Guía práctica para iniciar la programación en C de microcontroladores

Basada en la familia Atmel®AVR®XMEGA® B

Ejemplos sobre la tarjeta de desarrollo Xmega B1 Xplained.

Martha L Cano M

Esta guía es una introducción a la programación de microcontroladores usando el lenguaje C. A lo largo de esta guía, se presenta una teoría de funcionamiento general, la cual aplica a cualquier microcontrolador de 8 bits con arquitectura RISC; con base en esta teoría, en cada capítulo se explica la configuración de registros específicos de la familia de microcontroladores AVR®XMEGA® B de ATMEL® y para ilustrar las explicaciones, se desarrollan ejemplos para la tarjeta de desarrollo XMEGA B1 Xplained de Atmel.

Para entender los conceptos explicados y desarrollar los ejemplos aplicados, el lector debe tener conocimientos de programación en lenguaje C, de representación de números binarios, de lógica *booleana* y de circuitos eléctricos. A lo largo de la guía constantemente se hace referencia a hojas de especificaciones y otra literatura técnica de Atmel®; se recomienda ir descargando esta literatura a medida que se hace referencia a ella.

Contenido

1. Conociendo un microcontrolador

1.1 Elementos de un microcontrolador

Un microcontrolador es un circuito embebido, el cual reúne en un encapsulado varios componentes electrónicos, así como los caminos necesarios para interconectar estos componentes.

El componente principal de un microcontrolador es un microprocesador, el cuál comúnmente se referencia como CPU (unidad central de procesamiento). Para que un procesador pueda cumplir su función de procesamiento requiere de memorias (de programa y de datos), de buses (interconexiones), de un reloj y de un circuito de alimentación eléctrica. Además de embeber estos elementos, el microcontrolador también incluye otros componentes electrónicos, que tienen funciones específicas, los cuales le agregan versatilidad y flexibilidad al procesador. Estos componentes son los periféricos. Los periféricos más comunes son: puertos digitales de Entrada/Salida, temporizadores, contadores, conversores análogo-digital y digital-análogo e interfaces de comunicación.

Cada fabricante de microcontroladores, adiciona otros periféricos u otras funcionalidades concebidas para ciertas aplicaciones, con el fin de promover y facilitar el uso de sus microcontroladores en áreas específicas de la Electrónica.

En la Figura 1 se presenta el diagrama de bloques del microcontrolador ATXMEGA128B1 de Atmel. En este diagrama en bloques se puede ver que el microcontrolador tiene 100 pines, de los cuales 6 se usan para la alimentación positiva del microcontrolador (naranja), 6 para la tierra (negra), 19 para funciones digitales (azul), 21 para funciones análogas y osciladores (verde), 44 dedicados al manejo de un display LCD (azul claro), 2 para programación (azul oscuro) y 2 para otros propósitos (gris-blanco), incluso se aprecia que algunos pines están compartidos para dos de estas funciones.

En este diagrama de bloques se tienen los componentes mencionados al inicio: CPU, buses, memorias (SRAM, Flash, EEPROM), así como periféricos u otros componentes necesarios para la programación y depuración del código.

Todos los microcontroladores, de cualquier fabricante, presentan un diagrama en bloques en su hoja de especificaciones. Para familiarizarse con el microcontrolador con el cual se desea trabajar, un buen inicio es analizar el diagrama en bloques.

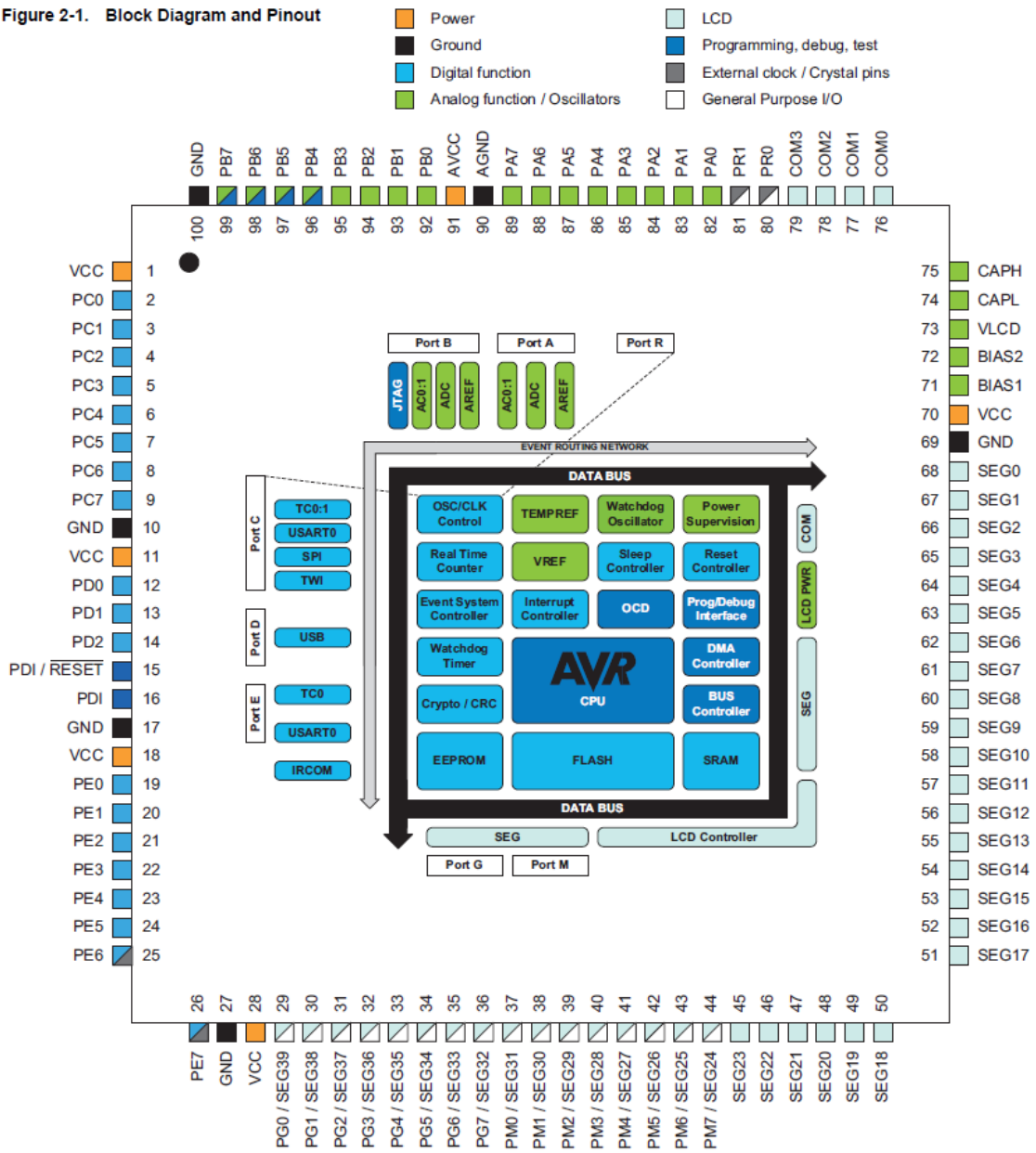


Figura 1 Diagrama en bloques del procesador ATXMEGA 128B1. Tomado de: ATxmega128B1_64B1.pdf Sección 2

Ejercicios

I. A partir del texto, explique en sus palabras una diferencia entre un microprocesador y un microcontrolador.

II. Investigue qué significan las siglas que se ven en los bloques azul y azul oscuro de la Figura 1.

III. Investigue qué es y para qué sirve un bus de datos en un microcontrolador

1.2 Memoria de programa

En general, todo microcontrolador cuenta con una memoria de programa y una memoria de datos. La memoria de programa almacena el código ejecutable, es decir el código que nosotros diseñamos, escribimos y programamos en el microcontrolador. Este código debe permanecer, aun si el microcontrolador no tiene alimentación eléctrica, por lo tanto, la memoria de programa es no volátil y por lo general se usa una memoria de tecnología flash. Apenas se le suministra energía eléctrica al microcontrolador, la CPU irá a leer la posición 0 de la memoria flash, en donde debe estar la primera instrucción del código ejecutable.

La memoria de programa tiene un tamaño limitado, el cual es relativamente pequeño comparado a los computadores de uso personal que conocemos. En el caso del microcontrolador ATXMEGA128B1, el tamaño de memoria de programa es de 128 kbytes.

1.3 Memoria de datos

En un microcontrolador, la memoria de datos se divide en tres secciones: la sección que contiene los registros de configuración del microcontrolador, la sección que contiene datos volátiles y la sección que contiene datos no volátiles.

En la sección de datos volátiles se almacenan las variables que usamos en nuestros programas. Por ejemplo, si realizamos la declaración y asignación de la variable ***i***, según la instrucción (I1), entonces el valor 35 correspondiente a esta variable se almacenará en la memoria de datos volátil.

$$int\ i = 35; \qquad (I1)$$

El valor de una variable almacenada en la sección de datos volátiles, se pierde cuando quitamos la alimentación eléctrica al microcontrolador.

En ocasiones, deseamos que ciertas variables se conserven aun si el microcontrolador no tiene alimentación eléctrica, en ese caso se deben almacenar en la sección de datos no volátiles de la memoria de datos.

Por último, los registros de configuración se usan para almacenar las características de funcionamiento de los periféricos. Esta configuración se realiza en el código que diseñamos, escribimos y programamos. A lo largo de esta guía se hablará ampliamente de los registros de configuración.

En la Figura 2 se presenta el mapa de la memoria de datos del microcontrolador ATXMEGA128B1, en el cual se puede apreciar:

- La sección de registros de configuración (I/O registers)
- la sección de datos no volátiles (EEPROM)
- la sección de datos volátiles (Internal SRAM)

Las referencias, 2k, 4k, y 8k hacen referencia al tamaño de cada sección: 2000, 4000 y 8000 posiciones de memoria respectivamente.

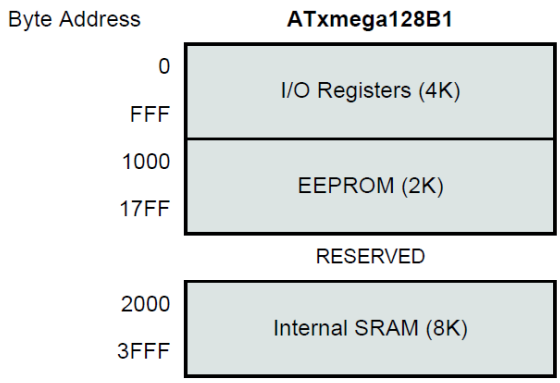


Figura 2 Mapa de memoria de datos del procesador ATXMEGA 128B1. Tomado de: ATxmega128B1_64B1.pdf Sección 7

Ejercicios

I. Investigue los términos “volátil” y “no volátil” para memorias y que tecnología se utiliza para la construcción de cada uno de estos tipos de memoria.

II. Compare las ventajas y desventajas entre una memoria SRAM y una EEPROM.

III. Investigue aproximadamente cuántas líneas de código tienen los programas que usted más usa en su computador personal y compárelas con el espacio de memoria disponible en el microcontrolador ATXMEGA128B1.

2. Interfaces y especificaciones eléctricas

2.1 Pines de un microcontrolador

Los pines de un microcontrolador son el punto de interacción entre el microcontrolador y el mundo exterior. Por lo general, en un microcontrolador de 8 bits, los pines se agrupan por 8 y se les asigna un prefijo que diferencia a cada grupo. Por ejemplo, en los microcontroladores Atmel se usan los prefijos PA, PB, PC etc., para cada grupo de 8 bits. Dentro de un mismo grupo cada pin tiene asignado un número del 0 al 7, así, el primer pin del grupo PA, se designa como PA0, el segundo pin como PA1, hasta el octavo pin que se designa como PA7. En el caso de los microcontroladores Microchip, los prefijos usados para los grupos son RA, RB, RC etc.; de igual forma que para Atmel, cada pin en el grupo se identifica con un número del 0 al 7, de tal forma que los nombres de un pin serán algo como RB0, RB1 etc.

Un solo pin de un microcontrolador puede ser compartido por diferentes periféricos; sin embargo, sólo un periférico puede tener control a la vez sobre un pin, lo cual se logra al configurar los registros del periférico. Para comprender mejor como comparten los periféricos los pines, se puede observar un diagrama en bloques detallado del microcontrolador ATXMEGA128B1, el cual se presenta en la Figura 3.

En este diagrama se observa como los diferentes grupos de pines van conectados a varios periféricos. Por ejemplo, los pines del grupo PC pueden ser usados por el TWIC, el SPIC, la USARTC0 y el TCC0:1, sin embargo, solo uno de estos periféricos los podrá usar a la vez.

2.2 Especificaciones eléctricas de un microcontrolador

Los pines del microcontrolador nos permiten adquirir y entregar señales desde y hacia el mundo exterior, sin embargo, estas señales deben estar acondicionadas para no exceder las especificaciones eléctricas del microcontrolador. Estas especificaciones siempre se podrán encontrar en la hoja de especificaciones del microcontrolador que se desea usar o en su defecto, en la hoja de especificaciones de la familia a la cual pertenece el microcontrolador.

La primera especificación que nos interesa es el **voltaje de alimentación**, esta especificación es un rango de voltaje; si el microcontrolador se alimenta con un voltaje menor al valor especificado no se encenderá, si se alimenta con un voltaje superior sufrirá un daño irreversible. La alimentación eléctrica siempre se debe proveer entre los pines de fuente (VCC o VDD) y tierra (GND o VSS), respetando la polaridad convencional. Si se invierte la polaridad de alimentación, también habrá un daño irreversible en el microcontrolador.

La siguiente especificación de gran importancia es el rango de voltaje que acepta un pin del microcontrolador. Si la función del pin es digital, se debe asegurar que en el pin haya un 0 o un 1, en términos de voltajes un 0 corresponde a 0 voltios y un 1 a Vcc voltios, donde Vcc es el voltaje con el cual se está alimentando el microcontrolador. Si la función del pin es análoga, por lo general el pin aceptará

cualquier voltaje entre 0 voltios y Vcc voltios. Exceder Vcc en cualquier pin del microcontrolador causará daños irreversibles.

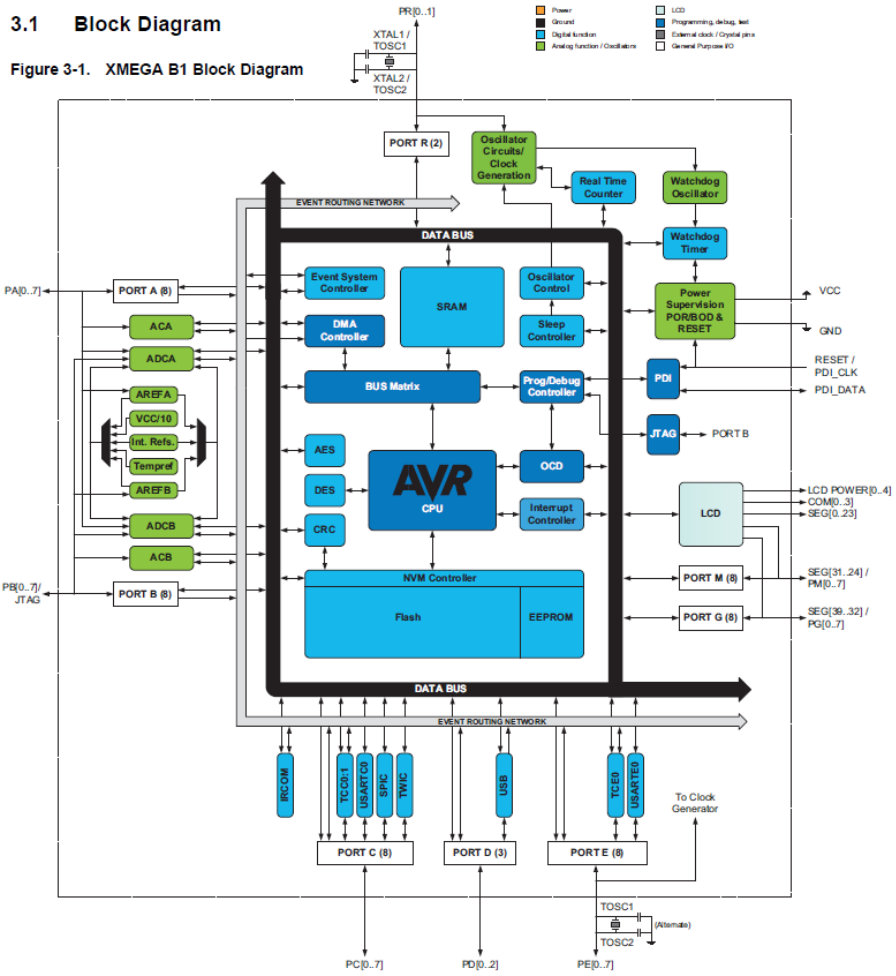


Figura 3 Diagrama en bloques detallado del ATXMEGA128B1. Tomado de: ATxmega128B1_64B1.pdf Sección 3

Las especificaciones de corriente son muy importantes, estas se refieren a cuánta corriente puede suministrar o recibir un pin del microcontrolador. Las especificaciones de corriente varían de un fabricante a otro, e incluso de una referencia de microcontrolador a otro, pero por lo general se sitúan alrededor de las decenas a las centenas de miliamperios.

Se recomienda analizar detalladamente las especificaciones eléctricas de un microcontrolador antes de conectar otros componentes electrónicos a sus pines.

La Tabla 1 presenta, las principales características eléctricas del microcontrolador ATXMEGAB1. En esta tabla podemos ver que el microcontrolador se debe alimentar con un voltaje entre 1.6 V y 4 V y que cada pin puede suministrar o recibir máximo 25 mA. Sin embargo, la suma de todos los pines de un mismo grupo no debe exceder 100 mA o 200 mA, según el grupo.

Tabla 1 Características eléctricas del microcontrolador ATXMEGA128B

Absolute maximum ratings	
V_{CC} : Power Supply voltage	-0,3v – 4V
I_{VCC} : Current into a Vcc pin	200 mA
I_{GND} : Current out of a GND pin	200 mA
V_{PIN} : Pin voltage with respect to GND and Vcc	-0.5; Vcc +0.5
I_{PIN} : I/O pin sink/source current	+-25 mA
General Operating Ratings	
V_{CC}	1,6 – 3,6 V
A_{VCC}	1.6 – 3.6 V
CLK_{CPU}	12Mhz max @1.6V 32Mhz max @3.6V
I_{cc}, Active power consumption	1.6mA @3V, 2MHz
I/O pin characteristics	
I_{OH}/I_{OL} : I/O pin source/sink current	20 mA
The sum of all I_{OH} or I_{OL} for PORTA and PORTB must not exceed	100 mA
The sum of all I_{OH} or I_{OL} for PORTC, PORTD, PORTE, and PDI must for each port not exceed	200 mA
ADC characteristics	
V_{REF}: Reference voltage	1 – A_{vcc}-0.6
V_{IN}: Input range	0 - V_{REF}

Ejercicios

I. A partir de lo explicado en esta sección indique 3 formas de dañar un microcontrolador.

II. Intente deducir qué significa cada una de las especificaciones de la Tabla 1, luego investigue su significado real y compárelo con lo que usted dedujo.

III. Descargue los esquemáticos de la tarjeta de desarrollo XMEGAB1 Xplained; a partir de estos esquemáticos indique con qué voltaje se alimenta el microcontrolador ATXMEGA128B1 que contiene esta tarjeta.

IV. Analice en los esquemáticos qué circuito está conectado a cada pin del microcontrolador ATXMEGA128B1.

3. Puertos digitales de entrada y salida

3.1 Dato digital de entrada

Los puertos de entrada salida (E/S) son periféricos que, a través de los pines del microcontrolador, permiten recibir datos digitales del entorno o enviar datos digitales hacia el entorno. Los datos digitales recibidos serán procesados por el microprocesador de acuerdo al código que tenga programado.

El dato digital de entrada más sencillo está conformado por un solo bit. Este es el caso del dato generado por un pulsador, conectado a una resistencia y un condensador como se muestra en la Figura 4. Para simplificar el análisis de este circuito se va a suponer que la corriente que va por el pin del micro es 0 A.

Cuando el pulsador está abierto, en estado estable, el condensador está cargado a V_{CC}, por lo tanto, en el pin del micro se ve un "1" lógico.

Cuando el pulsador se cierra, en estado estable, hay un corto circuito entre tierra y el pin del micro, a través del pulsador cerrado, por lo tanto, en el pin del micro se ve un "0" lógico.

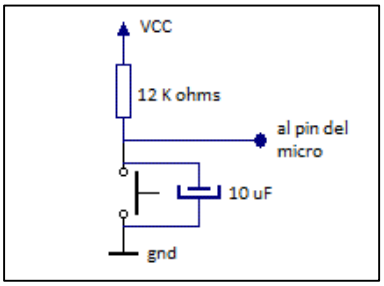


Figura 4 Un pulsador, el dato digital de entrada más sencillo

Con este circuito conectado al pin, se ve que el microcontrolador recibe un dato digital que puede ser 0 o 1.

Otras entradas digitales pueden ser las provenientes de sensores que tienen salidas digitales. Por lo general estos sensores ya entregan la señal acondicionada, es decir entregan V_{CC} (1 lógico) cuando se activan y Tierra (0 lógico) cuando se desactivan, o viceversa, sin embargo, se debe verificar que el V_{CC} del sensor coincida con el V_{CC} del microcontrolador. En el caso que se necesite algún acondicionamiento, las hojas de especificaciones del sensor lo dirán. Algunos ejemplos de sensores con salidas digitales son: sensor capacitivo, sensor infrarrojo, sensor de ultrasonido, sensor de nivel, entre muchos otros.

Si un dispositivo de entrada (pulsador, sensor, etc) entrega un "1" lógico al ser estimulado se dice que este dispositivo es "activo en alto". En este caso el valor que entrega por defecto o en reposo es un "0" lógico.

Si el dispositivo entrega un “0” lógico al ser estimulado se dice que este dispositivo es “activo en bajo”. En este caso el valor que entrega por defecto es un “1” lógico.

Ejercicios

I. ¿El circuito del pulsador de la Figura 4, es activo en alto o activo en bajo?

II. Diseñe un circuito para un pulsador que su activación sea opuesta al de la Figura 4.

III. Investigue a qué se refiere el término “rebote” en un pulsador.

IV. Investigue el circuito anti-rebote basado en un *schmitt trigger* y entienda su funcionamiento

3.2 Dato digital de salida

Para enviar datos digitales hacia el entorno, se pone un 1 lógico o un 0 lógico en un pin del microcontrolador, esto se logra por medio del código que se diseñe y programe en él.

Para que este dato sea recibido por el entorno debe existir un actuador conectado al pin del microcontrolador, el cual permita visualizar el dato digital o transmitirlo a otros dispositivos.

El actuador más sencillo que recibe un dato digital es un LED, conectado a una resistencia, como se ve en la Figura 5.

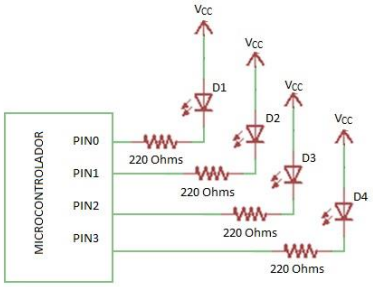


Figura 5 Un LED, el actuador más sencillo.

Cada LED de la Figura 5 se prende cuando en el pin correspondiente del microcontrolador hay un 0, esto se llama “activo en bajo”. Cuando hay un “1” en el pin correspondiente, este “1” es igual a V_{CC} por lo tanto el LED no se prende, ya que la diferencia de potencial entre sus terminales es 0 V.

La corriente que circula por cada LED cuando está encendido es 5.9 mA, esta es la misma corriente que entra al pin de microcontrolador, por lo tanto, se debe verificar que esta corriente no exceda la especificación de corriente del microcontrolador.

Una gran mayoría de los actuadores que se conectan a los pines de un microcontrolador funcionan con una corriente más alta que la que permite la especificación del pin, o con un voltaje más alto que la

fuente de alimentación de microcontrolador V_{CC}. En estos casos se debe diseñar una etapa de acondicionamiento de señal para el actuador, como la que se ve en la Figura 6. En el Apéndice C. Lo básico de circuitos eléctricos. se analiza este circuito.

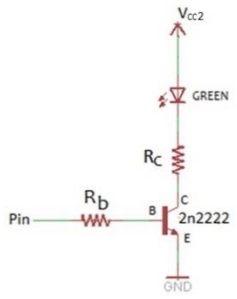


Figura 6 Acondicionamiento de señal digital de salida

Algunos ejemplos de otros actuadores digitales, es decir que tienen dos estados son: relevos, motores, chicharras u otros dispositivos mecánicos.

Una salida digital también puede ser la entrada digital de otro sistema (por ejemplo, de otro microcontrolador). En este caso se debe verificar que los dos sistemas se alimenten al mismo voltaje. Si se alimentan a voltajes diferentes se debe hacer un acondicionamiento de señal.

Ejercicios

I. En sus palabras, escriba una definición de activo en alto y activo en bajo para un actuador.

II. Diseñe un circuito para un LED “activo en alto”, es decir que se encienda cuando hay un 1 en el pin del microcontrolador.

3.3 Puertos E/S

Por lo general, se hace referencia a los puertos digitales E/S, usando el término abreviado en inglés, I/OPORT. Cada I/OPORT agrupa los 8 pines de un mismo grupo; esto facilita la configuración de los registros y la programación del código. Para hacer la administración de un I/OPORT existen unos registros especiales llamados registros de configuración.

Cuando un pin está siendo utilizado por el periférico I/OPORT, lo primero que se debe configurar es si este PIN actuará como una entrada digital o como una salida digital. Evidentemente, un PIN sólo podrá cumplir una de estas dos funciones a la vez y debe estar configurada explícitamente por medio de los registros de configuración.

Una vez se ha definido su función como entrada o como salida, se puede leer el PIN o escribir en el PIN, respectivamente. Esta lectura o escritura también se realiza usando los registros de configuración.

En la Figura 7, se presenta el circuito electrónico que maneja y acondiciona la señal de cada pin de un I/OPORT para el microcontrolador ATXMEGA128B1. En esta figura se puede apreciar los componentes que se activan cuando el PIN es configurado como salida, así como aquellos que se activan cuando el PIN es configurado como entrada.

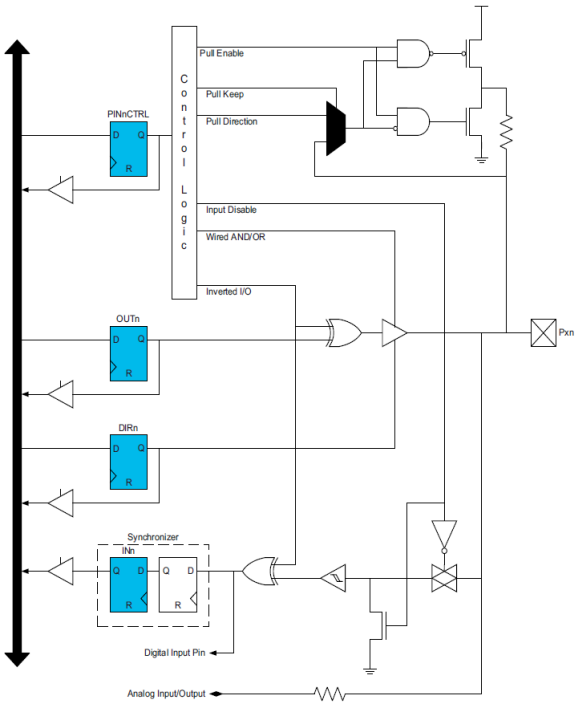


Figura 7 Circuito electrónico de un I/OPIN del ATXMEGA128B1. Tomado de: 8-bit Atmel XMEGA B Microcontroller.pdf Sección 12

3.4 Registros básicos de configuración de los I/O PORTS

Los registros de configuración definen el comportamiento de cada periférico. El número de registros por periférico varía según la complejidad de este, el fabricante y la familia del microcontrolador con el cual se está trabajando. Sin embargo, en el caso de los I/O PORTS siempre existirán 3 registros básicos, el que permite configurar la dirección de cada pin del puerto, el que permite leer el dato de cada pin del puerto cuando este ha sido configurado como entrada y el que permite escribir un dato en cada pin del puerto cuando este ha sido configurado como salida.

