LAB Assignment 2

Moving Average Filter

Dr. Ahmed Elhossini

Introduction

In this LAB assignment you are going to implement a system based on the multi-adder circuits using VHDL. The moving average filter system should be developed using Xilinx ISE, simulated and simulated after synthesis and place & route. Some components from the previous LAB assignment can be used. Integration with Zynq SoC is optional.

Pre-requisites

1. Basic Knowledge of VHDL.

Objectives:

1. Model Moving Average Filter using VHDL.
2. Simulate the system using ISE simulator (Behavioral/Post Place & Route)
3. Synthesis each circuit using the FPGA flow.
4. Report the area and critical path delay of each component.
5. Integrate the System with Zynq SoC as described in tutorials 1 and 2 (optional).

Equipment and Tools

1. Xilinx ISE 14.5 tool chain or later versions.
2. Xilinx EDK tool chain or later versions.
3. The ZedBoard.

Lab Time:

4 Weeks from November 23th until December 21th, 2016.

Introduction

Moving Average Filter is a type of Finite Impulse Response (FIR) filters that is used in digital signal processing application to smooth and remove noise from signal. The filter can be represented by Equation 1.

$$f(t) = \frac{1}{n} \cdot \sum_{i=0}^{n-1} s(t-i)$$

Equation 1

The filter adds **n** samples from the signal **S** at each time step and generates one output of the result. At each point **n** samples of the signal are stored and added by the internal circuit of the adder.

The effect of the filter is shown in Figure 1. In this LAB assignment you are required to build the moving average filter that will generate the same results as shown in Figure

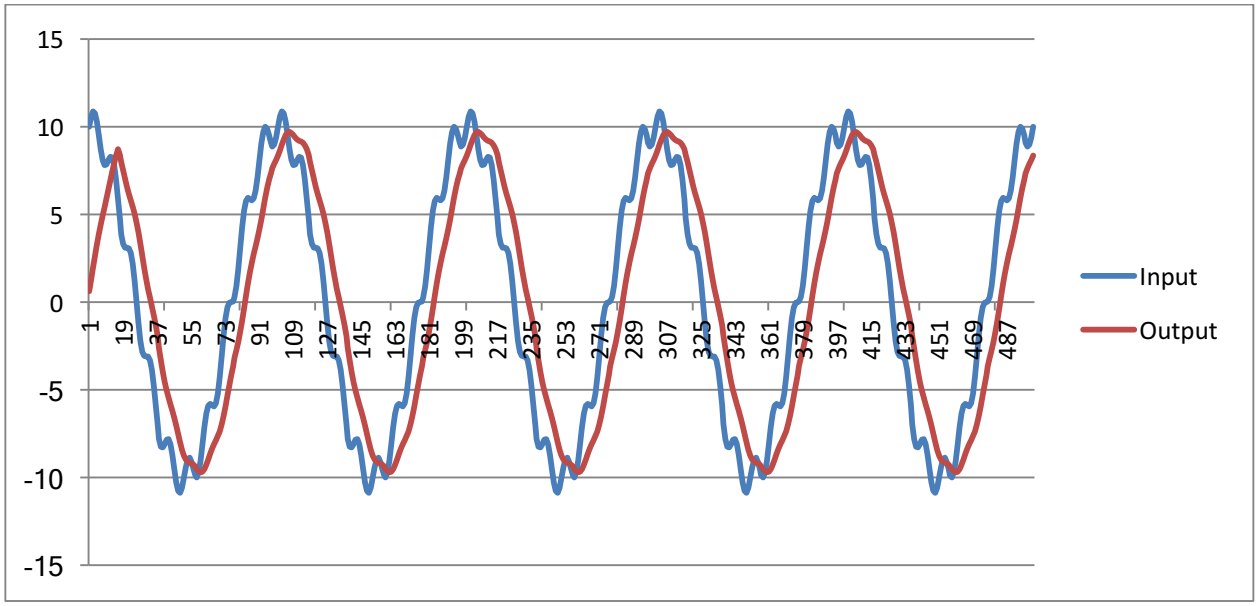

Figure 1 Moving Average Filter Response

Lab Steps

Part 1: Carry Save Adder

Implement the carry save adder circuit shown in *Figure 2 14 bits 4:2 carry save adder*. Build a project using ISE that implement this circuit. The 4:2 counter is composed of two full adders as shown in *Figure 3*. Reuse the Full Adder component from LAB assignment 1 to build this circuit.

