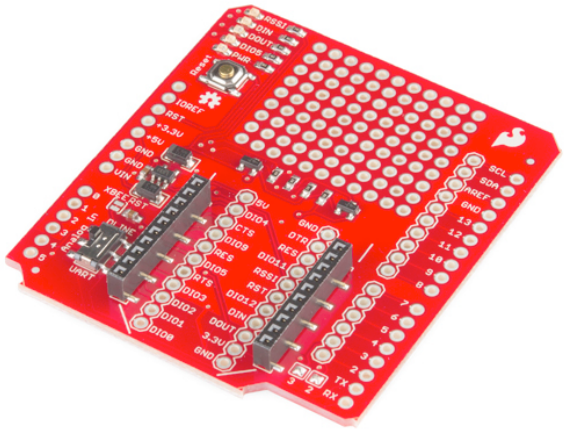
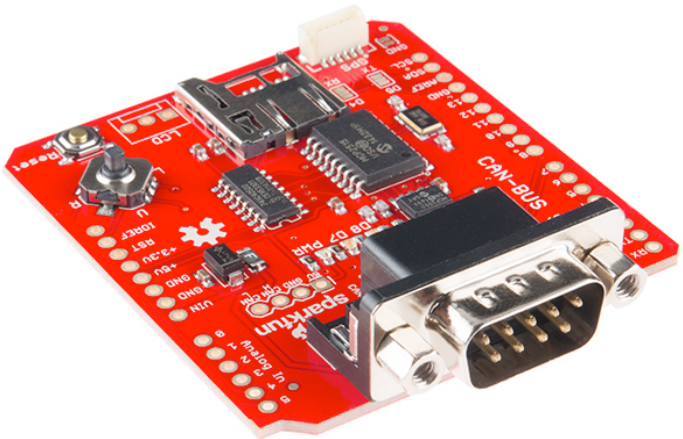
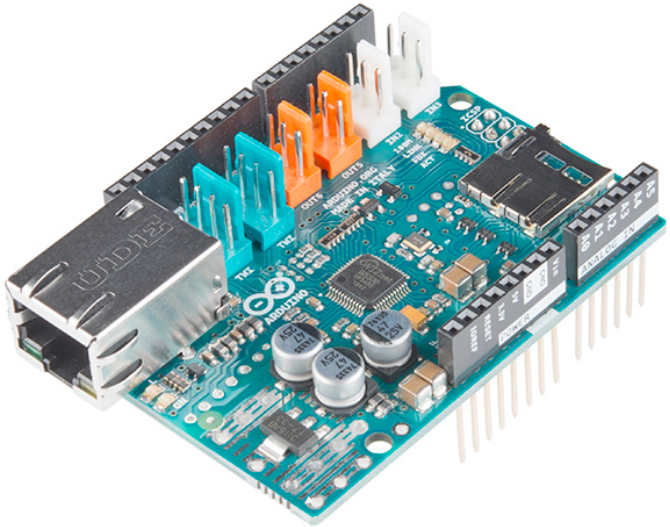
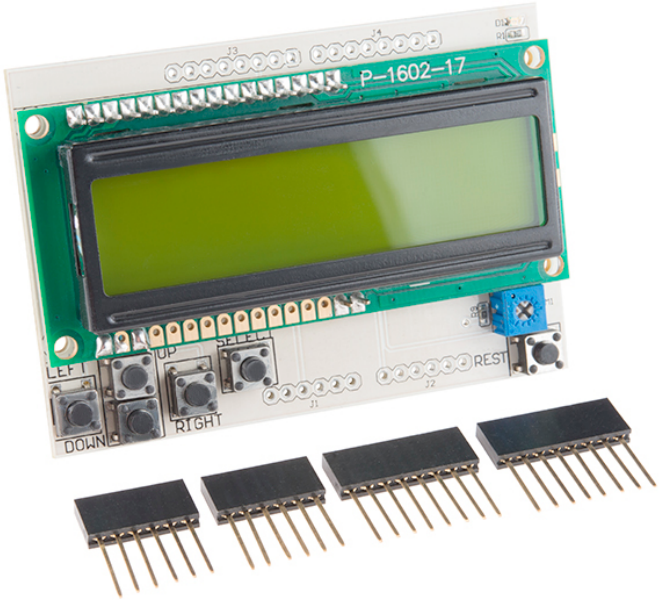
I/O: External Peripherals

