BILL ASHMANSKAS

PHYSICS 364 2010-11-29 : ANALOG ↔ DIGITAL CONVERSION

It is often necessary to convert the continuous signals of the real world into bits stored inside a computer or other digital circuit — or vice versa.

For instance, one may want to

- record hourly temperature, pressure, and wind data at a weather station and relay those data to remote forecasters
- record music or speech for Internet transmission
- record x-ray photon interaction energies, positions, etc., for CT or PET image reconstruction
- collect data from a physics experiment for computer analysis
- synthesize a sequence of tones to turn sheet music into audible music or textual phonemes into audible speech
- control a physical process via computer
- implement an "analog" filter that would be difficult or impossible using resistors, capacitors, opamps, etc.
- many, many more examples. Digital handling of physical signals is ubiquitous today.

Nyquist-Shannon sampling theorem:

> A function $x(t)$ containing no frequencies $f \geq B$ hertz is completely determined by giving its ordinates at a sequence of points $\left(\frac{1}{2B}\right)$ seconds apart.

Hence, for a system that samples a signal at interval T ~~seconds~~, the largest frequency that can be resolved is known as the "Nyquist frequency"

$$f_{Nyquist} = \frac{1}{2T}$$

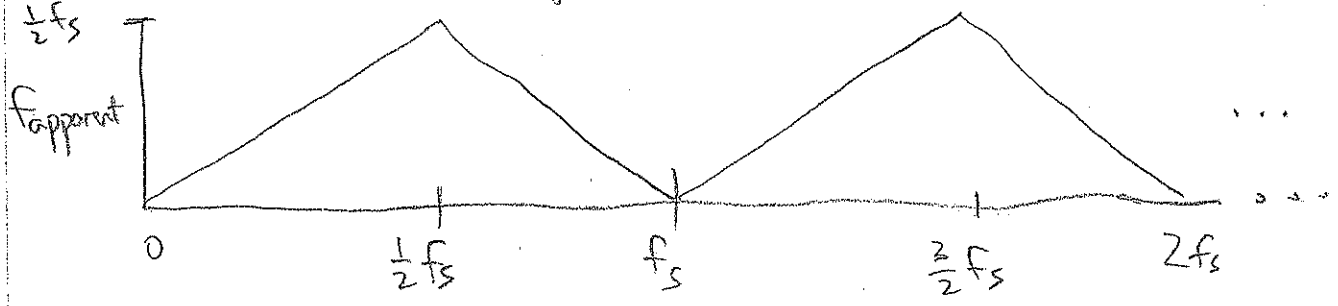
For a signal of bandwidth B, sometimes one refers to the "Nyquist sampling rate"

$$f_{sample} = 2B$$

which is a lower bound on the sampling rate needed to record the signal unambiguously.

For example, the upper range of human hearing is around 20 KHz (substantially lower as we age), so digital audio (for CD, MP3, etc.) is normally sampled at 44.1 KHz.

Frequencies above $f_{Nyquist} = \frac{1}{2} f_{sample}$ are "aliased" to lower apparent frequencies:



Thus, sampling is normally preceded by an "anti-aliasing" lowpass filter to ensure that any signal component at frequency $f \geq f_{Nyquist}$ is attenuated to an amplitude too small to be detected.

Since no realizable filter has a perfectly sharp cutoff, one normally "oversamples" by some factor $f_{sample}/2B$.

So far we have described quantization of a signal $v(t)$ at discrete time points t_i. To represent $v(t)$ inside a digital computer, both ordinate (v) and abscissa (t) must be quantized.

We quantify discrete time steps as "samples per second" and discrete amplitude steps as "bits per sample" and either "least significant bit" (LSB) or "full scale" = LSB · 2^{bps}

If we record 10 bits per sample with a 1mV LSB, the full-scale range is $2^{10} \cdot 1\text{mV} = 1.024\text{V}$.

A typical computer audio recording (e.g. .WAV format) samples at 44.1 KHz, 16 bits per sample, for each of two stereo channels. So one hour of music requires

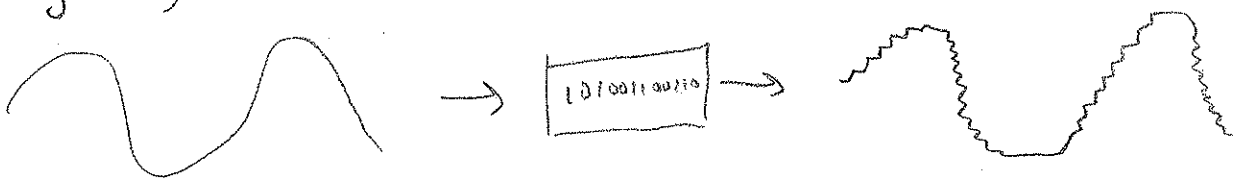
$$3600\text{s} \times \frac{44.1\ \text{KSPS} \times 16\ \text{bps} \times 2\ \text{channels}}{8\ \text{bits/byte}} = 635 \times 10^6\ \text{bytes}$$

$$= 606\ \text{MB}$$

Recall that a "70 minute" CD-R holds 650 MB of data.

Information

So far, we have considered the effects of quantization on a digital reproduction of an analog signal



Now consider the complementary problem: what sort of analog medium do you need in order to communicate N bits per second of digital data?

Shannon's Law states that the tightest upper bound on information rate transmitted on an analog channel is the "channel capacity"

$$C = B \log_2\left(1 + \frac{S}{N}\right) \qquad \text{(in bits per secon}$$

For instance $C = 16 \times 44100$ bps, $B = 20$ kHz

$\Rightarrow \sqrt{1+\frac{S}{N}}$ must exceed 2×10^5

or rms noise $\lesssim \frac{LSb}{3.1}$

(note S/N above is power ratio, not voltage ratio)

Or without oversampling,

$C = 16 \times 44100$ bps, $B = 22050$ Hz

$\Rightarrow \sqrt{1+\frac{S}{N}}$ must exceed 65536

or rms noise < 1 LSb

So **information capacity $\propto$ bandwidth $\times \log(S/N)$**

See Shannon 1948 "A Mathematical Theory of Communication"

