WWW.MOHANDESYAR.COM

خود آموز زبان توصیف سخت افزار

VERILOG

ابراهیم جهاندار
دانشگاه اصفهان - دانشکده فنی مهندسی
گروه برق - الکترونیک

Ebrahim Jahandar
University of Isfahan - Faculty of Engineering
Electronic Department
www.jahandar.ir

مهندس یار

WWW.MOHANDESYAR.COM

فهرست مطالب

مقدمه ۳
۱) سبک طراحی ۴
۲) سطوح طراحی در زبان Verilog ۷
۳) متد طراحی از بالا به پایین ۸
۴) تعاریف اولیه ۱۲
۵) قواعد دستوری و پایه ای زبان Verilog ۱۳
۶) مدل سازی در سطح گیت ۲۶
۷) مدل سازی سطح ثباتی (جریان داده ها) ۳۲
۸) عبارات ، عملگر ها و عملوند ها ۳۴
۹) مدل سازی رفتاری در Verilog ۴۰
۱۰) طراحی مدار در سطح رفتاری ۴۵
۱۱) مثال ها ۴۸
۱۲) تمارین ۶۳

مهندس یار

WWW.MOHANDESYAR.COM

مقدمه

Verilog یکی از زبان های توصیف سخت افزار می باشد. توسط یک زبان توصیف سخت افزار می توان رفتار یک سیستم دیجیتال مانند فلیپ فلاپ ، حافظه و یا پردازنده و ... را توصیف نمود. با استفاده از این زبان توصیف سخت افزاری می توان یک سیستم دیجیتالی ساده مانند یک فلیپ فلاپ و یا یک سیستم دیجیتالی پیشرفته نظیر یک میکروکنترلر را در هر سطحی توصیف نمود. توسط این زبان شما می توانید سخت افزار خود را در سطوح طراحی زیر توصیف نمایید :

کتابچه ای که پیش رو دارید خود آموز زبان توصیف سخت افزار Verilog می باشد. مخاطبان خاص این کتابچه دانشجویان رشته های برق، کامپیوتر – سخت افزار ، مهندسی پزشکی – بیو الکتریک و دیگر افراد علاقه مند به مدارات دیجیتال می باشد. خوانندگان این کتاب باید دقت بفرمایند که برای فهم کامل مطالب درون این کتابچه می بایست آشنایی خوبی با مدارات دیجیتال داشته و یا درس مدارات منطقی را گذرانده باشند.

از ویژگی های این خود آموز می توان به مطرح شدن مثال ها و مسائل متنوع بعد از هر مطلب آموزشی اشاره کرد. همچنین سعی بر آن شده از مطالبی که کاربرد کمتری دارند صرف نظر شود. خوانندگان در صورت تمایل می توان جهت فراگیری مسائل پیشرفته تر در Verilog به کتاب Verilog HDL – A Guide to Digital Design مراجعه نمایند. مطالب این کتابچه از کتاب های Verilog HDL – A Guide to Digital Design نوشته Samir Palnitkar ، جزوه آموزشی Verilog نوشته سعید صفری ، جزوه آموزشی Verilog نوشته Deepak Kumar Tala و ... جمع آوری و تدوین شده است.

در پایان از تمامی خوانندگان عزیز تقاضا دارم که ایراد و پیشنهادات خود را به آدرس پست الکترونیک اینجانب به نشانی e {at} jahandar.ir ارسال نموده و به هرچه بهتر شدن آن کمک نمایند. همچنین نسخه های جدید این کتابچه را می توانید از ادرس زیر دریافت نمایید.

www.jahandar.ir/files/Verilog-Hardware-Description-Language.pdf

تمامی حقوق این کتابچه برای نویسنده آن محفوظ می باشد. نسخه الکترونیکی کتابچه فوق بصورت مجانی در اینترنت قرار داده شده است ، فروش و یا هرگونه درآمد زایی از طریق این کتابچه مجاز نمی باشد منوط به اجازه از موئلف می باشد.

ابراهیم جهاندار – آذر ۱۳۹۰

مهندس یار

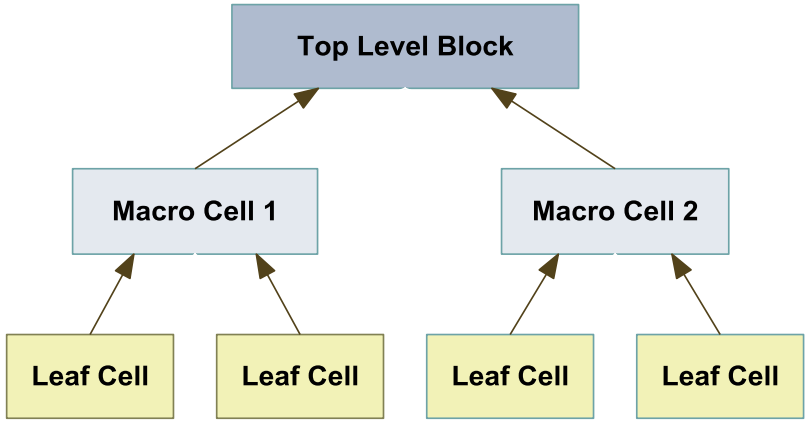
WWW.MOHANDESYAR.COM

۱) سبک طراحی

قبل از اینکه به جزئیات زبان Verilog به پردازیم، ابتدا نیاز است تا کمی درباره سبک طراحی توسط زبان های توصیف سخت افزار صحبت کنیم. زبان Verilog مانند دیگر زبان های توصیف سخت افزار به طراحان امکان طراحی از بالا به پایین و از پایین به بالا را می دهد، هریک از این متد ها دارای مزایا و معایبی می باشند که به آن اشاره می شود.

- **سبک طراحی از پایین به بالا[1]**

متد طراحی پایین به بالا یک متد سنتی در طراحی مدار های دیجیتال می باشد. در این متد سخت افزار دیجیتال از پایین ترین سطح یعنی سطح گیت طراحی می شود و در هر مرحله تا رسیدن به سخت افزار مورد نظر به پیچیدگی آن اضافه خواهد شد. با افزایش پیچیدگی سخت افزار های جدید پیاده سازی توسط این متد بسیار سخت و نا ممکن خواهد شد.



شکل ۱. متد طراحی پایین به بالا

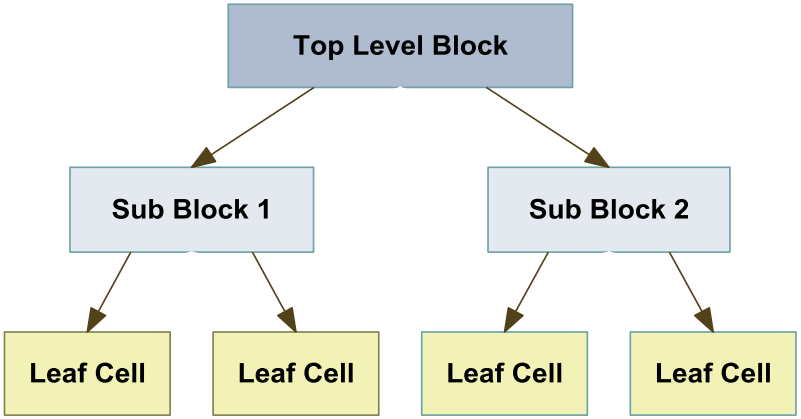
[1] Bottom to Up Design Methodology

مهندس یار

WWW.MOHANDESYAR.COM

- **سبک طراحی از بالا به پایین [1]**

متداول ترین متد در طراحی و توصیف یک سخت افزار متد طراحی از بالا به پایین می باشد، در این متد طراح بخش های مختلف یک سخت افزار را بصورت جداگانه و از بالا به پایین طراحی می کند. از مزیت های این متد می توان به قابلیت تست های اولیه، تغییر طراحی بخش های مختلف بدون ایجاد تغییرات در دیگر بخش ها ، طراحی روتین و ... اشاره کرد. در عوض پیاده سازی کامل این متد نیازمند اطلاعات و دید سیستمی می باشد که در مقایسه با متد قبل مشکل تر می باشد، به همین دلیل بیشتر طراحان سخت افزار در طراحی های خود از ترکیب هر دو متد جهت طراحی سیستم های دیجیتال استفاده می کنند.

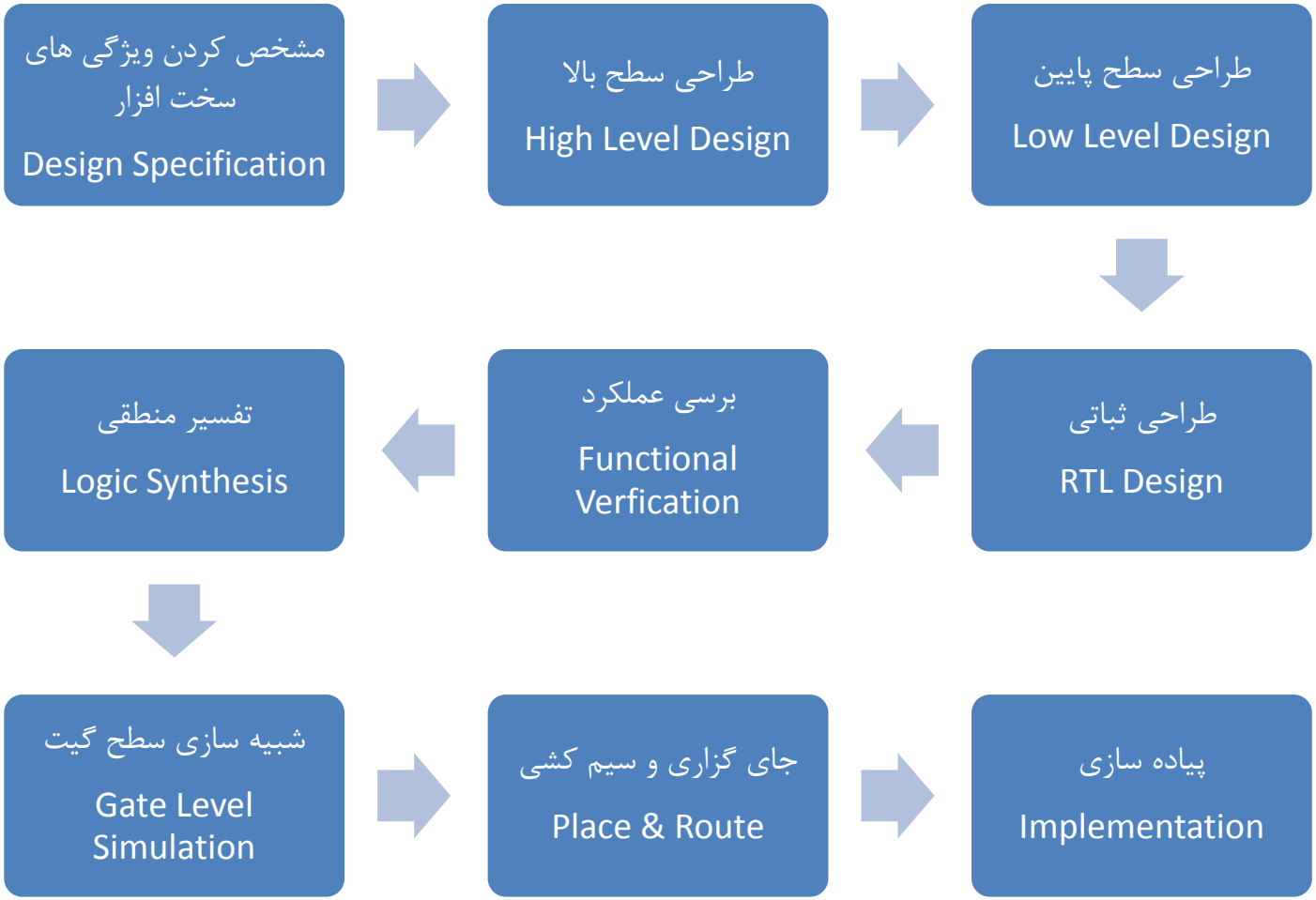


شکل ۲. متد طراحی بالا به پایین

[1] Top Down Design Methodology

مهندس یار

WWW.MOHANDESYAR.COM



شکل ۳. مراحل طراحی یک سخت افزار به روش بالا به پایین

مهندس یار

WWW.MOHANDESYAR.COM

۲) سطوح طراحی در زبان Verilog

این زبان از سطوح مختلفی جهت توصیف و طراحی سخت افزار پشتیبانی می کند. سه سطح مهم عبارتند از سطح رفتاری، سطح انتقال ثباتی و سطح گیت.

- **سطح توصیف گیت[1]**

در این سطح طراحی، سخت افزار مستقیماً توسط گیت های منطقی و اتصالات میان آنها توصیف می شود. در این سطح بصورت مستقیم با سیگنال[2] های ورودی، خروجی و داخلی سروکار داریم. طراحی در این سطح برای مدار های بزرگ و پیچیده کار بسیار مشکلی است و معمولا سخت افزار توسط سطوح بالاتر طراحی و توسط ابزار های سنتز[3] به سطح گیت تبدیل خواهد شد.

- **سطح توصیف انتقال ثباتی[4] (جریان داده ها)**

در این سطح مدار دیجیتالی با استفاده از ثبات ها و عملیات های ثباتی و انتقال داده ها بین یکدیگر طراحی می شود. در این سطح طراحی برای همزمان سازی بخش های مختلف طراح به یک سیگنال ساعت به وضوح احساس می شود.

