IMPLEMENTACIÓN DEL PROTOCOLO DE COMUNICACIÓN SPI EN VHDL PARA EL MODULO "PmodTC1"

Alumno: Chaparro Reyes Luis Edgar

chaparroreyes@gmail.com

Dr. Juan Antonio Jaramillo Gómez

jantonioj@yahoo.com

Instituto Politécnico Nacional (IPN)

Unidad Profesional Interdisciplinaria en Ingeniería

Y Tecnologías Avanzadas (UPIITA)

Abstract

En el presente trabajo, se muestra cómo implementar el protocolo de comunicación SPI en VHDL utilizando un módulo PmodTC1 de Digilent® con sensor de temperatura, que utiliza comunicación SPI para adquirir la medición de la temperatura en la unión fría del sensor, la temperatura de la unión del termopar tipo k y de los datos de fallas que sensa y convierte el ADC MAX31885 del PmodTC1, también se mencionan los principales datos a tomar en cuenta en la comunicación SPI como lo son la frecuencia del reloj de comunicación "SCK", El tiempo entre cada lectura del ADC "CS", el tamaño en bits del código del esclavo a leer, entre otros, así mismo se explica su importancia de cada uno para el diseño del programa en VHDL y así lograr la comunicación entre el maestro y el esclavo teniendo como finalidad encontrar el código binario del sensor y mostrar en los displays de la Nexys 2 la temperatura sensada y compensada en sistema decimal.

I. Introducción

La comunicación SPI por sus siglas en ingles "*Serial Peripheral Interface*" es un bus de tres líneas, sobre el cual se transmiten paquetes de información de "n" bits. Cada una de estas tres líneas porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es *full dúplex*, es decir, que dos de estas líneas trasfieren los datos (una en cada dirección) y la tercer línea es la del reloj. Algunos dispositivos solo pueden ser transmisores y otros solo receptores, generalmente un dispositivo que transmite datos también puede recibir.

Los dispositivos conectados al bus son definidos como maestros y esclavos. Un maestro es aquel que inicia la transferencia de información sobre el bus y genera las señales de reloj y control. Mientras que un esclavo es un dispositivo controlado por el maestro. Cada esclavo es controlado sobre el bus a través de una línea selectora llamada *Chip Select o Select Slave*, por lo tanto el esclavo es activado solo cuando esta línea es seleccionada. Generalmente una línea de selección es dedicada para cada esclavo. Toda la transferencia de los datos, son sincronizados por la línea de reloj de este bus. Un BIT es transferido por cada ciclo de reloj.

La mayoría de las interfaces SPI tienen 2 bits de configuración, llamados CPOL (*Clock Polarity* = Polaridad de Reloj) y CPHA (*Clock Phase* = Reloj de Fase). CPOL determina si el estado *Idle* de la

línea de reloj está en bajo (CPOL=0) o si se encuentra en un estado alto (CPOL=1). CPHA determina en que filo de reloj los datos son desplazados hacia dentro o hacia fuera. Por ejemplo si CPHA=0 los datos sobre la línea MOSI son detectados cada filo de bajada y los datos sobre la línea MISO son detectados cada filo de subida.

Cada BIT tiene 2 estados, lo cual permite 4 diferentes combinaciones, las cuales son incompatibles una de la otra. Por lo que si dos dispositivos SPI desean comunicarse entre sí estos deben tener el mismo, es decir la misma Polaridad de Reloj (CPOL) y la misma Fase de Reloj (CPHA). El modo requerido para una determinada aplicación, está dado por el dispositivo esclavo. La capacidad de multimodo combinada con un simple registro de desplazamiento hace que el bus SPI sea muy versátil. [1]

