ECE 544: FreeRTOS system using Board Files

(Example: Creating a FreeRTOS supporting System Using Board Files)

Vivado 2018.x

Revision 2.1

Updated by

Roy Kravitz (rkravitz@pdx.edu)

Original Version by

Sai Teja Bodanki

Maseeh College of Engineering and Computer Science 27-April-2018

Department of Electrical and Computer Engineering Revision 2.1

Free RTOS System:

A FreeRTOS needs at-lest the following Hardware support

- Memory - DDR SDRAM + On-Chip BRAM (total 128Mb + 128KB)
- Interrupt Controller
- Timer

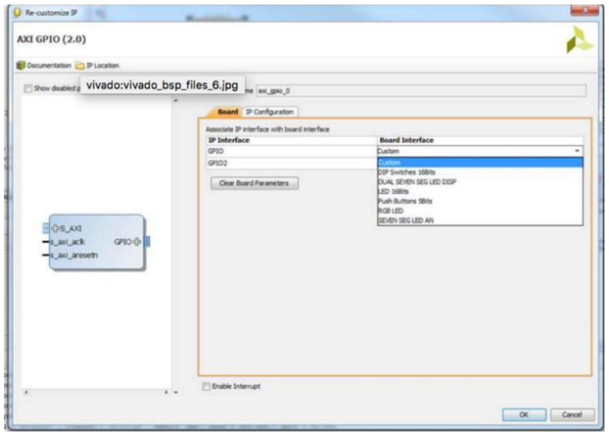
We are going to build our system using the Digilent board files. This is a different approach than we used on Project #1 where we created the embedded system and instantiated it in our own top level module (`n4fpga.v`) and constraints file (`n4DDRfpga.xdc`).

Vivado Board Files

In addition to the ability (requirement) to select a Xilinx FPGA when setting up a new Vivado project, Vivado also allows you to select a board type for FPGA development boards like the Nexys A7 (formerly the Nexys4 DDR).

By default, Vivado includes XML files for the different FPGA boards manufactured by Xilinx. These XML files define the different interfaces on the board. Interfaces such as Slide Switches, Push Buttons, LEDs, USB-UART, DDR Memory, Ethernet etc.

Digilent has created XML files for the Artix 7 FPGA boards including the Nexys 4 DDR and the Nexys A7. The XML files connect various IP blocks to specific pins on the Digilent FPGA platforms. For example, in the AXI GPIO IP customization window, you will be able to assign different interfaces that are available on your selected board to a specific GPIO IP block:



Associate IP interface with board interface

IP Interface	Board Interface
GPIO	Custom
GPIO2	

Board files make the designer's (that's you) task of creating an embedded system with the IP Integrator a little simpler and, potentially, with less errors.

- The right chip (like DDR SDRAM) with recommended configurations are set by default.
- No need to add pin constraints (XDC files are automatically generated)
- Faster block level design. (Don't worry about port names matching)
- The FPGA device is automatically selected.

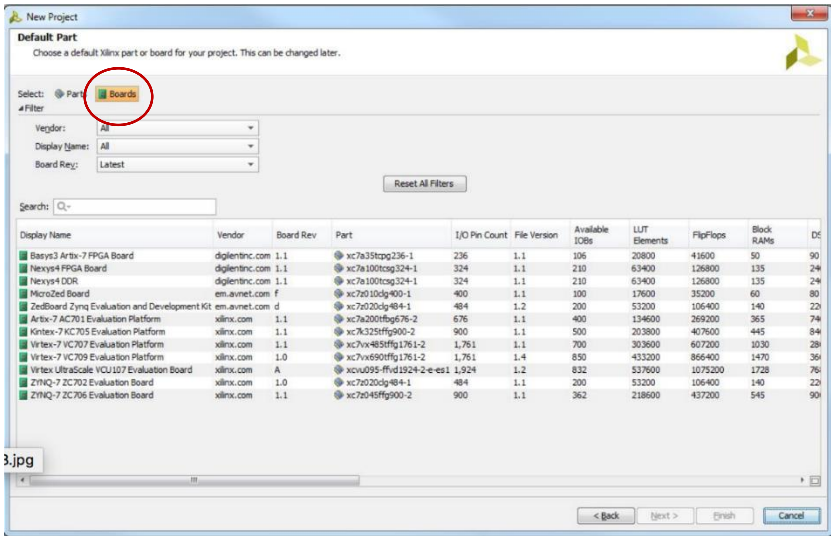
- Can concentrate more on Embedded system, rather than worrying about board set-up.

