Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0624 Laboratorio de Microcontroladores

EIE
Escuela de
Ingeniería Eléctrica

GPIO, interrupciones, timers y el ATtiny4313

MSc. Marco Villalta Fallas - marco.villalta@ucr.ac.cr

II Ciclo 2022

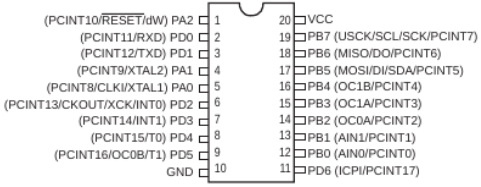
Intro: ATtiny4313

ATtiny4313

Descripción general

- Microcontrolador AVR de 8 bits
- Arquitectura RISC/Harvard.
- 2/4Kb Flash, 128/258 bytes de SRAM y 128/258 bytes de EEPROM
- 18 GPIOs agrupados en 3 puertos
- Timer/Counters de 8 y 16 bits.
- 4 canales PWM y comparador analógico
- USI, USART

PDIP/SOIC



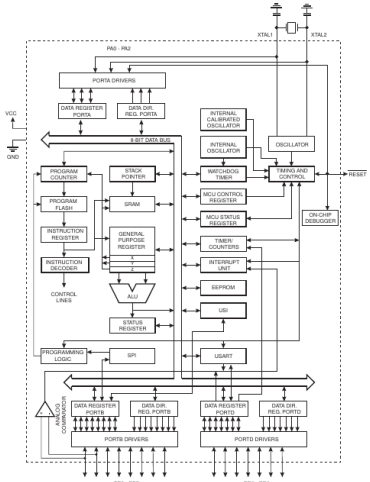
ATtiny4313

Arquitectura

Bloques importantes:

- CPU (No se muestra)
- RAM
- ROM/Flash
- EEPROM
- Timers
- Convertidores A/D
- Puertos I/O
- Comunicaciones (Seriales)

Figure 2-1. Block Diagram



GPIOs

GPIOs

Lectura/Escritura

- DDxn bit pertenece al registro DDRx: 1 para salida y 0 para entrada.
- PORTxn bit pertenece al registro PORTx. Cuando esta configurado como entrada un 1 activa la resistencia pull-up, para apagar la resistencia se debe configurar pin como salida.
- Cuando PORTxn esta configurado como salida: se escribe un 1 logico el pin se pone en alto , un 0 logico el pin se pone en bajo.
- Escribiendo un 1 a PINxn cambia el valor de PORTxn independiente del valor en DDRxn
- Independiente de la configuración de DDxn, el pin se puede leer en el bit PINxn
- Cuando se leen por interrupciones se debe configurar el registro PCMSKn y GIMSK

GPIOs

Registros

En el ATtiny4313 cada puerto/pin consiste de tres bits de registro: DDxn, PORTxn y PINxn (*x* es el puerto y *n* el pin)

10.3.5 PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x18 (0x38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.3.6 DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.3.7 PINB – Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

GPIOs

Registros de interrupción

GIMSK – General Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x3B (0x5B)	INT1	INT0	PCIE0	PCIE2	PCIE1	–	–	–	GIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
//Ejemplo de modificacion de registro
GIMSK  |= (1 << PCIE2);//Se habilita la interrupcion por PCIE2
```

Interrupciones

Interrupciones

- Polling: Técnica de software donde se consulta el estado de un periférico, no es muy deterministico.
- En lugar de hacer polling para ver si hay nuevos datos, es más eficiente que el periférico indique disposición de nuevos datos.
- Una interrupción es una notificación al CPU que un evento ha sucedido.
- Pueden ser disparadas por varios eventos/periféricos como I/Os, ADC, software, timers, etc.
- Deben ser atendidos (en **rutinas cortas**)

Como funcionan las interrupciones

1. Cuando una interrupciones se dispara la operacion del programa se detiene.
2. El estado del programa se guarda.
3. Procesador revisa espacio de memoria de registros de interrupción (Vectores de interrupción).
4. Se ejecuta la ISR. Cada interrupción tiene una ISR (Interrupt service routine)
5. Prosigue programa.

Vector de interrupciones ATtiny4313

Table 9-1. Reset and Interrupt Vectors

Vector No.	Program Address	Label	Interrupt Source
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	INT1	External Interrupt Request 1
4	0x0003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	0x0004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	0x0005	TIMER1 OVF	Timer/Counter1 Overflow
7	0x0006	TIMER0 OVF	Timer/Counter0 Overflow
8	0x0007	USART0, RX	USART0, Rx Complete
9	0x0008	USART0, UDRE	USART0 Data Register Empty
10	0x0009	USART0, TX	USART0, Tx Complete
11	0x000A	ANALOG COMP	Analog Comparator
12	0x000B	PCINT0	Pin Change Interrupt Request 0
13	0x000C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x000D	TIMER0 COMPA	Timer/Counter0 Compare Match A
15	0x000E	TIMER0 COMPB	Timer/Counter0 Compare Match B
16	0x000F	USI START	USI Start Condition
17	0x0010	USI OVERFLOW	USI Overflow
18	0x0011	EE READY	EEPROM Ready
19	0x0012	WDT OVERFLOW	Watchdog Timer Overflow
20	0x0013	PCINT1	Pin Change Interrupt Request 1
21	0x0014	PCINT2	Pin Change Interrupt Request 2

Interrupciones y GPIOs

- Interrupciones de GPIOs van del PCINT17...0
- PCIE2 se dispara si PCINT17...11 estan cambian.
- PCIE1 se dispara si PCINT10...8 estan cambian.
- PCIE0 se dispara si PCINT7...0 estan cambian.
- INT0 y INT1 se disparan por flanco postivo, negativo o un nivel bajo. (Ver MCUCR pg51).
- Si se quieren usar habilitar la interrupciòn global (Usar función sei()).

