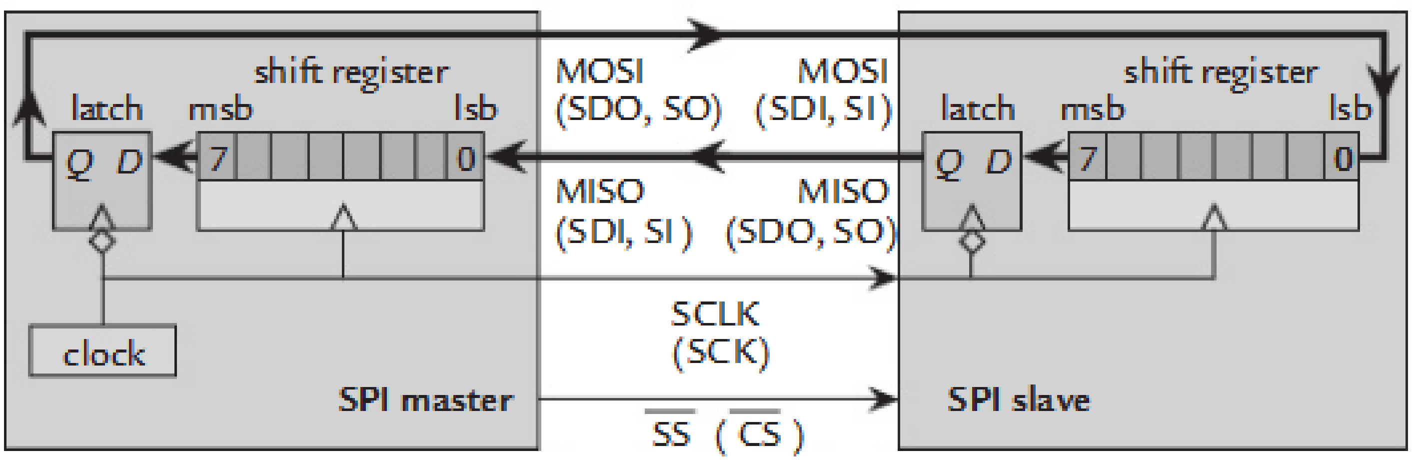
Serial Peripheral Interface

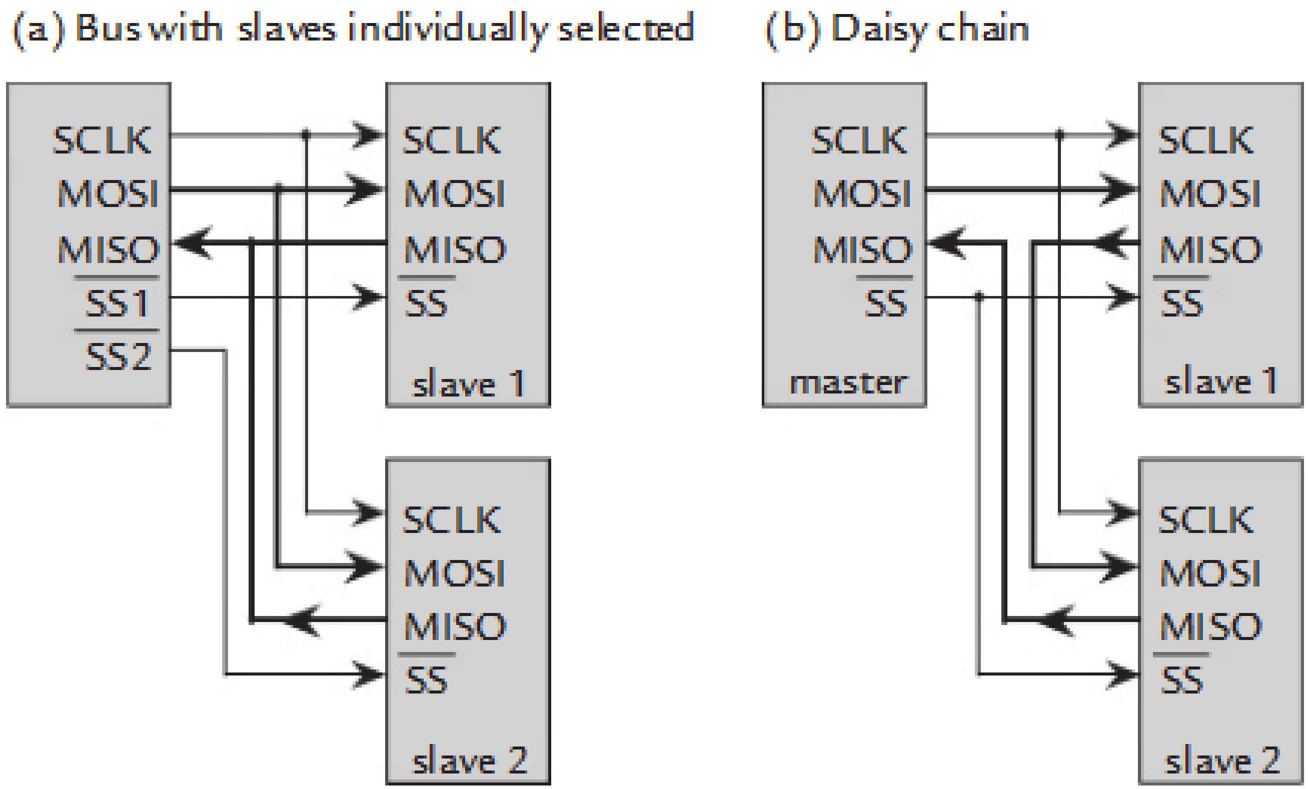
- Common serial interface on many microcontrollers
- Simple 8-bit exchange between two devices
    - Master initiates transfer and generates clock signal
    - Slave device selected by master
- One-byte at a time transfer
    - Data protocols are defined by application
    - Must be in agreement across devices

SPI Block Diagram

- 8-bits transferred in each direction every time
- Master generates clock
- MOSI: “Master Out Slave In”; MISO: “Master In Slave Out”
  - Connect MOSI to MOSI and MISO to MISO
  - Very clean terminology, unlike “TX” and “RX” which are easy to confuse
- Slave Select (SS) used to select one of many slaves
- Terminology varies:
  - Instead of SS, “Chip Select” (CS)
  - Instead of MOSI and MISO, SIMOD and SOMI



Hack: using SPI as a bus



Configuration details to watch out for

- CPHA (Clock PHase) aka ~CKPH (MSP430 terminology)
  - = 0 or =1, determines when data goes on bus relative to clock
- CPOL (Clock POLarity) aka CKPL (MSP430)
  - =0➔ clock idles low between transfers
  - =1➔ clock idles high between transfers

This leads to 4 SPI clock modes

Mode	CPOL/CKPL	CPHA/ ~CKPH
0	0	0
1	0	1
2	1	0
3	1	1

NB: on this slide,
~ means negation,
i.e. same as overbar

Takeaway message: make sure master and slave are configured the same way!

