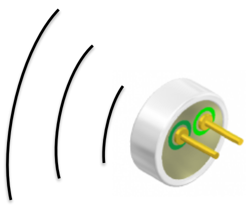
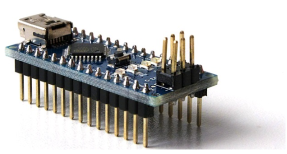
ETH Course 402-0248-00L: Electronics for Physicists II (Digital)

- **1: Setup uC tools, introduction**
- **2: Solder SMD Arduino Nano board**
- **3: Build application around ATmega328P**
- **4: Design your own PCB schematic**
- **5: Place and route your PCB**
- **6: Start logic design with FPGAs**

Exercise 3: “Sound volume robot”

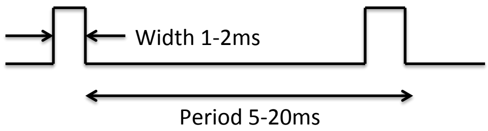
- measures sound volume and moves arm to indicate loudness
- microphone -> preamp -> ADC -> uC -> PWM output

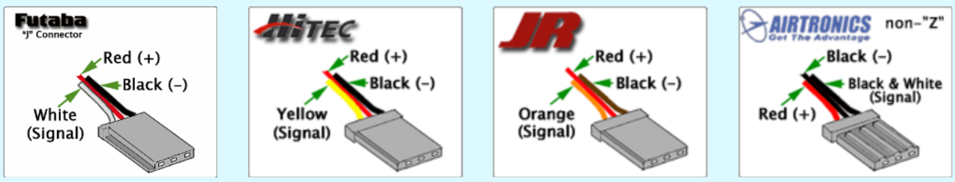



(debugging, programming)

“RC” servos (Radio-Control Servo-Motors)

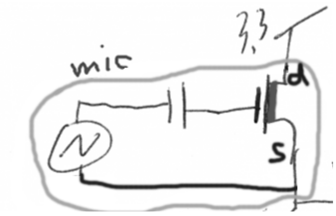
- Position controlled – Servo has internal position measurement and controller
- Rotation angle 120 degrees
- Pulse width from 1-2ms sets desired position
- Pulses must be sent at frequency 50-200Hz
- Pulse height >2V

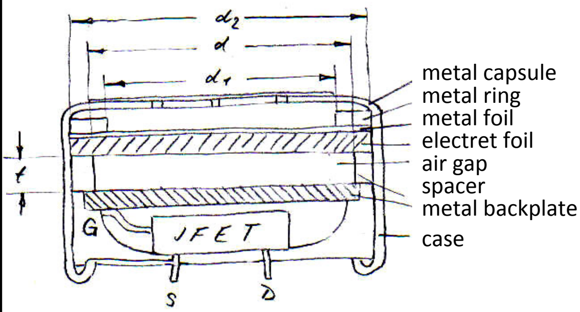


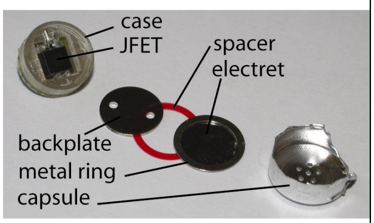


Electret Microphone

- Cheap (< 1$)
- Electret material, no polarization voltage is required
- Low-noise JFET buffer
- Metal foil is connected to source of the JFET through metal capsule

