
	UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos	Semestre: 2021-2	
	Prof. Dr. Saúl De La Rosa Nieves	Página 1 de 18	

TAREA-EXAMEN 2	
Título:	Lectura de sensores con CAN
Fecha:	17-08-2021
Preparado por:	Fiel Muñoz Teresa Elpidia
Evaluación:	

I. Planteamiento del proyecto

El planteamiento de este proyecto es la de lograr una comunicación con el protocolo CAN utilizando dos microcontroladores TM4C1294NCPDT, se debe hacer la lectura de datos de un microcontrolador al otro utilizando variables leídas de un sensor, esto a grandes rasgos.

A detalle lo que se realizó fue lograr una comunicación y transmisión continua de dos sensores analógicos que se traducen en dos potenciómetros, y que al llegar a un valor determinado (que puede variar dependiendo del sensor) encienda un led específico para cada potenciómetro, estos leds se encuentran en la tiva que recibe los datos, la que transmite es la que tiene los sensores.

II. Requerimientos del proyecto

Requerimientos de hardware

1. Como unidad de control del “Display” se utilizará el microcontrolador TM4C1294NCPDT.
2. Se deben obtener los datos de lectura de un sensor.

Requerimientos de software

1. Se debe de utilizar el protocolo de comunicación CAN.

III. Metodología

Como se mencionó anteriormente, este proyecto utilizará el protocolo CAN, por lo que la realización del proyecto requiere conocimientos previos sobre el protocolo además de conceptos vistos en microcontroladores y microprocesadores. La forma de realizarlo se comprende en los siguientes pasos:

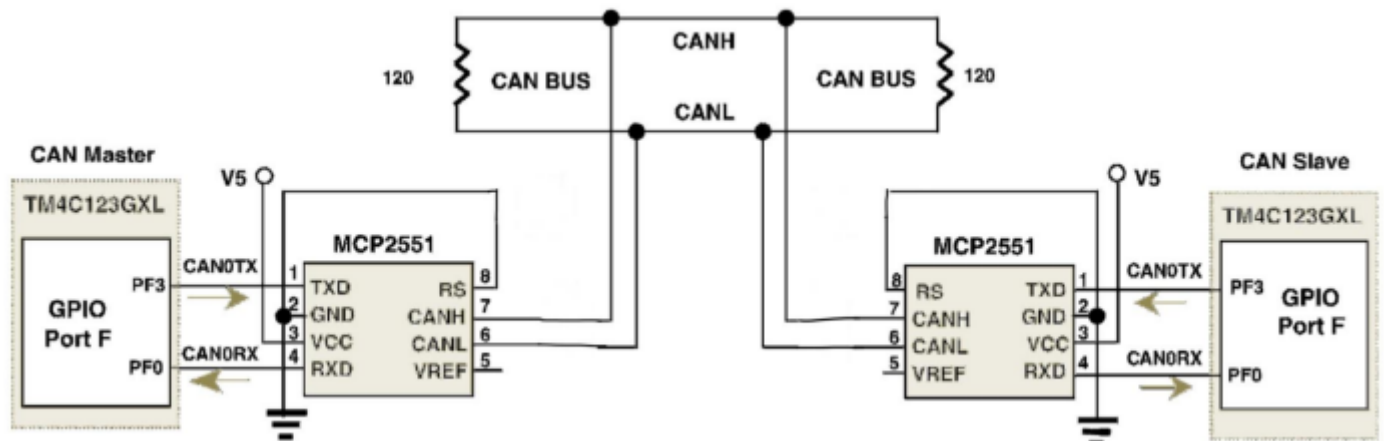
1. Primero se debe definir el planteamiento del proyecto, la parte principal de esto es realizar la comunicación CAN a través de dos microcontroladores.

- Posterior a eso se plantearon los requerimientos del proyecto, esto es a nivel de hardware como de software, se definen pocos de ellos porque la tarea se asignó verbalmente durante la clase, por lo que existen muchas partes que se dejan a criterio del diseñador, por eso solo incluí lo más básico a realizar.
- Una vez hecho esto, se procede a realizar el proyecto con un diseño preliminar, se plantea el diseño total de todo el sistema compuesto en subsistemas, en la lectura del ADC, en la comunicación CAN y en el encendido de leds para cuando los sensores llegaran a cierto nivel.

IV. Diseño conceptual

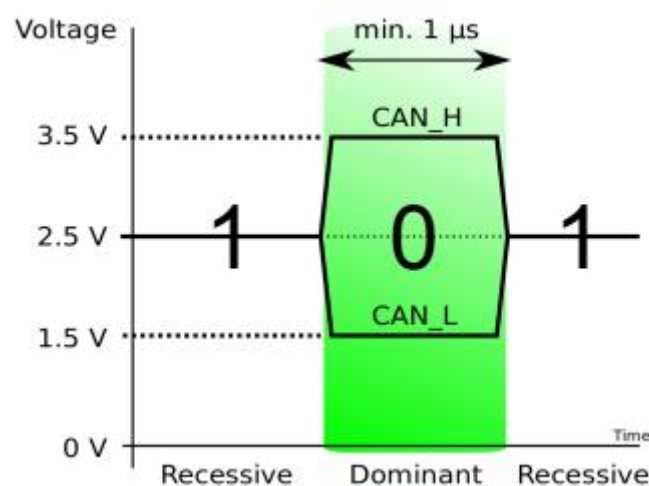
Bus CAN

Lo primero que hay que señalar en esta capa, es que CAN utiliza niveles lógicos diferenciales CAN_H y CAN_L. Se le denomina recesivo al 0 y dominante al 1. Para poder generar estas señales diferenciales se deberá utilizar un transceiver, el cual, se encargará de monitorear y generar estas señales diferenciales.



Capa física

Esta capa maneja dos niveles lógicos, que se definen a través de voltajes diferenciales y se nombran como dominante y recesivo, el dominante es el que tiene la mayor prioridad a la hora de mandar un dato. Es por eso que el dominante es un cero y se define como un voltaje diferencial en un valor aproximado de 3.5V, para el recesivo o CANL es de 1.5V.

