گزارش انجام پروژه‌ی درس زبان توصیف سخت‌افزار و مدارات

طراحی و بهینه‌سازی FIR Filter

استاد:

دکتر مجید نبی

دانشجویان:

محیا جمشیدیان ۹۵۲۵۱۳۳

فاطمه رحمانی ۹۵۲۷۰۸۳

نگار سخایی ۹۵۲۸۰۰۳

زینب صادقیان ۹۵۲۹۳۶۳

اسحاق موتابی ۹۳۳۱۵۷۳

تیر ماه ۱۳۹۷

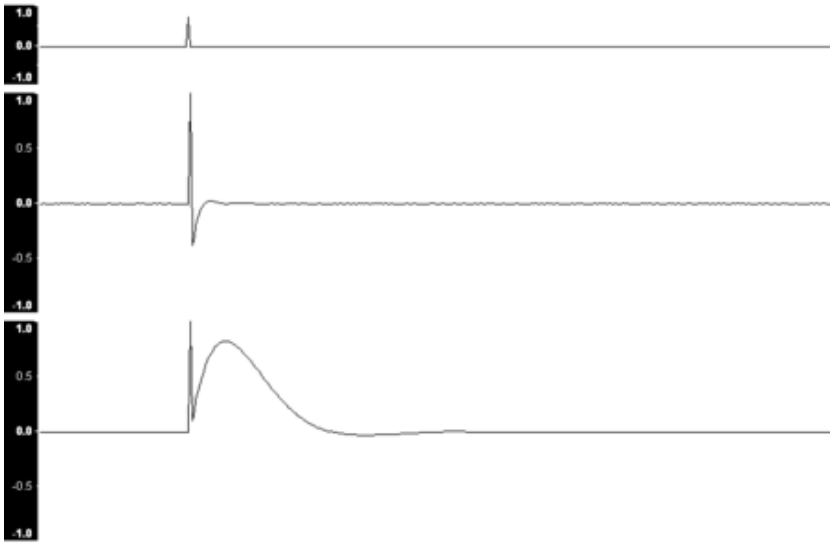
فهرست

● فاز اول

فاز اول را به چند بخش تقسیم میکنیم:

بخش اول: معرفی فیلتر FIR :

ابتدا بایستی با مقدمات فیلتر FIR آشنا شویم. فیلتر FIR که به اختصار **finite impulse respons** می‌باشد در واقع فیلتری است که برای پردازش سیگنال ها استفاده میشود. impulse response به سیگنال خروجی‌ای گفته می‌شود که به صورت یک ورودی کوچک و لحظه‌ای به سیستم داده شده باشد(شکل زیر).



بنابراین FIRشامل تعداد محدودی سیگنال پالس می‌باشد و در مقابل فیلتر IIR وجود دارد که محدود به تعدادی پالس نیست.

سیگنال مورد نظر ورودی شامل تعدادی نویز است که اگر waveform آن را مشاهده کنیم به صورت یک تعدادی پالس دیده میشود. پس این فیلتر عملا میتواند برای تغییر دادن این پالس ها مناسب باشد.

اما این فیلتر در حوزه زمان گسسته عمل می‌کند پس بایستی ابتدا سیگنال از زمان پیوسته تبدیل به زمان گسسته شود.که این‌کار با عمل نمونه برداری می‌توان انجام داد.

سپس ورودی گسسته به فیلتر داده می‌شود و در نهایت خروجی مورد نظر تنها برای یکی از داده ها بدست می آید.

این فیلتر یک فیلتر پایین‌گذر است که داده ها حول صفر بیشترین مقدار خود را دارند و داده های کناری بایستی مقدار کمتری داشته باشند.

شکل کلی یک فیلتر FIR به صورت مقابل است:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \\ &= \sum_{i=0}^{N} b_i \cdot x[n-i], \end{aligned}$$

که در واقع مجموع حاصل ضرب تعدادی ضریب در مقدار سیگنال نمونه برداری شده‌ی ورودی هستند. این ضرایب در واقع از روش‌های مختلفی بسته به نوع سیگنال ورودی و نوع کاری که قرار هست انجام شود ایجاد می‌شود. برای محاسبه بهتر می‌توان برای ضرایب فرمول و محاسبات مختلفی انجام داد که در اینجا ضرایب داده شده است. اما برای مثال شکل زیر یک نمونه دیگر از محاسبه ضرایب است که کاربردهای دیگری دارد.

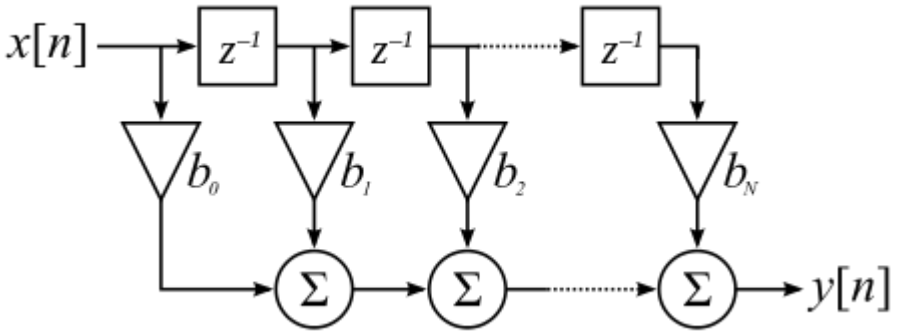
Symmetry of Coefficients
even: $b(k) = b(n+2-k), \quad k = 1, \ldots, n+1$
even: $b(k) = b(n+2-k), \quad k = 1, \ldots, n+1$
odd: $b(k) = -b(n+2-k), \quad k = 1, \ldots, n+1$
odd: $b(k) = -b(n+2-k), \quad k = 1, \ldots, n+1$

همانطور که دیدیم این فیلتر در واقع Convolution در حوزه زمان گسسته است که به نام discrete convolution شناخته می‌شود.

از کاربرد های این فیلتر می‌توان به حذف نویز در سیگنال های صوتی در ارتباطات مخابراتی نام برد برای مثال در رادیو های دیجیتالی سیگنال آنالوگ به دیجیتال (A/D Converter) تبدیل می‌شود و یا ممکن است که برعکس آن انجام شود.این نوع رادیو ها از این فیلتر برای انتخاب فرکانس مناسب بدون نیاز به عوض کردن سخت افزار استفاده می‌کنند.

بخش دوم: نحوه عملکرد فیلتر:

در این بخش بررسی دقیق‌تری میکنیم بر فیلتر FIR . همانطور که در شکل زیر مشاهده می‌شود:



این فیلتر شامل N بخش یکسان است که شامل یک ضرب کننده و یک جمع کننده است. بنابراین در این سوال ۱۹ ضریب در هر کدام از ۴۴۱۰۰۰ داده ورودی که در واقع از همان نقاط نمونه برداری شده است تک به تک باید ضرب شود و در انتها برای یک ورودی همه این ۱۹ حاصله باید جمع شوند و این عمل برای کل نقاط نمونه برداری شده باید تکرار شود.

