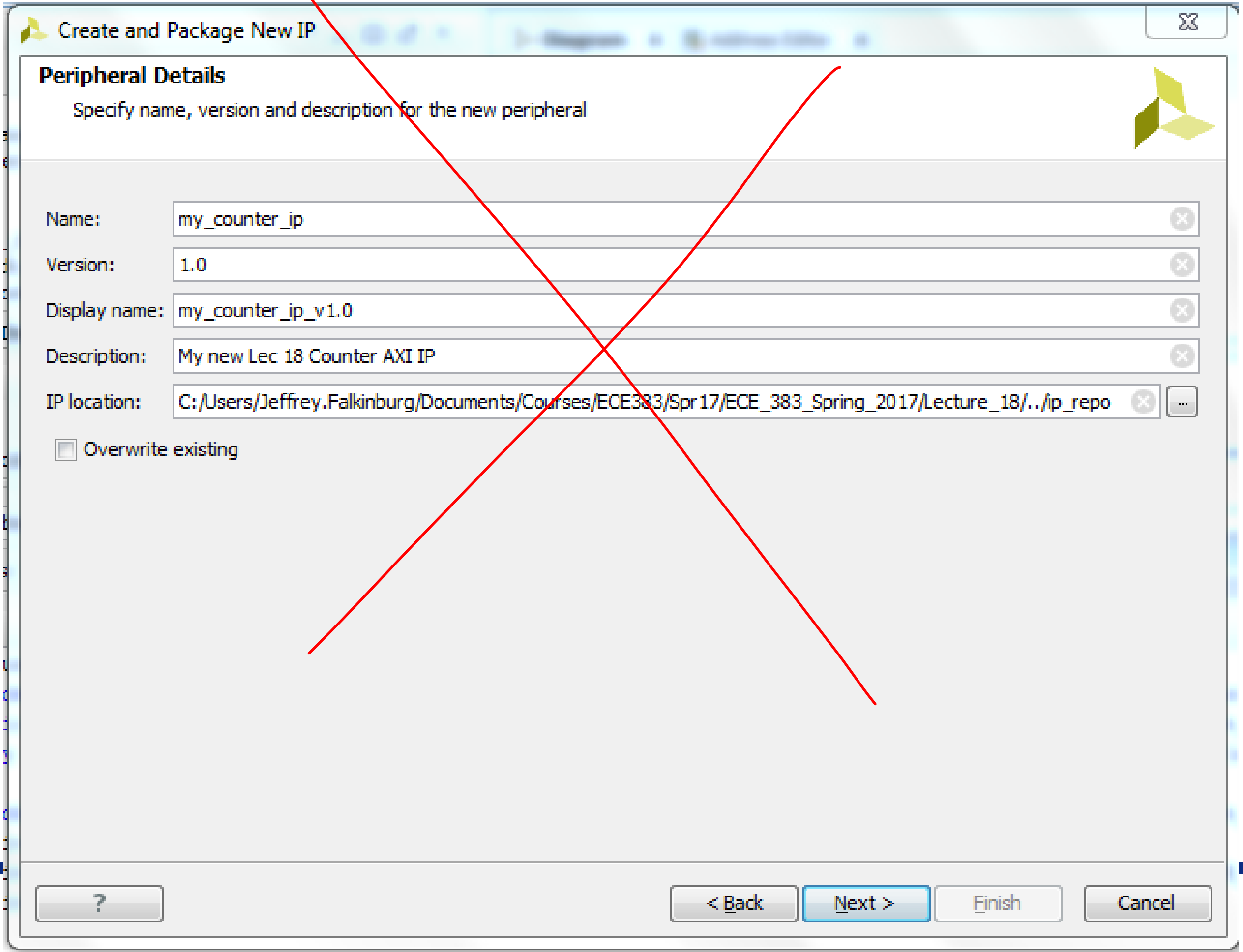
○ Package a specified directory
Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

[✓ Create a new AXI4 peripheral]
Create an AXI4 IP, driver, software test application, IP Integrator AXI4 BFM simulation and debug demonstration design.

# Xilinx Vivado – Create and Package Custom IP

- 2.2) Input “My_Counter_IP” in the name field and click **Next**



Create and Package New IP

**Peripheral Details**

Specify name, version and description for the new peripheral

Name: my_counter_ip

Version: 1.0

Display name: my_counter_ip_v1.0

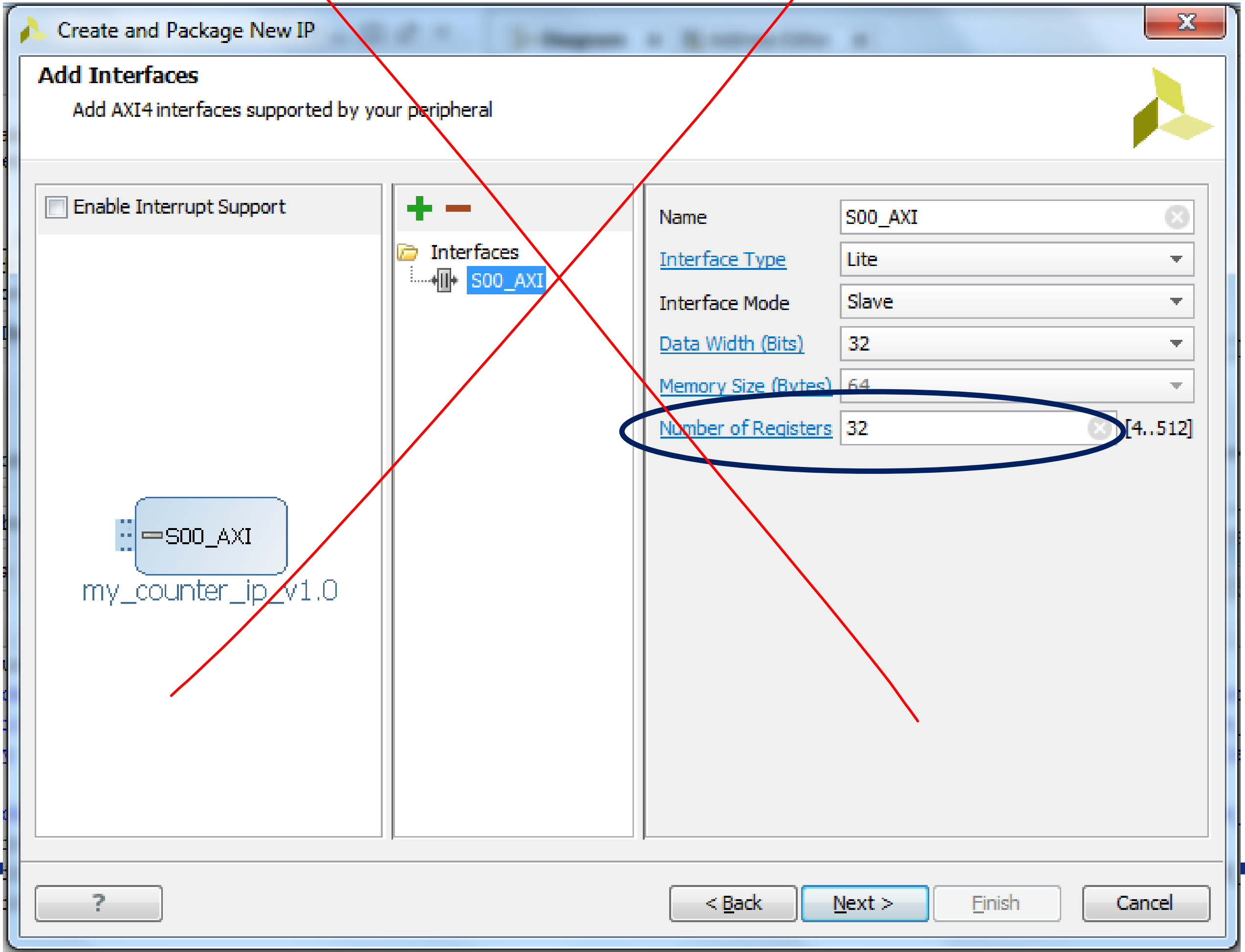
Description: My new Lec 18 Counter AXI IP

IP location: C:/Users/Jeffrey.Falkinburg/Documents/Courses/ECE383/Spr17/ECE_383_Spring_2017/Lecture_18/../ip_repo

☐ Overwrite existing

# Xilinx Vivado – Create and Package Custom IP

- 2.3) Change the number of Registers to 32 on the AXI interface and click **Next**



Create and Package New IP

**Add Interfaces**

Add AXI4 interfaces supported by your peripheral

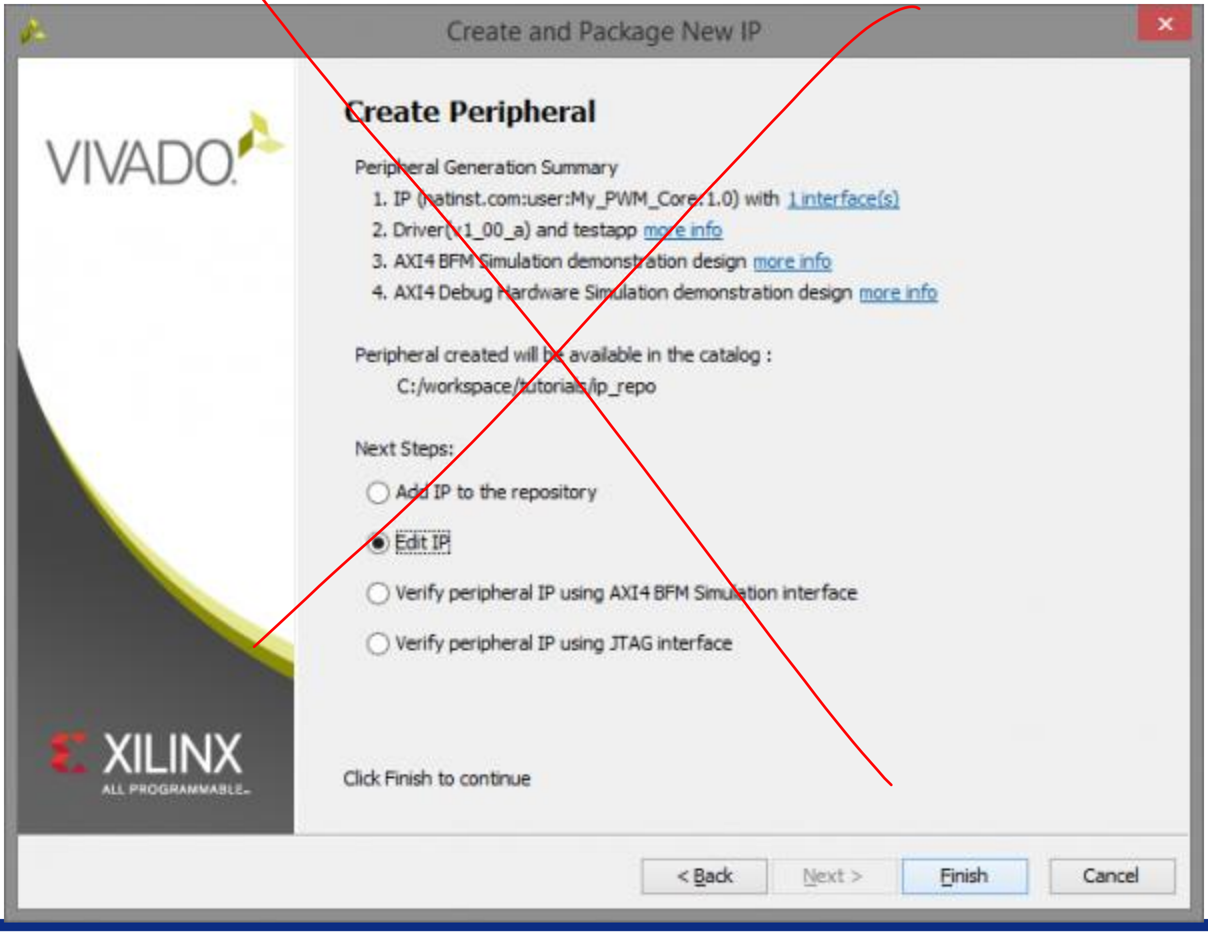
Enable Interrupt Support

Interfaces
S00_AXI

| | |
|---|---|
| Name | S00_AXI |
| Interface Type | Lite |
| Interface Mode | Slave |
| Data Width (Bits) | 32 |
| Memory Size (Bytes) | 64 |
| Number of Registers | 32 [4..512] |

# Xilinx Vivado – Create and Package Custom IP

- 2.4) Select **Edit IP** and click **Finish**



**Create Peripheral**

Peripheral Generation Summary

1. IP (natinst.com:user:My_PWM_Core:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 BFM Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :
C:/workspace/tutorials/ip_repo

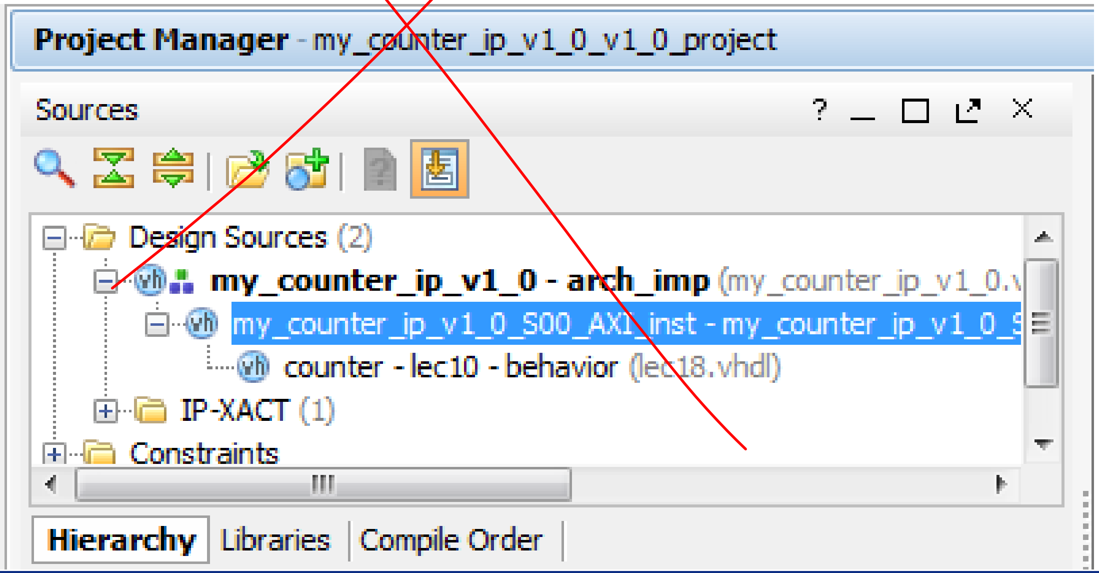
Next Steps:

- ○ Add IP to the repository
- [✓ Edit IP]
- ○ Verify peripheral IP using AXI4 BFM Simulation interface
- ○ Verify peripheral IP using JTAG interface

Click Finish to continue

# Xilinx Vivado – Create and Package Custom IP

- **3. Designing the IP core**
  - 3.1) A new instance of Vivado will open up for the new IP core. Expand the top level file **My_Counter_IP_v1_0**. Then double-click on **My_Counter_IP_v1_0_S00_AXI** to open it in the editor.



**Project Manager** - my_counter_ip_v1_0_v1_0_project

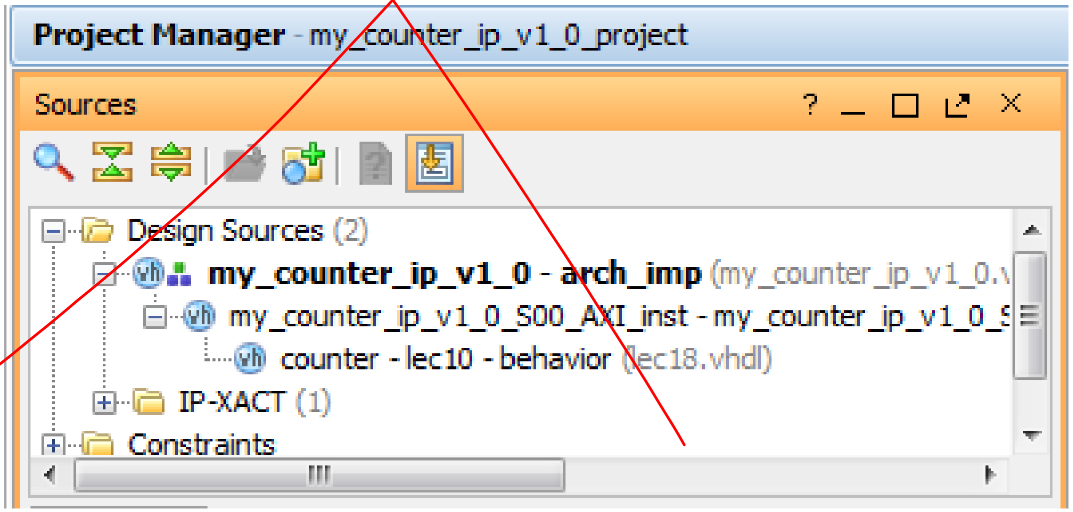
Sources

- Design Sources (2)
  - **my_counter_ip_v1_0 - arch_imp** (my_counter_ip_v1_0.v
    - my_counter_ip_v1_0_S00_AXI_inst - my_counter_ip_v1_0_S
      - counter - lec10 - behavior (lec18.vhdl)
  - IP-XACT (1)
- Constraints

UNITED STATES AIR FORCE ACADEMY

# Xilinx Vivado – Create and Package Custom IP

- **4. Add the Lec 10 Counter to the My_Counter_IP_v1_0**
  - by **"Add Sources" → lec18.vhd file**
  - Your counter will not yet be connected to the top level design


Project Manager - my_counter_ip_v1_0_project

Sources

Design Sources (2)
my_counter_ip_v1_0 - arch_imp (my_counter_ip_v1_0.v
my_counter_ip_v1_0_S00_AXI_inst - my_counter_ip_v1_0_S
counter - lec10 - behavior (lec18.vhdl)
IP-XACT (1)
Constraints

# Xilinx Vivado – Create and Package Custom IP

- **4.** The **lec18.vhd file** does not need to be modified
- **5.** The **My_Counter_IP_v1_0_S00_AXI.vhd** file needs to be modified with the changes seen in the Ninja version to support your counter (line numbers from Ninja version)
  - Lines 20, 112-122, 671, 759-766
- **6.** The **My_Counter_IP_v1_0.vhd** file needs to be modified with the changes seen in the version on Ninja
  - Lines 19, 59, 93

- The following slides, diagrams, and worksheet are intended to teach you what these modifications mean…
- Installation resumes on slide titled "**Packaging the IP core**"

