Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	**Embedded Systems Design**
Course Number:	**COE718**
Semester/Year (e.g.F2016)	**F2025**

Instructor:	**Gul Khan**

Assignment/Lab Number:	*Lab 2*
Assignment/Lab Title:	*Lab 2: Exploring Cortex-M3 Features for Performance Efficiency*

Submission Date:	**September 25, 2025**
Due Date:	**September 25, 2025**

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Grewal	**Arman**	**501100160**	**03**	**AG**

Table Of Contents

1.0 LAB OBJECTIVE

The objective of this lab is to implement and analyze bit banding, conditional execution, and barrel shifting on the NXP LPC1768 microcontroller using the Keil uVision IDE. The lab demonstrates how these ARM Cortex-M3 features can be applied to control LEDs and optimize performance. This process involves configuring memory aliasing, utilizing conditional branching methods, applying barrel shifter operations, and using debugging tools to evaluate execution times in both Debug and Target modes.

2.0 Calculations

For this lab, we were required to implement a bit-banding method to control the LEDs. To do this, the bit-band alias address for each LED pin had to be calculated. On the ARM Cortex-M3, the bit-band alias address is determined using the formula:

$$bit\_band\_alias\_address = 0x22000000 + (32 \times byte\_offset) + (4 \times bit\#)$$

Where 0x22000000 is the bit-band alias base address, bit# is the bit position (0-3) and byte offset is calculated as:

$$byte\_offset = target\_address - 0x20000000$$

Bit Banding Alias Calculations P1.28 (GPIO1, bit 28):

$$\begin{aligned} alias &= 0x22000000 + (32 \times (0x2009C034 - 0x20000000)) + (4 \times 28) \\ &= 0x22000000 + (32 \times 0x0009C034) + 0x70 \\ &= 0x22000000 + 0x01380680 + 0x70 = 0x233806F0 \end{aligned}$$

Bit Banding Alias Calculations P2.2 (GPIO2, bit 2):

$$\begin{aligned} alias &= 0x22000000 + (32 \times (0x2009C054 - 0x20000000)) + (4 \times 2) \\ &= 0x22000000 + (32 \times 0x0009C054) + 0x08 \\ &= 0x22000000 + 0x01330A80 + 0x08 = 0x23380A88 \end{aligned}$$

Thus, the bit banding alias addresses are $0x233806F0$ and $0x23380A88$ for P1.2 and P2.2 respectively.

3.0 Results

Bit Banding Exercise:

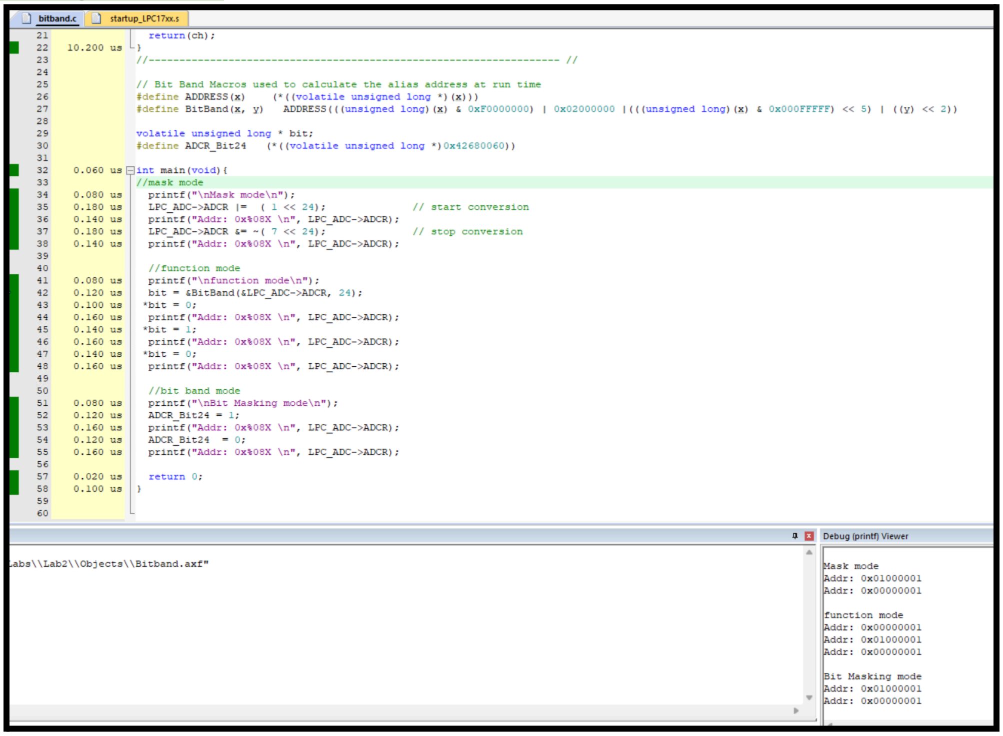

Figure 1.0: Bit banding exercise in debug mode with performance analyzer.

Mode	*Masking*	*Function*	*Direct Bit Band*
Time (us)	*0.36*	*0.5*	*0.24*

All these three modes all result in the same output however, there are varying times for all modes. It can be seen that the direct bit band is the fastest, followed by the masking and then function. The reason direct bit band is the fastest is because it allows one to directly write to one bit without having to read, change, and write back to a whole register (like bit masking). The only reason the function took the longest was the time it took to calculate the address.

Conditional Execution Exercise:

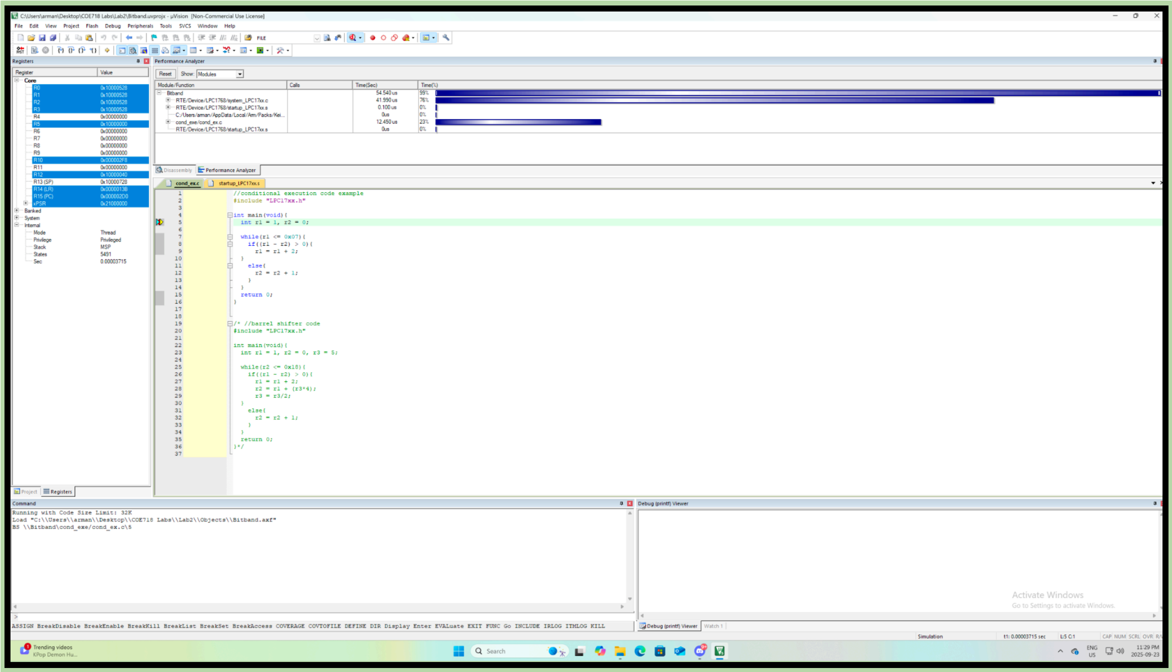

Figure 2.0: Example of Conditional Execution exercise in debug mode (time not accurate in this picture).

	Level 0	*Level 3*
Conditional Execution Time (us)	0.52	0.37us

It can be seen that the level three execution was much faster than level 0. The reason for this is because level three is known to optimize for time. This is done by using IT blocks. short if/else sequences are compiled into IT/ITE guarded instructions instead of branch/label pairs. Fewer taken branches result in faster execution.