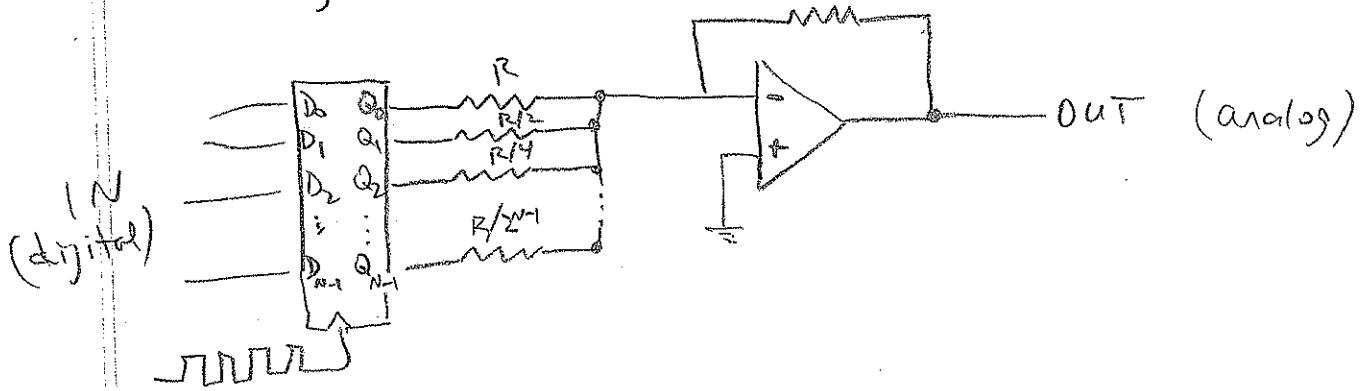
In practice, usable bandwidth of a transmission line is limited by dispersion (phase velocity varies with frequency), skin effect losses, dielectric losses, and cutoff frequencies for unwanted propagation modes.

"Noise" technically refers to thermal excitation, e.g. $V_{rms}^2/R \sim k_B T \cdot B$,

but may also refer to interference from unwanted signals. Jose will discuss noise and interference in next week's lecture.

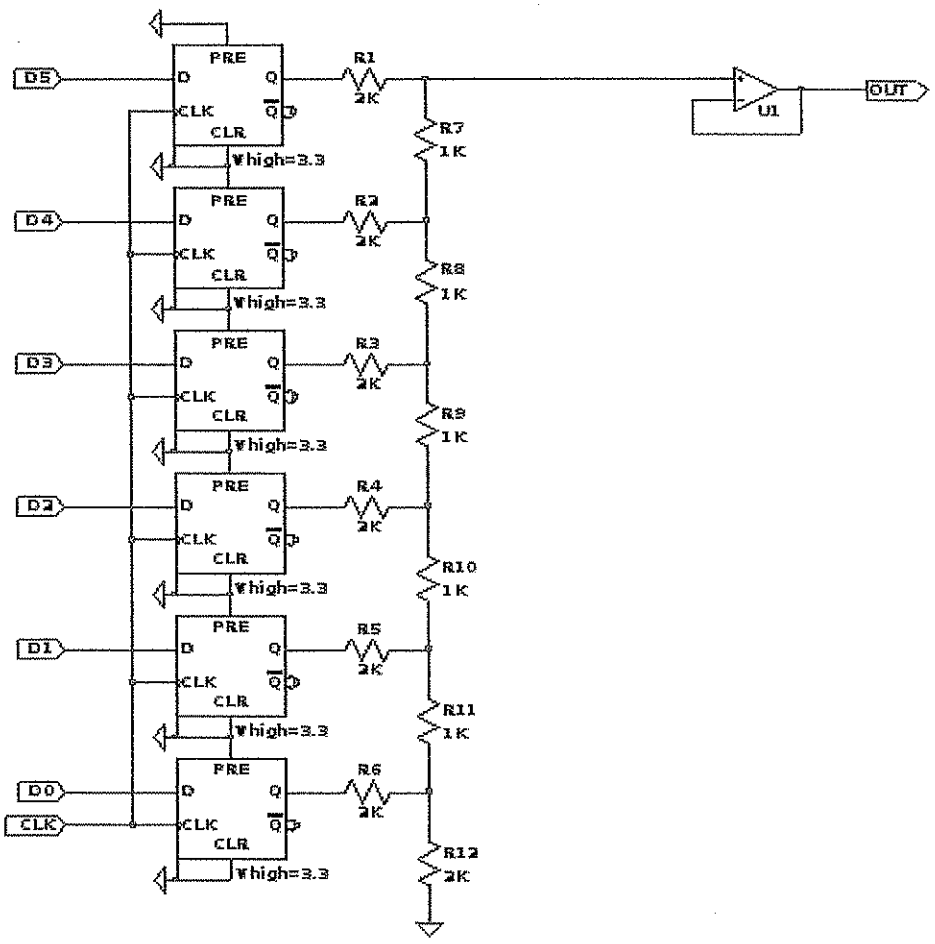
Digital-To-Analog Conversion

The most conceptually simple DAC design looks something like this:



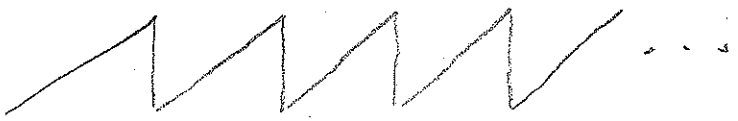
But it is difficult in practice to find precision resistors spanning many powers of 2, e.g. 1K, 2K, 4K, 8K, 16K, 32K, 64K, ...

So instead the "R-2R ladder" is much more common:



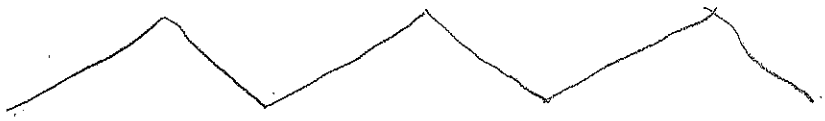
We'll build this R-2R ladder DAC circuit (6 bits) in lab this week.

Note that if you drive the DAC with the output of a 6-bit counter, you get a sawtooth wave output:



A modified 7-bit counter will give you a triangle wave:

```
assign dac[5:0] =
        count[6] ? (63 - count[5:0])
                 : count[5:0] ;
```



If you have a large enough FPGA to store the sine function in a lookup table, then you can easily play a pure tone with the DAC in place of the triangle. But let's stick with the triangle for now because it is easier to implement.

