Universidade Estadual de Campinas

MC613 - Laboratório de Circuitos Digitais

Professor Sandro Rigo

Relatório Final: Genius

Grupo 08

Caio Krauthamer
RA 165457 - Turma B - c165457@g.unicamp.br

Nathália Harumi
RA 175188 - Turma A - n175188@g.unicamp.br

Junho, 2018

Sumário

1 Teoria

Como conceitos abordados em aula, utilizamos principalmente:

Contadores

Com o uso de *Flipflops* e memória implícita, foi implementado um contador síncrono. Foram usados contadores para percorrer a sequência de cores tanto quando gerando cores quanto quando verificando a entrada de cores dada pelo jogador. Também foi utilizado quando mostrando as cores no monitor para conseguir um bom intervalo de tempo para mostrar e apagar a sequência.

Máquina de Estados

Máquinas de estados definem diferentes comportamentos para um circuito dependendo de seu estado anterior. Esse comportamento é implementado através de um processo que define o comportamento e as condições da transição de estado. Nesse projeto, foi utilizado uma *Máquina de Moore*, em que a saída (alterações feitas dentro de cada estado) depende exclusivamente do estado. Essa máquina é responsável por toda a lógica principal do trabalho em questão.

Memória

Memórias são dispositivos de armazenamento de dados. A memória implementada é de gravação síncrona e leitura assíncrona e utilizada para salvar a sequência de cores gerada durante o jogo.

2 Descrição do Sistema e Implementação

O sistema implementado é um jogo chamado ***Genius***. Nesse jogo, o jogador deve memorizar uma sequência de cores mostrada pelo sistema e repeti-la corretamente. A cada rodada, uma cor é adicionada a sequência, de forma a aumentar a dificuldade do jogo.

A implementação do sistema foi dada com 3 etapas principais de implementação: primeiramente, a lógica do sistema foi idealizada a partir do uso apenas da placa, com os botões e *leds* como interface entre usuário e sistema. Após, foi adicionada a *VGA*, com telas de início, fim e as cores mostradas na tela. Por fim, foi adicionado o mouse para usuário interagir com o jogo e retirados os botões e *leds* configurados na placa.

2.1 Primeira etapa: lógica do jogo

Essa primeira etapa contou com 4 blocos principais: *Genius* (que é a unidade de controle de todo o jogo), o *"ram_block"*(que é a memória principal do jogo), o *"generate_random_color"* (que gera as cores aleatórias do programa) e o *"checking_color"* (que confere se a cor escolhida pelo usuário é a cor correta da sequência).

ram_block

O *"ram_block"* é o bloco responsável pela memória principal do jogo. A implementação desse bloco foi inteiramente baseada no Laboratório 09 e, portanto, feito por instanciação direta. Como entrada, esse bloco recebe um sinal de *clock*, um *"write_enable"*, um dado para ser armazenado, um endereço para armazenar o dado/ler o dado e devolve o dado lido daquele endereço.

A memória do jogo é de escrita síncrona e leitura assíncrona, responsável por armazenar a sequência de cores geradas pelo próprio sistema. Dessa forma, o tamanho das palavras foram configurados para 3 bits, o necessário para descrever cada cor. Também, foi configurado uma memória de 1024 palavras, ainda que não utilizemos tudo devido uma decisão feita na Unidade de controle (que será citada em sua seção).

O teste desse bloco foi feito por meio do mesmo teste feito no Laboratório 09: uma *waveform* que testava as funcionalidades desse bloco.

generate_random_color

O *generate_random_color* é o bloco responsável por gerar aleatoriamente cores para serem adicionadas na sequência. Como entrada, ele recebe o *clock* de 50MHz e um *enable* e, como saída, ele devolve a cor gerada.

