CPEN311 Winter 2020 Term 2
University of British Columbia

Lab 5: Nios + Qsys + Direct Digital Synthesis + LFSR + Modulations + Clock Domains (+ a little bit of audio and VGA)

1. Introduction

In this laboratory, you will have the opportunity to practice several topics that were discussed in the course. These include, Nios, Qsys, DDS (Direct Digital Synthesis), LFSRs (Linear Feedback Shift Registers), Modulations (ASK, BPSK, FSK), and clock domain crossing.

This lab will also show you examples of usage of the VGA and further examples of audio and real-time signal processing in the FPGA. It furthermore gives an example of usage of a Real-Time Operating System (RTOS), in this case MicroC OS-II. However, the aspects mentioned in this paragraph are either optional/bonus material or are provided for you.

2. Lab Credits

This lab was written mostly by the TA Holguer Becerra and contains code that has been adapted from research conducted by the TAs Holguer Becerra and Jose Pinilla. You can read more about that research here:

http://ieeexplore.ieee.org/document/6644924/

Make sure you open the above link from within UBC since it requires a UBC library license to open (you can also use the UBC library's EZProxy service to open it from home). This reading is optional but very interesting.

3. Main Goals of the Lab

In this lab you will be provided a 2-channel "VGA oscilloscope" that can show signals on the VGA screen. You will need to instantiate a DDS and an LFSR, and connect the DDS and LFSR to a Nios-based QSYS system and the VGA oscilloscope. This Qsys system is almost complete, though you will need to instantiate 3 PIO modules to complete it and be able to connect the Qsys system to the DDS and the LFSR in order to generate and view the following modulations: ASK (Amplitude shift keying, also known as On-Off Keying (OOK)), BPSK (Binary Phase Shift

Keying), and FSK (Frequency Shift Keying). You will need to modify the Nios software in order to be able to generate the FSK signal based on the LFSR value; the other modulations will be generated purely in Verilog. You will then connect the modulated signals and DDS outputs to the VGA oscilloscope in order to be able to view them.

4. Viewing the Solution (.sof + .elf)

The first thing you need to do is to look at the solution in order to understand what it is you need to do.

1. Download the solution files dds_and_nios_lab.sof and lab5.elf from the website.

2. Connect your card to a VGA screen (or compatible TV) and a keyboard. Connect the audio output of the card either to headphones/speakers or to the VGA screen/TV if it has an audio input (you will need a different 3.5mm plug-to-plug cable for that, so probably it will be easier to connect to headphones/speakers)

3. Program the FPGA with dds_and_nios_lab.sof using the Quartus programmer

4. Open the Nios Command Shell from the Start menu

5. Download the lab5.elf file using the command **nios2-download -g lab5.elf**, as shown in the following figure:

```
$ nios2-download -g lab5.elf
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 1734KB in 28.5s (60.8KB/s)
Verified OK
Starting processor at address 0x0457E570
```

6. Once the ELF is loaded, you will see the following:
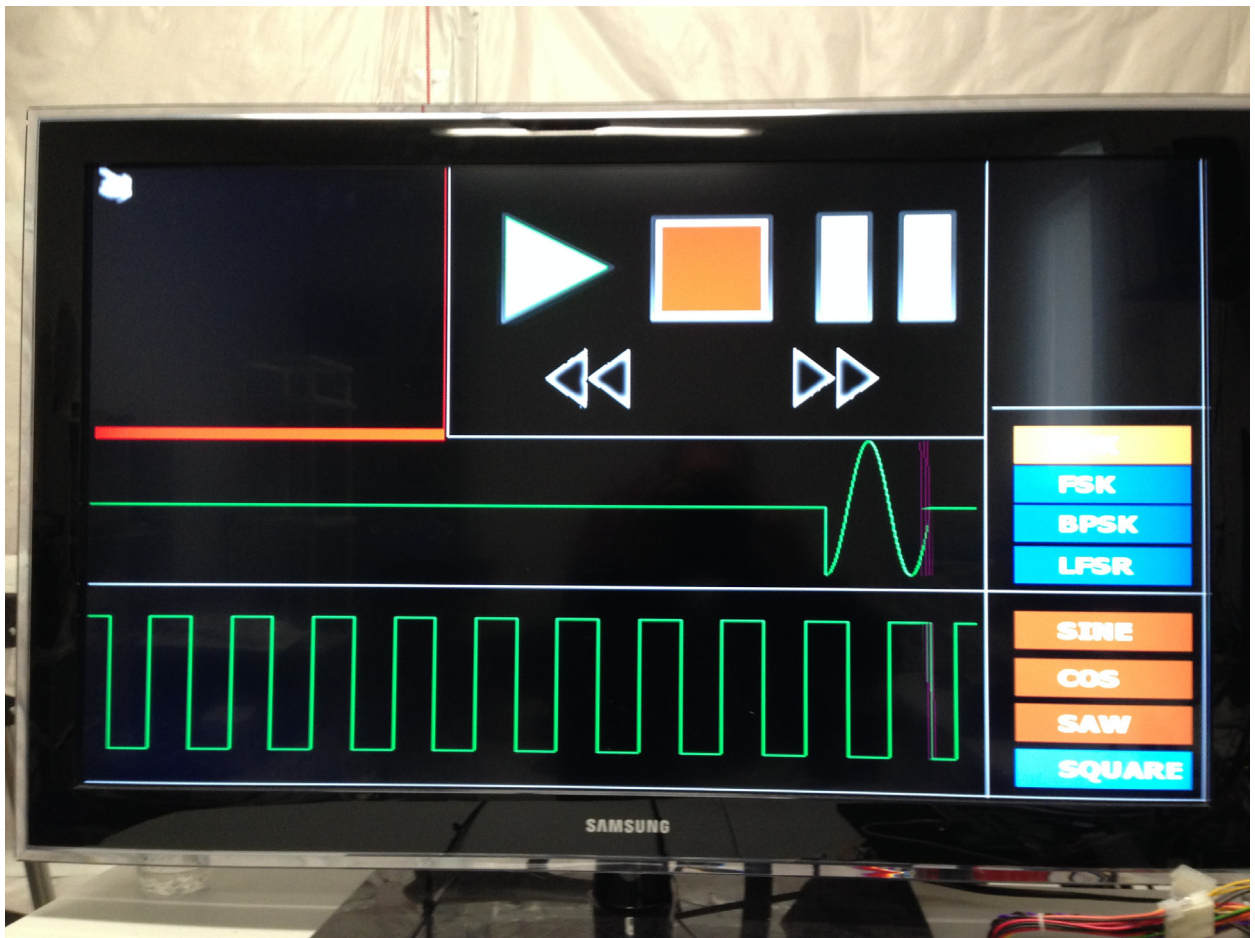
7. You can move the cursor (the small hand) with the arrow keys. Pressing Key[0] will center the cursor on the screen (particularly useful if the cursor wanders off-screen). Pressing the space bar is like clicking the mouse, it will click buttons that are under the cursor. To see a video of how to operate the solution, see here:

   https://youtu.be/oJneaw9j-Ws

8. Note that SW[0] toggles between scrolling and non-scrolling mode in the VGA oscilloscope. Play with that switch to see what it does.

We have not discussed video or VGA generation in class and this is outside the scope of the course - you will only need to use the VGA scope in this lab but not modify or understand its inner workings. However, if you are interested in learning more, you can start by reading chapters 13 and 14 in Chu's book. Interfacing to the keyboard is also provided for you (as it was in Lab 2 and 3). If you want to know more information about that, you can read chapter 9 in Chu's book.

5. Opening and running the template

Now that you understand what the solution does, it is time to look at the template and what you need to do.

1. Download the template ZIP file and unzip it (as always, in a path that does not contain spaces, or you will find that this will cause problems).

