ECE471: Embedded Systems – Fall 2026

Homework 8: 1-wire bus

Due: Friday 6 November 2026, 5:00pm

1. You will need a DS18B20 1-wire temperature probe (looks like a transistor) and pull-up resistor (4.7K or 10K, either works) as handed out in class. If you missed class, you can stop by my office to pick these up.

   You can view the datasheet for the DS18B20:
   https://web.eece.maine.edu/~vweaver/classes/ece471_2014f/datasheets/DS18B20.pdf

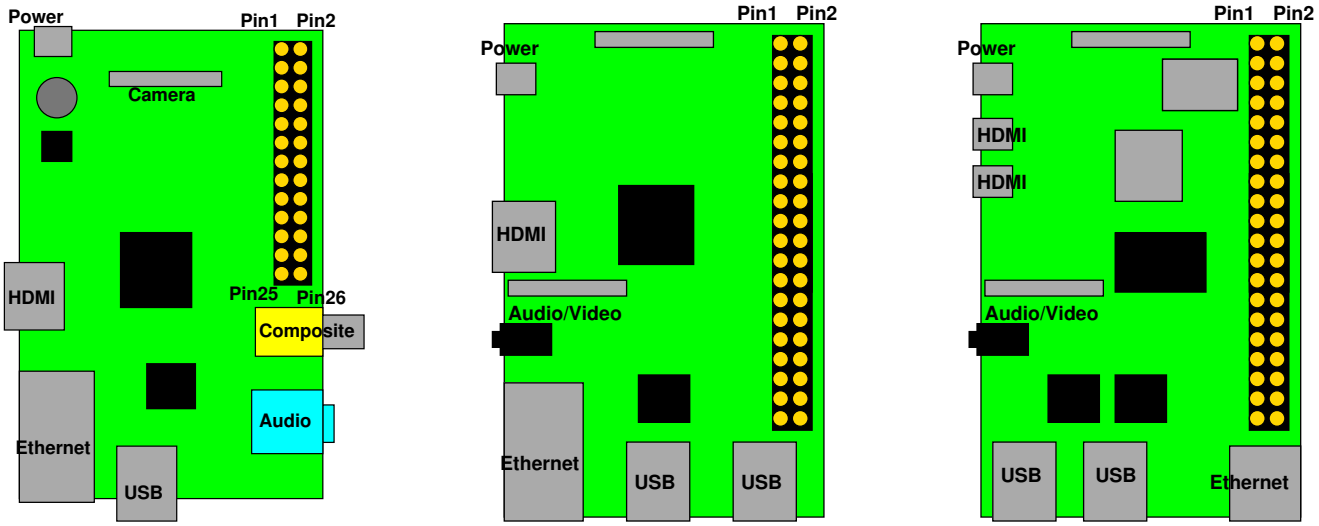


Figure 1: Location of header on Raspberry Pi Model B, Models B+/2/3, Model 4B/5B

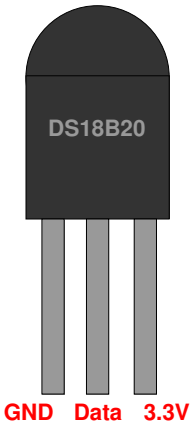


Figure 2: DS18B20 Pinout

Table 1: Raspberry Pi Header Pinout

3.3V	1	2	5V
GPIO2 (SDA)	3	4	5V
GPIO3 (SCL)	5	6	GND
GPIO4 (1-wire)	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

2. **Get the DS18B20 temperature readings over 1-wire** (6 points)

   First wire up the 1-wire device to the Raspberry Pi. You can use Figure 1 and Table 1 for guidance, as well as Figure 2.

   (a) Put the DS18B20 device on a breadboard:
   Connect `3.3V` on the Pi to `3.3V` on the DS18B20.
   Connect `GND` on the Pi to `GND` on the DS18B20.
   Connect `GPIO4` on the Pi to `Data` on the DS18B20.
   Connect `GPIO4` to 3.3V via the pullup resistor.

   (b) Enable 1-wire support in Linux running on your Pi.
   It should be possible to do this by running `raspi-config`, Interfacing Options, 1-wire, enable, and then rebooting.

   (c) If all went well, then if you run `lsmod | grep w1` it should show that both the `w1-gpio` and `w1_therm` modules (device drivers) are installed and running. (Note it's w-one the number, not w-lower-case-L).

