Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica
IEE2473 – Laboratorio de electrónica analógica y digital

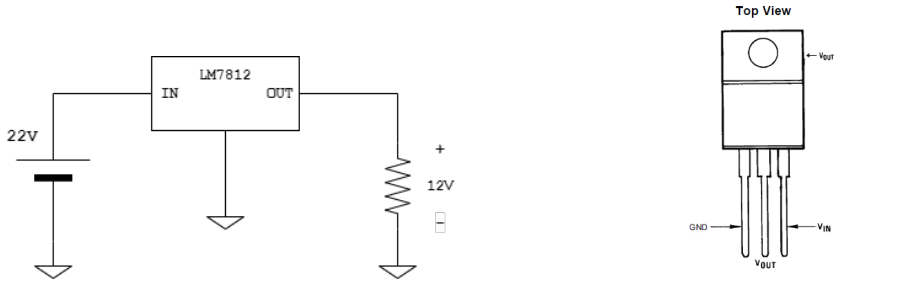
Experiencia 4: Sistema de Control de Temperatura

Grupo 17
Kevin Justiniano - Juan Lorca

Índice

1. Desarrollo de la Experiencia

1.1. Calefactor



(a) Circuito calefactor (b) Dibujo regulador [**1**]

Figura 1: Calefactor

Para la implementación de un calefactor que disipe un 1 W de potencia de implementó el circuito de la Figura-1a. El regulador utilizado fue el LM7812 el cual en condiciones de prueba [**1**] entrega a la salida el rango de voltajes (11.5, 12.5) V. Se definió para la entrada de este regulador un volatje de 22 V por lo que la caída de tensión entre entrada y salida de este fue de 10 V. Luego, con tal de que este regulador disipe 1 W, se conecta una impedancia R_L de 120 Ω, cuyo valor fue definido de acuerdo a la siguiente ecuación:

$$P_R = V \cdot I = 10 \cdot \frac{12}{R_L} = 1\ W \tag{1}$$

Si calculamos la potencia que disipa tal resistor de carga llegamos a:

$$P_{R_L} = \frac{12^2}{120} = 1.2\ W \tag{2}$$

lo cual es muy alto en comparación a la potencia tolerada por los resistores disponibles en el laboratorio y que fueron utilizados para esta experiencia. En solución a este problema, se tuvo que dividir la corriente que llega a la carga manteniendo la impedancia/resistencia de carga vista por el regulador. Con esto último en consideración, se conectaron 5 resitores R de 600 Ω en paralelo y se calculó que la potencia disipada en cada uno fue de 0.24 W, lo que se encuentra dentro del rango de toleracia de potencia para las resistencias utilizadas. El valor de los resistores anteriores fue determinado según la ecuación:

$$\frac{1}{R_L} = \sum_{i=1}^{N} \frac{1}{R} = \frac{5}{R} \tag{3}$$

En el armado del circuito, se optó por no utilizar disipadores térmicos puesto que para posteriores mediciones del sensor de temperatura no se lograron medir bien los cambios de esta y traducirlos a una variación de voltaje. El dibujo representativo

del regulador LM7812 utilizado se puede ver en la Figura-1b. Luego, el diseño final del circuito utilizado como estufa se puede ver en la Figura-2.

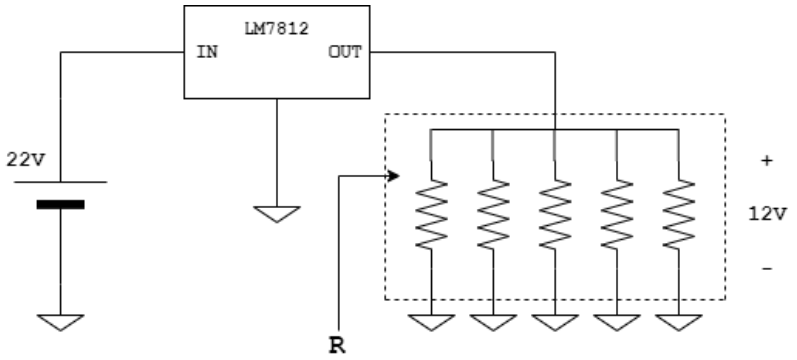


Figura 2: Diseño final calefactor

1.2. Ventilador y Control del Velocidad

1.2.1. MSP: Timer_A

El Timer_A de la MSP430 es un contador de 16 bits asíncrono a la ejecución de un set de instrucciones principales, por lo que puede actuar como un timer de interrupción. Además, la salida de este timer puede ser configurada como una señal PWM, lo que pudo haber sido de utilidad para esta experiencia.

