Center

For Education

Introduction to Embedded systems (Microcontrollers)

Final Revision - I

Center

For Education

مدينه نصر- شارع النزهة - عمارات الشركه السعوديه - امام البوابه الرابعه لدارالدفاع الجوى - بجوار سوبر ماركت مكه مول

للأستعلام : 01004341713 / 01152088956

2024 / 2025

Final Revision

Part 2

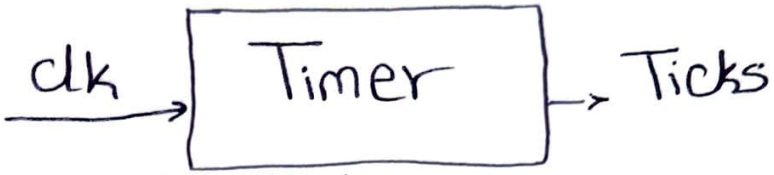
Content after Mid Term (1/2)

Content :-

1) optimization
2) SysTick Timer
3) Timers "GPTM"

2) Systick Timer

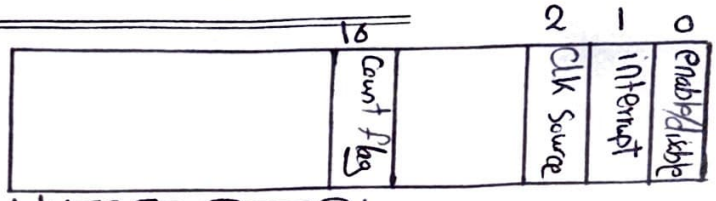
Timer is an independent hardware that takes a clock and make Ticks with every edge



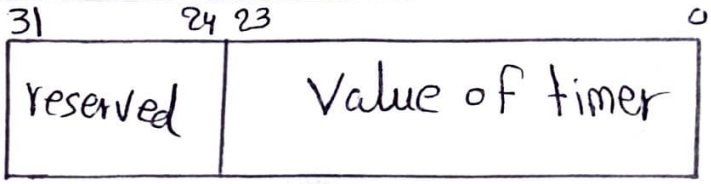
The Systick timer is a 24 bit_timer that means it can count from 0x00FFFFFF to 0x0000 0000 (16,777,216 Ticks)

Systick Timer has 3 registers to control it

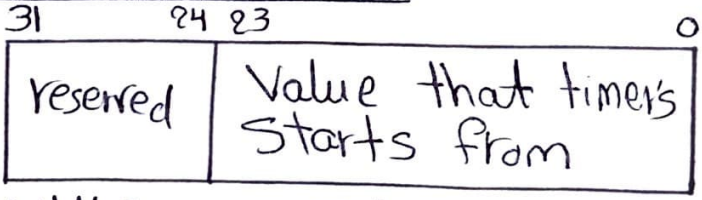
bit0 → 0 disable, 1 enable
bit1 → 0 no interrupt, 1 make interrupt
bit2 → 0 external clk, 1 internal clk



NVIC-SLCTRL



NVIC-ST-Current



NVIC-ST-Reload

bit 16 in CTRL Register "Count flag" is SET to 1 when timer reaches 0, this flag returns to 0 when the CTRL Register is Red. "LDR"

3) GPTM

Tiva-C has 12 Timers rather than Systick Timer in a module called GPTM "General purpose Timer module" the 12 Timers are :-

6 Timers 32-bit (0 → 5)

6 wide_Timers 64-bit (0 → 5)

each Timer can be splitted to 2 timers with half Range.

eg. Timer0 "32-bit" → Timer0A "16-bit"
→ Timer0B "16-bit"

GPTM Registers

① RCGCTIMER → used to unlock the clock to the Timer

Timer 0 → bit0
Timer 1 → bit1
⋮

② GPTMCFG → used to determine if the timer will work Concatinated "32-bit" or individual "16"

if GPTMCFG = 0; → 32-bit timer
else if GPTMCFG = 0x4; → 16-bit timer

③ GPTMCTL → to enable/disable timer
to choose if the timer should stop when the Cpu stops with the debugger

bit 0 → enable / disable
① ⓪

bit 1 → Stop with Cpu / don't stop
① ⓪

④ GPTM TAMR "Timer A mode Register"

bit 0,1 → To determine timer mode

B0 B1

0 1 → one shot

1 0 → periodic

bit 4 → to determine timer Direction

0 → Count Down

1 → Count up

⑤ GPTM RIS "Raw interrupt status"

Contains the flags for different timer modes

Bit 0 "TATORIS (Timer A Time out Raw interrupt status)

if 0 timer DiDn't finish Counting

if 1 timer finished Counting

⑥ GPTM ICR "interrupt clear register"

Contains bits corresponding to the flag bits in RIS

these bits are used to clear the flag bits

Bit 0 "TATOCINT (Timer A Time out clear interrupt)

if you write it to 1 then "TATORIS" is 0

and "TATOCINT" is 0 afterwards.

