Capítulo 12

Manejo de interrupciones

Capítulo Web

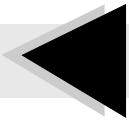
Competencias

Presentar la descripción y manejo de interrupciones del sistema Arduino, así como sus aplicaciones en control de procesos en tiempo real.

Desarrollar habilidades en:

- Descripción de las interrupciones del sistema Arduino.
- Tipos de interrupciones.
- Subrutinas de servicio de interrupciones.
- Control de procesos en tiempo real.
- Aplicaciones en control automático.

12.1 Introducción



Las interrupciones son recursos o mecanismos del microcontrolador para responder a eventos, permitiendo suspender temporalmente el programa principal, para ejecutar una subrutina de servicio de interrupción (ISR por sus siglas en inglés *Interrupt Service Routines*); una vez terminada dicha subrutina, se reanuda la ejecución del programa principal.

Las interrupciones se generan cuando dispositivos periféricos conectados a la tarjeta electrónica solicitan enviar información al microcontrolador, esto puede ser de manera asíncrona. También el proceso de interrupción se puede generar de manera periódica, es decir por medio de una señal digital (por ejemplo de un milisegundo de período) conectada a un pin específico del microcontrolador (INT0 o INT1) se puede atender tareas determinadas como adquisición de datos, monitoreo de sensores, cálculos numéricos, envío de comandos al robot, etc.

Polling

Representa una técnica antigua de interrupción, es un mecanismo de sondeo continuo, el cual consiste en que el mismo microcontrolador se encarga de monitorear (lecturas) el estado o información de un evento; el principal inconveniente es que los eventos no pueden ser monitoreados en forma periódica, ya que el microcontrolador consume tiempo en ejecutar diversas instrucciones que lo distrae de dicha actividad de sondeo, resultando una técnica ineficiente. La solución a esta problemática ha sido la incorporación al microcontrolador de un dispositivo controlador de interrupciones para atender cualquier tiempo de interrupción.

Por ejemplo, para monitorear el estado lógico que tiene un pin, habrá que leer su valor continuamente; si en un momento específico este pin tiene 5 V y no se lee, debido a que el microcontrolador está ejecutando otras instrucciones, se corre el enorme riesgo que cuando se lea, el valor del pin ha cambiado sin ser detectado por el microcontrolador, perdiendo información valiosa del sensor o sistema.

Monitorear información de un evento usando **polling**, genera un alto riesgo para perder lecturas o información del sistema, por lo tanto hoy en día, esta técnica resulta inadecuada e ineficiente.

En lugar de utilizar **polling**, se emplea un dispositivo denominado controlador de interrupciones (también conocido como unidad de interrupciones), la arquitectura AVR de los microcontroladores que se utilizan las tarjetas Arduino, incluyen una unidad de interrupciones (ver el diagrama a bloques de la figura 12.1), para controlar la solicitud de eventos o dispositivos periféricos.

Controlar la solicitud de interrupciones a través de la unidad de interrupciones evita consumir tiempo del microcontrolador para sensar o monitorear por sí mismo la información de un evento. La unidad de interrupciones le da versatilidad al microcontrolador.

Cuando se genera una interrupción, la unidad de interrupciones indica al microcontrolador que un evento requiere solicitud de interrupción, el microcontrolador consulta los registros de la unidad de interrupciones para determinar qué tipo de interrupción se está generando. También guarda el estatus del contador de programa, con el número de interrupción se determina por medio de la tabla de vectores de interrupción (tabla indexada en forma prioritaria, ver tabla 12.1) con las direcciones de la memoria flash donde se ubican las rutinas de servicio de interrupción ISR, de esta forma se puede acceder y ejecutar el correspondiente código de la ISR. Las direcciones donde residen las rutinas de servicio de interrupción ISR (tabla de vectores) se encuentran en las primeras localidades de memoria del espacio de programa, iniciando en la localidad 0x0000 con la interrupción con mayor prioridad: RESET. La tabla 12.1 presenta las interrupciones del microcontrolador Atmega 328P (8 bits, arquitectura AVR) que forma la plataforma electrónica del modelo Arduino UNO.

> Las interrupciones representan un atributo importante para todo sistema empotrado que le permite atender procesos del mundo exterior sin descuidar actividades relevantes en la ejecución del programa del usuario.

Por ejemplo, en control de robots manipuladores se requiere conocer la posición actual del robot, esto significa que un conjunto de encoders (sensores de posición digitales optoelectrónicos) están conectados a los puertos digitales del microcontrolador y no tendría sentido que el microcontrolador estuviera sensando (**polling**) a cada encoder para detectar que ya tienen listo el dato de posición. Esto

sería un proceso ineficiente, y pérdida de tiempo innecesario. En su lugar, se utiliza la interrupción INT0 que después del RESET es la de mayor prioridad, garantizando la adquisición de datos y procesamiento de la información en tiempo y forma. Un disparo electrónico o señal de control del encoder indica al microcontrolador el momento exacto para realizar la lectura de posición del robot.

También puede haber eventos que sucedan de manera asíncrona y que requieran mayor prioridad que otras, es el caso de una interrupción que está realizando cálculos numéricos de un algoritmo de control para robots manipuladores, en ese momento sucede un paro de emergencia, esta última actividad tiene mayor importancia que cualquier otra, hay que activar los frenos mecánicos y cortar el suministro de energía a los servomotores del robot. Es decir, se suspende inmediatamente los cálculos numéricos y atender el paro de emergencia.

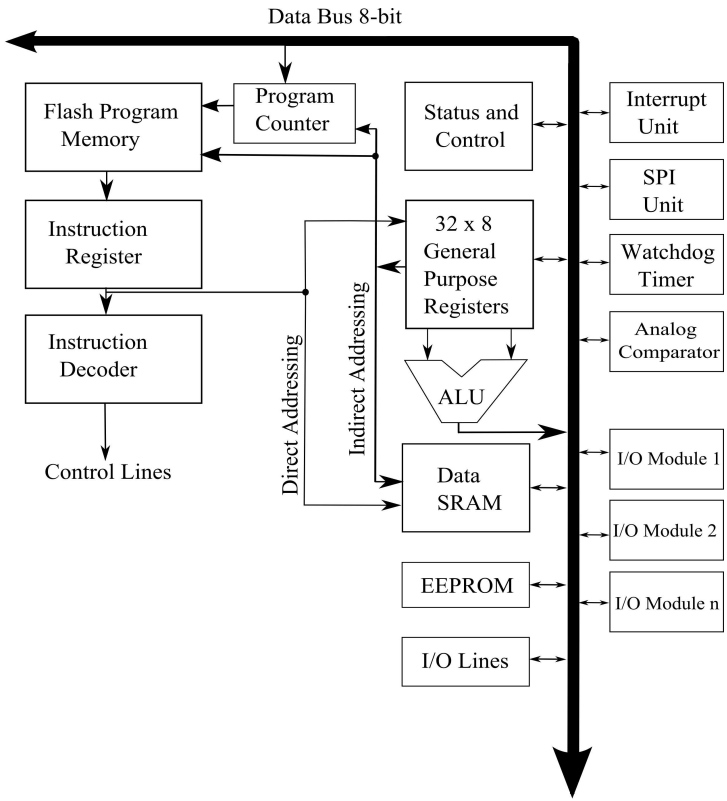
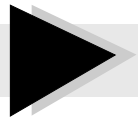


Figura 12.1 Diagrama a bloques del núcleo de microcontroladores AVR Atmega.



