DE1-SoC Tutorial

COE838: Systems-on-Chip Design
Lab 3

1. Objectives
The purpose of this lab is to introduce students to the HPS/FPGA design flow involved in SoC design using the DE1-SoC development board. Students will create a hardware prototype in VHDL for the Cyclone V using Quartus II and QSys. Using ARM DS-5 and a serial terminal, students will also learn how to control (FPGA) hardware through embedded C programs running on a Yocto Linux operating system. This lab will therefore provide students with the fundamentals for prototyping SoC designs from both a hardware and software perspective.

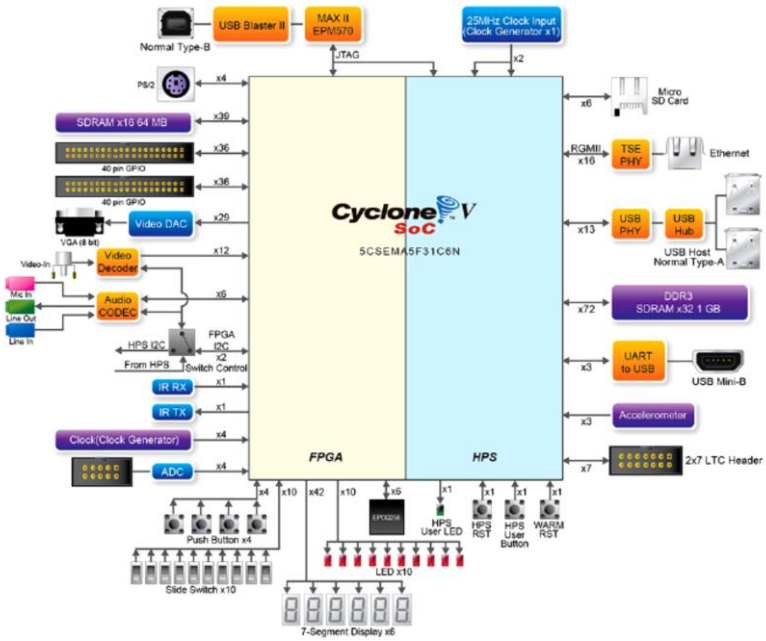

Fig. 1: Block Diagram of the Cyclone V HPS/FPGA Device for DE1-SoC

2. DE1-SoC Overview

2.1 HPS/FPGA Cyclone V Device
A general block diagram of the DE1-SoC dev board is provided in Fig. 1. The DE1-SoC contains a Cyclone V device which comprises of two distinct components - an FPGA and Hard Processor System (HPS). The HPS is a hard logic (soft processor) microprocessor unit (MPU) consisting of a dual-core ARM Cortex-A9 processor, on-chip memories, SDRAM, L3 interconnect, and support and interface peripherals. The HPS will be used to execute the software portion of your SoC design.

The Cyclone V's FPGA component consists of FPGA fabric, standard FPGA components (LUTs, CLBs, PLL etc), shared memory controllers, and general peripherals found on a standard FPGA dev board. The FPGA is used to prototype hardware for your SoC design, receiving and sending data to and from the HPS using AXI buses, bridges, and Avalon master-slave devices.

As seen in Fig. 1, the HPS and FGPA components each have their own pins and peripherals on the device. Consequently, the pins are not freely shared between the HPS and FPGA fabric. The FPGA and HPS pins must therefore be designated at different stages of the design flow.

2.2 HPS/FPGA Communication

The HPS-to-FPGA and FPGA-to-HPS master/slave connection block diagram used by the Cyclone V is provided in Fig. 2. This figure gives a detailed outline of how the HPS, FPGA and its associated peripherals communicate through a series of bridges and AXI buses. As seen in the figure, there are 3 main bridges used for communication: the FPGA-to-HPS bridge (f2h), the HPS-to-FPGA bridge (h2f), and the Lightweight HPS-to-FPGA bridge (**lwh2f**). For this lab and subsequent labs, we will only be concerned with the lwh2f bridge. Although only the lwh2f HPS-to-FPGA communication is shown in Fig. 2, the slaves are allowed to communicate back to the HPS through the FPGA-to-SDRAM connections provided by the FPGA's Avalon Memory Mapped (MM) Master.