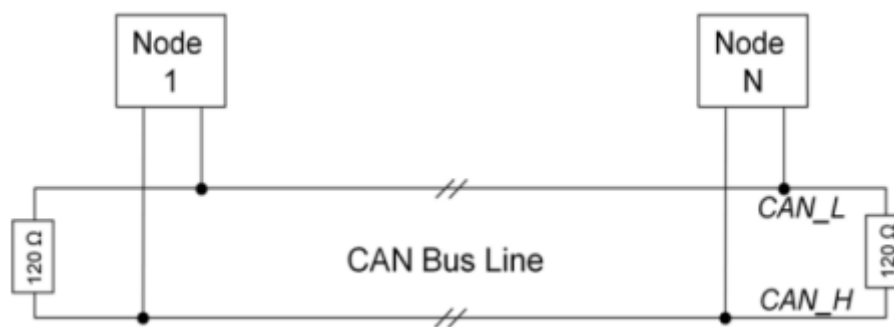


El flujo de bits en la línea de bus CAN se codifica de acuerdo con el método No retorno a Zero (NRZ) con relleno de bits (bit-stuffing). En aquellas partes del flujo de bits donde se aplica el método de relleno de bits, el transmisor, después de haber enviado cinco bits consecutivos de valor idéntico, insertará ("relleno") un bit adicional de valor inverso en el flujo de bits. El receptor reconocerá una secuencia de cinco bits consecutivos de idéntico valor y descartará el siguiente bit de relleno.

Cuando un nodo detecta seis bits consecutivos iguales en un campo susceptible de ser relleno lo considera un error y emite un error activo. Un error activo consiste en seis bits consecutivos dominantes y viola la regla de relleno de bits.

La capa física es la responsable de la transferencia de bits entre los distintos nodos que componen la red. Define aspectos tales como niveles de señal, codificación, sincronización y tiempos de bit. En CAN destacan principalmente dos configuraciones de la capa física: high-speed CAN y low-speed CAN. Ambas se basan en un par trenzado de cables, siendo la única diferencia que high-speed CAN ofrece una tasa de bits de 125 kbps a 1 Mbps y en cambio low-speed CAN de 10 kbps a 125 kbps. Ya sea en el caso de high-speed CAN o low-speed CAN los dos cables que utilizan reciben el nombre de CAN_H y CAN_L.

El objetivo de usar cables de par trenzado es el filtrado de las interferencias que pueden ocurrir en el bus. Para el envío de datos es necesario codificar estos mismos, CAN para esto utiliza una codificación NRZ (Non Return to Zero). A continuación, se explicarán las características tanto del bus High-speed CAN como del bus Low-speed CAN. El bus de High-speed CAN es un bus pasivo ya que al contrario que Low-speed CAN, el cual será activo, no tiene ninguna fuente externa de alimentación. Tanto highspeed CAN como low-speed CAN presenta una red cableada, en el caso de CAN H con una resistencia de $120\ \Omega$ en sus extremos cerrando el circuito, como se puede observar en el circuito puesto al inicio de esta parte.



CAN bit

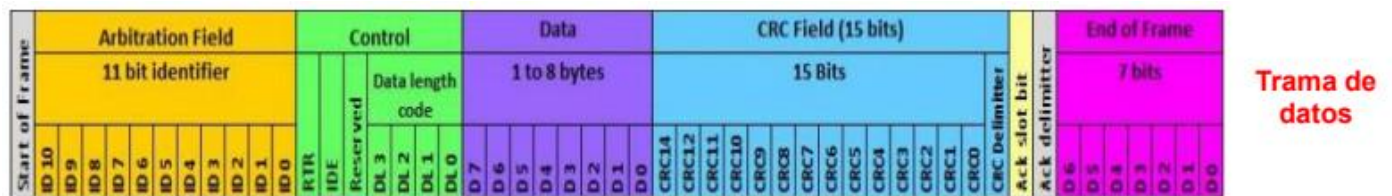
- Cada nodo tiene su propio reloj;
- Los parámetros de tiempo de bit se pueden configurar en cada nodo;
- Por lo tanto, se puede tener una tasa de bit común a pesar de la diferencia en los osciladores de cada nodo;
- Los nodos CAN compensan las diferencias entre tasas de bit mediante la "sincronización periódica en la trama de bits"

El reloj para cada uno de los nodos en bus CAN, se definirá por medio de los tiempos cuánticos, los cuales son la resolución mínima del tiempo de bit. Estos nos ayudarán a elegir de manera adecuada el tiempo para cada uno de los segmentos de nuestro reloj. A continuación, se definen algunas recomendaciones para considerar en cuántos tiempos cuánticos se recomienda dar a cada uno de los segmentos:

Segmento de Sincronización	1 tiempo cuántico (valor fijo)
Segmento de propagación	de 1 a 8 tiempos cuánticos
Segmento de Phase 1	de 1 a 8 tiempos cuánticos
Tiempo de procesamiento	≤ 2 tiempos cuánticos
Segmento de Phase 2	menor o igual al valor del segmento 1 + el tiempo de procesamiento

Trama de dato

En esta trama un elemento del bus transmite un dato a los demás nodos del bus y todos los reciben, sin embargo, de acuerdo con el identificador del mensaje enviado, así como la máscara y el filtro del receptor se logrará establecer si el mensaje es de interés para cada uno de los nodos que lo recibieron.



Trama remota

En esta trama no se va a mandar información, sino que se encarga de solicitar información alguno de los nodos conectados al bus.