(Note that even with very limited resources, you can use a neat rectangular ⇔ polar conversion algorithm called CORDIC to calculate sine values in an FPGA.

See en.wikipedia.org/wiki/CORDIC for details.)

You can easily make a simple synthesizer with an FPGA and a DAC, by replacing the counter above with a "phase accumulator"

```
wire [5:0] freq;   ← input from DIP switches
wire [9:0] accum;

dffe_Nbit #(10) accumff(.clk(clk), .ena(1),
                        .q(accum),
                        .d(accum + freq));

wire [6:0] count = accum[9:3];

assign dac[5:0] = .
        count[6] ? (63 - count[5:0])
                 : count[5:0];
```

So the DIP switches program how quickly the "phase" increases, hence what frequency triangle wave is synthesized by the DAC.

You will try this in lab this week, too.

Analog to Digital Conversion

Analog-to-Digital Conversion

There are 3 basic styles of ADC, though many variations exist. An m-bit ADC turns an analog sample into one of $N = 2^m$ possible digital codes

$$\text{code} = \left\lfloor V_{in} \cdot \frac{N}{V_{full\ scale}} \right\rfloor$$

← $\lfloor$floor$\rfloor$ operation

For a 6-bit ADC with $V_{full\text{-}scale} = 1\ V$,

code	V_{in}
0	0 V
1	.015625 V
2	.03125 V
3	.0469 V
4	.0625 V
5	.0781 V
...	...
63	.9844 V

A "ramp compare" ADC uses a linear search to find the right code, so it has conversion time $\propto N$. (a.k.a. linear slope, dual slope, wilkinson, ...)

A "successive approximation" ADC uses a binary search to find the right code, so it has conversion time $\propto \log_2(N)$.

A "flash" ADC uses a direct lookup (like a hash table in computer science), so it has conversion time $\propto 1$, i.e. largely independent of N.

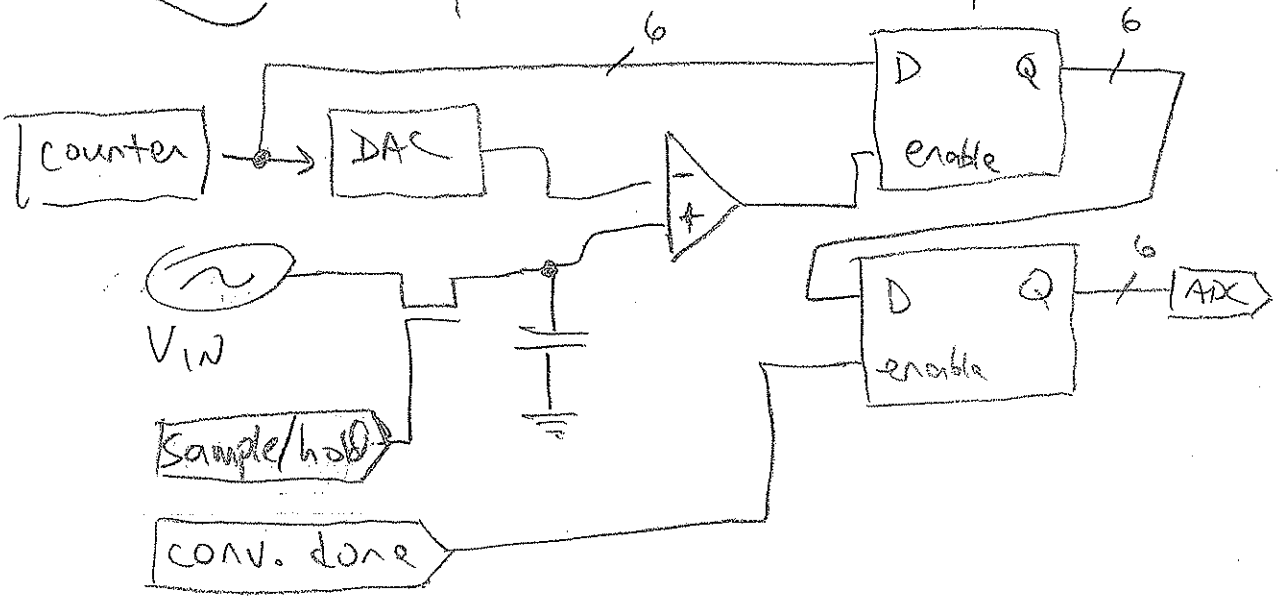
Let's see how this is done.

~ 1950
("Wilkinson" ADC)

example (built in lab)

Single-slope ramp-compare 6-bit ADC

- count from 0 to 63
- play value of counter to DAC
- use comparator to see when sampled voltage exceeds DAC output
- store value of counter where comparator transitions 1 → 0.
- when ramp is complete:
  - update ADC output value from stored counter
  - strobe "conversion done" line
  - close FET switch on sample & hold, to acquire a new sample



- lab12_adc_linear.v = source code
- lab12_adc_linear_output1.txt = detailed test bench output
- lab12_adc_linear_output2.txt = TB output showing only conv-done cycles

```
/*
 * lab12_adc_linear.v
 * demonstrate single-slope ADC operation for physics 364 week 12
 * begun 2010-11-28
 * by Bill Ashmanskas, ashmansk@hep.upenn.edu
 */


`default_nettype none
`timescale 1ns/1ns


/*
 * Implement single-slope ADC
 */
module adc (
    input wire        clk,              // 68 kHz clock, for 1 kHz sample rate
    input wire        cmp,              // 1 if sample > DAC value, else 0
    output wire [5:0] dac,              // DAC used for comparing with sample
    output wire [5:0] adc,              // ADC output value
    output wire       conv_done,        // indicate next ADC value is ready
    output wire       sample_hold       // closes FET switch for sample & hold
);
    // modulo-68 counter: ramp 0..63, then allow 4 extra ticks for processing
    wire [6:0] count;
    wire [6:0] count_next = (count==67) ? 0 : count+1;
    dffe_Nbit #(7) count_ff (.clk(clk), .ena(1), .d(count_next), .q(count));
    // ramp the DAC output during the 0..63 phase of counter
    assign dac = count[6] ? 0 : count[5:0];
    // new ADC value will be the largest DAC value for which sample>DAC
    wire [5:0] newadc;
    dffe_Nbit #(6) newadc_ff (.clk(clk), .ena(cmp), .d(dac), .q(newadc));
    // when 0..63 ramp is complete, copy newadc into ADC output register
    dffe_Nbit #(6) adc_ff (.clk(clk), .ena(count==64), .d(newadc), .q(adc));
    // when ADC output register has updated, strobe "conversion done" signal
    assign conv_done = (count==65);
    // before new ramp begins, tell sample & hold to get new sample
    assign sample_hold = (count==66);
endmodule
```

