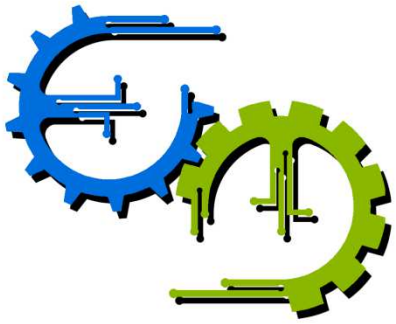
TRABALHO DE GRADUAÇÃO

OSCILADOR DIGITAL MULTIPLEXADO PARA SÍNTESE DE ÁUDIO EM DISPOSITIVO RECONFIGURÁVEL

Por,
Mateus Henriques Negrelli

Brasília, Dezembro de 2013



ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

OSCILADOR DIGITAL MULTIPLEXADO PARA SÍNTESE DE ÁUDIO EM DISPOSITIVO RECONFIGURÁVEL

POR,

Mateus Henriques Negrelli

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Márcio Pereira da Costa Brandão,
UnB/ CIC (Orientador) _____

Prof. Carlos Humberto Llanos Quintero,
UnB/ ENM _____

Prof. Jones Yudi Mori Alves da Silva,
UnB/ ENM _____

Brasília, Dezembro de 2013

FICHA CATALOGRÁFICA

NEGRELLI, MATEUS HENRIQUES
Oscilador digital multiplexado para síntese de áudio em dispositivo reconfigurável
[Distrito Federal] 2013.
xv, 36p., 210 x 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2013).
Trabalho de Graduação – Universidade de Brasília, Faculdade de Tecnologia.

1. Oscilador	2. FPGA
3. Síntese de Áudio	4. Pipeline
I. Mecatrônica/FT/UnB	II. Oscilador Digital Multiplexado para Síntese de Áudio em Dispositivo Reconfigurável

REFERÊNCIA BIBLIOGRÁFICA

NEGRELLI, M. H. (2013). Oscilador digital multiplexado para síntese de áudio em dispositivo reconfigurável. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 13, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 51p.

CESSÃO DE DIREITOS

AUTOR: Mateus Henriques Negrelli

TÍTULO DO TRABALHO DE GRADUAÇÃO: Oscilador digital multiplexado para síntese de áudio em dispositivo reconfigurável

GRAU: Engenheiro de Controle e Automação ANO: 2013

Mateus Henriques Negrelli
SMPW Qd 16 Cj 6 Lt 4 Cs H – Park Way.
71741-606 Brasília – DF – Brasil.

DEDICATÓRIA

Para o Spyke.

Mateus Henriques Negrelli

AGRADECIMENTOS

Com a entrega desse relatório, o último de vários, encerram-se longos cinco anos que passei cursando Engenharia Mecatrônica na Universidade de Brasília. Após deixar a versão final desse trabalho no ENE, estarei livre.

Ao longo desse último semestre, me perguntaram bastante a mesma coisa: o que pretendo fazer após a formatura. Essencialmente, o que farei com a tal liberdade. A todos, respondi com incerteza e dúvida, com “não sei”. Em troca, recebi sugestões: fazer um mestrado, conseguir um emprego, sair do país, fazer os três ao mesmo tempo, até mesmo começar outro curso do zero. Agora, a uma semana de defender meu trabalho de graduação, um momento que sempre pareceu inalcançável, é estranho perceber que nada mudou. Ainda não sei. Assim como fazer 18 anos, atingir a maioridade acadêmica não te torna subitamente alguém diferente. De repente você recebe vários direitos, várias obrigações, e ainda é a mesma pessoa que ontem tinha 17 anos.

Ninguém te conta isso, mas adultos de verdade são uma coisa rara de se ver. Quando você está na quarta série, os alunos da quinta são de outro nível. Quando você está na oitava série, o pessoal do primeiro ano mora em outro planeta. Ao longo do ensino médio, vive-se uma corrida de obstáculos para entrar na universidade, onde tudo vai ser diferente. Agora, prestes a receber um diploma na UnB, volta esse sentimento tão familiar, de que as pessoas na próxima camada é que sabem o que estão fazendo. Na verdade, tem gente que morre sem saber o que está fazendo, sem encontrar uma entrada para a crisálida mágica que te transforma em adulto.

Alguns dizem que a universidade é essa pupa, que com um diploma você faz suas asas, e pode finalmente voar. Imagino que entrei na fila errada em algum momento. Meia década depois, e tudo que observei foram alunos de engenharia se esforçando intensamente para passar nas matérias, para entregar os relatórios, para terminar os trabalhos. Para agradar os professores, para garantir as menções, para não sair do fluxo. Ensino Fundamental evolui para Ensino Médio no nível 15, Ensino Médio evolui para Ensino Superior no nível 18: a essência das tarefas realizadas é a mesma, muda apenas o grau de dificuldade.

Há um paradoxo aqui: em tantas aulas dadas por engenheiros que não são professores, somos ensinados como estudantes que não são engenheiros. É inevitável se sentir um idiota, às vezes, quando se é atingido por uma sensação de perda de tempo, quando o foco

de tantas horas-aula parece estar em um pedaço de papel que te chama de algo que ninguém te falou como ser: engenheiro. Adulto?

No começo do curso, parece que tentam. Te jogam nesse novo mundo, a universidade, praticamente sem orientação. Certamente, para finalmente nos dar a independência que só vem com a superação de desafios sem alguém segurando sua mão. Em teoria, lindo. Contudo, parece que ao longo do caminho o significado disso se perdeu, e agora o curso é uma luta onde professores abandonam o papel de mentor, e ativamente antagonizam seus alunos. Repito: há professores que são vistos literalmente como inimigos. Como obstáculos colocados ali, empecilhos que nada adicionam, apenas subtraem. Isso, pelo menos para mim, para um estudante, parece simplesmente errado. A universidade deveria ser um ambiente de cooperação, de troca de conhecimentos, de progresso. Aqueles que já se depararam com os problemas atuais deveriam avisar a nova geração, orientá-la, de maneira que o tempo da nova massa estudantil pudesse ser gasto solucionando problemas inéditos. Infelizmente, o que se vê são jovens obrigados a bater com a cabeça nas mesmas paredes, ano após ano, e ao fim de seus cinco anos tornando-se criaturas ultrapassadas. Certamente, um reflexo dos que ensinam sobre os ensinados.

Mesmo para terminar o curso, mantém-se a problemática falta de orientação. Entre muitos regulamentos, freqüentemente pouco razoáveis e citados sem qualquer consistência, não há uma cartilha do formando. Uma compilação das informações necessárias para evitar surpresas desagradáveis durante os processos de inscrição, realização e finalização do trabalho de graduação, do estágio curricular, até mesmo da entrada na lista de prováveis formandos. Fica a cargo do aluno sentir que falta algo na burocracia, por vezes via um colega que já passou por um desgosto desnecessário, e correr atrás de formulários que estavam silenciosamente à sua espera.

Vem, então, o mercado de trabalho, onde engenheiros recém-formados, inseguros e sub-capacitados tentam encontrar uma vaga em meio a empresas que querem contratar apenas profissionais que já conhecem as sutilezas da ocupação, a velha guarda. O destino encontrado por muitos engenheiros da nova geração, pior do que a clássica piada do estagiário, é a mais risível ainda posição de trainee: o jeitinho dado pelo mercado para recrutar os novos profissionais sem pagar-lhes aquilo que, alguém disse, eles merecem.

Por toda a jornada, começo, meio e fim, o panorama é ocasionalmente conturbado, invariavelmente sombrio. O valor dado pela sociedade a um diploma universitário parece se restringir àqueles que olham de fora da arena, que não têm poder para compensar um

estudante por gastar praticamente 20% de sua vida até agora em um ambiente abrasivo, onde surgem menos conclusões do que perguntas, cujo final é um tímido “não sei”.

Já me perguntaram se eu, tendo a chance de voltar no tempo e escolher, faria tudo de novo. Talvez em outra instituição, mas digo isso assombrado pela perspectiva de que a situação que conheci na Universidade de Brasília seja a norma, não uma exceção. Não sei.

Uma coisa eu sei, no entanto: nesse período de desilusão e desapontamento, encontrei alguns indivíduos que se destacaram, que me fizeram acreditar que ainda há exemplos dignos de serem seguidos no trágico quadro pintado até aqui. Serão eles, então, os primeiros a receber uma homenagem nessa seção de agradecimentos.

Agradeço a Celius Antônio Magalhães, por ter sido mais do que um professor, mas um verdadeiro educador, um homem que merece respeito sem fim. Agradeço pelo exemplo impecável, que certamente levarei como influência para toda a vida, e pela chance que me foi dada de ser seu monitor durante três semestres. Se alguém um dia elogiar a maneira como ensino, considero obra direta do Celius.

Agradeço a Márcio da Costa Pereira Brandão, por toda a orientação durante esse trabalho de graduação. Por pura sorte, consegui a chance de desenvolver um projeto numa área do conhecimento que genuinamente me interessa, algo que nem todos podem dizer sobre sua vida universitária. Agradeço por toda a animação, por todo o incentivo, por você acreditar que valia a pena até apresentar esse trabalho em simpósios e conferências. Sua grande motivação certamente foi contagiante.

Agradeço a Flávio de Barros Vidal, por toda a ajuda que me deu. Por trás do professor famosamente carrasco de OAC, há na verdade alguém sempre disposto a fornecer auxílio, indo até além do esperado. Foi graças a ele que me interessei por design de circuitos no Quartus (perdendo tantas horas de sono no Projeto 3), e foi graças a ele que consegui um estágio numa empresa onde posso trabalhar com desenvolvimento de projetos, e não com burocracia e uma máquina de café.

Aos três professores acima, agradeço por terem se importado e acreditado.

Agradeço também a Flávia Maria Guerra de Sousa Aranha Oliveira, João Yoshiyuki Ishihara, e Guilherme Caribé Carvalho, por me darem motivos para dizer que as matérias de controle e automação tiveram altos entre os baixos.

É claro que não só professores serão homenageados aqui. Existem aqueles que, mesmo não dando aula, participaram de toda essa odisséia.

Agradeço a Raiza Martins Dias, por toda a calma e estabilidade que me trouxe. Sem ela, tudo seria mais difícil.

Agradeço a minha família, por terem me apoiado mesmo sem acreditar muito no caminho que escolhi. Se um dia conheci alguém que conseguiu virar adulto, foi meu pai.

Agradeço a meus cachorros.

Agradeço ao pessoal do Chat das Galeras, pelas infindáveis horas de conversa e companheirismo, tanto online quanto IRL.

Agradeço aos gringos do Skype, que provavelmente não vão ler isso (e mesmo que leiam, só um sabe falar português). Apesar da distância, sua amizade é inegável.

Agradeço a amigos e amigas do Ensino Médio e do Ensino Fundamental que continuaram na minha vida ao longo do tempo, e também a alguns cuja presença se distanciou.

Agradeço à turma XXIV da Mecatrônica.

Agradeço ao pessoal do Cocobo, por tantas horas desperdiçadas no Amarelo.

Mateus Henriques Negrelli

RESUMO

Este trabalho descreve uma implementação orientada a FPGA's de um oscilador digital multiplexado. O sistema realiza, com apenas um dispositivo, o trabalho de centenas de dispositivos em paralelo. Inicialmente, apresenta-se um modelo teórico do oscilador, e em seguida a implementação em *software* de todos os componentes descritos. Posteriormente, há a adição de alguns blocos para lidar com particularidades trazidas pelo domínio prático. A estrutura geral é organizada em um *pipeline* com as tarefas necessárias divididas em três estágios. O objetivo final do projeto é fornecer um *framework* para síntese de áudio em FPGA's.

Palavras-chave: oscilador, FPGA, síntese de áudio, *pipeline*.

ABSTRACT

This work describes an FPGA-oriented implementation of a multiplexed digital oscillator. The system performs, with a single device, the work of hundreds of devices in parallel. Initially, a theoretical model of the oscillator is shown, and then the software implementation of all the described components. Afterwards, there's the addition of some blocks to deal with particularities brought by the practical domain. The general structure is organized in a pipeline with the necessary tasks divided in three stages. This project's ultimate goal is to provide an audio synthesis framework for FPGA's.

Keywords: oscillator, FPGA, audio synthesis, pipeline.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

f_K	Entrada de freqüência do gerador k na síntese aditiva	[Hz]
a_K	Entrada de amplitude do gerador k na síntese aditiva	[V]
f_S	Taxa de amostragem	[Hz]
T	Período de amostragem	[s]
L	Comprimento da tabela de seno	[amostras]
f_0	Freqüência da senóide	[Hz]
I	Amostras percorridas por ciclo na tabela	[amostras]
b	Bits do barramento completo do oscilador simples	[bits]
k	Bits mais significativos do barramento completo do oscilador simples	[bits]
f_{NS}	Freqüência do ciclo de componente	[Hz]
n	Número de componentes no oscilador multiplexado	[componentes]
m	Número de bits nas palavras que representam as amostras de áudio	[bits]

Siglas

FPGA	*Field-programmable Gate Array*
DC	*Direct Current*
DAC	*Digital-to-Analog Converter*
RAM	*Random Access Memory*
E/S	Entrada/Saída
PLL	*Phase-Locked Loop*
VHDL	VHSIC *Hardware Design Language*
VHSIC	*Very-High-Speed Integrated Circuits*
ROM	*Read-Only Memory*
MSB	*Most Significant Bits*
LSB	*Least Significant Bits*
LRC	*Left Right Clock*
BCLK	*Bit Clock*
DAT	*Data Channel*
MIDI	*Musical Instrument Digital Interface*
FM	*Frequency Modulation*

CAPÍTULO 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 [1]. Alia-se a isso o fato demonstrado por Fourier [2, 3] 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 [4]. 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 [5] 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.