El Timer_A es configurado por medio de sus registros:

- `TASSEL`: bits que eligen el `clk` asociado al contador.
- `TAIDEX`: bits asociados al valor de división del *source clk* definido anteriormente
- `TAxCCR0`: valor máximo del contador.
- `MC`: define el modo de control del timer. En este caso el modo `MC01` representa que el timer cuenta repetidamente desde cero hasta el valor de `TAxCCR0`.
- `OUTMODE`: 3 bits que definen el modo de salida. En particular el modo 7 define que la salida se reinicia cuando el timer cuenta hasta el valor `TAxCCRn`. Se establece cuando el temporizador cuenta con el valor `TAxCCR0`.

1.2.2. MSP: Señal PWM con *duty cycle* variable

Pese a las indicaciones e investigación anterior en el uso del contador `Timer_A` para la modulación y control del *duty cycle* de la señal PWM, se utilizó el framework de *Arduino* en la programación de la MSP. Este modo de trabajo nos entrega las funciones `analogFrequency(freq)` para definir la frecuencia `int freq` de la señal PWM y `analogWrite(Pin, dutyCycle)` escribir en el pin de la MSP `Pin` una señal PWM con un *duty cycle* definido por `int dutyCycle`. Esta variable `dutyCycle` es deinida por el programador y representa un entero de 8-bits que va desde 0 a 255, con 0 representando un *duty cycle* de un 1 % y 255 un *duty cycle* de un 99 %.

Con tal de definir un ciclo de trabajo variable, en el código asociado a la MSP se definió un contador de 8-bits el cual se incrementaba en 1 cada vez que el botón `PUSH_1` era presionado. Los resultados se esta señal PWM implementada se pueden ver en la Figura-4 donde se compara una señal cuadrada con ciclos de trabajo 1 %, 25 %, 50 %, 75 % y 99 % generada por el generador de funciones, con la PWM sintetizada por la MSP.

1.2.3. Driver para el Ventilador

El driver utilizado para accionar y luego incrementar o apagar la velocidad del motor DC asociado al ventilador viene dado por la Figura-3.

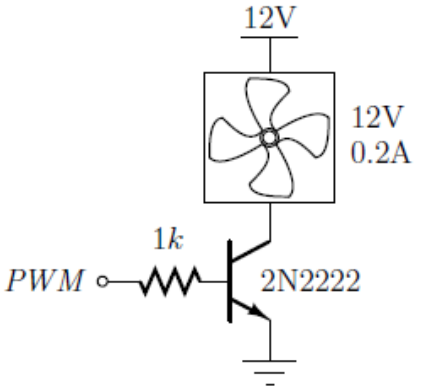


Figura 3: Driver ventilador [**2**]

La frecuecia de operación definida para nuestro sistema fue de 2 kHz. Utilizando esta última y el generador de funciones para excitar la base del transistor NPN con señales cuadradas de *duty cycle* variable y de amplitud 5 V, se obtuvieron los valores presentados en la Tabla-1 para corriente consumida por el ventilador respecto al ciclo de trabajo de la onda cuadrada:

duty cycle [%]	corriente [mA]
38	73
40	79
50	93
60	117
70	158
80	190
90	235
99	317

