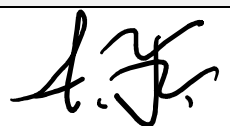


<b>Course Title:</b>	Embedded Systems Design
<b>Course Number:</b>	COE718
<b>Semester/Year (e.g.F2016)</b>	F2021

<b>Instructor:</b>	Sunbal Cheema
--------------------	---------------

<i>Assignment/Lab Number:</i>	Project Final Report
<i>Assignment/Lab Title:</i>	Mutexes and Semaphores for Avoiding Priority Inversion

<i>Submission Date:</i>	Friday, December 3rd, 2021
<i>Due Date:</i>	Friday, December 3rd, 2021

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Fahmy	Ahmad	500913092	4	

\*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report 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: <http://www.ryerson.ca/senate/current/pol60.pdf>

## 1. Introduction:

In this project we will be using minimal input and output specifications since we have limited access to an in-person board, namely an ARM Cortex M-3, to grow our knowledge of real-time Operating Systems, the board itself, and  $\mu$ Vision.

## 2. Objective:

We are tasked with creating an embedded application which performs multiple tasks, such as calculating the summation of a first-order Maclaurin Series and will execute the corresponding task based on user-input. We will implement  $\mu$ Vision functionalities such as solving Priority Inversion using semaphores and mutexes, and Joinable Threads to accomplish our goals in the most efficient manner possible while maintaining these functionalities.

## 3. Methods and Files:

\*Check appendix for code

### Files and their Methods:

**IRQ.c** - premade

#### **joystickFiles.c**

JoystickGetPos ()

- Uses Joystick\_GetState () to map the joystick position values to

#### **Task1.c**

recVarsToTask1 ()

- Receives input values in Task1.c and sets its global variables to the value of the corresponding arguments

factorial ()

- function that computes the factorial value of the received argument

calcSeries ()

- calculates the sum of the entire series using the above methods/functions and the math.h library functions

runTask1 ()

- starts computation for Thread 1 using the above functions/methods

#### **Task2.c**

recVarsToTask3 ()

- Receives input values in Task1.c and sets its global variables to the value of the corresponding arguments

hex2Dec ()

- Converts the hex code in char form into decimal form

decAssoc ()

- Associate the arguments in decimal RGB form into their hex color name (of the 16 basic HEX colors) using a series of switch-case statements

hexColorConvRUN ()

- starts computation for the Thread 2 using the above functions/methods

### **Task3.c**

recVarsToTask3 ()

- Receives input values in Task1.c and sets its global variables to the value of the corresponding arguments

planetChoose ()

- Uses the user inputs chosen planet to set the values of the planet mass, radius, air density and gravitational acceleration

allCalc ()

- Calculates the values of the escape velocities, the mass of fuel required to reach escape velocity, and the total mass of the rocket with fuel using the above functions/methods and math.h library functions

escapeVelAndMass ()

- Starts computation for Thread 3 using the above functions/methods

### **threadTasks.c**

Init\_task3JoinThreads (priorInvCh)

- creates Tasks using their ID's and sets the priority of each task

MutThreads ()

- creates and sets up the conditions for Mutex multithreading using osMutex... functions

SemThreads ()

- creates and sets up the conditions for Semaphore multithreading using osSemaphore... functions

RSemThreads ()

- creates and sets up the conditions for Resemble Semaphore multithreading using if...then conditions, switch-case statements, and the global variable resSemToken to allow each Thread to compute when it 'has' the Semaphore

threadPrioritySet ()

- receives the user's choice for the priority of the tasks and changes the priority of the Tasks using osThreadDef () and varying priority settings

Thread1, ..., Thread 3 ()

- uses if...then conditions to use the correct priority inversion method and computes/runs the thread functionality for Thread1, ..., Thread 3

## **joinThreads.c**

thread1a ()

- computes the value of the numerator for all values of 1-n

thread1b ()

- computes the total value of the denominators (all factorials) from 1-n

thread1c ()

- computes the total value of each series function by dividing the numerator and denominators of corresponding n values, given the numerator/denominator != NULL → have been computed already

computSum ()

- computes the total sum of the whole series from 1-n, after all threads have finished computing

joinThreadsRUN ()

- runs the functionality of Task 1 in 3 separate threads using the above functions/methods

## **projectMain.c**

Int main (void)

- Runs the main methods and programmed commands for the program

Joystick\_Initialize () - premade

- Initializes and starts/prepares up the joystick for user-input

Joystick\_GetState () - premade

- Gets the state of the joystick and outputs a corresponding value
  - o Values were mapped and can be seen in Tbl. 3

PrintJALL ()

- Prints all the possible joystick positions, their corresponding pin, and their input value (1-16)

osKernalInitialize () - premade

- Initializes and starts up OS for thread use

osKernalStart () - premade

- Starts the Kernel to multithread

ifWhileVarGrab ()

- Uses JoystickGetPos () to get inputs and then.....

scanTask1, ..., scanTask3

- Collects all the necessary values for each Task

passVarsToTasks () & passVarsToTask1, ..., and passVarsToTask3

- Passes user input values, mapped using JoystickGetPos (), to Tasks for calculation
- recVars... - Receives variables

mutexSemOrResSEM ()

- collects the user input for choosing the Mutex, Semaphore or Resemble Semaphore versions of the code, does this by setting the value for global variable priorInvCh to 1, 2, or 3

hexAssoc ()

- associates the argument integer to one of the corresponding basic 16 HEX colors (Tbl. 4)  
HEX value in char form

planetAssoc ()

- associates the argument integer to one of the corresponding 8 planets in our solar system (Tbl. 5b)

repeatEntriesTask1, ..., repeatEntriesTask3

- Repeats the user entries for each task
- Used for debugging and testing

printf () – premade

delay ()