⑦ GPTM IM "interrupt MASK"

Contains bits corresponding to the flag bits in RIS

When SET to 1 this Causes "Flag to Cause interrupt

Bit 0 "TATOIM" When SET to 1 this makes

"TATORIS" Cause interrupt

⑧ GPTM TAILR "Timer A interval load register"
used to put the number of ticks in

⑨ GPTMTAPR "Timer A prescaler"
used to determine the prescaler
Can take values from 0 → 255
Which means prescaler from 1 → 256

⑩ GPTMTAR "Timer A register"
Contains the value of the timer now

⑪ GPTMTAV "Timer A Value"
Contains the value of the timer now
+ the value of the prescaler

prescaler Value e.g 0x0005	Timer Value e.g 0xfe21

note When timer is splitted we user Timer A Registers for timer A and timer B registers for timer B
But when timer is Concatenated we use only Timer A Registe

prescaler with down timer is Time multiplication
& with up timer is Time extension

$$\text{no of ticks} = \frac{\text{Frequency (HZ)} * \text{Time (S)}}{\text{prescaler "1→256"}}$$

Reload Value = no of ticks - 1 "TAILR"

prescaler Value = prescaler - 1 "TAPR"

DATA SHEET

main.c | tm4c123gh6pm.h ×

```
#define TIMER0_CFG_R            (*((volatile unsigned long *)0x40030000))
#define TIMER0_TAMR_R           (*((volatile unsigned long *)0x40030004))
#define TIMER0_TBMR_R           (*((volatile unsigned long *)0x40030008))
#define TIMER0_CTL_R            (*((volatile unsigned long *)0x4003000C))
#define TIMER0_SYNC_R           (*((volatile unsigned long *)0x40030010))
#define TIMER0_IMR_R            (*((volatile unsigned long *)0x40030018))
#define TIMER0_RIS_R            (*((volatile unsigned long *)0x4003001C))
#define TIMER0_MIS_R            (*((volatile unsigned long *)0x40030020))
#define TIMER0_ICR_R            (*((volatile unsigned long *)0x40030024))
#define TIMER0_TAILR_R          (*((volatile unsigned long *)0x40030028))
#define TIMER0_TBILR_R          (*((volatile unsigned long *)0x4003002C))
#define TIMER0_TAMATCHR_R       (*((volatile unsigned long *)0x40030030))
#define TIMER0_TBMATCHR_R       (*((volatile unsigned long *)0x40030034))
#define TIMER0_TAPR_R           (*((volatile unsigned long *)0x40030038))
#define TIMER0_TBPR_R           (*((volatile unsigned long *)0x4003003C))
#define TIMER0_TAPMR_R          (*((volatile unsigned long *)0x40030040))
#define TIMER0_TBPMR_R          (*((volatile unsigned long *)0x40030044))
#define TIMER0_TAR_R            (*((volatile unsigned long *)0x40030048))
#define TIMER0_TBR_R            (*((volatile unsigned long *)0x4003004C))
#define TIMER0_TAV_R            (*((volatile unsigned long *)0x40030050))
#define TIMER0_TBV_R            (*((volatile unsigned long *)0x40030054))
#define TIMER0_RTCPD_R          (*((volatile unsigned long *)0x40030058))
#define TIMER0_TAPS_R           (*((volatile unsigned long *)0x4003005C))
#define TIMER0_TBPS_R           (*((volatile unsigned long *)0x40030060))
```

```
//*****
#define NVIC_SYS_PRI3_TICK_M    0xE0000000  // SysTick Exception Priority
#define NVIC_SYS_PRI3_PENDSV_M  0x00E00000  // PendSV Priority
#define NVIC_SYS_PRI3_DEBUG_M   0x000000FF  // Debug Priority
#define NVIC_SYS_PRI3_TICK_S    29
#define NVIC_SYS_PRI3_PENDSV_S  21
#define NVIC_SYS_PRI3_DEBUG_S   5

//*****

#define NVIC_SYS_CTRL_R         (*((volatile unsigned long *)0xE000ED10))
#define NVIC_CFG_CTRL_R         (*((volatile unsigned long *)0xE000ED14))
#define NVIC_SYS_PRI1_R         (*((volatile unsigned long *)0xE000ED18))
#define NVIC_SYS_PRI2_R         (*((volatile unsigned long *)0xE000ED1C))
#define NVIC_SYS_PRI3_R         (*((volatile unsigned long *)0xE000ED20))

#define GPIO_PORTA_AHB_DATA_BITS_R  ((volatile unsigned long *)0x40058000)
```