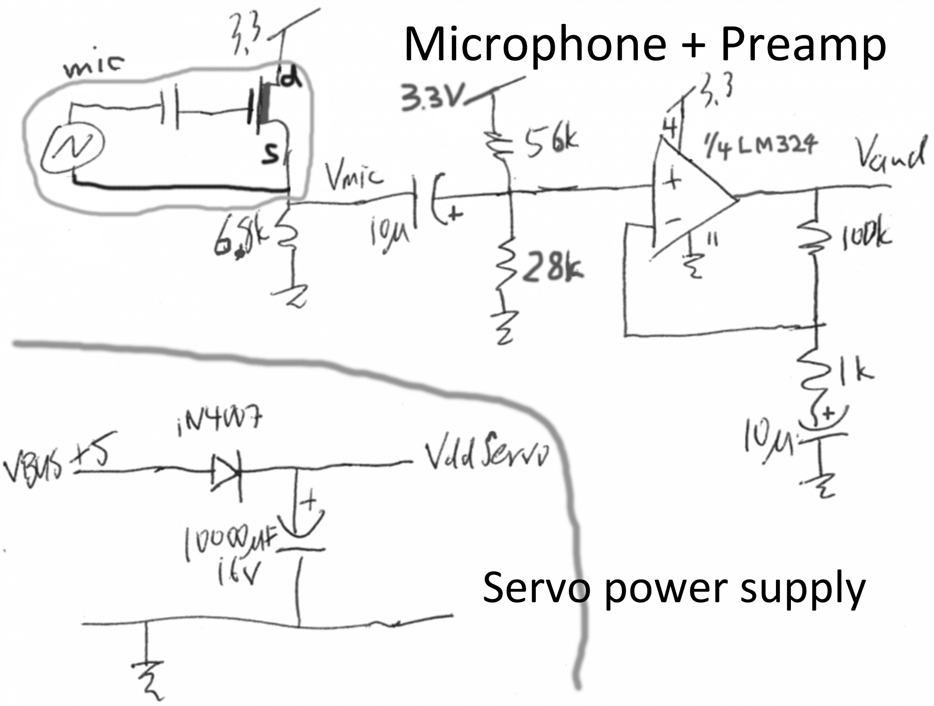






4

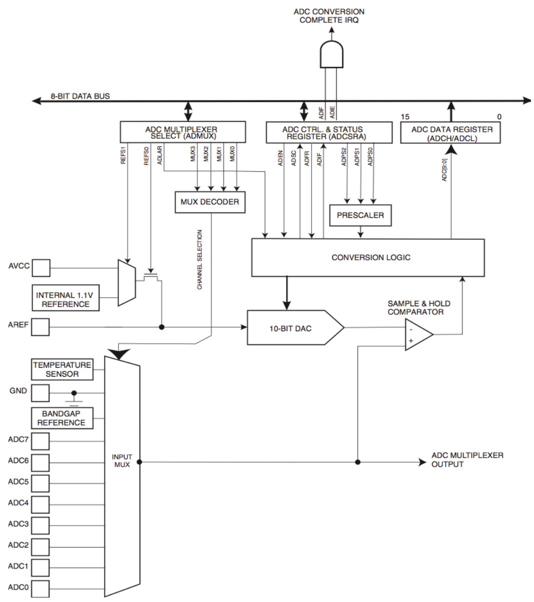
Microphone + Preamp

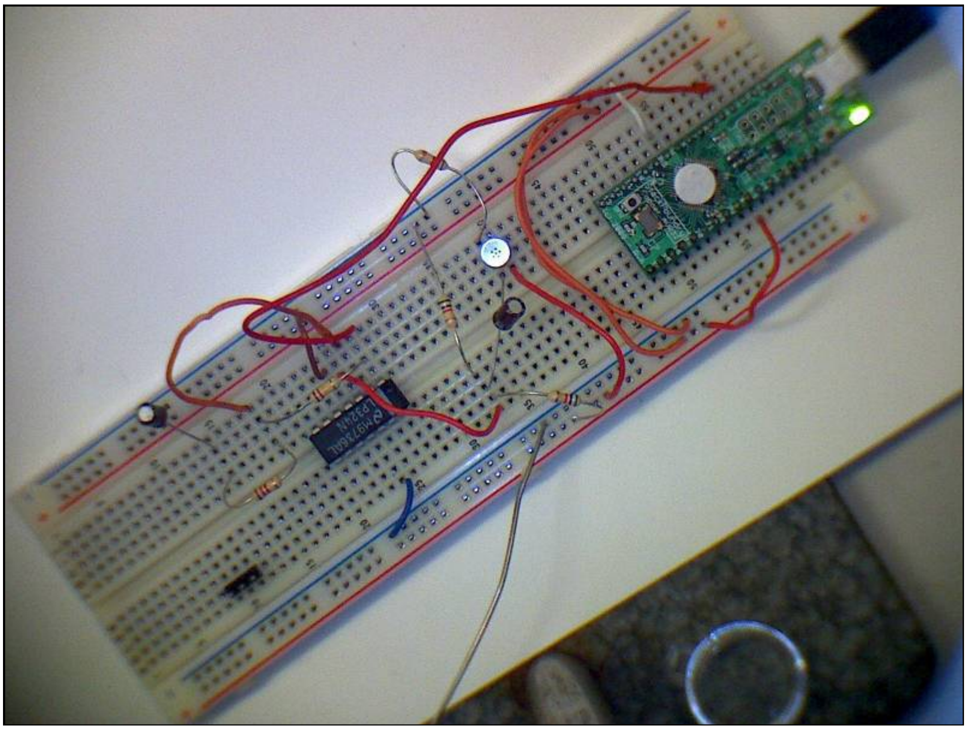


Servo power supply

ATmega328P Analog to Digital converter

- 10-bit Successive approximation register (SAR) type
- 8 multiplexed single-ended input channels
- Internal Temp sensor
- Max combined sample rate 79.6ks/s
- Interrupt on End of Conversion.
- Triggered by:
  - External Interrupt Request 0
  - Timer 0
  - Timer 1
  - Analog Comparator





- Fixed-point digital signal processing pipeline
- Using timer interrupts for regular ADC sampling intervals

Signal processing pipeline

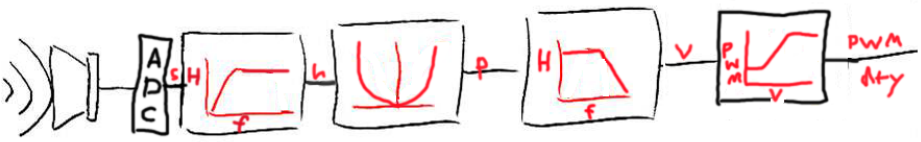
produces servo position corresponding to average sound volume

Mean removal
100Hz
High pass

Rectification
Square or absolute value

Smoothing
0.5Hz
lowpass

Servo Command
Normalizer+clipper



Mic

Some more about ADCs