- Custom delay function that delays computation for its received argument

osDelay () – premade

- os... delay function

#### 4. **Figures:**

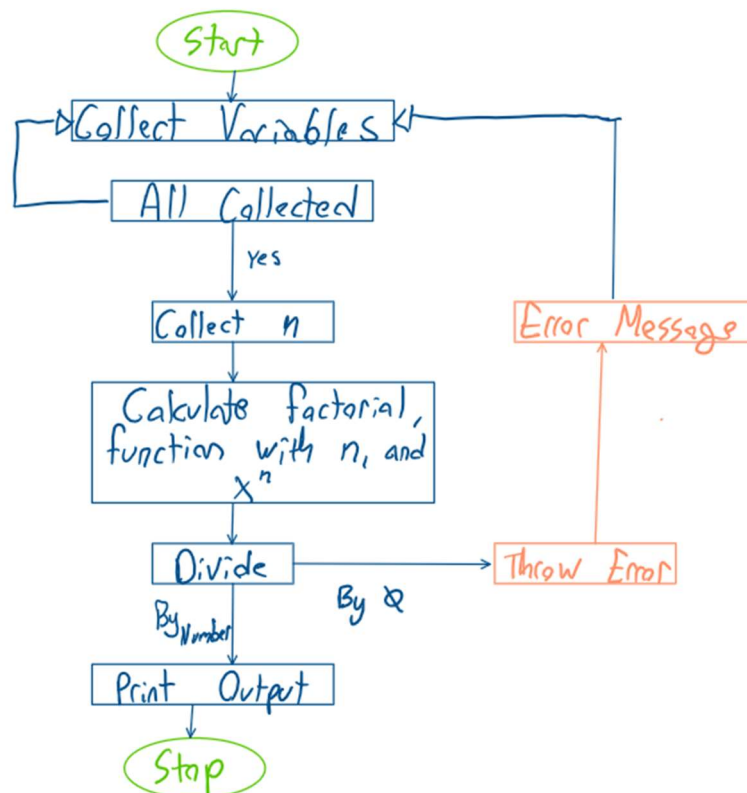


Figure 1a: Flow Chart for Task 1

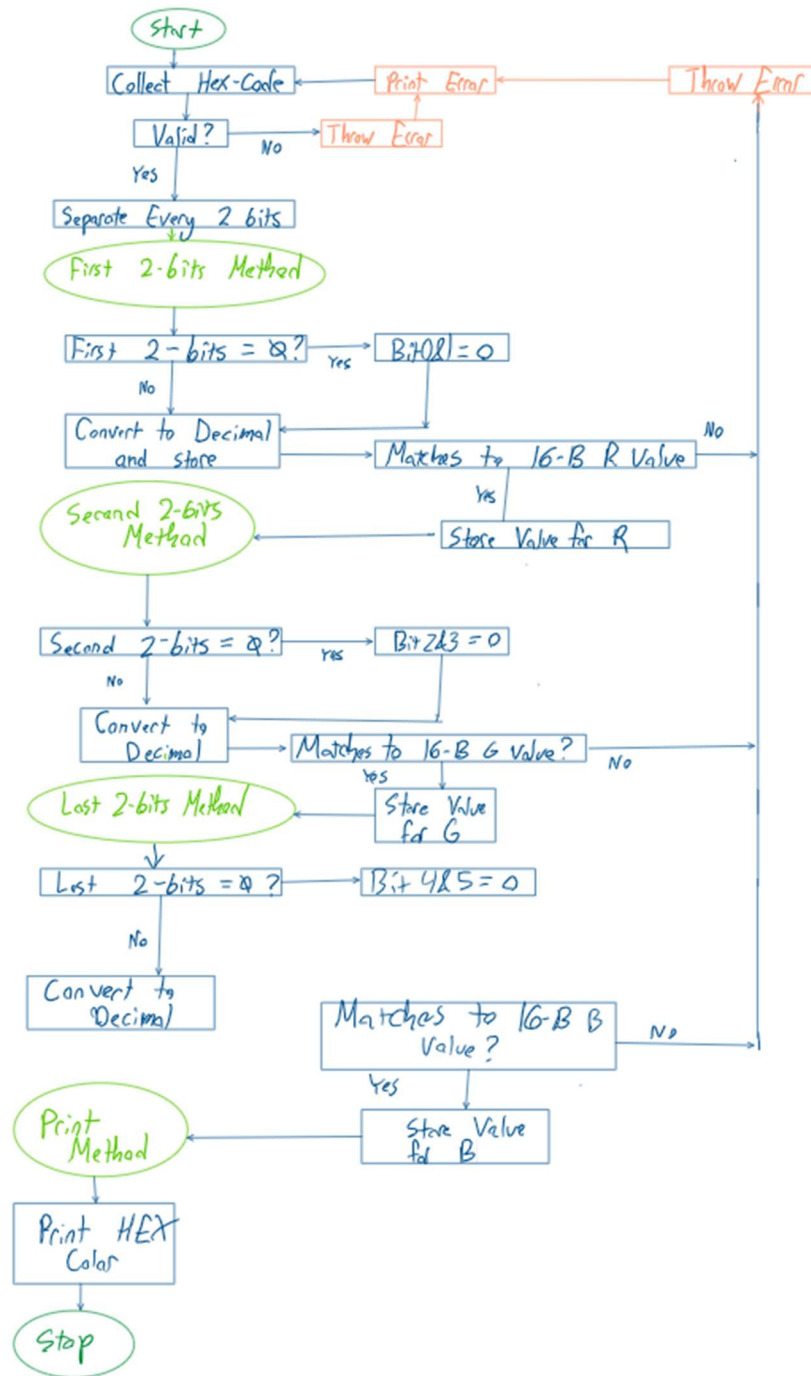


Figure 1b: Flow Chart for Task 2

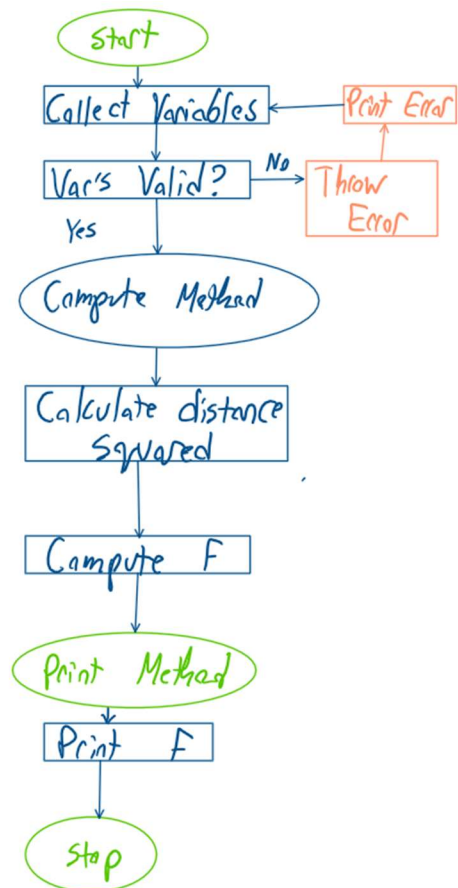


Figure 1c: Flow Chart for Task 3

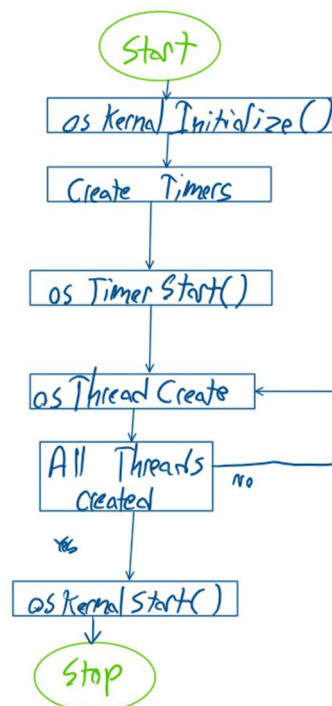


Figure 1d: Flow Chart for Thread Initialization, Creation and Start



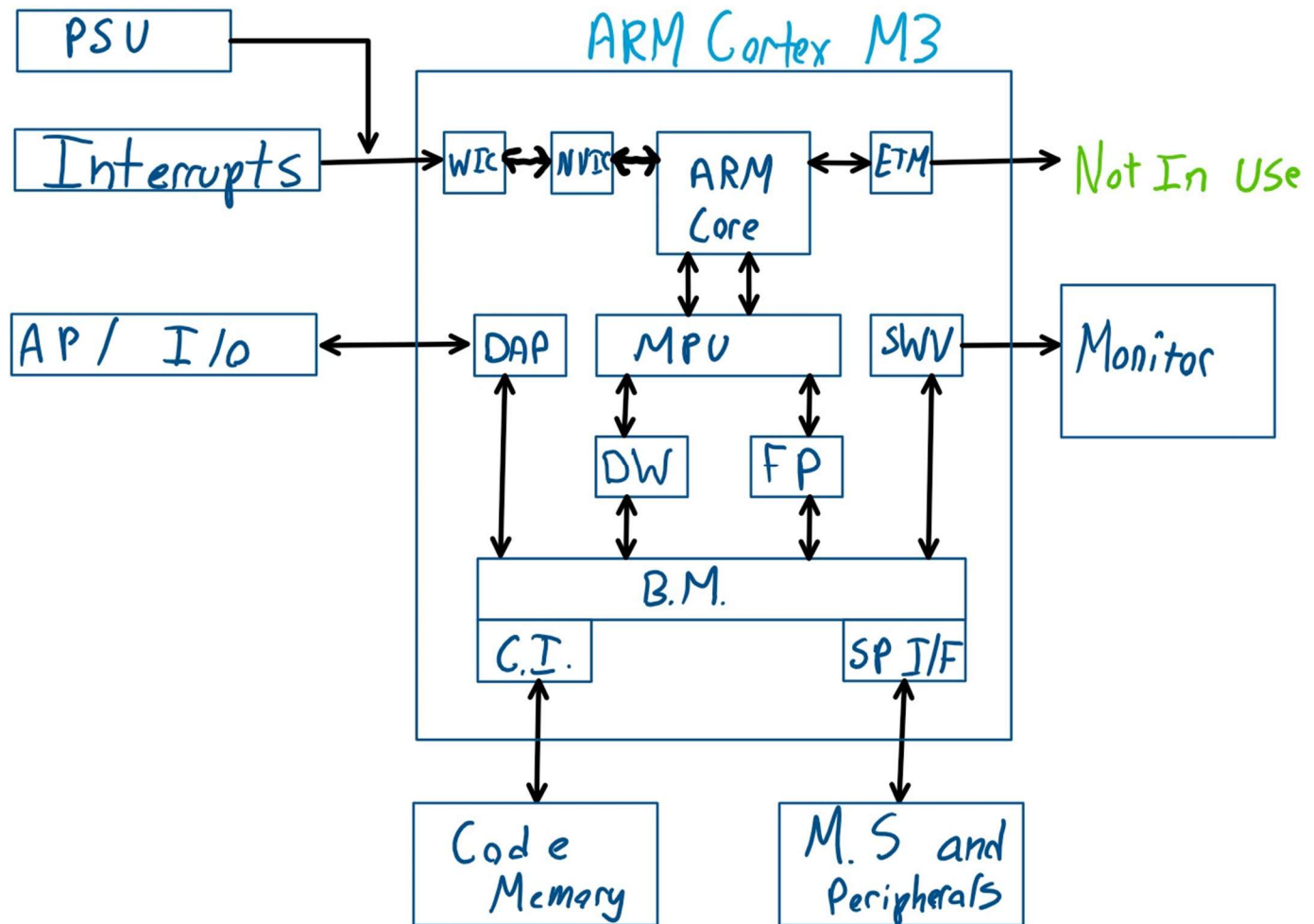


Figure 2: Arm Cortex-M3 Block Diagram

Item	Sub-Item	Acronym	Description
<b>ARM Cortex M3</b>			32-bit RISC ARM Processor Cores
	<b>ARM Core (Armv7-M)</b>		32-bit Processing Core
	<b>Memory Protection Unit</b>	MPU	Provides memory protection
	<b>Data Watchpoints</b>	DW	Breakpoint that indicates an area of the memory that is a specific data item to be watched
	<b>Flash Patch</b>	FP	Patches code and data from Code Space
	<b>Bus Matrix</b>	BM	Connects the processor and debug interface to the external busses

	<b>Code Interface</b>	CI	Interfaces with the Code Memory for use in the processor cores
	<b>SRAM and Peripheral I/F</b>	SP I/F	Static random-access memory (SRAM) is a volatile and fast memory used for cache and internal registers. Peripheral I/F allows communication to external peripherals such as keyboards, mice, and Memory Systems.
	<b>Serial Wire Viewer</b>	SWV	Data trace feature to display output
	<b>Debug Access Port</b>	DAP	Allows connection to an external Debugger and Access Port's to access on-chip system resources.
	<b>Nested Vector Interrupt Control</b>	NVIC	Prioritizes high-priority Status Flag Registers (N, V, I, and C) for use in programming
	<b>Embedded Trace Macrocell</b>	ETM	N/A
	<b>Wake Up Interrupt Controller</b>	WIC	Peripheral that detects Interrupts
<b>Interrupts</b>			A signal from a peripheral/program that requires the OS to stop and resolve the activated flag/interrupt
<b>Access Ports</b>		AP	A communication endpoint for host devices
<b>Monitor</b>			Displays information
<b>Code Memory</b>			The location in which the Computer Programs are stored
<b>Power Supply Unit</b>		PSU	Supplies power to the unit
<b>Memory System and Peripherals</b>		M.S and Peripherals	Memory Data and Peripherals such as mice, keyboards, etc.

