Bus Protocols
& Keyboard Input

Serial vs. Parallel

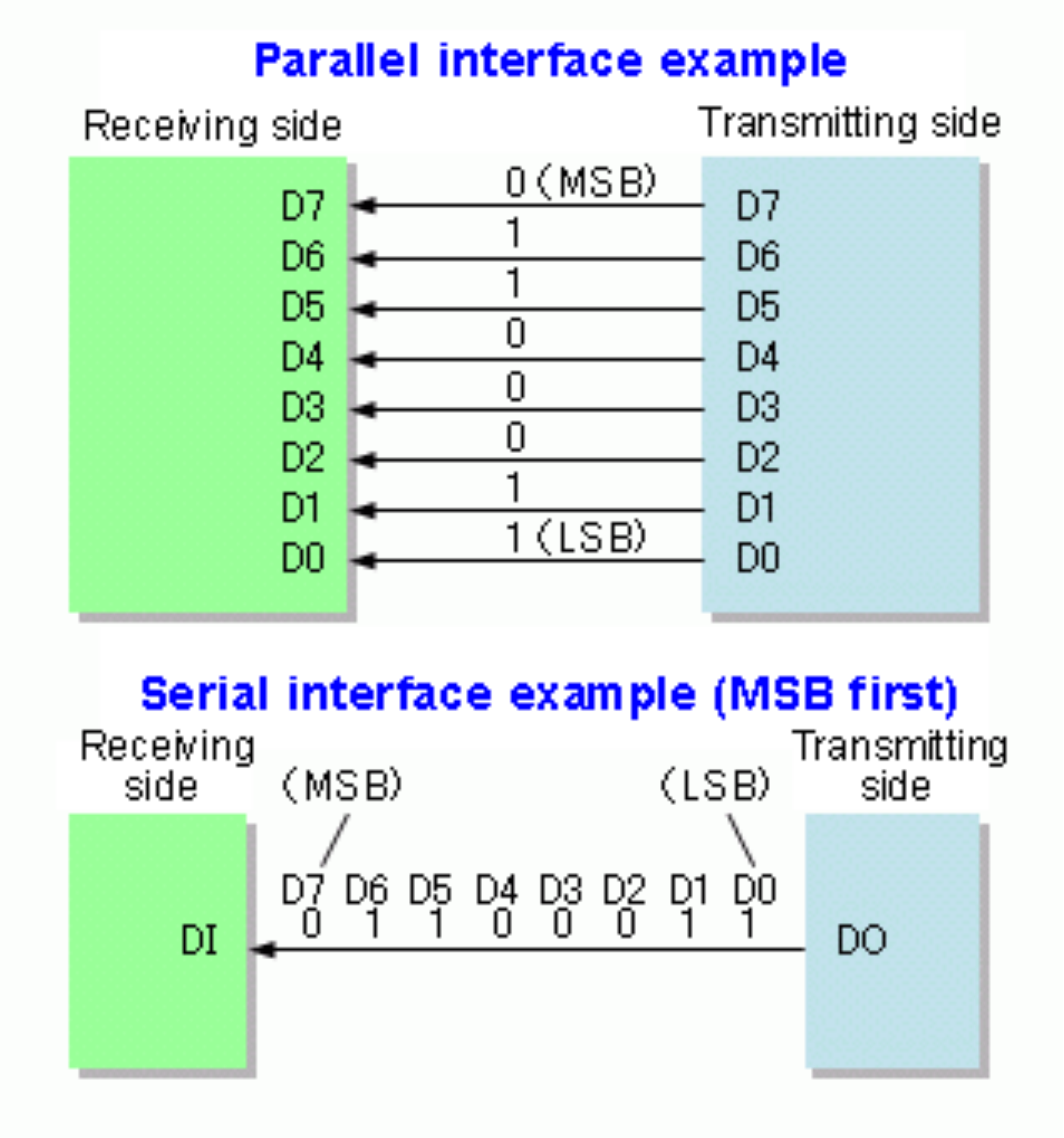
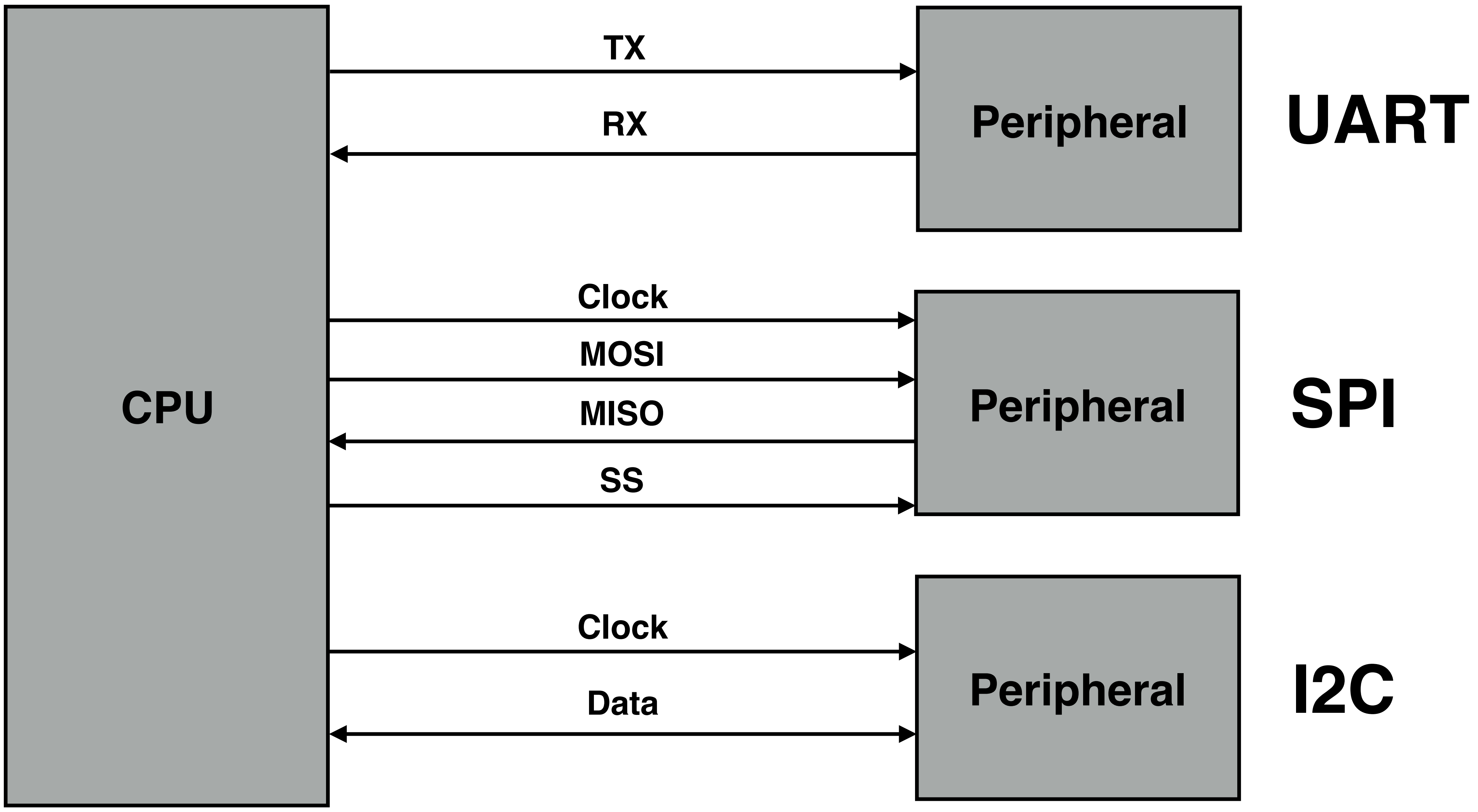


