# Project 1 Module 5 Debug Monitor :

The Debug Monitor implementation project has successfully fulfilled a range of essential requirements. These requirements encompassed the activation of Timer0 for precise timing, the seamless integration of crucial mbed components, and the successful importation of key source code files, including shared.h, Monitor.cpp, UART\_poll.cpp, and timer0.cpp. The Debug Monitor itself was effectively implemented, allowing for user interaction through commands like Normal, Debug, Version, Quite and new commands for register dump, memory dump, and stack memory dump. These additional commands significantly enhance the system's functionality and utility. Furthermore, the project underwent improvements, including enhancements to the visual appearance and the integration of flags for controlling LED blinking. Benchmarking was conducted successfully, and comprehensive documentation was generated using Doxygen.

## Time 0 Implementation with Green & Red LED’s :

We seamlessly integrated a precise system tick timer using the mbed ticker library, effectively running the timer0 function every 100 microseconds as specified in main.cpp. This ensured accurate timing for coordinating tasks within the embedded system, making it more reliable and functional. Additionally, we successfully added mbed components to control the green user on chip LED2 and thoroughly tested them, meeting project requirements and enhancing the system's capabilities.

We also incorporated code using flags in the timer0 function to make the Red GPIO LED blink at a 1-second period(500ms ON and 500ms OFF), fulfilling another project requirement. This addition improves the system's functionality by showcasing the effective use of flags to control hardware components.

#### Code for Green LED Implementation:

*DigitalOut greenLED(LED2);*

*extern volatile uint16\_t SwTimerIsrCounter;*

*Ticker tick; // Creates a timer interrupt using mbed methods*

*void flip(void) {*

*greenLED = !greenLED; // Toggle the state of the green LED*

*}*

*if ((SwTimerIsrCounter & 0x1FFF) > 0x0FFF)*

*{*

*// The bitwise AND operation masks (keeps) the 13 least significant bits*

*// (LSBs) of SwTimerIsrCounter and sets all other bits to 0.*

*// This effectively checks if the value of SwTimerIsrCounter is within the range of 0 to 0x1FFF.*

*// If the result of this operation is greater than 0x0FFF, it means that*

*// the value of SwTimerIsrCounter has reached or exceeded 0x0FFF within that range.*

*flip(); // Toggle Green LED*

*}*

#### Code for Read LED Implementation using flags:

DigitalOut redLED(PA\_7); // Red LED at Pin PA\_7

int redLEDTickCounter = 0; // Red LED Tick Counter

void toggleRedLED() {

redLED = !redLED; // Toggle the state of the red LED

}

// B. Heartbeat/ LED outputs

// Generate Outputs \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Create a 0.5-second LED heartbeat here.

redLEDTickCounter++;

if ((redLEDTickCounter % 78) == 0) {

toggleRedLED();

redLEDTickCounter = 0;

}

## Debug Monitor Serial Terminal Output:

Our current debug monitor features seven distinct commands: "Normal," "Quiet," "Debug," "Register Dump," "Stack Dump," "Memory Dump," and "Version." You can reference the screenshot of the entry point in the serial terminal, which is depicted in Figure(1) below.

A screenshot of a computer program

Description automatically generated

Figure : Debug Monitor Commands

### Version Information :

As depicted in the photograph titled "Figure 2 : Debug Monitor – Version Info, Normal & Quite Command", the debug monitor showcases its version as "2.0" with a date of "2016/09/29".

A screenshot of a computer program

Description automatically generated

Figure : Debug Monitor – Version Info, Normal & Quite Command

### Normal Command :

The photograph titled "Figure 2 : Debug Monitor – Version Info, Normal & Quite Command" reveals the interface in 'Normal' mode. In this state, readings for 'Flow', 'Temp', and 'Freq' are prominently displayed. These parameters seem to be dynamically updated, providing real-time insights.

Note : The values at each command are refreshed every 1.6384 seconds.

### Quiet Command Display :

As shown in the photograph titled "Figure 2 : Debug Monitor – Version Info, Normal & Quite Command", when the interface is in 'Quiet' mode, nothing will be printed or displayed. This behaviour suggests that in the 'Quiet' mode, the monitor silences its outputs, offering a minimalistic view as observed in the Figure (2).

### Register Command Display :

The provided image “Figure 3 : Debug Monitor – Normal & Register Dump” starts by showing the system in "NORMAL" mode where values of 'Flow', 'Temp', and 'Freq' are given. It then transitions to the "Registers Dump" mode. Under "Registers Dump", the system is printing the values stored in various registers. Registers from R0 to R12 are listed, each displaying a unique hexadecimal value. Additionally, special registers such as SP (Stack Pointer), LR (Link Register), and PC (Program Counter) are also shown with their respective values.

A screen shot of a computer

Description automatically generated

Figure : Debug Monitor –Normal & Register Dump

### Memory Command Display :

In the "Memory Dump" mode, as shown in the “Figure 4: Debug Monitor – Memory & Stack Dump” image, the system is dumping memory contents from a specified location. In this case, it's dumping memory at location "0X20010000" and it provides 64 bytes of data. The memory dump not only provides the hexadecimal representation of the stored data but also attempts to provide a character representation, as seen on the right side of the hex values. This can be useful in understanding if any recognizable patterns or strings are present in the memory. Code also includes a functionality to take user input for start address and length for memory dump which is commented out for submission.

A screenshot of a computer

Description automatically generated

Figure :s Debug Monitor – Memory & Stack Dump

### Stack Pointer Command Display :

The "Stack Dump" mode offers insights into the current status of the system's stack. At the beginning of this mode, the "Current Stack Pointer SP" is shown with its value "0x20017FE8". Following this, the contents of the stack are printed, starting from the current stack pointer and then displaying word values in descending memory locations (which is typical of a descending stack). Each word is associated with its memory location, and the content at that location is displayed in a hexadecimal format. The stack content provides valuable information about the data stored in the stack and can be crucial for debugging purposes, especially to trace back the execution flow or to check for any stack corruption.

# Documentation :

# We have successfully auto-generated documentation for your codebase using Doxygen. The generated documentation is available in either an HTML directory or a PDF file. Refer the doxygen folder in submitted zip file.

# Lab Questions :

## Question 1: What is the count shown in timer0 if you let it run for 30 seconds? Explain why it is this?

The count shown by timer0, is 37856 as seen in figure(5) below. The variable timer0\_count is constructed with 16 bits. Every 6.5 seconds, this 16-bit variable undergoes a reset due to the reloading of its count value. Within 30 seconds, this resetting occurs 4 times (6.5 multiplied by 4 equals 26 seconds). In the leftover 4 seconds, it increases to a count of 37856.

