ECE 471 – Embedded Systems
Lecture 15

Vince Weaver

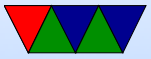
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

27 October 2015

Announcements

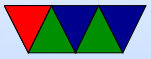
- HW7 will be posted today
- Hand out SPI hardware



HW6 Results

- Many people did not submit! Please try to even if you don't finish.

- Make sure you write ASCII '0' to GPIO to pull it low, not 0
  Also you need to point the write to something pointing to '0'. Can't just have write(fd,0,1);
  I was confused how the code worked despite never pulling SDA low, until I realized the sample code was printing all 1s to display.

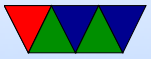


If you were checking errors, it would have reported EINVAL if you wrote non-ASCII 0 to the GPIO value.

- Why is it slow? What's the minimum time you can delay when using usleep()? Bitbanging in general?

- static in front of a function or global var means it only is called in this .c file.
  This is slightly different than the meaning of using static in front of a local variable (inside of a function).

- What's missing vs full protocol?
  Fidn't implement clock stretching: When stop bit, when

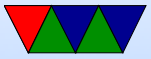


releasing SCL to go high, if it stays low the slave isn't ready to move on, should wait until it goes high.
Arbitration
Read byte
Paramaterized address

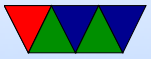
- Error checking. Be aware of errors.

- interrupt sources
  doorbell? more complex on pi2

- yes
  for answers a large list of questions in the affirmative?

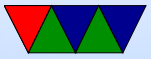


stress test? maybe



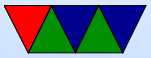
Midterm Results

- Not finished grading yet, hopefully by Thursday.



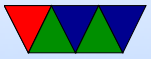
Project Preview

- Can work in groups
- Embedded system (any type, not just Pi)
- Written in any language (asm, C, python, C++, Java, etc.)
- Do some manner of input and some manner of output using the various capabilities we discussed
- I have a large amount of i2c, spi, and other devices that

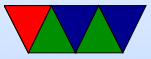


you can borrow if you want to try anything interesting.

- Past projects: games, robots, weather stations, motor controllers, music visualization, etc.
- Will be a final writeup, and then a 10 minute presentation and demo in front of the class during last week of classes.

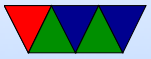


SPI review



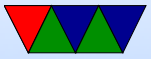
Errors

- No way to indicate errors
- Some chips will ignore if invalid data sent (wrong number of bits) some not

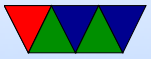


SPI advantages

- Full-duplex
- Fast (no set speed limit)
- Arbitrary message size in bits
- Low power (no pullup resistors)
- Can be implemented with minimal hardware (just a 74HC495 shift register)

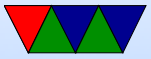


- No arbitration
- No unique ids
- Unidirectional signals
- Clock provided by master (no oscillator needed in slaves)



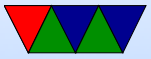
SPI disadvantages

- More pins (4 plus ground plus power plus one more each slave)
- Short distances
- No flow control
- No error reporting
- No standard

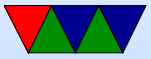


SPI vs i2c

- i2c benefits:
  - requires fewer wires
  - shared bus (no need for lots of chip select)
  - nack when data received
  - can have multiple masters
  - less susceptible to noise
  - can transmit longer distances
  - has a formal standard

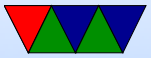


- spi benefits:
  - lower power
  - potentially faster, full-duplex
  - i2c can be brought down by one bad device



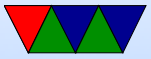
SPI bus on Raspberry Pi

- SPI1 is on the header
- Pin 23 – SCLK
- Pin 19 – MOSI
- Pin 21 – MISO
- Pin 24 – CE0
- Pin 26 – CE1
- Unlike some boards, no nIRQ (SPI interrupt) pin



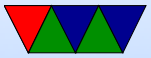
SPI bus on Linux

- `modprobe spidev`
- `modprobe spi-bcm2835`
  on older kernels, `modprobe spi-bcm2708`
- `dmesg | grep spi`



SPI dev interface

- `https://www.kernel.org/doc/Documentation/spi/spidev`
- `/dev/spidevB.C` (B=bus, C=slave number).
  On pi it is `/dev/spidev0.0`
- Other useful info in `/sys/devices/.../spiB.C`, `/sys/class/spidev/spidevB.C`
- To open the device, do something like the following
  `spi_fd=open("/dev/spidev0.0",O_RDWR);`



