ECE 471 – Embedded Systems
Lecture 23

Vince Weaver

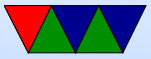
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

31 October 2025

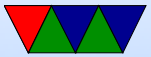
Announcements

- HW7 is due today
  Some people were having trouble with parts or Pi systems, try to test things out earlier rather than later
- HW8 (1-wire) will be posted
- Project topics due soon (7th)
- Please give me advance warning if want to borrow project parts so I can make sure I actually have it.

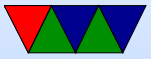


One-Wire Bus

- From Dallas Semiconductor (bought by Maxim in 2001)
- One data wire plus ground (how do you get power?)
  - Open collector, data line pulled high
  - Devices have capacitor to provide power when data line low "parasite power"
  - Low speed data and power over one wire (you also need ground)
- One controller
- Can have many devices

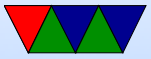


- 16.3kbit/s
- Up to 300m twisted pair (phone or Ethernet wire) [why is twisted better?]



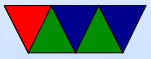
One-Wire Bus users

- Temp probes
- (old) Apple magsafe connector
- eeproms
- Single bit of GPIO
- Java rings?



One-Wire Protocol

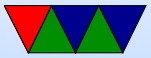
- Each device has unique 64-bit ID:
  - 8-bits of type
  - 48 bit ID
  - 8-bit CRC
- Typically 8-bit command followed by 8-bit data chunks



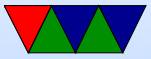
One-Wire Protocol – Detailed

Open Collector (BJT equivalent of MOSFET Open Drain)

- Write
  - Write 1 – Controller pull bus low for 1-15us
  - Write 0 – Controller pull bus low for 60-120us
- Read
  - Controller pull bus low for 15us (checks after another 15us).
  - Device does nothing if it's a 1
  - If it's a 0 it pulls the bus low for another 45us

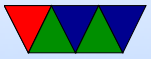


- Reset/Presence
  - Controller pulls bus low for 480us.
  - If a device is present it pulls bus low for 60us starting within 60us after the reset pulse.



Enumerating BUS (ROM commands)

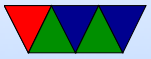
- How can you probe all 2^{64} possible addresses? `https://www.maximintegrated.com/en/app-notes/index.mvp/id/187`
- send a READ ROM request, returns 64-bit address. If multiple devices, then AND of all of them. (How do you detect this? Invalid CRC).
- SKIP ROM request sends command to all devices
- MATCH ROM request sends 64-bit address and only matched device responds



- SEARCH ROM –
  - Read first address bit from all devices on bus. Devices send their bit, followed by complement.

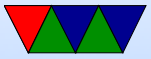
1	1	nothing there
0	1	all devices have 0 there
1	0	all devices have 1 there
0	0	conflict, you will have to probe both ways

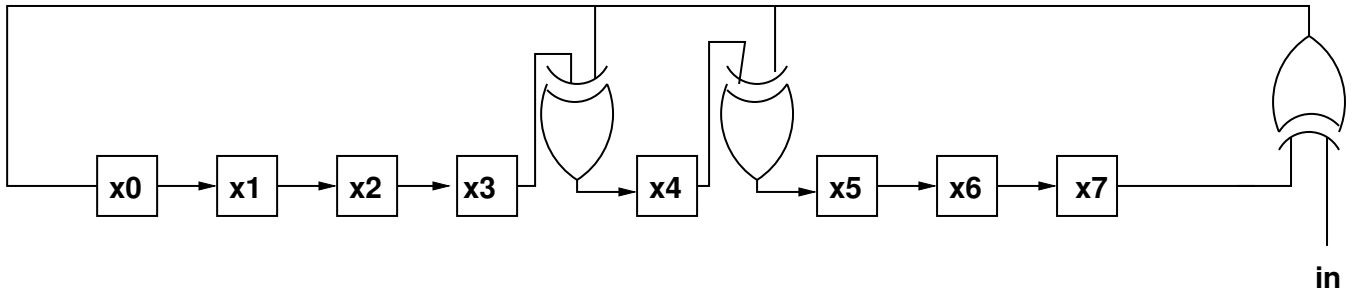
  - Then it does a binary search to enumerate all devices on bus.
  - Faster than probing all 2^{64} possible.

