UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RAFAEL DA SILVA BARBOZA

IMPLEMENTAÇÃO DE GERADOR DE FORMAS DE ONDAS ARBITRÁRIAS

Canoas, Novembro de 2012

RAFAEL DA SILVA BARBOZA

IMPLEMENTAÇÃO DE GERADOR DE FORMAS DE ONDAS ARBITRÁRIAS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Departamento:

Engenharia Elétrica

Área de Concentração

Processadores

Professor Orientador:

MSc. Eng. Eletr. Paulo César Cardoso Godoy – CREA-RS: 11.682-2

Canoas

2012

FOLHA DE APROVAÇÃO

Nome do Autor: Rafael da Silva Barboza

Matrícula: 032007268-1

Título: Implementação de Gerador de Formas de Ondas Arbitrárias

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Professor Orientador:

MSc. Eng. Eletr. Paulo César Cardoso Godoy

CREA-RS: 11.682-2

Banca Avaliadora:

MSc. Eng. Eletr. Miriam Noemi Cáceres

CREA-RS: 06.7231-D

Conceito Atribuído (A-B-C-D):

MSc. Eng. Eletr. Luís Fernando Espinosa Cocian

CREA-RS: 08.8866-D

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor Rafael da Silva Barboza	Orientador Paulo César Cardoso Godoy
Avaliador Miriam Noemi Cáceres	Avaliador Luís Fernando Espinosa Cocian

Relatório Aprovado em:

DEDICATÓRIA

Dedico aos meus pais...

AGRADECIMENTOS

A todos que colaboraram direta ou indiretamente na elaboração deste trabalho, o meu reconhecimento.

Agradeço muito aos meus pais Ildo e Ilizia e as minhas irmãs Leticia e Caroline pela contribuição e compartilhamento das dificuldades durante o curso de engenharia.

A minha noiva Carla pela paciência durante a minha ausência e pelo apoio durante toda a trajetória.

Ao Professor Paulo César Godoy pelo estímulo, dedicação e esforço pessoal proporcionado.

Aos colegas Gustavo Imperatori e Laércio Furini pelas sugestões e observações valiosas durante os longos períodos de estudos juntos.

EPÍGRAFE

Humildade Sempre...

RESUMO

BARBOZA, Rafael da Silva. **Gerador de Sinais Arbitrário**. 95 f. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2012.

O objetivo do projeto é o desenvolvimento de um gerador de funções arbitrário, implementado em um *kit* didático contendo um microcontrolador LPC2378 de 32-Bit ARM7. A primeira parte do trabalho apresenta o referencial teórico pertinente ao desenvolvimento do trabalho, tais como os princípios de funcionamento dos geradores de sinais, teoria da sintetização de frequências, método de conversão digital-analógica usando a topologia R-2R, interpolação linear e descrição do microcontrolador. Depois do referencial teórico, o capítulo três apresenta os materiais e métodos envolvidos para o desenvolvimento de cada etapa do projeto, bem como a descrição do *hardware* que é composto pelo *kit* didático e por uma placa de armazenamento dos pontos calculados, conversão de sinais digitais para analógicos e condicionamento do sinal. Além disso, são apresentadas as etapas de desenvolvimento e o funcionamento do *firmware* em linguagem C e do *software* especifico para a interface com o usuário desenvolvido em *Builder*. Por fim, são apresentados e discutidos os resultados obtidos pela simulação de funcionamento do sistema, no qual foi testada a geração dos sinais e realizada a análise de erro através de comparação dos sinais ideais com os sinais reais gerados.

Palavras chave: Gerador de Funções. Microcontrolador. Frequência

ABSTRACT

BARBOZA, Rafael da Silva. **Arbitrary Waveform Generator**. 95 f. Work of Conclusion of Course in Eletrical Engineering – Eletrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2012.

The aim of this project is to develop an arbitrary function generator, implemented in an instructional kit, which contains a LPC2378 microcontroller based on a 32-Bit ARM7. The first part of this paper brings out some theoretical reference about subjects that help in the development of this project: signal generator operating principles, direct digital synthesis (DDS), design of the digital to analog conversion with R/2R ladder networks, linear interpolation and microcontroller description. It also presents materials and methods that are used to build each part of the generator, as well as descriptions of the hardware, which is made up of didactic kit, board storage, digital to analog conversion, and signal conditioning. Besides, parts of the development and the operation of the firmware in C language, and the specific software to interface with the user on *Builder* are described. In order to conclude, the results are reported and discussed. They are obtained in the system functioning simulation, which tests signal generation, makes the error analyses through the comparison of ideal signals to generated real signals.

Keywords: Function Generator. Microcontroller. Frequency.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ARM – *Advanced RISC Machine*

ASCII - *American Standard Code for Information Interchange*

DAC – *Digital-to-analog Converter*

DCE - *Data Communication Equipment*

DDS – *Direct Digital Synthetis*

DMA - *Direct Memory Access*

DTE - *Data Terminal Equipment*

EIA - *Electronic Industries Alliance*

IBM - *International Business Machines Corporation*

LCD - *Liquid Crystal Display*

MAC - *Media Access Control*

PHY - *Physical Layer*

PLL - *Phase Locked Loop*

PLL - *Phase Locked Loop*

PWM – *Pulse-width Modulation*

RAM - *Random Access Memory*

RISC - *Reduced Instruction Set Computer*

RTC – *Real Time Clock*

SD – *Security Digital Card*

SPI - *Serial Peripheral Interface*

SRAM - *Static Random Access Memory*

USB – *Universal Serial Bus*

VCO - *Voltage-controlled Oscillator*

LISTA DE SÍMBOLOS

μHz – [micro-hertz]

dB – [decibél]

Hz – [hertz]

kB – [kilo-byte]

kHz – [kilo-hertz]

mA – [mili àmperes]

MHz – [mega-hertz]

ns – [nano segundos]

V – [Tensão Elétrica]

V/μs – [Volts / micro-segundo]

SUMÁRIO

1. INTRODUÇÃO

Atualmente, os geradores de função são instrumentos de grande utilidade na eletrônica, sendo necessários para o desenvolvimento de novos projetos, para testes e manutenções em equipamentos. Os geradores analógicos, muitas vezes, são considerados inadequados devido ao fato de não se conseguir a exatidão desejada, principalmente com relação à frequência dos sinais gerados. Comercialmente, um sistema de síntese digital direta (*Direct Digital Synthetis*) é uma boa alternativa ao sintetizador analógico tradicional, pois além de atender melhor os requisitos técnicos possuí uma larga escala de integração priorizando a redução dos componentes, gerando assim equipamento de menor custo.

Paralelamente, com os avanços das tecnologias, o desenvolvimento de novos produtos está sendo cada vez mais otimizado. Esta constante inovação atinge também os projetos eletrônicos através do uso de dispositivos embarcados, onde já é possível encontrar equipamentos com tamanhos cada vez mais reduzidos, porém com a capacidade de realizar grandes processamentos. Essa tecnologia é projetada especialmente para atender demandas bem definidas e específicas, com o objetivo de realizar as tarefas usando o mínimo possível de recursos, porém possuindo as principais funcionalidades potencializadas. Toda essa inovação é de grande valia, pois ao implantar o uso de um dispositivo embarcado, além de conseguir um sistema moldado e totalmente flexível, o custo é baixo.