Para el microcontrolador ATXMEGA128B1 estos 3 registros básicos son DIR, OUT e IN:

Registro DIR: Define la dirección del pin. Si es entrada o salida. Si se escribe un 1 en el bit n, el pin n es configurado como salida. Si se escribe un 0 en el bit n, el pin n es configurado como entrada.

Registro OUT: Cuando la dirección se define como salida, el registro OUT se usa para poner un valor en el pin. Si se escribe un 1 en el bit n, el pin n se llevará a un alto (VCC). Si se escribe un 0 en el bit n, el pin n se llevará a un bajo (0 V).

Registro IN: Este registro se usa para leer los valores en el pin. Se usa cuando el pin es declarado como entrada.

Ejemplo aplicado de puertos E/S

Se quiere alternar el encendido y el apagado del LED0 de la tarjeta AVR XMEGAB1 XPLAINED, cada vez que un pulsador externo es presionado.

Paso 1. Determinar a cuáles pines van conectados el LED y el Pulsador.

- *Observando los esquemáticos de la tarjeta de desarrollo vemos que el LED0 está conectado al pin 4 del puerto B*
- *Decidimos conectar el pulsador al pin 0 del puerto A, ya que en ese puerto no hay conectados otros elementos de la tarjeta.*

Paso 2. Configurar los registros DIR de los puertos B y A. Los registros de cada periférico están agrupados en una estructura de datos, para acceder a un registro se usa el nombre del periférico, seguido del dato de la estructura al cual se quiere acceder. La forma general de acceder a un registro es PERIFERICO.REGISTRO.

```
PORTB.DIR |= 0x10;      // El pin 4 del PORTB se declara como salida. Los demás bits no se
modifican.
PORTA.DIR &= ~(0x01);   // El pin 0 del PORTA se declara como entrada. Los demás bits no se
modifican.
```

Código 1 Declaración de la dirección de los pines

Paso 3. Escribir en el registro OUT, el valor inicial del pin de salida PB4

El LED0 es activo en bajo, por lo tanto, si se quiere que inicie apagado se debe escribir un 1 en el bit 4 del registro OUT.

```
PORTB.OUT |= 0x10;      // Pone en 1 el pin 4 del PORTB. No modifica los demás pines del
puerto.
```

Código 2 Escritura del registro OUT

Paso 4. Escribir el programa usando los registros OUT e IN.

Se quiere que el LED0 se prenda y apague alternadamente cada vez que se presiona el pulsador.

Supongamos que el pulsador que vamos a conectar es activo en bajo. Por lo tanto, debemos preguntar si el bit 0 del registro PORTA.IN está en 0 para alternar el valor del pin del LED0.

```
if((PORTA.IN & 0x01) == 0)//Se evalúa el bit 0 del registro IN del puerto A.
{
        if((PORTB.OUT & 0x10) == 0)//Se evalúa si el LED está encendido
        { PORTB.OUT |= 0x10;    }  //si está encendido se apaga
        else
        {PORTB.OUT & = ~(0x10); }       //si está apagado se prende
}
```

Código 3 Encendido y apagado de un LED

Paso 5. Incluir en un bucle infinito el programa para que se pregunte recurrentemente por el pulsador. El programa final es:

```
#include <avr/io.h>
int main(void)
{
        PORTB.DIR |= 0x10;
        PORTA.DIR &= ~(0x01);
        PORTB.OUT |= 0x10;
        while(1)
        {
                if((PORTA.IN & 0x01) == 0)
                {
                        if((PORTB.OUT & 0x10) == 0)
                               { PORTB.OUT |= 0x10;}
                        else
                               {PORTB.OUT & = ~0x10; }
                }
        }
}
```

Código 4 Programa ejemplo aplicado I/OPORT

Ejercicios

I. El Código 4 no cumple la especificación dada. Si se deja presionado el pulsador, el LED se prende y se apaga, sin embargo, la especificación pide "alternar el encendido y el apagado del LED0 **cada vez** que un pulsador externo es presionado ". Diseñe un código que cumpla la especificación.

II. Escriba, compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el código del ejercicio anterior. En un protoboard monte un pulsador activo en bajo usando un circuito antirrebote basado en un schmitt trigger, conéctelo al pin PA0 de la tarjeta y pruebe el programa diseñado.

III. Descargue la hoja de especificaciones “8-bit Atmel XMEGA B Microcontroller.pdf”, busque, lea y entienda la descripción de los registros descritos en esta sección.

3.5 Registros que facilitan la programación

Cada fabricante de microcontroladores adiciona otros registros a los periféricos I/OPORT para facilitar su uso o para que tengan mayor versatilidad. En la hoja de especificaciones de cada microcontrolador se describen la función y uso de estos registros.

En el microcontrolador ATXMEGA128B1, hay 5 registros que facilitan la escritura de los registros DIR y OUT, estos registros son DIRSET, DIRCLR, OUTSET, OUTCLR y OUTTGL.

Registro DIRSET: Este registro reemplaza el uso de las máscaras de bit. Permite escribir 1 en los bits deseados del registro DIR sin alterar los demás bits. Al escribir un 1 en un bit de DIRSET, el bit correspondiente en DIR se pondrá en 1. Al escribir un 0 en un bit de DIRSET, el bit correspondiente en DIR no se modifica.

Para el ejemplo aplicado de la sección anterior, la configuración del pin 4 del PORTB como salida se realiza con la instrucción del Código 5.

```
PORTB.DIRSET = 0x10;            // El pin 4 del puerto se declara como salida. Los demás bits
                                                                    del registro no se modifican.
```

Código 5 Uso del registro DIRSET

Registro DIRCLR: Este registro reemplaza el uso de las máscaras de bit. Permite escribir 0 en los bits deseados del registro DIR sin alterar los demás bits. Al escribir un 1 en un bit de DIRCLR, el bit correspondiente en DIR se pondrá en 0. Al escribir un 0 en un bit de DIRSET, el bit correspondiente en DIR no se modifica.

Para el ejemplo aplicado de la sección anterior, la configuración del pin 0 del PORTA como salida se realiza con la instrucción del Código 6.

```
PORTA.DIRCLR = 0x01;            // El pin 0 del puerto se declara como entrada. Los demás bits
                                                                    del registro no se modifican.
```

Código 6 Uso del registro DIRCLR

OUTSET: Este registro reemplaza el uso de las máscaras de bit. Permite escribir 1 en los bits deseados del registro OUT sin alterar los demás bits. Al escribir un 1 en un bit de OUTSET, el bit correspondiente en OUT se pondrá en 1. Al escribir un 0 en un bit de OUTSET, el bit correspondiente en OUT no se modifica.

Para el ejemplo aplicado de la sección anterior, la inicialización del pin 4 del PORTB, donde va conectado el LDE0, se realiza con la instrucción del Código 7.

```
PORTB.OUTSET = 0x10;            // Pone en 1 el pin 4 del puerto. No modifica los demás pines
                                del puerto.
```

Código 7 Uso del registro OUTSET

OUTCLR: Este registro reemplaza el uso de las máscaras de bit. Permite escribir 0 en los bits deseados del registro OUT sin alterar los demás bits. Al escribir un 1 en un bit de OUTCLR, el bit correspondiente en OUT se pondrá en 0. Al escribir un 0 en un bit de OUTSET, el bit correspondiente en OUT no se modifica.

OUTTGL: Este registro sirve para invertir el valor de los bits del registro OUT sin alterar los demás bits. Al escribir un 1 en un bit de OUTTGL, el bit correspondiente en OUT invertirá su valor. Esto significa que si en el pin correspondiente había un 1 se escribirá un 0 y si había un 0 se escribirá un 1. Al escribir un 0 en un bit de OUTTGL, el bit correspondiente en OUT no se modifica.

Para el ejemplo aplicado de la sección anterior, alternar el encendido y apagado del LED0 se realiza con la instrucción del Código 8 Uso del registro OUTTGL.

```
    PORTB.OUTTGL = 0x10;            // Cambia el valor del pin 4 del pin 4 del puerto. No
                                modifica los demás pines del puerto.
```

Código 8 Uso del registro OUTTGL

Al usar DIRSET, DIRCLR, OUTSET y OUTTGL el programa del ejemplo aplicado de la sección anterior se simplifica, como se puede ver en el Código 9 Programa simplificado I/OPORT.

```
#include <avr/io.h>
int main(void)
{
        PORTB.DIRSET = 0x10;
        PORTA.DIRCLR = 0x01;
        PORTB.OUTSET = 0x10;
        while(1)
        {
                if((PORTA.IN & 0x01) == 0)
                {
                        PORTB.OUTTGL = 0x10;
                }
        }
}
```

Código 9 Programa simplificado I/OPORT

Ejercicios

I. Usando máquinas de estado finitas, diseñe un código que cuente el número de veces que se presionó un pulsador conectado a PA0 y muestre ese valor en binario usando los 4 LEDs de la tarjeta de desarrollo XMEGAB1 Xplained. Utilice los registros explicados en esta sección.

II. Escriba, compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el código del ejercicio anterior. Monte el circuito que permite probar el código desarrollado, incluya el circuito antirrebote que usa un Schmitt trigger.

4. Temporizadores

4.1 Funcionamiento de un temporizador

En el diseño de sistemas microprocesadores es muy común que se deban realizar tareas de forma periódica, o que se deba esperar un tiempo antes de realizar el siguiente paso de un proceso. Para dar solución a estos requerimientos, hay dos opciones: que el procesador cuente él mismo el tiempo requerido y por lo tanto, que no pueda realizar ningún otro procesamiento mientras cuenta o que otro componente cuente por él e informe cuando el tiempo requerido haya transcurrido.

La primera opción es la que implementan algunas funciones de *Delay*, esta opción es ineficiente y en ocasiones inviable porque el microprocesador debe atender múltiples tareas y no puede quedarse esperando por una sola tarea. En la segunda opción, el componente que puede llevar el conteo se llama un temporizador, o **TIMER** en inglés**.** El TIMER es un periférico embebido en el microcontrolador, que se configura por medio de registros para que realice el conteo que se requiera.

Supongamos que el tiempo requerido es *T* segundos; teniendo en cuenta que el TIMER cuenta de uno e uno, lo primero que debemos determinar es a qué velocidad cuenta. La velocidad de conteo varía según el microcontrolador, pero siempre estará explícita en las hojas de especificaciones, por medio de la frecuencia de reloj de los periféricos (CLK_{PER}).

La frecuencia de reloj nos indica cuantos sucesos ocurren en un segundo, por ejemplo, una frecuencia de 2 MHz, indica que dos millones de sucesos ocurren en un segundo, así, la velocidad está definida por el inverso de la frecuencia y sus unidades son segundos. En (1) se calcula la velocidad de conteo para una frecuencia de 2MHz, este resultado indica que el TIMER contaría de uno en uno cada 0,5 µs. En (2) se presenta la ecuación general.

$$\frac{1}{2MHz} = 0{,}5\ \mu s \qquad (1)$$

$$\mathrm{Vel}_{\mathrm{conteo}} = \frac{1}{\mathrm{CLK}_{\mathrm{PER}}} \qquad (2)$$

Con la velocidad de conteo se puede determinar hasta cuánto debe contar el TIMER para cumplir el tiempo requerido *T*. En (3) se presenta la ecuación para determinar el número de conteos del TIMER

$$Conteo = \frac{T}{Vel_{conteo}} = \frac{T}{\frac{1}{CLK_{PER}}} \qquad (3)$$

Ejemplo numérico del conteo de un temporizador

Se requiere que el TIMER cuente un tiempo de 0.5 segundos, con un reloj a una frecuencia de 31250 KHz. Usando (3) se calcula que el TIMER deberá contar hasta 15625.

4.2 Temporizadores del ATXMEGA128B1

El microcontrolador ATXmega 128B1 tiene dos TIMERs tipo 0 y un TIMER tipo 1. En esta guía se hablará de los TIMERs tipo 0. Los nombre de los dos TIMER tipo 0 son: **TCC0 y TCE0,** estos nombres vienen de **TIMER C**ounter port**C** type**0** y **TIMER C**ounter port**E** type**0**, de lo cual se puede deducir que cada uno está asociado al puerto E y al puerto C.

La operación por defecto de estos TIMERs es:

- Con cada ciclo de reloj, el registro de 16 bits CNT aumenta en 1 su valor, cuenta desde cero hasta un valor almacenado en el registro de 16 bits PER.
- El registro CNT se compara continuamente con el valor en PER.
- Cuando CNT llega al valor almacenado en PER, se levanta una bandera y CNT se reinicia a 0.

La Figura 8 Diagrama de bloques del TIMER0 presenta el diagrama de bloques del TIMER tipo 0.

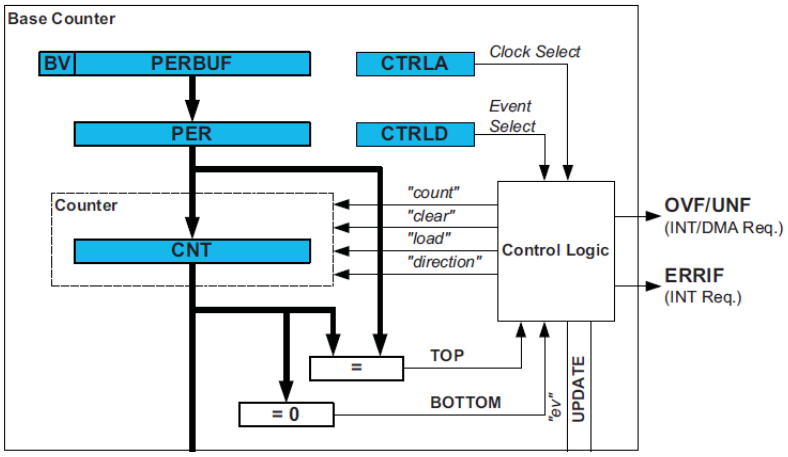


Figura 8 Diagrama de bloques del TIMER0. Tomado de: 8-bit Atmel XMEGA B Microcontroller.pdf Sección 13

4.3 Registros de configuración del TIMER

El ATXMEGA128B1 tiene 35 registros para la configuración y uso de cada TIMER tipo 0, para la función de conteo se usan 6 de estos registros.

Registro CTRLA: Este registro define la pre-escalización del reloj principal de los periféricos. Por defecto la frecuencia del reloj principal de los periféricos (CLK_{PER}) es 2MHz, al escoger una pre-escalización, la frecuencia principal se divide por el valor de pre-escalización y la frecuencia resultante es con la cual trabaja el TIMER.

La

Figura 9 Registro CTRLA del TIMER0. Tomado de: 8-bit Atmel XMEGA B Microcontroller.pdf Sección 13 presenta la descripción de este registro, incluida en la hoja de especificaciones del microcontrolador.

Bit	7	6	5	4	3	2	1	0
+0x00	–	–	–	–	CLKSEL[3:0]			
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:4 – Reserved**

These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.

- **Bit 3:0 – CLKSEL[3:0]: Clock Select**

These bits select the clock source for the timer/counter according to Table 13-3.

CLKSEL=0001 must be set to ensure a correct output from the waveform generator when the hi-res extension is enabled.

Table 13-3. Clock select options

CLKSEL[3:0]	Group Configuration	Description
0000	OFF	None (i.e, timer/counter in OFF state)
0001	DIV1	Prescaler: Clk
0010	DIV2	Prescaler: Clk/2
0011	DIV4	Prescaler: Clk/4
0100	DIV8	Prescaler: Clk/8
0101	DIV64	Prescaler: Clk/64
0110	DIV256	Prescaler: Clk/256
0111	DIV1024	Prescaler: Clk/1024
1nnn	EVCHn	Event channel n, n= [0,...,7]

Figura 9 Registro CTRLA del TIMER0. Tomado de: 8-bit Atmel XMEGA B Microcontroller.pdf Sección 13

En esta descripción se ve que los 4 bits menos significativos de este registro (CLKSEL[3:0]) son los que se deben escribir para cambiar la pre-escalización. Escribir 0000 en estos bits equivale a apagar el TIMER. Si se escribe 0100, se está seleccionando una pre-escalización de 8. Por lo tanto la frecuencia de conteo del TIMER será de 2Mhz/8, es decir de 250KHz. Una frecuencia de 250KHz corresponde a un ciclo de reloj de 1/250KHz, es decir 4µs. El resultado es que el contador hará un conteo cada 4 µs.

En total se tiene 7 opciones de pre-escalización. La última opción, 1nnn, se usa para otros fines y no se debe tener en cuenta en la función de conteo.

Registros PERH y PERL: Estos registros almacenan el valor hasta el cual debe llegar el registro CNT antes de levantar la bandera. Cada registro es de 8 bits, entre los dos tienen 16 bits. Por lo tanto, el máximo valor es 0xFFFF, en decimal esto es 65535. Al escribir PER en el código se incluye los dos registros PERH y PERL.

Registros CNTH y CNTL: Estos registros llevan el valor del conteo. Una vez el valor en estos registros llega al valor almacenado en los registros PER, el conteo se reinicia en 0. Cada registro es de 8 bits, entre los dos tienen 16 bits. Por lo tanto, el máximo valor al que pueden llegar es 0xFFFF, en decimal esto es 65535. Al escribir CNT en el código se incluye los dos registros CNTH y CNTL.

Registro INTFLAGS: Este registro guarda las banderas generadas por el TIMER. Cuando CNT alcanza el valor de PER pone en 1 la bandera OVFIF (bit 0 del registro INTFLAGS). Una vez se lea está bandera se debe "limpiar", poner en 0 la bandera. Para esto se escribe un 1 en el bit 0 del registro INTFLAGS, es decir se escribe un 1[1] en la bandera OVFIF.

Bit	7	6	5	4	3	2	1	0
+0x0C	CCDIF	CCCIF	CCBIF	CCAIF	–	–	ERRIF	OVFIF
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figura 10 Registro INTFLAGS del TIMER0. Tomado de: 8-bit Atmel XMEGA B Microcontroller.pdf Sección 13

Ejemplo numérico de la configuración de un temporizador

Se requiere que el TIMER TCC0 cuente un tiempo de 0.5 segundos. Esto significa que se debe configurar el TIMER para que contar desde 0 hasta el valor en PER le tome 0.5 segundos.

Modificamos la ecuación (3) de la sección 4.1 Funcionamiento de un temporizadorpara ajustarla a los nombres de los registros del TIMER tipo 0 del microcontrolador ATXMEGA128B1, la ecuación resultante se presenta en (4). En esta ecuación se introduce la división del reloj por la pre-escalización.

$$PER = T * \frac{CLK_{per}}{Pre - escalización} \qquad (4)$$

Para este ejemplo, el valor de *T* es 0.5 segundos, el valor de CLK_{PER} es 2MHZ, y se tiene dos incógnitas: PER y preescalización.

[1] No es un error tipográfico. Efectivamente para que la bandera baje a 0 se debe escribir un 1 en el bit.

De acuerdo a la descripción de los registros, PER puede tener cualquier valor entre 0 y 65535; La preescalización puede ser: 1, 2, 4, 8, 64, 256 o 1024. Se debe seleccionar cualquier combinación de PER y pre-escalización que satisfaga la ecuación.

La combinación prescalización = 64 y PER = 15625 la satisface.

4.4 Ejemplo aplicado de temporizadores

Se quiere alternar el encendido y el apagado de los cuatro LEDS de la tarjeta AVR XMEGAB1 XPLAINED, cada medio segundo.

Paso 1. Los 4 LEDS de la tarjeta están conectados a los pines PB4, PB5, PB6 y PB7. Se deben configurar estos pines como salida.

Paso 2. Inicializar los valores de los pines PB4 a PB7. Como los LEDS son activos en bajo, si se desea que inicien en bajo se escribirá un 1.

Paso 3. Seleccionar el TIMER que se va a usar y configurarlo para un tiempo de conteo de medio segundo. Se usan los valores obtenido en el ejemplo numérico anterior.

Paso 4. Incluir un bucle infinito para que se pregunte recurrentemente por la bandera del TIMER.

Paso 5. Preguntar por la bandera del TIMER. Si está en un 1 significa que ya se cumplió 0.5 segundos, entonces se alterna el valor de los LEDs y se “limpia” la bandera.

El programa final es el presentado en Código 10.

```
#include <avr/io.h>
#define LEDPORT PORTB
int main( void )
{
  LEDPORT.DIRSET = 0xF0;        // Paso 1. Definición de salidas.
  LEDPORT.OUTCLR = 0xF0;        // Paso 2. Inicia los 4 LEDs apagados

  /* Paso 3. Configuración del TIMER*/
  TCC0.PER = 0x3D09; //El periodo se establece en 0x3D09 en decimal:15625
  TCC0.CTRLA = 0x05;  // Selección de preescalización : DIV64

  while (1)   // Paso 4. Bucle infinito
  {
    /* Paso 5 */
    if((TCC0.INTFLAGS & 0x01) != 0)   //Revisa si la bandera de overflow (OVFIF) está en uno
    {
      TCC0.INTFLAGS = TCC0.INTFLAGS | 0x01;   //Limpia la bandera escribiendo un 1
      LEDPORT.OUTTGL = 0xf0;                  // Alterna el valor de los 4 LEDs
    }
  }
}
```

Código 10 Programa ejemplo aplicado TIMER

Ejercicios

I. Lea y entienda la descripción del bit OVFIF del registro INTFLAGS del timer0, que se encuentra en la sección 13.12 de la hoja de especificaciones “8-bit Atmel XMEGA B Microcontroller.pdf”

II. Investigue la aplicación de la directiva #define y deduzca cuál es el beneficio de usarla en el Código 10.

III. Escriba las instrucciones que configuran el TIMER TCC0 para que levante la bandera cada segundo

IV. Escriba las instrucciones que configuran el TIMER TCE0 para que levante la bandera cada 100ms

V. ¿Cuál es el máximo tiempo (en segundos) que puede contar el TIMER tipo 0?

VI. Una de las aplicaciones de los temporizadores es la de antirrebote por programa. El antirrebote por programa consiste en preguntar por el estado del pulsador, solamente cada cierto tiempo, para filtrar el rebote del pulsador. Complete el programa diseñado en el ejercicio I de la sección 3.5, para que sólo se pregunte por el estado del pulsador 8 veces por segundo, es decir cada 125ms.

VII. Escriba, compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el código del ejercicio anterior. Monte el circuito que permite probar el código desarrollado, no incluya el circuito antirrebote que usa Schmitt trigger.

5. Interfaz serial asíncrona

5.1 Comunicación serial entre dos dispositivos

Una comunicación serial se basa en la transmisión de unos y ceros entre dos o más dispositivos. En una transmisión serial los bits se envían uno a la vez y de forma consecutiva, por esto se llama serial, es decir que en el canal de transmisión habrá un solo bit a la vez. Para el caso de una comunicación alámbrica, sólo se necesita un hilo para establecer un canal de comunicación serial.

En una comunicación entre dos dispositivos, si existe un solo canal, los dispositivos deben alternarse para realizar el uso de este canal, es decir un dispositivo transmite y el otro recibe, esta comunicación se denomina *Half Dúplex*. También se puede establecer dos canales, uno en cada dirección, de tal forma que por un canal el primer dispositivo transmite y el segundo dispositivo recibe, mientras que por el segundo canal el primer dispositivo recibe y el segundo transmite. Esta comunicación denominada *Full Dúplex,* permite que cada dispositivo pueda transmitir y recibir al mismo tiempo.

Debido a que la comunicación es serial, los dispositivos que se están comunicando deben conocer las especificaciones de la señal que recibirán para poder interpretar correctamente la información recibida.

Por ejemplo: el dispositivo A debe enviar al dispositivo B el número 58. Como la comunicación es binaria deberá enviar “111010” que corresponde a 58. En esta comunicación surgen varios problemas esenciales para que los dos dispositivos se puedan entender:

1. Si se quiere enviar un número mayor a 58 se deberá usar más bits, si se quiere enviar un número menor se usarán menos bits, entonces, ¿Cómo sabe el dispositivo B cuántos bits contiene el dato? Se debe establecer una longitud fija del símbolo a transmitir. Si se establece que la longitud es de 8 bits, el dispositivo A deberá enviar “00111010” para enviar el número 58.
2. Cuando no hay un mensaje, el canal se mantiene en un valor constante (0 o 1), entonces, ¿Cómo sabe el dispositivo B que el mensaje ya inició si el primer bit que le envía A es igual al valor constante (0 o 1)?. Es necesario establecer un protocolo de comunicación: Al inicio de cada mensaje se envía un encabezado predeterminado, de esta forma el dispositivo B sabe que va a iniciar un mensaje. El envío del encabezado lo realiza el periférico USART, por lo tanto no es necesario realizarlo por *software*.
3. Cuando hay dos 0 seguidos o dos 1 seguidos, ¿Cómo sabe el dispositivo B que son dos bits diferentes? Hay varias formas de dar solución a este problema según el protocolo de comunicación que se esté usando, pero lo esencial de este punto es que los dos dispositivos deben manejar la misma frecuencia de transmisión/recepción. Una solución es agregar un hilo adicional por el cual se transmite una señal de reloj, la cual se utiliza para sincronizar la transmisión y la recepción, en este caso se habla de una comunicación serial síncrona. Otra solución es configurar una frecuencia de transmisión igual en los dos dispositivos (F_{BAUD}), la cual se expresa en Baudios (número de símbolos por segundo). En este caso se habla de una

comunicación asíncrona. Velocidades típicas en las interfaces seriales de microcontroladores son: 2400 bauds, 4800 bauds, 9600 bauds.

4. Es común que en el canal de comunicación un dato se altere debido al ruido, por lo tanto las comunicaciones seriales incluyen un bit de paridad el cual permite realizar una verificación de errores a la llegada. Esta verificación la realiza el periférico USART, por lo que no se debe realizar por *software*.
5. Por último, ¿Cómo sabe el dispositivo B que el mensaje ya terminó?, la respuesta es: cuando reciba 8 bits sabrá que ha terminado, sin embrago para mayor seguridad, se puede enviar un noveno bit denominado bit de parada.

La USART (Universal Synchronous/Asynchronous Receiver Transmitter) es un periférico que realiza la comunicación serial en un microcontrolador. A través de este periférico el microcontrolador puede comunicarse con un computador o con otro microcontrolador. La USART soporta una comunicación *Full-Duplex*, y puede operar en modo asíncrono o síncrono.

5.2 USART del ATXMEGA128B1

El microcontrolador ATXmega128B1 tiene dos USART: USARTC0 y USARTE0, asociados al PORTC y al PORTE respectivamente. Sin embargo, en la tarjeta de desarrollo XMEGAB1 xplained, sólo se debe usar la USARTC0 del PORTC, ya que en el puerto E se encuentran conectados los pulsadores capacitivos, y estos generarían ruido en la comunicación. El pin 2 del PORTC (PC2) tiene la función de recepción (RXD0), el pin 3 del PORTC (PC3) tiene la función de transmisión (TXD0).