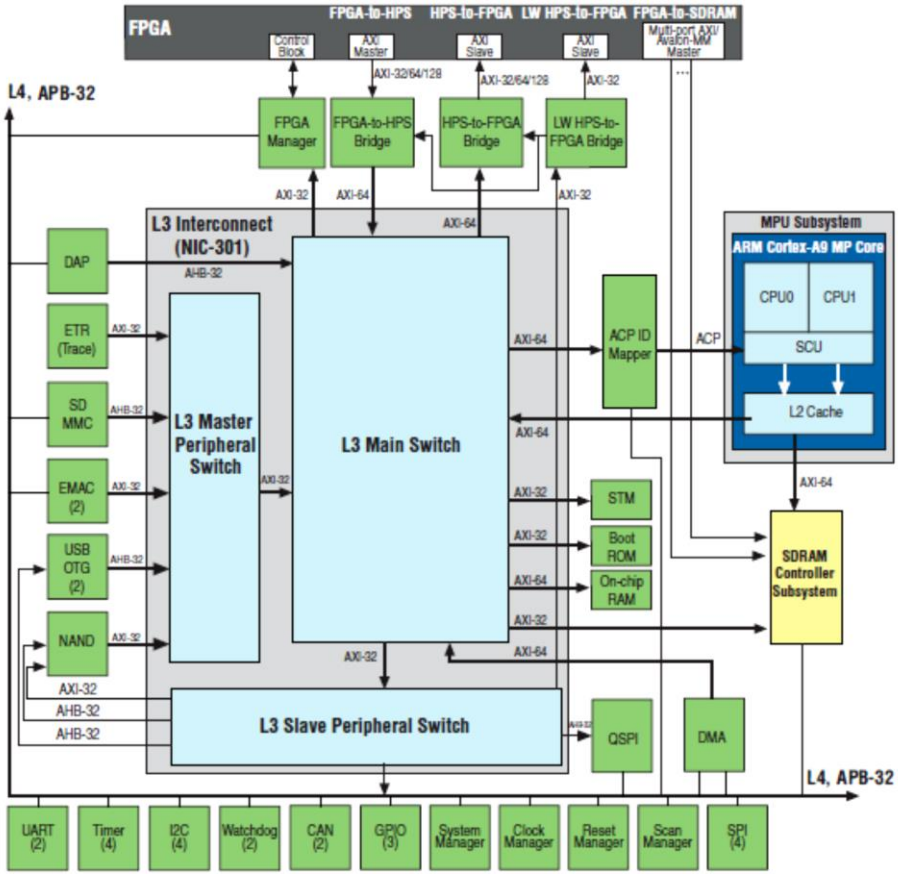

Fig. 2: HPS/FPGA Bus and Bridge Communication Block Diagram

Table I: Common Address Space Regions for Bridge Access

Region Name	Description	Base Address	Size
FPGA Slaves	For accessing FPGA slaves connected to the h2f bridge	0xC0000000	960MB
HPS Peripherals	Accessing slaves directly connected to the HPS	0xFC000000	64MB
Lightweight FPGA Slaves	Accessing slaves connected to the lwh2f bridge	0xFF200000	2MB

As seen in Fig. 1, the lwh2f bridge and AXI bus support a 32-bit datawidth between the HPS and FPGA (i.e one word). The HPS supports communication with the FPGA/peripherals through the L3 interconnect, which is connected to the HPS' (DDR3) SDRAM Controller. It is therefore essential that the SDRAM pins be configured correctly so that the HPS may read/write data to/from the SDRAM controller and establish communication between the L3 interconnect and FPGA.

Once all hardware has been correctly prototyped, communication between the HPS and FPGA is programmed through a memory mapped C application. Memory mapping allows the CPU to view and access the FPGA's address space (containing our components) so that we may read/write information as necessary, controlling the hardware through software. The C application you will develop uses APIs to send write (or receive read) data to (and from) specified memory addresses.

The IP components you add to your system each possess a base address. You will use the base addresses to access, control, and send data to and from your SoC components using your C application. These addresses will be generated for you as header files using the NIOS II Command shell. A general design flow can be seen in Fig. 3, with common base addresses given in Table I (the lightweight FPGA slave address will be of interest to your design).

Once your C application is complete, a binary is generated by compiling your software on a host computer. The binary must be placed on a USB which will be inserted and mounted to the HPS/FPGA system. You must then copy the binary from the USB to your HPS home directory to execute the application. Upon execution, the HPS will communicate with the FPGA prototype based on the APIs and functionality you have coded in your C application. You may access and interact with the HPS/ Yocto Linux OS from your host computer using a serial connection (minicom **or** the DS-5 terminal).

2.3 DE1-SoC HPS/FPGA Design Flow

The HPS/FPGA design flow is provided in Fig. 3. This figure outlines the design steps we will need to follow for prototyping a SoC design with a DE1-SoC based HPS/FPGA system.

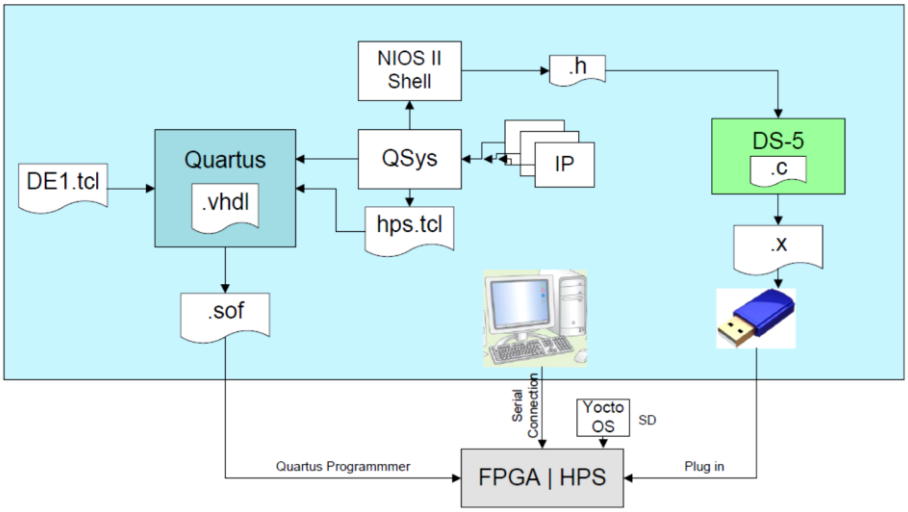

Fig 3. Overall HPS/FPGA Design Flow for Altera's DE1-SoC

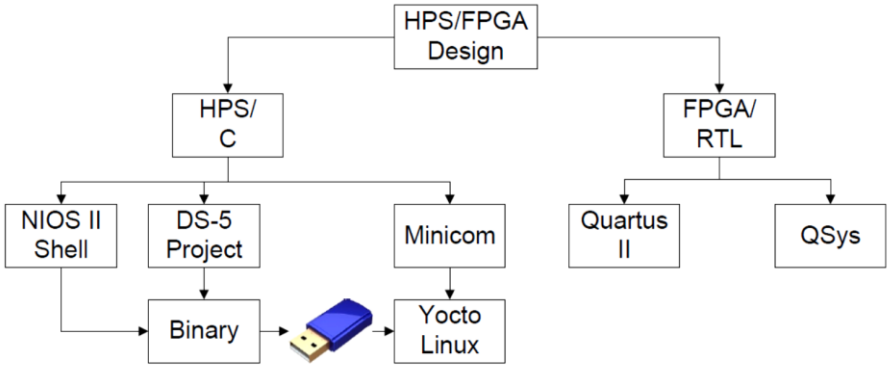

Fig. 4: Tools and Flow used for DE1-SoC Design

3. Tutorial

3.1 HPS/FPGA Design Structure Overview

There are many steps we must follow for designing a HPS/FPGA SoC. A structured block diagram outlining the general tools used for the hardware and software flows is given in Fig. 4. Consult this diagram when following the tutorial guide (along with Fig. 3) to understand what step you are currently executing in the SoC design flow. You will also be going back and forth between various development tools. Be prepared.

