See the given code at the top of
https://georgeyork.github.io/ECE383_web/lecture/lecture19.html
I'll refer to these as the "ninja" version.

0. Do Steps 1 to 11 of MicroBlaze_Install_Short_Version to create **Lecture_19** project

EXCEPTION: In step 3.3, do NOT check **Interrupt Controller** !!!!

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, **change version to 2.0**, 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_v2_0** to see **My_Counter_IP_v2_0_S00_AXI**

6. Modify **My_Counter_IP_v2_0.vhd**

5.1) Add **Lines 19, 59, 93** from the lec 18 Ninja files to this Vivado file
5.2) Looking at today's block diagram, you'll see that the "**roll**" bit will need to pushed out to a higher level for the interrupt, similar to the LED vector. So you will add 3 lines similar to 5.1 for "roll"

5. Modify **My_Counter_IP_v2_0_S00_AXI.vhd**

Looking at today's block diagram, you need to add a one-bit **Q register**, with the **flagQ** bit (read on slv_reg2), **clear** bit (write on slv_reg3), and **set** bit (hooked to **roll** from the counter)
5.1) from the lec 18 Ninja files to this Vivado file…
Add **Lines 20 (**plus add a similar line for **roll),**
Add lines **112-122, (**put **roll** in entity). I also added signal wires **setFlag, clearFlag, flagQ**
Near line **671,**

```
case loc_addr is
  when b"00000" =>
    reg_data_out <= X"000000" & std_logic_vector(Q);  -- need this
  when b"00001" =>
    reg_data_out <= slv_reg1;  -- keep as is
  when b"00010" =>
    reg_data_out <= slv_reg2;  -- need to change this to reading the flagQ bit
  when b"00011" =>
    reg_data_out <= slv_reg3; -- keep as is… the clear flag will be written on slv_reg3
```

Add lines **759-766** (plus add "roll" in port map… roll => setFlag)

Need a CSA line to hook **clearFlag** to the appropriate bit of **slv_reg**

Need a process statement to make the one-bit **flagQ** register

It's clock will be **S_AXI_ACLK**. Its synchronous reset will be **S_AXI_ARESETN**

Logic for setting **flaqQ**? (only change on rising edge of clock)

Reset or **clearFlag** should make it '0'

setFlag should make it a '1'

otherwise, flagQ <= flagQ

Need a CSA line to hook top level **roll** to **flagQ**

[so the lower counter **roll** pin will set **flaqQ,** which inturn is connected to the top level **roll**, which will cause an interrupt in the microblaze]

4. Add the Counter to the My_Counter_IP_v2_0

4.1) "**Add Sources**" → **lec18.vhd** file [add the version you created for HW#10 with the **roll** signal]

For Lab#3, you will need to recreate clk_wiz_0 and clk_wiz_1 (from lab#2)… Select "Global" synthesis option when "generating"

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 were we get to change our graphical interface. No changes at this time.

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

For Lab#3, need to move synthesis files from advanced to general (so can compile mixed vhdl and Verilog… change box with vhdl:synthesis to just synthesis… [need better instructions]

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_v2.0** block in the circuit diagram. Right click on output pin **LEDs** and select **Make External**

8.4) Notice the **roll** signal is exposed. We need to manually connect it to MicroBlaze **interrupt** pin.

- Click the **'+' sign** by the MicroBlaze **Interrupt,** and it will expand to 3 pins

- Click on the counter's **Roll** Signal and drag to the MicroBlaze's **Interrupt** pin and release.

8.5) 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 diagram… Mine were called **LED_0**. For Lab#3, almost all the xdc signals needed "_0" added)

Add sources → add or create constraints → Lec18.xdc

10. Verify Addressing Design

10.1) Click the **Address Editor** tab (next to **Diagram** tab)

10.2) Verify the addresses and range for the components match that of slide#15

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_19_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_19_counter** is our main working source folder.
18.3) Replace with our C-code from lec19.c
- I just opened **hello_world.c** and cut-n-pasted the code from **lec19.c** over the code in hello_world.c

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**.

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 Settings for STDIO Connection

21.1) From the *Project Explorer* panel, **right click** on the **lecture19_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 **lecture19_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.*

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 19**” will be displayed on the Console tab
Type “?” to see list of commands to control the counter and LEDs

If the ISR is working, every time the counter rolls over, the isr count should increase.