Interrupt using SPI, I2C and UART

e-Yantra Team

July 8, 2016

Contents

1 Objective

In this tutorial we will learn interrupt using SPI, I2C and UART protocol.

2 Prerequisites

- Python programming skills(Concept of threading)
- Knowledge about Arduino

3 Hardware Requirement

1. Raspberry Pi (I will be using Version 2 Model B+)
2. Power adapter
3. Connecting wires
4. Arduino
5. Resistor (3.3K)
6. Push Button

4 Software Requirement

1. PyScripter (version 2.7 or above)
2. Mobaxterm (for windows users)

5 Theory and Description

Interrupt
An Interrupt is a signal generated by some event external to CPU, which causes the CPU to stop what it is doing and go to separate piece of code known as ISR for execution. When the execution of ISR completes it starts main program from where it left.

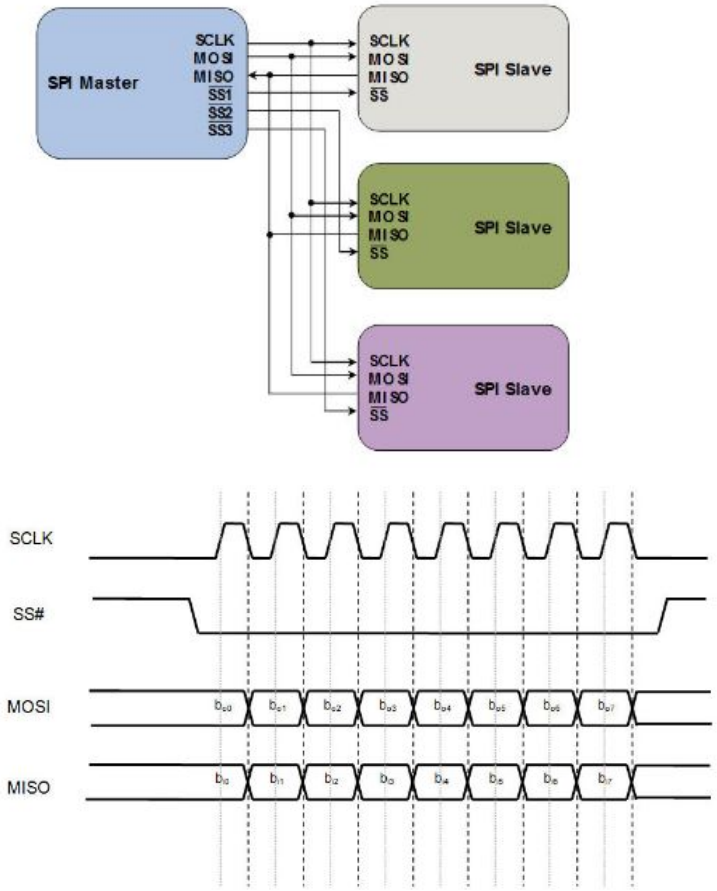
Types of Interrupt

- Hardware Interrupt
  Hardware interrupts are used by devices to communicate that they require attention from the operating system.These interrputs are requested by external peripherals such as keyboard, moving the moving trigger.

- Software Interrupt
  A software interrupt is a type of interrupt that is caused either by a special instruction in the instruction set or by an exceptional condition in the processor itself. A software interrupt is invoked by software, unlike a hardware interrupt, and is considered one of the ways to communicate with the kernel or to invoke system calls, especially during error or exception handling.

Interrupt through UART
Whenever CPU receives interrupt request from external device using UART. Control goes to receives data from devices. After that control returns to its main program execution.
Interrupt through I2C protocol
The Inter-integrated Circuit (I2C) Protocol is a protocol intended to allow multiple slave digital integrated circuits (chips) to communicate with one or more master chips. I2C requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 slave devices. I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices cant talk to each other over the bus and must take turns using the bus lines). Whenever CPU receives interrupt request from external device using I2C protocol.Control goes to receives data from devices. After that control returns to its main program execution.
Interrupt through SPI Protocol
SPI is a single-master communication protocol. This means that one central device initiates all the communications with the slaves. When the SPI master wishes to send data to a slave and/or request information from it, it selects slave by pulling the corresponding SS line low and it activates

