Universidad Autónoma de Baja California
FAC. DE CS. QUIM. E INGENIERIA
INGENIERIA EN COMPUTACION

PRACTICA 8a

Laboratorio de: Microprocesadores y microcontradores

Equipo:

López Madrigal Leonardo

Maestro:
García López Jesús Adán

Tijuana, B. C. 16 Abril, 2017

Programación del uC en lenguaje C y comunicación serie.

Objetivo: Mediante esta práctica el alumno aprenderá el uso básico de la programación en lenguaje C con las herramientas AVR Studio y WinAVR. Para ello el alumno

implementará los procedimientos comunes para inicializar y operar el puerto serie del microcontrolador.

Equipo:

- Computadora Personal.
- Módulo T-Juino

Teoría:

- Programación en lenguaje C en microcontroladores.
- Manejo del Periférico de Comunicación Serie 0 (UART0) del microcontrolador ATmega1280/2560.

Teoría:

- Programación en lenguaje C en microcontroladores.

C es un lenguaje bastante conciso y en ocasiones desconcertante. Considerado ampliamente como un lenguaje de alto nivel, posee muchas características importantes, tales como: programación estructurada, un método definido para llamada a funciones y para paso de parámetros, potentes estructuras de control, etc. Sin embargo gran parte de la potencia de C reside en su habilidad para combinar comandos simples de bajo nivel, en complicadas funciones de alto nivel, y en permitir el acceso a los bytes y words del procesador. En cierto modo, C puede considerarse como una clase de lenguaje ensamblador universal. La mayor parte de los programadores familiarizados con C, lo han utilizado para programar grandes máquinas que corren Unix, MS-DOS, e incluso Windows (programación de drivers). En estas máquinas el tamaño del programa no es importante, y el interface con el mundo real se realiza a través de llamadas a funciones o mediante interrupciones DOS. Así el programador en C sólo debe preocuparse en la manipulación de variables, cadenas, matrices, etc.

Con los modernos microcontroladores de 8 bits, la situación es algo distinta. Tomando como ejemplo el 8051, el tamaño total del programa debe ser inferior a los 4 u 8K (dependiendo del tamaño de la EEPROM), y debe usarse menos de 128 o 256 bytes de RAM. Idealmente, los dispositivos reales y los registros de funciones especiales deben ser direccionados desde C. Las interrupciones, que requieren vectores en direcciones absolutas también deben ser atendidas desde C.

Además, se debe tener un cuidado especial con las rutinas de ubicación de datos para evitar la sobre escritura de datos existentes. Uno de los fundamentos de C es que los parámetros (variables de entrada) se pasan a las funciones (subrutinas) en la pila, y los resultados se devuelven también en la pila. Así las funciones pueden ser llamadas desde las interrupciones y desde el programa principal sin temor a que las variables locales sean sobre escritas. Una seria restricción de la familia 8051 es la carencia de una verdadera pila.

En un procesador como el 8086, el apuntador de la pila tiene al menos 16 bits. Además del apuntador de pila, hay otros registros que pueden actuar como apuntadores a datos en la pila, tal como el BP (Base Pointer). En C, la habilidad para acceder a los datos en la pila es crucial. Como ya ha sido indicado, la familia 8051 está dotada de una pila que realmente sólo es capaz de manejar direcciones de retorno. Con 256 bytes disponibles, como máximo, para la pila no se pueden pasar muchos parámetros y realizar llamadas a muchas funciones. De todo ello, puede pensarse que la implementación de un lenguaje que como C haga un uso intensivo de la pila, es imposible en un 8051. Hasta hace poco así ha sido. El 8051, hace tiempo que dispone de compiladores C, que en su mayor parte han sido adaptados de micros más potentes, tal como el 68000. Por ello la aproximación al problema de la pila se ha realizado creando pilas artificiales por software. Típicamente se ha apartado un área de RAM externa para que funcione como una pila, con la ayuda de rutinas que manejan la pila cada vez que se realizan llamadas a funciones. Este método funciona y proporciona capacidad de repetir variables locales a distintos niveles sin sobre escritura, pero a costa de hacer los programas muy lentos. Por lo tanto, con la familia 8051, la programación en

lenguaje ensamblador ha sido la única alternativa real para el desarrollo de pequeños sistemas en los que el tiempo es un factor crítico.