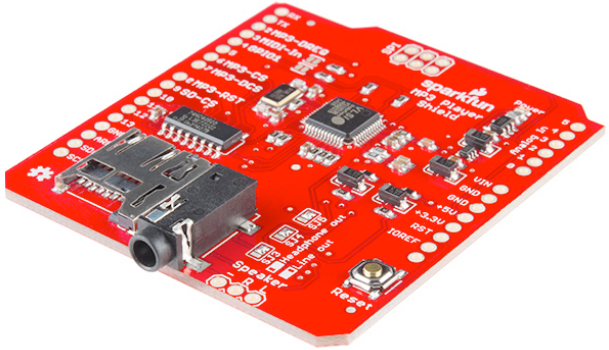
SPI, PS/2, and PWM

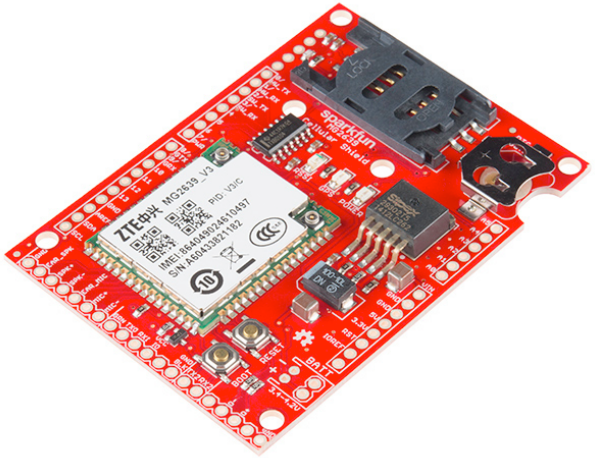


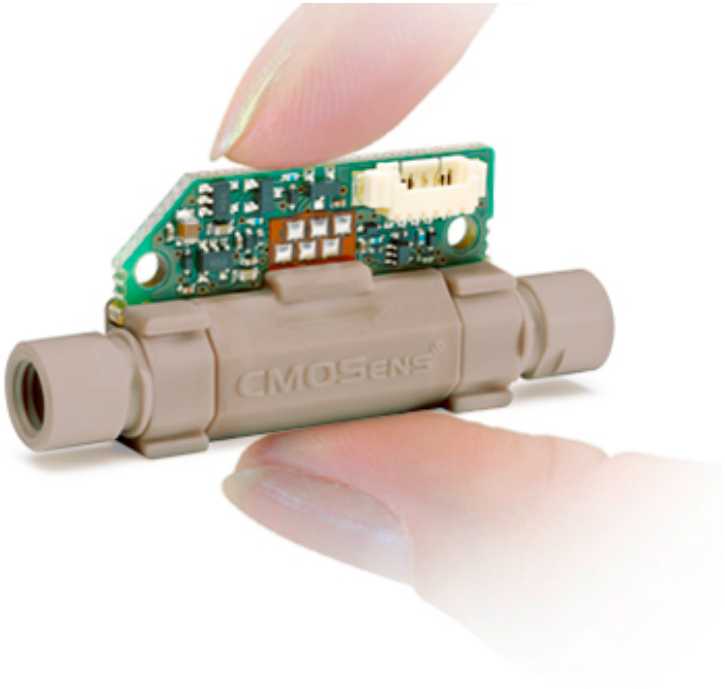

CAN-BUS

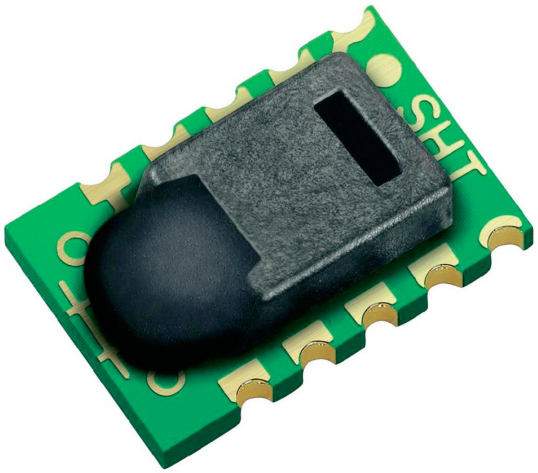



P-1602-17
LEFT
UP
SELECT
RIGHT
DOWN
REST






CMOSENS


SHT


LG

Serial vs. Parallel

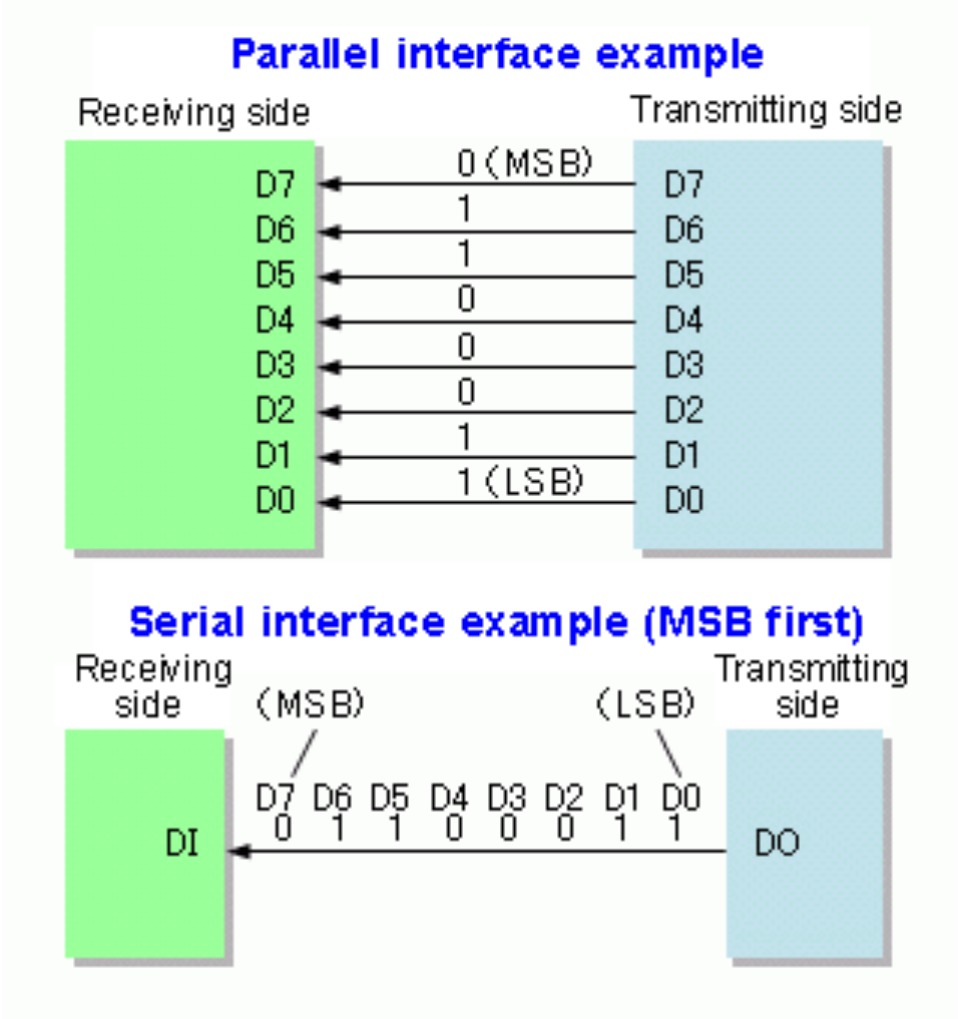
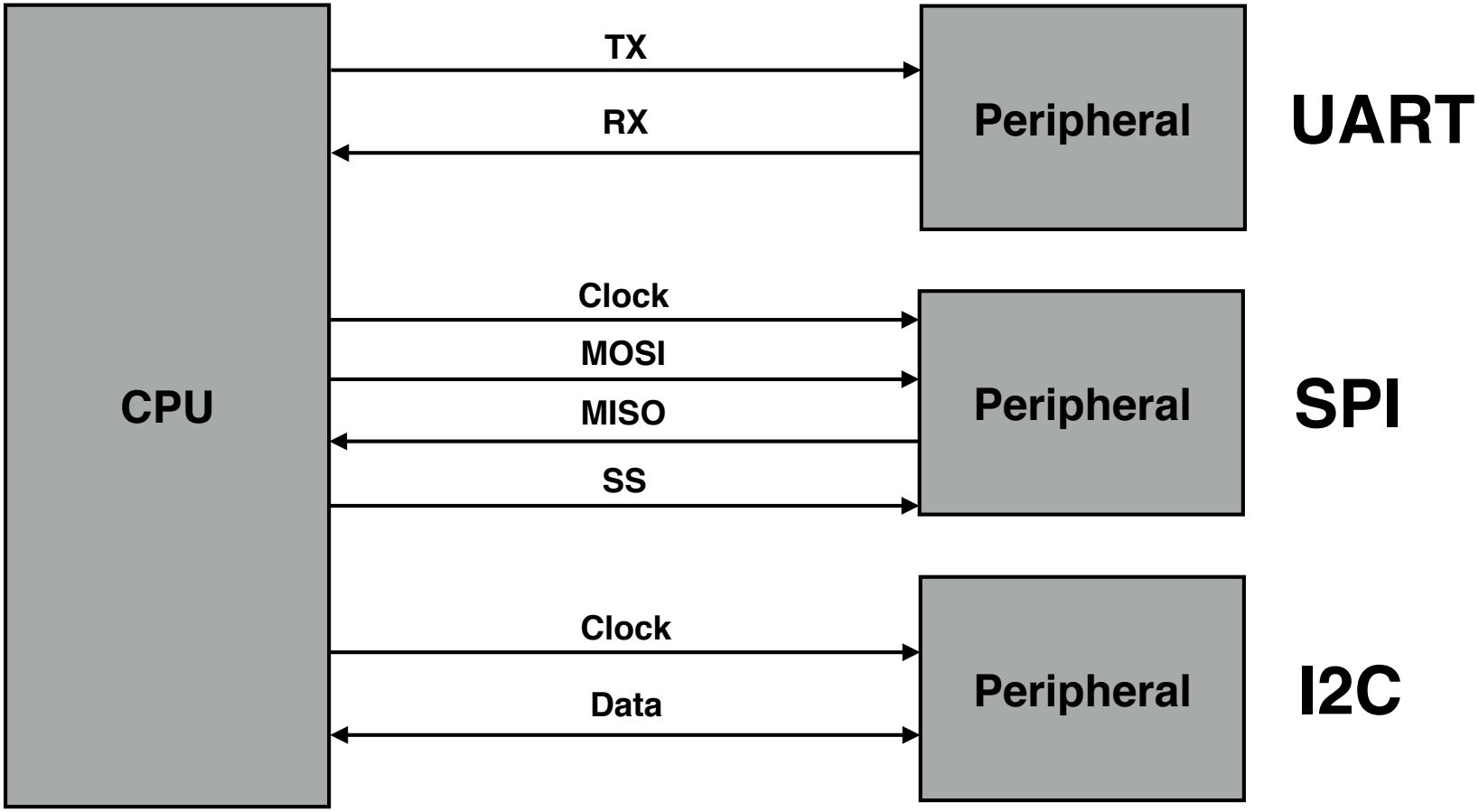


Figure from https://en.wikipedia.org/wiki/Parallel_communication#/media/File:Parallel_and_Serial_Transmission.gif

Bus Protocols



UART

- Used in your printf & the bootloader
- Asynchronous — no clock line
- Start bit, (5 to 9) data bits, (0 or 1) parity bit, (1 or 2) stop bits

