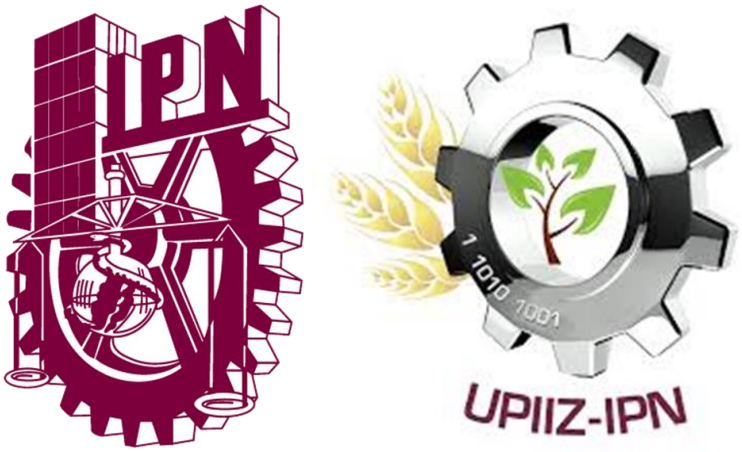
INSTITUTO POLITÉCNICO NACIONAL - UPIIZ



Practica 07

Comunicación UART

Bernardo Martínez Medina

Adrián Torres Aranda

Fecha de Entrega: 14/06/2022

Programa Académico: Implementación de sistemas digitales

Docente: Ramon Jaramillo Martínez

8to Semestre

ING MECATRÓNICA

- # Resumen.

Analizar, diseñar e implementar una interfaz de entrada/salida, para resolver un problema específico por medio del módulo UART.

- # Introducción.

La comunicación UART puede transmitir o recibir datos bit a bit, es por eso que recibe el nombre de comunicación serial, entre dos microcontroladores o también hacia un computador. La conexión a la PC es bajo el estándar de comunicación RS232. A continuación, se describen las características de este módulo UART.

- Operación Síncrona y Asíncrona.
- Operación Full Dúplex en modo Asíncrono.
- Operación Half Dúplex en operación Síncrona Maestro-Esclavo.
- Operación de 8 o 9 bits.
- Generador de Baud-Rate de Alta y Baja
- Calibración Automática de Velocidad de Baudios.
- Detección Automática de Recepción de datos.
- Detección de errores de OverRun y Frame (datos sobrescritos y dato invalido respectivamente).
- 2 interrupciones independientes; TX completado, RX completado [1].

Pasos para transmisión y recepción de datos a través del UART:

1. Configurar las líneas pin RX y pin TX como entrada y salida respectivamente, a través del registro TRISC.
2. Configurar el modo Síncrono o Asíncrono.
3. Configurar el Baud-Rate del USART a través del registro UBRRH y
4. Configurar Tamaño de byte 8/9 bits.
5. Habilitar la Transmisión y Recepción a través de los bits TXEN y RCEN respectivamente.
6. Habilitar la Interrupción de Transmisión/Recepción a través de los bits TXIE y RCIE respectivamente (opcional).

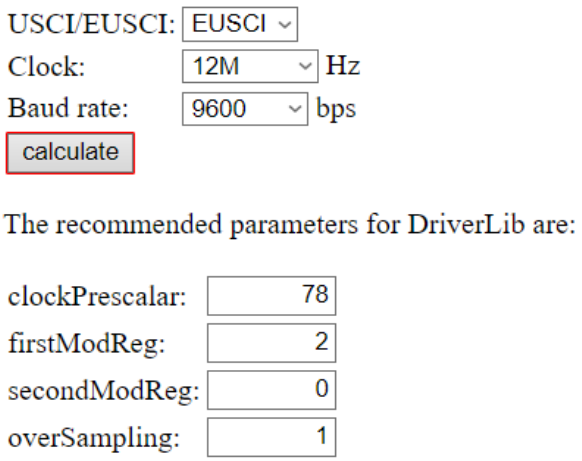
- # Desarrollo

1. ## Recepción de Información por puerto UART a 9600 Baudios.

En la primera parte del desarrollo de la práctica se tiene como objetivo crear un código donde esté involucrado el puerto UART para lograr enviar datos a una velocidad de 9600 baudios. Para lograr esto se utilizan los siguientes puertos.

