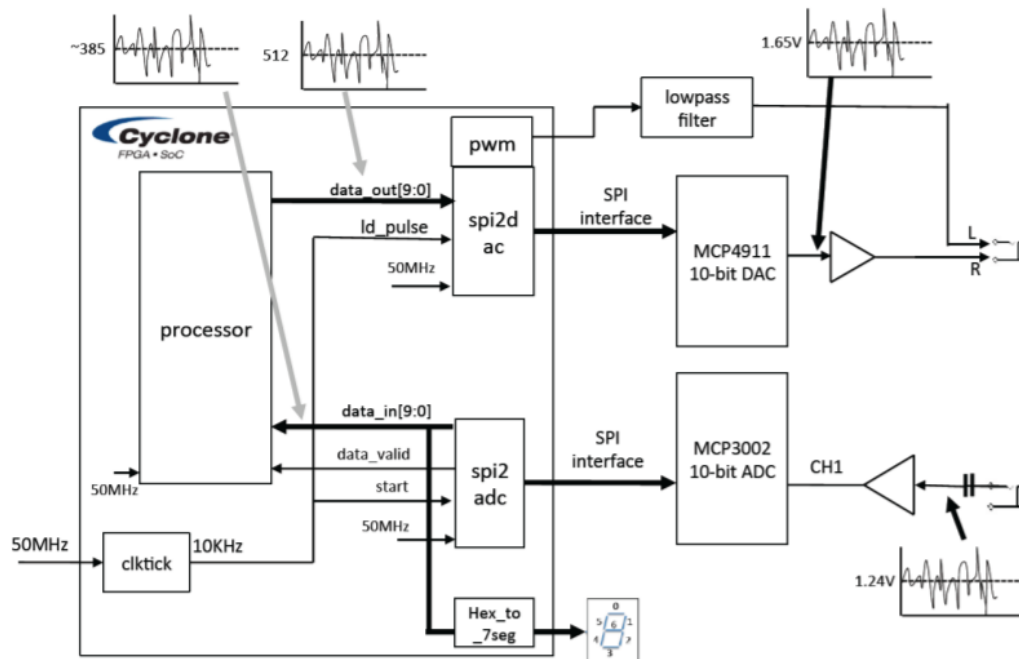


## Verilog Experiment - Part 4

### Experiment 16



The allpass processor module works in the following way:

1. Corrects the ADC converter data (which uses offset binary with 0V represented by a value of ~385), but subtracting the offset from data\_out[9:0] to obtain a 2's complement value x[9:0].
2. Connects X to Y, i.e. does nothing and hence "allpass".
3. Converts the Y value from 2's complement to offset binary for the DAC. The offset now is at 512 as shown below.

```

9  module processor (sysclk, data_in, data_out);
10
11      input          sysclk;          // system clock
12      input [9:0]    data_in;         // 10-bit input data
13      output [9:0]   data_out;        // 10-bit output data
14
15      wire           sysclk;
16      wire [9:0]     data_in;
17      reg [9:0]      data_out;
18      wire [9:0]     x,y;
19
20      parameter      ADC_OFFSET = 10'h181;
21      parameter      DAC_OFFSET = 10'h200;
22
23      assign x = data_in[9:0] - ADC_OFFSET;    // x is input in 2's complement
24
25      // This part should include your own processing hardware
26      // ... that takes x to produce y
27      // ... In this case, it is ALL PASS.
28      assign y = 4*x;
29
30      // Now clock y output with system clock
31      always @(posedge sysclk)
32          data_out <= y + DAC_OFFSET;
33
34  endmodule

```

My multiplication processor works in the same way but multiplies the 2's complement input by 4 in order to get a louder output. The spi2adc uses dot notation so that signal names inside the module connect to outside the module in any order which is much safer. Outside the processor the code is the same, so it is very flexible and reusable.

## Experiment 17

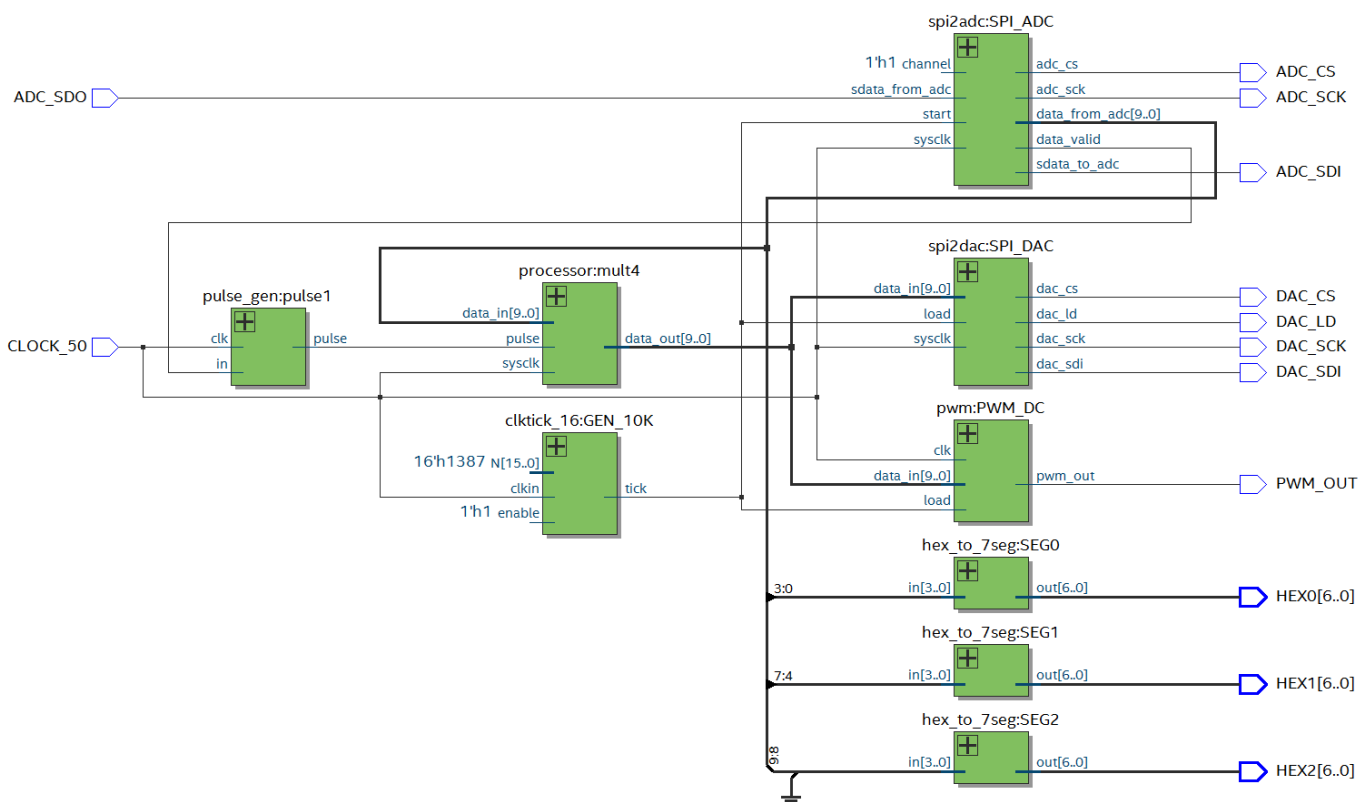
```

9  module processor (sysclk, pulse, data_in, data_out);
10
11      input          sysclk;          // system clock
12      input [9:0]    data_in;          // 10-bit input data
13      output [9:0]   data_out;         // 10-bit output data
14      input          pulse;
15
16      wire           sysclk;
17      wire [9:0]     data_in;
18      reg [9:0]       data_out;
19      wire [9:0]      x,y;
20      wire            full;
21      wire            dout;
22      wire [9:0]      q;
23
24      parameter      ADC_OFFSET = 10'h181;
25      parameter      DAC_OFFSET = 10'h200;
26
27      assign x = data_in[9:0] - ADC_OFFSET;    // x is input in 2's complement
28
29      // This part should include your own processing hardware
30      // ... that takes x to produce y
31      // ... In this case, it is ALL PASS.
32      FIFO            fifo1 (sysclk, x, dout && pulse, pulse, full,q);
33      dflip1          dflip1(sysclk, full, dout);
34
35      assign y = x*4 + {q[9], q[9:1]};
36
37      // Now clock y output with system clock
38      always @(posedge sysclk)
39          data_out <= y + DAC_OFFSET;
40
41  endmodule

```

This new processor produces a single echo on an audio input using an 8192 x 10 bit FIFO. The FIFO delays the output by 0.8192s until it is full and then sends a full signal, thus the writing of the samples thereafter become synchronous. The current input is added to the delayed input giving the echo effect. The echo is attenuated by 0.5 or 0.25 so that upon addition it does not saturate the signal. The sampling frequency is 10KHz.

