ECE 471 – Embedded Systems
Lecture 31

Vince Weaver

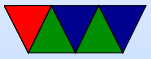
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

21 November 2025

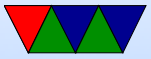
Announcements

- Don't forget projects
- Don't forget HW#9
- Project status report due 24th (Monday before Thanksgiving)

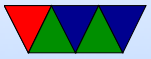


Project Status – Due Monday (24th)

- 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 5th, Monday 8th or Wednesday 10th vs Friday the 12th) (there will be some bonus for presenting early)
- Identify the problem: this can be brief, the idea is you picked the topic, but can you identify the hardware/ software problem you need to solve to complete it.

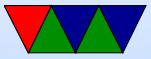


- For example: you're making a line-following robot.
- The hardware problem to be solved would be assembling a mobile platform that can detect a line
- The software problem to be solved is code that can read the sensor, detect the line, and provide feedback to the motors



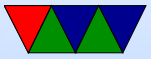
Other I/O You'll find on Embedded Boards

Prioritized for things people might be using for the projects



Starting Programs at Boot (old way)

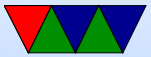
- `init` process starts first
- Traditionally would start various shell scripts under /etc (the name and order of these can vary a lot)
- Currently you can still put things you want to run at start in the `/etc/rc.local` shell script
- At some point this will stop working, due to systemd



Starting Programs at Boot (new way)

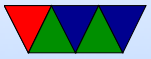
- Used when init is `systemd`
- Create a systemd unit file, in this example named `yourservice`

```
[Unit]
Description=ECE471 Startup

[Service]
Type=simple
ExecStart=/usr/local/bin/ece471_project
```



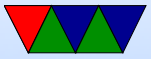
```
Restart=on-failure
KillMode=process

[Install]
WanteBy=multi-user.target
```

- Copy unit file to `/etc/systemd/system/`
- Start with `systemctl start youservice`
- Enable at every boot `systemctl enable youservice`



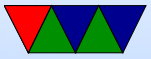
i2c/SPI boards

- If you borrowed a sensor, your best bet is the datasheet
- These are all adafruit breakouts so relatively well documented
- They have sample code, but for arduino or python
- If you run into trouble let me know. Some boards I have sample code for, others might take a bit longer
- Also if device doesn't work let me know. Actually happened last year one of the sensors was broken



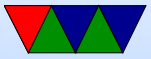
Wii Controllers

- If you want some interesting I/O you can find old controllers for the wii console
- The wiimotes themselves are Bluetooth and in theory can talk to a Pi (I haven't done this yet)
- The various sub-adapters talk i2c (like the nunchuck, or classic controller) and can easily be used with a pi



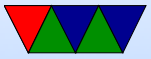
Wii Nunchuck

- Fairly easy to interface (I have breadboard adapters)
- 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-y, accelerometer x/y/z and c and z button data. More info can be found



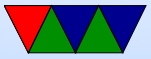
online.

- byte0 = joy-x
- byte1 = joy-y
- byte2 = top 8 bits of accelerometer x
- byte3 = top 8 bits of acc y
- byte4 = top 8 bits of acc z
- byte 5 is bottom 2 z,y,x then button c and z (inverted?)

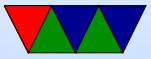


Keyboards

- Keyboards are just a matrix of keys that get scanned, ideally tell you when key pressed and when released
- Even older keyboards with embedded systems with microcontrollers doing the scanning
- Desktop PC systems went PC, XT, PS/2, then USB (also wireless/bluetooth)
- PS/2 is just a serial-like bus, clock/data and you can in theory bitbang it over GPIO but maintaining timing can be tough

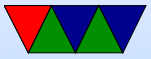


- USB is a lot more complex
- Luckily Linux abstracts all this



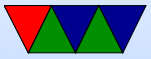
Linux and Keyboard

- 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.



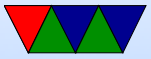
- See the `tcgetattr()` and `tcsetattr()` system calls
- There are libraries like ncurses that abstract this a bit. Also GUI and game libraries (SDL), pygame

```
struct termios oldattr, newattr;
tcgetattr (0, &oldattr); // save current
tcgetattr (0, &newattr);
newattr.c_lflag &= ~ICANON;
newattr.c_cc[VMIN] = 1;
newattr.c_lflag &= ~ECHO;
tcsetattr (0, TCSANOW, &newattr);
fcntl (0, F_SETFL, fcntl (0, F_GETFL) | O_NONBLOCK);
...
tcsetattr (0, TCSANOW, &oldattr); // restore original
```



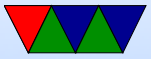
Faking Linux Input Events

- 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.

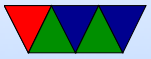


Digital Audio Output

- 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.

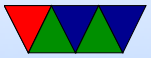


- A WAV file is basically this, has the samples (8 or 16-bit, stereo or mono) sampled at a regular frequency (often 44.1kHz) to play back, write the values to a DAC at the sample rate.



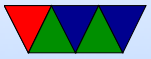
What if no DAC? (Pi has none)

- 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.
- TODO: plot



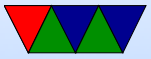
Saving space with audio compression

- This was more relevant before everyone put all their music in the cloud
- Can be tens of megabytes per song.
- Music can be compressed
- Lossy: MP3, ogg vorbis
- lossless AAC, FLAC

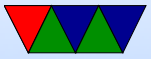


PWM GPIO on Pi

- You can't get good timings w/o real-time OS
- PWM lets you set a continuous cycle with duty cycle on GPIO
- On pi 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)

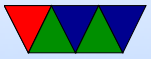


200MHz 32-bit processor, own instruction set, can control pins and memory, etc.



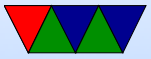
Linux Audio

- 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



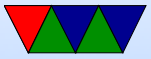
Linux Audio – Libraries

- On top of ALSA is often another abstraction layer
  - pulse-audio – from the same people who made systemd
  - A mixer interface to pick volumes, muting
- More Recently people using pipewire
  aside on DSP to make tiny speakers sound good, intentionally over-drive to sound good but have to model temperature, m1 Linux support has to be careful or you can blow out speakers in software
- JACK interface for low-audio sound between processes



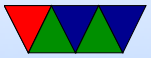
Linux Audio – Programming

- 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.



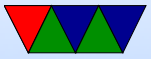
Audio Input

- Pi lacks a microphone or aux input
- if want audio in on your pi probably need a USB soundcard
- Linux can mostly abstract this
- I also have some microphone boards (SPI/i2c?)
- electret vs MEMS microphones
- Can also use ADC board we have, but need extra circuitry to match to the proper input signal



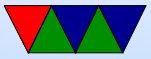
Pi Limitations

- 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

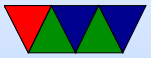


i2s

- 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

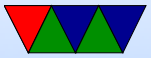


Other I/O You'll find on Embedded Boards



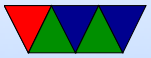
SD/MMC

- 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, SDXC up to 2TB
- Support different amounts of sustained I/O. Class rating 2, 4, 6, 10 (MB/s)
- Patents. Need license for making.



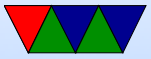
SD/MMC Hardware Interface

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



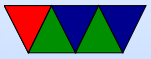
SD/MMC Software Interface

- 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



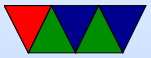
SDIO

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



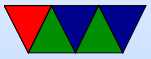
eMMC

- eMMC = like SD card, but soldered onto board

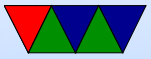


Camera Port

- The SoC has dedicated hardware for driving cameras
- 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.
- Might need to use special tool to get still images (mmal

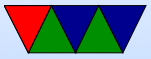


interface), until recently not using more common video-4-linux API



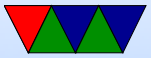
Camera Models

- 2013 – 5 Megapixel
- 2016 – 8 Megapixels (Model2)
- 2023 – 12 Megapixels (Model3) (wide/narrow versions)
- NoIR – infrared filter removed so can take IR images
- GS – Global (vs rolling) shutter version
- 2024 – Pi AI Camera



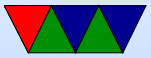
Touchscreen Display Port

- DSI
- Touchscreen display that can make a Pi look sort of like a smartphone
- GPU can output to it directly

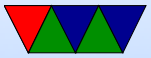


UART – serial port

- 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

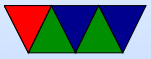


- 50feet (15m) w/o special cables
- 3-pin version (transmit, receive ground). Also 5-pin HW flow control (CTS/RTS). Can have 2-pin version if only want to transmit
- These days often hook up USB connector
- What does 9600N81 mean?



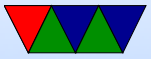
Pi Serial Ports

- Pi originally had two serial ports: good one and lousy one
  They switched them up with Pi3 (due to bluetooth)
- Pi4 adds a bunch more of them
- 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.



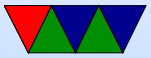
Pi SMI

- `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



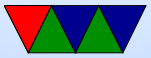
Other busses

- Beyond this we didn't cover in class this year



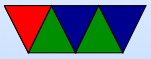
Networking

- Ethernet (Power over Ethernet)
- WiFi
- Bluetooth (see below)
- Take ECE435?

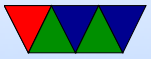


Bluetooth

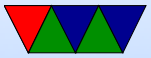
- Basic unit: piconet, master node and up to seven *active* slave nodes within 10m
- Many can exist in an area, and can be connected by a bridge. Connected piconets are called a scatternet
- There can also be up to 255 “parked” nodes in a picnoet
- When parked, can only respond to activation on beacon
- 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



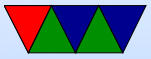
- Linux interface: depends on type. Filetransfer/obex. Audio (looks like an audio driver) network device, serial device



Bluetooth and Linux

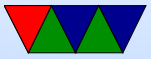
- Two competing stacks, BlueZ and Affix

```
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:
```



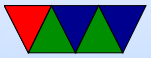
```
[NEW] Device AC:37:43:89:4C:02 HTC BS 02CA47
[NEW] Device AC:37:43:89:2F:86 HTC BS 86B06E
[CHG] Device AC:37:43:89:2F:86 RSSI: -90
[bluetooth]# scan on
Failed to start discovery: org.bluez.Error.InPr
[bluetooth]# connect AC:37:43:89:4C:02
```

- obexpushd. Appears as serial port?



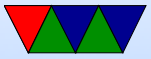
CANbus

- Automotive. Introduced by BOSCH, 1983
- One of OBD-II protocols
- differential, 2 wires, 1MBps important things like engine control
- single wire, slower cheaper, hvac, radio, airbags

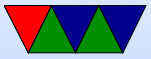


CANbus Protocol

- id, length code, up to 8 bytes of data id (usually 11 or 29 bits) type and who is sending it. Also priority (lower is higher) length is 4 bits. some always send 8 and pad with zeros
- Type is inferred from id. Can be things like engine RPM, etc
- DBC database has the ids and values. ASCII text database, hard to get legally. g

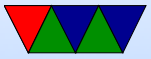


- Dominant/Recessive. Message with lowest ID wins arbitration.
- CAN-FD – extended version with larger sizes



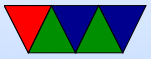
CANbus Linux

- Can4linux – `open("/dev/can0"); read(); write();` External project?
- SocketCAN – contributed by Volkswagen. In kernel. Uses socket interface. /Documentation/networking/can.txt



CANbus on Pi

- Not by default
- Can get SPI to CANbus adapters



USB

- We might cover this in a later lecture if there's time