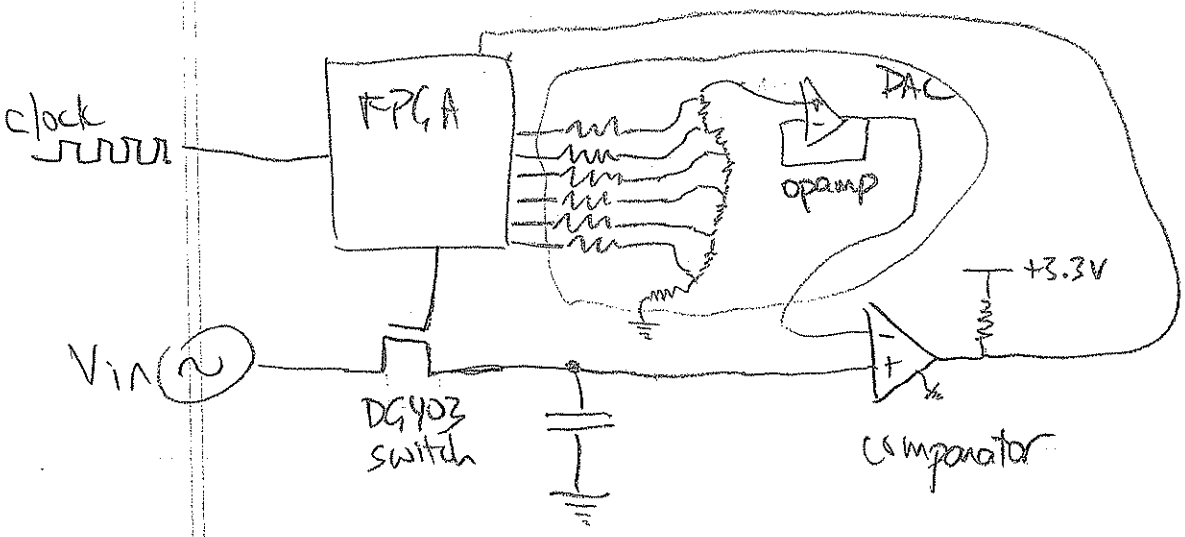
```
/*
 * Test bench for ADC
 */
module adc_tb;
    reg clk=0, cmp=0;
    wire [5:0] adc, dac;
    wire       conv_done, sample_hold;
    adc uut (.clk(clk), .cmp(cmp), .adc(adc), .dac(dac),
             .conv_done(conv_done), .sample_hold(sample_hold));
    parameter period=1000000/68;
    initial begin
        #200;
        forever begin
            #(period/2);
            clk = 1;
            #(period/2);
            clk = 0;
        end
    end
    reg [5:0] sample = 0;
    always @(*) begin
        cmp = (sample>=dac);
        if (sample_hold) sample = sample+13;
    end
    always @ (negedge clk) begin
        $display("t=%ld done=%ld sample=%ld dac=%ld cmp=%ld adc=%ld",
                 $time/period, conv_done, sample, dac, cmp, adc);
    end

    initial begin
        #(65*1000000);
        $finish;
    end
endmodule


/*
 * Implement N-bit-wide D-type flip-flop, with enable
 */
module dffe_Nbit #(parameter N=1)
(
    input  wire         clk,
    input  wire         ena,
    input  wire [N-1:0] d,
    output wire [N-1:0] q
);
    reg [N-1:0]         qreg=0;
    always @ (posedge clk) begin
        if (ena) qreg <= d;
    end
    assign q = qreg;
endmodule
```

```
t=1 done=0 sample=0 dac=1 cmp=0 adc=0
t=2 done=0 sample=0 dac=2 cmp=0 adc=0
t=3 done=0 sample=0 dac=3 cmp=0 adc=0
t=4 done=0 sample=0 dac=4 cmp=0 adc=0
t=5 done=0 sample=0 dac=5 cmp=0 adc=0
t=6 done=0 sample=0 dac=6 cmp=0 adc=0
t=7 done=0 sample=0 dac=7 cmp=0 adc=0
t=8 done=0 sample=0 dac=8 cmp=0 adc=0
t=9 done=0 sample=0 dac=9 cmp=0 adc=0
t=10 done=0 sample=0 dac=10 cmp=0 adc=0
t=11 done=0 sample=0 dac=11 cmp=0 adc=0
t=12 done=0 sample=0 dac=12 cmp=0 adc=0
t=13 done=0 sample=0 dac=13 cmp=0 adc=0
t=14 done=0 sample=0 dac=14 cmp=0 adc=0
t=15 done=0 sample=0 dac=15 cmp=0 adc=0
t=16 done=0 sample=0 dac=16 cmp=0 adc=0
t=17 done=0 sample=0 dac=17 cmp=0 adc=0
t=18 done=0 sample=0 dac=18 cmp=0 adc=0
t=19 done=0 sample=0 dac=19 cmp=0 adc=0
t=20 done=0 sample=0 dac=20 cmp=0 adc=0
t=21 done=0 sample=0 dac=21 cmp=0 adc=0
t=22 done=0 sample=0 dac=22 cmp=0 adc=0
t=23 done=0 sample=0 dac=23 cmp=0 adc=0
t=24 done=0 sample=0 dac=24 cmp=0 adc=0
t=25 done=0 sample=0 dac=25 cmp=0 adc=0
t=26 done=0 sample=0 dac=26 cmp=0 adc=0
t=27 done=0 sample=0 dac=27 cmp=0 adc=0
t=28 done=0 sample=0 dac=28 cmp=0 adc=0
t=29 done=0 sample=0 dac=29 cmp=0 adc=0
t=30 done=0 sample=0 dac=30 cmp=0 adc=0
t=31 done=0 sample=0 dac=31 cmp=0 adc=0
t=32 done=0 sample=0 dac=32 cmp=0 adc=0
t=33 done=0 sample=0 dac=33 cmp=0 adc=0
t=34 done=0 sample=0 dac=34 cmp=0 adc=0
t=35 done=0 sample=0 dac=35 cmp=0 adc=0
t=36 done=0 sample=0 dac=36 cmp=0 adc=0
t=37 done=0 sample=0 dac=37 cmp=0 adc=0
t=38 done=0 sample=0 dac=38 cmp=0 adc=0
t=39 done=0 sample=0 dac=39 cmp=0 adc=0
t=40 done=0 sample=0 dac=40 cmp=0 adc=0
t=41 done=0 sample=0 dac=41 cmp=0 adc=0
t=42 done=0 sample=0 dac=42 cmp=0 adc=0
t=43 done=0 sample=0 dac=43 cmp=0 adc=0
t=44 done=0 sample=0 dac=44 cmp=0 adc=0
t=45 done=0 sample=0 dac=45 cmp=0 adc=0
t=46 done=0 sample=0 dac=46 cmp=0 adc=0
t=47 done=0 sample=0 dac=47 cmp=0 adc=0
t=48 done=0 sample=0 dac=48 cmp=0 adc=0
t=49 done=0 sample=0 dac=49 cmp=0 adc=0
t=50 done=0 sample=0 dac=50 cmp=0 adc=0
t=51 done=0 sample=0 dac=51 cmp=0 adc=0
t=52 done=0 sample=0 dac=52 cmp=0 adc=0
t=53 done=0 sample=0 dac=53 cmp=0 adc=0
t=54 done=0 sample=0 dac=54 cmp=0 adc=0
t=55 done=0 sample=0 dac=55 cmp=0 adc=0
t=56 done=0 sample=0 dac=56 cmp=0 adc=0
t=57 done=0 sample=0 dac=57 cmp=0 adc=0