A screenshot of a computer program

Description automatically generated

Figure : Timer0 Count at 30 seconds

## Question 2: How much time does the code spend in the main loop versus in Interrupt Service Routines?

Utilizing the Timer class from the mbed API, we initiated the timer using timer.start() at the beginning of both our main.cpp and ISR functions. By employing timer.reset(), timer.stop() at the conclusion of each function, we obtained specific timings for both the main function and the ISR, as illustrated in Figure(6) below.

Time take by Main() in seconds : **0.6468 sec**.

A screen shot of a computer

Description automatically generated

Figure : Time taken by ISR

Time taken by ISR in seconds : **2 usec**.

A black screen with white text

Description automatically generated

Figure : Time taken by ISR

Times calculated is before any command given to debug monitor. The reason being value could change for the different debug monitor commands, so we found the best way could be to test it before sending any command to debug monitor.

## Question 3: Test each of the commands in the Debug Monitor and record the results. Explain anything you see that you did not expect. Are you able to display all the registers?

We thoroughly tested each of the commands in the Debug Monitor and documented the outcomes. The recorded results for each debug monitor command is already explained with screenshots in section [Debug Monitor Serial Terminal](#_Debug_Monitor_Serial) [Click Here].

We also encountered a few unexpected new line (\n\r) occurrences during testing, notably when the version was displayed for the first time, as additional newline characters were randomly included. Another issue seen was the first character inputted to serial terminal is never echoed back.

Furthermore, we successfully displayed the values of all registers, including r0 to r13, as well as pc, sp, and lr as seen in figure(3).

## Question 5: What is the new command you added to the debug menu, and what does it do? Capture a screenshot of the new monitor window.

Three new commands named “Register Dump”, “Memory Dump” and “Stack Dump” has been added in debug monitor code.

* Register Dump
  + The "Register Dump" command is designed to display the values and addresses of each register, spanning from r0 to r13. Additionally, it reveals the stack pointer, program counter, and link registers. A visual representation of this can be found in Figure (3).
* Memory Dump
  + The "Memory Dump" command serves as a HEX dump of the memory location. This command enables the dumping of memory at the address 0x20010000, complete with its corresponding ASCII representation. Various memory locations were tested, revealing that attempts to print memory locations such as the reserved memory space at 0x6000 0000 are not permitted. A visual representation of this can be found in Figure (4)
* Stack Dump
  + The "Stack Dump" command offers insights into the data present in the stack, presented in reverse chronological order, beginning from word 16 and proceeding to word 0. A visual representation of this can be found in Figure (4)

## Question 6: Estimate the % of CPU cycles used for the main background process, assuming a 100 millisecond operating cycle.

Time taken by Main() = 0.6468 seconds

Time taken by ISR = 2 usec = 0.000002 seconds

Since the Timer ISR is set at 100 microseconds and gets called 1000 times in 100 milliseconds, the total time for ISR in 100 milliseconds = 0.000002 seconds \* 1000 = 0.002 seconds

Total time for both Main() and ISR in 100 milliseconds = 0.6468 + 0.002 = 0.6488 seconds

Percentage of CPU cycles used for the main background process = (Time taken by Main() / Total time) \* 100 = (0.6468/0.6488) \* 100 = **99.69%**

We estimate that, during a 100 millisecond operating cycle, the main background process uses approximately 99.69% of the CPU cycles, calculated as (0.6468 seconds / 0.6488 seconds) \* 100.

## Question 7: What is your DMIPS estimate for the ST STM32F401RE MCU

Upon executing the Dhrystone benchmark and by referencing the Dhrystone documentation's methodology, as depicted in Figure 9, and using the provided formula. Each run yielded a DMIPS value of 110.352674 and DMIPS/MHz measure is seen as 5.9650 DMIPS/MHz . Consistent results were observed across all ten runs as displayed in Figure 8.

A screenshot of a computer screen

Description automatically generated

Figure : Dhrystone Analysis

A screenshot of a math test

Description automatically generated

Figure : Dhrystone Documentation Of Measurement Characteristics

# Code :

## Main.cpp

/\*\*----------------------------------------------------------------------------

\file main.cpp

-- --

-- ECEN 5803 Mastering Embedded System Architecture --

-- Project 1 Module 4 --

-- Microcontroller Firmware --

-- main.cpp --

-- --

-------------------------------------------------------------------------------

--

-- Designed for: University of Colorado at Boulder

--

--

-- Designed by: Tim Scherr

-- Revised by: Student's name

--

-- Version: 3.0

-- Date of current revision: 2022-06-20

-- Target Microcontroller: ST STM32F401RE

-- Tools used: ARM mbed compiler

-- ARM mbed SDK

-- ST Nucleo STM32F401RE Board

--

--

-- Functional Description: Main code file generated by mbed, and then

-- modified to implement a super loop bare metal OS.

--

-- Copyright (c) 2015, 2016, 2022 Tim Scherr All rights reserved.

--

\*/

#define MAIN

#include "shared.h"

#undef MAIN

#include "NHD\_0216HZ.h"

#include "DS1631.h"

#include "pindef.h"

// Define green LED pin

DigitalOut greenLED(LED2);

// External ISR counter

extern volatile uint16\_t SwTimerIsrCounter;

// Timer interrupt instance

Ticker tick;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Add code to control LED LD2 here,

// including a function to flip the LED state on and off

void flip(void) {

greenLED = !greenLED; // Toggle the state of the green LED

}

// Set up serial communication over USB

Serial pc(USBTX, USBRX);

// Create timer instance

Timer custom\_timer2;

/\*----------------------------------------------------------------------------

MAIN function

\*----------------------------------------------------------------------------\*/

int main()

{

custom\_timer2.start();

pc.baud(9600);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Add code to call timer0 function every 100 uS

tick.attach\_us(&timer0, 100);

// Print the initial banner

pc.printf("\r\nHello World!\n\n\r");

uint32\_t count = 0;

// initialize serial buffer pointers

rx\_in\_ptr = rx\_buf; /\* pointer to the receive in data \*/

rx\_out\_ptr = rx\_buf; /\* pointer to the receive out data\*/

tx\_in\_ptr = tx\_buf; /\* pointer to the transmit in data\*/

tx\_out\_ptr = tx\_buf; /\* pointer to the transmit out \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// uncomment this section after adding monitor code.

