

# Embedded System Design Lab Course

Rouven, Sascha, Uli / Mainz

Loosely based on slides from last year's embedded course at  
Bonn Univ.

# Outline of the Lab Course

1: Creating an embedded system

2: Adding EDK IP

...

3: Linux on embedded system

Some remarks:

The tutorial is using Xilinx “PlanAhead” (rather than ISE). Similar, but...

Since you are ISE experts now, you’ll learn to use the PlanAhead interface on the fly.

Note: PlanAhead is very similar in terms of click and feel to the future design environment “Vivado”. Better get used to it now...

# Embedded design – what's that ?

- FPGAs ideally suited for parallel, synchronous, fixed latency processing
- Simple slow control applications more easily done on micro controller
- Complex, but not speed or latency critical sequential algorithms can be easily described in C, C++, ...
- Several options to offload part of the functionality to a processor:
- Add microcontroller chip on the PCB
- Have embedded processor on (Xilinx) FPGA
  - Soft core : MicroBlaze, PicoBlaze (used for last year's tutorial at Bonn Univ.)
  - Hard core : PowerPC (Virtex-5) or ARM (Zynq)
- We have decided to go for ARM based designs on a Zynq device for this course (just to please the raspberry pi lovers, of course...)

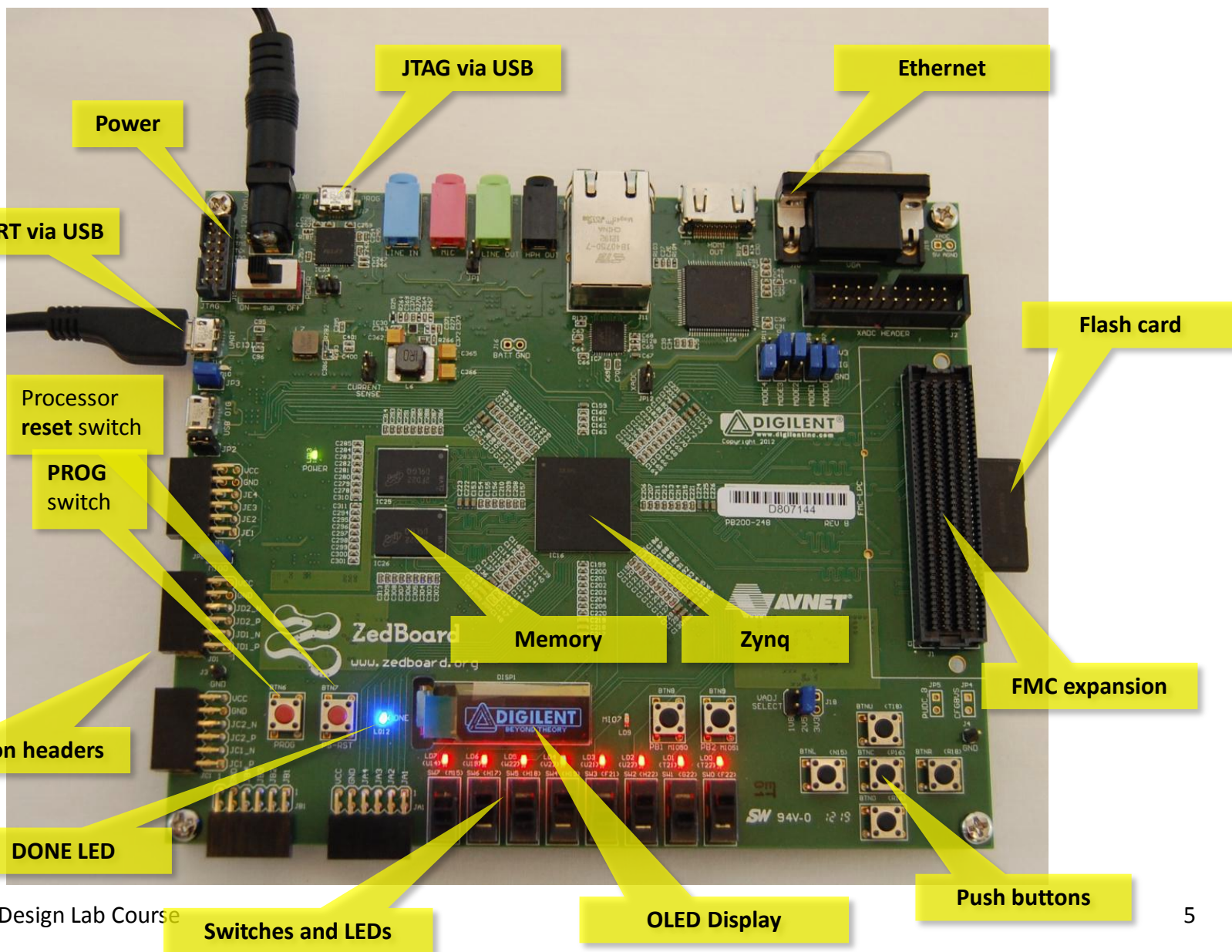
**→ ZedBoard**

# Prerequisites

- Hardware
  - Zedboard [www.zedboard.org](http://www.zedboard.org)
  - Mini-USB cable
  - (Ethernet cable)
- Software
  - PlanAhead 14.4 as installed (including device pack)
  - Probably better to use 32-bit tools
  - Alternatively free webpack software available at [www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm](http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm)
  - To actually run the stuff on hardware – additional Driver:  
<https://secure.cypress.com/?mpn=CY7C64225-28PVXC>  
download “Microsoft Certified USB UART Driver”
  - Putty.exe or any other terminal programme

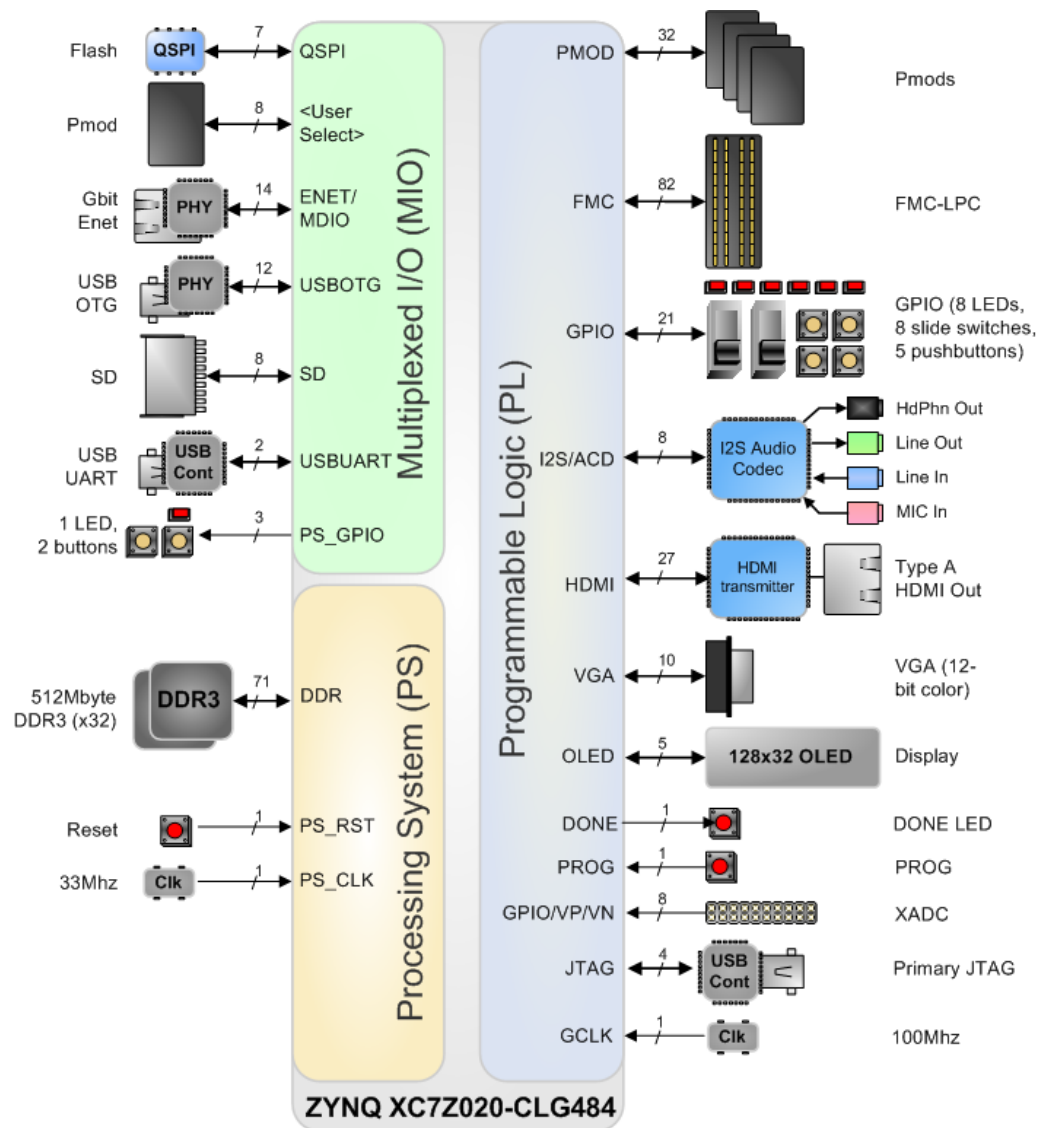
You can follow the design flow on your laptop computer. We’ve got a couple of boards to try out your results...

# ZedBoard



# ZedBoard Block Diagram

- Zynq device composed of an ARM processor and an Artix FPGA core
- Processor wired to typical PC peripheral equipment via MIO ports and to memory
- Some further peripherals connected via FPGA fabric
- FPGA fabric and I/O available for user I/O and functionality
- Ample connectivity via “PMOD” and “FMC” expansion sockets



# Tutorial 1: Creating an Embedded System

## **Scope of the tutorial**

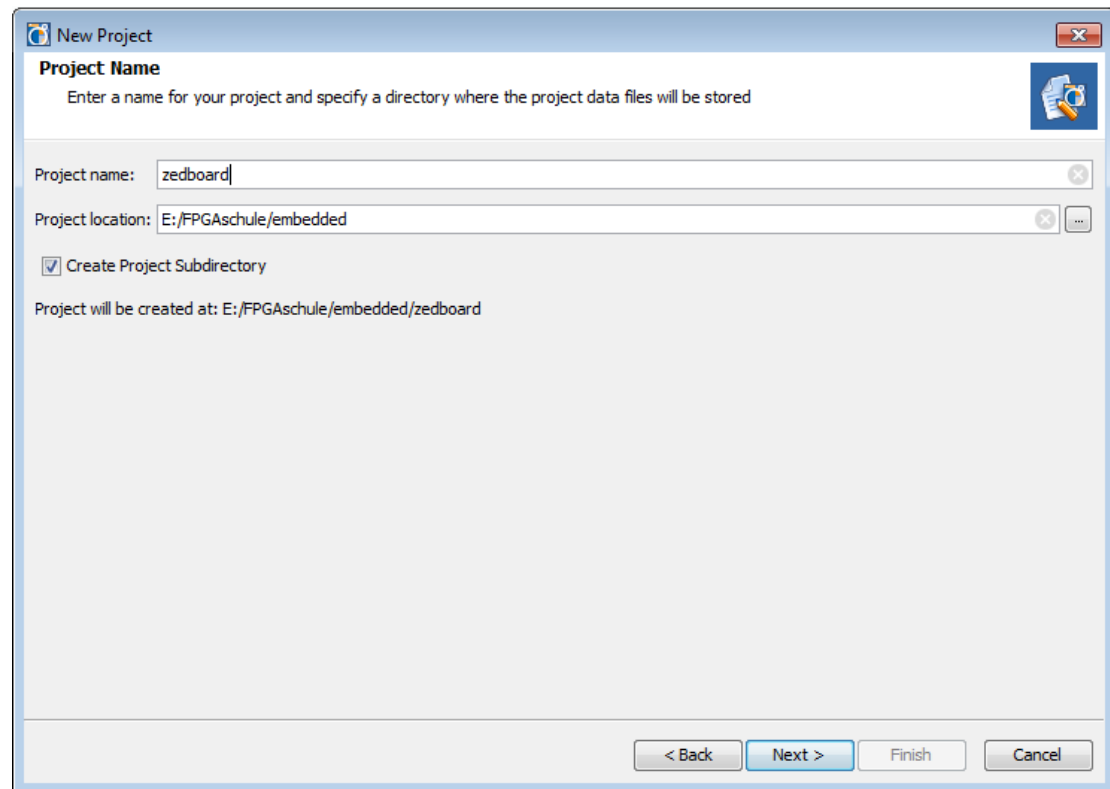
1.1 Generating the hardware platform

1.2 Understanding what has been created

# 1.1 Creating the Hardware Platform

Use **PlanAhead** to create all project files needed for an embedded system.

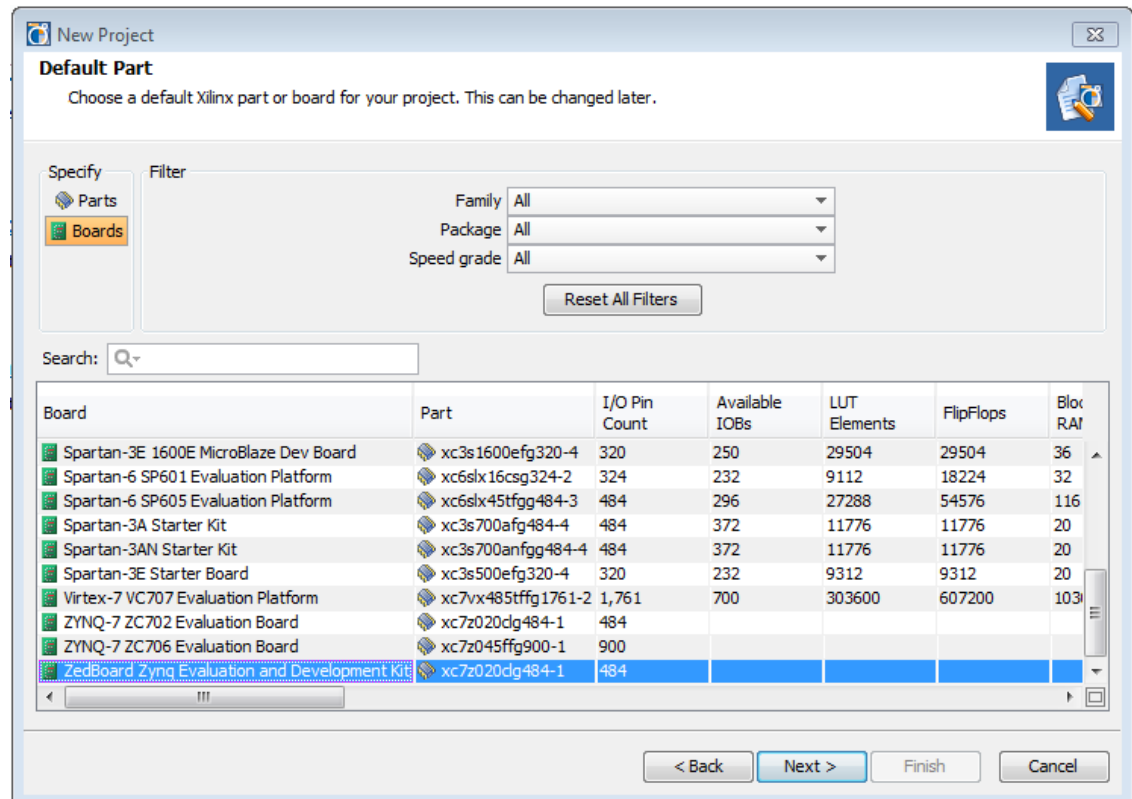
1. Start Xilinx PlanAhead, **Start > Programs > Xilinx ISE Design Suite 14.3 > PlanAhead > PlanAhead** and create a new project.
2. Set the Project Location to **~/embedded** and the Project Name to **zedboard**. Click Next.





