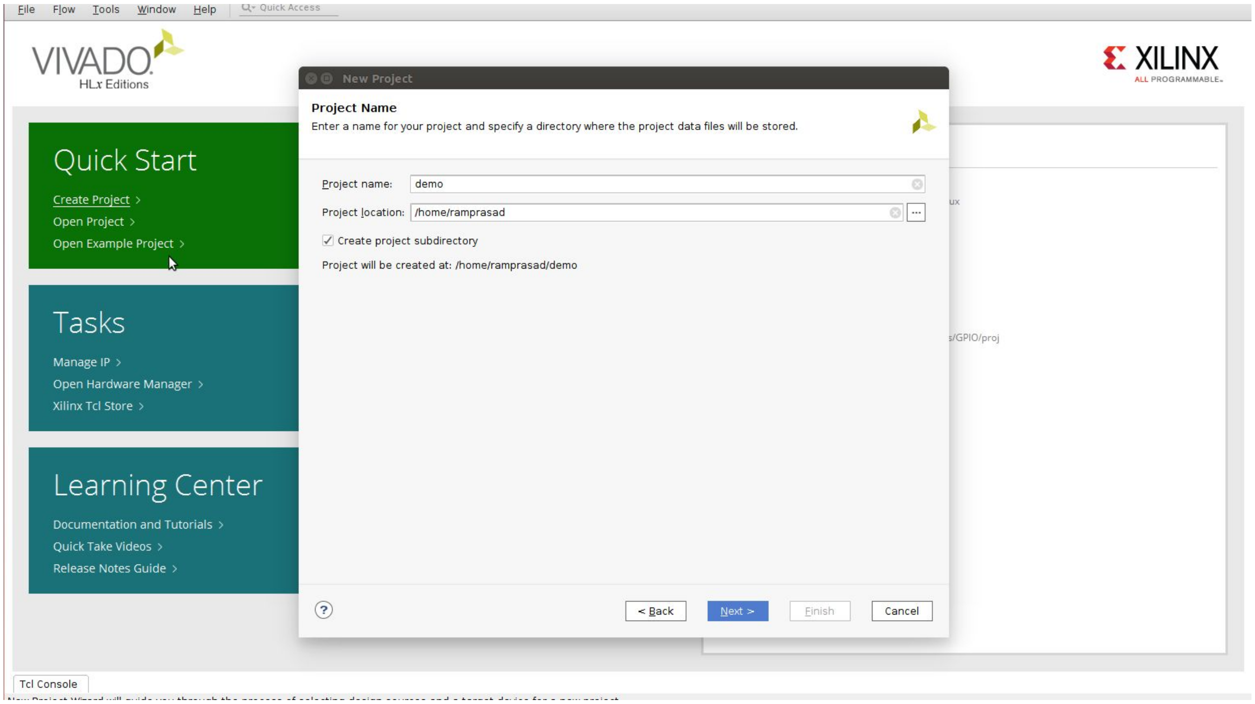
EDK Design for Linux Project

- Install the **Vivado Design Suite**.
- Load the Digilent libraries for Vivado. You can download them from https://github.com/Digilent/vivado-boards/archive/master.zip.
- Extract the new/board_files folder and copy the folder in your Vivado installation folder, under <Xilinx Folder>/Vivado/2016.2/data/boards/.
- Now,create a new project named “Demo” and click on next. 2



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: demo

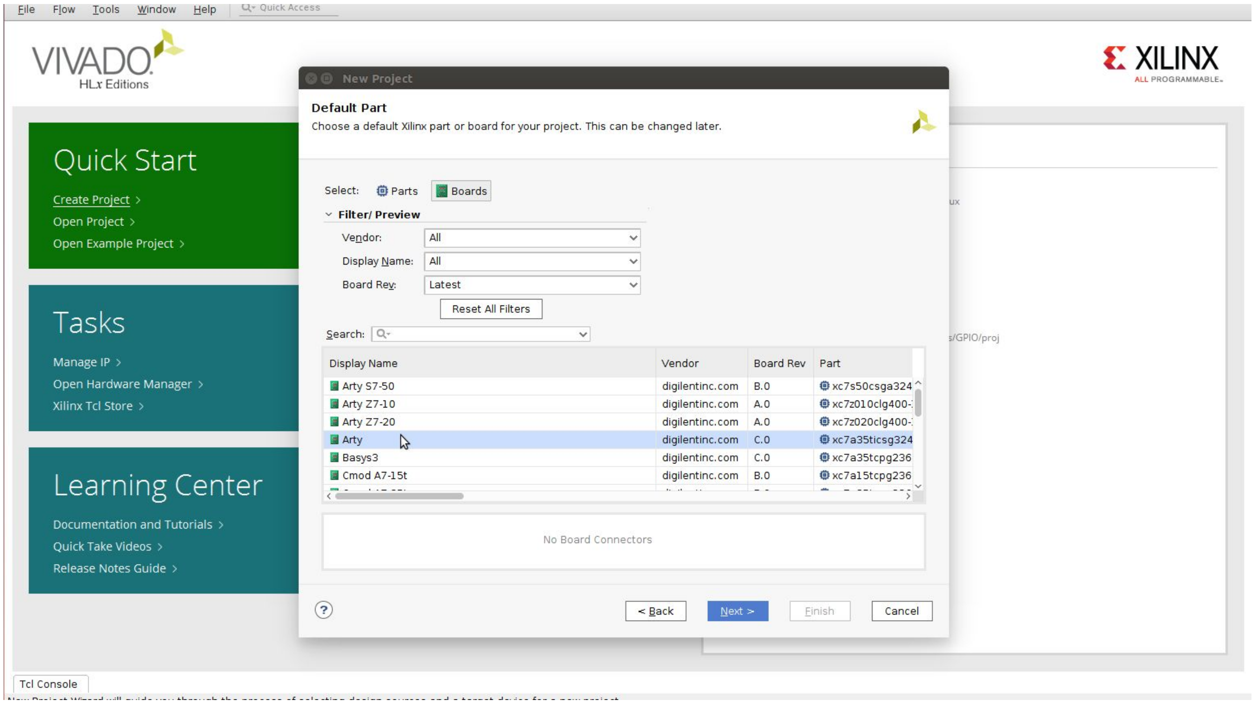
Project location: /home/ramprasad

☑ Create project subdirectory

Project will be created at: /home/ramprasad/demo

- select a **RTL Project**, and enable ‘**Do not specify sources at this time**’.

- Now select the **Boards tab** and search for the **Arty**. Then finish the project creation.



Default Part

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

Select: Parts Boards

Filter/ Preview

Vendor: All

Display Name: All

Board Rev: Latest

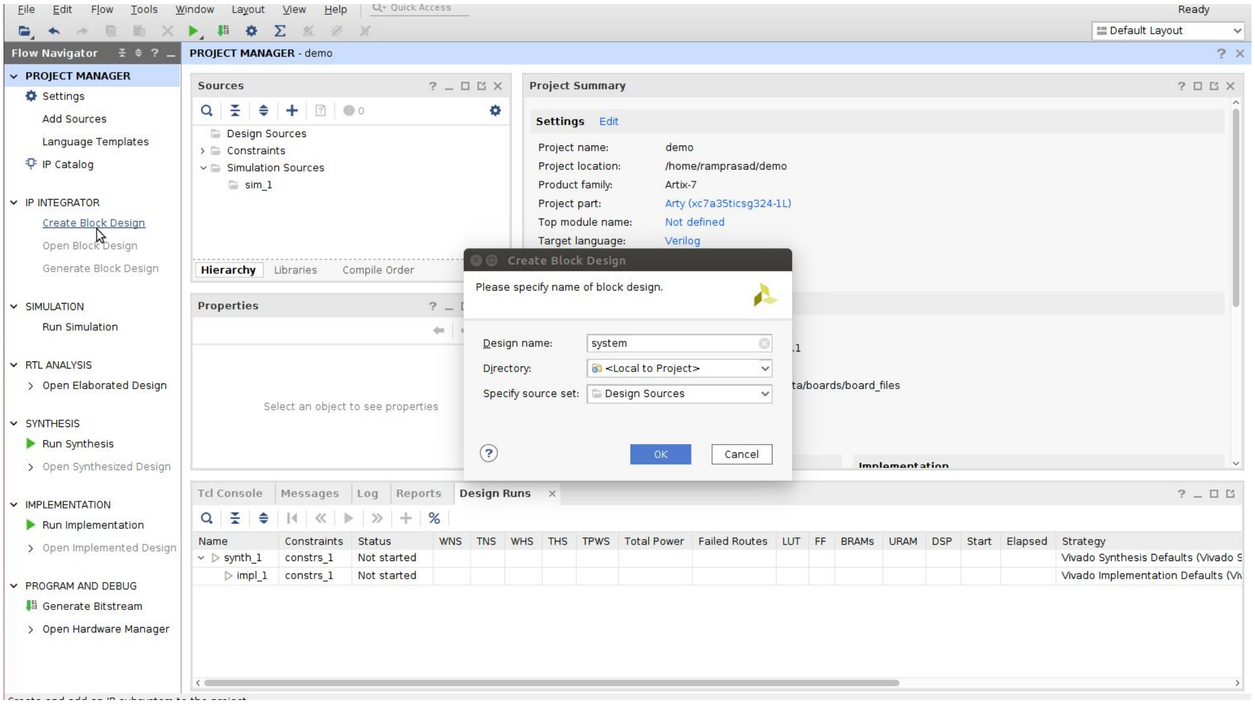
Reset All Filters

Search:

Display Name	Vendor	Board Rev	Part
Arty S7-50	digilentinc.com	B.0	xc7s50csga324
Arty Z7-10	digilentinc.com	A.0	xc7z010clg400-
Arty Z7-20	digilentinc.com	A.0	xc7z020clg400-
Arty	digilentinc.com	C.0	xc7a35ticsg324
Basys3	digilentinc.com	C.0	xc7a35tcpg236
Cmod A7-15t	digilentinc.com	B.0	xc7a15tcpg236

No Board Connectors

- Click on **Create Block Design** and specify the design name.



Project name: demo

Project location: /home/ramprasad/demo

Product family: Artix-7

Project part: Arty (xc7a35ticsg324-1L)

Top module name: Not defined

Target language: Verilog

Please specify name of block design.