Table 1: Block Diagram Item Descriptions and Acronyms

Task	Function	Input	Constants	Output
<b>1 (Joint Threads)</b>	$\sum_{n=1}^{\infty=n_{max}} \frac{2^n + 2^2}{n!}$	$n$	N/A	Sum of series with input n
<b>2</b>	Convert Hex Code to the corresponding Color Name of the 16 basic colors	Hex Code (#xxxxxx = 0xYYYYYY = XXXXXX)	N/A	Hex-Code corresponding Color Name

<b>3</b>	$v_e = \sqrt{\frac{2GM}{r}}$ $m_{RT} = m_R(e^{\Delta v/v_{ex}})$ $m_f = m_{RT} - m_R$ (Mass of fuel)	<u>Chosen Planet P: 1-8</u> <u>decides the</u> <u>following:</u>  M <sub>P</sub> (mass of planet)  r <sub>p</sub> (Radius of planet)  ρ (density of air for planet)  a (gravitational acceleration of planet) or use = g = 9.8 m/s <sup>2</sup>	G = 6.674E10 <sup>-11</sup> Nm <sup>2</sup> /kg <sup>2</sup>  Δv = v <sub>e</sub> - 0 = v <sub>e</sub> (Change in velocity, from rest to escape velocity)  m <sub>R</sub> (mass of rocket and payload) or use = 2000kg standard payload + rocket mass)  r <sub>R</sub> (radius of rocket) = 1m  V <sub>ex</sub> (exhaust velocity) or use = 5,000 m/s (standard exhaust velocity)	m <sub>f</sub> (mass of fuel)  v <sub>e</sub> (Escape velocity of the chosen planet)
----------	--	---	--	--

Table 2: Tasks/Main-Threads and their Functionality

<b>GPIO</b>	<b>Joystick_GetState () Value</b>	<b>JoystickGetPos () Value</b>	<b>Corresponding Direction</b>
N/A	0	1	Nothing Selected
20	4	2	Select
23	8	3	Up
24	2	4	Right
25	16	5	Down
26	1	...	Left
23, 24	10		Up-Right
23, 26	9		Up-Left
25, 26	17		Down-Left
24, 25	18		Down-Right
20, 23	12		Select-Up
20, 24	6		Select-Right

20, 25	20		Select-Down
20, 26	5	...	Select-Left
20, 23, 24	14	15	Select-Up-Right
20, 23, 26	13	16	Select-Up-Left
20, 25, 26	21	N/A	Select-Down-Left
20, 24, 25	22	N/A	Select-Down-Right
Impossible Configuration (ex. 24, 26 or 20, 23, 25)	3, 7, 11, 15, 19, 23, 24, 25, 26, 27, 28, 29, 30, 31 (Ex. 3 and 28)	N/A	Examples: Right-Left and Select-Up-Down

Table 3: Table Displaying Joystick Peripheral Mapping

Color	Name	Hex Code (RGB)	Decimal Values (R, G, B)	joystickPos ( ) Value:
	White	#FFFFFF	255, 255, 255	14
	Silver	#C0C0C0	192, 192, 192	15
	Gray	#808080	128, 128, 128	1
	Black	#000000	0, 0, 0	0
	Red	#FF0000	255, 0, 0	2
	Maroon	#800000	128, 0, 0	3
	Yellow	#FFFF00	255, 255, 0	11
	Olive	#808000	128, 128, 0	10
	Lime	#00FF00	0, 255, 0	13
	Green	#008000	0, 128, 0	12
	Aqua	#00FFFF	0, 255, 255	6
	Teal	#008080	0, 128, 128	7
	Blue	#0000FF	0, 0, 255	4
	Navy	#000080	0, 0, 128	5
	Fuchsia	#FF00FF	255, 0, 255	9
	Purple	#800080	128, 0, 128	8

Table 4: HEX Code Conversion for Task 2

Value:	Mass	$r_R$ (radius of rocket)	$V_{ex}$
Standard Rocket	2,000 kg	1m	5,000 m/s

Table 5a: Solar System Planet Masses, and inputs

Value:	Mass kg * $10^{24}$	$r_p$ (radius = distance from crust to core) m	$\rho$ (density of air for planet) kg/m <sup>3</sup>	$a$ (gravitation acceleration) m/s <sup>2</sup>
Mercury:	0.33011	2440500	0.02	3.7
Venus:	4.8675	6051800	65	8.87
Earth:	5.97237	6378137	1.2	9.8

<b>Mars:</b>	6.4171*10 <sup>-1</sup>	3389500	0.02	3.721
<b>Jupiter:</b>	1.8982*10 <sup>3</sup>	71492000	0.16	24.79
<b>Saturn</b>	5.6834*10 <sup>2</sup>	60268000	0.19	10.44
<b>Uranus:</b>	86.813	25559000	0.42	8.87
<b>Neptune:</b>	102.413	24764000	0.45	11.15

Table 5b: Solar System Planet Masses, and inputs

## 5. Design and Methodology:

Design of the program was done so to maximize organization and reduce redundancy, so code was programmed using the fastest possible methods (math.h pow () function instead of always doing the power using conventional operators) and methods were used to sort each desired task (Hex to Decimal, compute series, etc.) to allow repeated functionality without having to repeat code (instead one just had to call the function with the desired variables). Switch-case statements were used to reduce the need for long nested if...then ladders, however, if...then and while () conditions were used in short environments or in cases in which switch-cases were not optimal. Additionally, for the user-input portion of the program, inputs were received using the joystick methods and delays between each input, to allow the user ample time to select the desired input. Similarly, certain functionalities were put into methods to keep the program neat and organized, as well as to reduce the need to repeat code. The input variables were received and mapped using the joystickFiles.c, ifWhileVarGrab (), and passVarsToTasks1...3 () (and their subsequent method calls) methods to collect, send, and receive the methods in the corresponding Task files (Task1...3.c), to be computed using Task1...3. c's methods once the threads began computing. Then, the priority order of the functions was received and set in threadTasks.c using threadPrioritySet (). Next, the Priority Inversion method was chosen using mutexSemOrResSEM (). Lastly, the threads were created in threadTasks.c using Init\_task3JoinThreads (), and then the threads started computing with the command osKernelStart (). This differed with the joint threads which used Round Robin, First Come First Served scheduling algorithms, and join-threads in Task 1 to compute the tasks.

## 6. Past Work - Interim Report and Review:

### Interim Report - November 9<sup>th</sup>, 2021

#### Progress Made:

Progress made in the project thus far is start-up conditional-execution code that asks the user for input, the completion of Tasks 1, 2 and the partial completion of Task 3 (Fully/partially completed Tasks can be seen in Table 2). The code begins by asking the User for their desired Task to be executed, what mode they would like to execute the chosen Task in, etc. The joystick has also been implemented by powering up the peripheral with "LPC\_SC->PCONP|= (1 << 15);" and initializing the peripheral by using the Joystick\_Initialization () function which enables the GPIO clock and configures the pins. Additionally, to read the current position and state of the Joystick we used Joystick\_Stats () which returned the current position of the joystick in integer form. Further, to then print this value and its corresponding positions; the joystick position variations were mapped to their corresponding integer values (see Table 3) and put into a switch...case statement to print the corresponding directions using the printf() function, while physically impossible variations (such as Up-Down or Select-Left-Right) were sent to the 'default' portion of the switch statement as mapping them would've been a waste of system memory, data, and lines of code.

Task 1 accepts variables and coefficients for a Maclaurin Series (Table 2) and calculates the value at  $n=x$ , where  $x$  is a user-input value as well. The Task uses self-programmed functions such as power functions, factorials, a sum-for-loop, etc. Task 2 uses a series of if-then conditions to breakdown the user-input hex code into its correct category, then using the 2 HEX-bits converts them into binary to calculate the percentage of R, G, and B (as seen in Fig.4). The correct decoded basic-HEX-color (of the total 16, Tbl. 4) is then presented on the Monitor display. Task 3 implemented Isaac Newton's Formula for the Law of Universal Gravitation (as seen in Table 2). This was simply done by accepting the user-input masses and the distance between the objects, then followed the formula in multiplying, squaring, and dividing the variables using a programmed Macro to represent the constant  $G$  (Gravitational Constant).

As a Priority Inversion (PI) prevention method Mutexes and Semaphores were used, these allow us to solve the issues caused by PI in which critical shared resources are being accessed by a thread of lower-priority thus blocking a thread of higher-priority. The code is split into two versions in which one uses Mutexes and the other uses Semaphores, and then further implementing Semaphores without using the built-in RTX Semaphore functions. So far, the Mutexes have been implemented while the Semaphores have not. Block Diagrams were drawn (Figure 2) including the vital ARM Cortex-M3 cores and modules, inputs/outputs, memory, and peripherals. The descriptions, acronyms, and names of all the components used can be seen in Table 1. Lastly, flow charts were drawn for each completed Task to present the logical steps that is used to go from one step to the next.

#### Future-Plans:

Future-plans are to complete Task 3 and ensure its functionality, start and complete Tasks 4 and 5, implement Semaphores using the build-in Semaphores to further implement self-programmed Semaphores, and then create Joinable Threads for all the Tasks. As well as creating flow charts for the remaining Tasks and functionalities. The Tasks will be divided into multiple threads so they can run simultaneously, and so it will be necessary to join these threads together in a manner so to terminate all joined threads at the same time by making them wait for the other incomplete threads. This function of Joinable Threads will further be used so to allow non-implementation-specific sharing of thread-stacks, heap memory and so on to promote the sharing of resources which does not necessarily improve performance but can reduce the need for repeating certain functions and calculations.