12.2 Tipos de interrupciones

En los microcontroladores Atmega hay dos tipos de interrupciones, el primer tipo corresponde a eventos externos que generan un estado lógico (cambio de voltaje) y también por transición como un pulso electrónico de disparo (*triggered*), la transición se detecta por flanco de subida en la señal periódica (de bajo hacia alto, es decir: LOW → HIGH) o por flanco de caída (HIGH → LOW); esto habilita una bandera de interrupción (*interrupt flag*), dependiendo de la prioridad de interrupción, el contador del programa (*program counter*) toma la dirección de memoria de la tabla de vectores y salta a la localidad de memoria correspondiente donde se encuentra la rutina de la interrupción solicitada (*interrupt handling routine*).

De manera automática, por hardware se limpia la bandera de interrupción (*interrupt flag*); también se puede limpiar las banderas de interrupción por software, ya que tienen asociados sus respectivos bits de habilitación en el registro de estado (*status register*).

Similarmente, si más solicitudes de interrupción ocurren mientras se encuentra en proceso alguna interrupción, permanecerán en espera por orden de prioridad. Cuando el contador de programa sale de una interrupción, retornará al programa principal y ejecutará una o más instrucciones antes de atender alguna interrupción pendiente.

El segundo tipo de interrupciones corresponden a las interrupciones que pueden ser cambiadas o reasignadas por software en los pins del microcontrolador, estas interrupciones no necesariamente tienen banderas de interrupción.

12.2.1 Procesamiento de la interrupción

Cuando se genera una interrupción, el microcontrolador termina la ejecución de la instrucción en curso, salva el estado de registros y banderas, así como la dirección de memoria del contador de programa, entonces salta a la dirección de memoria

donde está almacenada la rutina de servicio de interrupción ISR para ejecutar dicha rutina que tiene como objetivo atender al dispositivo que generó la interrupción. Una vez que ha finalizado la rutina de la interrupción, el microcontrolador restaura el estado que había guardado y retorna al programa principal a partir de la siguiente instrucción donde se quedó el contador del programa.

La forma en que funcionan las interrupciones en los microcontroladores Atmega es por prioridades o jerarquía, la figura 12.2 muestra el diagrama de flujo de la interrupción de mayor prioridad como es el RESET (por hardware: ya sea por el botón de reset o al conectar la fuente de alimentación; también se puede generar esta interrupción por software). En la figura 12.2, si el proceso 1 solicita servicio de interrupción mientras se encuentra el microcontrolador atendiendo la interrupción del proceso 0, entonces su ejecución dependerá del nivel de prioridad, es decir si la interrupción del proceso 1 es de menor prioridad, tendrá que esperar a que termine la interrupción del proceso 0, de otra manera la interrupción del proceso 1 se ejecutará inmediatamente, suspendiendo momentáneamente la interrupción del proceso 0, al terminar la tarea del proceso 1, retornará a las actividades pendientes de la interrupción anterior.

Para trabajar control de procesos en tiempo real, los timers del microcontrolador pueden ser programados para generar una onda o señal cuadrada periódica y producir una frecuencia de muestreo adecuada para realizar las siguientes actividades: adquisición de datos de los sensores de posición, generar la señal de error de posición, cálculos de las operaciones matemáticas que involucra la ley de control (ganancias proporciona, derivativa, funciones trigonométricas e hiperbólicas) y envío de la información como señal de comando a los servomotores del robot para producir desplazamiento mecánico. Este ciclo se repite, considerando que el tiempo de máquina del sistema digital para llevar a cabo todas las operaciones involucradas en la ISR es menor al período de muestreo, es decir control en tiempo real.

La incorporación de convertidores analógico/digital, puertos entrada/salida, timers, dispositivos de comunicación serial como USART y diversos periféricos a los microcontroladores Atmega, adicional a las características tecnológicas que poseen con arquitectura AVR, los hacen ideales para utilizarse como sistemas empotrados con amplias aplicaciones en ingeniería robótica y mecatrónica.

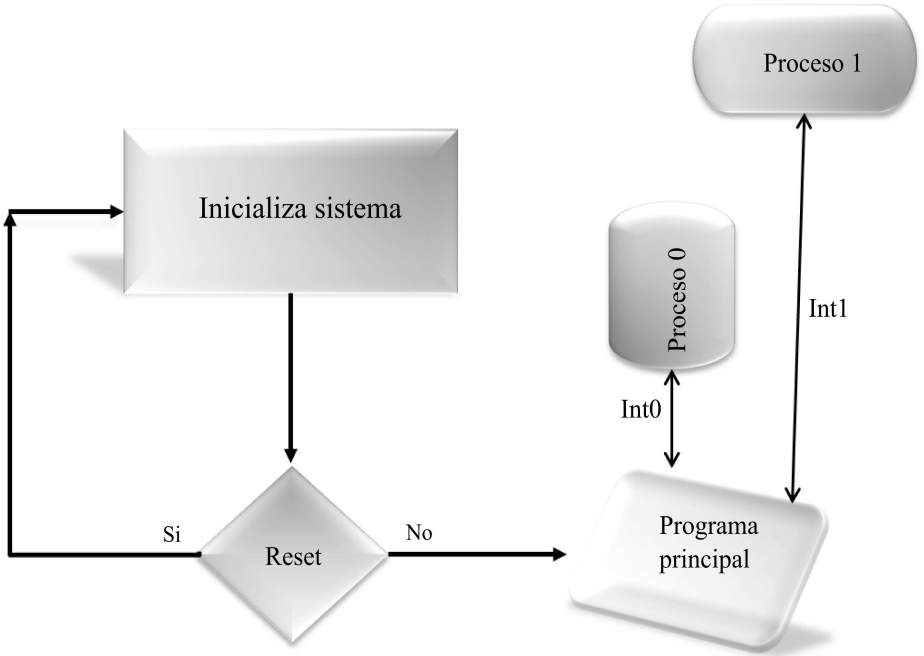


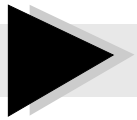
Figura 12.2 Jerarquía de interrupciones.