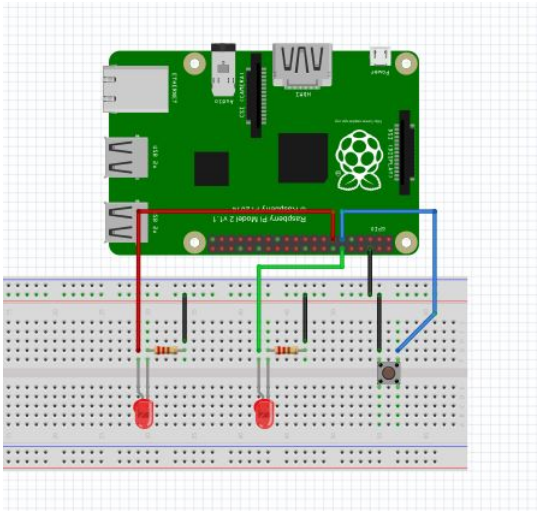
the clock signal at a clock frequency usable by the master and the slave. The master generates information onto MOSI line while it samples the MISO line.



6 Experiment

6.1 External Interrupt

In this experiment we will detect falling edge of GPIO pins. Gpio pin is connected to the switch. So, when switch is pressed, it is detected and ISR is called.



Code

```
# Switch - pin 11 and GND
# LED1 - pin 12
# LED2 - pin13
import time
import thread  # to create thread
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)  # set Rpi in board mode
GPIO.setup(11,GPIO.IN,GPIO.PUD_UP)  # set pin 11 as input and internal pu
                                    # is set high
GPIO.setup(12,GPIO.OUT) # set pin 12 as output
GPIO.setup(13,GPIO.OUT) # set pin 13 as output

# Function name :switch_pressed
# Input : None
# Output : Turn on led for 5 second and then turn it off.
# Example call: switch_pressed()
def switch_pressed():
        print 'Switch pressed'
        GPIO.output(12,True)
        time.sleep(5)
        GPIO.output(12,False)
        return

GPIO.add_event_detect(11,GPIO.FALLING, callback=switch_pressed,bouncetime=
                                        # Continuously check status of pi
try:
    while 1:
```

```
            GPIO.output(13,True) # turn on led connected to pin 13
            time.sleep(0.5)      #0.5 second delay
            GPIO.output(13,False) # turn off led
            time.sleep(0.5)

except KeyboardInterrupt:
    pass
GPIO.cleanup()              # cleanup GPIO values on GPIO pins
```

Same experiment is done by threading. Code for it is given below.

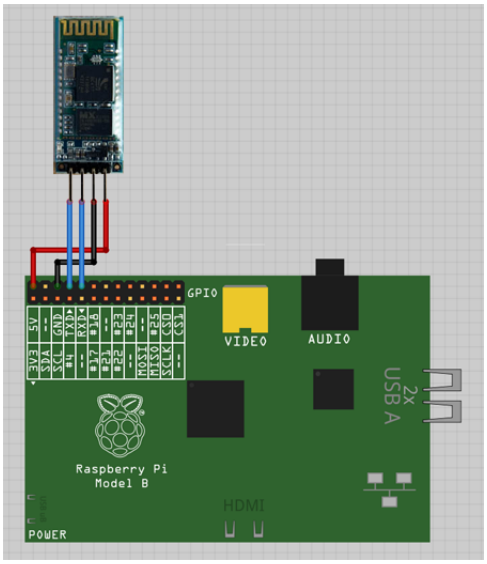
Code

```
import threading
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)                #Set RPi in Board Mode
GPIO.setup(11,GPIO.IN,GPIO.PUD_UP)      #Set a Pull up resistor for pin 11
GPIO.setup(12,GPIO.OUT)                 #Set pin 12 as output
GPIO.setup(13,GPIO.OUT)                 #Set pin 13 as output

# Function to
# Channel must be an integer 0-7
# Function name :ReadChannel
# Input : channel
# Output : data
# Example call: ReadChannel(channel)
def main_prog(name, delay, run_event):
    while run_event.is_set():
        led2_on()
        time.sleep(delay)
        led2_off()
        time.sleep(delay)

def switch_interrupt(name, delay, run_event):
    while run_event.is_set():
        input_state = GPIO.input(11)
        if input_state == False:
            print(name)
            led1_on()
            time.sleep(delay)
            led1_off()
        else:
            led1_off()

def led1_on():
    GPIO.output(12,True)
    return
def led1_off():
    GPIO.output(12,False)
    return
```

```
def led2_on():
    GPIO.output(13,True)
    return
def led2_off():
    GPIO.output(13,False)
    return


if __name__ == "__main__":
    run_event = threading.Event()
    run_event.set()
    d1 = 1
    t1 = threading.Thread(target = switch_interrupt, args = ("Switch Pres

    d2 = 5
    t2 = threading.Thread(target = main_prog, args = ("Main Program Execu

    t1.start()
    time.sleep(.5)
    t2.start()

    try:
        while 1:
            time.sleep(.1)
    except KeyboardInterrupt:
        print "attempting to close threads. Max wait =",max(d1,d2)
        run_event.clear()
        t1.join()
        t2.join()
        print "threads successfully closed"
        GPIO.cleanup()
```