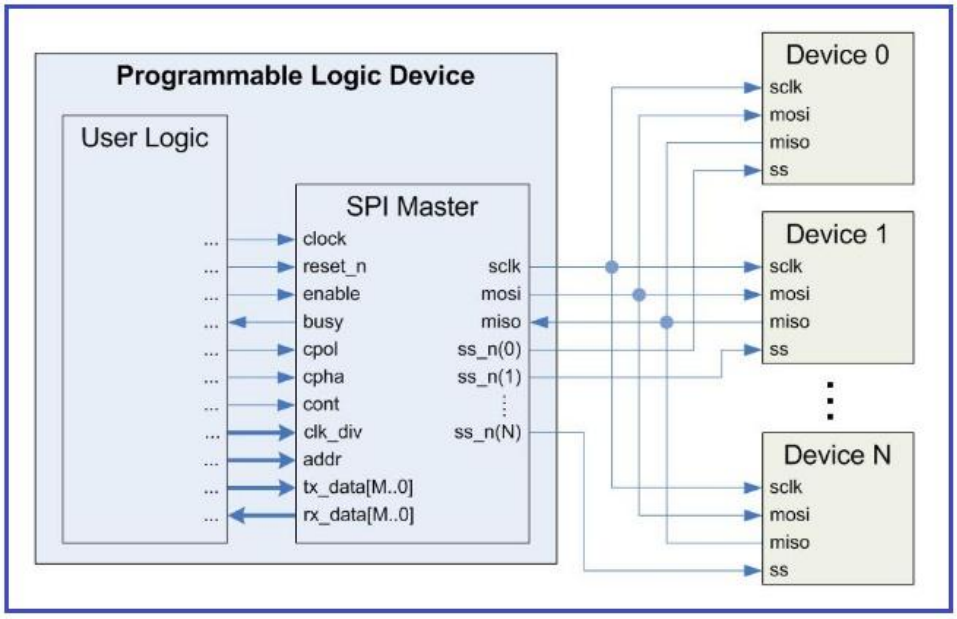

Figura 1. Ejemplo de Comunicación SPI con n esclavos. [2]

Para esta implementación se utiliza el sensor de Temperatura PmodTC1 de Digilent®, el cual está integrado por el ADC MAX31885 que es un convertidor analógico digital. Este módulo informa la temperatura sensada por un termopar de tipo K en 14 bits con una resolución de 0.25 °C es decir que tiene dos bits de números decimales, también nos da un numero de 12 bits que es la temperatura de compensación que está dada por la temperatura de la unión fría con una resolución de 0.0625 °C, es decir, utiliza 4 bits para números decimales, también da 3 bits reservados de valor 0 lógico y por ultimo 3 bits de detección de fallas, como se indican en [3], [4]. En la figura 2 se muestra el pmodTC1 con su termopar.

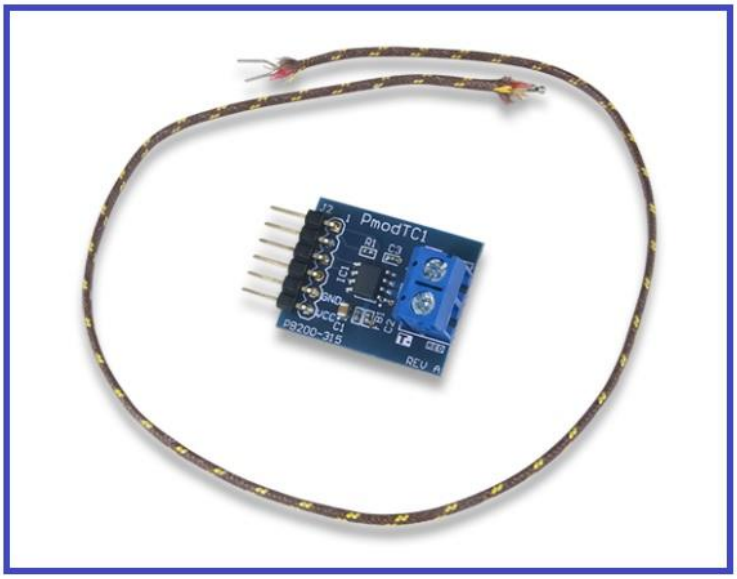

Figura 2. PmodTC1 con termopar tipo K. [3]

