طراحی ALU با استفاده از مدل‌سازی جریان داده (Dataflow) در VHDL

پارسا حجابی

چکیده

در این نوشتار گزارشی کامل از نحوه طراحی و پیاده‌سازی یک ALU ۱۶ بیتی یا همان واحد منطق و محاسبه در پردازنده با استفاده از مدل‌سازی جریان داده در زبان توصیف سخت‌افزار VHDL داده می‌شود و در آخر

کلمات کلیدی

ALU، VHDL، Dataflow،

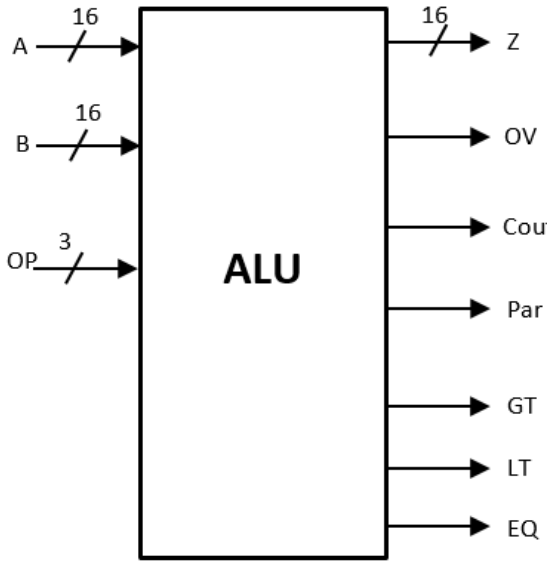
۱- مقدمه

ALU یا همان واحد حساب و منطق در پردازنده وظیفه انجام عملیات ریاضی مثل جمع و تفریق و عملیات منطقی مثل شیفت دادن به چپ یا راست دارد. این واحد معمولا به رجیستری مثل Data Register و Data Bus در پردازنده متصل است و بنابرنیاز ۲ یا ۳ و یا تعداد بیشتری ورودی دارد. همچنین هر ALU طوری طراحی می‌شود که بتواند چندین دستور را با ورودی‌های خودش انجام دهد و برای مشخص کردن نوع دستور یک ورودی به نام OP_Code دارد که بنابر تعداد دستورات ۲ یا ۳ و یا تعداد بیشتری بیت دارد.

ALUها غیر از حاصل عملیات حسابی و منطقی خروجی‌های بیشتری هم می‌توانند داشته باشند از جمله Flagهایی مانند Carry، Overflow، Parity و ... که هر کدام خروجی‌های ۱ بیتی هستند که جزئیات حاصل عملیات انجام شده را مشخص می‌کنند.

در این تمرین گزارشی کامل از طراحی یک ALU ۱۶ بیتی که به معنای این می‌باشد که ورودی‌ها و خروجی‌های اصلی آن ۱۶ بیتی هستند به همراه یک ورودی OP_Code ۳ بیتی که یعنی ۸ عملیات را می‌توان به کمک این ALU انجام داد را می‌دهیم.

در شکل (۱) می‌توان شکل کلی این ALU را به همراه ورودی‌ها و خروجی‌های آن مشاهده کرد.



شکل (۱): نمای کلی ALU مورد بحث به همراه ورودی‌ها و خروجی-های آن

همچینین در جدول (۱) لیست ۸ عملیاتی که قرار است با این ALU پیاده‌سازی شوند را می‌توان مشاهده کرد.

	Operation
0	Z = A nand B
1	Z = A nor B
2	Z = A xnor B
3	Z = not (A)
4	Z = A >> 1 (Arith)
5	Z = A << 1 (rotate)
6	Z = A + B
7	Z = A - B

جدول (۱): لیست عملیاتی که می‌توان به کمک این ALU انجام داد.

۲- مطالب اصلی

۲-۱- طراحی ALU

در ابتدا باید در فایل مخصوص ALU یک ENTITY جدید ایجاد کرد و پورت‌های ورودی و خروجی آن را کاملا مشخص نمود. برای نوشتن این تکه از کد دقیقا از روی شکل (۱) و ورودی و خروجی‌های آن کمک گرفتیم. برای خروجی‌های یک بیتی از STD_LOGIC و برای ورودی و خروجی‌های چند بیتی از STD_LOGIC_VECTOR استفاده کردیم [۱].

در ادامه یک ARCHITECTURE از ALU به نام alu_dataflow ایجاد کردیم که شامل ۳ سیگنال اصلی Z_temp و xor_out و Sum می‌باشد که به ترتیب برای مقدار دادن به سیگنال خروجی و محاسبه Parity به کمک XOR کردن بیت به بیت خروجی و محاسبه جمع یا تفریق‌ها استفاده شد. اولین سیگنال ۱۶ بیتی است به دلیل اینکه خروجی ۱۶ بیتی است. دومین سیگنال نیز ۱۶ بیتی است به دلیل اینکه برای محاسبه Parity باید خروجی اصلی را بیت به بیت XOR کرد و در نهایت آخرین مقدار به دست آمده همان Parity می‌باشد. سومین سیگنال ۱۷ بیتی است. علت وجود بیت اضافه انجام کامل عملیات جمع و به دست آوردن Overflow و Cout از روی آخرین بیت آن می‌باشد [۲].