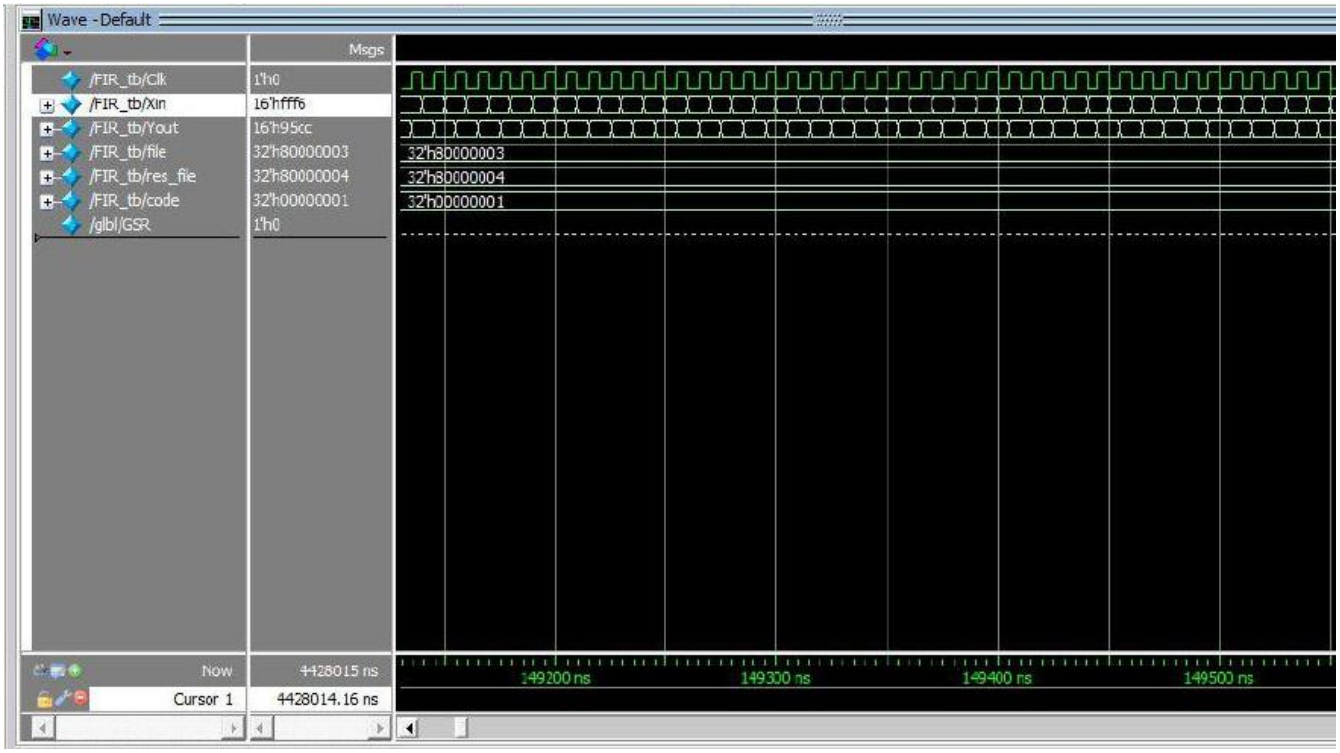
در نهایت هرکدام از این خروجی ها درواقع فیلتر روی آن اعمال شده است.

همانطور که از مقادیر ضرایب مشخص است این فیلتر تمرکز خود را حول نقطه صفر گذاشته است. به عبارت دیگر مقادیر حول صفر دارای دیتای مشخص‌تر و مهم‌تری هستند و سایر نقاط نویز است و باید با مقادیر ناچیز ضرایب ضرب شوند . در این صورت با این عمل تاثیر این مقادیر کمتر خواهد شد.

بخش سوم: توضیح کد ، خروجی ها و **waveform**:

در این قسمت کمی به صورت خلاصه کد و توضیحات مربوطه را ارائه می‌دهیم:

شکل زیر waveform خروجی حاصله از کد برنامه است:

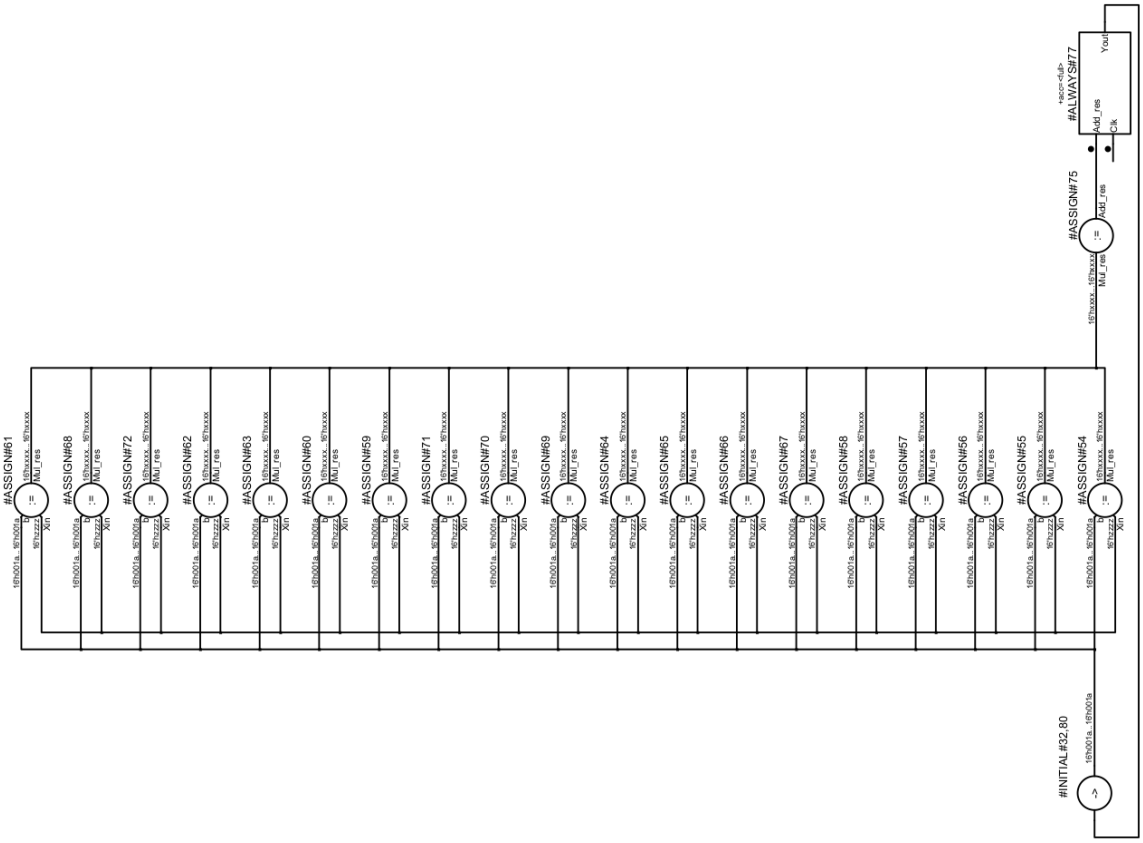


در این قسمت خروجی های هر قطعه را به قطعه دیگر به عنوان ورودی جمع کننده دادیم که در آخر تمام این ۱۹ قطعه را با هم جمع کند.

