Microeletrônica Digital

Identificador de sequência

Escopo

Esta atividade consiste nas seguintes etapas:

1. Implemente um circuito digital, de preferência em Verilog/SystemVerilog, que identifique a presença dos 4 bits menos significativos da sequência de bits referente ao código ASCII da primeira letra do seu nome recebidos serialmente. Exemplo: se seu nome for Gutemberg, a letra G possui código ASCII 0x47, ou seja, em binário teríamos a sequência 0111 a ser identificada;
2. Simule o circuito implementado. Utilize `$display` para visualizar o resultado. Você pode utilizar qualquer ferramenta. Por exemplo: EDA Playground, Icarus Verilog, Verilator, etc.;
3. Modifique o circuito digital para que este identifique a presença de uma sequência de 4 bits quaisquer em um conjunto de bits recebidos serialmente. A sequência a ser identificada será definida pelo usuário através de uma entrada específica de 4 bits (paralela).
4. Simule o circuito implementado e utilize $display para visualizar o resultado.
5. Utilize $dumpfile para gerar um arquivo .vcd e visualize as formas de ondas geradas. Você pode utilizar qualquer ferramenta. Por exemplo: EDA Playground, GTKWave, etc.

Extras

- Aprenda a gerar valores aleatórios usando `$urandom()` e `$urandom_range()`
    - Veja o padrão SystemVerilog IEEE 1800-2017 [1], seção 18.13.
- Aprenda a fazer um loop [2] e gere vários valores aleatórios.
- Realize o algoritmo em alto nível em SystemVerilog e compare com os valores gerados.

Referências:

[1] https://ieeexplore.ieee.org/document/8299595
[2] http://asic-world.com/systemverilog/procedure_ctrl1.html#Loop_statements

Ordenadores

Escopo

1. Esta atividade consiste nas seguintes etapas:
2. Implemente um circuito digital que ordene 8 números de 8 bits sem sinal;
3. Realize a simulação do circuito projetado e utilize `$display` para visualizar os dados de saída;
4. Implemente um circuito digital que realiza a filtragem da mediana de 9 entradas de 8 bits;
5. Realize a simulação do circuito projetado e utilize `$display` para visualizar os dados de saída.

Extras

- Aprenda a gerar valores aleatórios usando `$urandom()` e `$urandom_range()`
    - Veja o padrão SystemVerilog IEEE 1800-2017 [1], seção 18.13.
- Aprenda a fazer um loop [2] e gere vários valores aleatórios.
- Realize o algoritmo em alto nível em SystemVerilog e compare com os valores gerados.
    - A saída deve ser apenas "OK" se todos os testes passarem ou "ERRO - entradas xx xx xx - saída yy" caso algum teste dê errado.

Referências:

[1] https://ieeexplore.ieee.org/document/8299595
[2] http://asic-world.com/systemverilog/procedure_ctrl1.html#Loop_statements

Implementação RTL de um módulo de FFT

Especificação

- FFT de tamanho 16;
- Entrada e saída complexas;
- Módulo recebe 4 amostras por vez;
- Saída paralela com 16 amostras;
- Usar as seguintes fatorações:
    - 16 = 4 x 4;
    - 8 = 2 x 4.
- Latência máxima de 10 ciclos a partir da última entrada;
- Capaz de processar uma entrada por ciclo (1 FFT a cada 4 ciclos);

Interface

Em anexo (última página) está a casca (stub) do módulo a ser implementado.

- Dados de Entrada:
    - Array de 4 números complexos;
    - Cada número complexo é um array de tamanho 2: parte real (índice 0) e parte imaginária (índice 1);
    - Cada coordenada tem 8 bits que devem ser interpretados como ponto fixo em complemento de 2, sendo
        - 1 bit inteiro (o de sinal);
        - 7 bits fracionários;
        - O intervalo representado é, portanto: [-1; 1), com precisão 2^(-7).

- Dados de Saída:
    - Array de 16 números complexos
    - Cada coordenada terá 12 bits que devem ser interpretados como ponto fixo em complemento de 2 com
        - 8 bits inteiros (incluindo o de sinal);
        - 4 bits fracionários;
        - O intervalo representado é portanto [-128; 128) com precisão 2^(-4).
    - Para uma entrada uniformemente distribuída no disco unitário, cada coordenada de saída deve apresentar um erro médio inferior a 2^(-2) (4 unidades).

- Protocolo:
    - Sinal `i_valid` indica uma nova entrada a ser processada.
    - A cada quatro entradas, o módulo deve produzir uma saída, correspondendo à transformada dessas entradas concatenadas.
    - Sinal de saída `o_valid` indica uma nova saída calculada.

    - Portanto, para cada 4 ciclos em que `i_valid` está ativo, `o_valid` deve subir por um ciclo.
    - Reset síncrono, nível baixo ativo.

Bônus

Há várias tarefas que estendem ou complementam o desafio proposto. Elas também serão levadas em consideração. Neste caso, documentar o que foi feito.

Alguns exemplos:

- Tamanho da FFT e outros parâmetros configuráveis.
- Parâmetro ou entrada para seleção entre FFT e IFFT.
- Implementação em software do algoritmo.
- Testbench para o módulo.
    - Baseado a transações;
    - Auto-verificável;
    - Estímulos aleatórios;
    - Orientado a cobertura.
- Exploração arquitetural:
    - Comparação de desempenho entre duas arquiteturas alternativas.
    - Comparação de desempenho para vários valores de algum parâmetro do projeto.
    - (Análises empíricas ou teóricas.)

Referências

- https://dspguru.com/dsp/faqs/fft/

Avaliação

Soluções parciais também serão consideradas; por exemplo, implementações que desviem de algum ponto da especificação. Neste caso, documentar o que foi implementado e em que seu projeto difere da proposta.

De fato, pode ser uma boa ideia começar implementando o que seja mais simples para você, e depois tentar alinhar ao que foi pedido.

Deverão ser entregues os arquivos com código-fonte de sua autoria, mais um relatório. Por exemplo, se você escreveu um gerador de código para implementar o módulo RTL, submeter o gerador.

O relatório deve conter *impreterivelmente*

1. Lista de fontes que você usou para elaborar o trabalho, incluindo ferramentas, artigos, textos ou códigos de terceiros;
2. Instruções sobre como testar o código entregue (de preferência pela linha de comando);

3. Explicação do que foi feito;
4. Considerações que você julgar relevantes.

Para a tarefa principal, serão julgados aspectos como:

1. Atendimento das especificações;
2. Frequência máxima do relógio;
3. Custo de potência;
4. Generalidade do código;
5. Clareza do código e da documentação.

Anexo

```
module fft_16_4 #(
      int INPUT_WIDTH = 8,
      int OUTPUT_WIDTH = INPUT_WIDTH + 4
)
(
      output logic o_valid,
      output logic signed [OUTPUT_WIDTH-1:0] o_data[16][2],

      input logic i_valid,
      input logic signed [INPUT_WIDTH-1:0] i_data[4][2],

      input logic clk,
      input logic rst_sync_n
);
// seu design aqui
endmodule
```

Log2

Escopo

Esta atividade consiste nas seguintes etapas:

1. Implemente um circuito digital que calcule o valor de Log2;
2. Simule o circuito digital implementado e utilize `$display` para exibir o valor calculado.

Para esta atividade, considere uma entrada de 8 bits. O resultado deve ser apresentado em ponto-fixo, indicando a base utilizada.

Extras

- Aprenda a gerar valores aleatórios usando `$urandom()` e `$urandom_range()`
    - Veja o padrão SystemVerilog IEEE 1800-2017 [1], seção 18.13.
- Aprenda a fazer um loop [2] e gere vários valores aleatórios.
- Realize o algoritmo em alto nível em SystemVerilog e compare com os valores gerados.
    - A saída deve ser apenas "OK" se todos os testes passarem ou "ERRO - entradas xx xx xx - saída yy" caso algum teste dê errado.

Referência:

[1] https://ieeexplore.ieee.org/document/8299595
[2] http://asic-world.com/systemverilog/procedure_ctrl1.html#Loop_statements
[3] https://www.dsprelated.com/showthread/comp.dsp/123448-1.php
[4] http://www.ijsps.com/uploadfile/2014/1210/20141210051242629.pdf

Decodificador Huffman

A codificação de Huffman é um código de comprimento variável que utiliza a probabilidade de ocorrência dos símbolos no conjunto de dados para definir seu comprimento (número de bits). Ele é usualmente utilizado para compressão de dados e utiliza uma árvore binária completa no processo de codificação/decodificação, possuindo a garantia também de não possuir ambiguidade tendo em vista que nenhuma palavra-código é prefixo de outra palavra-código.

Escopo:

- Implementar um decodificador de Huffman em Verilog/SystemVerilog que realize o processo de decodificação segundo a tabela de símbolos apresentada abaixo;
- O decodificador deve receber os bits serialmente e retornar o número do símbolo (1 até 18) identificado;
- Faça um testbench que demonstre o funcionamento do decodificador em questão;
- Apresente as formas de onda obtidas através da simulação;
- Trate corretamente os erros que podem ocorrer.

Símbolo	Palavra-Código
S1	00
S2	01
S3	10
S4	110
S5	111000
S6	111001
S7	111010
S8	1110110
S9	1110111
S10	1111000
S11	1111001
S12	1111010
S13	1111011
S14	1111100
S15	1111101
S16	1111110
S17	11111110
S18	11111111

[1] https://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Huffman

Operador de Sobel

O operador de Sobel é um operação largamente utilizada em algoritmos para detecção de contornos em imagens[1]. Este filtro calcula o gradiente da intensidade da imagem em cada ponto, provendo informações sobre a variação de claro para escuro na imagem. Com isso, é possível identificar a velocidade que luminosidade se altera em cada direção da imagem, se de forma mais suave ou mais abrupta, permitindo identificar contornos existentes.

Escopo:

- Implemente em Verilog/SystemVerilog o operador de Sobel na forma sugerida em [2]. Para simplificar, substitua a equação (2) pela soma dos valores absolutos |G|=|Gx|+|Gy|.
- O código Systemverilog apresentado no link pode ser usado como ponto de partida, mas a avaliação irá considerar somente as suas contribuições.
- Implemente um testbench que permita avaliar o funcionamento do operador em questão;

[1] https://pt.wikipedia.org/wiki/Filtro_Sobel
[2] https://sistenix.com/sobel.html

Outras atividades

Você poderá realizar qualquer atividade que demonstre conhecimento na área de microeletrônica. Outras atividades possíveis seguem abaixo:

- Simular o PicoRV32 (https://github.com/YosysHQ/picorv32);
- Implementar um filtro de Sobel em Verilog/SystemVerilog para detecção de bordas em imagens;
- Simular IPs disponíveis no site http://opencores.org/projects;
- Utilizar IPs do site http://opencores.org/projects com a Arquitetura RISC-V (neste caso, será necessário trabalhar com barramento).
- Projetar seu próprio processador RISC-V em SystemVerilog;
- Implementar em software (em linguagem de sua escolha) uma FIFO de tamanho limitado e testá-la.
- Implementar e testar uma FIFO em hardware.