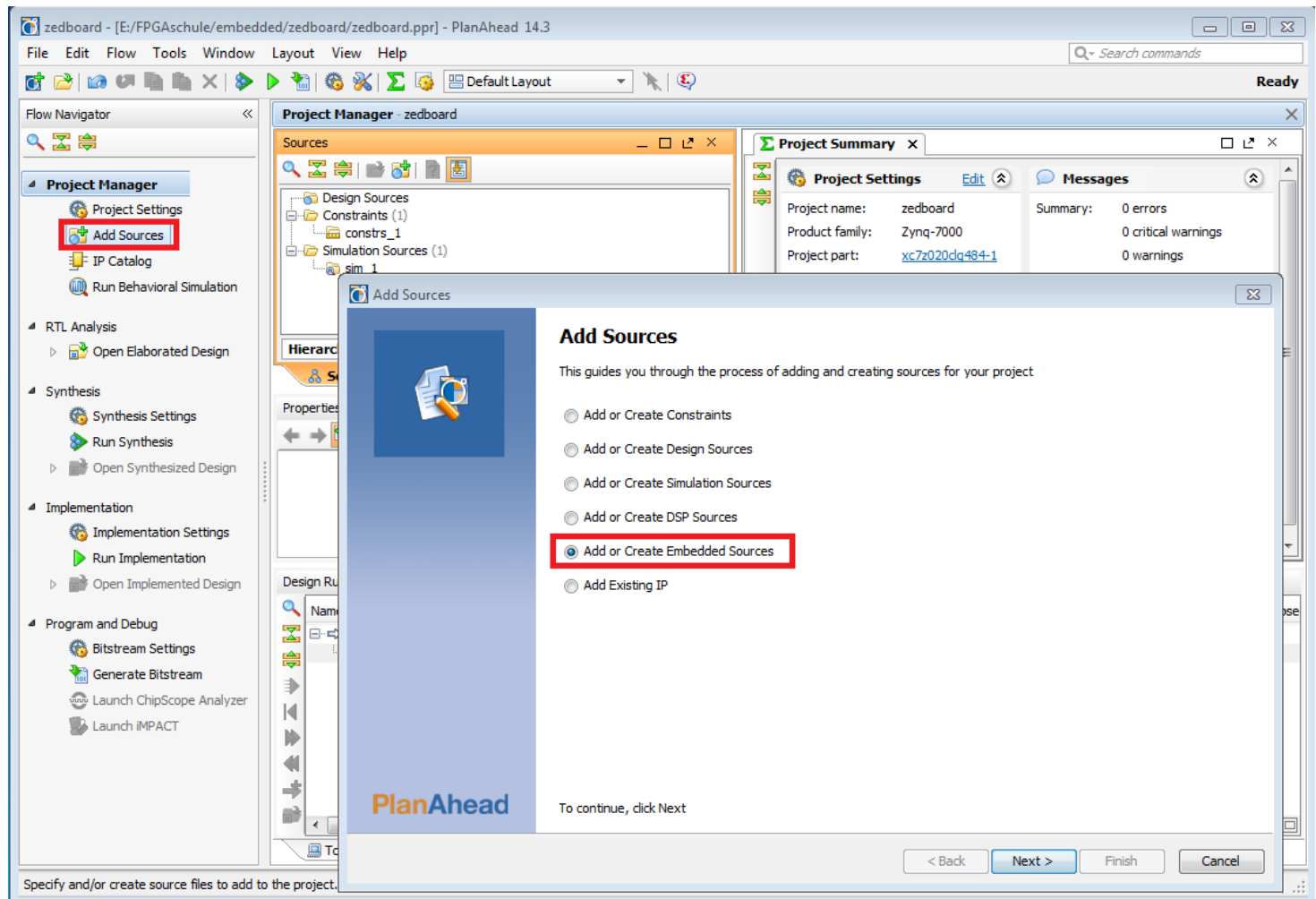
# 1.1 Creating the Hardware Platform

3. As Project Type, choose **RTL Project** and click **Do not specify sources at this time** and click Next.
4. As Default Part, switch to **Boards** and select the **ZedBoard Zynq Evaluation and Development Kit**. Click Next and Finish.



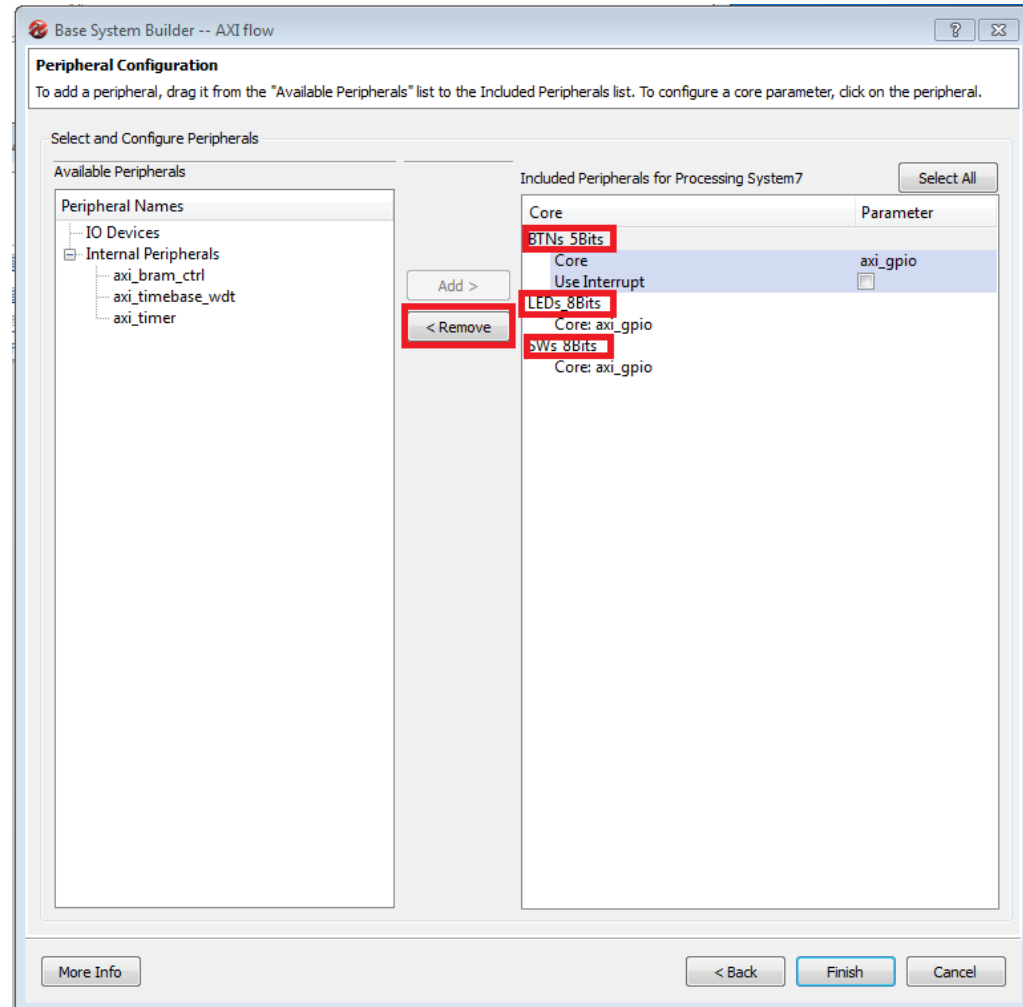
# 1.1 Creating the Hardware Platform

5. In the Flow Navigator, go to **Project Manager > Add Sources** then select **Add or Create Embedded Sources**. Click **Create Sub-Design** and name it **system**. Click Finish.