3.2 Hardware Design

3.2.1 Quartus Part-I

1. Open Quartus II 14.0. Note that you may only use **version 14.0** and above for Altera Cyclone V and Ryerson EE network compatibility.

2. Follow the directions to create a new project
    a. On the home screen, select "New Project Wizard"
    b. On the "Introduction" screen select "next"
    c. Page [1 of 5]
        i. In the first field, select your working directory- where you would like the project to be saved (preferable with a new folder name in your coe838 directory)
        ii. Give a project name in the second field. This should be copied automatically to the third field. For the purpose of this lab, let's use "LED_HEX_FPGA".
        iii. Select next
    d. Page [2 of 5].
        i. Select next
    e. Page [3 of 5]
        i. Under "Device Family" select Cyclone V
        ii. Under "Package" select FBGA
        iii. Under "Available Devices", select 5CSEMA5F31C6. Note you may also use the first few letters "5CSEMA5" in "name filter" to quickly find this device.
        iv. Select Next
    f. Page [4 of 5]
        i. Under "Tool Type" - "Simulation" select "ModelSim-Altera". Under "Formats" select VHDL
        ii. Select Next
    g. Page [5 of 5]
        i. Verify that all the information you have specified in the previous steps is correct.
        ii. Assuming all information is correct, select Finish.
    h. Your Quartus II project workspace should now open and resemble Fig. 5.

3. Navigate to the COE838 course directory coe838/labs/lab3/rtl and copy all the **files** in the /rtl folder to your project folder.

4. Go back to coe838/labs/lab3 and copy the **/ip folder** (the entire folder, not the files inside) to your project directory. Ensure sure that your project folder now contains an /ip folder.

5. Go back to Quartus II. Select "Project" - "Add/Remove files in project". Select the "..." button and select the LED_HEX_FPGA.vhd file. Press OK. Select ADD. Select Apply. Select OK.

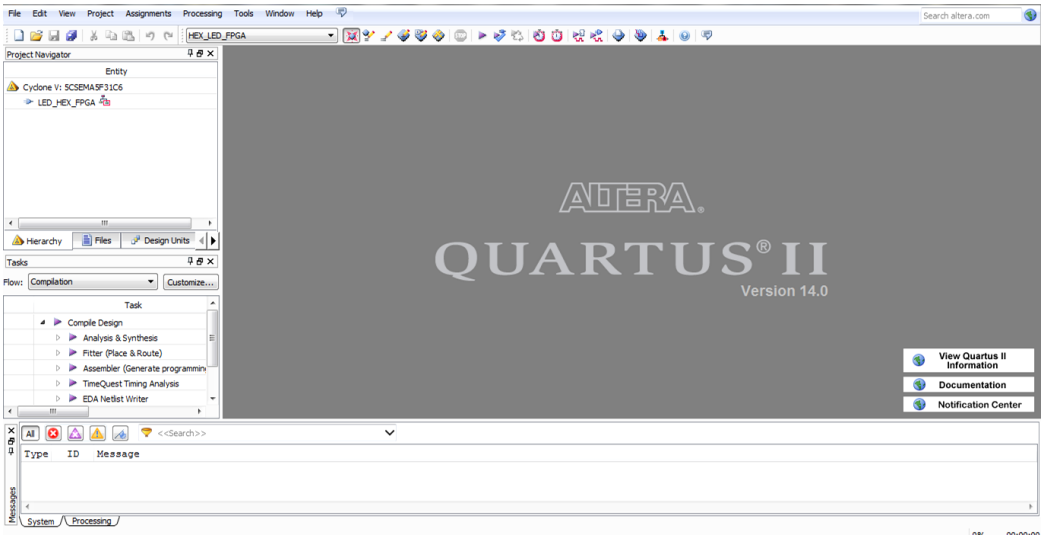

Fig. 5: Quartus II Project Workspace

6. In Quartus' left pane "Project Navigator" window, double click "LED_HEX_FPGA". This should open the VHDL file we just added to the project. If this does not occur, select "File" - "Open" - LED_HEX_FPGA.vhd. Select "Project" - "Set as Top-Level Entity". Then repeat step 6 to verify you have successfully selected this VHDL file as your top level-entity.

7. Observe the VHDL code provided to you. The entity contains keywords used by the Cyclone V for its HPS/FPGA pinouts. These names can be found in the *pin_assignment_DE1-SoC.tcl* file provided to you. Notice that there are comments noting where you should place the SoC component and it's port map given by QSys.

3.2.2 QSys

a) Adding Components

1. In Quartus, select "Tools" - "QSys". Wait for QSys to launch.
2. A window will open prompting you to select a file in your project folder. Select soc_system.qsys and press Open.
3. Another window will open to verify the component list and their compatibility with QSys. Once it has finished verification, select Close.
4. A QSys system will open containing a clock source and HPS IP block. Make sure to use this .qsys template for all your labs/project as it contains HPS parameter mappings specifically for the DE1-SoC. We will now add the custom IP cores needed for our SoC and make the proper connections to the HPS/FPGA bridges.
5. In the QSys "IP Catalog" left window pane, expand "Project" - "Terasic Technologies Inc". Highlight "SEG7_IF" and select "+ Add..".
6. A SEG7_IF parameters window will open. Ensure the following values are specified:
    a. SEG7_NUM: 6
    b. ADDR_WIDTH: 3
    c. DEFAULT_ACTIVE: 1
    d. LOW_ACTIVE: 1

   Click finished in the SEG7_IF window once complete.

7. Next, we will add a Parallel I/O (PIO) IP block to the system. We will need this to access the LEDs on the FPGA. Select "Processors and Peripherals" - "Peripherals" - "PIO (Parallel I/O)". Press "+Add...".

8. A Parallel I/O parameters window will open. Ensure the following values are specified:
   a. Width(1-32 bits): 10
   b. Direction: Output

   Note: the same PIO IP can be selected for accessing the *Switches* on the DE1-SoC.