- To set the write mode, use ioctl:

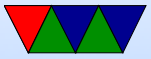
```
int mode=SPI_MODE_0;
result = ioctl(spi_fd, SPI_IOC_WR_MODE, &mode);
```

  Modes can be SPI_MODE_0 through 3, or else you can build them out of SPI_CPOL and SPI_CPHA values.
  Current mode can be read back with SPI_IOC_RD_MODE

- To set the bit order, use ioctl:

```
int lsb_mode=0;
result = ioctl(spi_fd, SPI_IOC_WR_LSB_FIRST, &lsb_mode);
```

  Current can be read with SPI_IOC_RD_LSB_FIRST
  Get/Set if MSB is first (common) or LSB is first.
  Empty bits padded to left with zeros no matter what the



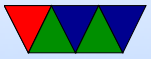
setting.

- `SPI_IOC_RD_BITS_PER_WORD`, `SPI_IOC_WR_BITS_PER_WOR`
  Number of bits in each transfer word. Default (0) is 8 bits.

- `SPI_IOC_RD_MAX_SPEED_HZ`, `SPI_IOC_WR_MAX_SPEED_HZ`
  Set the maximum clock speed.

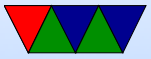
- By default using `read()` or `write()` on the device node will only do half-duplex.

- For full duplex support you need something like the



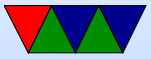
following:

```
#define LENGTH 3
int result;
struct spi_ioc_transfer spi;
unsigned char data_out[LENGTH]={0x1,0x2,0x3};
unsigned char data_in[LENGTH];

/* kernel doesn't like it if stray values, even in padding */
memset(&spi,0,sizeof(struct spi_ioc_transfer));

/* Setup full-duplex transfer of 3 bytes */
spi.tx_buf    = (unsigned long)&data_out;
spi.rx_buf    = (unsigned long)&data_in;
spi.len       = LENGTH;
spi.delay_usecs       = 0 ;
spi.speed_hz          = 100000 ;
spi.bits_per_word     = 8 ;
spi.cs_change         = 0 ;

/* Run one full-duplex transaction */
result = ioctl(spi_fd, SPI_IOC_MESSAGE(1), &spi) ;
```



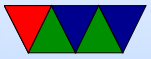
Analog Digital Converters on Raspberry PI

- Unlike many other embedded boards, the Pi has no A/D converters built in.
- You're stuck using SPI or i2c devices

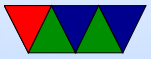


MCP3008

- For HW#7 we'll use the MCP3008 8-port 12-bit SPI A/D converter
- up to 100ksp (samples per second)
- Returns 10-bits of accuracy
- 8 single-ended inputs (vs ground) or 4 "pseudo-differential" inputs (vs each other)
- Config sent in each request packet

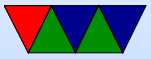


- Clock frequency must be long enough that the A/D has time to convert
- $V_{IN} = \frac{value \times V_{REF}}{1024}$



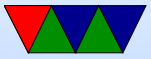
MCP3008 μcontroller mode

- Datasheet describes way to easily use from a device
- Send 3 bytes. First has value '1' (the start bit). The second has the top 4 bits being single/diff followed by 3 bits of which channel you want. The rest is all 0s for padding.
- You read back 3 bytes. First 13 bits are don't care (ignore) followed by 0 then the 10 bits of sample.
- XXXXXXXX XXXXX098 76543210

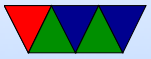


TMP36

- Linear temperature sensor
- The temperature can be determined with the following equation:
  $deg\_C = (100 \times voltage) - 50$
- Also the following might be useful:
  $deg\_F = (deg\_C \times \frac{9}{5}) + 32$
- Be careful hooking up! If vdd/gnd switched it heats up to scalding temperatures (the datasheet lists the pinout



from the bottom). If you catch it in time doesn't seem to be permanently damaged.



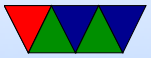
Floating Point in C

- Converting int to floating point:

```
int value=45;
double temp;

temp=value;      // works
temp=(float)value;  // casts make the conversion explicit
            // but can potentially hide bugs
```

- float vs double
  float is 32-bit, double 64-bit

- Constants 9/5 vs 9.0/5.0



The first is an integer so just “1”. Second is expected 1.8.

- Printing. First prints a double. Second prints a double with only 2 digits after decimal.

```
printf("%lf\n",temp);
printf("%.2lf\n",temp);
```