CAPÍTULO 2 – CONSIDERAÇÕES TEÓRICAS

2.1 SÍNTESE DE ÁUDIO

Som é uma onda mecânica, uma oscilação de pressão transmitida através de um meio físico, composta de freqüências dentro da faixa de audição. Como mencionado no Capítulo 1, pode-se decompor tal oscilação em componentes senoidais simples, chamadas harmônicos. Em outras palavras, qualquer forma de onda pode ser descrita em termos da soma de seus harmônicos, cada um com suas respectivas freqüência e amplitude. Na Figura 2.1, pode-se ver como um harmônico, ou tom puro, varia em relação ao tempo. Já na Figura 2.2, vê-se uma forma de onda mais complexa, composta pela soma de três tons puros com freqüências e amplitudes diferentes.

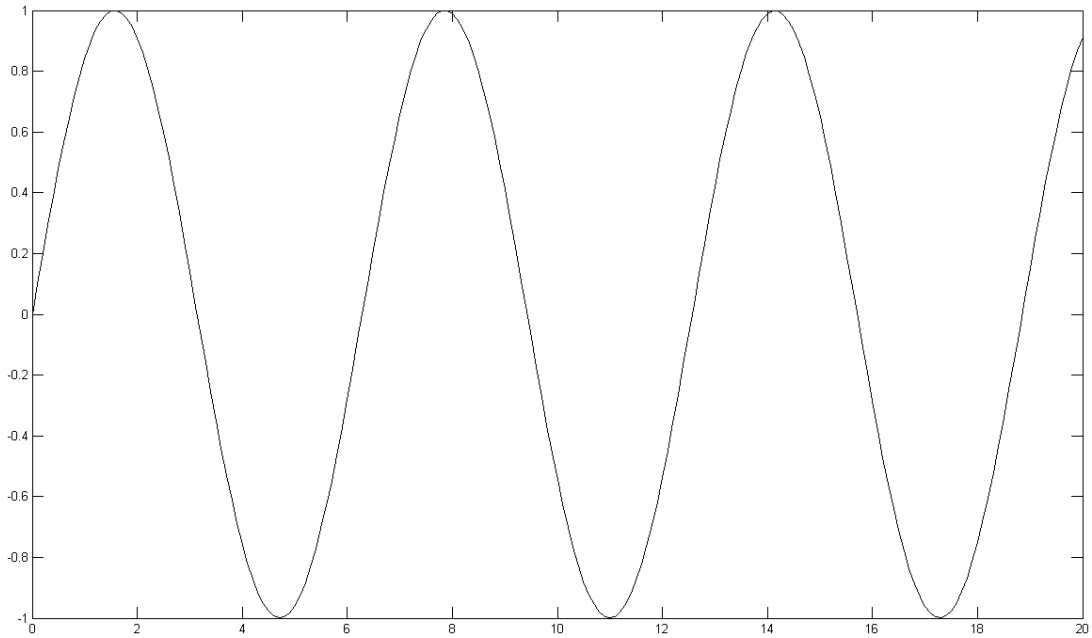


Figura 2.1. Um harmônico, dado por sen(x).

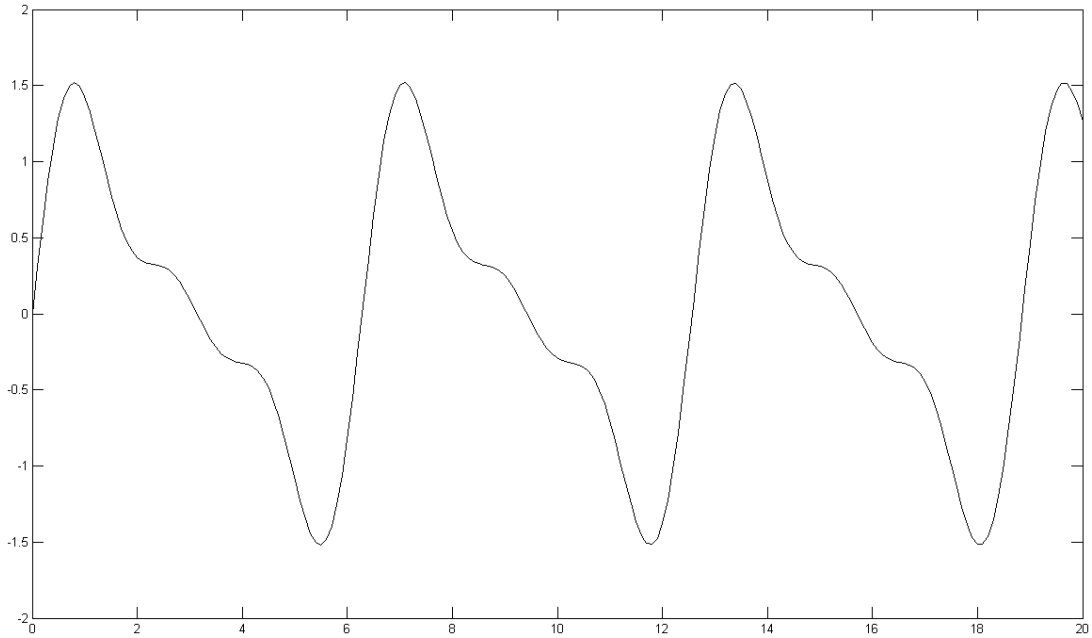


Figura 2.2. Três harmônicos, dados por sen(x) + 0.6sen(2x) + 0.3sen(3x).

O timbre de um som é definido pela relação entre as amplitudes e freqüências dos harmônicos combinados, e a síntese de áudio fornece meios de construir timbres, podendo-se emular instrumentos existentes ou criar sons completamente novos. Existem diversas técnicas de síntese de áudio, porém o foco desse trabalho será aquela conhecida como síntese aditiva.

2.1.1 SÍNTESE ADITIVA

Considerando-se instrumentos musicais sob a luz da teoria de Fourier, seus timbres consistem de múltiplas parciais harmônicas ou não-harmônicas, sendo cada parcial uma senóide com freqüência e amplitude dotadas de envoltórias temporais específicas. Valendo-se disso, a síntese aditiva gera som através da soma das saídas de múltiplos geradores de ondas senoidais.

Na Figura 2.3, observa-se um diagrama representando uma estrutura genérica de síntese aditiva: geradores de onda em paralelo, conectados a um somador cujo resultado é a saída do sistema. Cada gerador está associado a uma parcial do timbre, recebendo como entrada uma freqüência f_K e uma amplitude a_K.

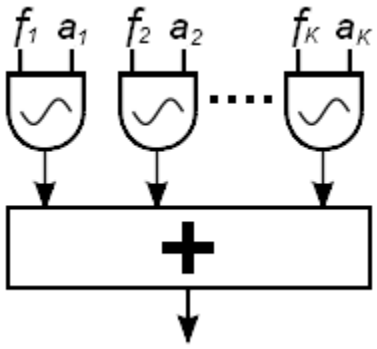


Figura 2.3. Estrutura genérica de síntese aditiva.

Em geral, uma série de Fourier, utilizada para a representação de sinais periódicos, contém um número infinito de componentes senoidais, sem limite superior para as freqüências e incluindo uma componente DC. No caso da síntese aditiva, as freqüências fora da faixa de audição humana podem ser omitidas, ou seja, os valores possíveis para f_K se restringem a uma faixa finita, limitando também o número de osciladores necessários para construir um timbre.

2.2 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), de amostras de um ciclo de uma senóide. Para armazenar o ciclo de onda necessário, usa-se normalmente uma tabela circular de comprimento L (Fig 2.4). A freqüência f_0 da senóide gerada é a razão entre f_S e o número de amostras percorridas por ciclo. Esse número é dado por L/I, onde I é o chamado incremento de ângulo de fase. Segue-se que, para incremento unitário, f_0 é expressa como na Fig 2.4.

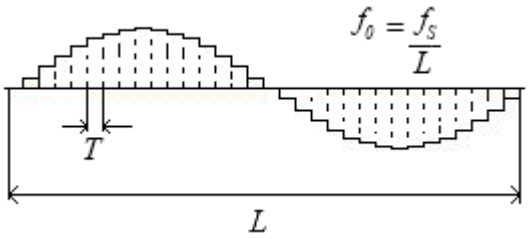


Figura 2.4. Tabela de *L* amostras representando uma onda senoidal.

Por exemplo, caso o incremento de indexação da tabela seja dobrado, o novo sinal terá metade das amostras em um ciclo, correspondendo ao dobro da freqüência. O resultado equivale basicamente ao oscilador realizando a leitura de uma tabela com metade das amostras (Fig 2.5).

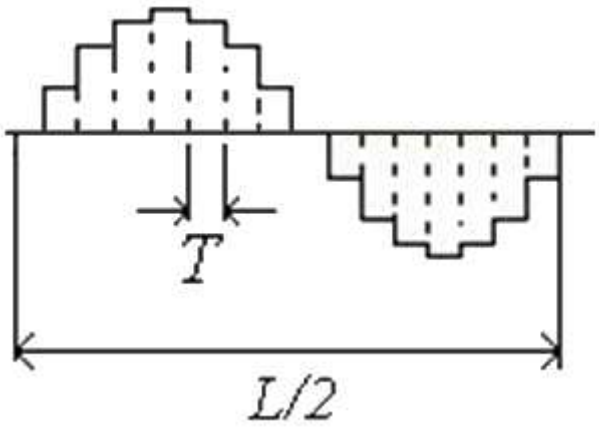


Figura 2.5. Tabela de *L*/2 amostras representando uma onda senoidal.

A Equação 1 apresenta 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.

$$I = \frac{f_0 . L}{f_S} \qquad (1)$$

A Figura 2.6 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, combinando as saídas de ambos os registradores, completa um acumulador de ângulo de fase;
- Um DAC que conecta o oscilador ao resto do circuito.

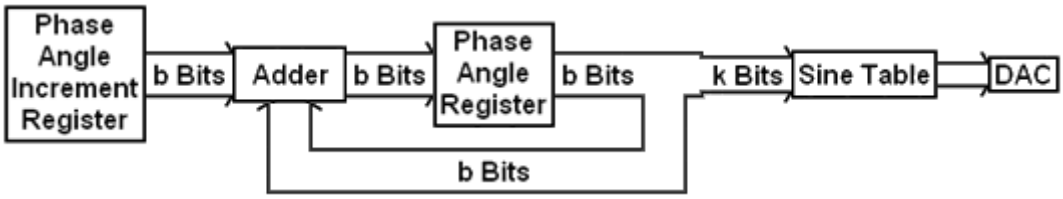


Figura 2.6. Arquitetura básica do oscilador simples.

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.

A escolha do valor de *b* está relacionada com os limites da capacidade humana de perceber passos de freqüência na saída audível do oscilador. Mostrou-se, através de estudos em psico-acústica, que para garantir *glissandos* e *vibratos* suaves, 20 é a largura de palavra mínima dos registradores no oscilador simples [5, 6].

A escolha de um comprimento adequado para a tabela de seno, por outro lado, visa obter uma relação sinal-ruído adequada para os sinais de áudio, conforme descrito em Moore [7].

2.3 PIPELINE

Na arquitetura de computadores, a técnica de *pipeline* é fundamental para tornar processadores mais rápidos, e sua essência se baseia em aumentar o *throughput* do sistema através do funcionamento em paralelo de suas partes, geralmente denominadas estágios [8].

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, mas o tempo de processamento do programa diminui. O aumento de velocidade é gerado pela paralelização das etapas, e está associado ao aumento de *throughput* mencionado acima. Nota-se que, para um programa com apenas uma instrução, a técnica de *pipeline* não oferece vantagem. Por outro lado, caso haja uma quantidade suficientemente grande de instruções, a aceleração fornecida pelo *pipelining* pode ser potencialmente igual ao número de estágios do sistema.

Por exemplo, na Figura 2.7 vê-se a execução de três instruções em um sistema sem *pipelining* implementado. Já na Figura 2.8, o sistema possui *pipelining*. Dessa maneira, um programa que normalmente levaria 9 ciclos de relógio para ser executado passa a levar apenas 5. Assumindo-se uma duração semelhante para os estágios, o processador com *pipeline* poderia ser três vezes mais rápido que o padrão caso o número de instruções tendesse a infinito. Os tempos de inicialização e finalização do processo impedem que isso ocorra na prática: enquanto o *pipeline* não está completamente cheio, a melhoria não é totalmente aproveitada. No caso da Figura 2.8, apenas o ciclo 3 representa o *pipeline* em funcionamento pleno.

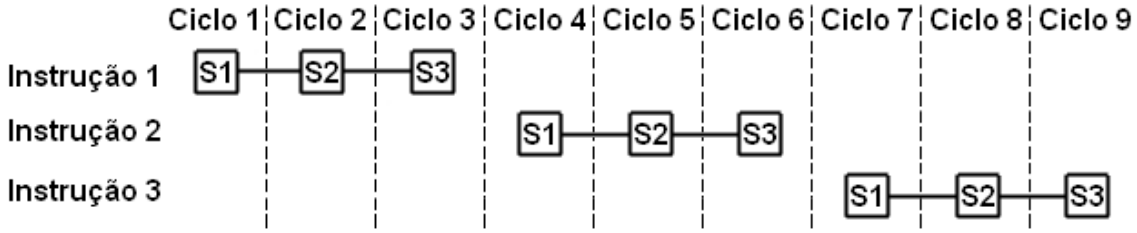


Figura 2.7. Execução de instruções em um sistema sem *pipeline*.

