علائم اختصاری و اصطلاحات فنی

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

فهرست مطالب

**No table of contents entries found.**

فهرست اشکال

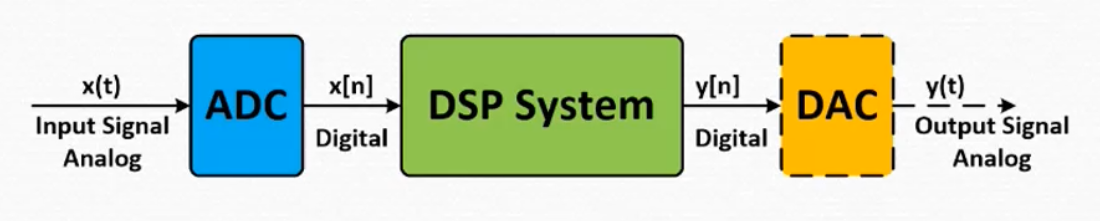
**No table of figures entries found.**

فهرست جداول

**No table of figures entries found.**

ما قصد نداریم الگوریتم های پردازش سیگنال را طراحی کنیم. بلکه قصد داریم این الگوریتم ها را بر روی FPGA پیاده سازی کنیم. طراحی این الگوریتم ها یک موضوع جداست که تخصص خاصی را می طلبد (معمولا فارغ التحصیلان مخابرات طراحان این الگوریتم ها هستند).

یک سیستم پردازش سیگنال :

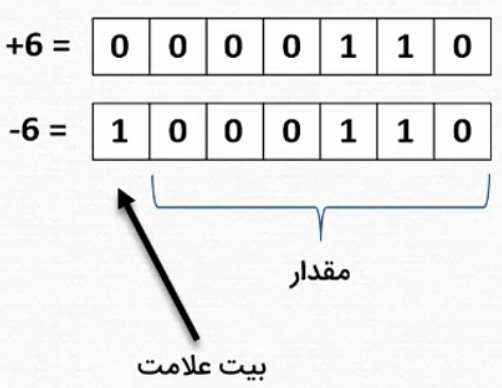


نحوه نمایش یک عدد منفی در دیجیتال:

برای این کار روش های مختلفی وجود دارد که عبارتند از:

* Sign & Magnitude (روش علامت و مقدار)

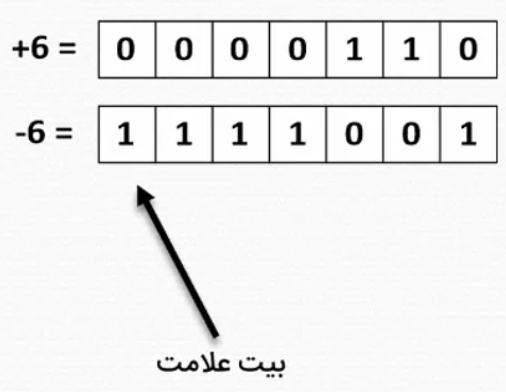
در این روش بیت پرارزش (MSB) تعیین کننده علامت عدد یعنی منفی یا مثبت می باشد. برای نمایش یک عدد منفی باید بیت MSB برابر با 1 باشد.



ایرادات این روش:

* پیاده سازی subtractor (تفریق کننده) مشکل می باشد. (نیاز به منابع زیادی برای این کار می باشد)
* دو مقدار برای عدد 0 وجود دارد.
* One’s complement (روش مکمل1)

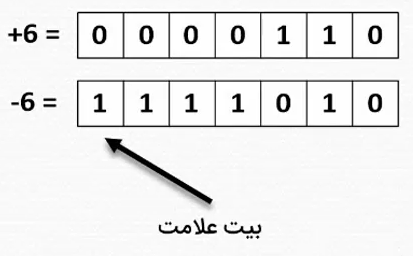
برای نمایش یک عدد منفی در این سیستم باید تمام بیت های مقدار مثبت آن عدد را قرینه کنیم.



در این روش برای پیاده سازی عمل تفریق می توان از همان مدار adder یا جمع کننده استفاده کرد ولی ایراد این روش آن است که در اینجا نیز برای مقدار 0 دو مقدار وجود دارد.

* Tow’s complement (روش مکمل1)

برای نمایش یک عدد منفی در این سیستم می توان مکمل1 آن عدد را محاسبه کرده و با یک 1 جکع کنیم. یکی روش دیگر نیز روش ذهنی است که در آن باید عدد مثبت را در نظر بگیریم و سپس تمام بیتهای 0 از سمت راست را بنویسیم، اولین 1 را نیز بنویسیم و مابقی بیت ها را قرینه کنیم.



در این روش علاوه بر اینکه پیاده سازی adder و subtractor با یک مدار امکان پذیر بوده و یک مقدار برای عدد صفر نیز وجود دارد، امکان جمع چندین عدد علامتدار که در بین راه سرریز داشته باشند وجود دارد. به عبارت دیگر این روش تضمین می کند که اگر ما بخواهیم تعدادی عدد با بیت مشخص را با هم جمع کنیم و بدانیم که نتیجه این جمع عددی خواهد شد که تعداد بیت های آن با تعداد بیت های این اعداد برابر است، در نتیجه سرریز های احتمالی در هنگام جمع کردن این اعداد با یکدیگر در نتیجه نهایی تاثیری نخواهد داشت.

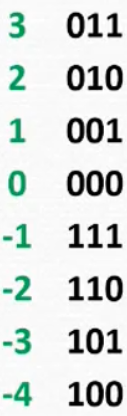
تمامی سیستم های دیجیتال از روش Two’s Complement برای نشان دادن اعداد علامت دار استفاده می کنند.

اگر تعداد بیتهای در نظر گرفته شده برای نمایش یک عدد را N در نظر بگیریم، رنج اعداد علامتداری که می توان توسط این تعداد بیت نمایش داد به شکل زیر خواهد بود:

یعنی اگر تعداد بیتهای در نظر گرفته شده برای نمایش عدد علامتدار را برابر با 3 بیت در نظر بگیریم با یان تعداد بیت می توان اعداد -4 تا +3 را نمایش داد.

در سیستم Two’s Complement اگر دو عدد n بیتی را با هم جمع کنیم و نتیجه در n بیت جا نشود، overflow (سرریز) رخ داده است.

به عنوان مثال، با 3 بیت می تاون اعداد زیر را نمایش داد:

اگر بخواهیم عدد 3 و 1 را باهم جمع کنیم، جواب برابر با 4 خواهد شد ولی نمی توان عدد +4 را توسط 3 بیت در سیستم Two’s Complement نمایش داد، لذا در صورت انجام این کار قطعا سرریز رخ خواهد داد و نتیجه که برابر با b”100” می باشد برابر با عدد -4 می باشد. در این حالت می گوییم wrapp around رخ داده است.

حال فرض کنیم می خواهیم جمع زیر را انجام دهیم:

2 + 3 - 4 + 1 - 3 = -1

باتوجه به اینکه می دانیم مقدار -1 در رنج مورد نظر ما می باشد به راحتی تمامی اعداد دیگر را با هم جمع کرده و اصلا به Overflow توجهی نمی کنیم.

فرض کنیم بخواهیم 2048 عدد 10 بیتی را با هم جمع کنیم و ندانیم مقدار نهایی چه عددی می شود بنابراین باید تعداد بیتهای در نظر گرفته شده برای عدد نهایی برابر با 10+11=21 بیت باشد. اما اگر بدانیم نتیجه نهایی این جمع نیز می تواند توسط یک عدد 10بیتی نمایش داده شود می توان تمامی این اعداد را به راحتی با هم جمع کرد و نگران OverFlow نبود و با این کار در مصرف منابع صرفه جویی بسیاری انجام دهیم.