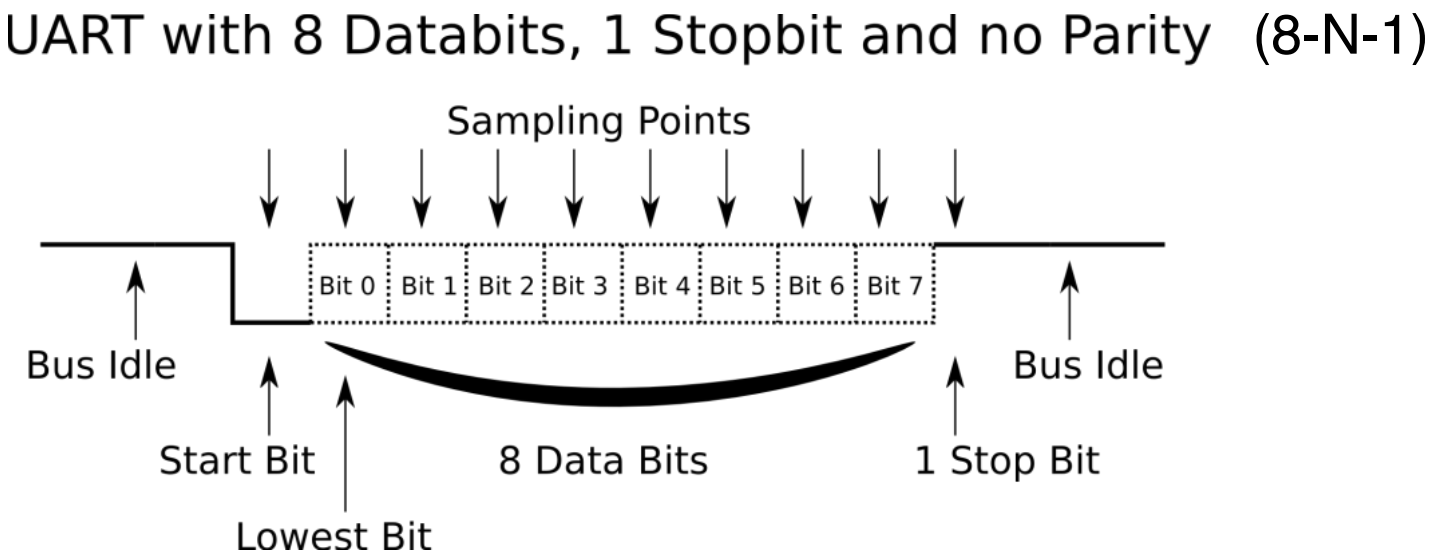


Figure from http://old.myhdl.org/lib/exe/detail.php/projects:rs232image.png?id=projects%3Auart_rs232_receiver_transmitter

UART timing demo

Parity Bits

- Error detection — discard if parity bit wrong
- Even parity: parity = XOR of data bits, ensures an even number of 1s (w/ parity bit)
- Odd parity: !even parity, ensures an odd number of 1s

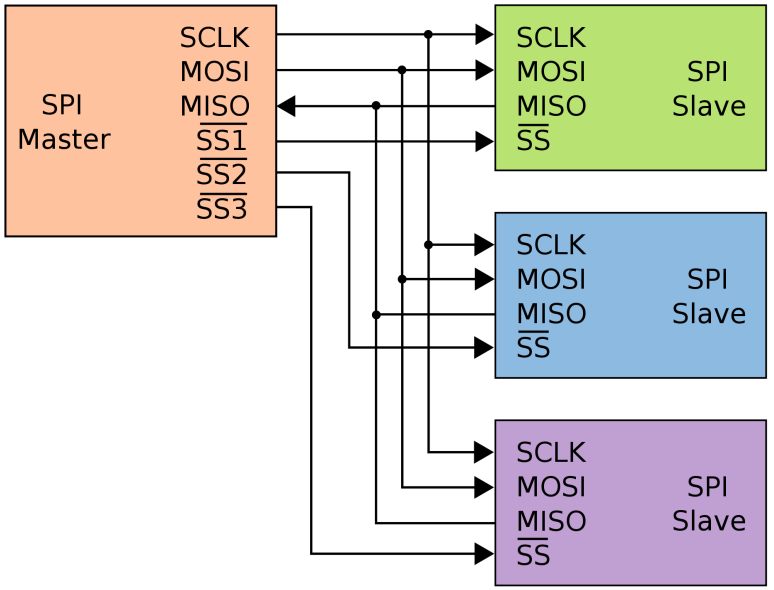
even

data	data	data	data	data	data	data	data	parity
1	1	0	1	0	1	1	0	1

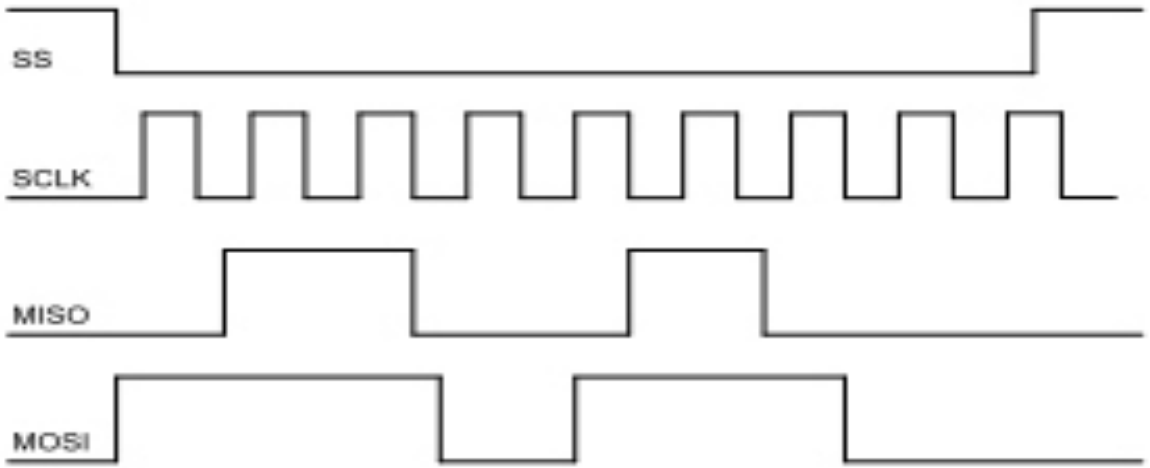
odd

data	data	data	data	data	data	data	data	parity
1	1	0	1	0	1	1	0	0

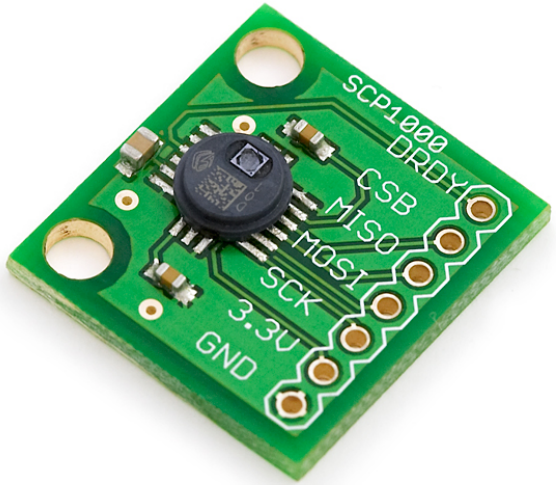
SPI



- Clocked by master
- Shared CLK, MOSI, MISO lines
- Active low chip select (slave select) lines to specify which peripheral is active

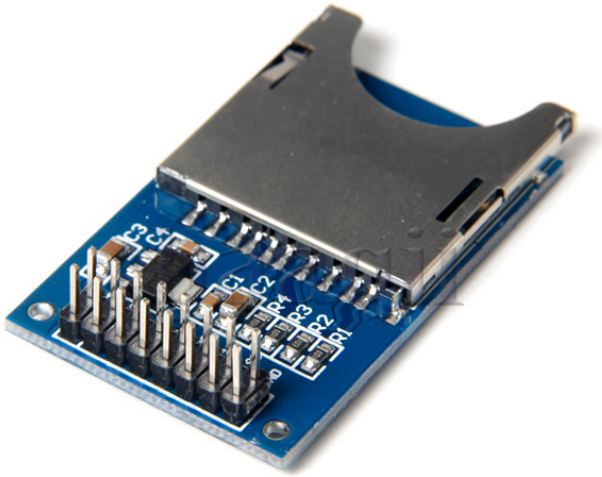


Figures from https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI_three_slaves.svg/2000px-SPI_three_slaves.svg.png (top), http://www.tequipment.net/RigolSD-SPI-DS4.html (bottom)



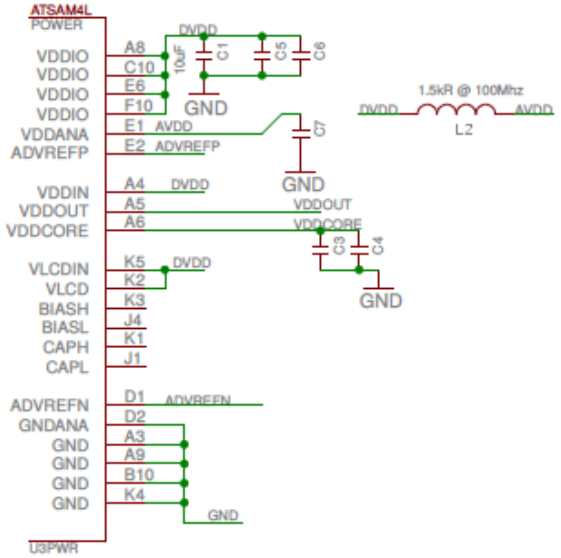
SPI Devices

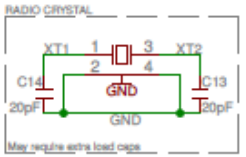
- Sensors (pressure, temperature, Hall effect)
- Control/Configure (ethernet switch, digital potentiometer, OLED display)
- SD Card
- Radios!

