Sheffield Hallam University  
Semester I of academic year 2022/23

**"SoC architectures and FPGA prototyping"**

**Lab 7 (optional) –** *Determining efficient C coding methods for the STM32H7 SoC*

This lab’s assignments will measure execution times required for

- adding differently defined constants;

- the same loop defined using various C language options;

- executing two assembly instructions and their combinations.

in order to find the most efficient ways to program typical tasks on the STM32H7 SoCs.

Please make sure that you compile the test codes with the compiler option O0 (no optimisation) with the only exception that will be mentioned specifically.

For all the assignments please use hardware acceleration options (cache enable and FPU settings) according to the **last digit of your student ID number** - **33034339**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| FPU | DP | No | DP | No | DP | No | DP | No | DP | No |
| Dcache | Yes | Yes | Yes | Yes | No | No | No | No | Yes | No |
| Icache | No | No | Yes | Yes | Yes | Yes | No | No | Yes | No |

Throughout this lab you will measure execution time of various code snippets using the following hardware and software mechanism:  
- ARM provides a 24 bit timer (system timer) with each of their Cortex M cores; this timer counts the system clock cycles and can trigger an interrupt when the clock count reaches some set value;  
- in the template project the system timer’s count is set to the value to trigger the interrupt every 1 ms; the interrupt service routine increments the value of the global variable thatcan be accessed by calling the subroutine **HAL\_GetTick(),** for example  **sta=HAL\_GetTick();  
-** therefore calling this subroutine tells the time in ms elapsed since the CPU started executing the code, and can be used for measuring time intervals;  
- as the duration of the clock cycle for the 400 MHz system frequency is only 2.5 ns, 400,000 instructions are required to increase the **Tick** count by 1. We need to loop every code snippet for a substantial number of times. ***Please use your student ID number for the loop counts throughout this lab. As the empty loop itself requires some time to get executed, this time is measured in variable*** looptime ***which needs to be subtracted from the measured time as shown in the examples.***

***The template project is available from BB, please copy it for every lab assignment, and add your codes strictly between the placeholders:***

***// YOUR CODE SHOULD START AFTER THIS LINE...***

***// YOUR CODE SHOULD END BEFORE THIS LINE***

***Assignment 1. Measuring execution time for adding differently defined constants***

Copy the template project into a directory ***lab\_7\_1***, and work in this new directory.

In many programs we would like to define parameters that are easy to modify in order to adapt the code to different requirements.

This can be done by using **#define** statements

**#define CONST1 17 // day of the month on which you were born**

**#define CONST2 1234567 // your student ID number**

or, alternatively, by using some variables for these values

**uint32\_t c1=CONST1, c2=CONST2, tmp1;**

These constants can be added to a variable in the following ways that are the same from the point of view of the programmer but may require different time to process by a microcontroller.

**tmp1 += CONST1; tmp1 += c1; // for CONST1**

**tmp1 += CONST2; tmp1 += c2; // for CONST2**

You will measure execution time for these four statements in the loop that uses your student ID number similarly to the code snippet below

**#define CONST1 17 // day of the month on which you were born**

**#define CONST2 1234567 // your student ID number**

**uint32\_t c1=CONST1, c2=CONST2, tmp1, ctr;**

**// code snippet for measuring time for tmp1 += CONST1**

**// (you will need to add three more snippets)**

**sta= HAL\_GetTick();**

**for (ctr=0; ctr<CONST2; ctr++) {**

**tmp1 += CONST1;**

**};**

**fin=HAL\_GetTick();**

**printf("Loop time = %d ms\n",fin-sta-looptime);**

***// please add three more snippets for the three other options***

After running the code, write down the execution times to the table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Integer += | | | |
| OP | CONST1 | CONST2 | c1 | c2 |
| Time, ms |  |  |  |  |

***Assignment 2. Measuring execution time for differently coded loops***

Open project from the folder directory ***lab\_7\_2***.

We use computers because they can do repetitive tasks many times very fast without any complains or errors. To tell a computer that we want it to repeat something, loops are used.

It is possible to use integer and floating point loop counters; it is possible to use constants or variables to define loops.

