Práctica 11
Serial y Teclado Matricial

Islas Bravo, Andrés.
Estudiante - Laboratorio de Microcontroladores.
Departamento de Arquitectura e Ingeniería
Instituto Tecnológico y de Estudios Superiores de Monterrey
Ciudad de México, México
a01339391@itesm.mx

Valverde López, Juan Pablo.
Estudiante - Laboratorio de Microcontroladores.
Departamento de Arquitectura e Ingeniería
Instituto Tecnológico y de Estudios Superiores de Monterrey
Ciudad de México, México
a01656127@itesm.mx

Zhu Chen, Alfredo.
Estudiante - Laboratorio de Microcontroladores.
Departamento de Arquitectura e Ingeniería
Instituto Tecnológico y de Estudios Superiores de Monterrey
Ciudad de México, México
a01651980@itesm.mx

Abstract
This practice is mainly focused in the appliance of serial communication, interruptions and polling. It's important the correct structure in this code in order to involve all the tasks that are supposed to be done. As its correct implementation of polling instructions presented in the main code, and then appliance of interruptions in order to communicate using USART.

I. Introducción

En el presente documento se abordarán todos los aspectos necesarios para la correcta comunicación entre en atmega16 y periférico perteneciente a proteus. Dicha comunicación se realizará mediante protocolo USART, para el cual se tendrán que tomar diversas decisiones cómo el revisar la paridad o la tasa de baudios que se utilizará para la comunicación. Así mismo la comunicación se puede realizar en diferentes longitudes de mensaje, así que eso también tendrá que ser acordado. Por otra parte se recibe números en binario mediante la opresión de una tecla matricial, pero dicho valor tendrá que ser transformado a código ASCII para poder ser transmitido y leído por la terminal de proteus y por la LCD.

II. Marco Teórico

El teclado matricial es un elemento muy útil para ofrecer información por parte del usuario de acuerdo con sus necesidades. En la primera práctica se habló del funcionamiento del circuito integrado MM74C922 para poder obtener el valor de la posición del teclado matricial 4x4. De esta forma, no se hablará a detalle el uso de este componente, sino que nos enfocaremos principalmente en la explicación de la comunicación serial.

La información se puede transmitir y recibir de dos formas, por medio de comunicación serial y de comunicación paralela. La primera tiene la ventaja de consumir menor energía y menor espacio, pero eso conlleva a costo de perder velocidad. Debido a la tendencia de la miniaturización de la electrónica, se ha utilizado ampliamente la comunicación serial para la mandar información entre dispositivos. Así mismo, se requieren pocas líneas de los puertos para la transmisión y recepción de datos, esto permite aprovechar las líneas para otros usos que se requieran y hacer que el sistema sea más compacto.

ATMega 16 cuenta con 3 protocolos de comunicación serial: USART(*Universal Synchronous and Asynchronous Receiver-Transmitter*), TWI(*Two Wire Interface*) y SPI(*Serial Peripheral Interface*). La diferencia entre la comunicación asíncrona y síncrona en el USART es que el segundo transmite su reloj de operación junto con la línea de transmisión o recepción, en cambio el primero no. El formato de USART que ATMega 16 puede manejar es la siguiente:

1. 1 bit de inicio
2. 5,6,7,8 o 9 bits de datos
3. sin paridad, paridad par o paridad impar
4. 1 o 2 bits de parada

Captura 1:Formato de la trama de USART. Tomado desde: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

El uso de la paridad es para checar si hubo algún error durante la transmisión de datos al checar la paridad junto con los bits de paridad. Esto se puede habilitar o deshabilitar de acuerdo con una palabra de control que se verá posteriormente. Para calcular el bit de paridad, se puede realizar de la siguiente manera, donde *d* es el bit n del dato.

$$P_{par} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0 \quad (1)$$

$$P_{impar} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1 \quad (2)$$

Por un lado, para transmitir datos, se guardan en un buffer de transmisión y se realiza mediante la instrucción *OUT UDR,R16*. Por otro lado, la recepción de datos se realiza mediante la lectura en el buffer de recepción con la instrucción *IN UDR,R16*. Las palabras de control del USART se describen en las siguientes tres tablas. Principalmente se configuran los primeros dos para indicar el modo de trabajo y el último contiene todas las banderas que se pueden leer.