O método implementado nesse bloco é baseado em um *clock* de 50MHz e da resposta do usuário, o que torna esse processo aleatório. A ideia é que a cada subida do *clock*, uma cor é gerada de forma sequencial. Assim, quando necessário, há um sinal *"enable"* que faz com que a cor que foi gerada por último seja a escolhida para ser a resposta desse bloco. Esse *enable* é ligado a resposta do usuário, fazendo com que a cor seja aleatoriamente escolhida.

O teste desse bloco foi apenas teste por exaustão: o bloco foi executado várias vezes e conferido se as saídas diferiam e não possuíam nenhum tipo de sequência detectável.

checking_color

O *checking_color* é o bloco responsável por verificar a sequência de cores que o jogador dá de entrada. Como entrada, ele recebe: o sinal do *clock* de 50MHz; o estado da máquina de estados que está na Unidade de Controle; a cor lida na memória; se o botão da placa foi clicado (será modificado posteriormente) e qual botão foi clicado (será modificado posteriormente). Como saída, devolve se o jogador errou a sequência e qual cor da sequência foi checada (a primeira, segunda...).

O método implementado se baseia na ideia de verificar cada cor da sequência se baseando em um contador interno. Quando o estado da máquina é *"eval_color"*, ou seja, o estado em que se verifica a sequência, esse bloco compara a cor lida na memória com a entrada do usuário. Caso seja igual, a *flag "fail"* é igual a 0; caso contrário, 1. Dado isso, é incrementado um contador interno que fala que já foram verificadas "x" cores da sequência. Tanto a *flag "fail"* e o contador são retornados. Caso o estado seja diferente do de verificação, o contador é zerado.

Esse bloco foi testado através de *waveform*.

Genius

O *genius* é a Unidade de Controle (**UC**) do jogo, ou seja, ele liga todos os blocos e implementa a lógica principal do programa. Como entrada e saída, a **UC** recebe e preenche sinais da placa, principalmente, e alguns sinais dos periféricos (que serão adicionados posteriormente. Até esse ponto, são apenas sinais da placa).

O bloco declara todas as componentes adicionadas até então: *ram_block, generate_random_color, checking_color* e *bin2hex* (bloco que transforma números de 4 *bits* em números hexadecimal para o mostrador da placa através de um bloco de *select*). Também declara todas os sinais auxiliares, que incluem sinais da *Máquina de Moore* e outros sinais para controle.

A lógica principal desse bloco se concentra exatamente na *Máquina de Moore* implementada. Essa máquina é responsável por transitar entre as fases: *"start"*, que espera o comando do jogador para iniciar o jogo; *"generating_color"*, que gera a cor da sequência; *"show_color"*, que mostra a sequência de cores pro jogador, incluindo a última cor anteriormente gerada; *"eval_color"*, que recebe as cores que o jogador coloca de entrada e as compara com as cores da sequência correta; e *"final"*, que é quando o jogador acerta todas as cores das 15 rodadas e vence o jogo.

Todas os componentes são instanciados também, sendo que eles são acionados através de sinais de *"enable"* que são ativados e desativados de acordo com os estados da máquina.

Ao iniciar o jogo, a máquina de estados está em *"start"*. Nesse ponto, todas as variáveis são inicializadas com os valores que lhes cabem, como '0' ou '-' (*don't care*). Quando o jogador aciona todos os botões da placa ao mesmo tempo, o estado é mudado para *"generating_color"*.

No *"generating_color"*, o *enable* para gerar a cor é acionado, de forma a ativar a instância do bloco *"generate_random_color"*. Essa cor é salva na memória (usando o bloco *"ram_block"*) no endereço que equivale a rodada que o jogador se encontra, ou seja, a quantas cores já foram geradas. Após, esse contador de cores é incrementado e o estado é trocado para *"show_color"*.

