



**BLG212E - MICROPROCESSOR SYSTEMS**  
**ASSIGNMENT 2 - Implementation of Cursor Based**  
**Paint using Timer Interruptions**  
**REPORT**

**NAME-SURNAME: İLKE BAŞAK BAYDAR**  
**STUDENT ID: 150140709**

## 1. INTRODUCTION

The aim of this assignment is to develop a SysTick Timer based drawing application with using ARM Cortex-M0+ microprocessor in Keil uVision IDE. This program sequentially reads instructions from given data array and draws on a memory-mapped virtual screen starting at 0x20001000 address using cursor movements.

## 2. DESIGN AND IMPLEMENTATION DETAILS

Initialization of interrupt registers is given below

```
34 ;REG DEFINITIONS
35 STCTRL EQU 0xE000E010 ;CONTROL AND STATUS REG ADDR
36 STRELOAD EQU 0xE000E014 ;RELOAD VALUE REG ADDR
37 STCURRENT EQU 0xE000E018 ;CURRENT VALUE REG ADDR
38 SCREEN_BASE EQU 0x20001000 ;CANVAS START ADDR
39 WIDTH EQU 32 ;CANVAS WIDTH 32 PX
40 HEIGHT EQU 8 ; CANVAS HEIGHT 8 PX
41
42 RELOAD_VAL EQU 0x0098967F ;RELOAD VALUE FOR 1 SECOND AT 10MHz-->(1 SEC * 10000000 HZ) -1 = 99999999 = 0x98967F HEX
43
44
```

And also SysTick\_Handler procedure can be shown:

```
66 SysTick_Handler PROC
67     LDR R0, =vari ;LOAD INDEX TO R0
68     LDR R1, [R0] ; R1 = CURRENT INDEX
69     PUSH {R1} ; PUSH INDEX TO STACK
70     LDR R0, =arr ; R0 =ARR[0]
71     LSLS R1, R1, #2 ; R1 = OFFSET (INDEX*4)
72     ADDS R0, R0, R1 ; GET ELEMENT'S ADDR (OFFSET + BASE)
73     LDR R1, [R0] ; LOAD VALOUE FROM ARR
74     MOVS R0, R1 ; MOVE VAL TO R0
75     POP {R1} ; RESTORE INDEX TO R1
76
77     CMP R0, #0xFF ; COMPARE LOADED VALUE WITH 0xFF WHICH IS TERMINATOR VALUE
78     BEQ EndHandler ;IF EQUAL BRANCH END
79
80
81     PUSH {R0, R1} ;SAVE R0 AND R1 IN STACK
82     LDR R0, =varb ; LOAD FLAG TO R0
83     MOVS R1, #1 ; MOVE 1 IN R1
84     STR R1, [R0] ; SET FLAG TO 1
85     POP {R0, R1} ;RESTORE R0 AND R1 FROM STACK TO REGS
86
87     ADDS R1, R1, #1 ; INDEX = INDEX +1
88     LDR R0, =vari ;LOAD ADDR OF INDX
89     STR R1, [R0] ; STORE CHANGED INDEX
90
91 EndHandler
92     BX LR;RETURN FROM INTERRUPT
93 ENDP
```

Implementation details with given restrictions as follows.

### 2.1.SysTick Timer

According to assignment PDF, reload register STRELOAD should be calculated and set based on a u-controller running at 10 MHz system clock with the need that interrupt happen once every second.

Reload value is calculated with given formula:

$$\text{Reload} = (1 \text{ second} * 10000000 \text{ Hertz}) - 1 = 9999999 \text{ decimal}$$

$$9999999 \text{ decimal} = 0x0098967F \text{ hex}$$

SysTickRetimer helper function is given below.

```

111 SysTickRetimer PROC ;HELPER FUNC TO SET RELOAD AND CLEAR CURRENT
112     PUSH {R0, R1, LR} ;SAVE REGS AND LR
113     LDR R0, =STRELOAD; LOAD SYSTICK RELOAD REG
114     LDR R1, =RELOAD_VAL ;LOAD CALCULATED RELOAD_VAL
115     STR R1, [R0]; STRE RELOAD_VAL INTO RELOAD REG
116
117     LDR R0, =STCURRENT ;LOAD STCURRENT REG
118     MOVS R1, #0 ;MOVE 0 TO R1 FOR CLEAR INSIDE
119     STR R1, [R0] ;WRITE CONFIG TO START TIMER
120
121     POP {R0, R1, PC} ;RESTORE REGS AND RETURN
122     ENDP
123

```

## 2.2. Implementation of Data Processing with *Process* Procedure

Process procedure is main handler fuction called when SysTick Handler flag varb equal to 1. Algorithm of this subroutine like this:

Firstly, index variable vari is read but since SysTick Handler has already incremented the index for next interrupt, it decrements the index by 1 to find current data that needs to be processed.