t=58 done=0 sample=0 dac=58 cmp=0 adc=0
t=59 done=0 sample=0 dac=59 cmp=0 adc=0
t=60 done=0 sample=0 dac=60 cmp=0 adc=0
t=61 done=0 sample=0 dac=61 cmp=0 adc=0
t=62 done=0 sample=0 dac=62 cmp=0 adc=0
t=63 done=0 sample=0 dac=63 cmp=0 adc=0
t=64 done=0 sample=0 dac=0 cmp=1 adc=0
t=65 done=1 sample=0 dac=0 cmp=1 adc=0
t=66 done=0 sample=13 dac=0 cmp=1 adc=0
t=67 done=0 sample=13 dac=0 cmp=1 adc=0
t=68 done=0 sample=13 dac=0 cmp=1 adc=0
t=69 done=0 sample=13 dac=1 cmp=1 adc=0
t=70 done=0 sample=13 dac=2 cmp=1 adc=0
t=71 done=0 sample=13 dac=3 cmp=1 adc=0
t=72 done=0 sample=13 dac=4 cmp=1 adc=0
t=73 done=0 sample=13 dac=5 cmp=1 adc=0
t=74 done=0 sample=13 dac=6 cmp=1 adc=0
t=75 done=0 sample=13 dac=7 cmp=1 adc=0
```

```
t=65 done=1 sample=0 dac=0 cmp=1 adc=0
t=133 done=1 sample=13 dac=0 cmp=1 adc=13
t=200 done=1 sample=26 dac=0 cmp=1 adc=26
t=268 done=1 sample=39 dac=0 cmp=1 adc=39
t=336 done=1 sample=52 dac=0 cmp=1 adc=52
t=404 done=1 sample=1 dac=0 cmp=1 adc=1
t=472 done=1 sample=14 dac=0 cmp=1 adc=14
t=540 done=1 sample=27 dac=0 cmp=1 adc=27
t=608 done=1 sample=40 dac=0 cmp=1 adc=40
t=676 done=1 sample=53 dac=0 cmp=1 adc=53
t=744 done=1 sample=2 dac=0 cmp=1 adc=2
t=812 done=1 sample=15 dac=0 cmp=1 adc=15
t=880 done=1 sample=28 dac=0 cmp=1 adc=28
t=948 done=1 sample=41 dac=0 cmp=1 adc=41
t=1016 done=1 sample=54 dac=0 cmp=1 adc=54
t=1084 done=1 sample=3 dac=0 cmp=1 adc=3
t=1152 done=1 sample=16 dac=0 cmp=1 adc=16
t=1220 done=1 sample=29 dac=0 cmp=1 adc=29
t=1288 done=1 sample=42 dac=0 cmp=1 adc=42
t=1356 done=1 sample=55 dac=0 cmp=1 adc=55
t=1424 done=1 sample=4 dac=0 cmp=1 adc=4
t=1492 done=1 sample=17 dac=0 cmp=1 adc=17
t=1560 done=1 sample=30 dac=0 cmp=1 adc=30
t=1628 done=1 sample=43 dac=0 cmp=1 adc=43
t=1696 done=1 sample=56 dac=0 cmp=1 adc=56
t=1764 done=1 sample=5 dac=0 cmp=1 adc=5
t=1832 done=1 sample=18 dac=0 cmp=1 adc=18
t=1900 done=1 sample=31 dac=0 cmp=1 adc=31
t=1968 done=1 sample=44 dac=0 cmp=1 adc=44
t=2036 done=1 sample=57 dac=0 cmp=1 adc=57
t=2104 done=1 sample=6 dac=0 cmp=1 adc=6
t=2172 done=1 sample=19 dac=0 cmp=1 adc=19
t=2240 done=1 sample=32 dac=0 cmp=1 adc=32
t=2308 done=1 sample=45 dac=0 cmp=1 adc=45
t=2376 done=1 sample=58 dac=0 cmp=1 adc=58
t=2444 done=1 sample=7 dac=0 cmp=1 adc=7
t=2512 done=1 sample=20 dac=0 cmp=1 adc=20
t=2580 done=1 sample=33 dac=0 cmp=1 adc=33
t=2648 done=1 sample=46 dac=0 cmp=1 adc=46
t=2716 done=1 sample=59 dac=0 cmp=1 adc=59
t=2784 done=1 sample=8 dac=0 cmp=1 adc=8
t=2852 done=1 sample=21 dac=0 cmp=1 adc=21
t=2920 done=1 sample=34 dac=0 cmp=1 adc=34
t=2988 done=1 sample=47 dac=0 cmp=1 adc=47
t=3056 done=1 sample=60 dac=0 cmp=1 adc=60
t=3124 done=1 sample=9 dac=0 cmp=1 adc=9
t=3192 done=1 sample=22 dac=0 cmp=1 adc=22
t=3260 done=1 sample=35 dac=0 cmp=1 adc=35
t=3328 done=1 sample=48 dac=0 cmp=1 adc=48
t=3396 done=1 sample=61 dac=0 cmp=1 adc=61
t=3464 done=1 sample=10 dac=0 cmp=1 adc=10
t=3532 done=1 sample=23 dac=0 cmp=1 adc=23
t=3600 done=1 sample=36 dac=0 cmp=1 adc=36
t=3668 done=1 sample=49 dac=0 cmp=1 adc=49
t=3736 done=1 sample=62 dac=0 cmp=1 adc=62
t=3804 done=1 sample=11 dac=0 cmp=1 adc=11
t=3872 done=1 sample=24 dac=0 cmp=1 adc=24
t=3940 done=1 sample=37 dac=0 cmp=1 adc=37
t=4008 done=1 sample=50 dac=0 cmp=1 adc=50
t=4076 done=1 sample=63 dac=0 cmp=1 adc=63
t=4144 done=1 sample=12 dac=0 cmp=1 adc=12
t=4212 done=1 sample=25 dac=0 cmp=1 adc=25
t=4280 done=1 sample=38 dac=0 cmp=1 adc=38
t=4348 done=1 sample=51 dac=0 cmp=1 adc=51
t=4416 done=1 sample=0 dac=0 cmp=1 adc=0
```

