CIRCUITOS DIGITALES Y MICROCONTROLADORES

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4																				
5					P															
6																				
7																				
8																				
9																				
10																				
11																				
12																				
13																				
14																				
15																				
16																				
17																				
18						L														
19																				
20																				
21																				
22																				
23																				
24																				
25																				
26																				
27																				
28																				
29																				
30																				
31																				
32																				
33																				
34																				
35																				
36																				
37																				
38																				
39																				
40																				

1 – Modificadores de acceso

1.1 – Static

Se puede utilizar en funciones y en variables, tanto locales como globales.

Las funciones declaradas como static son privadas, por lo que su alcance se limita únicamente al archivo .c donde está definida. Las variables locales declaradas como static poseen una dirección fija en memoria RAM que se conserva durante toda la ejecución del programa, por tanto, se puede recuperar su valor en todos los llamados a la función. Las variables globales y static son privadas y limitadas al archivo .c donde están definidas.

Ejemplo: static void estadoPin() {…} sólo se puede acceder desde su archivo .c

1.2 – Volatile

Se puede utilizar únicamente en variables. Si la variable puede llegar a cambiar su valor por acciones externas al programa (registros del MCU, flags usados en ISR), no se debe permitir que el compilador realice optimizaciones sobre la misma.

Prohíbe al compilador realizar optimizaciones sobre la variable.

Ejemplo: volatile uint8_t FLAG_TIMER = 0; modificado en ISR y Loop.

1.3 – Const

Se utiliza únicamente en variables. Indica al compilador que la misma es de sólo lectura (READ_ONLY), de modo que se prohíba su modificación en el programa luego de su inicialización. Estas variables pueden almacenarse en ROM (uso de PROGMEM).

Ejemplo: const char mensaje[] = “Hola!”; no puede ser re-asignado después.

1.4 – Extern

Se puede utilizar únicamente en variables globales. Indica que la variable está definida en otro archivo .c del proyecto, pero se desea utilizar en el archivo actual.

1.5 – Register

Se utiliza en variables locales. Sugiere al compilador que la misma sea colocada en algún registro de la CPU, con el objetivo de optimizar su tiempo de acceso.

2 – Real Time Operating System

Abreviado como “RTOS”, es un sistema operativo que provee respuestas a los eventos con un tiempo de respuesta acotado (hay un máximo). Puede ser soft o hard:

- Soft: se permite que a veces no se cumplan las cotas de tiempo (red telefónica).
- Hard: se debe garantizar que se cumplan los tiempos (control de planta nuclear).

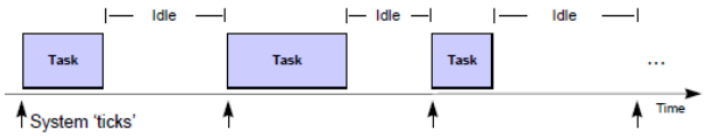
Las tareas pueden ser dependientes entre sí, pudiendo actuar cooperativamente, o bien, independientes, aunque todas comparten los recursos del MCU. La planificación de las tareas abarca las capas de Microkernel (scheduling) y Nanokernel (manejo de hilos).

El planificador es quien decide cuál es la siguiente tarea a ejecutar cuando la CPU se libera. Puede ser apropiativo (se puede detener una tarea antes que finalice para dar comienzo a otra), o no apropiativo (permite que las tareas se ejecuten hasta terminar).

Por otra parte, el planificador puede usar prioridades de tareas, las cuales pueden ser fijas[1] o dinámicas. Si el planificador es apropiativo y basado en prioridades, siempre se ejecuta la tarea habilitada de mayor prioridad. Si no es apropiativo, usa la prioridad para decidir cuando la actual finalice, nunca interrumpe la ejecución de una tarea.

2.1 – Real Time Interrupt

Abreviado como “RTI”, es la interrupción periódica de Timer. Se usa como base de tiempo del sistema para temporizar las tareas, cada interrupción es un tick. Cuando el CPU no ejecuta tareas, el MCU se puede poner en bajo consumo hasta el próximo tick.



2.2 – Simple Embedded Operating System

Es la base de un RTOS con esquema de planificación no apropiativa y basada en prioridades, es decir, no interrumpe a las tareas y decide cuál sigue según la prioridad.

Para lograrlo, se programa un planificador que decida cuál tarea ejecutar en base a una temporización RTI, y un despachador que ejecute las planificadas con prioridades.

[1] Las prioridades estáticas se dan por el orden en el código del dispatcher (consulta de flags).

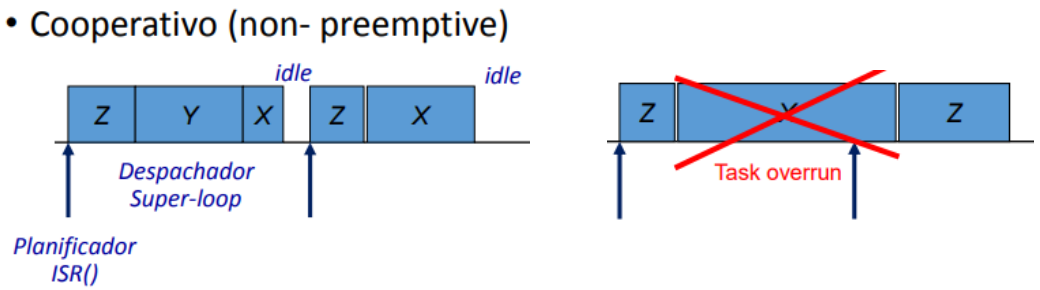
2.3 – Esquema de planificación cooperativo

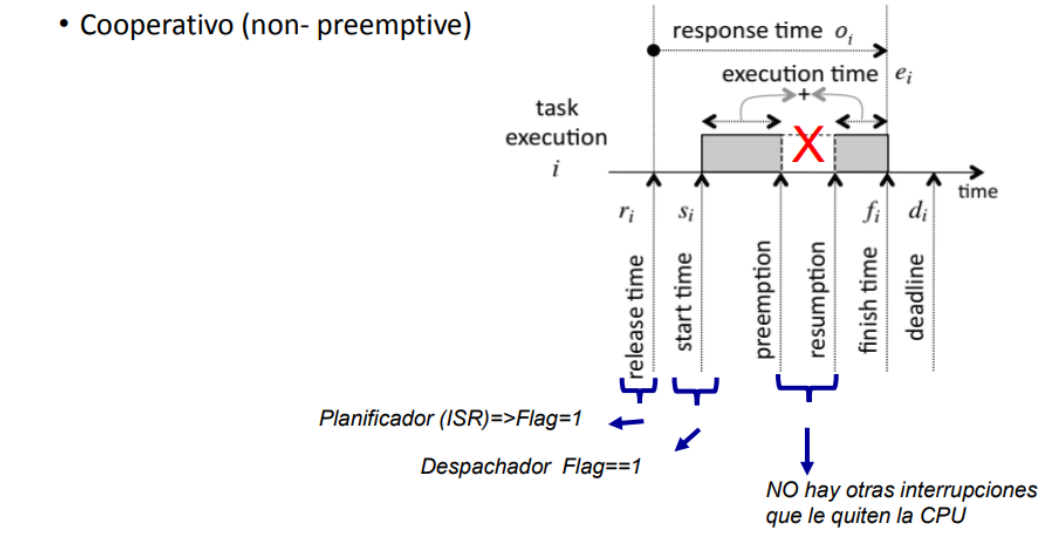
Se explicará el caso con interrupción única.

