4. Interrupciones Externas, Temporizadores y PWM

Los microcontroladores AVR tienen una gama amplia de recursos internos, en este capítulo se describen dos recursos fundamentales: las interrupciones externas y los temporizadores. La generación de señales moduladas en ancho de pulso (PWM*, Pulse Width Modulation*) no se realiza con un recurso adicional, las señales PWM se generan con una de las diferentes formas de operación de los temporizadores, no obstante, se dedica la sección 4.3 a esta forma de operación debido al gran número de aplicaciones que pueden desarrollarse con base en PWM.

Debe recordarse que la configuración y uso de los recursos internos de un MCU se realiza por medio de los Registros I/O correspondientes. Por ello, un sistema basado en los recursos trabaja de manera adecuada si se colocan los valores correctos en sus registros de control.

4.1 Interrupciones Externas

Las interrupciones externas sirven para detectar un estado lógico o un cambio de estado en alguna de las terminales de entrada de un microcontrolador, con su uso se evita un sondeo continuo en la terminal de interés. Son útiles para monitorear interruptores, botones o sensores con salida a relevador. En la tabla 4 se describen las interrupciones externas existentes en los AVR bajo estudio, en el ATMega8 se tienen 2 fuentes y en el ATMega16 son 3.

Tabla 4.1 Interrupciones externas y su ubicación en MCUs con encapsulado PDIP

ATMega8			**ATMega16**		
Fuente	Ubicación	Terminal	Fuente	Ubicación	Terminal
INT0	PD2	4	INT0	PD2	16
INT1	PD3	5	INT1	PD3	17
			INT2	PB2	3

Las interrupciones externas pueden configurarse para detectar un nivel bajo de voltaje o una transición, ya sea por un flanco de subida o de bajada. Con excepción de INT2, que sólo puede activarse por flancos. Las interrupciones pueden generarse aun cuando sus respectivas terminales sean configuradas como salidas.

Las transiciones en INT0/INT1 requieren de la señal de reloj destinada a los módulos de los recursos ($clk_{I/O}$, sección 2.8) para producir una interrupción, esta señal de reloj es anulada en la mayoría de los modos de bajo consumo (sección 2.9). Por el contrario, un nivel bajo en INT0/INT1 y las transiciones en INT2 no requieren de una señal de reloj para producir una interrupción, puede decirse que son eventos asíncronos, por lo que éstos son adecuados para despertar al microcontrolador, sin importar el modo de reposo.

4.1.1 Configuración de las Interrupciones Externas

La configuración de INT0 e INT1 se define en el registro **MCUCR** (*MCU Control Register*), los 4 bits más significativos de este registro están relacionados con los modos de bajo consumo de energía y fueron descritos en la sección 2.9, los 4 bits menos significativos son:

	7	6	5	4	3	2	1	0	
0x35	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	**MCUCR**

- **Bits 3 y 2 – ISC1[1:0]: Para configurar el sentido de INT1 (*ISC, Interrupt Sense Control*)**

  Definen el tipo de evento que genera la interrupción externa 1.

- **Bits 1 y 0 – ISC0[1:0]: Para configurar el sentido de INT0**

  Definen el tipo de evento que genera la interrupción externa 0.

En la tabla 4.2 se muestran los eventos que generan estas interrupciones, de acuerdo con el valor de los bits de configuración.

Tabla 4.2 Configuración del sentido de las interrupciones externas 0 y 1

ISCx1	ISCx0	Activación de la Interrupción
0	0	Por un nivel bajo de voltaje en INTx
0	1	Por cualquier cambio lógico en INTx
1	0	Por un flanco de bajada en INTx
1	1	Por un flanco de subida en INTx

x puede ser 0 ó 1

La configuración de INT2 se define con el bit **ISC2** ubicado en la posición 6 del registro **MCUCSR** (MCU Control and Status Register).

	7	6	5	4	3	2	1	0	
0x34	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	**MCUCSR**

En la tabla 4.3 se muestran las transiciones que generan la interrupción externa 2, en función del bit **ISC2**.

Tabla 4.3 Configuración del sentido de la interrupción externa 2

ISC2	Activación de la Interrupción
0	Por un flanco de bajada en INT2
1	Por un flanco de subida en INT2

Dado que la interrupción 2 es asíncrona, se requiere que el pulso generador del evento tenga una duración mínima de 50 nS para que pueda ser detectado.

4.1.2 Habilitación y Estado de las Interrupciones Externas

Cualquier interrupción va a producirse sólo si se activó al habilitador global de interrupciones y al habilitador individual de la interrupción de interés. El habilitador global es el bit **I**, ubicado en la posición 7 del registro de Estado (**SREG**, sección 2.4.1.1).

Los habilitadores individuales de las interrupciones externas se encuentran en el registro general para el control de interrupciones (**GICR**, *General Interrupt Control Register*), correspondiendo con los 3 bits más significativos de **GICR**:

	7	6	5	4	3	2	1	0	
0x3B	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	**GICR**

- **Bit 7 – INT1: Habilitador individual de la interrupción externa 1**
- **Bit 6 – INT0: Habilitador individual de la interrupción externa 0**
- **Bit 5 – INT2: Habilitador individual de la interrupción externa 2**

  No está disponible en un ATMega8.
- **Bits 4 al 2 – No están implementados**
- **Bits 1 y 0 – No están relacionados con las interrupciones externas**

El estado de las interrupciones externas se refleja en el registro general de banderas de interrupción (**GIFR**, *General Interrupt Flag Register*), el cual incluye una bandera por interrupción, estas banderas corresponden con los 3 bits más significativos de **GIFR**:

	7	6	5	4	3	2	1	0	
0x3A	INTF1	INTF0	INTF2	-	-	-	-	-	**GIFR**

- **Bit 7 – INTF1: Bandera de la interrupción externa 1**
- **Bit 6 – INTF0: Bandera de la interrupción externa 0**
- **Bit 5 – INTF2: Bandera de la interrupción externa 2**

  No está disponible en un ATMega8.
- **Bits 4 al 0 – No están implementados**

Las banderas se ponen en alto si el habilitador global y los habilitadores individuales están activados y ocurre el evento definido por los bits de configuración. La puesta en alto de una de estas banderas es lo que produce la interrupción, dando lugar a los

procedimientos descritos en la sección 2.6.1, la bandera se limpia automáticamente por hardware cuando se concluye con la atención a la interrupción. No es necesario evaluar las banderas por software, puesto que se tiene un vector diferente para cada evento.

4.1.3 Ejemplos de Uso de Interrupciones Externas

En esta sección se muestran dos ejemplos de uso de las interrupciones externas, documentando aspectos en la programación que deben ser considerados al momento de desarrollar otras aplicaciones.

Ejemplo 4.1: Realice un programa que conmute la salida menos significativa del puerto B (PB0) de un ATMega8 cada vez que es presionado un botón conectado en INT0, como se muestra en la figura 4.1.

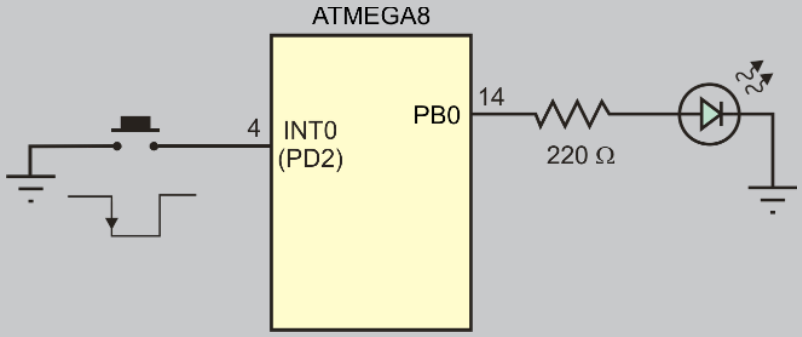


Figura 4.1 Manejo de un botón por interrupción

En el puerto D se habilita al resistor de *pull-up* para contar con un 1 lógico mientras no se presione el botón. El otro extremo del botón se conecta a tierra para insertar un 0 lógico al presionarlo, por lo tanto, la interrupción se configura por flanco de bajada.

La conmutación de la salida se realiza en la ISR, por lo que el programa principal queda ocioso.

Para la solución en ensamblador debe inicializarse al apuntador de pila, por la rutina de atención a la interrupción. La solución en ensamblador es:

```
        .include  <m8def.inc>      ; Biblioteca con definiciones

        .ORG    0x000              ; Vector de reset
        RJMP    Inicio


; La ISR se ubica en el vector de la interrupción, por ser la única a evaluar
        .ORG    0x001              ; Vector de la interrupción externa 0
        IN      R16, PORTB         ; Lee el último valor escrito
        EOR     R16, R17           ; Conmuta al LSB
        OUT     PORTB, R16         ; Escribe el resultado
```

```
        RETI

Inicio:                         ; Inicializaciones
        LDI     R16, 0x00
        OUT     DDRD, R16       ; Puerto D como entrada
        LDI     R16, 0xFF
        OUT     PORTD, R16      ; Resistor de Pull-Up en el puerto D

        OUT     DDRB, R16       ; Puerto B como salida

        LDI     R16, 0x04       ; Inicializa al apuntador de pila
        OUT     SPH, R16
        LDI     R16, 0x5F
        OUT     SPL, R16

        LDI     R16, 0B00000010 ; Configura INT0 por flanco de bajada
        OUT     MCUCR, R16
        LDI     R16, 0B01000000 ; Habilita la INT0
        OUT     GICR, R16

        CLR     R16             ; Estado inicial de la salida
        OUT     PORTB, R16

        LDI     R17, 0B00000001 ; Para conmutar con OR exclusiva

        SEI                     ; Habilitador global de interrupciones

Lazo:   RJMP    Lazo            ; Lazo infinito, permanece ocioso
```

Se observa que el trabajo del programa principal prácticamente es nulo, el recurso de la interrupción externa es el encargado de monitorear al botón y en su ISR se realiza la tarea deseada.

En lenguaje C es necesario incluir a la biblioteca **interrupt.h** para el manejo de las interrupciones, la solución en este lenguaje es:

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

ISR(INT0_vect) {                // ISR de la INT0

        PORTB = PORTB ^ 0x01;   // OR exclusiva para conmutar al LSB
}

int     main() {

        DDRD = 0x00;            // Puerto D como entrada
        PORTD = 0xFF;           // Resistor de Pull-Up en el puerto D

        DDRB = 0xFF;            // Puerto B como salida
```

```
        MCUCR = 0B00000010;          // Configura INT0 por flanco de bajada

        GICR = 0B01000000;           // Habilita la INT0

        PORTB = 0x00;                // Estado inicial de la salida

        sei();                       // Habilitador global de interrupciones

        while(1) {                   // Lazo infinito, permanece ocioso
            asm( "nop");
        }
}
```

La función `sei()` es para poner en alto al habilitador global de interrupciones.

La inclusión de la instrucción **`nop`** en el lazo infinito hace posible una simulación por pasos en el AVR Studio, si se omite, al no haber instrucciones dentro del **`while`**, no es posible simular la introducción de eventos en PD2 que produzcan la interrupción. Aunque el código máquina generado trabaja de manera correcta en el MCU.

El ejemplo siguiente muestra cómo una variable puede ser modificada por dos rutinas de atención a interrupciones diferentes.

Ejemplo 4.2: Realice un contador de eventos ascendente/descendente con salida en binario. Considere a las terminales INT0 e INT1 para el ingreso de eventos y muestre la salida en el puerto B, el hardware sugerido se muestra en la figura 4.2.

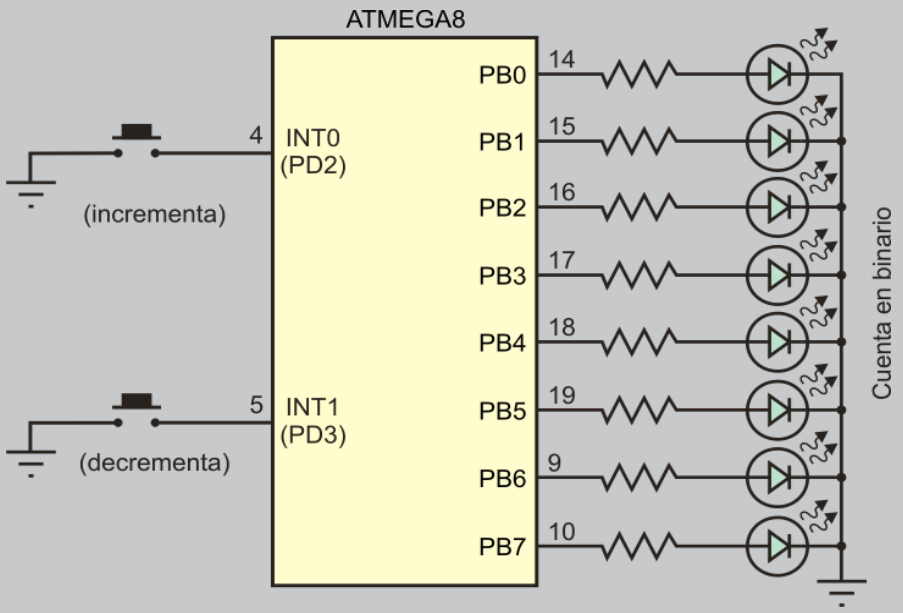


Figura 4.2 Manejo de dos interrupciones externas

Ambas interrupciones se configuran por flanco de bajada. En las ISRs se realiza la modificación del contador y la actualización de la salida, dejando ocioso al programa principal.

Para la solución en ensamblador, el valor del contador se lleva en el registro R17. La solución en ensamblador es:

```
       .include  <m8def.inc>      ; Biblioteca con definiciones

       .ORG   0x000               ; Vector de reset
       RJMP   Inicio

; Vectores de interrupciones
       .ORG   0x001               ; Vector de la interrupción externa 0
       RJMP   ISR_INT0

       .ORG   0x002               ; Vector de la interrupción externa 1
       RJMP   ISR_INT1

Inicio:                           ; Inicializaciones
       LDI    R16, 0x00
       OUT    DDRD, R16           ; Puerto D como entrada
       LDI    R16, 0xFF
       OUT    PORTD, R16          ; Resistor de Pull-Up en el puerto D

       OUT    DDRB, R16           ; Puerto B como salida

       LDI    R16, 0x04           ; Inicializa al apuntador de pila
       OUT    SPH, R16
       LDI    R16, 0x5F
       OUT    SPL, R16

       LDI    R16, 0B00001010     ; Configura INT1/INT0 por flanco de bajada
       OUT    MCUCR, R16
       LDI    R16, 0B11000000     ; Habilita INT1/INT0
       OUT    GICR, R16

       CLR    R17                 ; El contador inicia en 0
       OUT    PORTB, R17          ; Valor inicial de la salida
       SEI                        ; Habilitador global de interrupciones

Lazo:  RJMP   Lazo                ; Lazo infinito, permanece ocioso

ISR_INT0:                         ; Rutina de atención a la INT0
       INC    R17                 ; Incrementa al contador
       OUT    PORTB, R17          ; Actualiza la salida
       RETI

ISR_INT1:                         ; Rutina de atención a la INT1
       DEC    R17                 ; Reduce al contador
       OUT    PORTB, R17          ; Actualiza la salida
       RETI
```

En lenguaje C, debe manejarse una variable global con el valor del contador, para que pueda ser modificada por ambas rutinas de atención a las interrupciones. La solución en lenguaje C es:

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

unsigned  char cuenta;              // Variable global con el valor del contador

ISR(INT0_vect) {                    // ISR de la INT0
      cuenta++;                     // Incrementa al contador
      PORTB = cuenta;               // Actualiza la salida
}

ISR(INT1_vect) {                    // ISR de la INT1
      cuenta--;                     // Reduce al contador
      PORTB = cuenta;               // Actualiza la salida
}

int   main() {

      DDRD = 0x00;                  // Puerto D como entrada
      PORTD = 0xFF;                 // Resistor de Pull-Up en el puerto D

      DDRB = 0xFF;                  // Puerto B como salida

      MCUCR = 0B00001010;           // Configura INT1/INT0 por flanco de bajada

      GICR = 0B11000000;            // Habilita INT1 e INT0

      cuenta = 0;                   // Valor inicial del contador
      PORTB = cuenta;               // Inicializa la salida

      sei();                        // Habilitador global de interrupciones

      while(1) {                    // Lazo infinito, permanece ocioso
          asm( "nop");
      }
}
```

En lenguaje C, el optimizador de código del compilador aplica algunas reglas que pueden generar un comportamiento erróneo, cuando se manejan variables globales que se modifican en las ISRs.

Si la solución al ejemplo anterior se organiza de la siguiente manera:

```
unsigned  char cuenta;

ISR(INT0_vect) {                    // ISR de la INT0
      cuenta++;                     // Incrementa al contador
}

ISR(INT1_vect) {                    // ISR de la INT1
      cuenta--;                     // Reduce al contador
}
```

```
int    main() {

       . . .

       while(1) {
           PORTB = cuenta;
       }
}
```

Se esperaría que las ISRs modificaran a la variable `cuenta` y que posteriormente ese valor fuera actualizado en el puerto B dentro del programa principal. Esto no ocurre, la variable se modifica pero el puerto B no se actualiza. Esto se debe a que al hacer una revisión del programa principal y no encontrar cambios en la variable `cuenta`, el optimizador de código realiza una asignación única para **PORTB**.

Para resolver este problema, a la variable `cuenta` se le debe agregar el modificador **volatile**, declarándola de la siguiente manera:

```
volatile unsigned  char cuenta;
```

Con ello, se indica que la variable es modificada en diferentes ISRs y por lo tanto, no debe ser considerada con una asignación única dentro del programa principal.

En ambos ejemplos no se agregaron diagramas de flujo porque las soluciones se basaron en el hardware de las interrupciones externas y en sus ISRs. En el programa principal únicamente se realizaron las siguientes tareas:

1. Ubicación del apuntador de pila (versión en ensamblador)
2. Configuración de Entradas y Salidas
3. Configuración de la interrupción o interrupciones
4. Habilitación de las interrupciones externas
5. Activación del habilitador global de interrupciones (bit **I** en **SREG**)
6. En el lazo infinito, permanece ocioso

El trabajo de las ISRs en estos ejemplos es muy simple, en sistemas con mayor complejidad es conveniente realizar un diagrama de flujo para cada ISR. A diferencia del programa principal, que entra en un lazo infinito y jamás termina, las ISRs si tienen un final bien definido.

4.2 Temporizadores

Una labor habitual de los controladores es la determinación y uso de intervalos de tiempo concretos. Esto se hace a través de un recurso denominado Temporizador (*Timer*), el cual básicamente es un registro de *n*-bits que se incrementa de manera automática en cada ciclo de reloj.

El recurso puede ser configurado para que el registro se incremente en respuesta a eventos externos, en esos casos suele ser referido como un Contador de eventos (*Counter*), no obstante, por simplicidad, en este libro siempre es tratado como Temporizador, independientemente de que la temporización esté dada por eventos internos o externos.

Tanto el ATMega8 como el ATMega16 incluyen 3 temporizadores, 2 son de 8 bits y 1 de 16 bits. En la tabla 4.4 se listan los Registros I/O para cada temporizador, como los Registros I/O son de 8 bits, el temporizador 1 utiliza 2 de ellos.

Tabla 4.4 Registros de los temporizadores en los AVR

Temporizador	Tamaño	Registros	Dirección
Timer 0	8 bits	**TCNT0**	0x32
Timer 1	16 bits	**TCNT1H** : **TCNT1L**	0x2D, 0x2C
Timer 2	8 bits	**TCNT2**	0x24

4.2.1 Eventos de los Temporizadores

En los microcontroladores AVR, los eventos que se pueden generar por medio de los temporizadores son: Desbordamientos, coincidencias por comparación y captura de entrada. La ocurrencia de alguno de estos eventos se ve reflejada en el registro de banderas de interrupción (**TIFR**, *Timer Interupt Flag Register*), cuyo contenido es descrito en la sección 4.2.5.

4.2.1.1 Desbordamientos

Este evento ocurre cuando alguno de los temporizadores (**TCNT*n***) alcanza su valor máximo (*MAXVAL*) y se reinicia con 0, es decir, ocurre con una transición de 1's a 0's, esta transición provoca que una bandera (**TOV*n***) sea puesta en alto. Una aplicación puede sondear en espera de un nivel alto en la bandera, o bien, se puede configurar al hardware para que el evento produzca una interrupción.

Esta señalización indica que ha transcurrido un intervalo de tiempo o un número específico de eventos, el conteo tiene flexibilidad porque es posible cargar al registro para que inicie desde un valor determinado. Si se utiliza una carga, ésta debe repetirse cada vez que el evento es producido, para generar intervalos regulares.

En la figura 4.3 se esquematiza al temporizador generando eventos por desbordamiento,

se muestra la posibilidad de una carga paralela y la generación de la bandera.

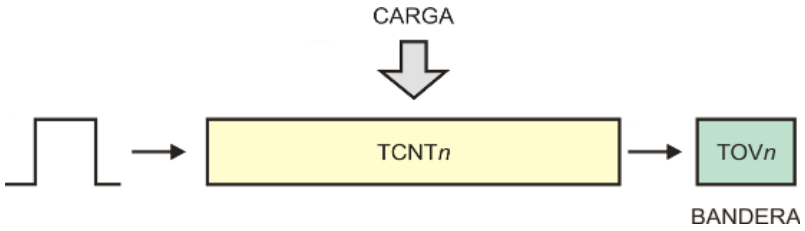


Figura 4.3 Desbordamientos en los temporizadores (*n = 0, 1, 2*)

Los desbordamientos pueden ser manejados por los 3 temporizadores (en la figura 4.3 *n* puede ser 0, 1 y 2). De hecho, los eventos por desbordamientos son un *de facto* en los temporizadores de los microcontroladores de diferentes fabricantes.

El valor máximo depende del número de bits del temporizador, queda determinado con la expresión:

$$MAXVAL = 2^{Tamaño(TCNTn)} - 1$$

4.2.1.2 Coincidencias por Comparación

En el hardware se dedica a un registro para comparaciones continuas (**OCRn**, *Output Compare Register*), en este registro se puede cargar un valor entre 0 y MAXVAL. En cada ciclo de reloj se compara al registro del temporizador con el registro de comparación, una coincidencia es un evento que se indica con la puesta en alto de una bandera (**OCF*n***, *Output Compare Flag*). La bandera puede sondearse por software o configurar al recurso para que genere una interrupción. En la figura 4.4 se esquematiza la generación de estos eventos.

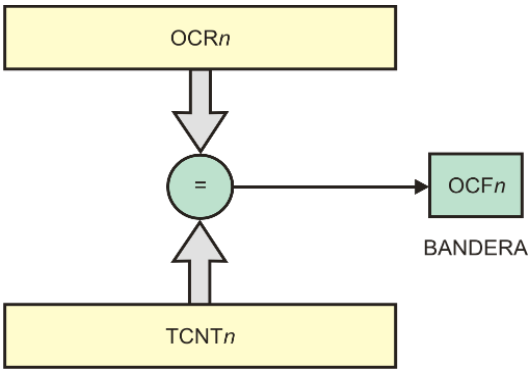


Figura 4.4 Coincidencias por comparación en los temporizadores

En un ATMega8 estos eventos pueden ser generados por los temporizadores 1 y 2, en un ATMega16 por los 3 temporizadores. En ambos microcontroladores, para el temporizador 1 se tienen 2 registros de comparación (**OCR1A** y **OCR1B**), por lo que con este temporizador se pueden manejar comparaciones con 2 valores diferentes.

Es posible configurar al temporizador para que se reinicie después de una coincidencia en la comparación y algunas terminales relacionadas pueden ajustarse, limpiarse o conmutarse automáticamente en cada coincidencia.

4.2.1.3 Captura de Entrada

Este tipo de eventos sólo es manejado por el temporizador 1, se tiene una terminal dedicada a capturar eventos externos (ICP, *Input Capture Pin*), un cambio en esta terminal provoca la lectura del temporizador y su almacenamiento en el registro de captura de entrada (**ICR**, *Input Capture Register*). El tipo de transición en ICP para generar las capturas es configurable, puede ser un flanco de subida o uno de bajada. En la figura 4.5 se muestra como se producen estos eventos.

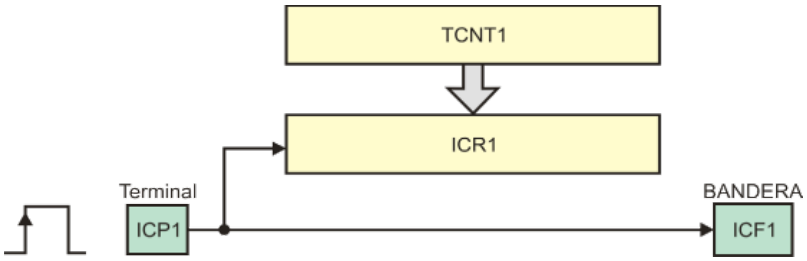


Figura 4.5 Captura de entrada

La bandera de captura de entrada (**ICF***, Input Capture Flag*) es puesta en alto indicando la ocurrencia del evento, esta bandera puede sondearse por software o se puede configurar al recurso para que genere una interrupción. Este evento puede ser útil para medir el ancho de un pulso externo.

4.2.2 Respuesta a los Eventos

Existen 3 formas para detectar los eventos producidos por los temporizadores y actuar ante ellos.

4.2.2.1 Sondeo (*Polling*)

El programa trabaja a un solo nivel (de acuerdo con la figura 2.13), en el cual destina un conjunto de instrucciones para evaluar de manera frecuente el estado de las banderas. Este método es el menos eficiente porque requiere de instrucciones adicionales en el programa principal e implica tiempo de procesamiento.

Como ejemplo, en la tabla 4.5 se muestra una secuencia de instrucciones que espera un evento por desbordamiento del temporizador 0. Al utilizar sondeo, las banderas se deben limpiar por software después de que ocurra el evento esperado, para ello se les debe escribir un 1 lógico.

Tabla 4.5 Detección por sondeo de un desbordamiento del temporizador 0

Versión en ensamblador

```
LOOP:  IN     R16, TIFR     ; Lee banderas
       SBRS   R16, TOV0     ; Brinca si la bandera TOV0 está en alto
       RJMP   LOOP
                            ; Regresa a esperar la bandera
       OUT    TIFR, R16     ; Limpia la bandera
       . . .                ; Continúa
```

Versión en Lenguaje C

```
       while  ( ! ( TIFR & 1 << TOV0 ) )  // Espera a la bandera
                    ;
       TIFR |= 1 << TOV0;                 // Limpia la bandera
       . . .
```

En una aplicación real, se ejecutan otras instrucciones mientras no ocurre el evento, para no dejar al programa dedicado únicamente a esperar y atender al evento.

4.2.2.2 Uso de Interrupciones

Todos los eventos de los temporizadores pueden generar interrupciones, para ello, éstas se deben activar en el registro de enmascaramiento de interrupciones (**TIMSK**, *Timer Interrupt Mask Register*), además de activar al habilitador global de interrupciones (bit **I** en **SREG**).

Con ello, el programa principal se ejecuta de manera normal, cuando ocurre un evento se concluye con la instrucción en proceso para dar paso a la ISR correspondiente y al concluir con su ejecución, el programa continúa con la instrucción siguiente a aquella que se estaba ejecutando cuando ocurrió el evento.

Esta forma de dar atención a los eventos es de las más ampliamente usadas, es eficiente porque en el programa principal no se invierte tiempo de procesamiento para esperar la ocurrencia de eventos.

También es conveniente para atender eventos de otros recursos. Cuando se emplean interrupciones, las banderas son limpiadas automáticamente por hardware.

4.2.2.3 Respuesta Automática

Los temporizadores 1 y 2 (y el 0 en el ATMega16) incluyen un módulo de generación de formas de onda utilizado para reaccionar únicamente por hardware ante eventos de comparación. Esto significa que con una coincidencia, automáticamente se modifican las terminales de comparación de salida (OC, *Output Compare*) para ponerse en alto, en bajo o conmutarse.

Con ello, en el software sólo se incluye la configuración del recurso y la atención a eventos se realiza de manera paralela a la ejecución del programa principal, sin requerir de instrucciones adicionales.

4.2.3 Pre-escalador

Un pre-escalador básicamente es un divisor de frecuencia que se antepone a los registros de los temporizadores proporcionándoles la capacidad de alcanzar intervalos de tiempo mayores. Un pre-escalador incluye 2 componentes, un contador de *n*-bits y un multiplexor para seleccionar diferentes posiciones de bit en el contador. El contador se incrementa en cada ciclo de reloj y por lo tanto, con el multiplexor pueden seleccionarse diferentes frecuencias.

Los microcontroladores AVR incluyen 2 pre-escaladores, uno compartido por los temporizadores 0 y 1, y el otro sólo utilizado por el temporizador 2. En ambos, el contador es de 10 bits y el multiplexor es de 8 a 1, pero difieren en su organización.

En la figura 4.6 se muestra la organización del pre-escalador compartido por los temporizadores 0 y 1, puede notarse que en realidad sólo comparten al contador, porque el hardware de selección es independiente.

Los bits de selección (**CS*xn***) están en los registros de configuración de los temporizadores. Después de un reinicio no hay señal de reloj porque tienen el valor de 0. Con estos bits puede seleccionarse la señal de reloj sin división, la señal de reloj con un factor de división entre 4 diferentes y 1 señal externa o su complemento (flancos de subida o bajada).

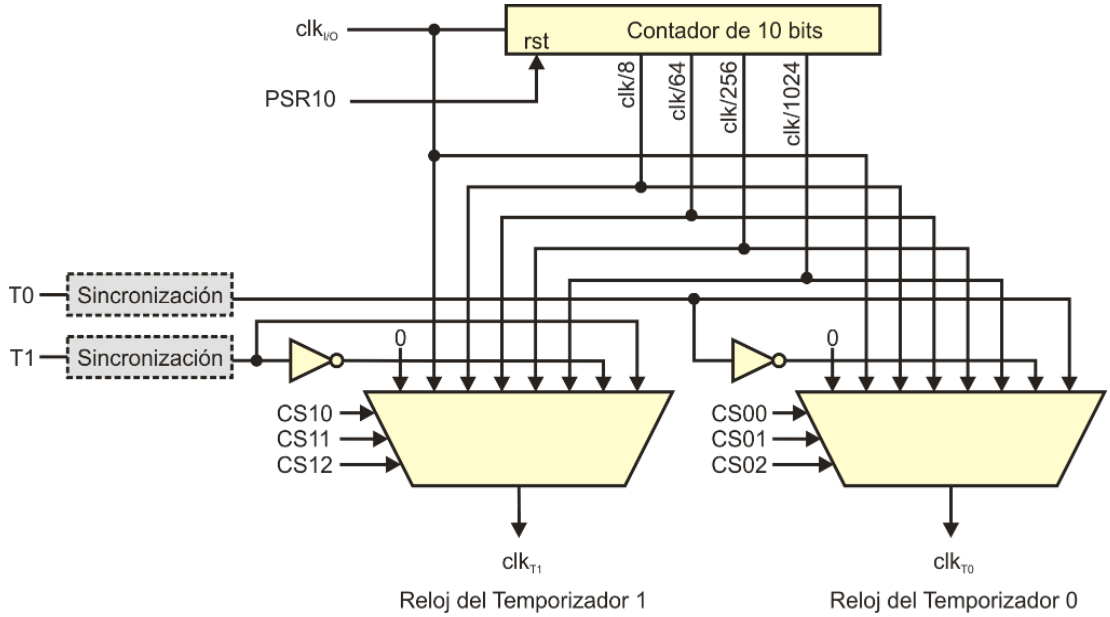


Figura 4.6 Pre-escalador compartido por los temporizadores 0 y 1

En la figura 4.7 se muestra la organización del pre-escalador utilizado por el temporizador 2, se observa que tiene 6 factores de división y que la señal externa es generada por un oscilador externo (TOSC1), permitiendo que el temporizador 2 trabaje a una frecuencia diferente al resto del sistema.

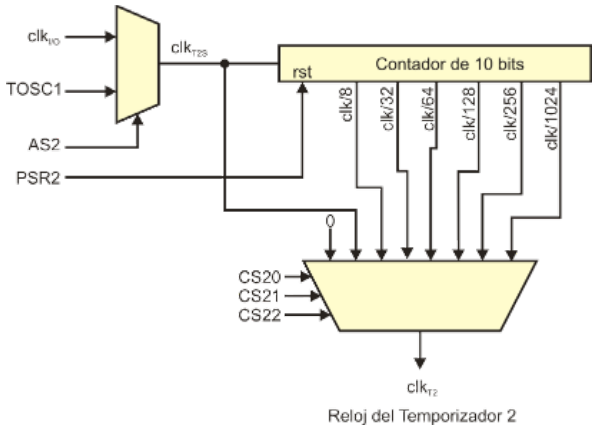


Figura 4.7 Pre-escalador del temporizador 2

Los contadores en los pre-escaladores tienen su señal de reinicio, la cual se activa con los bits **PSR10** y **PSR2**, estos bits corresponden con los 2 bits menos significativos del registro de función especial (**SFIOR**, *Special Function IO Register*), como se muestra a continuación:

	7	6	5	4	3	2	1	0	
0x30	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10	**SFIOR**

Los bits se ponen en alto por software para reiniciar a los contadores, y después de ello, automáticamente son limpiados por hardware. Por lo que si se realiza una lectura, siempre se obtiene el valor de 0.

4.2.4 Temporización Externa

Los temporizadores 0 y 1 pueden ser manejados por señales externas, T0 y T1, respectivamente. En estos casos se les conoce como contadores de eventos. Esto se mostró en la figura 4.6, sin embargo, en la figura 4.8 se muestran detalles de cómo se realiza la sincronización y la detección del flanco (subida o bajada), la etapa de sincronización asegura una señal estable a la etapa de detección de flanco, la cual se encarga de generar un pulso limpio cuando ocurre el flanco seleccionado. La presencia de estos módulos hace necesario que la frecuencia de los eventos externos esté limitada por $f_{clk\_I/O}/2.5$.

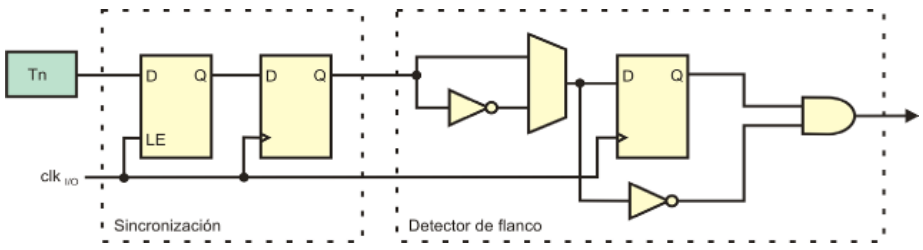


Figura 4.8 Acondicionamiento de las señales externas, para los temporizadores 0 y 1

El temporizador 2 también puede ser manejado por una señal externa, pero en este caso, se utiliza un oscilador externo que no se sincroniza con el oscilador interno (es asíncrono). El hardware se ha optimizado para ser manejado con un cristal de 32.768 KHz, esta frecuencia es adecuada para que, en combinación con el pre-escalador, se puedan generar fracciones o múltiplos de segundos. En la figura 4.9 se muestra cómo es posible seleccionar entre un oscilador externo o la señal de reloj interna.

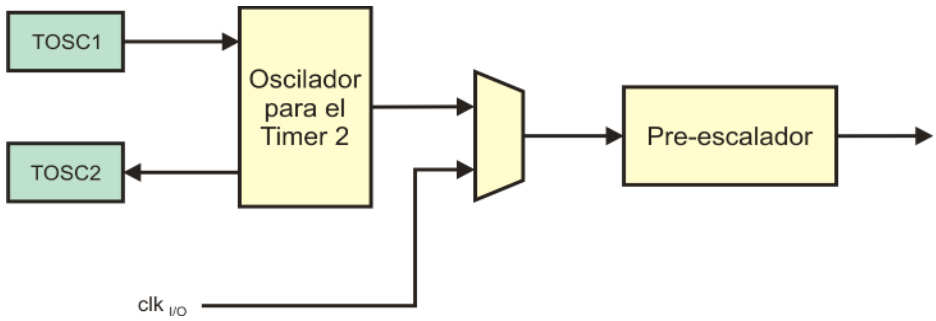


Figura 4.9 Uso de un oscilador externo para el temporizadores 2

La ventaja principal de un oscilador asíncrono para el temporizador 2 es que trabaja con una frecuencia independiente al resto del sistema, la cual es adecuada para manejar un reloj de tiempo real. Sin mucho esfuerzo, se puede realizar un reloj con displays de 7 segmentos y botones de configuración, de manera qué, mientras el temporizador 2 lleva el conteo de segundos, el resto del sistema está manejando a los elementos de visualización o sondeando los botones a una frecuencia mucho más alta.

4.2.5 Registros Compartidos por los Temporizadores

Como se describió en la sección 4.2.1, cuando ocurre un evento queda señalizado en el registro **TIFR**, el cual incluye los siguientes bits:

	7	6	5	4	3	2	1	0	
0x38	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

- **Bit 7 – OCF2: Bandera de coincidencia por comparación en el temporizador 2**
- **Bit 6 – TOV2: Bandera de desbordamiento en el temporizador 2**
- **Bit 5 – ICF1: Bandera de captura de entrada en el temporizador 1**
- **Bit 4 – OCF1A: Bandera de coincidencia con el comparador A del temporizador 1**
- **Bit 3 – OCF1B: Bandera de coincidencia con el comparador B del temporizador 1**

El temporizador 1 puede ser comparado con 2 valores diferentes.

- **Bit 2 – TOV1: Bandera de desbordamiento en el temporizador 1**
- **Bit 1 – OCF0: Bandera de coincidencia por comparación en el temporizador 0**

  Este bit no está implementado en el ATMega8 porque su temporizador 0 no maneja los eventos de coincidencia por comparación.

- **Bits 0 – TOV0: Bandera de desbordamiento en el temporizador 0**

Otro registro compartido por los 3 temporizadores es el registro **TIMSK**, el cual, como se mencionó en la sección 4.2.2.2, es utilizado para habilitar las interrupciones por los diferentes eventos de los temporizadores. Los bits en el registro **TIMSK** tienen una organización similar a los bits del registro **TIFR**, éstos se describen a continuación:

	7	6	5	4	3	2	1	0	
0x39	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	**TIMSK**

- **Bit 7 – OCIE2: Habilita la interrupción de coincidencia por comparación en el temporizador 2**
- **Bit 6 – TOIE2: Habilita la interrupción por desbordamiento del temporizador 2**
- **Bit 5 – TICIE1: Habilita la interrupción por captura de entrada en el temporizador 1**
- **Bit 4 – OCIE1A: Habilita la interrupción por coincidencia en el comparador A del temporizador 1**
- **Bit 3 – OCIE1B: Habilita la interrupción por coincidencia en el comparador B del temporizador 1**

  El temporizador 1 puede generar 2 interrupciones por comparación.

- **Bit 2 – TOIE1: Habilita la interrupción por desbordamiento del temporizador 1**
- **Bit 1 – OCIE0: Habilita la interrupción de coincidencia por comparación en el temporizador 0**

  En el ATMega8 este bit tampoco está implementado.

- **Bit 0 – TOIE0: Habilita la interrupción por desbordamiento del temporizador 0**

En un ATMega8 los temporizadores pueden generar 7 interrupciones diferentes y para un ATMega16 son 8, en la tabla 4.6 se muestran los vectores de las interrupciones relacionadas.

Tabla 4.6 Vectores de Interrupciones relacionadas con los temporizadores

Dirección		Evento	Descripción	Temporizador
ATMega8	ATMega16			
0x003	0x006	TIMER2_COMP	Coincidencia por comparación	2
0x004	0x008	TIMER2_OVF	Desbordamiento	2
0x005	0x00A	TIMER1_CAPT	Captura	1
0x006	0x00C	TIMER1_COMPA	Coincidencia en el comparador A	1
0x007	0x00E	TIMER1_COMPB	Coincidencia en el comparador B	1
0x008	0x010	TIMER1_OVF	Desbordamiento	1
0x009	0x012	TIMER0_OVF	Desbordamiento	0
	0x026	TIMER0_COMP	Coincidencia por comparación	0

4.2.6 Organización y Registros del Temporizador 0

El temporizador 0 es de 8 bits, su organización se muestra en la figura 4.10, en donde se observa que **TCNT0** (**0x32**) es el registro a incrementar y **OCR0** (**0x3C**, no implementado en el ATMega8) es el registro con el que se realiza la comparación en cada ciclo de reloj

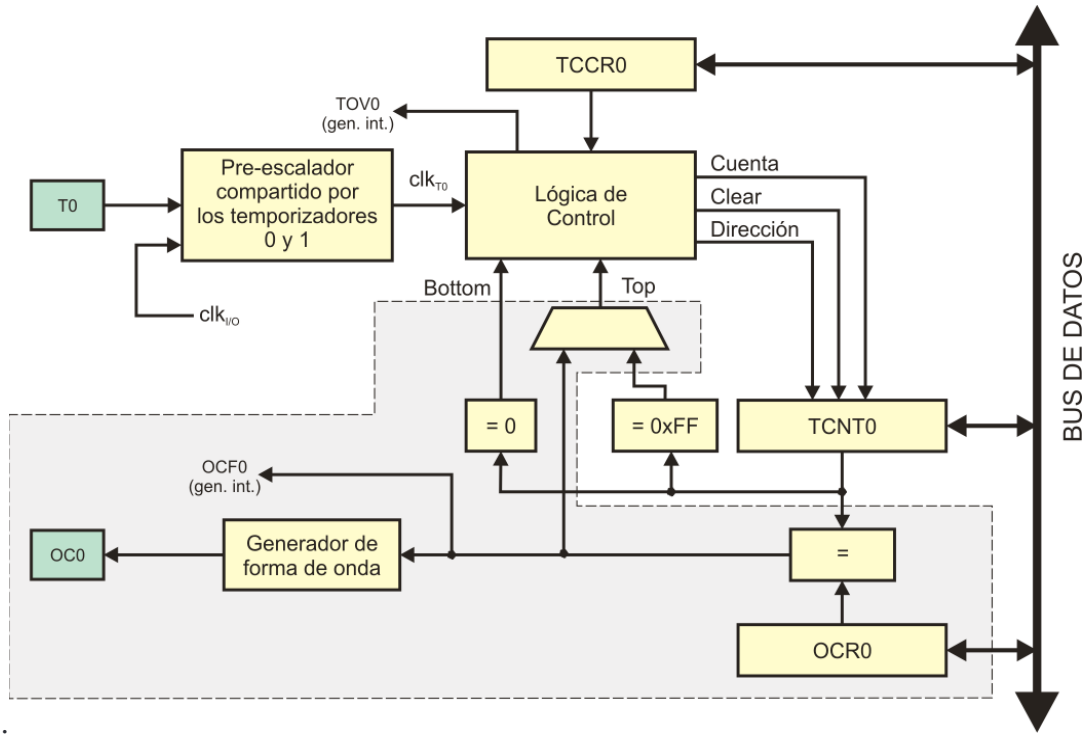


Figura 4.10 Organización del temporizador 0, la línea interrumpida encierra los recursos no disponibles en el ATMega8

En el ATMega8 sólo se pueden generar eventos de desbordamientos. En el ATMega16 también se pueden generar eventos de coincidencias por comparación y dar respuesta automática en la terminal OC0, en la figura 4.10 se han encerrado los recursos que no están en el ATMega8.

El temporizador 0 es controlado por el registro **TCCR0** (*Timer/Counter Control Register 0*), cuyos bits son descritos a continuación, cabe señalar que los bits del 3 al 7 se relacionan con los recursos de comparación y no están implementados en el ATMega8.

	7	6	5	4	3	2	1	0	
0x33	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	**TCCR0**

- **Bit 7 – FOC0: Obliga un evento de coincidencia por comparación (*Force Output Compare*)**

  La puesta en alto de este bit obliga a que ocurra un evento de coincidencia por comparación, incluyendo una respuesta automática en OC0, si es que fue configurada.

- **Bits 6 y 3 – WGM0[0:1]: Determinan el modo de operación del generador de formas de onda (*Waveform Generation Mode*)**

  Con estos 2 bits se puede seleccionar 1 de 4 modos de operación, éstos se describen en la sección 4.2.6.1

- **Bits 5 y 4 – COM0[1:0]: Configuran la respuesta automática en la terminal OC0 (*Compare Output Mode*)**

  Esta configuración se describe en la sección 4.2.6.2.

- **Bits 2, 1 y 0 – CS0[2:0]: Seleccionan la fuente de temporización (*Clock Select*)**

  Estos bits determinan la selección en el pre-escalador (sección 4.2.3) y se describen en la sección 4.2.6.3.

Con respecto a la figura 4.10, el bloque marcado como Lógica de Control determina el comportamiento del temporizador 0 a partir del valor de los bits en el registro **TCCR0**, después de un reinicio, el registro contiene ceros.

4.2.6.1 Generación de Formas de Onda con el Temporizador 0

En la tabla 4.7 se muestran los 4 posibles modos de generación de forma de onda, determinados por los bits **WGM0[1:0]**.

Tabla 4.7 Modos de generación de forma de onda

Modo	WGM01	WGM00	Descripción
0	0	0	Normal
1	0	1	PWM con fase correcta
2	1	0	CTC: Limpia al temporizador ante una coincidencia por comparación
3	1	1	PWM rápido

- **Modo 0:** Operación normal del temporizador, sólo se generan eventos de desbordamientos.
- **Modos 1 y 3:** Modos para la generación de PWM, en la sección 4.3 se describen todos los aspectos relacionados con PWM.
- **Modo 2:** Limpia al registro del temporizador (coloca 0's en **TCNT0**) tras una coincidencia en la comparación, se genera la bandera **OCF0** debido a que hubo una coincidencia.

4.2.6.2 Respuesta Automática en la Terminal OC0

En el modo normal o en el modo CTC, los bits **COM0[1:0]** definen el comportamiento descrito en la tabla 4.8 para la salida OC0, proporcionando una respuesta automática ante un evento de comparación. La respuesta en los modos de PWM se describe en la sección 4.3.

Tabla 4.8 Respuesta automática en OC0

COM01	COM00	Descripción
0	0	Operación Normal, terminal OC0 desconectada
0	1	Conmuta a OC0, tras una coincidencia por comparación
1	0	Limpia a OC0, tras una coincidencia por comparación
1	1	Pone en alto a OC0, tras una coincidencia por comparación

4.2.6.3 Selección del Reloj para el Temporizador 0

La selección de la señal de reloj está determinada por los bits **CS0[2:0]**, los cuales son descritos en la tabla 4.9. Estos bits se conectan directamente a los bits de selección del multiplexor del pre-escalador (sección 4.2.3). Después de un reinicio, el temporizador 0 está detenido porque no hay fuente de temporización.

Tabla 4.9 Bits para la selección del reloj en el temporizador 0

CS02	CS01	CS00	Descripción
0	0	0	Sin fuente de reloj (temporizador 0 detenido)
0	0	1	$clk_{I/O}$ (sin división)
0	1	0	$clk_{I/O}/8$ (del pre-escalador)
0	1	1	$clk_{I/O}/64$ (del pre-escalador)
1	0	0	$clk_{I/O}/256$ (del pre-escalador)
1	0	1	$clk_{I/O}/1024$ (del pre-escalador)
1	1	0	Fuente externa en T0, por flanco de bajada
1	1	1	Fuente externa en T0, por flanco de subida

4.2.7 Organización y Registros del Temporizador 1

La organización del temporizador 1 se muestra en la figura 4.11, este temporizador es el más completo porque es de 16 bits y puede manejar 3 tipos de eventos: por desbordamientos, por comparaciones con 2 valores diferentes y por captura de una entrada externa.

En la figura 4.11 se pueden ver 4 registros de 16 bits, cada uno requiere de 2 localidades en el espacio de Registros I/O: **TCNT1** (**0x2D:0x2C**) es el registro a incrementar, **OCR1A** (**0x2B:0x2A**) es uno de los registros con el que se realizan comparaciones, **OCR1B** (**0x29:0x28**) es el otro registro para las comparaciones e **ICR1** (**0x27:0x26**) es el registro para realizar capturas.

En lenguaje C se puede tener acceso directo a los 16 bits de cada uno de estos registros, en ensamblador se debe seguir un orden para las lecturas y escrituras, el cual se describe y justifica en la sección 4.2.7.4.