It is actually somewhat gratuitous to use a DAC to generate a linear ramp. More conventionally, one would use a current source to charge a capacitor for linear ramp.

In the lab, you will use the DG403 FET switch from Week 7 to make a sample & hold. You will use a resistor-ladder and an opamp follower to make a 6-bit DAC. You will use your Digilent CMOD FPGA module to sequence the digitization and to display the converted samples.



The main disadvantage of this style of ADC is that it tends to be quite slow. It is used for applications such as digital volt meters, where accuracy and cost are important, but speed is not a factor.

Note that each additional ADC bit doubles the conversion time.

Successive Approximation ADC

- Sample & hold
- Set most significant bit (bit 5 in our example) of DAC value; all others ∅
- If sample ≥ DAC, bit should stay 1; else ∅
- With bit 5 in its proper position, now try bit 4
- Etc. for bits 3, 2, 1, 0.
- So after 6 clock cycles, all 6 ADC bits are known (plus edge effects)

Going from an N-bit ADC to an N+1-bit ADC increases conversion time by a factor $\frac{N+1}{N}$, whereas conversion time would be doubled for a Wilkinson-type ADC.

Most high-speed ADCs today use successive approximation, though some use added tricks to determine more than one bit per clock cycle.

You will implement a 6-bit successive approximation ADC in this week's lab. Let's sketch out the design.

```
/*
 * lab12_adc_binary.v
 * demonstrate successive-approximation ADC operation for physics 364 week 12
 * begun 2010-11-29
 * by Bill Ashmanskas, ashmansk@hep.upenn.edu
 */


`default_nettype none
`timescale 1ns/1ns


/*
 * Implement successive-approximation ADC
 */
module adc (
    input wire        clk,              // 68 kHz clock, for ~ 7 kHz sampling
    input wire        cmp,              // 1 if sample > DAC value, else 0
    output wire [5:0] dac,              // DAC used for comparing with sample
    output wire [5:0] adc,              // ADC output value
    output wire       conv_done,        // indicate next ADC value is ready
    output wire       sample_hold       // closes FET switch for sample & hold
);
    // modulo-10 counter: ramp 0..5, then allow 4 extra ticks for processing
    wire [3:0] count;
    wire [3:0] count_next = (count==9) ? 0 : count+1;
    dffe_Nbit #(4) count_ff (.clk(clk), .ena(1), .d(count_next), .q(count));
    // Update DAC bits:
    //    on tick 0,1,2,3,4,5, bit 5,4,3,2,1,0 is set to 1;
    //    on tick 1,2,3,4,5,6, bit 5,4,3,2,1,0 is set to comparison outcome
    //    on tick 7, all bits are set to 0 for new cycle
    wire [5:0] dac_next;
    dffe_Nbit #(6) dac_ff (.clk(clk), .ena(1), .d(dac_next), .q(dac));
    assign dac_next[5] = (count==0) ? 1   :
                         (count==1) ? cmp :
                         (count==7) ? 0   : dac[5];
    assign dac_next[4] = (count==1) ? 1   :
                         (count==2) ? cmp :
                         (count==7) ? 0   : dac[4];
    assign dac_next[3] = (count==2) ? 1   :
                         (count==3) ? cmp :
                         (count==7) ? 0   : dac[3];
    assign dac_next[2] = (count==3) ? 1   :
                         (count==4) ? cmp :
                         (count==7) ? 0   : dac[2];
    assign dac_next[1] = (count==4) ? 1   :
                         (count==5) ? cmp :
                         (count==7) ? 0   : dac[1];
    assign dac_next[0] = (count==5) ? 1   :
                         (count==6) ? cmp :
                         (count==7) ? 0   : dac[0];
    // ...
    dffe_Nbit #(6) adc_ff (.clk(clk), .ena(count==7), .d(dac), .q(adc));
    // when ADC output register has updated, strobe "conversion done" signal
    assign conv_done = (count==8);
    // before new conversion begins, tell sample & hold to get new sample
    assign sample_hold = (count==9);
endmodule
```

```
t=1 done=0 sample=0 dac=32/100000 cmp=0 adc=0
t=2 done=0 sample=0 dac=16/010000 cmp=0 adc=0
t=3 done=0 sample=0 dac=8/001000 cmp=0 adc=0
t=4 done=0 sample=0 dac=4/000100 cmp=0 adc=0
t=5 done=0 sample=0 dac=2/000010 cmp=0 adc=0
t=6 done=0 sample=0 dac=1/000001 cmp=0 adc=0
t=7 done=0 sample=0 dac=0/000000 cmp=1 adc=0
t=8 done=1 sample=0 dac=0/000000 cmp=1 adc=0
t=9 done=0 sample=13 dac=0/000000 cmp=1 adc=0
t=10 done=0 sample=13 dac=0/000000 cmp=1 adc=0
t=11 done=0 sample=13 dac=32/100000 cmp=0 adc=0
t=12 done=0 sample=13 dac=16/010000 cmp=0 adc=0
t=13 done=0 sample=13 dac=8/001000 cmp=1 adc=0
t=14 done=0 sample=13 dac=12/001100 cmp=1 adc=0
t=15 done=0 sample=13 dac=14/001110 cmp=0 adc=0
t=16 done=0 sample=13 dac=13/001101 cmp=1 adc=0
t=17 done=0 sample=13 dac=13/001101 cmp=1 adc=0
t=18 done=1 sample=13 dac=0/000000 cmp=1 adc=13
t=19 done=0 sample=26 dac=0/000000 cmp=1 adc=13
t=20 done=0 sample=26 dac=0/000000 cmp=1 adc=13
t=21 done=0 sample=26 dac=32/100000 cmp=0 adc=13
t=22 done=0 sample=26 dac=16/010000 cmp=1 adc=13
t=23 done=0 sample=26 dac=24/011000 cmp=1 adc=13
t=24 done=0 sample=26 dac=28/011100 cmp=0 adc=13