TABLA I

CONFIGURACIÓN DE UCSRC(*USART CONTROL AND STATUS REGISTER C*)

URSEL|UMSEL|UPM1|UPM2|USBS|UCSZ1|UCSZ0|UCPOL		
URSEL(*Register Select*)	0	seleccionar registro UBRRH0
	1	seleccionar registro UCSRC
UMSEL(*Mode Select*)	0	Modo síncrono
	1	Modo síncrono
UPM1|UPM0*(Parity Mode*)	0	deshabilitado
	00	reservado
	01	paridad par
	10	paridad impar
USBS(*Stop Bit Select*)	0	1-bit de parada
	1	2-bits de parada
UCSZ1|UCSZ0(*Character Size*) tamaño del dato interesado a enviar	00	5-bits de dato
	01	6-bits de dato
	10	7-bits de dato
	11 con UCSZ2	8-bits de dato
	11 con UCSZ2 en 1	9-bits de dato
UCPOL(*Clock Polarity*)	0	Detección por flanco de bajada para modo síncrono
	1	Detección por flanco de subida para modo síncrono

TABLA II
CONFIGURACIÓN DE UCSRB(*USART CONTROL AND STATUS REGISTER B*)

RXCIE|TXCIE|UDRIE|RXEN|TXEN|UCSZ2|RXB8|TEXB8		
RXCIE(RX *Complete Interrupt Enable*) Interrupción cuando se haya completado la recepción	0	deshabilitado
	1	habilitado
TXCIE(TX *Complete Interrupt Enable*) Interrupción cuando se haya completado la transmisión	0	deshabilitado
	1	habilitado
RXCIE(USART Data Register Empty *Interrupt Enable*) Interrupción cuando se haya vaciado el buffer de transmisión	0	deshabilitado
	1	habilitado
RXEN(Receiver *Enable*)	0	deshabilitado
	1	habilitado
TXEN(*Transmitter Enable*)	0	deshabilitado
	1	habilitado
UZSZ2(*Character Size*) tamaño del dato interesado a enviar	0	trabajar con 8-bits de dato si UCSZ1|UCSZ2 están en 11
	1	trabajar con 9-bits de dato si UCSZ1|UCSZ2 están en 11
RXB8	bit 8 si se quiere trabajar con 9-bits de dato en la recepción	
TXB8	bit 8 si se quiere trabajar con 9-bits de dato en la transmisión	

TABLA III
UCSRA(*USART CONTROL AND STATUS REGISTER A*)

RXC|TXC|UDRE|FE|DOR|PE|U2X|MPCM		
RXC(*Receiver Complet*e)	Bandera en alto para indicar que se completó la recepción(Buffer de recepción lleno)	
TXC(*Transmitter Complete*)	Bandera en alto para indicar que se completó la transmisión	
UDRE(*USART Data Register Empty*)	Bandera en alto para indicar que el buffer de transmisión está vacío y está listo para nueva transmisión	
FE(*Frame Error*)	Bandera en alto para indicar que hubo error al recibir un bit de parada de 0 en lugar de 1	
DOR(*Data OverRun*)	Bandera en alto para indicar que llegó un nuevo dato y no se ha leído el anterior	
PE(*Parity Error*)	Bandera en alto para indicar que hubo error de paridad	
U2X(Duplicar velocidad de transmisión)	0	deshabilitado
	1	habilitado
MPCM(*Multi-Procesor Communication Mode*)	0	deshabilitar modo comunicación entre varios MCU
	1	habilitar modo comunicación entre varios MCU

