Application Note: AN00189

Using QuadSPI for application overlay data

This application note demonstrates how to use overlay regions and how to use QuadPSI flash memory for storing and loading overlay data.

This application note provides an example that will load an overlay function from QuadSPI flash memory to illuminate a colored LED depending on which, if any buttons are pressed.

Required tools and libraries

- xTIMEcomposer Tools Suite version 14.0 or later is required.

Required hardware

This application note is designed to run on an XMOS xCORE-200 series device.

The example code provided with the application has been implemented and tested on the xCORE-200 explorerKIT core module board but there is no dependancy on this board and it can be modified to run on any development board which uses an xCORE-200 series device.

Prerequisites

- This document assumes familiarity with the XMOS xCORE-200 architecture, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the *References* appendix.
- This document assumes familiarity with QuadSPI flash memory, the xCORE quadflash library and the XMOS tool XFLASH.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- The XMOS tools manual contains information regarding the use of xCORE devices[2].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xtimecomposer-user-guide

1 Introduction

xCORE-200 explorerKIT contains everything you need to start developing applications on the powerful xCORE-200 multicore microcontroller products from XMOS. It's easy to use and provides lots of advanced features on a small, low cost platform.

The xCORE-200 explorerKIT features our XE216-512 xCORE-200 multicore microcontroller. This device has sixteen 32bit logical cores that deliver up to 2000MIPS completely deterministically. The combination of 100/1000 Mbps Ethernet, high speed USB and 53 high performance GPIO make the xCORE-200 explorerKIT an ideal platform for functions ranging from robotics and motion control to networking and digital audio.

1.1 Block diagram

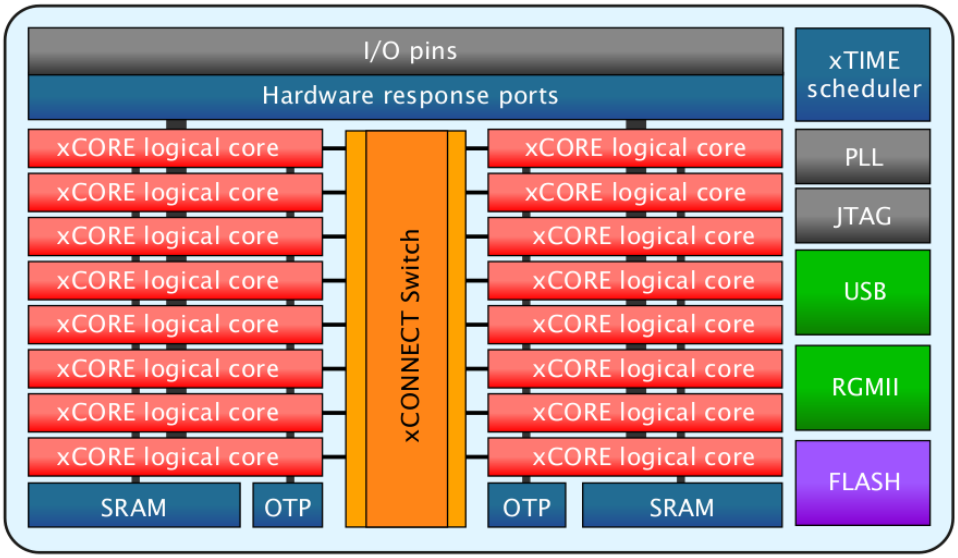

Figure 1: Block diagram of XE216-512 device on xCORE-200 explorerKIT

1.2 QuadSPI flash memory

The xCORE-200 explorerKIT features a QuadSPI flash device providing a 4-bit multiplexed I/O serial interface to boost performance while maintaining the compact form factor of standard serial flash devices. By using QuadSPI flash memory the overlay data is loaded significantly faster than with traditional SPI flash memory.

The QuadSPI flash memory is logically split between a boot and data partition.

The boot partition consists of a flash loader followed by a factory image and zero or more optional upgrade images. Each image starts with a descriptor that contains a unique version number and a header that contains a table of code/data segments for each tile used by the program and a CRC.

An overlay is a block of code and data that is loaded on demand at runtime. Each overlay has a predetermined region of memory that it is copied to called an overlay region. Several overlays may be associated with the same overlay region, but only one of these overlay can be loaded at any one time.

