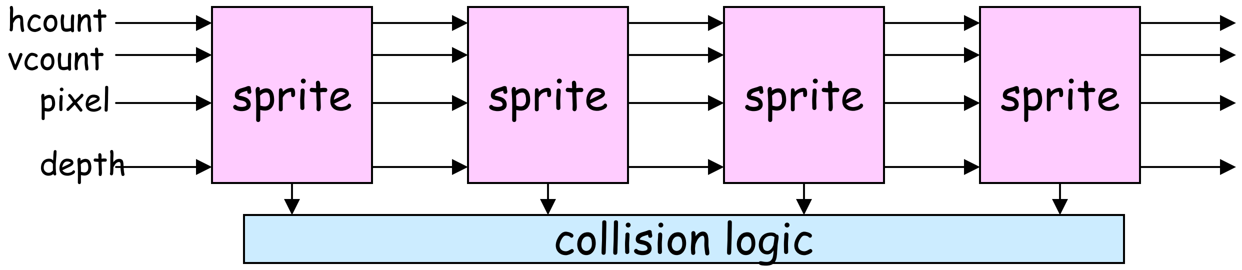
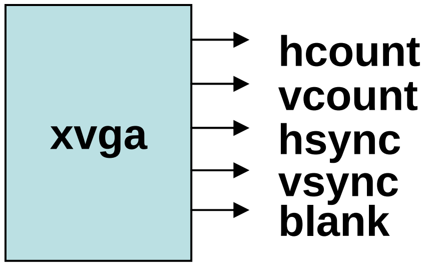
Game Graphics using Sprites

- Sprite = game object occupying a rectangular region of the screen (it's bounding box).
  - Usually it contains both opaque and transparent pixels.
  - Given (H,V), sprite returns pixel (0=transparent) and depth
  - Pseudo 3D: look at current pixel from all sprites, display the opaque one that's in front (min depth): see sprite pipeline below
  - Collision detection: look for opaque pixels from other sprites
  - Motion: smoothly change coords of upper left-hand corner
- Pixels can be generated by logic or fetched from a bitmap (memory holding array of pixels).
  - Bitmap may have multiple images that can be displayed in rapid succession to achieve animation.
  - Mirroring and 90° rotation by fooling with bitmap address, crude scaling by pixel replication, or resizing filter.