Design name: system

Directory: <Local to Project>

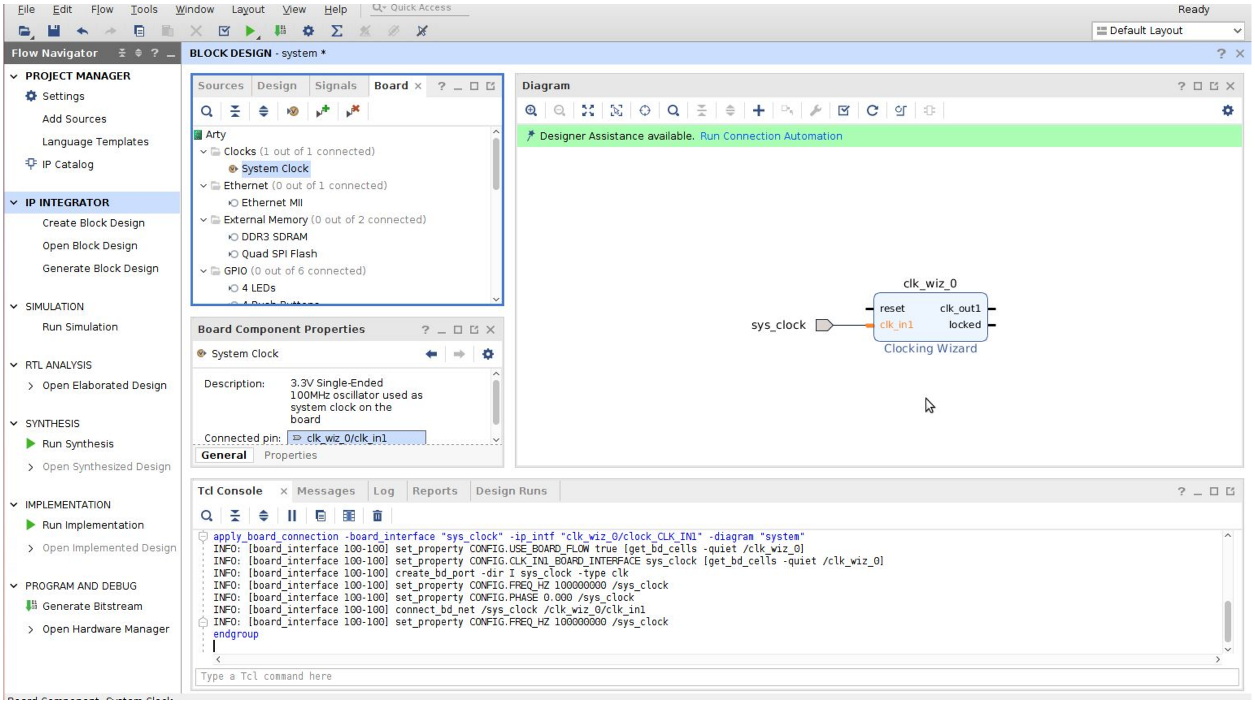
Specify source set: Design Sources

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado S
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Viv

- In the left panel select the **Board** tab, you will see the different blocks that are included in the Arty board, like System Clock, Ethernet MII, DDR3 SDRAM or

USB UART bridge. These blocks were created by Digilent based on existing IP Cores from Xilinx or another vendor.

- The first thing we include is the **System Clock**. Grab it from the list to the empty diagram on the right.



Board Component Properties: System Clock

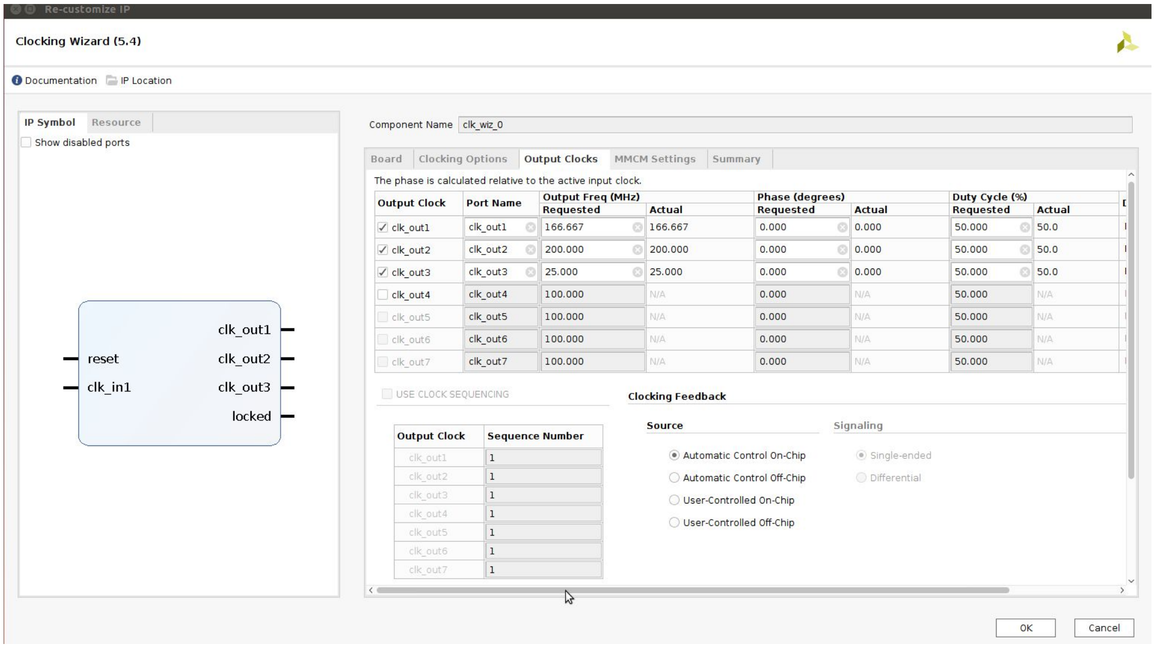
Description: 3.3V Single-Ended 100MHz oscillator used as system clock on the board

Connected pin: clk_wiz_0/clk_in1

Tcl Console

```
apply_board_connection -board_interface "sys_clock" -ip_intf "clk_wiz_0/clock_CLK_IN1" -diagram "system"
INFO: [board_interface 100-100] set_property CONFIG.USE_BOARD_FLOW true [get_bd_cells -quiet /clk_wiz_0]
INFO: [board_interface 100-100] set_property CONFIG.CLK_IN1_BOARD_INTERFACE sys_clock [get_bd_cells -quiet /clk_wiz_0]
INFO: [board_interface 100-100] create_bd_port -dir I sys_clock -type clk
INFO: [board_interface 100-100] set_property CONFIG.FREQ_HZ 100000000 /sys_clock
INFO: [board_interface 100-100] set_property CONFIG.PHASE 0.000 /sys_clock
INFO: [board_interface 100-100] connect_bd_net /sys_clock /clk_wiz_0/clk_in1
INFO: [board_interface 100-100] set_property CONFIG.FREQ_HZ 100000000 /sys_clock
endgroup
```

- Double click on the created block to edit the clock properties.Under Output Clocks tab, change the first **clk_out1** from 100.00 MHz to **166.67 MHz**. Then enable the **second** and **third** output clock and assign them **200.00 MHz and 25 MHz respectively**.



Component Name: clk_wiz_0

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
		Requested	Actual	Requested	Actual	Requested	Actual
☑ clk_out1	clk_out1	166.667	166.667	0.000	0.000	50.000	50.0
☑ clk_out2	clk_out2	200.000	200.000	0.000	0.000	50.000	50.0
☑ clk_out3	clk_out3	25.000	25.000	0.000	0.000	50.000	50.0
□ clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A
□ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A
□ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A
□ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A

□ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

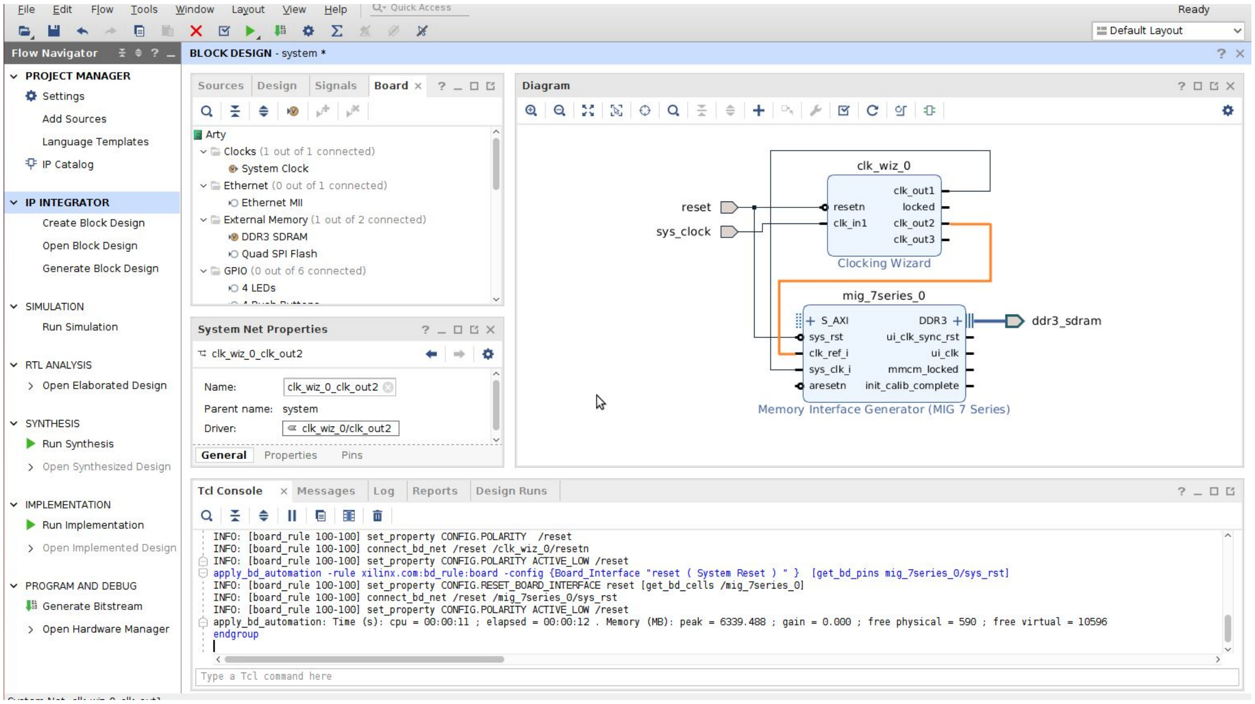
Source

● Automatic Control On-Chip
○ Automatic Control Off-Chip
○ User-Controlled On-Chip
○ User-Controlled Off-Chip

Signaling

● Single-ended
○ Differential

- At this moment we have the system clock sources. Next step is to include the **DDR3 SDRAM**. Grab the DDR3 block to the diagram, two ports are created, **clk_ref_i** and **sys_clk_i**, **delete them** and connect **clk_out1 to sys_clk_i** and **clk_out2 to clk_ref_i**.
- At the top of the design diagram view, a green banner is shown, click on Run Connection Automation, select all options and click Ok. At this moment your system must look like this screenshot.