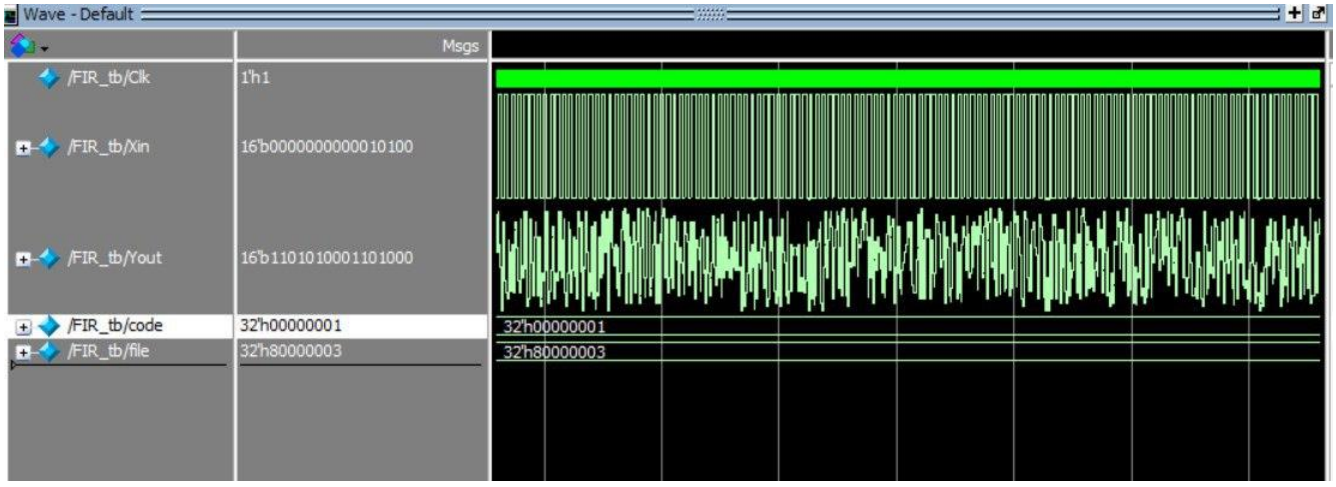
فایل باینری خروجی و نیز شکل مداری کد و نحوه اتصال کلاک و جمع کننده و ضرب کننده ها نیز در پیوست ذخیره شده است. در نهایت مقدار منابع مصرفی و نیز شکل خروجی مدار مورد نظر را نشان می‌دهیم :

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of 4 input LUTs:                822 out of   1,536   53%
Logic Distribution:
  Number of occupied Slices:             486 out of     768   63%
    Number of Slices containing only related logic:     486 out of     486 100%
    Number of Slices containing unrelated logic:          0 out of     486   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:          830 out of   1,536   54%
    Number used as logic:                822
    Number used as a route-thru:           8

  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

  Number of bonded IOBs:                  33 out of     124   26%
    IOB Flip Flops:                       16
  Number of MULT18X18s:                    4 out of       4  100%
  Number of BUFGMUXs:                      1 out of       8   12%

Average Fanout of Non-Clock Nets:                2.19

Peak Memory Usage:  368 MB
Total REAL time to MAP completion:  6 secs
Total CPU time to MAP completion:   2 secs
```

```
Device Utilization Summary:

   Number of BUFGMUXs                        1 out of 8      12%
   Number of External IOBs                  33 out of 124    26%
      Number of LOCed IOBs                   0 out of 33      0%

   Number of MULT18X18s                      4 out of 4     100%
   Number of Slices                        486 out of 768    63%
      Number of SLICEMs                      0 out of 384     0%



Overall effort level (-ol):   High
Placer effort level (-pl):    High
Placer cost table entry (-t): 1
Router effort level (-rl):    High

Starting initial Timing Analysis.  REAL time: 2 secs
Finished initial Timing Analysis.  REAL time: 2 secs