### **Review - December 3<sup>rd</sup>, 2021**

Much of the design for the program remained the same but the complexity of Task 1 was greatly reduced by removing the ability for the user to input information using the keyboard using the `scanf()` function. The main reason for this was due to the compiler compiling instantly for some reason and not giving the user ample time to input their values, this may have been due to an error in the `delay()` function but as time was waning the functionality was removed in all and instead simpler Tasks (Tbl. 2) were implemented. One main removal was the ability for the user to include a function in the summation, and instead of computing a Maclaurin Series to instead compute a summation of a series. This removed the need for the user to have to use the keyboard to input a function and removed the need to manually program the derivative functions for any possible function, as well as removing the need to implement a function to read and break apart the user string input and convert it into a function using BEDMAS (brackets and powers included) rules into something

computable by the CPU. These requirements increased the complexity exponentially and would have been possible given enough time but that was not the case.

## 7. Experimental Results:

<b>Input Variable:</b>	<b>Input Value:</b>
<b>Priority 1:</b>	3
<b>P 2:</b>	1
<b>P 3:</b>	2
<b>Planet #:</b>	3
<b>Value of n:</b>	3
<b>HEX code Association #:</b>	3
<b>Mutex (1), Semaphores (2), or Res Sem (3)</b>	1

Table 6a: Inputs used for computation times

<b>Method:</b>	<b>Total Computation Time (ms):</b>	<b>Task 1 (ms)</b>	<b>Task 2 (ms)</b>	<b>Task 3 (ms)</b>
<b>Mutexes</b>	0.5946	0.298	0.15	0.1466
<b>Semaphores</b>	0.601	~	~	~
<b>Resemble Semaphores</b>	0.630	~	~	~
<b>Joint Threads, FCFS</b>	0.643	0.311	~	~
<b>Joint Threads, RR</b>	0.612	0.311	~	~

Table 6: Computation Time with varying Multithreading Synchronization Methods and Joint Threads First Come First Served (FCFS) and Round Robin (RR) Tested with inputs in Tbl. 6a

## 8. Conclusion:

Based on the total computation times for the whole program and the tasks (not counting the user-input and delay times) it seems the fastest multithreading method was mutexes with 0.5946 milliseconds, and by using joint threads Task 1's computation time was increased from the fastest multithreading method's computation time, from 0.5946 to 0.612 in Round Robin. Of course, the multithreading synchronization methods aren't entirely for decreasing or increasing computation time, instead they are for decreasing the likelihood of the Priority Inversion problem and increasing synchronization between the tasks. The other computation times were as they appear in Table. 6. The three different types of multithreading synchronization methods were done to analyze the similarities between all three, and as such; Mutexes are a method for mutual exclusion – as one task operates the others must wait until the Mutex key is released. Semaphores are generalized mutexes in those different parts of the shared resources (Semaphore) can be used by different tasks 'simultaneously' but cannot use the same resource until the Semaphore is released. Lastly, Resemble Semaphores were programmed using a global variable and if...then conditions to check if the task was allowed to compute at that time depending on if it had the Resemble Semaphore Key/Global Variable which was implemented by assigning an integer value to each Task and checking if the global variable was equal to that corresponding integer value when wishing to run the thread.

## 9. References:

- [1] C. Griwodz, "Semaphores and other Wait-and-Signal mechanisms - PDF," University of Oslo, [Online]. Available: <https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF3150/h03/annet/slides/semaphores.pdf>. [Accessed 2 12 2021].
- [2] CMSIS, "Semaphores," [Online]. Available: [https://www.keil.com/pack/doc/cmsis/RTOS/html/group\\_\\_CMSIS\\_\\_RTOS\\_\\_SemaphoreMgmt.html](https://www.keil.com/pack/doc/cmsis/RTOS/html/group__CMSIS__RTOS__SemaphoreMgmt.html). [Accessed 2 12 2021].
- [3] CMSIS, "Mutexes," [Online]. Available: [https://www.keil.com/pack/doc/cmsis/RTOS/html/group\\_\\_CMSIS\\_\\_RTOS\\_\\_MutexMgmt.html](https://www.keil.com/pack/doc/cmsis/RTOS/html/group__CMSIS__RTOS__MutexMgmt.html). [Accessed 2 12 2021].
- [4] CMSIS, "Thread Management," CMSIS, [Online]. Available: [https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group\\_\\_CMSIS\\_\\_RTOS\\_\\_ThreadMgmt.html](https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__ThreadMgmt.html). [Accessed 2 12 2021].

## 10. Appendix (Code):

### projectMain.c

```
#include <wchar.h>

extern int Init_task3JoinThreads();
extern int recVarsToTask3();
extern int recVarsToTask2();
extern int recVarsToTask1();
extern int joystickGetPos();
extern void threadPrioritySet();

#include <stdio.h>
#include "projectMain.h"
#include "LPC17xx.h"           // Device header
#include "Board_LED.h"         // ::Board Support:LED
#include "Board_ADC.h"         // ::Board Support:A/D Converter
#include "Board_Joystick.h"    // ::Board Support:Joystick
#include <stdbool.h>
#include <string.h>
#include "cmsis_os.h"          // CMSIS RTOS header file
#include <stdlib.h>
#include <math.h>
#include "osObjects.h"         // RTOS object definitions

//----- ITM Stimulus Port definitions for printf ----- //
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port16(n)   (*((volatile unsigned short*)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))

#define DEMCR            (*((volatile unsigned long *)(0xE000EDFC))
#define TRCENA            0x01000000
```



```

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

//----- //

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

volatile unsigned long * bit;

#define GPIO_Bit28 (*((volatile unsigned long *)0x23380670)) //GPIO 1.28  CENTER  --->2.0
#define GPIO_Bit2  (*((volatile unsigned long *)0x23380A08)) //GPIO 2.2    UP
#define GPIO_Bit3  (*((volatile unsigned long *)0x23380E0C)) //GPIO 3.3    DOWN  --->2.3
#define GPIO_Bit4  (*((volatile unsigned long *)0x23381210)) //GPIO 4.4    LEFT  --->2.4
#define GPIO_Bit5  (*((volatile unsigned long *)0x23380214)) //GPIO 0.5    RIGHT --->2.5

#define LPC_GPIO(n)      ((LPC_GPIO_TypeDef *) (LPC_GPIO0_BASE + 0x00020*n))

#define PI 3.14
// #define RHO 207129
#define BLANK "\u2580"

#define osObjectsPublic          // define objects in main module

static char text[10];

static volatile uint16_t AD_dbg; //Variable to trace in LogicAnalyzer (should not read to often)

uint16_t ADC_last; // Last converted value

char jPosMain[68];

char hexCode[3][2];
char hexColor[6];

```

```

int taskOneInput;
int taskTwoInput;
int taskThreeInputINT;

char taskThreeInputVar[4][3] = { {"Mr"}, {"Rr"}, {"Vex"}, {"P"} };
char planet[8][8] = { {"Mercury"}, {"Venus"}, {"Earth"}, {"Mars"}, {"Jupiter"}, {"Saturn"}, {"Uranus"}, {"Neptune"} };

char colorName[16][8] = {"White", "Silver", "Gray", "Black", "Red", "Maroon", "Yellow", "Olive", "Lime", "Green", "Aqua", "Teal", "Blue", "Navy", "Fuchsia", "Purple"};
char colorHex[16][7] = {"FFFFFF", "C0C0C0", "808080", "000000", "FF0000", "800000", "FFFF00", "808000", "00FF00", "008000", "00FFFF", "008080", "0000FF", "000080", "FF00FF", "800080"};
int correspColor[16] = {14, 15, 1, 0, 2, 3, 11, 10, 13, 12, 6, 7, 4, 5, 9, 8};

double taskThreeInput[4];

int scanIn;
int taskChoice[5];
int taskOrder[3];
int mSRSVar;
int priorInvCH = 0;
int k = 0;
int planetNumMain;

int a=0, b=0, c=0, d=0, e=0;

long long int decimalNum[3];

bool running = 1;
bool taskChosenYet = 0;
bool taskCHOSEN[6];
bool boolTVars[3];

//////////////////////////////////// Methods/Functions
//custom delay function for e sconds

```

```

void delay(unsigned int e){
    int c, d, m;
    e = e*30000; //30000

    for (c = 0; c <= e/6; c++) {
        for (d = 0; d <= e; d++){

        }
        printf("-");
        //printf("delay");
    }
    printf("\nDONE DELAY \n\n");
}

//terminate program
void terminate(){
    printf("Program Terminating in 5s\n");
    delay(5000);
    running = 0;
}

//invalid entry
void invalidEntry(){
    printf("You entered %d\n INVALID ENTRY, PLEASE TRY AGAIN\n", scanIn);
    taskChosenYet = 0;
}

void printJALL(){
    printf("The possible Joystick configurations used for inputs are: ");
    printf("No Pins on | N/A == 0 ");
    printf("UP | P1.20 == 4 ");
    printf("RIGHT | P1.23 == 8 ");
    printf("DOWN | P1.24 == 2 ");
    printf("LEFT | P1.25 == 16 ");
    printf("UP-RIGHT | P1.26 == 1 ");
    printf("UP-LEFT | 23, 24 == 10 ");
    printf("DOWN-LEFT | 23, 26 == 9 ");
    printf("DOWN-RIGHT | 25, 26 == 17 ");
    printf("SELECT-UP | 20, 23 == 12 ");
    printf("SELECT-RIGHT | 20, 24 == 6 ");
    printf("SELECT-DOWN | 20, 25 == 20 ");
    printf("SELECT-LEFT | 20, 26 == 5 ");
    printf("SELECT-UP-RIGHT | 20, 23, 24 == 14 ");
    printf("SELECT-UP-LEFT | 20, 23, 26 == 13 ");
    printf("SELECT-DOWN-LEFT | 20, 25, 26 == 21 ");
    printf("SELECT-DOWN-RIGHT | 20, 24, 25 == 22 ");
}