- Manejo del Periférico de Comunicación Serie 0 (UART0) del micro-controlador ATmega1280/2560.

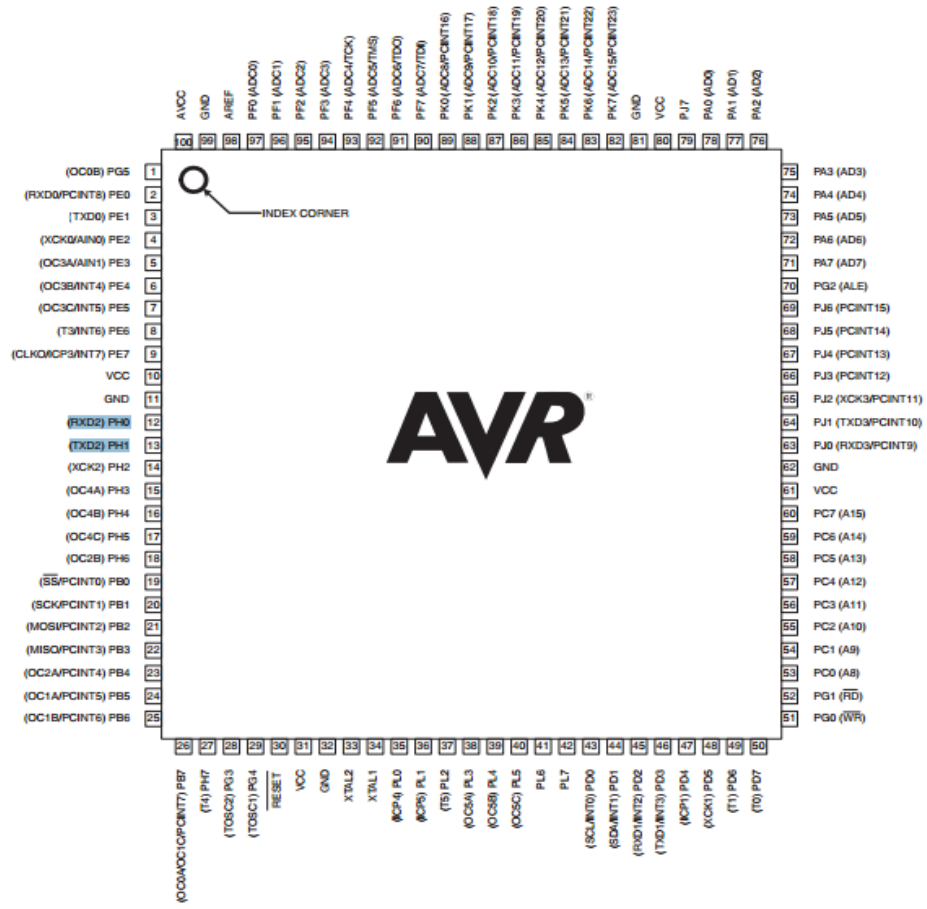
USART AVR, viene de receptor transmisor síncrono asíncrono universal, es una forma de comunicación entre dispositivos que tengan esta capacidad, donde los datos pueden ser enviados en grupos de 5, 6, 7, 8 o de 9 bits pero bit por bit, esto es en serie, por eso se dice que esta es una comunicación serial, en esta sección se comentará sobre la comunicación serial asíncrona utilizando el módulo USART del micro-controlador AVR, con el módulo USART AVR el micro-controlador puede comunicarse e intercambiar datos con el ordenador, con otros micro-controladores, etc.