TABLA III
UCSRA(*USART CONTROL AND STATUS REGISTER A*)

RXC|TXC|UDRE|FE|DOR|PE|U2X|MPCM	
RXC(*Receiver Complet*e)	Bandera en alto para indicar que se completó la recepción(Buffer de recepción lleno)
TXC(*Transmitter Complete*)	Bandera en alto para indicar que se completó la transmisión
UDRE(*USART Data Register Empty*)	Bandera en alto para indicar que el buffer de transmisión está vacío y está listo para nueva transmisión
FE(*Frame Error*)	Bandera en alto para indicar que hubo error al recibir un bit de parada de 0 en lugar de 1
DOR(*Data OverRun*)	Bandera en alto para indicar que llegó un nuevo dato y no se ha leído el anterior
PE(*Parity Error*)	Bandera en alto para indicar que hubo error de paridad
U2X(Duplicar velocidad de transmisión)	0 deshabilitado
	1 habilitado
MPCM(*Multi-Procesor Communication Mode*)	0 deshabilitar modo comunicación entre varios MCU
	1 habilitar modo comunicación entre varios MCU

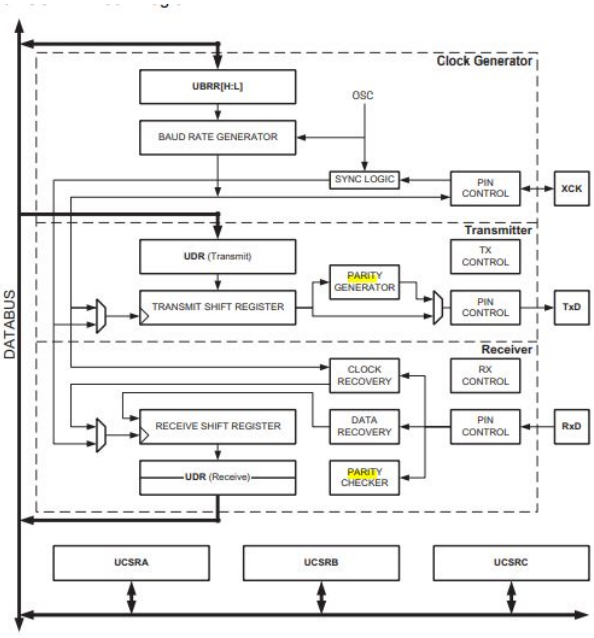
El registro UBRRH (*Baud Rate Register High*) y UBRRH (*Baud Rate Register Low*) permiten configurar la velocidad de transmisión en baudios. El valor se puede calcular con la siguiente fórmula:

$$Tasa\ de\ baudios = \frac{Fosc}{16(UBRR+1)}\ [\ bits/s] \qquad (3)$$

Generalmente se utilizan tasas de baudios establecidos para la comunicación serial, a continuación se muestra una tabla para cuando la frecuencia de oscilación del microcontrolador F_{OSC} sea 8MHz.

TABLA IV
VALORES PARA UBRRH|UBRRL PARA UNA FRECUENCIA DE OSCILACIÓN DE 8MHZ CON EL FIN DE ESTABLECER TASA DE BAUDIOS

Tasa de baudios a una frecuencia de oscilación de 8MHz [bits/s]	UBRRH	UBRRL
38400	0x0	0xC
19200	0x0	0x19
9600	0x0	0x33
4800	0x0	0x67
1200	0x1	0x9F



Captura 2:Diagrama de bloques del USART en ATMega16. Tomado desde: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

III. Desarrollo

La presente práctica emplea un teclado matricial 4x4, junto con su decodificador y una pantalla LCD de 16 caracteres con dos líneas. Además, como se trata de implementación serial, la comunicación entre el ATMega16 se llevará a cabo por medio de una terminal virtual disponible para simulaciones en Proteus. Cabe aclarar que habrá dos métodos de visualización dependiendo el modo seleccionado, uno donde la terminal virtual será el dispositivo de salida para la visualización de caracteres y otra será la LCD.

