Hola amigos, éste tutorial está basado en el documento “AVR318: Dallas 1Wire Master” y en la librería 1 wire proporcionada en la página:
http://www.microsyl.com/index.php/2010/03/23/onewire-library/#more-181

Teoría Protocolo 1 wire

Como la mayoría de nosotros sabemos 1-wire es un protocolo de comunicación creada por Maxim en donde participa un maestro, y uno o varios esclavos. La comunicación es asíncrona, half dúplex, ocupa sólo una línea para enviar y recibir datos, y los esclavos no requieren de una fuente externa de alimentación, ya que todos los dispositivos que poseen integrado este protocolo poseen en su interior un condensador que reúne la carga suficiente durante la transmisión de maestro a esclavo para que éste pueda enviar una respuesta.

En reposo el bus debe permanecer en estado alto y generalmente debe tener una resistencia externa pull up “strong”, es decir una resistencia de pequeño valor que reduzca la constante RC y aumente la velocidad de carga del condensador descrito anteriormente. Sin embargo, para este tutorial ocupamos un Atmega16, así que ocupamos la resistencia pullup integrada al microcontrolador. Muchos pensarán inmediatamente que éste no es adecuada para el bus ya que los valores que puede alcanzar son alrededor de los MegaOhm, pero, si observamos la tabla 12-1 de la hoja de datos del Atmega16 podemos observar que, para cierta configuración de los registros DDRx y Portx los pines actúan como Source o Sink, es decir, como fuente o sumidero y como nos interesa que se llene de carga el condensador interno de los esclavos, pues ocupamos una de las configuraciones para tal propósito.

Table 12-1. Port Pin Configurations

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Señales básicas

Hay cinco señales básicas las cuales se pueden dividir en “slots” de tiempo de 60 uS (mínimo) y éstas son “Escribir un 1”, “Escribir un 0”, “Leer”, “Pulso de Reset” y “Pulso de Presencia”.

Escribir un 1

La señal “Escribir un 1” se muestra en la figura 1. El maestro cambia el estado del bus, pasando del estado alto al estado bajo, y lo mantiene así durante un periodo de 1 a 15 uS. Luego libera el bus por el resto del slot de tiempo.

Escribir un 0

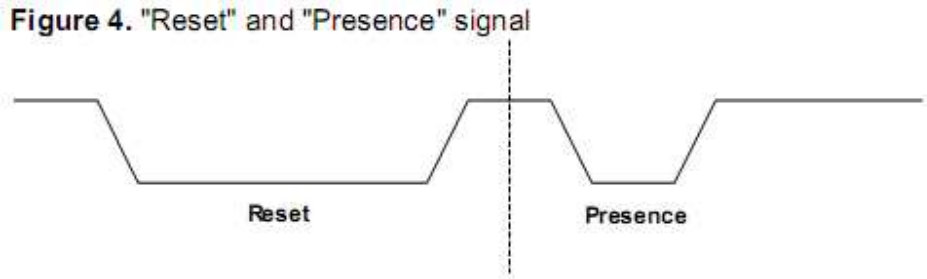
La señal “Escribir un 0” se muestra en la figura 2. El maestro lleva el bus al estado bajo durante un período de por lo menos 60 uS, con un ancho máximo de 120 uS.

Leer

La señal “Leer” se muestra en la figura 3. El maestro cambia el estado del bus al estado bajo durante un período que puede abarcar de 1 a 15 us. El esclavo por su parte mantiene el bus en estado bajo si desea enviar un 0 o si quiere enviar un 1 simplemente libera la línea.

Pulso de *reset* y presencia

Las señales *Reset* y presencia se muestran en la figura 4 y difieren ligeramente de las anteriores señales. Cuando el maestro desea enviar un pulso de *reset* cambia el estado del bus al estado bajo por lo menos durante 8 *slots* de tiempo, es decir, 480 us y luego lo libera. Si el esclavo está presente y quiere enviar un pulso de presencia, éste deberá cambiar el estado de la línea durante los primeros 60 us contados a partir del momento que el maestro libera el bus, y luego mantener este estado bajo durante otros 60 us como mínimo.