Figure from https://en.wikipedia.org/wiki/Parallel_communication#/media/File:Parallel_and_Serial_Transmission.gif

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**

UART with 8 Databits, 1 Stopbit and no Parity (8-N-1)

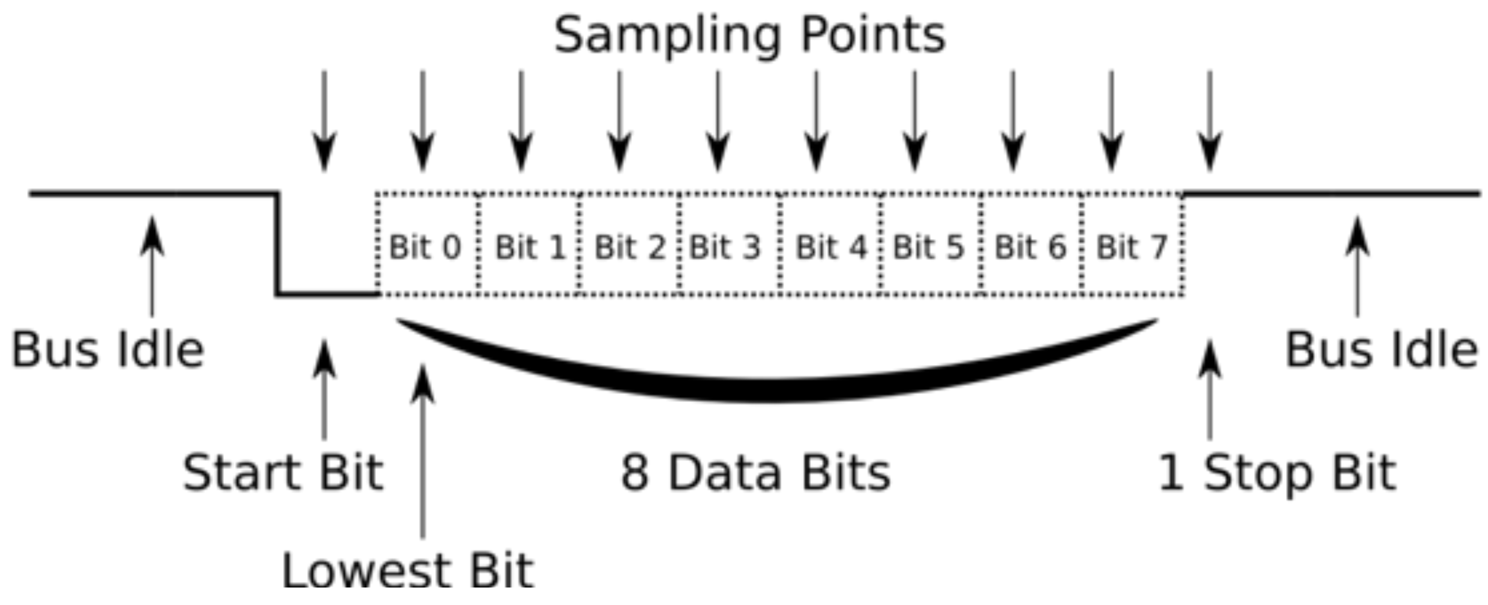


Figure from http://old.myhdl.org/lib/exe/detail.php/projects:rs232image.png?id=projects%3Auart_rs232_receiver_transmitter