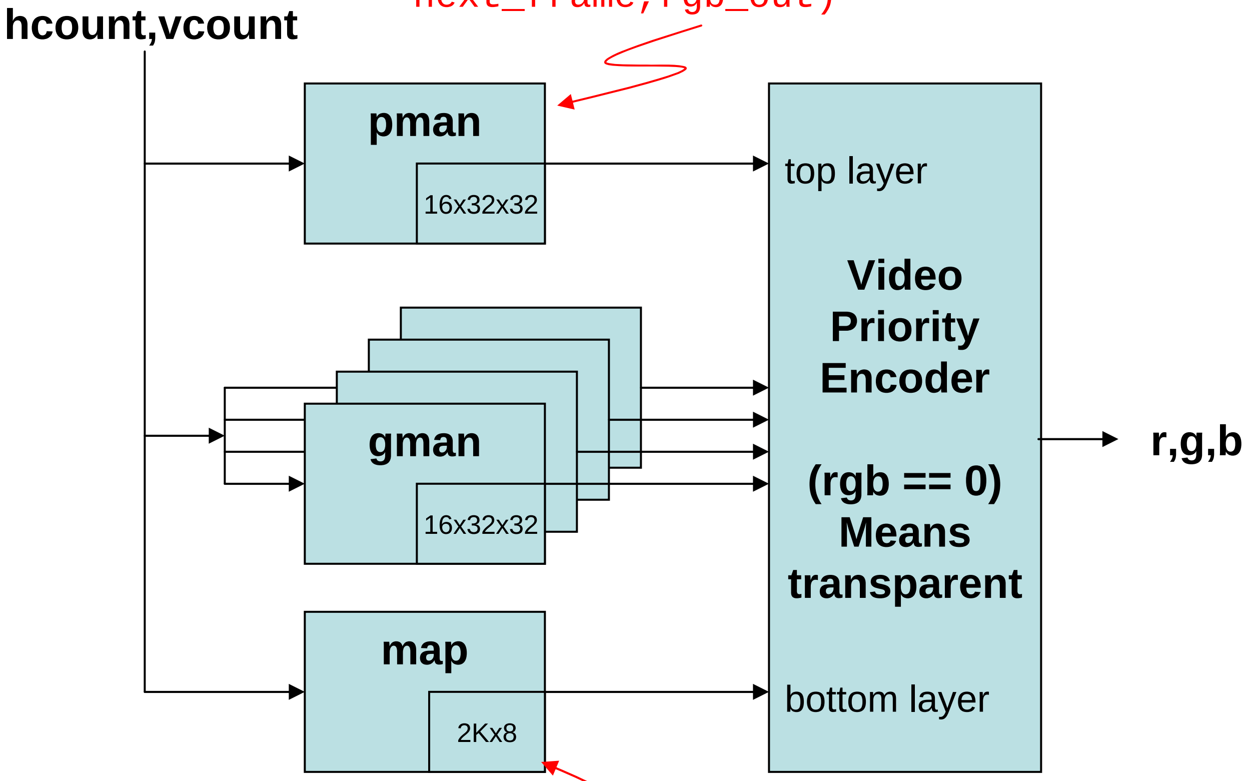


Demo (Pacman: video)



Sprite: rectangular region of pixels, position and color set by game logic. 32x32 pixel mono image from BRAM, up to 16 frames displayed in loop for animation:

```
sprite(clk,reset,hcount,vcount,xpos,ypos,color,
       next_frame,rgb_out)
```

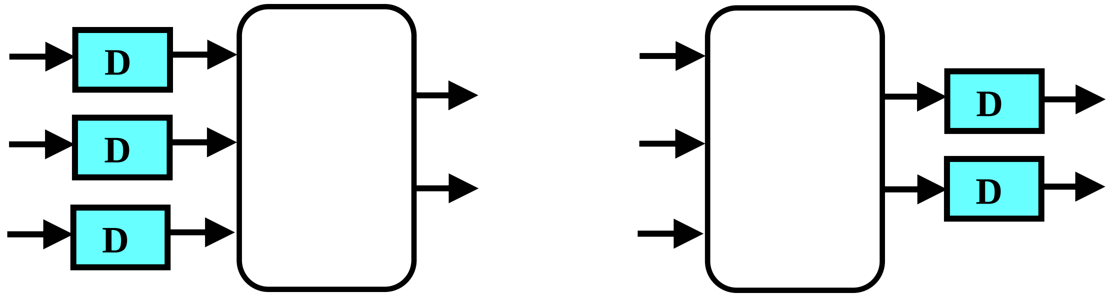


4 board maps, each 512x8
each map is 16x24 tiles (376 tiles)
Each tile has 8 bits: 4 for move direction (==0 for a wall), pills

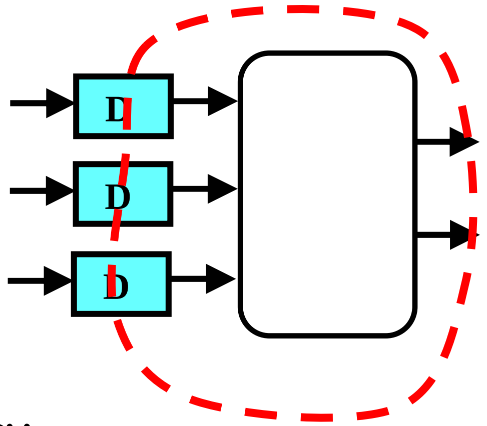
Retiming: A very useful transform

Retiming is the action of moving delay around in the systems

- Delays have to be moved from ALL inputs to ALL outputs or vice versa



Cutset retiming: A cutset intersects the edges, such that this would result in two disjoint partitions of these edges being cut. To retime, delays are moved from the ingoing to the outgoing edges or vice versa.



