آزمایشگاه معماری کامپیوتر

فهرست :

تهیه کننده : داریوش زین العابدینی

فهرست:

آشنایی با نرم افزار ALTERA MAX+PLUS II

قسمت اول

طراحی 1 bit Adder_Subtractor

نگاه کلی

در این آزمایش با نرم افزار MAX+PLUS II آشنا شده و مراحل طراحی شامل ایجاد شماتیک، کامپایل، شبیه سازی ودیباگ را در طی یک طرح ساده مشاهده خواهید کرد. در این آزمایش تأکید بر شبیه سازی جهت تشخیص خطاهای طراحی است.

متنی که با • شروع می‌شود دستورات قدم به قدم برای انجام آزمایش می‌باشد و بقیهٔ متن شامل توضیحات اضافی هر مرحله است. شکلها در انتهای دستورکار آمده است.

شروع MAX+PLUS II

- برای اجرای نرم افزارMAX+PLUS II درسیستم عاملWindows ، روی آیکون مزبور double-click کنید تا صفحهٔ MAX+PLUS II Manager ظاهرشود.

ایجاد یک شماتیک

- برای خود دایرکتوری در مسیر d:\Comp_Arch_Lab\user\your name ایجاد کنید.
- **New** را از منوی **File** انتخاب کرده، **Graphics Editor file (.gdf)** را به عنوان نوع فایل مشخص کنید.

فایل جدید Untitled-x نامیده خواهدشد(x یک عدد است).

- نامی برای فایل انتخاب کرده، آنرا در دایرکتوری خود ذخیره کنید. (در دایرکتوری خودتان فایلهای [illegible] ظاهر می‌شود)

جاگذاری عناصر

برای مشاهدهٔ کتابخانهٔ عناصر:

- **Enter Symbol** را از منوی **Symbol** انتخاب کنید.

لیستی از **Symbol Libraries** موجود را مشاهده خواهید کرد.

دایرکتوری **prim** عناصر اولیه(Primitive) نظیر گیتهای منطقی را شامل می‌شود.

دایرکتوری **mf(MacroFunctions)** اکثر عناصر سری 74xx را شامل می‌شود.

دایرکتوری **mega_lpm(MegaFunctions/Library of Prametrized Models)** شامل مدلهای پارامتری خاص است.

- بر روی دایرکتوری **prim** در لیست دایرکتوریها double-click کنید تا لیست عناصر موجود در آن رامشاهده کنید.
- بر روی دایرکتوری **mf** در لیست دایرکتوریها double-click کنید تا لیست عناصر موجود در آن رامشاهده کنید.
- عنصر **AND2** را در لیست عناصرکتابخانهٔ **prim** پیدا کرده وبر روی آن double-click کنید تا روی صفحه شماتیک قرارگیرد. دو عدد **AND2** دیگر به روش فوق روی صفحه قرار دهید.
- به همان روش قبل دو **XOR** و یک **OR3** به صفحه اضافه کنید.(شکل ۱)

با double-click برروی صفحه و تایپ نام عنصر آسانتر می توان آنرا جاگذاری نمود.

اتصال قطعات

با نزدیک کردن cursor ماوس به انتهای یک پین ، شکل آن به + عوض شده با کشیدن cursor ماوس از یک پین به پین دیگر یک سیم ایجاد می‌شود.

- پینهای گیتها را نظیر شکل ۱ به همدیگر وصل کنید. دقت کنید که نقاط اتصال • دیده شوند.

اگر **"Rubberbanding"** فعال باشد(آیکون سمت چپ تصویر)، با حرکت دادن یک قطعه طوری که یک پین با پین دیگر تماس پیدا کند یک اتصال ایجاد می‌شود ویا دور کردن قطعات ازهمدیگرسیم ویاباس دیده خواهد شد.

عنصر**input** به عنوان پین ورودی و**output** به عنوان خروجی مدار استفاده می‌شود.

- سه عدد **input** وسه عدد **output** جاگذاری و نظیر شکل ۱ سیم بندی کنید.

با استفاده از Copy وPaste راحتترمی‌توان عناصری را که چندین بار تکرار شده‌اند جاگذاری نمود.

با گذاشتن **Lable**(برچسب)برروی سیمها نیز می‌توان اتصالات را ایجاد کرد. چندین سیم جداگانه که **Label** یکسان دارند در واقع به هم متصلند. برای **Lable** گذاشتن روی یک سیم، بر روی آن کلیک کنید تا پررنگ شود و سپس **Lable** را تایپ کنید. **Lable** را از سیم دور نکنید چون باعث انفصال برچسب از سیم خواهد شد. برای چک کردن صحت **Lable** اگر سیم متناظر را انتخاب کنید **Lable** نیز پررنگ می‌شود.

Lable زدن به ورودیها و خروجیها

- پینهای خروجی را نظیر شکل ۱ با double-click بر روی **PIN_NAME** و تایپ نام، **Lable** بزنید.

ست کردن پروژه

در MAX+PLUS II از مفهوم **project** برای سازماندهی تمامی فایلهای مرتبط با یک طرح استفاده می‌شود.

- با انتخاب **Set Project to Current File** (File > Project menu) نام پروژه را به نام فایل خود ست کنید.

کامپایل کردن طرح

- Save & Check (File > Project menu) را انتخاب کنید تا فایلتان ذخیره شده و برای خطاها چک شود.

اگر طرحتان دارای خطا است، پیغام خطا را انتخاب کنید و دکمهٔ **LOCATE** را فشار دهید تا محل خطا را مشاهده کنید.

زمانیکه طرحتان بدون خطا چک شد برای کامپایل و شبیه سازی آماده است.

دو نوع شبیه‌سازی می‌توان بر روی طرح انجام داد:**functional** و **timing**.

در شبیه‌سازی **functional** هیچگونه تأخیر انتشار برای عناصر در نظرگرفته نمی‌شود وهمچنین تمامی گرههای طرح درون فایل **Simulator Netlist File (.snf)** ذخیره می‌شوند لذا امکان مشاهدهٔ آنها برای دیباگ وجود دارد.

شبیه‌سازی **timing** ، مناسب برای چک کردن زمانهای بحرانی، نظیرزمانهای setup وhold قبل از برنامه‌ریزی طرح درون یک قطعهٔ قابل برنامه ریزی(PLD) است.

- در حالیکه صفحهٔ کامپایل باز است، **Functional SNF Extractor** (Processing menu) انتخاب کنید. دکمهٔ **Start** را در **Compiler dialog box** فشار دهید تا پروژه کامپایل شود.

Waveform Editor (ویرایشگر شکل موج)

برای شبیه‌سازی طرح نیاز به ایجاد سیگنالهای محرک ورودی است و این کار با استفاده از **Waveform editor** انجام می‌شود.

- **Waveform Editor** (MAX+plusII menu)را انتخاب کنید.
- **Enter Nodes from SNF** (Node menu) را انتخاب کنید.

اکنون می‌توان مشخص کرد که چه گرههایی در زمان شبیه‌سازی دیده شوند.

- دکمهٔ **List** را فشار دهید تا لیست گرههای ورودی و خروجی را مشاهده کنید.
- دکمهٔ => را فشار دهید تا تمامی لیست درون **Selected Nodes & Groups** کپی شود.

برای مشاهدهٔ سیگنالهای رجیسترشده گزینهٔ **Registered** و برای گرههای خروجی مدارهای ترکیبی گزینهٔ **Combinational** باید انتخاب شود.

اکنون شکل موج سیگنالهای انتخاب شده را با مقادیر default آنها می‌توانید در **Waveform editor** مشاهده کنید(0 برای ورودیها و نامشخص X برای دیگر سیگنالها).

- **End Time**(File Menu) را انتخاب کنید و مقدار 1.0 us را به عنوان زمان شبیه سازی واردکنید.
- قسمتی از سیگنال X را توسط ماوس انتخاب و آیکون 1 (سمت چپ صفحه) را فشار دهید تا آن قسمت برابر 1 شود. با دنبال کردن روش فوق شکل موج سیگنالهای X,Y,Cin را نظیر شکل ۲ ایجاد کنید.
- Save & Simulate (File > Project menu) را انتخاب کنید. فایل شکل موج باهمان نام پروژه و پسوند **scf** ایجاد خواهد شد.

بعد از پایان Simulation شکل موج سیگنالهای خروجی باید شبیه شکل ۲ باشد. صحت عملکرد مدار را بررسی و در صورت نیاز طرح خود را دیباگ کنید تا مدار به درستی کار کند.
با انتخاب(File > Project menu) **Save&Compile&Simulate** می‌توان کامپایل وشبیه‌سازی رایکجاانجام داد.

***تعیین نوع قطعهٔ برنامه‌پذیر و شمارهٔ پینها**

بعدازاینکه طرحتان به‌درستی شبیه‌سازی شد، می‌توانید یک فایل برنامه‌ریزی ایجادوآنرا به‌یک قطعهٔ Altera PLD توسط برنامه‌ریز download کنید.
فایل **1b_fadd.gdf** را بازکرده، (Assign menu) **Device** را انتخاب کنید. MAX7000S را به عنوان Device Family و EPM7128SLC84-6 را به عنوان Device انتخاب کنید.
(Assign menu). **Pin/Location/Chip** را انتخاب و ضمن تایپ نام سیگنال ورودی یا خروجی شمارهٔ مورد نظر خود را وارد کنید. بدین ترتیب شمارهٔ پینها را کنار سیگنالهای متناظر مشاهده خواهید کرد.
(MAX+PLUSII menu) **Programmer** و سپس (File menu) **Select Programming File...** را انتخاب کنید و نام فایل **pof.** موردنظر خود را وارد کنید.

کامپایل و شبیه سازی timing

بعد از شبیه سازی **functional** و اطمینان از عملکرد طرح، می‌توان قبل از برنامه ریزی PLD شبیه سازی **timing** انجام داد تا درستی طرح از نظر زمانبندی نیز بررسی گردد.

- طرح خود را ذخیره وچک کنید.
- درحالیکه پنجرهٔ **Compiler** باز است، (Processing menu) **Timing SNF Extractor**را انتخاب کنید. سپس دکمهٔ **Start** را فشاردهید تا طرحتان برای قطعهٔEPM7128S Altera کامپایل شود.
- (File > Project menu) **Save & Simulate** را انتخاب کنید ودکمهٔ **Open SCF** را فشار دهید تا پنجرهٔ **Waveform editor** باز شود. بدین ترتیب تأخیرهای واقعی قطعه را مشاهده خواهید کرد.

ایجاد Symbol برای طرح

نرم افزار Max+Plus II امکان طراحی سلسله مراتبی (hierarchical) را داراست. طوریکه می‌توان یک **symbol** برای هرطرح ایجادکرد تا درطرحهای پیچیده‌تر به صورت یک عنصر قابل استفاده باشد.

- فایل طرح گرافیکی (**.gdf**) مدارتان را باز کنید. (File menu) **Create Default Symbol** را انتخاب کنید تا یک symbol درون فایلی با پسوند **sym.** ایجاد شود.
- (File menu) Edit Symbol را انتخاب کنید تاپنجرهٔ **Symbol editor** بازشود. با تغییر**symbol** آنرا به شکل مناسب در آورید(شکل ۳).

آرشیو پروژه

زمانیکه طرحتان کامل و از لحاظ عملکرد بررسی شد باید آنرا آرشیو کنید.

Archive (File > Project menu) را انتخاب کنید تا تمامی فایلهای مربوط بـه طرحتـان درون یـک دایرکتـوری جداگانه کپی شود. مسیر دایرکتوری را A:\your project name قرار دهید تا یک کپی از طرحتان را درون فلاپی داشته باشید.

ایجاد یک طرح سلسله مراتبی(hierarchical)

اکنون می‌خواهیم یک شماتیک top-level طراحی نماییم که از طرح قبلی به عنوان یک عنصراستفاده کند. درایـن طرح با استفاده از یک Full Adder مداری ساخته می‌شود که با توجه به سیگنالهای S1, S0 عملیات زیر را انجام دهد:

S1 S0	Y	Cin=0	Cin=1
00	0	S=A	S=A+1
01	B	S=A+B	S=A+B+1
10	/B	S=A+/B	S=A-B
11	1	S=A-1	S=A

- تمامی پنجره‌های باز نظیر **Graphics Editor, Simulator,Waveform Editor** و... را ببندید.
- یک فایل شماتیک جدید (**.gdf**) برای طرح خود ایجاد و با نام **1b_add_sub.gdf** ذخیره کنید.
- **Project** را به نام این فایل ست کنید.
- با double-click بر روی صفحه یک عدد عنصر **1b_fadd** بر روی صفحه قرار دهید.
- طرح خود را نظیر شکل۳ با قرار دادن عناصر لازم تکمیل کنید.
- پینهارا نظیر شکل۳ **Lable** بزنید.
- پروژه را به نام این فایل ست و آنرا **Save&Check** کنید.
- پروژه را **Save&Compile** کنید. شبیه سازی را **functional** انتخاب کنید.
- با اجرای **Waveform editor** شکل موج سیگنالهای ورودی را نظیر شکل٤ ایجاد کنید.
- پروژه را **Save&Simulate** کنید. شکل موج سیگنالهای خروجی باید نظیر شکل ٤ باشد.
- پس از اطمینان ازصحت عملکرد طرح یک **symbol** برای آن ایجاد و آنرا به شکل مناسب درآورید.
- پروژه را درون فلاپی آرشیو کنید.