Consiste en un sistema en el cual la ejecución de las tareas se encuentra planificada en base a una temporización tipo RTI (interrupción de tiempo real). Dicha interrupción se realiza periódicamente y se utiliza como marca de tiempo del sistema (tick).

En cada tick del sistema, un planificador o *scheduler* se encarga de decidir cuáles tareas corresponden ejecutarse, para que luego un despachador o *dispatcher* las ejecute en un orden acorde a sus prioridades. Como el esquema es cooperativo o *non-preemptive*, no se detienen o interrumpen hasta que sean terminadas (Run to Completion).

Se imponen restricciones de tiempo de ejecución a la tarea siguiente, de modo que la suma de los WCET (peor tiempo de ejecución) de cada tarea entre 2 ticks sea menor al tiempo de tick. La planificación requiere un análisis y es determinística. No hay conflictos por recursos compartidos o secciones críticas. El tiempo entre la finalización de la última tarea planificada y el siguiente tick se denomina "idle", y se puede poner al MCU en *sleep mode* (ahorro de energía) para que la CPU no esté ociosa hasta el próximo tick.





2.3.1 – Ejemplo con superposición

Planificar 3 tareas periódicas de 10, 25 y 100 ms. Para evitar la superposición cada 100 ms, se puede usar un tiempo de tick menor (por ejemplo, 1 ms), y desplazar el slot inicial de cada tarea. En este caso, X en 0 ms, Y en 1 ms y Z en 2 ms.

Inicializar contador de X en 1
Inicializar contador de Y en 4
Inicializar contador de Z en 19
Inicializar flags de X, Y, Z en cero

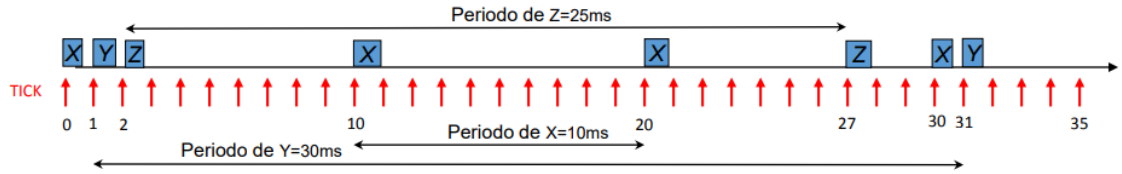
En cada interrupción periódica de 5 ms…
    Incrementar contadores de X, Y, Z

    Si el contador de X es igual a 2…
        Activar flag de X
        Reiniciar contador de X a cero
    Si el contador de Y es igual a 5…
        Activar flag de Y
        Reiniciar contador de Y a cero
    Si el contador de Z es igual a 20…
        Activar flag de Z
        Reiniciar contador de Z a cero

En cada tick del sistema…
    Si el flag de X está activo…
        Realizar tarea X
        Reiniciar flag de X
    Si el flag de Y está activo…
        Realizar tarea Y
        Reiniciar flag de Y
    Si el flag de Z está activo…
        Realizar tarea Z
        Reiniciar flag de Z

    Poner MCU en modo bajo consumo

2.3.2 – Ejemplo sin superposición



Inicializar contador de X en 9

Inicializar contador de Y en 23 ← *Desplazado en 1 slot*

Inicializar contador de Z en 97 ← *Desplazado en 2 slots*

Inicializar flags de X, Y, Z en cero

En cada interrupción periódica de 1 ms… ← *Subdividir el tick de 5 ms*

Incrementar contadores de X, Y, Z

Si el contador de X es igual a 10…

Activar flag de X

Reiniciar contador de X a cero

Si el contador de Y es igual a 25…

Activar flag de Y

Reiniciar contador de Y a cero

Si el contador de Z es igual a 100…

Activar flag de Z

Reiniciar contador de Z a cero

En cada tick del sistema…

Si el flag de X está activo…

Realizar tarea X

Reiniciar flag de X

Si el flag de Y está activo…

Realizar tarea Y

Reiniciar flag de Y

Si el flag de Z está activo…

Realizar tarea Z

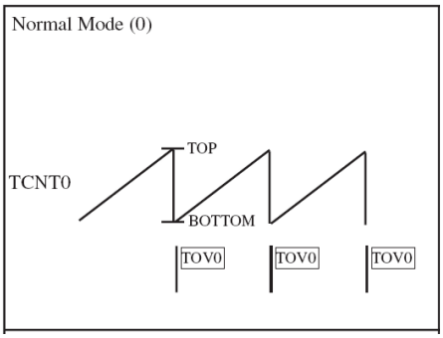
Reiniciar flag de Z

Poner MCU en modo bajo consumo

3 – Modos de Timer

3.1 – Normal

El contador TCNT se incrementa de a uno hasta sobrepasar el máximo valor del mismo, produciéndose una interrupción de Timer Overflow al activarse el flag TOV.



$$T_{OVF} = 2^8 * N * T_{clk} \quad \vee \quad T_{OVF} = 2^{16} * N * T_{clk}$$

$$f_{OVF} = \frac{f_{CLK}}{2^8 * N} \quad \vee \quad f_{OVF} = \frac{f_{CLK}}{2^{16} * N}$$

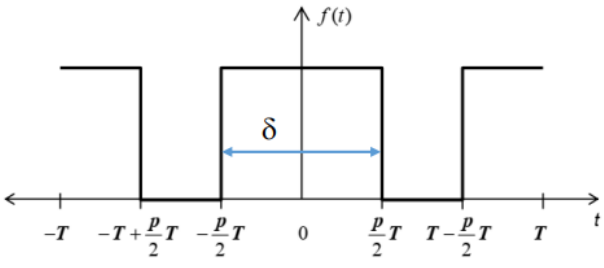
Las fórmulas anteriores valen siempre que TCNT se inicie en cero. Por defecto, lo es.

3.2 – Pulse Width Modulation

Abreviado como “PWM”, es una técnica de modulación digital. Tiene como salida una señal digital periódica, con pulsos de un ancho determinado. Se denomina ciclo de trabajo p al porcentaje de tiempo en que la señal está activa respecto del período total.

La información útil se encuentra en el ancho del pulso. Permite obtener una señal analógica a partir de una señal digital, por ejemplo, para controlar dispositivos analógicos con salidas digitales.