2. Open the project dds_and_nios_lab.qpf in Quartus (as always 13.0sp1 for DE2 and 14.1 for DE1SoC)

3. Compile the project. The result should be an SOF file. It should be named dds_and_nios_lab.sof or dds_and_nios_lab_time_limited.sof. I have not seen the latter filename generated but it may happen due to the fact that we are using a Nios II/f processor that sometimes requires a license. You may see the former SOF name generated in the lab (which has the full Quartus version and license) and the latter SOF name generated at home. In any case, the functionality of both files is identical as long as your USB Blaster cable is connected to your card.

4. Load the SOF file using the programmer. You should not see anything on the screen yet (the Nios software is needed for that)

5. Open the Nios II Software Build Tools for Eclipse (which you practiced using in the in-Class activity)

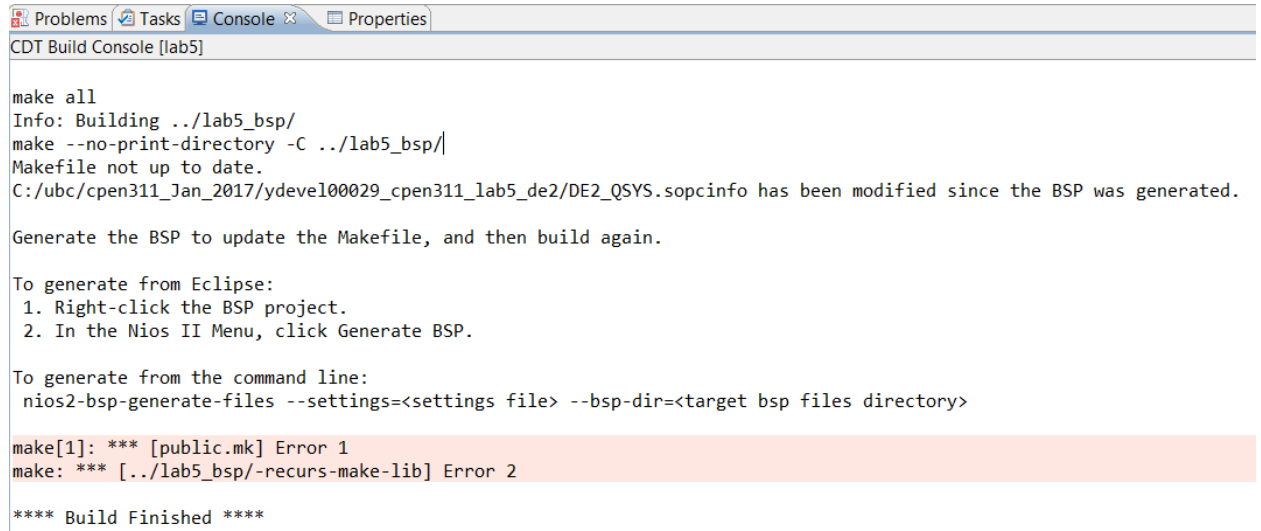6. Go to File->Import and select Import Nios II Software Build Tools Project
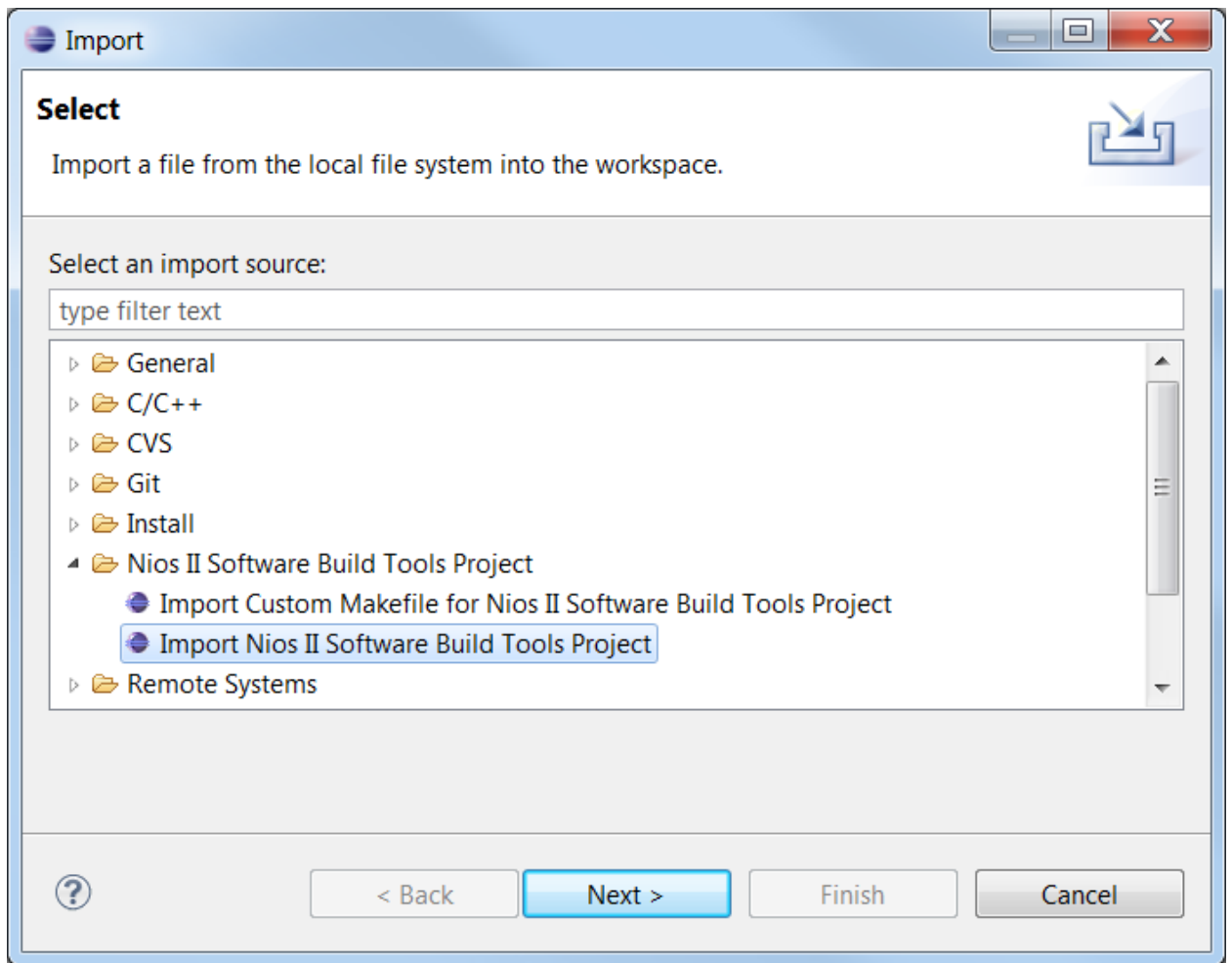
7. Import the application Nios code as follows, where of course the path to lab5 will be where you unzipped it, and press "Finish":