Board packages are limited to existing FPGA development boards that support them. If you are to work on a Custom FPGA Board you have to start from scratch much like we did in Project #1.

Installing the Digilent Board Files:

Refer to the following link for instructions on how to install the Digilent Board files into Vivado: https://reference.digilentinc.com/vivado/installing-vivado/start#installing_digilent_board_files

You should be able to see updated board files with Nexys A7 and Nexy4 DDR and various Digilent supported boards included in the list when you create a new project. If you don't see them try restarting Vivado.



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Vendor: All
Display Name: All
Board Rev: Latest
Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DS
Basys3 Artix-7 FPGA Board	digilentinc.com	1.1	xc7a35tcpg236-1	236	1.1	106	20800	41600	50	90
Nexys4 FPGA Board	digilentinc.com	1.1	xc7a100tcsg324-1	324	1.1	210	63400	126800	135	24
Nexys4 DDR	digilentinc.com	1.1	xc7a100tcsg324-1	324	1.1	210	63400	126800	135	24
MicroZed Board	em.avnet.com	f	xc7z010clg400-1	400	1.1	100	17600	35200	60	80
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.2	200	53200	106400	140	22
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676	1.1	400	134600	269200	365	74
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.1	500	203800	407600	445	84
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.1	700	303600	607200	1030	28
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.4	850	433200	866400	1470	36
Virtex UltraScale VCU107 Evaluation Board	xilinx.com	A	xcvu095-ffvd1924-2-e-es1	1,924	1.2	832	537600	1075200	1728	76
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.1	200	53200	106400	140	22
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.1	362	218600	437200	545	90

Steps to Build the System

Step	Screen	Action	Explanation/Comments
1	Vivado Home	Create a new Project	You know the Drill
2	Part Select	Select the Boards tab and pick the board of your interest	NEXYS 4 DDR
3	Create New Block Design	In the Window where you can see Source|Design|Signal|Board Click Board.	You will see all the available Peripherals connected on your board.
4	Click Clock folder→systemclock	Click system clock. (click ok with default config) A Clocking Wizard appears. Pick the configurations based on previous configurations (project 1)	(Add a 100Mhz, 50Mhz, and a 200Mhz(for DDR SDRAM))
5	Click Reset	Click OK with default parameters	NA
6	Add IP	Add a MicroBlaze IP (→Run Block Automation)	With the following configuration: • Local Mem -128KB • Cache -32KB • Debug and UART • Check – Interrupt Controller
7	Click External Memory/DDR2 SDRAM (Board tab)	Click Memory Interface Generator Block	Set the following configurations • →Next (Default) • Until you see a Change period window. Set to 5000 (200MhZ) • → Next (Default) • Until You see a Validate button. Press Validate (You should see Current Pinout is valid) • → Next, Agree • Done
8	Run Connection Automation	You should see a window where you can select `sys_clock` of `MIG_7_`Series	Click the sys_clock and click the drop box to set to

			the clock wizard 200Mhz clock Note: I (Roy) did not see this choice the first time but I was able to delete the MIG clock on the block diagram which brought up the Run Connection Wizard dialog where I could select the clock.
9	Add IP	Add two 16-bit GPIO's and an AXI Timer to your design. The GPIO that will be connected to the DIP switches should generate interrupts. Note: You can add these GPIO's from the board tab if you want to. In my (Roy) implementation I put the 16 LEDS to a GPIO (became GPIO 0) and the switches and buttons on a separate GPIO (became GPIO 1). After creating GPIO 1 I customized it to enable interrupts and routed the GPIO input to the 2nd input on the XLCONCAT block; the first input was the output from the timer.	These are the peripherals you need to run the FreeRTOS demo app we included. The AXI Timer 0 interrupt should be connected to interrupt controller. LED's should be connected to GPIO_0 and the DIP switches should be connected to GPIO 1. The Interrupt output from GPIO 1 should be connected to the interrupt controller through XLCONCAT and the GPIO configured to generate an interrupt. The interrupt will be asserted each time any of the GPIO inputs changes.

