Universidade Estadual de Campinas
MC613 - Turma A
Grupo F03

Projeto Final - Boliche

Integrantes:

Victor Ferrão Santolim
RA 187888
victorfsantolim@gmail.com

Victor Ferreira Ferrari
RA 187890
vferrari@mpc.com.br

Entrega: 18/06/2018

1) Introdução

1.1) Proposta

A ideia por trás do projeto foi escolhida a partir de uma lista de sugestões de projetos finais [1].

Trata-se de um contador automático de placar de boliche. Nele, é possível ter de um a seis jogadores, com rodadas alternadas, em dez turnos. Cada jogada tem como entrada os pinos derrubados na mesma por meio das dez *switches* e um sinal de fim de jogada por meio de um *push button* [2].

A pontuação do jogador é então calculada de acordo com as regras do boliche, descritas nas próximas seções, e o placar (interno) é atualizado. Ao detectar um *strike*, os segmentos do *display* [2] circulam por inteiro, e ao detectar um *spare*, parcialmente, por aproximadamente dois segundos.

Durante o jogo, o contador indica de qual jogador é a vez, sua pontuação, qual das jogadas ele está fazendo, e qual turno o jogo está. Ao final, o placar é mostrado em ordem decrescente de pontuação, do vencedor ao último colocado.

1.2) Regras do Boliche

Um jogo de boliche consiste de dez turnos, também chamados de *frames*. Em cada *frame*, todo jogador terá dois arremessos para derrubar a maior quantidade de pinos que puder com sua bola de boliche, com o máximo sendo dez por turno. Cada jogador tem sua rodada em uma ordem predeterminada.

Se um jogador conseguir derrubar todos os dez pinos em um único arremesso, essa jogada é chamada de *strike*. Se um jogador conseguir derrubar todos os dez pinos nos dois arremessos da rodada, essa jogada é chamada de *spare*. Pontos bônus são dados nas jogadas seguintes (na próxima para o *spare*, e nas duas próximas para o *strike*).

Se um jogador derrubar todos os dez pinos no décimo (e último) *frame*, ele tem direito a um total de três arremessos na rodada.

Sobre a pontuação, em geral, um ponto é dado para cada pino derrubado. Ou seja, se um jogador derrubar dois pinos em um arremesso e seis pinos no seguinte, ganhará oito pontos pela rodada. Se um jogador derrubar oito pinos no primeiro arremesso e nenhum no segundo, também ganhará oito pontos. No caso em que o jogador consiga derrubar todos os dez pinos dentro de um *frame*, há uma pontuação bônus.

No caso do *strike*, quando todos os dez pinos são derrubados com a primeira bola, o jogador recebe dez pontos mais um bônus de qualquer pontuação feita nas próximas duas bolas. Nesse caso, os pontos dados para os dois arremessos após um strike são contados duas vezes. Assim, no caso de um *strike* seguido de uma jogada que derruba quatro pinos e uma jogada que derruba três pinos, o total de pontos é 10+4+3=17 do *strike*, mais 4+3=7 do *frame* seguinte, totalizando 24 pontos.

No caso de dois *strikes* seguidos, na prática a jogada imediatamente após o último *strike* tem seus pontos contados duas vezes. Em um *frame*, o número máximo possível de pontos é 30, que é a quantidade de pontos no primeiro *frame* de três *strikes* seguidos (10+10+10). Um jogador que faz um *strike* no último turno recebe o direito de dois arremessos extra para contagem de pontos bônus. Esses pontos não são contados isoladamente, mas apenas como bônus do *strike*.

No caso do *spare*, quando os dez pinos são derrubados com os dois arremessos da rodada, o jogador recebe 10 pontos mais um bônus de qualquer pontuação feita no próximo

arremesso (apenas o primeiro arremesso é contado). Assim, no caso de um *spare* seguido de uma jogada que derruba quatro pinos e uma jogada que derruba três pontos, o total de pontos é 10+4=14 do *spare*, mais 4+3=7 do *frame* seguinte, totalizando 21 pontos.

Um jogador que faz um *spare* no último turno recebe o direito de um arremesso extra para contagem de pontos bônus. Esses pontos não são contados isoladamente, mas apenas como bônus do *spare*.

É possível haver, no máximo, 12 *strikes* em um jogo, para um total de 300 pontos, chamado de “jogo perfeito”. Para essas regras, foram usadas as referências [3] e [4].

1.3) Glossário de Termos

Arremesso: Movimento de jogar a bola com o objetivo de derrubar o máximo de pinos presentes possíveis.

Jogada: É equivalente a um arremesso. Cada jogador pode contar com uma a três jogadas por rodada.

Turno: Os conjuntos de rodadas de cada jogador em sequência representam um turno. Uma partida de boliche apresenta dez turnos.

Rodada: Conjunto de uma jogadas/arremessos para um mesmo jogador durante um turno.

Spare: Nome dado quando o jogador derruba, em dois arremessos da mesma jogada, os dez pinos. No primeiro, pode derrubar entre zero e nove pinos, e no segundo derruba os restantes.

Strike: É o nome dado quando o jogador derruba, em um só arremesso, dez pinos.

2) Aspectos Teóricos

Os único componente necessário para a implementação deste projeto foi uma placa FPGA (a usada foi a DE1-SoC). Isso significa que todos os aspectos de interface fazem parte das características padrão da placa, ou seja, *switches* e *push buttons*, assim como os *displays* de sete segmentos e os LEDs unitários [2].

O projeto se trata de um circuito sequencial, então muito do que foi visto para circuitos sequenciais foi utilizado. Todo o circuito é sincronizado com um *clock*, que é no caso o CLOCK_50 da placa FPGA, um *clock* de 50 MHz [2].

Como o circuito precisou de muitas partes sequenciais, utilizamos muitas estruturas do tipo *process*, estrutura da linguagem VHDL dentro da qual todas as operações são realizadas sequencialmente. Por isso, muito do código é simplesmente uma sequência de *processes* [5].

Algo muito importante também na criação de circuitos sequenciais é o conceito de máquina de estados. Utilizamos muitas máquinas de estado no projeto, seja para regimento dos turnos, vez de cada jogador, cada jogada, ou etapa do jogo. Tivemos máquinas de estado de Moore (como *Turnos*, *Jogadores*, *Controle* e os segmentos do visor em *Gira_Visores*) e de Mealy (como a máquina de estados das jogadas em *Rodada*)[6].