Overlays reduce the amount of memory needed to run your application since it is no longer necessary to reserve space for all your code and data - instead the tools only need to reserve space for the largest overlay that can be loaded into each overlay region.

In this application note example, three functions will be stored in an overlay region. Once booted from QuadSPI flash memory, the application note example will load a default function which will illuminate one of the LED's on the xCORE-200 explorerKIT core module board. When button 1 or button 2 is depressed, the default overlay function will be swapped for one of the other overlay functions stored in QuadSPI flash memory.

This application note demonstrates

- How to create and use overlay regions.
- How to use the XCC option -foverlay
- How to use XFLASH to store the overlay data in QuadSPI flash memory.

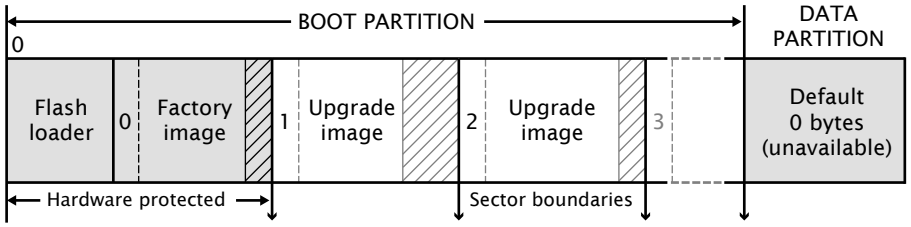

Figure 2: Flash format diagram

2 Using QuadSPI flash memory for overlay data

2.1 Source code structure for this application note

This application note example is stored in the following directory structure:

```
src                                     <-- top level project source directory
  qspi_overlays.xc                      <-- source file for project
Makefile                                <-- make file for project
```

2.2 Makefile support for using overlay regions loaded from QuadSPI

It is a requirement to tell the xCORE toolchain that the application is using overlay regions that will be loaded from QuadSPI flash memory. This is achieved by passing the compiler option `-foverlay=quadflash` with the associated build flags. To use this runtime the application must be booted from QuadSPI flash memory.

Options passed to xCORE build tools from makefile:

```
XCC_FLAGS = -O2 -g -foverlay=quadflash
```

2.3 The application chooseOverlayRegion function

The chooseOverlayRegion() function for this example is contained within the file `qspi_overlays.xc` and is as follows,

```
void chooseOverlayRegion()
{
  overlay_quadflash_init(move(g_qspi_ports_ptr), 100, 8);
  int current = 0;
  while (1)
  {
    select {
    case g_button_port when pinsneq(current) :> current:
      break;
    default:
      if ((current & BUTTON1) == 0)
        ledGreen(g_led_ports);
      else if ((current & BUTTON2) == 0)
        ledRed(g_led_ports);
      else
        ledBlue(g_led_ports);
      break;
    }
  }//while
}
```

Looking at this function in more detail you can see the following:

- An overlay initialization function is called
- An overlay region function is called when no buttons are depressed.
- An overlay region function is called when button 1 is depressed.
- An overlay region function is called when button 2 is depressed.

2.4 The application overlay regions

In this application note example there are three functions stored within a single overlay region.

```
//Function in an overlay region
[[overlay]]
void ledBlue(out port led_ports)
{
  char blue   = 2;
  led_ports <: blue;
}

//Function in an overlay region
[[overlay]]
void ledGreen(out port led_ports)
{
  char green = 4;
  led_ports <: green;
}

//Function in an overlay region
[[overlay]]
void ledRed(out port led_ports)
{
  char red   = 8;
  led_ports <: red;
}
```

When compiled, the functions `ledBlue`, `ledGreen` and `ledRed` will overlay each other in memory and therefore only one can be loaded into RAM at any point in time.