طبق گفته آقای ثقفی اگر قصد دارید روی یک سیگنال محاسبات ریاضی انجام دهیم، باید نوع آن را Signed و در غیر این صورت Unsigned تعریف کنیم. حتی Port ها را هم باید از همین دو نوع استفاده کرد ولی از دید من Port ها را فقط از نوع Std\_Logic یا Std\_Logic\_Vector تعریف کرد و سیگنال ها را از نوع Signed و UInsigned چرا که با این کار استفاده از IP ها و Port Map کردن یک ماژول و ساخت Test Bench نیز راحت تر می شود. هنوز بر سر استفاده از std\_logic یا ustd\_logic به نتیجه نهایی نرسیدم.

نمایش اعداد اعشاری:

برای نمایش اعداد اعشاری می توان از دوروش Fixed Point یا Floating Point استفاده کرد.

در روش Floating Point محل قرار گیری اعشار ثابت نیست و تغییر می کند.

از معایب آن می توان موارد زیر را نام برد:

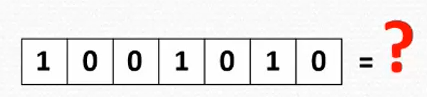
* پیاده سازی آن سخت است
* منابع مصرفی آن بالا است
* محاسبات آن زمان بر می باشد

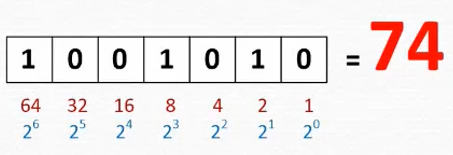
اما دلیل اینکه برای پیاده سازی DSP در FPGA از نمایش Fixed Point استفاده می کنیم موارد زیر را می توان نام برد:

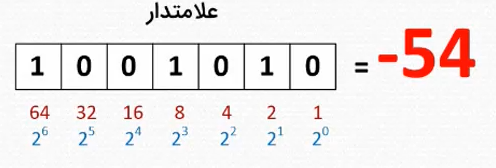
* مدار پیاده شده به این روش سرعت بسیار بالایی دارد
* منابع مصرفی آن نسبت به Floating Point بسیار کمتر است

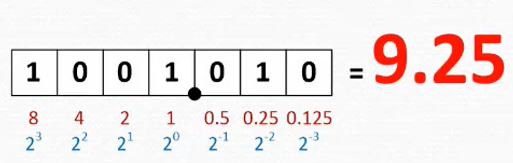
اگر بخواهیم از روش Floating Point در کار خود استفاده کنیم، باید تمامی اعداد خود را به صورت 32بیتی یا 64 بیتی در نظر بگیریم ولی در محاسبات به روش Fixed Point این مشکل را نخواهیم داشت.

در سیستم نمایش اعداد اعشاری به روش Fixed Point هیچ استانداردی وجود ندارد و استاندارد نمایش در واقع توسط طراح FPGA تعیین می شود. به عنوان مثال در هر بخشی از طرح طراح FPGA می توان برداشت متفاوتی از مقدار عدد 7بیتی زیر را داشته باشد:









پس مفهوم یک عدد باینری در این روش به تفسیری که از آن می شود بستگی دارد. یعنی Decimal Point در دیجیتال تحقق فیزیکی ندارد (توسط مثلا یک Flip Flop نمی توان آن را پیاده سازی کرد) و محل آ در ذهن طراح می باشد.

اگر یک عدد اعشاری را در نظر بگیریم، به قسمت صحیح و اعشاری آن در زبان انگلیسی اسامی زیر را می دهند.

Decimal Part

Fractional Part

Decimal Point

Binary Point

Radix Point

Integer Part

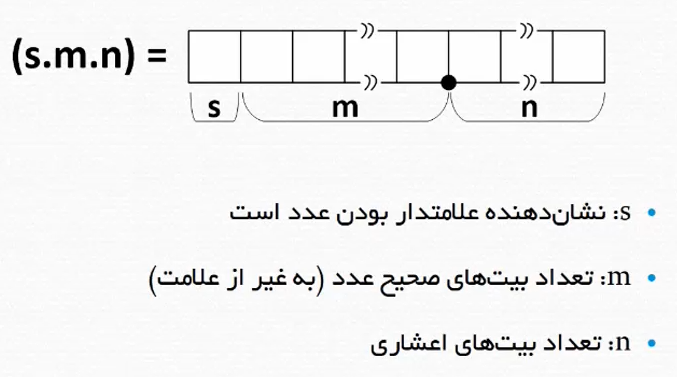
Whole Number

123.456

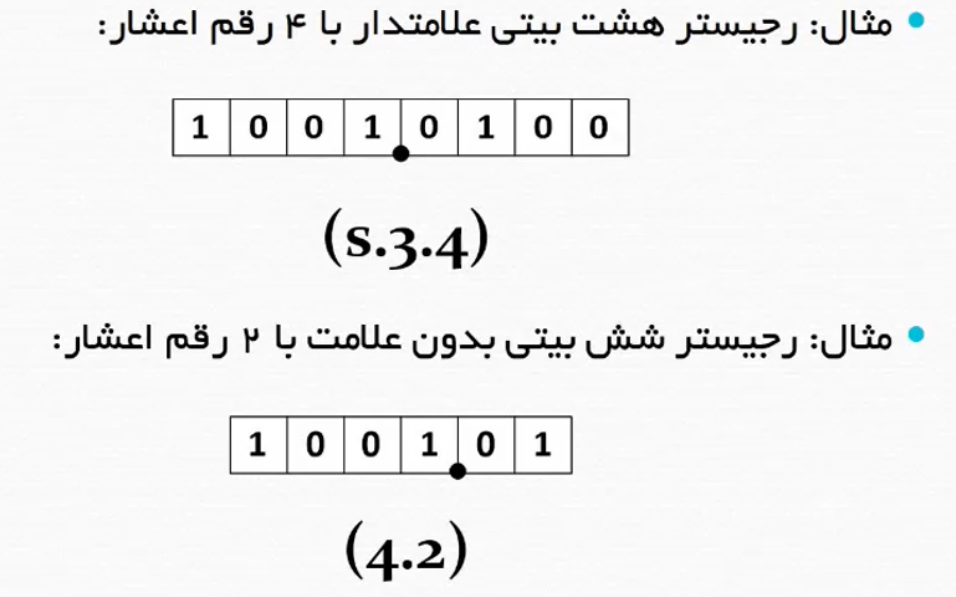
آنالیز مدل Fixed Point به روش s.m.n

ما برای پیاده سازی الگوریتمهای پردازش سیگنال به آن همه دقتی که Floating Point ارائه می دهد نیازی نداریم و به جای آن از مدل Fixed Point با عرض بیت کمتر استفاده می کنیم.

برای این منظور در سیستم خود به ازای هر سیگنال یا هر عملیات ریاضی، حداقل مقدار عرض بیت مورد نیاز را پیدا می کنیم. اما این محاسبات با یکدیگر در ارتباط هستند و جهت اطمینان از انجام صحیح محاسبات مورد استفاده در طرح خود از ابزاری به نام s.m.n استفاده میکنیم که به ما کمک می کند قبل از نوشتن کد FPGA، محاسبات اعشاری مورد نیاز برای طرحی که اجزای آن دارای عرض بیت های مختلف بوده و محل نقطه اعشار نیز برای هر کدام از این اجزا توسط ما تعیین شده، به درستی انجام بپذیرد.



