Department of Electrical & Electronic Engineeing
Imperial College London

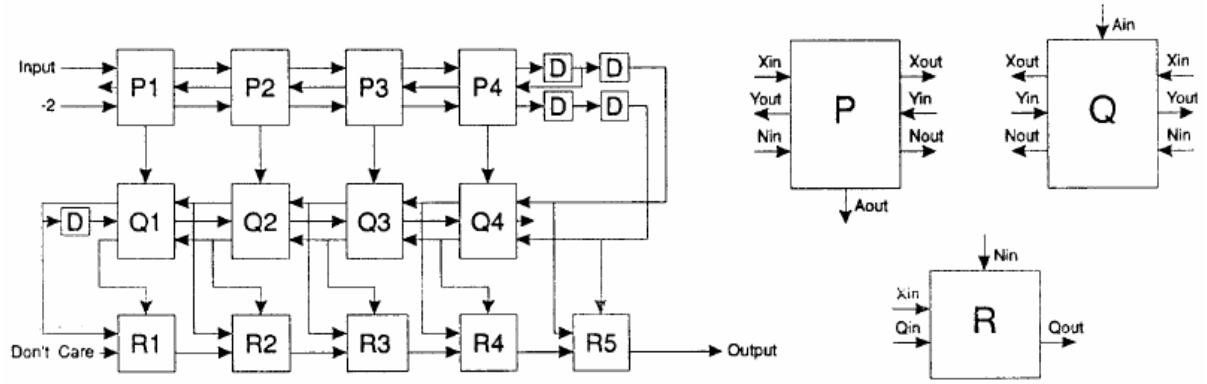
E4.20 Digital IC Design

Median Filter Project Specification

A median filter is used to remove noise from a sampled signal. It behaves in a similar way to a low-pass filter but is more effective at removing noise that is impulsive in nature. An example of impulsive noise is that generated from a car ignition system: each spark radiates a brief pulse of high energy that will only affect one or two signal samples leaving the remainder unchanged. A low-pass filter would smear out the noise pulse but a median filter will excise it almost perfectly.

In this project, you will design a median filter of order five. In the filter, each output sample is the third highest (= median) of five consecutive input samples. The filter introduces a delay and so the median of samples 1 to 5 is not in fact output until sample time 8.5. The input and output of the filter will be 4-bit signed numbers in the range -8 to +7.

The filter is implemented as a systolic array, i.e. a regular grid of simple processing elements (or cells) having only local interconnections. The array contains four different types of cell: these are labelled D, P, Q and R; their functions are defined below. Each cell introduces a half-clock delay: that is the cell's outputs change half a clock cycle later than its inputs. If you imagine the cells coloured like a chessboard, the outputs from all the white cells will change state together, while the outputs from all the black cells will change state half a clock cycle later.



The cells marked D are merely half cycle delays. The names of the inputs and outputs of the other cell types are indicated above; the outputs are defined as follows:

```
P:   Xout ← Xin; Yout ← Yin;
     if (Xin < Yin)   then   { Nout ← Nin+1; Aout ← 0}
                      else   { Nout ← Nin; Aout ← 1}

Q:   Xout ← Xin; Yout ← Yin;
     if (Xin < Yin)   then   { Nout ← Nin + Ain - 1}
                      else   { Nout ← Nin + Ain}

R:   if (Nin = 0)     then   { Qout ← Xin}
                      else   { Qout ← Qin}
```

Input samples are applied to the X_{in} input of cell P1; in the description below, the i^{th} input sample is denoted u_i. To understand how the circuit works, you need to follow the path between the input signal and the output. Input samples first travel from left to right along the top row (P cells: Xin/Xout) and then turn around and travel right to left along the middle row (Q cells: Xin/Xout). Finally they reverse direction again and travel left to right along the bottom

row (R cells Qin/Qout). Since each cell adds a half-sample delay, this whole procedure takes 7.5 sample. The only variation arises because the R cells act as switches and can if they wish connect Qout to Xin instead of to Qin. If R3 does this, for example, the signal can jump from the output of Q3 to the input of R3 thus eliminating four half-sample delays: Q2, Q1, R1 and R2. Thus if R1, R2, R3, R4 or R5 choose to switch, the input-to-output delay will be 7.5, 6.5, 5.5, 4.3, or 3.5 samples respectively. The output can therefore be any one of five consecutive samples as would be expected for a median filter.