Certas partes do projeto tiveram que ser temporizadas, ou seja, algo deve ocorrer depois de uma certa quantidade de tempo em segundos. Para atingirmos esse fim, utilizamos divisores de *clock* e o conhecimento da frequência do *clock* utilizado. Assim, a cada número de bordas de *clock* (e consequentemente, período de tempo) um pulso é gerado, e esse pulso é usado também como *clock*, para sincronizar certas partes.

Um divisor de *clock* ou, mais genericamente, um divisor de frequências é um circuito que recebe um sinal periódico, e aumenta esse período, assim diminuindo a sua frequência. Para isso, utiliza um contador, que é incrementado a cada borda (de subida) de *clock*. Quando esse contador chegar ao valor desejado, gera um pulso que dura até a próxima borda (de subida) de *clock* [7]. Esse sinal pode então ser utilizado para sincronizar outras partes do código.

Para mostrar nos *displays* um número com mais de um dígito, o que foi preciso para as pontuações dos jogadores, foi preciso converter cada dígito para um código de sete segmentos diferente. Para isso, utilizamos o código BCD. Esse código representa números em decimal a partir da representação binária de cada dígito. Ou seja, um número com três dígitos decimais é convertido para um código BCD de 12 bits, ou seja, quatro para cada dígito (com valor de 0 a 9) [8] .

No final do jogo, as pontuações de cada jogador são mostradas nos *displays* em ordem decrescente, e para isso algum modo de seleção ou ordenação deve ser utilizado. Então, decidimos implementar algo que segue a lógica do algoritmo de ordenação "*Bubble-Sort*. O *Bubble-Sort* é um algoritmo muito simples baseado em comparações, onde um *array* é percorrido, comparando elementos adjacentes dois-a-dois e trocando as posições dos que estiverem fora de ordem. Isso é repetido para os *n-1* primeiros elementos, e depois com os primeiros *n-2* itens [9].

3) Descrição do Sistema

3.1) Descrição da Interface (Manual de Utilização)

O contador de pontos do boliche implementado requer o uso apenas de um computador e da placa DE1-SoC para funcionamento. Após passar o programa para a placa, o usuário será levado à etapa de inicialização do jogo, onde deve ser selecionado o número de jogadores que participam da partida, conforme a Figura 1 abaixo:

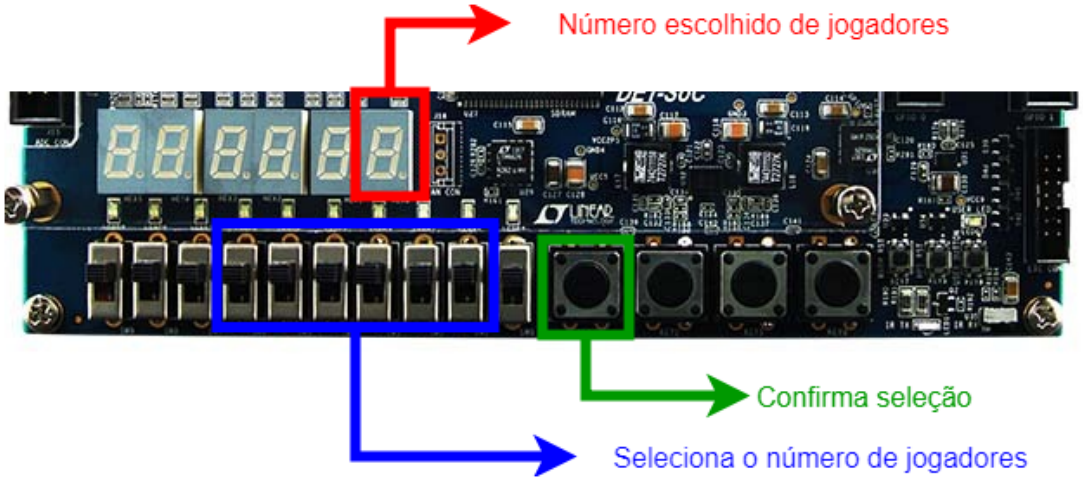

Figura 1 - Interface de inicialização

Deve ser acionado (posição para cima) apenas um *switch* entre 1 e 6, que representa o número de jogadores desejados, os demais devem estar em nível lógico baixo (posição para baixo). Se nenhum *switch* estiver acionado, o display ficará totalmente apagado, e caso haja uma seleção inválida, exibirá a letra “E” , que significa “Errado”. Caso uma seleção válida tenha sido realizada, o display HEX0 exibirá esse número, apenas para verificação. Nessa situação, pressionar o botão KEY3 confirmará o número de jogadores e iniciará o jogo, que possui interface conforme a Figura 2 seguinte:

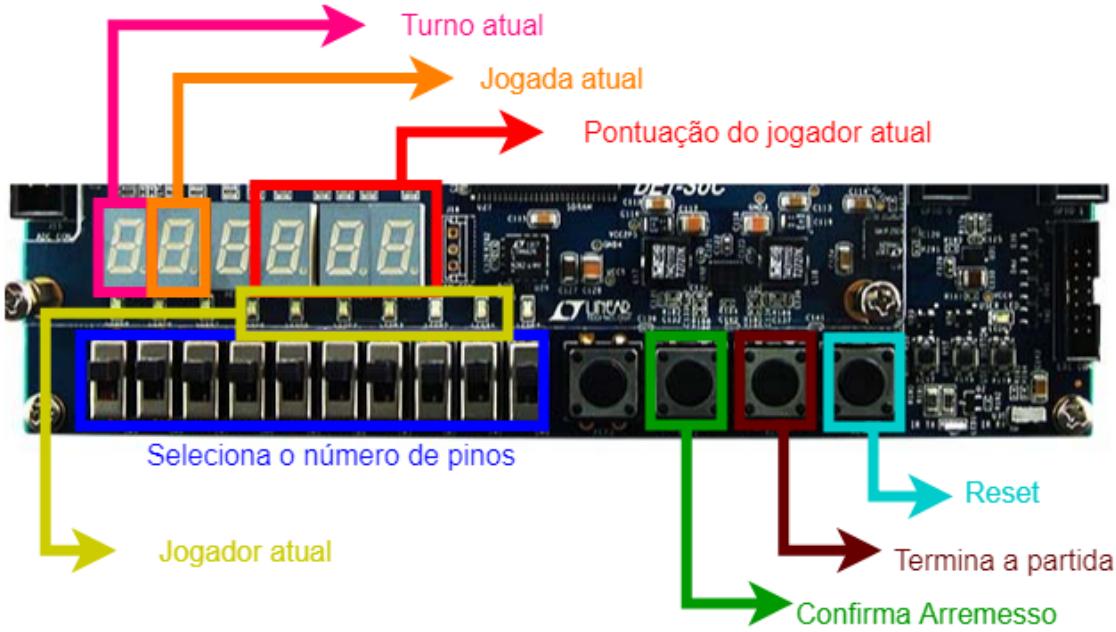

Figura 2 - Interface no decorrer do jogo

Nessa interface, o turno e jogada atuais são exibidos nos visores HEX5 e HEX4, respetivamente. O jogador atual é indicado por qual LED entre 1 e 6 está aceso. A pontuação que o jogador atual possui até antes do início desse turno é exibida nos visores HEX2 a HEX0, em decimal. A informação sobre os pinos derrubados (entre 0 e 10) deve ser inserida no jogo por meio dos *switches* 0 a 9, onde o switch ativado significa que o pino foi derrubado, e desativado significa que não foi derrubado. Para confirmar o arremesso , basta pressionar o botão KEY2 e o jogo automaticamente prosseguirá para o próximo arremesso esperado. Caso seja o segundo arremesso dentro da mesma rodada de um jogador, espera-se que o usuário mantenha os pinos já derrubados como ativados. Ao pressionar o botão de reset (KEY0), todas as pontuações são apagadas e o jogo retorna à interface de seleção de quantidade de jogadores. Em qualquer momento do jogo, pressionar o botão KEY1 encerrará a partida da forma como ela está e partirá para mostrar os resultados como na Figura 3 seguinte (o mesmo acontece no final da partida regular, ou seja, no final de todos turnos):

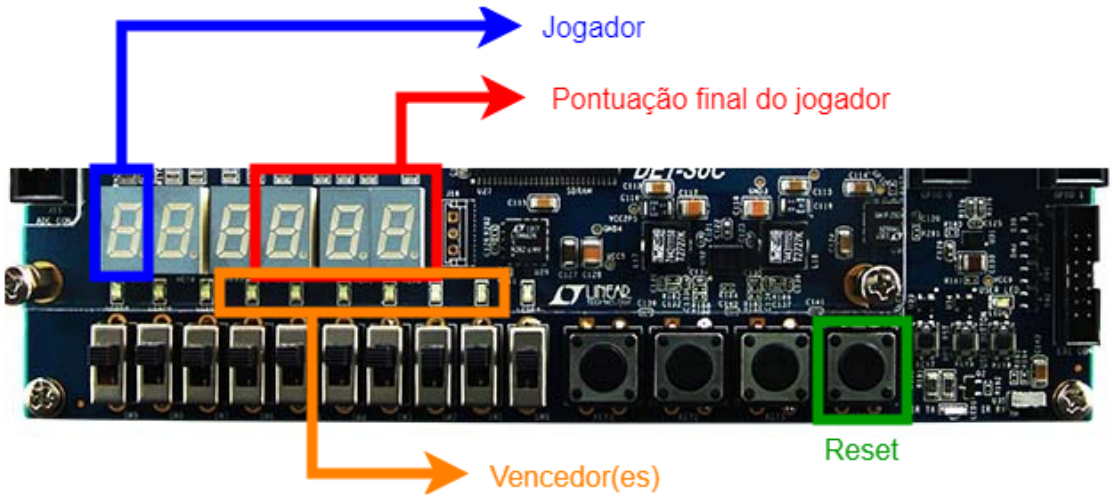

Figura 3 - Interface dos resultados

Após o término da partida, o jogo mudará para a interface de resultados, que indica com LEDS acesos o(s) vencedor(es). Nos displays, será exibido, em ordem de pontuação, os pares (número do jogador ; pontuação final desse jogador) em HEX5 e HEX2-HEX0, respectivamente. Em aproximadamente dois segundos, muda o jogador apresentado na tela e mostra o próximo em pontuação. Após a exibição do último colocado, esse processo se repete, começando novamente do primeiro colocado, até que seja apertado o botão HEX0 de reset, que retorna à interface de escolha de jogadores, deletando todos os resultados armazenados da partida.

3.2) Funcionamento Geral

A ideia geral da implementação desse trabalho é transformar as regras de uma partida de boliche em lógica pura, independente de interpretações, de modo que possa ser descrita por elementos lógicos padrões como portas lógicas, *flip-flops*, registradores, máquinas de estados e outros. Desse modo, se faz necessária a representação de todas as informações que envolvem o boliche, como pontuação, número de jogadores, turnos, *strikes*, *spares* e muitas outras em sinais lógicos binários, que somos capazes de tratar. Todas essas condições são necessárias para que o circuito final gerado seja sintetizável diretamente em hardware.

O contador de pontos do boliche desenvolvido é um grande circuito lógico que possui como entrada os botões e *switches* da placa de desenvolvimento e tem como saída dos LEDs e visores de sete segmentos. Esse circuito foi subdividido em várias partes que realizam tarefas independentes e paralelas, como ordenar um vetor lógico, transformar um vetor lógico contendo número binário em sinais para visor de sete segmentos, contabilizar a pontuação em uma rodada, detectar o fim de turno, etc. Essas subdivisões lógicas implementadas serão melhor apresentadas no tópico seguinte, e descritas em detalhes no tópico 3.5.

3.3) Divisão em Módulos

A modularização do circuito foi uma prioridade durante o desenvolvimento do projeto, inclusive por facilitar a organização e a visualização do código.

Por definição, o projeto possui três partes: Início, Andamento, e Fim. Na prática, cada uma dessas partes (exceto o início) foram divididas ainda mais, em entidades menores.

Para facilitar a descrição, neste relatório podemos classificar os módulos em três classes: ***principais***, ***importantes*** e ***auxiliares***.

