Universidad Autónoma de Baja California

Facultad de ciencias químicas e ingeniería

Materia.

Microprocesadores y Microcontroladores.

Maestro.

Garcia Lopez Jesus Adan.

Alumno

Gonzalez Cardiel Luis Enrique

Matricula:

1217258

Grupo:

561

Trabajo:

Practica No. 8

26/10/2018

Práctica 8

• Manejo de la sección de E/S del microcontrolador ATmega1280/2560

Objetivo: Mediante esta práctica el alumno analizará la implementación de retardos por software, así como también se familiarizará con la configuración y uso de puertos.

Equipo: - Computadora Personal con AVR Studio y tarjeta T-Juino.
Teoría: 1) Investigación a cerca de ensamblador en línea para GCC.
2) Análisis y cálculo del retardo por SW de la práctica.
3) Teoría sobre puertos de E/S (uC ATmega1280/2560)
4) Técnicas de anti-rebote de botones táctiles.

Descripción:
Implementar un programa en base al Listado 1, el cual revisa el estado del botón, y dependiendo de su duración que está presionado incrementa o decrementa un contador, que a su vez se muestra en el arreglo de Leds; como se muestra en la Fig. 2.

Listado 1:

```
#include <avr/io.h>

// Macros
#define SetBitPort(port, bit) __asm__ ( ?? )
#define ClrBitPort(port, bit) __asm__ ( ?? )

// Press States
#define NOT_PRESSED 0
#define SHORT_PRESSED 1
#define LONG_PRESSED 2

// Prototypes
Void    delay(uint16_t mseg);
void    InitPorts(void);
uint8_t checkBtn(void);
void    updateLeds(void);

// Global variables
uint8_t  globalCounter;
uint32_t millis;

int main(void){
      InitPorts();

      while(1){
            switch(check_Btn()){
                  case SHORT_PRESSED: globalCounter++;
                                      break;
                  case LONG_PRESSED:  globalCounter--;
                                      break;
            }
            updateLeds();
            delay(1);
            millis++;
      }
}
```

Macros a implementar:
1. SetBitPort(port, bit)
Macro que inserta la instrucción de ensamblador **SBI**, mediante inline assembly.

2. ClrBitPort(port, bit)
Macro que inserta la instrucción de ensamblador **CBI**, mediante inline assembly.

Funciones a implementar:
3. void delay(uint16_t mseg);
Función que debe de tardarse **n** ms en retornar, según se especifique en el parámetro de entrada. Con una exactitud de ±5 us.

4. void InitPorts(void);
Inicialización requerida de los puertos utilizados en esta práctica. **PE1** debe dejarse en un nivel alto.

5. uint8_t check_Btn(void);
Retorna el estado del botón, detectando entre NOT_PRESSED, SHORT_PRESSED y LONG_PRESSED. Donde el umbral para una larga duración es cualquiera que sea mayor a 1 s. Ignorando el rebote mecánico que se puede apreciar en la Fig 1.