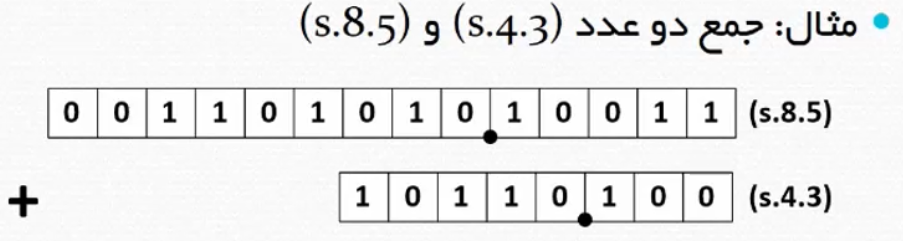
وجود s به معنی علامت داربودن عدد و عدم وجود آن به معنی بدون علامت بودن عدد می باشد.



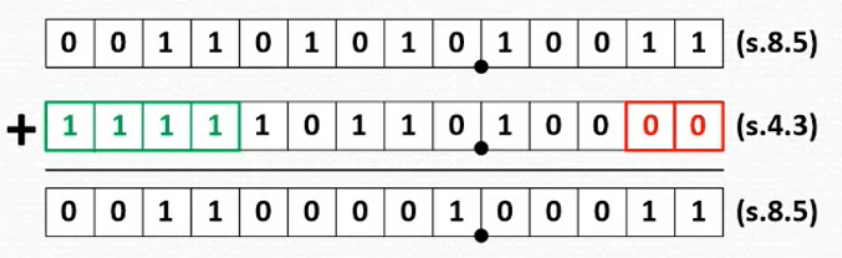
درست است که گفته شد نقطه اعشار در ذهن طراح می باشد ولی اثر خود را در پیاده سازی طرح نشان می دهد لذا باید با قوانین زیر آشنا باشیم.

* **جمع Fixed Point**

دو عدد باید طوری scale شوند که نقاط اعشاری هر دو عدد در زیر هم قرار بگیرند. به عبارت دیگر باید هر دو عدد دارای n یکسان باشند.



برای این کار یا می توان به سمت راست عدد دوم دو رقم 0 اضافه کرد یا از سمت راست عدد اول دو رقم را حذف کرد، که ما در اینجا روش اول را انجام می دهیم (اما در عمل از روش دوم یعنی کم کردن بیت استفاده می کنیم).

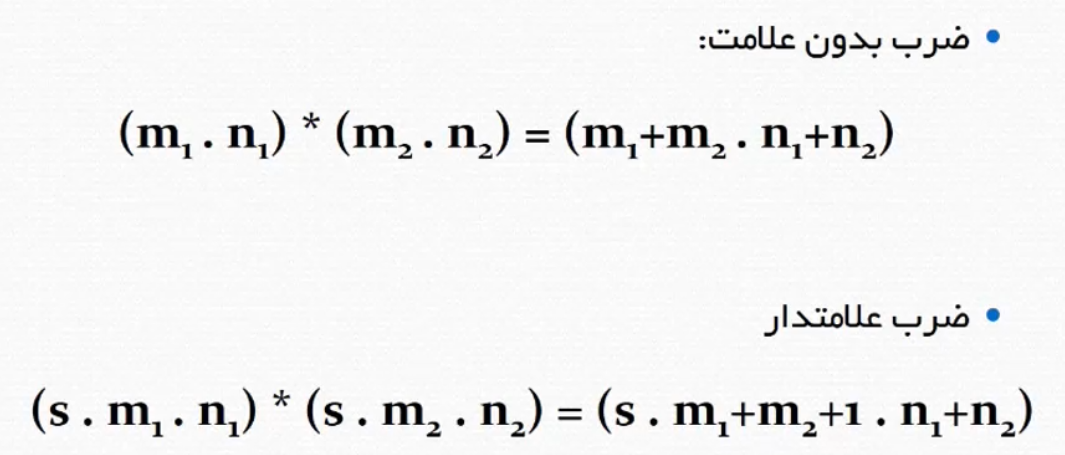


درست است که محل نقطه اعشاری در ذهن ما بود ولی اضافه کردن این دو صفر باید در کد نوشته شده در FPGA (یعنی به صورت سخت افزاری) انجام شود. این کار در VHDL توسط عملگر & یا concatenation انجام می شود.

قسمت سبز رنگ یعنی اضافه کردن و تکرار بیت علامت (در اینجا 1) به سمت چپ عدد دوم توسط Synthesizer انجام می شود و ما کاری در مورد آن انجام نمی دهیم. به این عمل Sign Extension گفته می شود یعنی تکرا بیت علامت.

* **ضرب Fixed Point**

در زبان VHDL برای ضرب دو سیگنال در یکدیگر، bitwise در نظر گرفته شده برای حاصلضرب برابر است با مجموع عرض بیت هر کدام از سیگنال ها.

****

این 1 اضافه شده در اینجا به خاطر وجود s در هر دو عدد می باشد چرا که عدد علامت دار حاصل یک بیت علامت بیشتر نمی تواند داشته باشد، مقدار این بیت قطعا با بیت علامت حاصلضرب یکسان می باشد. یعنی هر دو 0 و یا هر دو 1 می باشند.

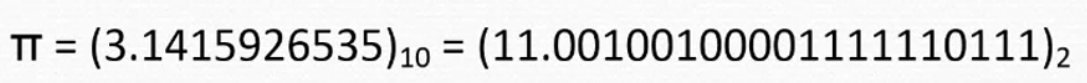
**نمایش یک عدد ثابت (کوانتیزاسیون)**

فرض کنیم بخواهیم عدد π که یک مقدار ثابت است را در درون FPGA پیاده سازی کنیم. اما بر اساس محاسبات خود به این نتیجه رسیده ایم که باید این عدد را با 20 بیت برای اعشار آن تقریب بزنیم.

عدد π برابر است با 3.1415926535897932384626433832795 برای آنکه آن را با 20 بیت برای اعشار آن تقریب بزنیم آن را در ضرب کرده و سپس روند می­کنیم.

حال عدد حاصل یعنی 3294199 را در مبنای 2 می نویسیم. نتیجه برابر خواهد بود با 1100100100001111110111

دو بیت سمت چپ قسمت صحیح و دو بیت سمت راست قسمت اعشاری را مشخص می کنند. عدد π تخمین زده شده با این بیت ها برابر با مقدار زیر خواهد بود.



این بار فرض کنیم که بر اساس محاسبات خود به این نتیجه رسیده ایم که باید این عدد را با 6 بیت برای اعشار آن تقریب بزنیم. برای این کار بایدآن را در ضرب کرده و سپس روند کنیم.

حال عدد حاصل یعنی 201 را در مبنای 2 می نویسیم. نتیجه برابر خواهد بود با 11001001

دو بیت سمت چپ قسمت صحیح و دو بیت سمت راست قسمت اعشاری را مشخص می کنند. عدد π تخمین زده شده با این بیت ها برابر با مقدار زیر خواهد بود.



به مقادیر به دست آمده در دو حالت دقت کنید.

* مقدار واقعی عدد π : 3.1415926535897932384626433832795
* مقدار عدد π با 20 بیت برای اعشار: 3.1415926535
* مقدار عدد π با 6 بیت برای اعشار: 3.140625

نباید از سمت چپ عدد بیت کم کرد چرا که مقدار عدد به هم می ریزد ولی کم کردن بیت از سمت راست باعث کاهش دقت اعشاری عدد می شود.