- **سطح توصیف رفتاری[5]**

در این سطح سخت افزار برحسب الگوریتم های همزمان طراحی می شود. در این روش نیازی به توصیف جزئیات سخت افزاری طرح نمی باشد. هر الگوریتم به تنهایی بصورت ترتیبی بیان می شود بطوری که دستورات یکی پس از دیگری اجرا می شوند و مجموعه این الگوریتم ها رفتار سخت افزار را توصیف می کند. توابع، تسک ها و بلوک ها از ابزار اصلی در این سطح طراحی می باشند.

---

[1] Gate Level Modeling
[2] Signal
[3] Synthesis
[4] Register Transfer Level
[5] Behavioral Modeling

مهندس یار

WWW.MOHANDESYAR.COM

۳) متد طراحی از بالا به پایین

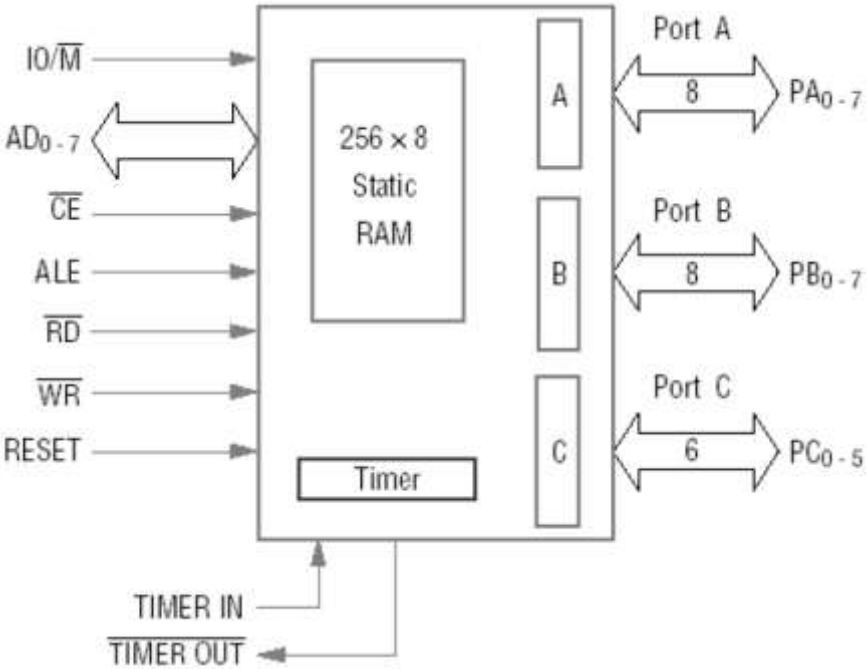
مراحل طراحی یک سخت افزار توسط این متد در زیر شرح داده شده است :

- مشخص کردن ویژگی های سخت افزار (Design Specification)

در مرحله ویژگی های کلی یک سخت افزار از قبیل نوع ورودی ها و خروجی و عملیات پردازشی بصورت کلی برسی می شود.

- طراحی سطح بالا (High Level Design)

در این مرحله ورودی و خروجی های سخت افزار بطور دقیق مورد برسی قرار می گیرند و برای دستیابی به خروجی های مورد نظر بلوک های مختلفی را تعبیه خواهیم کرد. اینکار درواقع شکاندن یک طراحی کلی به واحد های کوچک تر می باشد. این واحد های کوچکتر را بلوک می نامیم. برای مثال می خواهیم یک پردازنده ابتدایی طراحی کنیم ، بلوک های اصلی این پردازنده عبارتند از واحد پردازش مرکزی، واحد حافظه، ثبات ها، ورودی و خروجی و واحد کنترل می باشند.



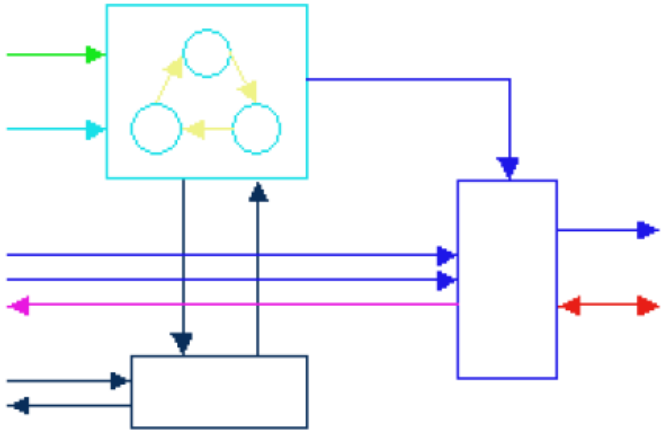
شکل ۴. نمونه ای از یک طراحی سطح بالا

مهندس یار

WWW.MOHANDESYAR.COM

- طراحی سطح پایین (Low Level Design)

در این مرحله طراح بر اساس ورودی و خروجی هر بلوک و نوع پردازشی که هر بلوک می بایست انجام دهد بلوک را پیاده سازی می کند. پیاده سازی در این مرحله توسط ثبات ها، دیکودر ها، مالتی پلکسر ها، شمارنده ها و ... انجام می شود.



شکل ۵. نمونه ای از یک طراحی سطح پایین

- طراحی ثباتی (RTL Design)

در این مرحله طراحی مرحله قبل به کد های Verilog و یا دیگر زبان های توصیف سخت افزار تبدیل می شود. تبدیل این طرح های مرحله قبل به کد های توصیف سخت افزار می بایست در چهارچوب و قوانین برنامه نویسی آن زبان صورت پذیرد.

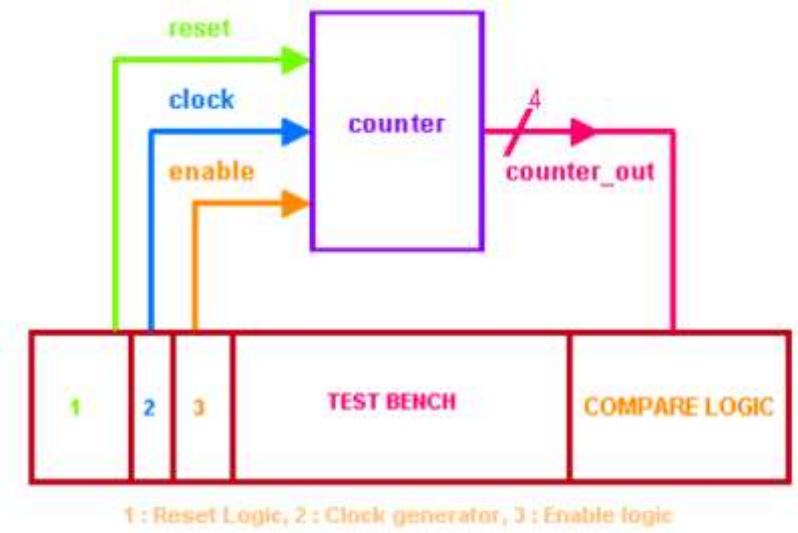
- برسی عملکرد (Functional Verification)

این مرحله شامل عملیاتی برای شبیه سازی سخت افزار طراحی شده می باشد. شبیه سازی به منظور اطمینان از کارکرد صحیح سخت افزار صورت می گیرد. در این مرحله می بایست از نرم افزار های شبیه سازی زبان توصیف سخت افزاری استفاده نمود. همچنین می بایست یک

مهندس یار

WWW.MOHANDESYAR.COM

سخت افزار جداگانه برای تست سخت افزار طراحی شده از قبل طراحی کرده باشیم . به این سخت افزار جدید Testbench گفته می شود. این سخت وظیفه تولید سیگنال های ورودی و سنجش سیگنال های خروجی سخت افزار طراحی شده را بر عهده دارد.



شکل ۶. سخت افزار تست جهت تست یک شمارنده

- تفسیر منطقی یا سنتز (Logic Synthesis)

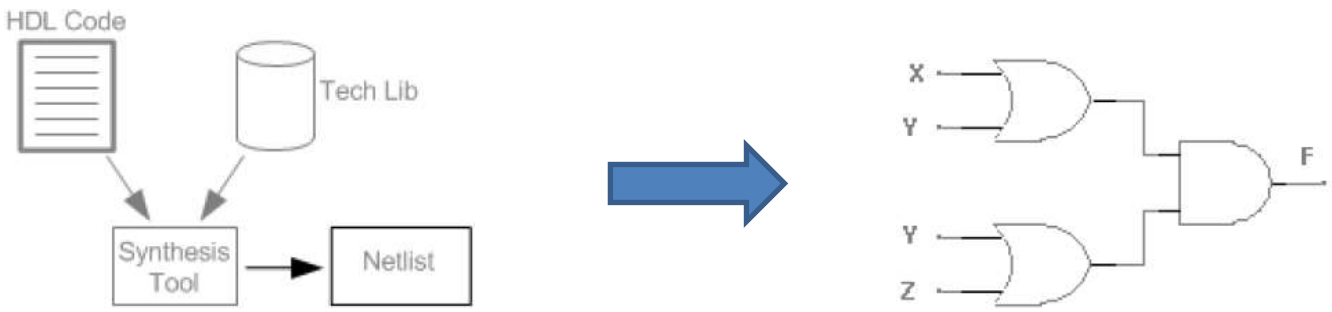
سنتز عملیات مشابه کامپایل در زبان های برنامه نویسی نرم افزاری می باشد. ابزار سنتز با استفاده از اطلاعات ورودی از قبیل نوع تکنولوژی سخت افزار مقصد، نقشه ورودی ها و خروجی ها و بانک اطلاعاتی اقدام به پردازش کد های RTL می نماید و در انتها آنها را تبدیل به نقشه سطح گیت خواهد کرد. ابزار سنتز همچنین پس از انجام عملیات سنتز اقدام به آنالیز مدار سنتز شده جهت اطمینان از همزمانی بخش های مختلف با یکدیگر و بوجود نیامدن مشکلات زمانی می نماید (قابل ذکر است که در عملیات سنتز تاخیر های موجود در سخت افزار های واقعی در نظر گرفته نمی شود) . به خروجی ابزار سنتز نت لیست گفته می شود که بیانگر گیت های منطقی و اتصالات میان آنهاست.

مهندس یار

WWW.MOHANDESYAR.COM

- شبیه سازی سطح گیت (Gate Level Simulation)

در این مرحله نت لیست تولید شده توسط ابزار سنتز شبیه سازی شده و عملکرد صحیح آن مورد برسی قرار خواهد گرفت .



شکل ۷. فرایند تبدیل کد های توصیف سخت افزار به کد های توصیف سطح گیت که به آن سنتز می گویند

- جای گذاری و سیم کشی (Place & Route)

این مرحله درواقع مربوط به طراحی مدار مجتمع مطابق با نت لیست تولید شده در مراحل قبلی می باشد.

مهندس یار

WWW.MOHANDESYAR.COM

۴) تعاریف اولیه

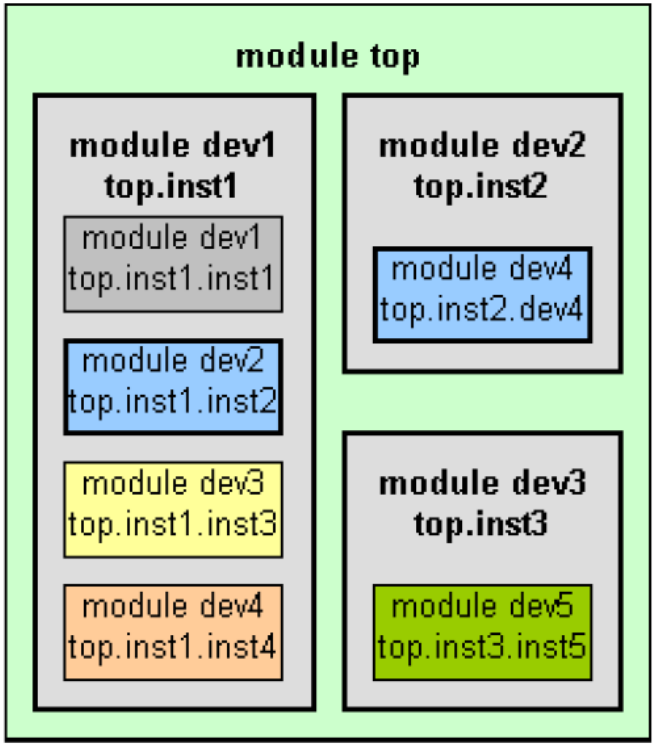
- **ماجول[1]**

برای توصیف سخت افزار های نسبتا پیچیده چاره ای جز تجزیه سخت افزار به بخش های کوچک تر نداریم، این بخش های کوچک را یک ماجول می نامیم. در زبان Verilog می توان مانند زبان های برنامه نویسی شی گرا (Java , C++ , C# ...) ساختار سلسله مراتبی[2] را بوسیله نمونه سازی از ماجول ها پیاده سازی کرد.

ماجول عملکرد مورد نیاز ماجول های بالاتر را فراهم می کند اما پیاده سازی درون آن را پنهان می کند.