Starting Placer
Total REAL time at the beginning of Placer: 2 secs
Total CPU  time at the beginning of Placer: 1 secs
```



خروجی نهایی بعد از تبدیل به سیگنال انالوگ توسط model sim هم به صورت زیر می‌باشد:



● فاز دوم

حال با در نظر گرفتن فاز اول پروژه و پیاده‌سازی معمولی این فیلتر، تلاش شد که آن را با توجه به زمان و یا مساحت مورد نیاز روی FPGA، بهینه سازی کنیم. این فاز در دو مرحله Resource Sharing و Pipelining انجام شد که شرح آن‌ها به صورت زیر است:

بهینه سازی به کمک **Resource Sharing** :

اولین تلاش برای بهینه‌تر کردن کد، استفاده از منابع مشترک است. در این مرحله، فقط یک ضرب کننده و یک جمع کننده از منابع FPGA مورد استفاده قرار گرفت و در هر کلاک، اعمال ضرب و جمع پشت سر هم توسط همان یک instance از Adder و Multiplier انجام می‌شوند.

دو ماژول مورد استفاده، برای بهینه سازی هر چه بیشتر کد، توسط core های آماده Xilinx پیاده‌سازی شده اند. ماژول ضرب کننده، برای استفاده کمتر از LUT های FPGA، از Multiplier های آماده در ساختار مدار استفاده می‌کند.

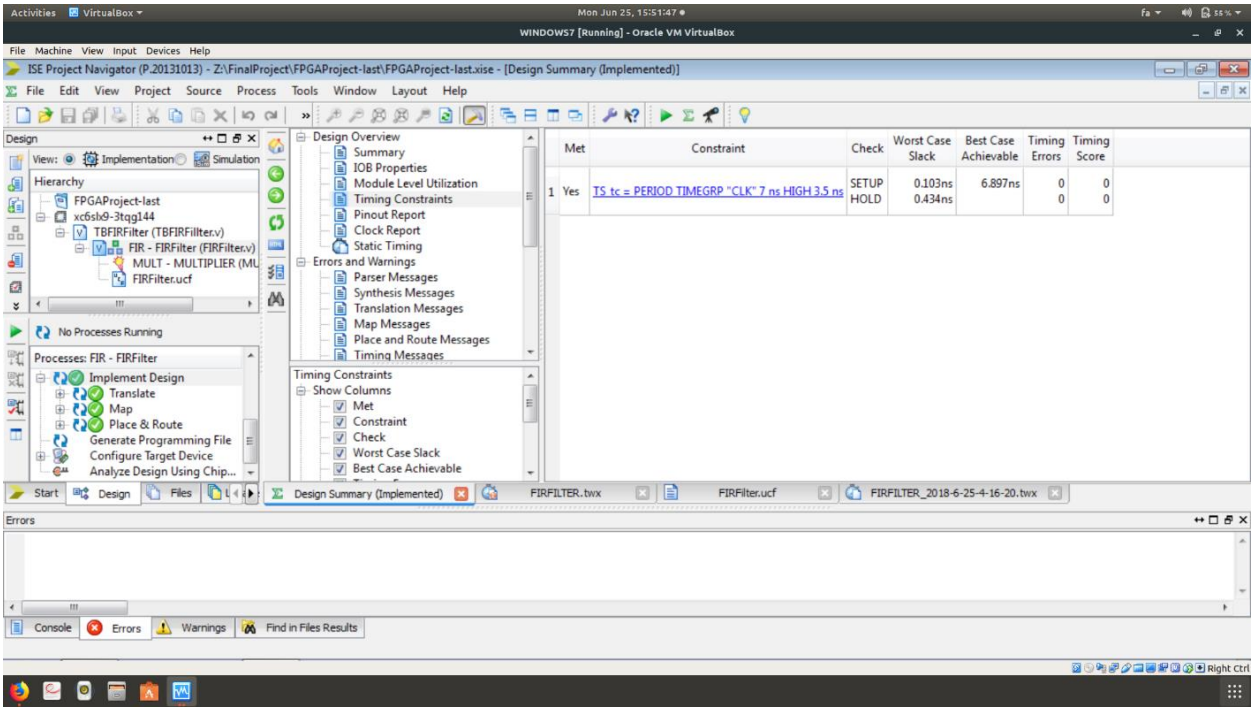
ماژول طراحی شده، در هر کلاک یک داده جدید دریافت کرده و با استفاده از آن و نمونه‌های قبلی و ذخیره شده، نتیجه را محاسبه می‌کند. زمانی که خروجی ماژول قابل اعتماد بود، خروجی data_valid برابر با یک خواهد شد. قابل توجه است که این ماژول برای کلاک‌های ابتدایی، خروجی قابل اعتمادی ندارد، چون منتظر می‌ماند تا اندازه tap های فیلتر، ورودی دریافت کند.

در نهایت، خروجی فیلتر بعد انجام تمام ضرب و جمع‌ها، یک عدد۳۳ بیتی خواهد بود که برای مقاصد ما قابل قبول نیست (خروجی باید هم عرض با ورودی، برابر با ۱۶ بیت باشد). برای دقیق‌تر بودن جواب این ماژول، خروجی صرفاً truncate نمی‌شود، یعنی به سادگی ۱۷ بیت کم‌ارزش‌تر را دور نمی‌ریزیم، بلکه آن را به نزدیک‌ترین عدد صحیح ۱۶ بیتی گرد می‌کنیم. (ر.ک. به پیوست اول).

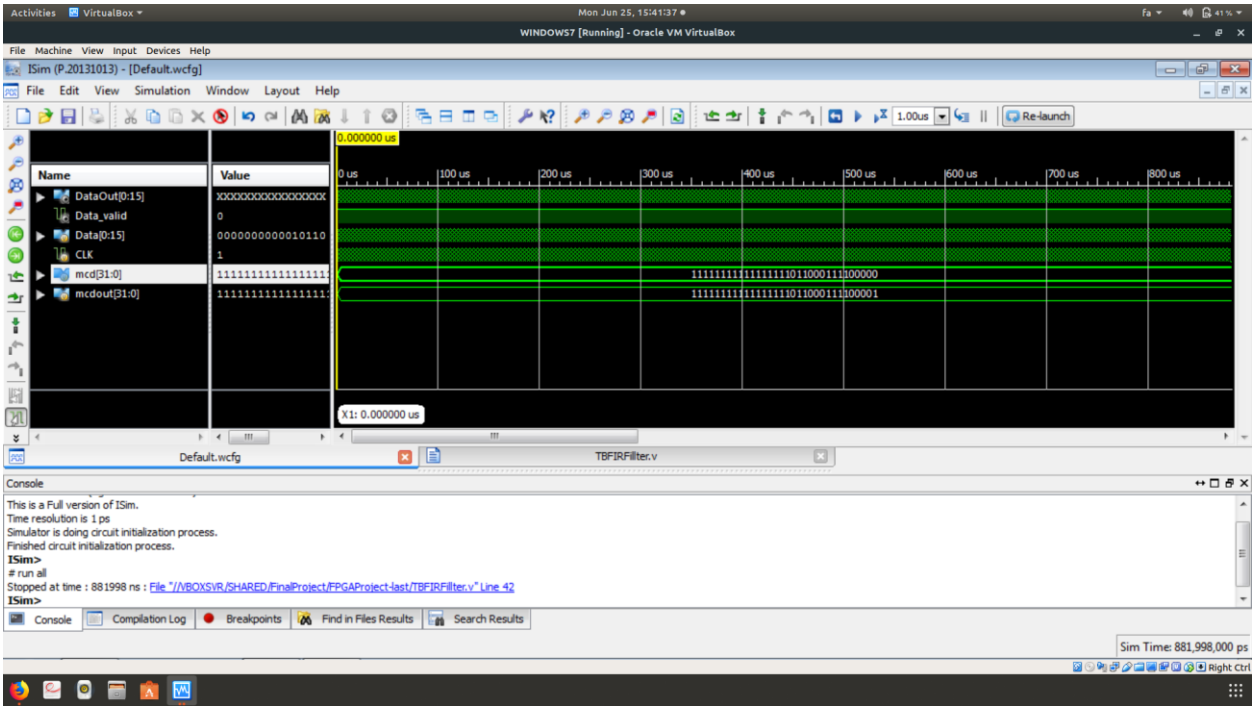
در نتیجه، منابع استفاده شده پس از سنتز طراحی، به شکل زیر خواهد بود:

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                    54 out of  11,440    1%
    Number used as Flip Flops:                  54
    Number used as Latches:                      0
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                         50 out of   5,720    1%
    Number used as logic:                       47 out of   5,720    1%
      Number using O6 output only:              28
      Number using O5 output only:               4
      Number using O5 and O6:                   15
      Number used as ROM:                        0
    Number used as Memory:                       0 out of   1,440    0%
    Number used exclusively as route-thrus:      3
      Number with same-slice register load:      2
      Number with same-slice carry load:         1
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                    20 out of   1,430    1%
  Number of MUXCYs used:                        32 out of   2,860    1%
  Number of LUT Flip Flop pairs used:           55
    Number with an unused Flip Flop:            10 out of      55   18%
    Number with an unused LUT:                   5 out of      55    9%
    Number of fully used LUT-FF pairs:          40 out of      55   72%
    Number of slice register sites lost
      to control set restrictions:               0 out of  11,440    0%
```

بهترین کلاک قابل دستیابی، توسط نرم‌افزار تقریباً ۶٫۸ نانوثانیه تشخیص داده شد.