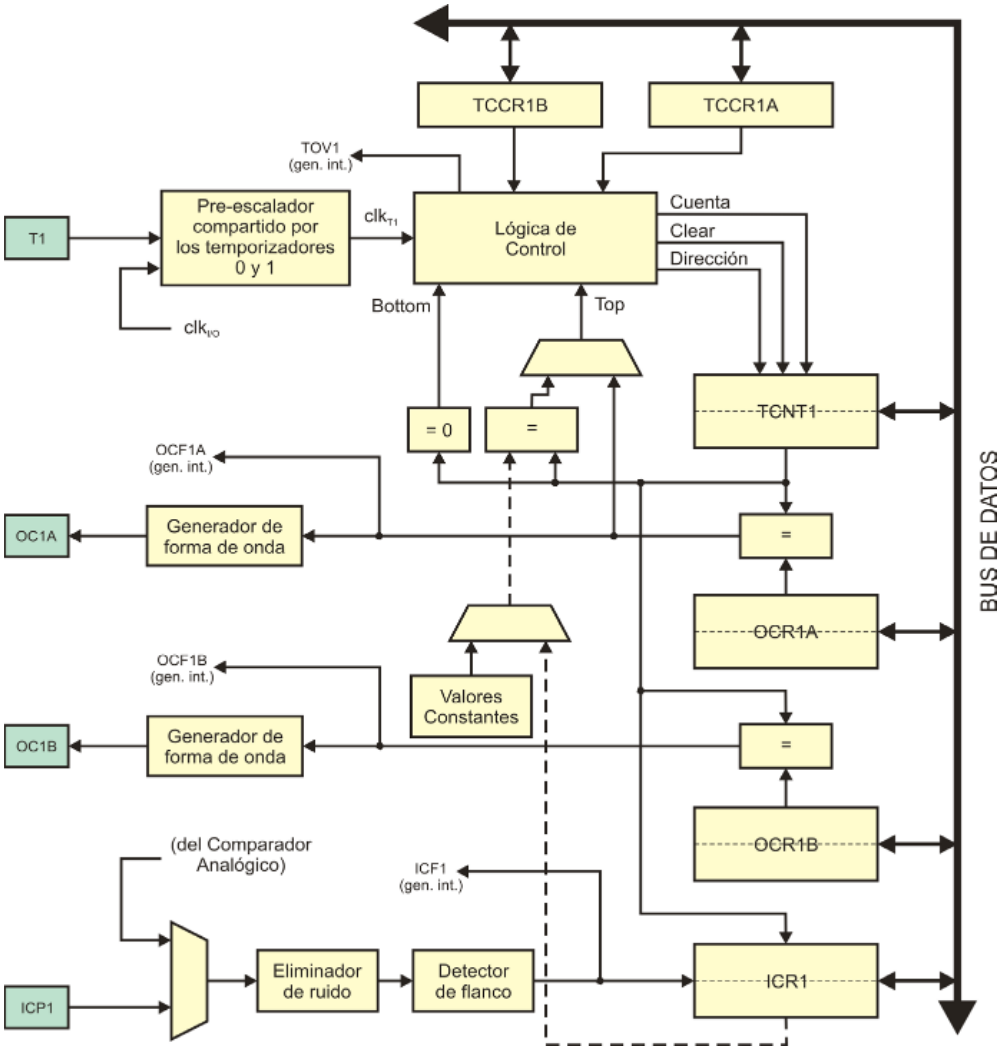
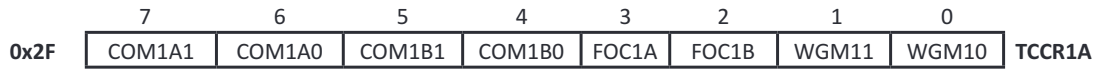


Figura 4.11 Organización del temporizador 1, es la misma para el ATMega8 y el ATMega16

El temporizador 1 es controlado por los registros **TCCR1A** y **TCCR1B** (*Timer/Counter Control Register 1A y 1B*), cuyos bits son descritos a continuación:



	7	6	5	4	3	2	1	0	
0x2F	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A

- **Bits 7 y 6 – COM1A[1:0]: Configuran la respuesta automática en la terminal OC1A**

  Para dar respuesta al comparador A, esta configuración se describe en la sección 4.2.7.2.

- **Bits 5 y 4 – COM1B[1:0]: Configuran la respuesta automática en la terminal OC1B**

  Para dar respuesta al comparador B, esta configuración se describe en la sección 4.2.7.2.

- **Bit 3 – FOC1A: Obliga un evento de coincidencia para el comparador A**

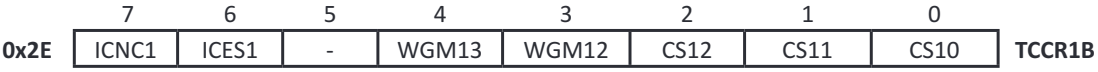
  La puesta en alto de este bit obliga a que ocurra un evento de coincidencia con el comparador A, incluyendo una respuesta automática en OC1A, si es que fue configurada.

- **Bit 2 – FOC1B: Obliga un evento de coincidencia para el comparador B**

  La puesta en alto de este bit obliga a que ocurra un evento de coincidencia con el comparador B, incluyendo una respuesta automática en OC1B, si es que fue configurada.

- **Bits 1 y 0 – WGM1[1:0]: Determinan el modo de operación del generador de formas de onda**

  Estos bits, en conjunción con otros 2 bits de **TCCR1B**, permiten seleccionar uno de sus modos de operación, éstos se describen en la sección 4.2.7.1.



	7	6	5	4	3	2	1	0	
0x2E	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

- **Bit 7 – ICNC1: Activa un eliminador de ruido para captura de entrada (*Input Capture Noise Canceler*)**

  Si el bit está activo, la entrada se filtra, de manera que, un cambio es válido si después de la transición, la señal se mantiene estable por cuatro muestras consecutivas. Esto retrasa un evento de captura por cuatro ciclos de reloj.

- **Bit 6 – ICES1: Selecciona el flanco de activación de la captura (*Input Capture Edge Select*)**

  Un evento externo en la terminal ICP1 hace que el temporizador 1 se copie en ICR1 y que la bandera ICF1 sea puesta en alto. Con este bit en bajo el evento es una transición de bajada y con el bit en alto es una transición de subida.

- **Bit 5 – No está implementado**

- **Bits 4 y 3 – WGM1[3:2]: Determinan el modo de operación del generador de formas de onda**

  Estos bits, en conjunción con otros 2 bits de **TCCR1A**, permiten seleccionar uno de sus modos de operación, éstos se describen en la sección 4.2.7.1

- **Bits 2, 1 y 0 – CS1[2:0]: Seleccionan la fuente de temporización (*Clock Select*)**

  Estos bits determinan la selección en el pre-escalador (sección 4.2.3) y se describen en la sección 4.2.7.3.

4.2.7.1 Generación de Formas de Onda con el Temporizador 1

El temporizador 1 incluye a los bits **WGM1[3:0]** para seleccionar entre 16 posibles modos de generación de forma de onda, no obstante, sólo 4 no están relacionados con PWM. En la tabla 4.10 se listan estos 4 modos y los 12 restantes se describen en la sección 4.3.

Tabla 4.10 Modos de generación de forma de onda que no están relacionados con PWM

Modo	WGM13	WGM12	WGM11	WGM10	Descripción
0	0	0	0	0	Normal
4	0	1	0	0	CTC: Limpia al temporizador ante una coincidencia con OCR1A
12	1	1	0	0	CTC: Limpia al temporizador ante una coincidencia con ICR1
13	1	1	0	1	Reservado

- **Modo 0:** Operación normal del temporizador, sólo se generan eventos de desbordamientos.

- **Modos 4 y 12:** Limpia al registro del temporizador (coloca 0's en **TCNT1**) tras una coincidencia en la comparación. En el modo 4 la comparación es con el registro **OCR1A** y en el modo 12 es con **ICR1**. En el modo 4, al ocurrir la coincidencia con **OCR1A** se genera la bandera **OCF1A**.

- **Modo 13:** Sin uso, en estas versiones de dispositivos.

En el modo 12, el registro **ICR1** contiene el valor máximo para el temporizador. Al asignarle una función alterna a este registro, la terminal ICP1 sólo funciona como entrada o salida general, de manera que no se pueden realizar tareas de captura.

4.1.2.1 Respuesta Automática en las Terminales OC1A y OC1B

En el modo normal y en los modos CTC, los bits **COM1A[1:0]** y **COM1B[1:0]** definen el comportamiento descrito en la tabla 4.11 para las salidas OC1A y OC1B, respectivamente, proporcionando una respuesta automática ante eventos de comparación.

Tabla 4.11 Respuesta automática en OC1A/OC1B

COM1A1 / COM1B1	COM1A0 / COM1B0	Descripción
0	0	Normal, terminales OC1A/OC1B desconectadas
0	1	Conmuta a OC1A/OC1B, tras una coincidencia por comparación
1	0	Limpia a OC1A/OC1B, tras una coincidencia por comparación
1	1	Pone en alto a OC1A/OC1B, tras una coincidencia por comparación

El comportamiento de las terminales OC1A/OC1B en los modos PWM se describe en la sección 4.3.

4.2.7.3 Selección del Reloj para el Temporizador 1

La selección de la señal de reloj está determinada por los bits **CS1[2:0]**, los cuales son descritos en la tabla 4.12. Estos bits se conectan directamente a los bits de selección del multiplexor del pre-escalador (sección 4.2.3). Después de un reinicio, el temporizador 1 está detenido porque no hay fuente de temporización.

Tabla 4.12 Bits para la selección del reloj en el temporizador 1

CS12	CS11	CS10	Descripción
0	0	0	Sin fuente de reloj (temporizador 1 detenido)
0	0	1	$clk_{I/O}$ (sin división)
0	1	0	$clk_{I/O}/8$ (del pre-escalador)
0	1	1	$clk_{I/O}/64$ (del pre-escalador)
1	0	0	$clk_{I/O}/256$ (del pre-escalador)
1	0	1	$clk_{I/O}/1024$ (del pre-escalador)
1	1	0	Fuente externa en T1, por flanco de bajada
1	1	1	Fuente externa en T1, por flanco de subida

4.2.7.4 Acceso a los Registros de 16 Bits del Temporizador 1

El temporizador 1 es de 16 bits y los Registros I/O son de 8 bits, por ello, el hardware incluye algunos mecanismos necesarios para su acceso, sin los cuales se presentaría un problema grave, porque el temporizador está cambiando en cada ciclo de reloj.

Para ilustrar este problema se asume que el hardware no incluye los citados mecanismos y que se requiere leer al temporizador 1, para dejar su contenido en los registros R17:R16. Si la lectura se realiza cuando el temporizador tiene

0x03FF, al mover la parte alta R17 queda con 0x03, en el siguiente ciclo de reloj el temporizador cambia a 0x0400, por lo que al leer la parte baja R16 queda con 0x00. Por lo tanto, el valor leído es 0x0300, que está muy lejos del valor real del temporizador.

Si la lectura se realiza en orden contrario, R16 queda con 0xFF y R17 con 0x04, el valor leído es de 0x04FF, que también muestra un error bastante significativo.

Para evitar estos conflictos y poder realizar escrituras y lecturas "al vuelo", es decir, sin detener al temporizador, se incorpora un registro temporal de 8 bits que no es visible al programador, el cual está conectado directamente con la parte alta del registro de 16 bits.

Cuando se escribe o lee la parte alta del temporizador, en realidad se hace la escritura o lectura en el registro temporal. Cuando se tiene acceso a la parte baja, se hacen lecturas o escrituras de 16 bits, tomando como parte alta al registro temporal. Esto significa que se debe tener un orden de acceso, una lectura debe iniciar con la parte baja y una escritura debe iniciar con la parte alta. El orden de las lecturas se ilustra con las instrucciones:

```
IN   R16, TCNT1L    ; lee 16 bits, el byte alto queda en el registro temporal
IN   R17, TCNT1H    ; lee del registro temporal
```

El orden para las escrituras se ilustra en la siguiente secuencia, con la que se escribe 1500 en el registro del temporizador 1:

```
LDI  R16, LOW(1500)    ; Escribe el byte bajo de la constante
LDI  R17, HIGH(1500)   ; Escribe el byte alto de la constante
OUT  TCNT1H, R17       ; Escribe en el registro temporal
OUT  TCNT1L, R16       ; Escritura de 16 bits
```

No es posible tener acceso sólo a un byte del temporizador.

En alto nivel no es necesario revisar el orden de acceso, puesto que se pueden emplear datos de 16 bits, pudiendo ser variables o registros.

4.1.3 Organización y Registros del Temporizador 2

El temporizador 2 es de 8 bits, difiere del temporizador 0 de un ATMega16 por su capacidad de sincronizarse con un oscilador externo, cuya salida posteriormente puede ser procesada por el pre-escalador. En la figura 4.12 se muestra su organización, donde se observa que el registro **TCNT2** (**0x24**) es el que se incrementa y el registro **OCR2** (**0x23**) es empleado para comparaciones.

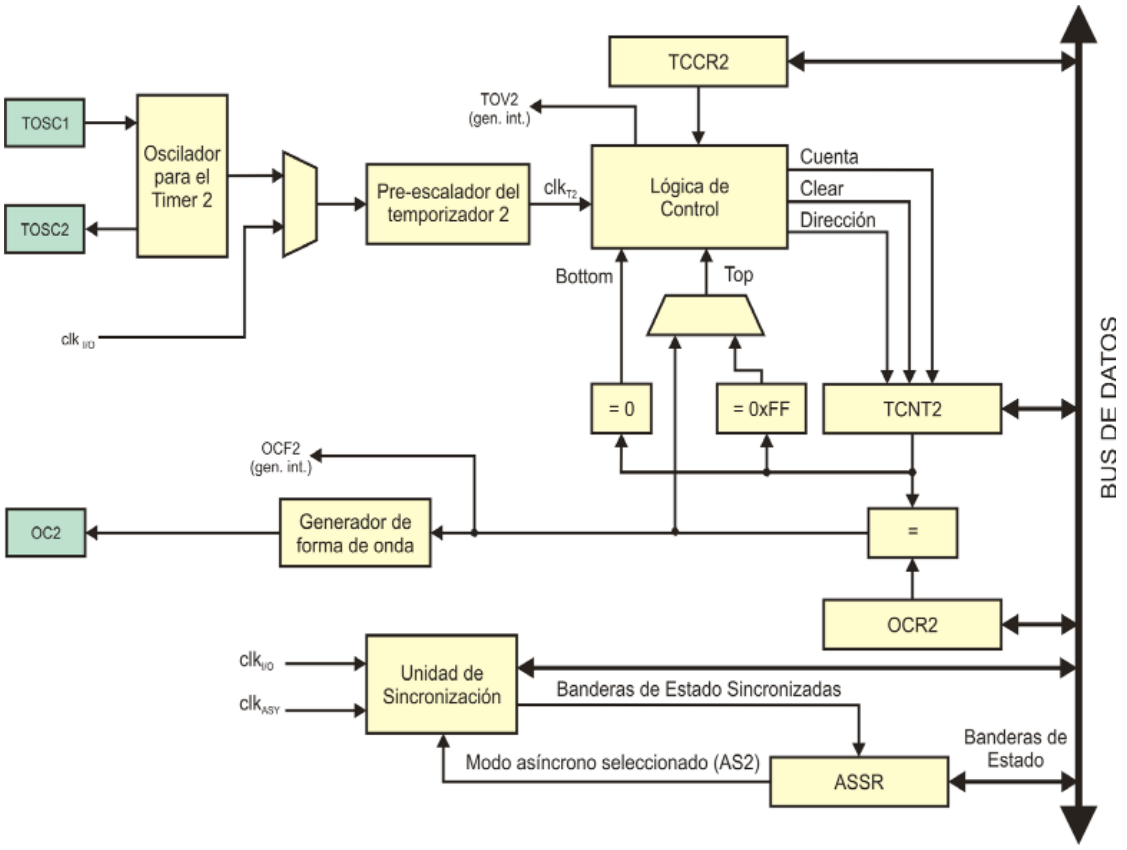


Figura 4.12 Organización del temporizador 2, es la misma para el ATMega8 y el ATMega16

El temporizador 2 es controlado por el registro **TCCR2** (*Timer/Counter Control Register 2*), cuyos bits se describen a continuación:

	7	6	5	4	3	2	1	0	
0x25	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	**TCCR2**

- **Bit 7 – FOC2: Obliga un evento de coincidencia por comparación (*Force Output Compare*)**

  La puesta en alto de este bit obliga a que ocurra un evento de coincidencia por comparación, incluyendo una respuesta automática en OC2, si es que fue configurada.

- **Bits 6 y 3 – WGM2[0:1] Determinan el modo de operación del generador de formas de onda (*Waveform Generation Mode*)**

  Con estos 2 bits se puede seleccionar 1 de 4 modos de operación, éstos se describen en la sección 4.2.8.1

- **Bits 5 y 4 – COM2[1:0]: Configuran la respuesta automática en la terminal OC2 (*Compare Output Mode*)**

  Esta configuración se describe en la sección 4.2.8.2.

- **Bits 2, 1 y 0 – CS2[2:0] Seleccionan la fuente de temporización (*Clock Select*)**

  Estos bits determinan la selección en el pre-escalador (sección 4.2.3) y se describen en la sección 4.2.8.3.

En la figura 4.12 también se observa al registro de estado asíncrono (***ASSR***, *Asynchronous Status Register*), el cual sirve para habilitar la operación asíncrona del temporizador 2 y para conocer su estado, sus bits son:

	7	6	5	4	3	2	1	0	
0x22	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB	**ASSR**

- **Bits 7 al 4 – No están implementados en el registro**

- **Bit 3 – AS2: Habilita la operación asíncrona**

  La puesta en alto de este bit hace que el temporizador 2 sea manejado desde un oscilador externo conectado en TOSC1 y TOSC2. Si se mantiene en bajo, el temporizador es manejado con el reloj $CLK_{I/O}$.

- **Bit 2 – TCN2UB: Bandera para indicar que el registro TCNT2 está ocupado por actualización (*Update Busy*)**

- **Bit 1 – OCR2UB: Bandera para indicar que el registro OCR2 está ocupado por actualización**

- **Bit 0 – TCR2UB: Bandera para indicar que el registro TCCR2 está ocupado por actualización**

Los 3 bits menos significativos son banderas de estado para indicar si un registro está ocupado por actualización, son necesarias cuando se habilita al oscilador externo porque el temporizador 2 trabaja a una frecuencia diferente a la del resto del sistema, generalmente es más baja. Por ello, antes de hacer una lectura o escritura en uno de estos registros (**TCNT2**, **OCR2** y **TCCR2**) se debe verificar si no hay un cambio en proceso. Las banderas se ajustan o limpian de manera automática.

La activación del oscilador externo puede ocasionar que los valores de los registros **TCNT2**, **OCR2** y **TCCR2** se alteren con el cambio de la señal de reloj. Por ello, es recomendable que primero se ajuste al bit **AS2** y que luego se definan los valores correctos para los citados registros.

4.2.8.1 Generación de Formas de Onda con el Temporizador 2

En la tabla 4.13 se muestran los 4 posibles modos de generación de forma de onda, determinados por los bits **WGM2[1:0]**.

Tabla 4.13 Modos de generación de forma de onda

Modo	WGM21	WGM20	Descripción
0	0	0	Normal
1	0	1	PWM con fase correcta
2	1	0	CTC: Limpia al temporizador ante una coincidencia por comparación
3	1	1	PWM rápido

- **Modo 0:** Operación normal del temporizador, sólo se generan eventos de desbordamientos.
- **Modos 1 y 3:** Modos para la generación de PWM, se describen en la sección 4.3.
- **Modo 2:** Limpia al registro del temporizador (coloca 0's en **TCNT2**) tras una coincidencia en la comparación, se genera la bandera **OCF2** debido a que hubo una coincidencia.

4.2.8.2 Respuesta Automática en la Terminal OC2

En el modo normal o en el modo CTC, los bits **COM2[1:0]** definen el comportamiento descrito en la tabla 4.14 para la salida OC2, proporcionando una respuesta automática ante un evento de comparación. La respuesta en los modos de PWM se describe en la sección 4.3.

Tabla 4.14 Respuesta automática en OC2

COM21	COM20	Descripción
0	0	Operación Normal, terminal OC2 desconectada
0	1	Conmuta a OC2, tras una coincidencia por comparación
1	0	Limpia a OC2, tras una coincidencia por comparación
1	1	Pone en alto a OC2, tras una coincidencia por comparación

4.2.8.3 Selección del Reloj para el Temporizador 2

La selección de la señal de reloj está determinada por los bits **CS2[2:0]**, los cuales son descritos en la tabla 4.15. En este caso, las 8 combinaciones son aplicables para el oscilador interno o externo.

Tabla 4.15 Bits para la selección del reloj en el temporizador 2

CS22	CS21	CS20	Descripción
0	0	0	Sin fuente de reloj (temporizador 0 detenido)
0	0	1	clk_{T2S} (sin división)
0	1	0	$clk_{T2S}/8$ (del pre-escalador)
0	1	1	$clk_{T2S}/32$ (del pre-escalador)
1	0	0	$clk_{T2S}/64$ (del pre-escalador)
1	0	1	$clk_{T2S}/128$ (del pre-escalador)
1	1	0	$clk_{T2S}/256$ (del pre-escalador)
1	1	1	$clk_{T2S}/1024$ (del pre-escalador)

El hardware del oscilador externo está optimizado para trabajar a una frecuencia de 32.768 KHz, la cual es adecuada para aplicaciones que involucren un reloj de tiempo real. El periodo que le corresponde es de 30.517578125 µS. Como el temporizador es de 8 bits, si no utiliza al pre-escalador genera un desbordamiento cada 256 x 30.517578125 µS = 7.8125 mS = 1/128 S.

Al emplear al pre-escalador se generan desbordamientos en otras fracciones o múltiplos de segundos reales, en la tabla 4.16 se muestran las frecuencias de operación del temporizador y los periodos de desbordamiento con diferentes factores de pre-escala.

Tabla 4.16 Periodos de desbordamiento en el temporizador 2 con un oscilador externo de 32.768 KHz

Pre-escala	Frecuencia del temporizador 2 (Hz)	Periodo de desbordamiento (S)
1	32 768	1/128
8	4096	1/16
32	1024	1/4
64	512	1/2
128	256	1
256	128	2
1024	32	8

Un reloj de tiempo real con base en un oscilador externo de 32.768 KHz es preciso porque no requiere de instrucciones adicionales para recargar al registro del temporizador, instrucciones que introducirían retrasos de tiempo. Además, puesto que la base de tiempo no dependería del oscilador interno, no sería indispensable realizar su calibración, la cual se mencionó en la sección 2.8.4.

4.2.9 Ejemplos de Uso de los Temporizadores

En esta sección se muestran 3 ejemplos de uso de los temporizadores, estos recursos son muy versátiles, los ejemplos ilustran algunas características representativas de su funcionamiento.

Ejemplo 4.3 Con un ATMega16, suponiendo que está trabajando con el oscilador interno de 1 MHz, genere una señal con una frecuencia de 5 KHz y un ciclo de trabajo del 50 %, utilizando al temporizador 0. Desarrolle las soluciones utilizando:

a) Desbordamiento con sondeo

b) Desbordamiento con interrupciones

c) Coincidencia por comparación con interrupciones, y

d) Coincidencia por comparación con respuesta automática

Por la diversidad de soluciones, sólo se desarrollan en lenguaje C, las soluciones en lenguaje ensamblador deben contar con la misma estructura.

Si $f = 5\ KHz$ entonces $T = 200\ \mu S$.

Puesto que el ciclo útil es del 50 %, se tiene $T_{ALTO} = 100\ \mu S$ y $T_{BAJO} = 100\ \mu S$. Al emplear un oscilador de 1 MHz, sin pre-escalador, el temporizador se incrementa cada $1\ \mu S$, por lo tanto, el temporizador debe contar 100 eventos antes de conmutar la salida.

a) Desbordamiento con sondeo: El programa debe configurar al temporizador, esperar la bandera, conmutar la salida y recargar al temporizador, para nuevamente esperar la bandera. Puesto que no hay condiciones para la salida, ésta se genera en PB0, el código correspondiente es:

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

int    main() {

       DDRB = 0xFF;                // Puerto B como salida
       PORTB = 0x00;               // Inicializa la salida

       TCNT0 = -100;               // Para que desborde en el evento 100
       TCCR0 = 0x01;               // Reloj, sin pre-escalador

       while(1) {                  // Lazo infinito
              while( ! ( TIFR & 1 << TOV0 ) )   // Sondea, espera la bandera
                     ;
              TIFR = TIFR | 1 << TOV0;   // Limpia la bandera, le escribe 1
              TCNT0 = -100;              // Recarga al temporizador
              PORTB = PORTB ^ 0x01;      // Conmuta la salida
       }
}
```

Esta solución tiene las siguientes desventajas: Se invierte tiempo de procesamiento en el sondeo y en la limpieza de la bandera de desbordamiento. Como consecuencia, el periodo es incorrecto por las instrucciones agregadas.

Para corregir el periodo deberían contabilizarse los ciclos requeridos por estas actividades y considerarlos en el valor de recarga.

El temporizador se recarga con -100 porque cuenta en orden ascendente, la representación en hexadecimal de -100 es 0x9C, que corresponde con 156. Partiendo de este número, cuando ocurre la transición de 0xFF a 0x00 han transcurrido 100 eventos.

b) Desbordamiento con interrupciones: Ahora en el programa principal se configura al temporizador 0 para que genere una interrupción cuando haya un desbordamiento y en la rutina de atención a la interrupción se conmuta la salida y recarga al temporizador.

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

ISR(TIMER0_OVF_vect) {     // Atiende un evento de desbordamiento
       TCNT0 = -100;              // Recarga al temporizador
       PORTB = PORTB ^ 0x01;      // Conmuta la salida
}

int    main() {

       DDRB = 0xFF;         // Puerto B como salida
       PORTB = 0x00;        // Inicializa la salida

       TCNT0 = -100;        // Para que desborde en el evento 100
       TCCR0 = 0x01;        // Reloj, sin pre-escalador

       TIMSK = 0x01;  // Habilita interrupción por desbordamiento del
       sei();         // temporizador 0 y al habilitador global

       while(1) {           // Lazo infinito, permanece ocioso
           asm("nop");// Se agrega para hacer posible la simulación
       }
}
```

Aunque ya no se invierte tiempo en el sondeo y en la limpieza de la bandera de desbordamiento, el periodo aún es incorrecto por la recarga del temporizador. La bandera de desbordamiento se limpia automáticamente por hardware.

c) Coincidencia por comparación con interrupciones: Para esta solución se carga el registro de comparación (**OCR0**) con 99, la cuenta inicia con 0, de 0 a 99 son 100 eventos. Se configura al recurso para que cuando ocurra una coincidencia se reinicie al temporizador (modo CTC) y se generé una interrupción en cuya ISR se conmute la salida.

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

ISR(TIMER0_COMP_vect) { // Atiende evento de coincidencia por comparación
       PORTB = PORTB ^ 0x01;      // Conmuta la salida
}

int    main() {

       DDRB = 0xFF;         // Puerto B como salida
       PORTB = 0x00;        // Inicializa la salida

       OCR0  = 99;          // de 0 a 99 son 100 eventos
       TCCR0 = 0x09;        // Reloj sin pre-escalador y activa modo CTC

       TIMSK = 0x02;        // Habilita interrupción de coincidencia por compara-
       sei();               // ción en el temporizador 0 y al habilitador global

       while(1) {           // Lazo infinito, permanece ocioso
           asm("nop");      // Sólo para simulación
       }
}
```

Esta solución es eficiente, la señal se genera con la frecuencia solicitada porque no es necesario recargar al temporizador, éste automáticamente es reiniciado en 0 cuando su valor coincide con el del registro **OCR0**, sólo es necesario considerar la conmutación de la salida en la ISR.

d) Coincidencia por comparación con respuesta automática: En esta solución la salida es generada por hardware, por lo debe ser en la terminal OC0 (PB3), el código es:

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

int    main() {

       DDRB = 0xFF;         // Puerto B como salida
       PORTB = 0x00;        // Inicializa la salida

       OCR0  = 99;          // de 0 a 99 son 100 eventos
       TCCR0 = 0x19;        // Reloj sin pre-escalador, activa modo CTC
                            // y habilita conmutación automática
       while(1) {           // Lazo infinito, permanece ocioso
           asm("nop");      // Sólo para la simulación
       }
}
```

Ésta es la mejor solución porque hace uso de los recursos de hardware disponibles en el MCU.

Ejemplo 4.4 Empleando un ATMega16 genere una señal con una frecuencia de 200 Hz y un ciclo de trabajo del 50 %, suponga que está trabajando con el oscilador interno de 1 MHz, muestre la solución más simple en Ensamblador y en lenguaje C.

Si $f = 200\ Hz$ entonces $T = 5\ mS$ esto significa: $T_{ALTO} = 2500\ \mu S$ y $T_{BAJO} = 2500\ \mu S$.

Un conteo de 2500 eventos no es posible con un temporizador de 8 bits, a menos que se utilice al pre-escalador. Por simplicidad, se emplea al temporizador 1 (16 bits) sin pre-escalador y se configura para una limpieza ante una coincidencia por comparación (modo CTC) y con respuesta automática en OC1A (PD5).

El código en lenguaje C es el siguiente:

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

int    main() {

       DDRD = 0xFF;          // Puerto D como salida
       PORTD = 0x00;         // Inicializa la salida

       OCR1A  = 2499;        // para 2500 eventos

       TCCR1A = 0x40;        // Reloj sin pre-escalador, activa modo CTC
       TCCR1B = 0x09;        // y habilita conmutación automática de OC1A

       while(1) {            // Lazo infinito, permanece ocioso
           asm("nop");       // Sólo para la simulación
       }
}
```

El programa en ensamblador es:

```
       .include      "m16def.inc"

       LDI    R16, 0xFF           ; Puerto D como salida
       OUT    DDRD, R16
       CLR    R16                 ; Inicializa la salida
       OUT    PORTD, R16

       LDI    R16, LOW(2499)      ; Carga registro de comparación
       LDI    R17, HIGH(2499)     ; para 2500 eventos

       OUT    OCR1AH, R17
       OUT    OCR1AL, R16

       LDI    R16, 0x40           ; Reloj sin pre-escalador, activa modo CTC
       OUT    TCCR1A, R16         ; y habilita conmutación automática de OC1A
       LDI    R16, 0x09
       OUT    TCCR1B, R16
aqui:  RJMP   aqui                ; Lazo infinito, permanece ocioso
```

Con base en los últimos 2 ejercicios se observa que, la generación de una señal a determinada frecuencia, empleando respuesta automática, básicamente se reduce a configurar el recurso. Para modificar la frecuencia sólo hay que modificar el valor del registro de comparación. Si se requiriera generar las 2 señales en forma simultánea, basta con juntar el código de los 2 ejemplos anteriores en un programa, como las instrucciones hacen referencia a recursos diferentes, no interfiere la configuración del temporizador 0 con la configuración del temporizador 1.

En el siguiente ejemplo se combina el uso de una interrupción externa y 2 de los temporizadores. Un aspecto interesante es que los temporizadores no trabajan en todo momento, sino que el inicio de su operación lo determina la interrupción externa.

Ejemplo 4.5 Configure al ATMega8 para que funcione como un timbre, de manera que genere un tono de 400 Hz durante 3 segundos cuando se presione un botón. En la figura 4.13 se muestra el acondicionamiento del hardware con la salida esperada.

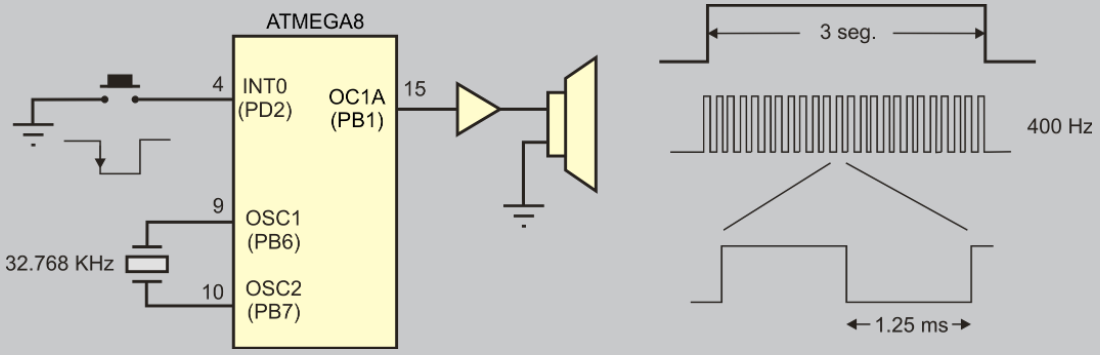


Figura 4.13 Hardware para el desarrollo del timbre

Cuando se manejan interrupciones de diferentes recursos, en ocasiones no es posible un planteamiento del software con un diagrama de flujo porque en el programa principal básicamente se realiza la configuración de los recursos y la funcionalidad del sistema queda determinada por las ISRs.

En esos casos, lo aconsejable es analizar la funcionalidad del sistema en su conjunto, determinando la tarea de cada recurso y definiendo cuándo debe ser activado o desactivado. En este ejemplo se utilizan 3 recursos:

1. La **interrupción externa 0** detecta si se ha sido presionado el botón, el recurso está activo desde que el programa inicia y se desactiva en su ISR, para evitar otras interrupciones externas mientras transcurre el intervalo de 3 segundos. El recurso es reactivado nuevamente en la ISR del temporizador 2, cuando terminan los 3 segundos.

2. El **temporizador 2** se emplea para el conteo de segundos, se configura para que trabaje con el oscilador externo y desborde cada segundo, generando una interrupción. Se activa en la ISR de la interrupción externa y se desactiva en su misma ISR, una vez que han concluido los 3 segundos.

3. El **temporizador 1** se emplea para generar el tono de 400 Hz, se configura para que trabaje con respuesta automática. Se activa en la ISR de la interrupción externa y se desactiva en la ISR del temporizador 2, una vez que ha finalizado el intervalo de 3 segundos.

El contador de segundos se inicializa en la ISR de la interrupción externa y se modifica en la ISR del temporizador 2, por lo tanto, debe manejarse con una variable global.

Con este análisis, se procede con la elaboración del programa, sólo se muestra la versión en lenguaje C.

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

volatile  unsigned char  segundos;

ISR(INT0_vect) {           // Atiende a la interrupción externa 0

       GICR = 0x00;               // Inhabilita la INT0
       // Complementa la configuración del temporizador 1
       TCNT1 = 0x0000;            // Garantiza al temporizador 1 en 0
       TCCR1B = 0x09;             // Activa al oscilador y al modo CTC
       // Complementa la configuración del temporizador 2
       segundos = 0;              // La cuenta de segundos inicia en 0
       TCNT2 = 0x00;              // Garantiza al temporizador 2 en 0
       TCCR2 = 0x05;              // Para que desborde cada segundo
       TIMSK = 0x40;              // Interrupción por desbordamiento
}

ISR(TIMER2_OVF_vect) {   // Interrupción por desbordamiento del timer 2

       segundos++;                // Ha pasado un segundo
       if( segundos == 3 ) {
              GICR = 0x40;        // Habilita la INT0
              TCCR1B = 0x00;// Detiene al temporizador 1 (quita el tono)
              TCCR2 = 0x00;       // Detiene al temporizador 2
              TIMSK = 0x00; //Desactiva su interrupción por desbordamiento
       }
}

int    main() {

       // Configuración de entradas y salidas
       DDRD = 0x00;               // Puerto D como entrada
       PORTD = 0xFF;              // Resistor de Pull-Up
       DDRB = 0xFF;               // Puerto B como salida

       // Configuración de la interrupción externa 0
       MCUCR = 0x02;              // Configura INT0 por flanco de bajada
       GICR = 0x40;               // Habilita la INT0

       // Configuración parcial del Temporizador 1
```

```
        OCR1A = 1249;               // Para 1.25 mS
        TCCR1A = 0x40;              // Configura para respuesta automática

        // Configuración parcial del temporizador 2
        ASSR = 0x08;                // Se usa al oscilador externo
        sei();                      // Habilitador global

        while(1) {                  // Lazo infinito, permanece ocioso
            asm("nop");             // Sólo para simulación
        }
}
```

4.3 Modulación por Ancho de Pulso (PWM)

La modulación por ancho de pulso es una técnica para generar "señales analógicas" en alguna salida de un sistema digital. Puede usarse para controlar la velocidad de un motor, la intensidad luminosa de una lámpara, etc. La base de PWM es la variación del ciclo de trabajo (*duty cycle*) de una señal cuadrada, en la figura 4.14 se muestra un periodo de una señal cuadrada con la definición del ciclo útil.

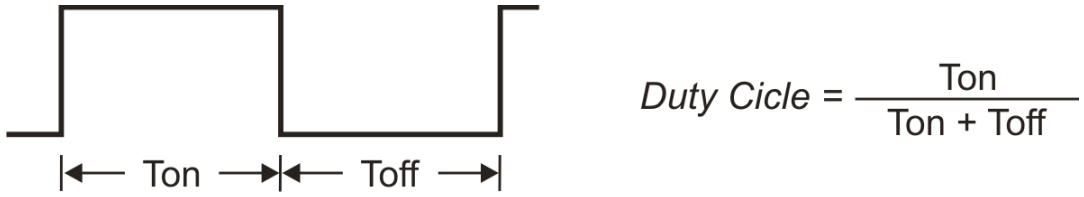


Figura 4.14 Definición del ciclo de trabajo

Al cambiar el ciclo de trabajo se modifica el voltaje promedio (V_{AVG}), el cual se obtiene con la ecuación:

$$V_{AVG} = \frac{1}{T}\int_0^T Vp\ dt = Vp\ \frac{Ton}{T}$$

Para un ciclo de trabajo del 50% el tiempo en alto es $T_{ON} = T/2$ y por lo tanto $V_{AVG} = Vp/2$.

En la mayoría de aplicaciones se conecta directamente la salida de PWM con el dispositivo a controlar, sin embargo, al conectar un filtro pasivo RC pasa bajas, como el mostrado en la figura 4.15, se genera una señal analógica.

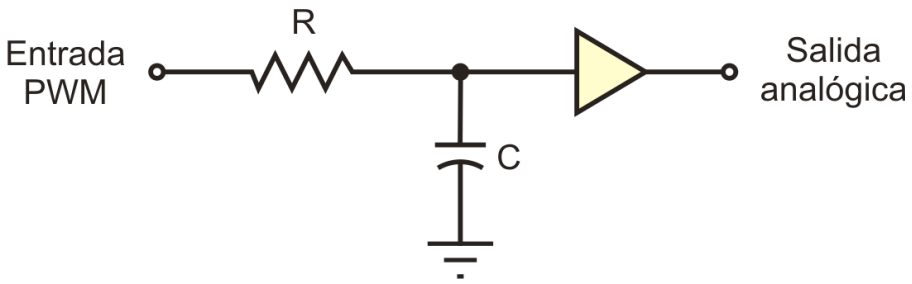


Figura 4.15 Filtro pasa bajas para generar una señal analógica

4.3.1 Generación de PWM con los Microcontroladores AVR

Los microcontroladores AVR pueden generar señales PWM en 3 modos diferentes:

- PWM rápido (*fast PWM*).
- PWM con fase correcta (*phase correct PWM*).
- PWM con fase y frecuencia correcta (*phase and frequency correct PWM*).

Las señales PWM se generan al configurar adecuadamente los temporizadores. Las salidas PWM quedan disponibles en las terminales OCx, en las figuras 4.10, 4.11 y 4.12 puede verse que estas salidas provienen de los módulos generadores de formas de onda en los diferentes temporizadores.

En la tabla 4.17 se indican los modos de PWM que se pueden generar con cada temporizador. Los diferentes modos se describen en las siguientes secciones.

Tabla 4.17 Modos de PWM para los diferentes temporizadores

AVR	Temporizador	PWM rápido	PWM con fase correcta	PWM con fase y frecuencia correcta
ATMega8	0			
	1	X	X	X
	2	X	X	
ATMega16	0	X	X	
	1	X	X	X
	2	X	X	

4.3.2 PWM Rápido

Es un modo para generar una señal PWM a una frecuencia alta, en este modo el temporizador cuenta de 0 a su valor máximo (MAX) y se reinicia, el conteo se realiza continuamente, de manera que, en algún instante de tiempo, el temporizador coincide con el contenido del registro de comparación (**OCRx**).

La señal PWM se genera de la siguiente manera: La salida OCx es puesta en alto cuando el registro del temporizador realiza una transición de MAX a 0 y es puesta en bajo cuando ocurre una coincidencia por comparación. Este modo se conoce como **No Invertido**. En el modo **Invertido** ocurre lo contrario para OCx, ambos modos se muestran en la figura 4.16, en donde puede notarse que el ancho del pulso está determinado por el valor del registro **OCRx**.

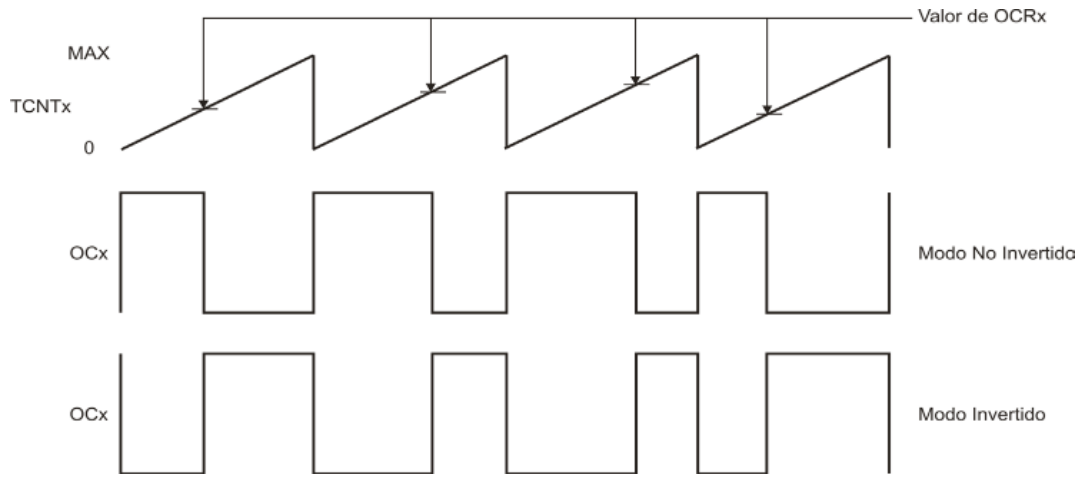


Figura 4.16 Señales de PWM rápido

Por el comportamiento del temporizador, al modo PWM rápido también se le conoce como un modo de pendiente única.

Puesto que el temporizador cuenta de 0 a MAX, la frecuencia de la señal de salida está dada por:

$$f_{PWM} = \frac{f_{clk}}{MAX + 1}$$

La frecuencia de salida se puede cambiar con el pre-escalador, con éste se modificaría el valor de f_{clk}.

Un problema se podría generar si se pretende escribir en el registro **OCRx** un valor menor al que en ese instante contiene el temporizador, esto se reflejaría en una variación de la frecuencia de la señal de salida. Para evitarlo, el acceso al registro **OCRx** se realiza por medio de un buffer doble. Cualquier instrucción que intente escribir en **OCRx** lo hace en un buffer intermedio, la escritura real en el registro **OCRx** se realiza en el momento que el temporizador pasa de MAX a 0, con ello, las señales siempre se generan con el mismo periodo.

4.3.3 PWM con Fase Correcta

En este modo, el temporizador cuenta en forma ascendente (de 0 a MAX), para después contar en forma descendente (de MAX a 0). La modificación de la terminal de salida OCx se realiza en las coincidencias por comparación con el registro **OCRx**.

En el modo **No Invertido** la salida OCx se pone en bajo tras una coincidencia mientras el temporizador se incrementa y en alto cuando ocurra la coincidencia durante el decremento. En el modo **Invertido** ocurre lo contrario para la salida OCx. El comportamiento de las señales de salida en este modo de PWM se muestra en la figura 4.17, en donde se observa que, para el modo no invertido, el pulso está centrado con el valor cero del temporizador (están en fase).

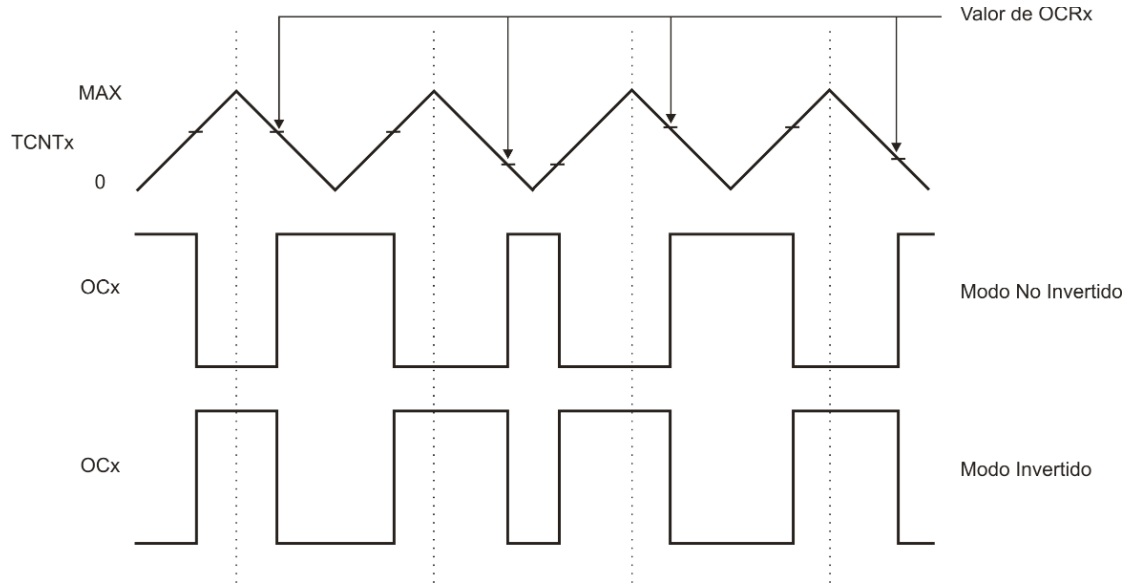


Figura 4.17 Señales de PWM con fase correcta

Por el comportamiento del temporizador, a este modo de PWM también se le conoce como un modo de pendiente doble.

La frecuencia de la señal generada está dada por:

$$f_{PWM} = \frac{f_{clk}}{2\,(MAX + 1)}$$

La cual también puede modificarse con el pre-escalador.

En este modo también se cuenta con un buffer doble para la escritura en el registro **OCRx**, las escrituras reales se realizan cuando el temporizador alcanza su valor máximo.

Cuando este modo es manejado por el temporizador 1 es posible modificar el valor del máximo durante la generación de las señales, con ello se cambia la frecuencia de la señal PWM, como se muestra en la figura 4.18. No obstante, la modificación del máximo en tiempo de ejecución da lugar a la generación de señales que no son simétricas.

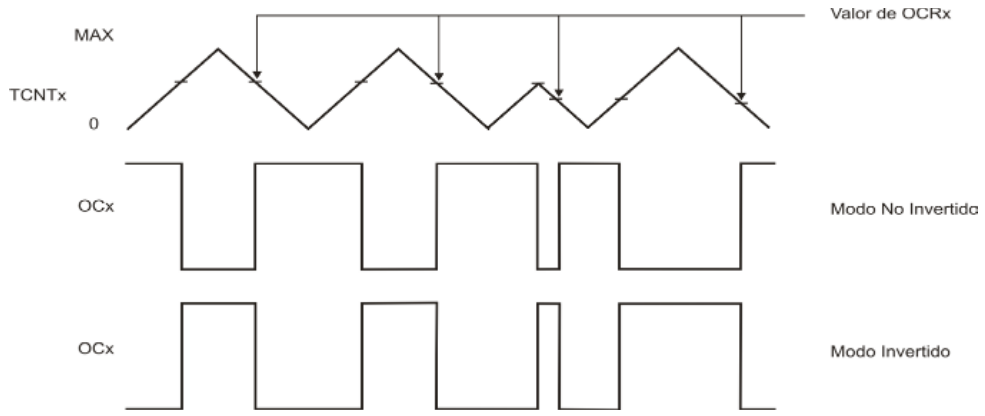


Figura 4.18 Señales de PWM con fase correcta y máximo variable

4.3.4 PWM con Fase y Frecuencia Correcta

Este modo es muy similar al modo de fase correcta, ambos modos son de pendiente doble, es decir, el temporizador cuenta de manera ascendente, alcanza su valor máximo, y luego cuenta en forma descendente.

Difieren en el momento de actualizar al registro **OCRx**, el modo de fase correcta actualiza al registro **OCRx** cuando **TCNTx** llega a su valor máximo, y el modo de fase

y frecuencia correcta lo hace cuando el registro **TCNTx** llega a cero. En la figura 4.19 se observan las señales generadas en el modo de PWM con fase y frecuencia correcta.

Si se utiliza un máximo constante, se tiene el mismo efecto al emplear uno u otro modo, pero si el máximo es variable, con el modo de fase correcta se pueden generar formas asimétricas, lo cual no ocurre con el modo de frecuencia y fase correcta. En la figura 4.19 puede notarse que la señal de salida es simétrica aun después de modificar el valor máximo del temporizador.

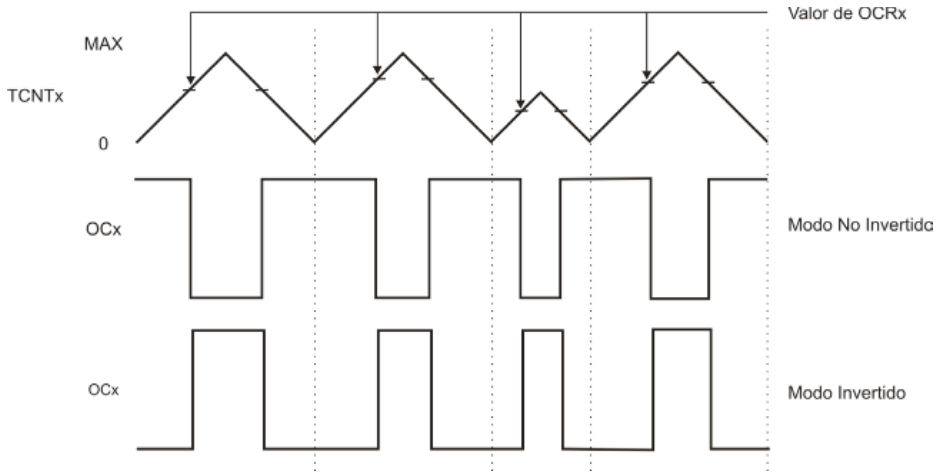


Figura 4.19 Señales de PWM con frecuencia y fase correcta

Este modo sólo puede ser generado por el temporizador 1, que es de 16 bits. Y únicamente está disponible con máximos variables, los cuales se definen en los registros **OCR1A** o **ICR1**.

4.3.5 El Temporizador 0 y la Generación de PWM

En un ATMega8 el temporizador 0 no puede generar señales PWM. En un ATMega16, con el temporizador 0 es posible generar los modos de PWM rápido y PWM con fase correcta.

