علائم اختصاری و اصطلاحات فنی

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

فهرست مطالب

**No table of contents entries found.**

فهرست اشکال

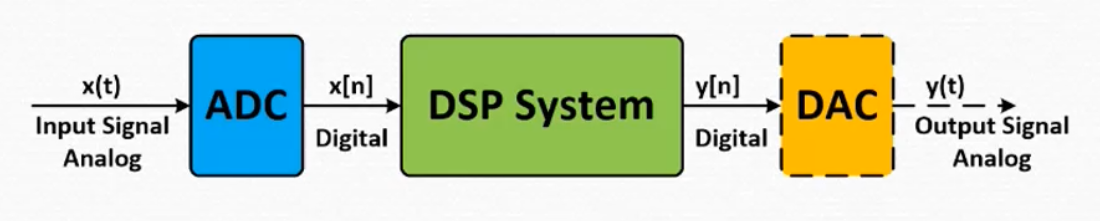
**No table of figures entries found.**

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 :

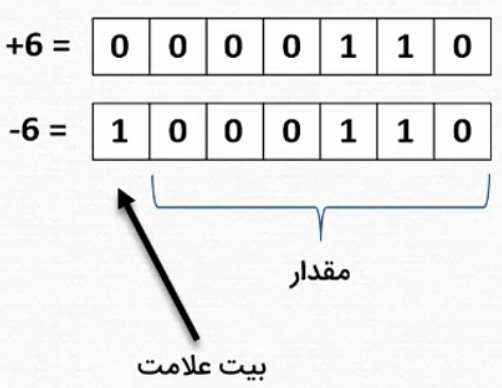


نحوه نمایش یک عدد منفی در دیجیتال:

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

* Sign & Magnitude (روش علامت و مقدار)

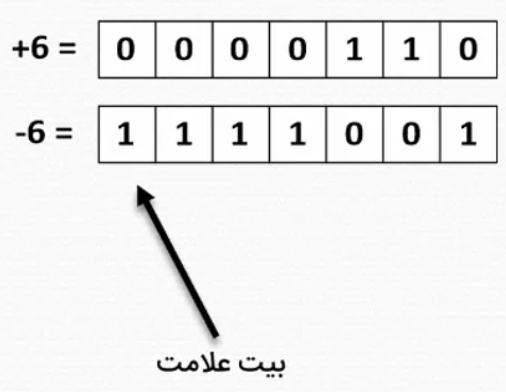
در این روش بیت پرارزش (MSB) تعیین کننده علامت عدد یعنی منفی یا مثبت می باشد. برای نمایش یک عدد منفی باید بیت MSB برابر با 1 باشد.



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)

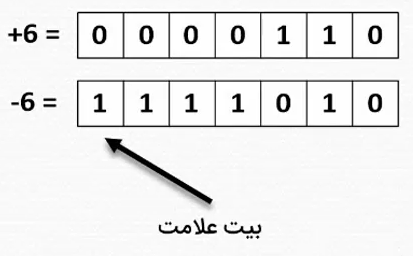
برای نمایش یک عدد منفی در این سیستم باید تمام بیت های مقدار مثبت آن عدد را قرینه کنیم.



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.

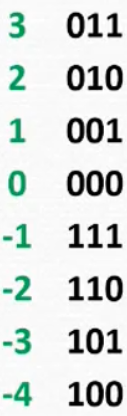
تمامی سیستم های دیجیتال از روش Two’s Complement برای نشان دادن اعداد علامت دار استفاده می کنند.

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:

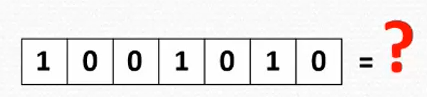
* پیاده سازی آن سخت است
* منابع مصرفی آن بالا است
* محاسبات آن زمان بر می باشد

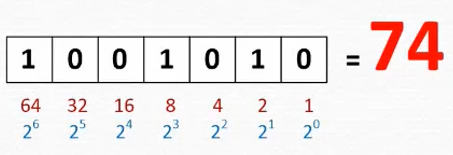
اما دلیل اینکه برای پیاده سازی DSP در FPGA از نمایش Fixed Point استفاده می کنیم موارد زیر را می توان نام برد:

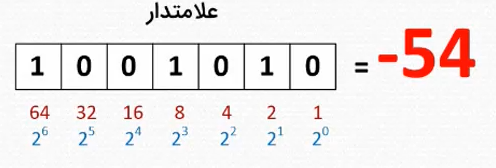
* مدار پیاده شده به این روش سرعت بسیار بالایی دارد
* منابع مصرفی آن نسبت به Floating Point بسیار کمتر است

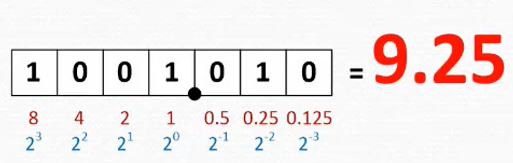
اگر بخواهیم از روش Floating Point در کار خود استفاده کنیم، باید تمامی اعداد خود را به صورت 32بیتی یا 64 بیتی در نظر بگیریم ولی در محاسبات به روش Fixed Point این مشکل را نخواهیم داشت.

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

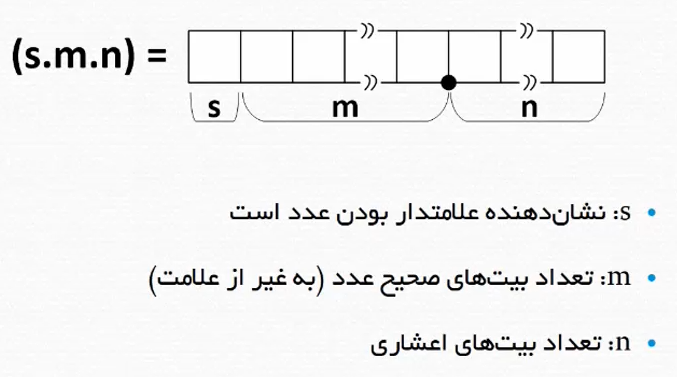
Whole Number

123.456

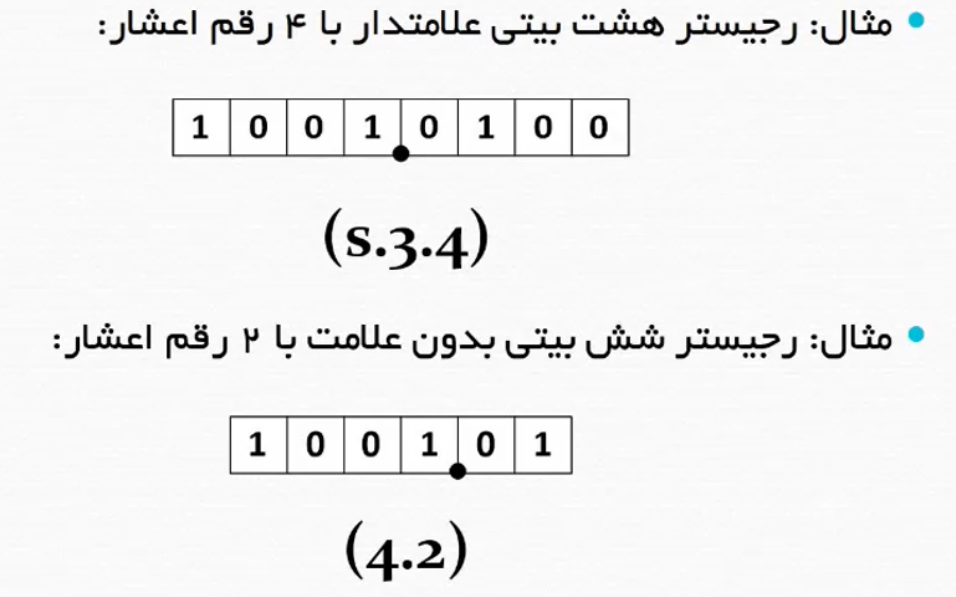
آنالیز مدل Fixed Point به روش s.m.n

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.