Mediante su representación en series de Fourier, se obtiene que los términos armónicos del seno son nulos ($B_n = 0$), y que el valor medio de la señal es proporcional al ciclo de trabajo:



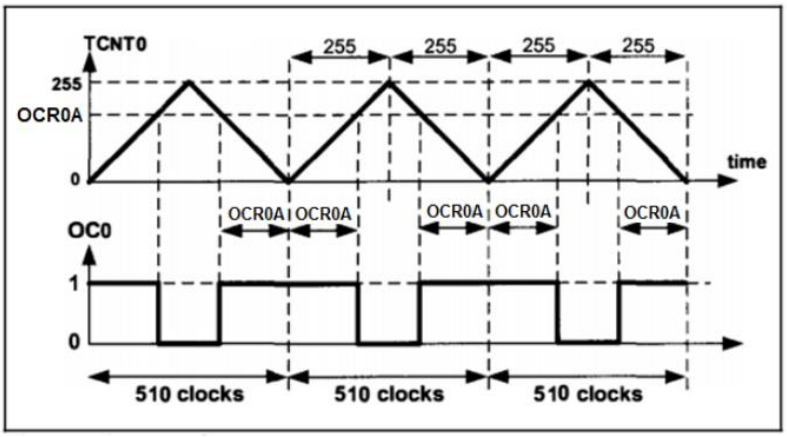
$$A_0 = \frac{1}{2T} \int_{-T}^{T} f(t)\, dt$$

$$A_0 = V_{DD} * p$$

La señal PWM tiene una componente de continua de amplitud A_0 (valor medio) que es proporcional al ciclo de trabajo, y las armónicas de amplitud A_n

3.2.1 – Fase Correcta

El contador TCNT no es reiniciado a cero cuando alcanza el valor máximo, sino que comienza a decrecer hasta alcanzar nuevamente el cero, repitiéndose el ciclo. Se utiliza OCR en ambas partes para los flancos. Mediante los bits COMnA0 y COMnA1 se puede elegir modo invertido (arranca en bajo) o no invertido (arranca en alto).



La frecuencia en este modo es: $F_{PWM} = F_{oscilador}/(2 * TOP * N)$

$$Duty\ Cycle = \frac{2 * OCRnA}{2 * TOP} * 100 \quad \vee \quad Duty\ Cycle = \frac{TOP - OCRnA}{TOP} * 100$$

Modo no invertido — Modo invertido

Para cualquiera de los 2 modos, es posible tener 0% o 100% de ciclo de trabajo.

3.2.2 – Fast PWM

El contador TCNT se incrementa de a uno hasta un valor máximo TOP. La señal tiene un flanco en TCNT = OCR, y luego otro cuando TCNT = TOP.

COM0A1=1
COM0A0=0
no invertido

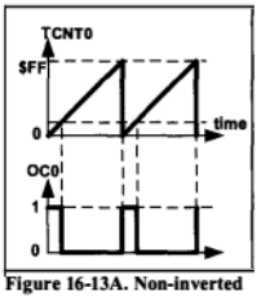


Figure 16-13A. Non-inverted

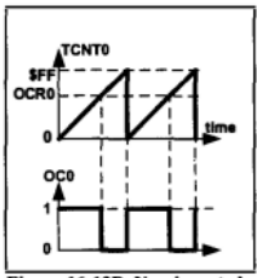


Figure 16-13B. Non-inverted

La frecuencia es:

$$F_{PWM} = \frac{F_{oscilador}}{(TOP + 1) * N}$$

En Timer 0, TOP es 255.

$$Duty\ Cycle = \frac{OCR + 1}{TOP + 1} * 100 \quad \vee \quad Duty\ Cycle = \frac{TOP - OCR}{TOP + 1} * 100$$

Modo no invertido (> 0%) — Modo invertido (< 100%)

3.3 – Clear Timer on Compare Match

Abreviado como “CTC”, es un modo donde el contador TCNT se incrementa de a uno hasta un máximo dado por el registro de comparación OCR. Luego vuelve a 0.

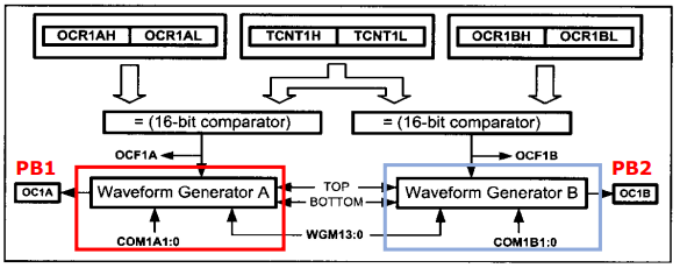
$$T_{CTC} = (OCR + 1) * N * T_{clk} \quad \Leftrightarrow \quad f_{CTC} = \frac{f_{CLK}}{(OCR + 1) * N}$$

$$T_{min} = (0 + 1) * N * T_{clk} = N * T_{clk} \qquad \wedge \qquad T_{max} = T_{OVF}$$

Las fórmulas anteriores valen siempre que TCNT se inicie en cero.

3.4 – Output Compare

En Normal y CTC, los bits COMnA0 y COMnA1, así como también los bits del tipo B, permiten elegir el comportamiento de una señal generada no PWM.

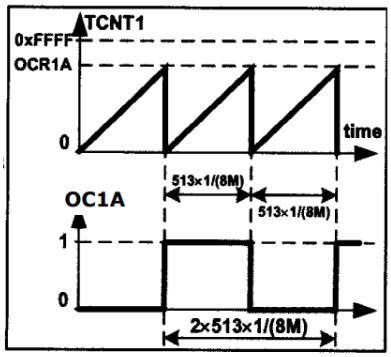


Si ambos bits son 0, entonces no hay señal generada. Para las señales cuadradas solo se activa el bit COMnA0 (o B0) según el pin (Toggle on compare match).

Siempre los flancos ocurren cuando el contador TCNT coincide con el registro OCR. La diferencia entre usar modo Normal y CTC para este objetivo tiene que ver con el periodo de la señal cuadrada. En Normal, es independiente del valor OCR, ya que el TOP es quien define el periodo, digamos que OCR es “la fase”. En CTC, TOP = OCR.

- Expresión general:

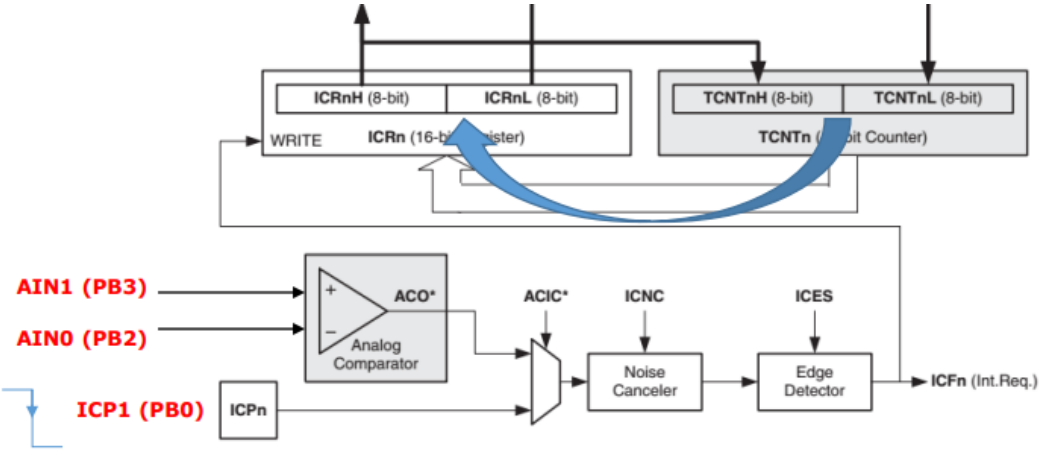
$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$



Ejemplo: si quiero hacer sonar la nota DO (311 Hz), si el sistema corre a 16 MHz, puedo elegir N y OCR. Para N = 1, $OCR = \frac{16\ MHz}{2*1*311\ Hz} - 1 \cong 25722$ (Timer 1).