El registro de control del temporizador 0 es el **TCCR0**, cuyos bits son:

	7	6	5	4	3	2	1	0	
0x33	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	**TCCR0**

La función de estos bits fue descrita en la sección 4.2.6. Los bits **WGM0[1:0]** se listaron en la tabla 4.7, con ellos se definen los modos de generación de forma de onda, los 2 que determinan la generación de señales PWM se muestran en la tabla 4.18.

Tabla 4.18 Modos para la generación de PWM con el temporizador 0

Modo	WGM01	WGM00	Descripción
1	0	1	PWM con fase correcta
3	1	1	PWM rápido

Bajo ambos modos de PWM, los bits **COM0[1:0]** determinan el comportamiento de la salida OC0, este comportamiento es descrito en la tabla 4.19.

Tabla 4.19 Comportamiento de la salida OC0 en los modos de PWM

COM01	COM00	Descripción
0	0	Operación Normal, terminal OC0 desconectada
0	1	Reservado (sin uso)
1	0	Modo no invertido
1	1	Modo invertido

El temporizador 0 es de 8 bits, por ello, el máximo valor del temporizador es de 255.

4.3.6 El Temporizador 1 y la Generación de PWM

Para el ATMega8 y ATMega16, con el temporizador 1 se pueden generar 2 señales en cualquiera de los 3 modos de PWM descritos con anterioridad: PWM rápido, PWM con fase correcta y PWM con frecuencia y fase correcta. Las 2 señales se pueden generar en forma simultánea en las terminales OC1A y OC1B, porque el temporizador 1 cuenta con 2 registros para comparación: **OCR1A** y **OCR1B**. Con estos registros se determina el ancho de pulso en cada señal, aunque las señales van a tener la misma frecuencia.

Los registros de control del temporizador 1 son el **TCCR1A** y **TCCR1B**, los bits de estos registros son:

	7	6	5	4	3	2	1	0	
0x2F	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	**TCCR1A**
0x2E	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	**TCCR1B**

La función de estos bits fue descrita en la sección 4.2.7. Los bits **WGM1[3:0]** definen los modos de generación de forma de onda, en la tabla 4.10 se listaron aquellos modos que no están relacionados con PWM, los modos que determinan la generación de señales PWM se muestran en la tabla 4.20.

Tabla 4.20 Modos para la generación de PWM con el temporizador 1

Modo	WGM13	WGM12	WGM11	WGM10	Descripción	Máximo
1	0	0	0	1	PWM con fase correcta de 8 bits	0x00FF
2	0	0	1	0	PWM con fase correcta de 9 bits	0x01FF
3	0	0	1	1	PWM con fase correcta de 10 bits	0x03FF
5	0	1	0	1	PWM rápido de 8 bits	0x00FF
6	0	1	1	0	PWM rápido de 9 bits	0x01FF
7	0	1	1	1	PWM rápido de 10 bits	0x03FF
8	1	0	0	0	PWM con frecuencia y fase correcta	ICR1
9	1	0	0	1	PWM con frecuencia y fase correcta	OCR1A
10	1	0	1	0	PWM con fase correcta	ICR1
11	1	0	1	1	PWM con fase correcta	OCR1A
14	1	1	1	0	PWM rápido	ICR1
15	1	1	1	1	PWM rápido	OCR1A

En la tabla 4.20 se observa el uso de 3 máximos fijos: 0x00FF, 0x01FF y 0x03FF, para que el temporizador trabaje con 8, 9 ó 10 bits, respectivamente. Los máximos fijos son utilizados en los modos de PWM rápido y PWM con fase correcta. No hay máximos fijos para el modo de PWM con frecuencia y fase correcta, ya que su comportamiento sería el mismo que el modo de fase correcta.

En los 3 modos de PWM se pueden emplear máximos variables, proporcionados por los registros **ICR1** u **OCR1A**. Sin embargo, si se destina a **OCR1A** para definir al máximo, únicamente se puede generar una señal PWM en la salida OC1B, utilizando a **OCR1B** como registro para modular el ancho del pulso.

Si alguna aplicación requiere la generación de 2 señales PWM a una frecuencia que no coincida con las proporcionadas por los valores fijos, aun considerando las variantes que proporciona el pre-escalador, debe emplearse al registro **ICR1** para definir el valor máximo del temporizador, anulando con ello la posibilidad de emplear los recursos de captura.

Los bits **COM1A[1:0]** y **COM1B[1:0]** determinan el comportamiento de las salidas OC1A y OC1B, respectivamente. Este comportamiento es descrito en la tabla 4.21.

Tabla 4.21 Comportamiento de la salidas OC1A/OC1B en los modos de PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Descripción
0	0	Operación Normal, terminal OC1A/OC1B desconectada
0	1	Reservado (sin uso)
1	0	Modo no invertido
1	1	Modo invertido

4.3.7 El Temporizador 2 y la Generación de PWM

Con el temporizador 2 es posible generar los modos de PWM rápido y PWM con fase correcta (ATMega8 y ATMega16). El registro de control del temporizador 2 es el **TCCR2**, cuyos bits son:

	7	6	5	4	3	2	1	0	
0x25	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	**TCCR2**

La función de estos bits fue descrita en la sección 4.2.8. Los bits **WGM2[1:0]** se listaron en la tabla 4.13, con ellos se definen los modos de generación de forma de onda, los 2 que determinan la generación de señales PWM se muestran en la tabla 4.22.

Tabla 4.22 Modos para la generación de PWM con el temporizador 2

Modo	WGM21	WGM20	Descripción
1	0	1	PWM con fase correcta
3	1	1	PWM rápido

Bajo ambos modos de PWM, los bits **COM2[1:0]** determinan el comportamiento de la salida OC2, este comportamiento es descrito en la tabla 4.23.

Tabla 4.23 Comportamiento de la salida OC2 en los modos de PWM

COM21	COM20	Descripción
0	0	Operación Normal, terminal OC2 desconectada
0	1	Reservado (sin uso)
1	0	Modo no invertido
1	1	Modo invertido

El temporizador 2 es de 8 bits, por ello, el máximo valor del temporizador es de 255.

4.3.8 Ejemplos de Uso de las Señales PWM

En esta sección se muestran 2 ejemplos para ilustrar el uso de la generación de señales moduladas en ancho de pulso.

Ejemplo 4.6 Empleando un ATMega8, generar una señal PWM en modo no invertido, con el ancho de pulso determinado por el valor del puerto D.

Puesto que no se han establecido otras restricciones, se utiliza al temporizador 2 (de 8 bits) empleando el modo de PWM rápido, la salida queda disponible en la terminal OC2 (PB3). La solución en lenguaje C es:

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

int  main() {

       DDRB = 0xFF;          //  Puerto B como salida

       PORTD = 0xFF;         // Resistores de Pull-Up en el puerto D
                             // el Puerto D por default es entrada

       TCCR2 = 0x69;         // PWM rápido, en modo No Invertido y sin
                             // pre-escalador

       while(1) {
              OCR2 = PIND;  // El ancho de pulso lo determina el puerto D
       }
}
```

Como no se emplea al pre-escalador, suponiendo que el microcontrolador va a trabajar con su oscilador interno de 1 MHz, la frecuencia de la señal PWM generada es de:

$$f_{PWM} = \frac{1\ MHz}{256} = 3.90\ KHz$$

La solución en ensamblador sigue la misma estructura, el código es:

```
       .include <m8def.inc>

       LDI    R16, 0xFF
       OUT    DDRB, R16     ; Puerto B como salida

       OUT    PORTD, R16    ; Resistores de Pull-Up en el puerto D
                            ; El Puerto D por default es entrada

       LDI    R16, 0x69;PWM rápido, en modo No Invertido y sin pre-escalador
       OUT    TCCR2, R16
loop:
       IN     R16, PIND     ; El ancho de pulso lo determina el puerto D
       OUT    OCR2, R16
       RJMP   loop
```

Ejemplo 4.7 Para el manejo de un servomotor se requiere de una señal PWM con un periodo de 20 mS. Con este periodo, el servomotor se mantiene en su extremo izquierdo (0°) si el tiempo en alto es de 0.9 mS, en su posición central (90°) con un tiempo de 1.5 mS y en su extremo derecho (180°) con 2.1 mS. En la figura 4.20 se muestran las señales requeridas por el servomotor.

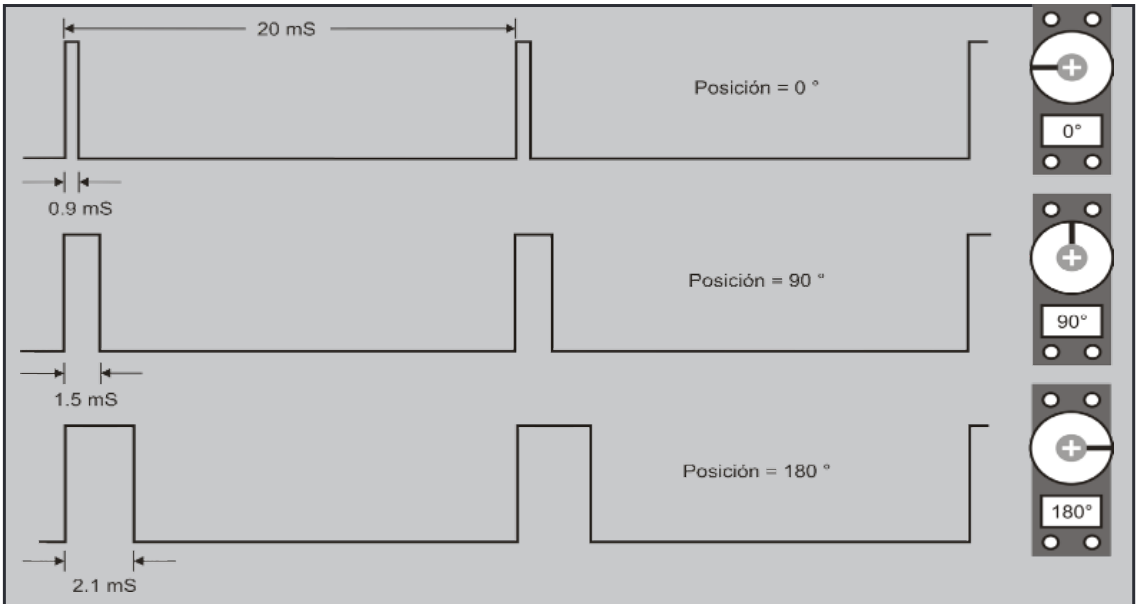


Figura 4.20 Señales para manejar un servomotor

Muestre la configuración del temporizador 1 para generar esta señal e indique los valores de los registros de comparación para la posición central y para cada uno de los extremos.

Se asume que el microcontrolador está operando con el oscilador interno de 1 MHz, por lo tanto, si no se utiliza al pre-escalador, el temporizador se incrementa cada 1 µS.

Para un periodo de 20 mS, utilizando PWM rápido, el temporizador 1 debe contar de 0 a 19 999, rango alcanzable con 16 bits.

La configuración requerida para generar la señal en OC1A es:

```
ICR1 = 19999;          // Valor máximo para el temporizador 1
TCCR1A = 0x82;         // COM1A[1:0] = "10" Modo no invertido,
                       // salida en OC1A
TCCR1B = 0x19;         // WGM1[3:0] ="1110",PWM rápido con máximo en ICR1
                       // Señal de reloj sin pre-escalador
```

Para que el servomotor esté en su extremo izquierdo (0 °): **OCR1A** = 899;

Para que el servomotor esté en su posición neutral (90 °): **OCR1A** = 1499;

Para que el servomotor esté en su extremo derecho (180 °): **OCR1A**= 2099;

Entre ambos extremos se tienen 1200 combinaciones posibles (2099 – 899), dado que con estos valores se consiguen posiciones intermedias, el servomotor puede manejarse con una resolución de 180 °/1200 = 0.15 °.

4.4 Ejercicios

Los problemas que a continuación se presentan están relacionados con los recursos que se han descrito en el presente capítulo, en algunos es necesario el uso de más de un recurso, haciendo necesario un análisis para determinar la función de cada recurso. Pueden resolverse con un ATMega8 o un ATMega16, empleando lenguaje ensamblador o lenguaje C.

1. Realice un sistema que simultáneamente genere 2 señales, la primera a 10 KHz y la segunda a 500 Hz, suponga que el microcontrolador va a trabajar con el oscilador interno de 1 MHz.

2. Con base en la figura 4.21, desarrolle el programa que controle el encendido de un horno para mantenerlo alrededor de una temperatura de referencia. El horno está acondicionado para generar 2 señales (*hot* y *cold*), debe encenderse si la temperatura está por debajo del nivel *cold* y apagarse si está por encima del nivel *hot*.

3. Empleando los recursos de captura del temporizador 1, desarrolle un programa que reciba y decodifique una secuencia serial de 8 bits de información modulada por el ancho de pulsos activos en bajo, como se muestra en la figura 4.22 (señales similares son manejadas por controles remoto comerciales). Después de detectar al bit de inicio, deben obtenerse los 8 bits de datos iniciando con el bit menos significativo, concluida la recepción, debe mostrarse el dato en cualquiera de los puertos libres (*sugerencia:* utilice 2 mS y 1 mS como referencias, si el ancho del pulso es mayor a 2 mS es un bit de inicio, menor a 2 mS pero mayor a 1 mS es un 1 lógico, menor a 1 mS, se trata de un 0 lógico).

4. En un supermercado se ha determinado premiar a cada cliente múltiplo de 200. Desarrolle un sistema basado en un AVR, el cual debe detectar al cliente número 200 y generar un tono de 440 Hz (aproximadamente), por 5 segundos, cuando eso ocurra. Los clientes deben presionar un botón para ser considerados. Se trata de una aplicación para los temporizadores, utilice al temporizador 0, manejado con eventos externos, para llevar el conteo de clientes. Al temporizador 1 para generar el tono de 440 Hz y al temporizador 2 para el conteo de segundos, empleando un oscilador externo de 32.768 KHz.

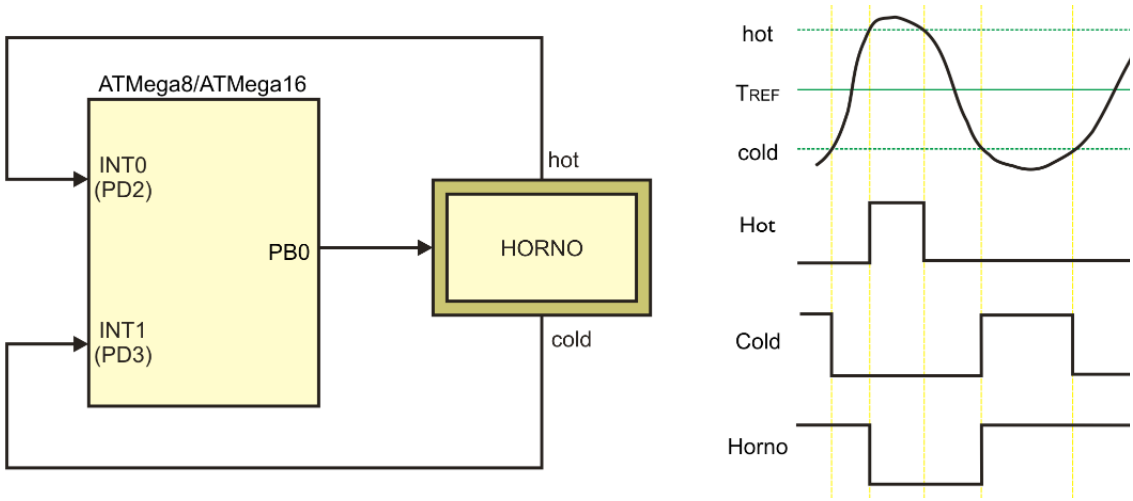


Figura 4.21 Control de un horno

5. Empleando una señal PWM, controle el encendido de un conjunto de LEDs ultra-brillantes, manejando 5 niveles de intensidad. El sistema debe contar con un botón para el cambio de intensidad, cada vez que el botón es presionado, la intensidad se debe incrementar el 20 % del valor máximo posible. Los LEDs deben estar apagados cuando se el sistema se enciende (0 % en el ancho de pulso). Si se alcanza la máxima intensidad (100 %) y se presiona nuevamente al botón, los LEDs deben apagarse.

   Configure para que la señal de salida tenga un periodo en el orden de décimas de milisegundos, para una operación adecuada de los LEDs.

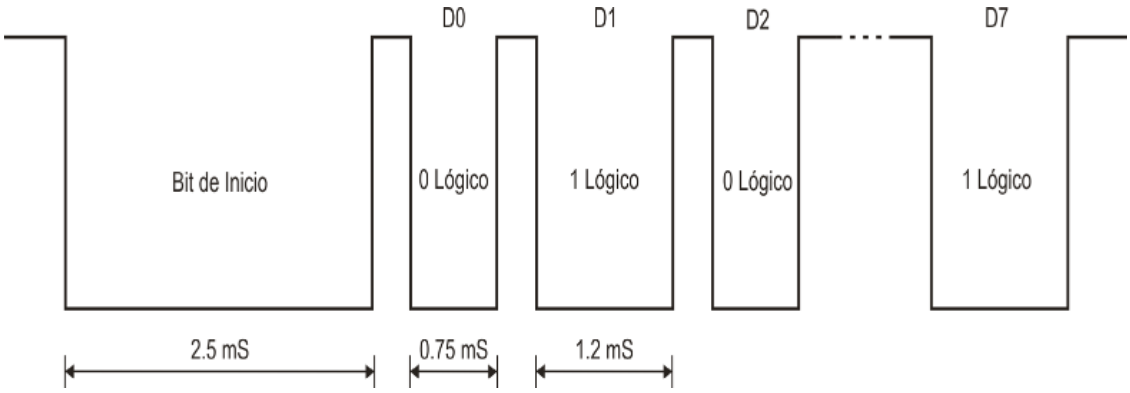


Figura 4.22 Señal modulada, la información está en los niveles bajos de voltaje

5. Recursos para el Manejo de Información Analógica

Aunque los microcontroladores AVR son elementos para el procesamiento de información digital, incluyen 2 recursos que les permiten obtener información generada por dispositivos analógicos. Se trata de un convertidor analógico a digital (ADC, *analog-to-digital converter*) y de un comparador analógico (AC, *analog comparator*), estos recursos se describen en el presente capítulo.

5.1 Convertidor Analógico a Digital

Un ADC recibe una muestra de una señal analógica, es decir, su valor en un instante de tiempo, a partir de la cual genera un número (valor digital). La función de un ADC es contraria a la que realiza un convertidor digital a analógico (DAC, *digital-to-analog converter*), un DAC genera un nivel de voltaje analógico a partir de un número o valor digital. En la figura 5.1 se ilustra la función de un ADC y un DAC, los microcontroladores AVR no incluyen un DAC, pero se revisa su funcionamiento porque el ADC embebido utiliza un DAC para poder operar.

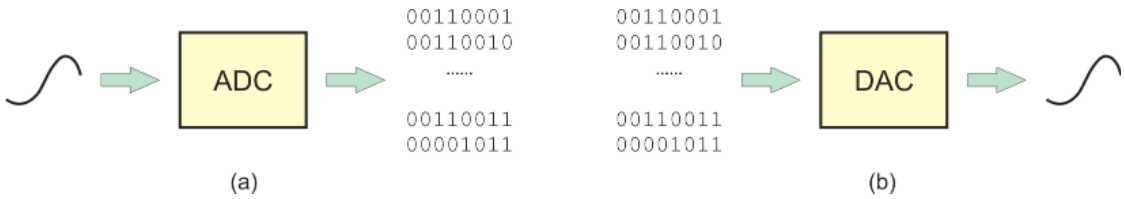


Figura 5.1 Funcionalidad de (a) un Convertidor Analógico a Digital (ADC), y de (b) un Convertidor Digital a Analógico (DAC)

5.1.1 Proceso de Conversión Analógico a Digital

Una señal analógica toma valores continuos a lo largo del tiempo, el proceso para convertirla a digital involucra 2 etapas: el muestreo y la cuantificación.

El muestreo básicamente consiste en tomar el valor de la señal analógica en un instante de tiempo, y mantenerlo, mientras el proceso concluye. Cada muestra se va tomando cuando ocurre un intervalo de tiempo predefinido, conocido como periodo de muestreo, cuyo inverso es la *frecuencia de muestreo.*

Para que la información contenida en la señal a digitalizar sea recuperada de manera correcta, se requiere que la frecuencia de muestreo sea por lo menos el doble de la frecuencia de la señal analógica. Por ejemplo, si se va a digitalizar una señal de audio acotada a 2 KHz, la frecuencia de muestreo por lo menos debe ser de 4 KHz.

La cuantificación consiste en la asociación de cada muestra con un número o valor digital, que pertenece a un conjunto finito de valores, el número de bits utilizado por el convertidor determina el total de valores. Cada muestra analógica se asocia con el valor digital más cercano. Por ejemplo, si el convertidor es de 8 bits, el conjunto tiene $2^8 = 256$ valores diferentes (0 a 255).

El número de bits y el voltaje máximo a convertir (V_{MAX}) determinan la *resolución* del ADC. La resolución se define como el cambio requerido en la señal analógica para que la señal digital se incremente en un bit. Por ejemplo, para un ADC de 8 bits capaz de recibir un voltaje máximo de 5 V, su resolución es de:

$$Resolución = \frac{V_{MAX}}{2^8 - 1} = \frac{5\,V}{255} = 19.60\,mV$$

En la figura 5.2 se muestran las etapas involucradas en el proceso de conversión analógica a digital, se toman 10 muestras de una señal analógica y se utilizan 3 bits para su cuantificación.

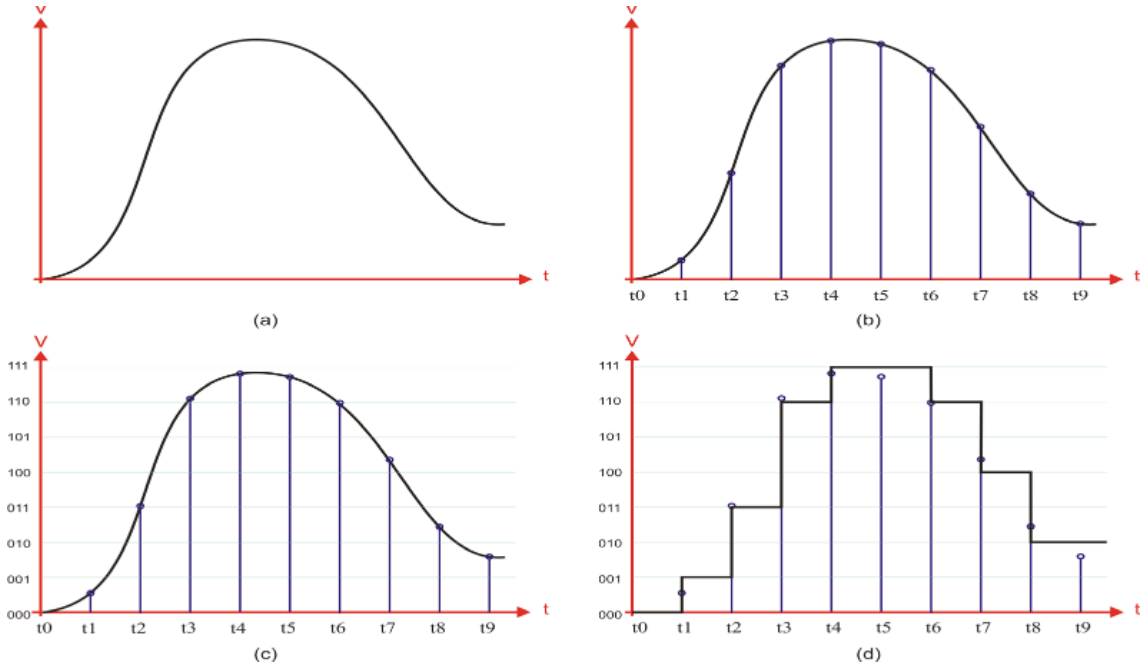


Figura 5.2 Proceso de conversión: (a) una señal analógica, (b) muestreo, (c) cuantificación y (d) señal digital

5.1.2 Hardware para la Conversión Digital a Analógico

El hardware para convertir una señal digital en su correspondiente valor analógico involucra una red resistiva R-2R, como se muestra en la figura 5.3. Por su estructura, la corriente se va dividiendo a la mitad en cada malla de la red. Por medio de interruptores digitales se define qué mallas contribuyen en la corriente de salida, los interruptores son controlados con el dato digital a convertir. Las corrientes se suman y se convierten a voltaje con la ayuda de un amplificador.

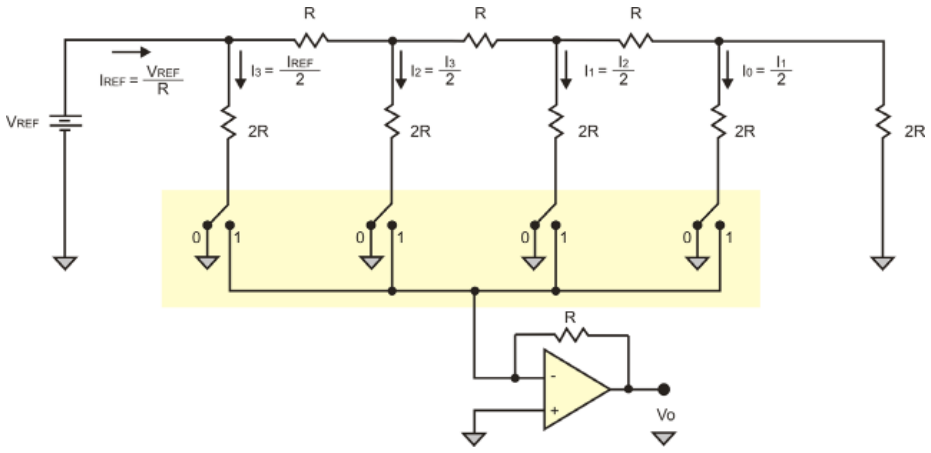


Figura 5.3 Red resistiva R-2R para una conversión digital a analógica

5.1.3 Hardware para la Conversión Analógico a Digital

Existen 3 configuraciones clásicas para estos convertidores, las cuales se caracterizan por su velocidad de conversión. Un ADC del tipo *integrador* es el más simple, en lo que a hardware se refiere, pero es demasiado lento, su tiempo de conversión está en el orden de milisegundos, por lo que sólo es aplicable si la señal analógica tiene variaciones mínimas a lo largo del tiempo. Otro tipo es el ADC de *aproximaciones sucesivas*, éste es más rápido, su tiempo de conversión está en el orden de microsegundos, puede emplearse para digitalizar señales de audio. Un convertidor de este tipo se encuentra empotrado en los microcontroladores AVR, por ello, en el siguiente apartado se describe su funcionamiento. El tercer tipo de ADC es un convertidor *paralelo* o *flash*, estos convertidores requieren de un hardware mucho más complejo, aunque por sus altas velocidades pueden ser empleados para digitalizar señales de video.

5.1.3.1 ADC de Aproximaciones Sucesivas

En la figura 5.4 se muestra la organización de un ADC de aproximaciones sucesivas de 4 bits. La señal analógica debe ubicarse en V_I y la salida digital resultante queda disponible en los bits *D3, D2, D1* y *D0*. La generación del dato digital no es inmediata, el proceso de conversión requiere de varios ciclos de reloj. Por ello, un ADC debe incluir señales de control (*inicio* y *fin*) para sincronizarse con otros dispositivos.

El bloque de muestreo y retención (*sample and hold*) tiene por objetivo mantener la muestra analógica durante el tiempo de conversión, es necesario porque la información analógica podría cambiar mientras se realiza la conversión y con ello, se generaría información incongruente.

La conversión inicia cuando la señal *inicio* es puesta en alto, la muestra es retenida y se realiza la primera aproximación poniendo en alto al bit más significativo del dato digital (*D3*). El DAC genera el valor analógico que corresponda a la combinación "1000", con el comparador se determina si este valor es mayor o menor que el valor analógico a convertir. Si el valor generado por el DAC es mayor, se remplaza el 1 de *D3* por un 0, en caso contrario el 1 se conserva. Hasta este momento se ha realizado la primera aproximación.

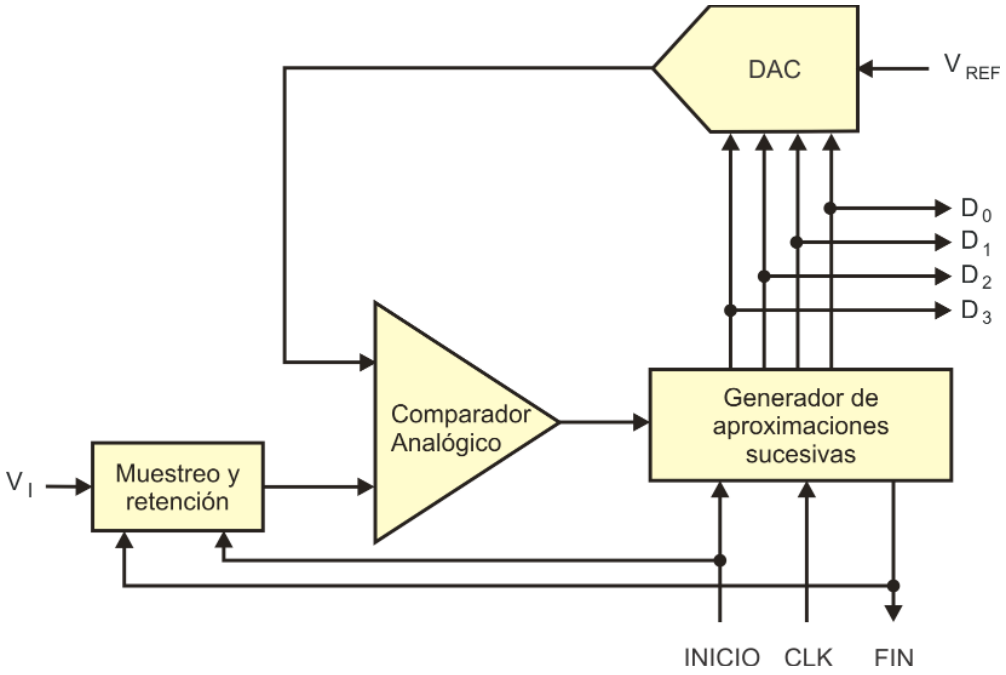


Figura 5.4 Organización de un ADC de aproximaciones sucesivas

El generador de aproximaciones sucesivas continúa colocando un 1 en *D2*, para nuevamente comparar el valor generado por el DAC con el valor a convertir y determinar si este 1 se mantiene o se remplaza por un 0. El mismo proceso debe realizarse para *D1* y *D0*. Una vez que se han generado todos los bits, se indica el fin de la conversión poniendo en alto a la señal *fin* al menos por 1 ciclo de reloj. El voltaje V_{REF} que se suministra al DAC determina el voltaje máximo que puede ser introducido en V_I.

Un comportamiento similar al de un ADC de aproximaciones sucesivas lo realizan las máquinas de cobro automático al momento de regresar el cambio después de recibir un pago (con la diferencia de que un billete o moneda puede figurar más de una vez). Por ejemplo, suponiendo que una máquina puede proporcionar billetes de $50.00 y $20.00 y monedas de $10.00, $5.00 y $1.00, si la máquina va a dar un cambio de $36.00, primero evalúa si alcanza un billete de a $50.00, puesto que no es así, lo descarta y pasa al siguiente. Prueba con uno de a $20.00 que si alcanza, lo proporciona y resta, quedando un residuo de $16.00. Otro de a $20.00 tampoco alcanza, por lo tanto proporciona una moneda de a $10.00. Queda un residuo de $6.00 que es cubierto con una moneda de $5.00 y otra de $1.00.

Algunos ADCs incluyen otras entradas de control como la habilitación del dispositivo (CE*, chip enable*) o la habilitación de la salida (OE*, output enable*) para que sean fácilmente conectados con microprocesadores o microcontroladores.

En ocasiones, la salida *fin* es conectada con la entrada *inicio*, de manera que cuando finaliza una conversión inmediatamente da inicio la siguiente, a este modo de operación se le conoce como “carrera libre”.

5.1.4 El ADC de un AVR

Los microcontroladores ATMega8 y ATMega16 incluyen un ADC de aproximaciones sucesivas de 10 bits, el cual está conectado a un multiplexor analógico que permite seleccionar 1 de 8 canales externos (ADC0, ADC1, ADC2, etc.), excluyendo al ATMega8 con encapsulado PDIP, éste sólo tiene 6 canales. En un ATMega16 se pueden emplear 3 canales para introducir una entrada diferencial (no referida a tierra) e incorporar un factor de ganancia configurable.

En la figura 5.5 se muestran los elementos para seleccionar el voltaje analógico en la entrada, la parte sombreada no está disponible en un ATMega8, para estos dispositivos, el voltaje analógico proviene directamente del multiplexor principal.

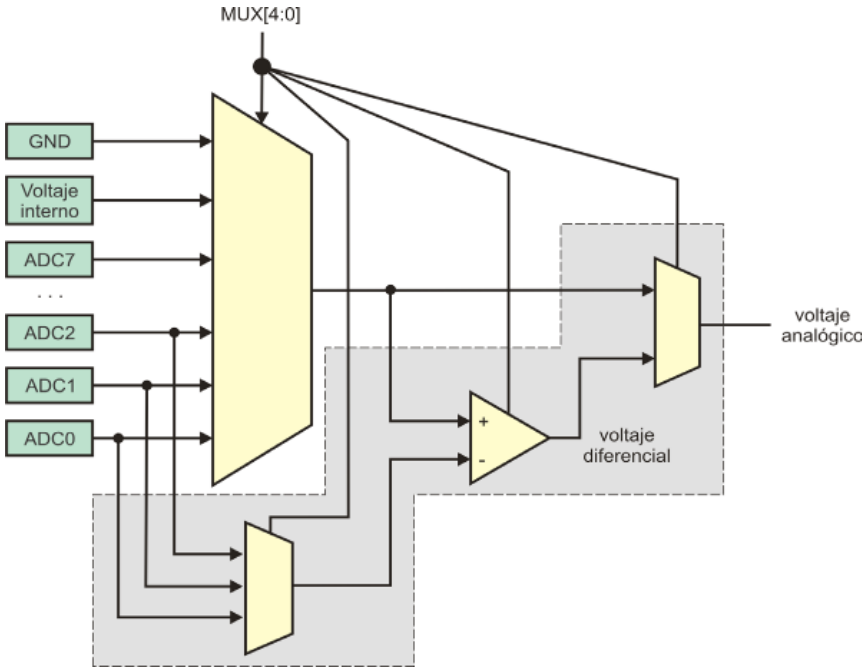


Figura 5.5 Etapa disponible para seleccionar el canal con la información analógica

Los bits **MUX[4:0]** son parte del registro **ADMUX**, éste es un Registro I/O disponible para el manejo del ADC y se describe en la siguiente sección. Como en un ATMega8 se tienen menos opciones, el bit **MUX(4)** no está implementado, en la tabla 5.1 se muestra la selección de la entrada para el ADC de un ATMega8 y en la tabla 5.2 para un ATMega16.

Tabla 5.1 Selección de la entrada para el ADC de un ATMega8

MUX[3:0]	Entrada
0000	ADC0
0001	ADC1
0010	ADC2
. . .	. . .
0111	ADC7
1000 - 1101	Sin uso
1110	Voltaje interno (1.23 V)
1111	0 V

Tabla 5.2 Selección de la entrada para el ADC de un ATMega16

MUX[4:0]	Entrada referida a tierra	Entrada diferencial (positiva)	Entrada diferencial (negativa)	Ganancia
00000	ADC0	No aplica		
00001	ADC1			
. . .	. . .			
00111	ADC7			

MUX[4:0]	Entrada referida a tierra	Entrada diferencial (positiva)	Entrada diferencial (negativa)	Ganancia
01000	No aplica	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	Voltaje interno (1.23 V)	No aplica		
11111	0 V			

Aunque el microcontrolador puede operar con osciladores en el orden de MHz, el ADC alcanza su máxima resolución si trabaja a una frecuencia entre 50 KHz y 200 KHz. Es posible emplear una frecuencia mayor con una resolución de 8 bits, pero esta frecuencia debería determinarse en forma práctica. Los AVR incluyen un pre-escalador de 7 bits para generar la frecuencia de trabajo del ADC a partir de la frecuencia del microcontrolador, éste se muestra en la figura 5.6.

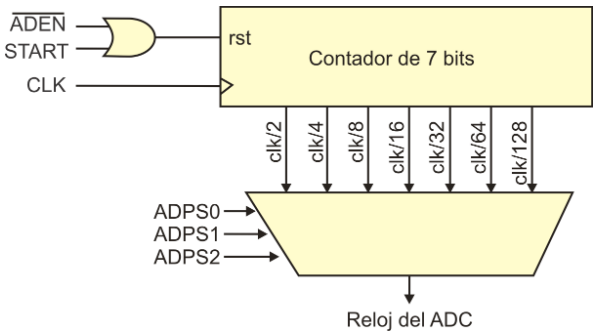


Figura 5.6 Pre-escalador del ADC

Las señales de control del pre-escalador son parte de los bits del Registro A de Control y Estado del ADC (**ADCSRA**, *ADC Control and Status Register A*), el cual se revisa en la siguiente sección. Los factores de división se seleccionan con los bits **ADPS[2:0]**, las diferentes opciones se muestran en la tabla 5.3.

Tabla 5.3 Selección del factor de división en el pre-escalador del ADC

ADPS2	ADPS1	ADPS0	Factor de división
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Con la primera conversión se inicializa la circuitería analógica, por lo que requiere de 25 ciclos de reloj, para las conversiones siguientes sólo se emplean 13 ciclos.

El ADC y el multiplexor para la selección de la entrada analógica reciben su alimentación en la terminal AVcc. La terminal está disponible para que la circuitería analógica pueda alimentarse con un voltaje diferente al de la parte digital, proporcionando las facilidades para un posible aislamiento. Sólo debe considerarse que AVcc no debe diferir más de ±0.3 V de Vcc.

En la mayoría de aplicaciones es suficiente con conectar a AVcc directamente con Vcc. De hecho, es recomendable realizar esta conexión aun si no se va a emplear al ADC, para la adecuada operación del puerto C de un ATMega8 y del puerto A de un ATMega16, porque en estos puertos se encuentran los multiplexores para las entradas analógicas, como una función alternativa.

Para la conexión de AVcc con Vcc el fabricante recomienda el uso de un filtro pasa bajas, como el mostrado en la figura 5.7, con la finalidad de cancelar el ruido. El filtro es importante si la entrada analógica tiene un valor máximo pequeño, en relación al voltaje de alimentación, o bien, si se utiliza una entrada diferencial, en el caso de un ATMega16.

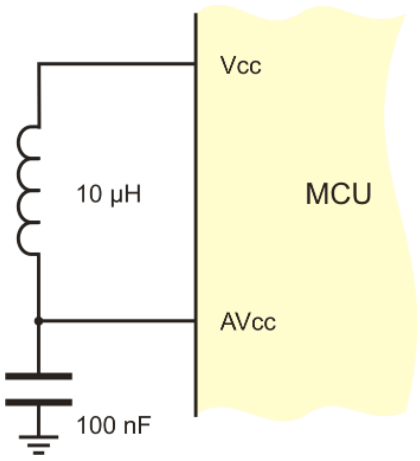


Figura 5.7 Filtro pasa bajas sugerido para conectar AVcc con Vcc

El ADC utiliza un DAC durante el proceso de conversión, por ser un ADC de aproximaciones sucesivas como el mostrado en la figura 5.4. El DAC requiere un voltaje de referencia (VREF), el cual puede ser proporcionado por diferentes fuentes, como se muestra en la figura 5.8, la selección de VREF se realiza con los bits **REFS1** y **REFS0**, del registro **ADMUX**. El voltaje de referencia determina el rango de conversión del ADC, si el voltaje analógico excede a VREF, es codificado como 0x3FF.

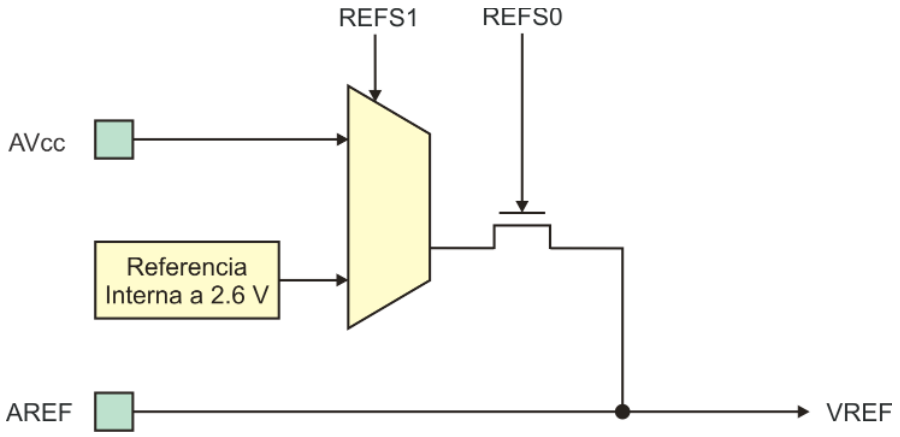


Figura 5.8 Hardware para el voltaje de referencia

El voltaje de referencia puede tomarse de la alimentación analógica (AVcc), de un voltaje interno o de la terminal AREF, en la tabla 5.4 se muestra la selección de estas fuentes. Si se utiliza a AVcc o al voltaje interno (opciones "01" y "11") es recomendable el uso de un capacitor de AREF a tierra, para que el voltaje de referencia tenga inmunidad al ruido.

Tabla 5.4 Alternativas para el voltaje de referencia

REFS1	REFS0	Voltaje de referencia
0	0	Voltaje externo en AREF, referencia interna apagada
0	1	Voltaje externo en AVcc
1	0	Reservado
1	1	Voltaje interno de 2.6 V

Una opción muy simple, desde el punto de vista práctico, consiste en la conexión de la terminal AREF con Vcc, empleando la combinación "00" para los bits **REFS1** y **REFS0**, esta opción también presenta inmunidad al ruido. Únicamente se requiere que la entrada analógica esté acondicionada para proporcionar un voltaje entre 0 y Vcc.

Un factor importante a determinar es la frecuencia máxima permitida en la señal analógica de entrada. Por ejemplo, si el microcontrolador está operando a 1 MHz, para alcanzar una resolución máxima el ADC debe operar con una frecuencia entre 50 y 200 KHz, con un factor de división de 8 se obtiene una frecuencia de 125 KHz. Si el ADC está dedicado sólo a un canal e ignorando el tiempo requerido por la primera muestra, los 13 ciclos por muestra conllevan a una razón de muestreo de 125 KHz/13 = 9.61 KHz. Por lo tanto, de acuerdo con el teorema del muestreo, la frecuencia máxima permisible para la señal de entrada es de 4.8 KHz.

Relacionando las señales de control de la figura 5.4 con el ADC de un AVR, se tiene que: el *inicio* de la conversión se realiza poniendo en alto al bit **ADSC** (*Start Conversion*), el *fin* se indica con la puesta en alto de la bandera **ADIF** (*Interrupt Flag*); ésta puede sondearse por software o bien, si se ajusta al bit **ADIE** (*Interrupt Enable*), va a producir una interrupción, estos bits están en el registro **ADCSRA**.

Un ATMega8 puede ser configurado para operar en un modo de carrera libre, de manera que al finalizar una conversión inicie con la siguiente, este modo se habilita con el bit **ADFR** (*ADC Free Running*). En el ATMega16, además del modo de carrera libre, se puede configurar al hardware para que el inicio de una conversión sea disparado por algún evento de otro recurso del microcontrolador.

5.1.5 Registros para el Manejo del ADC

El dato digital de 10 bits, resultante de la conversión, queda disponible en 2 Registros I/O: **ADCH** (para la parte alta) y **ADCL** (para la parte baja). En lenguaje C puede hacerse referencia a ambos tratándolos como **ADCW**. Los 10 bits de información están alineados a la derecha, de la siguiente manera:

	7	6	5	4	3	2	1	0	
0x05	-	-	-	-	-	-	ADC9	ADC8	**ADCH**
0x04	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	**ADCL**

La alineación puede cambiarse a la izquierda si se pone en alto al bit **ADLAR** (*ADC Left Adjust Result*) del registro **ADMUX**, con ello, la información se organiza como:

	7	6	5	4	3	2	1	0	
0x05	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	**ADCH**
0x04	ADC1	ADC0	-	-	-	-	-	-	**ADCL**

Esta organización es útil en aplicaciones donde resulte suficiente una resolución de 8 bits. En esos casos, sólo se emplearía al registro **ADCH** y ignorando el contenido del registro **ADHL**.

En el registro **ADMUX** (multiplexor del ADC) también se pueden seleccionar otros parámetros, sus bits son:

	7	6	5	4	3	2	1	0	
0x07	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	**ADMUX**

- **Bits 7 y 6 – REFS[1:0]: Bits para seleccionar el voltaje de referencia del ADC**

  La tabla 5.4 muestra las alternativas para el voltaje de referencia, necesario para la conversión.

- **Bit 5 – ADLAR: Bit para alinear el resultado de la conversión a la izquierda**

  Al poner en alto este bit, los 10 bits resultantes de la conversión se alinean a la izquierda, dentro de los registros **ADCH** y **ADCL**.

- **Bits 4 al 0 – MUX[4:0]: Bits para seleccionar el canal de entrada analógico**

  En un ATMega8 el bit **MUX(4)** no está implementado, para estos dispositivos se tienen las opciones descritas en la tabla 5.1. En los ATMega16 las opciones para la entrada analógica se describen en la tabla 5.2, en donde se observa que es posible seleccionar una entrada diferencial con algunos factores de ganancia.

El control del ADC se realiza con el Registro A de Control y Estado del ADC (**ADCSRA**, *ADC Control and Status Register A*), los bits de este registro son:

	7	6	5	4	3	2	1	0	
0x06	ADEN	ADSC	ADFR/ ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	**ADCSRA**

- **Bit 7 – ADEN: Habilitador del ADC**

  Con 0 el ADC está apagado y no es posible utilizarlo.

- **Bit 6 – ADSC: Inicio de conversión (*Start Conversion*)**

  La conversión inicia al escribirle 1, se limpia automáticamente por hardware. La primera conversión requiere de 25 ciclos de reloj, las siguientes de 13.

- **Bit 5 – ADFR ó ADATE: Activa el modo de carrera libre (*ADC Free Running*) o habilita un auto disparo (*ADC Auto Trigger Enable*)**

  La función de este bit depende del dispositivo, en un ATMega8 establece un modo de carrera libre, es decir, al terminar una conversión inicia con la siguiente. El bit **ADSC** debe ponerse en alto para dar paso a las conversiones.

  En un ATMega16 este bit habilita un auto disparo del ADC, es decir, el ADC inicia una conversión cuando ocurre un evento generado por otro recurso, el evento se configura con los bits **ADTS** en el registro **SFIOR**, con los cuales también puede elegirse el modo de carrera libre.

- **Bit 4 – ADIF: Bandera de fin de conversión**

  Se pone en alto indicando el fin de una conversión, puede generar una interrupción o sondearse vía software. La bandera se limpia automáticamente por hardware si se configura la interrupción. Al emplear sondeo, la bandera se limpia al escribirle nuevamente un 1 lógico.

- **Bit 3 – ADIE: Habilitador de interrupción**

  Habilita la interrupción por el fin de una conversión analógica a digital.

- **Bits 2 al 0 – ADPS[2:0]: Bits para seleccionar el factor de división del pre-escalador del ADC**

  El pre-escalador del ADC se mostró en la figura 5.6 y sus factores de división se describieron en la tabla 5.3.

En un ATMega16, además del modo de carrera libre, es posible iniciar automáticamente una conversión con un evento de otro recurso. El evento se define a través de los bits **ADTS[2:0]** (*ADC Auto Trigger Source*), estos bits son los más significativos del Registro I/O de Función Especial (**SFIOR**, *Special Function IO Register*), los bits de **SFIOR** son:

En la tabla 5.5 se muestra cómo seleccionar el modo de carrera libre y los eventos que automáticamente inician una conversión analógica a digital. Se observa que el modo de carrera libre corresponde con la combinación "000", por lo que si el registro **SFIOR** conserva su valor de reinicio (0x00), la activación del modo de carrera libre se realiza de la misma manera, tanto en un ATMega16, como en un ATMega8.

Tabla 5.5 Selección del modo de carrera libre o de la fuente de disparo de una conversión

ADTS2	ADTS1	ADTS0	Fuente de disparo
0	0	0	Modo de carrera libre
0	0	1	Comparador analógico
0	1	0	Interrupción externa 0
0	1	1	Coincidencia por comparación en el temporizador 0
1	0	0	Desbordamiento del temporizador 0
1	0	1	Coincidencia con el comparador B del temporizador 1
1	1	0	Desbordamiento del temporizador 1
1	1	1	Captura en el temporizador 1

5.1.6 Ejemplos de Uso del Convertidor Analógico a Digital

En esta sección se muestran 2 ejemplos del uso de un ADC, en el primer ejemplo las soluciones se muestran en ensamblador y lenguaje C. En el segundo ejemplo se asume que existe una biblioteca de funciones para el manejo de un display de cristal líquido (LCD, *liquid crystal display*) y por lo tanto, sólo se presenta la solución en lenguaje C.

Ejemplo 5.1 Desarrolle un sistema con base en un ATMega8, que mantenga la temperatura de una habitación en el rango entre 18 y 23 °C, aplicando la siguiente idea: Si la temperatura es mayor a 23 °C, el sistema debe activar un ventilador, si es menor a 18 °C, el sistema debe activar un calefactor. Para temperaturas entre 18 y 23 °C no se debe activar alguna carga.

La solución inicia con la definición del hardware, para detectar la temperatura se emplea un sensor LM35, el cual entrega 10 mV/°C. Puesto que el sistema trabaja con la temperatura ambiente, el voltaje proporcionado por el sensor es amplificado por un factor de 10, de manera que para temperaturas entre 0 y 50 °C, el sensor entrega voltajes entre 0 y 5 V. Con ello, el voltaje de alimentación del ATMega8 (5 V) puede emplearse como referencia para el ADC.

También se asume que el ventilador y el calefactor están acondicionados para activarse con una señal digital, en la figura 5.9 se muestra el hardware propuesto.

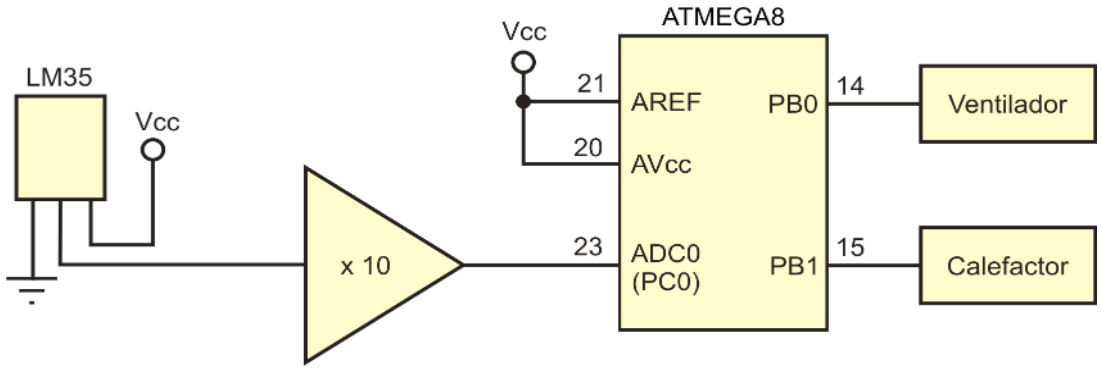


Figura 5.9 Hardware para el problema 5.1

En lo que refiere al software, puesto que el sistema está dedicado únicamente al control de temperatura, no es necesario un manejo de interrupciones. Dentro de un lazo infinito se inicia una conversión analógica a digital, se espera su culminación y el resultado es evaluado para determinar si se activa una salida.

Puesto que el ADC es de 10 bits, su resolución es de:

$$Resolución = \frac{V_{MAX}}{2^{10} - 1} = \frac{5\ V}{1023} = 4.8875\ mV$$

Con esta consideración, en la tabla 5.6 se tienen los valores digitales para las temperaturas de interés, el voltaje analógico a considerar es el de la salida del amplificador y el valor digital se ha redondeado para manipular números enteros.

Tabla 5.6 Valores digitales para las temperaturas de interés

Temperatura	Voltaje analógico (Van)	Valor digital (Van/Resolución)
18 °C	1.8 V	1.8 V/ 4.8875 mV = 368.28 ≈ 368
23 °C	2.3 V	2.3 V/ 4.8875 mV = 470.58 ≈ 471

Otro supuesto es que el ATMega8 trabaja a 1 MHz, para dividir esta frecuencia entre 8 y operar al ADC dentro del rango adecuado. El programa en lenguaje C es:

```
#define   inf   368          // Valor digital para 18 grados
#define   sup   471          // Valor digital para 23 grados

#include   <avr/io.h>

int   main(void) {
unsigned int   temp;

DDRB = 0xFF;                 // Puerto B como salida
PORTB = 0x00;                // Inicia con las salidas sin activar

ADMUX  = 0b00000000;         // Selecciona ADC0 y Vref en AREF
ADCSRA = 0b11000011;         // Habilita ADC, inicia conversión y divide entre 8
   while(1) {

     while( !(ADCSRA & 1 << ADIF ) );    // Espera fin de conversión
     ADCSRA |= 1 << ADIF;                // Limpia bandera
     temp = ADCW;                        // Obtiene la temperatura
     if( temp > sup )
        PORTB = 0x01;                    // Activa ventilador
     else if( temp < inf )
        PORTB = 0x02;                    // Activa calefactor
     else
        PORTB = 0x00;                    // Salidas sin activar

     ADCSRA |= 1 << ADSC;                // Inicia nueva conversión
   }
}
```

Para la versión en ensamblador, puesto que se comparan datos de 16 bits, primero se compara el byte menos significativo, si se genera un acarreo, éste debe ser considerado al comparar el byte más significativo, el código es el siguiente:

```
       .include      "m8def.inc"
       .EQU   inf = 368           ; Valor digital para 18 grados
       .EQU   sup  = 471          ; Valor digital para 23 grados

       CLR    R16
       OUT    ADMUX, R16          ; Selecciona ADC0 y Vref en AREF
       LDI    R16, 0b11000011
       OUT    ADCSRA, R16         ; Hab. ADC, inicia conversión y divide entre 8

       LDI    R16, 0xFF
       OUT    DDRB, R16           ; Puerto B como salida
       CLR    R16
       OUT    PORTB, R16          ; Inicia con las salidas sin activar

LOOP:  SBIS   ADCSRA, ADIF        ; Espera fin de conversión
       RJMP   LOOP
```

```
        SBI     ADCSRA, ADIF        ; Limpia bandera

; Se destinan R27 y R26 para el valor digital proporcionado por el ADC
        IN      R27, ADCH
        IN      R26, ADCL
                                    ; Comparación con el límite superior
        CPI     R26, LOW(sup)       ; Compara parte baja
        LDI     R16, HIGH(sup)
        CPC     R27, R16            ; Compara parte alta, considera acarreo
        BRLO    no_sup              ; brinca si temp < sup
        BREQ    no_sup              ; o si temp = sup
        LDI     R16, 0x01           ; Activa ventilador
        OUT     PORTB, R16          ; cuando temp > sup
        RJMP    fin_comp
no_sup:                             ; Comparación con el límite inferior
        CPI     R26, LOW(inf)       ; Compara parte baja
        LDI     R16, HIGH(inf)
        CPC     R27, R16            ; Compara parte alta, considera acarreo
        BRSH    no_inf              ; brinca si temp >= inf
        LDI     R16, 0x02           ; Activa calefactor
        OUT     PORTB, R16          ; cuando temp < inf
        RJMP    fin_comp
no_inf:
        LDI     R16, 0x00           ; salidas sin activar
        OUT     PORTB, R16          ; cuando temp < sup y temp > inf
fin_comp:
        SBI     ADCSRA, ADSC        ; Inicia nueva conversión
        RJMP    LOOP                ; Regresa al lazo infinito
```

El tiempo de ejecución del código puede mejorarse si se destinan algunos registros para el manejo de constantes y se asignan sus valores fuera del lazo infinito, con ello, se reduce el número de instrucciones a ejecutar en cada iteración.

Ejemplo 5.2 Realice un termómetro digital con un rango de 0 a 50 ^{0} C, con salida a un LCD y actualizando el valor de la temperatura cada medio segundo.

El LCD se conecta al puerto B de un ATMega8, como se muestra en la figura 5.10. Un LCD puede ser manipulado con una interfaz de 4 bits para que utilice sólo un puerto del microcontrolador. Para este ejemplo se asume que existe una biblioteca con las funciones necesarias para el manejo del LCD, la descripción de su funcionamiento es parte del capítulo 8.

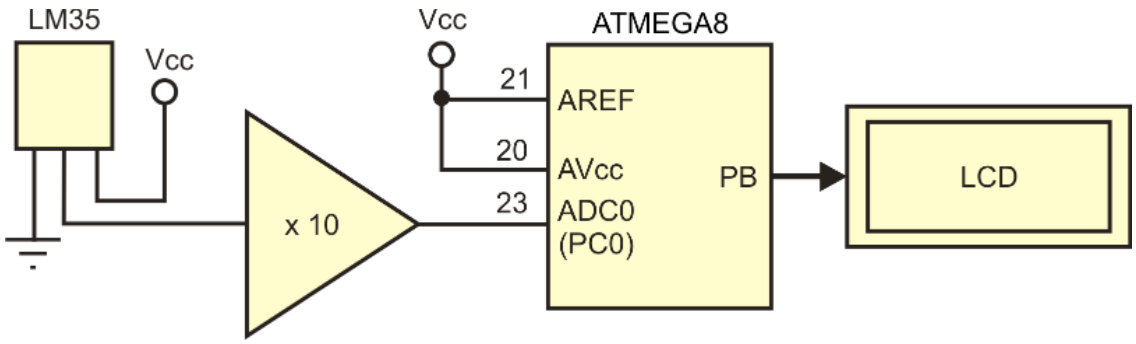


Figura 5.10 Hardware para el problema 5.2, referente al termómetro digital

Se emplea al temporizador 1 para obtener el intervalo de actualización de la temperatura. El temporizador genera un evento cada medio segundo y en su ISR inicia la conversión analógico a digital. Suponiendo el uso del oscilador interno de 1 MHz, el temporizador 1 debe interrumpir cada 500 000 ciclos, puesto que no es un valor que alcance en 16 bits se utiliza al pre-escalador con un factor de división de 8, dado que 500 000/8 = 62 500, este número si alcanza en 16 bits. Se utilizan eventos de comparación, configurando al modo CTC para limpieza automática del temporizador.

Al finalizar la conversión se produce otro evento, en cuya ISR se actualiza la temperatura en el LCD. El acondicionamiento del sensor es el mismo que en el ejercicio anterior.

El código en lenguaje C es:

```
#include  "LCD.h"                  // Biblioteca con las funciones para el LCD
#include <avr/io.h>
#include <avr/interrupt.h>

ISR (ADC_vect) {                   // ISR para el fin de conversión del ADC
float   temperatura;               // Variable con la temperatura
int     ent;                       // Variables auxiliares para convertir el valor
float   dec;                       // de la temperatura en una secuencia de
unsigned  char  d, u, f;           // caracteres

       temperatura = 0.0488758*ADCW;     // La temperatura en punto flotante
       ent = temperatura;                // Parte entera de la temperatura
       dec = temperatura - ent;          // Parte fraccionaria de la temperatura
       dec = dec * 10;                   // Para obtener sólo los decimales

// La información se convierte a caracteres
       d = ent/10;                 // Decenas de la parte entera de la temperatura
       u = ent - d*10;             // Unidades de la parte entera de la temperatura
       f = dec;                    // Sólo un dígito de la parte fraccionaria
// Despliegue de la información, byte por byte. Suma 0x30 para código ASCII
       LCD_cursor(0x08);                 // Ubica al cursor
       LCD_write_data(d + 0x30);         // Escribe decenas
       LCD_write_data(u + 0x30);         // Escribe unidades
```

```
      LCD_write_data('.');              // Escribe el punto decimal
      LCD_write_data(f + 0x30);         // Escribe decimales
}

ISR (TIMER1_COMPA_vect) {               // ISR del temporizador 1

      ADCSRA = ADCSRA | 0x40;           // Cada medio segundo inicia una
}                                       // Conversión Analógico a Digital

int main(void)                      // Programa Principal
{
      DDRB = 0xFF;                  // Puerto B como salida (para el LCD)
      LCD_reset();                  // Inicializa al LCD

      ADMUX = 0x00;                 // Entrada analógica en ADC0 y Vref en AREF
      ADCSRA = 0xCB;                // Habilita ADC, inicia conversión, divide por 8
                                    // y habilita interrupción por fin de conversión
      TIMSK = 0x10;                 // Configura al temporizador 1 para que
      OCR1A = 62499;                // interrumpa cada medio segundo, con eventos
      TCCR1A = 0x00;                // de comparación, modo CTC y un factor de
      TCCR1B = 0x0A;                // pre-escala de 8
      LCD_write_cad(" TEMP = ", 8);     // Cadena constante de 8 caracteres

      sei();                        // Habilitador global de interrupciones

      while( 1 ) {                  // En el lazo infinito permanece ocioso
            asm("nop");
   }
}
```

Al codificar en un lenguaje de alto nivel se simplifica el uso de datos en punto flotante.

5.2 Comparador Analógico

El comparador analógico es un recurso que indica la relación existente entre dos señales analógicas externas, es útil para aplicaciones en donde no precisa conocer el valor digital de una señal analógica, sino que es suficiente con determinar si ésta es mayor o menor que alguna referencia.

5.2.1 Organización del Comparador Analógico

En la figura 5.11 se muestra la organización del comparador, las entradas analógicas son tomadas de las terminales AIN0 y AIN1 (PD6 y PD7 en un ATMega8, y PB2 y PB3 en un ATMega16). Estas terminales se conectan a un amplificador operacional en lazo abierto, el cual se va a saturar cuando AIN0 sea mayor que AIN1, poniendo en alto al bit **ACO** (ACO, *Analog Comparator Output*).

Por medio de interruptores analógicos es posible remplazar la entrada AIN0 por una referencia interna cuyo valor típico es de 1.23 V, esto se consigue poniendo en alto al bit **ACBG** (*Analog Comparator Band Gap*). La entrada AIN1 puede remplazarse por la salida del multiplexor del ADC, de manera que es posible comparar más de una señal analógica con la misma referencia. Para ello se requiere que el ADC esté inhabilitado (**ADEN** = 0) y se habilite el uso del multiplexor por el comparador (**ACME** = 1).

Un cambio en el bit **ACO** puede generar una interrupción y además, producir un evento de captura en el temporizador 1, esto podría ser útil para determinar el tiempo durante el cual una señal superó a otra. La habilitación de la interrupción se realiza con el bit **ACIE** y la selección de la transición que genera la interrupción se realiza con los bits **ACIS[1:0]**, éstos y los demás bits para el manejo del comparador son parte del registro para el control y estado del comparador analógico (**ACSR**, *Analog Comparator Control and Status Register*).

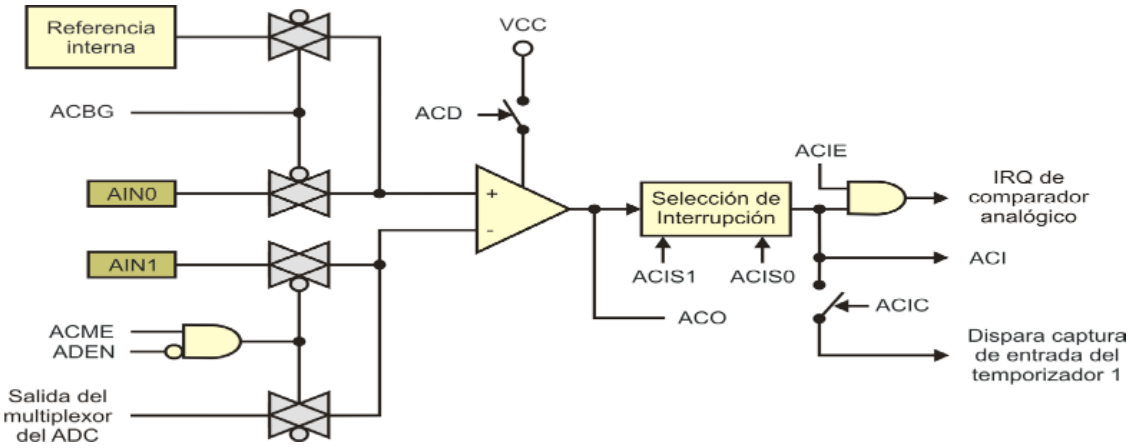


Figura 5.11 Organización del comparador analógico

5.2.2 Registros para el Manejo del AC

El registro **ACSR** es el más importante para el uso del comparador, los bits de este registro son:

	7	6	5	4	3	2	1	0	
0x08	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	**ACSR**

- **Bit 7 – ACD: Inhabilita al AC**

  El AC puede ser inhabilitado para minimizar el consumo de energía (con 0 el AC está activo). Si el AC está inactivo y se va a activar nuevamente, se sugiere inhabilitar su interrupción porque podría generar un evento.

- **Bit 6 – ACBG: Selecciona el voltaje interno (V_{BG}, *Band Gap*)**

  Al ponerlo en alto, la entrada en la terminal positiva proviene de un voltaje de referencia interno.

- **Bit 5 – ACO: Salida del comparador analógico**

  La salida del AC se sincroniza y se muestra en este bit, la sincronización toma 1 ó 2 ciclos de reloj.

- **Bit 4 – ACI: Bandera de interrupción**

  Su puesta en alto va a generar una interrupción, siempre que la interrupción esté habilitada (**ACIE** en alto). Su valor depende de las entradas del comparador y de los bits **ACIS[1:0]**.

- **Bit 3 – ACIE: Habilitador de interrupción**

  Habilita la interrupción por una transición en **ACO**, definida en los bits **ACIS[1:0]**.

- **Bit 2 – ACIC: Habilita la captura de entrada**

  Con la puesta en alto de este bit se conecta la salida del comparador con el hardware del temporizador 1, para disparar su función de captura de entrada. Se debe configurar el flanco requerido y la cancelación de ruido, además de habilitar la interrupción por captura de entrada, poniendo en alto al bit **TICIE1** del registro **TIMSK**.

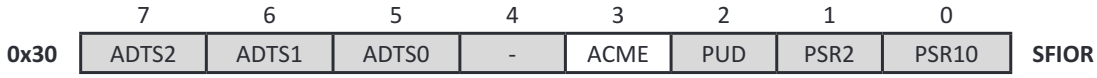
- **Bits 1 al 0 – ACIS[1:0]: Bits para seleccionar la transición para la interrupción**

  En estos bits se define la transición que produce la interrupción del comparador analógico, sus opciones se describen en la tabla 5.7.

Tabla 5.7 Selección de la transición que genera la interrupción del AC

ACIS1	ACIS0	Tipo de transición
0	0	Cualquier conmutación en ACO
0	1	Reservado
1	0	Flanco de bajada en ACO
1	1	Flanco de subida en ACO

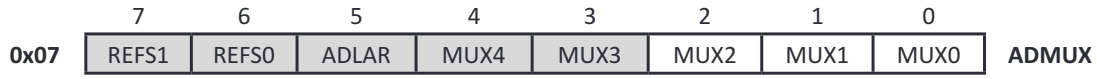
El registro de función especial **SFIOR** también se involucra con el comparador analógico porque en la posición 3 de este registro se encuentra al bit **ACME** (*Analog Comparator Multiplexor Enable*), este bit habilita el uso del multiplexor del ADC para proporcionar la entrada negativa al comparador.



De acuerdo con el hardware de la figura 5.11, el empleo del multiplexor también requiere que el bit **ADEN** (*ADC enable*) tenga 0, inhabilitando así al **ADC**. El bit **ADEN** es parte del registro **ADCSRA**:

Si el ADC se habilita, aunque el bit **ACME** tenga 1, la entrada negativa del comparador va a ser tomada de la terminal AIN1.

Cuando se conecta la salida del multiplexor con la entrada negativa del AC (**ACME** = 1 y **ADEN** = 0), el canal se selecciona con los bits **MUX[2:0]**, los cuales corresponden con los bits menos significativos del registro **ADMUX**, teniendo la posibilidad de emplear uno de 8 canales (6 en un ATMega8 con encapsulado PDIP).



Los bits **MUX[4:3]** no están involucrados en la selección del canal porque para la entrada negativa del AC no es aplicable el uso de valores constantes o entradas diferenciales, si se tratase de un ATMega16.

5.2.3 Ejemplos de uso del Comparador Analógico

El comparador es un recurso muy simple en su uso, por ello, en esta sección se muestran 2 ejemplos con la solución codificada únicamente en lenguaje C. Las versiones en ensamblador pueden desarrollarse con la misma estructura.

Ejemplo 5.3 Realice un sistema que encienda un ventilador cuando la temperatura esté por encima de 20 °C, en caso contrario que el ventilador permanezca apagado.

En la entrada AIN0 del comparador se conecta un sensor de temperatura acondicionado para que entregue un voltaje de 0 a 5 V ante un rango de temperatura de 0 a 50 °C y en la entrada AIN1 un voltaje constante de 2.0 V, con ello, el estado del ventilador corresponde con la salida del comparador, reflejada en el bit **ACO**, en la figura 5.12 se muestra el acondicionamiento del hardware.

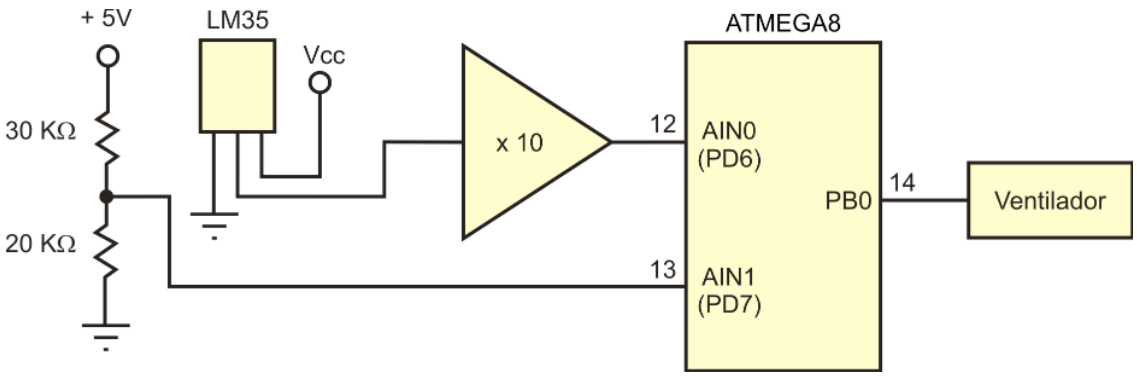


Figura 5.12 Hardware para mostrar el uso del comparador analógico

Con respecto al software, se emplea la interrupción por cambios en el AC, de manera que en la ISR únicamente se actualice la salida del ventilador de acuerdo con el bit **ACO**. El programa en lenguaje C es:

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

ISR (ANA_COMP_vect)                          // ISR para los cambios en el comparador
{
       if( ACSR & 1 << ACO )                 // Si la temperatura es alta
              PORTB = PORTB | 0x01;          // enciende el ventilado
       else
              PORTB = PORTB & 0xFE;          // Sino, lo apaga
}

int main(void) {                             // Programa Principal

       DDRB = 0xFF;          // Puerto B como salida
       PORTB = 0x00;         // Inicia con el ventilador apagado
       ACSR = 0x08;          // Configura al CA, el bit ACO inicia con 0,
                             // si la temperatura es alta produce la interrupción
       asm("sei");           // Habilitador global de interrupciones

       while (1)             // Lazo infinito, permanece ocioso
              asm("nop");
}
```

Ejemplo 5.4 Diseñe el control de un móvil seguidor de línea, compuesto por 2 llantas traseras manejadas por motores independientes y una llanta de libre movimiento al frente. El móvil debe seguir una línea blanca en una superficie con fondo negro, utilizando el comparador analógico y 2 canales del ADC de un ATMega8. Suponga que los sensores se han acondicionado de manera que, cuando se encuentran en una zona oscura presentan un voltaje mayor a 2.5 V y en una zona clara su voltaje está por debajo de ese nivel. Ubique los sensores al frente del móvil, uno en cada extremo, dejando la línea blanca al centro.

El estado de los sensores define el encendido de los motores. Si ambos detectan una zona oscura, los 2 motores deben estar encendidos (móvil en línea recta). Cuando un sensor detecta una zona clara (debido a una curva), se debe apagar al motor del mismo lado para provocar un giro en el móvil. Se espera que en ningún momento los 2 sensores detecten una zona clara, sin embargo, si eso sucede, también se deben encender los 2 motores, buscando que el móvil abandone esta situación no esperada.

En la figura 5.13 se hace un bosquejo del hardware, los sensores se conectan en 2 de los canales del ADC, para que alternadamente remplacen la entrada AIN1 del AC. En la entrada AIN0 se conecta una referencia constante de 2.5 V.

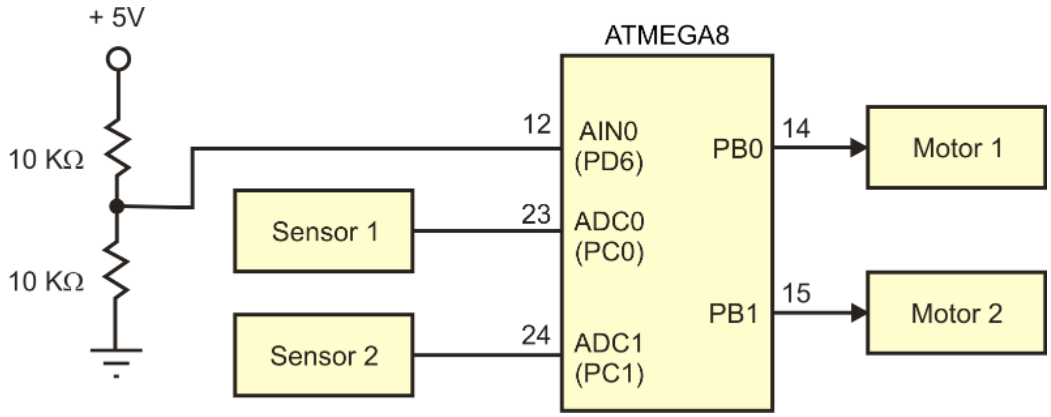


Figura 5.13 Hardware para el control de un móvil, seguidor de línea

Con respecto al software, en un lazo infinito se debe obtener el estado de los sensores para definir las salidas de los motores. No se utilizan interrupciones. El código en lenguaje C es:

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

int main(void)       {              // Programa Principal
unsigned  char       sens;          // Para el estado de los sensores

       DDRB = 0xFF;                 // Puerto B como salida
       PORTB = 0x03;                // Inicia con los 2 motores encendidos
                                   // El móvil inicia en una línea recta
       ADCSRA = 0x00;               // Asegura la desactivación del ADC
       SFIOR = 0x08;                // Conecta al multiplexor con el AC

       while(1) {
              sens = 0x00;
              ACSR = 0x80;                  // Desactiva al AC
              ADMUX = 0x00;                 // Selecciona el canal 0
              ACSR = 0x00;                  // Activa al AC
              asm("nop");                   // Espera el resultado del AC
              if(!(ACSR & 1 << ACO))        // Si ( Sensor 1 en zona oscura )
                     sens |= 0x01;          // motor 1 encendido

           ACSR = 0x80;                  // Desactiva al AC
```

```
            ADMUX = 0x01;              // Selecciona el canal 1
            ACSR = 0x00;               // Activa al AC
            asm("nop");                // Espera el resultado del AC
            if(!(ACSR & 1 << ACO))     // Si ( Sensor 2 en zona oscura )
                    sens |= 0x02;      // motor 2 encendido
                                          // Si sólo un sensor está en
            if( sens == 0x01 || sens == 0x02 )  //  la zona oscura, enciende
                    PORTB = sens;              // el motor correspondiente
            else
                    PORTB = 0b00000011;    // Sino enciende los 2 motores
               }
}
```

En el código se muestra que es necesario desactivar al comparador analógico antes de definir el canal de entrada. Una vez activado el comparador, es conveniente esperar uno o dos ciclos de reloj, para que la salida del CA sea estable.

5.3 Ejercicios

Los siguientes ejercicios son para utilizar el ADC o el AC en combinación con otros recursos, pueden implementarse en un ATMega8 o en un ATMega16, programando con ensamblador o lenguaje C.

1. Genere una señal PWM de 10 bits en modo no invertido, en donde el ciclo de trabajo sea proporcional al voltaje proporcionado por un potenciómetro conectado al canal 0 del ADC, como se muestra en la figura 5.14.

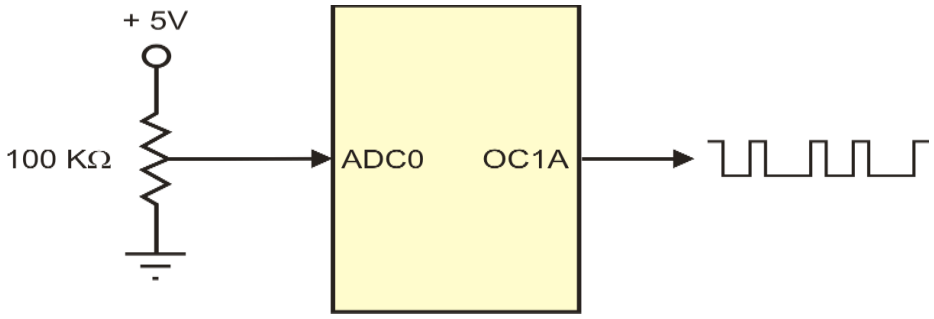


Figura 5.14 Generación de una señal PWM con un ciclo útil proporcional al voltaje en el ADC

2. Un *fotoresistor* iluminado con luz solar presenta una resistencia con un valor de 300 ohms, sin iluminación la resistencia está por encima de 20 Kohms. Desarrolle un circuito que encienda una lámpara si la resistencia en el sensor es mayor a 15 Kohms, y la apague cuando la resistencia esté por debajo de 2 Kohms.

   Se introduce una curva de histéresis para evitar oscilaciones. El *fotoresistor* debe acondicionarse para que entregue un voltaje proporcional a la resistencia. El problema puede resolverse de 2 formas diferentes:

a. El voltaje debido al *fotoresistor* se introduce al microcontrolador a través de su ADC y vía software se realizan las comparaciones.

b. Se utiliza al comparador, en una entrada se introduce el voltaje debido al sensor y en la otra se alternan los voltajes producidos con resistores de 2 y 15 Kohms.

3. Modifique el ejemplo 5.4 manejando los motores del móvil con señales PWM. En una línea recta, ambos motores deber tener un ciclo útil del 100 %. En una curva, el motor del sensor que detecte la zona clara debe trabajar con un ciclo útil del 20 % y el otro al 80 %, con la finalidad de que las curvas sean tomadas de una manera suave, para evitar un zigzagueo en el móvil.

4. Con respecto a la figura 5.15, las entradas Vin, Vref1 y Vref2 son señales analógicas y siempre ocurre que Vref1 < Vref2. De acuerdo con los resultados que proporcione el comparador analógico, utilizando el temporizador 1, genere una señal PWM a una frecuencia aproximada de 100 Hz, con un ciclo de trabajo de:

   - 10 %: si V_{in} es menor a V_{ref1}
   - 50 %: si V_{in} está entre V_{ref1} y V_{ref2}
   - 90 %: si V_{in} es mayor a V_{ref2}

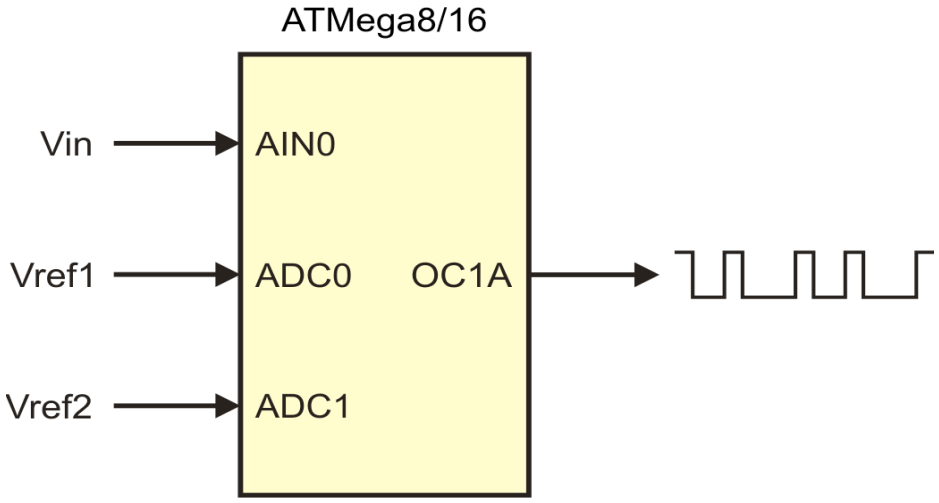


Figura 5.15 Generación de una señal PWM con un ciclo útil dependiente del AC

6. Interfaces para una Comunicación Serial

Los microcontroladores ATMega8 y ATMega16 incluyen 3 recursos para comunicarse de manera serial con dispositivos externos, empleando interfaces estándares. Estos recursos son: Un Transmisor-Receptor Serial Universal Síncrono y Asíncrono (USART, *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*), una Interfaz para Periféricos Seriales (SPI, *Serial Peripheral Interface*) y una Interfaz Serial de Dos Hilos (TWI, *Two-Wire Serial Interface*). Las 3 interfaces son descritas en el presente capítulo.

6.1 Comunicación Serial a través de la USART

La USART incluye recursos independientes para transmisión y recepción, esto posibilita una operación *full-duplex*, es decir, es posible enviar y recibir datos en forma simultánea. La terminal destinada para la transmisión de datos es TXD y para la recepción es RXD. Estas terminales corresponden con PD1 (TXD) y PD0 (RXD), tanto en el ATMega8 como en el ATMega16.

La comunicación puede ser síncrona o asíncrona, la diferencia entre estos modos se ilustra en la figura 6.1. En una comunicación síncrona, el emisor envía la señal de reloj, además de los datos, de esta forma el receptor va obteniendo cada bit en un flanco de subida o bajada de la señal de reloj, según se haya configurado (la terminal dedicada para el envío o recepción de la señal de reloj es XCK). En una comunicación asíncrona el emisor únicamente envía los datos, no se envía señal de reloj.

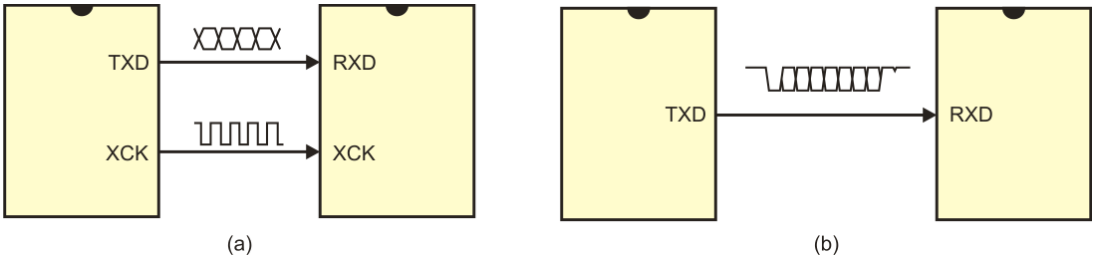


Figura 6.1 Comunicación serial (a) síncrona y (b) asíncrona

Por lo tanto, antes de cualquier transferencia de datos, para el transmisor y el receptor, se deben definir los siguientes parámetros:

- **Velocidad de transmisión (*Baudrate*):** Se refiere a la cantidad de bits que se transmiten en un segundo, se mide en *bits/segundo*, usualmente referido como *bauds* o simplemente *bps*.
- **Número de bits de datos**: Cada dato se integra por un número pre-definido de bits, pueden ser 5, 6, 7, 8 ó hasta 9 bits.

- **Bit de paridad:** Es un bit de seguridad que acompaña a cada dato, se ajusta automáticamente en alto o bajo para complementar un número par de 1's (paridad par) o un número impar de 1's (paridad impar). El emisor envía al bit de paridad después de enviar los bits de datos, el receptor lo calcula a partir de los datos recibidos y compara el bit de paridad generado con el bit de paridad recibido, una diferencia entre ellos indica que ocurrió un error en la transmisión. Por ello, es importante definir el uso del bit de paridad y configurar si va a ser par o impar.

- **Número de bits de paro:** Los bits de paro sirven para separar 2 datos consecutivos. En este caso, se debe definir si se utiliza 1 ó 2 bits de paro.

Una trama serial se compone de:

- 1 bit de inicio (siempre es un 0 lógico).
- 5, 6, 7, 8 o 9 bits de datos, iniciando con el menos significativo.
- Bit de paridad, par o impar, si se configuró su uso.
- 1 o 2 bits de paro (siempre son un 1 lógico).

En la figura 6.2 se ilustra una trama serial, considerando 8 bits de datos, paridad par y 1 bit de paro. Se observa que la línea de comunicación mantiene un nivel lógico alto mientras no se inicie con la transmisión de un dato.

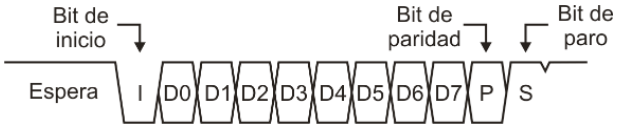


Figura 6.2 Trama serial de datos

6.1.1 Organización de la USART

La USART se compone de 3 bloques principales, éstos se observan en la figura 6.3. También pueden verse los registros **UCSRA**, **UCSRB** y **UCSRC**, con estos registros se realiza la configuración y se conoce el estado de la USART (**UCSR**, *USART Configuration and State Register*).

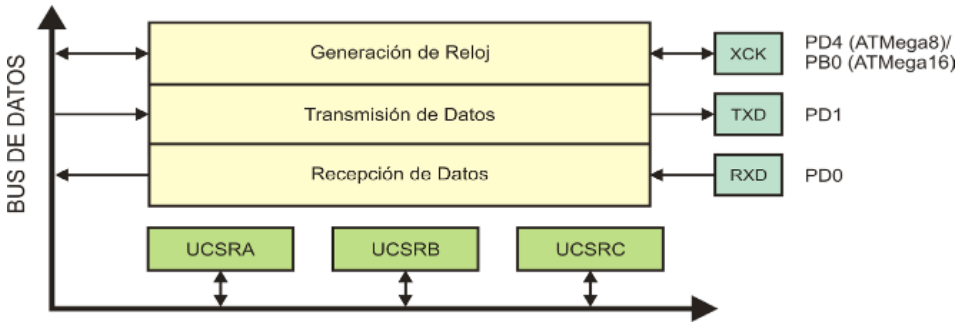


Figura 6.3 Organización de la USART

Los bloques de transmisión y recepción son independientes entre sí. El bloque generador de reloj proporciona las señales de sincronización a los otros 2 bloques.

6.1.1.1 Generación de Reloj y Modos de Operación

El bloque para la generación del reloj proporciona 2 señales de salida, como se muestra en la figura 6.4. La señal CLK1 es el reloj utilizado para transmisión/recepción asíncrona o bien para transmisión síncrona. La señal CLK2 es el reloj utilizado para recepción síncrona.

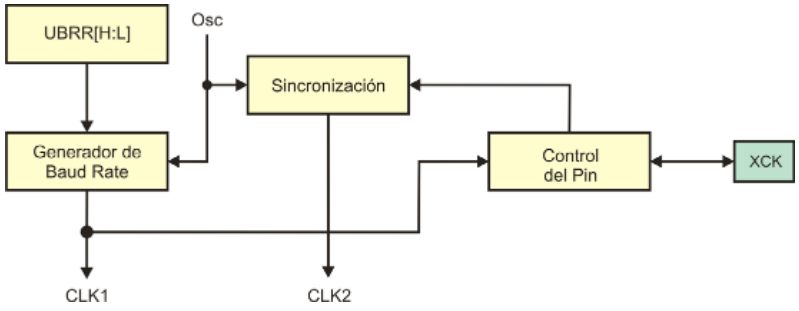


Figura 6.4 Bloque para la generación del reloj de la USART

El registro **UBRR** es la base para la generación de la señal CLK1 (**UBRR**, *USART Baud Rate Register*). Se compone de 2 registros de 8 bits, aunque sólo 12 de los 16 bits disponibles son utilizados.

El Generador de *Baud Rate* incluye un contador descendente cuyo valor máximo es tomado del registro UBRR, el contador recarga su valor máximo cuando llega a 0 o cuando se escribe en **UBRR[L]**. Una señal interna es conmutada cada vez que el contador alcanza un 0, por ello, la frecuencia base para CLK1 es la frecuencia del oscilador dividida entre **UBRR** + 1.

Dependiendo del modo de operación de la USART, esta señal interna es dividida entre 2, 8 ó 16. La USART soporta 4 modos de operación:

- Normal asíncrono: En el cual, la frecuencia de CLK1 es la frecuencia base entre 16.
- Asíncrono a doble velocidad: La frecuencia de CLK1 es la frecuencia base entre 8, el doble del modo normal.
- Síncrono como maestro: La frecuencia de CLK1 es la frecuencia base entre 2, se pueden alcanzar velocidades más altas al transmitir la señal de reloj.
- Síncrono como esclavo: En este caso, el microcontrolador está funcionando como un receptor síncrono, por lo que también recibe la señal de reloj en XCK. La señal externa debe sincronizarse con el oscilador interno. También se puede configurar el flanco en el que se obtienen los datos. La señal CLK2 queda disponible como el reloj para recepción síncrona.

La selección del modo de operación se realiza en los registros de configuración y estado de la USART. Específicamente, el bit **UMSEL** (bit 6 del registro **UCSRC**) se utiliza para seleccionar entre una operación síncrona (**UMSEL** = '1') y asíncrona (**UMSEL** = '0'). El bit **U2X** (bit 1 del registro **UCSRA**) sirve para seleccionar entre el modo normal asíncrono (**U2X** = '0') y el modo asíncrono a doble velocidad (**U2X** = '1').

En la tabla 6.1 se muestran las relaciones matemáticas para obtener la razón de transmisión (*baud rate*) a partir del valor del registro UBRR, no obstante, en la práctica, primero se determina a qué velocidad se va a transmitir, para después calcular el valor de UBRR. Estas relaciones también están en la tabla 6.1.

Tabla 6.1 Cálculo del Baud Rate

Modo	Baud Rate	Valor de UBRR
Normal asíncrono	$Baud\_Rate = \frac{fosc}{16(UBRR+1)}$	$UBRR = \frac{fosc}{16 \times Baud\_Rate} - 1$
Asíncrono a doble velocidad	$Baud\_Rate = \frac{fosc}{8(UBRR+1)}$	$UBRR = \frac{fosc}{8 \times Baud\_Rate} - 1$
Síncrono maestro	$Baud\_Rate = \frac{fosc}{2(UBRR+1)}$	$UBRR = \frac{fosc}{2 \times Baud\_Rate} - 1$

Debe tomarse en cuenta que al calcular el valor de **UBRR** se obtiene un número real, por ello, al redondear a entero se produce un error en el *Baud Rate*. El porcentaje de error se puede cuantificar con la expresión:

$$Error\ [\%] = \left(\frac{Baud\_Rate\_Real}{Baud\_Rate\_esperado} - 1\right) \times 100\ \%$$

En el ejemplo 6.1 se muestra cómo estimar el error en la razón de transmisión, un error en el rango de ± 5 % no afecta la comunicación.

Ejemplo 6.1 Se planea una transmisión asíncrona a 9600 bps con un oscilador de 1 MHz. Calcular el valor del registro **UBRR** y obtener el porcentaje de error, para una transmisión asíncrona normal y a doble velocidad.

En el modo normal asíncrono se tiene que:

$$UBRR = \frac{fosc}{16 \times Baud\_Rate} - 1 = \frac{1\ MHz}{16 \times 9600} - 1 = 5.51$$

Redondeando **UBRR** a 5, se calcula el *baud rate*:

$$Baud\_Rate = \frac{fosc}{16(UBRR+1)} = \frac{1\ MHz}{16\ (5+1)} = 10416.66\ bps$$

El porcentaje de error generado es de:

$$Error\ [\%] = \left(\frac{Baud\_Rate\_Real}{Baud\_Rate\_esperado} - 1\right) \times 100\ \% = \left(\frac{10416.66}{9600} - 1\right) \times 100\ \% = 8.5\ \%$$

Para el modo asíncrono a doble velocidad se tienen los siguientes resultados:

$$UBRR = \frac{1\ MHz}{8 \times 9600} - 1 = 12.02$$

Redondeando **UBRR** a 12, se tiene que:

$$Baud\_Rate = \frac{1\ MHz}{8\ (12 + 1)} = 9615.38\ bps$$

El porcentaje de error generado es de:

$$Error\ [\%] = \left(\frac{9615.38}{9600} - 1\right) \times 100\ \% = 0.16\ \%$$

El error se reduce drásticamente al usar el modo a doble velocidad.

En la tabla 6.2 se muestran algunos valores a emplear en el registro **UBRR** para razones de transmisión típicas, con diferentes osciladores y sus correspondientes porcentajes de error.

Tabla 6.2 Razones de transmisión típicas

Baud Rate (bps)	fosc = 1.0 MHz				fosc = 1.8432 MHz				fosc = 2.0 MHz			
	Normal		Doble Vel.		Normal		Doble Vel.		Normal		Doble Vel.	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
1200	51	0.2 %	103	0.2 %	94	0.0 %	191	0.0 %	103	0.2 %	207	0.2 %
2400	25	0.2 %	51	0.2 %	47	0.0 %	94	0.0 %	51	0.2 %	103	0.2 %
4800	12	0.2 %	25	0.2 %	23	0.0 %	47	0.0 %	25	0.2 %	51	0.2 %
9600	5	8.5 %	12	0.2 %	11	0.0 %	23	0.0 %	12	0.2 %	25	0.2 %
19200	2	8.5 %	5	8.5 %	5	0.0 %	11	0.0 %	5	8.5 %	12	0.2 %
115200	-	-	0	8.5 %	0	0.0 %	1	0.0 %	0	8.5 %	1	8.5 %

En el modo síncrono esclavo la USART recibe la señal de reloj en la terminal XCK, esta señal es sincronizada con el oscilador interno, requiriendo algunos ciclos de reloj para ello. Por esta razón, la frecuencia de la señal de reloj externa debe estar acotada por:

$$f_{XCK} = \frac{f_{osc}}{4}$$

En operaciones síncronas se debe definir el flanco de reloj en el cual son ajustados los datos (cuando se está transmitiendo) o en el que son muestreados (cuando se está recibiendo), esto se determina con el bit **UCPOL** (bit 0 del registro **UCSRC**). En la tabla 6.3 se muestran los flancos de activación (polaridad), dependiendo del valor del bit **UCPOL** (*Clock Polarity*).

Tabla 6.3 Polaridad en la transmisión/recepción síncrona

UCPOL	Ajuste TXD	Muestreo RXD
0	Flanco de subida	Flanco de bajada
1	Flanco de bajada	Flanco de subida

6.1.1.2 Transmisión de Datos

La figura 6.5 muestra la organización del bloque para la transmisión, donde se observa que los datos a ser transmitidos deben ser escritos en el Registro de Datos de la USART (**UDR**, *USART Data Register*). Con **UDR** realmente se hace referencia a dos registros ubicados en la misma dirección, un registro sólo de escritura para transmitir datos y un registro sólo de lectura para recibirlos.

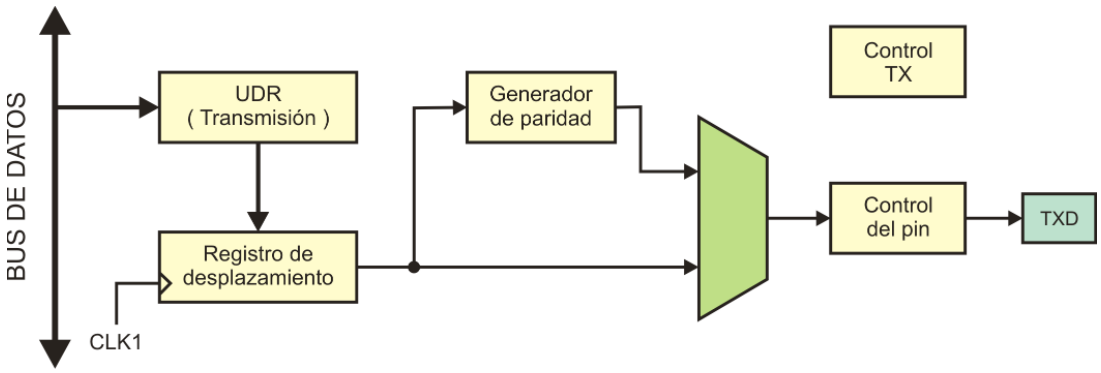


Figura 6.5 Bloque para la transmisión de datos

El bloque de transmisión de datos requiere de una habilitación, la cual se consigue poniendo en alto al bit **TXE** (bit 3 del registro **UCSRB**). La transmisión de un dato inicia cuando éste se escribe en el registro **UDR**, previamente se deben configurar los parámetros: velocidad, número de bits por dato, bit de paridad y número de bits de paro. Un registro de desplazamiento se encarga de la conversión de paralelo a serie.

El fin de la transmisión de un dato se indica con la puesta en alto de la bandera **TXC** (bit 6 del registro **UCSRA**), esta bandera puede sondearse por software o se puede configurar al recurso para que genere una interrupción. Si se utiliza sondeo, la bandera se limpia al escribirle un 1 lógico. No es posible escribir en el registro **UDR**

mientras hay una transmisión en proceso, por ello, además del uso de la bandera **TXC** es posible usar la bandera **UDRE** (bit 5 del registro **UCSRA**. **UDRE,** *USART Data Register Empty*). La bandera **UDRE** está en alto cuando el buffer transmisor (registro de desplazamiento) está vacío, por lo tanto, indica que es posible iniciar con una nueva transmisión. También puede sondearse por software o configurar el recurso para que genere una interrupción. El estado de la bandera **UDRE** cambia automáticamente, dependiendo de la actividad que ocurra en el registro **UDR**.

En la práctica únicamente se emplea una de las 2 banderas, **TXC** o **UDRE**. Usar ambas puede complicar la estructura de un programa, porque indican eventos similares cuando se transmite una sucesión de datos. Al finalizar con la transmisión de un dato, el buffer transmisor queda vacío.

El bit de paridad se genera por hardware conforme los datos se van transmitiendo. Éste queda disponible para ser transmitido después de los bits del dato.

6.1.1.3 Recepción de Datos

En la figura 6.6 se muestra la organización del bloque para la recepción de datos. También contiene un Registro de Datos de la USART (**UDR**), aunque en este caso es un registro sólo de lectura empleado para recibir datos seriales por medio de la USART. La recepción serial se basa en un registro de desplazamiento que realiza la conversión de serie a paralelo. El receptor debe habilitarse para que en cualquier momento pueda recibir un dato, la habilitación se realiza con la puesta en alto del bit **RXE** (bit 4 del registro **UCSRB**).

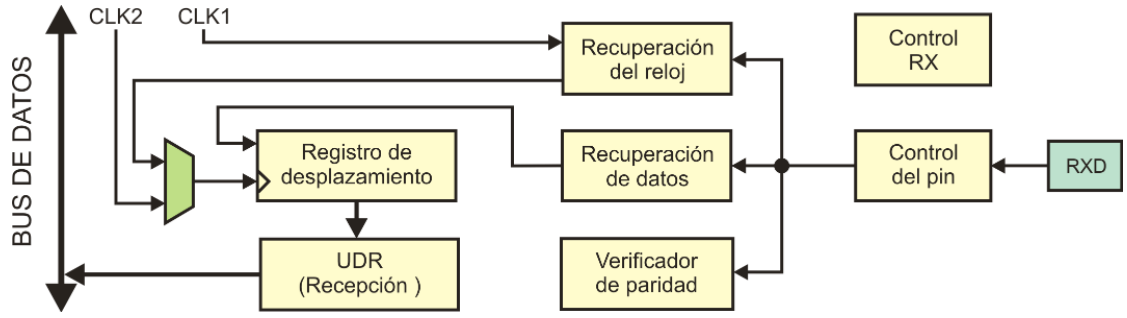


Figura 6.6 Bloque para la recepción de datos

El dato recibido es ubicado en el registro **UDR**, la indicación de que hay un dato disponible se realiza con la puesta en alto de la bandera **RXC** (Bit 7 del registro **UCSRA**). Esta bandera puede sondearse por software o configurar al recurso para que genere una interrupción. La lectura del registro **UDR** limpia a la bandera y libera al registro, quedando disponible para recibir un nuevo dato.

El cálculo del bit de paridad y su validación, comparándola con el valor del bit de paridad recibido, se realizan por hardware. Si existe un error, éste se indica con la puesta en alto de la bandera **PE** (bit 2 del registro **UCSRA**. **PE**, *Parity Error*).

El hardware es capaz de indicar la ocurrencia de otros 2 errores durante la recepción, con la puesta en alto de banderas. Un error de marco (*frame error*) se debe a la existencia de un bit de paro con un nivel bajo y se indica con la bandera **FE** (bit 4 del registro **UCSRA**). El otro error es debido a un exceso de datos por recepción, ocurre cuando el registro **UDR** tiene un dato listo para su lectura, el registro de desplazamiento contiene otro dato y hay un nuevo bit de inicio, este error se indica con la bandera **DOR** (bit 3 del registro **UCSRA**. **DOR**, *Data Over Run*).

6.1.2 Transmisión y Recepción de Datos de 9 Bits

Las transmisiones y recepciones se realizan por medio del registro **UDR**, el cual físicamente está compuesto por 2 registros de 8 bits, uno sólo de escritura para transmisión y otro sólo de lectura para recepción.

Sin embargo, la USART permite el envío y recepción de datos de 9 bits, para ello se emplean 2 bits del registro **UCSRB**. El bit **TXB8** (bit 0 del registro **UCSRB**) es el 9º en una transmisión de 9 bits y el bit **RXB8** (bit 1 del registro **UCSRB**) es el 9º durante una recepción de 9 bits. Para una transmisión primero debe definirse el valor de **TXB8**, porque la transmisión inicia cuando se escriba en el registro **UDR**. Y en una recepción primero debe leerse el valor de **RXB8** y luego al registro **UDR**. Tanto **TXB8** como **RXB8** corresponden con el bit más significativo en datos de 9 bits.

6.1.3 Comunicación entre Múltiples Microcontroladores

La USART de los AVR permite una comunicación entre más de 2 microcontroladores, bajo un esquema maestro-esclavos, pudiendo conectar hasta 256 esclavos. En la figura 6.7 se ilustra esta organización, los esclavos deben tener una dirección diferente, entre 0 y 255. Estas direcciones se pueden manejar como constantes en memoria *flash* o configurarse con interruptores en uno de los puertos del microcontrolador.

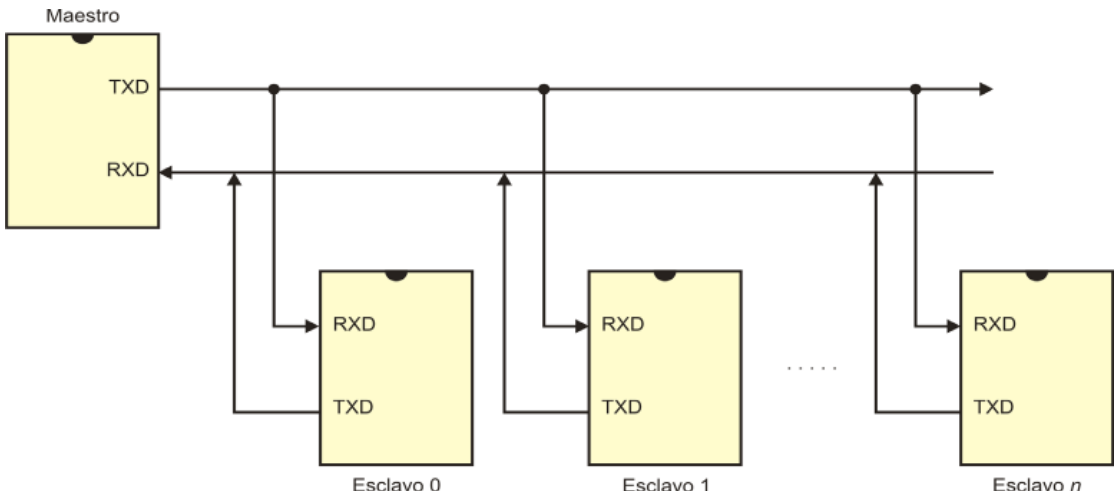


Figura 6.7 Organización maestro-esclavos para comunicar a múltiples microcontroladores

La comunicación utiliza un formato de 9 bits, donde el 9º bit (transmitido en **TXB8** o recibido en **RXB8**) sirve para distinguir entre dos tipos de información, con 0 se refiere a una trama de datos y con 1 a una trama de dirección.

La comunicación entre múltiples MCUs requiere el empleo del bit **MPCM** (bit 0 del registro **UCSRA**. **MPCM**, *Multiprocessor Comunication Mode*). Este bit es utilizado por los esclavos, si está en alto, únicamente pueden recibir tramas de dirección. Su puesta en bajo les permite recibir tramas de datos.

La comunicación se realiza de la siguiente manera:

1. Los esclavos habilitan el modo de comunicación entre multiprocesadores, con **MPCM** = 1. De manera que únicamente pueden recibir tramas de dirección (9º bit en 1).

2. El maestro envía la dirección del esclavo con el que va a interactuar, esta dirección es recibida por todos los esclavos.

3. Cada esclavo lee su registro **UDR** y lo compara con su dirección para determinar si ha sido seleccionado. El esclavo seleccionado limpia su bit **MPCM** en el registro **UCSRA**, quedando disponible para recibir datos.

4. El maestro y el esclavo seleccionado realizan el intercambio de datos (9º bit en 0), el cual pasa desapercibido por los otros esclavos, porque aún tienen su bit **MPCM** en alto.

5. Cuando el diálogo concluye, el esclavo seleccionado debe poner en alto su bit **MPCM**, quedando como los demás esclavos, en espera de que el maestro solicite su atención.

6.1.4 Registros para el Manejo de la USART

El registro **UDR** sirve para el manejo de datos en la USART. Físicamente se compone de 2 registros de 8 bits, uno sólo de lectura para la recepción de datos y otro sólo de escritura para transmisión, aunque ambos emplean la misma dirección.

	7	6	5	4	3	2	1	0	
0x0C	Byte recibido								**UDR** (Lectura)
	Byte a transmitir								**UDR** (Escritura)

Para definir la razón de transmisión (*baud rate*) se utiliza al registro **UBRR**, éste es un registro de 12 bits, por lo tanto, se integra por 2 registros de 8 bits: **UBRRL** para la parte baja y **UBRRH** para la parte alta:

	7	6	5	4	3	2	1	0	
0x20	URSEL	-	-	-	UBRR11	UBRR10	UBRR9	UBRR8	**UBRRH**
0x09	UBRR7	UBRR6	UBRR5	UBRR4	UBRR3	UBRR2	UBRR1	UBRR0	**UBRRL**

El registro **UBRRH** comparte su dirección (0x20) con el registro **UCSRC**, el cual es uno de los registros de control. Para ello, debe utilizarse al bit **URSEL** como un selector para escrituras. Con **URSEL** = 0 se escribe en **UBRRH** y con **URSEL** = 1 se escribe en **UCSRC**.

Para las lecturas, puesto que **UBRRH** y **UCSRC** hacen referencia a la misma dirección, una lectura aislada proporciona el valor de **UBRRH**. Para obtener el valor de **UCSRC** deben realizarse 2 lecturas consecutivas.

Los registros de control y estado de la USART son 3: **UCSRA**, **UCSRB** y **UCSRC**.

En el registro **UCSRA**, los 6 bits más significativos corresponden con las banderas de estado de la USART. Los bits de **UCSRA** son:

	7	6	5	4	3	2	1	0	
0x0B	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	**UCSRA**

- **Bit 7 – RXC: Bandera de recepción completa**

  Indica que hay un dato disponible por recepción. Puede generar una interrupción. Se limpia al leer al registro **UDR**.

- **Bit 6 – TXC: Bandera de transmisión completa**

  Indica que se ha concluido con la transmisión de un dato y por lo tanto, el buffer de salida está vacío. Puede generar una interrupción, con su atención se limpia automáticamente a la bandera. También puede limpiarse con la escritura de un 1.

- **Bit 5 – UDRE: Bandera de UDR vacío**

  Indica que el registro **UDR** de transmisión está vacío y por lo tanto, es posible transmitir un dato. También puede generar una interrupción. Su estado se ajusta automáticamente, de acuerdo con la actividad en el registro **UDR**.

- **Bit 4 – FE: Bandera de error de marco**

  Esta bandera se pone en alto cuando el bit de paro en el buffer receptor es 0. Se mantiene con ese valor hasta que se lee al registro **UDR**.

- **Bit 3 – DOR: Bandera de error por exceso de datos**

  Un exceso de datos significa que el registro receptor (**UDR**) tiene un dato disponible, el registro de desplazamiento contiene otro dato y se tiene un nuevo bit de inicio. Cuando la bandera es puesta en alto, mantiene su valor hasta que el registro **UDR** es leído.

- **Bit 2 – PE: Bandera de error por paridad**

  Se pone en alto cuando el bit de paridad generado no coincide con el bit de paridad recibido. Se mantiene con ese valor hasta que el registro **UDR** es leído.

- **Bit 1 – U2X: Dobla la velocidad de transmisión**

  En operaciones asíncronas, este bit debe ponerse en alto para cambiar del modo normal asíncrono, al modo asíncrono a doble velocidad.

- **Bit 0 – MPCM: Modo de comunicación entre multiprocesadores**

  Habilita un modo de comunicación entre multiprocesadores, bajo un esquema maestro-esclavos. El bit **MPCM** es empleado por los esclavos para que puedan recibir tramas de dirección o tramas de datos.

Los 3 bits más significativos de **UCSRB** son habilitadores para generar interrupciones, coinciden en orden con las banderas de **UCSRA**. Los bits del registro **UCSRB** son:

	7	6	5	4	3	2	1	0	
0x0A	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	**UCSRB**

- **Bit 7 – RXCIE: Habilitador de interrupción por recepción completa**

  Si es puesto en alto, y el habilitador global de interrupciones también, se produce una interrupción cuando se reciba un dato (**RXC** en alto).

- **Bit 6 – TXCIE: Habilitador de interrupción por transmisión completa**

  Si es puesto en alto, y el habilitador global de interrupciones también, se produce una interrupción cuando finalice la transmisión de un dato (**TXC** en alto).

- **Bit 5 – UDRIE: Habilitador de interrupción por buffer de transmisión vacío**

  Si es puesto en alto, y el habilitador global de interrupciones también, se produce una interrupción cuando el buffer transmisor esté vacío (**UDRE** en alto).

- **Bit 4 – RXEN: Habilitador del receptor**

  Debe ser puesto en alto para poder recibir datos a través de la USART, con esto, la terminal RXD ya no se puede utilizar como entrada o salida de propósito general.

- **Bit 3 – TXEN: Habilitador del transmisor**

  Debe ser puesto en alto para poder transmitir datos a través de la USART, con esto, la terminal TXD ya no se puede utilizar como entrada o salida de propósito general.

- **Bit 2 – UCSZ2: Bit 2 para definir el tamaño de los datos**

  Junto con otros 2 bits del registro **UCSRC** permiten definir el tamaño de los datos a enviar o recibir. En la tabla 6.4 se definen los diferentes tamaños, de acuerdo con el valor de **UCSZ[2:0]**.

- **Bit 1 – RXB8: 9º bit recibido**

  Durante la recepción de datos de 9 bits, los 8 bits menos significativos se reciben en el registro **UDR**, en **RXB8** se recibe al bit más significativo.

- **Bit 0 – TXB8: 9º bit a ser transmitido**

  Para transmitir datos de 9 bits, los 8 bits menos significativos deben ubicarse en el registro **UDR**, en **TXB8** debe colocarse al bit más significativo.

El registro **UCSRC** comparte su dirección con **UBRRH** (0x20). El bit **URSEL** se utiliza para seleccionar el registro a escribir. Para leer a **UCSRC** deben realizarse 2 lecturas consecutivas, una lectura aislada proporciona el valor de **UBRRH**. Los bits del registro **UCSRC** son:

	7	6	5	4	3	2	1	0	
0x20	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	**UCSRC**

- **Bit 7 – URSEL: Selector para escribir en UBRRH o UCSRC**

  Con **URSEL** = 0 se escribe en **UBRRH** y con **URSEL** = 1 se escribe en **UCSRC**.

- **Bit 6 – UMSEL: Selector del modo de la USART**

  Selecciona entre una operación síncrona o asíncrona. Con **UMSEL** = 0 la operación de la USART es asíncrona. Con **UMSEL** = 1 su operación es síncrona.

- **Bits 5 y 4 – UMP[1:0]: Bits para la configuración de la paridad**

  Con las combinaciones de estos bits se determina el uso del bit de paridad y se define su tipo. En la tabla 6.5 se muestran las diferentes opciones.

- **Bit 3 – USBS: Selector del número de bits de paro**

  Entre dato y dato puede incluirse sólo 1 bit de paro (**USBS** = 0) o pueden ser 2 bits de paro (**USBS** = 1).

- **Bits 2 y 1 – UCSZ[1:0]: Bits 1 y 0 para definir el tamaño de los datos**

  Junto con el bit **UCSZ2** del registro **UCSRB** definen el tamaño de los datos a enviar o recibir. En la tabla 6.4 se muestran los diferentes tamaños, de acuerdo con el valor de **UCSZ[2:0]**.

- **Bit 0 – UCPOL: Polaridad del reloj**

  En operaciones síncronas, **UCPOL** determina el flanco de reloj en el cual se ajustan los datos (cuando se está transmitiendo) o en el que son muestreados (cuando se está recibiendo). En la tabla 6.3 se mostraron los flancos de activación (polaridad), dependiendo del valor del bit **UCPOL**.

Tabla 6.4 Opciones para el tamaño de los datos

UCSZ2	UCSZ1	UCSZ0	Tamaño de los datos
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	Reservado
1	0	1	Reservado
1	1	0	Reservado
1	1	1	9 bits

Tabla 6.5 Activación del bit de paridad y sus diferentes tipos

UPM1	UPM0	Configuración
0	0	Sin bit de paridad
0	1	Reservado
1	0	Paridad Par
1	1	Paridad Impar

6.1.5 Ejemplos de Uso de la USART

Los siguientes ejemplos muestran cómo configurar a la USART y las rutinas básicas para transmisión y recepción, sondeando las banderas o utilizando interrupciones. Los primeros 4 ejemplos son realizados en ensamblador, el 5º es realizado en ensamblador y en lenguaje C, y el 6º sólo en lenguaje C.

Ejemplo 6.1 Escriba una rutina que configure a la USART para una comunicación asíncrona de 8 bits, sin paridad, a 9600 bps y sin manejo de interrupciones. Suponga que el microcontrolador opera a 1 MHz.