La comunicación usando la USARTC0 se compone de los siguientes datos:

- 1 bit de inicio
- 5,6,7,8 o 9 bits de datos
- Sin paridad, bit de paridad par o bit de paridad impar
- 1 o 2 bits de parada

La frecuencia de comunicación (FBAUD) es generada por el *Fractional baud rate generator*. Está determinada por el reloj de los periféricos (FPER), el periodo (BSEL) y por un factor de escala (BSCALE) de acuerdo a las ecuaciones de la Figura 11. En estas ecuaciones se presenta la máxima F_{BAUD} de acuerdo a la frecuencia de reloj de los periféricos. BSEL puede tomar cualquier valor entre 0 y 4095, BSCALE puede tomar cualquier valor entre -7 y 7, sin embargo, dependiendo de si este valor es negativo o positivo se debe usar una ecuación diferente.

Operating Mode	Conditions	Baud Rate[1] Calculation	BSEL Value Calculation
Asynchronous normal speed mode (CLK2X = 0)	BSCALE ≥ 0 $f_{BAUD} \le \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} \cdot 16(BSEL+1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE} \cdot 16 f_{BAUD}} - 1$
	BSCALE < 0 $f_{BAUD} \le \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{16((2^{BSCALE} \cdot BSEL)+1)}$	$BSEL = \frac{1}{2^{BSCALE}}\left(\frac{f_{PER}}{16 f_{BAUD}} - 1\right)$
Asynchronous double speed mode (CLK2X = 1)	BSCALE ≥ 0 $f_{BAUD} \le \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} \cdot 8 \cdot (BSEL+1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE} \cdot 8 f_{BAUD}} - 1$
	BSCALE < 0 $f_{BAUD} \le \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{8((2^{BSCALE} \cdot BSEL)+1)}$	$BSEL = \frac{1}{2^{BSCALE}}\left(\frac{f_{PER}}{8 f_{BAUD}} - 1\right)$
Synchronous and master SPI mode	$f_{BAUD} < \frac{f_{PER}}{2}$	$f_{BAUD} = \frac{f_{PER}}{2 \cdot (BSEL+1)}$	$BSEL = \frac{f_{PER}}{2 f_{BAUD}} - 1$

Figura 11 Ecuaciones para calcular la frecuencia de comunicación. Tomado de: 8-bit Atmel XMEGA B Microcontroller.pdf

Ejemplo numérico de la frecuencia de comunicación

Se quiere configurar una frecuencia de transmisión de 9600 Bauds para establecer una comunicación serial asíncrona, a partir de un reloj de periférico de 2 MHz.

A partir de los valores de f_{PER} (2 000 000) y f_{BAUD} (9600) se ve que se cumple la relación $f_{BAUD} \le \frac{f_{PER}}{16}$.

Se decide usar un valor de BSCALE positivo por lo tanto se usará la ecuación (5) para calcular los valores de BSCALE y BSEL. En esta ecuación hay dos incógnitas, por lo tanto, se debe fijar uno de los valores.

$$F_{BAUD} = \frac{F_{PER}}{2^{BSCALE} * 16.(BSEL+1)} \qquad (5)$$

La Tabla 2 presenta los valores de BSEL para los cada posible valor de BSCALE.

Tabla 2 Configuración de la frecuencia de comunicación de la USART

BSCALE	0	1	2	3	4	5	6	7
BSEL	12.02	5.51	2.25	0.62	-0.18	-0.59	-0.79	-0.89

De la tabla 2 se concluye que BSCALE debe estar entre 0 y 3, para los otros valores el valor de BSEL sería negativo, dejándolo fuera del rango permitido (0 a 4095). BSEL deber ser un número entero, se observa que en todos los casos el número obtenido es decimal, por lo tanto, se deberá aproximar el valor de BSEL. Para BSCALE = 0, se logra la mejor aproximación, de 12.02 a 12, entonces se selecciona esta opción.

5.3 Registros de configuración de la USART

El ATXMEGA128B1 tiene 7 registros para la configuración de la USART, los 6 registros usados para la configuración de una comunicación serial asíncrona manejada por medio de *polling*, se explican a continuación:

Registro CTRLC: Este registro define las características generales de la comunicación serial: síncrona o asíncrona, bit de paridad, bits de parada y longitud del símbolo. En la página 279 de la hoja de especificaciones del microcontrolador [1] se puede ver el detalle de este registro.

Registros BAUDCTRLA y BAUDCTRLB: Los 4 bits más significativos del registro BAUDCTRLB almacenan el valor de BSCALE, los 4 bits menos significativos de BAUDCTRLB y los 8 bits de BAUDCTRLA almacenan el valor de BSEL.

Registro CTRLB: Este registro permite habilitar o deshabilitar la recepción y la transmisión de la USART y doblar la transmisión de la USART.

Bit	7	6	5	4	3	2	1	0
+0x04	–	–	–	RXEN	TXEN	CLK2X	MPCM	TXB8
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figura 12 Registro CTRLB del periférico USART

Bit 4 – RXEN: Al poner este bit en 1 se habilita la recepción de la USART

Bit 3 – TXEN: Al poner este bit en 1 se habilita la transmisión de la USART

Bit 2 -CLK2X: Al poner este bit en 1 se dobla la f_{BAUD} en una comunicación asíncrona

Bit 1 y 0: Estos bits no se usan en el desarrollo de esta guía, pero no se deben modificar durante la escritura de los bits 7 a 4.

Registro DATA: Este registro se compone de dos subregistros, TXB y RXB. En TXB se escribe el dato que se quiere transmitir al otro dispositivo. En RXB se lee el dato recibido del otro dispositivo.

Para escribir en TXB se debe escribir en USARTC0.DATA. Por ejemplo, la instrucción `USARTC0.DATA = 58;` envía el número decimal 58 al otro dispositivo a través de la USARTC0.

Para leer RXB, es decir lo que otro dispositivo haya enviado al microcontrolador se debe leer USARTC0.DATA. Por ejemplo, la instrucción `Temp = USARTC0.DATA;` almacena en la varia temp, el valor recibido a través de la USARTC0.

Registro STATUS: Este registro guarda las banderas generadas por la USART.

Bit	7	6	5	4	3	2	1	0
+0x01	RXCIF	TXCIF	DREIF	FERR	BUFOVF	PERR	–	RXB8
Read/Write	R	R/W	R	R	R	R	R	R/W
Initial Value	0	0	1	0	0	0	0	0

Figura 13 Registro STATUS del periférico USART

Bit 7 – RXCIF: Esta bandera se pone en 1 cuando se ha recibido un dato a través de la USART, el cual se podrá leer en USARTC0.DATA. Cuando este dato se lee en USARTC0.DATA, la bandera se limpia automáticamente.

Bit 6 – TXCIF: Esta bandera se pone en 1 cuando se ha terminado de enviar los datos pendientes en la USART. Esta bandera se limpia escribiendo un 1 en el bit.

Bit 5 – DREIF: Esta bandera se pone en 1 cuando el subregistro TXB está listo para recibir un nuevo dato. Al escribir un nuevo dato en USARTC0.DATA, esta bandera se limpia automáticamente.

Bit 4 a Bit 0: Estos bits no se usan en el desarrollo de esta guía, pero no se deben modificar durante la escritura de los bits 7, 6 y 5

Para ver los detalles de cada registro se debe revisar la sección 21.15 (pag. 276) del Manual: 8-bit Atmel XMEGA B Microcontroller [1].

5.4 Ejemplo aplicado de comunicación serial asíncrona

En este ejemplo se implementa un programa sencillo en el cual el microcontrolador espera que otro dispositivo envíe la letra “N” para responder con un 0, en la siguiente recepción de “N” enviará un 1 y así sucesivamente hasta 20.

Observe cada una de las líneas del Código 11 Comunicación serial asíncrona. Se declaran 3 variables, 2 de tipo entero de 8 bits, i llevará el índice de respuesta y Tx almacenará el dato a transmitir. La tercera variable es de tipo char, en la cual se almacenará el dato recibido para luego compararlo con “N”.

Los pines de transmisión y recepción se deben declarar como salida y como entrada, respectivamente. El pin 3 del PORTC es el pin de transmisión y el pin 2 es el pin de recepción.

A continuación, se configura la USART en modo asíncrono, con longitud de símbolo de 8 bits, sin paridad y con un Stop bit. Para esto, en el registro USARTC0.CTRLC se escribe:

- *Bits 7:6 – CMODE[1:0]= 00.* Estos dos bits se ponen en 00 para seleccionar modo asíncrono.
- *Bits 5:4 – PMODE[1:0] = 00*. Estos dos bits se ponen en 00 para deshabilitar la paridad.
- *Bit 3 – SBMODE=0.* Este bit se pone en 0 para elegir un bit de parada al final de la transmisión.
- *Bit 2:0 – CHSIZE[2:0]=001* . Con este valor se selecciona un tamaño de símbolo de 8 bits.

Se configura el registro BUADCTRLA para tener una frecuencia de transmisión de 9600 Bauds, usando los valores calculados en el ejemplo numérico de la subsección anterior.

En el loop principal se pregunta si llegó un dato. Si llegó, se guarda el dato en la variable Rx_Buf. Se compara el valor que llegó con la letra “N”. Si es igual se pregunta si la USART está lista para enviar y se envía el valor de i. i se aumenta en 1 o se reinicia a 0 cuando llegue a 20.

```
/* ------------------ Inclusion of Std Headers ------------------------ */
#include <avr/io.h>

/* -------------------- Declaration of Data --------------------------- */

#define USARTPORT PORTC

/* ---------------------------- Main ---------------------------------- */
int main(void)
{
  /* ------------------ variable definition ------------------------ */
    uint8_t i=0;
    uint8_t Tx_Buf = 0;
    char Rx_Buf = 0;

  /* ------------------ USART Peripheral configuration ------------------------ */
    USARTPORT.DIRSET   = 0x08;   // Pin 3 (TXD0) as output.
    USARTPORT.DIRCLR   = 0x04;   // Pin 2 (RXD0) as input.

    USARTC0.CTRLC = 0x03;       // Asynchronous Mode, 8 Data bits, No Parity, 1 Stop bit
    USARTC0.BAUDCTRLA = 12;   //BSEL=12 and BSCALE =0, to have a transmission frequency of 9600 Bauds

    USARTC0.CTRLB |= 0x10;  //Enables USART reception
    USARTC0.CTRLB |= 0x08;  //Enables USART transmission

    /* ------------------ Infinite loop ------------------------ */
    while(1)
    {
       // Asks if a data was received
       if((USARTC0.STATUS & 0x80) != 0 )        //if bit 7 (RXIF) is set, a data was received
       {
           Rx_Buf = USARTC0.DATA;               // Read out the received data
           if(Rx_Buf=='N')
           {
                Tx_Buf = i;
                // Transmit a character, first wait that the USART is ready
                if( (USARTC0.STATUS & 0x20) != 0 ) // if bit  (DREIF) is set, The USART is ready to send a new data
                {
                    USARTC0.DATA = Tx_Buf;
                }
                (i<20) ? (i++):(i=0);  //if i is 19 it returns to 0
            }
       }
    }
}
```

Código 11 Comunicación serial asíncrona

Ejercicios

I. Cree un código que al presionar un pulsador envíe a través de la USARTC0 la letra “N”, y que al recibir un número entre 0 y 15, lo muestre de forma binaria en los 4 LEDs de la tarjeta de desarrollo XMEGAB1 Xplained, donde un LED encendido corresponde a un 1 y un LED apagado a un 0.

II. Compile y programe el código creado en el ejercicio anterior en la tarjeta de desarrollo XMEGAB1 xplained, en **otra tarjeta** compile y programe el Código 11. Interconecte los siguientes

pines entre las dos tarjetas: TX1-RX2, RX1-TX2, GND1-GND2. Donde los números 1 corresponden a una tarjeta y los 2 a la otra. Pruebe que el montaje funciona de acuerdo a lo desarrollado en los dos códigos.

5.5 Comunicación con un computador

Conversor serial

Actualmente la interfaz serial más usada en computadores es la USB (*Universal Serial Bus*). Este periférico no usa el mismo protocolo de comunicación que la USART, por lo tanto, para establecer una comunicación entre la USART de un microcontrolador y el USB de un computador, es necesario usar un **conversor USB a serial RS232 PL2303** como el que se ve en la Figura 16. Al conectar este conversor a un puerto USB del computador, se debe verificar en el administrador de dispositivos que el PC lo reconoce como un puerto (Figura 14). Tome nota del puerto al cual queda conectado. Si el PC no encuentra automáticamente el driver, este se debe descargar del siguiente link:

http://www.prolific.com.tw/US/ShowProduct.aspx?p_id=225&pcid=41

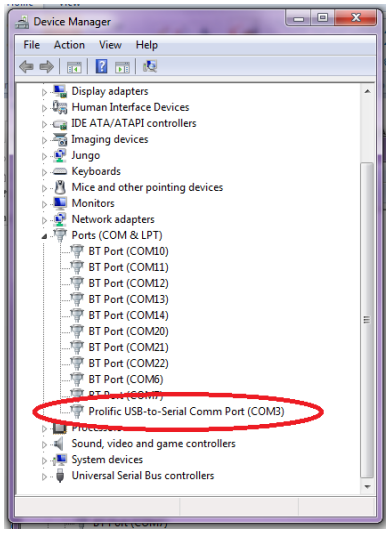


Figura 14 Driver conversor serial USB-USART

Terminal serial

Un programa de Terminal Serial permite enviar y recibir datos directamente a través de un puerto USB, algunos programas conocidos de terminal serial son: hyperterminal, *RealTerm*, PuTTY, entre otros. Se recomienda que el programa que se escoja tenga la posibilidad de enviar números hexadecimales, y de

configurar los parámetros de la comunicación como número del puerto, frecuencia de comunicación, tamaño del símbolo, bit de paridad y bit de parada.

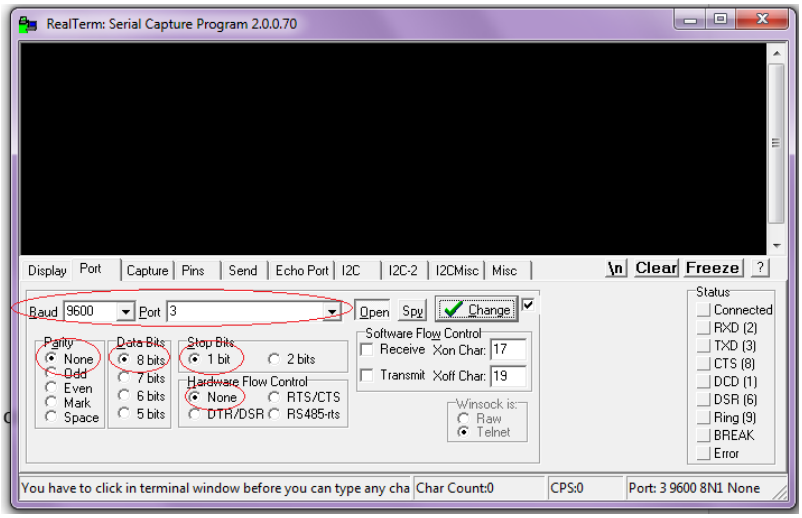


Figura 15 Configuración de la comunicación en RealTerm

La Figura 15, muestra la configuración de la comunicación en el programa RealTerm:

- Baud de 9600
- Puerto en el cual se encuentre el dispositivo. El puerto seleccionado es el 3, sin embargo, este puerto variará en cada computador. Debe revisar el Administrador de dispositivos para identificar el puerto.
- Paridad: ninguna.
- Bits del dato: 8 bits.
- Bit de parada: 1.
- Sin control de flujo por Hardware.

La configuración de los parámetros de comunicación debe coincidir con la configuración realizada en la USART del microcontrolador.

En RealTerm se debe dar click en “Change” para que los nuevos parámetros sean tomados.

Conexión entre PC y XMEGAB1 Xplained

La Figura 16 muestra el esquema de conexión entre el conversor serial y la tarjeta de desarrollo XMEGAB1 Xplained. El pin de recepción de la USART (PC2) se conecta al pin de transmisión del conversor, el pin de transmisión de la USART (PC3) se conecta al pin de recepción del conversor, finalmente se conectan las tierras de las dos tarjetas. **No se debe conectar los pines de alimentación, ni 5V, ni 3.3V**. **Esto causaría un daño en las dos tarjetas.**

Envío y recepción de datos

Una vez realizada la conexión entre dispositivos y la configuración de los parámetros de comunicación en el Terminal serial y en el microcontrolador, se puede hacer el envío y recepción de datos en el terminal serial.

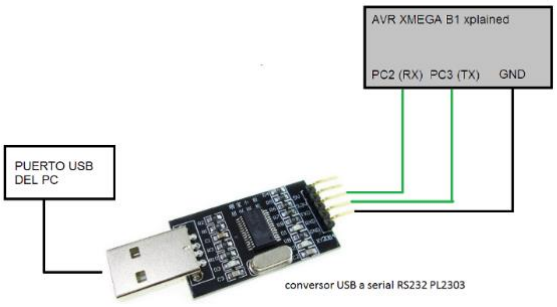


Figura 16 Circuito de comunicación serial entre la tarjeta XMEGAB1 Xplained y un PC

Cada terminal serial tiene una manera diferente de hacer el envío y la recepción. En la Figura 17, se muestra el envío en RealTerm. Para este programa se debe escribir el dato a enviar, en el ejemplo se ve una “N” y luego dar click en “Send Numbers” si quiere enviarse como un número decimal o “Send ASCII” si se quiere enviar como un carácter “ASCII”.

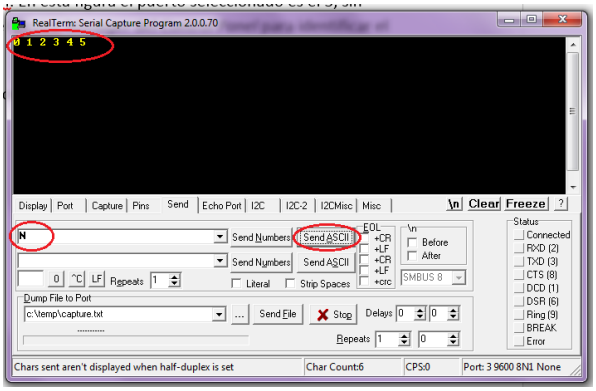


Figura 17 Envío y recepción de datos en RealTerm

Lo que se reciba desde el microcontrolador se verá en la consola (pantalla negra). En RealTerm el formato de visualización en consola puede ser elegido entre binario, decimal, hexadecimal o ASCII (Figura 18).

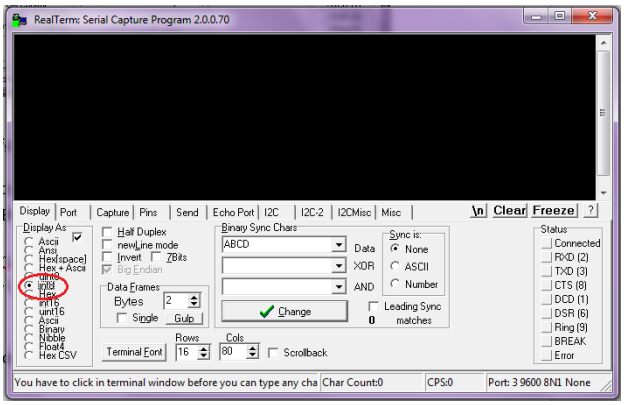


Figura 18 Formato de visualización en RealTerm

Ejercicios

I. Compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el Código 11 presentado en la sección 5.4 Ejemplo aplicado de comunicación serial asíncrona. Conecte la tarjeta de desarrollo a un computador siguiendo los pasos vistos en esta sección. Envíe una N desde el terminal serial, como respuesta debe recibir un 0. Si vuelve a enviar una N debe recibir un 1 y así sucesivamente hasta 20. Debe enviar la N en ASCII ya que el microcontrolador está esperando un char.

II. Compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el código creado en el ejercicio 5.4.I. Pruebe su código conectando la tarjeta de desarrollo a un computador. Debe recibir una “N” en el terminal serial, cada vez que presione un pulsador. Al enviar un número del 0 al 15 desde el terminal serial al computador, este número debe representarse en binario en los LEDs de la tarjeta.

6. Modulador por ancho de pulso (PWM)

6.1 Señal generada por el Modulador por Ancho de Pulso

En muchas aplicaciones es necesario generar valores análogos en las salidas del microcontrolador, sin embargo, la naturaleza digital del procesador sólo nos permite tener dos posibles valores a la salida: 0 (gnd) o 1 (Vcc). La técnica de modulación por ancho de pulso, (PWM por sus siglas en inglés) permite generar valores análogos usando únicamente 0 y 1.

La señal generada por el PWM tiene como base un tren de pulsos digitales con un periodo constante, pero con un ciclo útil variable, es decir que el ancho de pulso de la señal aumenta o disminuye.

La Figura 19 muestra 4 señales con el mismo periodo, pero con diferentes ciclos útiles. A menor ciclo útil el ancho de pulso será menor. El ciclo útil está dado por la ecuación (6).

$$DutyCycle\ (\%) = \frac{PulseWidth\ (s) * 100}{Period\ (s)} \qquad (6)$$

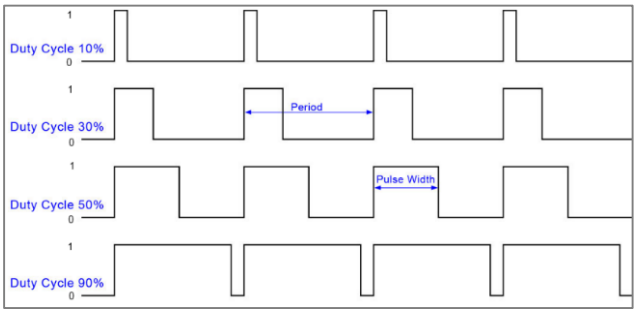


Figura 19 Señales con el mismo periodo y diferente ancho de pulso

Al controlar un actuador usando una señal PWM como las de la Figura 19, éste funcionará de acuerdo al valor eficaz de la señal. Por ejemplo si Vcc es 3,3 V y la señal tiene un ciclo útil del 10%, el actuador funcionará como si estuviera controlado por una señal constante de 330 mV. Pero si el ciclo útil es del 50%, el actuador funcionará como si recibiera una señal constante de 1.65 V, y así sucesivamente para los otros ciclos útiles [10]. De esta forma, se puede considerar que para el actuador el microcontrolador está produciendo un valor análogo a su salida.

La frecuencia de la señal PWM debe ser lo suficientemente alta con respecto a la tasa de respuesta del actuador para que éste sí funcione con el valor eficaz y no vea los voltajes 0 y Vcc de forma diferenciada.

Una señal PWM puede ser usada en varias aplicaciones, control de motores, control de LEDs RGB, cargadores fotovoltaicos etc.

6.2 PWM en el ATXMEGA128B1

En el microcontrolador ATXMEGA128B1 se genera una señal PWM usando un *TIMER/COUNTER* tipo 0 en modo **comparación**. En este modo el *counter* usa un contador base para contar ciclos de reloj y 4 canales de comparación (CCx) con los cuales compara el contador base. Cada canal de comparación tiene un pin asociado (PINx) en el cual se verá la señal PWM generada como producto de la comparación.

El ATXMEGA128B1 tiene dos *counters* tipo 0: TCC0 y TCE0, asociados al PORTC y al PORTE respectivamente. En esta guía se usará exclusivamente TCE0, sin embargo la misma teoría aplica al TCC0.

Funcionamiento del counter tipo 0 en modo comparación

El funcionamiento del comparador es el siguiente:

- El contador (registro CNT) inicia en 0 y aumenta en 1 su valor con cada ciclo de reloj.
- El reloj base para realizar el conteo es el reloj de los periféricos CLK_{per}, pre-escalizado.
- El PINx habilitado para entregar la señal PWM inicia en 1.
- Cuando el contador llega al valor comparación (CCx), el PINx baja a 0.
- El contador sigue contando (aumentando en 1 su valor con cada ciclo de reloj)
- Cuando el contador llega al valor del periodo (PER) reinicia a 0 y el PINx vuelve a 1. A partir de aquí se inicia nuevamente el ciclo.

La Figura 20 ilustra el funcionamiento del contador para generar una señal PWM en el PINx.

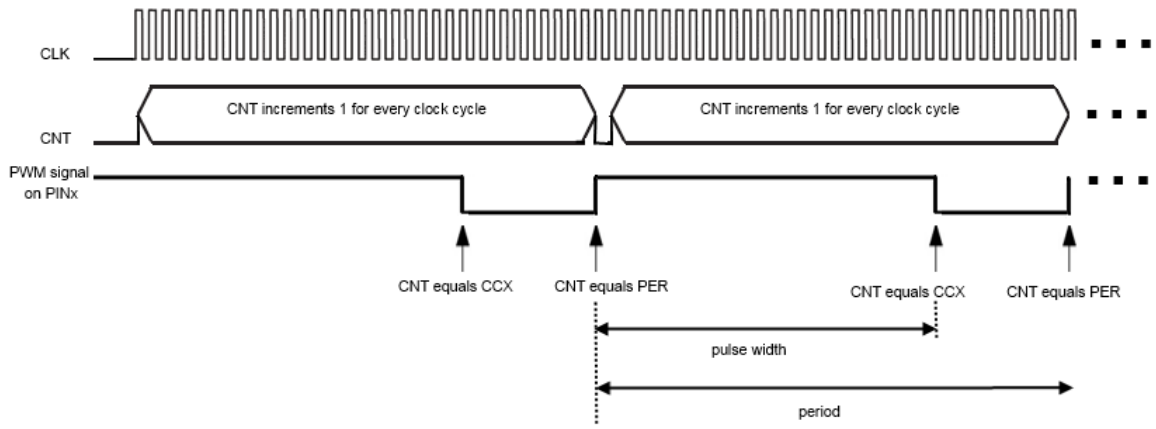


Figura 20 Generación de una señal PWM con el ATXMEGA128B1

Como se puede observar en la Figura 20, el periodo de la señal PWM es definido por el registro PER, mientras que CCx define el ciclo útil de la señal. CCx se almacena en el registro CCxBUF.

El counter tipo 0 tiene 4 canales de comparación (A, B, C y D), asociados a los pines 0, 1, 2 y 3 del puerto, respectivamente. Para el counter TCE0, los registros CCABUF, CCBBUF, CCCBUF y CCDBUF, definen los

ciclos útiles de las señales PWM de los pines PE0, PE1, PE2 y PE3. Sin embargo, hay un solo registro PER que define el periodo para los cuatro canales. Todas las señales PWM generadas tendrán el mismo periodo, pero pueden tener diferente ciclo útil.

Cálculo del periodo y del ciclo útil

El periodo de la señal PWM se calcula teniendo en cuenta la frecuencia de reloj del microcontrolador, la pre-escalización del conteo y el valor máximo de conteo (registro TCE0.PER). La ecuación (7) muestra la relación entre estas variables.

$$frec_{PWM} = frec_{reloj} \div preescalización \div TCE0.PER \qquad (7)$$

Ejemplo: Se desea tener una señal PWM con frecuencia 500Hz, a partir de un reloj de 2MHz. Se selecciona una preescalización de 1, el valor de TCE0.PER es: $TCE0.PER = 2000000 \div 1 \div 500 = 4000$

El ciclo útil de la señal está definido por el valor en el registro TCE0.CCxBUF de acuerdo a la ecuación (8).

$$TCE0.CCxBUF = TCE0.PER * ciclo_{útil} \div 100 \qquad (8)$$

Ejemplo: Se desea un ciclo útil del 75% en el canal A para la señal de 500 Hz del ejemplo anterior, el valor de TCE0.CCABUF es: $TCE0.CCABUF = 4000 * 75 \div 100 = 3000$

6.3 Registros de configuración del PWM

Los siguientes registros del TIMER/COUNTER 0 se usarán para la configuración de una señal PWM. Para ver los detalles de cada registro revisar la sección 13.12 (pag. 161) del Manual: 8-bit Atmel XMEGA B Microcontroller.