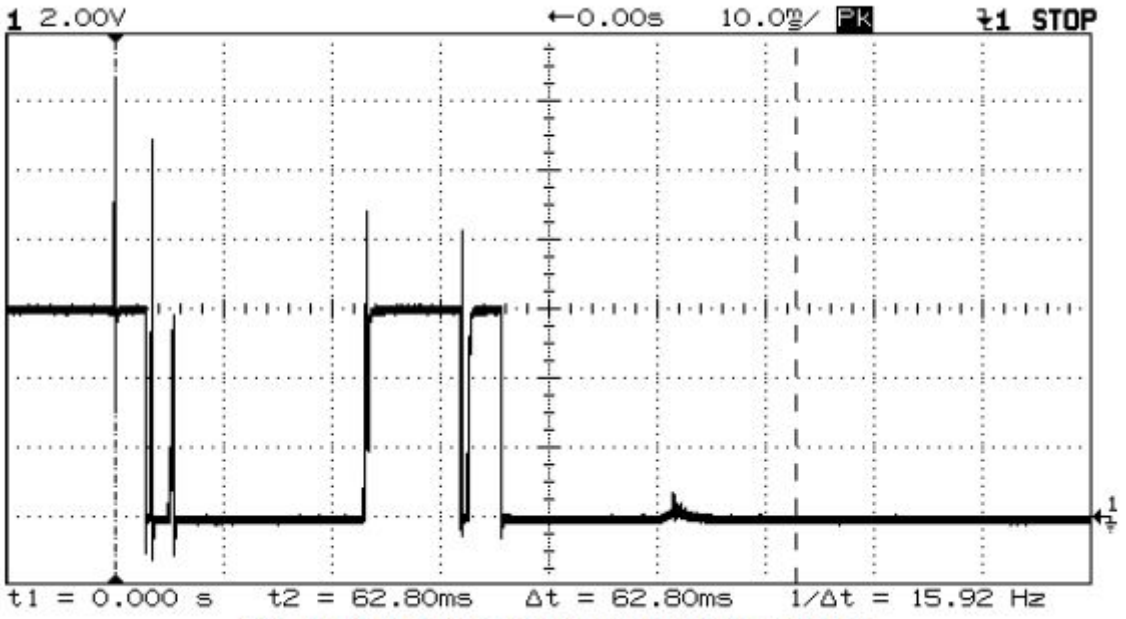


Fig. 1. Ejemplo del rebote mecánico de un botón.

6. void updateLeds(void);

Refleja el valor del contador global en los Leds como se muestra en la Fig. 2. Realizar los ajustes necesarios de tal forma que no se perciba que parpadean.

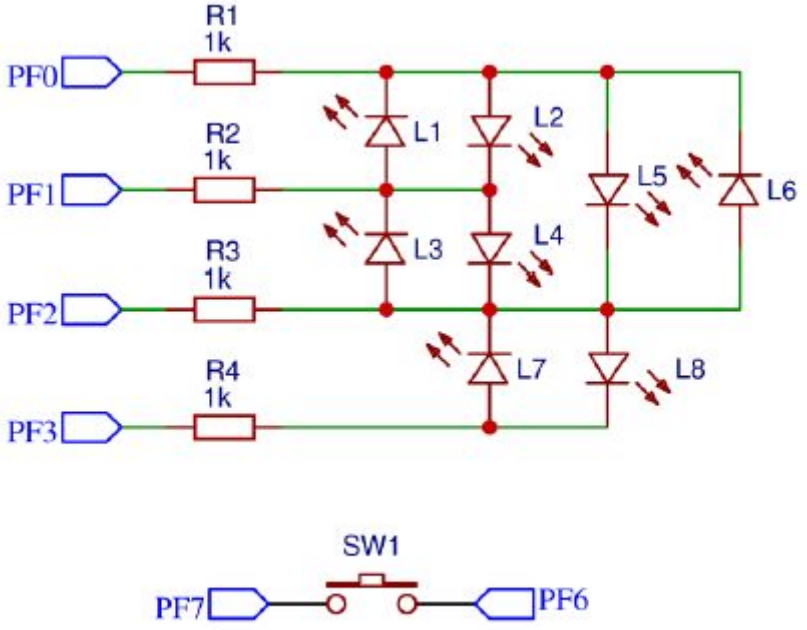


Fig. 2. Esquemático

Teoría

Investigación a cerca de ensamblador en línea para GCC.

Asm extendido.

