The Abdus Salam
International Centre
for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

2456-3

Joint ICTP-IAEA Workshop on Advances in Digital Spectroscopy

6 - 10 May 2013

Digital Filtering

M. Bogovac
IAEA, Vienna
Austria

IAEA

Digital Pulse Shaping

Implementation of a Trapezoidal filter in a FPGA by using MATLAB & Xilinx design tools

M. Bogovac

5/1/2013

Contents

1. Introduction

Trapezoidal/Triangular filtering is well known and efficient method to obtain spectrum with signal to noise ratio (peak resolution), which is close to optimal for many practical digital spectroscopy systems. Such a system comprises an analog section (fig 1a) and a digital section (fig 1b).

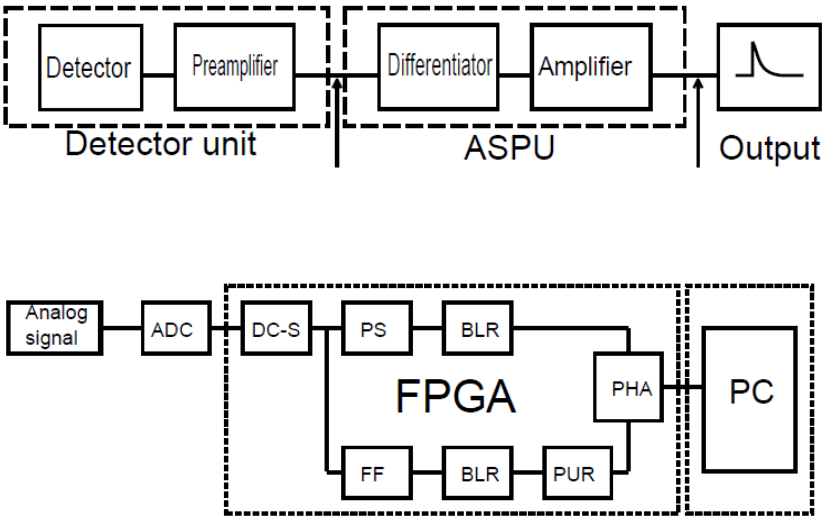


Fig 1a,b

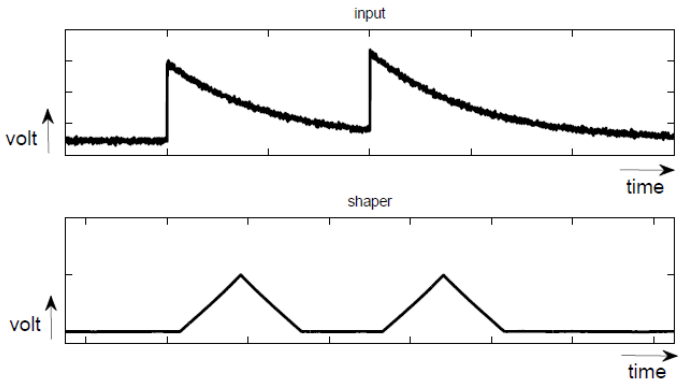


Fig 2

In the analog section, a current pulse produced by detector is integrated on a feedback capacitor of a charge sensitive preamplifier which produces step-like or slow decaying (50 usec) exponential pulse. In order to reduce effect of pile-up (superimposing of two close pulses) and remove low frequency components of noise (noise whitening), a differentiator is included in the section. This shortens decay time to a value that matches corner time constant of the noise generated by detector and preamplifier (usually a few microseconds). In the case of exponential pulse the differentiator may produce distorted exponential shape

(a pole in s-domain), which can be corrected by pole-zero correction circuit. Finally signal is amplified by a linear amplifier and ready for ADC conversion.

In the digital section, signal is converted by a fast ADC (50MHz or so) for real time processing. In the first stage, DC component is removed. This is done by DC suppressor or blocker (DC-S). After that, signal is shaped to trapezoidal/triangular shape (Fig 2), by shaping filter PS. The width of signal must be adjusted to optimal one, which depends on the noise corner time constant of detector-preamplifier system. Before amplitude of the shaped signal is measured, it can be additionally corrected for DC. This is done by a base line restorer (BLR). The distribution of amplitudes (number of pulses with same amplitude) is stored in a memory. In order to reject pile up events, input signal can be shaped to a trapezoid/triangle with very short width (much less than optimal). It makes easier to resolve two successive pulses and reject events when. This is done by fast filter (FF).

2. Design Flow

In this laboratory exercise you will design trapezoidal/triangular shaper and DC-blocker.

During the exercise, the analog section will be substituted by a waveform generator, which is able to generate a trail of exponential signals with short decay time of few microseconds. For digitalization and processing you will use a custom made board. The setup is shown on the fig 2.

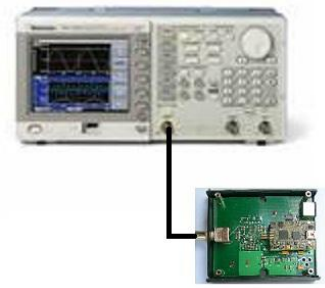


Fig 2

The main task will be to produce FPGA configuration file which contains data processing and communication logic with PC. The configuration file will be produce by using Xilinx ISE development environment.

The FPGA design will consist of 5 modules (Fig 3) that communicate to each other according to a generic wishbone bus protocol. These are:

1. Syscon – generates system clock
2. FX-2 Master module which is responsible for FPGA communication with PC via USB
3. Filter Slave module that contains filter algorithm, oscilloscope (RAM + trigger) and logic for communication with wishbone bus.
4. Intercon module that connects all master and slave modules in the design

The modules 1,2 and 4 are standard and already written in VHDL language using Xilinx ISE editor. You will re-use them only. The Module 5 you will design in graphical programming environment (instead of VHDL) by using computer program called Matlab/Simulink and a set of compatible libraries called Xilinx System generator. These tools use graphical blocks instead of VHDL language and they are able to synthesize VHDL code from graphical blocks.

The VHDL code generated by Matlab/Simulink / System Generator will be later coupled with the rest of modules (1,2,4) by using Xilinx ISE. Finally the ISE will translate, map and route design in a file that can be loaded into FPGA.

The communication with FPGA from PC will be done by a program written in C++.

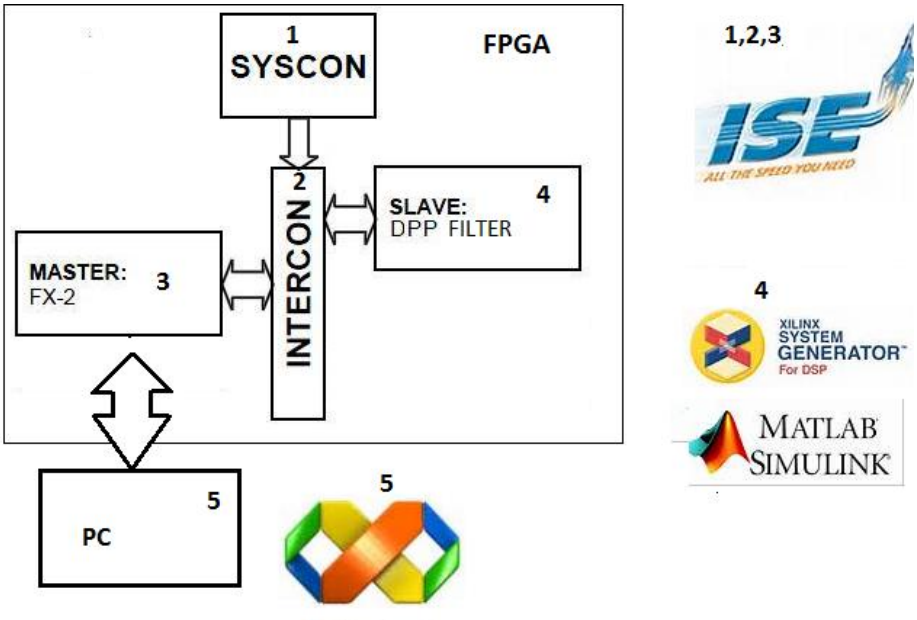
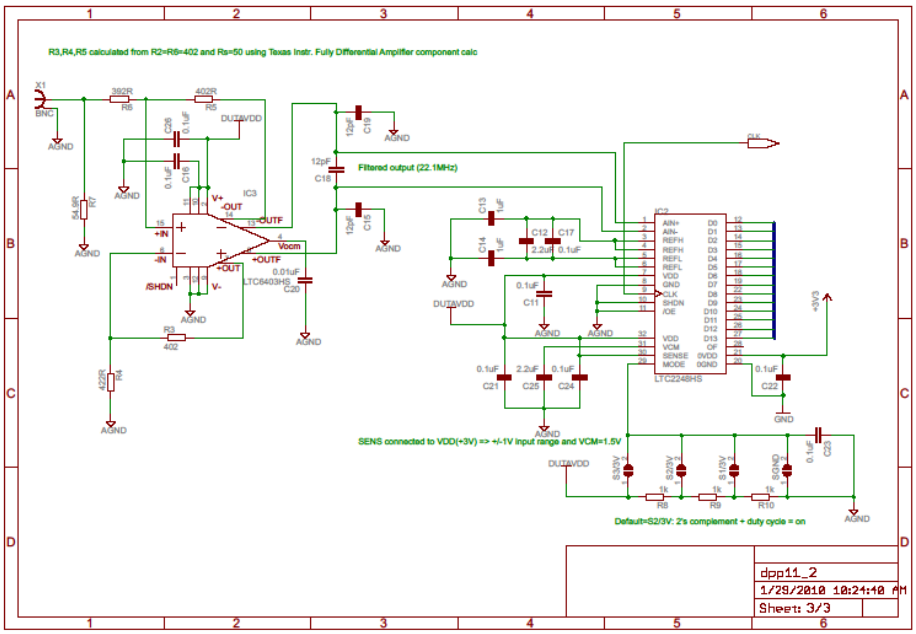

1
SYSCON
FPGA
2
INTERCON
SLAVE:
DPP FILTER
4
MASTER:
FX-2
3
5
PC
5
1,2,3
ISE
ALL THE SPEED YOU NEED
4
XILINX
SYSTEM
GENERATOR
For DSP
MATLAB
SIMULINK

Fig 3.

3. Hardware description

The single ended output signal with 2Vpp nominal amplitude from the Prefilter board is digitized and processed using the FPGA board. The digitization is done by using 14 bit ADC (LTC2248) at sampling rate of 48 MSamples/sec.

For optimal performance the ADC input is driven differentially around common mode voltage of 1.5V and 1Vpp amplitude range (each input swings for ±0.5V). This is accomplished by a low noise amplifier (LTC6403) with 50 Ohm equivalent input resistance operating in a single supply (+3V) configuration and with anti-aliasing filter at its output to restrict bandwidth of the signal to approximately half of the sampling rate.



The ADC is coupled to Spartan-3E FPGA (XC3S500E-4CPG132C). The FPGA is placed on a separate small-sized board (48x22 mm, EFM01 produced by Cesys GmbH, www.cesys.com) which provides two 34-pin connectors.

The board communicates with PC via USB port. Since the same port is used for powering both analog and digital part, no external power supply is needed but the board can operate only while it is connected to the running PC via USB cable.

4. Creating Simulink model

In this section you will create an empty Simulink model.

Start Matlab. Create a new working directories for example “Libraries > Documents > TriesteWorkshop > efm01_filter > sysgen” by using MATLAB’s button on the right side of the “Current Folder”. It opens a dialog which enables you to create new directory. Press OK when exiting from the dialog. It also changes current working directory.

Use ‘File New Model’ to create a new model. It opens a new window with title “untitled”. Use “View” menu command, expand “Model Browser Options” and check “Model Browser”.

Use File command to save new, empty model with name sg_filter.mdl in the working directory.

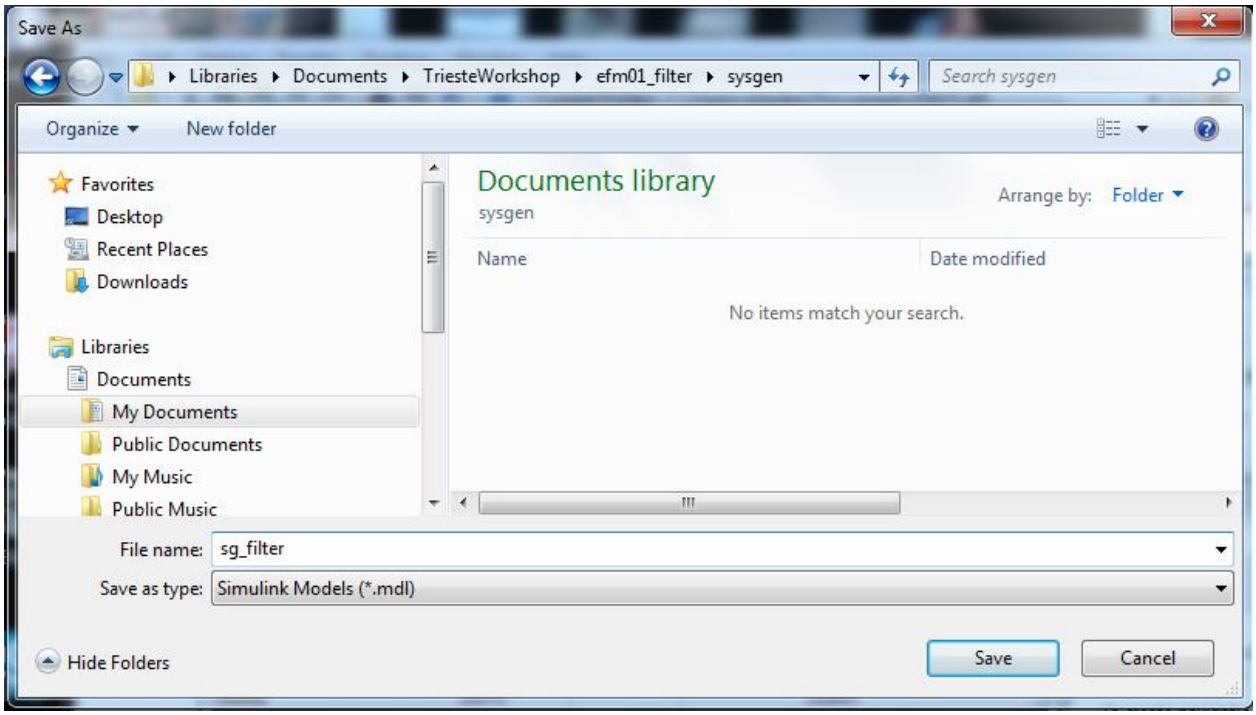


Fig 1.

Your design described in the model file will consist of several logical parts called subsystems. The subsystems consist of Matlab/Simulink blocks and Xilinx blocks. Only Xilinx blocks will be translated to VHDL code or other files needed to produce final FPGA configuration file. Xilinx and Matlab blocks are separated by blocks called “gate way”.

To see all available blocks use “View Library Browser” command in the model window. It opens Simulink Library Browser.

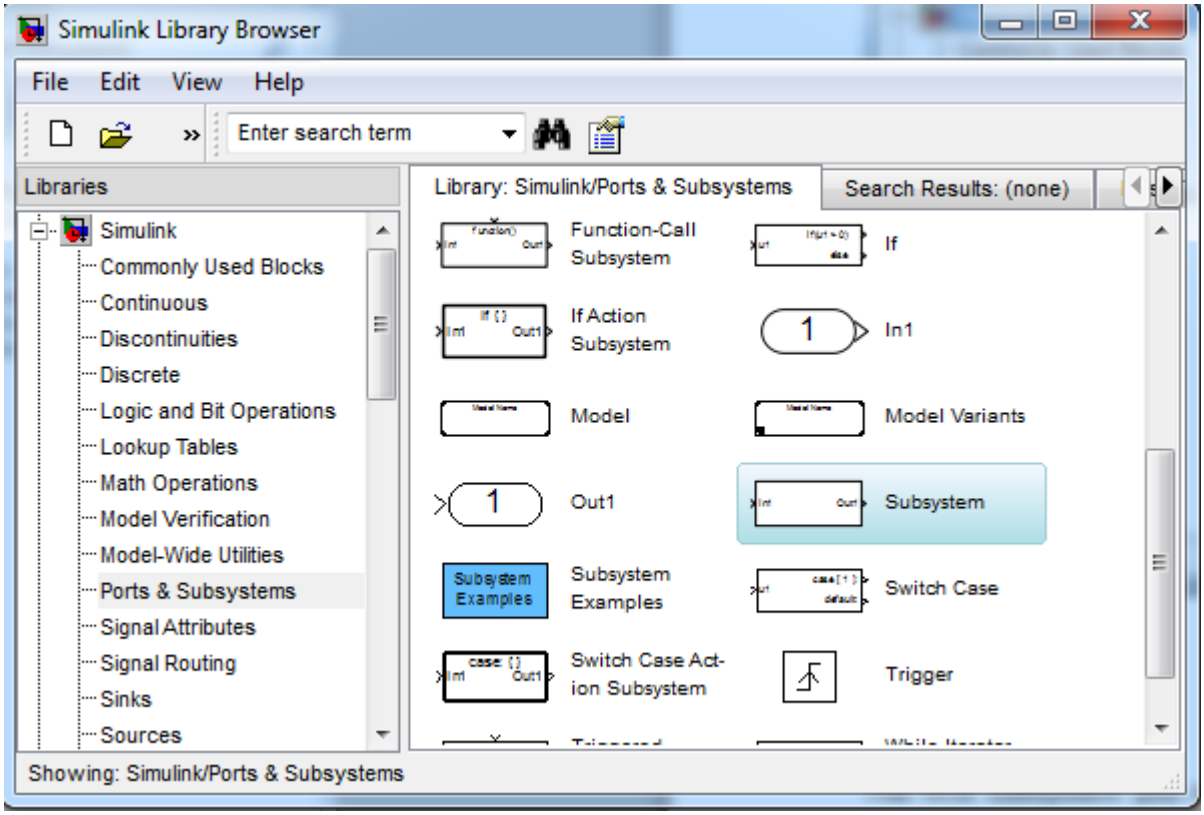

Simulink Library Browser
File
Edit
View
Help
Enter search term
Libraries
Simulink
Commonly Used Blocks
Continuous
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
Library: Simulink/Ports & Subsystems
Search Results: (none)
Function-Call Subsystem
If
If Action Subsystem
In1
Model
Model Variants
Out1
Subsystem
Subsystem Examples
Switch Case
Switch Case Action Subsystem
Trigger
Showing: Simulink/Ports & Subsystems

Fig 2.

5. Pulse generator subsystem

The first subsystem you need to create is called “pulse_generator”. It is used for the purpose of simulation and it generates input signals for the filter. The input signals will be periodic, exponential decay signal, with decay time in the range of few microseconds (5usec in this example). It corresponds to the output of a reset type preamplifier followed by a high pass filter with RC constant of 5usec.

In order to create the subsystem, first select “Subsystem” block in the Simulink Library Browser under the Simulink “Ports & Subsystems” as shown on the Fig 2, then drag it into the model window. In the second step change default subsystem name“Susbsystem1” into “pulse_generator” by double clicking on the name.

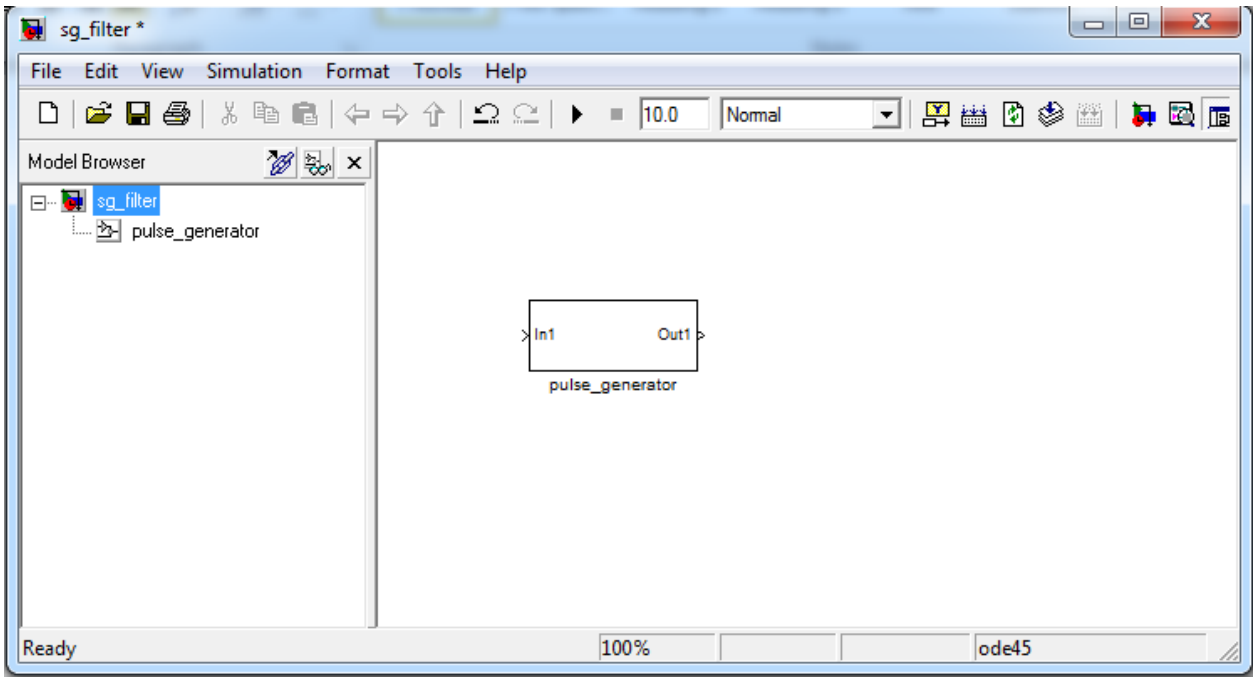


Fig 3.

In order to change “Pulse Generator” subsystem, click on the “pulse_generator” in the “Model Browser” window tree. It opens a new window, which contains only “in” and “out” ports. You must add there appropriate Simulink blocks using Simulink library Browser. These blocks are (see Fig 4.):

Block location	**Block name**
Sources	Pulse Generator
	Band Limited White Noise
Discrete	Memory
Commonly Used Blocks	Gain
	Scope
	Sum

Ports & Subsystems	Triggered Subsystem
Continuous	Zero-Pole

Table 1

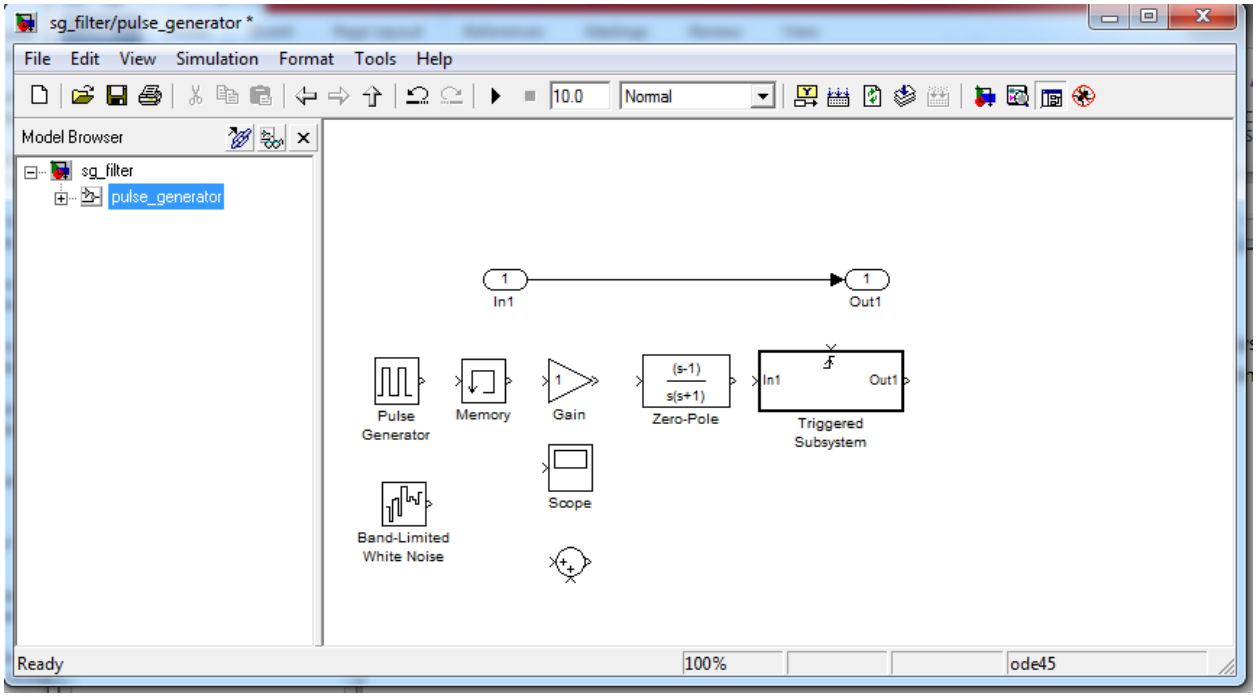


Fig 4.

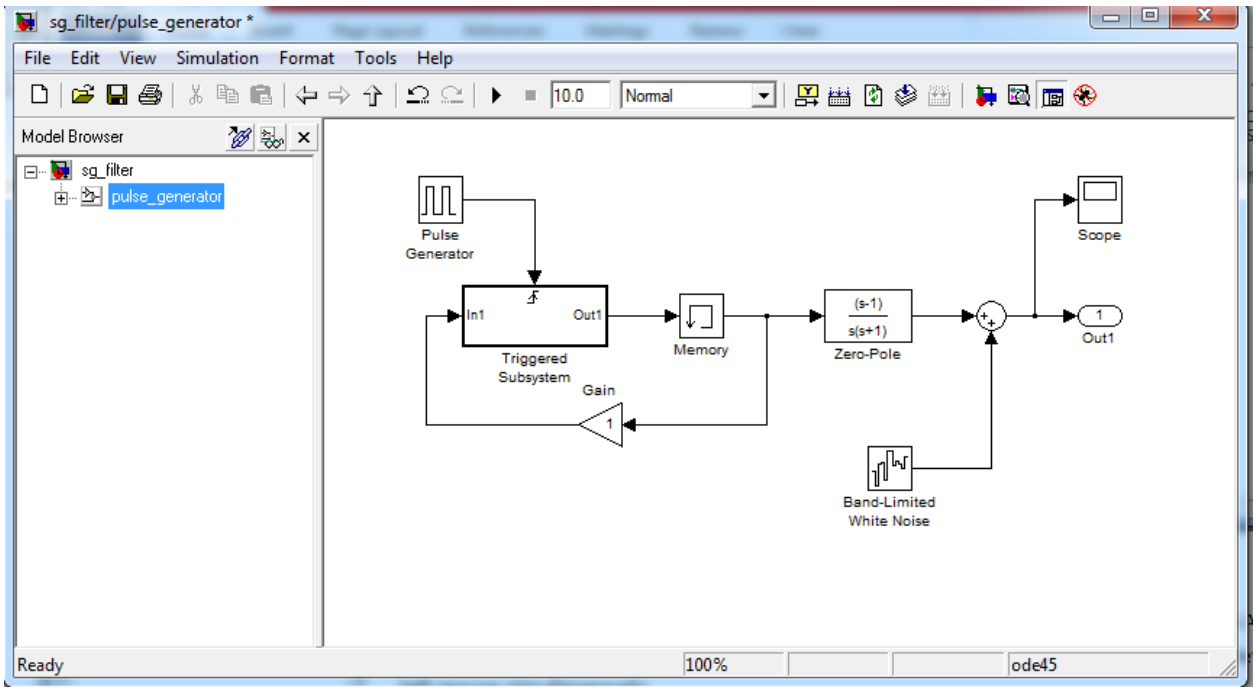


Fig 5.

Now, rearrange and connect blocks from fig 4 as it is shown on the figure 5. To connect block A with block B you can first select clock A, then move mouse to point onto the block B, finally press Ctrl key and left mouse simultaneously.

The Triggered subsystem should be defined. First open the Trigger subsystem – expand pulse generator in the Model Browser tree, and select Trigger Subsystem item. In the next step drag the following Simulink blocks inside Trigger Subsystem:

Sources	Step
Commonly Used Blocks	Sum

Finally connect the blocks as shown on the fig 6.

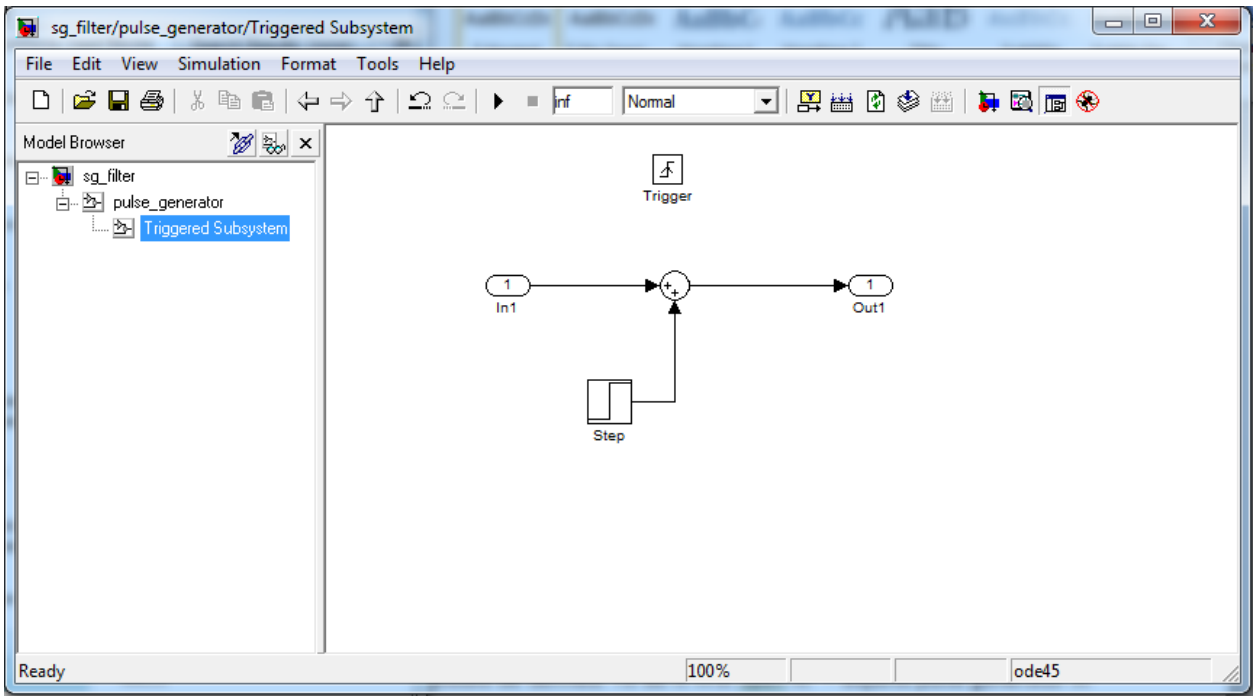


Fig 6.

Each block in the “pulse_generator” subsystem has several parameters that must be defined. Some of them are used with default values. To change required parameter in any of the block you can double click on the block. The following parameters must be changed:

Block name	Parameter	Value
Pulse Generator	Amplitude	1
	Period (secs)	Tpprd*1e-6
Zero-Pole	Zeros	[0]
	Poles	[-1e+6/Taud]

	Gain	[0.8]
Band-Limited white noise	Noise Power	[1e-12]
	Sample time	Tclkn
Triggered Subsystem -Step	Step Time	0
	Sample time	-1
Scope	General - Time Data	400e-6
	Data History - Limit points to	20000

Table 2.

You can observe that few parameters are defined by a variable (Tpprd, Taud, Tclkn). Therefore the variables must be defined somewhere. This can be done in a script file. To do this, create a new script file from MATLAB main window with using "File … New Script" menu command. It opens a new window named "Editor untitled". Enter the following lines below (copy past):

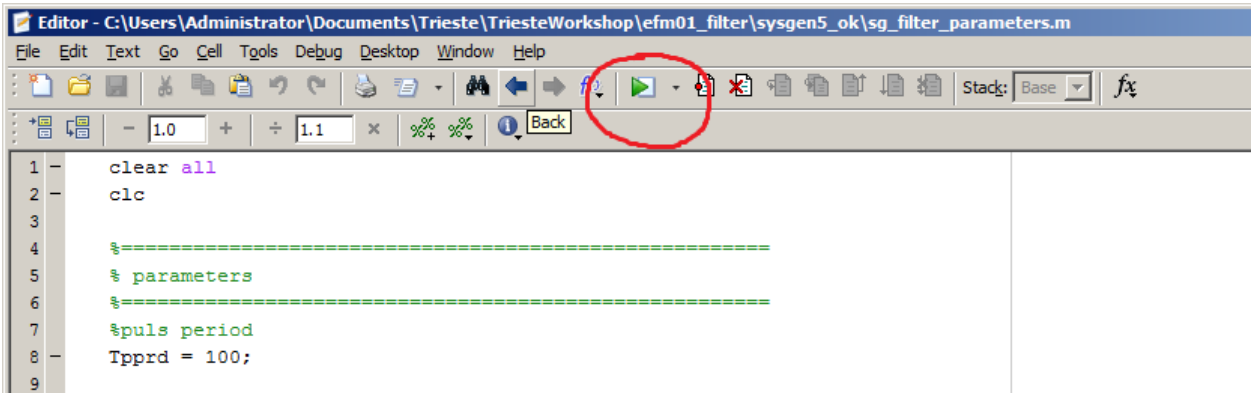
```
clear all
clc

%=====================================================
% parameters
%=====================================================
%pulse period
Tpprd = 100;

%clock period [usec]
Tclk = 1./50;
Tclkn = Tclk*1e-6;

%high pass filter differentiation constant
Taud = 5;
```

Save file with name sg_filter_parameters.m. Before using variables define in a script file, the script must be run. It can be done by menu command "Debug … Run", menu bar icon shown on the figure 7 up or by pressing key F5. Note that, after running the script, all variables appear in the main Matlab window - panel called Workspace.



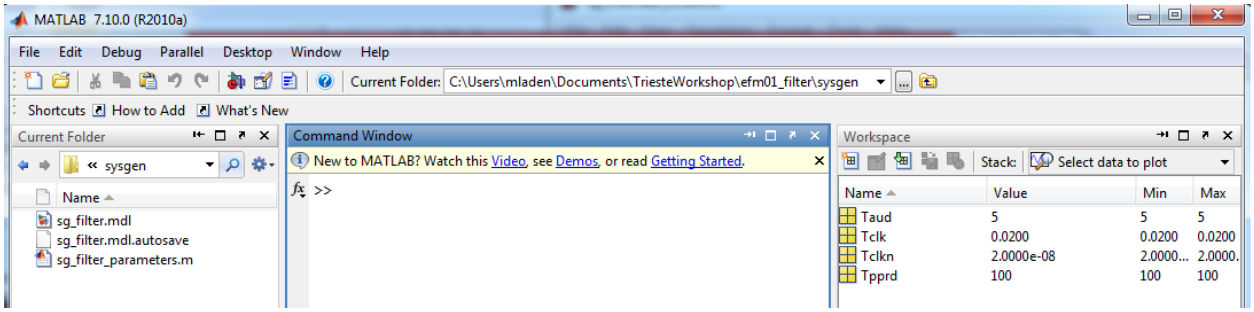


Fig 7.

The function of the Pulse generator can be explained as follows. The block called pulse generator, which simulates arrival time of a charge particle, triggers the Triggered subsystem with period Tpprd microseconds (time normalized by MATLAB). The triggered subsystem, Memory and Gain blocks simulate detector/preamplifier. On each trigger pulse a step signal is generated and added to the previous value of the output from the Memory block. On this way a staircase signal is created and its amplitude changes each Tpprd microseconds. The Zero-pole block is used to simulate high pass filter with RC constant equal Taud. Therefore the overall output of the subsystem is a periodic exponential pulse with decaying time Taud.

In order to start simulation, first define total simulation time. To do it use Simulink menu command “Simulation … Configuration parameters”. It opens a new dialog where you can define Stop time = 400e-6. After closing dialog you can observe that selected value appears in the menu bar. Therefore you can change the simulation stop time in the menu bar as well. Open Scope by double clicking on the Scope and start simulation. After a successful simulation you should observe a signal on the Scope as shown on the Fig 8. It shows an exponential function with decay time of 5usec.

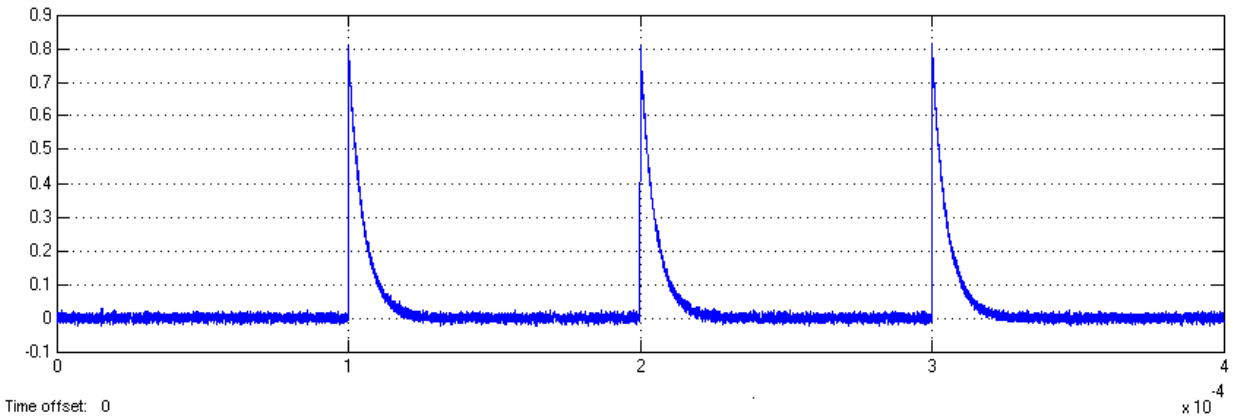


Fig 8.

6. ADC quantizer subsystem

For a purpose of simulation, the output from the pulse_generator subsystem can be additionally quantized by using Simulinks Sample-Hold and Idealized quntizer boxes. This can be done inside a separate subsystem. Therefore create a new subsystem and change his default name into “adc_quantizer” (use Ports & Subsystems to find Subsystem block). Drag the following boxes inside the new subsystem (see table below and Fig 9).

Block location	Block name
Simulink Extras – Additional Discrete	Idealized ADC quantizer
Simulink - Discrete	Zero order hold;
Simulink - Logic and Bit Operations:	Shift Arithmetic
Simulink - Commonly Used Blocks	Data Type conversion

Connect boxes as shown in the Fig 9.

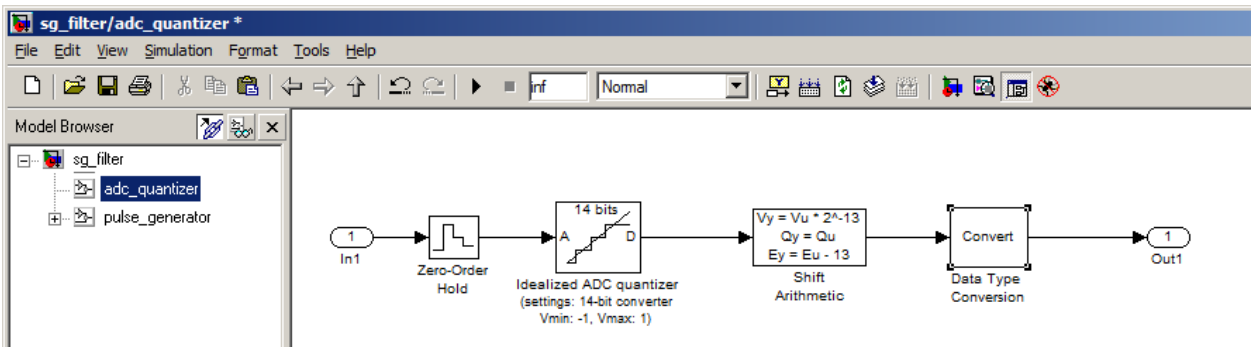


Figure 9a.

Table 3.

Change only blocks parameters that are listed in the table below:

Block name	**Parameter**	**Value**
Zero_Order Hold:	Sample time (-1 for inherited)	Tclkn
Idealized ADC quantizer	Number of converter bits	14
	Min input voltage at low output	-1
	Max input voltage at 2^n output (unreachable)	1
	Output data type	Int16
	Output negative values	Yes (checked)

Shift Arithmetic	Number of bits to shift right	0
	Number of places by which binary point shifts right	-13
Convert	Output data type	fixdt(1,14,13);
	Integer rounding mode	Round
	Saturate on integer overflow	Check box
	Sample time	Tclkn

Table 4

The Zero-Order Hold element samples and holds input signal in a full precision on rising edge of the clock until the next rising edge.

The Idealized ADC converts the full precision values stored in the Hold element into a closest quantizied number x_q. This type of quantization is called rounding (fig 9b).

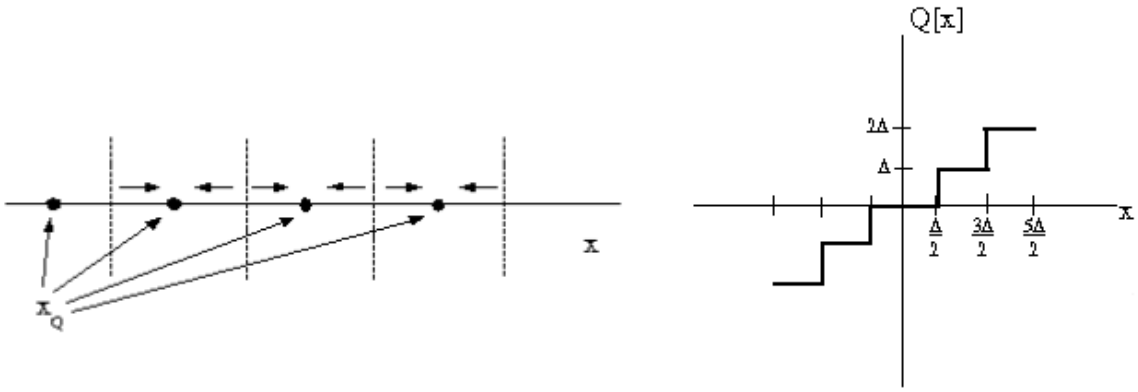


Fig 9b

The digitized values are represented with 14 bit binary number. With using 14 bits it is possible to define $2^{14} = 16384$ different values.

Number (decimal)	Number 14 bit binary
0	00000000000000
1	00000000000001
….	
8191	01111111111111
8192	10000000000000
…..	
16382	11111111111110
16383	11111111111111

Table 5

In order to convert binary number $b_{14}b_{13}\ldots.b_2b_1$ into decimal the following operation should be done

$$2^{13} \times b_{14} + 2^{12} \times b_{13} + \cdots + 2^1 \times b_2 + 2^1 \times b_1$$

Since the input signal is bipolar and range (+,-1) the ADC output will be between -8191 and + 8191 one. By using the next two block the representation of the number is changed into 2's complement fixed point. Two's complement representation allows the use of binary arithmetic operations on signed integers.

The 2's complement representation of an integer N is equal to the usual representation (shown above) if the number is positive. If the number is negative then the following two steps should be done:

1. Write number –N in binary representation (-N is positive)
2. Invert all bits in the above representation
3. Add 1

Number (decimal)	2's complement Number 14 bit binary	
-8191	10000000000001	Inv(8191)+1=Inv(01111111111111)+1
-8190	10000000000010	Inv(8190)+1=Inv(01111111111110)+1
….		
-2	11111111111110	Inv(2)+1=Inv(00000000000010)+1
-1	11111111111111	Inv(1)+1=Inv(00000000000001)+1
0	00000000000000	
1	00000000000001	
2	00000000000010	
….		
8190	01111111111110	
8191	01111111111111	

Table 6

After finishing the adc_quantizer subsystem you must connect it with pulse generator subsystem as shown on the figure 10. To see the effect of the quantizer you may add a scope as shown on the figure 10. Note that both signals are fed into the same input of the scope by using Mux box which can be found in the Simulink … Signal Routing.

The fixed point representation allows using non-integer values. In the above case the point is set to the right of the most significant bit (which is actually used for sign). This means that values are rescaled from interval (-8191,8191) to [-1,1) and represented by 13 places to the right of binary point and one place to the left. The equivalent decimal number is equal to

$$2^{-1} \times b_{13} + 2^{-2} \times b_{12} + \cdots + 2^{-12} \times b_2 + 2^{-13} \times b_1$$

with sign defined by bit b_{14}.

At this place it should be pointed out that quantization of the input signal, and further arithmetic operation introduce noise. First, you may notice a certain difference between exact analog value and its binary representation. This quantization error ε will be uniformly distributed in the interval

$$-\frac{2^{-B}}{2} \leq \varepsilon \leq \frac{2^{-B}}{2}$$

around mean value $m_\varepsilon = 0$, where $B = 13$, is number of binary places to the right of the binary point. The variance of this quantization error will be

$$\sigma_\varepsilon^2 = \frac{1}{\Delta} \int_{-\frac{2^{-B}}{2}}^{\frac{2^{-B}}{2}} (\varepsilon - m_\varepsilon)^2 \, d\varepsilon = \frac{2^{-2B}}{12}$$

Apart from above type of rounding quantization there is also another type called truncation quantization where value is represented by nearest lowest quantized level as shown on the figure 9c.

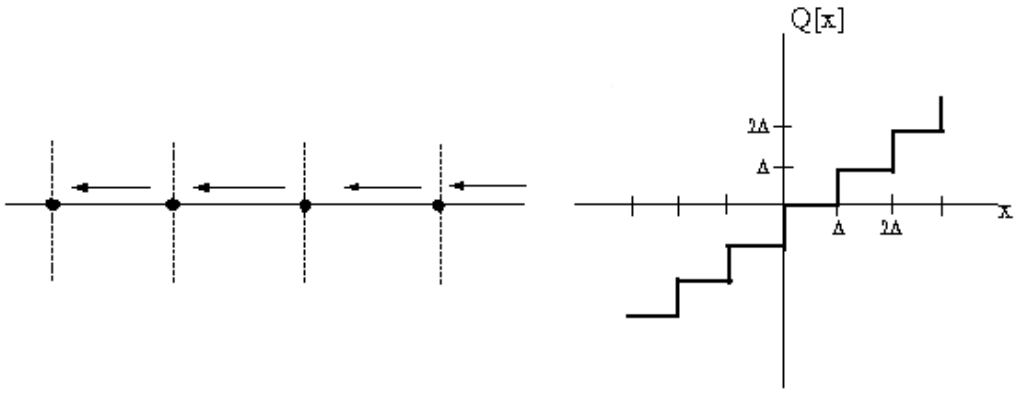


Fig 9c.

It can be shown that in this case

$$m_\varepsilon = \frac{2^{-B}}{2}$$

$$\sigma_\varepsilon^2 = \frac{2^{-2B}}{12}$$

To see the effect of quantization, add another convert box and scope as shown onto the figure 10.

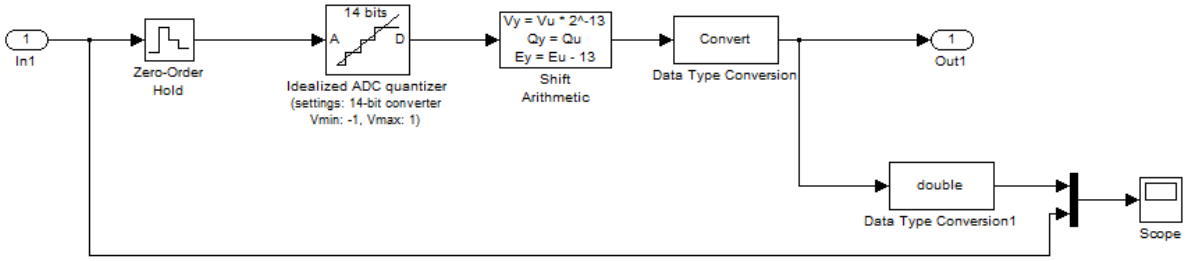

1
In1
Zero-Order
Hold
14 bits
A
D
Idealized ADC quantizer
(settings: 14-bit converter
Vmin: -1, Vmax: 1)
Vy = Vu * 2^-13
Qy = Qu
Ey = Eu - 13
Shift
Arithmetic
Convert
Data Type Conversion
1
Out1
double
Data Type Conversion1
Scope

Figure 10.

7. ADC interface subsystem

The filter will be implemented by using Xilinx's blocks, instead of Simulink's. The blocks and associated logic will be transferred to the hardware gates and data sampled by a real ADC fed into the FPGA through its I/O pins. Therefore the data fed into the filter must pass through Xilinx "Gateway In" block. The block can be found in Xilinx Blockset: Basic Elements. In order to keep design more structured you may put this block in a separate subsystem called adc_if as shown on the figure 11. Also change the block name into adc_data.

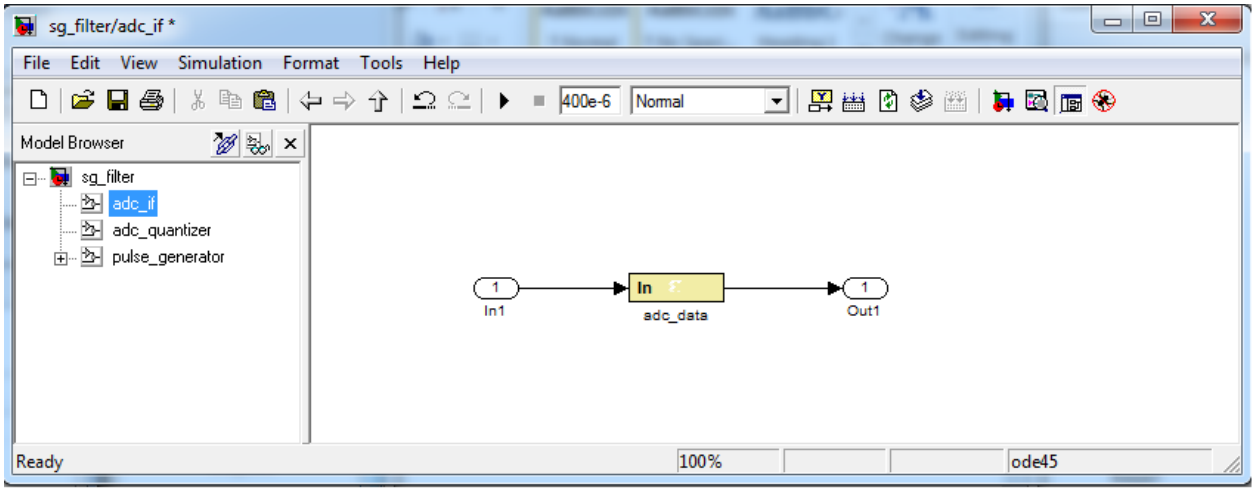


Fig 11.

The block's parameter that should be changed are:

Block name	Parameter name	Parameter value
Xilinx Gateway In	Number of bits	14
	Binary point	13
	Quantization	Truncate
	Sample period	Tclkn

Table 7

Note that the System Generator will translate adc_data In block into VHDL input port with the same name and signal size 14. During simulation the Gatway is fed by the adc_quantizer box. Therefore you must connect adc_if and adc_quantizer subsystems.

8. Trapezoidal filter subsystem

Now you can proceed with the implementation of the filter. For this purpose create a new subsystem and call it “filter”.

The filter’s transfer function can be factorized as follows

$$H(z) = (1 - e^{-\frac{T_{clk}}{\tau_d} z^{-1}}) \left(\frac{1 - z^{-n_a}}{1 - z^{-1}}\right) \left(\frac{1 - z^{-n_b}}{1 - z^{-1}}\right) \left(\frac{z^{-1}}{n_a}\right)$$

Apart from the clock period T_{clk} and high-pass filter constant τ_d, the formula has two other parameters that define flat-top $\tau_{ft} = n_a + n_b$ and peaking times $\tau_{pk} = n_a$ in clock periods unit. It is convenient to put these parameters in the sg_filter_parameters.m script file; Therefore write (copy & paste) the following lines into the file

```
%filter parameters
Taupk = 3;
Taupk_top = 2;
```

The above form suggests splitting the filter into four cascaded parts (stages). Therefore create four new subsystems inside the filter subsystem as shown on the figure 12.

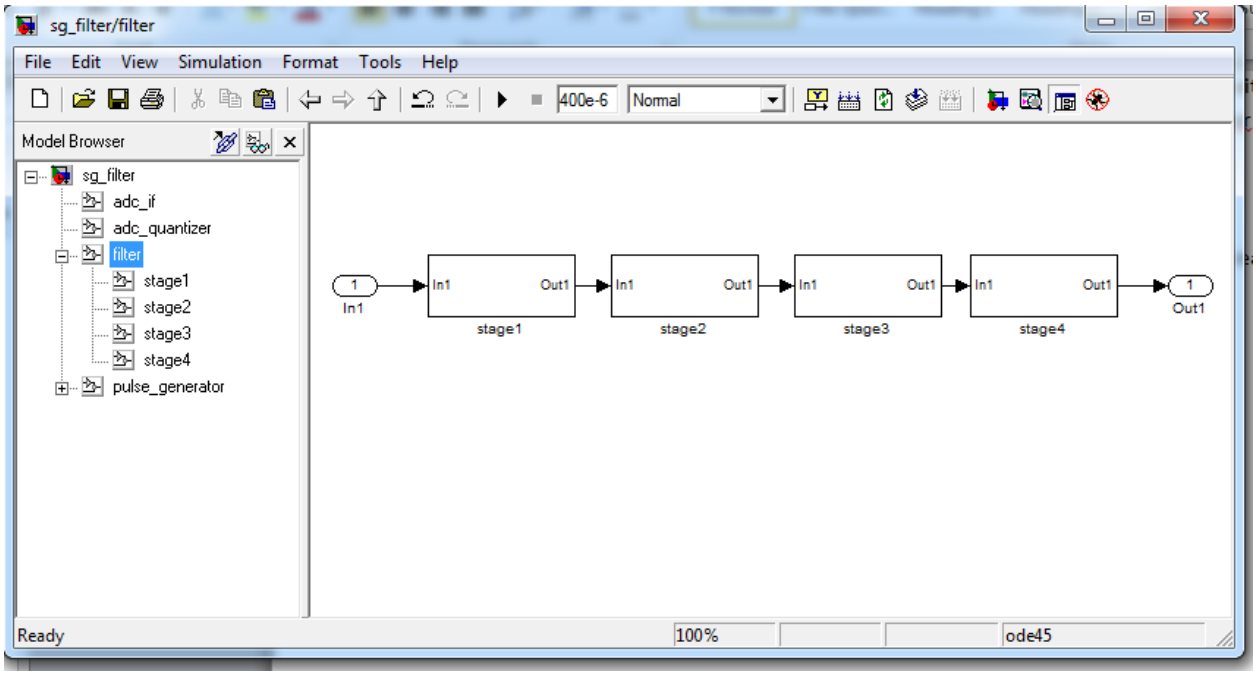


Fig 12.

The stages can be ordered in arbitrary way. It is convenient to do it in the same order as it is written in the above formula.

In the next section you will design and analyze all stages in detail.

1.1 Stage I

The transfer function of the first stage is

$$H(z) = (1 - b_{10}z^{-1})$$

Where $b_{10} = e^{-\frac{T_{clk}}{\tau_d}}$

The above formula, written in z-transform space, corresponds to the following recurrence relation in n space between input and outputs series of samples x_n and y_n respectively.

$$y_n = x_n - b_{10}x_{n-1}$$

1.1.1 Design

The constant b_{10} must be defined in the sg_filter_parameters.m script file. Therefore add the following line at the end of the file:

```
b10 = exp(-Tclk/Taud);
```

Run the script as explained under the "Pulse_generator subsystem". This will define corresponding variable in MATLAB's global workspace.

In order to implement stage 1 recursive relation you need the following blocks:

Block location	Block name
Xilinx Blocksets – Math	Add/Subtract
	Mult
Xilinx Blocksets – Basic Elements	Delay
	Constant

Table 8

Drag the blocks inside the stage1 subsystem and connect them as shown on the figure 13.

You need to define parameters for the each block. The optimal values for each parameter will be discussed later. Now change only parameters that are listed in the table below:

Block name	Parameter name	Parameter value
Mult	Precision	User defined
	Output type	Signed
	Number of bits	24
	Binary point	23
	Quantization	Round
	Overflow	Saturate

	Latency	0
Sub	Basic: Operation	Subtraction
	Output type: Precision	full
Constant	Basic: Type;	Unsigned
	Constant value	b10
	Number of bits	23
	Binary point	23
	Sample constant	yes
	Sample time	Tclkn

Table 9

All other parameters that are not listed above should not be changed.

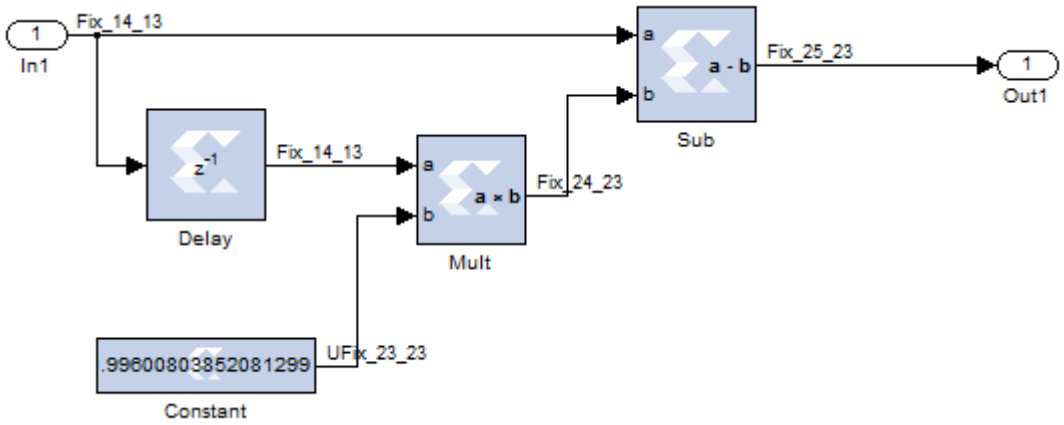


Fig 13.

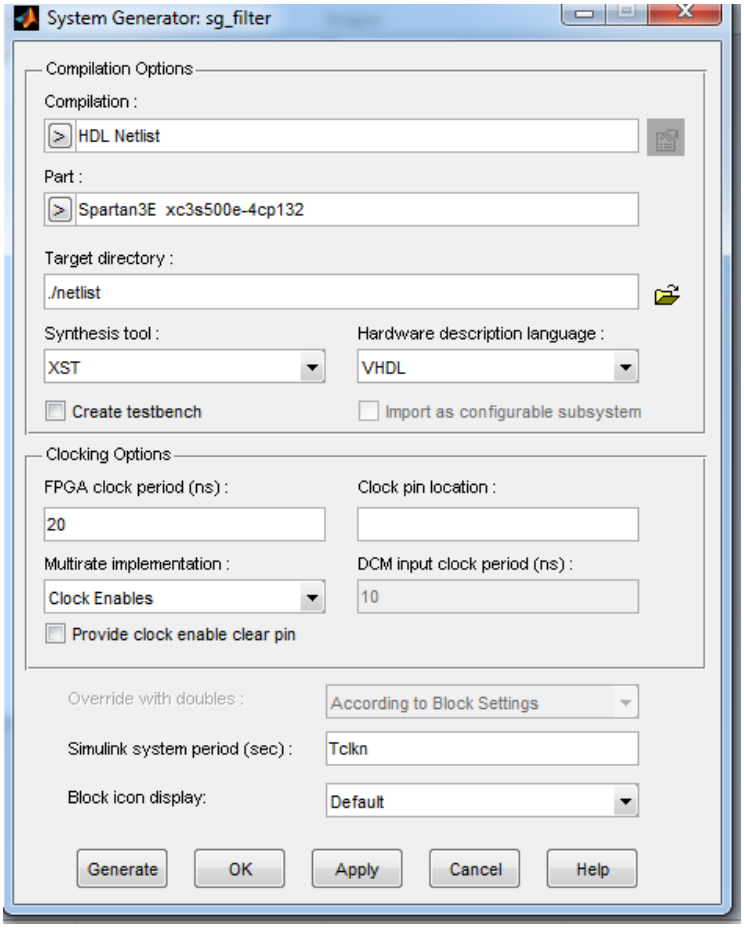
1.1.2 Simulation

Before System generator design can be simulated or translated into the hardware the design must include a System generator block. When creating a new design, it is a good idea to add a System Generator Block immediately. You can find it in the Xilinx Blockset's Basic Elements and Tools Libraries or in the Index library.

Drag the System Generator Block into the top level module and double click the block. Enter the following values for the block parameters:

Parameter name	Parameter value
Compilation	HDL Netlist
Part	Spartan3E xc3s500e-4cp132
Target directory	./netlist
Synthesis tool	XST

Hardware description language	VHDL
Create testbench	Not checked
Import as configurable system	Not checked
FPGA clock period (ns)	20
Clock pin location	Do not enter
Multirate implementation	Clock enables
Provide clock enable clear pin	unchecked
Simulink system period	Tclkn
Block icon display	Default



Press OK button to exit from the dialog.

Run simulation and observe shape of the output signal with using Scope block (fig 13). The shape is equal to a current signal produced by detector.

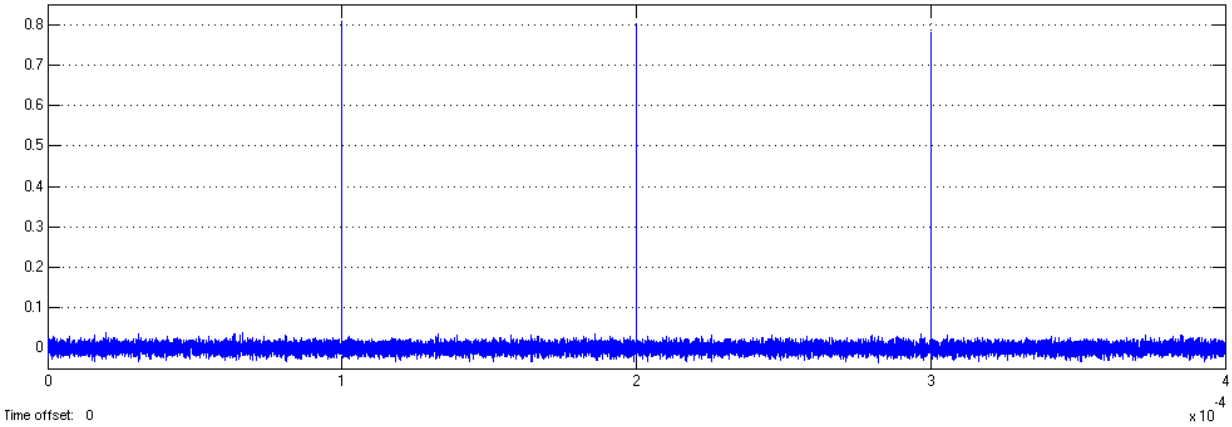


Fig 13

This shape can be calculated by entering formula for the input exponential decaying function (see Fig 8)

$$x_n = \begin{cases} e^{-\frac{T_{clk}}{\tau_d}n} & n \geq 0 \\ 0 & n < 0 \end{cases}$$

into filter's recursive relation

$$y_n = x_n - b_{10}x_{n-1}$$

This gives

$$y_n = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

1.2 Stage II

The stage 2 is defined by the following transfer function

$$H_2(z) = \frac{1 - z^{-n_a}}{1 - z^{-1}} = \frac{1}{1 - z^{-1}}(1 - z^{-n_a})$$

The first factor

$$H_1(z) = \frac{1}{1 - z^{-1}}$$

represents an integrator. The corresponding recursive relation is

$$y_n = y_{n-1} + x_n$$

The second factor

$$H_2(z) = 1 - z^{-n_a}$$

is called comb filter. Its recursive relation is

$$y_n = x_n - x_{n-n_a}$$

1.2.1 Design

The Xilinx provides a block that exactly performs first recursive relation. It is called Accumulator. Find it under Xilinx Blockset /Math and drag into the stage 2 subsystem.

In order to implement the second recursive relation, you need a subtractor and a delay line. Find subtractor under the Xilinx Blockset /Math and drag into the stage 2 subsystem.

The delay line can be constructed with using “Delay” and “Addressable shift register” blocks in the Xilinx Basic Elements. The first one can’t have variable delay length, therefore it is not convenient. The second is parametrazible but for large delays (several hundred) it takes a lot of resources (registers), also the length is limited to 1024. For large delays, more resource efficient solution is to use a block RAM and pass data through it, until desired amount of delay is achieved. The way how you can do it is shown in a model called ram_delay_line.mdl (fig 14). You can find it in the working directory and open it by Simulink Library Browser (located in).

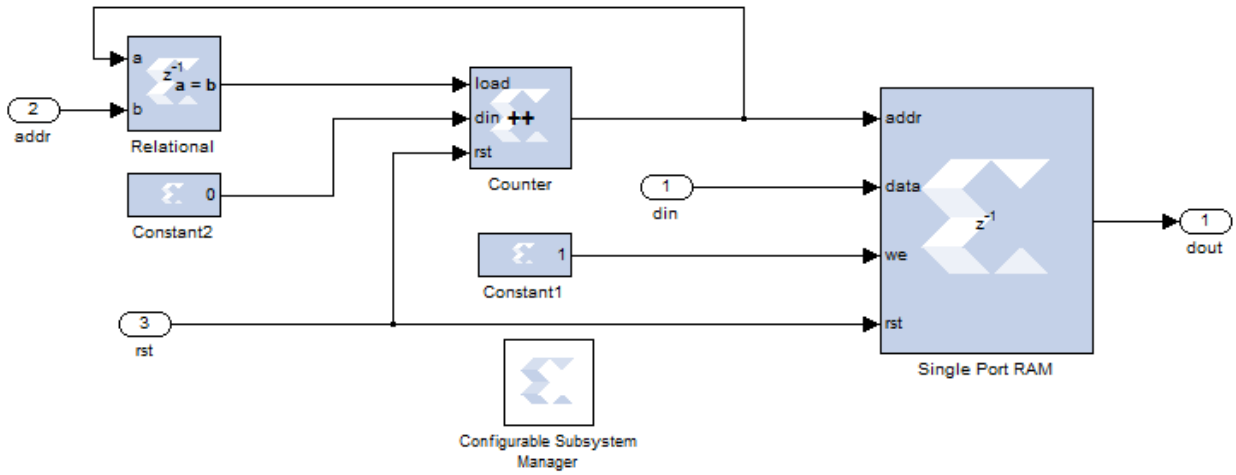


Fig 14.

The design exploates a Single port ram (implemented as a Block RAM, in contrast to Distributed memory), with write mode set to “Read before write”. The RAM address is defined by a counter The counter counts from 0 up to a number which defines number of clock delays and then starts from 0 again. On this way periodic addresses, with period equal to the required value of delay, are generated for writing to the RAM. Any write operation to

the RAM at a particular address also triggers read operation from the same address. Since the read is done before right, the RAM will always output a number that was stored in a previous counter cycle, that is with a required delay.

The ram_delay_line code was stored as a library and you can reuse it without writing your own code. To do it, activate top-most subsystem in the library (ram_delay_line) and drag delay_n subsystem into stage2 subsystem of your design (sg_filter).

In order to complete stage2 you need the following Xilinx blocks:

Block location	Block name
Xilinx Blocksets - Math	Accumulator
	Constant
	Add

Table 10

Connect blocks as shown on the figure 15.

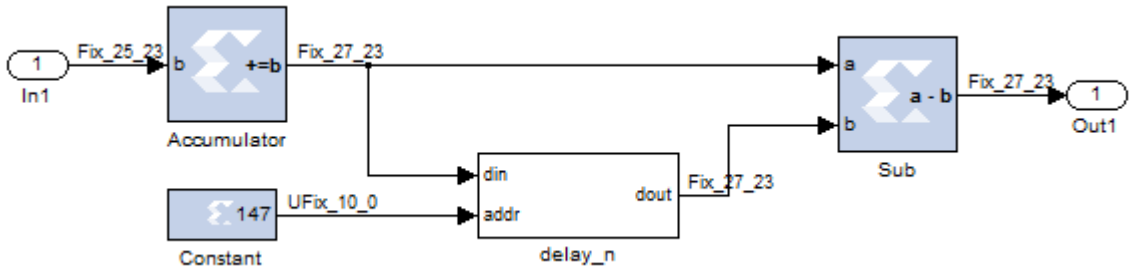


Fig 15.

It should be pointed out that Accumulator in the stage 2 has an internal unity feedback coefficient and it will result in a register overflow - summing element in the accumulator has a finite length and it will overflow after sufficient number of pulses. However, it will not be a problem if two conditions are met:

- the range of the number system is greater than or equal to the maximum value expected at the output
- the accumulator is implemented with two's complement (nonsaturating) arithmetic.

The above conclusions are well know from study of CIC filter (Cascaded Integrator and Comb) which is $H_2(z)$ basically.

The stage 2 subsystem is using only one parameter - delay length n_a (its physical meaning is peaking time) which is expressed in the number of clocks T_{clk}. Since the latency of the above implemented delay line is 3, in order to get proper delay you must use delay of $n_a - 3$ instead of n_a. Therefore add two new lines into the script file sg_filter_parameters.m

```
na = (Taupk/Tclk);
nad = na-3;
```

In order to place the values in the workspace, run the script as described in the "Pulse_generator subsystem".

You need to define block parameters. Change only parameters listed in the table below (explained later)

Block name	Parameter name	Parameter value
Accumulator:	Number of bits	27
	Overflow	Wrap
	Provide Async reset	Uncheck
Sub	Basic: Operation	Subtraction
	Output type: Precision	User defined
	Output type: Output type	Signed
	Output type: Number of bits	27
	Output type: Binary Point	23
	Output Type: Quantization	Round
	Output Type: Overflow	Wrap
Constant	Basic: Type	Unsigned
	Constant value	nad
	Number off bits	10
	Binary point	0
	Sample period	Tclkn

Table 11

1.2.2 Simulation

Add a Scope and connect it to the output of the stage 2. Run simulation and observe shape of the output signal (Fig 16)

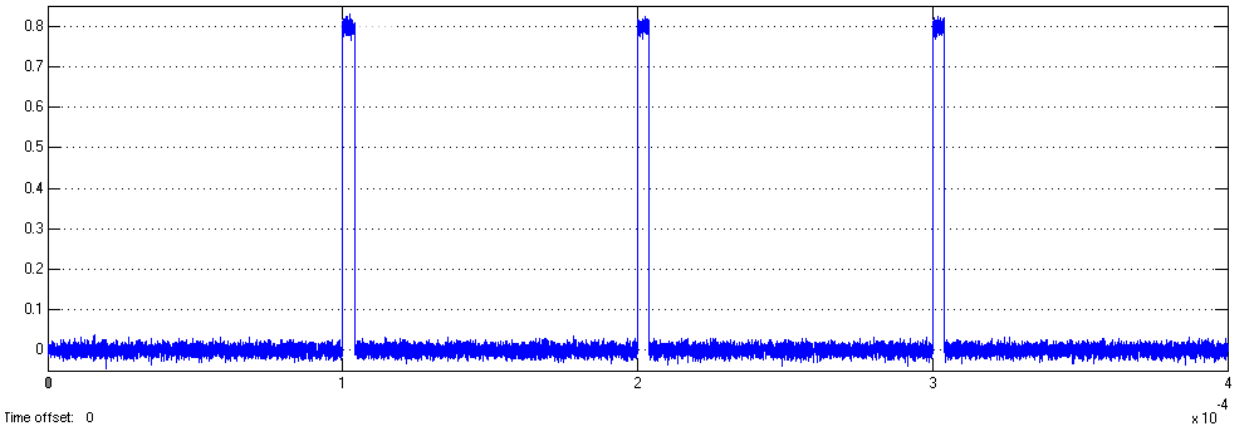


Fig 16.

In order to undersyand the shape, recall that transfer function of the stage 2 is

$$H_2(z) = \frac{1 - z^{-n_a}}{1 - z^{-1}}$$

which can be rewritten in the so called FIR (Finite Impulse Response) form as

$$H_2(z) = 1 + z^{-1} + \cdots + z^{-(n_a - 1)}$$

Its corresponding recursive relation is

$$y_n = x_n + x_{n-1} + \cdots + x_{n-(n_a - 1)}$$

Since the input x_n into the stage 2 (output from stage 1) is delta like signal, the output will be non-zero only during n_a successive clocks. During this period the total sum is always constant. Therefore the output has shape of rectangular signal having width equal to n_a and the amplitude equal to the amplitude of input delta function (see Fig 16).

1.3 About precision and gain of stages I and II

There are two main sources of errors that affect precision of the output of the stages, which exist even in the case that input signal into ADC has no noise. These are:

- rounding/truncation of the filter coefficient b_{10}
- ADC quantization
- rounding/truncation performed during arithmetic operations (multiplication).

1.3.1 Errors due to rounding/truncation of the filter coefficient b_{10}

If the coefficient b_{10} has an error $\varepsilon(b_{10})$ due to quantization and errors in the quantization of x_n are omitted, then output from the stage 1 is given by the formula

$$y_n(\alpha) = \begin{cases} 1 & n = 0 \\ -\varepsilon(b_{10}) e^{-\frac{T_{clk}}{\tau_d}(n-1)} & n > 0 \end{cases}$$

In the case of no error the relation would be

$$y_n = \begin{cases} 1 & n = 0 \\ 0 & n > 0 \end{cases}$$

Therefore, the error in y_n is $(y_n) = y_n(\alpha) - y_n$, which can be written as

$$\varepsilon(y_n) = \begin{cases} 0 & n \leq 0 \\ -\varepsilon(b_{10}) e^{-\frac{T_{clk}}{\tau_d}(n-1)} & n > 0 \end{cases}$$

The error $\varepsilon(y_n)$ in calculation of y_n causes maximal error $\varepsilon_{max}(r_n)$ in calculation of rectangular output r_n from stage 2 given by formula

$$\varepsilon_{max}(r_n) = \sum_{k=0}^{n_a-1} \varepsilon(y_n) \approx n_a e^{\frac{T_{clk}}{\tau_d}} \varepsilon(b_{10})$$

Taking into account that $n_a \leq 2^{10}$ (peaking times up to 20 µsec and clock period 20ns) it follows

$$\varepsilon(b_{10}) < \frac{1}{2^{13}} \frac{1}{2^{10}} = \frac{1}{2^{23}}$$

This means that b_{10} must be represented with at least 23 bits, because in this case rounding error is

$$\varepsilon(b_{10}) = \frac{1}{2} \frac{1}{2^{23}}$$

Since $b_{10} < 1$, its format must be 23.23. In order to observe effect of the precision onto the signal shape you can make simulation.

1.3.2 Simulation of errors due to rounding/truncation of the filter coefficient b_{10}

First, in the stage 2, change precision of the b_{10} to 18.18 and precision of multiplicator to 19.18. Then, in the pulse_generator subsystem, disconnect White noise block from Sum block. Check in the script file that $\tau_d = 5\mu sec$, $\tau_p = 3\mu sec$, $\tau_{ftop} = 2\mu sec$. Finally, run simulation and observe shape of the rectangular output of the stage 2. Notice that edges of the rectangular are deformed and a "noise" is superimposed on it: to see it, zoom (click left mouse and drag coursor) a part to the right of the rectangular edge (fig 16b) and top of the rectangle (fig 16c). ,c). You can see that error is larger than $2^{-13} = 1.2\ 10^{-4}$ (lest significant bit of ADC)

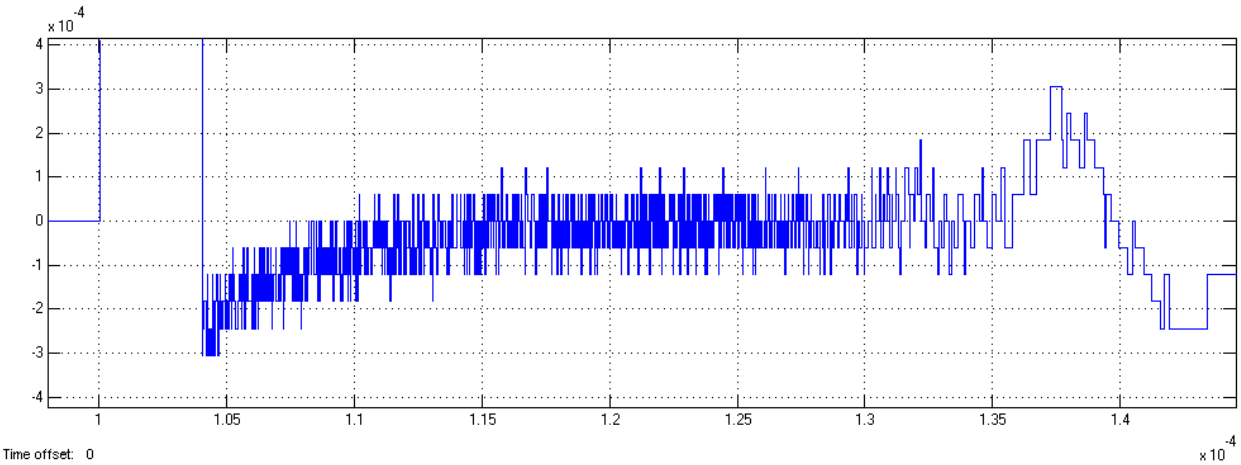


Fig 16b

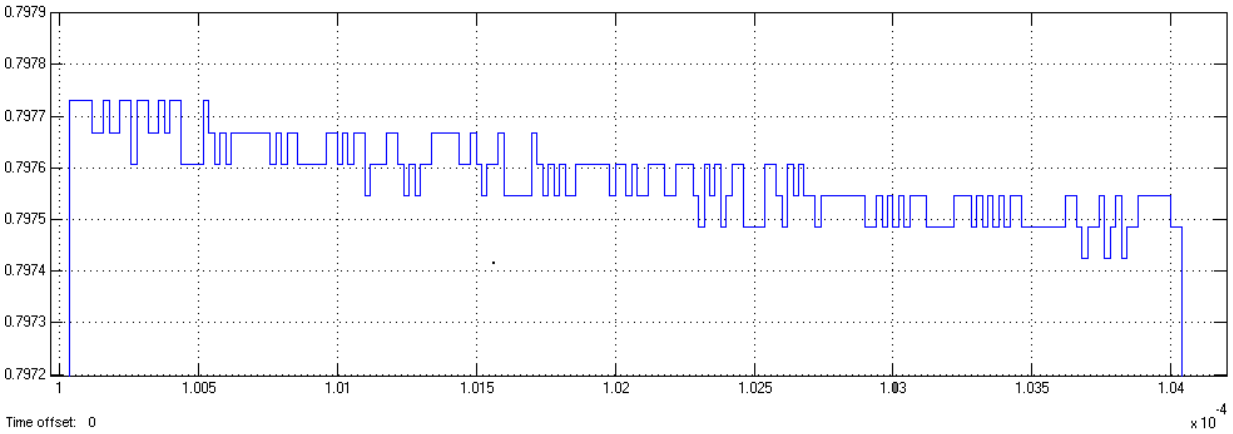
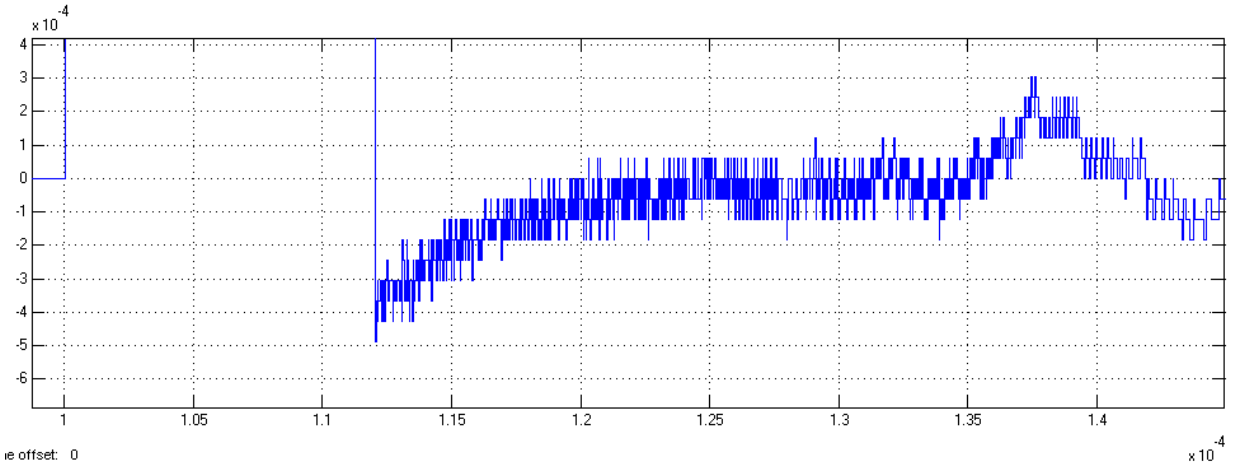


Fig 16c

Now increase peaking time in the script file for four times, from 3 to 12, run script file and run simulation. Notice that error has increased around two times.



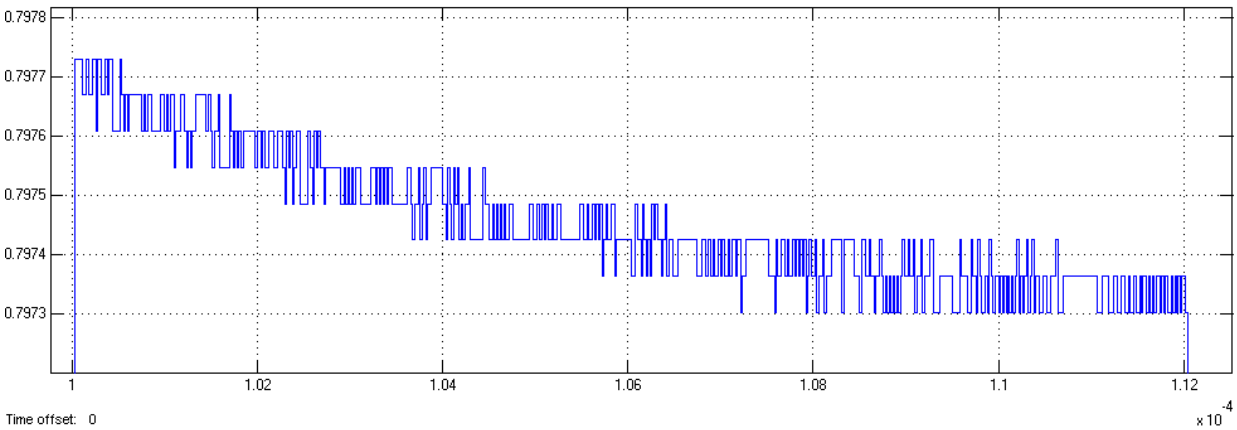


Fig 16d,e

1.3.3 Errors due to ADC quantization

In order to estimate errors due to ADC quantization, recall from the section "ADC quantizer subsystem", that quantization of input signal x_n which is sampled with 13 bit precision (format is 14.13, signed introduces errors that are modeled as a noise with variance σ_ε^2 equal to

$$\sigma_\varepsilon^2 = \frac{2^{-2B}}{12} = \frac{2^{-2\times 13}}{12}$$

This noise will propagate through the stages 1 and 2. In order to estimate total variance of the noise at output of the stage 2, it is useful to write total transfer function of the stage 1 and 2 $H_{12}(z) = H_1(z)H_2(z)$ as a sum of product. First note that10

$$H_{12}(z) = (1 - b_{10}z^{-1})(1 + z^{-1} + \cdots + z^{-(n_a-1)})$$

After performing multiplication, you can write

$$H_{12}(z) = \sum_{k=0}^{n_a} h_k z^{-k}$$

$$h_k = \begin{cases} 1 & k = 0 \\ 1 - b_{10} & 0 < k < n_a \\ -b_{10} & k = n_a \end{cases}$$

The above form of transfer function corresponds to the following sum-of-product relation

$$y_n = \sum_{k=0}^{n_a-1} h_k x_{n-k} = h_0 x_n + h_1 x_{n-1} + \cdots + h_{n_a} x_{n-(n_a-1)}$$

Assuming that errors are independent, the variance at output 2 due to ADC quantization will be

$$\sigma_{21}^2 = \sigma_\varepsilon^2 \sum_{k=0}^{n_a} h_k^2 \approx \left[n_a \left(\frac{T_{clk}}{\tau_d}\right)^2 + 2\right]\sigma_\varepsilon^2 = \left[\frac{\tau_p}{\tau_d}\frac{T_{clk}}{\tau_d} + 2\right]\sigma_\varepsilon^2 \approx 2\sigma_\varepsilon^2 = 2\frac{2^{-2B}}{12}$$

where it was assumed that

$$n_a = \frac{\tau_{pk}}{T_{clk}} \gg 1\,, \qquad \frac{\tau_p}{\tau_d} \le 8 \text{ and } \quad \frac{T_{clk}}{\tau_d} \le \frac{1}{50}$$

For example, using $T_{clk} = 0.02\mu sec$, $\tau_d = 5\mu sec$, $\tau_{pk} = 3\mu sec$ it gives $n_a = 250 \gg 1$ and

$$\frac{\tau_{pk}}{\tau_d}\frac{T_{clk}}{\tau_d} = \frac{3}{5}\frac{20}{5000} = 0.0024 \ll 2$$

1.3.4 Error due to quantization/rounding of sum-of-product (multiplicators)

The second source of error is related to the calculation of the sum-of-product. If the sum-of-products is calculated on such a way that rounding or truncation is performed after each multiplication (in the stage 1) then error can be modeled as a noise superimposed on the exact sum of product value. In that case, it can be shown that noise variance σ_2^2 is $n_a$times larger than variance due to single multiplication. If the multiplication is done with precision of $B_m$then

$$\sigma_{22}^2 = n_a \frac{2^{-2B_m}}{12}$$

Therefore the total noise at output of the stage 2 has variance

$$\sigma_2^2 = \sigma_{21}^2 + \sigma_{22}^2 = 2\frac{2^{-2B}}{12} + n_a \frac{2^{-2B_m}}{12} = \frac{2^{-2B}}{12}\left(2 + \frac{n_a}{2^{2(B_m - B)}}\right)$$

This noise would be dominant in the case that B_m is also represented by 13, as the input signal x_n is. The influence of the second term can be calculated taking into account that $B_m = 23, B = 13, \max(n_a) = 2^{10}$, which gives value of 2^{-10}. Note that B_m could be 5 bits less precise and still its contribution to the noise due to rounding would be 2 times smaller than due to ADC quantization. But we need to keep it much more precise because of its influence on the shape of the output signal from the stage 2.Therefore we can write

$$\sigma_2^2 = 2\frac{2^{-2B}}{12} = 2\frac{2^{-2\times 13}}{12} = (0.498 \times 10^{-4})^2$$

You can verify above result by simulation: change peaking time back to $\tau_p = 3\mu sec$, precision of the coefficient b_{10} in the stage 1 to 23.23 and precision of the multiplicator to 24.23. Run the script and finally run simulation. Observe shape at edges of the rectangle (fig 16d,e).

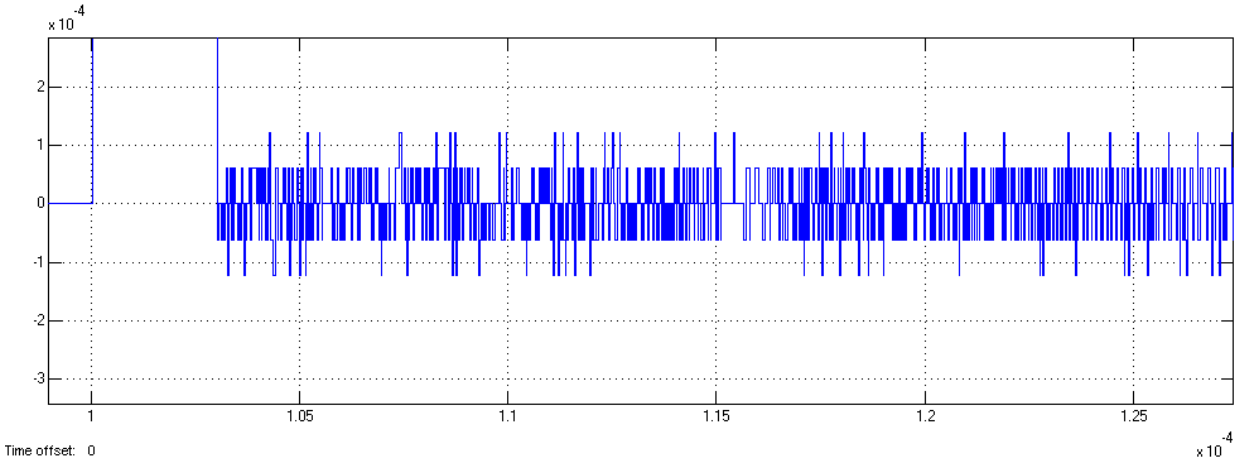


Fig 16f

nirst, notice that rectangular shape is not discorded because requirements on precision of b_{10} and multiplication in the stage 1, are both sufficient. Second notice that “noise” is les

then $3\sigma_2 = 1.5 \times 10^{-4}$ as predicted by above calculation. This means that only errors due to ADC quantization contributes to the errors in the stage 2, and they appear like a “noise”.

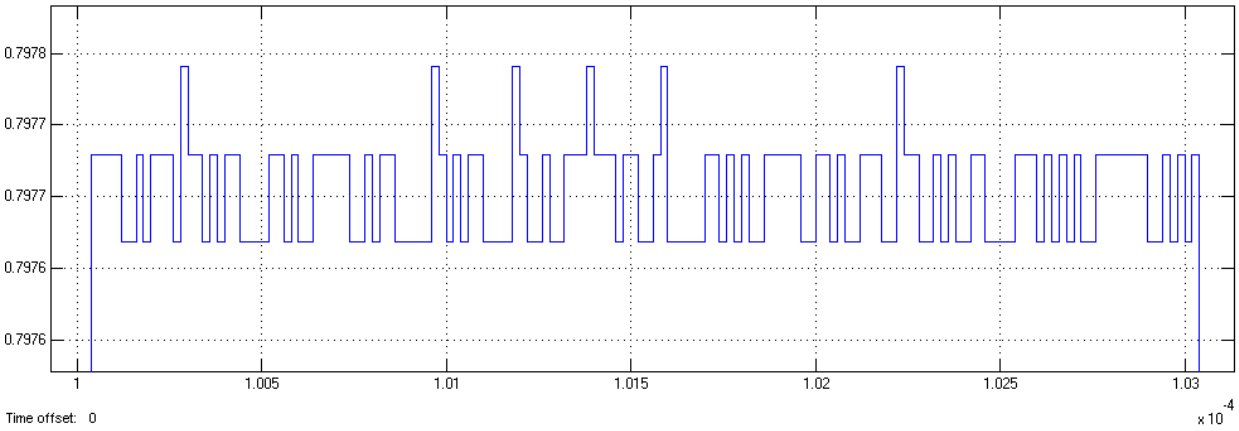


Fig 16e

1.3.5 Gain at DC frequency

If the input x_n is a constant cnst, then the output is $cnst * (1 - b_{10})$. Therefore gain at DC is

$$G_1 = 1 - b_{10} = 1 - e^{-\frac{T_{clk}}{\tau_d}} \cong \frac{T_{clk}}{\tau_d}$$

It should be noticed that clock period is order of 10ns and differentiation constant 1 to 10 microsec, therefore gain at DC=0 is of order of 1/100 to 1/1000. This has advantage that any DC offset at the output of the stage 1 is attenuated for factor of 100 to 1000, therefore it is very low.

Recall that the transfer function of the stage 2 can be rewritten in the form

$$H_2(z) = 1 + z^{-1} + \cdots + z^{-(n_a-1)}$$

The corresponding recursive relation is:

$$y_n = x_n + x_{n-1} + \cdots + x_{n-(n_a-1)}$$

In the case that input is a constant, than output is n_a - fold sum of a constant which means that gain G_2 of the stage 2 at DC is equal to

$$G_2 = n_a = \frac{\tau_p}{T_{clk}}$$

The total gain after the second stage at DC is

$$G_{12} = G_1 G_2 = \frac{T_{clk}}{\tau_d}\frac{\tau_p}{T_{clk}} = \frac{\tau_p}{\tau_d}$$

It is assumed that above ration can reach up to $8 = 2^3$ (it is usually much less because the optimal value of $\tau_p = \sqrt{3\tau_d}$ for the case of triangle). Since input signal is in the interval ± 1, and gain is 8, the output from the stage 2 can be represented by 4 bits to the left side of the binary point (3 bits for value + 1 bit for sign). Since its precision must be 23 bits to the right of the decimal point (as required by noise propagation), the stage 2 requires 27 bit accumulator and 27.23 bit subtractor in order to avoid overflows and decrease effect of error propagation to tolerable level.

After all operations in the stage 1 and 2 are done in precision required by noise propagation and overflow suppression, it is possible to round result down to 14 bits to the right of binary point, that is to output format of 18.14. This will save FPGA resources and it can be justified on the following way. The variance of error introduced by the above rounding is given by the relation:

$$\sigma_{23}^2 = \frac{2^{-2\times 14}}{12}$$

The total variance at output of the stage will be:

$$\sigma_2^2 = 2\frac{2^{-2B}}{12} + \frac{2^{-2\times 14}}{12} = \frac{2^{-2B}}{12}\left(2 + \frac{1}{4}\right) \cong 2\frac{2^{-2B}}{12}$$

In order to decrease output precision, add Converter block from Xilinx Blockset / Mat. Specify the following parameters:

Basic: Output precision: Number of bits= 18; Binary point = 14, Quantization= Round, Overflow = wrap

If DC is removed from the output of the stage 2 then the binary format of the output could be further reduced to 16.14 regardless of the ratio $\frac{\tau_p}{\tau_d}$.

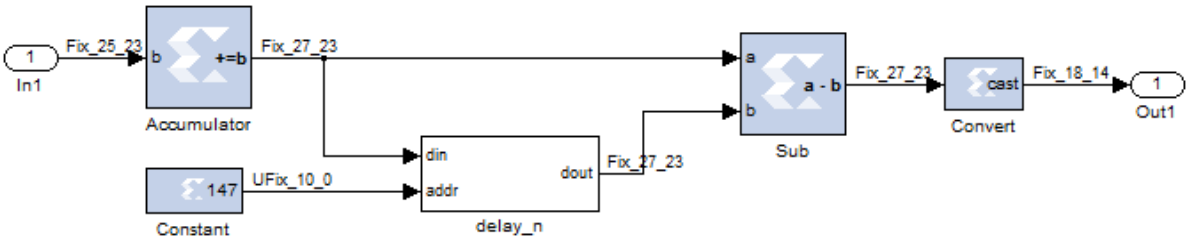


Fig 17.

The dependence of the precision of stage two on the ratio $\frac{\tau_p}{\tau_d}$ is given in the table below.

			Without DC removal	With DC removal
Maximal $\frac{\tau_p}{\tau_d}$	Stage 2 Acc width [bits]	Stage 2 Sub format [bits]	Stage 2 output format [bits]	Stage 2 output format [bits]
Less then 8	27	27.23	18.14	16.14
Less then 4	26	26.23	17.14	16.14
Less then 2	25	25.23	16.14	16.14

1.4 Stage III

The stage 3 has the following transfer function (IIR form):

$$H_3(z) = \frac{1 - z^{-n_b}}{1 - z^{-1}}$$

Transfer function of the stage 2 can be rewritten in a FIR form

$$H_3(z) = \frac{1 - z^{-n_b}}{1 - z^{-1}} = 1 + z^{-1} + \cdots + z^{-(n_b - 1)}$$

The corresponding recursive relation is:

$$y_n = x_n + x_{n-1} + \cdots + x_{n-(n_b-1)}$$

1.4.1 Design

The stage III has the same shape of transfer function as the stage II, you can use copy content of the stage 2 and past it into stage III. In order to do it, expand stage II, select its content (press Ctr-A) and press Ctr-C. After that expand stage II and, delet its content and press Ctrl-V. After coping content, remove Convert element.

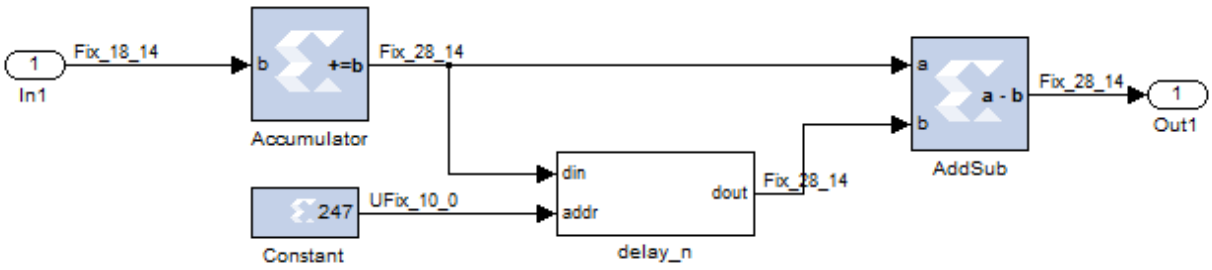


Fig 18.

Since delay in the stage 3 is defined by n_b instead of n_a, you need to define the first constant. To do it add following two lines in your parameter script file sg_filter_parameters.

```
nb = (Taupk_top+Taupk)/Tclk;
nbd = nb-3;
```

Run the file as usual in order to transfer the values into the MATLAB workspace.

In the next step, set the following parameters for the blocks:

Block name	Parameter name	Parameter value
Accumulator	Basic: Operation	Subtraction
	Output Type: Precision	User defined
	Output type: Output type	Signed
	Output type: Number of bits	28
	Output type: Binary Point	14
	Output type: Quantization	Round
	Output type: Overflow	Wrap
Constant	Basic Type	Unsigned
	Constant value	nbd
	Binary point	0
	Sampled constant	check
	Sampled period	Tclkn

Table 12

1.4.2 Precision

Recall that total noise due to ADC quantization and arithmetic operation at the output of the stage 2, which is equal to the input of the stage 3, is

$$\sigma_2^2 = 2\frac{2^{-2B}}{12}$$

The stage 3 contains only summation. The summation is done in full precision; therefore there will be no noise due to rounding/truncation. Only the noise from input to the input of the stage will propagate. The variance on the output will be

$$\sigma_3^2 = \sigma_2^2 \sum_{k=0}^{n_b-1} h_k^2 = \sigma_2^2 \sum_{k=0}^{n_b-1} 1 = n_b \sigma_2^2 = 2n_b \frac{2^{-2B}}{12}$$

On the other side, as it will be shown later, the stage 4 contains only multiplicator by factor $1/n_a$, therefore overall noise variance due to quantization will be

$$\sigma_4^2 = \frac{2n_b}{{n_a}^2} \frac{2^{-2B}}{12}$$

For a typical value of X-ray detector $\tau_d = 5\mu sec$, $\tau_p = 3\mu sec$, $\tau_{ftop} = 2$. Assuming sampling clock period $20nsec$, it follows $n_a = 250$, $n_b = 250$

$$\sigma_4^2 = \frac{2 \times 200}{200^2} \frac{2^{-2\times13}}{12} = \frac{2 \times 250}{250^2} = \frac{1}{125} \frac{2^{-2\times13}}{12} \cong 2^{-7} \frac{2^{-2\times13}}{12} = \frac{2^{-2\times16.5}}{12}$$

This means that precision due to all quantizations and roundings is effectively limited to 16 bits (note that input precision is 13 bit).

1.4.3 Gain at DC frequency

Taking into account expression for $H_3(z)$ and assuming that input is a constant, then output is n_b fold sum of the constant . This means that gain at DC (relative from output of the stage 2) is

$$G_3 = n_b = \frac{\tau_p + \tau_{ft}}{T_{clk}}$$

It is assumed that n_b can be represented by a 10 bits number, $n_b < 2^{10}$. Therefore accumulator length should be 10 bits wider then output from the stage 2, that is 30 bit in total. The precision of the subtractor should stay 14 bit to the right of the binary point. It comes out that format of the subtractor should be 30.14.

If DC would be removed from the output of the stage 2, than the output from stage 2 would have format 16.14 and consequently the stage 3 should have: accumulator width 26 bits and subtractor format 26.14. This value would still work for all $\frac{\tau_p}{\tau_d} < 8$ (apart from the period for which dc is not blocked and dc is enough large)

You can set this format before adding the DC removal block, but you should use only such values of τ_p and τ_d that $\frac{\tau_p}{\tau_d} < 1$.

		Without DC removal	
Maximal $\tau_p + \tau_{ft}$ for $T_{clk} = 20ns$ $[\mu sec]$	Maximal τ_p/τ_d $[\mu sec]$	Stage 3 Acc width [bits]	Stage 3 Sub format [bits]
Less then 5.12	Less then 8	30	30.14
Less then 2.56	Less then 4	28	28.14
Less then 1.28	Less then 2	26	26.14

Table 13

	With DC removal	
Maximal $\tau_p + \tau_{ft}$ for $T_{clk} = 20ns$ $[\mu sec]$	Stage 3 Acc width [bits]	Stage 3 Sub format [bits]
Less then 5.12	26	26.14
Less then 2.56	25	25.14
Less then 1.28	24	24.14

Table 14

The total gain from input of the stage I is

$$G_{123} = G_{12}G_3 = \frac{\tau_p}{\tau_d}\frac{\tau_p + \tau_{ft}}{T_{clk}}$$

1.4.4 Simulation

Add scope, connect it to the output of the stage 3 and start simulation. Increase data history of the scope (menu command parameters) to 20000, assuming time range is 10^{-4}. Observe the output on the scope. Stop measurement after output reach end of the scope. Note that shape is a trapezoid with significantly increased signal to noise ratio comparing to the input signal. Assuming that input signal has dc offset $DC = 0$ and amplitude $A_0 = 0$, then output amplitude A is

$$A = A_0G_1G_2G_3 = A_0\frac{\tau_p}{\tau_d}\frac{\tau_p+\tau_{ft}}{T_{clk}} = 0.8\frac{3}{5}\frac{3+2}{0.02} = 120,$$

Compare simulation with calculation.

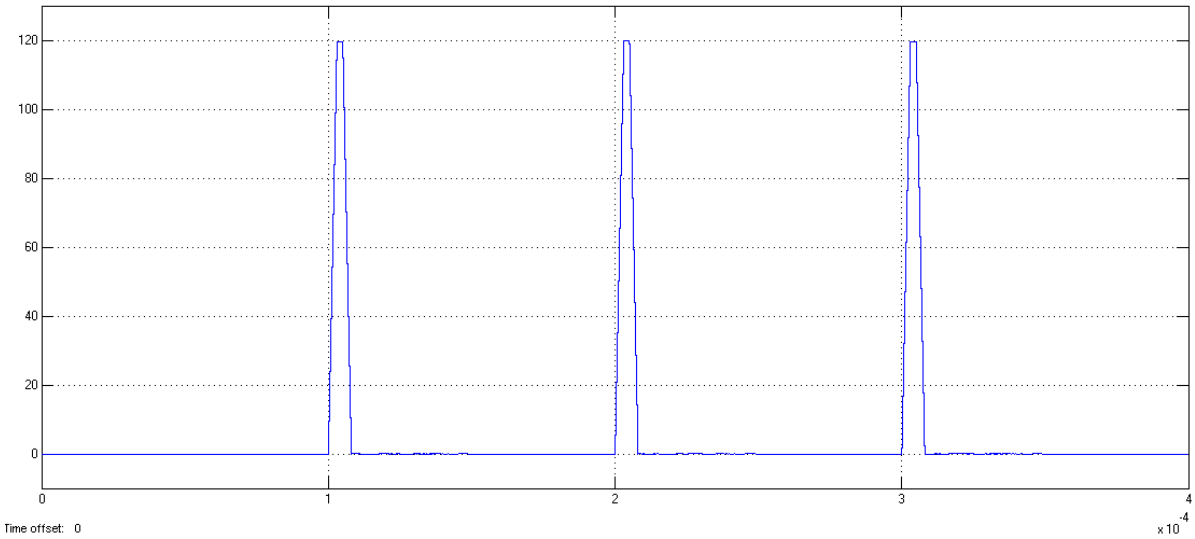


Fig 19

1.5 Stage IV

The last stage contains remaining part of the total transfer function which is

$$H_4(z) = \frac{z^{-1}}{n_a}$$

1.5.1 Design

To implement division by n_a it is more convenient to use multiplication with pre-calculated number $1/n_a$, which can be stored in a register. Therefore the stage 4 needs only one multiplicator, apart from a delay which does not affect amplitude of the final output from the overall filter. The constant should be declared in the script file as shown below

na_inv=1/na;

Run the script in order to take the change into effect.

Use the following parameters for the blocks:

Block name	Parameter name	Parameter value
Constant	Type	Unsigned
	Constant value	na_inv
	Number of bits	18
	Binary point	18
	Sample constant	yes
	Sample time	Tclkn
Delay line	Latency	1

Table 15

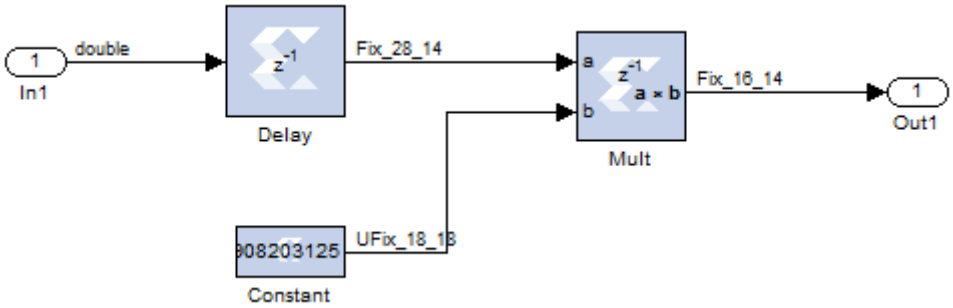


Fig 20.

1.5.2 Precision

It should be pointed out that precision of the rectangular signal at the output of the stage 2 is 14 bit, but result after the stage 4 is effectively obtained after averaging n_a values that differ only because of noise (the values forms rectangle). Therefore result has $\sqrt{n_a}$ times better precision. Since $n_a$is usually larger then 100 (peaking time 2microsec), improvement in precision is at least 10 times, that is 3 bits. It means that output precision of multiplier in the stage 4 can be increased for additional 3 bits.

1.5.3 Gain at DC frequency

The total gain at DC frequency is

$$G = G_1 G_2 G_3 G_4 = \left(\frac{T_{clk}}{\tau_d}\right)\left(\frac{\tau_p}{T_{clk}}\right)\left(\frac{\tau_p + \tau_{ft}}{T_{clk}}\right)\left(\frac{T_{clk}}{\tau_p}\right) = \frac{\tau_p + \tau_{ft}}{\tau_d}$$

In the case of a very large peaking time, the gain G can reach 8 and it requires additional 3 bits plus one sign bit (4 in total) to the left of the binary point in order to accommodate final result (assuming that input is limited to +-1). Since a DC removal will be inserted after the stage 2, the final output will be in the range +-2. Therefore output precision can be reduced to 16.14

In this case $\tau_d = 5\mu sec\ \tau_p = 3\mu sec\ \tau_{ft} = 2\mu sec$. Assuming $T_{clk} = 20ns$ it follows

$$G = \frac{20}{5000} \times \frac{3000}{20} \times \frac{5000}{20} \times \frac{20}{3000} = \frac{1}{250} \times \frac{150}{1} \times \frac{250}{1} \times \frac{1}{150} = 1$$

Therefore any DC offset at the input of the stage 1 is amplified by a factor of 1.

1.5.4 Simulation

Add scope, connect it to the output of the stage 3 and start simulation. Increase data history of the scope (menu command parameters) to 20000, assuming time range is 10^{-4}. Observe the output on the scope. Stop measurement after output reach end of the scope. Compare signal to noise ratio.

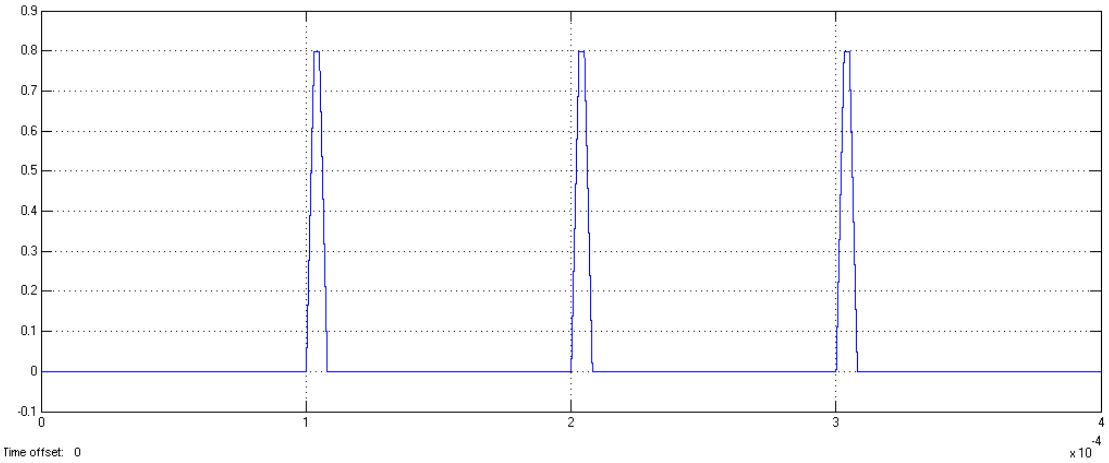


Fig 21.

Remove noise (disconnect Noise block in the pulse generator system) and start simulation. Stop it after 3 pulses passed. Zoom signal around the base and observe that error is very small (maximal value of error is around 2×10^{-4}). Compare the maximal error with variance calculated earlier.

$$\sigma_{tot}^2 = 2\frac{2^{-2B}}{12} + n_a\frac{2^{-2B_m}}{12} + \frac{2^{-2\times 14}}{12} = 2\frac{2^{-26}}{12} + \frac{400}{1024}\frac{2^{-26}}{12} + \frac{1}{4}\frac{2^{-26}}{12} = 0.28 \times 2^{-26}$$

$$\sigma_{tot} = 0.65 \times 10^{-4}$$

Which means that error is inside $3 \times \sigma_{tot}$.

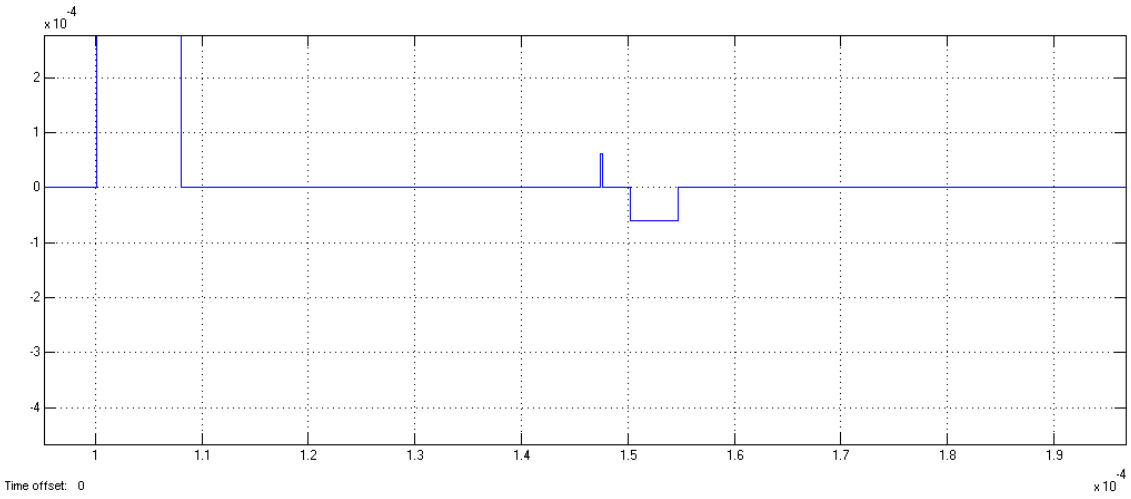


Fig 22

9. Scope subsystem

10. Bus interface subsystem

11. System generator compilation

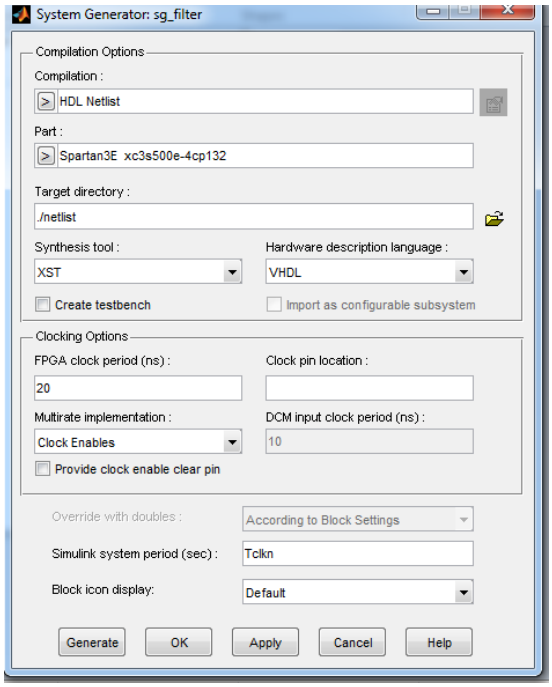
Expand top level module and click onto the System generator block. Press Generate button. After a minute or so the generation will finish. The files you will need later are created in the ../sysgen/netlist subdirectory. These are:

- sg_filter.vhd
- sg_filter_cw.vhd

, which contain VHD source code

- several files with NGC extension

,which contains detailed information of the design of special blocks like multiplicators, rams etc in terms of exact elements used in FPGA chip.

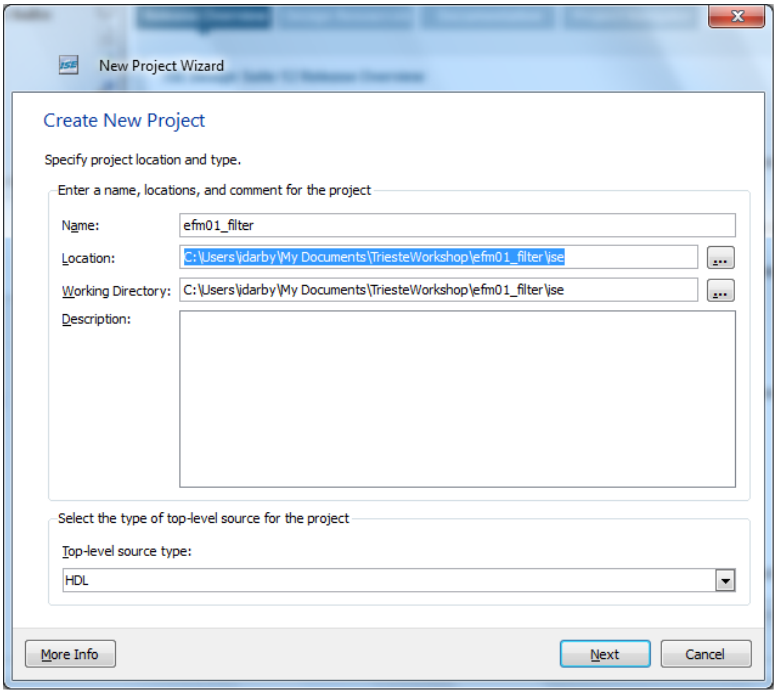


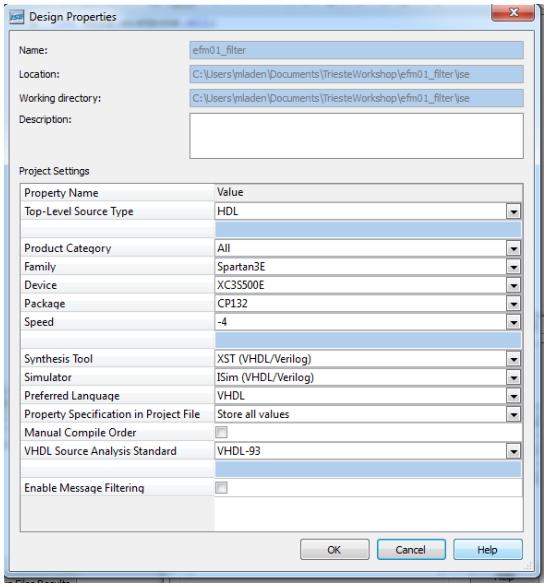
12. ISE

Open ISE by double clicking on the ISE icon located in the desktop.

Press “New Project…” button located in “Welcome to the ISE Design Suite” or using menu command “File … New Project”.

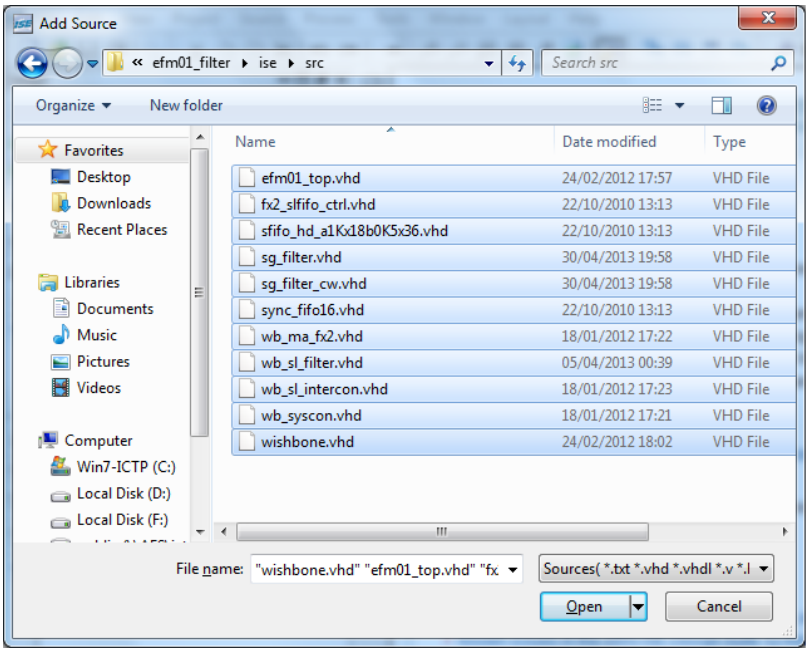
Name	efm01_filter
Location	C:\Users\idarby\My Documents\TriesteWorkshop\efm01_filter\ise
Working Directory	C:\Users\idarby\My Documents\TriesteWorkshop\efm01_filter\ise





From directory	mbogovac\TriesteWorkshop\efm01_filter\ise
Copy subdirectory	src
Into directory	C:\Users\idarby\Documents\TriesteWorkshop\efm01_filter\ise

With using menu command “Project … Add Source”, add all VHDL files in the ..\src subdirectory



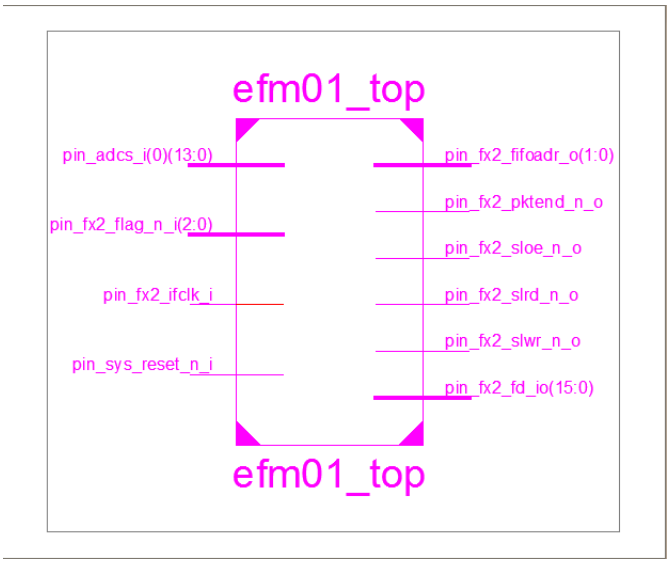
You need to define pins and constraints:

From directory	mbogovac\TriesteWorkshop\efm01_filter\ise\ucf
Copy file	efm01_ucf

Into directory	C:\Users\idarby\Documents\TriesteWorkshop\efm01_filter\ise

The file assigns ports of the top level module to physical FPGA I/O pins.

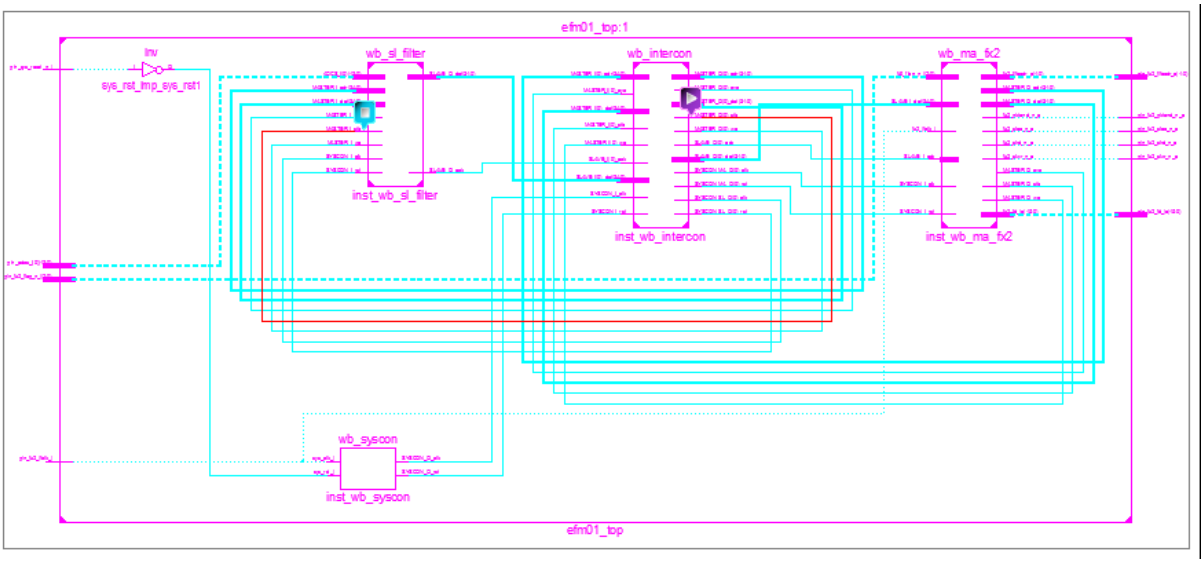
Now you can synthesize the design. First Select “efm01_top – RTL” in the Design View Hierarchy. Than right click on Synthesize –XST and select Run. It starts an external program called XST.EXE. During synthesize, the program translates VHDL code into a netlist: set of primitive components like mulitplexers – MUX5, …, MUX8, Arithmetic functions (DSP48, MULT18x18), Clock Buffers (IBUFG, BUFG), Shift register LUTS (SRL16, SRL32), RAM (block and distributed) … and their interconnections. The primitives are defined in a library called UNISIM and they are suitable for functional simulation. definThe netlist is saved in a file with extension NGC. You may view graphical representation of the netlist. To do it double click on View RTL schematics. It opens a dialog box. Select “Start with schematics of the top level”.



It shows you top level module with pins that are defined in a user constraints files. These are physical pins of FPGA

To see inside the module, right click on the block and select “Show block content”. You may unzoom it by using corresponding tool in the ISE Project Navigator toolbar. Notice basic components that are instantinated

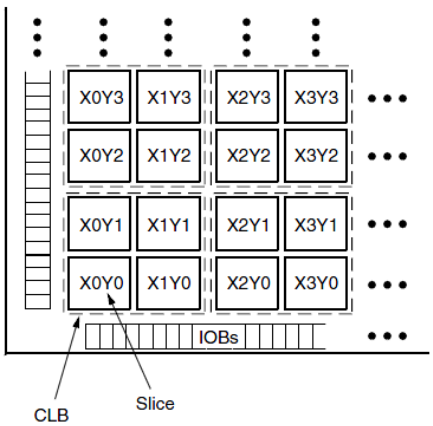
component	instance	
wb_sl_filter	Inst_wb_sl_filter	
wb_intercon	Inst_wb_intercon	
wb_syscon	Inst_wb_syscon	
wb_ma_fx2	Inst_wb_ma_fx2	



In the next phase you will generate FPGA programing file. First right click on "Generate Programming File" and select Process Properties. It opens new dialog box. Check Create binary configuration file and exit from dialog by pressing OK. Again right click on the "Generate Programming File" and select Run. It starts three remaining processes.

- Translation – program NGDBUILD.EXE translates generic primitives in NGC file to another primitives (defined in SIMPRIM library) that are suitable for timing simulation. The result is stored in a file with extension NGD
- Map – program MAP.EXE maps SIMPRIM primitives into FPGA physical elements like CLBs and IOBs of the target FPGA:
- Place and Routing – PAR.EXE

The Configurable Logic Blocks (CLBs) constitute the main logic resource for implementing synchronous as well as combinatorial circuits. Each CLB contains four slices (two pairs called SLICEM and SLICEL), and each slice contains two Look-Up Tables (LUTs) to implement logic and two dedicated storage elements that can be used as flip-flops or latches. The LUTs can be used as a 16x1 memory (RAM16) or as a 16-bit shift register (SRL16), and additional multiplexers and carry logic simplify wide logic and arithmetic functions. The Spartan3E-500 has 1164 CLBS organizing in 46 rows and 34 columns, 8672 slices and 17344 LUTS/Flip-Flops.



13. Design verification

Connect output of the AFG 3021B to Oscilloscope and terminate it with 50 Ohm. Press “Channel On” button on the AFG3021B.

Select output waveform of AFG 3021B to “Exponential decay time” as follows:

Press Function “More…” button. A menu appears on the right side of the display. Select “-more-“ by pressing button that is on the right side of the text and at the same height”. On a new menu select “Exponential Decay”. After that use Frequency/Period, Amplitude and Offset buttons to select values listed in the table below:

Parameter	Button	Value
Frequency	Frequency/Period	19.8 kHz
Amplitude	Amplitude/High	2.0 Vpp
Offset	Offset/Low	100 mV

In order to select above values, use knob and left, right arrows below the knob rather than keyboard. Press double time Frequency to get units in kHz. Observe output on the Scope.

You need to set proper output impedance, low and high limit and invert polarity (because of internal gain of -1 on the FPGA board). To do this use “Output menu”. The output menu can be accessed by pressing “Top Menu” button – it is listed on the bottom of the display.

Select

Parameter	Value
Load Impedance	50 Ohm
Invert	On
Offset	100 mV
High Limit	+3 V
Low Limit	-3 V

The AFG 3021B outputs Exponential decay signal

$$A_{out}(t) = A_{offset} - \frac{A_{set}}{2} e^{-\frac{t}{\tau_d}}$$

Therefore you should observe exponential decay signal with amplitude -1.0 V and 100mV offset. Connect signal to the FPGA board. You can use T- BNC connector to split signal temporary. In that case amplitude will drop because FPGA input impedance is also 50 Ohm and AFG2021B cannot drive two 50 Ohm inputs.

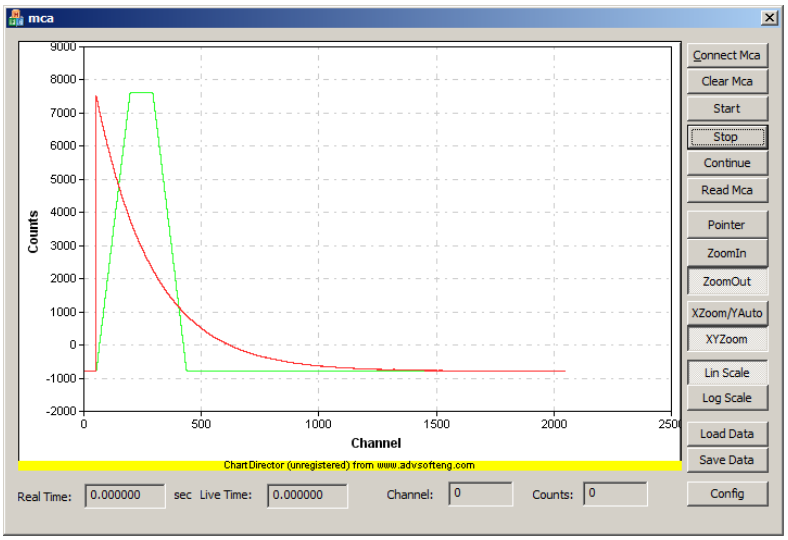
Start Visual Studi 2008 and open Mca project by using File Open Project/ Solution menu command . It open a dialog box. Browse to My Document\ Workshop\ efm01_filter \cpp directory and select "Mca" file of type Microsoft Visual Studio Solution.

Press menu command "Debug … Start Without Debugging" or Ctrl F5. It runs program.

In the program's graphical user interface (GUI) press "Connect MCA" button. This command will automatically load efm01.bin file into the FPGA (see class CEfm01Board, method Open, which invokes lower level method ProgramFPGAFromBIN from class ceDevice provided by vendor of the small FPGA module). To start filtering press button "Start". Observe Input exponential signal (red) and output trapezoidal signal from the filter (green) on the screen of the GUI. The default parameters for the trapezoidal signal used by the program are: differentiation constant $\tau_d = 5\ \mu sec$, peaking time $\tau_p = 3\ \mu sec$, flat top time $\tau_{ft} = 2\ \mu sec$. It can be changed with "Config" dialog box (can be accesed by pressing "Config" button)

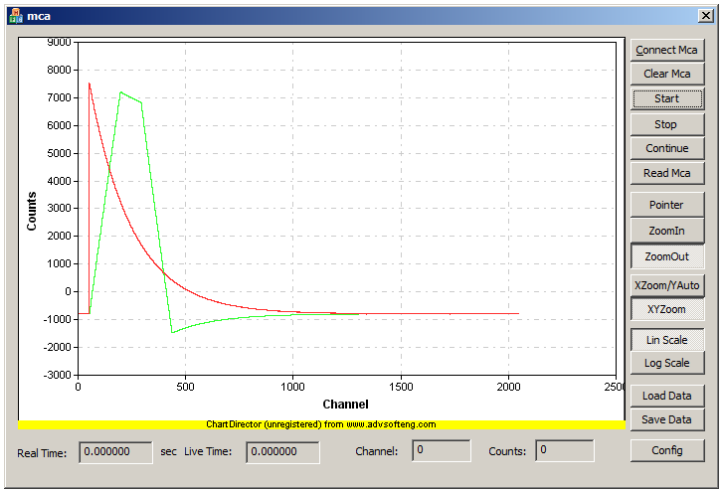
Notice that sides of the trapezoid have duration of 150 channels. Since 1 channel = 1 clock period = 2ns, it means that duration is 3 usec. It is equal to the peaking time as it should be. Also notice that duration of the flat – top is 100 channels, that is 2 usec.

Zoom display around edges of the trapezoid and check that they are very sharp.



Change Frequency of the input signal by turning knob of the AFG3021B clockwise to increase freq to 23.8 kHz and observe shape of the trapezoidal signal. Note that shape is distorted. By changing the frequency you actually change decay time, that is differentiation constant. This means that coefficient b_{10} of the filter transfer function does not match true value and signal is deformed. The same effect you may produce by changing differentiation time constant in the Config dialog

$$b_{10} = e^{-\frac{T_{clk}}{\tau_d}}$$



You may conclude that mismatch between assumed and real decay time of the input signal may introduce errors in the amplitude calculation.

Set frequency back to 19.8 kHz. Add small capacitor (4,7 nF) to ground at the output of the AFG3012 (use T-BNC connector). Oserve that trapezoid signal has deformed again. It gets “rounded left-top and right-bottom corners. The reason is that rise time of the exponential signal has increased and it does not match perfect exponential signal. Mathematically speaking, the input signal representation in the s-domain and later after ADC conversion in z-domain has an additional pole. In order to get trapezoidal signal in such a case, another transfer function should be used in the filter.

Again the problem that can arise is that, in the case of pile-up, amplitude of the second signal can be miscalculated and resolution can be distorted.

Observe that DC of both signals is equal. In order to explain it, recall that according to theoretical discussion, DC gain is

$$G = \frac{\tau_p + \tau_{ft}}{\tau_d}$$

An calculate gain assuming $\tau_d = 5\ \mu sec$, $\tau_p = 3\ \mu sec$, $\tau_{ft} = 2\ \mu sec$. Now increase the offset of the input signal by turning the knob on the AFG3021B clockwise and observe on the GUI display that DC offset of both signals decreases (inverted input).

The problem that can arise in a real spectroscopy system is that DC component of the signal can change slovely during measurement.. Since amplitude is always calculated in reference to a constant DC value, it will introduce degradation of resolution. To solve this problem it is necessary to block any DC component of the signal. To do this you need to go back to the design.

14. Adding DC Blocker

15. Testing DC-Blocker

After producing new FPGA configuration file efm01_filter.bin, copy it into the Document\ Workshop\ efm01_filter \cpp \Debug directory

Adjust output of the AFG3021B to: Amplitude = 2V, offset = 100mV. Run GUI program, connect to MCA and start measurement.

Notice that DC level of the signal is 0. Now change offset by turning knob clockwise in steps of 100mV. After each step observe input signal and shaped signal offset. You can go up to around 1V offset (theoreticaly -8192 in channels)

Now increase amplitude, up to around 3.8V and observe both signals on the GUI display. Note that you can go up to 16k in channels. You may observe on the real oscilloscope that input signal is around 2Vpp and DC offset around -1V.

It should be pointed out that under the current conditions (DC close to -1 and amplitude around 2V, ADC is most efficient utilized. This means that in the measurement with real signals from detector, initial DC should be close to -1V and input gain should be adjusted on such a way that amplitude of the signal for highest energy is close to 2V. In a real situation, a noise will be superimposed onto the input signal. Also all amplitudes of the noise must be sampled. Therefore, the true value of the initial DC and gain depends also on the noise.