Introduction

Install and activate LPCXpresso IDE

Our development board is an LPCXpresso 1549 from NXP and we use LPCXpresso development tools for development and debugging. LPCXpresso features an Eclipse based IDE with gcc toolchain and support for integrated debug probe found on the development board.

Go to https://www.lpcware.com/lpcxpresso and download and install LPCXpresso v8.2.2 (or newer if available).You need to activate the IDE to enable all of its features. During installation allow all suggested NXP drivers to be installed.

See LPCXpresso installation guide chapter 3 for instructions on how to activate the IDE. You can find it in the installation directory after you have installed LPCXpresso … great job NXP!

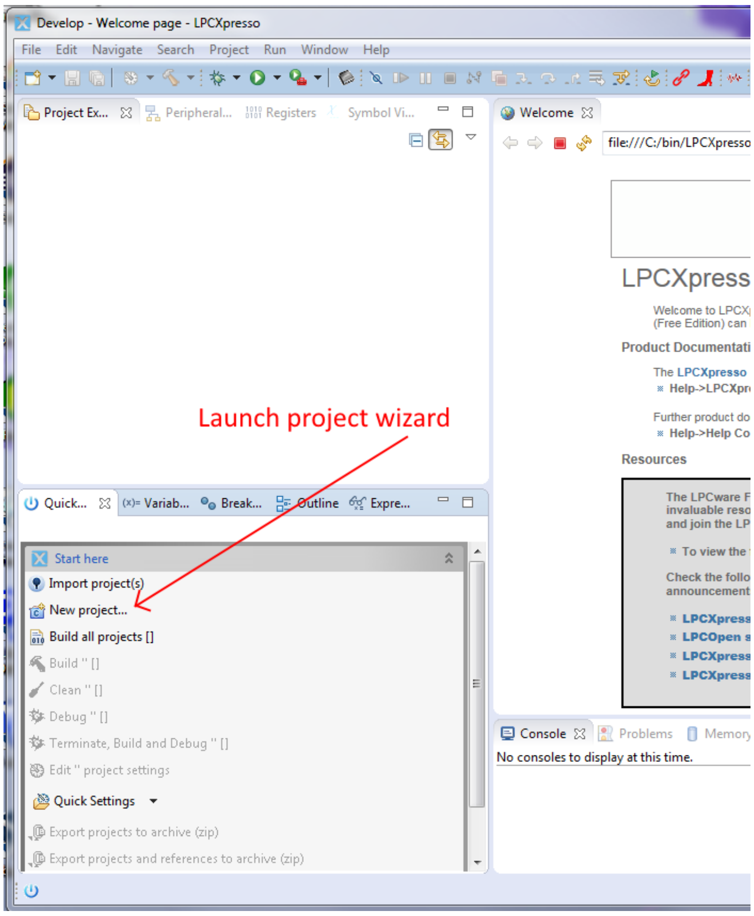
Go <installation directory>\lpcxpresso\Examples\LPCOpen and locate the latest LPCOpen 2.xx software package for LPCXpresso LPC1549 board. As of this writing the most recent version is 2.20. (The file is lpcopen_2_20_lpcxpresso_nxp_lpcxpresso_1549.zip)

Create a new workspace and import driver libraries

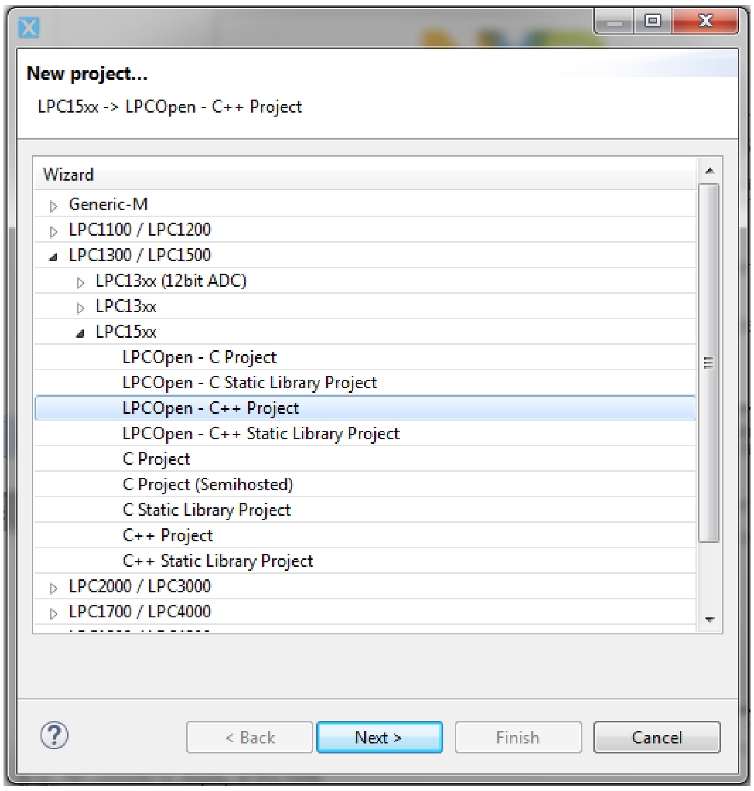
The IDE installation includes LPCOpen software platform that contains peripheral drivers and sample code. Each processor family requires different drivers and it is up to the developer to select and import driver packages to be used.

When you start the IDE you will be prompted to select a workspace. If you select a folder that doesn’t have a workspace an empty workspace is created in the directory. A workspace typically contains multiple projects: peripheral driver library projects and your own project(s) that make a reference to (i.e. use) the driver projects.

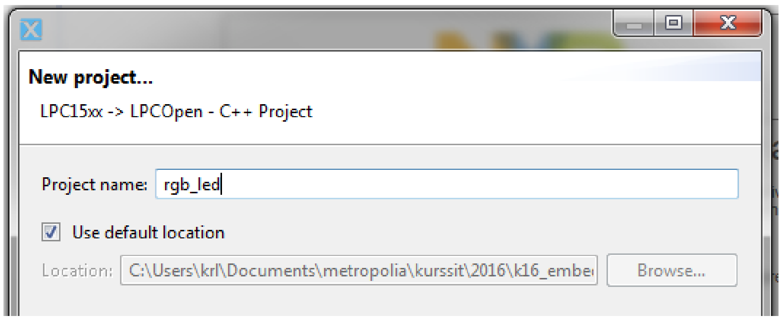
The easiest way to import drivers and to create a new project is to use the project wizard. The project wizard is launched from the quick start window. See screen shot below. The wizard will ask you to import driver libraries when you create the project. However first you need to select processor and project type.


Develop - Welcome page - LPCXpresso
File Edit Navigate Search Project Run Window Help
Project Ex... Peripheral... Registers Symbol Vi...
Welcome
file:///C:/bin/LPCXpresso
LPCXpress
Launch project wizard
Quick... Variab... Break... Outline Expre...
Start here
Import project(s)
New project...
Build all projects []
Build '' []
Clean '' []
Debug '' []
Terminate, Build and Debug '' []
Edit '' project settings
Quick Settings
Export projects to archive (zip)
Export projects and references to archive (zip)
Console Problems Memory
No consoles to display at this time.

Our development board is based on LPC1549 processor so we select LPC1500-series from the menu. Expand categories and select LPCOpen – C++ Project under LPC15xx.



When you press next you will be asked to enter a name for your project.



New project...

