NYU TANDON SCHOOL OF ENGINEERING Name: Ankita Gupta

EL648 – Real Time Embedded Systems ID: ag_9135

Exam 1 - Fall 2022

Instructions: Answer all questions on this paper. You may use any reference materials from class or from our class websites and/or reference sheets. Any cheating identified will result in a zero score on this exam.

Question 1:

In class we showed how to use "mixed" languages to code our embedded system. Suppose you would like to write an assembly function called "GetReg" which takes in a single parameter as an index to a general purpose register. For example, 0 corresponds to R0, 6 corresponds to R6 etc.)

The function then returns the value of that register as a 32 bit integer. In essence, GetReg() is a helper function to get the values of any general purpose register.

a. Write the arm assembly code required to implement the function GetReg() and expose it to your C code. Be sure to handle out-of-range values.
b. Write the C code to import the assembly function GetReg()and call the assembly function to return the values of R0 through R15 and print them to the serial terminal.
c. Submit both your .cpp and.S file.

Answer 1)

a)

The main.cpp file is defined as follows -

```
#include <mbed.h>
#include <iostream>
#include <stdio.h>
extern "C" uint32_t GetReg(uint32_t n);
int main() {
 uint32_t regValue;

 regValue = GetReg(1);
 printf(" The value of 1st register is : %ld\n",regValue);
```

The RegExp.s file (func) is defined as follows –

Please note: The values using MOV for all registers are initialized for testing purposes.

```
.syntax unified
.global GetReg

GetReg:
   mov r1, #1
   mov r2, #2
   mov r3, #3
   mov r4, #4
   mov r5, #5
   mov r6, #6
   mov r7, #7
   mov r8, #8
   mov r9, #9
   mov r10, #10
   mov r11, #11
   mov r12, #12

   cmp r0,#16
   bge Out_Of_Range

   cmp r0, #0
   beq regis0

   cmp r0, #1
   beq regis1

   cmp r0, #2
   beq regis2

   cmp r0, #3
   beq regis3

   cmp r0, #2
   beq regis4

   cmp r0, #5
   beq regis5
```

```
cmp r0, #6
beq regis6

cmp r0, #7
beq regis7

cmp r0, #8
beq regis8

cmp r0, #9
beq regis9

cmp r0, #10
beq regis10

cmp r0, #11
beq regis11

cmp r0, #12
beq regis12

cmp r0, #13
beq regis13

cmp r0, #14
beq regis14

cmp r0, #15
beq regis15

Out_Of_Range:
mov r0,#404
bx lr

regis0:
bx lr

regis1:
mov r0,r1
bx lr

regis2:
mov r0,r2
```

```
bx lr

regis3:
mov r0,r3
bx lr

regis4:
mov r0,r4
bx lr

regis5:
mov r0,r5
bx lr

regis6:
mov r0,r6
bx lr

regis7:
mov r0,r7
bx lr

regis8:
mov r0,r8
bx lr

regis9:
mov r0,r9
bx lr

regis10:
mov r0,r10
bx lr

regis11:
mov r0,r11
bx lr

regis12:
mov r0,r12
bx lr

regis13:
```

```
  mov r0,r13
  bx lr

  regis14:
  mov r0,r14
  bx lr

  regis15:
  mov r0,r15
  bx lr
```

Answer 1)

b)

The main.cpp file is defined as follows –

```
#include <mbed.h>
#include <iostream>
#include <stdio.h>
extern "C" uint32_t GetReg(uint32_t n);
extern "C" uint32_t Init_Rzero();
int main() {
 uint32_t regValue;
 for (int i=0; i<=15;i++)
 {
 regValue = GetReg(i);
 printf(" The value of %dth position register is : %ld\n",i,regValue);
 }

 regValue = GetReg(16);
 printf(" The value of Out of range register is automatically set to : %ld\n",regValue);
}
```