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**

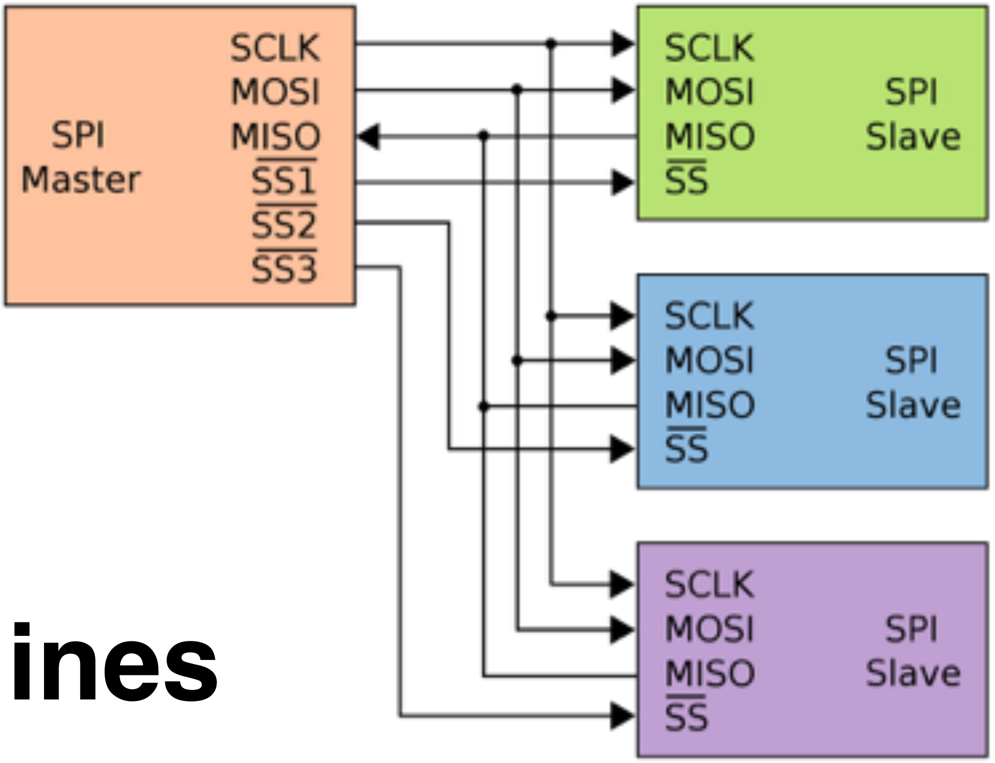
even

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

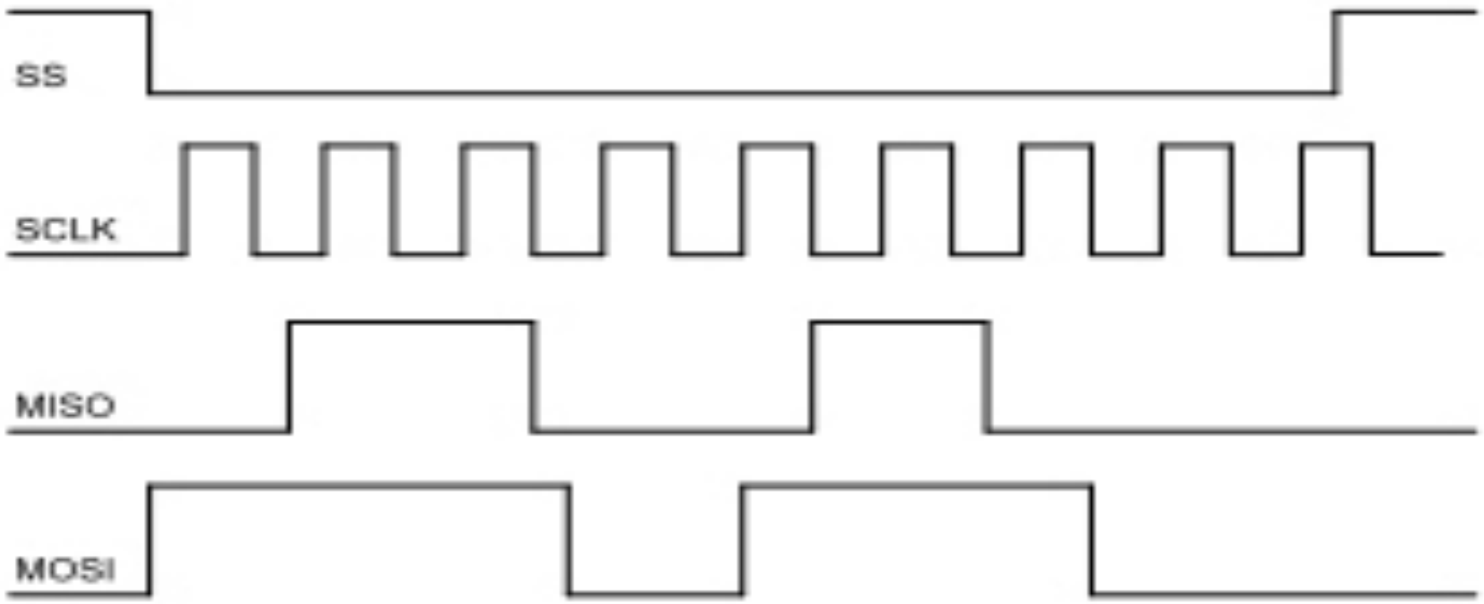
odd

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

SPI



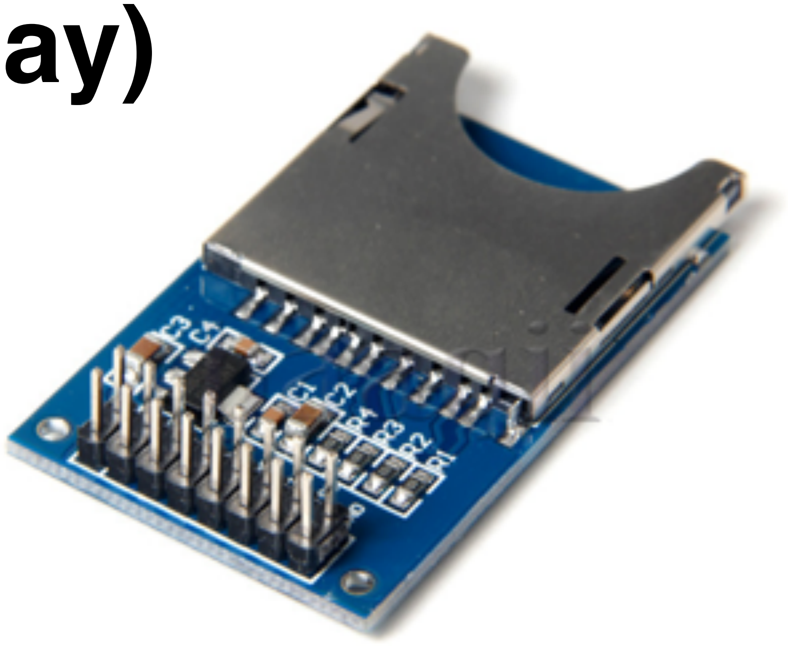
- **Clocked by master**
- **Shared CLK, MOSI, MISO lines**
- **Active low chip select (slave select) lines to specify which peripheral is active**



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)

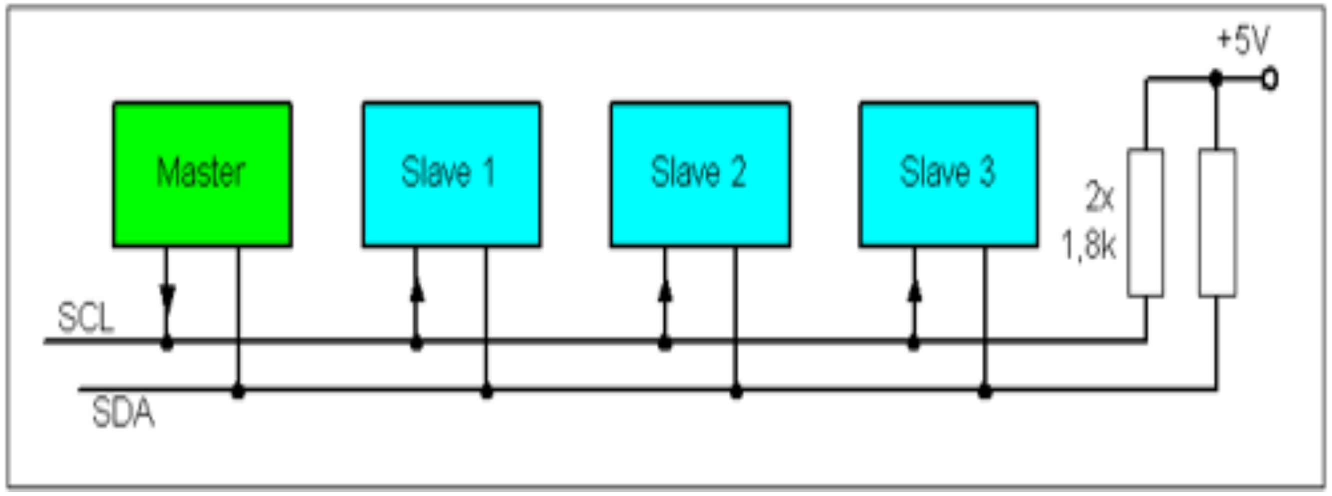
SPI Devices

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



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)

