

**Universidade Federal de Santa Catarina
Centro Tecnológico - CTC
Departamento de Engenharia Elétrica - EEL**

**Gustavo Emanuel Kundlatsch (17102810)
Pedro José Vieira de Souza (17100535)**

TURMA 01208B

**gustavo.kundlatsch@gmail.com
pedro.j.v.souza@gmail.com**

**Relatório de Projeto Final EEL5105
2017.1**

Florianópolis, 5 de Julho de 2017.

Conteúdo

1. Introdução.....	3
1.1 Comparadores e somadores.....	4
1.2 Contadores.....	5
1.3 Seletores e decodificadores.....	7
1.4 Registradores.....	9
1.5 Controlador.....	11
2. Resultados e conclusões.....	12
Anexo A – Observações.....	13

1. Introdução

O projeto final de Circuitos e Técnicas Digitais (EEL5105), do semestre 2017.1, é a implementação de um Test Drive, que consiste em um jogo interativo, cujo objetivo é conduzir um veículo, utilizando switches para mover o carro para cima e para baixo e botões de pressão para alterar a velocidade, por uma pista de obstáculos, que é representada por uma matriz 16x32, completando duas voltas completas (64 colunas).

O usuário inicia a execução no estado Init, e inicia o jogo pressionando enter (KEY1), passando então para o estado Setup, onde deve escolher uma das quatro fases, com os switches 8 e 7. Ao pressionar enter novamente, passamos para o estado Game.

O jogo começa na coluna 0 e linha 7, com velocidade 0. Os controles são SW0 (para cima), SW1 (para baixo), KEY3 (frear) e KEY4 (acelerar), sendo que a velocidade pode variar entre 0 (Stop) e 5 (Max). O hexadecimal da placa (HEX5...HEX0) será utilizado para mostrar informações para o jogador, sendo que há dois modos de display: quando SW9 = 1, será informado a velocidade e a posição, e quando SW9 = 0, será informado o estado atual da máquina de controle, o número de bônus (vidas) e um contador decrescente que é o tempo.

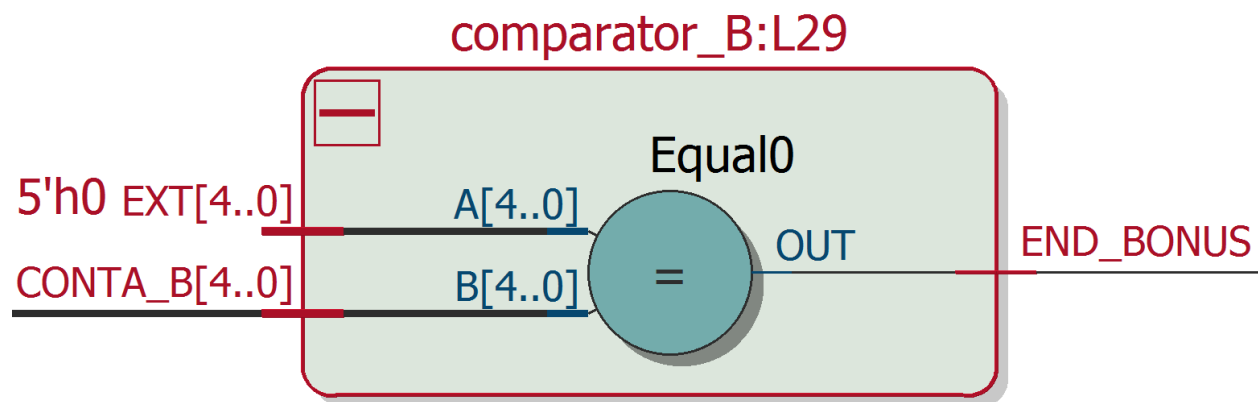
A cada batida, o número de bônus é reduzido em 1. Se o tempo ou o bônus chegar a zero, ou se o jogador completar as duas voltas, o jogo termina (passa para o estado End), e exibe a pontuação. Logo após o estado End, volta o estado Init. A qualquer momento o usuário pode reiniciar o jogo pressionando KEY0.

