Universidad Autónoma de Baja California

Facultad de ciencias quimicas e ingenieria

Microcontroladores y microprocesadores

Práctica 8. Manejo de la sección de E/S del microcontrolador ATmega1280/2560

Alumno

Caudillo Sanchez Diego

Matricula

1249199

Grupo

521

Docente

Jesus Adan Garcia

Fecha de entrega

20/Agosto/2020

Objetivo

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

Equipo

- Computadora personal con AVR Studio y tarjeta T-Juino.

Teoria

Investigar acerca de ensamblador en línea y fuera de línea para AVR-GCC

Ensamblador Inline

Es usado para escribir instrucciones en ensamblador en compilador de C. el compilador añadirá la instrucción ensamblador al código que se genere. La forma que tiene para declarar instrucciones ensamblador es la siguiente:

asm(code: output operand list: input operand list: clobber list);

Por ejemplo en la siguiente instrucción: *asm("in %0, %1": "=r" (value) : "I" (PORTD) :);*
Se lee un valor de un puerto (PORTD). Donde:

- La instrucción ensamblador debe declarases como una sola cadena:
  - "In %0, %1" donde: %0 se refiere al primer operando
- La lista de salidas (outputs) debe estar separadas por comas, en el ejemplo solo hay una.
  - "=r (value)"; que hace referencia a *%0*.
- Al igual que las salidas, la entradas también se separan por comas, igual que el punto anterior en este ejemplo se utiliza solo una entrada.
  - "I (PORTD)"; que hace referencia a *%1*.
- El registro clobber, se dejó vacío para este ejemplo. Sin embargo, si utiliza registros que no han pasado como operandos, se necesita informar al compilador sobre esto.

Note que se puede omitir el ultimo parametro con su respectivo signo de dos puntos (:). Pero es importante que los restantes no omitan su separación de dos puntos (:), como las instrucciones que no utilizan parámetros (Por ejemplo: *asm(cli : :);)*.

También es importante notar que por defecto, el compilador tiene su modo de optimización encendido, por lo que antes de ensamblar la instrucción, verifica y hace modificaciones, si es necesario para la optimización del programa, al código. Por ejemplo una variable que nunca se usó, o que se dejó usar en un punto del código. Para evitar esto, se puede usar el atributo *volatile*. Por ejemplo, usando el ejemplo anterior:

asm volatile ("in %0, %1" : "=r" (value): "I" (PORTD) :);

Cada operación de entrada y salida es descrita por un carácter seguido por una expresión en C en paréntesis.

Constraint	Used for	Range
a	Simple upper registers	r16 to r23
b	Base pointer registers pairs	y,z
d	Upper register	r16 to r31
e	Pointer register pairs	x,y,z
G	Floating point constant	0.0
I	6-bit positive integer constant	0 to 63
J	6-bit negative integer constant	-63 to 0
K	Integer constant	2
L	Integer constant	0
l	Lower registers	r0 to r15
M	8-bit integer constant	0 to 255
n	16-bit integer constant?	
N	Integer constant	-1
O	Integer constant	8, 16, 24
P	Integer constant	1
q	Stack pointer register	SPH:SPL
r	Any register	r0 to r31
t	Temporary register	r0
v	32-bit integer constant?	
w	Special upper register pairs	r24, r26, r28, r30
x	Pointer register pair X	x (r27:r26)
y	Pointer register pair Y	y (r29:r28)
z	Pointer register pair Z	z (r31:r30)

Las siguientes tablas muestran los mnemónicos del AVR. los cuales requieren operandos, y se caracter que le corresponde.

Mnemo	Constraints
adc	r,r
add	r,r
adiw	w,I
and	r,r
andi	d,M
asr	r
bclr	I
bld	r,I
brbc	I,label
brbs	I,label
bset	I
bst	r,I
cbi	I,I

cbr	d,I
com	r
cp	r,r
cpc	r,r
cpi	d,M
cpse	r,r
dec	r
elpm	t,z
eor	r,r
in	r,I
inc	r
ld	r,e
ldd	r,b
ldi	d,M
lds	r,label

lpm	t,z
lsl	r
lsr	r
mov	r,r
mul	r,r
neg	r
or	r,r
ori	d,M
out	I,r
pop	r
push	r
rol	r
ror	r
sbc	r,r
sbci	d,M

sbi	I,I
sbic	I,I
sbiw	w,I
sbr	d,M
sbrc	r,I
sbrs	r,I
ser	d
st	e,r
std	b,r
sts	label,r
sub	r,r
subi	d,M
swap	r

Análisis y cálculo del retardo por SW de la práctica.