3.5 – Capturador de entrada

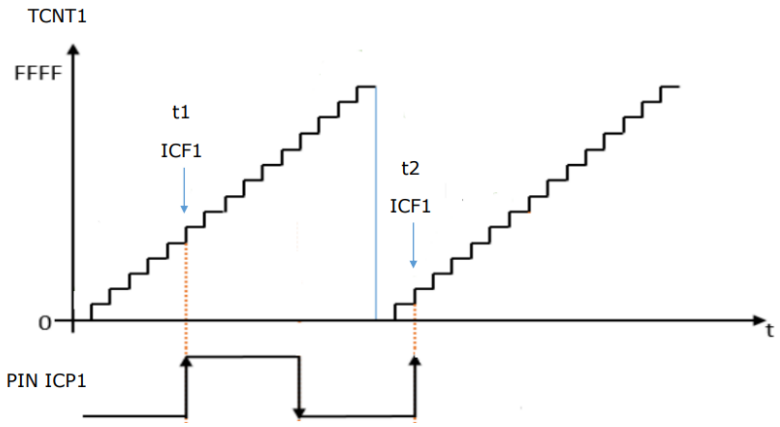
Para medir el período de una señal incógnita, se debe conectar dicha señal al terminal ICP1 del Timer 1. Cuando ocurre un flanco (de subida o de bajada según como esté configurado), se copia el valor del contador TCNT1 al registro ICR1 y además se activa el Flag ICF1, generando una interrupción de aviso de captura si está habilitada.



Luego, deben guardarse los valores de ICR1 obtenidos en dos flancos, y calcular:

$$T = (ICR1_{t2} - ICR1_{t1}) * N * T_{clk} \quad \Leftrightarrow \quad f = \frac{f_{CLK}}{N * (ICR1_{t2} - ICR1_{t1})}$$

Se observa que la frecuencia de la señal incógnita debe ser menor a la del MCU.



Recordar que en Timer 1, el contador toma valores del 0 al $2^{16} - 1 = 65535$

La resolución es: $\Delta T_{min} = DIF_{min} * N * T_{clk} = \pm N * T_{clk}$

El mínimo medible es: $T_{min} = (1 - 0) * N * T_{clk} = N * T_{clk}$

El máximo medible es: $T_{max} = 65535 * N * T_{clk}$

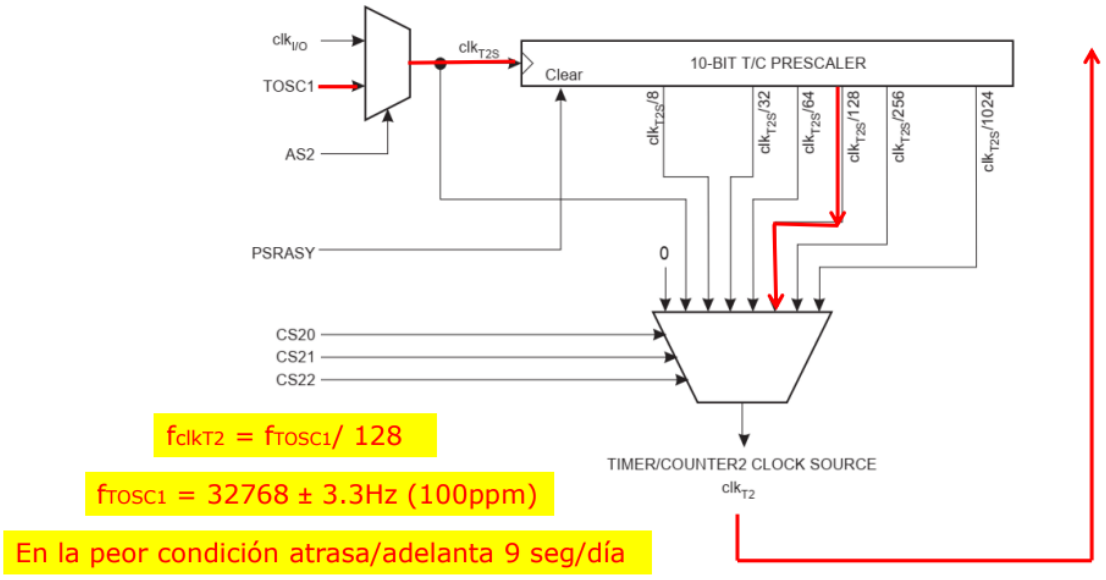
3.6 – Real Time Clock

Abreviado como “RTC” es un reloj que permite guardar la hora actual, con una mayor precisión, al poseer un oscilador de cristal, en general de frecuencia 32,768 kHz, justamente 2^{15} ciclos por segundo. La frecuencia, al ser baja, permite ahorrar energía.

En ATmega328P, está disponible para utilizarlo en Timer 2 (TCNT de 8 bits). El registro para configurarlo es ASSR (registro de estado asincrónico), con el bit AS2.

El preescaler necesario para que el Timer 2 desborde cada 1 segundo es:

$$f_{OVF} = \frac{f_{clk}}{2^8 * N} = 1\ Hz \quad \Leftrightarrow \quad N = \frac{32768\ Hz}{2^8 * 1\ Hz} = 128 = 2^7$$



Se pueden activar las interrupciones de desborde, que borran el Flag TOV2 automáticamente, o bien borrarlo manualmente escribiendo un 1: TIFR2 |= (1<<TOV2)

3.7 – Watchdog Timer

Es un mecanismo de protección ante fallas de software o hardware. Su función es contar los pulsos de reloj hasta un valor programado, para luego generar una interrupción o un RESET al alcanzarlo, es decir, cuando se produce un timeout. Dicho contador debe ser reiniciado por el software mediante la instrucción WDR (Watchdog Reset).

Cuenta con un prescaler de un oscilador de 128 kHz, que se puede elegir a partir de los bits WDP. La acción de interrumpir, reiniciar o ambas también es configurable.

4 – Máquina de Estados Finito

Es un modelo compuesto por un conjunto finito de estados, un estado inicial, un conjunto de estados finales, entradas, funciones de transición y salidas. Se describen por su expresión matemática (tuplas), diagramas de estados o tabla de transiciones (las filas son los estados actuales, las columnas las entradas, y las celdas el siguiente y salida).

La diferencia entre Moore y Mealy está en las salidas. En Moore sólo depende del estado actual, mientras que en Mealy depende del estado actual y las entradas.

Existen varias formas de implementarlas: MEF en software (por tablas) pudiendo ser temporizada (actualizada periódicamente) o no, por switch-case (pudiendo ser varios si hay más de 1 entrada, notar que en Moore las salidas van fuera de éstos), o con punteros a función (almacenados en un vector, lo cual permite que el tiempo de acceso a la función de estado sea el mismo para todos, a diferencia de switch-case donde afecta el orden).

