SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department

Real-Time Embedded Systems - ECEN 121
Lab 1

Writing and debugging some simple C programs

Andrew Wolfe

Prelab (feel free to ignore for this lab report):

This week's prelab must be done individually by each student.

You need to refamiliarize yourself with the STM32L476VGT6 Reference manual and the other Cortex M4 and STM documentation. Review the documents and answer the following questions.

1. What bits in the RCC_AHB2ENR register must be set to allow use of GPIO ports B and E?
   On a Bits 0-31 scheme:
   Bit 1 and bit 4
   RCC_AHB2ENR_GPIOBEN EQU (0x00000002)
   RCC_AHB2ENR_GPIOBEN EQU (0x00000010)
2. What is the address of the RCC_AHB2ENR register?
   0x4002104c
3. Which GPIO registers need to be configured in order to properly configure the red and green LED output pins to control those LEDs?
   We need to configure GPIO MODER and GPIO ODR. GPIO OTYPER controls reset states and should be included but I do not think you NEED them for configuration. GPIO OSPEEDR and PUPDR are also needed but this was not written in our original prelab.
4. Which bits must be set and/or cleared in each of those GPIO registers in order to configure the read and greed LEDs respectively?
   On a Bits 0-31 scheme:
   CLEARED:
   In GPIO MODER: red led = 4th and 5th bit; green led = 16th and 17th bit
   SET:
   In GPIO MODER: red led = 4th bit; green led = 16th bit
   In GPIO ODR: red led = 2nd bit; green led = 8th bit

Source:

```
GPIO_MODER_MODER2       EQU (0x00000030)
GPIO_MODER_MODER2_0     EQU (0x00000010)
GPIO_MODER_MODER2_1     EQU (0x00000020)

GPIO_MODER_MODER8       EQU (0x00030000)
GPIO_MODER_MODER8_0     EQU (0x00010000)
GPIO_MODER_MODER8_1     EQU (0x00020000)

GPIO_ODR_ODR_2          EQU (0x00000004)
GPIO_ODR_ODR_8          EQU (0x00000100)
```

Prelabs must be turned in by the Sunday before lab at 5PM.

Lab Procedure:

You are going to write and debug a simple C program in the Keil development environment.

Create an ECEN-121 Projects directory **on your Z: drive. You must always reference project code through the Z: drive pathways.** Within that directory create a subdirectory called Barebones C. Copy the file "Minimal C template.zip" from the Lab Software directory on the Google Drive and unzip it into this location.

For simple C projects in Keil – you can rename a project using the following steps:

• Rename the directory. I renamed mine from Minimal C template to Lab1p1. • In the directory – you can rename the project file. I renamed mine from project to Lab1p1. • Also rename the project.uvoptxm file to match. Mine is now Lab1p1.uvoptx. • If the directories DebugConfig, Listings, Objects, and/or RTE exist, delete them. • In other situations, there may be a file called project.uvguix.username (in my case project.uvguix.awolfe) – you would want to rename that one as well. If you have one that has my username – you should delete it.

If you open this project with the renamed project file, you can look at the main.c file. This is your main file, and as always, you start your code by running main().

This project should build with no errors. Try it.

This is a very barebones project. The file stm32l476xx.h includes definitions that help

you to access the various I/O devices on the chip. This replaces the stm32l476xx_constants.s file that we used for the same purpose in our Assembly environment. The defined names are similar, but not identical. You will need to search this file in order to know what names have been assigned to device registers.

The file startup_stm32l476xx.s is the first code that runs on reset. It sets up the memory map, creates interrupt vectors and default handlers, copies ROM variables to RAM, and calls main(). It is identical to the startup file we used in our Assembly projects that ran on the Discovery board.

The remaining files, SysClock.h and SysClock.c include code to set the clock on our discovery board to 80MHz (instead of the 4MHz default we used in ELEN-120). This happens when main() calls System_Clock_Init() as its first step. Speed rules.

Add the following global variables to your program. These should be declared before main().