Name: clk_wiz_0_clk_out2

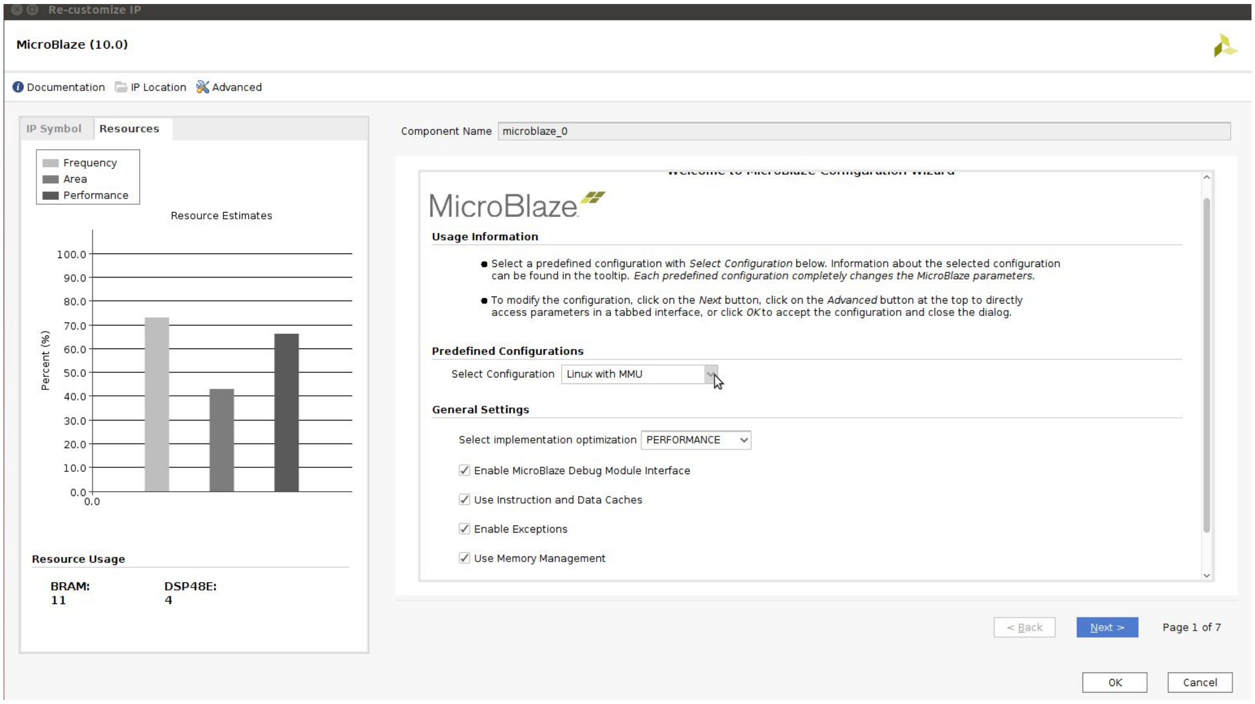
Parent name: system

Driver: clk_wiz_0/clk_out2

```
INFO: [board_rule 100-100] set_property CONFIG.POLARITY  /reset
INFO: [board_rule 100-100] connect_bd_net /reset /clk_wiz_0/resetn
INFO: [board_rule 100-100] set_property CONFIG.POLARITY ACTIVE_LOW /reset
apply_bd_automation -rule xilinx.com:bd_rule:board -config {Board_Interface "reset ( System Reset ) " }  [get_bd_pins mig_7series_0/sys_rst]
INFO: [board_rule 100-100] set_property CONFIG.RESET_BOARD_INTERFACE reset [get_bd_cells /mig_7series_0]
INFO: [board_rule 100-100] connect_bd_net /reset /mig_7series_0/sys_rst
INFO: [board_rule 100-100] set_property CONFIG.POLARITY ACTIVE_LOW /reset
apply_bd_automation: Time (s): cpu = 00:00:11 ; elapsed = 00:00:12 . Memory (MB): peak = 6339.488 ; gain = 0.000 ; free physical = 590 ; free virtual = 10596
endgroup
```

Type a Tcl command here

- Right click on the Diagram view background and select Add IP..., search for **Microblaze** and include it. Double click on the Microblaze block and under **Resources**, **Predefined Configurations**, select **Linux with MMU**.



Resource Usage

BRAM: 11

DSP48E: 4

Component Name: microblaze_0

Welcome to MicroBlaze Configuration Wizard

Usage Information

- Select a predefined configuration with *Select Configuration* below. Information about the selected configuration can be found in the tooltip. *Each predefined configuration completely changes the MicroBlaze parameters.*
- To modify the configuration, click on the *Next* button, click on the *Advanced* button at the top to directly access parameters in a tabbed interface, or click *OK* to accept the configuration and close the dialog.

Predefined Configurations

Select Configuration: Linux with MMU

General Settings

Select implementation optimization: PERFORMANCE

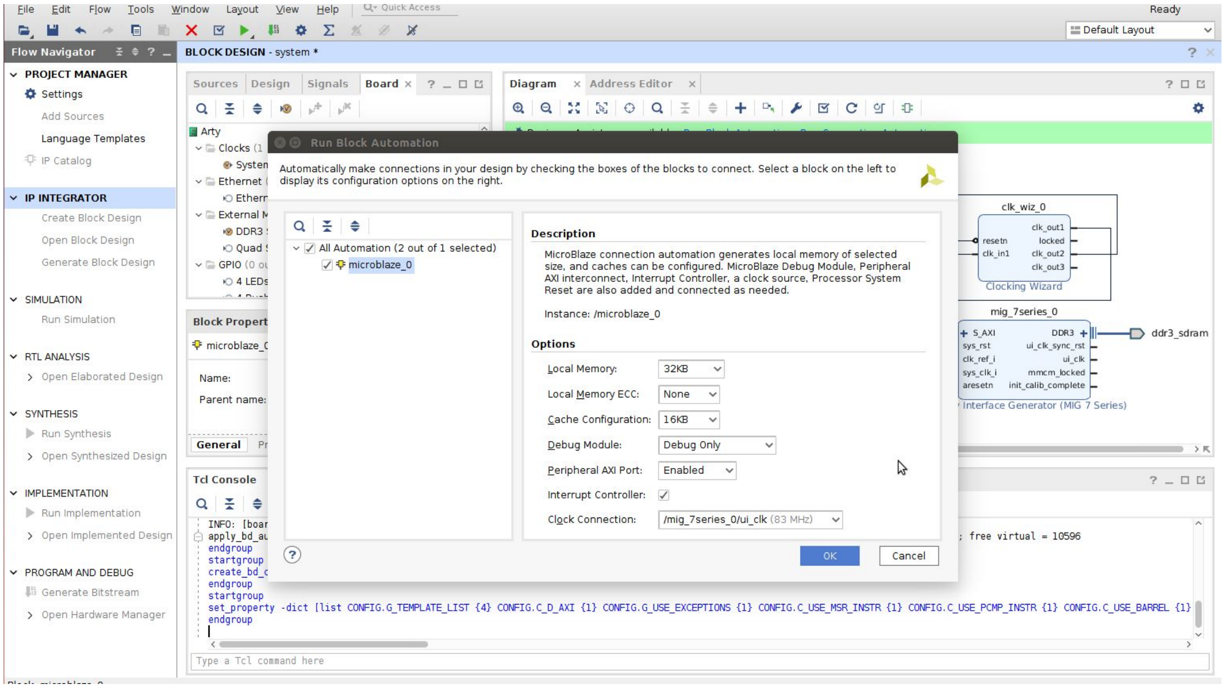
☑ Enable MicroBlaze Debug Module Interface

☑ Use Instruction and Data Caches

☑ Enable Exceptions

☑ Use Memory Management

- Run Block Automation from the green banner and select the following configuration:
  - Local Memory: 32KB
  - Cache: 16KB
  - Enable "Interrupt Controller
  - Clock Connection: ui_clk (83 MHz)



Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

☑ All Automation (2 out of 1 selected)
☑ microblaze_0

Description

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Options

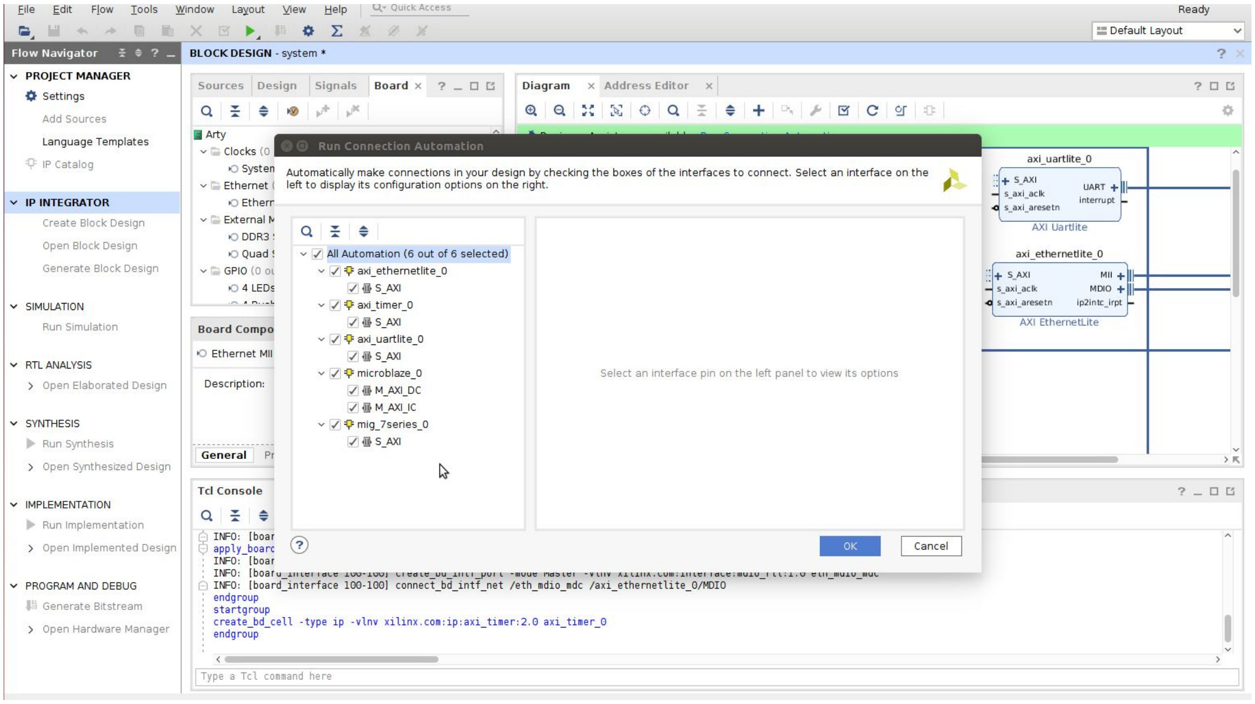
Local Memory: 32KB
Local Memory ECC: None
Cache Configuration: 16KB
Debug Module: Debug Only
Peripheral AXI Port: Enabled
Interrupt Controller: ☑
Clock Connection: /mig_7series_0/ui_clk (83 MHz)

OK   Cancel

```
INFO: [boar                                                     ; free virtual = 10596
apply_bd_au
endgroup
startgroup
create_bd_c
endgroup
startgroup
set_property -dict [list CONFIG.G_TEMPLATE_LIST {4} CONFIG.C_D_AXI {1} CONFIG.G_USE_EXCEPTIONS {1} CONFIG.C_USE_MSR_INSTR {1} CONFIG.C_USE_PCMP_INSTR {1} CONFIG.C_USE_BARREL {1}
endgroup
```

Type a Tcl command here

- The next one is the USB UART block. Grab and modify it to change **Baud Rate to 115200** under IP Configuration tab.
- Similarly drag Ethernet MII and Right click on the diagram background > Add IP... and search for **AXI Timer. Run Connection Automation from the green banner, check all again and click OK. Some obsolescence warnings will appear, ignore them.**



