ECE 471 – Embedded Systems
Lecture 20

Vince Weaver

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

`vincent.weaver@maine.edu`

20 October 2017

Announcements

- Project coming
- Only one person was in class Wednesday due to Career Fair
- HW7 will be posted today
- Hand out SPI hardware
- Hold on to your LED displays (and temperature measuring probes) until after Homework #9

SPI bus

- Serial Peripheral Interface Bus
- Synchronous full-duplex serial bus named/formalized by Motorola. No real standard.
- What does synchronous mean? (Separate clock line)
- What does full-duplex mean? (Transmit and receive at same time)

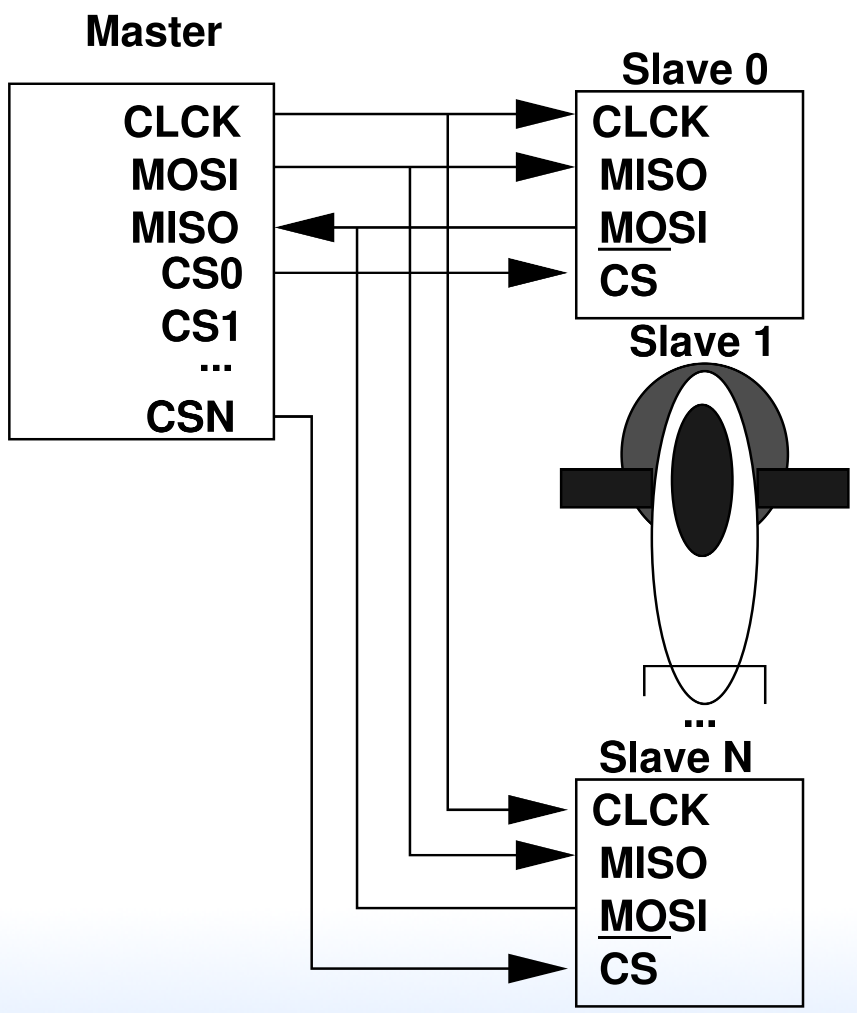
What used for?

- LCD displays
- Optional interface to SD cards
- LED strips
- JTAG

Hardware Setup

- Master/slave with multiple slave select lines
- 4-wire bus
- SCLK – serial clock (output from master)
- MOSI – master out, slave in
- MISO – master in, slave out
  Must be high impedance if more than one slave

- CS0, CS1, etc – slave chip selects



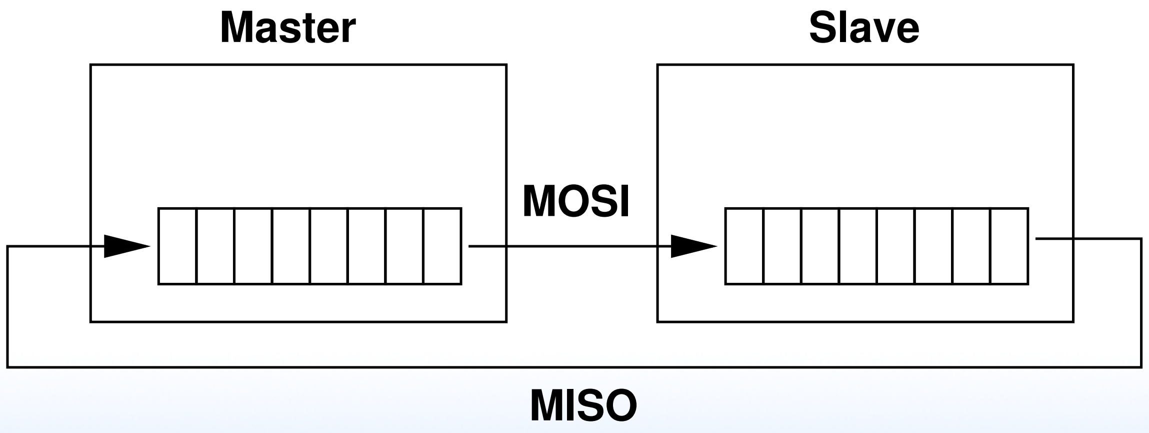
SPI protocol

- Pulls chip-select of desired slave low
- Master starts clock
  No set speed, just what the slave can handle.
  Up to a few MHz (Pi in theory 128MHz, 16MHz more realistic)
- Must both Send *and* receive (at same time over MISO/MOSI wires)
  Doesn't have to be useful data, but must be done both

ways

- Master transmits data bits as long as it has it. When done turns off clock and maybe deselects slave.