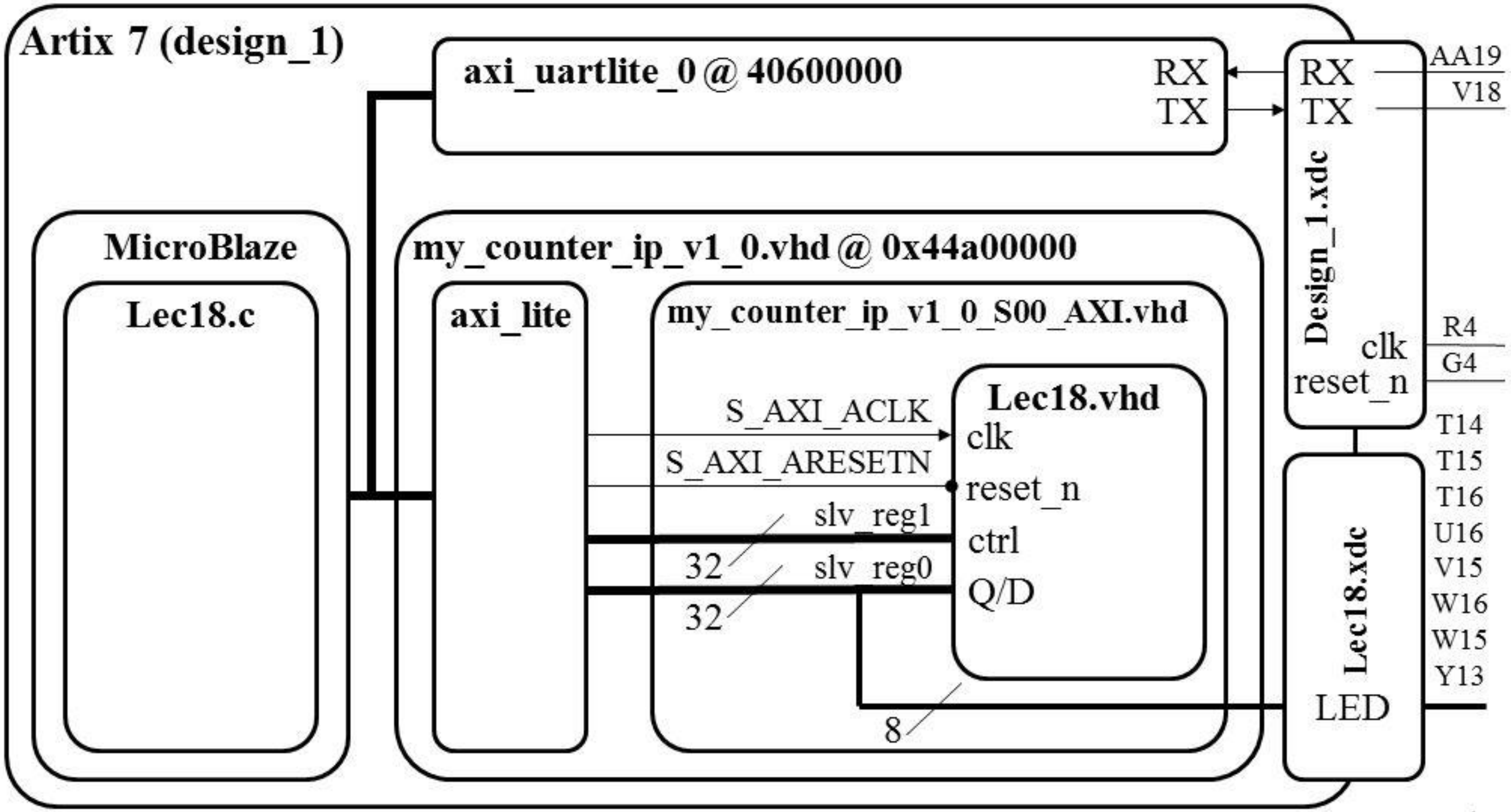
UNITED STATES
AIR FORCE
ACADEMY

## Part 1a: Hardware Questions/ Notes related to handout

- ~~Note: the truth table for the counter is in the comments.~~

UNITED STATES AIR FORCE ACADEMY

## MicroBlaze + Custom IP



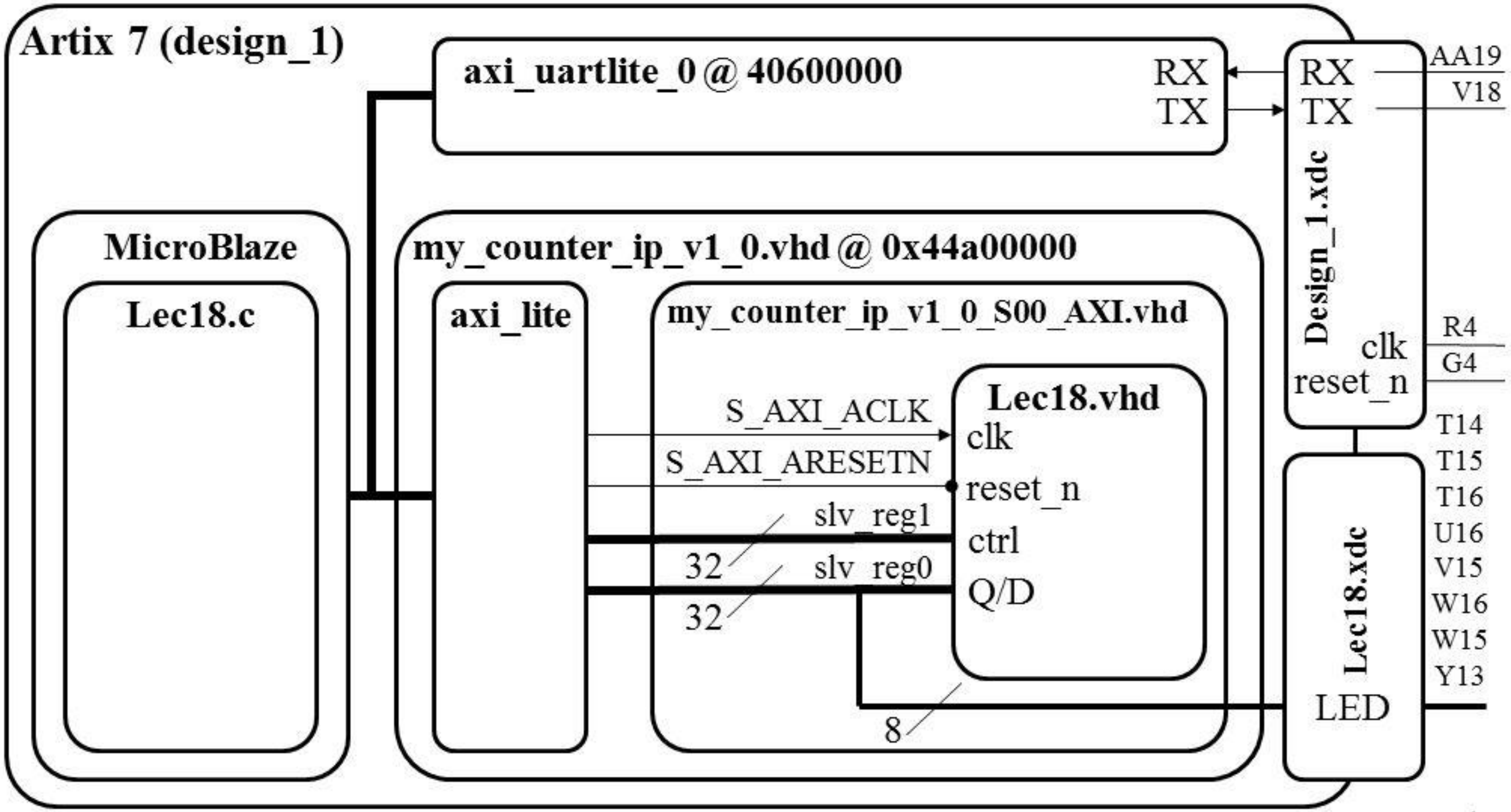
# Lec18.vhdl – Lec 10 Counter

```
-----------------------------------------------------------------------------
-- File:      lec18.vhdl
-- Ctrl:      00=Hold       01=Increment        10=Load       11=reset
-----------------------------------------------------------------------------
entity lec10 is
       generic( N           : integer := 4);
       port(  clk           : in  STD_LOGIC;
              reset_n              : in  STD_LOGIC;
              ctrl          : in std_logic_vector(1 downto 0);
              D             : in unsigned (N-1 downto 0);
              Q             : out unsigned (N-1 downto 0));
end lec10;
architecture behavior of lec10 is
       signal processQ: ___________________________________ ;
begin
       process(clk)
       begin
              if (rising_edge(clk)) then
                     if (reset_n = '0') then
                            processQ <= ___________________ ;
                     elsif (ctrl = "01") then
                            processQ <= ___________________ ;
                     elsif (ctrl = "10") then
                            processQ <= ___________________ ;
                     elsif (crtl = "11") then
                            processQ <= ___________________ ;
                     end if;
              end if;
       end process;
       Q <= processQ;
end behavior;
```

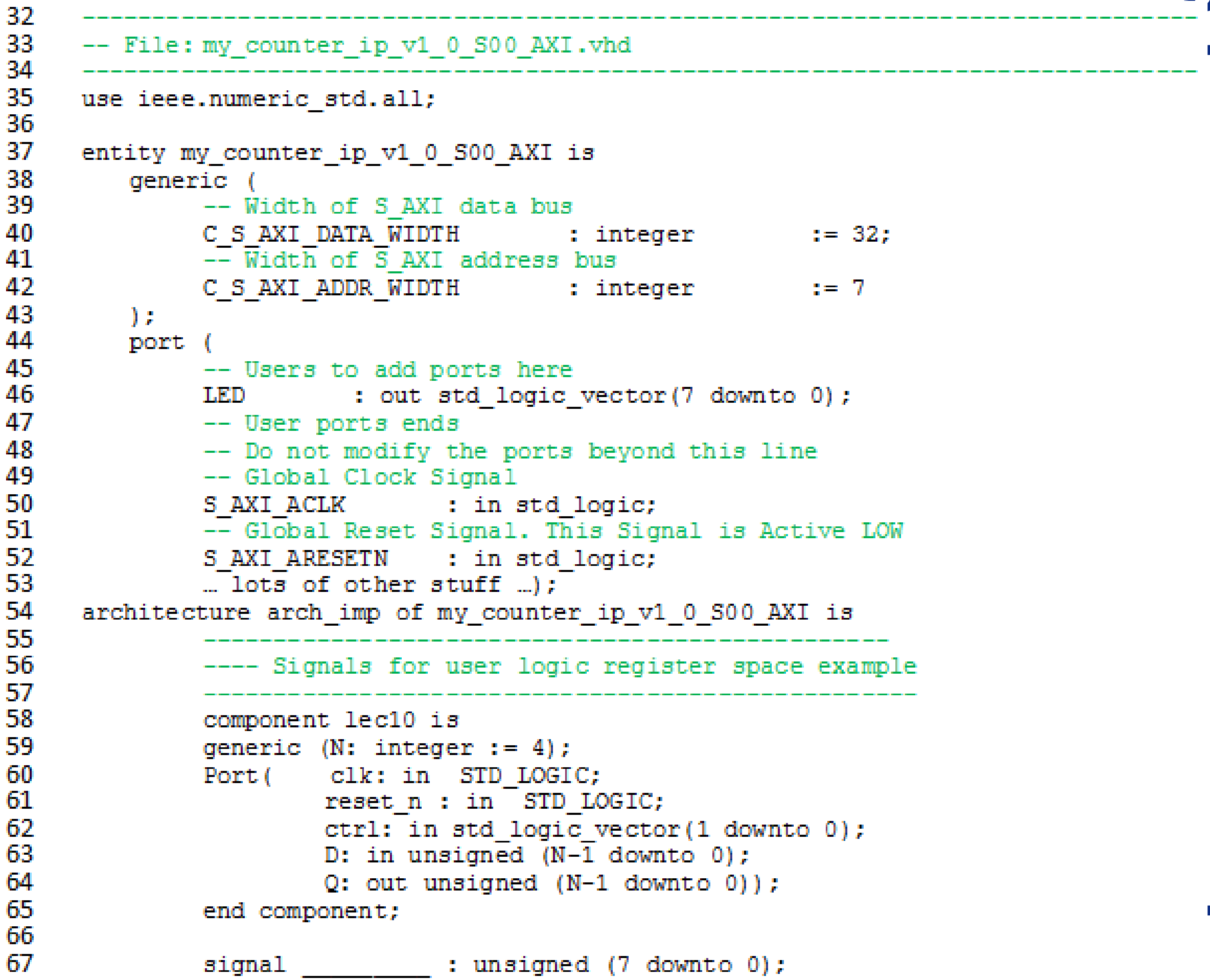
## Part 1b: Hardware Questions/ Notes related to handout

- Q: In my_counter_ip_v1_0_S00_AXI.vhd, what do the generics ..AXI_DATA_WIDTH, ..AXI_ADDR_WIDTH do?
- Q: In my_counter_ip_v1_0_S00_AXI.vhd, what two roles is slv_reg0 serving?
- Q: In my_counter_ip_v1_0_S00_AXI.vhd, what roles does slv_reg1 serve?
- Q: In my_counter_ip_v1_0_S00_AXI.vhd, slv_reg0 is on the left and right-hand side of an assignment. Identify the two lines where this happens.
- Q: In my_counter_ip_v1_0_S00_AXI.vhd, on line 62, what is the role does X"000000" serve?

# MicroBlaze + Custom IP



# my_counter_ip_v1_0_S00_AXI.vhd – User Logic



```
----------------------------------------------------------------------------------
-- File: my_counter_ip_v1_0_S00_AXI.vhd
----------------------------------------------------------------------------------
use ieee.numeric_std.all;

entity my_counter_ip_v1_0_S00_AXI is
    generic (
        -- Width of S_AXI data bus
        C_S_AXI_DATA_WIDTH        : integer        := 32;
        -- Width of S_AXI address bus
        C_S_AXI_ADDR_WIDTH        : integer        := 7
    );
    port (
        -- Users to add ports here
        LED        : out std_logic_vector(7 downto 0);
        -- User ports ends
        -- Do not modify the ports beyond this line
        -- Global Clock Signal
        S_AXI_ACLK        : in std_logic;
        -- Global Reset Signal. This Signal is Active LOW
        S_AXI_ARESETN     : in std_logic;
        … lots of other stuff …);
architecture arch_imp of my_counter_ip_v1_0_S00_AXI is
        ------------------------------------------------
        ---- Signals for user logic register space example
        --------------------------------------------------

        component lec10 is
        generic (N: integer := 4);
        Port(    clk: in  STD_LOGIC;
                reset_n : in  STD_LOGIC;
                ctrl: in std_logic_vector(1 downto 0);
                D: in unsigned (N-1 downto 0);
                Q: out unsigned (N-1 downto 0));
        end component;

        signal ________ : unsigned (7 downto 0);
```

# my_counter_ip_v1_0_S00_AXI.vhd – User Logic

```
begin
        -- Address decoding for reading registers
        loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
        case loc_addr is
                when b"00000" =>
                        reg_data_out <= X"000000" & std_logic_vector(Q);
                when b"00001" =>
                        reg_data_out <= slv_reg1;
                when b"00010" =>
                        reg_data_out <= slv_reg2;
                … lots more stuff here
        end case;
        -- Add user logic here
        counter: ___________
                generic map (____)
                port map(       clk =>          ____________________________,
                                reset_n =>      ____________________________,
                                ctrl =>         slv_reg1(1 downto 0),
                                D =>            unsigned(slv_reg0(7 downto 0)),
                                Q =>            _______________________________);
        LED <= std_logic_vector(Q);
        -- User logic ends
```

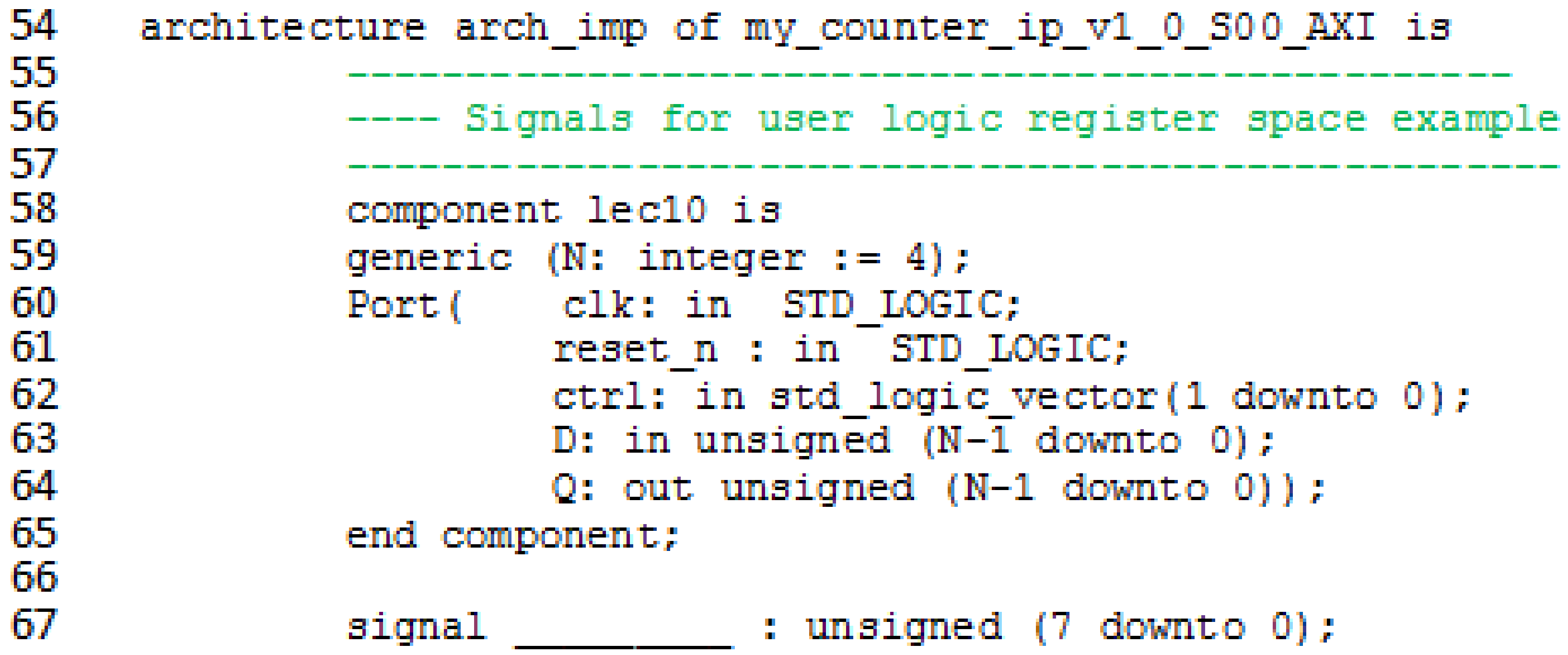
# Xilinx Vivado – Create and Package Custom IP

- **5. Modifying My_Counter_IP_v1_0_S00_AXI axi bus interface file**

```
------------------------------------------------------------------------
-- File: my_counter_ip_v1_0_S00_AXI.vhd
------------------------------------------------------------------------
use ieee.numeric_std.all;

entity my_counter_ip_v1_0_S00_AXI is
    generic (
        -- Width of S_AXI data bus
        C_S_AXI_DATA_WIDTH          : integer           := 32;
        -- Width of S_AXI address bus
        C_S_AXI_ADDR_WIDTH          : integer           := 7
    );
    port (
        -- Users to add ports here
        LED         : out std_logic_vector(7 downto 0);
        -- User ports ends
        -- Do not modify the ports beyond this line
        -- Global Clock Signal
        S_AXI_ACLK          : in std_logic;
        -- Global Reset Signal. This Signal is Active LOW
        S_AXI_ARESETN       : in std_logic;
        … lots of other stuff …);
```

# Xilinx Vivado – Create and Package Custom IP

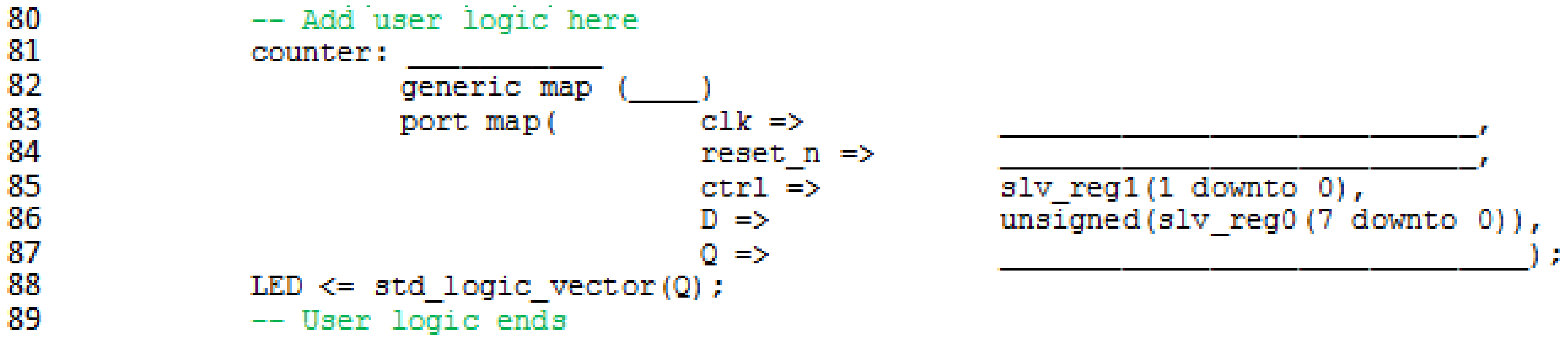
- **5. Modifying My_Counter_IP_v1_0_S00_AXI axi bus interface file**
  - Add the counter declaration and a signal for Q before begin in the my_counter_ip_v1_0_S00_AXI architecture:



```
architecture arch_imp of my_counter_ip_v1_0_S00_AXI is
        ------------------------------------------------
        ---- Signals for user logic register space example
        --------------------------------------------------
        component lec10 is
        generic (N: integer := 4);
        Port(    clk: in   STD_LOGIC;
                 reset_n : in   STD_LOGIC;
                 ctrl: in std_logic_vector(1 downto 0);
                 D: in unsigned (N-1 downto 0);
                 Q: out unsigned (N-1 downto 0));
        end component;

        signal _________ : unsigned (7 downto 0);
```

# Xilinx Vivado – Create and Package Custom IP

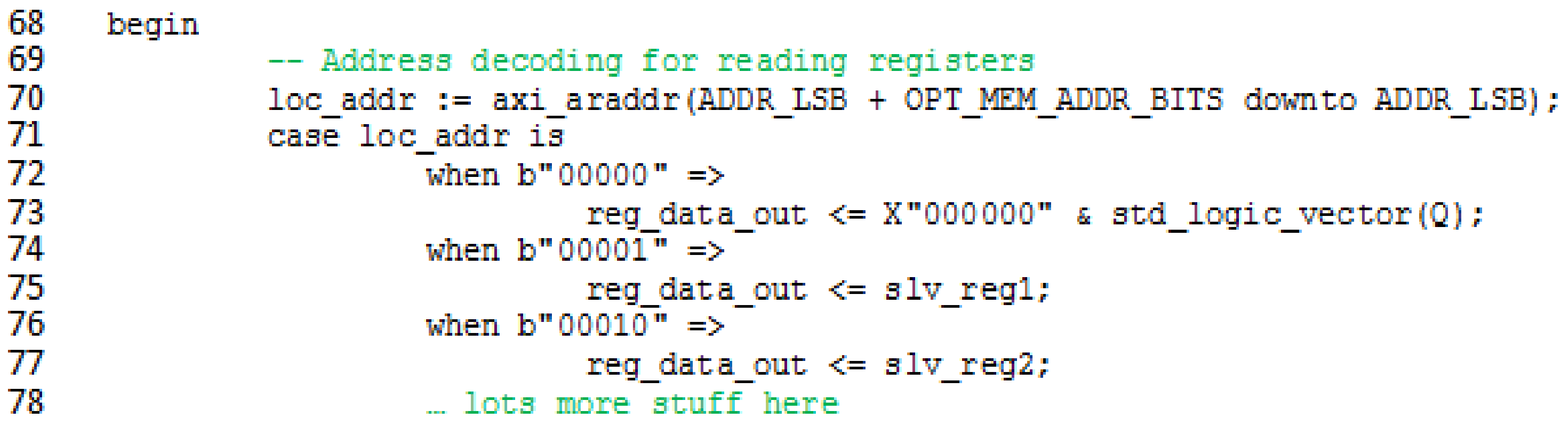
- **5. Modifying My_Counter_IP_v1_0_S00_AXI axi bus interface file**
  - Add the counter implementation and connect the wires within the my_counter_ip_v1_0_S00_AXI architecture:



```
-- Add user logic here
counter: ___________
        generic map (____)
        port map(         clk =>           __________________________,
                          reset_n =>       __________________________,
                          ctrl =>          slv_reg1(1 downto 0),
                          D =>             unsigned(slv_reg0(7 downto 0)),
                          Q =>             ____________________________);
LED <= std_logic_vector(Q);
-- User logic ends
```

# Xilinx Vivado – Create and Package Custom IP

- **5. Modifying My_Counter_IP_v1_0_S00_AXI axi bus interface file**
  - Connect the counter implementation Q output signal to slave register 0 (slv_reg0) in the my_counter_ip_v1_0_S00_AXI architecture:



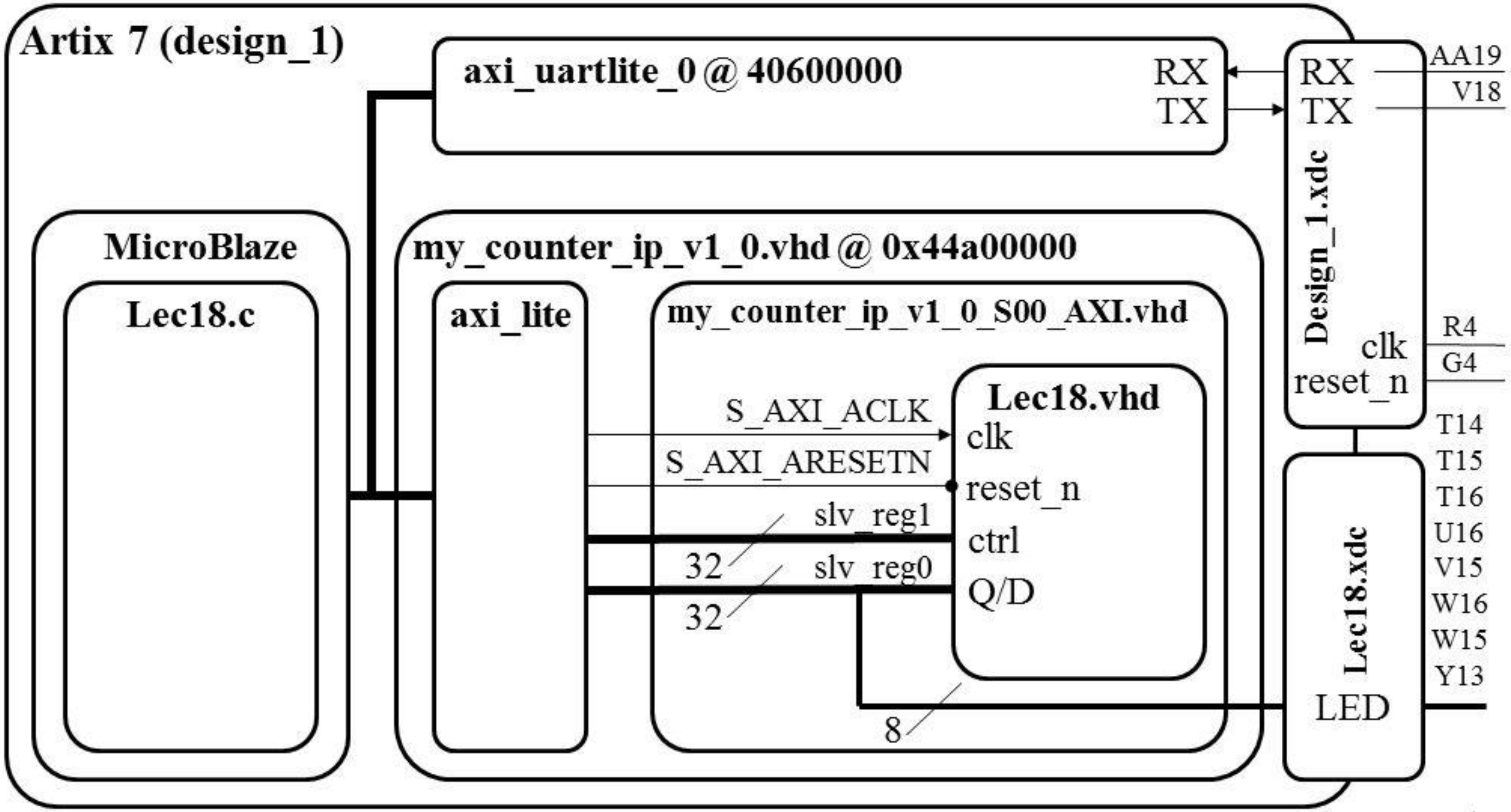
```
begin
        -- Address decoding for reading registers
        loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
        case loc_addr is
                when b"00000" =>
                        reg_data_out <= X"000000" & std_logic_vector(Q);
                when b"00001" =>
                        reg_data_out <= slv_reg1;
                when b"00010" =>
                        reg_data_out <= slv_reg2;
                … lots more stuff here
```