1. The system should have 4 inputs each is 11 bits. The most significant bit is a sign bit.
2. The system should add sign extension bits (Red dots in *Figure 2*).
3. The output is two 15 bits signed numbers.
4. Simulate the system using ISE simulator.

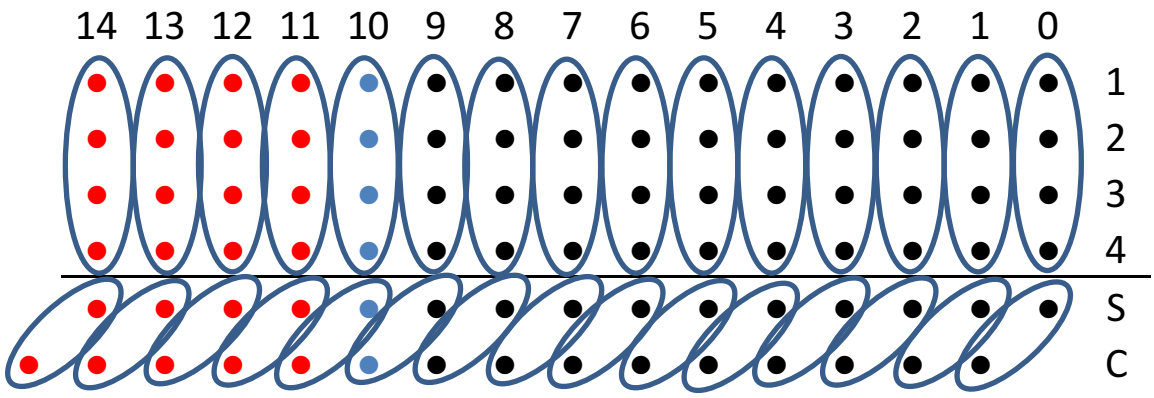

Figure 2 14 bits 4:2 carry save adder

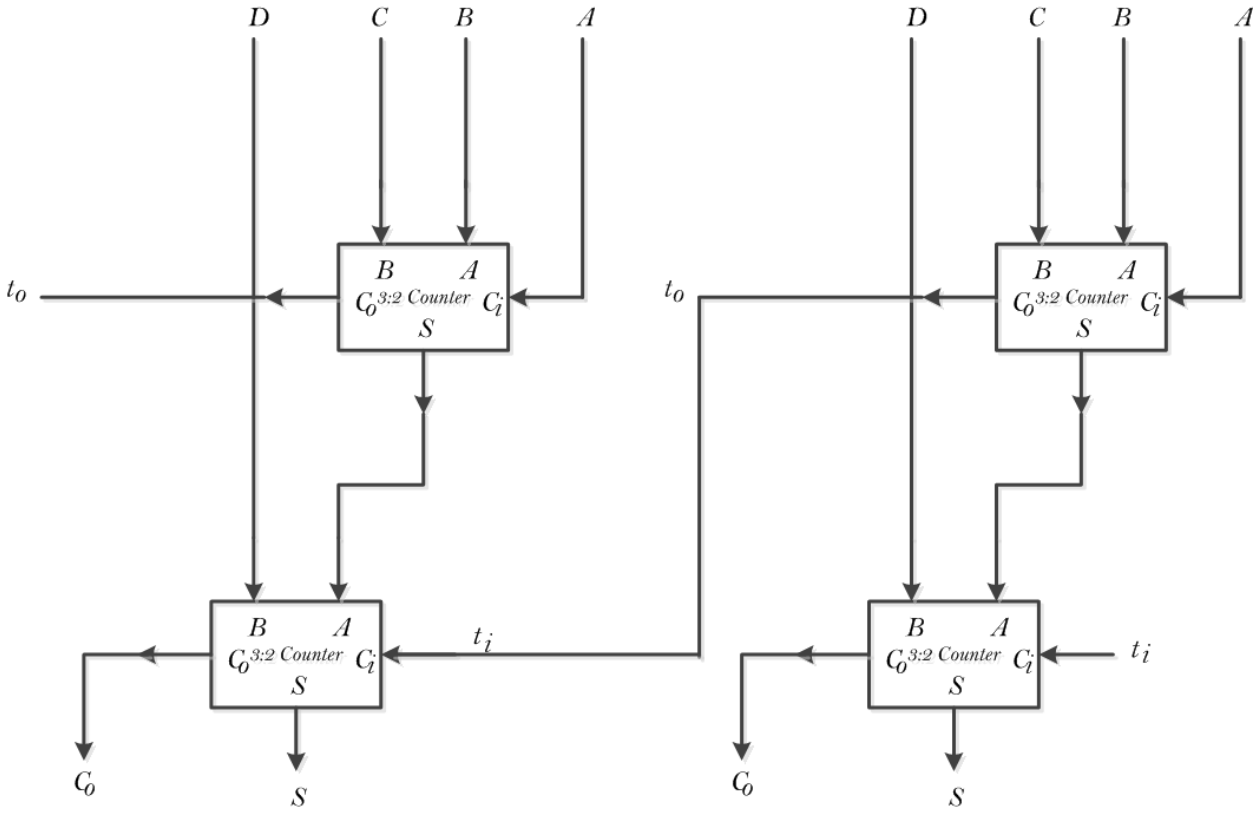

Figure 3 4:2 Counter

