EMBEDDED SYSTEMS DESIGN

EXPERIMENT 3 : INTRODUCTION TO SysTick Timer

NAME : Dixshant Kumar Jha EE23MT025
Kaushik Shivanand Powar EE23MT011
FACULTY : Dr. ABHIJIT KSHIRSAGAR

AIM :

Use the system timer (Systick) in Polled Mode to generate a waveform with f = 1kHz and d = 20%

MATERIALS REQUIRED:

1. EK-TM4C123GH6PM Board
2. Code Compiler Studio
3. Oscilloscope
4. Datasheet

PROCEDURE

1. Configure the GPIO pins inputs and outputs accordingly.
2. Write a Delay() function, produces a delay in microseconds, using the SysTick Timer on TM4C123GH6PM.
3. Toggle the LEDs with a duty cycle of 20% at the frequency of 1kHz.
4. Verify the frequency and duty cycle using oscilloscope.

SYSTICK TIMER

Systick is a 24 bit timer, decrementing timer. We control the Systick timer using 3 registers:

1. Systick Control and Status Register (**STCTRL**): To configure the Clock Frequency to the timer, enable the timer and enable the Systick Interrupt.
2. Systick Reload Value (**STRELOAD**): The reload value for the counter, the timer counts down from this value, is uploaded in this register.
3. Systick Current Value (**STCURRENT**): This shows the current value of the counter, which decrements every clock cycle starting from the value uploaded in the STRELOAD.

CALCULATION

1. The value of the STCURRENT register decrements every clock cycle, the clock frequency to Systick is 16MHz (as configured in STCTRL), therefore after every 16 clock cycles(counts) we get a delay of 1 microsecond.
2. To produce a delay of ‘x’ microseconds, we load the value ‘ x*16 ‘ in the STRELOAD register.
3. While configuring the Systick, at the beginning, load value ’0’ in the STCURRENT register.

Code

```
#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
/* SysTick memory-mapped registers */
#define STCTRL *((volatile long *) 0xE000E010)    // control and
                                                                status
#define STRELOAD *((volatile long *) 0xE000E014)    // reload value
#define STCURRENT *((volatile long *) 0xE000E018)    // current value

#define COUNT_FLAG  (1 << 16)  // bit 16 of CSR automatically set to 1
                               //   when timer expires
#define ENABLE      (1 << 0)    // bit 0 of CSR to enable the timer
#define CLKINT      (1 << 2)    // bit 2 of CSR to specify CPU clock




void Delay(int us)
{
    STCURRENT = 0;
    STRELOAD = us*16;                       // reload value for 'us'
microseconds
    STCTRL |= (CLKINT | ENABLE);        // set internal clock, enable
the timer

    while ((STCTRL & COUNT_FLAG) == 0)  // wait until flag is set
    {
        ;                               // do nothing
    }
    STCTRL = 0;                 // stop the timer

    return;
}
```

```
int main(void)
{
        SYSCTL_RCGC2_R |= 0x00000020;       /* enable clock to GPIOF
*/
        GPIO_PORTF_LOCK_R = 0x4C4F434B;     /* unlock commit register
*/
        GPIO_PORTF_CR_R = 0x1F;             /* make PORTF0
configurable */
        GPIO_PORTF_DEN_R = 0x1E;            /* set PORTF pins 4 pin */
        GPIO_PORTF_DIR_R = 0x0E;            /* set PORTF4 pin as input
user switch pin */
        GPIO_PORTF_PUR_R = 0x10;            /* PORTF4 is pulled up */

        while(1)
        {
            GPIO_PORTF_DATA_R = 0x0E;
            Delay(200);
            GPIO_PORTF_DATA_R = 0x00;
            Delay(800);
        }
}
```

Figure 1: Oscilloscope waveform using above code

For the freq = 991.37 Hz we get a time period of 1.0087 ms, it is 9 microsecond higher than expected and the duty cycle is higher than 20%, so I subtracted a value of '9' from the value, in the line from code

```
Delay(200);
```

which will reduce 9 microseconds from the on time thereby giving a frequency of near to 1kHz and reduce the duty cycle, which is higher than 20%.