- An uninitialized array of type int called fiblist that can hold 25 elements. •
  A string called string1 that is initialized to “Hello”
- A string called string2 that is initialized to “World”
- An uninitialized array of type char called string3 that can hold 255 elements.

Now write code in main() (after System_Clock_Init()) that fills the array fiblist[] with the first 25 elements of the Fibonacci sequence. The elements begin with 0, 1, 1, 2, …

2

You may create additional global variables or additional local variables within main().

Build without errors then run in the debugger using Run To Cursor Line and Step. Open the memory view debug window and configure it to show type int in decimal form. Type fiblist into the address box. You should be able to observe your code filling in the Fibonacci sequence.

Reset the debugger. This time, step through your code while looking at the Call Stack + Locals window. You should be able to see any local variables, such as iterators, changing as you step through. Quit the debugger when you are done.

3

Next - add code to main() that uses string1 and string2 to place the string “Hello World” into string3. Remember:

- Each C string is NULL terminated. There cannot be a NULL in the middle of a string.
- Neither string1 nor string 2 includes a space. string3 will include one.

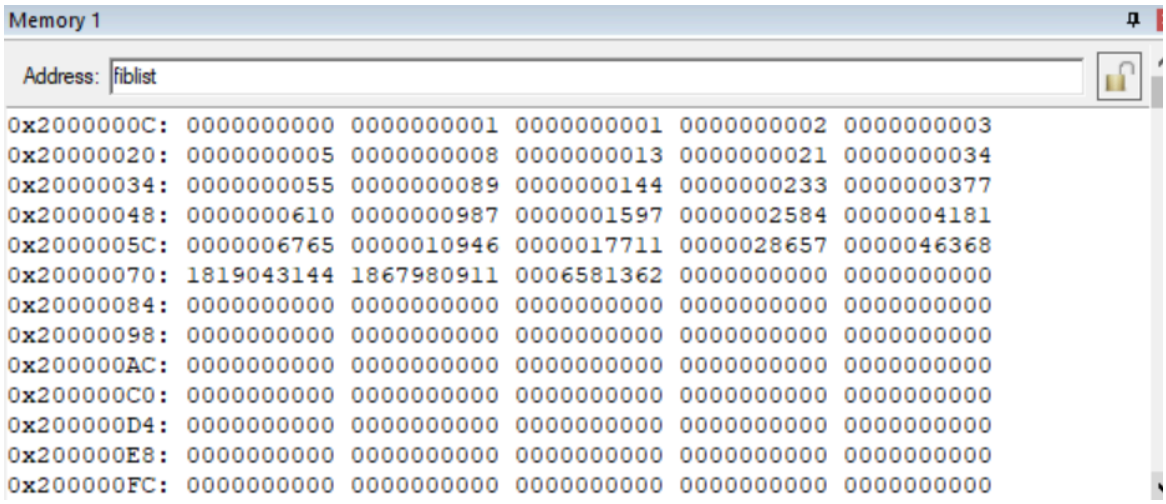
Once this code builds, debug it using the debugger. The memory window can be set to ASCII mode and type char. The memory window address can be set to string1, string2, or string3.