   Click Finish. Messages and warnings will appear in the bottom window. We will resolve these soon.

9. In the QSys "System Contents" window, follow the "Name" column, and right click "pio_0". Select rename and change the default name to "led_pio". We can leave SEG7_IF_0 since this name bares meaning. If you chose not to name the IP blocks with the names specified, be conscience of this fact when designing and coding your VHDL in Section 3.3.

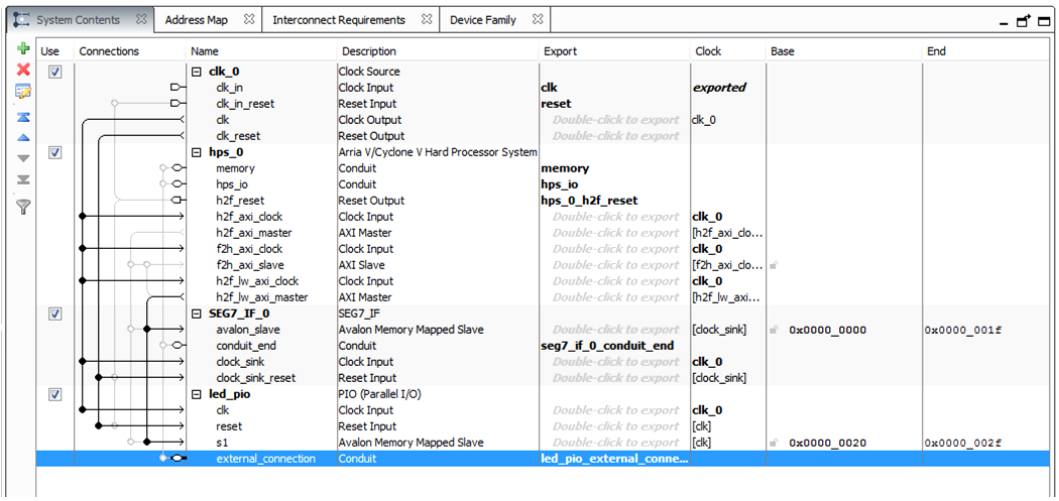

Fig. 6: QSys System Components and Connections

b) Component Connections

1. Next we will use the "Connections" column in QSys to map the IP blocks to the HPS. Each empty circle signifies a connection that may be made between two or more component (ports) in your system. A black/filled circle signifies that a connection/signal has been established between the two components.

2. Follow clk_0's clk (clk_0.clk) connection down to the SEG7_IF_0 and led_pio components. Establish a connection (i.e. press the empty circle to color black) between clk_0.clk and:
   - SEG7_IF_0's *clock_sink* port
   - led_pio's *clk* port

3. Make a connection between clk_0.clk_reset and:
   - SEG7_IF_0's *clock_sink_reset* port
   - led_pio's *reset* port

4. Follow hps_0.h2f_lw_axi_master and make a (slave) connection between:
   - SEG7_IF_0's *avalon_slave* port
   - led_pio's *s1* port

5. Follow led_pio's **external connection** port horizontally to its "export" field. There is a faded label which says *"Double-click to export"*. Double click the field. A default name will appear called "led_pio_external_connection". This process is referred to as exporting a **conduit**.

6. Similarly, export the SEG7_IF_0 conduit. It will be given the default name "seg7_if_0_conduit_end".
7. The base addresses present on the component conduits may overlap and generate an error in the Messages window. In QSys, select "System" - "Assign Base Address". Your QSys system and connections should now resemble Fig. 6. There should also be no warnings or errors in the Messages window.

> **Exporting Conduits** are used to create external connections so that other components may access and communicate with your IP. In this case, we export led_pio and SEG7 so that the HPS system may communicate with these components.
>
> Similarly, the **slave** connections you make in QSys allow the IPs to communicate with the FPGA fabric and HPS (using the lwh2f bridge/AXI bus and L3 interconnect). The slaves also possess base addresses that you may include in your memory mapped I/O software to control and communicated data to the IPs.

c) Generating the QSys System

1. In QSys, go to "Generate" - "HDL Example". Under "HDL Language" select VHDL. Press 'Copy' to copy the template and paste the VHDL code to your top-level entity (in the ARCHITECTURE section). We will use this code as a basis for instantiating and port mapping the HPS component into your SoC design. Press 'Close' when you finish copying the code.

2. Next in QSys, select "Generate" - "Generate HDL...".

3. The Generation window will open. Under Synthesis "Create HDL design files for synthesis:" select VHDL. Leave Create block symbol file (.bsf) enabled.

4. The directory in the field "Output Directory" should match your project directory, with an appended folder name "soc_system" which will be created.

5. Click "Generate" at the bottom. This will save the QSys system and generate your custom SoC VHDL block called "soc_system". Wait for the system to generate and select "Close" to close the generation window once complete.

   At this point, QSys has generated: 1) the soc_system VHDL files needed for your SoC design, 2) a .tcl script for HPS pin assignments and 3) a .qpf file that needs to be included in the Quartus project for successfully synthesizing this HPS/FPGA system.

   Next, we will generate the header files required for our HPS C software design. These .h files provide the names and base addresses of our QSys components. Therefore using these .h files we can simply #include the hps_0.h file in our code and virtually access these addresses through memory mapping.

6. In QSys, select Tools - NIOS II Command Shell [gcc4].

7. This will open a terminal. *cd* to your project directory.

8. Once in your project directory, type *sopc-create-header-files*. A message will appear in the terminal stating that the header files have been generated.

9. In the NIOS terminal, create a folder called "software" with the command *mkdir software*. Move the generated header files to the software folder using the command *mv *.h software*

10. Open the hps_0.h file in a text editor to view the generated base addresses and constant names that you will use to memory map these IPs in your c code. Minimize this window.

11. Go back to the NIOS terminal and close it. In QSys select "File" - "Save". Once saved, select "Finish" at the bottom to return back to Quartus II.

3.2.3 Quartus Part-II

a) Port Map the SoC System

1. Using the HDL code copied from QSys, instantiate the soc_system component in the ARCHITECTURE section.
   Note: Please use the ARCHITECTURE from QSys generated code. Use SIGNAL and PORTMAP part of the code from the Appendix.