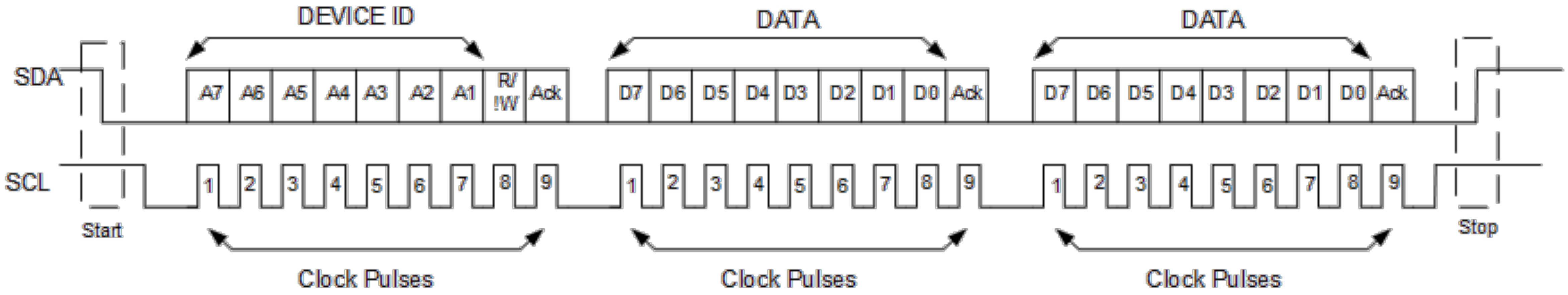
I2C



- **Only CLK & DATA lines**

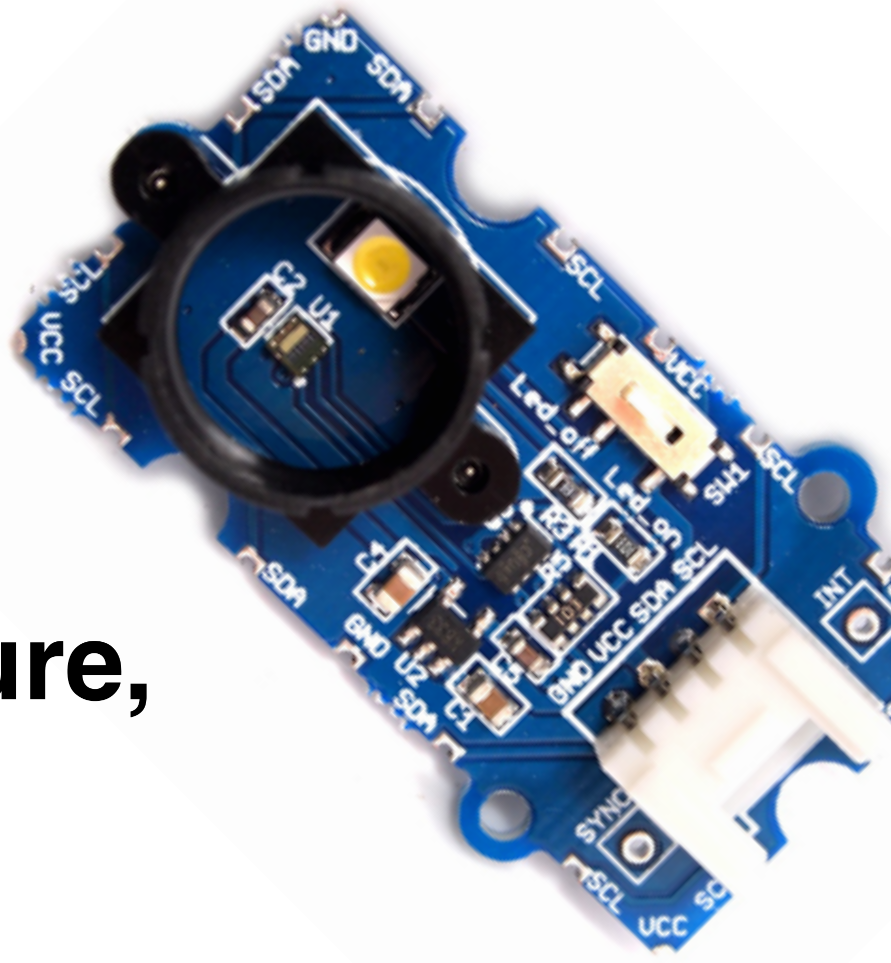
- **Clocked by master, sides alternate who sends data**

- **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)

