Faculty of Engineering and Architectural Science

Department of Electrical and Computer Engineering

Course Number	COE 718
Course Title	Embedded Systems Design
Semester/Year	F2020
Lab No	2
Instructor Name	Saber Amini
Section No	03

Submission Date	10/13/2020
Due Date	10/13/2020

Name	Student ID	Signature*
Vatsal Shreekant	500771363	

**By signing above, you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*

www.ryerson.ca/senate/current/pol60.pdf

Introduction

When implementing applications for embedded systems, there is often a need to clear and set individual bits within peripheral and SRAM registers. For instance, to check when an A/D conversion is complete, it is necessary to check the status flag for completion, obtain the value, and then reset the flag to obtain a new conversion.

The bitwise AND and OR masks are needed to check, set and clear the flags. The Cortex-M processors provide a more efficient implementation to perform these frequent actions, known as Bit Banding.

Bit Banding is a technique which allows individual bits in the SRAM and peripheral registers to be read or written to, as opposed to reading a whole register and making the desired bits. These registers are bit addressable.

Table 1: Address Allocation of SRAM and Peripheral Regions

0x43FFFFFF	
0x42000000	32 MB Bit band alias
0x41FFFFFF	
0x40100000	31 MB
0x40000000	1 MB Bit band region

0x23FFFFFF	
0x22000000	32 MB Bit band alias
0x21FFFFFF	
0x20100000	31 MB
0x20000000	1 MB Bit band region

The Cortex-M3 memory map includes two bit-band regions. These occupy the lowest 1MB of the SRAM and peripheral memory regions respectively.

SRAM: Bit-band region: 0x20000000 - 0x20100000 Bit-band alias: 0x22000000 - 0x23FFFFFF

PERI: Bit-band region: 0x40000000 - 0x40100000 Bit-band alias: 0x42000000 - 0x43FFFFFF

The mapping formula is:

bit_word_offset = (byte_offset * 32) + (bit_number * 4)

bit_word_address = bit_band_base + bit_word_offset

Table 2: Calculations:

SRAM Bit-band Base alias Base is	0x22000000
FIOPIN Base address is	0x2009C034
P1.28 LED Bit Number In Hex	0x1C
P1.29 LED Bit Number In Hex	0x1D
P1.31 LED Bit Number In Hex	0x1F
LPC_GPIO Base Word offset	0x0009C034

LPC_GPIO_BASE_word_offset = 0x2009C034 - 0x20000000 = 0x0009C034

byte_offset * 32 0x01380680

LPC_GPIO_Base_Word_offset * 0x20 = 0x0009C034 * 0x20 = 0x01380680

<u>bit_band_base+(byte_offset * 32) is shared for LED 0-2 and is equal to 0x23380680</u>

0x22000000 + (0x0009C034 * 0x20) = 0x22000000 + 0x01380680 = 0x23380680

<u>bit word address for led P1.28 is 0x233806F4</u>

0x23380680 + (0x1C * 0x4) = 0x23380680 + 0x00000074 = 0x233806F4

<u>bit word address for led P1.29 is 0x233806FC</u>

0x23380680 + (0x1D * 0x4) = 0x23380680 + 0x0000007C = 0x233806FC

<u>bit word address for led P1.31 is 0x233806F0</u>

0x23380680 + (0x1F * 0x4) = 0x23380680 + 0x00000070 = 0x233806F0

<u>Table 3: Performance of the 3 methods (Masking, Bit Band & Direct Bit Banding)</u>

Method	**Execution Time (-O0)[Microsecond]**	**Execution Time (-O3)[Microsecond]**	**Performance Improvement[Microsecond]**
Masking	0.09	0.07	0.02
BitBand() Function	0.11	0.08	0.03
Direct Bit Banding	0.04	0.02	0.02