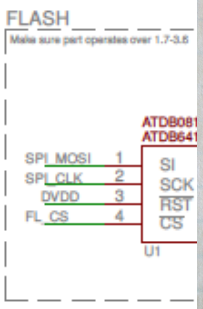


Figures from https://developer.mbed.org/media/uploads/MichaelW/scp1000d01.jpg (top)
http://raspberrypi.stackexchange.com/questions/28648/how-can-i-wire-this-sd-card-reader-to-raspberrypi (bottom)

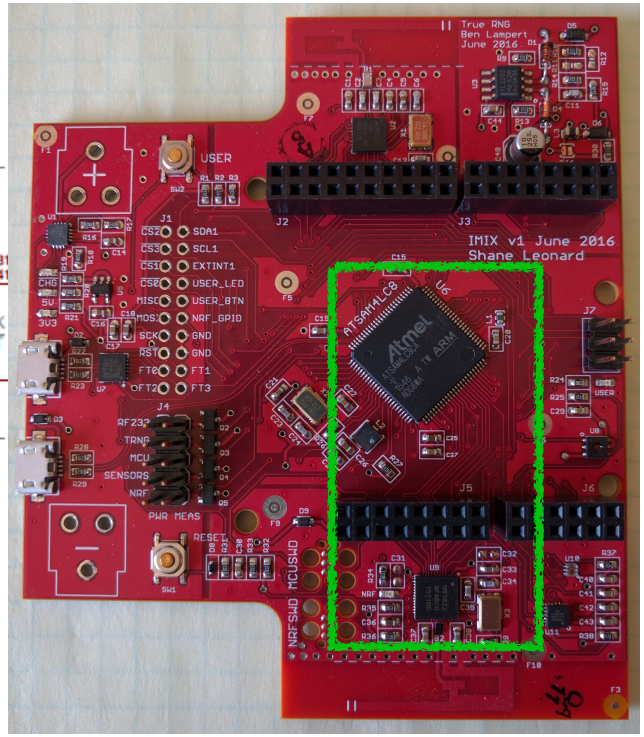
SPI Demo

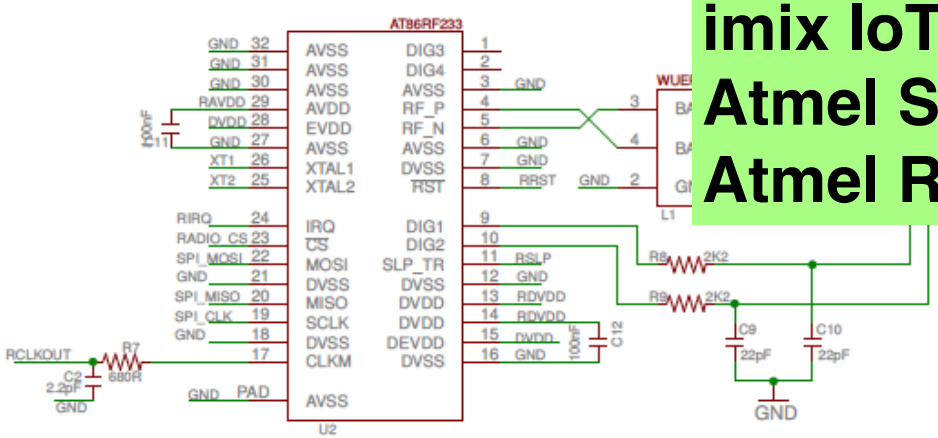






Errata:
C11 + C12 should be 100nF

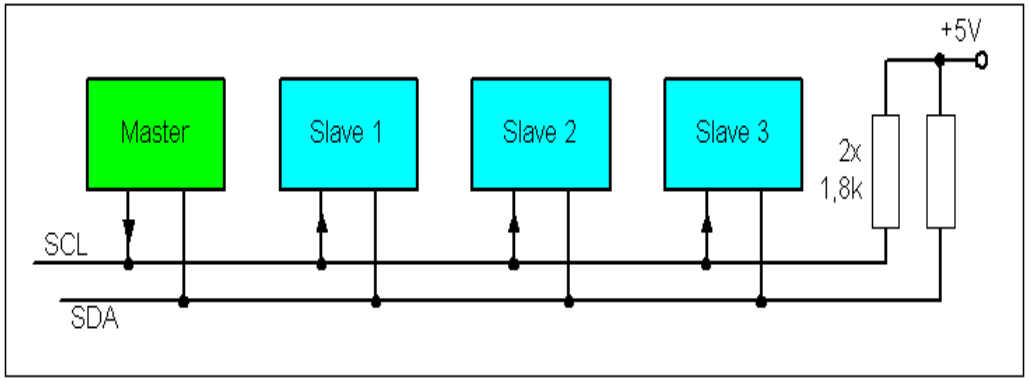




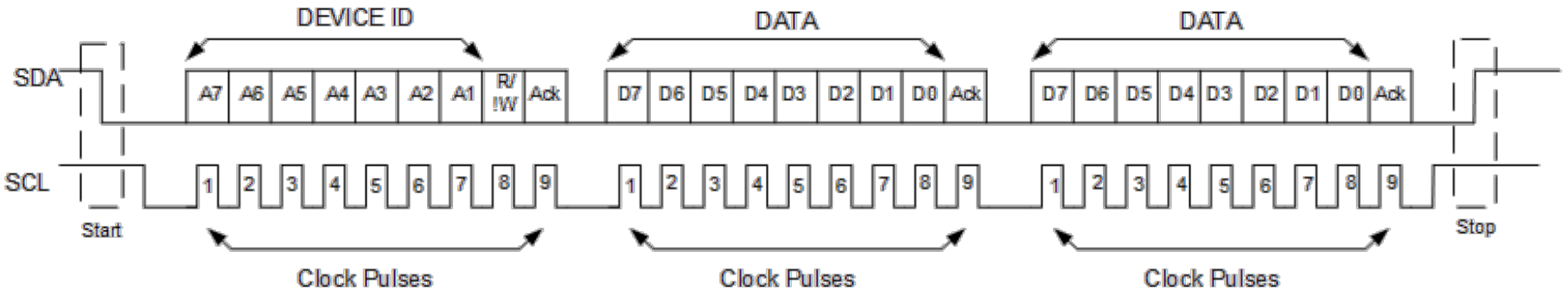
imix IoT research platform
Atmel SAM4L master (ARM Cortex M)
Atmel RF233 slave (802.15.4 radio)

BEARL060

I2C

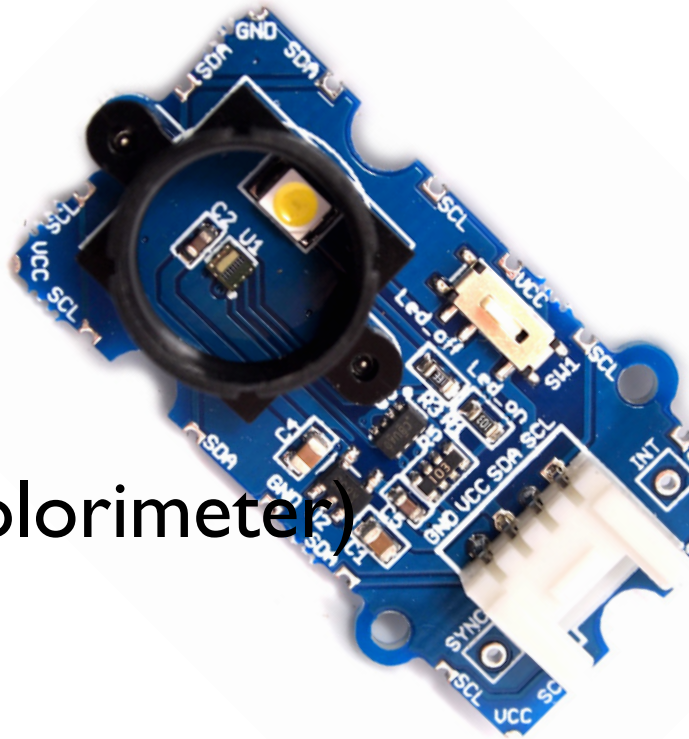


- Only CLK & DATA lines
- Clocked by master, sides alternate who sends data
- Shared bus, slave identified by 7 (or 10) bit address



Figures from http://www.cs.fsu.edu/~baker/devices/notes/graphics/i2cbus3.gif (top)
https://learn.digilentinc.com/Documents/chipKIT/chipKITPro/P08/Fig_1_Waveform.png (bottom)