Registro CTRLA : Permite configurar la preescalización del reloj en 1, 2, 4, 8, 64, 256 o 1024.

Registro CTRLB : Permite configurar,

- El modo de la generación de onda. Se selecciona “Single-slope PWM”
- La habilitación de los canales A, B, C y D.

Registro PER: En este registro se almacena el valor del periodo. Este registro es de 16 bits, por lo tanto el máximo valor que puede contener es 65536

Registros CCABUF, CCBBUF, CCCBUF y CCDBUF: Estos registros almacenan los valores CCxBUF que definen los ciclos útiles. Estos registros son de 16 bits, por lo tanto el máximo valor que pueden contener es 65536.

Código de ejemplo

En Código 12 se configura tres canales PWM en los pines 0, 1 y 2 del PORTE. La frecuencia de la señal es de 500Hz y el ciclo útil de cada canal es 75%, 50% y 25% para los canales A, B y C respectivamente.

Lea el código, identifique y entienda cada una de las instrucciones de acuerdo a lo explicado en el punto anterior. Para entender mejor el código debe consultar la descripción de cada registro en la sección 13.12 (pag. 161) del Manual: ATxmega128B1 / ATxmega64B1.

```
#include <avr/io.h>
#define PWM PORTE

int main( void )
{
    PWM.DIRSET = 0x07;  //Define los pines 0, 1 y 2 como salidas
    PWM.OUTSET = 0x07;  //Pone en 0 los pines 0, 1 y 2
    //PORTE.PIN0CTRL |= 0x40; //Si la aplicación requiere invertir la señal en el pin

    TCE0.CTRLB |= 0x03;  //Selecciona el modo SINGLESLOPE para la generación de la señal
    TCE0.CTRLB |= 0x70; //Habilita los canales A, B y C al poner en 1 el bit CCxEN correspondiente.

    TCE0.CTRLA |= 0x01; //Define la preescalización del conteo a 1.
    TCE0.PER = 4000; //Periodo = frecuenciaCPU ÷ preescalización ÷ frecuenciaPWM; frecuenciaPWM=500Hz

    TCE0.CCABUF = 3000; //Define el ancho de pulso del canal A. CCABUF=periodo*ciclo_util÷100;Ciclo_util=75%
    TCE0.CCBBUF = 2000; //Define el ancho de pulso del canal B. Ciclo_util=50%
    TCE0.CCCBUF = 1000; //Define el ancho de pulso del canal C. Ciclo_util=25%

    while(1)
    {
    }
}
```

Código 12 Configuración de tres señales PWM con diferente ciclo útil

En el Código 12 se puede observar dos hechos importantes.

1. Los pines donde se tendrán las señales PWM deben declararse como salida.
2. En el while(1) no es necesario escribir ningún código. Las señales se generarán en los pines a partir de la configuración hecha antes del while(1). Esto se debe a que el TIMER/COUNTER es un periférico que funciona de forma independiente al procesador, por lo tanto realizará el conteo y las comparaciones de manera autónoma. Se escribirá código dentro del while(1) si se desea variar el ciclo útil o el periodo durante la ejecución del programa.

6.4 Ejemplo aplicado: Control de velocidad y dirección de un motor DC

Un motor DC tiene dos terminales, al aplicar un voltaje DC entre estos dos terminales el motor gira en un sentido a una velocidad proporcional al valor de voltaje aplicado sobre los terminales. Al invertir la polaridad del voltaje entre los terminales del motor DC, éste gira en el sentido contrario. Para poder controlar el sentido de giro de un motor DC se usa un puente H.

En este ejemplo aplicado se usará un *driver* DRV8833 *Dual Motor Driver Carrier*, el cual tiene un puente H, el motor *Micro Metal Gearmotor y* la tarjeta de desarrollo AVRXMEGA B1. La Figura 21 muestra el circuito de conexión entre estos tres elementos. La especificación de corriente del motor es 360mA, esta es una corriente que el regulador de la tarjeta XMEGAB1 xplained puede suministrar.

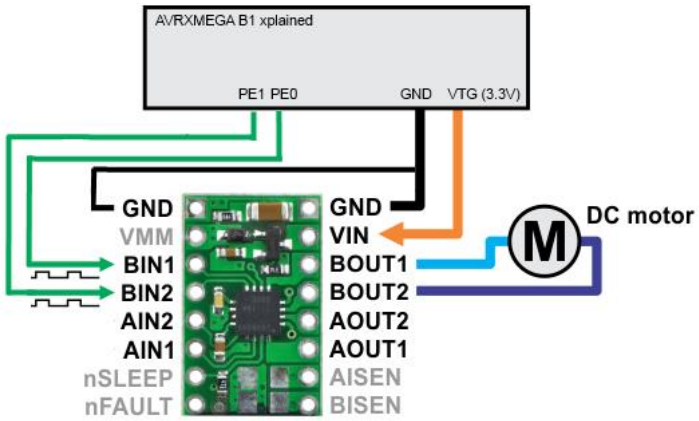


Figura 21 Circuito de control de un motor DC. Tomado y adaptado de [9]

Por seguridad la tarjeta se puede conectar a una toma de 110v a través de un adaptador de voltaje, en lugar de conectarla al puerto USB del computador. Se debe revisar que el adaptador tenga una especificación de corriente de al menos 800mA. Si se usa un motor de mayor potencia se deberá revisar tanto la especificación del regulador de la tarjeta como la especificación del adaptador de voltaje.

Variación de la velocidad y del sentido de giro del motor

La lógica de funcionamiento del DRV8833 se presenta en la Figura 22. Para que el motor gire, una de las entradas debe estar en 0, mientras que en la otra debe haber una señal PWM. El sentido del giro dependerá de la entrada en la cual se aplique la señal PWM; la velocidad del motor dependerá del ciclo útil.

BIN1	BIN2	BOUT1	BOUT2	FUNCTION
0	0	Z	Z	Coast/fast decay
0	PWM	L	PWM	Reverse
PWM	0	PWM	L	Forward
1	1	L	L	Brake/slow decay

Figura 22 Lógica del DRV8833. Tomado y adaptado de [9]

Ejercicios

I. Use el Código 12. Modifique el valor del registro TCE0.CCABUF para que el ciclo útil sea del 0% y del registro TCE0.CCBBUF para que el ciclo útil sea del 15%. Compile y programe la tarjeta. Debe observar que el motor gira lentamente en un sentido.

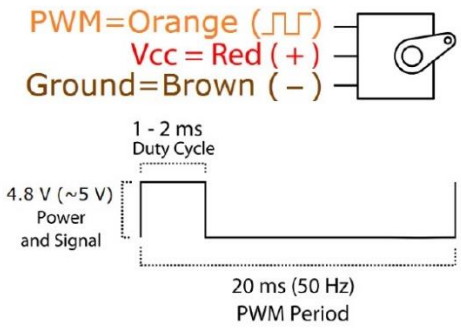
II. Invierta los valores de TCE0.CCABUF y TCE0.CCBBUF. Compile y programe la tarjeta. Debe observar que el motor gira a la misma velocidad, pero en sentido contrario.

III. Manteniendo siempre una de las dos señales en 0% modifique el ciclo útil de la otra señal y observe como la velocidad del motor cambia.

6.5 Ejemplo aplicado: Control de posición de un Servomotor

Los servomotores no tienen un giro continuo como los motores DC. Van de una posición a otra dependiendo de una señal de control.

Los servomotores tienen tres entradas: tierra, fuente y control (PWM). Su alimentación es por lo general entre 5V y 7V. En este ejemplo aplicado se usará el servomotor *Tower Pro micro servo 9G*, el cual se alimenta a 5v. Los pines de alimentación y la señal de control requerida para este servomotor se muestran en la Figura 23.



Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

Figura 23 Tower Pro micro servo 9g. Tomado de [15]

El pin *VCC* debe conectarse a V_BUS de la tarjeta de desarrollo XMEGAB1 Xplained en el conector J3, el pin Ground debe conectarse a la tierra de la tarjeta de desarrollo en cualquier conector. El pin PWM debe conectarse al pin PE0 del conector J4.

El microcontrolador debe suministrar al servomotor una señal PWM con periodo de 20 ms (50Hz), y un ancho de pulso variable de acuerdo con la posición a la cual se quiera llevar el motor. Un ancho de pulso de 1.5 ms pondrá el motor en la posición “0”, en el centro. Un ancho de pulso de 2ms pondrá el motor en la posición “90”, totalmente hacia la derecha. Un ancho de pulso de 1ms pondrá el motor en la

posición “-90”, totalmente hacia la izquierda. Un ancho de pulso entre 1ms a 2ms, pondrá al servo en la posición equivalente, entre -90° y 90°.

Teniendo en cuenta que el periodo de la señal de control es de 20ms, un ancho de pulso de 1,5ms es equivalente a un ciclo útil de 7.5%, un ancho de pulso de 2ms es equivalente a un ciclo útil de 10% y un ancho de pulso de 1ms es equivalente a un ciclo útil del 5%.

Ejercicios

I. Cree un proyecto con el código que se presenta en Código 13. Observe que la preescalización y el valor en PER se definieron tal que se obtiene una señal de periodo 50Hz. El registro CCABUF se configuró para tener un ciclo útil del 7,5%. Lo cual corresponde a la posición “0”, en el centro. Compile y programe la tarjeta. Debe observar que el servomotor se mueve a la posición 0. Si ya estaba en esa posición no hará nada.

II. Modifique el valor de CCABUF para tener un ciclo útil del 10%. Compile y programe, el motor se debe mover 90° hacia la derecha. Si no alcanza a llegar a 90° aumenta poco a poco el ciclo útil hasta alcanzar 90°.

III. Modifique el valor de CCABUF para tener un ciclo útil del 5%. Compile y programe, el motor se debe mover -90° hacia la izquierda. Si no alcanza a llegar a -90° disminuya poco a poco el ciclo útil hasta alcanzar -90°.

```
#include <avr/io.h>
#define PWM PORTE

int main( void )
{
        PWM.DIRSET = 0x01;      //Define el pin 0 como salida
        PWM.OUTSET = 0x01;      //Pone en 0 el pin 0

        TCE0.CTRLB |= 0x03;     //Selecciona el modo SINGLESLOPE para la generación de la señal
        TCE0.CTRLB |= 0x10;     //Habilita el canal A al poner en 1 el bit CCAEN.
        TCE0.CTRLA |= 0x04;     //Define la preescalización en 8.
        TCE0.PER = 5000;        //Periodo = frecuenciaCPU ÷ preescalización ÷ frecuenciaPWM;
                                //frecuenciaPWM=50Hz
        TCE0.CCABUF = 400;      //Define el ancho de pulso del canal A. CCABUF=periodo*ciclo_util÷100;
                                //Ciclo_util=7,5%
        while(1)
        {
        }
}
```

Código 13 Control de un servomotor

6.6. Ejemplo aplicado: Control de un LED RGB

Las señales PWM se pueden usar para controlar el color de un LED RGB, el cual se compone de 3 leds, uno rojo, uno verde y uno azul. Estos tres colores corresponden a los colores primarios del espectro electromagnético visible, a partir de los cuales se pueden obtener otros colores. Para entender como una señal PWM puede generar un color cualquiera se debe entender el concepto de círculo cromático.

Círculo cromático RGB

La Figura 24 muestra el círculo cromático RGB en el cual se distribuyen linealmente el rojo, el azul y el verde para formar colores secundarios y colores terciarios.

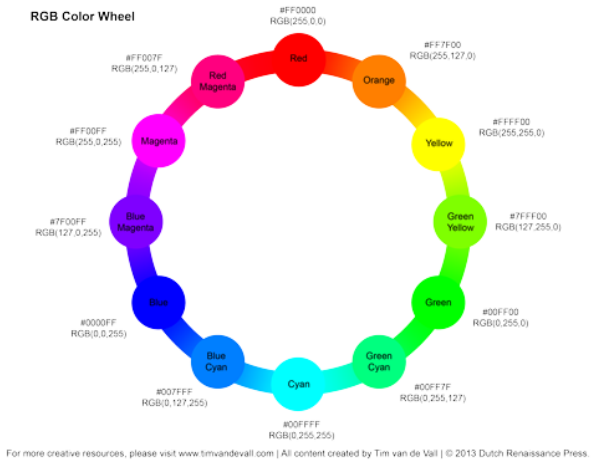


Figura 24 Círculo cromático. Tomado de [14]

Los colores secundarios y terciarios son el resultado de sumar los tres colores primarios con una intensidad lumínica específica para cada color. En el círculo cromático RGB esta intensidad lumínica se divide en 255 niveles, donde 0 es ninguna intensidad lumínica (color apagado) y 255 la máxima intensidad lumínica. Relacionando estos niveles con la señal PWM, 0 corresponde a un ciclo útil del 0% y 255 a un ciclo útil del 100%.

Por ejemplo, si se desea obtener el color *Cyan* {RGB (0,255,255)} , el led rojo debe tener un ciclo útil del 0%, el led verde de 100% y el led azul de 100%. Por otro lado si se desea obtener el color naranja {RGB(255,127,0)} , el led rojo debe tener un ciclo útil del 100%, el led verde de 50% y el led azul de 0%.

El círculo cromático se divide en 12 secciones, en cada una de las cuales sólo varia un color a la vez:

- Entre *Red* y *Yellow* aumenta la intensidad del led verde de 0 a 255.
- Entre *Yellow* y *Green* disminuye la intensidad del led rojo de 255 a 0.
- Entre *Green* y *Cyan* aumenta la intensidad del led azul de 0 a 255.

- Entre *Cyan* y *Blue* disminuye la intensidad el led verde de 255 a 0.
- Entre *Blue* y *Magenta* aumenta la intensidad del led rojo de 0 a 255.
- Entre *Magenta* y *Red* disminuye la intensidad del led azul de 255 a 0.

Circuito para controlar un LED RGB

El voltaje de encendido de los LEDs varía de acuerdo a su color. Los valores típicos de encendido de un LED RGB en encapsulado de 5mm transparente son: 2 voltios para el rojo, y 3.4 voltios para el verde y el azul [13].

El microcontrolador de la tarjeta Atmel Xmega B1 Xplained se encuentra alimentado a 3.3 voltios, por lo tanto las señales PWM generadas tendrán una amplitud máxima de 3.3 voltios, la cual no cumple con el voltaje de encendido del verde y del azul; por lo tanto se debe usar un transistor el cual permite usar una fuente de alimentación de 5 voltios para encender el LED a la vez que se usa la señal PWM como control de encendido y apagado del transistor. La Figura 25 muestra el circuito que se debe implementar para controlar un LED RGB de ánodo común.

Calcule las resistencias que van en serie con cada color del LED. Para realizar este cálculo use la ecuación (9), donde $Voltaje_{color}$ corresponde al voltaje de encendido del LED de cada color, $Voltaje_{fuente}$ es igual a 5 V y $Corriente_{LED}$ es el valor máximo de corriente que se quiere que pase por el LED, un valor típico de corriente es 10mA.

$$Rserie = \frac{Voltaje_{fuente} - Voltaje_{color}}{Corriente_{LED}} \qquad (9)$$

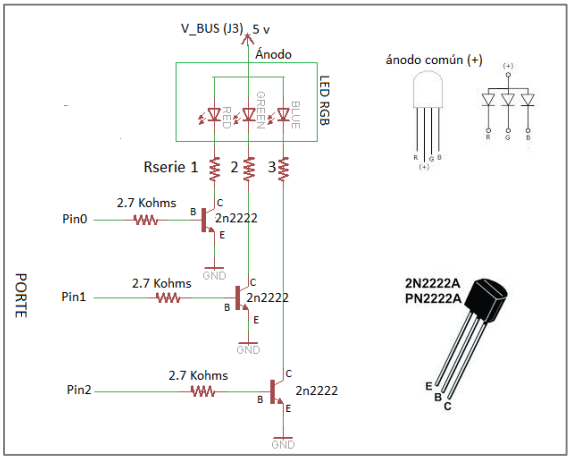


Figura 25 Circuito esquemático para controlar un LED RGB

Ejercicios

I. Cree un nuevo proyecto “GCC C Executable Project” con el Código 12. Modifique los valores de los registros TCE0.CCxBUF para cambiar los ciclos útiles en los pines PE0, PE1 y PE2 de tal forma que se vea el color Cyan en el LED RGB: el led rojo debe tener un ciclo útil del 0%, el led verde de

100% y el led azul de 100%. Compile y programe la tarjeta. Ponga una hoja blanca o un difusor de luz a unos centímetros del LED RGB y observe en el centro, donde se unen los tres colores, el color CYAN.

II. Cree un código que automáticamente recorra el círculo cromático, mostrando en el LED RGB todos los colores del círculo. Para controlar el tiempo que permanece en cada combinación de colores, use un temporizador. Compile y programe la tarjeta.

7. Conversor Análogo- Digital (ADC)

7.1 Adquisición de señales análogas

Muchas de las variables físicas que encontramos en el entorno son análogas, por ejemplo, la temperatura, la intensidad lumínica, la presión atmosférica, entre muchas otras. Estas señales pueden tomar infinitos valores y la variación de estos valores en el tiempo es continua. Diversas aplicaciones de sistemas embebidos se basan en la adquisición y análisis de estas señales, por lo tanto, los microcontroladores tienen la capacidad de recibir y procesar valores análogos.

En la Figura 26 se puede ver una señal análoga y una señal digital. El nombre de la señal análoga proviene del hecho de que esta señal es "análoga" a la señal física que representa. La magnitud puede tomar cualquier valor (en principio puede tomar infinitos valores) dentro de un rango específico, que puede ser positivo y negativo.

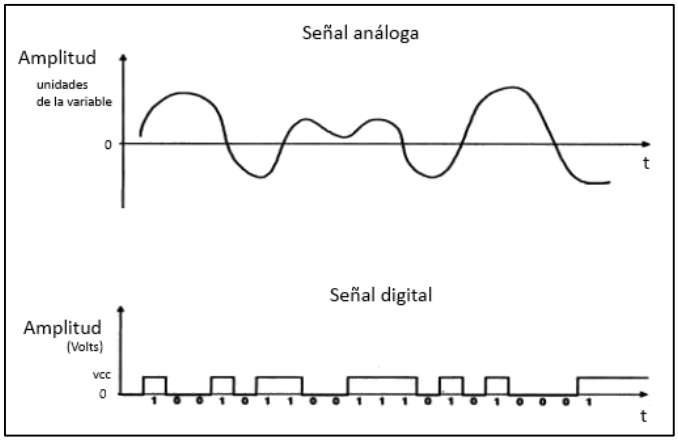


Figura 26 Señal Análoga vs señal digital

Para adquirir una variable física y procesarla en el microcontrolador, es necesario usar un transductor que convierta la magnitud de la variable física (°C, candelas, pascales, etc.) a un voltaje equivalente a esta magnitud.

Un transductor se caracteriza, entre otros, por:

- El rango máximo de entrada que está en la capacidad de medir.
- El rango máximo de voltaje que entregará a la salida.
- La función de transferencia entada-salida.

Un ejemplo de transductor, es el sensor de temperatura embarcado en la tarjeta de desarrollo AVR Xmega B1-Xplained. El sensor usa un termistor NTC de referencia NCP18WF104J03R, el cual varía su resistencia en función de la temperatura. Este termistor junto con un circuito de dos resistencias, alimentados entre fuente y tierra conforman el sensor de temperatura. El rango máximo de entrada de

este sensor es: -40°C a +125°C; el rango máximo de voltaje a la salida es: 1,047 V a 0,077 V; la función de transferencia es la que se muestra en la Figura 27, para valores de entrada entre -10°C y +50°C.

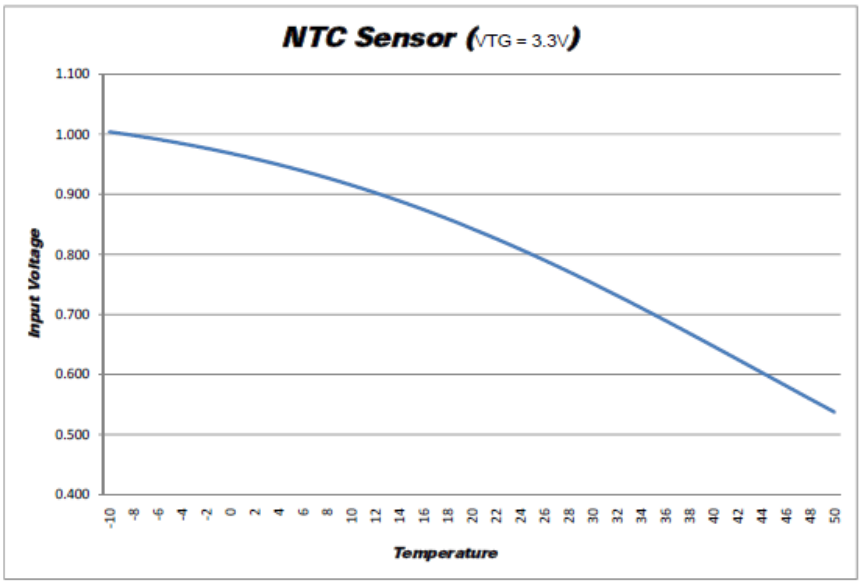


Figura 27 Temperatura vs Voltaje. Sensor NTC tarjeta AVR XMEGAB1 Xplained. Tomado de [3]

Como se puede observar en la Figura 27, a cada valor de temperatura le corresponde un valor de voltaje a la salida del transductor. Para el caso de este sensor de temperatura, su función de transferencia no es lineal, es una función exponencial determinada por las especificaciones del termistor.

7.2 Conversión análoga – digital

El ADC (Conversor Análogo Digital) es el periférico que adquiere el valor análogo en un pin de entrada del microcontrolador y lo convierte a un valor digital que pueda ser operado en el procesador.

La Figura 28 muestra la función de transferencia entrada - salida de un ADC. Esta función de transferencia es de tipo escalón debido a que hay un número limitado de bits para representar los posibles valores análogos. Entonces, es necesario dividir el rango de entrada en un número finito de subintervalos. A cada uno de estos subintervalos le corresponde un único valor digital de salida o código de salida.

Al número de subintervalos se le llama *la resolución* del ADC. Para el caso de la Figura 28 la resolución es de 8 subintervalos o de 3 bits (2^3=8).

Al tamaño de los subintervalos se le llama *cuantización*. La cuantización está definida por la ecuación (10) donde el *Voltaje de referencia* es el voltaje máximo en el rango de entrada.

$$Cuantización = \frac{Voltaje\ de\ referencia}{Resolución} = \frac{V_{ref}}{2^{\#bits}} \qquad (10)$$

En un ADC siempre es necesario especificar el voltaje de referencia, para que el ADC asigne el máximo código digital de salida a ese valor.

La cuantización permite determinar cuál será el código de salida por medio de la ecuación (11), donde los paréntesis cuadrados representan la parte entera.

$$Código\ de\ salida = \left[\frac{Voltaje\ de\ entrada}{Cuantización}\right] \qquad (11)$$

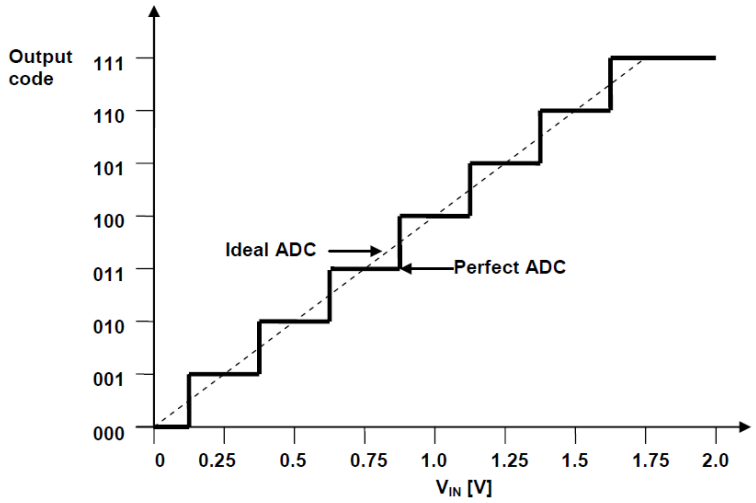


Figura 28 Función de transferencia de un ADC. Tomado de [7]

Ejemplo numérico para el código de salida en un ADC

En la Figura 28 se puede observar que el voltaje de referencia es igual a 2 V. Esto significa que si en la entrada se tiene 2 V a la salida se tendra el máximo código que permite la resolución, es decir 111.

Usando la ecuación (10) se puede calcular la cuantización del ADC: $Cuantización = \frac{2\ V}{2^3} = 250\ mV$.

Usando el valor de cuantización calculado y la ecuación (11) se puede determinar el código de salida para cualquier valor de voltaje de entrada.

Por ejemplo, si a la entrada se tiene un voltaje de 525 mV el código de salida es:

$$Código\ de\ salida = \left[\frac{525\ mV}{250\ mV}\right] = [2.1] = \ 2 = 0b010.$$

El código de salida se puede confirmar en la función de transferencia de la Figura 28.

7.3 Error y calibración

El ADC al redondear genera un error de *cuantización* (máximo $\pm 1/2$ subintervalo) el cual es inherente al proceso de conversión análogo-digital y por lo tanto no se puede corregir. Sin embargo, existen otras fuentes de error debidas al proceso de fabricación del circuito integrado: errores de desviación, errores de ganancia, no-linealidades diferenciales (DNL) e integrales (INL). Estos errores se pueden compensar usando aproximaciones polinomiales, para esto se deben realizar algoritmos de calibración y tablas de búsqueda. La nota de aplicación "*AVR120: Characterization and Calibration of the ADC on an AVR*" contiene información sobre como compensar los errores de ganancia y de desviación. Para aplicaciones con entrada unipolar no es necesario hacer este tipo de calibraciones.

Durante la producción cada ADC es calibrado para ajustar la escala de cuantización y mejorar la linealidad de la conversión. Los valores resultantes de esta calibración se almacenan en la memoria no volátil del microcontrolador. Para el caso de los microcontroladores de la familia XMEGA, se almacenan en la posición (ADCxCAL0/1). Antes de habilitar el ADC estos valores se deben leer desde esta posición y escribirse en los registros de calibración (CALL/CALH).

7.4 Velocidad de conversión

El ADC tarda un tiempo en el arranque, el muestreo y el mantenimiento para hacer la conversión de un dato. El **tiempo de arranque** es el tiempo necesario para garantizar una conversión correcta después de habilitar el ADC por primera vez. El **tiempo de muestreo y mantenimiento (tiempo de conversión)** es el tiempo que tarda el ADC en realizar la conversión. Se debe tener en cuenta que este tiempo haya transcurrido antes de leer el resultado.

Un ADC puede tener varios canales de entrada, pero sólo procesará y convertirá una entrada a la vez. Al hacer un cambio entre un canal y otro se debe esperar un tiempo adicional antes de iniciar una nueva conversión. Este tiempo se llama **tiempo de estabilización.**

A partir del tiempo de conversión y del reloj del ADC (CLK_{ADC}) se puede determinar el ancho de banda (máxima tasa de muestreo) de una señal análoga de entrada. En el caso de una conversión unipolar (referida a tierra), el tiempo de conversión es de 13 ciclos de reloj. Por lo tanto se pueden realizar N muestras por segundo de acuerdo a la siguiente ecuación: $N = CLK_{ADC} / 13$, siguiendo el teorema de Nyquist, el ancho de banda de la señal de entrada debe ser máximo la mitad de la tasa de muestreo. Por lo tanto, la máxima frecuencia de la señal análoga de entrada está limitada a $CLK_{ADC}/ 26$.

7.5 Registros de configuración del ADC

El microcontrolador ATXmega128B1 tiene dos ADC, uno asociado al PORTA (ADCA) y otro asociado al PORTB (ADCB), cada uno con resolución configurable de 8 bits o 12 bits. Cualquier pin del puerto correspondiente puede configurarse para que sea la entrada análoga del ADC.