SPI properties

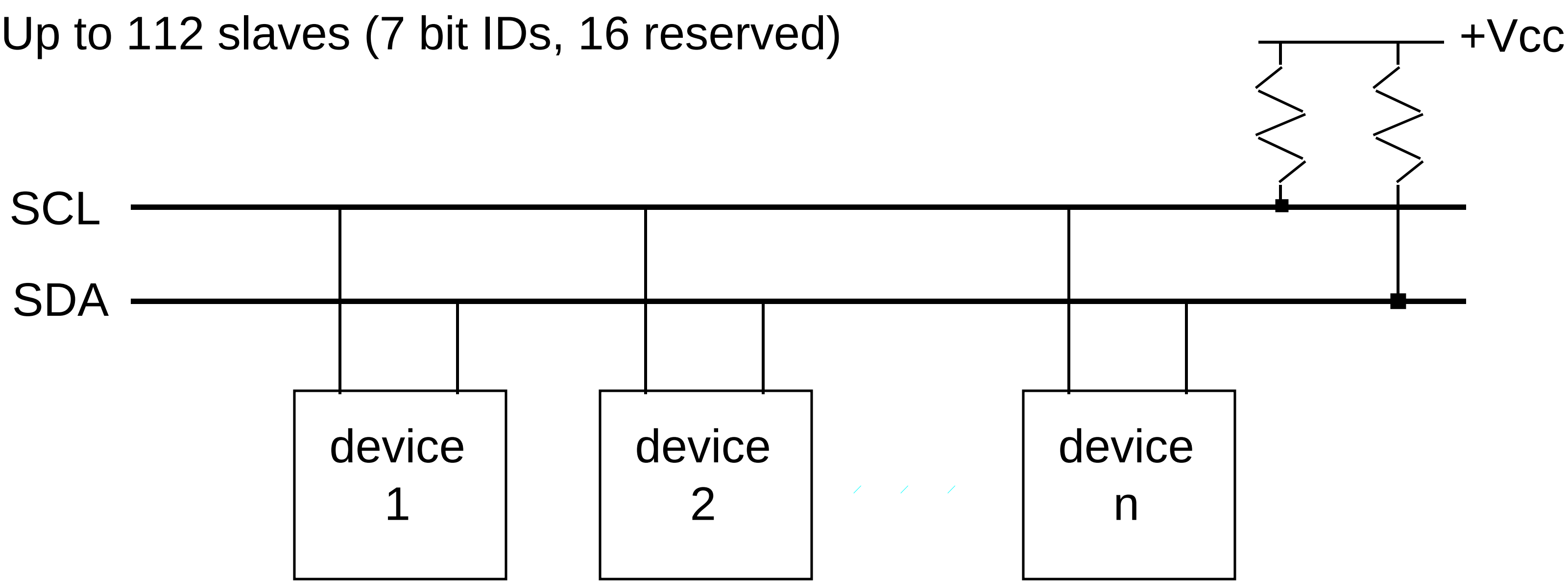
- Pros
  - Simplest way to connect 1 peripheral to a micro
  - Fast (10s of Mbits/s, not on MSP) because all lines actively driven, unlike I2C
  - Clock does not need to be precise
  - Nice for connecting 1 slave
- Cons
  - No built-in acknowledgement of data
  - Not very good for multiple slaves
  - Requires 4 wires
    - 3 wire variants exist…some get rid of full duplex and share a data line, some get rid of slave select

Inter-Integrated Circuit Bus (I2C)

- Supports data transfers
  - 10 kbit / s slow mode
  - 100 kbit / s standard mode
  - 400 kbit / s fast mode
  - 1 Mbit /s fast mode plus
  - 3.4Mbit / s high speed mode
- Philips (and others) provide many devices
  - microcontrollers with built-in interface
  - A/D and D/A converters
  - parallel I/O ports
  - memory modules
  - LCD drivers
  - real-time clock/calendars
  - DTMF decoders
  - frequency synthesizers
  - video/audio processors

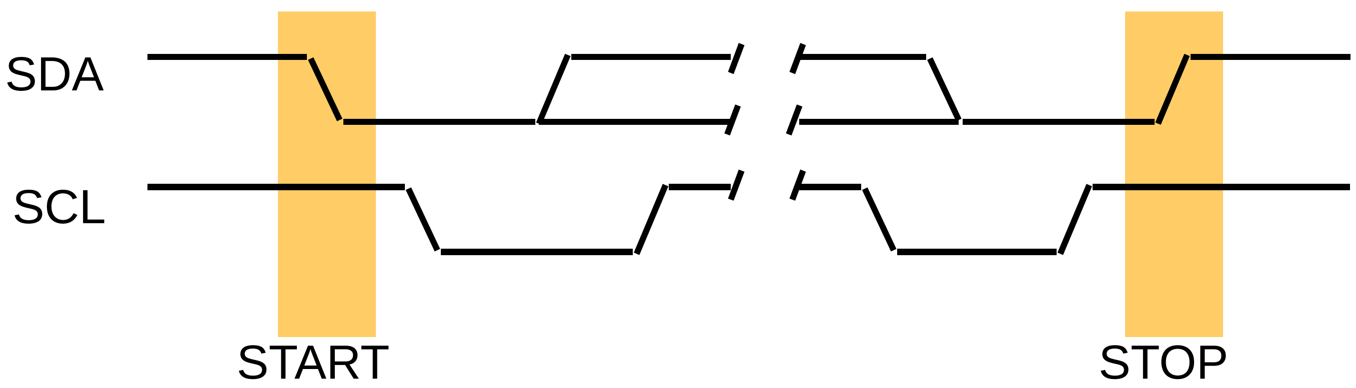
Inter-Integrated Circuit Bus (I2C)

- Modular connections on a printed circuit board
- Multi-point connections (needs addressing)
- Synchronous transfer (but adapts to slowest device)
- Similar to TWI (Two-Wire Interface) on Atmegas
- Pull up resistors pull lines high
- Devices on bus go from “high impedance” to ground to generate a low on bus
- 1 Master (generates clock, initiates communication)
- Up to 112 slaves (7 bit IDs, 16 reserved)

