URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

GSyC

Sistemas Empotrados y de Tiempo Real

Interrupciones

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

Polling

- El **polling** es una técnica antigua por la cual se sondea un estado de manera continua y constante.
- Técnica ineficiente:
  - El microcontrolador gasta ciclos sondeando, cuando la mayoría de las veces no es necesario ya que no hay datos disponibles.
  - Se pueden perder eventos/lecturas.
  - No existe prioridad.
  - Consume más energía.
  - No escala correctamente con numerosas entradas digitales.

Interrupciones

- Las interrupciones son **mecanismos** del microcontrolador para responder a **eventos**.
- Suspende **temporalmente** el programa principal.
- Ejecuta una subrutina de servicio de interrupción (ISR)
- Una vez terminada dicha subrutina, se **reanuda** la ejecución del programa principal.
- Las **interrupciones** son esenciales en **sistemas empotrados**, ya que le permite atender procesos del mundo exterior sin descuidar la ejecución del programa principal.
- Ejemplos: adquisición de datos, monitoreo de sensores, cálculos numéricos, envío de comandos al robot, etc.

¿Cómo funciona una interrupción?

- Cuando se genera una interrupción:
  - La unidad de interrupciones indica al microcontrolador que un evento **requiere** solicitud de interrupción.
  - El microcontrolador determina qué **tipo** de interrupción se está generando.
  - El microcontrolador salva el **contador de programa** y contexto.
  - Cada tipo de interrupción tiene una **ISR** asignada que se guarda en la tabla de vectores de interrupción.
  - Se ejecuta el código de la ISR.
  - Se restaura el contexto y se devuelve el control al programa que estaba en ejecución.

¿Cómo funciona una interrupción?