En ensamblaje básico en línea, solo teníamos instrucciones. En el montaje extendido, también podemos especificar los operandos. Nos permite especificar los registros de entrada, los registros de salida y una lista de registros comprimidos. No es obligatorio especificar los registros a usar, podemos dejar ese dolor de cabeza a GCC y probablemente encajen mejor en el esquema de optimización de GCC. De todos modos el formato básico es:

```
asm ( assembler template
    : output operands                /* optional */
    : input operands                 /* optional */
    : list of clobbered registers    /* optional */
    );
```

La plantilla del ensamblador consta de instrucciones de montaje. Cada operando se describe mediante una cadena de restricción de operandos seguida de la expresión C entre paréntesis. Un signo de dos puntos separa la plantilla del ensamblador del primer operando de salida y otro separa el último operando de salida de la primera entrada, si corresponde. Las comas separan los operandos dentro de cada grupo. El número total de operandos se limita a diez o al número máximo de operandos en cualquier patrón de instrucción en la descripción de la máquina, el que sea mayor.
Si no hay operandos de salida pero sí hay operandos de entrada, debe colocar dos puntos consecutivos alrededor del lugar donde irían los operandos de salida.
Ejemplo:

```
asm ("cld\n\t"
     "rep\n\t"
     "stosl"
     : /* no output registers */
     : "c" (count), "a" (fill_value), "D" (dest)
     : "%ecx", "%edi"
     );
```

Ahora, ¿qué hace este código? La línea anterior llena los fill_value counttiempos hasta la ubicación señalada por el registro edi. También le dice a gcc que, los contenidos de los registros eaxya edino son válidos. Veamos un ejemplo más para aclarar las cosas.

```
int a=10, b;
asm ("movl %1, %%eax;
     movl %%eax, %0;"
    :"=r"(b)        /* output */
    :"r"(a)         /* input */
    :"%eax"         /* clobbered register */
    );
```

Aquí lo que hicimos es hacer que el valor de 'b' sea igual al de 'a' usando instrucciones de ensamblaje. Algunos puntos de interés son:

- "b" es el operando de salida, referido por% 0 y "a" es el operando de entrada, referido por% 1.
- "r" es una restricción en los operandos. Veremos las restricciones en detalle más adelante. Por el momento, "r" le dice a GCC que use cualquier registro para almacenar los operandos. La restricción del operando de salida debe tener un modificador de restricción "=". Y este modificador dice que es el operando de salida y es de solo escritura.
- Hay dos% prefijados al nombre del registro. Esto ayuda a GCC a distinguir entre los operandos y los registros. Los operandos tienen un solo% como prefijo.
- El registro obstruido% eax después del tercer colon le dice a GCC que el valor de% eax debe modificarse dentro de "asm", por lo que GCC no usará este registro para almacenar ningún otro valor.

Cuando se complete la ejecución de "asm", "b" reflejará el valor actualizado, ya que se especifica como un operando de salida. En otras palabras, se supone que el cambio realizado en "b" dentro de "asm" se refleja fuera del "asm".
Ahora podemos ver cada campo en detalle.

Análisis y cálculo del retardo por SW de la práctica.

```
void delay(uint16_t mseg){                  //7 instrucciones de entrada
    uint16_t i=0;                 //2
    while(mseg!=0){                //-1
        i= 1999;                  //4
        asm("nop");                 //1
        while(i!=0){                //4m-1
            i--;                  //2
           asm("nop");           //1m
           asm("nop");
           }
         mseg--;                //2
        }

}                                            //5  de salida
```