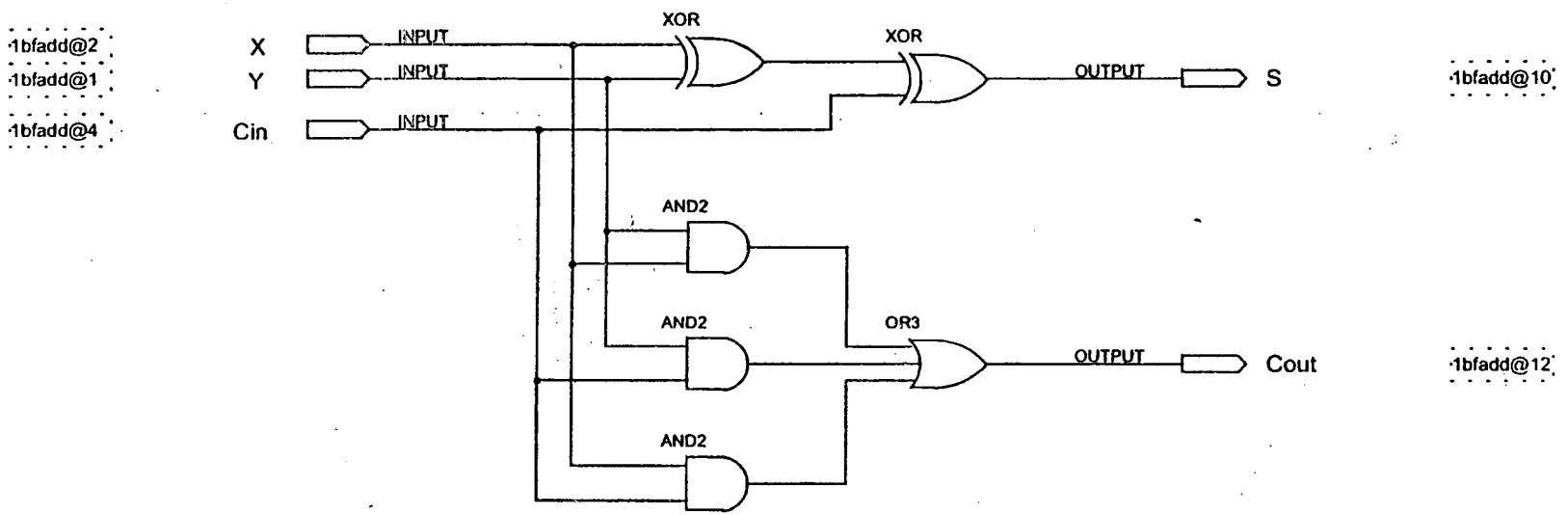


Figure 1

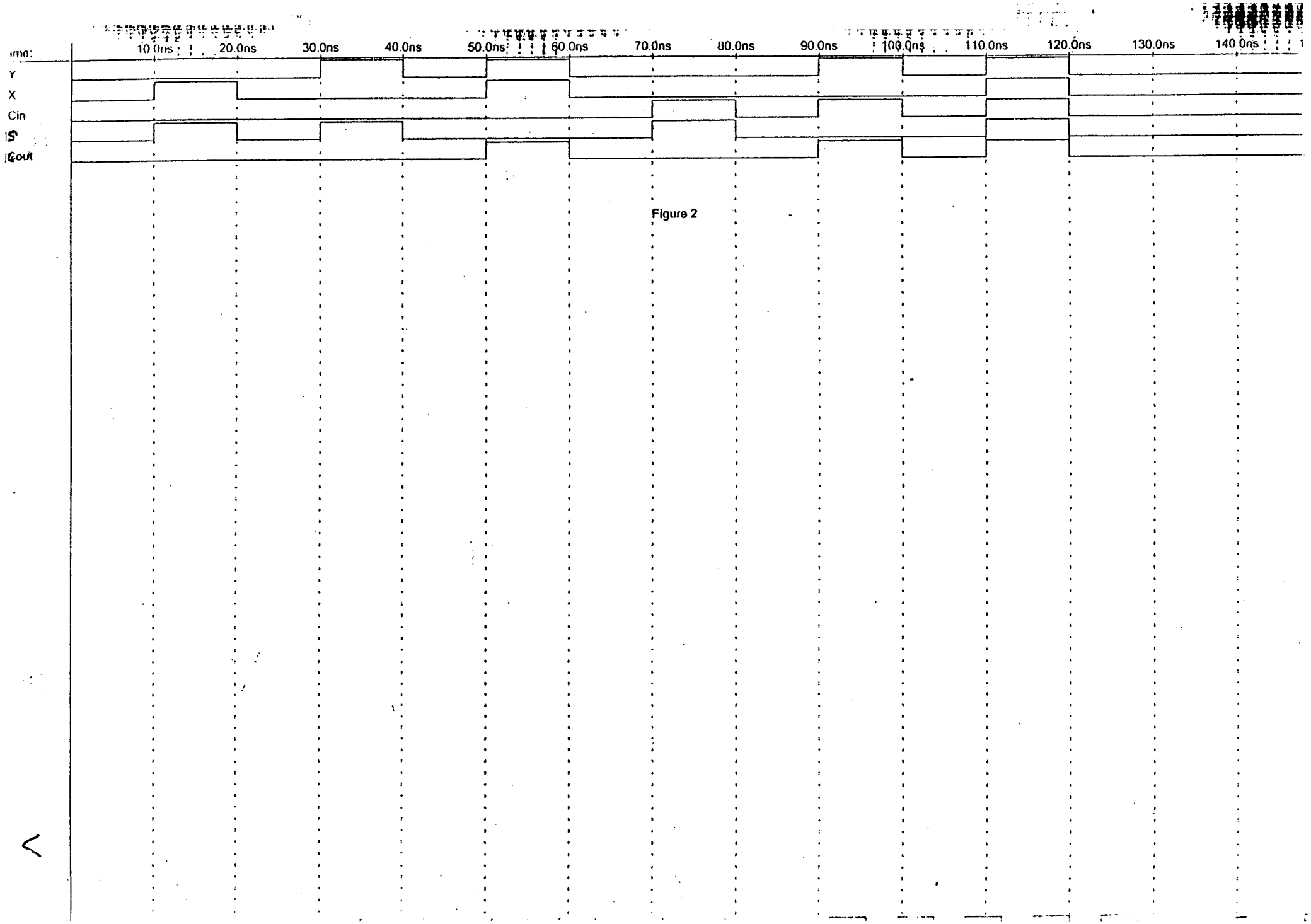


Figure 2

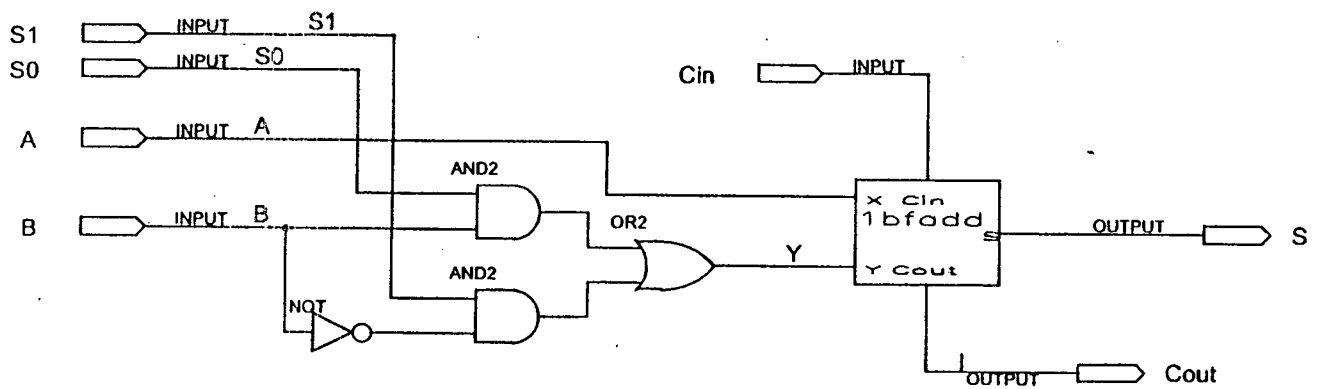


Figure 3

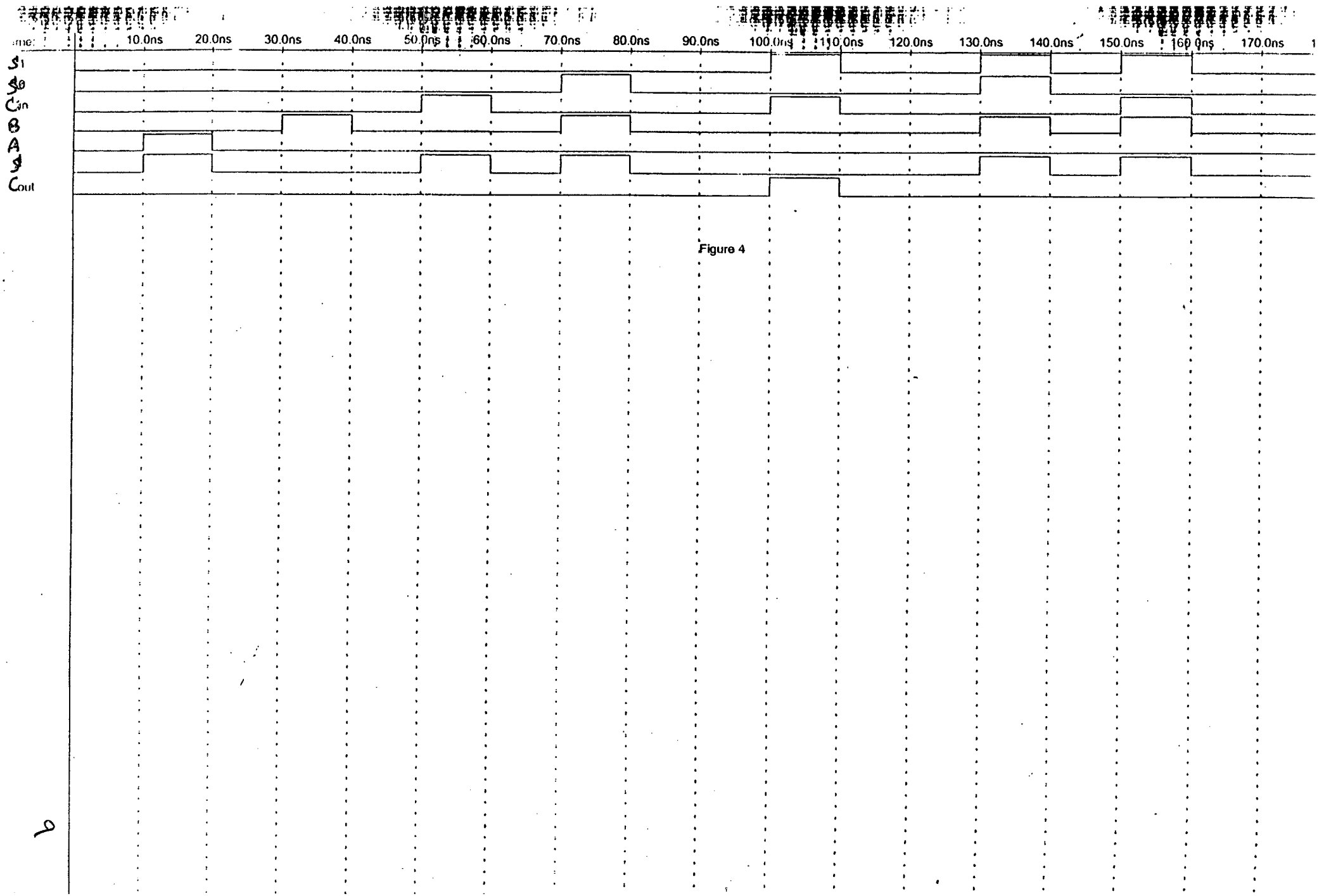


Figure 4

آشنایی با نرم افزار ALTERA MAX+PLUS II

قسمت دوم

طراحی ALU (Arithmatic Logic Unit)

نگاه کلی

دراین آزمایش بامفاهیم پیشرفته‌تری از نرم‌افزار MAX+PLUS II آشنا می‌شوید. این مفاهیم شامل ایجاد Symbol برای طرح و ویرایش آن جهت استفاده درطراحی سلسله مراتبی، استفاده از Bus ، آرایهٔ عناصر اولیه(primitive arrays) ،معرفی کتابخانهٔ عناصرپارامتریک LPM (Library of Parameterized Modules) نظیر مالتی پلکسر BUSMUX, LPM_MUX خواهد بود.

متنی که با • شروع می‌شود دستورات قدم به قدم برای انجام آزمایش می‌باشد و بقیهٔ متن شامل توضیحات اضافی هر مرحله است. شکلها در انتهای دستورکار آمده است.

طراحی واحد حسابی(Arithmatic Unit)

در این قسمت با استفاده ازطرح **1b_add_sub.gdf** آزمایش اول یک واحد حسابی 16 بیتی ایجاد خواهیم کرد که دارای جدول عملیاتی زیر است:

Sel1 Sel0	Cin=0	Cin=1
00	S=A	S=A+1
01	S=A+B	S=A+B+1
10	S=A+/B	S=A-B
11	S=A-1	S=A

- یک فایل شماتیک با نام **arith_unit.gdf** در دایرکتوری خود درمسیر d:\Comp_Arch_Lab\user\ ایجاد کنید.
- یک عدد عنصر **1b_add_sub** بر روی صفحه قرار دهید و ورودیها و خروجیهای آنرا نظیر شکل ۱ Lable بزنید. با استفاده از **Copy&Paste** 16 کپی از این عنصرایجاد و نظیر شکل ۱ به هم وصل کنید.

دقت کنید که برچسبهای(Lable) سیمها کاملاً نزدیک سیم متناظر قرار گیرد. بدین ترتیب تمامی سیمهایی که دارای Lable یکسان هستند به همدیگر متصل‌خواهندبود.

- ۵ عدد **input** و۳ عدد **output** بر روی صفحه قرار دهید ونظیر شکل ۱ Lable بزنید.

می‌دانیم که هرگاه C15, Cout متفاوت باشند overflow اتفاق می افتد: overflow=C15⊕Cout

- یک عدد **xor** وارد و نظیر شکل۱ سیم بندی کنید.
- دو عدد از **input**ها را A[15..0] وB[15..0] و یکی از **output**ها را S[15..0] برچسب بزنید. یک قطعه سیم به هرکدام از آنها متصل و با استفاده از منوی **toolbar** ضخامت آنرا برابر ▬▬ انتخاب کنید تا این سیم به عنوان باس(Bus) تعریف شود.
- پروژه را به نام این فایل ست و آنرا **Save&Check** کنید.

- پروژه را **Save&Compile** کنید. شبیه سازی را **functional** انتخاب کنید.
- با اجرای **Waveform editor** شکل موج سیگنالهای ورودی را نظیر شکل ۲ ایجاد کنید.
- پروژه را **Save&Simulate** کنید. شکل موج سیگنالهای خروجی باید نظیر شکل ۲ باشد.
- پس از اطمینان ازصحت عملکرد طرح یک **symbol** برای آن ایجاد و آنرا به صورت شکل ۵ درآورید.
- پروژه را درون فلاپی آرشیو کنید.

طراحی واحد منطقی(Logic Unit)

در این قسمت با استفاده از عناصر اولیه **(primitive arrays)** و عنصر پارامتری **LPM_MUX** یک واحد منطقی 16 بیتی خواهیم ساخت که دارای جدول عملیاتی زیر است:

S1 S0	Function
00	L=A∧B
01	L=A∨B
10	L=A⊕B
11	L=/A

- یک فایل شماتیک جدید با نام **logic_unit.gdf** در دایرکتوری خود درمسیر d:\Comp_Arch_Lab\user\ ایجاد کنید.
- عناصر **and2, or2, xor, not** را روی صفحه قرار دهید.
- دو عدد **Input** روی صفحه قرار دهید وآنها را A[0][15..0] و B[0][15..0] برچسب بزنید و نظیر شکل ۳ با استفاده از باس به گیتهای فوق وصل کنید.
- خروجیهای گیتها را به ترتیب C[0][15..0]، C[1][15..0]، C[2][15..0] و C[3][15..0] Lable بزنیدو آنها را نظیر شکل ۳ به هم متصل و باس حاصل را C[3..0][15..0] بنامید. این باس در واقع شامل 4 باس با پهنای 16 بیت خواهد بود.

استفاده ازعناصراولیه **(primitive)** به روش فوق آرایه‌ای ازعناصراولیه**(primitive arrays)** ایجاد می‌کند، لذا دیگرنیازی به‌تکراراین عناصربه تعدادپهنای باس نخواهدبود. به عنوان مثال گیت**and2** درشماتیک فوق آرایه‌ای از گیتهای**and2** است‌که به‌ترتیب C[0][0]=A[0][0]∧B[0][0] و C[0][1]=A[0][1]∧B[0][1] و...راتولید می‌کنند.

- از **mega_lpm library** یک عدد **lpm_mux** به شماتیک اضافه کنید. پنجرهٔ Edit ports/parameters بازخواهدشد.

کتابخانه **mega_lpm** شامل عناصر**macrofunction** پارامتری است که امکان مقیاس پذیری و سازگاری با تکنولوژیهای مختلف را میسرمی‌کند. این عناصردارای پارامترهای قابل تنظیم هستند تا از آنها بتوان درطرحهای مختلف استفاده کرد.

- پارامتر**LPM_SIZE** را از لیست انتخاب ومقدار آن **(parameter value)** را برابر 4 قرار دهید.

این پارامترتعداد باسهای ورودی به مالتی پلکسر را مشخص می کند که در این طرح برابر 4 می باشد.

- پارامتر**LPM_WIDTH** را از لیست انتخاب ومقدار آن **(parameter value)** را برابر 16 قرار دهید.

این پارامترعرض(تعداد بیتها) باسهای ورودی به مالتی پلکسر را مشخص می‌کند که در این طرح برابر 16 است.

- پارامترهای دیگر را بدون تغییر رهاکنید و دکمهٔ **OK** را فشار دهید.

بدین ترتیب عرض سیگنال []sel برابر $\log_2(LPM\_SIZE) = 2$ خواهد شد.

- ۲عدد **input** و ۱ عدد **output** دیگر به طرح اضافه کنید و آنها را نظیر شکل ۳ lable بزنید.

طرح نهایی شما باید شبیه شکل ۳ باشد.

- پروژه را به نام این فایل ست و آنرا **Save&Check** کنید.
- پروژه را **Save&Compile** کنید. شبیه سازی را **functional** انتخاب کنید.
- با اجرای **Waveform editor** شکل موج سیگنالهای ورودی را نظیر شکل ٤ ایجاد کنید.
- پروژه را **Save&Simulate** کنید. شکل موج سیگنالهای خروجی باید نظیر شکل٤ باشد.
- پس از اطمینان ازصحت عملکرد طرح یک **symbol** برای آن ایجاد کنید. **symbol** را ویرایش کنید تا به صورت شکل ٥ در آید.

با double-click بر روی هرپین× پنجرهٔ**enter pinstub** باز می‌شود. در قسمت **visible pinstub name** می‌توانید نام پین را به صورتی که می‌خواهید درطرحهای top-level دیده شود واردکنید.(**full pinstub name** نباید تغییر کند.)

- به روش فوق نام قابل مشاهدهٔ پینهای A[0][15..0] وB[0][15..0] را به ترتیب به A[15..0] وB[15..0] تبدیل کنید و **symbol** را ذخیره نمایید.
- پروژه را درون فلاپی آرشیو کنید.

