Universidad Nacional de La Plata
Facultad de Ingeniería

Trabajo practico 1-D

Cronometro

Gallo, Molinuevo Francisco
Pérez, Francisco

Problema

4. Deberá implementar además la siguiente lógica para hacer funcionar el reloj. Al presionar un pulsador de START el programa reanuda el conteo, mientras que al presionar un pulsador de STOP el conteo se detiene. Para reiniciar a cero el contador deberá presionar un pulsador de RESET. Tenga en cuenta el efecto de rebote de los pulsadores reales.

Interpretación:

Debemos implementar el ejercicio C propuesto anteriormente agregándole la opción de tener 3 botones que cumplan la función de START, STOP y RESET. Por lo que debemos afrontar el problema que generan utilizar botones mecánicos, denominado efecto rebote.

Solución:

Multiplexacion:

Junto con el ATMEGA328 y cuatro display de 7 segmentos se debe simular un cronometro.

Se utilizarán varios puertos como salida. Primero, se utilizará el Puerto B del 0 al 6 para mandar a cada display el número que le corresponde.
Como los display de 7 segmentos son independientes de cada uno, es decir, un display solo puede representar un número del 0 al 9, se debe multiplexar el número a representar inicialmente. Además, se tiene que tener en cuenta que solo 1 display puede estar encendido a la vez y mostrar el número que le corresponde. Ya que, como se mencionó, el puerto de salida es compartido. Hay que sincronizar que display encender y que valor mandar al puerto B. Para esto, se empleará 4 pines del puerto C.
Se utilizará 4 variables donde en cada una guardaremos el valor de uno los 4 dígitos (Ver Pseudocodigo 1.) y mandaremos este valor en BCD al display que corresponda.

Obtener el resto de dividir por 10 del entero para obtener el digito
Guardarlo en una variable

Si el número es distinto de 0 (Numero > 9)
Dividir numero por 10 y sacar el primer digito
Obtener el resto del entero al dividirlo por 10
Guardar décima en otra variable

Si el número es distinto de 0 (Numero > 99)
Dividir numero por 10 y sacarnos la decima
Dividir el numero por 10 y obtener el resto
Guardar centena en otra variable

Si el número es distinto de 0 (numero > 1000)
Dividir numero por 10 y sacar la centena
Dividir el numero por 10 y obtener el resto
Guardar milésima en otra variable

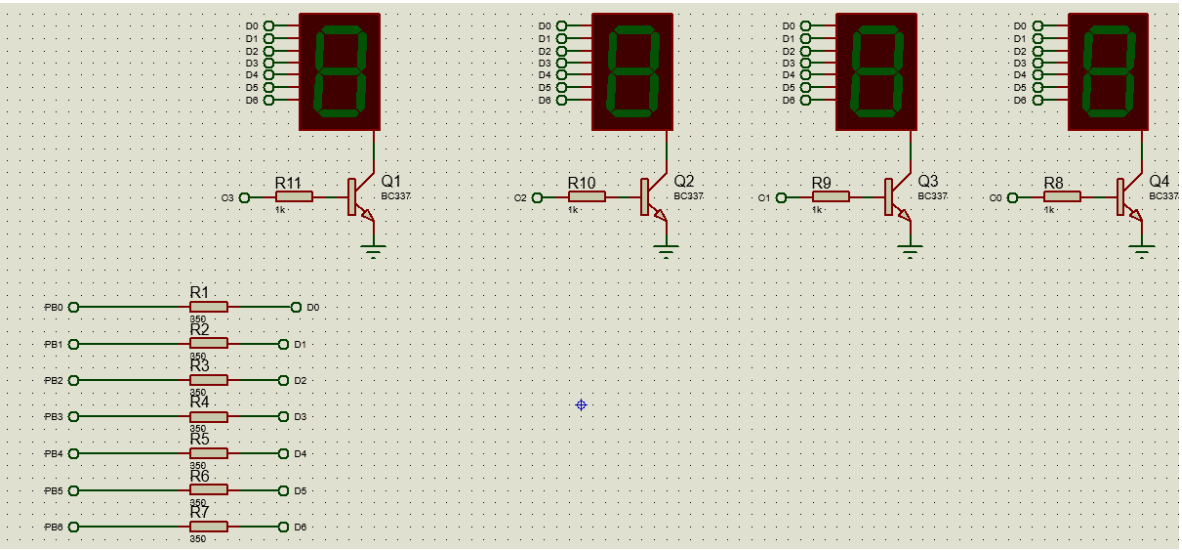
Pseudocodigo 1

El siguiente paso es mandar estas variables a los 4 display convertidos en BCD. El principal problema es la utilización de un solo puerto de salida como señal al display sobre qué valor tomar.

La solución es utilizar otro puerto de salida y alguna modificación circuital a los display para poder mostrar en un instante de tiempo cierto display con su número a representar, luego el siguiente y así.