Each function is responsible for driving one of the LED colors on the RGB LED's featured on the xCORE-200 explorerKIT core module board.

```
//LED port on explorerKIT
on tile[0] : out port g_led_ports = XS1_PORT_4F;
```

2.5 Initializing QuadSPI flash memory for overlay use

In order to use overlay regions from QuadSPI flash memory the xCORE function `overlay_quadflash_init` must be called.

```
overlay_quadflash_init(move(g_qspi_ports_ptr), 100, 8);
```

This function is declared in the header file `overlay_quadflash.h`

```
#include <overlay_quadflash.h>
```

This function requires three arguments. The first argument is a movable pointer to the structure `fl_QSPIPorts` also defined in the header file `overlay_quadflash.h`. The QuadSPI ports used to initialize this structure have been convieniently defined within the XCORE-200-EXPLORER.xn file. A single clock block is also required.

```
//QuadSPI Ports on explorerKIT
on tile[0] : static fl_QSPIPorts g_qspi_ports = {
  PORT_SQI_CS,
  PORT_SQI_SCLK,
  PORT_SQI_SIO,
  XS1_CLKBLK_1
};

//A movable pointer to the QuadSPI ports.
static fl_QSPIPorts * movable g_qspi_ports_ptr = &g_qspi_ports;
```

The QuadSPI clock frequency in MHz is specified as a ratio using the last two arguments of overlay_quadflash_init. In this case the clock frequency is set to 12.5 MHz (i.e. 100 / 8).

2.6 Deciding which function to load from the overlay region

Inside the while loop of the chooseOverlayRegion function the status of the buttons on the explorerKIT is tested to determine which function is loaded from the overlay region.

```
while (1)
{
  select {
  case g_button_port when pinsneq(current) :> current:
    break;
  default:
    if ((current & BUTTON1) == 0)
      ledGreen(g_led_ports);
    else if ((current & BUTTON2) == 0)
      ledRed(g_led_ports);
    else
      ledBlue(g_led_ports);
    break;
  }
}//while
```

The button status is obtained by performing an input on the 4-bit buffered port g_button_port

```
//Button port on explorerKIT
on tile[0] : in buffered port:4 g_button_port = XS1_PORT_4E;
```

If the value input from the g_button_port has changed then the new value is stored into the variable current.

```
case g_button_port when pinsneq(current) :> current:
  break;
```

If the value held in the current variable is equal to the value defined for NOBUTTON then the ledBlue function is loaded into RAM from QuadSPI flash memory. The blue LED on the xCORE-200 explorerKIT will now be illuminated.

If the value held in the current variable is equal to the value defined for BUTTON1 then the ledGreen function is loaded into RAM from QuadSPI flash memory. The green LED on the xCORE-200 explorerKIT will now be illuminated.

If the value held in the current variable is equal to the value defined for BUTTON2 then the ledRed function is loaded into RAM from QuadSPI flash memory. The red LED on the xCORE-200 explorerKIT will now be illuminated.

APPENDIX A - Example Hardware Setup

This application example is designed to run on the xCORE-200 explorerKIT core module board. The xCORE-200 explorerKIT core module board should be connected to both power and have the development adapters connected to a host machine to allow program download. This can be seen in the following image.