طراحی واحدحسابی- منطقی(Arithmatic Logic Unit)

در انتها با استفاده از دو عنصر سطح پایین(down-level) ایجاد شده یعنی **arith_unit** و **logic_unit** یک واحد حسابی-منطقی 16 بیتی خواهیم ساخت که دارای جدول عملیاتی زیر است:

G_Sel[3..0]	Function
0000	G=A
0001	G=A+1
0010	G=A+B
0011	G=A+B+1
0100	G=A+/B
0101	G=A-B
0110	G=A-1
0111	G=A
100x	G=A∧B
101x	G=A∨B
110x	G=A⊕B
111x	G=/A

- یک فایل شماتیک جدید با نام **alu.gdf** ایجاد کنید.
- یک عددعنصر **arith_unit** و یک عدد **logic_unit** روی صفحه قرار دهید.
- از **mega_lpm library** یک عدد **busmux** به شماتیک اضافه کنید. پنجرهٔ **Edit ports/parameters** بازخواهدشد.
- تنها پارامتراین عنصر**WIDTH** می‌باشد. مقدار **parameter value** آنرا برابر 16 تنظیم کنید. (16 پهنای باسها می باشد.) **sel** این مالتی پلکسر باید به سیگنال(M(mode که همان G_Sel[3] است وصل شود.
- پینهای **input** و **output** و اتصالات را نظیر شکل ۵ ایجاد و Lable بزنید.

می دانیم که پرچم N (علامت) همان بیتMSB خروجی ALU یعنی G[15] است. ولی نمی‌توان G[15] را برابر N برچسب زد، چون دوسمت یک پین باید دارای Lable یکسا باشند. برای اینکاراز عنصر**wire** استفاده می‌شود.

- یک عدد عنصر **wire** وارد و نظیر شکل ۵ سیم بندی کنید.

پرچم Z(zero)(صفر) برابر $Z=\overline{G[15] \vee G[14] \vee G[13] \vee \ldots \vee G[0]}$ است. برای اینکار ازیک عنصر پارامتری با نام **lpm_or** و یک **not** استفاده می‌کنیم.

- یک عدد عنصر **lpm_or** از**mega_lpm library** به طرح اضافه کنید. و در پنجرهٔ**Edit ports/parameters** مقدار پارامتر **LPM_SIZE** را برابر 16 و **LPM_WIDTH** را برابر ۱ ست کنید.

LPM_SIZE تعداد ورودیهای به هر گیت or یا همان پهنای باس و **LPM_WIDTH** پهنای پورتهای []data و []result می‌باشد.

طرح نهایی ALU باید شبیه شکل۵ باشد.

- پروژه را به نام این فایل ست و آنرا **Save&Check** کنید.
- پروژه را **Save&Compile** کنید. شبیه سازی را **functional** انتخاب کنید.
- با استفاده از **Waveform editor** شکل موجهای ورودی را به منظور تست ALU ایجاد کرده، پروژه را **save&simulate** کنید. در صورت عملکرد نادرست طرح آنرا بررسی و اصلاح نمایید.
- برای طرح خود یک **symbol** ایجاد کرده وآنرا ویرایش کنید تا به صورت شکل ALU ارائه شده در کتاب M. Mano, "Computer System Architecture ", 1997 (درس معماری) درآید.
- پروژه نهایی را درون فلاپی آرشیو کنید.

با انتخاب Hierarchy Display(Max+Plus II menu) می‌توانید سلسله مراتب طرح خود شامل تمامی عناصر تشکیل دهنده آن را مشاهده نمایید.

گزارش کار:

۱. تحویل پروژهٔ ALU شامل فایلهای alu.gdf، alu.scf.

۲. شبیه سازی timing پروژهٔ ALU و بدست آوردن تأخیرعملیات جمع 16 بیتی به روش ripple carry.

۳. طراحی یک 16 bit adder به روش Carry Look Ahead(CLA) و شبیه سازی پروژه.