Arbitraje

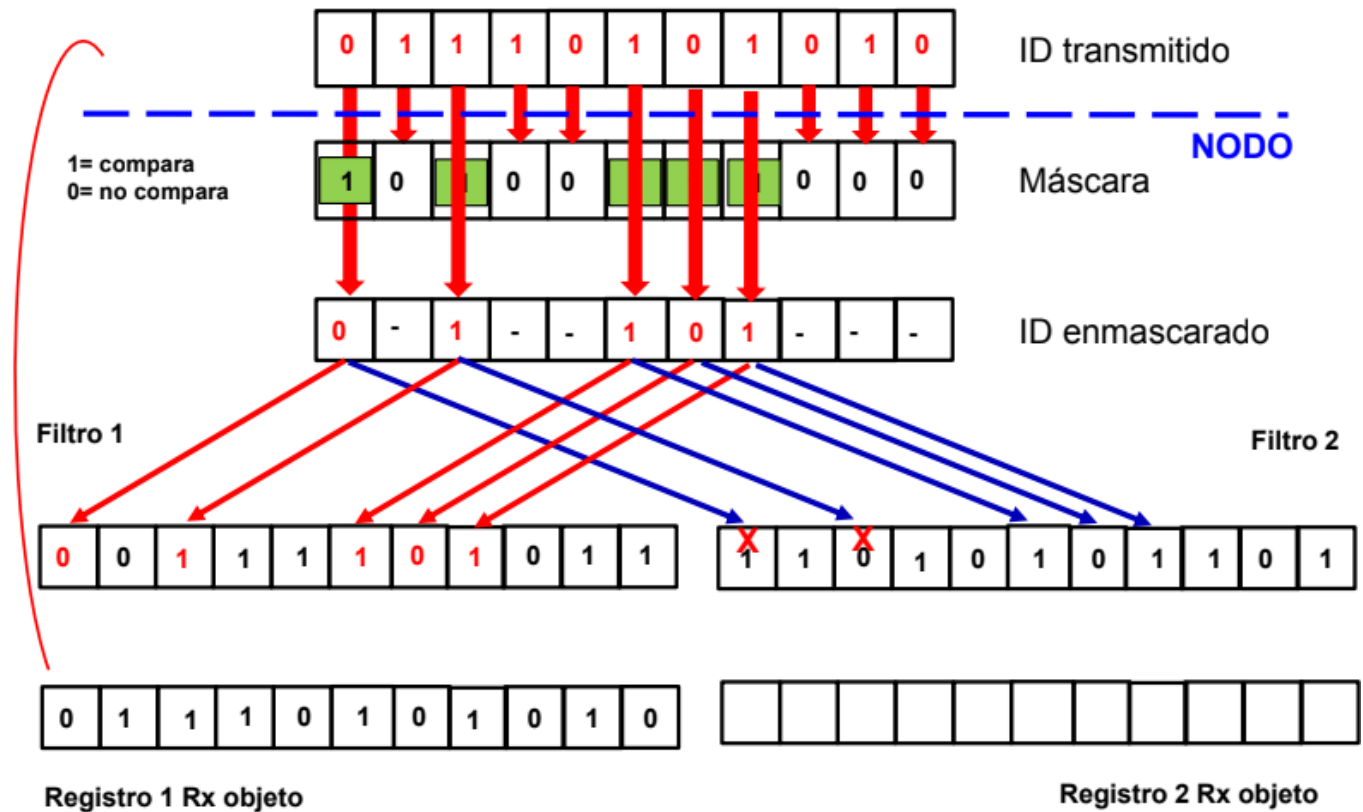
Como se mencionó anteriormente, las tramas llegan hacia todos los nodos, entonces, por medio del identificador se puede señalar la prioridad de cada uno de los mensajes que se están transmitiendo y, a su vez, indicar a que nodo corresponde la información o no.

Recepción de datos

En la comunicación CAN 2.0 no existe un campo de dirección como en otros protocolos, aquí todas las tramas se transmiten a todos los nodos, y cada nodo tiene la necesidad de seleccionar los mensajes que sean de su interés para recibir a través del identificador de trama.

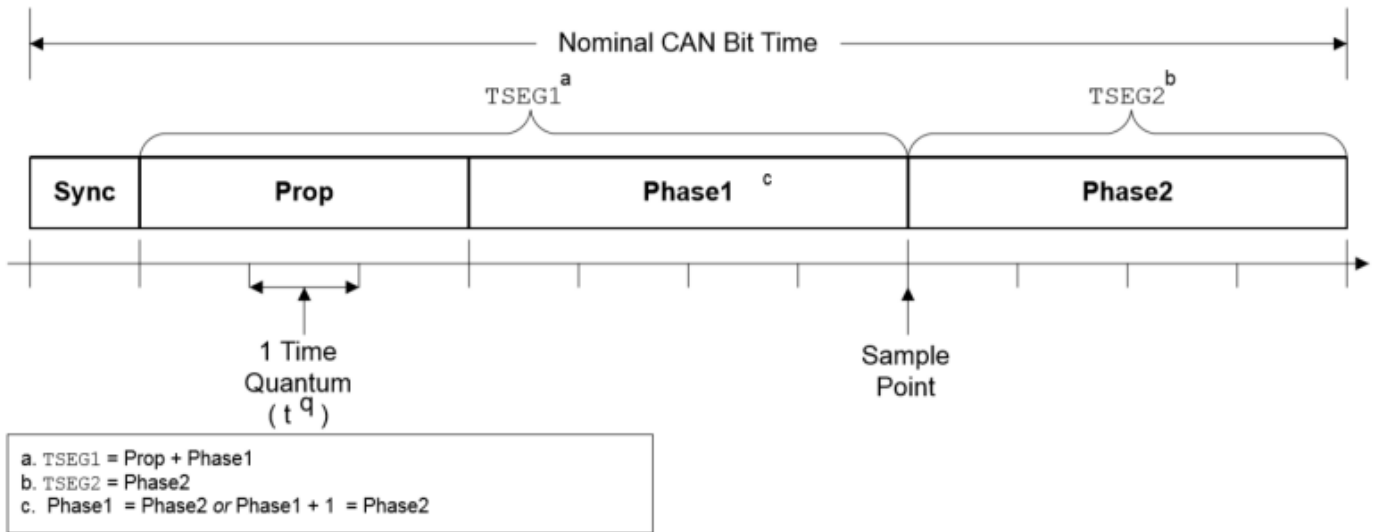
El identificador de la trama, define la prioridad de la trama y la información sobre el contenido del mensaje.

El nodo permite definir filtros de mensajes (generalmente un filtro asociado con cada registro Rx) y además definen una o más máscaras de recepción para declarar los identificadores de mensajes que les interesan. Una máscara de recepción especifica en qué bits del identificador de mensaje entrante deben operar los filtros para detectar una posible coincidencia.



En la imagen anterior se observa que un ID es recibido a un nodo y se le aplica una máscara, cuyo resultado se comparará con un filtro y, dependiendo de la coincidencia del ID enmascarado con el filtro, éste dejará pasar o no el mensaje, guardándolo en un registro objeto Rx.