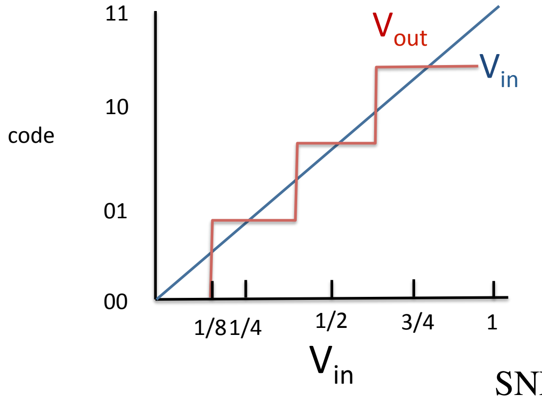
High resolution Low speed and power	Medium resolution Medium power	Low resolution but fast and hot
Single slope (imprecise)	SAR (good tradeoffs, most uC)	Flash (video rate, oscilloscopes)
Dual slope (precise but very slow)	Algorithmic (ΣΔ)	2-step

ADC specifications

INL	**Integral nonlinearity**	**Max absolute sample deviation in bits**
DNL	Differential nonlinearity	Max possible step size variation in bits
Sample rate		
Latency	In samples	How long in samples it takes for a conversion (can be >>1 for pipelined converter)
Reference voltage	Volts	Minimum resolution

"Quantization noise"

2-bit converter



$$V_Q = V_{out} - V_{in}$$

$$\overline{V_Q^{\,2}} = \frac{V_{LSB}^{\,2}}{12}$$

$$V_{QRMS} = \frac{V_{LSB}}{3.5}$$

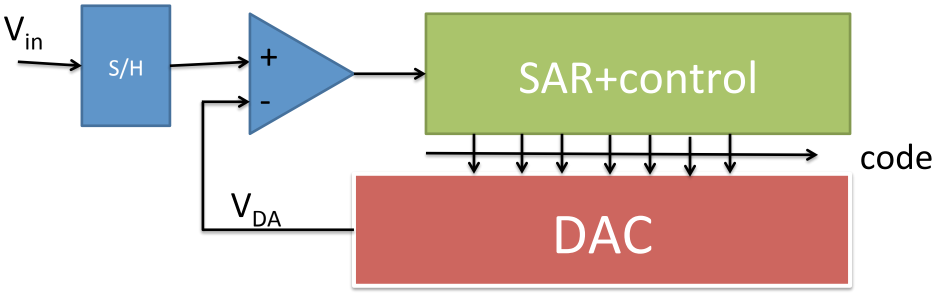
Max possible SNR? (Signal power/Noise power).
For uniformly distributed signal like a sawtooth, we get