$1ms = 16000$

$8 + 8m = 16000$

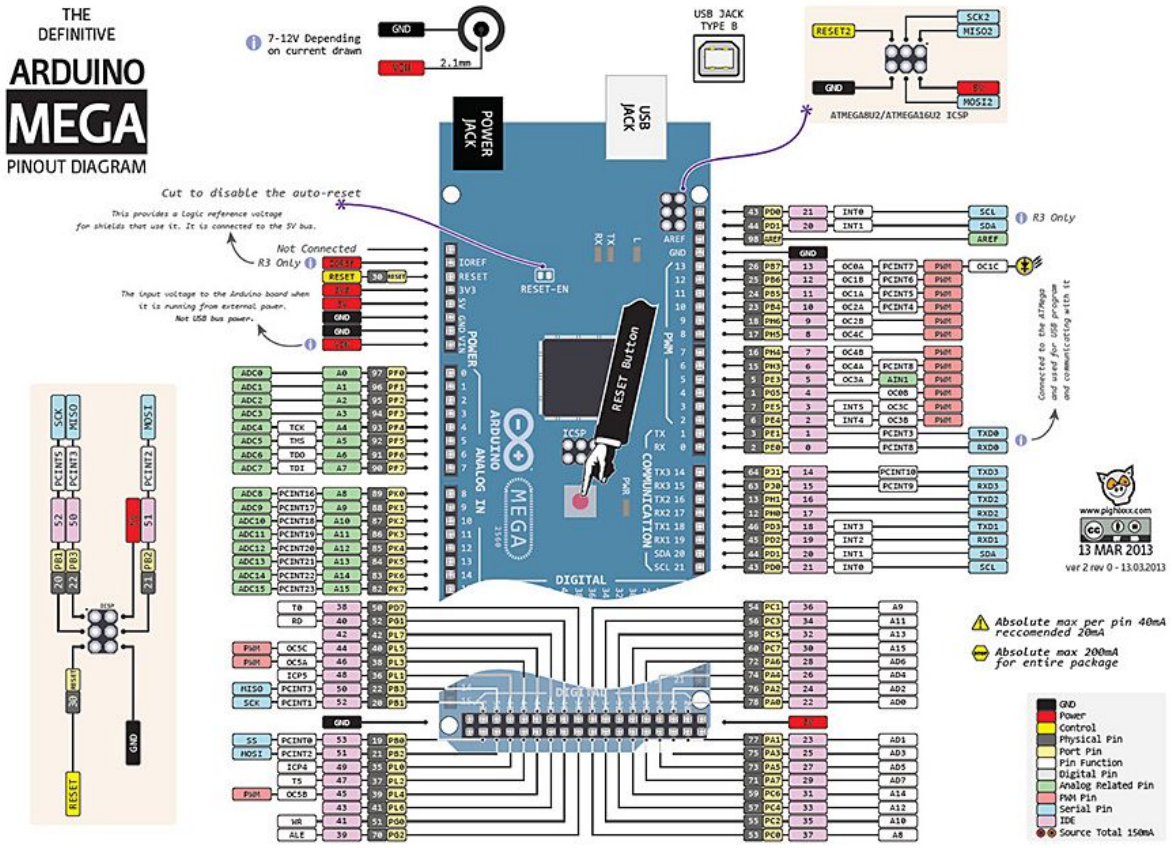
$m = \frac{16000-8}{8}$ = 1999

Se noto que ala hora de entrar al 2do while se consumian 2 ciclos mas por eso se le incrementaron 2 mas.

Teoría sobre puertos de E/S (uC ATmega1280/2560)

Puerto	Pines en la placa	Posicion	Observaciones
PortA	22,23,24,25,26,27,28,29	PA0 a PA7	
PortB	13,12,11,10,50,51,52,53	PB0 a PB7	
PortC	37,36,35,34,33,32,31,30	PC0 a PC7	
PortD	21,20,19*,18*,x,x,x,38	PD0 a PD7	* = PD2 y PD3, son pines de Rx1 y Tx1, cuidado al asignarles valor.
PortE	0*,1*,x,5,2,3,x,x	PE0 a PE7	x= puertos reservados PE2, PE6 y PE7. * = PE0 y PE1, son pines de Rx0 y Tx0, cuidado al asignarles valor.
PortF	0,1,2,3,4,5,6,7	PF0 a PF7	Pines Análogos.
PortG	x,x,39,x,4,x,	PG0 a PG5	x= Puertos reservados PG0, PG1, PG3 yPG5. PG6 y PG7 no aparecen.
PortH	17*,16*,x,6,7,8,9,x	PH0 a PH7	x= puertos reservados PH2 y PH7. * = PH0 y PH1, son pines de Rx2 y Tx2, cuidado al asignarles valor.
PortJ	15*,14*,x,x,x,x,x	PJ0 a PJ6	x= puertos reservados PJ2 al PJ6. PG7 no aparece. * = PJ0 y PJ1, son pines de Rx3 y Tx3, cuidado al asignarles valor.
PortK	8,9,10,11,12,13,14,15	PK0 a PK7	Pines Análogos.
PortL	49,48,47,46,45,44,43,42	PL0 a PL7	

By neospark2006



Registros

Atmel AVR es un microcontrolador de 8 bits. Todos sus puertos son de 8 bit de ancho. Cada puerto tiene 3 registros asociados con cada uno con 8 bits. Cada bit en esos registros configura los pines de un puerto particular. Bit0 de estos registros está asociado con Pin0 del puerto, Bit1 de estos registros está asociado con Pin1 del puerto, ... Y como sabio para otros bits.

Estos tres registros son los siguientes:

(x puede reemplazarse por A, B, C, D según el AVR que está utilizando)

- Registro DDRx
- Registro PORTx
- Registro PINx

Registro DDRx

DDRx (Registro de dirección de datos) configura la dirección de datos de los pines del puerto. Significa que su configuración determina si los pines del puerto se utilizarán para entrada o salida. Escribir 0 a un bit en DDRx hace que el pin del puerto correspondiente sea una entrada, mientras que escribir 1 a un bit en DDRx hace que el pin correspondiente del puerto sea la salida.

Ejemplo:

- para hacer todos los pines del puerto A como pines de entrada:
- DDRA = 0b00000000;
- para hacer todos los pines del puerto A como pines de salida:
- DDRA = 0b11111111;
- para reducir el nibble del puerto B como salida y el nibble más alto como entrada:
- DDRB = 0b00001111;

Registro PINx

PINx (Port IN) se utiliza para leer datos de los pines del puerto. Para leer los datos del pin del puerto, primero debe cambiar la dirección de los datos del puerto a la entrada. Esto se hace estableciendo bits en DDRx a cero. Si el puerto se convierte en salida, entonces la lectura del registro PINx le proporcionará los datos que se han emitido en los pines del puerto.

Ahora hay dos modos de entrada. O puede usar los pines del puerto como entradas indicadas por tri o puede activar la recuperación interna. Se explicará en breve.

Ejemplo:

- para leer los datos del puerto A.

```
DDRA = 0x00;    // Establecer el puerto a como entrada
x = PINA;       // Leer los contenidos del puerto a
```

Registro PORTx

PORTx se utiliza para dos propósitos.

1) Para generar datos: cuando el puerto se configura como salida

Cuando establece bits en DDRx en 1, los pines correspondientes se convierten en pines de salida. Ahora puede escribir datos en bits respectivos en el registro PORTx. Esto cambiará inmediatamente el estado de los pines de salida según los datos que haya escrito.

En otras palabras, para enviar datos a los pines del puerto, debe escribirlos en el registro PORTx. Sin embargo, no se olvide de establecer la dirección de los datos como salida.

Ejemplo:

- **para generar datos 0xFF en el puerto b**
    - ```
      DDRB = 0b11111111;      // establece todos los pines del puerto b como salidas
      PORTB = 0xFF;           // escribir datos en el puerto
      ```

- para generar datos en la variable x en el puerto a
    - ```
      DDRA = 0xFF;             // hacer puerto a como salida
      PORTA = x;               // variable de salida en el puerto
      ```

- para generar datos en solo el bit 0 del puerto c
    - ```
      DDRC | = 0b00000001;     // establece solo el 0º pin del puerto c como salida
      PORTC | = 0b00000001;       // hazlo alto.
      ```