2. Use the *u0: soc_system* template obtained from QSys to port map the soc_system ENTITY to your overall design. An example of the final VHDL can be found in the Appendix of this lab.

3. Next, we need to include the SoC system generated by QSys in our project: In Quartus select "Project" - "Add/Remove Files in Project". Navigate to the folder *soc_system/synthesis/* and select *soc_system.qip*. Press OK. Select "Add" - "Apply" - "OK".

b) Pin Assignment

1. In Quartus II, select "Tools" - "Tcl Scripts...". Select *pin_assignment_DE1-SoC.tcl* and click Run. A popup window will show up when successful. Click OK.

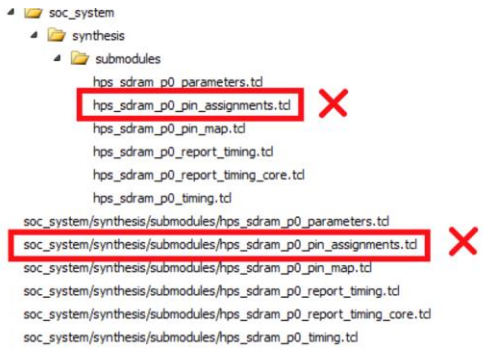

Fig. 7(a): Incorrect HPS Tcl Script Selection

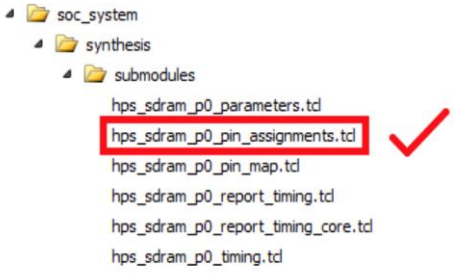

Fig. 7(b): Correct HPS Tcl Script Selection[1]

NOTE: A **bug** exists in Quartus 14.0. If you see *hps_sdram_p0_pin_assignments.tcl* listed more than once as shown in Fig. 7(a), exit Quartus and restart. Repeat steps 1 and 2. Before going to the next step, ensure that the window lists the .tcl script once as in Fig. 7(b).

2. Next, select the *hps_sdram_p0_pin_assignments.tcl* and click Run. A popup window will show up when successful. Click OK.

3. Once complete, click Close on the .tcl script window.

c) Compile Design

1. The final step is now to compile the design. Select "Processing" - "Start Compilation". Assuming that the RTL is coded correctly, **no errors** should exist. However there will likely be multiple warnings. You may ignore these.

[1] Obtained from *SoC-FPGA Design Guide 0.29* Draft, by EPFL's Sahand Kashani-Akhavan and Rene Beuchat

3.3 Software Design

1. On the EE network, navigate to Applications – Engineering – **Altera** Eclipse for DS-5. Launch the application.
2. Select your project workspace as the **/software** folder location you created earlier on. Press OK.

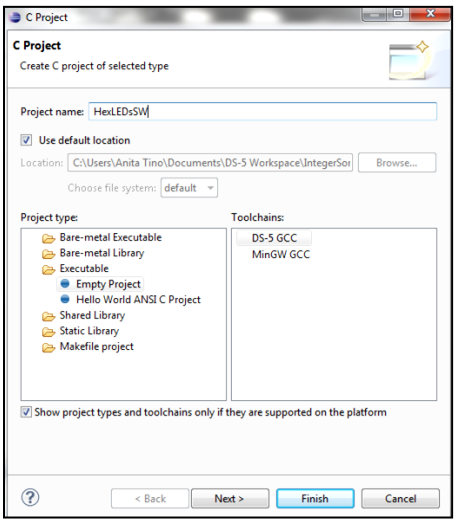

Fig. 8: Creating a New Project in DS-5

3. Create a new project by selecting "New" – "C-Project". A window will show up. In the Project type window select "Executable" – "Empty Project". Under Toolchains ensure the "DS-5 GCC" compiler is chosen. Name the project HexLEDsSW, ensure that your default project location is correct, and that your project specifications are identical to Fig. 8. Press Finish. Your workspace will appear.
4. Navigate to the course directory */coe838/labs/lab3/software* and copy all the **.c and .h files** in this folder to your lab3 directory's */software* folder. Next, create a zip file of all the contents in your software folder (including the .h file generated by NIOS II shell) and call it *software.zip*.
5. Go back to DS-5 and select "File" - "Import" - "General" - "Archive File". Click Next. Under "From archive file" select the browse button, navigate to your software folder and select the *software.*zip file. A list of files will show. Select "Finish". Your workspace will now have all the files included in your .zip file.
6. **Libraries:** Next, we will need to include the necessary libraries for compiling this project. Select "Project" - "Properties" - "C/C++ Build" - "Settings". Under the "Tool Settings" tab go to "GCC C Linker" - "Libraries". Select the  icon under the Libraries (-l) option. A window will popup. Type in *pthread* and press OK. Do this again to add the *m* and *rt* libraries.
7. **Includes:** Next, we must include a path specific to the Altera SoC hardware API files. In the same Project Properties window from step 6, navigate to "C/C++ General" - "Path and Symbols" - "Includes" tab, highlight **GNU C** and click "Add...". In the directory field type:

   /usr/local/Quartus-EDS-14.0/embedded/ip/altera/hps/altera_hps/hwlib/include

   and press OK. Select OK at the bottom of the Project Properties window.
8. Build your project using the shortcut CTRL-B or pressing the icon  . A successful build message should appear in the console window.

9. Next, plug in your USB to the host computer. Navigate to your */software* folder and into its **/Debug** folder. Assuming a successful build in Step 5, an executable file named *HexLEDsSW* (or the equivalent name which you gave your project) will exist. Copy this onto your USB stick. Eject the USB from the host computer.