El sistema tendrá dos funciones principales, cada vez que se presione un botón del teclado matricial, se enviará dicho símbolo por serial y se mostrará en la terminal de proteus. El teclado matricial se deberá atender por interrupción externa. Además, el microcontrolador será capaz de recibir por serial un símbolo, este será mostrado en la LCD.

La estructura principal de las interrupciones resultaría en:

```
.include "m16def.inc"
.org 0
rjmp main
.org 0x2
rjmp tecla
.org 0x16
rjmp rx
```

Captura 3: Configuración Vectores de Interrupción.

Donde el vector en hexadecimal “2” corresponde a la primera interrupción externa (INT0) que se está asociada al pin 2 del puerto D. Por otro lado, el próximo vector de polarización “16” corresponde a la recepción por USART.

Dentro de lo que se denominó como “main”, se encuentra la inicialización de la pila junto con la del display LCD, configuración de los puertos y serial. El vector cargado con 33, será un contador que nos permitirá sobreescribir los caracteres en las primeras líneas una vez ya se encuentren llenas.

```
;inicializaci◆n del display lcd
call initlcd_4bits
ldi r23,33

;inicializaci◆n del serial
ldi r16,0b10000110  ;asincrono, sin paridad, 1 bit de stop, enviar 8 bits
out ucsrc,r16
ldi r16,0b10011000  ;rxcie habilitado, rx y tx habilitado, enviar 8 bits
out ucsrb,r16
ldi r16,0   ;baud rate 9600
out ubrrh,r16
ldi r16,51
out ubrrl,r16
```

Captura 4: Inicialización General dentro del main

Para la primera interrupción se tiene:

```
tecla:
    in r15,sreg
    push r15
    in r20,pinc
    ldi zh,high(0x0400<<1)
    ldi zl,low(0x0400<<1)
    add zl,r20
    lpm r16,z
    out udr,r16
    polling:
        sbis ucsra,udre
        rjmp polling
    pop r15
    out sreg,r15
    reti
```

Captura 5. Interrupción externa “tecla”

Con la segunda interrupción:

```
rx:
    in r15,sreg
    push r15
    in r16,udr
    dec r23
    breq primeralinea
    cpi r23,16
    breq segundalinea
    rjmp rxreti
    primeralinea:
        clt ;resetear bandera t para indicar que se enviar◆ byte de instrucci◆n
        ldi databyte,0x01 ;clear display
        call sendlcd_4bits
        ldi r23,33
        ldi databyte,0b10000000 ;cambiar direccion del cursor al primer punto
        call sendlcd_4bits  ;enviar instrucci◆n
    segundalinea:
        clt ;resetear bandera t para indicar que se enviar◆ byte de instrucci◆n
        ldi databyte,0b11000000 ;cambiar direccion del cursor al primer punto
        call sendlcd_4bits  ;enviar instrucci◆n
    rxreti:
        set
        mov databyte,r16
        call sendlcd_4bits  ;enviar dato
        pop r15
        out sreg,r15
        reti
```

Captura 6. Interrupción externa “tecla”

Las subrutinas “sendlcd_4bits”, “initlcd_4bits”, “retardo10ms”, “retardo500us”, “retardo40us” no serán explicadas en este informe debido a que son subrutinas utilizadas con anterioridad.

Se definió una tabla en memoria FLASH donde se encuentran los equivalentes de ASCII de las posibles combinaciones de 4 bits.

