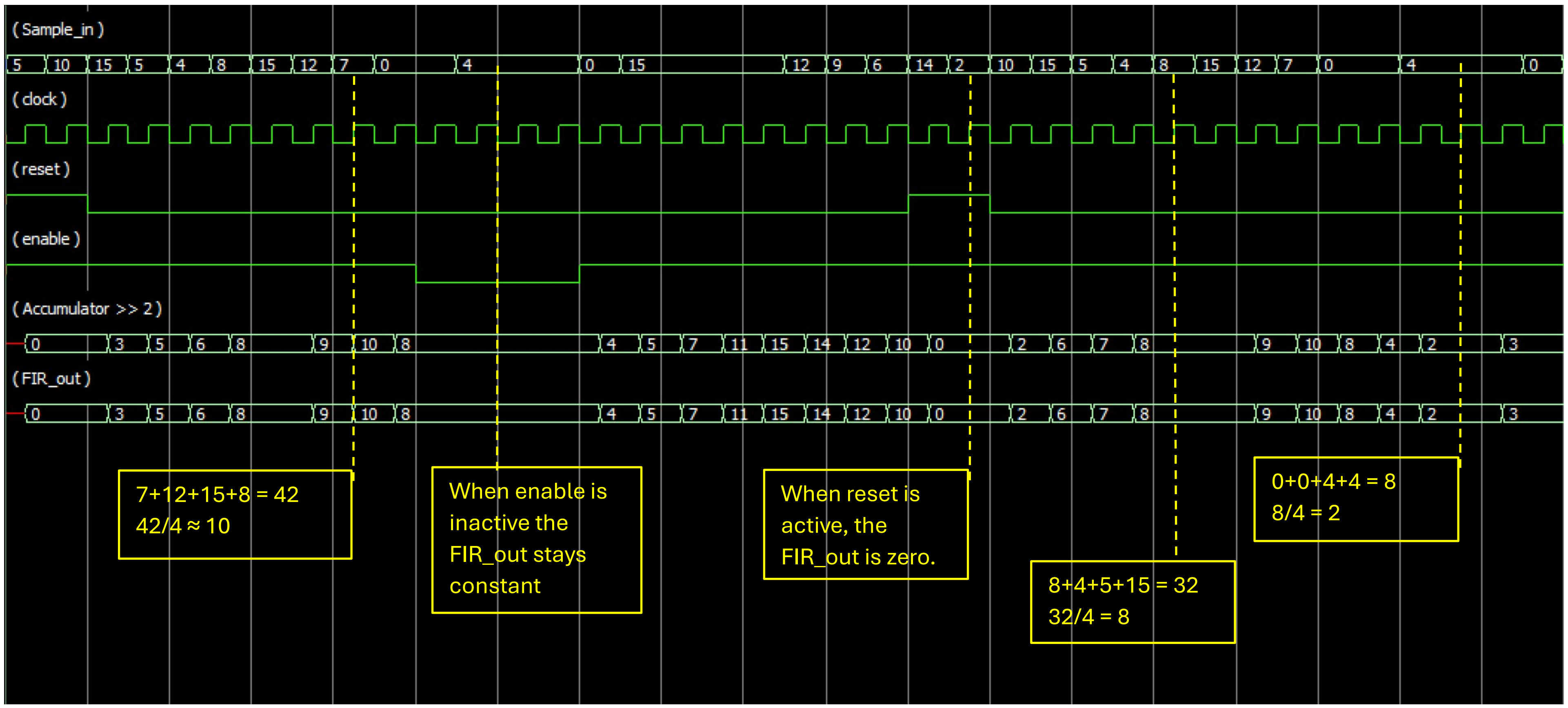
```
/*----------------------------------------------------------------------------------------
                 Lesson 9: Digital Signal Processing Applications
-----------------------------------------------------------------------------------------
Moving Average Filter: (Special FIR)
-------------------------------------
One of the special FIR filters is the moving average filter of overlapping windows of
4 input samples. The module should have reset and enable input signals. The code should
be parametrizable: The order of the filter should be one parameter
(starting with just 4 samples). We will restrict the FIR to always have an order
that is a power of 2 (2, 4, 8, 16 and so on). This allows us to use the right shifting
to implement division (Reference: lesson 3). The input sample word-Size should also be
a parameter (starting with 4 bits). Calculate the size of the output and the internal
registers correspondingly for the moving average filter.
Create a test-bench that tests the functionality of the Filter. The test should include an
impulse response (one sample equal to 4 embedded between 4 samples of zeros,
the reset case, the disabled and enabled cases, and maximum possible sample
values, as well as other test vectors. Investigate the effect of the rounding on the
moving average filter.  Combine your code, and simulation results in one pdf file.
----------------------------------------------------------------------------------------*/
```

```
module FIR_MovingAverage ( FIR_out, Sample_in, clock, reset, enable);

parameter FIR_order     = 4;
parameter sample_size   = 4;                      // maximum sample value is 15
parameter word_size_out = sample_size;            // Average size = the sample size
parameter Acc_size      = FIR_order*sample_size;
parameter shift_steps   = $clog2(FIR_order);      // division by right shift

output     [word_size_out -1: 0] FIR_out;

input      [sample_size   -1: 0] Sample_in;
input                            clock;
input                            enable, reset;

reg      [Acc_size-1: 0]     Acc;                          // Adding the samples
reg      [sample_size -1: 0] Sample_Array [1: FIR_order];

assign   FIR_out = Acc[sample_size-1 :0];
integer  k,c;

//Combinational logic:
always @ (Sample_Array)
begin
Acc = 0;
for (c=1; c<=FIR_order; c=c+1) begin
    Acc = Acc + Sample_Array[c];
    end
Acc = Acc >> shift_steps;
end
```

Add all the samples while keeping your code parametrizable for any number of samples, and the division using right shifting for powers of 2.

```
// Sequential logic:
always @ (posedge clock)
     if (reset)
         begin
         for (k = 1; k <= FIR_order; k = k+1)
              Sample_Array[k] <= 0;
         end
   else if (enable) begin

         Sample_Array [1]  <= Sample_in;
           for (k = 2; k <= FIR_order; k = k+1)
           Sample_Array[k] <= Sample_Array[k-1];
         end

endmodule
```



(Sample_in)
5 10 15 5 4 8 15 12 7 0 4 0 15 12 9 6 14 2 10 15 5 4 8 15 12 7 0 4 0
(clock)
(reset)
(enable)
(Accumulator >> 2)
0 3 5 6 8 9 10 8 4 5 7 11 15 14 12 10 0 2 6 7 8 9 10 8 4 2 3
(FIR_out)
0 3 5 6 8 9 10 8 4 5 7 11 15 14 12 10 0 2 6 7 8 9 10 8 4 2 3
7+12+15+8 = 42
42/4 ≈ 10
When enable is inactive the FIR_out stays constant
When reset is active, the FIR_out is zero.
8+4+5+15 = 32
32/4 = 8
0+0+4+4 = 8
8/4 = 2