
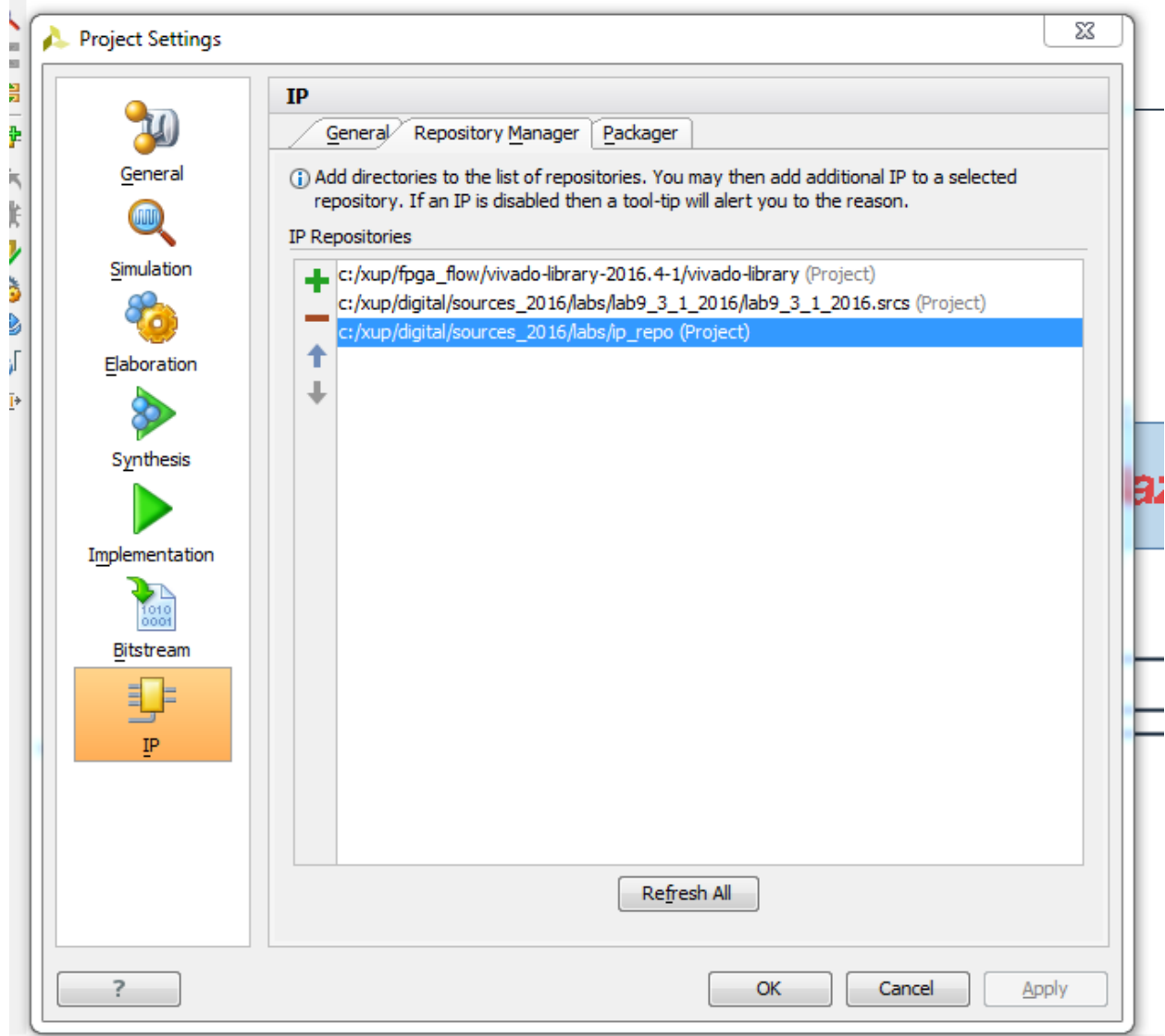


A few helpful hints if you want to use custom IP and/or PMOD connector modules for your final project (note: instructions are related to a stop watch project that was done in previous years but you can use the same basic approach for your final project):