Os módulos ***principais*** contêm a maior parte da lógica do projeto, normalmente são as mais extensas, sendo assim o núcleo do projeto. São, então, a entidade *top-level Boliche,* e as outras do núcleo: *Inicializador, Andamento, Calcula_Pontos, Rodada, e Final*.

Os módulos ***importantes*** são ainda específicos para o projeto em questão, e fundamentais para sua corretude, porém têm papéis mais específicos, como ordenação ou máquinas de estados. São, então, as máquinas de estados *Controle, Turnos, e Jogadores*, o módulo de ordenação *Ordena*, e *Gira Visores* (introduzido na seção 3.5).

Por último, os módulos ***auxiliares*** resolvem problemas isolados, comuns, mas que também aparecem no projeto. São, então, o divisor de *clock Clk_Div*, o conversor de um número em binário para código BCD *Conversor_BCD*, o contador de “1s” *Cont_1*, e o conversor de um número binário para código para *display* de sete segmentos *Bin2Dec*.

No total, são 15 módulos diferentes, entre as três categorias. Essa divisão é feita apenas para o relatório, pois na prática todos os componentes estão dentro do mesmo pacote (*boliche_pack*), e os arquivos estão todos na mesma pasta. Todos esses módulos estão descritos com mais detalhes na seção 3.5.

3.4) Diagramas de Blocos

A seguir, nas figuras 4, 5, e 6, estão os diagramas de blocos de algumas das entidades principais do circuito, que estão diretamente envolvidas no andamento de uma partida de boliche e no cálculo dos pontos. Nos diagramas, existem algumas setas com indicadores de “condicionais”. Estas são explicadas dentro da descrição do módulo correspondente na seção 3.5.