نکته‌ی دیگر اینکه در ماژول‌های نوشته شده، یک کلاک ورودی وجود دارد و نمونه برداری از فایل هم با همان کلاک انجام می‌شود، بنابراین سرعت اجرا و اتمام برنامه کمتر از فاز سوم خواهد بود. نتیجه سیمولیشن نیز به شکل زیر می‌باشد:



بهینه سازی با **Piplining**:

این قسمت از یک ماژول با سیگنال های ورودی دیتا، کلاک و دو سیگنال کنترلی استارت (start) و دان(done) تشکیل شده است.

با کمی دقت متوجه می‌شویم که ضرایب داده قرینه هستند(فیلتر ما symmetric است)، یعنی ضریب ۰ با ۱۹، ضریب ۱ با ۱۸ و به همین ترتیب با هم برابرند. پس تصمیم گرفتیم حین عملیات روی ۱۹ داده ابتدا داده های اول و آخر را با هم جمع کنیم و سپس در ضرایب مذکور ضرب کنیم. به این ترتیب به طور کلی یک ضرب کننده و دوتا جمع کننده نیاز داریم (با این روش یک ضرب از کل عملیات کم می‌شود). یکی از جمع کننده ها برای عملیاتی که توضیح دادیم، دیگری برای جمع کردن هر ۱۹ داده استفاده شده است.

هدف از وجود دو سیگنالی کنترلی تعریف شده این است که بدانیم کی عملیات روی ۱۹ داده در حال محاسبه تمام شده است تا داده جدید را ورودی بدهیم! یعنی هر y[n]که تولید می‌شود سیگنال دان یک می‌شود، با شروع شدن محاسبه جدید (که استارت آن را تعیین می کند) سیگنال صفر می‌شود.(به مدت یک کلاک سیگنال دان یک باقی می‌ماند.)

با آگاهی به قرینه بودن ضرائب برای انجام دادن اولین ۱۹ داده ورودی ۱۰ کلاک کافی است و در هر استیج پایپ لاین سه عملیات به صورت موازی انجام می‌شود(دو جمع و یک ضرب).(که اینکه در هر کلاک چه اتفاقی بیفتد با یک متغیر j و گذاشتن تعدادی رجیستر برای نگهداری نتیجه هر عمل مدیریت می‌شود.)

برای جمع کننده ها و ضرب کننده ها از Core خود آی ای ای استفاده شده است.

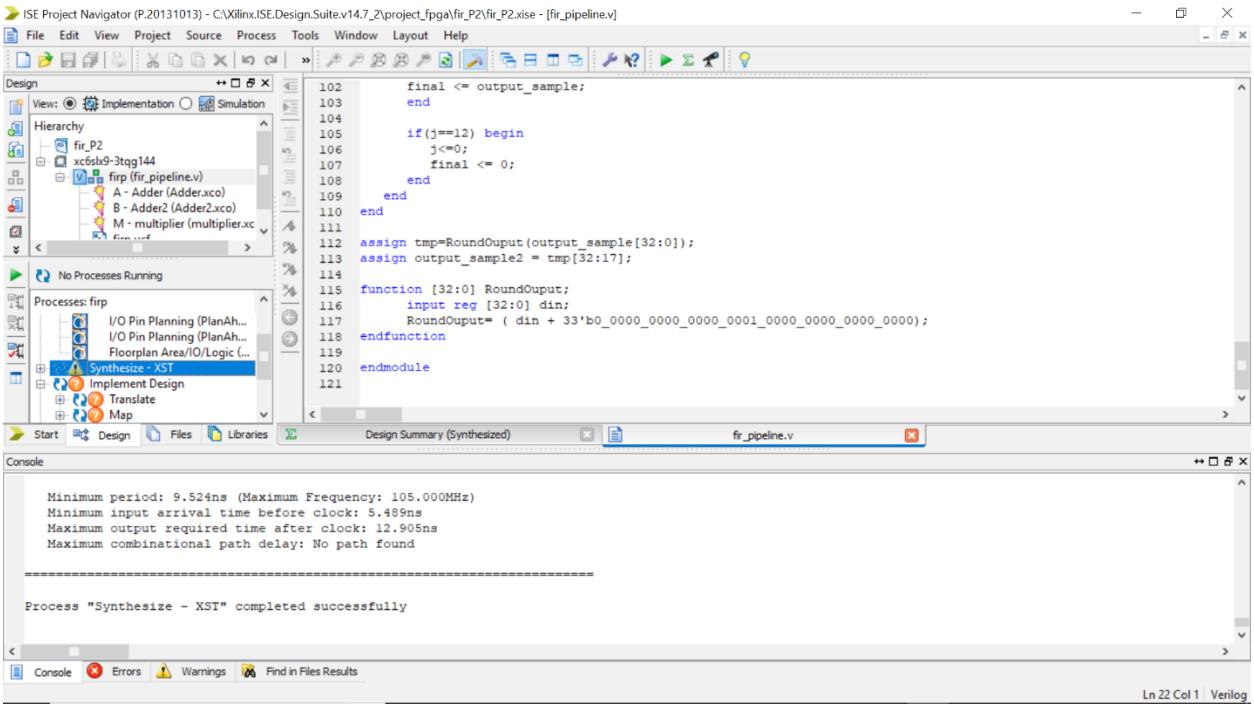
و در نهایت با توجه به ضرب ها و جمع‌هایی که روی داده ۱۶ بیتی ورودی انجام می‌شود به ناچار ۳۳ بیت خروجی خواهیم داشت! اما خروجی ماژول ۱۶ بیتی است. با تعریف فانکشنی به نام RoundOutput (ر.ک پیوست) خروجی محاسباتمان را گرد میکنیم و ۱۶بیت پر ارزش آن را خروجی می‌دهیم.

کلاکی که با این روش به دست آوردیم در شکل زیر مشاهده می‌شود:

```
Minimum period: 9.524ns (Maximum Frequency: 105.000MHz)
Minimum input arrival time before clock: 5.489ns
Maximum output required time after clock: 12.905ns
Maximum combinational path delay: No path found
```

که به طور دقیق تر در فایل داده شده همراه با گزارش می‌توان آن را بررسی کرد. سعی شد که در قسمت time constraint با دادن کلاک های بهتر بررسی کنیم ببینیم که آیا می‌توان به کلاک پایین تری دست یافت یا خیر. اما بهترین کلاکی که به دست آوردیم همین بود.