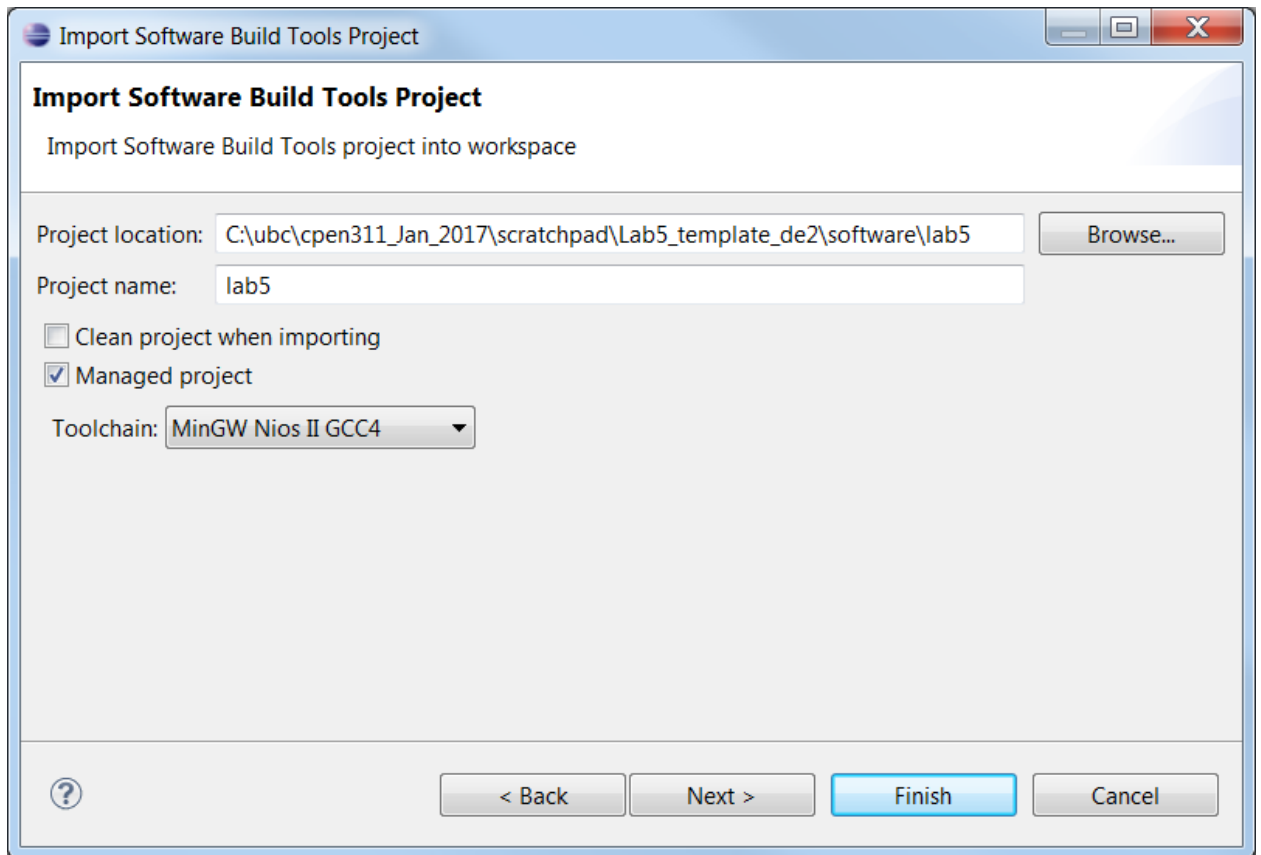
8. Go again to File->Import and Import the BSP (Board Support
   Package) as follows:
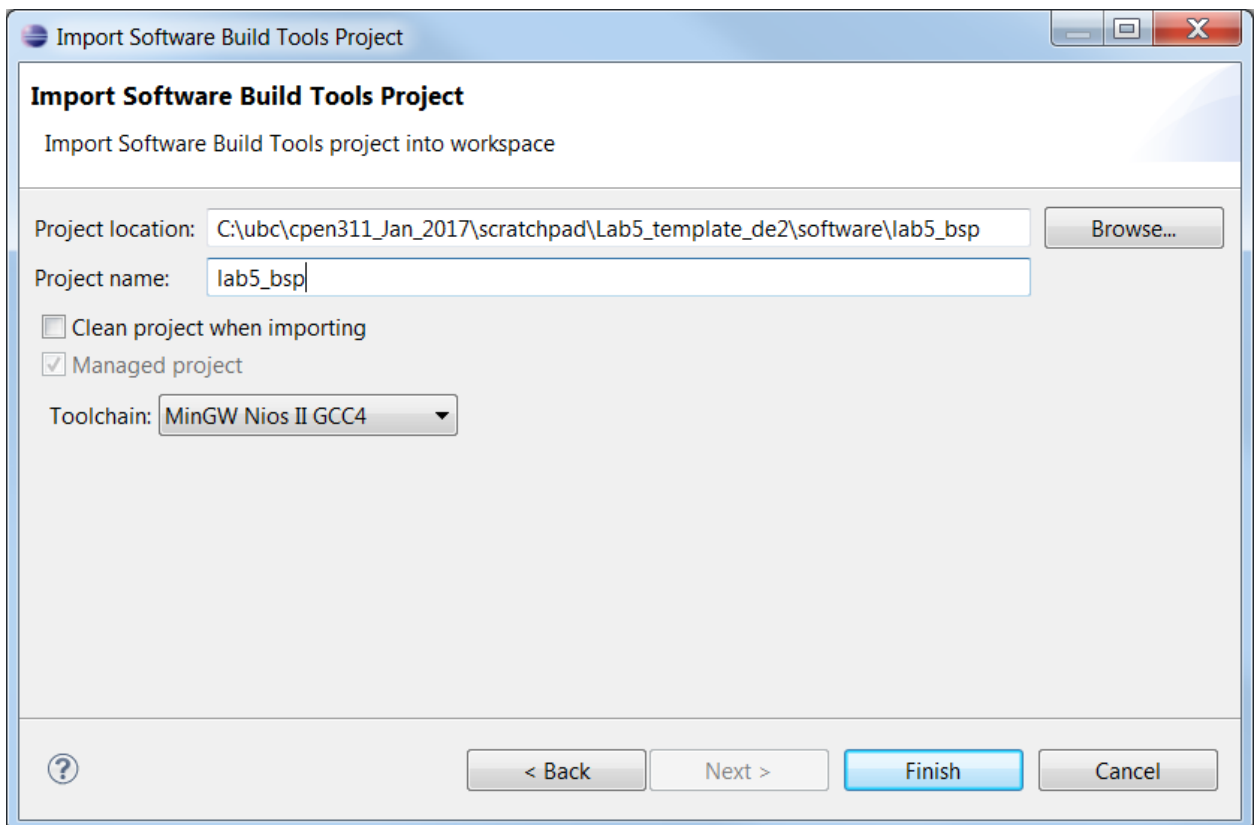
9. Build the lab5 project. You may get the following error:

```
Problems | Tasks | Console ⊠ | Properties
CDT Build Console [lab5]

make all
Info: Building ../lab5_bsp/
make --no-print-directory -C ../lab5_bsp/
Makefile not up to date.
C:/ubc/cpen311_Jan_2017/ydevel00029_cpen311_lab5_de2/DE2_QSYS.sopcinfo has been modified since the BSP was generated.

Generate the BSP to update the Makefile, and then build again.

To generate from Eclipse:
 1. Right-click the BSP project.
 2. In the Nios II Menu, click Generate BSP.

To generate from the command line:
 nios2-bsp-generate-files --settings=<settings file> --bsp-dir=<target bsp files directory>

make[1]: *** [public.mk] Error 1
make: *** [../lab5_bsp/-recurs-make-lib] Error 2

**** Build Finished ****
```
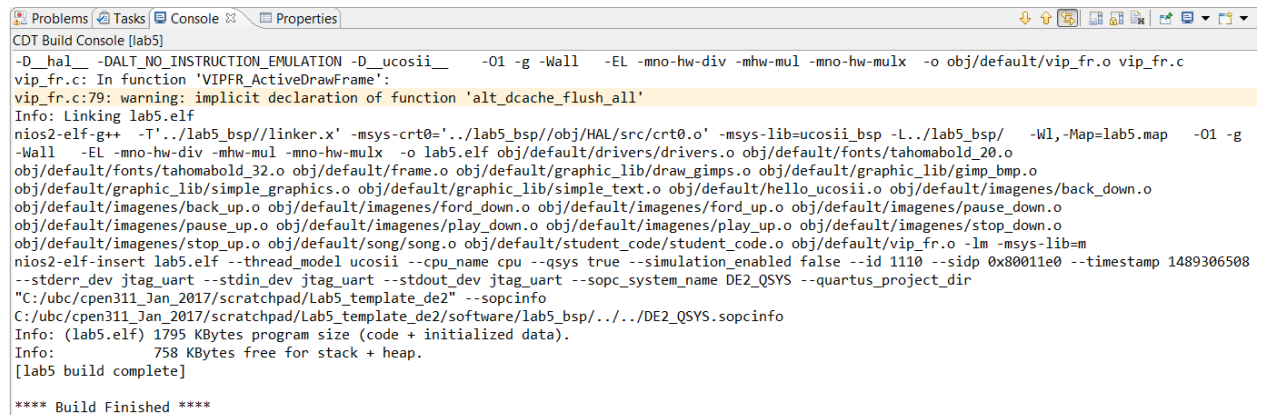
If so, follow the instructions "To generate from Eclipse" that contained therein to regenerate the BSP and then build the project again. You should regenerate the BSP in this manner every time you generate a new version of the Qsys system.

