Implementação Orientada a FPGA's de um Oscilador Digital Multiplexado

Mateus Henriques Negrelli, Márcio Brandão

Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Brasília – DF – Brasil

negrelli@aluno.unb.br, brandao@unb.br

Abstract. *This paper describes an FPGA-oriented implementation of a multiplexed digital oscillator. The multiplexation of oscillators is done through a pipeline of the tasks involved. The ultimate goal of this work is to provide an audio synthesis framework for an FPGA.*

Resumo. *Este artigo descreve uma implementação orientada a FPGA's de um oscilador digital multiplexado. A multiplexação dos osciladores é feita usando um pipeline das tarefas necessárias. O objetivo final desse trabalho é fornecer um framework de síntese de áudio em FPGA's.*

1. Introdução

Aplicações de síntese de áudio digital, em sua grande maioria, valem-se de osciladores em maior ou menor escala. Seu funcionamento é simples e conhecido: a partir de um conjunto de entradas de freqüência, amplitude e fase, um circuito contendo um ciclo de onda em memória gera como saída a forma periódica desejada [Roads 1996]. Alia-se a isso o fato demonstrado por Fourier [Lathi 1998, Spiegel 1974] de que qualquer sinal pode ser obtido a partir da combinação de ondas senoidais. Essa idéia é a base da técnica de síntese aditiva de áudio: um conjunto de osciladores senoidais operando em paralelo, com freqüências diferentes e amplitudes variáveis no tempo, fornece uma maneira versátil e flexível de gerar formas de onda mais complexas [Jansen 1991]. A síntese aditiva não é a única aplicação desta idéia, claro, mas é uma das mais difundidas.

Contudo, há maneiras mais eficientes e elegantes de se obter o mesmo resultado. Snell (1991) propõe um oscilador multiplexado em hardware que pode gerar, em tempo real, um formato de onda complexo através da soma ponderada de um grande número de componentes senoidais. A proposta depende das velocidades de *clock* disponíveis em processadores, muito superiores às freqüências padrão de amostragem de áudio: subdivide-se o ciclo de amostragem em segmentos, e em cada um deles obtém-se a amostra de áudio relativa a uma componente. Ao final, as várias amostras são somadas, gerando-se assim a onda composta.

Este trabalho apresenta a implementação de um oscilador multiplexado em FPGA, com capacidade de operar em tempo real, visando a criação de um *framework* para o desenvolvimento de aplicações envolvendo diferentes técnicas de síntese de áudio em dispositivos reconfiguráveis com poucos recursos.

1.1. Oscilador Simples

A idéia central de um oscilador simples baseado na técnica de busca em tabela é o envio para um conversor digital/analógico (DAC), a uma taxa de amostragem fixa f_S *(= 1/T)*, amostras de um ciclo de uma senóide. Para armazenar o ciclo de onda necessário, usa-se normalmente uma tabela circular de comprimento *L* (Figura 1).

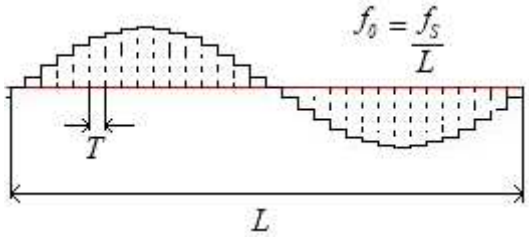

Figura 1. Ciclo amostrado de senóide.

Introduz-se aqui também a idéia do incremento *I*, o número de amostras percorridas na tabela a cada ciclo *T*. Caso o incremento de indexação da tabela seja dobrado, o novo sinal terá metade das amostras em um ciclo, correspondendo ao dobro da freqüência. A equação abaixo mostra a expressão geral da freqüência f_0 da senóide resultante em função do incremento *I*, do comprimento *L* da tabela do seno, e da freqüência f_S de amostragem.

$$I = f_0 \, . \, L \, / \, f_S$$

A Figura 2 apresenta a arquitetura básica de um oscilador simples baseado na técnica de busca em tabela. Verifica-se a presença de:

- uma memória com *L* palavras, contendo as amostras da senóide;
- um registrador armazenando o valor do incremento de ângulo de fase;
- outro registrador, para armazenar e atualizar o valor do ângulo de fase;
- um somador que completa o acumulador de ângulo de fase;
- um DAC que conecta o oscilador ao resto do circuito.

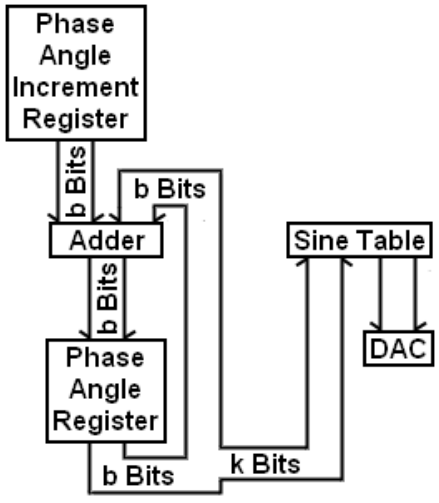

Figura 2. Estrutura geral do oscilador simples.

Vale notar que o registrador de incremento atende à representação de números reais em ponto fixo, com seus *k* bits mais significativos representando a parte inteira. O endereço de memória, portanto, passa por um processo de truncamento dos $b - k$ bits correspondentes à parte fracionária do registrador de incremento. Com a escolha de um comprimento adequado para a tabela de seno, é possível se obter uma relação sinal-ruído adequada para sinais de áudio, conforme descrito em Moore (1991).

1.2. Pipeline

Na arquitetura de computadores, a técnica de *pipeline* é fundamental para tornar processadores rápidos, e sua essência se baseia em aumentar o *throughput* do sistema através do funcionamento em paralelo de suas partes, geralmente denominadas estágios [Patterson e Hennessy, 2005].

Num processador sem *pipeline*, quando uma instrução passa de um estágio para o seguinte, os estágios anteriores ficam inativos até a finalização da instrução atual, quando a próxima finalmente entra no início do caminho de dados. Num processador com *pipeline*, contudo, ocorre a sobreposição de múltiplas instruções na execução. Isso quer dizer que, assim que uma instrução passa do primeiro estágio para o segundo, a instrução seguinte começa a ser processada.

Observa-se, então, o chamado paradoxo do *pipelining*: o tempo de execução total de cada instrução não se altera nesse novo paradigma. O aumento de velocidade é gerado pela paralelização das etapas, e está associado ao aumento de *throughput* mencionado acima.

Na Figura 3, vê-se a execução de três instruções num sistema com *pipelining* implementado. Um programa que normalmente levaria 9 ciclos de relógio para ser executado agora leva apenas 5. Assumindo-se uma duração semelhante para os estágios, o processador com *pipeline* seria três vezes mais rápido que o padrão caso o número de instruções tendesse a infinito.

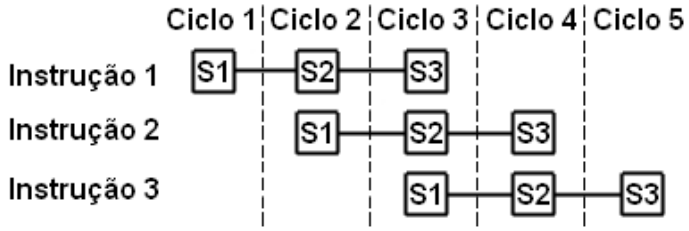

Figura 3. Exemplo de fluxo de instruções num *pipeline* de três estágios.

2. Oscilador Multiplexado

A Figura 4 apresenta a arquitetura do oscilador multiplexado em FPGA, baseada na implementação em hardware apresentada por Snell (1991).

Como mencionado na Seção 1, a base dessa proposta é a relativamente baixa taxa de amostragem de áudio quando comparada com a freqüência dos *clocks* de processamento computacional atualmente disponíveis: durante um único período de amostragem de áudio, podem-se realizar diversas operações de período curto, sincronizadas com o que chamamos de *clock* de componente. No caso do oscilador multiplexado, essas operações envolvem o cálculo das amostras associadas às várias

componentes senoidais de uma forma de onda complexa. Desse modo, faz-se necessário apenas um dispositivo para realizar uma tarefa normalmente associada a múltiplos osciladores simples.

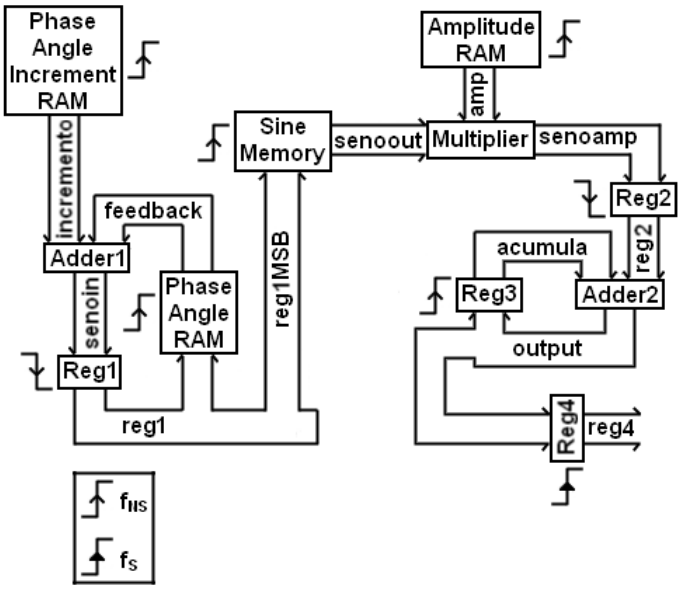

Figura 4. Estrutura geral do oscilador multiplexado.

O circuito é controlado por dois *clocks* principais (ver Seção 3.5), um associado à amostragem (ClkSamp, freqüência f_S) e o outro ao cálculo de componentes senoidais (ClkComp, freqüência f_{NS}). A taxa f_{NS} é o produto da freqüência f_S pelo número n de componentes (Figura 5).

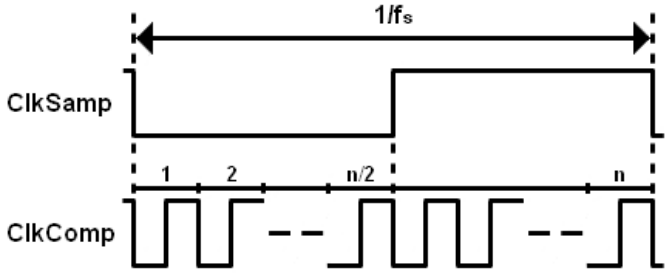

Figura 5. Comparação entre as freqüências f_S e f_{NS}.

O processo é dividido num *pipeline* de três estágios, separados pelos registradores Reg1 e Reg2. O primeiro estágio é a soma progressiva, para cada componente senoidal, de incrementos cujo valor base depende da freqüência desejada ($I = f_0.L/f_S$). O segundo estágio usa os valores de incremento calculados como endereços para acessar um ciclo de senóide em memória. A amplitude de cada componente senoidal é determinada, a cada período de amostragem, pelo multiplicador localizado imediatamente antes de Reg2, em conjunto com os valores internos da Amplitude RAM. O terceiro estágio, por fim, acumula as amostras das várias componentes em Reg3, preparando a saída para o *clock* de amostragem.

2.1. Memórias

A principal diferença entre o oscilador simples e o multiplexado é a presença, neste, de três memórias a mais. Num único ciclo de componente, todas as componentes são processadas uma vez, e seus valores associados devem permanecer armazenados para serem utilizados nos cálculos um período mais tarde. Entram em ação aí a Phase Angle Increment RAM, a Phase Angle RAM e a Amplitude RAM, memórias com número de palavras igual ao número de componentes senoidais, cujo valor máximo é dependente dos gargalos do circuito (ver Seção 2.4).

A quarta memória, Sine Memory, por outro lado, tem o mesmo papel que no oscilador simples, que é o de armazenar amostras de um ciclo completo de uma senóide para permitir a busca em tabela.

2.2. Endereçamento

Um circuito de endereçamento faz com que as três memórias passem sequencialmente por todas as suas palavras uma vez por ciclo de amostragem, sendo atualizados todos os valores no oscilador. A memória do seno é endereçada no circuito principal pelos bits mais significativos dos valores de ângulo de fase das componentes.

2.3. Registradores

Há registradores com duas funções principais no oscilador multiplexado. A primeira delas é, como mencionado anteriormente, a divisão do circuito em três estágios de *pipeline*. Isso se dá de maneira simples: os registradores Reg1 e Reg2 armazenam valores em um ciclo, passando-os adiante quando os três estágios estão prontos para processar a próxima componente. Reg4 está no final do terceiro estágio, o último do *pipeline*, e apenas trabalha como porta de saída do circuito.

A segunda função é a construção de acumuladores. As saídas de Reg1 e Reg3 são barramentos que realimentam um ponto anterior do oscilador, no intuito de manter somas atualizadas. No caso de Reg1, que desempenha simultaneamente as duas funções principais descritas, há a acumulação dos valores de ângulo de fase para cada componente senoidal, que posteriormente endereçam a Sine Memory. Reg5 acumula os valores finais de cada componente, gerando-se ali o valor final da amostra em si para cada período de amostragem.

2.4. Componentes Adicionais

Há somadores simples nos acumuladores, que produzem, a todo ciclo de componente, a adição do valor acumulado com o novo valor recebido para a componente atual. Há também um multiplicador que altera a saída da memória de seno por um fator de amplitude, tendo cada componente senoidal seu próprio fator armazenado na Amplitude RAM. Esse processo de multiplicação é o gargalo teórico do circuito, ou seja, a velocidade de funcionamento do oscilador é mais severamente limitada pelo segundo estágio do *pipeline*.

3. Implementação

Para implementar o oscilador multiplexado em FPGA, utilizou-se a versão 9.1 do programa Quartus II e o kit lógico DE2-70 [Altera, 2013]. Os componentes necessários

foram desenvolvidos com uma combinação de diagramas de blocos e programação VHDL [Mealy e Tappero, 2013].

Para esta implementação específica, escolheu-se construir o oscilador com capacidade para lidar com 384 componentes senoidais simultâneas, operando a uma freqüência de amostragem de 48 kHz, o que faz com que o *clock* de componente tenha que operar a 18,432 MHz.

Cada uma das partes constituintes do oscilador descritas na Seção 2 foi implementada em separado e testada isoladamente com respeito à sua resposta temporal. Desse modo, foi possível verificar e adequar os atrasos individuais de cada bloco para a operação de *pipeline*.

3.1. Memórias

A Sine Memory foi construída com uma simples memória ROM de 1024 palavras, contendo um ciclo completo de senóide. Esse comprimento de tabela possibilita a obtenção de uma relação sinal-ruído adequada para sinais de áudio [Moore, 1991].

Phase Angle Increment RAM, Phase Angle RAM e Amplitude RAM são memórias RAM comuns, com escrita e leitura, e número de palavras igual ao número máximo de componentes escolhido.

3.2. Endereçamento

O circuito de endereçamento utiliza um contador que, durante um ciclo de amostragem, conta de 0 a 383, para que as três memórias acessem os valores adequados a cada ciclo de componente. Ao se processar a última componente, a contagem é reiniciada.

3.3. Registradores

Com exceção de Reg3, que possui também uma entrada de *clear* assíncrono (explicada na Seção 3.5), todos os registradores trabalham com os pinos básicos de *clock* e entrada/saída de dados.

Além dos registradores mencionados previamente, a implementação prática do *pipeline* demandou um registrador auxiliar para atrasar em um ciclo de componente o valor de endereçamento. Esta medida se justifica pois, em um dado ciclo de componente *n*, o estágio 1 do *pipeline* processa a componente senoidal. Enquanto isso, o estágio 2 ainda processa a componente *n-1*, devendo lidar com a componente *n* somente no ciclo *n+1*. Dessa maneira o registrador auxiliar, para endereçar a Amplitude RAM, é endereçado pelo valor do ciclo de componente *n-1*.

3.4. Componentes Adicionais

Os somadores, por apresentarem um pequeno tempo de resposta quando comparado ao do multiplicador, são considerados componentes pouco críticos, e implementados com somadores paralelos padrão. Já para a implementação do multiplicador, levou-se em conta que multiplicadores construídos com unidades lógicas de uma FPGA podem ser bastante lentos se comparados a circuitos dedicados de multiplicação, estreitando o gargalo teórico mencionado anteriormente. Essa alternativa é disponibilizada em alguns kits lógicos de FPGA’s, e é a que se escolheu usar nesta implementação.

3.5. Esquema de Clock

Em implementações práticas de circuitos que devem funcionar em tempo real, um grande problema que surge ao se sair do âmbito teórico ideal são as restrições temporais dos tempos de *setup*, atrasos de processamento e necessidades de sincronização. O circuito do oscilador multiplexado não é exceção, exigindo cuidado em todos esses aspectos.

Primeiro, e mais evidente, o ciclo de componente deve ser longo o suficiente para comportar os atrasos de processamento de todos os blocos. Como mencionado anteriormente, todos os blocos foram testados individualmente, podendo-se tirar conclusões rápidas sobre a viabilidade do número de componentes senoidais escolhido.

Além disso, devem ser levados em consideração os tempos de *setup* dos vários blocos, principalmente das memórias. Como os registradores entre estágios de *pipeline* devem passar adiante os valores finais de cálculo, não os iniciais, não é possível usar um *clock* unificado para todos os componentes. Optou-se, então, por dividir as tarefas do oscilador entre as bordas de subida e descida do *clock* de componente: na borda de descida são ativados os registradores 1, 2 e auxiliar, além do circuito de endereçamento. Desse modo, as entradas de cada estágio estão prontas na borda de subida do relógio.

Por fim, o processo de amostragem em si gerou a necessidade de se criar um novo bloco, chamado Amostra Enable. O oscilador multiplexado não precisa receber como entrada um relógio na taxa de amostragem, apenas o *clock* de componente. Assumindo-se que a freqüência deste foi definida corretamente, o intervalo de tempo entre dois cálculos da mesma componente senoidal tem o valor $T = 1/f_S$. Segue-se desse raciocínio que, para corretamente amostrar a onda complexa, basta se extrair o valor no terceiro estágio do *pipeline* assim que for processada a última componente senoidal. A função de Amostra Enable é detectar a presença desta componente no último estágio do *pipeline* e, quando os cálculos forem concluídos, ativar a passagem de dados por Reg4, simultaneamente zerando o valor de Reg3. Em seguida, reinicia-se o ciclo de processamento para gerar a próxima amostra de áudio.

4. Sincronização com o Kit Lógico

Com a estrutura do oscilador multiplexado estabelecida, o próximo passo na implementação do *framework* de síntese foi a construção da interface entre o oscilador e o codec de áudio da FPGA, o WM8731. A DE2-70, em seus manuais, já fornece alguns blocos pré-prontos que facilitam esse processo, ficando a cargo do implementador a geração dos *clocks* necessários, e do canal serial de passagem de dados.

4.1. Clock de Entrada

A DE2-70 possui dois *clocks* internos, um de 50 MHz e outro de 28 MHz. Os blocos pré-prontos mencionados acima exigem um *clock* de 18,432 MHz, assim como o oscilador multiplexado. Desse modo, escolheu-se utilizar em sua entrada uma PLL para criar, a partir do *clock* de 50 MHz, um *clock* com o valor adequado.

4.2. Serialização de Dados

O modo de funcionamento padrão do codec da DE2-70 é o chamado *Left Justified*, ilustrado na Figura 6. Os valores das amostras de áudio são passados serialmente, do MSB para o LSB, divididos em dois canais. O *Left Right Clock* (LRC) separa os canais: uma borda de subida indica o começo do canal esquerdo, e uma borda de descida, o começo do canal direito. O *Bit Clock* (BCLK) sinaliza a passagem de dados do oscilador para o WM8731: uma borda de subida indica que o codec deve receber o valor atual do canal de dados (DAT).

Vale notar que o LRC tem freqüência f_S, de 48 kHz. Assim, BCLK tem freqüência igual a n vezes f_S, onde n é o número de bits nas palavras que representam as amostras de áudio. Os valores padrão para n são 16, 20 ou 24, todos abaixo da ordem de grandeza do número de componentes senoidais no oscilador multiplexado. Isso significa que o BCLK não é um gargalo do circuito, com bastante tempo disponível para a realização de suas operações associadas.

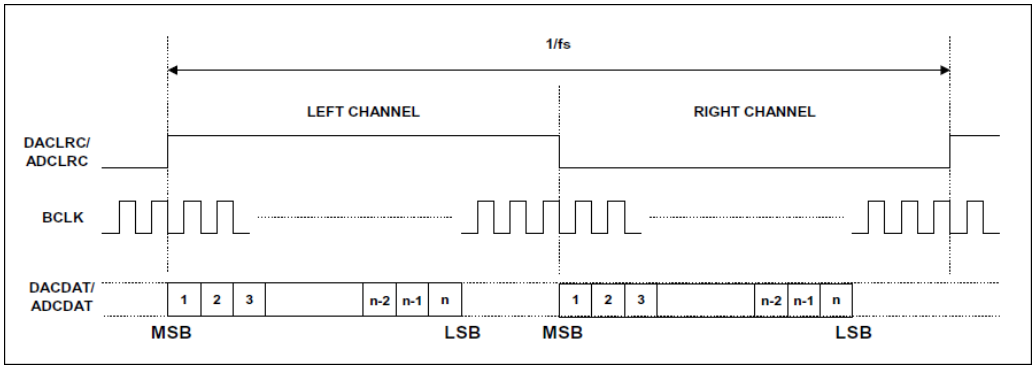

Figura 6. Left Justified Mode do codec WM8731.

O oscilador multiplexado, em todos os seus estágios, trabalha com dados paralelos. Para possibilitar a interação com o codec, adicionou-se um novo registrador após Reg4, que recebe sua saída e serializa o valor obtido, gerando DAT. Esse registrador opera com *Bit Clock* negado, de modo que a relação entre BCLK e DAT aconteça como na Figura 6. Além disso, ele usa um sinal de *load*, descrito na seção seguinte, para carregar o valor paralelo a ser serializado.

4.3. Bloco de Sincronização

Para cuidar da geração síncrona dos *clocks* associados ao Left Justified Mode, construiu-se um novo bloco. Ele recebe como entrada a saída da PLL, de freqüência 18,432 MHz e, contando os ciclos desse relógio, gera as bordas de subida e descida para LRC e BCLK.

Adicionalmente, o bloco de sincronização usa o contador associado a LRC para prever as trocas de nível e enviar adequadamente o sinal de *load* para o registrador de serialização. Sendo assim, a interface entre o oscilador multiplexado e o WM8731 é realizada de maneira ordenada, sem risco de offsets temporais que comprometam o funcionamento do circuito completo.

5. Normalização

Com a estrutura descrita até aqui, já é teoricamente possível gerar sons na FPGA. Contudo, propõe-se ainda um novo ponto de preocupação: cada componente senoidal possuirá uma amplitude máxima, proveniente da quantidade de bits nas saídas das memórias de seno e amplitude. Esse máximo individual de cada componente será apenas 1/384 do valor total com o qual o oscilador está preparado para lidar.

Em outras palavras, o problema é que, para entradas com poucas componentes, a saída terá amplitudes reduzidas. A solução dada foi substituir-se `Reg4` por um bloco de normalização, que permite ao usuário amplificar o sinal acumulado.

Esse bloco funciona do mesmo modo que `Reg4`, com a adição de um fator de *shift*. Sua função é realizar *shift lefts* aritméticos sobre o valor final da amostra a cada ciclo, de maneira que somas de poucas componentes possam aproveitar melhor a faixa dinâmica disponibilizada. O usuário pode escolher a execução de zero a nove *shifts*, dependendo de ele estar usando 384 componentes, 256, 128, 64, e assim por diante.

É importante notar que cabe ao próprio usuário saber quantas componentes estão em uso, e a escolha correspondente do número de *shifts*. O mau uso dessa função, com excesso de movimentações à esquerda, pode corromper os valores finais das amostras, deteriorando a relação sinal-ruído do sinal de áudio.

6. Testes Iniciais

Após as considerações feitas até agora, decidiu-se confirmar na prática os resultados previstos, com um teste simples que utilizou um número reduzido de componentes senoidais correspondentes a notas da escala de Dó maior. Estas podiam ser acionadas por meio de chaves do kit lógico, de modo que se pudesse verificar as freqüências das diversas senóides, os intervalos musicais entre elas, e a ação do mecanismo de normalização.