به علاوه امتحان کردیم که از ضرب کننده و جمع کننده هایی که خودشان نیز پایپ لاین هستند، استفاده کنیم که ببینیم نتیجه بهتری به دست می‌آید یا خیر ولی نتیجه گرفتیم برای مورد ما استفاده از ضرب کننده و جمع کننده های پایپ لاین شده نتیجه کلی را بهبود نمی‌بخشد.

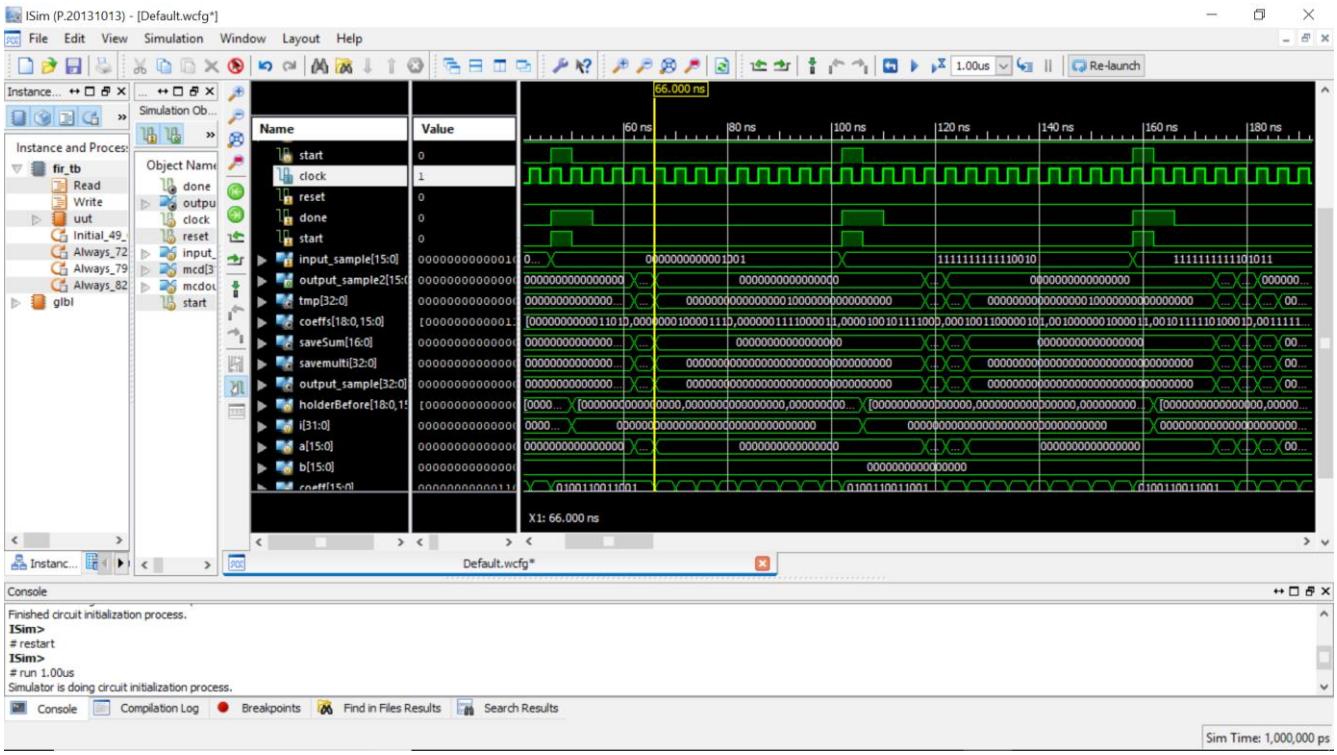


از آن جایی که برای پایپ لاین کردن از رجیستر باید استفاده میکردیم حدود ۳٪ از کل رجیستر ها را برنامه استفاده می‌کند.

در ادامه تمام حافظه ای که برای Implement نیاز داریم در شکل زیر دیده می‌شود:

```
Slice Logic Utilization:
 Number of Slice Registers:              388  out of  11440     3%
 Number of Slice LUTs:                   332  out of   5720     5%
    Number used as Logic:                332  out of   5720     5%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     649
   Number with an unused Flip Flop:      261  out of    649    40%
   Number with an unused LUT:            317  out of    649    48%
   Number of fully used LUT-FF pairs:     71  out of    649    10%
   Number of unique control sets:          6

IO Utilization:
 Number of IOs:                          36
 Number of bonded IOBs:                  35  out of    102    34%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                1  out of     16     6%
 Number of DSP48A1s:                      1  out of     16     6%
```

در نهایت نمایی از کل موج نهایی را مشاهده می‌کنیم:



فاز سوم

در نهایت، پس از تلاش برای بهینه‌سازی ماژول‌هایی که خودمان برای FIR Filter نوشته ایم، از Core آماده Xilinx برای این مقصود استفاده می‌کنیم. این Core ها، توسط متخصصینی طراحی شده که اطلاعات کامل در مورد ساختار FPGA ها دارند و به همین دلیل، استفاده از آن‌ها ساده‌تر و مطمئن‌تر از طراحی ماژول ها به صورت دستی ست.

در این فاز، از FIR Compiler ورژن ۶,۳ استفاده می‌کنیم.

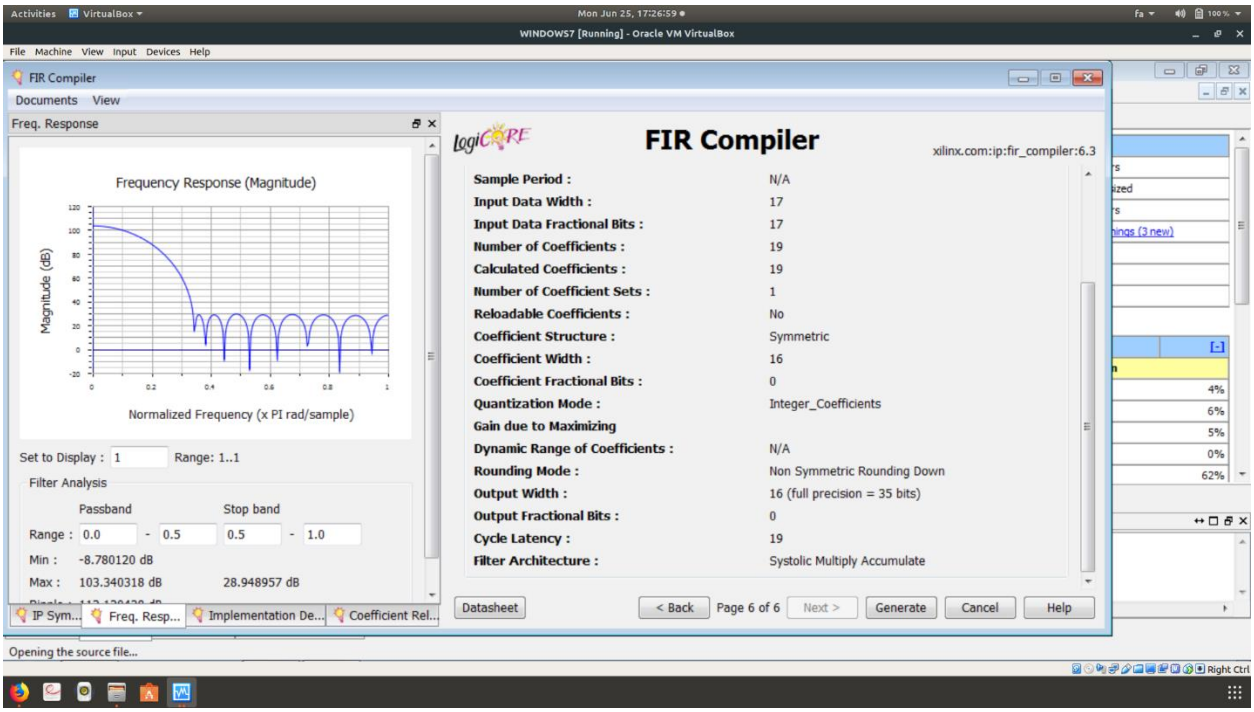
شکل ورودی و خروجی‌ها:

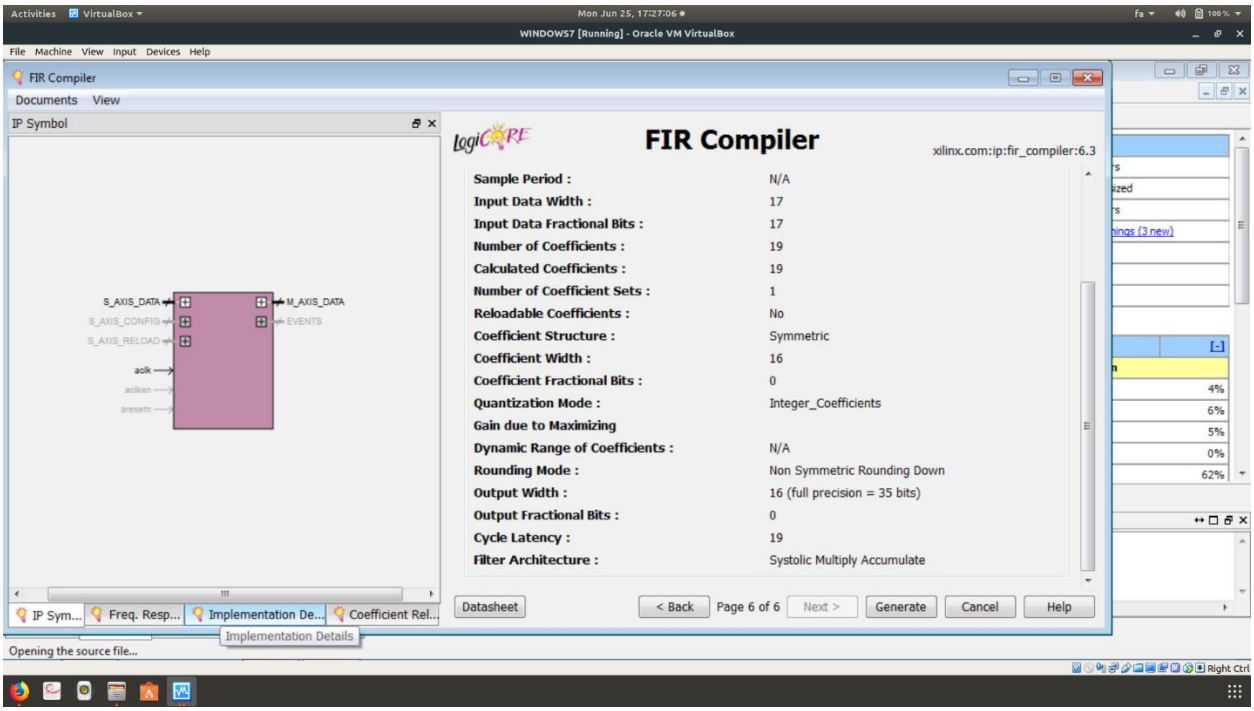
Name	Direction	Optional	Description
s_axis_data_tready	Output	no	TREADY for input DATA channel. Asserted by core to indicate core is ready to accept data.
s_axis_data_tdata	Input	no	TDATA for input DATA channel. Conveys the data stream to be filtered. See TDATA Structure for internal structure.

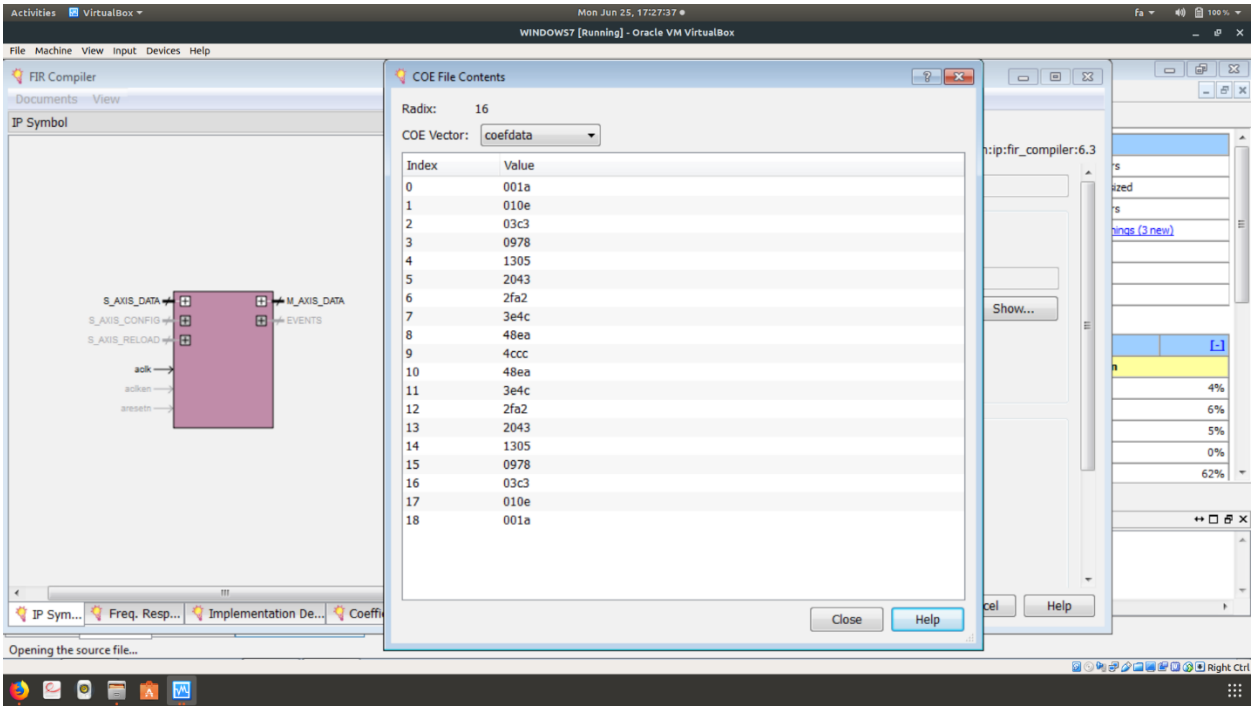
s_axis_data_tdata	Input	no	TDATA for input DATA channel. Conveys the data stream to be filtered. See TDATA Structure for internal structure.

m_axis_data_tvalid	Output	no	TVALID for output DATA channel. Asserted by core to indicate data is available for transfer.