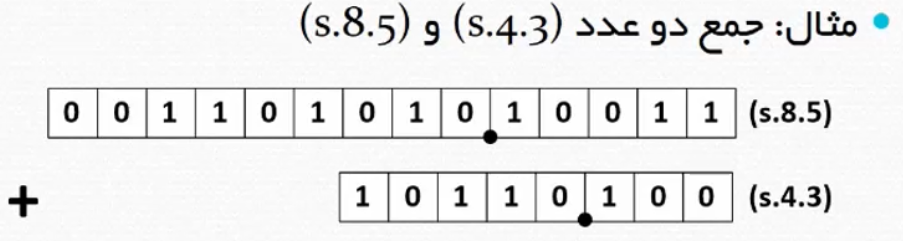
وجود s به معنی علامت داربودن عدد و عدم وجود آن به معنی بدون علامت بودن عدد می باشد.



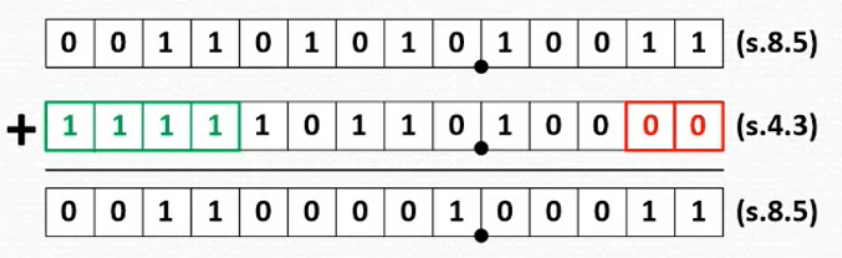
درست است که گفته شد نقطه اعشار در ذهن طراح می باشد ولی اثر خود را در پیاده سازی طرح نشان می دهد لذا باید با قوانین زیر آشنا باشیم.

* **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).

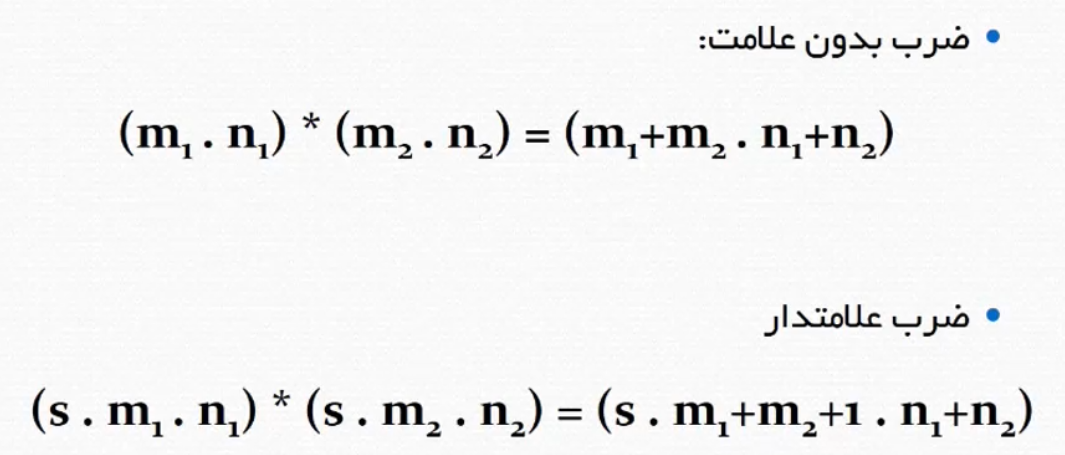


درست است که محل نقطه اعشاری در ذهن ما بود ولی اضافه کردن این دو صفر باید در کد نوشته شده در FPGA (یعنی به صورت سخت افزاری) انجام شود. این کار در VHDL توسط عملگر & یا concatenation انجام می شود.

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.

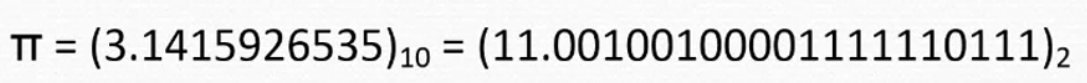
**نمایش یک عدد ثابت (کوانتیزاسیون)**

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 متلب 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 متلب را نیز برابر با 100Mhz در نظر بگیریم تا شبیه سازی الگوریتم مورد نظر در متلب 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.