/\* send a starting message to the terminal \*/

pc.printf("\n\r\n\r\*\*\*\*\* Project 1 Module 5 - Viraj\_Kiran \*\*\*\*\*\n\r\n\r");

UART\_direct\_msg\_put("\r\nSystem Reset\r\nCode ver. ");

UART\_direct\_msg\_put( CODE\_VERSION );

UART\_direct\_msg\_put("\r\n");

UART\_direct\_msg\_put( COPYRIGHT );

UART\_direct\_msg\_put("\r\n");

set\_display\_mode();

custom\_timer2.stop();

//pc.printf("Time Taken For Main: %f seconds\n\r", custom\_timer2.read());

while(1) /// Cyclical Executive Loop

{

count++; // counts the number of times through the loop

// \_\_enable\_interrupts();

// \_\_clear\_watchdog\_timer();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// uncomment this section after adding monitor code.

serial(); // Polls the serial port

chk\_UART\_msg(); // checks for a serial port message received

monitor(); // Sends serial port output messages depending

// on commands received and display mode

if ((SwTimerIsrCounter & 0x1FFF) > 0x0FFF)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// The bitwise AND operation masks (keeps) the 13 least significant bits (LSBs)

// of SwTimerIsrCounter and sets all other bits to 0. This effectively checks

// if the value of SwTimerIsrCounter is within the range of 0 to 0x1FFF.

// If the result of this operation is greater than 0x0FFF, it means that the

// value of SwTimerIsrCounter has reached or exceeded 0x0FFF within that range.

{

flip(); // Toggle Green LED

}

//Write your code here for any additional tasks

} /// End while(1) loop

}

## Timer0.cpp

/\*\*----------------------------------------------------------------------------

\*

\* \file timer0.cpp

-- --

-- ECEN 5803 Mastering Embedded System Architecture --

-- Project 1 Module 4 --

-- Microcontroller Firmware --

-- Timer0.cpp --

-- --

-------------------------------------------------------------------------------

--

-- Designed for: University of Colorado at Boulder

--

--

-- Designed by: Tim Scherr

-- Revised by: Student's name

--

-- Version: 3.0

-- Date of current revision: 2022-06-20

-- Target Microcontroller: ST STM32F401RE

-- Tools used: ARM mbed compiler

-- ARM mbed SDK

-- ST Nucleo STM32F401RE Board

--

--

Functional Description:

This file contains code for the only interrupt routine, based on the System

Timer.

The System Timer interrupt happens every

100 us as determined by mbed Component Configuration.

The System Timer interrupt acts as the real time scheduler for the firmware.

Each time the interrupt occurs, different tasks are done based on critical

timing requirement for each task.

There are 256 timer states (an 8-bit counter that rolls over) so the

period of the scheduler is 25.6 ms. However, some tasks are executed every

other time (the 200 us group) and some every 4th time (the 400 us group) and

so on. Some high priority tasks are executed every time. The code for the

tasks is divided up into the groups which define how often the task is

executed. The structure of the code is shown below:

I. Entry and timer state calculation

II. 100 us group

A. Fast Software timers

B. Read Sensors

C. Update

III. 200 us group

A.

B.

IV. 400 us group

A. Medium Software timers

B.

V. 800 us group

A. Set 420 PWM Period

VI 1.6 ms group

A. Display timer and flag

B. Heartbeat/ LED outputs

VII 3.2 ms group

A. Slow Software Timers

VIII 6.4 ms group A

A. Very Slow Software Timers

IX. Long time group

A. Determine Mode

B. Heartbeat/ LED outputs

X. Exit

--

-- Copyright (c) 2015, 2022 Tim Scherr All rights reserved.

\*/

#include "shared.h"

#include "mbed.h"

#define System Timer\_INCREMENT\_IN\_US 1000

typedef unsigned char uchar8\_t;

typedef unsigned char bit;

typedef unsigned int uint32\_t;

typedef unsigned short uint16\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Configurations \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definitions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

volatile uchar8\_t swtimer0 = 0;

volatile uchar8\_t swtimer1 = 0;

volatile uchar8\_t swtimer2 = 0;

volatile uchar8\_t swtimer3 = 0;

volatile uchar8\_t swtimer4 = 0;

volatile uchar8\_t swtimer5 = 0;

volatile uchar8\_t swtimer6 = 0;

volatile uchar8\_t swtimer7 = 0;

volatile uint16\_t SwTimerIsrCounter = 0U;

uchar8\_t display\_timer = 0; // 1 second software timer for display

uchar8\_t display\_flag = 0; // flag between timer interrupt and monitor.c, like

// a binary semaphore

static uint32\_t System\_Timer\_count = 0; // 32 bits, counts for

// 119 hours at 100 us period

static uint16\_t timer0\_count = 0; // 16 bits, counts for

// 6.5 seconds at 100 us period

static uchar8\_t timer\_state = 0;

static uchar8\_t long\_time\_state = 0;

// variable which splits timer\_states into groups

// tasks are run in their assigned group times

// DigitalOut BugMe (PTB9); // debugging information out on PTB9

#ifdef \_\_cplusplus

}

#endif

DigitalOut redLED(PA\_7); // Red Led At Pin PA\_7

int redLEDTickCounter = 0; // Red Led Tick Counter

//bool redLEDFlag = false; // Red Led Flag

extern Serial pc;

void toggleRedLED() {

redLED = !redLED; // Toggle the state of the red LED

}

Timer custom\_timer;

int count30s = 0; // 30 Seconds Tick Counter

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Start of Code \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// I. Entry and Timer State Calculation

void timer0(void)

{

custom\_timer.reset();

custom\_timer.start();

// BugMe = 1; // debugging signal high during Timer0 interrupt on PTB9

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Determine Timer0 state and task groups

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

timer\_state++; // increment timer\_state each time

if (timer\_state == 0)

{

long\_time\_state++; // increment long time state every 25.6 ms

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 100 us Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// II. 100 us Group

// A. Update Fast Software timers

if (swtimer0 > 0) // if not yet expired,

(swtimer0)--; // then decrement fast timer (1 ms to 256 ms)

if (swtimer1 > 0) // if not yet expired,

(swtimer1)--; // then decrement fast timer (1 ms to 256 ms)

// B. Update Sensors

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 200 us Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((timer\_state & 0x01) != 0) // 2 ms group, odds only

{

;

} // end 2 ms group

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 400 us Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if ((timer\_state & 0x02) != 0)

{

// IV. 400 us group

// timer states 2,6,10,14,18,22,...254

// A. Medium Software timers

if (swtimer2 > 0) // if not yet expired, every other time

(swtimer2)--; // then decrement med timer (4 ms to 1024 ms)

if (swtimer3 > 0) // if not yet expired, every other time

(swtimer3)--; // then decrement med timer (4 ms to 1024 ms)

// B.

} // end 4 ms group

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 800 us Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if ((timer\_state & 0x04) != 0)

{

// V. 8 ms group

// timer states 4, 12, 20, 28 ... 252 every 1/8

// A. Set

} // end 8 ms group

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 1.6 ms Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if ((timer\_state & 0x08) != 0)

{

// VI 1.6 ms group

// timer states 8, 24, 40, 56, .... 248 every 1/16

} // end 1.6 ms group

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 3.2 ms Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if ((timer\_state & 0x10) != 0)

{

// VII 3.2 ms group

// timer states 16, 48, 80, 112, 144, 176, 208, 240

// A. Slow Software Timers

if (swtimer4 > 0) // if not yet expired, every 32nd time

(swtimer4)--; // then decrement slow timer (32 ms to 8 s)

if (swtimer5 > 0) // if not yet expired, every 32nd time

(swtimer5)--; // then decrement slow timer (32 ms to 8 s)

// B. Update

} // end 3.2 ms group

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 6.4 ms Group A \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if ((timer\_state & 0x20) != 0)

{

// VIII 6.4 ms group A

// timer states 32, 96, 160, 224

// A. Very Slow Software Timers

if (swtimer6 > 0) // if not yet expired, every 64th

// time

(swtimer6)--; // then decrement very slow timer (6.4 ms to 1.6s)

if (swtimer7 > 0) // if not yet expired, every 64th

// time

(swtimer7)--; // then decrement very slow timer (64 ms to 1.6s)

// B. Update

} // end 6.4 ms group A

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 6.4 ms Group B \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else

{

// IX. 6.4 ms group B

// timer states 0, 64, 128, 192

// A. Update

// A. Display timer and flag

display\_timer--; // decrement display timer every 6.4 ms. Total time is

// 256\*6.4ms = 1.6384 seconds.

if (display\_timer == 1)

display\_flag = 1; // every 1.6384 seconds, now OK to display

// B. Heartbeat/ LED outputs

// Generate Outputs \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Create an 0.5 second LED heartbeat here.

// Increment the counter for the red LED ticks

redLEDTickCounter++;

// Check if the counter has reached the value of 78

if ((redLEDTickCounter % 78) == 0) {

// Toggle the state of the red LED

toggleRedLED();

// Reset the counter

redLEDTickCounter = 0;

}

/\* Code With Red LED Flag

if ( ((redLEDTickCounter % 78) == 0) && (redLEDFlag)) {

toggleRedLED();

redLEDTickCounter = 0;

}

else {

redLEDFlag = false;

redLEDTickCounter = 0;

}

\*/

} // end 6.4 ms group B

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Long Time Group \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (((long\_time\_state & 0x01) != 0) && (timer\_state == 0))

// every other long time, every 51.2 ms

{

// X. Long time group

//

// clear\_watchdog\_timer();

}

// Re-enable interrupts and return

System\_Timer\_count++;

timer0\_count++;

SwTimerIsrCounter++;

// Bugme = 0; // debugging signal high during Timer0 interrupt on PTB9

// unmask Timer interrupt (now done by mBed library)

// enables timer interrupt again (now done by mBed Library

// Code To Print Timer 0 Count After 30 Seconds & To Print Time Taken In ISR

//count30s++;

/\*

if ((count30s % 300000) == 0) {

pc.printf("Timer 0 Count After 30 Seconds: %d\n\r", timer0\_count);

count30s = 0;

}

\*/

custom\_timer.stop();

//pc.printf("Time Taken For ISR: %f micro seconds\n\r", custom\_timer.read()\*1000000);

}

## Monitor.cpp

/\*\*----------------------------------------------------------------------------

\file Monitor.cpp

-- --

-- ECEN 5003 Mastering Embedded System Architecture --

-- Project 1 Module 4 --

-- Microcontroller Firmware --

-- Monitor.cpp --

-- --

-------------------------------------------------------------------------------

--

-- Designed for: University of Colorado at Boulder

--

--

-- Designed by: Tim Scherr

-- Revised by: Student's name

--

-- Version: 2.0

-- Date of current revision: 2022-06-20

-- Target Microcontroller: ST STM32F401RE

-- Tools used: ARM mbed compiler

-- ARM mbed SDK

-- ST Nucleo STM32F401RE Board

--

--

Functional Description: See below

--

-- Copyright (c) 2015, 2022 Tim Scherr All rights reserved.

--

\*/

#include <stdio.h>

#include "shared.h"

#include "memory.h"

//extern bool redLEDFlag;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Set Display Mode Function

\* Function determines the correct display mode. The 3 display modes operate as

\* follows:

\*

\* NORMAL MODE Outputs only mode and state information changes

\* and calculated outputs

\*

\* QUIET MODE No Outputs

\*

\* DEBUG MODE Outputs mode and state information, error counts,

\* register displays, sensor states, and calculated output

\* (currently not all features are operation, could be enhanced)

\*

\* There is deliberate delay in switching between modes to allow the RS-232 cable

\* to be plugged into the header without causing problems.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void set\_display\_mode(void)

///

/// \brief Displays a selection menu over UART.

///

/// This function sends a series of messages over UART to display a menu

/// allowing the user to choose between different modes. Each mode is

/// highlighted with a different color.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void set\_display\_mode(void)

{

UART\_direct\_msg\_put("\033[37m\r\n+------------------------+\033[0m");

UART\_direct\_msg\_put("\r\n\033[37m| Select Mode |\033[0m");

UART\_direct\_msg\_put("\033[37m\r\n|------------------------|\033[0m");

UART\_direct\_msg\_put("\r\n\033[32m| NOR - Normal |\033[0m");

UART\_direct\_msg\_put("\r\n\033[33m| QUI - Quiet |\033[0m");

UART\_direct\_msg\_put("\r\n\033[35m| DEB - Debug |\033[0m");

//UART\_direct\_msg\_put("\r\n\033[31m| RED - Red Led(1s) |\033[0m");

UART\_direct\_msg\_put("\r\n\033[34m| REG - Registers Dump |\033[0m");

UART\_direct\_msg\_put("\r\n\033[31m| MEM - Memory Dump |\033[0m");

UART\_direct\_msg\_put("\r\n\033[36m| STK - Stack Dump |\033[0m");

UART\_direct\_msg\_put("\r\n\033[97m| V - Version# |\033[0m");

UART\_direct\_msg\_put("\033[37m\r\n+------------------------+\033[0m");

UART\_direct\_msg\_put("\033[0m\r\nSelect: ");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void chk\_UART\_msg(void)

///

/// \brief - fills a message buffer until return is encountered, then calls

/// message processing

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Improve behavior of this function

void chk\_UART\_msg(void)

{

uchar8\_t j;

while( UART\_input() ) // becomes true only when a byte has been received

{ // skip if no characters pending

j = UART\_get(); // get next character

if( j == '\r' ) // on a enter (return) key press

{ // complete message (all messages end in carriage return)

//UART\_msg\_put("");

UART\_direct\_msg\_put("->");

UART\_msg\_process();

}

else

{

if ((j != 0x02) ) // if not ^B

{ // if not command, then

UART\_put(j); // echo the character

}

else

{

;

}

if( j == '\b' )

{ // backspace editor

if( msg\_buf\_idx != 0)

{ // if not 1st character then destructive

UART\_msg\_put(" \b");// backspace

msg\_buf\_idx--;

}

}

else if( msg\_buf\_idx >= MSG\_BUF\_SIZE )

{ // check message length too large

UART\_msg\_put("\r\nToo Long!");

msg\_buf\_idx = 0;

}

else if ((display\_mode == QUIET) && (msg\_buf[0] != 0x02) &&

(msg\_buf[0] != 'D') && (msg\_buf[0] != 'N') &&

(msg\_buf[0] != 'V') && (msg\_buf[0] != 'R') &&

(msg\_buf[0] != 'M') && (msg\_buf[0] != 'S') &&

(msg\_buf\_idx != 0))

{ // if first character is bad in Quiet mode

msg\_buf\_idx = 0; // then start over

}

else { // not complete message, store character

msg\_buf[msg\_buf\_idx] = j;

msg\_buf\_idx++;

if (msg\_buf\_idx > 3)

{

UART\_msg\_process();

}

}

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void UART\_msg\_process(void)

///UART Input Message Processing

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void UART\_msg\_process(void)

{

uchar8\_t chr,err=0;

// unsigned char data;

// Check if the first character of the message buffer is an uppercase letter

if( (chr = msg\_buf[0]) <= 0x60 )

{ // Upper Case

switch( chr )

{

// DEBUG Mode

case 'D':

if((msg\_buf[1] == 'E') && (msg\_buf[2] == 'B') && (msg\_buf\_idx == 3))

{

display\_mode = DEBUG;

UART\_direct\_msg\_put("\r\n\033[35mMode=DEBUG\033[0m\n");

display\_timer = 0;

}

else

err = 1;

break;

// NORMAL Mode

case 'N':

if((msg\_buf[1] == 'O') && (msg\_buf[2] == 'R') && (msg\_buf\_idx == 3))

{

display\_mode = NORMAL;

UART\_direct\_msg\_put("\r\n\033[32mMode=NORMAL\033[0m\n");

//display\_timer = 0;

}

else

err = 1;

break;

// QUIET Mode

case 'Q':

if((msg\_buf[1] == 'U') && (msg\_buf[2] == 'I') && (msg\_buf\_idx == 3))

{

display\_mode = QUIET;

UART\_direct\_msg\_put("\r\n\033[33mMode=QUIET\033[0m\n");

display\_timer = 0;

}

else

err = 1;

break;

// VERSION Info

case 'V':

display\_mode = VERSION;

UART\_direct\_msg\_put("\033[97m\r\n");

UART\_direct\_msg\_put(CODE\_VERSION);

UART\_direct\_msg\_put("\033[0m\r\nSelect: ");

display\_timer = 0;

break;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Add other message types here

/\* For Red Led Flag Mode

case 'R':

if((msg\_buf[1] == 'E') && (msg\_buf[2] == 'D') && (msg\_buf\_idx == 3))

{

display\_mode = RED;

UART\_msg\_put("\r\n\033[31mMode=RED LED(1s)\033[0m\n");

display\_timer = 0;

}

else

err = 1;

break;

\*/

// Register Dump

case 'R':

if((msg\_buf[1] == 'E') && (msg\_buf[2] == 'G') && (msg\_buf\_idx == 3))

{

display\_mode = REGISTERS;

UART\_msg\_put("\r\n\033[34mMode=Registers Dump\033[0m\n");

display\_timer = 0;

}

else

err = 1;

break;

// Memory Dump

case 'M':

if((msg\_buf[1] == 'E') && (msg\_buf[2] == 'M') && (msg\_buf\_idx == 3))

{

display\_mode = MEMORY;

UART\_msg\_put("\r\n\033[31mMode=Memory Dump\033[0m\n");

display\_timer = 0;

}

else

err = 1;

break;

// Stack Dump

case 'S':

if((msg\_buf[1] == 'T') && (msg\_buf[2] == 'K') && (msg\_buf\_idx == 3))

{

display\_mode = STACK;

UART\_msg\_put("\r\n\033[36mMode=Stack Dump\033[0m\n");

display\_timer = 0;

}

else

err = 1;

break;

// DEFAULT

default:

err = 1;

}

}

// Display error messages based on the error code

else

{ // Lower Case

switch( chr )

{

default:

err = 1;

}

}

if( err == 1 )

{

UART\_direct\_msg\_put("\n\rEntry Error!");

}

else if( err == 2 )

{

UART\_direct\_msg\_put("\n\rNot in DEBUG Mode!");

}

else

{

msg\_buf\_idx = 0; // put index to start of buffer for next message

;

}

msg\_buf\_idx = 0; // put index to start of buffer for next message

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/// \fn is\_hex

/// Function takes

/// @param a single ASCII character and returns

/// @return 1 if hex digit, 0 otherwise.