O estado *"show_color"* é responsável por mostrar sequência de cores para o jogador. Nesse bloco, o endereço de leitura na memória é alterado para *"colors_shown"*, que é um contador de quantas cores da sequência já foram apresentadas pro usuário. Com a cor lida pelo *"ram_block"*, essa cor é apresentada. Para que isso seja feito em um intervalo de tempo consistente, há um contador de 1 a 70 milhões que dá o atraso necessário. Na metade da contagem, a cor começa a aparecer e permanece por mais 35 milhões de iterações. Quando o contador atinge 70 milhões, a cor para de aparecer, o número de cores mostrados é incrementado. A cada iteração, também se verifica se o número de cores mostrados é igual ao número da rodada (que é o mesmo número de cores geradas para a sequência). Caso seja, o estado é trocado para *"eval_color"*. Para a implementação dessa funcionalidade, não foi usado um *loop*, mas sim a própria máquina de estados. Assim, a cada vez que se executa esse estado, o contador é incrementado em 1 e, portanto, se executa esse estado repetidamente até que se iguale as cores mostradas e o número da rodada.

O estado seguinte, *"eval_color"*, é o que controla as entradas do jogador e as compara com a sequência correta. Para isso, o bloco *"checking_color"* é acionado. Caso o *"checking_color"* encontre alguma falha do usuário, o estado volta para *"start"* para começar um novo jogo; caso todas as cores sejam verificadas e o número de rodadas chegue a 15, o estado é trocado para *"final"*; por fim, caso toda a sequência seja verificada e esteja correta, mas ainda não é a rodada final, o estado é trocado para *"generating_color"* para gerar uma nova cor para a sequência do jogo. Nesse ponto, é possível trocar o número de rodadas que indiquem a vitória do jogador. Atualmente, esse valor está para 15 rodadas por decisão do grupo.

O estado *"final"*, nesse ponto de desenvolvimento, é igual ao estado *"start"*, com a diferença que o usuário aperta os 4 botões da placa para mudar para o estado *"start"*.

2.2 Segunda etapa: adicionando *VGA*

A segunda etapa da implementação se ateve em adicionar o monitor no projeto. Mesma com essa adição, os *leds* adicionados na primeira etapa foram mantidos para efeito de testes. A adição da *VGA* trouxe diferenças, principalmente, em relação à saída do programa, que passou a ser feita no monitor. Portanto, essa etapa fez com que usuário precisasse manipular tanto a placa (seus botões para descrever a sequência que ele acreditava ser a correta) quanto o monitor.

VGA_con e VGA_pll

Esses dois blocos foram integralmente fornecidos durante a disciplina, sem necessidade de alteração. São blocos que lidam com a leitura dos sinais da *VGA* e tratamento desses sinais.

VGA_controller

O *"VGA_controller"* faz todo o controle do monitor entre ler/enviar os sinais tradados e fazer todas a manipulação de *VGA*. Para isso, ele usa dois processos que varrem toda a tela, *pixel* a *pixel*.

Para a tela de *"start"* e *"final"*, há uma série de condições para pintar os *pixels* que formam as palavras *"START"* e *"END"*. Outra forma de fazer seria usando um *bitmap*, mas essa foi a primeira implementação idealizada e mantida até o fim do projeto. Além disso, caso o estado seja *"checking_color"*, que espera a entrada do jogador, todas as cores são mostradas na tela e isso também é feito a partir de uma série de condições que pintam os quadrados coloridos. Ademais, no estado *"show_color"*, esse bloco também é responsável por mostrar a cor da sequência através de condições que selecionam os pixels que precisam ser pintados e de que cor.

ram_block, generate_random_color e checking_color

Esses blocos permaneceram sem alteração entre a primeira e a segunda etapa.

Genius

Esse bloco, além de toda lógica implementada e explicada na subseção anterior, declarou o novo componente *"VGA_controller"* e o instanciou para fazer a comunicação com o monitor. Além disso, o estado de *"show_color"* começou a mandar sinal para o esse novo bloco e não mais apenas para a placa.

2.3 Terceira etapa: adicionando mouse e finalizando projeto

A terceira e última parte de implementação foi responsável pela adição do *mouse* e da verificação de possíveis problemas. Essa parte contou com a modificação de mais blocos, uma vez que a entrada foi alterada da placa para o periférico. Os blocos finais estão representados na figura página seguinte, que ilustra os blocos e a comunicação entre eles.