A tendência de otimização dos projetos eletrônicos é cada vez mais marcante. Com o crescente avanço tecnológico, precisão, confiabilidade e eficiência são cada vez mais exigidas, além do aumento da qualidade e quantidade da produção. Esse crescimento está diretamente ligado aos avanços da microeletrônica e da potencialização das necessidades de mercado (Maitelli,2003).

O objetivo deste trabalho é desenvolver um gerador de funções a partir da tecnológica embarcada e do método de DDS, interagindo com um protótipo eletrônico, possibilitando o ajuste de frequência de maneira digital e a amplitude de forma manual. Este projeto busca obter uma alternativa para facilitar a geração de sinais, através do emprego de dispositivos integrados que possibilitem o desenvolvimento de novas tecnologias de maneira amigável, facilitando o emprego

do instrumento. Quanto à geração, será possível fornecer sinais de forma arbitrária, onde o usuário terá a possibilidade de desenhar via *software* especifico o sinal a ser gerado.

A grande vantagem de um gerador de funções feito com DDS em relação a outro, feito com técnicas analógicas, está na precisão da frequência, devido ao fato da mesma estar diretamente ligada à precisão do cristal oscilador; e ainda, na resolução da frequência selecionada, que pode chegar a μHz (micro-hertz), dependendo da quantidade de *bits* da palavra de ajuste de frequência. Quanto maior a quantidade de *bits*, mais a resolução se aproxima dos μHz (Osicom Technologies Inc,1983).

O desenvolvimento e estudo do trabalho estão estruturados em quatro capítulos. O capítulo um situa o leitor e expõe os objetivos do projeto. O capítulo dois apresenta a fundamentação teórica para a realização do projeto, onde são abordadas as principais características e as técnicas necessárias que compõem o desenvolvimento do gerador de funções. No capítulo três serão expostos os detalhes do protótipo, assim como a estruturação do *software* e *hardware* necessários. Por fim, no capítulo quatro serão apresentadas as conclusões e sugestões para futuras melhorias.

2. REFERENCIAL TEÓRICO

2.1. Gerador de Funções

O gerador de funções é um instrumento que permite o fornecimento de sinais elétricos de formas de onda, frequências e amplitudes diversas. Tem a capacidade de gerar sinais senoidais, triangulares, quadrados e dente-de-serra, com frequências e amplitudes variáveis.

Esses equipamentos possuem diversos ajustes, porém a quantidade é variável entre os diversos modelos. Dentre as principais características destaca-se a capacidade de gerar níveis de tensão variáveis para os sinais gerados ao longo do tempo. A Figura 2-1 apresenta um gerador de funções comercial:

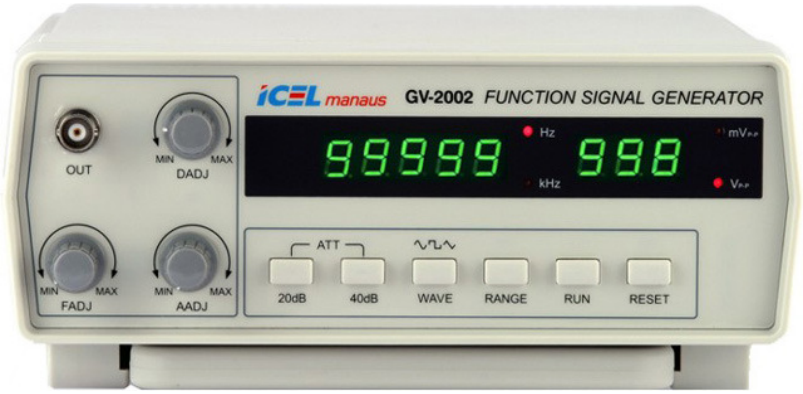


Figura 2-1 - Gerador de funções marca Icel, modelo GV-2002.
Fonte: http://www.icel-manaus.com.br/produto_descricao.php?id=28&.

2.1.1. Tipos de Sinais Fornecidos

Os sinais fornecidos pelos geradores de função geralmente possuem amplitude e frequências máximas e mínimas delimitadas e podem variar de maneira contínua, possuindo um número finito de valores em um determinado intervalo de tempo.

Todo o sinal é uma função que possuí uma amplitude variando com deslocamento no tempo. Assim pode-se definir a amplitude como o nível máximo ou mínimo de tensão alcançado pelo sinal gerado. Já a frequência representa o número

de vezes em que a amplitude percorreu o caminho entre os pontos máximos e sucessivos do sinal durante um ciclo, sendo obtida através da seguinte equação:

$$f = \frac{1}{T} \qquad \text{Equação (2.1)}$$

Onde:

f – frequência (Hz);

T – tempo de um ciclo (s);

A Figura 2-2 ilustra os possíveis sinais gerados por um gerador de funções:

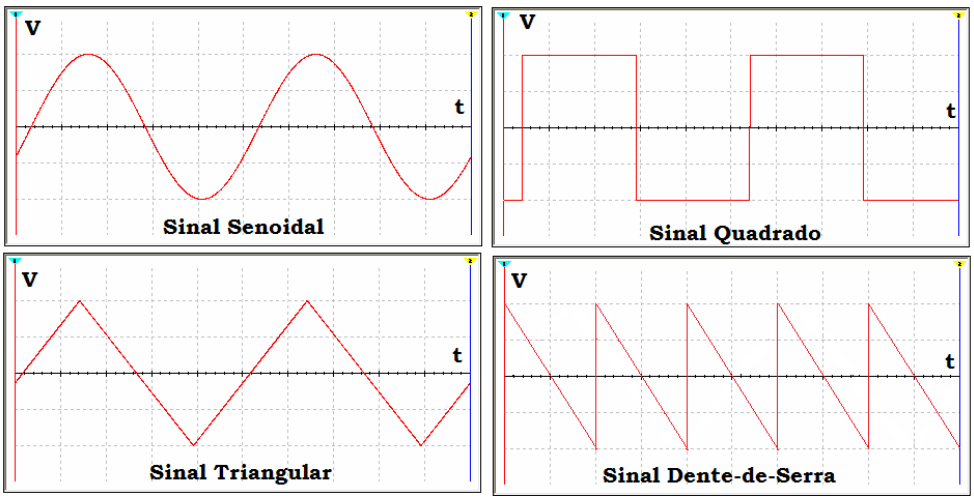


Figura 2-2 - Forma de sinais básicos fornecidos pelo gerador de função

Graficamente representada, as alterações de amplitude nas ondas de tensão mostram as variações do módulo no eixo horizontal. Assim, as tensões acima do eixo horizontal são as que possuem polaridade positiva (+), enquanto as tensões abaixo do eixo horizontal têm polaridade negativa (-). O eixo horizontal demostra a variação do tempo (GUSSOW, 1995).

A onda senoidal é derivada do movimento circular propagando-se ao longo do tempo no eixo vertical, e obedece a uma função seno ou cosseno. Pode ser medida em uma base temporal ou angular. Como o tempo para completar um ciclo completo depende da frequência da onda, geralmente os valores em uma onda senoidal são delimitados por medidas angulares, e por esse motivo devem ser expressos em graus ou em radianos. Seu período é igual ao tempo necessário para que se complete um ciclo completo de onda.