6.2 Communication between RPi and Android device using UART protocol

In this experiment, main program in python remain in execution when we send data from android device. It does does wait for data. When data receive it prints it on the screen.



Connections of Bluetooth modules is as follows:

Code

```
import threading
import time
import serial
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setup(13,GPIO.OUT)

ser=serial.Serial('/dev/ttyAMA0',baudrate=9600,parity=serial.PARITY_NONE,

# Function to read Serial data
# Function name :read
# Input : ser
# Output : read_value
# Example call: read(ser)
def read(ser):
    read_value=""
    while True:
        ch=ser.read()          # reads the data
        read_value+=ch         # appends the string in read_value
        if ch=='\r' or ch=='':
            return read_value
```

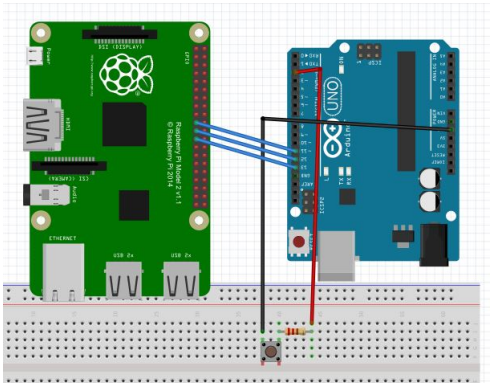
```
def Serial_Data(name,run_event):
    while run_event.is_set():
        re_va=read(ser)
        ser.write("\r"+repr(re_va)) # writes the data received
        print (re_va)
    ser.write("\r\n Connection terminated:") # writes the data to serial
    return

def main_program(name,delay,run_event):
    while run_event.is_set():
        led_on()
        time.sleep(delay)
        led_off()
        time.sleep(delay)
    return

def led_on():
    GPIO.output(13,True)
    return

def led_off():
    GPIO.output(13,False)
    return


if __name__ == "__main__":
    run_event = threading.Event()
    run_event.set()
    d1=1
    t1 = threading.Thread(target = Serial_Data, args = ("Serial_data",run

    d2 = 0.5
    t2 = threading.Thread(target = main_program, args = ("Main",d2,run_ev

    t1.start()
    time.sleep(.5)
    t2.start()

    try:
        while 1:
            time.sleep(.1)
    except KeyboardInterrupt:
        print "attempting to close threads. Max wait =",max(d1,d2)
        run_event.clear()
```

```
t1.join()
t2.join()
print "threads successfully closed"
GPIO.cleanup()
```

6.3 Interrupt using SPI Protocol

In this experiment, Arduino interrupts RPi using SPI protocol. On Arduino board we have intefaced a switch and interrupt occurs when switch is pressed. In Python we make a thread polling the status of switch continuousally and in another thread our main program continues.

Connections:



Code

```
import threading
import time
import spidev


value = [0x08]
spi = spidev.SpiDev()
spi.open(0,0)
i = 0

# Function to read Serial data
# Function name :read
# Input : ser
# Output : read_value
# Example call: read(ser)
```

```
def Receive_data():
    while run_event.is_set():
        number = spi.readbytes(1)
        if (number !=):
            print (number)
            print "thread1"
        time.sleep(1)
    return

def Send_data():
    i=0;
    while run_event.is_set():
        #resp=spi.xfer2(value)
        #print resp
        print "thread2"
        time.sleep(1)
    return




if __name__ == "__main__":
    run_event = threading.Event()
    run_event.set()
    d1=1
    t1 = threading.Thread(target = Receive_data, args = ())

    d2 = 0.5
    t2 = threading.Thread(target = Send_data, args = ())

    t1.start()
    time.sleep(.5)
    t2.start()

    try:
        while 1:
            time.sleep(.1)
    except KeyboardInterrupt:
        print "attempting to close threads. Max wait =",max(d1,d2)
        run_event.clear()
        t1.join()
        t2.join()
        print "threads successfully closed"
```

Arduino UNO Code:

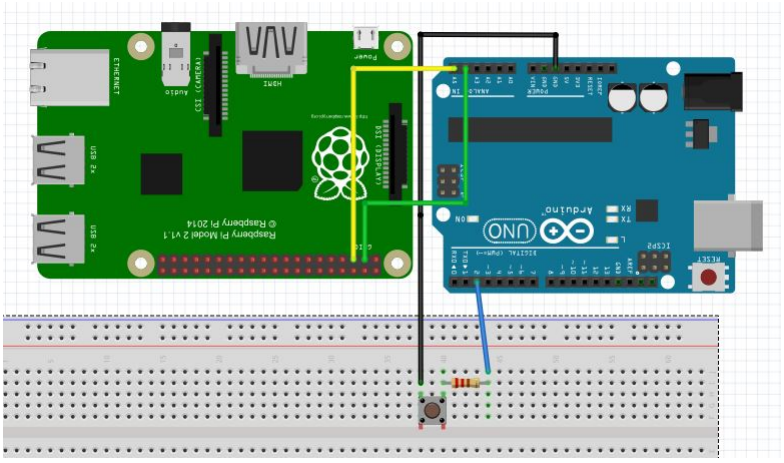
```
#include <SPI.h>

char buf [100];
volatile byte pos;
volatile boolean process_it;

void setup (void)
{
  Serial.begin (9600);   // debugging

  // have to send on master in, *slave out*
  pinMode(MISO, OUTPUT);

  // turn on SPI in slave mode
  SPCR |= _BV(SPE);

  // get ready for an interrupt
  pos = 0;   // buffer empty
  process_it = false;

  // now turn on interrupts
  SPI.attachInterrupt();

}  // end of setup


// SPI interrupt routine
ISR (SPI_STC_vect)
{
byte c = SPDR;  // grab byte from SPI Data Register

  // add to buffer if room
  if (pos < sizeof buf)
    {
    buf [pos++] = c;

    // example: newline means time to process buffer
    if (c == '\n')
      process_it = true;
```

```
    }  // end of room available
}  // end of interrupt routine SPI_STC_vect

// main loop - wait for flag set in interrupt routine
void loop (void)
{
  if (process_it)
    {
    buf [pos] = 0;
    Serial.println (buf);
    pos = 0;
    process_it = false;
    }  // end of flag set

}  // end of loop
```

6.4 Interrupt using I2C Protocol

In this experiment, Arduino interrupts RPi using I2C protocol. On arduino board we have intefaced a switch and interrupt occurs when switch is pressed. In Python we make a thread polling the status of switch continuousally and in another thread our main program continues..

Connections are as shown below:



Code

```
import threading
import smbus
```

```
import time

bus = smbus.SMBus(1)

address = 0x04

# Function to read Serial data
# Function name :read
# Input : ser
# Output : read_value
# Example call: read(ser)

def Serial_Data(name,run_event):
    while run_event.is_set():
        number = bus.read_byte(address)
        if (number != 0):
            print (number)
        time.sleep(2)
    bus.write_byte(address,0)
    return

def main_program(name,run_event):
    i=0;
    while run_event.is_set():

        bus.write_byte(address,i);
        print ("RPI: Hi Arduino, I sent you", i)
        i = i + 1
        time.sleep(1)
    return




if __name__ == "__main__":
    run_event = threading.Event()
    run_event.set()
    d1=1
    t1 = threading.Thread(target = Serial_Data, args = ("Serial_data,run_

    d2 = 0.5
    t2 = threading.Thread(target = main_program, args = ("Main",run_event

    t1.start()
```

```
    time.sleep(.5)
    t2.start()

    try:
        while 1:
            time.sleep(.1)
    except KeyboardInterrupt:
        print "attempting to close threads. Max wait =",max(d1,d2)
        run_event.clear()
        t1.join()
        t2.join()
        print "threads successfully closed"
```

Arduino UNO Code:

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x04
int button_state = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(2,INPUT);
  Serial.begin(9600); // start serial for output
  // initialize i2c as slave
  Wire.begin(SLAVE_ADDRESS);

  // define callbacks for i2c communication
  //Wire.onReceive(receiveData);
  Serial.println("Ready!");
}

void loop() {
  // put your main code here, to run repeatedly:
button_state = digitalRead(2);
if (button_state == HIGH){
  Wire.write("1");
}
else{
  Wire.write("1");
}
}
```

7 Exercise

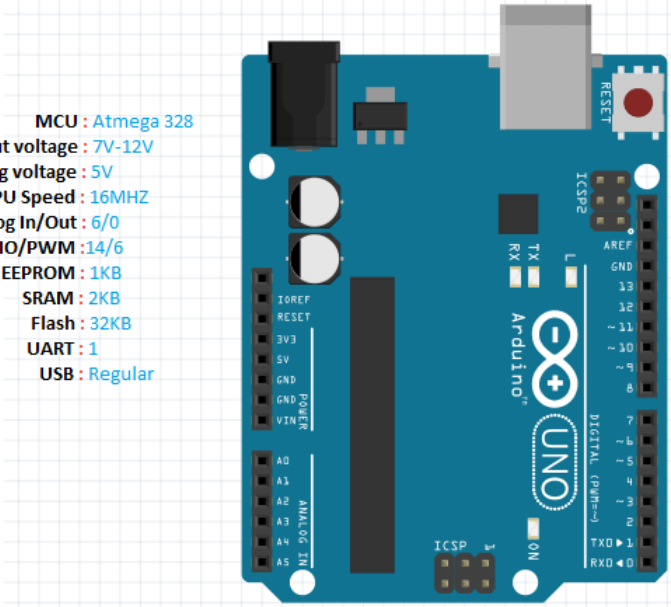
1. Interface a dc motor through L293D motor driver IC using hardware PWM pins(GPIO 18 or IC pin 12) on RPi.

8 Appendix

8.1 PIN Diagram

Arduino Board Pins

MCU : Atmega 328
Input voltage : 7V-12V
Operating voltage : 5V
CPU Speed : 16MHZ
Analog In/Out : 6/0
Digital IO/PWM :14/6
EEPROM : 1KB
SRAM : 2KB
Flash : 32KB
UART : 1
USB : Regular



ARDUINO PIN		MICROCONTROLLER PIN
0	-	PD0(RXD)
1	-	PD1(TXD)
2	-	PD2(INT0)
3	-	PD3(INT1)
4	-	PD4
5	-	PD5
6	-	PD6
7	-	PD7
8	-	PB0
9	-	PB1
10	-	PB2(SS')
11	-	PB3(MOSI)
12	-	PB4(MISO)
13	-	PB5(SCK)
A0	-	PC0
A1	-	PC1
A2	-	PC2
A3	-	PC3
A4	-	PC4(SDA)
A5	-	PC5(SCL)

8.2 Datasheets

ATMEGA328

`www.mouser.com/pdfdocs/Gravitech_ATMEGA328_datasheet.pdf`

9 References

1. https://www.techopedia.com/definition/22195/software-interrupt
2. http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/