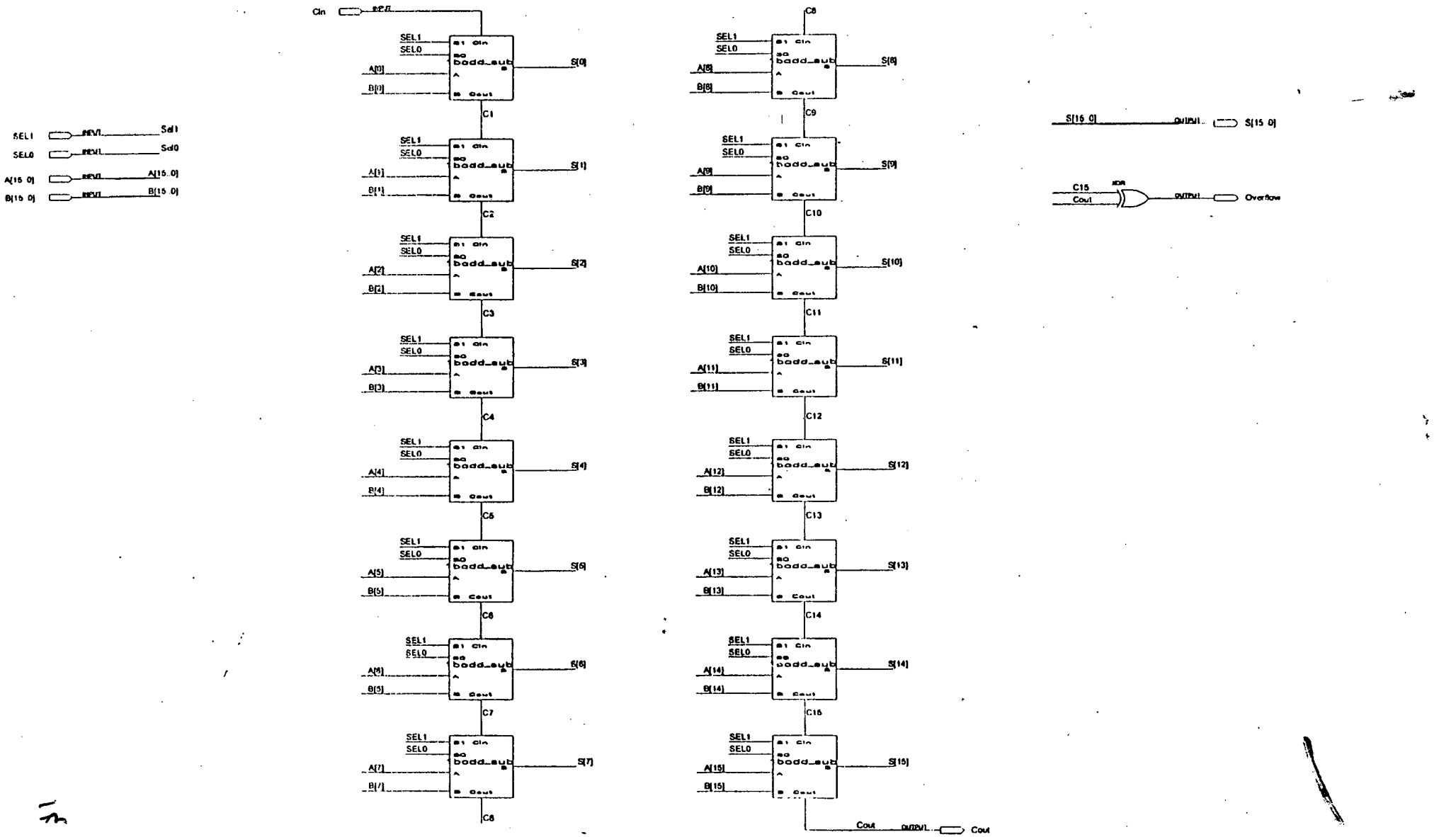


Figure 1

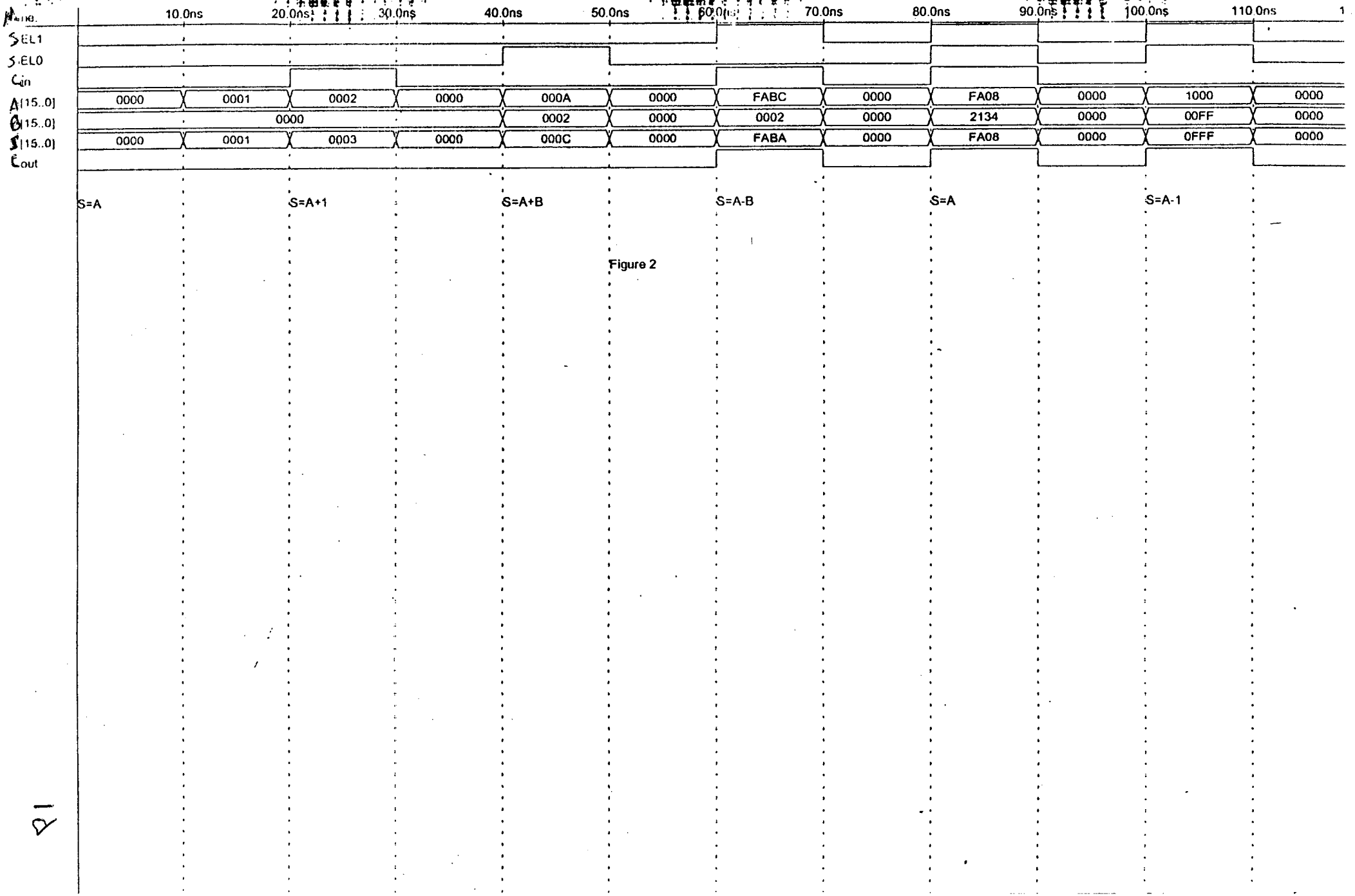


Figure 2

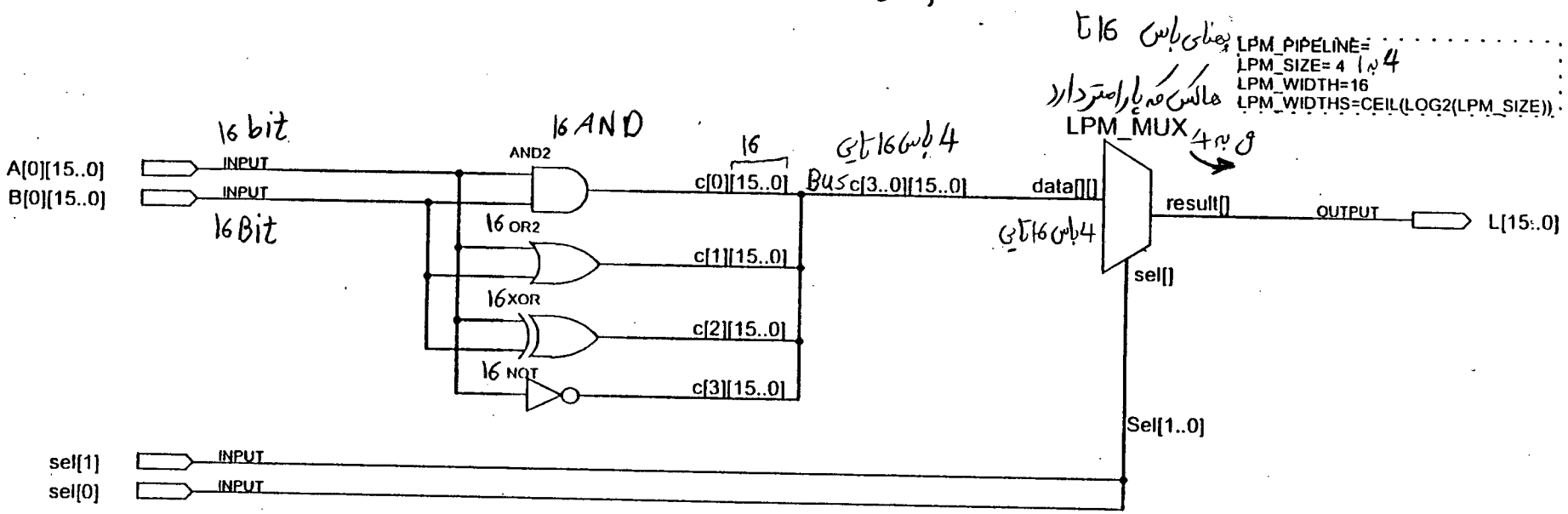


Figure 3

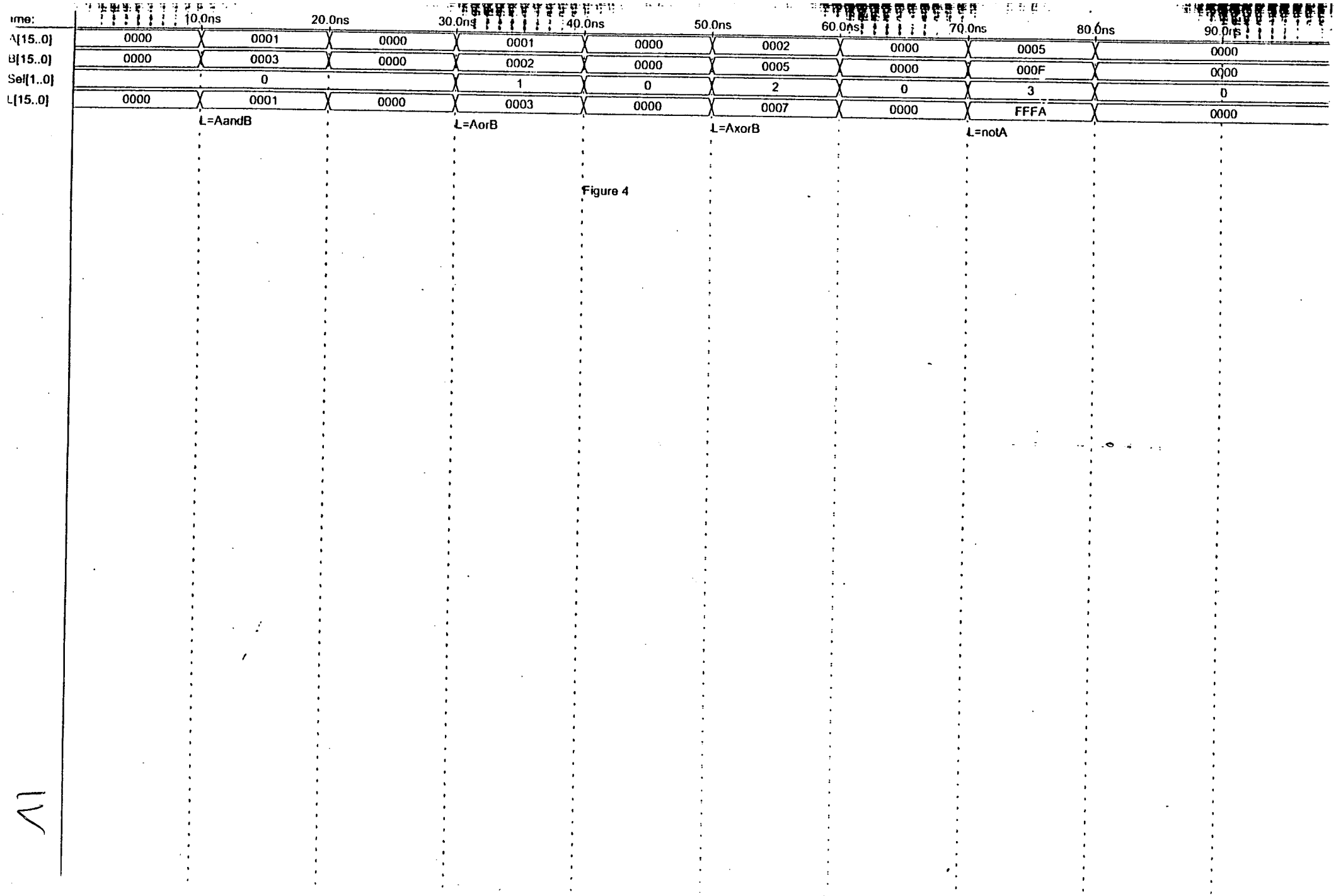


Figure 4

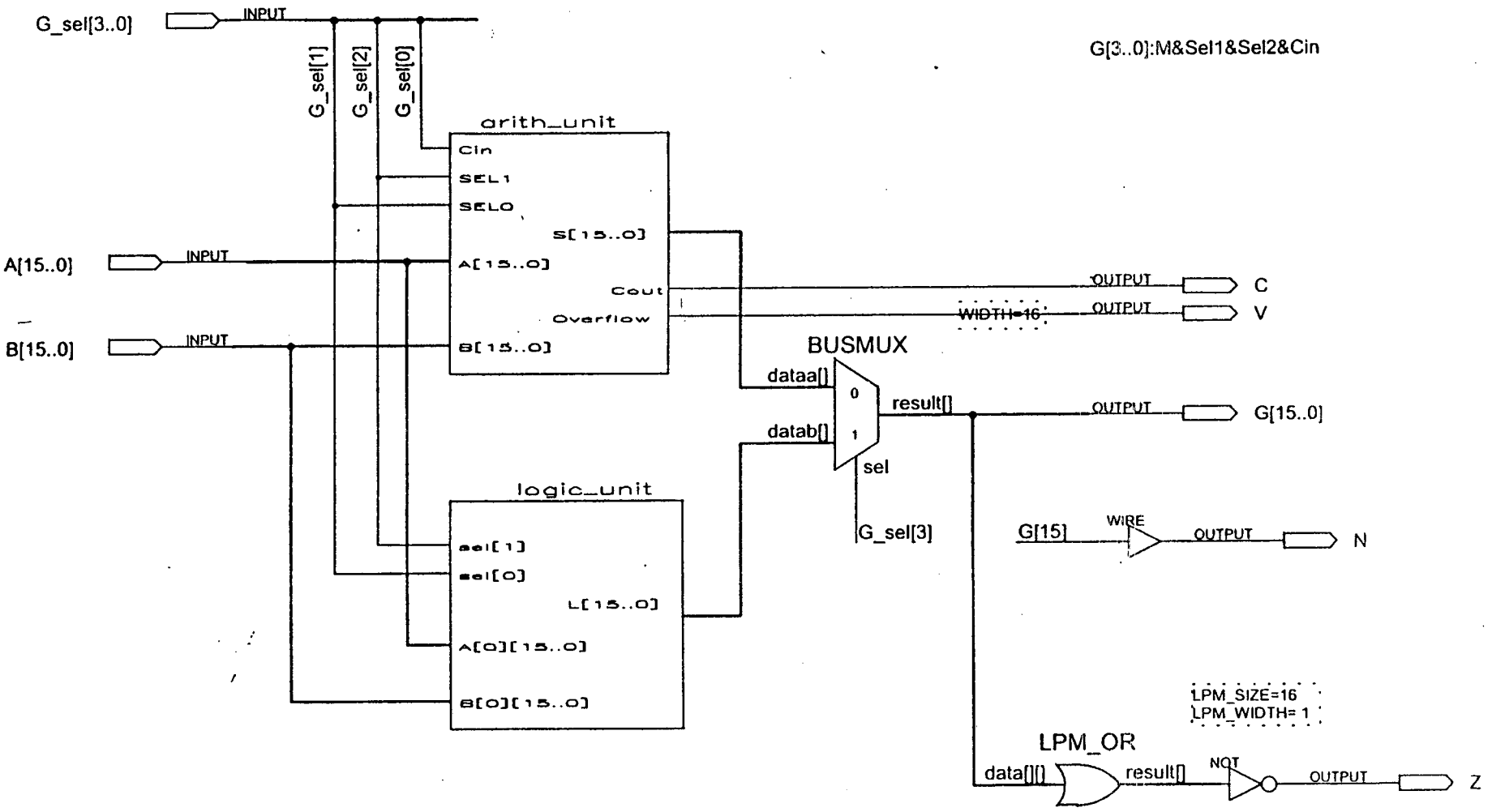


Figure 5

آشنایی با زبان توصیف سخت افزار Verilog

طراحی Shifter ، Function Unit

Register File

پروسهٔ طراحی سیستمهای دیجیتال

طراحی سیستمهای دیجیتال به صورت بالا به پایین (top-down design)با استفاده از ابزارهای طراحی به کمک کامپیوتر(CAD) شامل مراحل زیر است:

۱. *ایدهٔ طراحی* :توصیف رفتار سیستم به صورت فلوچارت یا کد رفتاری(**Behavioral**)

۲. *طراحی Data Path و Control Unit* : توصیف **RTL**، رجیسترها و باس سیستم، روال کنترل جریان داده میان رجیسترها ازطریق باس

۳. *طراحی منطقی*: استفاده از گیتهای پایه وflip flopها برای پیاده‌سازی رجیسترها و واحدهای منطقی. نتیجهٔ این مرحله از طراحی یک **netlist** است که عناصر و ارتباطات آنها را نشان می دهد.

٤. *طراحی فیزیکی*: تبدیل **netlist** مرحلهٔ قبل به لیست ترانزیستور یا **Layout**

۵. *ساخت*: استفاده از لیست ترانزیستوریا **layout** به منظوربرنامه ریزی یک قطعهٔ برنامه‌پذیر(**FPGA**) یا تولید **mask** برای ساخت IC

مدلسازی سیستمهای دیجیتال اصولاً به دومنظور شبیه‌سازی(**simulation**) و سنتز(**synthesis**) صورت می‌پذیرد.

در عمل طراح سیستم تنها معماری وتوصیف Data Path و Control را برعهده دارد وقسمت زیادی از پروسهٔ طراحی توسط ماشین انجام می‌گیرد. در پایان هرکدام از مراحل فوق طراح نتایج را بررسی و در صورت نیاز طرح رااصلاح می‌کند تا وارد مرحلهٔ بعدی شود.

ابزارهای مدلسازی شامل قلم وکاغذ، بردبورد و تجهیزات prototyping، برنامه‌های تولید شماتیک و نهایتاً زبانهای توصیف سخت افزارHDL (Hardware Description Lnguages) است. این ابزارها امکان مدلسازی سیستم را درنهایت دقت در اختیار طراح قرارمی‌دهند.

زبانهای توصیف سخت افزار به منظور شبیه‌سازی، مدلسازی، تست، طراحی و مستندسازی سیستمهای دیجیتال بکار می‌روند. نرم افزارهای موجود برای HDLها شامل شبیه ساز وابزارهای سنتز می باشند. شبیه سازجهت بررسی عملکرد سیستم و تست آن و برنامه‌های سنتز برای تولید اتوماتیک سخت افزار بکار می‌روند. این نرم‌افزارها همچنین شامل کتابخانه‌های عناصرمعمول هستند که طراح می‌تواند درطرح خود ازآنها بهره ببرد.

HDL ها در سه سطح از تجرید(**abstraction**) با نامهای **behavioral**(رفتاری)و **dataflow**(جریان داده) و**structural** (ساختاری) توانایی توصیف سیستم را دارند.

*توصیف رفتاری **(behavioral)*** مجردترین نوع توصیف سخت‌افزار است. این روش عملکرد سیستم را شبیه زبانهای برنامه‌نویسی توصیف می‌کند و هیچ اطلاعاتی دررابطه با نحوهٔ پیاده‌سازی طرح ارائه نمی‌دهد. این روش برای شبیه سازی سریع ایدهٔ طرح و بررسی عملکرد آن مناسب است.

*توصیف جریان داده **(dataflow)*** نحوهٔ انتقال داده میان رجیسترها و باس سیستم را به صورت جملات همزمان (concurrent) ارائهٔ می‌دهد. این توصیف اطلاعات جامعی از سخت افزار و تأخیرها را شامل می‌شود و برای سنتز مناسترمی‌باشد.

*توصیف ساختاری **(structural)*** پایین‌ترین سطح وشامل بیشترین جزئیات است لذا به راحتی قابل سنتز می‌باشد. این توصیف شامل لیستی از عناصر همزمان و اتصالات آنها می‌شود.

زبانهای VHDL و Verilog دو زبان استاندارد توصیف سخت افزار می‌باشند. هر کدام از این زبانها دارای syntax خاص خود است(نظیرپاسکال و C)که برای یادگیری کامل آن باید به کتابهای مرجع رجوع کرد.

نرم افزار Max+Plus II دارای کامپایلر سه زبان AHDL، VHDLوVerilog است. البته این نرم‌افزار از تمامی ویژگیهای زبان Verilog پشتیبانی نمی‌کند(نظیر **memory** ، **Initial**و..) لذا دراستفاده از این ویژگیها دچار محدودیت خواهیم بود.(برای بهره‌گیری ازتمامی ویژگیهای Verilog می‌توان از نرم افزارهای دیگر استفاده کرد.)

مقدمه‌ای بر Verilog

Verilog یک زبان توصیف سخت افزار(HDL) است که به طراح امکان توصیف سیستم در سطوح مختلفی از تجرید را می‌دهد. این سطوح شامل سطح ترانزیستور وگیت، سطح **dataflow** وتوصیف رفتاری(**behavioral**) است. در این آزمایش با بالاترین سطح، یعنی توصیف رفتاری آشنا خواهیم شد.

توصیف سلسله مراتبی سیستم دیجیتالی

Verilog یک سیستم دیجیتالی را به صورت مجموعه‌ای از **module** ها توصیف می‌کند. هرکدام از این **module**ها شامل یک سری پورتهای ورودی وخروجی و توصیف محتویات **module** است. محتویات **module** شامل توصیف ساختاری یا رفتاری آن و یا ترکیبی ازاین دو است. معمولاتوصیف سیستم دارای یک **module** سطح بالا است که از چندین **module** سطح پایینتر که خود آنها نیز ترکیبی از **module**های مختلف هستند تشکیل شده است. طراحی سلسله مراتبی یک محیط انعطاف‌پذیر برای تغییرات احتمالی در طرح ورسیدن به یک توصیف دقیقتر را ممکن می‌سازد.

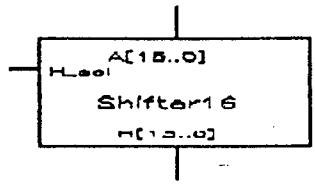
نگاه کلی

در این آزمایش مدول **Shifter** با قابلیت شیفت به چپ و راست ومدول Register File 8x16 از کتاب M.Mano, "Computer System Architecture" توسط زبان Verilog توصیف و نحوهٔ استفاده ازایـن **module** هـا در طـرح نهایی را مشاهده خواهید کرد.

طراحی 16 bit shifter

در این قسمت واحد **Shifter** درون Function Unit که یک عنصرترکیبی(combinational) است با قابلیت شیفت به چپ و راست طراحی‌خواهیم کرد:

H_Sel	Function
0	shift right
1	shift left



این واحد را شبیه آزمایشهای قبلی می‌توان با استفاده ازعناصرموجود درکتابخانه‌های نرم‌افزاریـــه صـورت شـماتیک طراحی کرد ولی ما دراین آزمایش، این module رابا استفاده اززبان Verilog درسطح رفتاری توصیف خواهیم کرد.

- بعد از اجرای نرم افزار Max+Plus II ، **Open(File Menu)** را انتخاب کنید.
- **Text Editor files** و ***.v** را به عنوان نوع فایل انتخاب کنید.
- ازلیست فایلهای درون دایرکتوری خود فایل **shifter16.v** را انتخاب کنید تـا فـایل سـورس توصیـفVerilog واحد shifter را مشاهده کنید.

توضیح محتویات فایل shifter16.v

فایل shifter16.v حاوی تعریف مدول 16 bit shifter است.

۱. تعریف module

تعریف **module** در Verilog بسیارشبیه تعریف procedure درسایر زبانها است. syntax تعریف یک **module** بـه شکل زیراست:

```
module <module_name> [(<list_of_ports>)];
<input declarations>
<output declarations>
<statements>
endmodule
```

لیست پورتها شامل اسامی پورتهای ورودی و خروجی است که **module** به کمک آنها با سایر **module**ها درمدل سخت افزاری ارتباط برقرار می‌کند.

بعد از جملهٔ تعریف **module** ،هرپورتی باید به عنوان ورودی یا خروجی معرفی شود. شکل زیر تعریف مدول shifter را نشان می‌دهد:

```
module Shifter16 ( A, H_sel, H);
input [15:0]A;
input H_sel;
output [15:0]H;

      <statements removed>
endmodule
```

نام مدول **shifter16** و دارای ۳ پورت با نامهای A, H_sel, H است که دوتای آنها ورودی ویکی خروجی است. این واحد دارای یک پورت ورودیA با پهنای 16 بیت است که با توجه به H_sel به چپ یا راست شیفت داده می‌شود و نتیجه در پورت خروجیH با پهنای 16 بیت قرارمی‌گیرد. هرکدام ازپورتهای A وH به صورت یک بردار 16 بیتی تعریف شده است.([ ]برای تعیین محدودهٔ بردار استفاده می شود.) اگر محدوده‌ای برای پورت تعریف نشود به عنوان یک بیت منظورخواهد شد، بدین ترتیب H_sel تک بیتی خواهد بود. کد فوق هیچ اطلاعاتی در مورد عملکردshifter ارائه نمی دهد ولی ساختار خارجی مدول را که توسط مدولهای دیگر مشاهده می‌شودکاملا مشخص می‌کند.

۲. *انواع دادهٔ Wire و Reg*

Verilog دارای دونوع دادهٔ اصلی net وreg است. ما ازwire که گونه‌ای net است استفاده خواهیم کرد. ویژگی اصلی wire، برخلاف reg این است که ،قدارخود را حفظ نمی‌کند وتنها برای ارسال داده‌ای که همواره دارای تحریک است مناسب می‌باشند. در عوض reg قابلیت حفظ مقدار خود رادارد. اگر بخواهیم خروجی واحد shifter مقدار خود را حفظ کند باید آنرا به صورت reg تعریف کنیم:

```
reg [15:0]H;
```

این جمله پورت H را به عنوان یک رجیستر16 بیتی تعریف می‌کند. بقیهٔ پورتها از نوع wire فرض خواهند شد.

۳. *مدلسازی رفتاری*

مدلسازی رفتاری بسیارشبیه زبانهای برنامه‌نویسی است. ساختارهای **if-then-else** و**case statements** و حلقهٔ **for** ... ، دردسترس هستند. علاوه برساختارهای فوقVerilog دستورات جدیدی برای‌توصیف سخت افزارنیز دارد.

هر module دارای دونوع کد می‌تواند باشد: **concurrent bodies**(هم‌روند)و**procedural bodies**(رویه‌ای)

در **concurrent bodies** تمامی جملات به صورت هم‌روند اجرا می‌شوند و ترتیب جملات اهمیتی ندارد.

در**procedural bodies** جملات به صورت متوالی(نظیر سایر زبانها) و تنها در صورت یک اتفاق(مثلا لبهٔ کلاک) بررسی می‌شوند.

متغیرهایی ازنوعwire درون **concurrent bodies** با استفاده از دستور **assign** مقداردهی می‌شوند، درحالیکه متغیرهای reg درون **procedural bodies** مقداردهی می‌شوند.