II. Desarrollo

Bien para comenzar con el diseño de nuestro programa en VHDL es necesario tener claro la frecuencia del reloj de comunicación con la que se va a trabajar para la comunicación SPI, ya que si no se tiene esta frecuencia no se puede iniciar la comunicación con el esclavo, el cual en esta aplicación es el PmodTC1, de igual forma es necesario saber el tiempo entre cada conversión del ADC MAX31885 ya que este indica el tiempo en alto de CS y el tiempo en bajo es para poder leer el código de 32 bits que nos da el PmodTC1, todos estos datos siempre se encuentran en la hoja de especificaciones del sensor que se vaya a utilizar y en este caso el *datasheet* del MAX31885. [4]

En las figuras 2 y 3 se muestran los datos necesarios para la comunicación SPI del *datasheet* y de igual forma este nos muestra cómo es que se da la comunicación SPI mediante los tiempos que este establece por diseño, ver figura 4.

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS
Thermocouple Temperature Data Resolution			0.25		°C
Internal Cold-Junction Temperature Error		-2		+2	°C
		-3		+3	
Cold-Junction Temperature Data Resolution			0.0625		°C
Temperature Conversion Time (Thermocouple, Cold Junction, Fault Detection)	t_{CONV}		70	100	ms
Thermocouple Conversion Power-Up Time	t_{CONV_PU}	200			ms

Figura 2. Parámetros de Conversión y tiempos de conversiones. [4]

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS
Input Leakage Current	I_{LEAK}	-1		+1	µA
Input Capacitance	C_{IN}		8		pF
Serial-Clock Frequency	f_{SCL}			5	MHz
SCK Pulse-High Width	t_{CH}	100			ns
SCK Pulse-Low Width	t_{CL}	100			ns
SCK Rise and Fall Time				200	ns
$\overline{CS}$ Fall to SCK Rise	t_{CSS}	100			ns
SCK to $\overline{CS}$ Hold		100			ns
$\overline{CS}$ Fall to Output Enable	t_{DV}			100	ns
$\overline{CS}$ Rise to Output Disable	t_{TR}			40	ns
SCK Fall to Output Data Valid	t_{DO}			40	ns
$\overline{CS}$ Inactive Time		200			ns

Figura 3. Parámetros de tiempos para la comunicación SPI. [4]

