ECE 471 – Embedded Systems
Lecture 14

Vince Weaver

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

vincent.weaver@maine.edu

2 October 2020

Announcements

- Homework #4 was due.
- Homework #5 will be posted today

Announcements Pi Problems

- Some people have been having trouble with the Pi! Sorry about that, it was much easier to help with these issues before remote/distancing.
- If your Pi keeps shutting down/rebooting, and if you have a display and a little lightning bolt appears in the corner, this means a power issue. Try a different USB connector/charger, also make sure your wiring is right and you don't have anything shorted
- If you are getting GPIO `ioctl()` errors about the

interface being busy, make sure you don't have previous code running (common ways for this to happen are if you close an ssh window without killing the program first, or accidentally pressing control-Z and putting a running GPIO job in the background)

Homework Extensions

- If you are having issues keeping you from submitting homework on time, try submitting what you have and include a note saying why it's not done.
- You can turn in assignments late, you might lose a few points depending how long it's overdue, but you definitely will get some points (not a zero). Just try to turn in as soon as possible
- I don't like to give long extensions because that may cause you to get behind on next week's homework.

- Remember we drop your lowest homework grade.
- Don't stay up all night fighting a Raspberry Pi problem! Get some rest; turning it in a day or two late won't hurt your grade much.

i2c

- Inter-Integrated Circuit, Invented by Philips (now NXP) in 1982
- Broadcom and others for some reason call it TWI "Two Wire Interface"
- Two-wires (4 if you include Vdd and Ground)
- Since 2006, no licensing fees (though do have to pay to reserve number)

Why is i2c popular?

- Stable standard
- Relatively easy to implement
- Not many wires
- Good enough
- Cheap

Uses of i2c

- SMbus
- DDC (VGA/HDMI) (video card / monitor communication)
- Configuring SDRAM
- Temp sensor and fan chips on motherboards
- Wii nunchuck

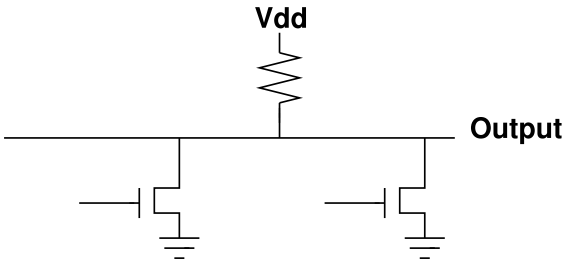
Protocol Overview

- Serial Data Line (SDA) and Serial Clock (SCL), Open Drain, Pulled up by resistors
- Open drain means output can be wired together

  If not driven, high-Z, line floats high

  If driven, pulls to zero

  Can have multiple connected to one line, "wired-and"

  (wired-nor)
- 7-bit (or 10-bit) address

- Speed: (actual transfers slower due to overhead)
  - Standard=100kbits/s
  - slow=10kbits/s
  - v1 1992 added fast=400kbits/s + 10-bit addr
  - v2 1998 High-speed 3.4Mbits/s w power saving
  - v3 2007 fast plus 1Mbits/s (20ma)
  - v4 2012 5MHz UFm (Ultra Fast mode), USDA, USCL, no pull-ups, unidirectional
  - v5, v6 no major changes
- Length of bus limited to a few meters (400pF)
  You can get extenders (LTC4311?)

Master/Slave Terminology

- Traditionally the main controller driving the bus was called the "Master" and the devices were called "Slaves"
- There has been a recent movement to use other terms for this
- I will use "controller" and "device" instead, but you will find that various specs, documents, and Linux interfaces use the old terminology

High-level Protocol

- Controller (generates clock, init transaction)
  Device (responds)
- Can be multiple masters and slaves
- Controller sends start bit, 7-bit device address, then read/write bit
- Device responds with ACK
- Reads and writes are 8 bits data, followed by 1 ACK bit
- Send stop bit when done
- Address and Data sent Most-significant Bit first

Low-level Protocol

- Busses start out floating high (by pull-up resistors)
- Start bit: SDA transition high-low while SCL high
- To transmit bit, master sets SCL low, then sets SDA to value, lets SCL float high, wait 4us, set SCL low for next cycle
- After every 8-bits other side sends ACK bit. The master toggles the clock then reads the SDA value.
  - If master reads 0, everything is OK
  - If writing, and read 1, means error or not there (why?)

  - ○ If reading, and read 1, means done reading
- Stop bit: SDA transition low-high while SCL high (only start/stop SDA transitions happen when SCL is high).

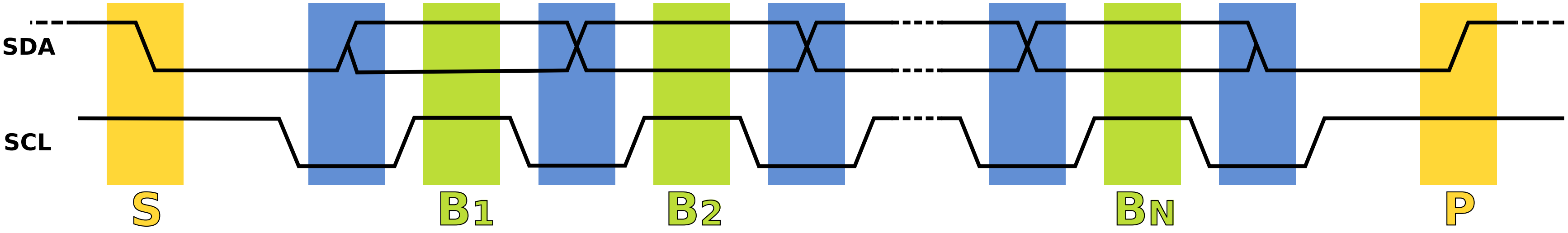


Figure 1: Protocol diagram from Wikipedia

Other Protocol Notes

- Clock stretching: device can hold SCL low until it is done processing, master should check to be sure SCL floated back high before continuing. (broken on pi before pi4)
- Arbitration: controllers monitor SDA and won't start unless idle. Deterministic arbitration.

  If tries to send a 1 and notices something else is pulling to zero, then a collision and stops. Low addresses automatically win.
- Repeated Start: can send multiple messages or to

multiple devices without sending stop but instead sending a new start bit

Message Types

- Controller writing **to** device:
  Sends start, address, write bit (0), waits for ACK (low), then sends 8 bits of data, waits for ACK, etc.

- Controller reading **from** device:
  Sends start, address, read bit (1), waits for ACK (low), then waits for 8 bits, sends ACK if wants more, otherwise stop if done.

i2c Reserved Addresses

Address	R/W Bit	Description
000 0000	0	General call address
000 0000	1	START byte (helps make polling cheaper)
000 0001	X	CBUS address
000 0010	X	Reserved for different bus format
000 0011	X	Reserved for future purposes
000 01XX	X	Hs-mode master code
111 10XX	X	10-bit slave addressing
111 11XX	X	Reserved for future purposes

10-bit addresses work by using special address above with first 2 bits + R/W, then sending an additional byte with the lower 8 bits.

SMbus

- Enhanced i2c bus interface
- Has stricter rules about some signals
- Can do more advanced things, such as have slaves send notifications to master

i2c and Rasp-pi

- 3.3V
- default speed is 100kHz. You can change this with the baudrate= module parameter.
- The Pi actually has multiple i2c busses, only one commonly used
  - i2c-1: The generic one on pins 3+5 (built-in pullups)
  - i2c-0: on Model B and newer one on camera interface
  - on Model 2B/3B one for “hat” EEPROM
  - on Model 3B/4B GPIO extender, driven by GPU?

Setting up i2c Rasp-pi Linux Driver

- These days the best way to do this is run `sudo raspi-config` and select (5) Interfacing Options, (P5) i2c, then say yes enable it. You might have to reboot
- In the old days you might have to manually set things up
  - `modprobe i2c-bcm2835` (in even older days this was called i2c-bcm2708) and `i2c-dev`
    May also want to edit `/etc/modules` and remove from blacklist `/etc/modprobe.d/raspi-blacklist.conf`

Other i2c Rasp-pi Linux Driver Notes

- May want to install i2c-tools if possible `apt-get i2c-tools`
- i2c port 1 (`/dev/i2c-1`). Used to be i2c-0 in really old days. Other boards (beaglebone black) likely different.
- Note that clock-stretching does not work on Pi before model 4.
- Note that repeated-start also might not be supported, though the driver might have a workaround. Use the `struct i2c_rdwr_ioctl_data` ioctl interface to send

the data

Linux i2c interface

- Like with GPIOs, kernel can drive it, or be exposed to userspace
- i2c-dev module must be installed (and i2c driver)
- Open the device node, `/dev/i2c-1`
- Use ioctls I2C_SLAVE to set the address of the device we wish to talk to.
- Use standard read or write calls to communicate with

the device

- Close the device when done.

- i2c device addresses are 7 bits, but when sent the r/w bit is put at end. This can be confusing; some spec sheets will list a slave address as 0xE0/0xE1 (8 bits, including r/w) but Linux exports this as 0x70 (0xE0 shifted right by 1).

Sample i2c Linux code

For more details on this, see the HW#5 handout.

```
unsigned char buffer[17];
int display_fd;

/* open */
display_fd = open("/dev/i2c-1", O_RDWR);
if (display_fd < 0) fprintf(stderr,"Error!\n");

/* set slave address */
result=ioctl(display_fd, I2C_SLAVE, 0x70);
if (result < 0) fprintf(stderr,"Error!\n");

/* writing */
buffer[0]= HT16K33_REGISTER_SYSTEM_SETUP | 0x01;
```

```
if ( (write(display_fd, buffer, 1)) !=1) {
        fprintf(stderr,"Error!\n");
}

/* closing */
close(display_fd);
```

i2c on the Pi – detecting

```
i2cdetect -y -r 1
```

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

LED Driver Chip

- This is a ht16k33, datasheet available:

  `http://www.adafruit.com/datasheets/ht16K33v110.pdf`

- Supports up to 16x8 LEDs, as well as keypad input. Can dim display, also blink. Common cathode.

  ```
      -|>|- common
  ```

- Works by rapidly scanning all segments fast enough cannot see.

- To set up, write byte commands, high 4 bits command lower 4 bits data.

- To set up full display, write the pointer offset of internal framebuffer, than 16 bytes of on/off data.

- Actual LED hooked up is a BL-Q56D-43UG 4x7 segment Ultra-Green display (or similar, colors vary), common cathode.

- How do you set address? (have more than one display hooked up?)

Benefit of OS

- Code is portable across all machines with i2c bus
- Can use same code on Gumstix, Rasp-Pi, Beaglebone, etc.
- Only change would be to update the bus number (It's i2c-3 on gumstix for example).