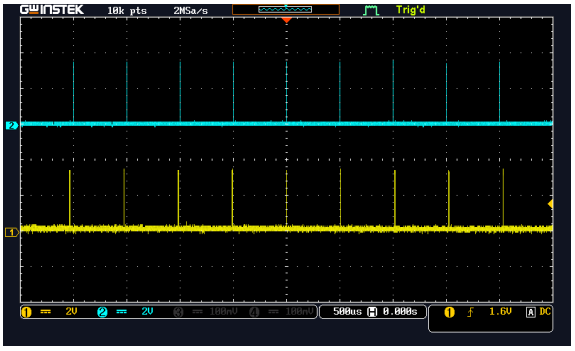
Tabla 1: Mapeo ciclo de trabajo - corriente

Dada esta tabulación, se pudo obtener una ecuación lineal para la corriente consumida por el ventilador y el *duty cycle* (d_c) de la onda cuadrada utilizada para accionar el driver:

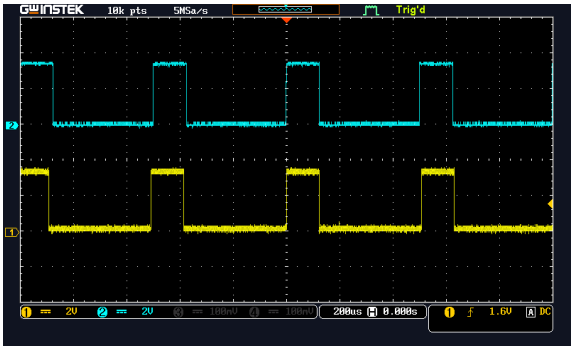
$$I(d_c) = 0.362 \cdot d_c - 0.0804 \tag{4}$$

Es importante destacar que para la frecuencia de operación definida, el torque asociado al motor del ventilador no lograba vencer la inercia para un d_c menor al 37 %.

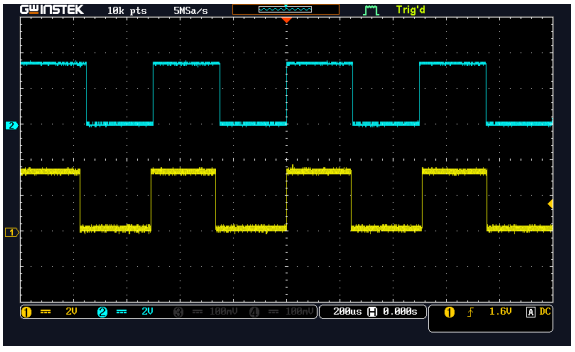
La siguiente figura demuestra el funcionamiento de la señal PWM programada (amarillo) y la generada a partir del generador de funciones (azul), graficadas en el osciloscopio del laboratorio.



(a) *duty cycle* 1 %

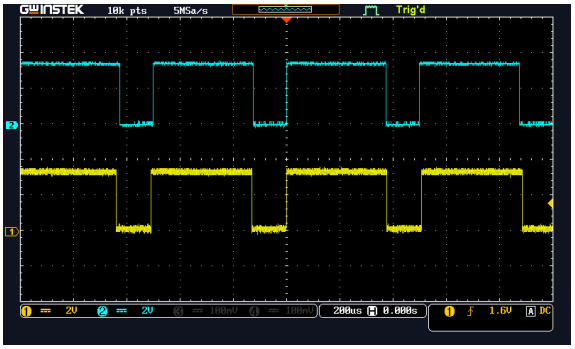


(b) *duty cycle* 25 %

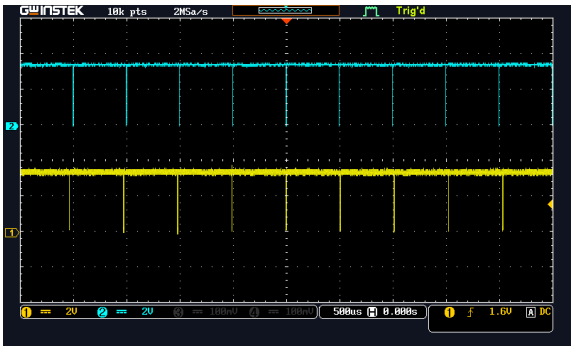


(c) *duty cycle* 50 %

Figura 4: Señales PWM



(a) *duty cycle* 75 %



(b) *duty cycle* 99 %

La intensidad de corriente que fluye a través del motor del ventilador determina la velocidad de operación de este último. **Esta velocidad puede ser controlada a partir del ciclo de trabajo de la señal PWM utilizada en el driver**. Dado que la señal PWM con *duty cycle* varible actúa como un switch con tiempo de ON-OFF variable, el voltaje, y por ende la corriente, **promedio en el tiempo** que reciben los terminales del motor DC es variable de acuerdo a que tan anchos o delgados son los pulsos, por lo tanto dependen del *duty cycle* de la señal PWM.