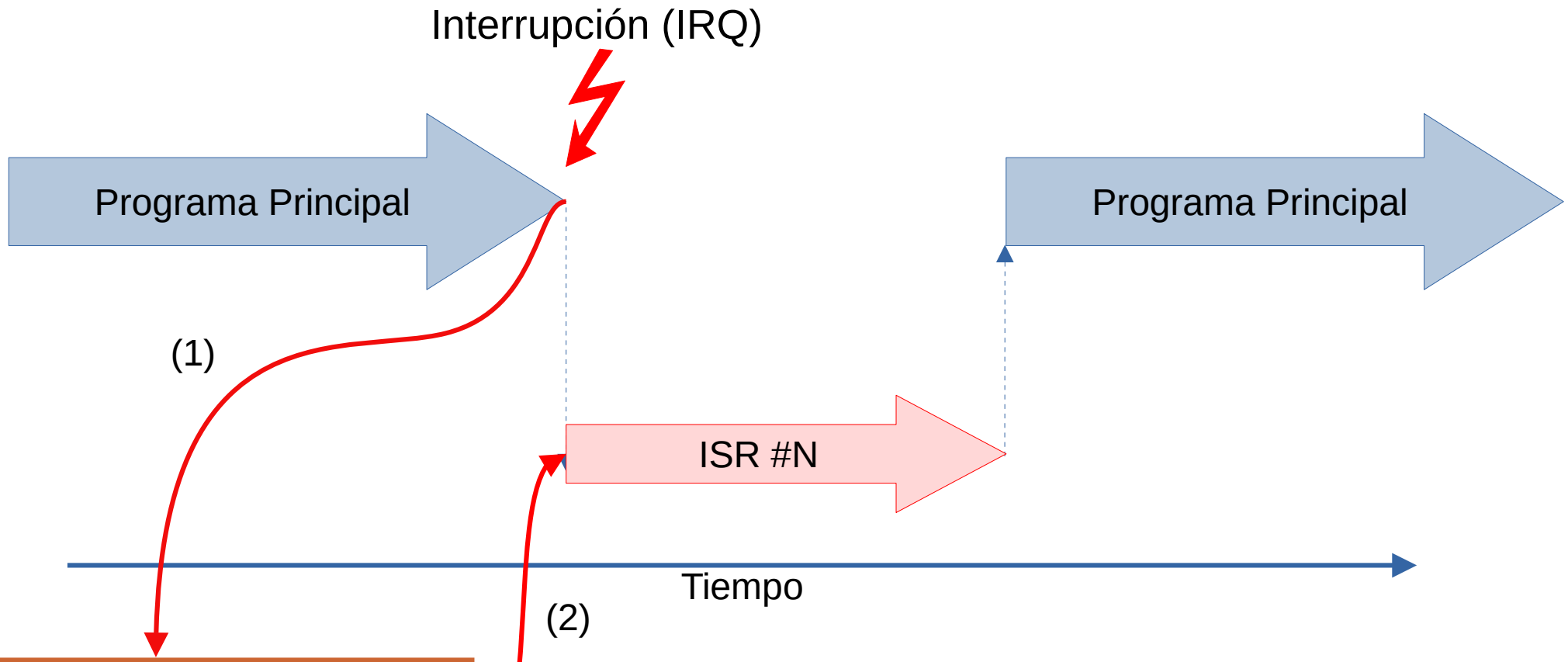


Table de Vectores de Interrupción	
IRQ #1	ISR #1
...	...
IRQ #N	ISR #N

Tipos de Interrupciones

- En los microcontroladores Atmega nos encontrados con interrupciones hardware y software.
- Interrupciones **hardware** nos ayudan a determinar cambios en los pines físicos de entrada.
  - *Arduino UNO* dispone de 2 pines para interrupciones por hardware:
    - Int 0: Pin 2
    - Int 1: Pin 3
  - Arduino MEGA dispone de 6 pines.
- Utilizados para sensores y lecturas del entorno exterior.

Tipos de Interrupciones

- Interrupciones **software** se generan a través de timers o contadores. Son generadas cuando pasa un tiempo determinado.
- Arduino UNO dispone de:
  - 2 Timers de 8bit: Timer0, Timer2
  - 1 Timer de 16bit: Timer1

- Timer0 es usado para *millis()*, por lo que el uso de este timer hará que la función millis() no funcione correctamente.

Interrupciones ATmega328P

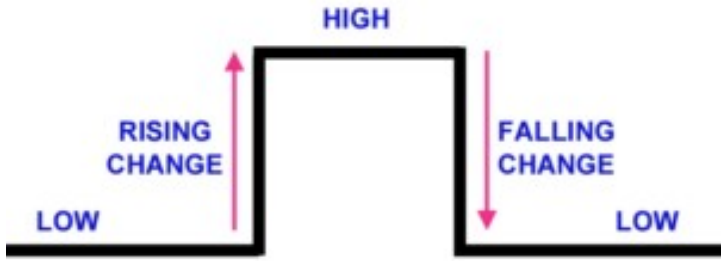
Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown- system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B

Interrupciones ATmega328P

14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Interrupciones Hardware

- Las interrupciones hardware puede ser configuradas para detectar estados lógicos (voltaje) o transición de pulsos.
  - *LOW*: La interrupción salta cuando el pin está en estado LOW
  - *HIGH*: La interrupción salta cuando el pin está en estado HIGH (sólo disponible en Arduino DUE).
  - *CHANGE*: La interrupción salta cuando el pin cambia de estado, ya sea LOW a HIGH, o viceversa.
  - *RISING*: La interrupción salta cuando el pin pasa de LOW a HIGH.
  - *FALLING*: Esta es la inversa de la de arriba, y por lo tanto salta cuando el pin pasa de HIGH a LOW



ISR

- Interrupt Service Routine.
- Deben ejecutar el mínimo tiempo posible.
- Preferiblemente código sencillo y simple.
- Solo puede ejecutar una ISR a la vez, en caso de varias, se ejecutan secuencialmente.
- Normalmente la función de la ISR se limita a activar un flag, incrementar un contador, o modificar una variable.
- No añadir en una ISR:
  - Cálculos complejos
  - Comunicación (serial, I2C y SPI)
- Siempre que sea posible, no realizar cambios lógicos en entradas salidas/digitales.