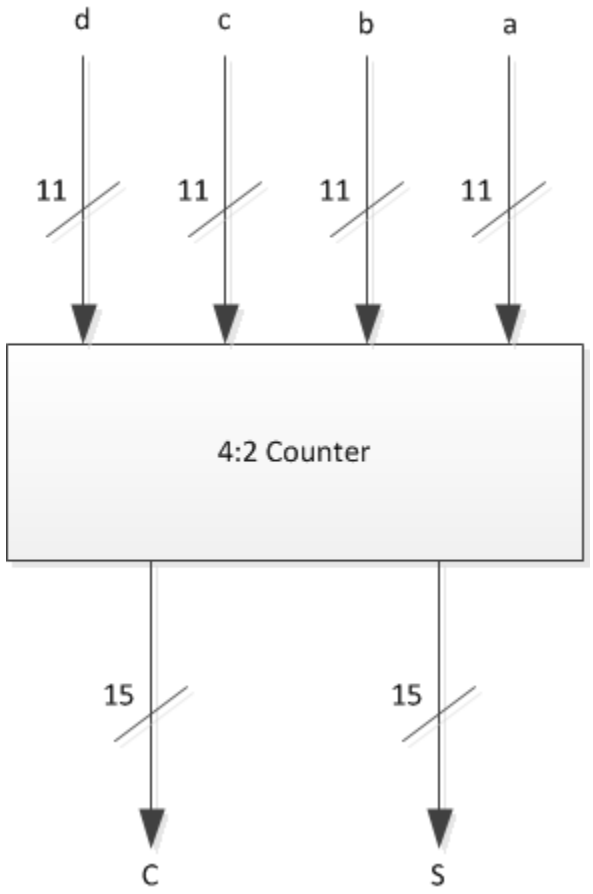

Figure 4 Carry Save Adder Module (4 Inputs)

Part 2: 16 Input Carry Save Adder

Use the 4 inputs Carry save adder module of Part 1 to build a 16 Input Carry-Save-Adder Network as shown in Figure 6. Use the fastest adder circuit you implemented in LAB assignment 1 to implement the final adder circuit.

The 11 bit format is shown in Figure 1. Each 11 bits input should follow this format.