Si las MEF son temporizadas, es importante que la función *MEF_Update* sea no bloqueante, lo mismo si son cooperativas (una se ejecuta después de otra).

4.1 – Pulsador con rebote

El efecto rebote aparece cuando los pulsadores son accionados. Sucede que los contactos deben moverse físicamente desde una posición a otra, lo cual produce un rebote mecánico (el circuito se abre y se cierra múltiples veces) hasta que se estabilizan en la nueva posición. Es decir, al pulsar o soltar el botón, se producen fluctuaciones en la señal que pasa por sus contactos, leyéndose como 0 o 1 múltiples veces en poco tiempo.

Una de las formas de resolver este problema es con MEF Mealy de 4 estados:

1) Button Up, cuando se mantiene en nivel alto. Cualquier flanco lo pasa a Falling.
2) Button Falling. Se pasa al estado Up o Down según el próximo valor leído.
3) Button Down, cuando se mantiene en nivel bajo. Un flanco lo pasa a Raising.
4) Button Raising. Se pasa al estado Up o Down según el próximo valor leído.

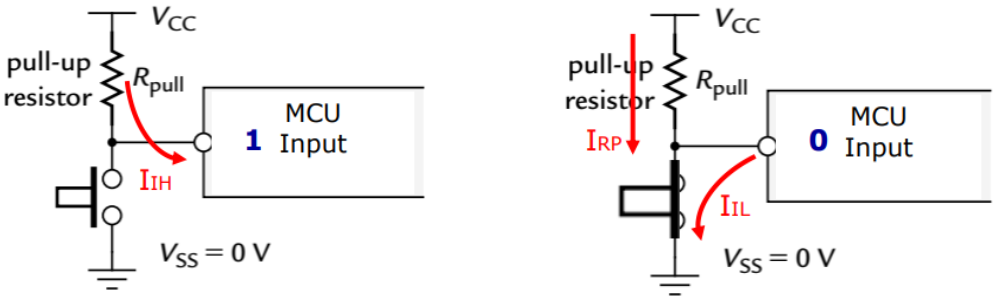
Las salidas importantes son las siguientes (caso Pull-Up, 0 es presionado).

1) Se presionó el botón, cuando pasa del estado Falling a Down.
2) Se soltó el botón, cuando pasa del estado Raising a Up.
3) Se mantiene presionado (sigue en Up) o en reposo (sigue en Down).

5 – Configuraciones de un pin

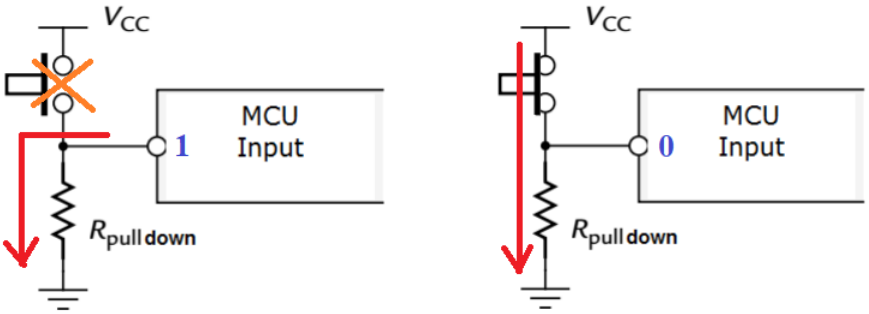
Mediante los registros DDRx, se puede establecer el comportamiento de cada pin del puerto x (A-D) modificando su bit asociado. Para poder escribir en un pin, primero debe escribirse un 1 en su bit correspondiente de DDRx, luego podrá asignarse el valor deseado en el registro PORTx. Por otra parte, para leer de un pin, primero se escribe un 0 en su bit de DDRx, y luego se lee el valor del registro PINx.

Por otra parte, existe el modo Pull-Up para las lecturas, para lo cual se escribe un 0 en el bit de DDRx y luego se escribe un 1 en PORTx. Se utiliza una resistencia que está incorporada dentro del MCU, que afecta el funcionamiento de la siguiente forma:



Por ejemplo, con un pulsador, si sus contactos están abiertos, para evitar que quede abierto al “aire”, circula una corriente despreciable desde VCC a la entrada del pin, por lo tanto, la tensión en el pin está en alto y se interpreta como “1” (inverso a sin Pull-Up). El otro caso es cuando los contactos están cerrados, entonces circula una corriente de VCC a tierra, por lo tanto, en la entrada del pin está en nivel bajo “0” (directo a tierra).

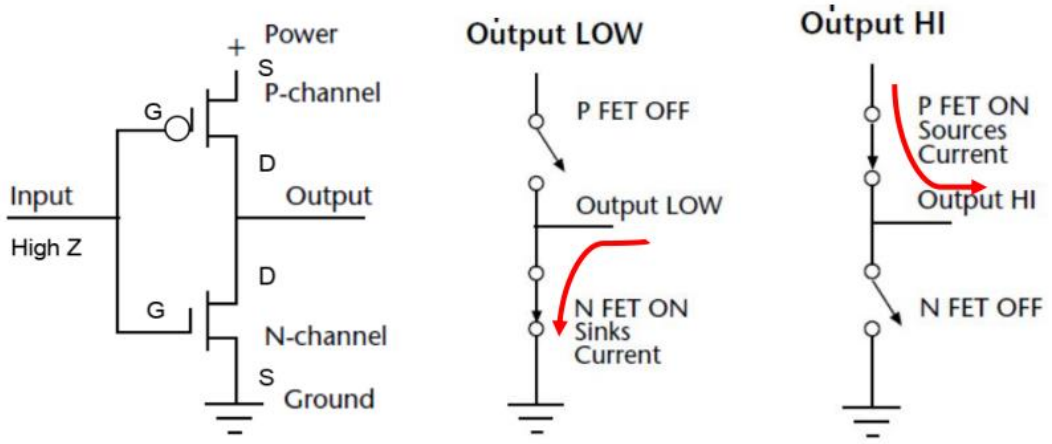
También se puede optar por una resistencia Pull-Down, externa al MCU:



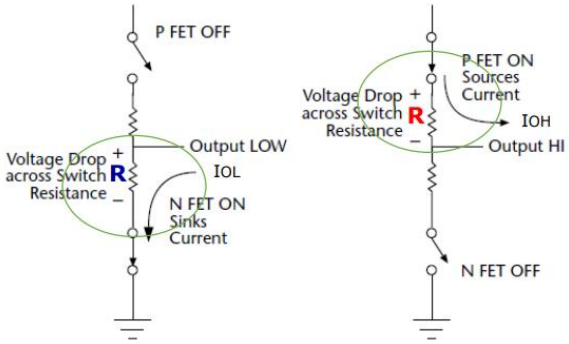
Cuando los contactos están abiertos, circula una corriente desde el pin hasta tierra, pasando por la resistencia de Pull Down, en el pin hay un nivel alto o “1”. En cambio, si los contactos están cerrados, circula una corriente desde VCC a tierra, quedando el pin en un nivel bajo o “0” al circular una corriente despreciable por él. ¡Leo igual que Pull-Up!

