Assignment 2: Implementation of Cursor Based Paint using Timer Interruptions

BLG212E Fall 2025

1 Assignment Description

In this assignment, students are required to develop an ARM Cortex-M0 assembly program that receives input from an input device using interrupt subroutines. The input represents a brush stroke based on the cursor's location. Because connecting an actual input device to the microprocessor is difficult in the simulation environment, timer interrupts will be used to simulate the input device using a predefined list of inputs.

For this assignment, you will:

- implement a timer interrupt,
- simulate predefined user inputs using the timer interrupt,
- update the memory that represents a single-pixel value on the screen,
- implement the solution within a defined scope, using a limited number of registers, and
- a technical report explaining key details of your implementation.

2 Assignment Details

The program must compile and run successfully in the Keil µVision IDE v5 using the default project configuration. Every line of assembly code must include a clear and meaningful explanatory comment; any lines without comments will not be accepted.

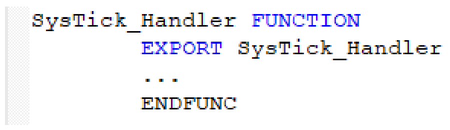
The assignment is divided into parts for your convenience. Breaking a large problem into smaller tasks is an effective approach that makes the work more manageable. Therefore, it is recommended that you complete the assignment by following the parts in the order provided. Additionally, hints are included in each part whenever necessary.

2.1 Part 1: Timer Interrupt

In this part, students are expected to use the system timer SysTick on the ARM Cortex-M0 processor. For this purpose, use the following registers and function:

- STCTRL register
- STRELOAD register
- SysTick_Handler function

Configure STCTRL to operate with the system clock. Calculate and set STRELOAD based on a microcontroller running at a 10 MHz system clock, with the requirement that the interrupt should occur every 1 second. Implement the SysTick_Handler function in assembly; this predefined function is automatically called when the SysTick interrupt occurs. You must export this function. Please use Figure 1 as a reference.



```
SysTick_Handler FUNCTION
        EXPORT SysTick_Handler
        ...
        ENDFUNC
```

Figure 1: SysTick_Handler

Note that interrupts take more time than regular operations. Therefore, it is common practice to keep interrupt service routines short by only capturing the necessary information and then using that information during the main program execution.

In the report, show your calculations and the content of the registers. Show your unit test results for the timer interrupt.

3 Part 2: Using Timer Interrupt to Simulate User Inputs

I/O devices usually use interrupts to communicate with a microcomputer. In this assignment, we assume the user is operating an I/O device—such as a joystick—to provide input to a cursor, moving it based on how long the input is held. The user can also select colors, represented by numerical values, using the I/O device.

The input from the I/O device is represented using 2 bytes (0xKK). The meaning of the input is determined by the first byte (the most significant byte):

- 'A': Change color of the brush. The new color is given in the second byte.
- '0,1,2,3': Direction of the brush stroke; 0:Up, 1:Right, 2:Down, 3:Left. The lenght of the stroke is given in the second byte.

Examples:

- '0xA2': Change color of the brush to color 2 represented with '0x22' value.
- '0x03': Paint upper 3 cells of the cursor location.

Note that the order of operations in the painting process is: first change the position, then paint. Repeat this sequence until the given length is completed.

To simulate user input, we will use a predefined array of values. The system will retrieve the next input from the array every 1 second. '0xFF' represents the end of the input array.

Example data sections are given in *data.s* file. Copy one of the example at the top of your code to use it. An example array is also given in Figure 2.

```
        AREA myData, DATA, READONLY ; Define a readonly data section
arr     DCD 0xA8, 0x14, 0x24, 0x32, 0x02, 0xFF
```

Figure 2: Example Input Array

In this part, you will implement the handler function for the timer interrupt to load the next input every second.