درون ARCHITECTURE مقادیر خروجی محاسبه و مقداردهی شده‌اند. ابتدا Sum مقدار دهی شده است. برای محاسبه درست جمع و تفریق در صورتی که OP_Code برابر با ۶ یا ۷ شد ابتدا به ابتدای هر دو ورودی A و B یک بیت صفر Concat می‌کنیم و سپس عملیت جمع و تفریق را انجام می‌دهیم و بعدا از طریق بیت هفدهم این سیگنال مقدار Overflow و Cout را محاسبه می‌کنیم [۳].

در ادامه به کمک دستور WITH – SELECT مقادیر مختلفی را به ازای هر OP_Code در سیگنال Z_temp می‌نویسیم. برای این کار از جدول (۱) کمک می‌گیریم. برای دستور اول از دستور NAND، دستور دوم از دستور NOR، دستور سوم از دستور XNOR و برای دستور چهارم از دستور NOT کمک می‌گیریم که به طور پیش فرض در مدل‌سازی جریان داده وجود دارند. برای دستور پنجم با حفظ علامت شیفت به راست می‌دهیم. برای این کار بیت پانزدهم یا بیت علامت سیگنال ورودی A را با چهارده بیت دیگر Concat می‌کنیم. برای دستور ششم یا شیفت چرخشی بیت پانزدهم را از راست با چهارده بیت دیگر سیگنال ورودی Concat A می‌کنیم. برای دستور هفتم و هشتم کافی است ۱۶ بیت راست سیگنال Sum را که قبلا محاسبه کردیم در Z_temp قرار دهیم [۴].

در ادامه Overflow و Cout را به کمک بیت هفدهم سیگنال Sum که در گذشته مقداردهی شده بود محاسبه می‌کنیم. برای جمع Overflow برابر است با ۴ XOR مقدار بیت هفدهم و شانزدهم Sum و بیت شانزدهم سیگنال‌های ورودی A و B اما برای تفریق Overflow ۲ XNOR مقدار بیت شانزدهم Sum و بیت شانزدهم دومین عدد یا همان B در اینجا می‌باشد و AND این مقدار با ۲ XOR مقدار بیت شانزدهم A و B [5].

مقدار Cout نیز برابر است با بیت هفدهم Sum که محاسبه شده بود.

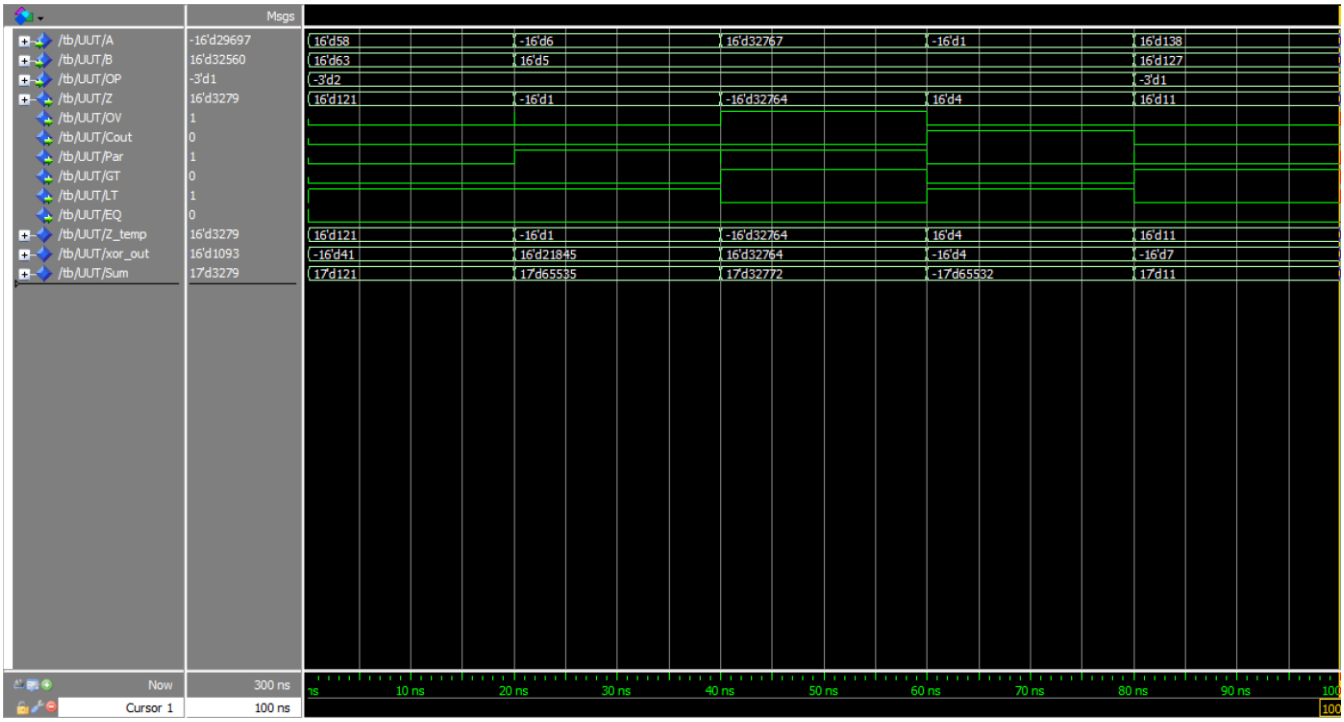
در ادامه مقدار Parity به همان نحوه‌ای که در اسلایدهای استاد مهدیانی به کمک FOR GENERATE موجود می‌باشد محاسبه می‌شود.

و در آخر مقدار نهایی Z_temp در سیگنال خروجی اصلی که همان Z باشد ریخته خواهد شد.

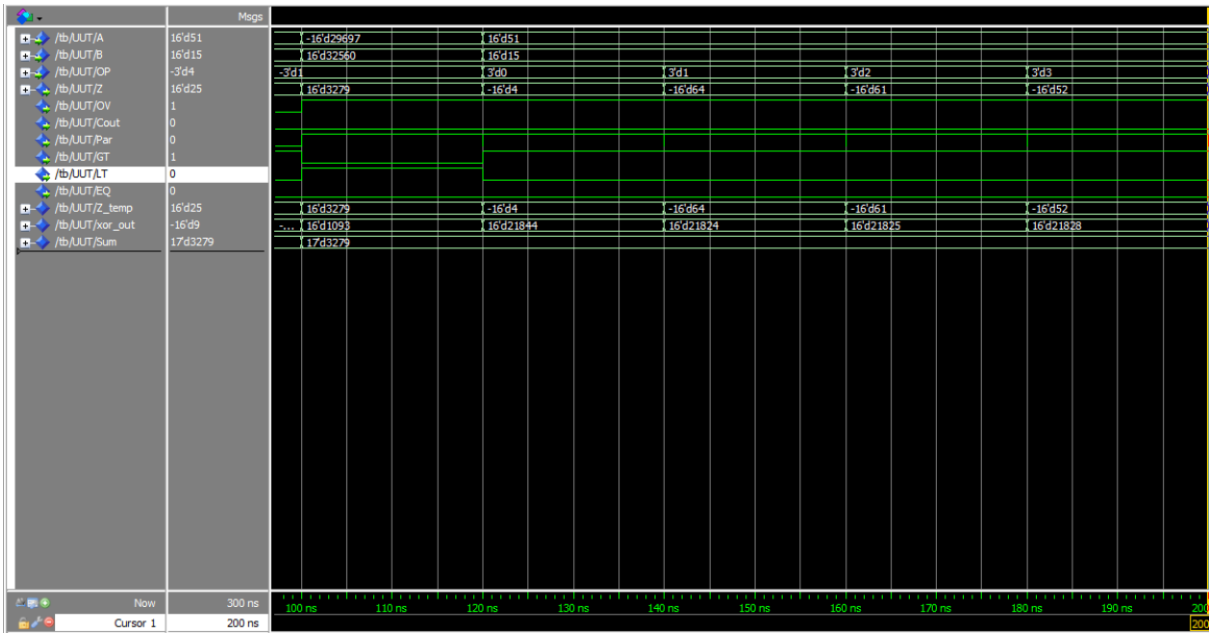
۲-۲- طراحی Testbench

در فایل مخصوص Testbench در ابتدا یک ENTITY کاملا خالی ساخته می‌شود و در ادامه ARCHITECTURE آن با نام alu_testbench ساخته شد.

در Process اصلی این Testbench هر ۲۰ نانوثانیه به سیگنال‌های ورودی ALU مقادیر مختلفی داده شد و خروجی در Wave که شکل‌های (۲) و (۳) را تشکیل می‌دهد مشاهده شد.



شکل (۲): ورودی ها و خروجی‌ها به همراه مقادیر سیگنال‌ها به صورت ده دهی تا ۱۰۰ نانوثانیه



شکل (۳): ورودی‌ها و خروجی‌ها به همراه مقادیر سیگنال‌ها به صورت ده دهی تا ۲۰۰ نانوثانیه

۳- نتیجه

پیاده‌سازی با استفاده از مدل‌سازی جریان داده بسیار سطح بالا و نزدیک به زبان انسان می‌باشد و همان‌طور که دیدیم به صورت جزئی به انجام عملیات به صورت سخت افزاری نپرداختیم.

در پیاده‌سازی نرم‌افزاری می‌توان با استفاده از waveها از درست کار کردن کد در سطح بالا مطمئن شد اما هیچ تضمینی وجود ندارد که کد سطح بالای ما در سنتز هم به طور صحیح کار کند.

مراجع

[1] alu.vhd file. lines: 8 to 21
[2] alu.vhd file. lines: 25 to 27
[3] alu.vhd file. lines: 31 to 32 & 44 to 48
[4] alu.vhd file. lines: 34 to 42
[5] https://www.doc.ic.ac.uk/~eedwards/compsys/arithmetic/index.html