5.1 – Inversor CMOS

Idealmente, está compuesto de 2 transistores MOSFET (P y N):



En el caso real, ahora existe una caída de tensión para cada nivel:

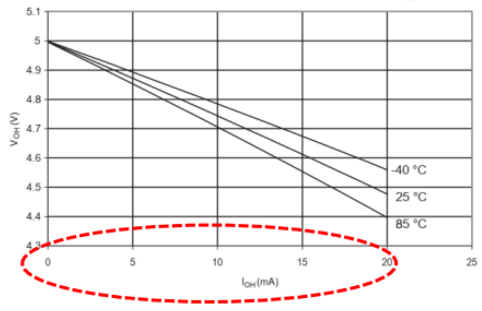


$$V_{OL} = I_{OL} * R < 0.9V$$

$$V_{OH} = V_{CC} - I_{OH} * R > 4.2V$$

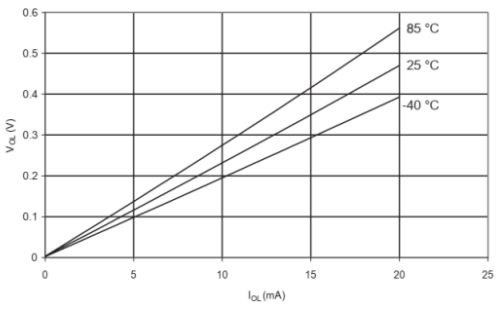
Es decir, el nivel bajo no es 0V, y el nivel alto no es 5V o VCC. Sin embargo, los mínimos y máximos son conocidos.

Figure 29-162. ATmega328P: I/O Pin Output Voltage vs. Source Current (V_{CC} = 5 V)



Zonas seguras de operación

Figure 29-160. ATmega328P: I/O Pin Output Voltage vs. Sink Current (V_{CC} = 5 V)



Zonas seguras de operación

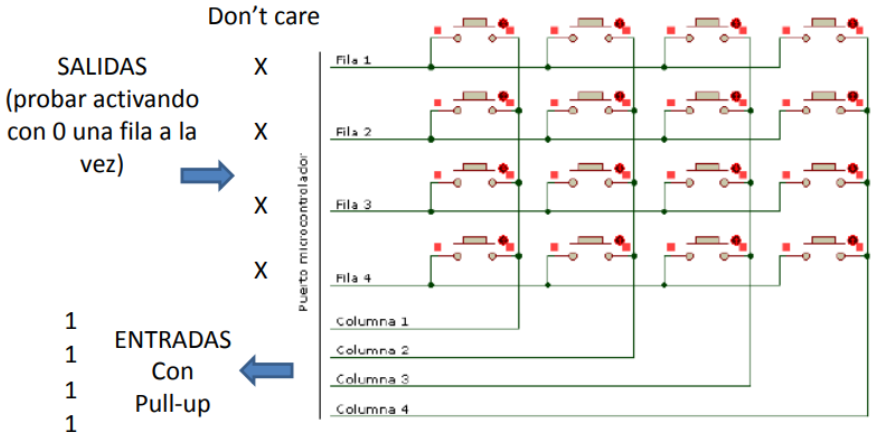
Si por ejemplo se conecta un LED a un pin del MCU, y se configura para encender el mismo cuando la salida es "1" (la corriente circula del pin a tierra), la corriente es:

$$I_{OH} = \frac{V_{OH} - V_{LED}}{R_{LED}} \qquad donde\ R_{LED}\ se\ elige\ para\ no\ quemar\ el\ LED$$

5.2 – Teclado matricial

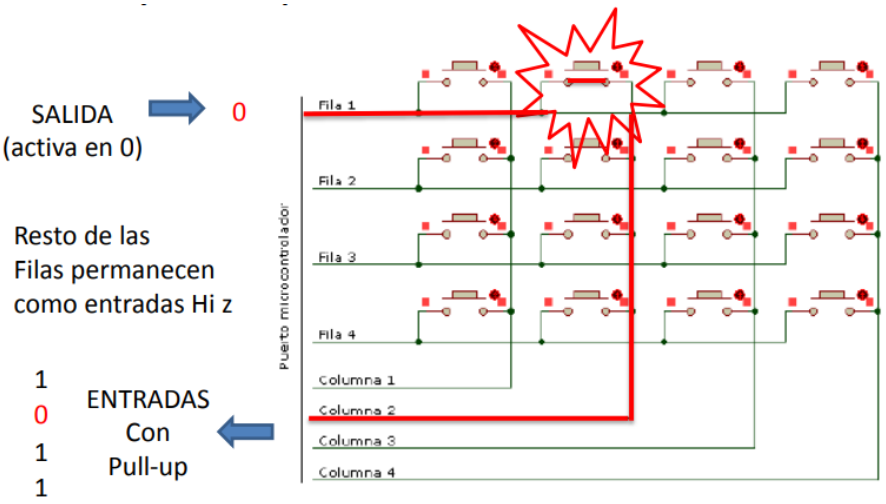
Es una matriz de pulsadores, distribuidos en filas y columnas. Por cada fila y cada columna se conecta la misma a un terminal del MCU (si es 4x4, son 8 pines utilizados).

Se deben configurar las columnas como entradas con Pull-Up (o Pull-Down) y las filas como salidas. Recordemos que en Pull-Up, si los contactos están abiertos se lee un nivel alto "1" como entrada del pin. Para mantener dicho funcionamiento, en las salidas, que reemplazan a "tierra", debe escribirse un "0" una a la vez para encuestar.



Al escribir un "1" en las salidas, es como si se ignorara el estado del pulsador para dicha fila y columna, ya que si está presionado la tensión es la misma en ambos extremos, pero al escribir un "0", si un pulsador está presionado habrá una caída de tensión por dicha rama (contactos cerrados), haciendo que en la columna se lea un nivel bajo "0".

Si en toda una fila no hay ningún pulsador presionado, la salida correspondiente estará en alta impedancia, ya que el circuito está abierto.



6 – LCD

Es un dispositivo que permite mostrar información en formato de texto. Posee un pin RW, que permite elegir si la siguiente operación es de escritura (0) o de lectura (1), luego tiene 8 pines D0-D7 donde se reciben o escriben los bits, un pin Enable (E) que al recibir un flanco de bajada realiza la operación según el valor de RW, y por último un pin de Register Select (RS) para seleccionar el de comandos (0) o el de datos (1).

Además de los 2 registros mencionados, posee una memoria RAM dividida en dos partes: CGRAM que almacena la fuente de 8 caracteres personalizados, y DDRAM que contiene los datos a mostrar en el LCD en formato ASCII. Por último, existe un cursor que apunta a las direcciones de CGRAM o DDRAM, por ejemplo, al escribir un dato en el registro de datos, se escribirá donde apunta este cursor.