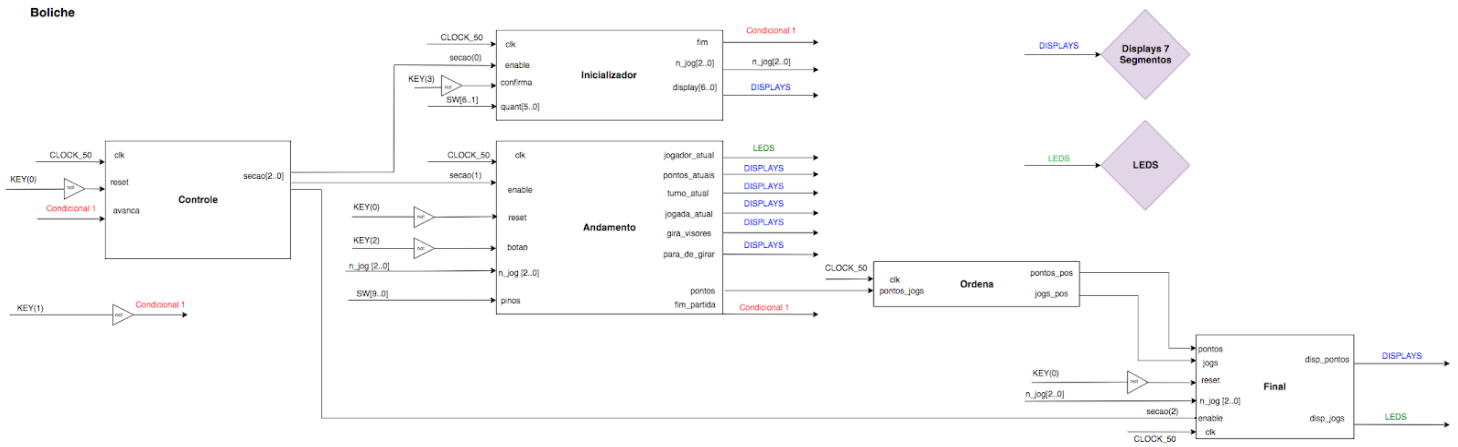

Figura 4 - Diagrama da Entidade Top-Level Boliche

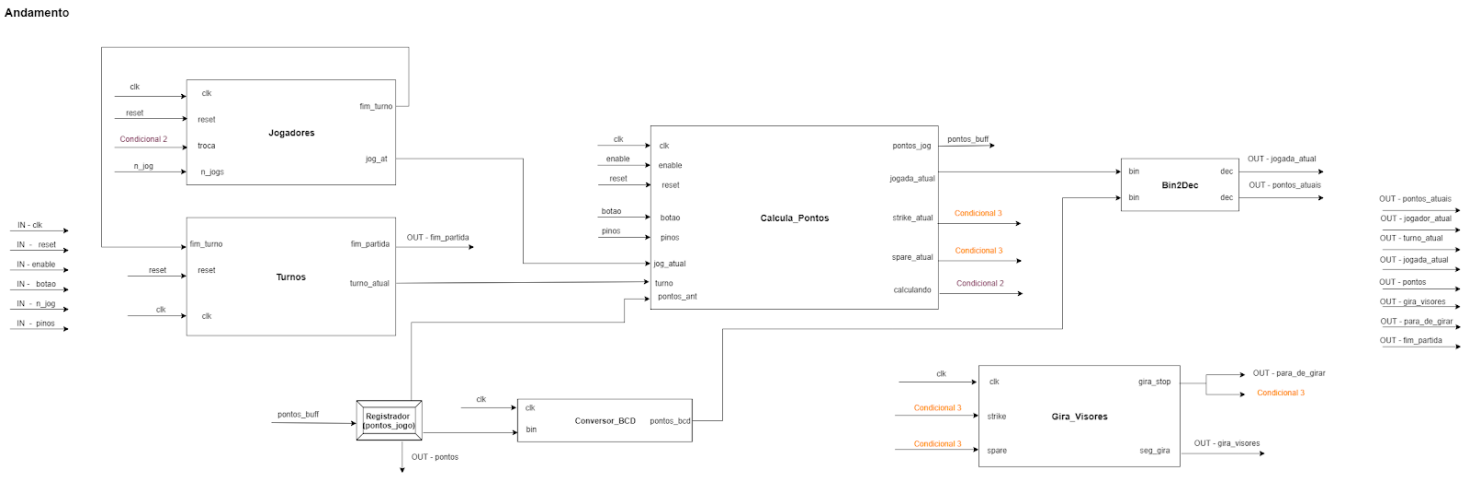

Figura 5 - Diagrama da Entidade Principal Andamento

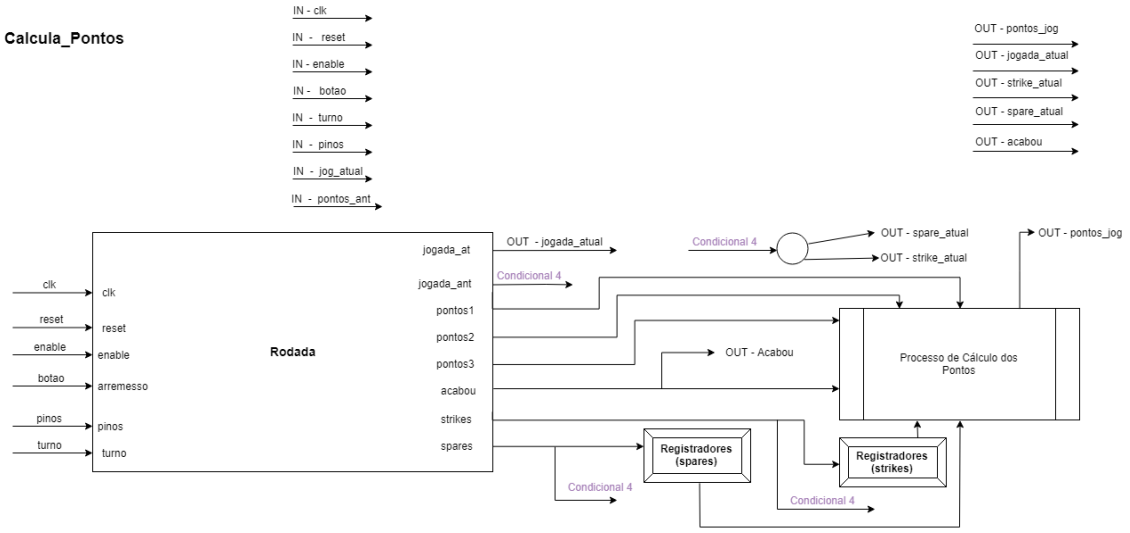

Figura 6 - Diagrama da Entidade Principal Calcula_Pontos

3.5) Descrição dos Módulos

3.5.1) Entidades Principais

Boliche: É a entidade *top-level*, ou seja, recebe como entrada alguns sinais lógicos da placa FPGA, que são CLOCK_50 (clock de 50MHz), SW (as dip-switches da placa) e KEY (os push buttons da placa). CLOCK_50 é o clock que sincroniza a maior parte do circuito (exceto algumas seções).

Sua lógica pode ser vista como a visão mais geral do projeto: uma máquina de estados (*Controle*) administra os três diferentes estados do sistema. Primeiro, o jogo é iniciado, com a leitura do número de jogadores (*Inicializador)*, depois há o jogo de boliche, onde toda a lógica da contagem de pontos é vista (*Andamento*). No final do jogo, os jogadores são ordenados decrescentemente por seus pontos (*Ordena)*, e as pontuações são mostradas aos jogadores nessa ordem (*Final*), e o(s) vencedor(es).

O sinal para trocar o estado na entidade *Controle* depende da **CONDICIONAL 1** descrita no diagrama, que leva em consideração o estado atual, o pressionar do botão KEY(1) e os sinais de fim de *Inicializador* e *Andamento*, que também são os responsáveis por avançar os estado. O objetivo é controlar qual das entidades relativas às etapas do jogo deve estar em funcionamento, assim como determinar qual das informações recebidas devem ser apresentadas nos Displays e nos LEDs.

As saídas da entidade são também diretamente à placa: os seis *displays* de 7 segmentos (HEX0-HEX5), e alguns LEDs (LEDR). Por isso, as informações para cada saída variam de acordo com a etapa do jogo. Nos *displays*: o número selecionado de jogadores (antes de confirmar), no início; a situação atual do jogo (turno, jogada, e pontos do jogador atual), no meio; a sequência decrescente de pontos e seus respectivos jogadores, no final; e quando há um *strike* ou *spare*, uma sequência que circula os segmentos por um determinado tempo. Nos LEDs: o jogador atual, durante o jogo, e o(s) jogador(es)

vencedor(es), no final. A maior parte dessas saídas é fornecida pelas outras entidades que comunicam diretamente com esta entidade.

Inicializador: É responsável por iniciar o jogo, ou seja, receber pelas dip-switches o número de jogadores (o número da switch é a quantidade escolhida de jogadores), e registrar esse número ao receber o sinal de um push button (KEY(3)). O número escolhido é usado em múltiplas ocasiões no circuito.

Andamento: Essa entidade é a parte mais importante do circuito e recebe como entrada, além dos sinais de *clock*, *enable* e *reset,* um sinal de acionamento de botão (KEY(2), da entidade *Boliche*), o número de jogadores na partida (de *Inicializador*) e os valores lógicos dos switches da placa (representando os pinos, e vindos de *Boliche*).

A funcionalidade da entidade é principalmente abrigar as máquinas de estados *Turnos* e *Jogadores*, avançando seus respectivos estados a medida que os rodadas são realizadas.

A troca de jogadores é baseada na **CONDICIONAL 2** vista no diagrama, onde, após o sinal de fim ser dado por *Rodada*, há uma espera de uma borda de *clock* após o início do cálculo para salvar os pontos (dados por *Calcula_Pontos)* no registrador correspondente, e então o jogador pode ser trocado. Para cada jogador em cada turno, a pontuação do jogador é computada com base no seu desempenho naquela rodada e eventuais bônus de pontuação decorrentes de *strikes/spares* em turnos anteriores (*Calcula_Pontos*).

As pontuações dos jogadores ficam salvas em registradores, e elas são constantemente passadas por um conversor para código BCD (*Conversor_BCD)*, e então cada dígito é transformado em sinais para *displays* de 7 segmentos (*Bin2Dec*). O turno e jogada atuais também são transformados em valores lógicos para os *displays*.

Também gera a saída que circula os segmentos, utilizada quando detecta-se um *strike* ou *spare* (*Gira_Visor)*. O sinal de *enable* de *Gira_Visor* é baseado na **CONDICIONAL 3** vista no diagrama, onde o botão é o que primeiro desencadeia a verificação de *strike* e *spare* no sinal enviado por *Calcula_Pontos*, e esse sinal continua o mesmo até o sinal pela entidade *Gira_Visor,* para parar de girar, ser dado.

