Getting Started in ECE 544: Building a Microblaze MCS Embedded System

Vivado and Vitis 2022.x
Nexys A7 (Nexys4 DDR)
Revision 1.4 (Last updated: 10-January-2025)

Roy Kravitz

Disclaimer

I have always felt that learning by mistakes is a good way to learn. So don't worry if you can't get your system to work in the first try; consult with your colleagues, make mistakes, learn, and try again. If you really get stuck, post your problem(s) to the Canvas discussion forum…and your solution(s) after you figure it out. If you find any problems with this write-up, please let me know.

Additional Disclaimer

This is the first update to the Getting Started guide since I rolled out this documentation and release package for Vivado 2022.x so please be patient. Use the discussion forum and let me know of any problems or things that are hard to understand and/or wrong in this guide.

Table of Contents

Revision History

Revision	By	Date	Description
1.0	RK	01-Jan-2023	First Release. Created from the ECE 544 Getting Started Guide Rev. 9.0
1.1	RK	06-Jan-2023	Changes for asynchronous reset in rgbPWM and a few others, all minor
1.2	RK	08-Jan-2023	Incorporated suggestions from Mehul, all minor
1.3	RK	10-Jan-2024	Ported to Vivado 2022.x. No functional changes
1.4	RK	10-Jan-2025	Minor updates. No functional change

Using this guide

The purpose of this guide/tutorial is to help you create your first Microblaze-based embedded system project. This document does not replace the Xilinx documentation, nor is it intended to be an exhaustive How-To, but my hope is that this document will make your first excursion into generating and writing applications for embedded systems in the Vivado/Vitis environment a bit less mysterious and a whole lot less overwhelming.

I have tried to provide step-by-step instructions to guide you through the process of creating a target hardware platform and a software application. While you may be tempted to blindly follow the steps, I encourage you **not to**. You will be using Vitis and Vivado and the IP Integrator throughout the entire term – you need to understand how to make these complex, and occasionally quirky, tools do your bidding. Blindly following the steps in this guide without attempting to understand the process and what the tools are doing for you is a recipe for problems later in the term. You do not want to be learning how to use the tools two nights before a project is due. Explore the options available to you as you work through this guide. Take the time to read the documentation (and you will be reading lots of it) and watch the Xilinx Quick Take videos. Work through the process on a system of your own creation. Using this guide will get you going but it will not really make you a master of the tool chain. That is up to you.

A FINAL NOTE. When you get stuck (and I can almost guarantee that you will at some point during the term) we will try to help but we are not omniscient. We must do the same things you should be doing – examine the log files, try to make sense out of obscure error messages, search Google, search the Xilinx knowledge base and search the Xilinx User forums. We simply cannot do that for everybody in the class so try hard to figure out the problem by yourself before giving up. If you do give up, post your problem in the discussion forums on Canvas – it is possible somebody else encountered and overcame a similar problem. These tools work most of the time so as much as you would like to blame the tools or cast dispersion on Xilinx or blame the instructor or the documentation – look closely at your work…like it or not, the problem is probably there. ‘Nuf said.

The Xilinx embedded tool chain

The Xilinx embedded tool chain consists of two major GUI-oriented applications. Xilinx provides additional components that are not part of the process for creating Xilinx FPGA-based embedded systems but are integrated into Vivado. These ancillary components include the integrated logic simulator, the integrated logic analyzer and the IP Create and Package wizard.

The major components in the Xilinx embedded tool chain are:

- **Vivado** – This is the project manager and synthesis and place and route tool for Xilinx FPGA-based designs. It is the core of Xilinx’s tool technology. The Vivado Project Manager is used to create and add HDL and constraints into a project and run the synthesis, place and route, timing analysis and bitstream generation and download processes. The Vivado GUI actions are TCL-based so if you are familiar w/ TCL you

may find it easier to type commands directly into the TCL console built into Vivado.

- **IP Integrator (IPI)** – IP Integrator is used to generate embedded system hardware. It provides a block diagram-based canvas to create Microblaze (used in this course) or Zynq (ARM core)-based systems customized for your application. Xilinx includes a rich set of peripheral modules (called Xilinx IP) that can be added to your system and interconnected through a connection wizard in the IPI. You can also include third-party IP and your own custom peripherals. In fact, you will use several IP blocks provided by third parties (including myself).

  As a SoC system designer, you want to treat the Xilinx IP and third-party IP as black-boxes with well-defined interfaces which your application accesses through vendor-provided drivers. The good thing about IP Integrator is that it greatly simplifies creating the embedded system for your application so you can focus on your application-specific hardware. The bad news is there is a lot of behind-the-scenes processing going on – when everything works well it is way-cool, but when it does not…well, be prepared to spend time on Google, in the Xilinx knowledge base and user forums, poring through pages of log files and reading all sorts of documentation. Unfortunately, there is always at least one student in the course that has inexplicable trouble with the tools – here's to hoping that student is not you.

- **Vitis –** This is the Xilinx Software Development Kit. Vitis is based on the Eclipse platform (an open-source industry standard) and the GNU tool chain. The Eclipse-based GUI for Vitis provides the mechanisms to create and debug assembly language, C and C++ application projects and code. Vitis is tightly integrated into the Vivado flow, however. To use Vitis you first import a hardware description file (`.xsa`) from Vivado and build one or more "platforms" for your application. A platform includes a board support package containing the OS and drivers that will be used in your application. Once you have a platform package for your specific hardware you can create and debug application(s) that run on the platform you created.

  This guide uses the "standalone OS" environment. The "standalone" environment provides the framework for the Xilinx drivers, most notably support for interrupt handling. Vitis has the capability to add and configure the standalone OS and the FreeRTOS real-time kernel into your application along with additional libraries like LWIP (TCP/IP) and RAM- and Flash-based file systems. It is also possible to create a Linux-based system image using separate "Petalinux" tools and utilities, but we will not be doing that in the assigned projects. Fair game for a final project, though.

Building a Microblaze MCS-based embedded system

Vivado, IPI and Vitis work together to provide a tightly coupled environment for building SoC hardware and software for Xilinx FPGAs. You (the design engineer/architect) make use of these tools to specify the hardware to be used by your software application. After the hardware has been synthesized and Implemented (Xilinx-speak for place and route) you create the software drivers and application program(s) for your target application. This guide will lead you through the tasks needed to build an HDL implementation of a full-blown embedded system platform consisting of a 32-bit embedded microcontroller (Microblaze MCS), an IO module, and custom hardware . You will test the system by compiling, linking, and loading an application which makes use of the hardware and the associated drivers.

The target application for this Getting Started guide uses a Microblaze application and custom PWM (pulse width modulation) hardware to control the intensity of the red, green, and blue segments of RGB0 (one of the two Red-Green-Blue LEDs on the Nexys A7)

NOTE: Digilent renamed the Nexys 4 DDR board to Nexys A7 in 2019. Hardware-wise they are compatible with each other.

Building the hardware and executing the test program is a multiple step process. The major tasks are:

1. Create a Vivado RTL project that includes a 3-channel PWM generator written in Verilog 2001 and a constraints file in the project that maps the ports in your top-level design with the pins on the FPGA that are connected to the devices (LEDs. Switches, etc.) on the Nexys A7 board.
2. Create the embedded system using the Create Block Design wizard (IP Integrator)
3. Configure the Microblaze MCS IOModule and create the external ports for the embedded system (IP Integrator)
4. Synthesize the embedded system from the IPI block diagram of your embedded system (Vivado)
5. Synthesize, place and route and generate configuration files for the FPGA (Vivado)
6. Export the target hardware platform to Vitis and create a platform project and BSP (Board Support Package) for your application.
7. Import and execute the test application (Vitis)

These tasks are described in more detail in the sections that follow.