Then using calculated index, it loads the relevant byte from the array into R1.

To decode 8-bit data read, it splits it into two parts: Upper 4 bits contains the instruction type (SetColor or DoMovement) or direction information and lower 4 bits carries color value or length of movement. If upper 4 bits == 0xA then this is branched SetColor, else this is branched DoMovement.

Implementation of this part as follows.

```

125 Process PROC
126     PUSH {R0, R1, LR} ;STORE REG IN STACK
127     LDR R0, =vari ;LOAD INDEX ADDR
128     LDR R1, [R0] ;LOAD INDEX VALUE
129     SUBS R1, R1, #1 ;INDEX = INDEX -1
130
131     LDR R0, =arr ;LOAD ARR[0]
132     LSL R1, R1, #2 ;OFFSET = INDX*4
133     ADDS R0, R0, R1 ;CALC ELEMENT'S ADDR
134     LDR R1, [R0] ;LOAD INPUT VAL IN R1
135
136     MOVS R0, R1;MOVE INPUT TO R0 FOR REQUESTED CHANGES
137     LSR R0, R0, #4 ;SHIFT RIGHT 4 FOR SEPARATING UPPER BITS
138     CMP R0, #0xA ;COMPARE UPPER BITS WITH COLOR
139     BEQ SetColor ;IF EQUAL, BRANCH SET COLOR
140
141     B DoMovement ;ELSE BRANCH MOVE

```

### a. *SetColor* Logic

If it is branched SetColor, firstly masking operation is implemented with only takes bottom 4 bits of data (actually color part). After that, to comply with assignment requirement, it is copied and pasted next 4 bits and written the new generated color colde to clobal varc variable.

Implementation of SetColor logic is given below.

```
143 SetColor
144     MOVS R0, #0x0F; FOR MASKING MOVE 0x0F TO R0
145     ANDS R1, R1, R0; MASK UPPER BITS TO GET COLOR
146
147     MOVS R0, R1 ; COPY LOWER 4 BIT TO R0
148     LSLS R0, R0, #4 ; SHIFT LEFT R0 BY 4 (0x20)
149     ORRS R1, R1, R0 ; 0x20 OR 0x02 = 0x22
150
151     LDR R0, =varc ; LOAD ADR OF COLOR
152     STRB R1, [R0] ; STORE NEW COLOR BYTE
153     B EndProcess ; BRANCH END PROCESS
```

### b. *DoMovement* Logic

If it is branched DoMovement, firstly the bottom 4 bits are masked to get length counter in R0 and then data shifted 4 bits to the right to get direction in R1. After these instructions, compare lenght with 0, if length == 0, branch exiti else push length into stack and branch PaintLoop.

Implementation of DoMovement logic as follows.

```
155 DoMovement
156     MOVS R0, #0x0F ; FOR MASKING MOVE 0x0F TO R0
157     ANDS R0, R1, R0 ; MASK TO GET LENGTH
158     CMP R0, #0 ; COMPARE LENGHT WITH 0
159     BEQ EndProcess ; IF LEN == 0 BRANCH END PROCESS
160     PUSH {R0} ; ELSE PUSH LENGTH TO STACK
161     LSRS R1, R1, #4 ; SHIFT RIGHT BY 4 FOR DIRECTION
```

### c. *PaintLoop*

Paint loop repeats advance and paint operation required to draw a line until specified length is reached. Logic of this loop can be represented in C like this:

```
while(length > 0){
    MoveCursor(direction);
    PaintPixel();
    length --;
```

}

Implementation of this loop with assembly is given below.

```
163 PaintLoop
164     BL MoveCursor ;BRANCH HORIZONTAL AND VERTICAL POSITIONS
165     BL PaintPixel ;BRANCH PAINT MEMORY
166
167     POP {R0} ;POP LENGTH
168     SUBS R0, R0, #1 ; LENGTH= LENGTH -1
169     CMP R0, #0 ;COMPARE LENGTH AND 0
170     BEQ EndProcess ; If EQUAL BRANCH END PROCESS
171
172     PUSH {R0} ;ELSE PUSH UPDATED LENGTH TO STACK
173     B PaintLoop ;BRANCH PAINT LOOP
```

#### d. *MoveCursor* Implementation

MoveCursor procedure is that updates the cursor's horizontal and vertical coordinates (varx and vary) according to given direction and prevents the cursor from going off screen with checking boundaries.