3.4 Hardware/Software Execution

3.4.1 Software Preparation

1. Power on the DE1-SoC board located at your workstation.
2. You may create a serial connection to the ARM processor (running Yocto Linux) in **one** of the following two ways:
   a. **DS-5 Terminal** – In DS-5, select "Window" – "Show View" – "Other" – "Terminal" – "Terminal". A window for specifying the serial connection details will appear. Ensure that the following is selected:
      - Connection Type: Serial
      - Baud Rate: 115200

      The rest of the information should remain as defaulted. Press OK. In the terminal window select the  icon to establish a connection with HPS and its Linux OS.

      OR

   b. **Minicom** – On the host computer, open a terminal. Type *minicom SOC-USB0*. A connection will be established with the board opening a new window session within the terminal.
3. If no text is displayed in the terminal, press enter. A prompt from the OS will show up asking for login credentials. The first prompt will show:
   - *socfpga login: bob*
   - This signifies that your username is *bob*. Next a password prompt will show.
   - *Password: bob*
   - Thus your username and password is bob
4. Next, plug in your USB drive into one of the USB ports at the back of the DE1-SoC board. A message will appear listing that you have added a device to the system located at /dev/sda1. You now need to mount your USB to Linux so that you can access the files on your drive. To do this, in the serial terminal type *mount /mnt*
5. Since you do not have any executable privileges on your mounted device, you must copy your executable to your working directory. Do this by typing the command: *cp /mnt/HexLEDsSW* . (include the . when issuing the command to signify that you wish to copy the executable in the 1st argument to this directory).

3.4.2 Hardware Preparation

1. Now go back to **Quartus**. Launch the programmer by going to "Tools" – "Programmer", or click the  icon. The Quartus Programmer window will open.
2. Press the *Hardware Setup* button at the top left of the window. From the *Available hardware items* select DE1-SoC. Double click the field. This should change the *Currently selected hardware* to the DE1. If not, select it manually from the drop down list. Press close.
3. In the Programmer window's left panel, select *Auto Detect*. This will automatically detect the device connected to your host computer via JTAG. A popup window will appear, select the second option which reads "5CSEMA5" and press OK.

4. Next, we need to upload the .sof file to the FPGA. In the left pane, select "Add File...". Navigate to your project folder's *output_files* folder and select the *HEX_LED_FPGA.sof* file. Once selected press OK.

5. Now you will have 3 chips/devices present in the programmer. Highlight the 5CSEMA5 device containing no .sof file (in the top window). Right click the device and select delete. You should now have the SOCVHPS and .sof device present in the programmer space as shown in Fig. 9.

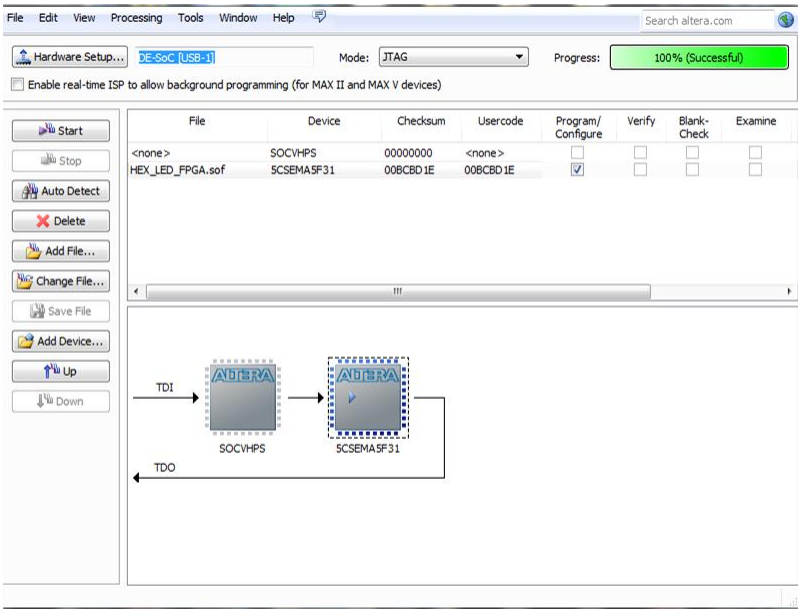

Fig. 9: Quartus Programmer Window

6. In the Programmer's top window, make sure that the .sof device has a checkmark in the "Program/Configure" checkbox. In the left pane, select start. A status bar in the upper right corner will show the progress of your .sof bitstream upload to the DE1-SoC. You should also see a green light on the board. Wait until it is 100% successful. Your FPGA (hardware prototype) has now been successfully configured as an HPS/FPGA system.

3.4.3 Executing Applications (with embedded Linux - Yocto)

1. Switch back to the minicom (or DS-5 terminal) serial terminal. In the terminal, type in:

   launcher HexLEDsSW

   This will execute the binary (we cross-compiled in DS-5) as an application on your HPS/FPGA system. You should see the LEDs rotate and blink, and the HEX switch numbers and display a message at the end. To stop execution, simply press CTRL-C. The status of the SoC will also be displayed in the serial terminal.

2. Analyze the main.c file and it's support code files. Read the comments and understand the logistics of creating a software application for virtual memory mapping a HPS/FPGA system in Linux.

4. What to Hand In

Using the techniques learned throughout this tutorial, students are to design a custom HPS/FPGA SoC. The FPGA hardware prototype should consist of:

- 10 LEDs
- 10 Switches
- HEX display

The .c code running on the HPS must provide the following functionality:

- Upon startup of the application, a "light show" will occur with the LEDs and HEX display
- Once the light show has completed:
    - the HEX should display a message (other than "HELLO")
    - When a switch is pressed up, it's corresponding LED should light. When switched down, the LED should also turn off accordingly.
- Your program should display information pertaining to the mode and/or state your SoC system is during execution in the serial terminal.

This lab is due week 7. Please print out and hand in:

- All .c code implemented for your software design
- The hps_0.h file generated with the NIOS II command shell
- The top-level VHDL you have used to design your hardware prototype
- A screenshot of your QSys system with all associated IP blocks and their respective connections.
- A screenshot of the terminal window for a sample execution of your SoC system.

The lab must be handed in with the Ryerson University cover page, dated and signed. Your lab instructor will also ask you to demo your work during submission. Be prepared.