٤. جملهٔ *Always*

این جمله شروع یک **procedural body** رامشخص می‌کند.

```
always @(A or H_sel)
begin
      if (H_sel)
            H = {A[14:0], 1'b0};
      else
            H = {A[15], A[15:1]};      //sign extension
end
```

always @(A or H_sel) بیان می‌کند که جملات درون این بلوک تنها زمانی که یک تغییردر Aیـا H_sel اتفـاق بیافتد ارزیابی شوند. بدین ترتیب زمانیکه ورودی A و یا سیگنال کنترلی H_sel تغییر کنند، اگر H_sel برابر ۱باشد A یک بیت به چپ شیفت داده شده درون H نوشته می‌شود. اینکار توسط دستور:

```
H = {A[14:0], 1'b0};
```

انجام می‌گیرد. درواقع این دستور بیتهای A[14:0] را به جای بیتهای H[15:1] جاگذاری و یـک بیـت 0 بـه جـای H[0] قرار می‌دهد ({ } به معنی به هم چسباندن است.) و اگر H_sel=0 دستور:

```
H = {A[15], A[15:1]};
```

A را با حفظ علامت یک بیت به راست شیفت داده درون H می‌نویسد.

تفاوت اساسی این کد با زبانهای دیگر این است که این جملات نه فقط یکبـار بلکـه بـه طوردائـم تـا پایـان زمـان شبیه‌سازی بررسی و اجرا می‌شوند. تمامی کد مدول shifter درزیر آمده است:

```
// M. Mano 1997, "Computer System Architecture";
// 16_bit Shifter Unit(without any effect on flags)
// Shifter unit of Function unit with ability to shift to left&right


module Shifter16 ( A, H_sel, H);

input [15:0]A;
input H_sel;
output [15:0]H;

reg [15:0]H;

always @(A or H_sel)
begin
      if (H_sel)
            H = {A[14:0], 1'b0};
      else
            H = {A[15], A[15:1]};      //sign extension
end

endmodule
```

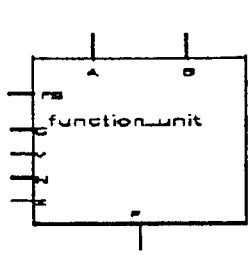
برای توضیح بیشتر به مرجع اشاره شدهٔ فوق مراجعه کنید.

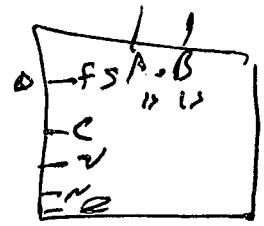
- پروژه را به نام این فایل ست و Save&Check کنید.
- پروژه را کامپایل functional کنید.
- بااجرایWaveform Editor شکل‌موج سیگنالهای تست ورودی راایجاد وصحت عملکردمدول رابررسی کنید.
- برای این مدول یک symbol ایجاد و آنرا edit کنید تا به شکل مناسب درآید.
- پروژه را آرشیو کنید.

طراحی Function Unit

اکنون می‌توانید بااستفاده از مدولهای ALU و Shifter طراحی Function Unit کامپیوتر پایه را با جدول عملیاتی زیرآغاز کنید:

FS[4..0]	Function
00000	F=A
00001	F=A+1
00010	F=A+B
00011	F=A+B+1
00100	F=A+/B
00101	F=A-B
00110	F=A-1
00111	F=A
0100x	F=A∧B
0101x	F=A∨B
0110x	F=A⊕B
0111x	F=/A
10000	F=sr A
10001	F=sl A





- یک فایل جدید شماتیک با نام **function_unit.gdf** ایجاد و با استفاده از عناصر **alu**، **shifter16** و **busmux** مدول **function_unit** را طوری طراحی کنید که جدول عملیاتی فوق راپیاده‌سازی نماید.
- پروژه را کامپایل و شبیه سازی **functional** کنید.(باایجاد شکل موج مناسب برای سیگنالهای ورودی عملکرد طرح را بررسی نمایید.)
- یک **symbol** مناسب برای **function_unit** ایجاد کنید.
- پروژهٔ خود را آرشیو نمایید.

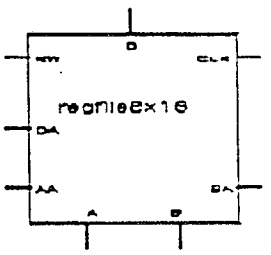
طراحی Register File

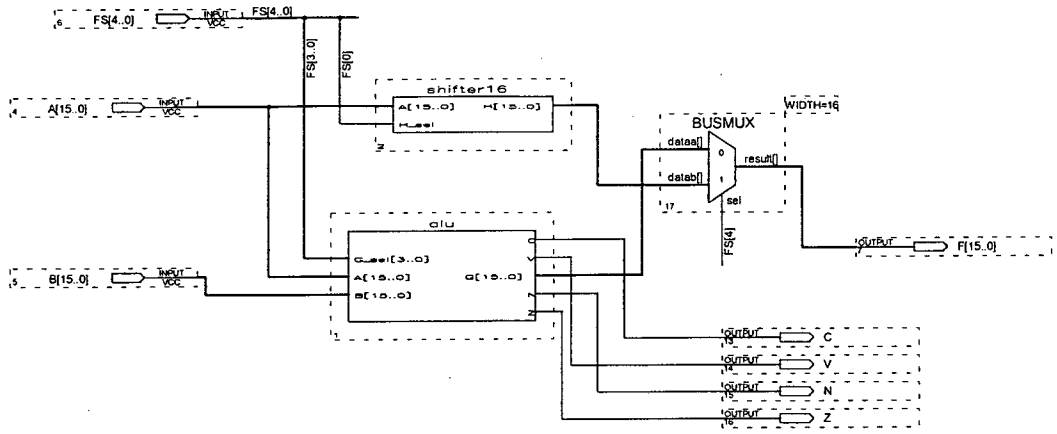
در این مرحله، توصیف رفتاری(**behavioral**) یک Register File 8x16 را بااستفاده از Verilogخواهیم نوشت.

این رجیسترفایل دارای 8 رجیستر16 بیتی R0, R1, ..., R7 با قابلیت خواندن همزمان دو رجیستر است.

خواندن ازرجیستر فایل: آدرس رجیسترهای منبع از طریق خطوط آدرس(AA(A Address و(BA(B Address مشخص می شود و محتویات رجیسترها به ترتیب در پورتهای خروجی A وB قرار می‌گیرد.

نوشتن در رجیستر فایل: آدرس رجیسترمقصد توسط خطوط آدرس (DA(D Address مشخص و مقدارپورتD درصورت فعال(1) بودن سیگنال RW(Register Write) با لبهٔ بالارونده کلاک درون رجیستر مقصد نوشته می‌شود.




FS[4..0]
INPUT
VCC
FS[4..0]
FS[3..0]
FS[0]
A[15..0]
shifter16
A[15..0]
H[15..0]
H_sel
WIDTH=16
BUSMUX
data[]
datab[]
result[]
sel
FS[4]
OUTPUT
F[15..0]
B[15..0]
alu
G_sel[3..0]
A[15..0]
B[15..0]
G[15..0]
C
V
N
Z

- New(File Menu) را انتخاب و Text Editor files و v.* را به عنوان نوع فایل مشخص کنید.
- فایل جدید را با نام regfile8x16.v ذخیره کنید.
- می توانید درون فایل با نوشتن توضیحات (comment) بعداز// طرح خود را خواناتر کنید.
- اولین قدم برای ایجاد یک **module** با استفاده از Verilog تعریف آن است. با تایپ جملات زیر در ادیتور، مدول **regfiel8x16** را با مشخص کردن پورتهای ورودی و خروجی تعریف کنید:

```
module regfile8x16(CLK, RW, AA, BA, DA, A, B, D);
endmodule
```

توصیف مدول میان این دوخط نوشته خواهد شد.

- قدم بعدی تعیین نوع پورتها است. نظیرمثال قبل نوع پورتها را مشخص کنید:

```
input CLK;              // Clock for write port
input RW;               // Register Write enable signal
input [2:0] AA;         // Read address for source A read port
input [2:0] BA;         // Read address for source B read port
input [2:0] DA;         // Store address for dest. D write port
output [15:0] A;        // Source A read port
output [15:0] B;        // Source B read port
input [15:0] D;         // Destination write por
```

- مرحلهٔ بعدتعریف متغیرهاست. این مدول دارای 8 رجیستر16 بیتی R0, R1, ..., R7 نیز هست که باید تعریف شوند. این رجیسترهارا به صورت زیر تعریف کنید:

```
reg [15:0] R0, R1, R2, R3, R4, R5, R6, R7;          // A 8 x 16 bit memory array
```

همچنین پورتهای خروجی A و B باید از نوع **reg** تعریف شوندتا مقدار خود را حفظ کنند:

```
reg [15:0] A, B;          // Declare read ports as registers
```

- اکنون باید توصیف رفتار مدول را بنویسیم. هرگاه خطوط آدرس AA یا هرکدام از رجیسترهای R0-R7 تغییر کنند باید پورت خروجی A به روز شود. جملات زیرتوصیف فوق را پیاده‌سازی می کنند:

```
always @(AA or R0 or R1 or R2 or R3 or R4 or R5 or R6 or R7)
begin
      case (AA)
                  0: A=R0;          // Fetch A data using AA
                  1: A=R1;
                  2: A=R2;
                  3: A=R3;
                  4: A=R4;
                  5: A=R5;
                  6: A=R6;
                  7: A=R7;
      endcase
end
```

- به همین ترتیب برای پورتB نیز این کاررا انجام دهید.

جملات فوق عملیات خواندن از رجیسترفایل را پیاده سازی می کنند.

در ادامه توصیف نوشتن در رجیسترفایل را می نویسیم.

- درلبهٔ بالارونده کلاک(**posedge clk**) درصورت 1 بودن RW مقدار پورت D درون رجیستر مشخص شده توسط DA نوشته می شود. جملات زیر این عملیات را انجام می دهند:

```
always @(posedge CLK)                    // At positive edge of clock...
begin
      if (RW)
      begin
            case (DA)
                  0: R0=D;               // Write D data to register addressed by DA
                  1: R1=D;
                  2: R2=D;
                  3: R3=D;
                  4: R4=D;
                  5: R5=D;
                  6: R6=D;
                  7: R7=D;
            endcase
      end
end
```

بدین ترتیب توصیف رفتاری **regfile8x16** کامل خواهد بود.

Verilog دارای متغیرهایی ازنوع حافظه**memory** است(آرایه‌ای از رجیسترها) که امکان ایجاد رجیسترفایل را به روش بسیاربهتری میسرمی‌سازد ولی به دلیل عدم پشتیبانی Max+Plus II از این ویژگی(وبرخی ویژگیهای دیگر)ازروش فوق استفاده شده است. برای بهره‌گیری ازتمامی ویژگیهای Verilog باید ازکامپایلرهای کاملتری استفاده نمود.(کتاب مرجع)

- پروژه رابه نام این‌فایل ست‌کرده آنرا**Save&Check&Compile** کنید. درصورت لزوم خطاهای طرح را رفع نمایید.
- مدول را شبیه سازی نموده، عملکرد آنرا بررسی نمایید.
- برای مدول یک **symbol** ایجاد و آنرا به شکل مناسب درآورید.
- پروژه را آرشیو کنید.

گزارش کار:

۱. تحویل نتایج شبیه‌سازی برای مدولهای Function Unit و Register File.

۲.توصیف رفتاری(**behavioral**) یک مدول shifter با قابلیت چندین شیفت به چپ وراست را به زبان Verilog بنویسید. این shifter دارای یک پورت اضافی بانام N به پهنای 4 بیت خواهد بودکه تعداد شیفت رامشخص می‌کند. طرح خود را شبیه‌سازی و عملکرد آنرا بررسی نمایید.

طراحی Data Path و Control Unit

کامپیوتر Single-Cycle

نگاه کلی

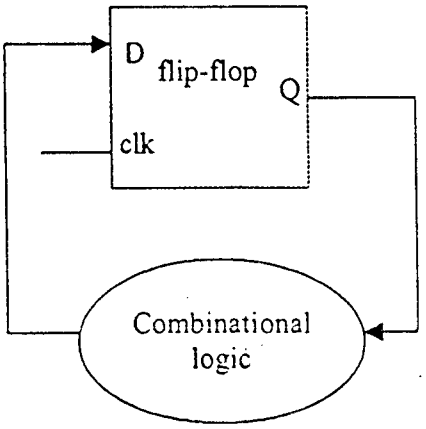
در این آزمایش با استفاده از ماجولهای Function Unit و Register File و برخی عناصر دیگر، Data Path و Control Unit کامپیوترتک سیکل (Single-Cycle) پایه را مدلسازی خواهیم نمود. برخی ازمدولها(نظیرشمارندهٔ برنامه PC وکدگشای دستورالعمل Instruction Decoderو ...) توسط Verilog توصیف خواهد شد. همچنین نحوهٔ استفاده از عناصر حافظه **(lpm_ram, lpm_rom)** را مشاهده خواهید نمود.

نحوهٔ عملکرد کامپیوتر تک سیکل (Single-Cycle)

همانطورکه می‌دانید اجرای هردستورالعمل درکامپیوتر تک سیکل تنها در طول یک سیکل کلاک انجام می‌گیرد. بدین صورت که در لبهٔ بالارونده هر کلاک PC به حافظهٔ دستورالعمل(Instruction Memory) اعمال و دستورالعمل واکشی (fetch) می‌شود. سپس‌این‌دستورالعمل کدگشایی شده(توسط Instruction Decoder) وسیگنالهای‌کنترلی به data path فرستاده می‌شوند. نهایتاً درلبهٔ بالارونده بعدی کلاک داده‌ای دررجیسترفایل یا حافظهٔ داده(Data Memory) نوشته می‌شود وهمزمان دستورالعمل بعدی fetch می‌گردد. بدین ترتیب مدت زمان یک سیکل ساعت بایدبزرگتریامساوی بامدت زمان‌اجرای طولانی‌ترین دستورالعمل‌باشدتا سیستم درست عمل کند.

روشهای زمانبندی

روش فوق که درآن از زمانبندی تحریک شونده با لبه(**edge-triggered**) استفاده شده است آسانترین روش زمانبندی محسوب می‌شود و با رعایت چند قانون ساده به درستی عمل خواهدکرد. در واقع با این فرض که کلاک به طورهمزمان به تمامی رجیسترها وحافظه می‌رسد، اگر طول کلاک به اندازهٔ کافی بزرگ باشد طرح به درستی عمل خواهد کرد وهیچگونه **race**(مسابقه) اتفاق نخواهد افتاد. **Race** زمانی اتفاق می افتد که محتویات یک رجیستر به سرعت نسبی عناصر منطقی مختلف بستگی پیدا کند.



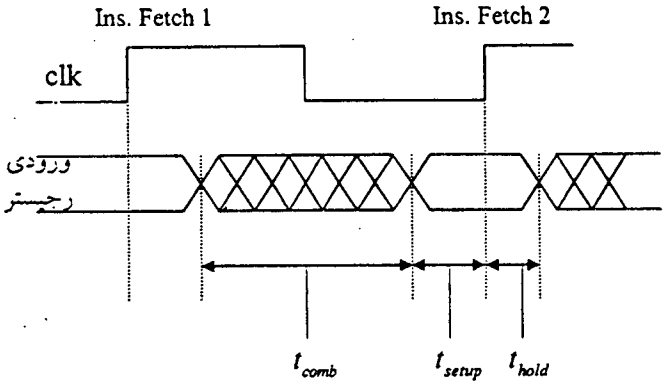
در روش زمانبندی تحریک شونده با لبه لازم است که دوشرط زیربرقرار باشد:

۱- مدت زمان یک سیکل ساعت بزرگتر از مقدار:

$t_{prop} + t_{comb} + t_{setup}$

باشد که در آن t_{comb} ماکزیمم تأخیر مدار ترکیبی، t_{setup} مدت زمانی که ورودی رجیستر قبل از لبهٔ کلاک باید ثابت و پایدارنگه داشته شود و t_{prop} ماکزیمم تأخیرانتشاررجیستراست.