- **نمونه[3]**

یک ماجول الگویی از یک عنصر واقعی را پیاده سازی میکند، هنگامی که از این ماجول استفاده می شود Verilog یک نمونه از این الگو را بصورت خودکار خواهد ساخت و ورودی و خروجی های مورد نظر شما را به آن متصل خواهد کرد. این نمونه درواقع یک کپی از ماجول اصلی می باشد. برای مثال شما یک ماجول ثبات ۸ بیتی طراحی میکنید و با نمونه سازی از این ماجول در بخش های مختلف چندین ثبات دیگر از همین نوع را پیاده سازی خواهید کرد. به این کار Instantiation یا نمونه سازی و این عنصر را Instance یا نمونه می نامند.



شکل ۸. نمایی از یک ماجول و ماجول های درون آن

[1] Module
[2] Hierarchy Structure
[3] Instance

مهندس یار

WWW.MOHANDESYAR.COM

۵) قواعد دستوری و پایه ای زبان Verilog

قواعد و دستور زبان Verilog شباهت زیادی به زبان C دارد. زبان Verilog نسبت به حروف کوچک و بزرگ حساس می باشد. تمامی حروف اختصاصی در این زبان مانند زبان C با حرف کوچک نوشته می شوند. در Verilog هم مانند C پس از هر توکن (دستور و داده) می بایست از علامت ; (Semi colleen) استفاده نمود.

- **فاصله خالی**

در زبان Verilog فاصله خالی در هنگام سنتز توسط ابزار سنتز در نظر گرفته نمی شود. فقط در مواردی که فاصله درون یک رشته و یا به عنوان جدا کننده توکن ها (دستورات و داده ها) استفاده شده باشد در در نظر گرفته خواهد شد. کرکتر های زیر به عنوان فاصله خالی در نظر گرفته خواهند شد :

Blank Space (\s) , Tab (\t) , Carriage return (\r), New line (\n)

- **توضیحات**

نحوه توضیح نویسی در زبان Verilog کاملا مشابه به زبان C می باشد .

توضیح تک خطی : // Comments

توضیحات تک خطی با علامت // شروع می شوند و با یک شروع یک خط جدید (کارکتر Carriage Return) خاطمه می یابند.

توضیح چند خطی : /* Multi Line Comments */

توضیحات چند خطی با */ شروع و با /* تمام می شوند. در این حالت توضیحات می تواند دارای چندین خط باشد.

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۱ : در این مثال با نحوه توضیح گذاری صحیح و اشتباه آشنا می شود، عبارتی که با پس زمینه زرد مشخص شده است از قواعد زبان Verilog پیروی نمی کند.

در این مثال با نحوه توضیح گذاری چند خطی آشنا خواهید شد.

```
/* 1-bit adder , Example of Multi Line
Comment In Verilog
The Comment Starts with /* And End With next Token You see */
module adder(
    a, b,
    ci, sum, // Verilog will Ignore White Spaces
    co          // So You Can Define All of Your Variables in One Or In Separated Lines
);

// The next line represent Wrong Single Line
Commenting, this line will generate Verilog Syntax Error During Synthesis.

// Input Ports
input a; input b;
input ci;

// Output Ports
output sum; output co;

// Data Types
wire     a; // After a token like C you must put
wire     b; wire ci; wire sum; wire co;
```

همانطور که دیده می شود، می توان تمامی متغیر ها را در یک خط و بصورت پشت هم و یا در خطوط جداگانه و بصورت ترکیبی تعریف نمود. از نظر زبان Verilog این دو هیچ فرقی با هم ندارند. اما برای پیروی از یک الگوی واحد بهتر است متغیر هایی که در یک دسته بندی قرار می گیرند (مثلا تمام متغیر های ورودی ، ویا تمام متغیر های ثباتی و ...) را در یک خط تعریف نمود.

مهندس یار

WWW.MOHANDESYAR.COM

- **کلید واژه ها[1] در Verilog**

در Verilog تمامی کلید واژه ها (Keywords) بصورت حروف کوچک می باشند، از آنجایی که Verilog یک زبان حساس به حروف کوچک و بزرگ است لذا می توان از کلید واژه ها بصورت حروف بزرگ برای بیان متغیر ها ، نام ها و ... استفاده نمود.

به نام های اختصاری در این زبان Unique Name گفته می شود.

نکته : توصیه می شود از کلید واژه ها حتی در صورت حروف بزرگ بودن به عنوان Unique Name استفاده نکنید .

مثال ۲ : در این مثال کلماتی که به رنگ آبی نوشته شده اند کلید واژه های زبان Verilog می باشند. اما کلمات مشکی رنگ Unique Name می باشند.

```
Reg             // a Verilog Keyword
Output          // a Verilog Keyword
Reg             // a Unique Name ( Not Keyword )
REG             // a Unique Name ( Not Keyword )
```

- **معرف ها[2] در Verilog**

به نام های اشیا در این زبان معرف (Identifier) گفته می شود، نام ثبات ها و ماجول ها همگی از این نوع هستند.

- معرف ها می بایست با یکی از حروف الفبا (توجه کنید فقط حروف الفبا و نه اعداد و نشانه ها) و یا علامت _ (Underscore) شروع شوند.
- معرف ها می توانند حاوی حروف الفبا (کوچک و بزرگ) ، اعداد (۰ تا ۹) و علامت دلار ($) باشند.

---

[1] Keyword
[2] Identifier

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۳ : در این مثال نحوه نام گذاری صحیح معرف ها را خواهیم دید ، معرف هایی که با پس زمینه زرد رنگ مشخص شده اند صحیح نمی باشند و قواعد زبان Verilog را رعایت نمی کنند.

```
// Valid Identifiers
clock_input            counterEnable              Carray$Output
i386                    A                          _TempRegister

// Wrong Identifiers
$i386        2ndClockSource        Colck#2
```

در زبان Verilog هم چنین می توان معرف ها را بصورت Escaped تعریف نمود، در این حالت شما می توانید از همه کرکتر های قابل چاپ جهت نام گذاری معرف استفاده کنید. این کرکتر ها عبارتند از تمامی حروف الفبا انگلیسی، اعداد و نشانه ها (کرکتر های اسکی از کد هگزدسیمال 0x21 تا 0x7E). در این روش نام گذاری معرف با کرکتر بک اسلش (\ ) شروع شده و تا زمانی که اولین فاصله مشاهده شود ادامه دارد. به عبارتی دیگر کرکتر شروع معرف بک اسلش و کرکتر اتمام آن فاصله می باشد.

مثال ۴ : تمامی معرف های زیر صحیح می باشند.

```
// Valid Identifiers
\486_up        \Q!        \1,2,3        \reset*        \clock#2        \!Enable
```

- **اعداد صحیح در Verilog**

در زبان Verilog می توان اعداد را به فرمت های دسیمال ، هگز دسیمال ، اوکتال و یا باینری تعریف نمود. اعداد منفی در Verilog بصورت متمم ۲ تعریف خواهند شد. در تعریف اعداد می توان از علامت _ (Under Score) برای جدا سازی اعداد استفاده نمود، Verilog بصورت خودکار این علامت را از اعداد حذف می کند. (باید توج داشت که از این علامت نمی توان در ابتدای عدد استفاده کرد).

مهندس یار

WWW.MOHANDESYAR.COM

Verilog امکان تعریف اعداد به صورت های زیر را فراهم می کند :

- اعداد دارای اندازه و بی اندازه (اعداد بی اندازه به صورت پیشفرض ۳۲ بیتی تعریف می شوند).
- اعداد در مبنا های مختلف نظیر باینری ، اوکتال ، دسیمال و هگز دسیمال. (نشانگر مبنا با حرف کوچک نمایش داده می شوند ، ارقام هگز با حروف بزرگ و کوچک نمایش داده می شوند.)

فرمت کلی اعداد در Verilog بصورت زیر می باشد :

<Size>'<Radix><Value>

Value = مقدار ، Radix = مبنا ، Size = اندازه

افزودن فاصله بین نشانگر های اندازه ، مبنا و مقدار اعداد مجاز بوده Verilog بطور خودکار این فاصله ها را حذف می کند.

مثال ۵ : در این مثال با انواع نمایش های مختلف آشنا خواهید شد.

نحوه نمایش	مقدار ذخیره شده واقعی	توضیحات
1	00000000000000000000000000000001	unsized 32 bits
8'hAA	10101010	sized hex
6'b10_0011	100011	sized binary
'hF	00000000000000000000000000001111	unsized 32 bit hex
8'F1	11110001	8bit sized hex
16'FF F0	1111111111110000	16bit with separated digits
16'FF_F0	1111111111110000	16bit with separated digits

مهندس یار

WWW.MOHANDESYAR.COM

در Verilog اعداد از سمت راست شروع و با توجه اندازه مشخص شده ادامه می یابد.

- اگر اندازه کوچکتر از مقدار باشد ، مقادیر اضافه سمت چپ حذف می گردند.
- اگر اندازه بزرگتر از مقدار باشد ، مقادیر باقی مانده سمت چپ با توجه به آخرین بیت سمت چپ خوانده شده پر خواهند شد. اگر آخرین بیت 0 و یا 1 باشد با 0 ، اگر اخرین بیت Z و یا X باشد به ترتیب با Z و X پر خواهد شد.

مثال ۶ :

نحوه نمایش	مقدار ذخیره شده واقعی	توضیحات
6'hCA	001010	Truncated , not 11001010
6'hA	001010	Filled with two 0 on left
16'bZ	Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z	Filed with 16 Z
8'bx	X X X X X X X X	Filled with 8 X

- **اعداد حقیقی در Verilog**

زبان Verilog قابلیت پشتیبانی از اعداد حقیقی بصورت متغیر و ثابت را دارا می باشد. Verilog به طور خودکار این اعداد را روند کرده و به عدد صحیح قابل ذخیره سازی در حافظه تبدیل می کند. اعداد حقیق نمی توانند شامل Z یا X باشند. اعداد حقیقی می توانند با نمایش دسیمال و یا علمی تعریف شوند.

<Value>.<Value>
<Mantissa>E<Exponent>

Mantissa = رقم اعشاری ، Exponent = توان

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۷ : عبارت های زیر همگی نحوه نمایش صحیح اعداد حقیقی می باشند.

```
// Valid Real Numbers
1.2        0.6        3.142673        3.6E8        2.7461E6
```

- **اعداد علامت دار و بی علامت در Verilog**

در Verilog مانند زبان های دیگر نمی توانیم علامت دار بودن و یا نبودن اعداد را مشخص کنیم. هر عددی که علامت – قبل از آن آورده شود منفی و در غیر این صورت مثبت خواهد بود. Verilog اعداد منفی را بصورت متمم دو ذخیره می کند.

- **پورت[1] ها در Verilog**

پورت ها در واقع ابزار ارتباط ماجول ها با دنیای بیرون و یکدیگر می باشند. هر ماجول می تواند دارای تعدادی ورودی و تعدادی خروجی باشد. تمام ماجول ها بجز ماجول مادر (بالاترین ماجول در سلسله مراتب) می توانند دارای پورت باشند.

پورت ها را می توان بصورت ورودی ، خروجی و یا ورودی/خروجی تعریف نمود. قالب کلی تعریف پورت بصورت زیر می باشد :

```
input  [range_vals:range_var] list_of_identifiers;
output [range_vals:range_var] list_of_identifiers;
inout  [range_vals:range_var] list_of_identifiers;
```

برای ارتباط پورت ها با دنیای بیرون دو روش در زبان Verilog وجود دارد : روش ترتیبی و روش نام گذاری

در روش ترتیبی به هنگام نمونه سازی از ماجول می بایست متغیر های اتصال به پورت ها را دقیقا به همان ترتیبی که در تعریف پورت آورده شده است بیاوریم. در این روش اگر بخواهیم یک پورت خروجی به جایی متصل نباشد کافی است جای آن را خالی بگذاریم.

[1] Port

مهندس یار

WWW.MOHANDESYAR.COM

در روش نام گذاری از اسامی پورت ها که در تعریف ماجول آورده شده است استفاده می نماییم . معمولا زمانی که تعداد ماجول ها و پورت های آنها افزایش می یابد و بخاطر سپردن ترتیب آنها دشوار می شود استفاده می شود. در این روش اگر بخواهیم یک پورت خروجی به جایی متصل نباشد کافی است نام آن را نیاوریم.

قالب استفاده از روش نام گذاری بصورت زیر می باشد :

```
.port_name( external_signal_name )
```

مثال ۸ : در بخش اول این مثال چند حالت تعریف پورت ها آورده شده است . در دو بخش دوم دو روش ارتباط دهی نمایش داده شده است. جمع کننده های ۱ تا ۶ (adder1 تا adder6) همگی یک نمونه از ماجول adder که قبلا تعریف کرده بودیم می باشند (در صفحات قبل فقط پورت های این ماجول تعریف شده اند نه اجزای داخلی آن).

جمع کننده های ۱ و ۴ ، ۲ و ۵ ، ۳ و ۶ دارای ورودی و خروجی های یکسانی هستند، تنها فرق آنها در روش ارتباط آنهاست. در جمع کننده های ۱ تا ۳ از روش ترتیبی و در جمع کننده های ۴ تا ۶ از روش نام گذاری استفاده شده است.

```
input          clk;          // Clock input
input  [15:0] data_in;       // 16bit Data input    bus
output [7:0]  data_out;      // 8bit Data output bus
inout  [15:0] data_bus       // 16bit Bi-Directional data bus

adder1    adder( data_a, data_b, carry_i, summation, carry_o);
adder2    adder( data_a, data_b, carry_i, summation);
adder3    adder( data_a, data_b, carry_i, , carry_o);

adder4    adder( .a(data_a), .b(data_b), .sum(summation), .ci(carry_i), .co(carry_o));
adder5    adder( .a(data_a), .b(data_b), .sum(summation), .ci(carry_i));
adder6    adder( .a(data_a), .b(data_b), .ci(carry_i), .co(carry_o));
```

مهندس یار

WWW.MOHANDESYAR.COM

- **انواع داده ها در Verilog**

دو نوع داده مهم در Verilog عبارتند از Net و Register . داده Net نمایانگر ارتباط پایه ای میان قسمت های مختلف است. داده Register نمایانگر یک متغیر برای ذخیره سازی اطلاعات است. هر یک از داده های فوق می توانند بصورت تکی و یا برداری[1] تعریف شوند.