- UART
- ADC

Pero antes de asignar valores primero se tuvo que obtener los valores para configurar la ratio de baudios en el puerto UART utilizando el EUSCI. Y con los valores comunes como un reloj de 12 Mhz ya que es la velocidad a la que trabaja el puerto UART se obtuvieron los valores del prescalar del reloj y del UCxBRF como se muestran en la siguiente imagen.



Con eso se elaboró el siguiente código para la tarjeta de desarrollo:

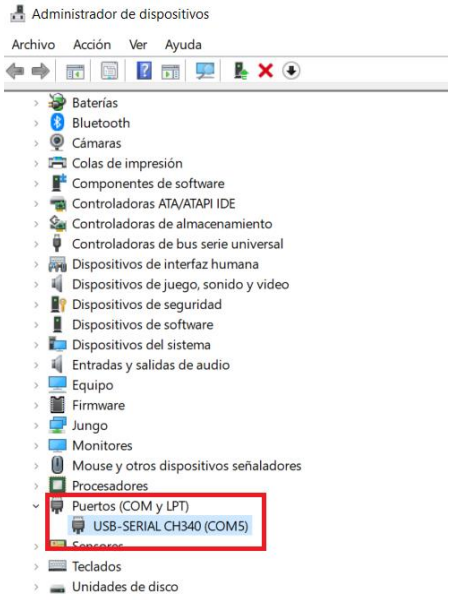
```
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>

/*Variables globales*/
uint8_t i = 0, j=0, k=0;
char msg[30];

static volatile uint16_t ValorADC;
static volatile float VoltajeNormalizado;
static volatile float y;
static volatile float y_n;
static volatile float alpha = 0.5;

static volatile uint16_t cont=1;
static volatile float aux=0;
static volatile float SMA=0;


/* Página que permite calcular los valores del bloque EUSCI
 *http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html
 */
const eUSCI_UART_ConfigV1 uartConfig =
{
        EUSCI_A_UART_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source
        78,                                     // BRDIV = 78
        2,                                       // UCxBRF = 2
        0,                                       // UCxBRS = 0
        EUSCI_A_UART_NO_PARITY,                  // No Parity
        EUSCI_A_UART_LSB_FIRST,                  // LSB First
        EUSCI_A_UART_ONE_STOP_BIT,               // One stop bit
        EUSCI_A_UART_MODE,                       // UART mode
        EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION,  // Oversampling
        EUSCI_A_UART_8_BIT_LEN                  // 8 bit data length
```

```
};


int main(void)
{
    /* Halting WDT  */
    MAP_WDT_A_holdTimer();

    ValorADC=0;
    VoltajeNormalizado=0;

    /*Configura Flash estado de espera*/
    FlashCtl_setWaitState(FLASH_BANK0,1);
    FlashCtl_setWaitState(FLASH_BANK1,1);

    /*Habilitamos FPU*/
    FPU_enableModule();
    FPU_enableLazyStacking();

    /*Configuración ADC*/
    ADC14_enableModule();
    ADC14_initModule(ADC_CLOCKSOURCE_SMCLK, ADC_PREDIVIDER_32, ADC_DIVIDER_8, ADC_NOROUTE);

    /*GPIO Configuracion*/
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5, GPIO_PIN5,
GPIO_TERTIARY_MODULE_FUNCTION);

    /*Configurar ADC Memoria*/
    ADC14_configureSingleSampleMode(ADC_MEM0, true);
    ADC14_configureConversionMemory(ADC_MEM0, ADC_VREFPOS_AVCC_VREFNEG_VSS, ADC_INPUT_A0,
ADC_NONDIFFERENTIAL_INPUTS);

    /*Configuración timer en modo manual*/
    ADC14_enableSampleTimer(ADC_MANUAL_ITERATION);
    ADC14_enableConversion();
    ADC14_toggleConversionTrigger();



    /* P1.2 and P1.3 en modo UART  */
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
            GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

    /* DCO to 12MHz */
    PCM_setPowerState(PCM_AM_LDO_VCORE1);
    CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_12);

    /* Configuración UART con base a la estructura de arriba */
    UART_initModule(EUSCI_A0_BASE, &uartConfig);

    /* Habilitamos UART */
    UART_enableModule(EUSCI_A0_BASE);

    /* Interrupciones */
    UART_enableInterrupt(EUSCI_A0_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
    Interrupt_enableInterrupt(INT_EUSCIA0);

    ADC14_enableInterrupt(ADC_INT0);
    Interrupt_enableInterrupt(INT_ADC14);

    Interrupt_enableMaster();



    while(1)
    {
        __delay_cycles(6000000);                        // delay
        ADC14->CTL0 |= ADC14_CTL0_ENC | ADC14_CTL0_SC; //habilitar conversion
```