El uso de recursos compartidos de las tarjetas Arduino puede significar fuentes de error en hardware como en software, por ejemplo:

- En comunicación serial, cuando se envía una cadena de caracteres por el puerto serie y antes del envío se produce una interrupción que también utiliza este mismo puerto de comunicación los dos mensajes quedarán mezclados generando errores en la transmisión.
- En los puertos digitales el intercambio de información puede quedar contaminada cuando dos o mas interrupciones comparten los mismos pins digitales para envío de comandos o interface con dispositivos periféricos.
- La misma situación aplica con las variables globales, diversas interrupciones pueden modificar el valor de esas variables y provocar errores numéricos en otras funciones o subrutinas. De ahí, que es recomendable utilizar en las variables globales el modificador **volatile**.

Tabla 12.1 Interrupciones del microcontrolador Atmega328.

Vector	Localidad	Fuente	Características	Identificador
1	0x000	RESET	Pin externo, reset al encender (*power-on*), apagón parcial (*brown-out*), reset generado por *watchdog system.*	
2	0x002	INT0	Solicitud de interrupción externa 0, pin 2.	INT0_vect
3	0x004	INT1	Solicitud de interrupción externa 1, pin 3.	INT1_vect
4	0x006	PCINT0	Solicitud de interrupción cambio de pin 0 (pins 8 al 13).	PCINT0_vect
5	0x008	PCINT1	Solicitud de interrupción cambio de pin 1 (pins A0 a A5).	PCINT1_vect
6	0x00A	PCINT2	Solicitud de interrupción cambio de pin 2 (pins 0 al 7).	PCINT2_vect
7	0x00C	WDT	Interrupción tiempo-fuera del Watchdog.	WDT_vect
8	0x00E	TIMER2 COMPA	Timer/Contador2 Match A.	TIMER2_COMPA_vect
9	0x010	TIMER2 COMPB	Timer/Contador2 compara Match B.	TIMER2_COMPB_vect
10	0x012	TIMER2 OVF	Sobreflujo del Timer/Contador2.	TIMER2_OVF_vect
11	0x014	TIMER1 CAPT	Captura de eventos Timer/Contador1.	TIMER1_CAPT_vect
12	0x016	TIMER1 COMPA	Compara Match A del Timer/Contador1.	TIMER1_COMPA_vect
13	0x018	TIMER1 COMPB	Compara Match B del Timer/Contador1.	TIMER1_COMPB_vect
14	0x01A	TIMER1 OVF	Sobreflujo del del Timer/Contador1.	TIMER1_OVF_vect
15	0x01C	TIMER0 COMPA	Compara Match A del Timer/Contador0.	TIMER0_COMPA_vect
16	0x01E	TIMER0 COMPB	Compara Match B del Timer/Contador0.	TIMER0_COMPB_vect
17	0x020	TIMER0 OVF	Sobreflujo del Timer/Contador0.	TIMER1_OVF_vect
18	0x022	SPI, STC	Transferencia serial completa SPI.	SPI_STC_vect
19	0x024	USART, RX	Recepción serial completa RX.	USART_RX_vect
20	0x026	USART, UDRE	Registro vacío de datos.	USART_UDRE_vect
21	0x028	USART, TX	Transmisión serial completa TX.	USART_TX_vect
22	0x02A	ADC	ConveISRón completa del ADC.	ADC_vect
23	0x02C	EE READY	Memoria EEPROM lista.	EE_READY_vect
24	0x02E	ANALOG COMP	Comparador analógico.	ANALOG_COMP_vect
25	0x030	TWI	Interface serial 2-cables (2-wire serial interface I2C).	TEI_vect
26	0x032	SPM READY	Listo para almacenar programa en memoria.	SPM_READY_vect



12.3 Rutinas de servicio de interrupción

Las rutinas o subrutinas de servicio de interrupción ISR son funciones que realizan actividades de adquisición de datos, por ejemplo: lectura de señales analógicas por medio de convertidores analógico digital, sensores digitales (encoders); también llevan a cabo, manipulación y procesamiento de datos, cálculo numérico, envío de información digital o comandos a través de puertos y convertidores digital analógico. Generalmente, este tipo de funciones o subrutinas no tienen parámetros o argumentos de entrada y tampoco retornan datos (salvo casos excepcionales).

Una característica deseable de las ISR es que deben tener el código necesario para llevar a cabo todas las necesidades de la tarea específica del proceso que atienden; es decir, se requiere que sea tan rápidas como sea posible.

Cuando el contador del programa atiende una interrupción indica al microcontrolador que suspenda temporalmente las instrucciones que está ejecutando en ese momento, y transfiera el contador del programa al tipo de interrupción donde se tiene una ISR con el código específico para llevar a cabo la actividad solicitada; después de terminar el código del tipo de IRS, el contador del programa retorna al conjunto de instrucciones que estaba ejecutando en el programa principal.

El sistema Arduino tiene un conjunto de funciones para manejo de interrupciones, como las que a continuación se describen.

```
attachInterrupt(num_int, ISR, mode)
```

Esta función específica la subrutina de servicio de interrupción **ISR** a ejecutar por el microcontrolador, se debe indicar el número de interrupción **num_int**; con esta información la función **attachInterrupt(...)** coloca en la tabla de vectores (indexada por **num_int**) la dirección de memoria donde reside la subrutina del usuario **ISR**, la cual no contiene parámetros o argumentos de entrada y ningún tipo de dato retorna.

La función **attachInterrupt(...)** no retorna datos.

La forma en que se detecta o activa la interrupción se especifica en el argumento **modo**, pudiendo ser cualquiera de las siguientes 4 constantes: **LOW**, **CHANGE**, **RISING** y **FALLING**; tal y como se muestra en la tabla 12.2.

Tabla 12.2 Modos de disparo o activación de las interrupciones.

LOW	Esta constante indica que se activa la interrupción cuando hay un estado LOW (0 V) en el pin asociado al tipo de interrupción.
CHANGE	Se dispara la interrupción cuando el pin cambia de valor de voltaje o estado.
RISING	Flanco de subida: cuando hay una transición de bajo (LOW) hacia alto (HIGH), se genera una solicitud de interrupción.
FALLING	Flanco de bajada: cuando existe una transición de alto (HIGH) hacia bajo (LOW), entonce se activa la interrupción.