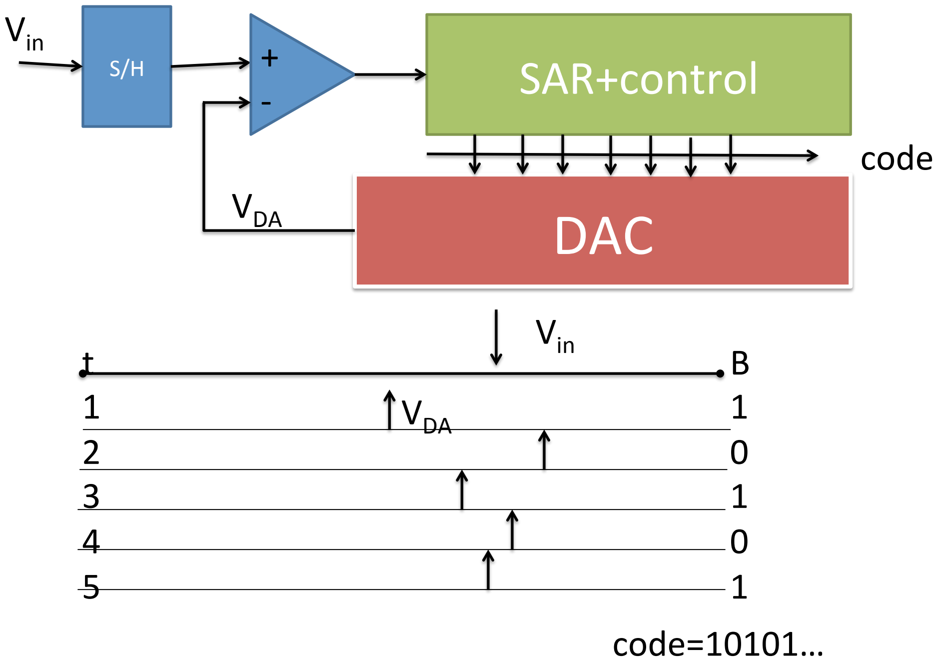
$$\text{SNR}=\left(\frac{V_{REF}}{12}\Big/\frac{V_{LSB}}{12}\right)^2 = 2^N$$

$$= 20\log_{10} 2^N \text{dB} = 6N \text{ dB}$$

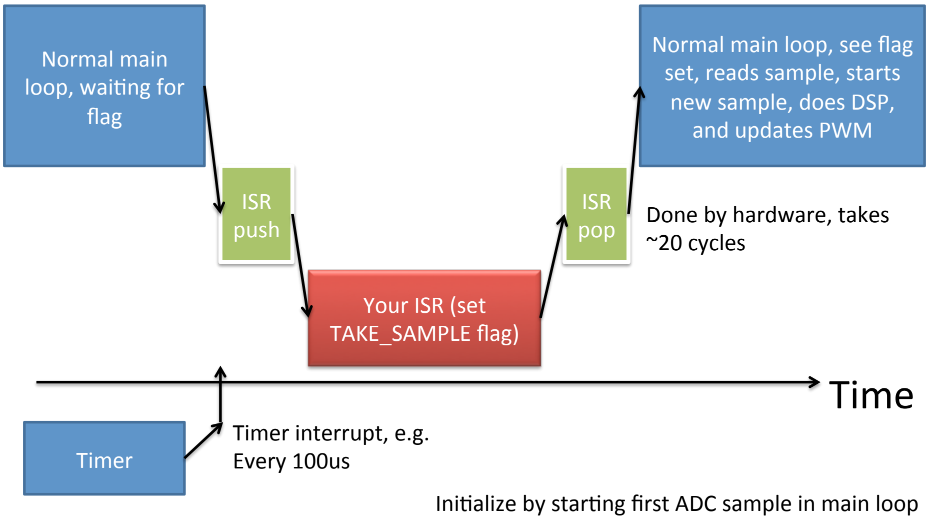
e.g. for N=10, SNR=60dB

Successive Approximate Register (SAR) ADC





Using timer interrupts for regular ADC sampling intervals in an Interrupt Service Routine (ISR)



ISR

```
void tc_irq(void) {
  // Increment the counter, which is also used to determine servo updates
  tc_tick++;

  // set a flag to tell main loop to take a sample
  takeSampleNow = TRUE;

  // Toggle a GPIO pin (this pin is used as a regular GPIO pin).
  digitalWrite(13,!digitalRead(13)); // debug, should toggle at desired sample rate
}
```

