Remote Camera Trigger

Final Report

Van Bruns, Jr and Nazreen Rusli

Table of Contents

Introduction

For ELEC3607: Embedded Systems, Semester 2 of 2010, we were required to design and implement an embedded system using an LPCXpresso board. As a team, group 10 selected the topic of a remote camera trigger. The area of high-speed photography provides many challenges that can be handled through a sophisticated embedded system, and we hoped to be able to work through some of these challenges.

Problem Definition

A remote camera trigger is a system that has a sensor to detect when a picture should be taken, and responds by sending the appropriate signal to a camera to take the picture. In our case, we chose a motion detection sensor, which signals a camera through an infrared emitter. When the system is in picture mode, and the sensor detects movement, the system signals the camera to capture the moving object.

Scope and Limitations

We initially wanted to create a system that would capture a picture of something like the umbrella-shaped water splash on the cover page. Such a system would require a very fast-response, very accurate motion sensor for the appropriate calculations to decide the timing of the picture. Due to the cost limits of the project, we had to use a sensor that was neither fast nor accurate, and as a result the project was scoped to detecting movement and responding by taking a picture based on a given delay.

Design

As with most embedded systems, the design can be broken into two parts: the hardware and software designs.

Hardware

The circuit for the remote camera trigger is comprised of the following components:

- LPC 1343 main board
- LPC 1343 daughter board
- 5V power supply
- VTE1163H infrared emitting diode
- 220Ω Resistor
- XL-MaxSonar-EZ MB1240 (sensor)

A layout of these components can be found in Appendix A. The sensor is powered by the 5V supply, as is the main board. The IR emitting diode is powered by one of the pulse-width modulated (PWM) ports from the main board, which outputs 5V. This first must pass through the resistor so we can step down the voltage to 3V, which is what is required for the diode. The analog output from the sensor is routed into one of the analog-to-digital converter (ADC) ports.

The main purpose for using the LPC 1343 daughter board is to provide the push buttons, seven-segment displays (SSD), and LED indicators. One of the push buttons is used to toggle between the two modes of the program: set delay mode and picture mode. Two push buttons are used to increment and decrement the delay when in set delay mode. The SSD are used to display the delay value. One of the LED indicators is used to show when we're in picture mode, indicating that movement detected by the sensor will trigger the camera to take a picture.

To complete the system, we need a camera to receive the IR signal. We've selected the Nikon D90 with remote trigger option, since it was available for us to use. The signal needed to trigger this camera will be detailed in the next section.

Software

The function of the software is outlined using pseudo code.

Initialization

```
void initialize()
{
   <initialize for delays>
   <initialize for PWM output of frequency 38.4kHz>
   <initialize for daughter board>
   <initialize for ADC input from sensor>
   <initialize DB display>
}
```

RF Signal

```
void irSignalOn(int delayInMicroseconds)
{
   <start PWM>
   delay(delayInMicroseconds);
}

void irSignalOff(int delayInMicroseconds)
{
   <stop PWM>
   delay(delayInMicroseconds);
}
```

Picture Capture Signal

```
void takePicture()
{
   // When the signal is on, it is actually a square wave of
   // frequency 38.4kHz. The argument into irSignalOn and irSignalOff
   // is how long to delay in microseconds. Total time to send signal
   // is 135.58millisec.
   irSignalOn(2000);
   irSignalOff(27830);
   irSignalOn(390);
   irSignalOff(1580);
   irSignalOn(410);
   irSignalOff(3580);
   irSignalOn(400);
   irSignalOff(63200);
   irSignalOn(2000);
   irSignalOff(27830);
   irSignalOn(390);
   irSignalOff(1580);
   irSignalOn(410);
   irSignalOff(3580);
   irSignalOn(400);
   irSignalOff(0);
}
```

Sensor Capture

```
int detectADInput()
{
   if (AD input has changed) return 1;

   return 0;
}
```

Main Loop

```
int main(void)
{
   initialize();

   while(1)
   {
      if (mode toggle button pressed) toggleMode();
      if (decrement button pressed && inSetDelayMode()) delay--;
      if (increment button pressed && inSetDelayMode()) delay++;
      if (inPictureMode() && detectADInput())
      {
         takePicture();
         toggleMode();
      }
   }
}
```

Implementation and Problems Encountered

Like the design, the implementation has been broken into hardware and software. We again refer to Appendix A for the hardware schematic when connecting all the components. Any pictures referenced can be found in Appendix B. The code for the software can be found in Appendix C.

Hardware

Once the LPC 1343 board has been supplied from a 5V power source, it can power the rest of the components of the system. While debugging the software, the board is actually powered through the USB, but we'll cover that in the software section. Without power from the USB, we must connect 5V in and ground.

With the daughter board connected, we found the best place for this to be from the top of the main board. There, you should see eight pairs of connectors somewhat in the centre of the board. Looking at the remote camera trigger picture, you should see the 5V in connected as a yellow wire. Ground could be connected much in the same way using the connectors just below the 5V in. We suggest wiring these to the main board now, but leaving them disconnected from the power source. Just make sure the loose ends don't touch any other wires or components when using the USB as a power source.

Before connecting the main board with the daughter board, we need to connect wires to the daughter board for ground and TX. There is a set of three connectors on the side of the daughter board opposite of the SSD that provide GND, RX, and TX. As seen in the remote camera trigger picture, the yellow wire is TX and the red wire is GND.

Now, connect the main board with the daughter board. The pins from the main board match up precisely with connectors from the daughter board. Make sure to have the USB cable connector on the main board oriented on the same side as the SSD on the daughter board, as can be seen in the remote camera trigger picture.

Before mounting the sensor on the breadboard, it is easier to attach some wires to its connectors. Add wires to the following connectors: V+, GND, and 3. V+ will be supplied with 5V, GND will go to ground, and pin 3 is the analog output and will go to an ADC.

The remaining components should now be connected. We used a basic breadboard as a base for connecting all our components for the system. This allowed us to break down our system when finished with the project. Lay out the resistor, IR emitting diode, and sensor. Then, connect the daughter board pins from underneath the board into the breadboard. Use some wire to anchor the end of the breadboard onto the peg legs of the daughter board. This is just to provide some stability.

Only thing missing now are the connections between the components. Connect the TX wire from the daughter board to the resistor. It doesn't matter which end of the resistor. Connect the other end of the resistor to the anode (positive) terminal of the diode. This will be the terminal furthest away from the small tab on the side of the diode's metal base. Connect the cathode (negative) terminal of the diode to GND. Connect the V+ wire from the sensor to the 5V wire coming from the top of the main board. The GND wire should obviously go to ground. Connect the pin-3 wire to AD6, which should be labelled as PIO1_10. Starting from the GND pin, which is closest to the end with the set of connectors GND, RX, and TX, P1-10 should be pin 13 if GND is pin 1.

It was difficult for us to find a useable ADC with the daughter board connected. With it connected, most of the pins become occupied by the daughter board's components that are adding the extra functionality it provides. Luckily, we were able to find AD6 was not being used by the component it is connected to.