نت (Net) ها در واقع اتصالات واقعی میان مدارات هستند. نت ها در Verilog توسط کلید واژه wire تعریف می شود و مقدار پیشفرض آن Z می باشد.

در Verilog مجموعا چهار مقدار برای هر نت وجود دارد ، این چهار مقدار عبارتند از :

مقدار	توضیح
0	سطح منطقی 0 ، شرط نادرست
1	سطح منطقی 1 ، شرط درست
x	مقدار نا مشخص
z	حالت امپدانس بالا (شناور)

ثبات (Register) ها نیز در Verilog توسط کلید واژه reg تعریف می شوند.

- ثبات ها آخرین مقداری که به آنها داده شده است را تا زمانی که مقدار جدیدی به آنها داده شود در خود نگه داری می کنند.
- می توان یک آرایه از ثبات ها را بوجود آورد که در این صورت به آن حافظه (Memory) گفته می شود.
- از ثبات ها به عنوان متغیر در بلوک های ساخت یافته(Procedural) استفاده می شود.

[1] Vector

مهندس یار

WWW.MOHANDESYAR.COM

- اگر بخواهید یک سیگنال حاوی مقدار را به یک بلوک ساخت یافته انتقال دهید باید از ثبات استفاده کنید.

چهار نوع ثبات اصلی در Verilog عبارتند از :

نوع ثبات	توضیح
reg	متغیر بدون علامت
integer	متغیر علامت دار ۳۲ بیتی
time	عدد صحیح ۶۴ بیتی
real	متغیر ممیز شناور با دقت مضاعف

- **رشته ها[1] در Verilog**

یک رشته مجموعه ای از کاراکتر ها می باشد که درون دو علامت " (double quotation) قرار گرفته اند. باید توجه داشت در Verilog رشته ها بصورت تک خطی می باشند و نمی توان آنها را بصورت چند خطی تعریف نمود. در Verilog کرکتر ها بصورت ۸ بیت و بصورت اسکی (ASCII) تعریف می شوند. در Verilog بر خلاف C نیازی به کرکتر انتهایی (Termination Character) برای نمایش انتهای رشته نمی باشد. رشته ها در Verilog در یک ثبات ذخیره می شوند و طول ثبات باید به اندازه رشته ورودی بزرگ باشد. زمانی که یک ثبات رشته در Verilog بزرگتر مقدار داده شده با آن باشد، Verilog بطور خودکار بیت های سمت چپ اضافی را با صفر پر خواهد کرد. کرکتر های خاص در Verilog را می توان با پیشوند بک اسلش (\ ) تعریف نمود ، برخی از این کرکتر ها عبارتند از :

[1] Strings

مهندس یار

WWW.MOHANDESYAR.COM

نمایش کرکتر	توضیح
\n	کرکتر خط جدید
\t	کرکتر تب
\\	کرکتر بک اسلش
\"	کرکتر دبل کوتیشن
\%	کرکتر درصد

- **آرایه[1] ها Verilog**

در Verilog می توان، آرایه ای از داده ها و یا بردار ها تعریف نمود. در Verilog بر خلاف زبان های نرم افزاری نمی توان آرایه های چند بعدی تعریف نمود.

فرمت کلی تعریف آرایه به صورت زیر می باشد :

```
<array_type> <array_name> [ #first_element : #last_element ]
```

تفاوت آرایه و بردار در این است که یک آرایه می تواند مجموعه ای از عناصر با طول های مختلف را در خود نگه داری کند اما بردار تعداد مشخصی از یک عنصر است.

- **حافظه[2] ها در Verilog**

در Verilog حافظه ها بصورت آرایه ای از ثبات ها تعریف می شوند. فرمت کلی تعریف یک حافظه در Verilog بصورت زیر می باشد :

```
reg [ #word_size : 0 ] array_name [0:array_size]
```

مثال ۹ : در این مثال نحوه تعریف ثبات ها ، نت ها و ثبات های برداری ، متغیر های حقیقی و صحیح و حالت برداری آنها ، رشته و حافظه آشنا می شوید . توجه داشته باشید که رشته پس از تعریف و درون یک بلوک ساخت یافته مقدار دهی می شود.

---

[1] Array
[2] Memory

مهندس یار

WWW.MOHANDESYAR.COM


```
reg register;                    // Declaring a one bit Register

wire [15:0] Databus;             // a 16 nets Representing a 16bit bus
reg  [31:0] AddressReg;          // a 32 bit Register

integer Counter;                 // Integer Data Declaration
real    Real_Data;               // Real Data Declaration
integer Output_Data[15:0];       // Array of 16 integers

reg membit[0:1023];              // 1K x 1bit memory
reg [7:0] membyte[0:1023];       // 1K x 8bit memory

reg [8*17:0] version ;           // Declare a register variable that is 18 bytes
initial
    version = " version  1.0";   // Filling String inside initial procedural  block
```

- **ماجول ها در Verilog**

با ماجول ها در Verilog قبلا آشنا شدیم ، در این بخش به نحوه تعریف ماجول و ویژگی های آن می پردازیم. قالب کلی تعریف ماجول به صورت زیر می باشد :

```
module module_name ( port list );
        port declarations (if ports present)
        parameters (optional)
        Declaration of wires, regs and other variables
        Data flow statement (assign)
        Instantiation of lower level module
        always and initial blocks, all behavioral
        statements go in these blocks
        tasks and functions
end module
```

مهندس یار

WWW.MOHANDESYAR.COM

- **قوانین اتصال پورت ها در Verilog**

به هنگام اتصال پورت های یک ماجول به جهان خارج باید به نکاتی توجه داشت :

- پورت های ورودی ماجول باید از نوع net باشند و این پورت ها می توانند به متغیر هایی از نوع reg و یا net در جهان خارج متصل شوند.
- پورت های خروجی ماجول می توانند از نوع reg و یا net باشند و این پورتها باید به متغیر هایی از نوع net در جهان خارج متصل شوند.
- پورت های دو سویه ماجول باید از نوع net باشند ، و این پورتها می توانند به متغیر هایی از نوع reg و یا net در جهان خارج متصل شوند.
- پورتهای ماجول و متغیر های خارجی متصل به آنها باید از نظر طول منطبق باشند

- **پیش پردازش[1] در Verilog**

در Verilog نیز مانند تمامی زبان های برنامه نویسی دیگر می توان پارامتر های پیش پردازش را تعریف نمود و در صورت نیاز از آنها استفاده نمود. یکی از پرکاربر ترین دستورات پیش پردازشی در Verilog دستورdefine می باشد. توسط این دستور می توان یک مقدار ، دستور ، شرط ، ... را با یک نام تعریف نمود و از آن پس فقط از آن نام در کد ها استفاده کرد. قالب کلی استفاده از دستور define به صورت زیر است:

```
`define <Definition-Name> <Definition-Statement>
```

برای استفاده از این تعریف در کد ها فقط کافی است نام آن را به همراه علامت ` بیاورید. به صورت زیر :

```
`<Definition-Name>
```

از این دستور در مثال های آخر کتابچه به دفعات استفاده خواهیم کرد.

[1] Preprocessing

مهندس یار

WWW.MOHANDESYAR.COM

۶) مدل سازی در سطح گیت

در مدل سازی سطح گیت مدار را بصورت مجموعه ای از گیت های پایه که به یکدیگر متصل شده اند بیان می کنیم. Verilog هم مانند دیگر زبان های توصیف سخت افزار درون خود دارای گیت های منطقی[۱] ، خطوط انتقال[۲] و سوییچ[۳] ها می باشد اما با توجه به وجود ابزار های سنتز از این اشیا به ندرت در طراحی ها استفاده می شود. برای شروع ابتدا باید انواع گیت های پایه ای که در Verilog وجود دارند را معرفی کنیم.

در Verilog شش گیت پایه ای اصلی وجود دارد ، این گیت ها عبارتند از گیت های AND ، NAND ، OR ، NOR ، XOR و XNOR . همگی این گیت ها دارای چند ورودی و یک خروجی می باشند. اولین پورت هر گیت خروجی آن و پورت های بعدی ورودی های گیت را تشکیل می دهند بدین ترتیب می توان گیت هایی با تعداد ورودی های مختلف تعریف نمود.

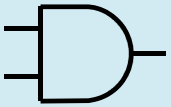
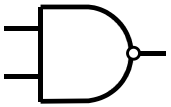
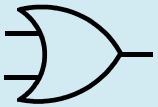
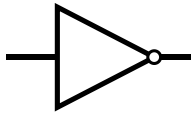
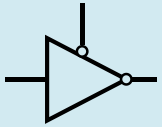
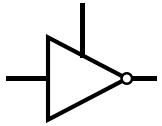
۶ گیت انتقالی اصلی عبارتند از گیت های وارونگر[۴] ، بافر[۵] ، بافر سه[۶] حالته فعال پایین، بافر سه حالته فعال بالا ، وارونگر سه حالته فعال پایین و وارونگر سه حالته فعال بالا. پورت آخر گیت های دو حالته ورودی و دیگر پورت ها خروجی هستند ، بدین ترتیب می توان گیت هایی با تعداد خروجی های دلخواه تعریف نمود. در گیت های سه حالته پورت اول خروجی ، پورت دوم ورودی و پورت سوم پورت کنترل می باشد.

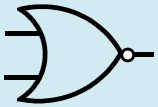
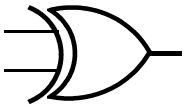
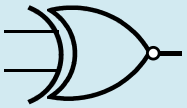
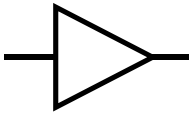
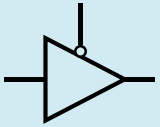
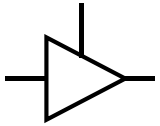
---

[1] Logic Gates
[2] Transmission Gates
[3] Switch
[4] Inverter
[5] Buffer
[6] Tri State Buffer

مهندس یار

WWW.MOHANDESYAR.COM

نام گیت	نماد	کلید واژه
گیت AND با N ورودی		and
گیت NAND با N ورودی		nand
گیت OR با N ورودی		or
گیت NOT با N خروجی		not
وارونگر سه حالته فعال پایین		notif0
وارونگر سه حالته فعال بالا		notif1

نام گیت	نماد	کلید واژه
گیت NOR با N ورودی		nor
گیت XOR با N ورودی		xor
گیت XNOR با N ورودی		xnor
گیت بافر با N خروجی		buf
بافر سه حالته فعال پایین		bufif0
بافر سه حالته فعال بالا		bufif1

برای طراحی مدار در سطح گیت ابتدا باید آنرا بصورت مجموعه ای از گیت های پایه در آورد، سپس با تعریف net های مورد نیاز این گیتهای پایه را به یکدیگر متصل نمود. به این کار اصطلاحا سیم بندی یا Wiring گفته می شود.

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۱۰ : در این مثال با استفاده دو گیت NAND یک گیت AND می سازیم.

```
module and_from_nand(X, Y, F);
        input  X, Y;
        output F;

        wire   W;

        nand U1(X, Y, W);
        nand U2(W, W, F);

endmodule
```

در این ماجول X و Y ورودی و F خروجی می باشد.

مثال ۱۱ : در این مثال با ساخت یک فلیپ فلاپ نوع دی آشنا می شویم.

```
module dff(Q, Q_BAR, D, CLK);

        output Q,Q_BAR;
        input D, CLK;

        nand U1(X, D, CLK);
        nand U2(Y, X, CLK);
        nand U3(Q, Q_BAR, X);
        nand U4(Q_BAR, Q, Y);

endmodule
```

در این ماجول D و CLK ورودی ، Q و Q_BAR خروجی می باشند.

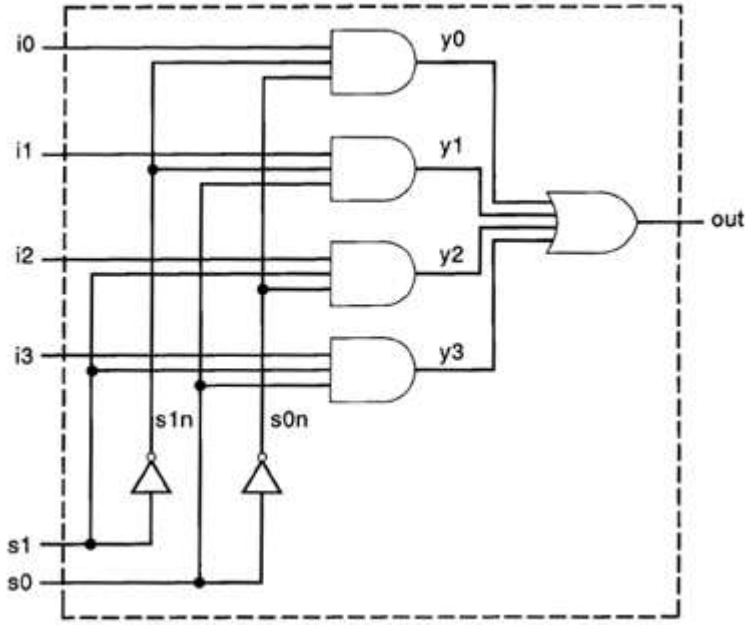
توجه کنید گیت nand می تواند دارای تعداد نا متناهی ورودی باشد. در این صورت اولین آرگومان خروجی و دیگر آرگومان ها ورودی می باشند.