///

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uchar8\_t is\_hex(uchar8\_t c)

{

if( (((c |= 0x20) >= '0') && (c <= '9')) || ((c >= 'a') && (c <= 'f')) )

return 1;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \fn DEBUG and DIAGNOSTIC Mode UART Operation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void monitor(void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Spew outputs \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

switch(display\_mode)

{

case(QUIET):

{

//redLEDFlag = false;

UART\_msg\_put("\r\n ");

display\_flag = 0;

}

break;

case(VERSION):

{

//redLEDFlag = false;

display\_flag = 0;

}

break;

case(NORMAL):

{

//redLEDFlag = false;

if (display\_flag == 1)

{

UART\_msg\_put("\r\n\033[32mNORMAL ");

UART\_msg\_put(" Flow: ");

// ECEN 5803 add code as indicated

// add flow data output here, use UART\_hex\_put or similar for

// numbers

UART\_msg\_put(" Temp: ");

// add flow data output here, use UART\_hex\_put or similar for

// numbers

UART\_msg\_put(" Freq: \033[0m");

// add flow data output here, use UART\_hex\_put or similar for

// numbers

display\_flag = 0;

//wait\_ms(500);

}

}

break;

case(DEBUG):

{

//redLEDFlag = false;

if (display\_flag == 1)

{

UART\_msg\_put("\r\n\033[35mDEBUG ");

UART\_msg\_put(" Flow: ");

// ECEN 5803 add code as indicated

// add flow data output here, use UART\_hex\_put or similar for

// numbers

UART\_msg\_put(" Temp: ");

// add flow data output here, use UART\_hex\_put or similar for

// numbers

UART\_msg\_put(" Freq: \033[0m");

// add flow data output here, use UART\_hex\_put or similar for

// numbers

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ECEN 5803 add code as indicated \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Create a display of error counts, sensor states, and

// ARM Registers R0-R15

//print\_registers();

// Create a command to read a section of Memory and display it

//dump\_memory();

// Create a command to read 16 words from the current stack

// and display it in reverse chronological order.

//display\_last\_16\_stack\_words();

// clear flag to ISR

display\_flag = 0;

//wait\_ms(500);

}

}

break;

/\* Red Led Flag Mode

case(RED):

{

redLEDFlag = true;

if (display\_flag == 1) {

UART\_msg\_put("\r\n\033[31mRed LED(1s)\033[0m ");

display\_flag = 0;

}

}

break;

\*/

// Check if the current mode is REGISTERS

case(REGISTERS):

{

// The following line is commented out, but if active, it would turn off a red LED flag

//redLEDFlag = false;

// Check if the display\_flag is set

if (display\_flag == 1) {

// Send a new line and set the text color to blue

UART\_direct\_msg\_put("\n\r\033[34m");

// Print the register values

print\_registers();

// Reset the text color to default and send a new line

UART\_direct\_msg\_put("\033[0m\n\r");

// Reset the display\_flag

display\_flag = 0;

}

}

break; // End of REGISTERS case

// Check if the current mode is MEMORY

case(MEMORY):

{

// The following line is commented out, but if active, it would turn off a red LED flag

//redLEDFlag = false;

// Check if the display\_flag is set

if (display\_flag == 1) {

// Send a new line and set the text color to red

UART\_direct\_msg\_put("\n\r\033[31m");

// Dump the memory values

dump\_memory();

// Reset the text color to default and send a new line

UART\_direct\_msg\_put("\033[0m\n\r");

// Reset the display\_flag

display\_flag = 0;

}

}

break; // End of MEMORY case

// Check if the current mode is STACK

case(STACK):

{

// The following line is commented out, but if active, it would turn off a red LED flag

//redLEDFlag = false;

// Check if the display\_flag is set

if (display\_flag == 1) {

// Send a new line and set the text color to cyan

UART\_direct\_msg\_put("\n\r\033[36m");

// Display the last 16 words of the stack

display\_last\_16\_stack\_words();

// Reset the text color to default and send a new line

UART\_direct\_msg\_put("\033[0m\n\r");

// Reset the display\_flag

display\_flag = 0;

}

}

break; // End of STACK case

default:

{

UART\_msg\_put("Mode Error");

}

}

}

## UART\_Poll.cpp

/\*\*-----------------------------------------------------------------------------

\file UART\_poll.cpp

-- --

-- ECEN 5803 Mastering Embedded System Architecture --

-- Project 1 Module 4 --

-- Microcontroller Firmware --

-- UART\_poll.c --

-- --

-------------------------------------------------------------------------------

--

-- Designed for: University of Colorado at Boulder

--

--

-- Designed by: Tim Scherr

-- Revised by: Student's name

--

-- Version: 3.0

-- Date of current revision: 2022-06-20

-- Target Microcontroller: ST STM32F401RE

-- Tools used: ARM mbed compiler

-- ARM mbed SDK

-- ST Nucleo STM32F401RE Board

--

--

-- Functional Description: This file contains routines that support messages

-- to and from the UART port. Included are:

-- Serial() - a routine to send/receive bytes on the UART port to

-- the transmit/receive buffers

-- UART\_put() - a routine that puts a character in the transmit buffer

-- UART\_get() - a routine that gets the next character from the receive

-- buffer

-- UART\_msg\_put() - a routine that puts a string in the transmit buffer

-- UART\_direct\_msg\_put() - routine that sends a string out the UART port

-- UART\_input() - determines if a character has been received

-- UART\_hex\_put() - a routine that puts a hex byte in the transmit buffer

--

-- Copyright (c) 2015, 2022 Tim Scherr All rights reserved.

--

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Configurations \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\*/

#include <stdio.h>

#include "shared.h"

//#include "MKL25Z4.h"

// NOTE: UART0 is also called UARTLP in mbed

// Using USART2 for virtual serial port in STM32F401RE

#define OERR (USART2->SR & USART\_SR\_ORE) // Overrun Error bit

#define CREN (USART2->CR1 & USART\_CR1\_RE) // continuous receive enable bit

#define RCREG USART2->DR // Receive Data Register

#define FERR (USART2->SR & USART\_SR\_FE) // Framing Error bit

#define RCIF (USART2->SR & USART\_SR\_RXNE) // Receive Interrupt Flag (full)

#define TXIF (USART2->SR & USART\_SR\_TXE) // Transmit Interrupt Flag (empty)

#define TXREG USART2->DR // Transmit Data Register

#define TRMT (USART2->SR & USART\_SR\_TC) // Transmit Shift Register Empty

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Start of code \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uchar8\_t error\_count = 0;

/// \fn void serial(void)

/// function polls the serial port for Rx or Tx data

void serial(void) // The serial function polls the serial port for

// received data or data to transmit

{

// deals with error handling first

if ( OERR ) // if an overrun error, clear it and continue.

{

error\_count++;

// resets and sets continous receive enable bit

USART2->CR1 = USART2->CR1 & (!USART\_CR1\_RE);

USART2->CR1 = USART2->CR1 | USART\_CR1\_RE;

}

if ( FERR){ // if a framing error, read bad byte, clear it and continue.

error\_count++;

RCREG; // This will also clear RCIF if only one byte has been

// received since the last int, which is our assumption.

