ECE 471 – Embedded Systems Lecture 8

Vince Weaver

http://www.eece.maine.edu/~vweaver

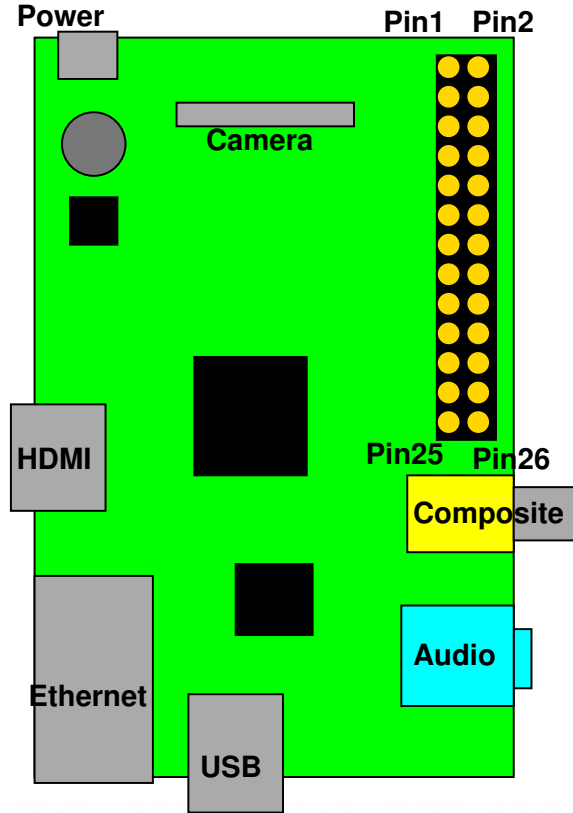vincent.weaver@maine.edu

25 September 2014

Announcements

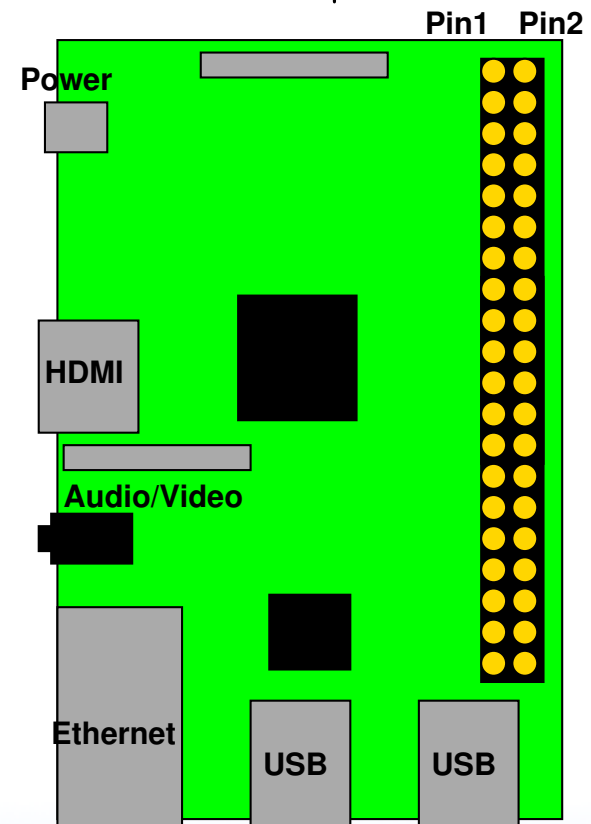- For Homework 3 you'll need a breadboard, some wire, an LED, and some resistors.

Brief Overview of the Raspberry Pi Board

Model B

Power

Pin1   Pin2

Camera

Pin25   Pin26

HDMI

Composite

Audio

Ethernet

USB

Model B+

Pin1   Pin2

Power

HDMI

Audio/Video

Ethernet

USB   USB

Rasp-pi Header

- Model B has 17 GPIOs (out of 26 pins), B+ has 9 more (out of 40)

- 3.3V signaling logic. Need level shifter if want 5V or 1.8V

- Linux by default configures some for other purposes (serial, i2c, SPI)

Rasp-pi Header

3.3V	1	2	5V
GPIO2 (SDA)	3	4	5V
GPIO3 (SCL)	5	6	GND
GPIO4	7	8	GPIO14 (UART_TXD)
GND	9	10	GPIO15 (UART_RXD)
GPIO17	11	12	GPIO18 (PCM_CLK)
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10 (MOSI)	19	20	GND
GPIO9 (MISO)	21	22	GPIO25
GPIO11 (SCLK)	23	24	GPIO8 (CE0)
GND	25	26	GPIO7 (CE1)

ID_SD (EEPROM)	27	28	ID_SC (EEPROM)
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

How you enable GPIO on STM32L

A lot of read/modify/write instructions to read current register values and then to shift/mask to write out updated bitfields.

- Enable GPIO Clock
- Set output mode for GPIO.
- Set GPIO type.
- Set pin clock speed.
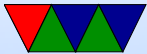- Set pin pull-up/pull-down
- Set or clear GPIO pin.

How you enable on BCM2835 (Rasp-pi)

- Documented in BCM2835 ARM Peripherals Manual

- 53 GPIOs (not all available on board)

- Similar to how done on STM32L... but we have an operating system

Letting the OS handle it for you

Linux GPIO interface

- `Documentation/gpio/sysfs.txt`

- sysfs and string based

A few low-level Linux Coding Instructions

Enable a GPIO for use

To enable GPIO 17:
write "17" to /sys/class/gpio/export
To disable GPIO 17:
write "17" to /sys/class/gpio/unexport

```c
char buffer[10];
fd=open("/sys/class/gpio/export",O_WRONLY);
if (fd<0) fprintf(stderr,"\tError enabling\n");
strcpy(buffer,"17");
write(fd,buffer,2);
close(fd);
```

Set GPIO Direction

To make GPIO 17 an input:

write "in" to /sys/class/gpio/gpio17/direction

To make GPIO 17 an output:

write "out" to /sys/class/gpio/gpio17/direction

```c
fd=open("/sys/class/gpio/gpio17/direction",O_WRONLY);
if (fd<0) fprintf(stderr,"Error!\n");
write(fd,"in",2);
close(fd);
```

Read/Write GPIO Value

To read value of GPIO 17:

read /sys/class/gpio/gpio17/value To write value of GPIO 17:

write /sys/class/gpio/gpio17/value

```
fd=open("/sys/class/gpio/gpio17/value",O_WRONLY);
if (fd<0) fprintf(stderr,"Error writing!\n");
write(fd,"1",1);
close(fd);
```

Delay

- Busy delay (like in 271).
  `for(i=0;i<1000000;i++);`
  Harder to do in C. Why?
  Compiler optimizes away.

- `usleep()` puts process to sleep for a number of microseconds. But can have issues if want exact delay. Why? OS potentially context switches every 100ms.

- Other ways to implement: Set up PWM? Timers?

Implementations

- Busy loop. Bad, burns CPU / power

- usleep(). But can take a while to respond.

- Interrupt when ready! poll()

GPIO Interrupts on Linux

May need a recent version of Raspbian.
First write "rising", "falling", or "both" to
/sys/class/gpio/gpio17/edge.
Then open and poll /sys/class/gpio/gpio17/value.

```c
struct pollfd fds;
int result;

fd=open("/sys/class/gpio/gpio18/value",O_RDONLY);
fds.fd=fd;
fds.events=POLLPRI|POLLERR;
while(1) {
    result=poll(&fds,1, -1);
    if (result<0) printf("Error!\n");
    lseek(fd,0,SEEK_SET);
read(fd,buffer,1); }
```

Debouncing!

- Pull-up / Pull-down resistor. Why?

- Noisy switches, have to debounce

- Manual, no built-in debounce like on STM32L

Why Use an Operating System?

- Provides Layers of Abstraction

  - Abstract hardware: hide hardware differences. same hardware interface for classes of hardware (things like video cameras, disks, keyboards, etc) despite differing implementation details
  - Abstract software: with VM get linear address space, same system calls on all systems
  - Abstraction comes at a cost. Higher overhead, unknown timing

- Multi-tasking / Multi-user

- Security, permissions (Linus dial out onto /dev/hda)

- Common code in kernel and libraries, no need to re-invent

What's included with an OS

- kernel / drivers – Linux definition

- also system libraries – Solaris definition

- low-level utils / software / GUI – Windows definition
  Web Browser included?

- Linux usually makes distinction between the OS Kernel
  and distribution. OSX/Windows usually doesn't.

Operating Systems Types

- Monolithic kernel – everything in one big address space. Something goes wrong, lose it all. Faster

- Microkernel – separate parts that communicate by message passing. can restart independently. Slower.

- Microkernels were supposed to take over the world. Didn't happen. (GNU Hurd?)

- Famous Torvalds (Linux) vs Tannenbaum (Minix) flamewar

Common Desktop/Server Operating Systems

- Windows
- OSX
- Linux
- FreeBSD / NetBSD / OpenBSD
- UNIX (Irix/Solaris/AIX/etc.)
- BeOS/Haiku

Embedded Operating Systems

- Microsoft WinCE, Windows Mobile
- Linux / Android
- VXworks – realtime OS, used on many space probes
- Apple iOS
- QNX – realtime microkernel UNIX-like OS, owned by Blackberry now
- Cisco iOS

Embedded Linux Distributions

- linaro – consortium that work on ARM software

- openwrt – small distro initially designed for wireless routers

- yocto – Linux Foundation sponsored embedded distro

- maemo – embedded distro originally by Nokia (obsolete)

- MeeGo – continuation of maemo, also obsolete

- Tizen – Follow up on MeeGo, by Samsung and Intel

- Ängstrom – Merger of various projects

- And many others. It's very easy to put together a Linux distribution

Linux/UNIX History

Why is Linux used in Embedded Systems?