**Quantization (کوانتیزاسیون)**: یعنی کاهش دقت یک عدد با کم کردن تعداد بیت های در نظر گرفته شده برای قسمت اعشاری.

**Quantization noise or Quantization error**: وقتی تعداد بیت های قسمت اعشاری یک عدد را کم می کنیم در مقدار آن عدد نسبت به مقدار واقعی مقداری خطا به وجود می آید به این خطا نویز کوانتیزاسیون یا خطای کوانتیزاسیون گفته می شود.

**پیاده سازی اولین الگوریتم:**

وقتی یک System Man یک الگوریتم را طراحی می کند، میمکن است آن را به چندین شکل در اختیاز ما قرار دهد:

* یک فرمول به ما می دهد شامل جمع و ضرب و . . .
* یک مدل از آن الگوریتم را در نرم افزار Matlab Simulink به ما می دهد
* یا یک فایل mfile یا حتی فایلی که با زبان c آن را نوشته است در اختیار ما قرار می دهد

فارغ از هر کدام از این موارد بهتر است با مدل ارائه شده را به یک فایل Matlab Simulink تبدیل کنیم یعنی یک مدل به صورت بلوک دیاگرام در نرم افزار Matlab داشته باشیم.

نکته مهم! : در Matlab Simulink برای تعیین Sample Time مربوط به بلوک ها، باید کلاک FPGA را در ننظر داشته باشیم، یعنی اگر کلاک FPGA برابر با 100Mhz باشد باید Sample Time در Matlab را نیز برابر با 100Mhz در نظر بگیریم تا شبیه سازی الگوریتم مورد نظر در Matlab با پیاده سازی آن در FPGA کاملا یکسان باشد.

در Matlab Simulink برای بلوک هایی که ورودی دارند، می توانیم مقدار Samle Time را برابر با -1 قرار دهیم، این یعنی اینکه Sample Time بلوک مورد نظر باید برابر با Sample Time ورودی آن باشد.

برای کوانتیزه کردن مدل Floating Point در نرم افزار Matlab، از بلوک Data Type Conversion استفاده می کنیم، البته برخی از بلوک ها این قابلی را در درون خود دارند.

**نمایش عدد Fixed Point در Matlab**

در نرم افزار Matlab نوع نمایش اعداد Fixed Point مقداری با روش s.m.n که گفته شد متفاوت می باشد. فرض کنیم یک عدد علامتدار به شکل زیر داریم:

111.10100011

از طریق روش s.m.n این عدد را به شکل (s.2.8) نمایش می دهیم.

اما در Matlab این عدد را به شکل (1.11.8) نمایش می دهیم.

1 به معنی علامتدار بودن عدد می باشد(و نه به این معنا که بیت علامت 1 است)

11 مجموع همه بیت های در نظر گرفته شده (قسمت صحیح و اعشاری و بیت علامت)

8 تعداد بیت های در نظر گرفته شده برای قسمت اعشاری

پس در زمان تحلیل بر روی کاغذ از روش s.m.n و در هنگام استفاده از Matlab از روش خود Matlab استفاده می کنیم.

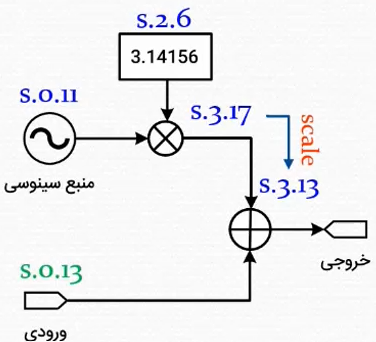
برای کوانتیزه کردن مدل Floating Point باید ابتدا ورودی های خارجی را کوانتیزه کنیم، سپس ورودی های داخلی مانند منابع سینوسی و ثابت داخل FPGA و در مرحله بعد کوانتیزه کردن عملگرها مانند جمع و ضرب و . . . .

عرض بیت ورودی های اصلی کاملا مشخص هستند و ما به عنوان طراح مقدار آنها را می دانیم، این ورودی ها می توانند خروجی های ADC یا . . . باشند. مثلا اگر ADC ما 14 بیتی باشد لذا ورودی اصلی خود را 14 بیتی در نظر می گیریم.

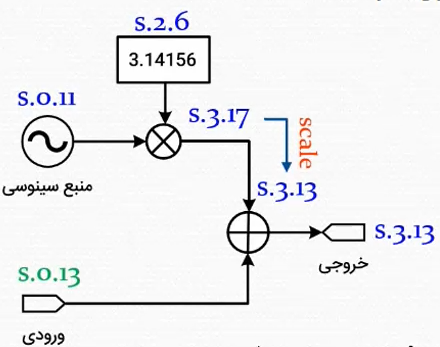


به علت اینکه این قسمت خارج از FPGA و در جهان بیرون می باشد نیازی به کوانتیزه کردن آن نداریم.

به شکل زیر دقت کنید:



همواره عمل ضرب باعث افزایش بخش صحیح و اعشاری عدد حاصلضرب می شود ولی معمولا عرض بیت های قسمت اعشاری که بیشتر از عرض بیت اعشاری ورودی اصلی باشند قابل خذف کردن می باشند و ارزش چندانی ندارند.



به خروجی دقت کنید. قانونا باید از آنالیز s.m.n، عدد خروجی را s.4.13 می گذاشتیم چون می دانیم جمع دو سیگنال در VHDL را باید در یک سیگنال با عرض بیتی به اندازه یک بیت بیشتر قرار داد چون ممکن است جمع دو عدد باعث به وجود آمدن یک بیت سرریز شود. اما ما در اینجا این کار را نکردیم چون در مرحله قبل که جمع کردن سیگنال های ورودی داخلی بود، یک بیت به قسمت صحیح بابت ضرب شدن دو عدد در یکدیگر اضافه کرده ایم که می دانیم همان تکرار عدد علامت است، پس دیگر در هنگام تعیین عرض بیت خروجی نیازی نیست که به واسطه جمع وروذی های داخلی و خارجی یک بیت اضافه برای قسمت صحیح در نظر بگیریم.

پس اگر در الگوریتم و طرح خود یک جمع کننده داشته باشیم که یکی یا هر دو ورودی آن حاصل یک ضرب علامتدار باشد، نیازی نیست برای کنترل سرریز یک بیت به عرض بیت های خروجی جمع کننده اضافه کنیم. چرا که این بیت قبلا در حاصلضرب اضافه شده است.

در نرم افزار Matlab هنگام کوانتیزه کردن تعیین کردن floor همان Truncation می باشد.

در نرم افزار Matlab برای اینکه ورودی و خروجی خود را در یک فایل بریزیم از بلوکی با نام To Workspace استفاده می کنیم.

**Scaling**: به طور کلی به تغییر محل نقطه اعشاری scaling گفته می شود.

این کار به دو شکل انجام می شود:



در عمده موارد ما از Scaling واقعی استفاده می کنیم. مثلا در هنگام Trauncate کردن یا . . .

ولی در برخی موارد می توان با استفاده از Scaling مجازی جادو کرد! مثلا فرض کنیم در بخشی از عملیات پیاده سازی نیاز داریم یک مقدار را در عدد 2 ضرب کنیم، برای این کار می توان از یک ضرب کننده استفاده کرد و در نتیجه مقداری Resource را مورد استفاده قرار داد و یا فقط در هنگام آنالیز طرح در ذهن خود این ضرب در 2 را در نظر بگیریم، یعنی مثلا هنگام آنالیز s.m.n یک بیت از n کم کرده و به m اضافه کنیم، بدون اینکه از هیچ سخت افزاری استفاده کنیم. پس توسط Scaling مجازی می توان جادو کرد یعنی اینکه ضرب و تقسیم بر اعدادی که مضرب 2 هستند را می توان به صورت ذهنی و نه به صورت سخت افزاری پیاده سازی کرد.