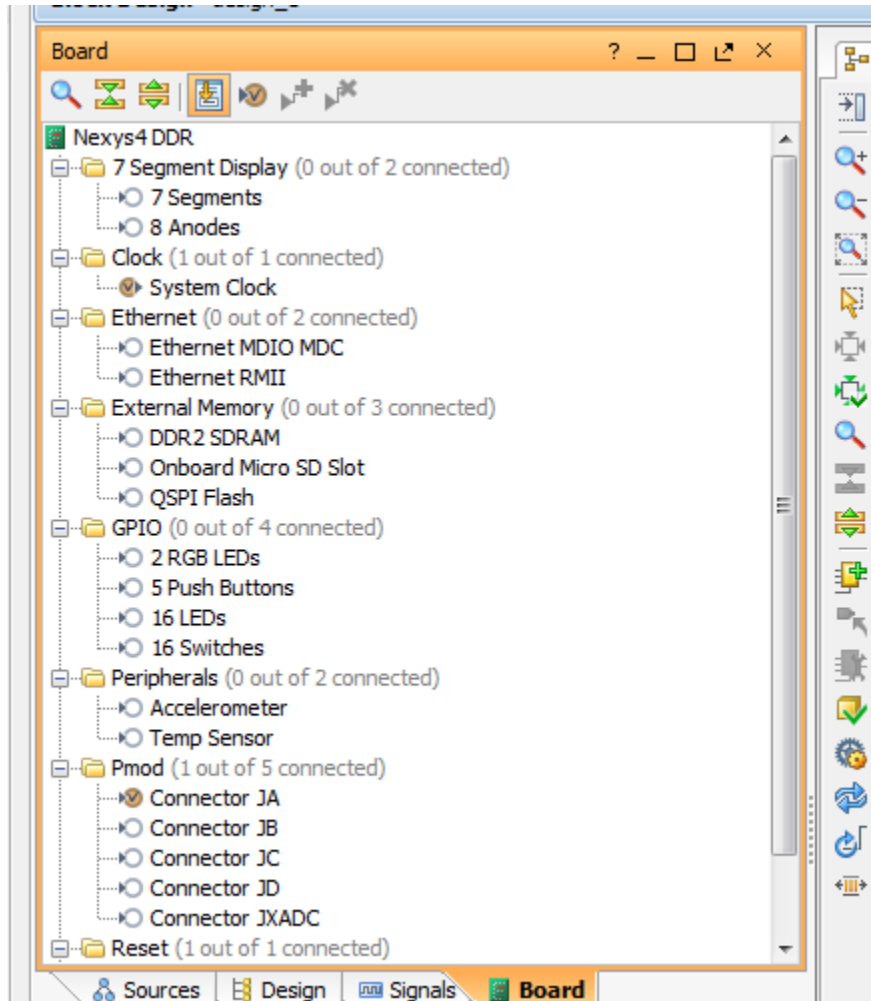
1. When you create custom IPs they will be put in an “ip_repo” directory in your vivado root directory (i.e. where vivado by default looks for projects). You can add this ip_repo as a repository to your design by opening or creating a block design and then clicking on the “ip settings icon”  and then selecting the IP menu and repository manager tab and clicking on the green + symbol to add a repository (see figure below third line of IP Repositories)