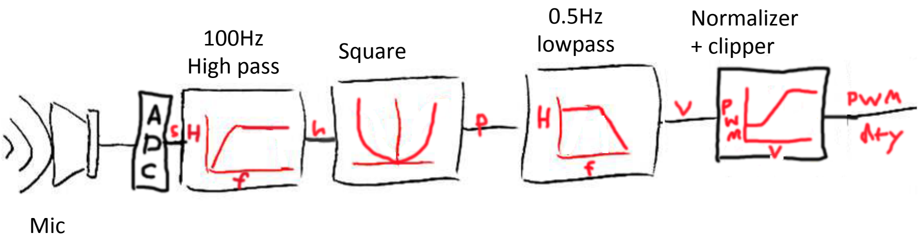
Timer Counter (TC) setup

- Download MsTimer2.zip and unzip in your Arduino/libraries folder.
- Add #include <MsTimer2.h> at the beginning.
- Setup(): Add the following lines:

```
MsTimer2::set(time in us,t2_ovf);
MsTimer2::start();
```

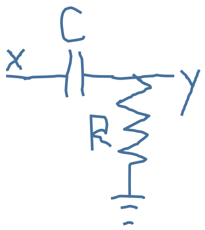
- From now, for each Timer2 overflows, t2_ovf() will be executed. You need to declare and write code for t2_ovf() function.

Fixed point signal processing pipeline



We need a digital low & high pass filters, like an RC or CR filter

A simple IIR high pass filter (discrete time)



$$\frac{y}{R} = C\left(\dot{x} - \dot{y}\right)$$

$$RC\dot{y} + y = RC\dot{x}$$

$$\tau\dot{y} + y = \tau\dot{x}$$

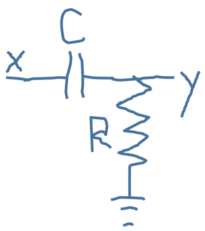
$$\tau\left(\frac{y_{t+\delta t} - y_t}{\delta t}\right) + y_t = \tau\left(\frac{x_{t+\delta t} - x_t}{\delta t}\right)$$

$$\alpha = \frac{\delta t}{\tau}$$

$$\begin{aligned} y_{t+\delta t} &= y_t - \alpha y_t + x_{t+\delta t} - x_t \\ &= (1-\alpha)y_t + x_{t+\delta t} - x_t \end{aligned}$$

A simple IIR high pass digital filter

(fixed point, using binary shift operations)



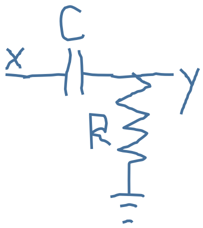
$$y_{t+\delta t} = (1-\alpha)y_t + x_{t+\delta t} - x_t$$

If $\alpha = \frac{1}{2^n}$, then

$$(1-\alpha)y_t = \frac{2^n - 1}{2^n} y_t = \left[\left(y_t \ll n\right) - y_t\right] \gg n$$

$$y_{t+\delta t} = \left[\left(y_t \ll n\right) - y_t\right] \gg n + \left(x_{t+\delta t} - x_t\right)$$

What is the time constant?



$\alpha = \frac{\delta t}{\tau}$

Suppose $\delta t = 100$us (10kHz sample rate)

and $\alpha = 1/256$ (n=8).

Then

$\tau = 100$us x 256=25.6ms

Corner frequency $f_{3dB} = \frac{1}{2\pi\tau} = 6.2Hz$

To filter with n times longer time constant, you can skip n samples

DSP code sample

```
void device_task(void) {

if (takeSampleNow) {  // flag set in timer ISR
takeSampleNow=FALSE;
// signal processing
int adcval = analogRead(apin); // 0-1023=5V

if (initialized)
        audMean = ((adcval-audMean)>>NTAU1)+audMean; // TODO mix old and new value
else
        audMean = adcval; // init filter with first reading

// only update meanSq at TAU2 interval, so to produce effective time constant that
is TAU2 times tau of audMean filtering
if(dspCounter--==0){
    dspCounter=TAU2;
    long diff = adcval - audMean; // signed diff of sample from mean
    long sq = diff * diff; // square diff
    if (initialized)
        meanSq = ((sq-meanSq)>>NTAU1)+meanSq; // low pass square diff
    else
        meanSq = sq;
}
}
}
```