Scaling کجازی در واقع همون کاری هستش که ما در پروژه های خودمون هم انجام میدیم و یک سیگنال رو در یک مقداری ضرب می کنیم.

**7 گام پیاده سازی الگوریتم های پردازش سیگنال**

1. **بررسی مدل Floating Point در محیط Matlab و رفع ابهامات**

مدل اراده شده با ما می تواند در قالب های گوناگون مثل فایل C، فرمول یا ... باشد باید آن را به یک مدل Floating Point در متلب تبدیل کنیم و هماهنگی های لازم را با طراح system یعنی طراح الگوریتم انجام دهیم.

1. **کوانتیزاسیون مدل Floating Point و تهیه مدل Fixed Point یا طلایی**

یک کپی از مدل Floating Point ایجاد می کنیم و آن را مدل طلایی می نامیم

1. **بررسی اثرات خطای کوانتیزاسیون و محاسبه کمترین عرض بیت ها**

این مرحله، مهمترین مرحله کار ما می باشد. که در سه مرحله انجام می شود:

* کوانتیزاسیون ورودی های اصلی
* کوانتیزاسیون ورودی های داخلی
* کوانتیزاسیون عملگرها

1. **انجام شبیه سازی و ذخیره سازی ورودی و خروجی در فایل متنی**

این کار به کمک بلوک های To Work Space در محیط Matlab Simulink انجام می شود. بعد از سبیه سازی در Matlab ورودی های به مدل Fixed Point و خروجی های مدل را در یک فایل قرار می دهیم. علت قرار دادن ورودی به مدل Fixed Point در یک فایل متنی این است که قرار است از آن به عنوان ورودی به مدل VHDL استفاده کنیم. علت قرار دادن خروجی مدل Fixed Point در یک فایل متنی این است که قرار است در ادامه این خروجی را با خروجی مدل VHDL مقایسه کنیم.

1. **پیاده سازی الگوریتم به زبان VHDL بر مبنای نتایج مدل طلایی**
2. **شبیه سازی کد VHDL با اعمال همان ورودی های مدل طلایی**
3. **مقایسه خروجی مدل طلایی با خروجی شبیه سازی کد VHDL در Matlab**

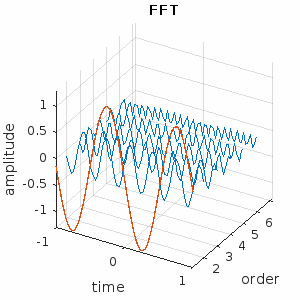
**Filter و FFT**

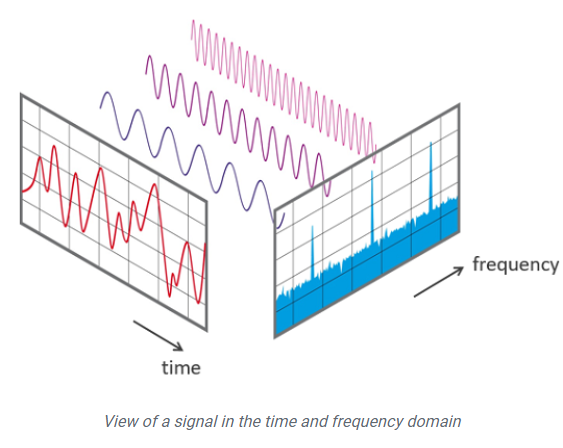
یک الگوریتم پردازش سیگنال در واقع یک مجموعه از محاسبات ریاضی است که از بلوک هایی تشکیل شده که به 3 بخش تقسیم می شوند:

* عملگرهای اصلی (جمع، ضرب، تقسیم، منها)
* توابع خاص (Sin، Cos، رادیکال و . . . )
* بلوکهای پردازشی پایه (filter، FFT و . . . )

**تبدیل فوریه:** هر سیگنالی را می توان با جمع مجموعه ای از سیگنال های Sin و Cos با فرکانس های مختلف ساخت. که به هر کدام از این سیگنال های Sin و Cos یک Component گفته می شود. تبدیل فوریه معمولا برای تبدیل یک سیگنال در طیف زمانی به سیگنالی در طیف فرکانسی مورد استفاده قرار می‌گیرد. FFT یک الگوریتم است که برای محاسبه[تبدیل فوریه گسسته](https://blog.faradars.org/%D8%AA%D8%A8%D8%AF%DB%8C%D9%84-%D9%81%D9%88%D8%B1%DB%8C%D9%87-%DA%AF%D8%B3%D8%B3%D8%AA%D9%87/) (Discrete Fourier Transform) یا DFT و نیز معکوس آن (IDFT) **با سرعت بالا** مورد استفاده قرار می‌گیرد.



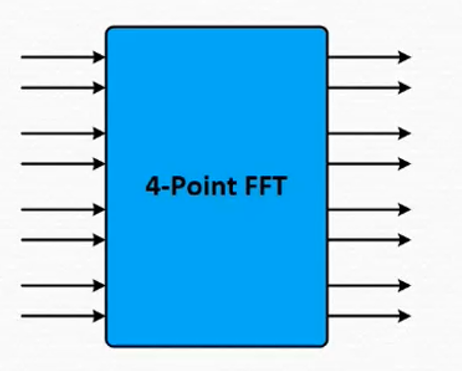




نمودار تبدیل فوریه به ما نشان می دهد که سیگنال اصلی ما از چه مولفه های فرکانسی (موج های سینوسی با چه فرکانسی) تشکیل شده است و نقش هر کدام از این مولفه ها در تشکیل آن سیگنال چقدر می باشد (در واقع دامنه هر کدام از این مولفه ها چقدر است)

FFT علاوه بر اینکه عملیات تبدیل فوریه را با سرعت بالایی انجام میدهد، از نظر پیاده سازی در FPGA و مصرف منابع نیز بسیار بهینه و مناسب می باشد.

ورودی FFT یک سیگنال Descreat یا گسسته می باشد و نه یک سیگنال آنالوگ پیوسته. یکی از پارامترهای FFT تعداد نقاط (Point) آن می باشد. این پارامتر مشخص کننده تعداد ورودی هایی است که باید به FFT داد. فرض کنیم از یک سیگنال توسط یک ADC با فرکانس 1Khz، تعداد 1000 نمونه در یک ثانیه برداشته ایم. این 1000 نقطه را می توان به یک FFT 1000 نقطه ای داد تا محاسبات لازم را بر روی آنها انجام داده و فرکانس های سینوسی سازنده سیگنال اصلی را به ما بدهد. خروجی FFT برابر با همان تعداد نقاط ورودی است یعنی اگر یک FFT، 100 نقطه ای باشد و 100 ورودی بگیرد، تعداد 100 فرکانس که بیان کننده فرکانس سیگال های تشکیل دهنده سیگنال اصلی هستند را به همراه دامنه آنها به ما می دهد.