۲- مجموع تأخیر انتشارمدارترکیبی ورجیستر باید بزرگتراز زمان t_{hold} باشد که درآن t_{hold} حداقل زمانی است که ورودی رجیستر بعداز لبهٔ کلاک باید پایدارباقی بماند. این بدان معنی است که در لبهٔ کلاک که خروجی رجیستر تغییرمی کند این تغییربعدازمدت زمان t_{hold} به ورودی رجیستر انتشارپیدا کند. این شرط تقریباً همواره برقرار است زیرا t_{hold} رجیستربسیارکوچک است.



موضوع دیگری که باید مدنظر قراربگیرد مشکل Clock skew است که عبارت از رسیدن لبهٔ کلاک به عناصر حافظه در زمانهای متفاوت بدلیل اختلاف در طول مسیرهایی است که سیگنال کلاک طی می‌کند. روش زمانبندی تحریک شونده بالبه دارای دو نقص است: نیاز به مدارمنطقی بیشتر و سرعت کم. روش دیگر clocking استفاده از زمانبندی تحریک شونده باسطح است.(Computer Organization&Design D.Patterson, H. Hennessy 1998)

مدلسازی Program Counter(PC)

شمارندهٔ برنامه یک رجیستر16 بیتی با قابلیت افزایش خودکار(AutoIncrement) درلبهٔ بالارونده هرکلاک است. این ماجول را با استفاده ازVerilog توصیف خواهیم کرد.

- New(File Menu) را انتخاب وText Editor files و *.v را به عنوان نوع فایل مشخص کنید.
- فایل جدید را با نام PC.v ذخیره کنید.
- این ماجول دارای یک ورودی clk و یک خروجی InsAdd[15..0] است. module را تعریف ونوع پورتها را تعیین کنید.
- خروجی InsAdd را به صورت رجیستر تعریف کنید تا مقدار خود را حفظ نماید.
- مقداراولیهٔ PC را برابرx0000 قراردهید. اینکارتوسط دستورinitial بهصورتی که درکد زیرمی‌بینیدانجام می‌گیرد.

• در لبهٔ بالارونده کلاک مقدار PC یک عدد افزایش می‌یابد. این کاررا بااستفاده ازدستور**always** پیاده‌سازی کنید.

کد کامل ماجول به شکل زیر خواهد بود:

```
// M. Mano 1997, "Computer Architecture"
// 16-bit Autoincrement Program Counter(PC)

module PC ( clk, InsAdd);

input clk;
output [15:0] InsAdd;

reg [15:0] InsAdd;

initial
begin
      InsAdd[15:0] = 16'b0;      //16  0: 16 bit 0
end

always @(posedge clk)
begin
      InsAdd <= InsAdd+1;
end

endmodule
```

• پروژه را **Save&Compile** کنید.

می‌توانید طرح خود را شبیه سازی کنید تا از صحت عملکرد آن مطمئن شوید.

• یک **symbol** مناسب برای ماجول طراحی کنید.

مدلسازی Instruction Decoder

کدگشای دستورالعمل یک مدارترکیبی(combinational) است که ورودی آن دستورالعمل (instruction) و خروجی آن سیگنالهای کنترلی است که به data path اعمال می‌شوند. این ماجول راباVerilog توصیف‌خواهیم کرد.

• یک فایل Verilogجدید با نام **Instruction_Decoder.v** ایجاد کنید.

• Module را تعریف و نوع پورتهارا مشخص کنید. این ماجول دارای یک ورودی Ins و چندین خروجی DA,AA, BA, MB, FS, MD, RW, MW است.

• تمامی این خروجیها از نوع **wire** (بدون حافظه) هستند. آنها را از نوع **wire** تعریف کنید.

قدم بعدی تعیین رابطهٔ خروجیها با ورودی است. قبلا گفته شد که مقداردهی متغیرهایی ازنوع **wire** توسط دستور **assign** انجام می‌گیرد. این دستور تنهادر قسمت همروند(**concurrent body**) ماجول قابل استفاده است و برای تعریف مدارهای ترکیبی مناسب می‌باشد.

• خروجیها را براساس Ins[15:0] بااستفاده از دستور **assign** تولیدکنید. به عنوان مثال می‌دانیم که RW=Ins[14]+Ins[13] . جملهٔ زیر توصیف Verilog این عملیات منطقی است:

```
assign RW = ~Ins[14]|Ins[13];     //~:not,     &:and,   |:or,   ^:xor
```

می توان برای این جملات تأخیر نیز وارد کرد تا مدلسازی دقیقتر باشد. به عنوان مثال جملهٔ زیر :

```
assign #10  RW = ~Ins[14]|Ins[13];
```

عبارت سمت چپ تساوی را بعد از 10*unit_delay به RW نسبت می‌دهد. (unit_delay درابتدای کد مثلا برابر 1ns ست می‌شود: (Verilog Digital System Design, Z. Navabi)('timescale 1ns/100ps

کد کامل ماجول **Instruction_decoder** به صورت زیر خواهد بود:

```
// M. Mano 1997, "Computer Architecture";
// Single Cycle Instruction decoder

module Instruction_Decoder ( Ins, MW, RW, MD, FS, MB, BA, AA, DA);

input [15:0] Ins;
output MW, RW, MD, MB;
output [2:0] BA, AA, DA;
output [4:0] FS;

wire MW, RW, MD, MB;
wire [2:0] BA, AA, DA;
wire [4:0] FS;

assign DA[2:0] = Ins[8:6];
assign AA[2:0] = Ins[5:3];
assign BA[2:0] = Ins[2:0];
assign MB = Ins[15];
assign FS[4:0] = Ins[13:9];
assign MD = Ins[14];
assign RW = ~Ins[14]|Ins[13];
assign MW = ~RW;

endmodule
```

- پروژه را **Save&Compile** کنید.
- یک **symbol** مناسب برای ماجول طراحی کنید.

مدلسازی Zero_Fill

ماجول دیگری که در data path کامپیوتر تک سیکل استفاده شده است Zero_Fill می باشد. ورودی این ماجول مقدار3 بیتی Const (همان Ins[2:0])است که 13 بیت 0 (13'b0) به سمت چپ آن چسبانده شده و خروجی 16 بیتی تولید می‌شود. اینکار به منظور اطمینان از0 بودن بیتهای [15..3] ورودی Mux انجام می‌گیرد.

- یک فایل Verilogجدید با نام **Zero_Fill.v** ایجاد کنید.
- Module را تعریف و نوع پورتهارا مشخص کنید. این ماجول دارای یک ورودی Ins[2:0] و یک خروجی Const[15:0] است.
- Const[15:0] را ازنوع **reg** تعریف کنید.
- با استفاده از دستور**always** هرگاه تغییری درورودی Ins اتفاق بیافتد خروجی با دستور مناسب Const={13'b0, Ins[2:0]} به روز شود.
- پروژه را **Save&Compile** کنید.
- یک **symbol** مناسب برای ماجول طراحی کنید.

```
module Zero_Fill ( Const, Ins);
input [2:0] Ins;
output [15:0] Const;
reg [15:0] Const;
always @(Ins)
begin
    Const = {13'b0, Ins[2:0]};
end
endmodule
```

استفاده از عناصرحافظه در Max+Plus II

همانطورکه قبلاً دیدید Max+Plus II دارای کتابخانه‌ای با نام **mega_lpm library** است که شامل عناصر پارامتری می‌شود. عناصرحافظه نظیرROM وRAMجزو این کتابخانه می‌باشند و قابلیت تنظیم پهنای خطوط داده وآدرس را دارند. این عناصر همچنین دارای یک فایل آغازدهی(Memory Initialization File **.mif*) هستند که می‌توان محتویات حافظه را درآن نوشت.

طراحی Data Path و Control کامپیوترSingle-Cycle

در این مرحله از آزمایش بااستفاده از عناصری که تاکنون طراحی کرده‌ایم کامپیوتر تک سیکل را مدلسازی خواهیم نمود. اگرچه ما ماجولهای مختلف را در طراحی top-down به روشهای متفاوتی نظیر شماتیک وHDL مدلسازی کرده‌ایم ولی این ماجولها درطرح سطح بالا قابل‌ترکیب هستند. طبیعتاً استفاده ازتنها یک ابزارمدلسازی (مثلاً HDL) باعث دقت و قابلیت انعطاف بیشتر طرح خواهد بود.

- فایل شماتیک کامپیوتر تک سیکل را با نام **Single_Cycle.gdf** ایجاد و ذخیره کنید.
- عناصر **Function_Unit**، **RegFile8x16**، **Instruction_Decoder**، **PC**و **Zero_Fill** را برروی صفحه قرار دهید.
- دو عدد **BusMux** وارد و پارامتر**WIDTH** آنها را برابر 16 ست کنید.
- یک عدد **lpm_rom** وارد شماتیک کرده، درصفحهٔ ادیت پارامترها مقادیر پارامترها را به صورت زیر ست کنید:

LPM_FILE="ins_mem.mif" (نام فایلی که درآن محتویات ROM را خواهیم نوشت.)

LPM_WIDTH=16 (پهنای باس داده برابر16 بیت است.)

LPM_WIDTHAD=8 (پهنای باس آدرس را برابر 8 قرار می دهیم.)

بقیهٔ پارامترها را بدون تغییر رها کنید.

از این ROM به عنوان حافظهٔ کد(Instruction Memory)استفاده خواهیم کرد.

پارامتر **LPM_FILE** فایل آغازدهی ROM را مشخص می‌کند. این فایل را **ins_mem.mif** نامیده‌ایم که بعداً آنرا ایجاد و ادیت خواهیم کرد. در واقع این فایل محتویات ROM را مشخص می‌کند. برای تست کامپیوتر تک سیکل، برنامه های اسمبلی کامپایل شده به زبان ماشین را درون این فایل وارد کرده و اجرا خواهیم کرد.

پارامتر **LPM_WIDTH** پهنای باس داده را مشخص می‌کند که در این طرح برابر 16 بیت است.

پارامتر **LPM_WIDTHAD** پهنای باس آدرس را مشخص می‌کند. درطرح کامپیوتر تک سیکل باس آدرس 16 بیتی(64KW حافظه)است ولی به منظور سادگی طرح و کم کردن زمان کامپایل تنها 8 بیت خط آدرس در نظر گرفته شده است بدین ترتین ROM دارای256 کلمه خواهد بود.(MAX+Plus II دارای محدودیت در مقدار فضای حافظهٔ می‌باشد و توانایی کامپایل حافظه‌هایی با حجم بزرگ را ندارد.)

- یک عدد lpm_ram-dq وارد شماتیک کرده، درصفحهٔ ادیت پارامترها، مقادیرپارامترها را بـه صـورت زیرسـت کنید:

LPM_FILE="data_mem.mif" (نام فایلی که درآن محتویات RAM را خواهیم نوشت.)

LPM_WIDTH=16 (پهنای باس داده برابر16 بیت است.)

LPM_WIDTHAD=8 (پهنای باس آدرس را برابر 8 قرار می دهیم.)

بقیهٔ پارامترها را بدون تغییر رها کنید.

از این RAM به عنوان حافظـهٔ داده(Data Memory)اسـتفاده خواهیـم کـرد. کتابخانـهٔ maga_lpm دارای انـواع RAM می‌باشدکه lpm_ram_dq با پورتهای داده ورودی وخروجی مجزا مناسـبترین آنـها بـرای طـرح کـامپیوتر تک سیکل می‌باشد.

پارامتر LPM_FILE فایل آغازدهی RAM را مشخص می‌کند. این فایل را data_mem.mif نـامیده‌ایم کـه بعـدا آنرا ایجاد و ادیت خواهیم کرد. ازاین فایل برای نوشتن داده‌های اولیه درون حافظهٔ داده و تست برنامه های اسمبلی استفاده می‌نماییم.

در اینجا نیزجهت سادگی طرح، پهنای باس آدرس برابر 8 بیت ست شده است لذا طرح تنها دارای 256 کلمـهٔ داده خواهد بود.

- شماتیک Data Path و Control کامپیوتر تک سیکل را نظیر شکل۱ کامل کنید.

دقت کنید که خطوط آدرس عناصرحافظه نظیر شکل۱، تنها شامل 8 بیت کم ارزش باس آدرس باشد.

- پرچمها را به عنوان خروجی طرح قرار دهید.(اگرطرح دارای خروجی نباشد خطا اعلام خواهد شد.)
- پروژه را Save&Check کنید و درصورت وجود خطا آنها را رفع نمایید.(Warning هایی را کـه عـدم وجـود فایلهای آغازدهی عناصرحافظه را اعلام می‌کنند نادیده بگیرید.)
- پروژه را کامپایل functional کنید.

شبیه سازی کامپیوترSingle-Cycle

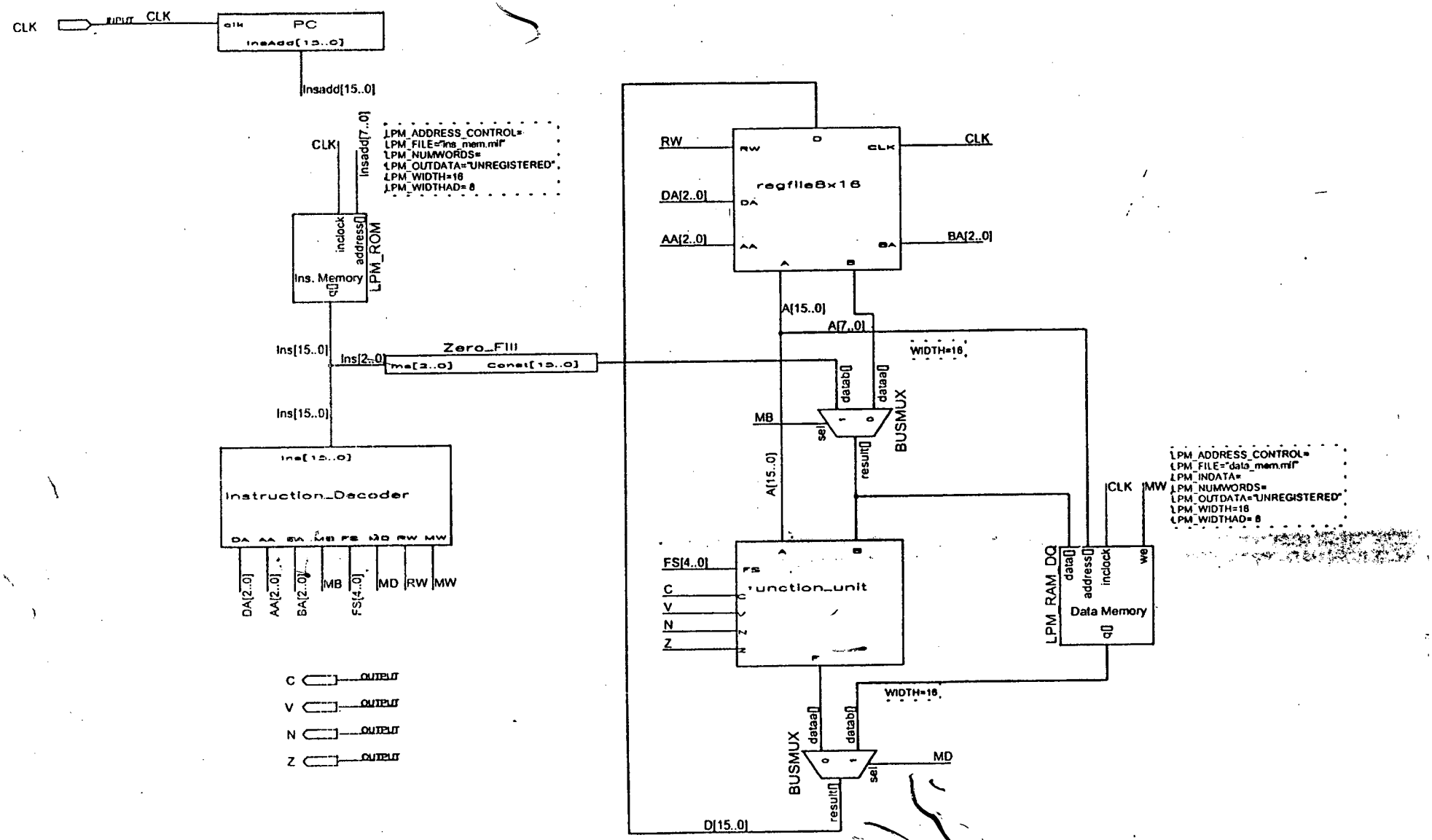
در این قسمت با نوشتن یک برنامهٔ اسمبلی و قراردادن آن درون حافظهٔ کد، طرح را تست خواهیم نمود.