Cada ADC tiene 21 registros de configuración y datos, sin embargo, para aplicaciones sencillas con entradas unipolares y resolución de 8 bits, se usan 7 de estos registros. Para conocer en mayor detalle el uso y configuración de todos los registros se puede consultar el manual "8-bit Atmel XMEGA B Microcontroller" [1], capítulos 26.15 y 26.16.

Registro CTRLB: Este registro permite configurar:

- La limitación de corriente, por defecto se configura en "No limit"
- El modo de conversión: "signed" si se va a trabajar con valores negativos. "unsigned" si sólo se va a trabajar con valores positivos.
- Freerun: En el modo "free running", una vez se termina una conversión, inicia una nueva conversión automáticamente.
- Resolución: se puede seleccionar resolución de 12 bits con alineación a la derecha, 12 bits con alineación a la izquierda y 8 bits.

Registro REFCTRL: Este registro permite realizar varias configuraciones, de las cuales la principal es el voltaje de referencia del ADC. Se puede seleccionar como referencia 1v, Vcc/1.6, Vcc/2, un voltaje externo en el pin AREF del puerto A o un voltaje externo en el pin AREF del puerto B.

Registro SAMPCTRL: Este registro permite aumentar el tiempo de muestreo del ADC. Cuando se usa un sensor con alta sensitividad se debe aumentar el tiempo de muestreo. Este es el caso del sensor de temperatura de la tarjeta AVRXMEGA B1.

Registro CH0.CTRL: Este registro permite:

- Configurar la ganancia del ADC. Por defecto se configura en ganancia 1x
- Iniciar una conversión (poniendo el bit 7 en 1)
- Configurar el modo de entrada: Unipolar (referido a tierra), interno, diferencial o diferencial con ganancia.

Registro CH0.MUXCTRL: Este registro permite configurar:

- La entrada positiva del ADC. Se puede seleccionar cualquiera de los pines del puerto o un voltaje interno.
- La entrada negativa del ADC. Esta se usa únicamente cuando el modo de entrada es diferencial.

Registro CTRLA: El bit 0 de este registro habilita el ADC.

Registro CH0.INTFLAGS: El bit 0 de este registro corresponde a la bandera del ADC. Cuando este bit se pone en 1 significa que el ADC terminó una conversión. Para "limpiar" la bandera se debe escribir un 1 en este mismo bit.

Registros CH0.RESL y CHO.RESH: En estos registros el ADC guarda el resultado de la conversión.

7.6 Pasos para hacer una conversión simple con el ADC

A continuación se listan los pasos para realizar una conversión simple:

1. Configurar como entrada el pin en el cual se va a medir el valor análogo.
2. Cargar los valores de calibración del ADC con el llamado de la función: ADC_CalibrationValues_Load(&ADCB)
3. Configurar el registro CTRLB en “no limit current, unsigned, no freerun, 8 bit resolution”
4. Configurar la referencia interna de voltaje en el registro REFCTRL.
5. Configurar el tiempo de muestreo en el registro SAMPCTRL.
6. Configurar el registro CH0.CTRL en “single ended, gain factor x1”.
7. Seleccionar el canal de entrada a través del registro CH0.MUXCTRL.
8. Habilitar el ADC a través del registro CTRLA.
9. Iniciar una conversión a través del registro CH0.CTRL.
10. Preguntar por la bandera de fin de conversión.
11. Leer el registro CH0.RESL para obtener el resultado de la conversión.

7.7 Ejemplo aplicado

La tarjeta de desarrollo XMEGAB1 Xplained tiene embarcados 3 sensores con salida análoga: Sensor de temperatura, sensor de giro (potenciómetro) y sensor de luz, conectados a PB0, PB1 y PB2 respectivamente.

En este ejemplo se implementa un sistema que lee continuamente el sensor de temperatura y el sensor de luz. De acuerdo con el valor de temperatura adquirido varía el ciclo útil de una señal PWM en el pin PE0, el cual debe estar conectado a un motor DC o a un ventilador; a mayor temperatura el motor gira más rápido. De acuerdo con el valor de intensidad lumínica adquirido, se enciende más o menos LEDs de la tarjeta, a menor intensidad lumínica prende más LEDs.

El ejemplo se diseñó usando una máquina de estados finitas (MEF) de dos estados, la cual se muestra en la Figura 29.

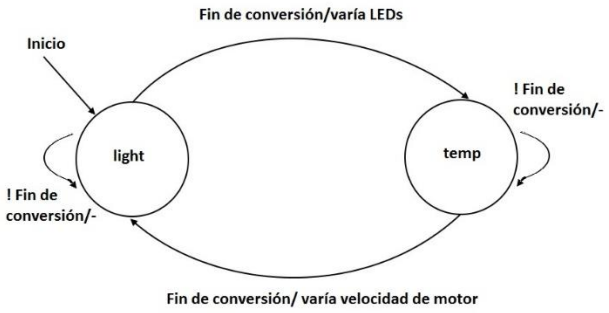


Figura 29 Máquina de estados finitos ejemplo ADC

El primer estado, *light*, corresponde a la adquisición del valor de luz; la máquina permanece en este estado mientras el ADC esté realizando la conversión del valor análogo entregado por el sensor de luz. Cuando la conversión termina se realiza la transición al segundo estado, *temp*; durante la transición se cambia el encendido o apagado de los LEDs, de acuerdo al valor entregado por el ADC.

El estado *temp*, corresponde a la adquisición del valor de temperatura; la máquina permanece en este estado mientras el ADC esté realizando la conversión del valor análogo entregado por el sensor de temperatura. Cuando la conversión termina se realiza la transición al estado *light*; durante la transición se cambia la velocidad del motor de acuerdo al valor entregado por el ADC.

El Código 14 muestra la implementación de este ejemplo. Lea el código, identifique y entienda cada una de las instrucciones de acuerdo a lo explicado en este capítulo.

El sensor de temperatura está conectado al pin 0 del puerto B (PB0) y el sensor de luz al pin 2 del puerto B (PB2), puesto que los dos se encuentran en el mismo puerto es necesario usar el ADCB y alternar la fuente de entrada a este ADC. En el código se observa que durante la transición de un estado al otro se cambia el canal del ADC y se inicia una nueva conversión.

Se escogió 1 V como valor de referencia del ADC, porque los valores que nos interesan de los dos sensores se encuentran por debajo de 1v.

Creación del proyecto para probar el ejemplo aplicado

1. Cree un nuevo proyecto de tipo “GCC C Executable Project” y nómbrelo como desee.
2. En el archivo principal .c del nuevo proyecto copie el Código 14.
3. Copie los siguientes archivos **en la misma carpeta de Windows** donde se encuentra el archivo principal .c del nuevo proyecto: adc_driver.c, adc_driver.h, y avr_compiler.h
4. Compile y programe la tarjeta.

```c
/* ------------------ Inclusion of Std Headers ------------------------ */
#include <avr/io.h>
#include "adc_driver.h"
#include "adc_driver.c"

/* --------------------- Declaration of Data --------------------------- */
#define LEDPORT PORTB
#define ADCPORT PORTB
#define PWM PORTE

typedef enum STATE STATE;
enum STATE { temp, light};//two states: one for temperature measures and the other for light measures

uint8_t temp_table[256] =
{125,125,125,125,125,124,123,122,121,120,119,118,117,116,115,114,113,112,111,110,110,109,108,108,107,106,
106,105,104,104,103,102,102,101,100,100,99,98,98,97,96,96,95,94,94,93,92,92,91,90,90,89,89,88,88,87,87,86
,86,85,85,84,83,83,82,81,80,80,79,79,78,78,77,77,76,76,75,75,74,74,73,73,72,72,71,71,70,70,69,69,68,68,68
,67,67,66,66,65,65,65,64,64,63,63,63,62,62,61,61,61,60,60,59,59,59,58,58,57,57,57,56,56,56,55,55,54,54,54
,53,53,53,52,52,51,51,51,50,50,50,49,49,49,48,48,48,47,47,47,46,46,46,45,45,44,44,43,43,43,42,42,42,41,41
,41,41,40,40,39,39,38,38,38,38,37,37,36,36,36,35,35,35,34,34,34,33,33,32,32,31,31,31,30,30,29,29,29,28,28
,27,27,27,26,26,25,25,25,24,24,23,23,22,22,22,21,21,20,20,19,19,18,18,17,17,16,16,15,14,14,13,13,12,11,11
,10,10,9,8,8,7,6,6,5,4,3,3,2,1,0,0,0,0,0,0,0,0,0,0};//Function table for the temperature sensor

/* ----------------------- Prototype of functions ---------------------- */
void ADC_result(uint8_t *res, STATE st);         //Converts the ADC result to duty cycle or LED sequence

/* --------------------------- Main ---------------------------------- */
int main(void)
{
    /* ------------------ variable definition ------------------------ */
     STATE state = light;                //state variable begins in light state
     uint8_t light_result =0;
     uint8_t temp_result =0;

    /* ------------------ Peripheral configuration ------------------------ */

     //IOPORTS
     ADCPORT.DIRCLR = 0x0F;      //Pin0 to pin3 of PORTB as inputs
     PWM.DIRSET = 0x03;  //Pin0 to pin 2 of PORTE as outputs
     PWM.OUTSET = 0x03;  //Pin0 to pin 2 to zero
     LEDPORT.DIRSET = 0xF0;      //Pin4 to pin7 of PORTB as outputs
     LEDPORT.OUTSET = 0xF0;      //4 leds OFF

     //ADC
     ADC_CalibrationValues_Load(&ADCB);  //Moves calibration values from memory to CALL/CALH.
     ADCB.CTRLB = 0x04;  //no limit current, unsigned, no freerun, 8 bit resolution
     ADCB.REFCTRL = 0x00;        //internal reference voltage of 1 volt
     ADCB.SAMPCTRL = 0x3F;       //Configurates the maximum sample time
     ADCB.CH0.CTRL=0x01 ;        //single ended, gain factor x1
     ADCB.CH0.MUXCTRL= 0x10;     //Selects pin 2 of PORTB as the ADC input (light sensor)
     ADCB.CTRLA |=0x01;  //Enables ADC

     //PWM
     TCE0.CTRLB |= 0x03;//Selects SINGLESLOPE for signal generation
     TCE0.CTRLB |= 0x30;//Enables channels A and B, by writing 1 to the CCxEN bit.
     TCE0.CTRLA |= 0x01;//Sets the prescaler to 1.
     TCE0.PER = 4000;            //Period = CPUfrequency ÷ prescaler ÷ PWMfrecuency; PWMfrecuency=500Hz
     TCE0.CCABUF = 0;            //Sets the Duty cycle for channel A. CCABUF=period*duty_cycle÷100;

     ADCB.CH0.CTRL|=0x80;        //Starts an ADC conversion

     //Continues next page…
```

Código 14 Ejemplo aplicado ADC

```
	/* ------------------ Infinite loop ----------------------- */


	while (1) {
	/* --------Finite state machine-------*/
	    switch (state){
	        case light:         //state 1: the system reads,converts and shows a light measure
	            if((ADCB.CH0.INTFLAGS & 0x01) !=0)       //Asks if the ADC conversion is complete
	            {
	                ADCB.CH0.INTFLAGS |= 0x01;    //Clears the ADC flag
	                ADC_result( &light_result, state);//gets the ADC conversion result and changes the LEDs
	                ADCB.CH0.MUXCTRL= 0x00;       //Selects pin 0 of PORTB as the ADC input (temperature sensor)
	                ADCB.CH0.CTRL|=0x80;          //Starts an ADC conversion
	                state = temp;                 //changes the state to temp
	            }
	        break;
	        case temp:          //state 2: the system reads, converts and shows a temperature measure
	            if((ADCB.CH0.INTFLAGS & 0x01) !=0)       //Asks if the ADC conversion is complete
	            {
	                ADCB.CH0.INTFLAGS |= 0x01;    //Clears the ADC flag
	                ADC_result( &temp_result, state);//gets the ADC conversion result and changes the DC
	                ADCB.CH0.MUXCTRL= 0x10;       //Selects pin 2 of PORTB as the ADC input (light sensor)
	                ADCB.CH0.CTRL|=0x80;          //Starts an ADC conversion
	                state = light;                //changes the state to light
	            }
	        break;
	    }
	}
}
/****************************** Local tasks ******************************/
*   void ADC_result(uint8_t *res, STATE st)
*   Purpose: converts the ADC result into duty cycle or into led sequence*/

void ADC_result(uint8_t *res, STATE st)
{
	/* The ADC reference is Vref= 1 volt and the resolution is 8 bits. This means that the quantization is equal
	to: Vref/2^resolution= 3,9 mV  */
	if(st==light)
	{
	    /* We have 4 LEDS and we want them to turn on as the light intensity decreases. We have 4 different
	    combinations: 1 LED ON, 2 LEDs ON, 3 LEDs ON and 4 LEDs ON. Then, we divide the ADC result into 4 segments,
	    and we assign a value to light_result (*res) according to the segment and to the LEDs we want to be ON*/

	    if ((ADCB.CH0.RESL>=0) && (ADCB.CH0.RESL<64))
	    { *res = 0x10;}
	    if ((ADCB.CH0.RESL>=64) && (ADCB.CH0.RESL<128))
	    { *res = 0x30;}
	    if ((ADCB.CH0.RESL>=128) && (ADCB.CH0.RESL<192))
	    { *res = 0x70;}
	    if ((ADCB.CH0.RESL>=192) && (ADCB.CH0.RESL<=255))
	    { *res = 0xF0;}
	    LEDPORT.OUT = ~*(res);    //Turns ON the LEDs according to the value on res
	}
	else
	{
	    /* when the result of the ADC is 0xFF (1 volt) it corresponds to a temperature of -8c (according to the
	    datasheet). When the result of the ADC is 0x00 (0 volt) it corresponds to a temperature of 125C (according
	    to the datasheet)*/
	    /* The transfer function is not linear, is exponencial, so we cannot calculate the degrees using a simple
	    formula. A simple way to solve the problem is to create an array with 256 elements, each one with the value
	    of the temperature for the corresponding value of the ADC result*/

	    *res =  temp_table[ADCB.CH0.RESL];
	    TCE0.CCABUF = (*res)*15;       //Sets the Duty cycle for channel A. Scales it so 125 corresponds to
	    100% of duty cycle.
	}
}
```

Código 15 Ejemplo aplicado ADC

Ejercicios

I. A partir de los esquemáticos de la tarjeta XMEGA B1 Xplained [4] y de la guía de usuario de la tarjeta [3], responda las siguientes preguntas:

   a. ¿En cuál pin del microcontrolador está conectado el potenciómetro embarcado en la tarjeta de desarrollo?

   b. ¿Qué valor de voltaje se tiene en el pin para Resistencia 0 en el potenciómetro y qué valor para resistencia 100kΩ?

   c. De acuerdo con la respuesta de la pregunta anterior, ¿cuál voltaje de referencia se debe seleccionar, entre los voltajes de referencia disponibles?

   d. ¿Cuál es la cuantización (en mV y Ω) de la conversión al usar resolución de 8 bits y el voltaje de referencia seleccionado en la pregunta anterior?

   e. Escriba la ecuación de la función de transferencia que permite pasar del valor binario obtenido a ohmios.

II. Modifique el Código 14 para incluir la lectura del potenciómetro, usando un tercer estado en la MEF y varíe el color de un LED RGB de acuerdo a la posición de giro del potenciómetro. Para la variación del color de un LED RGB revise la sección 6.6. Ejemplo aplicado: Control de un LED RGB.

III. Basándose en el Código 14, diseñe un programa que muestre en la pantalla LCD el valor de temperatura adquirido por medio del sensor de temperatura, cambie la intensidad de retroiluminación de la pantalla LCD de acuerdo a la intensidad lumínica del ambiente adquirida con el sensor de luz y encienda la barra de niveles del LCD, de manera proporcional a la posición de giro del potenciómetro. Para el uso de la pantalla LCD revise el Apéndice G. Pulsadores capacitivos y Pantalla de Cristal Líquido.

Apéndice A. Primeros pasos con la tarjeta de desarrollo Xmega B1 xplained

Para programar la tarjeta de desarrollo XMegaB1 Xplained, debemos descargar Atmel Studio, este programa es una interfaz de desarrollo (IDE) gratuita, que se puede descargar en la página oficial de Atmel.

La segunda herramienta que requerimos es un programador, en la Figura 30 se muestra una imagen de los tres programadores más populares para sistemas Atmel : AVRISP mkII (a), Atmel-ICE (b) y AVR Dragon (c).

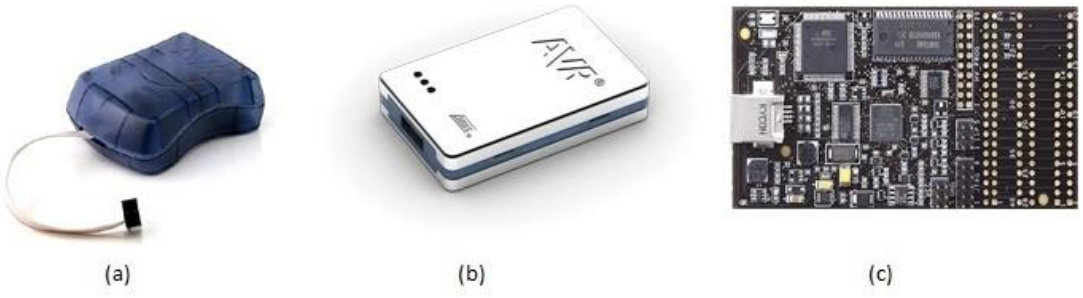


Figura 30 Programadores Atmel® (a) AVRISP mkII (b) Atmel-ICE (c) AVR Dragon.

A continuación, se presentan los pasos que se deben seguir para instalar el programador.

1. Conectar el programador a un puerto USB del computador donde se tiene instalado Atmel Studio.
2. Si el driver no se instala automáticamente se debe hacer una instalación manual. En el administrador de dispositivos de Windows, se elige la opción “agregar hardware heredado” (Figura 31 Instalación manual del programador. Paso 1). El programador no debe estar conectado al computador.
3. Seleccionar siguiente.
4. Seleccionar la instalación manual del hardware.
5. Seleccionar mostrar todos los dispositivos y siguiente.
6. Seleccionar usar disco.
7. Buscar el driver windrvr6, el cual se encontrará en la carpeta de instalación de Atmel. La ubicación precisa, dependerá de la versión instalada. En la Figura 32 se presenta una de las posibles ubicaciones.

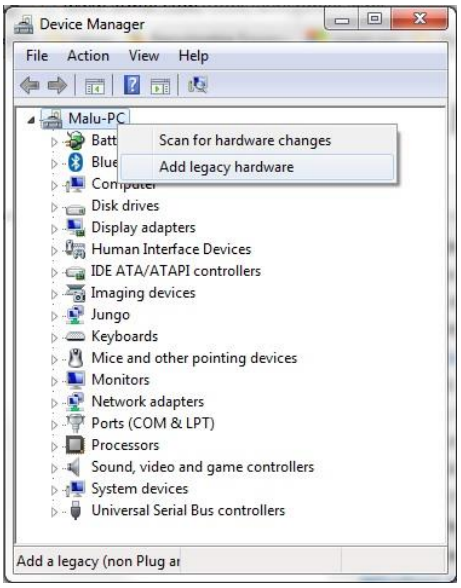


Figura 31 Instalación manual del programador. Paso 1

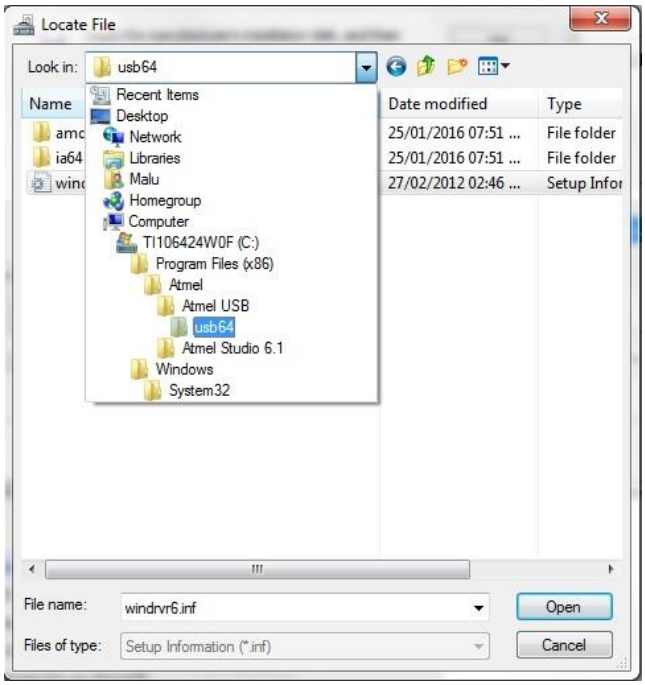


Figura 32 Ubicación del driver de instalación

8. Conectar el programador a un puerto del computador.
9. Verificar en el administrador de dispositivos, que los drivers “WinDriver” y el driver propio del programador, por ejemplo “AVRISP mkII”, se encuentren instalados bajo el menú “Jungo”. Figura 33.

10. Es común que se presenten problemas al instalar el programador. Se pueden encontrar respuestas a problemas específicos, en el foro de la comunidad Atmel: http://www.avrfreaks.net/

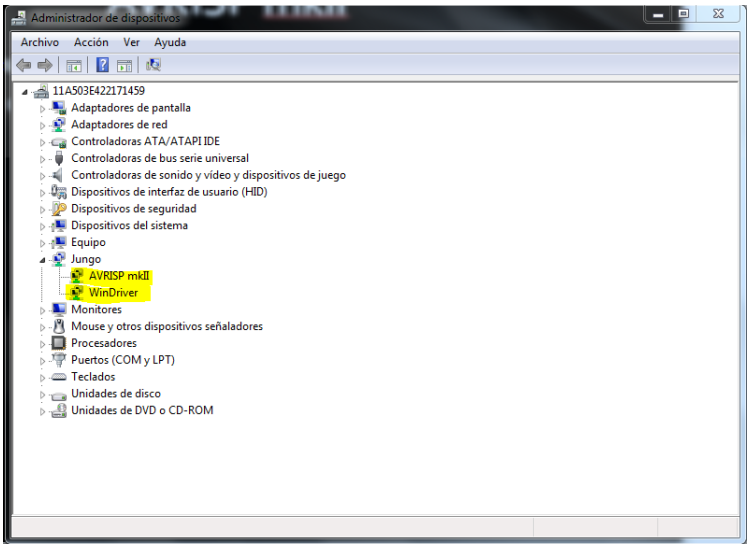


Figura 33 Drivers instalados

A continuación se presentan los pasos para programar un código demo en la tarjeta de desarrollo AVR XMEGA B1 Xplained.

1. Conectar el programador a un puerto USB del computador donde se tiene instalado Atmel Studio.
2. La tarjeta de desarrollo debe tener puesto un jumper en el conector J5 (VTG-VXM), encerrado en un círculo rojo en la Figura 34. No alimente la tarjeta si este jumper no está puesto.
3. Conectar la tarjeta de desarrollo a un puerto USB o a un adaptador USB, para darle alimentación eléctrica, y conectarla al programador usando el cable de cinta plana que viene incluido, como se muestra en la Figura 34. El conector de la cinta plana debe conectarse en el sentido correcto; en la Figura 34 se muestra el sentido que se debe usar para el programador Atmel-ICE; para el programador AVRISP MKII, el sentido correcto es con la pestaña del conector hacia la parte inferior de la tarjeta de desarrollo.

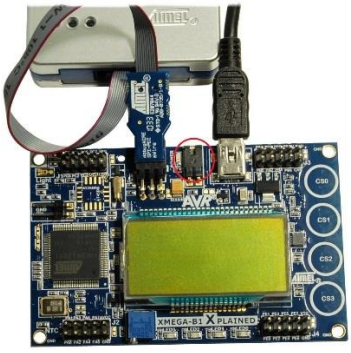


Figura 34 Conexión de la tarjeta de desarrollo XMEGAB1 Xplained

4. Abrir Atmel Studio y seleccionar File ->New Project…
5. Seleccionar GCC C Executable Project
6. Darle un nombre al Proyecto y luego seleccionar OK. Figura 35. Para este ejemplo usaremos el nombre “miproyecto”

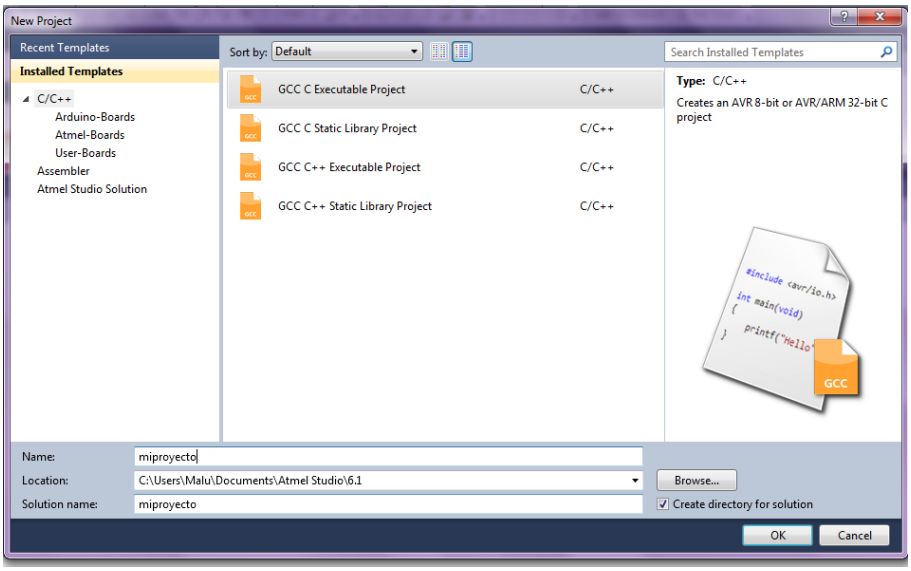


Figura 35 Crear un proyecto Atmel Studio

7. Seleccionar el microcontrolador ATxmega128B1 y luego seleccionar OK. Figura 36. En este momento se abre la venta principal del proyecto creado.