10.   When building is successful you should see something like this (for example for the DE2):

```
Problems | Tasks | Console ⊠ | Properties
CDT Build Console [lab5]
-D__hal__ -DALT_NO_INSTRUCTION_EMULATION -D__ucosii__     -O1 -g -Wall    -EL -mno-hw-div -mhw-mul -mno-hw-mulx  -o obj/default/vip_fr.o vip_fr.c
vip_fr.c: In function 'VIPFR_ActiveDrawFrame':
vip_fr.c:79: warning: implicit declaration of function 'alt_dcache_flush_all'
Info: Linking lab5.elf
nios2-elf-g++  -T'../lab5_bsp//linker.x' -msys-crt0='../lab5_bsp//obj/HAL/src/crt0.o' -msys-lib=ucosii_bsp -L../lab5_bsp/   -Wl,-Map=lab5.map   -O1 -g
-Wall   -EL -mno-hw-div -mhw-mul -mno-hw-mulx  -o lab5.elf obj/default/drivers/drivers.o obj/default/fonts/tahomabold_20.o
obj/default/fonts/tahomabold_32.o obj/default/frame.o obj/default/graphic_lib/draw_gimps.o obj/default/graphic_lib/gimp_bmp.o
obj/default/graphic_lib/simple_graphics.o obj/default/graphic_lib/simple_text.o obj/default/hello_ucosii.o obj/default/imagenes/back_down.o
obj/default/imagenes/back_up.o obj/default/imagenes/ford_down.o obj/default/imagenes/ford_up.o obj/default/imagenes/pause_down.o
obj/default/imagenes/pause_up.o obj/default/imagenes/play_down.o obj/default/imagenes/play_up.o obj/default/imagenes/stop_down.o
obj/default/imagenes/stop_up.o obj/default/song/song.o obj/default/student_code/student_code.o obj/default/vip_fr.o -lm -msys-lib=m
nios2-elf-insert lab5.elf --thread_model ucosii --cpu_name cpu --qsys true --simulation_enabled false --id 1110 --sidp 0x80011e0 --timestamp 1489306508
--stderr_dev jtag_uart --stdin_dev jtag_uart --stdout_dev jtag_uart --sopc_system_name DE2_QSYS --quartus_project_dir
"C:/ubc/cpen311_Jan_2017/scratchpad/Lab5_template_de2" --sopcinfo
C:/ubc/cpen311_Jan_2017/scratchpad/Lab5_template_de2/software/lab5_bsp/../../DE2_QSYS.sopcinfo
Info: (lab5.elf) 1795 KBytes program size (code + initialized data).
Info:           758 KBytes free for stack + heap.
[lab5 build complete]

**** Build Finished ****
```
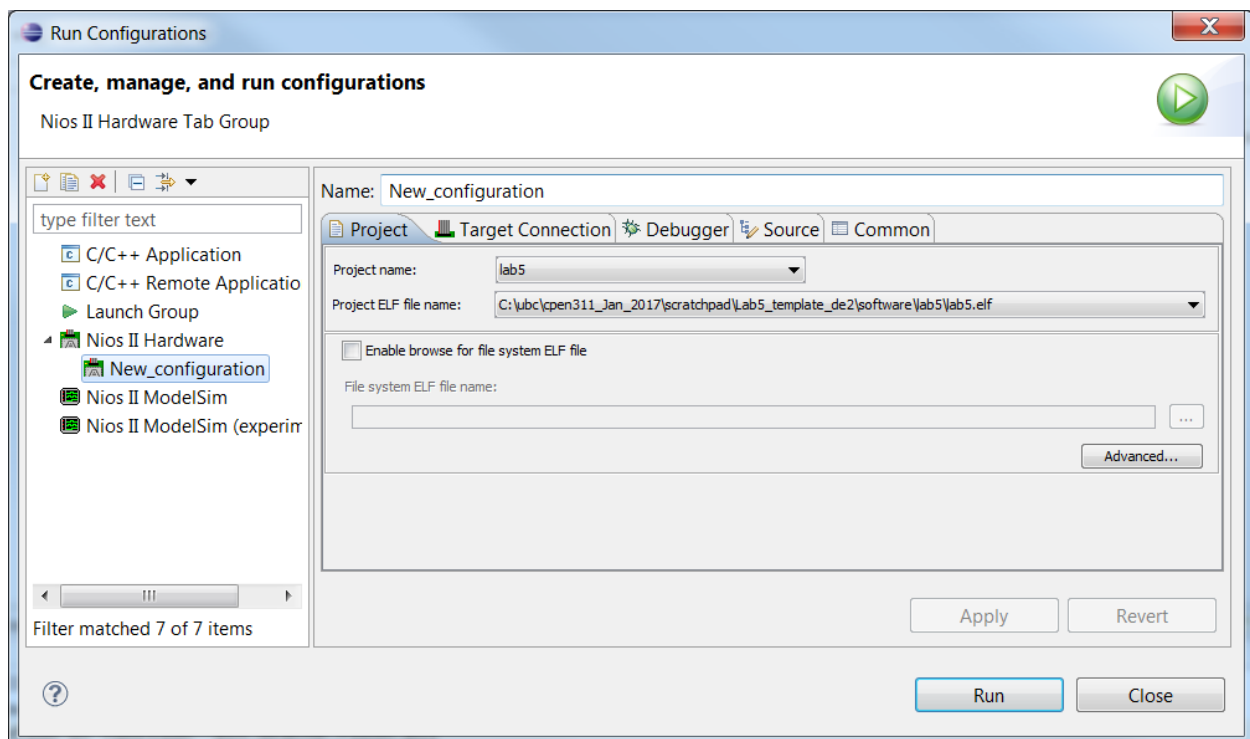
11. Go to Run->Run Configurations, and select Nios Hardware as shown here:
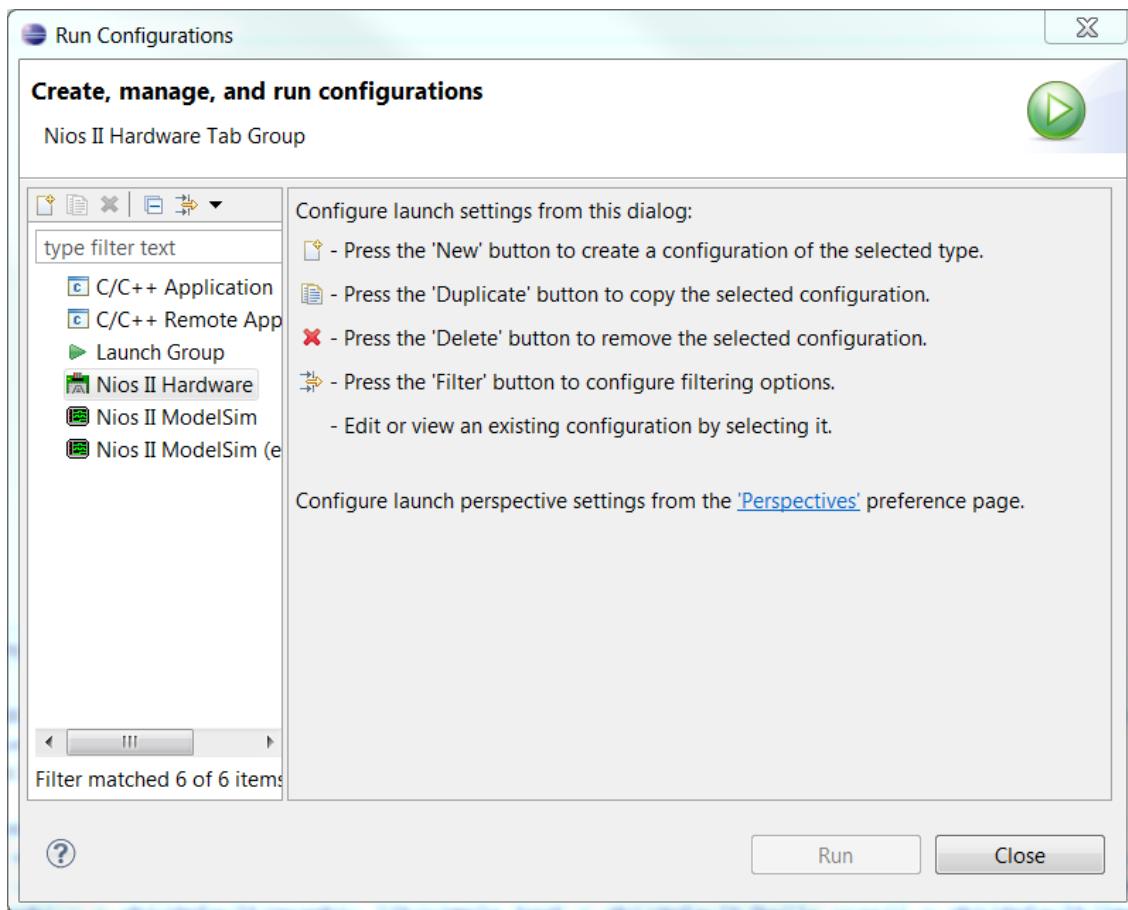
12. Right click on Nios Hardware and select "New". Create a new run
    configuration as shown below (it should be created for you
    automatically with the right fields filled in)