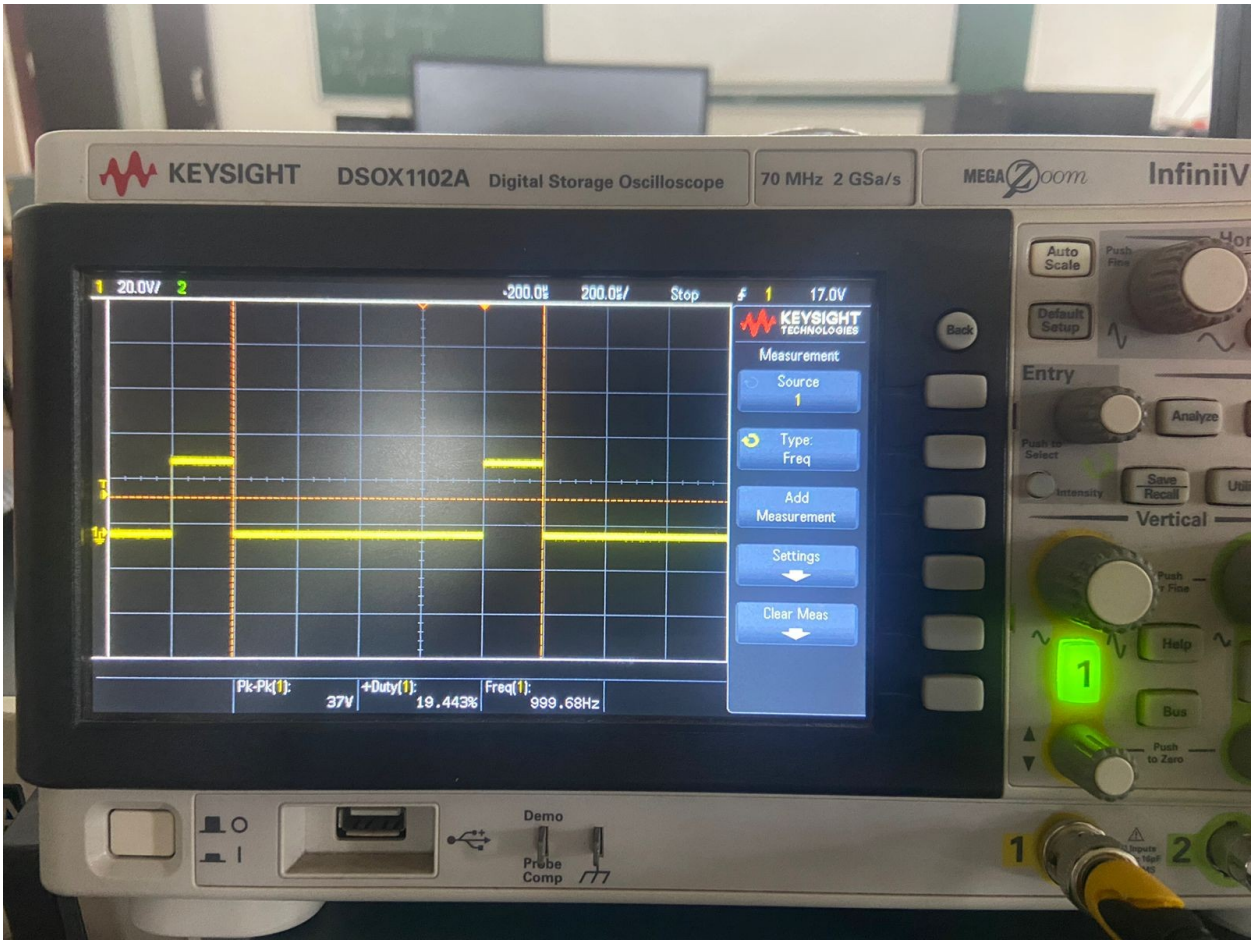

Figure 2: Oscilloscope Waveform after modifying the code

Modified Code:

```
#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
/* SysTick memory-mapped registers */
#define STCTRL *((volatile long *) 0xE000E010)    // control and status
#define STRELOAD *((volatile long *) 0xE000E014)    // reload value
#define STCURRENT *((volatile long *) 0xE000E018)    // current value

#define COUNT_FLAG  (1 << 16)   // bit 16 of CSR automatically set to 1
                        //   when timer expires
#define ENABLE      (1 << 0)    // bit 0 of CSR to enable the timer
#define CLKINT      (1 << 2)    // bit 2 of CSR to specify CPU clock

#define CLOCK_MHZ 16

void Delay(int us)
{
   STCURRENT = 0;
   STRELOAD = us*16;                    // reload value for 'us' microseconds
   STCTRL |= (CLKINT | ENABLE);       // set internal clock, enable the timer

   while ((STCTRL & COUNT_FLAG) == 0)  // wait until flag is set
   {
      ;                          // do nothing
   }
   STCTRL = 0;                      // stop the timer

   return;
}

int main(void)
{
      SYSCTL_RCGC2_R |= 0x00000020;       /* enable clock to GPIOF */
      GPIO_PORTF_LOCK_R = 0x4C4F434B;     /* unlock commit register */
      GPIO_PORTF_CR_R = 0x1F;            /* make PORTF0 configurable */
      GPIO_PORTF_DEN_R = 0x1E;           /* set PORTF pins 4 pin */
      GPIO_PORTF_DIR_R = 0x0E;           /* set PORTF4 pin as input user switch pin */
      GPIO_PORTF_PUR_R = 0x10;           /* PORTF4 is pulled up */
      //STCURRENT = 0;
      while(1)
      {
         GPIO_PORTF_DATA_R = 0x0E;
         Delay(200-9);
         GPIO_PORTF_DATA_R = 0x00;
         Delay(800);
      }

}
```