El CAN 2.0 de la tiva cuenta con 32 mensajes objeto almacenados en la memoria ram con una tasa de hasta 1Mbit por segundo. Para poder programar el CAN bit en la TIVA se hace lo siguiente:



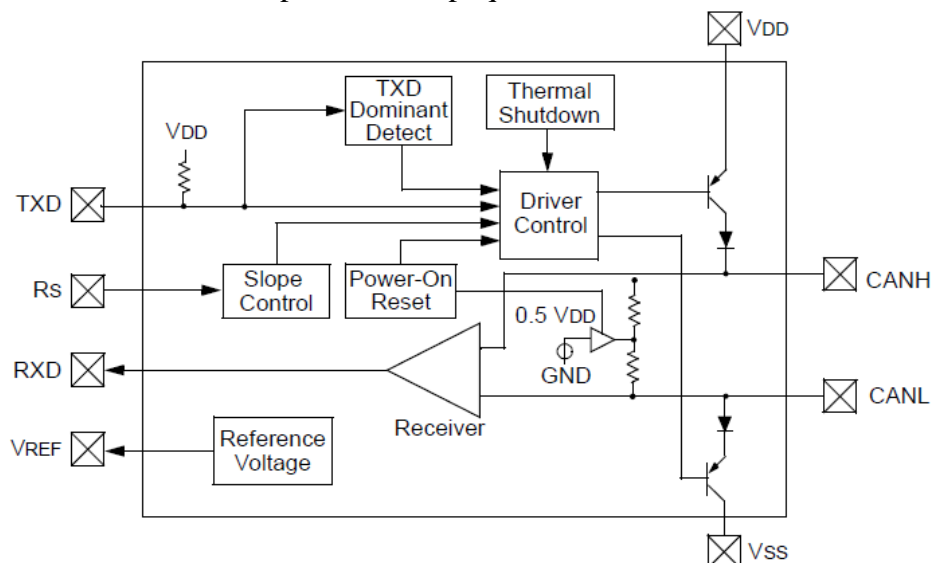
Para los valores en el registro CANBIT, la interpretación real del hardware de este valor es tal que se use uno más que el valor programado aquí.

CANBIT Register Field	Setting
TSEG2	Phase2 - 1
TSEG1	Prop + Phase1 - 1
SJW	SJW - 1
BRP	BRP

Por ende, el tiempo de bit es de 4 a 25 tiempos cuánticos.

Tranceptor MCP2551

Transceptor CAN, también fabricado por Microchip, que hace de interfaz entre el controlador CAN y el bus.



ADC

El convertidor analógico digital se utilizará para la lectura de los dos potenciómetros, esto se realiza para poder leer en un cierto rango de valores una variable externa, en este caso son potenciómetros, pero se pueden utilizar distintos tipos de sensores. Para poder lograr esto se tiene que dar un tiempo de reloj al ADC, se configura la tasa de muestreo y se da prioridad. En este caso utilicé el ADC0 con el secuenciador 2, por lo que se tiene que deshabilitar el secuenciador que se va a utilizar, y se configura el tipo de disparo que se tendrá el secuenciador.

Timers

En este proyecto si tuve que utilizar timers para que la tarjeta pueda recibir datos fuera de la función principal y que no consuma tiempo del main para que el sensado se realice de manera continua, para esta parte también utilizamos interrupciones. Para realizar esto debemos asignar el reloj al timer, y configurar el timer de 32 bits, después se configura el valor de recarga y el modo de flanco.

V. Diseño de detalle

Ahora bien, antes de pasar al código comentado se realizará una especie de pinout dónde se va a especificar que componentes se conectarán en cada uno de los programas, ya que para hacer funcionar este proyecto necesitamos de dos códigos distintos, uno que es el del receptor y otro que es el del transmisor.

Receptor

Componente	Puerto	Bits	Datos
MCP2551	A	0,1	PA0-Receptor y PA1-Transmisor
Leds	K	0,1	PK0-S1 y PK1-S2

Transmisor

Componente	Puerto	Bits	Datos
MCP2551	A	0,1	PA0-Receptor y PA1-Transmisor
Sensor 1	E	2	AIN1-Entrada ADC0S2
Sensor 2	E	3	AIN0-Entrada ADC0S2

Código comentado

Tiva 1: Receptor

```



/*
 * main.c: Parte Receptor
 *
 * Created on: 16 ago. 2021
 * Author: Teresa Fiel
 */

```

```

#include <stdbool.h>
#include <stdint.h>

```


	<p align="center">UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos</p>		<p>Semestre: 2021-2</p>	 <p align="center">INGENIERÍA ELÉCTRICA ELECTRÓNICA</p>
	<p align="center">Prof. Dr. Saúl De La Rosa Nieves</p>	<p align="center">Grupo 1</p>	<p align="center">Página 8 de 18</p>	

```
#include "inc/tm4c1294ncpdt.h"
#include "IEEE_CAN.h"
#include "driverlib/sysctl.h"
```

```
int Sensor_1 = 0;
int Sensor_2 = 0;
```

```
float tp= 0; //Tiempo de propagacion
uint32_t encendido = 0; //bit de encendido, se transmitiran 32 bits
int i=0;
```



```
//-----
//%%%%%%%% INICIALIZACIi:½N DE PUERTOS ASOCIADOS AL CAN0 %%%%%%%%%%%
// CAN0Rx: PA0 CAN0Tx: PA1
//-----
```

```
void Config_Puertos(void){ //(TM4C1294NCPDT)
    SYSCTL_RCGCGPIO_R|=0x1; //Reloj Puerto A
    while((SYSCTL_PRGPIO_R&0x1)==0){}
    GPIO_PORTA_AHB_CR_R=0x3;
    GPIO_PORTA_AHB_AFSEL_R=0x3; //PA0 y PA1 funcii:½n alterna
    GPIO_PORTA_AHB_PCTL_R=0x77; //Funcii:½n CAN a los pines PA0-PA1
    GPIO_PORTA_AHB_DIR_R=0x2; //PA1 Salida Tx y PA0 Entrada Rx
    GPIO_PORTA_AHB_DEN_R=0x3; //Hab funcii:½n digital PA0 y PA1
}
```

```
//-----
//%%%%%%%% INICIALIZACIi:½N CAN0 %%%%%%%%%%%
//-----
```

```
void Config_CAN(void){
    SYSCTL_RCGCCAN_R=0x1; //Reloj modulo 0 CAN
    while((SYSCTL_PRCAN_R&0x1)==0){}
    //Bit Rate= 1 Mbps CAN clock=16 [Mhz]
    CAN0_CTL_R=0x41; //Deshab. modo prueba, Hab. cambios en la config. y hab.
    inicializacion
    CAN0_BIT_R=0x2BC0; //TSEG2=4 TSEG1=9 SJW=0 BRP=0
    //Lenght Bit time=[TSEG2+TSEG1+3]*tq
    //=[(Phase2-1)+(Prop+Phase1-1)+3]*tq
    CAN0_CTL_R&=~0x41; //Hab. cambios en la config. y deshab. inicializacion
    CAN0_CTL_R|=0x2; //Hab de interrupcii:½n en el mi:½dulo CAN
    NVIC_EN1_R|=((1<<(38-32)) & 0xFFFFFFFF); //(TM4C1294NCPDT)
} //El 38 sale de p.116
```

```
//-----
//%%%%%%%% RESOLUCION DE ERRORES %%%%%%%%%%%
//-----
```


	UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos	Semestre: 2021-2	
	Prof. Dr. Saúl De La Rosa Nieves	Grupo 1 Página 9 de 18	

//-----

```
void CAN_Error(void){
    static int ent=0;
    if(CAN0_STS_R&0x80){
        if(ent){
            NVIC_APINT_R|=0x4; //Reinicio de todo el sistema
        }else{
            CAN0_CTL_R=0x41; //Hab. cambios en la config. y hab. inicializacion
            CAN0_CTL_R|=0x80; //Hab. modo prueba
            CAN0_TST_R|=0x4; //Hab. Modo silencio
            CAN0_CTL_R&=~0x41; //Hab. cambios en la config. y deshab.
            inicializacion
            SysCtlDelay(333333);
            CAN0_CTL_R=0x41; //Hab. cambios en la config. y hab. inicializacion
            CAN0_TST_R&=~0x4; //Deshab. Modo silencio
            CAN0_CTL_R&=~0x41; //Hab. cambios en la config. y deshab.
            inicializacion
            ent++;
        }
    }
}
```

//-----

//%%%%%%%%%% HARDWARE DEL MONITOR %%%%%%%%%%%

//-----

//Esta función iniciará todos los leds y botones para poder activar el otro microcnontrolador



```
void Monitor_init (void) {
    SYSCTL_RCGCGPIO_R |= 0x2200; // 1) Habilita reloj para Puerto K,P (P.382)
    while((SYSCTL_PRGPIO_R & 0x2200) == 0){}; // Se espera a que el reloj se
estabilice (p.499)
    //Configuracion de puerto K LEDS Indicadores
    //PIN 2 -----INDICADOR PARA SENSOR 1
    //PIN 3 -----INDICADOR PARA SENSOR 2
    GPIO_PORTK_DIR_R = 0xFF; // PK Salidas
    GPIO_PORTK_DEN_R = 0xFF; // PK // Habilita funcion digital
}
```

//-----

//%%%%%%%%%% INTERRUPCION DEL CAN0 %%%%%%%%%%%

//-----

```
void Inter_CAN0(void){
    uint8_t NoInt;
    uint32_t Rx[3];
```



	<p>UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos</p>		<p>Semestre: 2021-2</p>	 <p>INGENIERÍA ELÉCTRICA ELECTRÓNICA</p>
	<p>Prof. Dr. Saúl De La Rosa Nieves</p>	<p>Grupo 1</p>	<p>Página 10 de 18</p>	

```

NoInt=CAN0_INT_R; //Lectura del apuntador de interrupciones ¿Qué localidad
recibió el dato?
CAN0_STS_R&=~0x10; //Limpieza del bit de recepcion
if(NoInt==0x2)
{ //Recibe datos de los potenciómetros
  Rx[0]=CAN_Rx(NoInt); //Recepción de datos
  Sensor_1 = Rx[0] & 0xFFF; //Se obtienen los primeros 16 bits
  Sensor_2 = (0xFFF & ( Rx[0] >>16)); //se obtienen los siguientes 16
bits
  //Parte del sensor 1
  if (Sensor_1 < 4096/2) //Cuando es menor a 3.3/2 Volts
    GPIO_PORTK_DATA_R |= 0x01; //Enciende LED para el sensor 1 PK0
  else
    GPIO_PORTK_DATA_R = GPIO_PORTK_DATA_R & 0xFE; //APAGA UNICAMENTE
ESE LED
  //Parte del sensor 2
  if (Sensor_2 > 500) //Cuando es menor a 2.5 Volts
    GPIO_PORTK_DATA_R |= 0x02; //Enciende LED para el sensor de 2 PK1
  else
    GPIO_PORTK_DATA_R = GPIO_PORTK_DATA_R & 0xFD; //APAGA UNICAMENTE
ESE LED
}
}
//-----
//%%%%%%%%%%%%%% FUNCION QUE PREPARA LOS 32 MENSAJES OBJETO
%%%%%%%%%%%%%%
//-----
void localidadesCAN(void)
{
  //Localidad 2 Rx con Msk
  CAN_Memoria_Arb(0x333,false,0x2); //Lectura de ambos potenciómetros
  CAN_Memoria_CtrlMsk(0x111,4,false,true,false,0x2);
}

//-----
//%%%%%%%%%%%%%% PROGRAMA PRINCIPAL %%%%%%%%%%%%%%%
//-----
void main(void){
  Monitor_init (); // Inicializacion del hardware que no es CAN
  Config_Puertos();
  Config_CAN();
  localidadesCAN();
  while(1){}
}

```

	<p align="center">UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos</p>	<p>Semestre: 2021-2</p>	 <p align="center">INGENIERÍA ELÉCTRICA ELECTRÓNICA</p>
	<p>Prof. Dr. Saúl De La Rosa Nieves</p>	<p>Página 11 de 18</p>	