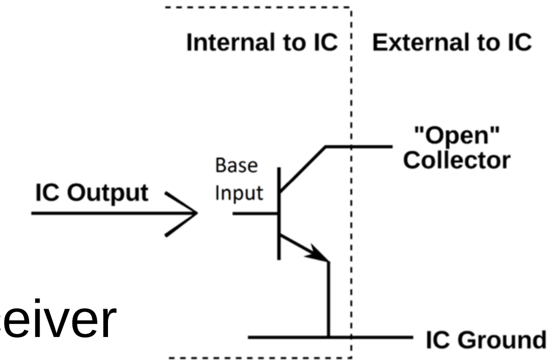


Serial data format

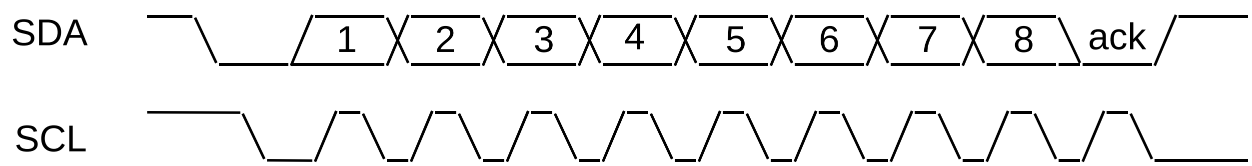
- SDA going low while SCL high signals start of data
- SDA going high while SCL high signals end of data
- SDA can change when SCL low
- SCL high (after start and before end) signals that a data bit can be read



Byte transfer



- Byte followed by a 1 bit acknowledge from receiver
- Open-collector (open drain) wires
  - sender allows SDA to rise
  - receiver pulls low to acknowledge after 8 bits

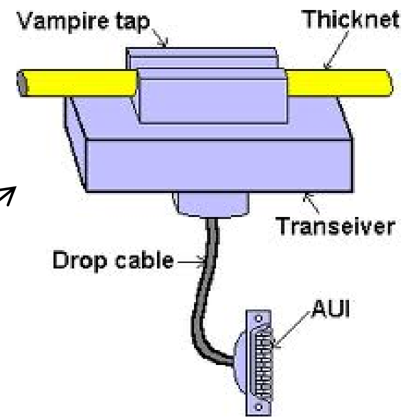


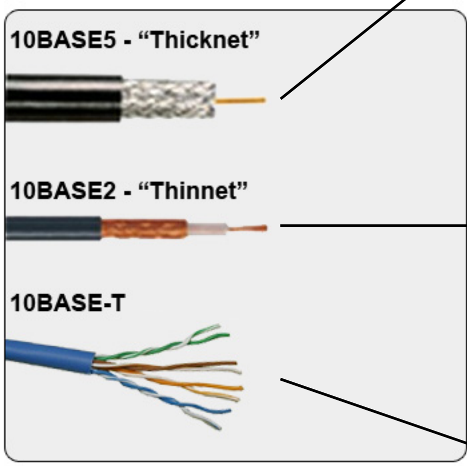
- Multi-byte transfers
  - first byte contains address of receiver
  - all devices check address to determine if following data is for them
  - second byte usually contains address of sender

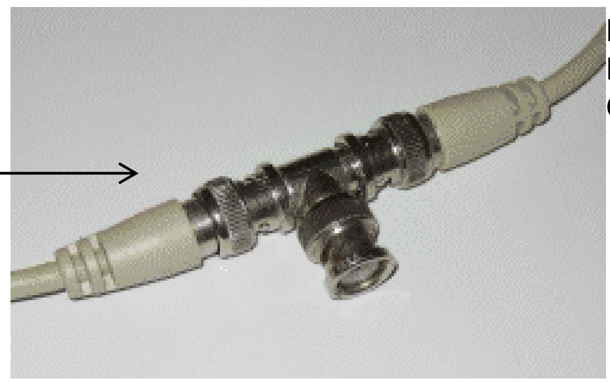
Ethernet

- Inspired by early wireless network: “Aloha” network from U. Hawaii
- Local area network
  - “Classic”: 10Mbps serially on shielded co-axial cable
  - “Switched”: 100Mbps, 1000Mbps, 10,000Mbps
- Developed by Xerox in late 70s
  - still most common LAN
- High-level protocols to ensure reliable data transmission
- CSMA-CD: carrier sense multiple access with collision detection

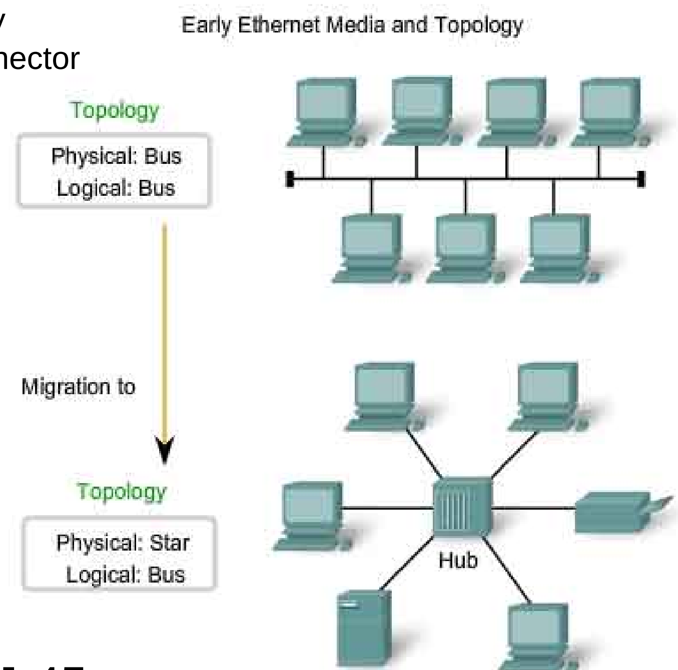
Ethernet



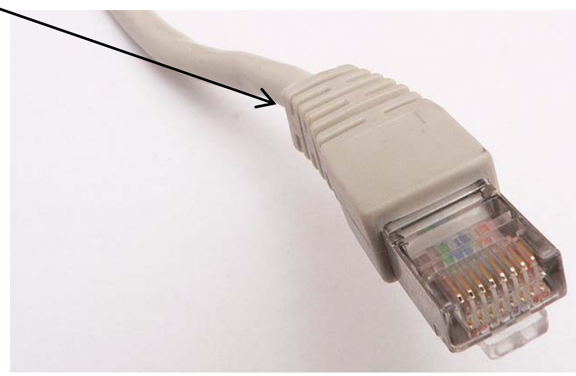




BNC
Bayonet
Navy
Connector



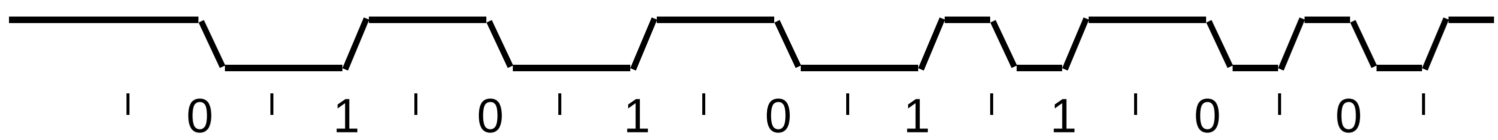
10: 10 Mbps
100: 100 Mbps
BASE: Baseband signaling
(not modulated RF, like the 100s MHz signals on your TV cable)
5: 500m max range
2: 200m max range
T: Twisted pair