Barrel Shifting Exercise:

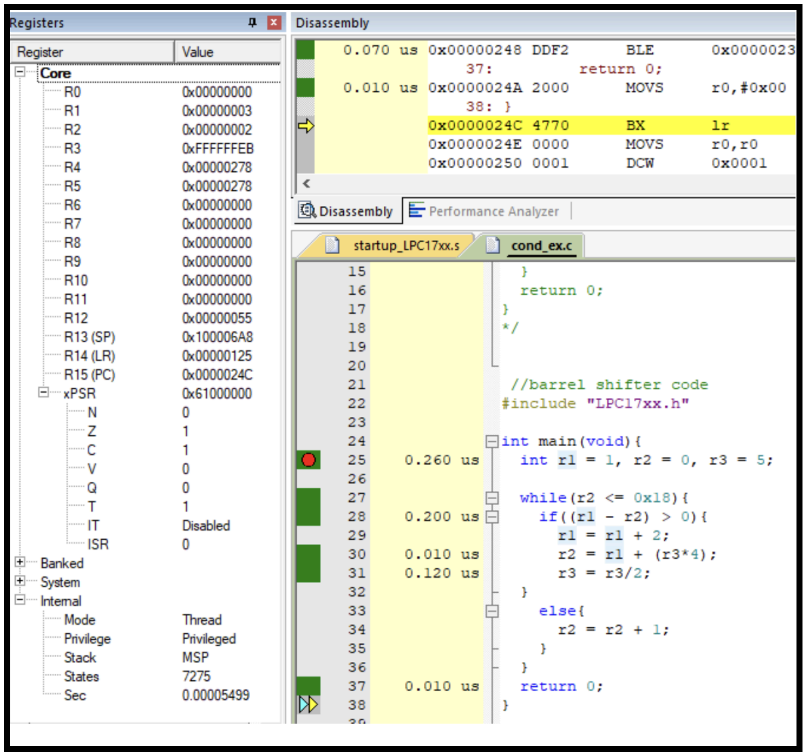

Figure 3.0: Barrel Shifting Exercise.

This exercise showed how the Cortex-M3's barrel shifter combines shifts and arithmetic in a single instruction, improving efficiency. It also demonstrated how compiler optimizations at level -O3 use both the barrel shifter and conditional execution to reduce instructions and speed up execution.

Lab Assignment Code:

Target (Demo) Version:

```
#include "LPC17xx.h"
#include <stdio.h>
#include "Board_LED.h"
#include "GLCD.h"

/* ---------------- Config ---------------- */
#define __FI        1               /* 16x24 font for GLCD */
#define __USE_LCD   1               /* set to 1 to enable LCD */
#define ROTATE_INTERVAL_MS  3000u     /* change mode every 3 seconds    */

/* ------- ITM (printf over SWO) ------- */
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))
#define DEMCR           (*((volatile unsigned long *)(0xE000EDFC)))
#define TRCENA          0x01000000
```

```
struct __FILE { int handle; };
FILE __stdout, __stdin;

int fputc(int ch, FILE *f) {
  if (DEMCR & TRCENA) { while (ITM_Port32(0) == 0); ITM_Port8(0) = (unsigned
char)ch; }
  return ch;
}

//------------------------------------------------------------------------- //

// Bit Band Macros used to calculate the alias address at run time
#define ADDRESS(x)      (*((volatile unsigned long *)(x)))
#define BitBand(x, y)   ADDRESS(((unsigned long)(x) & 0xF0000000) | 0x02000000 |
\
                        (((unsigned long)(x) & 0x000FFFFF) << 5) | ((y) << 2))

/* Hardcoded calculated LED ALIAS values */
#define LED1_ALIAS (*(volatile unsigned long *)0x233806F0)   /* P1.28 */
#define LED2_ALIAS (*(volatile unsigned long *)0x23380A88)   /*P2.2 */

/* Delay Fucntion */
static void delay_ms(int ms) {
  if (ms <= 0) return;
  for (volatile int i = 0; i < ms * 8000; ++i) { __NOP(); }
}

/* ======================== Auto-rotate via SysTick (10 ms)
======================== */

typedef enum { M_MASK, M_FUNC, M_DIRECT, M_BARREL, M_MAX } Mode;

static volatile Mode      g_mode = M_MASK;
static volatile uint8_t  g_tick10ms_flag = 0;                              /*
Flag that turns on every 10ms, causing program to double check moode we are in */

#define ROTATE_TICKS        (ROTATE_INTERVAL_MS / 10u)   /*
ROTATE_INTERVAL_MS, every 300 ticks of SysTick (i.e., every 3 seconds), the
mode changes */

void SysTick_Handler(void){
  static uint32_t ms10 = 0;
  g_tick10ms_flag = 1;
  if (++ms10 >= ROTATE_TICKS) {
    ms10 = 0;
```

```
    g_mode = (Mode)((g_mode + 1) % M_MAX);
    ITM_Port8(0) = '.';
  }
}

/* ---------------- LED methods (use simple if/else) ----------------
   NOTE: At -O3 the compiler typically emits an IT/ITE block for these if/else,
        so this satisfies the lab's "conditional execution" requirement.       */

static void led_Mask_mode(void) {
  printf("\nMask mode (if/else ? IT at -O3)\n");
  static int t = 0; t ^= 1;                  /* flip 0/1 each call */

  if (t == 0) LED_On(0);  else LED_Off(0);   /* LED index 0 */
  delay_ms(500);
  if (t == 0) LED_Off(3); else LED_On(3);    /* LED index 3 (opposite sense) */
  delay_ms(500);
}

static void led_function_mode(void) {
  printf("\nFunction bit-band (if/else ? IT at -O3)\n");
  static int t = 0; t ^= 1;

  volatile unsigned long *a1 = &BitBand(&LPC_GPIO1->FIOPIN, 28);  /* P1.28 alias
*/
  volatile unsigned long *a2 = &BitBand(&LPC_GPIO2->FIOPIN,  2);  /* P2.2  alias
*/

  if (t == 0) *a1 = 1; else *a1 = 0;         /* write 1/0 to alias */
  delay_ms(500);
  if (t == 0) *a2 = 0; else *a2 = 1;
  delay_ms(500);
}

static void led_direct_bitband_mode(void) {
  printf("\nDirect bit-band (if/else ? IT at -O3)\n");
  static int t = 0; t ^= 1;

  if (t == 0) LED1_ALIAS = 1; else LED1_ALIAS = 0;  /* P1.28 alias */
  delay_ms(500);
  if (t == 0) LED2_ALIAS = 0; else LED2_ALIAS = 1;  /* P2.2  alias */
  delay_ms(500);
}

/* ------------- Barrel shifter demo (LCD optional) ------------- */
static void method_barrel_shifter(void) {
```

```
  printf("\nBarrel shifter demo\n");
  uint8_t pattern = 0x01;
  int leds = (int)LED_GetCount();
  for (int i = 0; i < leds*2; ++i) {
    LED_SetOut((uint32_t)(pattern & ((1<<leds)-1)));
    delay_ms(250);
    pattern = (uint8_t)(pattern << 1);
    if (pattern == 0) pattern = 0x01;
  }
  LED_SetOut(0);
}

/*-----------------------------main--------------------------------*/

int main (void){
  __enable_irq();                /* ensure interrupts are on */

  LED_Initialize();
  printf("Init done\n");

  /* Set as Output */
  LPC_GPIO1->FIODIR |= (1U<<28);    /* P1.28 output */
  LPC_GPIO2->FIODIR |= (1U<<2);     /* P2.2  output */

  /* SysTick @ 10 ms via CMSIS system clock */
  SystemCoreClockUpdate();
  if (SysTick_Config(SystemCoreClock / 100)) {   /* 100 Hz -> 10 ms */
    printf("SysTick config ERROR\n");
    while (1) { /* trap */ }
  }

#if __USE_LCD
  /* -------- LCD init + static header -------- */
  GLCD_Init();
  GLCD_Clear(White);
  GLCD_SetBackColor(Blue);
  GLCD_SetTextColor(Yellow);
  GLCD_DisplayString(0, 0, __FI, (unsigned char*)"   COE718 Lab 2   ");
  GLCD_SetBackColor(White);
  GLCD_SetTextColor(Blue);
  GLCD_DisplayString(2, 0, __FI, (unsigned char*)"Mode:            ");
#endif

  Mode last = (Mode)0xFF;

  while (1) {
```

```
    /* optional: check 10 ms flag for housekeeping/prints */
    if (g_tick10ms_flag) {
      g_tick10ms_flag = 0;
      if (g_mode != last) {
        last = g_mode;
        /* Console (SWO) */
        switch (g_mode) {
          case M_MASK:   printf("Mode: Masking\n");         break;
          case M_FUNC:   printf("Mode: Function BitBand\n"); break;
          case M_DIRECT: printf("Mode: Direct BitBand\n");   break;
          case M_BARREL: printf("Mode: Barrel Shifter\n");   break;
          default: break;
        }
#if __USE_LCD
        /* LCD update */
        GLCD_SetTextColor(Red);
        switch (g_mode) {
          case M_MASK:   GLCD_DisplayString(2, 6, __FI, (unsigned char*)"Masking
"); break;
          case M_FUNC:   GLCD_DisplayString(2, 6, __FI, (unsigned char*)"Function
BB    "); break;
          case M_DIRECT: GLCD_DisplayString(2, 6, __FI, (unsigned char*)"Direct BB
"); break;
          case M_BARREL: GLCD_DisplayString(2, 6, __FI, (unsigned char*)"Barrel
Shifter: Using Logical Shift Left"); break;
        }
        GLCD_SetTextColor(Blue);
#endif
      }
    }

    /* Run the selected demo */
    switch (g_mode) {
      case M_MASK:   led_Mask_mode();           break;
      case M_FUNC:   led_function_mode();       break;
      case M_DIRECT: led_direct_bitband_mode();  break;
      case M_BARREL: method_barrel_shifter();    break;
      default: break;
    }

    /* Small idle*/
    for (volatile uint32_t i = 0; i < 50000; ++i) __NOP();
  }

  /* not reached */
  return -1;
```

```
}
```