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

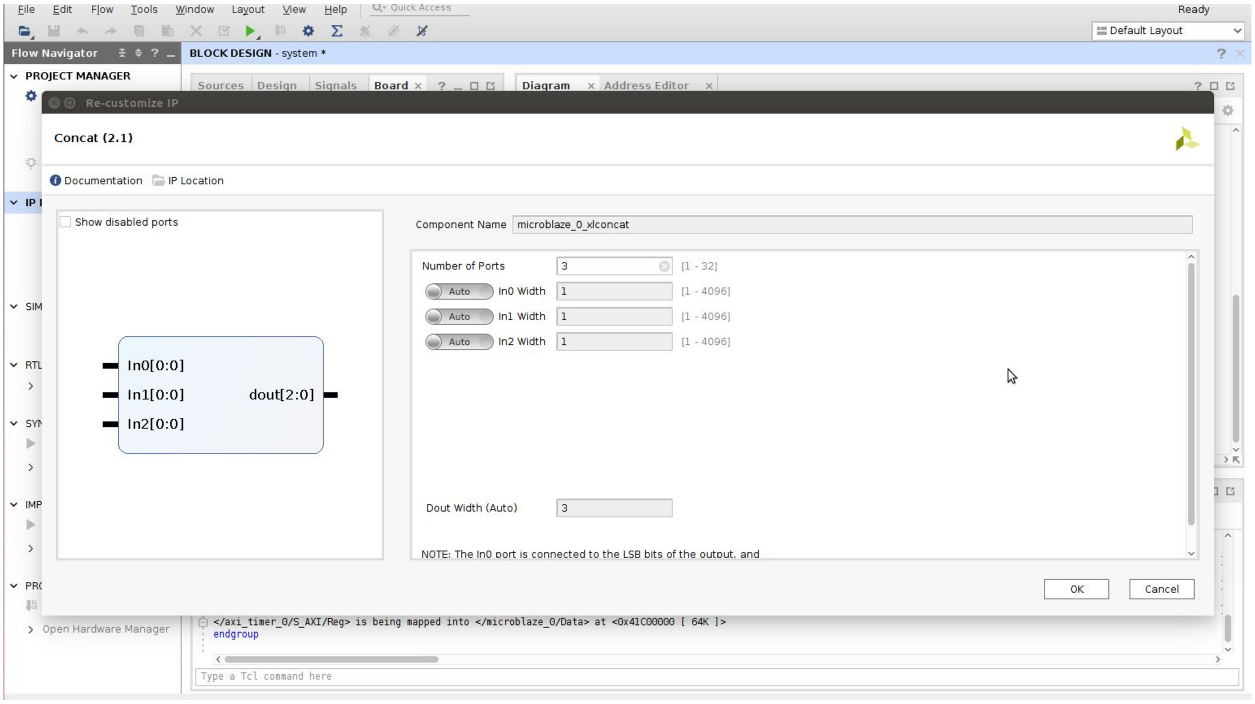
- ☑ All Automation (6 out of 6 selected)
  - ☑ axi_ethernetlite_0
    - ☑ S_AXI
  - ☑ axi_timer_0
    - ☑ S_AXI
  - ☑ axi_uartlite_0
    - ☑ S_AXI
  - ☑ microblaze_0
    - ☑ M_AXI_DC
    - ☑ M_AXI_IC
  - ☑ mig_7series_0
    - ☑ S_AXI

Select an interface pin on the left panel to view its options

OK Cancel

```
INFO: [boar
apply_boarc
INFO: [boar
INFO: [board_interface 100-100] create_bd_intf_port -mode Master -vlnv xilinx.com:interface:mdio_rtl:1.0 eth_mdio_mdc
INFO: [board_interface 100-100] connect_bd_intf_net /eth_mdio_mdc /axi_ethernetlite_0/MDIO
endgroup
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:axi_timer:2.0 axi_timer_0
endgroup
```

- We now have three interrupt sources: UART, Ethernet and Timer. The interrupts controller block is the AXI Interrupt Controller and interrupt input is managed with the Concat block. Double click on Concat and select 3 ports.



Re-customize IP

Concat (2.1)

Documentation IP Location

□ Show disabled ports

Component Name: microblaze_0_xlconcat

Number of Ports: 3 [1 - 32]

Auto In0 Width: 1 [1 - 4096]

Auto In1 Width: 1 [1 - 4096]

Auto In2 Width: 1 [1 - 4096]

Dout Width (Auto): 3

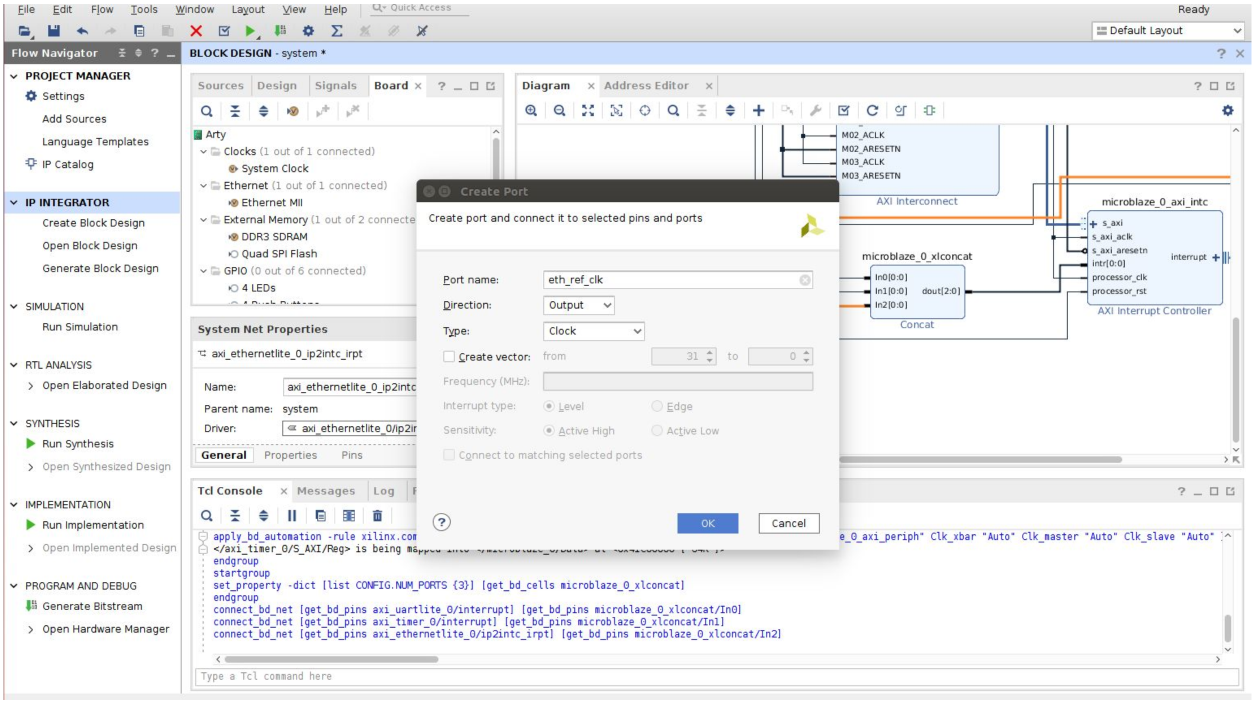
NOTE: The In0 port is connected to the LSB bits of the output. and

OK Cancel

```
</axi_timer_0/S_AXI/Reg> is being mapped into </microblaze_0/Data> at <0x41C00000 [ 64K ]>
endgroup
```

The first port will be the UART. Connect AXI UartLite Interrupt with the In0[0:0] of the Concat block.
The second interrupt will be the Timer. Connect AXI Timer block Interrupt with the In1[0:0].
The third is the Ethernet. Connect **AXI Ethernet Lite ip2itc_irpt to In2[0:0] port of the Concat block.**

- We need to create the input for the Ethernet Reference Clock. Right click on diagram and select Create Port…
  Port name: eth_ref_clk
  Direction: output
  Type: clock
  Connect this **eth_ref_clk to clk_out3 on the Clocking Wizard block**.



Create Port

Create port and connect it to selected pins and ports

Port name: eth_ref_clk
Direction: Output
Type: Clock
□ Create vector: from 31 to 0
Frequency (MHz):
Interrupt type: ● Level ○ Edge
Sensitivity: ● Active High ○ Active Low
□ Connect to matching selected ports

OK Cancel

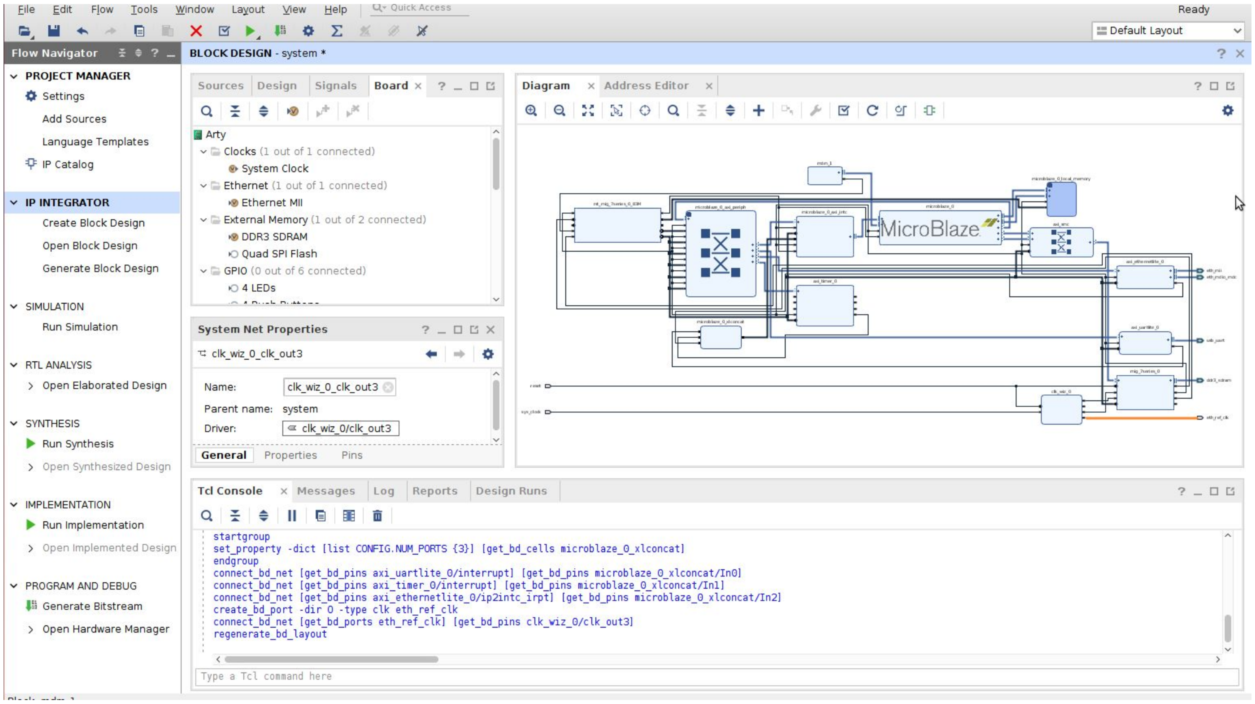
```
apply_bd_automation -rule xilinx.com                                  e_0_axi_periph" Clk_xbar "Auto" Clk_master "Auto" Clk_slave "Auto"
</axi_timer_0/S_AXI/Reg> is being ma
endgroup
startgroup
set_property -dict [list CONFIG.NUM_PORTS {3}] [get_bd_cells microblaze_0_xlconcat]
endgroup
connect_bd_net [get_bd_pins axi_uartlite_0/interrupt] [get_bd_pins microblaze_0_xlconcat/In0]
connect_bd_net [get_bd_pins axi_timer_0/interrupt] [get_bd_pins microblaze_0_xlconcat/In1]
connect_bd_net [get_bd_pins axi_ethernetlite_0/ip2intc_irpt] [get_bd_pins microblaze_0_xlconcat/In2]
```