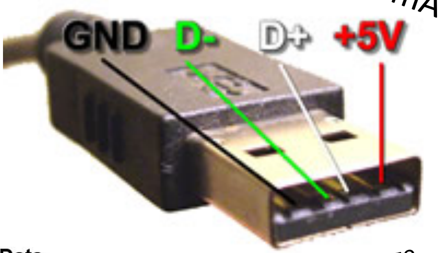
USB – Universal Serial Bus

- Physical layer
- User perspective (coder)
- Under the hood
  - Device side
  - Host side
- Achieving high performance

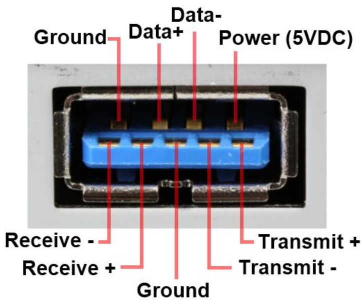
USB Physical layer

Up to 200mA

- Up to USB 2.0 – full (12Mbps) and high (480Mbps) speed
- USB 3.0 super speed (5Ggbs)



Low voltage differential signal



USB definitions

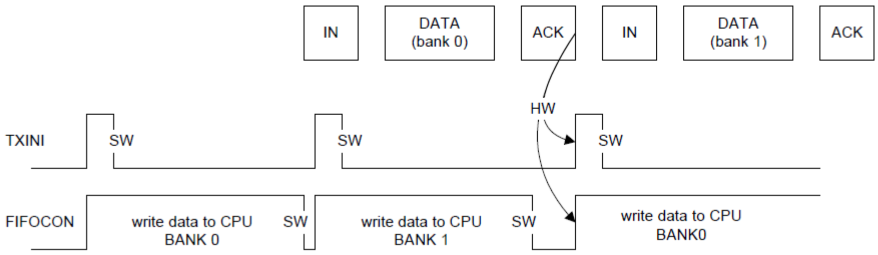
- IN means towards the host (the PC)
- OUT means towards the device (uC)

Endpoints – multiple virtual channels

Pipe/Endpoint	Mnemonic	Max. Size	Max. Nb. Banks	DMA	Type
0	PEP0	64 bytes	1	N	Control
1	PEP1	64 bytes	2	Y	Isochronous/Bulk/Interr
2	PEP2	64 bytes	2	Y	Isochronous/Bulk/Interr
3	PEP3	64 bytes	2	Y	Isochronous/Bulk/Interr
4	PEP4	64 bytes	2	Y	Isochronous/Bulk/Interr
5	PEP5	256 bytes	2	Y	Isochronous/Bulk/Interr
6	PEP6	256 bytes	2	Y	Isochronous/Bulk/Interr

Can be double buffered

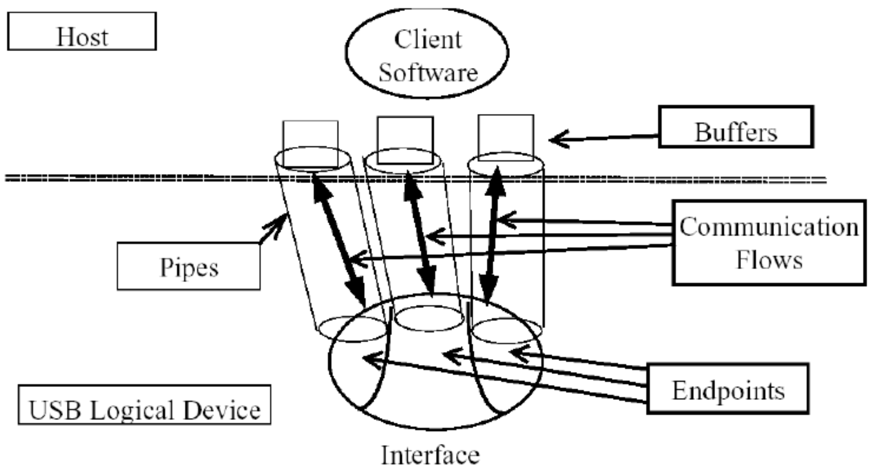
Double-buffered transfers can increase continuity



- When the bank is empty, TXINI and FIFOCON are set, what triggers an EPnINT interrupt if TXINE is one.
- The user acknowledges the interrupt by clearing TXINI.
- The user writes the data into the current bank by using the USB Pipe/Endpoint nFIFO Data virtual segment (see "USB Pipe/Endpoint n FIFO Data Register (USBFIFOnDATA)" on page 483), until all the data frame is written or the bank is full (in which case RWALL is cleared and the Byte Count (BYCT) field in UESTAn reaches the endpoint size).
- The user allows the controller to send the bank and switches to the next bank (if any) by clearing FIFOCON.