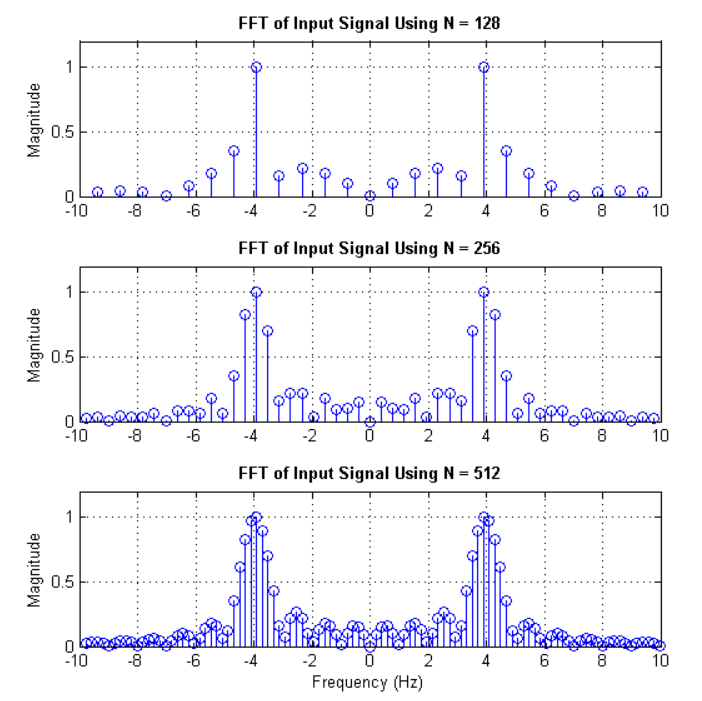


همانطور که در شکل بالا مشاهده می شود، با وجود 4 نقطه ای بودن FFT تعداد ورودی ها و خروجی های آن 8 عدد می باشد, علت آن این است که الگوریتم FFT یک الگوریتم مختلط می باشد لذا ورودی و خروجی های آن نیز اعداد مختلط می باشند و به ازای هر نقطه باید یک عدد مختلط که دارای یک بخش حقیقی و یک بخش موهومی می باشد را به آن بدهیم و یک عدد مختلط نیز دریافت کنیم.

یکی دیگر از پارامترهای FFT، فرکانس نمونه برداری(Fs) می باشد، چرا که FFT باید بداند نقاط ورودی به آن با چه فرکانسی نمونه برداری شده اند (فرکانس ADC) که بر اساس آن بتواند محاسبات خود را انجام دهد. فرکانس مولفه هایی که FFT در خروجی میدهد، حداکثر برابر با Fs می باشد. به عنوان مثال فرض کنیم از یک سیگنال توسط یک ADC با فرکانس 1Mhz نمونه برداری شده است و ما 8 عدد از این نمونه ها را به FFT می دهیم. FFT، تعداد 8 عدد سیگنال با فرکانس های از 0 تا 1Mhz را که تشکیل دهنده سیگنال اصلی هستند به ما می دهد.

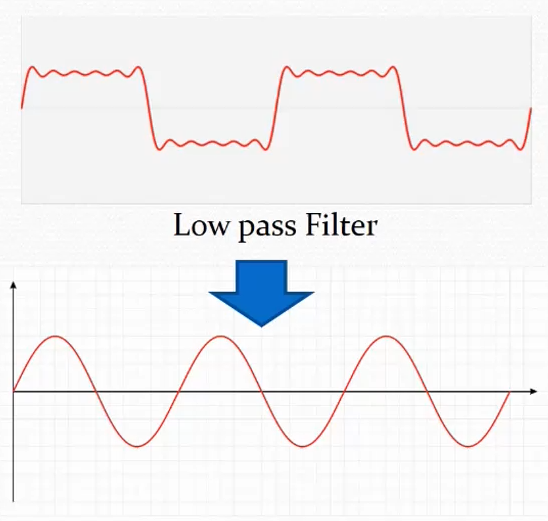
یکی دیگر از پارامترهای FFT، Resolution آن می باشد که برابر است با تقسیم فرکانس نمونه برداری(Fs) بر تعداد نقاط نمونه برداری.

مشخص است که هر چه تعداد نقاط نمونه برداری بیشتر باشد، رزولوشن FFT کمتر شده و دقت آن بالاتر می رود یعنی FFT دقیقتر می تواند به ما بگوید که مولفه های فرکانسی تشکیل دهنده یک سیگنال، کدام ها هستند. مثلا اگر یک مولفه با فرکانس 6.25 Mhz داشته باشیم و رزولوشن FFt برابر با 1Mhz باشد، FFT نمی تواند آن مولفه را دقیقا نشان دهد چون دقت بینهایت ندارد و فقط مولفه های با فرکانس 6Mhz و 7Mhz را نمایش می دهد.

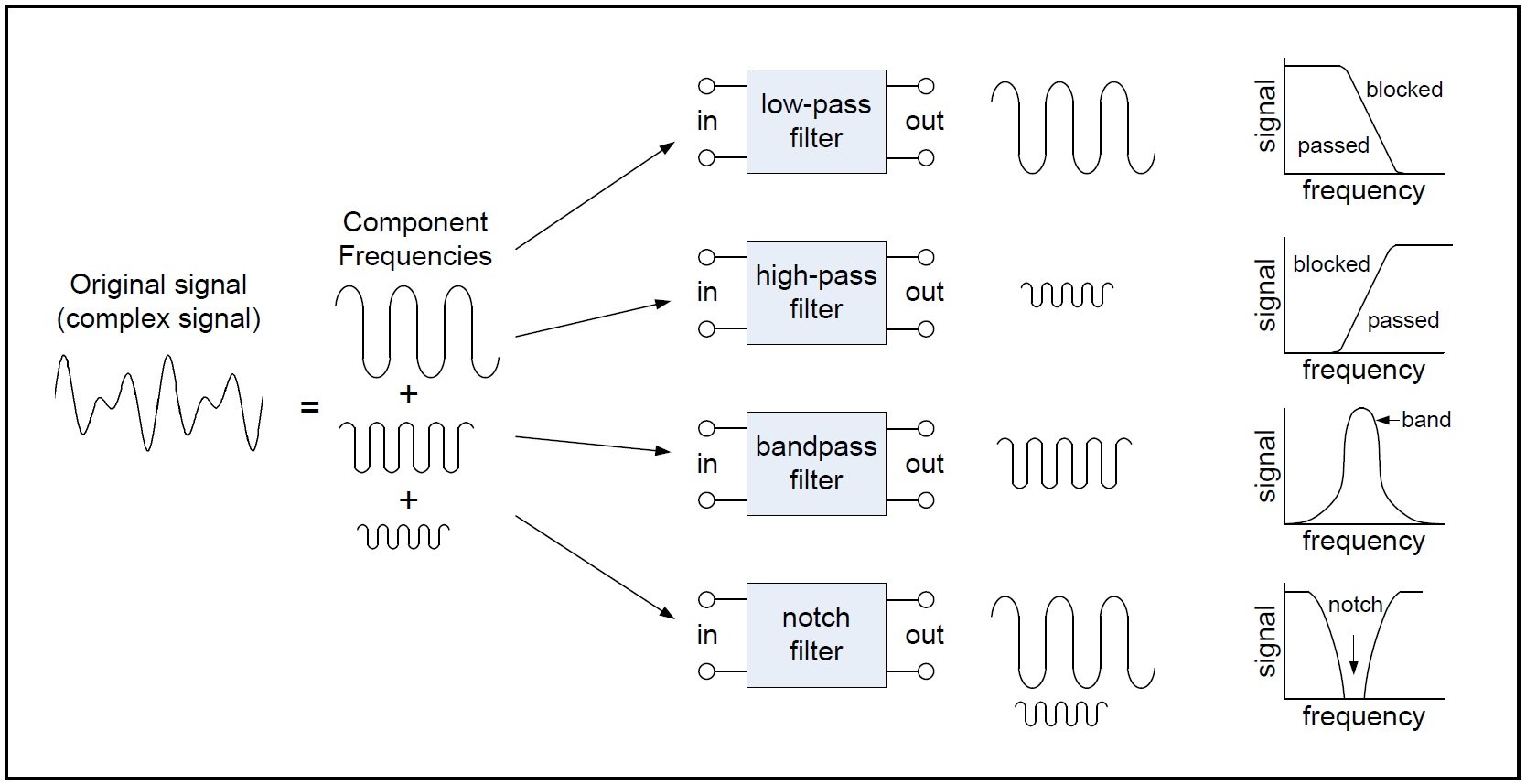


فیلتر: یک فیلتر وقتی یک سیگنال را در ورودی دریافت کند می تواند برخی از مولفه های فرکانسی آن سیگنال را (سیگنال های سینوسی که مجموع آنها سازنده سیگنال اصلی می باشد) از خود عبور ندهد.