Here we are going to measure time required to execute FOR loops using various options in order to find out which is the best option for the STM32H7 MCU.

Two different types of FOR loops you will use are shown below for integer and float variables  
**for (ctr1=0; ctr1<CONST2; ctr1++); // constants int**

**for (ctr1=ctrsta; ctr1<ctrfin; ctr1+=ctrstep); // variables int**

**for (fctr=0.; fctr<(float)CONST2; fctr+=1.); // constants float**

**for (fctr=fctrsta; fctr<fctrfin; fctr+=fctrstep); // variables float**

You will need to run these codes

- for integer **ctr, ctrsta, ctrfin, ctrstep (uint32\_t)**

-for floating point **fctr, fctrsta, fctrfin, fctrstep (float -** add ***f*** at the start of the name to not have errors during compile).

Here is the snippet of code to run the loop for integer constants

**#define CONST1 1234567 // your student ID number**

**#define CONST2 100**

**uint32\_t ctrsta=0, ctrfin=CONST2, ctrstep=1;**

**float fctr, fctrsta=0., fctrfin=(float)CONST2, fctrstep=1.;**

**//snippet for measuring time for integer constants**

**// (you will need copy it three more times)**

**sta= HAL\_GetTick();**

**for (ctr=0; ctr<CONST1; ctr++) {**

**for (ctr1=0; ctr1<CONST2; ctr1++); // constants int**

**};**

**fin= HAL\_GetTick();**

**printf("constants int = %d ms\n",fin-sta-looptime);**

**HAL\_Delay(10);**

***//*** please add three more snippets for the three other options

Run the code and note the execution times.

Change the compiler optimisation option to O3, compile the code again, and note the execution times

|  |  |  |
| --- | --- | --- |
| **FOR loop**  **execution time, ms** | **O0** | **O3** |
| Integer counter constants  variables |  |  |
|  |  |
| FP counter constants  variables |  |  |
|  |  |

***Assignment 3. Measuring execution times for assembly instructions and their combinations***

Copy the template project into a directory ***lab\_7\_3***, and work in this new directory.

Sometimes use of assembly instructions is beneficial, and compiler we use allows mixing C and assembly instructions (but one needs to be careful not to break the compiled C code by inserting inappropriate assembly instructions).

This assignment is dedicated to measurement of the execution time of some assembly instructions that is complicated by the pipelined and superscalar microarchitecture of Cortex M7 in the STM32H7 SoC.

In order eliminate influence of the above mentioned microarchitectural features, we will measure the execution time within the loop that includes many NOP (no operation) instructions to flush the pipeline before and after executing the instructions of interest as follows:

**uint32\_t tmp1, tmp2, tmp3, \*tmp4=&tmp3;**

**#define CONST 1234567 // your ID number**

**//snippet for measuring time for two arithmetic**

**sta= HAL\_GetTick();**

**for (ctr=0; ctr< CONST; ctr++) { // your ID number**

**// flush the pipeline before the instruction to be timed**

**\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();**

**\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();**

**// here come the two arithm instruction(s) to be timed**

**\_\_asm("ADD %[res],%[in1],%[in2]"**

**: [res] "=r" (tmp3)**

**: [in1] "r" (tmp2), [in2] "r" (tmp1) );**

**\_\_asm("ADD %[res],%[in1],%[in2]"**

**: [res] "=r" (tmp3)**

**: [in1] "r" (tmp2), [in2] "r" (tmp1) );**

**// flush the pipeline after the instructions to be timed**

**\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();**

**\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();\_\_nop();**

**};**

**fin= HAL\_GetTick();**

**printf("two arithmetic = %d ms\n",fin-sta-looptime);**

**// please add three more snippets for the three other options**

**( \_\_nop()** is an ***intrinsic*** ***instruction*** - instruction present in the assembly language but not defined in the standard C; it made accessible through the call to a fictional subroutine, the compiler will just place this instruction instead calling any code; ***note the DUAL underscore \_ \_*** ***at the start of this subroutine’s name***)

You will need to run this code with two same memory access instructions, with two same arithmetic instructions, with the arithmetic instruction followed by the memory access instruction, and the memory access instruction followed by the arithmetic instruction (4 code snippets).