Host vs. Device

For the USBB in host mode, the term "pipe" is used instead of "endpoint" (used in device mode).
A host pipe corresponds to a device endpoint



The key to high performance on host side: *Asynchronous* or *Overlapped* IO

- On the host side, an Input-Output (IO) thread manages the USB IO.
- Multiple buffers (which can be much larger than the device FIFO size) are submitted to the USB driver / host controller to be filled by the USB controller.

1. When a buffer is filled, the IO thread is notified asynchronously, which wakes it up.
2. The IO thread processes the buffer, and then gives it back to the controller. The IO thread then notifies the main user code that data is available, e.g. by writing to a software queue.

- That way, the user doesn't *block* waiting for data
- Our *pyusb* example doesn't do this yet

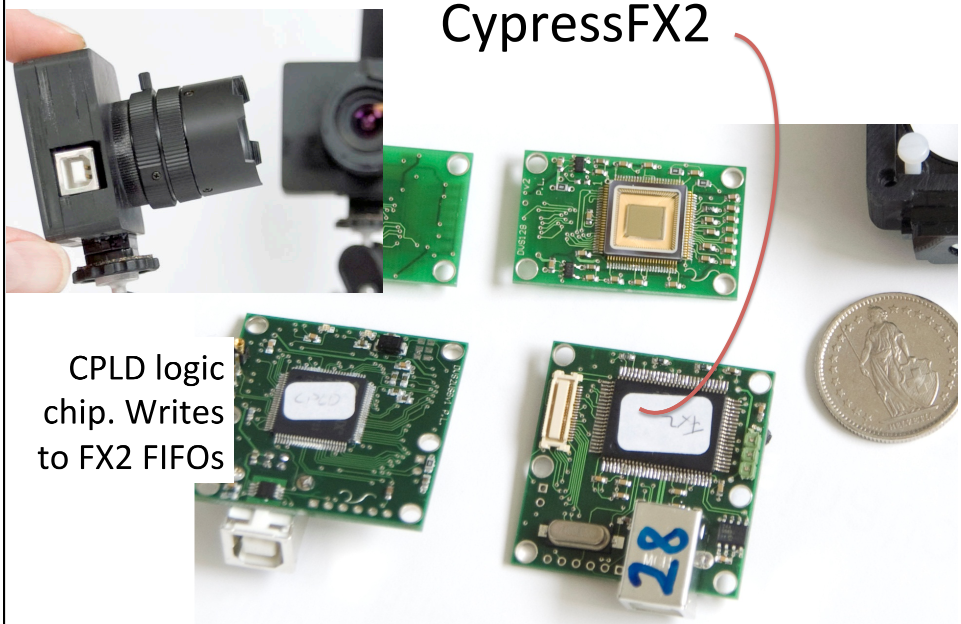
USB performance

- USB full speed (12Mbps): about 1MBps
- USB high speed (480Mbps): about 40MBps
- USB super speed (5Gbps): ??

ICs for USB

- USB full speed
  - Many uC. Also FTDI.
- USB high speed
  - CypressFX2
- USB super speed
  - CypressFX3

CypressFX2



CypressFX3

Cypress EZ-USB® FX3™ is the next-generation SuperSpeed USB 3.0 peripheral controller that enables developers to add USB 3.0 device functionality to any system.

EZ-USB FX3 has a fully configurable, General Programmable Interface (GPIF™ II) that can interface with any processor, ASIC, image sensor, or FPGA. GPIF™ II is an enhanced version of the original GPIF™ in FX2LP, Cypress's flagship USB 2.0 product. It provides easy and glue-less connectivity to popular industry interfaces such as

ftdichip.com

- uC UART – USB interface; looks like COM serial port on host side.
- Max speed is only 12Mbaud for the UART port unfortunately