Para realizarmos essa implementação, utilizaremos 7 componentes: os Mapas, que consistem na declaração das matrizes de bits que formam as pistas com obstáculos; os Contadores, responsáveis pelas contagens decrescentes de tempo e bônus, e pela contagem ascendente de posição; Comparadores e somador, que determinam se o jogo acabou gerando bits correspondentes, e determina a quantidade final de pontos pela fórmula $\text{Posição Horizontal Final} + (2 \times \text{Bônus})$; Registradores, que criam os arrays de controle de velocidade e posição (horizontal e vertical); o Controlador, que é a máquina de controle de todo o jogo; Seletores, que são multiplexadores que vão fornecer os sinais usados por outros blocos e as saídas para os LEDs e Displays; e o Debouncer, que evita que pressionar um KEY dure muitos ciclos de clock.




1.1 Comparadores e somadores

Para descrever os comparadores e somadores, optamos pela abordagem comportamental por duas razões: a proximidade a outras linguagens de programação que aprendemos em ciência da computação, e a praticidade para visualizar o comportamento do circuito.

A imagem abaixo mostra o diagrama de blocos do comparador do bônus:



Podemos ver esse comparador funcionando no teste abaixo, onde a primeira onde é o número de bônus sobrando, a segunda é um comparador e terceira é o resultado dessa comparação.

 /comparator_b/CONTA_B	00000	00010						00000		
 /comparator_b/EXT	00000	00010			00011			00000		
 /comparator_b/END_BONUS	0									

1.2 Contadores

A FSM_clock tem um papel fundamental em nosso trabalho. Ela é uma máquina de estados finitos programada para funcionar como um seletor de clocks. No jogo, esse seletor será usado para implementar a aceleração e desaceleração do jogo. Quando o jogador desejar acelerar, será selecionado um clock “mais rápido”. Quando o jogador resolver desacelerar, será escolhido um clock “mais lento”.

Para termos esse efeito, usaremos o clock da placa para criar clocks “falsos”, segundo o código abaixo:

```
if falling_edge(CLOCK_50) then --Esse if é executado quando o clock está passando do estado 1 para o estado 0
  counterA <= counterA + 1; --Aqui, o contador é acrescido de 1
  if counterA = x"2FAF07F" then --Essa condição define a frequência do clock feito com o contador
    CLKA <= '1'; --Assim, o clockA recebe um sinal 1 (ativado)
    counterA <= x"0000000"; --e o contador é zerado
  else
    CLKA <= '0'; --caso contrário, o clockA recebe sinal 0 (não é ativado)
  end if;
end if;
```

O mesmo código é usado para gerar os outros clocks, alterando apenas a primeira condicional (counterA precisa ser = x"2FAF07F", counterB = x"17D783F", e assim por diante, gerando diferentes frequências. Além da FSM clock, o projeto é constituído por outras três FSM: control, speed e position.