The RegExp.s file (func) is defined as follows –

Please note: The values using MOV for all registers are initialized for testing purposes except r4 since it is used for iteration.

```
.syntax unified
.global GetReg

GetReg:
   mov r1, #1
   mov r2, #2
   mov r3, #3
   mov r5, #5
   mov r6, #6
   mov r7, #7
   mov r8, #8
   mov r9, #9
   mov r10, #10
   mov r11, #11
   mov r12, #12

   cmp r0,#16
   bge Out_Of_Range

   cmp r0, #0
   beq regis0

   cmp r0, #1
   beq regis1

   cmp r0, #2
   beq regis2

   cmp r0, #3
   beq regis3

   cmp r0, #2
   beq regis4

   cmp r0, #5
   beq regis5

   cmp r0, #6
   beq regis6
```

```
cmp r0, #7
beq regis7

cmp r0, #8
beq regis8

cmp r0, #9
beq regis9

cmp r0, #10
beq regis10

cmp r0, #11
beq regis11

cmp r0, #12
beq regis12

cmp r0, #13
beq regis13

cmp r0, #14
beq regis14

cmp r0, #15
beq regis15

Out_Of_Range:
mov r0,#404
bx lr

regis0:
bx lr

regis1:
mov r0,r1
bx lr

regis2:
mov r0,r2
bx lr
```

```
regis3:
mov r0,r3
bx lr

regis4:
mov r0,r4
bx lr

regis5:
mov r0,r5
bx lr

regis6:
mov r0,r6
bx lr

regis7:
mov r0,r7
bx lr

regis8:
mov r0,r8
bx lr

regis9:
mov r0,r9
bx lr

regis10:
mov r0,r10
bx lr

regis11:
mov r0,r11
bx lr

regis12:
mov r0,r12
bx lr

regis13:
mov r0,r13
bx lr
```

```
regis14:
mov r0,r14
bx lr

regis15:
mov r0,r15
bx lr
```

Question 2:

Repeat problem 1, but now modify your code segments so that the function GetReg() takes in one parameter, rArray, which is the address of an array of 16 integers. The GetReg() function should populate the rArray with the contents of all 16 general purpose registers, in order. For example, rAarray[0]=R0, rArray[1]=R1 etc.) In addition, the function should return the contents of the CPSR register.

Answer 2)

The main.cpp file is defined as follows –

```
#include <mbed.h>

#include <iostream>

#include <stdio.h>

extern "C" uint32_t GetReg2(uint32_t *n);

int main() {

 uint32_t rArray[16];

 uint32_t cpsr = GetReg2(rArray);

 for (int i=0; i<16; i++)

 {

 printf("The value of %dth position by pointer is : %ld",i,rArray[i]);

 }

 printf("The value of CPSR is %d", cpsr);

}
```

The RegExp2.s file (func2) is defined as follows –

Please note: The values using MOV for all registers are initialized for testing purposes.

```
.syntax unified
.global GetReg2
GetReg2:
   mov r1, #1
   mov r2, #2
   mov r3, #3
   mov r4, #4
   mov r5, #5
   mov r6, #6
   mov r7, #7
   mov r8, #8
   mov r9, #9
   mov r10, #10
   mov r11, #11
   mov r12, #12

   add r0, r0, #4
   str r1, [r0]

   add r0, r0, #4
   str r2, [r0]

   add r0, r0, #4
   str r3, [r0]

   add r0, r0, #4
   str r4, [r0]

   add r0, r0, #4
   str r5, [r0]

   add r0, r0, #4
   str r6, [r0]

   add r0, r0, #4
   str r7, [r0]

   add r0, r0, #4
   str r8, [r0]
```

```
add r0, r0, #4
str r9, [r0]

add r0, r0, #4
str r10, [r0]

add r0, r0, #4
str r11, [r0]

add r0, r0, #4
str r12, [r0]

add r0, r0, #4
str r13, [r0]

add r0, r0, #4
str r14, [r0]

add r0, r0, #4
mov r1, r15
str r1, [r0]

bx lr
```

Question 3:

Consider the following ARM assembly code segment (*in italics*). Describe what this function does and the contents of R0, R1, R2, and R3 for each iteration and when the function terminates.