Software

It is assumed that the reader of this report has sufficient knowledge of how to setup projects in LPCXpresso. For an explanation of project setup, we refer you to Appendix D for a full explanation. The rest of the software implementation discussion will focus on how we implemented it.

The first thing we needed to do in the code was to initialize all the setting for the main board. Knowing how to setup the settings came from thorough exploration of the user manual, which can be found on NXP Semiconductor's website. We give a brief explanation of how it was setup.

For initDelay and initPWM from timer.c, the clock for CT32B0 and CT32B1 needed to be turned on in SYSAHBCLKCTRL. PWM output required that we configure PIO1_7, which is the same as TX, in LPC_IOCON. For initDB in daughter_board.c, multiple settings in IOCON had to be changed to avoid interference with the GPIO ports. For projectInit in main.c, multiple settings had to be changed in IOCON, SYSCON, and ADC. These can be summarized as the following:

- PIO1_10 needed to be set to AD6, with a resistor mode of inactive, and an input mode of analog.
- The ADC clock had to be enabled in SYSAHBCLKCTRL and power needed to be given to ADC through PDRUNCFG.
- The ADC had to be setup to use AD6 in BURST hardware scan mode.

With everything initialized, we stepped through each piece, adding the needed implementation. We added daughter board code based on previous labs, which provided the SSD, push buttons, and an LED. We added timer code, again based on previous labs, which used CT32B1 for the delays and CT32B0 for the PWM output. The timing was changed from milliseconds to microseconds to provide the needed timing resolution. The rest important part of the implementation was then left to main.c.

We created two modes for the program: SET_DELAY_MODE and PICTURE_MODE. This allows a way to differentiate when the system is in each of these states. With only two states, we added a toggle method that would switch the mode and turn on an LED when in PICTURE_MODE.

The implementation for irSignalOn, irSignalOff, and takePicture basically looks like the pseudo code. Including the code from timer provides PWM and delay functions. A delay was added at the beginning of takePicture to incorporate the delay input by the push buttons and displayed on the SSD. The displayed delay on the SSD is multiplied by a factor of 50 milliseconds.

The method detectADInput was implemented to indicate when the input has changed. adValue stores the reading from a call to the method, so the value is used to compare with the current input to decide if it has changed. The value can also be used after a method call when the specific value is needed for calculations.

The main loop was setup with the following priority:

1. Check push button 1: Toggle mode when detected
2. Check push button 3: Delay decremented when detected
3. Check push button 4: Delay incremented when detected
4. Check for change in AD input: Take picture when detected
5. Delay a small amount of time then loop again.

Testing Methodology

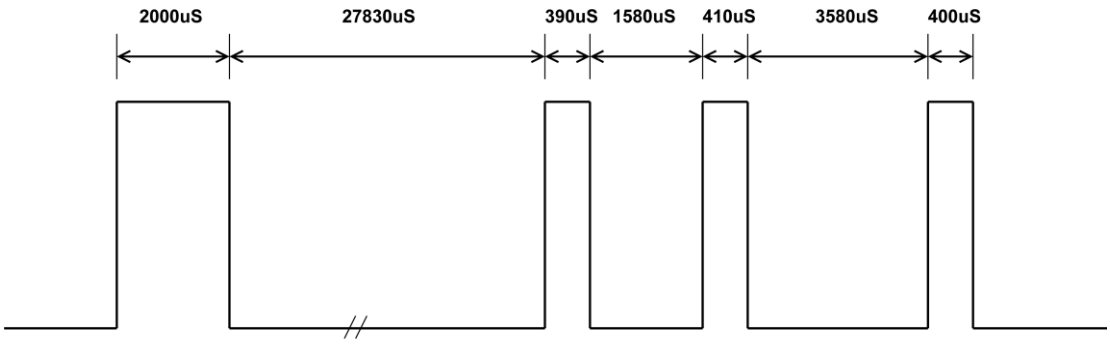
When testing the program, it was important to verify each part of the program was working before the whole system was tested. We suggest the following:

1. Test the daughter board functions, including SSD, push buttons, and LED.
2. Test the IR signal.
3. Test the sensor output.

Being in debug mode, we verified that PB1 switched program mode and changed the LED, PB3 decremented the SSD, and PB4 incremented the SSD. Also in debug mode, the value of the sensor output was sent to the console to verify how it responds to movement. This is when we figured out our sensor was not going to work for our original design. We tested the IR signal as follows.

IR Signal

A portion of the signal required to trigger the camera to take a picture is shown on the next page. This portion is repeated twice before the camera fires. Using an oscilloscope, the PWM output from TX should be measured and checked against this waveform. It is important to note that in diagram, when the signal goes high, it is actually oscillating between high and low at a frequency of 38.4 kHz.



Complete System

The complete system should be tested as follows:

1. Verify we start in SET_DELAY_MODE. In this mode, PB3 will decrement the delay stopping at 00, PB4 will increment the delay stopping at 99, and L4 should not be lit.
2. Verify PB1 switches mode. By pressing PB1, we should go into PICTURE_MODE, which means PB3 and PB4 won't affect the delay and L4 becomes lit.
3. Verify the sensor triggers the camera to take a picture when in PICTURE_MODE. Make sure there is no motion in front of the sensor when going into PICTURE_MODE, or the camera might fire right away.

Results

A collection of the photos taken when walking in front of the sensor can be found below.

Potential Improvements

The quality of the sensor really limited what we could do with the system. If we had a sensor that has a faster response time and is much more accurate, we could have setup the system like our original design, where we would drop an object in front of the sensor. Based on the change in distance as the object passes through the sensor's detection cone, we could calculate the time it would take for the object to drop at particular distance, snapping a picture right as the object hits the water.

Based on kinematics and geometry, we could use the following equations to solve for the time it will take for the object to drop from the sensor's horizon to the water, removing the dependence on using a delay:

$$h_1 = \sqrt{{d_1}^2 - {d_2}^2}$$

$$v_1 = \frac{h_1}{t_1} + \frac{1}{2} * g t_1$$

$$t_2 = \frac{-v_1 + \sqrt{{v_1}^2 + 2gh_2}}{g}$$

It may be of use to note that all values in these equations are positive and only the positive root is used for t_2. The last equation is a combination of the quadratic equation and the kinematic equation for height. A diagram of this system can be found in Appendix A.

Another thing that could be improved is the distance required between the camera and the remote camera trigger. As the system is now, the camera can't be more than 30 cm from the trigger. To increase this distance, we'd have to use a more powerful IR emitter.

Conclusion

In this project, we have designed and implemented a system using an LPCXpresso board. Implementing the remote camera trigger became difficult when we realized our sensor wasn't sufficient for our initial design. Even though the system ended up being less sophisticated than we had originally wanted, it still provided the mechanism for remote-triggering a Nikon D90 camera, and we learned much from the whole experience.