Task 0 – Get familiar with Vivado and IP Integrator

Simply put, the rest of this document will make the most sense only if you have done some prep work. Those of you who have used the Xilinx tools in the past are most likely familiar with Vivado. You know much of what you need to know, except perhaps how to use IPI to generate your target embedded system. Fortunately doing that is a few clicks away.

For those readers who are new to the Xilinx tool chain and Vivado, your best bet is to visit https://www.amd.com/en.html . There are several documents and videos to help you get started. Here is a good jumping off place:

https://www.amd.com/en/developer/browse-by-resource-type/training.html

For those readers who are familiar with the Xilinx tool chain or ASIC and FPGA tool chains from other vendors, DocNav (the Vivado document catalog) provides a good jumping off point. DocNav can be started from the Vivado opening screen or from the Start Menu and provides access to tutorials, videos, reference manuals, user guides and the like. We prefer to start in the *Design Hub* view, but the *Catalog* view provides a search function with filtering to narrow down the list. Consider watching a few of the Vivado Quick Take videos; they provide a good introduction. Work through one or more of the tutorials before diving into the user manuals and reference guides. There are literally thousands of pages of documentation – it is overwhelming but, please, do not skip this type of preparation and simply dive into this Getting Started guide. If you do that you will quickly get lost in the details. It would not be unreasonable to spend a few hours looking at videos and reading before trying to create your first design…it will be worth the effort.

Note: the figures and screenshots referred to in these tables are at the end of the document.

Task 1 – Create a Vivado Project and add RTL and constraints

Step	Screen	Action	Explanation/Comments
0.5		Install Vitis 2022.2 on your PC and add the Digilent Board files to your installation of Vivado by following the instructions in this link: https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-vitis Note: Installing Vitis also installs Vivado, DocNav, and the cable drivers. The installation allows you to customize the installation. For example, you only need to install the Series 7 devices. You can deselect all of the AI-related, Ultrascale flow, and DSP tools to shrink the download size (which is in the 100GB range)	The PCs in the Capstone Lab in FAB and in WCC 213 have Vivado/Vitis installed but consider installing the tools on your PC if it has the capacity and performance to run them. You can do most of your development without entering the labs.
1	Start Vivado	Start Vivado and brings up the start screen	The getting Started system has a top-level module that instantiates the embedded system you will create with IP Integrator.
2	Create New Project	Click on *Create Project* (or *File/Project/New*) and then click on *Next*>. Browse to the directory you want to create the project in and name the project. Check the Create project subdirectory box. Click on *Next* >	Name of directories and files must not include spaces.
3	Project Type	Select *RTL Project*. We will add the HDL and constraints files in the next step. Click on *Next* >	
4	Add Sources	Press + (or *Add Files*) in the *Add Sources* tab and add `Nexys7fpga.v` and `rgbPWM_r2.v` from the `hdl` directory in the release package. Uncheck the *Copy Sources into project* option. Click on *Next* > Press + (or *Add Files*) in the *Add Constraints* tab and add `Nexys7fpga.xdc` from the `constraints` directory in the release package. Uncheck the *Copy constraints into project* option. Click on *Next* >	We will try to work with single copies of all the source files.

5	Default Part/Boards	Select the `Nexys A7-100T` (or `Nexys4 DDR` if you are using the FPGA boards in the labs or have access to a `Nexys 4 DDR`). It shouldn't really matter which of the boards you select because the boards are functionally equivalent, but you may as well choose the board type you will be debugging your design on. Click on *Next > and Finish* Note: I clicked on the Refresh button which fetches the latest board files from the Xilinx "Board Store" GitHub repository. It took about 10 minutes to do the update. The refresh added the the Digilent board files. You have to install (you see the download icon) the board file – if you do, click on the download icon.	We are going to use the Digilent board files instead of selecting a specific FPGA. The board files make simple work of adding board-specific functions and constraints to your project.

Task 2 – Create embedded system & configure MicroBlaze MCS (IP Integrator/Block Design Generator)

We have our top-level module and have applied constraints (pin constraints, not timing constraints), but note that there is a **?** next to `embsys_i` in the Sources/Hierarchy tab (you may have to expand the nexys7fpga .v file by clicking on the > next to the file name). The **?** is there because you have not created/configured the embedded system *(embsys)*. We will do that now.

Step	Screen	Action	Explanation/Comments
1	Vivado Main Screen	Select *Flow Navigator/IP Integrator/Create Block Design*. This will bring up the Create Block Design dialog. Name the design *embsys* and click OK. This will open a Block design screen (which will be empty). Click on *Add IP (+)*. You may want to float and maximize the diagram.	The name of the block diagram must match the name of the module being instantiated in `Nexys7fpga.v`
2	Block Design Diagram	Search for `MicroBlaze MCS` in the *Add IP* dialog and drag or double-click `MicroBlaze MCS` into your diagram.	
3	Block Design Diagram	*Double-click* on the `MicroBlaze MCS` block (or *right-click* in the block and select *Customize Block…* This will open the tabbed window that you will use to configure the MicroBlaze and IO module.	
4	Re-customize IP/MicroBlaze MCS/**Board**	Use the dropdown boxes to configure the board as follows: ● Clk – select `sys_clock` ● Reset – select `reset` ● GPIO1 – `dip switches 16 bits` ● GPIO2 – `push buttons 5 bits` ● GPIO3 – `Custom` ● GPIO4 – `Custom` ● UART – `Custom` Figure 2	We want GPIO3 to be assigned to leds 16 bits but there appears to be a strangeness in the Neyxs4 DDR board file. We will fix this in Task 3.
5	Re-customize IP/MicroBlaze MCS/**MCS**	Use the dropdown boxes to configure the board as follows: ● Memory Size – `32KB` ● Enable Debug Support – `DEBUG AND UART` Figure 3	
6	Re-customize IP/MicroBlaze MCS/**UART**	In Step 6 we selected DEBUG AND UART. This will configure the Debug IP (called MDM for **M**icroBlaze **D**ebug **M**odule) with a serial interface that can be connected to the Vitis console (`stdin` and `stdout`). That is the only serial port you need for this project but if you would like a 2nd serial interface that you can connect to a terminal emulator program such as `PuTTy` you can enable the transmitter and receiver and change the baud rate.	Since I only used the MDM UART, I left the UART transmitter and receiver disabled – up to you.
7	Re-customize IP/MicroBlaze MCS/**FIT**	Select the FIT 1 tab (there are 4 Fixed Interval timers in the MicroBlaze MCS IO Module). ● Use Timer option – check it ● Generate Interrupt – uncheck it. The getting started project does not use interrupts. ● Number of Clocks Between Strobes – 500 The getting started project does not use FIT2, FIT3, or FIT4 Figure 4	500 x 10^{-8} x 2048 => 97Hz which is fast enough that your eye will average out the LED intensity. 10^{-8} is the 10ns clock pulse (sys_clk is 100MHz) and 2048 is the period of the PWM signal.
8	Re-customize IP/MicroBlaze	The IO Module includes 4 programmable timers. We do not use any of them in the getting started project so uncheck any of the PIT1, PIT2, PIT3,	