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۱۲ : در این مثال با ساخت یک مالتی پلکسر[1] آشنا می شویم.

```
module mux4_to_1 ( out, i0, i1, i2, i3, s1, s0 );
    output out;
    input i0, i1, i2, i3, s1, s0;

    // Internal wire declaration
    wire s0n, s1n, y0, y1, y2, y2;

    // Gate instantiations
    not ( s1n, s1 );
    not ( s0n, s0 );
    and ( y0, i0, s1n, s0n );
    and ( y1, i1, s1n, s0 );
    and ( y2, i2, s1 , s1n );
    and ( y3, i3, s1 , s0 );
    or ( out, y0, y1, y2, y3 );
endmodule
```



شکل ۹. نموادر منطقی یک مالتی پلکسر چهار به یک

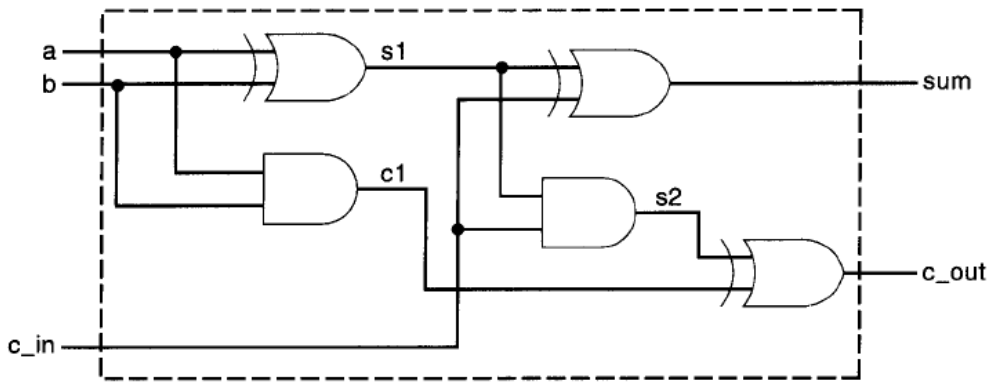
[1] Multiplexer

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۱۳ : در این مثال یک جمع کنند کامل ۴ بیت را بررسی خواهیم کرد.

برای ساخت یک جمع کننده چهار بیت به چهار جمع کننده کامل ۱ بیتی با بیت نقلی ورودی و خروجی نیازمندیم که نمودار و کد Verilog آن را در ادامه می بینید. در این مثال از سطح توصیف گیت برای توصیف مدار داخلی جمع کننده استفاده شده است.



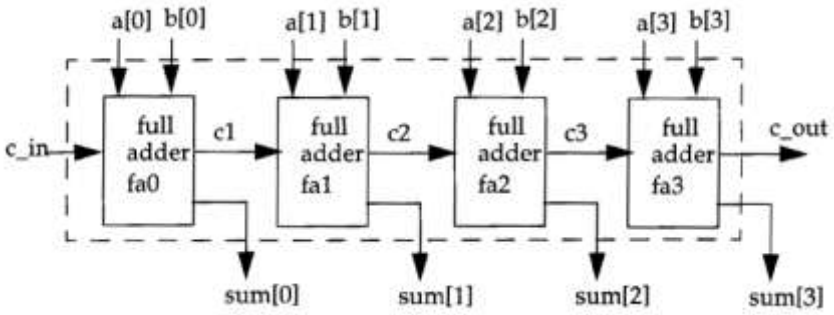
شکل ۱۰. نمودار منطقی یک جمع کننده کامل

```
module fulladd(sum, c_out, a, b, c_in);
    output sum, c_out;
    input a, b, c_in;

    wire s1, c1, c2;

    xor (s1, a, b);
    and (c1, a, b);

    xor(sum, s1, c_in);
    and(c2, s1, c_in);

    or(c_out, c2, c1);

endmodule
```

مهندس یار

WWW.MOHANDESYAR.COM

در مرحله بعد می بایست چهار عدد از این جمع کننده ها را به طریق زیر به هم متصل کنیم.



شکل ۱۱. نمودار یک جمع کننده چهاربیتی با استفاده از چهار جمع کننده ۱ بیت

```
module fulladder4(sum, c_out, a, b, c_in);

    output [3:0] sum;
    output c_out;

    input [3:0] a,b;
    input c_in;

    wire c1, c2, c3;

    // Making Four Instances of fulladder which defined before
    fulladd fa0(sum[0], c1,     a[0], b[0], c_in);
    fulladd fa0(sum[1], c2,     a[1], b[1], c1);
    fulladd fa0(sum[2], c3,     a[2], b[2], c2);
    fulladd fa0(sum[3], c_out,  a[3], b[3], c3);

endmodule
```

همانطور که در کد Verilog مشاهده می شود در این مثال چهار بار از جمع کننده تک بیت که در بخش قبل طراحی کرده بودیم نمونه سازی کردیم و توسط آن یک جمع کننده چهار بیت را طراحی نمودیم.

مهندس یار

WWW.MOHANDESYAR.COM

۷) مدل سازی سطح ثباتی (جریان داده ها)

در مدل سازی ثباتی به نحوه انتقال اطلاعات بین ثبات ها و پردازی اطلاعات اهمیت می دهیم. در این قسمت به چگونگی مدل سازی در سطح جریان داده در زبان Verilog و مزایای آن می پردازیم.

- **مقداری دهی مداوم[1]**

توسط این دستور می توان یه مقدار را روی یک Net قرار داد. این دستور همواره در حال اجرا بوده و هرگاه یکی از عملوند های تغییر کند مقدار Net هم همراه با آن (و یا با تاخیر) تغییر خواهد کرد. قالب کلی استفاده از این دستور به صورت زیر است :

```
assign <delay> Assignment-Lists;
```

سمت چپ این دستور باید یک متغیر یا یک بردار از نوع net و یا الحاقی از این دو نوع باشد.

- **مقدار دهی مداوم ضمنی[2]**

در این روش به جای اینکه یک متغیر را از جنس Net تعریف کنیم و سپس توسط دستور assign یک مقدار را روی آن قرار دهیم، می توانیم این عمل را در هنگام تعریف Net انجام دهیم.

[1] Continuous Assignment
[2] Implicit Continuous Assignment

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۱۴ : در این مثال نحوه مقدار دهی بصورت مداوم و مداوم ضمنی را مشاهده خواهید کرد.

```
module mymodule_continuous_assignment( out, i1, i2 );
    output out;
    input i1,i2;
    assign out = i1 & i2;   // Continuous Assignment
endmodule

module mymodule_implicit_continuous_assignment( out, i1, i2 );
    output out;
    input i1,i2;
    wire out = i1 & i2;     // Implicit Continuous Assignment
endmodule
```

- **تاخیر ها[1]**

به منظور واقعی تر شدن نتایج شبیه سازی می توان در Verilog تاخیر های که ناشی از تاخیر انتشار گیت های منطقی[2] می باشند را لحاظ کرد. باید توجه داشت این تاخیرات فقط در نتایج شبیه سازی تاثیر می گذارند و در سنتز و واقعیت اثری نخواهند داشت.

- **تاخیر با قاعده**

در این حالت یک تاخیر را پس از دستور assign و قبل از net می آوریم. هر تغییری که در یکی از سیگنال های سمت راست رخ دهد، باعث می شود پس از گذشت زمان تاخیر ، عبارت سمت راست مجددا ارزیابی شود و سپس در متغیر سمت چپ قرار گیرد. به این ترتیب تاخیری که در اینجا داریم از نوع inertial است و این امر باعث می شود که پالسهایی با عرض کمتر از مقدار تاخیر مشخص شده به خروجی منتشر نشود.

---

[1] Delay
[2] Logic Gates Propagation Delay

مهندس یار

WWW.MOHANDESYAR.COM

- **تاخیر به هنگام تعریف net**

در این حالت به هنگام تعریف net تاخیر مورد نظر را برای آن مشخص می کنیم. از این پس هر تغییری که روی این net انجام شد، با تاخیر مشخص شده اعمال می شود. این عمل در مورد توصیف سطح گیت نیز قابل استفاده است.

مثال ۱۵ : در این مثال با نحوه ایجاد تاخیر های مورد بحث با قاعده و تاخیر به هنگام تعریف نت آشنا می شوید.

```
// ----- Inertial Delay Example -------
module and_with_delay( o1, i1, i2);
    output o1;
    input i1, i2;

    assign #5 o1 = i1 & i2; // Inertial Delay
endmodule

// ----- Implicit Delay Example ------
wire #5 out;
assign out = i1 & i2;        // Implicit Delay during Assignment
```

۸) عبارات ، عملگر ها[1] و عملوند[2] ها

- یک عبارت از ترکیب عملوند ها و عملگر ها ساخته می شود
- یک عملوند می تواند یکی از انواع داده ای باشد که قبلا به آن اشاره شد.
- عملگر ها روی عملوند ها عملیاتی را انجام می دهند تا نتیجه مطلوب بدست آید. لیست عملگر های موجود در Verilog در جدول صفحه بعد آمده است.

[1] Operator
[2] Operand

مهندس یار

WWW.MOHANDESYAR.COM

در مورد عملگر های حسابی[1] باید به موارد زیر توجه داشت :

- اگر هر دو عملوند صحیح باشد خارج قسمت را بر می گرداند.
- اگر هریک از عملوند ها دارای بیت x باشند ، نتیجه عملیات x خواهد بود.
- عملگر ٪ باقی مانده تقسیم را بر می گرداند و علامت حاصل برابر علامت عملوند اول است.
- بهتر است اعداد منفی را در عبارات بصورت اعداد صحیح بکار برد، زیرا در غیر این صورت به مکمل ۲ تبدیل می شوند که ممکن است باعث نتایج غیر منتظره[2] شوند.

در مورد عملگر های منطقی[3] باید به موارد زیر توجه داشت :

- نتیجه عملگر های منطقی یک بیت است : 0 نادرست ، 1 درست و X نا معلوم .
- اگر عملوند اول 0 باشد معادل نادرست[4]، اگر 1 باشد معادل درست[5] و اگر X باشد نا معلوم ارزیابی خواهد شد.

---

[1] Arithmetic Operators
[2] Unexpected Result
[3] Logic Operators
[4] False Condition
[5] True Condition

مهندس یار

WWW.MOHANDESYAR.COM

نوع عملگر	سمبل عملگر	عملیات	عملوند ها
عملگر های رابطه ای[1] ، تساوی ، بیتی	<	کوچکتر از	۲
	>	بزرگتر از	۲
	<=	کوچکتر و مساوی	۲
	>=	بزرگتر و مساوی	۲
	==	تساوی	۲
	!=	عدم تساوی	۲
	===	تساوی نوع حروف	۲
	!==	عدم تساوی نوع حروف	۲
	~	معکوس بیتی	۱
	&	و بیتی	۲
	|	یا بیتی	۲
	^	یا اختصاصی بیتی	۲
	^~ یا |~^	معکوس یا اختصاصی بیتی	۲

نوع عملگر	سمبل عملگر	عملیات	عملوند ها
عملگر های حسابی و منطقی	*	ضرب	۲
	/	تقسیم	۲
	+	جمع	۲
	-	تفریق	۲
	%	باقی مانده	۲
	!	معکوس منطقی	۱
	&&	و منطقی	۲
	||	یا منطقی	۲
عملگر های الحاق و تکرار	{ }	الحاق	نا محدود
	{ { } }	تکرار	نا محدود
عملگر های شیفت	>>	شیفت راست	۲
	<<	شیفت چپ	۲

در مورد عملگر های تساوی[2] باید به موارد زیر توجه داشت :

- نتیجه a==b برابر 0 یا 1 یا x است. اگر یک بیت از یکی از عملوند ها x یا z باشند نتیجه x می شود.
- نتیجه a!=b برابر 0 یا 1 یا x است. اگر یک بیت از یکی از عملوند ها ها x یا z باشد نتیجه x می شود.
- نتیجه a===b برابر 0 یا 1 است. a و b بیت به بیت با هم مقایسه می شوند.

[1] Relational
[2] Equality

مهندس یار

WWW.MOHANDESYAR.COM

- نتیجه a!==b برابر 0 یا 1 است. a و b بیت به بیت باهم مقایسه می شوند.

در مورد عملگر های الحاق[1] باید به موارد زیر توجه داشت :

- عملوند ها حتما باید اعداد اندازه دار باشند تا Verilog قادر به محاسبه اندازه نتیجه باشد.
- عملوند ها می توانند net ، reg و یا برداری از آنها و یا اعداد اندازه دار باشند.

اپراتور تکرار[2] مشخص می کند که چندبار عدد داخل { } باید تکرار شود .