A onda quadrada é caracterizada pela alternância regular e instantânea entre um estado de amplitude mínimo e outro estado de amplitude máximo, sendo que cada um destes estados tem duração igual. São geralmente utilizadas como

referências de tempo para circuitos eletrônicos devido as suas transições rápidas e intervalos de tempos precisos. Entretanto, essa forma de onda pode conter uma grande quantidade de harmônicas, interferindo em circuitos.

A onda triangular é caracterizada por uma ascendência linear até a amplitude máxima da onda, seguida imediatamente por uma descendência linear até a amplitude mínima. Os tempos de subida e descida podem ser iguais ou diferentes. São geralmente empregadas em controles de modulação para acionamentos elétricos por possuir uma quantidade de harmônicas relativamente baixas.

Já a onda dente de serra é um caso particular de onda triangular, ocorrendo nos casos extremos em que os tempo de subida ou de descida de uma onda triangular são iguais a zero, pode ser classificadas como descendentes ou ascendentes.

2.2. Gerador de Funções Arbitrário

Os geradores arbitrários estão sendo amplamente empregados, proporcionando uma maior flexibilidade, precisão e um alto desempenho, pois em muitos casos são exigidos sinais controláveis para a simulação do funcionamento normal de um circuito. Ao contrário dos geradores de funções normais, esse modelo de equipamento é capaz de gerar qualquer forma de onda arbitrariamente, pois se fazem do uso de técnicas de processamento digital de sinais para a sintetização de frequências e geração de amplitude.

Para o desenvolvimento das técnicas de processamento dos sinais a serem gerados são usualmente empregados algum modelo de microcontrolador. E em muitos casos a interface para a escolha da forma de onda a ser gerada é feita via *software* em um computador convencional, sendo necessária a ligação entre *hardware* e *software* através de alguma forma de comunicação convencional. A Figura 2-3 retrata uma onda senoidal sendo gerada a partir de *software* especifico:

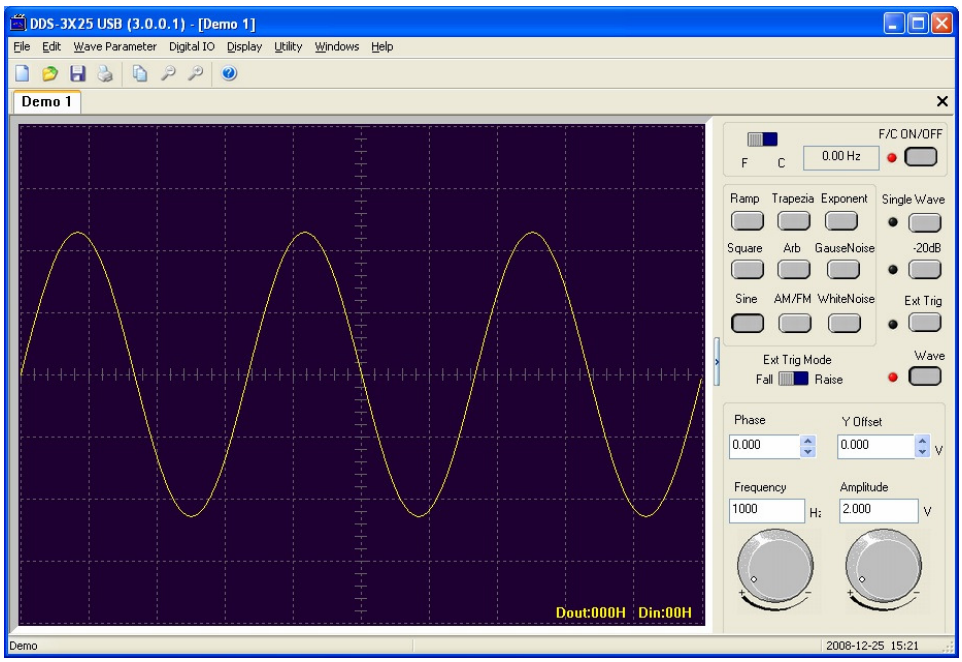


Figura 2-3 - Geração de senóide via software

Fonte: http://www.hantek.net/Product/DDS-3x25/User's%20Guide.pdf

Na Figura 2-3 pode-se observar que é possível arbitrar tanto a frequência quanto a amplitude da forma de onda a ser gerada. Além disso, é possível fazer as escolha dentre as diversas formas de onda disponíveis de geração.

Como principais características dos geradores de sinais arbitrários destaca-se a possibilidade de geração de qualquer modelo de sinal a gerado, entretanto deve-se destacar também:

- Alta performance com custo baixo;
- Resolução alta, pois empregam o uso de Conversores Analógico/Digital de no mínimo 14 *bits*;
- Possibilidade de operação em alta frequência;
- Operação por síntese direta digital;

Deve-se salientar que apesar de este modelo de gerador ser altamente recomendado para diversas atividades no ramo da eletrônica, alguns equipamentos ainda deixam a desejar devido as limitações de *hardware*, cujas propriedades podem variar consideravelmente dentro das especificações dos fabricantes, devido ao aumento de temperatura.

2.3. Sintetização de Frequências

A sintetização de frequência é a técnica utilizada para gerar, a partir de uma referência, uma outra frequência previamente determinada. Sua precisão está diretamente ligada à referência, sendo que os sinais sintetizados são medidos em partes por milhão. O sucesso do processo de síntese está agregado à pureza espectral da frequência gerada, sendo que a frequência de saída será sempre diretamente proporcional a frequência de referência (Analog Devices,1999).

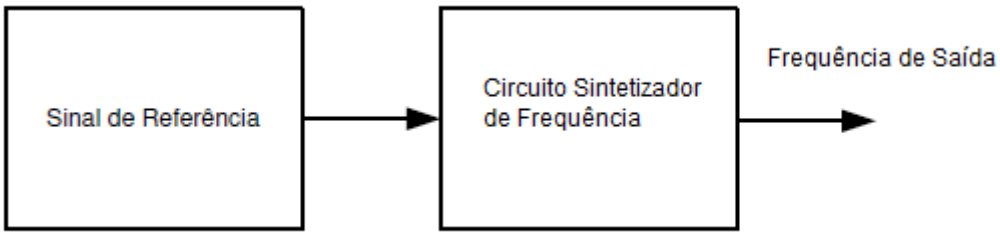


Figura 2-4 - Princípio do sintetizador de frequência

O termo sintetização de frequências é usado para designar a implementação das técnicas de síntese existentes, sendo capaz de gerar uma frequência de saída dentre um grande número de frequências de saídas possíveis, sob comando externo. Sendo que, a frequência de saída é múltipla de uma frequência padrão, ou seja:

$$fs = \frac{N}{M} \times fr \qquad \text{Equação (2.2)}$$

Onde:

fr – frequência de saída (Hz);

N e M – números inteiros;

fr – frequência padrão(Hz);

Atualmente, existem três métodos para a sintetização de frequências:

- Analógica Direta – Ocorre a combinação de misturador/filtro/divisor;
- Analógica Indireta – Através do uso do *Phase Locked Loop* (PLL);
- Digital Direta;

2.3.1. Síntese Analógica Direta

Na síntese analógica direta são usadas manipulações matemáticas para a geração da frequência desejada. É considerada de maneira "direta" por não usar o processo de correção de erro, com isso, a obtenção do sinal de saída está diretamente relacionada à qualidade do sinal de referência. Como principal vantagem destaca-se a possibilidade de trocas de frequências de maneira rápida, pois não depende de realimentação, como a síntese indireta. Porém, sua implementação necessita de alto custo, pois são necessários circuitos muito complexos, principalmente quando há a necessidade de uma alta resolução (Analog Devices,1999).

Essa sintetização possuí a capacidade de retornar as frequências selecionadas anteriormente, através do uso da "memória de fase", e nesse caso, ocorre o retorno da frequência na mesma fase em que estava. Essa característica pode ser de grande utilidade em determinadas situações, mas deve ser levado em consideração o alto custo desse modelo de sintetizador, pois o mesmo necessita de um grande número de filtros para cada uma das frequências de operação.

2.3.2. Síntese Analógica Indireta

Nesse método de sintetização, é empregado o uso de *Phase Locked Loop* (PLL), que são circuitos que podem ser sintetizados para reconhecer um sinal ou frequência pré-determinadas. Seu principio de funcionamento é mostrado pela Figura 2-5:

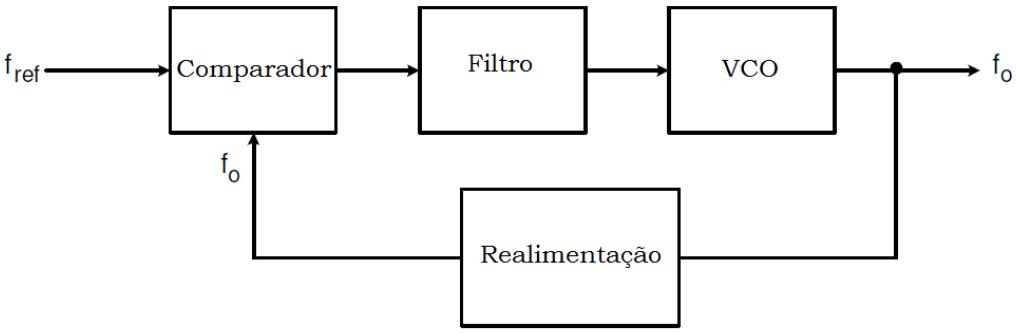


Figura 2-5 - Diagrama de blocos de um PLL

O PLL é formado por três circuitos principais: um comparador de fase, um oscilador controlado por tensão (VCO) e um filtro. Seu funcionamento é baseado em um sistema no qual o VCO gera um sinal de frequência proporcional a uma tensão aplicada em sua entrada. Logo após, uma amostra do sinal de saída é entregue em uma das entradas do comparador de fase, através da realimentação. O comparador

de fase compara o sinal vindo da realimentação com um sinal de referência. Esta comparação resulta em um sinal proporcional à diferença de frequência ou de fase dos sinais de entrada, o sinal obtido do comparador de fase é entregue ao filtro que por sua vez, entrega do VCO uma tensão contínua e filtrada gerando um sinal de frequência estável.

Existem duas características indispensáveis para o funcionamento da síntese analógica: exatidão do sinal de saída, e estabilidade de frequência. A exatidão implica no erro do sinal de saída, fazendo com que o mesmo tenha a tendência de ir para zero com o circuito em regime permanente. Já a estabilidade de frequência, exige que o circuito se comporte como um sistema instável após estar travado (Osicom Technologies Inc,1983).

Na síntese indireta não são permitidas trocas rápidas de frequências, além disso, possuem um ruído de fase mais significativo. Todavia, possuem grande empregabilidade em sistemas de telecomunicação em geral.

2.3.3. Síntese Digital Direta (DDS)

A sintetização direta digital tem como princípio o uso de sistemas processamento de dados digitais para geração de um sinal de saída com frequência e amplitude precisamente controladas e referenciadas a uma frequência de *clock* (Analog Devices,1999).

Está técnica de sintetização é de grande valia por ter a capacidade de resolução de micro *hertz* na frequência de saída. É considerada extremamente rápida na transição de frequências, não apresentando *overshoots*, diferenças de tempo e elimina a necessidade de sintonias manuais e ajustes associados à variação de parâmetros, condições essas que são típicas nos circuitos analógicos. A Figura 2-6 retrata o diagrama de blocos do sintetizador digital:

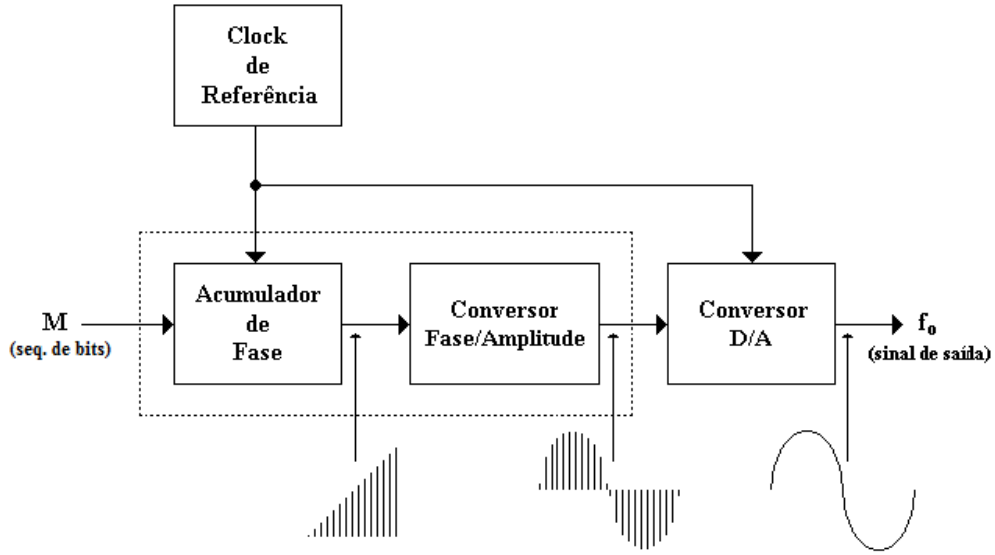


Figura 2-6 - Configuração básica de um sintetizador DDS

Para uma sintetização digital faz-se necessário o uso de um somador e um registrador de fase. O somador é utilizado para a realização da soma da sequência de *bits*, ou palavra, de ajuste de frequência com a saída do registrador de fase, que também já armazena o resultado dessa soma. Após a realização da soma, é usada a parte mais significativa dos *bits* resultantes para o endereçamento da tabela onde está definida a forma de onda desejada. A cada ciclo de *clock* é efetuada uma nova soma e um novo acesso à tabela, porém para que se construa um ciclo completo da forma de onda a ser gerada são necessários 2^n ciclos de *clock*, divididos pelo número da palavra de ajuste. O cálculo da frequência de saída do sintetizador é expressado pela equação abaixo:

$$f_{saida} = \frac{M \times f_{clk}}{2^n} \qquad \text{Equação (2.3)}$$

Onde:

f_{saida} = frequência da saída (Hz);

M – sequência de *bits*;

f_{clk} = frequência de *clock* do sistema (Hz);

n = é o tamanho da sequência de *bits* de ajuste.

De acordo com a equação 2.3, o fator determinante para a frequência de saída (f_{saida}) do sintetizador é a palavra de ajuste M, pois em um DDS o valor de 2^n e a frequência de *clock* são fixas.