<u>Procedure</u>

1) Load cond_ex example project and complete the instructions in the lab manual.
2) Select the following packages under ‘Manage Run-Time Environment’ window and select OK button:
   Board Support>LEDb.
   CMSIS>CORE
   Compiler>Event Recorderd
   Device > Startup,GPIO, PIN
3) Modify the ‘cond_ex.c’ file to include the following functions: Masking, BitBand, Direct Bit Banding and Barrel Shifter.

```
* Name:    cond_ex.c
 * Purpose: LED Flasher for MCB1700
 *----------------------------------------------------------------------------*/

//barrel shifter code
#include "LPC17xx.h"
```

```
#define ADDRESS(x) (*((volatile unsigned long *)(x)))
#define BitBand(x, y)  ADDRESS(((unsigned long)(x) & 0xF0000000) | 0x02000000
|(((unsigned long)(x) & 0x000FFFFF) << 5) | ((y) << 2))
#define GPIO1_LED31 (*((volatile unsigned long*)0x233806FC))
#define GPIO2_LED2 (*((volatile unsigned long *) 0x23380A88))
#include <string.h>
#include <stdio.h>

int main(void){
	char text[10];
		int r1 = 1, r2 = 1, r3 = 2;
	sprintf(text, "Hello");
volatile unsigned long * GPIO1_LED28 ;
volatile unsigned long * GPIO1_LED29 ;
volatile unsigned long * GPIO2_LED4 ;
	GPIO1_LED28 = &BitBand(&LPC_GPIO1->FIOPIN1, 28);
	GPIO2_LED4 = &BitBand(&LPC_GPIO2->FIOPIN0, 4);

while(1){
if((r1 - r2) < r3){
				printf("bit banding\n");
				GPIO2_LED2 = 1;//LED P2.2 ON using BB
				GPIO1_LED31 = 1;//LED P1.31 ON using BB
				//bit banding
				r1 += 1; //math for conditional execution

			}else if((r1 - r2) > r3){
				printf("function mode\n");
				*GPIO2_LED4 = 1;//LED P1.29 ON using function
				*GPIO1_LED28 = 1;//LED P1.28 ON using function
				//function mode
				r1 = 2;

			}else{
				printf("mask mode\n");
				LPC_GPIO1->FIOPIN |=  ( 1 << 29);
				LPC_GPIO2->FIOPIN |=  ( 1 << 3);
				//mask mode
				r1 += 3;

			}
	int i, j;
		for(i = 0; i < 2000; i++){
		for(j = 0; j < 2000; j++);
	}

	LPC_GPIO2->FIOPIN &=  ~( 1 << 3);
```

```
        LPC_GPIO2->FIOPIN &=  ~( 1 << 29);
        *GPIO1_LED28 = 0;
        *GPIO1_LED29 = 0;
        GPIO2_LED2 = 0;
        GPIO1_LED31 = 0;
}
/*
//bit band mode
GPIO1_LED31 = 1 ; // on
GPIO1_LED31 = 0 ; // o f f

//function mode
GPIO1_LED28 = &BitBand(&LPC_GPIO1->FIOPIN1, 28);
GPIO1_LED28 = &BitBand(&LPC_GPIO1->FIOPIN1, 28);
*GPIO1_LED28 = 1 ; // on
*GPIO1_LED28 = 2 ;
*/
//masking mode
// LPC_GPIO1->FIOPIN &=  ~( 1 << 29);
// LPC_GPIO2->FIOPIN &=  ~( 1 << 3);
}
```

4) Compile project using the build button and start the simulation by selecting the debug button.
5) Select Peripherals→GPIO Fast Interface→Port 1 and Port 2 from toolbar and run.

Conclusion

When comparing and analyzing the results under debug mode, it is evident that directly accessing and switching the bit has the lowest time and this was the initial assumption before implementing the code. The BitBand() function has the most inferior performance in comparison to the other two and is close to masking due to the time it takes to calculate the address to target. The masking method is somewhat more efficient than BitBand() simply because it doesn't need to run calculations to access the bits.

References

1) NXP User Manual, https://www.nxp.com/docs/en/user-guide/UM10360.pdf, 2020
2) ARM Keil User Guide, https://www.keil.com/support/man/docs/mcb1700/mcb1700_intro.htm, 2020