Question 6: (9 Marks)

During in circuit debugging session, the following C and disassembly subset windows are shown. Starting from instruction at 0x84, Step-Over command is forwarded to instruction at 0x86 and then 0x88. Find the values stored at Timer0_TAR_R at each step by filling the following table. The number of clock cycles needed to execute relevant assembly instructions are given in the table.

main.c × tm4c123gh6pm.h

```
#include "tm4c123gh6pm.h"
void timer0A_delayMs(int ttime);

int main() {
    SYSCTL_RCGCGPIO_R = 0x20U;
    GPIO_PORTF_DIR_R = 0x0EU;
    GPIO_PORTF_DEN_R = 0x0EU;
    SYSCTL_RCGCTIMER_R |= 0x01;
    TIMER0_CTL_R = 0x0;
    TIMER0_CFG_R = 0x04;
    TIMER0_TAMR_R = 0x02;
    TIMER0_TAPR_R = 5;
    while (1) {
        GPIO_PORTF_DATA_R = 0x08U;
        timer0A_delayMs(100);
        GPIO_PORTF_DATA_R = 0x00U;
        timer0A_delayMs(100);
    }
}
void
timer0A_delayMs(int ttime)
{
    TIMER0_CTL_R = 0x0;
    TIMER0_ICR_R = 0x1;
    TIMER0_TAILR_R =0xFFF;
    TIMER0_CTL_R |= 0x03;
    [redacted];
}
```

Disassembly

Go to | Memory

Disassembly

```
TIMER0_CTL_R = 0x0;
    0x72: 0x2400      MOVS    R4, #0
    0x74: 0x60cc      STR     R4, [R1, #0xc]
TIMER0_ICR_R = 0x1
    0x76: 0x2501      MOVS    R5, #1
    0x78: 0x624d      STR     R5, [R1, #0x24]
TIMER0_TAILR_R =0xFF;
    0x7a: 0x628a      STR     R2, [R1, #0x28]
TIMER0_CTL_R |= 0x03;
    0x7c: 0x68cb      LDR     R3, [R1, #0xc]
    0x7e: 0xf043 0x0303 ORR.W R3, R3, #3
    0x82: 0x60cb      STR     R3, [R1, #0xc]
while ((TIMER0_RIS_R & 0x1) == 0) ;
    [redacted]        LDR     R3, [R1, #0x1c]
    0x86: 0x07dc      LSLS    R4, R3, #31
    0x88: 0xd5fc      BPL.N   0x84
}
    0x8a: 0x2300      MOVS    R3, #0
    0x8c: 0x6003      STR     R3, [R0]
timer0A_delayMs(100);
    0x8e: 0x60cb      STR     R3, [R1, #0xc]
    0x90: 0x2401      MOVS    R4, #1
    0x92: 0x624c      STR     R4, [R1, #0x24]
    0x94: 0x628a      STR     R2, [R1, #0x28]
    0x96: 0x68cb      LDR     R3, [R1, #0xc]
```

Assembly Instruction at ...	Number of Clock Cycles	Timer0_TAR_R ?
0x84	6	0x FFe
0x86	4	0x FFe
0x88	4	0x FFd

TAPR = 0x5;

TAILR = 0x FFF;

TAV

0005 | 0FFF

0x84

0005 0FFe

0x86

0001 0FFe

0x88

0003 0FFd

TAR

0000 0F F F

0000 0FFe

0000 0FFe

0000 0FFd

Summer 21

Question 4: (8 Marks)