Tiva 2: Transmisor

```

/*
 * main.c: Parte Transmisor
 *
 * Created on: 16 ago. 2021
 * Author: Teresa Fiel
 */

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c1294ncpdt.h"
#include "IEEE_CAN.h"
#include "driverlib/sysctl.h"

uint64_t Rx[5];
uint32_t MSJ = 0; //Mensaje de 32 bits a enviar
int i=1;

//Variables Globales

int encendido = 0;
int Sensor_1 = 0;
int Sensor_2 = 0;

float tp= 0; //Tiempo de propagacion
int c = 0;



//-----
//%%%%%%%% INICIALIZACION DE PUERTOS ASOCIADOS AL CAN0 %%%%%%%%%%
// CAN0Rx: PA0 CAN0Tx: PA1
//-----

void Config_Puertos(void){ //(TM4C1294NCPDT)
    SYSCTL_RCGCGPIO_R|=0x1; //Reloj Puerto A
    while((SYSCTL_PRGPIO_R&0x1)==0){}
    GPIO_PORTA_AHB_CR_R=0x3;
    GPIO_PORTA_AHB_AFSEL_R=0x3; //PA0 y PA1 funcii¿%n alterna
    GPIO_PORTA_AHB_PCTL_R=0x77; //Funcii¿%n CAN a los pines PA0-PA1
    GPIO_PORTA_AHB_DIR_R=0x2; //PA1 Salida Tx y PA0 Entrada Rx
    GPIO_PORTA_AHB_DEN_R=0x3; //Hab funcii¿%n digital PA0 y PA1
}

//-----
//%%%%%%%%% INICIALIZACI¿%N CAN0 %%%%%%%%%%
//-----

void Config_CAN(void){

```

	UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos		Semestre: 2021-2	 INGENIERÍA ELÉCTRICA ELECTRÓNICA
	Prof. Dr. Saúl De La Rosa Nieves	Grupo 1	Página 12 de 18	

```

SYSCTL_RCGCCAN_R=0x1; //Reloj modulo 0 CAN
while((SYSCTL_PRCAN_R&0x1)==0){}
//Bit Rate= 1 Mbps CAN clock=16 [Mhz]
CAN0_CTL_R=0x41; //Deshab. modo prueba, Hab. cambios en la config. y hab.
inicializacion
CAN0_BIT_R=0x2BC0; //TSEG2=4 TSEG1=9 SJW=0 BRP=0
//Lenght Bit time=[TSEG2+TSEG1+3]*tq
//=[(Phase2-1)+(Prop+Phase1-1)+3]*tq
CAN0_CTL_R&=~0x41; //Hab. cambios en la config. y deshab. inicializacion
CAN0_CTL_R|=0x2; //Hab de interrupci n en el m dulo CAN
NVIC_EN1_R|=((1<<(38-32)) & 0xFFFFFFFF); //(TM4C1294NCPDT)
} //El 38 sale de p.116

//-----
//%%%%%%%%%% RESOLUCI N DE ERRORES %%%%%%%%%%%
//-----



void CAN_Error(void){
    static int ent=0;
    if(CAN0_STS_R&0x80){
        if(ent){
            NVIC_APINT_R|=0x4; //Reinicio de todo el sistema
        }else{
            CAN0_CTL_R=0x41; //Hab. cambios en la config. y hab. inicializacion
            CAN0_CTL_R|=0x80; //Hab. modo prueba
            CAN0_TST_R|=0x4; //Hab. Modo silencio
            CAN0_CTL_R&=~0x41; //Hab. cambios en la config. y deshab.
inicializacion
            SysCtlDelay(333333);
            CAN0_CTL_R=0x41; //Hab. cambios en la config. y hab. inicializacion
            CAN0_TST_R&=~0x4; //Deshab. Modo silencio
            CAN0_CTL_R&=~0x41; //Hab. cambios en la config. y deshab.
inicializacion
            ent++;
        }
    }
}

void Sensores_init (void) {
    //Funcion que habilitar  todos los sensores a utilizar.

    SYSCTL_RCGCGPIO_R |= 0x2210; // 1) Habilita reloj para Puerto E,K,P (P.382)
    SYSCTL_RCGCTIMER_R |= 0x8; //RELOJ Y HABILITA TIMER 3 (p.380)
    while((SYSCTL_PRGPIO_R & 0x2210) == 0){}; // Se espera a que el reloj se
estabilice (p.499)

    //CONFIGURACION DEL PUERTO PARA EL ADC.
    //Configuracion del puerto E para potencio metro, sensor luz y sensor agua

```

	<p>UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos</p>		<p>Semestre: 2021-2</p>	
	<p>Prof. Dr. Saúl De La Rosa Nieves</p>	<p>Grupo 1</p>	<p>Página 13 de 18</p>	

```



GPIO_PORTE_AHB_DIR_R = 0x00; // 2) PE3 entrada (analógica)
GPIO_PORTE_AHB_AFSEL_R |= 0x0E; // 3) Habilita Función Alterna de
PE3, PE2, PE1 AIN0, AIN1, AIN2
GPIO_PORTE_AHB_DEN_R = 0x00; // 4) Deshabilita Función Digital de PE3, PE2,
PE1, AIN2
GPIO_PORTE_AHB_AMSEL_R |= 0x0E; // 5) Habilita Función Analógica de
PE3, PE2, PE1, AIN2

//Configuración de puerto K LEDs Indicadores de encendido o apagado

GPIO_PORTK_DIR_R = 0xFF; // PK Salidas
GPIO_PORTK_DEN_R = 0xFF; // PK // Habilita función digital
//Configuración del ADC

SYSCTL_RCGADC_R |= 0x01; // 6) Habilita reloj para ADC0(p. 396)
while((SYSCTL_PRADC_R & 0x01) == 0); // Se espera a que el reloj se
estabilice
//MAPA DE REGISTROS EN 1073
ADC0_PC_R = 0x07; // 7) (p.1159) Maxima tasa de muestreo 1M muestra/seg
ADC0_SS PRI_R = 0x1023; // 8) SS2 con la más alta prioridad
ADC0_ACTSS_R = 0x000; // 9) Deshabilita SS2 antes de cambiar configuración
de registros (p. 1076)
ADC0_EMUX_R = 0x0500; // 10) Se configura SS2 para disparar muestreo por
timer (p.1091) Interrupcion
ADC0_SAC_R = 0x0; // 11) Se configura para no tener sobremuestreo por
hardware(default)(p. 1105)
ADC0_CTL_R = 0x0; //12) Se configura con referencias internas (default VDDA
and GNDA) (p. 1107)
ADC0_SSOP2_R = 0x0000; // 13) Se configura para salvar los resultados del
ADC en FIFO (default)(p. 1134)
ADC0_SSEMUX2_R = 0; // 16) Canales del SS2 para 1º y segunda muestra en
AIN(15:0) (p.1125)
ADC0_SSMUX2_R = 0x0210; // 15) Se configura entradas 1ºmuestra=AIN 0,
2ºmuestra=AIN 1(p.1109), 3 muestra AIN2
ADC0_SSTSH2_R = 0x0000; // 14) Se configura el ADC para un periodo de 4
para tmp S&H en el secuenciador 2 (default) (p. 1134)
//Esto se hace para que sea estable el sensor
ADC0_SSCTL2_R = 0x0600; // 17) Final con tercera muestra Si: Sensor tmp,no:
AIN, Habilita interrupcion; No:muestra diferencial (p.1111)
ADC0_IM_R = 0x0004; // 18) habilita interrupción SS2 (p. 1081) //ACTIVA
INTERRUPCION CUANDO
//TERMINE DE CONVERTIR TERMINANDO EL TIMER
NVIC_EN0_R |= 1<<(16-0); //HABILITA LA INTERRUPCION 16 (ADC_SS2)
ADC0_ACTSS_R |= 0x0004; // 19) Habilita SS2 (p. 1076)
//No se necesita establecer una prioridad en las interrupciones porque no
son continuas