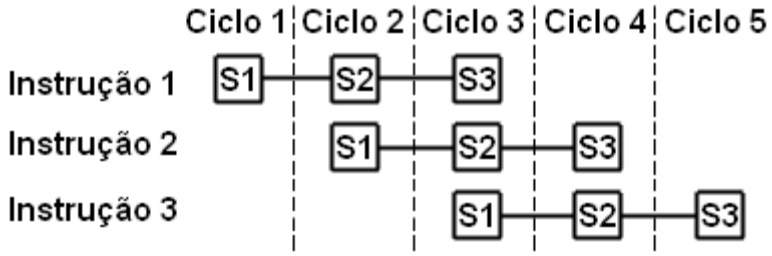


Figura 2.8. Execução de instruções em um sistema com *pipeline*.

Vale notar que nem todo sistema tem seus estágios perfeitamente balanceados. Em caso de desbalanceamento, os ciclos de relógio serão limitados pelo tempo de execução dos estágios mais lentos. Isso é outro fator que prejudica a realização da aceleração máxima que o *pipeline* pode oferecer em teoria.

É necessário fazer aqui uma ressalva: na implementação do oscilador multiplexado, não são processadas instruções propriamente ditas. Mesmo assim, chama-se de *pipeline* a estrutura desenvolvida, por causa de sua separação em estágios e da divisão em unidades (as componentes senoidais) do fluxo de dados.

2.4 FPGA

Um FPGA (*Field-programmable Gate Array*) é um circuito integrado cujo diferencial é a ausência de uma programação fixa implementada pelo fabricante. Como indicado em seu próprio nome, FPGA's podem ser configurados pelo usuário para executar aplicações específicas e customizadas, geralmente escritas em uma linguagem de descrição de *hardware*. Essas aplicações podem ser implementadas temporariamente, com uma posterior reconfiguração do dispositivo alterando completamente sua função. Observa-se na Figura 2.9 a estrutura geral de um FPGA, composta por três tipos de componentes básicos: blocos lógicos, interfaces de entrada e saída, e uma matriz de interconexões.

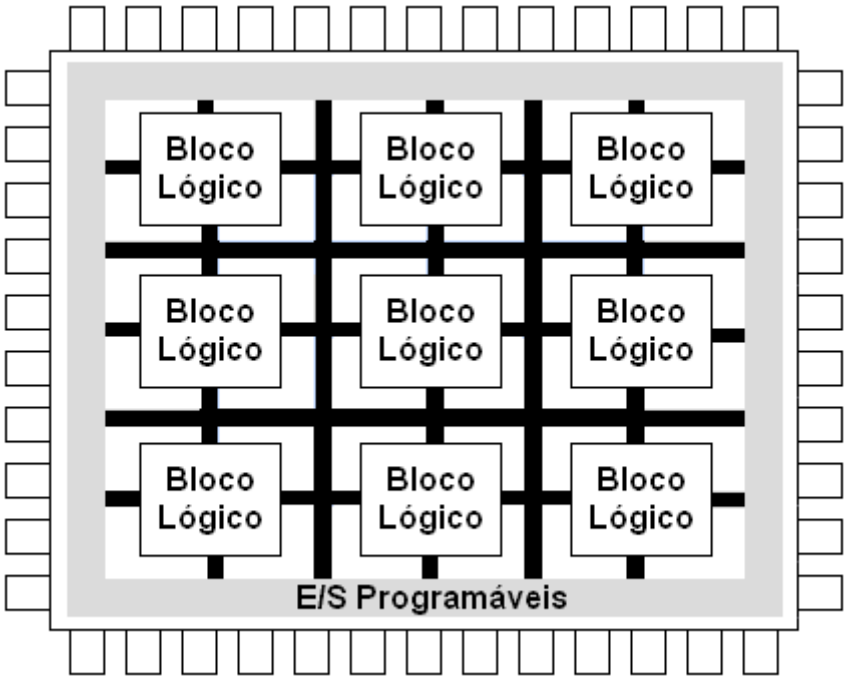


Figura 2.9. Estrutura geral de um FPGA.

Os blocos lógicos podem ser configurados para implementar funções combinacionais complexas, ou simples portas lógicas. Vale notar que, na maioria dos FPGA's, os blocos lógicos incluem elementos de memória, indo de *flip-flops* básicos a RAM's completas. As interfaces de entrada e saída fornecem meios de comunicação entre o dispositivo e o meio externo, e placas FPGA geralmente possuem grande variedade de E/S disponíveis. Por fim, a matriz de interconexões permite o roteamento de dados ao longo dos blocos lógicos, em várias configurações diferentes, e a conexão dessas estruturas com as interfaces de E/S.

Comparados com circuitos integrados dedicados, os FPGA's possuem capacidade de processamento restrita. Por outro lado, sua vantagem é a grande versatilidade associada à reprogramação, tornando os FPGA's ideais para prototipagem e teste de circuitos.

CAPÍTULO 3 – OSCILADOR MULTIPLEXADO

A Figura 3.1 apresenta os três estágios da arquitetura do oscilador multiplexado em FPGA, baseada na implementação em *hardware* proposta por Snell [5].

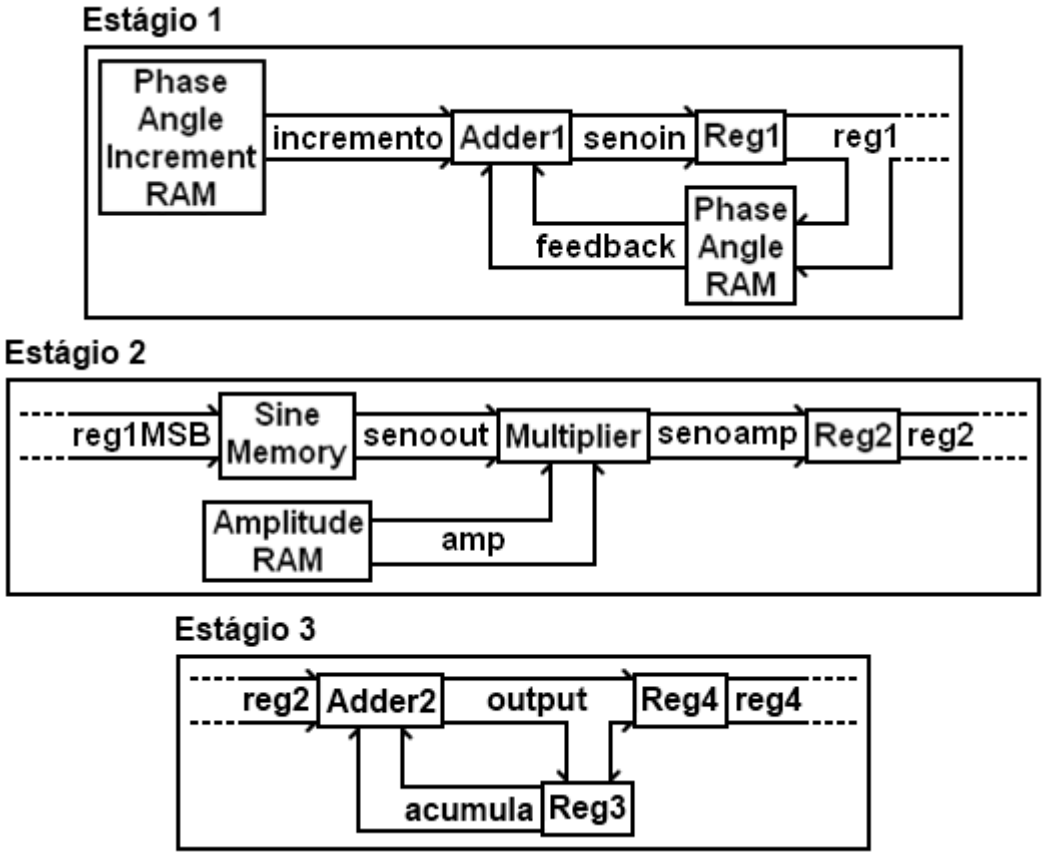


Figura 3.1. Arquitetura básica do oscilador multiplexado.

Como mencionado no Capítulo 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.

O circuito é controlado por dois *clocks* principais (ver Seção 5.6), 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 de f_S pelo número n de componentes (Fig. 3.2). Todos os blocos do sistema utilizam ClkComp em sua entrada, exceto Reg4, que por estar associado ao processo de amostragem utiliza ClkSamp.

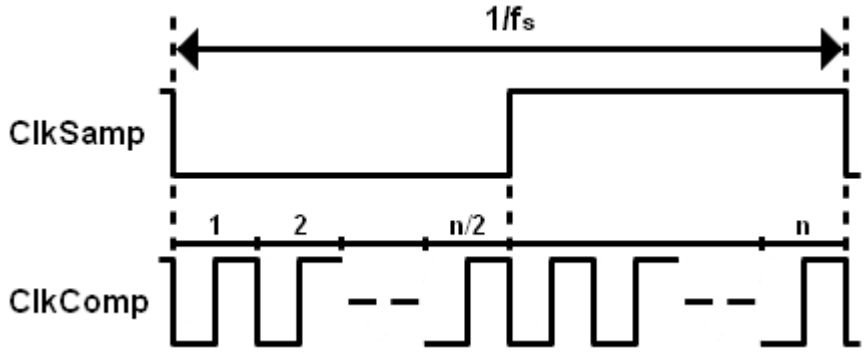


Figura 3.2. *Clocks* de controle do oscilador multiplexado.

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, como endereços para acessar um ciclo de senóide em memória, os bits mais significativos dos valores de incremento calculados. 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.

3.1 MEMÓRIAS

A principal diferença entre o oscilador simples e o multiplexado é a presença, neste, de três memórias a mais. Num único ciclo de amostragem, 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 3.4).

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.

3.2 ENDEREÇAMENTO

Um circuito de endereçamento (oculto na Fig. 3.1) faz com que as três memórias passem seqüencialmente 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.

3.3 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. Reg3 acumula os valores finais de cada componente, gerando-se ali o valor final da amostra em si para cada período de amostragem.

3.4 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*.

CAPÍTULO 4 – CONSIDERAÇÕES PRÁTICAS

4.1 PLACA ALTERA DE2-70

A placa DE2-70 é um FPGA produzido pela marca Altera, e pertence à família Cyclone II de dispositivos [9]. É uma versão com mais memória e unidades lógicas da placa DE2, e é direcionada ao uso laboratorial em colégios e universidades, para aprendizado de lógica digital e organização de computadores. Abaixo, uma visão geral da DE2-70, onde se podem observar seus vários componentes. Em seguida, uma tabela com alguns dados básicos sobre a placa.

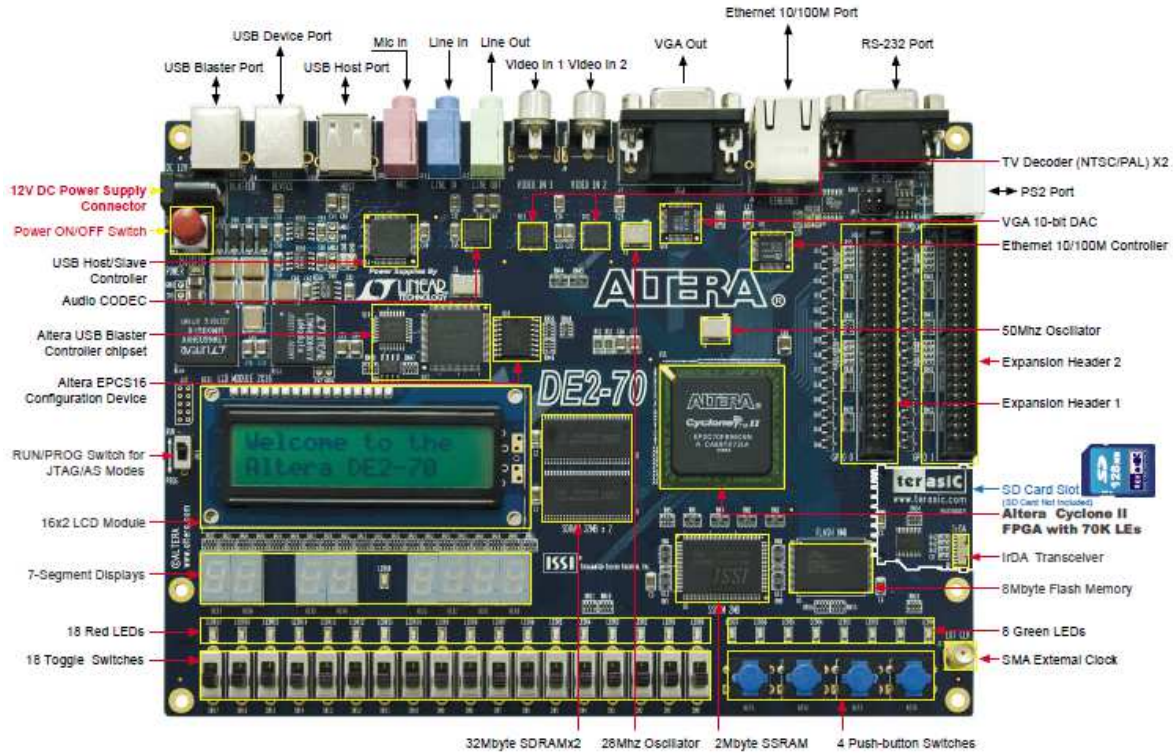


Figura 4.1. Visão geral da placa Altera DE2-70.

Tabela 4.1. Dados básicos sobre a placa Altera DE2-70.

Dado	Quantidade
Elementos lógicos	68416
RAM's M4K	250
Bits totais de RAM	1152000
Multiplicadores dedicados	150
PLL's	4
Pinos de E/S para usuário	622

Para a realização desse trabalho, não são necessários todos os recursos disponíveis no FPGA. Pelo escopo da aplicação, o foco será direcionado ao codec de áudio WM8731 e à saída de linha de áudio. Além disso, é válido mencionar os *clocks* internos, usados para controlar o sistema, e os *toggle switches*, que possibilitam a intervenção do usuário em testes.