UNITED STATES
AIR FORCE
ACADEMY

## Part 1c: Hardware Questions/ Notes related to handout

- ~~Q: If you want a signal to go outside the Artix 7 chip...~~
  - ~~What files must it appear on the entity description?~~
  - ~~What other files must contain information about the signal?~~
- ~~Q: If you want a signal to go to the MicroBlaze...~~
  - ~~What files must it appear on the entity description?~~
  - ~~In order for the MicroBlaze to read the signal, what must you do?~~
  - ~~In order for the MicroBlaze to write to the signal, what must you do?~~

# MicroBlaze + Custom IP



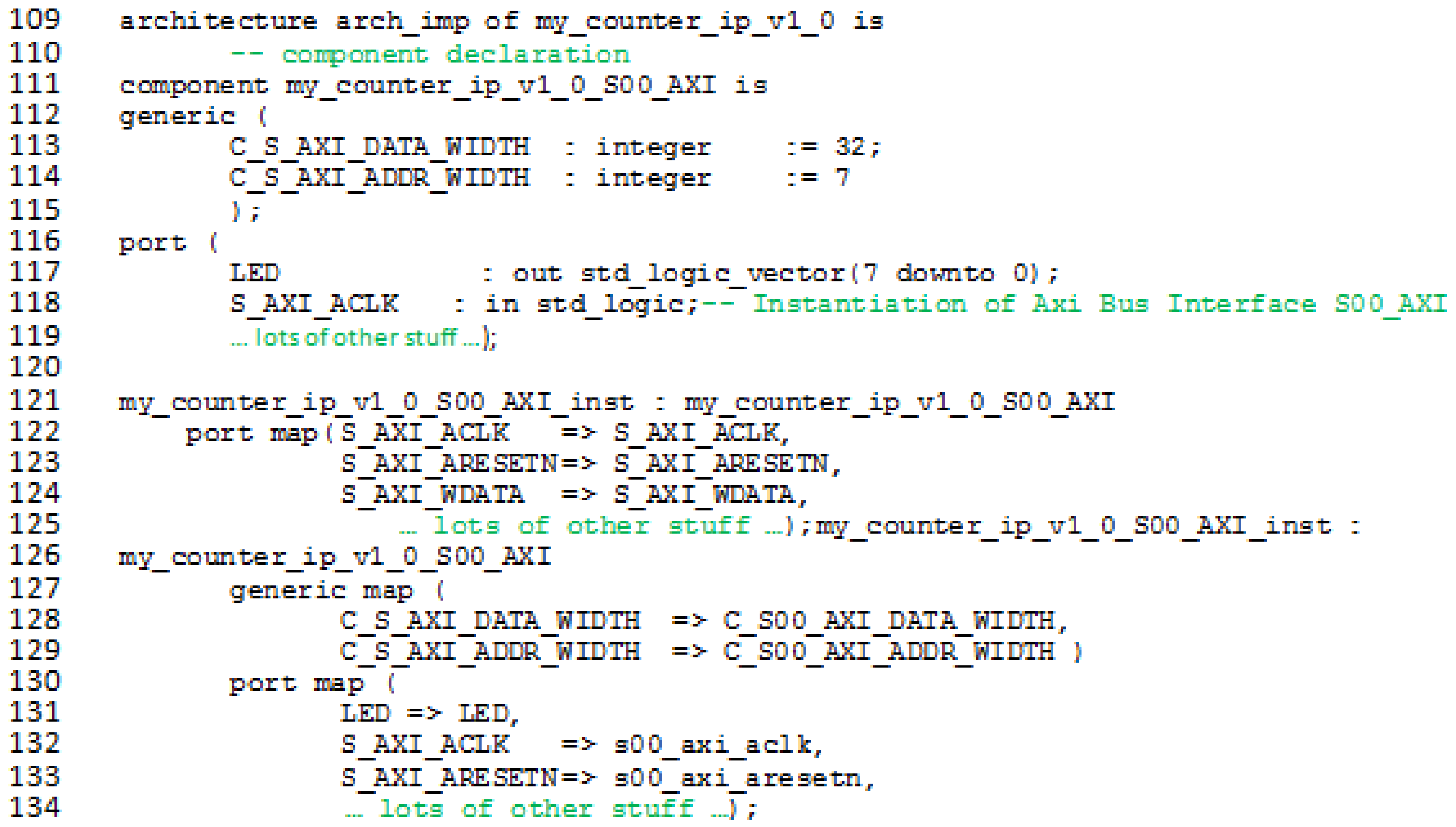
# My_Counter_IP_v1_0.vhd – Top Level

```
-------------------------------------------------------------------------------
-- my_counter_ip_v1_0.vhd
-------------------------------------------------------------------------------
use ieee.numeric_std.all;
entity my_counter_ip_v1_0 is
generic (
        C_S00_AXI_DATA_WIDTH     : integer          := 32;
        C_S00_AXI_ADDR_WIDTH     : integer          := 7
        );
port (
        -- Users to add ports here
        LED : out std_logic_vector(7 downto 0);
        -- User ports ends
        -- Do not modify the ports beyond this line
        -- Ports of Axi Slave Bus Interface S00_AXI
        s00_axi_aclk      : in std_logic;
        s00_axi_aresetn   : in std_logic;
        … lots of other stuff …);

architecture arch_imp of my_counter_ip_v1_0 is
      -- component declaration
component my_counter_ip_v1_0_S00_AXI is
generic (
      C_S_AXI_DATA_WIDTH  : integer     := 32;
      C_S_AXI_ADDR_WIDTH  : integer     := 7
      );
port (
      LED               : out std_logic_vector(7 downto 0);
      S_AXI_ACLK    : in std_logic;-- Instantiation of Axi Bus Interface S00_AXI
      … lots of other stuff …);
```

*Integrity - Service - Excellence*

UNITED STATES AIR FORCE ACADEMY

# My_Counter_IP_v1_0.vhd – Top Level

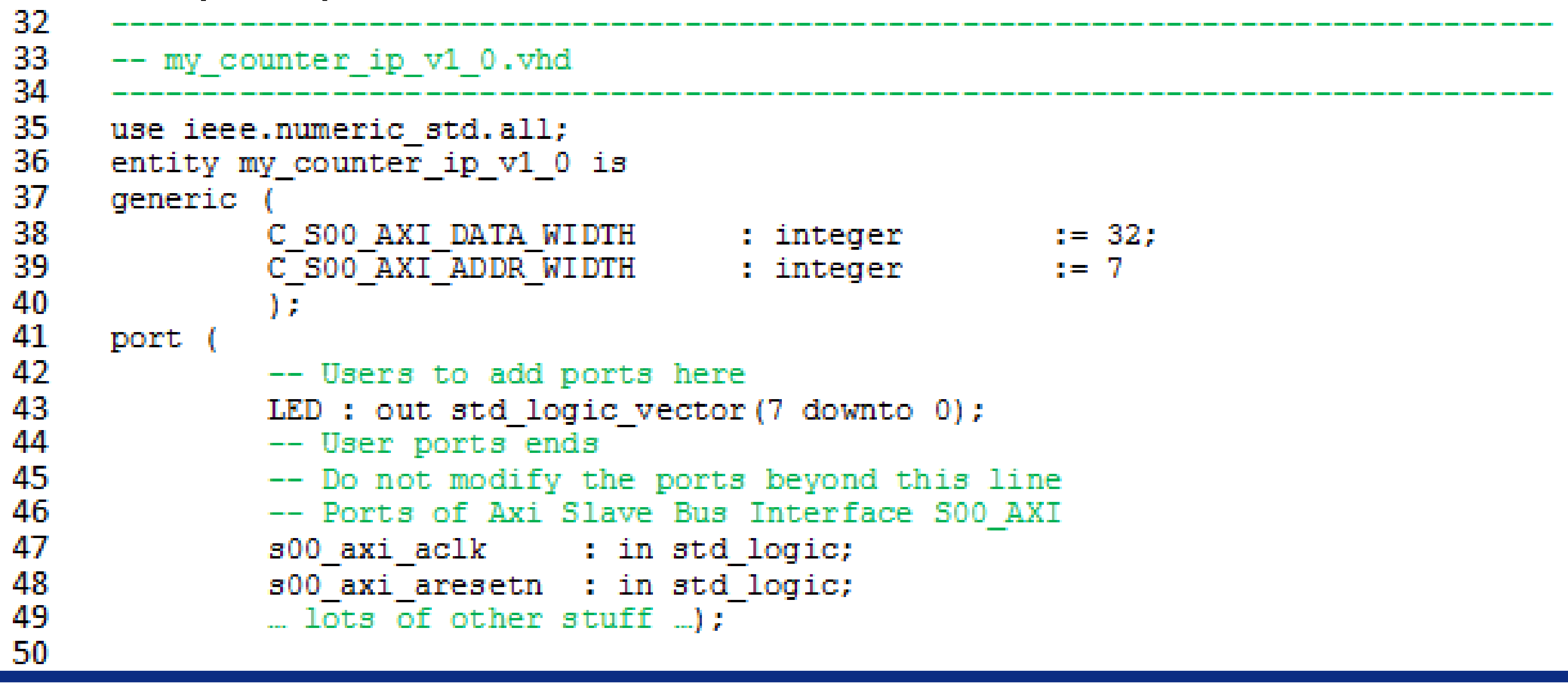


```
architecture arch_imp of my_counter_ip_v1_0 is
      -- component declaration
component my_counter_ip_v1_0_S00_AXI is
generic (
      C_S_AXI_DATA_WIDTH  : integer     := 32;
      C_S_AXI_ADDR_WIDTH  : integer     := 7
      );
port (
      LED               : out std_logic_vector(7 downto 0);
      S_AXI_ACLK    : in std_logic;-- Instantiation of Axi Bus Interface S00_AXI
      .... lots of other stuff ...);

my_counter_ip_v1_0_S00_AXI_inst : my_counter_ip_v1_0_S00_AXI
    port map(S_AXI_ACLK    => S_AXI_ACLK,
             S_AXI_ARESETN=> S_AXI_ARESETN,
             S_AXI_WDATA   => S_AXI_WDATA,
                  ... lots of other stuff ...);my_counter_ip_v1_0_S00_AXI_inst :
my_counter_ip_v1_0_S00_AXI
      generic map (
             C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
             C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH )
      port map (
             LED => LED,
             S_AXI_ACLK    => s00_axi_aclk,
             S_AXI_ARESETN=> s00_axi_aresetn,
             ... lots of other stuff ...);
```

# Xilinx Vivado – Create and Package Custom IP

- **6. Modifying My_Counter_IP_v1_0 top level file**
  - Add a port for the LEDs in the my_counter_ip_v1_0 entity between the comments to expose it to externally:

    port (



```
------------------------------------------------------------------------------
-- my_counter_ip_v1_0.vhd
------------------------------------------------------------------------------
use ieee.numeric_std.all;
entity my_counter_ip_v1_0 is
generic (
        C_S00_AXI_DATA_WIDTH      : integer        := 32;
        C_S00_AXI_ADDR_WIDTH      : integer        := 7
        );
port (
        -- Users to add ports here
        LED : out std_logic_vector(7 downto 0);
        -- User ports ends
        -- Do not modify the ports beyond this line
        -- Ports of Axi Slave Bus Interface S00_AXI
        s00_axi_aclk      : in std_logic;
        s00_axi_aresetn   : in std_logic;
        … lots of other stuff …);
```

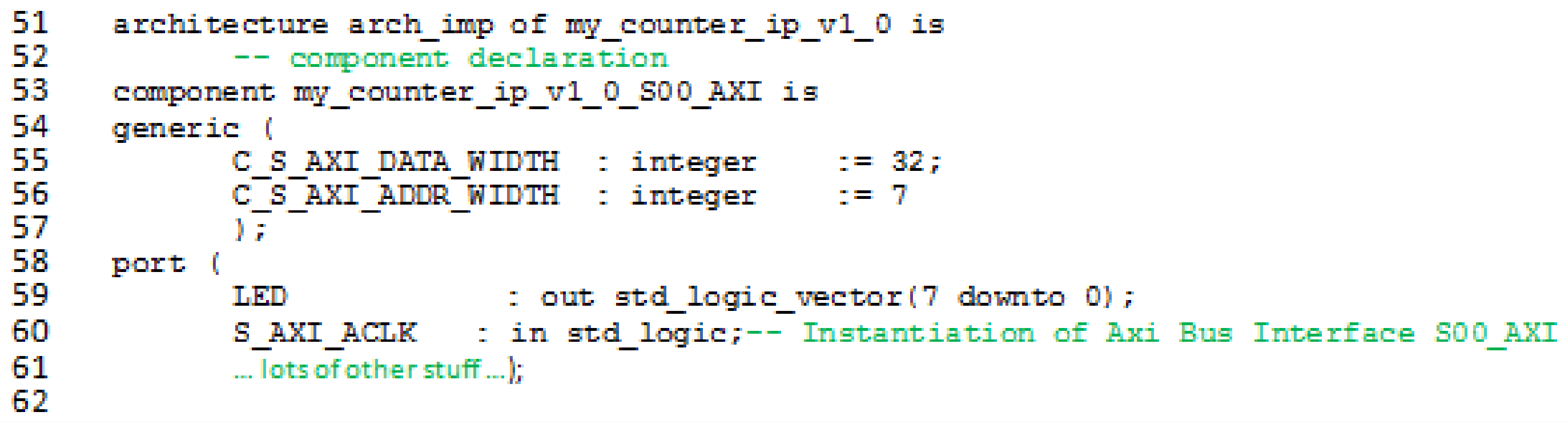
# Xilinx Vivado – Create and Package Custom IP

- **6. Modifying My_Counter_IP_v1_0 top level file**
  - Add a port for the LEDs in the my_counter_ip_v1_0_S00_AXI component declaration:

    port (

    **LED : out std_logic_vector(7 downto 0);**

    S_AXI_ACLK : in std_logic;

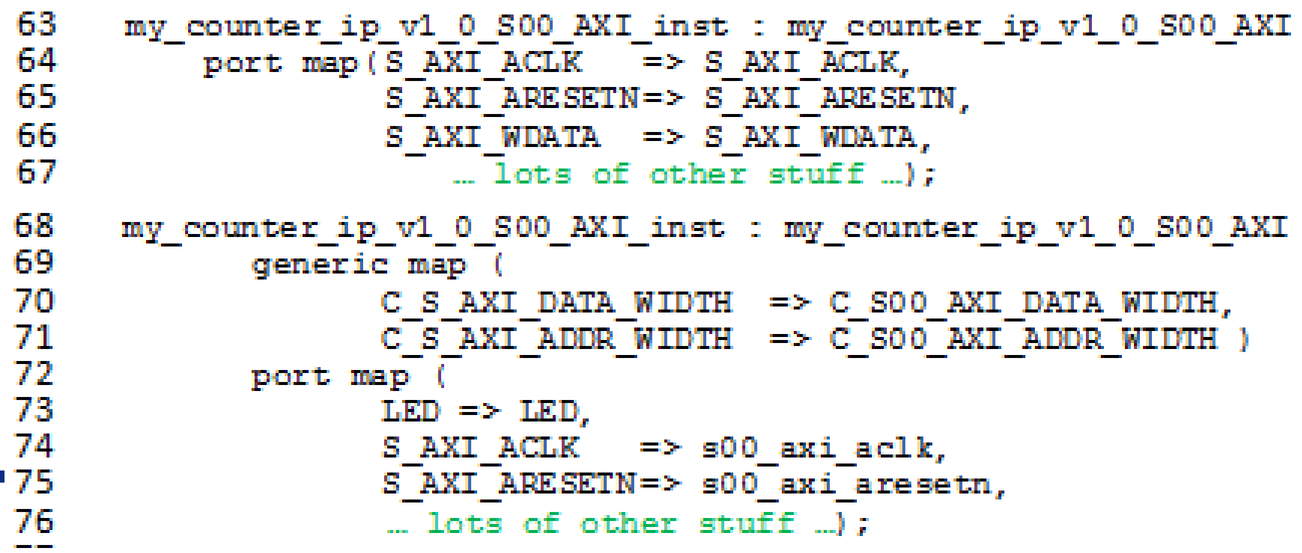


```
architecture arch_imp of my_counter_ip_v1_0 is
        -- component declaration
component my_counter_ip_v1_0_S00_AXI is
generic (
        C_S_AXI_DATA_WIDTH  : integer    := 32;
        C_S_AXI_ADDR_WIDTH  : integer    := 7
        );
port (
        LED               : out std_logic_vector(7 downto 0);
        S_AXI_ACLK    : in std_logic;-- Instantiation of Axi Bus Interface S00_AXI
        ... lots of other stuff ...);
```

# Xilinx Vivado – Create and Package Custom IP

- **6. Modifying My_Counter_IP_v1_0 top level file**
  - Add a port map for the LEDs in the my_counter_ip_v1_0_S00_AXI component instantiation:

```
port map (
        LED => LED,
        S_AXI_ACLK  => s00_axi_aclk,
```



```
my_counter_ip_v1_0_S00_AXI_inst : my_counter_ip_v1_0_S00_AXI
    port map(S_AXI_ACLK    => S_AXI_ACLK,
             S_AXI_ARESETN=> S_AXI_ARESETN,
             S_AXI_WDATA   => S_AXI_WDATA,
                 … lots of other stuff …);

my_counter_ip_v1_0_S00_AXI_inst : my_counter_ip_v1_0_S00_AXI
        generic map (
                C_S_AXI_DATA_WIDTH   => C_S00_AXI_DATA_WIDTH,
                C_S_AXI_ADDR_WIDTH   => C_S00_AXI_ADDR_WIDTH )
        port map (
                LED => LED,
                S_AXI_ACLK    => s00_axi_aclk,
                S_AXI_ARESETN=> s00_axi_aresetn,
                … lots of other stuff …);
```

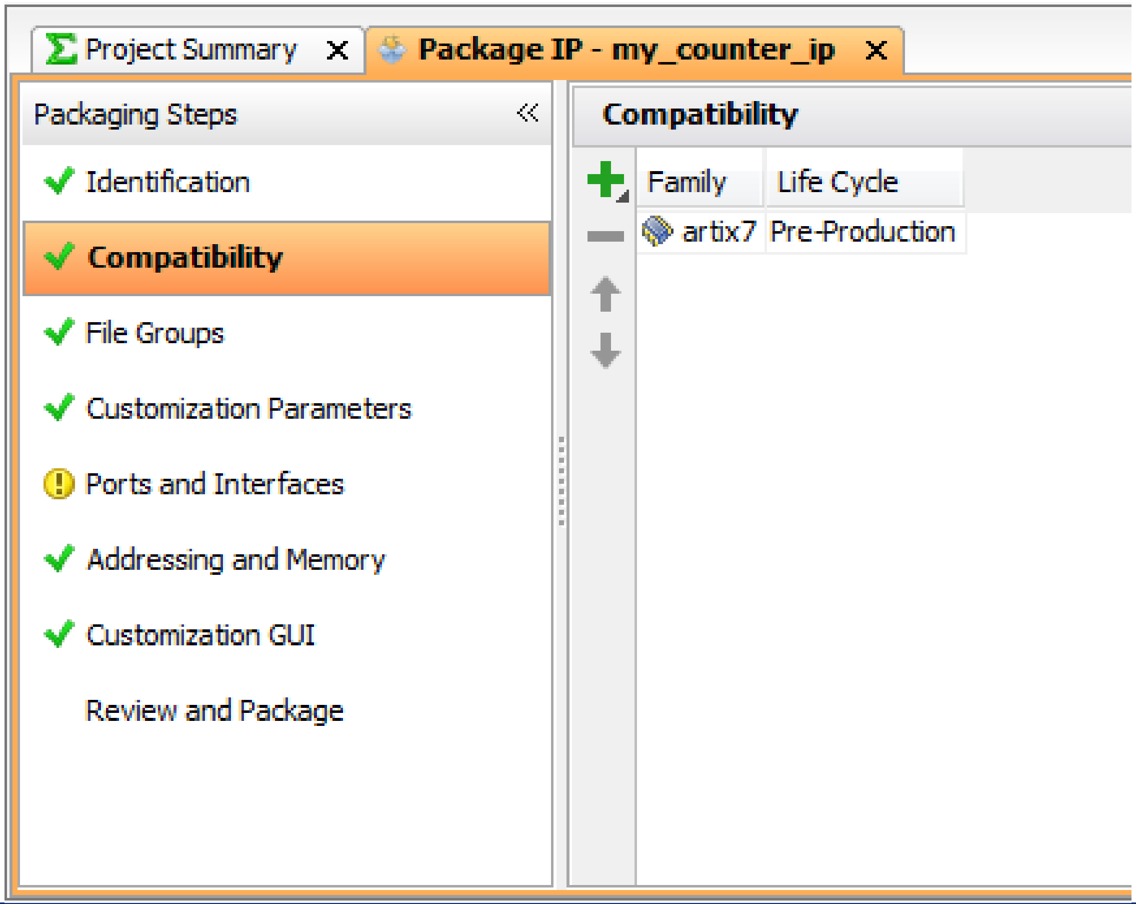
UNITED STATES AIR FORCE ACADEMY

# Packaging the IP core

- **7. Packaging the IP core**
  - Now that we have written the core, it is time to package up the HDL to create a complete IP package.
  - 7.1) Now click on **Package IP** in the Flow Navigator and you should see the Package IP tab.
  - Select **Compatibility** (under Packaging Steps) and make sure “Artix7” are present. If those are not there, you can add them by clicking the plus button. The Life Cycle does not matter at this point.

