EXPERIMENT 5 - TIMER PROGRAMMING USING MSP432R

LAB TITLE AND CODE : EMBEDDED COMPUTING LAB (19CCE283)
EXPERIMENT NUMBER : 5
DATE : 17/05/2022, TUESDAY

* **AIM :**

Control an LED by developing a timer program using system Tick, Timer 32 and Timer A timers that can be interfaced with an MSP43x microcontroller.

a) **LED CONTROL USING SYSTICK TIMER :**

1. Configure functionality of P2.1 as simple GPIO pins.
2. Configure direction of P2.1 as output for GREEN-LED.
3. Reload register value for generating 1 Hz or 1 sec delay ; MCLK = 30,00,000 Hz
4. Clear STCurrent Value Register.
5. Enable SysTick to begin counting down and disable interrupt generation. For the clock source, system clock (MCLK) is only implemented.
6. In an infinite while-loop, perform the following operations-
   (i) check whether the SysTick has counted down to zero and if COUNTFLAG is set.
   (ii) If yes, trigger the GREEN-LED.

→ C PROGRAM CODE -

```
# include "msp.h"

// Main Function:
int main (void)
{
  P2 -> SEL0 &= ~2; // configure functionality of P2.1 as simple GPIO pins
  P2 -> SEL1 &= ~2;
  P2 -> DIR |= 2; // configure direction of P2.1 as output for GREEN-LED
```