```
conf_USART:
       CLR    R16           ; Valor de recarga para una velocidad de 9600 bps
       OUT    UBRRH, R16
       LDI    R16, 12
       OUT    UBRRL, R16

       LDI    R16, 0x02     ; Modo asíncrono a doble velocidad
       OUT    UCSRA, R16

       LDI    R16, 0x18     ; Habilita receptor y transmisor
       OUT    UCSRB, R16    ; sin interrupciones

       LDI    R16, 0x86     ; 1 bit de paro, sin paridad y datos de 8 bits
       OUT    UCSRC, R16
       RET
```

Ejemplo 6.2 Realice una rutina que espere hasta recibir un dato por el puerto serie y lo coloque en R17.

```
Recibe:
       SBIS   UCSRA, RXC          ; Espera a que haya un dato disponible
       RJMP   Recibe              ; por recepción
       IN     R17, UDR            ; Lo deja en R17
       RET
```

Ejemplo 6.3 Escriba una rutina que transmita un carácter ubicado en R17, la rutina se debe asegurar que el registro UDR de transmisión está vacío.

```
Transmite:
       SBIS   UCSRA, UDRE         ; Garantiza que el buffer transmisor
       RJMP   Transmite           ; está vacío
       OUT    UDR, R17            ; Transmite el contenido de R17
       RET
```

Ejemplo 6.4 Apoyado en las rutinas de los 3 ejemplos anteriores, escriba un programa “eco” que transmita cada dato recibido.

```
        .include  <m8def.inc>

        LDI     R16, 0x04           ; Ubica al apuntador de pila
        LDI     R17, 0x5F
        OUT     SPH, R16
        OUT     SPL, R17

        RCALL   conf_USART          ; Configura la USART
loop:
        RCALl   Recibe              ; Espera a recibir un dato
        RCALL   Transmite           ; Transmite el dato recibido
        RJMP    loop
```

Ejemplo 6.5 Repita el ejemplo 6.4 usando interrupciones.

Se requiere de la interrupción por recepción para que en cualquier momento el MCU pueda recibir un dato, cuando esto sucede, se asegura que el buffer de transmisión está vacío para iniciar con el envío, el programa principal permanece ocioso. La solución en lenguaje ensamblador es:

```
        .include  <m8def.inc>

        .org    0x000
        RJMP    inicio

        .org    0x00B
        RJMP    ISR_USART_RXC

        .org    0x013               ; Después de los vectores de interrupciones
inicio:
        LDI     R16, 0x04           ; Ubica al apuntador de pila
        LDI     R17, 0x5F
        OUT     SPH, R16
        OUT     SPL, R17
        RCALL   conf_USART          ; Configura la USART
        SEI                         ; Habilitador global de interrupciones
loop:
        RJMP    loop                ; Ocioso, la interrupción recibe el dato

conf_USART:
        CLR     R16                 ; Para una velocidad de 9600 bps
        OUT     UBRRH, R16
        LDI     R16, 12
        OUT     UBRRL, R16
        LDI     R16, 0x02           ; Modo asíncrono a doble velocidad
        OUT     UCSRA, R16
        LDI     R16, 0x98           ; Habilita receptor y transmisor
        OUT     UCSRB, R16          ; con interrupción por recepción
        LDI     R16, 0x86           ; 1 bit de paro, sin paridad y datos de 8 bits
        OUT     UCSRC, R16
        RET
```

```
ISR_USART_RXC:
       IN     R17, UDR            ; Recibe un dato serial
Buffer_vacio:
       SBIS   UCSRA, UDRE         ; Garantiza que el buffer está vacío
       RJMP   Buffer_vacio
       OUT    UDR, R17            ; Transmite el contenido de R17
       RETI
```

La solución en lenguaje C es:

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

void   conf_USART();              // Función para configurar la USART

ISR(USART_RXC_vect) {
unsigned  char  dato;
       dato = UDR;
       while(!( UCSRA & 1 << UDRE ));    // Garantiza que el buffer está vacío
       UDR = dato;                       // Transmite el dato recibido
}

int    main() {
       conf_USART();              // Configura a la USART
       sei();                     // Habilitador global de interrupciones
       while(1)                   // En el lazo infinito permanece ocioso
              asm("nop");
}

void  conf_USART() {

       UBRRH = 0;                 // Para una velocidad de 9600 bps
       UBRRL = 12;
       UCSRA = 0x02;              // Modo asíncrono a doble velocidad
       UCSRB = 0x98;              // Habilita receptor y transmisor
                                  // con interrupción por recepción
       UCSRC = 0x86;              // 1 bit de paro, sin paridad y datos
                                  // de 8 bits
}
```

Las soluciones son muy similares, porque realmente es el recurso el que se encarga de la recepción y transmisión.

Ejemplo 6.6 Escriba un programa que envíe un carácter ASCII imprimible por el puerto serie, cada vez que se presione un botón. Los caracteres ASCII imprimibles están en el rango de 0x20 a 0x7E. Además, en cualquier momento puede arribar serialmente un dato y éste debe mostrarse en el puerto B. En la figura 6.8 se muestra el hardware requerido.

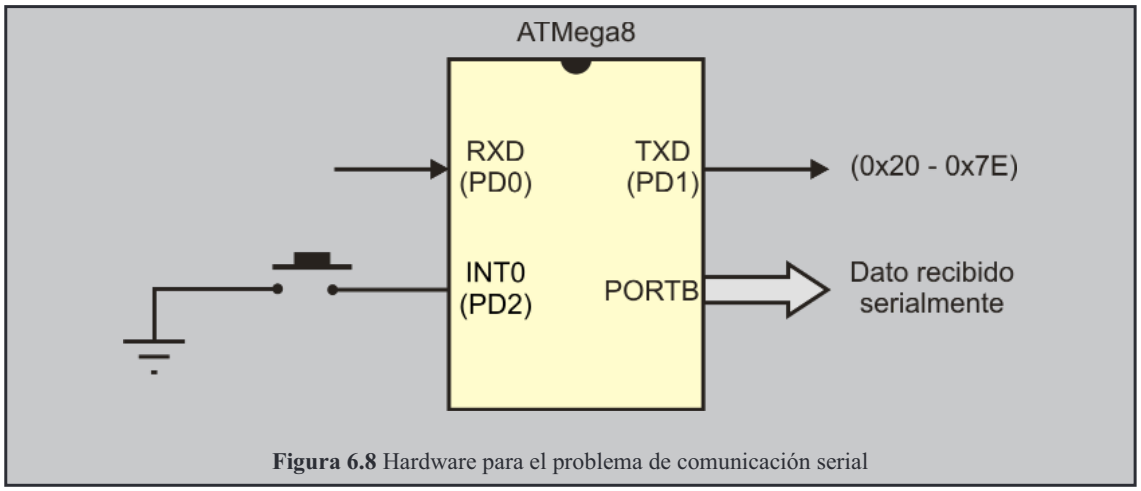


Figura 6.8 Hardware para el problema de comunicación serial

Se utilizan 2 interrupciones: por recepción, para ubicar el dato recibido en el puerto B y una interrupción externa, que prepara y envía el siguiente carácter imprimible, asegurando que no hay una transmisión en proceso. El código en lenguaje C es:

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

// Variables globales
unsigned  char dato = 0x20;       // El dato a enviar
void   conf_USART();              // Función de configuración de la
USART

ISR(USART_RXC_vect) {             // ISR de recepción serial
      PORTB = UDR;                // El dato recibido se muestra en el
puerto B
}

ISR(INT0_vect) {                          // ISR de la INT0
      while(!( UCSRA & 1 << UDRE ));      // Garantiza que el buffer
                                          // está vacío
      UDR = dato;                         // Transmite el dato actual
      dato++;                             // Prepara el dato siguiente
      if( dato == 0x7F )                  // Si llegó al límite
            dato = 0x20;                  // Inicia nuevamente
}

int    main() {

      conf_USART();               // Configura a la USART
      DDRD = 0b00000010;          // sólo TXD es salida en PORTD
      PORTD = 0b00000100;         // Resistor de Pull-Up en INT0
      DDRB = 0xFF;                // Puerto B como salida
      MCUCR = 0B00000010;         // Configura INT0 por flanco de bajada
      GICR = 0B01000000;          // Habilita la INT0
      sei();                      // Habilitador global de interrupciones
      while(1)                    // En el lazo infinito permanece ocioso
            asm("nop");
```

```
}

void  conf_USART() {

       UBRRH = 0;                  // Para una velocidad de 9600 bps
       UBRRL = 12;
       UCSRA = 0x02;               // Modo asíncrono a doble velocidad
       UCSRB = 0x98;               // Habilita receptor y transmisor
                                   // con interrupción por recepción
       UCSRC = 0x86;               // 1 bit de paro, sin paridad y datos de 8 bits
}
```

El ejemplo anterior muestra cómo los recursos de transmisión y recepción son independientes.

6.2 **Comunicación Serial por SPI**

La Interfaz Serial Periférica (SPI, *Serial Peripheral Interface*) establece un protocolo estándar de comunicaciones, usado para transferir paquetes de información de 8 bits entre circuitos integrados. El protocolo SPI permite una transferencia síncrona de datos a muy alta velocidad entre un microcontrolador y dispositivos periféricos o entre microcontroladores.

En la conexión de dos dispositivos por SPI, siempre ocurre que uno funciona como Maestro y otro como Esclavo, aunque es posible el manejo de un sistema con múltiples Esclavos. El Maestro es quien determina cuándo enviar los datos y quien genera la señal de reloj. El Esclavo no puede enviar datos por sí mismo y tampoco es capaz de generar la señal de reloj. Esto significa que el Maestro tiene que enviar datos al Esclavo para obtener información de él.

En la figura 6.9 se muestra una conexión entre dos dispositivos vía SPI, esta interfaz no es propia de los microcontroladores, también puede incluirse en memorias, sensores, ADCs, DACs, etc., estos dispositivos usualmente funcionan como Esclavos. Puede verse que la base del protocolo SPI son registros de desplazamiento con sus correspondientes circuitos de control.

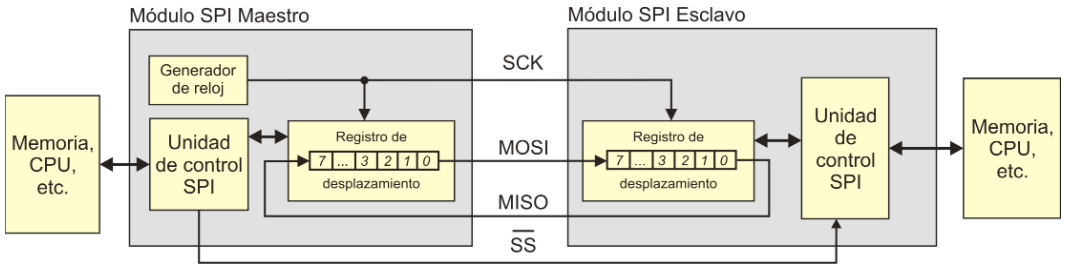


Figura 6.9 Conexión entre 2 dispositivos vía SPI

Las señales que intervienen en la comunicación son:

- **MOSI** (*Master Output, Slave* Input): Señal de salida del Maestro y entrada en el Esclavo. Típicamente el envío de información inicia con el bit menos significativo.

- **MISO** *(Master Input, Slave Output*): Señal de entrada al Maestro y salida en el Esclavo. Proporciona el mecanismo para que el Esclavo pueda dar respuesta al Maestro.

- **SCK** (*Shift Clock*): Es la señal de reloj para sincronizar la comunicación. Es generada por el Maestro.

- **SS** (*Slave Select*): Esta señal es útil para el manejo de múltiples Esclavos, para ello, deben realizarse los arreglos necesarios en el hardware, de manera que sólo se habilite un Esclavo en cada transmisión. Esta señal generalmente es activa en un nivel bajo.

El envío y recepción se realizan en forma simultánea, mientras el Maestro envía datos por MOSI recibe una respuesta del Esclavo por MISO. Sincronizando el envío y recepción con la señal de reloj, es decir, en el mismo ciclo de reloj se trasmite y recibe un bit. Debido a esto, los registros de desplazamiento de 8 bits pueden ser considerados como un registro de desplazamiento circular de 16 bits. Esto significa que después de 8 pulsos de reloj, el Maestro y el Esclavo han intercambiado un dato de 8 bits.

6.2.1 Organización de la Interfaz SPI en los AVR

Bajo el protocolo SPI, un microcontrolador AVR puede funcionar como Maestro o como Esclavo, de acuerdo con los requerimientos de la aplicación. En la figura 6.10 se muestra la organización del hardware para la interfaz SPI. El recurso se habilita con la puesta en alto del bit **SPE** del registro de control (**SPCR**, *SPI Control Register*), ninguna operación con la interfaz SPI es posible sin esta habilitación. La configuración como Maestro se realiza con la puesta en alto del bit **MSTR** en el registro **SPCR**, en caso contrario, el dispositivo funciona como Esclavo.

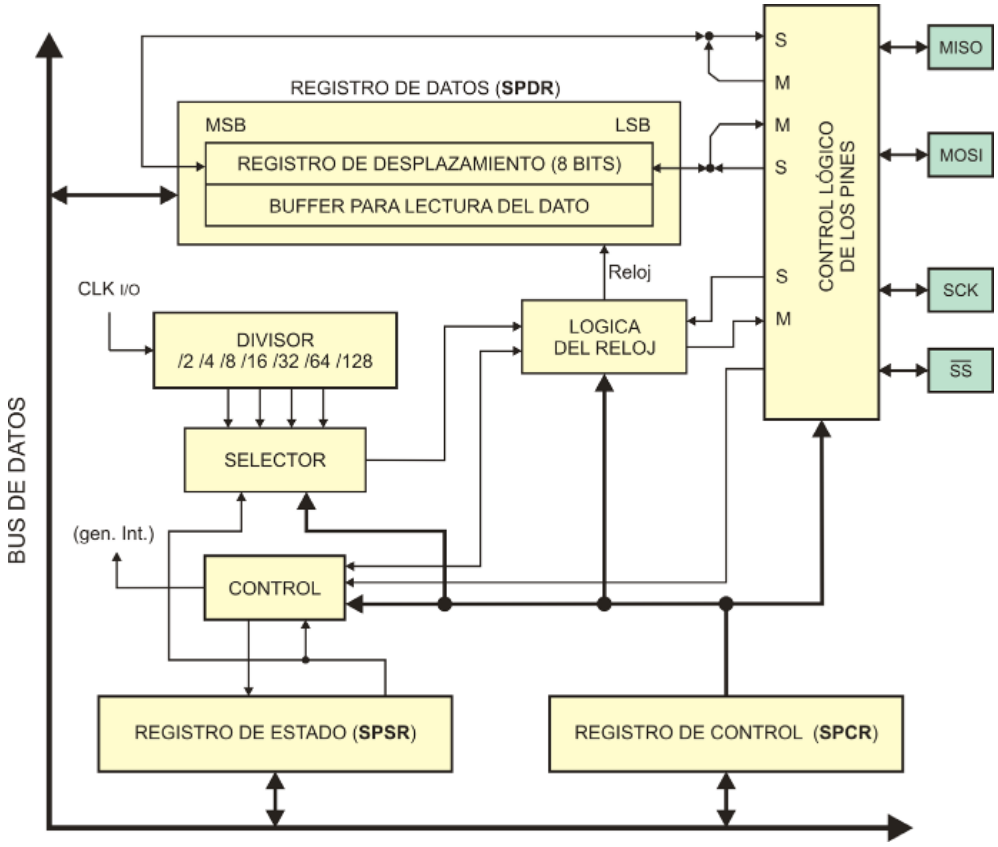


Figura 6.10 Organización de la interfaz SPI en los AVR

El modo en el que cada dispositivo funciona determina si las líneas: MISO, MOSI, SCK y SS, son señales de entrada o de salida, ésta es la tarea del módulo para el control lógico de los pines. El flujo de datos se establece de acuerdo con la tabla 6.6, donde se observa que sólo los pines de entrada se configuran automáticamente, los pines de salida tienen que ser configurados por software para evitar daños.

Tabla 6.6 Flujo de datos en la interfaz SPI

PIN	Maestro	Esclavo
MOSI	Definido por el usuario	Entrada
MISO	Entrada	Definido por el usuario
SCK	Definido por el usuario	Entrada
SS	Definido por el usuario	Entrada

La interfaz SPI incluye un buffer simple para la transmisión y un buffer doble para la recepción. El acceso a ambos se realiza por medio del registro de datos (**SPDR**, *SPI Data Register*).

La transmisión de un dato inicia después de escribir en el registro **SPDR**. No se debe escribir un dato nuevo mientras un ciclo de envío-recepción esté en progreso. En los AVR puede elegirse si se envia primero al bit más significativo o al menos significativo, esto se determina con el bit **DORD** (*data order*) del registro **SPCR**.

Los bits recibidos son colocados en el buffer de recepción inmediatamente después de que la transmisión se ha completado. El buffer de recepción tiene que ser leído antes de iniciar con la siguiente transmisión, de lo contrario, se va a perder el dato recibido. El dato del buffer de recepción se obtiene con la lectura del registro **SPDR**.

El fin de una transferencia se indica con la puesta en alto de la bandera **SPIF** en el registro de estado (**SPSR**, *SPI Status Register*). Este evento produce una interrupción si se habilitó la interrupción por transferencia serial completa vía SPI, la cual se habilita con la puesta en alto del bit **SPIE** en el registro **SPCR**, también se requiere la activación del habilitador global de interrupciones (bit **I** del registro **SREG**). Aunque la bandera **SPIF** también puede sondearse por software.

Una colisión de escritura ocurre si se realiza un acceso al registro **SPDR** mientras hay una transferencia en progreso, debido a que se podrían corromper los datos de la transferencia actual. Una colisión de escritura generalmente es el error de un Esclavo, porque desconoce si el Maestro inició con una transferencia. El Maestro sabe si hay una transferencia en progreso, por ello, no debería generar errores de colisión de escritura, no obstante, el hardware puede detectar estos errores tanto en modo Maestro como en modo Esclavo. Se indica con la puesta en alto de la bandera **WCOL**, en el registro **SPSR**. Las banderas **WCOL** y **SPIF** se limpian con la lectura del registro **SPSR** y el acceso al registro **SPDR**.

6.2.2 Modos de Transferencias SPI

Las transferencias son sincronizadas con la señal de reloj (SCK), un bit es transferido en cada ciclo. Para que la interfaz SPI sea compatible con diferentes dispositivos, la sincronización de los datos con el reloj es flexible, el usuario puede definir la polaridad de la señal de reloj y la fase del muestreo de datos. La polaridad se refiere al estado lógico de la señal de reloj mientras no hay transferencias. La fase define si el primer bit es muestreado en el primer flanco de reloj (en fase) o en el flanco siguiente, insertando un retraso al inicio para asegurar que la interfaz SPI receptora está lista.

En el registro **SPCR** se tienen 2 bits para configurar estos parámetros, el bit **CPOL** es para definir la polaridad y con el bit **CPHA** se determina la fase. Con los valores de **CPOL** y **CPHA** se tienen 4 combinaciones que corresponden con los modos de transferencia, éstos se definen en la tabla 6.7 y en la figura 6.11 se muestran los diagramas de tiempo para distinguirlos.

Tabla 6.7 Modos de transferencias SPI

CPOL	CPHA	Modo SPI	Descripción
0	0	0	Espera en bajo, muestrea en fase
0	1	1	Espera en bajo, muestrea con un retraso
1	0	2	Espera en alto, muestrea en fase
1	1	3	Espera en alto, muestrea con un retraso

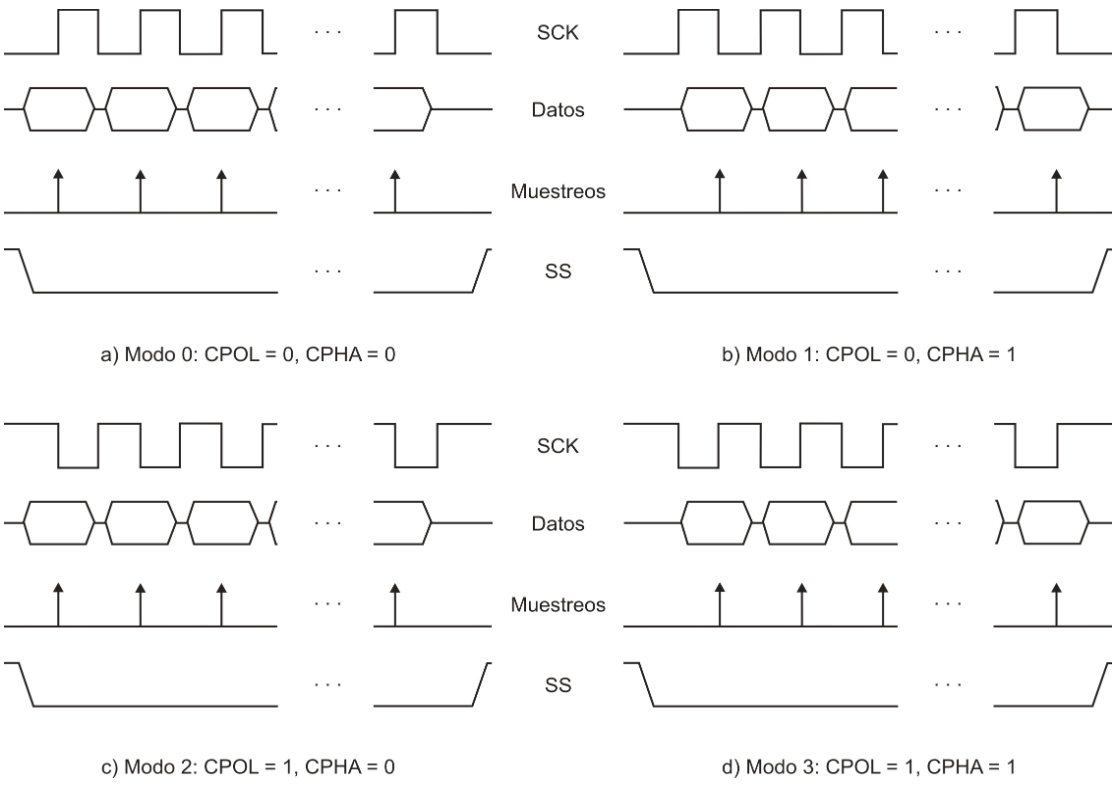


Figura 6.11 Modos de transferencias SPI

6.2.3 Funcionalidad de la Terminal SS

Si el AVR es configurado como Esclavo (bit **MSTR** del registro **SPCR** en bajo), la interfaz SPI se activa cuando existe un nivel bajo en SS. Cuando en la terminal SS se coloca un nivel alto, la interfaz SPI queda pasiva y no reconoce los datos de entrada, si hay datos parcialmente recibidos en el registro de desplazamiento, éstos van a ser desechados. Además, la terminal SS ayuda a mantener al Esclavo sincronizado con el reloj del Maestro, esto también puede notarse en la figura 6.11.

Si el AVR es configurado como Maestro (bit **MSTR** del registro **SPCR** en alto), el usuario debe determinar la dirección de la terminal SS. Como salida, SS es una salida general que no afecta a la interfaz SPI. Típicamente se debería conectar con la terminal SS de un Esclavo, activándolo o desactivándolo por software.

Si SS se configura como entrada, se le debe suministrar un nivel alto para asegurar su operación como Maestro. Si algún periférico introduce un nivel bajo, significa que otro Maestro está intentando seleccionar al MCU como Esclavo para enviarle datos. Para evitar conflictos en las transferencias, en la interfaz SPI automáticamente se realizan las siguientes acciones:

1. El bit **MSTR** del registro **SPCR** es limpiado para que el MCU sea tratado como Esclavo. Al ser un Esclavo, las terminales MOSI y SCK se vuelven entradas.

2. La bandera **SPIF** en el registro **SPSR** es puesta en alto, de manera que si el habilitador de la interrupción y el habilitador global están activos, se va a ejecutar la ISR correspondiente.

Por lo tanto, si la interfaz SPI siempre va a funcionar como Maestro, y existe la posibilidad de que la terminal SS sea llevada a un nivel bajo, en la ISR debe ajustarse el valor del bit **MSTR**, para mantenerlo operando en el modo correcto.

Si una aplicación requiere el manejo de un Maestro y varios Esclavos, otras terminales del Maestro deben funcionar como habilitadoras, conectándose con las terminales SS de cada uno de los Esclavos. Cuando el Maestro requiere intercambiar información con un Esclavo, primero debe habilitarlo por software. En la figura 6.12 se muestra la conexión de un Maestro y 3 Esclavos, el Maestro se basa en un ATMega16 y cada Esclavo es un ATMega8.

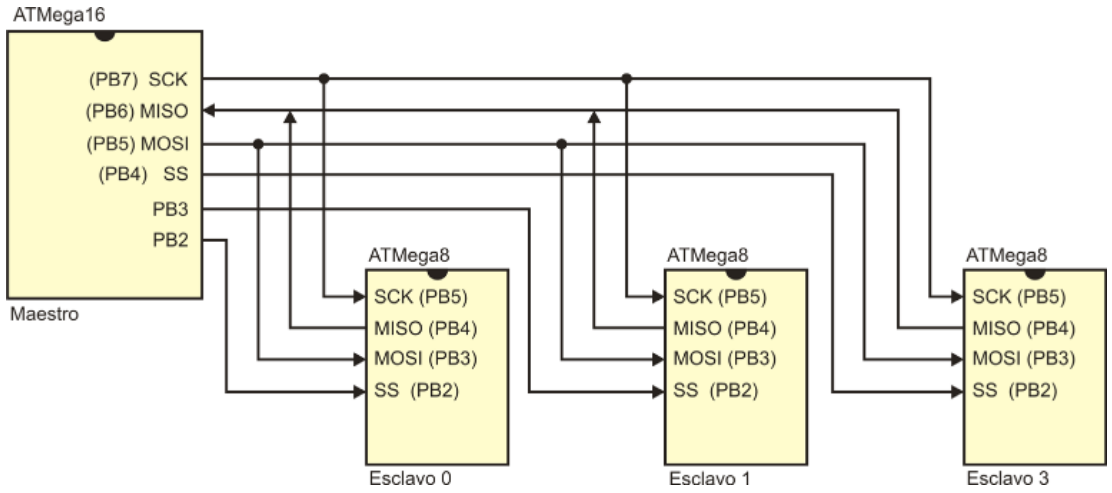
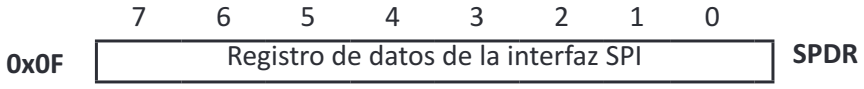


Figura 6.12 Conexión de un Maestro y 3 Esclavos por SPI

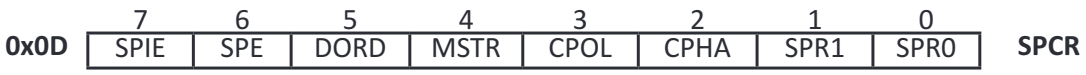
La figura 6.12 ilustra la forma de conectar 3 microcontroladores como Esclavos. En la práctica, los Esclavos suelen ser dispositivos con una tarea específica, como ADCs, DACs, memorias, etc.

6.2.4 Registros para el Manejo de la Interfaz SPI

El registro **SPDR** es el buffer para transmisión y recepción de la interfaz SPI, está conectado directamente con el registro de desplazamiento. Una escritura en **SPDR** da inicio a la transmisión de un dato. Una lectura en **SPDR** proporciona el dato recibido en el registro de desplazamiento.



El registro de control es el **SPCR**, sus bits son:



- **Bit 7 – SPIE: Habilitador de interrupción por SPI**

  Debe estar en alto, junto con el habilitador global, para que la bandera **SPIF** genere una interrupción por transferencia serial completa vía SPI.

- **Bit 6 – SPE: Habilitador de la interfaz SPI**

  Habilita a la interfaz SPI, debe estar en alto para realizar transferencias por esta interfaz.

- **Bit 5 – DORD: Orden de los datos**

  Con un 0 en **DORD**, se transfiere primero al bit más significativo (MSB). Con un 1 se transfiere primero al bit menos significativo (LSB).

- **Bit 4 – MSTR: Habilitador como Maestro**

  Un 1 en **MSTR** habilita a la interfaz como Maestro. Un 0 la deja como Esclavo.

- **Bit 3 – CPOL: Polaridad del reloj**

  Determina la polaridad del reloj (SCK) cuando la interfaz SPI está inactiva. En la figura 6.11 se mostró el efecto de este bit en la señal SCK.

- **Bit 2 – CPHA: Fase del reloj**

  Determina si los datos son muestreados en fase con el reloj o si se inserta un retardo inicial de medio ciclo de reloj. En la figura 6.11 se mostró el efecto de este bit en el muestreo de datos.

- **Bits 1 y 0 – SPR[1:0]**

  Determinan la frecuencia a la cual se genera la señal de reloj SCK. No tienen efecto si el MCU está configurado como Esclavo. En la tabla 6.8 se muestran las diferentes frecuencias obtenidas a partir del reloj del sistema. También se observa el efecto del bit **SPI2X**, con el que se duplica la frecuencia de la señal SCK.

Tabla 6.8 Frecuencia de la señal de reloj (SCK)

SPI2X	SPR1	SPR0	Frecuencia de SCK
0	0	0	fosc/4
0	0	1	fosc/16
0	1	0	fosc/64
0	1	1	fosc/128
1	0	0	fosc/2
1	0	1	fosc/8
1	1	0	fosc/32
1	1	1	fosc/64

El registro de estado es el **SPSR**, cuyos bits son:

	7	6	5	4	3	2	1	0	
0x0E	SPIF	WCOL	-	-	-	-	-	SPI2X	**SPSR**

- **Bit 7 – SPIF: Bandera de fin de transferencia SPI**

  Produce una interrupción, si se habilitó la interrupción por transferencia serial completa vía SPI y al habilitador global de interrupciones. Aunque también puede sondearse por software.

- **Bit 6 – WCOL: Bandera de colisión de escritura**

  Se pone en alto si se escribe en el registro **SPDR** mientras hay una transferencia en progreso. Las banderas **WCOL** y **SPIF** se limpian con la lectura del registro **SPSR** y el acceso al registro **SPDR.**

- **Bits 5 al 1 – No están implementados**

- **Bit 0 – SPI2X: Duplica la frecuencia de transmisión**

  En la tabla 6.8 se observa el efecto del bit **SPI2X**, duplicando la frecuencia de la señal SCK, definida por los bits **SPR1** y **SPR0**.

6.2.5 Ejemplos de Uso de la Interfaz SPI

En los siguientes ejemplos se ilustra cómo utilizar a la interfaz SPI, cuando el MCU funciona como Maestro o como Esclavo. Puesto que el papel del Maestro difiere del Esclavo, en cada problema se muestran 2 soluciones, codificándolas únicamente en lenguaje C.

Ejemplo 6.7 Sin emplear interrupciones, configure un ATMega8 como Maestro y otro como Esclavo. El Maestro debe enviar una cadena de caracteres terminada con el carácter nulo (0x00). El Esclavo debe colocar cada carácter recibido en su puerto D. Después de enviar al carácter nulo, el Maestro debe solicitar al Esclavo la longitud de la cadena y colocarla en su puerto D.

Suponga que los dispositivos están operando a 1 MHz y configure para que las transmisiones se realicen a 125 KHz.

Se asume que ambos dispositivos se energizan al mismo tiempo, aun con ello, el Maestro espera 100 mS para que el Esclavo esté listo. Después de enviar al carácter nulo, nuevamente el Maestro espera 100 mS antes de solicitar la longitud de la cadena, para evitar errores por colisión de escritura. Es posible realizar diagramas de flujo, puesto que no se emplean interrupciones. El comportamiento del Maestro se describe en el diagrama de la figura 6.13.

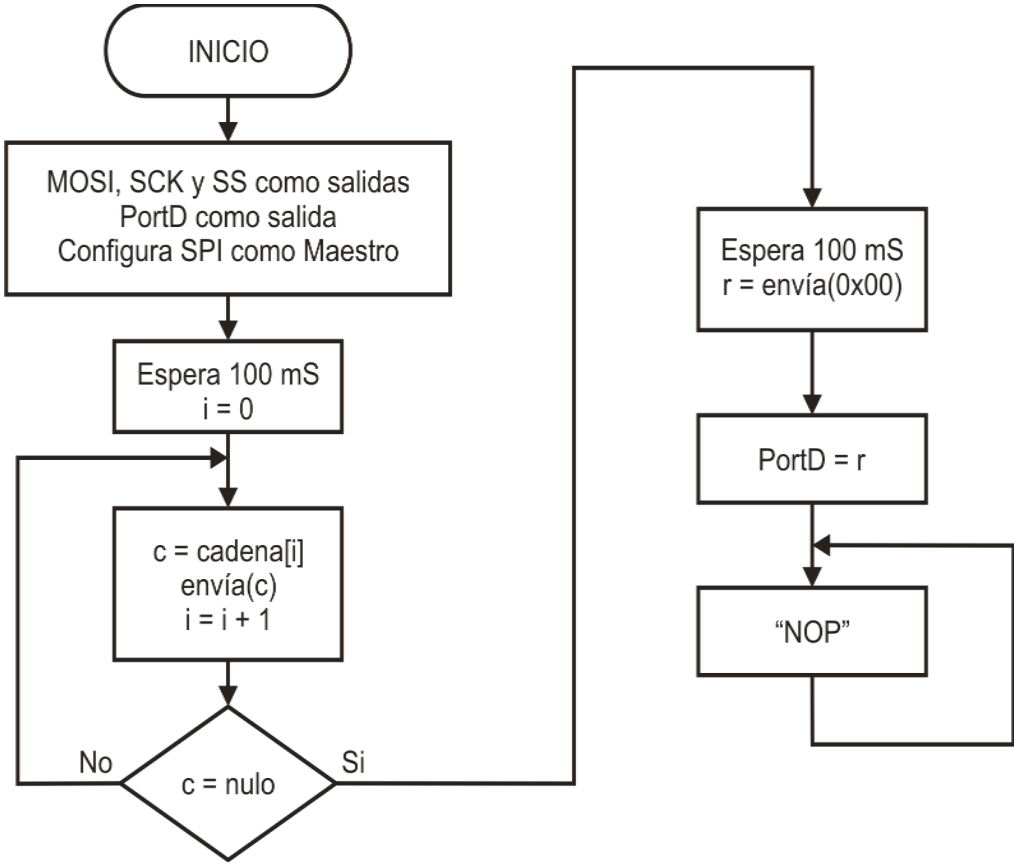


Figura 6.13 Comportamiento del Maestro

La función que envía un dato por SPI regresa el dato recibido, sin embargo, no es importante la respuesta mientras se envíen los caracteres, la respuesta se ignora en las llamadas realizadas dentro del ciclo repetitivo. Por el contrario, al solicitar el número de caracteres sólo la respuesta es importante, podría enviarse cualquier carácter.

El Esclavo tiene un comportamiento diferente, el cual se ilustra en la figura 6.14. Donde se observa que el Esclavo por sí mismo no da una respuesta al Maestro, su respuesta es colocada en el registro **SPDR** y espera a que el Maestro la solicite.

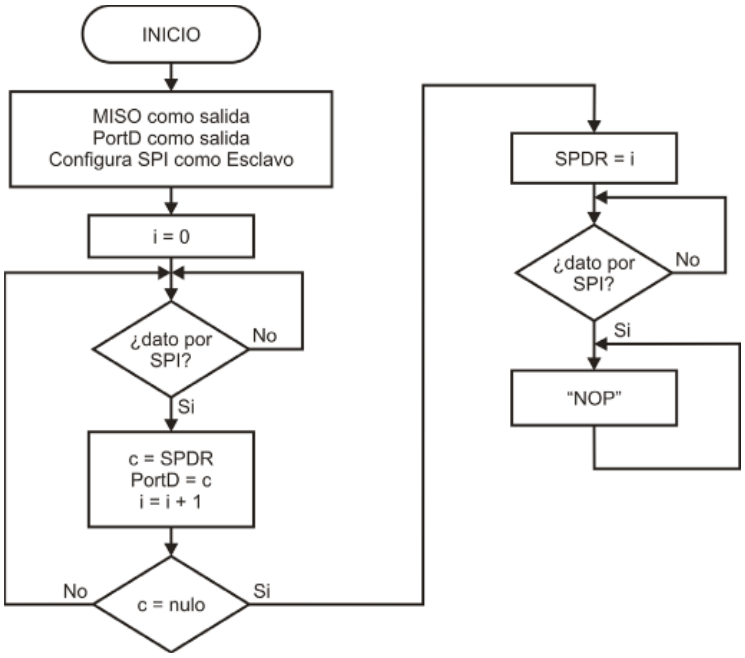


Figura 6.14 Comportamiento del Esclavo

El código en lenguaje C para el Maestro, considerando una cadena constante, es:

```
#define F_CPU 1000000UL             // Frecuencia de trabajo de 1 MHz
#include      <util/delay.h>        // Funciones para retrasos
#include      <avr/io.h>            // Definiciones de Registros I/O

unsigned char envia_SPI(unsigned char dato);   // Prototipo de la función

int  main() {                       // Programa principal
unsigned  char i, r;
char   c;
char   cadena[] = "cadena de prueba";

DDRB = 0b00101100;                  // MOSI, SCK y SS como salidas
DDRD = 0xFF;                        // Puerto D como salida
PORTB = 0x04;                       // SS en alto, Esclavo inhabilitado

SPCR = 0x51;                  // Habilita la interfaz SPI como Maestro
SPSR = 0x01;                        // Ajustando para 125 kHz

_delay_ms(100);                     // Espera a que el Esclavo esté listo
i = 0;                              // índice para el arreglo de datos

do {                                // Envía caracteres hasta encontrar
      c = cadena[i];                // al carácter nulo
      envia_SPI(c);
      i = i + 1;
} while( c != 0x00);

_delay_ms(100);                     // Espera a que el Esclavo escriba
```

```
                                    // respuesta
r = envia_SPI(0x00);                // Solicita respuesta
PORTD = r;                          // Muestra respuesta

while(1)                            // Lazo infinito
      asm("nop");
}
/* Función para enviar un dato por SPI, debe considerarse la
habilitación e inhabilitación del Esclavo.                          */

unsigned  char  envia_SPI(unsigned char  dato) {
unsigned  char  resp;

PORTB &= 0xFB;                      // SS en bajo, Esclavo habilitado
SPDR = dato;                        // Dato a enviar

while(!(SPSR & 1 << SPIF));         // Espera fin de envío

resp = SPDR;                        // Lee la respuesta del Esclavo
PORTB |= 0x04;                      // SS en alto, Esclavo inhabilitado
return  resp;                       // Regresa la respuesta
}
```

El código para el Esclavo es:

```
#include     <avr/io.h>             // Definiciones de Registros I/O

int  main() {                       // Programa principal
unsigned  char i;
char   c;

DDRB = 0b00010000;                  // MISO como salida
DDRD = 0xFF;                        // Puerto D como salida
SPCR = 0x41;                        // Habilita la interfaz SPI como Esclavo
SPSR = 0x01;                        // Ajustando para 125 kHz
                                    // Aunque puede omitirse en el Esclavo
i = 0;                              // índice para contar los datos
do {
      while(!(SPSR & 1 << SPIF));       // Espera a recibir un dato
      c = SPDR;                         // Lee el dato
      PORTD = c;                        // Muestra el dato
      i = i + 1;
} while( c != 0x00);

SPDR = i;                           // Deja lista la respuesta
while(!(SPSR & 1 << SPIF));         // Espera una petición del Maestro
c = SPDR;                           // Lee, sólo para limpiar la bandera

while(1)                            // Lazo infinito
      asm("nop");}
```

En la solución del ejemplo anterior, se observa que si se utilizaran interrupciones, no se optimizaría al programa del Maestro, puesto que el Maestro define en qué momento

se realiza la transferencia de un dato. Por el contrario, el Esclavo se beneficiaría ampliamente porque puede estar desarrollando otras actividades en lugar de un sondeo continuo.

Ejemplo 6.8 Tomando como base el diagrama de la figura 6.12, al Maestro colóquele 2 arreglos de interruptores (con 8 y 2 interruptores) y un botón. Los 2 interruptores son para seleccionar un Esclavo y los 8 para introducir un dato. Con ello, cada vez que el botón es presionado, el Maestro debe enviar el dato al Esclavo seleccionado. Puesto que las direcciones de los Esclavos son 0, 1 y 2, si se selecciona el 3, el dato se debe mandar a todos los Esclavos (difusión). Cuando un Esclavo reciba un dato, lo debe mostrar en su puerto D.

El hardware, complementado con los interruptores y LEDs, se muestra en la figura 6.15.

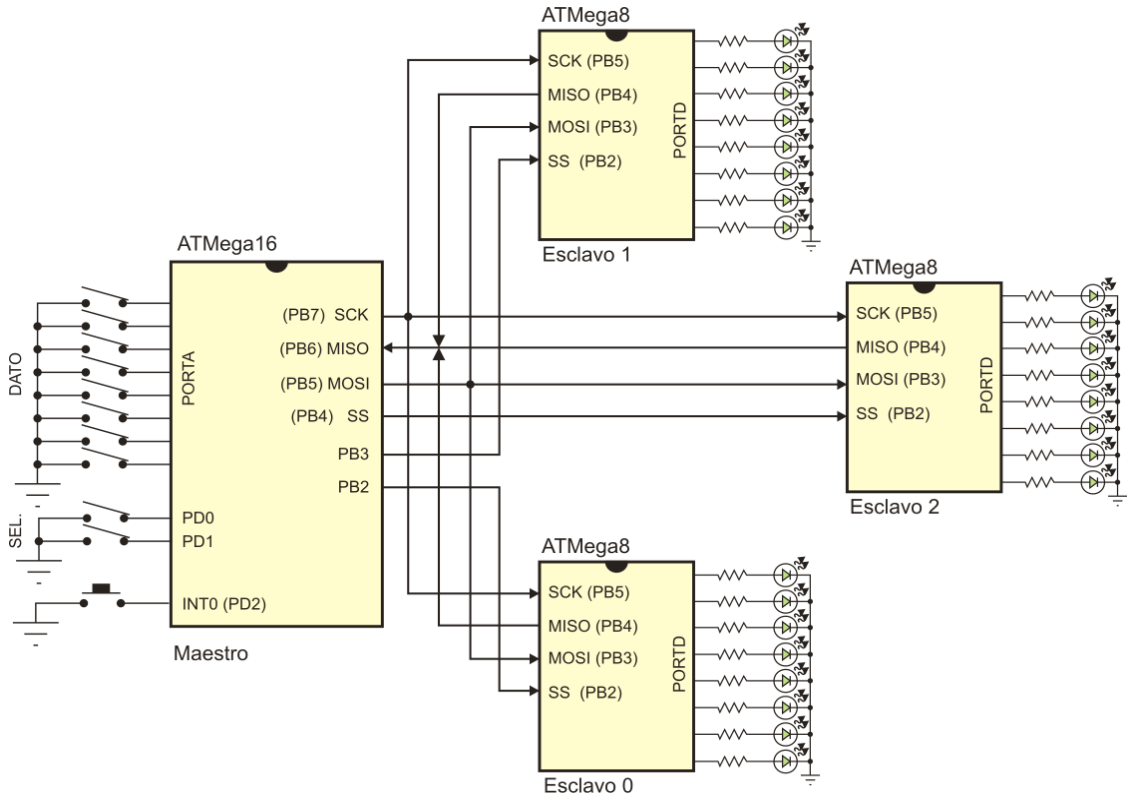


Figura 6.15 Envío de información de un Maestro a 3 Esclavos por SPI

En el Maestro se utiliza la interrupción externa para solicitar un envío. En su ISR se espera hasta que la transferencia concluya, para evitar errores de colisión de escritura. Podría ocurrir que un rebote en el botón solicite un envío mientras una transferencia está en progreso. El programa para el Maestro es:

```
#include     <avr/io.h>                  // Funciones de entrada/salida
#include     <avr/interrupt.h>           // Funciones de las interrupciones

unsigned char envia_SPI(unsigned char dato);   // Prototipo de la función
ISR(INT0_vect) {                  // Rutina que se ejecuta al presionar el botón
unsigned char  sel, dato;

sel = PIND & 0x03;                // Obtiene el número de Esclavo seleccionado
dato = PINA;                      // Obtiene el dato a enviar
switch(sel) {
      case 0 : PORTB &= 0b11111011;     // Habilita al Esclavo 0
               envia_SPI(dato);         // envía el dato
               PORTB |= 0b00000100;     // Inhabilita al Esclavo 0
               break;
      case 1 : PORTB &= 0b11110111;     // Habilita al Esclavo 1
               envia_SPI(dato);         // envía el dato
               PORTB |= 0b00001000;     // Inhabilita al Esclavo 1
               break;
      case 2 : PORTB &= 0b11101111;     // Habilita al Esclavo 2
               envia_SPI(dato);         // envía el dato
               PORTB |= 0b00010000;     // Inhabilita al Esclavo 2
               break;
      default:                          // Difusión a los 3 Esclavos
               PORTB &= 0b11111011;     // Habilita al Esclavo 0
               envia_SPI(dato);         // envía el dato
               PORTB |= 0b00000100;     // Inhabilita al Esclavo 0
               PORTB &= 0b11110111;     // Habilita al Esclavo 1
               envia_SPI(dato);         // envía el dato
               PORTB |= 0b00001000;     // Inhabilita al Esclavo 1
               PORTB &= 0b11101111;     // Habilita al Esclavo 2
               envia_SPI(dato);         // envía el dato
               PORTB |= 0b00010000;     // Inhabilita al Esclavo 2
      }
}

int  main() {                     // Programa principal

DDRA = 0x00;                      // Puerto A como entrada
DDRB = 0b10111100;                // MOSI, SCK y SS(s) como salidas
DDRD = 0x00;                      // Puerto D como entrada

PORTA = 0xFF;                     // Resistores de pull-up en el puerto A
PORTD = 0x07;                     // Resistores de pull-up en el puerto D

PORTB = 0b00011100;               // Esclavos inhabilitados
SPCR = 0x51;                      // Habilita la interfaz SPI como Maestro
SPSR = 0x01;                      // Ajustando para 125 KHz

MCUCR = 0x02;                     // INT0 por flanco de bajada
GICR = 0x40;                      // Habilita la INT0
sei();                            // Habilitador global de interrupciones
```

```
while(1)                          // Lazo infinito
       asm("nop");
}

unsigned  char  envia_SPI(unsigned char  dato) {
unsigned  char  resp;

SPDR = dato;                      // Dato a enviar
while(!(SPSR & 1 << SPIF));       // Espera fin de envío
resp = SPDR;                      // Lee la respuesta del Esclavo

return  resp;                     // Regresa la respuesta
}
```

En este ejemplo, la habilitación e inhabilitación del Esclavo se realiza en la ISR de la interrupción 0 y no en la función `envia_SPI`, esto porque la función es utilizada para que el Maestro envíe datos a 3 Esclavos diferentes.

Los Esclavos reciben los datos por interrupciones, en su lazo infinito permanecen ociosos. Los 3 Esclavos ejecutan el mismo programa, el cual es:

```
#include      <avr/io.h>                // Funciones de entrada/salida
#include      <avr/interrupt.h>         // Funciones de las interrupciones

ISR(SPI_STC_vect){          // ISR por fin de transferencia por la interfaz SPI
       PORTD = SPDR;        // Lee y muestra el dato recibido
}

int  main() {               // Programa principal

DDRB = 0b00010000;          // MISO como salida

DDRD = 0xFF;                // Puerto D como salida

SPCR = 0xC1;  // Habilita la interfaz SPI, Esclavo con interrupciones

SPSR = 0x01;                // Ajustando para 125 KHz

sei();                      // Habilitador global de interrupciones

while(1)                    // Lazo infinito

       asm("nop");
}
```

El ejemplo anterior sirve como una referencia para expandir el número de puertos de un MCU, siempre que los periféricos a manejar en los nuevos puertos no requieran un tiempo de respuesta rápido, dado que la información fluye de manera serial.

6.3 Comunicación Serial por TWI

La Interfaz Serial de Dos Hilos (TWI, *Two Wire Serial Interface*) es un recurso disponible para que diferentes microcontroladores (u otros dispositivos) se comuniquen por medio de un bus bidireccional de 2 líneas, una para reloj (SCL) y otra para datos (SDA). En la figura 6.16 se muestra la forma en que los diferentes dispositivos se conectan, como hardware externo únicamente se requiere de 2 resistencias conectadas a Vcc (*pull-up*). El protocolo TWI permite al diseñador conectar hasta 128 dispositivos diferentes, cada uno con su propia dirección.

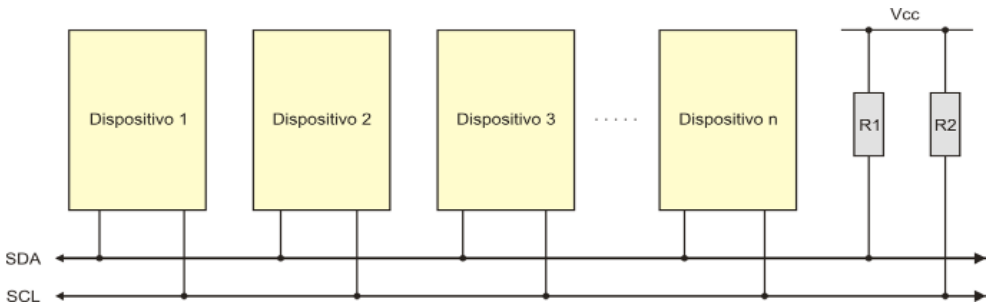


Figura 6.16 Bus de interconexión TWI

Esta interfaz es compatible en su operación con el bus I^2C, el cual es un estándar desarrollado por *Philips Semiconductor* (ahora *NXP Semiconductor*). Por lo tanto, puede utilizarse para el manejo de una gama muy amplia de dispositivos, como manejadores de LCDs y LEDs, puertos remotos de entrada/salida, RAMs, EEPROMs, relojes de tiempo real, ADCs, DACs, etc.

Los dispositivos deben contar con los mecanismos de hardware necesarios para cubrir con los requerimientos inherentes al protocolo TWI. Sus salidas deben manejar un tercer estado, siendo de colector o drenaje abierto. Las resistencias de *pull-up* imponen un nivel lógico alto en el bus cuando todas las salidas están en un tercer estado. Si en una o más salidas hay un nivel bajo, el bus va a reflejar ese nivel bajo. Con ello, se implementa una función AND alambrada, la cual es esencial para la operación del bus.

El protocolo TWI maneja un esquema Maestro-Esclavo, no obstante, cualquier dispositivo puede transmitir en el bus. Por ello, deben distinguirse los siguientes términos:

- **Maestro:** Dispositivo que inicia y termina una transmisión, también genera la señal de reloj (SCL).
- **Esclavo:** Dispositivo direccionado por un Maestro.
- **Transmisor:** Dispositivo que coloca los datos en el bus.
- **Receptor:** Dispositivo que lee los datos del bus.

6.3.1 Transferencias de Datos vía TWI

Cada bit transferido en la línea SDA debe acompañarse de un pulso en la línea SCL. El dato debe estar estable cuando la línea de reloj esté en alto, como se muestra en la figura 6.17(a). Las excepciones a esta regla se dan cuando se están generando las condiciones de INICIO y PARO.

El Maestro inicia y termina la trasmisión de datos, por lo tanto, es el Maestro quien genera las condiciones de INICIO y PARO, éstas son señalizadas cambiando el nivel en la línea SDA cuando SCL está en alto, se muestran en la figura 6.17 (b). Entre estas condiciones el bus se considera ocupado y ningún otro Maestro puede tomar control de él. Una situación especial se presenta si un nuevo INICIO es generado antes de un PARO, esta condición es referida como un INICIO REPETIDO y es utilizada por un Maestro cuando desea iniciar con una nueva transferencia, sin renunciar al control del bus. El INICIO REPETIDO se comporta como un INICIO y también es mostrado en la figura 6.17 (b).

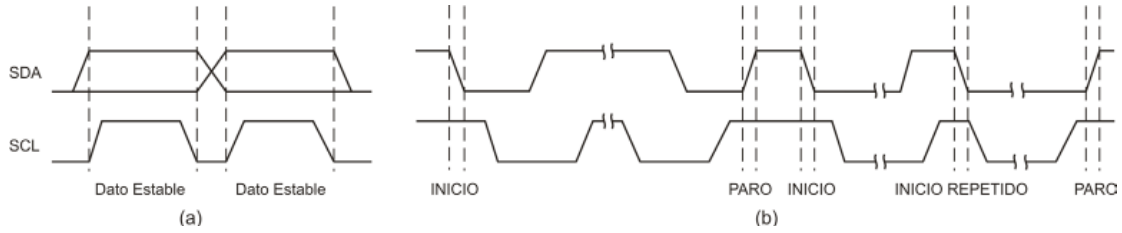


Figura 6.17 (a) Formato para datos válidos, y (b) Condiciones de INICIO, PARO e INICIO REPETIDO

6.3.1.1 Formato de los Paquetes de Dirección

Los paquetes de dirección, en el bus TWI, tienen una longitud de 9 bits, de los cuales, 7 bits son para la dirección de un Esclavo (iniciando con el MSB), 1 bit de control (R/W) y 1 bit de reconocimiento. El bit de control determina si se realiza una lectura (R = 1) o una escritura (W = 0). El bit de reconocimiento sirve para que el Esclavo direccionado dé respuesta al Maestro, colocando un 0 en la señal SDA durante el 9º ciclo de la señal SCL (respuesta referida como **ACK**). Si el Esclavo se encuentra ocupado o por alguna razón no da respuesta al Maestro, la señal SDA se va a mantener en alto (respuesta referida como **nACK**), entonces, el Maestro puede transmitir una condición de PARO o un INICIO REPETIDO, para iniciar nuevas transmisiones. A una trama de dirección con una petición de lectura se le refiere como SLA+R y con una petición de escritura es referida como SLA+W. En la figura 6.18 se muestra el formato de un paquete de dirección.

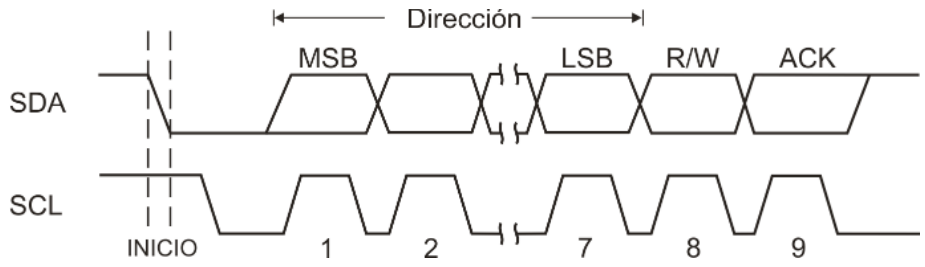


Figura 6.18 Formato de un paquete de dirección

Los Esclavos pueden tener cualquier dirección, excepto la 000 0000, la cual está reservada para una llamada general (GCA, *General Call Address*), es decir, del Maestro a todos los Esclavos. En una llamada general, el Maestro transmite el mismo mensaje a todos los Esclavos, los Esclavos deben responder con **ACK**. Las llamadas generales sólo son para peticiones de escritura (W = 0). Una petición de lectura en una llamada general provocaría una colisión en el bus, porque los Esclavos podrían transmitir datos diferentes.

6.3.1.2 Formato de los Paquetes de Datos

Los paquetes de datos también son de 9 bits, 8 bits para el dato (iniciando con el MSB) y un bit de reconocimiento. Durante las transferencias de datos, el Maestro genera la señal de reloj y las condiciones de INICIO y PARO. El receptor es el responsable de generar la señal de reconocimiento, poniendo en bajo a la señal SDA durante el 9º bit de SCL (**ACK**). En la figura 6.19 se muestra el formato de un paquete de datos. El Maestro no siempre es el transmisor, también un Esclavo puede serlo.

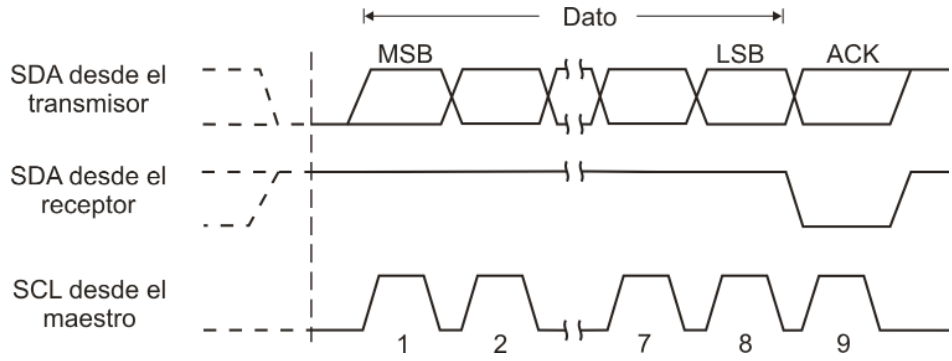


Figura 6.19 Formato de un paquete de datos

6.3.1.3 Transmisión Completa: Dirección y Datos

Un mensaje incluye una condición de INICIO, una SLA+R/W, uno o más paquetes de datos y una condición de PARO. Un mensaje vacío (INICIO seguido de un PARO) es ilegal y debe evitarse. La AND alambrada con las resistencias de *pull-up* sirve para coordinar la comunicación entre el Maestro y el Esclavo, si el Esclavo requiere un tiempo mayor para procesamiento entre bits, puede colocar un nivel bajo en SCL, retrasando con ello al bit siguiente. Esto no afecta el tiempo en alto de la señal SCL, el

cual es controlado por el Maestro. Por lo tanto, el Esclavo puede modificar la velocidad de transmisión de datos en el bus TWI.

En la figura 6.20 se ilustra la transmisión de un mensaje. Entre la SLA+R/W y la condición de PARO se pueden transferir muchos bytes de información, dependiendo del protocolo implementado por el software de la aplicación.

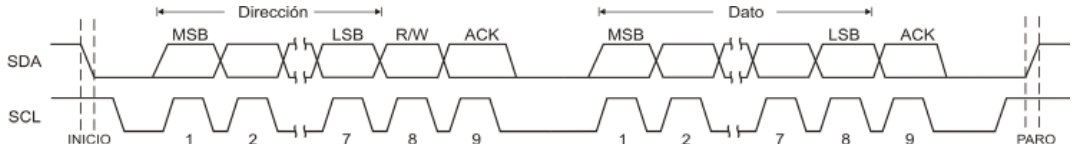


Figura 6.20 Transmisión típica de datos por TWI

6.3.2 Sistemas Multi-Maestros

El protocolo TWI permite el manejo de varios Maestros. Si dos o más Maestros intentan iniciar con una transmisión al mismo tiempo, pueden ocurrir 2 problemas que deben resolverse para que la comunicación proceda de manera normal. Estos problemas son:

- Sólo un Maestro puede concluir con la transmisión. Un proceso de selección, conocido como arbitración, define qué Maestro va a continuar con las transferencias.
- Los Maestros pueden generar señales de reloj que no están en fase. Se debe generar una señal de reloj sincronizada con el reloj de los diferentes Maestros, porque con ella va a realizarse el proceso de arbitración.

La AND alambrada es fundamental en la solución de estos problemas. En la figura 6.21 se muestra la señal de reloj en la línea SCL, resultante de la combinación de la señal de reloj de los Maestros A y B.

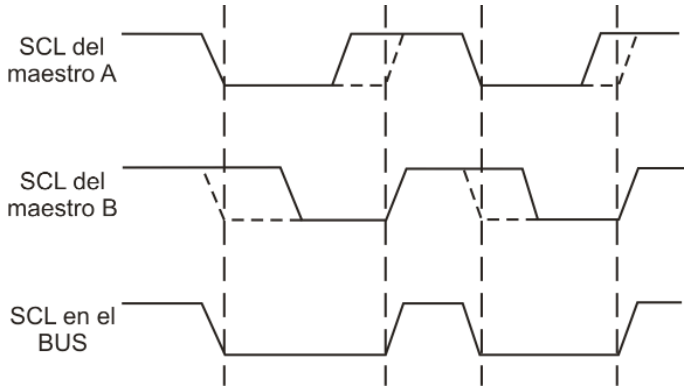


Figura 6.21 Sincronización de las señales de reloj de 2 Maestros

La AND ocasiona que en la señal de reloj resultante, el tiempo en alto sea más corto que en las señales de reloj de los Maestros. Esto porque corresponde con la intersección de las 2 señales. En consecuencia, el tiempo en bajo es más grande. Los Maestros que intenten transmitir en el bus se deben sincronizar con la señal de reloj resultante.

La arbitración la realizan los Maestros, monitoreando la línea SDA después de colocar un dato. Si el valor leído no coincide con el valor colocado por un Maestro, éste ha perdido la arbitración. Esto significa que un Maestro pierde la arbitración cuando coloca un 1 en el bus y en el mismo ciclo de reloj otro Maestro coloca un 0 (la AND hace que lea un 0), en la figura 6.22 se ilustra este proceso. Los Maestros que pierdan el proceso de arbitración inmediatamente deben conmutarse al modo de Esclavos, porque pueden ser direccionados por el Maestro ganador. El Maestro perdedor debe dejar la línea SDA en alto, pero puede continuar generando la señal de reloj hasta concluir con el paquete actual, de dirección o dato.

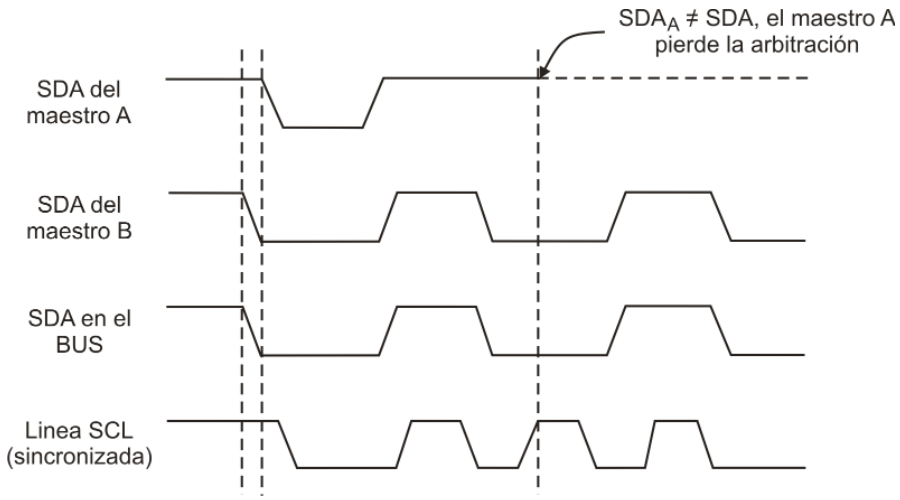


Figura 6.22 Proceso de arbitración ente 2 Maestros

La arbitración continúa hasta que sólo queda un Maestro y puede requerir de muchos bits. Si más de un Maestro direcciona al mismo Esclavo, la arbitración sigue en el paquete de datos. Existen algunas condiciones ilegales de arbitración, las cuales deben evitarse:

- Entre una condición de INICIO REPETIDO y el bit de un dato.
- Entre una condición de PARO y el bit de un dato.
- Entre una condición de INICIO REPETIDO y una condición de PARO.

Es responsabilidad del programador evitar que estas condiciones ocurran. Para ello, es conveniente que en un sistema multi-Maestros todas las transmisiones contengan el mismo número de paquetes de datos.

6.3.3 Organización de la Interfaz TWI

La interfaz TWI se compone de diferentes módulos, su organización se muestra en la figura 6.23. Todos los registros de la interfaz son accesibles por el núcleo AVR, desde el bus interno. Los registros se describen en la sección 6.3.4.

6.3.3.1 Terminales SCL y SDA

Son las terminales para la conexión con un bus TWI. Ambas incluyen un control de *slew-rate*, para cumplir con las especificaciones de la interfaz, y un filtro capaz de suprimir picos con una duración menor a 50 nS. El bus TWI requiere de 2 resistores externos de *pull-up*, es posible habilitar y usar los resistores de las terminales SCL y SDA, para no agregar hardware adicional.

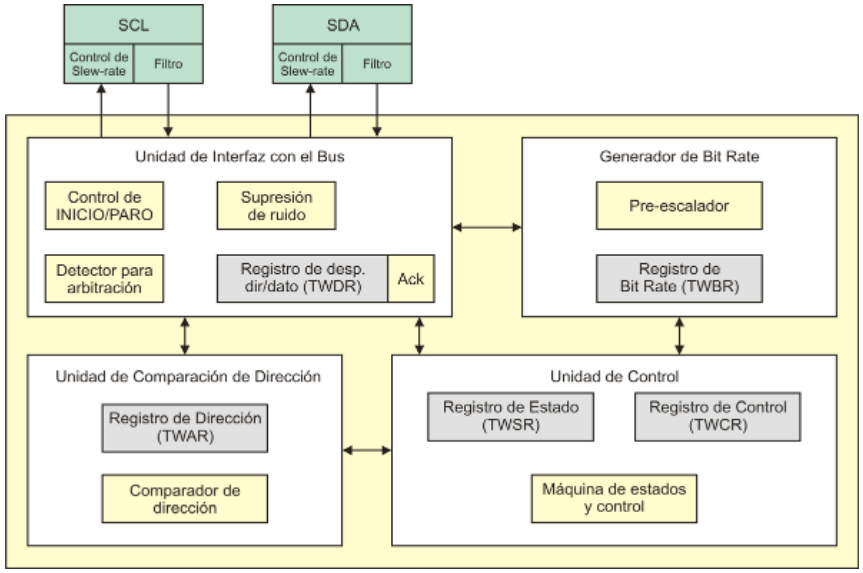


Figura 6.23 Organización de la Interfaz TWI

6.3.3.2 Generador de Bit Rate

Este módulo determina el periodo de la señal SCL, cuando el MCU está operando en modo Maestro. El periodo depende del valor del Registro de Bit Rate (**TWBR**) y de la configuración del pre-escalador, la cual se realiza con los bits **TWPS[1:0]** del Registro de Estado de la Interfaz TWI (**TWSR**). Considerando los valores del registro **TWBR** y de los bits **TWPS**, la frecuencia de SCL se genera de acuerdo con la ecuación:

$$frecuencia_{SCL} = \frac{frecuencia_{CPU}}{16 + 2(TWBR) \cdot 4^{TWPS}}$$

Si el MCU opera como Esclavo, no genera la señal SCL, sólo requiere operar a una frecuencia por lo menos de 16 veces la frecuencia de SCL.

Debe considerarse que un Esclavo puede prolongar el tiempo en bajo de la señal SCL, modificando el periodo promedio de la señal de reloj en el bus TWI.

6.3.3.3 Unidad de Interfaz con el Bus

Este módulo incluye al Registro de Desplazamiento para Datos y Direcciones (**TWDR**), en este registro se ubica la dirección o dato a ser transmitido, o la dirección o dato recibido. El bit Ack, adyacente al registro **TWDR** en la figura 6.23, es para el manejo del bit de reconocimiento. Este bit no es accesible por software, sin embargo, cuando se están recibiendo datos, puede ser puesto en alto o en bajo manipulando al Registro de Control de la interfaz TWI (**TWCR**). Cuando se está transmitiendo, el valor del bit AcK recibido se determina por los bits de estado en el registro **TWSR**.

El bloque para el Control de INICIO/PARO es responsable de generar y detectar las condiciones de INICIO, INICIO REPETIDO y PARO. Las condiciones de INICIO y PARO se detectan aun si el MCU está en alguno de los modos de reposo, "despertando" al MCU si fue direccionado por un Maestro.

El bloque Detector para arbitración incluye el hardware necesario para monitorear continuamente la actividad en el bus y determinar si una arbitración está en proceso, cuando la interfaz TWI ha iniciado una transmisión como Maestro. Si la interfaz TWI pierde el proceso de arbitración, debe informar a la Unidad de Control para que se muestre el estado y se realicen las acciones necesarias.

6.3.3.4 Unidad de Comparación de Dirección

El bloque Comparador de Dirección evalúa si la dirección recibida coincide con los 7 bits del Registro de Dirección de la Interfaz TWI (**TWAR**). Si el bit **TWGCE** del registro **TWAR** está en alto, el comparador está habilitado para reconocer llamadas generales en la interfaz TWI, por lo tanto, las direcciones entrantes también se deben comparar con la GCA. Ante una coincidencia, se informa a la Unidad de Control, para que realice las acciones correspondientes. La interfaz TWI puede o no reconocer su dirección, dependiendo de su configuración en el registro **TWCR**.

El comparador de dirección trabaja aun cuando el MCU se encuentra en modo de reposo, "despertando" al MCU si fue direccionado por un Maestro. Si ocurre otra interrupción mientras se realiza la comparación, la operación en la interfaz TWI es abortada y el recurso regresa a un estado ocioso. Por ello, antes de llevar al MCU a un modo de reposo, es conveniente únicamente activar la interrupción por la interfaz TWI.

6.3.3.5 Unidad de Control

La Unidad de Control monitorea los eventos que ocurren en el bus y genera respuestas de acuerdo con la configuración definida en el registro **TWCR**. Si un evento requiere atención, se pone en alto a la bandera **TWINT** y en el siguiente ciclo de reloj se actualiza al Registro de Estado (**TWSR**), mostrando el código que identifica al evento. El registro **TWSR** sólo tiene información relevante después de que la bandera **TWINT** es puesta en alto, en otras circunstancias, contiene un código de estado especial, indicando que

no hay información disponible. Tan pronto como la bandera **TWINT** es puesta en alto, la línea SCL es ajustada a un nivel bajo para permitir que la aplicación concluya con sus tareas por software, antes de continuar con las transmisiones TWI.

La bandera **TWINT** es puesta en alto ante las siguientes situaciones:

- Se transmitió una condición de INICIO o INICIO REPETIDO.
- Se transmitió una SLA+R/W.
- Se perdió una arbitración.
- La interfaz fue direccionada, con su dirección de Esclavo o por una GCA.
- Se recibió un dato.
- Recibió una condición de PARO o INICIO REPETIDO, mientras estaba direccionada como Esclavo.
- Ocurrió un error en el bus, debido a una condición ilegal de INICIO o PARO.

6.3.4 Registros para el Manejo de la Interfaz TWI

La interfaz TWI es manejada por 5 registros, para los datos (**TWDR**), para la dirección como esclavo (**TWAR**), para la velocidad de transmisión (**TWBR**), para el control (**TWCR**) y para el estado (**TWSR**). En esta sección se describe el papel de cada uno de estos registros.

El registro **TWDR** es el buffer para transmisión y recepción de datos. En modo transmisor, **TWDR** contiene el próximo dato a ser transmitido. En modo receptor, **TWDR** contiene el último dato recibido. Este registro sólo puede ser escrito después de la notificación de un evento, con la puesta en alto del bit **TWINT**, por ello, el registro no puede ser modificado por el usuario hasta después de que ocurra la primera interrupción. El contenido de **TWDR** permanece estable tan pronto como **TWINT** es puesto en alto.

	7	6	5	4	3	2	1	0	
0x03	Registro de datos de la interfaz TWI								**TWDR**

La interfaz TWI debe contar con una dirección para que pueda funcionar como Esclavo, aunque también puede atender a llamadas generales (GCA). En el Registro de Dirección (**TWAR**) se definen ambos parámetros, los bits de este registro son:

	7	6	5	4	3	2	1	0	
0x02	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	**TWAR**

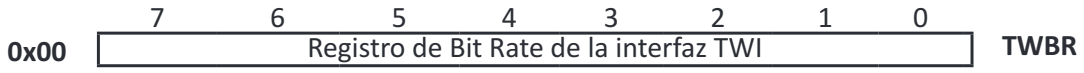
- **Bits 7 al 1 – TWA[6:0] Bits de dirección (como Esclavo)**

  En estos 7 bits se define la dirección de la interfaz TWI como Esclavo.

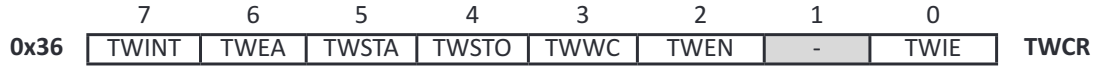
- **Bit 0 – TWGCE: Habilitador para reconocer la dirección de llamadas generales**

  Si está en alto, la interfaz TWI reconoce y puede dar respuesta a una GCA.

El registro base para definir la razón de transmisión de datos es el Registro de Bit Rate (**TWBR**). Con el valor de **TWBR** y con la selección en el pre-escalador, se define la frecuencia de la señal SCL, cuando el MCU trabaja como Maestro.



El Registro para el Control de la Interfaz TWI es el **TWCR**, con este registro es posible realizar diferentes tareas: Habilitar a la interfaz, aplicar una condición de INICIO, generar una condición de PARO y controlar las transferencias del bus al registro **TWDR**. Los bits del registro **TWCR** son:



- **Bit 7 – TWINT: Bandera de interrupción por la interfaz TWI**

  Su puesta en alto indica que en el bus ocurrió un evento que requiere atención por software. Se produce una interrupción si el habilitador global (bit **I** en **SREG**) y el habilitador individual (bit **TWIE**) están activados. Este bit debe ser limpiado por software escribiéndole un 1 lógico, aun cuando se configure su interrupción. Esto es necesario porque la señal SCL mantiene un nivel bajo mientras el bit **TWINT** está en alto, en espera de que por software se conozca al estado de la interfaz y se le dé respuesta. Una vez que la bandera se ha limpiado, ya no se puede tener acceso a los registros **TWAR**, **TWSR** y **TWDR**.

- **Bit 6 – TWEA: Habilitador de la generación del bit de reconocimiento (Ack)**

  Cuando este bit está en alto, la interfaz TWI genera el bit de reconocimiento si ocurre alguna de las siguientes situaciones:

  1. Se ha recibido la dirección del Esclavo.

  2. Se ha recibido una llamada general, estando el bit **TWGCE** de **TWAR** en alto.

  3. Se ha recibido un dato, en el modo Maestro Receptor o Esclavo Receptor.

  Si el bit **TWEA** tiene un nivel bajo, la interfaz TWI está virtualmente desconectada del bus.

- **Bit 5 – TWSTA: Bit para una condición de INICIO en el bus TWI**

  Se escribe un 1 en este bit para que llegue a ser un Maestro en el bus TWI. Si el bus está libre genera una condición de INICIO. Si el bus no está libre, la interfaz espera hasta detectar una condición de PARO, para después generar una nueva condición de INICIO, reintentando el acceso al bus como Maestro. El bit **TWSTA** debe limpiarse por software después de que la condición de INICIO ha sido transmitida.

- **Bit 4 – TWSTO: Bit para una condición de PARO en el bus TWI**

  En el modo Maestro, un 1 en este bit genera una condición de PARO en el bus TWI. El bit se limpia automáticamente por hardware después de la condición de PARO. En modo Esclavo, un 1 en **TWSTO** puede recuperar a la interfaz de una condición de error. No se genera una condición de PARO, pero la interfaz regresa a un modo de Esclavo sin direccionar, llevando a las líneas SCL y SDA a un estado de alta impedancia.

- **Bit 3 – TWWC: Bandera de colisión de escritura**

  Esta bandera se pone en alto si se intenta escribir en el registro **TWDR** cuando el bit **TWINT** está en bajo. La bandera se limpia cuando se escribe en **TWDR** después de que el bit **TWINT** es puesto en alto.

- **Bit 2 – TWEN: Habilitador de la interfaz TWI**

  Con este bit en alto se habilita la operación de la interfaz TWI. La interfaz toma el control de las terminales SCL y SDA, habilitando al limitador de *slew-rate* y al filtro eliminador de ruido. Si en el bit **TWEN** se escribe un 0, la interfaz se apaga y todas las transmisiones terminan.

- **Bit 1 – No está implementado.**

- **Bit 0 – TWIE: Habilitador de interrupción por TWI**

  Si este bit está en alto, y el bit **I** de **SREG** también, se genera una interrupción cuando la bandera **TWINT** es puesta en alto.

El estado de la interfaz TWI se conoce por medio del registro **TWSR**. En este registro también se define el factor de pre-escala, los bits de **TWSR** son:

	7	6	5	4	3	2	1	0	
0x01	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	**TWSR**

- **Bits 7 al 3 – TWS[7:3] Bits del estado de la interfaz TWI**

  En estos 5 bits se refleja el estado de la interfaz y del bus. Los diferentes códigos de estado se revisan en la siguiente sección, la cual describe los modos de transmisión

de la interfaz TWI. Por la presencia de los bits **TWPS[1:0]**, para la lectura correcta del estado se debe utilizar una máscara que únicamente conserve a los bits **TWS[7:3]**.

- **Bit 2 – No está implementado**
- **Bits 1 y 0 – TWPS[1:0]: Bits para la selección del factor de pre-escala en la interfaz TWI**

  En la tabla 6.9 se muestran los diferentes factores de pre-escala, con este factor y el valor del registro **TWBR**, se define la frecuencia a la que se va a generar la señal SCL cuando el MCU trabaja como Maestro.

Tabla 6.9 Factores de pre-escala, para definir la razón de transmisión por TWI

TWPS1	TWPS0	Factor de pre-escala
0	0	1
0	1	4
1	0	16
1	1	64

6.3.5 Modos de Transmisión y Códigos de Estado

La interfaz puede operar en 4 modos: Maestro Transmisor (MT), Esclavo Transmisor (ST), Maestro Receptor (MR) y Esclavo Receptor (SR). Una aplicación puede requerir más de un modo de operación. Por ejemplo, si un MCU va a manejar una memoria EEPROM vía TWI, con el modo MT puede escribir en la memoria y con el modo MR puede leer de ella. No obstante, si en la misma aplicación otro MCU direcciona al primero, éste también puede funcionar en los modos ST y SR. La aplicación decide cual es el modo más conveniente.

En los siguientes apartados se describen los modos de transmisión, listando los códigos de estado que se generan en cada transferencia. Un código de estado se genera cuando la bandera **TWINT** es puesta en alto, la actividad en el bus es detenida (señal SCL en bajo), por software debe leerse el estado de la interfaz y preparar la respuesta, en función de la aplicación. Con la respuesta lista debe limpiarse a la bandera **TWINT**, para continuar con las actividades del bus.

6.3.5.1 Modo Maestro Transmisor

Un MCU en el modo MT envía una cantidad de bytes a un MCU en el modo SR, esto se muestra en la figura 6.24. Un MCU entra al modo Maestro después de transmitir una condición de INICIO, posteriormente, el formato de la dirección determina si va a ser MT o MR. Para el modo MT se debe enviar una SLA+W (W = 0).

Una condición de inicio se genera escribiendo el siguiente valor en **TWCR:**

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
valor	1	X	1	0	X	1	0	X

Con el cual, la interfaz TWI es habilitada (**TWEN** = 1), se da paso a la condición de INICIO (**TWSTA** = 1) y se limpia la bandera **TWINT** (escribiéndole un 1). Con ello, si el bus está disponible se transmite la condición de inicio, la bandera **TWINT** es puesta nuevamente en alto y en los bits de estado, del registro **TWSR**, se obtiene el código 0x08.

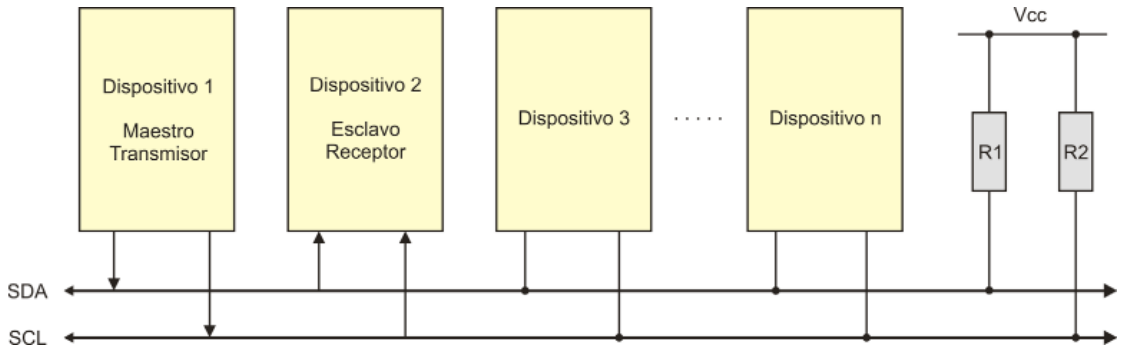


Figura 6.24 El dispositivo 1 transfiere datos en el modo Maestro Transmisor

En la tabla 6.10 se muestran los estados posibles en el modo MT, con una descripción del estado de la interfaz y las posibles acciones a seguir. Para todos los códigos se asume un enmascaramiento de los bits del pre-escalador (**TWPS**), de manera que se consideran como ceros.

Tabla 6.10 Estados posibles en el modo Maestro Transmisor

Código de Estado	Estado del bus y de la interfaz serial	Posibles acciones a realizar en la interfaz, en respuesta al estado
0x08	Una condición de INICIO ha sido transmitida	1. Transmitir SLA+W, recibir ACK o nACK
0x10	Una condición de INICIO REPETIDO ha sido transmitida	1. Transmitir SLA+W, recibir ACK o nACK 2. Transmitir SLA+R, conmutar la interfaz a MR
0x18	Se ha transmitido una SLA+W y recibido un ACK	1. Transmitir un byte de datos, recibir ACK o nACK 2. Transmitir un INICIO REPETIDO 3. Transmitir una condición de PARO 4. Transmitir una condición de PARO seguida de una condición de INICIO
0x20	Se ha transmitido una SLA+W y recibido un nACK	
0x28	Se ha transmitido un byte de datos y recibido un ACK	
0x30	Se ha transmitido un byte de datos y recibido un nACK	
0x38	Se ha perdido una arbitración al enviar una SLA o un byte de datos	1. Liberar al bus, únicamente limpiando a la bandera **TWINT** 2. Transmitir una condición de INICIO, cuando el bus esté libre

Para entrar al modo MT, el Maestro debe escribir una SLA+W en el registro **TWDR** para su transmisión. La transmisión de la SLA+W inicia cuando se limpia la bandera **TWINT**, para ello, en el registro **TWCR** debe escribirse el valor:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	1	X	0	0	X	1	0	X

Después de transmitir la SLA+W y recibir el bit de reconocimiento, la bandera **TWINT** es puesta en alto y en el registro **TWSR** se obtiene uno de los posibles estados: 0x18, 0x20 ó 0x38. Las acciones a realizar dependen del código de estado (tabla 6.10). Si la SLA+W se transmitió con éxito, el MCU Maestro está listo para enviar uno o varios datos. El dato a enviar debe colocarse en **TWDR**, mientras **TWINT** esté en alto. Después de escribir el dato, la bandera **TWINT** debe limpiarse escribiéndole un 1, en el registro **TWCR** debe escribirse el valor mostrado anteriormente. La actividad en el bus continúa y el dato escrito en **TWDR** es enviado. Este esquema se repite con cada uno de los datos. Cuando la bandera **TWINT** esté en alto, el dato debe ser escrito en **TWDR**, luego, debe limpiarse a la bandera para que el dato se envíe.

Una vez que se ha concluido con el envío de datos, el Maestro debe enviar una condición de PARO o una de INICIO REPETIDO. El valor del registro **TWCR** para una condición de PARO es:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	1	X	0	1	X	1	0	X

Un INICIO REPETIDO se solicita con el mismo valor que el de una condición de INICIO. Después de un INICIO REPETIDO (estado 0x10) la interfaz puede tener acceso al mismo o a otro Esclavo, sin transmitir una condición de PARO. El INICIO REPETIDO habilita a un Maestro a conmutar entre Esclavos o cambiar de Maestro Transmisor a Maestro Receptor, sin perder el control del bus.

6.3.1.1 Modo Maestro Receptor

Un MCU en el modo MR recibe una cantidad de bytes de un MCU en el modo ST, esto se muestra en la figura 6.25. Un MCU entra al modo Maestro después de transmitir una condición de INICIO, posteriormente, el formato de la dirección determina si va a ser MT o MR. Para el modo MR se debe enviar una SLA+R (R = 1).

Una condición de inicio se genera escribiendo el siguiente valor en **TWCR:**

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
valor	1	X	1	0	X	1	0	X

Con el cual, la interfaz TWI es habilitada (**TWEN** = 1), se da paso a la condición de INICIO (**TWSTA** = 1) y se limpia la bandera **TWINT** (escribiéndole un 1). Con ello, si el bus está disponible se transmite la condición de inicio, la bandera **TWINT** es puesta nuevamente en alto y en los bits de estado, del registro **TWSR**, se obtiene el código 0x08.

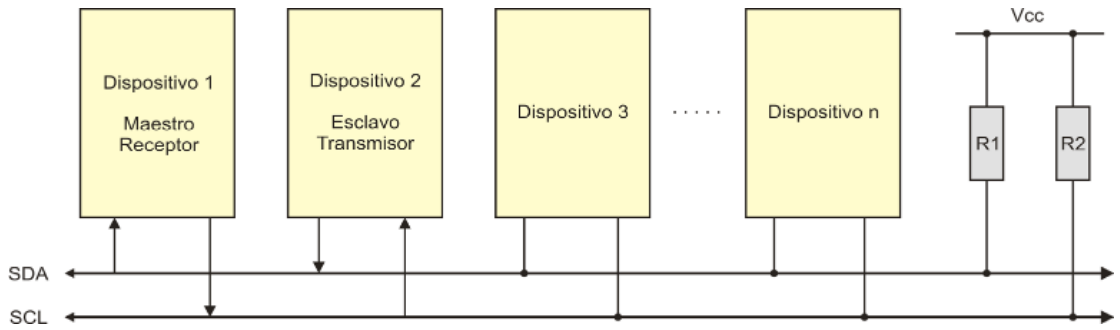


Figura 6.25 El dispositivo 1 recibe datos en el modo Maestro Receptor

En la tabla 6.11 se muestran los estados posibles en el modo MR, con una descripción del estado de la interfaz y las posibles acciones a seguir. Para todos los códigos se asume un enmascaramiento de los bits del pre-escalador (**TWPS**), de manera que se consideran como ceros.

Tabla 6.11 Estados posibles en el modo Maestro Receptor

Código de Estado	Estado del bus y de la interfaz serial	Posibles acciones a realizar en la interfaz, en respuesta al estado
0x08	Una condición de INICIO ha sido transmitida	1. Transmitir SLA+R, recibir ACK o nACK
0x10	Una condición de INICIO REPETIDO ha sido transmitida	1. Transmitir SLA+R, recibir ACK o nACK 2. Transmitir SLA+W, conmutar la interfaz a MT
0x38	Se perdió una arbitración al enviar una SLA+R o se envió un nACK	1. Liberar al bus, únicamente limpiando a la bandera **TWINT** 2. Transmitir una condición de INICIO, cuando el bus esté libre
0x40	Se ha transmitido una SLA+R y recibido un ACK	1. Recibir un byte de datos y dar respuesta con un ACK (**TWEA** = 1) 2. Recibir un byte de datos y dar respuesta con un nACK (**TWEA** = 0)
0x50	Se ha recibido un byte de datos y respondido con un ACK	
0x48	Se ha transmitido una SLA+R y recibido un nACK	1. Transmitir un INICIO REPETIDO 2. Transmitir una condición de PARO 3. Transmitir una condición de PARO seguida de una condición de INICIO
0x58	Se ha recibido un byte de datos y respondido con un nACK	

Para entrar al modo MR, el Maestro debe escribir una SLA+R en el registro **TWDR** para su transmisión. La transmisión de la SLA+R inicia cuando se limpia la bandera **TWINT**, para ello, en el registro **TWCR** debe escribirse el valor:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	1	X	0	0	X	1	0	X

Después de transmitir la SLA+R y recibir el bit de reconocimiento, la bandera **TWINT** es puesta en alto y en el registro **TWSR** se obtiene uno de los posibles estados: 0x38, 0x40 ó 0x48. Las acciones a tomar dependen del código de estado (tabla 6.11).

Si la SLA+R se transmitió con éxito, el MCU Maestro está listo para recibir uno o varios datos. Cada dato recibido se lee de **TWDR** cuando **TWINT** es puesta en alto, respondiendo con un bit ACK mientras se siguen recibiendo datos. La respuesta con reconocimiento y la limpieza de la bandera **TWINT** se realizan al escribir en el registro **TWCR** el siguiente valor:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	1	1	0	0	X	1	0	X

Una respuesta con un nACK le indica al Esclavo que ya no se van a recibir más datos. El valor a escribir en el registro **TWCR** es:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	1	0	0	0	X	1	0	X

Una vez que se ha concluido con la recepción de datos, el Maestro debe enviar una condición de PARO o una de INICIO REPETIDO. El valor del registro **TWCR**, para una condición de PARO, es:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
valor	1	X	0	1	X	1	0	X

Un INICIO REPETIDO se solicita con el mismo valor que el de una condición de INICIO. Después de un INICIO REPETIDO (estado 0x10) la interfaz puede tener acceso al mismo o a otro Esclavo, sin transmitir una condición de PARO. El INICIO REPETIDO habilita a un Maestro a conmutar entre Esclavos o cambiar de Maestro Receptor a Maestro Transmisor, sin perder el control del bus.

6.3.5.4 Modo Esclavo Receptor

Un MCU en el modo SR recibe una cantidad de bytes de un MCU en el modo MT, esto se muestra en la figura 6.26.

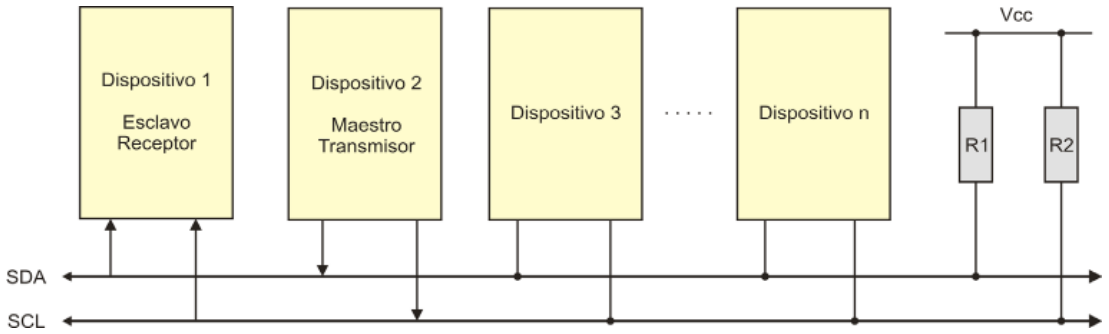


Figura 6.26 El dispositivo 1 recibe datos en el modo Esclavo Receptor

El MCU debe contar con una dirección a la cual debe responder como esclavo, esta dirección se define con los bits **TWA[6:0]**, los cuales corresponden con los 7 bits más significativos del registro **TWAR**. En el bit **TWGCE** (bit menos significativo de **TWAR**) se habilita al MCU para que también responda a una dirección de llamada general (GCA).

En el registro **TWCR** debe habilitarse a la interfaz TWI (**TWEN** = 1) y preparar una respuesta de reconocimiento (**TWEA** = 1), para ello, en este registro se debe escribir el valor:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	0	1	0	0	0	1	0	X

Con los registros **TWAR** y **TWCR** inicializados, la interfaz queda en espera de ser direccionada por su dirección de esclavo (o por una GCA, si fue habilitada) seguida por el bit de control del flujo de datos. La interfaz opera en el modo SR si el bit de control es 0 (*Write*), en caso contrario, entra al modo ST. Después de recibir su dirección, la bandera **TWINT** es puesta en alto y en los bits de estado del registro **TWSR** se refleja el código que determina las acciones a seguir por software. La interfaz también pudo ser llevada al modo SR si perdió una arbitración mientras estaba en modo Maestro.

En la tabla 6.12 se muestran los estados posibles en el modo SR, con una descripción del estado de la interfaz y las posibles acciones a seguir. Para todos los códigos se asume un enmascaramiento de los bits del pre-escalador (**TWPS**), de manera que se consideran como ceros.

Tabla 6.12 Estados posibles en el modo Esclavo Receptor

Código de Estado	Estado del bus y de la interfaz serial	Posibles acciones a realizar en la interfaz, en respuesta al estado
0x60	Se ha direccionado como Esclavo con una SLA+W y enviado un ACK	1. Recibir un byte de datos y regresar un ACK (**TWEA** = 1) 2. Recibir un byte de datos y regresar un nACK (**TWEA** = 0)
0x68	Se perdió una arbitración en una SLA+R/W como Maestro, se ha direccionado como esclavo con una SLA+W y enviado un ACK	
0x70	Se ha direccionado como Esclavo con una GCA y enviado un ACK	
0x78	Se perdió una arbitración en una SLA+R/W como Maestro, se ha direccionado como esclavo con una GCA y enviado un ACK	
0x80	Se ha recibido un byte de datos y respondido con un ACK, previamente se había direccionado con una SLA+W	
0x90	Se ha recibido un byte de datos y respondido con un ACK, previamente se había direccionado con una GCA	
0x88	Se ha recibido un byte de datos y respondido con un nACK, previamente se había direccionado con una SLA+W	1. Conmutar a un modo de Esclavo no direccionado, desactivando la interfaz para no reconocer su propia SLA o la GCA (**TWEA** = 0) 2. Conmutar a un modo de Esclavo no direccionado, capaz de reconocer su propia SLA o la GCA (**TWEA** = 1) 3. Conmutar a un modo de Esclavo no direccionado, desactivando la interfaz para no reconocer su propia SLA o la GCA y enviar un bit de inicio, cuando el bus esté disponible 4. Conmutar a un modo de Esclavo no direccionado, capaz de reconocer su propia SLA o la GCA y enviar un bit de inicio, cuando el bus esté disponible
0x98	Se ha recibido un byte de datos y respondido con un nACK, previamente se había direccionado con una GCA	
0xA0	Se ha recibido una condición de PARO o de INICIO REPETIDO, mientras estaba direccionado como Esclavo	

Si el bit **TWEA** es reiniciado durante una transferencia, la interfaz coloca un nACK en SDA después de recibir el próximo dato. Esto se puede hacer para que un Esclavo indique que no le es posible recibir más datos. Con el bit **TWEA** se puede aislar temporalmente a la interfaz del bus, con un 0 no reconoce su SDA o la GCA, pero el monitoreo continúa realizándose, de manera que puede ocurrir un reconocimiento tan pronto como **TWEA** es puesta en alto.

El reloj del sistema es omitido en algunos modos de reposo, sin embargo, si el bit **TWEA** está en alto, la interfaz TWI va a reconocer su SLA o a la GCA utilizando al reloj del bus (SCL) como mecanismo para su sincronización. Con ello, la interfaz "despierta" al MCU aunque la señal SCL se mantenga en bajo en espera de que la bandera **TWINT** sea limpiada. Esto hace que en el registro **TWDR** no se refleje el último byte presente en el bus. Una vez que el MCU está activo, las transferencias siguientes emplean al reloj del sistema. Si el MCU tiene un tiempo de ajuste largo, durante ese tiempo la señal SCL se mantiene en bajo y se detienen las transferencias en el bus.

6.3.5.4 Modo Esclavo Transmisor

Un MCU en el modo ST transmite una cantidad de bytes a un MCU en el modo MR, esto se muestra en la figura 6.27.

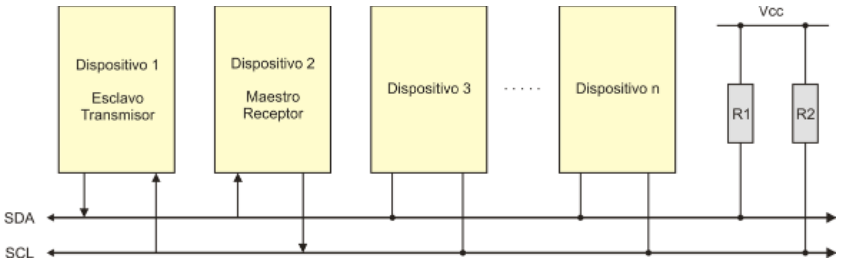


Figura 6.27 El dispositivo 1 envía datos en el modo Esclavo Transmisor

El MCU debe contar con una dirección a la cual va a responder como esclavo, esta dirección se define con los bits **TWA[6:0]**, los cuales corresponden con los 7 bits más significativos del registro **TWAR**. En el bit **TWGCE** (bit menos significativo de **TWAR**) se habilita al MCU para que también responda a una dirección de llamada general (GCA).

En el registro **TWCR** debe habilitarse a la interfaz TWI (**TWEN** = 1) y preparar una respuesta de reconocimiento (**TWEA** = 1), para ello, en este registro se debe escribir el valor:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Valor	0	1	0	0	0	1	0	X

Con los registros **TWAR** y **TWCR** inicializados, la interfaz queda en espera de ser direccionada por su dirección de esclavo (o por una GCA, si fue habilitada, aunque en una llamada general no se solicita a los esclavos transmitir datos, porque se provocaría una colisión en el bus) seguida por el bit de control del flujo de datos. La interfaz opera en el modo ST si el bit de control es 1 (*Read*), en caso contrario, entra al modo SR. Después de recibir su dirección, la bandera **TWINT** es puesta en alto y en los bits de

estado del registro **TWSR** se refleja el código que determina las acciones a seguir por software. La interfaz también pudo ser llevada al modo ST si perdió una arbitración mientras estaba en modo Maestro.

En la tabla 6.13 se muestran los estados posibles en el modo ST, con una descripción del estado de la interfaz y las posibles acciones a seguir. Para todos los códigos se asume un enmascaramiento de los bits del pre-escalador (**TWPS**), de manera que se consideran como ceros.

Si el bit **TWEA** es limpiado, la interfaz transmite el último dato y el MCU conmuta a un esclavo sin direccionar. En los bits de estado se puede obtener 0xC0 ó 0xC8, dependiendo de si el Maestro Receptor transmitió un ACK o un nACK después de recibir el último dato. Si el Maestro intenta continuar con las transferencias, éstas van a ignorarse. El Maestro podría demandar más datos generando señales de ACK, en cuyo caso recibiría 1's en la línea de datos.

Tabla 6.13 Estados posibles en el modo Esclavo Transmisor

Código de Estado	Estado del bus y de la interfaz serial	Posibles acciones a realizar en la interfaz, en respuesta al estado
0xA8	Se ha direccionado como Esclavo con una SLA+R y enviado un ACK	1. Transmitir un byte de datos y recibir un ACK (**TWEA** = 1) 2. Transmitir un byte de datos y recibir un nACK (**TWEA** = 0)
0xB0	Se perdió una arbitración en una SLA+R/W como Maestro, se ha direccionado como esclavo con una SLA+R y enviado un ACK	
0xB8	Se transmitió un byte de datos en **TWDR** y se recibió un ACK	
0xC0	Se transmitió un byte de datos en **TWDR** y se recibió un nACK	1. Conmutar a un modo de Esclavo no direccionado, desactivando la interfaz para no reconocer su propia SLA o la GCA (**TWEA** = 0) 2. Conmutar a un modo de Esclavo no direccionado, capaz de reconocer su propia SLA o la GCA (**TWEA** = 1) 3. Conmutar a un modo de Esclavo no direccionado, desactivando la interfaz para no reconocer su propia SLA o la GCA y enviar un bit de inicio, cuando el bus esté disponible 4. Conmutar a un modo de Esclavo no direccionado, capaz de reconocer su propia SLA o la GCA y enviar un bit de inicio, cuando el bus esté disponible
0xC8	Se transmitió el último byte de datos en **TWDR** (TWEA = 0) y se recibió un ACK	

Con el bit **TWEA** se puede aislar temporalmente a la interfaz del bus, con un 0 no reconoce su SDA o la GCA, pero el monitoreo continúa realizándose, de manera que puede ocurrir un reconocimiento tan pronto como **TWEA** es puesta en alto.

6.3.5.5 Estados Misceláneos

La interfaz TWI incluye 2 códigos que no corresponden con alguno de los 4 modos de operación.

Estos estados se describen en la tabla 6.14, en donde se asume un enmascaramiento de los bits del pre-escalador (**TWPS**), de manera que se consideran como ceros.

Tabla 6.14 Estados misceláneos

Código de Estado	Estado del bus y de la interfaz serial	Posibles acciones a realizar en la interfaz, en respuesta al estado
0xF8	No hay información relevante disponible, **TWINT** = 0	1. Esperar o proceder con la siguiente transferencia
0x00	Error en el bus, debido a una condición ilegal de INICIO o PARO	1. Sólo el hardware interno es afectado. El bit **TWSTO** debe ser puesto en alto, pero no se envía una condición de PARO en el bus. El bus es liberado y el bit **TWSTO** es limpiado.

Un error en el bus puede deberse a una posición ilegal durante la transferencia de un byte de dirección, un byte de datos o un bit de reconocimiento. **TWINT** es puesta en alto cuando ocurre un error. Para recuperar al bus, la bandera **TWSTO** debe ser puesta en alto y la bandera **TWINT** en bajo. Con esto, la interfaz entra a un modo de Esclavo no direccionado y limpia a la bandera **TWSTO**. Las líneas del bus, SDA y SCL, son liberadas y no se transmite una condición de paro.

6.3.6 Ejemplos de Uso de la Interfaz TWI

Los modos de operación de la interfaz TWI le proporcionan a los AVR una capacidad extraordinaria para el desarrollo de sistemas “inteligentes” conectados como nodos en una red de 2 hilos, en donde cada MCU puede conmutarse, en tiempo de ejecución, entre los modos Maestro y Esclavo. En esta sección se muestran 2 ejemplos del uso de la interfaz TWI, codificados en lenguaje C. No obstante, las aplicaciones posibles van mucho más allá de los ejemplos acá descritos.

Ejemplo 6.9 Repita el ejemplo 6.8 utilizando la interfaz TWI. Se trata de un maestro y 3 esclavos, el Maestro tiene 2 arreglos de interruptores (uno con 8 y otro con 2 interruptores) y un botón. El arreglo de 2 interruptores es para seleccionar un Esclavo y el de 8 para introducir un dato. Con ello, cada vez que se presiona al botón, debe enviarse el dato al Esclavo seleccionado.

Las direcciones para los esclavos son 1, 2 y 3, y utilice la dirección 0 para una llamada general. Cuando un Esclavo reciba un dato, lo debe mostrar en su puerto D.

El hardware para este problema se muestra en la figura 6.28, el Maestro incluye los interruptores y el botón, cada Esclavo incluye su conjunto de LEDs. Además de incluir los resistores de *Pull-Up* con un valor de 2 Kohms.

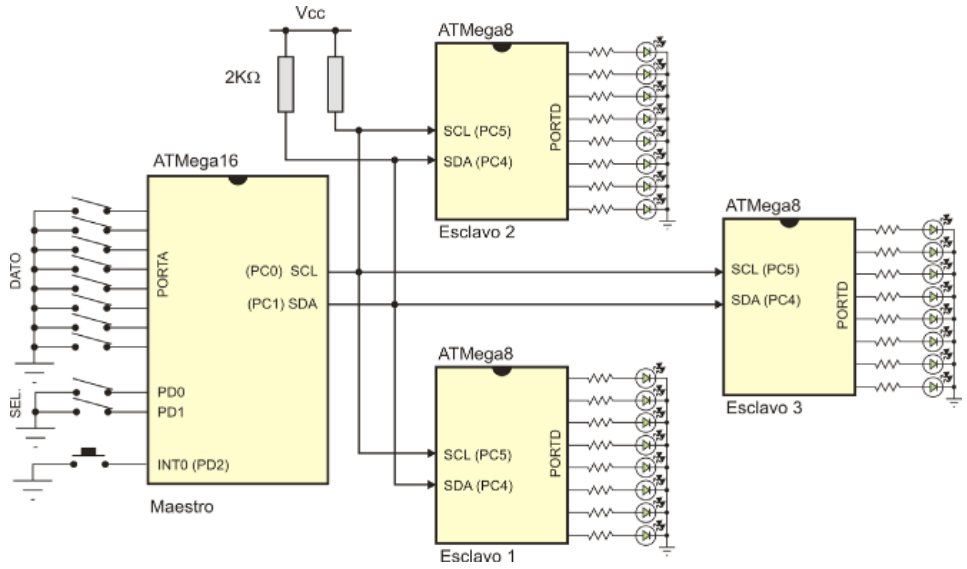


Figura 6.28 Envío de información de un Maestro a 3 Esclavos por TWI

En el Maestro se utiliza la interrupción externa (INT0), en la ISR de la INT0 se habilita a la interfaz TWI con su interrupción y se transmite una condición de inicio. Posteriormente, toda la actividad del bus es manejada en la ISR de la interfaz TWI, en donde se evalúan los bits de estado, para dar respuesta a sus diferentes valores.

El código en lenguaje C para el Maestro es:

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

ISR(INT0_vect) {

// Habilita a la interfaz TWI, su interrupción y transmite una condición
de inicio
       TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN) | (1 << TWIE);
}

ISR(TWI_vect) {
unsigned  char  estado, esclavo, dato;

estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI

switch(estado) {                          // Actúa según el estado
   case  0x08:                            // Se transmitió la condición de
                                          // inicio
               esclavo = PIND & 0x03;     // Lee la dirección del esclavo
               esclavo = esclavo << 1;    // Ajusta a los 7 MSB y
                                          // compone la SLA+W
```

```
            TWDR = esclavo;           // Ubica la SLA+W
            TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWIE);
            break;
   case  0x18:                        // Transmitió la SLA+W con
                                      // reconocimiento
            dato = PINA;
            TWDR = dato;              // Prepara el dato a enviar
            TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWIE);
            break;
   default:  //No hubo ACK con la SLA+W o trasmitió el dato con un ACK o un nACK
             // Genera la condición de paro
      TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN) | (1 << TWIE);
            break;
   }
}

int  main() {                         // Programa principal

DDRA = 0x00;                          // Puerto A como entrada
DDRD = 0x00;                          // Puerto D como entrada
PORTA = 0xFF;                         // Resistores de pull-up en el puerto A
PORTD = 0x07;                         // Resistores de pull-up en el puerto D

TWBR = 0x02;                          // Para definir la frecuencia de reloj
TWSR = 0x00;                          // Pre-escalador en 0(factor de pre-escala = 1)
                                      // para una frecuencia de 31.25 KHz
                                      // (fosc=1 MHz)
MCUCR = 0x02;                         // INT0 por flanco de bajada
GICR = 0x40;                          // Habilita la INT0

sei();                                // Habilitador global de interrupciones

while(1)                              // Ocioso en el lazo infinito
       asm("nop");
}
```