Appendices

A: Design

Schematic Diagram

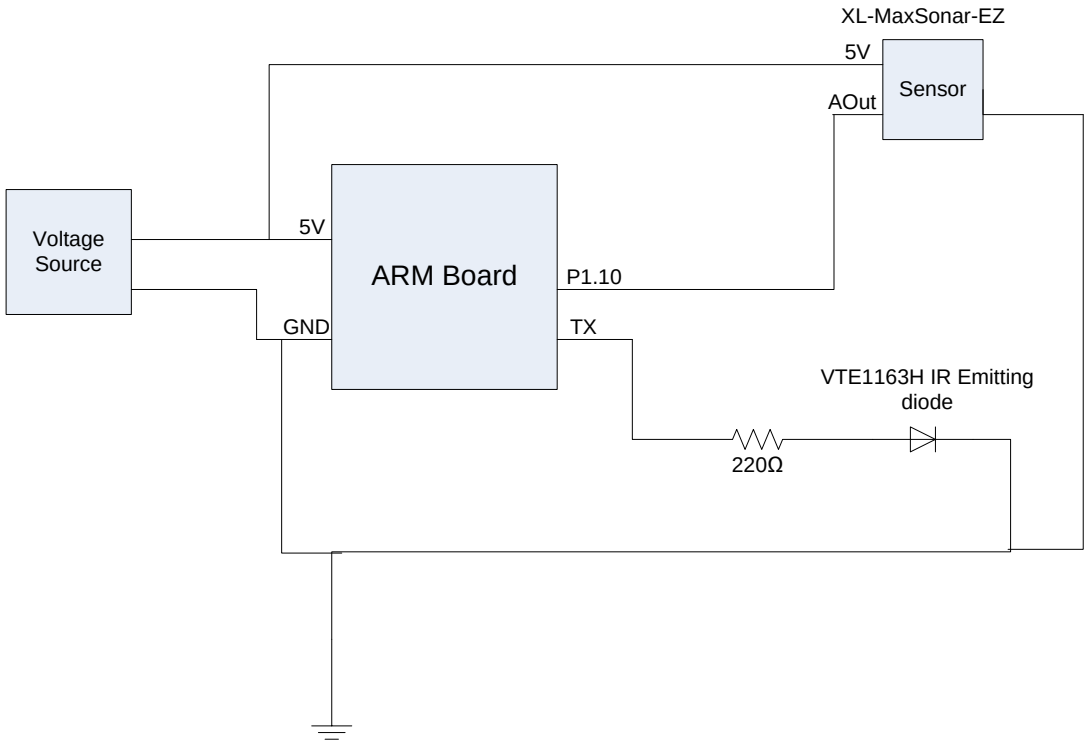
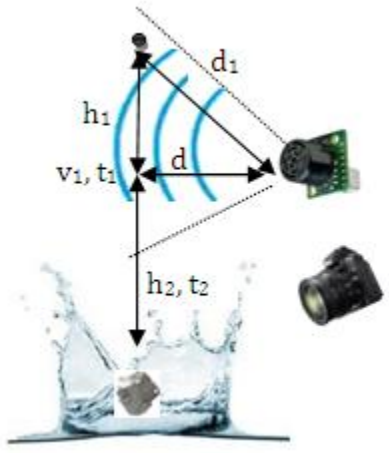


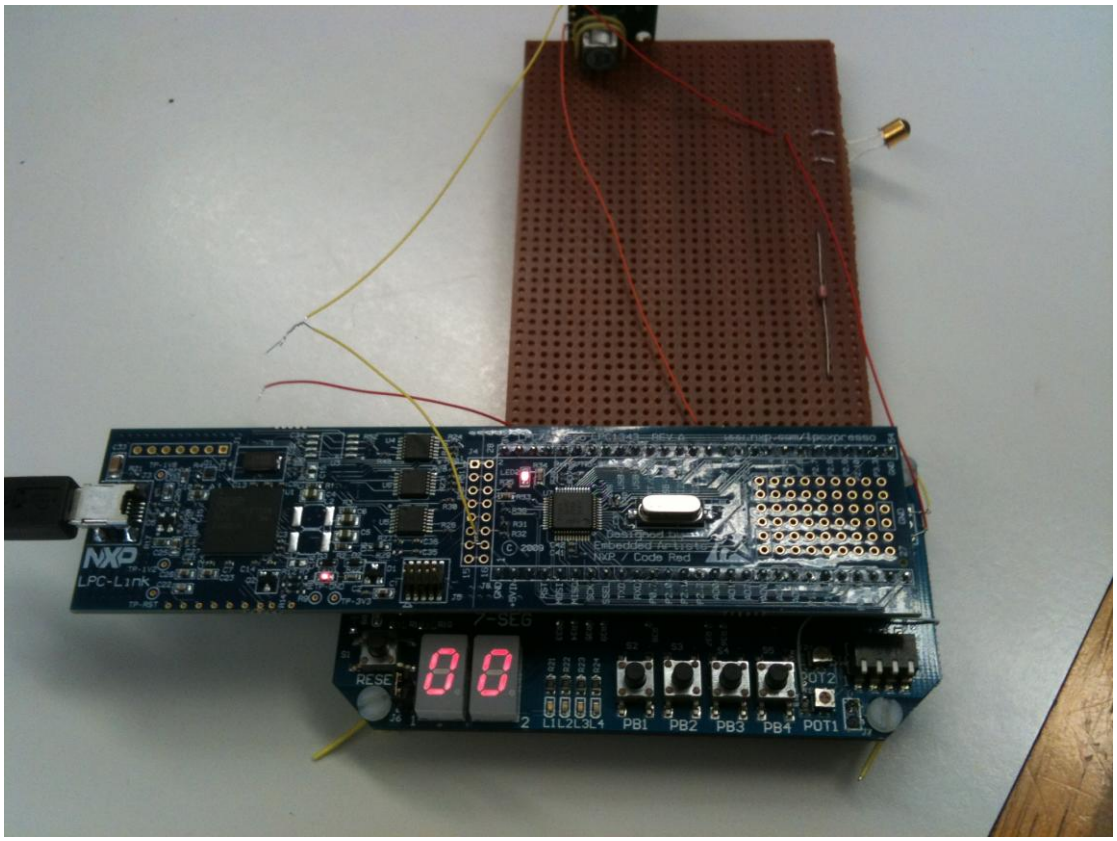
Diagram for Potential Design



B: Setup Pictures

Remote Camera Trigger

You can see the final product for the remote camera trigger below. Note that when connected to a 5V power supply, the USB cable would be disconnected, the yellow wires would be connected to 5V, and the red wire would be connected to ground.



Trigger + Camera

Notice the infrared emitter is pointed at the IR sensor on the camera.

Camera

The Nikon D90 with remote trigger option.

Remote Control Camera Setting

This is the setting required for the camera to receive the signal from the remote camera trigger.

Infrared Camera Sensor

This is the infrared sensor on the camera that receives the signal from the remote camera trigger. The IR emitter on the trigger must be pointed at this.