Note: In the Address Editor you see the Instruction has MIG and BRAM. Anything other than the BRAM and MIG can be excluded from the Instruction segment. (Right click and Select Exclude).

- You can now Save your design and Generate Output Products (this will take a while) and go to sources.
- Right Click on your design and select create HDL wrapper. Select the option that allows you to edit the HDL wrapper file.
- Let Vivado pick the top and select ok.

- Synthesize, Implement, Generate Bitstream, Export hardware (and bitstream) to SDK

Building and running the Getting Started App in SDK

The process is not that different than creating an app for the standalone OS. Build a FreeRTOS BSP instead of a Standalone OS BSP. Create a new application using the FreeRTOS BSP. Note that the FreeRTOS BSP relies on the standalone OS so all of the standalone functionality is there for you to use…however, leave the interrupt setup to FreeRTOS. It has its own mechanisms. Once your app starts the FreeRTOS scheduler its “rules” apply, not those of the Standalone OS.

I started by building a Hello World app when I created the application for FreeRTOS. It worked out-of-the-box showing me that the hardware configuration and BSP I built worked. This app sends a message between two tasks and quits after 10 seconds.

After I had the Hello World app working I built a new app on the same FreeRTOS BSP and imported the Getting Started test program. It also worked after a fashion so I made some changes to it. The app starts off by blinking the LEDs and then waits for you to flip one of the switches. Every time a switch is flipped the LED pattern changes from 0x00FF to 0xFF00.

A few words of caution. These are things I ran into, your mileage may vary. ☺

- I had to reload the FPGA every time I finished running the app. Even though the reset signal is routed to the CPU reset button on the board my implementation must not have cleanly reinitialized the board. I didn’t spend any time figuring this out. If you do than please post your findings.
- I don’t think the caches are enabled even though they are included in the design. It seemed that my calls to sleep() took much longer than the 1 second I asked form. I didn’t explore this either since I had something that worked and I wanted to post this Getting Started guide.
- I could not terminate an app running in the Debug perspective. That terminate option wasn’t available to me.
- I occasionally had trouble getting the debugger to start correctly. Reloading the FPGA usually fixed this but one time I had to relaunch SDK. I have seen other strangenesses with Eclipse on Windows where it doesn’t seem to shut down cleanly.

Important notes for Project #2

Note: Although the Board files can be used to connect any of the board-specific I/O devices (LEDs, RGB LEDs, 7-Segment Display, buttons and witches), this guide only used the board file to include DDR2 SDRAM and the peripherals needed to run the FreeRTOS demo program.

You may want to follow the Project #1 approach using the IP Integrator to connect/customize the other IP blocks that your embedded system needs. (Nexys4IO, AXI Timer, GPIO's, Pmods):

- You will continue adding Pmod's like you've done with the Project 1.
- You will use the same constraint file released in the Project #1 but should only uncomment the pins that you added at the top level. Board-specific constraint files are generated automatically for anything added using the Board tab in the IP Integrator (ex: the SDRAM)
- You can re-edit or create a fresh top module for Project 2. Make sure you have the port names same as constraints file. It may be helpful to refer to the n4fpga.v top level files for Project #1. For example, the Pmod ports are bi-directional.
- You may have to change the directions of port list based on the connectivity of PmodHB3 (Dir and EN are outputs from the embedded system, SA and SB are inputs.
- Do not disturb the PmodOLED (JA) or PmodENC(JD). Use JC or JB for PmodHB3. Thus you need to re-edit the port list of only JC/JB and leave JA and JD as it is.