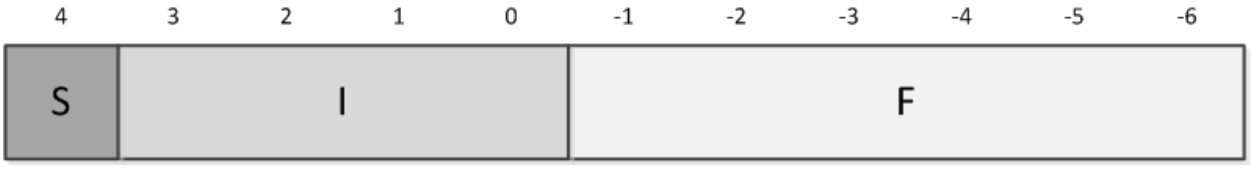

Figure 5 Input number format

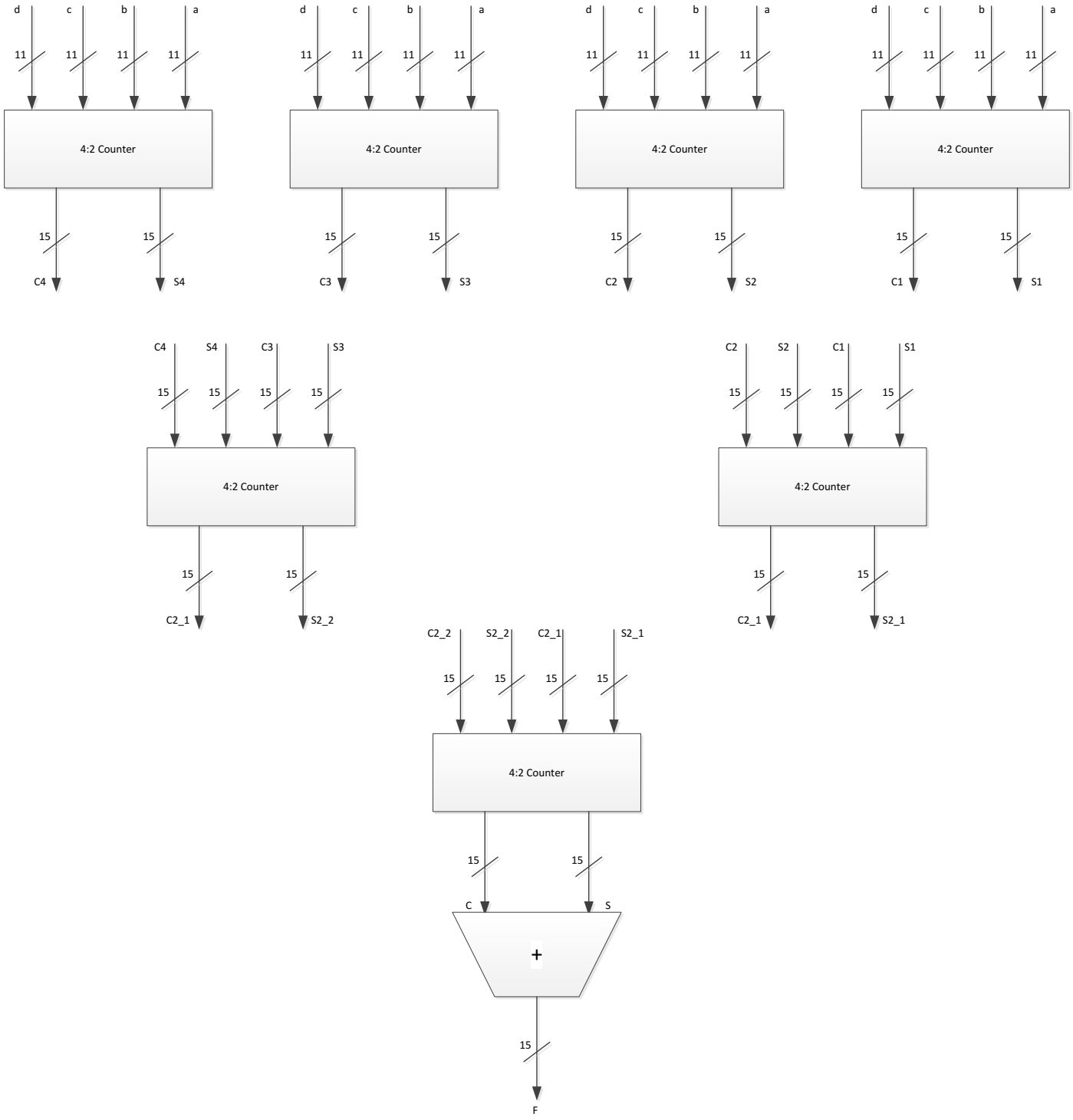

Figure 6 16 Carry Save Adder

Part 3: Register Array

As discussed in the lectures, registers are required to store samples temporarily for processing. The register array structure is shown in Figure 7. VHDL code for a single 8bit register is shown in Figure 8.