در مورد عملگر های شرطی باید به موارد زیر توجه داشت :

- قالب استفاده از عملگر شرطی به صورت زیر است

```
<Condition> ? <True-Statement>  : <False-Statement>;
```

- عبارت شرط مورد ارزیابی قرار می گیرد ، اگر شرط برقرار باشد عبارت برقراری شرط و اگر شرط برقرار نباشد عبارت عدم برقراری شرط اجرا می شود.

مثال ۱۶ : در این مثال نحوه استفاده از عملوند ها و عملگر ها را خواهید آموخت. (لازم به ذکر است این کد ها فقط جهت آموزش نحوه عملکرد عملگر ها می باشند)

[1] Concatenation
[2] Replication

مهندس یار

WWW.MOHANDESYAR.COM


```
//------- Arithmetic Operations --------//
A = 4'b0011; B = 4'b0100; D = 6; E = 4;

A*B     // Evaluated to 4’b1100
D/E     // Evaluated to 1
A+B     // Evaluated to 4’b0111

in1 = 4'b101x; in2 = 4'b1010;
sum = in1 + in2; // Evaluated to 4'x

//------- Logic Operations --------//
A = 3; B = 0;
A && B // Evaluated to 0 ( False )
A || B // Evaluated to 1 ( True )
!A     // Evaluated to 0 ( False )

A = 2'bx0; B = 2'b10;
A && B // Evaluated to x ( Unknown )

//------- Concatenation & Replication --------//
A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;
y = { B, C }; // Result y is 4’b0010
y = { A, B ,C, D, 3'b001 }; // Result y is 11’b1_00_10_110_001


//------- Conditional Operations --------//
// Models functionality of a tri state buffer

assign addr_bus = drive_enable ? addr_out : 32'bz;

//------- Equality Operations --------//
A = 3; B = 3; X = 4'b1010; Y = 4'b1101; Z = 4'b1xzz; M = 4'b1xzz; N = 4'b1xxx;

A == B // Result is logical 0
A != B // Result is logical 1
X == Z // Result is x
Z === M       // Result is logical 1
Z === N       // Result is logical 0
M !== M       // Result is logical 1
```

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۱۷ : در مثال زیر توصیف یک مالتی پلکسر ۴ به ۱ ، جمع کننده چهار بیتی و یک مقایسه گر تک بیت قابل توسعه را مشاهده می کنید.

```
module mux4_to_1 ( out, i0, i1, i2, i3, s1, s0 );
    output out;
    input i0, i1, i2, i3, s1, s0;

    assign out = s1 ? ( s0 ? i3 : i2 ) : ( s0 ? i1 : i0 );

endmodule


module fulladd4 ( sum, c_out, a, b, c_in );
    output [3:0] sum;
    output c_out;
    input [3:0] a, b;
    input c_in;

    assign { c_out, sum } = a + b + c_in; // Concatenating assignment

endmodule


module Comparator ( a_gt_b, a_eq_b, a_lt_b, a, b, gt, eq, lt );
    output a_gt_b, a_eq_b, a_lt_b;
    input a, b, gt, eq, lt;

    assign a_gt_b = ( a & gt ) | ( ~b & gt ) | ( a & ~b );
    assign a_eq_b = ( a & b & eq ) | ( ~a & ~b & eq );
    assign a_lt_b = ( ~a & lt ) | ( b & lt ) | ( ~a & b );

endmodule
```

مهندس یار

WWW.MOHANDESYAR.COM

۹) مدل سازی رفتاری در Verilog

در Verilog می توان سخت افزار را در سه سطح کلی توصیف کرد ، این سطوح عبارتند از :

- سطح توصیف رفتاری : مدل سازی سطح بالا که در آن رفتار منطقی سیستم مدل سازی می شود.
- مدل سازی ثباتی : در این سطح سخت افزار به وسیله ثبات ها مدل می شود.
- مدل سازی پایه ای : در این حالت سخت افزار در دو سطح گیت و ثباتی توصیف می شود.

• بلوک های ساخت یافته[1]

کد های توصیف رفتاری می بایست درون این بلوک ها قرار بگیرند ، البته استثناعاتی نیز وجود دارد که بعدا به آن اشاره می شود. دو نوع بلوک ساخت یافته عبارتند از بلوک های Initial و Always . بلوک های ساخت یافته دارای ساختار ترتیبی می باشند. درون یک ماجول می توان چندین بلوک initial و always تعریف نمود.

- بلوک هایInitial فقط یک بار و در زمان صفر اجرا می گردند.
- بلوک های Always همیشه و بطور مداوم در حال اجرا می گردند.

بلوک های ساخت یافته فقط توانایی تعیین کردن مقادیر ثبات ها را دارند و نمی توانند مقداری را به Net ها (داده های wire) نسبت دهند. اما شما می توانید درون بلوک های ساخت یافته، یک ثبات را با مقادیر یک ثبات، یک Net و یا یک مقدار ثابت مقدار دهی کنید.

[1] Procedural Blocks

مهندس یار

WWW.MOHANDESYAR.COM

اگر چندین بلوک initial داشته باشیم ، همه بلوک ها در زمان صفر با هم و بطور جداگانه اجرا می شوند و هر بلوک بطور مستقل و از سایر بلوک ها خاتمه می یابد.

در صورتی که بخواهیم درون یک بلوک ساخت یافته چندین دستور داشته باشیم باید از دستورات begin و end و یا دستورات fork و join استفاده نماییم. درصورتی که از دستورات begin و end استفاده کنیم دستورات درون بلوک یکی پس از دیگری اجرا می گردند. در صورتی که از دستورات fork و join استفاده نماییم تمام دستورات درون بلوک همزمان با یکدیگر اجرا می شوند.

مثال ۱۸ : در این مثال نحوه استفاده از بلوک های ساخت یافته را مشاهده خواهیم کرد. در این مثال فرض کنید زمان صفر کردن ثبات های clk ، reset و data ۳ واحد زمان و زمان صفر کردن ثبات enable ۵ واحد زمان باشد. جلوی هر دستور زمان اتمام انجام آن بر حسب واحد زمان نوشته شده است تا تفاوت دستورات begin و fork نمایان شود.

```
initial
    begin
        clk = 0;          // Execute finish at #3 clk
        reset = 0;        // Execute finish at #6 clk
        enable = 0;       // Execute finish at #11 clk
        data = 0;         // Execute finish at #14 clk
    end

initial
    fork
        clk = 0;          // Execute finish at #3 clk
        reset = 0;        // Execute finish at #3 clk
        enable = 0;       // Execute finish at #5 clk
        data =0;          // Execute finish at #3 clk
    join
```

مهندس یار

WWW.MOHANDESYAR.COM

- **بلوک های شرطی**

Verilog نیز مانند دیگر زبان های برنامه نویسی دارای بلوک های شرطی می باشد. قالب کلی دستورات بلوک شرطی به صورت زیر می باشد :

```
if(Condition)
      Statement which executed when Condition is True
else if(Second Condition)
      Statement Which executed when Second Condition is True
else
      Statement Which executed when None of these Conditions are True
```

عبارات شرطی else و else if اختیاری می باشند. در یک بلوک شرطی می توان به دفعات از عبارت else if استفاده نمود.

مثال ۱۹ : در این مثال یک شمارنده دو جهته بسیار ساده را برسی مدل می کنیم.

```
if (reset == 1'b0)
    counter = 4'b0000;
else if (enable == 1'b1 && up_en == 1'b1)
    counter = counter + 1'b1;
else if (enable == 1'b1 && down_en == 1'b1);
    counter = counter - 1'b0;
else
    counter = counter;
```

- **دستور Case**

کارکرد این دستور دقیقا همانند دیگر زبان های برنامه نویسی می باشد. نحوه استفاده از این دستور را در خلال مثال های بعدی نمایش می دهیم.

مهندس یار

WWW.MOHANDESYAR.COM

- **کنترل زمان در بلوک های ساخت یافته**

در حالت کلی بلوک های ساخت یافته در زمان صفر شروع به کار می کنند. بعضی وقت ها نیاز به فعال سازی بلوک در زمان های خاصی احساس می شود. در Verilog می توان یک بلوک ساخت یافته را طوری تعریف نمود تا در لبه یک پالس یا ترکیبی از پالس های مختلف مشخص فعال شود.

- **کنترل زمان مبنی بر رویداد[1]**

یک رویداد به معنای تغییر یک ثبات یا یک Net است. سه نوع کنترل زمان مبتنی بر رویداد را در ادامه برسی خواهیم کرد.

کنترل رویداد با قاعده : علامت @ برای مشخص کردن کنترل رویداد استفاده می شود. دستورات می توانند با تغییر مقدار یک سیگنال با لبه بالارونده یا پایین رونده یک سسیگنال اجرا شوند. لبه بالا رونده و پایین رونده در جدول زیر مشخص شده اند.

لبه	مقدار قبلی	مقدار جدید
بالارونده	z	1
	x	1
	0	1,x,z
پایین رونده	z	0
	x	0
	1	0,x,z

مثال ۲۰ : در این مثال با نحوه کنترل یک بلوک در یک رویداد با قاعده را برسی می کنیم. عبارت اول در هر لبه ، عبارت دوم در بله بالا رونده ، عبارت سوم در لبه پایین رونده و عبارت چهارم در لبه بالا رونده اجرا خواهند شد.

[1] Event

مهندس یار

WWW.MOHANDESYAR.COM


```
@(clock) q = d;          // Triggered with any change in clock
@(posedge clock) q = d; // Triggered positive edge of clock
@(negedge clock) q = d; // Triggered negative edge of clock
q = @(posedge clock) d; // d is evaluated immediately and assigned to q at positive edge of clock
```

کنترل رویداد با نام : در Verilog می توان یک رویداد را تعریف نمود و در مواقع لزوم آن را تحریک[1] نماییم. تعریف این رویداد با کلید واژه event و تحریک کردن آن با -> انجام می شود.

کنترل چند رویداد : گاهی اوقات چند سیگنال داریم که تغییر در یکی از آنها سبب تریگر شدن اجرای یک مجموعه از دستورات می شود. این امر توسط or کردن رویداد ها یا سیگنال ها انجام می شود. لیست این رویداد ها یا سیگنال ها به لیست حساسیت[2] مشهور است.

مثال ۲۱ : در این مثال با نحوه استفاده از رویداد ها با نام و کنترل با چند رویداد آشنا می شوید.

```
// ----- Event Control by Name ------
event rec_data; // Defining Event

always @(posedge clock)                        // Triggering at Positive Edge
    begin
        if( last_data_packet ) ->rec_data;     // Triggering defined event
    end

always @(rec_data)
    data_buf = { data[0] , data[1], data[2], data[3] };

// ----- Event Control by Multiply Signals ------
always @(posedge clock or reset)
    begin
    if( reset ) q = 0;
    else q = d;
end
```

[1] Trigger
[2] Sensitivity List

مهندس یار

WWW.MOHANDESYAR.COM

۱۰) طراحی مدار در سطح رفتاری

- **طراحی مدار های ترکیبی در سطح رفتاری :**

برای طراحی مدار های ترکیبی در سطح رفتاری، باید تمام ورودی های مدار را در لیست حساس بدنه always ذکر کرد. به هنگام توصیف مدار باید توجه داشت که تمام شرط های if باید دارای else باشند تا از ایجاد مدار ترتیبی جلوگیری شود.

مثال ۲۲ : در مثال زیر توصیف یک مالتی پلکسر ۴ به ۱ را خواهید دید.

```
// Behavioral model of a 4-to-1 multiplexer
module mux4_to_1 ( out, i0, i1, i2, i3, s1, s0 );
    output out;
    reg out;
    input i0, i1, i2, i3, s1, s0;
    always @( i0 or i1 or i2 or i3 or s1 or s0 )
        begin
            case { s1, s0 }
                2'b00 : out = i0;
                2'b01 : out = i1;
                2'b10 : out = i2;
                2'b11 : out = i3;
                defualt : out = 1'bx;
            endcase
        end
endmodule
```

- **طراحی مدار های ترتیبی در سطح رفتاری**

طراحی این مدار ها با ذکر چند مثال بیان می کنیم.

مهندس یار

WWW.MOHANDESYAR.COM

مثال ۲۳ : یک فلیپ فلاپ نوع D حساس به لبه بالا رونده با Reset سنکرون طراحی کنید.

```
module d_ff ( d, clk, rst, q );
    input d, clk, rst;
    output q;
    reg q;
    always @( posedge clk )
        begin
        if (rst)
            q = 1'b0;
        else
            q = d;
    end
endmodule
```

در این مثال d ورودی داده ، clk پالس ساعت و q خروجی فلیپ فلاپ می باشد. همانطور که مشاهده می شود پس یک پالس ساعت clk بلوک always اجرا می شود و اگر پالس ریست ۱ باشد خروجی برابر صفر خواهد شد و اگر ۱ نباشد خروجی برابر ورودی خواهد شد.

مثال ۲۴ : یک فلیپ فلاپ نوع D حساس به لبه بالا رونده با Reset آسکنرون طراحی کنید.

```
module d_ff ( d, clk, rst, q );
    input d, clk, rst;
    output q;
    reg q;
    always @( posedge clk or posedge rst)
        begin
            if (rst)
                q = 1'b0;
            else
                q = d;
        end
endmodule
```

همانطور که مشاهده می شود در شمارنده با ریست آسنکرون بلوک always در لبه بالا رونده پالس ساعت و یا پالس ریست اجرا می شود که اگر ریست ۱ باشد خروجی q صفر خواهد شد و اگر ریست ۱ نباشد به معنای پالس ساعت برای فلیپ فلاپ می باشد که q را برابر ورودی یعنی d قرار می دهد.