Na Figura 2-7 têm-se os pontos que formarão a função de saída para o sintetizador DDS (Analog Devices,1999):

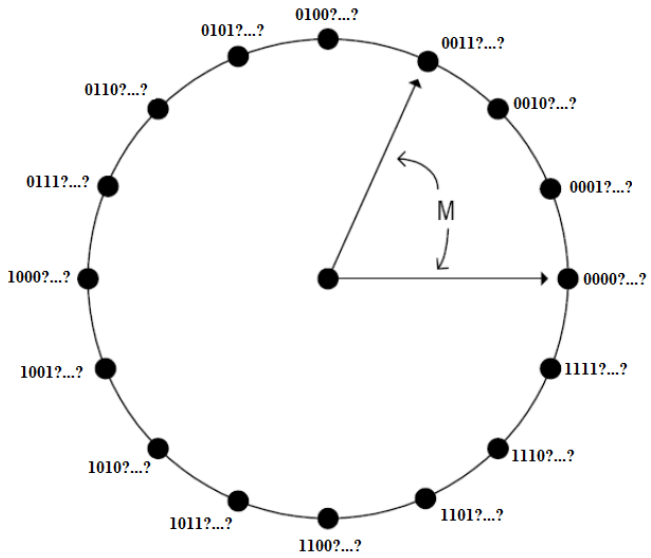


Figura 2-7 - Rodas das fases

No caso de o sintetizador estar gerando uma senóide, cada um dos pontos da roda de fases corresponde a um *step* da forma de onda que será amostrada, sendo assim, um ciclo completo da roda de fases corresponde a uma senóide gerada pelo sintetizador. Os 4 *bits* mais significativos da sequência de *bits* M fornecem uma senóide com 16 *steps*, considerando-se, por exemplo, que o DDS seja de 8 *bits*, a roda de fases dará uma volta completa a cada 256 ciclos de *clock* de referência (2^8 = 256). Dessa forma, para cada volta da roda de fases, corresponderá a um ciclo completo do sinal de saída, a partir disso tem-se a frequência desejada.

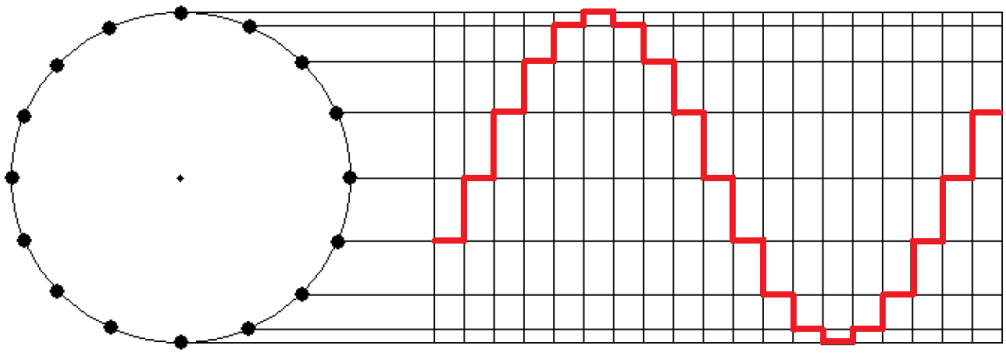


Figura 2-8 - Roda das fases formando a senóide (Analog Devices, 1999).

Outro fator a ser considerado é que o tamanho da tabela onde estão contidos os pontos é determinante para a qualidade do sinal a ser gerado, ou seja, quanto maior for a quantidade de informações, melhor será a qualidade do sinal, pois o ruído será cada vez menor à medida que os pontos de armazenamento são aumentados. Após essa etapa de construção do sinal desejado, deve ser feita a conversão digital-analógico do sinal a ser gerado.

2.3.3.1. Efeitos da Amostragem de Sinal na DDS

O processamento digital na DDS e sua amostragem podem causar respostas indesejadas na frequência de saída do dispositivo, como ruídos espectrais e *jitter,* que é o deslocamento dinâmico dos limites dos sinais digitais em relação a sua posição média.

O desempenho do *clock* de referência irá intervir diretamente na performance do *clock* sintetizado de saída de um dispositivo DDS, sendo que sinais espúrios e ruídos no *clock* de referência são transferidos para a saída do DDS com a mesma relação de síntese. Aspectos de qualidade importantes como a estabilidade da frequência (em Hz), *jitter* de borda (dado em *n*s) e ruído de fase devem ser considerados na escolha do gerador de *clock* de referência f_{clk} (Analog Devices,1999).

Os sinais espúrios ocorrem devido ao truncamento de *bits* necessário durante a etapa de geração de fase no DDS e ao algoritmo utilizado para executar a transformação e escrita do sinal. Estes sinais são apresentados na saída do circuito, mas o ruído do sinal de fase excessivo acaba mascarando a sua presença. Além disso, a truncagem de fase tende a introduzir mais erros no sistema, sendo que, o erro é caracterizado pelo comportamento da truncagem do circuito. Dessa forma, para taxas que exigem mais *bits* do que os disponíveis, o erro resultante da truncagem do ruído de fase origina então espúrias no espectro, e sua magnitude e a distribuição dependem da palavra digital que está sendo convertida.

Existem outros elementos que podem influenciar diretamente na amostragem do sinal. O ruído da fonte de alimentação é potencialmente um causador de ruídos espúrios no circuito, e o *layout* da placa que pode vir a sofrer interferência de circuitos periféricos. Contudo, é possível minimizar a influência de tais elementos através de um bom dimensionamento do circuito, de forma a garantir um bom isolamento entre os sinais analógicos e digitais, além de um bom desacoplamento do ruído de alimentação, e outros circuitos.

O ruído e as amplitudes de saída do DDS podem ser reduzidos quando se diminui ou se divide seu fator da palavra de ajuste. O ruído de fase do *clock* de referência pode ser calculado pela seguinte equação:

$$dBc = -20\log\left(\frac{f_{clk}}{f_{saida}}\right) \qquad \text{Equação (2.4)}$$

Isso mostra que, quanto maior a frequência de *clock* em relação a frequência de saída, menor será o ruído de fase provindo do cristal. Por exemplo, para se gerar um sinal sintetizado de 10MHz usando um *clock* de referência de 100MHz, o ruído de fase será atenuado em 20dB na saída. O ruído de fase absoluto permanece o mesmo, o que diminui é o ruído de fase relativo à frequência da onda de saída. Esse princípio pode ser visualizado na figura abaixo (Analog Devices,1999).