13.   Press "Run" to run the code. If the "Run" button is greyed out, make sure you have loaded the SOF and then go to the "Target Connection" tab and press "Refresh Connections", perhaps more than once, until the "Run" button is available (see figure below). If that does not work, then select "Ignore Mismatched System ID" and "Ignore Mismatched System Timestamp" and then press Run. Note that ignoring the ID and timestamp is not recommended in general since you want to make sure that your software matches the hardware that it is supposed to run on - if there is a mismatch then you need to regenerate the BSP and recompile the software and that should solve it. So do not ignore the ID and timestamp lightly - the code may not work if you override these settings. However, I have seen cases where the Eclipse environment is buggy and does not recognize the ID or timestamp when they are correct. Therefore ignoring these may be the correct thing to do if you encounter that problem.
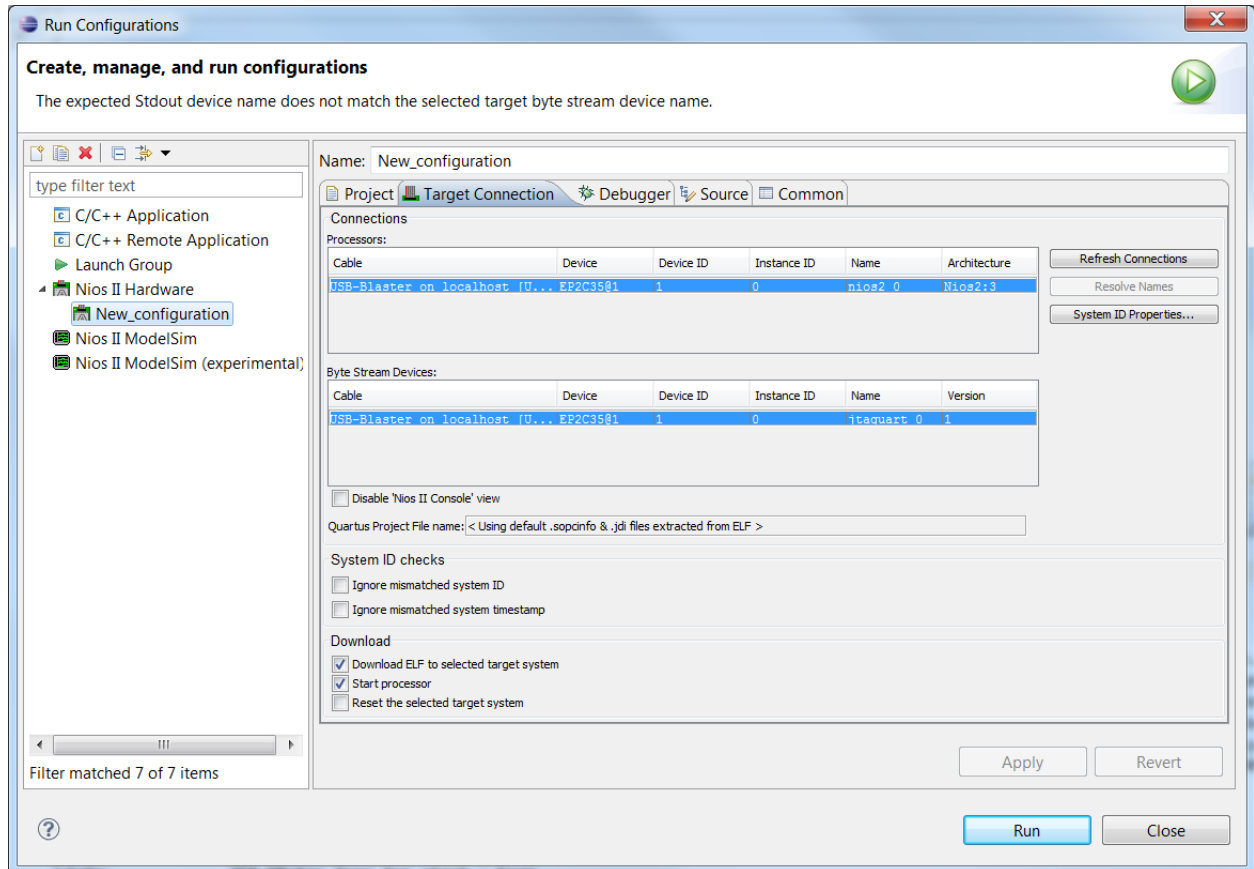
When you press "Run", you should see the following text in the "Console" tab:

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Processor is already paused
Reading System ID at address 0x080011E0: verified
Initializing CPU cache (if present)
OK

Downloading 04000000 ( 0%)
```

```
Downloading 0457E420 ( 0%)
Downloading 04580000 ( 0%)
Downloading 04590000 ( 4%)
Downloading 045A0000 ( 7%)
Downloading 045B0000 (11%)
Downloading 045C0000 (15%)
Downloading 045D0000 (18%)
Downloading 045E0000 (22%)
Downloading 045F0000 (26%)
Downloading 04600000 (29%)
Downloading 04610000 (33%)
Downloading 04620000 (37%)
Downloading 04630000 (41%)
Downloading 04640000 (44%)
Downloading 04650000 (48%)
Downloading 04660000 (52%)
Downloading 04670000 (55%)
Downloading 04680000 (59%)
Downloading 04690000 (63%)
Downloading 046A0000 (66%)
Downloading 046B0000 (70%)
Downloading 046C0000 (74%)
Downloading 046D0000 (77%)
Downloading 046E0000 (81%)
Downloading 046F0000 (85%)
Downloading 04700000 (88%)
Downloading 04710000 (92%)
Downloading 04720000 (96%)
Downloading 0472FB24 (99%)
Downloading 04730000 (99%)
Downloaded 1734KB in 28.8s (60.2KB/s)