Para la realización de esta práctica es necesario crear retardos por software, lo que implica hacer cálculos para lograr ese retardo que solicitan las instrucciones.

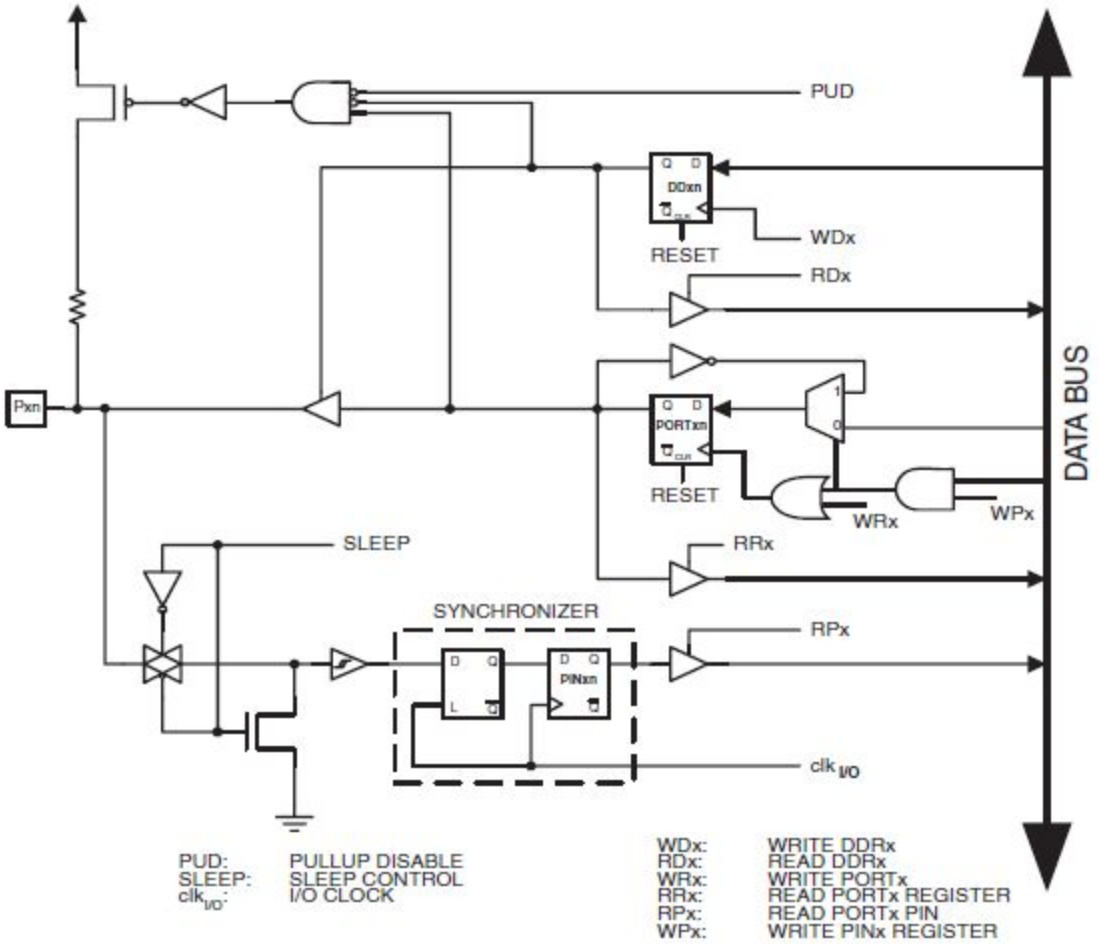
- **Retardo de 250ms**
  - Primeramente se necesita saber que velocidad tiene el cristal que se está utilizando. En el caso del ATmega2560, se trabaja a 16MHz, por tanto $\frac{1}{16MHz}$ que nos da como resultado, 62.5 ns (nanosegundos) para ejecutar una sola instrucción. Posterior a esto, podemos simplemente dividir el tiempo del *retardo solicitado*, con el *tiempo de ejecución por instrucción*, para obtener el número de instrucción necesarias para llegar al retardo que pide la práctica (250 ms). Por tanto, obtenemos lo siguiente: $\frac{250ms}{62.5ns}$ = 4,000,000 de instrucciones para alcanzar el tiempo de retardo requerido. Una vez finalizado este análisis, podemos seguir con la parte de programación, para determinar qué valor toman los registros y cómo se va estructurar el programa para obtener el retardo exacto, tomando en cuenta las instrucciones de *call* y *ret*.
  - En el siguiente código, podemos observar cómo se estructura el procedimiento para los retardos (varía depende del tiempo del retardo). En los registros A, B y C (R16, R17 y R18 respectivamente) inicialmente son desconocidos. Para encontrar sus valores se necesita volver hacer otro análisis de código y determinar la ecuación para estas instrucciones.
  -

```
;R16 = A
;R17 = B
;R17 = C
DELAY250MS:
    LDI A, val1     ;1
L1:
    LDI B, val2     ; 1*A
L2:
    LDI C, val3     ; 1*AB
L3:
    NOP             ; 1*ABC
    NOP             ; 1*ABC
    NOP             ; 1*ABC
    DEC C           ; 1*ABC
    BRNE L3         ; (2C -1)*AB
    DEC B           ; 1*AB
    BRNE L2         ; (2B -1)*A
    DEC A           ; 1*A
    BRNE L1         ; (2A - 1)
    RET             ; 5
```