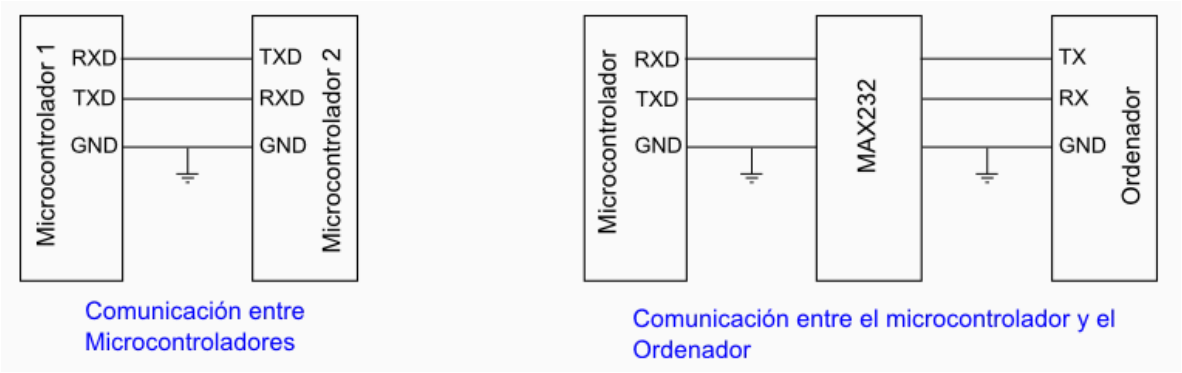
El micro-controlador AVR utilizado como referencia será el ATmega1280/2560, los pines de este micro-controlador que trabajan con el módulo USART AVR son el pin RXD o pin receptor y el pin TXD o pin transmisor, los que en la imagen están resaltados.

El pin RXD es el pin para la recepción de datos, El pin TXD es el pin para la transmisión de datos.

1. Pin Configurations

Figure 1-1. TQFP-pinout ATmega640/1280/2560





Programación de USART0

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

UDR0 en este registro se pondrá el carácter que se quiera transmitir, y también se encontrará el carácter recibido, tiene esta doble función, n es reemplazado por 0.

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/write	R	R/W	R	R	R	R	R/W	R/W	
Initial value	0	0	1	0	0	0	0	0	

UCSR0A Registro de control y estado A del módulo USART AVR, n es reemplazado por 0.

Descripción de cada bit:

RXC0 Este bit se pone a 1 automáticamente cuando se ha completado la recepción de algún dato en el registro UDR0, se pondrá a 0 automáticamente cuando se haya leído el dato, si se a habilitado el uso de la interrupción por recepción del módulo USART AVR, este bit se utiliza para detectar la interrupción.

TXC0 Este bit se pone a 1 automáticamente cuando se ha completado la transmisión de algún dato que se encontraba en el registro UDR0, se pondrá a 0 automáticamente cuando se cargue otro dato en el registro UDR0 a ser transmitido, si se a habilitado el uso de la interrupción por transmisión del módulo USART AVR, este bit se utiliza para detectar la interrupción.

UDRE0 Este bit al ponerse a 1 en forma automática indica que el registro UDR0 está vacío por lo que se le podrá cargar con algún dato. Cuando se cargue con algún valor el registro UDR0 este bit se pondrá automáticamente a 0. Se puede habilitar la interrupción por detección de que el registro UDCR0 está vacío y este bit será el que indique esa interrupción.

FE0 Este bit se pondrá a 1 automáticamente cuando hay un error en la recepción de algún dato, el error se detecta cuando el bit de parada del dato es un 0, el que normalmente debe de ser un 1. Se recomienda siempre poner este bit a 0 antes de recibir algún dato.

DOR0 Este bit se pondrá a 1 automáticamente cuando se sobrescribe algún dato del registro UDR0 que no haya sido leído, se pondrá a 0 automáticamente cuando se lea el dato, se recomiendo poner este bit a 0 antes de recibir algún dato.

UPE0 Este bit se pondrá a 1 automáticamente cuando se produce un error de paridad en la recepción de algún dato, se pondrá a 0 automáticamente cuando se lea el dato, se recomiendo poner este bit a 0 antes de recibir algún dato.