// resets and sets continous receive enable bit

USART2->CR1 = USART2->CR1 & (!USART\_CR1\_RE);

USART2->CR1 = USART2->CR1 | USART\_CR1\_RE;

}

else // else if no frame error,

{

if ( RCIF ) // Check if we have received a byte

{ // Read byte to enable reception of more bytes

// For PIC, RCIF automatically cleared when RCREG is read

// Also true of Freescale KL25Z and STM32F401RE

\*rx\_in\_ptr++ = RCREG; /\* get received character \*/

if( rx\_in\_ptr >= RX\_BUF\_SIZE + rx\_buf )

{

rx\_in\_ptr = rx\_buf; /\* if at end of buffer, circles rx\_in\_ptr

to top of buffer \*/

}

}

}

if (TXIF) // Check if transmit buffer empty

{

if ((tx\_in\_ptr != tx\_out\_ptr) && (display\_mode != QUIET))

{

TXREG = \*tx\_out\_ptr++; /\* send next char \*/

if( tx\_out\_ptr >= TX\_BUF\_SIZE + tx\_buf )

tx\_out\_ptr = tx\_buf; /\* 0 <= tx\_out\_idx < TX\_BUF\_SIZE \*/

tx\_in\_progress = YES; /\* flag needed to start up after idle \*/

}

else

{

tx\_in\_progress = NO; /\* no more to send \*/

}

}

// serial\_count++; // increment serial counter, for debugging only

serial\_flag = 1; // and set flag

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_direct\_msg\_put puts a null terminated string directly

\* (no ram buffer) to the UART in ASCII format.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void UART\_direct\_msg\_put(const char \*str)

{

while( \*str != '\0' )

{

TXREG = \*str++;

while( TXIF == 0 || TRMT == 0 ) // waits here for UART transmit buffer

// to be empty

{

// \_\_clear\_watchdog\_timer();

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_put puts a byte, to the transmit buffer at the location

\* pointed to by tx\_in\_idx. The pointer is incremented circularly as described

\* previously. If the transmit buffer should wrap around (should be designed

\* not to happen), data will be lost. The serial interrupt must be temporarily

\* disabled since it reads tx\_in\_idx and this routine updates tx\_in\_idx which is

\* a 16 bit value.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void UART\_put(uchar8\_t c)

{

\*tx\_in\_ptr++ = c; /\* save character to transmit buffer \*/

if( tx\_in\_ptr >= TX\_BUF\_SIZE + tx\_buf)

tx\_in\_ptr = tx\_buf; /\* 0 <= tx\_in\_idx < TX\_BUF\_SIZE \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_get gets the next byte if one is available from the receive

\* buffer at the location pointed to by rx\_out\_idx. The pointer is circularly

\* incremented and the byte is returned in R7. Should no byte be available the

\* function will wait until one is available. There is no need to disable the

\* serial interrupt which modifies rx\_in\_idx since the function is looking for a

\* compare only between rx\_in\_idx & rx\_out\_idx.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uchar8\_t UART\_get(void)

{

uchar8\_t c;

while( rx\_in\_ptr == rx\_out\_ptr ); /\* wait for a received character,

indicated \*/

// when pointers are different

// this could be an infinite loop, but

// is not because of UART\_input check

c = \*rx\_out\_ptr++;

if( rx\_out\_ptr >= RX\_BUF\_SIZE + rx\_buf ) // if at end of buffer

{

rx\_out\_ptr = rx\_buf; /\* 0 <= rx\_out\_idx < RX\_BUF\_SIZE \*/

// return byte from beginning of buffer

} // next time.

return(c);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_input returns a 1 if 1 or more receive byte(s) is(are)

\* available and a 0 if the receive buffer rx\_buf is empty. There is no need to

\* disable the serial interrupt which modifies rx\_in\_idx since function is

\* looking for a compare only between rx\_in\_idx & rx\_out\_idx.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uchar8\_t UART\_input(void)

{

if( rx\_in\_ptr == rx\_out\_ptr )

return(0); /\* no characters in receive buffer \*/

else

return(1); /\* 1 or more receive characters ready \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_msg\_put puts a null terminated string through the transmit

\* buffer to the UART port in ASCII format.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void UART\_msg\_put(const char \*str)

{

while( \*str != '\0' )

{

\*tx\_in\_ptr++ = \*str++; /\* save character to transmit buffer \*/

if( tx\_in\_ptr >= TX\_BUF\_SIZE + tx\_buf)

tx\_in\_ptr = tx\_buf; /\* 0 <= tx\_in\_idx < TX\_BUF\_SIZE \*/

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_low\_nibble\_put puts the low nibble of a byte in hex through

\* the transmit buffer to the UART port.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//void UART\_low\_nibble\_put(uchar8\_t c)

//{

// UART\_put( hex\_to\_asc( c & 0x0f ));

//}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_high\_nibble\_put puts the high nibble of a byte in h

\* UART port.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//void UART\_high\_nibble\_put(unsigned char c)

//{

// UART\_put( hex\_to\_asc( (c>>4) & 0x0f ));

//}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEX\_TO\_ASC Function

\* Function takes a single hex character (0 thru Fh) and converts to ASCII.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uchar8\_t hex\_to\_asc(uchar8\_t c)

{

if( c <= 9 )

return( c + 0x30 );

return( ((c & 0x0f) + 0x37 )); /\* add 37h \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ASC\_TO\_HEX Function

\* Function takes a single ASCII character and converts to hex.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uchar8\_t asc\_to\_hex(uchar8\_t c)

{

if( c <= '9' )

return( c - 0x30 );

return( (c & 0xdf) - 0x37 ); /\* clear bit 5 (lower case) & subtract 37h \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_hex\_put puts 1 byte in hex through the transmit buffer to

\* the UART port.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void UART\_hex\_put(unsigned char c)

{

UART\_put( hex\_to\_asc( (c>>4) & 0x0f )); // could eliminate & as >> of uchar8\_t

// by definition clears upper bits.

UART\_put( hex\_to\_asc( c & 0x0f ));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The function UART\_direct\_hex\_put puts 1 byte in hex directly (no ram buffer)

\* to the UART.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void UART\_direct\_hex\_put(unsigned char c)

{

TXREG = hex\_to\_asc( (c>>4) & 0x0f );

while( TXIF == 0 )

{

// \_\_clear\_watchdog\_timer();

}

TXREG = hex\_to\_asc( c & 0x0f );

while( TXIF == 0 )

{

// \_\_clear\_watchdog\_timer();

}

}

## Memory.cpp

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

///

/// \file memory.cpp

///

/// \brief Functions to interact with memory, registers, and the UART interface.

///

/// \author Kiran Jojare, Viraj Patel

///

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "memory.h"

#include "mbed.h"

#include "stdio.h"

extern Serial pc;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void read\_serial\_input(char \*buffer, int length)

///

/// \brief Reads input from the serial interface.