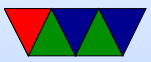


Cyclic Redundancy Check (CRC)

- Can detect wrong (flipped) bits in a transmission
- Fast to do in hardware, just a shift register and xor gates
- 1-wire CRC Can detect all double-bit errors, any overall odd number of errors, any cluster within an 8-bit window
- if CRCs with itself gets 0 at the end, how hardware detects correct address.
- $X^8 + X^5 + X^4 + X^1$
- Fill with zero, shift values in.



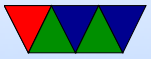

x0
x1
x2
x3
x4
x5
x6
x7
in



Hardware Interface

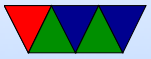
Possible ways to implement this:

- Use a GPIO and a pull-up resistor
- Use a serial UART. Needs extra circuitry to hook both TXD and RXD to bus
- USB/i2c/network connected
- Dedicated hardware?



Linux Interface

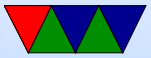
- "w1" driver merged in 3.6 kernel (a while ago now)
  - Driver for various interfaces, including bit-banging over GPIO (w1-gpio)
  - `/sys/bus/w1/devices/22-0000001d84f2/w1_slave`
  - read value and get ASCII dump of transaction
- OWFS – another driver, not in main kernel. Lets you export one-wire devices as a filesystem



One-Wire on Raspberry Pi

- The recommended way to enable things is to use `raspi-config`, interface options, 1-wire, enable on boot (you might need to reboot.

- Otherwise you can manually `sudo modprobe w1-gpio` and `sudo modprobe w1-therm` but on device tree systems (meaning, most recent Raspbian distributions) this might not be enough to make sure GPIO4 gets set up properly.

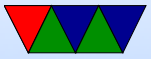


Temperature Interface w1_therm

- `cd /sys/bus/w1/devices/`
- `ls`
- `cd 28-000005aaf7ed` The serial number will differ (each unique)
- `cat w1-slave`

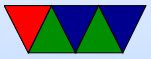
```
82 01 4b 46 7f ff 0e 10 70 : crc=70 YES
82 01 4b 46 7f ff 0e 10 70 t=24125
```

- Valid if the last value in first line is YES (passes CRC)
- second line has temperature in mili-degrees Celsius



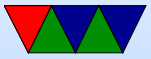
DS18B20

- -55 to 125C
- +/- 0.5C from -10 to 85C
- 9 to 12 bit resolution (configurable)
  Takes longer to convert more bits
- Converts temp in 93ms - 750ms
- Can set alarm (if go over a temp, a high bit set in result)
- small EEPROM can store alarm, config (number of bits)
  etc

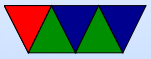


DS18B20 – Low Level Transaction

- Controller resets
- Controller listens for device to see it is present
- Controller sends MATCH ROM (0x55) then sends the 64-bit ID of the device it wants to talk to
- Controller sends a CONVERT T (0x44)
- Controller holds line high during conversion so the device has enough power to do the calculations
- Controller sends reset
- Controller listens for response



- Controller sends MATCH ROM (0x55) then the 64-bit ID
- Controller sends READ SCRATCHPAD (0xbe)
- Controller reads 8 bytes from device and CRC. If CRC wrong, tries again.

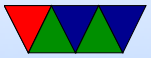


DS18B20 – Decoding the Data

- 9 bytes from device:

```
82 01 4b 46 7f ff 0e 10 70 :  crc=70 YES
```

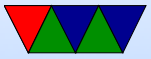
Byte 0/Byte1 = LSB/MSB Temperature = 0x0182
Byte 2 = TH register (high temp alarm, 8-bit)
Byte 3 = TL register (low temp alarm, 8-bit)
Byte 4 = config register 7f = 12 bit
Byte 5,6,7 reserved (5=0xff, 7=0x10)
Byte 8 = CRC



- Temperature is signed fixed point...

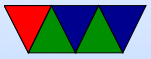
```
0x0182 = SSSS S654 3210 -1-2-3-4
         0000 0001 1000  0 0 1 0
```

$$2^4 + 2^3 + 2^{-3} = 16 + 8 + \frac{1}{8} = 24.125^oC = 75F$$

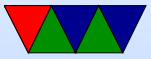


Less Convoluted Way of Getting the Temperature

- This seems like a lot of work just to get the temperature!
- Someone realized this and relatively recently added a "temperature" file to get the value w/o all the work
- Why not use "temperature" file?
- Had to look in linux source code, w1_therm.c
- git log / git bisect
- Added May 11 2020, first appeared Linux 5.8
- Not even the main feature, as an afterthought



- Better fits the sysfs philosophy
- Any reason not to use it? What if kernel version too old...
- For now lets still parse things the old way as you might run on a system that's too old to have this new version

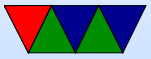


C string review

String manipulation is famously horrible in C. There are many ways to get the "YES" and "t=24125" values out of the text file. Any you choose is fine.

- If you trust the Linux kernel developers to keep a "stable ABI" then you can just read in the entire line into a string (array of chars) with `fgets()` and index into the string array to look for 'Y'

- Alternately you can use `fscanf()` to read the file. Again

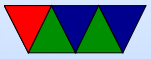


you have to trust the format won't change and that the YES always happens the same number of values into the file. One helpful hint, putting a '*' in a conversion (like `%*s` tells scanf to read in the value but ignore it.

- `sscanf()` is like `fscanf()` but reads from a string

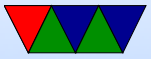
- Converting string to decimal or floating point
  `atoi()`, `atof()`, `strtod()`

- Comparing strings. Can you just use ==? NO!
  Be careful using `strcmp()` (or even better, `strncmp()`,



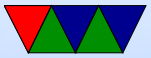
has unusual return value
less than, 0 or greater than depending. 0 means match

- If you had a string that was "V=YES" how could you start a string compare starting with the 2nd byte? Pointer math? Use `string+2` or even `&string[2]`.



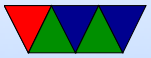
HW#8 – C string review

- String manipulation is famously horrible in C.
- There are many ways to get the "YES" and "t=24125" values out of the text file for HW#8.
- Any way you choose is fine.



C String Review

- This is tricky to get right
- It's relevant to Computer Security, the next topic we will cover

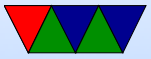


What is a C string? – essentially a hack

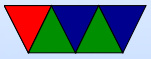
- A NUL (zero) (note: not NULL) terminated array
H	e	l	l	o	\0
- Note this is really:

0x48	0x65	0x6c	0x6c	0x6f	0x00
- Note in C, arrays are essentially just pointers
- Can statically declare: (compiler puts the 0 on end for you)

```
char string1[6]="Hello";
char string1[]="Hello";  // autosize
```

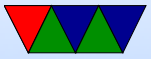


```
char *string2="Hello";
```



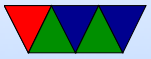
C String Review

- Many issues with array of bytes vs string, especially in other languages. Complicated if Unicode or UTF8. Windows / java and wchar (16-bit chars)
- You can use either pointer or array access to get a value (string[0] is the same as *string)
- Note that double quotes indicate a string, while single quotes indicate a single character



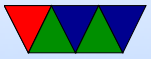
Upsides of C strings

- Fast and simple to deal with in assembly language
- Can quickly make short and cryptic functions to manipulate them
- ???



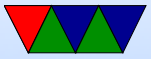
Downsides of C strings

- No way to tell the maximum size from the pointer
- Can only find out current size of string by iterating to find end
- The C library has a lot of helper functions, many of which are flawed in deep ways



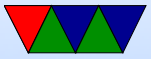
Other String Implementations

- Pascal-style strings, first byte is the length
  - Always know length, no need to strlen()
  - Maximum size (if 8-bit than max 256 chars)
- Higher level / object oriented languages (python, C++?) still have some sort of array of chars inside, but wrap it with extra info to provide safer access to things



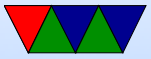
C string pitfalls – Writing off the End

- What happens when web form on your device's web interface asking "name" and you allocate 64 bytes but don't check, and someone types 4096 bytes
- What's the worst case?
- Crash your program?
- Corrupt data?
- Complete system compromise?



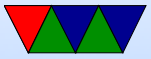
Can the C-library string functions save you?

- The standard strcpy(char *dst, char *src)
  - will happily go off the end if destination smaller than source
- strncpy(char *dst, char *src, int size)
  - added destination-size parameter, also pads dest with 0
  - NOTE: will leave off (!) the NUL terminator if not fit
- strlcpy(char *dst, char *src, int size)
  - always terminates destination

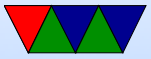


- if destination full, you lose a byte as it is silently truncated and last byte made NUL
- No error is indicated if this happens
- why a problem? example: say want to remove file.txt˜but got got truncated to file.txt instead?
- `https://lwn.net/Articles/507319/`

- Kernel has strscpy(char *dst, char *src, int size)
  - always returns valid string
  - returns a negative ERRNO on failure

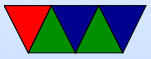


HW#8 Challenge – Reading from File



Method One – File I/O Using fscanf()

- The "stream" file interface in C lets you used buffered I/O and is slightly higher level than `open()`/`close()`
- Open a file with: `FILE *fff;`
  `fff=fopen("filename","r");`
  Check for errors! fff==NULL if it fails to open
- close a file with `fclose(fff);`
- you can read a string using `fscanf(fff,"%s",string);`

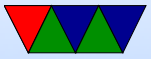


notes on scanf() functions

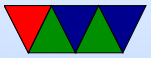
- printf() like interface

```
char string[256];
int x;
scanf("%d %s",&x,string);
```

  - Types to read like in printf, d for integer, s for string
  - Useful trick, %*s the asterisk means read but don't output, useful for skipping things
  - Result goes to a pointer. Note a string is already a pointer so no need for an ampersand

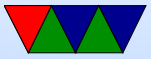


- scanf() reads from standard input (keyboard)
- fscanf() reads from file
- sscanf() reads from a string



Method Two – Read Entire File into RAM

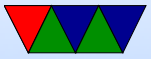
- There are multiple ways to read files into a string in C
  Assume `char string[1024];`
  - `fd=open("filename",RD_ONLY);`
    `result=read(fd,string,1023); close(fd);`
  - `FILE *fff; fff=fopen("filename","r");`
    `fread(buffer,size,count,fff); fclose(fff)`
- If you are treating things as a string, be sure to NUL-terminate `string[result]=0;`



Hardcoded sizes

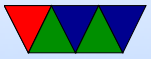
- In the last example I was being lazy and hardcoded a 1k size instead

  Can you make that dynamic?
- Use `stat()` to get filesize, then use `malloc()` to allocate space? Be sure to `free()` when done



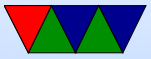
Other ways to access file contents

- Advanced: use `mmap()`
- You can also use `fgets(buffer,size,fff);` to bring in one line at a time
- What about `gets()`? Dropped from C libraries as being too unsafe! No size so just writes forever

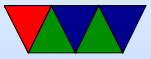


Finding a location / substring in a larger string

- If you trust the Linux kernel developers to keep a "stable ABI" you can assume the temperature will always be a fixed offset and hard code it. This can be a bit dangerous.
- You can use the `scanf()` series of functions to parse the string (either fscanf() directly, or sscanf() on the string) One helpful hint, putting a '*' in a conversion (like `%*s` tells scanf to read in the value but ignore it.

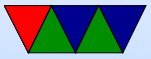


- You can use the `strstr()` search for substring C-library function to search for substrings, i.e. `strstr(string,"NO");` (haystack, needle)
- Maybe in conjunction with `strtok()`?
- You can manually parse the array.
  Using array syntax, something like:
  ```
  i=0; while(string[i]!=0) {
  if (string[i]=='t') break; i++ }
  ```
  Using pointer syntax, something like:
  ```
  char *a; a=string; while(*a!=0) {
  if (*a=='t') break; a++; }
  ```



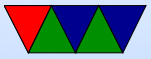
Pointing into a string

- If you searched for "t=" you might now have a pointer a to something like "t=12345". To point to 12345 you can just add 2 to the string pointer.
- `printf("%s\n",string+2);`
- `printf("%s\n",&string[2]);`



Converting string to decimal or floating point

- `atoi(char *string)` converts string to integer. What happens on error?
- `strtol()` will give you an error but is more complex to use
- `atof()` and `strtod()` will do floating point



Comparing strings

- Can you just use ==? NO!
- Be careful using `strcmp()` (or even better, `strncmp()`
  they have unusual return value
  less than, 0 or greater than depending. 0 means match
  So you want something like
  `if (!strcmp(a,b)) do_something();`