Então, a entidade tem como saída os valores lógicos para exibição em *display* (pontos atuais, salvo na própria entidade turno atual, de *Turnos*, jogada atual, de *Rodada* e gira visores, de *Gira_Visor*), sinal para acender os LEDs (jogador atual, de *Jogadores*), sinais de dados (pontos, para serem usados por *Ordena*) e sinais de *flag* (para girar os visores e sinalizar fim da partida, usados por *Boliche*).

Calcula_Pontos: Recebe como entrada, além de os sinais de *clock*, *enable* e *reset*, o sinal de ativação do botão, os pinos (ambos de *Boliche*), jogador atual, turno e pontos anteriores (de *Andamento*).

Sua função é utilizar a lógica de *Rodada* para receber a quantidade de pinos derrubados nos arremessos realizados pelo jogador, e a partir dessas informações atualizar a sua pontuação, somando-a com a pontuação obtida nessa rodada e levando em consideração os bônus causados por *strikes/spares* nos turnos anteriores.

Caso um *spare* tenha ocorrido, os pontos da próxima jogada são duplicados, e caso um *strike* tenha ocorrido, as duas jogadas posteriores a ele têm seus pontos duplicados,

exceto na última rodada, onde é possível ter três jogadas, mas elas não se afetam dessa maneira.

Esse procedimento se baseia em várias condicionais que tratam os casos especiais de pontuação no boliche. Como saída retorna os sinais de pontos do jogador após a rodada (que são salvos em *Boliche*), a jogada atual (vinda de *Rodada*), sinais de eventos de *strike* e *spare* e flag de finalização da vez do jogador (vinda de *Rodada*).

Os sinais que indicam se acabou de ocorrer um *strike* ou *spare* são determinados com base na **CONDICIONAL 4**. Como esses resultados são salvos independentemente, e dependem da jogada anterior, essa condicional é necessária. Serão usados em *Andamento* para definir o sinal de funcionamento de *Gira_Visores*.

Rodada: Essa parte do circuito realiza a obtenção de cada um dos arremessos, transforma essa quantidade de pinos derrubados em pontos (contando o número de bits “1” no vetor lógico que representa os pinos), e define se houve *strike* ou *spare*, tomando as medidas pertinentes.

Recebe, então, além do *clock* e sinais de *enable* e *reset*, os pinos derrubados (valores lógicos das dip-switches), um sinal de botão pressionado (ambos de *Boliche,* passando por *Andamento* e *Calcula_Pontos*), e os números do jogador e turno atuais (de *Andamento*).

Então, conta a quantidade de pinos derrubados em cada jogada, e tem como saída a jogada atual, a jogada anterior (usadas em *Andamento*), os pontos de cada jogada (usadas em *Calcula_Pontos*), sinais lógicos que indicam se houve *strikes* ou *spares* na rodada (salvos em *Calcula_Pontos* e usados em *Andamento*), e um sinal de fim de rodada (usado em *Calcula_Pontos* e *Andamento*).

Final: É a entidade responsável por administrar a parte final do jogo, ou seja, o placar final. Recebe como entrada, além do *clock*, sinal de *enable* (da entidade *Controle)* e do sinal de *reset* (de um push button), o número de jogadores (de *Inicializador*), um vetor ordenado decrescentemente com os pontos de cada jogador, e um vetor com a ordem dos jogadores em relação aos pontos ordenados (ambos de *Ordena)*.

Então, utiliza a máquina de estados *Jogadores* para retornar como saída, a cada intervalo de tempo, um par jogador-pontuação (já em formato para o *display* de 7 segmentos), formando uma sequência que segue a ordem recebida, circularmente (ou seja, quando chega ao final, retorna ao começo). Essa saída é então colocada nos *displays* pela entidade *Boliche*, se pertinente.

3.5.2) Entidades Importantes

Controle: Máquina de estados simples: recebe, além do *clock*, uma entrada lógica que indica o momento de avançar para o próximo estado, e retorna um vetor lógico que indica a etapa do jogo (e também serve como sinal *enable*). Possui três estados: início, meio e fim, e é uma máquina limitada, ou seja, não retorna ao primeiro estado por outro motivo além de *reset*.

Recebe as entradas da entidade *Boliche*, e sua saída é usada como sinal de *enable* para muitas entidades, mas principalmente para as três principais (*Inicializador, Andamento e Final)*, e para definir as saídas da entidade *Boliche.*

Turnos: Máquina de estados simples: recebe, além do *clock* e *reset,* uma entrada lógica que indica o momento de avançar para o próximo turno, e retorna um vetor lógico que indica o turno atual do jogo e um sinal de fim de jogo quando a máquina chega ao final. Possui onze estados: “t1” a “t10” e “fim”, e é uma máquina limitada, ou seja, não retorna ao primeiro estado por outro motivo além de *reset*.

Jogadores: Máquina de estados que recebe, além do *clock* e *reset,* uma entrada lógica que indica o momento de avançar para o próximo jogador e o número de jogadores na partida.

Como saída, retorna um vetor lógico que indica o jogador atual e um sinal de fim de turno, usado como entrada de *Turnos*. Possui sete estados: “jog1” a “jog6” e “fim”, e é uma máquina circular, ou seja, retorna ao primeiro jogador após a máquina de estados chegar no estado fim.

Gira Visor: Recebe como sinais de entrada o *clock*, sinal de *strike* e sinal de *spare* (ambos recebidos de *Andamento*, que traduz o estado de *strike* ou *spare* para o sinal de funcionamento quando pertinente).

Seu funcionamento baseia-se em uma máquina de estados, onde cada estado é um segmento do visor. Utilizando dois divisores de *clock*, geram-se sinais de avançar a máquina de estados ou de parar de girar após dois segundos. A máquina de estados percorre os segmentos superiores do display quando o sinal de *spare* está ativo, e todos os segmentos quando o sinal de *strike* está ativo.

Seus sinais de saída são os vetores lógicos para *display* (passado desta entidade por *Andamento* diretamente para *Boliche*) e sinal de parada, utilizado como condição na entidade *Boliche*.