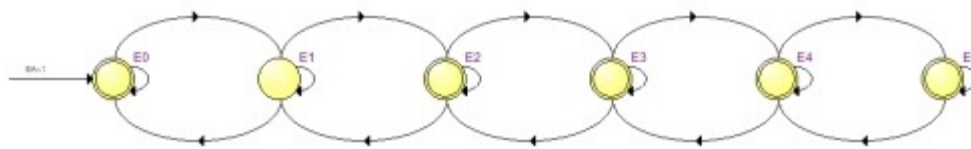


Figura 1: Diagrama de estados da FSM control

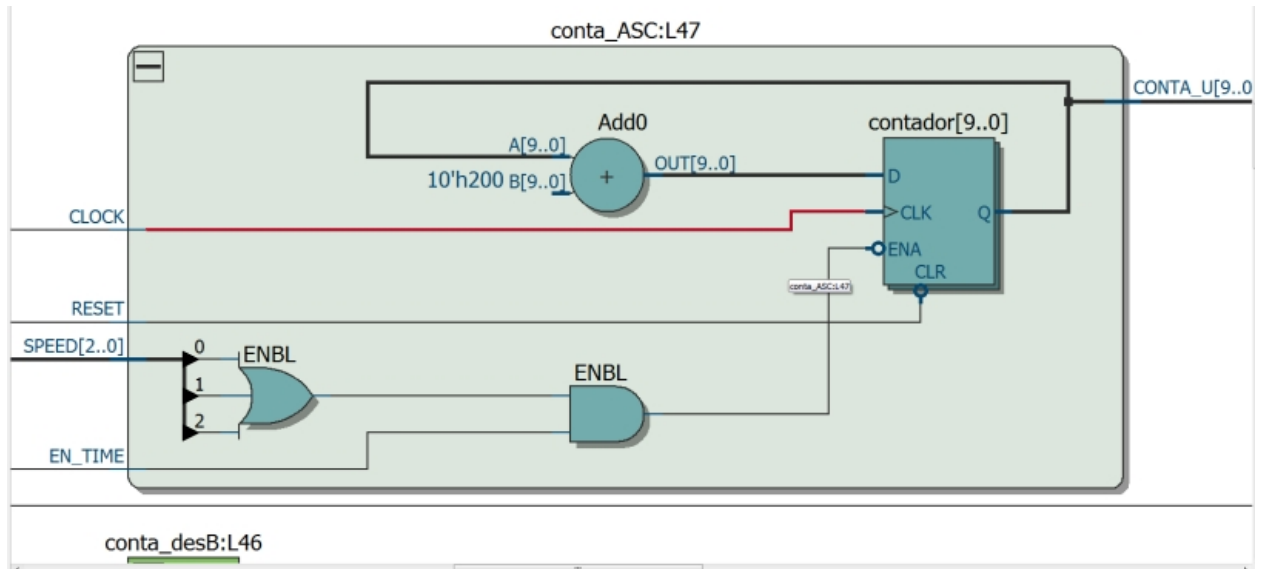
Já os contadores são utilizados para fazer a contagem decrescente de bônus e tempo, e ascendente de posição, e são utilizados em outras partes do código para ter o controle da situação atual do jogador (FSM control, que verifica se o usuário pode continuar o jogo ou se já perdeu) e da posição atual do jogador dentro da matriz que é a pista de obstáculos (FSM position).

Elas foram desenvolvidas assim como fizemos em sala de aula, como demonstra o arquivo “conta_desA.vhdl”, que faz a contagem decrescente que é usada no “comparador_time.vhdl”, que compara o tempo com zero (fim do tempo, conseqüentemente, fim do jogo).

No teste abaixo, verificamos a integridade de nosso código. A variável CONTA_U é a saída da conta ascendente, e o contador é a bloco lógico dentro dele, que será explicado logo abaixo.

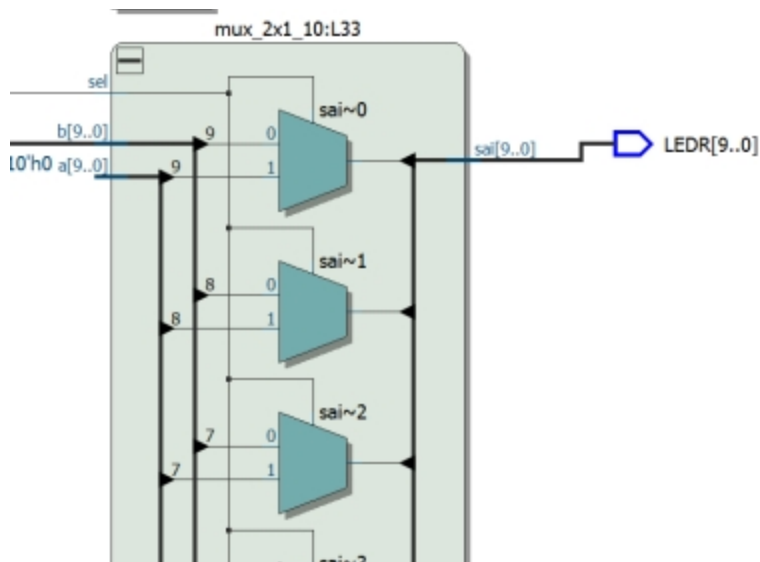
/conta_asc/CONTA_U	0000010110	0000010011	0000010100	0000010101	0000010110
/conta_asc/ENBL	0				
/conta_asc/contador	0000010111	0000010011	0000010100	0000010101	0000010110