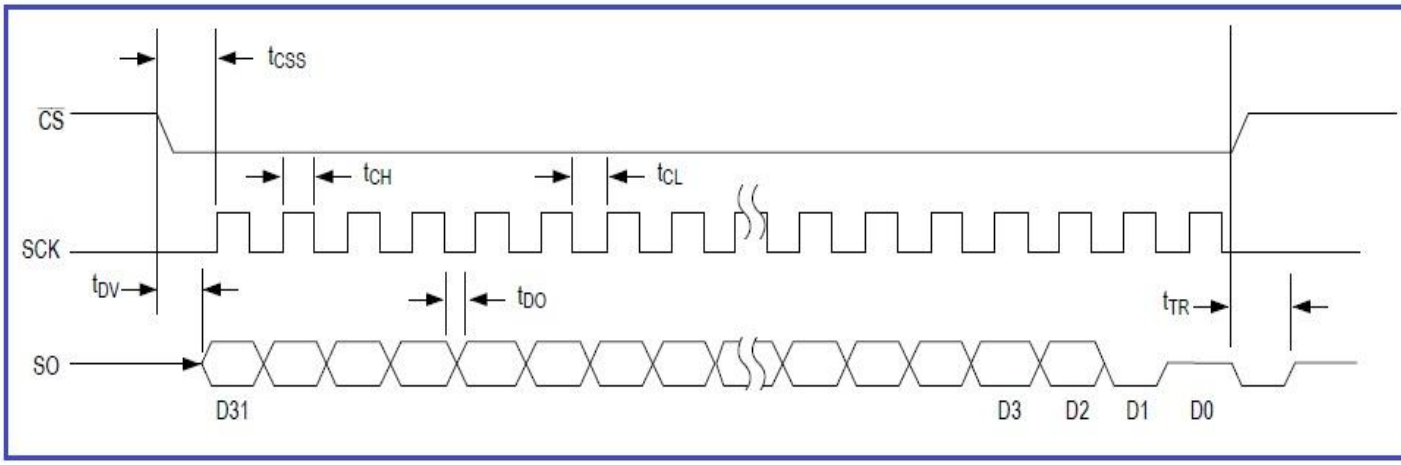

Figura 4. Tiempos de la Interfaz Periférica en Serie. [4]

Después de obtener los datos del datasheet, se hace un estudio de los datos y de los tiempos que requiere el sensor para el protocolo de comunicación SPI, en este caso en el *datasheet* del PmodTC1 se observa que los datos de la figura 2 y 3 al identificarlos en la figura 4 se tiene un tiempo de 100ns entre el flanco de bajada de CS y de la primer adquisición del código binario de 32 bits del esclavo que es el tiempo t_{CSS}, de igual forma hay un tiempo de 100ns entre la última adquisición del código del esclavo y el flanco de subida de CS, la frecuencia del reloj de comunicación es de 5MHz por lo que el tiempo en alto y en bajo es de 100ns para cada uno, dando un periodo igual a 200ns. También se obtuvo el tiempo entre cada adquisición de 32 bits que se puede elegir mientras los tiempos estén entre el intervalo de 70ms y los 100ms, por lo que se propone el uso de un tiempo de 80ms entre cada adquisición de 32 bits del esclavo (PmodTC1) mediante el master (Nexys 2) ya que esta entre el valor típico y el valor máximo permitido por el datasheet del MAX31885.

Para el diseño de esta comunicación SPI con el PmodTC1 no es necesario programar con CPOL y con CPHA ya que solo se va hacer la comunicación con un solo esclavo y no es necesaria la versatilidad de comunicar en los cuatro casos que se describen en [1] [5] y se toma el caso con el que se trabaja según el *datasheet* en el cual por la figura 4 se observa que CPOL=0 y que CPHA=0 dándonos como resultado el caso D de [1] y así el diseño se simplifica al comenzar el reloj de comunicación en un estado bajo y que cada lectura de los 32 bit se haga cuando este el flanco de subida de este reloj.

Una vez que se tienen los datos anteriores ahora es necesario saber cuánto es el tiempo que va a durar CS en estado bajo ya que es el tiempo de adquisición de datos, este tiempo se obtiene sumando el tiempo de los 32 bits, los 100ns antes de comenzar la adquisición y los 100ns después de terminar la adquisición, como se puede ver la suma de estos dos últimos tiempos nos da como resultado un ciclo del reloj de comunicación y los 32 bit necesitan 32 ciclos por lo que el resultado en tiempo de comunicación (reloj SCK) es de 32 ciclos que es igual a 6600ns en bajo y un tiempo de 80ms en alto para CS.

Ahora se procede hacer el código en VHDL ya que se tienen todos los datos necesarios para realizar el tipo D de comunicación SPI para cuando CPOL=0 y CPHA=0. Para empezar se declaran los puertos de entrada (clk, MISO, Enc, reset) y de salida (SCK, SS, MUX, salida). En la

arquitectura se crea un proceso para generar el reloj de comunicación de 5MHz con un divisor de frecuencia del reloj interno del FPGA a utilizar y comenzando con un estado bajo como se muestra en figura 5, y después se hace otro proceso para generar CS que en el código lo escribí como SS con 80ms en estado alto y 6600ns en estado bajo como se ve en la figura 6. Cabe mencionar que utilice señales para trabajar internamente con los relojes SCK y SS llamados en el código clk_5 y clk_SS respectivamente.

```
RelojSCK: process (clk) begin
   if(clk'event and clk='1') then
      cnt <= cnt+1;
      if(cnt = 5) then
         clk_5 <= not clk_5;
         cnt <= 1;
      end if;
   end if;
end process RelojSCK;
SCK <= clk_5;
```