4.2 PROGRAMA QUARTUS II

Também produzido pela Altera, o Quartus II é uma ferramenta em *software* para análise e síntese de *designs* HDL (ver Seção 4.3) [10]. Ele permite que desenvolvedores compilem seus projetos, realizem análises temporais, simulem a resposta da estrutura a diferentes entradas, e configurem o dispositivo físico alvo com o programa previamente analisado.

A versão Web Edition do Quartus II pode ser baixada de graça na internet, e fornece recursos de programação e compilação para um número limitado de dispositivos Altera. A família Cyclone de FPGA's de baixo custo recebe suporte completo nesta edição do Quartus, permitindo que desenvolvedores pequenos e instituições de educação possam evitar lidar com os custos de um *software* de desenvolvimento.

Abaixo, mostram-se algumas figuras retratando a interface do Quartus II em sua versão 9.1, com alguns focos específicos:

- Tela inicial;
- Construção de diagramas de blocos;
- Programação VHDL;
- Análise temporal por vetores de forma de onda;
- *Pin planner* (para associar as entradas e saídas do programa ao dispositivo físico);
- *Programmer* (para efetivamente implementar o programa no dispositivo físico).

Usa-se especificamente a versão 9.1 do programa, desatualizada, porque versões grátis posteriores não contam com a simulação via *vector waveforms* (Fig. 4.5), considerada essencial para verificar o funcionamento do oscilador multiplexado a cada ciclo de componente e, mais ainda, para encontrar eventuais erros durante o desenvolvimento.

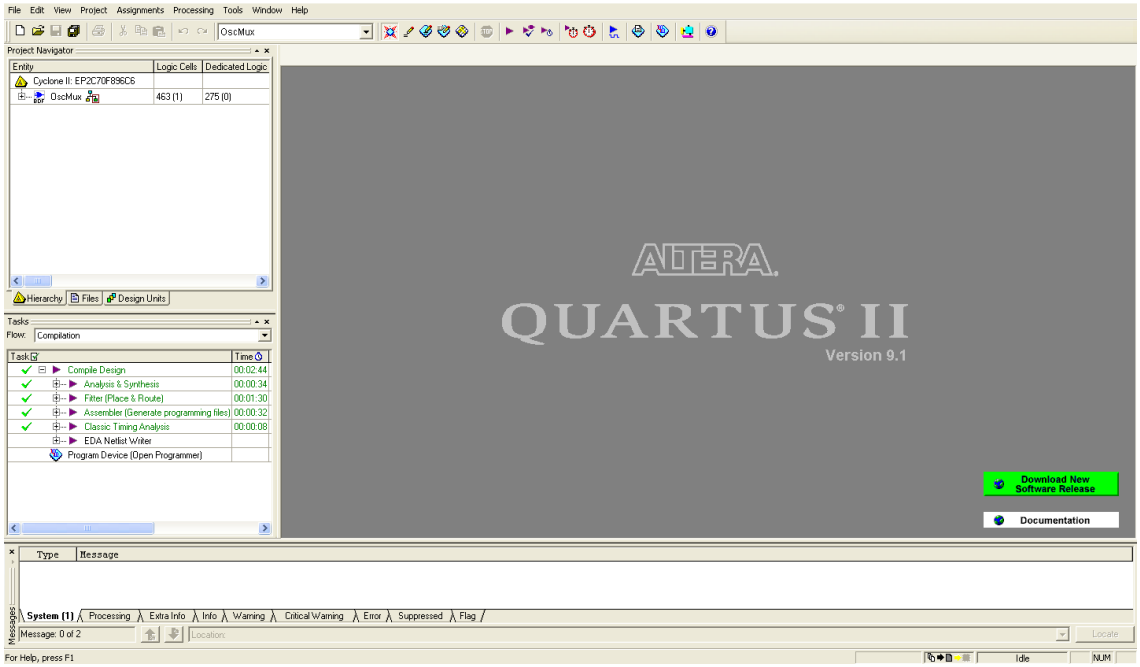


Figura 4.2. Tela inicial do Quartus II.

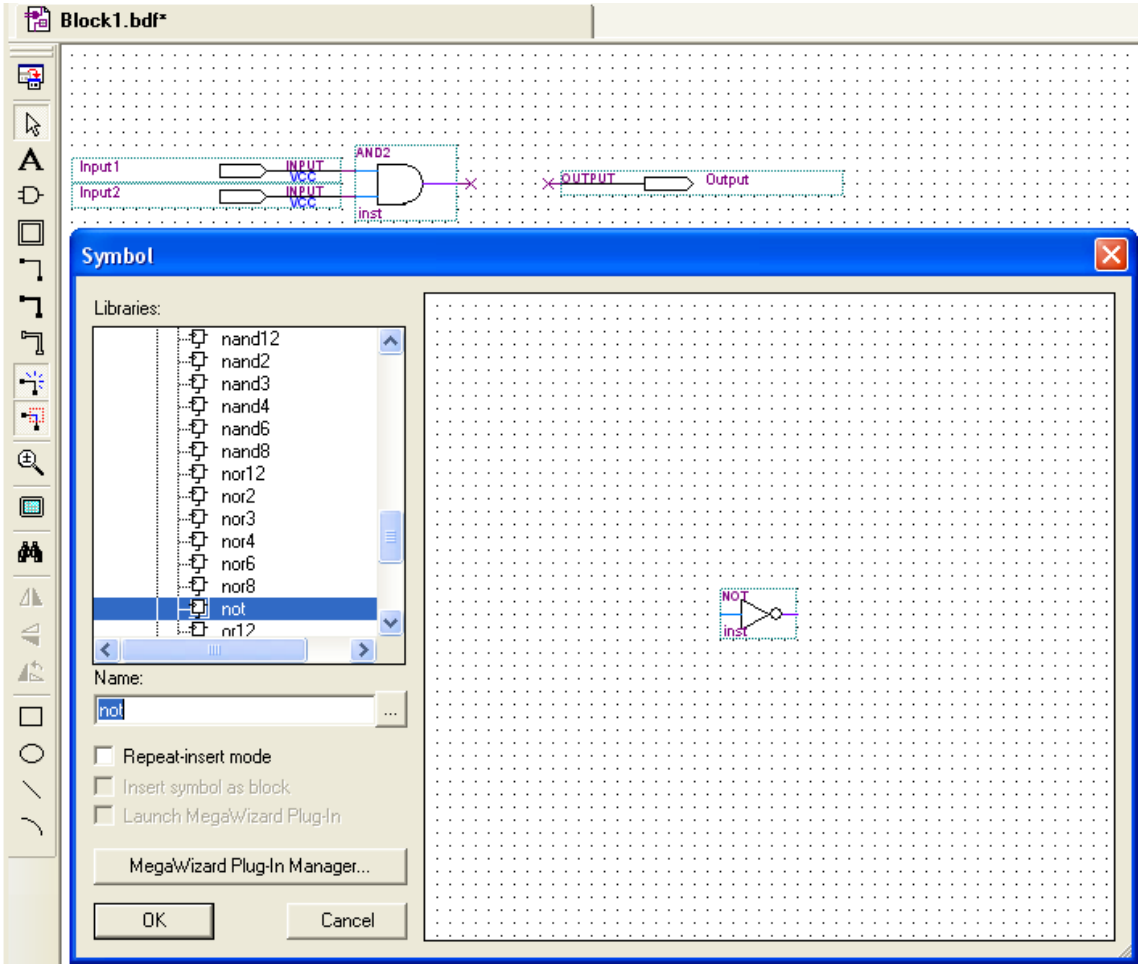


Figura 4.3. Construção de diagramas de blocos no Quartus II.

Vhdl1.vhd*

```
1   library IEEE;
2       use IEEE.std_logic_1164.all;
3       use IEEE.numeric_std.all;
4
5   entity fig is
6       port
7       (
8           Input1, Input2 : in std_logic;
9           Output : out std_logic
10      );
11  end fig;
12
13  architecture fig_rtl of fig is
14
15  signal tmp_signal : std_logic;
16
17  begin
18
19      tmp_signal <= Input1 and Input2;
20      Output <= not(tmp_signal);
21
22  end fig_rtl;
```

Figura 4.4. Programação VHDL no Quartus II.

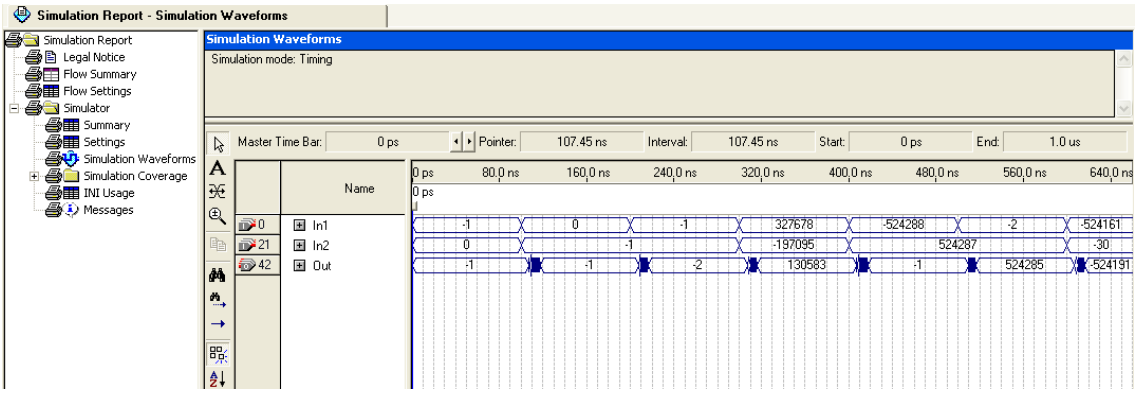


Figura 4.5. Análise temporal por vetores de forma de onda no Quartus II.

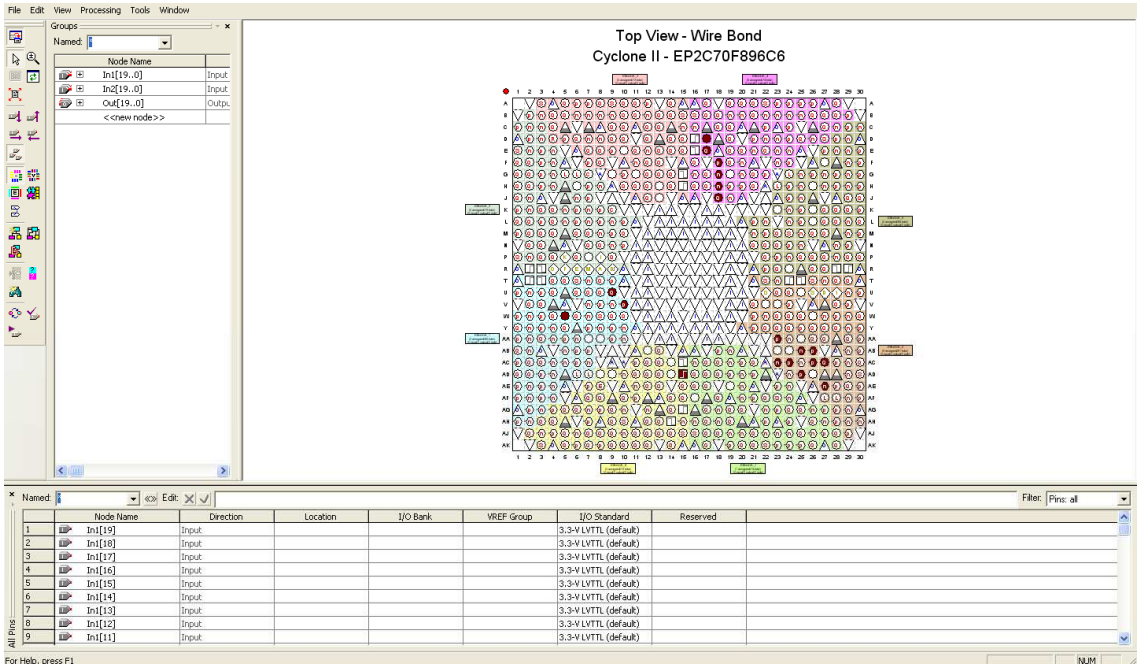


Figura 4.6. *Pin planner* no Quartus II.

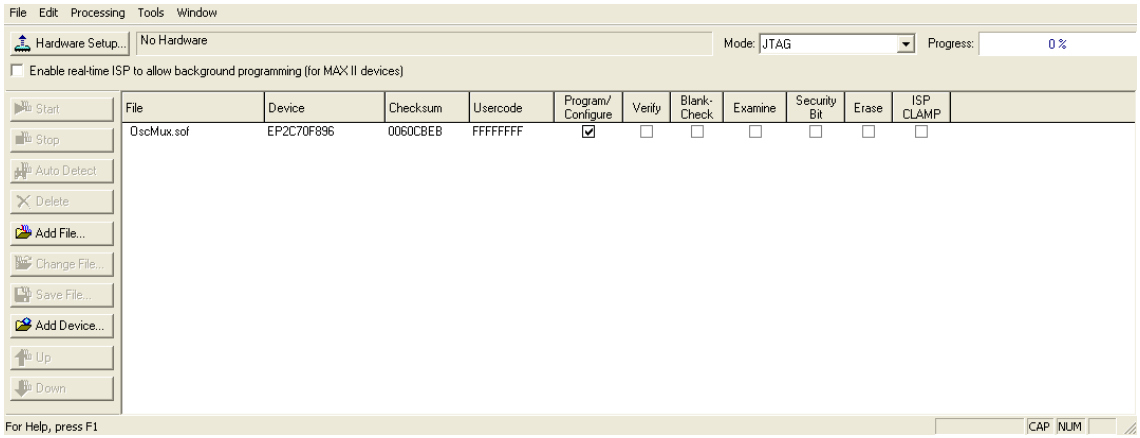


Figura 4.7. *Programmer* no Quartus II.