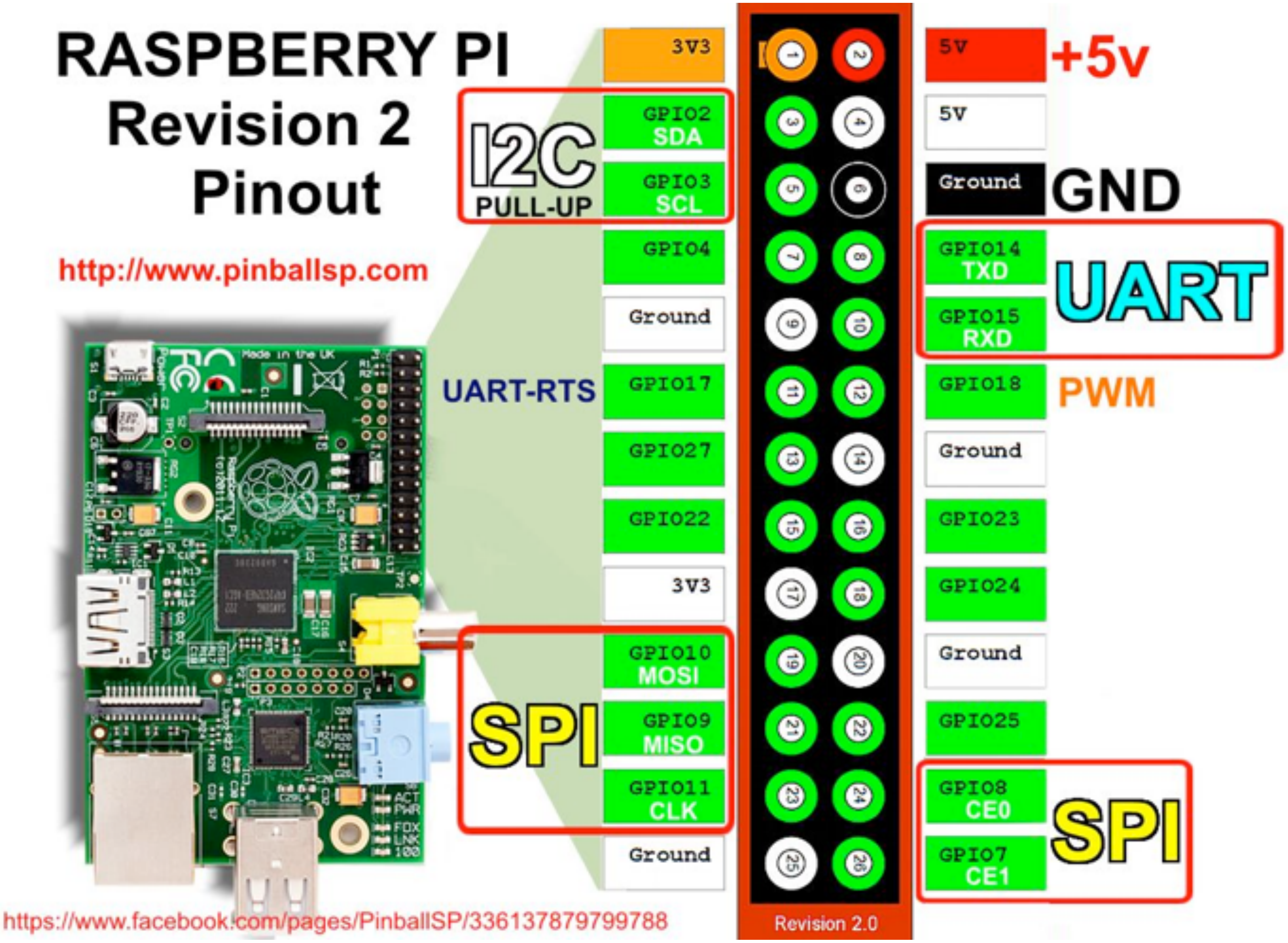
I2C Devices



- 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



RASPBERRY PI
Revision 2
Pinout

http://www.pinballsp.com

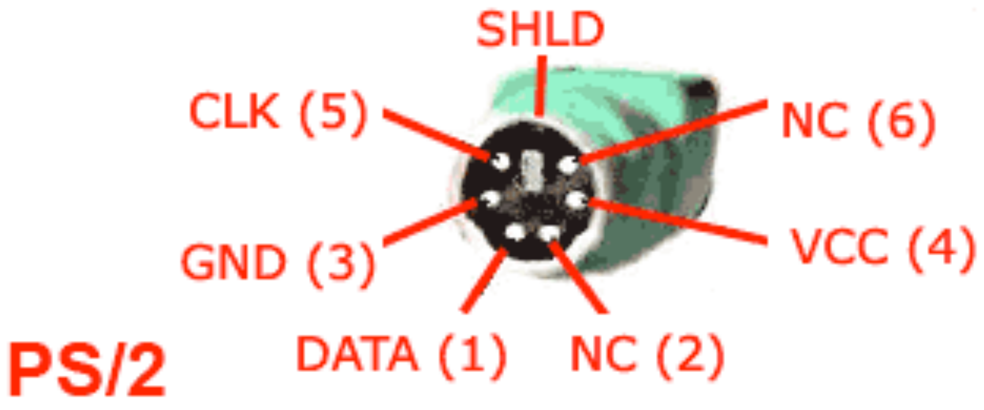
	3V3	1	2	5V	+5v
I2C PULL-UP	GPIO2 SDA	3	4	5V	
	GPIO3 SCL	5	6	Ground	GND
	GPIO4	7	8	GPIO14 TXD	UART
	Ground	9	10	GPIO15 RXD	
UART-RTS	GPIO17	11	12	GPIO18	PWM
	GPIO27	13	14	Ground	
	GPIO22	15	16	GPIO23	
	3V3	17	18	GPIO24	
SPI	GPIO10 MOSI	19	20	Ground	
	GPIO9 MISO	21	22	GPIO25	
	GPIO11 CLK	23	24	GPIO8 CE0	SPI
	Ground	25	26	GPIO7 CE1	

Revision 2.0

https://www.facebook.com/pages/PinballSP/336137879799788

PS/2 Keyboard

- PS/2 is an old serial protocol for keyboards
- 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 — Need pull-up resistors

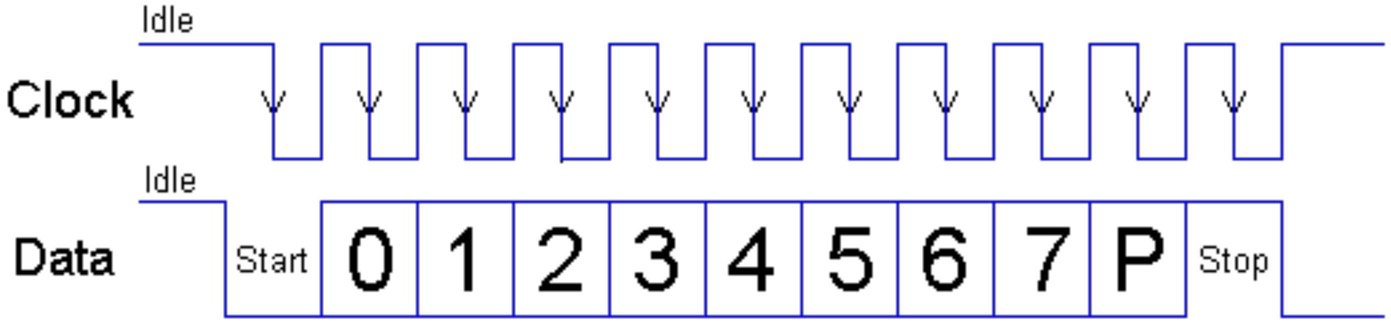
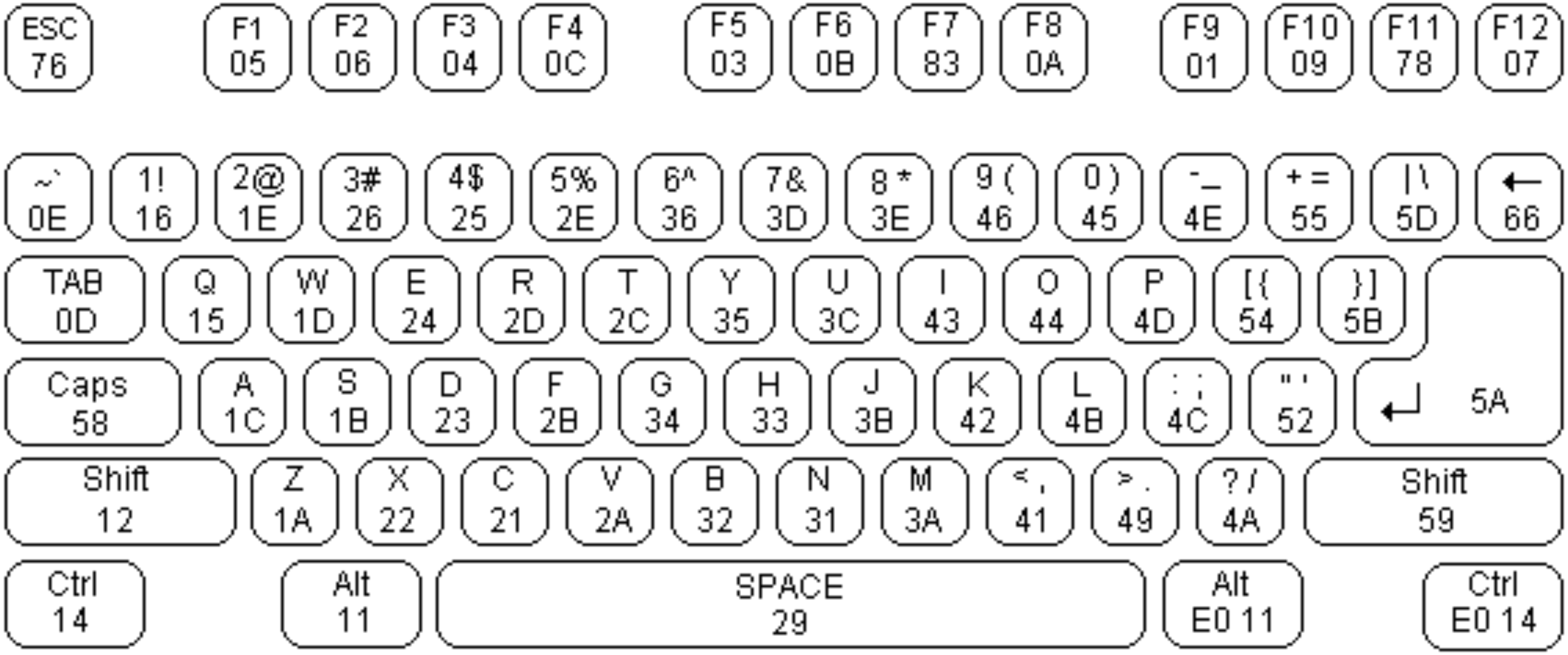


Figure from http://retired.beyondlogic.org/keyboard/keyboar1.gif

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.)