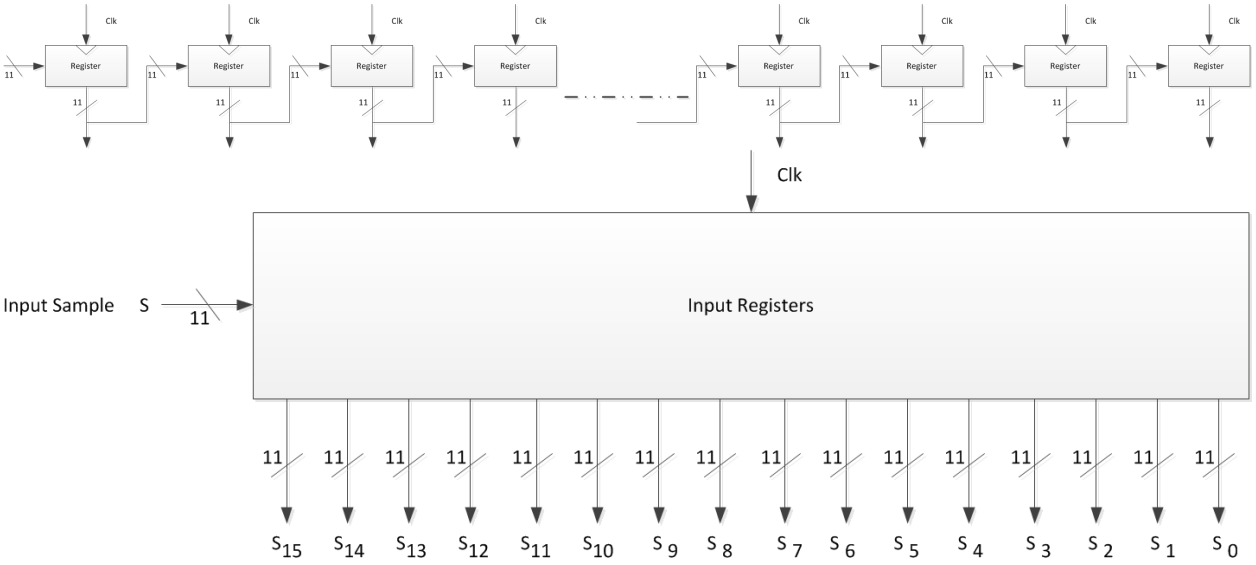

Figure 7 Register Array

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Reg8bits is
    port ( d : in  STD_LOGIC_VECTOR (7 downto 0);
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           q : out  STD_LOGIC_VECTOR (7 downto 0));
end Reg8bits;
architecture Behavioral of Reg8bits is
begin
        ureg : process (rst, clk)
          begin
                    if rst = '1' then
                              q <= (others=>'0');
                    else
                              if clk'event and clk='1' then
                                        q <= d;
                              end if;
                    end if;
          end process;
end Behavioral;
```

Figure 8 VHDL code for 8 bit register

For this part perform the following steps:

a. Modify the code in Figure 8 for 11 bits registers.
b. Model the register array shown in Figure 7 using VHDL.
c. Simulate the register array to verify its functionality.

Part 4: Building the Moving Average Filter

Complete the design of the moving average filter by adding register array to store samples for processing as shown in *Figure 9*. The overall design should have an entity interface that looks like the code shown in *Figure 10*.

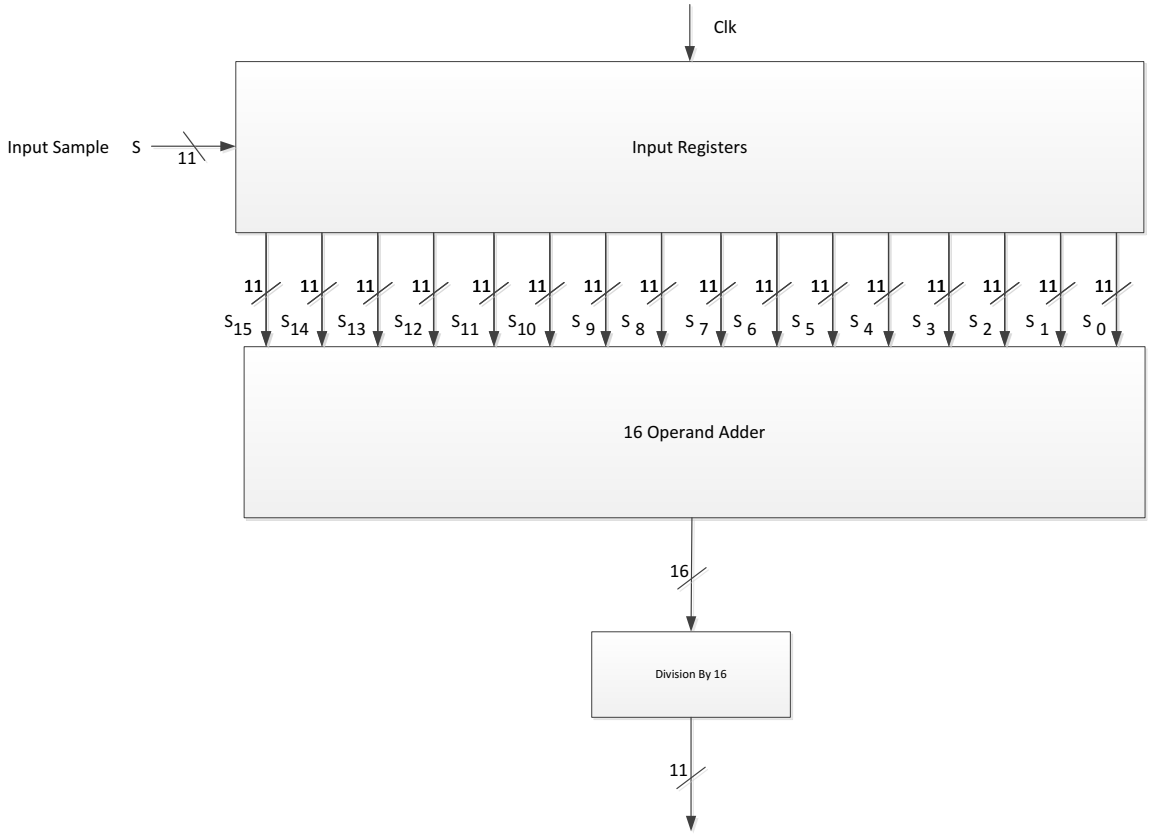

Figure 9 Complete Moving-average Filter

For this part perform the following tasks:

a. Model the complete system using VHDL.

b. Simulate the design using ISim. For this step two files are provided with this document. The first is **movingaverage_tb.vhd** which is a VHDL test-bench to simulate the design. It reads samples from a file **input.txt** (also provided with this document) and apply these samples to the system after each clock. The output of the filter is then stored in another file – **output.txt.**