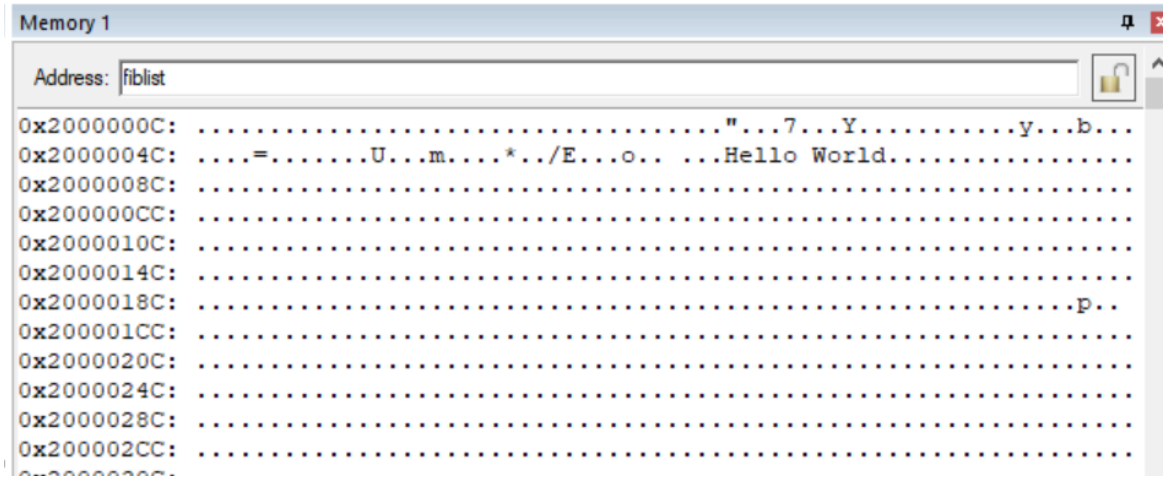
Problem 1 Code:

```
#include "stm32l476xx.h"
#include "SysClock.h"

int fiblist[25];
char string1[] = "Hello";
char string2[] = "World";
char string3[255];
int main(void){
	System_Clock_Init(); // Switch System Clock = 80 MHz
	int i;
	fiblist[0] = 0;
	fiblist[1] = 1;
	for(i=2; i < 25; i++)
	{
		fiblist[i] = fiblist[i-1] + fiblist[i-2];
	}
	i=0;
	while(string1[i] != 0)
	{
		string3[i]=string1[i];
		i++;
	}
	int j = 0;
	string3[i]= ' ';
	i++;
	while(string2[j] != 0)
	{
		string3[i] = string2[j];
		i++;
		j++;
	}
}
```

Screenshots:





Writing and debugging a barebones GPIO program in C

On the class Google Drive in the Examples directory – download the file: Blink – Subroutines.zip.

Unzip this directory into your Barebones C subdirectory. Open the project file in the directory. You should be able to build and download this project. It turns on both LEDs then flashes them.

You should be able to read and understand this program. It is one of the programs I wrote while working on ELEN-120 lab 3. It uses a bunch of subroutines to initialize the GPIO ports for the red and green LED, to turn them on and off, and to perform time delays.

You are going to write and debug a simple C program in the Keil development environment that duplicates the functionality of my program in C.

Inside the subdirectory called Barebones C that you created, copy the file "Minimal C template.zip" from the Lab Software directory on the Google Drive and unzip it into this location. Alternatively, you can start from a copy of the C program you wrote already. Rename the new project using the rules on page 1:

If you open this project with the renamed project file, you can look at the main.c file. This is your main file, and as always, you start your code by running main().

This project should build with no errors. Try it.

Now – you need to implement the functionality of my assembly-language blink program. Remember that register names may be different in C, but they can generally be determined by searching stm32l476xx.h.

You can include all of your subroutines in main.c or you can add a separate C file for the subroutines.

The first thing you need to do is write code to initialize the ports. This means you need to:

- Enable the port clocks for GPIOB and GPIOE
- Set the correct port bits to output mode
- Configure those bits to push-pull mode

You don't need to do this exactly the way I did – but you need one or more initialization subroutines that you call from main() that accomplish this configuration.

Next, write a subroutine called Red_LED_On() that turns on the red LED. Test it. Once it works, write and test Red_LED_Off() and Red_LED_Toggle().

Next, write a subroutine called Green_LED_On() that turns on the green LED. Test it. Once it works, write and test Green_LED_Off() and Green_LED_Toggle().

Next write a subroutine called delaymicro(unsigned int x) that will delay by x µs. You really cannot use my assembly code as a template since:

1. You are not writing assembly.
2. The clock rate is 80MHz on this app – mine was at 4MHz.

You will need some experimentation. See if you can make it accurate to within 5% at 500,000 µs.

If you just write a loop that does nothing, the C compiler may optimize it away. We went over how to avoid that in class.

Finally – use those routines to duplicate the functionality of my program. Demo this program to the TA.