U2X0 Este bit interviene en la velocidad de los datos, esto es en los baudio que es la cantidad de bits por segundo en la comunicación serial, si es puesto a 0 se dice que la velocidad será normal y si es puesto a 1 se dice que será a doble velocidad.

MPCM0 Este bit es utilizado en el modo síncrono y es para detectar cuál de los micro-controladores esclavo ha sido elegido. En modo asíncrono se pondrá a 0.

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial value	0	0	0	0	0	0	0	0	

UCSR0B Registro de control y estado B del módulo USART AVR, n es reemplazado por 0.

Descripción de cada bit:

RXCIE0 Al poner este bit a 1 se habilita el uso de la interrupción USART AVR por recepción.

TXCIE0 Al poner este bit a 1 se habilita el uso de la interrupción USART AVR por transmisión.

UDRIE0 Al poner este bit a 1 se habilita el uso de la interrupción USART AVR la detección de que el registro UDR0 se quedó vacío.

RXEN0 Al poner este bit a 1 se habilita el uso del pin RXD para la recepción del módulo USART AVR. Se habilita es uso de la recepción.

TXEN0 Al poner este bit a 1 se habilita el uso del pin TXD para la transmisión del módulo USART AVR. Se habilita es uso de la transmisión.

UCSZ02 Este bit junto con los bits 2 y 1 del registro **UCSR0C** es para elegir de cuantos bits serán los datos a recibir o transmitir en la comunicación serial.

RXB80 Si se elige la comunicación serial a 9 bits, este será el noveno bit en la recepción del dato.

TXB80 Si se elige la comunicación serial a 9 bits, este será el noveno bit en la transmisión del dato.

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	1	1	0	

UCSR0C Registro de control y estado C del módulo USART AVR, n es reemplazado por 0.

Descripción de cada bit:

UMSEL01 y UMSEL00 son para elegir el modo de trabajo del módulo USART AVR, según la siguiente tabla.

Table 22-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

Note: 1. See "USART in SPI Mode" on page 227 for full description of the Master SPI Mode (MSPIM) operation.

UPM01 y UPM00 son para elegir si se utilizará o no algún bit de paridad para la detección de errores según la siguiente tabla:

Table 22-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

Note: 1. See "USART in SPI Mode" on page 227 for full description of the Master SPI Mode (MSPIM) operation.

USBS0 Este bit selecciona el número de bits de parada que se inserta por el transmisor. El receptor ignora este ajuste, según la siguiente tabla:

Table 22-6. USBS Bit Settings

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

UCSZ01 y UCSZ00 junto con **UCSZ02** del registro **UCSR0B** son para elegir de cuantos bits serán los datos a recibir o transmitir en la comunicación serial, según la siguiente tabla:

Table 22-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

UCPOL0 Este bit es utilizado en el bit síncrono, en el modo asíncrono se pondrá a 0.

Table 22-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRHn
	UBRR[7:0]								UBRRLn
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UBRR0L y UBRR0H Juntos estos registros hacen un solo registro de 16 bits el **UBRR0**, es donde se debe cargar el valor con el que se elige la velocidad de transmisión de los datos, esto es los baudios o la cantidad de bits por segundo que se enviarán en la comunicación serial con el módulo USART AVR, trabaja junto con el bit **U2X0** del registro **UCSR0A**.

El valor a cargar en el registro UBRR se calcula con las siguientes fórmulas que dependerán de la velocidad elegida con el bit **U2X0**, de la frecuencia del oscilador utilizado y de si la comunicación será síncrona o asíncrona.