# Xilinx Vivado – Create and Package Custom IP

- 7. Packaging the IP core



Project Summary ✕ | Package IP - my_counter_ip ✕

Packaging Steps

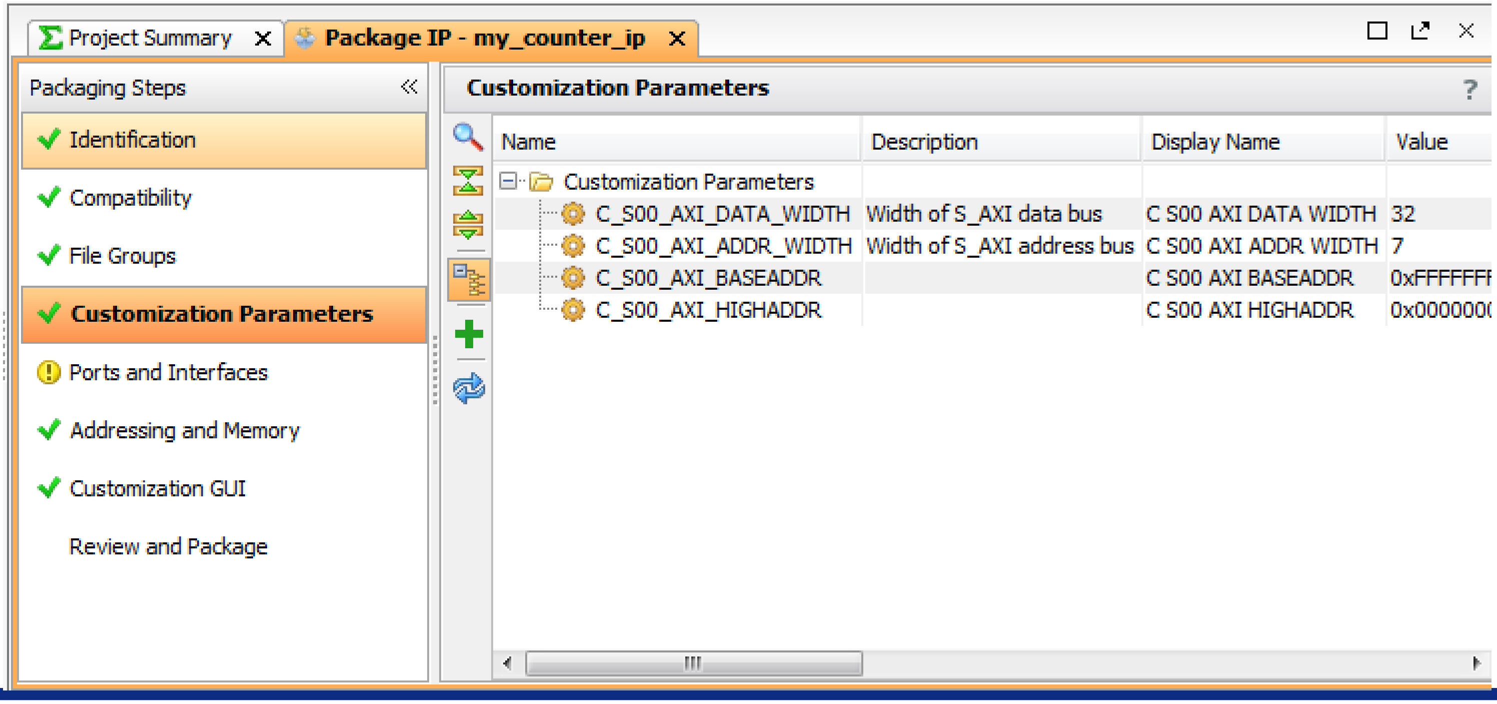
- ✓ Identification
- ✓ **Compatibility**
- ✓ File Groups
- ✓ Customization Parameters
- ! Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- Review and Package

**Compatibility**

| Family | Life Cycle |
|---|---|
| artix7 | Pre-Production |

# Xilinx Vivado – Create and Package Custom IP

- 7.2) Select **Customization Parameters** and select the line for **Merge Changes from Customization Parameters Wizard**. This will have the My_Counter_IP parameters from the top file.

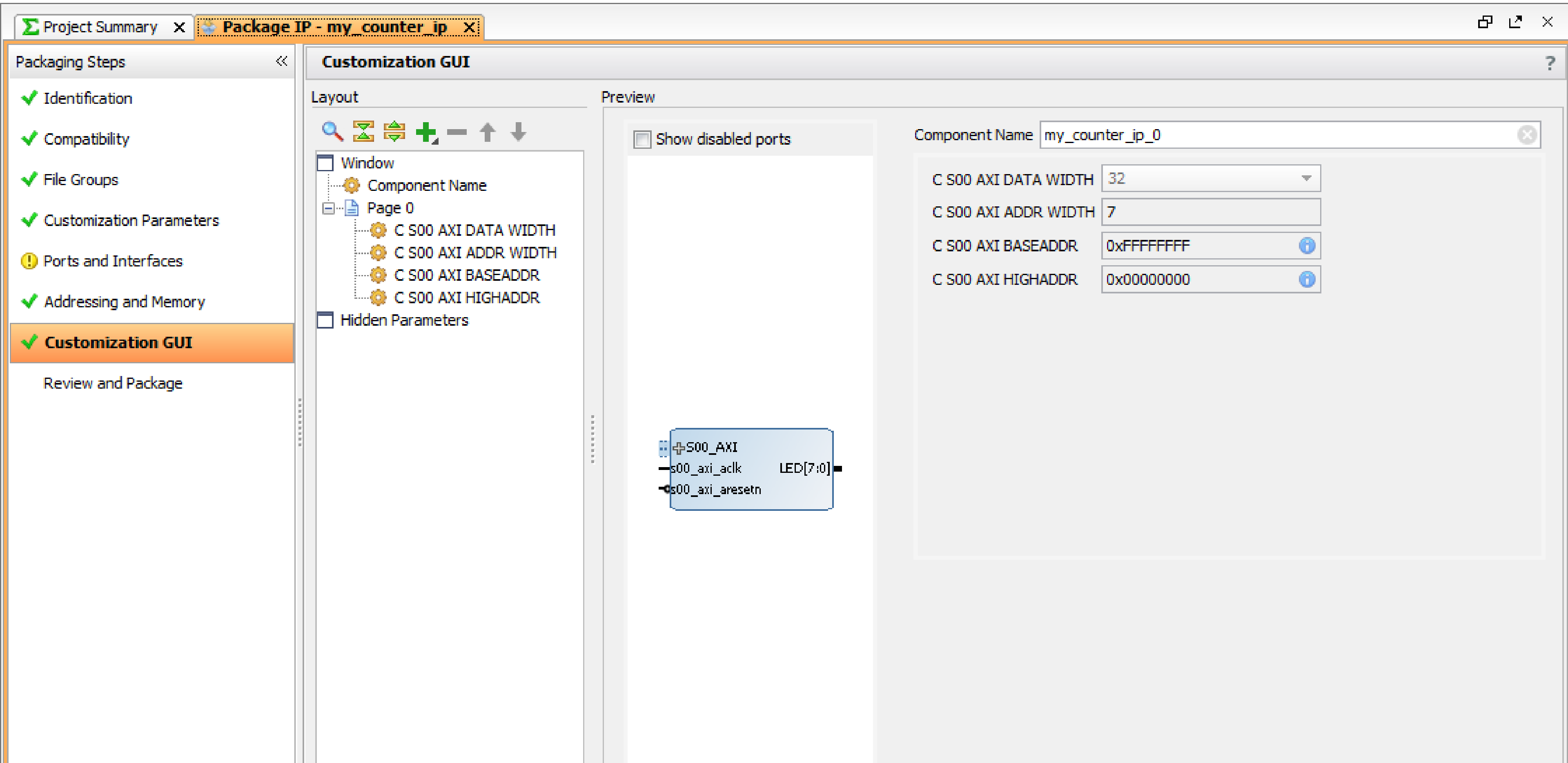


Customization Parameters

| Name | Description | Display Name | Value |
|---|---|---|---|
| Customization Parameters | | | |
| C_S00_AXI_DATA_WIDTH | Width of S_AXI data bus | C S00 AXI DATA WIDTH | 32 |
| C_S00_AXI_ADDR_WIDTH | Width of S_AXI address bus | C S00 AXI ADDR WIDTH | 7 |
| C_S00_AXI_BASEADDR | | C S00 AXI BASEADDR | 0xFFFFFFF |
| C_S00_AXI_HIGHADDR | | C S00 AXI HIGHADDR | 0x000000( |

# Xilinx Vivado – Create and Package Custom IP

- 7.3) Select **Customization GUI**. This is were we get to change our graphical interface.  No changes at this time.


Project Summary
Package IP - my_counter_ip

Packaging Steps
- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ! Ports and Interfaces
- ✓ Addressing and Memory
- ✓ **Customization GUI**
- Review and Package

Customization GUI

Layout
- Window
  - Component Name
  - Page 0
    - C S00 AXI DATA WIDTH
    - C S00 AXI ADDR WIDTH
    - C S00 AXI BASEADDR
    - C S00 AXI HIGHADDR
- Hidden Parameters

Preview

Show disabled ports

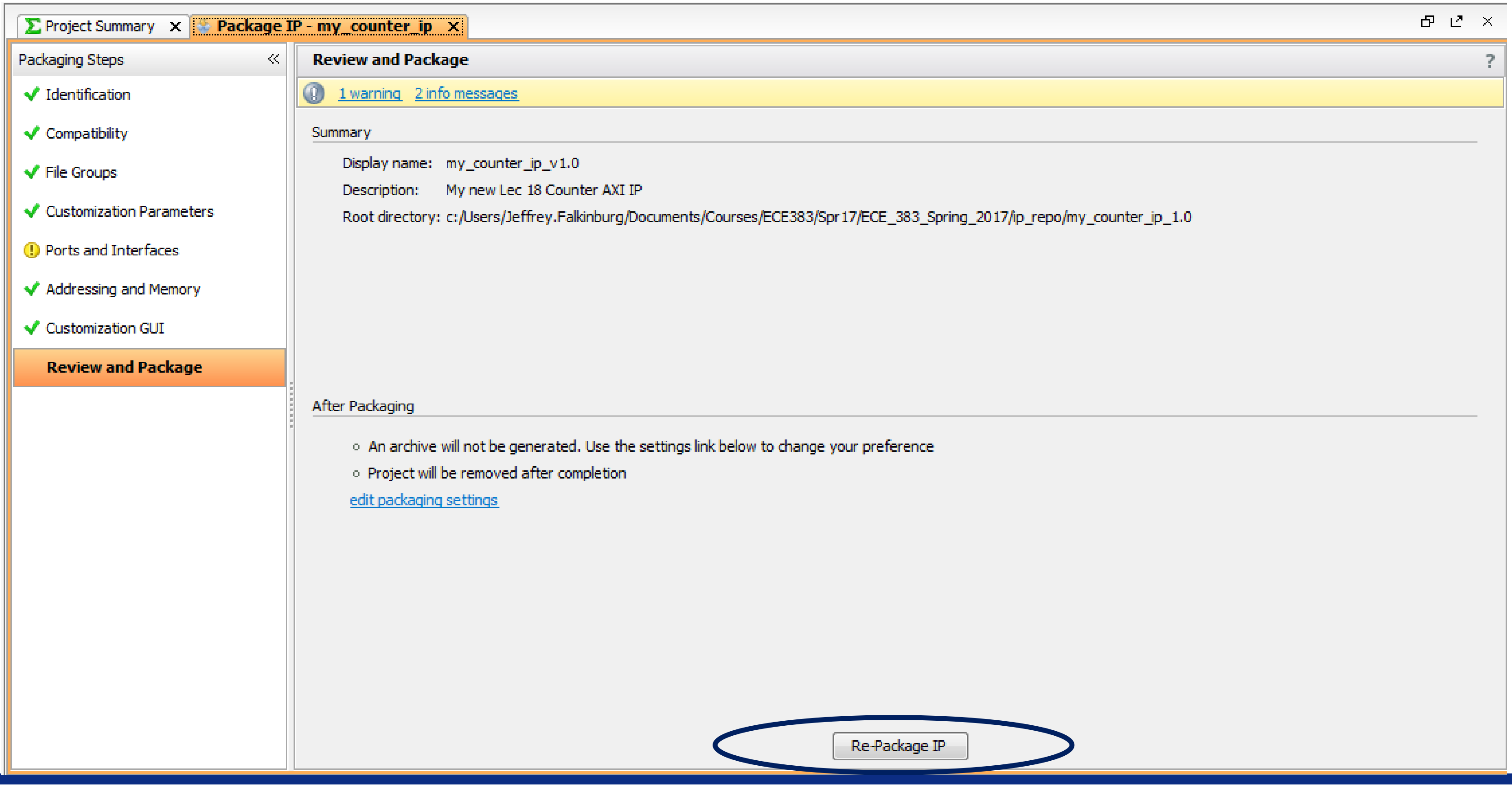
| | |
|---|---|
| Component Name | my_counter_ip_0 |
| C S00 AXI DATA WIDTH | 32 |
| C S00 AXI ADDR WIDTH | 7 |
| C S00 AXI BASEADDR | 0xFFFFFFFF |
| C S00 AXI HIGHADDR | 0x00000000 |

# Xilinx Vivado – Create and Package Custom IP

- 7.3) Select **File Groups**. and select the line for **Merge Changes...**

# Xilinx Vivado – Create and Package Custom IP

- 7.4) Now the core should be complete so select **Review and Package** and click the **Re-package IP** button.



Review and Package

1 warning 2 info messages

Summary

Display name: my_counter_ip_v1.0
Description: My new Lec 18 Counter AXI IP
Root directory: c:/Users/Jeffrey.Falkinburg/Documents/Courses/ECE383/Spr17/ECE_383_Spring_2017/ip_repo/my_counter_ip_1.0

After Packaging

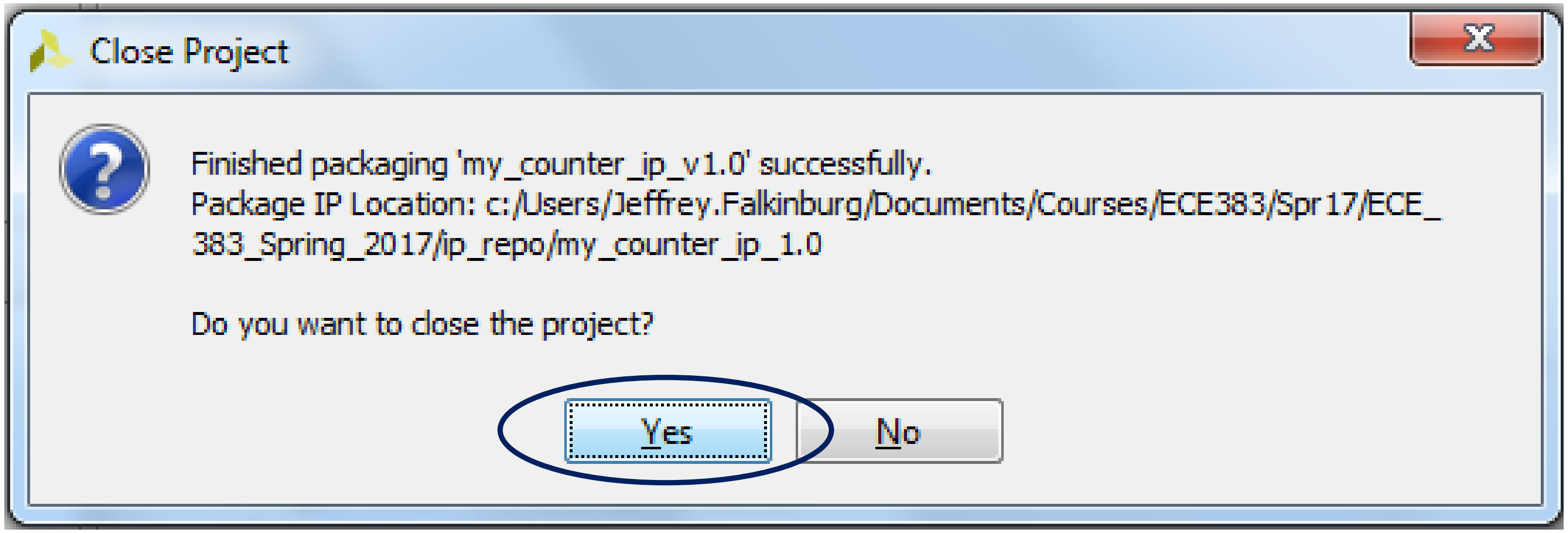
- An archive will not be generated. Use the settings link below to change your preference
- Project will be removed after completion

edit packaging settings

UNITED STATES AIR FORCE ACADEMY

# Xilinx Vivado – Create and Package Custom IP

- 7.5) A popup will ask if you want to close the project, Select **Yes**.



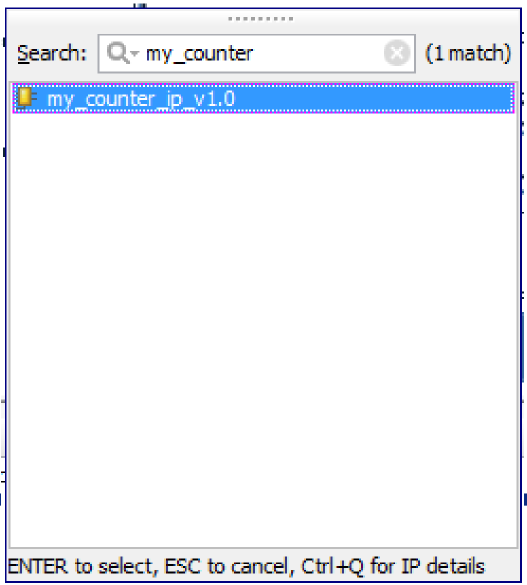
Finished packaging 'my_counter_ip_v1.0' successfully.
Package IP Location: c:/Users/Jeffrey.Falkinburg/Documents/Courses/ECE383/Spr17/ECE_383_Spring_2017/ip_repo/my_counter_ip_1.0

Do you want to close the project?

[✓ Yes] No

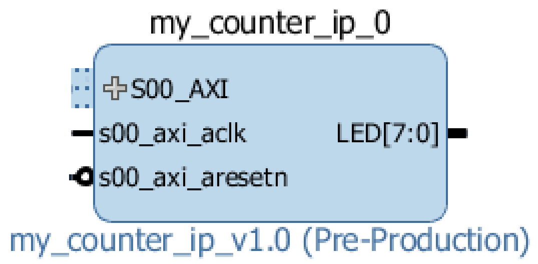
# Xilinx Vivado – Create and Package Custom IP

- 8. Add Custom IP to your design
  - 8.1) In the project manager page of the original window, click **Open Block Design**. This adds a block design to the project.
  - 8.2) Use the **Add IP** button to add your **my_counter**


Search: my_counter (1 match)

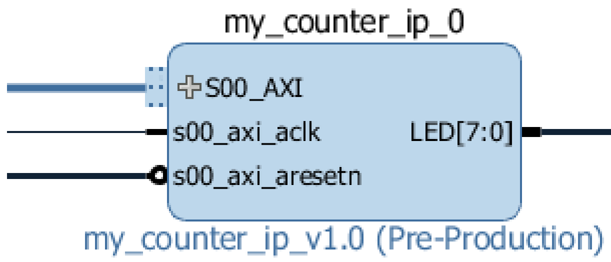
my_counter_ip_v1.0

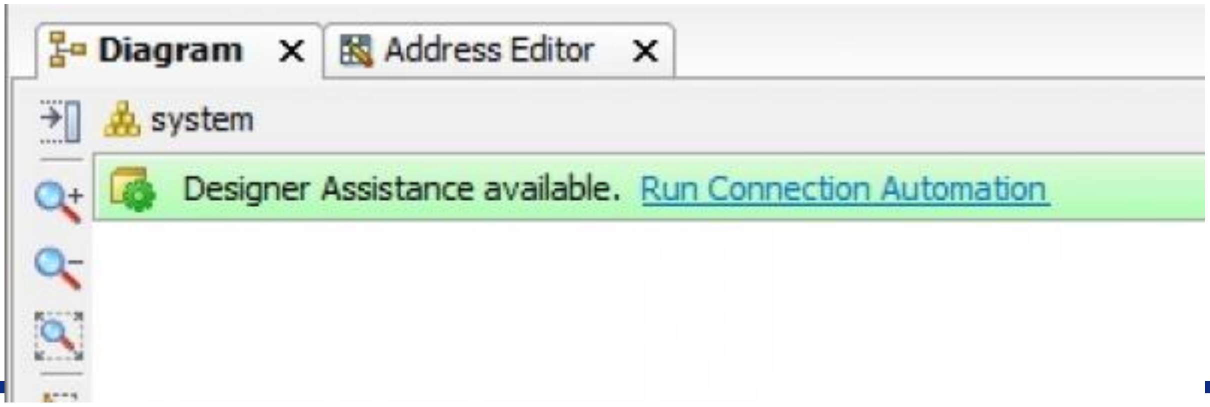
ENTER to select, ESC to cancel, Ctrl+Q for IP details



# Xilinx Vivado – Create and Package Custom IP

- 8. Add Custom IP to your design
  - 8.3) Right click on your **my_counter** output pin **LEDs** and select **Make External** and then run **Connection Automation**





Designer Assistance available. Run Connection Automation

# Xilinx Vivado – Create and Package Custom IP

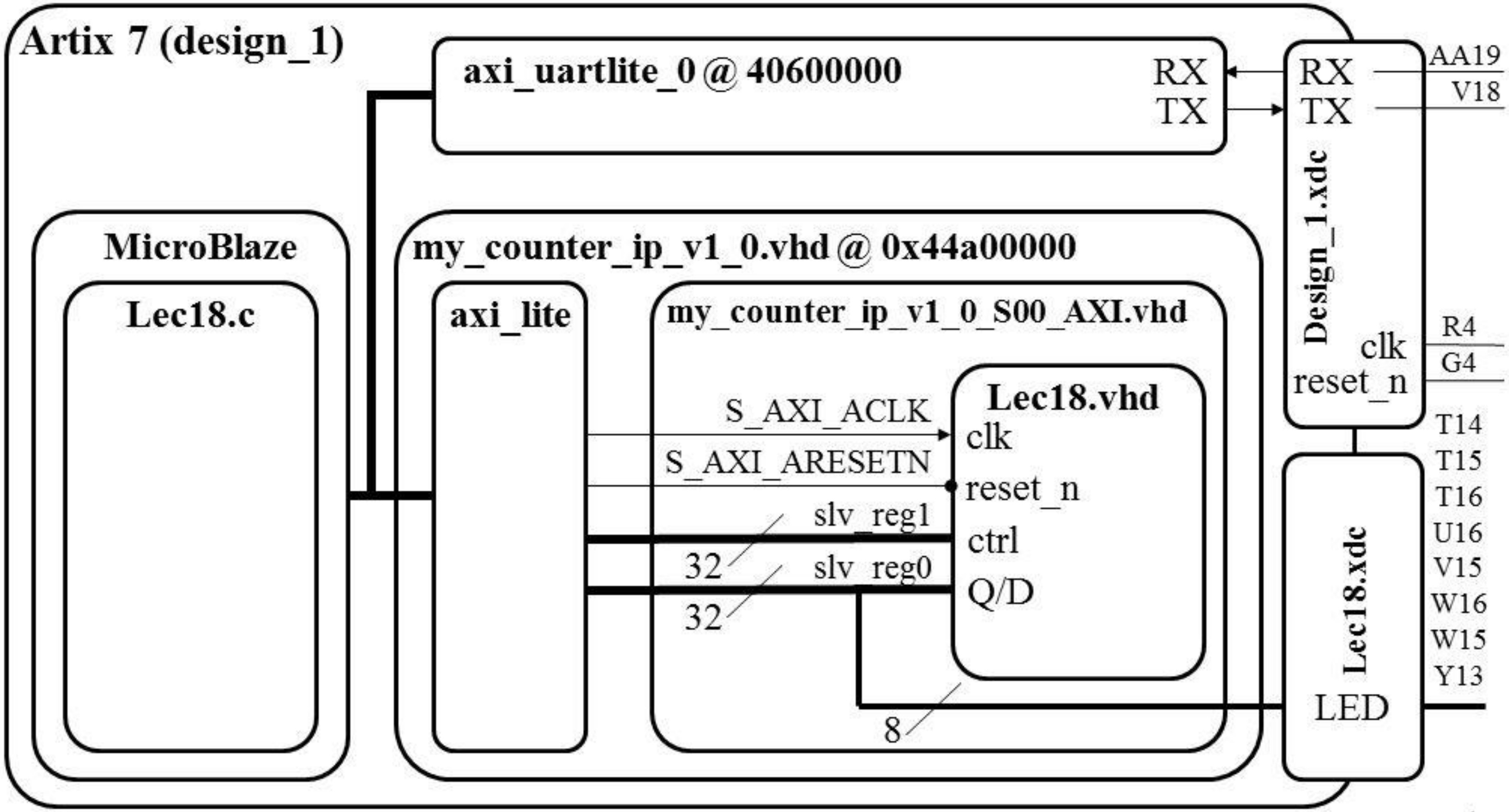
- 8. Add Custom IP to your design
  - 8.5) Your custom IP should now be connected.  Now you need to add a constraints file to add the LED net to the pins on the Artix 7 chip by adding the following lines to the **Lec18.xdc** file.