به عنوان مثال یک فیلتر پایین گذر:



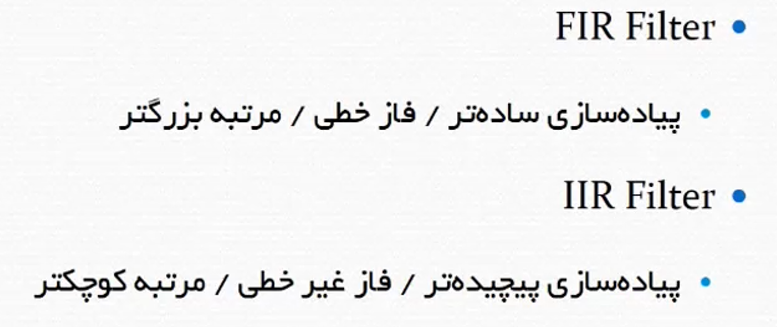
یک شکل عالی برای نشان دادن انواع فیلتر:



**فیلترهای دیجیتال:**

فیلترهای دیجیتال در هر الگوریتم پردازش سیگتل دیجیتالی وجود دارند. لذا بسیار حائز اهمیت می باشند. به طور کلی فیلترهای دیجیتال به دو ساختار گروه اصلی تقسیم می شوند:

* FIR (Finite Impulse Response)
* IIR (Infinite Impulse Response)



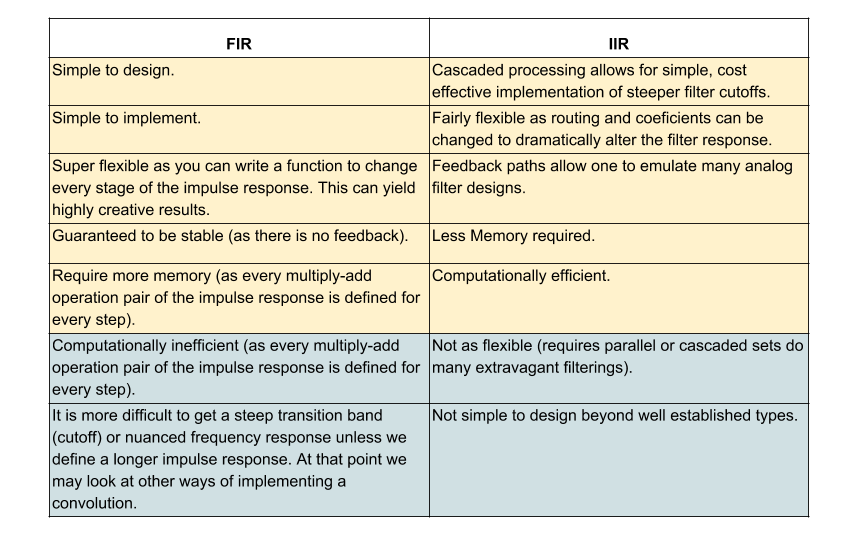
منظور از فاز خطی و غیر خطی چیست؟

وقتی یک سیگنال از یک فیلتر عبور می کند، علاوه بر حذف برخی مولفه های فرکانسی سک اتفاق دیگر نیز به واسطه ذات فیتر بر روی سیگنال خروجی ظاهر می شود و آن اختلاف فاز سیگنال ورودی و سیگنال خروجی از فیلتر است. این اختلاف فاز ایجاد شده بستگی به فرکانس سیگنال ورودی دارد.

اگر اختلاف فاز بین سیگنال ورودی و خروجی یک فیلتر به ازای سیگنال های ورودی با فرکانس های مختلف را در یک نمودار رسم کنیم، و شکل ایجاد شده یک خط باشد می گوییم فیلتر فاز خطی دارد و در غیر این صورت فاز فیلتر غیرخطی می باشد:



اگر شکل بالا اختلاف فاز بین ورودی و خروجی یک فیلتر را بر حسب فرکانس سیگنال ورودی نشان دهد، فیلتر مورد نظر دارای فاز خطی بوده است یعنی یک فیلتر FIR بوده است. در فیلتر های با فاز خظی می توان با استفاده از روشهایی، منابع مصرفی در FPGA جهت پیاده سازی را کاهش داد.



هر چقدر مرتبه (Order) یک فیلتر بالاتر باشد، منابع بیشتری برای پیاده سازی آن در FPGA مورد نیاز است.

**ساختارهای محبوب فیلتر FIR برای پیاده سازی:**

* Direct Form I

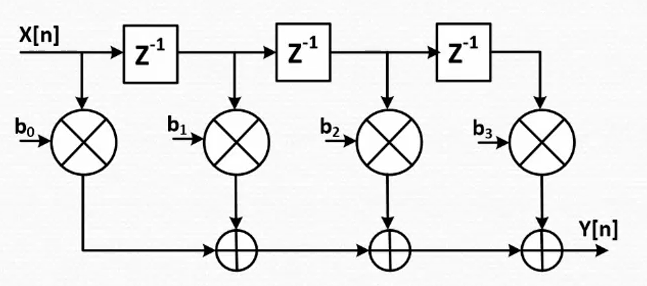


Figure 1: نمایش Flow Graph مربوط به ساختار Direct Form

Filter Coefficients (ظرایب فیلتر): وقتی یک طراح سیستم، فیلتری را طراحی کند و از ما بخواهد آن را در FPGA پیاده سازی کنیم تنها کافی است که ضرایب آن فیلتر و نوع آن را به ما بگوید. مثلا بگوید یک فیلتر FIR از نوع Direct Form با این ضرایب را پیاده سازی کنید.

Filter Order (مرتبه یا درجه فیلتر): مرتبه فیلتر بالا برابر با 3 می باشد.

Transfer Function یا تابع انتقال فیلتر بالا برابر خواهد بود با:

* Transposed Structure

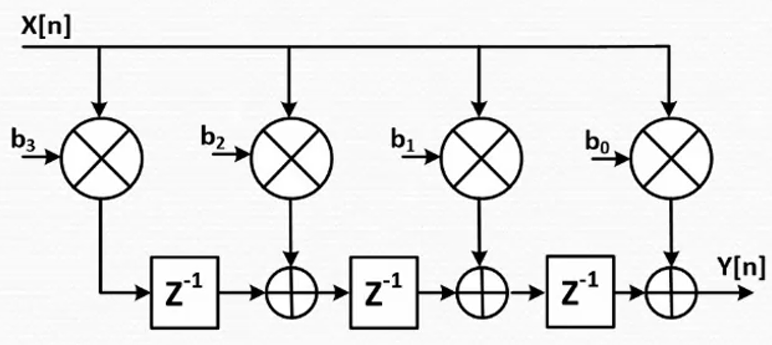


Figure 2: نمایش Flow Graph مربوط به ساختار Transposed Structure

نسبت به ساختار Direct Form، این ساختار برای پیاده سازی مناسب تر است.