Calculo del delay();

```
void delay( void){
    volatile unsigned int i;
    for(i=0;i<0xffff;i++);
}
```

```
13:                delay();
+000000AE:   940E00B1    CALL    0x000000B1        Call subroutine
14:                }
+000000B0:   CFFC        RJMP    PC-0x0013         Relative jump
@000000B1: delay
17:        void delay( void){
+000000B1:   93CF        PUSH    R28               Push register on stack
+000000B2:   93DF        PUSH    R29               Push register on stack
+000000B3:   B7CD        IN      R28,0x3D          In from I/O location
+000000B4:   B7DE        IN      R29,0x3E          In from I/O location
+000000B5:   9722        SBIW    R28,0x02          Subtract immediate from word
+000000B6:   B60F        IN      R0,0x3F           In from I/O location
+000000B7:   94F8        CLI                       Global Interrupt Disable
+000000B8:   BFDE        OUT     0x3E,R29          Out to I/O location
+000000B9:   BE0F        OUT     0x3F,R0           Out to I/O location
+000000BA:   BFCD        OUT     0x3D,R28          Out to I/O location
20:            for(i=0;i<0xffff;i++);
+000000BB:   821A        STD     Y+2,R1            Store indirect with displacement
+000000BC:   8219        STD     Y+1,R1            Store indirect with displacement
+000000BD:   C005        RJMP    PC+0x0006         Relative jump
+000000BE:   8189        LDD     R24,Y+1           Load indirect with displacement
+000000BF:   819A        LDD     R25,Y+2           Load indirect with displacement
+000000C0:   9601        ADIW    R24,0x01          Add immediate to word
+000000C1:   839A        STD     Y+2,R25           Store indirect with displacement
+000000C2:   8389        STD     Y+1,R24           Store indirect with displacement
+000000C3:   8189        LDD     R24,Y+1           Load indirect with displacement
+000000C4:   819A        LDD     R25,Y+2           Load indirect with displacement
+000000C5:   EF2F        SER     R18               Set Register
+000000C6:   3F8F        CPI     R24,0xFF          Compare with immediate
+000000C7:   0792        CPC     R25,R18           Compare with carry
+000000C8:   F7A9        BRNE    PC-0x0A           Branch if not equal
+000000C9:   9622        ADIW    R28,0x02          Add immediate to word
+000000CA:   B60F        IN      R0,0x3F           In from I/O location
+000000CB:   94F8        CLI                       Global Interrupt Disable
+000000CC:   BFDE        OUT     0x3E,R29          Out to I/O location
+000000CD:   BE0F        OUT     0x3F,R0           Out to I/O location
+000000CE:   BFCD        OUT     0x3D,R28          Out to I/O location
+000000CF:   91DF        POP     R29               Pop register from stack
+000000D0:   91CF        POP     R28               Pop register from stack
+000000D1:   9508        RET                       Subroutine return
+000000D2:   CFFF        RJMP    PC-0x0000         Relative jump
```

CALL	5
PUSH r28	2
PUSH r29	2
IN r28, 0x3D	1
IN r29, 0x3E	1
SBW r28, 0x02	2
IN r0, 0x3F	1
CLI	1
OUT	1
OUT	1
OUT	1
STD	2
STD	2
RJMP	2
LDD	2n
LDD	2n

ADIW	2n
STD	2n
STD	2n
LDD	2n + 2
SUB	2n + 2
SBC	2n
BRNE	2n – 1 + 2
ADIW	2
IN	1
CLI	1
OUT	1
OUT	1
OUT	1
POP	2
POP	2
RET	5

Os = 24 + 8n + 4 + 6n + n + 1 + n + 1 + 2 + 2n – 1 + 16

Os = 47 + 18n

Os = 47 + 18 (65534) = 1179659/16000000

Os = 0.073

Conclusión:

López Madrigal Leonardo

En la práctica 8a hicimos varias funciones de getch, putchar, gets, puts, atoi e itoa enfoncadas al micro-controlador 1280/2560 con la manera en que maneja la comunicación con el USART el cual usamos para transmitir o recibir, el MTTY tiene cierto *baudaje* que podemos cambiar y en el programa pudimos elegir cual usaríamos, en nuestro caso fue 9,600 y 19,200 el cual si no estaba bien asignado no podía entrar al programa de la flash y se miraba basura en la terminal, identifique los registros mapeados en memoria que son para la comunicación, e hice unos ajustes a varias funciones del datasheet del atmega 1280/2560.

Bibliografía o referencias:

- **http://www.atmel.com/Images/Atmel-0856-AVR-Instruction-Set-Manual.pdf**