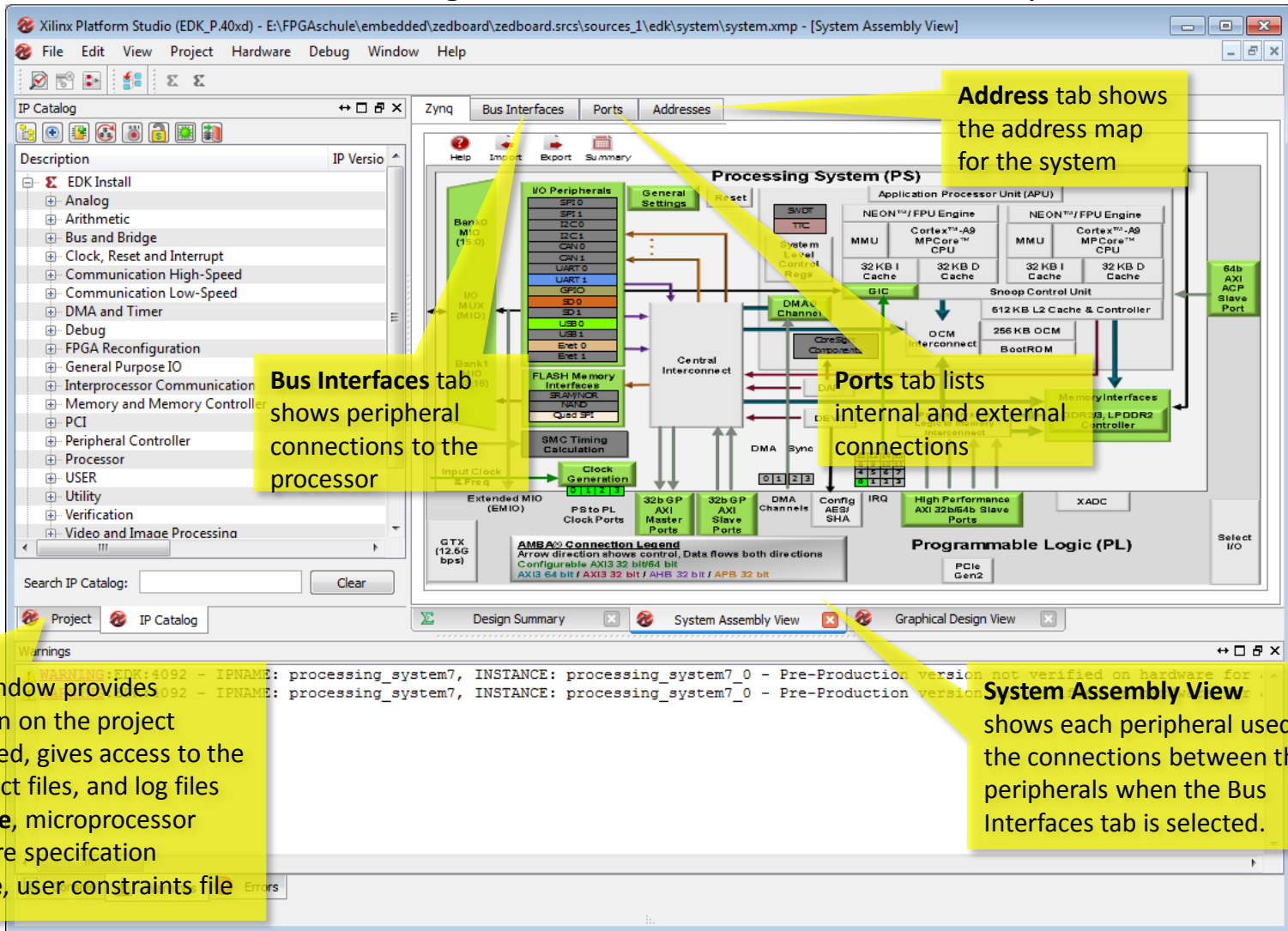
# 1.1 Creating the Hardware Platform

6. Xilinx Platform Studio opens. A message will appear asking if you want to create a **Base System** using the BSB wizard, click **Yes**.
7. Select the **AXI system** then click **OK**.
8. In the Board and System Selection Window select **ZedBoard Zynq Evaluation and Development Kit**. Click **Next**.
9. In the Peripheral Configuration, **remove all** included Peripherals by clicking the **Remove** button. Click **Finish**.



# 1.2 Understanding the System

Use the **Xilinx Platform Studio** to generate the embedded hardware platform.



## 1.2 Understanding the System

10. Select **Project → Generate Block Diagram Image** to view the block diagram for the project. The block diagram shows the connections between the different busses and components in the system. A jpeg image of the block diagram gets saved in your project in a folder named *blockdiagram*
11. A datasheet of the system can also be generated. Go to **Project → Generate and View Design Report** to view the design report. This is an html file that is generated in a *report* subfolder.
12. To view the general project options go to **Project > Project Options...** Click **Cancel** to close the window.
13. Close XPS when finished.