System Net Properties

axi_ethernetlite_0_ip2intc_irpt

Name: axi_ethernetlite_0_ip2intc
Parent name: system
Driver: axi_ethernetlite_0/ip2ir

- Right click over the diagram and select Regenerate Layout. At this moment your design should look like this:



clk_wiz_0_clk_out3

Name: clk_wiz_0_clk_out3

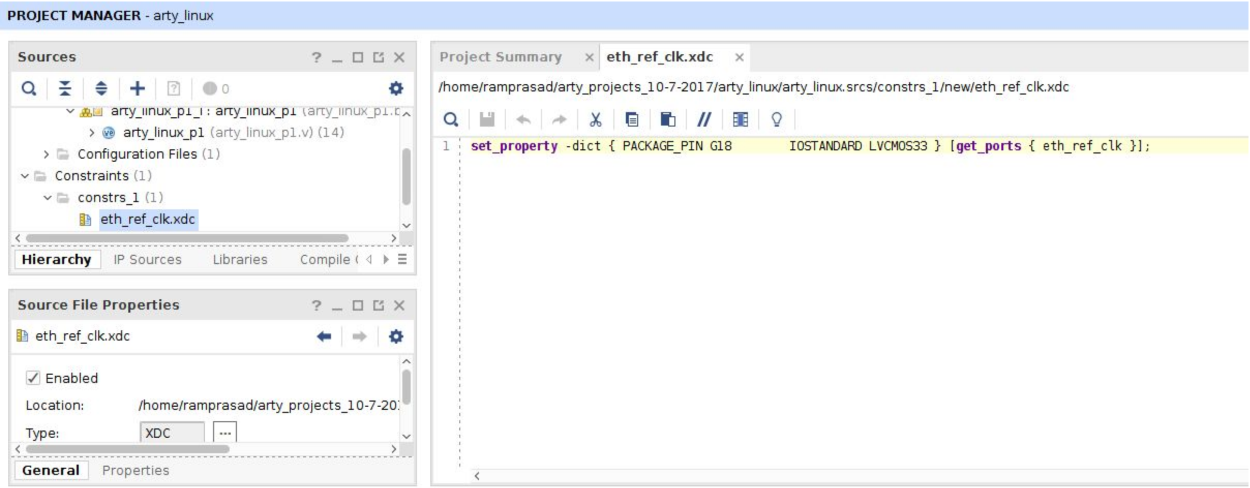
Parent name: system

Driver: clk_wiz_0/clk_out3

```
startgroup
set_property -dict [list CONFIG.NUM_PORTS {3}] [get_bd_cells microblaze_0_xlconcat]
endgroup
connect_bd_net [get_bd_pins axi_uartlite_0/interrupt] [get_bd_pins microblaze_0_xlconcat/In0]
connect_bd_net [get_bd_pins axi_timer_0/interrupt] [get_bd_pins microblaze_0_xlconcat/In1]
connect_bd_net [get_bd_pins axi_ethernetlite_0/ip2intc_irpt] [get_bd_pins microblaze_0_xlconcat/In2]
create_bd_port -dir O -type clk eth_ref_clk
connect_bd_net [get_bd_ports eth_ref_clk] [get_bd_pins clk_wiz_0/clk_out3]
regenerate_bd_layout
```

Type a Tcl command here

- we have finished creating the FPGA system. Right click and select Validate Design. Vivado should tell you that the validation is successful, if not, review all your steps.
- Go on Block Design to Sources Tab, right click on system.bd block and click **Create HDL Wrapper**, Let Vivado manage wrapper and auto-update. It will create all Verilog code connecting your system.
- On the Sources tree, expand constraints and right click over constrs_1, then select Add Sources...Select Add or create constraints and Create File.
  File Type: XDC
  File Name: eth_ref_clk
  File Location: <Local to Project>
  Finish and double click on Constraints > constrs_1 > eth_ref_clk.xdc. Paste the following line into the file: “set_property -dict { PACKAGE_PIN G18
  IOSTANDARD LVCMOS33 } [get_ports { eth_ref_clk }];”



/home/ramprasad/arty_projects_10-7-2017/arty_linux/arty_linux.srcs/constrs_1/new/eth_ref_clk.xdc

```
set_property -dict { PACKAGE_PIN G18       IOSTANDARD LVCMOS33 } [get_ports { eth_ref_clk }];
```

☑ Enabled

Location: /home/ramprasad/arty_projects_10-7-20

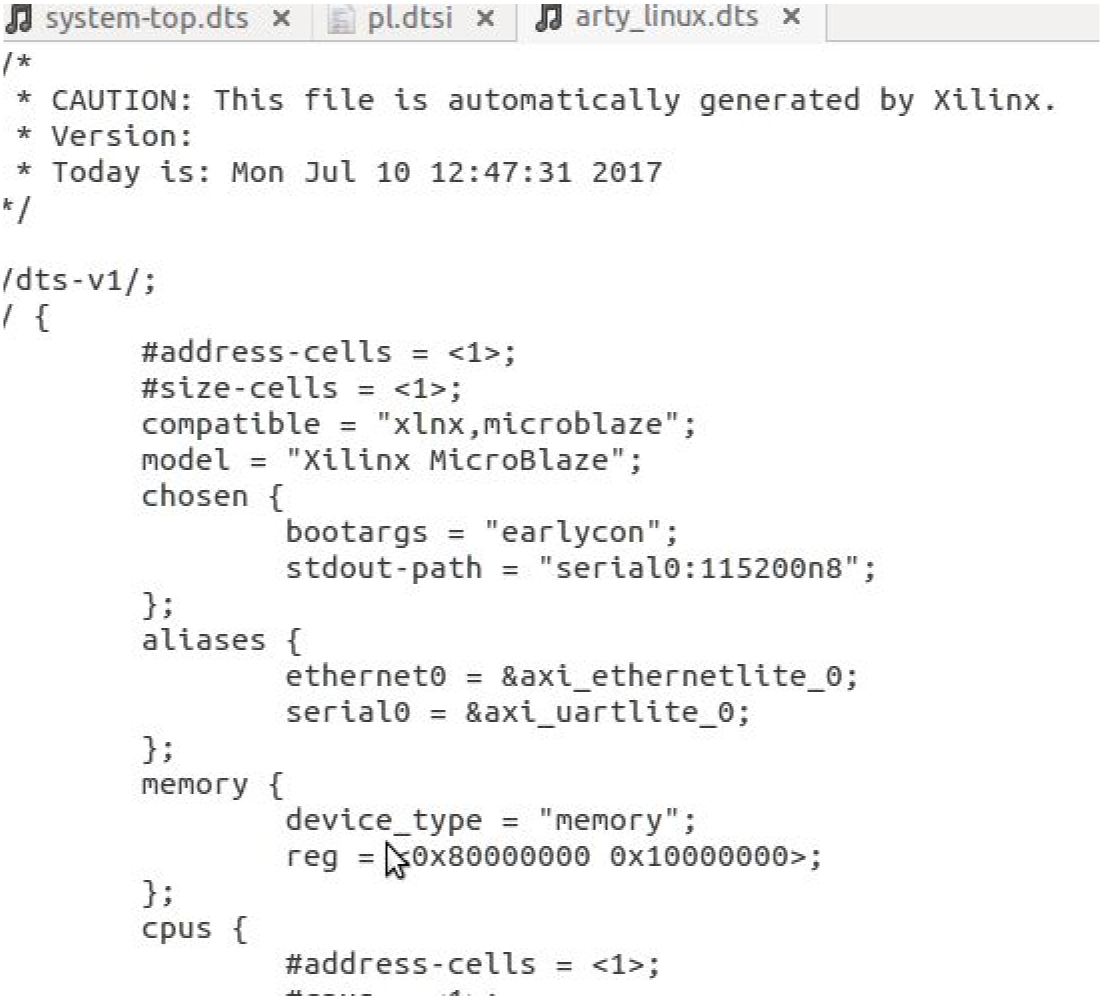
Type: XDC

- Now we are ready to generate th Bitstream file. Go to **Flow Navigator** and click on **Generate Bitstream**. As we did not executed the simulation, elaboration, synthesis nor implementation, Vivado will tell you that bitstream will be generated once synthesized and implemented. Click OK and wait some minutes. It may last from some minutes to an hour.
- Go to File > Export > Export Hardware. Check Include Bitstream.

Linux DTS creation in SDK

- We need to generate is the Linux Device Tree, which contains the hardware data needed by the booter and operative system to know how to deal with the created hardware.
- To create our hardware we used IP cores from Xilinx, like AXI_Uartlite or the own Microblaze so we need to download the linux device tree for Xilinx. Open the terminal and type the following line “git clone https://github.com/Xilinx/device-tree-xlnx” Extract device-tree-xlnx-master on Vivado project *.sdk subfolder.
- Vivado project, go to File > Launch SDK. Once Eclipse is opened, go to Xilinx Tools > Repositories to include the new folder as a repository. Then click New ... and find the extracted “.cd” extension file

- Go to **File > New > Board Support Package** and at the bottom, select device_tree. Default settings are ok to continue so click on Finish to create the project.(In the future if you want to change the BSP settings you can double click to the .mss file and modify them.)
- It is time to create a .dts file containing the Linux Device Tree which will be included in the buildroot to build the kernel. We are going to merge the files "pl.dtsi" and "system_top.dts" created by the SDK.
- Open terminal and go to the directory where the pl.dtsi file is present. It will be present in the ".sdk" directory that has been created.
- Open both the files using "gedit". From system_top.dts file copy the line copy the /dts-v1/; to the "pl.dtsi" file, previous to "/ {". In the same way, copy from "chosen {" to the closing "};" before the "cpus {" line. It would be a bit vi tedious to understand the copying step. But it would be easy if you open both the files and look for the lines. Let me help you with that. After copying and pasting the lines from system_top.dts to pl.dtsi the resultant file (which I named as 'arty_linux.dts') should be looking like the following screenshot.