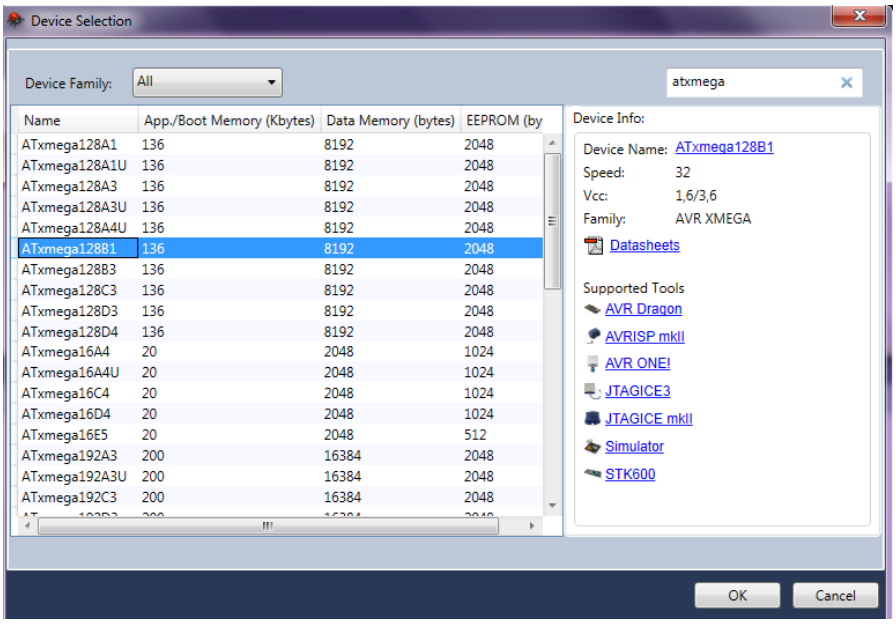


Figura 36 Selección del microcontrolador

8. Al lado derecho de la ventana, en el *Solution Explorer* dentro de la carpeta “miproyecto” se encuentra el archivo miproyecto.c . Presionar dos veces sobre este archivo para abrirlo. En este archivo se escribirá el programa. Figura 37

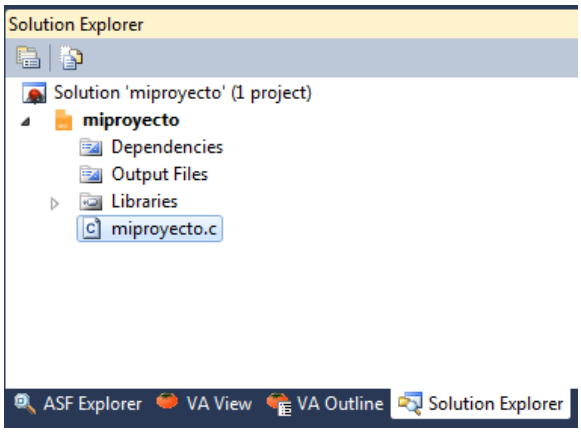


Figura 37 Solution Explorer

9. En el archivo miproyecto.c escribir el Código 16 miproyecto.c
10. Seleccionar Build->Build Solution o presionar F7 para construir el proyecto.

```
#include <avr/io.h>
#define LEDPORT PORTB
int main( void )
{
  LEDPORT.DIRSET = 0xf0;    // Los pines 4 a 7 como salida. No toca los demás.
  LEDPORT.OUTCLR = 0xf0; // Clr los 4 MSB. No toca los demás

  /*Reloj del sistema (por defecto: oscilador interno de 2MHz => periodo: 0,5 us)*/
  TCC0.PER = 0x3D09; //El periodo establece el valor TOP para el TIMER. Qué tan lejos va a contar.
  TCC0.CTRLA = 0x05;  // Selección de la fuente de reloj. DIV64, preescalización de 64

  while (1)
  {
    if((TCC0.INTFLAGS & 0x01) != 0)   //Revisa si la bandera de overflow (OVFIF) está en uno
    {
      TCC0.INTFLAGS = TCC0.INTFLAGS | 0x01;      //Limpia la bandera escribiendo un 1
      LEDPORT.OUTTGL = 0xf0; // Toggle los 4 MSB. No toca los demás
    }
  }
}
```

Código 16 miproyecto.c

11. Seleccionar Tools->Device Programming.

12. Seleccionar la opción adecuada en la lista *Tool* y *Device* como se muestra en la Figura 38 y presionar *Apply*.

13. Seleccionar *Memories*, verificar que en la casilla flash se encuentre el archivo miproyecto.elf y presionar *Program*. Figura 39

14. Debe aparecer una verificación de la programación de la tarjeta.

15. Observará que los 4 LEDS de la tarjeta se encienden y se apagan alternadamente con un periodo de un segundo.

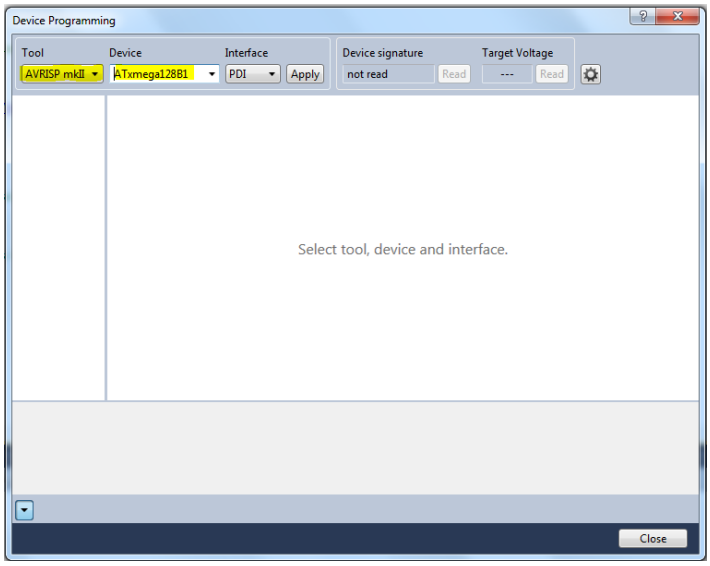


Figura 38 Selección de las herramientas para la programación.

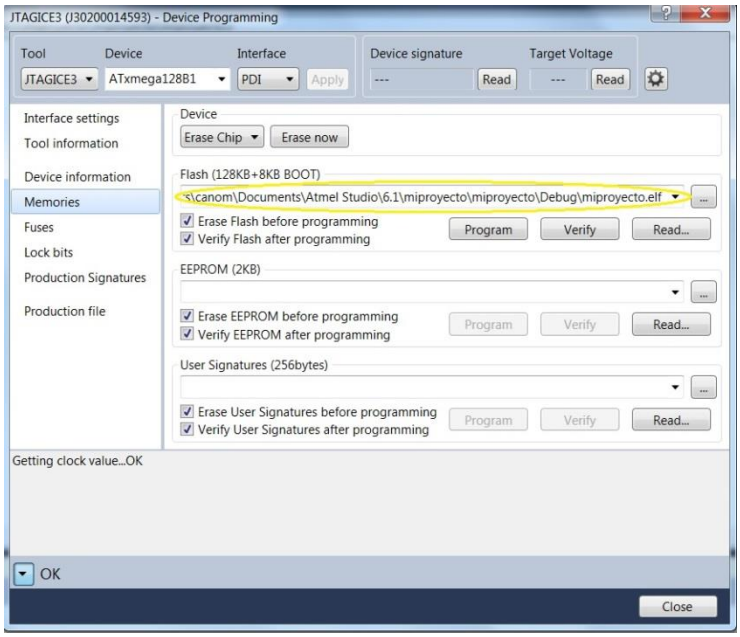


Figura 39 Programación de la tarjeta de desarrollo

Apéndice B. Máscaras de bits

Las máscaras de bits se usan habitualmente en la programación de alto nivel de microcontroladores, para poder tener acceso a bits individuales de un registro, sin alterar o sin leer los demás bits del registro. Para entender el uso de la máscaras de bits, primero se debe comprender el uso de algunos operadores binarios.

Operadores binarios

Los operadores binarios permiten hacer operaciones bit a bit entre números binarios.

Operador binario "OR"

El resultado de una operación binaria *OR* es 1 cuando alguno de los dos operandos es 1. La Tabla 3 muestra la tabla de verdad de este operador, el cual en C se representa con el símbolo |.

Tabla 3 Tabla de verdad del operador OR

Operando A	Operando B	Resultado A | B
0	0	0
0	1	1
1	0	1
1	1	1

Cuando se usa el operador *OR* para operar números binarios de varios bits, se aplica el operador bit a bit. En el siguiente ejemplo se ilustra una operación *OR* entre dos números binarios.

Ejemplo con operador "OR"

Operar 0b00100111 *OR* 0b10010001[2]

El operador se aplica bit a bit:

	0b	0	0	1	0	0	1	1	1
OR	0b	1	0	0	1	0	0	0	1
	0b	1	0	1	1	0	1	1	1

El resultado de la operación es 0b10110111.

[2] El prefijo "0b" se usa para indicar que el número está en formato binario. Este mismo prefijo se usa en programación en C.

Por facilidad, cuando se programa en C se usa la representación hexadecimal del número, en ese caso en C, la operación del ejemplo anterior se escribe:

0x27 | 0x97

Y el resultado es 0xB7[3].

Operador binario AND

El resultado de una operación binaria AND es 1 cuando los dos operandos son 1. La Tabla 4 muestra la tabla de verdad de este operador, el cual en C se representa con el símbolo &.

Tabla 4 Tabla de verdad del operador AND

Operando A	Operando B	Resultado A & B
0	0	0
0	1	0
1	0	0
1	1	1

Cuando se usa el operador *AND* para operar números binarios de varios bits, se aplica el operador bit a bit. En el siguiente ejemplo se ilustra una operación *AND* entre dos números binarios.

Ejemplo con operador AND

Operar 0b00100111 *AND* 0b10010001

El operador se aplica bit a bit:

	0b	0	0	1	0	0	1	1	1
AND	0b	1	0	0	1	0	0	0	1
	0b	0	0	0	0	0	0	0	1

El resultado de la operación es 0b00000001.

En C, usando representación hexadecimal, la operación de este ejemplo se escribe:

0x27 & 0x97

Y el resultado es 0x01.

[3] El prefijo “0x” se usa para indicar que el número está en formato hexadecimal.

Operador complemento a uno

El operador complemento a uno se aplica a un solo bit, el resultado es la negación del bit. La Tabla 5 muestra la tabla de verdad de este operador, el cual en C se representa con el símbolo ~.

Tabla 5 Tabla de verdad del operador complemento a uno

Operando A	Resultado ~A
0	1
1	0

Cuando se usa el operador ~ para operar un número binarios de varios bits, se aplica el operador bit a bit. En el siguiente ejemplo se ilustra una operación ~ a un número binario.

Ejemplo de complemento a uno

Operar ~0b00100111.

El operador se aplica bit a bit:

~	0b	0	0	1	0	0	1	1	1
	0b	1	1	0	1	1	0	0	0

El resultado de la operación es 0b11011000

En C, usando representación hexadecimal, la operación de este ejemplo se escribe:

~0x27 = 0xD8

Máscara de bits con *OR*

Las máscaras de bits se usan para cambiar uno o más bits de un registro sin alterar los demás bits del registro. De la tabla de verdad del operador *OR*, se deduce que, si se aplica la operación *OR* a un bit A con un 1, el resultado siempre será 1, sin importar el valor inicial del bit A.

A | 1 = 1 *sin importar el valor de A*

Por otro lado, si se aplica la operación OR a un bit A con un 0, el resultado será el valor inicial de A.

A | 0 = A

Realizar operaciones *OR* con 1 o con 0, es útil si se quiere escribir un 1 en un bit específico de un registro, pero no se quiere alterar los demás bits. En el siguiente ejemplo se ilustra una máscara de bits usando el operador *OR*.

Ejemplo de máscara de bits con OR

Se tiene un número binario X de 8 bits, se quiere escribir un 1 en el bit 4[4], sin alterar el valor de los otros bits, teniendo en cuenta que no se conoce el valor actual de X.

Se realiza una operación *OR* entre el número X y 0b00010000, al aplicar el operador bit a bit se tiene:

	0b	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
OR	0b	0	0	0	1	0	0	0	0
	0b	X_7	X_6	X_5	1	X_3	X_2	X_1	X_0

Como se observa en el resultado, se escribió un 1 en el bit 4 sin alterar los demás bits.

En C, usando representación hexadecimal, la operación de este ejemplo se escribe:

X | 0x10

Donde X es una variable o un registro del microcontrolador.

Máscara de bits con *AND*

El operador *AND* permite escribir un 0 en uno o varios bits específicos de un registro, sin alterar los demás bits. De la tabla de verdad del operador *AND*, se deduce que, si se aplica la operación *AND* a un bit A con un 0, el resultado siempre será 0, sin importar el valor inicial del bit A.

A | 0 = 0 *sin importar el valor de A*

Por otro lado, si se aplica la operación *AND* a un bit A con un 1, el resultado será el valor inicial de A.

A | 1 = A

En el siguiente ejemplo se ilustra una máscara de bits usando el operador *AND*.

Ejemplo 1 de máscara de bits con AND

Se tiene un número binario X de 8 bits, se quiere escribir un 0 en los bits 2 y 7, sin alterar el valor de los otros bits, teniendo en cuenta que no se conoce el valor actual de X.

Se realiza una operación *AND* entre el número X y 0b01110111, al aplicar el operador bit a bit se tiene:

	0b	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
AND	0b	0	1	1	1	1	0	1	1

[4] En representación binaria, el bit 0 es el bit menos significativo, es decir el bit más a la derecha.

0b	0	X_6	X_5	X_4	X_3	0	X_1	X_0

Como se observa en el resultado, se escribió un 0 en los bits 2 y 7 sin alterar los demás bits.

En C, usando representación hexadecimal, la operación de este ejemplo se escribe:

```
X & 0x7B
```

Donde X es una variable o un registro del microcontrolador.

La máscara de bits usando el operador *AND* también es útil cuando se desea conocer el valor de un bit específico de un registro.

Ejemplo 2 Interrogación de un bit

En la condición,

```
if((X & 0x01) !=0)
```

primero se realiza la operación (X & 0x01), el resultado de esta operación se compara con 0, si es diferente significa que la condición es verdadera y se ejecutarán las instrucciones incluidas en el if.

El detalle de la operación (X & 0x01) es:

	0b	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
AND	0b	0	0	0	0	0	0	0	1
	0b	0	0	0	0	0	0	0	X_0

del resultado de esta operación se observa que la condición de verdad depende únicamente del valor del bit 0 (X_0), con la máscara bits se logra aislar el bit 0, para conocer su valor.

Máscara de bits con Complemento a uno

Es común usar el operador complemento a uno en combinación con el operador *AND* cuando se desea escribir un 0 en uno o varios bits específicos de un registro.

En el ejemplo 1 de la subsección anterior se presentó la forma de escribir un 0 en los bits 2 y 7 del registro X, al operar el registro con 0b01111011 por medio del operador *AND*. Esto mismo se logra si se opera el registro X con ~(0b10000100).

El detalle de esta operación es:

0b	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0

AND	~(0b	1	0	0	0	0	1	0	0)
	0b	0	X_6	X_5	X_4	X_3	0	X_1	X_0

El operador complemento a uno tiene mayor precedencia que el operador *AND*, por lo tanto, primero se realiza la negación bit a bit y luego se realiza la operación *AND*. El resultado es el mismo que el obtenido en el ejemplo 1.

En C, usando representación hexadecimal, esta operación se escribe:

X & ~(0x84)

Esta forma es usada porque ~(0x84) tiene más sentido con respecto a la posición del bit en el cual se quiere escribir un 0.

Apéndice C. Lo básico de circuitos eléctricos.

Especificación de potencia en componentes electrónicos

Los componentes electrónicos (resistencias, leds, transistores, fuentes de alimentación) tienen una especificación de potencia, la cual no se debe exceder.

La potencia (expresada en watios) es igual a la multiplicación del voltaje entre los terminales del dispositivo (expresado en voltios) por la corriente que circula de un terminal al otro (expresada en amperios), ecuación (12)

$$P = V * I \qquad (12)$$

De acuerdo a (12), es necesario conocer el voltaje y la corriente de un componente para poder conocer la potencia. En la siguiente subsección se presenta el cálculo de corriente de un circuito de encendido de un LED.

Cálculo de corriente que circula por un LED

El circuito de la Figura 40, presenta la conexión eléctrica de los LEDs de la tarjeta de desarrollo XMEGAB1 Xplained. Cada LED se prende cuando en el pin correspondiente del microncontrolador hay un 0, esto se llama: “activo en bajo”. Cuando hay un “1” en el pin correspondiente, este “1” es igual a 3.3V por lo tanto el LED no se prende ya que la diferencia de potencial entre sus terminales es 0 voltios.

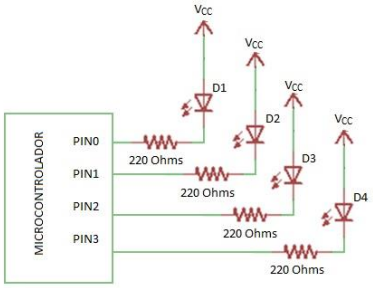


Figura 40 Circuito esquemático de 4 LEDS activos en bajo

Cuando hay un 0 en el pin del microcontrolador, este cero significa que hay una diferencia de potencial igual a 0 entre el pin y tierra, por lo tanto, es como si *virtualmente* el pin estuviera conectado a tierra.

Por leyes de Kirchhoff, la suma de los voltajes en una malla es igual a cero. El circuito formado por fuente, LED, Resistencia y pin del microcontrolador es una malla, como la fuente suministra energía, su voltaje tiene el signo contrario al voltaje del LED y de la resistencia, los cuales consumen energía. La ecuación (13) presenta la suma de voltajes de la malla.

$$V_{CC} - V_{LED} - V_R = 0 \quad (13)$$

Que es igual a:

$$V_{CC} = V_{LED} + V_R \quad (14)$$

El valor del voltaje de la fuente para la tarjeta de desarrollo XMEGAB1 Xplained es V_{CC} = 3.3V, el voltaje de encendido del LED es dado por el fabricante; el voltaje de encendido de los LEDs usados en la tarjeta es V_{LED} = 2V. Con estos dos valores se puede despejar V_R en (14), V_R = 1,3V.

Por ley de Ohm, el voltaje entre los terminales de una resistencia es igual al valor de la resistencia multiplicado por la corriente que circula de un terminal a otro, ecuación (15).

$$V_R = I_R * R \quad (15)$$

A partir de (15) y de los valores conocidos V_R y R, se puede calcular la corriente por la resistencia, ecuación (16).

$$I_R = \frac{1{,}3v}{220\Omega} = 0{,}0059A \quad (16)$$

La corriente que circula por cada LED cuando está encendido es: 5,9 mA. Esta es la misma corriente que entra al pin de microncontrolador.

A partir de estos cálculos se puede definir la potencia transferida al LED y a la resistencia, desde la fuente, ecuaciones (17)

$$P_{LED} = V_{LED} * I_{LED} = 2V * 5{,}9\,mA = 11{,}8\,mW \quad (17)$$

$$P_R = V_R * I_R = 1{,}3V * 5{,}9\,mA = 7{,}67\,mW$$

Los LEDs y las resistencias usadas en este circuito deben tener una especificación de potencia mayor a 11,8 mW y 7,67mW respectivamente.

Análisis de un circuito de acondicionamiento de señal con transistor

Algunos actuadores funcionan con una corriente más alta que la que permite la especificación del pin o del puerto, y/o con un voltaje más alto que la fuente del microcontrolador (3.3 v). Por lo tanto se debe hacer una etapa de salida como acondicionamiento de señal.

El circuito de la Figura 41 permite encender o apagar un LED usando una fuente de 5 voltios, por medio de la salida digital de un pin del microcontrolador.

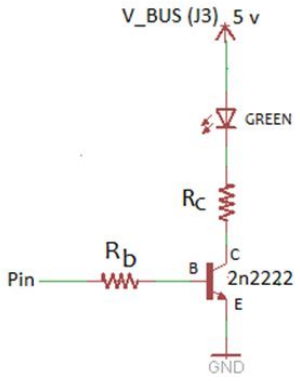


Figura 41 Circuito de acondicionamiento de señal para un LED alimentado con fuente de 5 voltios

El transistor 2n2222 funciona como un interruptor.

Caso 1: Cuando en el pin del micro hay un 1 (3.3v) hay corriente que ingresa en la base (B). Esto hace que el transistor deje pasar corriente entre el colector (C) y el emisor (E), por lo tanto el LED se enciende. En este caso el transistor funciona como un interruptor cerrado, y se encuentra en estado de saturación.

Caso 2: Cuando en el pin del micro hay un 0, no hay corriente que ingrese en la base (B) del transistor. Esto hace que el transistor se comporte como un interruptor abierto y no deje pasar corriente entre el colector y el emisor, por lo tanto el LED se apaga. En este caso el transistor se encuentra en estado de corte.

Análisis eléctrico del caso 1:

Cuando el transistor está saturado, entre B y E hay 0.6 voltios[5]; entre C y E hay 0.2 voltios[5]. Para asegurar que el transistor se encuentre en saturación la corriente por la base debe ser aproximadamente igual a 10 veces la corriente que circula de colector a emisor.

El voltaje de encendido de un LED verde es 3.5 voltios, por esto se eligió conectarlo a V_BUS que es igual a 5 voltios.

Cálculo del valor de la resistencia Rc

1. Se define la corriente con la cual se quiere encender el LED (se debe tener en cuenta la máxima corriente o la máxima potencia que soporta). Ej: 10 mA, que es igual a 0.01 A.

2. Por leyes de Kirchhoff, el voltaje sobre la resistencia R_C está dado por la ecuación (18).

$$V_R = V_{BUS} - V_{LED} - V_{CE} = 5 - 3{,}5 - 0{,}2 = 1{,}3\ V \qquad (18)$$

[5] Estos valores varían de acuerdo a la referencia y a las características de cada transistor

3. A partir del voltaje hallado en (18) y de la corriente decidida en el punto 1, por ley de Ohm se puede calcular el valor de resistencia de R_C. Ecuación (19)

$$R_c = \frac{V_{RC}}{I_{RC}} = \frac{1{,}3}{0{,}01} = 130\ ohms \qquad (19)$$

4. La potencia consumida por la resistencia es igual a: P_{RC} = 1,3 x 0,01 = 0,013 W, por lo tanto se debe usar una resistencia cuya especificación de potencia sea mayor a 13 mW

Cálculo del valor de la resistencia R_B

1. La corriente por la base debe ser aproximadamente 10 veces menor a la corriente por el colector- emisor. Como la corriente C-E seleccionada fue 10 mA por la base debe circular 1 mA, esto es 0,001 A.

2. Por leyes de Kirchhoff, el voltaje sobre la resistencia R_B está dado por la ecuación (20).

$$R_B = V_{Pin} - V_{BE} = 3{,}3 - 0{,}6 = 2{,}7\ V \qquad (20)$$

3. A partir del voltaje hallado en (20) y de la corriente definida en el punto 1, por ley de Ohm se puede calcular el valor de la resistencia R_B. Ecuación (21).

$$R_b = \frac{V_{RB}}{I_{RB}} = \frac{2{,}7}{0{,}001} = 2700\ ohms = 2{,}7\ Kohms \qquad (21)$$

4. La potencia de la resistencia es iguala a: P_{RB} = 2,7 x 0,001 = 0,0027 W, por lo tanto se debe usar una resistencia cuya especificación de potencia sea mayor a 2 mW

Ejemplos de circuitos de acondicionamiento de señal con transistor

El circuito de la Figura 42 presenta un acondicionamiento de señal para dos sistemas digitales con alimentación diferente

Cuando en el pin del micro haya un 0, el transistor está en corte, luego en el pin de entrada del sistema 2 hay VCC, lo cual representa un 1 para ese sistema.

Cuando en el pin del micro haya un 1, el transistor está en saturación, entre C y E hay 0,2v, por lo tanto, en el pin de entrada del sistema 2 se ve, un valor cercano a 0 voltios, el cual es interpretado como un 0 lógico.

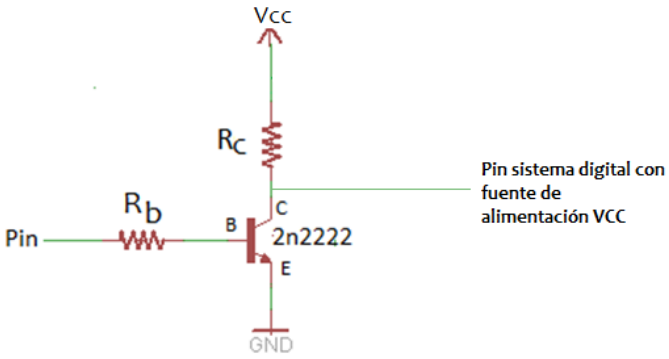


Figura 42 Acondicionamiento de señal para fuentes diferentes

Con este circuito se logra conectar dos sistemas con alimentaciones diferentes, sin embrago se invierte la lógica, cuando el primer sistema pone un 1, el segundo sistema recibe un 0 y viceversa. Esto se debe tener en cuenta a la hora de programar los dos sistemas.

La Figura 43, presenta un circuito de acondicionamiento para manejar un relevo. El relevo consume un valor de corriente constante, se debe revisar la especificación de corriente del relevo y verificar que la fuente de alimentación VCC sea capaz de suministrar esta corriente.

A partir de la corriente pedida por el relé y de la condición que la corriente por base es igual a 1/10 de la corriente E-C, se puede calcular Rb. Se debe verificar que la corriente por base no supere las especificaciones del pin, del puerto o del microcontrolador. Si las supera se debe poner una etapa adicional.

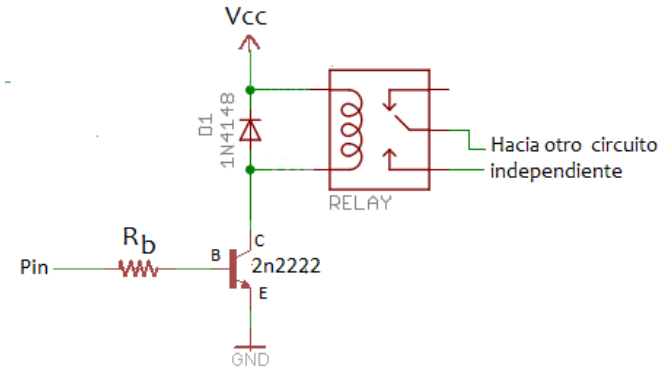


Figura 43 Acondicionamiento de señal para manejar un relevo

Para manejar un *buzzer* se reemplaza el diodo y el relé por el *buzzer* respetando la polaridad de este.

Apéndice D. Máquinas de estados finitos

Programación multitarea

Un sistema embebido se concibe para que esté en la capacidad de atender diversas tareas de procesos diferentes que pueden tener poca o nula relación entre ellos. Por ejemplo, un teléfono atiende la comunicación con la red celular, responde a la interfaz de usuario a través de la pantalla tacto sensible, al tiempo que lleva el reloj actualizado y maneja otras aplicaciones en el fondo.

En un computador o un dispositivo móvil, el sistema operativo (SO) se encarga de repartir el tiempo del procesador (o procesadores) entre todos los procesos y las tareas que se estén ejecutando. En un microcontrolador, que no cuenta con un SO, se debe realizar una programación que permita atender todos los procesos y las tareas de manera secuencial y recurrente.

Lo mínimo que se debe realizar para asegurar la atención de los diversos procesos y tareas de un microcontrolador es:

1. Para asegurar la recurrencia, el programa debe correr en un bucle infinito, para esto el programa principal se escribe dentro de un *while(1).*
2. Los procesos se programan de manera secuencial, y dentro de ellos se llaman las tareas de cada proceso, estas tareas deben ocupar el procesador un tiempo limitado.
3. Una buena práctica, es modelar los procesos con máquinas de estados finitos, en las cuales se llaman las tareas.
4. Ninguna tarea debe ocupar al procesador innecesariamente por ejemplo **con bucles de espera o delays.**

```
#include <avr/io.h>
int main(void)
{
        PORTB.DIRSET = 0x10;
        PORTA.DIRCLR = 0x01;
        PORTB.OUTSET = 0x10;
        while(1)
        {
                while((PORTA.IN & 0x01) != 0)
                {
                }
                PORTB.OUTTGL = 0x10;
                Otratarea();
                Otratarea();
        }
}
```

En el Código 17, se presenta un ejemplo en el cual el procesador es ocupado innecesariamente; en la

Código 17 Ocupación innecesaria de un procesador

instrucción *while*, resaltada en rojo, el procesador no ejecuta ningún procesamiento mientras no se cumpla la condición, esto implica que el procesador no podrá atender otras tareas.

Máquinas de estados finitos

Las máquinas de estados finitos (FSM por Finite State Machine), permiten que el código sea más eficiente, más fácil de depurar y ayudan a organizar el flujo del programa. Una FSM consiste en modelar un proceso a partir de un número finito de estados, en cada uno de los cuales el proceso tiene un comportamiento conocido y diferenciado. El proceso pasa de un estado a otro cuando la condición que activa esa transición se satisface, momento en el cual se pueden producir cambios en las salidas o en las variables internas del sistema. Un sistema puede tener varios procesos y por lo tanto varias FSM.