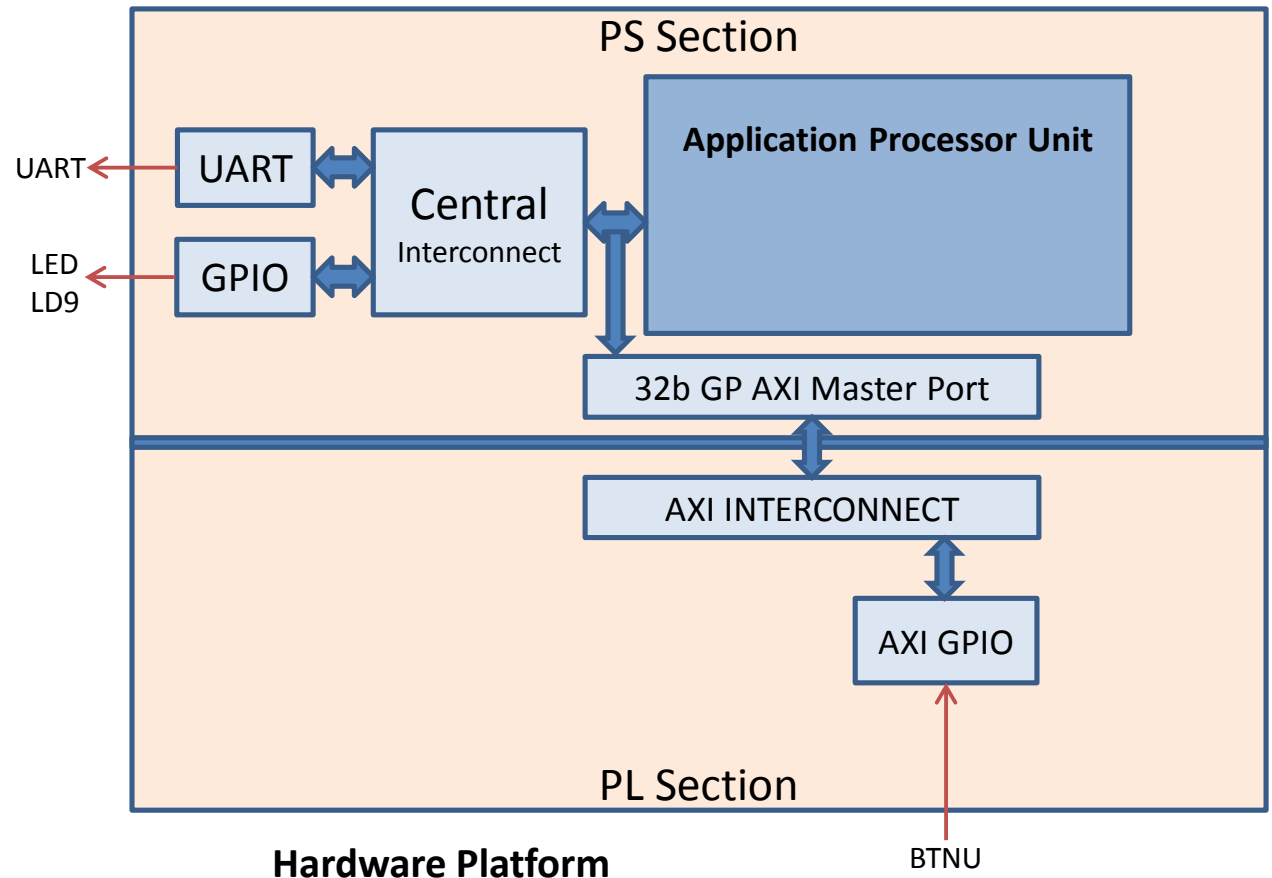
# Tutorial 2: Adding EDK IP

## Scope of the tutorial

2.1 Adding a new peripheral

2.2 Writing code for the peripheral

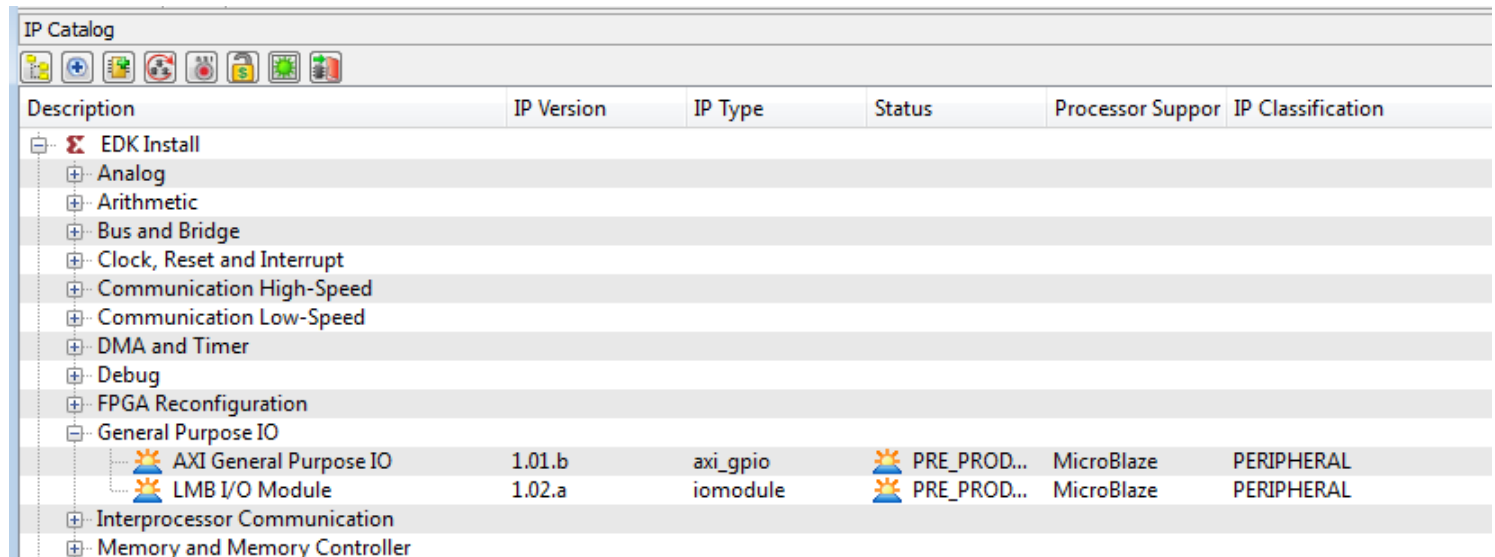
2.3 Testing the system



## 2.1 Adding a New Peripheral

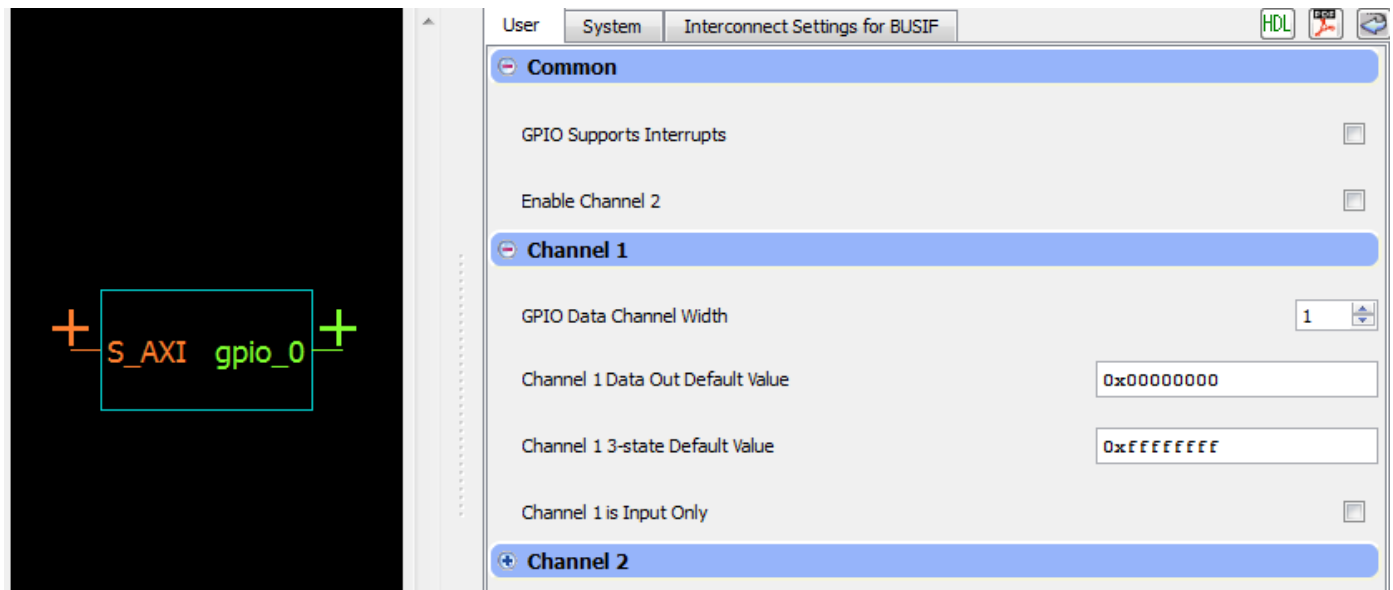
We will use Xilinx Platform Studio (XPS) to add and connect a new peripheral to the existing system.

1. In PlanAhead, double click ***system.xmp*** in the Sources Tab. This will open the system in XPS.
2. Select the **IP Catalog** tab in the project window. The IP catalog lists all the processor peripherals available with extended information. Expand the **General Purpose IO** option. The peripherals can be sorted by column field. Right click on the peripheral to view its datasheet or change log information.



## 2.1 Adding a New Peripheral

3. Right click **axi\_gpio** version 1.01.b on the list and select **Add IP**.
4. The peripheral configuration window will open to configure the peripheral. Select **Channel 1** and change the **GPIO Data Channel Width** from **32 to 1**.
5. Click **OK**.
6. When IP is added, a pop-up window will appear asking to automatically connect your new IP to the processing\_system7\_0. Click **OK**.





## 2.1 Adding a New Peripheral

- Click on the **Addresses** tab. The addresses view shows the address space for all the peripherals. The **Lock** box prevents the address for that peripheral from being changed when generating new addresses.

Generate Address Button

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Lock	Bus Name
processing_system7_0's Address Map							
processing_system7_0	C_DDR_RAM_B...	0x00000000	0x1FFFFFFF	512M		<input checked="" type="checkbox"/>	
axi_gpio_0	C_BASEADDR	0x41200000	0x4120FFFF	64K	<input checked="" type="checkbox"/> S_AXI	<input type="checkbox"/>	axi_interconnect_1
processing_system7_0	C_UART1_BASE...	0xE0001000	0xE0001FFF	4K		<input checked="" type="checkbox"/>	
processing_system7_0	C_GPIO_BASEA...	0xE000A000	0xE000AFFF	4K		<input checked="" type="checkbox"/>	
processing_system7_0	C_ENET0_BASE...	0xE000B000	0xE000BFFF	4K		<input checked="" type="checkbox"/>	
processing_system7_0	C_SDIO0_BASE...	0xE0100000	0xE0100FFF	4K		<input checked="" type="checkbox"/>	
processing_system7_0	C_USB0_BASEA...	0xE0102000	0xE0102FFF	4K		<input checked="" type="checkbox"/>	
processing_system7_0		0xE0104000	0xE0104FFF	4K		<input checked="" type="checkbox"/>	

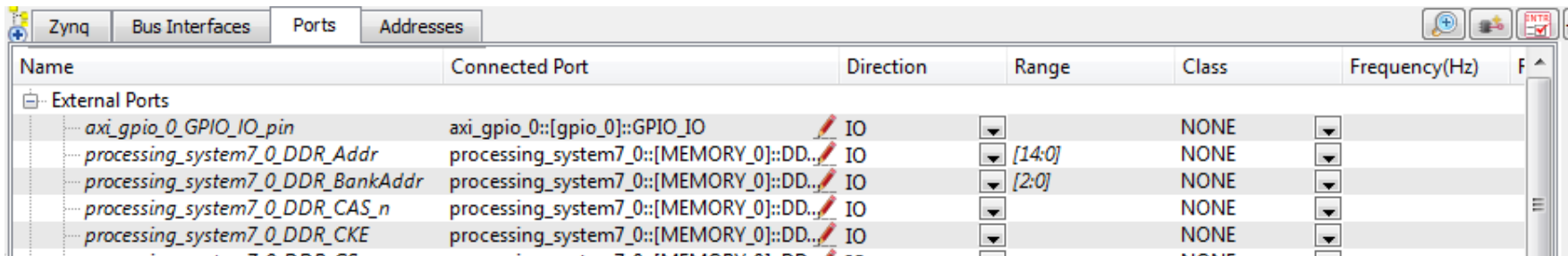
Alternatively: re-generate non-overlapping addresses for all peripherals

Change generated address parameters if necessary

- Click on the **Ports** tab. The Ports view shows the internal connections between the peripherals as well as the external ports connections
- Expand **axi\_gpio\_0** from the list. It will show the connections available for the peripheral.
- Expand the **(IO\_IF) gpio\_0** selection. Look at the GPIO datasheet for a description of each port. The datasheet can be found by right-clicking on **axi\_gpio\_0**.