1.3. Sensor, Acondicionamiento y ADC

El sensor LM335 fue utilizado para la medición de la temperatura del regulador de voltaje. Este sensor opera en un rango de -40°C a 100°C de temperatura y con un rango de voltaje asociado de 1.8 V [**3**]. Con tal de polarizar el sensor y no pasarnos de los límites de corriente establecidos en el datasheet [**3**], utilizamos uno de los pines de 5V de la MSP como alimentación y una resistencia de 9.1 $k\Omega$. La Figura-6 muestra la vista inferior de este sensor.

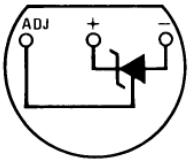


Figura 6: Vista inferior sensor de temperatura [**3**]

1.3.1. Acondicionamiento

Para acondicionar la señal recibia por el sensor de temperatura se utilizó el OpAmp LM358P que nos permitió utilizar la fuente de 5 V entregada por la MSP. El circuito implementado se muestra en la Figura-?? y las simulaciones realizadas en la Figura-8. Los valores de los resistores fueron escogidos tal de permitir un paso de corriente de salida que se hunda a tierra y por lo tanto tener una caída de tensión de pracicamente cero. En base a esto, las resistencias de polarización toman los valores grandes que se observan en el cicuito de acondicionamiento. Por último, se agrega una resistencia de 330 Ω para obtener un corriente dentro de los márgenes que puede entregar el OpAMP, así como definir elvoltaje máximo que puede recibir la MSP de 3.3 V.

En definitiva, la resistencia de salida está diseñada para entregar el valor máximo de corriente del OpAmP y en base a eso generar una tensión, mienras que la resistencias de polarización fueron diseñadas tal de entregar los valores cercanos a cero de la forma más precisa posible, ajustando la corriente que puede entrar siguiendo las gráficas del Datasheet y así hundir una corriente baja paraq ue la salida llegue cerca de los 0V.

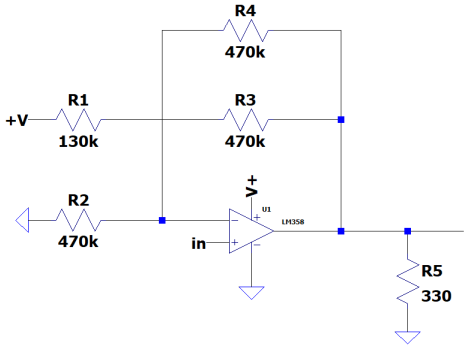


Figura 7: circuito acondicionador implementado con LM358P

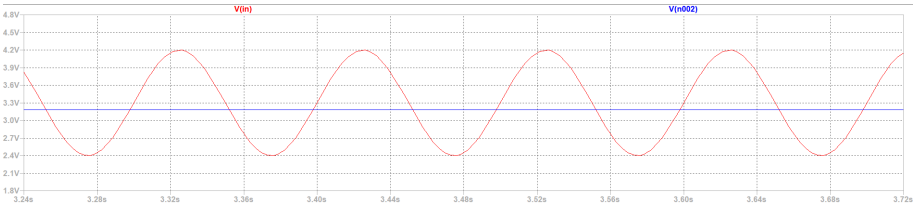


Figura 8: simulación del LM358

1.3.2. MSP: ADC *single-channel single − conversion*

Se define el modo de conversión con los dos bits dados por el registro `CONSEQx = 00`. Cada valor es muestreado y convertido a bits una vez y luego es escrito en una memoria asociada al ADC: `ADC12MEM`. El canal asociado al muestreo y a la conversión es seleccionado por medio de los bits `INCHx`, donde `x` va desde 0 a 15.

