Multiple Firmware Booting

This application note demonstrates how to use XFLASH option `--loader` to change the behaviour of selecting which image to boot by the flash loader.

This application note provides an example that uses a button to select the image that gets booted. When the button is in the up position at power on, the highest even number image version will be booted. When the button is in the down position at power on, the highest odd number image version will be booted.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.0

Required hardware

This application note is designed to run on an XMOS startKIT.

The example code provided with the application has been implemented and tested on the startKIT but there is no dependancy on this board and it can be modified to run on any development board.

Prerequisites

- This document assumes familiarity with the XMOS xCORE 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.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].

---

[1] http://www.xmos.com/published/glossary

1 Overview

1.1 Introduction

The 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.

The flash loader scans flash memory searching for valid images to boot. By default the flash loader boots the image with the highest version number. This default behavior can be modifed by writing some simple functions and using the XFLASH option --loader.

An application designer may wish to change this default behavior because a device may need to operate differently depending on the position of a switch, jumper, button or if an external device is attached or detached. Using the XFLASH option --loader gives the application designer control to decide which image version to boot based on one or more of these inputs.

In this application note example, a button is used to select which version is booted.

This application note demonstrates

- How to write and compile the functions used to select the version to be booted.
- How to use the XFLASH option --loader to ensure these functions are used by the flash loader.
- How to provide an upgrade to the versions.

2 Multiple Image Boot application note

2.1 Source code structure for this application note

There are four projects contained within this source code structure. There is also the file `loader.xc` which contains the functions required to modify the behavior of the flash loader.

These are stored in the following directory structure:

```
src                              <-- top level project source directory
  app_factory_image_button_down  <-- project
    src                          <-- source directory for project
      main.xc                    <-- source file for project
    Makefile                     <-- make file for project
  app_factory_image_button_up    <-- project
    src                          <-- source directory for project
      main.xc                    <-- source file for project
    Makefile                     <-- make file for project
  app_upgrade_image_button_down  <-- project
    src                          <-- source directory for project
      main.xc                    <-- source file for project
    Makefile                     <-- make file for project
  app_upgrade_image_button_up    <-- project
    src                          <-- source directory for project
      main.xc                    <-- source file for project
    Makefile                     <-- make file for project
  loader                         <-- directory for the loader functions
    loader.xc                    <-- functions used to select the image version to boot
```

Project app_factory_image_button_down is used to create a factory image version that boots when the button on the startKIT is pressed as power is turned on. This repeatedly flashes a diamond pattern on the LED's where the LED's are either on or off.

Project app_factory_image_button_up is used to create a factory image version that boots when the button on the startKIT is not pressed as power is turned on. This repeatedly flashes a criss-cross pattern on the LED's where the LED's are either on or off.

Project app_upgrade_image_button_down is used to create an upgrade image version that boots when the button on the startKIT is pressed as power is turned on. This repeatedly flashes a diamond pattern on the LED's where the LED's smoothly transition between on and off.

Project app_upgrade_image_button_up is used to create an upgrade image version that boots when the button on the startKIT is not pressed when power is turned on. This repeatedly flashes a criss-cross pattern on the LED's where the LED's smoothly transition between on and off.

The file loader.xc contains the functions required to change the flash loaders default behavior of selecting the image with the highest version number.

Two modules used to drive GPIO on the XMOS startKIT are also used but are not described as they fall outside the scope of this application note.

2.2 Writing the functions to select and boot the desired image

The functions for selecting and booting the desired image are contained within the file `loader.xc`.

IMPORTANT NOTE:

```
As of tools 13.2 the loader functions may get called twice, once before the pll is reset and once after.
Application designers need to take this into careful consideration when writing the following functions.
```

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
```

```
/* Port for button on startKIT board. */
in port p_button = XS1_PORT_32A;

/* Enum for representing button state. */
enum button_val
{
  BUTTON_UP,
  BUTTON_DOWN
};

/* Store the button position. */
int buttonPosition;

/* Enum for representing the potential interest in the image. */
enum interest
{
  NOT_INTERESTED = 0,
  INTERESTED = 1
};

/* Store the version of the image in memory that will potentially be booted. */
int candidateImageVersion = -1;

/* Store the address of the image in memory that will potentially be booted. */
unsigned candidateImageAddress;

void init(void)
{
  unsigned buttonVal;

  /* Read state of button. */
  p_button :>  buttonVal;