## 2.1 Adding a New Peripheral

11. Expand the **External Ports** to view the new connection. The name of the new external port is **axi\_gpio\_0\_GPIO\_IO\_pin**.



Name	Connected Port	Direction	Range	Class	Frequency(Hz)
External Ports					
axi_gpio_0_GPIO_IO_pin	axi_gpio_0::[gpio_0]::GPIO_IO	IO		NONE	
processing_system7_0_DDR_Addr	processing_system7_0::[MEMORY_0]::DD...	IO	[14:0]	NONE	
processing_system7_0_DDR_BankAddr	processing_system7_0::[MEMORY_0]::DD...	IO	[2:0]	NONE	
processing_system7_0_DDR_CAS_n	processing_system7_0::[MEMORY_0]::DD...	IO		NONE	
processing_system7_0_DDR_CKE	processing_system7_0::[MEMORY_0]::DD...	IO		NONE	

We need to update the design information for SDK. Since we are managing this embedded project from within PlanAhead, it will create the XML file and export the design to SDK.

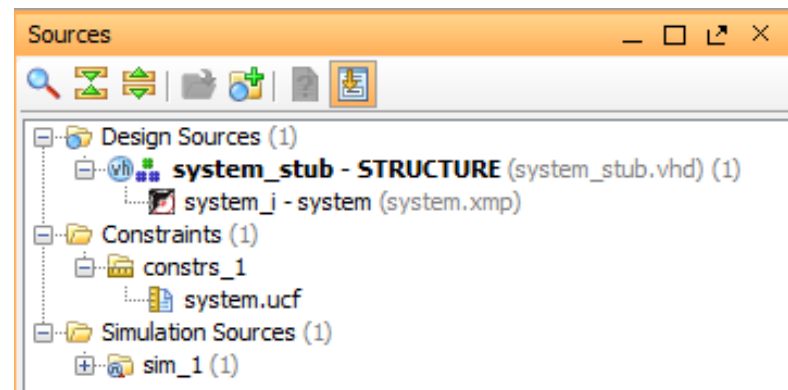
12. Close **XPS**.

## 2.1 Adding a New Peripheral

Since we've added a new port to our system, it needs to be added to the HDL source. Additionally, we'll need to update the constraint file to add the pinout information for the button.

13. In Sources Tab of the Project Manager, right click **system.xmp** and select **Create Top HDL**. The PlanAhead tool generates the `system_stub.v` top-level module for this design.
14. Double Click **system\_stub.v** to take a look at it. You can see that the new **axi\_gpio\_0\_GPIO\_IO\_pin** ports have been added automatically.
15. In the **Flow Navigator**, click **Add Sources** and **Add or Create Constraints**.
16. Click Create File and name it **system**. Click Finish.

17. Double click the new `system.ucf` file to edit it.



## 2.1 Adding a New Peripheral

18. In the UCF file add the line:

- **NET axi\_gpio\_0\_GPIO\_IO\_pin IOSTANDARD=LVCMOS25 | LOC=T18;**

This will connect the AXI GPIO pin to the T18 Pin of the PL section (Push Button “BTNU”)

19. Save and close the **UCF file**.

20. In the Flow Navigator, click ***Generate Bitstream***. Ignore any critical warnings that appear.

21. After it has finished, click ***File > Export > Export Hardware for SDK***.

22. Check the ***Launch SDK*** check box.

## 2.2 Writing Code for the New Peripheral

To test the new peripheral we will create a new software application in **Platform Studio SDK** and use the GPIO device drivers.

1. The peripheral datasheets and address map can be found under the hardware platform. Expand the **system\_hw\_platform** project and double-click on the **system.xml** file.
2. Open the **axi\_gpio** datasheet to view the GPIO register map. The **GPIO\_Data** Register is located at the base address of the peripheral, which is 0x41200000 for the Push Button.

*Table 4: Registers*

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
C_BASEADDR + 0x00	GPIO_DATA	Read/Write	0x0	Channel 1 AXI GPIO Data Register.
C_BASEADDR + 0x04	GPIO_TRI	Read/Write	0x0	Channel 1 AXI GPIO Three-state Register.
C_BASEADDR + 0x08	GPIO2_DATA	Read/Write	0x0	Channel 2 AXI GPIO Data Register.
C_BASEADDR + 0x0C	GPIO2_TRI	Read/Write	0x0	Channel 2 AXI GPIO Three-state Register.

## 2.2 Writing Code for the New Peripheral

3. Go to **File > New > Application Project**.
4. Name the project **Led\_app** and make sure that a new Board Support Package will be created. Click **Next**.
5. Select **Empty Application** from the project templates. Click **Finish**.

Project name:

☒ Use default location

Location:

Choose file system:

Target Hardware

Hardware Platform:

Processor:

Target Software

OS Platform:

Language: ☒ C ☐ C++

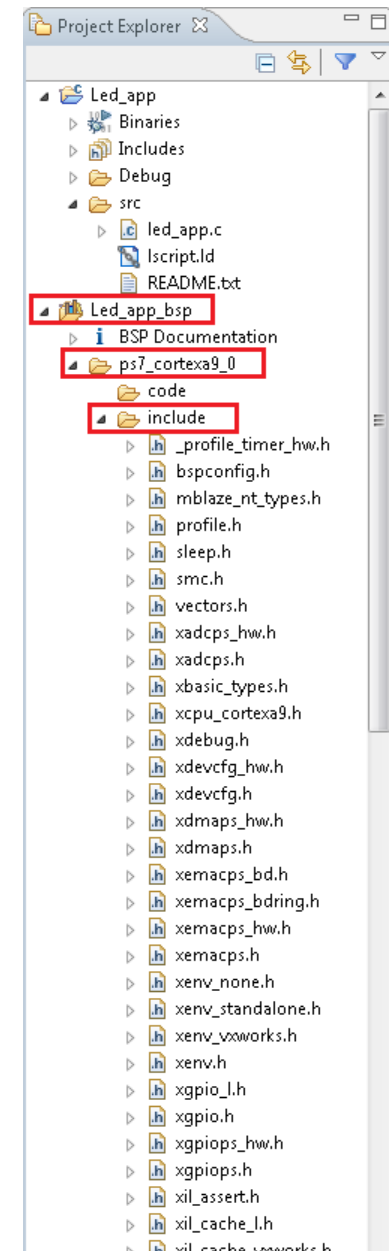
Board Support Package: ☒ Create New  ☐ Use existing

## 2.2 Writing Code for the New Peripheral

We will add code after the print statement to turn-on the LED when the Button is pressed.

6. On the left side expand the **Led\_app\_bsp** project. The **BSP Documentation** section contains the documentation for the device drivers. The **ps7\_cortexa9\_0** folder contains the header files, compiled libraries, and sources for the Board Support Package.
7. Expand **ps7\_cortexa9\_0** then expand the **include** directory. Double click on the **xparameters.h** file to view the hardware parameters for the system. Using the macros will isolate the software from the actual hardware.

```
/* ***** */  
  
/* Definitions for driver GPIO */  
#define XPAR_XGPIO_NUM_INSTANCES 1  
  
/* Definitions for peripheral AXI_GPIO_0 */  
#define XPAR_AXI_GPIO_0_BASEADDR 0x41200000  
#define XPAR_AXI_GPIO_0_HIGHADDR 0x4120FFFF  
#define XPAR_AXI_GPIO_0_DEVICE_ID 0  
#define XPAR_AXI_GPIO_0_INTERRUPT_PRESENT 0  
#define XPAR_AXI_GPIO_0_IS_DUAL 0
```