طراحی فیلتر:

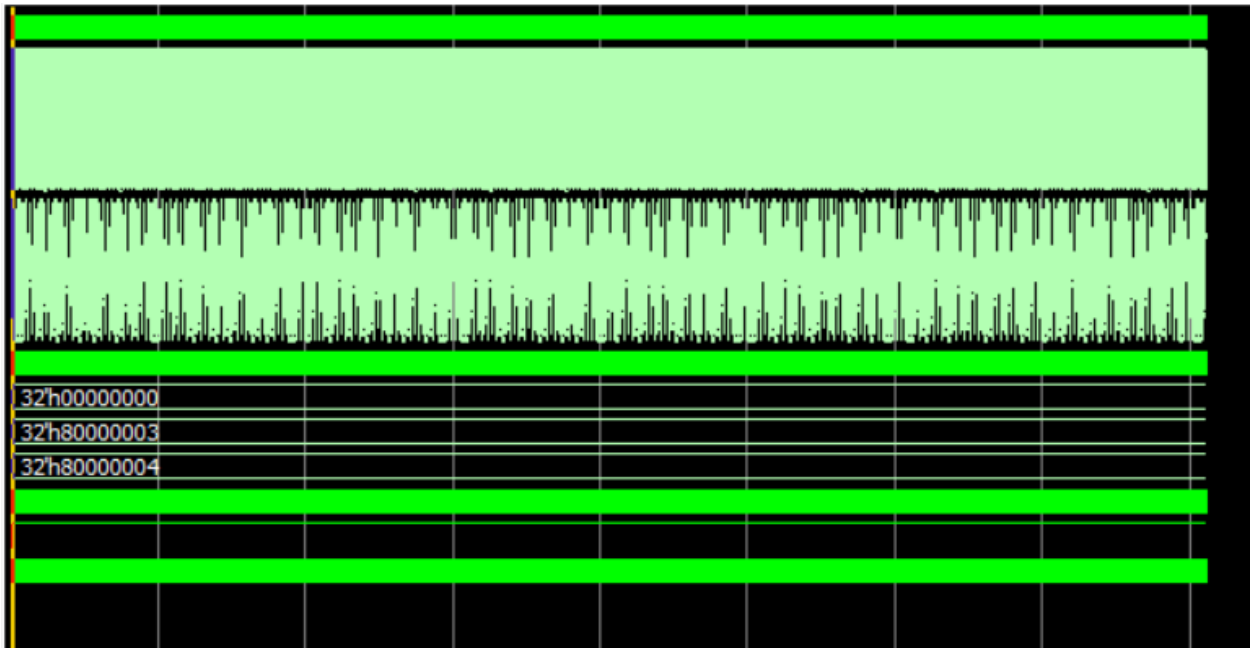

FIR Compiler
Documents View
Freq. Response
Frequency Response (Magnitude)
Magnitude (dB)
Normalized Frequency (x PI rad/sample)
Set to Display : 1
Range: 1..1
Filter Analysis
Passband
Stop band
Range : 0.0 - 0.5 0.5 - 1.0
Min : -8.780120 dB
Max : 103.340318 dB 28.948957 dB
IP Sym...
Freq. Resp...
Implementation De...
Coefficient Rel...
FIR Compiler
xilinx.com:ip:fir_compiler:6.3
Sample Period : N/A
Input Data Width : 17
Input Data Fractional Bits : 17
Number of Coefficients : 19
Calculated Coefficients : 19
Number of Coefficient Sets : 1
Reloadable Coefficients : No
Coefficient Structure : Symmetric
Coefficient Width : 16
Coefficient Fractional Bits : 0
Quantization Mode : Integer_Coefficients
Gain due to Maximizing
Dynamic Range of Coefficients : N/A
Rounding Mode : Non Symmetric Rounding Down
Output Width : 16 (full precision = 35 bits)
Output Fractional Bits : 0
Cycle Latency : 19
Filter Architecture : Systolic Multiply Accumulate
Datasheet
< Back
Page 6 of 6
Next >
Generate
Cancel
Help
Opening the source file...


FIR Compiler
Documents View
IP Symbol
S_AXIS_DATA
S_AXIS_CONFIG
S_AXIS_RELOAD
aclk
aclken
aresetn
M_AXIS_DATA
EVENTS
IP Sym...
Freq. Resp...
Implementation De...
Coefficient Rel...
Implementation Details
FIR Compiler
xilinx.com:ip:fir_compiler:6.3
Sample Period : N/A
Input Data Width : 17
Input Data Fractional Bits : 17
Number of Coefficients : 19
Calculated Coefficients : 19
Number of Coefficient Sets : 1
Reloadable Coefficients : No
Coefficient Structure : Symmetric
Coefficient Width : 16
Coefficient Fractional Bits : 0
Quantization Mode : Integer_Coefficients
Gain due to Maximizing
Dynamic Range of Coefficients : N/A
Rounding Mode : Non Symmetric Rounding Down
Output Width : 16 (full precision = 35 bits)
Output Fractional Bits : 0
Cycle Latency : 19
Filter Architecture : Systolic Multiply Accumulate
Datasheet
< Back
Page 6 of 6
Next >
Generate
Cancel
Help
Opening the source file...



خواندن دیتا از فایل، به صورت خط به خط و با دستور **fread** انجام می‌شود.

خروجی آنالوگ تست این ماژول، به شکل زیر می‌باشد:



- پیوست

تابع Rounding Output:

```
function [0:32] RoundOutput;
        input reg [0:32] din;
        RoundOutput = (din + 33'b0_0000_0000_0000_0001_0000_0000_0000_0000) ;
endfunction
```

این تابع، خروجی را با عدد ۲/۱ جمع می‌کند و سپس، بیت‌های کم‌ارزش را دور می‌ریزد. این روش، در Core آماده FIR Compiler، متناظر با Non-symmetric Rounding to Positive است.