Esse contador é demonstrado na imagem a seguir de acordo com as portas lógicas que ele usa para fazer a contagem, assim como vimos durante o curso, nas aulas teóricas:



1.3 Seletores e decodificadores

Utilizamos a abordagem comportamental por questão de praticidade, pois desenvolvemos um mux em aula, da mesma maneira. Abaixo, temos a imagem de um mux 2x1 de dez bits:



A estrutura se repete até o bit 9, englobando um total de dez bits. O código de todos os mux é escrito de maneira similar, assim como mostrado a seguir, no caso para o mux 2x1 de dez bits:

```
architecture bhv of mux_2x1_10 is
begin
    sai <= a when -- saída recebe a quando
        sel = '1' -- o selecionador for igual a um
    else
        b; -- caso contrário, recebe b
end;
```

10/a	0000111111	0000111111					
10/b	1100000000	1100000000					
10/sai	1100000000	1100000000			0000111111		
10/sel	0						

Essa comparação é feita bit a bit, de acordo com o tamanho do dado recebido. A variável sel é o selecionador que vem da FSM control, e servirá para realizar a comparação entre as duas variáveis a e b, que são arbitrárias. Nessa simulação, forçamos valores para o a e para o b. No select, selecionamos 0,

dessa maneira, a saída sai recebeu o valor de b. Quando selecionamos o select como 1, sai recebeu o valor de a.

Os demais mux apresentam o mesmo comportamento, tendo mais ou menos entradas, sendo maiores ou menores, de acordo com a necessidade.

1.4 Registradores

Os registradores de nosso projeto são constituídos por dois componentes essenciais: a FSM speed e a FSM position, que darão ao nosso circuito a essência do jogo, passando a sensação de se correr na pista de testes.

A FSM speed faz o controle da velocidade do carro do jogador. A velocidade do carro é guardado na variável speed, e de acordo com a ação de acelerar ou desacelerar importa pelo jogador, o estado muda. Caso deseje acelerar, a máquina verifica se é possível ir para um estado mais rápido. De modo similar isso acontece ao tentar ir mais devagar.

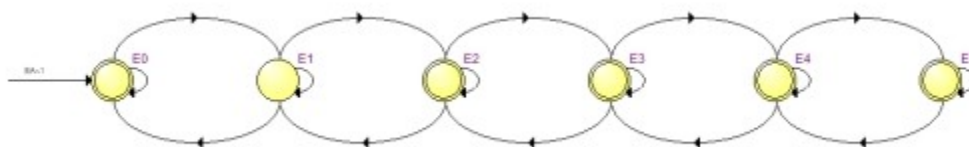
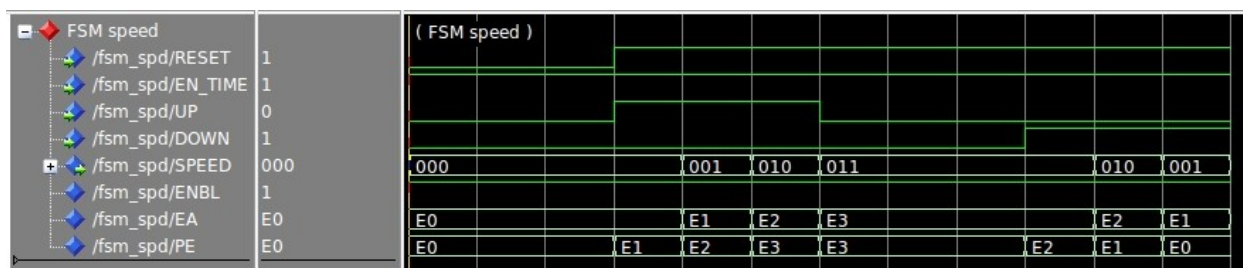