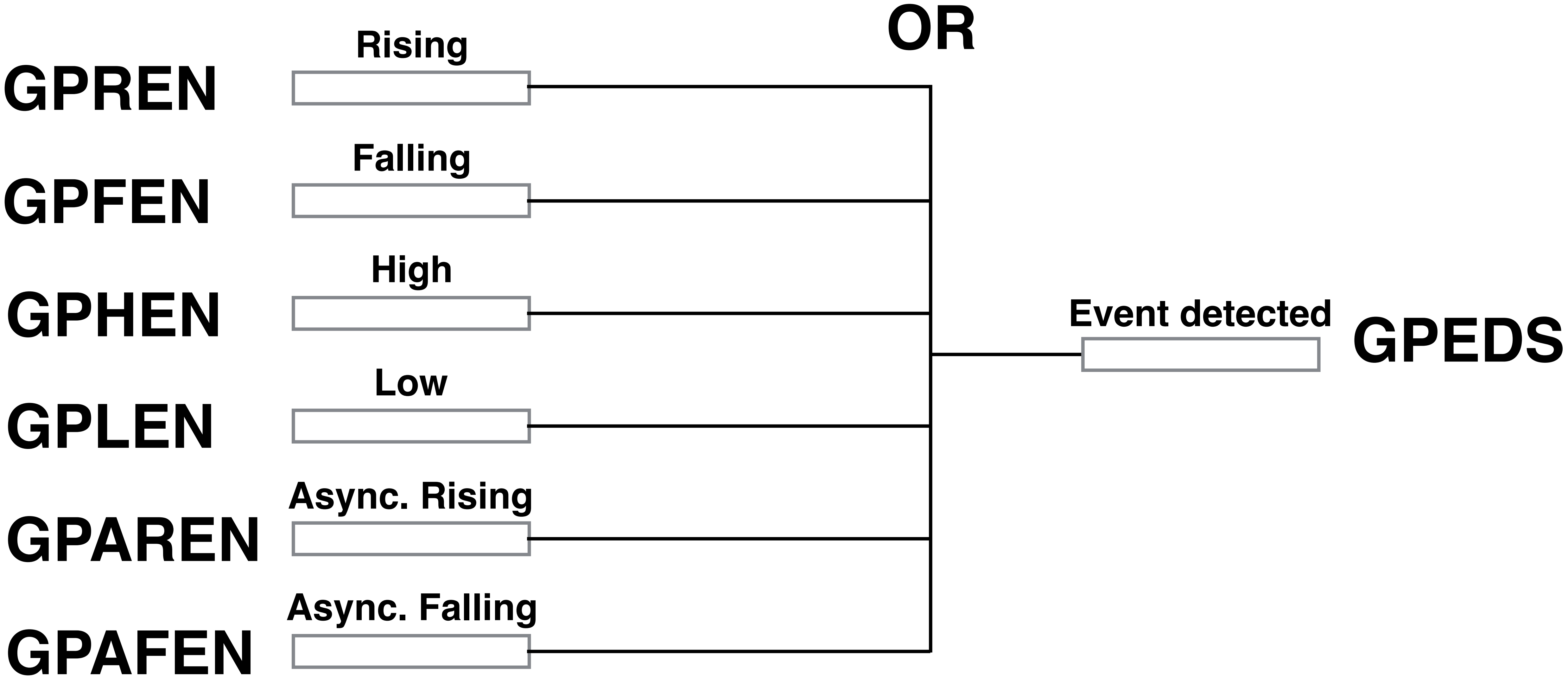
- Sometimes want the extra info (e.g. games) so interface directly with scancodes