- Con el análisis anterior podemos resolver el sistema de la siguiente manera
    1. Ecuación inicial
        - 6ABC+3AB+3A+10 = 4,000,000
    2. Dejamos las incógnitas de un lado
        - 6ABC+3AB+3A = 4,000,000 -10
    3. Simplificamos la ecuación, dividiéndola en un número común
        - 2ABC+AB+A = 3,999,990/3 = 1,333,330
    4. Sacamos el factor común de la ecuación.
    5. A(1+B+2BC) = 1,333,330
    6. Ahora se busca un número para esa variable que al dividir no haya residuo. Para A, se le asignó el valor de 151. Recordar que los registros son de 8 bits por lo que solo pueden usarse números del 0 al 255.
        - 1+B+2BC = (1,333,330) / A ; (A=151)
        - B+2BC = 8830-1
    7. Realizar lo mismo que en el paso #7.
        - B(1+2C) = 8829
        - 1+2C = 8829/B ; (B = 109)
        - 2C = 81-1
        - C = 80/2 = 40
    8. Finalmente, tenemos que A=151, B=109 y C=40
    9. Sustituyendo en la ecuación obtenemos que:
        - 6(151*109*40)+3(151*109)+C(151) + 10 = 4,000,000.

Teoria sobre puertos de E/S (μC ATmega1280/2560)

Los puertos de E/S del microcontrolador tienen verdadera funcionalidad del Escribir-Modificar-Leer cuando se utilizan como puertos digitales generales. Lo que significa que la dirección de un solo bit de un puerto puede ser cambiada sin intencionalmente cambiar la dirección de cualquier otro pin con las instrucciones de SBI y CBI. Lo mismo implica al modificar el valor de la dirección o la habilitación de resistores pull-up internos en caso de que sean configurados como entrada.



Cada pin del puerto consiste de tres bits de registro.

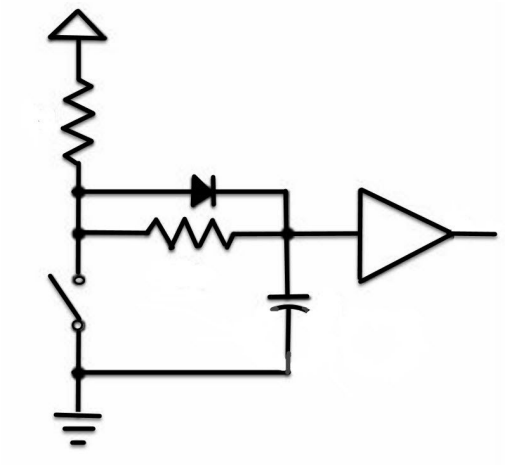
- **DDxn** afecta la dirección del pin, si es uno, Pxn es configurado como salida. Caso contrario se configura como entrada.
- **Pxn/PORTxn** es escrito un uno al ser de entrada, el resistor pull-up se activa. Para desactivarlo (Hi-Z) cuando se activa un reset. Si es escrito uno cuando es configurado como pin de salida, la salida es de uno, de ser cero la salida es cero.
- **PINxn** regresa los valores de entrada en las entradas de PORTx. Esto permite que se puedan utilizar configuración diferentes de los pines en un solo puerto, dando flexibilidad.

DDxn	PORTxn	PUD (in MCUCR)	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)

Técnicas de anti-rebote de botones táctiles.

Mediante Hardware

- Resistor capacitor: este circuito es utilizado para eliminar el rebote con un capacitor. Cuando el switch está abierto, el voltaje a través del capacitor que inicialmente cero ahora se dirige al resistor R1 y R2.