Debug (Performance Analysis) Version:

```
#include "LPC17xx.h"
#include <stdio.h>
#include "Board_LED.h"

/* ---------------- Config ---------------- */
#define __FI        1                /* 16x24 font for GLCD */
#define __USE_LCD   1                /* set to 1 to enable LCD */
#define ROTATE_INTERVAL_MS  3000u     /* change mode every 3 seconds    */

/* ------- ITM (printf over SWO) ------- */
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))
#define DEMCR           (*((volatile unsigned long *)(0xE000EDFC)))
#define TRCENA          0x01000000

struct __FILE { int handle; };
FILE __stdout, __stdin;

int fputc(int ch, FILE *f) {
  if (DEMCR & TRCENA) { while (ITM_Port32(0) == 0); ITM_Port8(0) = (unsigned
char)ch; }
  return ch;
}

//------------------------------------------------------------------ //

// Bit Band Macros used to calculate the alias address at run time
#define ADDRESS(x)      (*((volatile unsigned long *)(x)))
#define BitBand(x, y)   ADDRESS(((unsigned long)(x) & 0xF0000000) | 0x02000000 |
\
                        (((unsigned long)(x) & 0x000FFFFF) << 5) | ((y) << 2))

/* Hardcoded calculated LED ALIAS values */
#define LED1_ALIAS (*(volatile unsigned long *)0x233806F0)   /* P1.28 */
#define LED2_ALIAS (*(volatile unsigned long *)0x23380A88)   /*P2.2 */

/* ---------------- LED methods (use simple if/else) ----------------
   NOTE: At -O3 the compiler typically emits an IT/ITE block for these if/else,
       so this satisfies the lab's "conditional execution" requirement.       */
```

```
static void led_Mask_mode(void) {
  printf("\nMask mode (if/else ? IT at -O3)\n");
  static int t = 0; t ^= 1;                /* flip 0/1 each call */

  if (t == 0) LED_On(0);  else LED_Off(0);   /* LED index 0 */
  if (t == 0) LED_Off(3); else LED_On(3);    /* LED index 3 (opposite sense) */
}

static void led_function_mode(void) {
  printf("\nFunction bit-band (if/else ? IT at -O3)\n");
  static int t = 0; t ^= 1;

  volatile unsigned long *a1 = &BitBand(&LPC_GPIO1->FIOPIN, 28);  /* P1.28 alias
*/
  volatile unsigned long *a2 = &BitBand(&LPC_GPIO2->FIOPIN,  2);  /* P2.2  alias
*/

  if (t == 0) *a1 = 1; else *a1 = 0;         /* write 1/0 to alias */
  if (t == 0) *a2 = 0; else *a2 = 1;
}

static void led_direct_bitband_mode(void) {
  printf("\nDirect bit-band (if/else ? IT at -O3)\n");
  static int t = 0; t ^= 1;

  if (t == 0) LED1_ALIAS = 1; else LED1_ALIAS = 0;  /* P1.28 alias */
  if (t == 0) LED2_ALIAS = 0; else LED2_ALIAS = 1;  /* P2.2  alias */
}

/* ------------- Barrel shifter demo (LCD optional) ------------- */
static void method_barrel_shifter(void) {
  printf("\nBarrel shifter demo\n");
  uint8_t pattern = 0x01;
  int leds = (int)LED_GetCount();
  for (int i = 0; i < leds*2; ++i) {
    LED_SetOut((uint32_t)(pattern & ((1<<leds)-1)));
    pattern = (uint8_t)(pattern << 1);
    if (pattern == 0) pattern = 0x01;
  }
  LED_SetOut(0);
}

/*------------------------------main---------------------------------*/

int main (void){
```

```
  __enable_irq();                /* ensure interrupts are on */

  LED_Initialize();
  printf("Init done\n");

  /* Set as Output */
  LPC_GPIO1->FIODIR |= (1U<<28);    /* P1.28 output */
  LPC_GPIO2->FIODIR |= (1U<<2);     /* P2.2  output */

  /* Run each method exactly once */
  led_Mask_mode();
  led_function_mode();
  led_direct_bitband_mode();
  method_barrel_shifter();

  printf("\nAll methods executed once. Idling...\n");

  while (1) { __NOP(); }

  // return -1;
}
```

