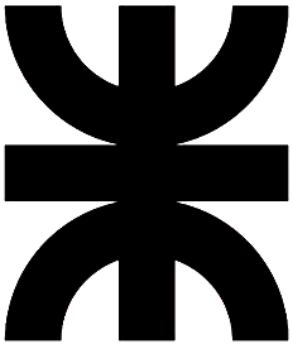
UNIVERSIDAD TECNOLÓGICA NACIONAL



FACULTAD REGIONAL PARANÁ

CARRERA: Ingeniería Electrónica
CÁTEDRA: Técnicas Digitales II

Trabajo Práctico N°7
Medidor de frecuencia - simple

ALUMNOS:
Battaglia Carlo
Escobar Gabriel

Paraná, 16 de noviembre de 2022

1. Desarrollo

1.1. El circuito se basará en la placa Arduino UNO, si lo cree conveniente, puede agregar una etapa de "driver" entre el generador y la placa Arduino.

1.2. Se inyectará una señal cuadrada, desde un generador de funciones, a un pin convenientemente seleccionado, para la medición de la frecuencia.

Para la generación de la señal cuadrada se utilizó un Arduino Mega2560 el cual ofrece una señal con una frecuencia de oscilación de 8$[MHz]$, 1$[MHz]$, 125$[kHz]$, 31,25$[kHz]$, 7,8125$[kHz]$, cuyos valores se corresponden a la señal sin preescalado del *clock*, *clock*/8, *clock*/64, *clock*/256 y *clock*/1024 respectivamente.

1.3. El funcionamiento general del circuito es:

1.3.1. Se inyectará una señal cuadrada y se deberá enviar el valor de la frecuencia en el monitor serie que trae el entorno de Microchip Studio.

1.3.2. El software del Arduino deberá hacer los ajustes de Hz, KHz, MHz, etc. cuando sea conveniente.

1.3.3. ¿Qué sucede si le cambia el ciclo de trabajo a la señal del generador de funciones?

Al cambiar el ciclo de trabajo a la señal del generador obtenemos el mismo funcionamiento debido a que la frecuencia de la señal no se ve afectada. Lo que varia es el instante donde la señal conmuta de un estado alto a uno bajo, pero como las interrupciones se producen únicamente en flancos ascendentes, el funcionamiento no se va a ver afectado debido a que el periodo de la señal continúa siendo el mismo.

1.3.4. ¿Qué sucede si le agrega un offset a la señal del generador de funciones?

Al agregar un offset a la señal se corre el riesgo de que, en cierto momento, la señal deje de operar como una señal cuadrada debido al efecto Schmitt Trigger que poseen las entradas (de PWM) del Arduino UNO.

Este efecto determina dos umbrales diferentes para establecer que la señal se encuentra en un estado lógico “HIGH” (alto) o un estado lógico “LOW” (bajo), por lo que si el offset agregado es tal que alcance el umbral inferior, una vez que la señal este en alto, por más que conmute, no llegara a tomar un valor bajo nuevamente y será tomada como una señal que se encuentra constantemente en alto por lo tanto no existirá conmutación y no se producirán interrupciones debido a que no se lograran distinguir los flancos ascendentes.

1.3.5. ¿Qué sucede si ingresa una sinusoidal?

Si ingresa una señal sinusoidal producirá un efecto similar al de la señal cuadrada debido a que una vez que se detecte que esta supere el umbral superior de la entrada del Arduino se tomará como una señal en alto y una vez que decaiga por debajo del umbral inferior se la tomará como una señal en bajo. La frecuencia de la señal no se verá afectada.

1.3.6. ¿Qué sucede si ingresa una triangular?

Ídem inciso anterior.

2. Código

```
volatile unsigned long timerCounts;
volatile boolean counterReady;
unsigned long overflowCount;
unsigned int timerTicks;
unsigned int timerPeriod;

void startCounting (unsigned int ms)  //cuenta ticks en un
   intervalo de ms pasado como argumento
{
  counterReady = false;
  timerPeriod = ms;
  timerTicks = 0;
  overflowCount = 0;

  // resetea timer1 y timer2
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR2A = 0;
  TCCR2B = 0;

  TIMSK1 |= (1<<TOIE1);   // habilita interrupción por
   overflow en el timer 1
```

```
21
22    // 16 MHz (62.5 nanosegundos por tick) con preescalado de
       128 el contador incrementa cada 8 microsegundos.
23    // 125 ticks = 1 ms
24    TCCR2A |= (1<<WGM21) ;    // timer 2 modo CTC (match)
25    OCR2A  = 124;             // interrumpe cuando llega a 125 (
       cero incluído)
26
27    TIMSK2 |= (1<<OCIE2A);    // habilita interrupción por match
       en timer 2
28
29    // resetea el registro de los contadores 1 y 2
30    TCNT1 = 0;
31    TCNT2 = 0;
32
33    GTCCR |= (1<<PSRASY);         // sacar preescalado
34    TCCR2B |= (1<<CS20) | (1<<CS22) ;  // usar preescalado de
       128
35
36    TCCR1B |=  (1<<CS10) | (1<<CS11) | (1<<CS12);   // detecta
       flanco ascendente en pin T1 (D5)
37  }
38
39  ISR (TIMER1_OVF_vect)   // función de callback para
       interrupción de overflow del timer 1
40  {
41    ++overflowCount;               // cuenta la cantidad de
       overflows en el timer 1
42  }
43
44  ISR (TIMER2_COMPA_vect)   // función de callback para
       interrupción de comparación del timer 2
45    {
46    unsigned int timer1CounterValue;
47    timer1CounterValue = TCNT1;
48    unsigned long overflowCopy = overflowCount;
49
50    if (++timerTicks < timerPeriod)
51      return;
52
53    // chequea si se perdió un overflow
54    if ((TIFR1 & (1<<TOV1)) && timer1CounterValue < 256)
55      overflowCopy++;
56
57    TCCR1A = 0;
58    TCCR1B = 0;
59
60    TCCR2A = 0;
61    TCCR2B = 0;
```

```
62
63    TIMSK1 = 0;
64    TIMSK2 = 0;
65
66    // calcula
67    timerCounts = (overflowCopy << 16) + timer1CounterValue;
       // (PROBAR overflowCopy * 65536)
68    counterReady = true;
69    }
70
71  void setup ()
72  {
73    Serial.begin(115200);
74  }
75
76  void loop ()
77  {
78    String unit;
79
80    startCounting (500);
81
82    while (!counterReady) { }
83
84    float frq = (timerCounts *  1000.0f) / timerPeriod;
85    if(frq < 1000.0) unit = "Hz.";
86    else if(frq < 1000000.0f)
87    {
88      unit = "kHz.";
89      frq = frq/1000.0f;
90    }
91    else
92    {
93      unit = "MHz.";
94      frq = frq/1000000.0f;
95    }
96
97    Serial.print ("Frequency: ");
98    Serial.print (frq);
99    Serial.println (unit);
100 }
```