```

	<p align="center">UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos</p>		Semestre: 2021-2	 <p align="center">INGENIERÍA ELECTRICA ELECTRONICA</p>
	<p align="center">Prof. Dr. Saúl De La Rosa Nieves</p>	<p align="center">Grupo 1</p>	Página 14 de 18	

```



// SINCRONIZACIÓN DEL PLL PARA UTILIZAR PIOSC
SYSCTL_PLLFREQ0_R |= SYSCTL_PLLFREQ0_PLLPWR; // encender PLL
while((SYSCTL_PLLSTAT_R&0x01)==0); // espera a que el PLL fije su
frecuencia
SYSCTL_PLLFREQ0_R &= ~SYSCTL_PLLFREQ0_PLLPWR; // apagar PLL
ADC0_ISC_R = 0x0004; // Se recomienda Limpiar la bandera RIS del SS2

// -----
// CONFIGURACION DEL TIMER DEL ADC
// -----
//
TIMER3_CTL_R=0X0000000; //DESHABILITA TIMER 3 PARA CONFIGURAR (p.986)
TIMER3_CFG_R= 0X00000000; //CONFIGURA TIMER DE 32 BITS (p. 976)
//TIMER3_TAMR_R= 0X00000002; //CONFIGURAR PARA MODO PERIODICO CUENTA HACIA
ABAJO (p. 977)
TIMER3_TAMR_R= 0X00000012; //CONFIGURAR PARA MODO PERIODICO CUENTA HACIA
ARRIBA (p. 977)
TIMER3_TAILR_R= 0XFFFFFF; // VALOR DE RECARGA (p.1004) 20ms //Cada 20ms hace
la interrupcion aprox
//TIMER3_TAILR_R= 0X0004E200; // VALOR DE RECARGA (p.1004)
TIMER3_TAPR_R= 0X00; // PRESCALADOR DE TIMER A, SOLO PARA MODOS DE 16 BITS
(p.1008)
TIMER3_ADCEV_R = 0X01; // HABILITA MODO CAPTURA DEL TIMER 3 COMO EVENTO DE
DISPARO PARA EL ADC
TIMER3_ICR_R= 0X00000001 ; //LIMPIA POSIBLE BANDERA PENDIENTE DE TIMER3
(p.1002)
//TIMER3_IMR_R |= 0X00000001; //ACTIVA INTRRRUPCION DE TIMEOUT (p.993)
TIMER3_CTL_R |= 0X00000021; //HABILITA TIMER 3 Y ACTIVA TRIGGER PARA EL ADC
(p.986)
}

//-----
//%%%%%%%%%%%%%% INTERRUPCION DEL CAN0 %%%%%%%%%%%%%%%
//-----

void Inter_CAN0(void){
uint8_t NoInt;
NoInt=CAN0_INT_R; //Lectura del apuntador de interrupciones
CAN0_STS_R&=~0x10; //Limpieza del bit de recepcion
i++;
if(NoInt==0x1){
Rx[0]=CAN_Rx(NoInt); //Recepción de datos
encendido = Rx[0]; //Recibe si se enciende o no el coche
}
if(NoInt==0x4){//Solicitud de trama remota
Rx[1]=CAN_Rx(NoInt); //Recepción de datos
}
}