Efecto de la interrupción

- Arduino utiliza interrupciones para realizar el conteo del tiempo mediante *millis()* y *micros()*
- Efectos dentro de la ISR:
  - Durante la ejecución de la ISR, arduino no actualiza los valores de las funciones *millis()*, y *micros()* funcionaría erróneamente después de 1-2 ms.
  - No se contabiliza el tiempo de ejecución de la ISR.
- Efectos fuera de la ISR:
  - *millis()* no actualiza su valor, y *delay()* no podría funcionar correctamente.
- El uso de interrupciones provee de funcionalidad muy potente pero hay que ser cuidadosos con su uso.

Interrupciones Hardware

- Las interrupciones hardware las activaremos usando la siguiente llamada:

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

  - Pin: Es el pin físico asociado a la interrupción
  - ISR: callback a la función de tratamiento de interrupción
  - Mode: Estado lógico para activar la interrupcción
    - HIGH, LOW
    - CHANGE
    - RISING, FALLING

Interrupciones Hardware

- Otras funciones interesantes para la gestión de interrupciones son:
  - Anular interrupción
    - `detachInterrupt(interrupt)`
  - Desactivar ejecución de interrupciones
    - `noInterrupts()`
  - Reactivar Interrupciones
    - `interrupts()`

Volatile

- Directiva del compilador asociada a definición de **variables**.
- Fuerza al compilador a cargar esa variable siempre desde memoria **RAM** y en una operación atómica (y no desde uno de los registros internos).
- La variable tiene que ser **cargada siempre** antes de su lectura, ya que ha podido ser modificada de forma ajena al flujo principal del programa.
- En arduino, este caso solo se da en **interrupciones**.
  - Una interrupción puede cambiar el estado de una variable que luego se usará en el loop() principal.
- El uso de volatile **ralentiza** la ejecución. Solo usar en situaciones donde sea realmente necesario.

Interrupciones Hardware

- Enceder/Apagar un LED a través de interrupciones (botón)

```
const int INT_PIN = 2;

volatile byte state = LOW;


void blink() {

  state = !state;
  digitalWrite(LED_BUILTIN, state);

}

void setup() {

  Serial.begin(9600);

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(INT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(INT_PIN), blink, CHANGE);

}

void loop() {
  delay(10);
  Serial.println(state);
}
```

Una variable debe ser definida como 'volatile' si es modificada en una ISR y queremos que se actualice el valor de esa variable en otras funciones.

Cambiamos el estado de la salida digital en la ISR

Asociamos la interrupción del pin #2 a la ISR definida por blink(), para que se ejecute cuando cambie el estado de la señal lógica (CHANGE)

Interrupciones por Hardware

```
const int INT_PIN = 2;

volatile byte state = LOW;
long startTime = 0;
const int timeThreshold = 200;

void blink() {
  if (millis() - startTime > timeThreshold) {
    startTime = millis();
    Serial.println("Interruption: " + String(millis()));
    state = !state;
    digitalWrite(LED_BUILTIN, state);
  }
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(INT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(INT_PIN), blink, CHANGE);

}

void loop() {
  // put your main code here, to run repeatedly:
  delay(100);
  Serial.println(state);
```

Interrupciones por Software

- Las interrupciones por **software** son sensibles a la precisión y exactitud del reloj interno del sistema.
- Las interrupciones software basadas en timers utilizan la misma idea de una **alarma** en un despertador.
- El contador del timer se va incrementando hasta que finaliza el conteo y salta la interrupción.
- Son muy utilizadas para implementar sistemas **multitasking** en entornos mono-core.