7 – RESET

Es un evento (señal digital) que inicializa el sistema en un estado por defecto, perfectamente conocido. Se interpreta como un pedido de interrupción especial, que no se puede deshabilitar ni tampoco enmascarar. En ATmega328P tiene 4 fuentes:

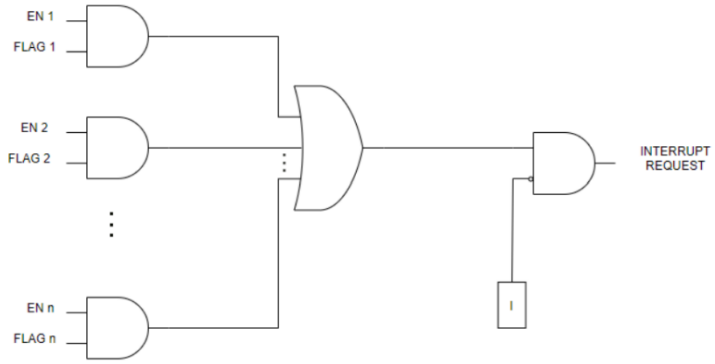
- ✓ POWER ON: circuito que genera la señal RESET al encender el sistema.
- ✓ BROWN OUT: es un circuito monitor de alimentación que genera la señal RESET si VCC se encuentra fuera de los márgenes seguros de operación.
- ✓ RESET EXTERNO: a través de un pin, por ejemplo, conectado a un pulsador.
- ✓ WATCHDOG: cuando su contador alcanza un valor programado por timeout.

7.1 – Mecanismo

1) El controlador de interrupciones suspende la ejecución de instrucciones CPU.
2) Se inicializan todos los registros I/O a sus valores por defecto.
3) Se espera que la alimentación y la fuente de reloj se estabilicen.
4) Se inicializa el contador de programa con el vector de RESET.
5) Se revisa el MCU Status Register para conocer la causa del RESET.

8 – Interrupciones

Luego de un RESET, las demás interrupciones son enmascaradas. Cada periférico posee bits específicos para habilitar o deshabilitar la generación de interrupciones. Para que la CPU pueda aceptar interrupciones, el programador debe habilitarlas globalmente.



Las instrucciones sei() y cli() escriben un 1 o 0 en el bit I del registro de estado SREG para habilitar o no globalmente.

Si ocurren pedidos mientras I=0 quedarán pendientes para cuando se habilite I=1.

8.1 – Mecanismo

1) Cuando un pedido es aceptado, la CPU finaliza la instrucción actual y almacena el contador de programa (PC), que apunta a la próxima instrucción, en la pila.
2) Se deshabilita la máscara global de interrupciones, es decir, I = 0.
3) Se busca la posición memoria asignada a la interrupción en la tabla de vectores.
4) Se reemplaza el PC para ejecutar el salto a la ISR correspondiente.
5) La CPU ejecuta la ISR hasta encontrar la instrucción de retorno RETI.
6) Se recupera el contador de programa de la pila y se habilita la máscara I = 1.

8.2 – Prioridad

Si hay varios pedidos en un mismo instante, se elige como interrupción a la que aparece primero en la tabla de vectores de interrupción, es decir, lo determina el orden.

8.3 – Tipo de funcionamiento

Las interrupciones pueden ser nivel o por flanco. En el primer caso, el periférico que la genera se encarga de colocar y mantener el nivel de la línea para que el MCU atienda, y luego el periférico es quien también libera el nivel de línea si fue atendido.

En las interrupciones por flanco, el periférico produce un flanco en la línea, queda registrado en un Flip Flop a modo de Flag (tiene memoria). Luego es el MCU quien debe limpiar el Flag luego de ser atendida, sin necesidad de comunicárselo al periférico.

8.4 – Tipo según origen

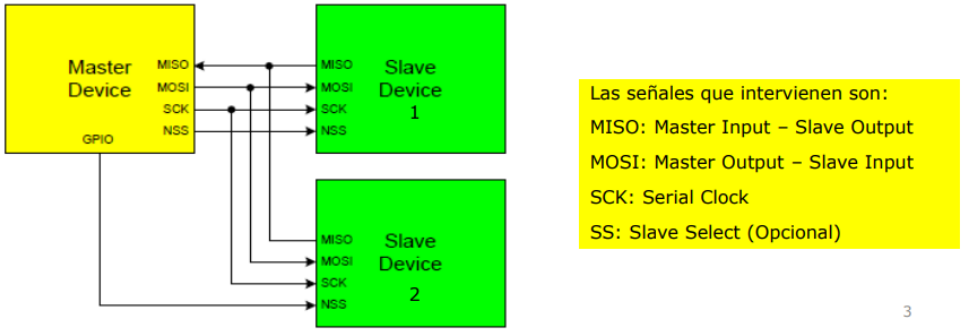
Las interrupciones pueden ser internas (producidas por los periféricos integrados en el mismo chip, como Timer), o externas (producidas por dispositivos conectados al MCU por los terminales INT0 e INT1, que sean capaces de generarlas).

En el registro EIMSK se habilitan y deshabilitan las de INT0 o INT1. Luego, en el registro de control se configura el tipo de activación (cuando se mantiene en nivel bajo, o cuando se produce un flanco de subida, o de bajada, o cualquiera de los dos).

9 – Interfaz Periférico Serie

Abreviado como “SPI”. Es una interfaz que permite la comunicación serie full-duplex sincrónica con otros periféricos o microcontroladores, con tasas del orden Mbps.

Que sea Full-Duplex significa que la recepción y transmisión de datos se realizan de manera simultánea con 2 buses, y que sea sincrónico significa que se utiliza una señal de reloj, que en este caso es “SCK”, generada por un Master y distribuida a cada Slave.



El funcionamiento es el siguiente: el maestro y cada esclavo poseen un registro de desplazamiento de datos. En una transferencia, por cada pulso de reloj se intercambian datos entre el registro de Master y Slave, y se actualizan en el flanco opuesto.

Los pulsos de reloj se pueden programar de 4 modos, según la polaridad y fase.

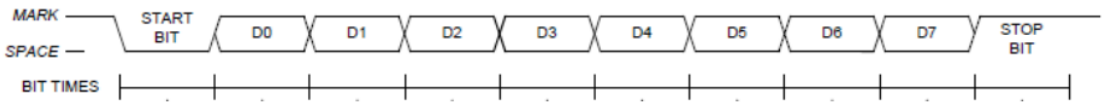
Como ventajas se tiene que el protocolo es flexible (libre tamaño de bloques, de la trama de bits, del orden de transmisión), la implementación en hardware es simple (no requiere mecanismos de arbitraje, reloj único), y hay versiones aún más eficientes.

Como desventajas se tiene que el direccionamiento se realiza por Slave Select, el maestro no puede saber si el esclavo está desconectado, y sólo puede existir 1 maestro.

10 – Transmisor Receptor Universal Asincrónico

Abreviado como “UART”, es un periférico integrado en el MCU que permite la comunicación asincrónica con otros dispositivos, con tasas estándar hasta 38400 bps.

Que sea asincrónica significa que el transmisor y receptor no utilizan un reloj en común, ya que se asume una tasa de transferencia conocida, y además la trama de datos contiene un bit de inicio y un bit de fin, como el formato 8N1 (8 bit, no parity, 1 stop).