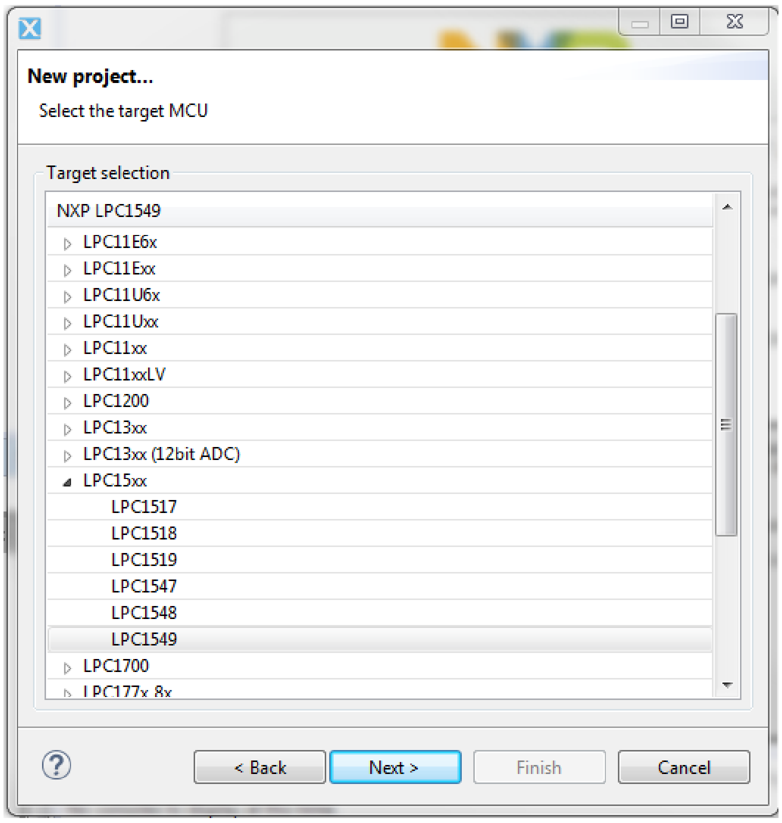
LPC15xx -> LPCOpen - C++ Project

Project name: rgb_led

☑ Use default location

Location: C:\Users\krl\Documents\metropolia\kurssit\2016\k16_embe

After naming the project you are prompted to select the target MCU. Expand LPC15xx and select LPC1549.



New project...

Select the target MCU

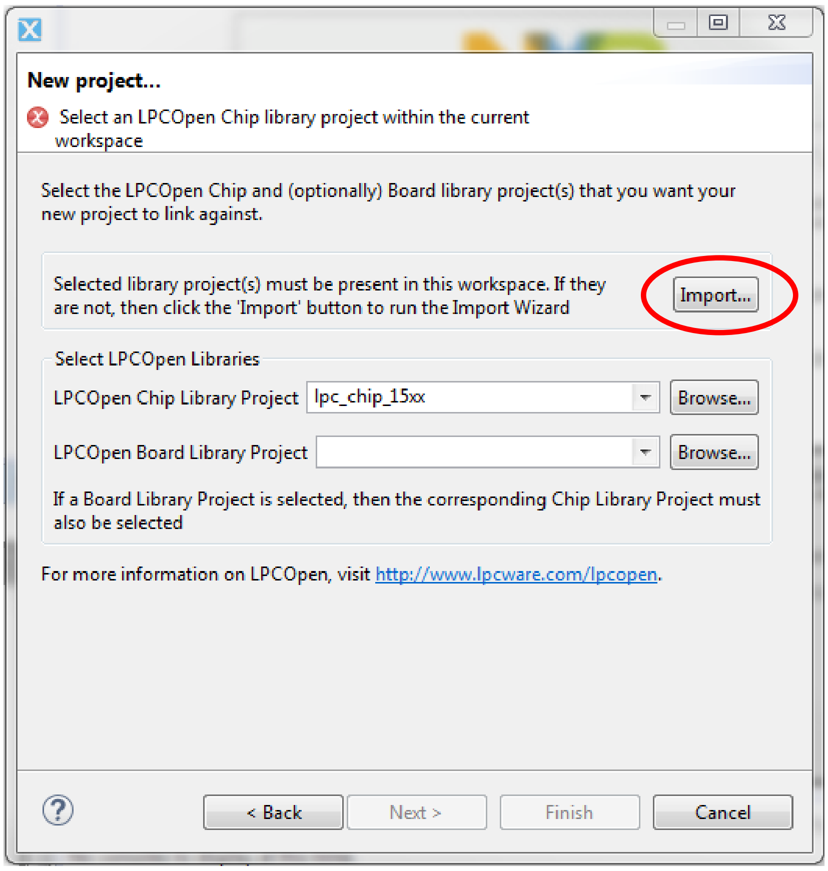
Target selection

NXP LPC1549

- LPC11E6x
- LPC11Exx
- LPC11U6x
- LPC11Uxx
- LPC11xx
- LPC11xxLV
- LPC1200
- LPC13xx
- LPC13xx (12bit ADC)
- LPC15xx
  - LPC1517
  - LPC1518
  - LPC1519
  - LPC1547
  - LPC1548
  - LPC1549
- LPC1700
- LPC177x_8x

Then you have the option of importing the driver libraries. **If you created a new workspace then you must import the libraries.**

Click Import.



New project...

Select an LPCOpen Chip library project within the current workspace

Select the LPCOpen Chip and (optionally) Board library project(s) that you want your new project to link against.

Selected library project(s) must be present in this workspace. If they are not, then click the 'Import' button to run the Import Wizard

Select LPCOpen Libraries

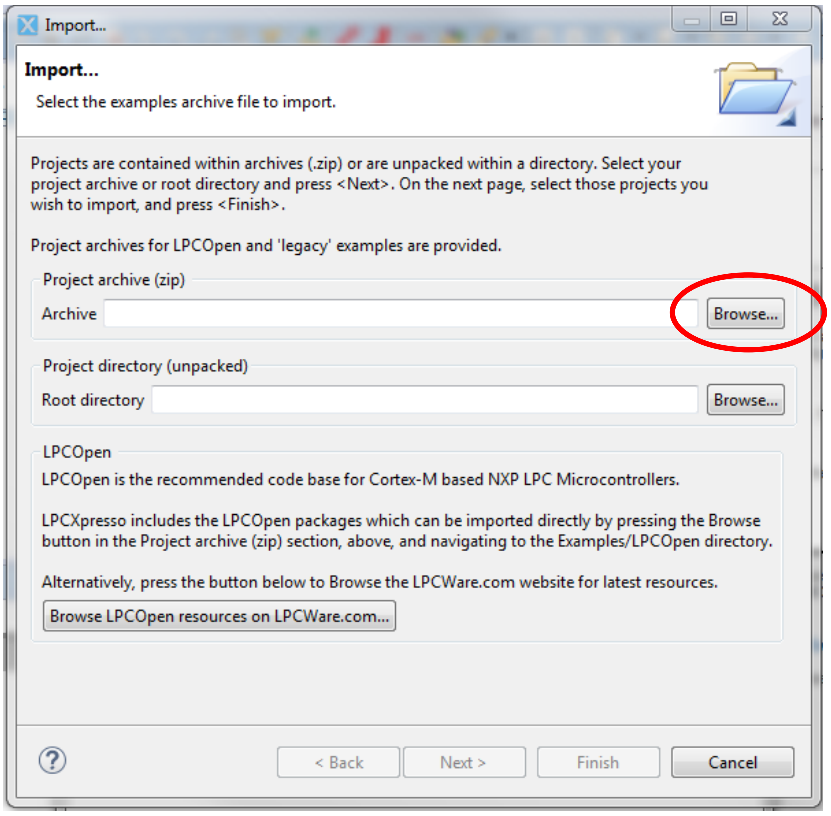
LPCOpen Chip Library Project: lpc_chip_15xx

LPCOpen Board Library Project:

If a Board Library Project is selected, then the corresponding Chip Library Project must also be selected

For more information on LPCOpen, visit http://www.lpcware.com/lpcopen.

In the import wizard click Browse on Project Archive.


Import...

Import...

Select the examples archive file to import.

Projects are contained within archives (.zip) or are unpacked within a directory. Select your project archive or root directory and press <Next>. On the next page, select those projects you wish to import, and press <Finish>.

Project archives for LPCOpen and 'legacy' examples are provided.

Project archive (zip)

Archive [ ] Browse...

Project directory (unpacked)

Root directory [ ] Browse...

LPCOpen

LPCOpen is the recommended code base for Cortex-M based NXP LPC Microcontrollers.