Add sources → add or create constraints

```
## LEDs
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS25 } [get_ports { LED[0] }]; #IO_L15P_T2_DQS_13 Sch=led[0]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS25 } [get_ports { LED[1] }]; #IO_L15N_T2_DQS_13 Sch=led[1]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS25 } [get_ports { LED[2] }]; #IO_L17P_T2_13 Sch=led[2]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS25 } [get_ports { LED[3] }]; #IO_L17N_T2_13 Sch=led[3]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS25 } [get_ports { LED[4] }]; #IO_L14N_T2_SRCC_13 Sch=led[4]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS25 } [get_ports { LED[5] }]; #IO_L16N_T2_13 Sch=led[5]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS25 } [get_ports { LED[6] }]; #IO_L16P_T2_13 Sch=led[6]
set_property -dict { PACKAGE_PIN Y13   IOSTANDARD LVCMOS25 } [get_ports { LED[7] }]; #IO_L5P_T0_13 Sch=led[7]
```

UNITED STATES
AIR FORCE
ACADEMY

# MicroBlaze + Custom IP



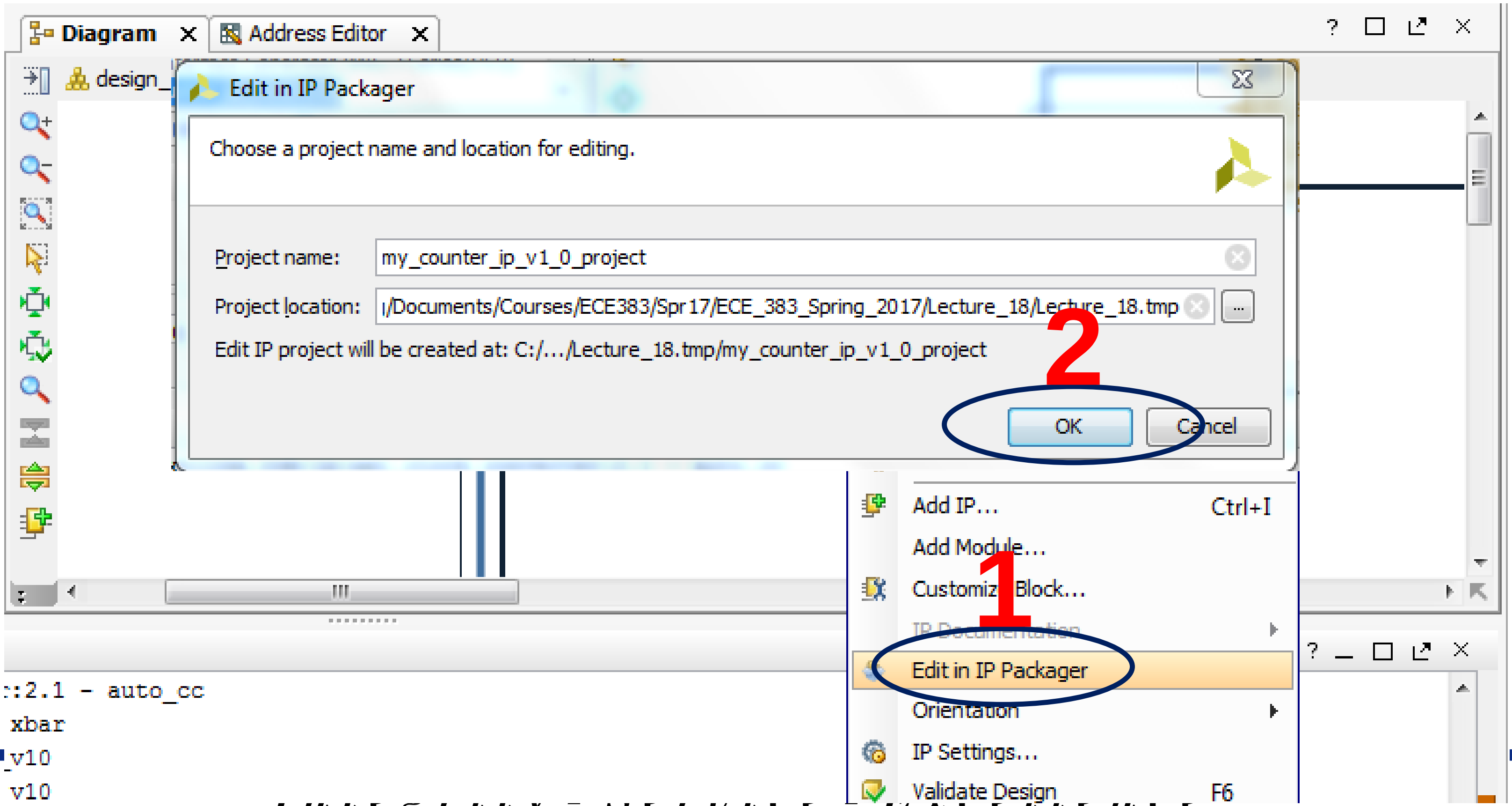
UNITED STATES AIR FORCE ACADEMY

## Updating Custom IP

- For initial class demo: **skip to slide titled “validate design**.”
- For Homework#10, when you need to add the “roll” to your custom hardware, follow the next few slide on “**updating Custom IP**”

# Updating Custom IP

- click on **My_Counter** block, **right-click**, and click **Edit in IP Packager** if you want to modify! (like adding “roll”)



Edit in IP Packager

Choose a project name and location for editing.

Project name: my_counter_ip_v1_0_project

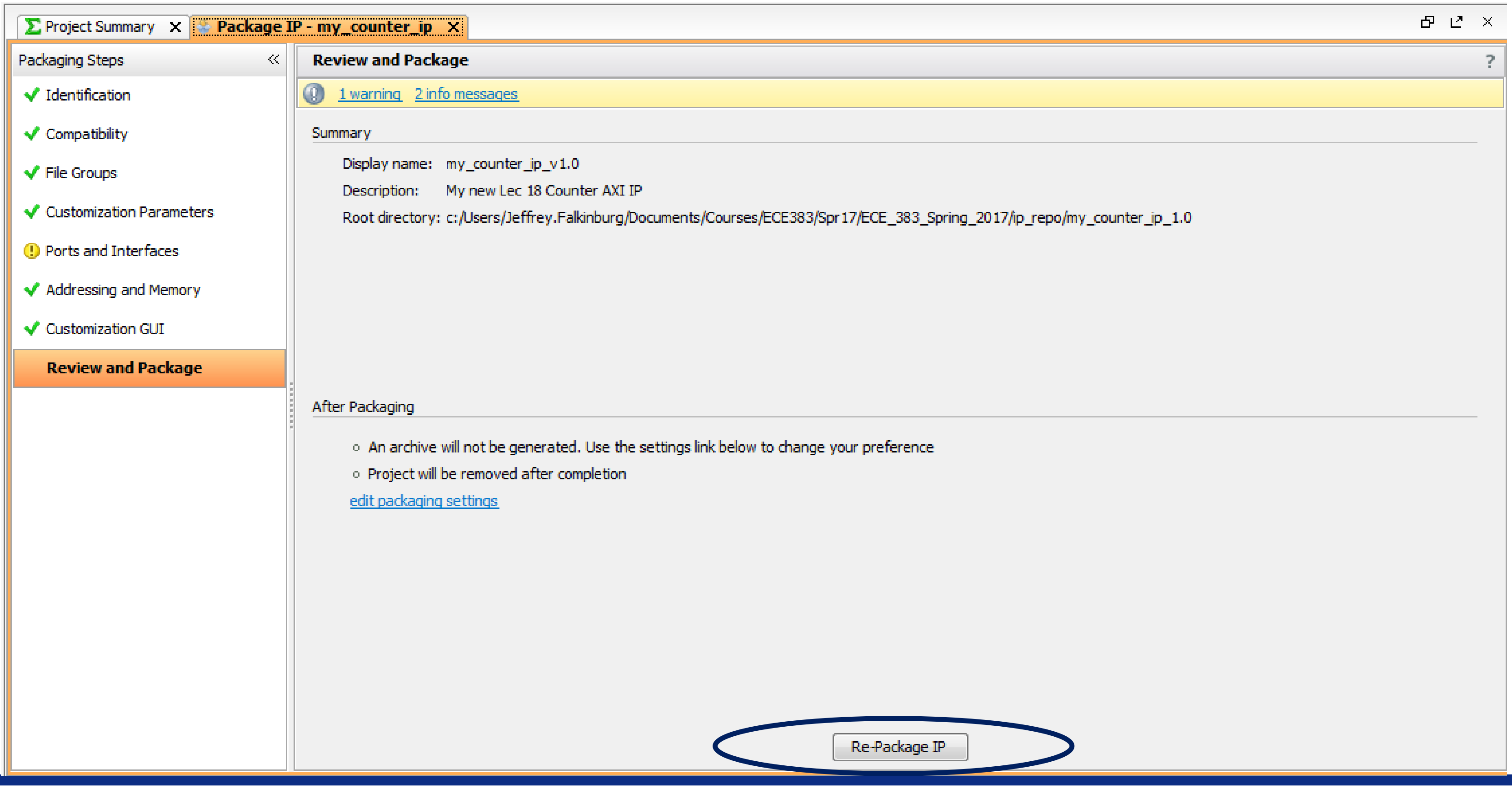
Project location: ı/Documents/Courses/ECE383/Spr17/ECE_383_Spring_2017/Lecture_18/Lecture_18.tmp

Edit IP project will be created at: C:/.../Lecture_18.tmp/my_counter_ip_v1_0_project

```
::2.1 - auto_cc
xbar
_v10
v10
```

# Xilinx Vivado – Create and Package Custom IP

- Now that you updated the core you need to re-select **Review and Package** and click the **Re-package**



Review and Package

Summary

Display name: my_counter_ip_v1.0
Description: My new Lec 18 Counter AXI IP
Root directory: c:/Users/Jeffrey.Falkinburg/Documents/Courses/ECE383/Spr17/ECE_383_Spring_2017/ip_repo/my_counter_ip_1.0

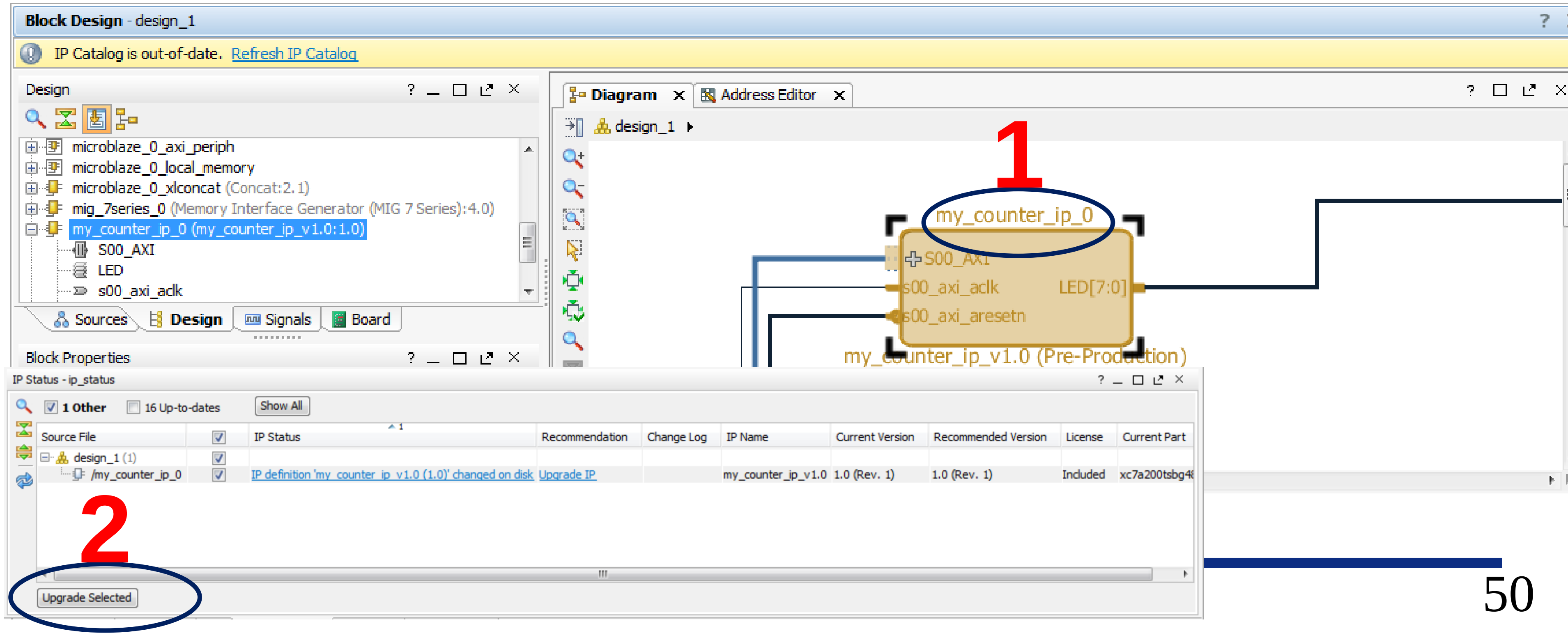
After Packaging

- An archive will not be generated. Use the settings link below to change your preference
- Project will be removed after completion

UNITED STATES AIR FORCE ACADEMY

# Updating Custom IP

- click **Show IP Status** (on yellow bar in block design)
- Select **my_counter** block
- Click **Upgrade Selected** (box at bottom of screen)
- **Ok**, and **Generate**



**Block Design** - design_1

IP Catalog is out-of-date. Refresh IP Catalog

Design

- microblaze_0_axi_periph
- microblaze_0_local_memory
- microblaze_0_xlconcat (Concat:2.1)
- mig_7series_0 (Memory Interface Generator (MIG 7 Series):4.0)
- my_counter_ip_0 (my_counter_ip_v1.0:1.0)
  - S00_AXI
  - LED
  - s00_axi_aclk

IP Status - ip_status

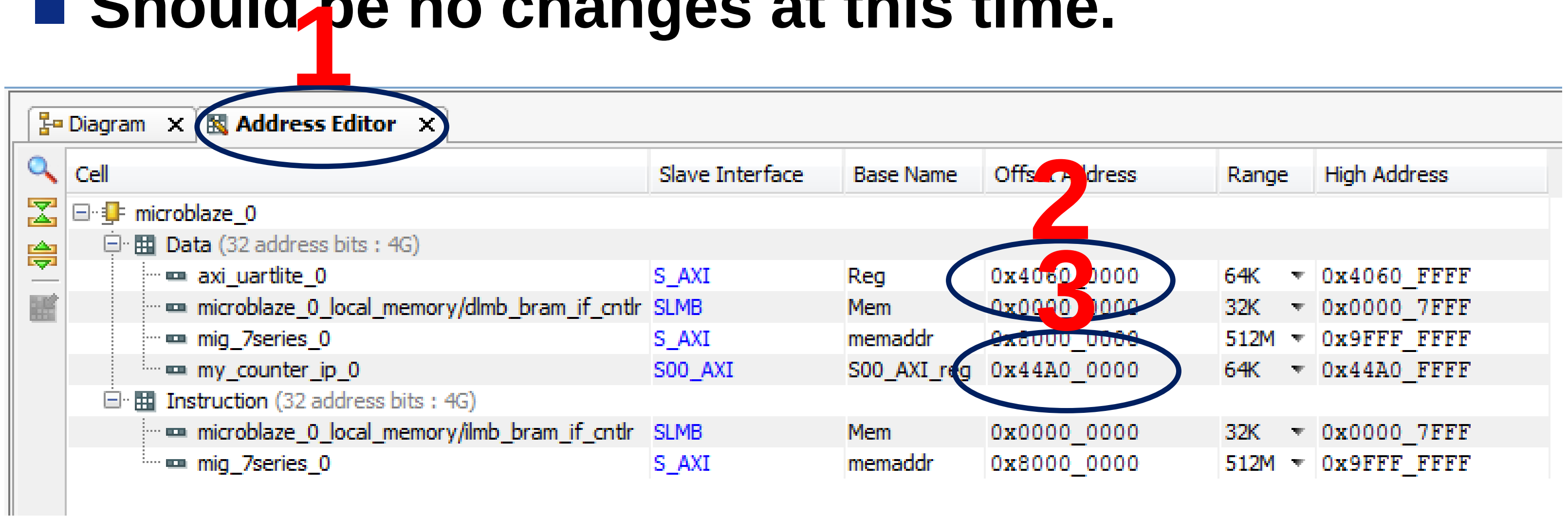
☑ 1 Other ☐ 16 Up-to-dates Show All

| Source File | ✓ | IP Status | Recommendation | Change Log | IP Name | Current Version | Recommended Version | License | Current Part |
|---|---|---|---|---|---|---|---|---|---|
| design_1 (1) | ✓ | | | | | | | | |
| /my_counter_ip_0 | ✓ | IP definition 'my_counter_ip_v1.0 (1.0)' changed on disk | Upgrade IP | | my_counter_ip_v1.0 | 1.0 (Rev. 1) | 1.0 (Rev. 1) | Included | xc7a200tsbg48 |

Upgrade Selected

## Verify Design

- You should verify the addressing for all your design components before continuing.
- Verify that the base addresses are the same addresses used in the template C-code.
- Should be no changes at this time.



| Cell | Slave Interface | Base Name | Offset Address | Range | High Address |
|---|---|---|---|---|---|
| microblaze_0 | | | | | |
| Data (32 address bits : 4G) | | | | | |
| axi_uartlite_0 | S_AXI | Reg | 0x4060_0000 | 64K | 0x4060_FFFF |
| microblaze_0_local_memory/dlmb_bram_if_cntlr | SLMB | Mem | 0x0000_0000 | 32K | 0x0000_7FFF |
| mig_7series_0 | S_AXI | memaddr | 0x8000_0000 | 512M | 0x9FFF_FFFF |
| my_counter_ip_0 | S00_AXI | S00_AXI_reg | 0x44A0_0000 | 64K | 0x44A0_FFFF |
| Instruction (32 address bits : 4G) | | | | | |
| microblaze_0_local_memory/ilmb_bram_if_cntlr | SLMB | Mem | 0x0000_0000 | 32K | 0x0000_7FFF |
| mig_7series_0 | S_AXI | memaddr | 0x8000_0000 | 512M | 0x9FFF_FFFF |

## Validate and Export Design

1. First click **validate** design_1 (click the check mark)
2. Regenerate the design_1 HDL wrapper.

   - Right click **design_1** → "create VHDL wrapper" → OK
3. Finally you need to generate the Generate Design bitstream
4. Take a coffee break while it builds

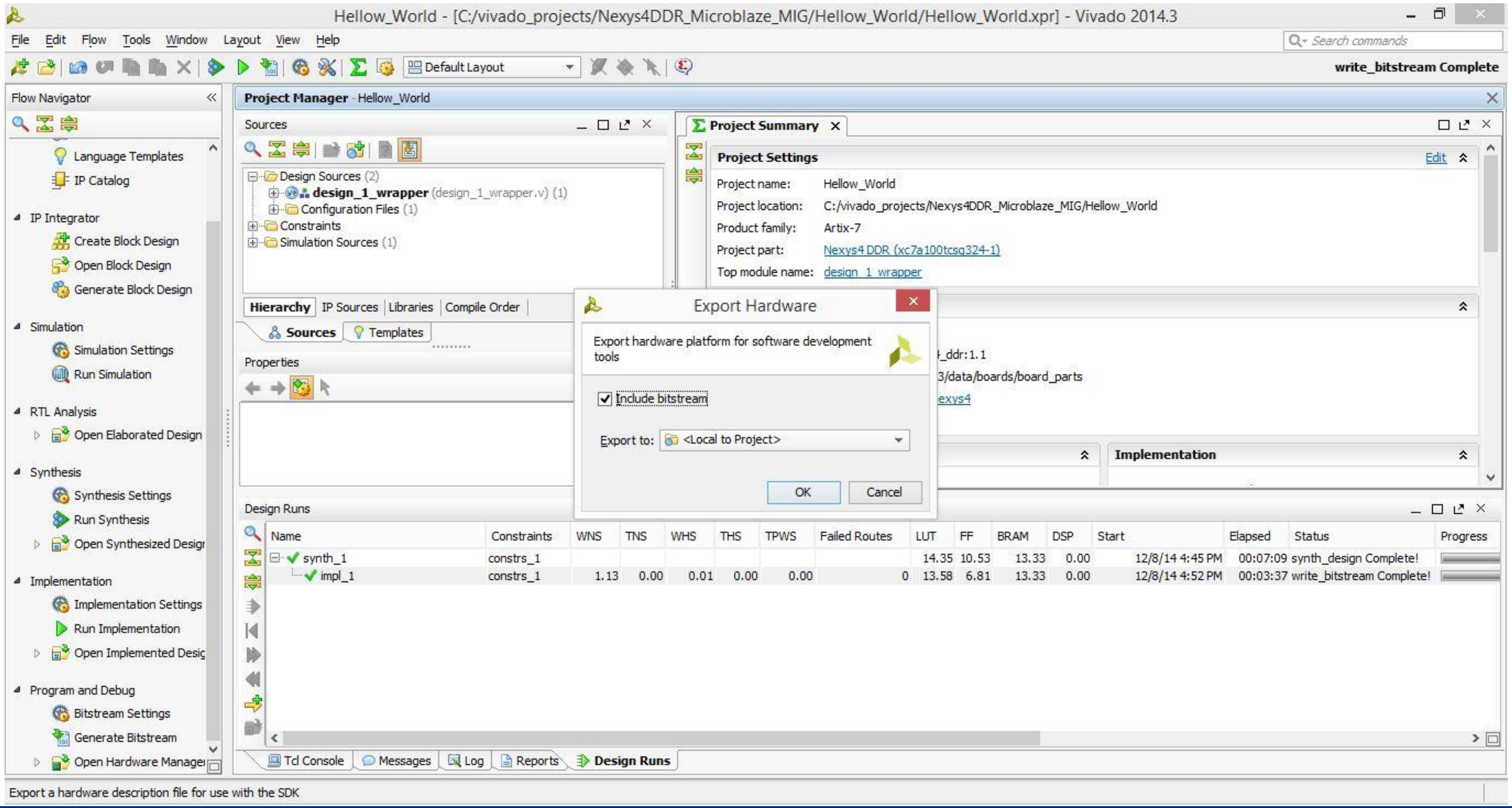
One MIG error is okay (more is not okay)

- apply_bd_automation -rule xilinx.com:bd_rule:mig_7series -config {Board_Interface "ddr3_sdram" } [get_bd_cells mig_7series_0]
  - [BD 41-1273] Error running apply_rule TCL procedure: can't read "board_if": no such variable ::xilinx.com_bd_rule_mig_7series::apply_rule Line 48

UNITED STATES
AIR FORCE
ACADEMY

## Export Design

- Exporting Hardware Design to SDK…be sure to include the bitstream… (File→**Export Hardware**)



Hellow_World - [C:/vivado_projects/Nexys4DDR_Microblaze_MIG/Hellow_World/Hellow_World.xpr] - Vivado 2014.3

write_bitstream Complete

**Project Manager** - Hellow_World

**Project Settings**

| | |
|---|---|
| Project name: | Hellow_World |
| Project location: | C:/vivado_projects/Nexys4DDR_Microblaze_MIG/Hellow_World |
| Product family: | Artix-7 |
| Project part: | Nexys4 DDR (xc7a100tcsg324-1) |
| Top module name: | design_1_wrapper |

Design Sources (2)
- design_1_wrapper (design_1_wrapper.v) (1)
- Configuration Files (1)

Constraints
Simulation Sources (1)

**Export Hardware**

Export hardware platform for software development tools

☑ Include bitstream

Export to: <Local to Project>

OK Cancel

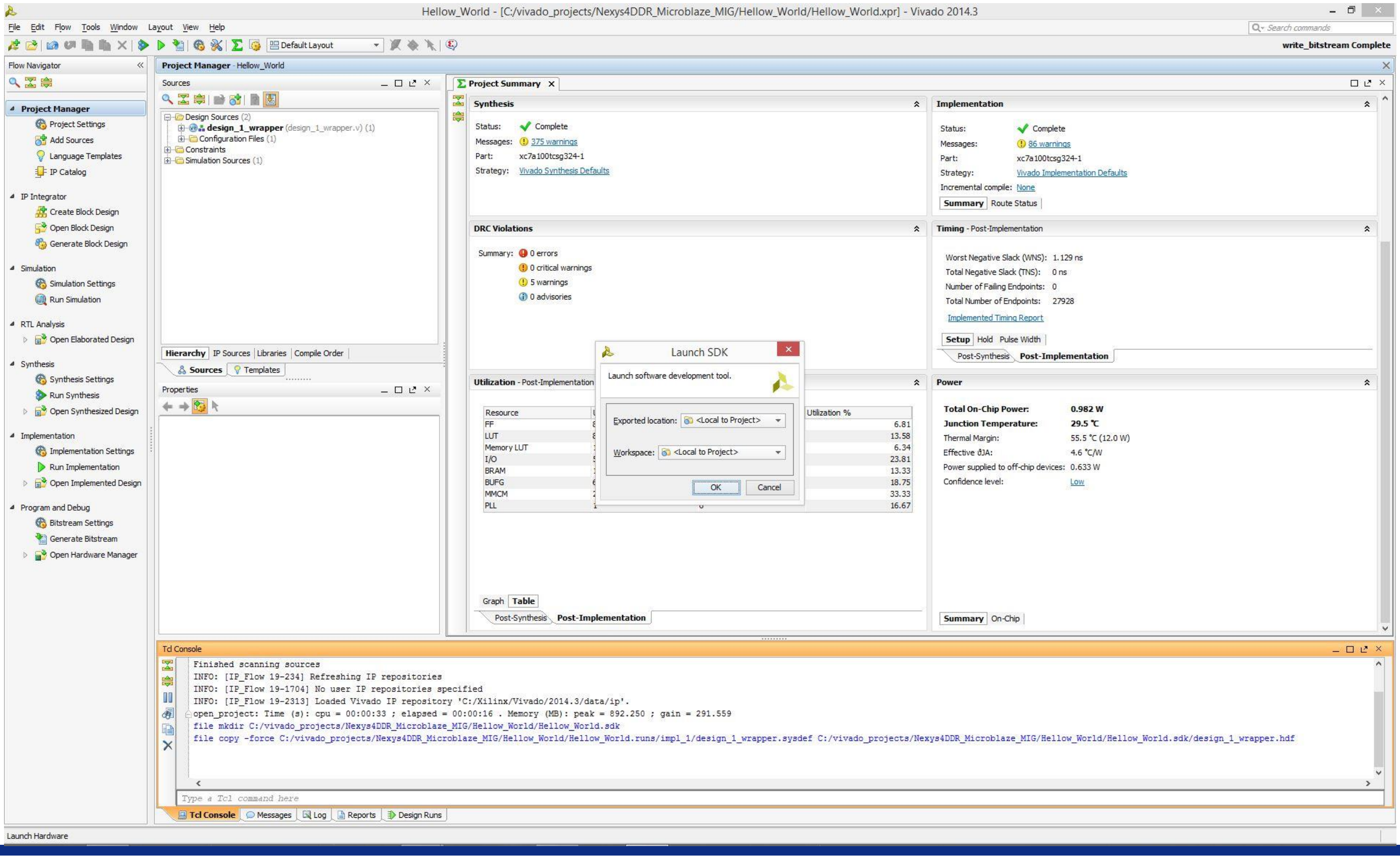
Design Runs

| Name | Constraints | WNS | TNS | WHS | THS | TPWS | Failed Routes | LUT | FF | BRAM | DSP | Start | Elapsed | Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| synth_1 | constrs_1 | | | | | | | 14.35 | 10.53 | 13.33 | 0.00 | 12/8/14 4:45 PM | 00:07:09 | synth_design Complete! |
| impl_1 | constrs_1 | 1.13 | 0.00 | 0.01 | 0.00 | 0.00 | 0 | 13.58 | 6.81 | 13.33 | 0.00 | 12/8/14 4:52 PM | 00:03:37 | write_bitstream Complete! |

Export a hardware description file for use with the SDK

# Launch SDK

- Go to **File** and select **Launch SDK** and click **OK**.



Hellow_World - [C:/vivado_projects/Nexys4DDR_Microblaze_MIG/Hellow_World/Hellow_World.xpr] - Vivado 2014.3

Launch SDK

Launch software development tool.

Exported location: <Local to Project>

Workspace: <Local to Project>

OK Cancel

Synthesis — Status: Complete; Messages: 375 warnings; Part: xc7a100tcsg324-1; Strategy: Vivado Synthesis Defaults

Implementation — Status: Complete; Messages: 86 warnings; Part: xc7a100tcsg324-1; Strategy: Vivado Implementation Defaults; Incremental compile: None

DRC Violations — Summary: 0 errors, 0 critical warnings, 5 warnings, 0 advisories

Timing - Post-Implementation — Worst Negative Slack (WNS): 1.129 ns; Total Negative Slack (TNS): 0 ns; Number of Failing Endpoints: 0; Total Number of Endpoints: 27928

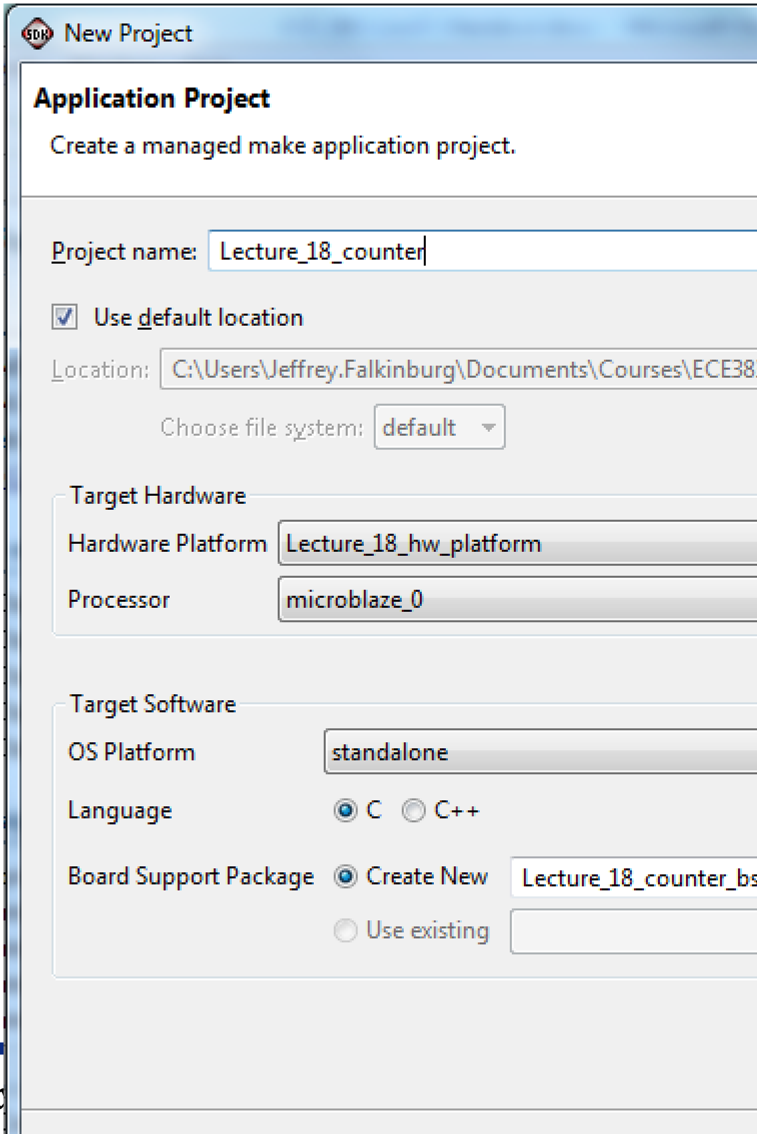
Power — Total On-Chip Power: 0.982 W; Junction Temperature: 29.5 ℃; Thermal Margin: 55.5 ℃ (12.0 W); Effective θJA: 4.6 ℃/W; Power supplied to off-chip devices: 0.633 W; Confidence level: Low

| Resource | Utilization % |
|---|---|
| FF | 6.81 |
| LUT | 13.58 |
| Memory LUT | 6.34 |
| I/O | 23.81 |
| BRAM | 13.33 |
| BUFG | 18.75 |
| MMCM | 33.33 |
| PLL | 16.67 |

```
Finished scanning sources
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2014.3/data/ip'.
open_project: Time (s): cpu = 00:00:33 ; elapsed = 00:00:16 . Memory (MB): peak = 892.250 ; gain = 291.559
file mkdir C:/vivado_projects/Nexys4DDR_Microblaze_MIG/Hellow_World/Hellow_World.sdk
file copy -force C:/vivado_projects/Nexys4DDR_Microblaze_MIG/Hellow_World/Hellow_World.runs/impl_1/design_1_wrapper.sysdef C:/vivado_projects/Nexys4DDR_Microblaze_MIG/Hellow_World/Hellow_World.sdk/design_1_wrapper.hdf
```

# New SDK Project

- File → New → Application Project
- Give it a name → Next



New Project

**Application Project**

Create a managed make application project.

Project name: Lecture_18_counter

[✓] Use default location

Location: C:\Users\Jeffrey.Falkinburg\Documents\Courses\ECE38

Choose file system: default

Target Hardware

Hardware Platform: Lecture_18_hw_platform

Processor: microblaze_0

Target Software

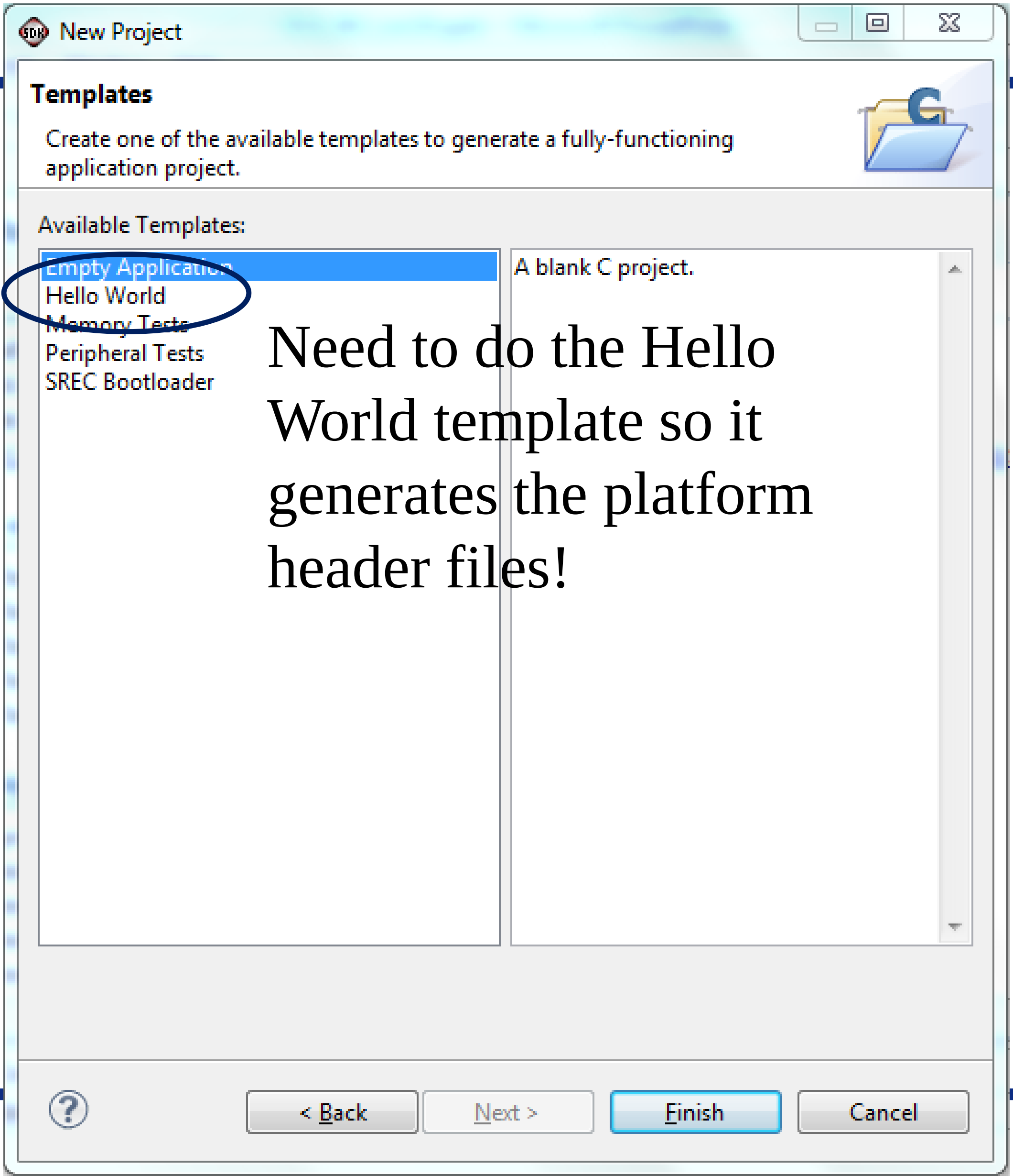
OS Platform: standalone

Language: [● C] ○ C++

Board Support Package: [● Create New] Lecture_18_counter_bs

○ Use existing

# New SDK Project



**Templates**

Create one of the available templates to generate a fully-functioning application project.

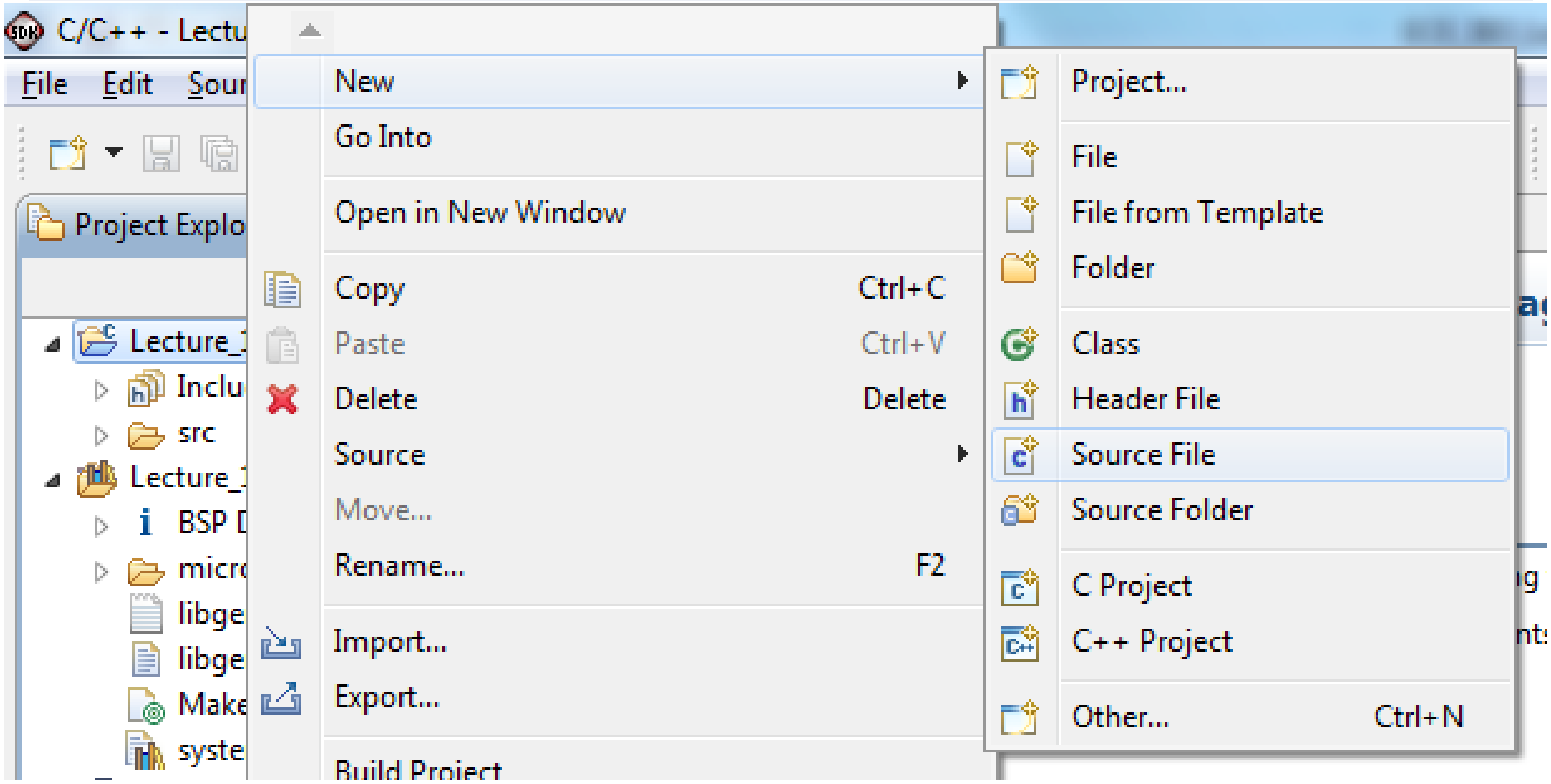
Available Templates:

- Empty Application
- Hello World
- Memory Tests
- Peripheral Tests
- SREC Bootloader

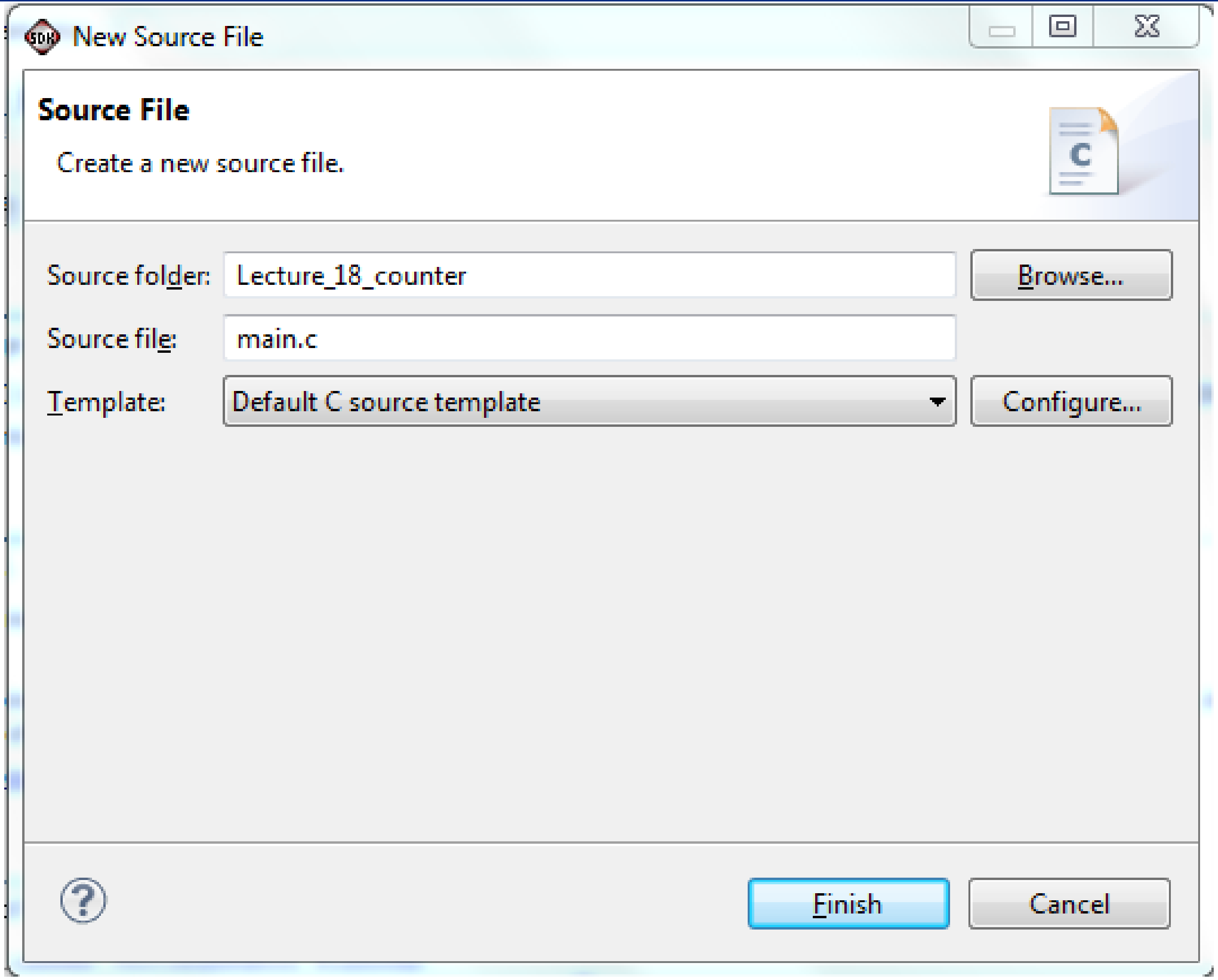
A blank C project.

Need to do the Hello World template so it generates the platform header files!

# New C Source File



# New C Source File


New Source File

Source File

Create a new source file.

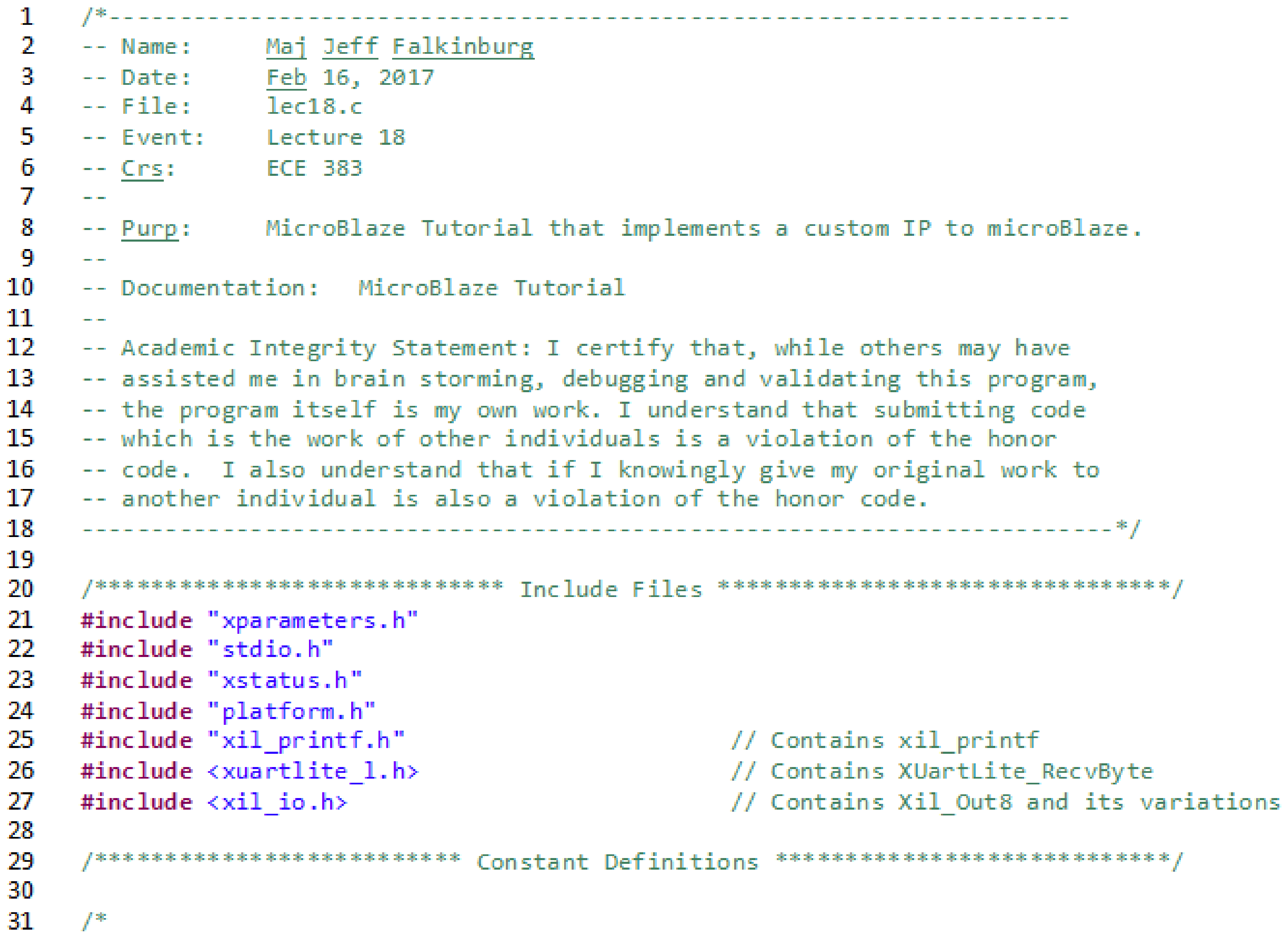
Source folder: Lecture_18_counter

Source file: main.c

Template: Default C source template

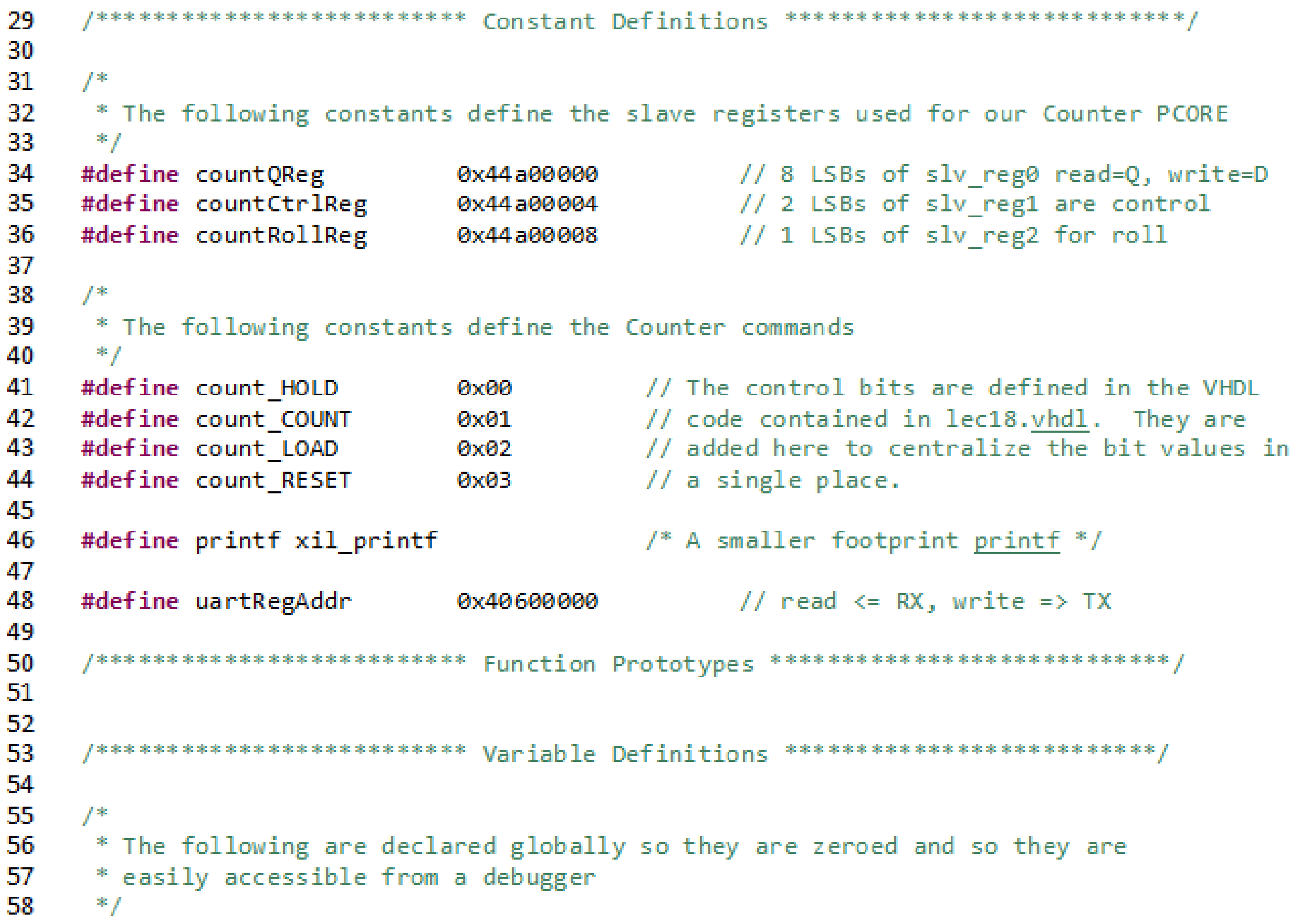
UNITED STATES AIR FORCE ACADEMY

# Lec18.c



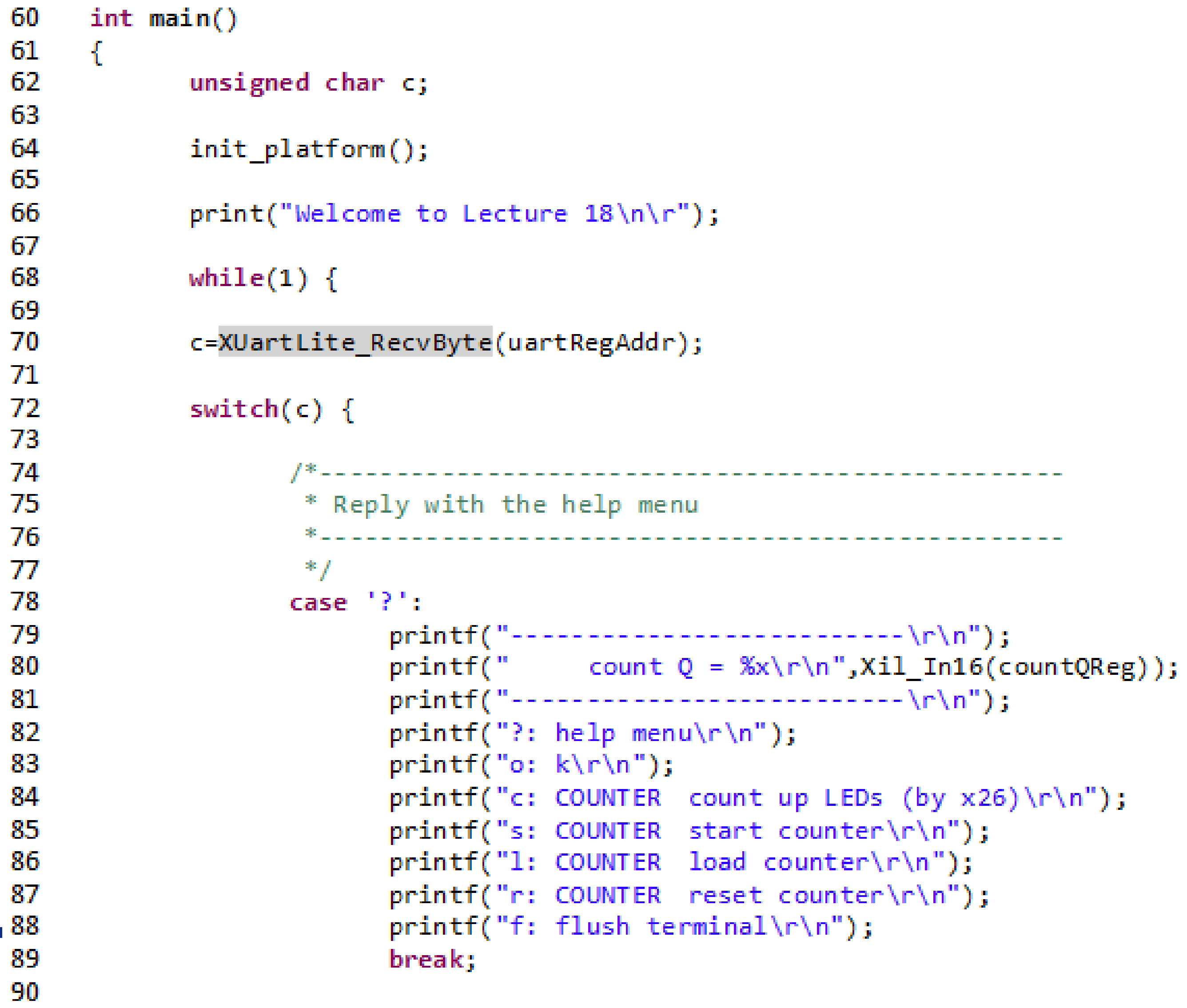
```
/*---------------------------------------------------------------------
-- Name:       Maj Jeff Falkinburg
-- Date:       Feb 16, 2017
-- File:       lec18.c
-- Event:      Lecture 18
-- Crs:        ECE 383
--
-- Purp:       MicroBlaze Tutorial that implements a custom IP to microBlaze.
--
-- Documentation:   MicroBlaze Tutorial
--
-- Academic Integrity Statement: I certify that, while others may have
-- assisted me in brain storming, debugging and validating this program,
-- the program itself is my own work. I understand that submitting code
-- which is the work of other individuals is a violation of the honor
-- code.  I also understand that if I knowingly give my original work to
-- another individual is also a violation of the honor code.
------------------------------------------------------------------------*/

/****************************** Include Files **********************************/
#include "xparameters.h"
#include "stdio.h"
#include "xstatus.h"
#include "platform.h"
#include "xil_printf.h"                             // Contains xil_printf
#include <xuartlite_l.h>                            // Contains XUartLite_RecvByte
#include <xil_io.h>                                 // Contains Xil_Out8 and its variations

/*************************** Constant Definitions ***************************/

/*
```

# Lec18.c



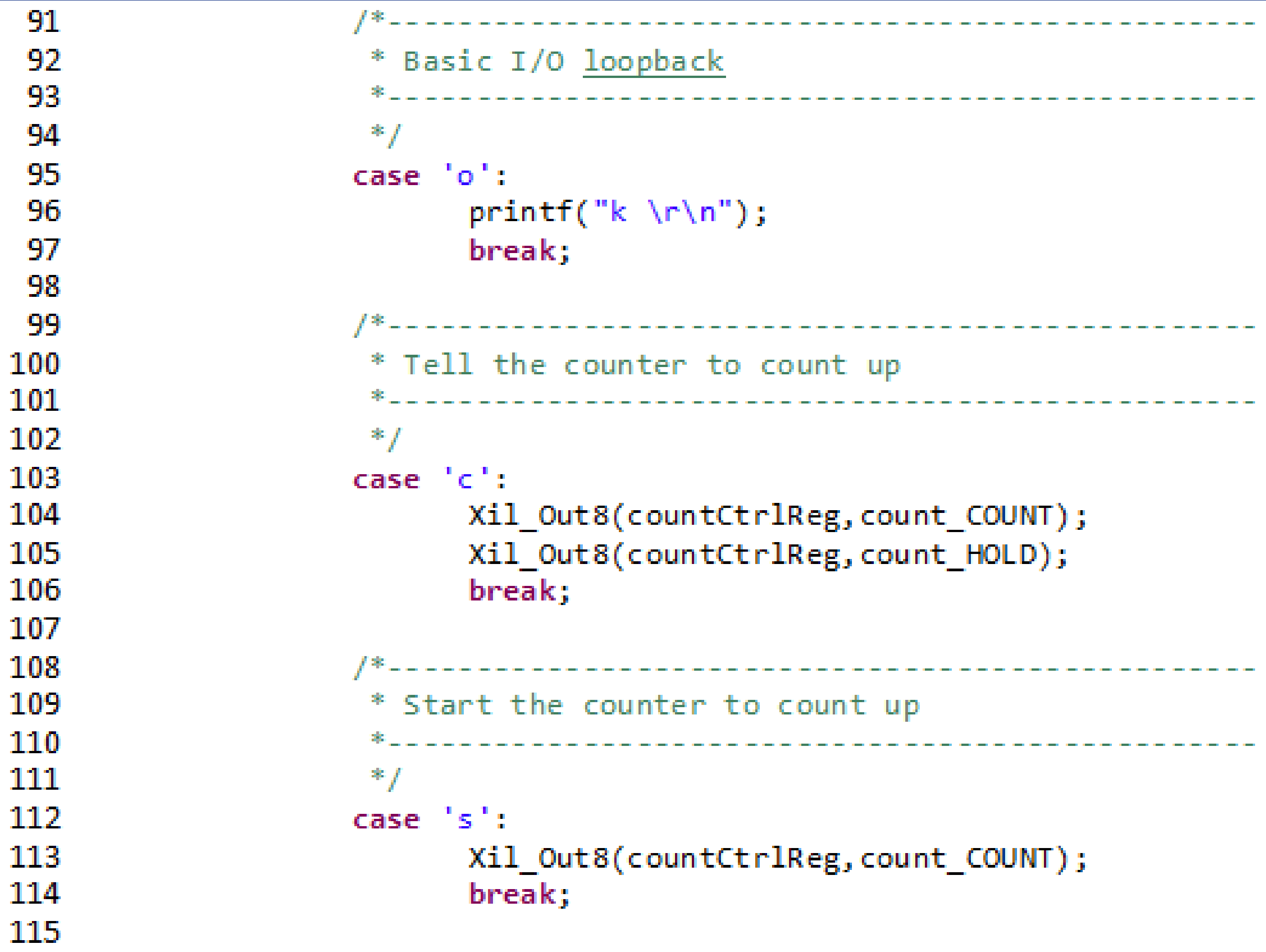
```c
/************************* Constant Definitions ***************************/

/*
 * The following constants define the slave registers used for our Counter PCORE
 */
#define countQReg       0x44a00000          // 8 LSBs of slv_reg0 read=Q, write=D
#define countCtrlReg    0x44a00004          // 2 LSBs of slv_reg1 are control
#define countRollReg    0x44a00008          // 1 LSBs of slv_reg2 for roll

/*
 * The following constants define the Counter commands
 */
#define count_HOLD      0x00          // The control bits are defined in the VHDL
#define count_COUNT     0x01          // code contained in lec18.vhdl.  They are
#define count_LOAD      0x02          // added here to centralize the bit values in
#define count_RESET     0x03          // a single place.

#define printf xil_printf             /* A smaller footprint printf */

#define uartRegAddr     0x40600000          // read <= RX, write => TX

/************************* Function Prototypes ****************************/


/************************* Variable Definitions ***************************/

/*
 * The following are declared globally so they are zeroed and so they are
 * easily accessible from a debugger
 */
```

# Lec18.c



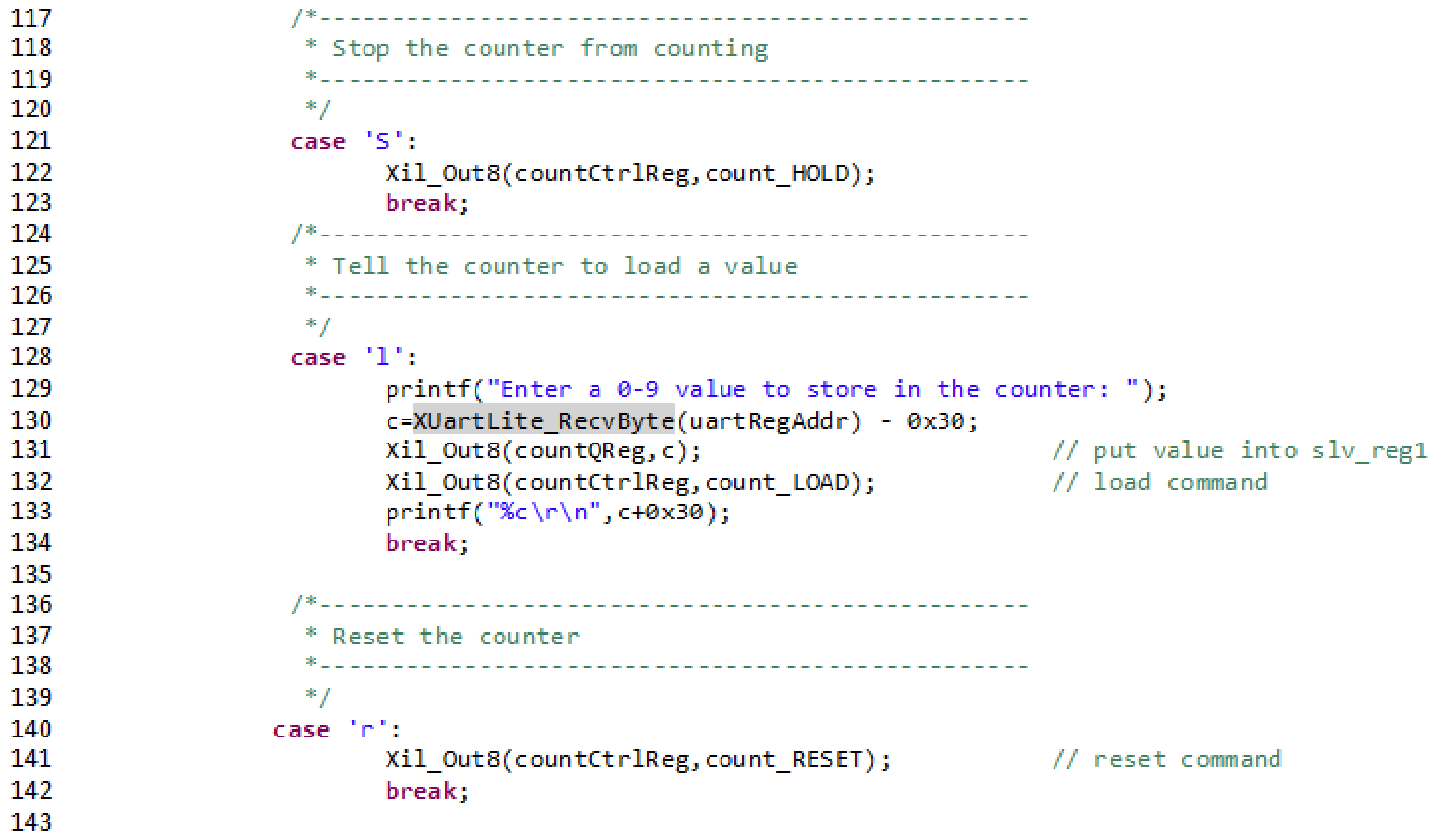
```c
int main()
{
    unsigned char c;

    init_platform();

    print("Welcome to Lecture 18\n\r");

    while(1) {

    c=XUartLite_RecvByte(uartRegAddr);

    switch(c) {

        /*-------------------------------------------------
         * Reply with the help menu
         *-------------------------------------------------
         */
        case '?':
            printf("--------------------------\r\n");
            printf("     count Q = %x\r\n",Xil_In16(countQReg));
            printf("--------------------------\r\n");
            printf("?: help menu\r\n");
            printf("o: k\r\n");
            printf("c: COUNTER  count up LEDs (by x26)\r\n");
            printf("s: COUNTER  start counter\r\n");
            printf("l: COUNTER  load counter\r\n");
            printf("r: COUNTER  reset counter\r\n");
            printf("f: flush terminal\r\n");
            break;
```