Note: The switches and their state will need to be read using the Altera API. The

uint32_t alt_read_word(uint32_t address);

may be used to access a switch(es) state.

Hints:

-Several other API functions exist including *alt_setbits_word()* and *alt_clrbits_word()* that may help in your code design. To access these function declarations: In DS-5, hover over an instance of *alt_write_word(..)* and press CTRL and left-click the mouse. This will create a link and DS-5 will navigate to the function's declaration. This declaration file will also contain multiple API functions you will likely need for your lab.

- The Switch(es) pin names that must be used in your VHDL can be found in pin_assignment_DE1-SoC.tcl.

Appendix

```
------------------------------
-- HED_LED_FPGA Tutorial
-- Lab 3
-- COE838 Systems-on-Chip Design
-- Created By: Anita Tino
------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY LED_HEX_FPGA IS
        PORT( CLOCK_50, HPS_DDR3_RZQ,HPS_ENET_RX_CLK, HPS_ENET_RX_DV : IN STD_LOGIC;
              HPS_DDR3_ADDR                        : OUT STD_LOGIC_VECTOR(14 DOWNTO 0);
              HPS_DDR3_BA                          : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
              HPS_DDR3_CS_N                               : OUT STD_LOGIC;
              HPS_DDR3_CK_P, HPS_DDR3_CK_N, HPS_DDR3_CKE    : OUT STD_LOGIC;
              HPS_USB_DIR, HPS_USB_NXT, HPS_USB_CLKOUT              : IN STD_LOGIC;
              HPS_ENET_RX_DATA                     : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
              HPS_SD_DATA, HPS_DDR3_DQS_N          : INOUT STD_LOGIC_VECTOR(3 DOWNTO 0);
              HPS_DDR3_DQS_P                       : INOUT STD_LOGIC_VECTOR(3 DOWNTO 0);
              HPS_ENET_MDIO                               : INOUT STD_LOGIC;
              HPS_USB_DATA                                : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
              HPS_DDR3_DQ                          : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
              HPS_SD_CMD                           : INOUT STD_LOGIC;
              HPS_ENET_TX_DATA, HPS_DDR3_DM        : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
              HPS_DDR3_ODT, HPS_DDR3_RAS_N, HPS_DDR3_RESET_N        : OUT STD_LOGIC;
              HPS_DDR3_CAS_N, HPS_DDR3_WE_N                                 : OUT STD_LOGIC;
              HPS_ENET_MDC, HPS_ENET_TX_EN                                  : OUT STD_LOGIC;
              LEDR                                 : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
              HEX0, HEX1, HEX2, HEX3, HEX4, HEX5: BUFFER STD_LOGIC_VECTOR(6 DOWNTO 0);
              HPS_USB_STP, HPS_SD_CLK, HPS_ENET_GTX_CLK                   : OUT STD_LOGIC);
END LED_HEX_FPGA;

ARCHITECTURE Behaviour OF LED_HEX_FPGA IS
    component soc_system is
        port (clk_clk                           : in    std_logic                     := 'X';
            hps_0_h2f_reset_reset_n           : out   std_logic;
            hps_io_hps_io_emac1_inst_TX_CLK   : out   std_logic;
            hps_io_hps_io_emac1_inst_TXD0     : out   std_logic;
            hps_io_hps_io_emac1_inst_TXD1     : out   std_logic;
            hps_io_hps_io_emac1_inst_TXD2     : out   std_logic;
            hps_io_hps_io_emac1_inst_TXD3     : out   std_logic;
            hps_io_hps_io_emac1_inst_RXD0     : in    std_logic                     := 'X';
            hps_io_hps_io_emac1_inst_MDIO     : inout std_logic                     := 'X';
            hps_io_hps_io_emac1_inst_MDC      : out   std_logic;
            hps_io_hps_io_emac1_inst_RX_CTL   : in    std_logic                     := 'X';
            hps_io_hps_io_emac1_inst_TX_CTL   : out   std_logic;
            hps_io_hps_io_emac1_inst_RX_CLK   : in    std_logic                     := 'X';
            hps_io_hps_io_emac1_inst_RXD1     : in    std_logic                     := 'X';
            hps_io_hps_io_emac1_inst_RXD2     : in    std_logic                     := 'X';
            hps_io_hps_io_emac1_inst_RXD3     : in    std_logic                     := 'X';
            hps_io_hps_io_sdio_inst_CMD       : inout std_logic                     := 'X';
            hps_io_hps_io_sdio_inst_D0        : inout std_logic                     := 'X';
            hps_io_hps_io_sdio_inst_D1        : inout std_logic                     := 'X';
            hps_io_hps_io_sdio_inst_CLK       : out   std_logic;
            hps_io_hps_io_sdio_inst_D2        : inout std_logic                     := 'X';
            hps_io_hps_io_sdio_inst_D3        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D0        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D1        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D2        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D3        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D4        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D5        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D6        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_D7        : inout std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_CLK       : in    std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_STP       : out   std_logic;
            hps_io_hps_io_usb1_inst_DIR       : in    std_logic                     := 'X';
            hps_io_hps_io_usb1_inst_NXT       : in    std_logic                     := 'X';
            memory_mem_a                      : out   std_logic_vector(14 downto 0);
            memory_mem_ba                     : out   std_logic_vector(2 downto 0);
            memory_mem_ck                     : out   std_logic;