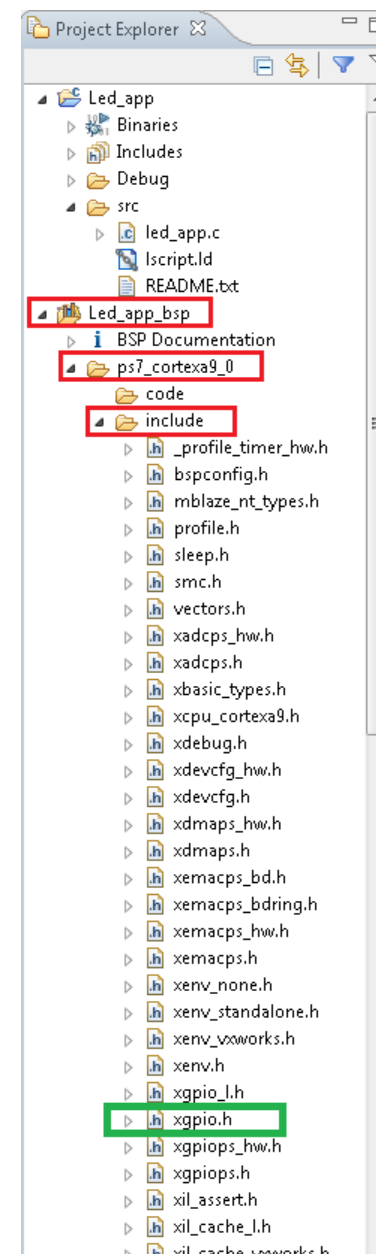
## 2.2 Writing Code for the New Peripheral

8. Inside the expanded **include** directory for **ps7\_cortexa9\_0** are all the driver header files for the different peripherals. Double-click on **xgpio.h** to view the GPIO functions.
9. Click on **XGpio\_DiscreteRead** to view the format to read the GPIO registers.

```
/* ***** Function Prototypes ***** */

/*
 * Initialization functions in xgpio_sinit.c
 */
int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
XGpio_Config *XGpio_LookupConfig(u16 DeviceId);

/*
 * API Basic functions implemented in xgpio.c
 */
int XGpio_CfgInitialize(XGpio *InstancePtr, XGpio_Config * Config,
                        u32 EffectiveAddr);
void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel,
                            u32 DirectionMask);
u32 XGpio_GetDataDirection(XGpio *InstancePtr, unsigned Channel);
u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
void XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Mask);
```

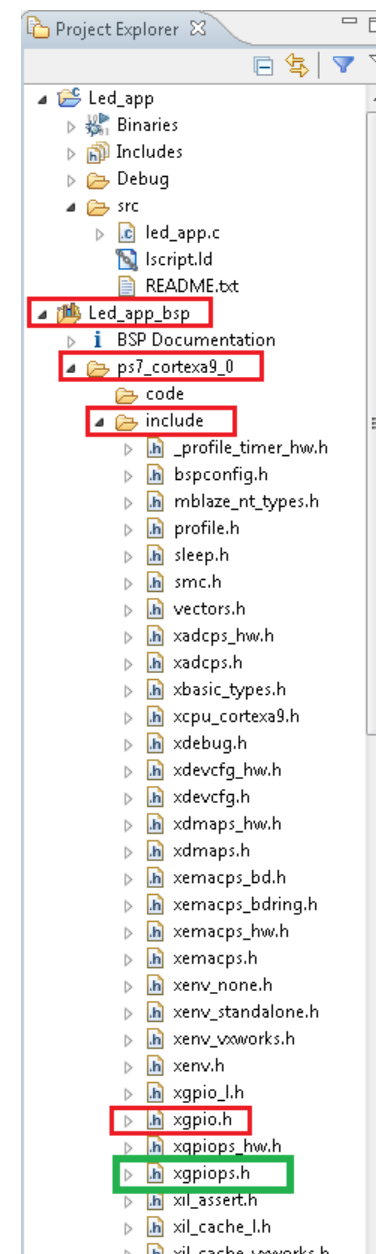




## 2.2 Writing Code for the New Peripheral

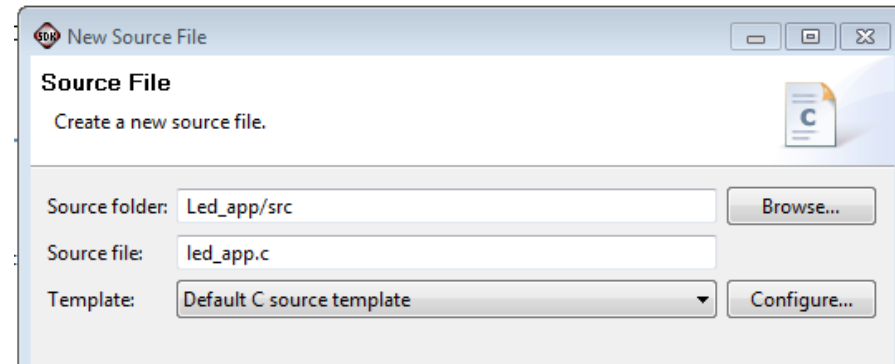
10. Also double click on **xgpiops.h**. We will also use the function **XGpioPs\_WritePin** to write to the LED LD9 which is automatically connected to the **MIO pin 7** in the **PS**. The Base Address for the device can be found in the **xparameters.h** file. The Data Register has an offset of 0x00.

```
/*  
 * Pin APIs in xgpiops.c  
 */  
int XGpioPs_ReadPin(XGpioPs *InstancePtr, int Pin);  
void XGpioPs_WritePin(XGpioPs *InstancePtr, int Pin, int Data);  
void XGpioPs_SetDirectionPin(XGpioPs *InstancePtr, int Pin, int Direction);  
int XGpioPs_GetDirectionPin(XGpioPs *InstancePtr, int Pin);  
void XGpioPs_SetOutputEnablePin(XGpioPs *InstancePtr, int Pin, int Enable);  
int XGpioPs_GetOutputEnablePin(XGpioPs *InstancePtr, int Pin);
```



## 2.2 Writing Code for the New Peripheral

11. We need to add a source file for the new empty C project. Select the ***Led\_app\src*** folder and go to **File > New > Source File**. Enter ***led\_app.c*** for the file name. Click **Finish**.



12. **Copy** the prepared contents into the ***led\_app.c*** file.

13. Save the ***led\_app.c*** file. The application will be compiled when saved. The **Project** menu gives options to change the behavior for building the application.
14. Look through the C-code and make sure you recognize the **XGpio\_DiscreteRead** and **XGpioPs\_WritePin** functions to read the Push button and write to the LED.