1.3.3. Lectura de Temperatura

Dado que el framework utilizado fue el de Arduino, sólo tuvimos que localizar un pin de la MSP encargado de leer señales analógicas. Nuestra elección fue el pin P6_0, y la función encargada de leer la información que llegaba a ese pin fue `analogRead(P6_0)`. Esta última función realiza la conversión análoga-digital y retorna un número cuantizado de 12-bits. Para leer el valor de voltaje real que se medía a la salida del bloque acondicionador se implementó la siguiente operación:

$$V = N \cdot \frac{V_{R^+} - V_{R^-}}{2^{12} - 1} \tag{5}$$

con N representando el valor de voltage digital y las referencias dadas por la MSP en $V_{R^+} = 3.3$ V y $V_{R^-} = 0.0$ V.

Además, para deshacerse del ruido del sensor sujeto a tierra, se tomaron 10 mediciones de temperatura las cuales se promediaron para definir la temperatura final que sería utilizada para cerrar el lazo de control.

1.4. Display Alfanumérico de 16x2

Se utilizó una pantalla LCD de 2 filas y 16 columnas. Esta pantalla posee de 16 pines digitales los cuales se dividen en: tierra de señal (Vss), alimentación de 5 V (Vdd), alimentación para el panel LCD (Vo), registro de selector de datos o instrucciones (RS), opción de lectura-escritura (RW), pin para habilitar comunicación de señales (EN), buses de datos (D0-D7) y pines para el control de luz de fondo (A-K). Se utilizaron 12 de los 16 pines disponibles, deshabilitando los pines de datos D0-D3. La conexión realizada para el funcionamiento del LCD fue la misma que el diagrama mostrado en [**2**], sólo que el pin de alimentación 5 V de la MSP fue utilizado. Tal diagrama se puede observar en la Figura-9

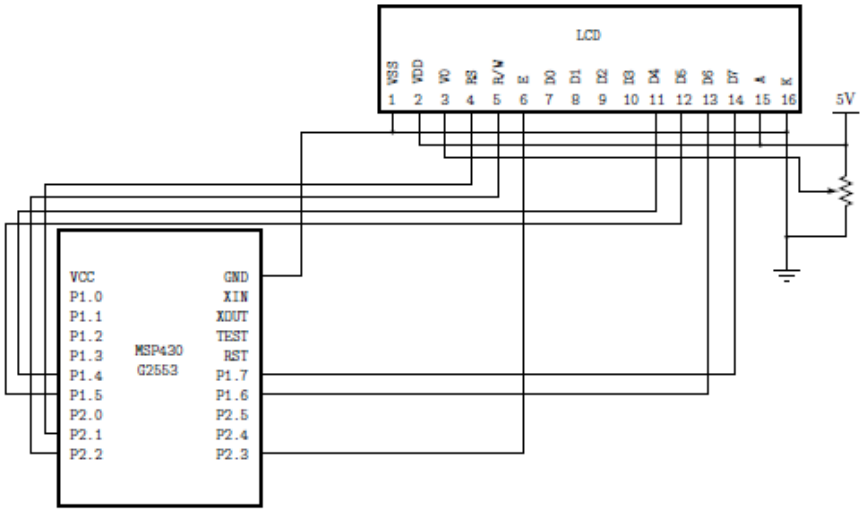


Figura 9: Diagrama LCD [**2**]

1.4.1. Protocolo de funcionamiento

A continuación se presenta el diagrama de señales en la Figura-10 y se describen cada uno de los pines utilizados. Es importante mencionar que los pines D0 a D3 son llamados *low data lines* no fueron utilizados, por lo que sólo tuvimos 4 bits disponibles para hacer operaciones en la pantalla. Otro alcance a considerar es que los pines utilizados del microcontrolador deben ser de *General Purpose Input/Output* (GPIO). Nosotros utilizamos aquellos pines disponibles en la MSP asociados a la función `digitalWrite()`.