            memory_mem_ck_n                   : out   std_logic;
            memory_mem_cke                    : out   std_logic;
            memory_mem_cs_n                   : out   std_logic;
            memory_mem_ras_n                  : out   std_logic;
            memory_mem_cas_n                  : out   std_logic;
            memory_mem_we_n                   : out   std_logic;
            memory_mem_reset_n                : out   std_logic;
            memory_mem_dq                     : inout std_logic_vector(31 downto 0) := (others => 'X');
```

```
            memory_mem_dqs                    : inout std_logic_vector(3 downto 0)  := (others => 'X');
            memory_mem_dqs_n                  : inout std_logic_vector(3 downto 0)  := (others => 'X');
            memory_mem_odt                    : out   std_logic;
            memory_mem_dm                     : out   std_logic_vector(3 downto 0);
            memory_oct_rzqin                  : in    std_logic                     := 'X';
            reset_reset_n                     : in    std_logic                     := 'X';
            seg7_if_0_conduit_end_export      : out   std_logic_vector(47 downto 0);
            led_pio_external_connection_export : out   std_logic_vector(9 downto 0));
    end component soc_system;

SIGNAL hex5_tmp, hex4_tmp, hex3_tmp, hex2_tmp, hex1_tmp, hex0_tmp, hps_0_h2f_reset_reset_n : STD_LOGIC;

BEGIN
 u0 : component soc_system
        port map (
            clk_clk                           => CLOCK_50,
            reset_reset_n                     => '1',
            memory_mem_a                      => HPS_DDR3_ADDR,
            memory_mem_ba                     => HPS_DDR3_BA,
            memory_mem_ck                     => HPS_DDR3_CK_P,
            memory_mem_ck_n                   => HPS_DDR3_CK_N,
            memory_mem_cke                    => HPS_DDR3_CKE,
            memory_mem_cs_n                   => HPS_DDR3_CS_N,
            memory_mem_ras_n                  => HPS_DDR3_RAS_N,
            memory_mem_cas_n                  => HPS_DDR3_CAS_N,
            memory_mem_we_n                   => HPS_DDR3_WE_N,
            memory_mem_reset_n                => HPS_DDR3_RESET_N,
            memory_mem_dq                     => HPS_DDR3_DQ,
            memory_mem_dqs                    => HPS_DDR3_DQS_P,
            memory_mem_dqs_n                  => HPS_DDR3_DQS_N,
            memory_mem_odt                    => HPS_DDR3_ODT,
            memory_mem_dm                     => HPS_DDR3_DM,
            memory_oct_rzqin                  => HPS_DDR3_RZQ,
            hps_io_hps_io_emac1_inst_TX_CLK   => HPS_ENET_GTX_CLK,
            hps_io_hps_io_emac1_inst_TXD0     => HPS_ENET_TX_DATA(0),
            hps_io_hps_io_emac1_inst_TXD1     => HPS_ENET_TX_DATA(1),
            hps_io_hps_io_emac1_inst_TXD2     => HPS_ENET_TX_DATA(2),
            hps_io_hps_io_emac1_inst_TXD3     => HPS_ENET_TX_DATA(3),
            hps_io_hps_io_emac1_inst_RXD0     => HPS_ENET_RX_DATA(0),
            hps_io_hps_io_emac1_inst_MDIO     => HPS_ENET_MDIO,
            hps_io_hps_io_emac1_inst_MDC      => HPS_ENET_MDC,
            hps_io_hps_io_emac1_inst_RX_CTL   => HPS_ENET_RX_DV,
            hps_io_hps_io_emac1_inst_TX_CTL   => HPS_ENET_TX_EN,
            hps_io_hps_io_emac1_inst_RX_CLK   => HPS_ENET_RX_CLK,
            hps_io_hps_io_emac1_inst_RXD1     => HPS_ENET_RX_DATA(1),
            hps_io_hps_io_emac1_inst_RXD2     => HPS_ENET_RX_DATA(2),
            hps_io_hps_io_emac1_inst_RXD3     => HPS_ENET_RX_DATA(3),
            hps_io_hps_io_sdio_inst_CMD       => HPS_SD_CMD,
            hps_io_hps_io_sdio_inst_D0        => HPS_SD_DATA(0),
            hps_io_hps_io_sdio_inst_D1        => HPS_SD_DATA(1),
            hps_io_hps_io_sdio_inst_CLK       => HPS_SD_CLK,
            hps_io_hps_io_sdio_inst_D2        => HPS_SD_DATA(2),
            hps_io_hps_io_sdio_inst_D3        => HPS_SD_DATA(3),
            hps_io_hps_io_usb1_inst_D0        => HPS_USB_DATA(0),
            hps_io_hps_io_usb1_inst_D1        => HPS_USB_DATA(1),
            hps_io_hps_io_usb1_inst_D2        => HPS_USB_DATA(2),
            hps_io_hps_io_usb1_inst_D3        => HPS_USB_DATA(3),
            hps_io_hps_io_usb1_inst_D4        => HPS_USB_DATA(4),
            hps_io_hps_io_usb1_inst_D5        => HPS_USB_DATA(5),
            hps_io_hps_io_usb1_inst_D6        => HPS_USB_DATA(6),
            hps_io_hps_io_usb1_inst_D7        => HPS_USB_DATA(7),
            hps_io_hps_io_usb1_inst_CLK       => HPS_USB_CLKOUT,
            hps_io_hps_io_usb1_inst_STP       => HPS_USB_STP,
            hps_io_hps_io_usb1_inst_DIR       => HPS_USB_DIR,
            hps_io_hps_io_usb1_inst_NXT       => HPS_USB_NXT,
            hps_0_h2f_reset_reset_n           => hps_0_h2f_reset_reset_n,
           led_pio_external_connection_export => LEDR,
         seg7_if_0_conduit_end_export(47)    => hex5_tmp,
         seg7_if_0_conduit_end_export(46 DOWNTO 40)=>HEX5,
         seg7_if_0_conduit_end_export(39)                    =>hex4_tmp,
         seg7_if_0_conduit_end_export(38 DOWNTO 32)=>HEX4,
         seg7_if_0_conduit_end_export(31)                    =>hex3_tmp,
         seg7_if_0_conduit_end_export(30 DOWNTO 24)=>HEX3,
         seg7_if_0_conduit_end_export(23)                    =>hex2_tmp,
         seg7_if_0_conduit_end_export(22 DOWNTO 16)=> HEX2,
         seg7_if_0_conduit_end_export(15)                    => hex1_tmp,
         seg7_if_0_conduit_end_export(14 DOWNTO 8) => HEX1,
         seg7_if_0_conduit_end_export(7)                       => hex0_tmp,
         seg7_if_0_conduit_end_export(6 DOWNTO 0)  =>HEX0
        );

End Behaviour;
```