t=25 done=0 sample=26 dac=26/011010 cmp=1 adc=13
t=26 done=0 sample=26 dac=27/011011 cmp=0 adc=13
t=27 done=0 sample=26 dac=26/011010 cmp=1 adc=13
t=28 done=1 sample=26 dac=0/000000 cmp=1 adc=26
t=29 done=0 sample=39 dac=0/000000 cmp=1 adc=26
t=30 done=0 sample=39 dac=0/000000 cmp=1 adc=26
t=31 done=0 sample=39 dac=32/100000 cmp=1 adc=26
t=32 done=0 sample=39 dac=48/110000 cmp=0 adc=26
t=33 done=0 sample=39 dac=40/101000 cmp=0 adc=26
t=34 done=0 sample=39 dac=36/100100 cmp=1 adc=26
t=35 done=0 sample=39 dac=38/100110 cmp=1 adc=26
t=36 done=0 sample=39 dac=39/100111 cmp=1 adc=26
t=37 done=0 sample=39 dac=39/100111 cmp=1 adc=26
t=38 done=1 sample=39 dac=0/000000 cmp=1 adc=39
t=39 done=0 sample=52 dac=0/000000 cmp=1 adc=39
t=40 done=0 sample=52 dac=0/000000 cmp=1 adc=39
t=41 done=0 sample=52 dac=32/100000 cmp=1 adc=39
t=42 done=0 sample=52 dac=48/110000 cmp=1 adc=39
t=43 done=0 sample=52 dac=56/111000 cmp=0 adc=39
t=44 done=0 sample=52 dac=52/110100 cmp=1 adc=39
t=45 done=0 sample=52 dac=54/110110 cmp=0 adc=39
t=46 done=0 sample=52 dac=53/110101 cmp=0 adc=39
t=47 done=0 sample=52 dac=52/110100 cmp=1 adc=39
t=48 done=1 sample=52 dac=0/000000 cmp=1 adc=52
t=49 done=0 sample=1 dac=0/000000 cmp=1 adc=52
t=50 done=0 sample=1 dac=0/000000 cmp=1 adc=52
t=51 done=0 sample=1 dac=32/100000 cmp=0 adc=52
t=52 done=0 sample=1 dac=16/010000 cmp=0 adc=52
t=53 done=0 sample=1 dac=8/001000 cmp=0 adc=52
t=54 done=0 sample=1 dac=4/000100 cmp=0 adc=52
t=55 done=0 sample=1 dac=2/000010 cmp=0 adc=52
t=56 done=0 sample=1 dac=1/000001 cmp=1 adc=52
t=57 done=0 sample=1 dac=1/000001 cmp=1 adc=52
t=58 done=1 sample=1 dac=0/000000 cmp=1 adc=1
t=59 done=0 sample=14 dac=0/000000 cmp=1 adc=1
t=60 done=0 sample=14 dac=0/000000 cmp=1 adc=1
```

```
t=8 done=1 sample=0 dac=0/000000 cmp=1 adc=0
t=18 done=1 sample=13 dac=0/000000 cmp=1 adc=13
t=28 done=1 sample=26 dac=0/000000 cmp=1 adc=26
t=38 done=1 sample=39 dac=0/000000 cmp=1 adc=39
t=48 done=1 sample=52 dac=0/000000 cmp=1 adc=52
t=58 done=1 sample=1 dac=0/000000 cmp=1 adc=1
t=68 done=1 sample=14 dac=0/000000 cmp=1 adc=14
t=78 done=1 sample=27 dac=0/000000 cmp=1 adc=27
t=88 done=1 sample=40 dac=0/000000 cmp=1 adc=40
t=98 done=1 sample=53 dac=0/000000 cmp=1 adc=53
t=108 done=1 sample=2 dac=0/000000 cmp=1 adc=2
t=118 done=1 sample=15 dac=0/000000 cmp=1 adc=15
t=128 done=1 sample=28 dac=0/000000 cmp=1 adc=28
t=138 done=1 sample=41 dac=0/000000 cmp=1 adc=41
t=148 done=1 sample=54 dac=0/000000 cmp=1 adc=54
t=158 done=1 sample=3 dac=0/000000 cmp=1 adc=3
t=168 done=1 sample=16 dac=0/000000 cmp=1 adc=16
t=178 done=1 sample=29 dac=0/000000 cmp=1 adc=29
t=188 done=1 sample=42 dac=0/000000 cmp=1 adc=42
t=198 done=1 sample=55 dac=0/000000 cmp=1 adc=55
t=207 done=1 sample=4 dac=0/000000 cmp=1 adc=4
t=217 done=1 sample=17 dac=0/000000 cmp=1 adc=17
t=227 done=1 sample=30 dac=0/000000 cmp=1 adc=30
t=237 done=1 sample=43 dac=0/000000 cmp=1 adc=43
t=247 done=1 sample=56 dac=0/000000 cmp=1 adc=56
t=257 done=1 sample=5 dac=0/000000 cmp=1 adc=5
t=267 done=1 sample=18 dac=0/000000 cmp=1 adc=18
t=277 done=1 sample=31 dac=0/000000 cmp=1 adc=31
t=287 done=1 sample=44 dac=0/000000 cmp=1 adc=44
t=297 done=1 sample=57 dac=0/000000 cmp=1 adc=57
t=307 done=1 sample=6 dac=0/000000 cmp=1 adc=6
t=317 done=1 sample=19 dac=0/000000 cmp=1 adc=19
t=327 done=1 sample=32 dac=0/000000 cmp=1 adc=32
t=337 done=1 sample=45 dac=0/000000 cmp=1 adc=45
t=347 done=1 sample=58 dac=0/000000 cmp=1 adc=58
t=357 done=1 sample=7 dac=0/000000 cmp=1 adc=7
t=367 done=1 sample=20 dac=0/000000 cmp=1 adc=20
t=377 done=1 sample=33 dac=0/000000 cmp=1 adc=33
t=387 done=1 sample=46 dac=0/000000 cmp=1 adc=46
t=397 done=1 sample=59 dac=0/000000 cmp=1 adc=59
t=407 done=1 sample=8 dac=0/000000 cmp=1 adc=8
t=417 done=1 sample=21 dac=0/000000 cmp=1 adc=21
t=427 done=1 sample=34 dac=0/000000 cmp=1 adc=34
t=437 done=1 sample=47 dac=0/000000 cmp=1 adc=47
t=447 done=1 sample=60 dac=0/000000 cmp=1 adc=60
t=457 done=1 sample=9 dac=0/000000 cmp=1 adc=9
t=467 done=1 sample=22 dac=0/000000 cmp=1 adc=22
t=477 done=1 sample=35 dac=0/000000 cmp=1 adc=35
t=487 done=1 sample=48 dac=0/000000 cmp=1 adc=48
t=497 done=1 sample=61 dac=0/000000 cmp=1 adc=61
t=507 done=1 sample=10 dac=0/000000 cmp=1 adc=10
t=517 done=1 sample=23 dac=0/000000 cmp=1 adc=23
t=527 done=1 sample=36 dac=0/000000 cmp=1 adc=36
t=537 done=1 sample=49 dac=0/000000 cmp=1 adc=49
t=547 done=1 sample=62 dac=0/000000 cmp=1 adc=62
t=557 done=1 sample=11 dac=0/000000 cmp=1 adc=11
t=567 done=1 sample=24 dac=0/000000 cmp=1 adc=24
t=577 done=1 sample=37 dac=0/000000 cmp=1 adc=37
t=587 done=1 sample=50 dac=0/000000 cmp=1 adc=50
t=597 done=1 sample=63 dac=0/000000 cmp=1 adc=63
t=607 done=1 sample=12 dac=0/000000 cmp=1 adc=12
t=617 done=1 sample=25 dac=0/000000 cmp=1 adc=25
t=627 done=1 sample=38 dac=0/000000 cmp=1 adc=38
t=637 done=1 sample=51 dac=0/000000 cmp=1 adc=51
t=647 done=1 sample=0 dac=0/000000 cmp=1 adc=0
t=657 done=1 sample=13 dac=0/000000 cmp=1 adc=13
t=667 done=1 sample=26 dac=0/000000 cmp=1 adc=26
t=677 done=1 sample=39 dac=0/000000 cmp=1 adc=39
```

Finally, the very fastest ADCs are "flash" ADCs, which require 2^N comparators for an N-bit ADC.

So your 6-bit ADC, if done as a flash, would require 64 comparators, plus an "priority encoder" to pick the highest "1" input.

This is prohibitively expensive & power hungry for more than 8 bits or so.

This is why a digital oscilloscope is normally good for only 8 bit per sample.

from HH

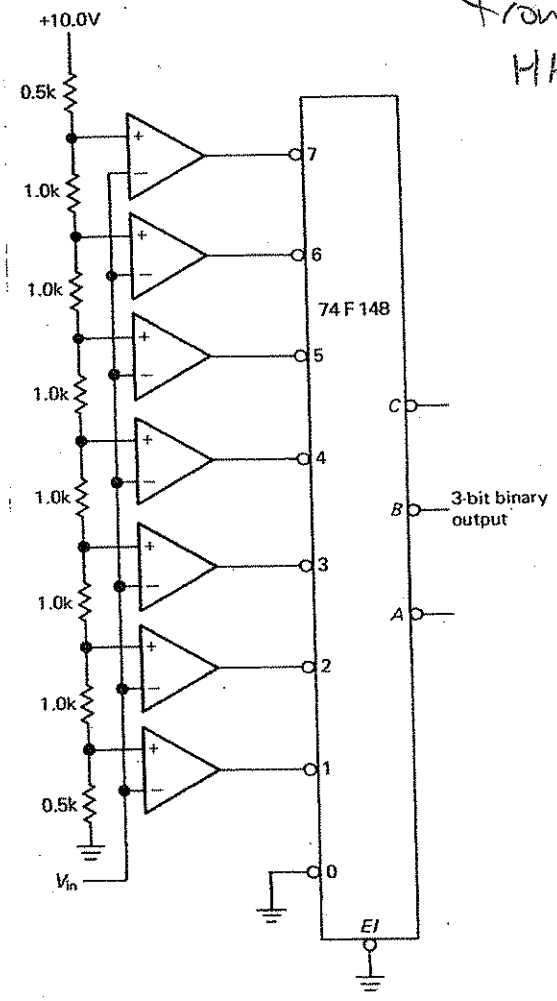


Figure 9.49. Parallel-encoded ("flash") A/D converter (ADC).

Sketch of Lab 12

(I) Build 6-bit R-2R DAC, using FPGA, resistors, and opamp.

- (a) Clock the DAC using function generator
- (b) Build a 555 timer circuit to provide the clock, so that function generator is free for other use.

(II) Build 6-bit Wilkinson-style ADC, sampling at 1 KHz.

- (a) Display output on LED bar graph.
- (b) Build a second R-2R DAC to display ADC output.
- (c) Drive ADC input with fcn generator; watch ADC output on scope. Look for aliasing as frequency climbs above f_N.

(III) Re-program FPGA to turn your ADC into a successive-approximation ADC.

Watch the "binary search" proceed on the scope during each conversion cycle.