La frecuencia la cual sucede este cambio de encendido y apagados es tan rápido que el ojo humano no lo percibe, pareciendo que siempre estos están encendidos.

El puerto de salida que utilizaremos es el C y el modelo circuital está conformado por:



C0, C1, C2 y C3 corresponden a los pines del puerto C.

Las conexiones que se le hacen a los display es que, si sobre el pin que está conectado a ese circuito es un valor ‘alto’ este se encuentra encendido, mientras que si esta en ‘bajo’ se encuentra apagado.

Para finalizar, luego de obtener los 4 dígitos debemos sincronizar el encendido del display y el digito que debe representar ver Pseudocodigo 1.2

```
Prender solo display 1
Mandar al puerto B el primer digito en BCD
Prender solo display 2
Mandar al puerto B el segundo digito en BCD
Prender solo display 3
Mandar al puerto B el tercer digito en BCD
Prender solo display 4
Mandar al puerto B el cuarto digito en BCD
```

Pseudocodigo 1.2: showDisplay()

Donde *Prender solo el display x* significa de mandar un valor en alto a ese pin y un bajo a todos los demás.

Botones mecánicos:

A diferencia de otro ejercicio, se necesita obtener entradas externas, en este caso, un botón.

Se decidió utilizar los primeros 3 pines del puerto D como entrada a cada botón (**RESET, START, STOP**). Se debe tener en cuenta el problema de la utilización de botones mecánicos (efecto rebote).

¿Qué es el efecto rebote?

El botón mecánico produce un efecto rebote el cual provoca errores en los algoritmos por valores inválidos, donde, además, es afectado por el envejecimiento del dispositivo.

Figura 1.

La *Figura 1* muestra como una señal eléctrica luego de presionar y liberar el pulsador tiene un tiempo hasta que esta sea estable, por lo que se recomienda esperar hasta esta se estabilice.

Se plantean 2 soluciones:

Anti-rebote determinado por un 'delay': luego de verificar si el pulsador fue presionado, esperar un determinado tiempo donde, durante este tiempo, el procesador está realizando ciclos vacíos, es decir, instrucciones que no hacen nada. Es una forma fácil de implementar, pero a veces no es recomendable ya que se está, durante muchos ciclos, haciendo nada.

Anti-rebote determinado por ciclos: Otra forma más eficiente es calcular aproximadamente cuantos ciclos tarda en realizar la iteración del microcontrolador. Luego de conocer este dato, se puede determinar cuántos ciclos son los milisegundos que queremos esperar hasta consultar sobre el estado del botón. Una vez la señal este estable, debemos esperar a que el botón sea soltado, ya que queremos solo realizar una acción cuando el botón es pulsado y luego soltado.

Hay que tener en cuenta de que el oscilador del MCU que utilizamos es de 16Mhz.

Sea c (tiempo de duración de un ciclo) $c = \frac{1s}{16Mhz} = 62{,}5ns$

Una vez obtenido este resultado, se decide consultar sobre el estado del botón cada 200 ms (se supone que el botón no es presionado más de 2 veces por segundo).

Donde d (ciclos proporcional a 200 ms) $d = \frac{200ms}{62{,}5ns} = 3.200.000\ ciclos$

Utilizando la herramienta de Debug del compilador, podemos conocer cuántos ciclos consume la iteración del programa

INSERTAR FOTO DEL DEBUG

Por lo tanto, para tener 3,200,000 de ciclos, necesitamos $160280 * X = 3.200.000$

$$X = 20\ iteraciones$$

Entonces cada x tiempo, vamos a consultar el valor del botón, si este está estable, realizaremos las acciones que estén ligadas a esa acción.

Para verificar que la señal es estable, se consulta si sobre 2 iteraciones la señal del botón fue la misma, funcionando como un doble checkeo.

¿Qué solución vamos a utilizar?

De las 2 opciones, se utilizó la primera ya que la segunda no se obtuvieron los resultados esperados.

La forma de implementar cuando un botón fue presionado y luego soltado fue:

```
Si el botón fue presionado
   Esperar tiempo anti-rebote
   Mientras el botón este presionado
      No hacer nada
   Espero tiempo anti-rebote (ya que el botón fue soltado)
   Realizar acción
```

Pseudocodigo 1.3

Para finalizar el tema de los botones, se definirá la acción de cada uno donde:

START: Si el botón START es presionado, hará que la variable que se utiliza para llevar el número que se representa en los display se incremente

STOP: Si el botón STOP es presionado, hará que la variable que se utiliza el para llevar el número que se representa en los display no cambie

RESTART: Si el botón RESTART hará que la variable que se utiliza el para llevar el número que se representa en los display se inicialice en 0