Existe otra forma sintaxis de para la función **attachInterrupt(...)**, la cual se describe de la siguiente forma:

```
attachInterrupt(pin, ISR, mode)
```

Esta sintaxis sólo funciona para la tarjeta Arduino Due, donde el argumento **pin** representa el número de pin o puerto digital asociado al tipo de interrupción.

Adicional a las constantes de la tabla 12.2, sólo para la tarjeta Due, también admite la constante **HIGH**, indicando que el disparo de la interrupción sucede cuando hay un valor alto (**HIGH**) en el pin asociado al tipo de interrupción.

Los microcontroladores Atmega que se utilizan en las tarjetas Arduino tienen dos clases diferentes de interrupciones:

Externas: para los microcontroladores Atmega168/328 (modelos Arduino UNO, Nano, Duemilanove) tienen asociadas las interrupciones INT0 e INT1, mapeadas a los pins 2 y 3, respectivamente.

Estas interrupciones son muy rápidas y eficientes, ya que pueden ser disparadas por hardware en transiciones de subida y bajada, también por nivel de voltaje (0 V o LOW).

Interrupciones por cambio de pins para las tarjetas Arduino que se basan en microcontroladores Atmega, las interrupciones pueden ser habilitadas en las 20 señales de los microcontroladores Atmega168/328.

Para mayor información sobre cambio de pins para interrupciones consulte la librería **PinChangeInt**, cuya descripción se encuentra en la página:

playground.arduino.cc/Main/PinChangeInt

La tabla 12.3 presenta la asignación de pins al tipo de interrupción para los modelos de tarjetas Arduino: UNO, Ethernet, Mega2560 y Leonardo.

Tabla 12.3 Interrupciones en las tarjetas Arduino.

Modelo	int0	int1	int2	int3	int4	int5
UNO, Ethernet	pin 2	pin 3				
Mega2560	pin 2	pin3	pin 21	pin 20	pin 19	pin 18
Leonardo	pin 3	pin 2	pin 0	pin 1	pin 7	

Las siguientes tablas muestran en forma individual por modelo de tarjeta Arduino la forma en que son mapeados en la función **attachInterrupt(...)** el número de interrupción al número de pin digital.

Tabla 12.4 Asignación de pins digitales por interrupción del modelo Arduino UNO.

Attachinterrupt	Nombre	Pin (puerto digital)
0	INT0	pin 2 (D2)
1	INT1	pin 3 (D3)

Tabla 12.5 Asignación de pins digitales por interrupción del modelo Arduino Mega2560.

Attachinterrupt	Nombre	Pin (puerto digital)
0	INT4	pin 2 (D2)
1	INT5	pin 3 (D3)
2	INT0	pin 21 (D21)
3	INT1	pin 20 (D20)
4	INT2	pin 19 (D19)
5	INT3	pin 18 (D18)

Importante

El modelo Arduino Due tiene mayor capacidad de interrupciones, permite añadir funciones de interrupción en todos los pins, especificando el número de pin en la función **attachInterrupt()**.

Dentro de las funciones de servicio de interrupción (añadidas a través de **attachInterrupt()**), la función **delay()** no trabaja.

De igual forma para la función **millis()**, el valor que retorna no se incrementará.

Tabla 12.6 Asignación de pins digitales por interrupción del modelo Arduino Leonardo.

Attachinterrupt	Nombre	Pin (puerto digital)
0	INT0	pin 3 (D3)
1	INT1	pin 2 (D2)
2	INT2	pin 0 (D0)
3	INT3	pin 1 (D1)
4	INT6	pin 7 (D7)

```
detachInterrupt(num_int)
```

Esta función retira la dirección de memoria de la ISR del usuario, por lo que, deshabilita el número de interrupción de la tabla de vectores proporcionado en el argumento **num_int**.

```
detachInterrupt(pin
```

Esta sintaxis sólo aplica al modelo de tarjeta Arduino Due, donde **pin** indica el número de pin de la interrupción a deshabilitar.

Sugerencias para escribir una ISR

Cuando escriba una rutina de servicio de interrupción (ISR) tome en cuenta los siguientes aspectos claves:

- Mantener el código necesario y depurado dentro de la ISR.
- No usar funciones **delay(...)**.

Sugerencias para escribir una ISR

- No usar funciones **Serial.print(...)**.
- Las variables globales que se compartan con el programa principal y diversas funciones, se recomienda utilizar el modificador **volatile**.
- Variables compartidas en diferentes partes del programa.
- Dentro de la ISR no utilizar instrucciones que deshabiliten o activen interrupciones.

Habilitando/deshabilitando interrupciones

Las interrupciones pueden ser temporalmente deshabilitadas limpiando la bandera de interrupciones (con excepción a la interrupción RESET, la cual no puede ser deshabilitada).

`interrupts()`

Re-habilita interrupciones, cuando previamente las interrupciones fueron deshabilitadas con la función **noInterrupts()**.

Esta función no tiene parámetros o argumentos de entrada, ni retorna datos.

`noInterrupts()`

Esta función deshabilita interrupciones. No tiene parámetros de entrada, ni retorna datos.

```
interrupts();// habilita interrupción.
// También se puede usar la siguiente instrucción.
set(); // habilita la bandera de interrupciones.
```

Similarmente, para deshabilitar interrupciones:

```
noInterrupts();// deshabilita interrupción.
// Limpia la bandera de interrupción. cli();
```

Por ejemplo:

```
noInterrupts(); //deshabilita interrupción.
i++; //incrementa contador.
interrupts(); //activa interrupción.
```

Cuando se apaga (deshabilita) una interrupción, se asegura que temporalmente el contador **i** que está dentro de la ISR no incremente su valor.

Interrupciones vacías

Cuando se quiere generar una interrupción, pero no hacer nada en particular se puede utilizar **EMPTY_INTERRUPT**, por ejemplo:

```
EMPTY_INTERRUPT (PCINT1_vect);
```

volatile

Variables compartidas entre funciones ISR y funciones normales o con el lazo principal **loop()** deben ser declaradas usando el modificador **volatile** para indicarle al compilador que dicha variables podrían cambiar en cualquier tiempo.

Por lo que, el compilador deberá recargar la variable cuando se le haga referencia.

♣ ♣ Ejemplo 12.1

Usando la interrupción INT0 (pin 2 de la tarjeta Arduino UNO) realizar el incremento de un contador t como múltiplos de $h = 0.001$, es decir: $t_k = t_{k-1} + h$. Exhibir el resultado en el monitor serial del ambiente de programación Arduino.