Figura 5. Reloj de comunicación SCK a 5MHz.

```
RelojSS: process (clk) begin
   if(clk'event and clk='1') then
      if(cnt2 = 1600000) then
         clk_SS <= '0';
      else
         cnt2 <= cnt2+1;
      end if;

      if(cnt2 = 1600330) then
         clk_SS <= '1';
         cnt2 <= 1;
      else
         cnt2 <= cnt2+1;
      end if;
   end if;
end process RelojSS;
SS <= clk_SS;
```

Figura 6. Reloj de SS para la adquisición de los 32 bits.

Después, se genera el proceso para la activación de una máquina de estados que tiene como activación la señal "Enc" de encendido y a su vez se activa cuando el reloj SS está en estado bajo para poder hacer la adquisición de los 32 bits del esclavo, con cambios entre estado actual y estado futuro se hace mediante el flanco de subida del reloj de comunicación "SCK", y por ultimo un botón de reset para paralizar en cualquier momento la adquisición de datos, quedándose con el valor de la adquisición anterior de haber oprimido dicho botón, como se ve en la figura 7.

```
Act_Maquina: process(Enc, clk_SS, clk_5, reset) begin
   if (Enc='1') then
      if (clk_SS='0') then
         if (reset ='1') then       --En caso de darse un Reset
            Estado_actual <= E0;
         elsif (clk_5'event and clk_5='1') then
            Estado_actual <= Estado_siguiente;
         end if;
      else
         Estado_actual <= E0;
      end if;
   else
      Estado_actual <= E0;
   end if;
end process Act_Maquina;
```

Figura 7. Activación de la Maquina de Estados para la Adquisición de los 32 bits del esclavo.

Por último se hace el proceso de la máquina de estados donde se hace la adquisición del código binario de 32 bits del PmodTC1 por lo que se necesitan 33 estados ya que son 32 estados para la adquisición y un estado extra para permanecer ahí durante el tiempo en que CS está en estado alto, es decir el tiempo en que el PmodTC1 hace su conversión analógica a digital de la temperatura del termopar tipo k, la temperatura de la unión fría, de los 3 bits reservados y de la verificación de fallas por cortos en Vcc o Gnd o por no estar conectado el termopar como se describe en [4]. En la siguiente figura se muestra el proceso de la máquina de estados pero de forma abreviada ya que se repite el código en los 32 estados de adquisición.

```
Maquina: process(Estado_actual, MISO) begin
   case Estado_actual is
      when E0 =>
         Datos   <= Datos;
         num_bin <= num_bin;
         Estado_siguiente <= E32;
      when E32 =>
         num_bin(11) <= MISO;
         Estado_siguiente <= E31;
      when E31 =>
         num_bin(10) <= MISO;
         Estado_siguiente <= E30;
      when E30 =>
         num_bin(9) <= MISO;
         Estado_siguiente <= E29;
               :   :   :
               :   :   :
               :   :   :
      when E3 =>
         Datos(2) <= MISO;
         Estado_siguiente <= E2;
      when E2 =>
         Datos(1) <= MISO;
         Estado_siguiente <= E1;
      when E1 =>
         Datos(0) <= MISO;
         Estado_siguiente <= E0;
   end case;
end process Maquina;
```

Figura 8. Máquina de Estados para la Adquisición de los 32 bits del PmodTC1.

Una vez terminada la máquina de estados y los demás procesos ya se puede obtener el código binario de 32 bits del PmodTC1 y así poder trabajar con este código, de esta misma forma se pueden programar otros sensores o cualquier dispositivo que use el protocolo SPI como interfaz de comunicación y solo es necesario ver los valores de CPOL y CPHA para ver si el reloj de comunicación "SCK" comienza en alto o en bajo y de igual forma si la adquisición del código del esclavo se hace en su flanco de subida o de bajada como se ve en [1] [5].