Para una mayor eficiencia en el programa, utilizaremos una variable que nos determine que evento sucedió. Si durante la ejecución el botón START no fue pulsado, no haremos la lógica explicada anteriormente sobre multiplexacion. Ya que, si este botón no fue pulsado, la variable contadora no varía y el número que se manda a los display es el mismo. Haciendo así, varias divisiones y asignaciones de más.

```
Si el botón START fue presionado
   Eliminar efecto rebote
   Multiplexar variable
   Incrementar variable contadora
Si el botón STOP fue presionado
   Eliminar efecto rebote
   No hacer nada sobre la variable contadora
Si el botón RESET fue presionado
   Eliminar efecto rebote
   Reiniciar variable contadora
```

Pseudocodigo 1.4

Programa:

```
Lazo infinito
Si el botón START fue presionado
   Pongo el flag en 1
Si el botón RESTART fue presionado
   Inicializo en 0 las variables contador y las variables de cada digito
Si el botón STOP fue presionado
   Pongo el flag en 0

Si el flag esta en 1
   Hago la multiplexacion
   Incremento variable contadora

Muestro en los display el numero
```

El flag es una opción que se tomó para poder manipular que botón se presionó y así también minimizar lógica. Dependiendo de esta variable, el programa hace o no ciertas cosas

Conclusión:

Luego de utilizar el Proteus para confirmar el correcto funcionamiento, se puede ver que el algoritmo si logra su objetivo. Se tuvo mucho en cuenta el efecto rebote de cada botón. Se intentó realizar el algoritmo con un algoritmo de polling por ciclos, pero no funcionaba como debía. La utilización de modulos creados en los informes anteriores facilito hacer este algoritmo. Por lo tanto la mayoría de las cosas realizadas en este ejercicio, fue implementado anteriormente.

Anexo

```
#include <avr/io.h>
#define F_CPU 16000000UL // Defino la frecuencia de oscilador en 16MHz
#include <util/delay.h>

const float DELAY = 2.5;
const uint8_t BCDTABLE[10] = {0b00111111, 0b00000110, 0b01011011, 0b01001111,
0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111};


//Anti rebote
uint8_t debounce(uint8_t pin){
	if(!(PIND & (1<<pin))){
		_delay_ms(20);
		while(!(PIND & (1<<pin)));
		_delay_ms(20);
		return 1;
	}
	return 0;
}



// Obtener centesima de segundo, segundo, decima y centesima del contador;
void separarDigitos(uint16_t* contador, uint8_t* digito1, uint8_t* digito2, uint8_t*
digito3, uint8_t* digito4) {
	uint16_t aux = 0;
	aux = *contador;
	if(aux != 0){
		*digito1 = aux % 10;
		aux = aux / 10;
	}
	if(aux !=0){
		*digito2 = aux % 10;
		aux = aux / 10;
	}
	if(aux != 0){
		*digito3 = aux % 10;
		aux = aux / 10;
	}
	if(aux != 0){
		*digito4 = aux % 10;
	}
}


// Por cada DELAY ms, muestro el valor que le corresponde a cada digito en el
dispositivo de 7 segmentos.
void showDisplays(int digito1, int digito2, int digito3, int digito4){
	PORTB = BCDTABLE[digito1];
	PORTC = 0b00000001;
	_delay_ms(DELAY);
```

```
        PORTB = BCDTABLE[digito2];
        PORTC = 0b00000010;
        _delay_ms(DELAY);
        PORTB = BCDTABLE[digito3];
        PORTC = 0b00000100;
        _delay_ms(DELAY);
        PORTB = BCDTABLE[digito4];
        PORTC = 0b00001000;
        _delay_ms(DELAY);
        PORTB = BCDTABLE[digito1];
        PORTC = 0b00000001;
}


int main(void)
{
        uint16_t contador = 0;
        uint8_t digito1 = 0;
        uint8_t digito2 = 0;
        uint8_t digito3 = 0;
        uint8_t digito4 = 0;
        uint8_t start = 0;


        DDRB = 0xFF; // Seteamos todos los puertos de B como Output
        DDRC = 0x0F; // Seteamos los primero 4 puertos de C como Output
        // Los primeros 4 puertos los utilizamos para definir que segmento mostramos
en ese instante de tiempo

        // Para los pulsadores de STOP, START y RESET.
        DDRD = 0x00;

    while (1)
    {
                // Chequear el RESET.
                // Si el PIND0 esta en 1 (y todos los demas en 0)
                if (debounce(PIND0)) {
                        contador = 0;
                        digito1=0;
                        digito2=0;
                        digito3=0;
                        digito4=0;

                }
                // START
                if (debounce(PIND1)) {
                        start = 1;
                }
                // STOP
                if (debounce(PIND2)) {
                        start = 0;
                }
                if (start) {
                        separarDigitos(&contador, &digito1, &digito2, &digito3,
&digito4);
                        contador++;
```

```
                }
                showDisplays(digito1, digito2, digito3, digito4);
        }

}
```