	MCS/**PIT**	or PIT4 timers that are enabled (uncheck *Use Timer* for all 4 PITs)	
9	Re-customize IP/MicroBlaze MCS/**GPO**	The IO Module includes 4 configurable **G**eneral-Purpose **I**nput/**O**utput peripherals. Each of these peripheral blocks can be configured by selecting/deselecting or setting the options. We have "hardwired" the configuration of GPO1, GPO2, and GPO3 in Step 4 by assigning those 3 peripheral blocks to board functions. Each of the GPIO peripheral blocks can be either an input, an output, or both an input and an output. GPOx is the output functionality of the peripheral. Not much to do here: ● GPO1 – assigned to the switches. Input only – cannot be configured ● GPO2 – assigned to the push buttons. Input only – cannot be configured. ● GPO3 – This output will be connected to the 16 LED's on the Nexys4 board. Enable it by checking the Use GPO option and setting the number of bits to 16. ● GPO4 – This output port will be connected to the Control Register input of the `rgbPWM_0` block. Enable it by checking the *Use GPO* option and set *Number of bit*s to 32. Figure 5	
10	Re-customize IP/MicroBlaze MCS/**GPI**	The IO Module includes 4 configurable **G**eneral **P**urpose **I**nput/**O**utput peripherals. Each of these peripheral blocks can be configured by selecting/deselecting or setting the options. We have "hardwired" the configuration of GPI1and GPI2 in Step 4 by assigning those peripheral blocks to board functions. Each of the GPIO peripheral blocks can be either an input, an output, or both an input and an output. GPIx is the input functionality of the peripheral: ● GPI1 – assigned to the switches. Input only – cannot be configured. Should be 16-bits wide ● GPI2 – assigned to the push buttons. Input only – cannot be configured. Should be 5-bits wide ● GPI3 – assigned to the LEDs. GPI functionality should be disabled ● GPI4 – The getting started project uses GPO4 as the control register for `rgbPWM_0`. Uncheck the USE GPI option. GPIO4 has been configured as output-only. Figure 6	
11	Re-customize IP/**Interrupts**	The getting started project does not use either internal (to the IO Module) or external interrupts. Uncheck the USE EXTERNAL INTERRUPTS option.	
12	Re-customize IP	Congratulations! You have essentially built your first custom microcontroller. Review the settings in all of the tabs, fix any that are incorrect, and click the *OK* button.	

Task 3 – Complete the embedded system (Block Design Generator)

The Block Design Generator has brought us more than 50% of the way towards building our embedded system, but there is more work to be done. We must add additional peripherals, customize them if necessary and wire the blocks together. This is done in the Design pane of the Block Design Generator. Open your Vivado project and the Block design if you closed them after completing Task 2.

Step	Screen	Action	Explanation/Comments
1	Block Design Diagram	Add an instance of the `rgbPWM_r2.v` by *right-clicking* in an empty area of the Block design screen and selecting *Add module*… If you added `rgbPWM_r2.v` when you created the Vivado project and *Added Sources*, you should be able to select the module. Click on *OK* > Figure 7	
2	Block Design Diagram	Customize the `rgbPWM_0` block by selecting and *right-clicking* the `rgbPWM_0` block and selecting *Customize Block…* (you can also double click on the block). Confirm/set the parameters as shown: ● Divide Count – `500` ● Max Count – `2048` ● Polarity – `"1"` ● Use Divider – `"0"` Click on *OK*>	If you examine the source code for `rgbPWM_r2.v` you will see these parameters are defined in the module signature. The IP Integrator automatically converts parameters so that can be customized
3	Block Design Diagram	Click on *Run Connection Automation* and select `microblaze_mcs_0`. *Deselect (for now)* `rgbPWM_0` Click on each of the Ports under Microblaze_mcs_0 ● Clock – select `sys_clock` ● Reset – select `reset` ● GPIO1 – select `dip switches 16 bits` ● GPIO2 – select `push buttons 5 bits` ● GPIO3 – select `leds 16 bit` ● GPIO4 – select `Custom` *Review the GPI and GPO tabs to make sure they are correct. GPI1 and GPI2 should be enabled. GPO3 and GPO4 should be enabled.* Click on *OK*>	

4	Block Design Diagram	Connect the clock and reset and Control Register signals to `rgbPWM_0` ● Click on the reset port of `rgbPWM_0`. This should highlight the port and change the mouse pointer to a pencil. Draw a line from `reset` on `rgbPWM_0` to the `reset` signal coming into the block design. Note that `reset` is asserted low (bubbles on both the `MicroBlaze MCS` port and `the rgbPWM_0 port`) ● Click on the `clk` port of `rgbPWM_0`. This should highlight the port and change the mouse pointer to a pencil. Draw a line from `clk` on `rgbPWM_0` to the `FIT1_Toggle` output from the `MicroBlaze MCS`. ● Right-clock on the `rgbPWM_0` block and select Make External. This will make the `rgbPWM` unconnected inputs and outputs external and available at `NexysA7fpga.v` (top level). ● Right-click in an empty area of the Block Design diagram and select Regenerate Layout. This doesn't change any functionality but does clean up the drawing. Figure 8.	`NexysA7fpga.v` connects the GPIO4 output to the Control register input to `rgbPWM`. Making the connection from a GPIO enables the application program to write to the `rgbPWM` control register by writing to GPIO4 in the IOModule.
5	Block Design	Save the Block diagram by selecting *File/Save Block Diagram* or clicking on the *Save* icon.	
6	Block Design Diagram	Validate the design by right-clicking in an empty area of the Block design and selecting *Validate Design* (or you can click on the *Validate Design* button in the Block Design Window. You will get one critical warning which you can ignore as shown in Figure 9. You may get additional critical warnings during Synthesis related to this same reset signal. The warning is truthful – reset is asynchronous to the FIT1_toggle clock to rgbPWM, so let's fix it. You can eliminate this warning by doing what the message says and adding a Processor Reset block can connecting it as shown in Figure 10 Figures 9 and 10	
7	Block Design Diagram	You may have noticed the icon at the top left of the MicroBlaze MCS block. If you haven't, notice it now and click on it. Doing this will show that the MicroBlaze MCS block is hierarchical and is composed of several IP blocks. When you build an embedded system for the projects you will be using the full-blown Microblaze and run a step called Run Block Automation which adds and connects many of the same blocks. The DMLB and LMLB blocks implement the 32KB of memory we configured into the MicroBlaze MCS. MDM is the debug block that connects the Microblaze with the source code debugger in Vitis. IO Module is the IO IP block we have configured. Click the *Address Editor* button. The table shows how memory addresses have been allocated (all peripherals in MicroBlaze-based systems are memory mapped). According to the table memory addresses are assigned as follows: ● Instruction/Data memory – 0x0000_0000 – 0x0000_7FFF ● IO Module registers – 0x8000_0000 – 0x8000_FFFF	

		• MDM registers – 0x4000_0000 – 0x4000_FFFF Click on the `X` at the top-right of the Window to close the diagram and return to your embedded system blocks diagram.	
7	Vivado main screen	Select *Flow Navigator/IP Integrator/Generate Block Design*. The IP Integrator will assemble all the HDL from all your peripherals and the `Microblaze MCS`, create `tcl` scripts and `makefiles`, and build an HDL version of your embedded system and its peripherals. If you have configured and connected everything properly the block design will be generated successfully. If generation fails check all your connections, make necessary changes, and generate the outputs again. NOTE: MY BLOCK DESIGN HAD MORE THAN 100 WARNINGS AFTER THE OUTPUT PRODUCTS WERE GENERATED. I LOOKED THROUGH THE WARNINGS AND DIDN'T FIND ANYTHING RELATED TO MY DESIGN. YOU SHOULD DO THE SAME SINCE THE WARNINGS CAN ALSO POINT OUT PARTS OF YOUR BLOCK DESIGN THAT MAY NOT BE CORRECT	Generating your block design will take several minutes (about 5 minutes on my 4 core Intel I7 laptop w/ 16GB of RAM and a 1TB SSD). The good news is that this step synthesizes your embedded system which will greatly shorten the synthesis time of your entire design.