Ordena: Entidade responsável por ordenar de modo decrescente em conjunto o número dos jogadores e suas respectivas pontuações, com base nas pontuações, para serem exibidas posteriormente em ordem e indicar o(s) vencedor(es).

Recebe como entrada o *clock* e os pontos dos jogadores, e seus sinais de saída são um vetor com os pontos ordenados e outro com os números dos jogadores correspondentes à pontuação no mesmo índice. Para essa ordenação, utiliza a ideia do algoritmo de ordenação *“Bubble-Sort”*.

3.5.3) Entidades Auxiliares

Conversor BCD: Recebe como entrada a fonte de *clock* e um vetor binário de 9 bits correspondente a um valor de pontuação. Tem como objetivo transformar esse vetor lógico binário em um número na base 10, extrair dele o seu dígito de unidades, dezenas e centenas (em base decimal), e converter cada um deles em binário novamente (vetores de 4 bits) para serem concatenados e enviados na saída em um vetor de 12 bits.

É utilizado pelas entidades *Andamento* e *Final* para separar a pontuação dos jogadores em dígitos e depois enviar como entrada para *Bin2Dec*, que converte em sinais lógicos do *display* de 7 segmentos em decimal para serem exibidos.

Bin2Dec: É a entidade que recebe como entrada um vetor lógico de 4 bits, representado em binário o valor de um dígito decimal, realiza a conversão para segmentos do *display* e produz na saída um vetor lógico de 7 bits que os contém. A saída está em formato pronto para ser usado diretamente na placa em *displays* HEX. É utilizado em *Inicializador, Andamento,* e *Final*.

Clk_Div: É uma entidade simples que tem como único objetivo reduzir a frequência do *clock* de entrada por um divisor genérico. Recebe como entrada o *clock* base e um sinal de funcionamento (*enable*). Tem como saída o clock com frequência dividida.

Seu funcionamento se baseia em um contador, que acumula ciclos do clock de frequência até atingir o valor do divisor, gerando assim um pulso na saída e repetindo o processo. É utilizado em atividades que requerem temporização, como na entidade *Gira Visor*, e em *Final*.

Cont_1: Recebe como entrada um vetor lógico de 10 bits correspondente ao estado de cada pino (derrubado ou não derrubado). Soma quantos foram os bits com valor lógico alto (pinos derrubados) e tem como saída um vetor de 4 bits que corresponde ao somatório de pinos derrubados em binário. É utilizado na entidade *Rodada*.

3.6) Testes

Os testes foram realizados principalmente na placa, por conta da quantidade de variáveis a serem modificadas manualmente, como os diferentes estados de múltiplas máquinas. Os resultados, ao final, foram positivos, com nenhum erro encontrado (além de certas peculiaridades descritas na seção 3.7).

A ferramenta utilizada durante a disciplina para simular os circuitos dos laboratórios (University) não pôde ser utilizada para muitas das entidades do projeto, pois não reconhecia um dos pacotes utilizados, que contêm os tipos definidos para múltiplos registradores (*boliche_tipos*). Por isso, tentamos diretamente o ModelSim para testar a parte principal do projeto *(Calcula_Pontos*). Assim, a lógica de cálculo de pontos pôde ser testada fora da placa.

Outras entidades não foram testadas pois são mais simples, diretas (a maioria), e por falta de tempo (já que o ModelSim é uma ferramenta que demanda mais trabalho, volume de código e tempo que o University).

A simulação da entidade *Calcula_Pontos* é vista na figura 7 a seguir:

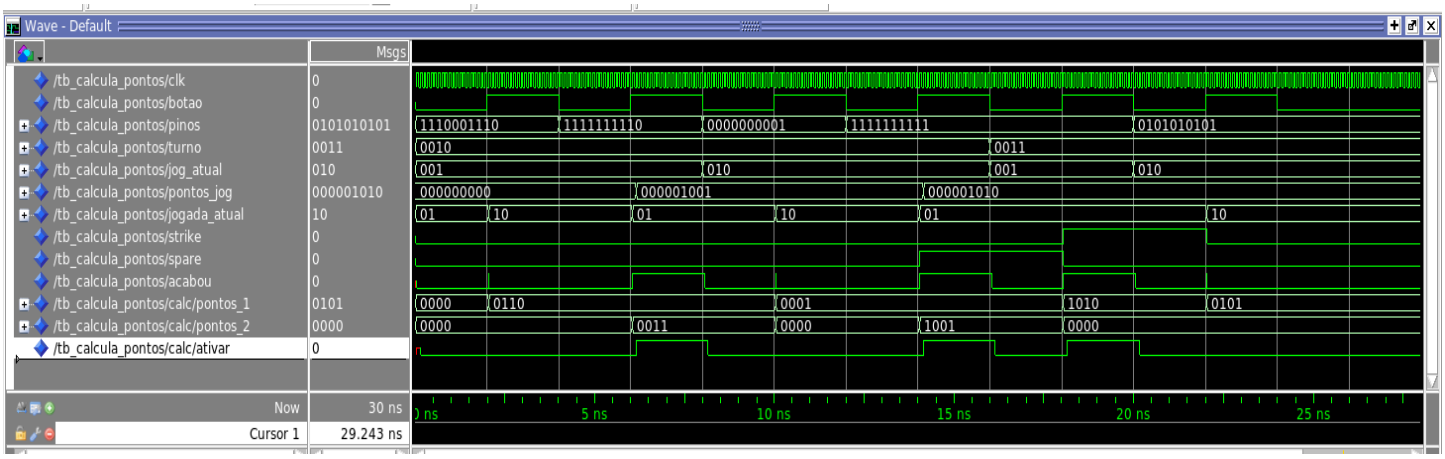

Figura 7 - Simulação da entidade Calcula_Pontos

Foi simulada uma seção de uma partida de dois jogadores (turnos 2 e 3), onde ocorre uma rodada normal, um *spare* e um *strike*. Pelos sinais *strike e spare* é possível ver que esses casos especiais foram tratados corretamente. Pelos sinais *pontos_1 e pontos_2*, é possível ver que os pontos de cada jogada foram lidos corretamente, e pelos sinais *ativar e acabou* é visível que o fim da rodada é sinalizado corretamente. *Pontos_jog* é o sinal que abriga a pontuação final do jogador, mas por estarmos testando apenas a lógica, esse valor é a pontuação da rodada.