  /* Button is up. */
  if((buttonVal & 1) == 1)
  {
    buttonPosition = BUTTON_UP;
  }
  /* Button is down. */
  else
  {
    buttonPosition = BUTTON_DOWN;
  }
}

int checkCandidateImageVersion(int imageVersion)
{
  /* If the button is up and imageVersion is even and imageVersion is higher */
  /* than the last candidateImageVersion then this is a potential version */
  /* for booting. */
  if ((buttonPosition == BUTTON_UP) &&
      ((imageVersion % 2) == 0) &&
      (imageVersion > candidateImageVersion))
  {
    return INTERESTED;
  }
  /* If the button is down and imageVersion is odd and imageVersion is higher */
  /* than the last candidateImageVersion then this is a potential version */
```

```
  /* for booting. */
  else if ((buttonPosition == BUTTON_DOWN) &&
          ((imageVersion % 2) != 0) &&
          (imageVersion > candidateImageVersion))
  {
    return INTERESTED;
  }

  /* Not a potential firmware image in all other cases. */
  return NOT_INTERESTED;
}

void recordCandidateImage(int imageVersion, unsigned imageAddress)
{
  /* Save the imageVersion that we are interested in. */
  candidateImageVersion = imageVersion;
  /* Save the imageAddress of the imageVersion that we are interested in. */
  candidateImageAddress = imageAddress;
}

unsigned reportSelectedImage(void)
{
  /* Return the candidateImageAddress of the image that we are interested in. */
  return candidateImageAddress;
}
```

In this example the button on input port XS1_PORT_32A is used to decide which image version to boot. The include file <xs1.h> is required for this port definition.

```
/* Port for button on startKIT board. */
in port p_button = XS1_PORT_32A;
```

The first function that is called by the flash loader is the init() function. This function is called before the flash loader starts to search flash memory looking for images to boot.

In this example the init() function is used to test the value of the button on input port XS1_PORT_32A and save this value to the global symbol buttonPosition. The enumerator button_val is used to represent the current status of the button.

```
/* Enum for representing button state. */
enum button_val
{
  BUTTON_UP,
  BUTTON_DOWN
};
```

```
/* Store the button position. */
int buttonPosition;
```

```
void init(void)
{
  unsigned buttonVal;

  /* Read state of button. */
  p_button :> buttonVal;

  /* Button is up. */
  if((buttonVal & 1) == 1)
  {
    buttonPosition = BUTTON_UP;
  }
  /* Button is down. */
  else
  {
    buttonPosition = BUTTON_DOWN;
  }
}
```

After calling init(), the flash loader scans flash memory looking for valid images to boot. When the flash loader finds an image header it performs a CRC check of the first 256 byte page of that image. If the first page of the image is valid then the flash loader calls the function checkCandidateImageVersion(int imageVersion). The imageVersion parameter indicates the version of the upgrade image found by the flash loader. All images contained within flash memory must have a unique version number. The function checkCandidateImageVersion(int imageVersion) must return an integer value indicating whether this imageVersion is of interest or not. The value 1 is used to represent that the imageVersion is of interest. The value 0 is used to represent that the imageVersion is not of interest.

```
/* Enum for representing the potential interest in the image. */
enum interest
{
  NOT_INTERESTED = 0,
  INTERESTED = 1
};
```

In this example the function checkCandidateImageVersion(int imageVersion) checks the value of the button position stored in the global symbol buttonPosition. If the button is in the up position and the imageVersion is an even version number and the imageVersion is higher than the value stored in the global symbol candidateImageVersion, then INTERESTED is returned. If the button is in the down position and the imageVersion is an odd version number and the imageVersion is higher than the value stored in the global symbol candidateImageVersion, then INTERESTED is returned. In all other cases the value NOT_INTERESTED is returned.

```
int checkCandidateImageVersion(int imageVersion)
{
  /* If the button is up and imageVersion is even and imageVersion is higher */
  /* than the last candidateImageVersion then this is a potential version */
  /* for booting. */
  if ((buttonPosition == BUTTON_UP) &&
      ((imageVersion % 2) == 0) &&
      (imageVersion > candidateImageVersion))
  {
    return INTERESTED;
  }
  /* If the button is down and imageVersion is odd and imageVersion is higher */
  /* than the last candidateImageVersion then this is a potential version */
  /* for booting. */
  else if ((buttonPosition == BUTTON_DOWN) &&
          ((imageVersion % 2) != 0) &&
          (imageVersion > candidateImageVersion))
  {
    return INTERESTED;
  }