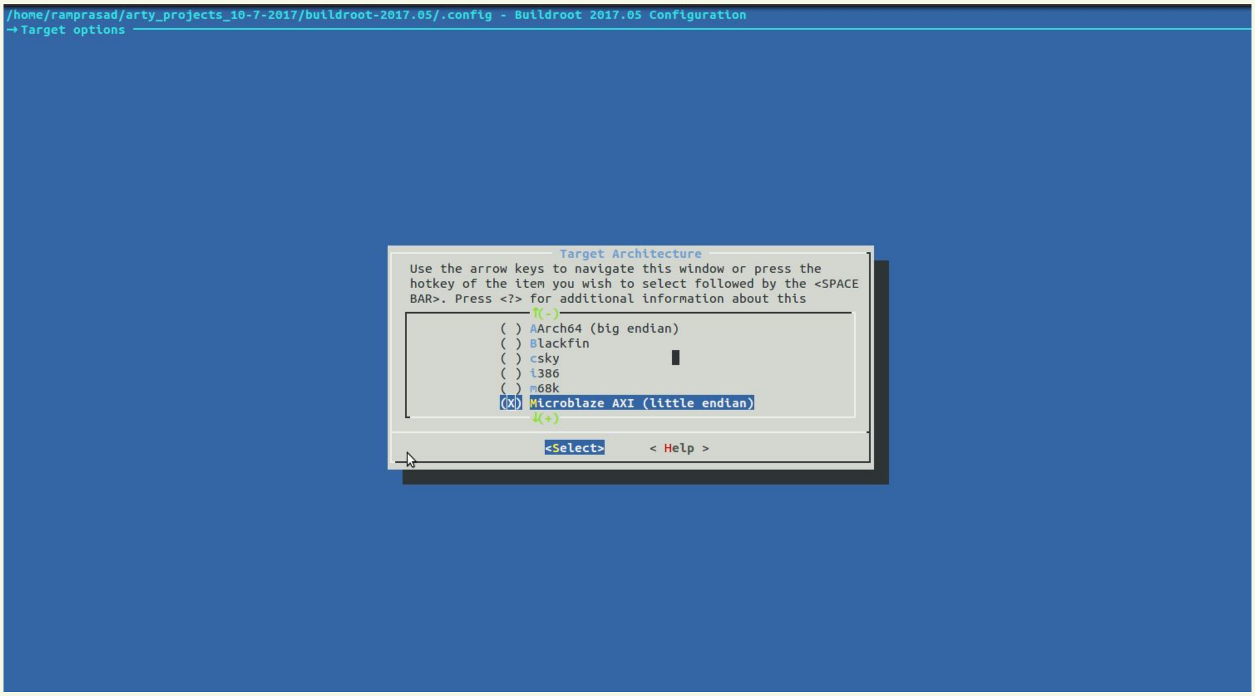
```
/*
 * CAUTION: This file is automatically generated by Xilinx.
 * Version:
 * Today is: Mon Jul 10 12:47:31 2017
*/

/dts-v1/;
/ {
	#address-cells = <1>;
	#size-cells = <1>;
	compatible = "xlnx,microblaze";
	model = "Xilinx MicroBlaze";
	chosen {
		bootargs = "earlycon";
		stdout-path = "serial0:115200n8";
	};
	aliases {
		ethernet0 = &axi_ethernetlite_0;
		serial0 = &axi_uartlite_0;
	};
	memory {
		device_type = "memory";
		reg = <0x80000000 0x10000000>;
	};
	cpus {
		#address-cells = <1>;
```

Building Linux Through BuildRoot/Busybox Method

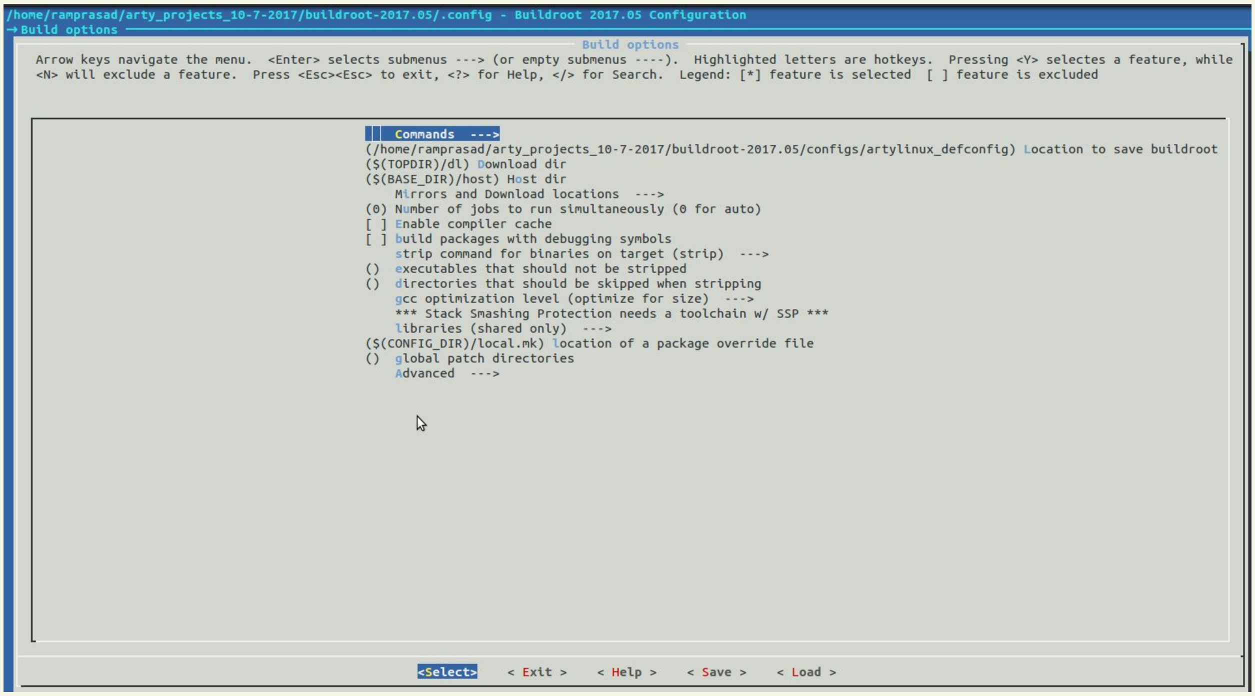
- Now we need is a computer that can run Linux. We have to download FPGA Bitfile.
- First we should install the needed application for compilation. For that go to terminal and type
  “sudo apt-get install build-essential bison flex gettext libncurses5-dev texinfo autoconf automake libtool”

- Create a working folder and into it, download the buildroot version 2017.5. I included it also at the end of this tutorial, and extract it.
  “wget http://buildroot.uclibc.org/downloads/buildroot-2017.5.tar.gz
  tar -xvf buildroot-2017.5.tar.gz
  cd buildroot-2017.5”
- We need to include our device tree information so we create the board/arty folder and copy our artylinux.dts file into it. In terminal type
  mkdir board/arty
  cp ../../artylinux.dts board/arty/

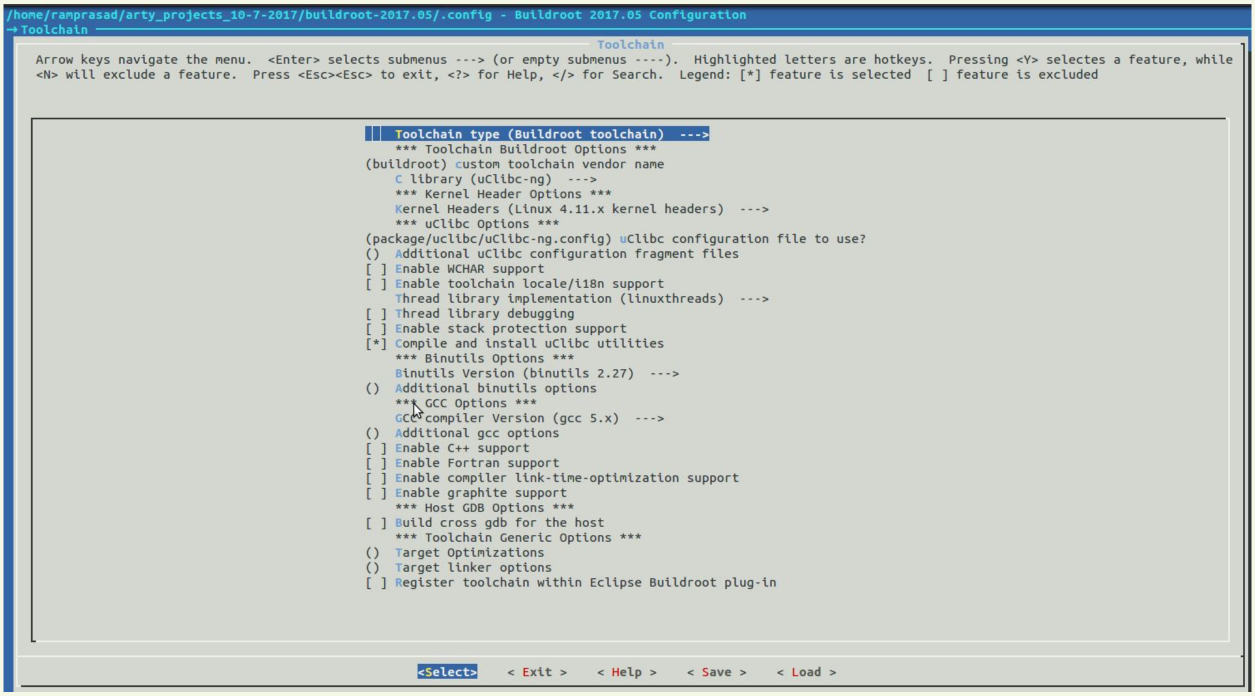
- The next step is to define the kernel properties. This procedure can be done through different ways but the most graphical is to use the menuconfig option from the makefile. In terminal type Now open a terminal on the Arty Serial COM port and click on the Prog button (near the micro USB) to reload the FPGA config. The bootloader process will appear and after few seconds the Linux image will be Now open a terminal on the Arty Serial COM port and click on the Prog button (near the micro USB) to reload the FPGA config. The bootloader process will appear and after few seconds the Linux image will be “make menuconfig”.