Instanciado o circuito em modo de testes na FPGA, os resultados verificados se encaixaram no esperado: o kit lógico produziu som com sucesso, e confirmou-se auditivamente que as notas foram geradas de maneira correta, tanto melódica quanto harmonicamente. O processo de normalização também desempenhou seu papel como previsto, permitindo controlar a amplitude do sinal de saída satisfatoriamente para diversas configurações de componentes.

7. Discussão

A intenção inicial desse trabalho, como mencionado na Seção 1, era implementar uma ferramenta capaz de operar em tempo real, processando formas de onda elaboradas para síntese de áudio em dispositivos com capacidade de processamento restrita.

O primeiro ponto a ser mencionado é a operação em tempo real, verificada com sucesso. O oscilador multiplexado atualiza e acumula valores constantemente a partir de suas memórias, calculando-se novas amostras a cada ciclo, sendo que o usuário pode interferir ativamente na geração desses resultados.

Em seguida, fala-se sobre o aspecto da complexidade das formas de onda passíveis de geração. Como descrito na Seção 6, até agora testes práticos foram

realizados com no máximo oito componentes senoidais. Todavia, foi verificada numericamente, via simulação no Quartus, a capacidade de paralelização de 384 componentes do oscilador multiplexado. Os trabalhos futuros mencionados na Seção 8 serão essenciais para a confirmação auditiva dos resultados de dezenas, ou até centenas de componentes sendo processadas simultaneamente, com valores de incremento e amplitude cuidadosamente calculados para gerar timbres elaborados.

Consideram-se, assim, alcançados os objetivos iniciais propostos. No entanto, avaliamos que o sistema atual ainda pode receber melhorias, tornando-se mais versátil, e capaz de executar operações mais complexas. Basicamente, a estrutura do oscilador multiplexado ainda deve ser atualizada antes de poder ser usada como o cerne de aplicações variadas de síntese de áudio.

8. Trabalhos Futuros

O próximo passo nesse trabalho é integrar o oscilador multiplexado em um sistema completo de síntese aditiva na placa DE2-70. Os componentes a serem desenvolvidos são, a princípio, geradores de envoltória tanto de amplitude quanto de freqüência, e um sistema de reescrita em tempo real que permita atualizar a Phase Angle Increment RAM e a Amplitude RAM com os valores das envoltórias associadas a cada componente. Além disso, deve ser implementada alguma forma de interface com o usuário.

Os próximos esforços serão voltados à implementação dos geradores de envoltórias de amplitude com reescrita da Amplitude RAM em tempo real. Esses avanços poderão ser usados para um funcionamento análogo com as envoltórias de freqüência e a Phase Angle Increment RAM. Quanto à interface, um desenvolvimento natural seria o controle do sistema via dispositivos MIDI. Além disso, o *framework* do oscilador multiplexado pode ser aplicado a outras técnicas de síntese além da aditiva.

Referências

Altera, DE2-70 Development and Education Board. Disponível em http://www.altera.com/education/univ/materials/boards/de2-70/unv-de2-70-board.html. Julho, 2013.

Altera, Quartus II Web Edition Software. Disponível em http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html. Julho, 2013.

Jansen, C., “Sine Circuitu: 10,000 high quality sine waves without detours”, Proceedings of the ICMC, 1991, pp. 222-225.

Lathi, B. P., “Signal processing and linear systems”, Berkeley-Cambridge Press, 1998.

Mealy, B. and Tappero, F., “Free Range VHDL”. Disponível em http://www.freerangefactory.org/dl/free_range_vhdl.pdf. Julho, 2013.

Moore, F. R., “Table lookup noise for sinusoidal digital oscillators”, Foundations of Computer Music, MIT Press, Cambridge, 1991, pp. 326-334.

Patterson, D. A. and Hennessy, J. L., Computer Organization and Design, 3rd ed., Elsevier, 2005, pp. 367-453.

Roads, C., “Computer Music Tutorial”, MIT Press, Cambridge, 1996.

Snell, J., “Design of a digital oscillator that will generate up to 256 low-distortion sine waves in real time”, Foundations of Computer Music, MIT Press, Cambridge, 1991, pp. 289-325.

Spiegel, M. R., “Schaum’s outline of Fourier analysis with applications to boundary value problems”, McGraw-Hill, 1974.