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



OR

GPREN

Rising

GPFEN

Falling

GPHEN

High

GPLEN

Low

GPAREN

Async. Rising

GPAFEN

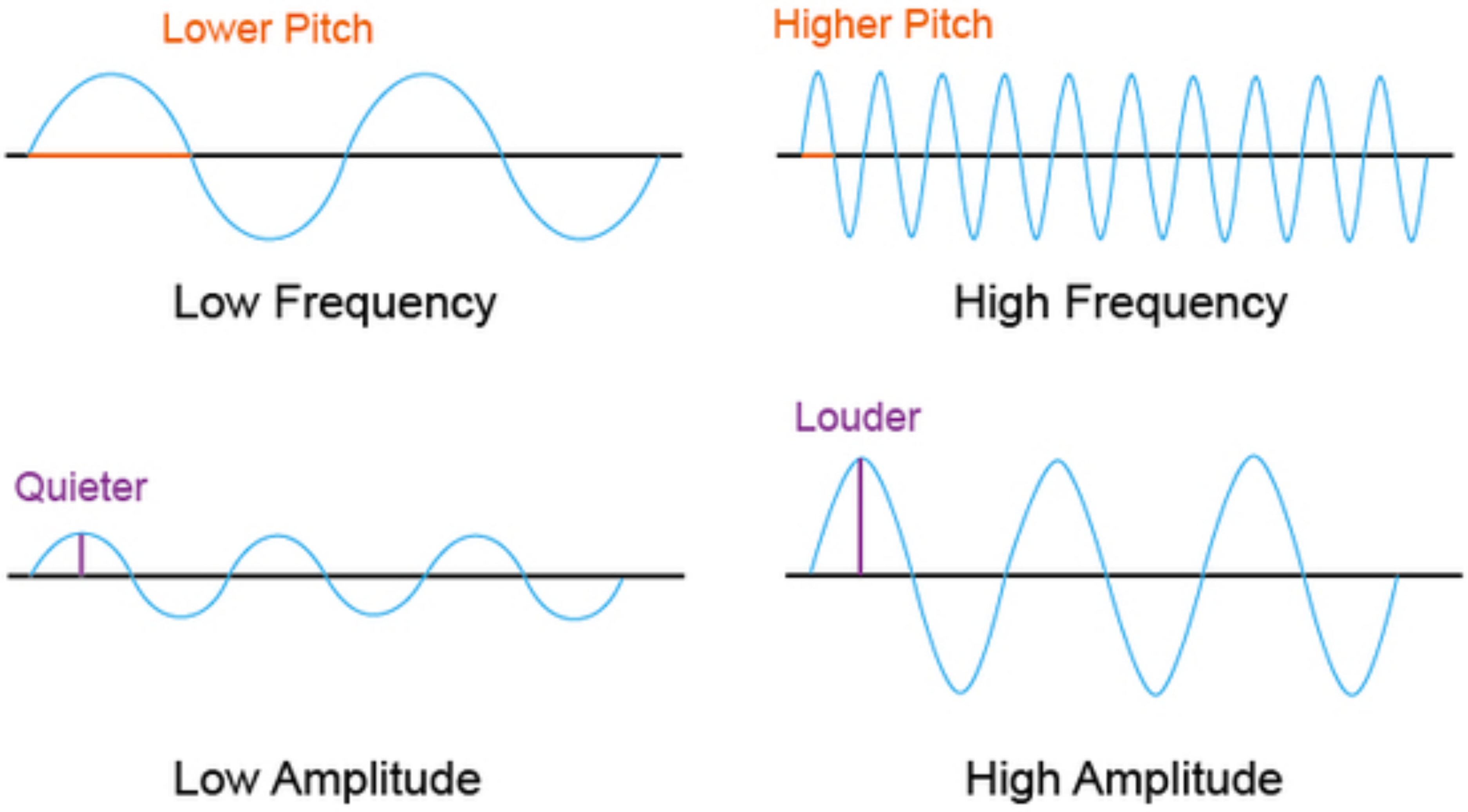
Async. Falling

Event detected

GPEDS

PWM & Sound

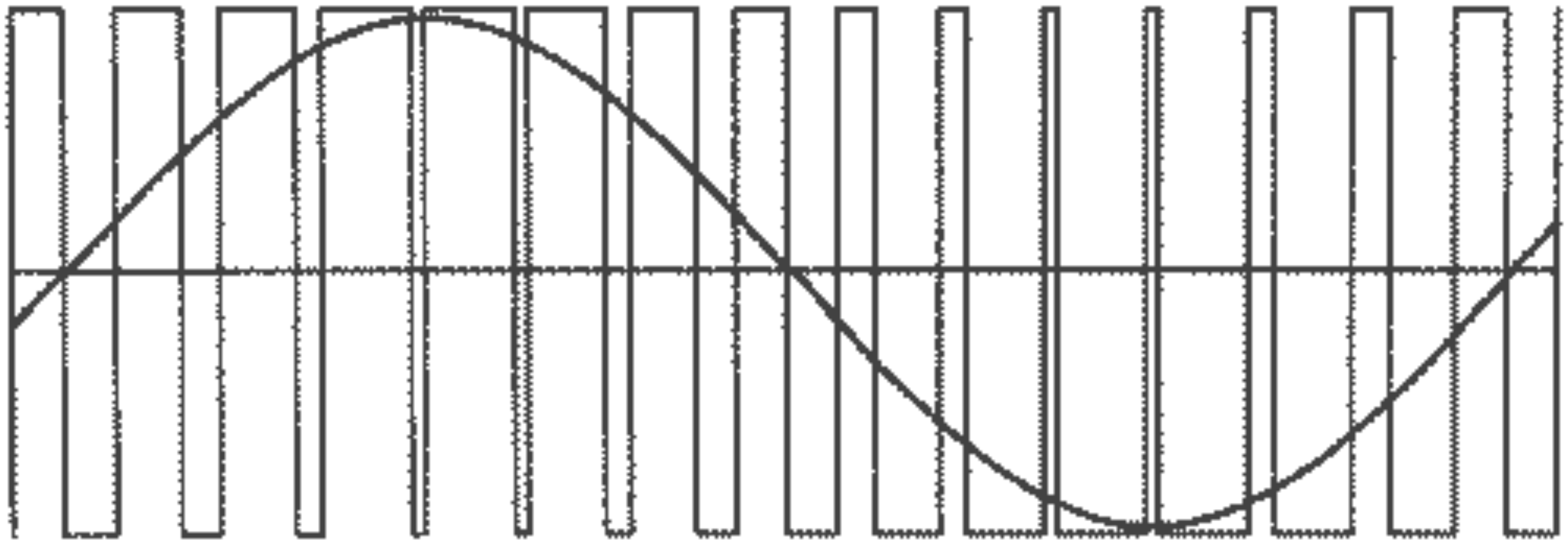
Sound Waves



(c) teachwithict.weebly.com

Pulse Width Modulation

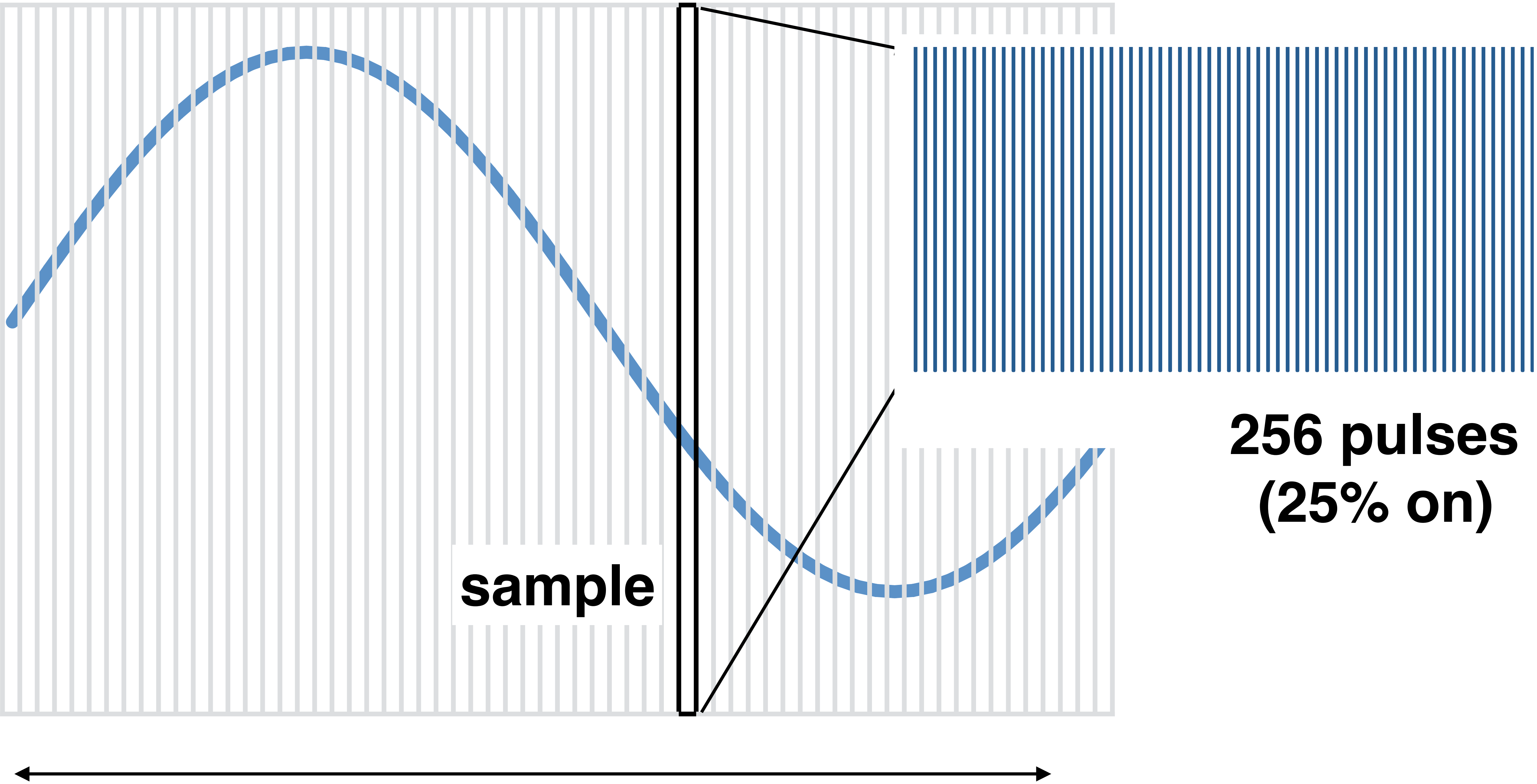
Can simulate continuous values with fast enough PWM clocking