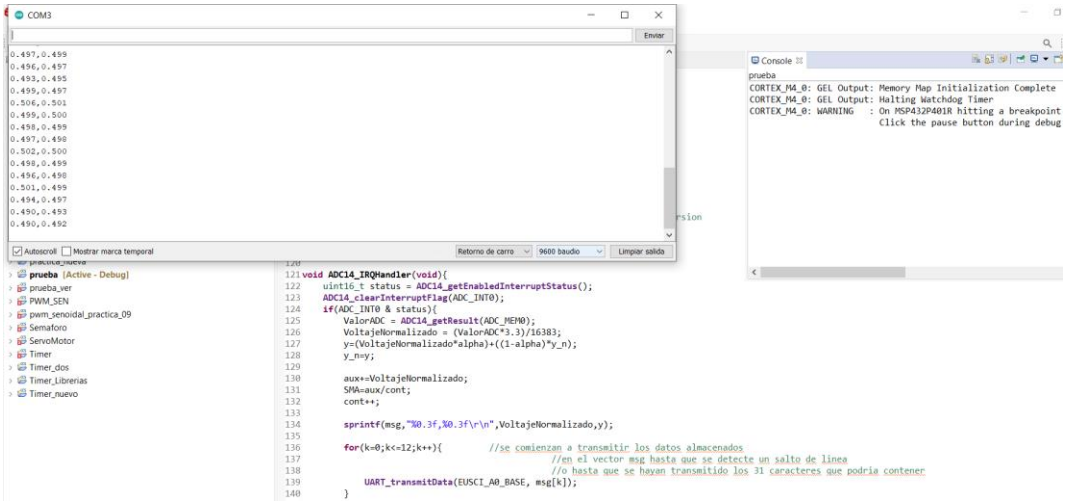
```
    }
}


void ADC14_IRQHandler(void){
    uint16_t status = ADC14_getEnabledInterruptStatus();
    ADC14_clearInterruptFlag(ADC_INT0);
    if(ADC_INT0 & status){
        ValorADC = ADC14_getResult(ADC_MEM0);
        VoltajeNormalizado = (ValorADC*3.3)/16383;
        y=(VoltajeNormalizado*alpha)+((1-alpha)*y_n);
        y_n=y;

        aux+=VoltajeNormalizado;
        SMA=aux/cont;
        cont++;

        sprintf(msg,"%0.3f,%0.3f\r\n",VoltajeNormalizado,y);

        for(k=0;k<=12;k++){         //se comienzan a transmitir los datos almacenados
                                    //en el vector msg hasta que se detecte un salto de
linea
                                    //o hasta que se hayan transmitido los 31 caracteres
que podria contener
            UART_transmitData(EUSCI_A0_BASE, msg[k]);
        }



    }
}
```

Con el código listo la forma de implementarlo cambia un poco a la rutinaria de cualquier práctica, esto principalmente a que se está haciendo uso del puerto EUSCI ya que al este soportar múltiples modos de comunicación serial se puede hacer uso del “serial plotter” incorporado en la aplicación de Arduino.

Por lo tanto, la forma de conexión para probar el código es solo mediante el cable de enlace del mps432A ver que puerto COM se activa y elegir ese desde las configuraciones de Arduino.