Retiming Synchronous Circuitry

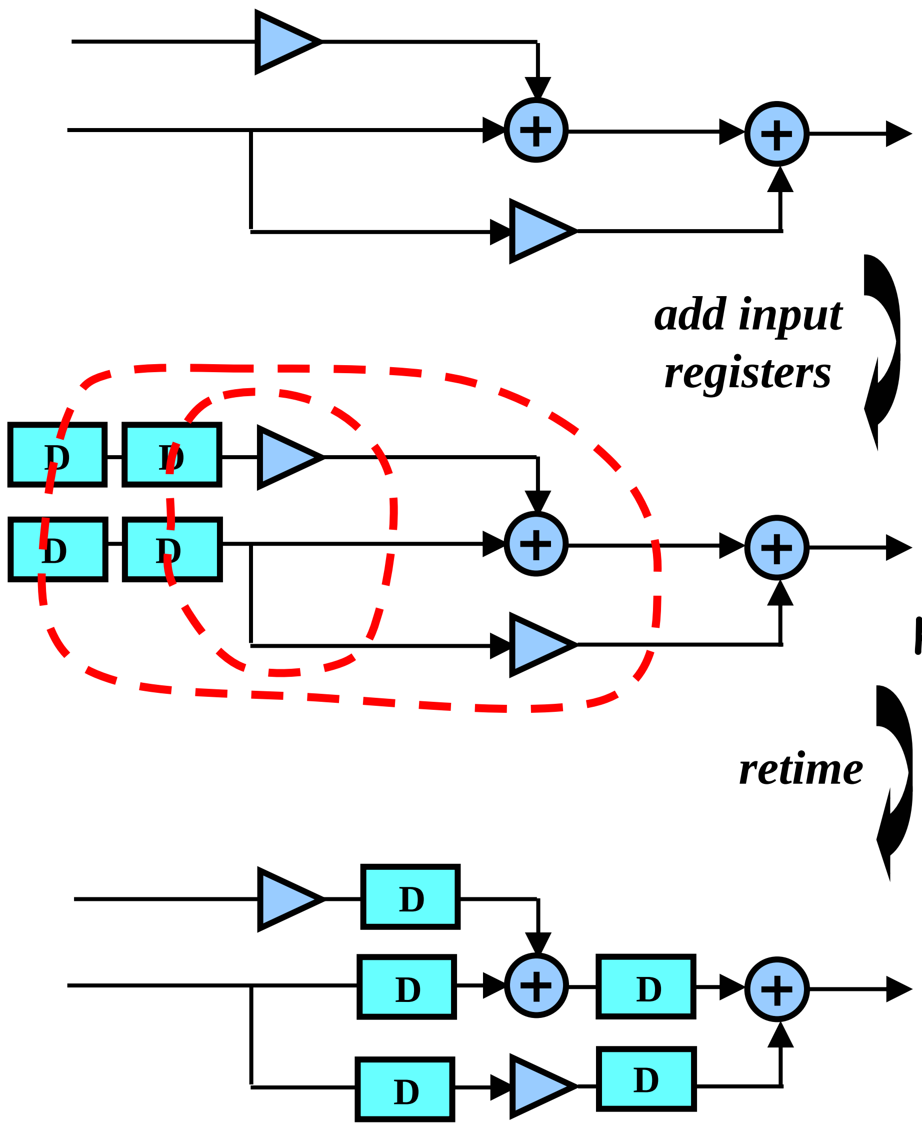
Charles E. Leiserson and James B. Saxe

August 20, 1986.

