ECE 471 – Embedded Systems
Lecture 31

Vince Weaver

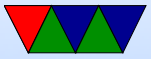
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

21 November 2022

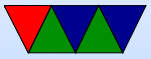
Announcements

- Don't forget project status reports due Tuesday (22nd)
  - A one-line statement of your project topic
  - A short summary of the progress you've made so far
  - List any parts you need that you don't have yet
  - List if you're willing to present early (Friday the 2rd, Monday 5th or Wednesday 7th vs Friday the 9th) (there will be some bonus for presenting early)
- Don't forget HW#9

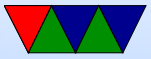


HW#9 Notes – Building Separate Files

- In previous homeworks we put everyting in one C file
- This isn't really practical for large projects
- By splitting things up into smaller files you can have some benefits
  - Easier to organize/find code
  - Can re-use code easier
  - Less chance of merge conflicts when multiple people working on project
  - Can take common code and make libraries

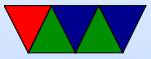


- For example in the homework, we could put temperature read code into its own file with a `double get_temperature(void)` interface
- For other C files to see this, you need to export the definition. Usually this is done by putting the advance definition `double get_temperature(void);` in a .h header file and then including it in the other files
- Note: don't put full C functions in header files. I know this is a C++ thing but it's usually frowned upon when programming in C
- Each file does not need a `main()` function, you only



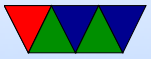
need one per combined program.

- To link the various .o files together involves the "linker". However it's easier to just let gcc do it (gcc knows how to run the linker for you) `gcc -o display_temp display.o temperature.o`
- The linker merges the .o files into one big executable, and makes sure the placeholders to functions/variables in all of the files get the right addresses/pointers to where things live in the finished executable.
- How do you make sure when you change one C file that everything that uses it is also rebuilt? A well-crafted

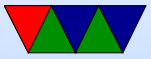


Makefile will have all these dependencies in place and will rebuild everything properly.

- Static vs Dynamic linking redux (we did discuss that earlier in the semester)

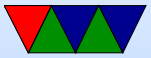


Other I/O You'll find on Embedded Boards

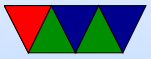


Digital Audio

- How can you generate audio (which is analog waves) with a digital computer?
- One way is PCM, Pulse Code Modulation, i.e. use a DAC.
  - Sample the sound at a frequency (say 44.1kHz), and take amplitude (16-bit audio, 64k possible values)
  - Why 44.1kHz? Nyquist theorem. Twice sample rate to reproduce properly. 22kHz roughly high end of human hearing.

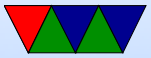


- A WAV file is basically this, has the samples (8 or 16-bit, stereo or mono) sampled at a regular frequency (often 44.1kHz) to play back, write the values to a DAC at the sample rate.



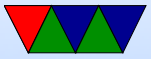
What if no DAC? (Pi has none)

- Can do PWM, Pulse-Width Modulation
- By varying the width of pulses can have the average value equal to an intermediate analog value. For example with duty cycle of 50% average value is 1/2 of Vdd
- Can be "converted" to analog either by a circuit, or just by the inertia of the coil in a speaker.



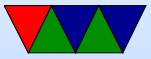
Saving space

- Can be tens of megabytes per song.
- Music can be compressed
- Lossy: MP3, ogg vorbis
- lossless AAC, FLAC



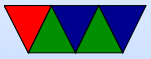
PWM GPIO on Pi

- You can't get good timings w/o real-time OS
- Available on GPIO18 (pin 12)
- Can get 1us timing with PWM in Hardware
  Software: 100us Wiring Pi, less with GPIO interface.
- Which would you want for hard vs soft realtime?
- Other things can do? Beaglebone black as full programmable real-time unit (PRU)
  200MHz 32-bit processor, own instruction set, can control pins and memory, etc.

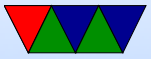


Linux Audio

- In the old days audio used to be just open `/dev/dsp` or `/dev/audio`, then `ioctl()`, `read()`, `write()`
- These days there's ALSA (Advanced Linux Sound Architecture)
  The interface assumes you're using the ALSA library, which is a bit more complicated.
  - Handles things like software mixing (if you want to play two sounds at once)
  - Other issues, like playing sound in background

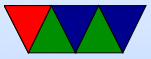


- On top of that is often another abstraction layer, pulse-audio
- A mixer interface to pick volumes, muting
- For quick hack can use `system()` to run a command-line audio player like `aplay`
- Better idea might be to use a library such as SDL-mixer which handles all of this in a portable way with a nice library interface.



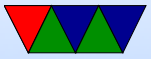
Pi Limitations

- Pi interface is just a filter on two of the PWM GPIO outputs
- Also can get audio out over HDMI.
- If you want better, can get i2s hat
- Pi lacks a microphone input, so if want audio in on your pi probably need a USB adapter.



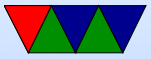
i2s

- PWM audio not that great
- i2s lets you send packets of PWM data directly to a DAC
- At least 3 lines. bit clock, word clock (high=right/low=left stereo), data
- Pi support i2s on header 5



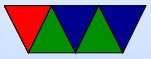
SD/MMC

- MultiMediaCard (MMC) 1997
- Secure Digital (SD) is an extension (1999)
- SDSC (standard capacity), SDHC (high capacity), SDXC (extended capacity), SDIO (I/O)
- Standard/Mini/Micro sizes
- SDHC up to 32GB, SDCX up to 2TB
- Support different amounts of sustained I/O. Class rating 2, 4, 6, 10 (MB/s)
- Patents. Need license for making.



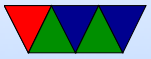
SD/MMC Hardware Interface

- 9 pins (8 pins on micro)
- Starts in 3.3V, can switch to 1.8V
- Write protect notch. Ignored on pi?



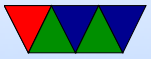
SD/MMC Software Interface

- SPI bus mode
- One bit mode – separate command and data channels
- Four-bit mode
- Initially communicate over 1-bit interface to report sizes, config, etc.
- DRM built in, on some boards up to 10% of space to handle digital rights



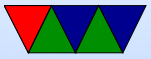
SDIO

- SDIO – can have I/O like GPS, wireless, camera
- Can actually fit full Linux ARM server on a wireless SDIO card

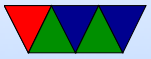


eMMC

- eMMC = like SD card, but soldered onto board

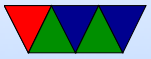


More Embedded Board Busses/Interfaces



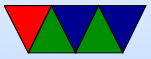
Starting Programs at Boot

- init process starts first
- Traditionally would start various shell scripts under /etc (the name and order of these can vary a lot)
- Possibly with advent of systemd this will change
- Currently you can still put things you want to run at start in `/etc/rc.local`



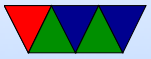
Wii Nunchuck

- Fairly easy to interface
- Put onto i2c bus. Device 0x52
- Send handshake to initialize. Use longer one (0xf0/0x55/0xfb/0x00) not the simpler one you might find(0x40/0x00). This works on generic nunchucks and possibly also disables encryption of results.
- To get values, send 0x00, usleep a certain amount, and read 6 bytes. This includes joy-x, joy-x, accelerometer



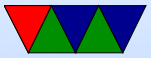
x/y/z and c and z button data. More info can be found online.

byte0 = joy-x, byte1 = joy-y, byte2 = top8 acc x, byte3 = top8 acc y, byte4 = top8 acc z, byte 5 is bottom 2 z,y,x then button c and z (inverted?)

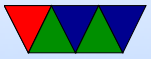


Linux and Keyboard

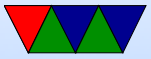
- Old ps/2 keyboard just a matrix of keys, controlled by a small embedded processor.
  Communication via a serial bus. Returns "keycodes" when keypress and release and a few others.
- Many modern keyboards are USB, which requires full USB stack. To get around needing this overhead (for BIOS etc) support bit-bang mode. OS usually has abstraction layer that supports USB keyboards same as old-style



- Linux assumes "CANONICAL" input mode, i.e. like a teletype. One line at a time, blocking input, wait until enter pressed.
- You can set non-CANONICAL mode, async input, and VMIN of 1 to get reasonable input for a game. Arrow keys are reported as escape sequences (ESCAPE-[-A for up, for example).
- Even lower-level you can access "RAW" mode which gives raw keycode events, etc.
- See the `tcgetattr()` and `tcsetattr()` system calls
- There are libraries like ncurses that abstract this a bit.

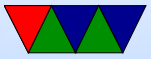


Also GUI and game libraries (SDL).



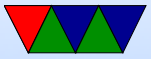
Faking Linux Input Events

- How to insert input events into Linux, i.e. have a software program fake keyboard/mouse/joystick events.
- Linux supports a "uinput" kernel driver that lets you create user input.
- There is some info on a library that makes this easier here: `http://tjjr.fi/sw/libsuinput/`
- It has examples for keyboard and mouse. Joystick should be possible but there's no sample code provided.
- Python wrappers seem to exist too.

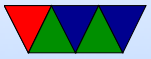


Camera Port

- The SoC has dedicated hardware for driving cameras
- 5megapixel, CSI port (Camera Serial Interface) plus i2c bus to command it.
- Can read data in parallel, directly, without needing USB overhead.
- These chips often used in cell-phones, so makes sense to have support for camera-phone without extra chip being needed.

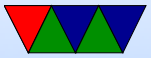


Touchscreen Display Port

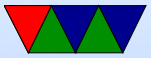


UART – serial port

- Note: Asynchronous, no clock (unlike USART) how do both sides agree on speed?
- Often useful on embedded boards and old systems, might be only way to reliably connect
- RS-232, originally for teletypes
- 3-15V high, -3 to -15V low
- start/stop bits, parity, bit-size
- Hardware vs Software flow control
- Speeds 300bps - 115000bps and beyond

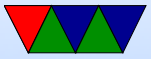


- 50feet (15m) w/o special cables
- 3-pin version (transmit, receive ground). Also 5-pin HW flow control (CTS/RTS). Can have 2-pin version if only want to transmit
- These days often hook up USB connector
- What does 9600N81 mean?



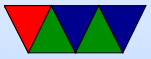
Pi Serial Ports

- Raspberry Pi has two serial ports, good one and lousy one
  They switched them up with Pi3
- Pi does TTL (5v/0) not RS232
- Does support HW flow control, but need to activate those pins custom, is a bit complicated
- Use TTL to USB serial converter usually.
  Tell story of the prolific bricking the firmware?



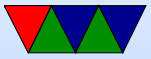
Pi SMI

- `https://iosoft.blog/2020/07/16/raspberry-pi-smi`
- Secondary Memory Interface
- Available on Pis
- Allows creating wide parallel bus out of GPIOs
- Not well documented

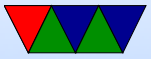


Bluetooth

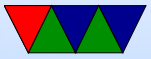
- Basic unit: piconet, master node and up to seven *active* slave nodes within 10m
- Many can exist in an area, and can be connected by a bridge. Connected piconets are called a scatternet
- There can also be up to 255 “parked” nodes in a picnoet
- When parked, can only respond to activation on beacon
- Hold and siff?
- Slaves designed to be cheap, so dumb. Master is smart and runs them. slave/slave communication not possible



- Master broadcasts clock 312.5us. Master transmits in even, slave in odd.
- Radio layer – 2.4GHz, 10 meters. 79 channels of 1MHz.
- pairing
- Bluetooth V1.1 has 13 different application protocols.
- Bluetooth 4.0 (Bluetooth Low Energy) (2010)
  - 25Mbps/200 feet
  - Entirely new stack, designed for low power rapid setup links
  - Not backwards compatible, but same frequency range
  - New profiles



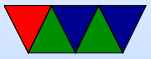
- Linux interface: depends on type. Filetransfer/obex. Audio (looks like an audio driver) network device, serial device



Bluetooth and Linux

- Two competing stacks, BlueZ and Affix

```
sudo bluetoothctl
[sudo] password for vince:
[NEW] Controller B8:27:EB:52:DD:E8 linpack-test
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# scan on
Discovery started
[CHG] Controller B8:27:EB:52:DD:E8 Discovering:
```



```
[NEW] Device AC:37:43:89:4C:02 HTC BS 02CA47
[NEW] Device AC:37:43:89:2F:86 HTC BS 86B06E
[CHG] Device AC:37:43:89:2F:86 RSSI: -90
[bluetooth]# scan on
Failed to start discovery: org.bluez.Error.InPr
[bluetooth]# connect AC:37:43:89:4C:02
```

- obexpushd. Appears as serial port?