Un sistema de encendido y apagado de un bombillo por medio de la activación de un interruptor, tiene un solo proceso que tiene el mismo fin que el sistema completo, encender y apagar un bombillo. Este proceso se puede modelar con la FSM de la Figura 44.

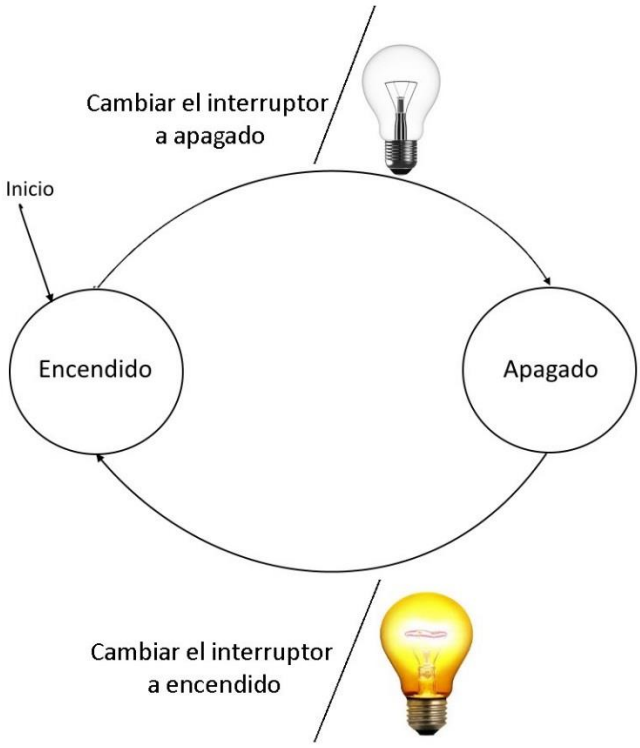


Figura 44 FSM de un sistema de encendido de un bombillo

Los círculos representan los estados del proceso, encendido y apagado, los arcos representan los cambios de un estado a otro, los cuales dependen de la activación del interruptor; durante esta

transición el bombillo se enciende o se apaga; el sistema permanece en el estado, mientras no se cumpla la condición de transición nuevamente.

En una FSM, uno de los estados será el estado inicial del sistema, lo cual se indica con una flecha de inicio o *reset*, en el caso de la FSM del proceso de encendido y apagado de un bombillo, el estado **Encendido** se define como el estado inicial.

Identificar los estados y las transiciones de un sistema

Los estados pueden relacionarse con los modos de operación del proceso, en los cuales el comportamiento o las funciones que debe realizar el sistema son diferentes. Siempre debe existir una condición de transición de un estado a otro, esta condición debe provenir de componentes externos al procesador, por ejemplo, de un periférico o de un componente de interfaz con el usuario; si no es posible identificar una condición de transición entre dos estados, esto indica que no son dos estados diferenciados.

Diagrama de estado

Una FSM se representa gráficamente por medio de un grafo, compuesto por nodos y por arcos dirigidos de un nodo a otro. Los nodos representan los estados y los arcos las transiciones. En los arcos se escriben las condiciones y las salidas que ocurren en la transición, separadas por el símbolo /. Este grafo se llama el diagrama de estados de la FSM.

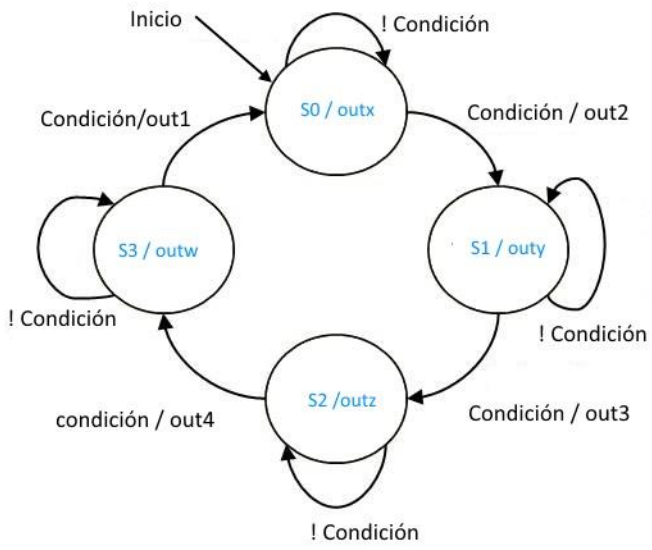


Figura 45 Diagrama de estados de un FSM

La Figura 45 presenta el diagrama de estados de una FSM general. Las siguientes características siempre deben existir en una FSM y su diagrama de estado:

- Se debe definir un estado inicial, el cual se indica por la flecha de inicio.
- La FSM debe ser **mínima,** esto significa que no deben existir estados equivalentes. Dos estados son equivalentes si tienen el mismo comportamiento y las transiciones son equivalentes.
- Las transiciones siempre deben tener una condición de transición, que depende de un componente externo al procesador.
- La FSM debe ser **fuertemente conexa**, esto significa que de cualquier estado se puede pasar a cualquier otro estado, haciendo una combinación de condiciones.
- La FSM no debe ser ambigua, es decir que no debe existir duda sobre el estado al que se debe pasar para una condición.
- Si la transición produce una salida del procesador hacia su entorno, está se debe indicar en la transición. Los cambios en variables internas no son salidas.
- Aunque no es habitual que durante los estados se produzcan salidas, si la aplicación requiere salidas durante los estados, estas se deben indicar en el estado.

Otras características que no son imperativas, pero sí deseables en una FSM y su diagrama de estados son:

- Los estados y las condiciones se designan con nombres o siglas sencillas.
- En los estados se puede indicar la condición de permanencia en ese estado.

Codificación de una FSM

Una vez que se ha concebido y diseñado la FSM que modela el proceso, se implementa por medio de código. Los siguientes pasos indican una forma de realizar esta implementación:

1. Se declara una variable de estado que lleva el control del estado actual del proceso
2. Los estados de la FSM se implementan usando un *switch/case*
3. Las transiciones se implementan usando *if* dentro de cada *case*.
4. Dentro de los *if* se realiza el cambio de estado, se llaman las tareas (funciones) que realizan los eventos de salida de la transición.
5. Dentro de los *case,* pero fuera de los *if*, se llaman las tareas (funciones) que el microcontrolador debe realizar de forma recurrente.

6. Las variables del proceso son locales, se pasan por referencia a las funciones si se van a modificar.
7. Las tareas (funciones) se implementan usando programación estructurada.

Ejemplo de FSM

Usando máquinas de estado finitas, diseñe un sistema que cumpla la siguiente especificación: Al presionar la primera vez un pulsador, activo en bajo, se enciende el LED0, al presionarlo la segunda vez se enciende el LED1 y se apaga el LED0, la tercera vez se prende el LED2 y se apaga el LED1, la cuarta vez se prende el LED3 y se apaga el LED2, la quinta vez se apagan todos y el ciclo vuelve a empezar.

El código debe realizar un solo cambio cada vez que se presione el pulsador, si el pulsador permanece pulsado no debe seguir alternando los LEDs.

En este sistema tenemos un solo proceso, para el cual se deben definir los estados. No existe una única máquina de estados posible, para este sistema, se pueden definir dos posibles FSM:

1. Dos estados, uno cuando el pulsador no está presionado, y otro cuando el pulsador está presionado. Esta FSM se enfoca en el comportamiento del pulsador, de tal forma que se pueda diferenciar cuando el usuario permanece con el pulsador presionado. Los cambios de los LEDs se realizan durante una de las transiciones.

2. Cinco estados, uno para cada LED encendido y uno para todos los LEDS apagados, LED0_ON, LED1_ON, LED2_ON, LED3_ON, ALL_OFF. Esta FSM se enfoca en el comportamiento de los LEDs, las transiciones entre estados se realizan cuando el pulsador haya sido presionado y soltado, esto último se debe asegurar en el código antes de realizar la transición.

También es posible realizar una combinación de las dos FSM, agregando un estado de *pulsador presionado* entre cada estado de la segunda FSM, obteniendo 9 estados. Cuando se agregan demasiados estados, la FSM puede perder su capacidad de simplificar el código, es importante buscar un balance entre la complejidad del problema y la complejidad de la FSM; en este caso, nueve estados para un problema sencillo pueden resultar en un código menos eficiente.

A continuación, se presenta el diseño e implementación de la primera FSM.

La FSM tiene dos estados:

- **Espera:** El pulsador no está presionado. A este estado se le llamará S0
- **Pulsador presionado:** El pulsador está presionado. A este estado se le llamará S1

La **¡Error! No se encuentra el origen de la referencia.** describe los estados, las condiciones de transición y las salidas

Tabla 6 Tabla de estados de sistema embebido

Estado actual	Estado Siguiente	Condición de la transición	Salida
S0	S1	Pulsador en 0	-
S1	S0	Pulsador en 1	Se apaga el LED encendido y se enciende el siguiente LED, o se pagan todos o se enciende el primero

El diagrama de estados se presenta en la Figura 46.

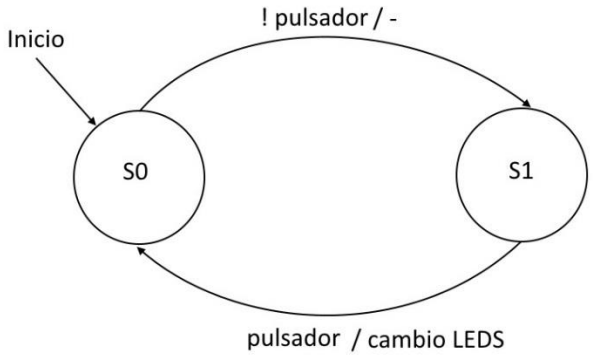


Figura 46 Diagrama de estados de sistema embebido

Este proceso tiene una única tarea, la cual que se encarga de hacer el cambio en las salidas conectadas a los LEDs. Una tarea se codifica mediante una función.

Además de la variable de estado, el proceso no necesita otras variables.

Para implementar el código se inicia por el encabezado, donde se incluye el .h del microcontrolador; se definen los nombres de los puertos; se define el tipo de dato STATE y su enumeración, según los estados de la máquina de estado; y se declara el prototipo de la función. Para la FSM de este ejemplo los estados S0 y S1, se llamaron *Idle* y *debouncing* respectivamente. Código 18

Código 18 Encabezado FSM

```
/* ------------------ Inclusion of Std Headers ------------------------ */

#include <avr/io.h>

/* -------------------- Declaration of Data --------------------------- */

#define LEDPORT PORTB                   //definition of LEDPORT

#define PULSEPORT PORTA                 //definition of PULSEPORT

typedef enum STATE STATE;               // definition of data type STATE

enum STATE {Idle, debouncing};          //enumeration of STATES

/* ----------------------- Prototype of functions ---------------------- */

void LEDS_TGL(void);//task that toggles LEDs' values.
```

Se incluye el *main*, con la declaración e inicialización de la variable de estado, la configuración de los pines de entrada y salida, y la inicialización de las salidas. Código 19.

```
int main(void)

{

        STATE state = Idle;             //state variable begins in Idle

        LEDPORT.DIRSET = 0xF0;          // Pin4 to pin7 of PORTB as outputs

        PULSEPORT.DIRCLR = 0x01;        // Pin0 of PORTA as input

        LEDPORT.OUTSET = 0XF0;          //4 leds OFF
```

Código 19 Main FSM

A continuación, en un *loop* infinito, se codifica la máquina de estados usando un *switch/case* y las transiciones usando *if*s. La tarea LEDS_TGL, se llama en la transición del estado *debouncing* al estado *idle.* Código 20.

```
while(1)//Loop infinito
{
        switch(state) {
           case Idle:                          // Idle State. S0
              if((PULSEPORT.IN & 0x01) == 0)    // Asks if the button is pressed.
              {
                  state = debouncing;           // changes to debouncing state.
              }
           break;
           case debouncing:                     // Debouncing state. S1
              if((PULSEPORT.IN & 0x01) != 0)    // Asks in the button is realeased.
              {
                  state = Idle;         // changes to debouncing state.
                  LEDS_TGL();           // Calls LEDS_TGL
               }                // If the button is still pressed, the FSM will stay in
                                // debouncing state.
            break;
        }
}
}
```

Código 20 Loop FSM

Finalmente se desarrolla el código de la función LEDS_TGL. El Código 20 presenta una forma de codificar esta función.

```
void LEDS_TGL()
{
        switch( (LEDPORT.OUT & 0xF0) ) {
                case 0xF0:
                   LEDPORT.OUTTGL = 0x10;
                break;
                case 0xE0:
                   LEDPORT.OUTTGL = 0x30;
                break;
                case 0xD0:
                   LEDPORT.OUTTGL = 0x60;
                break;
                case 0xB0:
                   LEDPORT.OUTTGL = 0xC0;
                break;
                case 0x70:
                   LEDPORT.OUTTGL = 0x80;
                break;
        }
}
```

Código 21 LEDS_TGL

Ejercicios

I. Complete el programa presentado en este apéndice, con el código que permita implementar un antirrebote para el pulsador.

II. Escriba, compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el código del ejemplo presentado en este apéndice. Monte el circuito que permite probar el código desarrollado, si no ha realizado el ejercicio I, incluya el circuito antirrebote que usa un Schmitt trigger.

III. Desarrolle la tabla de estados, el diagrama de estados y el código para la segunda FSM que da solución al problema planteado en este apéndice, la cual se basa en 5 estados. Incluya el código de antirrebote.

IV. Escriba, compile y programe en la tarjeta de desarrollo XMEGAB1 Xplained, el código del ejercicio III. Monte el circuito que permite probar el código desarrollado, si no incluyó antirrebote por programa, incluya el circuito antirrebote que usa un Schmitt trigger.

Apéndice E. Metodología de diseño de sistemas embebidos

En esta metodología se contemplan 3 etapas, la primera es la **especificación de diseño**, en la cual, a partir del planteamiento del problema, se seleccionan los componentes necesarios para la solución, se define un diagrama en bloques y se analizan las restricciones de tiempo que tiene el sistema. La segunda etapa es la de **concepción y diseño**, en la cual se diseñan las máquinas de estados finitos, se diseñan los circuitos de acondicionamiento de señal y se desarrolla el código de programa. Finalmente, la tercera etapa es la de **implementación y pruebas**.

Los pasos de diseño de las máquinas de estados finitos y su codificación, se presentan en el Apéndice D. Máquinas de estados finitos, los demás pasos de las primeras dos etapas se presentan a continuación, por medio de un ejemplo.

Planteamiento del problema

El sistema debe mostrar en dos displays 7 segmentos, el número de veces que se ha pulsado un pulsador. Cuando el número de veces llegue a 99 la cuenta reinicia en 00

Los displays 7 segmentos deben controlarse usando despliegue dinámico para ahorrar pines.

Especificaciones de diseño

Selección y caracterización de componentes

Para desarrollar la solución se usará:

- Tarjeta de desarrollo AVR XMEGA-B1 xplained
- Pulsador Qtouch activo en bajo, embarcado en la tarjeta de desarrollo.
- Display 7 segmentos de dos dígitos cátodo común. Referencia LB-202 BL.

Diagrama de bloques

Diagrama en bloques que incluya los componentes externos a la tarjeta (pulsadores, displays, chip de decodificación) y los componentes embarcados en la tarjeta.

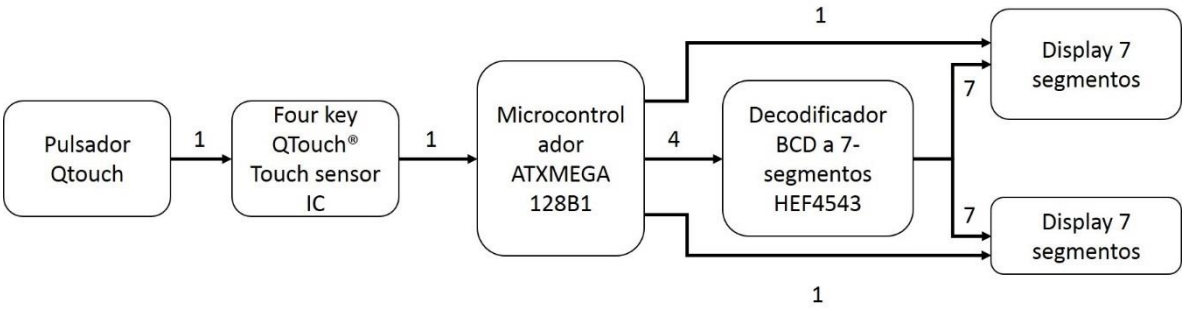


Figura 47 Diagrama en bloques de sistema embebido

Restricciones de tiempo

El sistema tiene una restricción de tiempo dada por el manejo de los displays 7 usando despliegue dinámico. Cada display se debe refrescar cada 10Hz, por lo tanto, el despliegue dinámico se debe realizar a una tasa mínima de 20Hz, lo cual equivale a un periodo máximo de 50ms.

Concepción y diseño

Máquina de estados finitos

Se siguen los pasos presentados en el Apéndice D. Máquinas de estados finitos, para concebir y diseñar la FSM que modela el único proceso del sistema.

Diseño o análisis de los circuitos de acondicionamiento de señales de entrada y salida

El pulsador Qtouch está embarcado en la tarjeta de desarrollo por lo tanto no es necesario realizar un circuito de acondicionamiento externo a la tarjeta. Para esta solución se usará el pulsador Qtouch CS0. La señal que proviene del pulsador QTouch CS0 pasa por un circuito integrado (IC) AT42QT1040 que acondiciona la señal, la salida de este IC está conectada al Pin 0 del puerto E. Es una señal activa en bajo, de colector abierto. Por ser de colector abierto es necesario activar la resistencia de pull-up del pin0 del puerto E.

Se usará un decodificador BCD 7-segmentos (HEF4543) para manejar los datos de los dígitos y un transistor 2n2222 para manejar el cátodo común de cada dígito. El decodificador se alimentará a la misma alimentación del microcontrolador de 3.3v.

La Figura 48 muestra el circuito esquemático del decodificador y del display de dos dígitos.

Se usarán los pines PA0 a PA3 para manejar los 4 bits de datos que van al decodificador. El pin PA4 activará el dígito 1 y el pin PA5 activará el dígito 2.

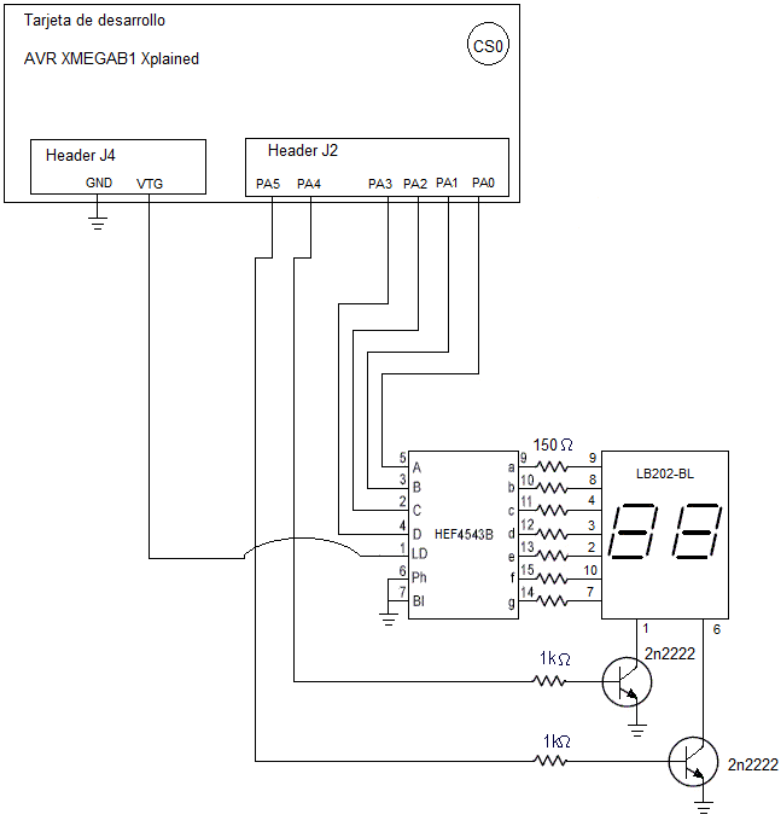


Figura 48 Circuito esquemático de sistema embebido

Cálculo de las resistencias en serie con cada segmento

La hoja de especificaciones del decodificador HEF4543 indica que la máxima corriente que puede entregar o recibir por cada pin es 10mA. La hoja de especificaciones del display LB202-BL indica que el voltaje de encendido de cada segmento es 2.1 voltios.

Cuando un segmento esté encendido habrá 3.3v en el pin de salida correspondiente del decodificador. Por lo tanto, la diferencia de potencial sobre la resistencia será: 3.3v - 2.1v = 1.2v

Para no exceder la especificación de corriente se manejará una corriente de 8mA por cada segmento. Para asegurar esta corriente la resistencia debe ser igual a: 1.2v ÷ 0.008A = 150 Ω

Cálculo de las resistencias de base de los transistores

Se seleccionó una resistencia de base de 1KΩ. Con este valor de resistencia cuando el transistor esté saturado circulará una corriente de (3.3v – 0,6v) ÷ 1000Ω = 2.7mA.

Cuando los 7 segmentos del mismo dígito estén encendidos al tiempo, por el cátodo común circulará 8mA x 7 = 56mA. Esta es la corriente de colector del transistor. En este caso la corriente por base (2,7mA) será 20 veces menor que la corriente de colector. Esta relación es suficiente para mantener el transistor en saturación.

Cuando 1 sólo segmento esté encendido, por el cátodo común circulará 8mA. Esta es la corriente de colector. En este caso la corriente por base (2,7mA) será 3 veces menor que la corriente de colector. Esta relación es suficiente para mantener el transistor en saturación.

Desarrollo del código de programa

Se siguen los pasos presentados en el Apéndice D. Máquinas de estados finitos para desarrollar el código de la FSM diseñada.

Implementación

Montaje, programación y pruebas

Finalmente se hace el montaje del circuito, la programación de la tarjeta y las pruebas del sistema.

Apéndice F. Depuración en Atmel® Studio

Es común que, al programar un microcontrolador, el sistema no funcione como se había planeado o simplemente no funcione en lo absoluto, debido a fallas en el diseño o en la implementación del código. Para poder diagnosticar en dónde se encuentra la falla, es de gran utilidad realizar un proceso de depuración o *debugging*. La depuración permite ver, en la ejecución de cada instrucción, como van cambiando los valores en los registros y en las variables del código; también permite cambiar manualmente estos valores para orientar el comportamiento del sistema.

Para hacer depuración del código en tiempo real sobre el microcontrolador es necesario tener un depurador o *debugger*. Algunos programadores también son depuradores, este es el caso de los programadores con referencia ICE y *Dragon* de Atmel.

Atmel Studio® también cuenta con un simulador que permite depurar el código sin programarlo en el microcontrolador, lo cual es muy útil cuando no se cuenta con un depurador.

En este apéndice se presenta un paso a paso de la depuración en Atmel Studio®, el cual es válido para depuración con programador y para depuración con simulador. Un manual completo de *debugging* se encuentra en: http://www.atmel.com/webdoc/atmelstudio/atmelstudio.Debug.html

Paso a paso de depuración

Paso 1: Abra su proyecto en Atmel Studio.

Paso 2: Conecte el programador/depurador a su circuito y al computador, como si fuera a hacer una programación regular. Omita este paso si va a usar el simulador de Atmel.

Paso 3:

Para iniciar una sesión de depuración seleccione en el menú ***Debug*** la opción ***Start Debugging and Break*****.** Si tiene conectado un debugger y Atmel lo reconoce, pasará directamente al paso 4. Si no tiene conectado un *debugger*, o si Atmel no lo reconoce, aparecerá una ventana de aviso como la de la Figura 49, que indica que no hay una herramienta de depuración seleccionada.

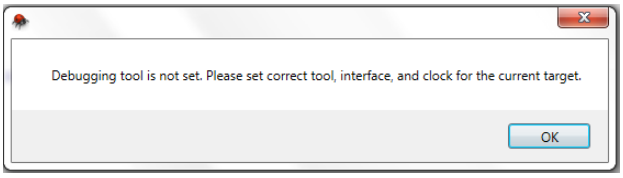


Figura 49 Aviso de depurador no seleccionado

Al dar **OK,** se abre una nueva pestaña en la ventana principal de Atmel Studio (Figura 50), donde puede seleccionar el ***debugger/programmer***. Escoja ***simulator*** si no tiene un ***debugger*** conectado. Luego guarde (CTRL+S) y seleccione nuevamente **Debug -> Star debugging and break** (o ALT+F5).

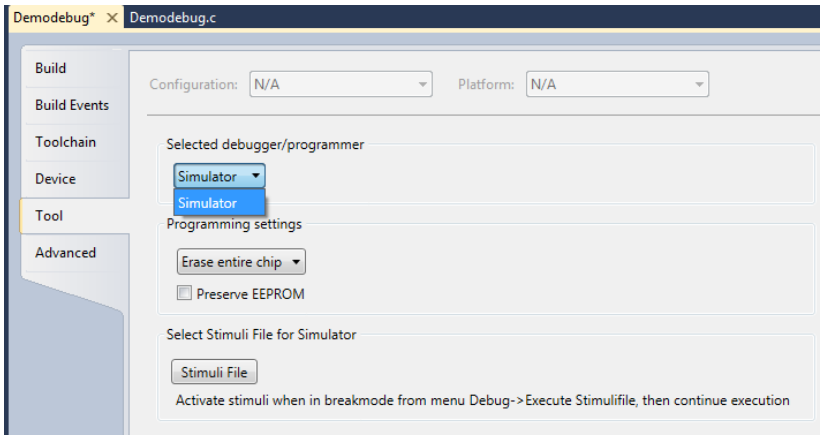


Figura 50 Selección del depurador

Paso 4:

Atmel inicia una depuración, parando en la primera instrucción del programa. En la Figura 51, se ve una flecha amarilla al lado izquierdo de la primera instrucción, esta flecha muestra que hay una sesión de depuración en curso y que el proceso está parado en esa instrucción.

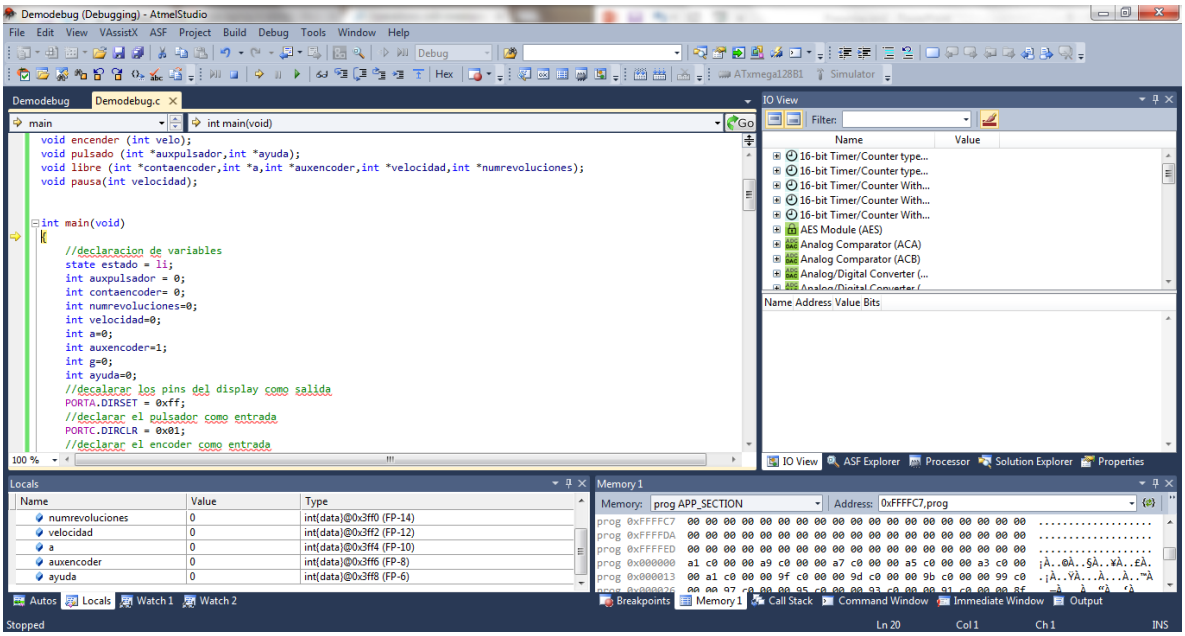


Figura 51 Inicio de una depuración en Atmel Studio

Al lado derecho de la ventana principal de Atmel Studio, aparece la ventana ***IO View***. Si no ve esta ventana seleccione **Debug -> Windows -> I/O view.**

En la parte inferior de la ventana principal de Atmel Studio, aparece la ventana ***Locals***. Si no ve esta ventana, seleccione **Debug -> Windows -> Locals.**

Por último, cierre las ventanas **Memory** que estén abiertas.