(d) Modify the provided `temp_1wire.c` file to read values from the DS18B20 and print them to the screen. See the classnotes for more details.

i. Find the unique-id for your DS18B20 device.
Look for a directory under `/sys/bus/w1/devices/` that looks something like `28-000005aaf7ed`.

ii. Modify the provided `read_temp()` routine so it returns the temperature in degrees C as a floating point value.

A. Open the 1wire file, which looks something like this:
`/sys/bus/w1/devices/28-000005aaf7ed/w1-slave`
Please do not hard-code this device name directly into the call where you open the file. Instead, modify the `sensor_name.h` file, which is included by `temp_1wire.c`. Then when you open the file use `SENSOR_NAME` as the filename. This makes things a bit easier to grade.

B. Read the file. There are two values you want to grab, the `YES` value at the end of the first line and the `t=24125` value from the end of the second line.
One way to do this is to use `fscanf()` but you can use any method that works (there are many).
To use fscanf of a string, do something like

```
char string1[256],string2[256];
FILE *fff;
fff=fopen("filename","r");
fscanf(fff,"%s %s",string1,string2);
```

HINT: If you want to read a string but not actually store the value (in effect, skipping a value from the stream) you can do something like:

```
fscanf(fff,"%*s %s",string1);
```

The asterisk there means to read the value but not store it anywhere.

C. Read the YES/NO value from the end of the first line and check if the value is YES. If it isn't, print an error and return an invalid temperature value. (HINT, you can use `strcmp()` if you want, but be careful, it works backward from what you might expect). Next, read the t=XXXXX value. It is in miliCelsius, so first get the value (skipping the t= part; there are various ways to do this, remember that in C strings are just arrays of char, and arrays are just pointers).
Convert the integer miliCelsius value to floating point regular Celsius, and return.

iii. In an infinite loop, read the temperature value in C and print it to the screen once a second.

(e) If you encounter an error, print a message and exit the program.

(f) Be sure to close any files that you open when you are done with them.

(g) Always comment your code!

3. **Something Cool** (1 point)

   Copy your code to `temp_cool.c` and modify it to do something cool. Some example ideas:

   - (Easy) Print the temperature in a different unit (Fahrenheit, Kelvin, etc.)
   - (Medium) hook up an analog discovery board and plot a 1-wire transaction. How does it compare to theory? Send an image along with your assignment.
   - (Medium Hard) Automatically search the `/sys/bus/w1/devices/` directory for a sensor name rather than hard-coding it in.
   - (Hard) Make the code also read the TMP36 sensor from last homework, and compare how closely the temperatures match.
   - (Hard) Get the sensor working in parasite mode (without Vdd). Write up what you needed to do to get this to work. In some cases with the right value resistor this might just work. Sometimes it requires doing more setup, and maybe some kernel or device-tree parameters.

4. **Questions** (2 points)

   Answer the following in the README file:

   (a) One-wire in theory only uses 1-wire for data plus ground. In this lab we hooked up Vdd too. Why did we do that? (Hint, look up what's needed to use parasite power mode in the DS18B20 datasheet).

   (b) If you wanted to connect your pi to a temperature probe 100 yards (approximately 100 meters) away, and your choices were i2c, spi, or 1-wire, which bus would you use and why?

   (c) What is the **generic** name for the code/program responsible for loading the operating system kernel into memory and running it?

   (d) Why is the `/boot` partition on the Pi a FAT32 filesystem?

5. **Linux Fun** (1 point)

   On Linux you can create shell scripts, which are simple scripts that run shell commands.

   The first line contains the path to the program that you want to interpret the path. For example

   ```
   #!/bin/sh
   ```

   What follows are just a list of commands as you'd type at the prompt.

   The "echo" command is used to print text.

   ```
   echo "Hello World"
   ```

   Lines starting with a hash # are comment lines.

   (a) Edit the `sample.sh` file (the provided file is empty) to use the /bin/sh shell, and then print a message of your choice using `echo`
   Make the file executable by using the command `chmod +x sample.sh`
   Be sure to add comments to your script!
   Test your shell script by running `./sample.sh`

6. **Submitting your work**

- Run `make submit` which will create a `hw8_submit.tar.gz` file containing `Makefile`, `README`, `sample.sh`, `temp_cool.c` and `display_temp.c`.
  You can verify the contents with `tar -tzvf hw8_submit.tar.gz`
- e-mail the `hw8_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!