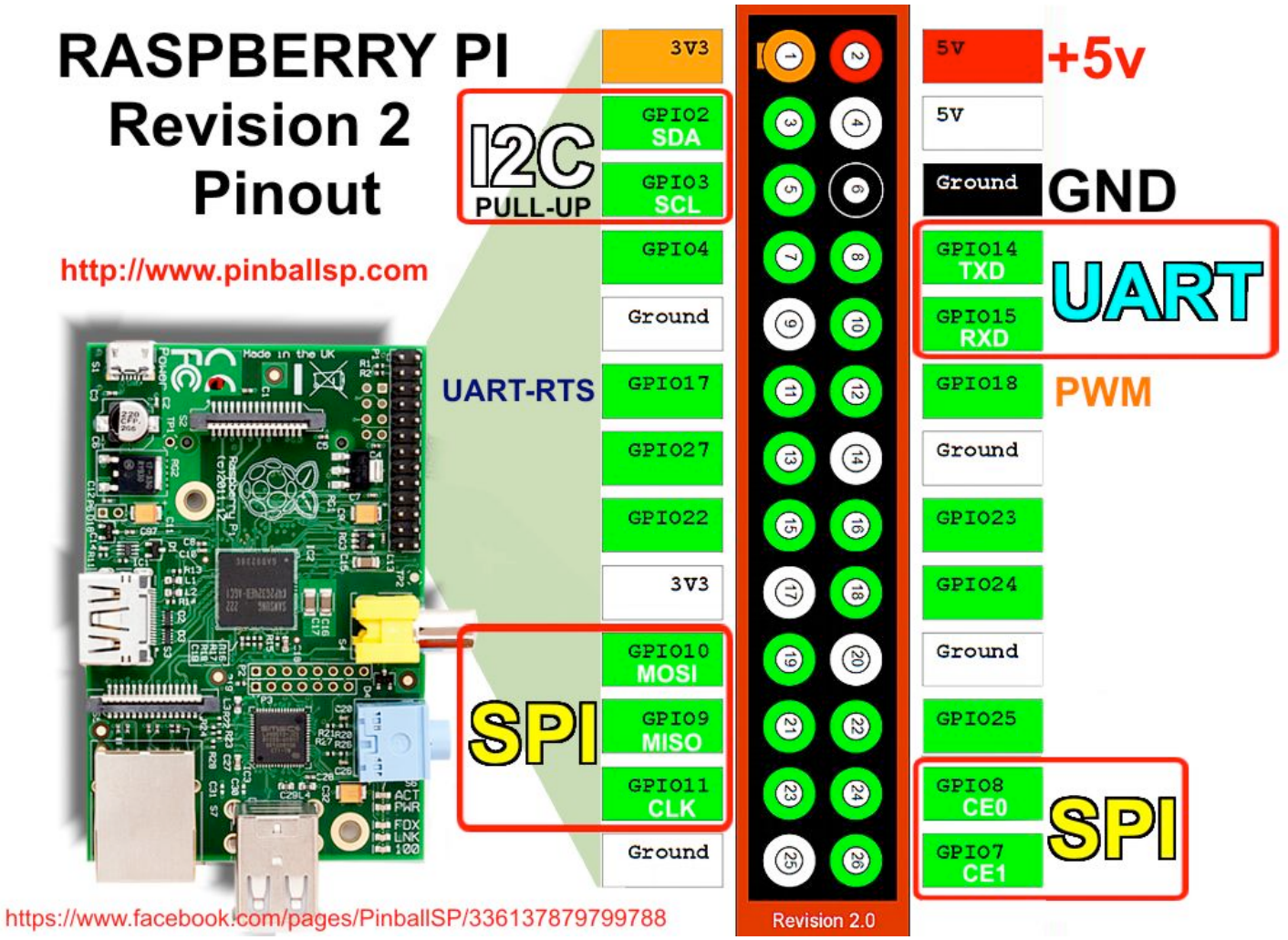
I2C Devices



- Sensors (pressure, temperature, colorimeter)
- Control/Configure (HDMI display)
- ADC & DAC

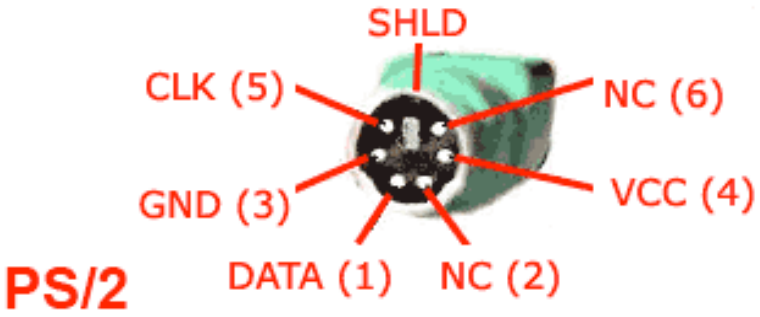
Figure from http://www.seeedstudio.com/wiki/Grove_-_I2C_Color_Sensor

Raspberry Pi Header Pins



PS/2 Keyboard

- PS/2 is an old serial protocol for keyboards
- Synchronous: *CLK* and DATA lines



PS/2 Protocol

- 8-Odd-1 (8 data bits, odd parity, 1 stop bit)
- Data changes when clock line goes high
- Read data when clock is low
- Open-collector CLK & DATA
  - High is an open circuit
  - Low is connected to ground
  - Need a pull-up resistor to make high actually high

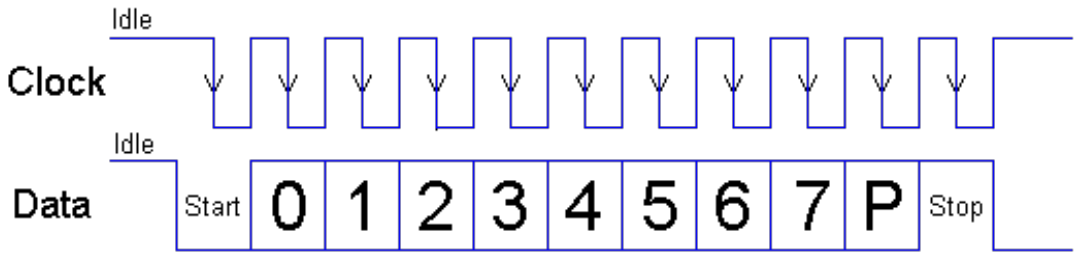
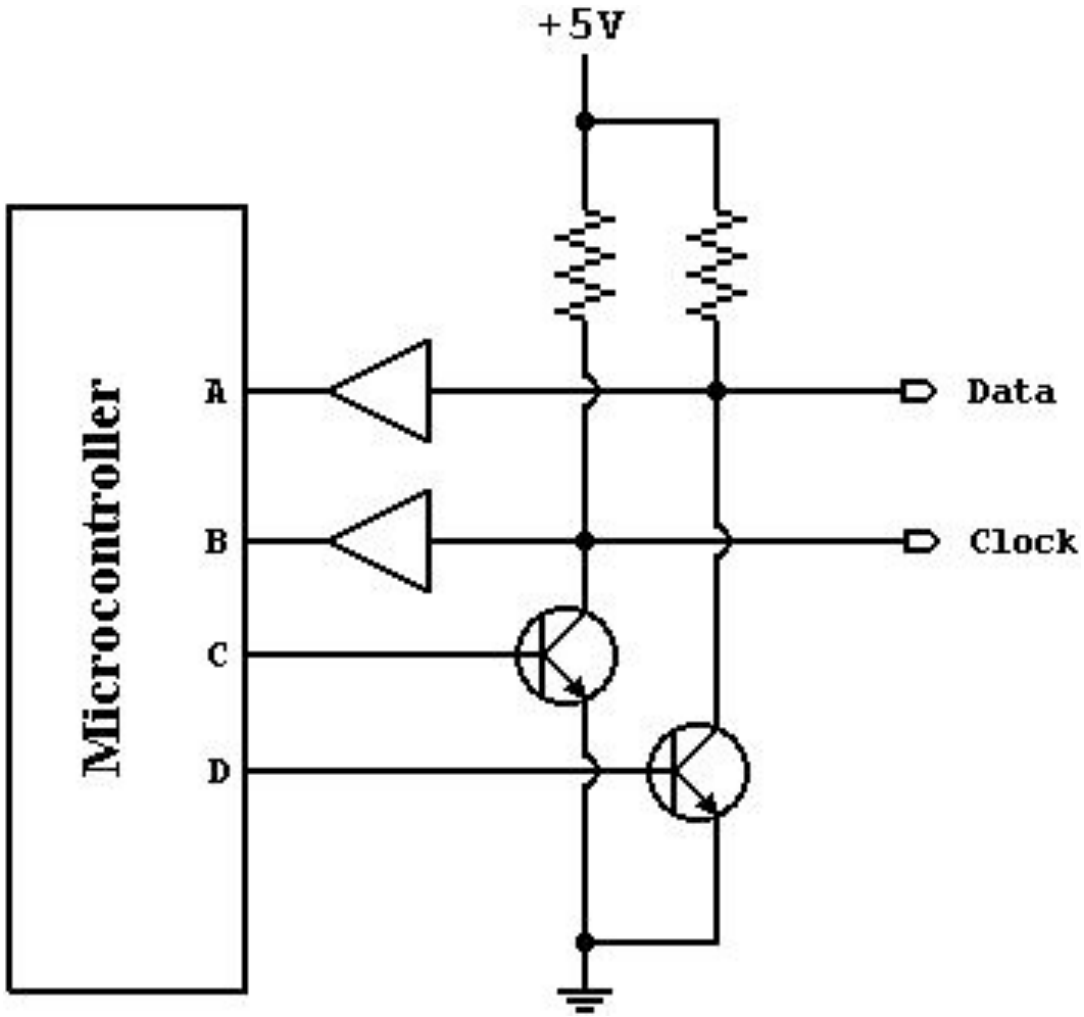