Task 4 - Synthesize, Implement and Generate bitstream for the FPGA (Vivado)

The file "embsys.v" is generated from the block diagram and it contains a module "embsys" which has all the pins/ports used by the design. These are the ports which we use to connect to pins on the Nexys A7 board by instantiating embsys in the top-level module of your design file. Having succeeded at Tasks 2 and 3 your embedded system should have been added to your project. If there is still a **?** next to the embsys_i icon in the Hierarchy window you may not have named your block design "embsys". Edit the top-level file to match the name of your block design. That should remove the **?** but the design may not synthesize correctly because there is a mismatch between the port properties and names of your block design and the wires defined in the top-level module. You should correct any mismatches before attempting to synthesize the design. The comparison is done visually by looking at the instantiation in the top-level module (nexysA7fpga.sv) and comparing it to a template that can be provided by the IP Integrator.

To view the instantiation template of IP Integrator, go to *Project Manager/Sources pane*, right-click on embsys and select *View Instantiation Template*. You can right-click on that tab to float the window and then double click on the NexysA7fpga .v file in the *Hierarchy* screen to display its contents in a floating window. Do a port-by-port comparison between the embsys instantiation at the bottom of the template and the embsys instantiation in the top level nexysA7fpga module. You want the pin names to match and be connected to the appropriate top-level ports. Note that some of the top-level ports are different then the port names on the embedded system:

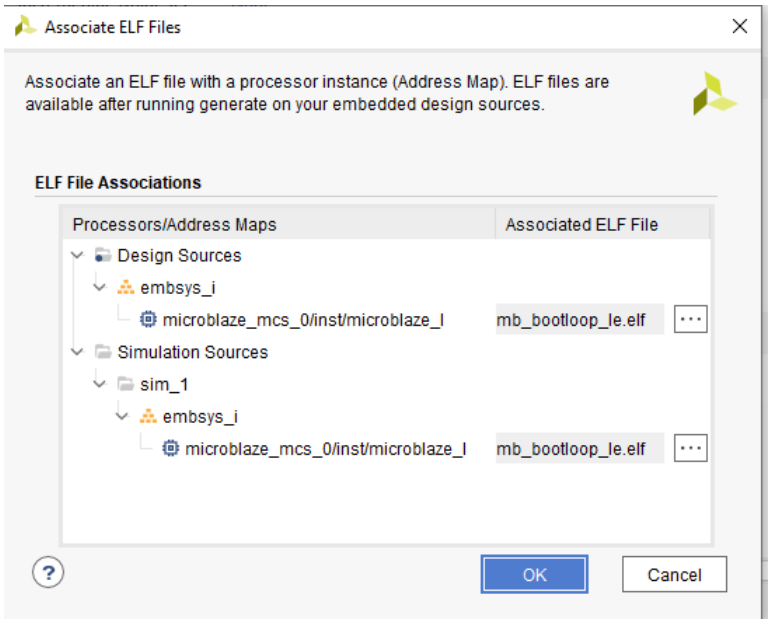
```
embsys embsys_i
       (.clkPWM_0(clkPWM),
        .dip_switches_16bits_tri_i(sw),
        .led_16bits_tri_o(led),
        .push_buttons_5bits_tri_i(push_buttons_5bits_tri_i),
        .reset(btnCpuReset),
        .rgbBLUE_0(RGB1_Blue),
        .rgbGREEN_0(RGB1_Green),
        .rgbRED_0(RGB1_Red),
        .sys_clock(clk)
   );
```

After you have reconciled the port names, save the NexysA7fpga.v file, and execute *Synthesis* from the Vivado Project Manager. Check all warnings carefully, and be on the lookout for unconnected signals, port width mismatches, signals driven to 0 and/or optimized out, etc. that are related to your top-level module. If synthesis fails, check the port names and widths again, open the Block design and look for missing inputs or outputs, and, in general, trust but verify. For example, even though I double checked when I first built my Getting Started system Synthesis generated a warning that btnCPUReset was unconnected. It turns

out that I had a typo – the name of the signal was `btnCpuReset`.

After you have successfully synthesized the design and resolved the warnings (my design shows 6 additional warnings after synthesis but none of them pointed to problems in my `NexysA7fpga.v` connections), *Run Implementation*. This design will take several minutes to place and route. Check the logs carefully when a process is done, particularly if you get an Error or a Critical Warning.

The next step is to include an executable program in the bitstream. It would not be unusual to skip this step but there is a potential handoff problem between Vivado and Vitis in 2022.2 which necessitates downloading the bitstream into the FPGA from Vivado rather than from Vitis. To complete this step, select Tools/Associate ELF files from the Vivado main menu. This will open a dialog like the one shown here:



The defaults should be OK. `mb_bootloop_le.elf` is the bootloader used by Vitis to load an application and prepare it for debug or execution in Vitis. Click on the OK button.

Remember this step because you can use it to build a bitstream that includes your working application which will automatically start running when you configure the FPGA. But…we're getting ahead of ourselves.

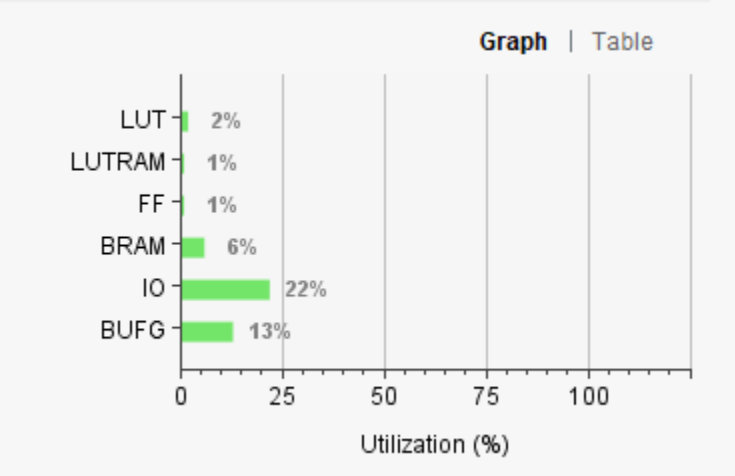
Click on *Generate Bitstream* from the *Flow Navigator* menu. *Generate Bitstream* creates the .bit configuration file for your design.

The last step in the hardware creation process is to export the hardware and bitstream. Exporting the design prepares the bitstream, drivers, etc. for handoff to Vitis. To export a design, select *File/Export/Export Hardware* from the *Flow Navigator* screen and include the

bitstream. Note the directory that the file is exported to – you will need it later.

Pat yourself on the back - you have created your first target hardware platform with the Xilinx embedded tool chain. With the hardware platform complete it is time to move on to the application software.