مهندس یار

WWW.MOHANDESYAR.COM

یک شمارنده ۴ بیتی دو جهته با ورودی بارگذاری سنکرون و ریست آسنکرون طراحی کنید که با لبه پایین رونده پالس clk کار کند.

```
module counter ( clk, ld, rst, u_d, d_in, q );
    input clk, ld, rst, u_d;
    input [3:0] d_in;
    output [3:0] q;
    reg [3:0] q;

    always @( negedge clk or posedge rst)
        begin
            if (rst)
                q = 4'b0000;
            else if( ld )
                q = d_in;
            else if( u_d )
                q = q + 1;
            else
                q = q - 1;
    end
endmodule
```

مثال ۲۵ : یک جمع کننده / تفریق کننده ۸ بیتی به روش توصیف رفتاری را طراحی کنید.

در این جمع کننده a و b ورودی های برداری ۸ بیتی ، oper عملیات جمع یا تفریق و res خروجی می باشد. توجه داشته باشید در این کد از سریز صرف نظر شده است.

```
module addsub(a, b, oper, res);
    input    oper;
    input  [7:0] a;
    input  [7:0] b;
    output [7:0] res;
    reg    [7:0] res;
    always @(a or b or oper)
    begin
        if (oper == 1'b0)
            res = a + b;
        else
            res = a - b;
        end
endmodule
```

مهندس یار

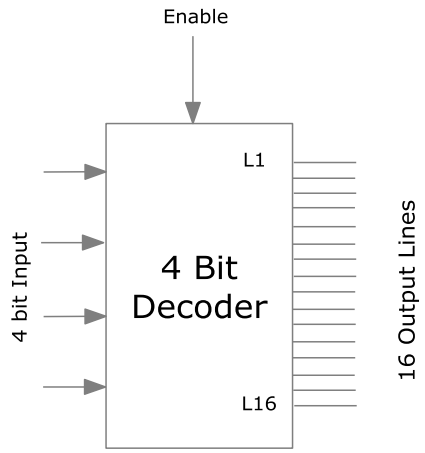
WWW.MOHANDESYAR.COM

۱۱) مثال ها

در این بخش چندین مثال را با توضیحات برسی خواهیم کرد.

- **دیکودر چهار بیتی**

در این مثال سخت افزار یک دیکودر چهاربیتی را به دو صورت توصیف می کنیم . در ابتدا این دیکودر را با دستور case و در ادامه آن را با دستور assign پیاده سازی خواهیم کرد.



شکل ۱۲. بلوک دیاگرام یک دیکودر ۴ بیتی با پایه فعال ساز

توصیف سخت افزاری این دیکودر را به دو روش برسی می کنیم :

مهندس یار

WWW.MOHANDESYAR.COM


```
module decoder_using_case ( binary_in, decoder_out, enable);
    input [3:0] binary_in ;
    input  enable;
    output [15:0] decoder_out;

    reg [15:0] decoder_out;

    always @ (enable or binary_in) // Will be Triggered if binary_in or enable changed
    begin
        decoder_out = 0;
        if (enable) begin
            case (binary_in)
                4'h0 : decoder_out = 16'h0001;
                4'h1 : decoder_out = 16'h0002;
                4'h2 : decoder_out = 16'h0004;
                4'h3 : decoder_out = 16'h0008;
                4'h4 : decoder_out = 16'h0010;
                4'h5 : decoder_out = 16'h0020;
                4'h6 : decoder_out = 16'h0040;
                4'h7 : decoder_out = 16'h0080;
                4'h8 : decoder_out = 16'h0100;
                4'h9 : decoder_out = 16'h0200;
                4'hA : decoder_out = 16'h0400;
                4'hB : decoder_out = 16'h0800;
                4'hC : decoder_out = 16'h1000;
                4'hD : decoder_out = 16'h2000;
                4'hE : decoder_out = 16'h4000;
                4'hF : decoder_out = 16'h8000;
            endcase
        end
    end
endmodule


module decoder_using_case ( binary_in, decoder_out, enable);
    input [3:0] binary_in  ;
    input  enable ;
    output [15:0] decoder_out ;

    wire [15:0] decoder_out ;

    assign decoder_out = (enable) ? (1 << binary_in) : 16'b0 ;

endmodule
```

مهندس یار

WWW.MOHANDESYAR.COM

- **مقسم فرکانس**

در این مثال یک مقسم فرکانس فرکانس ½ را برسی خواهیم کرد. اساس کار یک مقسم فرکانس مانند یک شمارنده می باشد. اما در این مثال برای تقسیم فرکانس به دو فقط کافی است در هر لبه بالارونده پالس ساعت پالس خروجی را وارون نماییم.

```
module clk_div_by2 (clk_in, enable,reset, clk_out);

    input clk_in, reset, enable;
    output clk_out;

    wire clk_in, enable;
    reg clk_out;

    always @ (posedge clk_in)
    begin
        if (reset)
            clk_out <= 1'b0;
        else if (enable)
            clk_out <= !clk_out ;
    end
endmodule
```

- **انکودر اولویت چهار بیتی :**

یک دیکودر و انکودر معمولی را در مثال های قبل برسی کردیم. حال نوبت به توصیف سخت افزار یک انکودر اولیت دار می رسد. در این انکودر ورودی برداری ۱۶ بیتی encoder_in مشخص کننده خروجی چهار بیتی binary_out خواهد بود.

مهندس یار

WWW.MOHANDESYAR.COM


```
module pri_encoder_using_if ( binary_out, encoder_in, enable);  // Priority Encoder
    output [3:0] binary_out;
    input    enable , [15:0] encoder_in;
    reg [3:0] binary_out ;
    always @ (enable or encoder_in)
    begin
        binary_out = 0;
        if (enable) begin
         if (encoder_in[0] == 1) begin
            binary_out = 1;
         end else if (encoder_in[1] == 1) begin
            binary_out = 2;
         end else if (encoder_in[2] == 1) begin
            binary_out = 3;
         end else if (encoder_in[3] == 1) begin
            binary_out = 4;
         end else if (encoder_in[4] == 1) begin
            binary_out = 5;
         end else if (encoder_in[5] == 1) begin
            binary_out = 6;
         end else if (encoder_in[6] == 1) begin
            binary_out = 7;
         end else if (encoder_in[7] == 1) begin
            binary_out = 8;
         end else if (encoder_in[8] == 1) begin
            binary_out = 9;
         end else if (encoder_in[9] == 1) begin
            binary_out = 10;
         end else if (encoder_in[10] == 1) begin
            binary_out = 11;
         end else if (encoder_in[11] == 1) begin
            binary_out = 12;
         end else if (encoder_in[12] == 1) begin
            binary_out = 13;
         end else if (encoder_in[13] == 1) begin
            binary_out = 14;
         end else if (encoder_in[14] == 1) begin
            binary_out = 15;
         end
        end
    end
 endmodule
```

مهندس یار

WWW.MOHANDESYAR.COM

- **شیفت رجیستر با ورودی و خروجی سریال**

در این مثال یک شیفت رجیستر با شیفت چپ ، ورودی و خروجی سریال و حساس به لبه بالا رونده ساعت را توصیف خواهیم کرد.

```
module shift (clk, si, so);
    input    clk,si;
    output    so;
    reg    [7:0] tmp;
    always @(posedge clk)
    begin
        tmp        <= tmp << 1;
        tmp[0]    <= si;
    end

    assign so = tmp[7];
endmodule
```

- **شیفت رجیستر چپ / راست ، ورودی سریال ، خروجی موازی**

در این مثال یک شیفت رجیستر سنکرون با ورودی پالس ساعت (clk) ، ورودی سریال (si) ، انتخاب گر شیفت چپ/راست (left_right) و خروجی موازی (po) را توصیف می کنیم.

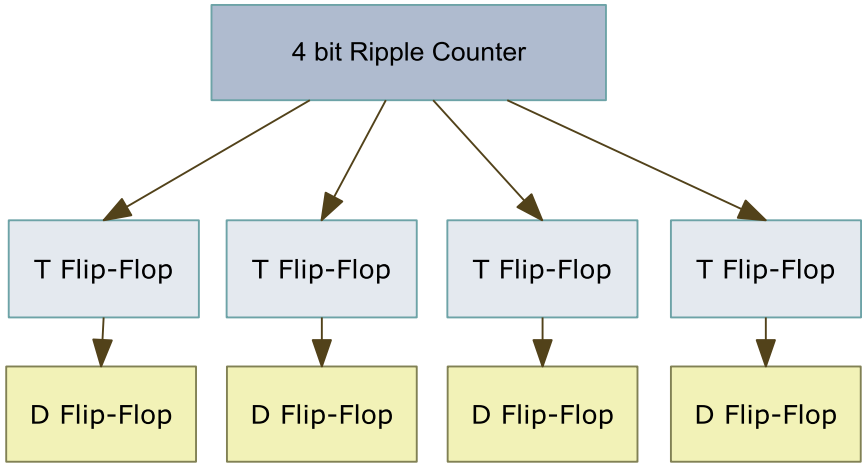
```
module shift (clk, si, left_right, po);
    input  clk, si, left_right;
    output po;
    reg    [7:0] tmp;
    always @(posedge clk)
    begin
        if(left_right == 1'b0)
            tmp <= { tmp[6:0], si };
        else
            tmp <= { si, tmp[7:1] };
    end

    assign po = tmp;
endmodule
```

مهندس یار

WWW.MOHANDESYAR.COM

- **شمارنده ٤ بیتی**

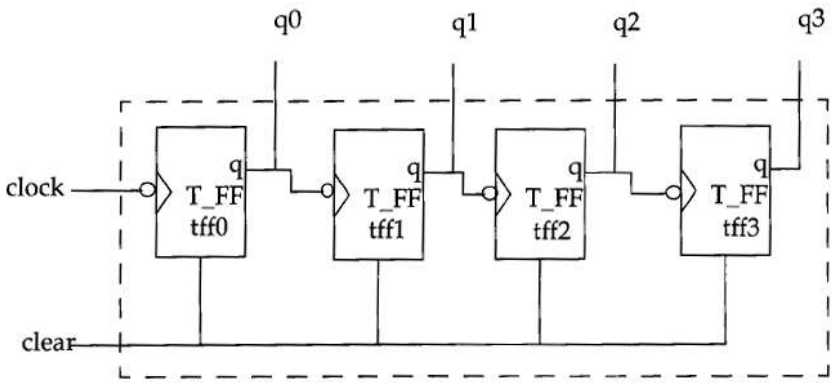
در این قسمت یک شمارنده موجی با بیت نقلی را بررسی خواهیم کرد. در این طراحی ما از متد بالا به پایین استفاده خواهیم کرد. ابتدا با توصیف بالاترین بلوک شروع خواهیم کرد. در این مثال بالاترین بلوک شمارنده موجی می باشد. یک شمارنده را می توان با روش های مختلفی مدل کرد. در یک شمارنده از تعدادی فلیپ فلاپ نوع T استفاده شده است که پس از مدل سازی بلوک شمارنده باید آنها را مدل سازی کنیم.



شکل ۱۳. طراحی بالا به پایین یک شمارنده

مهندس یار

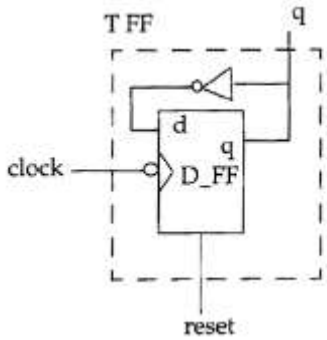
WWW.MOHANDESYAR.COM



شکل ۱۴. بلوک دیاگرام شمارنده چهار بیتی

```
module ripple_carry_counter(q, clk, reset);
    output [3:0] q;
    input clk, reset;
    T_FF    tff0(q[0], clk, reset);
    T_FF    tff1(q[1], q[0], reset);
    T_FF    tff2(q[2], q[1], reset);
    T_FF    tff3(q[3], q[2], reset);
endmodule
```

در ماجول بالا از چهار نمونه از ماجول T_FF که مدل یک فلیپ فلاپ نوع T است استفاده شده است. حال باید ماجول T_FF را توصیف کنیم.