Solución

En este ejemplo, se utiliza el pin2 de la tarjeta Arduino UNO, el cual está asociado a la interrupción INT0 (ver tablas 12.1 y 12.3); también, se emplea la función **attachInterrupt(num_int, rutina_interruptcion, CHANGE)** donde se ha seleccionado la constante **CHANGE** para indicar cambios de niveles de voltaje (LOW y HIGH), esta es la forma para detectar o activar la interrupción (ver tabla 12.2). Al pin 2 debe conectarse el pin 13, el cual conmuta esos niveles de volaje.

El procedimiento para ejecutar el sketch **cap12_intA** del cuadro de código Arduino 12.1 se encuentra indicado en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**, página 37. No obstante, para comodidad al lector, dicho procedimiento se describe a continuación. El código fuente del sketch **cap12_intA** se encuentra disponible en el sitio Web de esta obra (Web):

- En el menú **Herramientas** seleccionar el modelo de tarjeta, puerto USB y configurar velocidad de comunicación serial USB en 9600 Baudios.
- Compilar el sketch mediante el icono ✓.
- Descargar el código de máquina del sketch a la tarjeta Arduino usando →.
- Desplegar resultados con el monitor serial (activar con 🔍).
- **Importante**: para exhibir adecuadamente los resultados del sketch, maximice la ventana del monitor serial y active la opción **desplazamiento automático**.

Código Arduino 12.1: sketch cap12_intA

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 12 **Manejo de interrupciones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "**Te acerca al conocimiento**".

Sketch cap12_intA.ino

```
1  float h=0.001;
2  int pin=13;
3  volatile float t=0;
4  volatile int pulso = LOW;
5  int num_int=0; //conectar el pin2 al pin13.
6                 //0 corresponde a la interrupción int0 (pin2)
7  . void setup(){
8      Serial.begin(9600);
9      pinMode(pin, OUTPUT);
10     attachInterrupt(num_int, rutina_interruptcion, CHANGE);
11 }
12 void rutina_interruptcion(){// ISR
13     t=t+h;//genera contador: t_k = t_{k-1} + h.
14     if(t >=10.0)
15         t=0;
16     pulso = !pulso;//conmuta pulso que se envía al pin 13→pin 2.
17 }
18 void loop(){
19     Serial.println(t,4); //envía al monitor la variable tiempo t.
20     delay(100);
21     digitalWrite(pin, pulso);//pin 13→ pin 2, envía pulsos para generar interrupción.
22     if((t>5.0) && (t<5.5))
23         Serial.println("Captura de evento...." );
24     delay(100);
25 }
26 //Para ejecutar este sketch, consulte el procedimiento de la página17.
```

♣ ♣ ♣ Ejemplo 12.2

Graficar en **MATLAB** las siguientes funciones:

$\text{sen}(t)$, $1 - 2.37^{-t}$, $\frac{t}{\sqrt{1+t^2}}$, $\tanh(t)$, $\frac{t}{1+|t|}$ y $\frac{\text{senh}(t)}{1+\cosh(t)}$.

Programar dichas funciones en una subrutina de servicio de interrupción ISR, generar el tiempo discreto como $t_k = t_{k-1} + h$, donde $h = 0.001$ segundos.

Solución

Se utiliza el pin 2 (INT0) para generar la solicitud de interrupción, la forma de activarla es por cambios de voltajes 0 y 5 V para Arduino UNO o 3.3 V para otros modelos (LOW y HIGH). Para eso se emplea un puerto digital programado como salida, por ejemplo, el pin 13, el cual está conectado al pin 2.

El cuadro de código Arduino 12.2 contiene la descripción del sketch **cap12_funciones**. La subrutina de servicio de interrupción ISR se denomina **rutina_interruptcion()**; en esta función se realiza la conmutación de pulsos del pin 13 → pin 2, el incremento del tiempo t_k y la función $y = \text{sen}(t)$ (las demás funciones el usuario las puede programar). Para ejecutar este sketch referirse a la página 17 (omita el paso del monitor serial del ambiente de programación Arduino) y para graficar en **MATLAB** utilice el script **cap12_graficar.m**, cuya documentación se presenta en el código fuente 12.3.

La figura 12.3 muestra la función $\text{sen}(t)$ que se está ejecutando por interrupciones en la tarjeta Arduino y su desplegado en **MATLAB**.

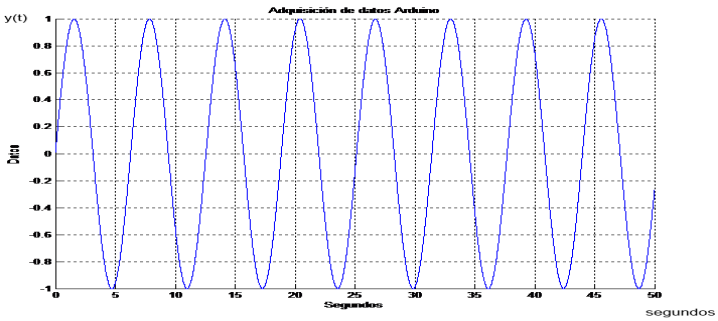


Figura 12.3 Gráfica de la función $\text{sen}(t)$ en **MATLAB**.

Código Arduino 12.2: sketch cap12_funciones

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 12 **Manejo de interrupciones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "**Te acerca al conocimiento**".

Sketch cap12_funciones.ino

```
1  #include <math.h>
2  float h=0.001;
3  int pin=13;
4  volatile float t=0, y;
5  volatile int pulso = LOW;
6  int num_int=0; //Conectar el pin2 al pin13.
7                 //0 corresponde a la interrupción int0.
8  void setup(){
9      Serial.begin(9600);
10     pinMode(pin, OUTPUT);
11     attachInterrupt(num_int, rutina_interruptcion, CHANGE);
12 }
13 void envia_datos(float x, float y){//envía datos t, y(t) a MATLAB.
14     Serial.print(y);
15     Serial.print(" " )
16     Serial.println(x);
17 }
18 void rutina_interruptcion(){//ISR
19     pulso = !pulso; //conmuta el pulso del pin 13 conectado al pin 2.
20     t=t+h; //genera base de tiempo: t_k = t_{k-1} + h.
21     y=sin(t);//funciones a graficar: 1 - 2.37^{-t}, t/sqrt(1+t^2), tanh(t), t/(1+|t|), senh(t)/(1+cosh(t)).
22 }
23 void loop(){
24     digitalWrite(pin, pulso);//envía pulsos pin 13→ pin 2, genera interrupción.
25     envia_datos(t,y);//envía datos de una función f(t) para graficar en MATLAB.
26     delay(10);
27 }//Para ejecutar este sketch, consulte el procedimiento de la página17, además
28 //desde el ambiente de MATLAB ejecute el script cap12_graficar.m.
```

Código MATLAB 12.3 cap12_graficar.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

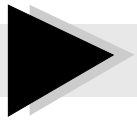
Capítulo 12 **Manejo de interrupciones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "**Te acerca al conocimiento**".