Figura 2: Diagrama de blocos da FSM speed

O teste de ondas abaixo mostra o comportamento da FSM. Ele começa com o reset ligado (pressionado), e após três ciclos de clock ele é solto (reset = 0) e UP é forçado como um. Mais três ciclos e UP volta para zero e DOWN é setado para um. Por fim, retornamos DOWN para zero. Observe como a saída se altera em cada caso:



Já a FSM position é a máquina que recebe como entrada dois switches, uma delas quando está ativa avança para o próximo estado, quando outra está ativa, retorna para o estado anterior e quando nenhuma das duas estiver ativa, o estado permanece o mesmo. Dessa maneira, geramos o controle que o usuário terá para controlar o carro para cima e para baixo. Esse desenvolvimento faz com que seja impossível o jogador “loopar” (ir da última linha da matriz para a primeira, o que não faz sentido no contexto do programa).

1.5 Controlador

A FSM control faz toda a parte de controle dos quatro estados que nosso projeto possui: Init, Setup, Game e And (já descritos na introdução). Por meio das comparações descritas nas últimas cinco linhas de cada declaração de estado, definimos qual será o próximo. Por exemplo, quando o estado é game, é necessário o jogo acabar (duas voltas completas na pista), os bônus acabarem ou o tempo se esgotar.

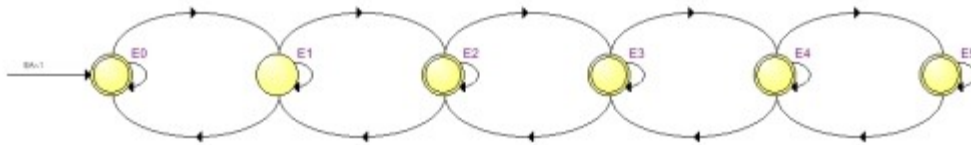
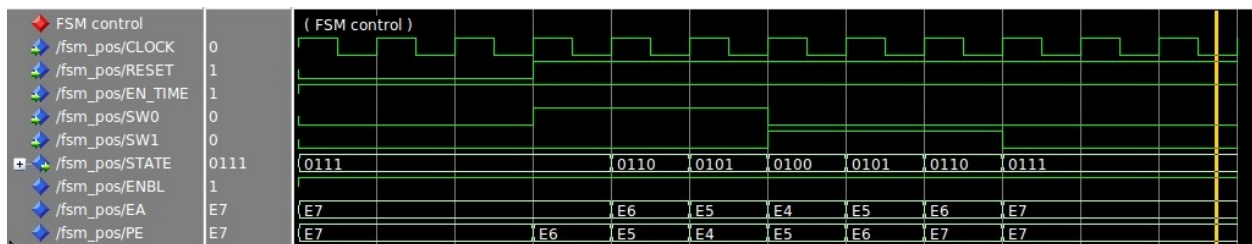


Figura 4: Diagrama de estados da FSM control

O diagrama de ondas abaixo mostra o comportamento da FSM control quando ela foi submetida aos testes:



Note que de início o reset está ativo. A cada três ciclos de clock alteramos entre diferentes testes: primeiro desativamos o reset e colocamos o SW0 como um. Depois, desativamos o SW0 e forçamos o SW1 como um. Por último, testamos o caso de ambos desativados. Vale ressaltar que caso ambos estejam em um, o comportamento será o mesmo de quando ambos estão em zero, por isso é trivial mostrar esse caso de teste.