Mediante Software

- Esta solución involucra el activar un procedimiento al momento de presionar el botón por software, justo después de entrar se tiene que realizar un delay, por lo que se ignoran señales de rebote durante este periodo de tiempo (en milisegundos), al momento de que se acabe este delay. Se puede revisar de nuevo el estado del botón de ser necesario.

Descripción

Implementar un programa el cual es un simple juego de “Simón dice” mediante LEDs y botones. Tiene como entrada 8 botones y 8 LEDs de salida; los cuales se deberán conectar como se muestra en la Fig. 2.

1. Simón prenderá cualquier LED de forma aleatoria durante 1 segundo.

2. El Jugador deberá de replicar la acción presionando el botón correspondiente al LED que se activó.
3. La “memoria” de Simón es solo de 6 acciones, así que lo anterior se repite solo 6 veces agregando una acción/indicador más en cada iteración; concatenando las acciones anteriores en cada nueva acción.
4. Si el jugador replica correctamente las acciones en las 6 iteraciones, entonces ha ganado, de lo contrario al equivocarse en alguna de las secuencias ha perdido y se termina el juego, esperando a que se presione cualquier botón para reiniciar el juego.

Las secuencias que se deberán mostrar sobre los LEDs son las siguientes:

- *Para el caso de Ganar:*
  Secuencia de walking-cero del MSB al LSB, actualizándose cada 100 ms (S_0=0b11111111, S_1=0b01111111, …, S_8=0b11111110, y repetir).
- *Para el caso de Fallar:*
  Secuencia donde medio *nibble* está apagado y el otro prendido, alternando cada 250 ms.

Al iniciar el juego mostrar cualquier de las secuencias anteriores para esperar a que el Jugador presiona cualquier botón para iniciar la partida.

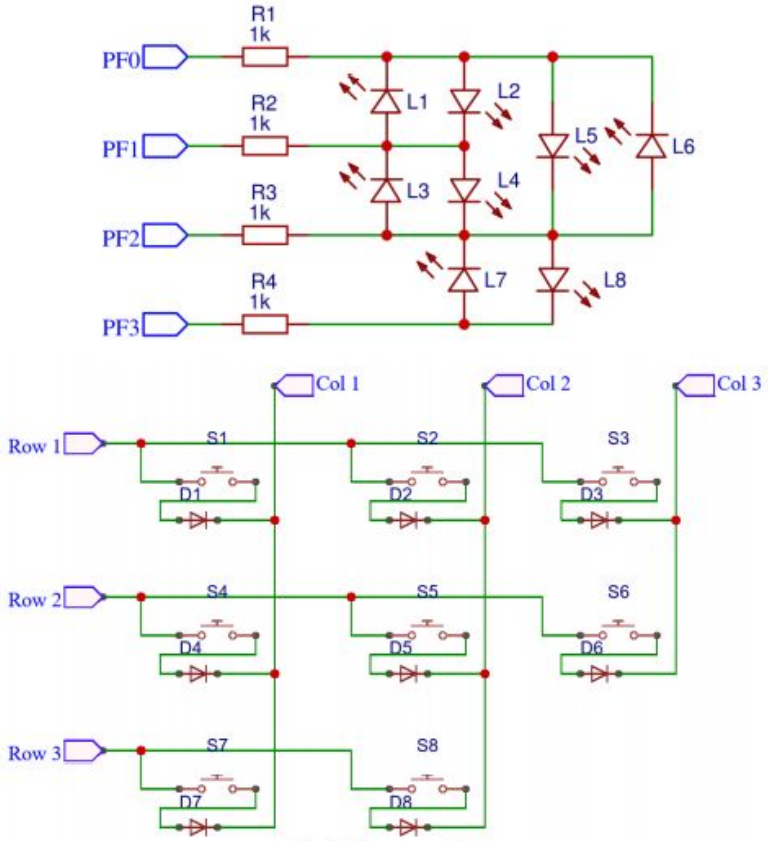


Fig. 2. Esquemático

Comentarios y conclusiones.

Con la conclusión de esta práctica, se aprendió a utilizar los puertos de Entrada/Salida del AVR. Que son de gran utilidad y más fáciles de manipular al momento de programar.

Un pequeño detalle que no tome en cuenta fue es cuando se programan los pines de entrada y que se activa la resistencia pull-up interna, por defecto el pin lee un valor de 1 lógico, lo que se debe de tomar en consideración cuando se esperen valores diferentes de 1.

Bibliografía y referencias.

Evidencia de la practica: **https://youtu.be/43mwJS0sL7g**

Kipp, H. (2002). G*CC-AVR Inline Assembler Cookbook*

Switch Bounce and How to Deal with It**.** *switch debounce*

AVR 1280/2560 User Manual