C: Software

The following lists the code developed during this project. The GPIO files are courtesy of NXP Semiconductor.

main.c

```
/*
===============================================================================
 Name        : main.c
 Author      : Van Bruns, Jr and Nazreen Rusli
 Version     : Alpha 0.0
 Copyright   : (C) Copyright
 Description : This is the main loop of our Remote Camera Trigger program.
```

```
                It contains an initialisation method and some convenience methods.
================================================================================
*/

#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

#include "timer.h"
#include "daughter_board.h"
#include "gpio.h"

// Modes of operation.
#define SET_DELAY_MODE 0
#define PICTURE_MODE 1

// Range of delay offset.
#define DELAY_OFFSET_MIN 0
#define DELAY_OFFSET_MAX 99

int programMode = SET_DELAY_MODE;
int pictureOffset = DELAY_OFFSET_MIN;
uint32_t adValue = 0;

// Toggles the program mode, going from either set delay mode to picture mode
// or picture mode to set delay mode. Sets LED to show when we're in picture mode.
void toggleProgramMode()
{
   programMode = (programMode == SET_DELAY_MODE) ? PICTURE_MODE : SET_DELAY_MODE;
   GPIOSetValue(LED_PORT, LED_BIT, programMode);
}

// Outputs PWM to the IR diode for the given amount of time.
void irSignalOn(int delayInMicroseconds)
{
   pwmOn();
   delay(delayInMicroseconds, 0);
}

// Turns the IR diode off for the given amount of time.
void irSignalOff(int delayInMicroseconds)
{
   pwmOff();

   // Our waveform was off by a factor of 288 cycles, so that number of cycles
   // was removed from the delay. Not sure where the extra time was coming from.
   delay(delayInMicroseconds, 288);
}

// Triggers the camera to take a picture.
void takePicture()
{
   // If there is a picture offset, then delay by the offset
   // with a factor of 50millisec.
   if (pictureOffset)
   {
      delay(pictureOffset * 50000, 0);
   }

   // When the signal is on, it is actually a square wave of frequency 38.4kHz.
   // The argument into irSignalOn and irSignalOff is how long to delay in
microseconds.
   // Total time to send signal is 135.58msec.
   irSignalOn(2000);
   irSignalOff(27830);
   irSignalOn(390);
   irSignalOff(1580);
   irSignalOn(410);
   irSignalOff(3580);
   irSignalOn(400);
   irSignalOff(63200);
   irSignalOn(2000);
   irSignalOff(27830);
   irSignalOn(390);
   irSignalOff(1580);
   irSignalOn(410);
```

```
   irSignalOff(3580);
   irSignalOn(400);
   irSignalOff(50000);
}

// Returns whether AD input was detected.
// The actual AD value will be set into adValue.
int detectADInput()
{
   int detected = 0;
   uint32_t input;

   // The input is bits 6-15 of AD6.
   input = (LPC_ADC->DR6>>6) & 0x3FF;

   // If this is not the first time setting adValue
   // and the new value if different from the previous,
   // then input was detected.
   if ((adValue != 0) && (adValue != input))
      detected = 1;

   adValue = input;

   return detected;
}

// Initialises the system for detecting AD input from the sensor
// and triggering the camera.
void projectInit()
{
   initDelay();
   initPWM(26, 13); // 38.4kHz
   initDB();

   LPC_IOCON->PIO1_10 |=  0x01; // Select function AD6 for 1.10
   LPC_IOCON->PIO1_10 &= ~0xC0; // Mode = Inactive (00 = no resistor)
   LPC_IOCON->PIO1_10 &= ~0x80; // Analog input mode (0 for analog input mode)

   LPC_SYSCON->SYSAHBCLKCTRL |= 0x2000; // Enable clock for ADC
   LPC_SYSCON->PDRUNCFG &= ~0x10; // Power on ADC (bit 4 -> 0 for power on)

   LPC_ADC->CR &= ~0x07000000; // START = No Start (000)
   LPC_ADC->CR &= ~0x000000FF; // Clear select bits
   LPC_ADC ->CR |= 0x00000040; // Select AD6 to be converted
   LPC_ADC->CR  |= 0x00010000; // BURST = Hardware scan mode (1)
   LPC_ADC->CR  |= 0x01000000; // START = Start conversion now (001)

   setSSDValue(pictureOffset); // Set SSDs based on default
   GPIOSetValue(LED_PORT, LED_BIT, programMode); // Set LED to display default mode
}

// Main loop.
int main(void)
{
   projectInit();

   while(1)
   {
      // Check if PB1 is pressed and debounce if it is.
      // If it is pressed for the full debounce, switch modes.
      if (detectPB(PB1))
      {
         if (debouncedPBDetect(PB1) == PB_PRESSED)
         {
            toggleProgramMode();
            delay(300000,0); // Delay by 300milliseconds for extra debounce

            // If we've moved into picture mode, call detectADInput once to read
            // in the current AD value and effectively "eat" the first detection.
            // After taking a picture, and changing back into picture mode, we
            // always seemed to immediately detect input. This fixes that problem.
            if (programMode == PICTURE_MODE)
            {
               detectADInput();
            }
         }
```

```
      }
      // Check if PB3 is pressed and if the offset is not at min, then debounce.
      // If it is pressed for the full debounce, decrement offset.
      else if (detectPB(PB3)
            && (programMode == SET_DELAY_MODE)
            && (pictureOffset != DELAY_OFFSET_MIN))
      {
         if (debouncedPBDetect(PB3) == PB_PRESSED)
         {
            pictureOffset--;
            setSSDValue(pictureOffset);
         }
      }
      // Check if PB4 is pressed and if the offset is not at max, then debounce.
      // If it is pressed for the full debounce, increment offset.
      else if (detectPB(PB4)
            && (programMode == SET_DELAY_MODE)
            && (pictureOffset != DELAY_OFFSET_MAX))
      {
         if (debouncedPBDetect(PB4) == PB_PRESSED)
         {
            pictureOffset++;
            setSSDValue(pictureOffset);
         }
      }
      // If we've switched to picture mode.
      else if (programMode == PICTURE_MODE)
      {
         // Take a picture if AD input detected.
         if (detectADInput())
         {
            takePicture();
            toggleProgramMode();
            delay(500000,0);
         }
      }
      // Else, just delay and let's loop again.
      else
      {
         delay(10, 0);
      }
   }

   return 0 ;
}
```