Archivo cap12_graficar.m Versión de MATLAB 2014a

```
clear all
close all
clc
format short
%En este ejemplo se utiliza COM3, esto depende de los recursos y características de su computadora.
canal_serie = serial('COM3','BaudRate',9600,'Terminator','CR/LF');
fopen(canal_serie);%abrir puerto.
xlabel('Segundos');
ylabel('Datos');
title('Adquisición de datos Arduino');
grid on; hold on;
prop = line(nan,nan,'Color','b','LineWidth',1);
datos = fscanf(canal_serie,'%f%f',[2,1]);%leer el puerto serie
clc;
disp('Adquisición de datos de la tarjeta Arduino UNO');
i=1;
while i<5000
    datos = fscanf(canal_serie,'%f%f',[2,1]);%leer el puerto serie.
    y(i)=datos(1,1);
    x(i)=datos(2,1);
    set(prop,'YData',y(1:i),'XData',x(1:i));
    drawnow;
    i=i+1;
end
fclose(canal_serie);%cierra objeto serial.
delete(canal_serie);%libera memoria.
clear canal_serie;%
```



12.4 Aplicaciones en tiempo real

Los sketchs **cap12_intA** y **cap12_funciones** previamente analizados (ver cuadros de código Arduino 12.1 y 12.2), correspondientes a los ejemplos 12.1 y 12.2, respectivamente trabajan en función del tiempo, el cual depende de los ciclos de reloj que el microcontrolador tarde en ejecutar la función **digitalWrite(pin, pulso)** y otras actividades de programación. Sin embargo, dicha interrupción no corresponde a un proceso en tiempo real.

Para trabajar en tiempo real es necesario generar una base de tiempo a través de un timer interno o reloj externo, tal que en forma periódica establezca una señal de interrupción, con la frecuencia de muestreo adecuada de tal forma que el tiempo empleado en realizar todas las operaciones dentro de la IRS sean menor al período de muestreo (la frecuencia de muestreo es el inverso del período de muestreo).

Control de procesos en tiempo real

Control de procesos físicos en tiempo real se refiere al tiempo de máquina que emplea el microcontrolador para realizar todas las operaciones aritméticas, adquisición de datos, procesamiento de la información, cálculo numérico y envío de comandos, de tal forma, que este tiempo de cómputo debe ser menor al período de muestreo, seleccionado previamente por el usuario.

La respuesta de un sistema en tiempo real depende de realizarlo en el menor tiempo posible, en el caso de que existan retardos, éstos deberán ser aceptables en relación al tipo de proceso a controlar y al valor del período de muestreo.

En esta sección se establecen ejemplos en adquisición de datos y control, cuya base de tiempo es de 2 mseg como período de muestreo, el cual se genera a partir del TIMER 1 de la tarjeta Arduino UNO.

♣ ♣ Ejemplo 12.4

Graficar en **MATLAB** la siguiente función:

$$y(t) = \cos(t)\ \text{sen}^2(t)\,[\,1 - e^{-2\ \cos(t)}\,]$$

Programar a la función $y(t)$ en una subrutina de servicio de interrupción ISR; generar con el TIMER 1 el tiempo discreto $t_k = t_{k-1} + h$, donde el período de muestreo h es de 1 mseg.

Solución

El sketch **cap12_funcionIntA** cuya descripción se presenta en el cuadro de código Arduino 12.4, contiene la programación en lenguaje C para ejecutar la función $y(t) = \cos(t)\ \text{sen}^2(t)\left(1 - e^{-2\ \cos(t)}\right)$ en tiempo real, la base de tiempo (1 mseg) que permite disparar a la rutina de servicio de interrupción ISR es por medio del timer 1 de la tarjeta Arduino, se intercepta la interrupción 12 (TIMER1_COMPA_vect) localizada en la dirección de memoria 0x16 (realiza un salto a la dirección de memoria donde está localizada la IRS). Observe que, en el lazo principal de programación **loop()**{...} únicamente se tiene la función **envia_datos(t,y)** para enviar los datos a **MATLAB**.

Para ejecutar el sketch **cap12_funcionIntA** utilice el procedimiento de la página 17 (omita el paso del monitor serial del ambiente de programación Arduino). La figura 12.4 corresponde la función $y(t)$; el proceso de graficado y transmisión de datos en **MATLAB** no es en tiempo real.

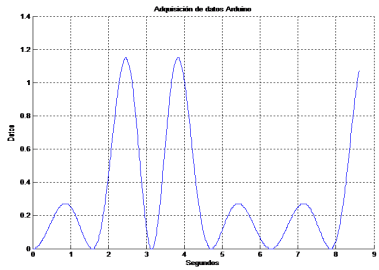


Figura 12.4 Gráfica de la función $y(t) = \cos(t)\ \text{sen}^2(t)\left(1 - e^{-2\ \cos(t)}\right)$.

Código Arduino 12.4: sketch cap12_funcionIntA

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 12 **Manejo de interrupciones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "**Te acerca al conocimiento**".

Sketch cap12_funcionIntA.ino

```
1  #include <math.h>
2  float h=0.001, y, t=0;
3  //Envía datos para graficar en MATLAB.
4  void envia_datos(float x, float y){
5      Serial.print(y);
6      Serial.print(" " );
7      Serial.println(x);
8  }
9  //Subrutina de servicio de interrupción ISR, intercepción de la int 12.
10 ISR(TIMER1_COMPA_vect){//interrupción 12, localidad de memoria 0x016.
11 //Función a ejecutar: y(t_k) = cos(t_k) sen^2(t_k) [1 - e^(-2 cos(t_k))].
12     y=cos(t)*sin(t)*sin(t)*(1-exp(-2*cos(t)));
13     t=t+h;//tiempo discreto t_k = t_(k-1) + h.
14 }
15 void setup() {
16     Serial.begin(9600);
17     // Configuración del timer 1.
18     TCCR1A = 0;
19     TCCR1B = bit(WGM12) |bit(CS10) | bit (CS12);
20     OCR1A = 12;
21     TIMSK1 = bit (OCIE1A);
22 }
23 void loop() {
24     envia_datos(t,y);
25 }//Para ejecutar este sketch, consulte el procedimiento de la página 17, además
26 //desde el ambiente de MATLAB ejecute el script cap12_graficar.m.
```

♣ ♣ Ejemplo 12.4

Adquirir lecturas de datos de un sensor (fotocelda) conectado al canal analógico A0 de la tarjeta Arduino. Implementar un algoritmo de adquisición de datos en tiempo real y graficar la respuesta del sensor en **MATLAB**.

Solución

Una modificación del cuadro de código Arduino 12.4 para realizar adquisición de datos de una fotocelda o cualquier sensor se presenta en el sketch **cap12_AdqInt** (ver cuadro 12.5); se emplea el canal analógico A0. La adquisición de datos se realiza en tiempo real por medio del timer 1, cada milisegundo, se adquieren muestras del sensor y en lazo principal de programación **loop()**{...} se envían datos a **MATLAB** para su representación gráfica. La figura 12.5 contiene las lecturas de voltaje de la fotocelda; el proceso de graficado y transmisión de datos en **MATLAB** no es en tiempo real.

La señal del sensor es procesada por un filtro discreto pasa bajas para atenuar el ruido: $F(t_k) = e^{-\lambda h} F(t_{k-1}) + (1 - e^{-\lambda h})\, y(t_{k-1})$; donde λ es la frecuencia de corte, h es el período de muestreo, $F(t_k)$ es la señal filtrada de la respuesta del sensor $y(t_k)$. Para ejecutar el sketch **cap12_AdqInt** utilice el procedimiento de la página 17 (omita el paso del monitor serial del ambiente de programación Arduino).

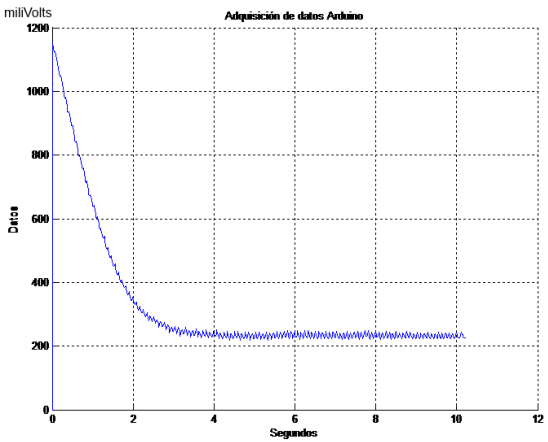


Figura 12.5 Adquisición de datos de una fotocelda..

Código Arduino 12.5: sketch cap12_AdqInt

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 12 **Manejo de interrupciones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "**Te acerca al conocimiento**".

Sketch cap12_AdqInt.ino

```
1  #include <math.h>
2  float h=0.001, y, t=0;
3  //Envía datos para graficar en MATLAB.
4  void envia_datos(float x, float y){
5      Serial.print(y);
6      Serial.print(" " );
7      Serial.println(x);
8  }
9  //Subrutina de servicio de interrupción ISR, intercepción de la int 12.
10 ISR(TIMER1_COMPA_vect){//interrupción 12, localidad de memoria 0x016.
11     static float F_k=0, lambda=25, y_a=0;
12     F_k=exp(-lambda*h)*F_k+(1-exp(-lambda*h))*y_a;
13     y=5000.0*(analogRead(A0))/2048;//lectura de sensores.
14     y_a=y;
15     t=t+h;
16 }
17 void setup() {
18     Serial.begin(9600);
19     // Configuración del timer 1.
20     TCCR1A = 0;
21     TCCR1B = bit(WGM12) | bit(CS10) | bit (CS12);
22     OCR1A = 12;
23     TIMSK1 = bit (OCIE1A);
24 }
25 void loop() {
26     envia_datos(t,y);
27 }//Para ejecutar este sketch, consulte el procedimiento de la página17, además
28 //desde el ambiente de MATLAB ejecute el script cap12_graficar.m.
```

♣ ♣ ♣ Ejemplo 12.5

Considere el siguiente sistema dinámico de primer orden:

$$\begin{aligned} \dot{x}(t) &= -ax(t) + bu(t) \\ u(t) &= k_p \operatorname{senh}(\tilde{x}(t)) \end{aligned}$$

donde $a = 3$, $b = 3$, el error de posición se define como: $\tilde{x}(t) = x_d - x(t)$, siendo la posición deseada $x_d = 1$ y $k_p = 85$ es la ganancia proporcional.

a) Implemente el sistema dinámico en la tarjeta Arduino UNO en tiempo real, con un período de muestreo $h = 0.001$ segundos.

b) Grafique la respuesta del sistema $x(t)$ en **MATLAB**.

Solución

Para implementar el sistema dinámico $\dot{x}(t) = -ax(t) + bu(t)$ en las tarjetas Arduino, se requiere de su versión discreta $x(t_k)$ dada por: $x(t_k) = e^{-ah}x(t_{k-1}) + \frac{b}{a}[\,1 - e^{-ah}\,]\,u(t_{k-1})$, donde h es el período de muestreo y $x(t_k)$ es la solución discreta del sistema $x(t)$.

La rutina de servicio de interrupción ISR contiene la implementación discreta del sistema, así como la ley de control $u(t_k)$ para obtener la convergencia del estado del sistema al estado deseado $x_d = 1$, conforme el tiempo discreto t_k evoluciona (ver sketch **cap12_SisDina**, descrito en el cuadro de código Arduino 12.6).

La ejecución en la tarjeta Arduino UNO con el algoritmo de la solución discreta $x(t_k)$ del sistema dinámico se realiza en tiempo real, dado por la base de tiempo (1 mseg) del timer 1. La gráfica 12.6 muestra la convergencia del estado $x(t_k) \to x_d$; esta gráfica no es en tiempo real, debido a que el envío de datos por comunicación serie establece su propia velocidad de comunicación (9600 Baudios), más el tiempo de graficado en **MATLAB**.

Para ejecutar el sketch **cap12_SisDina** utilice el procedimiento de la página 17 (omita el paso del monitor serial del ambiente de programación Arduino).

Código Arduino 12.6: sketch cap12_SisDina

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 12 **Manejo de interrupciones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "**Te acerca al conocimiento**".