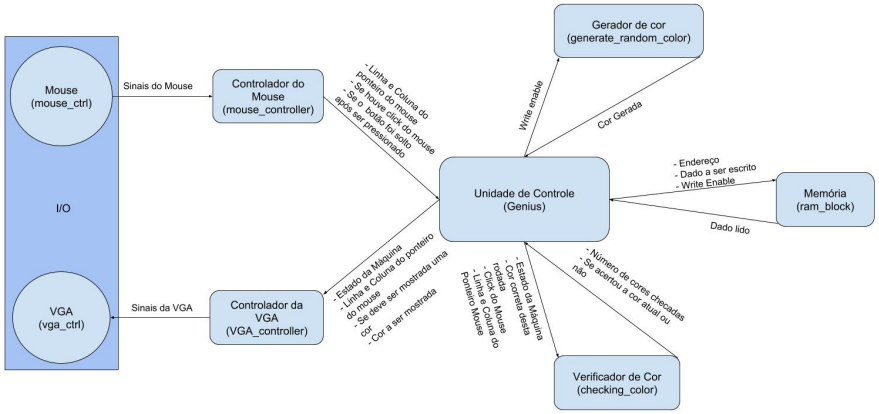

Figura 1. Diagrama de blocos final utilizada do trabalho. [4]

mouse_ctrl e ps2_iobase

Assim como o *"VGA_con"* e o *"VGA_pll"*, esses dois blocos foram fornecidos em aula e não necessitaram nenhuma alteração. Eles tratam os sinais de entradas e saídas do monitor.

mouse_controller

O *mouse_controller* é um bloco baseado em um código fornecido em aula, porém com muitas modificações. Ele possui de entrada e saída sinais de interface entre a placa e o *mouse*.

Para fazer o movimento do *mouse*, o bloco possui um processo que estabiliza a linha e coluna do ponteiro através do tratamento dos sinais recebidos pelo periférico a partir de derivadas e outras contas com sensibilidade do *mouse* e os limites do monitor. Esse processo é sensível a qualquer mudança de movimento que o periférico detectar.

Além do movimento do *mouse*, o *"mouse_controller"* trata os cliques, de forma a deixar consistente e sem oscilações qualquer clique que o jogador fizer no botão direito. Isso inclui oscilações devido a duração do clique.

VGA_controller

Apesar de maior parte desse bloco se manter igual, há uma alteração que implementa a exibição do cursor do *mouse* no monitor. Para isso, foi adicionada uma entrada no bloco que recebe a posição do cursor. Como os processos nessa implementação varrem todo o monitor, *pixel* a *pixel*, quando o *pixel* que está

sendo tratado no monitor coincidir com o da posição do *mouse*, esse *pixel* é pintado de branco. Assim, o cursor fica disponível para o jogador. O tratamento de posição do cursor para não estourar as margens do monitor é feito no bloco *"mouse_controller*, como explicado acima.

VGA_con, VGA_pll, ram_block, generate_random_color

Esses blocos permaneceram sem alteração entre a segunda e a terceira etapa.

checking_color

Esse bloco, ao invés de receber a entrada do usuário através dos botões, passou a receber a entrada pela posição e clique do mouse. Dessa forma, para conferir a cor que o jogador escolheu, cada cor tem rua região pré-definida nesse método e se verifica o clique do cursor está dentro da região da cor correta - diferente de antes, que era apenas conferir qual botão havia sido acionado.

Genius

Esse bloco passou a declarar e instanciar a componente *"mouse_controller"*. Além disso, foram tirados todos os sinais que faziam a comunicação de entrada e saída com a placa, ou seja, os botões e *leds* que eram a interface com o jogador. Também, passou a enviar os sinais de posição do cursor e indicador de clique para o bloco *"checking_color"*, que antes utilizava sinais vindos dos botões da placa.