4.3 PROGRAMAÇÃO VHDL

VHDL é uma sigla dentro de uma sigla: o V representa VHSIC, *Very High-Speed Integrated Circuit*, e o HDL quer dizer *Hardware Description Language*. Como indicado em seu nome, VHDL é uma linguagem de programação, completa com sintaxe e regras de uso. Contudo, ao contrário de linguagens computacionais de alto nível, VHDL é usada primariamente para descrever *hardware* [11].

Efetivamente, isso quer dizer que VHDL não é seqüencial, como C ou Java, mas trabalha com concorrência: instruções VHDL são executadas todas ao mesmo tempo, independente do tamanho da implementação. Essa diferença de paradigma implica uma diferença no processo de codificação VHDL, onde tentativas de escrever código em estilo alto nível costumam gerar resultados ineficientes, ou simplesmente não funcionais. Em outras palavras, não se deve pensar em VHDL como programação propriamente dita, mas como *design* de *hardware*, onde as linhas de código explicitam as funcionalidades de blocos lógicos.

Na Figura 4.8 observa-se um circuito simples, que executa operações paralelas, ou concorrentes. Mudanças nas entradas de qualquer uma das portas lógicas podem ocasionar

mudanças na saída, e todas as entradas devem ter seus valores considerados simultaneamente. Logo abaixo, na Figura 4.9, apresenta-se um exemplo de código VHDL que equivale a esse circuito.

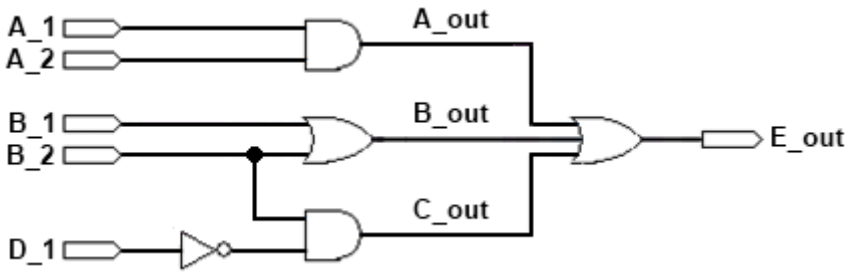


Figura 4.8. Circuito exemplo para programação VHDL.

```
-- declaração de bibliotecas
library IEEE;
use IEEE.std_logic_1164.all;

-- entidade
entity circ_exemplo is
    port
    (
        A_1, A_2, B_1, B_2, D_1 : in std_logic;
        E_out : out std_logic
    );
end circ_exemplo;

-- arquitetura
architecture arq_circ_exemplo of circ_exemplo is

signal A_out, B_out, C_out : std_logic;

begin

    A_out <= A_1 and A_2;
    B_out <= B_1 or B_2;
    C_out <= (not D_1) and B_2;
    E_out <= A_out or B_out or C_out;

end arq_circ_exemplo;
```

Figura 4.9. Código VHDL equivalente ao circuito exemplo.

Há dois propósitos principais para linguagens de descrição de *hardware*, das quais VHDL é apenas uma: a modelagem de circuitos e sistemas digitais, e sua subseqüente simulação e teste. Além disso, modelos HDL podem ser traduzidos para um formato que pode ser usado para gerar circuitos digitais reais, num processo conhecido como síntese.

Outra opção disponível para modelar o comportamento de circuitos digitais são linguagens baseadas em representações gráficas, mais fáceis de usar por causa de sua curva de aprendizado confortável. O próprio Quartus II disponibiliza diagramas de blocos para facilitar a implementação de circuitos. No entanto, esse tipo de metodologia requer uma parcela significativa de tempo dedicada a “construir” o modelo: posicionar blocos, conectar fios e verificar a integridade da estrutura. Com linguagens HDL, o usuário pode se concentrar em descrever exatamente como um circuito funciona, sem ter que se preocupar com os detalhes de sua realização física, que ficam a cargo do sintetizador.

CAPÍTULO 5 – IMPLEMENTAÇÃO

Para implementar o oscilador multiplexado em FPGA, utilizou-se a versão 9.1 do programa Quartus II e o kit lógico DE2-70. Os componentes necessários foram desenvolvidos com uma combinação de diagramas de blocos e programação VHDL.

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 (ver Seção 5.7).

Cada parte constituinte do oscilador descrita na Seção 3 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* (ver Seção 5.5).

5.1 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 [7].

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.

5.2 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.

5.3 REGISTRADORES

Com exceção de Reg3, que possui também uma entrada de *clear* assíncrono (ver Seção 5.6), 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 n. 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, disponibiliza o valor do ciclo de componente n - 1.

5.4 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 blocos lógicos de um FPGA podem ser bastante lentos se comparados a circuitos dedicados de multiplicação, estreitando o gargalo teórico

mencionado anteriormente. Essa alternativa dedicada é disponibilizada em alguns kits lógicos de FPGA's (Tab. 4.1), e é a que se escolheu usar nessa implementação.

5.5 TESTES DE TEMPO

Nessa subseção, mostram-se os testes de tempo realizados para os componentes descritos nas seções acima. A Tabela 5.1 resume numericamente os resultados encontrados.

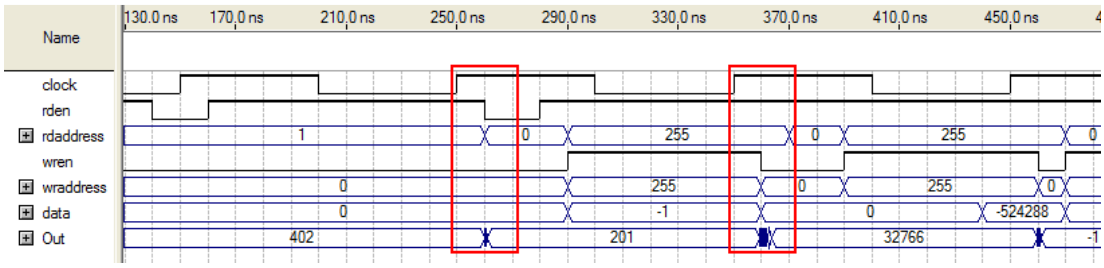


Figura 5.1. Teste de tempo da Phase Angle Increment RAM e da Phase Angle RAM.

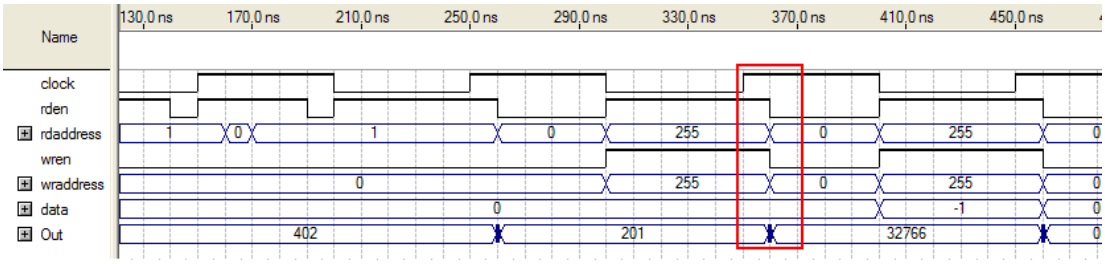


Figura 5.2. Teste de tempo da Amplitude RAM.

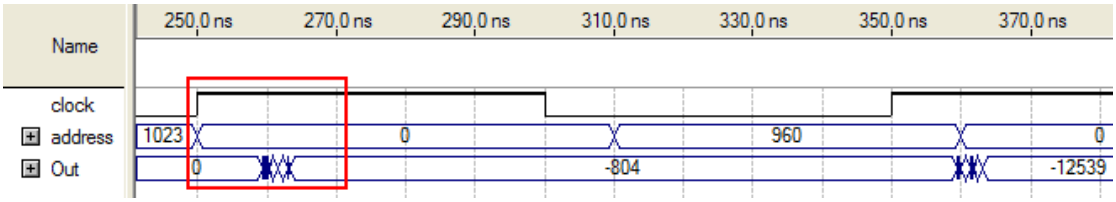


Figura 5.3. Teste de tempo da Sine Memory.

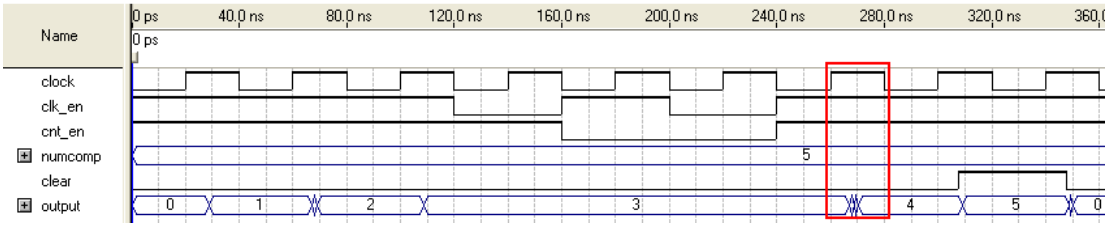


Figura 5.4. Teste de tempo do circuito de endereçamento.

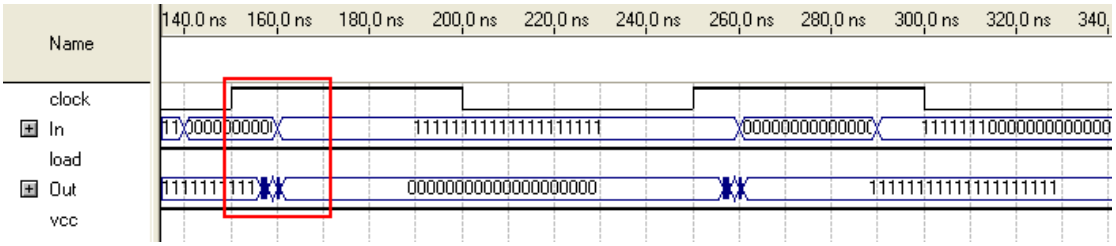


Figura 5.5. Teste de tempo de Reg1.

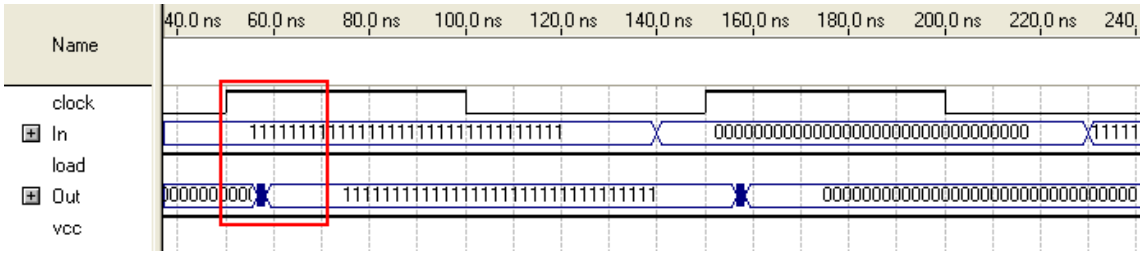


Figura 5.6. Teste de tempo de Reg2.

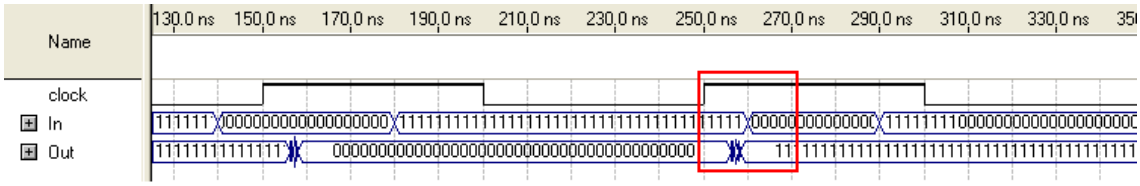


Figura 5.7. Teste de tempo de Reg3.

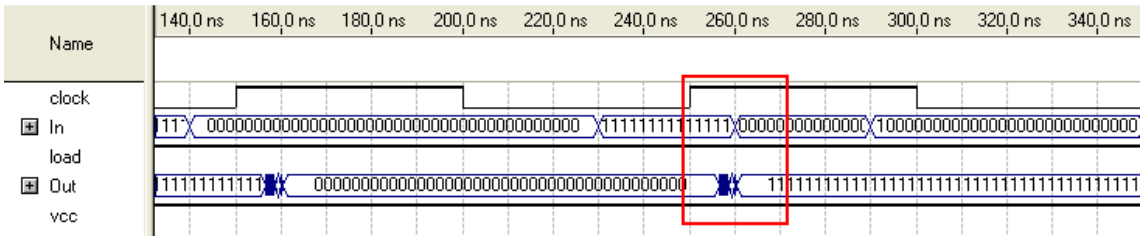


Figura 5.8. Teste de tempo de Reg4.

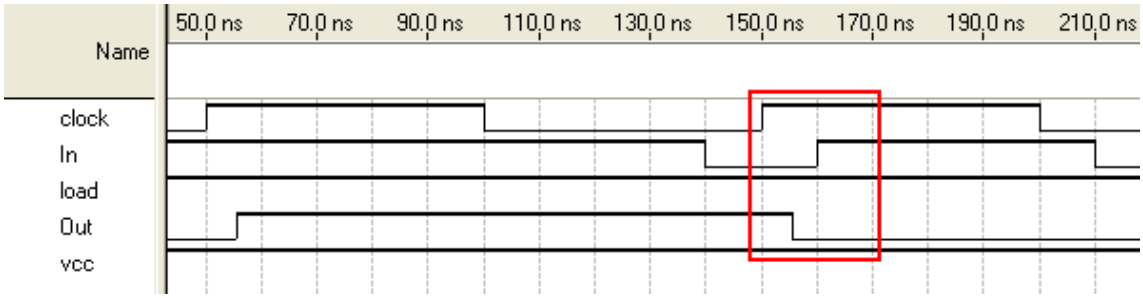


Figura 5.9. Teste de tempo do registrador auxiliar.

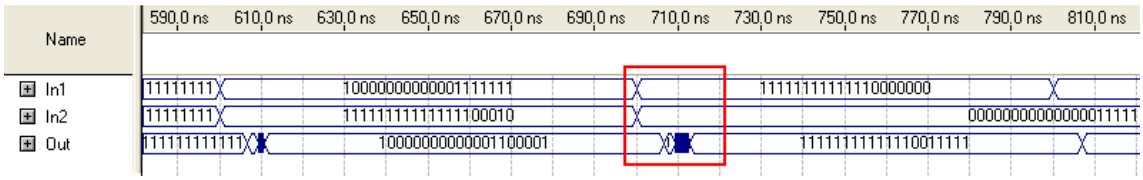


Figura 5.10. Teste de tempo do somador 1.

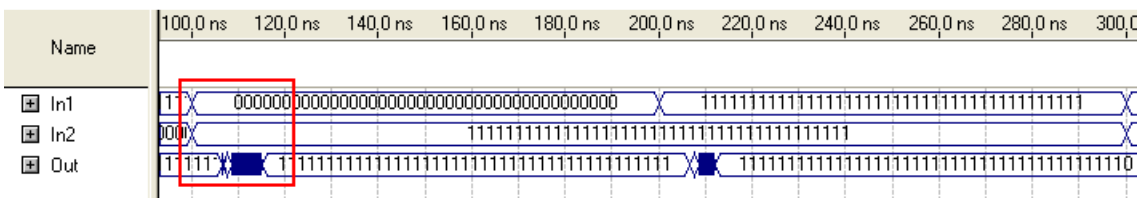


Figura 5.11. Teste de tempo do somador 2.

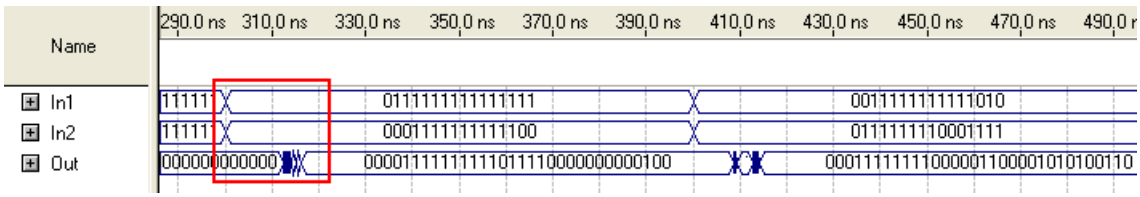


Figura 5.12. Teste de tempo do multiplicador.

Tabela 5.1. Resumo numérico dos resultados obtidos nos testes de tempo.

Componente	Tempo Analisado	Atraso (ns)
Phase Angle Increment RAM / Phase Angle RAM	*Hold* Escrita	10
	Hold Leitura	10
	Escrita	20
	Leitura	20
Amplitude RAM	*Hold* Escrita	10
	Hold Leitura	10
	Escrita	20
	Leitura	15
Sine Memory	*Hold* Leitura	10
	Leitura	20
Endereçamento	*Setup*	15
Reg1	*Setup*	15
	Hold	10
Reg2	*Setup*	15
	Hold	10
Reg3	*Setup*	15
	Hold	10
Reg4	*Setup*	15
	Hold	10
Reg Aux	*Setup*	10
	Hold	10
Adder1	*Setup*	20
Adder2	*Setup*	20
Multiplicador	*Setup*	25

5.6 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 mostrado anteriormente, eles 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 (Fig 3.1). 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, mais rápido. 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 dessa 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.

O código VHDL referente ao bloco Amostra Enable pode ser visto no Anexo II.

5.7 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 do FPGA, o WM8731. A DE2-70, em seus tutoriais, 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 (ver Seção 5.7.2).

Nota-se aqui que, em seu modo de funcionamento padrão, esses blocos pré-prontos exigem um *clock* de 18,432 MHz. Foi justamente para tornar a sincronização com o kit lógico mais segura que se escolheu implementar o oscilador com capacidade para 384 componentes senoidais: com o codec e o oscilador multiplexado sendo controlados à mesma freqüência, pode-se usar um único sinal de relógio para coordenar o sistema completo, evitando possíveis defasagens não-intencionais.

5.7.1 CLOCK DE ENTRADA

A DE2-70 possui dois *clocks* internos, um de 50 MHz e outro de 28 MHz. Escolheu-se, então, utilizar na entrada do circuito uma PLL para criar, a partir do *clock* de 50 MHz, um *clock* com o valor adequado de 18,432 MHz [12].

5.7.2 SERIALIZAÇÃO DE DADOS

O modo de funcionamento padrão do codec da DE2-70 é o chamado *Left Justified*, ilustrado na Fig. 5.13. 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. No caso do oscilador multiplexado, ambos os canais recebem dados idênticos, ou seja, o sistema opera em modo mono. 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 $2m$ vezes f_S, onde m é o número de bits nas palavras que representam as amostras de áudio. Os valores padrão para m são 16, 20 ou 24 bits, 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.

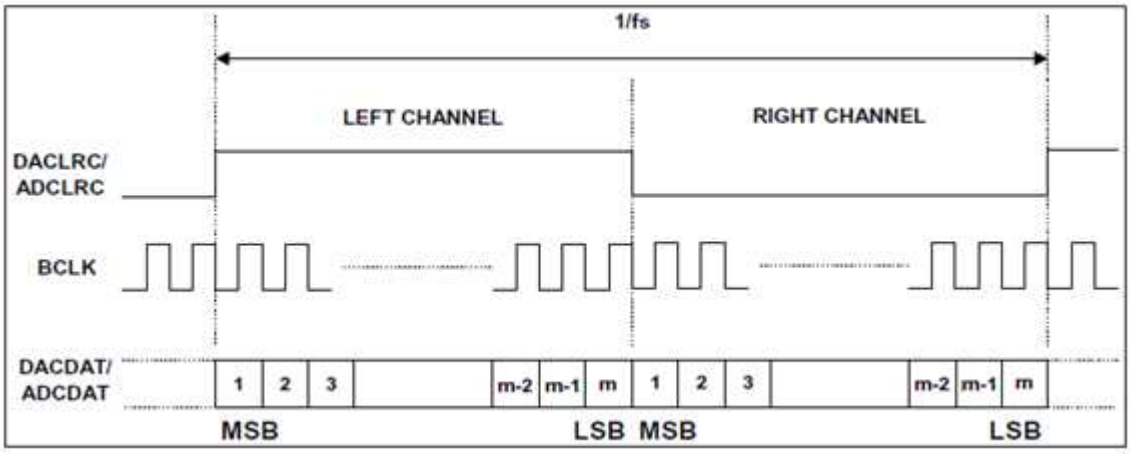


Figura 5.13. Funcionamento *Left Justified* do codec WM8731.

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 Fig. 5.13, ou seja, os dados ficam prontos no canal serial antes da borda de subida de BCLK. Além disso, o novo registrador usa um sinal de *load*, descrito na seção seguinte, para carregar o valor paralelo a ser serializado.

5.7.3 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.

O código VHDL referente ao bloco de sincronização pode ser visto no Anexo III.

5.8 NORMALIZAÇÃO

Com a estrutura descrita até aqui, já é teoricamente possível gerar sons no 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*, de acordo com a Tab. 5.2.

Tabela 5.2. Relação entre o número de componentes ativas e o fator de *shift* recomendado.

Nº de componentes	**Nº recomendado de *shifts***
1	9
2 – 3	8
4 – 7	7
8 – 15	6
16 – 31	5
32 – 63	4
64 – 127	3
128 – 255	2
256 – 384	1

O valor máximo nove foi escolhido porque o decimal 384 ocupa nove dígitos binários. No entanto, como a situação de uso mínimo (uma componente) já ocupa um bit, há uma folga de segurança: todas as configurações descritas na Tab. 5.2 suportam um *shift* a mais do que o recomendado, ou seja, sacrifica-se uma pequena parcela da faixa dinâmica em favor da proteção do usuário contra pequenos erros de cálculo ou faltas de atenção.

Mesmo assim, é importante notar que cabe unicamente 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.

Tomou-se outra medida de segurança nessa fase do desenvolvimento: o MSB da entrada do multiplicador que a Amplitude RAM usa foi mudado para zero *hardwired*. Desse modo, não há risco de que um valor de amplitude mal inserido inverta o sinal do número enviado pela Sine Memory.

O código VHDL referente ao bloco de normalização pode ser visto no Anexo IV.

5.9 TESTES INICIAIS

Após as considerações feitas até agora, decidiu-se confirmar na prática os resultados previstos, com um teste simples desenvolvido especialmente para esse propósito:

- Preencheram-se as oito primeiras palavras da Phase Angle Increment RAM com os valores de incremento correspondentes a oito notas centrais da escala de Dó maior (Tab. 5.3);
- A Phase Angle RAM recebeu apenas valores zero, e a Sine Memory recebeu um ciclo de onda senoidal, ou seja, ambas foram mantidas em seu funcionamento padrão;
- A Amplitude RAM, por sua vez, foi completamente desconectada do circuito, e substituída por uma estrutura de testes baseada em multiplexadores, que funciona da seguinte maneira: por meio dos *toggle switches* da DE2-70, a amplitude das oito primeiras componentes de cada amostra pode ser chaveada entre zero e o valor máximo. Da nona componente em diante, a amplitude é fixada em zero.

Tabela 5.3. Valores de incremento colocados na Phase Angle Increment RAM.

Nota	Valor MIDI	Freqüência	Incremento	20 bits
C4	60	261,6255653006	5,581345393	00000001011001010011
D4	62	293,6647679174	6,264848382	00000001100100001111
E4	64	329,6275569129	7,032054547	00000001110000100000
F4	65	349,2282314330	7,450202271	00000001110111001101
G4	67	391,9954359817	8,362569301	00000010000101110011
A4	69	440,0000000000	9,386666667	00000010010110001011
B4	71	493,8833012561	10,53617709	00000010101000100101
C5	72	523,2511306012	11,16269079	00000010110010100110

A idéia desse teste é que o FPGA se torne um instrumento musical rudimentar, com acesso a uma oitava da escala de Dó maior e timbres puramente senoidais. Dessa maneira, podem-se verificar as freqüências individuais das diversas senóides, os intervalos musicais entre elas, e a ação do mecanismo de normalização.

Instanciado o circuito em modo de testes no 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.

Terminam aqui as alterações realizadas na estrutura do projeto até o presente momento. Sua forma atual pode ser vista no Anexo I, dividida em partes para fácil entendimento.

CAPÍTULO 6 – CONCLUSÕES

A intenção inicial desse trabalho, como mencionado no Capítulo 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 5.9, 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 (Fig 6.1). Os trabalhos futuros mencionados na Seção 6.1 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.

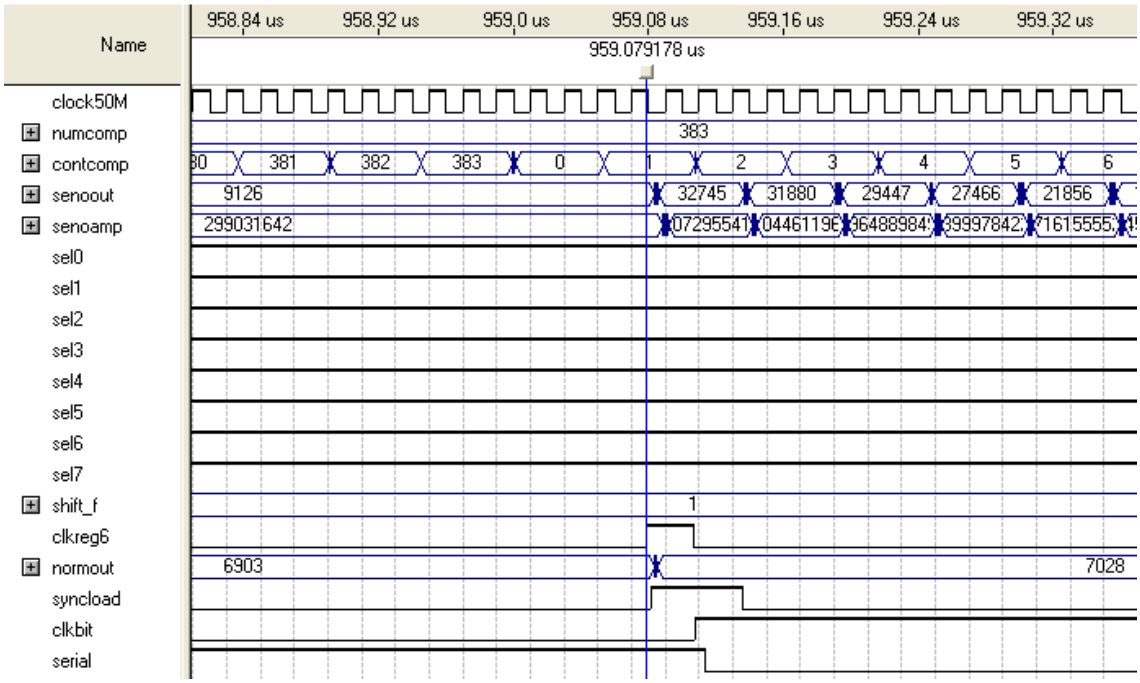


Figura 6.1. Teste de tempo do circuito completo, com 384 componentes simultâneas.

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.

6.1 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;
- 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;
- Forma de interface com o usuário.

O foco mais imediato será voltado 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.

Nota-se que a Phase Angle RAM já passa por um processo de leitura e escrita simultâneas que pode ser utilizado como base para a reescrita em tempo real dos geradores de envoltória. Ainda está em aberto, todavia, o modo como o sistema receberá as envoltórias propriamente ditas: tabelas internas pré-carregadas com pares de pontos e durações parecem ser uma boa solução, mas não foi tomada ainda uma decisão final.

Quanto à interface, um desenvolvimento natural seria o controle do sistema via dispositivos MIDI, com os quais a DE2-70 consegue se comunicar. O trabalho ficaria reduzido a construir um novo bloco que fosse capaz de receber mensagens MIDI e decompô-las em suas partes relevantes, distribuindo os valores por todo o oscilador multiplexado. Isso traria, claro, novas preocupações com a sincronização do sistema, que passaria a ter mais um elemento funcionando com sua própria temporização.

Por fim, fazem-se algumas considerações adicionais:

- O *framework* do oscilador multiplexado pode ser aplicado a outras técnicas de síntese além da aditiva, como, por exemplo, a síntese FM;
- Como dito na Seção 5.7.2, as amostras de áudio enviadas para o codec podem ter mais de 16 bits, o que permitiria aumentar a precisão das senóides calculadas;
- Uma mudança cujo custo-benefício deve ser avaliado é expressar os dados em representação ponto flutuante ao invés de ponto fixo, o que poderia eliminar a necessidade do bloco de normalização;
- O efeito de *panning* pode ser adicionado ao oscilador multiplexado, caso os dois canais do DAT passem a receber dados diferentes.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ROADS, C. **Computer nusic tutorial**. Cambridge: MIT Press, 1996.

[2] LATHI, B. P. **Signal processing and linear systems**. Berkeley-Cambridge Press, 1998.

[3] SPIEGEL, M. R. Schaum's outline of Fourier analysis with application to boundary value problems. McGraw-Hill, 1974.

[4] JANSEN, C. Sine Circuitu: 10,000 high quality sine waves without detours. In: INTERNATIONAL COMPUTER MUSIC CONFERENCE. Michigan Publishing, **Proceedings.** 1991. p.222-225.

[5] SNELL, J. Design of a digital oscillator that will generate up to 256 low-distortion sine waves in real time. In: ROADS, C., STRAWN, J. (ed.). **Foundations of computer music**. Cambridge: MIT Press, 1991. p.289-325.

[6] RAKOWSKI, A. Pitch discrimination at the threshold of hearing. In: SEVENTH INTERNATIONAL CONGRESS ON ACOUSTICS. Budapest, **Proceedings.** 1971.

[7] MOORE, F. R. Table lookup noise for sinusoidal digital oscillators. In: ROADS, C., STRAWN, J. (ed.). **Foundations of computer music**. Cambridge: MIT Press, 1991. p.326-334.

[8] PATTERSON, D. A., HENNESSY, J. L. Enhancing performance with pipelining. In:___. **Computer organization and design.** 3. ed. Elsevier. 2005. p.368-465.

[9] ALTERA. DE2-70 Development and Education Board. Altera. Disponível em <http://www.altera.com/education/univ/materials/boards/de2-70/unv-de2-70-board.html> Acesso em: 11 de dezembro 2013

[10] ALTERA. Quartus II Web Edition Software. Altera. Disponível em <http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html> Acesso em: 11 de dezembro 2013

[11] MEALY, B., TAPPERO, F. Free Range VHDL. Disponível em <http://www.freerangefactory.org/dl/free_range_vhdl.pdf> Acesso em 11 de dezembro 2013

[12] BANERJEE, D. PLL Performance, Simulation and Design. Disponível em <http://www.ti.com/tool/pll_book> Acesso em 12 de dezembro 2013

ANEXOS

Anexo I – Estrutura do oscilador multiplexado no Quartus II.

Anexo II – Código VHDL referente ao bloco Amostra Enable.

Anexo III – Código VHDL referente ao bloco de sincronização.

Anexo IV – Código VHDL referente ao bloco de normalização.

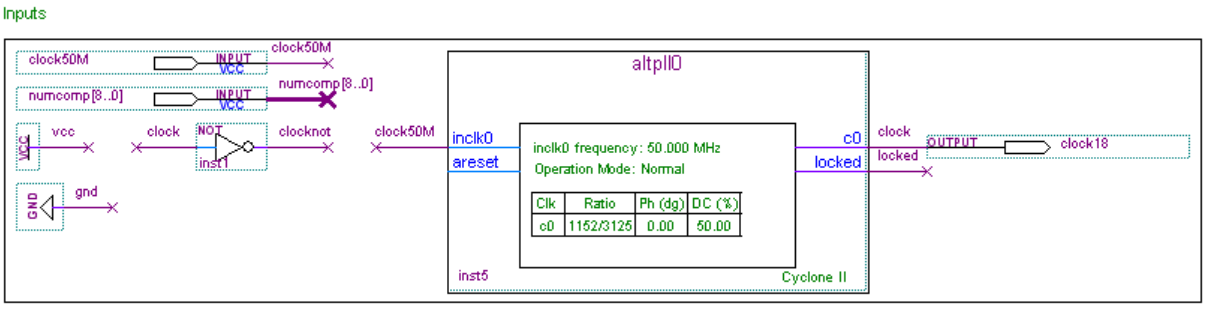


Figura I.1. *Inputs* do oscilador multiplexado.

Os *inputs* do circuito se resumem a:

- *Clock* interno de 50 MHz da DE2-70;
- Conversão desse *clock*, via PLL, para 18,432 MHz;
- Negação do *clock* de 18,432 MHz, para implementar o acesso às bordas de descida;
- Número de componentes usado pelo circuito de endereçamento;
- Bits alto e baixo para controle do circuito geral, quando necessário.

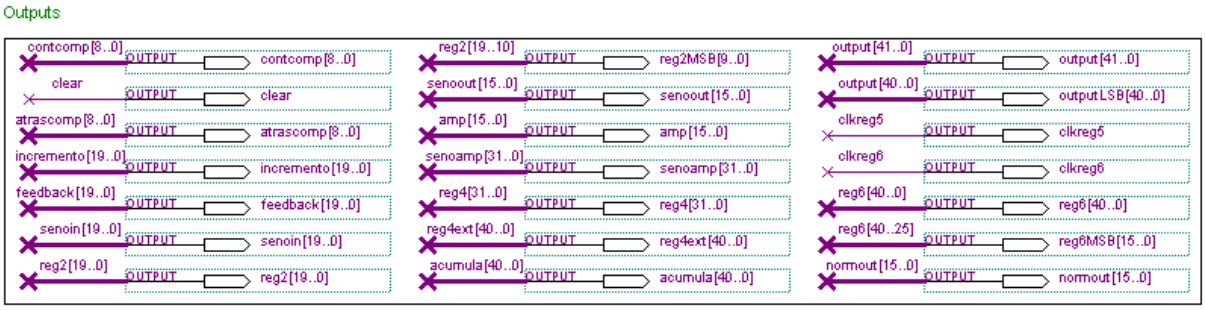


Figura I.2. *Outputs* de teste do oscilador multiplexado.

Os vários *outputs* vistos na figura acima foram utilizados puramente para fins de teste da estrutura, e foram indispensáveis para encontrar os erros ao longo do desenvolvimento. Atualmente, com o circuito funcionando bem, esses pinos servem apenas para possibilitar a simulação via *vector waveforms* do oscilador multiplexado.

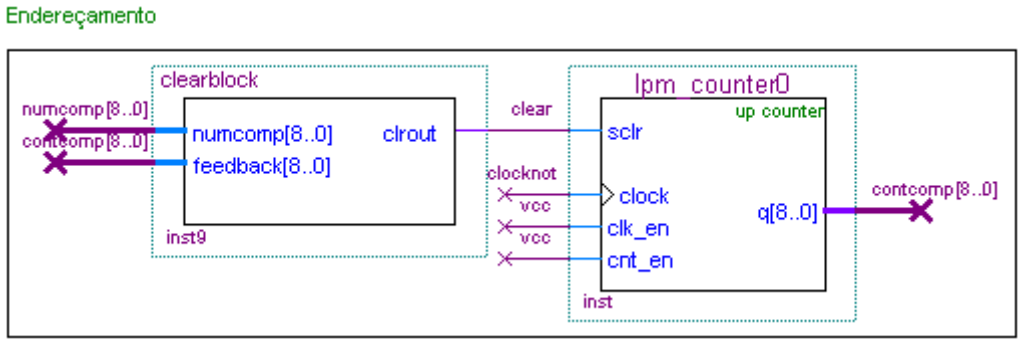


Figura I.3. Circuito de endereçamento do oscilador multiplexado.

O circuito de endereçamento é composto por um contador crescente simples, e um bloco que zera a contagem quando ela chega ao número de componentes definido pelos *inputs*.

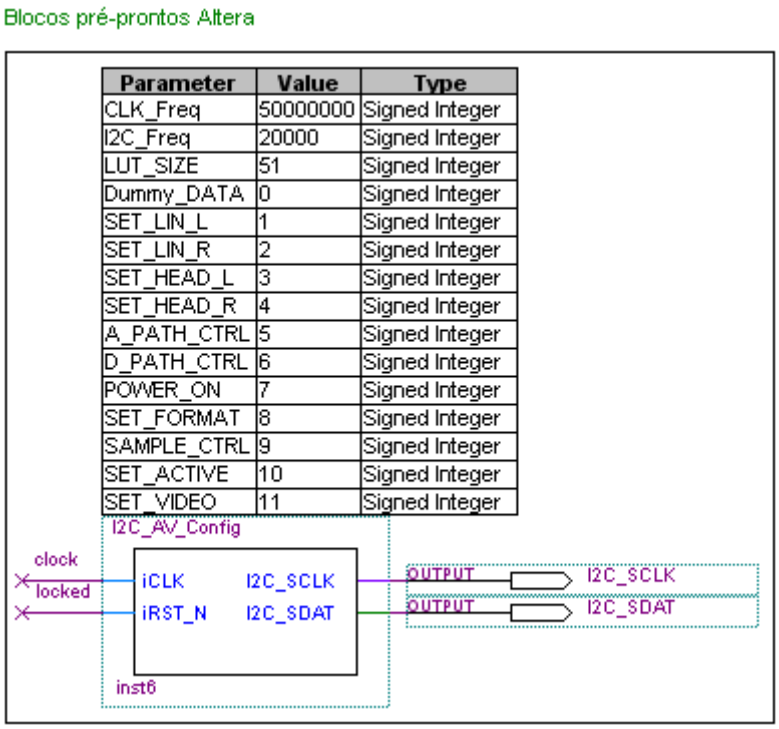


Figura I.4. Blocos pré-prontos dos tutoriais da Altera.

O bloco acima foi obtido em um dos tutoriais que são fornecidos no CD de instalação dos *softwares* relacionados à DE2-70, e sua função é realizar a comunicação da implementação do usuário com o codec WM8731. Todas as suas configurações foram mantidas inalteradas.

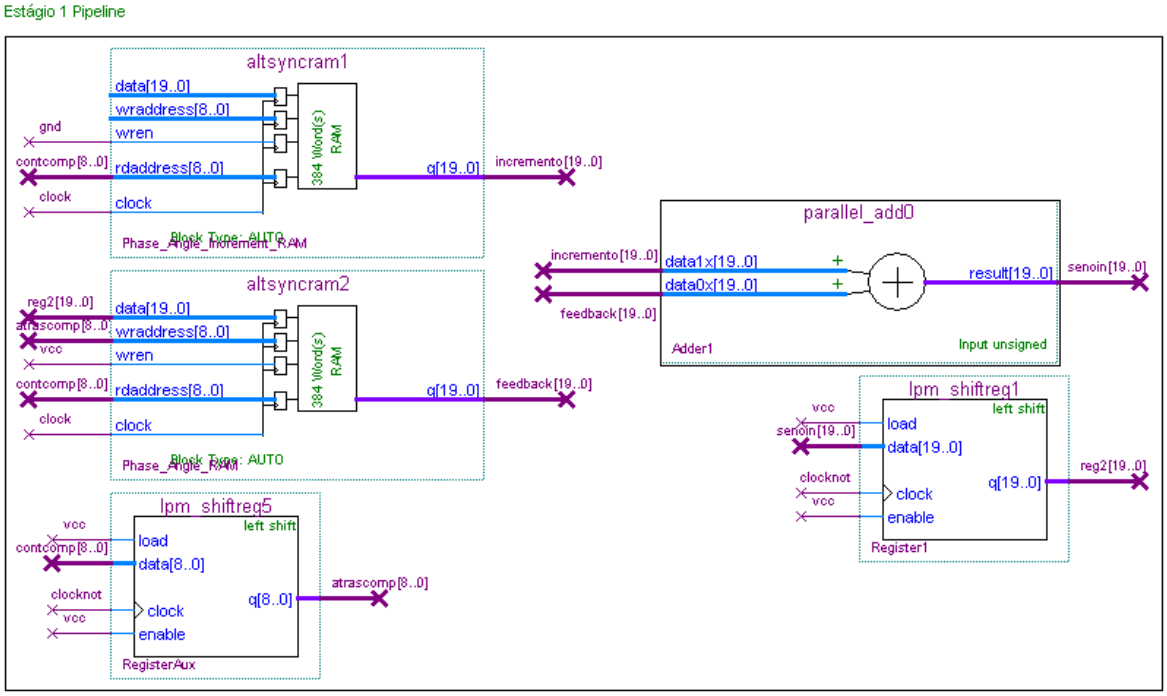


Figura I.5. Estágio 1 do *pipeline* do oscilador multiplexado.

A estrutura do primeiro estágio do *pipeline* é bastante similar àquela mostrada na Fig. 3.1, com a adição do registrador auxiliar.

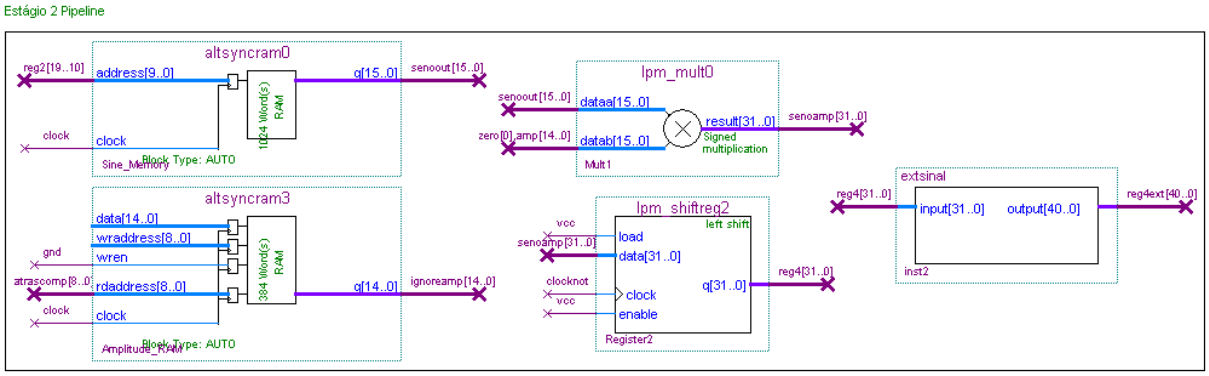


Figura I.6. Estágio 2 do *pipeline* do oscilador multiplexado.

O estágio 2 do *pipeline* também segue a Fig. 3.1, dessa vez com a adição de um bloco de extensão de sinal, para evitar a perda de informação nas somas do estágio seguinte.

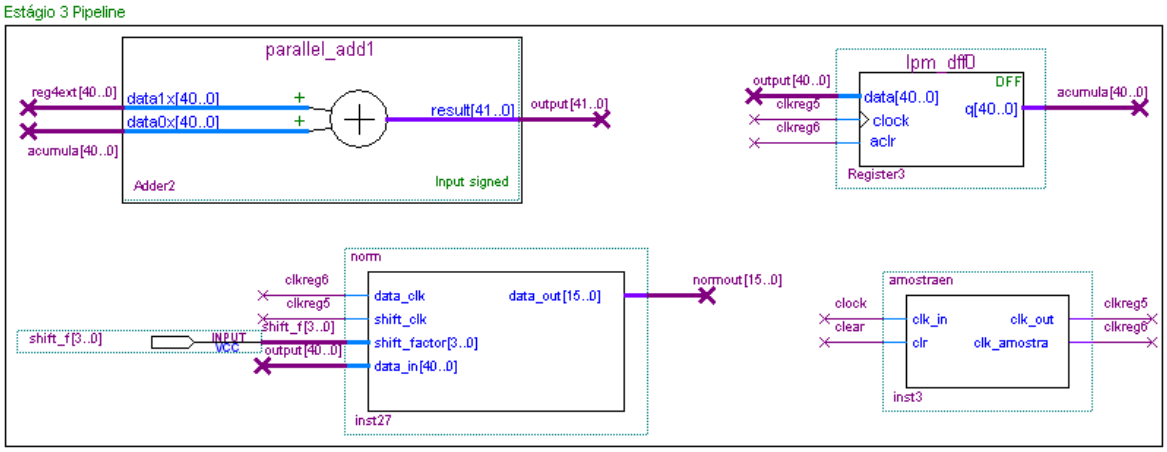


Figura I.7. Estágio 3 do *pipeline* do oscilador multiplexado.

Na estrutura do estágio 3 do *pipeline*, vêem-se tanto o bloco de normalização quanto o Amostra Enable, atualizações sobre o formato inicial na Fig. 3.1.

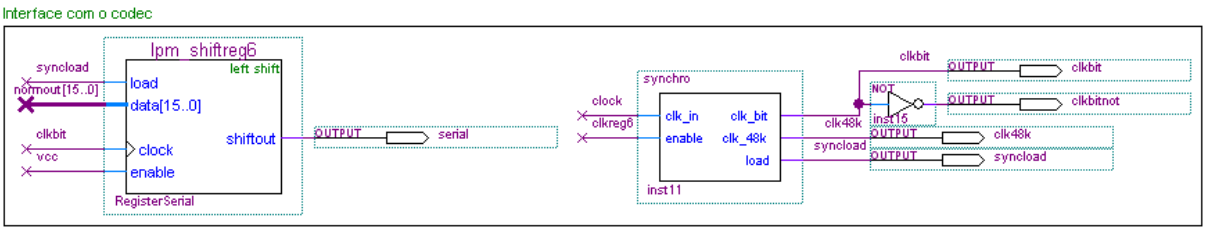


Figura I.8. Interface do oscilador multiplexado com o codec.

Aqui, observam-se o registrador de serialização e o bloco de sincronização, descritos nas Seções 5.7.2 e 5.7.3, respectivamente.

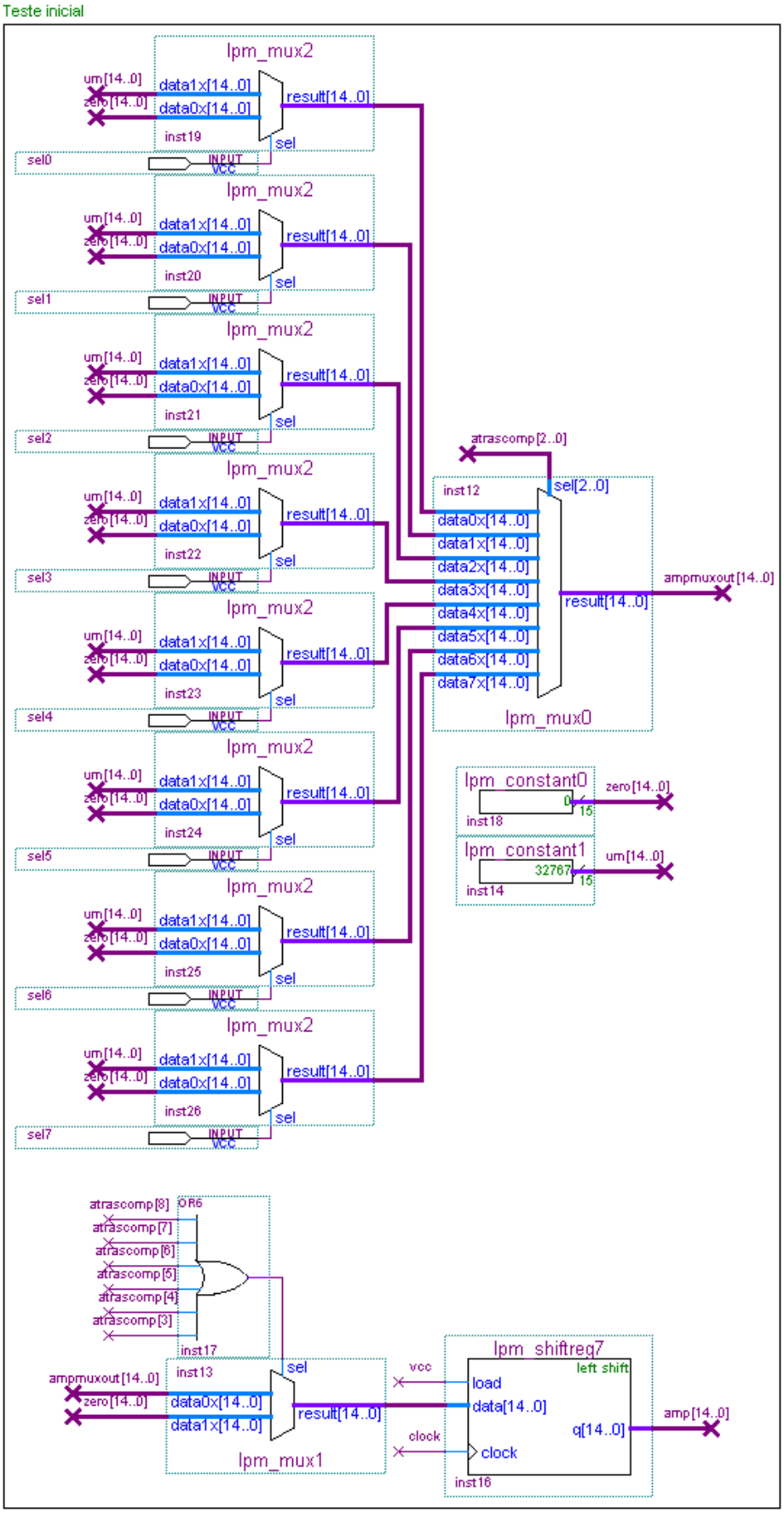


Figura I.9. Estrutura de testes mencionada na seção 5.9.

Por fim, mostra-se a estrutura de testes baseada em multiplexadores que foi utilizada nos testes iniciais do projeto.

```
library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.numeric_std.all;

entity amostraen is
    port
    (
        clk_in, clr : in std_logic;
        clk_out, clk_amostra : out std_logic
    );
end amostraen;

architecture amostraen_rtl of amostraen is

signal tmp_amostra : std_logic;
signal start_flag : std_logic := '0';
signal counter : unsigned(1 downto 0);

begin

    pulse_gen : process(clk_in) is
    begin
        if (rising_edge(clk_in)) then
            if (start_flag = '1') then
                if (counter = "01") then
                    counter <= counter + 1;
                elsif (counter = "10") then
                    start_flag <= '0';
                else
                    counter <= counter + 1;
                end if;
            elsif (clr = '1') then
                counter <= "00";
                start_flag <= '1';
            end if;
        elsif (falling_edge(clk_in)) then
            if (counter = "01") then
                tmp_amostra <= '1';
            else
                tmp_amostra <= '0';
            end if;
        end if;
    end process pulse_gen;

    clk_out <= clk_in and not(tmp_amostra);
    clk_amostra <= clk_in and tmp_amostra;

end amostraen_rtl;
```

Figura II.1. Código VHDL referente ao bloco Amostra Enable.

```
library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.numeric_std.all;

entity synchro is
    port
    (
        clk_in, enable : in std_logic;
        clk_bit, clk_48k, load : out std_logic
    );
end synchro;

architecture synchro_rtl of synchro is

signal tmp_bit : std_logic;
signal tmp_48k : std_logic;
signal tmp_load : std_logic;
signal start_flag : std_logic := '0';
signal bitcounter : unsigned(2 downto 0);
signal samplecounter : unsigned(7 downto 0);

begin

    start : process(enable) is
    begin
        if (rising_edge(enable)) then
            start_flag <= '1';
        else
            start_flag <= start_flag;
        end if;
    end process start;

    bit_clock : process(clk_in) is
    begin
        if (start_flag = '1') then
            if (rising_edge(clk_in)) then
                if (bitcounter = "101") then
                    bitcounter <= "000";
                else
                    bitcounter <= bitcounter + 1;
                end if;
            elsif (falling_edge(clk_in)) then
                if (bitcounter = "000") then
                    tmp_bit <= not(tmp_bit);
                else
                    tmp_bit <= tmp_bit;
                end if;
            end if;
        else
            tmp_bit <= tmp_bit;
        end if;
    end process bit_clock;

    sample_clock : process(clk_in) is
```

```
    begin
        if (start_flag = '1') then
            if (rising_edge(clk_in)) then
                if (samplecounter = "10111111") then
                    samplecounter <= "00000000";
                else
                    samplecounter <= samplecounter + 1;
                end if;
            elsif (falling_edge(clk_in)) then
                if (samplecounter = "00000000") then
                    tmp_48k <= not(tmp_48k);
                else
                    tmp_48k <= tmp_48k;
                end if;
            end if;
        else
            tmp_48k <= tmp_48k;
        end if;
    end process sample_clock;

    load_gen : process(clk_in) is
    begin
        if (samplecounter = "00000000") then
            tmp_load <= '1';
        else
            tmp_load <= '0';
        end if;
    end process load_gen;

    clk_bit <= tmp_bit;
    clk_48k <= tmp_48k;
    load <= tmp_load;

end synchro_rtl;
```

Figura III.1. Código VHDL referente ao bloco de sincronização.

```
library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.numeric_std.all;

entity norm is
    port
    (
        data_clk, shift_clk : in std_logic;
        shift_factor : in integer range 0 to 9;
        data_in : in signed(40 downto 0);
        data_out : out signed(15 downto 0)
    );
end norm;

architecture norm_rtl of norm is

signal tmp_shift_factor : integer range 0 to 9;
signal tmp_data_in : signed(40 downto 0);

begin

    max : process(shift_clk) is
    begin
        if (shift_factor > 9) then
            tmp_shift_factor <= 9;
        else
            tmp_shift_factor <= shift_factor;
        end if;
    end process max;

    shift : process(data_clk) is
    begin
        if (rising_edge(data_clk)) then
            tmp_data_in <= shift_left(data_in, tmp_shift_factor);
        else
            tmp_data_in <= tmp_data_in;
        end if;
    end process shift;

    data_out <= tmp_data_in(40 downto 25);

end norm_rtl;
```

Figura IV.1. Código VHDL referente ao bloco de normalização.