Please note: I have added comments in blue around the instructions being implemented

```
        AREA    myData, DATA            //telling us that it doesn't contains the reg but data
        ALIGN                           //aligns to a specified loc
str     DCB     "123",0                 //the input in char(str) value
        AREA  MyFunc, CODE              // telling us that it contains reg
        EXPORT MyFunc                   //similar to declaring a global variable
;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
 MyFunc:
//Name of the label
        LDR     r1, =str
//fetch the contents of the
memory pointer
MOVS r2, #0
//assigns 0 to r2 register
loop:
        LDRB r0, [r1], #1               //r0 = [ r1] + 1
        CBZ     r0, stop                //if r0=0, then jump to 'stop' label
        CMP     r0, #0x30               //compare input with 30 value
        BLT stop //if less than
        30 then 'stop'
        CMP r0, #0x39                   //compare input with 39 value
        BGT stop //if greater
        than 30 then 'stop'
        SUBS r0, r0, #0x30              //r0=r0-30
        ADD r3, r2, r2, LSL #2          //r3=r2 + r2 shift left by 2
        ADD r2, r0, r3, LSL #1          //r2 = r0 + r3 shift left by 1
        B loop                          //initiate 'loop' again
stop B  stop
//stop loop
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ENDP                            //end of program/proc
```

Answer 3)

The above program is converting the string to decimal (digital) value. The ascii value of digits lies between 30 to 39 as 0 to 9.

The content of the variables is changing in the following ways (inside the loop)–

Iteration 1:

R1 – [str] + 4

R0 – 1

R3 – 0+0*4 = 0

R2 – 1+0*2 = 1

Iteration 2:

R1 – [str] + 4

R0 – 2

R3 – 1+1*4 = 5

R2 – 2+5*2 = 12

Iteration 3:

R1 – [str] + 4

R0 – 3

R3 – 12+12*4 = 60

R2 – 3+60*2 = 123

Iteration 4:

R0 will be 0. This would make the program terminate and the last values of –

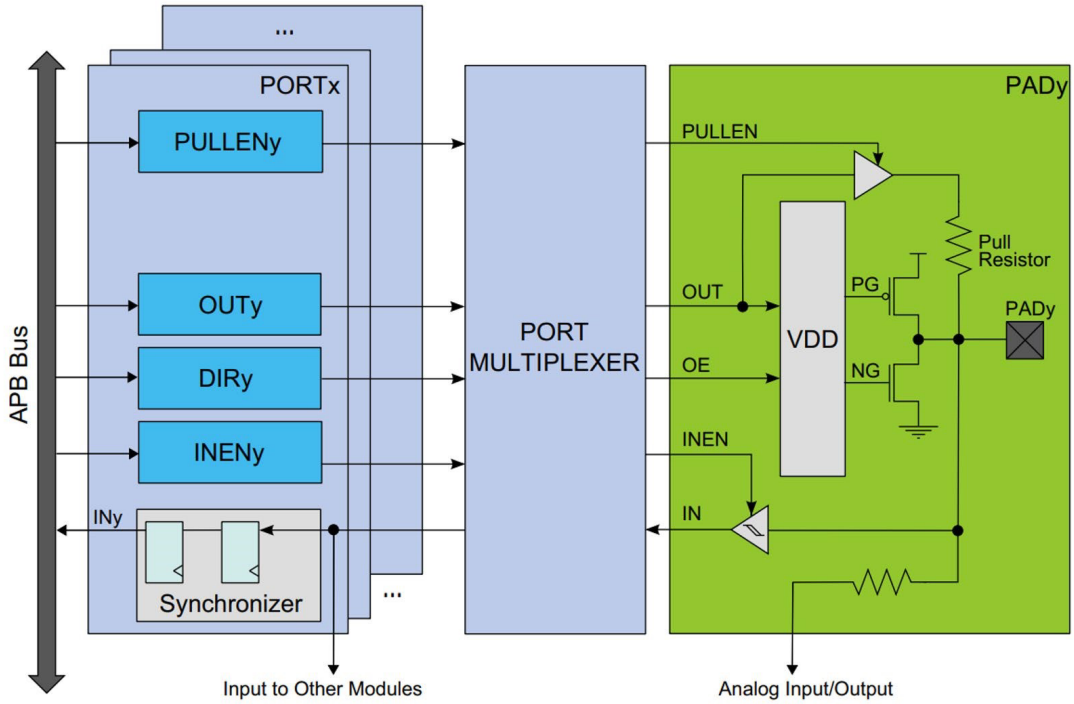
R1 – [str] + 4

R2 – 123 (Final Decimal)

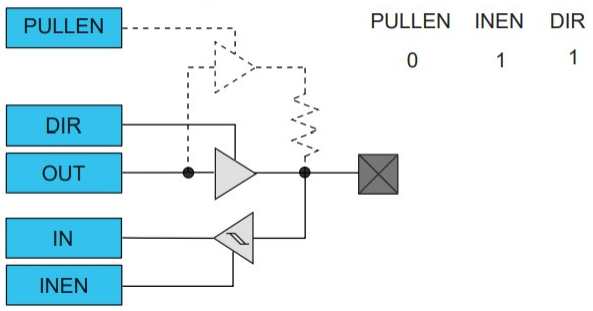
R3 – 60

Question 4:

In class we demonstrated that by controlling a few registers, we can configure our GPIO for several different applications. The example in our notes shows the configurations for the AtmelSAMD21 (M0+) 32 bit ARM microcontroller (see slide 12). A snap shot is shown below:



Using the datasheet for the ATSAMD21 microcontroller (easily found online and in this assignment), write a c function configIO() that configures the PULLEN,OUT,DIR, and INEN so that PortA pins 0-16 are set to Inputs with pull-up resistors, and pins 17-31 are setup as totempole output (input enabled) as seen below. See Section 23 of the datasheet. You can use any HAL macros or definitions of registers if you wish.



Answer 4)

The following assumptions are made on the basis of the attachment from table 23.8.11 and 23.6.3.2 diagrams in addition to base offset mentioned in the PPTs. As we are writing and updating the bits, we are using WRCONFIG.

Considering the first 16 bits follows the rule as
PULLEN = 0
INEN = 1
DIR = 1
OUT = not defined

Considering the last 16 bits follows the rule as Totem Pole (Input enabled)
PULLEN = 1
INEN = 1
DIR = 0
OUT = 1

The C code would be as follows –

```
#include <stdio.h>
#include <stdint.h>

#define REG_PORT_DIR0     (0x41004400U)
#define REG_PORT_OUT0     (0x41004410U)
#define REG_PORT_WRCONFIG0 (0x41004428U)
#define REG_PORT_PINCFG0

void configIO(){
   *REG_PORT_DIR0 = 0x00000000U;
   *REG_PORT_DIR0 = 0xFFFF0000U; //
   *REG_PORT_OUT0 |= 0x0000FFFFU; //

   //Configuring WRCONFIG for the lower 16-bits
   *REG_PORT_WRCONFIG0 &= ~(1 << 31) // setting individual binary bits
   *REG_PORT_WRCONFIG0 |= (1 << 30) // setting individual binary bits
   *REG_PORT_WRCONFIG0 |= (0b11 << 17) // PULLEN = 1; INEN = 1 (0-16bits)

   //Configuring WRCONFIG for the upper 16-bits
   *REG_PORT_WRCONFIG0 |= (1 << 31) // setting individual binary bits
   *REG_PORT_WRCONFIG0 |= (1 << 30) // setting individual binary bits
   *REG_PORT_WRCONFIG0 |= (0b01 << 17) // PULLEN = 0; INEN = 1 (17-32bits)
};

int main(){
   configIO();}
```