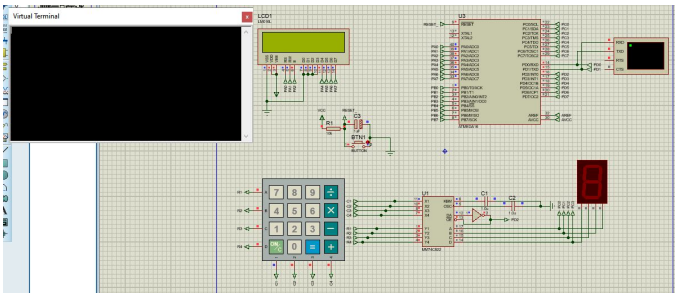
```
.org 0x400
.db '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'
```

Captura 7: Tabla caracteres equivalente ASCII.

Para comprobar el funcionamiento del sistema, se utilizará Proteus en su versión 8.8, donde se carga el archivo .hex que es generado en el directorio del proyecto.

Comprobación por Proteus.

Las salidas para ese sistema serán las ya mencionadas, pantalla LCD y la terminal virtual, dependiendo el sentido de la comunicación.

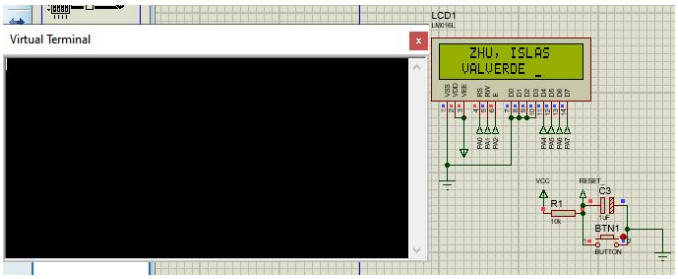


Captura 8: Sistema que integra al ATMega16, LCD, Switches y Terminal Virtual.

La frecuencia a la cual trabajará el ATMega16 se mantiene, por lo que se configura "CKSEL Fuses" que prepara al MCU a trabajar de cierta forma, en este caso específico "0100", lo que significa que funcionará con su reloj interior, hecho con un arreglo de RC a 8MHz.

IV. Resultados

Captura 9: Envío de datos desde el teclado matricial a la terminal virtual por serial.



Captura 10: Escritura en terminal virtual por medio del teclado físico de la PC y visualización en la LCD. Comunicación Serial.

V. Conclusiones

Andrés Islas Bravo:

Se practica permitió un involucramiento más a profundidad respecto a las comunicaciones entre dispositivos. se entendió la diversidad que existe entre las mismas y se empezó por contemplar un protocolo simple de detección de errores. el realizar todo este en codigo ensamblador permitió un dominio absoluto de lo sucedido en cada acción, sin embargo se pudo visualizar la importancia de futuros cursos donde los otros protocolos de comunicación más sólidos podrán ser implementados con mayor facilidad debido a las librerías ya existentes.

Juan Pablo Valverde López:

La implementación de interrupciones hace del sistema algo más sólido y mejor acondicionado para el trabajo real. Ahora, junto con la comunicación serial, se abre un mundo de aplicaciones entre dispositivos así como su implementación y desarrollo claro para futuras prácticas. El objetivo de la práctica se cumplió, el sistema fue manejado por medio de interrupciones y comunicado por serial, implementando conceptos vistos en clase.

Alfredo Zhu Chen:

Esta práctica me permitió manejar la comunicación serial USART de manera asíncrona en el teclado matricial para transmitir y por otro lado recibir un dato por serial en una terminal de Proteus para mostrarlo finalmente en una LCD. El objetivo de la práctica se cumplió porque se pudo implementar la transmisión y recepción de forma serial y asíncrona. Fue muy interesante la actividad ya que se estudió un protocolo ampliamente usando en la industria y tiene muchas aplicaciones para que lo usemos. Finalmente, logré probar el código escrito en el programa sobre la simulación de Proteus para observar el funcionamiento de manera visual.

VI. Referencias

[2] ATMEL. (2010). 8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash – AtMega Atmega16L. [Archivo PDF]. Obtenido desde: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf