به نام خدا

فرزان رحمانی ۹۹۵۲۱۲۷۱

گزارش تمرین سری سوم طراحی کامپیوتری سیستم های دیجیتال

سوال اول

توضیح کد پکیج:

کد من یک پکیج VHDL را پیاده‌سازی می‌کند که عملیات‌های ماتریسی اعداد مختلط را با استفاده از procedures و توابع پیاده‌سازی می‌کند. در ادامه، توضیحی از اجزای این پکیج آمده است:

- در بخش کتابخانه‌ها، کتابخانه‌های مورد نیاز برای پیاده‌سازی استفاده شده‌اند، از جمله **IEEE.STD_LOGIC_1164** برای استفاده از نوع داده **std_logic** و **IEEE.MATH_REAL** برای استفاده از نوع داده **real**.
- در بخش **package**، پکیج **complex_matrix_pkg** تعریف شده است.
- در این پکیج، نوع داده سفارشی **complex** تعریف شده است که دارای دو قسمت حقیقی و خیالی است.
- ثابت **matrix_size** به عنوان اندازه عمومی برای ابعاد ماتریس تعریف شده است.
- نوع داده **matrix** به کمک نوع داده **complex** تعریف شده است که یک آرایه از ابعاد **matrix_size × matrix_size** است.
- تابع **add_matrices** برای جمع دو ماتریس ورودی تعریف شده است. این تابع ماتریس خروجی را برمی‌گرداند.
- تابع **multiply_matrices** برای ضرب دو ماتریس ورودی تعریف شده است. این تابع ماتریس خروجی را برمی‌گرداند.
- رویه **fill_matrix_randomly** برای پر کردن یک ماتریس به صورت تصادفی تعریف شده است.
- رویه **print_matrix** برای چاپ ماتریس به صورت متنی تعریف شده است.
- متغیرهای مشترک **seed1** و **seed2** تعریف شده‌اند که مقدار اولیه آنها **999** است.
- در بخش **package body**، بدنه پکیج تعریف شده است.
- تابع **add_matrices** برای جمع دو ماتریس ورودی پیاده‌سازی شده است.
- تابع **multiply_matrices** برای ضرب دو ماتریس ورودی پیاده‌سازی شده است.
- رویه **fill_matrix_randomly** برای پر کردن یک ماتریس به صورت تصادفی پیاده‌سازی شده است. در این رویه از تابع **uniform** برای تولید اعداد تصادفی استفاده شده است.
- رویه **print_matrix** برای چاپ ماتریس به صورت متنی پیاده‌سازی شده است.
- در پکیج، نوع ماتریس با استفاده از نوع داده **complex**، توابع و رویه‌های مختلفی برای جمع، ضرب، پر کردن تصادفی و چاپ ماتریس‌ها ارائه می‌شود. ابعاد ماتریس با استفاده از ثابت **matrix_size** مشخص می‌شود و می‌توان آن را تغییر داد تا ماتریس‌های با ابعاد مختلف را پشتیبانی کند.

توضیح کد پیاده کننده پکیج:
کدی که از پکیج **complex_matrix_pkg**استفاده می‌کند، عملیات ماتریسی را با استفاده از توابع و رویه‌های ارائه شده در این پکیج انجام می‌دهد. این کد عملیات زیر را انجام می‌دهد:

۱. ابتدا پکیج **complex_matrix_pkg**را با استفاده از دستور **use**فراخوانی می‌کند.

۲. در بلاک **architecture**، اجزای مربوط به تست واحد (**unit under test**) تعریف می‌شوند. این اجزا شامل یک مولفه (**component**)با نام **binary_to_unary**است که دارای ورودی **binary**و خروجی **unary**است. سیگنال‌های مربوط به ورودی و خروجی نیز تعریف می‌شوند.

۳. در بلاک **begin**، مولفه **binary_to_unary**با استفاده از **port map**به سیگنال‌های ورودی و خروجی متناظر اتصال داده می‌شود.

۴. در بلاک **process**، الگوی تست برای مولفه **binary_to_unary**تعریف می‌شود. در این الگو، ابتدا منتظر می‌ماند تا ۱۰۰ نانوثانیه گذشته، سیگنال **binary**را برابر با "۰۰۰" قرار می‌دهد. سپس دوباره منتظر می‌ماند تا ۱۰۰ نانوثانیه گذشته و سیگنال **binary**را برابر با "۰۰۱" قرار می‌دهد. این فرآیند برای مقادیر دیگر **binary**تکرار می‌شود.

۵. در پایان، بلاک **end**این بلاک **process**و بلاک **architecture**را می‌بندد.

به طور خلاصه، کد فوق یک تست بنچ (**testbench**) برای مولفه **binary_to_unary**است که با استفاده از مقادیر مختلف برای سیگنال ورودی **binary**، عملکرد مولفه را تست می‌کند.

توضیح کد تست:
کد فوق یک تست بنچ (**testbench**) برای مولفه **matrix_operations**است که عملکرد آن را تست می‌کند. توضیحات مربوط به این تست بنچ به شرح زیر است:

۱. در بلاک **architecture**، اجزای مربوط به تست واحد (**unit under test**) تعریف می‌شوند. این اجزا شامل یک مولفه (**component**)با نام **matrix_operations**است که دارای ورودی‌های **add**، **mul**، **fill**، **print_A**، **print_B**، **print_Added**و **print_Multiplied**است. سیگنال‌های مربوط به ورودی و خروجی نیز تعریف می‌شوند.

۲. در بلاک **begin**، مولفه **matrix_operations**با استفاده از **port map**به سیگنال‌های ورودی و خروجی متناظر اتصال داده می‌شود.

۳. در بلاک **process**، الگوی تست برای مولفه **matrix_operations**تعریف می‌شود. در این الگو، با استفاده از توالی‌های مختلف از سیگنال‌های ورودی، عملکرد مولفه را تست می‌کند. ابتدا منتظر می‌ماند تا ۱۰۰ نانوثانیه گذشته و سیگنال **fill**را برابر با '۱' قرار می‌دهد تا ماتریس‌ها به طور تصادفی پر شوند. سپس متوالی سیگنال‌های **print_A**و **print_B**را برابر با '۱' قرار می‌دهد تا ماتریس A و B نمایش داده شوند. سپس سیگنال **add**را برابر با '۱' قرار داده و ماتریس‌ها جمع شوند. در ادامه، سیگنال **print_Added**را برابر با '۱' قرار می‌دهد تا ماتریس جمع شده نمایش داده شود. سپس سیگنال **mul**را برابر با '۱' قرار داده و ماتریس‌ها ضرب شوند. در پایان، سیگنال‌ها را به طور تصادفی مقداردهی می‌کند تا به حالت اولیه برگردند.

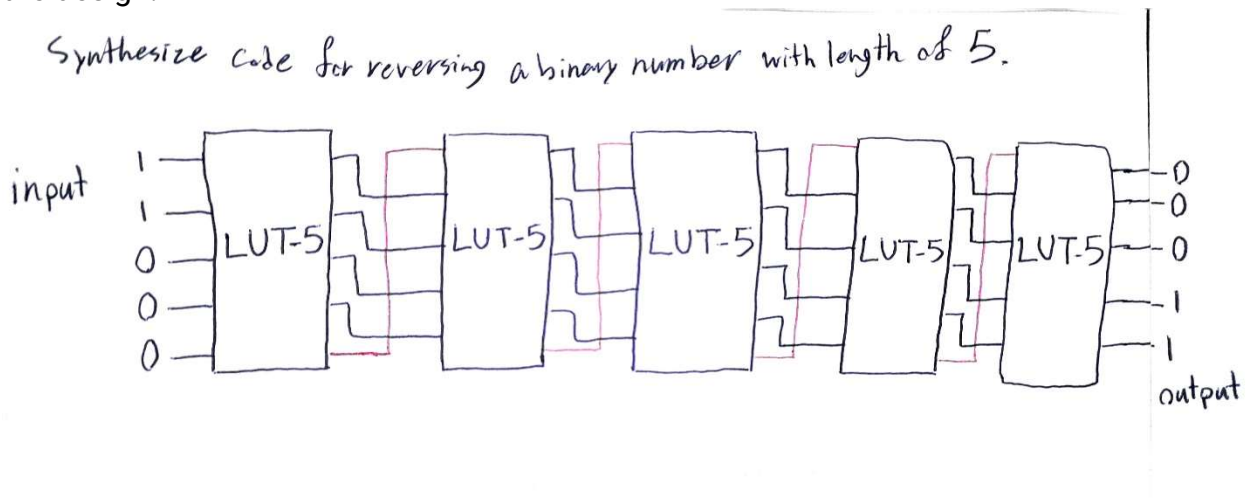
۴. در پایان، بلاک **end**این بلاک **process**و بلاک **architecture**را می‌بندد.

۵. در بخش **configuration**، پیکربندی **TESTBENCH_FOR_matrix_operations**برای تست بنچ تعریف شده است. این پیکربندی مشخص می‌کند که مولفه **matrix_operations**از نوع **behavioral**استفاده شود.

به طور خلاصه، کد فوق یک تست بنچ برای مولفه **matrix_operations**است که با استفاده از مقادیر مختلف برای سیگنال‌های ورودی، عملکرد مولفه را تست می‌کند و نتایج را در سیگنال‌های خروجی مشاهده می‌کند.

سوال دوم

For binary reverser function, consider a scenario that we want to reverse a binary number with length of 5. Synthesize code for such scenario and draw schematic view of the design.



توضیح کد پکیج:

کد من یک پکیج با نام `recursive_functions` تعریف می‌کند که شامل دو تابع است: `bcd_to_binary` و `binary_reverser`.

۱. تابع `bcd_to_binary`:

- تابع `bcd_to_binary` یک ورودی به نام `bcd_num` از نوع `std_logic_vector` دریافت می‌کند و یک خروجی از نوع `unsigned` برمی‌گرداند.

- این تابع برای تبدیل اعداد BCD (Binary-Coded Decimal) به اعداد باینری استفاده می‌شود.

- الگوریتم تابع به این صورت است:

* در صورتی که طول `bcd_num` برابر با ۴ باشد، خروجی برابر با `۰۰۰۰` است.

* در غیر این صورت:

- تابع `bcd_to_binary` را با زیرمجموعه `bcd_num` با طول `(bcd_len-1)-4` فراخوانی می‌کند.

- مقدار `bcd_val` را با جمع عدد BCD موجود در `bcd_num` با استفاده از تابع `to_integer(unsigned(bcd_num(bcd_len - 1 downto bcd_len - 4)))` در جایگاه مناسب قرار می‌دهد. این جمع بر اساس توان ۱۰ و تعداد رقم‌های باینری قبلی انجام می‌شود (`۱۰**(4 / (bcd_len-4))`).

- نتیجه را به عنوان خروجی تابع برمی‌گرداند.

۲. تابع `binary_reverser`:

- تابع `binary_reverser` یک ورودی به نام `Input` از نوع `std_logic_vector` دریافت می‌کند و یک خروجی از نوع `std_logic_vector` برمی‌گرداند.

- این تابع برای برعکس کردن عدد باینری ورودی استفاده می‌شود.

- الگوریتم تابع به این صورت است:

* در صورتی که طول `Input` برابر با ۱ باشد، خروجی برابر با `Input` است.

* در غیر این صورت:

- تابع `binary_reverser` را با زیرمجموعه `Input` با طول `Input'length-2` فراخوانی می‌کند.

- برای خروجی `Result`، بیت آخر `Input` را در جایگاه اول قرار می‌ده

د و بقیه بیت‌ها را به ترتیب برعکس می‌کند.

- نتیجه را به عنوان خروجی تابع برمی‌گرداند.

توضیح کد پیاده سازی پکیج:

کد من یک ماژول با نام `q2` را پیاده‌سازی می‌کند که از پکیج `Recursive_Functions_Pkg` استفاده می‌کند و دو ورودی (`bin_in` و `bcd_in`) و دو خروجی (`bin_out` و `bcd_out`) دارد.

۱. ورودی و خروجی `bin_in` و `bin_out`:

- `bin_in` یک ورودی از نوع `std_logic_vector` با طول ۵ بیت است که عدد باینری ورودی را نمایش می‌دهد.

- `bin_out` یک خروجی از نوع `std_logic_vector` با طول ۵ بیت است که عدد باینری برعکس شده را نمایش می‌دهد.

- فرآیند `process` بر اساس مقدار ورودی `bin_in`، اگر مقدار آن مشخص نباشد (`"UUUUU"`)، عدد باینری برعکس شده را به عنوان خروجی `bin_out` قرار می‌دهد.

۲. ورودی و خروجی `bcd_in` و `bcd_out`:

- `bcd_in` یک ورودی از نوع `std_logic_vector` با طول ۸ بیت است که عدد BCD ورودی را نمایش می‌دهد.

- `bcd_out` یک خروجی از نوع `std_logic_vector` با طول ۸ بیت است که عدد باینری تبدیل شده از عدد BCD ورودی را نمایش می‌دهد.

- فرآیند `process` بر اساس مقدار ورودی `bcd_in`، اگر مقدار آن مشخص نباشد (`"UUUUUUUU"`)، عدد باینری تبدیل شده از عدد BCD را به عنوان خروجی `bin_out` قرار می‌دهد.

بنابراین، این ماژول از پکیج `Recursive_Functions_Pkg` استفاده می‌کند تا توابع `binary_reverser` و `bcd_to_binary` را برای برعکس کردن عدد باینری و تبدیل عدد BCD به باینری پیاده‌سازی کند.

توضیح کد تست:

کد فوق یک تست‌بنچ برای ماژول `q2` ایجاد می‌کند و عملکرد آن را تست می‌کند. تست‌بنچ، ورودی‌ها و خروجی‌های ماژول را مشخص می‌کند و مقادیر ورودی را به ماژول می‌دهد. سپس نتایج را بررسی می‌کند.

این تست‌بنچ شامل موارد زیر است:

۱. تعریف کامپوننت: تست‌بنچ کامپوننت `q2` را تعریف می‌کند که ورودی‌ها و خروجی‌های آن را شامل می‌شود.

۲. سیگنال‌ها: سیگنال‌های مورد استفاده برای ورودی‌ها و خروجی‌های ماژول را تعریف می‌کند. این سیگنال‌ها شامل `bin_in` (ورودی عدد باینری)، `bin_out` (خروجی برعکس شده عدد باینری)، `bcd_in` (ورودی عدد BCD) و `bcd_out` (خروجی تبدیل شده عدد BCD به باینری) است.

۳. ماژول زیر تست‌بنچ: ماژول `q2` را با استفاده از `UUT` تعریف می‌کند و ورودی‌ها و خروجی‌ها را به آن وصل می‌کند.

۴. فرآیند تست: در این فرآیند، مقادیر ورودی `bin_in` و `bcd_in` را تنظیم می‌کند. ابتدا مقادیر اولیه به عنوان ورودی‌ها ارسال می‌شوند (`"۱۱۰۰۰"` برای `bin_in` و `"۰۰۰۱۰۰۱۰"` برای `bcd_in`) و سپس پس از گذشت زمان (`wait for 100ns`)، مقادیر دیگری به عنوان ورودی‌ها ارسال می‌شوند (`"۱۰۱۰۱"` برای `bin_in` و `"۰۰۱۱۰۰۱۱"` برای `bcd_in`) و در نهایت منتظر می‌ماند (`wait`) تا پایان تست.

۵. تنظیمات تست‌بنچ: تنظیمات مورد نیاز برای اجرای تست را تعیین می‌کند، از جمله

استفاده از نسخه سنتز شده (`synth`) از ماژول `q2`.

با استفاده از این تست‌بنچ، عملکرد ماژول `q2` با ورودی‌های مختلف مانند `"۱۱۰۰۰"` و `"۱۰۱۰۱"` برای `bin_in` و `"۰۰۰۱۰۰۱۰"` و `"۰۰۱۱۰۰۱۱"` برای `bcd_in` بررسی می‌شود. نتایج این تست بر اساس خروجی‌های متناظر (`bin_out` و `bcd_out`) بررسی می‌شود.

سوال سوم

توضیح کد سوال:
کد من شامل سه بخش است که به توضیحات زیر تقسیم می‌شود:

1. بسته `image_filter_pkg`:

در این بسته، مقادیر ثابت و نوع‌های داده مورد استفاده در ماژول‌های دیگر تعریف می‌شود. مقادیر ثابت شامل اندازه تصویر (`IMAGE_SIZE_X` و `IMAGE_SIZE_Y`)، اندازه فیلتر (`FILTER_SIZE`) و تعداد واحدها (`MAX_UNITS`) است. نوع‌های داده تعریف شده شامل `pixel_t` (برای مقادیر پیکسل تصویر) و `filter_coeff_t` (برای ضریب‌های فیلتر) است. همچنین، نوع‌های آرایه `image_t` (برای نگهداری تصویر) و `filter_t` (برای نگهداری فیلتر) تعریف می‌شوند.

2. ماژول `convolution_unit`:

این ماژول واحد کانوولوشن را تعریف می‌کند. ورودی‌های آن شامل سیگنال کلاک (`clk`)، سیگنال ریست (`reset`)، پنجره تصویر (`window`) و فیلتر (`filter`) است. خروجی آن نیز شامل نتیجه کانوولوشن (`result`) است. در طرح این ماژول، با دریافت سیگنال کلاک و ریست، نتیجه کانوولوشن اعمال شده بر روی پنجره تصویر و فیلتر محاسبه و در خروجی (`result`) قرار می‌گیرد. عملکرد کانوولوشن شامل محاسبه مقدار `tmp` که نتیجه جمع ضرب عناصر پنجره تصویر و فیلتر است و سپس تقسیم آن بر ۹ است.

3. ماژول `main_component`:

این ماژول شامل کامپوننت اصلی است که واحدهای کانوولوشن را نمونه‌برداری می‌کند و آن‌ها را به هم متصل می‌کند. این ماژول تعداد واحدهای کانوولوشن را با استفاده از جنریک `N` دریافت می‌کند. ورودی‌های آن شامل

سیگنال کلاک (`clk`)، سیگنال ریست (`reset`)، تصویر ورودی (`image_in`)، فیلتر ورودی (`filter_in`) و نقشه ویژگی (`feature_map`) است. در طرح این ماژول، ابتدا واحدهای کانوولوشن نمونه‌برداری می‌شوند و سپس با استفاده از سیگنال کلاک و ریست، تصویر ورودی به پنجره‌های مختلف تقسیم می‌شود و نتیجه کانوولوشن در نقشه ویژگی قرار می‌گیرد. در انتها، نقشه ویژگی در یک فایل متنی ذخیره می‌شود.

بسته `image_filter_pkg`:
کد فوق یک بسته با نام `image_filter_pkg` تعریف می‌کند که اندازه تصویر، فیلتر، انواع داده، توابع و رویه‌ها را تعریف می‌کند. توضیحات جزئی‌تر در ادامه آمده است:

- `IMAGE_SIZE_X` و `IMAGE_SIZE_Y` تعیین کننده ابعاد تصویر هستند و به صورت ثابت در نظر گرفته شده‌اند و مقدار آنها برابر با ۳۰ است.

- `FILTER_SIZE` تعیین کننده ابعاد فیلتر است و نیز به صورت ثابت تعریف شده است و مقدار آن برابر با ۳ است.

- `MAX_UNITS` تعداد واحدهای مجاز برای استفاده در برنامه را مشخص می‌کند و نیز به صورت ثابت تعریف شده است و مقدار آن برابر با ۲۸ است (که می‌تواند ۱، ۲، ۴، ۷، ۱۴ یا ۲۸ باشد).

- `pixel_t` و `filter_coeff_t` زیرنوع‌های داده را تعریف می‌کنند. `pixel_t` برای نگهداری مقادیر پیکسل‌ها در تصویر (بین ۰ تا ۲۵۵) و `filter_coeff_t` برای نگهداری ضرایب فیلتر (بین ۰/۰ تا ۱/۰) است.

- `image_t` نوع داده‌ای آرایه را تعریف می‌کند که برای نگهداری تصویر استفاده می‌شود. این آرایه اندازه‌گیری با ابعاد `IMAGE_SIZE_X` در `IMAGE_SIZE_Y` دارد.

- `filter_t` نوع داده‌ای آرایه را تعریف می‌کند که برای نگهداری فیلتر استفاده می‌شود. این آرایه اندازه‌گیری با ابعاد `FILTER_SIZE` در `FILTER_SIZE` دارد.

اطلاعات و توابع و رویه‌های اضافی می‌توانند در این بسته تعریف شوند، که در این مورد در کد ارائه شده تعریف دیگری وجود ندارد.

ماژول `convolution_unit`:
این کد یک واحد کانولوشن (convolution unit) را تعریف و پیاده‌سازی می‌کند. واحد کانولوشن ورودی‌هایی مانند سیگنال ساعت (`clk`)، سیگنال تنظیم (`reset`)، پنجره (`window`) و فیلتر (`filter`) را می‌گیرد و خروجی محاسبه شده را در سیگنال `result` تولید می‌کند.

در معماری واحد کانولوشن (`behavioral`)، یک فرآیند با ورودی‌های `clk` و `reset` تعریف شده است. در این فرآیند، یک متغیر موقت به نام `tmp` از نوع `real` با مقدار اولیه ۰/۰ تعریف می‌شود. سپس با استفاده از یک ساختار شرطی، وقتی سیگنال `reset` برابر با '۱' است، مقدار خروجی (`result`) برابر با ۰/۰ قرار داده می‌شود. در غیر این صورت، در هر لبه صعودی (`rising_edge`) سیگنال `clk`، محاسبات کانولوشن انجام می‌شود.

محاسبات کانولوشن در داخل دو حلقه `for` انجام می‌شود. حلقه اول `i` را از ۰ تا ۲ و حلقه دوم `j` را نیز از ۰ تا ۲ می‌گرداند. در هر مرحله از حلقه، مقدار پنجره در موقعیت (`i`, `j`) با ضرب مقدار فیلتر متناظر در همان موقعیت، به متغیر موقت `tmp` اضافه می‌شود.

سپس مقدار `tmp` تقسیم بر ۹/۰ شده و در نهایت در سیگنال خروجی `result` قرار داده می‌شود. این محاسبه معمولاً با استفاده از ماتریس پنجره و ماتریس فیلتر برای اعمال یک عمل کانولوشن ساده استفاده می‌شود.

به طور خلاصه، این کد یک واحد کانولوشن را تعریف و پیاده‌سازی می‌کند که با دریافت پنجره و ف

یلتر، مقدار خروجی را با استفاده از عمل کانولوشن محاسبه می‌کند.

ماژول `main_component`:
این کد یک مولفه اصلی (main component) را تعریف و پیاده‌سازی می‌کند که شامل ایجاد و اتصال واحدهای کانولوشن (convolution units) است.

در قسمت انتیتی (entity) مولفه اصلی، یک پارامتر عمومی به نام `N` تعریف شده است که پیش‌فرض آن برابر ۲۸ است. این مولفه دارای ورودی‌های `clk` (سیگنال ساعت)، `reset` (سیگنال تنظیم)، `image_in` (تصویر ورودی)، `filter_in` (فیلتر ورودی) و خروجی `feature_map` (نقشه ویژگی) است.

در معماری ساختاری (structural) مولفه اصلی، ابتدا واحدهای کانولوشن (convolution units) را تعریف و به صورت موازی (parallel) ایجاد می‌کنیم. این واحدها با استفاده از کامپوننت `convolution_unit` تعریف شده‌اند و با استفاده از اتصال پورت‌ها (`port map`) به ورودی‌ها و خروجی‌های مولفه اصلی متصل می‌شوند. سیگنال `CU_outputs` نیز برای دریافت خروجی‌های محاسبه شده در هر واحد کانولوشن ایجاد می‌شود.

در فرآیند اصلی (`process`)، با استفاده از سیگنال `clk` و `reset`، عملکرد مولفه اصلی تعریف می‌شود. در صورتی که سیگنال `reset` برابر با '۱' باشد، مقادیر متغیرهای `current_index` (شاخص فعلی) و `feature_map` (نقشه ویژگی) را صفر قرار می‌دهد. در غیر این صورت، با هر لبه صعودی سیگنال `clk`، مراحل کانولوشن را اجرا می‌کند.

در ابتدا، پنجره x۳۳ برای موقعیت فعلی را از تصویر ورودی استخراج می‌کند و در سیگنال `win` قرار می‌دهد. سپس نتیجه به دست آمده را در نقشه ویژ

گی (`feature_map`) ذخیره می‌کند و شاخص فعلی را به روزرسانی می‌کند. در صورتی که شاخص فعلی به انتهای آرایه `CU_outputs` برسد، به شاخص اول برمی‌گردد. این عملکرد باعث می‌شود تا تمام واحدهای کانولوشن به ترتیب و به صورت پیوسته کانولوشن را بر روی تصویر انجام دهند.

سپس، یک فرآیند دیگر برای ذخیره نقشه ویژگی در یک فایل متنی با پسوند TXT ایجاد می‌شود. با استفاده از پیش‌تعریف شده `std.textio`، یک فایل متنی با نام "feature_map.txt" باز می‌شود. سپس نقشه ویژگی در فایل ذخیره می‌شود، به طوری که هر مقدار در خط جداگانه قرار می‌گیرد. در انتها، فایل بسته می‌شود.

به این ترتیب، کد مذکور نقشه ویژگی را با استفاده از واحدهای کانولوشن محاسبه کرده و نتیجه را در یک فایل متنی ذخیره می‌کند.

توضیح کد تست:
کد زیر یک تست برای ماژول `main_component` انجام می‌دهد. این کد شامل دو بخش است: بخش اصلی کد تست و بخش فراخوانی ماژول تحت آزمون.

در بخش اصلی کد تست، ابتدا ماژول تحت آزمون (`main_component`) با استفاده از `generic map` و `port map` فراخوانی می‌شود و پارامترها و سیگنال‌ها به پورت‌های ماژول منتقل می‌شوند.

سپس دو فرآیند (process) تعریف می‌شوند. در فرآیند اول، سیگنال `clk` به ترتیب به مقدار '۰' و '۱' تغییر می‌کند. این فرآیند برای تولید سیگنال کلاک برای ماژول تحت آزمون استفاده می‌شود.

در فرآیند دوم، سیگنال `reset` ابتدا به مقدار '۱' تنظیم می‌شود و پس از گذشت ۱۰۰ns به مقدار '۰' تغییر می‌کند. همچنین سیگنال‌های `image_in` و `filter_in` به ترتیب با الگوهای داده شده مقداردهی می‌شوند. این الگوها شامل مقادیری برای ماتریس تصویر و ماتریس فیلتر هستند.

در این کد، مقادیر تصویر و فیلتر به صورت ثابت تعریف شده‌اند و به ماژول تحت آزمون منتقل می‌شوند. الگوهای داده شده برای تصویر و فیلتر می‌توانند تغییر کنند و بر اساس آن نتیجه‌ی خروجی ماژول تحت آزمون بررسی می‌شود.

پس از اجرای این کد تست، خروجی‌های ماژول `main_component` که شامل سیگنال `feature_map` است، بررسی می‌شود تا نتیجه‌ی درستی از عملکرد ماژول تحت آزمون به دست آید.

پایان