```

```

void hexAssoc(int x){
    switch(x){
        case 1: //WHITE
            hexCode[0][0] = 'F'; hexCode[0][1] = 'F'; hexCode[1][0] = 'F'; hexCode[1][1] =
'F'; hexCode[2][0] = 'F'; hexCode[2][1] = 'F';

            case 2: //SILVER
                hexCode[0][0] = 'C'; hexCode[0][1] = '0'; hexCode[1][0] = 'C'; hexCode[1][1] =
'0'; hexCode[2][0] = 'C'; hexCode[2][1] = '0';

            case 3: //GRAY
                hexCode[0][0] = '8'; hexCode[0][1] = '0'; hexCode[1][0] = '8'; hexCode[1][1] =
'0'; hexCode[2][0] = '8'; hexCode[2][1] = '0';

            case 4: //BLACK
                hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 5: //RED
                hexCode[0][0] = 'F'; hexCode[0][1] = 'F'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 6: //MAROON
                hexCode[0][0] = '8'; hexCode[0][1] = '0'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 7: //YELLOW
                hexCode[0][0] = 'F'; hexCode[0][1] = 'F'; hexCode[1][0] = 'F'; hexCode[1][1] =
'F'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 8: //OLIVE
                hexCode[0][0] = '8'; hexCode[0][1] = '0'; hexCode[1][0] = '8'; hexCode[1][1] =
'0'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 9: //LIME
                hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = 'F'; hexCode[1][1] =
'F'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 10: //GREEN
                hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = '8'; hexCode[1][1] =
'0'; hexCode[2][0] = '0'; hexCode[2][1] = '0';

            case 11: //AQUA
                hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = 'F'; hexCode[1][1] =
'F'; hexCode[2][0] = 'F'; hexCode[2][1] = 'F';
    }
}

```

```

        case 12: //TEAL
            hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = '8'; hexCode[1][1] =
'0'; hexCode[2][0] = '8'; hexCode[2][1] = '0';

        case 13: //BLUE
            hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = 'F'; hexCode[2][1] = 'F';

        case 14: //NAVY
            hexCode[0][0] = '0'; hexCode[0][1] = '0'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = '8'; hexCode[2][1] = '0';

        case 15: //FUCHSIA
            hexCode[0][0] = 'F'; hexCode[0][1] = 'F'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = 'F'; hexCode[2][1] = 'F';

        case 16: //PURPLE
            hexCode[0][0] = '8'; hexCode[0][1] = '0'; hexCode[1][0] = '0'; hexCode[1][1] =
'0'; hexCode[2][0] = '8'; hexCode[2][1] = '0';
    }
}

void planetAssoc(int x){
    switch(x){
        case 1:
            planetNumMain = 1;

        case 2:
            planetNumMain = 2;

        case 3:
            planetNumMain = 3;

        case 4:
            planetNumMain = 4;

        case 5:
            planetNumMain = 5;

        case 6:
            planetNumMain = 6;

        case 7:
            planetNumMain = 7;

        case 8:
            planetNumMain = 8;
    }
}

```

```

    }
}

//REPEAT inputs
void repeatEntriesTask1(){

    printf("%d\n", taskOneInput);
}

void repeatEntriesTask2(char inputTask2[3][2]){
    printf("0x");
    for(int l = 0; l < 3; l++){
        for(int m=0; m < 2; m++) {
            printf("%c", inputTask2[l][m]);
        }
    }
    printf("\n");
}

void repeatEntriesTask3(double inputTask3[4]){
    for(int l = 0; l < 4; l++){
        printf("%lf", inputTask3[l]);
    }
    printf("\n");
}

void repeatEntriesTask4(int tempmSRSVar){

    printf("%d", tempmSRSVar);
    printf("\n");
}

////SCAN inputs
void scanTask1(){
    printf("Task 1\n_____ \n Enter the value of n (1-16) you'd like use in the sum, using
the joystick\n"); //TASK 1 get input Var's

    delay(20);

    taskOneInput = joystickGetPos();
}

void scanTask2(){
    printf("Task 2\n_____ \n Enter the hex CODE's associated number (1-16) using the
joystick\n"); //TASK 2 get input Var's

    delay(20);
}

```

```

    hexAssoc(joystickGetPos());
}

void scanTask3(){
    printf("\nTask 3\n_____ \n Choose the number of the planet (1-8)\n"); //TASK 3 get
input Var's

    delay(20);

    planetAssoc(joystickGetPos());
}

//GRAB VARS FROM USER
void ifWhileVarGrab() {
    int task1VarsChosen=0, task2VarsChosen=0, task3VarsChosen=0, allTaskVars=0;

    while(allTaskVars == 0) {
        if(taskOrder[0] == 1 && taskOrder[1] == 2 && taskOrder[2] == 3) {
            scanTask1();
            //repeatEntriesTask1();

            scanTask2();
            //repeatEntriesTask2(hexCode);

            scanTask3();
            //repeatEntriesTask3(taskThreeInput);

            allTaskVars = 1;
        }

        if(taskOrder[0] == 1 && taskOrder[1] == 3 && taskOrder[2] == 2) {
            scanTask1();
            // repeatEntriesTask1();

            scanTask3();
            //repeatEntriesTask3(taskThreeInput);

            scanTask2();
            //repeatEntriesTask2(hexCode);

            allTaskVars = 1;
        }
    }
}

```

```

if(taskOrder[0] == 2 && taskOrder[1] == 1 && taskOrder[2] == 3) {
    scanTask2();
    //repeatEntriesTask2(hexCode);

    scanTask1();
    //repeatEntriesTask1();

    scanTask3();
    //repeatEntriesTask3(taskThreeInput);

    allTaskVars = 1;
}

if(taskOrder[0] == 2 && taskOrder[1] == 3 && taskOrder[2] == 1) {
    scanTask2();
    //repeatEntriesTask2(hexCode);

    scanTask3();
    //repeatEntriesTask3(taskThreeInput);

    scanTask1();
    //repeatEntriesTask1();

    allTaskVars = 1;
}

if(taskOrder[0] == 3 && taskOrder[1] == 1 && taskOrder[2] == 2) {
    scanTask3();
    //repeatEntriesTask3(taskThreeInput);

    scanTask1();
    //repeatEntriesTask1();

    scanTask2();
    //repeatEntriesTask2(hexCode);

    allTaskVars = 1;
}

if(taskOrder[0] == 3 && taskOrder[2] == 2 && taskOrder[2] == 1) {
    scanTask3();
    //repeatEntriesTask3(taskThreeInput);

    scanTask2();

```



```

        //repeatEntriesTask2(hexCode);

        scanTask1();
        //repeatEntriesTask1();

        allTaskVars = 1;
    }
}

///PASSING VARS to TASKS
void passVarsToTask1(int taskOneInput){
    recVarsToTask1(taskOneInput);
}

void passVarsToTask2(char hexCode[3][2]){
    recVarsToTask2(hexCode);
}

void passVarsToTask3(/*double taskThreeInput[4]*/ int planetNumMain){
    recVarsToTask3(planetNumMain);
}

void passVarsToTasks(/*int taskOneInput, char hexCode[3][2], double taskThreeInput[4]*/){
    passVarsToTask1(taskOneInput);
    passVarsToTask2(hexCode);
    passVarsToTask3(planetNumMain);
}

void mutexSemOrResSEM(/*int tempmSRSVar[3]*/){
    printf("Would you like to run Mutex (1), Semaphore (2), or Resemble Semaphore (3)?\n\n");
    printf("Please enter in the form: GPIO nothing (1) or 1.23 (UP - 2) or 1.24 (RIGHT - 3) \n\n using\n\n");
    printf("only pins: nothing, 23, and 24");

    delay(20);
    mSRSVar = joystickGetPos();

    //while(priorInvCH == 0) {
        switch(mSRSVar){
            case 1: { //MUTEX
                //repeatEntriesTask4(mSRSVar);
                printf("MODE: MUTEX\n\n");
                priorInvCH = 1;

                break;
            }
        }
    }
}

```

```

        case 2: { //SEM
                    //repeatEntriesTask4(mSRSVar);
                    printf("MODE: SEMAPHORE\n\n");
                    priorInvCH = 2;

                    break;
                }

        case 3: { //RESSEM
                    //repeatEntriesTask4(mSRSVar);
                    printf("MODE: RESEMBLE SEMAPHORE\n\n");
                    priorInvCH = 3;

                    break;
                }
    }
}

//}

}

/*-----
Main function
-----*/
int main (void) {
    int32_t res;

    uint32_t AD_sum    = 0U;
    uint32_t AD_cnt    = 0U;
    uint32_t AD_value  = 0U;
    uint32_t AD_print  = 0U;

    LED_Initialize();           // LED Initialization
    ADC_Initialize();           // ADC Initialization
    Joystick_Initialize();       // Joystick Initialization

    LPC_SC->PCONP      |= (1 << 15);           // Powering Up Joystick

    /* P1.20, P1.23..26 is GPIO (Joystick) */
    LPC_PINCON->PINSEL3 &= ~( (3<< 8) | (3<< 14) | (3<< 16) | (3<< 18) | (3<< 20));

    /* P1.20, P1.23..26 ids input */
    LPC_GPIO2->FIODIR   &= ~( (1<<20) | (1<<23) | (1<<24) | (1<<25) | (1<<26));

    SystemCoreClockUpdate();
    SysTick_Config(SystemCoreClock/100U); // Generate interrupt each 10 ms

    printf("WELCOME TO AHMAD FAHMY's, 500913092, COE718 FINAL PROJECT\n");
    printf("TASK 1 : 0 | Calculate the Series of up till input n\n");