برنامهٔ اسمبلی زیردومقدار ثابت را درون رجیسترهای R1 وR2 قرار داده(با فرض مقادیر اولیهٔ رجیسترها برابر0) و آنها را باهم جمع می کند و نتیجه را در رجیستر R3 می نویسد:

```
ADI     R1,R0,5       // R1<-R0+5
ADI     R2,R0,3       // R2<-R0+3
ADD     R3,R2,R1      // R3<-R1+R2
```

کد زبان ماشین این برنامه به صورت زیرخواهد بود:

```
x8445          //1000010001000101
x8483          //1000010010000011
x04D1          //0000010011010001
```



Data Path & Control of Single-cycle Computer

- Simulator(MAX+plus II menu) را انتخاب کنیدتا پنجرهٔ simulator باز شود.
- Initialize Memory... (Initialize menu) را انتخاب نمایید تا پنجرهٔ Initialize Memory باز شود.
- LPM_ROM را از لیست Memory Name انتخاب کنید.

لیستی که مشاهده می‌کنید محتویات حافظهٔ کد را نشان می‌دهد. می‌توانید با انتخاب هرآدرس، دادهٔ 16 بیتی آنرا تعیین کنید.

- کد زبان ماشین فوق را وارد این لیست کنید.
- دکمهٔ Export File... را انتخاب کنید ونام فایل را ins_mem.mif قرار داده در دایرکتوری خود ذخیره نمایید.

بدین ترتیب فایل ins_mem.mif ایجاد و هنگام شبیه سازی مورد استفاده قرار می‌گیرد.

- Waveform Editor را اجرا و گرههای CLK، Insadd[15..0]، Ins[15..0] و R0-R7 را انتخاب نمایید.
- سیگنال CLK را انتخاب وروی آیکون Overwrite Clock (سمت چپ صفحه) کلیک کرده و OK کنید.

بدین ترتیب سیگنال کلاکی با فرکانس مشخص (مثلا 100MHz) تولید خواهد شد.

- پروژه را Save&Simulate کنید.
- با بررسی شکل موج و مقادیر رجیسترها در Waveform Editor درستی نتیجه را بررسی کنید.(بعد از 3 کلاک مقدار R3 باید برابر 8 باشد.)
- پروژه را آرشیو کنید.

گزارش کار:

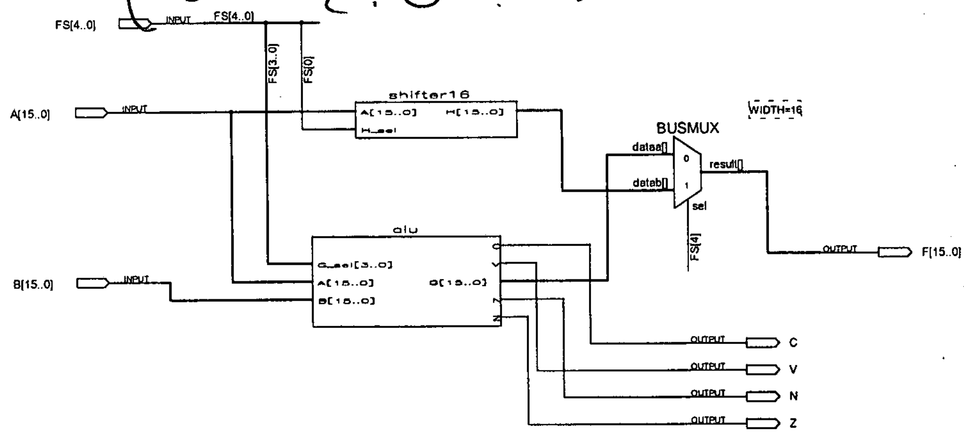
۱. تحویل نتایج شبیه سازی برنامهٔ اسمبلی فوق.

۲. Data Path طراحی شده توانایی اجرای کدامیک از دستورات LDI، BR(Branch)، BZ(Branch Zero)، AND، MOV و XOR را دارد؟ در صورت عدم امکان توضیح دهید که چه قسمتهایی از Data Path و Control باید تغییرکند تا طرح از این دستورات نیز پشتیبانی نماید.

۳. برنامه‌ای بنویسید که محل دو مقدار موجود درحافظهٔ داده (مثلا در آدرسهای x00 و x01) را تعویض نماید(swap). برای اینکار ابتدا حافظهٔ داده را به کمک فایل data_mem.mif مقداردهی اولیه کرده وکد ماشین برنامه را درون حافظهٔ کد(فایل ins_mem.mif)بنویسید، سپس برنامه را شبیه سازی نمایید.

jump مطلق است (دقیقاً آدرس حافظه را درون PC می‌گذاریم) ← فرق jump و Branch

Branch نسبی است (offset را درون PC می‌گذارد و offset با آن جمع می‌شود.)



MB: نشان‌دهنده روش آدرس‌دهی است اگر ۱ باشد آدرس‌دهی بلافصل است و داده مستقیم (zero-fill) می‌آید اگر صفر باشد از رجیستر فایل می‌آید.

نوع دستورات ← حافظه ← Load (خواندن از حافظه) ≡ نوشتن به ثبات

store (نوشتن به حافظه) (داده از ثبات یا بلافصل می‌آید)

برای دستور store بسته به RW و CLK مقدار درون حافظه نوشته می‌شود.

MD: برای دستور Load مقدار آن را می‌گیریم و مقدار آن خوانده شده درون رجیستر فایل نوشته می‌شود.

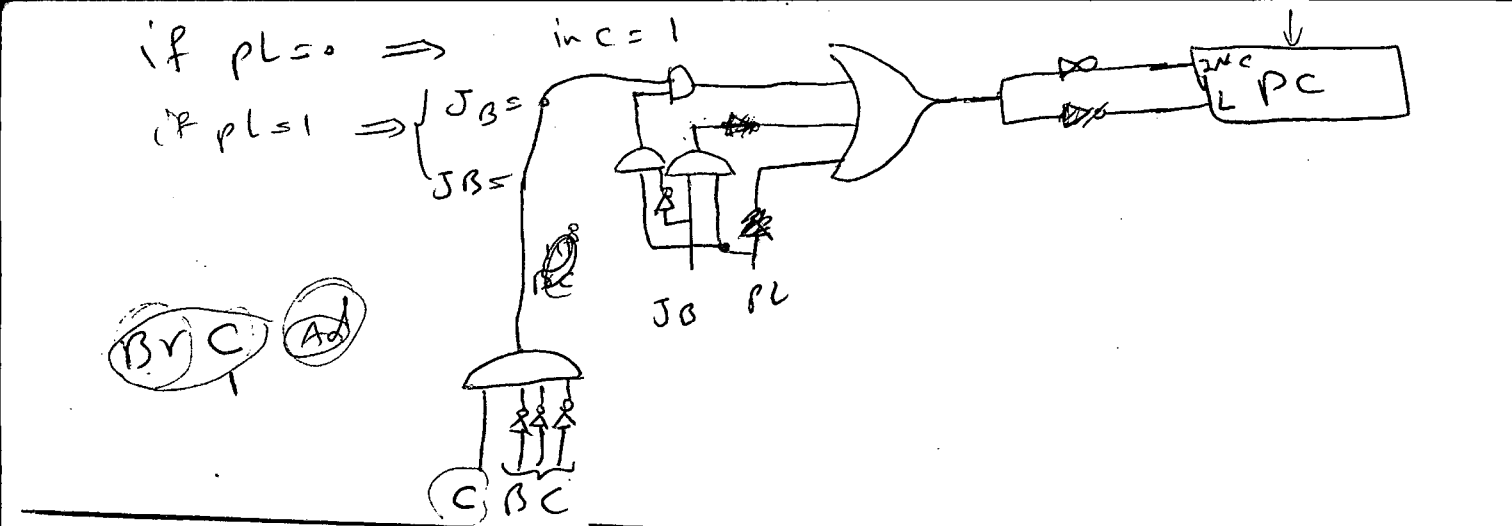
صفحه ۴۳۲ ECC

سه بیت ۱۲ و ۱۴ و ۱۵ که IC آپکد می‌باشد برای تعیین عملگر می‌باشد جدول ECC - دو سطر اول در واقع نشان‌دهنده Bit 13 است که عمل درون Function unit می‌باشد.

PL: Load Program Counter کنترل کننده PC مشخص می‌کند

BC jump: دستور پرش که جایی است که ۱ است، در بقیه مواقع صفر.

JB: مشخص کننده عمل jump و یا Branch می‌باشد.



طراحی Data Path و Control Unit

کامپیوتر Multi-Cycle

نگاه کلی

در این آزمایش Data Path و Control Unit کامپیوترچند سیکل (Multi-Cycle) پایه را مدلسازی خواهیم نمود. برخی ازمدولها(نظیرشمارندهٔ برنامه PC ورجیستر دستورالعمل Instruction Registerو ...) توسط Verilog توصیف و Data Path سیستم طراحی خواهد شد. Control Unit به صورت یک ماشین حالت (FSM) با استفاده از Verilog توصیف می‌شود.

نحوهٔ عملکرد کامپیوتر چند سیکل (Multi-Cycle)

اجرای هردستورالعمل درکامپیوتر چند سیکل درطول چندین سیکل کلاک انجام می‌گیرد. درآزمایش قبل مشاهده کردید که درکامپیوتر تک سیکل طول هر سیکل ساعت باید حداقل برابر مدت زمان اجرای دستورالعمل با بیشترین زمان اجرا باشد. این کار موجب می‌شود که در بعضی از دستورالعملهایی که سریعتر اجرا می‌شوند پردازنده بیکار بماند وباعث کاهش کارایی سیستم گردد. با تقسیم اجرای هردستورالعمل به چندین کلاک و درعین حال کاهش زمان سیکل کلاک، می‌توان دستورالعملهای مختلف را در تعداد کلاکهای متفاوت اجرا کرد. بدین ترتیب اجرای یک دستورالعمل پیچیده ممکن است 10 کلاک به طول انجامد ولی دستورالعملهای ساده‌ترمثلاً در 3 کلاک اجرا شوند. مزیت دیگر اینکار آن است که از ماجولهای مختلف موجود در Data Path می‌توان در طی اجرای یک دستورالعمل چندین بار استفاده کرد. به عنوان مثال در کامپیوتر چند سیکل تنها یک واحد حافظه وجود خواهد داشت که هم شامل کد وهم شامل داده است. به علاوه، طرح Multi-Cycle امکان ایجاد سیستم Pipeline را فراهم می‌نماید که نوعی پردازش موازی درون پردازنده محسوب می‌شود و کارایی سیستم را درحد قابل توجهی افزایش می‌دهد.

Data Path وControl کامپیوتر چندسیکل

Data Path کامپیوتر چندسیکل دارای تفاوتهایی با کامپیوتر تک سیکل است:

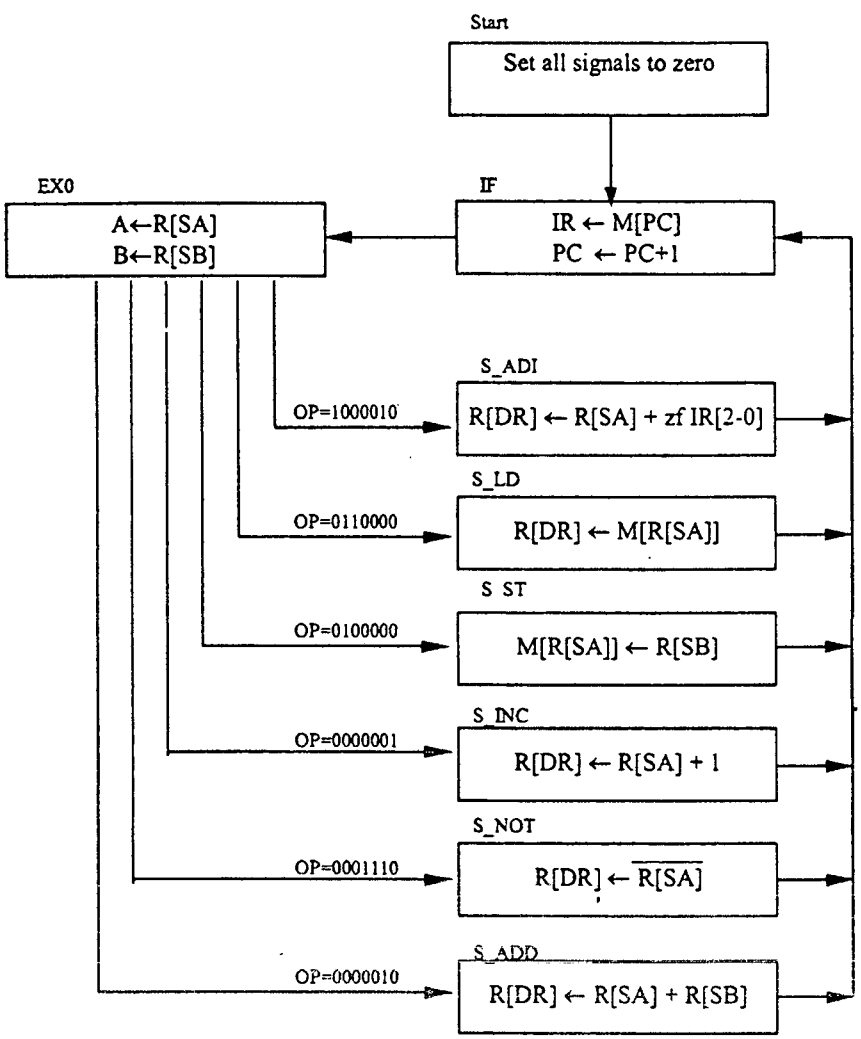
۱. در کامپیوتر چندسیکل نیاز به رجیسترهای اضافی است تا خروجیهای ماجولهای مختلف را درطی کلاکهای متوالی ذخیره نماید. یکی از این رجیسترها IR(Instruction Register) است که دستورالعمل واکشی شده از حافظه را حفظ می‌کند.

۲. نیاز به مالتی پلکسرهای اضافی است تا امکان استفاده از یک ماجول به منظورهای متعدد فراهم شود.

۳. Register File دارای یک رجیستراضافی R8 برای استفادهٔ موقت در بعضی از دستورالعملها(نظیر LRI) است. لذا طرح Register File باید اصلاح شود.

٤. PC نیاز به یک سیگنال کنترلی PI(PC Increment) دارد و لذا طرح قبلی باید اصلاح شود.

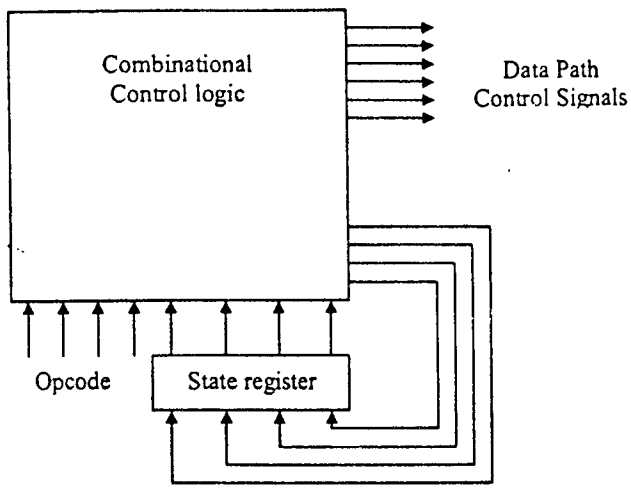
Control Unit کامپیوتر چندسیکل یک ماشین حالت(FSM) است که دیاگرام حالت یا ASM chart آن ابتدا باید مشخص شود تا بتوان آنرا طراحی کرد. در این آزمایش تنها 6 دستورالعمل را درنظرگرفته‌ایم لذا دیاگرام حالت به صورت زیر خواهد بود:



برای توصیف دقیق واحد کنترل باید مقادیر سیگنالهای کنترلی درون هرحالت (state) مشخص شود تا عملیات موردنظر اجرا گردد.

واحد کنترل را می توان به دو روش Hardwired و Microprogrammed پیاده‌سازی کرد. در این آزمایش واحد کنترل Hardwired بررسی می‌شود.

شکل کلی واحد کنترل Hardwired به صورت زیر است:



مدلسازی Program Counter(PC)

شمارندهٔ برنامه یک رجیستر16 بیتی با یک سیگنال کنترلی PI(PC Increment) است که اگر برابر1 باشد درلبهٔ بالارونده کلاک مقدار PC یک واحد افزایش می‌یابد.

- توصیف این ماجول را با نام **PC1** به زبانVerilog بنویسید.
- پروژه را **Save&Compile&Simulate** کنید.
- یک **symbol** مناسب برای ماجول طراحی کنید.