During in circuit debugging session, the following C and disassembly subset windows are shown. Starting from instruction 0x84, Step-Over command is forwarded to instruction at 0x86 and then 0x88. **Find the values stored at Timer0_TAR_R after each step by filling the following table.** The number of clock cycles needed to execute relevant assembly instruction are given in the table.

```
#include "tm4c123gh6pm.h"
void timer0A_delayMs(int ttime);

int main() {
    SYSCTL_RCGCGPIO_R = 0x20U;
    GPIO_PORTF_DIR_R = 0x0EU;
    GPIO_PORTF_DEN_R = 0x0EU;
    SYSCTL_RCGCTIMER_R |= 0x01;
    TIMER0_CTL_R = 0x0;
    TIMER0_CFG_R = 0x04;
    TIMER0_TAMR_R = 0x02;
    TIMER0_TAPR_R = 3;
    while (1) {
        GPIO_PORTF_DATA_R = 0x08U;
        timer0A_delayMs(100);
        GPIO_PORTF_DATA_R = 0x00U;
        timer0A_delayMs(100);
    }
}
void
timer0A_delayMs(int ttime)
{
    TIMER0_CTL_R = 0x0;
    TIMER0_ICR_R = 0x1;
    TIMER0_TAILR_R =0xFFF;
    TIMER0_CTL_R |= 0x03;
    [redacted] ;
}
```

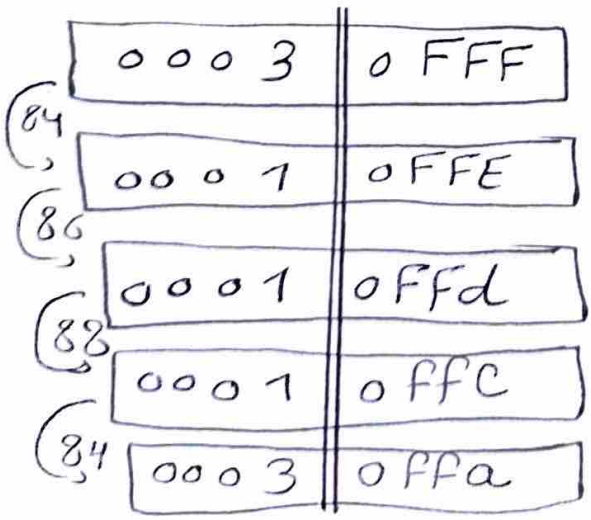
```
Disassembly
Goto                Memory
Disassembly
    TIMER0_CTL_R = 0x0;
        0x72: 0x2400      MOVS    R4, #0
        0x74: 0x60cc      STR     R4, [R1, #0xc]
    TIMER0_ICR_R = 0x1;
        0x76: 0x2501      MOVS    R5, #1
        0x78: 0x624d      STR     R5, [R1, #0x24]
    TIMER0_TAILR_R =0xFFF;
        0x7a: 0x628a      STR     R2, [R1, #0x28]
    TIMER0_CTL_R |= 0x03;
        0x7c: 0x68cb      LDR     R3, [R1, #0xc]
        0x7e: 0xf043 0x0303 ORR.W R3, R3, #3
        0x82: 0x60cb      STR     R3, [R1, #0xc]
    while ((TIMER0_RIS_R & 0x1) == 0) ;
        0x84: 0x69cb      LDR     R3, [R1, #0x1c]
        0x86: 0x07dc      LSLS    R4, R3, #31
        0x88: 0xd5fc      BPL.N   0x84
    }
        0x8a: 0x2300      MOVS    R3, #0
        0x8c: 0x6003      STR     R3, [R0]
    timer0A_delayMs(100);
        0x8e: 0x60cb      STR     R3, [R1, #0xc]
        0x90: 0x2401      MOVS    R4, #1
        0x92: 0x624c      STR     R4, [R1, #0x24]
        0x94: 0x628a      STR     R2, [R1, #0x28]
        0x96: 0x68cb      LDR     R3, [R1, #0xc]
```

Assembly Instruction at ...	Number of Clock Cycles	Timer0_TAR_R ?
0x84	6	0x FFe
0x86	4	0x ffd
0x88	4	0x ffc
0x84	6	0x ffa

TAPR= 0x 3;

TAILR=0x FFF

TAV



Question 5: (6 Marks)

During in circuit debugging session, the following C code snap shot is shown. Fill in (your best expectations) the memory locations when BP1 is hit and BP2 is hit during debugging.