```

	<p align="center">UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO FACULTAD DE INGENIERÍA Sistemas Embebidos</p>		<p>Semestre: 2021-2</p>	 <p>INGENIERÍA ELÉCTRICA ELECTRÓNICA</p>
	<p>Prof. Dr. Saúl De La Rosa Nieves</p>	<p>Grupo 1</p>	<p>Página 15 de 18</p>	

```

}

void ADC0_SS2IntHandler(void){
    //LA INTERRUPCION SE DISPARA CADA 20ms CUANDO TERMINA DE HACER LA
    CONVERSIÓN
    //LIMPIA BANDERA
    MSJ = 0;
    if (encendido == 1)
    {
        Sensor_1 = (ADC0_SS2FIFO2_R&0xFFF); //RESULTADO DE LA PRIMERA MUESTRA
        Sensor_2 = (ADC0_SS2FIFO2_R&0xFFF); //RESULTADO DE LA SEGUNDA MUESTRA

        //Mensaje a enviar de ambos sensores
        MSJ =(0x00000FFF & Sensor_1); //Byte 0-1
        MSJ|=(0xFFFF0000 & ( Sensor_2<<16)); //Byte 2-3
        CAN_Memoria_Dato(MSJ,0x2); //TX SENSORES
        CAN_Tx(0x2);//Transmite datos sensores
        SysCtlDelay(213); //Funcion de retardo: SysCtlDelay(1) = 187.5 ns, 100us
    }
    ADC0_ISC_R = 0x0004; //Limpia la bandera RIS del ADC0
} //Fin de interrupcion

//-----
//%%%%%%%%%%%%%% FUNCION QUE PREPARA LOS 32 MENSAJES OBJETO
//%%%%%%%%%%%%%%
//-----

void localidadesCAN(void){
    //Localidad 2 Tx, Transmisor
    CAN_Memoria_Arb(0x333,true,0x2); //ID:0X2222 True, mandar-- a localidad 0x2
    CAN_Memoria_CtrlMsk(0,4,false,false,false,0x2);
}

//-----
//%%%%%%%%%%%%%% PROGRAMA PRINCIPAL %%%%%%%%%%%%%%%
//-----

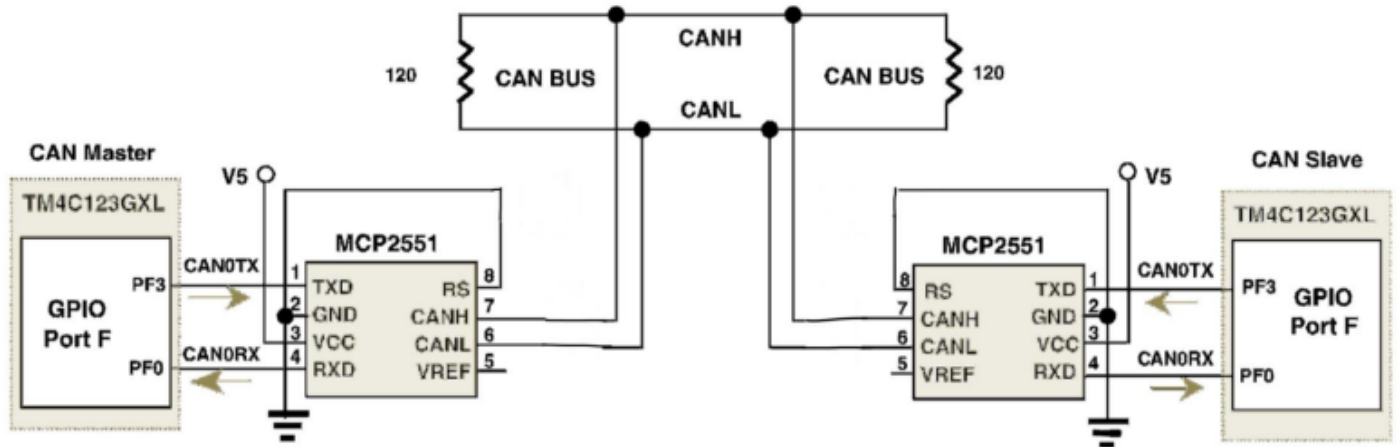
void main(void){
    Sensores_init(); //Funcion que inicializa ADC0 para el sensor de
    temperatura y el sensor de luz.
    Config_Puertos();
    Config_CAN();
    localidadesCAN(); //Se preparan los mensajes objeto
    encendido = 1;
    while(1)
    {}
}

```


}

VI. Construcción

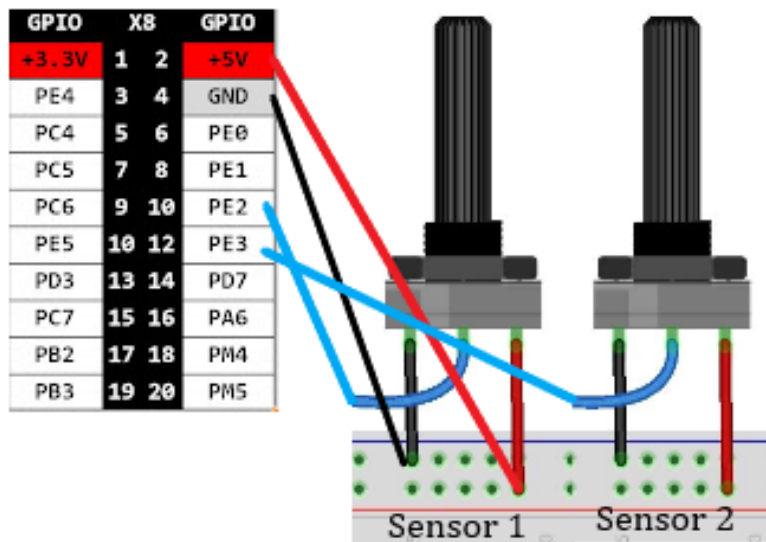
Diagrama de conexiones



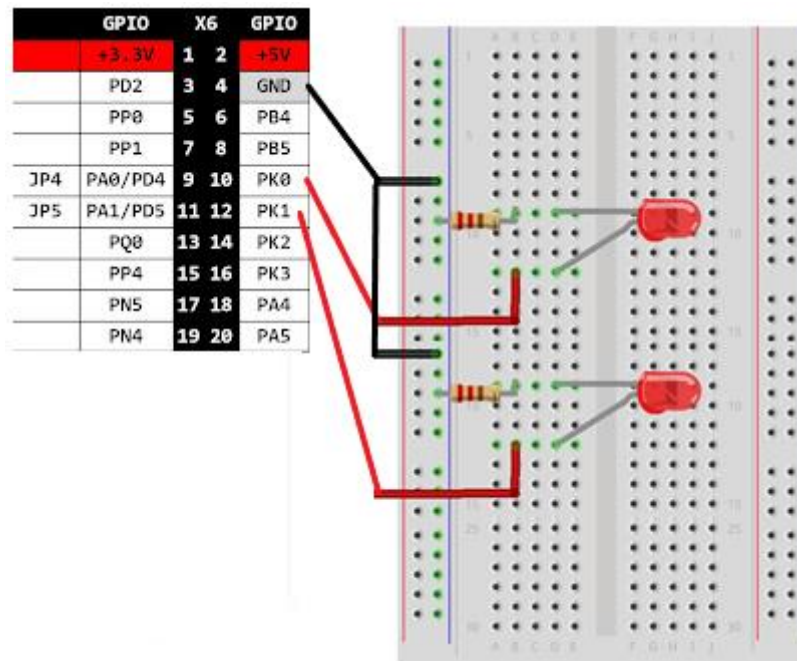
Tx PA1

Rx PA0

Transmisor



Receptor



VII. Resultados y – Conclusiones

En conclusión, este proyecto fue de utilidad para concretar los conocimientos de CAN, ya que se realizó una conexión entre ambas tarjetas pero a su vez se pudo utilizar con una finalidad, esto se realizó haciendo funcionar dos potenciómetros con un convertidor analógico que se incluye en el transmisor, y se manda por la comunicación a la otra tarjeta, la cuál establece un límite alto para un sensor y otro bajo para el restante, esto nos da una posibilidad de realizar aplicaciones a futuro con sensores, no únicamente analógicos sino que se puede implementar cualquier tipo de sensor para poder monitorear a distancia ciertos fenómenos.

Alambrado final