daughter_board.h

```
/*
===============================================================================
 Name        : daughter_board.h
 Author      : Van Bruns, Jr and Nazreen Rusli
 Version     : Alpha 0.0
 Copyright   : (C) Copyright
 Description : This defines functions and constants for daughter_board.c.
===============================================================================
*/

#ifndef DAUGHTER_BOARD_H_
#define DAUGHTER_BOARD_H_

// Ports for the push buttons.
#define PB1_PORT 0
#define PB2_PORT 0
#define PB3_PORT 3
#define PB4_PORT 3

// Bits for the push buttons.
#define PB1_BIT 6
#define PB2_BIT 10
#define PB3_BIT 0
#define PB4_BIT  1

// LED constants.
#define LED_PORT 1
#define LED_BIT 8
#define LED_ON 1  // Output of 1 means LED on.
```

```
#define LED_OFF 0  // Output of 0 means LED off.

// Seven-segment display constants.
#define ONES_DIGIT 0
#define TENS_DIGIT 1
#define SSDCOUNT 2
#define SSDSIZE 7
#define DIGSIZE 10 // number of digits (0-9)

// Constants representing the push buttons used in debouncedPBDetect.
#define PB1 1
#define PB2 2
#define PB3 3
#define PB4 4

// Return values for debouncedPBDetect.
#define PB_ERROR -1
#define PB_NOT_PRESSED 0
#define PB_PRESSED 1

// Initialises the needed daughter board components.
void initDB();

// Displays the given value on the two SSDs.
// Value must comply with: 0 <= value <= 99
// Returns a value of -1 if out of the needed range.
int setSSDValue(int value);

// Detects whether the given push button is pressed.
int detectPB(int button);

// Detects whether the given push button is pressed.
// button should be one of the following: PB1, PB2, PB3, or PB4.
// Returns either PB_PRESSED, PB_NOT_PRESSED, or PB_ERROR.
int debouncedPBDetect(int button);

#endif /* DAUGHTER_BOARD_H_ */
```

daughter_board.c

```
/*
============================================================================
 Name        : daughter_board.c
 Author      : Van Bruns, Jr and Nazreen Rusli
 Version     : Alpha 0.0
 Copyright   : (C) Copyright
 Description : This implements functions needs for the daughter board.
============================================================================
*/

#include "LPC13xx.h"
#include "gpio.h"
#include "timer.h"
#include "daughter_board.h"

// This table represents the on/off values for the segments of each digit.
int ssdtab[DIGSIZE][SSDSIZE] =
    {{1, 1, 1, 1, 1, 1, 0},
    {0, 1, 1, 0, 0, 0, 0},
    {1, 1, 0, 1, 1, 0, 1},
    {1, 1, 1, 1, 0, 0, 1},
    {0, 1, 1, 0, 0, 1, 1},
    {1, 0, 1, 1, 0, 1, 1},
    {1, 0, 1, 1, 1, 1, 1},
    {1, 1, 1, 0, 0, 0, 0},
    {1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 0, 1, 1}};

// This table represents the ports for the two SSDs.
int ssdport[SSDCOUNT][SSDSIZE] = {{0, 0, 2, 0, 0, 1, 0},
                                  {0, 0, 0, 0, 1, 1, 2}};

// This table represents the bits for the two SSDs.
int ssdbit[SSDCOUNT][SSDSIZE] = {{9, 8, 11, 2, 7, 0 ,11},
                                 {1, 3, 5, 4, 9, 11, 6}};

// Initialises the needed daughter board components.
```

```
void initDB()
{
   int j;

   GPIOInit();

   // Initialise LPC_IOCON registers so they don't interfere with the GPIO ports.
   LPC_IOCON->PIO0_1 = 0xD0;
   LPC_IOCON->PIO0_4 = 0;
   LPC_IOCON->PIO0_5 = 0;
   LPC_IOCON->PIO2_6 = 0;
   LPC_IOCON->JTAG_TMS_PIO1_0 |= 1;
   LPC_IOCON->JTAG_TDI_PIO0_11 &= ~0x07;
   LPC_IOCON->JTAG_TDI_PIO0_11 |= 0x01;

   // Initialise all segments as outputs.
   for (j = 0; j < SSDSIZE * SSDCOUNT; j++)
      // (j / SSDSIZE) tells us which SSD, (j % SSDSIZE) tells us which segment.
      GPIOSetDir(ssdport[j / SSDSIZE][j % SSDSIZE], ssdbit[j / SSDSIZE][j % SSDSIZE],
1);

   // Initialise PB1, PB3, and PB4 as inputs.
   GPIOSetDir(PB1_PORT, PB1_BIT, 0);
   GPIOSetDir(PB3_PORT, PB3_BIT, 0);
   GPIOSetDir(PB4_PORT, PB4_BIT, 0);

   // Initialise the LED for output.
   GPIOSetDir(LED_PORT, LED_BIT, 1);
}

// Uses the tables and given value and digit to "decode" into the right SSD output.
void ssddecode(int value, int digit)
{
   int j;

   for (j = 0; j < SSDSIZE; j++)
      GPIOSetValue(ssdport[digit][j], ssdbit[digit][j], ssdtab[value][j]);
}

// Displays the given value on the two SSDs.
// Value must comply with: 0 <= value <= 99
// Returns a value of -1 if out of the needed range.
int setSSDValue(int value)
{
   if ((value > 99) || (value < 0))
      return -1;

   ssddecode(value / DIGSIZE, TENS_DIGIT);
   ssddecode(value % DIGSIZE, ONES_DIGIT);

   return 0;
}

// Detects whether the given push button is pressed.
// button should be one of the following: PB1, PB2, PB3, or PB4.
// debounce is only internally used.
// Returns either PB_PRESSED, PB_NOT_PRESSED, or PB_ERROR.
int commonPBDetect(int button, int debounce)
{
   LPC_GPIO_TypeDef *gpio;
   int bit;
   int i = 0;

   // Set the appropriate gpio and bit for the given button.
   switch (button)
   {
      case PB1:
         bit = PB1_BIT;
         gpio = LPC_GPIO0;
         break;
      case PB2:
         bit = PB2_BIT;
         gpio = LPC_GPIO0;
         break;
      case PB3:
         bit = PB3_BIT;
```

```
            gpio = LPC_GPIO3;
            break;
         case PB4:
            bit = PB4_BIT;
            gpio = LPC_GPIO3;
            break;
         default:
            return PB_ERROR;
      }

      // If we just want to detect whether pressed without debounce.
      if (!debounce)
         return !gpio->MASKED_ACCESS[(1<<bit)];

      do
      {
         // If the gpio bit is 0, increment.
         if (!gpio->MASKED_ACCESS[(1<<bit)])
            i++;
         // Else, not enough consecutive 0's and the button wasn't really pressed.
         else
            return PB_NOT_PRESSED;

         delay(10000, 0);
      }
      // Five consecutive 0's means the push button was pressed.
      while (i < 5);

      return PB_PRESSED;
   }

   // Detects whether the given push button is pressed.
   int detectPB(int button)
   {
      return commonPBDetect(button, 0);
   }

   // Detects whether the given push button is pressed.
   // button should be one of the following: PB1, PB2, PB3, or PB4.
   // Returns either PB_PRESSED, PB_NOT_PRESSED, or PB_ERROR.
   int debouncedPBDetect(int button)
   {
      return commonPBDetect(button, 1);
   }
```