Verifying 04000000 ( 0%)
Verifying 0457E420 ( 0%)
Verifying 04580000 ( 0%)
Verifying 04590000 ( 4%)
Verifying 045A0000 ( 7%)
Verifying 045B0000 (11%)
Verifying 045C0000 (15%)
Verifying 045D0000 (18%)
Verifying 045E0000 (22%)
Verifying 045F0000 (26%)
Verifying 04600000 (29%)
Verifying 04610000 (33%)
Verifying 04620000 (37%)
Verifying 04630000 (41%)
Verifying 04640000 (44%)
Verifying 04650000 (48%)
Verifying 04660000 (52%)
Verifying 04670000 (55%)
Verifying 04680000 (59%)
Verifying 04690000 (63%)
Verifying 046A0000 (66%)
Verifying 046B0000 (70%)
Verifying 046C0000 (74%)
Verifying 046D0000 (77%)
Verifying 046E0000 (81%)
Verifying 046F0000 (85%)
Verifying 04700000 (88%)
Verifying 04710000 (92%)
Verifying 04720000 (96%)
Verifying 0472FB24 (99%)
Verifying 04730000 (99%)
Verified OK
Starting processor at address 0x0457E570
```
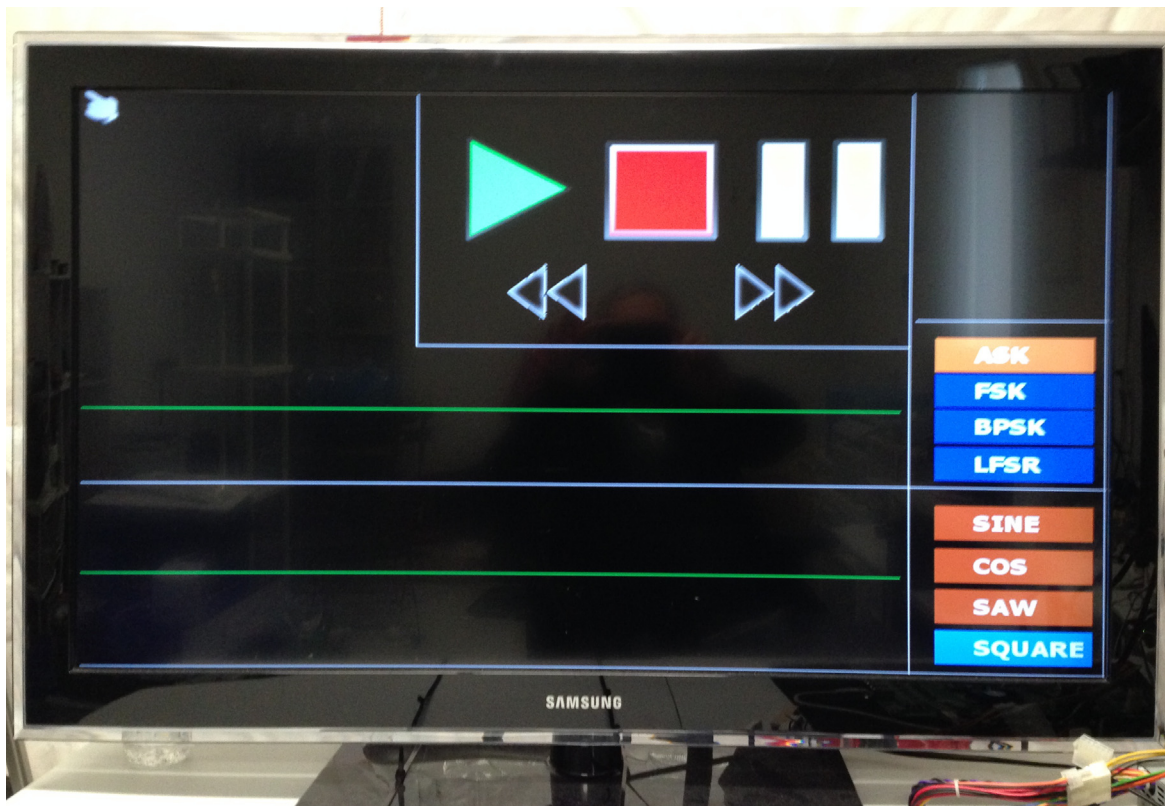
After that text appears, you should see the VGA screen as follows:

You will see that the oscilloscope channels are both showing 0. The buttons and cursor will work but have no effect. Audio is disabled (more about that in a minute). Note that there may be noise coming out of the audio port for the template, this is normal. If that happens just disconnect the headphones/speaker or the audio connection to the monitor to avoid hearing annoying noise.

Your job in this lab is to generate the signals that are to be viewed by the oscilloscope channels and interface that to the Nios and QSYS so that it is all controllable via the keyboard, and connect the signals to the VGA circuitry so that they are viewable in the VGA.

Regarding audio, compilation of the audio circuit adds 50% to the compilation time and adding the histogram visualization in the top left corner more than doubles the compile time. In order to avoid these issues, and since audio is not a primary goal of this lab, the audio has been disabled by default in the templates. Do not enable the audio portion until you have finished the lab and want to get bonus marks. To enable the audio, uncomment the following line in dds_and_nios_lab.v:

`define ENABLE_AUDIO_DEMO

This will enable the audio functionality but not the histogram visualization. To enable that, you must also change the line:

parameter COMPILE_HISTOGRAM_SUPPORT = 0;

to:

```
parameter COMPILE_HISTOGRAM_SUPPORT = 1;
```

Again, note that enabling audio and histogram visualization will increase compile times by about 50% and 100% respectively. Note that enabling histogram visualization without also enabling the audio has no effect. You can look at how the enabling/disabling is done at the end of dds_and_nios_lab.v and there you will see some examples of more advanced SystemVerilog syntax.

6. Overview of what you need to do to the code

What you need to do to get from the template to the solution and get full points are the following:

1. (7% of Functionality score) Instantiate a 5-bit LFSR that runs at a clock rate of 1 Hz

2. (7% of Functionality score) Instantiate a DDS that generates a 3 Hz carrier sine

3. (7% of Functionality score) Use the LFSR to modulate the DDS carrier sine to generate ASK (OOK) and BPSK signals

4. (7% of Functionality score) Connect the modulated signals and DDS outputs and LFSRs, through muxes that are controlled by the Nios, to the VGA oscilloscopes for display

5. (12% of Functionality score) Generation of the FSK signal: Connect the LFSR, the LFSR clock, and the DDS to the QSYS in order to generate FSK using the Nios and interrupts

6. (5% of the Reliability score) In every case where clock domains are crossed, use appropriate clock crossing logic

Note that all audio manipulation is purely bonus material and is not part of the primary goals of lab.

A carrier frequency of 3 Hz and a data frequency of 1 Hz is quite low. Most communications systems operate at much higher frequencies (for example, your cellphones operate at gigahertz carrier frequencies and data rates of tens of megahertz). However, this is not to say that a communications system that operates at single Hz frequencies is useless: military submarines sometimes use this frequency range in order be able to receive communications while deep underwater. If you are interested you can Google "Extremely Low Frequency Communications". We are using this frequency band in this lab in order to be able to easily view the signals in the VGA oscilloscope.

# 7. Instantiating the LFSR and LFSR Clock

This task is worth 5% of the Functionality score.

We will instantiate a 5-bit LFSR to give us a pseudo-random bit sequence which we will use to modulate the DDS.

The schematic of the LFSR is shown in the following figure:
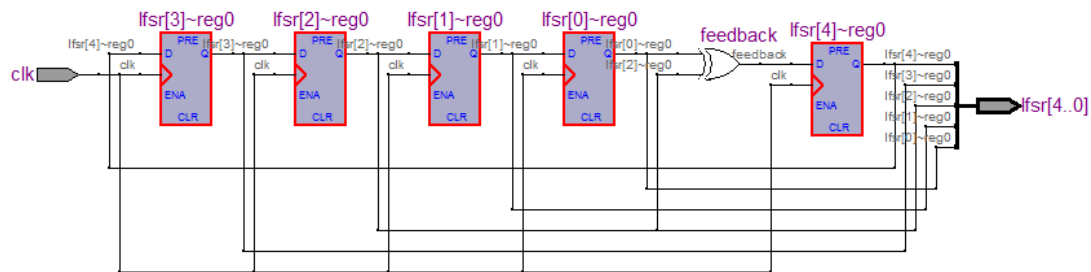
Write a Verilog module that implements this circuit and simulate it to make sure you get the PN5 sequence. The sequence should be (numbers are hexadecimal):
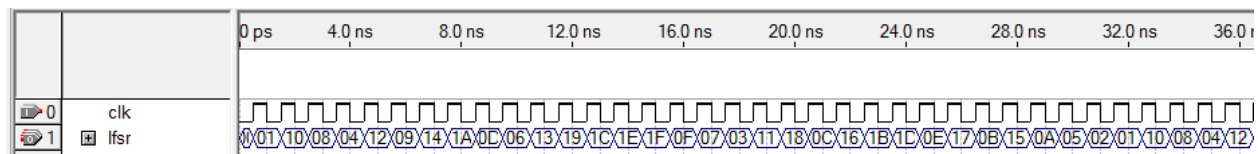
Note that the sequence repeats after 31 values, which means this is an m-sequence since $2^5-1 = 31$. Furthermore note that you must make sure that your LFSR has a non-zero initial value via appropriate initialization or reset code (not shown in the schematic) because if it is 0 it will stay at 0 (as a simulation will show). The value of 0 is the only value which the LFSR does not take on out of the possible 32 values that it has (this is always the case for m-sequences, the period of 2^m-1 because the "-1" is there since the LFSR does not take on the value 0).

Use a clock divider from Lab 1 to generate a 1 Hz clock from CLOCK_50. Use this to drive the LFSR.

8. Instantiating the DDS

This task is worth 5% of the Functionality score.

Read the file waveform_gen.pdf, which is the datasheet of the DDS core that is provided. Instantiate that module in your Verilog code. For now, calculate the DDS tuning word that would generate a 3 Hz carrier and put that constant as the input to the DDS. The DDS clock should be CLOCK_50 (50 MHz). Note that the input of the DDS is a 32-bit tuning word (unsigned) and the outputs are 12-bit wide (signed, 2's complement).

9. Modulating the DDS using the LFSR

This task is worth 5% of the Functionality score.

Take bit 0 of the LFSR as the pseudo-random bit sequence. Use this bit to modulate the DDS to generate ASK (OOK) and BPSK signals as taught in class. The modulated signal should be a waveform whose sample rate is 50 MHz (just like the DDS). Make sure to do any clock domain crossing properly. You should get 12-bit signed 2's complement modulated signals as the output of these modulations.

10. Connecting the signals to the VGA oscilloscope

This task is worth 5% of the Functionality score.

At this stage you should have all the signals except FSK generated. You should now connect them to the VGA oscilloscope logic in order to be able to view them. The signals of interest are as follows:

logic [11:0] actual_selected_modulation;

is the signal that goes to the top oscilloscope channel, and:

logic [11:0] actual_selected_signal;

is the signal that goes to the bottom oscilloscope channel.

The signal:

wire [3:0] modulation_selector;

comes from the Nios via the Qsys and selects the signal going to the top oscilloscope channel (only lower 2 bits of modulation_selector are used, values from 0 to 3).

The signal:

wire [7:0] signal_selector;

comes from the Nios via the Qsys and is used to select the signal going to the bottom oscilloscope channel (only lower 2 bits of signal_selector are used, values from 0 to 3).

Note that the DDS outputs signals with a sample rate of 50 MHz whereas **actual_selected_signal** and **actual_selected_modulation** should be signals that are synchronous to the VGA oscilloscope sampling signal "sampler" which has a frequency of 200 Hz. Make sure to include appropriate clock domain crossing logic.

Note that **modulation_selector** and **signal_selector** come from the Nios/Qsys and are controlled by the cursor - this part is already done for you. You can look into the code for the Nios in order to see how this is done if this interests you.

11. Generating the FSK signal

This task is worth 10% of the Functionality score.

In order to generate the FSK signal, we will use a hybrid software/hardware approach. The goal here is to give you some practice in using the Qsys and Nios and to give you some additional experience in hybrid hardware/software signal processing (we did something similar in Lab 3).

The big picture is this: the LFSR clock (of 1 Hz) will generate an interrupt to the Nios every rising edge. That interrupt will cause the Nios to look a the LFSR value (via reading a PIO) and depending on bit 0 of the LFSR, it will write a DDS word that makes the DDS go to 1 Hz if LFSR[0]==0, or 5 Hz if LFSR[0]==1, or in other words we will modulate the DDS to generate binary FSK with frequencies of 1 and 5 Hz (so a center frequency of 3 Hz). Note that this modulation **must** be done in this way, not by Verilog, to receive points (this is similar to Lab 3 - we could do this in Verilog and it might even be more efficient, but the point here is to practice Nios + Qsys skills).

To complete this task, we will do the following:

1. Generate a 1-bit PIO to capture the rising edge of the clock signal of the LFSR. The PIO will have the following instantiation:

Connect it to the 50 MHz clock and reset, to the Nios and to an external conduit as appropriate to be able to connect to the LFSR clock (see in-class activity to refresh your memory if needed). You **must** name this module lfsr_clk_interrupt_gen.

2. To instantiate a PIO that allows the Nios to read the LFSR value, instantiate an input PIO with the appropriate parameters and connect it accordingly, as follows:

| System Contents ✕ | Address Map ✕ | Interconnect Requirements ✕ | 🔧 Parameters ✕ |

System: DE1_SoC_QSYS   **Path:** lfsr_val

PIO (Parallel I/O)
altera_avalon_pio

▾ Basic Settings

Width (1-32 bits): `32`

Direction:
- ◯ Bidir
- ◉ Input
- ◯ InOut
- ◯ Output

Output Port Reset Value: `0x0000000000000000`

▾ Output Register

☐ Enable individual bit setting/clearing

▾ Edge capture register

☐ Synchronously capture

Edge Type: `RISING ▾`

☐ Enable bit-clearing for edge capture register

▾ Interrupt

☐ Generate IRQ

IRQ Type: `LEVEL ▾`

Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture
register is logic true. Available when synchronous capture is enabled

▾ Test bench wiring

☐ Hardwire PIO inputs in test bench

Drive inputs to: `0x0000000000000000`

It too must be connected to the 50 MHz clock and reset and to the Nios. You **must** name this module lfsr_val.

3. To instantiate a PIO that allows the Nios to control the DDS tuning word, instantiate an output PIO with the appropriate parameters and connect it accordingly, as shown below:

It too must be connected to the 50 MHz clock and reset and to the Nios. You **must** name this module dds_increment.

4. Once you have connected these modules in the Qsys, select "System->Assign Base Addresses" in the menu in order to assign them base addresses automatically. Then generate the Qsys system.

5. After the Qsys has generated, connect the new ports of the Qsys to your logic in dds_and_nios_lab.v in order be able to control the DDS and monitor the LFSR via those PIOs. Once you have done that, press "Compile" in Quartus to compile the design.

6. In the Eclipse software development environment, right click on the lab5_bsp project and select "Nios II->Generate BSP". Once you do that, you should see in the "system.h" file in the lab5_bsp project the following constants (search for them):

LFSR_CLK_INTERRUPT_GEN_BASE

LFSR_VAL_BASE

DDS_INCREMENT_BASE

7. Go the "lab5" software project and look in the subfolder "student_code" in the file "student_code.c"

```c
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
void handle_lfsr_interrupts(void* context)
#else
void handle_lfsr_interrupts(void* context, alt_u32 id)
#endif
{
    #ifdef LFSR_VAL_BASE
    #ifdef LFSR_CLK_INTERRUPT_GEN_BASE
    #ifdef DDS_INCREMENT_BASE

// Your code goes here!