c. Using any plotting tool, plot the input and output and verify it is performing as shown in *Figure 1*.

d. Implement the complete design and calculate the maximum operating frequency. In the lecture the requirement was 10MHz. Verify that we achieve this target.

e. If the frequency target is not met, modify the design by applying techniques we studied in the course to increase the operating frequency.

f. Perform Post-Translate simulation (Post Synthesis) to verify the functionality after simulation.

```
entity movingaverage is
    port (
    sin : in STD_LOGIC_VECTOR (10 downto 0);
    clk : in STD_LOGIC;
    sout : out STD_LOGIC_VECTOR (10 downto 0);
    rst : in STD_LOGIC);
end movingaverage;
```

Figure 10 Entity interface for the moving average filter

Part 5: Integration with Zynq-SoC

This part is optional. The integration with Zynq SoC requires building a custom IP-core that will be connected to the ARM core using the AXI-bus as illustrated in Tutorial 2. You can use one of software registers for input data, and another one for the output data. The clock input needs to be connected to a clock source. For this we can also use one bit from these registers and toggle it from the software side. Also you can use on-board clock signal, but you will have to synchronize data input/output with this clock.

Deliverables:

1. **VHDL code for all modules implemented.**
2. **Simulation results for each part (Behavior and Post Place & Route)**
    a. **Wave form for each stage.**
    b. **Input and output files and their plot for the final design as shown in Figure 1.**
3. **Frequency analysis and design modifications (and modification you made to the design presented in this document should be reported).**