gpio.h

```
/****************************************************************************
 *   gpio.h:  Header file for NXP LPC13xx Family Microprocessors
 *
 *   Copyright(C) 2008, NXP Semiconductor
 *   All rights reserved.
 *
 *   History
 *   2008.09.01  ver 1.00    Preliminary version, first Release
 *   2009.12.09  ver 1.05    Mod to use mask registers for GPIO writes + inlining (.h)
 *
****************************************************************************/
#ifndef __GPIO_H
#define __GPIO_H

#define PORT0     0
#define PORT1     1
#define PORT2     2
#define PORT3     3

void GPIO_IRQHandler(void);
void GPIOInit( void );
void GPIOSetInterrupt( uint32_t portNum, uint32_t bitPosi, uint32_t sense,
      uint32_t single, uint32_t event );
void GPIOIntEnable( uint32_t portNum, uint32_t bitPosi );
void GPIOIntDisable( uint32_t portNum, uint32_t bitPosi );
uint32_t GPIOIntStatus( uint32_t portNum, uint32_t bitPosi );
void GPIOIntClear( uint32_t portNum, uint32_t bitPosi );

static LPC_GPIO_TypeDef (* const LPC_GPIO[4]) = { LPC_GPIO0, LPC_GPIO1, LPC_GPIO2,
LPC_GPIO3 };
```

```
/*****************************************************************************
** Function name:    GPIOSetValue
**
** Descriptions:     Set/clear a bitvalue in a specific bit position
**                in GPIO portX(X is the port number.)
**
** parameters:       port num, bit position, bit value
** Returned value:      None
**
*****************************************************************************/
static __INLINE void GPIOSetValue( uint32_t portNum, uint32_t bitPosi, uint32_t bitVal
)
{
   LPC_GPIO[portNum]->MASKED_ACCESS[(1<<bitPosi)] = (bitVal<<bitPosi);
}

/*****************************************************************************
** Function name:    GPIOSetDir
**
** Descriptions:     Set the direction in GPIO port
**
** parameters:       port num, bit position, direction (1 out, 0 input)
** Returned value:      None
**
*****************************************************************************/
static __INLINE void GPIOSetDir( uint32_t portNum, uint32_t bitPosi, uint32_t dir )
{
   if(dir)
      LPC_GPIO[portNum]->DIR |= 1<<bitPosi;
   else
      LPC_GPIO[portNum]->DIR &= ~(1<<bitPosi);
}

#endif /* end __GPIO_H */
/*****************************************************************************
**                            End Of File
*****************************************************************************/
```

gpio.c

```
/*****************************************************************************
 *   gpio.c:  GPIO C file for NXP LPC13xx Family Microprocessors
 *
 *   Copyright(C) 2008, NXP Semiconductor
 *   All rights reserved.
 *
 *   History
 *   2008.07.20  ver 1.00    Preliminary version, first Release
 *   2009.12.09  ver 1.05    Mod to use mask registers for GPIO writes + inlining (.h)
 *
*****************************************************************************/
#include "LPC13xx.h"       /* LPC13xx Peripheral Registers */
#include "gpio.h"

/* ====================
 * CodeRed - Modified file to extract out interrupt handler related code,
 * which is really application project specific.
 * Set TIMER16_GENERIC_INTS to 1 to reenable original code.
 * ==================== */
//#define GPIO_GENERIC_INTS 1

#ifdef GPIO_GENERIC_INTS
volatile uint32_t gpio0_counter = 0;
volatile uint32_t gpio1_counter = 0;
volatile uint32_t gpio2_counter = 0;
volatile uint32_t gpio3_counter = 0;
volatile uint32_t p0_1_counter  = 0;
volatile uint32_t p1_1_counter  = 0;
volatile uint32_t p2_1_counter  = 0;
volatile uint32_t p3_1_counter  = 0;

/*****************************************************************************
** Function name:    PIOINT0_IRQHandler
**
** Descriptions:     Use one GPIO pin(port0 pin1) as interrupt source
**
```

```
** parameters:       None
** Returned value:      None
**
******************************************************************************/
void PIOINT0_IRQHandler(void)
{
  uint32_t regVal;

  gpio0_counter++;
  regVal = GPIOIntStatus( PORT0, 1 );
  if ( regVal )
  {
   p0_1_counter++;
   GPIOIntClear( PORT0, 1 );
  }
  return;
}

/******************************************************************************
** Function name:    PIOINT1_IRQHandler
**
** Descriptions:     Use one GPIO pin(port1 pin1) as interrupt source
**
** parameters:       None
** Returned value:      None
**
******************************************************************************/
void PIOINT1_IRQHandler(void)
{
  uint32_t regVal;

  gpio1_counter++;
  regVal = GPIOIntStatus( PORT1, 1 );
  if ( regVal )
  {
   p1_1_counter++;
   GPIOIntClear( PORT1, 1 );
  }
  return;
}

/******************************************************************************
** Function name:    PIOINT2_IRQHandler
**
** Descriptions:     Use one GPIO pin(port2 pin1) as interrupt source
**
** parameters:       None
** Returned value:      None
**
******************************************************************************/
void PIOINT2_IRQHandler(void)
{
  uint32_t regVal;

  gpio2_counter++;
  regVal = GPIOIntStatus( PORT2, 1 );
  if ( regVal )
  {
   p2_1_counter++;
   GPIOIntClear( PORT2, 1 );
  }
  return;
}

/******************************************************************************
** Function name:    PIOINT3_IRQHandler
**
** Descriptions:     Use one GPIO pin(port3 pin1) as interrupt source
**
** parameters:       None
** Returned value:      None
**
******************************************************************************/
void PIOINT3_IRQHandler(void)
{
  uint32_t regVal;
```

```
  gpio3_counter++;
  regVal = GPIOIntStatus( PORT3, 1 );
  if ( regVal )
  {
   p3_1_counter++;
   GPIOIntClear( PORT3, 1 );
  }
  return;
}
#endif //GPIO_GENERIC_INTS

/*****************************************************************************
** Function name:    GPIOInit
**
** Descriptions:     Initialize GPIO, install the
**                GPIO interrupt handler
**
** parameters:       None
** Returned value:      true or false, return false if the VIC table
**                is full and GPIO interrupt handler can be
**                installed.
**
*****************************************************************************/
void GPIOInit( void )
{
  /* Enable AHB clock to the GPIO domain. */
  LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);

#ifdef __JTAG_DISABLED
  LPC_IOCON->JTAG_TDO_PIO1_1  &= ~0x07;
  LPC_IOCON->JTAG_TDO_PIO1_1  |= 0x01;
#endif

  /* Set up NVIC when I/O pins are configured as external interrupts. */
  NVIC_EnableIRQ(EINT0_IRQn);
  NVIC_EnableIRQ(EINT1_IRQn);
  NVIC_EnableIRQ(EINT2_IRQn);
  NVIC_EnableIRQ(EINT3_IRQn);
  return;
}

/*****************************************************************************
** Function name:    GPIOSetInterrupt
**
** Descriptions:     Set interrupt sense, event, etc.
**                edge or level, 0 is edge, 1 is level
**                single or double edge, 0 is single, 1 is double
**                active high or low, etc.
**
** parameters:       port num, bit position, sense, single/doube, polarity
** Returned value:      None
**
*****************************************************************************/
void GPIOSetInterrupt( uint32_t portNum, uint32_t bitPosi, uint32_t sense,
         uint32_t single, uint32_t event )
{
  switch ( portNum )
  {
   case PORT0:
     if ( sense == 0 )
     {
      LPC_GPIO0->IS &= ~(0x1<<bitPosi);
      /* single or double only applies when sense is 0(edge trigger). */
      if ( single == 0 )
        LPC_GPIO0->IBE &= ~(0x1<<bitPosi);
      else
        LPC_GPIO0->IBE |= (0x1<<bitPosi);
     }
     else
      LPC_GPIO0->IS |= (0x1<<bitPosi);
     if ( event == 0 )
      LPC_GPIO0->IEV &= ~(0x1<<bitPosi);
     else
      LPC_GPIO0->IEV |= (0x1<<bitPosi);
   break;
```

```
    case PORT1:
      if ( sense == 0 )
      {
       LPC_GPIO1->IS &= ~(0x1<<bitPosi);
       /* single or double only applies when sense is 0(edge trigger). */
       if ( single == 0 )
         LPC_GPIO1->IBE &= ~(0x1<<bitPosi);
       else
         LPC_GPIO1->IBE |= (0x1<<bitPosi);
      }
      else
       LPC_GPIO1->IS |= (0x1<<bitPosi);
      if ( event == 0 )
       LPC_GPIO1->IEV &= ~(0x1<<bitPosi);
      else
       LPC_GPIO1->IEV |= (0x1<<bitPosi);
    break;
    case PORT2:
      if ( sense == 0 )
      {
       LPC_GPIO2->IS &= ~(0x1<<bitPosi);
       /* single or double only applies when sense is 0(edge trigger). */
       if ( single == 0 )
         LPC_GPIO2->IBE &= ~(0x1<<bitPosi);
       else
         LPC_GPIO2->IBE |= (0x1<<bitPosi);
      }
      else
       LPC_GPIO2->IS |= (0x1<<bitPosi);
      if ( event == 0 )
       LPC_GPIO2->IEV &= ~(0x1<<bitPosi);
      else
       LPC_GPIO2->IEV |= (0x1<<bitPosi);
    break;
    case PORT3:
      if ( sense == 0 )
      {
       LPC_GPIO3->IS &= ~(0x1<<bitPosi);
       /* single or double only applies when sense is 0(edge trigger). */
       if ( single == 0 )
         LPC_GPIO3->IBE &= ~(0x1<<bitPosi);
       else
         LPC_GPIO3->IBE |= (0x1<<bitPosi);
      }
      else
       LPC_GPIO3->IS |= (0x1<<bitPosi);
      if ( event == 0 )
       LPC_GPIO3->IEV &= ~(0x1<<bitPosi);
      else
       LPC_GPIO3->IEV |= (0x1<<bitPosi);
    break;
    default:
      break;
  }
  return;
}

/*****************************************************************************
** Function name:    GPIOIntEnable
**
** Descriptions:     Enable Interrupt Mask for a port pin.
**
** parameters:       port num, bit position
** Returned value:      None
**
*****************************************************************************/
void GPIOIntEnable( uint32_t portNum, uint32_t bitPosi )
{
  switch ( portNum )
  {
   case PORT0:
     LPC_GPIO0->IE |= (0x1<<bitPosi);
   break;
   case PORT1:
     LPC_GPIO1->IE |= (0x1<<bitPosi);
   break;
```

```
   case PORT2:
     LPC_GPIO2->IE |= (0x1<<bitPosi);
   break;
   case PORT3:
     LPC_GPIO3->IE |= (0x1<<bitPosi);
   break;
   default:
     break;
  }
  return;
}

/*****************************************************************************
** Function name:    GPIOIntDisable
**
** Descriptions:     Disable Interrupt Mask for a port pin.
**
** parameters:       port num, bit position
** Returned value:      None
**
*****************************************************************************/
void GPIOIntDisable( uint32_t portNum, uint32_t bitPosi )
{
  switch ( portNum )
  {
   case PORT0:
     LPC_GPIO0->IE &= ~(0x1<<bitPosi);
   break;
   case PORT1:
     LPC_GPIO1->IE &= ~(0x1<<bitPosi);
   break;
   case PORT2:
     LPC_GPIO2->IE &= ~(0x1<<bitPosi);
   break;
   case PORT3:
     LPC_GPIO3->IE &= ~(0x1<<bitPosi);
   break;
   default:
     break;
  }
  return;
}

/*****************************************************************************
** Function name:    GPIOIntStatus
**
** Descriptions:     Get Interrupt status for a port pin.
**
** parameters:       port num, bit position
** Returned value:      None
**
*****************************************************************************/
uint32_t GPIOIntStatus( uint32_t portNum, uint32_t bitPosi )
{
  uint32_t regVal = 0;

  switch ( portNum )
  {
   case PORT0:
     if ( LPC_GPIO0->MIS & (0x1<<bitPosi) )
      regVal = 1;
   break;
   case PORT1:
     if ( LPC_GPIO1->MIS & (0x1<<bitPosi) )
      regVal = 1;
   break;
   case PORT2:
     if ( LPC_GPIO2->MIS & (0x1<<bitPosi) )
      regVal = 1;
   break;
   case PORT3:
     if ( LPC_GPIO3->MIS & (0x1<<bitPosi) )
      regVal = 1;
   break;
   default:
     break;
```

```
  }
  return ( regVal );
}

/*****************************************************************************
** Function name:    GPIOIntClear
**
** Descriptions:     Clear Interrupt for a port pin.
**
** parameters:       port num, bit position
** Returned value:      None
**
*****************************************************************************/
void GPIOIntClear( uint32_t portNum, uint32_t bitPosi )
{
  switch ( portNum )
  {
   case PORT0:
     LPC_GPIO0->IC |= (0x1<<bitPosi);
   break;
   case PORT1:
     LPC_GPIO1->IC |= (0x1<<bitPosi);
   break;
   case PORT2:
     LPC_GPIO2->IC |= (0x1<<bitPosi);
   break;
   case PORT3:
     LPC_GPIO3->IC |= (0x1<<bitPosi);
   break;
   default:
     break;
  }
  return;
}

/*****************************************************************************
**                            End Of File
*****************************************************************************/
```