4 Part 3: Updating the Memory with Simulated User Inputs

The simulated user inputs represent brush strokes based on the cursor's location. To support this behavior, several variables—such as the cursor position—must be used. You may use the variables provided in Figure 3. These variables have the following meanings:

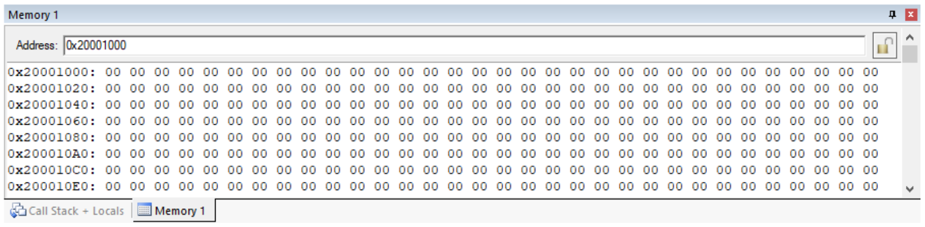
- varx: x position of the cursor(horizontal)
- vary: y position of the cursor(vertical)
- varc: selected color
- vari: index pointer for the input array
- varb: flag for available data from the input array

```
        AREA myDataRW, DATA, READWRITE ; Define a readwrite data section
varx    DCD 0
vary    DCD 0
varc    DCB 0
vari    DCD 0
varb    DCD 0
```

Figure 3: Example Input Array

The variables may be stored as decimal values or bytes. The provided types are chosen arbitrarily, and you may select different data types as needed for your design.

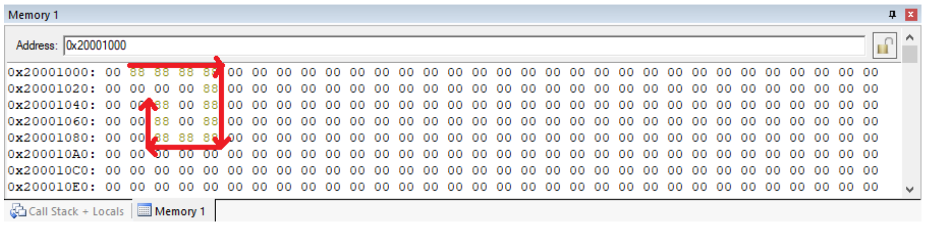
The variables represent the color values on a canvas. The canvas has a size of ('0x20', '0x08') and starts at the memory address '0x20001000'. The (0,0) point is on the top left. The canvas can be visualized in the simulator's memory view if it is sized correctly, as shown in Figure 4.



```
0x20001000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200010A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200010C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200010E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 4: Example Input Array

Using the inputs provided in the example array, a color is selected first, and then a pattern consisting of four strokes is drawn. The resulting pattern is shown in Figure 5.



```
0x20001000: 00 88 88 88 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001020: 00 00 00 00 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001040: 00 00 88 00 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001060: 00 00 88 00 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20001080: 00 00 88 88 88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200010A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200010C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200010E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 5: Example Input Array

Note that if a movement would cause the cursor to go out of bounds, the cursor should remain at its previous valid position, preventing it from moving outside the canvas limits.

In this part, you are expected to implement several functionalities that simulate drawing on the canvas using the simulated user inputs. This section may be challenging; therefore, it is recommended that you make use of functions. Implementing the following functions will be highly beneficial in managing the complexity of the task.

- goUp, goRight ...
- paint, changeColor ...
- SysTickEnable, SysTickDisable, SysTickRetimer ...

Note that using such functions can help you test individual components, thereby improving the overall quality of your work.

5 Part 4: Scope, Using Limited Number of Registers

Using functions in a project like this can create challenges related to register sharing. This issue can be addressed through proper scope management. In microcomputers, the stack is used to store the previous values of registers, allowing those registers to be reused during function execution. After the function completes, the stored values are restored to their corresponding registers.

In this assignment, it is sufficient to use only two registers (in addition to the variables defined above) with proper scope management. In this part, you are required to implement scope management for function calls so that no more than two registers (R0 and R1) are used. Be sure to explain your approach in your report.

6 Part 5: Report

You are expected to write a short report for this assignment explaining your calculations and any key details that are worth noting. Include visualizations of the output and provide documentation for your functions. Be sure to describe your overall approach.

7 Submission

Submit the following items through Ninova by the specified deadline.

Submission:

- assembly code
- report

Submission Deadline: Check Ninova.

Check before you submit:

- Every part of your code must include a clear explanation. In particular, key decision points that lead to branches should be thoroughly documented. Providing explanations for these important sections will help you work through the assignment more smoothly.
- In accordance with academic honesty principles, you are expected to complete this assignment on your own. Any violation of academic integrity, including cheating, will be penalized according to ITU regulations.

For questions or requests for clarification, you may contact the corresponding teaching assistant at akab@itu.edu.tr