Interrupciones por Software

- Las Timers en Arduino permiten una configuración a bajo nivel escribiendo en los registros apropiados.
- Utilizaremos la librería *TimerOne* para interactuar con el timer1 de Arduino
  - https://www.arduino.cc/reference/en/libraries/timerone/
- Compatible con la arquitectura AVR, por lo que soporta:
  - Arduino Micro
  - Arduino Leonardo
  - Arduino Mega
  - Arduino Nano
  - Arduino Uno
  - Arduino Yún

Interrupciones por Software

- Encender/Apagar un LED

```
#include <TimerOne.h>

int ledstate = LOW;

void blinkLED() {

  digitalWrite(LED_BUILTIN, ledstate);
  ledstate = !ledstate;

}

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  Timer1.initialize(500000);
  Timer1.attachInterrupt(blinkLED);

  Serial.begin(9600);
}

void loop() {

  delay(100);
}
```

Incluimos la librería TimerOne

Definimos la ISR que tratará la interrupción por software

Inicializamos el timer con el tiempo periódico en el cual se lanzará una interrupción software.

Asignamos la ISR al Timer1

Evitar interrupciones anidadas

- Es posible que recibamos más interrupciones de las que podemos/queremos tratar.
- Eso puede llevar a un mal funcionamiento del sistema.
- Es posible inhabilitar las interrupciones mientras estamos ejecutando la ISR.

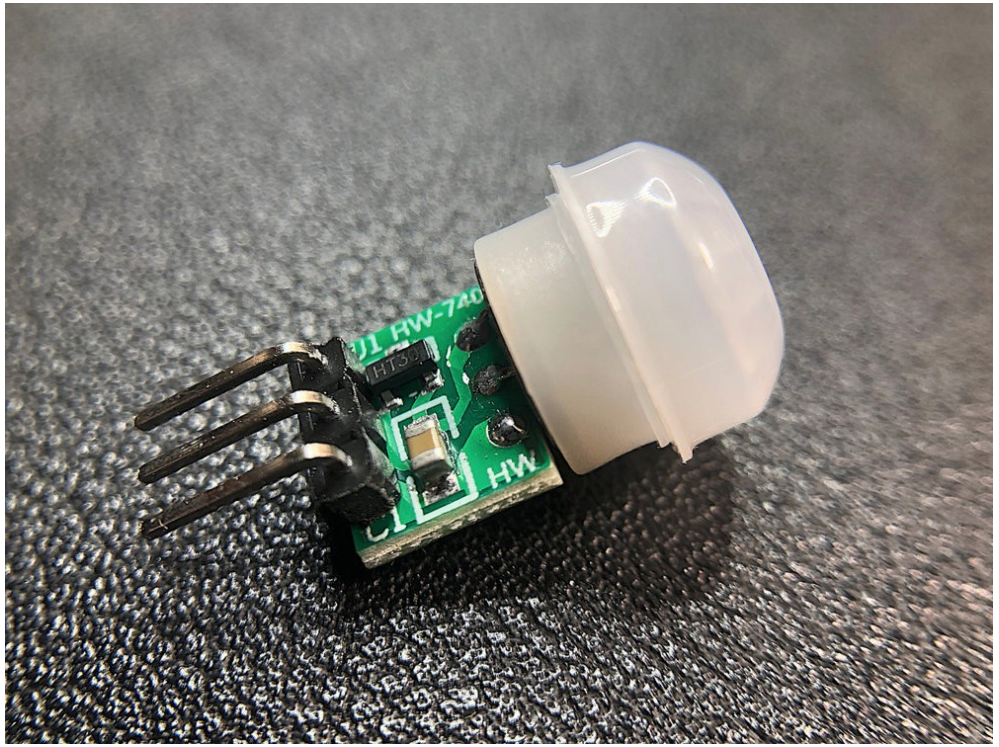
```
void ISR() {
  noInterrupts()
  ... //código que atiende la interrupción
  interrupts()
}
```