Generación de las señales

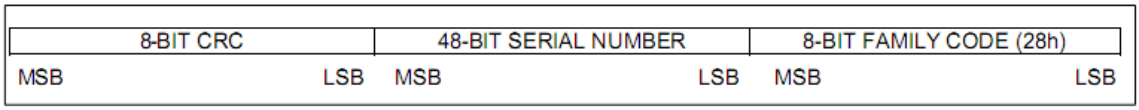
Existen dos maneras distintas para generar las señales descritas anteriormente en un atmega16:

a) Por software: cambiando el valor y dirección de los pines, y generando el delay necesario.
b) Por USART: conectando RX y TX al mismo bus usando un buffer open collector, y generando las señales usando un baudrate determinado.

En este tutorial, se describirá el primer método ya que es sencilla de implementar y se puede modificar para que cualquier pin de propósito general del microcontrolador sea capaz de manejar por lo menos un esclavo.

Funciones o Comandos ROM

Antes de comenzar a describir los algoritmos necesarios, debemos tener en cuenta que todos los dispositivos 1 wire poseen un identificador único en su ROM el cual nos facilita la identificación y direccionamiento de todos los esclavos que están conectados a la red Microlan.



Este identificador consta de 64 bits y se puede distinguir tres partes:

a) El código de familia (8 bits).
b) El número serial (48 bits).
c) El valor CRC (8 bits), que es calculado tomando los 56 bits anteriores.

Para poder manejar los datos anteriores existe un grupo de instrucciones llamados Comandos ROM que son comunes para todos los dispositivos 1 wire, y otro grupo llamado Comandos Memory que son propias de cada dispositivo esclavo. A continuación describiré los comandos ROM:

Comando READ ROM (h33)

El comando “READ ROM” se puede usar sólo con un esclavo para leer su identificador único de 64 bits. Si hay varios dispositivos esclavos conectados al bus, se producirá un error de CRC.

Comando SKEEP ROM (hCC)

El comando “SKEEP ROM” se utiliza para direccionar a uno o muchos esclavos a la vez, siendo útil para enviar una instrucción o comando Memory y así ejecutar tareas en forma simultánea.

Comando MATCH ROM (h55)

El comando “MATCH ROM” se usa para consultar a un dispositivo esclavo su número de identificación. Una vez que el comando “MATCH ROM” se envía, el identificador de 64 bits se transmite completo en el bus. Cuando esto termina, sólo al dispositivo identificado se le permite contestar hasta que éste reciba un pulso de *reset* por parte del maestro.

Comando SEARCH ROM (hF0)

El comando “SEARCH ROM” se puede utilizar cuando no se conoce la cantidad de esclavos conectados a un bus. Es el más complejo, ya que al enviar esta instrucción todos los esclavos colocan sus identificadores además de los complementos de cada bit. Más adelante describiré el algoritmo necesario para ejecutar este comando.

Juntando todo

Todos los dispositivos 1 wire siguen una secuencia básica de comunicación:

1) El maestro envía un pulso de “reset”.
2) El esclavo(os) responde con un pulso de “presencia”.
3) El maestro envía un comando ROM.
4) El maestro envía un comando Memory.

Es muy importante seguir esta secuencia cada vez que se acceda a un dispositivo 1 wire, ya que si no se hace éste simplemente no responderá y se perderán datos del esclavo. Sin embargo, hay una excepción a esta regla: la instrucción SEARCH ROM. Después de enviarse, el programa puede volver al paso 1.

Inicialización

El primer paso es inicializar el bus 1 wire, y consiste en dejar el o los pines elegidos como inputs y activar las resistencias pull up. Teniendo esto en cuenta podemos empezar escribir la primera función en lenguaje C.

```
void owire_release_bus()
{       OWIRE_DDR &= ~(1<<OWIRE_PIN_0);          //Input
        OWIRE_OUT |= (1<<OWIRE_PIN_0);           //Activando resistencia pull up
}
```

Donde OWIRE_DDR corresponde al registro DDRx o registro que establece la dirección del pin, y OWIRE_OUT, al registro PORTx que activa o desactiva las resistencias pull up internas. Esta función es importante ya que permite que el bus vuelva al estado de reposo o Idle en ciertos momentos claves del programa.