```

```

printf("TASK 2 : 1 | Convert one of the 16 basic HEX COLOR CODEs to COLOR name\n");
for(int s =0; s<16; s++){
    printf("    (%d) color HEX:%s    color:", s+1, colorHex[s]);

    //textcolor(correspColor[s]);
    printf("    color name: %s\n", colorName[s]); //\u2580

}

char o = NULL;
int k = 0;

printf("TASK 3 : 2 | Calculates the excape velocity (Ve), Mass of fuel (Mf) and Max
Altitude (Ya) using Joinable Threads for \n");
printf("a rocket of mass (2000kg), on a planet (P) of your choice in the solar system
from 1-8 corresponding to the planet's \n");
printf("ranked position compared to the others. Mercury = 1, Venus = 2, Earth = 3 ...
Uranus = 8\n");

printf("    Planet Details:\n_____ \n");
for(int z = 0; z<8; ){
    printf("    %s (%d):", planet[z], z);

    printf("    Mp (mass): X, Rp (radius): Y, RHO (air density): Z, and a
(gravitational acceleration): A\n");/////////////////////////CHANGE RHO
    //printf(" (%d): Mp (mass): X, Rp (radius): Y, RHO (air density), and a
(gravitational acceleration): Z\n", s+1);
    z++;
}

//TEST//printf("%d %d %d %d", correspColor[0], correspColor[1], correspColor[2],
correspColor[3]);
/* //PRINT ASCII's
int x = 0;
for(;;){
    printf(" %x %c\n ", x, x);
    x++;
}
*/

printf("TERMINATE : 99\n");
printJALL();

printf("\nPlease enter the order/priority of tasks:\n GPIO nothing (0)\n 1.23 (UP - 1)
\n 1.24 (RIGHT - 2)\n using only pins: nothing, 23, and 24\n");
Joystick_Initialize();

```

```

    delay(20);
    taskOrder[0] = joystickGetPos();
    delay(20);
    taskOrder[1] = joystickGetPos();
    delay(20);
    taskOrder[2] = joystickGetPos();

    osKernelInitialize ();                // initialize CMSIS-RTOS
    printf("\n\n");
    delay(20);

    ifWhileVarGrab();                    // user enters the input-variables for each Task
    //passVarsToTasks(taskOneInput, hexCode, taskThreeInput);
    passVarsToTasks();

    delay(20);

    mutexSemOrResSEM();

    threadPrioritySet(taskOrder);
    Init_task3JoinThreads(/*taskOrder,*/ priorInvCH);

    osKernelStart ();                    // start thread execution
    osDelay(osWaitForever);
}

```

## ThreadTasks.c

```

#include "cmsis_os.h"                    // CMSIS RTOS header file
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "LPC17xx.h"
#include <stdbool.h>

extern int runTask1();
extern int hexColorConvRUN();
extern int escapeVelAndMass();
extern void delay();

//----- ITM Stimulus Port definitions for printf ----- //
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port16(n)   (*((volatile unsigned short*)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))

#define DEMCR            (*((volatile unsigned long *)(0xE000EDFC)))

```

```

#define TRCENA                0x01000000

/*
struct __FILE { int handle; };
FILE __stdout;
FILE __stdin;

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

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

//volatile unsigned long * bit;

#define GPIO_Bit28 (*((volatile unsigned long *)0x23380670)) //GPIO 1.28

#define GPIO_Bit2    (*((volatile unsigned long *)0x23380A08)) //GPIO 2.2

#define GPIO_Bit3    (*((volatile unsigned long *)0x23380E0C)) //GPIO 3.3

#define GPIO_Bit4    (*((volatile unsigned long *)0x23381210)) //GPIO 4.4

#define GPIO_Bit5    (*((volatile unsigned long *)0x23380214)) //GPIO 0.5

#define LPC_GPIO(n)    ((LPC_GPIO_TypeDef *) (LPC_GPIO0_BASE + 0x00020*n))

unsigned int A=0;
unsigned int x=0;
unsigned int i=0;

unsigned int nB=0;
unsigned int j = 0;
int resSemToken = 0;

double B=0;
double C=0;
double nC=0;

```

```

double D=0;
double E=0;

char taskOneInputThread[101];
char hexCodeThread[3][2];
double taskThreeInputThread[4];

bool boolMUT[3] = { 0, 0, 0};
bool semaphorLive = 0;
bool resSemaphoreLive = 0;
    bool doneTask[3] = {0, 0, 0};

//for threads
void Thread1 (void const *argument); // thread function
void Thread2 (void const *argument); // thread function
void Thread3 (void const *argument); // thread function

osMutexDef(threadMut);
osMutexId(threadMut_id);

//for sems
osSemaphoreId semaphore;
/*
osSemaphoreId semThread1;
osSemaphoreId semThread2;
osSemaphoreId semThread3;
*/

osThreadId tid1_Thread, tid2_Thread, tid3_Thread; // thread id

osThreadDef (Thread1, osPriorityNormal, 1, 0);
osThreadDef (Thread2, osPriorityNormal, 1, 0);
osThreadDef (Thread3, osPriorityNormal, 1, 0);

osSemaphoreDef(semaphore);

void threadPrioritySet(int taskOrder[3]){
    int prioritySet = 0;
    //while(prioritySet == 0) {
        if(taskOrder[0] == 1 && taskOrder[1] == 2 && taskOrder[2] == 3) {
            osThreadDef (Thread1, osPriorityRealtime, 1, 0); //
thread object, priority +3 == 1 in manual
            osThreadDef (Thread2, osPriorityHigh, 1,
0); // thread object, priority +2 == 2 in manual
            osThreadDef (Thread3, osPriorityAboveNormal, 1, 0); //
thread object, priority +1 == 3 in manual
            prioritySet = 1;

```

```

        printf("ORDER SET: %d, %d, %d", taskOrder[0], taskOrder[1],
taskOrder[2]);
    }

    else if(taskOrder[0] == 1 && taskOrder[1] == 3 && taskOrder[2] == 2) {
        osThreadDef (Thread1, osPriorityRealtime, 1, 0); //
thread object, priority +3 == 1 in manual
        osThreadDef (Thread3, osPriorityHigh, 1,
0); // thread object, priority +2 == 2 in manual
        osThreadDef (Thread2, osPriorityAboveNormal, 1, 0); //
thread object, priority +1 == 3 in manual
        prioritySet = 1;
        printf("ORDER SET: %d, %d, %d", taskOrder[0], taskOrder[1],
taskOrder[2]);
    }

    else if(taskOrder[0] == 2 && taskOrder[1] == 1 && taskOrder[2] == 3) {
        osThreadDef (Thread2, osPriorityRealtime, 1, 0); //
thread object, priority +3 == 1 in manual
        osThreadDef (Thread1, osPriorityHigh, 1,
0); // thread object, priority +2 == 2 in manual
        osThreadDef (Thread3, osPriorityAboveNormal, 1, 0); //
thread object, priority +1 == 3 in manual
        prioritySet = 1;
        printf("ORDER SET: %d, %d, %d", taskOrder[0], taskOrder[1],
taskOrder[2]);
    }

    else if(taskOrder[0] == 2 && taskOrder[1] == 3 && taskOrder[2] == 1) {
        osThreadDef (Thread2, osPriorityRealtime, 1, 0); //
thread object, priority +3 == 1 in manual
        osThreadDef (Thread3, osPriorityHigh, 1,
0); // thread object, priority +2 == 2 in manual
        osThreadDef (Thread1, osPriorityAboveNormal, 1, 0); //
thread object, priority +1 == 3 in manual
        prioritySet = 1;
        printf("ORDER SET: %d, %d, %d", taskOrder[0], taskOrder[1],
taskOrder[2]);
    }

    else if(taskOrder[0] == 3 && taskOrder[1] == 1 && taskOrder[2] == 2) {
        osThreadDef (Thread3, osPriorityRealtime, 1, 0); //
thread object, priority +3 == 1 in manual
        osThreadDef (Thread1, osPriorityHigh, 1,
0); // thread object, priority +2 == 2 in manual
        osThreadDef (Thread2, osPriorityAboveNormal, 1, 0); //
thread object, priority +1 == 3 in manual

```

```

        prioritySet = 1;
        printf("ORDER SET: %d, %d, %d", taskOrder[0], taskOrder[1],
taskOrder[2]);
    }

    else if(taskOrder[0] == 3 && taskOrder[1] == 2 && taskOrder[2] == 1) {
        osThreadDef (Thread3, osPriorityRealtime, 1, 0); //
thread object, priority +3 == 1 in manual
        osThreadDef (Thread2, osPriorityHigh, 1,
0); // thread object, priority +2 == 2 in manual
        osThreadDef (Thread1, osPriorityAboveNormal, 1, 0); //
thread object, priority +1 == 3 in manual
        prioritySet = 1;
        printf("ORDER SET: %d, %d, %d", taskOrder[0], taskOrder[1],
taskOrder[2]);
    }

    else {
        printf("Invalid Priority Settings, Try again\n");
        prioritySet = 0;
    }

    //}
}

void MutThreads(){
    boolMUT[0] = 1;
    boolMUT[1] = 1;
    boolMUT[2] = 1;

    threadMut_id = osMutexCreate(osMutex(threadMut));
}

void SemThreads(){
    semaphorLive = 1;
    semaphore = osSemaphoreCreate(osSemaphore(semaphore), 1);
}

void RSemThreads(){
    resSemaphoreLive = 1;
}

int Init_task3JoinThreads(/*int taskOrder[3],*/ int priorInvCHThread) {
    tid1_Thread = osThreadCreate (osThread(Thread1), NULL);
    tid2_Thread = osThreadCreate (osThread(Thread2), NULL);