چرا؟

برای اینکه در ساختار direct Form در هر کلاک یک خروجی (y[n]) تولید شود، به سه عدد جمع و 4 عدد ضرب نیاز داریم:

اما در ساختار Transposed فقط به یک ضرب و یک جمع نیاز داریم. چرا؟ چون هر کدام از در واقع یک رجیستر می باشند که محتوای یک محاسبه ای در آنها ریخته می شود.



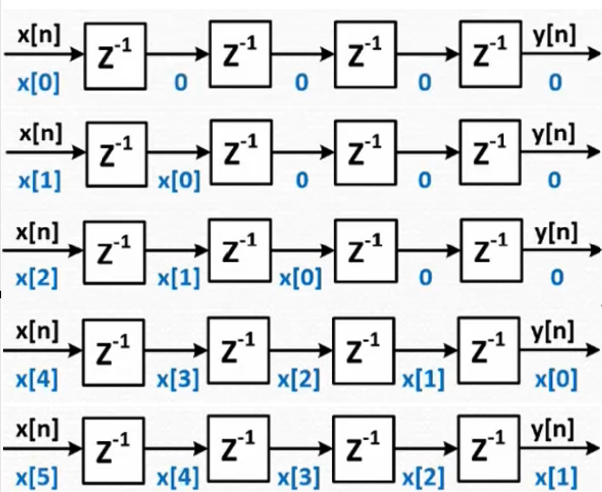
پس در این ساختار مقدار y[n] طبق فرمول زیر محاسبه می شود:

اصلا مهم نیست که محتوای رجیستر α در چند کلاک آماده شده است و درست است که برای محاسبه سایر رجیسترها هم در همان کلاک محاسباتی در حال انجام است، مهم این است که در لحظه محاسبه خروجی در هر کلاک مقدار آن مشخص می باشد. در واقع از هر رجیستر تا رجیستر دیگری فقط به یک جمع و یک ضرب نیاز داریم. پس ساختار Transposed سرعت بالاتری نسبت به Direct Form دارد.

دو ساختار Direct Form و Transposed در صورت یکسان بودن ضرایب هیچ تفاوتی در پاسخ نخواهند داشت، یعنی طراح سیستم برایش اهمیتی ندارد که ما طبق کدام ساختار فیلتر را پیاده سازی می کنیم، چرا که خروجی هر دو فیلتر یکسان خواهد بود ولی استفاده از Transposed کمک می کند که سرعت اجرای الگوریتم بالاتر باشد.

**مفهوم Delay Line**

مشاهده می شود که در هر کلاک، مقدار ورودی در رجیستر بعدی قرار میگیرد و بعد از زمان مشخصی به خروجی می رسد.



کلاک چهارم

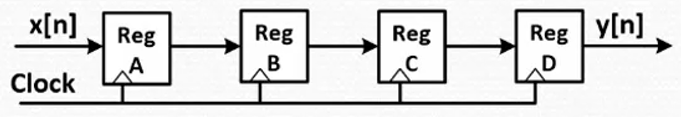
کلاک پنجم

کلاک سوم

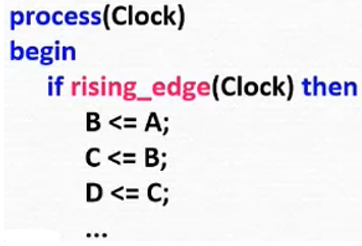
کلاک دوم

کلاک اول

مدار Delay Line در سخت افزار در واقع به صورت Flip Flop ها ساخته می شود:



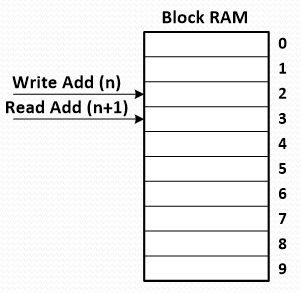
کد توصیف سخت افزار در زبان VHDL برای پیاده سازی این delay Line به شکل زیر نوشته می شود:



Delay Line را می توان علاوه بر رجیسترها به کمک حافظه بلوکی نیز پیاده سازی کرد.

اگر Delay Line مورد نظر جهت پیاده سازی یک Delay Line بزرگ باشد (مثلا یک تاخیر با طول 100 Clock) و به مقدار های میانی آن نیازی نداشته باشیم(فقط به مقادیر ورودی و خروجی نیاز داشته باشیم)، می توانیم از Block Memory برای ساخت Delay Line استفاده کنیم.

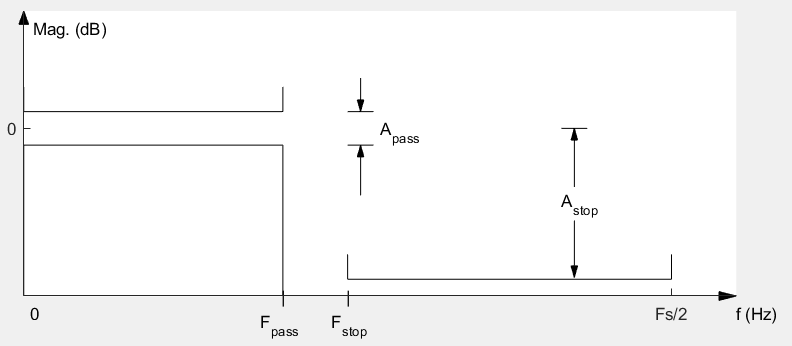
برای این کار یک Dual Port Ram را به شکل زیر مورد استفاده قرار می دهیم:



دو شمارنده را در کد خود تعریف می کنیم که همواره یکی از آنها یک واحد از دیگری بیشتر می باشد. از شمارنده کمتر برای نوشتن روی Block Ram و از شمارنده بیشتر برای خواندن از روی Block Ram استفاده می کنیم. در این صورت اگر تعداد سلول های Block Ram برابر با 100عدد باشد، همواره آنچه از Block Ram خوانده می شود دیتایی است که 100 Clock قبل روی Block Ram و در آن آدرس مورد نظر نوشته شده است.

ابزار fdatool در Matlab که جدیدا با نام Filter Designer ارائه می شود.

در ابزار Filter Designer تصویر زیر را می توان برای Low Pass Filter مشاهده کرد:



Fpass: سیگنال های با فرکانس کمتر از Fpass از فیلتر عبور می کنند. (مشخص کننده Pass Band)

Fstop: سیگنال های با فرکانس بیشتر از Fstop از فیلتر عبور نمی کنند و در واقه دامنه آنها به اندازه Astop کاهش پیدا می کند.(مشخص کننده Stop Band)

سیگنال های با فرکانس بین Fpass و Fstop از فیلتر عبور می کنند ولی دامنه آنها بسیار تضعیف می شود.

1. Fpass = passband frequency (in Hz)
2. Fstop = stopband frequency (in Hz)
3. Apass = max ripple allowed in passband (in dB)
4. Astop = min attenuation in stopband (in dB)

Apass: مقدار ریپل مجاز در ناحیه Pass Band (هر چه کمتر باشد فیلتر در ناحیه Pass Band به فیلتر ایده آل نزدیکتر می باشد)

Astop: مقدار تضعیف دامنه سیگنال در ناحیه Stop Band (هرچه بیشتر باشد فیلتر در ناحیه Stop Band به فیلتر ایده آل نزدیکتر می باشد)