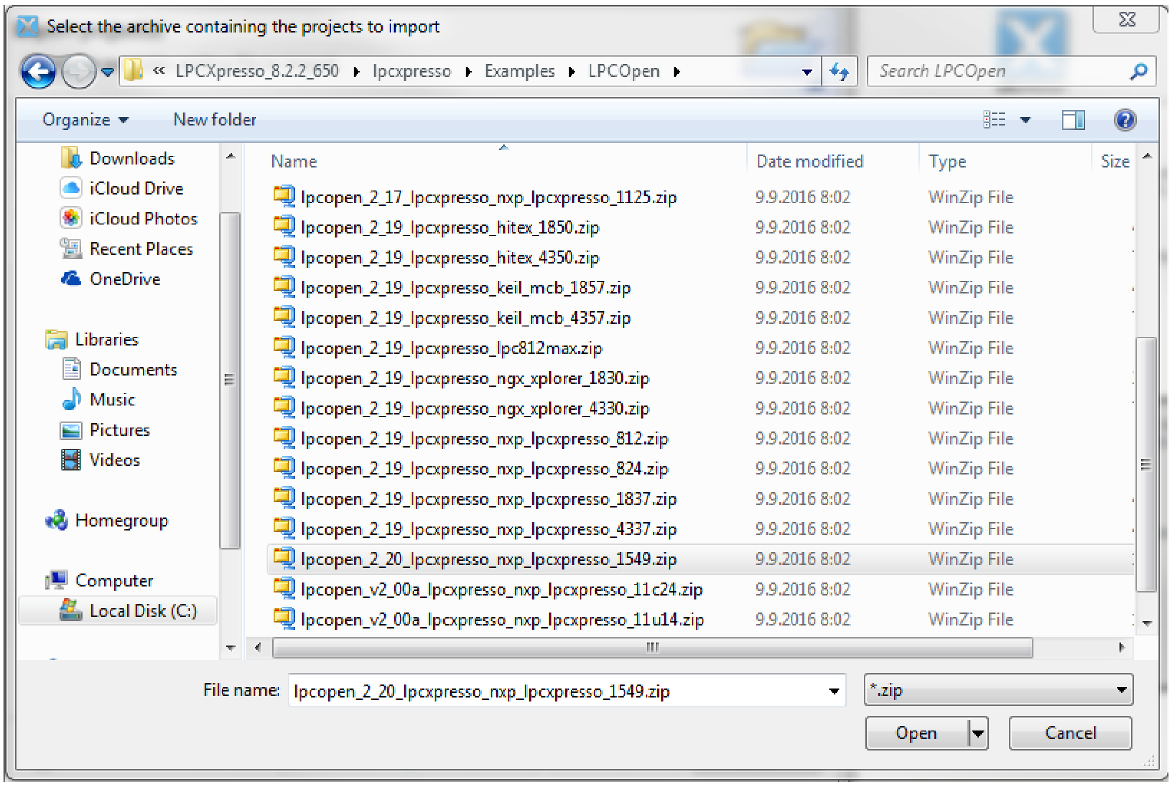
LPCXpresso includes the LPCOpen packages which can be imported directly by pressing the Browse button in the Project archive (zip) section, above, and navigating to the Examples/LPCOpen directory.

Alternatively, press the button below to Browse the LPCWare.com website for latest resources.

Browse LPCOpen resources on LPCWare.com...

< Back | Next > | Finish | Cancel

Then navigate to the directory where the LPCOpen software package is and select **lpcopen_2_20_lpcxpresso_nxp_lpcxpresso_1549.zip** and click Open.


Select the archive containing the projects to import

« LPCXpresso_8.2.2_650 ▸ lpcxpresso ▸ Examples ▸ LPCOpen ▸

Name	Date modified	Type	Size
lpcopen_2_17_lpcxpresso_nxp_lpcxpresso_1125.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_hitex_1850.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_hitex_4350.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_keil_mcb_1857.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_keil_mcb_4357.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_lpc812max.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_ngx_xplorer_1830.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_ngx_xplorer_4330.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_nxp_lpcxpresso_812.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_nxp_lpcxpresso_824.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_nxp_lpcxpresso_1837.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_19_lpcxpresso_nxp_lpcxpresso_4337.zip	9.9.2016 8:02	WinZip File	
lpcopen_2_20_lpcxpresso_nxp_lpcxpresso_1549.zip	9.9.2016 8:02	WinZip File	
lpcopen_v2_00a_lpcxpresso_nxp_lpcxpresso_11c24.zip	9.9.2016 8:02	WinZip File	
lpcopen_v2_00a_lpcxpresso_nxp_lpcxpresso_11u14.zip	9.9.2016 8:02	WinZip File	

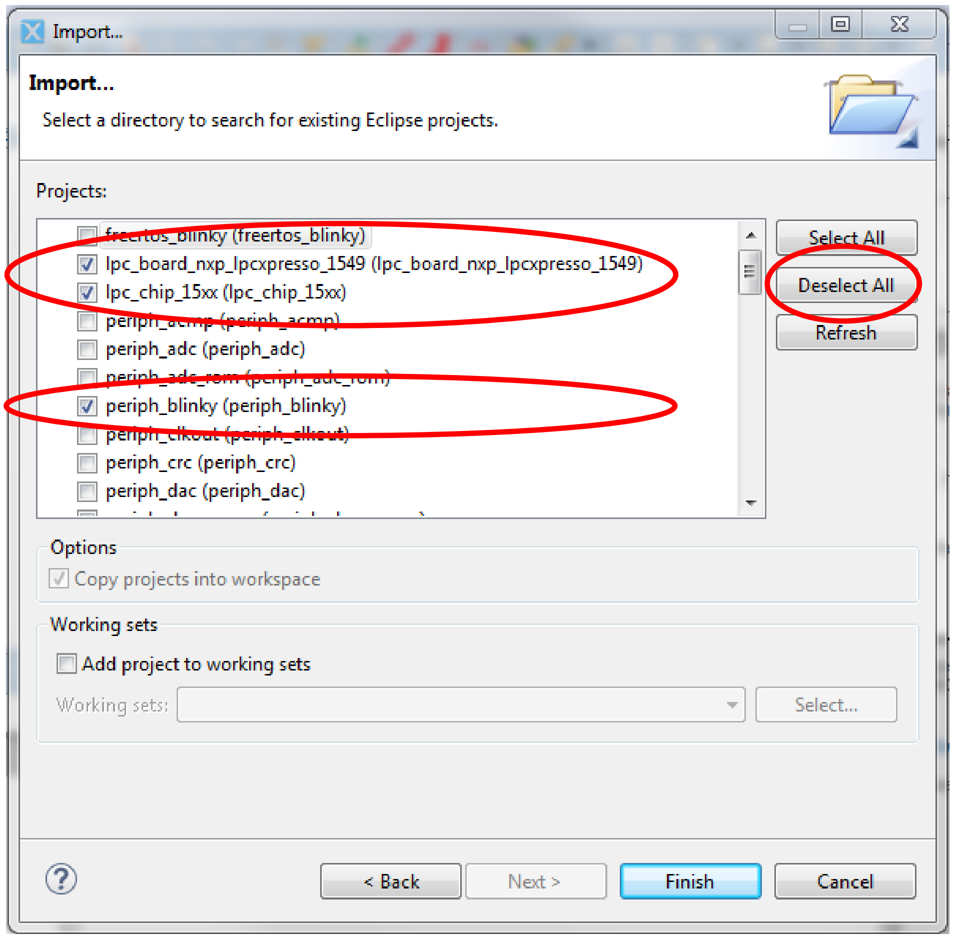
File name: lpcopen_2_20_lpcxpresso_nxp_lpcxpresso_1549.zip

*.zip

Open | Cancel

When you are back in the import wizard click Next.

Then the wizard asks you to select projects to import. Click **Deselect A**ll and then select the following three projects:



Import...

Import...

Select a directory to search for existing Eclipse projects.