***Please copy and paste the required assembly instructions from the file Assembly.c***

Select your arithmetic instruction based on the last digit of your student ID number from the following table - **33034339**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| instr | SUB1 | ADD1 | SUB2 | ADD2 | SUB1 | ADD1 | SUB2 | ADD2 | SUB1 | ADD2 |

The instruction you will need to use in the code will be respectively

**SUB1:**

**\_\_asm("SUB %[res],%[in1],%[in2]" // tmp1 = tmp3 – tmp2;**

**: [res] "=r" (tmp1)**

**: [in1] "r" (tmp2), [in2] "r" (tmp3) );**

**SUB2: \_\_**

**\_\_asm("SUB %[res],%[in1],%[in2]" // tmp3 = tmp1 – tmp2;**

**: [res] "=r" (tmp3)**

**: [in1] "r" (tmp2), [in2] "r" (tmp1) );**

**ADD1:**

**\_\_asm("ADD %[res],%[in1],%[in2]" // tmp1 = tmp3 + tmp2;**

**: [res] "=r" (tmp1)**

**: [in1] "r" (tmp2), [in2] "r" (tmp3) );**

**ADD2: \_\_**

**\_\_asm("ADD %[res],%[in1],%[in2]" // tmp3 = tmp1 + tmp2;**

**: [res] "=r" (tmp3)**

**: [in1] "r" (tmp2), [in2] "r" (tmp1) );**

Select your memory access instruction based on the penultimate digit of your student ID number from the following table - **33034339**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| instr | LDR1 | STR1 | LDR2 | STR2 | LDR1 | STR1 | LDR2 | STR2 | LDR1 | STR1 |

The instruction you will need to use in the code will be respectively

**LDR1: \_\_**

**\_\_asm("LDR %[res],%[in1]" // tmp1 = \*tmp4;**

**: [res] "=r" (tmp1)**

**: [in1] "m" (\*tmp4) );**

**LDR2:**

**\_\_asm("LDR %[res],%[in1]" // tmp3 = \*tmp4;**

**: [res] "=r" (tmp3)**

**: [in1] "m" (\*tmp4) );**

**STR1:**

**\_\_asm("STR %[res],%[in1]" // \*tmp4 = tmp1;**

**: [res] "=r" (tmp1)**

**: [in1] "m" (\*tmp4) );**

**STR2:**

**\_\_asm("STR %[res],%[in1]" // \*tmp4 = tmp1;**

**: [res] "=r" (tmp3)**

**: [in1] "m" (\*tmp4) );**

Note the execution times in the following table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | arithm  then  arithm | mem access  then  mem access | arithm  then  mem access | mem access  then  arithm |
| Execution time, ms |  |  |  |  |

***Lab 7***

|  |
| --- |
| Throughout the lab the following settings for the hardware acceleration were used  FPU – DP / none  D-cache – enabled / disabled  I-cache – enabled/disabled  This loop count was used throughout the lab: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Assignment 1.  Screenshot of the code snippet presented in the lab sheet modified with your data  …  Measured execution times in the table below   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Integer += | | | | | OP | CONST1 | CONST2 | c1 | c2 | | Time, ms |  |  |  |  |   Explain the reason(s) for getting the presented results  … |
| Assignment 2.  Screenshot of the code snippet presented in the lab sheet modified with your data  …  Execution times for differently defined loops   |  |  |  | | --- | --- | --- | | **FOR loop**  **execution time, ms** | **O0** | **O3** | | Integer counter constants  variables |  |  | |  |  | | FP counter constants  variables |  |  | |  |  |   Comment on the measured figures (option O0) and explain why they are different if they are  …  Comment on the differences between the execution times obtained using the O0 and O3 compiler options  … |
| Assignment 3.  Screenshot of the code snippet presented in the lab sheet modified with your data for arithmetic then memory access instruction  …   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | arithm  then  arithm | mem access  then  mem access | arithm  then  mem access | mem access  then  arithm | | Execution time, ms |  |  |  |  |   Comment on the measured figures and explain why they are different if they are  … |