## 2.2 Writing Code for the New Peripheral

### Standalone\_BSP

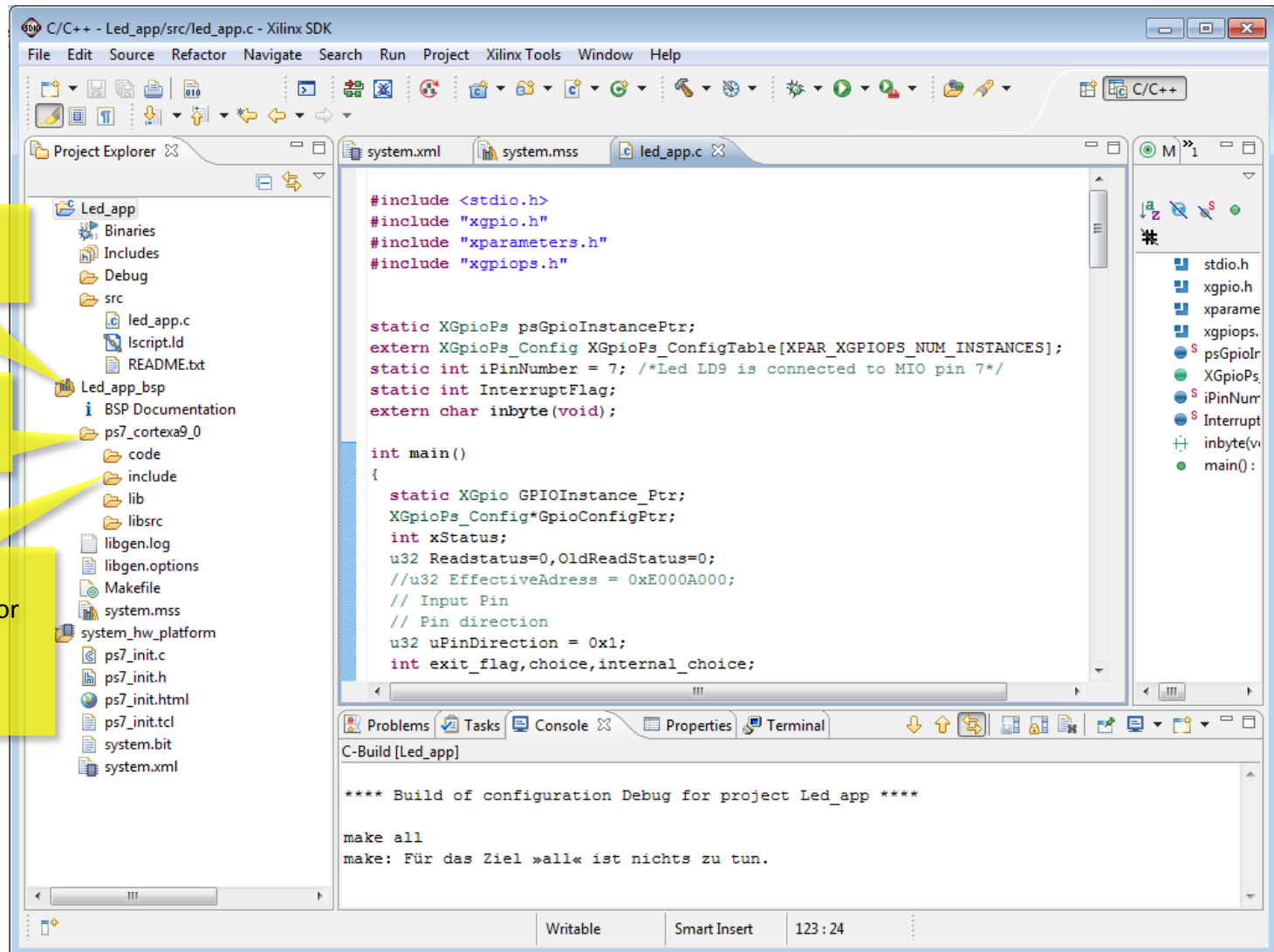
Board support package  
documentation

### ps7\_cortexa9\_0

header files, compiled  
libraries, BSP sources


### ps7\_cortexa9\_0 -> include

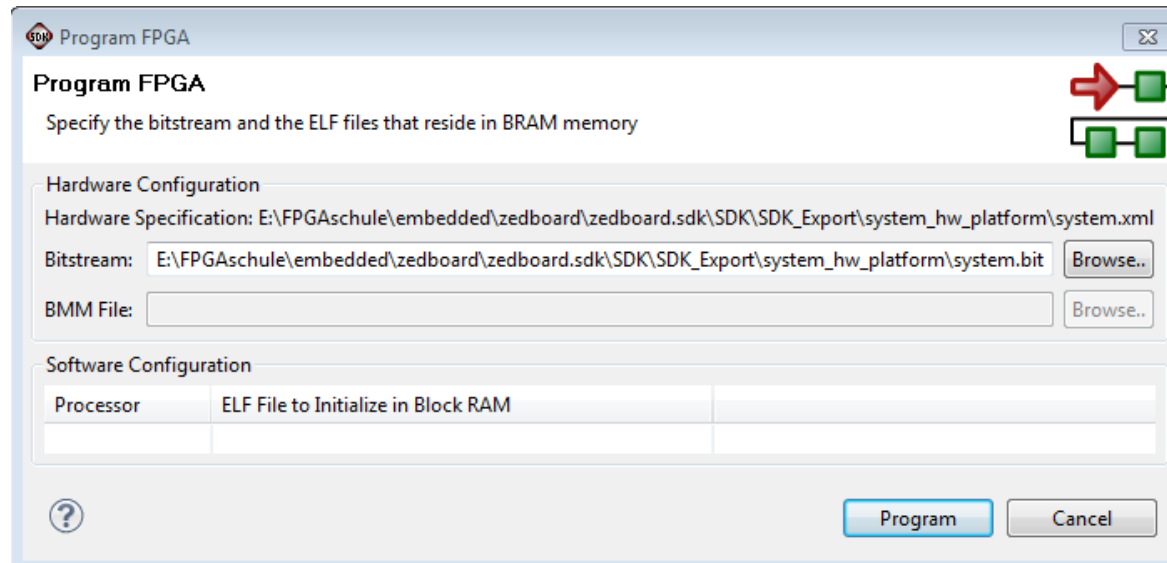
- *xparameters.h*: macros for HW / SW isolation
- *xgpio.h*: (driver files \*.h)
- *xgpiops.h*




## 2.3 Testing the Generated System

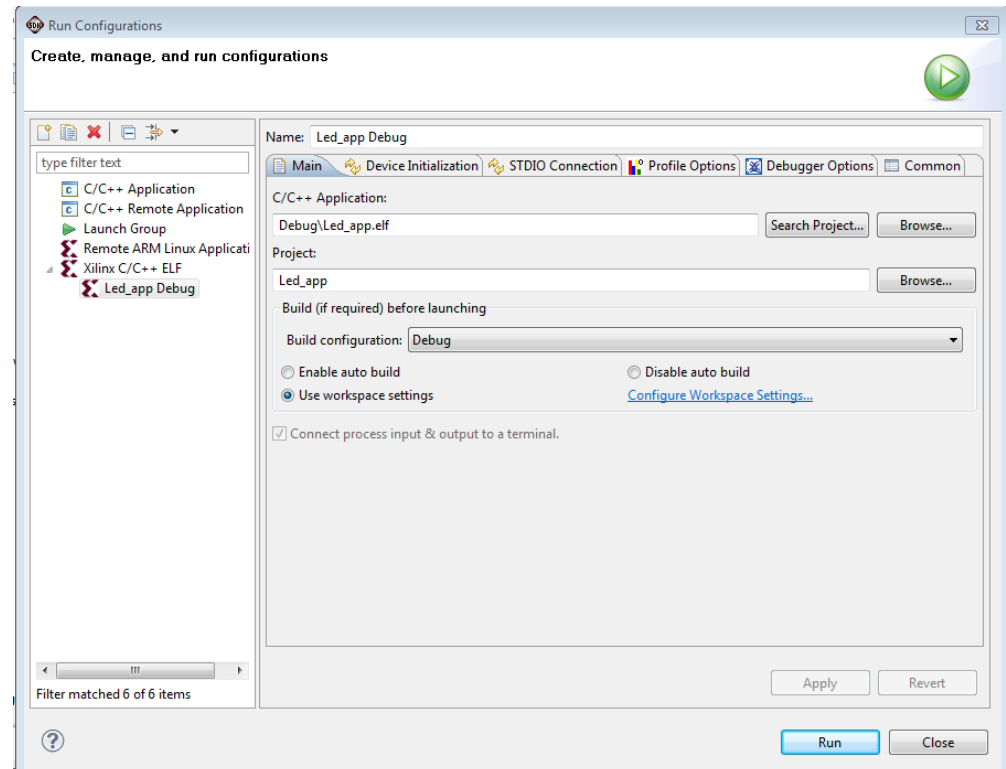
At first, the hardware configuration is downloaded, and the processor is reset to a state which allows the download of an application.

1. In **SDK**, click on the **Program FPGA** icon 
  - For the Bitstream, make sure it is the right one named **system.bit**.
  - Click on **Program**.



## 2.3 Testing the Generated System

2. In the SDK Project Explorer View, right-click on the **Led\_app** project and select **Run As > Run Configurations...**
3. Select **Xilinx C/C++ ELF** and click on the **New Launch Configuration** icon 
4. In the SDK **Run Configurations** window, don't make any changes.
5. Click **Run**.



## 2.3 Testing the Generated System

6. Connect with **putty** to the ZedBoard to see the output.
7. When finished **Stop/Terminate** the application by pressing the red Stop/Terminate button in the Console window.
8. Close **SDK**. This is the end of this Tutorial.

```
COM4 - PuTTY
#### Application Starts ####

SELECT the Operation from the Below Menu
##### Menu Starts #####
Press '1' to switch LED on
Press '2' to switch LED off
Press any other key to Exit
##### Menu Ends #####
Selection : 1
Press Switch 'BTNU' push button on board

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
BTNU PUSH Button pressed
LED 'LD9' Turned ON

Returning to Main Menu.....

SELECT the Operation from the Below Menu
##### Menu Starts #####
Press '1' to switch LED on
Press '2' to switch LED off
Press any other key to Exit
##### Menu Ends #####
Selection : 2
Press Switch 'BTNU' push button on board

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
BTNU PUSH Button pressed
LED 'LD9' Turned OFF

Returning to Main Menu.....

SELECT the Operation from the Below Menu
##### Menu Starts #####
Press '1' to switch LED on
Press '2' to switch LED off
Press any other key to Exit
##### Menu Ends #####
Selection : 3

*****
LED turned off
BYE
*****
█
```

# And now for the experts

- Try to get that running on top of Linux ...