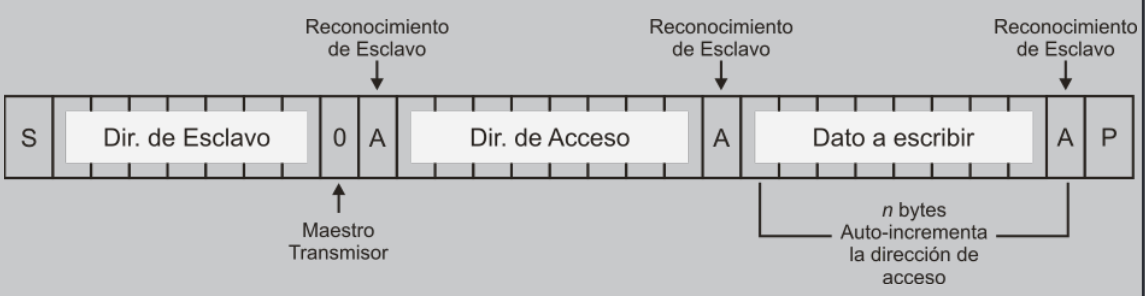
En los esclavos también se utiliza la interrupción por TWI, en su ISR se conoce el estado del bus, al cual se le da respuesta. Cada esclavo debe tener su propia dirección, el código siguiente es para un esclavo con dirección 0x01, para cambiar la dirección sólo se debe modificar la constante.

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

#define DIR_ESLVO   0x01

ISR(TWI_vect) {
unsigned  char  dato, estado;

estado = TWSR & 0xFC;                 // Obtiene el estado de la interfaz TWI

switch(estado) {
```

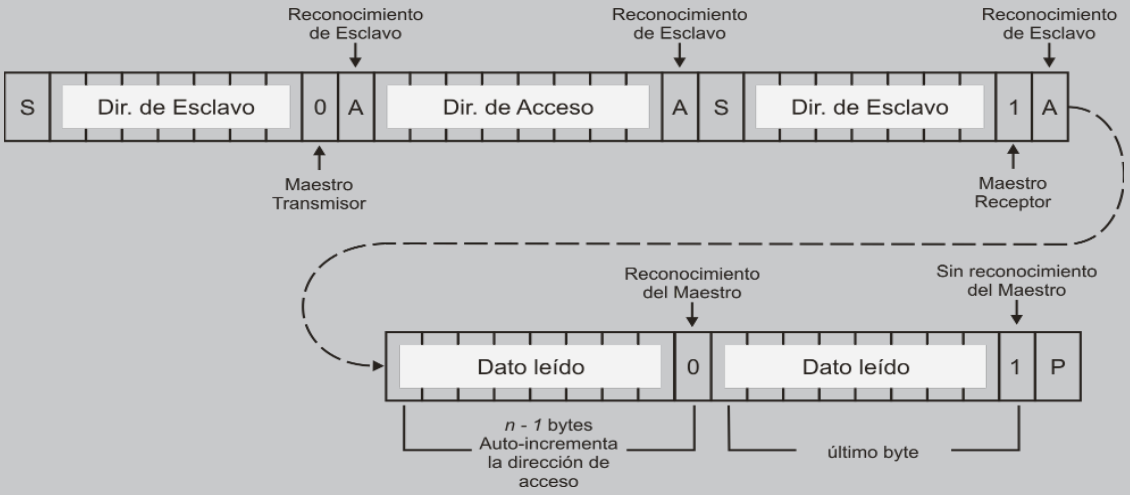
```
        case 0x60:                  // Esclavo direccionado con su SLA o la GCA
        case 0x70:    TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) |
                      (1 << TWIE);
                      break;
        case 0x80:                  // Recibió un dato, previamente direccionado
        case 0x90:                  // con su SLA o  la GCA
                      dato = TWDR;  // Obtiene el dato recibido
                      PORTD = dato;
                      TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) |
                      (1 << TWIE);
                      break;
        default:                    // Situación no esperada
                      TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN) |
                      (1 << TWIE);
                      break;
        }
}


int  main() {                       // Programa principal
unsigned  char  dir;


DDRD = 0xFF;                        // Puerto D como salida

dir = DIR_ESLVO << 1;               // Asigna la dirección del esclavo
dir = dir | 0x01;                   // habilitado para reconocer a la GCA
TWAR = dir;

// Habilita a la interfaz TWI, su respuesta de reconocimiento y su interrupción

TWCR = (1 << TWEA) | (1 << TWEN) | (1 << TWIE);

sei();                              // Habilitador global de interrupciones
while(1)                            // Ocioso en el lazo infinito
        asm("nop");
}
```

Se observa que la difusión hacia todos los esclavos es parte de la interfaz.

Ejemplo 6.10 El circuito PCF8570 es una memoria RAM de 256 x 8 bits, con interfaz I^2C y dirección configurable entre 0x50 y 0x57 (con 3 terminales). Esta memoria trabaja en los modos Esclavo Receptor (para escritura de datos) y Esclavo Transmisor (para lectura de datos). En la figura 6.29 (a) se muestra como escribir datos y en la 6.29 (b) puede verse como leerlos.



(a) El Maestro transmite y el esclavo recibe



b) El Maestro lee después de detectar la presencia del Esclavo

Figura 6.29 (a) Escritura de datos (b) Lectura de datos

Considerando una memoria PCF8570 con la dirección de Esclavo 0x50, conectada a la interfaz TWI de un AVR, realice una función que escriba *n* bytes en la memoria, recibiendo un arreglo con los datos, la cantidad de datos a escribir y la dirección donde deben iniciar las escrituras; y una función que lea *n* bytes de la memoria, recibiendo un arreglo para colocar los datos, la cantidad de datos a leer y la dirección donde deben iniciar las lecturas.

Ambas funciones regresan 0x01 si el acceso a la memoria se realizó con éxito y 0x00 si no hubo reconocimiento del esclavo o si ocurrió una falla durante el proceso. No se configura la frecuencia, para el programa completo debe tomarse en cuenta que la memoria puede operar con una frecuencia máxima de 100 KHz y tampoco se utilizan interrupciones.

El código de la función para la escritura de datos en la RAM es:

```
char  escribe_RAM(char datos[], unsigned char n, unsigned char dir) {
unsigned char i, estado;
```

```
// Habilita a la interfaz TWI y transmite una condición de INICIO
TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
while(!(TWCR & (1 << TWINT)));          // Espera fin de condición de INICIO
estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI
if( estado != 0x08) {                   // Falló al intentar tomar el bus
      TWCR = (1 << TWINT) | (1 << TWEN);      // Limpia la bandera
      return  0x00;                     // Regresa sin éxito
}

TWDR = 0xA0;                            // Ubica la SLA+W (0x50 << 1 + 0)
TWCR = (1 << TWINT) | (1 << TWEN);      // Transmite la SLA+W
while(!(TWCR & (1 << TWINT)));
estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI
if( estado != 0x18) {                   // Falló al enviar la SLA+W
      TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
                                        // Condición de PARO
      return  0x00;                     // Regresa sin éxito
}

TWDR = dir;                             // Transmite la dirección de acceso
TWCR = (1 << TWINT) | (1 << TWEN);
while(!(TWCR & (1 << TWINT)));          // Espera finalice el envío
estado = TWSR & 0xFC;     // Obtiene el estado de la  interfaz TWI
if( estado != 0x28) {                   // Falló al enviar un dato
      TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
                                        // Condición de PARO
      return  0x00;                     // Regresa sin éxito
}

for( i = 0; i < n; i++ ) {                    // Transmite los datos
      TWDR = datos[i];                        // Ubica dato a transmitir
      TWCR = (1 << TWINT) | (1 << TWEN);      // Inicia el envío
      while(!(TWCR & (1 << TWINT)));          // Espera finalice el envío
      estado = TWSR & 0xFC; // Obtiene el estado de  la interfaz
      if( estado != 0x28) {                   // Falló al enviar un dato
            TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);// PARO
            return  0x00;                     // Regresa sin éxito
      }
}
TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);     // Condición de PARO
return  0x01;                           // Regresa con éxito
}
```

El código de la función para la lectura de datos en la RAM es:

```
char   lee_RAM(char datos[], unsigned char n, unsigned char dir) {
unsigned char i, estado;

// Habilita a la interfaz TWI y transmite una condición de INICIO
TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
while(!(TWCR & (1 << TWINT)));    // Espera fin de condición de INICIO
estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI
if( estado != 0x08) {             // Falló al intentar tomar el bus
```

```
      TWCR = (1 << TWINT) | (1 << TWEN);// Limpia la bandera
      return  0x00;                     // Regresa sin éxito
}
TWDR = 0xA0;                        // Ubica la SLA+W (0x50 << 1 + 0)
TWCR = (1 << TWINT) | (1 << TWEN);      // Transmite la SLA+W
while(!(TWCR & (1 << TWINT)));
estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI
if( estado != 0x18) {                   // Falló al enviar la SLA+W
      TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
                                        // Condición de PARO
      return  0x00;                     // Regresa sin éxito
}

TWDR = dir;                         // Transmite la dirección de acceso
TWCR = (1 << TWINT) | (1 << TWEN);
while(!(TWCR & (1 << TWINT)));          // Espera finalice el envío
estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI
if( estado != 0x28) {                   // Falló al enviar un dato
      TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
                                        // Condición de PARO
      return  0x00;                     // Regresa sin éxito
}
// Transmite una condición de INICIO (INICIO REPETIDO)
TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
while(!(TWCR & (1 << TWINT)));    // Espera fin de condición de INICIO
estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz TWI
if( estado != 0x10) {             // Falló con el inicio repetido
      TWCR = (1 << TWINT) | (1 << TWEN);      // Limpia la bandera
      return  0x00;                     // Regresa sin éxito
}

TWDR = 0xA1;                            // Ubica la SLA+R (0x50 << 1 + 1)
TWCR = (1 << TWINT) | (1 << TWEN);      // Transmite la SLA+R
while(!(TWCR & (1 << TWINT)));
estado = TWSR & 0xFC;         // Obtiene el estado de la interfaz TWI
if( estado != 0x40) {                   // Falló al enviar la SLA+R
      TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
                                       // Condición de PARO
      return  0x00;                     // Regresa sin éxito
}
for( i = 0; i < n - 1; i++) {           // Recibe n - 1 datos
      TWCR = (1 << TWINT) | (1 << TWEA)| (1 << TWEN);
                                        // Habilita respuesta
      while(!(TWCR & (1 << TWINT)));    // Espera dato del esclavo
      estado = TWSR & 0xFC;      // Obtiene el estado de la interfaz
      if( estado != 0x50) {             // Falló al recibir un dato
            TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);// PARO
            return  0x00;               // Regresa sin éxito
      }
      datos[i] = TWDR;                  // Dato recibido con éxito
}
```

```
TWCR = (1 << TWINT) | (1 << TWEN);       // Recibe el último dato
                                        // (sin el bit de reconocimiento)
while(!(TWCR & (1 << TWINT)));           // Espera dato del esclavo
estado = TWSR & 0xFC;              // Obtiene el estado de la interfaz
if( estado != 0x58) {                    // Falló al recibir un dato
       TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
                                         // Condición de PARO
       return  0x00;                     // Regresa sin éxito
}
datos[i] = TWDR;                         // Último dato recibido con éxito

TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);// Condición de PARO
return  0x01;                                    // Regresa con éxito}
}
```

Un problema con las funciones del ejemplo 6.10 es que cuando no tienen éxito regresan con el valor 0x00, sin importar en que etapa del proceso se encontraban. Las funciones pueden mejorar si se establece un código para cada uno de los errores más frecuentes, de manera que por software se conozca el motivo del final de las transferencias.

6.4 Ejercicios

En los siguientes ejercicios se combina el uso de las interfaces seriales con los recursos revisados en los capítulos anteriores. Pueden ser resueltos en lenguaje C o ensamblador.

1. Haga una comparativa de las 3 interfaces seriales revisadas en el presente capítulo, mostrando las ventajas y desventajas que cada una de ellas tiene.

2. Realice un **sistema para el control de la temperatura ambiente**, generando una señal PWM de 10 bits para activar un ventilador de DC. La temperatura de referencia se debe recibir por el puerto serie (USART), como un número entero (1 byte). Si la diferencia de la temperatura actual (obtenida de un sensor LM35) con la temperatura de referencia es mayor o igual a 3 °C, la señal de activación del ventilador debe tener un ciclo útil del 100 %. Si la diferencia está entre 0 °C y 3 °C, al ancho de pulso debe ser proporcional a esta diferencia. Cuando la temperatura actual está por debajo de la referencia, el ventilador debe estar apagado. La temperatura de referencia puede arribar en cualquier momento.

3. Construya una **marquesina para exhibir mensajes, con 5 matrices de 5 x 7 LEDs**. Utilice una red SPI con 1 Maestro y 5 Esclavos, todos ellos con base en microcontroladores ATMega8. Que cada matriz sea manejada por un Esclavo, quienes deben recibir el

código ASCII del carácter a mostrar, generando los puntos en su respectiva matriz, con base en una tabla de constantes. El usuario va a enviar la cadena a exhibir y su tamaño desde una PC al Maestro, a través de la USART. Para un desplazamiento del mensaje de derecha a izquierda, el Maestro continuamente debe modificar el carácter de cada Esclavo. Un Esclavo básicamente va a funcionar como un driver para una matriz de 5 x 7 LEDs, con una interfaz SPI para recibir el carácter a exhibir.

- Proponer el hardware.
- Desarrollar el programa del maestro y de los esclavos (es el mismo para los 5 esclavos).

4. Repita el ejercicio anterior, pero utilice la interfaz TWI en lugar de emplear a la interfaz SPI. ¿Cuál de las dos interfaces (SPI y TWI) considera es más adecuada para este problema? Justifique su respuesta.

5. Se requiere de un **sistema para votaciones**, capaz de manejar hasta 100 votantes. Diseñe el sistema de manera que los votantes dispongan de un circuito con un indicador luminoso (LED) y 5 botones. Con el indicador luminoso encendido, los votantes deben elegir alguna opción, presionando uno de los botones.

   Organice el sistema con base en una red TWI, donde los circuitos de votación sean los Esclavos y que el Maestro sea el colector de los votos. Después de cada elección, el maestro debe enviar 5 pares ordenados a través de su USART. Cada par especifica cuantos votos se tuvieron en cada una de las opciones. Proponga el hardware y desarrolle el software, del maestro y de los esclavos.

6. El circuito DS1307 es un Reloj/Calendario de Tiempo Real, con interfaz I^2C y dirección de Esclavo 0x68 (1101000). El circuito trabaja en los modos Esclavo Receptor (para configurar la fecha y la hora) y Esclavo Transmisor (para leer la fecha y la hora), siguiendo secuencias similares a las mostradas en las figuras 6.29 (a) y en la 6.29 (b). En la tabla 6.15 se muestra la organización de los registros para la fecha y la hora, puede verse que la información está en BCD.

   Escriba las funciones para:

   - Ajustar la hora, recibiendo un arreglo de caracteres con los segundos, minutos y hora (esc_hora(**unsigned char** hora[])).
   - Leer la hora, colocando los datos en un arreglo (lee_hora(**unsigned char** hora[])).
   - Ajustar la fecha, recibiendo un arreglo de caracteres con el día de la semana, día del mes, mes y año (esc_fecha(**unsigned char** fecha[])).

- Leer la fecha, colocando los datos en un arreglo (lee_fecha(**unsigned char** fecha[])).

Tabla 6.15 Registros para mantener la fecha y hora

Dirección	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Función	Rango
00 h	CH	Decenas de segundos			Unidades de segundos				Segundos	00-59
01 h	0	Decenas de minutos			Unidades de minutos				Minutos	00-59
02 h	0	12	PM /AM	Dec. de hora	Unidades de horas				Horas	1-12 + AM/PM
		24	Dec. de hora		Unidades de horas					00-23
03 h	0	0	0	0	0	Día (semana)			Día (semana)	01-07
04 h	0	0	Dec. día (mes)		Unidades del día (mes)				Día (mes)	01-31
05 h	0	0	0	D. Mes	Unidades del mes				Mes	01-12
06 h	Decenas del año				Unidades del año				Año	00-99
07 h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	-
08h-3Fh									RAM 56 x 8	00h-FFh

Respecto a la tabla 6.15, con el registro de control se configura al hardware para generar una señal cuadrada a diferentes frecuencias. En las funciones a desarrollar, ignore el registro y la memoria RAM disponible. El bit CH en la dirección 00h sirve para detener momentáneamente al reloj (*clock halt*), el valor de este bit no se debe modificar. Las funciones sólo se deben encargar del acceso al bus I^2C, las validaciones antes de una escritura o interpretaciones después de una lectura quedan fuera del contexto del problema, serían parte del programa principal..

7. Recursos Especiales

En este capítulo se describen 4 recursos de los microcontroladores AVR que pueden ser utilizados para complementar la funcionalidad de algunas aplicaciones, se trata del Perro Guardián (WDT, *watchdog timer*), la sección de arranque en la memoria de programa, los *Bits de Configuración y Seguridad*, y la interfaz JTAG, aunque ésta no se incluye en los ATMega8, sólo en los ATMega16.

7.1 Watchdog Timer de un AVR

El WDT es un temporizador que reinicia al MCU con su desbordamiento, es manejado por un oscilador interno, el cual tiene una frecuencia de 1 MHz cuando el MCU está alimentado con 5 V. En la figura 7.1 se muestra la organización del WDT, puede verse como a través de un pre-escalador es posible conseguir diferentes intervalos de tiempo, el intervalo se define con los bits **WDP[2:0]** del Registro de Control del WDT (**WDTCR**, *Watchdog Timer Control Register*). En la tabla 7.1 se muestran los intervalos de tiempo para las diferentes opciones del pre-escalador.

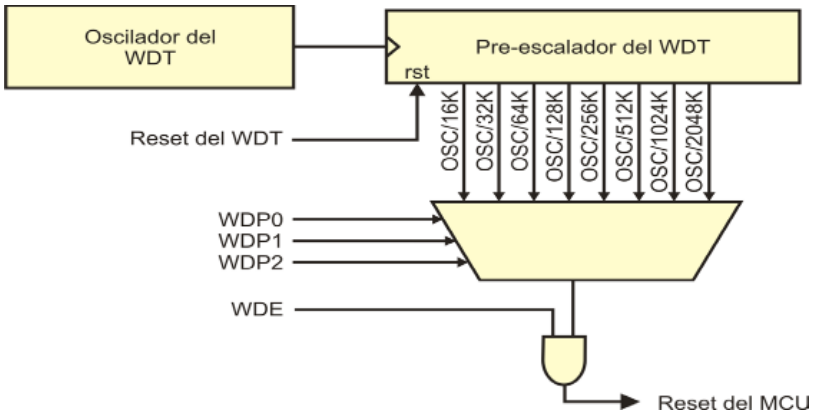


Figura 7.1 Organización del Watchdog Timer

Tabla 7.1 Factores de pre-escala del *Watchdog Timer*

WDP2	WDP1	WDP0	Ciclos	Tiempo (Vcc = 3.0 V)	Tiempo (Vcc = 5.0 V)
0	0	0	16K (16, 384)	17.1 mS	16.3 mS
0	0	1	32K (32, 768)	34.3 mS	32.5 mS
0	1	0	64K (65, 536)	68.5 mS	65 mS
0	1	1	128K (131, 072)	0.14 S	0.13 S
1	0	0	256K (262, 144)	0.27 S	0.26 S
1	0	1	512K (524, 288)	0.55 S	0.52 S
1	1	0	1, 024K	1.1 S	1.0 S
1	1	1	2, 048 K	2.2 S	2.1 S

En la tabla 7.1 se observa que el periodo es ligeramente menor cuando el voltaje de alimentación es de 5V, con respecto a un voltaje de 3.0 V. Si el periodo expira sin un reinicio al WDT, el MCU es reiniciado, ejecutando las instrucciones desde el vector de reset. La causa queda registrada en **MCUCSR.**

El WDT de un ATMega8 puede habilitarse en dos formas diferentes: Activando al fusible **WDTON**, que es parte de los *Bits de Configuración y Seguridad*, al momento de programar al dispositivo, o bien, modificando al bit WDE (*Watchdog Timer Enable)* del registro **WDTCR** durante la ejecución de una aplicación. Un ATMega16 no incluye al fusible **WDTON**, por lo tanto, la activación del WDT sólo se puede realizar con el bit **WDE** del registro **WDTCR**.

7.1.1 Registro para el Manejo del WDT

El registro de control del WDT es el **WDTCR**, cuyos bits son:

	7	6	5	4	3	2	1	0	
0x21	-	-	-	WDCE/ WDTOE	WDE	WDP2	WDP1	WDP0	**WDTCR**

- **Bits 7 al 5 – No están implementados**

- **Bit 4 – WDCE: Modifica la habilitación del WDT (ATMega8)/ WDTOE: Habilita la salida del WDT (ATMega16)**

  Este bit tiene un nombre diferente en los dispositivos bajo estudio, pero su funcionalidad es la misma. El bit es parte de un mecanismo de seguridad que evita la desactivación errónea del WDT, en un ATMega8 sólo si el WDT fue habilitado vía software. Para inhabilitar al WDT, este bit debe ponerse en alto y dentro de los siguientes 4 ciclos de reloj debe limpiarse al bit **WDE**. Pasado ese tiempo, el bit **WDCE/WDTOE** automáticamente es limpiado por hardware.

- **Bit 3 – WDE: Habilitador del WDT**

  Un 1 en este bit habilita al WDT y un 0 lo inhabilita. Sin embargo, la inhabilitación requiere que el bit **WDCE/WDTOE** esté en un nivel alto. La secuencia para inhabilitar al WDT es la siguiente:

  1. Con la misma operación, poner en alto a los bits **WDCE/WDTOE** y **WDE**. La escritura de **WDE** es parte de la secuencia, es necesaria aunque de antemano ya tenga un nivel alto.

  2. Dentro de los siguientes 4 ciclos de reloj escribir un 0 lógico en **WDE**, con ello el WDT ha quedado inhabilitado.

  Si en un ATMega8 el WDT fue habilitado activando al fusible **WDTON**, no se puede inhabilitar aun siguiendo la secuencia anteriormente descrita. En estos casos, la secuencia es útil para modificar el periodo de desbordamiento del WDT.

- **Bits 2 al 0 – WDP[2:0]: Bits para la selección del factor de pre-escala del WDT**

  Con estos bits se define el factor de pre-escala para alcanzar diferentes intervalos de tiempo, antes de un desbordamiento del WDT. En la tabla 7.1 se mostraron los intervalos de tiempo que se consiguen con las diferentes combinaciones de estos bis. La modificación de estos bits también requiere la secuencia de seguridad anteriormente descrita, excepto que en el paso 2 se involucra la modificación de los bits **WDP[2:0]**.

Ejemplo 7.1 Codifique una rutina o función para detener al WDT, en lenguaje ensamblador y en lenguaje C, enfocada a un ATMega8.

La rutina en lenguaje ensamblador es:

```
WDT_Off:
       WDR                                          ; Reinicia al WDT
       IN     R16, WDTCR
       ORI    R16, (1 << WDCE) | (1 << WDE)         ; 1 lógico en WDCE y en WDE
       OUT    WDTCR, R16
       CLR    R16
       OUT    WDTCR, R16                            ; Apaga al WDT
       RET
```

La función en lenguaje C es:

```
void WDT_Off(void) {
       asm("WDR");                                   // Reinicia al WDT

       WDTCR = WDTCR | (1 << WDCE) | (1 << WDE);// 1 lógico en
                                                     //  WDCE y en WDE