main.c × tm4c123gh6pm.h

```
#include "tm4c123gh6pm.h"
void Timer0_Init(void);
void timer0A_delayMs();
int main() {
    SYSCTL_RCGCTIMER_R |= 0x01;
    SYSCTL_RCGCGPIO_R = 0x20U;
    GPIO_PORTF_DIR_R = 0x0EU;
    GPIO_PORTF_DEN_R = 0x0EU;
    Timer0_Init();
    while (1) {
        GPIO_PORTF_DATA_R = 0x08U;
        timer0A_delayMs();
        GPIO_PORTF_DATA_R = 0x00U;
        timer0A_delayMs();
    }
}
void Timer0_Init(void){
    TIMER0_CTL_R = 0x0;
    TIMER0_CFG_R = 0x4;
    TIMER0_TAMR_R = 0x12;
}
void timer0A_delayMs()
{
    TIMER0_CTL_R = 0x0;
    TIMER0_TAILR_R =0xFFFF;
BP1 TIMER0_TAPR_R = 0x01;
    TIMER0_ICR_R = 0x1;
BP2 TIMER0_CTL_R |= 0x03;
    while ((TIMER0_RIS_R & 0x1) == 0) ;
}
```

Memory When BP1 is hit

Memory 1

Go to 0x40030010 Memory

0x40030000		
0x40030008	00000000	
0x40030010	00000000	00000000
0x40030018	00000000	
0x40030020	00000000	
0x40030028		
0x40030030	0000FFFF	0000FFFF
0x40030038		00000000
0x40030040	00000000	00000000
0x40030048		0000FFFF
0x40030050		0000FFFF

Memory When BP2 is hit

Memory 1

Go to 0x40030010 Memory

0x40030000		
0x40030008	00000000	
0x40030010	00000000	00000000
0x40030018	00000000	
0x40030020	00000000	
0x40030028		
0x40030030	0000FFFF	0000FFFF
0x40030038		00000000
0x40030040	00000000	00000000
0x40030048		0000FFFF
0x40030050		0000FFFF

Question 6: (6 Marks)

During same debugging session of Q5, the following disassembly window snap shot is shown. Opposite to each instruction, is the number of clock cycles needed for execution. Starting from the green line, assembly is stepped over one after one

Line No
1
2
3
4

```
while ((TIMER0_RIS_R & 0x1) ==
                                  LDR     R0, [R0]
0xae: 0x6800
0xb0: 0x07c0                      LSLS    R0, R0, #31
0xb2: 0xd5fb                      BPL.N   0xae
}
```

Clock Cycles
5
6
4
4

What will be the content of R0 after execution of the first assemble line?

Assume that the relevant TAV register reached 0x0001FFF0 when the program is stopped at Line 1. Fill in (your best expectations) the memory locations when program is moved from Line 1 and back again to starting point.

Program moved from Line 1 to Line 2

Memory 1 — Go to: 0x4003001c — Memory

0x40030000		
0x40030008	00000000	
0x40030010	00000000	00000000
0x40030018	00000000	
0x40030020	00000000	
0x40030028		
0x40030030	0000FFFF	0000FFFF
0x40030038		00000000
0x40030040	00000000	00000000
0x40030048		0000FFFF
0x40030050		0000FFFF

Program moved from Line 2 to Line 3

Memory 1 — Go to: 0x4003001c — Memory

0x40030000		
0x40030008	00000000	
0x40030010	00000000	00000000
0x40030018	00000000	
0x40030020	00000000	
0x40030028		
0x40030030	0000FFFF	0000FFFF
0x40030038		00000000
0x40030040	00000000	00000000
0x40030048		0000FFFF
0x40030050		0000FFFF

Program moved from Line 3 to Line 4

Memory 1 — Go to: 0x4003001c — Memory

0x40030000		
0x40030008	00000000	
0x40030010	00000000	00000000
0x40030018	00000000	
0x40030020	00000000	
0x40030028		
0x40030030	0000FFFF	0000FFFF
0x40030038		00000000
0x40030040	00000000	00000000
0x40030048		0000FFFF
0x40030050		0000FFFF

Program moved from Line 4 to Line 1

Memory 1 — Go to: 0x4003001c — Memory

0x40030000		
0x40030008	00000000	
0x40030010	00000000	00000000
0x40030018	00000000	
0x40030020	00000000	
0x40030028		
0x40030030	0000FFFF	0000FFFF
0x40030038		00000000
0x40030040	00000000	00000000
0x40030048		0000FFFF
0x40030050		0000FFFF