An "aside":
It is interesting to note how little of the FPGA capacity is used to build this (hopefully) fully functional embedded system.



The Nexys A7-100T (Nexys4 DDR) board contains a large FPGA so there is plenty of capacity to implement larger embedded systems…a feat you will accomplish as the course progresses. Even so, in all my years of using this board I think I've only exceeded 50% utilization a handful of times and then it's usually limited to BRAMS (Block RAM) and IO.

Task 5 - Import the target hardware platform to Vitis and create the target software environment (Vitis)

Software development/debug in the Xilinx embedded system tool chain is done using Vitis.

*Note: Xilinx moved Vitis to **V**isual **S**tudio **C**ode in 2023. We are using Vitis 2022.2 in this edition of the Getting Started project but expect the Vitis IDE to be different if you move to a later edition of Vivado/Vitis.*

Vitis includes a Microblaze port of the GNU tool chain. Since the GNU tools are command line based, Xilinx has wrapped them in an Eclipse-based **I**ntegrated **D**evelopment **E**nvironment, Eclipse is an open-source platform that has been adapted to many CPU architectures, many tool chains, and many, many vendor product offerings.

One Eclipse concept worth mentioning is "Perspectives." The GUI changes depending on what the user is doing. For example, there is one "Perspective" for code development and another for program debug. You can switch between multiple open perspectives using labeled buttons in the upper right corner of the screen. It is not the intent of this guide to provide a deep look at Vitis.

For this exercise, our tasks are to:

- configure Vitis for our target hardware
- create an appropriate application platform and board support package
- import a working application

In this section we will take care of the first two items.

Step	Screen	Action	Explanation/Comments
1	Vivado main screen	Open Vitis from Vivado by selecting *Tools/Launch Vitis* from the Vivado main screen.	You can open Vitis from the Windows *Start* menu, but for the first time after you export the hardware it seems better to open Vitis from Vivado.

2	Select a workspace	Vitis may "propose" a workspace directory when you open it from Vivado. NOTE: Vitis may display a welcome screen or pane. If it does, take a few minutes to review introductory videos and tutorials. Closing the Welcome tab will bring up (or expand) the Vitis main screen.	You may want to create a Vitis workspace local to your project even though it makes it inconvenient to share software between different hardware systems or you may want to create a common workspace for all your Vitis projects. The choice is yours.
3	Create a new Platform project and board support package	● From the Vitis workspace select *File/New/Platform Project*. This will open the ***Create a New Platform Project…*** window. Enter a Platform project name and *Click Next>* ● Select *Create from hardware specification (XSA)* and *Click Next>* from the **Platform Project Specification** window. o Select the `.xsa` file to build the platform from. Browse to your Vivado project and select the `.xsa` file. (My project's `.xsa` file was at `<my project location>\synth\gs_system\nexysA7fpga.xsa` o Select **standalone** and **microblaze_mcs_0_microblaze_I** Click *Finish*	
4	Project Explorer	Check that the hardware system and board support package look correct. Click through the platform project until you find the `.xsa` file you generated the platform from. Double-click on the file and check that all your devices have been included and all have valid address ranges. The address ranges are where the registers and memory controlling the peripheral are based. Click through the platform project until you find the Board Support Package entry and double click on it. This should bring up a Board Support pane. Check the Peripheral drivers to make sure that all the peripherals have drivers assigned to them (i.e., they do not say "generic" or "none"). You can use the Modify BSP Settings to reassign `stdin` to `mdm` (MicroBlaze Debug Module) instead of `uartlite`.	If you use `uartlite` for `stdin` and `stdout` you should configure a terminal application such as `PuTTy` to see the message that the test application puts out. Note: At time of writing, there may be an unsolved bug with `uartlite` output, so we recommend using `mdm`.

		You will note that your platform project is Out-of-date. Rebuild it by *right-clicking* and selecting *Build Project*	
5	Project Explorer	Expand the *standalone_domain/bsp/microblaze_mcs_0_microblaze_I /include* folder and find the *xparameters.h file*. Get familiar with the file because it contains the `#defines` for all the devices in your system.	

Task 6 – Import and execute the test application (Vitis)

We are almost there. We have synthesized and implemented our target hardware system (including the embedded computer system) and prepared Vitis for application development and debug by importing the hardware description and building a target platform for the application to run on. All that's left is to create the target application, download the target hardware system to the FPGA, download the application to the CPU memory (32KB of Block RAM in the target hardware system) and then run and/or debug the program. Our target application for this guide is the system test application called *gsproj_app.c*. The other source files are *platform.c, platform.h and platform_config.h.*

Step	Screen	Action	Explanation/Comments
1		Reopen Vitis if you have exited it. Vitis should remember your workspace and the last platform project.	
2	Project Explorer	Select the *File/New/Application Project* menu item. This should bring up the *Create an Application project* Dialog. Name your application project. Vitis will fill in a System Project name. *Click Next>*	
3	New Project/Platform	Select the platform you created and *Click Next>* Clicking Next should bring up the Domain dialog. Make sure the `standalone on Microblaze_mcs_0_microblaze_I` domain is selected *Click Next>*	
4	New Project/Templates	Select *Empty Application(C)* and Click on *Finish*	This step creates a (mostly empty) framework for a C application program. You can create and work with more than one C application in a software platform and more than one software platform for a hardware platform
5	Project Explorer	In the system application hierarchy, double click on `src/lscript.ld`. This will bring up the Link map for your application. You do not need to do anything with this file for the getting started project app, but it's worth perusing because it shows all sorts of useful information about how/where your app is loaded into memory.	You may implement systems with more complicated memory maps, so it is good to know how to manipulate the load map.

6	Project Explorer	Import the `gsproj_app` files by right-clicking on the `/src` folder in the Vitis project window and selecting *Import Sources*.	
7	File System	Browse to the `application` directory for your getting started project. Select all the source (.h and .c) files. Click on *Finish*	Once you have successfully built the target application you are ready to download the hardware configuration to the FPGA and the application image to the Microblaze memory.
8	Project Explorer	Now that you have your application, all you need to do is build it. Click on *Project/Build Project* in the Vitis menu**.** Doing this will build (re-build) the standalone OS and your application. Fix any problems and build the project again until both the standalone OS and your application build successfully.	Vitis makes use of the `make` utility and will build only what is necessary. You can use *Project/Clean* to force the standalone OS and/or your application to build. Note: There may be warnings and infos but there should be no errors.
9		You are done with the software creation process. Now it is time to run the getting started application. The application copies the slide switches to the LEDs and uses the pushbuttons to change the duty cycles of the 3 segments of RGB1. Connect the Nexys board to your PC with the USB cable and power up the board. Your PC should find the Xilinx cable drivers and open a hardware session with the FPGA board. The self-test on the board should start running. If the board does not power up check the jumpers to make sure the USB connector is powering the board (refer to the Reference manuals for your board).	
10	Program FPGA	IMPORTANT: Note that this step could fail because there was a handoff problem between Vivado and Vitis. The problem can be fixed with some ugly edits to files within the .xsa file but it is a very poor workaround. In my experiments it is limited to the way MicroBlaze MCS instances are named so (fingers crossed) we should be OK when we switch to a full-blown MicroBlaze-based system . If this step fails, there is a simple background in Step 10a. Select *Xilinx/Program FPGA*. should bring up the Program FGPA dialog. Click on *Generate*	You can also initialize the bitstream to include the executable, meaning the program will start up automatically once the FPGA is configured.