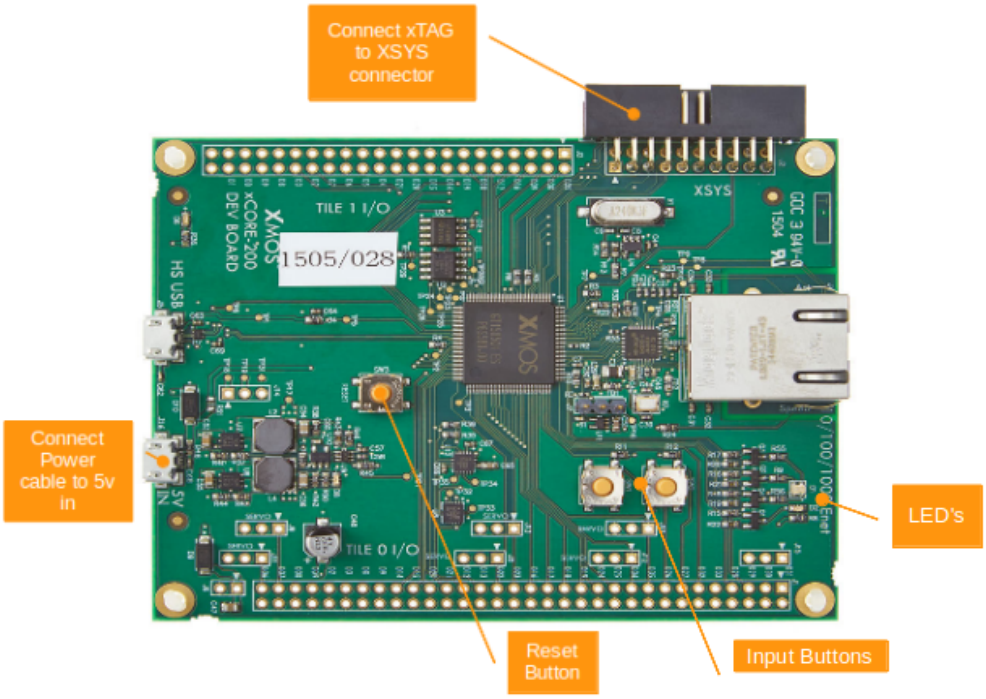

Figure 3: XMOS xCORE-200 explorerKIT

The hardware should be configured as displayed above for this example:

- The XTAG debug adapter should be connected to the XSYS connector and the XTAG USB cable should be connected to the host machine
- The xCORE-200 explorerKIT should have the power cable connected
- The RESET button can be used to repeatedly boot the xCORE-200 device from QuadSPI

APPENDIX B - Launching the example application

Once the example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer Studio the application and data.bin file can be written to the QuadSPI flash memory of the xCORE-200 explorerKIT module board.

Once built there will be a bin directory within the project which contains the binary for the xCORE-200 tile. The xCORE-200 binary has a XMOS standard .xe extension.

B.1 Writing to flash memory from the command line

From the command line the xflash tool is used to download the code to the flash device on the xCORE-200 explorerKIT module board. The complete XFLASH command line is as follows:

```
> xflash --boot-partition-size 0x80000 bin/qspi-overlays.xe
```

B.2 Writing to flash memory from xTIMEcomposer Studio

From xTIMEcomposer Studio there is the Flash As mechanism to edit the flash configuration and download the code to the flash device on the xCORE-200 explorerKIT core module board. Within the flash configuration editor the XFLASH options can be explicitly set. In this example, the Boot Partition Size checkbox is ticked and a corresponding value of 0x80000 is inserted.

Once XFLASH has sucessfully programmed the flash memory device on the xCORE-200 explorerKIT module board, the xCORE-200 device is reset. The blue LED will be illuminated as no buttons are currently depressed. Depress the button labelled as SW1 and the blue LED will go out with the green LED illuminated instead. Releasing the button SW1 will disable the green LED and illuminate the blue LED once more. Depress the button labelled as SW2 and the blue LED will go out with the red LED illuminated instead. Releasing the button SW2 will disable the red LED and illuminate the blue LED once more.