Figure from http://retired.beyondlogic.org/keyboard/keyboar1.gif

Open Collector



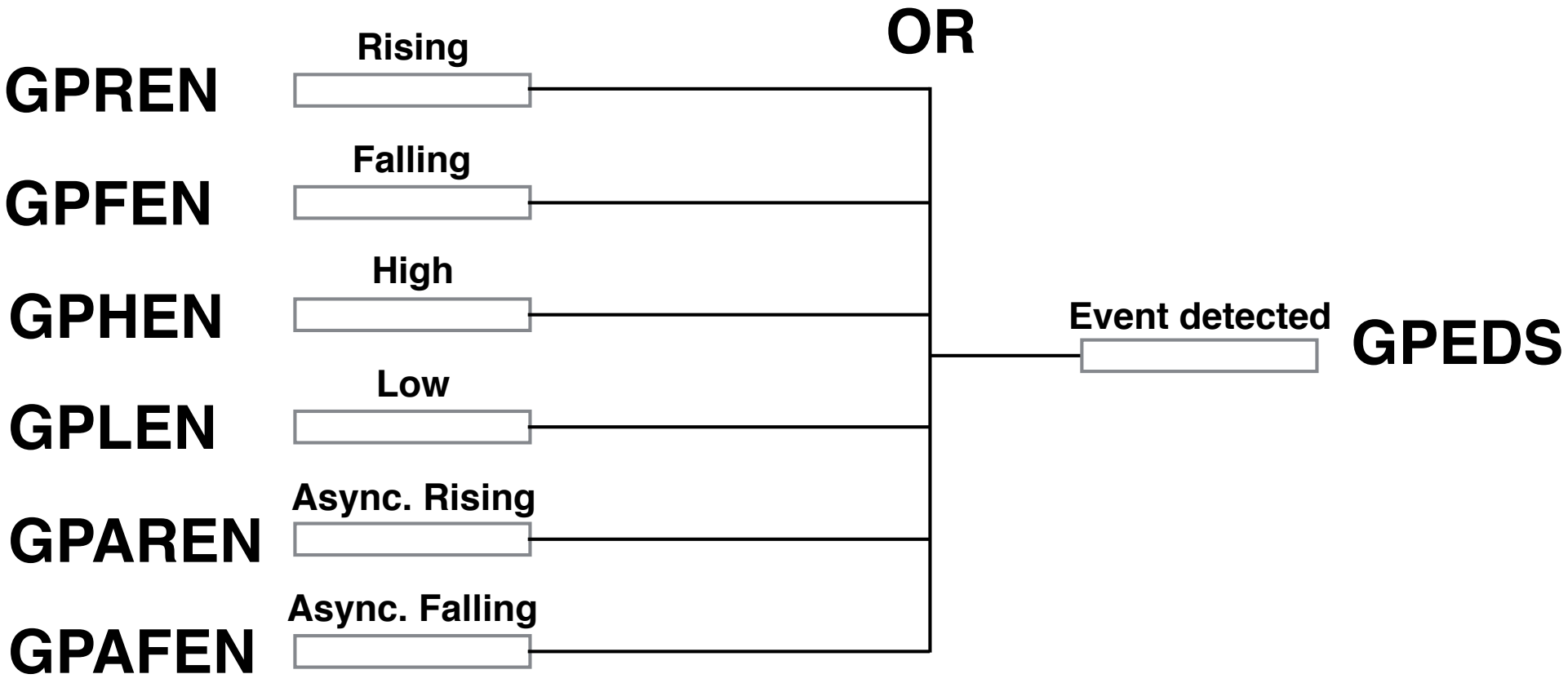
- Asserting C will bring clock low
  - Otherwise goes high from pull-up resistor
- Asserting D will bring data low
  - Otherwise goes high from pull-up resistor

GPIO Event Detection

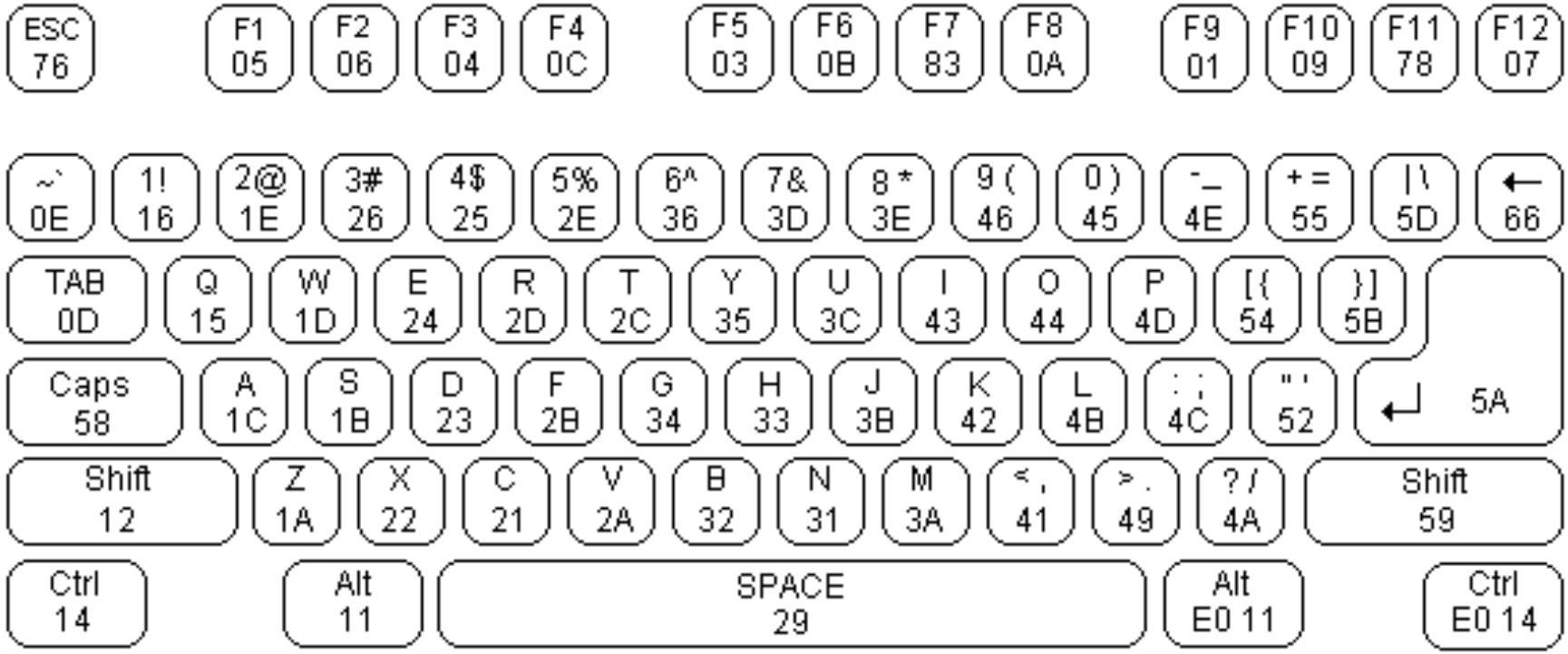
- Can detect falling / rising edge, low / high level
- Set bit for pin in appropriate GPIO event detect enable register (e.g. GPFEN)
- Once enabled, events on that pin will set a bit in the GPIO event detect status register (GPEDS)
- Check GPEDS register for event
- Clear event by writing 1 to bit in GPEDS

See BCM2835-ARM-Peripherals manual pages 96-100

GPIO Event Detection



Keyboard Data



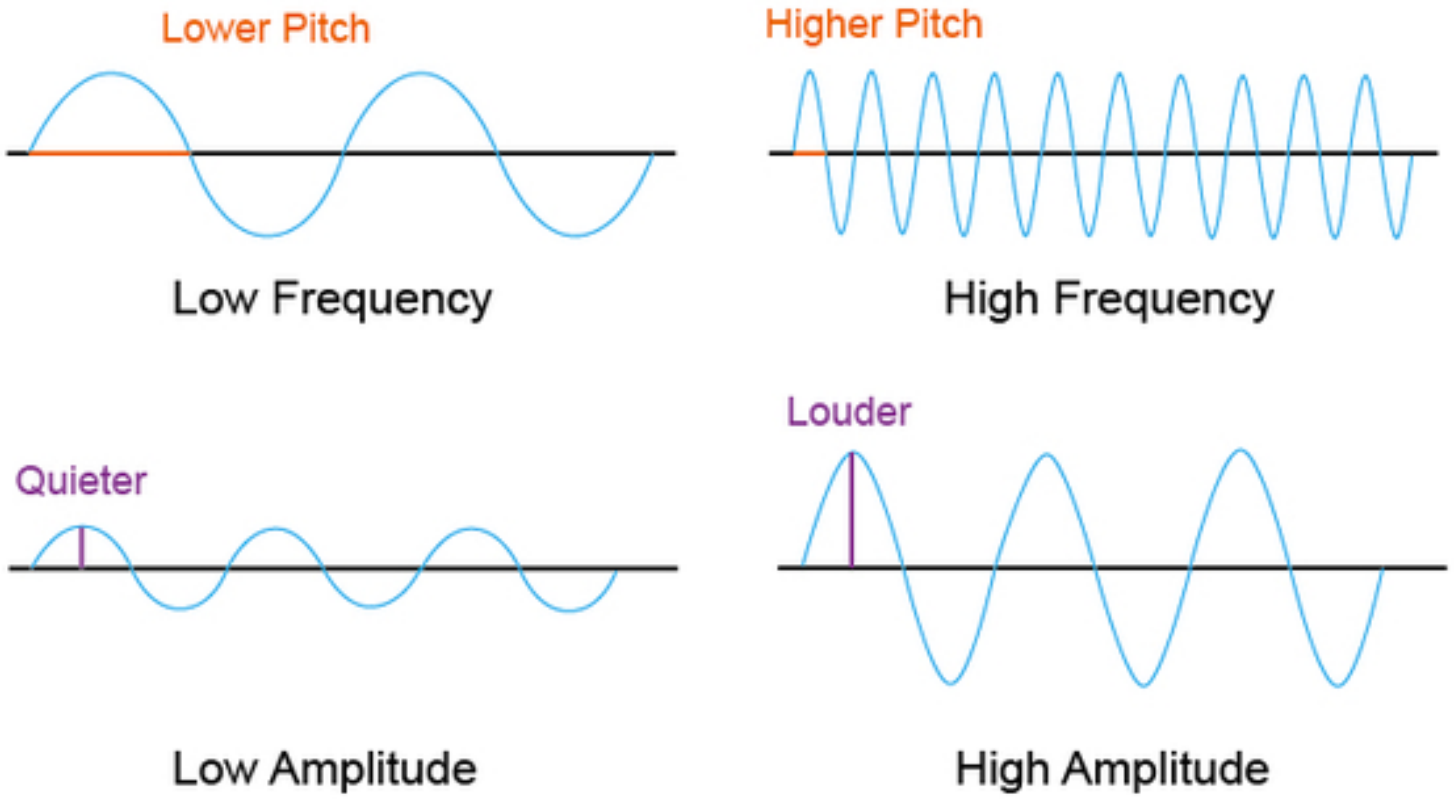
Key	Action	Scan Code
A	Make (down)	0x1C
A	Break (up)	0xF0 0x1C
Shift L	Make (down)	0x12
Shift L	Break (up)	0xF0 0x12

Keys != Characters

- Keyboard scancodes usually converted to ASCII bit stream
- Conversion throws away some info (left-shift vs. right-shift, multiple keys pressed, alt/cmd + key, etc.)
- Sometimes want the extra info (e.g. games) so interface directly with scancodes

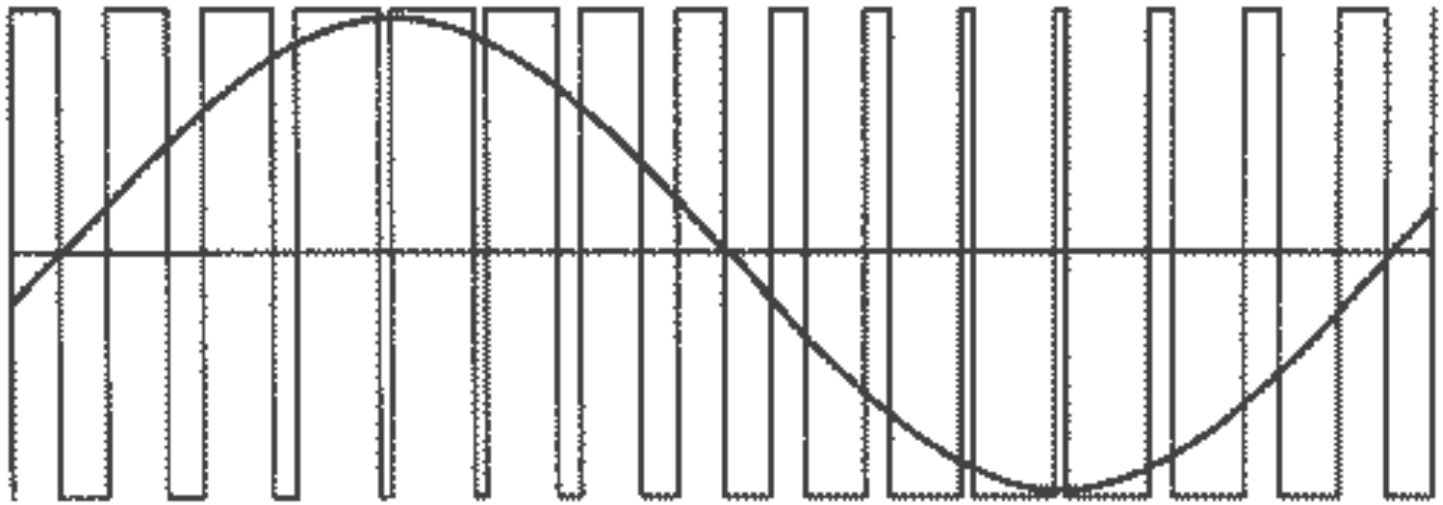
PWM & Sound

Sound Waves



Pulse Width Modulation

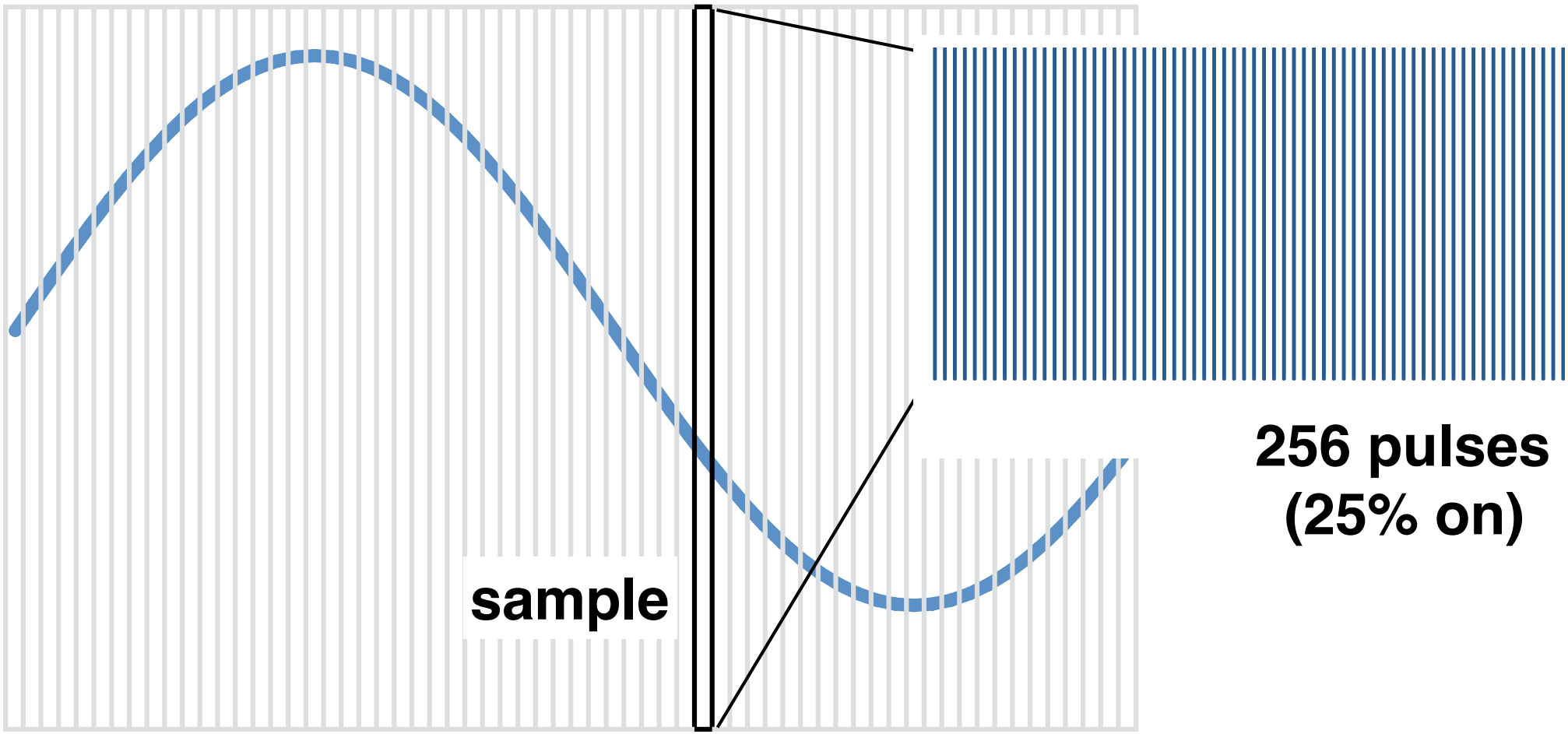
Can simulate continuous values with fast enough PWM clocking



Hardware PWM Support

- Start with a 19.2MHz clock, divide it to specify the time slots of on/off (e.g., divider of 2.375 = 8,192kHz)
- Divide wave into steps (e.g., 64)
- Divide each step into train of (e.g., 256) pulses
- Tell hardware how many pulses should be high

Example: Sine



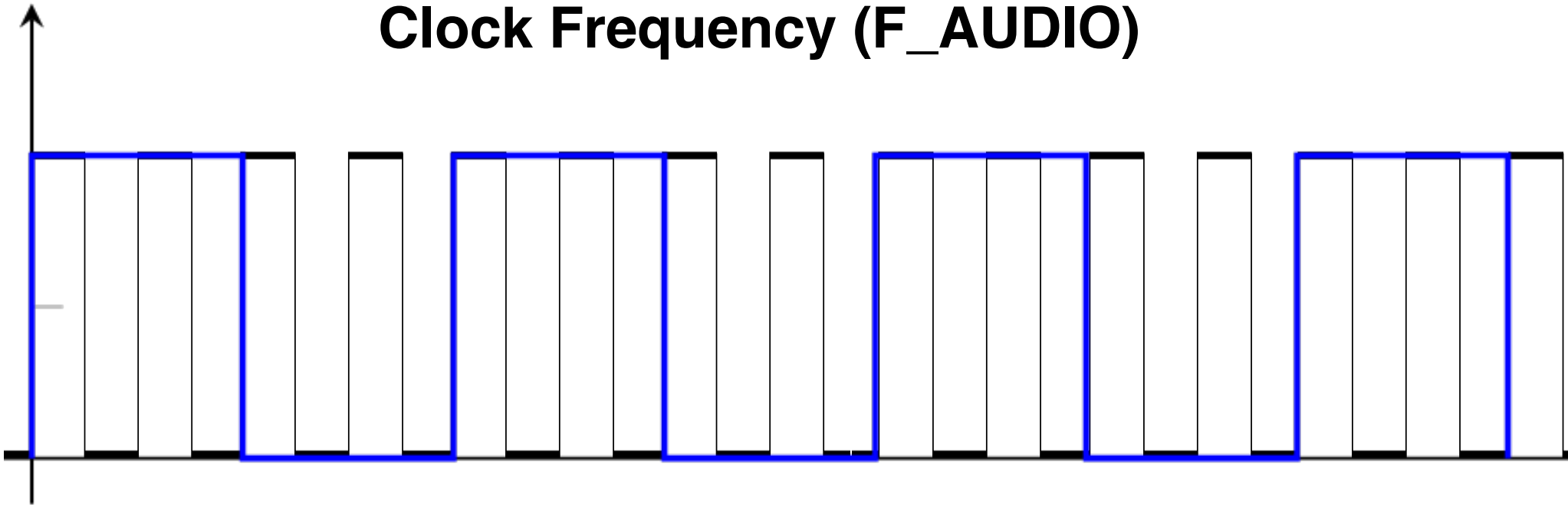
64 samples

1kHz wave * 64 samples * 256 pulses = 8,192kHz

Square Wave

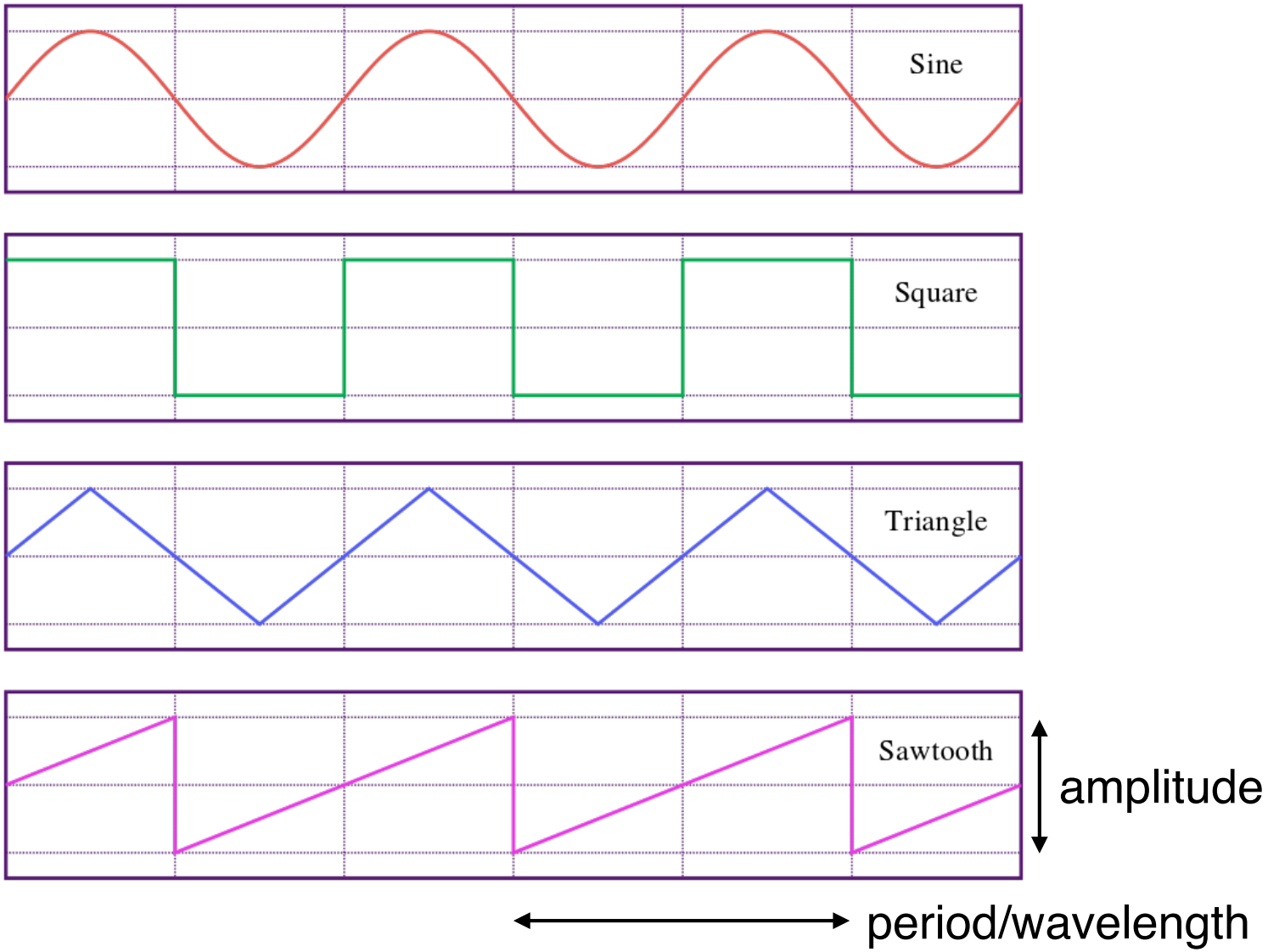
Tone Frequency

Clock Frequency (F_AUDIO)



Range = 4, Width = 2

Waveforms


Sine
Square
Triangle
Sawtooth
amplitude
period/wavelength