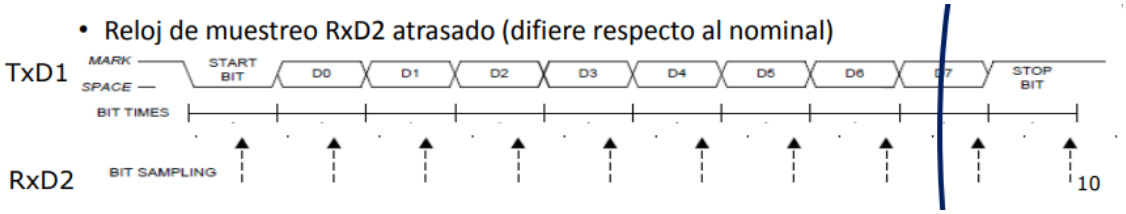


El ATmega328P incorpora un USART, que además permite transmitir un reloj con el objetivo de aumentar al máximo la tasa de transmisión. Tiene entonces 3 partes:

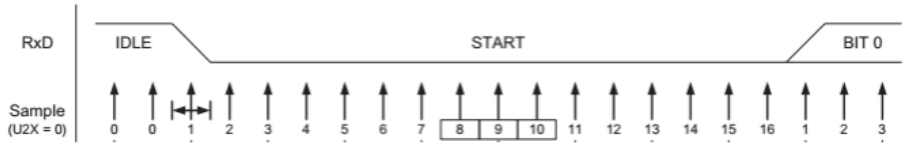
- Generador de reloj: Baud Rate Generator según frecuencia de MCU y UBRR.

$$Baud\ Rate = \frac{F_{oscilador}}{(UBRR + 1) * 16} \qquad UBRR\ posee\ 12\ bits$$

Si por ejemplo se desea transmitir a 9600 bps, para 16 MHz el valor más cercano es UBRR = 103. El error no puede superar el 5% (10 x mitad de tiempo de bit).



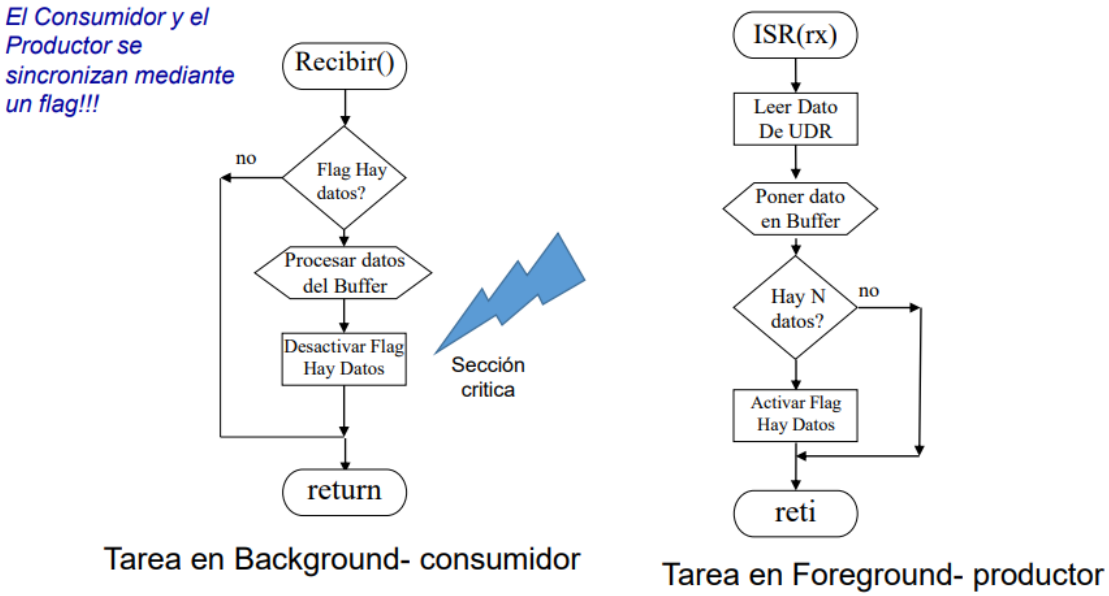
- Transmisor: posee un registro UDR, que si está vacío (flag UDRE = 1), el usuario puede cargar un dato para transmitir, que será copiado a un Shift Register.
- Receptor: posee un registro UDR de sólo lectura, que recibe el contenido de un Shift Register, y al completarse se activa el flag RXC. Se muestran los datos de entrada a una tasa 16 veces mayor a la seleccionada para poder detectar el bit de comienzo y sincronizarse con el centro de bits de los datos.



Para 9600 bps y 8N1, la transmisión tarda 1,04 ms, y la recepción 0,065 ms.

10.1 – Modelo Productor Consumidor

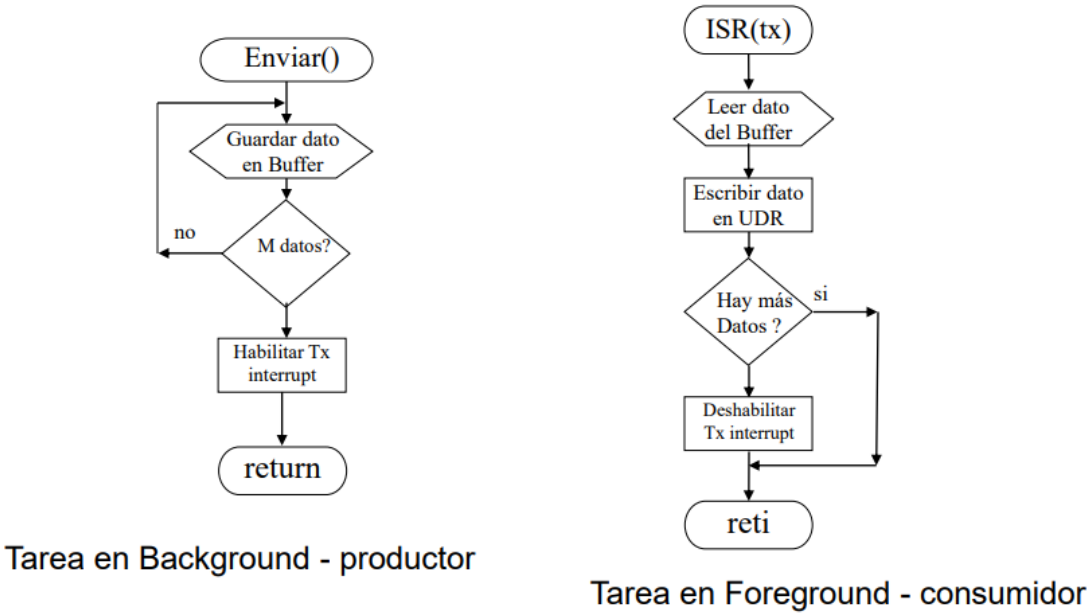
Consiste en el uso de estructuras FIFO (búfer) que recibe datos producidos por tareas y luego los entrega a los consumidores. Ambas partes pueden operar con una gran cantidad de datos y trabajar a diferentes velocidades. En la arquitectura Background – Foreground los productores pueden ser ISR y los consumidores serían las tareas del Loop, con comunicación entre ellas mediante Flags. En la recepción un ejemplo es:



Tarea en Background- consumidor

Tarea en Foreground- productor

En la transmisión se reemplaza el Flag por habilitación de la interrupción:



Tarea en Background - productor

Tarea en Foreground - consumidor