Projects:

- [ ] freertos_blinky (freertos_blinky)
- [✓] lpc_board_nxp_lpcxpresso_1549 (lpc_board_nxp_lpcxpresso_1549)
- [✓] lpc_chip_15xx (lpc_chip_15xx)
- [ ] periph_acmp (periph_acmp)
- [ ] periph_adc (periph_adc)
- [ ] periph_adc_rom (periph_adc_rom)
- [✓] periph_blinky (periph_blinky)
- [ ] periph_clkout (periph_clkout)
- [ ] periph_crc (periph_crc)
- [ ] periph_dac (periph_dac)

Options

[✓] Copy projects into workspace

Working sets

[ ] Add project to working sets

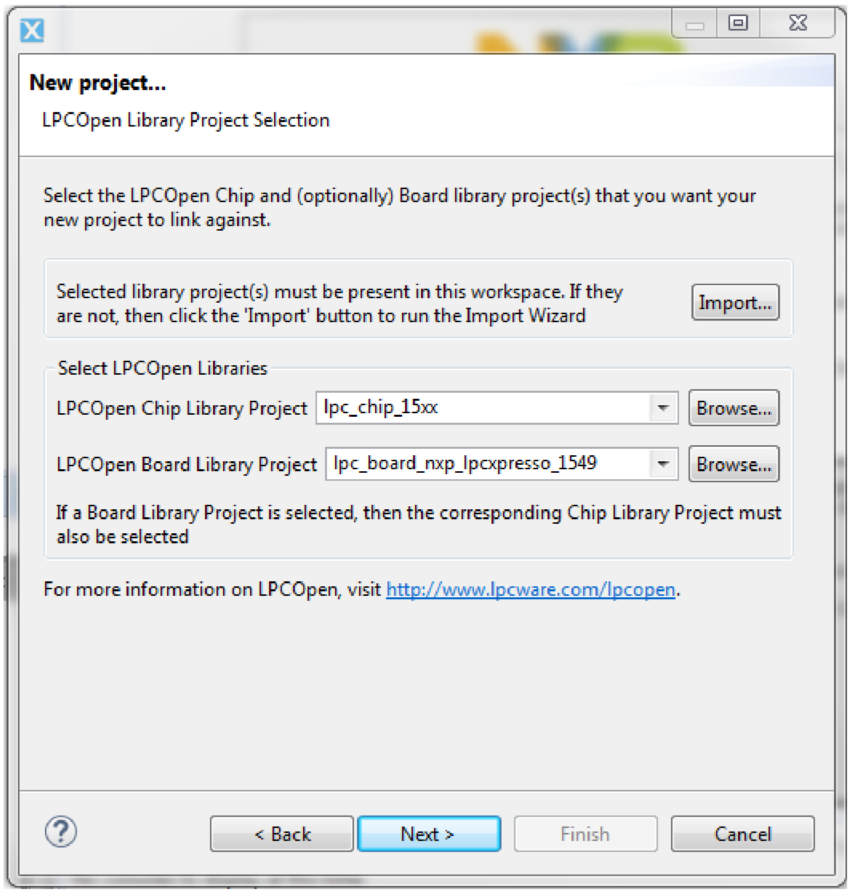
Working sets:

Press Finish.

Note: The third project (periph_blinky) is imported only for the sake of testing the debugger. Later when you create a new workspace you don’t need to import this project.

No, you’re not done yet.

You are taken back to the project creation wizard where you need to select which LPCOpen libraries to use. Just select the only available library in both of the drop down menus and press next.


New project...

LPCOpen Library Project Selection

Select the LPCOpen Chip and (optionally) Board library project(s) that you want your new project to link against.

Selected library project(s) must be present in this workspace. If they are not, then click the 'Import' button to run the Import Wizard

Select LPCOpen Libraries

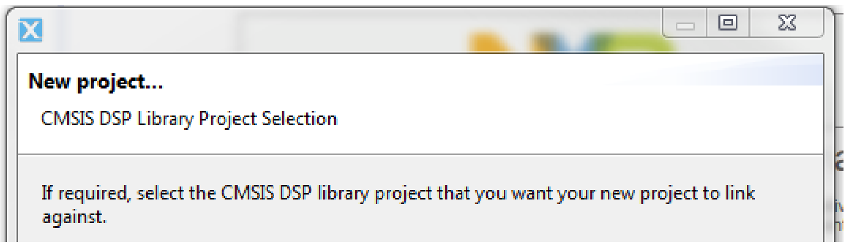
LPCOpen Chip Library Project: lpc_chip_15xx

LPCOpen Board Library Project: lpc_board_nxp_lpcxpresso_1549

If a Board Library Project is selected, then the corresponding Chip Library Project must also be selected

For more information on LPCOpen, visit http://www.lpcware.com/lpcopen.

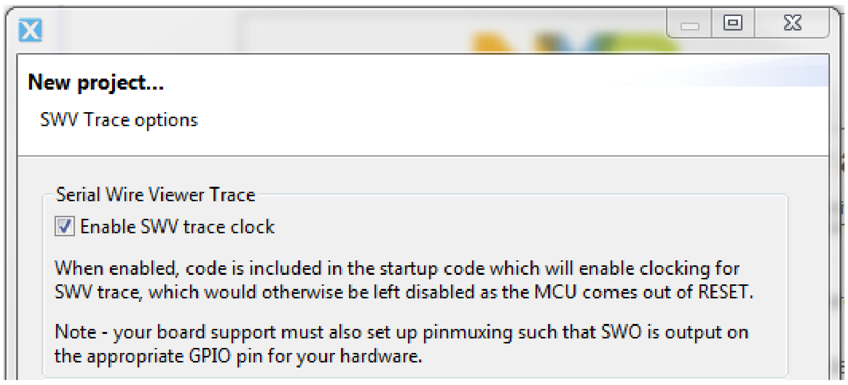
In the next menu you may import DSP libraries. Don't import anything – just click next.


New project...

CMSIS DSP Library Project Selection

If required, select the CMSIS DSP library project that you want your new project to link against.

Leave SWV trace options at their defaults and click next


New project...

SWV Trace options

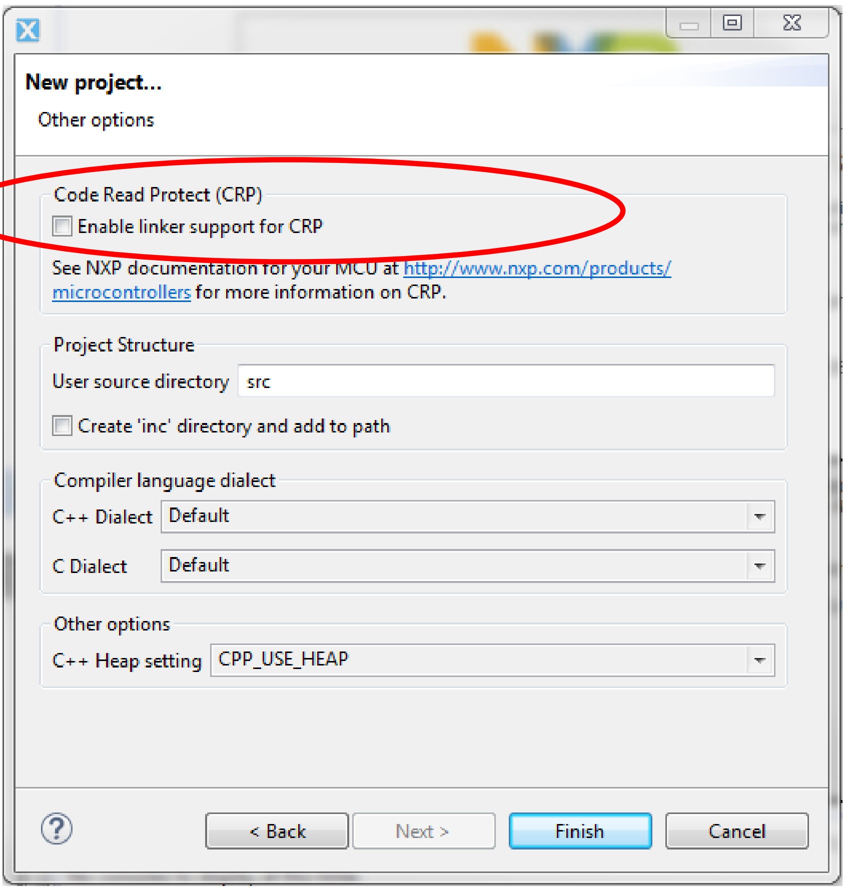
Serial Wire Viewer Trace

☑ Enable SWV trace clock

When enabled, code is included in the startup code which will enable clocking for SWV trace, which would otherwise be left disabled as the MCU comes out of RESET.