```



```

    tid3_Thread = osThreadCreate (osThread(Thread3), NULL);
    //if(!tid1_Thread) return(-1);

    //choose MODE
    switch(priorInvCHThread){
        case 1:{
            MutThreads(); // chooses MUTEX
        }

        case 2:{
            SemThreads(); // chooses SEMAPHORES
        }

        case 3:{
            RSemThreads(); // chooses RES SEMAPHORES
        }
    }

    return(0);
}

void Thread3 (void const *argument) { //TASK C
    while(doneTask[2] == 0){
        if(boolMUT[2] == 1){
            //osMutexWait(threadMut_id, osWaitForever); //for MUT 1
            osMutexWait(threadMut_id, 1); //for MUT 1
            printf("RUNNING 3");
            escapeVelAndMass();
            osMutexRelease(threadMut_id);
        }

        if(semaphorLive == 1){
            osSemaphoreWait(semaphore, osWaitForever);
            escapeVelAndMass();
            osSemaphoreRelease(semaphore);
        }

        if(resSemaphoreLive == 1){
            if(resSemToken == 0){
                resSemToken = 3;
            }

            if(resSemToken == 3){
                escapeVelAndMass();
                resSemToken = 0;
            }
        }
    }
}

```

```

        doneTask[2] = 1;
    }
}

void Thread2 (void const *argument) { //TASK B
    while(doneTask[1] == 0){
        if(boolMUT[1] == 1){
            //osMutexWait(threadMut_id, osWaitForever);
            osMutexWait(threadMut_id, 2);
            printf("RUNNING 2");

            hexColorConvRUN();
            osMutexRelease(threadMut_id);
        }

        if(semaphorLive == 1) {
            delay(10);
            int val = osSemaphoreWait(semaphore, 2);
            if(val > 0){
                hexColorConvRUN();
                osSemaphoreRelease(semaphore);
            }
        }

        if(resSemaphoreLive == 1){
            delay(10);
            if(resSemToken == 0){
                resSemToken = 2;
            }

            if(resSemToken == 2){
                hexColorConvRUN();
                resSemToken = 0;
            }
        }
        doneTask[1] = 1;
    }
}

void Thread1 (void const *argument) { //TASK A
    while(doneTask[0] == 0){
        if(boolMUT[0] == 1){
            //osMutexWait(threadMut_id, osWaitForever);
            osMutexWait(threadMut_id, 3);
            printf("RUNNING 1");
            runTask1();
            osMutexRelease(threadMut_id);
        }
    }
}

```

```

    }

    else if(semaphorLive == 1) {
        delay(10);
        int val = osSemaphoreWait(semaphore, 1);
        if(val > 0){
            runTask1();
            osSemaphoreRelease(semaphore);
        }
    }

    else if(resSemaphoreLive == 1) {
        delay(10);
        if(resSemToken == 0){
            resSemToken = 1;
        }

        if(resSemToken == 1){
            runTask1();
            resSemToken = 0;
        }
    }
    doneTask[0] = 1;
}
}

```

## Task1.c

```

#include <stdio.h>
#include <string.h>
#include <math.h>

char taskOneInputTASK[101];

double inputDoubles[5];
double extraDerivDoubleAll = 1;
char varsWPow[10][3];
char function[10];
char derivsAllAND0;
char inputVar;

int finalOut = 0;
int fullSizeVar = 0;
int n = 0;
double seriesTot = 0;

```

```

double seriesN = 0;

/*
void fullSize(){
    int i=0;

    while(fullSizeVar<i){
        if(taskOneInputTASK[i] == NULL && taskOneInputTASK[i+1] == NULL){
            fullSizeVar = i;
            i++;
        }
    }
}
*/

/*
void recVarsToTask1(char taskTempOneInputTASK[101]){
    for(int i = 0; i < fullSizeVar; i++){
        taskOneInputTASK[i] = taskTempOneInputTASK[i];
    }
}*/

/*
void breakApart(){
    int i=0;
    int currentVarWPow = 0;
    int fullSize = 0;

    while(fullSize < fullSizeVar){ //runs through all

        //runs through all until space (32) and (array cell behind value not a different
type OR array cell behind not 0*/)
        while(taskOneInputTASK[i] != 32 && (taskOneInputTASK[i] != taskOneInputTASK[i-1] ||
((i-1) == 0)*/*)){
            //for(int i=0; i < sizeof(&taskOneInputTASK); i++){
                if(taskOneInputTASK[i] == '0'){ //check if double //needs to change
                    inputDoubles[0] = taskOneInputTASK[i];
                }

                if(taskOneInputTASK[i] == 'c'){ // check if char //needs to
change
                    if(taskOneInputTASK[i+1] == 'c'){ ////check if double

                }
            }
        }
    }
}

```

```

    }

    else { //IS char or operator
        if(taskOneInputTASK[i] == 42){ ////multiply operator

        }

        if(taskOneInputTASK[i] == 43){ ////add operator

        }

        if(taskOneInputTASK[i] == 45){ ////subtract operator

        }

        if(taskOneInputTASK[i] == 47){ ////divide operator

        }

        else if( (taskOneInputTASK[i]>='a' && taskOneInputTASK[i]<='z') ||
(taskOneInputTASK[i]>='A' && taskOneInputTASK[i]<='Z')){ // IS var
            inputVar = taskOneInputTASK[i];
            if(taskOneInputTASK[i+1] == 94){
                while(taskOneInputTASK[i] != 32 && (taskOneInputTASK[i] !=
taskOneInputTASK[i-1] || ((i-1) == 0)*//*)){
                    for(int d=0; d<3; d++){
                        varsWPow[currentVarWPow++][d] = taskOneInputTASK[i];
                    }
                }
            }
        }
    }
}

void derivAND0(int current){

    //do derivative of x
    derivsAllAND0 = varsWPow

    extraDerivDoubleAll = extraDerivDoubleAll * (varsWPow[current][2]-1); //put the power
brought down here and multiply all together

}

void calcMac(){ ///you need to program BEDMAS

```

```

    int i=0;
    while(i < sizeof(taskOneInputTASK)){
        if(taskOneInputTASK[i] == inputVar){
            derivAND0(i);

        }

        finalOut += finalOut;
    }
}
*/

long factorial(int nFact){
    int fact =1;
    for(int i=1;i<nFact+1;i++){
        fact=fact*i;
    }
    return fact;
}

void recVarsToTask1(int tempN){
    n = tempN;
}

void calcSeries(){
    for(int i=1; i <n+1;i++){
        seriesN = (pow(2, i)+4);

        seriesN = seriesN / factorial(i);
    }
    seriesTot = seriesTot + seriesN;
}

void runTask1(){
    //fullSize();
    //breakApart();
    //calcMac();

    calcSeries();

    printf("the sum of the series with n= %d is %lf", n, seriesTot);

    //
}

```

## Task2.c

```
#include <stdio.h>

char taskTwoInputTASK[3][2];
char hexColorTask[6];
char hexCodeTask[3][2];

long long int decimalNumTask[3];

void recVarsToTask2(char taskTempTwoInputTASK[3][2]){
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 2; j++){
            taskTwoInputTASK[i][j] = taskTempTwoInputTASK[i][j];
        }
    }
}

void hex2Dec(char hex[3][2], long long int decimal[3]) { /////convert 2-bits hex to decimal
    long long base = 1;
    int value, length;

    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 2; j++){
            if(hex[i][j] <= '0' && hex[i][j] >= '9')
            {
                decimal[i] += (hex[i][j] - 48) * base;
                base *= 16;
            }
            else if(hex[i][j] <= 'A' && hex[i][j] >= 'F')
            {
                decimal[i] += (hex[i][j] - 55) * base;
                base *= 16;
            }
            else if(hex[i][j] <= 'a' && hex[i][j] >= 'f')
            {
                decimal[i] += (hex[i][j] - 87) * base;
                base *= 16;
            }
        }
    }
}

void decAssoc(long long int decimal[3], char color[6]){
    switch(decimal[0]) {
```

```

// 00 xx xx | Black, Lime, Green, Aqua, Teal, Blue, or Navy
case 0: { // 0

    switch(decimal[1]) {
        // 00, 00 xx | Black, Blue, or Navy
        case 0: { // 0

            switch(decimal[2]) {
                // 00 00, 00 | Black
                case 0: { // 0
                    color = "Black";
                    break;
                }

                // 00 00, 80 | Navy
                case 128: { // 0x80
                    color = "Navy";
                    break;
                }

                // 00 00, FF | Blue
                case 256: { // 0xFF
                    color = "Blue";
                    break;
                }
            }
        }
    }

    // 00, 80 xx |
    case 128: { // 0x80

        switch(decimal[2]) {
            // 00 80, 00 | Green
            case 0: // 0
                color = "Green";
                break;

            // 00 80, 80 | Teal
            case 128: // 0x80
                color = "Teal";
                break;
        }
    }

    case 256: { // 0xFF

        switch(decimal[2]) {

```



```

        // 00 FF, 00 | Lime
        case 0: //0
            color = "Lime";
            break;

        // 00 FF, FF | Aqua
        case 256: //0xFF
            color = "Aqua";
            break;
    }
}

// 80 xx xx | Gray, Olive, Maroon, and Purple
case 128: { // 0x80
    switch(decimal[1]) {
        // 80, 00 xx
        case 0: {

            switch(decimal[2]) {
                // 80 00, 00 | Maroon
                case 0: // 0
                    color = "Maroon";
                    break;
            }

            // 80 00, 80 | Purple
            case 128: // 0x80
                color = "Purple";
                break;
        }
    }

    switch(decimal[1]) {
        //80, 80 xx
        case 128: // 0x80

            switch(decimal[2]) {
                // 80 80, 00 | Olive
                case 0: // 0
                    color = "Olive";
                    break;

                // 80 80, 80 | Gray
                case 128: // 0x80
                    color = "Gray";
                    break;
            }
        }
    }
}

