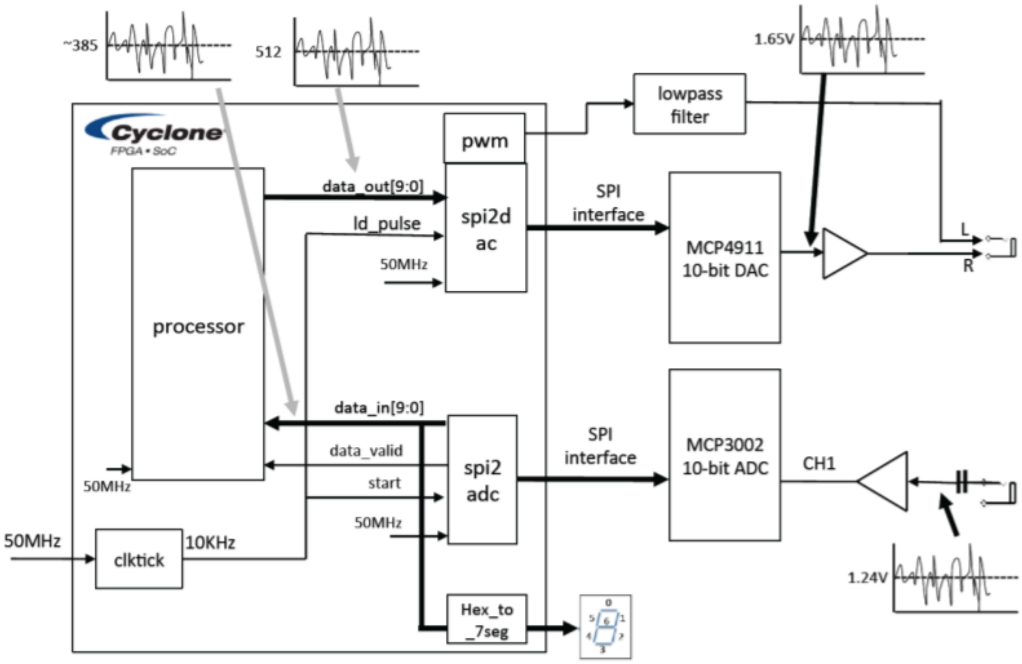
Verilog Experiment - Part 4

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.

```
module processor (sysclk, 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

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

        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.
        assign y = 4*x;

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

endmodule
```

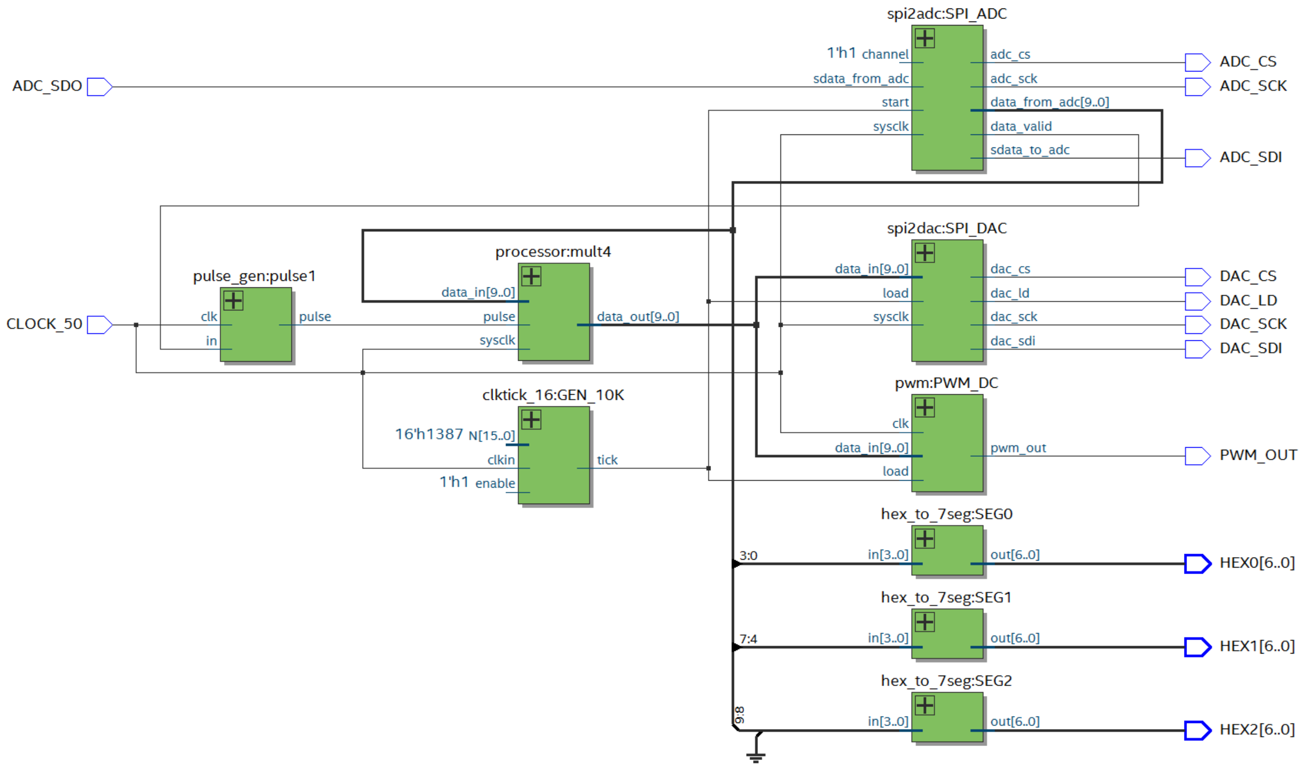
My multiplication processor works in the same way but multiplies the 2's compliment input by 4 in order to get a louder output. The spi2adc uses dot notation so that signal names inside the module connect to outside the module in any order which is much safer. Outside the processor the code is the same,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

Experiment 17

```
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, x, dout && full, pulse, full,q);
	dflip				dflip1(sysclk, pulse, 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 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.

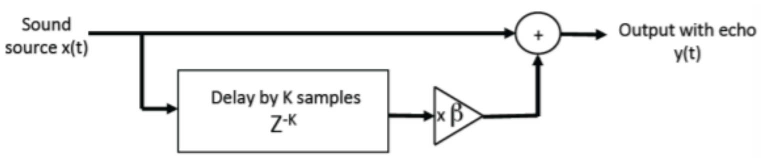
The top- level stays the same, only the processor changes.



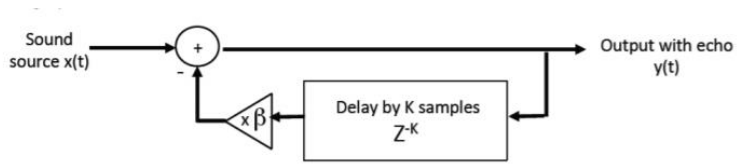
<u>Experiment 18</u>

```
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 && full, pulse, full,q);
        dflip                           dflip1(sysclk, pulse, 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. It is subtracted not added otherwise you would get positive feedback which causes instability. The difference is highlighted in the block diagrams below.

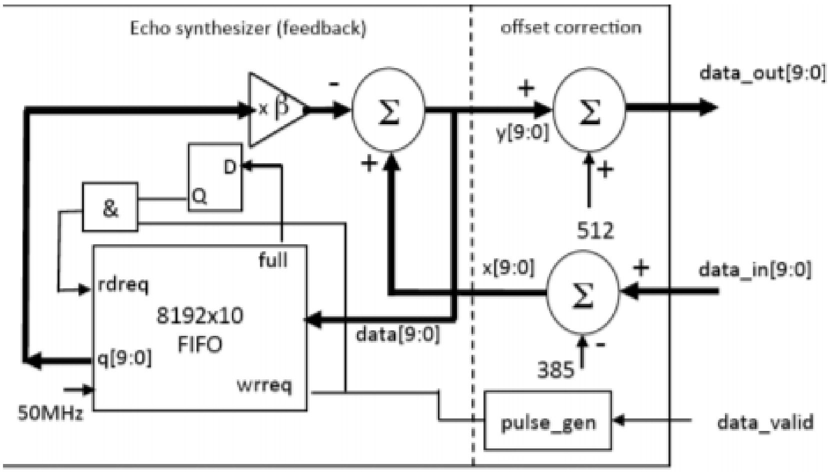


ONE ECHO



MULTIPLE ECHOES

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. There is something wrong in this diagram. The full and pulse gen signal should be swapped, so full goes into and gate and pulse goes to D flip-flop otherwise, the processor would give values every other cycle, this was also true in exercise 17.

The offset is taken away before and then added after processing, so that during processing arithmetic can be implemented on a 2's compliment number in order to obtain correct values.