See the given code at the top of
https://georgeyork.github.io/ECE383_web/lecture/lecture18.html
I'll refer to these as the "ninja" version.

0. Do Steps 1 to 11 of MicroBlaze_Install_Short_Version

Or, just open your lecture_17 vivado project and "Save As" an new project named Lecture_18

1. Open your Lecture_18 Vivado project

Go to **Tools → Create and package IP → next**

2. Create your custom IP project

2.1) Select **Create a new AXI4 peripheral** and click **Next**
2.2) Input "**My_Counter_IP**" in the name field and click **Next**
2.3) Change the number of Registers to **32** on the AXI interface and click **Next**
2.4) Select **Edit IP** and click **Finish**

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.

5. Modify **My_Counter_IP_v1_0_S00_AXI.vhd**

5.1) Add **Lines 20, 112-122, 671, 759-766** from the Ninja file to this Vivado file
5.2) Try to understand what these lines of code do by going through the powerpoint slides and handout for this lesson (like slides 14-31)

6. Modify **My_Counter_IP_v1_0.vhd**

5.1) Add **Lines 19, 59, 93** from the Ninja file to this Vivado file
5.2) Try to understand what these lines of code do by going through the powerpoint slides and handout for this lesson

4. Add the Lec 10 Counter to the My_Counter_IP_v1_0

4.1) "**Add Sources**" → **lec18.vhd** file
4.2) Try to understand what these lines of code do by going through the powerpoint slides and handout for this lesson

7. Packaging the IP core

7.0) Now click on **Package IP** in the Flow Navigator and you should see the Package IP tab.
7.1) 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.
7.2) Select **Customization Parameters** and select the line for **Merge Changes from Customization Parameters Wizard**.
7.3) Select **Customization GUI**. This is where we get to change our graphical interface. No changes at this time.

7.3) Select **File Groups**. and select the line for **Merge Changes...**
7.4) select **Review and Package** and click the **Re-package IP** button.
7.5) A popup will ask if you want to close the project, Select **Yes**.

8. Add Custom IP to your design

8.1) In the project manager page of the original window, click **Open Block Design**.
8.2) Use the **Add IP** (**plus sign**) button to add your **my_counter** IP you just created
8.3) Find your **my_counter_ip_v1.0** block in the circuit diagram. Right click on output pin **LEDs** and select **Make External** and then run **Connection Automation**
8.5) Now you need to add a constraints file to add the LED net to the pins on the Artix 7 chip by adding the the constraints **Lec18.xdc** file. (Ensure the names of the output pins match the block diagram… Mine were called **LED_0**, different than the example Lec18.xdc file)
Add sources → add or create constraints → Lec18.xdc

[Note: later when you do HW#10, you will need to modify this Custom IP counter… insert step 24 at the end of this file here then… you can skip this for now]

10. Verify Addressing Design

10.1) Click the **Address Editor** tab (next to **Diagram** tab)
10.2) Verify the addresses for the components match that of slide#47
Uart is at 0x4060_0000; my_counter is at 0x44A0_0000

11. Validate Design

11.1) Select **Validate Design** (**check box symbol or F6**).

12. Creating or Regenerate the HDL System Wrapper

12.1) right click on ***design_1*** and select **Create HDL Wrapper**.
Let Vivado manage the wrapper.

13. Generating Bit File

13.1) In the **Flow Navigator panel** on the left, under **Program and Debug** select the **Generate Bitstream** option.
13.3) After the bitstream has been generated, a message prompt will pop-up on the screen. You don't have to open the Implemented Design for this demo. Just click on **Cancel**.
[Note: one MIG error [BD 41-1273] is okay]

14. Exporting Hardware Design to SDK

14.1) On the top left corner of the window, from the tool bar click on **File** and select **Export Hardware**. Make sure the generated **bitstream** is included by **checking the box**.

15. Launching SDK

15.1) Go to **File** and select **Launch SDK** and click **OK**.

17. Creating New Application Project in SDK

17.1) Go to **File** in the main tool bar and select **New → Application Project**.
Project Name = **Lecture_18_counter**
Create New under **Board Support Package**.
Click **Next**.

18. Selecting Hello World Application from available templates

18.1) Select **Hello World** under *Available Templates* on the left panel and click **Finish**.
18.2) **Lecture_18_counter** is our main working source folder.
18.3) Replace with our C-code from lec18.c
- I just opened **hello_world.c** and cut-n-pasted the code from **lec18.c** over the code in hello_world.c
- Or you can follow the instructions in the powerpoint slides for a different method.

18.4) Try to understand what these lines of code do by going through the powerpoint slides and handout for this lesson (like slides 55-62)

19. Verify Linker Script File for Memory Region Mapping & Stack/Heap

19.0) **Double click** on the ***lscript.ld*** to open.
19.1) In the linker script, take a look at the **Section to Memory Region Mapping** box. If you did the *Make DDR3 External* step then the target memory region **must** read **mig_7series_0**. Scroll down to check if this applies to all rows. If for any region it does not say **mig_7series_0**, then click on the row under the **Memory Region** column and select **mig_7series_0**.
19.2) My stack size = **0x400** and Heap size = **0x800** [we made need to increase this when our C program gets larger]
19.3) microblaze…bram… size = **0x7FB0**

20. Programming FPGA with Bit File

20.1) Make sure that the Nexys Video board is turned on and connected to the host PC with the provided micro USB cable. Then click on the **Program FPGA** button to open the Program FPGA window. Make sure that the ***Hardware Platform*** is selected as **design_1_wrapper_hw_platform_0**.
[Note: if you did "save as" in Step 0, this may now be **design_1_wrapper_hw_platform_1**]
In the software configuration box, under *ELF File to Initialize in Block RAM ()* column, the row option must read **bootloop**. If not, click on the row and select **bootloop**.
Now click on **Program**.

21. Run Configuration

21.1) 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, click on **Xilinx C/C++ application(GDB)** and then select **lecture18_counter.elf**. *Note: In case you see* **lecture18_counter** ***Debug*** *instead of lecture18_counter.elf in this step, you can still run it without any issues.*
[Note: if you did "save as" in Step 0, you may not see **lecture18_counter.elf** as an option. Double Click **Xilinx C/C++ application(GDB)** and it should appear.]

Now click on **Apply** and **Run**.

22. Use Tera Term Terminal Emulator

Note: SDK appears to no longer support UART messages in its console, so we will need to use an external terminal emulator like Tera Term. http://en.wikipedia.org/wiki/Tera_Term (http://en.wikipedia.org/wiki/Tera_Term) to know what Tera Term is. You can download and install Tera Term from this link http://ttssh2.sourceforge.jp/index.html.en

Establish a serial connection with the correct communication port inside Tera Term. Tera Term may find your COM port your USB is using automatically. If not you can find the COM port your USB is using my going to windows **Device Manager** and clicking on **Ports (COM & LPT)**. Mine is sometimes **COM3** and sometimes **COM5**. Typically the settings will be **8 Data Bits**, **No Parity Bit**, **1 Stop Bit**.

23.1) “**Welcome to Lecture 18**” will be displayed on the Console tab
Type “?” to see list of commands to control the counter and LEDs

Now go and try to add the “roll” signal for HW#10. You’ll need to modify your Counter Custom IP, adding these steps (insert after step 8 above)

24. Updating Custom IP

24.0) Go back to your project in Vivado, click **Open Block Design**
24.1) Right click on **My_Counter_IP_1.0** block in the circuit diagram, and click **Edit in IP Packager** if you want to modify! (like adding “roll”)
24.2) Make changes to appropriate files, such as **My_Counter_IP_v1_0_S00_AXI.vhd** and **lec18.vhd** [see hints below]
24.3) Now that you updated the core you need to re-select **Review and Package** and click the **Re-package.** [same as steps 7.0, 7.4, 7.5 above]
24.4) Now back in Vivado design_1, a yellow bar on the top of the **Block Design – Design_1 window** should have a blue link titled **Show IP Status**. Click this link.
24.5) select your **my_counter_ip_0** block in the circuit diagram, and then click **upgrade selected** (box at the bottom of the screen) → **ok** → **Generate**
24.6) resume with step 10.

[Note: For HW#10, you’ll also need to make changes to the C code to interface with the “roll” signal]

Hints on HW#10
First understand the Lecture 18 block diagram with the roll signal.

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');

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_reg0, 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**

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));
```

Since "c", or count up, tends to count up by 0x26, it is hard to hit the count of xFF directly, so you can see roll = 1. Here are two example cases you could add to your C code which might be helpful:

-- add a command choices under "?"

```
printf("m: max out counter\r\n");
```

-- case max out the counter, method 1

```
/* This increments the counter until the roll signal goes high. */
case 'm':
    while(!Xil_In16(countRollReg)){
          Xil_Out8(countCtrlReg,count_COUNT);
          Xil_Out8(countCtrlReg,count_HOLD);
    }
    break;
```

-- case max out the counter, method 2

```
/* or have the counter stop when the counter reaches 0xFF to
examine the roll signal */
case 'm':
    while(Xil_In16(countQReg) != 0xff){
          Xil_Out8(countCtrlReg,count_COUNT);
          Xil_Out8(countCtrlReg,count_HOLD);
    }
    break;
```

See other hints in the HW#10 assignment