In addition to forming the input of Q4, the Xout signal from P4 is delayed and then sent back along the top row as Yin/Yout. By counting half-sample delays, it is easy to see that u_i will meet samples u_{i-4}, u_{i-3}, u_{i-2}, u_{i-1} in cells Pl, P2, P3 and P4 respectively. In each case, if u_i is lower in value, Nout will be set to Nin+l. Thus as the signal N travels from left to right, it is incremented by one each time u_i is less than one of the four previous samples. When u_i emerges from P4:Xout, the number at P4:Nout has counted how many of the previous four samples are greater than or equal to u_i. Since P1:Nin is initialised to -2, it will have been incremented up to 0 if and only if u_i is less than two of the previous four samples and greater than or equal to the other two. If this is the case, u_i must be the median; cell R5 will switch and cause u_i to be emitted after a further 1.5 samples.

The outputs from P4 are delayed by one sample and then sent back into Q4 as Xin and Nin. Here u_i is again compared with sample u_{i-4} and the comparison will yield the same result as it did in cell P1. Since Nin will be decremented in Q4 if and only if it was incremented in P1, Q4 exactly undoes the effect of P1. Half a sample earlier however, u_i was also being compared to sample u_{i+1} in cell P4 and the result of this comparison is sent to Q4 as the Ain input. This causes Nin to be incremented if u_i is less than or equal to u_{i+1} .Thus Nout from Q4 is the result of comparing u_i with u_{i-3}, u_{i-2}, u_{i-1} and u_{i+1}. As before, the result will be 0 if u_i is the median and if this is the case, R4 will switch and u_i will be output at the appropriate time.

In a similar way Q3, Q2 and Q1 respectively detect when the median is the third, second or first of a group of five and cause the corresponding R cell to switch.

Implementation Considerations

You need to decide on the widths of the various signal paths. All the X, Y and Q signal must be 4 bits wide so that they can cope with input signals in the range -8 to +7. You will need to work out the required width for the N signals by working out the range of possible values that it can take at each point in the circuit; note that this range varies from place to place in the circuit.

Although the diagram on the previous page has an elegant symmetry, there are several economies that can be made:

1) The signals Yout from P3 and Xout from 44 are identical and need only be generated once. Note that while this saves you four latches, it will complicate the wiring required. Similar economies can be made for the other P and Q cells.

2) Since one of the R1 inputs is “Don’t Care”, the entire cell can be replaced by a delay. This can in turn be merged with the D cell above which acts on the same signal. The Nout output from Q1 is now redundant as is the Yout output from Q4.

Administrative Matters

Deadline for completion: First day of Spring term (3 Jan 08)
Deadline for report: Second Monday of Spring term (7 Jan 08)
Report (one per group) should both be in both paper and electronic forms, and include:

- description of circuit designed (full schematic and layout)
- block diagram showing different modules in the chip
- plot of the entire chip
- evidence that it works (from simulation plots)
- test strategy and testbench
- a description of contribution from each member, signed by all!

E4.20 Digital IC Design Course Work 2010-2011

- To design a ***Biquadratic Digital Filter*** based on **distributed arithmetics** as a full custom chip
- To test the design using Electric
- You will find the following references useful:
  - Peled and B. Liu, “A New Hardware Realization of Digital Filters”, *IEEE Trans. on Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.
  - S. A. White, ``Applications of Distributed Arithmetic to Digital Signal Processing'', *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.
  - The Role of Distributed Arithmetic in FPGA-based Signal Processing, *http://www.xilinx.com/appnotes/theory1.pdf*

What is a Digital Biquad Filter?

- Transfer function:

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}$$

- This can be rearranged as a difference equation:-

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} - b_1 y_{n-1} - b_2 y_{n-2}$$

- This can be generalised to an inner-product calculation:

$$y = \sum_{k=1}^{N} A_k x_k$$

Distributed Arithmetic (1)

- Let us express x_k in its 2's complement binary form:

$$x_k = -x_{k0} + x_{k1}2^{-1} + x_{k2}2^{-2} + \ldots\ldots + x_{k(B-1)}2^{-(B-1)}$$
$$= -x_{k0} + \sum_{i=1}^{B-1} x_{ki}2^{-i}$$

- Then:

$$y = \sum_{k=1}^{N} A_k \left[-x_{k0} + \sum_{i=1}^{B-1} x_{ki}2^{-i} \right] = -\sum_{k=1}^{N} x_{k0}A_k + \sum_{k=1}^{N}\sum_{i=1}^{B-1} x_{ki}A_k 2^{-i}$$

Distributed Arithmetic (2)

- Let us expand this to:

MSB of x_N

$$\begin{aligned} y = & -\left[x_{10}A_1 + x_{20}A_2 + x_{30}A_3 + \ldots\ldots\ldots + x_{N0}A_N\right] \\ & + \left[x_{11}A_1 + x_{21}A_2 + x_{31}A_3 + \ldots\ldots\ldots + x_{N1}A_N\right]2^{-1} \\ & + \left[x_{12}A_1 + x_{22}A_2 + x_{32}A_3 + \ldots\ldots\ldots + x_{N2}A_N\right]2^{-2} \\ & \bullet \\ & + \left[x_{1(B-2)}A_1 + x_{2(B-2)}A_2 + x_{3(B-2)}A_3 + \ldots\ldots\ldots + x_{N(B-2)}A_N\right]2^{-(B-2)} \\ & + \left[x_{1(B-1)}A_1 + x_{2(B-1)}A_2 + x_{3(B-1)}A_3 + \ldots\ldots\ldots + x_{N(B-1)}A_N\right]2^{-(B-1)} \end{aligned}$$

LSB of x1

LSB of xN

Use ROM as table lookup

- We can avoid any multiplication by table lookup:
  - Use (x_{1i}, x_{2i}, x_{3i}, ... x_{Ni}) as address to a ROM
  - Store pre-calculated partial product for each line in ROM:

$$\Phi(x_{1i}, x_{2i}, x_{3i}..., x_{Ni}) = A_1 x_{1i} + A_2 x_{2i} + A_3 x_{3i} + ... + A_N x_{Ni}$$

- We can calculate y three operations: ROM lookup, shift, add/subtract:

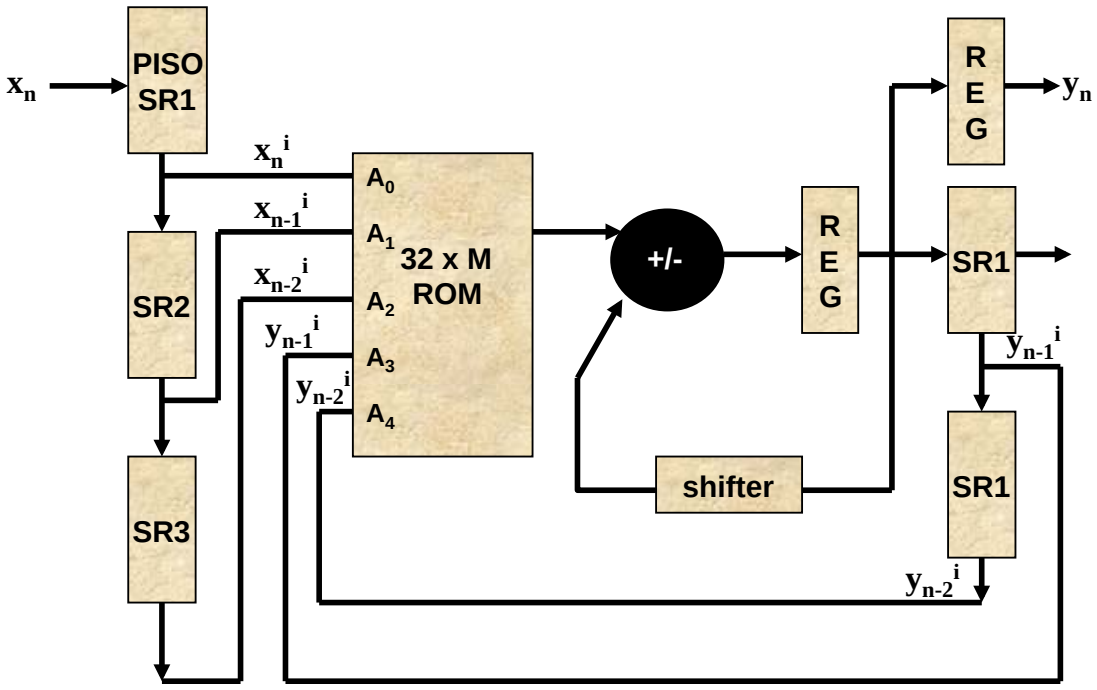
ROM lookup

$$y = -\Phi(x_{10}, x_{20}, x_{30}..., x_{N0}) + \sum_{i=1}^{B-1} 2^{-i}\Phi(x_{1i}, x_{2i}, x_{3i}..., x_{Ni})$$

Add/Sub

Shifting

Implementation of Biquad



Testing your design

- You should design a notch filter with the following pole-zero locations in the z-plan: