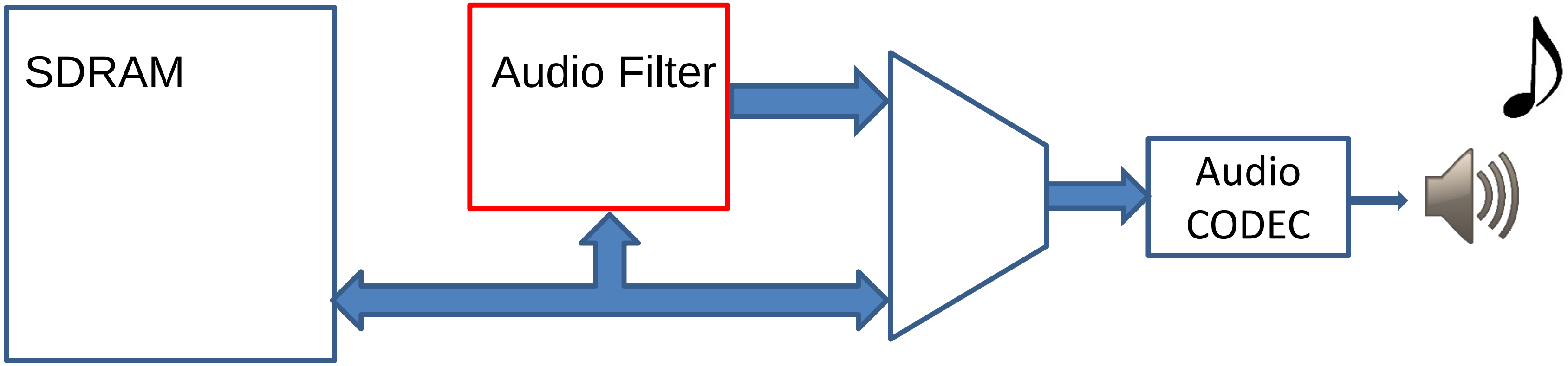
Lab 8 Guide

Purpose

- The overall purpose of Lab 8 is to implement a low-pass and a high pass digital filter in VHDL.

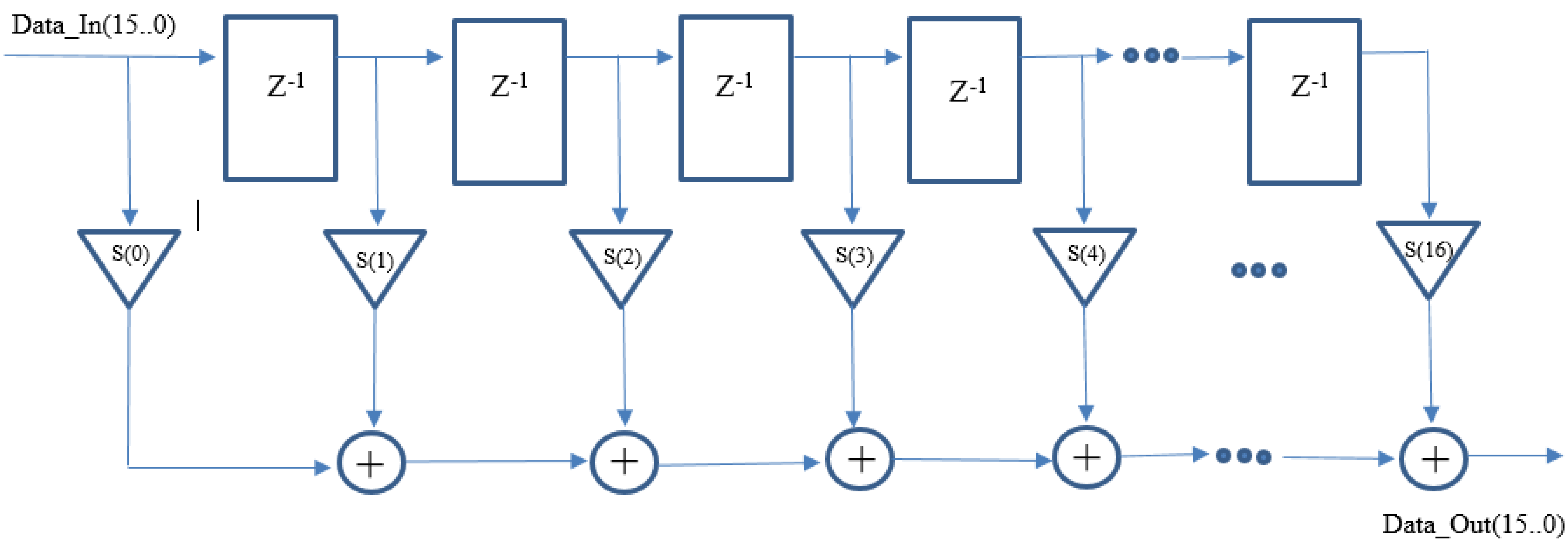


Filter Design

- Open a new .vhd file for each filter
  - **It needs to be in its own file for simulation**

```
entity low_pass_filter is
  port (
        clk        : in std_logic;     -- CLOCK_50
        reset_n    : in std_logic;     -- active low reset
        data_in    : in signed(15 downto 0); --Audio sample, in 16 bit fixed point format (15 bits of assumed decimal)
        filter_en  : in std_logic;  --This is enables the internal registers and coincides with a new audio sample
        data_out   : out signed(15 downto 0) – This is the filtered audio signal out, in 16 bit fixed point format
  );
end low_pass_filter;
```