       WDTCR = 0x00;                                 // Apaga al WDT
}
```

El WDT se reinicia para evitar que desborde mientras se ejecuta la rutina. Para un ATMega16 es suficiente con remplazar al bit **WDCE** con el bit **WDTOE**.

7.2 Sección de Arranque en la Memoria de Programa

La memoria Flash está particionada en dos secciones, una de aplicación y otra de arranque. En la mayoría de sistemas no se considera esta partición y se dedica todo el espacio a la sección de aplicación. Aunque en la sección de arranque podría ubicarse una secuencia de código que corresponda con una aplicación ordinaria, la sección está orientada para que el usuario pueda ubicar un cargador, es decir, un pequeño programa que facilite modificar una parte o toda la sección de aplicación. Esto significa que los microcontroladores AVR incluyen los mecanismos de hardware necesarios para hacer que una aplicación pueda ser actualizada por sí misma, realizando una autoprogramación.

7.2.1 Organización de la Memoria Flash

La memoria Flash está organizada en páginas, en un ATMega8 cada página es de 32 palabras de 16 bits y en un ATMega16 las páginas son de 64 palabras de 16 bits. Toda la memoria, considerando la sección de aplicación y la sección de arranque, está dividida en páginas. Ambos dispositivos, ATMega8 y ATMega16, incluyen 128 páginas en su memoria Flash. En la figura 7.2 se esquematiza esta organización. El acceso para el borrado y la escritura en la memoria Flash implica la modificación de una página completa. La lectura es diferente, ésta si puede realizarse con un acceso por bytes.

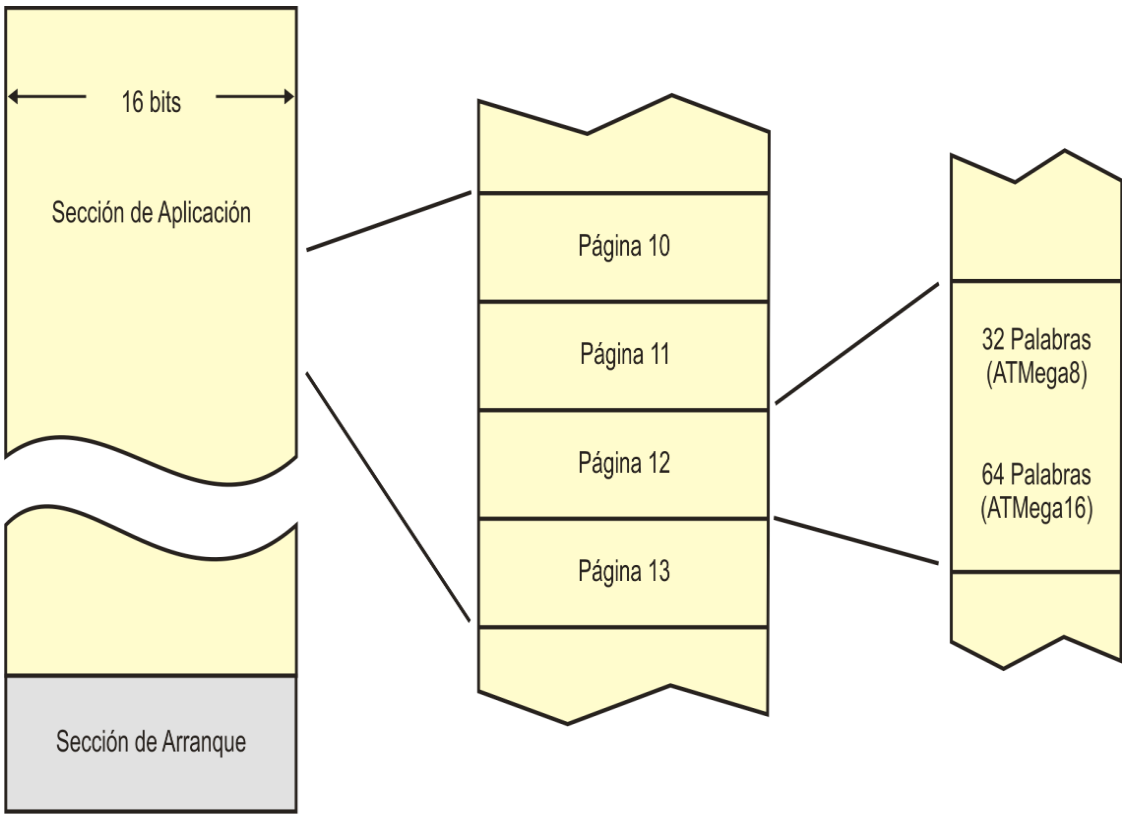


Figura 7.2 Organización de la memoria flash en páginas

La división de la memoria Flash se realiza con la programación de los fusibles **BOOTSZ1** y **BOOTSZ0**, los cuales son parte de los Bits de Configuración y Seguridad, con ellos se determina cuantas páginas son destinadas a la sección de arranque. En la tabla 7.2 se pueden ver las páginas destinadas para cada una de las secciones en un ATMega8 y en un ATMega16, en función del valor programado en estos fusibles. Un fusible tiene un 1 si está sin programar, los dispositivos son comercializados con **BOOTSZ[1:0]** = "00", es decir, los fusibles están programados para proporcionar el espacio máximo a la sección de arranque.

Tabla 7.2 División de la memoria en una sección de aplicación y una de arranque

BOOTSZ1	BOOTSZ0	Tamaño (Sección de arranque)	Páginas (ATMega8/ ATMega16)	Sección de Aplicación (ATMega8/ATMega16)	Sección de Arranque (ATMega8/ATMega16)
1	1	128 palabras	4 / 2	0x000 – 0xF7F / 0x0000 – 0x1F7F	0xF80 – 0xFFF/ 0x1F80 – 0x1FFF
1	0	256 palabras	8 / 4	0x000 – 0xEFF / 0x0000 – 0x1EFF	0xF00 – 0xFFF / 0x1F00 – 0x1FFF
0	1	512 palabras	16 / 8	0x000 – 0xDFF / 0x0000 – 0x1DFF	0xE00 – 0xFFF / 0x1E00 – 0x1FFF
0	0	1024 palabras	32 / 16	0x000 – 0xBFF / 0x0000 – 0x1BFF	0xC00 – 0xFFF / 0x1C00 – 0x1FFF

Una aplicación puede tener acceso al código escrito en la sección de arranque, por medio de llamadas a rutinas o saltos, pero si se programa al fusible **BOOTRST**, se consigue que después de un reset el CPU ejecute las instrucciones desde la sección de arranque y no desde la dirección 0.

En la memoria flash también se encuentran los vectores de las interrupciones, en las tablas 2.1 y 2.2 se mostraron las direcciones que les corresponden en un ATMega8 y en un ATMega16. No obstante, los vectores pueden moverse a la sección de arranque, esto se realiza con los bits **IVSEL** e **IVCE** del registro general para el control de interrupciones (**GICR**), correspondiendo con sus 2 bits menos significativos:

	7	6	5	4	3	2	1	0	
0x3B	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	**GICR**

- **Bit 1 – IVSEL: Selecciona la ubicación de los vectores de las interrupciones**
  Cuando **IVSEL** tiene un 0, los vectores de las interrupciones se ubican al inicio de la memoria Flash. Si el bit **IVSEL** tiene un 1, los vectores son desplazados al inicio de la sección de arranque, cuya dirección depende de los fusibles **BOOTSZ1** y **BOOTSZ0**, como se mostró en la tabla 7.2.

- **Bit 0 – IVCE: Habilita el cambio en la ubicación de los vectores de las** interrupciones

  Para evitar cambios no deseados en la ubicación de los vectores de las interrupciones, cualquier ajuste requiere la puesta en alto del bit **IVCE** y dentro de los 4 ciclos de reloj siguientes se debe escribir el valor deseado en **IVSEL**, con la escritura de un 0 en **IVCE**.

La programación del fusible **BOOTRST** es independiente del estado del bit **IVSEL**, esto significa que aunque un programa reinicie en la dirección 0 de la memoria flash, sus vectores de interrupciones podrían estar colocados al inicio de la sección de arranque, o bien, un programa podría iniciar su ejecución en la sección de arranque, conservando los vectores de interrupciones al inicio de la memoria Flash.

Además, el fusible **BOOTRST** se programa durante la descarga del código en el dispositivo, mientras que el estado del bit **IVSEL** se define durante la ejecución del programa. Por lo tanto, independientemente de la dirección de inicio, los vectores de las interrupciones pueden moverse en tiempo de ejecución.

7.2.2 Acceso a la Sección de Arranque

En el ejemplo 7.2 se ilustra el acceso a la sección de arranque, sin considerar la posibilidad de un cargador para autoprogramación, se muestra cómo un programa inicia en la sección de arranque y, si se conmuta una entrada, cómo el programa pasa a ejecutar el código ubicado en la sección de aplicación.

Ejemplo 7.2 Realice un programa para un ATMega8 que en la sección de arranque haga parpadear un LED ubicado en la terminal PB0, y en la sección de aplicación haga parpadear a otro LED, ubicado en PB1. El programa debe abandonar la sección de arranque cuando se presione un botón conectado en INT0.

Para que el programa comience su ejecución en la sección de arranque, el fusible **BOOTRST** debe activarse durante la programación del dispositivo. Si los fusibles **BOOTSZ[1:0]** se mantienen con "00", quedan disponibles 32 páginas para la sección de arranque (iniciando en 0xC00).

La solución se desarrolla en lenguaje ensamblador, dado que en lenguaje C no es posible establecer la dirección para ubicar el código generado, el programa con la solución es:

```
       .org   0x000              ; Sección de aplicación
inicio:
       LDI    R16, 0x04          ; Ubica al apuntador de pila
       OUT    SPH, R16
       LDI    R16, 0x5F
       OUT    SPL, R16
       LDI    R16, 0xFF          ; Puerto B como salida
       OUT    DDRB, R16
Lazo:                            ; Lazo infinito
       SBI    PORTB, 1           ; Parpadeo en PB1
       RCALL  Espera_500mS
       CBI    PORTB, 1
       RCALL  Espera_500mS
       RJMP   Lazo

       .org   0xC00              ; Sección de arranque
       RJMP   ini_boot
       .org   0xC01              ; Vector de la INT0 desplazado
       RJMP   inicio             ; Brinca a la sección de aplicación
                                 ; Una instrucción RETI sería ignorada
ini_boot:
       LDI    R16, 0x01          ; Mueve los vectores de interrupciones
       OUT    GICR, R16          ; Para detectar la INT0 desde la
       LDI    R16, 0x02          ; sección de arranque
       OUT    GICR, R16

       CLR    R16                ; Puerto D como entrada
       OUT    DDRD, R16
       LDI    R16, 0xFF
       OUT    DDRB, R16          ; Puerto B como salida
```

```
       OUT    PORTD, R16          ; Pull-Up en el puerto D

       LDI    R16, 0x04           ; Ubica al apuntador de pila
       OUT    SPH, R16
       LDI    R16, 0x5F
       OUT    SPL, R16

       LDI    R16, 0x02           ; Configura la INT0 por flanco de bajada
       OUT    MCUCR, R16
       LDI    R16, 0x40           ; Habilita la INT0
       OUT    GICR, R16
       SEI                        ; Habilitador global de interrupciones
Lazo_boot:                        ; Lazo infinito
       SBI    PORTB, 0            ; Parpadeo en PB0
       RCALL  Espera_500mS
       CBI    PORTB, 0
       RCALL  Espera_500mS
       RJMP   Lazo_boot
;
; Rutina para esperar 500 mS
;
Espera_500mS:
       LDI    R18, 2
et3:   LDI    R17, 250
et2:   LDI    R16, 250
et1:   DEC    R16                 ; Itera 250 veces y requiere de 4 uS
       NOP                        ;
       BRNE   et1                 ;      250 x 4 uS = 1000 uS = 1 mS
       DEC    R17                 ;
       BRNE   et2                 ;             1 mS x 250 = 250 mS
       DEC    R18                 ;
       BRNE   et3                 ;             250 mS x 2 = 500 mS
       RET
```

Se observa que el código ubicado en la sección de aplicación está completo, es decir, incluye la configuración de los puertos y la inicialización del apuntador de pila. No se considera el estado resultante de la ejecución de la sección de arranque, porque debe funcionar adecuadamente aun sin ejecutar el código de la sección de arranque (sin programar al fusible **BOOTRST**). También puede verse que es posible invocar una rutina ubicada en la sección de arranque desde la sección de aplicación.

En el código también se mostró cómo desplazar los vectores de interrupciones, para que el evento del botón sea atendido en la sección de arranque. En la ISR de la INT0 básicamente se realiza la bifurcación a la sección de aplicación, no es posible insertar una instrucción **RETI** porque regresaría la ejecución al lazo principal de la sección de arranque. Sin embargo, en la sección de aplicación quedaron inhabilitadas las interrupciones porque el bit **I** de **SREG** quedó desactivado al no ejecutarse la instrucción **RETI.**

7.2.3 Cargador para Autoprogramación

Aunque el ejemplo 7.2 funciona correctamente, no tiene mucho sentido colocar una aplicación o parte de ella en la sección de arranque. Esta sección está orientada para que el usuario pueda ubicar un cargador, es decir, un programa que permita modificar una parte o toda la sección de aplicación.

El MCU iniciaría ejecutando al cargador (ubicado en la sección de arranque), el cual esperaría durante un tiempo determinado por si desde una PC u otro sistema se solicita la actualización del programa de aplicación, utilizando alguna de las interfaces incluidas en el MCU (USART, SPI o TWI). Si durante ese periodo se establece la comunicación, por esta interfaz se debe realizar la actualización del código, empleando algún protocolo que garantice la adecuada transferencia de datos; concluida la actualización se debe continuar con la ejecución del programa de aplicación. Si el periodo termina sin una petición para la comunicación con otro sistema, el MCU simplemente da paso a la ejecución de la aplicación.

Si el cargador requiere el uso de interrupciones, debe incluir instrucciones para ubicar a los vectores de interrupciones en la sección de arranque, por lo tanto, antes de bifurcar a la sección de aplicación, los vectores deben regresarse a la sección de aplicación, para que el código de la aplicación no bifurque erróneamente a la sección de arranque.

En la figura 7.3 se ilustra esta idea, la cual en apariencia es simple, no obstante, deben tomarse en cuenta algunos aspectos relevantes que son descritos en las siguientes subsecciones.

7.2.3.1 Restricciones de Acceso en la Memoria Flash

Como parte de los *Bits de Configuración y Seguridad,* los AVR incluyen 4 fusibles para definir el nivel de protección en cada una de las secciones de la memoria Flash. Con ellos se determina si:

- La memoria Flash queda protegida de una actualización.
- Sólo la sección de arranque queda protegida.
- Sólo la sección de aplicación queda protegida.
- Ambas secciones quedan sin protección, permitiendo un acceso total.

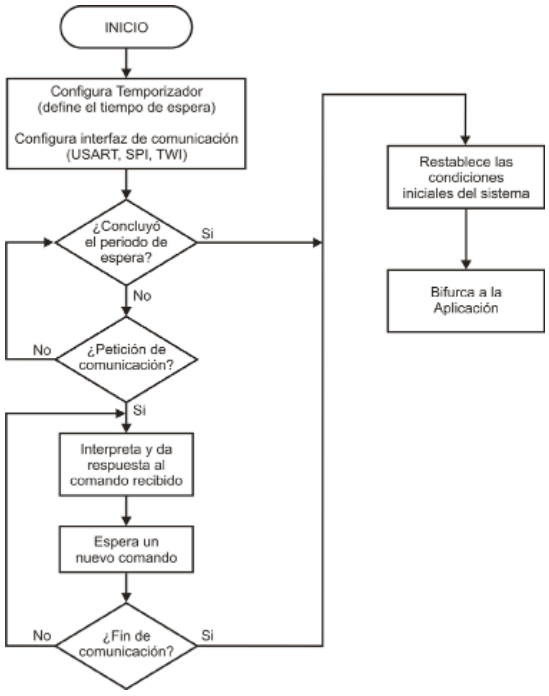


Figura 7.3 Flujo natural de un cargador para autoprogramación

Los fusibles se denominan Bits de Bloqueo de Arranque (**BLB**, *Boot Lock Bits*) y se organizan por pares, el par 0 está dedicado a la sección de aplicación y el par 1 está dedicado a la sección de arranque. En las tablas 7.3 y 7.4 se describen los diferentes niveles de protección, donde puede notarse que la seguridad se determina por las restricciones impuestas a las instrucciones **`LPM`** y **`SPM`**, orientadas a transferencias con memoria de código.

Los fusibles **BLB** tienen un 1 cuando están sin programar. La programación puede realizarse en el momento de configurar al dispositivo, aunque es posible una modificación vía software, en tiempo de ejecución.

Tabla 7.3 Bits de bloqueo para la sección de aplicación

Modo	BLB02	BLB01	Protección
1	1	1	Sin restricciones para las instrucciones **SPM** o **LPM** en la sección de aplicación.
2	1	0	La instrucción **SPM** no puede escribir en la sección de aplicación.
3	0	0	La instrucción **SPM** no puede escribir en la sección de aplicación y a **LPM**, ejecutada desde la sección de arranque, no se le permite leer en la sección de aplicación. Si los vectores de interrupciones son colocados en la sección de arranque, las interrupciones se inhabilitan mientras se ejecute desde la sección de aplicación.
4	0	1	A la instrucción **LPM,** ejecutada desde la sección de arranque, no se le permite leer en la sección de aplicación. Si los vectores de interrupciones son colocados en la sección de arranque, las interrupciones se inhabilitan mientras se ejecute desde la sección de aplicación.

Tabla 7.4 Bits de bloqueo para la sección de arranque

Modo	BLB12	BLB11	Protección
1	1	1	Sin restricciones para las instrucciones **SPM** o **LPM** en la sección de arranque.
2	1	0	La instrucción **SPM** no puede escribir en la sección de arranque.
3	0	0	La instrucción **SPM** no puede escribir en la sección de arranque y a **LPM**, ejecutada desde la sección de aplicación, no se le permite leer en la sección de arranque. Si los vectores de interrupciones son colocados en la sección de aplicación, las interrupciones se inhabilitan mientras se ejecute desde la sección de arranque.
4	0	1	A la instrucción **LPM,** ejecutada desde la sección de aplicación, no se le permite leer en la sección de arranque. Si los vectores de interrupciones son colocados en la sección de aplicación, las interrupciones se inhabilitan mientras se ejecute desde la sección de arranque.

7.2.3.2 Capacidades para Leer-Mientras-Escribe

Además de la división entre la sección de aplicación y la sección de arranque, la Flash también está dividida en dos secciones de tamaño fijo, la primera es una sección que posibilita la capacidad para Leer-Mientras-Escribe (RWW, *Read While Write***)** y la segunda sección restringe el acceso, de manera que No se puede Leer-Mientras-Escribe (NRWW, *No Read While* Write).

El tamaño de la sección NRWW ocupa el tamaño más grande de la sección de arranque, en la tabla 7.5 se muestran los tamaños de ambas secciones para un ATMega8 y para un ATMega16.

Tabla 7.5 Tamaño de las secciones RWW y NRWW

Dispositivo	Páginas RWW	Páginas NRWW	Sección RWW	Sección NRWW
ATMega8	96	32	0x000 – 0xBFF	0xC00 – 0xFFF
ATMega16	112	16	0x0000 – 0x1BFF	0x1C00 – 0x1FFF

Por lo tanto, de acuerdo al tamaño elegido para la sección de arranque, un cargador puede ocupar toda o parte de la sección NRWW, pero nunca va a poder utilizar una parte de la sección RWW. Mientras que una aplicación si puede llegar a ocupar ambas secciones de la memoria Flash.

La diferencia fundamental entre la sección RWW y la NRWW es que:

- Cuando se está borrando o escribiendo una página localizada en la sección RWW, la sección NRWW puede ser leída durante esta operación.
- Cuando se está borrando o escribiendo una página ubicada en la sección NRWW, la CPU se detiene hasta que concluya la operación.

La ubicación del cargador en la sección NRWW hace posible la lectura y ejecución de sus instrucciones consistentes en la modificación del código de la aplicación, situado en la sección RWW. Por lo tanto, Leer-Mientras-Escribe significa lecturas en la sección NRWW relacionadas con escrituras en RWW. Si un cargador intenta modificarse a sí mismo, la ejecución de la CPU queda detenida mientras se realice el borrado o la escritura de una página. En la figura 7.4 se ilustra este comportamiento, en donde puede verse que el apuntador Z es empleado para referenciar la ubicación de la página a modificar.

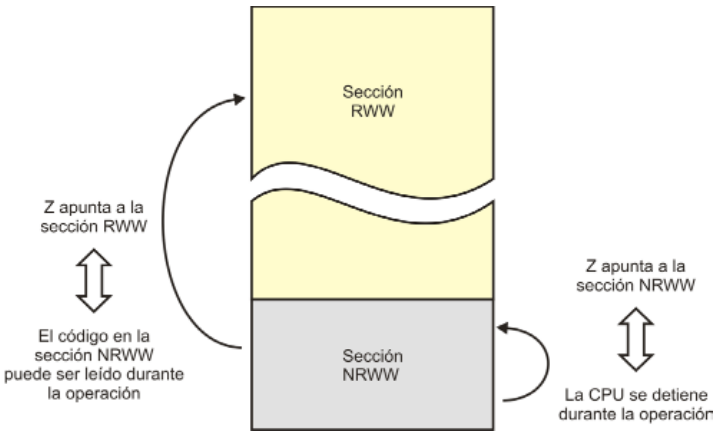


Figura 7.4 Lee de la sección NRWW mientras escribe en la sección RWW

En general, si una secuencia de código se va a mantener sin cambios y la aplicación requiere que incluya instrucciones para modificar un espacio variable en la memoria de código, es necesario que la secuencia fija se encuentre en la sección NRWW y el espacio variable en la sección RWW. La secuencia fija no forzosamente debe ser parte de un cargador.

7.2.3.3 Escritura y Borrado en la Memoria Flash

En la descripción de la instrucción **`SPM`** se indicó que con ella se escribe una palabra de 16 bits en la memoria de código, la palabra debe estar en los registros R1:R0 y la dirección de acceso en el registro Z. Sin embargo, la memoria Flash está organizada en páginas (figura 7.2) y no puede ser modificada por localidades individuales, la escritura o borrado se realiza en páginas completas.

Por lo tanto, el hardware incluye un buffer de memoria del tamaño de una página, en el que se hace un almacenamiento temporal antes de escribir en la memoria Flash. El espacio del buffer es independiente de la memoria SRAM, es sólo de escritura y debe ser llenado palabra por palabra.

El registro Z se utiliza para direccionar una palabra en el buffer, en un ATMega8 sólo se utilizan 5 bits (32 palabras) y en un ATMega16 se emplean 6 bits (64 palabras), en ambos casos, el bit menos significativo de Z es ignorado. En la figura 7.5 se ilustra el direccionamiento en un ATMega8.

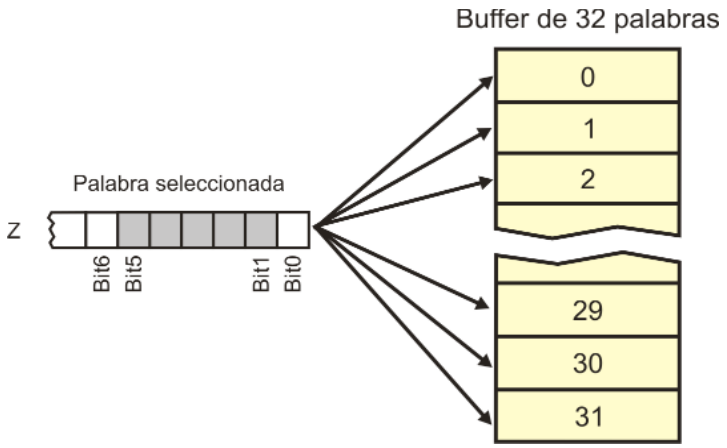


Figura 7.5 Direccionamiento en el buffer temporal

Para escribir una palabra en el buffer, ésta se coloca en R1:R0, con Z se apunta a la dirección deseada, se pone en alto el habilitador de escritura en memoria de programa (**SPMEN**, *Store Program Memory Enable*)**,** que corresponde con el bit menos significativo del Registro para el Control del Almacenamiento en Memoria de Programa (**SPMCR**, *Store Program Memory Control Register*)**,** y finalmente, se ejecuta la instrucción **`SPM`,** dentro de los 4 ciclos de reloj siguientes a la habilitación de escritura. No es posible la escritura de un byte individual.

Una vez que un buffer se ha llenado, su contenido puede ser copiado en la memoria Flash. Ésta y otras tareas se realizan por medio del registro **SPMCR**, los bits de este registro son:

	7	6	5	4	3	2	1	0	
0x37	SPMIE	RWWSB	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	**SPMCR**

- **Bit 7 – SPMIE: Habilita la interrupción por almacenamiento en la memoria de programa**

  Si este bit y el bit **I** de **SREG** están en alto, se produce una interrupción tan pronto como el bit **SPMEN** es puesto en bajo. El bit **SPMEN** se pone en bajo cuando concluye una operación sobre la memoria flash.

- **Bit 6 – RWWSB: Sección RWW ocupada**

  Bandera que se pone en alto cuando se está realizando una operación de autoprogramación (borrando o escribiendo una página) en la sección RWW y por lo tanto, no se puede realizar otro acceso a la sección RWW. La limpieza de la bandera se realiza con el apoyo del bit **RWWSRE**.

- **Bit 5 – No está implementado**

- **Bit 4 – RWWSRE: Habilita la lectura de la sección RWW**

  Si en la sección RWW se está realizando una operación de autoprogramación (borrando o escribiendo una página), la bandera **RWWSB** y el bit **SPMEN** van a tener un nivel alto. Al concluir con la operación, el bit **SPMEN** es puesto en bajo, pero la sección RWW aún permanece bloqueada. Para rehabilitar su acceso, los bits **RWWSRE** y **SPMEN** deben ser puestos en alto al mismo tiempo y durante los 4 ciclos de reloj siguientes se debe ejecutar la instrucción `SPM`, sin importar el contenido de los registros R1:R0 y del apuntador Z.

- **Bit 3 – BLBSET: Ajuste de los Bits de Bloqueo de Arranque**

  Con este bit se realiza la escritura y lectura de los fusibles de **BLB**, empleados para restringir el acceso a las diferentes secciones en la memoria Flash (ver la sección 7.2.2.1).

  Para una escritura, los bits **BLBSET** y **SPMEN** deben ser puestos en alto al mismo tiempo y durante los 4 ciclos de reloj siguientes se debe ejecutar la instrucción **`SPM`.** El valor a escribir debe estar en el registro R0, de acuerdo con la siguiente distribución:

1	1	BLB12	BLB11	BLB02	BLB01	1	1	**R0**

  Sin importar el valor de R1 y colocando 0x0001 en el apuntador Z. El bit **BLBSET** se pone en bajo al completar la escritura o si la instrucción **`SPM`** no se ejecutó dentro de los 4 ciclos de reloj (falló al intentar la escritura). Para una lectura, los bits **BLBSET** y **SPMEN** deben ser puestos en alto al mismo tiempo y durante los 3 ciclos de reloj siguientes se debe ejecutar la instrucción **`LPM`**. El valor leído queda disponible en el registro señalado como destino en la instrucción.

- **Bit 2 – PGWRT: Habilita la escritura de una página**

  Este bit hace posible que el contenido del buffer temporal sea escrito en la memoria Flash, en la página direccionada con la parte alta del apuntador Z. Para ello, los bits **PGWRT** y **SPMEN** deben ser puestos en alto al mismo tiempo y durante los 4 ciclos de reloj siguientes se debe ejecutar la instrucción **`SPM`**. El valor de los registros R1:R0 es ignorado. El bit se limpia automáticamente al concluir la escritura de la página o si la instrucción **`SPM`** no se ejecutó dentro de los 4 ciclos de reloj (falló al intentar la escritura). Si Z direcciona una página de la sección NRWW, la CPU se detiene mientras se realiza la escritura de la página.

- **Bit 1 – PGERS: Habilita el borrado de una página**

  Este bit hace posible el borrado de una página en la memoria Flash, la página debe ser direccionada con la parte alta del apuntador Z. Los bits **PGERS** y **SPMEN** deben ser puestos en alto al mismo tiempo y durante los 4 ciclos de reloj siguientes se debe ejecutar la instrucción **`SPM`**. El valor de los registros R1:R0 es ignorado. El bit se limpia automáticamente al concluir el borrado de la página o si la instrucción **`SPM`** no se ejecutó dentro de los 4 ciclos de reloj (falló al intentar el borrado). Si Z direcciona una página de la sección NRWW, la CPU se detiene mientras se realiza el borrado de la página.

- **Bit 0 – SPMEN: Habilitador de escritura en memoria de programa**

  Este bit debe ponerse en alto para modificar la memoria de programa o a los fusibles relacionados, solo o en combinación con otros bits del registro **SPMCR**, y dentro de los 4 ciclos de reloj siguientes se debe ejecutar la instrucción **`SPM`**.

  Si únicamente se puso en alto al bit **SPMEN**, la instrucción **`SPM`** escribe la palabra contenida en R1:R0 en el buffer de almacenamiento temporal, en la dirección apuntada por Z. Si **SPMEN** y **RWWSRE** se pusieron en alto, la instrucción `SPM` desbloquea el acceso a la sección RWW y limpia la bandera **RWWSB**. Si **SPMEN** y **BLBSET** se pusieron en alto, la instrucción **`SPM`** escribe en los fusibles de **BLB** el dato contenido en R0. Si **SPMEN** y **PGWRT** se pusieron en alto, la instrucción **`SPM`** escribe el contenido del buffer temporal en la memoria Flash, en la página direccionada con la parte alta del apuntador Z. Si **SPMEN** y **PGERS** se pusieron en alto, la instrucción **`SPM`** borra la página direccionada con la parte alta del apuntador Z.

  Además, si **SPMEN** y **BLBSET** se ponen en alto, y posteriormente se ejecuta a la instrucción **`LPM`**, se leen los fusibles de **BLB** y su valor queda en el registro indicado en la instrucción.

7.2.3.4 Direccionamiento de la Flash para Autoprogramación

El contador de programa (PC) es el registro encargado de direccionar la memoria de código. Los bits del PC se dividen en 2 partes, por la organización en páginas de la memoria Flash, con los bits más significativos se selecciona una página (PCPAGE) y con los bits menos significativos una palabra (PCWORD), dentro de la página. Las operaciones relacionadas con la instrucción **`SPM`** involucran el uso del apuntador Z, el cual toma la misma distribución de bits que el PC, pero sin incluir a su bit menos significativo. En la figura 7.6 se muestra la distribución de los bits para direccionar la memoria Flash.

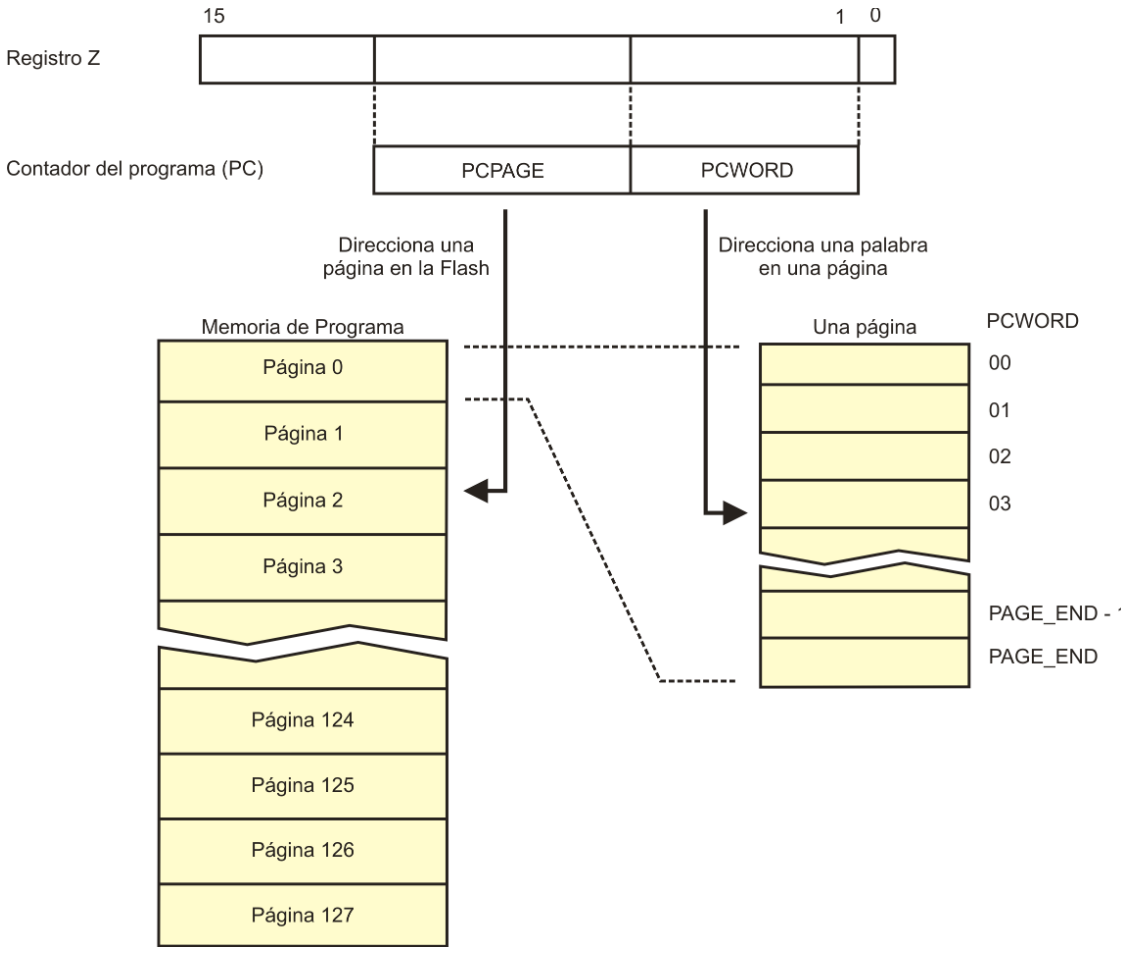


Figura 7.6 Distribución de bits para direccionar la memoria de código

En la tabla 7.6 se muestra el tamaño y ubicación de cada una de las partes del PC, para un ATMega8 y un ATMega16.

Tabla 7.6 Organización y acceso a la memoria de código

MCU	Páginas	Palabras por página	Tamaño de PCWORD	Tamaño de PCPAGE	Ubicación de PCWORD	Ubicación de PCPAGE
ATMega8	128	32	7 bits	5 bits	PC[11:5]	PC[4:0]
ATMega16	128	64	7 bits	6 bits	PC[12:6]	PC[5:0]

Al momento de escribir o borrar una página se debe considerar el tamaño de PCWORD y su ubicación en el PC, ya que de ello depende el valor que toma el apuntador Z. Por ejemplo, si para un ATMega8 se requiere que Z apunte a la página 5, se le debe escribir el valor de 0x0140, de acuerdo con la figura 7.7.

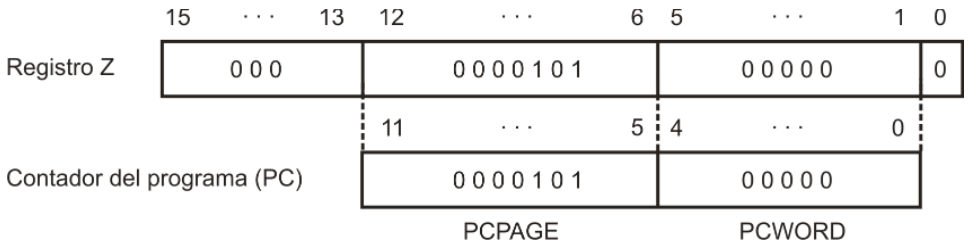


Figura 7.7 Ejemplo para definir el valor de Z, en un ATMega8

Las operaciones de escritura y borrado se realizan en varios ciclos de reloj, esto requiere la inserción de un latch intermedio en el que se captura la dirección proporcionada por el apuntador Z, de manera que, una vez iniciada la operación, el apuntador Z puede ser utilizado para otras tareas.

La única operación realizada con la instrucción **`SPM`** que no utiliza al apuntador Z, es para la escritura de los fusibles de **BLB**, en la cual, el contenido de Z es ignorado y no tiene efecto en la operación.

La instrucción **`LPM`** también hace uso del apuntador Z, sin embargo, el acceso para la carga de memoria de programa se realiza por bytes, por ello, esta instrucción si utiliza al bit menos significativo del registro Z.

7.2.3.5 Programación de la Flash

Es necesario borrar una página de la memoria Flash antes de escribirle nuevos datos, los cuales inicialmente deben estar en el buffer temporal. El proceso puede realizarse con dos secuencias diferentes. La primera secuencia consiste en: Llenar el buffer temporal, borrar la página y escribir el contenido del buffer temporal en la página borrada. La segunda secuencia implica: Borrar la página, llenar el buffer temporal y escribir el contenido del buffer temporal en la página borrada.

No es posible escribir sólo una fracción de una página, si esto es necesario, en el buffer deben incluirse ambas partes, la fracción a modificar y el contenido que va a permanecer sin cambios, para posteriormente escribir la página completa en la memoria Flash.

7.3 Bits de Configuración y Seguridad

Los microcontroladores AVR incluyen un conjunto de fusibles que pueden ser programados para proteger el contenido de un dispositivo o definir su comportamiento. Los fusibles se organizan en 3 bytes y tienen un 1 lógico cuando están sin programar.

El contenido del primer byte, para un ATMega8 y un ATMega16, se muestra en la tabla 7.7, en donde puede notarse que este byte está relacionado con la seguridad del dispositivo. En las posiciones 7 y 6 no hay fusibles implementados. Los que se ubican

en las posiciones de la 5 a la 2 son los Bits de Bloqueo de Arranque **(BLB),** están orientados a proteger el contenido de la memoria de código y se organizan por pares, el par 0 (**BLB02** y **BLB01**) está dedicado a la sección de aplicación y el par 1 (**BLB12** y **BLB11**) a la sección de arranque.

Tabla 7.7 Fusibles relacionados con la seguridad del MCU

No. Bit	Nombre	Descripción	Valor por default
7	-		1
6	-		1
5	BLB12	Bit de Bloqueo de Arranque.	1
4	BLB11	Bit de Bloqueo de Arranque.	1
3	BLB02	Bit de Bloqueo de Arranque.	1
2	BLB01	Bit de Bloqueo de Arranque.	1
1	LB2	Bit de seguridad.	1
0	LB1	Bit de seguridad.	1

En las tablas 7.3 y 7.4 se describieron los diferentes niveles de protección (tabla 7.3 para la sección de aplicación y tabla 7.4 para la de arranque), donde se mostró que la seguridad se determina por las restricciones impuestas a las instrucciones **`LPM`** y **`SPM`**, relacionadas con transferencias de memoria de código.

Los fusibles de las posiciones 1 y 0 protegen el contenido de las memorias Flash y EEPROM, ante programaciones futuras, se tienen 3 niveles de seguridad, los cuales se describen en la tabla 7.8.

Tabla 7.8 Modos de protección del contenido de la memoria Flash y EEPROM

Modo	LB2	LB1	Tipo de protección
1	1	1	Sin protección para las memorias Flash y EEPROM.
2	1	0	La programación futura de las memorias Flash y EEPROM es inhabilitada en los modos de programación Serial y Paralelo. Los fusibles también son bloqueados en los modos de programación Serial y Paralelo.
3	0	0	La programación y verificación futura de las memorias Flash y EEPROM es inhabilitada en los modos de programación Serial y Paralelo. Los fusibles también son bloqueados en los modos de programación Serial y Paralelo.

De acuerdo con la tabla 3.8, si se va a programar el modo 2 o el modo 3 con los fusibles **LB[2:1]**, primero se deben configurar los demás fusibles, antes de perder el acceso.

Los otros 2 bytes de fusibles son para configuración, en el segundo byte difieren los dos fusibles ubicados en las posiciones más significativas, de un ATMega8 con los de un ATMega16. En la tabla 7.9 se describe el segundo byte de fusibles y se resalta la diferencia entre ambos dispositivos.

Se observa que en un ATMega16 la interfaz JTAG está habilitada por default, esto hace que el puerto C no pueda emplearse por completo como entrada y salida de propósito general, si se requiere de su uso y no se va a emplear a la interfaz JTAG, ésta debe ser inhabilitada.

El tercer byte de fusibles se muestra en la tabla 7.10, estos fusibles están relacionados con el sistema de *reset* y el sistema de reloj, su aplicación fue descrita en las secciones 2.7 y 2.8.

Tabla 7.9 Segundo byte de fusibles y el primero relacionado con la configuración del MCU

No. Bit	Nombre	Descripción	Valor por default
7	RSTDISBL (ATMega8)	Determina si PC6 es una I/O genérica o terminal de *reset*.	1 (PC6 es *reset*)
7	OCDEN (ATMega16)	Habilita al hardware incluido para depuración (OCD, *On-Chip Debug*).	1 (ODC inhabilitado)
6	WDTON (ATMega8)	Define si el WDT va a estar siempre activado.	1 (Sin programar, el WDT se habilita con WDTRC)
6	JTAGEN (ATMega16)	Habilita a la interfaz JTAG.	0 (Interfaz JTAG habilitada)
5	SPIEN	Habilita la programación serial y descarga de datos vía SPI.	0 (Programación SPI habilitada)
4	CKOPT	Opciones del oscilador (Descrito en la sección 2.8).	1
3	EESAVE	El contenido de la EEPROM es preservado durante el borrado del dispositivo.	1 (La EEPROM no se preserva)
2	BOOTSZ1	Determina el tamaño de la sección de Arranque (tabla 7.2).	0
1	BOOTSZ0		0
0	BOOTRST	Selecciona la ubicación del vector de *reset* (sección 7.2).	1 (Vector de *reset* en 0x000)

Tabla 7.10 Fusibles relacionados con la seguridad del MCU

No. Bit	Nombre	Descripción	Valor por default
7	BODLEVEL	Define el nivel para el detector de un bajo voltaje, para un reinicio.	1
6	BODEN	Habilita al detector de bajo voltaje.	1
5	SUT1	Seleccionan el tiempo de establecimiento.	1
4	SUT0		0
3	CKSEL3	Seleccionan la fuente de reloj.	0
2	CKSEL2		0
1	CKSEL1		0
0	CKSEL0		1

Si se planea la modificación de los fusibles, es conveniente realizar su lectura antes de escribir los nuevos valores. Algunas herramientas de programación no realizan una lectura automática, de manera que si se modifican sólo los fusibles de interés sin conservar el valor de los restantes, podría conducir a que el microcontrolador no opere o que lo haga en forma incorrecta.

7.4 Interfaz JTAG

El acrónimo JTAG **(***Joint Test Action Group***)** hace referencia a la norma IEEE 1149.1, originalmente desarrollada para la evaluación de circuitos impresos. Actualmente con JTAG se describe una interfaz serial empleada para la programación y depuración de microcontroladores u otros dispositivos programables, como los FPGAs. Esta interfaz está incluida en los microcontroladores ATMega16 y cumple con ambos propósitos: programación y depuración.

7.4.1 Organización General de la Interfaz JTAG

En la figura 7.8 se muestra la organización general de la interfaz JTAG y su vinculación con un sistema digital. Desde alguna computadora u otro maestro se puede evaluar al sistema digital, para ello únicamente se emplean 4 señales:

- TDI **(***Test Data In***)** Entrada de datos de prueba.
- TDO **(***Test Data Out***)** Salida de datos de prueba.
- TMS **(***Test Mode Select***)** Selección del modo de prueba.
- TCK **(***Test Clock***)** Reloj.

La señal de *reset* no es parte del estándar, por lo que su disponibilidad depende del dispositivo en particular.

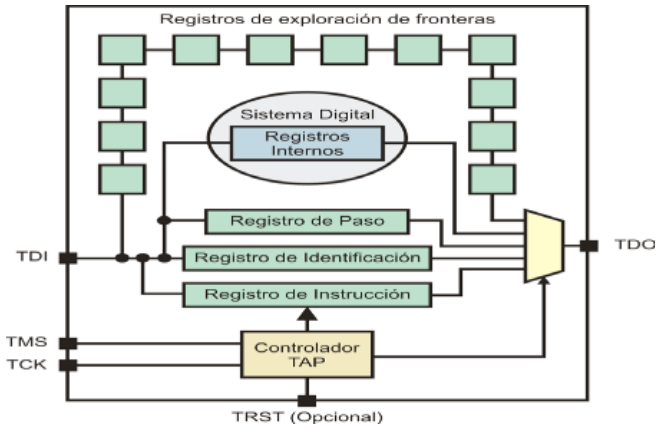


Figura 7.8 Organización de la Interfaz JTAG

La interfaz incluye un conjunto de registros que rodean al sistema digital bajo evaluación, todos funcionando bajo un esquema de desplazamiento serial. El registro de instrucción mantiene al comando recibido por parte del maestro mientras se ejecuta. El registro de identificación debe contener un código con el que se describe al dispositivo. Los registros de exploración de fronteras **(***boundary-scan***)** permiten al maestro conocer las actividades que ocurren en el entorno del dispositivo bajo exploración.

Es posible conectar varios dispositivos en serie, conectando la salida TDO de un dispositivo con la entrada TDI de otro. El registro de paso permite un flujo directo de información, omitiendo la evaluación de un dispositivo en particular, dentro de una cadena serial.

Con la interfaz JTAG también se puede conocer el estado del sistema, dado que es posible la lectura de sus registros internos.

El controlador del Puerto de Acceso para Pruebas (TAP, *Test Access Port***)** es el módulo que coordina a la interfaz JTAG. Se trata de una máquina de estados que es controlada por las señales TCK y TMS. La máquina es sincronizada por la señal de reloj recibida en TCK. Con la señal TMS se selecciona el modo de funcionamiento, es decir, su valor determina la secuencia de trabajo del controlador TAP.

7.4.2 La Interfaz JTAG y los Mecanismos para la Depuración en un AVR

Con la interfaz JTAG incluida en los ATMega16 se puede programar y depurar al dispositivo. En la figura 7.9 se muestra la organización de la interfaz y también pueden apreciarse todos los módulos agregados para hacer posible la depuración.

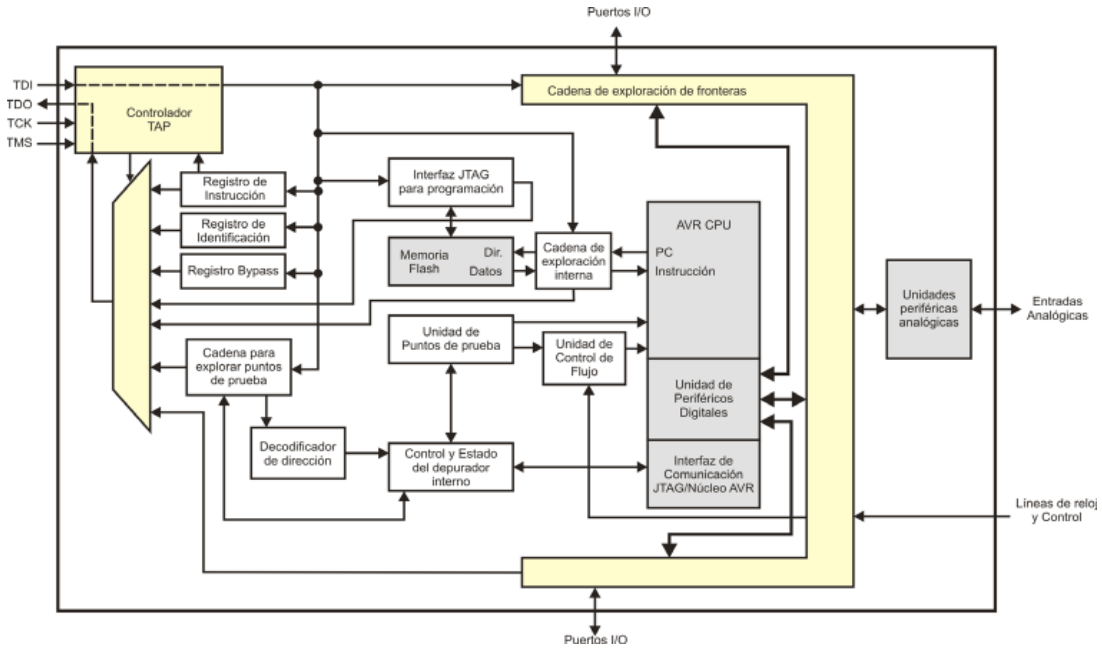


Figura 7.9 Organización de la Interfaz JTAG y su relación con el depurador interno

La relación de las señales de la interfaz JTAG con las terminales de los puertos se muestra en la tabla 7.8.

Tabla 7.8 Ubicación de las terminales de la interfaz JTAG en un ATMega16

Terminal JTAG	Ubicación	Terminal en un ATMega16
TCK	PC2	24
TMS	PC3	25
TDO	PC4	26
TDI	PC5	27

Los dispositivos ATMega16 son distribuidos con la interfaz JTAG habilitada, por lo que en principio el puerto C no está completamente disponible para entradas o salidas. Para el uso del puerto completo se debe modificar al fusible **JTAGEN**, el cual es parte de los *Bits de Configuración y Seguridad***,** (sección 7.3). No obstante, si alguna aplicación requiere la inhabilitación temporal de la interfaz JTAG, ésta se realiza con el bit **JTD** (*JTAG Disable*) ubicado en la posición 7 del registro **MCUCSR**.

	7	6	5	4	3	2	1	0	
0x34	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	**MCUCSR**

El bit **JTD** tiene un 0 lógico después de un *reset***,** por lo tanto la interfaz JTAG está habilitada (siempre que el bit **JTAGEN** esté programado). La inhabilitación requiere la escritura de un 1 lógico. Pero para evitar habilitaciones o inhabilitaciones no intencionales, la escritura del bit **JTD** requiere de 2 escrituras con el mismo valor, dentro de un periodo de 4 ciclos de reloj.

El bit **JTRF** (bit 4 de **MCUCSR**) también está relacionado con la interfaz JTAG, es una bandera que se pone en alto cuando hay un reinicio del sistema debido a la instrucción AVR_RESET, de la interfaz JTAG (en la sección 2.7 se describieron las opciones de reinicio).

Aunque la interfaz JTAG y los recursos de depuración se muestran como un hardware complejo, en la práctica su uso es transparente, el software AVR Studio incluye los mecanismos para la programación y depuración de dispositivos. Empleando alguna plataforma de desarrollo, como la tarjeta *AVR Dragon*[7] .

La depuración hace uso del Registro del Depurador Interno (**OCDR**, *On-Chip Debug Register*)**,** el cual proporciona un canal de comunicación entre el programa ejecutándose en el núcleo AVR y el depurador. Los bits de este registro son:

	7	6	5	4	3	2	1	0	
0x31	IDRD	Registro del Depurador Interno							**OCDR**

7 AVR Dragon es una tarjeta de bajo costo (aproximadamente $60.00 USD) desarrollada por Atmel, con la cual es posible programar microcontroladores AVR por los puertos SPI y JTAG. Desde el puerto JTAG también es posible la depuración, si el microcontrolador tiene una memoria hasta de 32 Kbyte. La programación y depuración se realizan desde el AVR Studio, aun si el programa se desarrolló en Lenguaje C.

El bit **IDRD (***I/O Debug Register Dirty***)** es una bandera que se pone en alto indicando una escritura en el registro **OCDR**. El depurador limpia la bandera cuando ha leído la información.

En un ATMega16 el registro **OCDR** comparte la dirección con el registro **OSCCAL**, el cual es utilizado para calibrar al oscilador interno (descrito en la sección 2.8.4). Por lo tanto, se tiene acceso al registro **OCDR** sólo si el fusible **OCDEN** fue programado, en otro caso, con la dirección 0x31 se tiene acceso al registro **OSCCAL**. El fusible **OCDEN** es parte de los *Bits de Configuración y Seguridad***.**

7.5 Ejercicios

1. Para evaluar la funcionalidad del WDT:

    a) Desarrolle un contador binario de 16 bits, empleando 2 puertos de un microcontrolador AVR. Habilite al WDT pero en el código no incluya instrucciones para su reinicio. Observe hasta qué número alcanza el contador antes de reiniciar la cuenta y estime el periodo de desbordamiento del WDT.

    b) Modifique los bits **WDP[2:0]** del registro **WDTCR** y observe el cambio en el periodo de desbordamiento. Estime estos periodos con diferentes combinaciones de **WDP[2:0]**.

    c) Coloque la instrucción `WDR` dentro del lazo que incrementa al contador y observe cómo la cuenta ya no se reinicia.

2. Empleando la sección de arranque y la sección de aplicación, desarrolle un sistema con una doble funcionalidad, por ejemplo, podrían conectarse 4 displays de 7 segmentos con un bus de datos común y las aplicaciones posibles: Un contador ascendente/descendente y una marquesina de mensajes. Ubique la aplicación 1 en la sección de arranque y la 2 en la sección de aplicación. Agregue el hardware y software necesario para una comunicación serial y acondicione para que el sistema inicie con la aplicación 1 y tras recibir un comando serial, conmute a la aplicación 2.

3. Desarrolle un cargador para autoprogramación con base en el diagrama de flujo de la figura 7.3, utilice a la USART como medio de comunicación, de manera que la nueva aplicación provenga de una PC. Pruebe cargando diferentes aplicaciones.

4. En el ejemplo 3.1 se hizo parpadear a un LED conectado en la terminal PB0 a una frecuencia aproximada de 1 Hz, implemente este ejemplo y observe cómo cambia la frecuencia del parpadeo al modificar el valor de los fusibles **CKSEL[3:0]** *(Bits de Configuración y Seguridad)*, haciendo que el MCU opere a 2, 4 u 8 MHz (los ATMega8 y ATMega16 son distribuidos operando a una frecuencia de 1 MHz).