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



sample

256 pulses
(25% on)

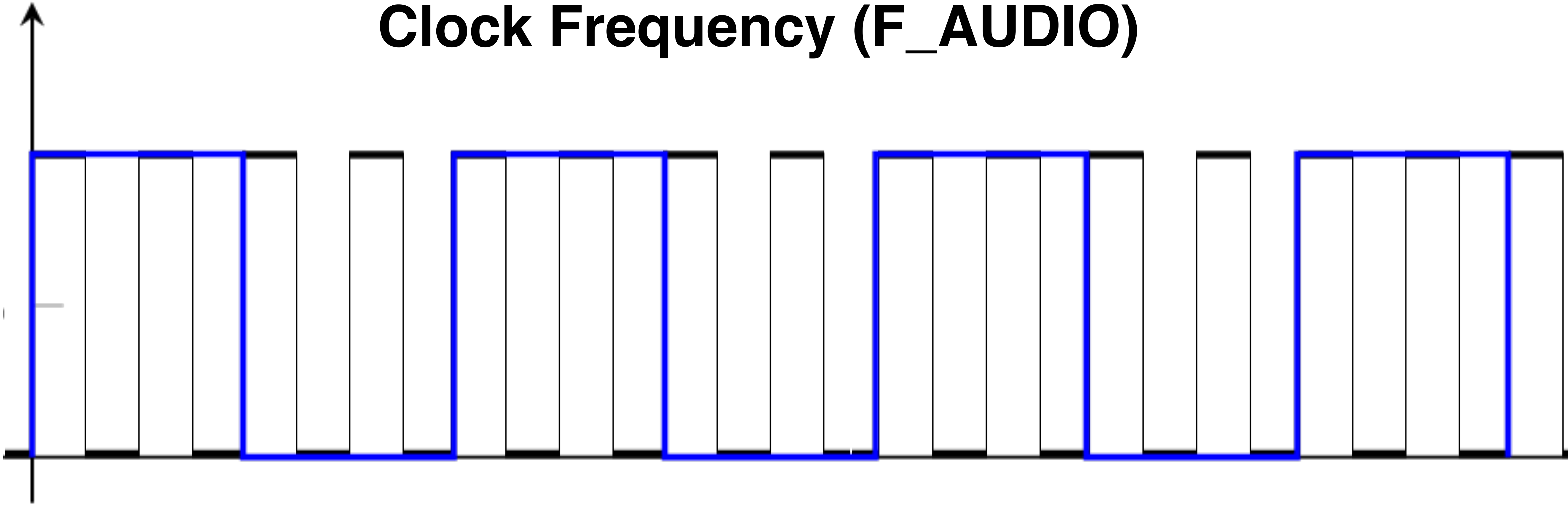
64 samples

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

Square Wave

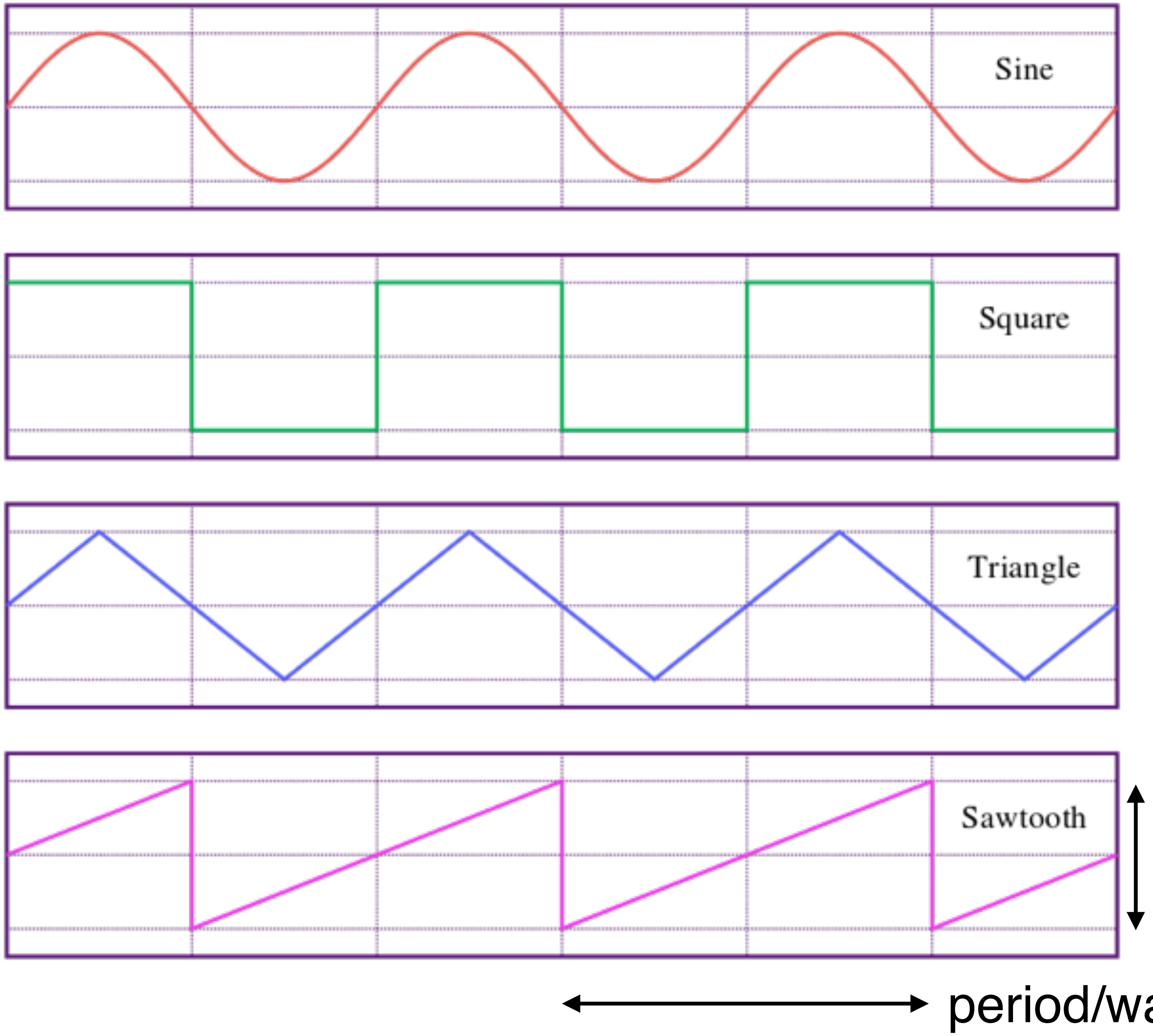
Tone Frequency

Clock Frequency (F_AUDIO)



Range = 4, Width = 2

Waveforms



amplitude

period/wavelength