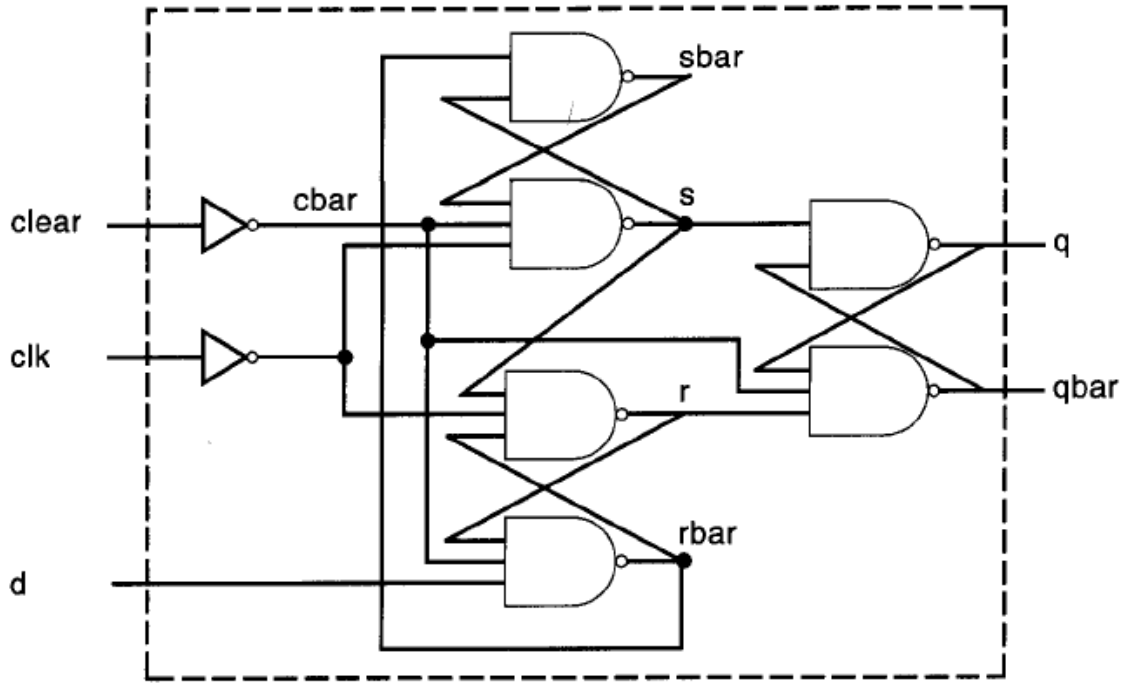
شکل ۱۵. بلوک دیاگرام یک فلیپ فلاپ نوع T

مهندس یار

WWW.MOHANDESYAR.COM


```
module T_FF(q, clk, reset);
    output q;
    input clk, reset;

    D_FF dff0(q, , ~q, clk, reset); // ~q mean's Q-Bar ( Q Not ), qbar not needed leave it unconnected
endmodule
```

در روند توصیف فلیپ فلاپ T از یک فلیپ فلاپ نوع D استفاده شده است که نوبت به توصیف آن می رسد.



شکل ۱۶. نمودار منطقی یک فلیپ فلاپ نوع D

مهندس یار

WWW.MOHANDESYAR.COM


```
module D_FF(q, q_bar, d, clk, reset);

    output q;
    input d, clk, reset;

    wire s, sbar, r, rbar, cbar;

    assign cbar = ~sbar;

    assign    sbar = ~ (rbar & s),
            s = ~(sbar & cbar & ~clk),
            r = ~(rbar & ~clk & s),
            rbar = ~(r & cbar & d);

    assign    q = ~(s & qbar),
            qbar = ~(q & r & cbar);

endmodule
```

در این مثال فلیپ فلاپ در سطح جریان داده ها مدل سازی شده است که البته می توان این فلیپ فلاپ را به روش ساده تری در سطح توصیف رفتاری مدل سازی کرد که کد آن را در ادامه مشاهده می کنید.

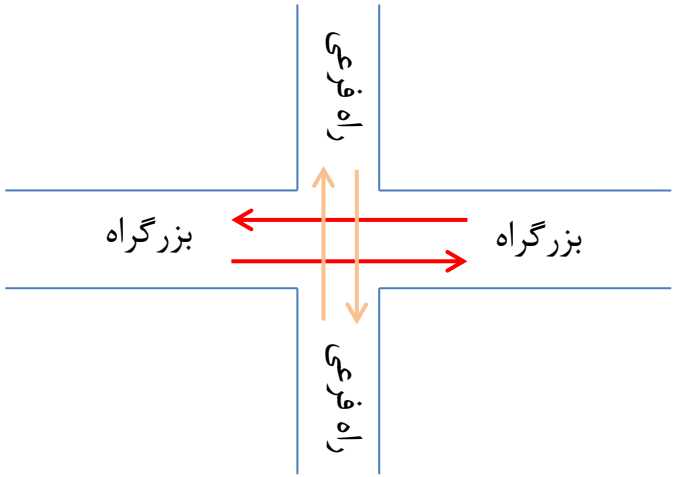
```
module D_FF(q, q_bar, d, clk, reset);

    output q;
    input d, clk, reset;
    reg q;

    always @ ( posedge reset or negedge clk)
    begin
        if(reset)
            q = 1'b0;
        else
            q = d;

        qbar = ~q;

    end

endmodule
```

مهندس یار

WWW.MOHANDESYAR.COM

- **کنترل کننده چراغ راهنما در یک بزرگراه**

در این مثال جالب ، رفتار یک چراغ راهنما را در تقاطع یک بزرگراه و یک راه فرعی برسی خواهیم کرد. ابتدا باید مدل رفتاری این چراغ را استخراج کنیم.

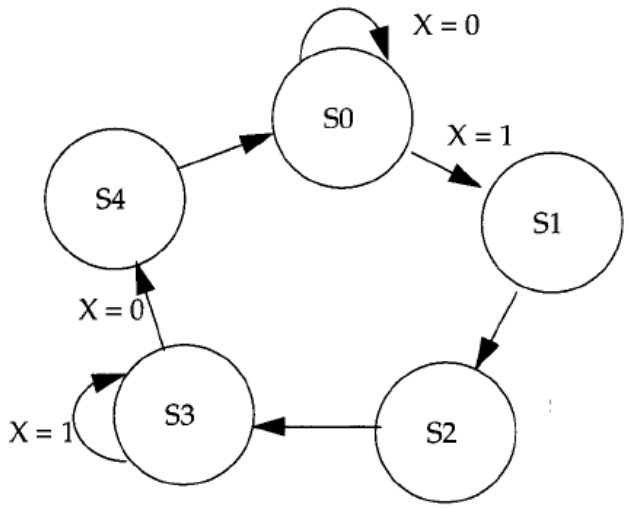


- برای راه فرعی سنسوری در نظر می گیریم تا وجود ماشین ها را در راه فرعی را مشخص کند. این سنسور سیگنال X را به کنترلر ارسال می کند. زمانی که X یک باشد به معنای وجود ماشین در راه فرعی می باشد.
- برای هر راه یک سیگنال ترافیک در نظر می گیریم که نمایانگر وضعیت چراغ در آن راه می باشد
- بزرگراه در برابر راه فرعی دارای اولویت بالاتری می باشد. چراغ بزرگراه به صورت پیشرفض سبز می باشد.
- زمانی که تعدادی ماشین در راه فرعی منتظر سبز شدن چراغ می مانند چراغ راه فرعی باید سبز شده تا به ماشین ها اجزاه عبور دهد . این چراغ باید تا زمانی که همه ماشین ها از راه فرعی عبور نکردند سبز بماند.
- زمانی که دیگر ماشینی از راه فرعی عبور نمی کند چراغ راه فرعی ابتدا زرد و سپس قرمز می شود.
- در هر یک از مراحل می بایست تاخیر هایی در نظر گرفته شود که قابل تغییر باشند.

مهندس یار

WWW.MOHANDESYAR.COM

در مرحله بعد می بایست نمودار حالت را برای این کنترلر رسم نماییم.



شکل ۱۷. نمودار حالت کنترلر چراغ راهنمایی

حالت ها به صورت زیر می باشند :

حالت	چراغ راه اصلی	چراغ راه فرعی
S0	سبز	قرمز
S1	زرد	قرمز
S2	قرمز	قرمز
S3	قرمز	سبز
S4	قرمز	زرد

مهندس یار

WWW.MOHANDESYAR.COM

بهترین راه برای مدل سازی سخت افزار این کنترلر استفاده توصیف رفتاری می باشد. در این مثال از عبارات پیش پردازنده جهت هرچه انعطاف پذیر و خوانا تر شدن کد ها استفاده می شود.

```
`define TRUE     1'b1
`define FALSE    1'b0
`define RED      2'd0
`define YELLOW   2'd1
`define GREEN    2'd2

//State definition                    // Highway    Secondary Road
`define     S0          3'd0          // G            R
`define     S1          3'd1          // Y            R
`define     S2          3'd2          // R            R
`define     S3          3'd3          // R            G
`define     S4          3'd4          // R            Y


// Delays
`define     Y2RDELAY     3     // Yellow to Red delay
`define     R2GDELAY     2     // Red to Green delay

module sig_control( hwy, secondary , X, clock, clear);

    // I/O Ports
    output [1:0] hwy, secondary ;

    // 2-bit output for 3 states of signal
    reg [1:0] hwy, secondary ;

    input X; // if 1 indicate there is car on the secondary  road
    input clock, clear;

    reg [2:0] state;
    reg [2:0] next_state;

    // Signal Controller Starts in S0 State
    initial
    begin
        state = `S0;
        next_state = `S0;
```

مهندس یار

WWW.MOHANDESYAR.COM


```
    hwy =       `GREEN;
    secondary   = `RED;
end

// state changes only at positive edge of clock
always @ (posedge clock)
    state = next_state;

// Compute Values of Highway Signal & secondary  Road Signal
always @(state) // only executed when state changes
begin
    case(state)
        `S0: begin
                hwy = `GREEN;
                secondary  = `RED;
            end
        `S1: begin
                hwy = `YELLOW;
                secondary  = `RED;
            end
        `S2: begin
                hwy = `RED;
                secondary  = `RED;
            end
        `S3: begin
                hwy = `RED;
                secondary  = `GREEN;
            end
        `S4: begin
                hwy = `RED;
                secondary  = `YELLOW;
            end
    endcase
end

always @(state or clear or X)
begin
    if(clear)
        next_state = `S0;
    else
        case(state)
            `S0: begin
                    if(x)
                        next_state = `S1;
```

مهندس یار

WWW.MOHANDESYAR.COM


```
                        else
                            next_state = `S0;
                    end
                `S1: begin
                        repeat (`Y2RDELAY) @ (posedge clock);
                        next_state = `S2;
                    end
                `S2: begin
                        repeat (`R2GDELAY) @ (posedge clock);
                        next_state = `S3;
                    end
                `S3: begin
                        if(x)
                            next_state = `S3;
                        else
                            next_state = `S4;
                    end
                `S4: begin
                        repeat (`Y2RDELAY) @ (posedge clock);
                        next_state = `S0;
                    end

                default: next_state = `S0;
            endcase;
    end
endmodule
```

در این مثال از دستور repeat جهت ایجاد تاخیر استفاده شده است. قالب کلی استفاده از این دستور به صورت زیر است :

```
repeat (<number-of-repeats> @ (<at-specified-event>)
begin
      <Statement>
end
```

استفاده از دستورات begin و end و کد Statement اختیاری می باشد. مانند مثال بالا. در این صورت این دستور فقط به تعداد پالس های ساعت مشخص شده تاخیر ایجاد می کند.

مهندس یار

WWW.MOHANDESYAR.COM

- **مولد بیت توازن[1]**

```
module parity_using_assign (
    data_in     , //  8 bit data in
    parity_out    //  1 bit parity out
);
    output  parity_out;
    input [7:0] data_in;
    wire parity_out;

    assign parity_out =  (data_in[0] ^ data_in[1]) ^
                         (data_in[2] ^ data_in[3]) ^
                         (data_in[4] ^ data_in[5]) ^
                         (data_in[6] ^ data_in[7]);

endmodule
```

[1] Parity bit Generator

مهندس یار

WWW.MOHANDESYAR.COM

۱۲) تمارین

۱) یک مبدل کد BCD به EX۳ چهاربیت را با توصیف سطح گیت طراحی کنید.

۲) یک شمارنده BCD چهار بیت بالا پایین شمار سنکرون ، با بازنشانی آسنکرون و پایه فعال ساز فعال بالا طراحی کنید.

۳) یک کنترلر کننده تکی نمایشگر هفت قسمتی کاتد مشترک طراحی کنید.

۴) یک مالتی پلکسر ۱۶ به ۱ را با طراحی مالتی پلکسر ۲ به ۱ و ۴ به ۱ و گسترش آنها طراحی کنید.

۵) یک دیکودر ۲ به ۴ طراحی کنید.

۶) یک رجیستر سنکرون ۳۲ بیتی با ورودی و خروجی موازی و بصورت مجزا و پایه بازنشانی (ریست) طراحی کنید.

۷) یک حافظه RAM با عرض کلمه ۸ بیت و تعداد ۱۰۲۴ کلمه به همراه پایه های فعال ساز ، خواندن / نوشتن و ورودی کلاک طراحی کنید.

۸) مالتی پلکسر تمرین قبل را گسترش داده و آن را بصورت برداری پیاده سازی نمایید . عرض باس مالتی پلکسر ۳۲ بیت باشد.

۹) کنترلر نمایشگر هفت قسمتی تمرین قبل را گسترش داده و توسط آن کنترلر یک نمایشگر هفت قسمتی چهار رقمی مالتی پلکس شده کاتد مشترک طراحی نمایید. (این نوع نمایشگر درواقع ۴ نمایشگر هفت قسمتی درون یک پکیج می باشد که پایه های مشترک هر نمایشگر بصورت مجزا و پایه های رقم های همه نمایشگر ها بصورت متصل به یکدیگر از پکیج بیرون آمده اند. برای راه اندازی این نمایشگر می بایست با سرعت بالایی سر های آند مشترک هریک از نمایشگر ها را سوییچ کرد و عدد مورد نظر را توسط ۷ پایه رقم ها به نمایش در آورد. برای فهم بهتر عملکرد این نمایشگر ها عبارت Multiplexed 7Segment Display را در اینترنت جستجو نمایید)

مهندس یار