1. **Vss:** GND.
2. **Vdd:** 5 V de alimentación.
3. **Vo:** Control de contraste para los caractéres enseñados en el display. Va conectado a un potenciómetro.
4. **RS:** Fue conectado a uno de los pines de la MSP. Si $RS = HIGH/ON$, la pantalla LCD puede recibir nuevos datos para mostrar en la pantalla, pero si $RS = LOW/OFF$, sólo recibe comandos.
5. **R/W:** Fue conectado a GND ya que para efectos de esta experiencia no nos importaba entregarle instrucciones de escritura a la lógica de control del LCD, sólo de lectura. Para ello $R/W = LOW/OFF$.
6. **EN:** Pin encargado del *light switch* para el display. Fijamos $E = HIGH/ON$ para enseñar los caracteres en el display.
7. **D4-D7:** Pines de datos y comandos enviados.
8. **A - K:** Ánodo y cátodo para el *back light*

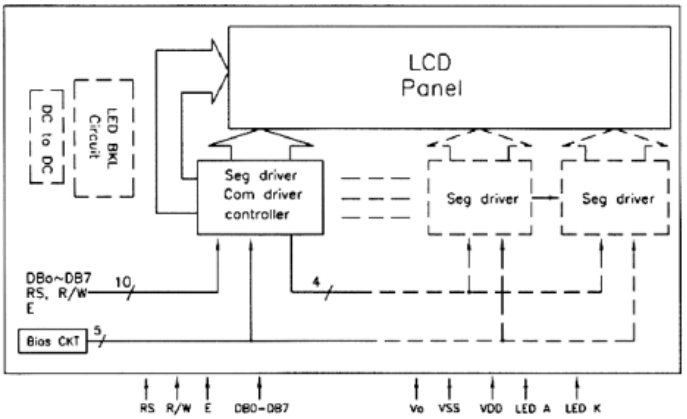


Figura 10: Diagrama de señales LCD [**4**]

Un pseudo código de lo que se debería hacer para hacer funcionar el display LCD sería:

```
(1) Revisar si el LCD se encuentra "ocupado";
(2) Encender la luz para enseñar caracteres:
    EN = 1;
(3) Que reciba instrucciones de lectura:
    R/W = 0;
(4) Decidir entre enviar dato o comando:
    if RS = 0: enviar comando;
    if RS = 1: enviar datos;
```

1.4.2. Programa Implementado en la MSP

Dado que la programación en la MSP fue de alto nivel, se utilizó la librería `LiquidCristal` para el manejo de las señales al display LCD. Se inicializó y configuró el display por medio del comando: `LiquidCrystal lcd(RS, EN, D4, D5, D6, D7)` y luego con el set-up del código se imprimieron los caracteres fijos dados por la temperatura (T), la potencia consumida por el ventilador (P), la ganancia proporcional del controlador (Kp) y la ganancia integral del controlador (Ki). La Figura-**??** muestra el display funcionando.

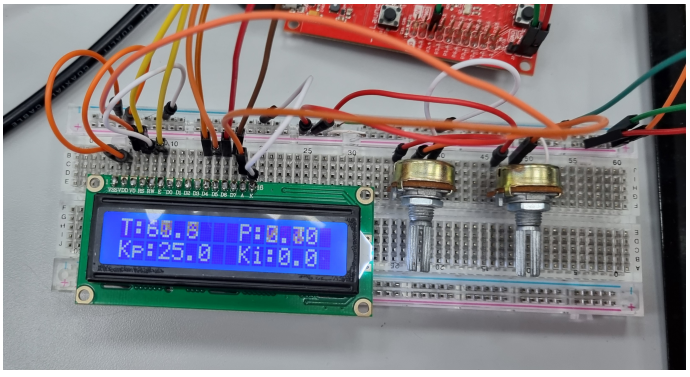


Figura 11: Display LCD

1.5. Control PI

1.5.1. Diseño del controlador

Se optó por un controlador PI discreto del tipo incremental. No se implementó un controlador PID simplemente por que la variación de temperatura es un proceso relativamente lento y una acción que se preocupe por la tasa de cambio de esta variable no tendría un gran efecto en el control. La ecuación para el controlador incremental fue la siguiente:

$$u[n] = u[n-1] + K_p(e[n] - e[n-1]) + Ki \cdot T_s \cdot e[n] \tag{6}$$

dónde $u[n]$ representa el valor de los actuadores en el tiempo n y $e[n]$ el error en tiempo real que viene dado por:

$$e[n] = T_{ref} - T[n] \tag{7}$$

con T_{ref} una señal de tipo escalón y $T[n]$ la temperatura del regulador de voltaje en el tiempo n. También, el valor de $T_s = 0.1\ s$ define el periodo de muestreo definido para el recorrido secuencial del código.

Se definieron valores para la saturación de los actuadores en 0 y 1000 respectivamente definiendo una función `bound()`. Luego, este valor fue mapeado al *dutyCycle* para la señal PWM escrita en el pin P2_5 de la MSP. El mapeo realizado fue el siguiente:

$$duty = \textbf{int}(255 \cdot \frac{u[n]}{1000}) \tag{8}$$

Esto último puede ser interpretado como la normalización requerida para definir el ciclo de trabajo de nuestra PWM, la cual va de un rango entre 0 y 255.

1.6. PC

1.6.1. Funcionamiento modo UART módulo USCI MSP430

Para acceder al modo UART por medio de los módulos USCI, los bits `USCI` del tipo `A` deben ser configurados. Al seleccionar los módulos `USCI_Ax` los *jumpers* Tx y Rx deben ser cortocircuitados para permitir comuniación serial por medio del puerto micro USB (MSP) al puerto USB (PC).

1.6.2. Envío de datos PC-MSP

Como el framework de Arduino para al programación de microcontroladores es de alto nivel, este dispone de los comandos `Serial` para el envío de datos de la MSP al PC o del PC hacia la MSP. Es necesario definir un *baud rate* en el set-up del código y así establecer una tasa de envío serial de los bits. En particular, las funciones utilizadas fueron: `Serial.println()` para imprimir caracteres enviados desde la MSP al PC en un terminal, y `Serial.readStringUntil()` con `Serial.parseFloat()` para enviar strings y números flotantes desde el PC (en un terminal) a la MSP.

El puerto serial utilizado fue el entregado por la extensión *Serial Monitor* del editor de código VS-Code/Plataform-IO. Este terminal nos permite escribir los datos-comandos a configurar y los envía serialmente estableciendo una comunicación PC-MSP en tiempo real.

1.7. Lazo de control

Finalmente, el lazo de control que representa el sistema de control de temperatura se puede observar en la Figura-12.

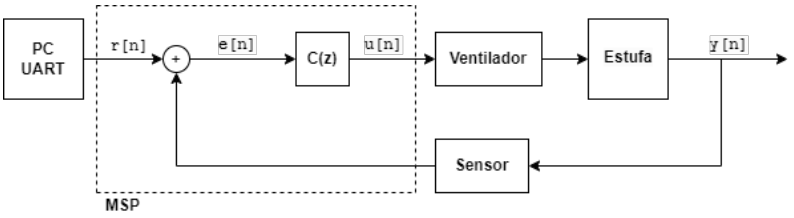


Figura 12: Lazo de control

Referencias

[1] Texas Instruments. (2016). *LM340, LM340A and LM78xx Wide VIN 1.5-A Fixed Voltage Regulators* (Rev. A, Julio 2016). Texas Instruments Incorporated.

[2] C. Garcés, *Experiencia 4: Control de Temperatura*. Laboratorio de Electrónica Analógica y Digital. IEE2473, Pontificia Universidad Católica de Chile, 2023.

[3] Texas Instruments. (1999). *LMx35, LMx35A Precision Temperature Sensors* (Rev. A, Febrero 2015). Texas Instruments Incorporated.

[4] Microtios Technology. (S.f.). *MTC16205D Datasheet*. Datasheet Archive. `https://www.datasheetarchive.com/datasheet?id=28971cfdea51120982da97f57cba7cd505c211&type=M&term=mtc16205d`