The top- level stays the same, only the processor changes.



## Experiment 18

```

module processor (sysclk, pulse, data_in, data_out);

    input                sysclk;           // system clock
    input [9:0]          data_in;          // 10-bit input data
    output [9:0]          data_out;        // 10-bit output data
    input                pulse;

    wire                 sysclk;
    wire [9:0]           data_in;
    reg [9:0]            data_out;
    wire [9:0]           x,y;
    wire                 full;
    wire                 dout;
    wire [9:0]           q;

    parameter            ADC_OFFSET = 10'h181;
    parameter            DAC_OFFSET = 10'h200;

    assign x = data_in[9:0] - ADC_OFFSET;    // x is input in 2's complement

    // This part should include your own processing hardware
    // ... that takes x to produce y
    // ... In this case, it is ALL PASS.
    FIFO                 fifo1 (sysclk, y, dout && pulse, pulse, full,q);
    dflip                dflip1(sysclk, full, dout);

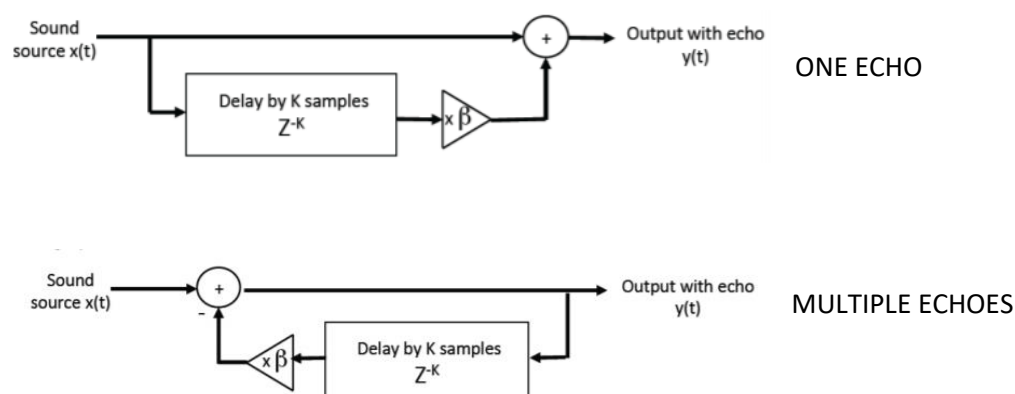
    assign y = x*4 - {q[9], q[9:1]};

    // Now clock y output with system clock
    always @(posedge sysclk)
        data_out <= y + DAC_OFFSET;

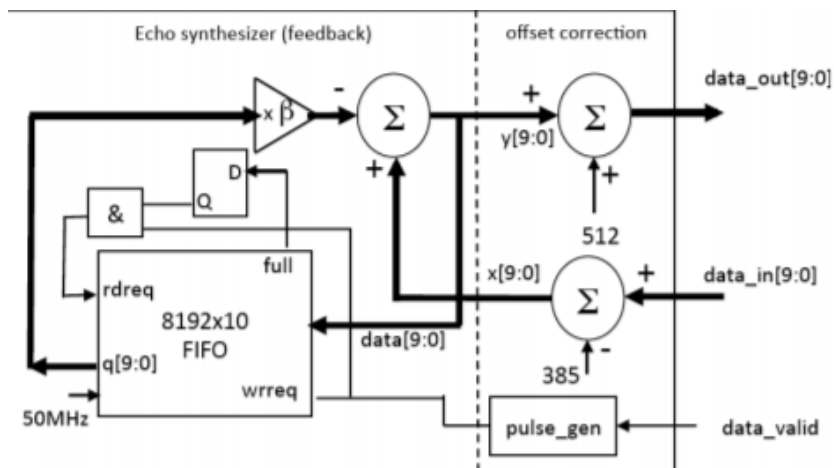
endmodule

```

This processor is very similar to the previous one, except that the delayed output is fed back in and subtracted from the current output giving multiple echoes. The difference is highlighted in the block diagrams below.



### Simplified Processor Block Diagram



One can see how this can be changed to obtain experiment 17 processor, as described above. I didn't show RTL block diagram as too difficult to easily see difference between experiment 17 and 18.