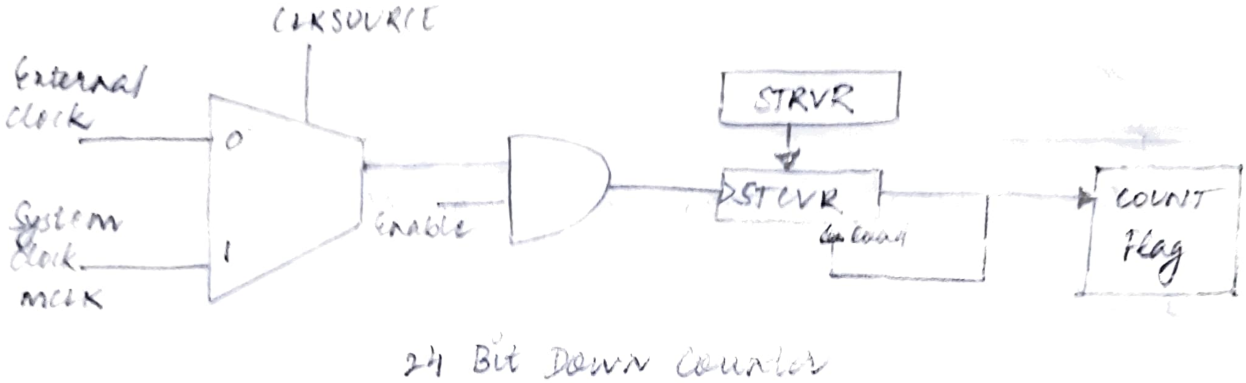


Figure 1 - System Tick Timer Internal structure

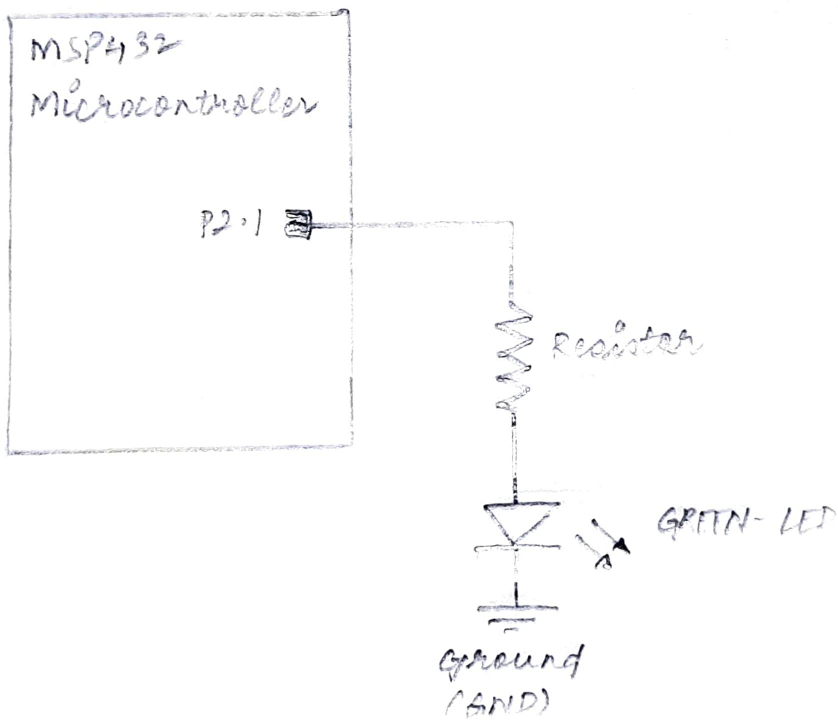


Figure 2 - LED Control using System Tick Timer

```
SysTick → LOAD = 3000000-1; // Reload register value for
generating 1 Hz or 1 sec delay; MCLK = 30,00,000 Hz
SysTick → CTRL = 5; // Enables SysTick to begin counting down;
Interrupt generation is disabled; System clock MCLK.
SysTick → VAL = 0; // Clear STCurrent Value Register

// Infinite Loop (An embedded program does not stop):
while (1)
{
  // The SysTick has counted down to zero:-
  if (SysTick → CTRL & 0x10000) // If COUNTFLAG is set
    P2 → OUT ^= 2; // Trigger the GREEN-LED
}
}
```

5) <u>LED CONTROL USING TIMER 32 TIMER:</u>

1. Configure functionality of P2.1 as simple GPIO pins.
2. Configure direction of P2.1 as output for GREEN-LED.
3. Reload register value for generating 1 Hz or 1 sec delay; Assume prescale unit to be equal to 1; Set MCLK = 30,00,000 Hz
4. Enable timer to begin counting down and mode bit in periodic mode. The counter generates an interrupt at a constant interval, reloading the value from T32LOADn register.
5. Disable timer interrupt enable bit and set prescale bits to 00. clock is divided by 1. Select 32-bit counter operation.
6. Select wrapping mode, i.e., the timer continues counting when it reaches to zero.
7. In an infinite while-loop, perform the following operations-
   (i) Wait till timer is completed.
   (ii) Clear the raw interrupt.
   (iii) Trigger the GREEN-LED.

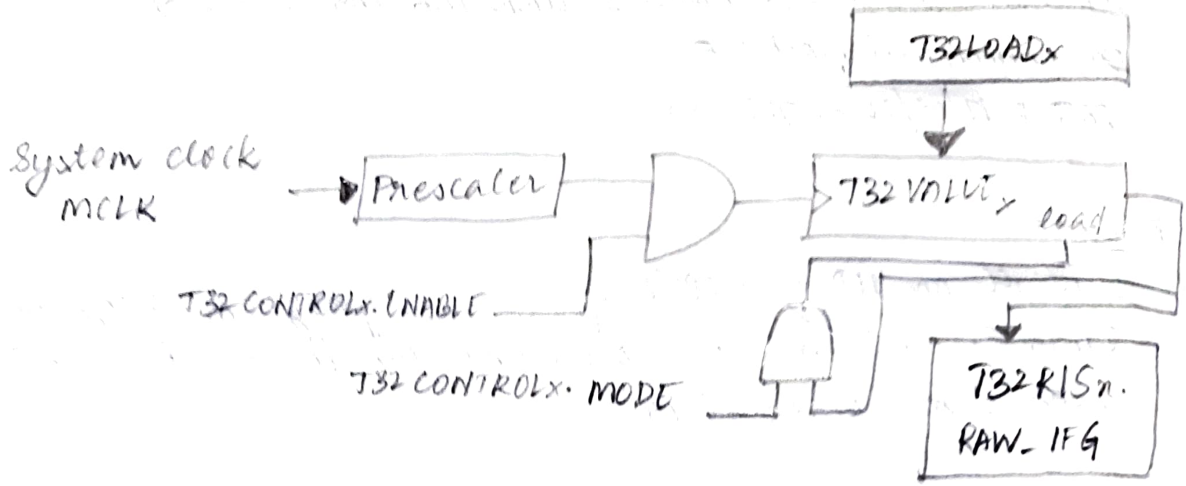


Figure 3- Timer32 Timer Internal structure

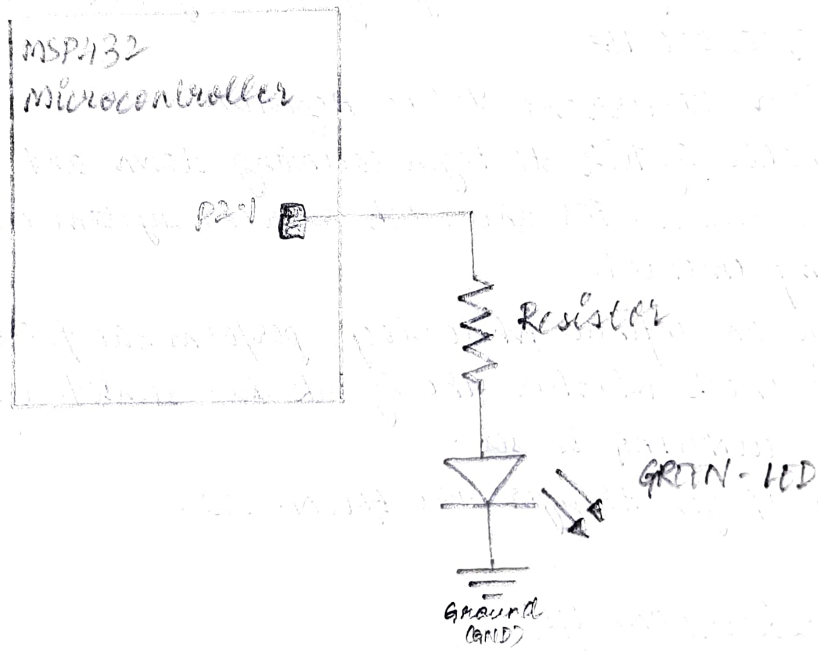


Figure 4- LED Control using Timer32 Timer

→ C PROGRAM CODE -

```
#include "msp.h"

// Main Function:
int main(void)
{
    P2->SEL0 &= ~2; // Configure functionality of P2.1 as simple
GPIO pins
    P2->SEL1 &= ~2;
    P2->DIR |= 2; // configure direction of P2.1 as output for
GREEN-LED

    TIMER32_1->LOAD = 3000000-1; // Reload register value for
generating 1 Hz or 1 sec delay; MCLK = 30,00,000; Prescale unit = 1

    /* Enables timer to begin counting down; Periodic mode;
       Disable timer interrupt enable; set prescale bits to 00;
       clock is divided by 1; select 32-bit counter operation;
       wrapping mode (Timer 1 Timer control Register) */
    TIMER32_1->CONTROL = 0xC2;

    // Infinite Loop (An embedded program does not stop):
    while (1)
    {
        while ((TIMER32_1->RIS & 1) == 0); // wait until timer is
completed (Timer 1 Raw Interrupt status Register)
        TIMER32_1->INTCLR = 0; // Any write to the T32INTCLR1
register clears the interrupt output from the counter.
        P2->OUT ^= 2; // Trigger the GREEN-LED
    }
}
```

c) LED CONTROL USING TIMER A :

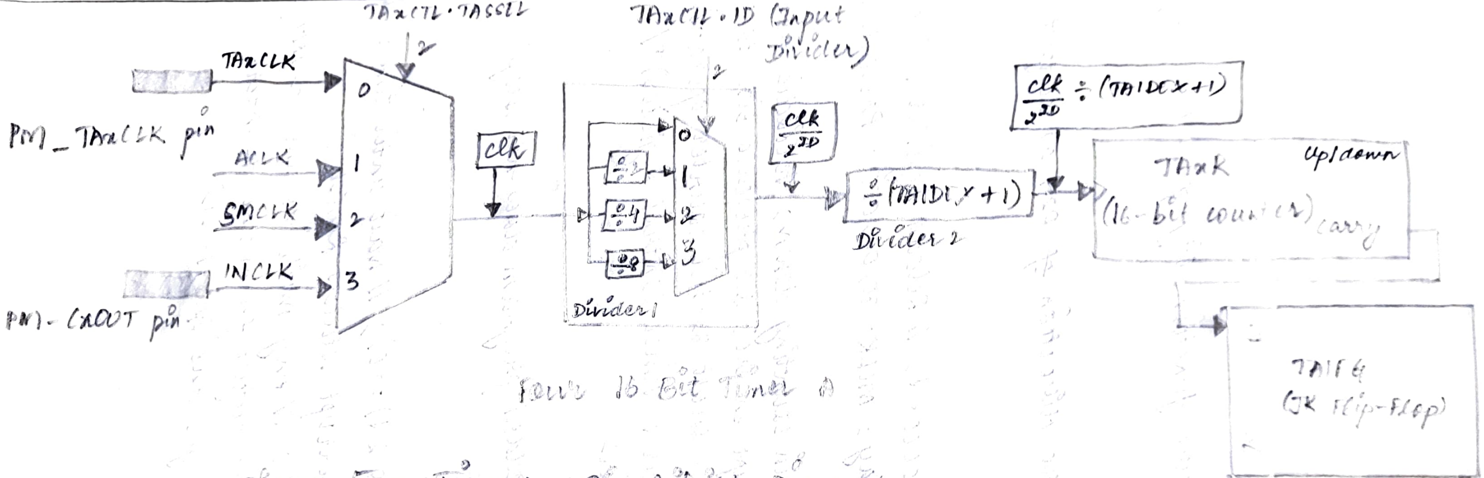
① Configure functionality of P2.1 as simple GPIO pins.
② Configure direction of P2.1 as output for GREEN-LED.
③ Set TimerA interrupt flag to 1 (timer overflowed). TAIFG will be cleared on writing 1 to this bit. Disable interrupt.
④ Set mode control bits to 01, up mode. The timer counts up to TAxCCR0. Set input divider bits to 11, divide by 8. These bits select the divider for the input clock.
⑤ Set clock source select bits to 10, SMCLK (internal clock).
⑥ For generating 1 Hz or 1 sec delay, set CCR[0] = 46875 - 1. Assume $2^{ID} = 8$ and TAIDEX = 7. Set system clock (SMCLK) equal to 30,00,000 Hz
⑦ In an infinite while-loop, perform the following operations-
(i) Wait until the CCIFG is set.
(ii) Clear interrupt flag.
(iii) Trigger the GREEN-LED.

→ C PROGRAM CODE -

```
#include "msp.h"

// Main Function:
int main (void)
{
  P2->SEL0 &= ~2; // Configure functionality of P2.1 as simple GPIO pins.
  P2->SEL1 &= ~2;
  P2->DIR |= 2; // Configure direction of P2.1 as output for GREEN-LED

  /*
    Timer overflowed; Interrupt disabled; Not clear; up mode;
    Interrupt divider; Select clock source as SMCLK
  */
  TIMER_A1->CTL = 0x02D1;
```