Más sobre interrupciones

- La interrupción para *timer0* es usado para *millis()*
- Muchas librerías externas usan directamente *millis()* para generar ejecuciones periódicas, por lo que usan internamente interrupciones.
- No uses '*volatile*' si no es necesario, ralentiza la ejecución ya que el microcontrolador tiene que copiar valores de variables constantemente
- Es posible configurar la frecuencia del timer, eso es, la rapidez con la que cuenta. A una *f=100Hz*, tenemos un T=*1/100*, que son 10 ms.
  - Un contador de 8bits [0-255], generaría una interrupción cada 255*10 ms = 2.55 segundos.

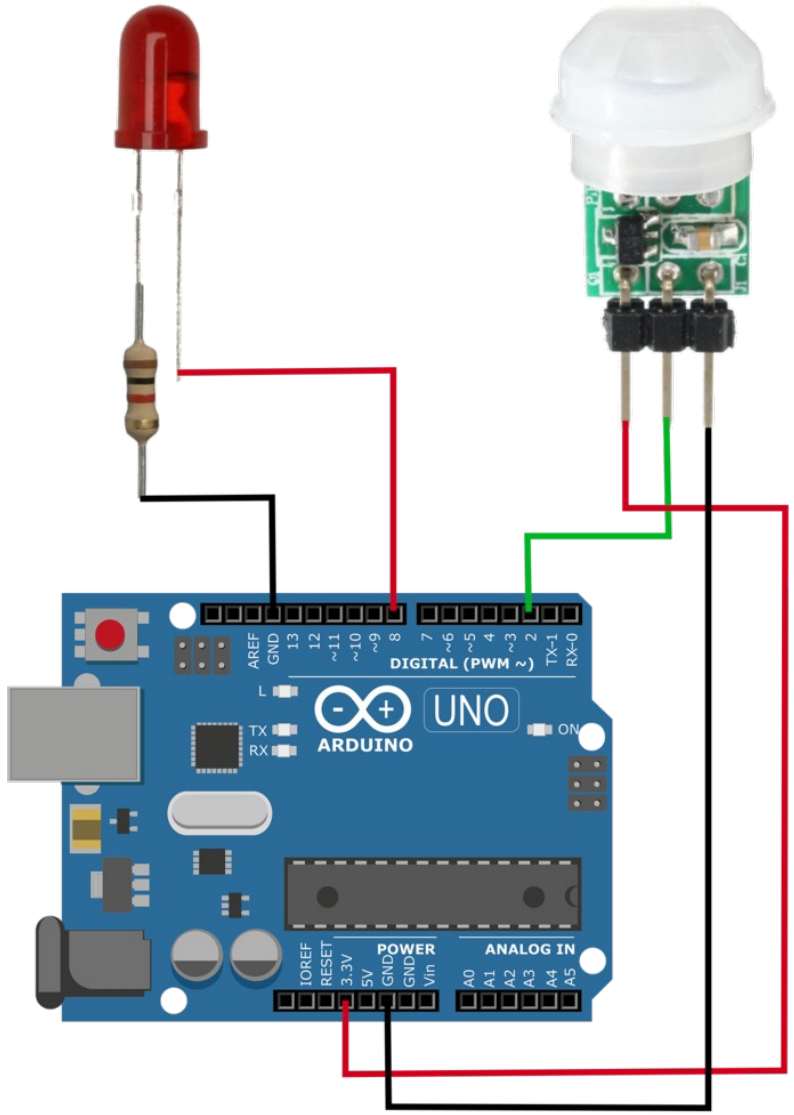
Ejemplo

- Sensor de presencia mediante interrupciones



1. VCC
2. OUT
3. GND

Ejemplo


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM ~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

Bibliografia

- ATmega328P (datasheet)
  - https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
- Arduino Software Internals: A Complete Guide to How Your Arduino Language and Hardware Work Together

URJC Escuela Técnica Superior de Ingeniería de Telecomunicación