///

/// This function captures characters from the UART until a newline or a carriage

/// return character is detected. Additionally, it handles backspaces by erasing

/// the previously entered character.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void read\_serial\_input(char \*buffer, int length) {

int count = 0;

while (count < length - 1) {

while (!pc.readable());

char c = pc.getc();

if (c == '\r' || c == '\n') {

break;

} else if (c == 0x08 || c == 0x7F) { // Backspace detected

if (count > 0) {

count--;

pc.putc(0x08); // Move cursor one position back

pc.putc(' '); // Replace last character with space

pc.putc(0x08); // Move cursor one position back again

}

} else {

pc.putc(c); // Echo back to terminal for other characters

buffer[count++] = c;

}

}

buffer[count] = '\0';

}

/\*

void read\_serial\_input(char \*buffer, int length) {

int count = 0;

while (count < length - 1) {

while (!pc.readable());

char c = pc.getc();

pc.putc(c); // Echo back to terminal

if (c == '\r' || c == '\n') {

break;

}

buffer[count++] = c;

}

buffer[count] = '\0';

}

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R1 register.

\_\_asm uint32\_t get\_r0() {

MOV R0, R0

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R2 register.

\_\_asm uint32\_t get\_r1() {

MOV R0, R1

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R3 register.

\_\_asm uint32\_t get\_r2() {

MOV R0, R2

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R4 register.

\_\_asm uint32\_t get\_r3() {

MOV R0, R3

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R5 register.

\_\_asm uint32\_t get\_r4() {

MOV R0, R4

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R6 register.

\_\_asm uint32\_t get\_r5() {

MOV R0, R5

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R7 register.

\_\_asm uint32\_t get\_r6() {

MOV R0, R6

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R8 register.

\_\_asm uint32\_t get\_r7() {

MOV R0, R7

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R9 register.

\_\_asm uint32\_t get\_r8() {

MOV R0, R8

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R10 register.

\_\_asm uint32\_t get\_r9() {

MOV R0, R9

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R11 register.

\_\_asm uint32\_t get\_r10() {

MOV R0, R10

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R12 register.

\_\_asm uint32\_t get\_r11() {

MOV R0, R11

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of R13 register.

\_\_asm uint32\_t get\_r12() {

MOV R0, R12

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of Stack Pointer register.

\_\_asm uint32\_t get\_sp() {

MOV R0, SP

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of Link Register.

\_\_asm uint32\_t get\_lr() {

MOV R0, LR

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \brief Fetch value of Program Counter register.

\_\_asm uint32\_t get\_pc() {

MOV R0, PC

BX LR

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void print\_registers(void)

///

/// \brief Prints the values of the registers.

///

/// This function sends the current values of the processor's registers over UART.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_registers() {

pc.printf("Printing Register Values\n\r");

pc.printf("R0: 0x%08lX\r\n", get\_r0());

pc.printf("R1: 0x%08lX\r\n", get\_r1());

pc.printf("R2: 0x%08lX\r\n", get\_r2());

pc.printf("R3: 0x%08lX\r\n", get\_r3());

pc.printf("R4: 0x%08lX\r\n", get\_r4());

pc.printf("R5: 0x%08lX\r\n", get\_r5());

pc.printf("R6: 0x%08lX\r\n", get\_r6());

pc.printf("R7: 0x%08lX\r\n", get\_r7());

pc.printf("R8: 0x%08lX\r\n", get\_r8());

pc.printf("R9: 0x%08lX\r\n", get\_r9());

pc.printf("R10: 0x%08lX\r\n", get\_r10());

pc.printf("R11: 0x%08lX\r\n", get\_r11());

pc.printf("R12: 0x%08lX\r\n", get\_r12());

pc.printf("SP: 0x%08lX\r\n", get\_sp());

pc.printf("LR: 0x%08lX\r\n", get\_lr());

pc.printf("PC: 0x%08lX\r\n", get\_pc());

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void dump\_memory(void)

///

/// \brief Dumps memory content from a given address for a given length.

///

/// This function sends the content of the memory from a specified address

/// and for a given length over UART.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void dump\_memory(void) {

/\* Custom Input Code

char input\_buffer[32];

uint32\_t memory\_location;

uint32\_t length;

pc.printf("SP: 0x%08lX\r\n", get\_sp());

pc.printf("Enter memory location in hex format (e.g., 0x20010000): ");

read\_serial\_input(input\_buffer, sizeof(input\_buffer));

sscanf(input\_buffer, "%lx", &memory\_location);

pc.printf("\r\nEnter length: ");

read\_serial\_input(input\_buffer, sizeof(input\_buffer));

sscanf(input\_buffer, "%lu", &length);

pc.printf("\r\nYou entered location: 0x%08lX and length: %lu\r\n", memory\_location, length);

\*/

uint32\_t memory\_location = 0x20010000;

uint32\_t length = 64;

pc.printf("Dumping memory at Location 0X%08X %d Bytes:\r\n", memory\_location, length);

uint32\_t start\_address = memory\_location ;

const uint32\_t bytes\_per\_line = 16;

uint8\_t \*ptr = (uint8\_t \*)start\_address;

for (uint32\_t i = 0; i < length; i += bytes\_per\_line) {

// Print the memory address

pc.printf("0x%08lX: ", (uint32\_t)(ptr + i));

// Print the hex values

for (uint32\_t j = 0; j < bytes\_per\_line; j++) {

if (i + j < length) {

pc.printf("%02X ", ptr[i + j]);

} else {

pc.printf(" "); // for alignment when length is not a multiple of bytes\_per\_line

}

}

pc.printf(" ");

// Print the ASCII values

for (uint32\_t j = 0; j < bytes\_per\_line; j++) {

if (i + j < length) {

char c = ptr[i + j];

if (c < 32 || c > 126) { // non-printable chars

c = '.';

}

pc.printf("%c", c);

}

}

pc.printf("\r\n");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/// \fn void display\_last\_16\_stack\_words(void)

///

/// \brief Displays the last 16 words in the stack.

///

/// This function sends the last 16 words of the current stack over UART.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display\_last\_16\_stack\_words() {

uint32\_t \*stack\_ptr = (uint32\_t \*)get\_sp();

pc.printf("Print Stack \n\r");

pc.printf("Current Stack Pointer SP: 0x%08lX\r\n", stack\_ptr);

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

pc.printf("Word %d: 0x%08X : %08X\r\n", 16 - i, stack\_ptr - i,\*(stack\_ptr-i));

}

}