Filter Design



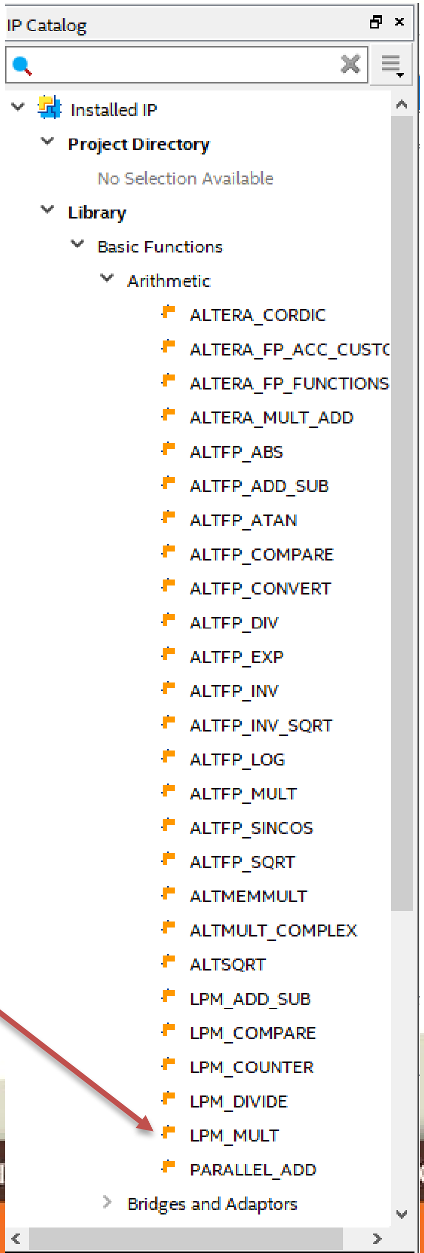
Filter Design - Multipliers

- Each triangle represents a multiplier.
  - **The signal coming into the multiplier should be multiplied by the corresponding filter coefficient**
  - **Make these constants**

	Low Pass			High Pass	
Coefficient	Value	16-bit fixed point		Value	16-bit fixed point
S(0)	0.0025			0.0019	
S(1)	0.0057			-0.0031	
S(2)	0.0147			-0.0108	
S(3)	0.0315			0.0	
S(4)	0.0555			0.0407	
S(5)	0.0834			0.0445	
S(6)	0.1099			-0.0807	
S(7)	0.1289			-0.2913	
S(8)	0.1358			0.5982	
S(9)	0.1289			-0.2913	
S(10)	0.1099			-0.0807	
S(11)	0.0834			0.0445	
S(12)	0.0555			0.0407	
S(13)	0.0315			0.0	
S(14)	0.0147			-0.0108	
S(15)	0.0057			-0.0031	
S(16)	0.0025			0.0019	

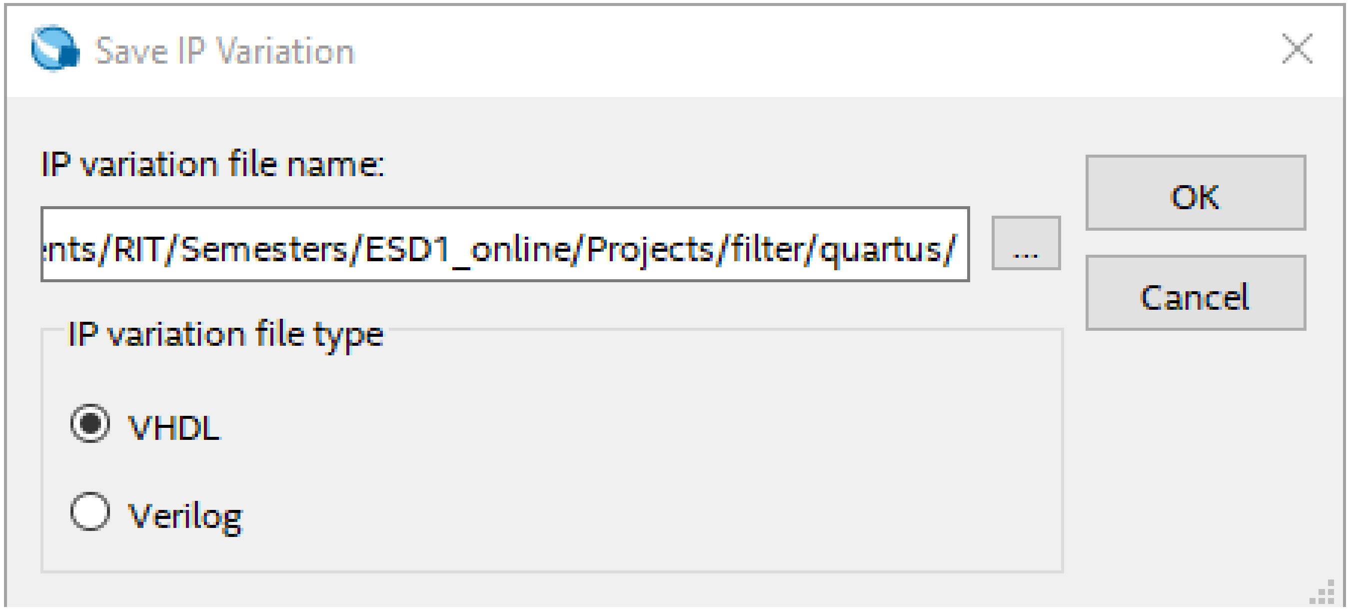
Filter Design - Multipliers

- Create the multiplier using the MegaWizard
- Choose LPM_MULT



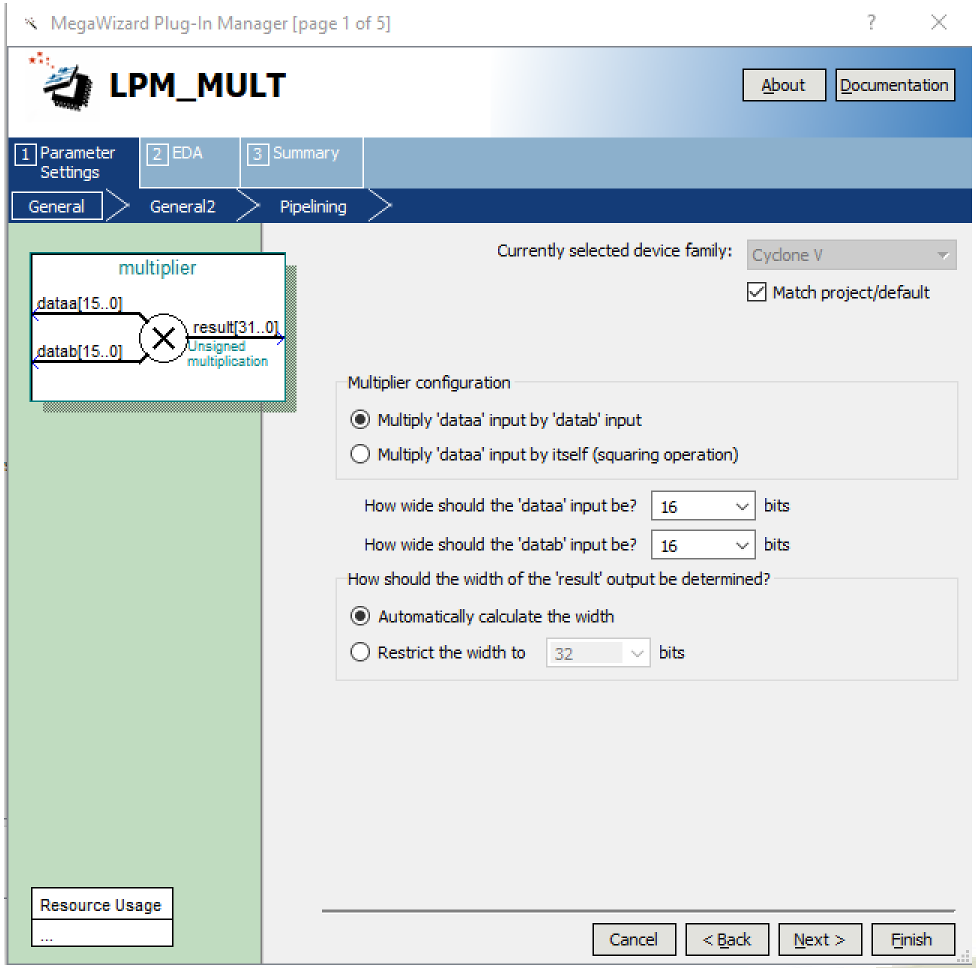
Filter Design- Multiplier

- Choose VHDL
- Give it a name



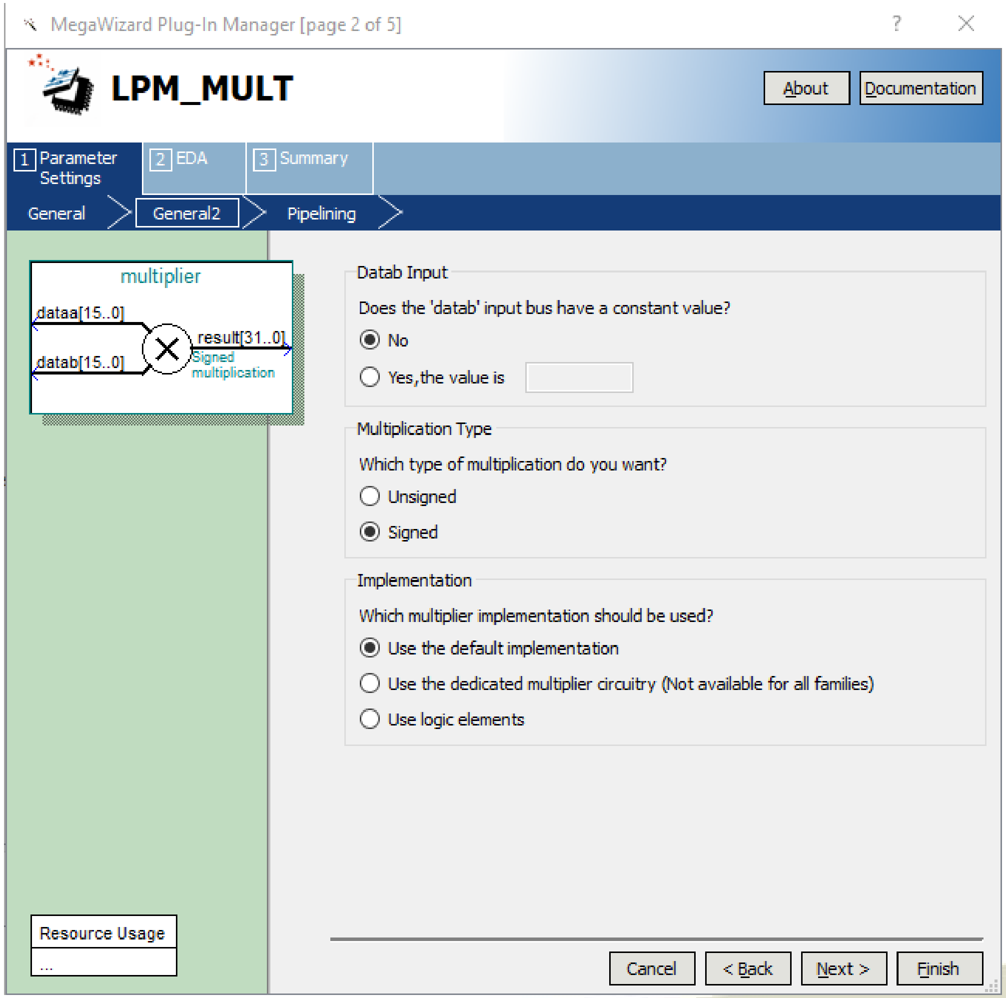
Filter Design- Multiplier

- Set the size to 16 bits



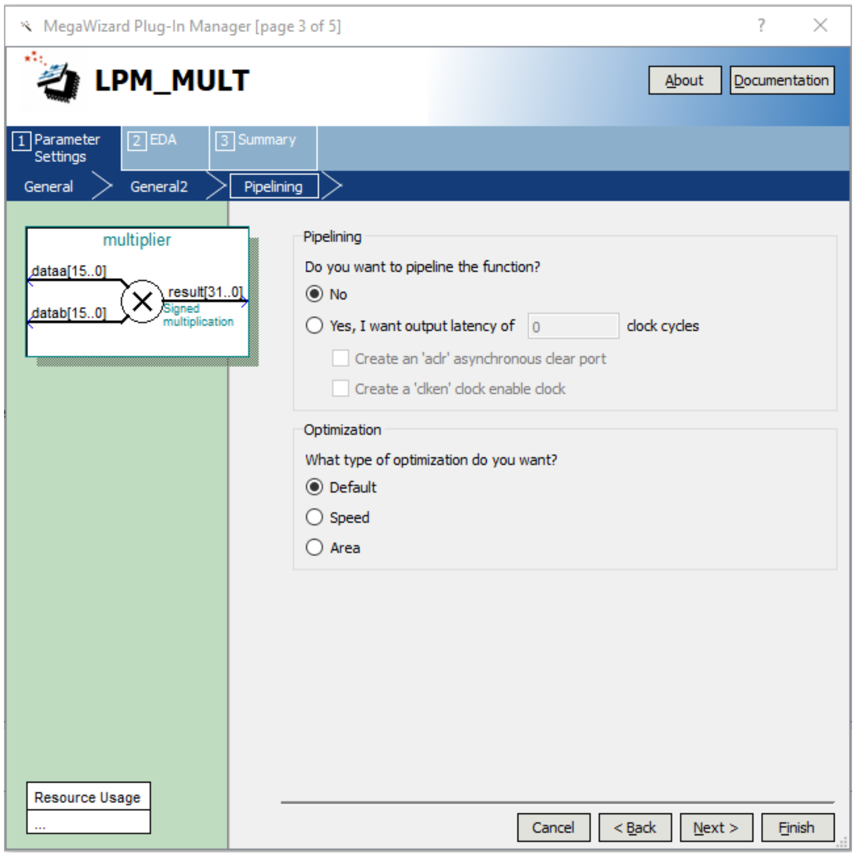
Filter Design - Multiplier

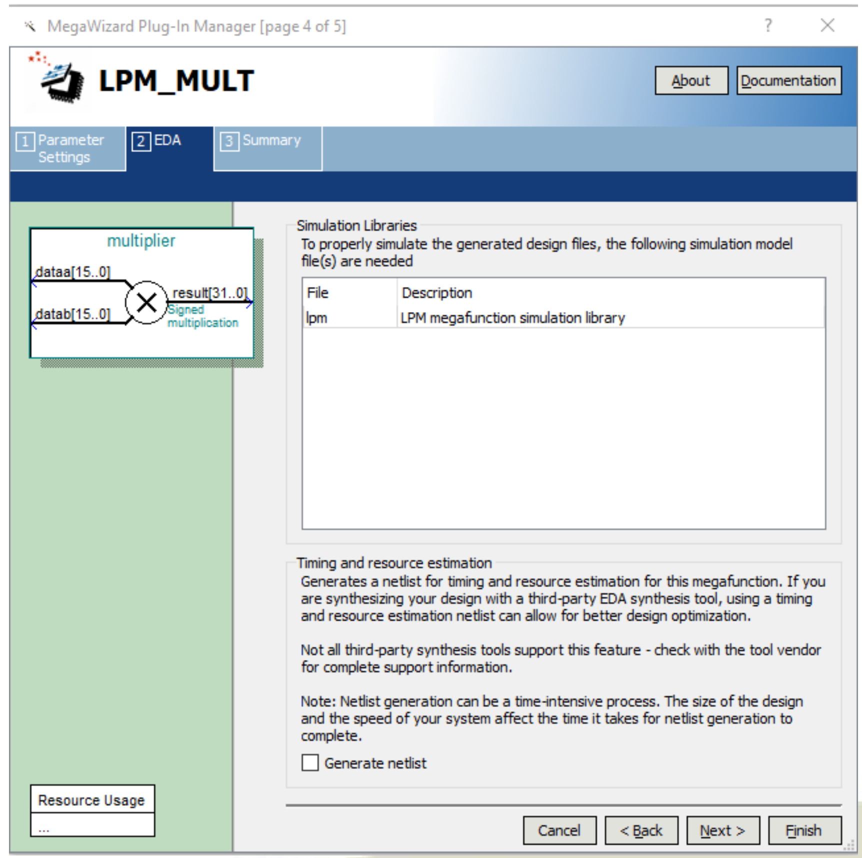
- Choose signed



Filter Design - Multiplier

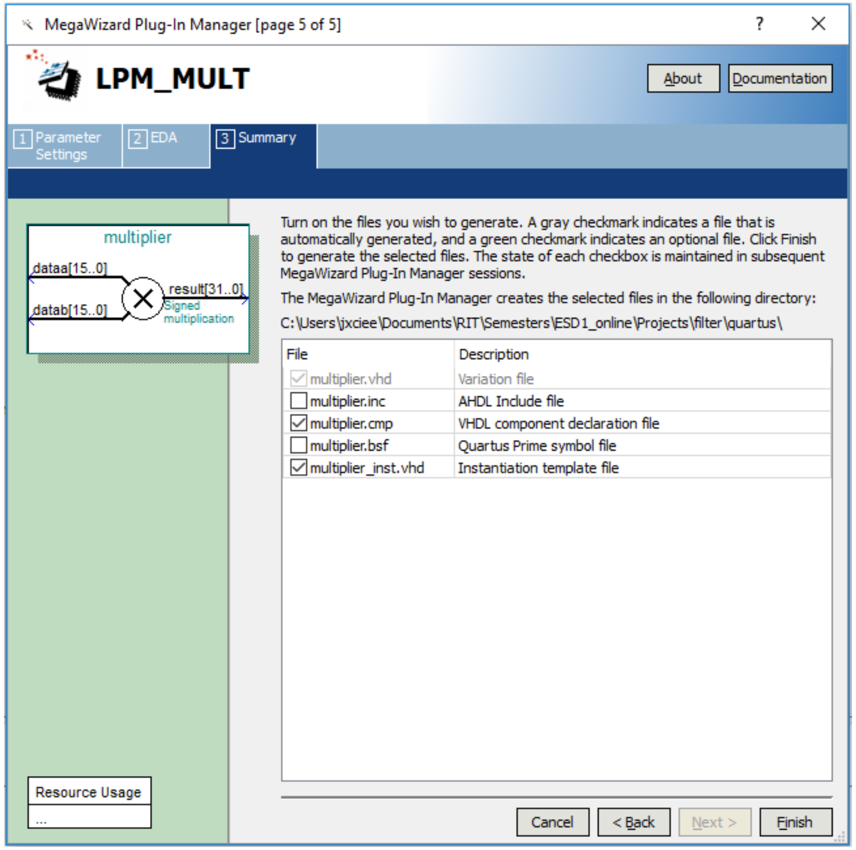
Accept defaults





Filter Design - Multiplier

- Choose to generate the multiplier_inst.vhd file.
- This contains the port maps
- After finishing, click yes to add the .qip file to your project



Filter Design - Multipliers

- Create one multiplier
  - **Declare it as a component**
  - **Instantiate 16 multipliers**
    - Port map dataa to the signal at the triangle’s input
    - Port map datab to the coefficient constants from the table above

Filter Design – adders

- Each circle is an adder
  - **You can use +**
  - **Be sure to include signed library**
    - use ieee.std_logic_signed.all;
  - **Because of signed library, sum can be the same length as the numbers being added**

Filter Design - delays

- Each block with a Z^{-1} is a 16 bit register with an enable
- Review DSD notes if you do not remember how to write one!
- The enable signal should be filter_en
- There are 16 registers
  - **A component may make it easier**

Fixed Point Notation

- Base-10 Integer Numbers
  - **Signed or unsigned integer**
  - **Uses numbers 0 to 9**
  - **Decimal point assumed to the far right**
  - **Each digit position represents power of 10**

Thousands (10^3) column

Hundreds (10^2) column

Tens (10^1) column

Ones (10^0) column

$1999._{10} = 1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 9 \times 10^0$

Fixed Point Notation

- Base-10 Real Numbers
  - **Signed or unsigned integer**
  - **Uses numbers 0 to 9**
  - **Decimal point fixed between digits**
  - **Each digit position still represents power of 10**
    - Digits right of the decimal point are negative power

Tens (10^1) column

Ones (10^0) column

Tenths (10^{-1}) column

Hundredths (10^{-2}) column

$$19.99_{10} = 1\text{x}10^1 + 9\text{x}10^0 + 9\text{x}10^{-1} + 9\text{x}10^{-2}$$

Fixed Point Notation

- Binary Integer Numbers
  - Uses numbers 0 and 1
  - Each digit position represents power of 2
  - Decimal point assumed to the far right

Eights (2^3) column

Fours (2^2) column

Twos (2^1) column

Ones (2^0) column

$$1001._2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Fixed Point Notation

- Binary Real Numbers
  - **Uses numbers 0 and 1**
  - **Each digit position represents power of 2**
  - **Decimal point assumed to the far right**

Twos (2^1) column

Ones (2^0) column

Halves (2^{-1}) column

Fourths (2^{-2}) column

$$10.01_2 \quad = 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

Fixed Point Notation

- Arithmetic Example
  - **Compare Base-10 addition to Base-2 addition**
    - Base-10: Each digit position ➔ power of 10
    - Base-2: Each digit position ➔ power of 2

Base-10	Base-2
1.25	1.01
+ 1.50	+ 1.10
2.75	10.11

- Both methods come up with the same answer

Fixed Point Notation

- Decimal point location
  - **Implicitly fixed in the binary number**
  - **0's used to pad unused position with fixed point number**
  - **Ex: 8-bit fixed-point number with 3 bit for fraction**
    - $00010.110_2 = 1*2^1 + 1*2^{-1} + 1*2^{-2} = 2 + 0.5 + 0.25$
    - $00010.110_2 = 2.75$
  - **Ex: 8-bit fixed-point number with 5 bit for fraction**
    - $000.10110_2 = 1*2^{-1} + 1*2^{-3} + 1*2^{-4} = 0.5 + 0.125 + 0.0625$
    - $000.10110_2 = 0.\ 6875$

Fixed Point Notation

- Fixed-point numbers
  - **Shifting Binary Numbers**
    - Divide by 2 occurs with each position shift right
    - Multiply by 2 occurs with each position shift left
  - **Represent a shifted version of real number**
    - Ex: 53 = 53.0 = 0x35 = 0011_0101
    - Ex: 26.5 ➔ 26.5 * 2 = 53.0 = 0x35 = 0011_010.1
- Conversion process to fixed-point
  - **Multiple real number by 2^n**
    - Where n is the number of bit allocated to fraction
  - **Ex: Represent 26.565 in fixed-point notation**
    - Using 16-bits with 8-bits for fractional part
    - $26.565 * 2^8 = 6800.64 \approx 6800$ = 0x1A90

Fixed Point Notation

- Signed fixed-point Example
  - **Represent -0.25 in Fixed point notation**
  - **Using 8-bits for fractional part & 1 bit for sign**
  - **8-bits for fraction**
    - 0.25 ➔ 0.25 * 2^8 = 0x40 = 0_0100_0000
    - Take 1’s complement: 0x1BF = 1_1011_1111
    - Add 1 for 1’s complement: 0x1C0 = 1_1100_0000
- Fixed Point Notation in VHDL
  - **Work with them in VDHL as signed type**
  - **Treat them the same as if they represented whole numbers**

Filter Design – Fixed Pt. Notation

- This filter uses 16-bit signed numbers with 15 bits after implied decimal point Example:
  - **S(6)= -0.0807**
  - **Multiply 0.0807 by 2^{15} (because there are 15 bits after decimal point)**
    - $0.0807 \times 2^{15} = 2644.3776 \approx 2644$
  - **Convert to hex**
    - 0x0A54
  - **Apply 2’s complement**
    - 0xF5AB + 1 = 0xF5AC

Note: Use the “Nerd Calc” app on your phone to make conversions to hex and 2’s complement easier

Filter Design – Size Matching

- The multiplier outputs will be 32 bits
- The additions are still 16 bits
- When you multiply two numbers that each have 15 decimal places, the product will have 30 decimal places
  - **The 32 bit output from the multiplier will have 2 bits to the left of the decimal point and 30 to the right**
  - **To maintain the 16-bit fixed pt format:**
    - You can remove the least significant 15 bits
    - You can also remove 1 bit from the left of the decimal point because it is just sign extension

MultOut_n <= MultOutFull_n(30 downto 15);

Simulation

- You will have to write a test bench to simulate your filter
- The testbench will read sample data from a provided Excel input file
- The testbench will write the filtered results to another Excel file
- A template is given in the lab handout

TEXTIO package – For file I/O

- Include “use ieee.std.textio.all”
- Important functions and procedures:
  - **Readline(…)**
  - **Read(…)**
  - **Writeline(…)**
  - **Write(…)**
  - **Endfile(…)**
- Additional types
  - **Text**
  - **line**

Reading From Files

- Declaration of the input file

FILE stimulus : text open read_mode is “stim_in.txt”;

  - **The file_handle is stimulus**
  - **It is of type text**
  - **It is opened in the read mode**

- Obtaining data from the input file

```
For i in 0 to 39 loop                    //if you know there are 40 samples
    Readline(stimulus,  L);              //read a line and put it in variable L
    Read(L, i_int);                      //read a integer from L
    Array(i)  <= i_int;                  //store it in an array
    Wait until clk = ‘1’;
End loop
```

Reading From Files

- Function definitions
  - **Readline – reads a complete line from the file pointed at by the file_handle. The line is put in a buffer of type LINE**
  - **Read – reads from the line into a variable. You have to know what type the data in the input file is so that you can read it into the proper variable type.**

Writing to Files

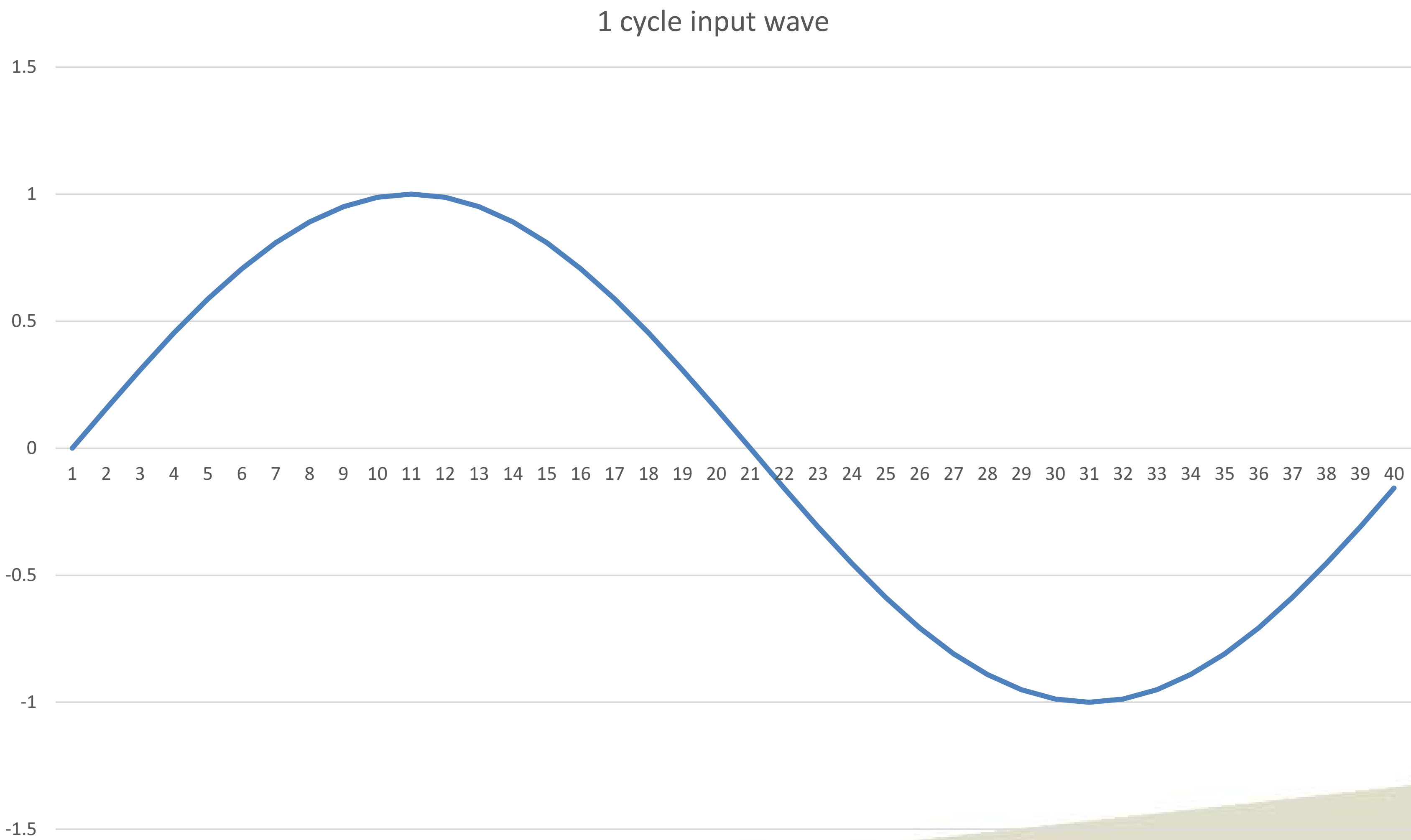
- Declaration of the output file

FILE O_file: text open write_mode is "stim_out.txt";

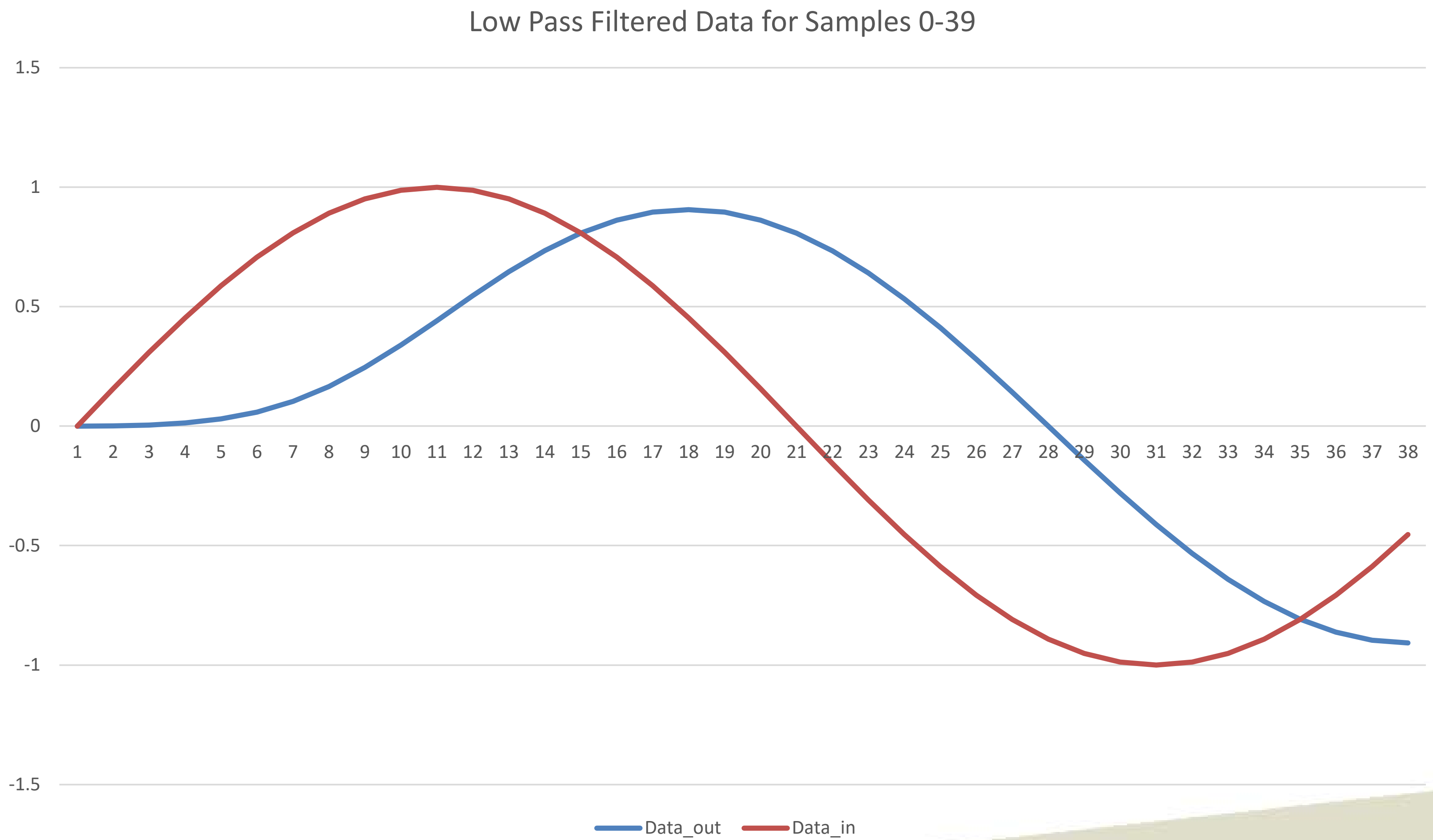
  - **The file handle is O_file**
  - **It is of type text**
  - **It is opened in write mode**

- Write functions
  - **Write(L, o_int) – writes an integer to a buffer L of type LINE**
    - Writes data to a line
  - **Writeline(O_file, L) – writes a complete line to the output file**
    - Writes an entire line to a file

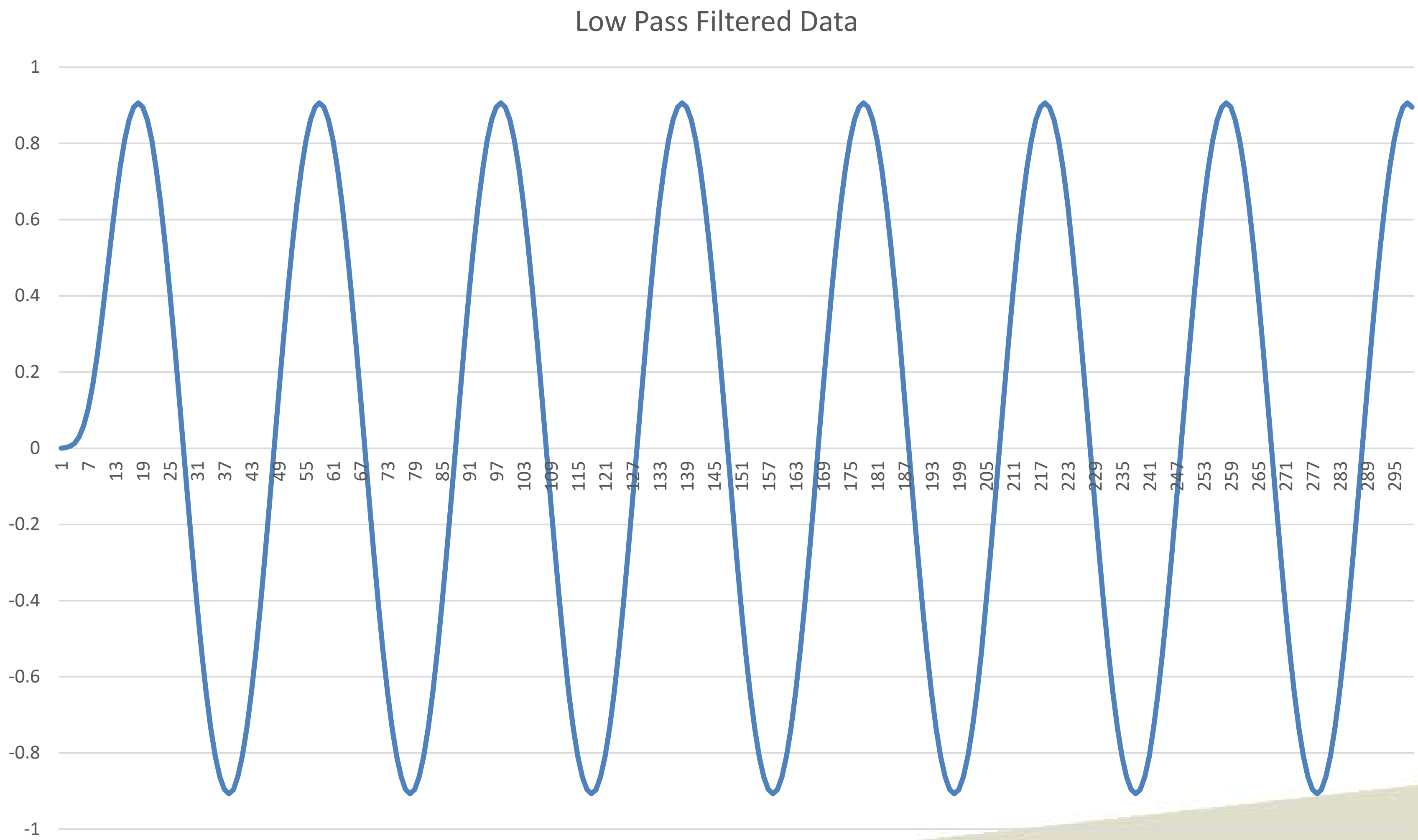
Input Wave from Excel File



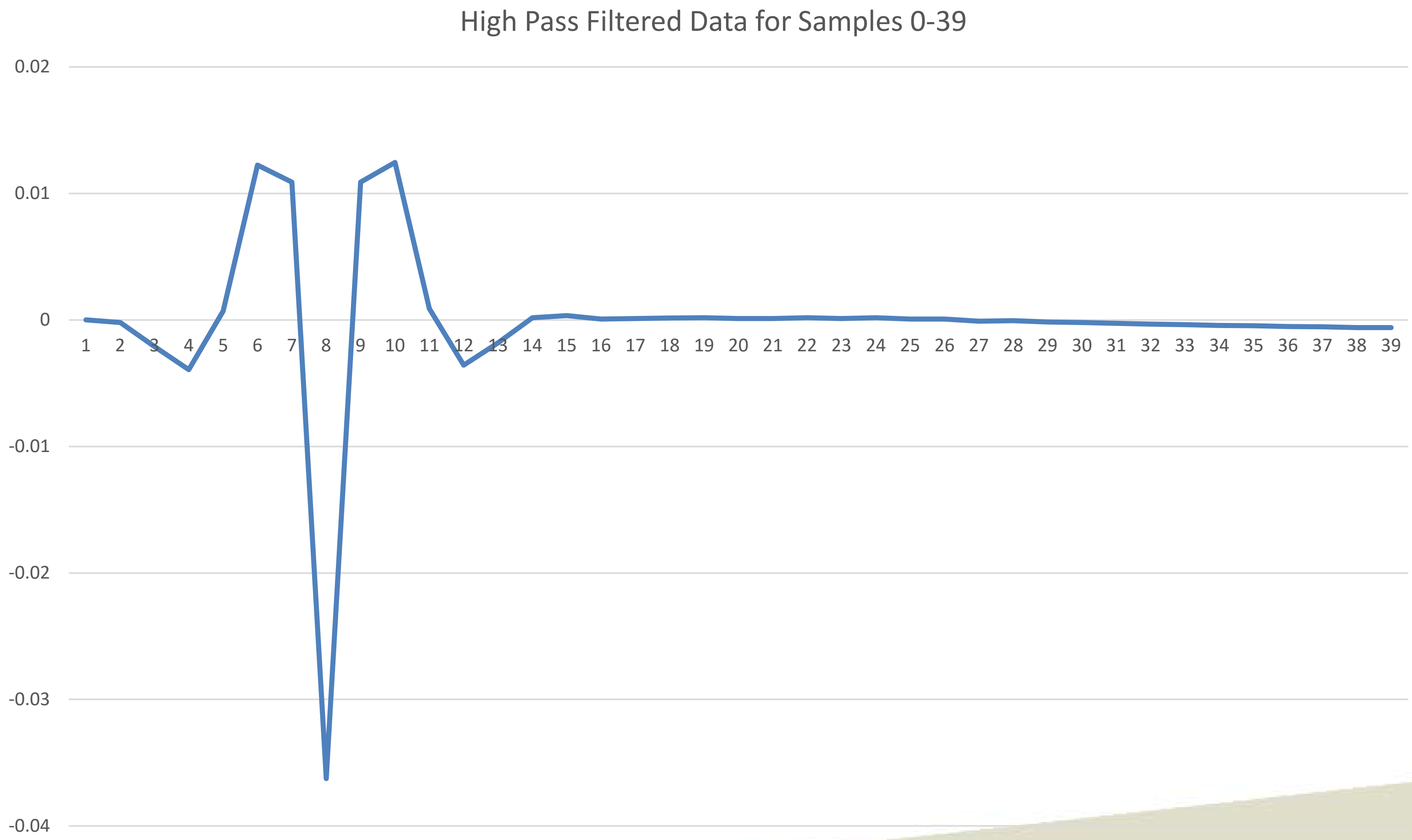
Low Pass Filtered Data



Low Pass Filtered Data Over Time



High Pass Filtered Data



High Pass Filtered Data Over Time