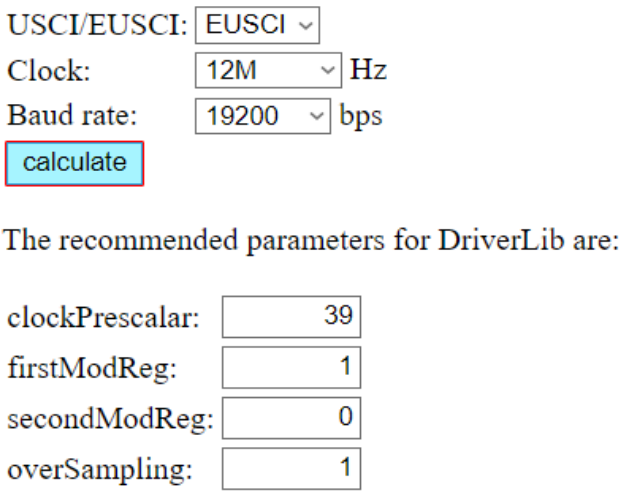


En la siguiente imagen podemos ver como al seleccionar una velocidad de 9600 baudios en el serial plotter de Arduino, el mensaje de “VoltajeNormalizado” de nuestro código se imprime en la ventana de este, donde se hace un salto de línea después de escribir 12 caracteres como se especificó.



2. Envió de datos a una velocidad mayor sin pérdida de información.

Por otro lado, avanzando con la segunda parte de la práctica se optó principalmente por elegir una velocidad de 19200 baudios donde análogamente a la parte anterior primero se calcularon los valores a poner en el prescalar del reloj, el cual cambia de 78 a 39, al igual que sus otros valores.

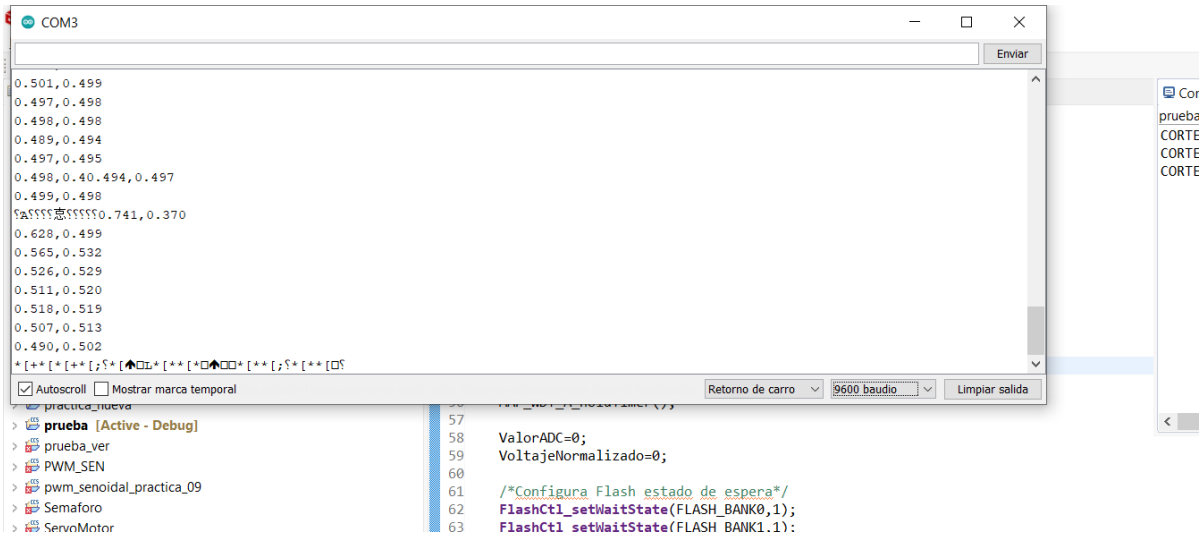


Por lo tanto, la parte del código que será modificada solo es esta sección:

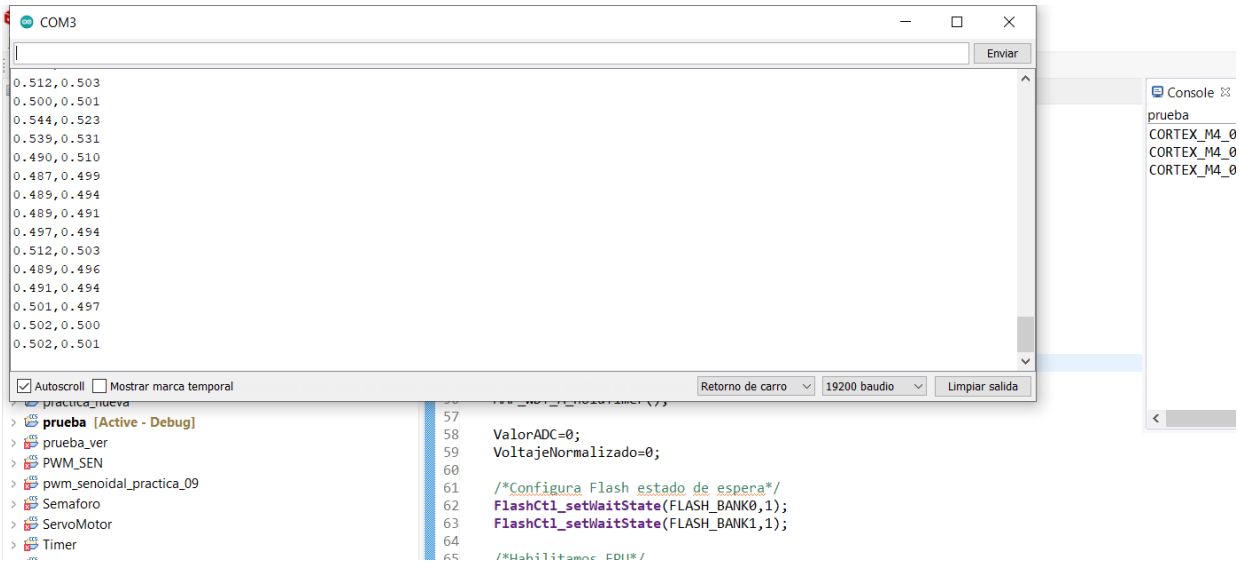
```
const eUSCI_UART_ConfigV1 uartConfig =
{
EUSCI_A_UART_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source
39,                                     // BRDIV = 78
1,                                       // UCxBRF = 2
0,                                       // UCxBRS = 0
EUSCI_A_UART_NO_PARITY,                  // No Parity
EUSCI_A_UART_LSB_FIRST,                  // LSB First
EUSCI_A_UART_ONE_STOP_BIT,               // One stop bit
EUSCI_A_UART_MODE,                       // UART mode
EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION,  // Oversampling
EUSCI_A_UART_8_BIT_LEN                  // 8 bit data length
};
```

Entonces para probar que el programa funcione correctamente se hicieron varias pruebas por medio del serial plotter.

Donde al poner el serial plotter con la velocidad del punto pasado de 9600 baudios, el mensaje junto al voltaje normalizado que se tiene que enviar tiende a fallar, produciéndose una pérdida de información.



Mientras que al cambiar la velocidad del serial plotter a 19200, el mensaje se vuelve a enviar con normalidad sin ninguna perdida de información como se puede observar en la imagen siguiente.



- # Análisis de Resultados

Al haber realizado esta práctica se obtiene una configuración nueva para la tarjeta mps432 en cuestión del tipo de comunicación que se puede generar o usar para la tarjeta. Como se sabe el tipo de comunicación UART es un protocolo configurable donde la velocidad de los bauds es una configuración importante donde se estén realizando él envió de datos. Ya que si estos son

diferentes existe una pérdida de información. Toda esta información calculada a partir de las configuraciones estándar del UART.

Partiendo a otras partes de la practica también se pueden destacar configuraciones hechas como la configuración de un reloj compartido y los bits de parada para indicar donde es que inician los datos y donde terminan.

- # Conclusiones

Adrián Torres Aranda

Esta práctica fue más sencilla terminar esta práctica además de lo que pedía era algo fácil y ya se sabía generar una velocidad mayor y sin pérdida de información, también él envió de datos. Ya que estos fue lo que se vio en clase además de tener el código y poder entender la transmisión y recepción de datos.

Bernardo Martínez Medina

Al haber implementado la practica hemos conocido no solo las diferentes configuraciones que tiene el mps432 con el UART si no las características que esta comunicación tiene en el mundo de la industria. Como por ejemplo que solo requiere dos cables, gracias a su forma de envió de bits, y hablando de bits como es que el bit de paridad evita errores en la comunicación sin la necesidad de tener una señal de reloj. Aunque por otro lado en la misma practica se vieron detalles como que no se permite a comunicación entre maestro y esclavo simultáneamente y que al usarse en largas distancias podrian resultar mejor pensar en otros tipos de comunicación ya que al usar el mismo buffer solo se puede enviar y transmitir individualmente.

- # Referencias.

[1] «Maker Electrónico,» 24 marzo 2017. [En línea]. Available: https://www.makerelectronico.com/comunicacion-serial-uart-usart-rs232-pic18f4550/. [Último acceso: 18 junio 2022].