4.0 Analysis

Debug (Performance Analysis):

Method	Execution Time (-O0) (us)	Execution Time (-O3) (us)	Performance Improvement (us)
Masking	1.06us	0.96us	0.10us
BitBand() Function	0.32us	0.21us	0.11us
Direct Bit Banding	0.32us	0.12us	0.20us

Above is the table that measured the execution times for the different methods in us. It can be seen that all the different methods yielded different times. The masking method is seen to be slower in both levels. The reason for this is because the masking approach has to read, modify and write to the pins which takes up a lot of time. It can be seen that the direct bit band is the fastest, followed by the masking and then function. The reason direct bit band is the fastest is because it allows one to directly write to one bit without having to read, change, and write back to a whole register (like bit masking). There is also a big difference between the execution times between level zero and three. The reason for this is because level three is known to optimize for time. This is done by using IT blocks. short if/else sequences are compiled into IT/ITE guarded instructions instead of branch/label pairs. Fewer taken branches result in faster execution.

Target (Demo) Version Analysis:

In this version of the code, the delays were very important to implement, as it allowed for a pause in the methods. To measure the time of the delay function, the profiling tool was used. For a delay function call of delay(500), it took 0.56s at level zero. On level three, it took 0.4s. As mentioned before, this is the result of the optimization in level three, making it more efficient and causing the delay to be shorter in level three.

The barrel shifter method shows how the ARM cortex M3 chip is able to perform shifts as one line of execution. It starts with the value 0x01 and shifts it left by one position at each step, effectively doubling the value each time. This shifting continues through all the LEDs, and when the value overflows, the sequence resets back to 0x01. The loop runs twice, so the pattern traverses the full LED set two complete times.

4.0 Conclusion

Between the two versions (Debug and Demo), the key differences were a result of what purpose they were used for and where they were going to be run. First the Demo version was created and it was built so that it could be demonstrated and the different modes could be observable to the human eye on the board. To do this, it had delays and implemented a SysTick rotation to cycle through the modes. The debug mode on the other hand was designed for only profiling and running analysis. Thus, all delays, LCD operations and SysTick was removed so everything could be timed accurately and easily. Having both was necessary, as the debug version helped with analysis and troubleshooting. The demo version reflected a code optimized for real world execution. With the debug version and running profile analyzer, it was easy to spot and understand the difference between level -O0 and level -O3.

Overall, this lab demonstrated how bit banding, conditional execution, and the barrel shifter improve efficiency on the Cortex-M3. It also highlighted how compiler optimizations significantly reduce execution time, and why using both Debug and Target versions is essential for balancing accurate analysis with realistic application performance.