```

```

    }
}

}

// C0 xx xx
case 192: { // 0xC0

    // C0, C0 xx
    if (decimal[1] == 192) { // 0xC0

        //C0 C0, C0
        if(decimal[2] == 192) { // 0xC0
            color = "Silver";
            break;
        }
    }
}

// FF xx xx
case 256: { // 0xFF

    switch(decimal[1]) {
        //FF, 00 xx
        case 0: // 0

            switch(decimal[2]) {
                // FF 00, 00 | Red
                case 0: // 0
                    color = "Red";
                    break;

                // FF 00, FF | Fuchsia
                case 256: // 0xFF
                    color = "Fuchsia";
                    break;
            }
        //FF, FF xx
        case 256: // 0xFF

            switch(decimal[2]) {

                // FF FF 00
                case 0: // 0
                    color = "yellow";
                    break;
            }
        }
    }
}

```



```

}

case 6: { //aqua - cyan
    printf("\033[1;36m");
    printf("AQUA");
    break;
}

case 7: { //teal
    printf("\033[0;36m");
    printf("TEAL");
    break;
}

case 8: { //magenta - purple
    printf("\033[0;35m");
    printf("PURPLE");
    break;
}

case 9: { //fusch
    printf("\033[1;35m");
    printf("FUCHSIA");
    break;
}

case 10: { //olive
    printf("\033[0;33m");
    printf("OLIVE");
    break;
}

case 11: { //yellow
    printf("\033[1;33m");
    printf("YELLOW");
    break;
}

case 12: { //green
    printf("\033[0;32m");
    printf("GREEN");
    break;
}

case 13: { //lime
    printf("\033[1;32m");
    printf("LIME");

```

```

        break;
    }

    case 14: { //white
        printf("\033[1;37m");
        printf("WHITE");
        break;
    }

    case 15: { //light gray
        printf("\033[0;37m");
        printf("SILVER");
        break;
    }

    case 16: //reset
        printf("\033[0m");
        break;
    }
}
*/

void hexColorConvRUN() {
    hex2Dec(taskTwoInputTASK, decimalNumTask); //decimal number recieved
    decAssoc(decimalNumTask, hexColorTask);

    printf("The Associated Hex color to");
    for(int i=0; i < 4; i++) { ///print HEX CODE
        for(int j=0; j < 2; j++) {
            printf(" %c ", hexCodeTask[i][j]);
        }
    }
    printf("is %s", hexColorTask); //print COLOR
}

```

### Task3.c

```

#include <stdio.h>
#include <stdbool.h>
#include <math.h>

#define G (6.674*(pow(10, -11)))

double planetVars[4];

```

```

bool planetChosen = 0;

//double taskThreeInputTASK[1];
int planetNum;

double output[3];

void recVarsToTask3(/*double taskTempThreeInputTASK[4]*/ int x){
    //for(int i = 0; i < 4; i++){
        // taskThreeInputTASK[i] = taskTempThreeInputTASK[i];
    //}
    planetNum = x;
}

void planetChoose(P){ //put all the numbers for each PLANET
    while(planetChosen == 0) {
        switch(P){
            case 1:{ //Mercury
                planetVars[0] = 3.3011*pow(10,23);
                planetVars[1] = 4880*1000/2;
                planetVars[2] = 0.020;
                planetVars[3] = 3.7;
                planetChosen = 1;
            }

            case 2:{ //Venus
                planetVars[0] = 4.8675*pow(10,24);
                planetVars[1] = 6051.8*1000;
                planetVars[2] = 65;
                planetVars[3] = 8.87;
                planetChosen = 1;
            }

            case 3:{ //Earth
                planetVars[0] = 5.97237*pow(10,24);
                planetVars[1] = 6378.137*1000;
                planetVars[2] = 1.2;
                planetVars[3] = 9.8;
                planetChosen = 1;
            }

            case 4:{ //Mars
                planetVars[0] = 6.4171*pow(10,23);
                planetVars[1] = 3389.5*1000;
                planetVars[2] = 0.020;
                planetVars[3] = 3.721;
            }
        }
    }
}

```

```

        planetChosen = 1;
    }

    case 5:{ //Jupiter
        planetVars[0] = 1.8982*pow(10,27);
        planetVars[1] = 71492*1000;
        planetVars[2] = 0.16;
        planetVars[3] = 24.79;
        planetChosen = 1;
    }

    case 6:{ //Saturn
        planetVars[0] = 5.6834*pow(10,26);
        planetVars[1] = 60268*1000;
        planetVars[2] = 0.19;
        planetVars[3] = 10.44;
        planetChosen = 1;
    }

    case 7:{ //Uranus
        planetVars[0] = 86.813*pow(10,24);
        planetVars[1] = 25559*1000;
        planetVars[2] = 0.42;
        planetVars[3] = 8.87      ;
        planetChosen = 1;
    }

    case 8:{ //Neptune
        planetVars[0] = 102.413*pow(10,24);
        planetVars[1] = 24764*1000;
        planetVars[2] = 0.45;
        planetVars[3] = 11.15;
        planetChosen = 1;
    }

    default:
        printf("You entered %d\n INVALID ENTRY, PLEASE TRY AGAIN\n", P);
//confirmation
        planetChosen = 0;
    }
}

void allCalc(){
    //Escape Vel
    output[0] = sqrt((2*G*planetVars[0])/planetVars[1]);

```

```

    //total mass of rocket
    //output[1] = taskThreeInputTASK[0]* exp((output[0] / taskThreeInputTASK[2]));
    output[1] = 2000* exp((output[0] / 5000));

    //mass of fuel
    //output[2] = output[1] - taskThreeInputTASK[0];
    output[2] = output[1] - 2000;
}

void escapeVelAndMass(){
    planetChoose(planetNum); //chooses the corresponding planet
    allCalc();

    printf("Your necessary Escape Velocity is %lf: ", output[0]); printf("m/s \n");
    printf("The total mass of your rocket with fuel should be approximately %lf: ",
output[1]); printf("kg \n");
    printf("The total mass of rocket fuel should be approximately %lf: ", output[2]);
printf("kg \n");
}

```

## JoystickFiles.c

```

#include "Board_Joystick.h" // ::Board Support:Joystick
#include <string.h>
#include <stdio.h>

int joystickGetPos(/*char position[]*/) {
    switch(Joystick_GetState()){
        case 0:{
            //strcpy(position, "No Direction, No Select");
            printf("1 - No Direction, No Select\n");
            return(1);
            break;
        }

        case 8:{
            //strcpy(position, "Up");
            printf("2 - Up\n");
            return(2);
            break;
        }

        case 2:{
            //strcpy(position, "Right");
            printf("3 - Right\n");

```



```

        return(3);
    break;
}

case 16:{
    //strcpy(position, "Down");
    printf("4 - Down\n");
    return(4);
break;
}

case 1:{
    //strcpy(position, "Left");
    printf("5 - Left\n");
    return(5);
break;
}

case 4:{
    //strcpy(position, "Select");
    printf("6 - SELECT\n");
    return(6);
break;
}

case 10:{
    //strcpy(position, "Up-Right");
    printf(" 7 - Up-Right\n");
    return(7);
break;
}

case 9:{
    //strcpy(position, "Up-Left");
    printf("8 - Up-Left\n");
    return(8);
break;
}

case 17:{
    //strcpy(position, "Down-Left");
    printf("9 - Down-Left\n");
    return(9);
break;
}

```

```

case 18:
    //strcpy(position, "Down-Right");
    printf("10 - Down-Right\n");
    return(10);
break;

case 12:
    //strcpy(position, "Select-Up");
    printf("11 - Select-Up\n");
    return(11);
break;

case 6:
    //strcpy(position, "Select-Right");
    printf("12 - Select-Right\n");
    return(12);
break;

case 20:
    //strcpy(position, "Select-Down");
    printf("13 - Select-Down\n");
    return(13);
break;

case 5:
    //strcpy(position, "Select-Left");
    printf("14 - Select-Left\n");
    return(14);
break;

case 14:
    //strcpy(position, "Select-Up-Right");
    printf("15 - Select-Up-Right\n");
    return(15);
break;

case 13:
    //strcpy(position, "Select-Up-Left");
    printf("16 - Select-Up-Left\n");
    return(16);
break;

/*
case 21:
    strcpy(position, "Select-Down-Left");

```

```

        break;

        case 22:
            strcpy(position, "Select-Down-Right");
            break;
        /*
        default:{
            printf("Physically Impossible combination, try using non-opposing
positions");
            return(0);
        }
    }
}

int scanfToJ(char scanIn){
    int x = Joystick_GetState();

    return x;
}

```

## IRQ.c

```

/*-----
 * Copyright (c) 2004-2020 Arm Limited (or its affiliates). All
 * rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * 1.Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * 2.Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * 3.Neither the name of Arm nor the names of its contributors may be used
 * to endorse or promote products derived from this software without
 * specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS AND CONTRIBUTORS BE

```

```

* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*-----
* Name:      IRQ.c
* Purpose:   IRQ Handler
*-----*/

#include "projectMain.h"

#include "LPC17xx.h"           // LPC17xx Definitions
#include "Board_LED.h"
#include "Board_ADC.h"

volatile uint8_t clock_1s;      // Flag activated each second

void SysTick_HandlerIRQ (void);

/*-----
SysTick IRQ Handler (occurs every 10 ms)
*-----*/
void SysTick_HandlerIRQ (void) {
    static uint32_t ticks = 0U;
    static uint32_t timetick = 0U;
    static uint32_t leds = 0x01U;

    if (++ticks == 100U) {      // Set Clock1s to 1 every second
        ticks    = 0U;
        clock_1s = 1U;
    }

    // Blink the LEDs depending on ADC_Converted Value
    if (++timetick >= ADC_last) {
        timetick = 0U;
        LED_SetOut(leds);
        leds <<= 1;
        if (leds == (1U << LED_GetCount())) {
            leds = 0x01U;
        }
    }

    ADC_StartConversion();      // Start ADC conversion
}

```