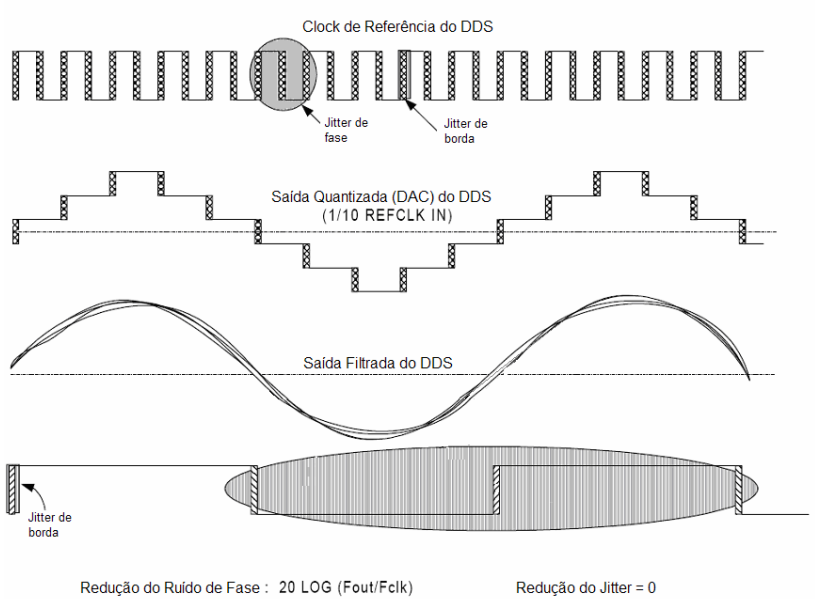


Figura 2-9 - Redução do ruído de fase após sintetização pelo DDS

Como a onda senoidal gerada pelo sintetizador de DDS não é pura, existem harmônicas da frequência fundamental, portanto, existe a necessidade de se filtrar o sinal de saída do dispositivo. As magnitudes das frequências harmônicas dependem da relação entre a frequência de referência e a frequência gerada. Quanto maior for está relação, menores serão as magnitudes das frequências harmônicas.

A Figura 2-10 exemplifica a diferença entre um sinal puro e um amostrado (Analog Devices,1999):

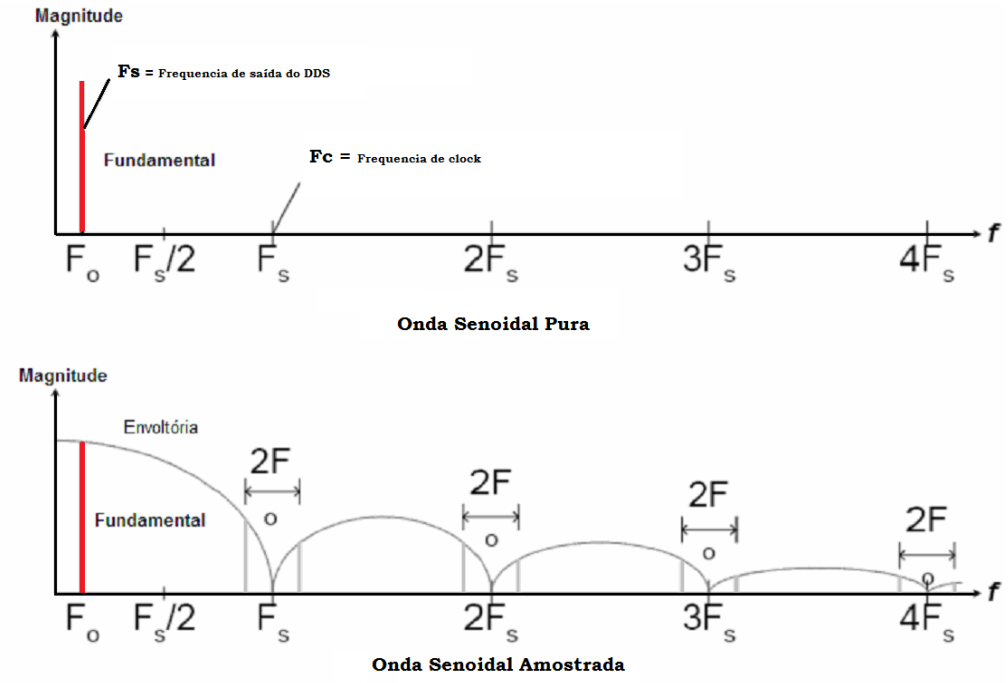


Figura 2-10 - Frequências pura e amostrada

A Figura 2-11 mostra a aplicação de um filtro passa-baixa utilizado para eliminar as frequências harmônicas (Analog Devices,1999).

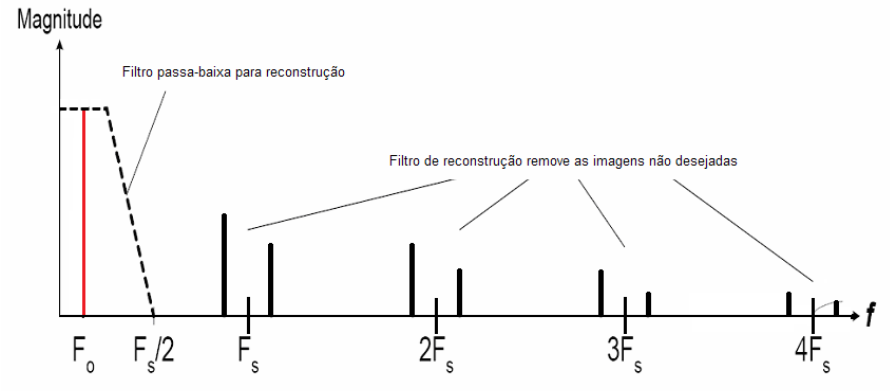


Figura 2-11 - Aplicação do filtro no sinal sintetizado

A saída do sintetizador é um seno amostrado contendo várias componentes de frequências não desejadas que formam o *jitter*, no caso de não haver o filtro de saída do circuito. Sua quantidade será igual a um ciclo de *clock* do sinal de entrada. Entretanto, ao se filtrar o sinal muitos desses sinais indesejados podem ser atenuados da onda sintetizada e pode-se obter uma onda senoidal mais pura.

2.4. Interpolação

A interpolação é o método que permite construir um novo conjunto de dados a partir de um conjunto de dados pontuais previamente fornecidos,

permitindo fazer a reconstituição aproximada de uma função apenas conhecendo algumas de suas abscissas e respectivas ordenadas. Assim, a função resultante passa nos pontos fornecidos usando como pontos intermediários os resultados obtidos no cálculo de obtenção dos mesmos.

A aproximação de funções é uma das ideias mais antigas da análise numérica e ainda uma das mais usadas. É bastante fácil entender por que razão isso acontece. Os polinômios são facilmente computáveis, suas derivadas e integrais são novamente polinômios, suas raízes podem ser encontradas com relativa facilidade, etc. (Dálcidio,2000).

Existem diversas formas de interpolar ordenadas a fim de se descobrir os valores para pontos intermediários em uma reta, como por exemplo, interpolação polinomial que é aplicada quando a função interpoladora é um polinômio e a interpolação quadrática que opera usando polinômios de segunda ordem. Entretanto, como este trabalho emprega somente o método da interpolação linear, será descrito abaixo somente o detalhamento matemático do mesmo.

2.4.1. Interpolação Linear

A interpolação linear é o método que traça um segmento de reta entre dois pontos consecutivos em um plano de coordenadas, utilizando-se de uma função linear com um polinômio de primeiro grau. Desse modo, esse método faz a aproximação linear de uma função qualquer que originalmente está representando as coordenadas em um intervalo descontínuo. (Dálcidio,2000).