A máquina de estados final está apresentada abaixo, com seus respectivos *triggers* de troca de estado e suas variáveis modificadas dentro dos estados (com exceção da *"eval_color"*, que não possui nenhuma variável representada na imagem que dependa de enunciados de condição).

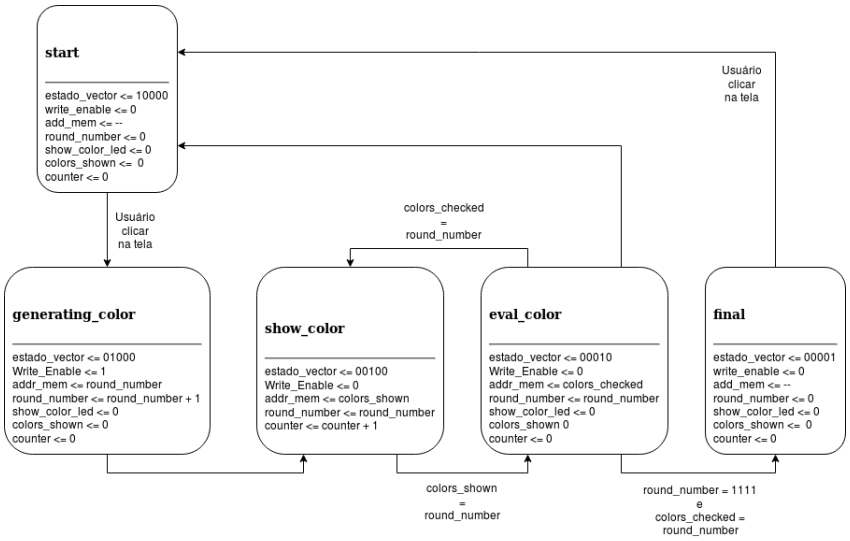

Figura 2. Esquema da máquina de estados final utilizada no trabalho. [5]

3 Conclusões

A elaboração desse trabalho colocou em uso muitos dos conceitos vistos em sala de aula, incluindo "Máquinas de estados" e "Memória". Foi possível compreender a extensão do conhecimento obtido durante a aula e entender as facilidades e dificuldades de cada tópico.

A parte que desprendeu de menos trabalho foi a confecção da memória *"ram_block"*, uma vez que o grupo teve facilidade nesse tópico durante a aula. Assim, foi possível reutilizar o código já produzido no Laboratório 09. Por outro lado, o grupo teve dificuldade na configuração e sincronização do mouse no que tange o ponteiro do mouse e sua exibição no monitor.

Dessa forma, a elaboração do *Genius* ajudou a compreender melhor a linguagem *VHDL* e os conceitos de circuitos digitais, fazendo mais completa a compreensão do conteúdo da disciplina.

4 Errata

Nos comentários dos *VHDL* enviados, onde está escrito *'Mealy Machine'*, leia-se *'Moore Machine'*.

5 Referências e Links das Imagens

Abaixo encontra-se as referências utilizadas durante o curso e para a confecção do trabalho, bem como o link das imagens adicionadas nesse relatório para conferência de autenticidade e visualização em melhor resolução.

Referências

[1] Stephen Brown and Zvonko Vranesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill.

[2] James O. Hamblen and Michael D. Furman. *Rapid Prototyping of Digital System - A Tutorial Approach*. Second Edition. Kluwer Academic Publishers

[3] Peter J. Ashenden. *The VHDL Cookbook*
`https://tams.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf`

[4] Caio Krauthamer e Nathália Harumi *Diagrama de Blocos.*
`https://docs.google.com/drawings/d/1eZc6H3PAFW4ASX2akwmCl8fmGFqYnKZJbAY1n89qUo8/edit?usp=sharing`

[5] Caio Krauthamer e Nathália Harumi *Máquina de Estados: Genius*
`https://drive.google.com/file/d/1Vb0Q-pd-G1Fs5XIbQ9vcgfeO4cD5j1Hy/view?usp=sharing`