  /* Not a potential firmware image in all other cases. */
  return NOT_INTERESTED;
}
```

If the function checkCandidateImageVersion(int imageVersion) returns NOT_INTERESTED the flash loader continues to search flash memory looking for further upgrade image versions.

If the function checkCandidateImageVersion(int imageVersion) returns INTERESTED the flash loader performs a full CRC check of the image. If the CRC is valid then the flash loader calls the function recordCandidateImage(int imageVersion, unsigned imageAddress). The parameter imageVersion is passed in by the flash loader and shall be the same version number as given to the function checkCandidateImageVersion(int imageVersion). The parameter imageAddress is also passed in by the flash loader and represents the address in flash memory where this version of the image is found.

In this example the function recordCandidateImage(int imageVersion, unsigned imageAddress) saves the imageVersion value into the global symbol candidateImageVersion and saves the imageAddress value into the global symbol candidateImageAddress.

```
/* Store the version of the image in memory that will potentially be booted. */
int candidateImageVersion = -1;

/* Store the address of the image in memory that will potentially be booted. */
unsigned candidateImageAddress;
```

```
void recordCandidateImage(int imageVersion, unsigned imageAddress)
{
  /* Save the imageVersion that we are interested in. */
  candidateImageVersion = imageVersion;
  /* Save the imageAddress of the imageVersion that we are interested in. */
  candidateImageAddress = imageAddress;
}
```

On return from the function recordCandidateImage the flash loader continues to search flash memory looking for additional images. The process of calling checkCandidateImageVersion(int imageVersion) and recordCandidateImage(int imageVersion, unsigned imageAddress) repeats until all of flash

memory has been searched.

Once the search is complete the flash loader calls the function reportSelectedImage(). This function returns the address in flash memory of the image that is to be booted.

In this example the function reportSelectedImage() is used to return the value of the candidateImageAddress variable that represents the address of the highest valid image to boot.

```
unsigned reportSelectedImage(void)
{
  /* Return the candidateImageAddress of the image that we are interested in. */
  return candidateImageAddress;
}
```

APPENDIX A - Example Hardware Setup

This application note example is designed to run on the xCORE startKIT module board which has a single button, and 9 LED's positioned as a 3x3 square. The xCORE startKIT module board should be connected to a host machine via the USB cable supplied with the board to allow program download.
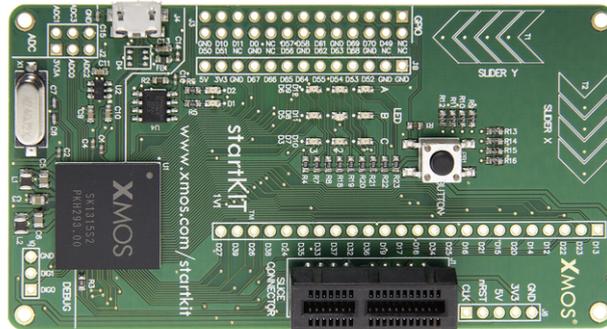
Figure 1: XMOS startKIT core module board setup

Once the flash memory on the xCORE startKIT has been appropriately programmed with this application note example, the LED's flash with the following patterns.

With the button on the xCORE startKIT in the up position when power is applied, the criss-cross LED pattern flashes on and off.
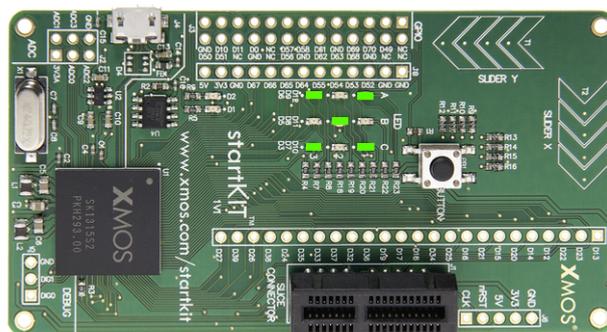
Figure 2: XMOS startKIT core module board button up boot

With the button on the xCORE startKIT in the down position when power is applied, the diamond LED pattern flashes on and off.
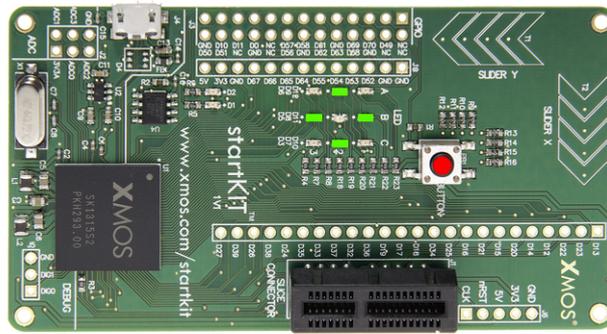
Figure 3: XMOS startKIT core module button down boot

APPENDIX B - Launching the example applications

B.1   Building the example application projects

Change directory into the top level `src` directory:

```
> cd src/
```

The top level `Makefile` builds the 4 example application projects and their dependancies. This can be launched from the command line using the tool xmake:

```
> xmake
```

Once built there is a `bin` directory within each application project which contains the binary with a XMOS standard .xe extension for that application project.

B.2   Building the loader functions

In order to build the loader functions described within this application note and defined in the file `loader/loader.xc` the XMOS `xcc` tool is used on the command line with the following options:

```
> xcc -c loader/loader.xc -o loader/loader.o
```

The option -c causes the xcc tool to compile and assemble the input file without linking.

B.3   Writing the default applications to the flash device

From the command line the `xflash` tool is used to download the code to the flash device on the xCORE startKIT module board. The XFLASH option `--factory` is used to specify the default factory image with a version number of 0. As the file `loader.xc` specifies that the button in the up position boots all even number images then the application project `app_factory_image_button_up` is used with the `--factory` option. Only one application can be specified with the XFLASH option `--factory`, all other applications must use the XFLASH option `--upgrade` and specify the version number. The application project `app_factory_image_button_down` is used with the XFLASH option `--upgrade` and has an odd version number of 1. The XFLASH option `--loader` is used to include the compiled loader functions when XFLASH creates the flash loader. The complete XFLASH command line is as follows:

```
> xflash --target=STARTKIT --factory app_factory_image_button_up/bin/app_factory_image_button_up.xe --upgrade
  ↪ 1 app_factory_image_button_down/bin/app_factory_image_button_down.xe --loader loader/loader.o
```

Once XFLASH has sucessfully programmed the flash memory device on the startKIT module board, the xCORE device is reset. The default application to boot should therefore be the image for the button up position. In this instance the LED's should start flashing a criss cross pattern.

Remove the power from the startKIT module board. Now press and hold the button on the device whilst applying power to the board. This time the device should boot the default image for the button in the down position. In this instance the LED's should start flashing in a diamond pattern.

B.4   Writing the upgrade applications to the flash device

In this application note example the command line tool `xflash` is used to create and download the upgrade images onto the xCORE startKIT module board.

As the file `loader.xc` specifies that the button in the up position boots all even number images then the application project `app_upgrade_image_button_up` is used with the `--upgrade` option and has an even version number of 2. The application project `app_upgrade_image_button_down` is used with the

--upgrade option and has an odd version number of 3. The complete XFLASH command line is as follows:

```
> xflash --target=STARTKIT --factory app_factory_image_button_up/bin/app_factory_image_button_up.xe --upgrade
   ↪ 1 app_factory_image_button_down/bin/app_factory_image_button_down.xe --upgrade 2
   ↪ app_upgrade_image_button_up/bin/app_upgrade_image_button_up.xe --upgrade 3 app_upgrade_image_button_down
   ↪ /bin/app_upgrade_image_button_down.xe --loader loader/loader.o
```

Once XFLASH has sucessfully programmed the flash memory device on the startKIT module board, the xCORE device is reset. The default application to boot should therefore be the upgrade image for the button up position. In this instance the LED's should start transitioning a criss cross pattern between on and off. The effect is a much smoother fade in and out of the LED's.

Remove the power from the startKIT module board. Now press and hold the button on the device whilst applying power to the board. This time the device should boot the upgrade image for the button in the down position. In this instance the LED's should start transitioning a a diamond pattern between on and off. The effect is a much smoother fade in and out of the LED's.

In normal practice in-field upgrade would be integrated into the application running on the xCORE. In this case XFLASH is used to create an upgrade image that is written to a binary file, and the application is responsible for writing the upgrade images to flash memory (e.g. using the libflash library). The complete XFLASH command line might look something as follows in this instance:

```
> xflash --target=STARTKIT --upgrade 2 app_upgrade_image_button_up/bin/app_upgrade_image_button_up.xe --
   ↪ upgrade 3 app_upgrade_image_button_down/bin/app_upgrade_image_button_down.xe --factory-version 13.2 -o
   ↪ my_upgrade_images.bin
```

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

XMOS startKIT Hardware Guide

http://www.xmos.com/published/startkit-hardware-manual

APPENDIX D - Full source code listing

D.1   Source code for loader.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>

/* Port for button on startKIT board. */
in port p_button = XS1_PORT_32A;

/* Enum for representing button state. */
enum button_val
{
  BUTTON_UP,
  BUTTON_DOWN
};

/* Store the button position. */
int buttonPosition;

/* Enum for representing the potential interest in the image. */
enum interest
{
  NOT_INTERESTED = 0,
  INTERESTED = 1
};

/* Store the version of the image in memory that will potentially be booted. */
int candidateImageVersion = -1;

/* Store the address of the image in memory that will potentially be booted. */
unsigned candidateImageAddress;

void init(void)
{
  unsigned buttonVal;

  /* Read state of button. */
  p_button :>  buttonVal;

  /* Button is up. */
  if((buttonVal & 1) == 1)
  {
    buttonPosition = BUTTON_UP;
  }
  /* Button is down. */
  else
  {
    buttonPosition = BUTTON_DOWN;
  }
}

int checkCandidateImageVersion(int imageVersion)
{
  /* If the button is up and imageVersion is even and imageVersion is higher */
  /* than the last candidateImageVersion then this is a potential version */
  /* for booting. */
```

```
  if ((buttonPosition == BUTTON_UP) &&
      ((imageVersion % 2) == 0) &&
      (imageVersion > candidateImageVersion))
  {
    return INTERESTED;
  }
  /* If the button is down and imageVersion is odd and imageVersion is higher */
  /* than the last candidateImageVersion then this is a potential version */
  /* for booting. */
  else if ((buttonPosition == BUTTON_DOWN) &&
           ((imageVersion % 2) != 0) &&
           (imageVersion > candidateImageVersion))
  {
    return INTERESTED;
  }

  /* Not a potential firmware image in all other cases. */
  return NOT_INTERESTED;
}

void recordCandidateImage(int imageVersion, unsigned imageAddress)
{
  /* Save the imageVersion that we are interested in. */
  candidateImageVersion = imageVersion;
  /* Save the imageAddress of the imageVersion that we are interested in. */
  candidateImageAddress = imageAddress;
}

unsigned reportSelectedImage(void)
{
  /* Return the candidateImageAddress of the image that we are interested in. */
  return candidateImageAddress;
}
```

D.2 Source code for project app_factory_image_button_down

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include "startkit_gpio.h"

// This function is combinable - it can share a core with other tasks.
[[combinable]]
static void glow(client startkit_led_if leds)
{
  timer tmr;
  int period = 1 * 1000 * 1000 * 100; // period from off to on = 1s;
  int level = LED_OFF;                 // the level of led brightness;
  unsigned pattern = 0b010101010;      // the pattern output to the leds;
  int timestamp;                       // holds the current timestamp;

  // Take the initial timestamp of the 100Mhz timer.
  tmr :> timestamp;

  while (1)
  {
```

```
    select
    {
    // After 'delay' ticks do this.
    case tmr when timerafter(timestamp + period) :> void:
      if (level == LED_ON)
      {
        level = LED_OFF;
      }
      else
      {
        level = LED_ON;
      }
      // set the leds.
      leds.set_multiple(pattern, level);
      // update the timestamp for the next timeout.
      timestamp += period;
      break;
    }
  }
}

//Required ports to drive the GPIO on the startKIT.
startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4A, XS1_PORT_4B, XS1_CLKBLK_1};

// 'main' sets up the system, consisting of two tasks - one to drive
// the i/o and one to run the application that communicates with that driver.
int main()
{
  // These interface connections link the application to the gpio driver.
  startkit_led_if i_led;
  par
  {
    on tile[0].core[0]: startkit_gpio_driver(i_led, null,
                                             null, null,
                                             gpio_ports);
    on tile[0].core[1]: glow(i_led);
  }
  return 0;
}
```

D.3 Source code for project app_factory_image_button_up

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include "startkit_gpio.h"

// This function is combinable - it can share a core with other tasks.
[[combinable]]
static void glow(client startkit_led_if leds)
{
  timer tmr;
  int period = 1 * 1000 * 1000 * 100; // period from off to on = 1s;
  int level = LED_OFF;                // the level of led brightness;
  unsigned pattern = 0b101010101;     // the pattern output to the leds;
```

```
  int timestamp;                       // holds the current timestamp;

  // Take the initial timestamp of the 100Mhz timer.
  tmr :> timestamp;

  while (1)
  {
    select
    {
    // After 'delay' ticks do this.
    case tmr when timerafter(timestamp + period) :> void:
      if (level == LED_ON)
      {
        level = LED_OFF;
      }
      else
      {
        level = LED_ON;
      }
      // set the leds.
      leds.set_multiple(pattern, level);
      // update the timestamp for the next timeout.
      timestamp += period;
      break;
    }
  }
}


startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4A, XS1_PORT_4B, XS1_CLKBLK_1};

// 'main' sets up the system, consisting of two tasks - one to drive
// the i/o and one to run the application that communicates with that driver.
int main()
{
  // These interface connections link the application to the gpio driver.
  startkit_led_if i_led;
  par
  {
    on tile[0].core[0]: startkit_gpio_driver(i_led, null,
                                             null, null,
                                             gpio_ports);
    on tile[0].core[1]: glow(i_led);
  }
  return 0;
}
```