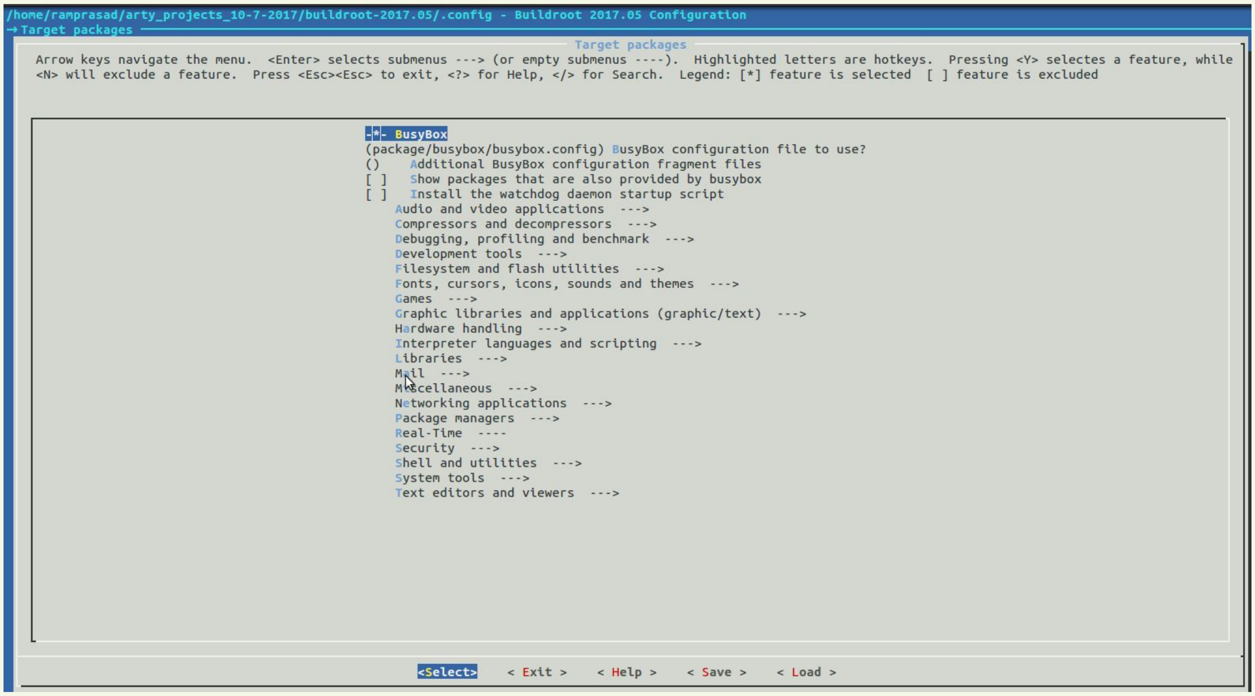
```
/home/ramprasad/arty_projects_10-7-2017/buildroot-2017.05/.config - Buildroot 2017.05 Configuration
→ Target options
                              Target Architecture
  Use the arrow keys to navigate this window or press the
  hotkey of the item you wish to select followed by the <SPACE
  BAR>. Press <?> for additional information about this
                 ↑(-)
             ( ) AArch64 (big endian)
             ( ) Blackfin
             ( ) csky
             ( ) i386
             ( ) m68k
             (X) Microblaze AXI (little endian)
                 ↓(+)

                 <Select>      < Help >
```



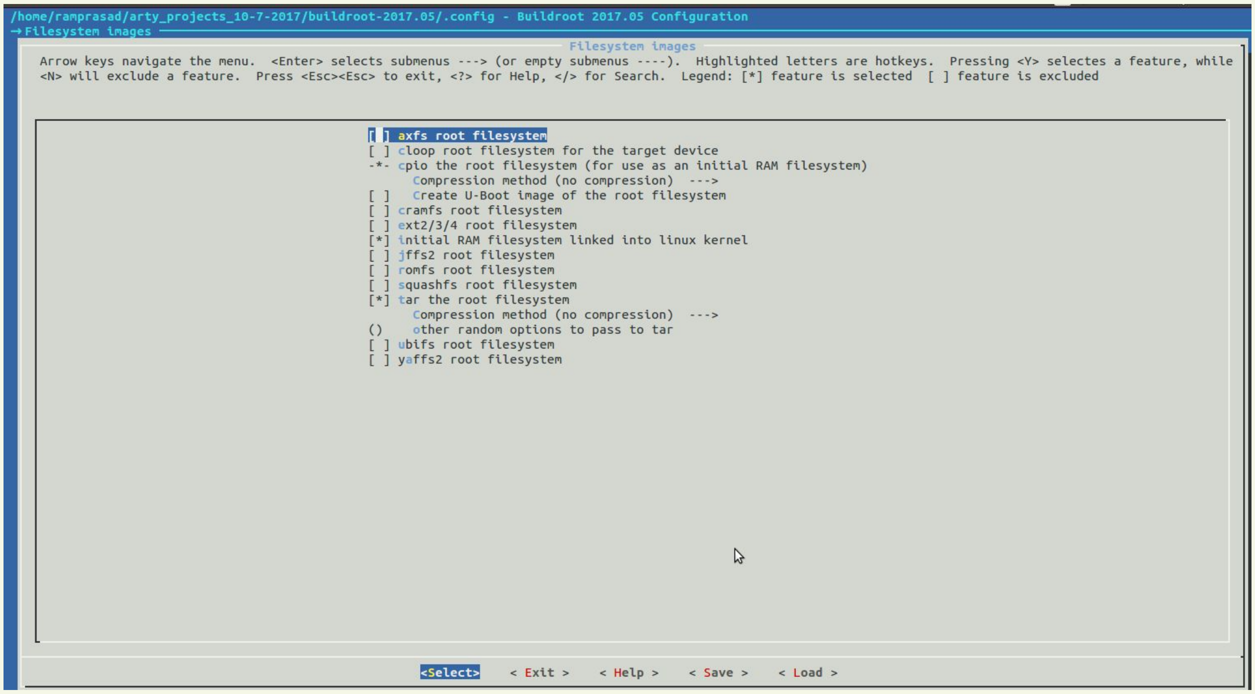
```
/home/ramprasad/arty_projects_10-7-2017/buildroot-2017.05/.config - Buildroot 2017.05 Configuration
→ Build options
                                            Build options
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
 <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded

                                    Commands  --->
                                (/home/ramprasad/arty_projects_10-7-2017/buildroot-2017.05/configs/artylinux_defconfig) Location to save buildroot
                                ($(TOPDIR)/dl) Download dir
                                ($(BASE_DIR)/host) Host dir
                                    Mirrors and Download locations  --->
                                (0) Number of jobs to run simultaneously (0 for auto)
                                [ ] Enable compiler cache
                                [ ] build packages with debugging symbols
                                    strip command for binaries on target (strip)  --->
                                ()  executables that should not be stripped
                                ()  directories that should be skipped when stripping
                                    gcc optimization level (optimize for size)  --->
                                    *** Stack Smashing Protection needs a toolchain w/ SSP ***
                                    libraries (shared only)  --->
                                ($(CONFIG_DIR)/local.mk) location of a package override file
                                ()  global patch directories
                                    Advanced  --->

                          <Select>    < Exit >    < Help >    < Save >    < Load >
```



```
/home/ramprasad/arty_projects_10-7-2017/buildroot-2017.05/.config - Buildroot 2017.05 Configuration
→ Toolchain
                                                        Toolchain
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
 <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded

                                        Toolchain type (Buildroot toolchain)  --->
                                        *** Toolchain Buildroot Options ***
                                    (buildroot) custom toolchain vendor name
                                        C library (uClibc-ng)  --->
                                        *** Kernel Header Options ***
                                        Kernel Headers (Linux 4.11.x kernel headers)  --->
                                        *** uClibc Options ***
                                    (package/uclibc/uClibc-ng.config) uClibc configuration file to use?
                                    ()  Additional uClibc configuration fragment files
                                    [ ] Enable WCHAR support
                                    [ ] Enable toolchain locale/i18n support
                                        Thread library implementation (linuxthreads)  --->
                                    [ ] Thread library debugging
                                    [ ] Enable stack protection support
                                    [*] Compile and install uClibc utilities
                                        *** Binutils Options ***
                                        Binutils Version (binutils 2.27)  --->
                                    ()  Additional binutils options
                                        *** GCC Options ***
                                        GCC compiler Version (gcc 5.x)  --->
                                    ()  Additional gcc options
                                    [ ] Enable C++ support
                                    [ ] Enable Fortran support
                                    [ ] Enable compiler link-time-optimization support
                                    [ ] Enable graphite support
                                        *** Host GDB Options ***
                                    [ ] Build cross gdb for the host
                                        *** Toolchain Generic Options ***
                                    ()  Target Optimizations
                                    ()  Target linker options
                                    [ ] Register toolchain within Eclipse Buildroot plug-in


                                  <Select>    < Exit >    < Help >    < Save >    < Load >
```



```
/home/ramprasad/arty_projects_10-7-2017/buildroot-2017.05/.config - Buildroot 2017.05 Configuration
→ Target packages
                                         Target packages
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
  <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded

                                    -*- BusyBox
                                    (package/busybox/busybox.config) BusyBox configuration file to use?
                                    ()    Additional BusyBox configuration fragment files
                                    [ ]   Show packages that are also provided by busybox
                                    [ ]   Install the watchdog daemon startup script
                                        Audio and video applications  --->
                                        Compressors and decompressors  --->
                                        Debugging, profiling and benchmark  --->
                                        Development tools  --->
                                        Filesystem and flash utilities  --->
                                        Fonts, cursors, icons, sounds and themes  --->
                                        Games  --->
                                        Graphic libraries and applications (graphic/text)  --->
                                        Hardware handling  --->
                                        Interpreter languages and scripting  --->
                                        Libraries  --->
                                        Mail  --->
                                        Miscellaneous  --->
                                        Networking applications  --->
                                        Package managers  --->
                                        Real-Time  ----
                                        Security  --->
                                        Shell and utilities  --->
                                        System tools  --->
                                        Text editors and viewers  --->

                              <Select>    < Exit >    < Help >    < Save >    < Load >
```



```
/home/ramprasad/arty_projects_10-7-2017/buildroot-2017.05/.config - Buildroot 2017.05 Configuration
→ Filesystem images
                                         Filesystem images
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while
  <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded

                                    [ ] axfs root filesystem
                                    [ ] cloop root filesystem for the target device
                                    -*- cpio the root filesystem (for use as an initial RAM filesystem)
                                          Compression method (no compression)  --->
                                    [ ]   Create U-Boot image of the root filesystem
                                    [ ] cramfs root filesystem
                                    [ ] ext2/3/4 root filesystem
                                    [*] initial RAM filesystem linked into linux kernel
                                    [ ] jffs2 root filesystem
                                    [ ] romfs root filesystem
                                    [ ] squashfs root filesystem
                                    [*] tar the root filesystem
                                          Compression method (no compression)  --->
                                    ()    other random options to pass to tar
                                    [ ] ubifs root filesystem
                                    [ ] yaffs2 root filesystem

                              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Download and testing the bitfile and linux image:

- In the SDK, go to Xilinx Tools > Program FPGA. As we exported the SDK including bitstream, the hardware system wrapper is here configured. Click on Program and wait until the process finishes.
- Copy your created Linux image to the .sdk project subfolder.
- In the terminal type <XMD> and the XMD console will be opened.
- In the XMD console type <connect mb mdm>
- Now go to the directory which had the image file with <.elf> extension using cd command.
- Now we have to download the image file. Type <dow .elf> in the xmd console.
- After the download is completed type <con 0x80000000>
- Now it will ask for the user id and password. The user id is ‘root’ and the password is also ‘root’.

Petalinux Flow

Project Initialisation:

- Source the PetaLinux package, It will check the requirement and indicate if any missing package. Install if packages requires.

- Type in command line $ source <Peta linux installation directory>/settings.sh

Creating a New Project:

- To create a new package we need to use petalinux-create command.

<petalinux-create -t project --template microblaze --name projec_name>

(In this step i have created a new project name called peta_p2. I have kept the peta_p2 in a directory called petalinux_arty_fullsystem along with the .hdf file)

- Import the hardware description file to the project. To do so we should use petalinux-config command.
  $ petalinux-config --get-hw-description -p <project directory>
  (here the project directory denotes the project that we created i.e peta_p2)

- It’ll open config-menu, where we left it to be default configuration. Save the configuration.

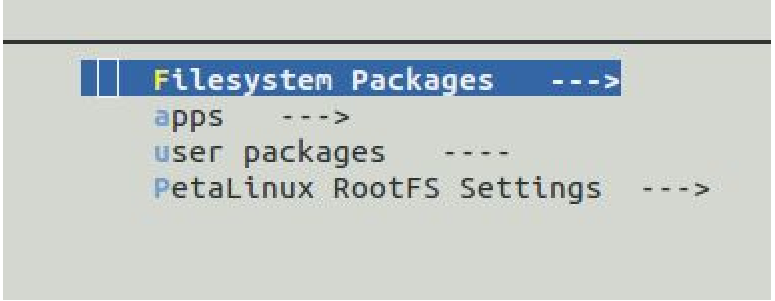
Add User Application to PetaLinux:

- To create a user application, we again need the petalinux-create command.
- From the project directory enter the following command.
  ```
  $ petalinux-create -t apps --template c++ --name app1
  ```
- The new apps will be created in the project folder in “...../recipes-apps/gpioapp”
- The above command will create a app1 directory in <project_directory>/project-spec/meta-user/recipes-apps --template c++
- From the above command will create a basic c++ template of Hello World! program source code in the files directory. Add your custom source code in that directory and modify the Makefile to compile all your source codes.
- Now the application is ready for compilation

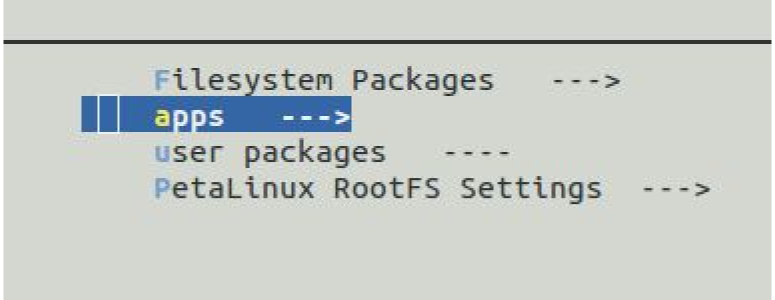
Compile and building the PetaLinux:

- We are building our application with the petalinux build. We have to tell petalinux that we wrote a application and we need to build it. To do so we should use petalinux-config command.
  ```
  $ petalinux-config -c kernel
  $ petalinux-config -c rootfs
  ```

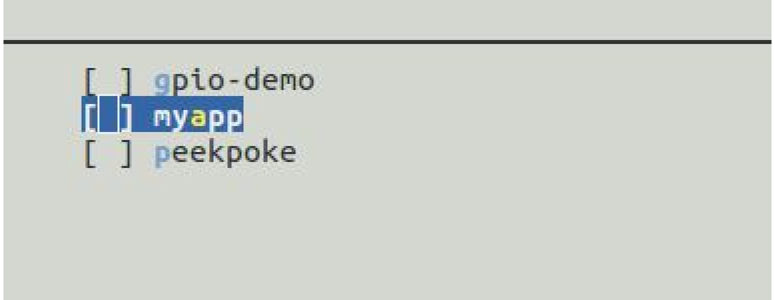
- The menu config for root file system will open. We have to choose apps. In that menu our app will be visible and we need to toggle on our app. Save the config file and then petalinux is ready for final building.



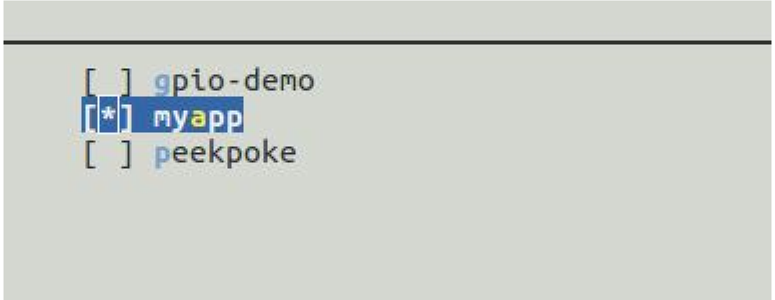
```
    Filesystem Packages   --->
    apps   --->
    user packages   ----
    PetaLinux RootFS Settings  --->
```



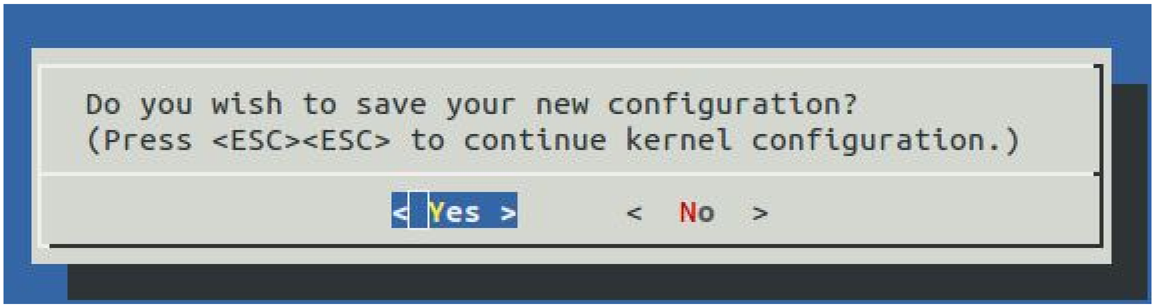
```
    Filesystem Packages   --->
    apps   --->
    user packages   ----
    PetaLinux RootFS Settings  --->
```



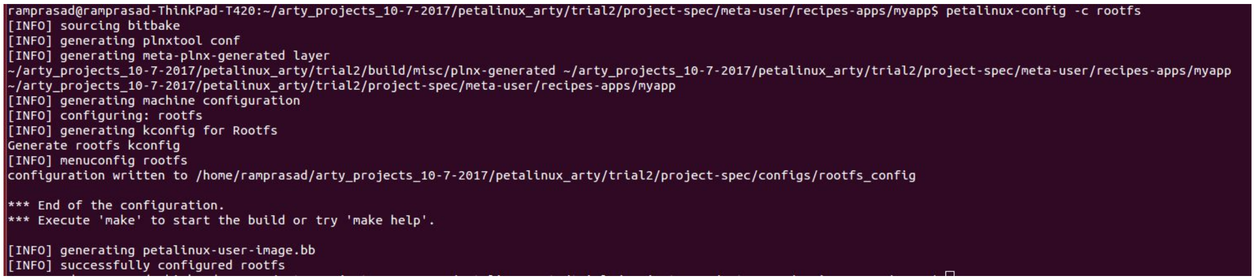
```
[ ] gpio-demo
[ ] myapp
[ ] peekpoke
```