El segundo paso es implementar las funciones necesarias para las cuatro señales básicas. El documento "AVR318: Dallas 1Wire Master" nos muestra los algoritmos para cada una de las señales además de los delays necesarios para escribir nuestras funciones:

Figure 13. Bit transfer layer functions.

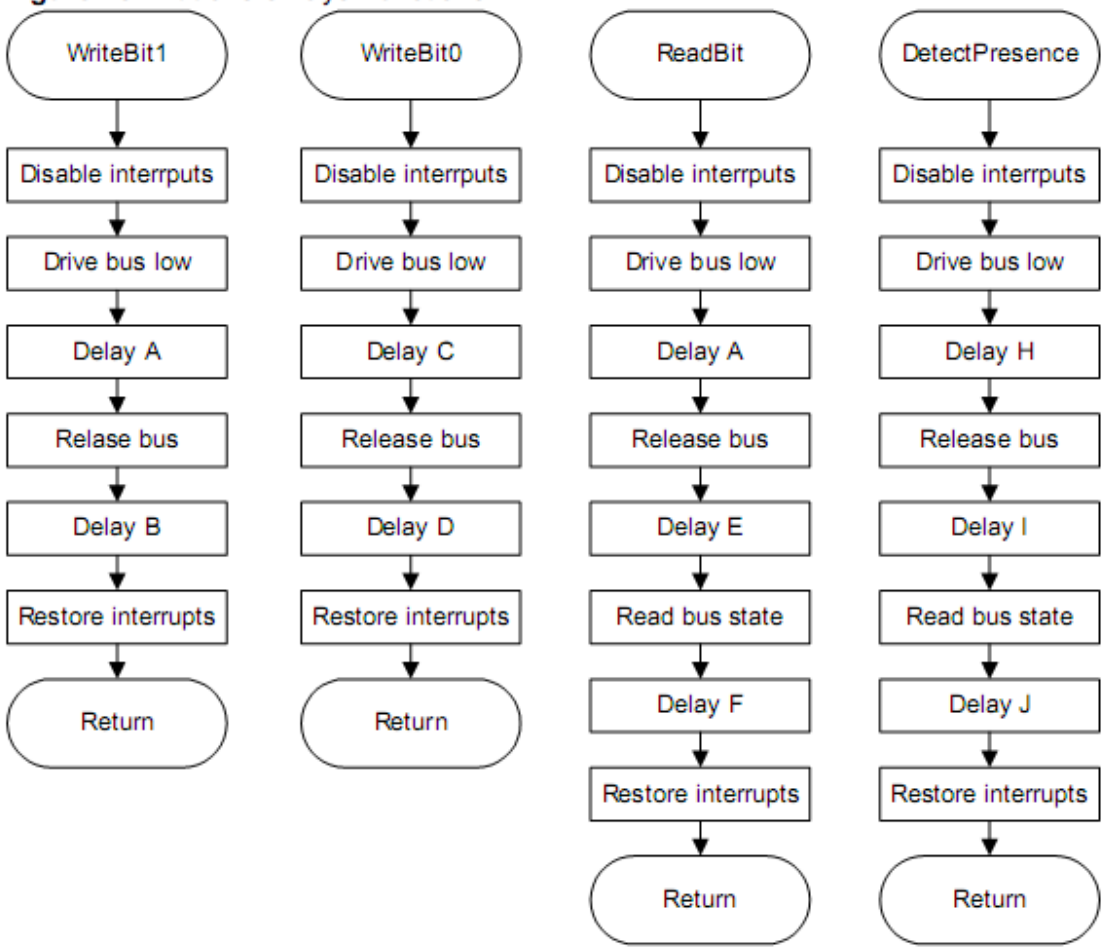


Table 3. Bit transfer layer delays

Parameter	Recommended delay (µs)
A	6
B	64
C	60
D	10
E	9
F	55
G	0
H	480
I	70
J	410

Comencemos con la señal reset/presence. Si observamos el algoritmo paso por paso entonces tenemos:

1) Desactivar interrupciones: podemos perfectamente desactivar todas las interrupciones con la instrucción “cli()” que desactiva el bit de “Habilitación de interrupciones globales” del registro SREG.

2) Cambiar el estado Idle del bus al estado bajo: podemos observar que esta parte es contraria a la función owire_release_bus() ya que configura el pin elegido como salida y desactiva su pullup interno.

```
OWIRE_OUT &= ~(1<<OWIRE_PIN_0);   // Desactivar pull up interno

OWIRE_DDR |= (1<<OWIRE_PIN_0);      //Configurar pin como salida
```

3) Delay H: un delay de 480 us.

4) Liberar el bus: usando la función owire_release_bus().

5) Delay I: un delay de 70 us.

6) Leer la señal de presencia: si realmente hay un esclavo conectado, entonces después de los 70 us, se podrá leer perfectamente un 0. Por lo tanto escaneamos el bit correspondiente en el registro PINx (OWIRE_IN)

```
OWIRE_IN & (1<<OWIRE_PIN_0)
```

7) Delay J: un delay de 410 us.

8) Finalmente, restaurar las interrupciones con la instrucción sei();

Si juntamos todo en una sola función nos queda así:

```
uint8_t owire_reset_presence()
{
        uint8_t estado;
        cli();
        OWIRE_OUT &= ~(1<<OWIRE_PIN_0);
        OWIRE_DDR |= (1<<OWIRE_PIN_0);
        delay_us(480);
        owire_release_bus();
        delay_us(70);
        estado = (OWIRE_IN & (1<<OWIRE_PIN_0));
        delay_us(410);
        sei();
        return estado;
}
```

Ahora sigamos con las señales enviar un 1 y enviar un 0, que son similares. Lo único en lo que difieren son sus delays. Para escribir un 1 tenemos:

1) Desactivar interrupciones.

2) Cambiar el estado Idle del bus al estado bajo.

3) Delay A: un delay de 6 us.

4) Liberar el bus

5) Delay B: un delay de 64 us.

6) Restaurar interrupciones.

```
void write_one()
{                   cli();
                    OWIRE_OUT &= ~(1<<OWIRE_PIN_0);   // Desactivar pull up interno
                    OWIRE_DDR |= (1<<OWIRE_PIN_0);     //Configurar pin como salida
                    delay_us(6);
                    owire_release_bus();
                    delay_us(64);
                    sei();
}
```

Para enviar un 0 utilizamos las líneas anteriores, cambiando sólo los delays según lo mostrado en el algoritmo anterior:

```
void write_zero()
{
                    cli();
                    OWIRE_OUT &= ~(1<<OWIRE_PIN_0);   // Desactivar pull up interno
                    OWIRE_DDR |= (1<<OWIRE_PIN_0);     //Configurar pin como salida
                    delay_us(60);
                    owire_release_bus();
                    delay_us(10);
                    sei();
}
```

Y finalmente, leer un bit:

1) Desactivar interrupciones.
2) Cambiar el estado Idle del bus al estado bajo.
3) Delay A: un delay de 6 us.
4) Liberar el bus: usando la función owire_release_bus().
5) Delay E: un delay de 9 us.
6) Leer la señal de presencia.
7) Delay F: un delay de 55 us.
8) Restaurar las interrupciones con la instrucción sei();

```
uint8_t owire_read_bit()
{
uint8_t estado;
        cli();
        OWIRE_OUT &= ~(1<<OWIRE_PIN_0);   // Desactivar pull up interno
        OWIRE_DDR |= (1<<OWIRE_PIN_0);     //Configurar pin como salida
        delay_us(6);
        owire_release_bus();
        delay_us(9);
        estado = (OWIRE_IN & (1<<OWIRE_PIN_0));
        delay_us(55);
        return estado;
        sei();
}
```

Finalmente tenemos la base para escribir los programas que queramos para controlar o manipular todos los dispositivos 1 Wire que queramos.

ENJOY! :D