III. Resultados

Una vez teniendo el programa en VHDL solo queda hacer unas pruebas para comprobar su correcto funcionamiento, estas pruebas constan de conectar el PmodTC1 a una FPGA (Nexys 2) la cual hará la adquisición de la temperatura del termopar mediante el protocolo de comunicación SPI desplegando su valor en los 4 display de la Nexys 2, pero solo la parte entera es decir los primeros 12 bits de los 32 adquiridos, cabe mencionar que el valor obtenido de la temperatura del termopar va ser más grande que la temperatura real ya que es necesario hacer una compensación con el numero obtenido por la temperatura de unión fría que hace que quede igual a la temperatura ambiente y de ahí sense como si fuera un termómetro o un sistema de medición de temperatura convencional, por lo que es necesario compensar la temperatura que se va a mostrar pues es el resultado de restar la temperatura de unión fría o de referencia a la temperatura del termopar pero no es una resta sencilla, ya que estos dos números que da el PmodTC1 tienen un bit de signo por lo que su número real puede tener

valores negativos o positivos según el valor de este bit y para obtener este valor numérico es necesario seguir el algoritmo de resta de números con bit de signo descrito en [6] que explica cómo obtener un numero negativo de un código binario y como sumar o restar estos tipos de números binarios. También, hay que hacer una conversión de un código binario de 12 bits a un código de 16 bit donde se pueda leer el valor del código binario en números binarios del 0 al 9 en bloques de 4 bits para lograr desplegarlos como unidades, decenas, centenas y unidades de millar siendo este el mismo número en código binario compensado pero ahora para poder observarlo en el los 4 display de la Nexys 2 como se hace en [7]. Finalmente se utiliza una compuerta OR para los 3 bits reservados y para los bits de detección de fallas ya que estos siempre deben tener un valor lógico de "0" pero si detecta una falla manda un valor lógico "1" y este hará que con la compuerta se encienda un led de la Nexys. Los valores decimales de la temperatura compensada se ven en cuatro LED's ya que son cuatro bits de números decimales. En la figura 9 se muestra la conexión del Pmod a la Nexys 2 y como es que sensa la temperatura ambiente y se tiene a su lado un multímetro con medición de temperatura, se observa que se llega a tener un valor muy cercano a la temperatura ambiente real sensada por el multímetro dando como resultado una correcta comunicación SPI.

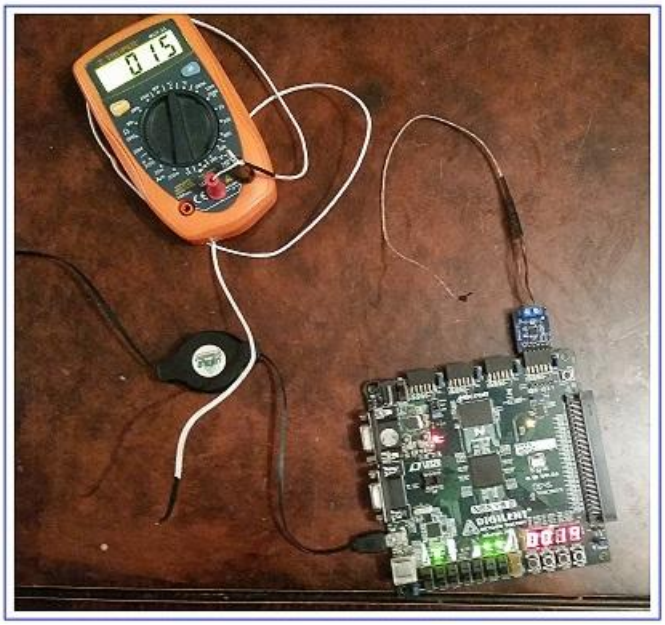

Figura 9. Conexión de PmodTC1 y Nexys 2, mostrando temperatura ambiente.

El código completo en VHDL para la comunicación SPI con el PmodTC1 lo dejo en el siguiente link de descarga:

https://www.dropbox.com/sh/vgz0bpk3bh2yszo/AADJEgQFFV5YjaFgmW3XmSTWa?dl=0

En el siguiente link se muestra el funcionamiento del código en VHDL en la Nexys 2 y con el PmodTC1 para que observen los resultados del protocolo de comunicación SPI.

https://youtu.be/aKEgdi5Gzy4

IV. Conclusiones

El protocolo de comunicación SPI es sumamente estricto en los tiempos que necesita para la comunicación ya que si no es la frecuencia a la que debe ir el reloj de comunicación no hace la adquisición de datos y envía datos erróneos, incluso ni siquiera hace la comunicación, de igual forma si no se observa como es la configuración entre el maestro y el esclavo se tendrán errores de cuadratura y no se sabrá que valores tienen CPOL y CPHA y por lo tanto no sabe cómo es la comunicación ni de que caso para poder así determinar cómo empezar el reloj de comunicación si en alto o en bajo y si debes detectar bits en el flanco de bajada o de subida, obteniendo nuevos errores de comunicación de igual forma se tiene que calcular o ver el tiempo inactivo de la comunicación SPI ya que el ADC con el que se trabaje siempre va a necesitar un tiempo para poder realizar su conversión analógica a digital.

Como recomendación siempre guiarse para hacer cualquier programa de protocolo de comunicación SPI observar el *datasheet* del componente con el que se va a trabajar ya que ahí están todas sus especificaciones y están documentadas por diseño de fabricación y es un hecho que estas sirven para lograr la comunicación entre master y esclavo, en mi caso los Pmod de DIGILENT® casi siempre cuentan con programas ejemplo para programación en VHDL o Verilog y el protocolo de comunicación con el que trabaja, pero para el módulo PmodTC1 no había y fue tedioso aprender cómo hacerlo sin una base, habían códigos pero ninguno se semejaba a lo que necesitaba este componente y lo tuve que hacer desde cero y siempre me base en su *datasheet* para realizar la comunicación SPI y el despliegue de la temperatura compensada.

V. Referencias

[1] Ingeniería en Microcontroladores, "Protocolo SPI (Serial Peripherical Interface)", México Distrito Federal. [En línea]. Disponible en: http://www.i-micro.com/pdf/articulos/spi.pdf Consultado el 12 de Noviembre de 2017.

[2] S. Larson, R. Nelson, "Serial Peripheral Interface (SPI) Master (VHDL)". (2017, Septiembre 8). [En línea]. Disponible en: https://eewiki.net/pages/viewpage.action?pageId=4096096 Consultado el 12 de Noviembre de 2017.

[3] DIGILENT®, "PmodTC1". National Instruments Company. [En línea]. Disponible en: https://reference.digilentinc.com/reference/pmod/pmodtc1/start Consultado el 12 de Noviembre de 2017.

[4] Datasheet MAX31855. "Cold-Junction Compensated Termocouple to Digital Converter". [En línea]. Disponible en: https://datasheets.maximintegrated.com/en/ds/MAX31855.pdf. Consultado el 13 de Noviembre de 2017.

[5] Embedded Micro. "Serial Peripheral Interface (SPI)". (2015). [En línea]. Disponible en: https://embeddedmicro.com/tutorials/mojo/serial-peripheral-interface-spi. Consultado el 14 de Noviembre de 2017.

[6] www.profesores.frc.utc.edu.ar. "D001-ARITMETICA". [En línea]. Disponible en: http://www.profesores.frc.utn.edu.ar/electronica/tecnicasdigitalesi/pub/file/cursoCavallero/D_001-ARITMETICA.pdf. Consultado el 14 de Noviembre de 2017.

[7] C. Ramos, "De binario a siete segmentos: la conversión". (2016, Mayo 16). [En línea]. Disponible en: http://www.estadofinito.com/binario-bcd-7seg/. Consultado el 15 de Noviembre de 2017.