Paso 5:

Despliegue el menú ***Debug*.**

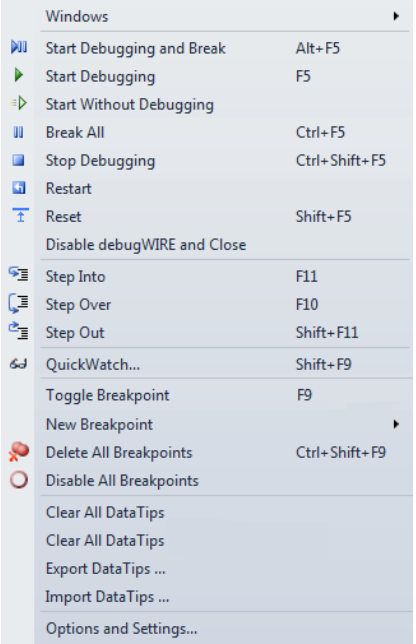


Figura 52 Menú Debug

Observe las siguientes 3 opciones y su función:

Step Into: Ejecuta una instrucción

Step Over: Similar a **Step Into**. Step Over ejecuta una instrucción; sin embargo, si la instrucción contiene una llamada a una función o a una subrutina, la función o subrutina se ejecuta completamente.

Step Out: Continua la ejecución hasta que la función actual se complete.

Ensaye cada una de estas opciones.

Paso 6:

Avance instrucción por instrucción usando la opción **Step Into** (F11). A medida que la flecha amarilla va avanzando, observe cómo cambian los registros en la ventana IO View. Para esto, debe buscar el periférico que está siendo afectado en esa instrucción, seleccionarlo y observar el detalle de los registros que se muestran en la parte inferior de la ventana **IO View**.

Para ilustrar este proceso, a continuación, se muestra un ejemplo. En la Figura 53, se tiene un código con una instrucción resaltada; esta instrucción afecta el registro DIRSET del PORTA. En la ventana IO View se selecciona el periférico PORTA, lo que produce que los registros de ese periférico aparezcan en la parte inferior.

Al ejecutar la instrucción, por medio de Step into (F11), los registros DIR cambian su valor, el programa los resalta en rojo para mostrar que hubo un cambio. La Figura 54, muestra la nueva posición de la flecha amarilla, lo cual indica que la instrucción `PORTA.DIRSET=0xff` ya fue ejecutada; En la ventana IO View, resaltados en rojo, se muestra los cambios en los registros DIR del PORTA.

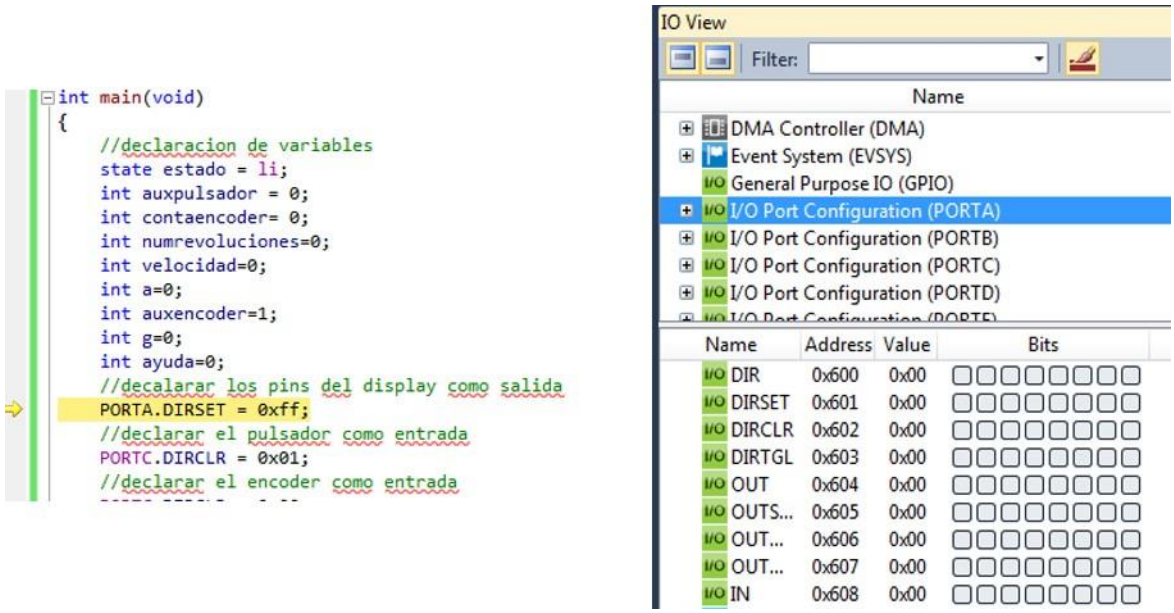


Figura 53. Izquierda: Instrucción a depurar. Derecha: periférico PORTA y sus registros

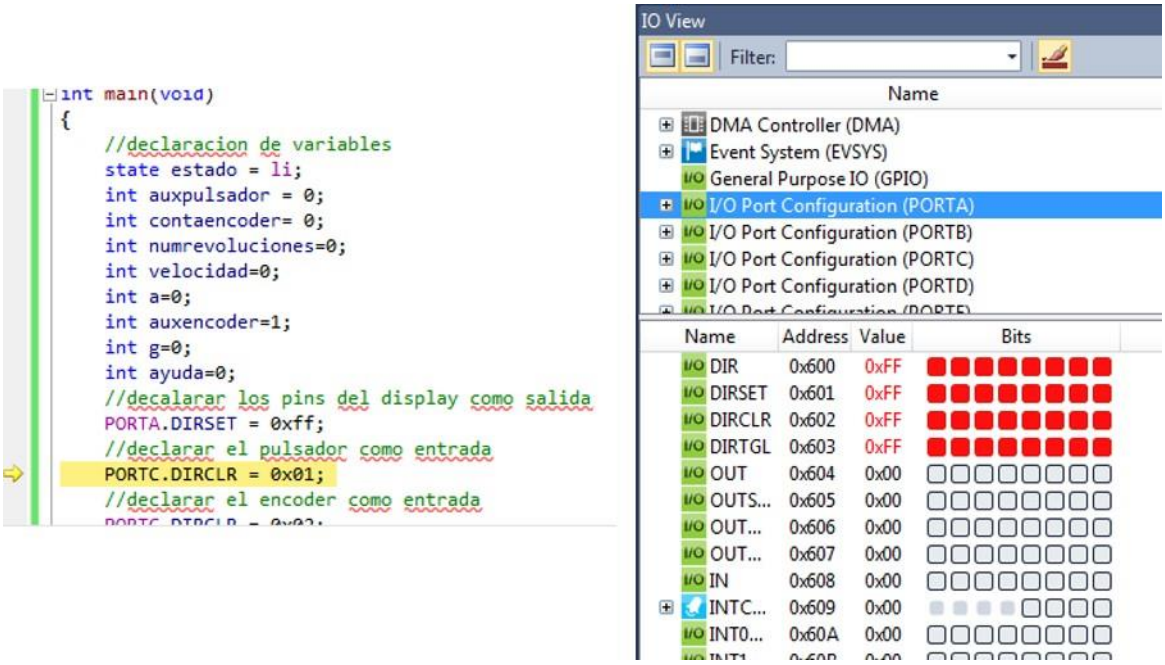


Figura 54 Izquierda: instrucción ya depurada. Derecha: Registros modificados

Paso 6:

A medida que avanza la depuración instrucción por instrucción, observe como cambian los valores de las variables locales de su código, en la ventana **Locals**. La Figura 55 muestra la ventana **Locals** para el código de ejemplo de la Figura 53, en esta figura se ven los valores de inicialización de las variables locales, incluida la variable de estado que inicia en *li*.

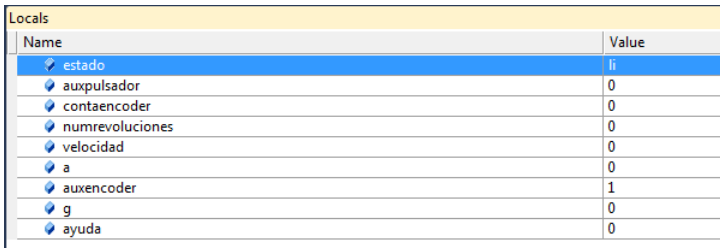


Figura 55 Ventana Locals

Paso 7:

Incluya uno o varios *breakpoints* haciendo doble click en la barra lateral izquierda al lado de una instrucción. Debe aparecer un círculo rojo en la barra lateral y la instrucción se resalta en rojo, como se muestra en la Figura 56.

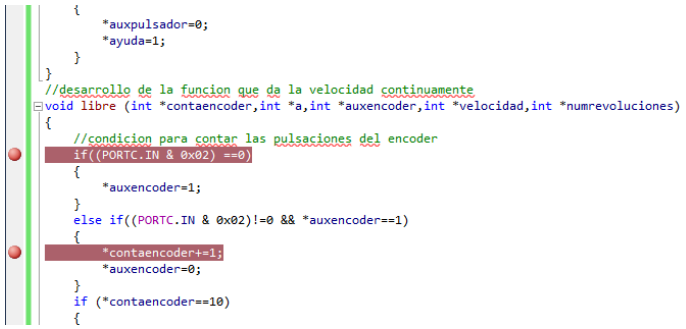


Figura 56 Inclusión de breakpoints en el código

Luego de incluir los *breakpoint*s, seleccione **Debug -> Continue** (o F5). La depuración continúa hasta que encuentra un *breakpoint*. Esta opción reemplaza el **Step into** y se usa cuando se quiere observar los registros y las variables en un punto específico del código.

Paso 8:

Si la depuración se está haciendo en tiempo real sobre un microcontrolador, active las entradas al microcontrolador (por ejemplo, un pulsador) y verique que la acción esperada se cumpla, usando breakpoints, observando los registros, las variables u observando si los actuadores conectados al microcontrolador cambian de estado.

Si se está realizando la depuración por medio del simulador de Atmel Studio, es necesario emular las entradas para probar el código. La emulación de las entradas se puede realizar de dos formas:

- Cambiando manualmente el valor del registro IN del puerto correspondiente.
- Cambiando manualmente el valor de la variable que sería afectada con la activación de una entrada.

Para ilustrar este proceso, a continuación, se presenta un ejemplo. La Figura 57, muestra el código de una máquina de estados finitos (MEF) con dos estados *li* y *pa*. La posición de la flecha amarilla indica que

la MEF está en el estado *li*; para pasar al estado *pa* es necesario presionar un pulsador, activo en alto que se encuentra en el pin0 del PORTC.

```
switch(estado)
{
    case li: //estado libre, debe enseñar la velocidad en el display.
    libre(&contaencoder,&a,&auxencoder,&velocidad,&numrevoluciones);
    encender (velocidad);
    pulsado(&auxpulsador,&ayuda);
    if(ayuda==1)
    {
        g=velocidad;
        estado = pa;
        ayuda=0;
    }
    break;

    case pa: //estado pausa, debe tomar la ultima velocidad y enseñarla en el display.
    libre(&contaencoder,&a,&auxencoder,&velocidad,&numrevoluciones);
    pausa(g);
    pulsado(&auxpulsador,&ayuda);
    if(ayuda==1)
    {
        estado = li;
        ayuda=0;
    }

    break;
}
```

Figura 57 Código de MEF de dos estados

Para emular un 1 en el bit0 del PORTC:

1. Seleccione PORTC en la ventana IO view.
2. Luego, en la parte inferior, seleccione el registro IN.
3. Dé click en el cuadrado correspondiente al bit0, el cuadrado pasará de estar vacío a tener un relleno, esto indica que el valor del bit es 1. La Figura 58 muestra el resultado de esta secuencia.

Para verificar que la emulación del pulsador tuvo efecto:

4. Agregue un *breakpoint* dentro del estado al cual debe pasar la MEF al presionar el pulsador.
5. Seleccione **Debug-> Continue (F5).**
6. El depurador debe detenerse en el *breakpoint*, dentro del estado pa.
7. Para emular el fin de la activación del pulsador, cambie el bit 0 a 0, siguiendo los pasos del 1 al 3.

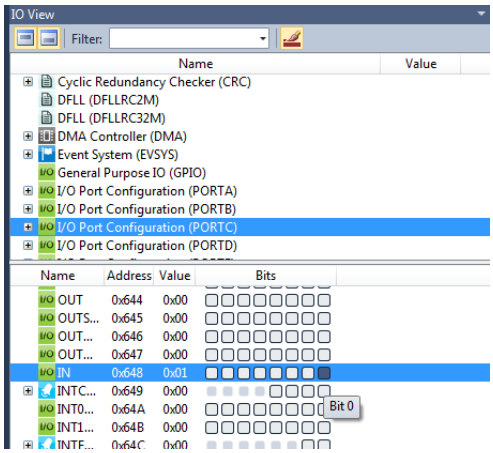


Figura 58 Escritura de un valor en un bit para emular una entrada

Otra forma de cambiar el estado de la MEF, consiste en cambiar directamente el valor de la variable "estado". Para esto, en la ventana **Locals**, se da doble click en el valor *li* de la variable y se escribe *pa*, se finaliza con un enter para que el nuevo valor sea tomado.

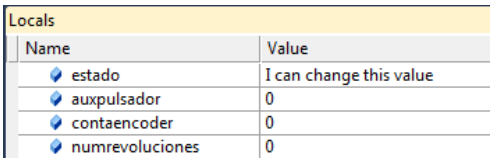


Figura 59 Cambio del valor de una variable

Comentario final

Los bits de registros de otros periféricos también se pueden cambiar. Por ejemplo, se puede cambiar la bandera del *Timer*, para poder probar el código que se ejecuta cuando esta bandera es verdadera. La Figura 60, muestra resaltado en azul, el registro INTFLAGS del periférico TCC0. Para poner en 1 la bandera, sólo se debe dar click en el bit 0 de este registro.

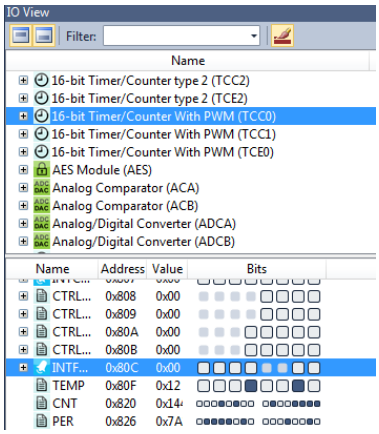


Figura 60 Cambio del valor de la bandera de TCC0

Apéndice G. Pulsadores capacitivos y Pantalla de Cristal Líquido.

Pulsadores capacitivos Qtouch

Los pulsadores capacitivos *Qtouch* están embarcados en la tarjeta de desarrollo, se identifican por un círculo blanco con las letras CS0, CS1, CS2 y CS3. Estos pulsadores tienen la misma funcionalidad que un pulsador mecánico, al acercar el dedo el pulsador se activa, al alejarlo se desactiva.

La señal que proviene de cada pulsador QTouch pasa por un circuito integrado (IC) AT42QT1040 que acondiciona la señal. La Figura 61 muestra el circuito esquemático correspondiente a este acondicionamiento de señal. Las salidas del IC AT42QT1040, están conectadas a los pines 0 a 3 del PORTE del microcontrolador ATXMEGA128B1. Son señales activas en bajo, de colector abierto.

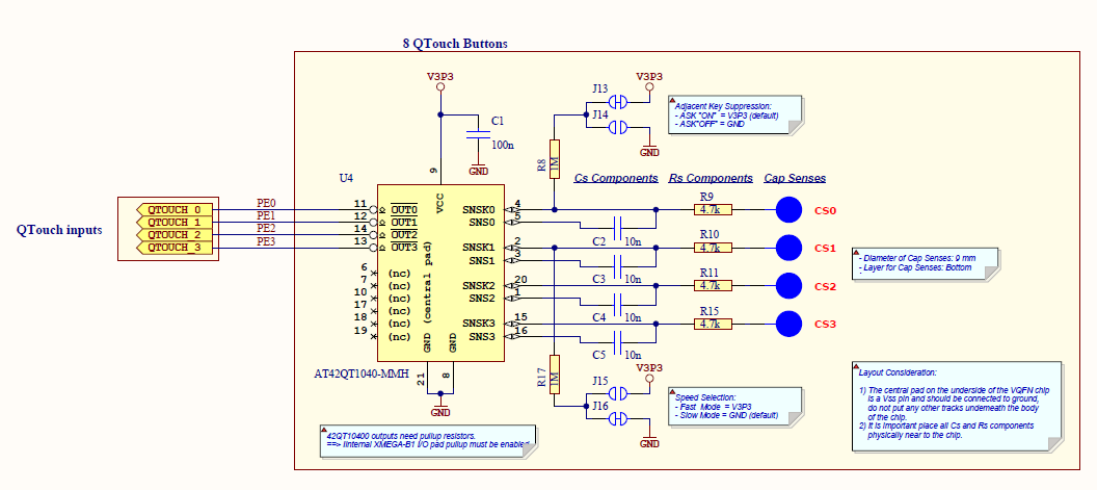


Figura 61 Acondicionamiento de señal sensores Qtouch.

Por ser de colector abierto es necesario activar la resistencia de pull-up del pin correspondiente al pulsador que se quiera usar. La instrucción presentada en Código 22, activa la resistencia de pull-up para el pin 0 del PORTE. Para activar las resistencias de los pines 1 a 3, se debe reemplazar “PIN0CTRL” por “PINxCTRL”, donde x es el número del pin.

```
PORTE.PIN0CTRL = PULSEPORT.PIN0CTRL| 0x18; //pull-up resisitor en el pin0;
```

Código 22 Activación de resistencia de Pull-up para Qtouch

Pantalla de Cristal Líquido

La tarjeta de desarrollo AVR XMEGA-B1 Xplained tiene una pantalla de cristal líquido (LCD) embarcada. Esta pantalla, cuya referencia es C42048a, tiene 4 pines *comunes* y 40 pines *segmentos*, a través de ellos y realizando un despliegue dinámico se puede manejar los 160 segmentos que posee la pantalla. En la Figura 62 se muestran los 160 segmentos.

El microcontrolador ATXMEGA128B1 tiene un periférico exclusivo para el manejo del LCD y 44 pines exclusivos para conectar la pantalla. Aprender a manejar en detalle este periférico no hace parte de los objetivos de esta guía, se usará una librería que permitirá configurar el periférico y manejar el LCD.

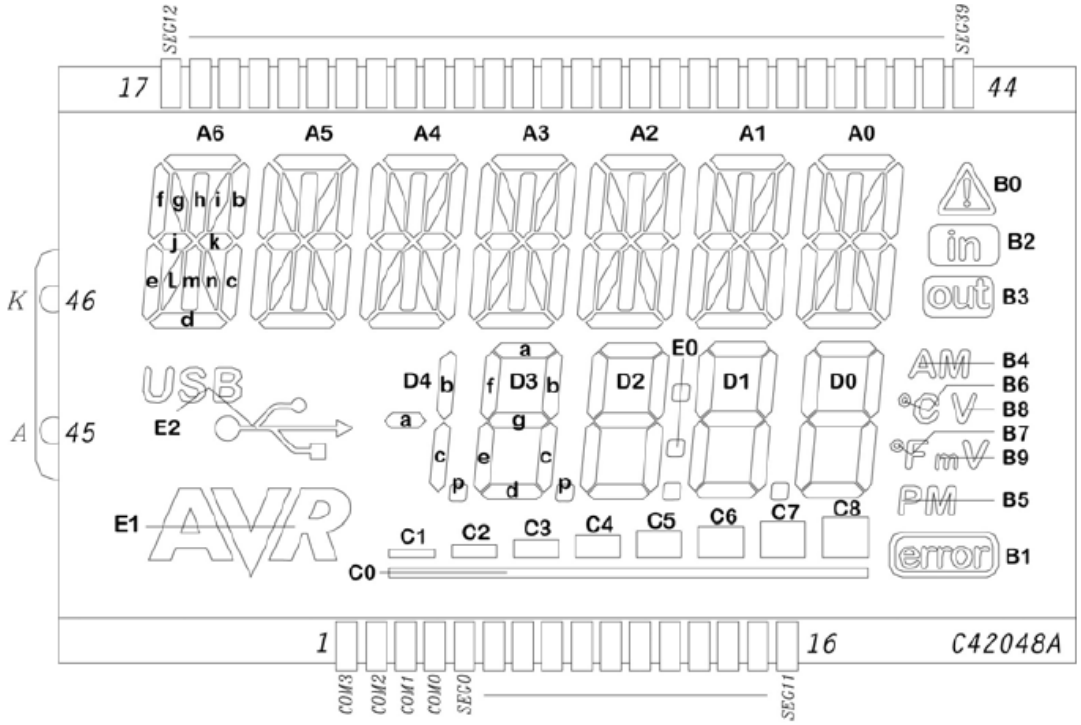


Figura 62 Segmentos del LCD. Tomado de [3]

	SEG																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
COM0	C0	B0	D0-a	D0-f	D1-a	D1-f	D2-a	D2-f	D3-a	D3-f	D4-a	B9	A6-h	C1	A6-a	A6-g	A5-h	C3	A5-a	A5-g
COM1	E2	B2	D0-b	D0-g	D1-b	D1-g	D2-b	D2-g	D3-b	D3-g	D4-b	B8	A6-i	A6-f	A6-b	A6-j	A5-i	A5-f	A5-b	A5-j
COM2	E1	B3	D0-c	D0-e	D1-c	D1-e	D2-c	D2-e	D3-c	D3-e	D4-c	B7	A6-k	A6-e	A6-c	A6-L	A5-k	A5-e	A5-c	A5-L
COM3	E0	B1	—	D0-d	D1-p	D1-d	D2-p	D2-d	D3-p	D3-d	D4-p	B6	A6-n	A6-d	C2	A6-m	A5-n	A5-d	C4	A5-m

	SEG																			
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
COM0	A4-h	C5	A4-a	A4-g	A3-h	C7	A3-a	A3-g	A2-h	B5	A2-a	A2-g	A1-h	—	A1-a	A1-g	A0-h	—	A0-a	A0-g
COM1	A4-i	A4-f	A4-b	A4-j	A3-i	A3-f	A3-b	A3-j	A2-i	A2-f	A2-b	A2-j	A1-i	A1-f	A1-b	A1-j	A0-i	A0-f	A0-b	A0-j
COM2	A4-k	A4-e	A4-c	A4-L	A3-k	A3-e	A3-c	A3-L	A2-k	A2-e	A2-c	A2-L	A1-k	A1-e	A1-c	A1-L	A0-k	A0-e	A0-c	A0-L
COM3	A4-n	A4-d	C6	A4-m	A3-n	A3-d	C8	A3-m	A2-n	A2-d	B4	A2-m	A1-n	A1-d	—	A1-m	A0-n	A0-d	—	A0-m

Figura 63 Mapa de pines para cada segmentos del LCD

Para el uso de la librería se debe tener en cuenta el mapa de pines que se muestra en la Figura 63. Este mapa relaciona la combinación que se debe activar para encender cada segmento. Por ejemplo, para encender el ícono de grados centígrados (C°), el cual se identifica con el nombre B6 en la Figura 62, se debe activar el COM3 y el SEG11, de acuerdo al mapa de la Figura 63.

Para incluir la librería que permite usar el LCD, busque en el directorio de instalación de Atmel los siguientes archivos: lcd.h, lcd.c, c42048a.c, c42048a.h y avr_compiler.h. Cópielos en la misma carpeta de Windows donde se encuentra el archivo principal .c del proyecto en el cual quiera hacer uso del LCD.

Siga los siguientes pasos para inicializar el LCD:

1. En el main, en la sección de configuración de periféricos o de inicialización de variables, llame la función: `c42048a_init();`
2. Inmediatamente establezca el nivel de contraste deseado con la función: `c42048a_set_contrast(60);`

A continuación se presenta una descripción de las funciones principales de esta librería, estas funciones se podrán usar a lo largo del código según lo requiera el proyecto:

```
void c42048a_bar_graph(uint8_t val):
```

Enciende la barra progresiva del LCD de acuerdo al valor de *val*. Si *val* es mayor a 0 enciende C1, si *val* es mayor a 32 enciende C1 y C2 , así sucesivamente; para val mayor a 224 enciende C1 a C8.

```
void c42048a_set_numeric_dec(uint16_t val):
```

Muestra un número decimal en el campo numérica del LCD (D0 a D4). *Val* corresponde al valor que será mostrado.

```
void c42048a_set_text(const uint8_t *data):
```

Muestra una palabra alfanumérica en el campo alfanumérico del LCD (A0 a A6). *Data* es la palabra que será mostrada.

```
void c42048a_set_pixel(uint8_t pix_com, uint8_t pix_seg):
```

Enciende un segmento específico del LCD. Los parámetros pix_com y pix_seg son el COM y el SEG que corresponden al segmento, de acuerdo al mapa de la Figura 63.

```
void c42048a_clear_pixel(uint8_t pix_com, uint8_t pix_seg):
```

Apaga un segmento específico del LCD. Los parámetros pix_com y pix_seg son el COM y el SEG que corresponden al segmento, de acuerdo al mapa de la Figura 63.

Bibliografía

[1] 8-bit Atmel XMEGA B Microcontroller. www.atmel.com

[2] ATxmega128B1 / ATxmega64B1. www.atmel.com

[3] AN AVR1912: Atmel XMEGA-B1 Xplained. Hardware User guide. www.atmel.com

[4] xmega-B1 Xplained Rev4_Design_Documentation.PDF www.atmel.com

[5] AN AVR1926: XMEGA-B1 Xplained Getting Started Guide www.atmel.com

[6] Sedra, Smith. Circuitos microelectrónicos. Oxford University press.

[7] Application note AVR127: Understanding ADC Parameters. www.atmel.com

[8] Application note AVR120: Characterization and Calibration of the ADC on an AVR. www.atmel.com

[9] https://www.pololu.com/product/2130

[10] Application note AVR131: Using the AVR's High-speed PWM. www.atmel.com

[11] Atmel AVR32852: Building Custom Application using ASF Example Projects. www.atmel.com

[12] Datasheet LED - RGB Clear Common Anode - SparkFun Electronics

[13] Datasheet DRV8833 Dual H-Bridge Motor Driver

[14] *http://timvandevall.com/rgb-color-wheel-hex-values-printable-blank-color-wheel-templates/*

[15] SG90 9 g Micro Servo.pdf

[16] Application note AVR1510: Xplain training - XMEGA USART. www.atmel.com

[13] Atmel AT42QT1040 Datasheet. www.atmel.com