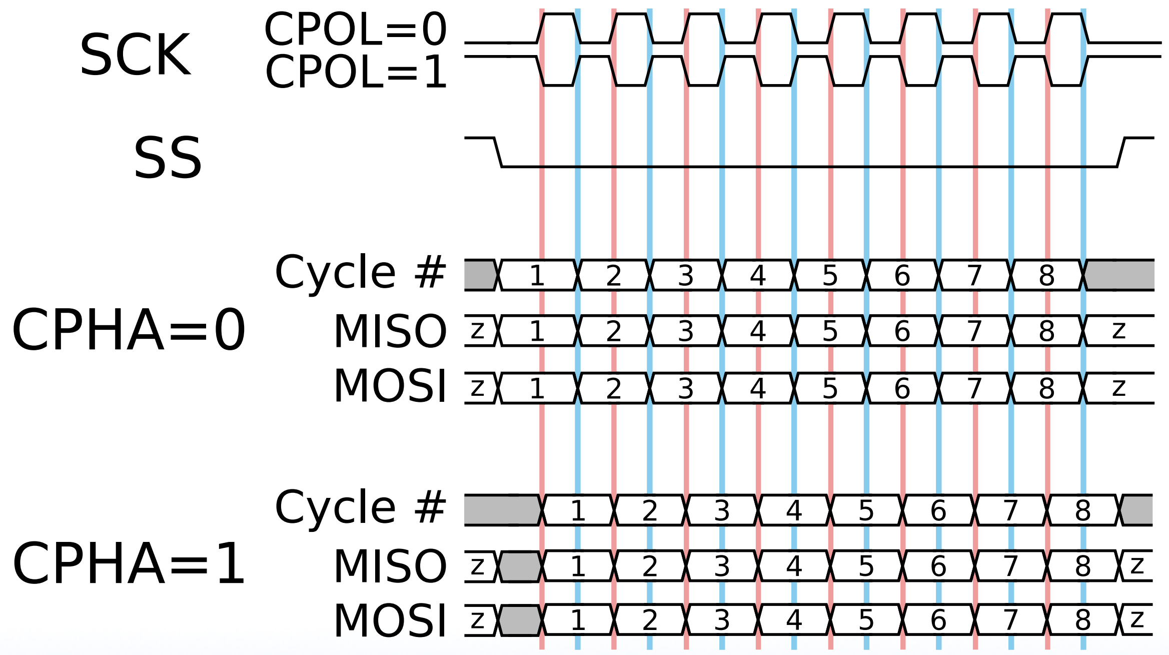
- It's basically just a shift register in the master and slave, and you rotate through enough bits to swap the values in each, then both sides can read out the transfer.



Clock Polarity/Phase

- Many have adopted Freescale's terminology
- CPOL=0 – base clock is zero
  - CPHA=0 – data captured on rising edge
  - CPHA=1 – data captured on falling edge
- CPOL=1 – base clock is one
  - CPHA=0 – data captured on falling edge
  - CPHA=1 – data captured on rising edge

- Also given as “mode” numbers, 0 - 3. CPOL/CPHA. This can vary by manufacturer. Check your data sheet!
- Timing diagram from Wikipedia (CC BY-SA 3.0)



Connection

- “independent” – One slave per select line
- “daisy-chain” – MISO to MOSI, like long chain of shift registers, only need one SS line.

Interrupts

- Possible... think touch screens and such. Not officially specified

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 (10 feet or so?)
- 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

- On recent Pis, SPI is enabled through devicetree. You can run `sudo raspi-config`, select advanced, then SPI, then say yes to enable and at boot.
- On older systems you might have to do this manually by `modprobe spi-bcm2835`; even older kernels it has a different name: `modprobe spi-bcm2708`
- `dmesg | grep spi` will show useful debug
- To get the user interface `modprobe spidev`

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 10-bit SPI A/D converter
- up to 100ksp (samples per second)
- 2.7 to 5.5V
- 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
- $V_{IN} = \frac{value \times V_{REF}}{1024}$
  Yes, this seems wrong (can never have full V_{REF} output) but this is what the data sheet says to use

MCP3008 μcontroller mode

- Datasheet describes way to easily use from a device
- Send 3 bytes. First has value ‘1’ (start bit). The second has the top 4 bits being single/diff followed by 3 bits of channel you want. The rest is all 0s for padding.
- 00000001 SCCC0000 00000000
- 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:

```c
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);
```