مدلسازی Instruction Register

رجیستردستورالعمل(IR) یک رجیستر16 بیتی با یک سیگنال کنترلی IL(Instruction Load) است که اگر برابر1 باشد درلبهٔ بالارونده کلاک ورودی(دستورالعمل) را درون خود ذخیره می‌کند.

- توصیف این ماجول را با نام **IR** به زبانVerilog بنویسید.
- پروژه را **Save&Compile&Simulate** کنید.
- یک **symbol** مناسب برای ماجول طراحی کنید.

اصلاح Register File

Register File کامپیوتر چند سیکل دارای 9 رجیستر16 بیتی است و لذا نیاز به خطوط آدرس 4 بیتی دارد. هرگاه یکی ازآدرسها برابر 1xxx باشد R8 به عنوان رجیستر موقت انتخاب خواهد شد.

- توصیف این ماجول را با نام **Regfile9x16** به زبانVerilog بنویسید.
- پروژه را Save&Compile&Simulate کنید.
- یک symbol مناسب برای ماجول طراحی کنید.

می‌توانید ماجولهای فوق را شبیه سازی کنید تا از صحت عملکرد آنها مطمئن شوید.

طراحی Control Unit ماشین Multi-Cycle

این ماجول را به کمک زبان Verilog مدلسازی می‌کنیم.

- یک فایل Verilogجدید با نام **Control_Unit.v** ایجادکنید.

ابتدا مقادیرپارامترها را بااستفاده ازدستور **define`** تعریف می‌کنیم اینکار باعث خواناتر شدن کدخواهد شد. دستور:

```
`define start 4'd0          //start=0000b
```

start را برابرعدد دسیمال 0 به طول 4 بیت قرارمی‌دهد. چون ماشین دارای 9 حالت است 4 بیت کافی خواهد بود.

- پارامترهای حالتها(state)را به صورت زیر تعریف نمایید.

```
//*********** State parameters
`define start 4'd0          //start=0000b
`define IF 4'd1             //IF=0001b
`define EX0 4'd2            //EX0=0010b
`define S_ADI 4'd3          //S_ADI=0011b
`define S_LD 4'd4           //S_LD=0100b
      ...
      ...
```

- پارامترهای opcode را به صورت زیر تعریف نمایید.

```
//*********** opcode parameters
`define ADI 7'b1000010
`define LD 7'b0110000
`define ST 7'b0100000
`define INC 7'b0000001
`define NOT 7'b0001110
`define ADD 7'b0000010
```

- پارامترهایFS را به صورت زیر تعریف نمایید.

```
//*********** Function Select parameters
`define transfer_A 5'b00000
`define inc_A 5'b00001
`define A_add_B 5'b00010
`define A_add_B_plus_1 5'b00011
`define A_add_not_B 5'b00100
`define A_sub_B 5'b00101
      ...
      ...
```

- ماجول رابامشخص‌کردن ورودیهاوخروجیها وتعیین نوع آنها تعریف‌کنید. خروجیها بایدازنوع**reg** تعریف شوند.

طراحی ماشین حالت نیاز به دو متغیر با نامهای **present_sate** و**next_state** دارد تا دنبالهٔ تغییرحالتها حفظ شود. **present_state** حالت فعلی که ماشین درآن قرار دارد را نشان می‌دهد و **next_state** حالت بعدی را که به آن گذر خواهد شد مشخص می‌کند.

- این دو متغیر را به صورت زیر تعریف کنید:

```
reg [3:0] present_state, next_state;    // variables to keep track of state transitions
```

- حالت اولیهٔ ماشین درلحظهٔ شروع**start** است. به‌کمک ساختار**initial** حالت اولیهٔ ماشین رابرابر**start** قراردهید.

```
initial
begin
      present_state=`start;
      next_state=`start;
end
```

در لبهٔ بالارونده کلاک حالت سیستم با توجه به opcode و حالتی که ماشین درآن قرار دارد عوض می‌شود.

- با استفاده از ساختار **always** به صورت زیر می توان این عملیات را پیاده سازی نمود:

```
always @(posedge CLK)                  // At positive edge of clock change state
      present_state=next_state;
```

هرگاه که حالت سیستم عوض می‌شود سیگنالهای کنترلی Data Path باید تغییر می‌کنند. همچنین حالت بعدی که ماشین به آن گذرخواهد کرد نیز باید مشخص شود.

- با استفاده از ساختار **always** و **case** به صورت زیر می‌توان این عملیات را انجام داد:

```
always @(present_state or opcode)
begin
      {MW, MM, RW, MD, MB, TB, TA, TD, PI, IL}=10'b0000000000;
      FS[4:0]=5'b0000;

      case (present_state)

            `start:
            begin
                  next_state=`IF;
            end

            `IF:
            begin
                  MM=1;
                  IL=1;
                  PI=1;
                  next_state=`EX0;
            end

            `EX0:
            begin
                  case (opcode)
                        `ADI:  next_state=`S_ADI;
                        `LD:   next_state=`S_LD;
                        `ST:   next_state=`S_ST;
                        `INC:  next_state=`S_INC;
                        `NOT:  next_state=`S_NOT;
                        `ADD:  next_state=`S_ADD;
                        default: next_state=`start;
                  endcase
            end

            `S_ADI:
            begin
                  MB=1;
                  RW=1;
                  FS[4:0]=`A_add_B;
                  next_state=`IF;
            end

            `S_LD:
            begin
                  MD=1;
                  RW=1;
                  next_state=`IF;
            end

            ...
            ...
            ...

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

داخل این کد **procedural**، هنگام تغییرحالت ابتدا تمامی سیگنالهای کنترلی غیرفعال می‌شوند سپس با توجه به حالت فعلی ماشین (**present_state**) سیگنالهای کنترلی لازم فعال شده ونهایتا حالت بعدی ماشین(**next_state**) مشخص می‌گردد. کد توصیف ماجول را به شیوهٔ فوق تکمیل کنید.

- پروژه را **Save&Compile&Simulate** کنید.
- یک **symbol** مناسب برای ماجول طراحی کنید.

طراحی ماشین Multi-Cycle

اکنون می‌توانید کامپیوترچندسیکل را مدلسازی نمایید.

- شماتیک Data Path و Control کامپیوتر چند سیکل را نظیر شکل ۱ کامل کنید.

برای اینکه سیستم درست کار کند لازم است که حافظهٔ ماشین آسنکرون باشد لذا نیازی به سیگنال کلاک ورودی نیست. برای آسنکرون کردن خواندن و نوشتن در حافظه پارامترهای این عنصر را به صورت زیر ست کنید:

- پارامترهای**inclock** و **outclock** حافظه را **unused** انتخاب کنید.
- پارامترهای LPM_ADDRESS_CONTROL ، LPM_INDATA و LPM_OUTDATA را برابر "UNREGISTERED" قرار دهید.

دقت کنید که خطوط آدرس حافظه نظیر شکل ۱، تنها شامل 8 بیت کم ارزش باس آدرس باشد. دراینجا نیزجهت سادگی طرح، پهنای باس آدرس برابر 8 بیت قرارداده شده است لذا طرح تنها دارای 256 کلمهٔ حافظه خواهد بود.

- پروژه را **Save&Check** کنید و درصورت وجود خطا آنها را رفع نمایید.
- پروژه را کامپایل **functional** کنید.

شبیه سازی ماشین Multi-Cycle

برنامهٔ اسمبلی زیردومقدار ثابت را درون رجیسترهای R1 وR2 قرار داده(با فرض مقادیر اولیهٔ رجیسترها برابر0) و آنها را باهم جمع می‌کند و نتیجه را دررجیستر R3 می نویسد:

```
ADI     R1,R0,5       // R1<-R0+5
ADI     R2,R0,3       // R2<-R0+3
ADD     R3,R2,R1      // R3<-R1+R2
```

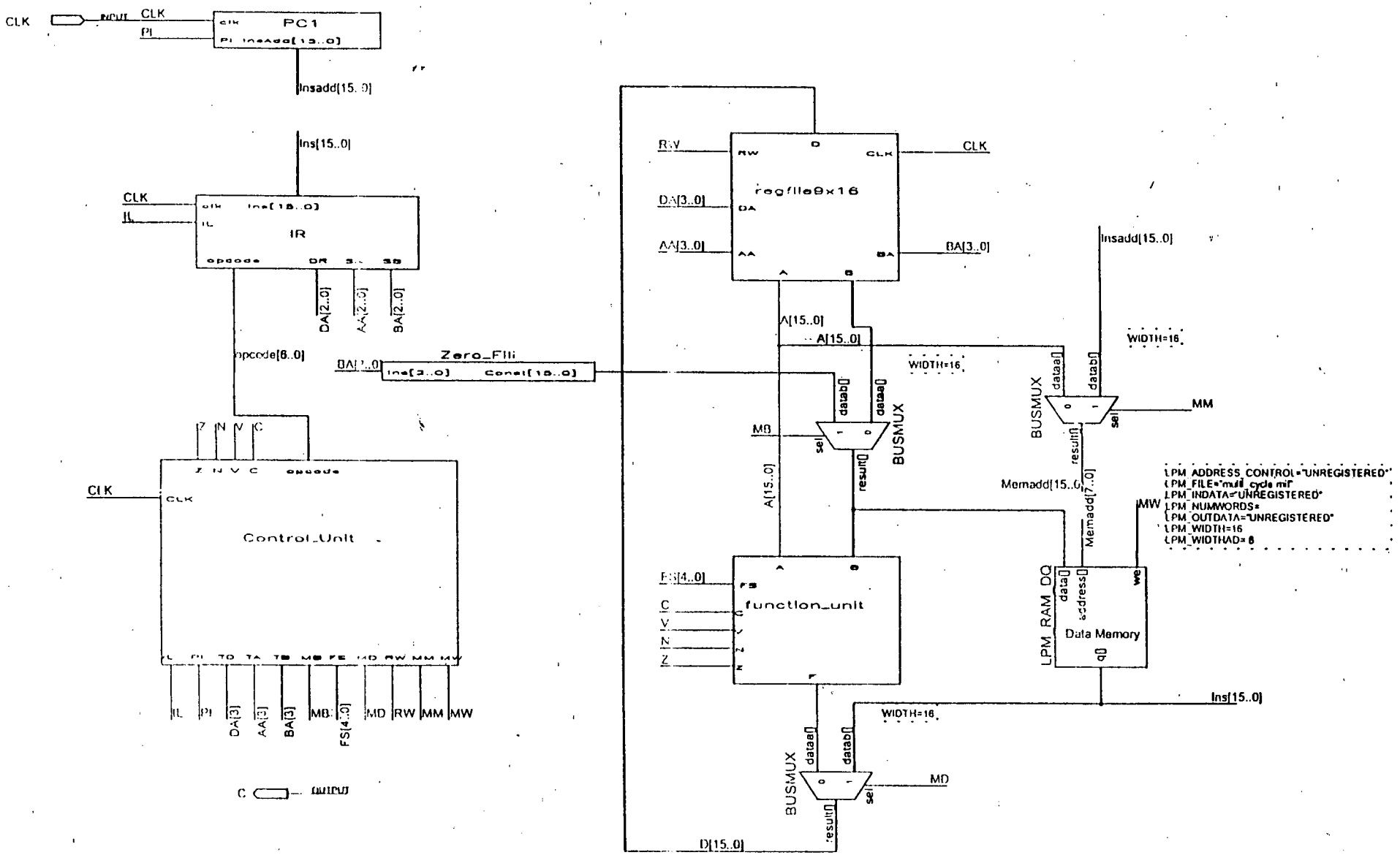
- کد زبان ماشین فوق را درون حافظه سیستم ذخیره کنید.
- پروژه را **Save&Simulate** کنید.
- با بررسی شکل موج و مقادیر رجیسترها در **Waveform Editor** درستی نتیجه را بررسی کنید.(بعد از 9 کلاک مقدار R3 باید برابر 8 باشد.)

گزارش کار:

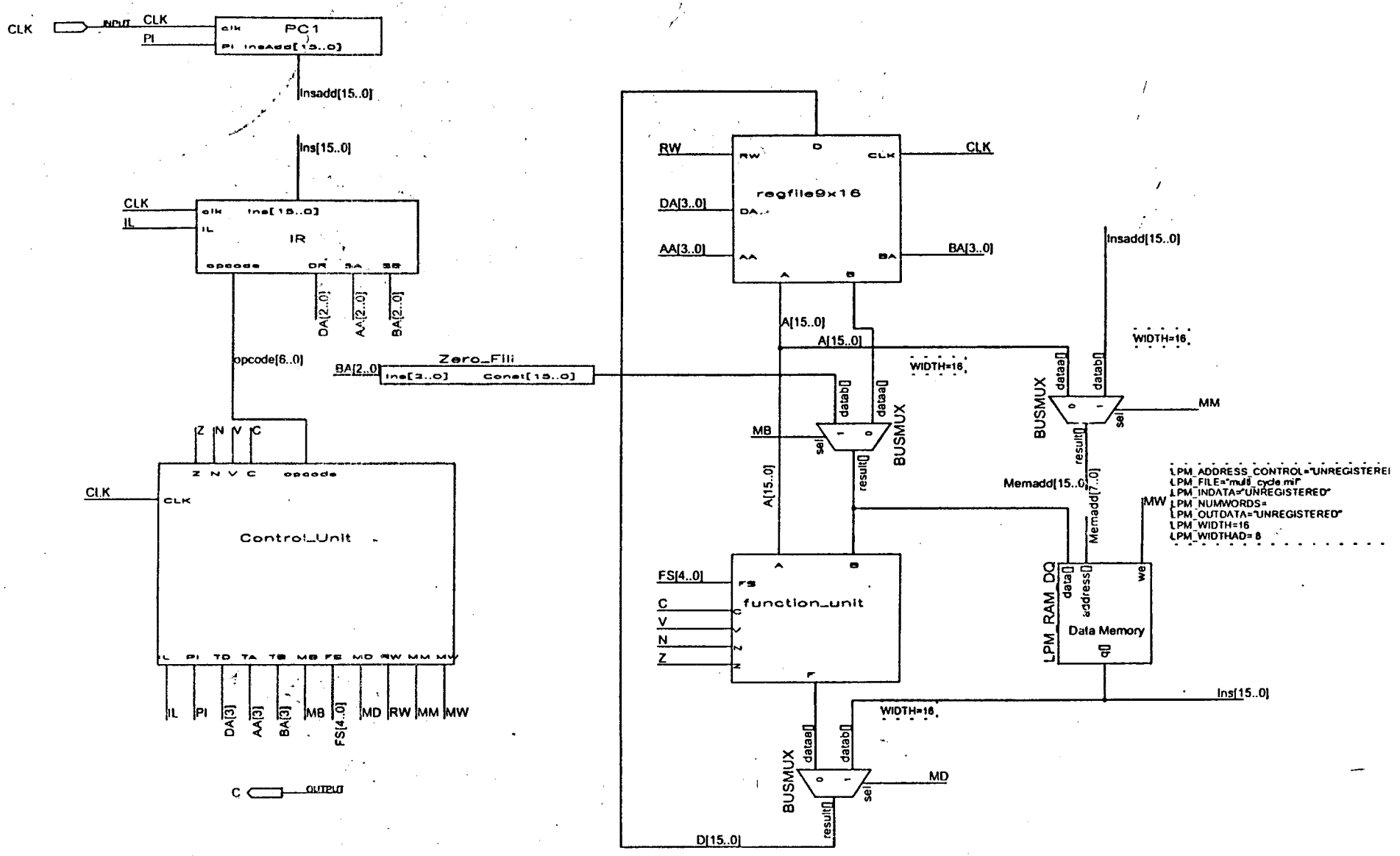
۱. تحویل نتایج شبیه سازی برنامهٔ اسمبلی فوق.

۲. Data Path وControl ماشین چندسیکل را اصلاح کنید تا دستورات BR(Branch) وBZ(Branch Zero) را نیز پشتیبانی نماید. ابتدا برای این دستورالعملها opcode درنظربگیریدوسپس دیاگرام حالت و توصیف واحد کنترل را اصلاح کنید.

۳. برنامه‌ای بنویسید که ماکزیمم مقداریک آرایهٔ 10 عنصری را پیدا کند. برنامه را بر روی ماشین اصلاح شدهٔ فوق اجرا و درستی عملیات را بررسی کنید.



Data Path & Control of Multi-cycle Computer



Data Path & Control of Multi-cycle Computer