2. Resultados e conclusões

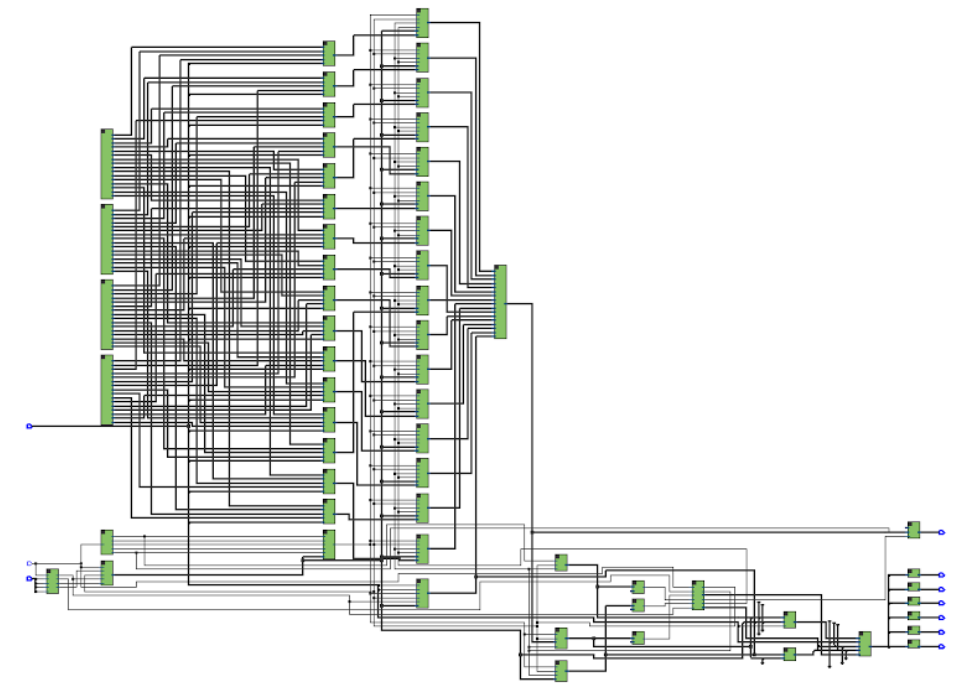
Por fim, todos os arquivos são ligados em nosso topo, que está dividido em sessões, e comentado para a melhor visualização. Das partes importantes de citar sobre o processo, escolhemos as seguintes:

Primeiro colocamos o sincronizador, que foi fornecido pelo professor. Esse é um bloco isolado, pois serve apenas para evitar que a grande quantidade de ciclos de clock pela qual o usuário mantém um botão pressionado possa atrapalhar a execução de nosso programa.

Depois, enaltecemos os mapas. Eles são divididos em dezesseis sinais de trinta e dois bits, que são selecionados por um mux 4x1 também de trinta e dois bits. São esses mapas que serão utilizados pelo FSM position com o importante papel de serem a pista por onde o corredor precisa passar por duas voltas.

Por último, é essencial descrevermos as ligações feitas com o controle. Ele recebe a pontuação objetivo e as informações dos casos de: ter acabado as vidas, ter acabado o tempo, se o reset foi pressionado. Com isso, ele processa o estado atual do jogo (que é mostrado no display). Além disso, é responsabilidade desse com ponteiro as saídas dos seletores dos leds e do display, que são a interface com o usuário.

Tendo terminado os nossos componentes, testamos o programa na placa DE1, e obtivemos um resultado quase completamente satisfatório, tendo como falha apenas o retorno ao jogo após ter dado um reset. Quando você reseta, a velocidade e a posição são mantidos.



Anexo A – Observações

Como dupla gostamos do projeto, achamos que ele traz uma proposta interessante, pois usa tudo que foi aprendido em sala de aula de forma concisa. A nossa experiência com uso de VHDL foi difícil, devido à falta de conhecimento e familiaridade com linguagens de descrição de hardware, o que causou problemas no desenvolvimento no trabalho. O sinal de reset tem um erro o qual não conseguimos resolver e, a princípio, haviam erros no display de sete segmentos, porém os mesmos foram resolvidos.