Sketch cap12_SisDina.ino

```
1  #include <math.h>
2  volatile float h=0.001, y, t=0;
3  void envia_datos(float x, float y){//envía datos para graficar en MATLAB.
4      Serial.print(y);
5      Serial.print(" " );
6      Serial.println(x);
7  }
8  ISR(TIMER1_COMPA_vect){//ISR x(t_k) = e^{-ah}x(t_{k-1}) + b/a [1 - e^{-ah}] u(t_{k-1}).
9      static float x_k, a=3, b=3, x_ka=0, u_k, u_ka=0, xd=1, xtilde, kp=85;
10     xtilde=xd-x_k;//error de posición: x̃(t_k) = x_d - x(t_k).
11     u_k=kp*tanh(xtilde);// control: u(t_k) = k_p tanh(x̃(t_k)).
12     x_k=exp(-h*a)*x_ka+(b/a)*(1-exp(-a*h))*u_ka;
13     x_ka=x_k;// x(t_{k-1}) = x(t_k).
14     u_ka=u_k;// u(t_{k-1}) = u(t_k).
15     t=t+h;//t_k = t_{k-1} + h.
16     y=x_k;// y(t_k) = x(t_k).
17 } void setup() {
18     Serial.begin(9600);
19     TCCR1A = 0;
20     TCCR1B = bit(WGM12) | bit(CS10) | bit (CS12);
21     OCR1A = 12;
22     TIMSK1 = bit (OCIE1A);
23 }
24 void loop() {
25     envia_datos(t,y);// envía t_k, y(t_k) a MATLAB.
26 }//Para ejecutar este sketch, consulte el procedimiento de la página17, además
27 //desde el ambiente de MATLAB ejecute el script cap12_graficar.m.
```

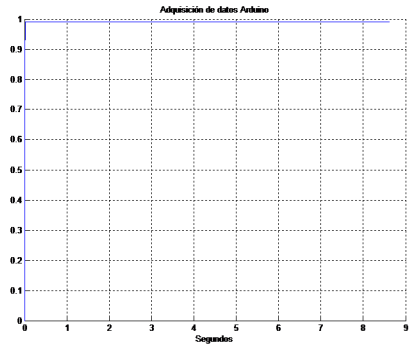


Figura 12.6 Respuesta del sistema $x(t_k)$ con acción de control tanh(...).

12.5 Resumen

INTERRUPCIONES son mecanismos de los microcontroladores que permiten realizar cálculos numéricos, procesamiento de datos, adquisición de información, algoritmos de control, envío de comandos, etc. La arquitectura AVR de los microcontroladores Atmega contiene una unidad de interrupciones que controla la solicitud de periféricos o dispositivos conectados a las tarjetas Arduino, de esta forma el microcontrolador no pierde tiempo en estar sensando o por monitoreo (**polling**) solicitudes de eventos.

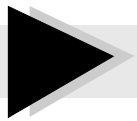
La forma más eficiente para controlar procesos es que la unidad de interrupciones indique al microcontrolador el tipo y jerarquía de solicitudes de interrupciones, esto se realiza mediante la tabla de vectores de interrupción, la cual es una lista indizada en forma prioritaria de todas las interrupciones del microcontrolador. Mayor relevancia se obtiene, cuando el proceso de control no es asíncrono; es mucho mejor establecer una señal de activación estable, que involucre un período de muestreo. Esto se puede obtener usando los timers de las tarjetas Arduino o generando un reloj externo conectado a las interrupciones externas (por ejemplo en el modelo Arduino UNO: INT0 e INT1).

La forma de instalar rutinas de servicio de interrupción ISR es por medio de la función **attachInterrupt(...)**, la cual no requiere librerías adicionales, es una función propia que reconoce el proceso de compilación sin ningún problema. Otra

forma, es interceptando interrupciones, como es el caso de la interrupción número 12 del timer 1 (TIMER1_COMPA_vect), se instala la ISR con el código del usuario y mediante tics de tiempo previamente programados se realiza la ejecución del proceso en tiempo real.

En el sitio Web de las tarjetas Arduino, el usuario puede encontrar información adicional sobre manejo de interrupciones:

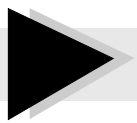
playground.arduino.cc/Main/PinChangeInt



12.6 Referencias selectas

En esta sección se proporciona al usuario enlaces electrónicos y sitios Web con información técnica, del manejo de interrupciones para los microcontroladores Atmega que constituyen la plataforma electrónica de las tarjetas Arduino.

- http://www.Atmega.com/avr
- http://www.Atmega.com/products/microcontrollers/avr/default.aspx
- playground.arduino.cc/Main/PinChangeInt
- http://www.arduino.cc



12.7 Problemas propuestos

Los conceptos sobre el manejo de interrupciones y sus aplicaciones potenciales con los microcontroladores que utilizan los modelos Arduino son tópicos no triviales, por lo tanto, a continuación se propone un conjunto de preguntas con la finalidad de valorar los conocimiento adquiridos por el usuario en este capítulo.

12.7.1 ¿Qué es una interrupción?

12.7.2 Describa la técnica de **polling**.

12.7.3 ¿Por qué se requiere la unidad de control de interrupciones en la arquitectura AVR de los microcontroladores Atmega?

12.7.4 ¿Cuáles son las interrupciones externas del modelo Arduino UNO?

12.7.5 Describa los pins digitales que soportan las interrupciones externas del modelo Arduino UNO.

12.7.6 Describa las interrupciones internas del modelo Arduino UNO.

12.7.7 ¿Qué significa interceptar una interrupción?

12.7.8 ¿Cómo define un proceso en tiempo real?

12.7.9 ¿Qué es control en tiempo real?

12.7.10 Diseñe un circuito externo que genere una señal digital periódica de 2.5 mseg, tal que esa señal se conecte al pin 2, para activar la interrupción INT0 de la tarjeta Arduino UNO. Implemente un rutina de servicio ISR que incremente una variable contador; muestre los resultados en el monitor del ambiente de programación Arduino.

12.7.11 Repita la pregunta inmediata anterior, por medio del reloj externo, diseñar una rutina de servicio de interrupciones ISR tal que calcule en tiempo real las siguientes funciones

a) $\frac{\text{sen}(t)}{1+\cos^2(t)}$.

b) $\frac{t}{1+e^{-2t}}$.

Graficar las funciones en **MATLAB**.

12.7.12 ¿Son en tiempo real las gráficas en **MATLAB** de la pregunta inmediata anterior?

12.7.13 En referencia al ejemplo 12.2 (sketch **cap12_funciones**, ver cuadro de código Arduino 12.2) grafique en **MATLAB** las siguientes funciones: $1-2.37^{-t}$, $\frac{t}{\sqrt{1+t^2}}$, $\tanh(t)$, $\frac{t}{1+|t|}$ y $\frac{\text{senh}(t)}{1+\cosh(t)}$.