Benefits of retiming:

- Modify critical path delay
- Reduce total number of registers

Pipelining, Just Another Transformation (Pipelining = Adding Delays + Retiming)

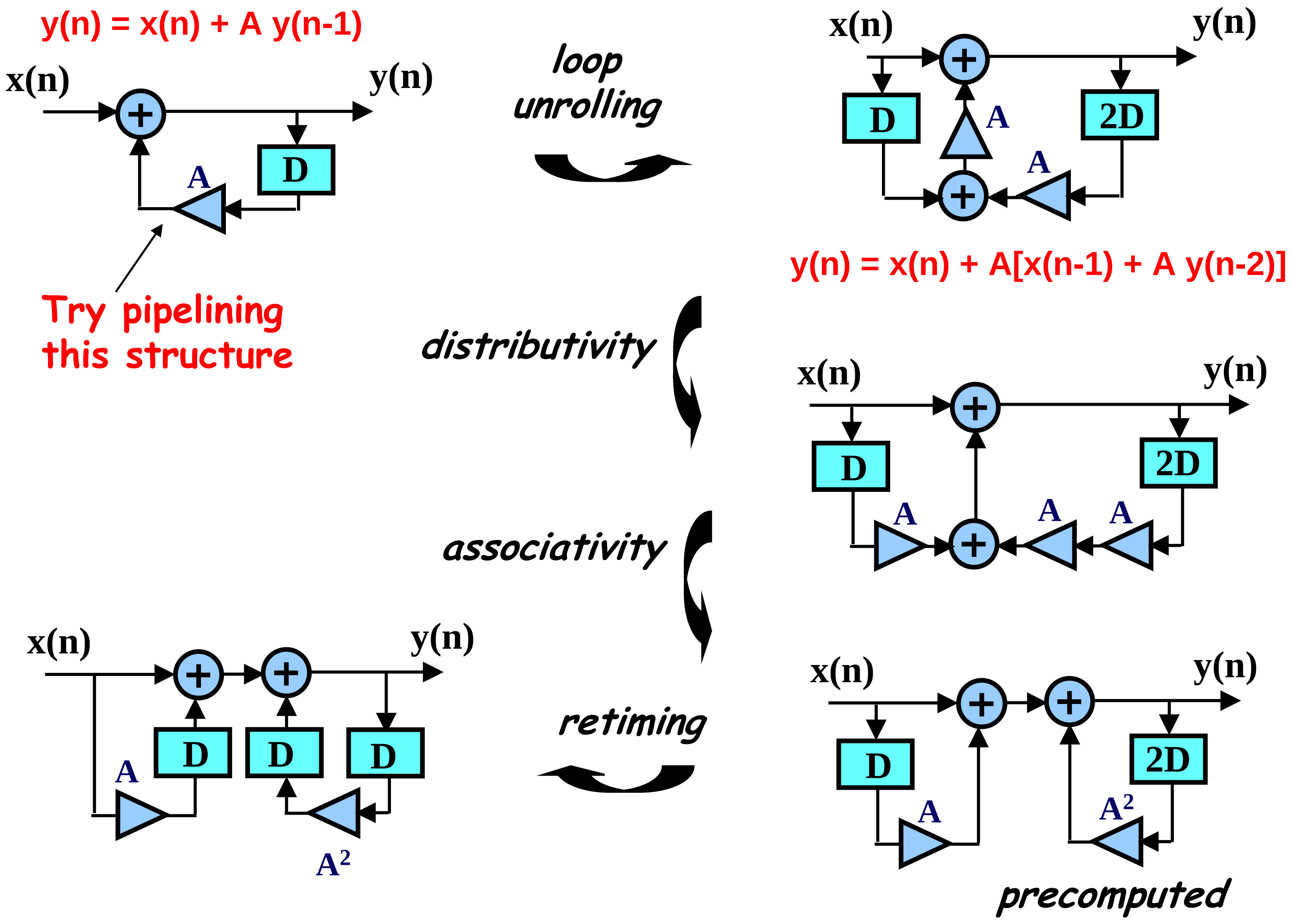


Contrary to retiming, pipelining adds extra registers to the system

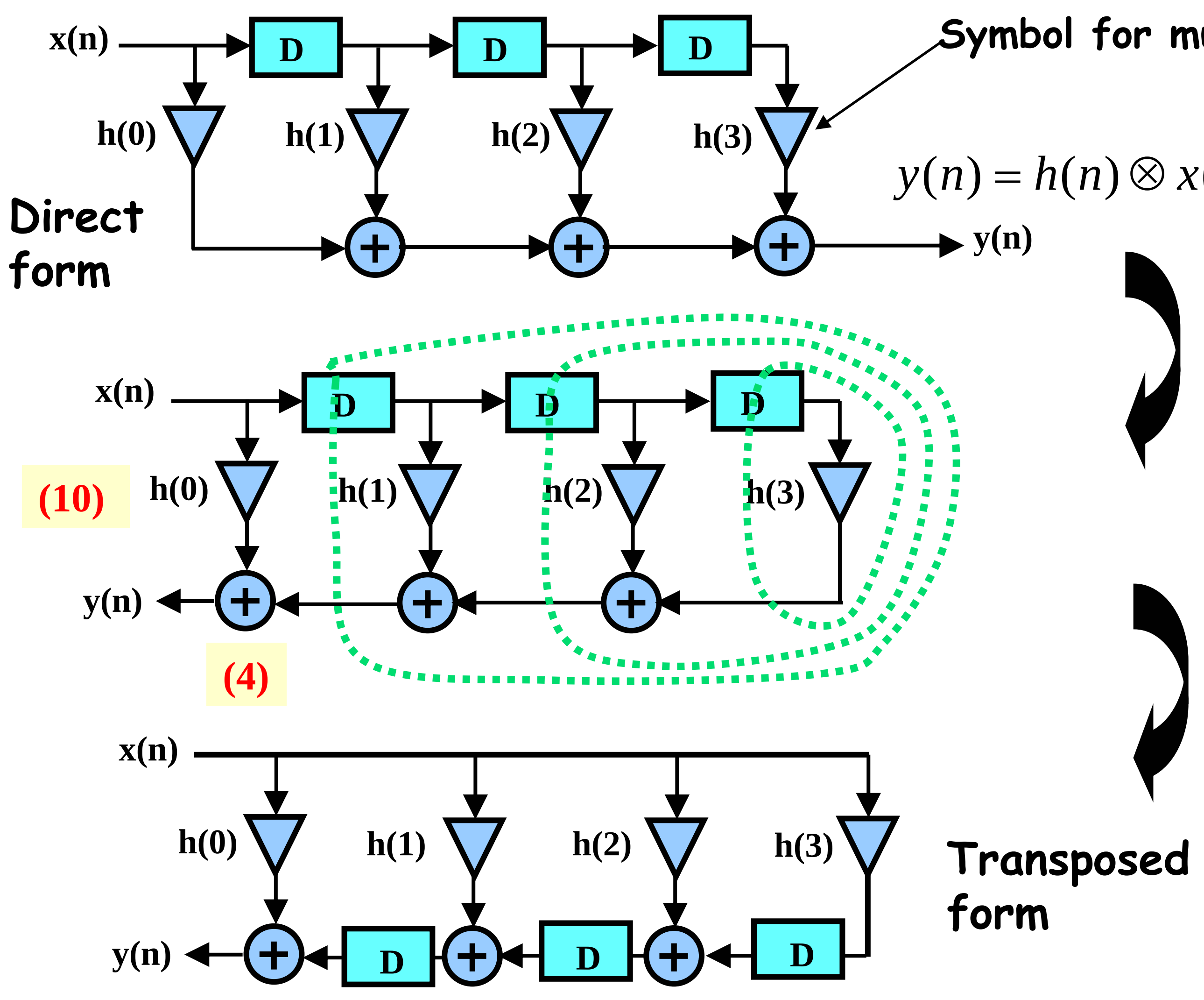
How to pipeline:

1. Add extra registers at *all* inputs (or, equivalently, *all* outputs)
2. Retime

The Power of Transforms: Lookahead



Retiming Example: FIR Filter



$$y(n) = h(n) \otimes x(n) = \sum_{i=0}^{K} x(n-i) \cdot h(i)$$

associativity of addition

T_{clk} = 22 ns

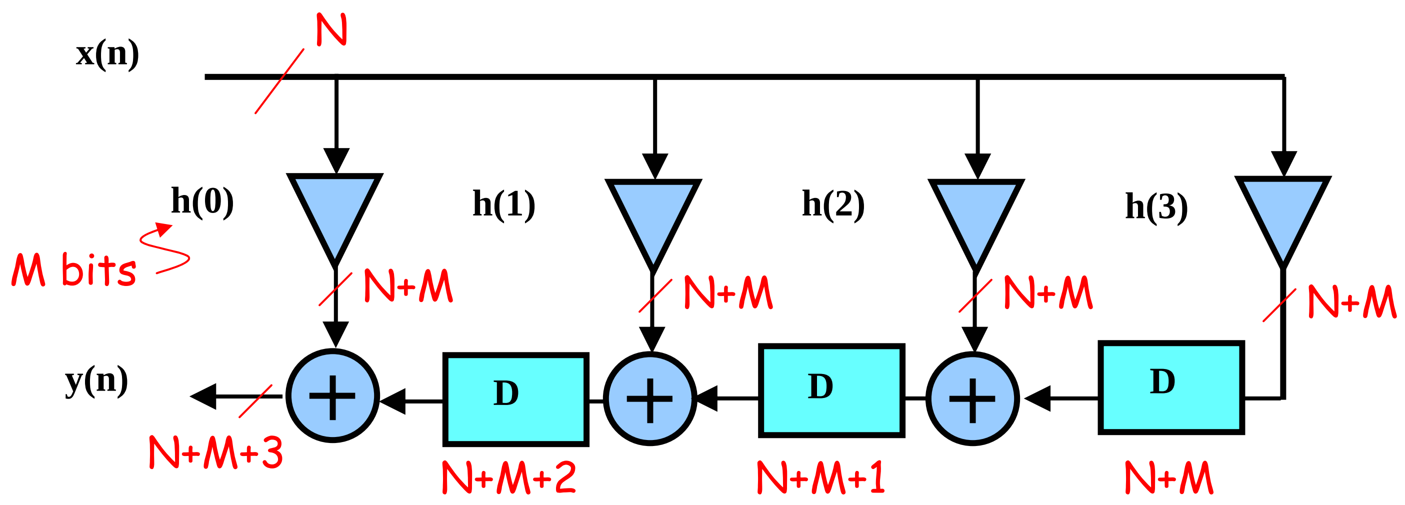
retime

T_{clk} = 14 ns

Note: here we use a first cut analysis that assumes the delay of a chain of operators is the sum of their individual delays. This is not accurate.