In your lab report, explain how you determined the timing of the delaymicro() function.

In our main function, we looped the blinking continuously and tested several different multipliers. We settled on a multiplier of 10x where x is the number of microseconds inputted by the user. Using a video-timer, we observed 10 blinking iterations and we ended up getting times of 10.00, 10.08, and 10.04 seconds. Averaging these out and dividing by 10 gets us 1.004 which is well within the 5% error range (.95-1.05).

Problem 2 Code:

```
#include "stm32l476xx.h"
#include "SysClock.h"

void enablePortB(void)
{
	//RCC_AHB2ENR was added into stm32l476xx.h (4c)
	int *p = (int *)(RCC_BASE + RCC_AHB2ENR);
	int x = *p;
 x |= RCC_AHB2ENR_GPIOBEN;
	*p = x;
}

void b2ToPushPull(void)
{
	//GPIO_MODER was added into stm321476xx.h (0x0)
	int x = GPIOB_BASE;
	int y = GPIO_MODER_MODER2_0;
	int z = GPIO_MODER; // this is not needed because GPIOB_MODER = 0 but may want this
format for later
	x += z;
	z = *((int *)x);
	z |= y;
	y <<= 1;
	z &= ~y;
	*((int *)x) = z;
```

```
}

void enablePortE(void)
{
	//RCC_AHB2ENR was added into stm32l476xx.h (4c)
	int *p = (int *)(RCC_BASE + RCC_AHB2ENR);
	int x = *p;
 x |= RCC_AHB2ENR_GPIOEEN;
	*p = x;
}

void e8ToPushPull(void)
{
	//GPIO_MODER was added into stm321476xx.h (0x0)
	int x = GPIOE_BASE;
	int y = GPIO_MODER_MODER8_0;
	int z = GPIO_MODER; // this is not needed because GPIOB_MODER = 0 but may want this
format for later
	x += z;
	z = *((int *)x);
	z |= y;
	y <<= 1;
	z &= ~y;
	*((int *)x) = z;
}

void redOn(void)
{
	int *p = (int *)(GPIOB_BASE+GPIO_ODR);
	int x = *p;
 x |= GPIO_ODR_ODR_2;
	*p = x;
}

void greenOn(void)
{
	int *p = (int *)(GPIOE_BASE+GPIO_ODR);
	int x = *p;
 x |= GPIO_ODR_ODR_8;
	*p = x;
}

void redOff(void)
{
	int *p = (int *)(GPIOB_BASE+GPIO_ODR);
	int x = *p;
 x &= ~(GPIO_ODR_ODR_2);
	*p = x;
}

void greenOff(void)
{
	int *p = (int *)(GPIOE_BASE+GPIO_ODR);
	int x = *p;
 x &= ~(GPIO_ODR_ODR_8);
```

```
	*p = x;
}
void redTog(void)
{
	int *p = (int *)(GPIOB_BASE+GPIO_ODR);
	int x = *p;
  x ^= GPIO_ODR_ODR_2;
	*p = x;
}

void greenTog(void)
{
	int *p = (int *)(GPIOE_BASE+GPIO_ODR);
	int x = *p;
  x ^= GPIO_ODR_ODR_8;
	*p = x;
}

void delayMicro(unsigned int x)
{
	int j;
	for(j=0; j < 10*x; j++)
	{
		x++;
		if(x)
		{
			x--;
		}
	}
}

int main(void){
	System_Clock_Init(); // Switch System Clock = 80 MHz
	enablePortB();
	b2ToPushPull();
	enablePortE();
	e8ToPushPull();
	unsigned int x;
	redOn();
	greenOn();
	x = 5000000;
	delayMicro(x); //delay to show it turning off
	redOff();
	greenOff();
	delayMicro(x); //delay to show it turning off

	x = 500000; //delay of .5 sec in between turning off and on
	int k = 1;
	while(k)
	{
		k--;
		redTog();
		greenTog();
		delayMicro(x);
		k++;
```

```
        }
}
```

No Screenshots for Problem 2