Mesmo essa parte da lógica possui dezenas de casos especiais, quando se inclui o último turno e combinações de *strikes* e *spares* em sequência, então é inviável a inclusão de todas as simulações neste relatório. Todos esses casos especiais foram verificados na placa, e muitos deles foram mostrados na demonstração em aula.

A coerência dos resultados com as regras foram testados com um contador similar, mas implementado em software [10].

3.7) Modificações/Particularidades

O projeto final não passou por muitas modificações a partir da proposta. Ou seja, o resultado final é quase igual ao que a proposta previa. Porém, algumas pequenas alterações foram feitas, em relação à proposta original ou as implementações usuais de contadores de placar do boliche, por motivos específicos**.**

Uma das alterações foi na representação do turno atual. Na proposta original, a ideia seria representar os turnos numericamente de 1 a 10 no *display* de sete segmentos. Porém, ao analisarmos a quantidade de visores disponíveis na placa e a quantidade de informações que precisávamos representar, percebemos que fazer desse jeito deixaria os visores muito confusos e cheios de informação no décimo turno, pois todos os visores estariam sendo utilizados. Por isso, decidimos modificar essa parte, e representar o número do turno em hexadecimal (de 1 a A). Assim, utilizamos apenas um visor para representação dos turnos, e podemos deixar um visor sem propósito, deixando com uma aparência mais organizada e limpa.

Algo similar ocorreu com a representação do jogador atual. Embora a proposta não explicitamente diga que essa informação deve estar nos visores de sete segmentos, decidimos utilizar os LEDs para isso, por falta de espaço.

Normalmente, contadores de placar de boliche[10] usam uma representação sequencial de pontos/turno e, quando há um *strike* ou *spare*, esperam até que os bônus tenham sido feitos para atualizar a pontuação, ou seja, esperam até que a pontuação da rodada seja completa para mostrar ao jogador a pontuação. Por muitas questões, como a utilização dos visores de sete segmentos e o abandono da representação sequencial (e consequentemente a lógica do circuito), as pontuações são atualizadas imediatamente. Ou seja, se houver um *strike*, a pontuação é imediatamente incrementada em 10, e no próximo turno, a pontuação é duplicada e somada também imediatamente.

4) Conclusão

A implementação de um contador de placar para boliche foi bem executada, seguindo as exigências da lista de propostas [1]. Foi possível “traduzir” na íntegra as regras do boliche para a forma de circuito lógico, sem nenhuma simplificação das particularidades que essa modalidade esportiva oferece, principalmente nas pontuaçãoes bônus e rodada final.

O circuito lógico implementado é sintetizável e funcionou bem nas placas DE1-SoC. A compilação indicou que foram utilizados 472 registradores, 953 de um total de 32070 ALMs (3%) e 67 de um total de 457 pinos (15%) indicando que a placa está sendo subutilizada. Dada essa constatação, pode ser vantajoso financeiramente utilizar placas mais baratas e com menos recursos para servir como controlador de boliche, ou desenvolver uma placa dedicada para esse fim (ASIC), apenas com a lógica mínima necessária.

A lógica implementada poderia ser mantida para um caso real de controlador de boliche, bastando algumas modificações nos inputs/outputs. Os *switches* seriam substituídos por sensores que detectam a queda dos pinos, e o botão de avançar jogadas seria substituído por um sensor que detecta se a bola chegou ao fim da pista. Há a possibilidade de disponibilizar todas as informações que estamos mostrando nos displays e LEDs em uma interface gráfica de monitor. Caso houvesse tempo hábil, esse implementação em monitor poderia ser executada utilizando a interface VGA disponível na placa DE1-SoC, porém não seria possível implementar algumas funcionalidades obrigatórias na proposta escolhida, como o Gira Visores.

Em suma, é possível concluir que implementar a lógica do boliche diretamente em nível de circuito lógico de hardware em linguagem VHDL apresenta nível de dificuldade muito acima do que estávamos habituados em linguagens de software alto nível como C ou Java. Porém a maneira como foi feita nos permite sintetizar circuitos reais independentes de software, completamente otimizados para a tarefa específica que ele foi projetado para realizar e com grande capacidade de paralelização das atividades. O mesmo resultado (em termos de funcionalidade) poderia ter sido implementado com um microcontrolador e linguagem C, porém provavelmente apresentaria menor eficiência , desperdício de vários recursos do microcontrolador, além de maior custo envolvido.

Referências:

[1] Lista de Propostas. Disponível em:
< http://www.ic.unicamp.br/~sandro/cursos/mc613/1s2018/propostas.html>. Acesso em 13/06/2018;
[2] Manual da DE1-SoC. Disponível em:
<http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/DE1-SoC_User_manual.pdf>. Acesso em 13/06/2018;
[3] Bowling rules. Disponível em:
<https://www.playerssports.net/page/bowling-rules>. Acesso em 28/05/2018.
[4] Ten-pin bowling. Disponível em:
<https://en.wikipedia.org/wiki/Ten-pin_bowling>. Acesso em 17/05/2018.
[5] Process. Disponível em:
<http://www.gmvhdl.com/process.htm>. Acesso em 15/06/2018.
[6] RIGO, Sandro. Aula 8 - Máquinas de estado. Disponível em:
<http://www.ggte.unicamp.br/eam/pluginfile.php/463122/mod_resource/content/1/aula8_fsm-v2018.pdf>
Acesso em 17/05/2018 via plataforma Moodle.
[7] Frequency Divider with VHDL. Disponível em:
<https://www.codeproject.com/Tips/444385/Frequency-Divider-with-VHDL>. Acesso em 23/06/1998.
[8] SOUZA, Antonio Carlos. Aula 4 - Código BCD. Disponível em:
<http://www.ifba.edu.br/professores/antoniocarlos/aula4ads.pdf>. Acesso em: 17/06/2018.
[9] ALMEIDA, Jussara. Bubble-Sort. Disponível em:
<http://www2.dcc.ufmg.br/disciplinas/aeds2_turmaA1/bubblesort.pdf>. Acesso em 17/06/2018.
[10] Contador de Placar do Boliche. Disponível em:
<http://bowlinggenius.com> . Acesso em 28/05/2018.