		If the process fails you will see a message like this one in the Console window:
		
		…
		`update_mem failed`
		`ERROR: [Updatemem 57-85] Invalid processor specification of: embsys_microblaze_mcs_0_2/microblaze_I. The known processors are: embsys_i/microblaze_mcs_0/inst/microblaze_I`
		..
		
		Go to Step 10a
		
		If the process succeeds Click on *Program.*
		
		If all goes well, you should get a dialog box saying the FPGA configuration is complete. Initializing the bitstream and downloading it to the FPGA could take a minute or so.
		
		The Digilent Nexys A7 self-test should stop running.
		
		Important: If you took ECE 540, it is likely that your computer is currently using the driver that works with Catapult SDK and not the Digilent USB drivers. If the Program FPGA operation fails, follow the instructions in Appendix E of the ECE 540 Getting Started Guide to roll the FPGA board driver back to the FTDI driver.	

10a	Vivado	NOTE: Follow these steps to program the FPGA from Vivado (make sure the Nexys board is powered up and running its self-test ● From the Flow Navigator click on *Open Hardware Manager* ● In the *HARDWARE MANAGER* window click on *Open Target/Autoconnect* This should open the Hardware connection server ● Click on *Program device* ● Check that the correct `.bit` file is selected. If not, use the … to select the directory and file ● Click on the *Program* button This should download your `.bit` configuration file to the FPGA. Visually you will see that the self-test has stopped running.	
11	Vitis/Project Explorer Pane	The final step before running the program is to download the application image (*<application project_name>.elf* in this case) to the CPU memory. There are two run environments – *Run as…* and *Debug as… Debug as…* executes the program under a debugger.	If you have enabled the UART in MicroBlaze MCS you can use it by connecting to a terminal

		The debugger allows you to set breakpoints, look at variables, single step, etc. *Run as…* simply loads and runs the program – no debugger. Since you will most likely be debugging a program, we will run the program under the debugger. *Right Click* on the application project and select *Debug Configurations.* Double-click on *Single Application Debug* and open the *Target Setup* tab. Deselect *Program FPGA* (especially important if you were forced to use 10a) prevents the FPGA from being reprogrammed every time an application is launched. Click on the *Debug* button to launch the debugger. Vitis will rebuild the software environment and application (if necessary), initialize the debugger, and download the executable file. If Vitis is successful it will open the Debug Perspective and wait for you to execute debug commands. There are icons to single step into functions, single step around functions, stop, start, set breakpoints, etc. Refer to the Vitis tutorials for more details. If you configured the MDM as Debug with UART the output from the program should appear in the Console window. If you included the IOModule UART in your hardware you can use PuTTY or your favorite terminal emulator to display the output. In Windows you can use the device manager to find out which port is assigned to the Xilinx serial monitor. Baud rate should be whatever you set it to when you customized the MicroBlaze MCS and configured the UART.	emulator on your PC. Don't forget to set the baud rate to match what you configured in *MicroBlaze MCS?UART* If Vitis does not automatically build your application and/or the platform project you can right-click on them and select *Build…* A good indicator of when a project must be rebuilt is (out of date) next to the project. Not 100% sure but pretty good – when in doubt, rebuild…make will only do what is necessary
12	Vitis Debug Perspective	Test the operation of the program. Experiment with the debugger commands and when you are confident that program seems to be working click on the green arrow. This will cause the program to run until it hits a breakpoint or exit. You can stop the program by clicking on the red box (stop). You can reset the system by pressing CPU_RESET. Pressing PROG will clear the FPGA, after which you will have to download the FPGA contents and the program again to restart it.	NOTE: Most embedded system programs do not have a mechanism to exit. The programs run in an infinite loop that many of us call "the main loop."

<finis>

Figures and screenshots

Figure 1 (Task 1, Step 5)

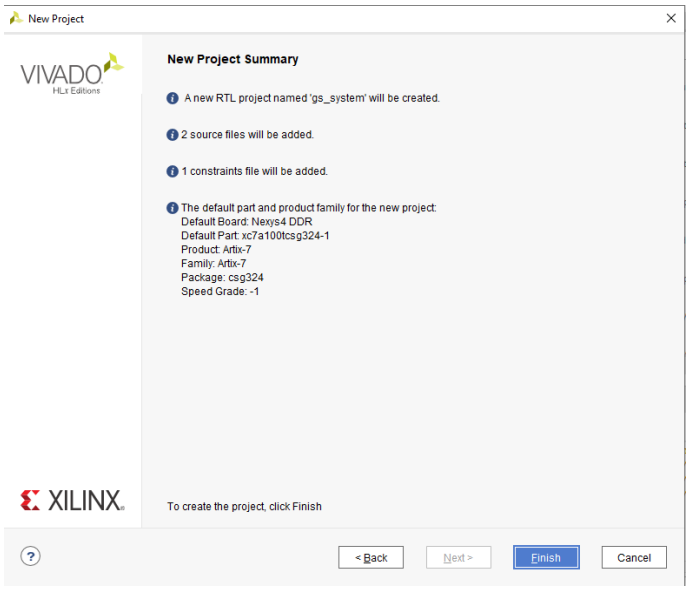


Figure 2 - Task2, Step 4 (Board configuration)

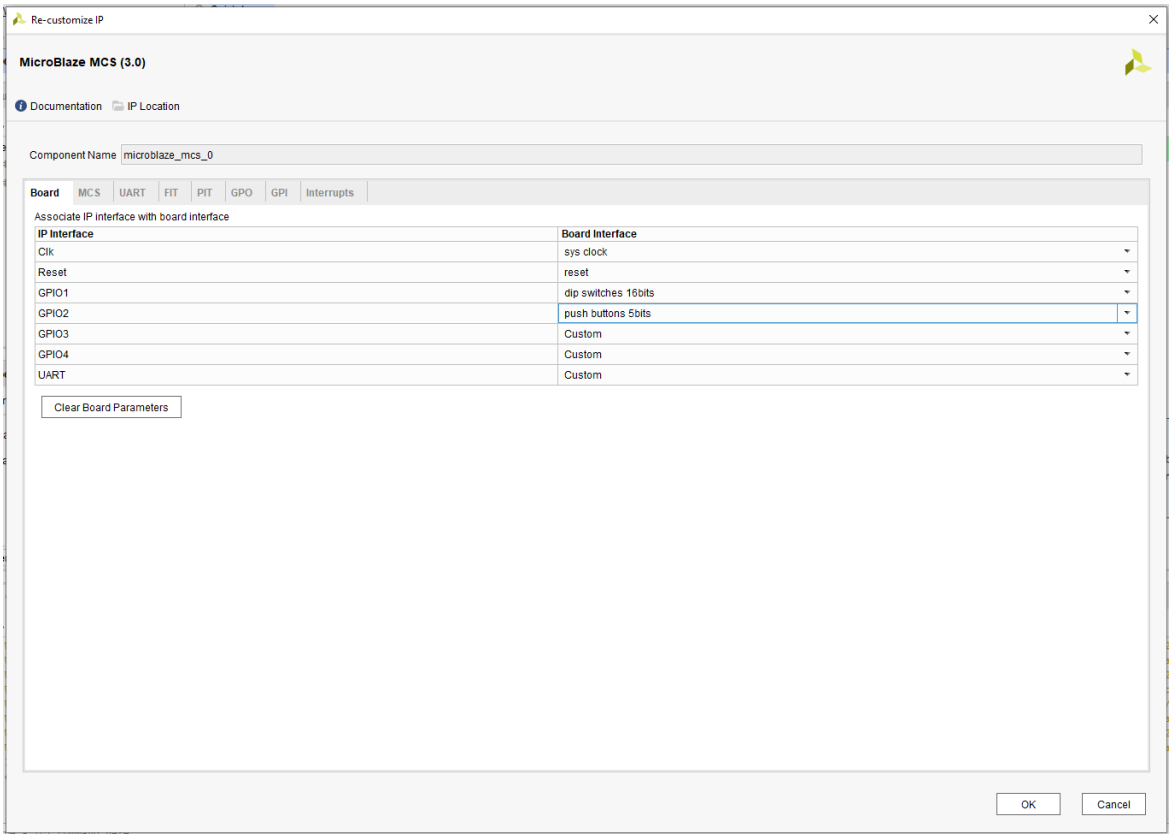


Figure 3- Task2, Step 5 (MCS configuration)