Four 16 Bit Timer A

Figure 5 - TimerA Simplified Diagram

```
    TIMER_A1 -> EXO = 7; // Divider 2 = TAIDEX + 1 = 7+1 = 8
    TIMER_A1 -> CCR [0] = 47685-1 ; // For generating 1 Hz or 1 sec
delay ; Assume 2^ID = 8, TAIDEX = 7; SMCLK = 30,00,000

    // Infinite Loop (An embedded program does not stop):
    while (1)
    {
      while ((TIMER_A1 -> CCTL [0] & 1) == 0); // Wait until the
CCIFG flag is set
      TIMER_A1 -> CCTL [0] &= ~1; // Clear interrupt flag
      P2 -> OUT ^= 2; // Trigger the GREEN-LED
    }
}
```

d) <u>LED CONTROL USING TIMER PROGRAMMING</u>:

(Turn LED ON for 30 ms and LED OFF for 40 ms)

1. Initialize a variable "temp" for iteration, to turn LED ON and OFF consecutively.
2. Configure functionality of P2.1 as simple GPIO pins.
3. Configure direction of P2.1 as output for GREEN-LED.
4. Reload register value for generating 1 Hz or 1 sec. delay : MCLK = 30,00,000 Hz
5. Clear STCurrent Value Register.
6. Enable SysTick to begin counting down and disable interrupt generation. For the clock source, system clock (MCLK) is only implemented.
7. Turn ON P2.1 GREEN-LED.
8. In an infinite while-loop, perform the following operations -
   (i) Check whether the SysTick has counted down to zero and if COUNTFLAG is set.
   (ii) If yes, check whether (temp % 2 == 0), i.e. even number -
      - If yes, turn OFF P2.1 GREEN-LED.
      - else, turn ON P2.1 GREEN-LED
   (iii) Increment "temp" by +1.

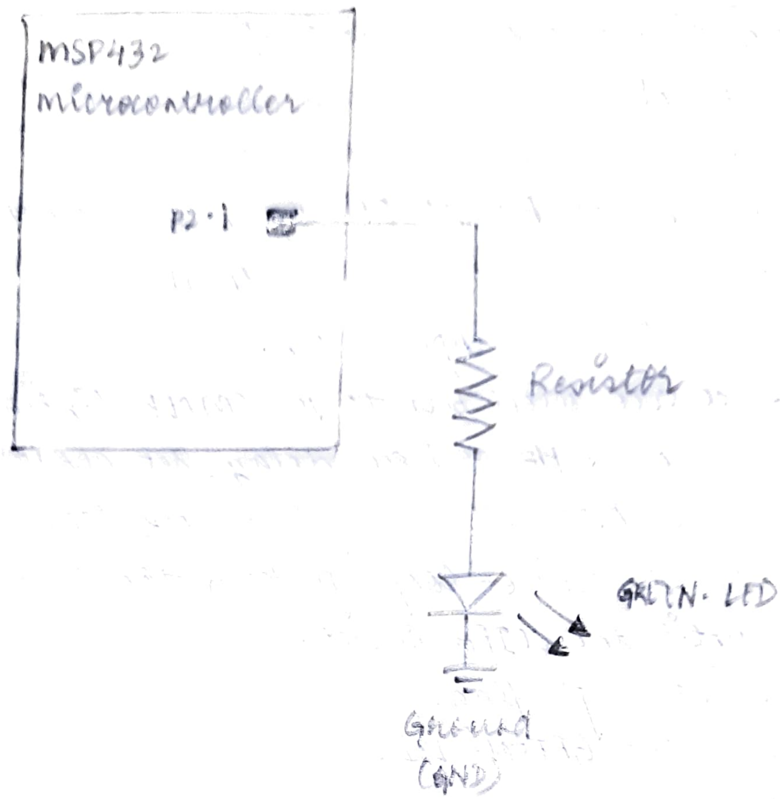


Figure 6- LED Control using Timer A

→ C PROGRAM CODE -

```
#include "msp.h"
int temp = 0; // variable for iteration to turn LED ON and OFF
consecutively.

// Main Function:
int main (void)
{
  P2 → SEL0 &= ~2; // Configure functionality of P2.1 as simple
GPIO pins
  P2 → SEL1 &= ~2;
  P2 → DIR |= 2; // Configure direction of P2.1 as output for
GREEN-LED

   SysTick → LOAD = 90000-1; // Reload register value for
generating 30 milliseconds delay: MCLK = 30,00,000 HZ
   SysTick → VAL = 0; // Clear ST Current Value Register
   SysTick → CTRL = 5; // Enables SysTick to begin counting down;
Interrupt generation is disabled; System clock MCLK
   P2 → OUT |= 2; // Turn ON P2.1 GREEN-LED

   // Infinite Loop (An embedded program does not stop):
   while (1)
   {
    // The SysTick has counted down to zero:-
    if (SysTick → CTRL & 0x10000) // If COUNTFLAG is set

     // Generate 40 milliseconds delay:-
     if (temp % 2 == 0)
     {
      SysTick → LOAD = 120000-1;
      SysTick → VAL = 0;
      SysTick → CTRL = 5;
      P2 → OUT &= ~2; // Turn OFF P2.1 GREEN-LED
```

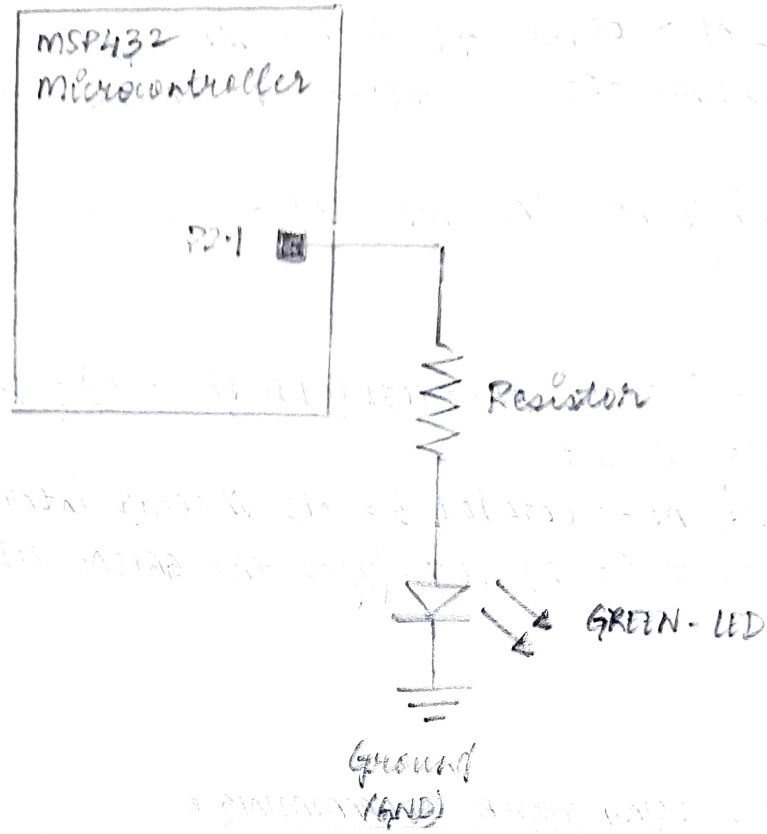


Figure 7 - LED Control using Timer Programming
(Turn LED ON for 80 ms and LED OFF for 40 ms)

```
// Generate 30 milliseconds delay :-
else
{
  SysTick -> LOAD = 90000 -1;
  SysTick -> VAL = 0;
  SysTick -> CTRL = 5;
  P2 -> OUT |= 2; // TURN ON P2.1 GREEN LED
}

 temp ++;
 }
}
```

* **RESULT:**

Thus, an LED was controlled by developing a timer program using system Tick, Timer32 and TimerA timers that can be interfaced with an MSP43x microcontroller. All simulation results were verified successfully.