FIR design issues

- Keeping track required numeric precision

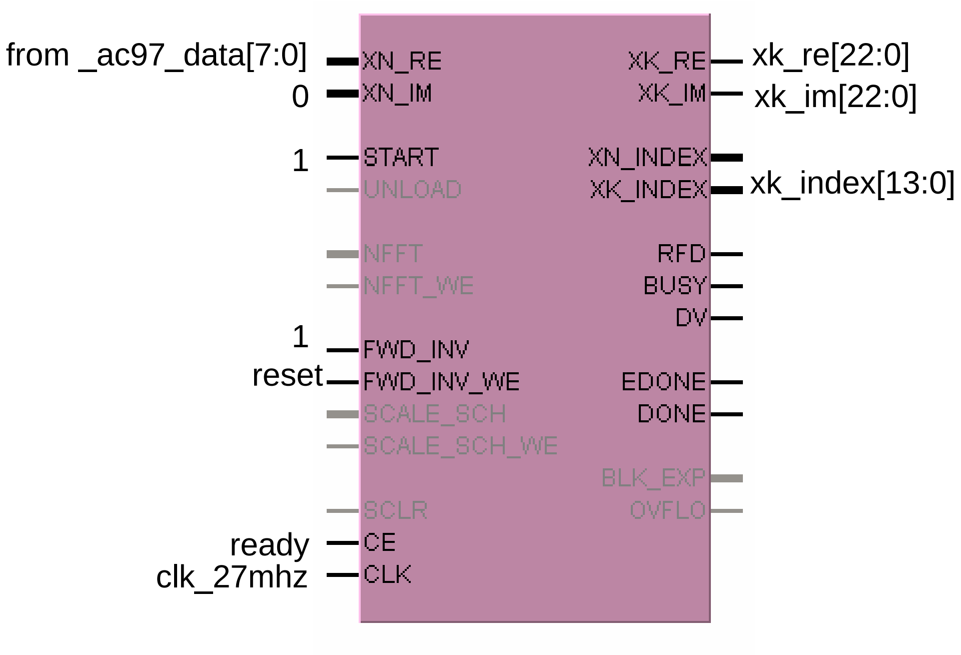


- Scale fractional coefficients to integer values by multiplying by 2^C to get C-bit coefficients. Remember to divide filter output by same scale factor (division by 2^C doesn't require logic, just eliminate the C low order bits).

- Xilinx IP Core has generators for many different FIR types

FFT example

```
// Transform length: 16384
// Implementation options: Pipelined, Streaming I/O
// Transform length options: none
// Input data width: 8
// Phase factor width: 8
// Optional pins: CE
// Scaling options: Unscaled
// Rounding mode: Truncation
// Number of stages using Block Ram: 7
// Output ordering: Bit/Digit Reversed Order
fft16384u fft(.clk(clock_27mhz), .ce(reset | ready),
             .xn_re(from_ac97_data[7:0]), .xn_im(8'b0),
             .start(1'b1), .fwd_inv(1'b1), .fwd_inv_we(reset),
             .xk_re(xk_re[22:0]), .xk_im(xk_im[22:0]), .xk_index(xk_index[13:0]));
```



FFT of AC97 data

To process AC97 samples:

- use Pipelined mode (input one sample in each cycle, get one sample out each cycle).
  - FFT expects one sample each cycle, so hook READY to CE so that FFT only cycles once per AC97 frame
- use Unscaled mode, do scaling yourself
  - Number of output bits = (input width) + NFFT + 1
  - NFFT is $\log_2$(size of FFT)
- let number of FFT points = P, assume 48kHz sample rate
  - there are P frequency bins
  - positive freqs in bins 0 to (P/2 – 1)
  - negative freqs in bins (P/2) to (P-1)
  - each bin covers (48k/P)Hz
  - Use XK_INDEX to tell which bin's data you're getting out
  - Typically you want magnitude = sqrt(xk_re^2 + xk_im^2)

Verilog Event Processing

- "Active" events
  - Continuous assignments
  - Statements within active `always` blocks
    - Blocking assignments (=)
    - RHS of non-blocking assignments (<=)
- Active events are evaluated in *arbitrary order*
  - Interleaved execution of statements in different active `always` blocks or continuous assignments is possible
  - Statements are executed sequentially only with respect to other statements within the same `always` block
- Assignments to LHS of non-blocking assignments happens after all active events have been processed
- Because of interleaved execution, blocking assignments can lead to *nondeterministic behavior* (this is bad!).

= vs. <= inside always

```
module main;
  reg a,b,clk;



  initial begin
    clk = 0; a = 0; b = 1;
    #10 clk = 1;
    #10 $display("a=%d b=%d\n",a,b);
    $finish;
  end
endmodule
```

A

```
always @(posedge clk) begin
  a = b;    // blocking assignment
  b = a;    // execute sequentially
end
```

B

```
always @(posedge clk) begin
  a <= b;  // non-blocking assignment
  b <= a;  // eval all RHSs first
end
```

C

```
always @(posedge clk) a = b;
always @(posedge clk) b = a;
```

D

```
always @(posedge clk) a <= b;
always @(posedge clk) b <= a;
```

E

```
always @(posedge clk) begin
  a <= b;
  b = a;    // urk! Be consistent!
end
```

Rule: always change state using <= (e.g., inside `always @(posedge clk)`…)