Implementation of MoveCursor is given below.

```
179 MoveCursor PROC
180     PUSH {R0,R1,LR} ;SAVE REGS
181     CMP R1, #0 ;COMPARE DIRECTION AND 0
182     BEQ GoUp ;IF 0 GO UP
183     CMP R1, #1 ;COMPARE DIRECTION AND 1
184     BEQ GoRight ;IF 1 GO RIGHT
185     CMP R1, #2 ;COMPARE DIRECTION AND 2
186     BEQ GoDown ;IF 2 GO DOWN
187     CMP R1, #3 ;COMPARE DIRECTION AND 3
188     BEQ GoLeft ;IF 3 GO LEFT
189     B ExitMove ;ELSE BRANCH EXIT MOVE
```

Horizontal movements are given below.

```

209 GoLeft
210     LDR R0, =varx ;LOAD HORIZONTAL ADDR
211     LDR R1, [R0] ;LOAD VARX VLAUE
212     SUBS R1, R1, #1 ; VARX = VARX-1
213     CMP R1, #0xFF; COMPARE 0xFF (0-1)
214     BEQ ExitMove ;IF EQUAL EXIT
215     STR R1, [R0] ;ELSE STORE NEW VARX
216     B ExitMove ;BRANCH EXIT
217
218 GoRight
219     LDR R0, =varx ;LOAD HORIZONTAL ADDR
220     LDR R1, [R0] ;LOAD VARX VLAUE
221     ADDS R1, R1, #1 ; VARX = VARX+1
222     CMP R1, #WIDTH ;COMPARE WIDTH LIMIT
223     BEQ ExitMove ;IF REACH LIMIT, BRANCH EXIT
224     STR R1, [R0] ;ELSE STORE NEW VARX
225     B ExitMove ;BRANCH EXIT

```

And also vertical movements is like this.

```

200 GoDown
201     LDR R0, =vary ;LOAD VERTICAL ADDR
202     LDR R1, [R0] ;LOAD VARY VALUE
203     ADDS R1, R1, #1 ;VARY = VARY +1
204     CMP R1, #HEIGHT ;COMPARE UPPER LIMIT
205     BEQ ExitMove ; IF EQUAL EXIT
206     STR R1, [R0] ;ELSE STORE NEW VARY
207     B ExitMove ;BRANCH EXIT
208
209 GoLeft
210     LDR R0, =varx ;LOAD HORIZONTAL ADDR
211     LDR R1, [R0] ;LOAD VARX VLAUE
212     SUBS R1, R1, #1 ; VARX = VARX-1
213     CMP R1, #0xFF; COMPARE 0xFF (0-1)
214     BEQ ExitMove ;IF EQUAL EXIT
215     STR R1, [R0] ;ELSE STORE NEW VARX
216     B ExitMove ;BRANCH EXIT

```

### e. *PaintPixel* Logic

This procedure calculates the exact physical address in memory using current cursor position (varx, vary) and writes the value from global color variable varc to that address with STRB instruction.

Since memory map is linear, to convert coordinates to physical address this formula is implemented:

$$SCREEN\_BASE = 0x20001000$$

$$Target\ Address = SCREEN\_BASE + (y*WIDTH) + x$$

**Row offset (Calculate y):** vary is shifted left to evaluate  $y*32$  which  $32 = 2^5$

**Column offset (Calculate x):** varx value is added them up the row offset.

Implementation of this procedure is given below.

```
231 PaintPixel PROC
232     PUSH {R0, R1, LR} ;STORE REGS IN STACK
233
234     LDR R0, =vary ;LOAD VERTICAL ADDR
235     LDR R0, [R0] ;LOAD VERTICAL VALUE
236     LSLS R0, R0, #5 ; VARY = VARY*32 (OFFSET)
237
238     LDR R1, =varx ;LOAD HORIZONTAL ADDR
239     LDR R1, [R1] ;LOAD HORIZONTAL VALUE
240     ADDS R0, R0, R1 ;R0 = VARY*32 + VARX (OFFSET + VARX)
241
242     LDR R1, =SCREEN_BASE ;LOAD SCREEN_BASE ADDR
243     ADDS R0, R0, R1 ;BASE + OFFSET TO GET PX ADDR
244
245     LDR R1, =varc ;LOAD CURRENT COLOR ADDR
246     LDRB R1, [R1] ;LOAD COLOR VALUE
247     STRB R1, [R0] ;PAINT BYTE OF COLOR
248
249     POP {R0, R1, PC};RESTORE REGS AND RETURN
250     ENDP
```

### 3. RESULTS

#### 3.1.Shorter Example

```
5 Shorter Example
6
7     AREA myData, DATA, READONLY ; Define a read only data section
8 arr   DCD 0xA8, 0x14, 0x24, 0x32, 0x02, 0xFF
9     AREA myDataRW, DATA, READWRITE ; Define a read write data section
10 varx  DCD 0
11 vary  DCD 0
12 varc  DCB 0
13 vari  DCD 0
14 varb  DCD 0
```

Result can be seen in memory like this.