```
[ ] gpio-demo
[*] myapp
[ ] peekpoke
```



```
Do you wish to save your new configuration?
(Press <ESC><ESC> to continue kernel configuration.)
```



```
ramprasad@ramprasad-ThinkPad-T420:~/arty_projects_10-7-2017/petalinux_arty/trial2/project-spec/meta-user/recipes-apps/myapp$ petalinux-config -c rootfs
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/arty_projects_10-7-2017/petalinux_arty/trial2/build/misc/plnx-generated ~/arty_projects_10-7-2017/petalinux_arty/trial2/project-spec/meta-user/recipes-apps/myapp
~/arty_projects_10-7-2017/petalinux_arty/trial2/project-spec/meta-user/recipes-apps/myapp
[INFO] generating machine configuration
[INFO] configuring: rootfs
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] menuconfig rootfs
configuration written to /home/ramprasad/arty_projects_10-7-2017/petalinux_arty/trial2/project-spec/configs/rootfs_config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[INFO] generating petalinux-user-image.bb
[INFO] successfully configured rootfs
```

- The build command that will build the petalinux is “petalinux-build”

- This may take some time for building.

Boot the Linux in emulator:

- To boot the linux image to the emulator we must use the following command. <petalinux-boot --qemu --kernel>
- Now the petalinux will be booted in the emulator and is ready for use. Id root and password root.
- To run the app in the emulator just call the app project name.
# app1

Download bitfile and Linux Image:

- Open the created Peta linux directory.
- Now we should source the Peta Linux. For that go to petabinaries/settings.sh
- Now the petalinux get sourced.
- Go to the directory that has image file. It will be with .elf extension. The file be in …./images/linux/images.elf
- Type <dow image.elf>
- This will download the image file in the board.

- Open a new terminal and type <sudo putty -serial /dev/tty USB1 -serfg 115200,8,n,1,N>
- Sometime the error may occur while downloading the image file. In that case type <stop> in the command line and then type <reset>. Again try to download the image file.

Flash Image and Bit file into Board:

- On Vivado, open the synthesis view, clicking over "Open Synthesized Design", then go to Tools > Edit Device Properties... and select the following settings:
    - General > **Enable Bitstream compression: YES**
    - **Configuration > Configuration Rate (MHz): 33**
    - **Configuration Modes > Select Master SPI x4 option**
- Click on BitStream Settings and Select .bin file option and click generated bitstream.
- Now we have to program the target device.In hardware manager tab click on program device and open the target device.
- Click on xc7a35t part and select **Add Configuration Memory Device.**
- **Select Micron N25Q128-3.3V and right click over the appeared device on the Hardware list and select Program Configuration Memory Device.**
- **Search for the .bin image and click over Program file and press ok.**

Bootloader:

- Go to Vivado > File > **Launch SDK**
- In the SDK, create a new project going to **File > New > Application Project**
- Input your project name and click Next. Then select the "**SREC SPI Bootloader**" **template**.
- In the SDK open the the generated *_bsp folder, right click over it and select **Board Support Package Settings**. At **Overview > Standalone > xilisf** settings, set the **serial_flash_family value to 5**. This defines that we have a **Micron Quad SPI flash device**.
- Now in the bootloader project source folder, open the blconfig.h file and set the **FLASH_IMAGE_BASEADDR** define value.
- Define the **FLASH_IMAGE_SIZE** in the bootloader.c source, for example if the size is 0x00900000,then it denotes 9 Mb.
- In Vivado, open the **Project Manager view** and go to **Tools > Associate ELF Files.**

- For Design and Simulation Sources, search for the **.elf file** and regenerate the Bitstream sources.
- go to **Hardware Manager** view and then go to **Tools > Generate Memory Configuration File**.In it create the .mcs file.
- When the .mcs file is created we have to again program the device with the .mcs file.
- This may take some time to get executed. When it gets completed press the program button in the board.
- Now the bootloader process will appear and after few seconds the Linux image will be starting to boot.

GPIO Application Run and Test:

- Go to the Putty terminal and type < gpio -demo -g 508 -o 0xf >
- Here 508 is the address of the GPIO port. To find the address of the port we have to go to the root directory.
- That path will be present in the .c file in the project directory.
- The path is </sys/class/gpio>. In the putty directory go that path and you will the root file there. Open it to find the address of the pins.
- Likewise we have to choose the address of the gpio ports to run the application.
- Example for GPIO input : command is < gpio -demo -g <gpioaddress> -i 0xf >. Here -i denotes the input type command.
- Example for GPIO output : command is <gpio -demo -g 508 -o 0xf>. Here -o denotes the output type of command.