timer.h

```
/*
===============================================================================
 Name        : timer.h
 Author      : Van Bruns, Jr and Nazreen Rusli
 Version     : Alpha 0.0
 Copyright   : (C) Copyright
 Description : This defines functions and constants for timer.c.
===============================================================================
*/

#ifndef TIMER_H_
#define TIMER_H_

// Initialises the system so delays can be used.
void initDelay();

// Initialises for PWM output using the given period and off period.
void initPWM(uint32_t period, uint32_t offperiod);

// Initialises for prescaled PWM output using the given period, off period, and
prescale.
void initPWMPrescale(uint32_t period, uint32_t offperiod, uint32_t prescale);

// Waits (delays) until the given time has passed. Cycle offset can be used for
// situations where you want to account for cycles already run by previous code.
// microsec is in microseconds
// cycleOffset is just the number cycles to remove from the delay
void delay(uint32_t microsec, uint32_t cycle_offset);

// Turns the PWM output on.
void pwmOn();

// Turns the PWM output off.
void pwmOff();

#endif /* TIMER_H_ */
```

timer.c

```
/*
===============================================================================
 Name        : timer.c
 Author      : Van Bruns, Jr and Nazreen Rusli
 Version     : Alpha 0.0
 Copyright   : (C) Copyright
 Description : This implements the functions needs for using delays and PWM.
===============================================================================
*/

#include "LPC13xx.h"
#include "timer.h"

// Initialises the system so delays can be used.
void initDelay()
{
   // Enable the clock to CT32B1.
   LPC_SYSCON->SYSAHBCLKCTRL |= (1<<10);

   LPC_TMR32B1->PR = 0; // prescale value
   LPC_TMR32B1->TCR = 0x01; // start timer
}

// Initialises for PWM output using the given period and off period.
void initPWM(uint32_t period, uint32_t offPeriod)
{
   // Just use scale of one.
   initPWMPrescale(period, offPeriod, 1);
}

// Initialises for prescaled PWM output using the given period, off period, and
prescale.
void initPWMPrescale(uint32_t period, uint32_t offPeriod, uint32_t prescale)
{
   uint32_t periodCycles; // The number of cycles before stopping the clock.
   uint32_t offCycles; // The number of cycles before changing MR1.

   // Enable the clock to CT32B0.
   LPC_SYSCON->SYSAHBCLKCTRL |= (1<<9);

   // Initialise P1.7 for PWM output.
   LPC_IOCON->PIO1_7 &= ~0x07;
   LPC_IOCON->PIO1_7 |= 0x02;

   // Calculate the number of cycles for the period and off period.
   periodCycles = period * ((SystemCoreClock/LPC_SYSCON->SYSAHBCLKDIV) / 1000000);
   offCycles = offPeriod * ((SystemCoreClock/LPC_SYSCON->SYSAHBCLKDIV) / 1000000);

   LPC_TMR32B0->TCR = 0x02; // Reset the counter.
   LPC_TMR32B0->PC = prescale - 1; // Set the prescale value.
   LPC_TMR32B0->MR0 = periodCycles; // Set the number of cycles before stopping.
   LPC_TMR32B0->MR1 = offCycles; // Set the number of cycles before changing MR1.
   LPC_TMR32B0->MCR = 0x02; // Set MR0 to reset the clock.
   LPC_TMR32B0->PWMC |= 0x02; // Set PWM enabled for MR1.

   // Make sure PWM output is initially off.
   pwmOff();
}

// Waits (delays) until the given time has passed. Cycle offset can be used for
// situations where you want to account for cycles already run by previous code.
// microsec is in microseconds
// cycleOffset is just the number cycles to remove from the delay
void delay(uint32_t microsec, uint32_t cycleOffset)
{
   uint32_t cycles; // The number of cycles before stopping the clock.

   // Convert microseconds to cycles and offset by the given offset.
   cycles = microsec * ((SystemCoreClock/LPC_SYSCON->SYSAHBCLKDIV) / 1000000) -
cycleOffset;

   // If we have no cycles to delay, just return.
   if (cycles < 0)
      return;
```

```
    LPC_TMR32B1->TCR = 0x02; // Reset the counter.
    LPC_TMR32B1->MR2 = cycles; // Set the number of cycles before stopping.
    LPC_TMR32B1->MCR = 0x100; // Set MR2 to stop the clock.
    LPC_TMR32B1->TCR = 0x01; // Start the timer.

    while (LPC_TMR32B1->TCR & 0x01); // Wait until the clock has been stopped.
}

// Turns the PWM output on.
void pwmOn()
{
    LPC_TMR32B0->TCR = 0x01;
}

// Turns the PWM output off.
void pwmOff()
{
    LPC_TMR32B0->TCR = 0x02;
}
```

D: Project Setup

Open LPCXpresso v3.4 on your PC. For installation help and where to download this software, go to the website for NXP Semiconductors: http://ics.nxp.com

You should be prompted to select an area for your workspace, so provide your workspace folder. Create a new project, selecting the type "C Project" and clicking next. Select the type "NPX LPC1300 Hello World (semihosting)", provide a project name, and click next. Check the "Use CMSIS" checkbox and click next. You should be able to click next on the subsequent steps until you get to finish. Make sure "LPC1343" is selected and click finish.

Your new project should have been added. Now, select the ""Import from a project archive (zip) file" button from the toolbar. We need to import the example project "CMSISv1p30_LPC13xx", so browse and select the associated zip file for this project. The location will be dependent on your system configuration. Now, click finished to add the project.

Expand to the src folder of your project. There should be two files: cr_startup_lpc13.c and main.c. cr_startup_lpc13.c should never be modified. Open main.c and replace all the code with the provided main.c from Appendix C. Create a file for each of the other provided files.

The project should now be ready to be compiled and run, but we suggest phasing in the code piece by piece to make sure everything is working properly. This phasing will be covered in the testing methodology section.

E: Abbreviations

ADC – analog-to-digital converter
GND – ground
IR – infrared
LED – light-emitting diode
PB – push button
PWM – pulse-width modulated
SSD – seven-segment display

F: References

[1] "XL- MaxSonar®- EZ4™ (MB1240)",[Online]. Available:
http://www.maxbotix.com/uploads/MB1240-MB1340_Datasheet.pdf
[Accessed: Oct 26, 2010]

[2] PerkinElmer Inc.,"GaAIAs Infrared Emitting Diodes, TO-46 Lensed Package – 880nm", [Online]. Available:
http://www.perkinelmer.com/PDFS/downloads/dts_vte1163.pdf
[Accessed: Oct 26, 2010]

[3] "LPC1311/13/42/43 User manual", [Online]. Available:
http://ics.nxp.com/support/documents/microcontrollers/pdf/user.manual.lpc13xx.pdf
[Accessed: Oct 26, 2010]

[4] "LP1311/13/42/43", [Online] Available:
http://ics.nxp.com/products/lpc1000/datasheet/lpc1311.lpc1313.lpc1342.lpc1343.pdf
[Accessed: Oct 26, 2010]

[5] "IR Remote Control for Nikon", [Online] Available:
http://www.bigmike.it/ircontrol/
[Accessed: Oct 26, 2010]

[6] "Water Splash", [Online] Available:
http://homepage.mac.com/weepul/watersplash1.jpg
[Accessed: Oct 26, 2010]