2) Para activar / desactivar las resistencias de extracción: cuando el puerto se configura como entrada
Cuando configura los bits en DDRx a 0, es decir, hace que los pines del puerto sean entradas, los bits correspondientes en el registro PORTx se utilizan para activar / desactivar los registros de recuperación asociados con ese pin. Para activar el registro de recuperación, establezca el bit en PORTx en 1, y para desactivar (es decir, para hacer que el puerto pin tri establecido) establezca en 0.
En el modo de entrada, cuando está habilitado el pull-up, el estado predeterminado del pin se convierte en '1'. Entonces, incluso si no conecta nada para fijar y si intenta leerlo, se leerá como 1. Ahora, cuando conduzca externamente ese contacto a cero (es decir, conecte a tierra / o hacia abajo), solo entonces se leerá como 0.

Sin embargo, si configura pin como tri-state. Luego el pin entra en estado de alta impedancia. Podemos decir que ahora está simplemente conectado a la entrada de algún OpAmp dentro de la unidad uC y ningún otro circuito lo está conduciendo desde la unidad uC. Así, el pin tiene una impedancia muy alta. En este caso, si el pin se deja flotando (es decir, se mantiene desconectado), incluso una pequeña carga estática presente en los objetos circundantes puede cambiar el estado lógico del pin. Si intenta leer el bit correspondiente en el registro pin, no se puede predecir su estado. Esto puede hacer que su programa se vuelva loco, si depende de la entrada de ese pin en particular.
Por lo tanto, mientras toma las entradas de los pines / usa microinterruptores para tomar la entrada, siempre habilite las resistencias de arranque en los pines de entrada.
NOTA: mientras se usa el ADC en el chip, los pines del puerto ADC deben configurarse como entrada de tres estados.
Ejemplo:

- para hacer un puerto como entrada con dominadas habilitadas y leer datos del puerto a
      ```

```
DDRA = 0x00;        // hacer el puerto a como entrada
PORTA = 0xFF;       // habilitar todos los pull-ups
y = PINA;           // leer los datos del puerto a los pines
```

- para hacer el puerto b como tri entrada indicada

```
DDRB = 0x00;        // hacer el puerto b como entrada
PORTB = 0x00;        // deshabilita los pull-ups y haz que sea tri estado
```

- para reducir el nibble del puerto a como salida, mayor nibble como entrada con pull-ups habilitados

```
DDRA = 0x0F;        // plumín inferior> salida, plumín superior> entrada
PORTA = 0xF0;        // plumilla inferior> establecer pines de salida en 0,
                     // plumilla superior> habilitar pull-ups
```

Técnicas de anti-rebote de botones táctiles.

Una de las técnicas antirebote es dejar pasar un determinado tiempo la presionar el botón ya que este genera ruido al tocar las láminas de dicho botón se recomendó realizar un delay de 60 ms para que se estabilice el botón y poder leer el dato verdadero.También se recomendó poner un delay de 60 ms ya que al soltar el botón se generan otro picos esto no se implementó en el diseño ya que no hubo problemas al soltar el botón.

Conclusión:

En esta práctica se puede observar como mediante Software se puede quitar el ruido a un push-Button, también como mediante los GPIOs al configurarlos como Pull-Up se puede generar una entrada Virtual como la que simulamos con el Push-Button, se utilizaron también máquinas de estados para simular que nuestro sistema está trabajando con hilos al no dejar de hacer una actividad en un tiempo máximo, las actividades que realizaba eran mínimas y nunca se le dio un delay de más de 1 ms por lo tanto todo lo ejecutaba demasiado rápido y siempre estaba realizando una acción.

Bibliografía

https://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html
https://www.nongnu.org/avr-libc/user-manual/inline_asm.html
https://saber.patagoniatec.com/wp-content/uploads/2014/06/Mega2-900.jpg
http://www.elecrom.com/avr-tutorial-2-avr-input-output/