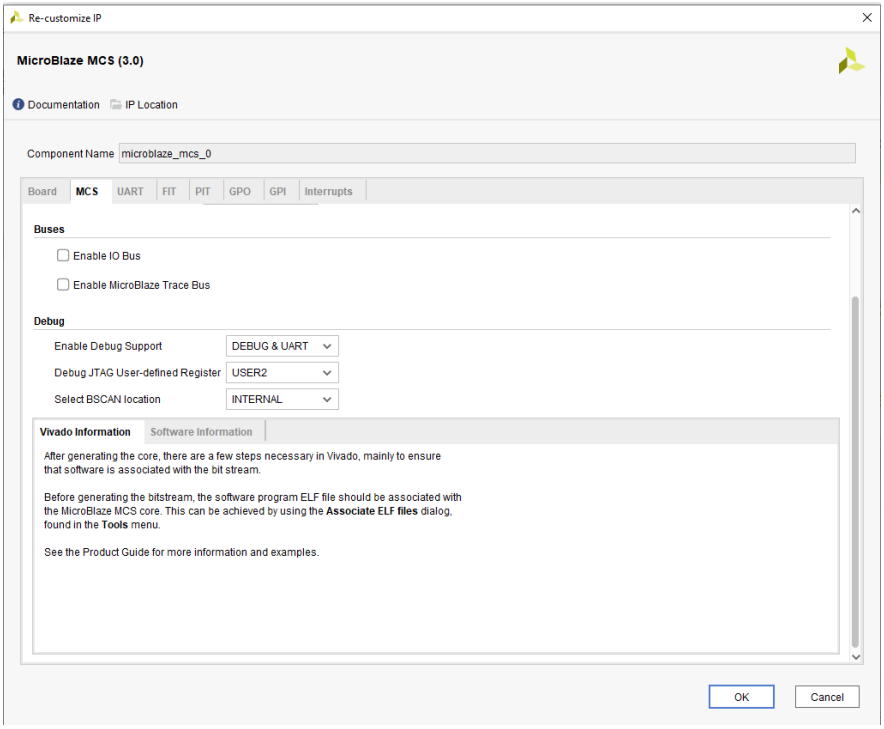


Figure 4- Task2, Step 7 (**F**ixed **I**nterval **T**imer(s) configuration)

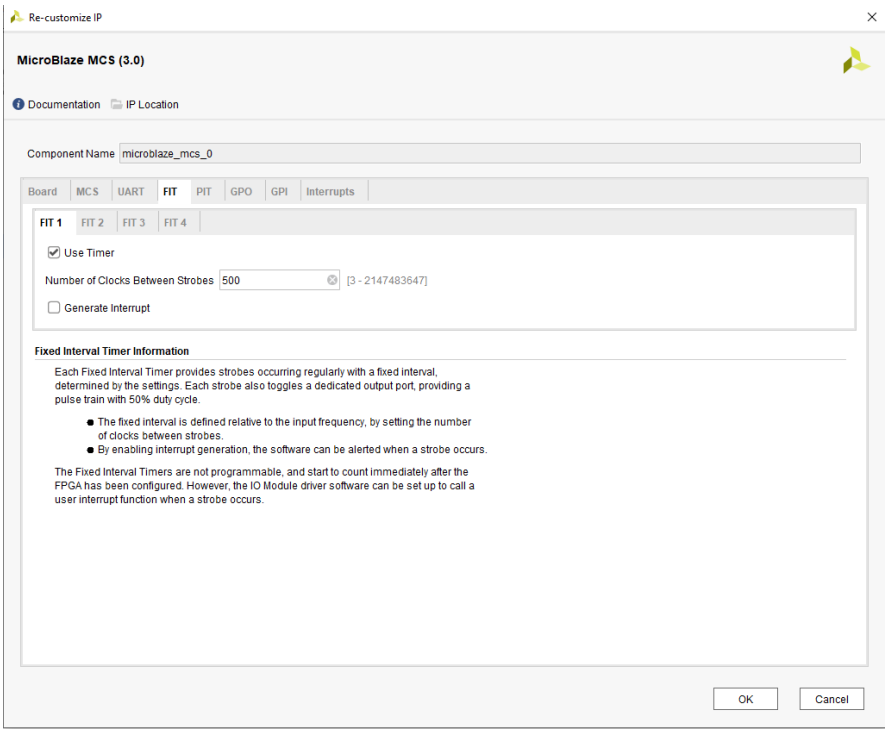


Figure 5- Task2, Step 9 (**GPO**(s) configuration)

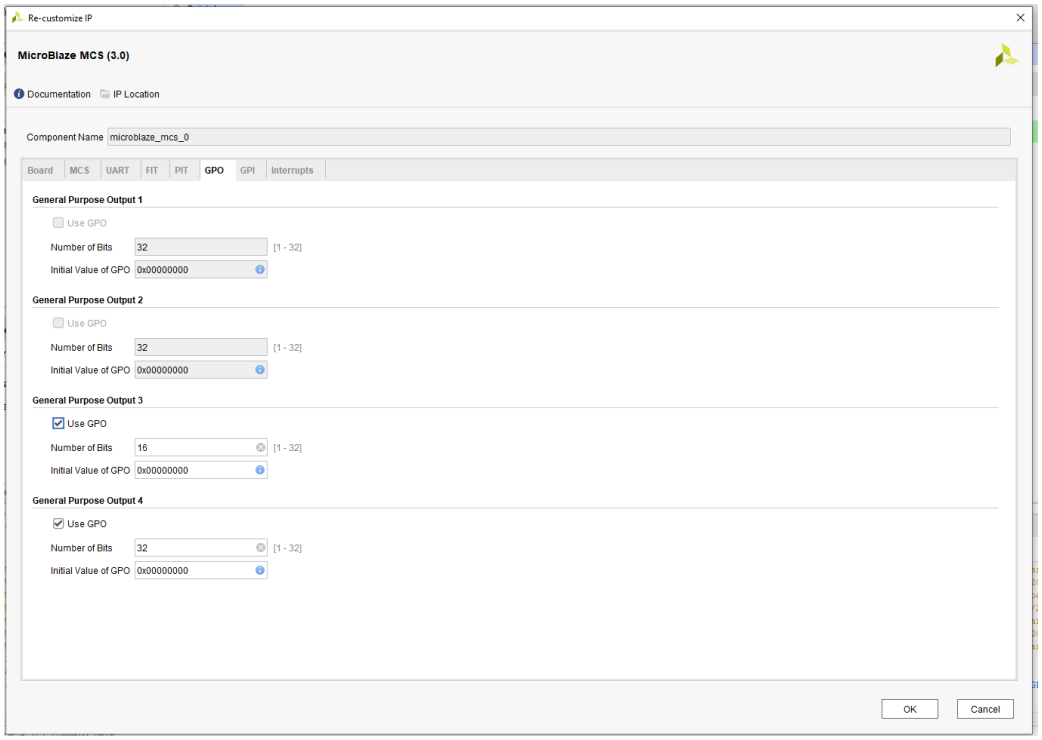


Figure 6- Task2, Step 10 (**GPI**(s) configuration)