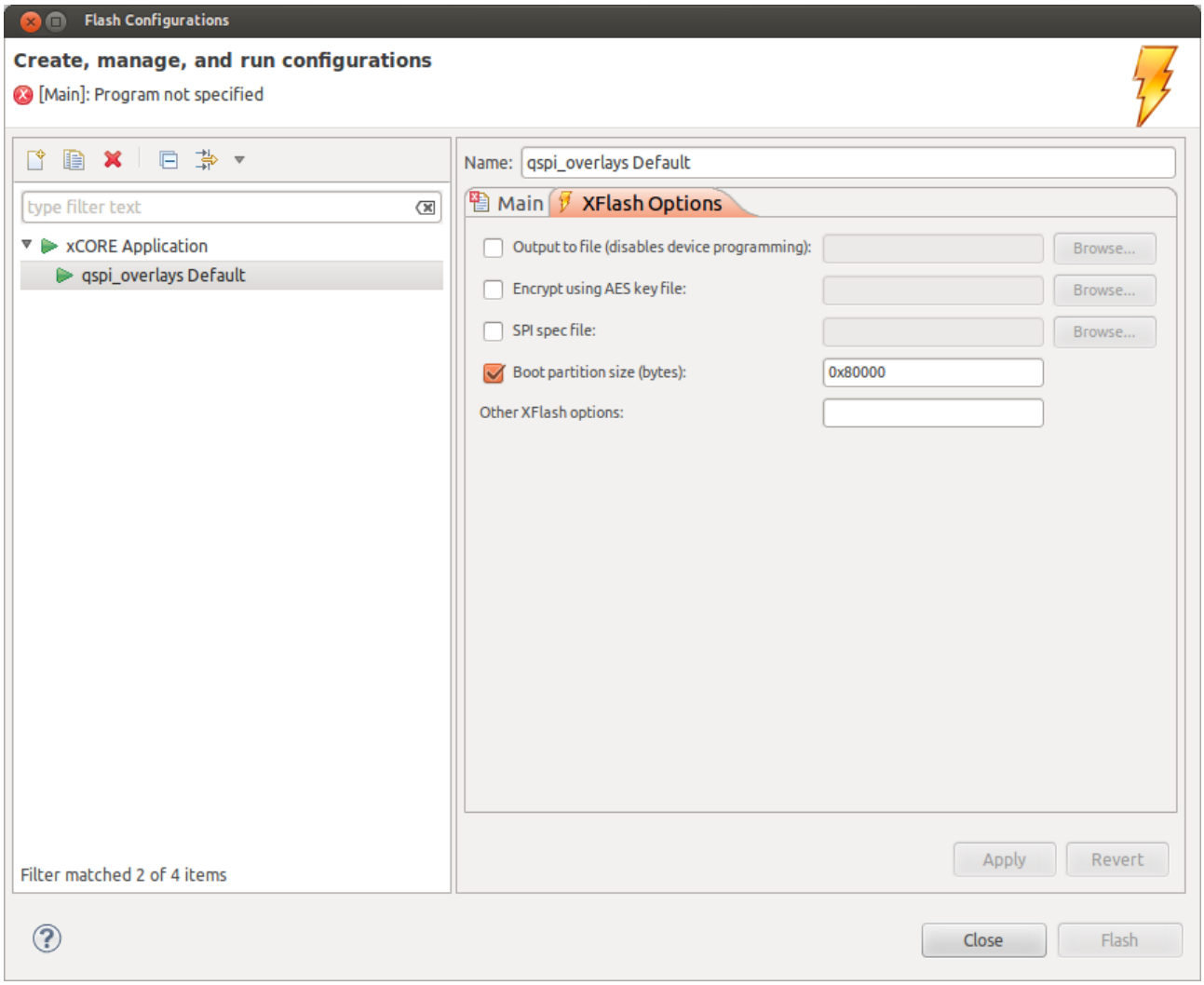

Figure 4: xTIMEcomposer studio flash configuration editor

APPENDIX C - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

APPENDIX D - Full source code listing

D.1 Source code for qspi_overlays.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <platform.h>
#include <stdio.h>
#include <print.h>
#include <overlay_quadflash.h>

//Defines that represent button state on explorerKIT.
#define NOBUTTON 0
#define BUTTON1 1 //Bit 0
#define BUTTON2 2 //Bit 1

//LED port on explorerKIT
on tile[0] : out port g_led_ports = XS1_PORT_4F;

//Button port on explorerKIT
on tile[0] : in buffered port:4 g_button_port = XS1_PORT_4E;

//QuadSPI Ports on explorerKIT
on tile[0] : static fl_QSPIPorts g_qspi_ports = {
  PORT_SQI_CS,
  PORT_SQI_SCLK,
  PORT_SQI_SIO,
  XS1_CLKBLK_1
};

//A movable pointer to the QuadSPI ports.
static fl_QSPIPorts * movable g_qspi_ports_ptr = &g_qspi_ports;

//Function in an overlay region
[[overlay]]
void ledBlue(out port led_ports)
{
  char blue   = 2;
  led_ports <: blue;
}

//Function in an overlay region
[[overlay]]
void ledGreen(out port led_ports)
{
  char green = 4;
  led_ports <: green;
}

//Function in an overlay region
[[overlay]]
void ledRed(out port led_ports)
{
  char red   = 8;
  led_ports <: red;
}

//Function that will initialize overlays for use and then load the correct
```

```
//function depending on the status of the buttons
void chooseOverlayRegion()
{
  overlay_quadflash_init(move(g_qspi_ports_ptr), 100, 8);
  int current = 0;
  while (1)
  {
    select {
    case g_button_port when pinsneq(current) :> current:
      break;
    default:
      if ((current & BUTTON1) == 0)
        ledGreen(g_led_ports);
      else if ((current & BUTTON2) == 0)
        ledRed(g_led_ports);
      else
        ledBlue(g_led_ports);
      break;
    }
  }//while
}

int main(void)
{
  par
  {
    on tile[0]:
    {
      chooseOverlayRegion();
    }

    on tile[1]:
    {
      while(1);
    }
  }

  return 0;
}
```