D.4   Source code for project app_upgrade_image_button_down

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include "startkit_gpio.h"

// This function is combinable - it can share a core with other tasks.
```

```
[[combinable]]
static void glow(client startkit_led_if leds)
{
  timer tmr;
  int period = 1 * 1000 * 1000 * 100; // period from off to on = 1s;
  unsigned res = 30;                   // increment the brightness in this
                                       // number of steps;
  int delay = period / res;            // how long to wait between updates;
  int level = 0;                       // the level of led brightness;
  unsigned pattern = 0b010101010;      // the pattern output to the leds;
  int timestamp;                       // holds the current timestamp;
  int dir = 1;                         // step increase in led brightness;

  // Take the initial timestamp of the 100Mhz timer.
  tmr :> timestamp;

  while (1)
  {
    select
    {
    // After 'delay' ticks do this.
    case tmr when timerafter(timestamp + delay) :> void:
      // increase the output level of the led.
      level += dir * (LED_ON / res);
      if (level > LED_ON)
      {
        level = LED_ON;
        dir = -1;
      }
      if (level < 0)
      {
        level = 0;
        dir = 1;
      }
      // set the leds.
      leds.set_multiple(pattern, level);
      // update the timestamp for the next timeout.
      timestamp += delay;
      break;
    }
  }
}


startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4A, XS1_PORT_4B, XS1_CLKBLK_1};

// 'main' sets up the system, consisting of two tasks - one to drive
// the i/o and one to run the application that communicates with that driver.
int main()
{
  // These interface connections link the application to the gpio driver.
  startkit_led_if i_led;
  par
  {
    on tile[0].core[0]: startkit_gpio_driver(i_led, null,
                                             null, null,
                                             gpio_ports);
```

```
    on tile[0].core[1]: glow(i_led);
  }
  return 0;
}
```

D.5 Source code for project app_upgrade_image_button_up

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include "startkit_gpio.h"

// This function is combinable - it can share a core with other tasks.
[[combinable]]
static void glow(client startkit_led_if leds)
{
  timer tmr;
  int period = 1 * 1000 * 1000 * 100; // period from off to on = 1s;
  unsigned res = 30;                  // increment the brightness in this
                                      // number of steps;
  int delay = period / res;           // how long to wait between updates;
  int level = 0;                      // the level of led brightness;
  unsigned pattern = 0b101010101;     // the pattern output to the leds;
  int timestamp;                      // holds the current timestamp;
  int dir = 1;                        // step increase in led brightness;

  // Take the initial timestamp of the 100Mhz timer.
  tmr :> timestamp;

  while (1)
  {
    select
    {
    // After 'delay' ticks do this.
    case tmr when timerafter(timestamp + delay) :> void:
      // increase the output level of the led.
      level += dir * (LED_ON / res);
      if (level > LED_ON)
      {
        level = LED_ON;
        dir = -1;
      }
      if (level < 0)
      {
        level = 0;
        dir = 1;
      }
      // set the leds.
      leds.set_multiple(pattern, level);
      // update the timestamp for the next timeout.
      timestamp += delay;
      break;
    }
  }
}
```

```
startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4A, XS1_PORT_4B, XS1_CLKBLK_1};

// 'main' sets up the system, consisting of two tasks - one to drive
// the i/o and one to run the application that communicates with that driver.
int main()
{
  // These interface connections link the application to the gpio driver.
  startkit_led_if i_led;
  par
  {
    on tile[0].core[0]: startkit_gpio_driver(i_led, null,
                                             null, null,
                                             gpio_ports);
    on tile[0].core[1]: glow(i_led);
  }
  return 0;
}
```