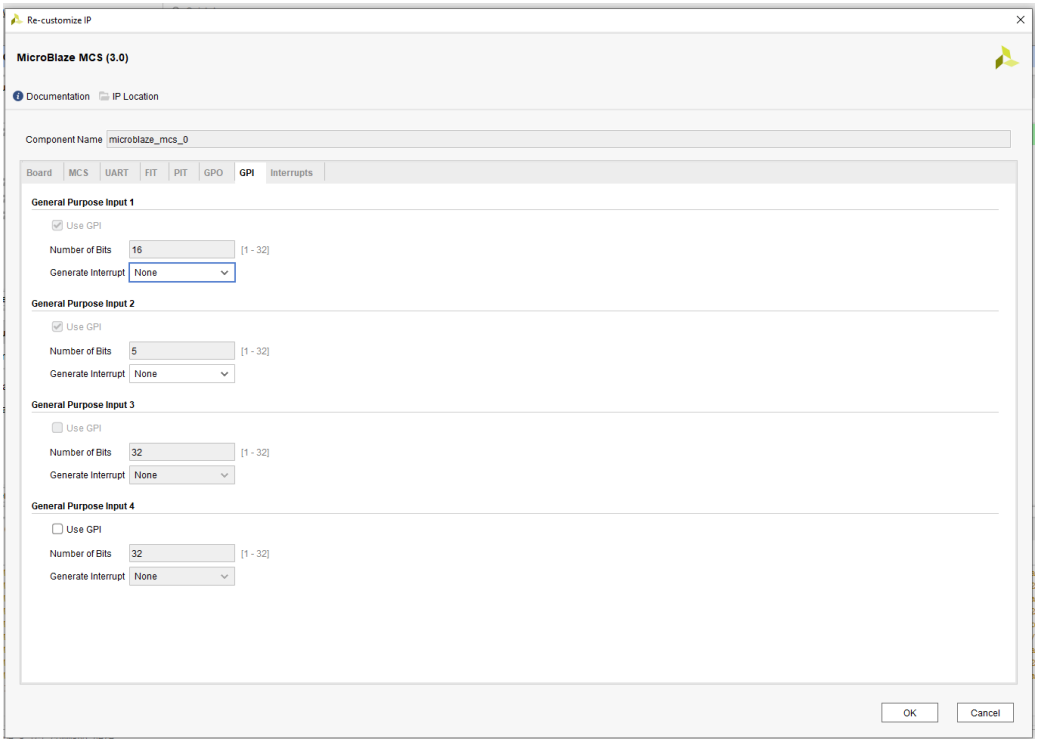


Figure 7- Task3, Step 1

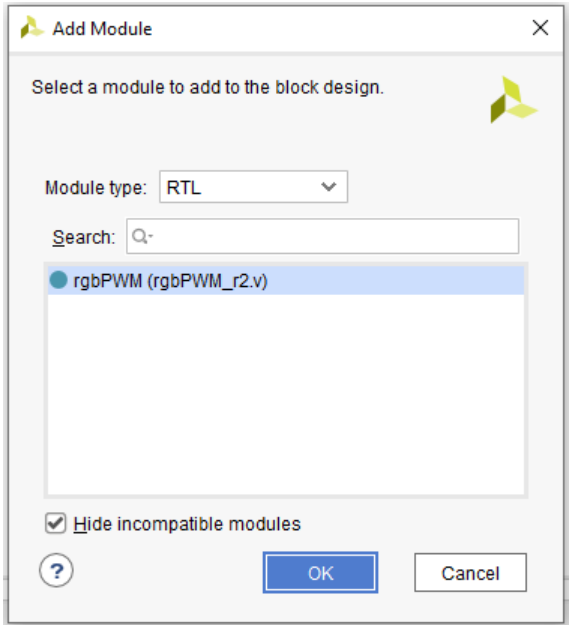


Figure 8- Task3, Step 4

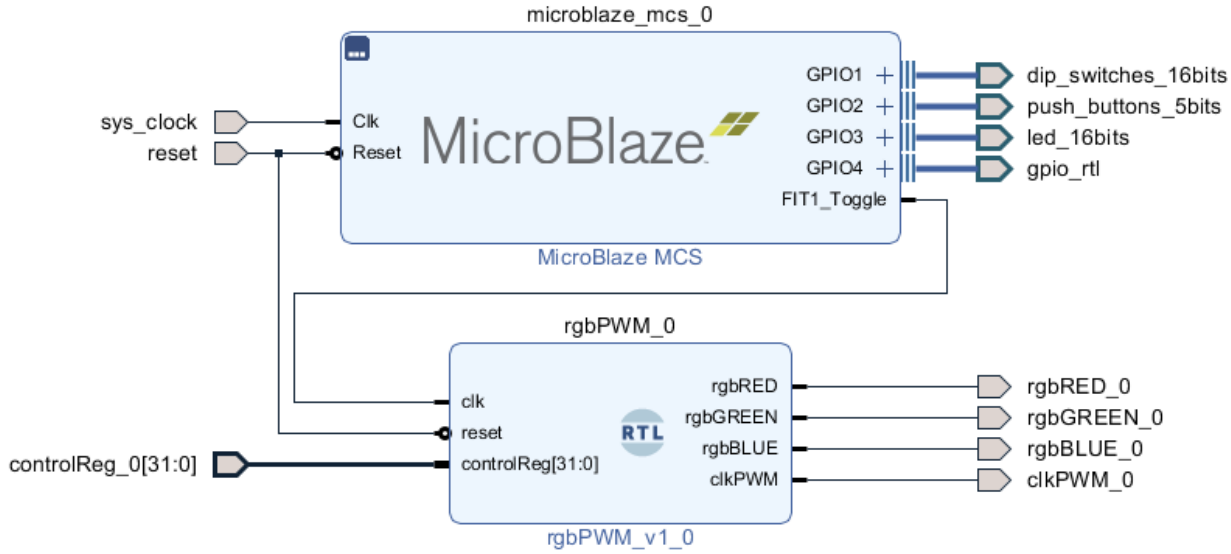


Figure 9 - Task3, Step 6

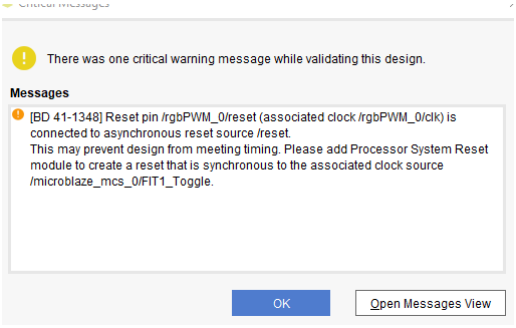


Figure 10 – Task 3, Step 6