# Lec18.c



```c
/*---------------------------------------------------
 * Basic I/O loopback
 *---------------------------------------------------
 */
case 'o':
        printf("k \r\n");
        break;

/*---------------------------------------------------
 * Tell the counter to count up
 *---------------------------------------------------
 */
case 'c':
        Xil_Out8(countCtrlReg,count_COUNT);
        Xil_Out8(countCtrlReg,count_HOLD);
        break;

/*---------------------------------------------------
 * Start the counter to count up
 *---------------------------------------------------
 */
case 's':
        Xil_Out8(countCtrlReg,count_COUNT);
        break;
```

# Lec18.c



```
/*----------------------------------------------------
 * Stop the counter from counting
 *----------------------------------------------------
 */
case 'S':
        Xil_Out8(countCtrlReg,count_HOLD);
        break;
/*----------------------------------------------------
 * Tell the counter to load a value
 *----------------------------------------------------
 */
case 'l':
        printf("Enter a 0-9 value to store in the counter: ");
        c=XUartLite_RecvByte(uartRegAddr) - 0x30;
        Xil_Out8(countQReg,c);                        // put value into slv_reg1
        Xil_Out8(countCtrlReg,count_LOAD);            // load command
        printf("%c\r\n",c+0x30);
        break;

/*----------------------------------------------------
 * Reset the counter
 *----------------------------------------------------
 */
case 'r':
        Xil_Out8(countCtrlReg,count_RESET);           // reset command
        break;

```

# Lec18.c

```c
             /*-----------------------------------------------
              * Clear the terminal window
              *-----------------------------------------------
              */
            case 'f':
                    for (c=0; c<40; c++) printf("\r\n");
                    break;

             /*-----------------------------------------------
              * Unknown character was
              *-----------------------------------------------
              */
             default:
                    printf("unrecognized character: %c\r\n",c);
                    break;
        } // end case

    }

    cleanup_platform();

    return 0;

} // end main
```

UNITED STATES AIR FORCE ACADEMY

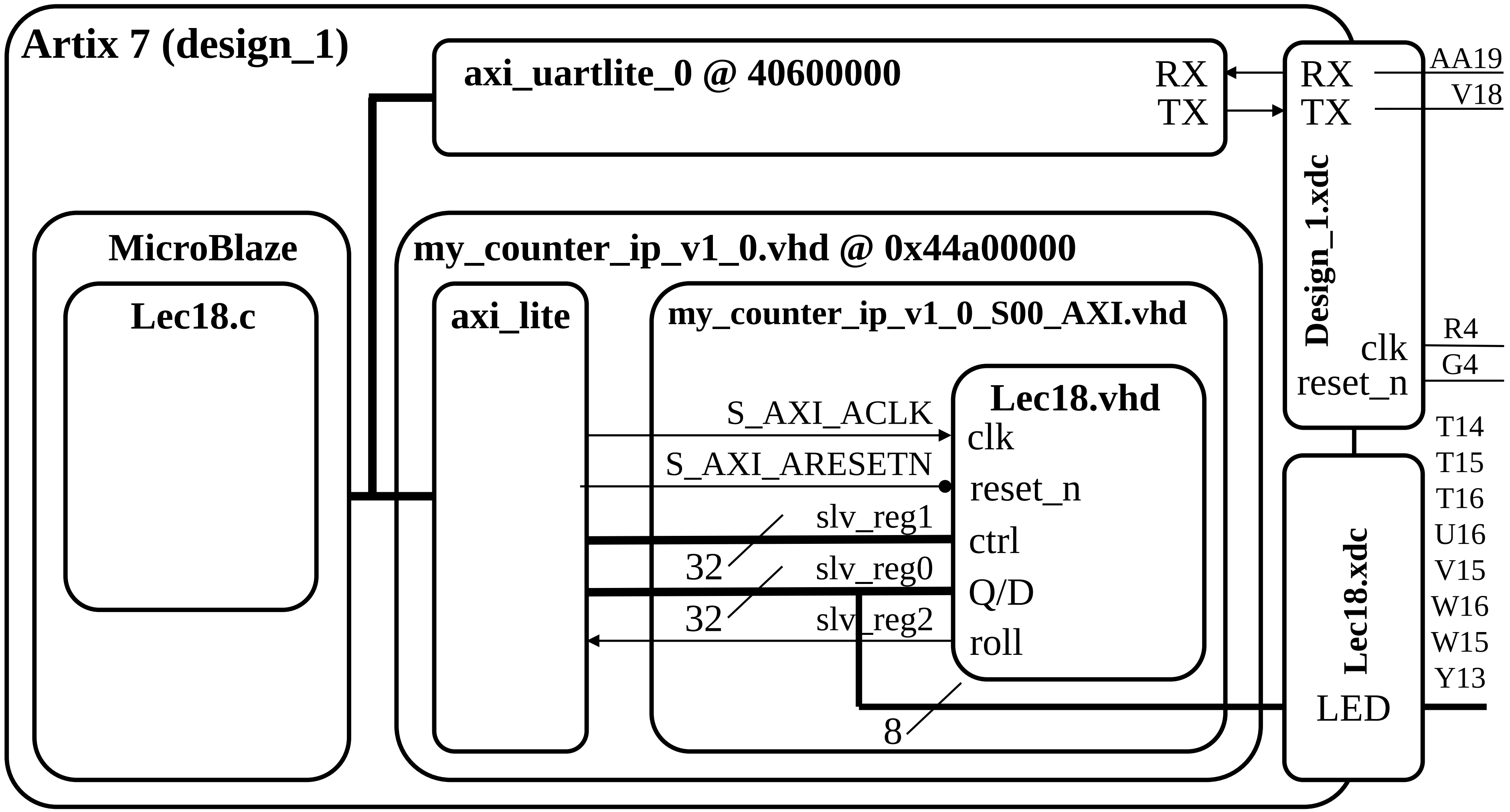
# Part 2: Software Questions/ Notes related to handout

- Why doesn't the 'c' command cause the counter to count up by 1?
- On line 130, why did I subtract 0x30?
- After loading the counter on line 132, something should be done that is missing.
- What line of VHDL code in my_counter_ip_v1_0_S00_AXI.vhd is "activated" when line 80 executes?
- What line of VHDL code in my_counter_ip_v1_0_S00_AXI.vhd is "activated" when line 141 executes?
- What line of VHDL code in lec18.vhdl "activated" when line 141 executes?
- What appears to be the naming convention for hardware registers?

UNITED STATES AIR FORCE ACADEMY

# MicroBlaze + Custom IP

## Now add “roll” for HW#10



## Add C code to Source File

UNITED STATES AIR FORCE ACADEMY

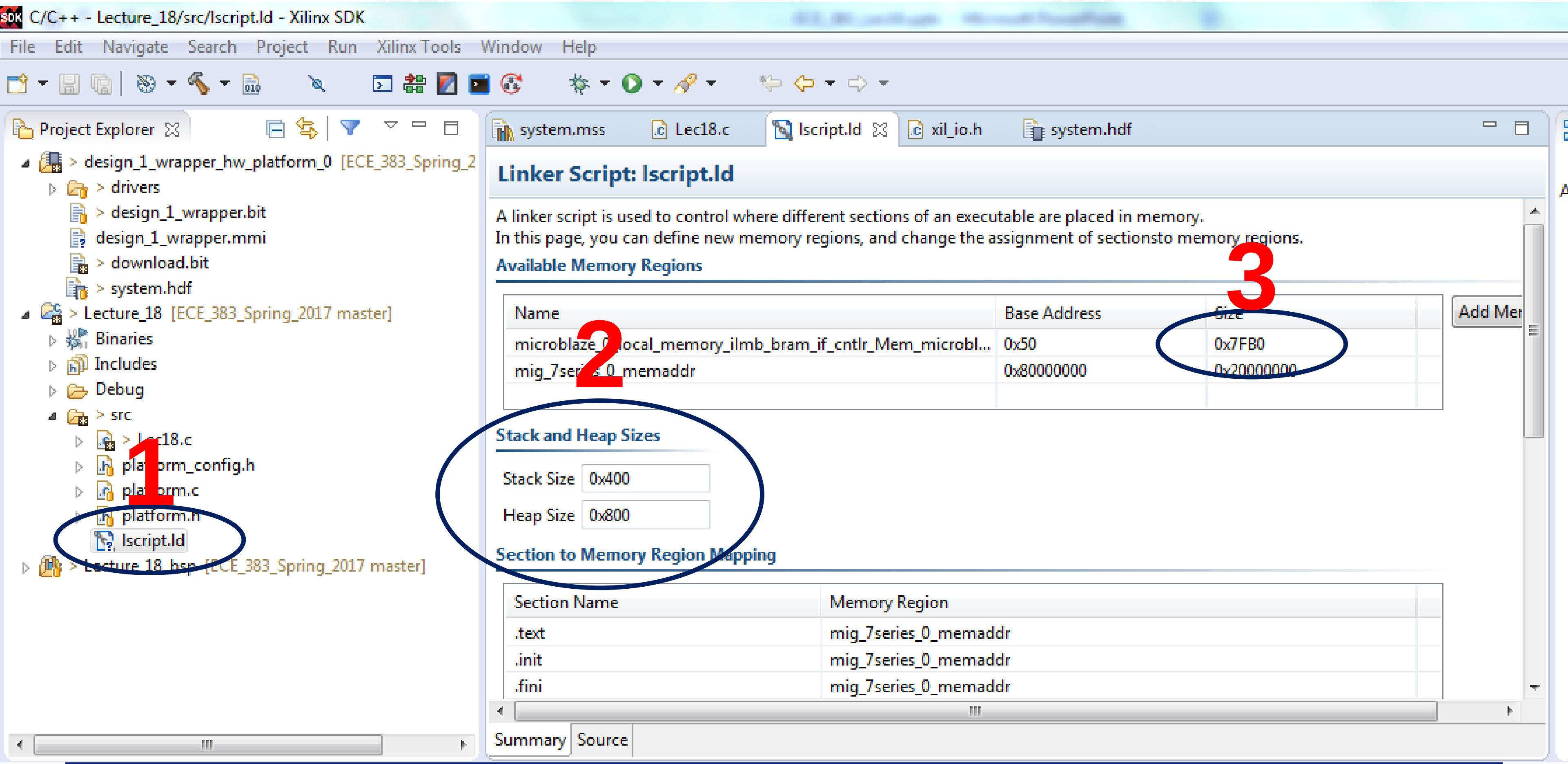
## Xilinx Software Development Kit - SDK

- In the SDK environment, you program the hardware built in the previous step.
- The key concept here is that the peripheral defined in Vivado design are accessible through the slave registers as memory mapped devices.
- Verify your my_counter_ip_v1_0 Base Address in system.hdf file is assigned to be 0x44a00000.
- In the my_counter_ip_v1_0_S00_AXI.vhdl file, I (arbitrarily) assigned counter ports to slave register according to the table below.

| Signal | direction | Slave Register | Address |
|---|---|---|---|
| D | Input | slv_reg0(7 downto 0) | 0x44a00000 |
| ctrl | Input | slv_reg1(1 downto 0) | 0x44a00004 |
| Q | Output | slv_reg0(7 downto 0) | 0x44a00000 |

# Increase Stack and Heap Size

- You may have to add more Stack and Heap to your



C/C++ - Lecture_18/src/lscript.ld - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Tools Window Help

Project Explorer

- design_1_wrapper_hw_platform_0 [ECE_383_Spring_2
  - drivers
  - design_1_wrapper.bit
  - design_1_wrapper.mmi
  - download.bit
  - system.hdf
- Lecture_18 [ECE_383_Spring_2017 master]
  - Binaries
  - Includes
  - Debug
  - src
    - Lec18.c
    - platform_config.h
    - platform.c
    - platform.h
    - lscript.ld
- Lecture_18_bsp [ECE_383_Spring_2017 master]

system.mss | Lec18.c | lscript.ld | xil_io.h | system.hdf

**Linker Script: lscript.ld**

A linker script is used to control where different sections of an executable are placed in memory.
In this page, you can define new memory regions, and change the assignment of sectionsto memory regions.

**Available Memory Regions**

| Name | Base Address | Size |
|---|---|---|
| microblaze_0_local_memory_ilmb_bram_if_cntlr_Mem_microbl... | 0x50 | 0x7FB0 |
| mig_7series_0_memaddr | 0x80000000 | 0x20000000 |

**Stack and Heap Sizes**

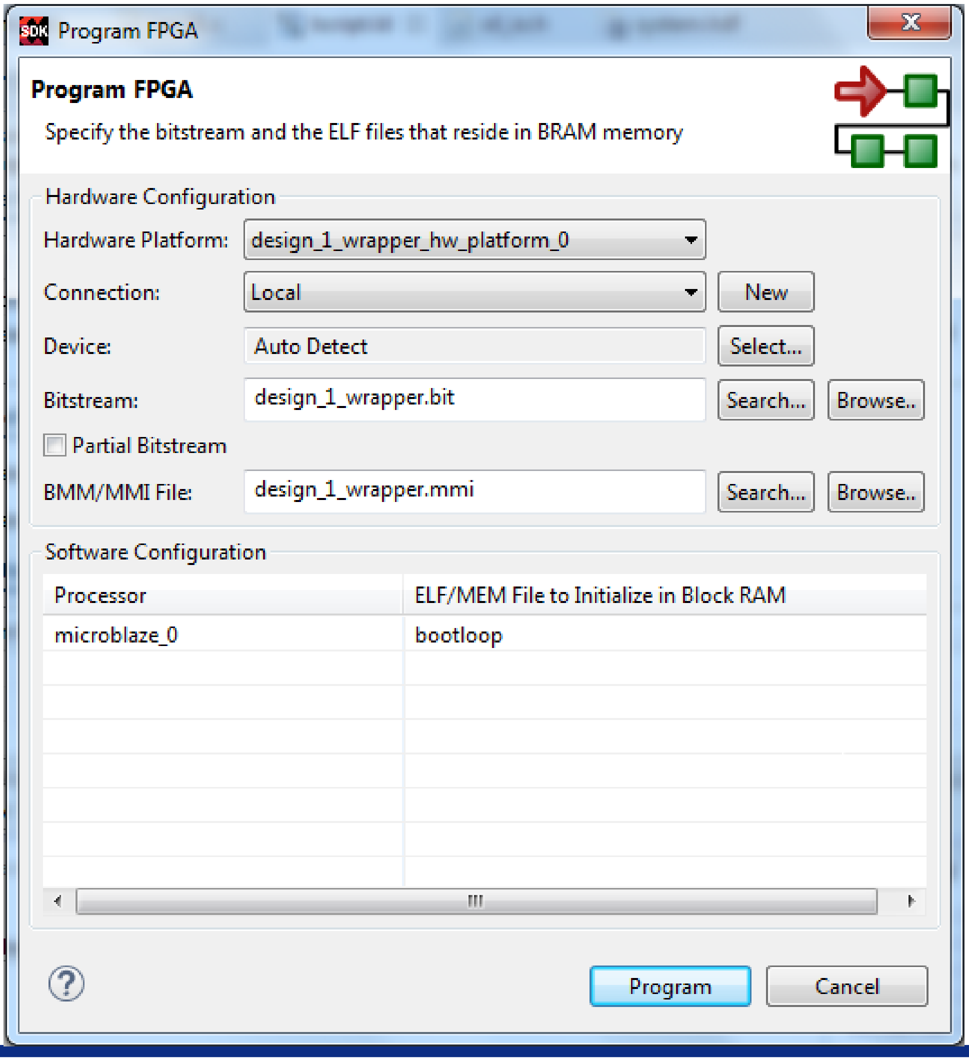
Stack Size 0x400

Heap Size 0x800

**Section to Memory Region Mapping**

| Section Name | Memory Region |
|---|---|
| .text | mig_7series_0_memaddr |
| .init | mig_7series_0_memaddr |
| .fini | mig_7series_0_memaddr |

# Build and Export to FPGA



Program FPGA

**Program FPGA**

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Platform: design_1_wrapper_hw_platform_0

Connection: Local

Device: Auto Detect

Bitstream: design_1_wrapper.bit

☐ Partial Bitstream

BMM/MMI File: design_1_wrapper.mmi

Software Configuration

| Processor | ELF/MEM File to Initialize in Block RAM |
|---|---|
| microblaze_0 | bootloop |

# Test program using UART

- Run Configuration Settings for STDIO Connection
  - From the Project Explorer panel, right click on the **Lecture18_counter** project folder. At the bottom of the drop down list, select **Run As** and then select **Run Configurations**.

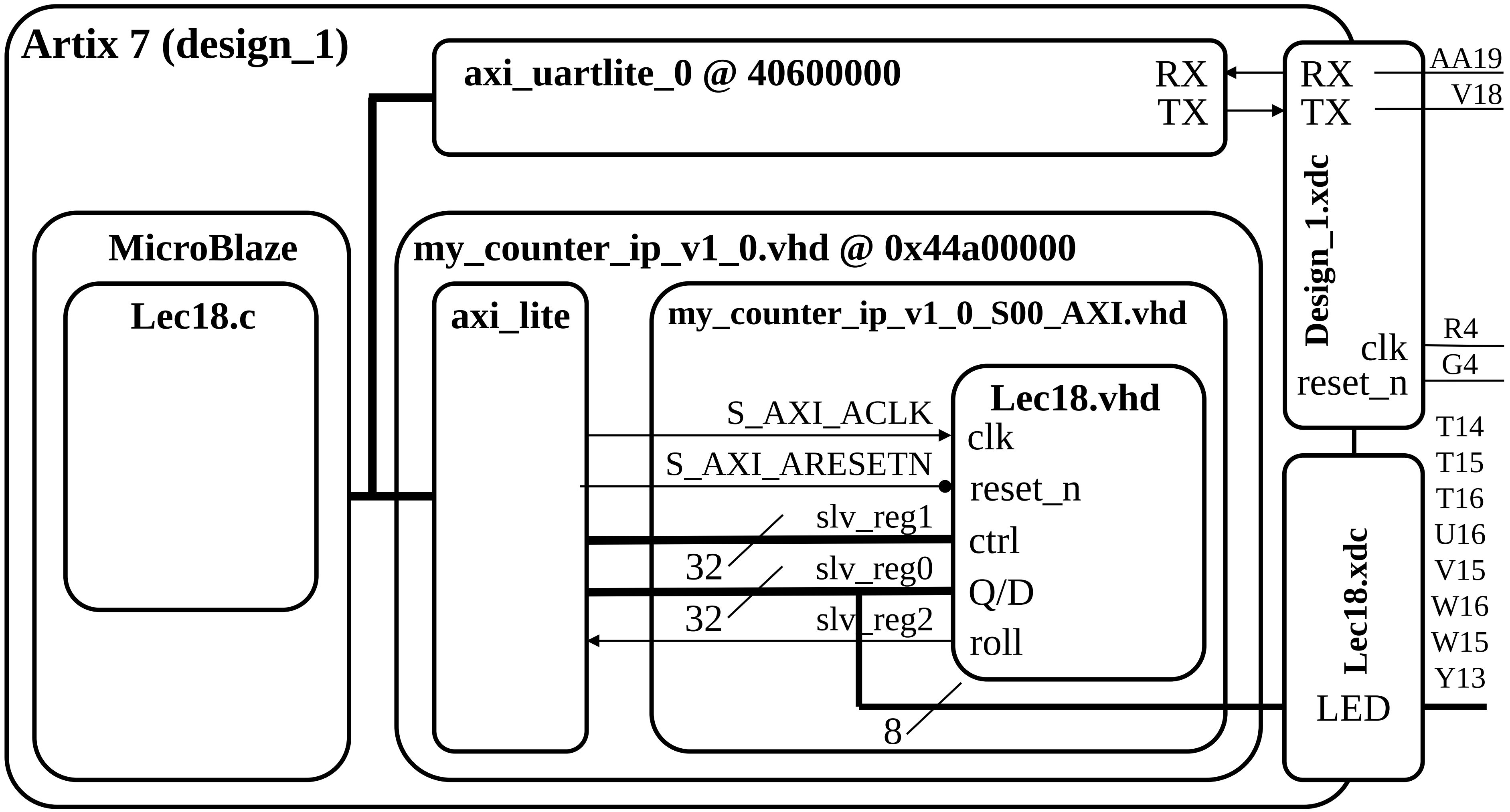
- The Run Configurations window is divided into two main sections. In the left panel, under Xilinx C/C++ application(GDB), select **Lecture18_counter.elf** or **Lecture18_counter Debug**

- ~~On the right side of this window, you will see five main tabs. Select the **STDIO Connection tab**.~~

- COM Port Selection for STDIO Connection
  - Port name should be the correct UART port. For me it showed up as COM4. Select Baud Rate as 9600. Have the **Connect STDIO to Console** box checked.  (uncheck if you want to use another terminal emulator)
- Now click on **Apply** and **Run**.
- “**Welcome to Lecture 18**” will be displayed on the Console tab
  - Type “?” to see list of commands to control the counter and LEDs

# MicroBlaze + Custom IP

## Now add “roll” for HW#10



UNITED STATES AIR FORCE ACADEMY

# Now try HW#10

## Hints

**lec18.vhd**

-- entity will need “roll” signal added

-- architecture will need to set “roll” to ‘1’ when Q is the **maxCount**.

Since the counter size is Generic based on size N, to create **maxCount**, I added….

signal **maxCount**: unsigned (N-1 downto 0);

and CSA…

**maxCount** <= (others => '1');

roll_out <= '1' when Q = maxCount else '0';

**My_Counter_IP_v1_0_S00_AXI.vhd**

-- need to update counter’s entity with new roll signal… (around line 116)

-- need an internal wire signal created to hook up to roll… I called this **roll_sig** (around line 122)

-- your microblaze will be reading “roll”, not writing to it. Your current design reads “Q” vector on slv_reg2, so you need to modify this to read “roll” bit on slv_reg2. So in the last line below, slv_reg2 will need to be replaced with a way to read **roll_sig**.

(near lines 673-679)

```
case loc_addr is
  when b"00000" =>
    reg_data_out <= X"000000" & std_logic_vector(Q);
  when b"00001" =>
    reg_data_out <= slv_reg1;
  when b"00010" =>
    reg_data_out <= slv_reg2;   -- here is where we hook up roll_sig
```

-- need to update counter’s entity where it is instantiated, with new roll signal… (around line 767), and connect “roll” to **roll_sig**

UNITED STATES AIR FORCE ACADEMY

# Now try HW#10

## More Hints

**HelloWorld.c or Lec18.c or main.c**

-- the register location for “roll” is defined for you

```
#define   countRollReg       0x44a00008         // 1 LSBs of slv_reg2 for roll
```

-- need to add code to read the roll `countRollReg` register. Could add it as a printf under the “?” command similar to reading the Q count value:

```
printf("        count Q = %x\r\n",Xil_In16(countQReg));
```

See other hints in the HW#10 assignment