The Raspberry Pi

In the 1970's and early 1980's computer hobbyists thrived on Apple II's, Z-80 processor computers, and many others. They could modify the behavior of the computer's hardware and learn computer science at the most fundamental level. My Z-80 computer used a (initially) free UCSD Pascal Operating system. By the late 1980's and 1990's this type of learning opportunity was greatly inhibited as computer hardware became a commodity and documentation of the hardware being hidden from the owner.

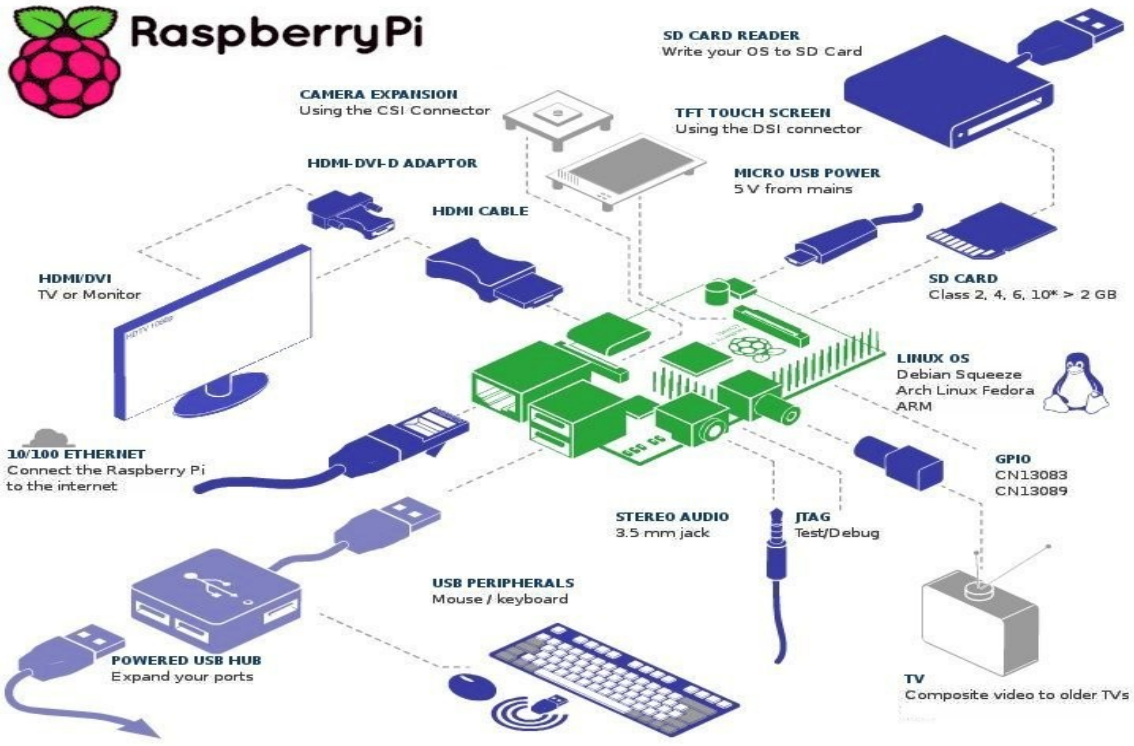
Graphical user interfaces (GUI's) also made programming much more complicated, and basic tools for creating GUI programs were expensive and tended to lock the programmer into a particular supplier and graphic system.

Furthermore, as computers got faster and used more delicate circuits, hacking the hardware risked making an expensive computer into a useless "brick." As a result, by the early 2000's students skilled in interfacing computers to hardware were becoming increasingly rare.

The Raspberry Pi computer was created in the late 2000's to bring back the possibility of hardware hacking and the resultant hardware-software interfacing skills. Its hardware specifications, operating system software, and application software were open to examination and modification with the sole exception of some of the core firmware of the central RISC processor. The price of the Raspberry Pi Model B is $35, so killing it with faulty voltage connections was not as painful as with more expensive computers. In addition, the electronics hobby community is providing a wealth of information on the Internet to help novices. Virtually all common Linux software has been ported to the Raspberry Pi including scientific computing software. A free version of Mathematica has even been made available.

As it turns out, the low cost of these computers has led to their use in scientific and industrial applications throughout the world - from the bottom of the sea to the stratosphere. They are also finding increasingly-wide use by students in developing countries and in schools at all levels. Over 2.5 million were sold in the first two years of their release.

Useful Links: http://en.wikipedia.org/wiki/Raspberry_Pi
http://en.wikipedia.org/wiki/Raspberry_Pi_Foundation
http://www.raspberrypi.org/



Overview of Hardware Features, Accessories, and Add-on Boards

The Raspberry Pi has a 700 MHz processor with 512 MB RAM and a wide variety of input/output ports. Basically, it has the capabilities of a desktop computer sold in year 2000. The Pi alone costs $35 **without** keyboard/mouse, speakers, monitor, camera or power supply. The Raspberry Pi has the following:

2 USB 2.0 Ports (An externally-powered hub is necessary for power-hungry USB peripherals.)
1 100-Mb/s Ethernet Port
512 MB RAM
SD/SDHC card slot for storage (I use a 32 GB card)
HDMI video port (with hardware decompression and compression of 1920x1080 video)
CSI Camera interface for an HD Camera
Stereo analog audio port
Standard analog video port
Broadcom BCM2835 System-on-chip with 700 MHz ARM1176JZF-S RISC processor
18 GPIO pins that are shared with USART, SPI, TWI, and PWM interface protocols
5 VDC power required, but internal system voltage is 3.3 V

Required accessories (These assume you connect to the Raspberry Pi from another computer by an Ethernet ssh connection.):

5VDC, 2A power supply ($13) via USB-micro connector (Typically only 3 W are used, but the extra power is for accessories.)
SD/SDHC card with operating system (e.g. 8 GB with NOOBS system configuration software, $10). The SD/SDHC card is used for "disk" storage. I use 32 GB class 10 cards ($22).

Additional accessories for stand-alone use:

HD monitor with HDMI interface or old TV with RCA connectors for video input. DMI interface monitors can be used with an $8 adapter cable. VGA monitors can be used with a $25 HDMI to VGA adapter.
Stereo speakers if not included with the monitor.
Keyboard/mouse with USB or USB-wireless interface.
If wired Ethernet is not available, WiFi via USB adapter (e.g. Edimax EW-7811Un, $10)
Powered USB hub if power-hungry USB accessories are desired such as a USB hard disk.

Add-on boards - Three are described below, but an **extensive listing of add-on boards is at *http://elinux.org/RPi_Expansion_Boards***

Camera - HD capable, two types available, normal or near infra-red sensitive ($25 for either one)

The infra-red camera is actually the same as the normal camera except that a filter that blocks IR is not installed. It is sensitive to near IR (≈900 μm), but not far IR so an IR lighting source is required to take pictures in the dark.

Other specifications are:

Sensor type: OmniVision OV5647 Color CMOS QSXGA (5-megapixel)	
Sensor size: 3.67 x 2.74 mm	Pixel Count: 2592 x 1944
Pixel Size: 1.4 x 1.4 um	Lens: f=3.6 mm, f/2.9
Angle of View: 54 x 41 degrees	Field of View: 2.0 x 1.33 m at 2 m
Full-frame SLR lens equivalent: 35 mm	Fixed Focus: 1 m to infinity
Video: 1080p at 30 fps with codec H.264 (AVC)	Up to 90 fps Video at VGA
Board size: 25 x 24 mm (not including flex cable)	

Gertboard ($50) - a general-purpose expansion board for the GPIO pins

3 quad-buffers (74HC244N) to protect Raspberry Pi GPIO pins
3 push-buttons 12 LEDs
6 open-collector "switches" (ULN2803a) for controlling up to 50V at 0.5 A each
1 motor controller (ROHM BD6222HFP) to provide variable speed and reversibility for up to 18 V and 2 A.
10-bit A/D converter (MCP3002) - for 1024 levels of voltages
8-bit D/A converter (MCP4802) - for outputting 256 levels of voltage
ATmega 328P RISC Microprocessor with the following capabilities and example programs:

Logic voltage: 3.3 V	12 MHz clock
32 kB of self-programmable Flash memory with optional boot code section	
1 kB permanent EEPROM memory for "fuses"	2 kB Static RAM memory for data
23 programmable I/O pins	2 8-bit timers with prescalers and compare features
1 16-bit timer with prescaler, compare, & capture features	6 pulse-wave modulation (PWM) channels using the timers
10-bit A/D multiplexed to 6 channels	USART serial interface
SPI serial interface	TWI serial interface
Watchdog timer with separate clock	25 interrupts supporting the above features

Bitscope micro ($145) - A USB probe with software optimized for the Raspberry Pi.

It performs the functions of an digital storage oscilloscope, logic analyzer, spectrum analyzer, waveform generator and data recorder.

Details at *http://www.raspberrypi.org/blog/#bitscope-micro* and *http://bitscope.com/product/BS05/*

Overview of Software

A wide range of powerful software is available at no charge for the Raspberry Pi since it runs Linux. With very few exceptions (like Mathematica), the software is "open source" meaning that the curious person can download the source code, study it, and modify the software. The only restriction is that when improvements are made and used for commercial purposes, the improvements must be openly shared. That allows the entire open source community to benefit from the improvements.

This software is being continually improved, and the improvements are made conveniently available from free, but secure, on-line sources.

It is especially important important that the Linux operating system is open source so that the curious student can examine its source code and see exactly how it is designed and optimized. For example, the file system details, algorithms for multi-tasking and memory management, and how the separation of user and superuser capabilities are accomplished can be completely understood by studying their source code.

Programming is supported by assemblers, compilers and utilities for all major computer languages.

The bash command-line shell is standard along with numerous general purpose command-line utilities and editors for file creation, modification, and examination. Access via wireless or Ethernet over a secure-shell connection allows "headless" operation from a remote computer. All types of networking protocols and utilities are supported. A Raspberry Pi can be configured to be a network web and mail server with a sophisticated firewall system, all using free, open-source software.

A graphic user interface (GUI) is available using either of two widget libraries, QT or GTK+ with abundant on-line examples and documentation so that the anyone can create their own graphical applications.

Advanced programs for desktop publishing, spreadsheet calculations, photo and video editing, and vector graphics editing are available, but with the RAM memory limitation of 500 MB and the processor speed of 700 MHz, those programs are sluggish. It is best to do that type of work on a faster computer with several GB of RAM.

The processor GPIO pins (Details at *http://elinux.org/RPi_BCM2835_GPIOs*) and camera are most easily accessed via C or Python libraries allowing efficient input from sensors, control of switches and motors, and interfacing with dedicated microprocessors. This hardware hacking is where the Raspberry Pi really shines and will be the focus of the remainder of this presentation.

The diagram on the next page shows the 26 pins on the GPIO header. Certain pins are for power: ground (labeled GND), +3.3 V (labeled as 3V3) and + 5.0 V (labeled as 5V0). It is common for processor pins to have multiple purposes selected by program control, but 7 pins are easiest to use general purposes:

P1 Header #	P1-11	P1-12	P1-13	P1-15	P1-16	P1-18	P1-22
GPIO #	GPIO 17	GPIO 18	GPIO 27	GPIO 22	GPIO 23	GPIO 24	GPIO 25

The diagram also shows how certain groups of pins have alternate functions that can perform various types of communication or control protocols. Pin P1-7 (GPIO 4) is usually set to supply a clock signal to external circuits, but can be used for other purposes.

When connecting circuits to these pins, it is best to have the Pi powered off and to wear a ground strap. Of course, you should make sure that you correctly locate the desired pin.

Raspberry Pi GPIO Cheat Sheet

CYNTECH COMPONENTS
ELECTRO-MECHANICAL SPECIALISTS

PIMORONI

I²C

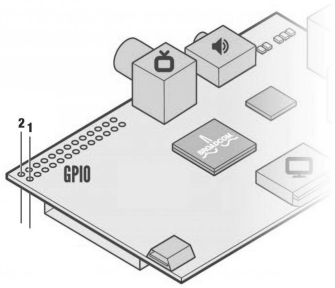
A low-speed interface used to communicate with multiple simple devices and sensors via a two-wire interface.

Inter-Integrated Circuit (I²C) is a serial bus interface which supports multiple devices and only requires two wires for communication (no separate clock or device select needed). It is, however, limited to relatively low speeds (usually 10-100kbit/s).

CLK

Clock signals are used to provide a pulse that can synchronise different parts of a system that perform actions which are time sensitive to each other.

GPCLK0 is a general purpose clock that generates a square-wave clock signal up to a maximum frequency of around 75MHz.



UART

The UART pins on the Raspberry Pi are primarily provided for access to the serial console which is a relatively advanced feature that most people won't need to use.

Universal Asynchronous Receiver/Transmitter (UART) is a method of transmitting data over a serial connection. Both of the communicating devices contains a shift register that converts the bytes of data being transmitted into a stream of bits.

PWM

Provides an 'analogue style' supply that can be used for controlling motors and LEDs.

With PWM (pulse-width modulation) the amount of power delivered to the device is controlled by switching the supply on and off very quickly, typically thousands of times a second.

			3V3	1	2	5V0	
	I²C	GPIO 2	SDA0	3	4	5V0	
		GPIO 3	SCL0	5	6	GND	
	CLK	GPIO 4	GPCLK0	7	8	TXD	GPIO 14
			GND	9	10	RXD	GPIO 15
		GPIO 17	P17	11	12	PWM	GPIO 18
		GPIO 27	P27	13	14	GND	
		GPIO 22	P22	15	16	P23	GPIO 23
			3V3	17	18	P24	GPIO 24
	SPI	GPIO 10	MOSI	19	20	GND	
		GPIO 9	MISO	21	22	P25	GPIO 25
		GPIO 11	SCLK	23	24	CE0	GPIO 8
		GND	GND	25	26	CE1	GPIO 7

UART

PWM

Original (Rev 1) Raspberry Pi users:
The original Raspberry Pi had slightly different GPIO pin numbering. GPIO 2 was GPIO 0, GPIO 3 was GPIO 1, and GPIO 27 was GPIO 21.

SPI

Often used to read more complicated sensors, drive simple displays, or communicate between devices.

Serial Peripheral Interface Bus (SPI) is a synchronous full-duplex (two way) serial connection. Communication happens between a master device and slave device with the master device providing synchronisation.

The data is transmitted on the MOSI (master-out, slave-in) and MISO pins (master-in, slave-out) pins. Each transmission is synchronised by a clock pulse on SCLK.

Making an LED Blink

An LED (light-emitting diode) gives off light when an electrical current is sent through it in a specified direction. If the current is too high, the LED is destroyed; if it is too low no significant light is produced. I will use an LED which requires about 2 mA of current. If connected directly to the 3.3 V terminals of the Raspberry Pi GPIO pins, it will draw too much current and will be destroyed. The excessive current could also possibly damage the processor connection driving that pin. The solution is to put a suitable resistor in series with the LED with a value given by $R = \frac{3.3\ V - 1.4\ V}{0.002\ \text{A}} = 950\ \Omega$ where the 1.4 V value is typical of ordinary LEDs irrespective of their current rating. A 1000 Ω resistor is close enough to this optimum value. It is typical that one of the two leads of an LED is longer than the other to allow easy identification of the lead that must be connected to the positive voltage coming from the GPIO pin. Although it doesn't matter which lead has the resistor connected to it, I usually connect it to the positive lead. Again, be sure that the resistor is in series with a lead of the LED when connecting it to 3.3 V. If the resistor is not there, too much current will flow and the LED will instantly be silently destroyed.

The following C program assumes that the positive side of the LED (via the resistor!) is connected to GPIO pin 11 which is pin 11 on the Pi's GPIO header (see diagram). The freely-available bcm2835 C library needs to be installed (See *http://www.airspayce.com/mikem/bcm2835/*).

```
// blink.c
//
// Example program for bcm2835 library
// Blinks a pin on an off every 0.5 secs
//
// After installing bcm2835, you can build this
// with something like:
// gcc -o blink blink.c -l bcm2835
// sudo ./blink
//
// Or you can test it before installing with:
// gcc -o blink -I ../../src ../../src/bcm2835.c blink.c
// sudo ./blink
//
// Author: Mike McCauley
// Copyright (C) 2011 Mike McCauley
// $Id: RF22.h,v 1.21 2012/05/30 01:51:25 mikem Exp $
#include <bcm2835.h>
// Blinks on RPi Plug P1 pin 11 (which is GPIO pin 17)
#define PIN RPI_GPIO_P1_11
int main(int argc, char **argv)
{
    if (!bcm2835_init())
        return 1;
    // Set the pin to be an output
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);
    // Blink
    while (1)
    {
        // Turn it on
        bcm2835_gpio_write(PIN, HIGH);
        // wait a bit
        bcm2835_delay(500);
        // turn it off
        bcm2835_gpio_write(PIN, LOW);
        // wait a bit
        bcm2835_delay(500);
    }
    bcm2835_close();
    return 0;
}
```

Using an Ultrasonic Range Detector

This example is written in the Python language using a library package designed for the Raspberry Pi. It GPIO pin 23 to tell an inexpensive ultrasonic detector board to emit an ultrasonic pulse and then uses GPIO pin 24 to measure the time before the echo is detected. It then calculates and prints the distance. This is repeated 1000 times.

```
#!/usr/bin/python
#+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
#|R|a|s|p|b|e|r|r|y|P|i|-|S|p|y|.|c|o|.|u|k|
#+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
#
# ultrasonic_1.py
# Measure distance using an ultrasonic module
#
# Author : Matt Hawkins
# Date   : 09/01/2013

# Import required Python libraries
import time
import RPi.GPIO as GPIO

# Use BCM GPIO references
# instead of physical pin numbers
GPIO.setmode(GPIO.BCM)

# Define GPIO to use on Pi
GPIO_TRIGGER = 23
GPIO_ECHO    = 24

print "Ultrasonic Measurement"

# Set pins as output and input
GPIO.setup(GPIO_TRIGGER,GPIO.OUT)  # Trigger
GPIO.setup(GPIO_ECHO,GPIO.IN)      # Echo

# Set trigger to False (Low)
GPIO.output(GPIO_TRIGGER, False)

# Allow module to settle
time.sleep(0.5)

def getDistance():
    # Send 10us pulse to trigger
    GPIO.output(GPIO_TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)
    start = time.time()

    while GPIO.input(GPIO_ECHO)==0:
      start = time.time()

    while GPIO.input(GPIO_ECHO)==1:
      stop = time.time()

    # Calculate pulse length
    elapsed = stop-start

    # Distance pulse travelled in that time is time
    # multiplied by the speed of sound (cm/s)
    # Speed of sound is nominally 34300 cm/s
    distance = elapsed * 34500

    # That was the distance there and back so halve the value
    distance = distance / 2

    print "Distance : %.1f" % distance
    time.sleep(0.02)

for i in range(1000):
    getDistance()

# Reset GPIO settings
GPIO.cleanup()
```

Using the HD Camera

There are two special connections on the Raspberry Pi, a CSI-2 (camera serial interface, revision 2) connector and a DSI (display serial interface). At the present only the camera interface is supported, but it is expected that support for the DSI is forthcoming.

There are two camera modules that can be used with the CSI-2 connector, they are identical except that one has an infra-red filter (Raspberry Pi Camera) and the other doesn't (Raspberry Pi PiNoIR Camera). Normal cameras have an infra-red filter because the camera chips are sensitive to near infra-red light, but such light with its longer wavelength introduces some blurring into the image. The infra-red filter, which is not user removable, prevents the infra-red light from getting to the sensor chip. The infra-red sensitivity, however, can be very useful. Biologists can check the health of plants better by looking at the reflected infra-red rather than reflected green light. Night surveillance cameras can use the infra-red sensitivity along with infra-red floodlights inconspicuously monitor activity in the dark. The PiNoIR camera is the one sensitive to infra-red in spite of its name which suggests otherwise. (The name really means "no IR filter.")

It is important not to be confused by the difference between sensitivity to "near" infra-red where an IR floodlight must be used to see at night and sensitivity to "far" infra-red which allows body heat to be detected without any lighting. The PiNoIR is only sensitive to near infra-red.

Both cameras use the same software and otherwise have the same specifications provided earlier in these notes.

The commands to take pictures or videos have a great many options, but the simplest using defaults are

`raspistill -o SomeName.jpg`	Take a single 2592x1944 picture, compress to jpeg and write to default.jpg
`raspistill -tl 2000 -t 10000 -o timelapse%04d.jpg`	Take a sequence of 2592x1944 images every 2 s over a total time of 10 s, writing files in format `timelapse0001.jpg`, timelapse0002.jpg, etc.
`raspivid -t 10000 -o SomeName.h264`	Make a 1920x1080x25fps video lasting 10 seconds at HD

When using the infra-red capability using modest infra-red lighting, it is useful to set the exposure to "night" by adding the option "-ex night".

Videos are played using the omxplayer as follows, but is is often useful to improve the packaging of the video by using a program called MP4Box as follows:

```
MP4Box -add SomeName.h264 SomeName.mp4
```

If MP4Box is not yet installed, you can install it by doing

```
sudo apt-get install gpac
```

then play it using

```
omxplayer SomeName.mp4
```

The ability to have full-programming capability with a quality camera has allowed me to set up a "critter cam" monitoring a gully on the ranch in the foothills where I live. The python program controlling the camera calculates the sunrise and sunset times using information about the Earth's elliptical orbit about the sun and the precise location of the ranch. It then starts the camera 18 minutes before sunrise, takes 2 frame/sec videos all day long (usually in 1 hour files), and then ends 18 minutes after sunset. These videos can then be watched at 12x normal speed to look for interesting critters.

Open source programs are available to look for motion in videos, but human inspection is hard to beat as there is much uninteresting motion in the videos.

Setting the critter cam up near a water source or an established wild animal trail while using ultrasonic detectors to start video clips is another option. The Pi can also be programmed to turn on infra-red floodlights for night videos.

The Gertboard Expansion Board

My favorite expansion board for the Raspberry Pi is called the Gertboard after its creator Gert van Loo. It was designed to demonstrate a wide variety of hardware control possible via the Raspberry Pi GPIO ports. Its capabilities have been listed earlier in these notes and allows control of motors, lights, and relays, output and input of analog voltages, and interaction with push-buttons, but the part I find most exciting is that it contains a very capable microprocessor, the ATmega328P. The Raspberry Pi can compile assembly code, send it to the microprocessor, and receive data acquired. This is an excellent way for students to become familiar with assembly language programming of microprocessors.

650 pages of documentation for the ATmega328P are freely available (only about 320 pages are crucial to the programmer) so that all aspects of its myriad capabilities can be exercised by the inquisitive student at no cost other than the Gertboard.

The "Hello, World!" of assembly language programming is one that blinks an LED. Here is its code named `BlinkingLEDUsingTimer0.asm`:

```
.NOLIST
.INCLUDE "m328Pdef.inc"
.LIST
.org    0x0000
; Have timer/counter-0 cause a toggle of the state of OC0A (PIND6) each time the count
;    becomes 0.  Connect PD6 to B1 and jumper B1 output to see blinking.
ldi     r16,(0<<COM0A1)|(1<<COM0A0)|(0<<COM0B1)|(0<<COM0B0)|(0<<WGM01)|(0<<WGM00)
out     TCCR0A,r16

sbi  DDRD,DDD6         ; Prepare PIND6 to be an output to drive LED.

; System clock, by default, is its calibrated RC oscillator operating at 8 MHz.
; The pre-scaler, by default, is set to divide by 8 so ClkIO is 1 uS.
; The following sets the timer/counter-0 divider to divide by 1024 so it ticks every 1024 uS
; Since this is an 8-bit counter it will rollover every 256*1024 uS and toggle the LED.
; The LED will then have a cycle time of 512*1024 us = 524288 uS
; Note: If CS00 is set to 1 instead, the blinking cycle time will be 262144 uS.
ldi  r16,(0<<FOC0A)|(0<<FOC0B)|(0<<WGM02)|(1<<CS02)|(0<<CS01)|(1<<CS00)
out  TCCR0B,r16

; Interrupts will be disabled by default

; To minimize power consumption, disable power to all subsystems except the timer/counter-0.
; Note: Default is 0x00, all subsystems powered.
ldi     r16,(1<<PRTWI)|(1<<PRTIM2)|(0<<PRTIM0)|(1<<PRTIM1)|(1<<PRSPI)|(1<<PRUSART0)|(1<<PRADC)
sts     PRR,r16

ldi  r16,(0<<SM2)|(0<<SM1)|(0<<SM0)|(1<<SE)  ; Go to idle sleep
out  SMCR,r16
SLEEP             ; Since interrupts are disabled, it will never wake up
```

This program is assembled on the Pi using the following instruction:

```
avra  BlinkingLEDUsingTimer0.asm
```

which produces several files, one of which gives the numerical codes that need to be loaded into the flash program memory of the microprocessor. These codes are sent to it via an SPI (serial peripheral interface) connection using a python program called `uploadingViaSPI.py`. The microprocessor starts running the code once the uploading is complete.

The next step is to learn about hardware interrupts which are crucial to the efficient use of microprocessors for controlling equipment. This is best explained by an extension of the above example to use an interrupt when the timer reaches an overflow condition and wraps around. The program is called `BlinkingLEDUsingTimer0WithOVFInterrupt.asm`

```
.NOLIST
.INCLUDE "m328Pdef.inc"
.LIST
.DEF overflowCount = R22
.DEF statusFlags   = R23
```

```
.org 0x0000
jmp  RESET
     reti     ; EXT_INT0
     nop
     reti     ; EXT_INT1
     nop
     reti     ; PCINT0
     nop
     reti     ; PCINT1
     nop
     reti     ; PCINT2
     nop
     reti     ; WDT
     nop
     reti     ; TIM2_COMPA
     nop
     reti     ; TIM2_COMPB
     nop
     reti     ; TIM2_OVF
     nop
     reti     ; TIM1_CAPT
     nop
     reti     ; TIM1_COMPA
     nop
     reti     ; TIM1_COMPB
     nop
     reti     ; TIM1_OVF
     nop
     reti     ; TIM0_COMPA
     nop
     reti     ; TIM0_COMPB
     nop
jmp  TIM0_OVF ; TIM0_OVF
     reti     ; SPI_STC
     nop
     reti     ; USART_RXC
     nop
     reti     ; USART_UDRE
     nop
     reti     ; USART_TXC
     nop
     reti     ; ADC
     nop
     reti     ; EE_RDY
     nop
     reti     ; ANA_COMP
     nop
     reti     ; TWI
     nop
     reti     ; SPM_RDY
     nop

;*********************  Reset Handler  **********************

RESET:                ; Program initialization
ldi  r16,high(RAMEND)  ; Initialize stack pointer
out  SPH,r16
ldi  r16,low(RAMEND)
out  SPL,r16

; Have timer/counter-0 cause a toggle of the state of OC0A (PIND6) each time the count
;    becomes 0.
; Also, use the timer's overflow interrupt to count interrupts in a byte-sized variable,
;    and then to toggle PIND3 each time that count wraps around to zero.
; System clock, by default, is the calibrated RC oscillator operating at 8 MHz.
; The pre-scaler, by default, is set to divide by 8 so ClkIO is 1 uS.
; The timer/counter-0 divider is set to divide by 64 so it will tick every 64 uS
; Since this is an 8-bit counter it will rollover every 256*64=16384 uS and toggle PIND6.
; The overflow interrupt will be triggered and the overflow interrupt handler will toggle PIND3
;    each time it rolls over, every 256*256*64=4194304 uS or about every 4 seconds.
;    If the overflow count rolls over PIND3 will be toggled.  To see it change,
;    connect PD3 to Buf2 on the Gertboard, and set the output jumper on B2.
; The LED will then have a cycle time of 2*256*256*64 us = 8388608 uS

ldi  r16,(0<<COM0A1)|(1<<COM0A0)|(0<<COM0B1)|(0<<COM0B0)|(1<<WGM01)|(0<<WGM00)
out  TCCR0A,r16        ; Set up the CTC (Clear Timer on Compare Match) mode.
```

```
ldi  r16,0xFF
out  OCR0A,r16          ; Set the compare value to 0xFF
sbi  DDRD,DDD6          ; Prepare PIND6 to be an output to drive LED.
sbi  DDRD,DDD3          ; Prepare PIND3 to toggle when overflowCount becomes 0

ldi  r16,(0<<FOC0A)|(0<<FOC0B)|(0<<WGM02)|(0<<CS02)|(1<<CS01)|(1<<CS00)
out  TCCR0B,r16         ; Set the timer clock divider to ClkIO/64

ldi  r16,(0<<OCIE0B)|(0<<OCIE0A)|(1<<TOIE0)
sts  TIMSK0,r16         ; Enable overflow interrupt for timer/counter-0

; To minimize power consumption, disable power to all subsystems except the timer/counter-0.
; Note: Default is 0x00, all subsystems powered.
ldi     r16,(1<<PRTWI)|(1<<PRTIM2)|(0<<PRTIM0)|(1<<PRTIM1)|(1<<PRSPI)|(1<<PRUSART0)|(1<<PRADC)
sts     PRR,r16

sei               ; Enable interrupts globally

ldi  r16,(0<<SM2)|(0<<SM1)|(0<<SM0)|(1<<SE)  ; Go to idle sleep
out  SMCR,r16
REPEAT:
SLEEP                   ; Sleep until interrupt occurs
rjmp REPEAT             ; After interrupt is handled, go back to sleep

TIM0_OVF:
in   statusFlags,SREG

inc  overflowCount
brne END_TIM0_OVF
sbi  PIND,PIND3

END_TIM0_OVF:
out  SREG,statusFlags
reti
```

A truly useful application of the Raspberry Pi and Gertboard monitors the clicks of a Geiger counter, assigns times to each click with 1 ms precision, and periodically sends the data over the SPI interface for storage on the SD card of the Raspberry Pi. It is described at

http://yosemitefoothills.com/Electronics/RaspberryPi/GeigerCounter.html

with the actual ATmega328P assembly program listing is at

http://yosemitefoothills.com/Electronics/RaspberryPi/loggingWithSPIInterrupts.asm.

The processor program uses interrupts for the timer overflow, the detection of the click pulse, when the Raspberry Pi has requested data, and when all data has finished being sent. Interrupts allow the processor to run without missing data, and use of the microprocessor allow the Raspberry Pi to do other independent tasks concurrently.

Trying to use the Raspberry Pi directly to catch the pulses of the Geiger counter would not work well because the Pi is doing many varied tasks at once like networking, displaying data, etc.