RJ-45

Serial data format

- Manchester encoding
  - signal and clock on one wire (XORed together)
  - "0" = low-going transition
  - "1" = high-going transition

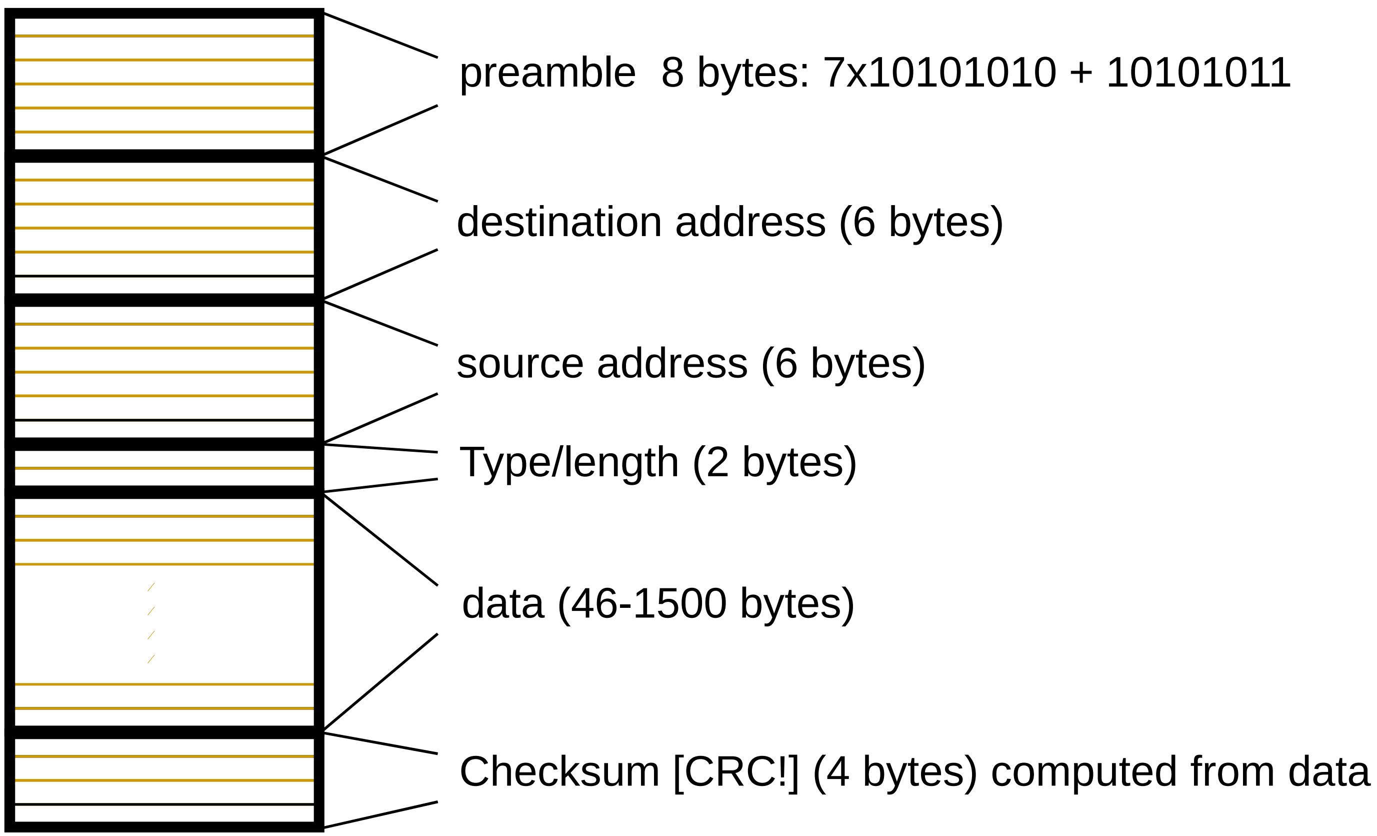


- preamble at beginning of data packet contains alternating 1s and 0s
- ➔10MHz square wave for 6.4us….allows rcv to synch clock to tx
- preamble is 64 bits long: 10101. . . 01011

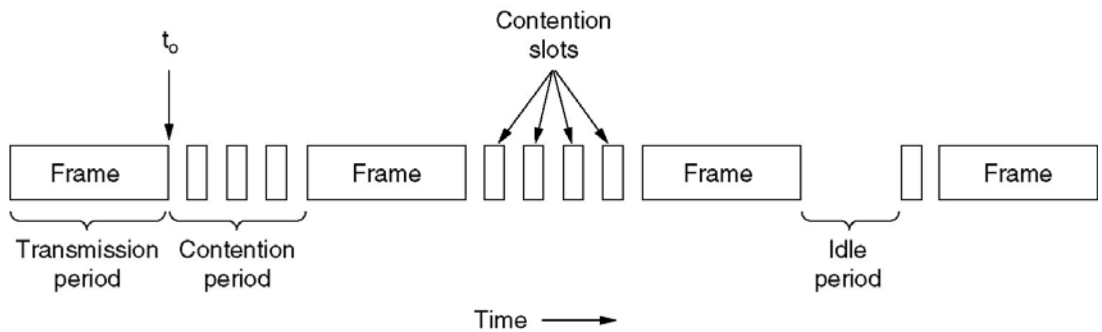
Extra 1 signals Start Of Frame (SOF)

Ethernet packet

- Packet size: 64 bytes (min!) to 1518 bytes + 8 bytes of preamble



Arbitration



- CSMA/CD
  - "Carrier Sense Multiple Access with Collision Detection"
- Wait for line to be quiet for a while then transmit
  - detect collision
  - average value on wire should be exactly between 1 and 0
  - if not, then two transmitters are trying to transmit data
- If collision, stop transmitting
  - wait a random amount of time and try again
  - if collide again, pick a random number from a larger range (2x) and try again
- Exponential backoff on collision detection
  - "Random exponential backoff" or "binary exponential backoff"
  - Key innovation in Ethernet…Bob Metcalf's thesis at Harvard
- Try up to 16 times before reporting failure

EtherCAT

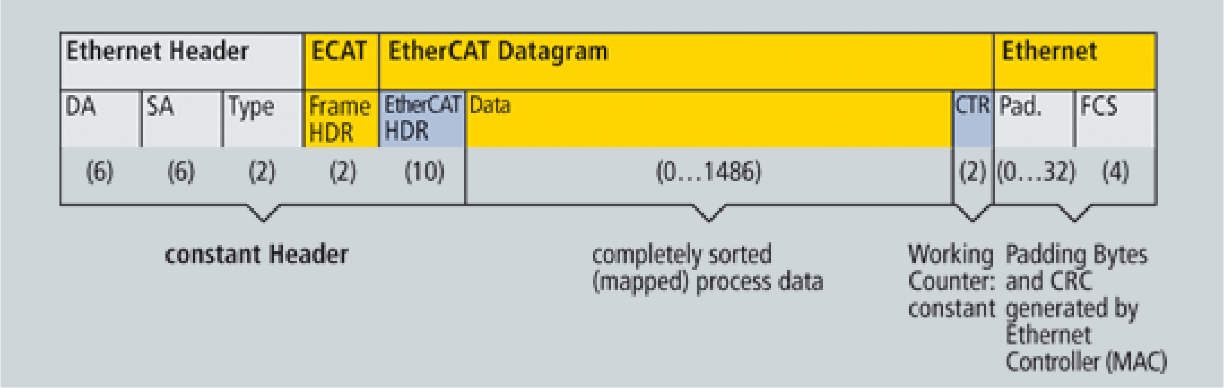
- “Ethernet for Control Automation Technology”
- Ethernet with
  - Short update times (cycle times)
  - Low communication jitter
  - Low hardware costs
- Not currently very well-known, but
  - Used in Willow Garage PR2 robot!

EtherCAT based!

How it works

- Master/Slave, Master/Master, and Slave/Slave (via Master) supported
- Master side: conventional Ethernet MAC HW
  - i.e., plug an EtherCAT network into the back of your laptop!
  - Need alternate driver SW
- Slave side: custom hardware
  - Ethernet packets ingested & regenerated by slaves
    - (This would not be possible in classic bus-style Ethernet)
  - Slave can extract or insert “EtherCAT datagrams” into the data portion of the Ethernet packet
  - Slave has to replace Ethernet CRC if it adds an EtherCAT datagram to the Ethernet Frame

Master:



How it works

- To master, it looks like there is just one Ethernet device out there (even if there are multiple EtherCAT slaves)
- In each slave, processing is done by dedicated hardware
  - Ensures fast, real-time behavior
  - “Telegrams” processed directly “on the fly”
- Many EtherCAT datagrams fit in a single Ethernet packet
- Many devices can be addresses in a single EtherCAT datagram
- Frame overhead is amortized over many messages, improving net efficiency

Slaves:

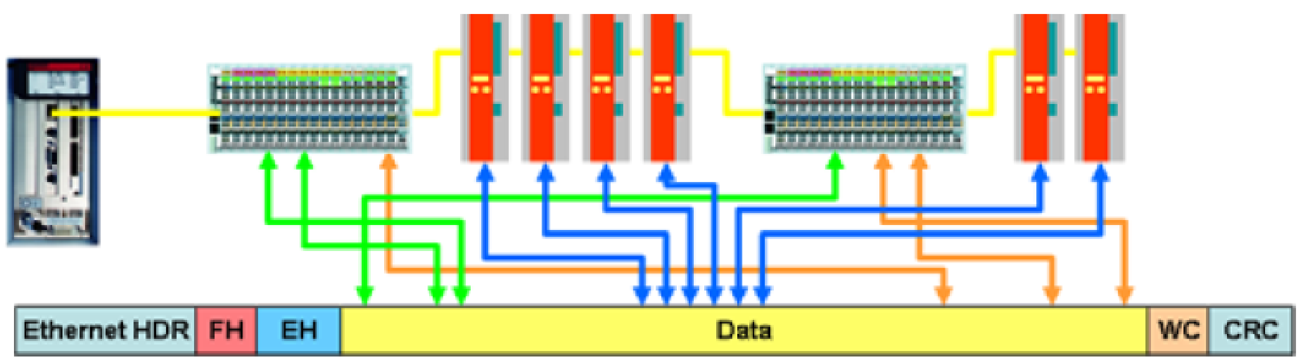
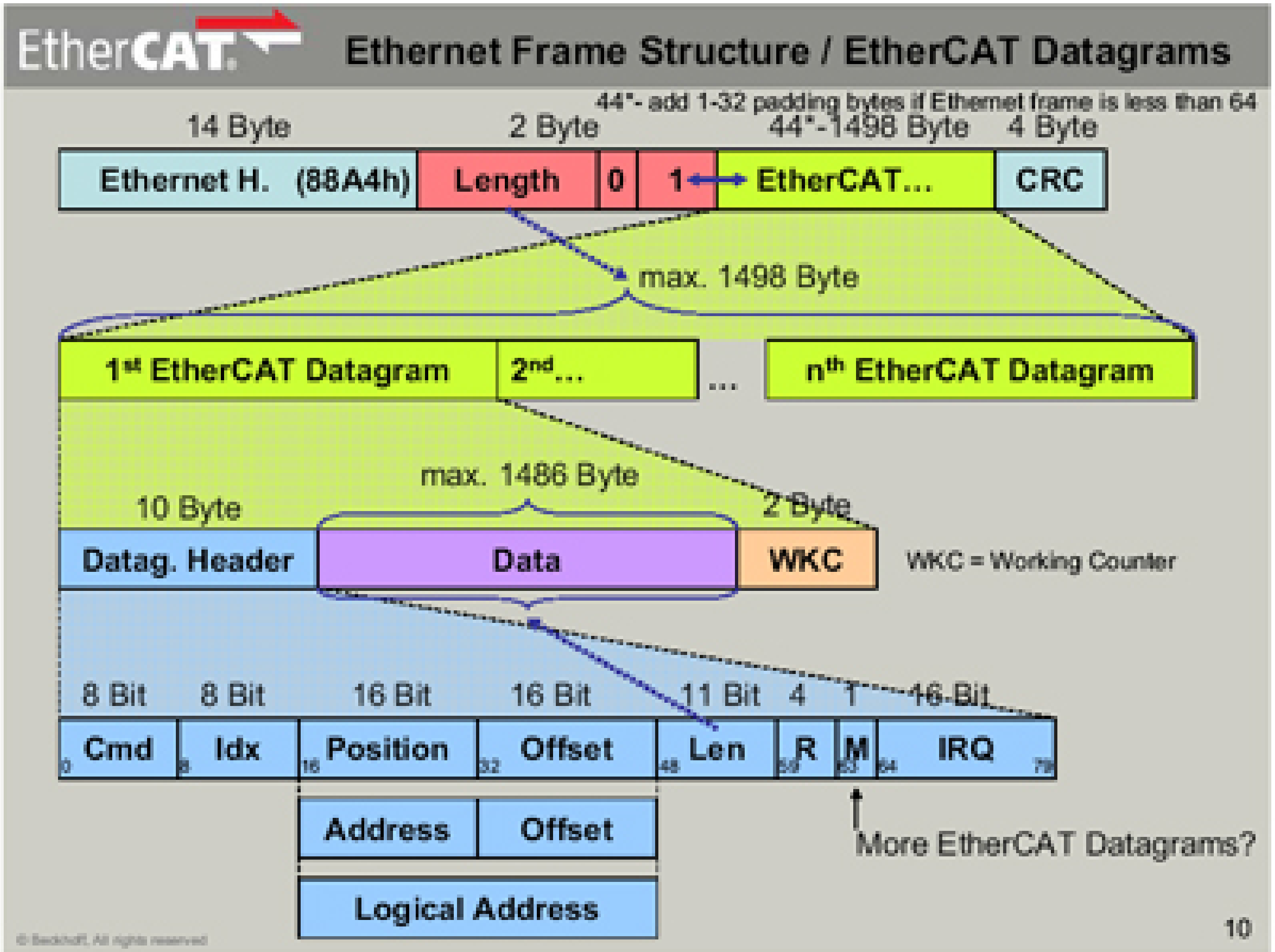


Figure 4: Devices map data directly in frame

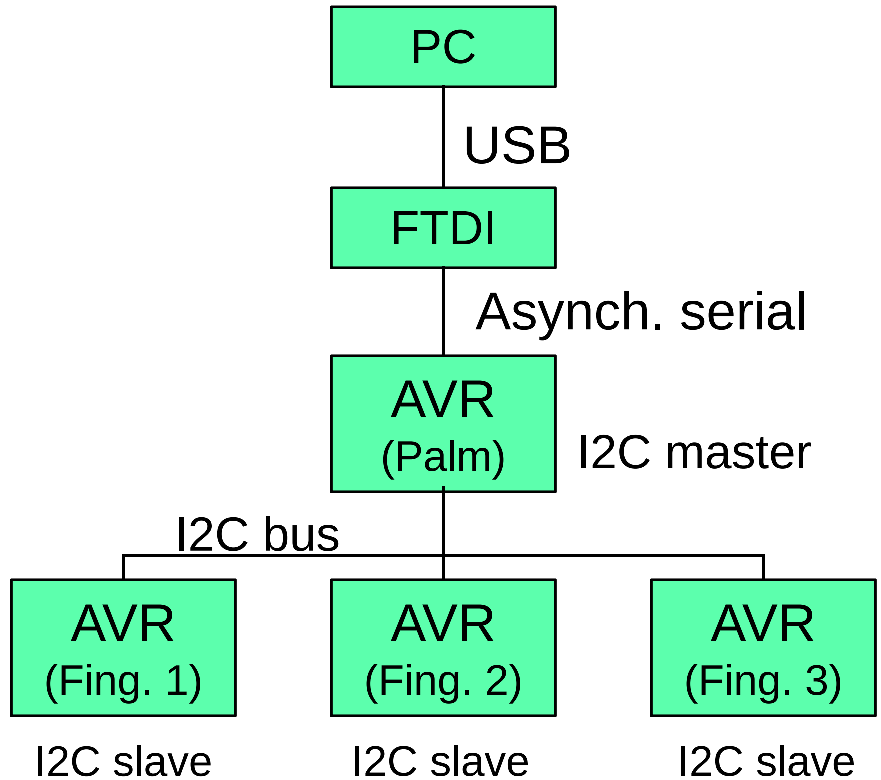
EtherCAT Datagrams



Research examples using these comms schemes---Pretouch sensing

- We will see E-Field Sensing, USB Virtual COM, I2C, SPI, and EtherCAT in action

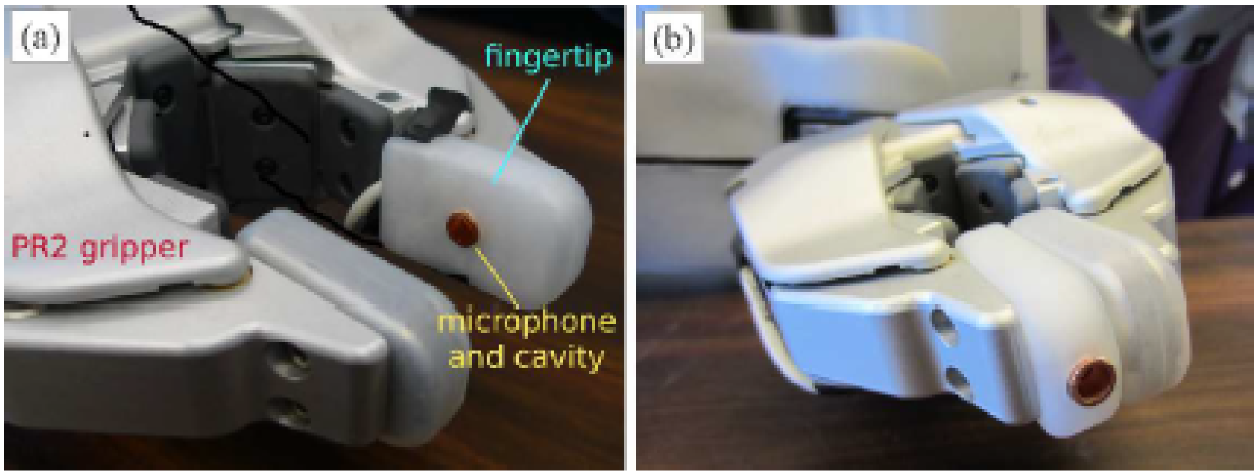
Electric Field Pretouch



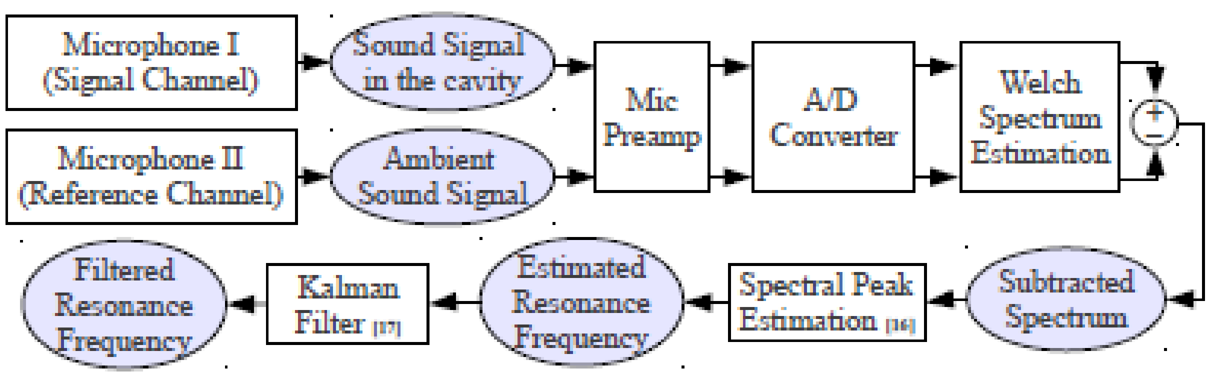
Same sensing technique you are using in lab

An Electric Field Pretouch System for Grasping and Co-Manipulation,
ICRA-2010.
B. Mayton, L. LeGrand, J.R. Smith

Sensor Design on PR2



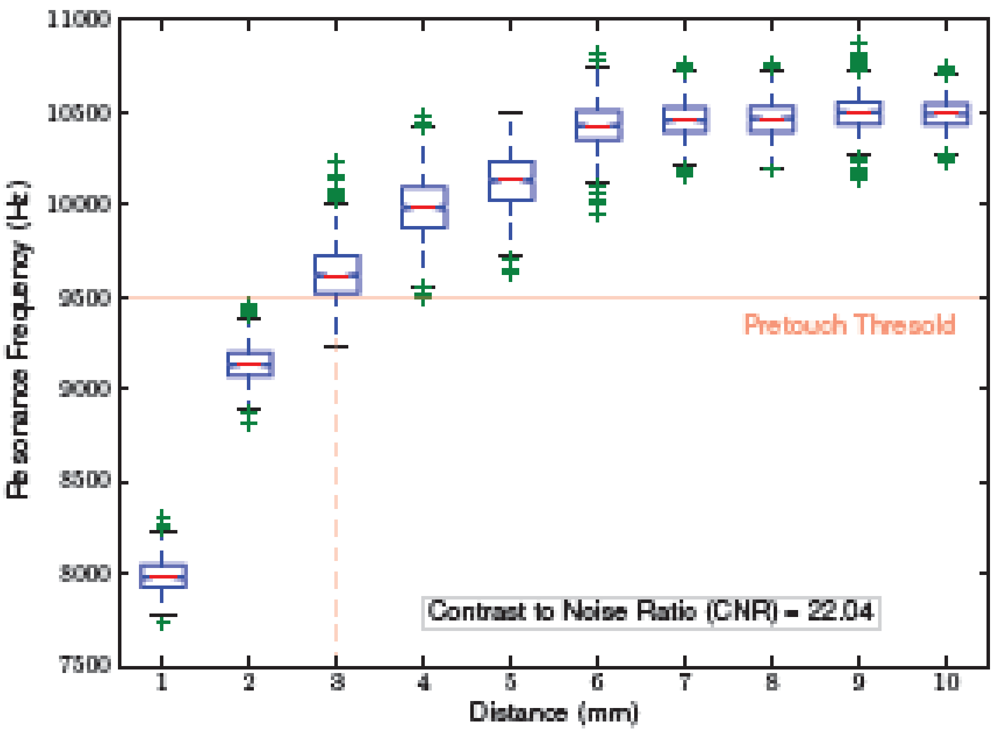
Sensor size on fingertips: 5mm(diameter) x 8mm(length)



Utilize ambient noise → Passive!

Movies
shown
here

Sensor Characterization: Performance



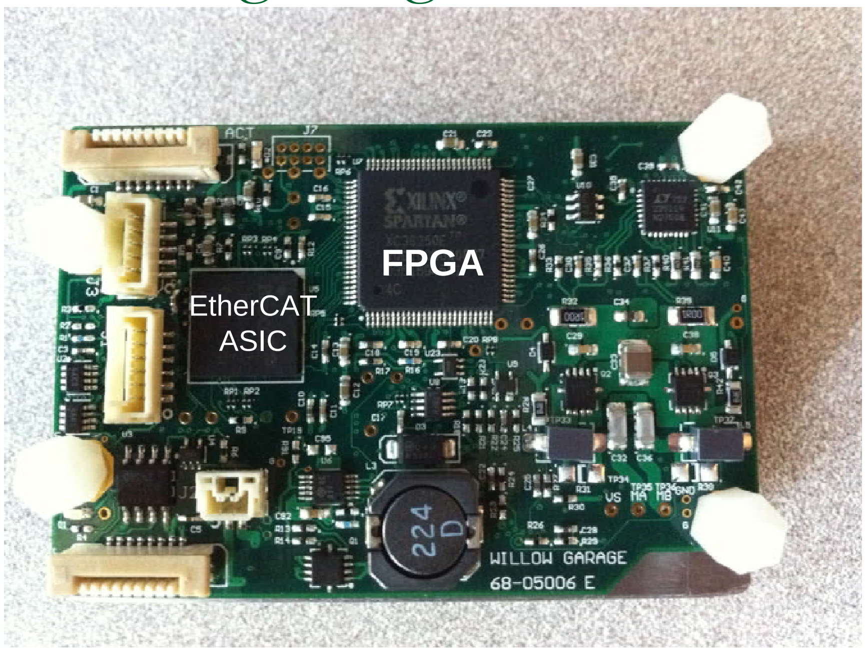
The box and whisker plot of 1000 estimated resonance frequencies at 1-10 mm.

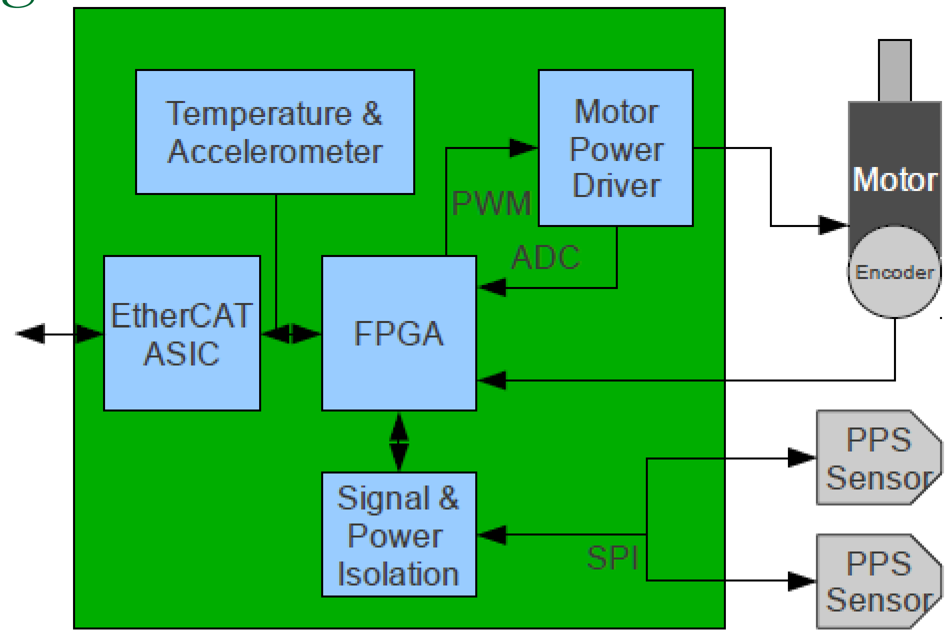
Application Parameters

Frequency: 9500 Hz
Distance: 3 mm

HW Architecture

Integrating w/ Willow Garage PR2 Motor Control Board





Our sensor HW: mic, op-amp, micro w/ ADC, SPI interface
To integrate our new sensor into PR2 robot, replace PPS sensor, use SPI
PicoBlaze soft microcontroller implemented in FPGA
Program PicoBlaze in Pico asm to talk to our new sensor over SPI

even though byte-level SPI is standard, at a higher level our data format is different, so we need to reprogram the SPI master to change from PPS sensor to our mic sensor

PicoBlaze

Soft processor from Xilinx

What is it?

- Configuration of FPGA gates to implement a microcontroller within FPGA fabric

Specs

- 8 bit SPI master peripheral
- 2x 8bit timers to allow easy comms timing
- 512 byte double-buffer for sending sensor data to computer

Why soft processor?

- Easier to program than FPGA
- Faster to reprogram: seconds vs minutes
- Safe: cannot brick the MCB with a bad PicoBlaze program…can easily brick MCB with bad FPGA code

Why PicoBlaze?

- Free license from Xilinx to use on Xilinx FPGAs
- Very small
- Simple, predictable timing (2 clock cycles per instruction)

SPI Master

There is a peripheral that perform fast SPI transfers to a slave device. The peripheral will transfer 8bits of SPI data to/from slave device with selectable clock speed.

Currently, the SPI peripheral only runs in mode 0 (CPOL=0, CPHA=0). There [Wikipedia Serial Peripheral Interface] page that has a good explanation of the different SPI modes.

SPI clock speed

The system clock runs at 25Mhz. This clock can be divided down to produce the clock speed used for the SPI transfers. The formula for determining the SPI clock speed from the divisor value is:

SPI Clock Frequency = 25Mhz / 2 / (Divisor+1) = 12.5Mhz / (Divisor + 1)

The divisor is an 8-bit value so the slowest clock speed available is :

12.5Mhz / (255+1) = 48.8kHz

The fastest (theoretical) clock speed available is 12.5 Mhz. However, high speed SPI signaling has never been tested and may require proper signal termination on the figure tip devices.

SPI data transfer

The SPI peripheral transfers 1-byte data at a time. Starting transfer is as simple as writing SPI_DATA_REG with new output value. A busy status flag can be polled to wait for transfer to complete. To retrieve the received data, the SPI_DATA_REG can be read after a SPI transfer has completed.

Unlink other protocols (ie RS232 serial), SPI always receives 1 byte of input while sending 1 byte of output data that is sent. In some situations either received data does not matter. On these cases, DATA_REG does not need to be read after transfer is complete. In other situations, only the input data matters. For these cases, start a SPI transfer by writing “dummy” data to SPI_DATA_REG.

SPI chip selects

The SPI bus has two chip selects, one for each finger. The chip selects are active-low – they are assert with the signal is low. The digital isolator used for the chip selects is slower than the isolator used for the SPI bus. After asserting a chip-select, the program should wait for 1 or 2 microseconds before transferring SPI data.

SPI code example

Below is an example of performing communication with an Analog Devices LTC1867L 16-bit ADC. The LTC1867L is given a 7bit command telling it what input channel to convert. While sending the device a new command, the device provides the 16-bit result for the last conversion.
In the example a 16 bit transfer is performed as 2x 8-bit transfers. Only the first 7bits of command data matter, for the remain 9bits we send zeros. The command value used in the example is 1000000b, which ends up being 0x8000 when 9 bits of zeros appended to end.

```
; First set SPI clock speed to 500kHz.
;   Divisor = 12.5Mhz / 500kHz  + 1   =   26   = 0x1A
         load s0, 1A
         output s0, SPI_CLOCK_REG
; Assert Chip-select 1
         load s0, SPI_ASSERT_CHIPSEL_1_FLAG
         output s0, SPI_CTRL_REG
; Wait for 2us for chip-select to propagate to device
         load s0, 2
         call delay_us
; Send MSByte of command to device  (0x80)
         load s0, 80
         output s0, SPI_DATA_REG
; Now wait for transfer to complete
         wait_loop_1:
                  input s0, SPI_CTRL_REG
                  test s0, SPI_BUSY_FLAG
                  jump NZ, wait_loop_1
; Get save first byte read from LTC1C1967L into s2
         input s2, SPI_DATA_REG
; Send LSByte of command (0x00)
         load s0, 0
         output s0, SPI_DATA_REG
; Now wait for transfer of second byte to complete
         wait_loop_2:
                  input s0, SPI_CTRL_REG
                  test s0, SPI_BUSY_FLAG
                  jump NZ, wait_loop_2
; Save second byte read from LTC1C1967L into s1
         input s1, SPI_DATA_REG
; De-assert chip-select
         load s0, SPI_DEASSERT_CHIPSEL_FLAG
         output s0, SPI_CTRL_REG
```