    #endif
    #endif
    #endif
}
```

The entire software code that you need to write for this lab (can be done in 10 or so lines) goes into this interrupt routine. This interrupt routine is called every time a clock edge is detected in the lfsr_clk_interrupt_gen PIO. In this interrupt routine you need to do the following:

(a)   read the LFSR value and check bit 0.

(b)   If LFSR bit 0 is 0, then write the tuning word corresponding to 1 Hz to the dds_increment PIO. If LFSR bit 0 is 1, write the tuning word corresponding to 5 Hz to the dds_increment pio.

(c)   Reset the edge capture mechanism in the lfsr_clk_interrupt_gen PIO to prepare for the next clock edge

Most of the above is straightforward. I do recommend, for efficiency, pre-calculating the DDS words for 1 Hz and 5 Hz (instead of at every interrupt - this is wasteful of processor cycles and adds unnecessary delay). Furthermore, resetting the edge capture register in the PIO is described in "C Example - An ISR to Service a Button PIO Interrupt" on page 8-15 of the file nios_2_sw_development_guide.pdf.

8. Once you have finished writing the software, compile it and load it. Make sure that when you select FSK in the graphical user interface, the signals are as expected. If not, as always, use the debugging tools at your disposal to figure it out and if that fails ask a TA for help or post a question on Piazza.

12. Bonus points
There are a variety of ways to get bonus points:

1. (10%) Implement QPSK modulation and show it in the VGA Oscilloscope. Note that the QPSK data in both the I and Q arms must be pseudo-random and different from each other. Note that the solution SOF file has a mistake in it insofar as how the QPSK waveforms are generated. Refer to the graphs of QPSK waveforms in the lecture slides instead.

2. (5%) Add at least 2 additional songs and "Next" and "Previous" buttons in the GUI to be able switch songs. The provided script "readbin.m" which converts a bin file into a C vector and which you can use to convert songs into a form suitable for the Nios program. Furthermore, in the appendix to this document a method has been provided to convert images to C vectors, which can be used to generate images for the "Next" and "Previous" buttons.

3. (5%) Add Nios PIOs, buttons, and software to change the colors of the graphs upon command.

Regarding any bonus that involves the audio, this is undocumented code so part of the challenge is understanding what it does. The instructor or TAs will not be able to help you there.

13. Specific grading requirements for this lab

You **MUST** use a Nios for this lab. If you do not use the Nios, you will get 0%.

To get full points for simulation, you need to show simulations of:

1. The LFSR

2. The DDS (this does not have to show the modulations but that would be nice)

3. Any clock-crossing modules you use

4. Any FSMs you wrote for this lab

If in doubt about what to do, remember that you can always load the solution SOF and see what it does.

Do not forget the rules of good design:

- Always design while thinking about the **hardware** implementation
- Use **simple** structures
- **Incremental** design and test

- **Modular design**: **divide** the work, work in **parallel** in the group
- Use the **RTL viewer**
- Verification and test: Use **simulations**, LEDs, 7-segments, SignalTap, LCD scope (or SignalTap equivalent)
- Write **Clean**, **Neat**, and **Legible** code
- Give **meaningful** names to variables, use **comments**
- Go over and understand the warnings given by Quartus during compilation
- The circuit should be correct by **design**, not just by simulation.
- The code should be **verifiable** by **inspection**
- Design in a **modular** fashion, always thinking about future **reutilization** of the module.
- **Re-use** proven modules, instead of re-inventing the wheel all the time

Good luck and have fun!

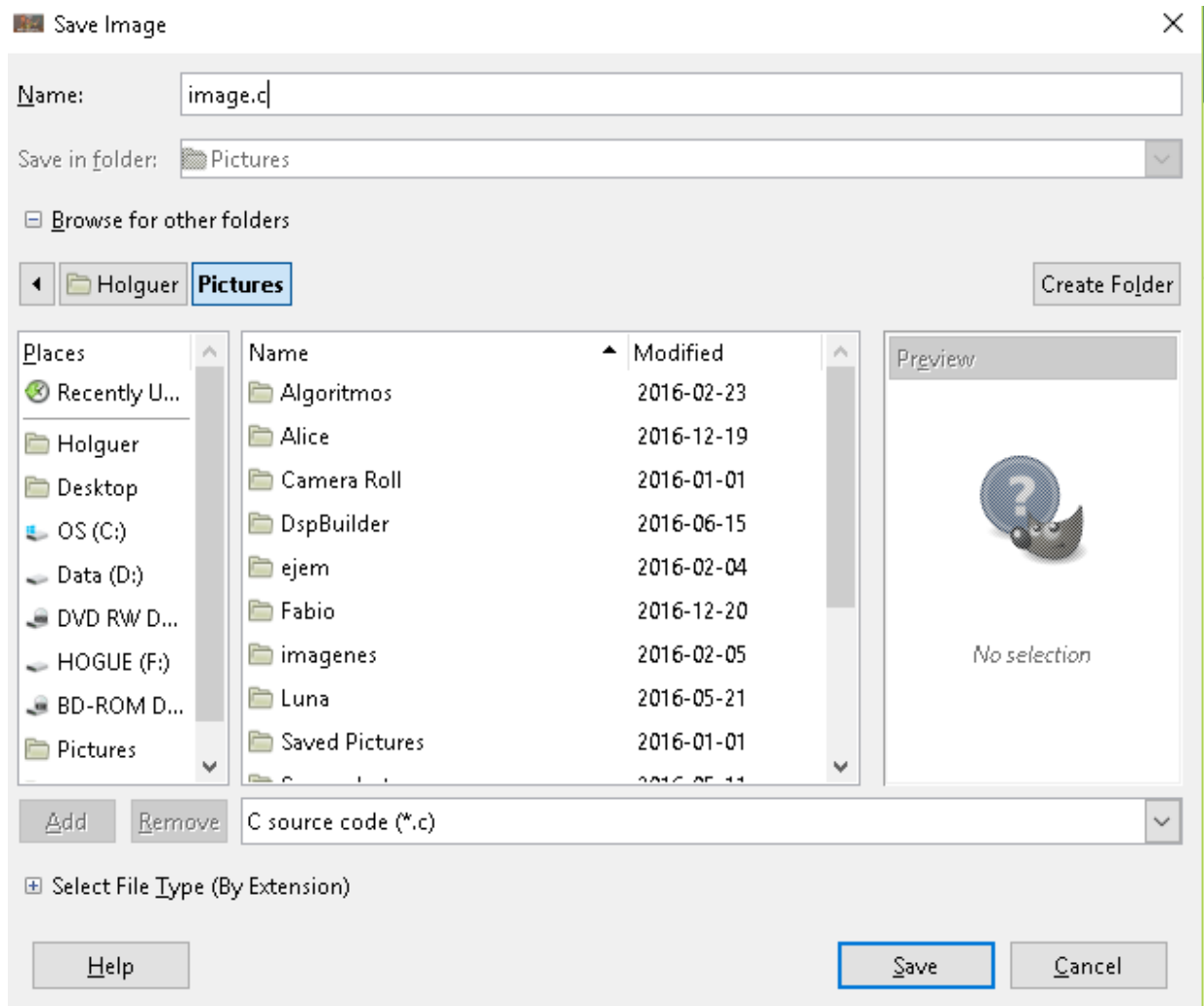## 14. Appendix - How to convert an image into a .c file

How to convert an image into a .c file:
1) Download the image processing program "Gimp" (https://www.gimp.org/)
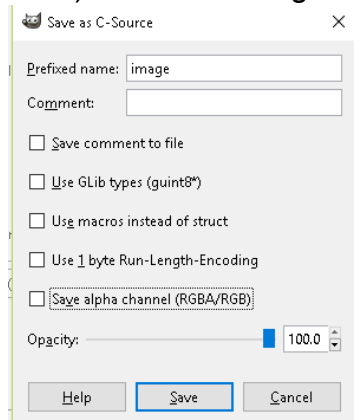2) Open an image using Gimp.
3) Select "Save as"

4) Select the type .c file.

5) Save the image as follows, using the name of your preference:

6) Open the generated file, and erase the "static"

```
static const struct {
  unsigned int    width;
  unsigned int    height;
  unsigned int    bytes_per_pixel; /* 3:RGB, 4:RGBA */
  unsigned char   pixel_data[500 * 306 * 3 + 1];
} image = {
  500, 306, 3,
```

```
const struct {
  unsigned int    width;
  unsigned int    height;
  unsigned int    bytes_per_pixel; /* 3:RGB, 4:RGBA */
  unsigned char   pixel_data[500 * 306 * 3 + 1];
} image = {
  500, 306, 3,
```

7) Copy the .c to the Nios II project.