Note - your board support must also set up pinmuxing such that SWO is output on the appropriate GPIO pin for your hardware.

Now you are at the last stage of creating the project. The last stage in important! **Uncheck** Enable linker support for CRP. CRP is a security measure that prevents read back of code from the processor. Leaving this enabled can turn your board in to a fairly expensive brick.



New project...

Other options

Code Read Protect (CRP)

☐ Enable linker support for CRP

See NXP documentation for your MCU at http://www.nxp.com/products/microcontrollers for more information on CRP.

Project Structure

User source directory: src

☐ Create 'inc' directory and add to path

Compiler language dialect

C++ Dialect: Default

C Dialect: Default

Other options

C++ Heap setting: CPP_USE_HEAP

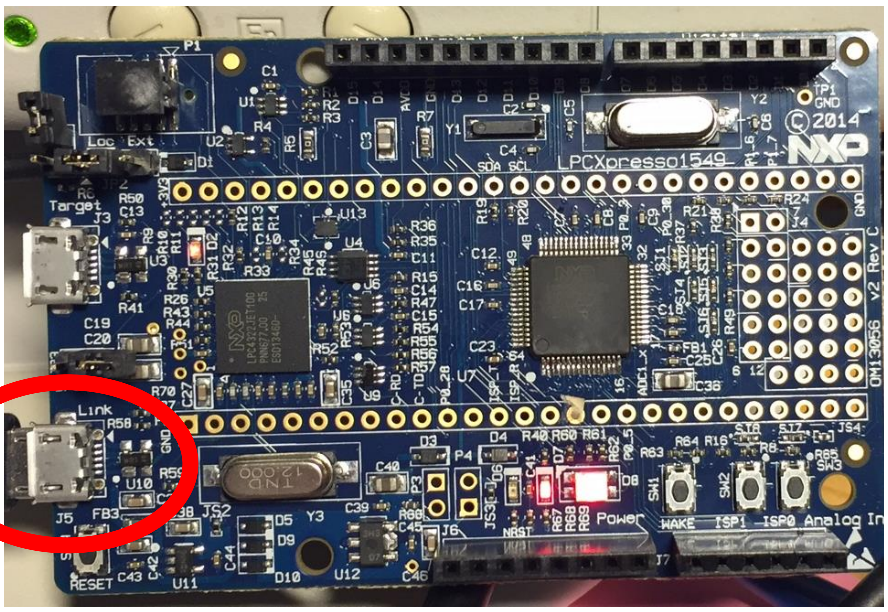
Press Finish.

Now you are done with creating a project!

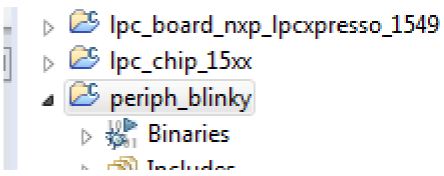
Development board and debugging

You have just created a workspace with driver libraries, an empty C++ project and a simple test project. The test project called periph_blinky uses ARM systick timer to make a led blink. We'll use that project to test that the debugger connection works properly.

Connect your development board to your computer. Note that development board has two micro-USB connectors. Make connection using the **Link** connector (next to the reset button).

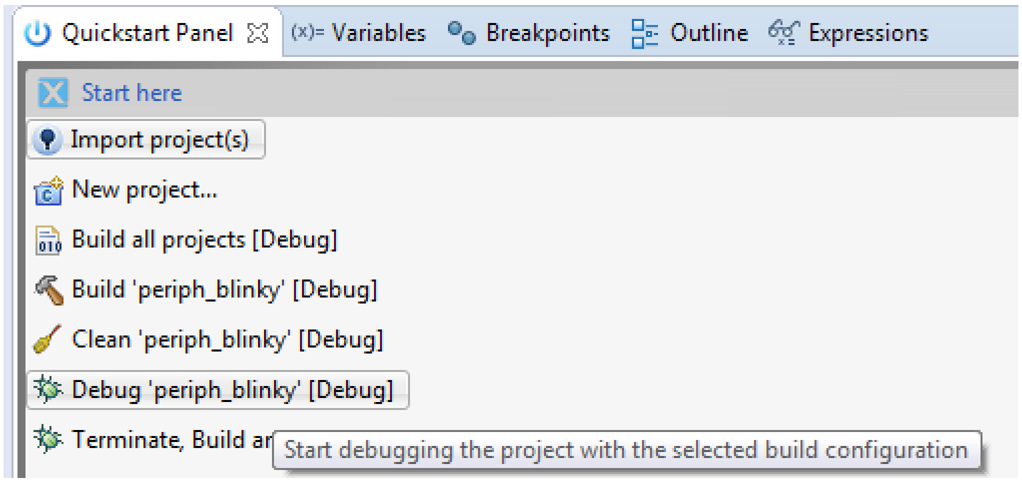


Then click on periph_blink project in Project explorer.



lpc_board_nxp_lpcxpresso_1549
lpc_chip_15xx
periph_blinky
Binaries

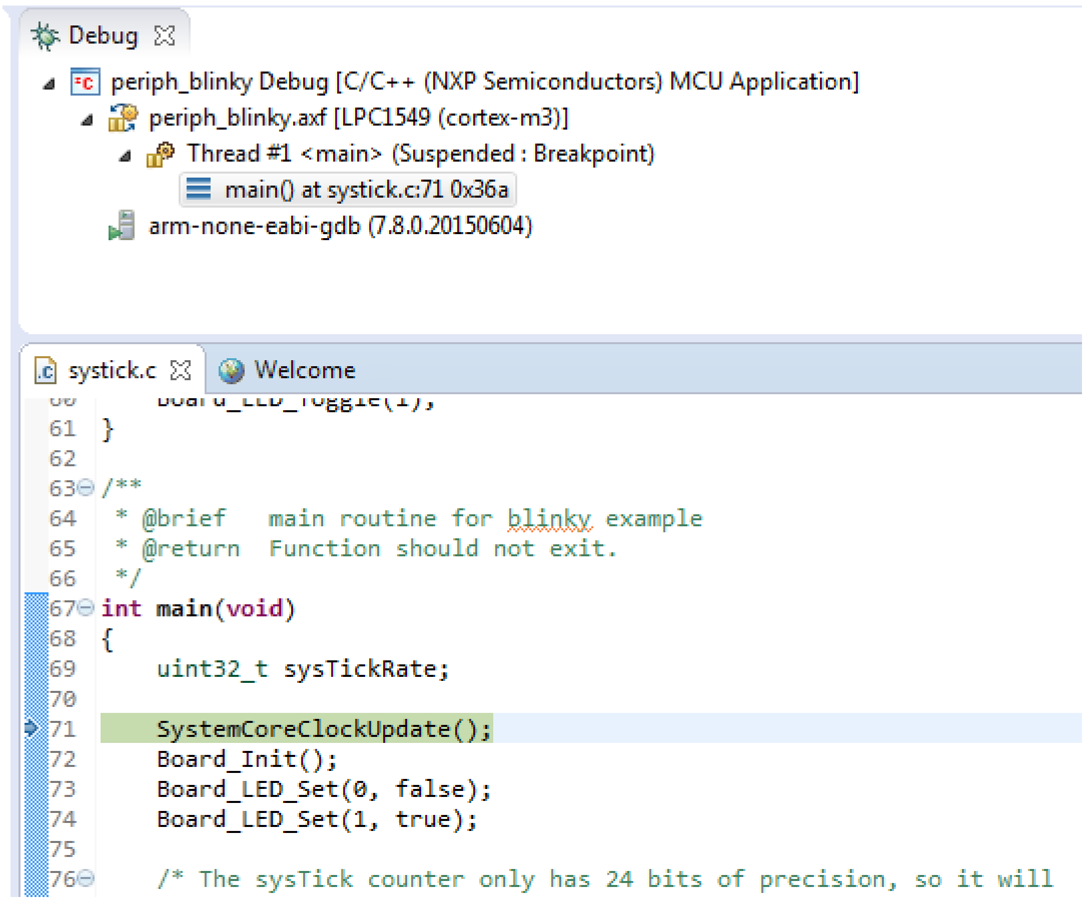
Then click Debug in the Quick start Panel.