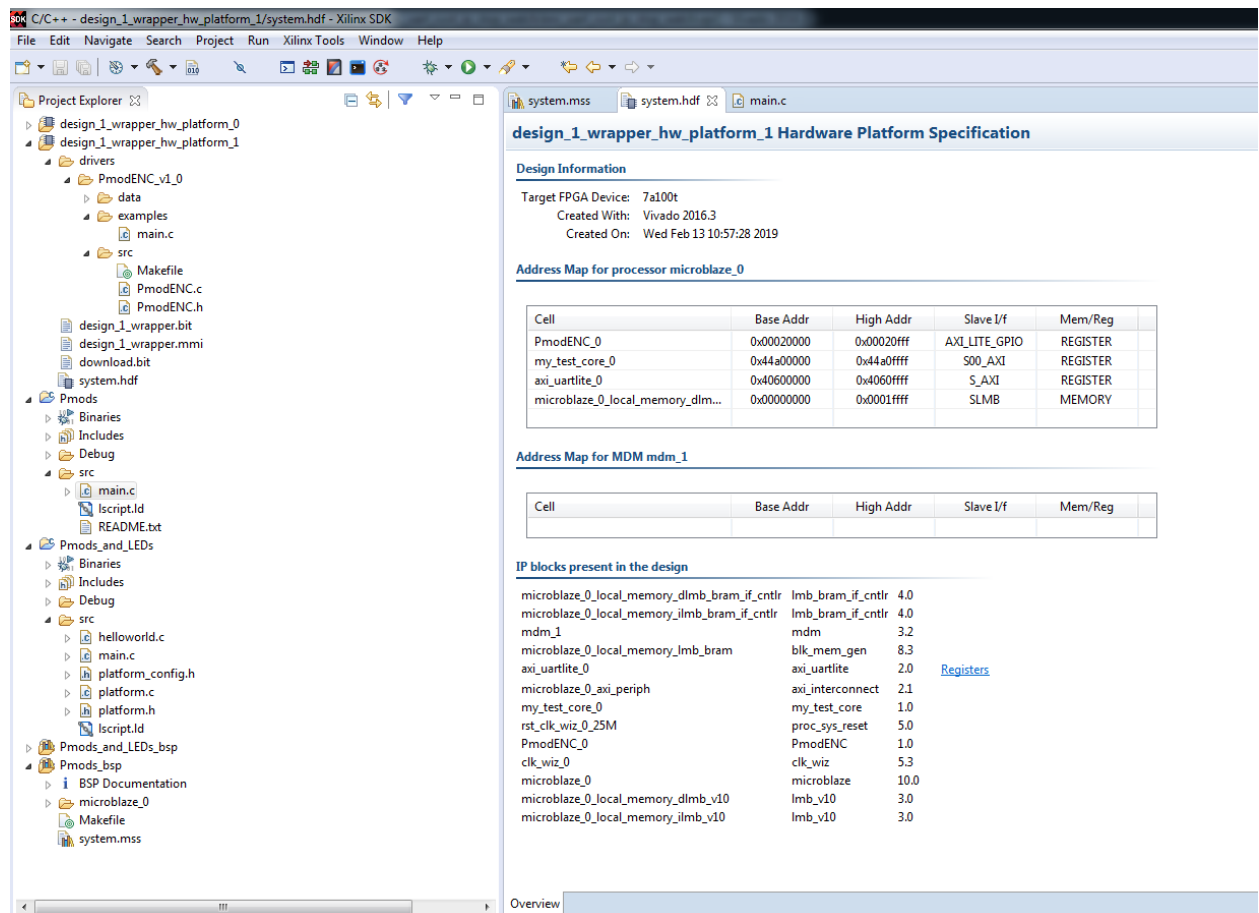
2. You can create and IP for an entire project by clicking on the Package IP icon of the flow navigator. So for example, you could modify the rudimentary stop watch project you did in lab9_3_1 and then package the whole project as an IP that you can load into your top level design. Note: when you do this the new IP is not put in the “ip_repo” directory but instead is

placed in the project directory you are using so you need to add the .../project_name/project_name.srscs directory as a new repository (see figure above second line of IP Repositories).

3. In the Pmod tutorial you used the “board” menu to select which IOs you would use (see figure below). You then created a design wrapper which automatically configured the IO without requiring a constraint (.xdc) file. For the stopwatch project, you will want to use the board menu items for connection to the Pmod, UART, etc. but you will also need an (.xdc) file for connections to the LEDs, seven segment displays, etc. Note: the (.xdc) file should only have the I/Os that are not already defined using the “board” menu selections.



4. When you make a copy of a project or modify a project and then create a new design wrapper, when you open up the SDK it will still have the old design wrapper platform (see figure below) . To use the new platform you need to select the new platform and right click and create a new project. This will create a new system.hdf (hardware platform specification) and system.mss file (board support package info) that will be used for your new/modified hardware. Also note in the figure below the system.hdf file contains the memory map addresses that you can use in your software to read from the Pmod encoder and drive your stopwatch hardware.



- In the custom IP tutorial you created an AXI interface with some added user logic to control LEDs on the Nexys4DDR board. The LEDs were controlled using the built in 32 bit slave registers of the AXI interface (see `slv_reg0`, `slv_reg1`, `slv_reg2` and `slv_reg3` highlighted in red in code example from `My_PWM_Core_v1_0_S00_AXI.v` below). You can use these same register to control you stop watch circuits. Note: In the SDK these were addressed as `MY_PMW`, `MY_PMW+4`, `MY_PMW+8` and `MY_PMW+12` (see code example below for 32 bit writes from `main.c` of custom IP tutorial) as the microblaze processor uses an 8 bit data bus.

```
// Add user logic here
reg [15:0] counter = 0;

//simple counter
always @(posedge S_AXI_ACLK) begin
    if(counter < PWM_COUNTER_MAX-1)
        counter <= counter + 1;
    else
        counter <= 0;
end

//comparator statements that drive the PWM signal
assign PWM0 = slv_reg0 < counter ? 1'b0 : 1'b1;
```

```
assign PWM1 = slv_reg1 < counter ? 1'b0 : 1'b1;  
assign PWM2 = slv_reg2 < counter ? 1'b0 : 1'b1;  
assign PWM3 = slv_reg3 < counter ? 1'b0 : 1'b1;
```

```
// User logic ends
```

SDK writes to 32 bit slave registers:

```
Xil_Out32(MY_PWM, num);  
Xil_Out32((MY_PWM+4), num);  
Xil_Out32((MY_PWM+8), num);  
Xil_Out32((MY_PWM+12), num);
```