```
ISR(PCINT2_vect) //SIGNAL tmb sirve pero para mi ISR tiene mas sentido
{
        //Intrucciones de la rutina de interrupcion
}
```

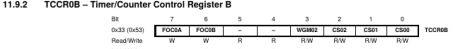
Timers

Timers

- Los timer/counter permiten medir el intervalo de tiempo o contar eventos internos o externos
- Pueden generar interrupciones
- Velocidad de cuenta en función de la fuente de reloj y la configuración de escala (*prescaler*).

Que es un prescaler?

- Es un circuito contador/divisor utilizado para reducir una señal eléctrica de alta frecuencia a una frecuencia menor por medio de una división
- Típicamente utiliza de entrada el reloj de sistema y lo divide para alimentar a un timer
- Los valores de división se configuran y son valores potencias de 2.
- Existen diferentes circuitos prescaler dependiendo del periférico.
- En el ATtiny4313 el prescaler para el Timer 0 se configura con el registro TCCR0B, con los Bits 2:0. (Ver pg 85 y tabla 11-9)



11.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Sistema de reloj

- Gran mayoría de microcontroladores modernos admiten diferentes fuentes de reloj.
- Fuentes de reloj pueden ser internas o externas.
- En El ATtiny4313 se puede seleccionar un reloj externo, utilizar oscilador interno RC a 4 Mhz o oscilador interno RC a 8 Mhz.
- Configuración por defecto: Oscilador interno RC a 8 Mhz con división por 8. Reloj de sistema de 1Mhz.

Timer/Counter0 del ATtiny4313

- Timer0 es el contador de 8 bits en el ATtiny4313
- Se utilizan los registros TCCR0A, TCCR0B, OCR0A, OCR0B, TIMSK y TIFR para operarlo.
- Se puede leer el estado por polling o con interrupciones.

Timer/Counter0 del ATtiny4313

Registro TCCR0A

- Registro de control A para el contador 0
- Se configura funcionalidad del pin I/O conectado a OC0A y/o OC0B.
- Se establecen modos de comparación. (Normal, PWM, CTC)

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
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	

Timer/Counter0 del ATtiny4313

Registro TCCR0B

- Registro de control B para el contador 0
- Configuración forzadas de comparación
- Se selecciona fuente de reloj para el Timer/Counter 0

11.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter0 del ATtiny4313

Registro OCR0A

- Valor con el que se compara la cuenta del Counter0.
- Existe tambien el registro OCR0B que sirve para tener otro valor de comparacion.

11.9.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x36 (0x56)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

Timer/Counter0 del ATtiny4313

Registro TIFR

- Registro de banderas por interrupción del Timer/Counter 0
- Indicación de fin de cuenta. Se lee si el tiempo medido se ha completado. **(TOV0)**
- Se debe resetear bandera por interrupción si se utiliza método por polling. **(TOV0)**
- Tiene otras banderas como OCF0A y OCF0B que indican interrupcion por comparacion.

11.9.7 TIFR – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x38 (0x58)	TOV1	OCF1A	OCF1B	–	ICF1	OCF0B	TOV0	OCF0A	TIFR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter0 del ATtiny4313

Método por interrupción

- Timer0 es el contador de 8 bits en el ATtiny4313
- Se utilizan los registros TCCR0A, OCR0A, OCR0B,TCCR0B y TIFR para operarlo.
- En caso de utilizar interrupciones se debe modificar adicionalmente el registro TIMSK, modificar el bit de interrupción en TIFR.
- Se pueden tener interrupciones por comparacion con OCR0A,OCR0B y overflow.
- Habilitar la interrupción global (Usar función sei());
- Se debe atender la interrupción con la ISR asignada:

```
ISR(TIMER0_COMPA_vect) //ISR es la macro que atiende interrupciones
{
    // Instrucciones para atender interrupcion
}
```

Watchdog Timer

- Un watchdog timer prove una forma controlada/delicada para recuperarse de un problema de sistema
- Por ejemplo se utiliza en caso de un problema de HW o un loop infinito.
- Si es habilitado se debe resetear periódicamente.
- En caso de no resetearse se dispara interrupción o el uC se reinicia.

Hola mundo

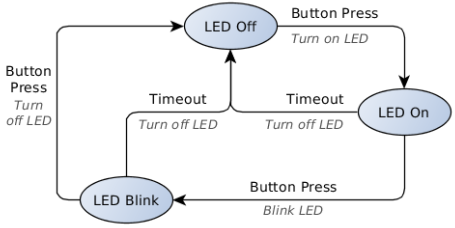
Hola ATtiny

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

int main(void)
{
  DDRB = 0x08; //Configuracion del puerto

  //Parpadear
  while (1) {
    PORTB = 0x00; _delay_ms(500); //Tambien se puede hacer PORTB &= ~(1 << PB3);
    PORTB = 0x08; _delay_ms(500); //Tambien se puede hacer PORTB |=  (1 << PB3);
  }
}
```

FSM

FSM

- Una máquina de estados tiene un número finito de estados para un sisstema determinado.
- Hacen al código más eficiente, más fácil de depurar y ayuda a organizar el flujo del programa.
- Promueven buenas técnicas de diseño de firmware.
- Permiten crear un sistema manejado por eventos que puede cambiar la respuesta a entradas basadas en el estado.



Estructura de una FSM

- Una FSM tiene un conjunto de entradas, salidas y estados conocidos.
- Entradas: Cualquier evento que requiere el sistema para generar una salida o cambiar su comportamiento.
- Transiciones de estado: La flecha en el diagrama de flujo representa una transición de estado.
- Salidas: Acciones necesarias que el sistema debe realizar como respuesta a una entrada.
- Estados: Un estado indica que debe realizar el sistema cuando un evento sucede.

Estado	Entrada	Salida
LED Off	Timeout	Ninguna
LED On/ LED Blink	Timeout	Apagar LED
LED Off	Boton presionado	Encender LED
LED On	Boton presionado	Iniciar parpadeo LED
LED Blink	Boton presionado	Apagar LED

Programando FSM en C

Para programar en C una FSM existen varias formas.

- Instrucciones if/else
- Evaluacion del estado en un case/switch
- Con structs/enums
- Utilizando lookup tables
- Con punteros a funciones

Ejemplo parcial con if/else

```
if(estado==led_off){
    if(button_press){

            encender led;          // Salida
            estado = led_on;  // Transicion de estado
    }
    if(timeout)
            estado = led_off;                   // hacer nada
}
else if(estado==led_on){
    if(button_press){
            parpadear led;              // Salida
            estado = led_blink;   // Transicion de estado
    }
    if(timeout){
            apagar led;            // Salida
            estado = led_off;      // Transicion de estado
    }
}
else if(estado==led_blink){
    if(button_press){
            apagar led;            // Salida
            estado = led_off; // Transicion de estado
    }
    if(timeout){
            apagar led;            // Salida
            estado = led_off; // Transicion de estado
    }
}
```

Ejemplo parcial con switch/case

```
switch(estado){
    case led_off:
        switch(system_input){
            case button_press:
                encender led;          // Salida
                estado = led_on;  // Transicion de estado
                break;
            case timeout:
                break;                    // hacer nada
        }
        break;
    case led_on:
        switch(entrada){
            case button_press:
                parpadear led;              // Salida
                estado = led_blink;   // Transicion de estado
                break;
            case timeout:
                apagar led;            // Salida
                estado = led_off;      // Transicion de estado
                break;
        }
        break;
    case led_blink:
        switch(entrada){
            case button_press:
                apagar led;            // Salida
                estado = led_off; // Transicion de estado
                break;
```

Recomendaciones para el laboratorio

- Leer con calma la hoja de datos del microcontrolador
- Leer documentación de librería/archivos de encabezado
- Ir paso por paso
- Visualice el programa como una máquina de estados
- Preguntar