Quickstart Panel
(x)= Variables
Breakpoints
Outline
Expressions
Start here
Import project(s)
New project...
Build all projects [Debug]
Build 'periph_blinky' [Debug]
Clean 'periph_blinky' [Debug]
Debug 'periph_blinky' [Debug]
Terminate, Build ar
Start debugging the project with the selected build configuration

IDE then compiles the program and starts a debugging session. On the first run you will be prompted to select a debugger. You should see CMSIS DAP in the list and it should be only one in the list. If you don't see it then click Refresh. If that doesn't help, try using a different USB port.

When your debugging session starts the program execution stops at the first executable line of main function. You can step into or over source lines, set break points and run your program. When you let the program continue you should see then on board led blinking vigorously. When you suspend the running program the cursor will typically be in the infinite loop at the end of main function.



```
periph_blinky Debug [C/C++ (NXP Semiconductors) MCU Application]
  periph_blinky.axf [LPC1549 (cortex-m3)]
    Thread #1 <main> (Suspended : Breakpoint)
      main() at systick.c:71 0x36a
  arm-none-eabi-gdb (7.8.0.20150604)
```

```
}

/**
 * @brief   main routine for blinky example
 * @return  Function should not exit.
 */
int main(void)
{
    uint32_t sysTickRate;

    SystemCoreClockUpdate();
    Board_Init();
    Board_LED_Set(0, false);
    Board_LED_Set(1, true);

    /* The sysTick counter only has 24 bits of precision, so it will
```

Exercise 1

The next thing to do is to modify the program so that the blinking slows down.

Stop debugging session and edit SysTick_Handler() function. Add a variable that counts up to 15. When the counter reaches 15 it is reset to zero and the leds are toggled. Note that leds shall toggle only when the counter reset is performed. To preserve the counter value between calls to SysTick_Handler the variable must be declared with static keyword. For example:

```
static int counter = 0;
```

Test your program to see if blinking got slower. If you got the counter right the led should change colour once per second.

Interrupts

Interrupts are a mechanism that allows processor to react to events (almost) immediately. When an interrupt is triggered the current CPU state is saved in to stack and CPU jumps to a function. The function that is executed is called an interrupt handler. The interrupt handler needs to be in a processor specific memory location. Usually compiler or development environment takes care of placing the interrupt handler in a proper place. With our board the interrupt handlers must have an interrupt specific name. Systick interrupt handler must be called SysTick_Handler. Interrupt handler neither has parameters nor returns a value.

Typically interrupt handler and main program need to communicate with each other. The simplest form of communication is through shared variables. One should keep the asynchronous nature of interrupts in mind when using shared variable. Shared variables must be declared with volatile type qualifier. Volatile tells the compiler not to optimize access to volatile variables and forces the variables to be always placed in the memory. Most compilers try to allocate variables into register to improve performance thus volatile keyword is needed. Unfortunately variables in registers can't be shared between main program and an interrupt handler so volatile must be used to place shared variables in the memory.

You will learn more about interrupts in later exercises.

Exercise 2

Change the systick timer frequency in periph_blinky to 1000 Hz (one interrupt/millisecond). Systick interrupt rate is configured with SysTick_Config().

Change the interrupt handler so that the green led blinks at 1 Hz frequency and the red led at 2 Hz.

Input and output

The IO-pins on our development board are bidirectional and individually configurable. The driver package includes functions for setting pin direction and for reading or writing the pin state. Put processor has three IO ports that are called PIO0, PIO1 and PIO2. Each port has 32 pins.

Study Board_LED-functions board.c in lpc_board_nxp_lpcxpresso_1549-project for an example of configuring and using pins. In the Arduino receptacles schematic diagram (later in this document) you see wires marked with PIOx_y, where x is the IO port number and y is the pin number.

The IO pin direction is configured with Chip_GPIO_SetPinDIROutput and Chip_GPIO_SetPinDIRInput. Both functions take three parameters: pointer to IO configuration registers, port number and pin number.

For example:

```
        Chip_GPIO_SetPinDIROutput(LPC_GPIO, 1, 8);
        Chip_GPIO_SetPinDIRInput(LPC_GPIO, 0, 24);
```

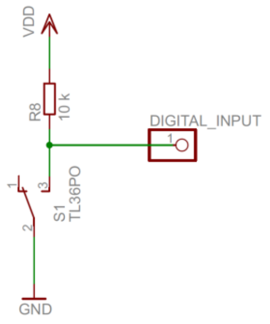
The above configures [port 1, pin 8] as an output and [port 0, pin 24] as an input.

When a pin is configured as an input some additional configuration may be required depending on what is connected to the pin.

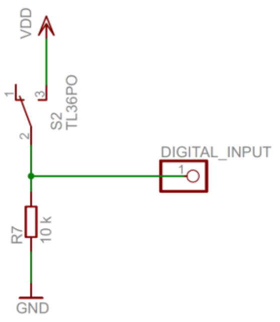
When simple (single pole) push button is connected to an input a pull up or pull down resistor is required to ensure that a pin always has a stable value. Pull up/pull down resistor will force the pin into a default state if the pin is left floating (unconnected).

A pin with button and a pull up resistor has default value of one. When the button is pressed the pin goes low and reads zero.

A pin with button and a pull down resistor has default value of zero. When the button is pressed the pin goes high and reads one.



Picture 1 Pull up resistor



Picture 2 Pull down resistor

Our gambling shield has grounding buttons and switches s we need to use pull up resistors in the input pins. Most modern microcontrollers have integrated pull up/pull down resistors that can be enabled (=connected) with software. Our LPC1549 makes no exception to this so we need to configure pull ups to read gambling shield pins.

Some pins can be used as analog inputs or connected to internal peripherals (for example UART).

The following example shows how configure one pin as a digital input with pull up resistor and inverted input. The default value when a button is not pressed is zero with a pull up resistor LPC1549 allows inputs to be inverted before the value is passed CPU. When inversion is enabled a pressed button reads one and not pressed reads zero.

