ELEC-240 Lab2
Controlling GPIOs with Blocking Delays on the STM32F429 Nucleo-144 Development Board [*]

Stuart MacVeigh, Paul Davey

October 4, 2018

1 Introduction

This lab exercise introduces you to Loops and Conditional Loops in embedded systems. Loops have a variety of functions one of which is creating crude delays.

To generate a crude delay the programmer can command the micro-controller to perform a pointless task which takes up time before moving on to other relevant tasks. The easiest way to achieve this is to get the micro-controller to increment a variable from zero up to a pre-defined amount to generate the desired delay. The larger the pre-defined value, the longer it will take the micro-controller to count to that value hence a longer delay.

A **VERY** approximate equation for determining the pre-defined value is

$$Val = \frac{F_{APB}}{4} \times T_{Delay}$$

which will achieve roughly the right value. Depending on the timing specificness of the application this value can then be fine-tuned through trial and error.

Delays of this type are extremely crude as the actual time taken to count up to the pre-defined value can be affected by many factors such as how long it's taken to run the other tasks in the program where the delay is located and interrupt service routines that may be running which will put the delay loop on hold while they run.

1.1 Learning Outcomes

By the end of this lab exercise you should be able to:

1. Demonstrate an understanding of how blocking loops work
2. Create and edit blocking loops to achieve a desired time delay
3. Create subroutines of various types and demonstrate an understanding of their benefits
4. Demonstrate an understanding of masking and its importance when manipulating control register values

[*]document produced by Stuart MacVeigh

1.2 Code management

It is highly recommended to either download a fresh copy of the example code or to make a copy of the previous Lab example code before editing it for a new lab or task. This is highly advantageous as it allows you to retain older versions of the code (aka Revision History) which can be recalled in the event that you make any irreparable mistakes when editing for a new task.

Please keep this in mind throughout all Labs!

2 Masking

Peripherals on the micro-controller such as the GPIOs are controlled via special registers like the MODER register, which controls the pin function (input or output), and ODR register, which controls the pin state (high or low). Both of these registers appear in the example code.

Controlling the peripheral often involves manipulating only certain bits inside these registers which means simply overwriting the current value with a new one, using the '=' operator is incorrect as it will overwrite the states of the other bits which we do not want to change.

Therefore it is necessary to use Masking (ANDing and ORing) where the existing value in the register is "Masked" with a second value so as to set or clear only the relevant bits while leaving the others unchanged.

In embedded software the OR operator is '|' and the AND operator is '&'.

Some basic examples are:

- 0011b | 0100b = 0111b
- 0011b | 0010b = 0011b
- 0011b | 0000b = 0011b
- 0011b & 0001b = 0001b
- 0011b & 0111b = 0011b
- 0011b & 0100b = 0000b

3 Port Control

The STM32F429 family micro-controllers have two ways of controlling output GPIOs.

Writing to the 'ODR' register performs a 'Read-Modify-Write' where the contents of the register are read into the accumulator, the relevant bits are changed then the new value is written back to the register to control the GPIO pins. This method takes up a lot of instructions which can be inefficient for timing specific and high speed applications. Becuase of this there exists a 'BSRR' register which is simply written to and will update a GPIO pin using a single instruction.

Writing a '1' to a bit the lower half (bits 0→15) of the BSRR register will set the corresponding pin high; the bit in the register is then auto-cleared.

Writing a '1' to the same bit in the upper half (bits 16→ 31) of the BSRR register will clear that same pin; the bit in the register is then auto-cleared.

Task 1

Download the ‘LAB2 example code.zip’ from the DLE and extract to a local directory. Inside the extracted folder there are several source code files labelled ‘blinky1.c’ to ‘blinky5.c’. Blinky1.c contains a simple piece of code to flash the Green LED on the Nucleo-144 board. This code is also in its most basic form and while it will successfully build and run it contains many malpractices.

Blinky2.c to Blinky5.c show step by step how to identify and rectify these malpractices until eventually Blinky5.c contains the same piece of code but is now written to a professional level.

You can follow step by step how the code is improved by removing Blinky1.c and importing Blinky2.c then recompile and run, repeat this process until you reach Blinky5.c.

Task 2

Blinky5.c has shown how to control the Green LED with correctly formatted code.

Now edit the code so the Red and Blue LEDs replicate the behaviour of the Green LED. These are attached to pins PB7 and PB14. See Table 1 for a summary of the relevant registers involved in setting up and controlling the port pins.

Task 3

Create a second delay routine called “delay2” that uses a ‘while’ loop instead of a ‘for’ loop and test it. This will involve creating a new subroutine. Refer to Table 2 for information.

Task 4

Create a third delay routine called “delay3” that can be called with a variable (input argument) that defines the time of the delay. Test it. Refer to Table 2 for information.

Task 5

Write a program to send an SOS message in Morse code using the LEDs. Refer to Table 2 for Morse code information.

Register Name	**Reference Manual Section**
Hardware Clock Enable Register (AHB1ENR)	6.3.10
GPIO port mode register (MODER)	8.4.1
GPIO port output type register (OTYPER)	8.4.2
GPIO port output speed register (OSPEEDR)	8.4.3
GPIO port pull-up/pull-down register (PUPDR)	8.4.4
GPIO port input data register (IDR)	8.4.5
GPIO port output data register (ODR)	8.4.6
GPIO port bit set/reset register (BSRR)	8.4.7

Table 1: Table of relevant control registers for configuring and controlling GPIOs

4 Support Documentation

Document Name	**Contained Information**
UM1974 User manual	• Pin identification and the supported special functions • Circuit schematics • Jumper and component identification • Header pinouts
RM0090 Reference manual	• MCU memory and peripherals architecture • Peripheral control registers, addresses and bit-fields
Morse Code Information Page (Wiki) Morse Code Look-up chart	• Development and history of morse code • Complete morse Look-up chart for alphabetical characters • Brief overview of morse code timing
While Loop (Wiki)	Refer to the "C programming language" section
For Loop (Wiki)	Refer to the "Loop Counter" section for C-programming example
Subroutines in C	

Table 2: Table of relevant support documentation
(The document names are hyperlinks, please click on them to access the documents)