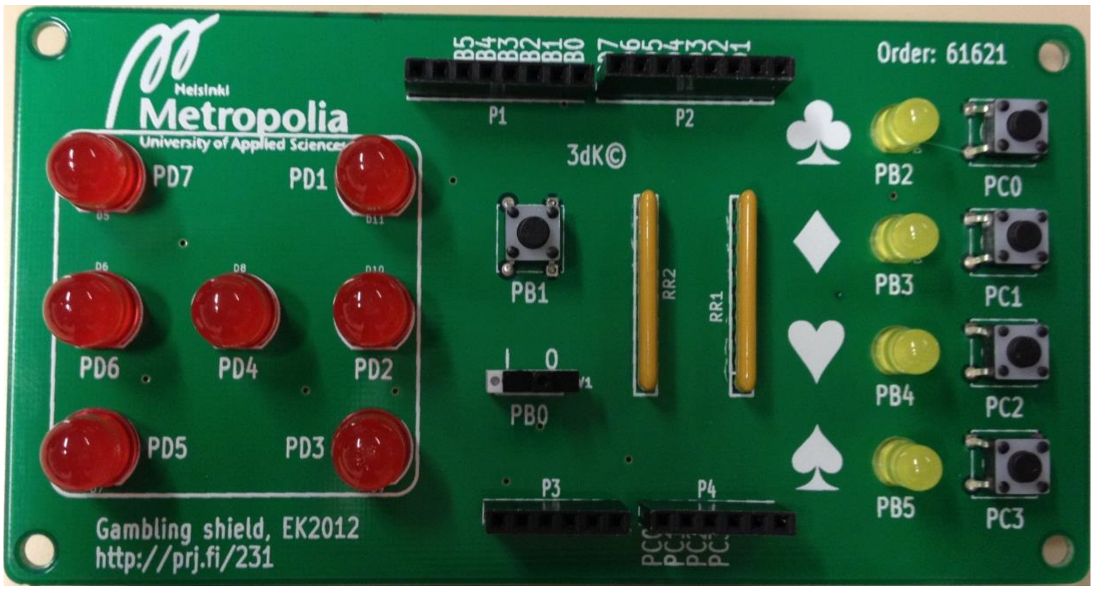
```
// Set Port0.Pin8 as a digital input with pull up
Chip_IOCON_PinMuxSet(LPC_IOCON, 0, 8, (IOCON_MODE_PULLUP | IOCON_DIGMODE_EN | IOCON_INV_EN));
Chip_GPIO_SetPinDIRInput(LPC_GPIO, 0, 8);
```

Gambling shield

Gambling shield is an Arduino compatible shield that goes in to the Arduino receptacles on the LPCXpresso board. You must match the markings on the shield with the processor pins in the schematic below. Each led, button and switch has an identifier next to it and same identifiers are used to indicate which pin in Arduino pin headers they are connected to.

Dice shaped leds are connected to PD1 – PD7. Card suit leds are connected to port BB2 –PB5. Switches and buttons are connected to PB and PC. See shield printing for connection details.

PD1 – 7 and PB2 – 5 must be configured as outputs and all pins with a button or switch must be configured as inputs

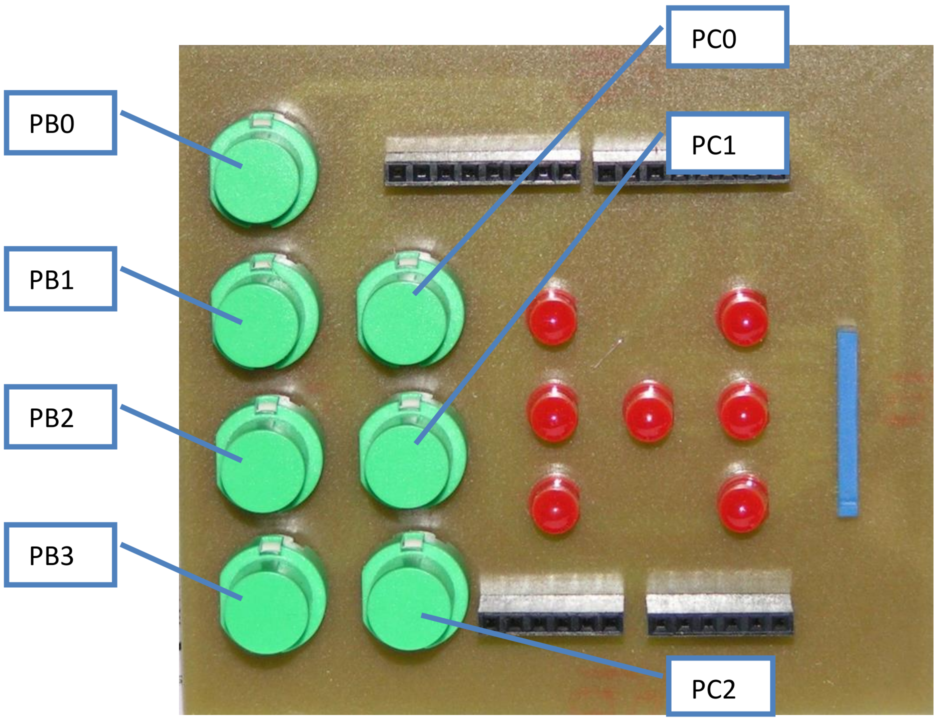


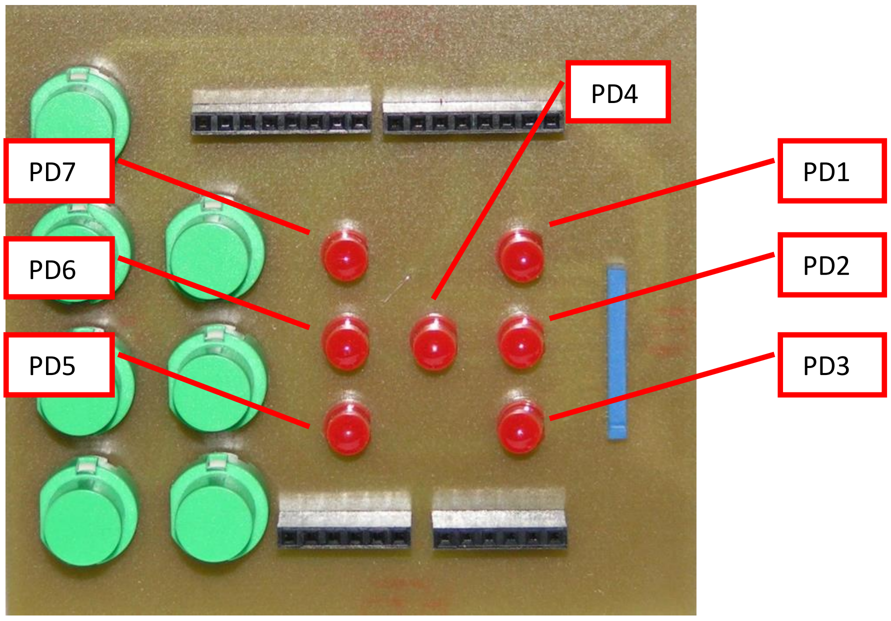
Gambling shield configuration:

- Input: D.0, B.0, B.1, C.0, C.1, C.2, C.3
- Outputs: D.1 – D.7, B.2 – B.5

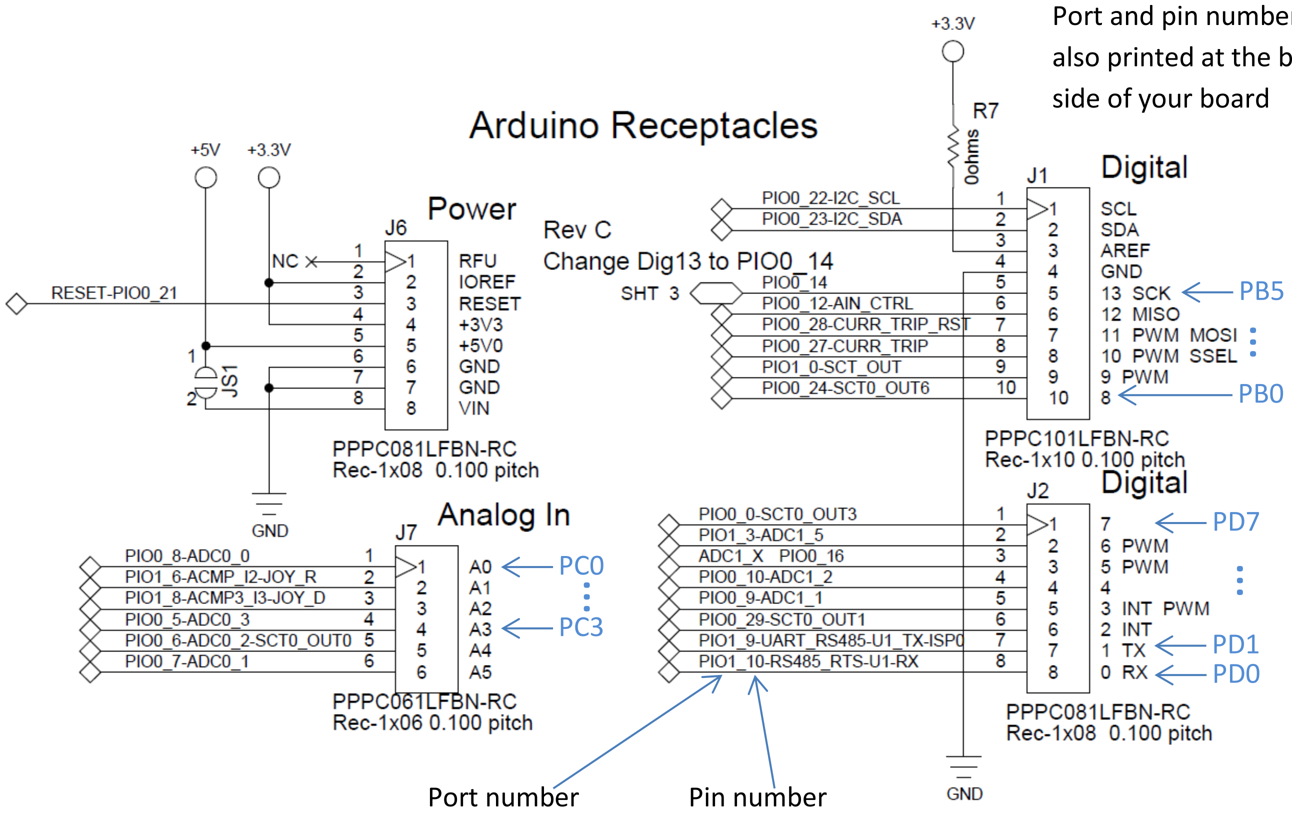
Old gambling shield

The old version of the gambling shield has more buttons and a slightly different button layout. Note that using buttons PC0, PC1 or PC2 allows you to test your dice with either of the gambling shield types.





Port and pin numbers are also printed at the bottom side of your board



The schematic above indicates correspondence of gambling shield markings and Arduino receptacles. The blue texts on the right hand side of connector symbols are the markings found on the gambling shield. To the left of each connector symbols you can find which processor pin each Arduino pin is connected to. The first digit after PIO indicates port number (0 – 2) and the number after the underscore indicates pin number (0 – 31). This pair is needed to configure the pin mode and direction in your program.

Remember to enable pull ups and digital mode on the input pins when gambling shield is used.

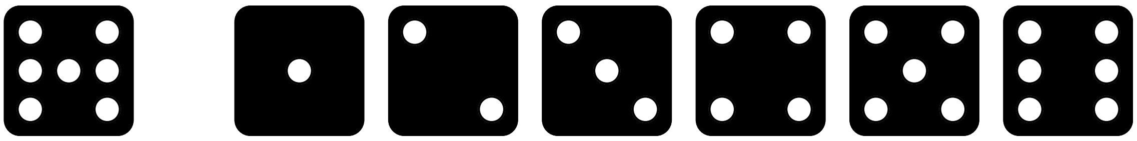
Exercise 3 – Programming assignment

Design electronic dice according to the specification below.

Electronic dice consists of seven leds and two buttons.

One button is a test button which turns on all the leds and keeps the leds on until the button is released. When the test button is released all leds are switched off.

The second button is an operate button which when pressed and released randomly displays a number between 1 and 6. Number is displayed until user operates dice again or if test button is pressed. All leds must be off while the operate button is pressed.



Outputs of test button and numbers 1 – 6 are shown in the picture above.

Implementation requirements

To get a random number make a variable run in circle between 1 - 6 at high frequency, while the button is pressed. High counting frequency prevents user from picking numbers by releasing the button at a specific time. When the button is released display the number that was left in the counter when button was released.

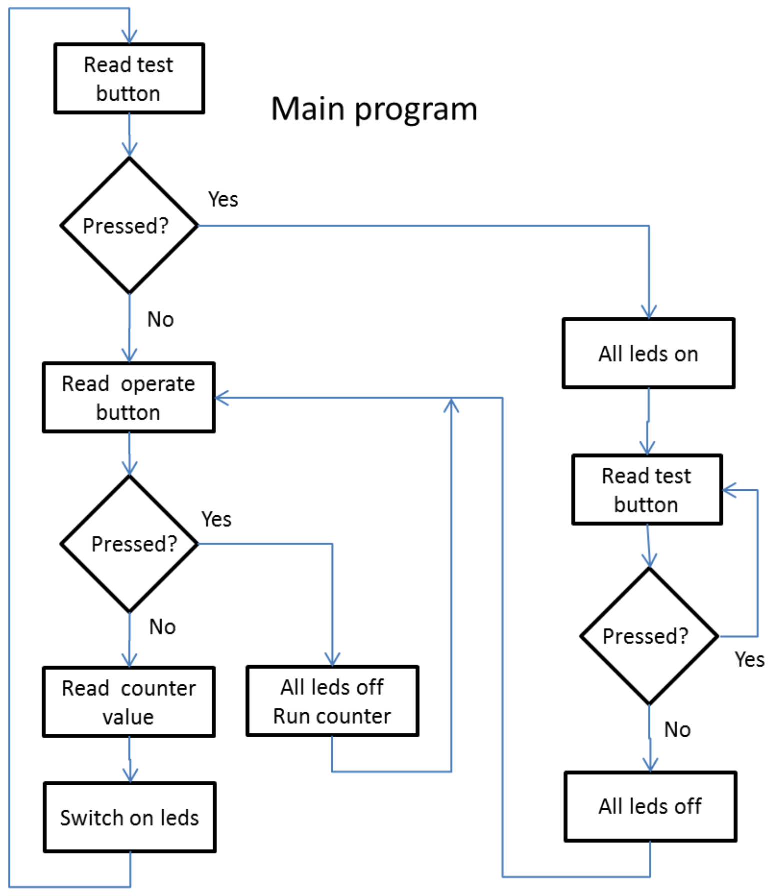
Write a class for displaying a number with dice shaped led pattern.

- The class must have a method for displaying the number. When the display method is called with 0 all leds are switched off and when called with a number that is greater than six then all leds are switched on. When number is between 1 and 6 the number is displayed as specified in the picture above
- The constructor must configure the direction and mode of the output pins and switch all leds off

```
class Dice {
public:
             Dice();
             virtual ~Dice();
             void Show(int number);
} /* you need to figure out yourself the private members that you need */
```

The classes help in raising the abstraction level of a program by hiding the implementation details behind the class interface. LPCXpresso IDE has wizard for creating a class. The wizard creates header and code templates and adds them to your project.

Flow chart



Example code

One way of configuring the gambling shield dice leds as outputs.

```
#define MAXLEDS 7
static const uint8_t diceport[] = { 0, 1, 0, 0, 0, 0, 1 };
static const uint8_t dicepins[] = { 0, 3, 16, 10, 9, 29, 9 };

int idx;

for (idx = 0; idx < MAXLEDS; idx++) {
             /* Set the GPIO as output with initial state off (low) */
             Chip_GPIO_SetPinDIROutput(LPC_GPIO, diceport[idx], dicepins[idx]);
             Chip_GPIO_SetPinState(LPC_GPIO, diceport[idx], dicepins[idx], false);
}
```

Digital inputs with pull ups and inverting input

```
// Set Port0.Pin8 as a digital input with pull up
Chip_IOCON_PinMuxSet(LPC_IOCON, 0, 8, (IOCON_MODE_PULLUP | IOCON_DIGMODE_EN | IOCON_INV_EN));
Chip_GPIO_SetPinDIRInput(LPC_GPIO, 0, 8);
// Set Port0.Pin24 as a digital input with pull up
Chip_IOCON_PinMuxSet(LPC_IOCON, 0, 24, (IOCON_MODE_PULLUP | IOCON_DIGMODE_EN | IOCON_INV_EN));
Chip_GPIO_SetPinDIRInput(LPC_GPIO, 0, 24);
```