A Figura 2-6 apresenta o gráfico da interpolação linear:

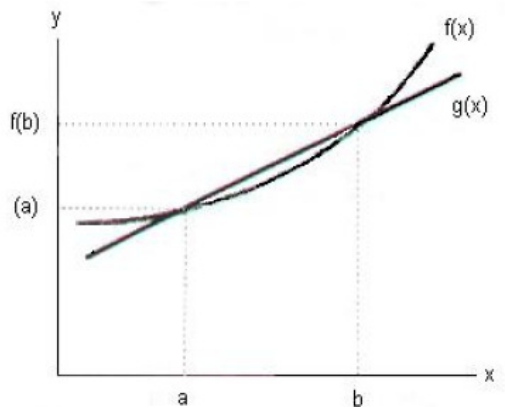


Figura 2-12 – Interpolação linear

O método da interpolação linear é empregado quando se dispõe somente dos valores das coordenadas do intervalo no plano e dos valores correspondentes ao

caminho ao qual a reta a ser gerada deve percorrer. Como, nesse caso, não se conhece a expressão matemática da função *f(x)* que se deseja representar, aplica-se o interpolador de primeiro grau para que possa ter uma função contínua que consequentemente interceptará todos os valores de coordenadas fornecidos. A equação geral da reta é definida como:

$$f(x) = ax + b \qquad \text{Equação (2.5)}$$

Onde:

a e b são valores conhecidos e constantes.

Usando a equação acima, deve-se substituir a função *g(x)* entre dois pontos, geralmente conhecidos como a e b, pelo polinômio de interpolação *f(x)*, tal que *g(a)* = *f(a)* e *g(b)* = *f(b)*. A fórmula geral para o cálculo da interpolação linear é apresentada na equação 2.6:

$$g(x) = \frac{b-x}{b-a} f(a) + \frac{x-a}{b-a} f(b) \qquad \text{Equação (2.6)}$$

Onde:

f(a) e *f(b)* são conhecidos como valores de *f(x)* em $x = a$ e $x = b$ respectivamente.

Para todos os valores calculados deve-se levar em consideração o erro associado aos mesmos. Isso ocorre devido ao fato de que os valores calculados são aproximados, pois na maioria dos casos não se conhece a equação que rege os valores calculados através da interpolação. A Equação 2.9 expressa o erro de interpolação:

$$e(x) = \frac{1}{2}(x-a)(x-b) f''(\xi) \qquad \text{Equação (2.7)}$$

Onde:

ξ é dependente de x e deve estar compreendido entre os valores de a e b.

2.5. Microcontrolador LPC 2378 ARM7

A tecnologia ARM foi orginalmente desenvolvida na empresa Acorn Computer Limited de Cambrige, Inglaterra, entre 1983 e 1985. Foi o primeiro processador com tecnologia RISC (Computador de Conjunto Reduzido de Instruções) a ser desenvolvido. Em 1990, a ARM Limited foi estabelecida como uma

empresa separada especificamente para ampliar a exploração da tecnologia ARM. Após o seu licenciamento, este processador tornou-se um líder de mercado, por possuir baixa potência e custo acessível. A Figura 2-13 apresenta a previsão de uso dos principais processadores encontrados no mercado durante os próximos dois anos:

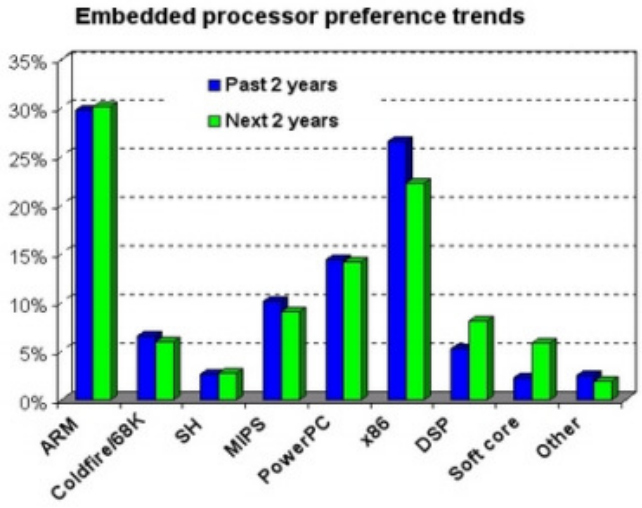


Figura 2-13 - Plataformas mais utilizadas e previsões de uso
Fonte: www.linuxdevices.com

Atualmente, os processadores ARM são muito utilizados em novos projetos eletrônicos. Sendo concebido para ser um processador simples, possuí uma grande versatilidade e seu desenvolvimento contempla obter o melhor nível de desempenho possível. Além disso, possuem baixo consumo de energia. Com uma tecnologia que manipula em memória os seus periféricos de entrada ou saída, o processadores ARM possuem um conjunto de instruções abertos.

O microcontrolador LPC2378 da Philips baseado no *core* tipo ARM7-TDMI trabalha com 32 *bits* com e *pipeline* de 3 estágios. Pode operar com frequências acima de 72MHz, tendo a capacidade de executar dois conjuntos de instruções: ARM ou *Thumb*. No modo ARM todas as instruções tem 4 *bytes*, já nos conjuntos de instruções *Thumb* a maioria das instruções é de 2 *bytes*. A Figura 2-14 apresenta a estrutura interna do microcontrolador LPC2378:

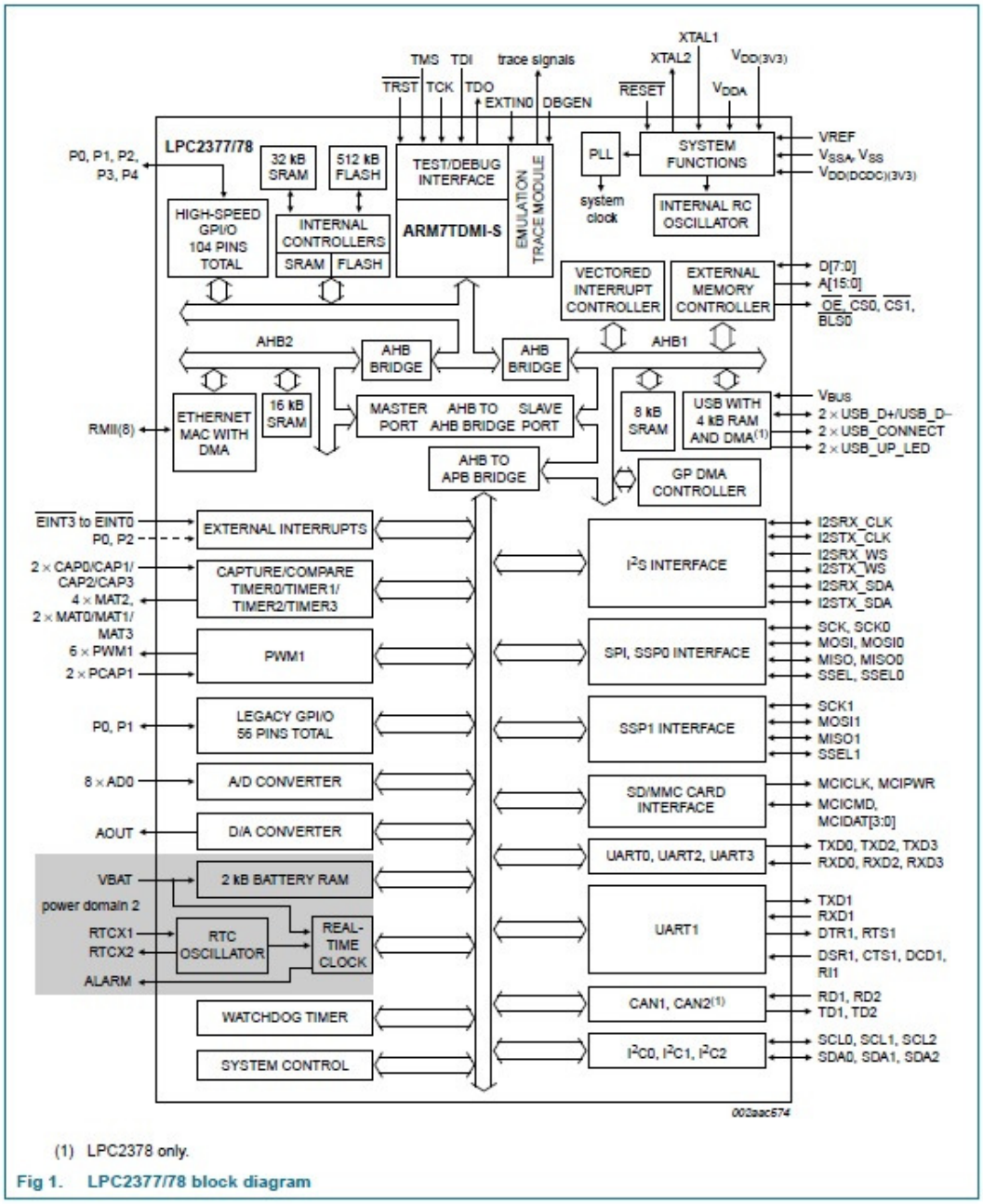


Fig 1. LPC2377/78 block diagram

Figura 2-14 - Arquitetura do processador LPC2378.
Fonte: http://www.nxp.com/documents/data_sheet/LPC2377_78.pdf

Esse chip possui uma memória flash de programação de 512kB nos sistemas ISP, 32kB de SRAM sobre o barramento ARM diretamente para permitir à CPU alta performance, 16kB de RAM estática para a interface Ethernet, 8kB de RAM estática para a interface USB, sistema duplo AHB para atuação simultânea do USB DMA e do Ethernet DMA, controle avançado para interrupção de até 32 vetores, interface serial, interface Ethernet MAC com controle DMA associado, interface USB 2.0, porta direta para PHY, dois canais de interface CAN, controle SPI, possui interface para acréscimo de cartão SD, pinos de entradas e saídas

configuráveis, 8 entradas conversoras de analógico para digital de 10 bits, quatro entradas de contagem rápida, um bloco para PWM/Timer, relógio em tempo real com pinos separados, temporizador do tipo *watchdog*, alimentação de 3,0 a 3,6 Volts, cristal de 1 a 24MHz, oscilador interno, além de muitas outras funções (NXP, 2011).

2.5.1. Mapa de Memória do ARM7

O mapeamento de memória do LPC2378 é divido entre a memória *Flash*, que é não volátil e SRAM volátil. A memória *flash* possuí uma capacidade de 512kB e pode ser usada tanto para códigos quanto para armazenamento de dados, já a memória SRAM possuí 32*kbytes*, podendo ser acessada através de 8, 16 ou 32 bits. Analisando-a pode observar que, acima dos endereços da memória *flash*, estão definidos os endereços da memória SRAM, e por último os endereços destinados a interrupções e periféricos.

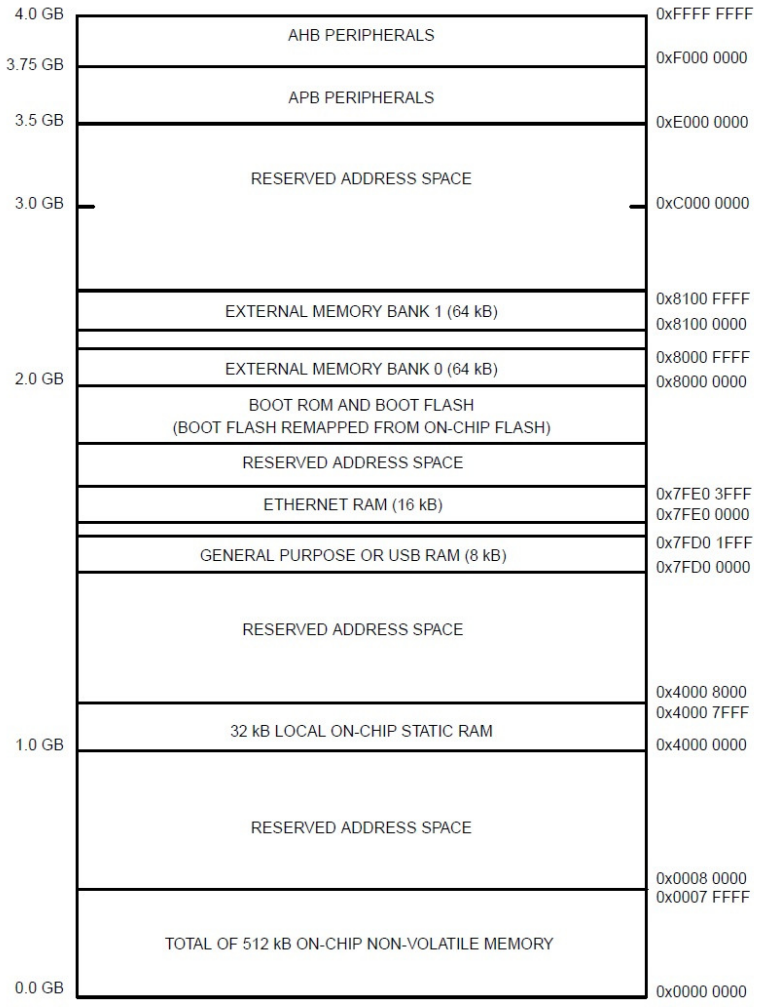


Figura 2-15 - Mapeamento de memória do LPC2378
Fonte: http://www.nxp.com/documents/data_sheet/LPC2377_78.pdf

2.6. Conversor Digital para Analógico (DAC)

A necessidade de se enviar dados para o mundo real a partir de um meio eletrônico é cada vez mais constante. A cada dia aumenta a necessidade de se controlar as variáveis analógicas fazendo a decodificação de um sinal formado por uma sequência de bits. Para que isso aconteça faz-se necessário o emprego dos conversores digitais – analógicos, conhecidos também como DAC.

Com os avanços da tecnologia muitas informações passaram a ser geradas por meios eletrônicos usando os processamentos de dados. Porém a grande maioria das grandezas físicas são analógicas por natureza, e essa necessidade de ligação entre o digital e o analógico faz com que sinais compostos basicamente de valores discretos no tempo e na amplitude sejam transformados em sinais contínuos que possuam variação ao longo do tempo.

Para se formar um sinal analógico a partir de um dado digital deve-se primeiramente obter a resolução ou tamanho do degrau do conversor digital – analógico empregado. A resolução é definida como a menor alteração que pode ocorrer na saída analógica como resultado de uma mudança na entrada digital, sendo sempre igual ao peso do *bit* menos significativo do conversor, pois o nível lógico da saída analógica mudará conforme a alteração do valor digital da entrada de um degrau para o próximo. Assim, o nível analógico de saída será modificado a cada incremento do valor digital na entrada do conversor. A Figura 2-16 apresenta um conversor D/A com resolução de 1V operando um contador de 4 bits na entrada sendo incrementado a cada pulso de *clock* de referência (Tocci & Widmer, 1998):

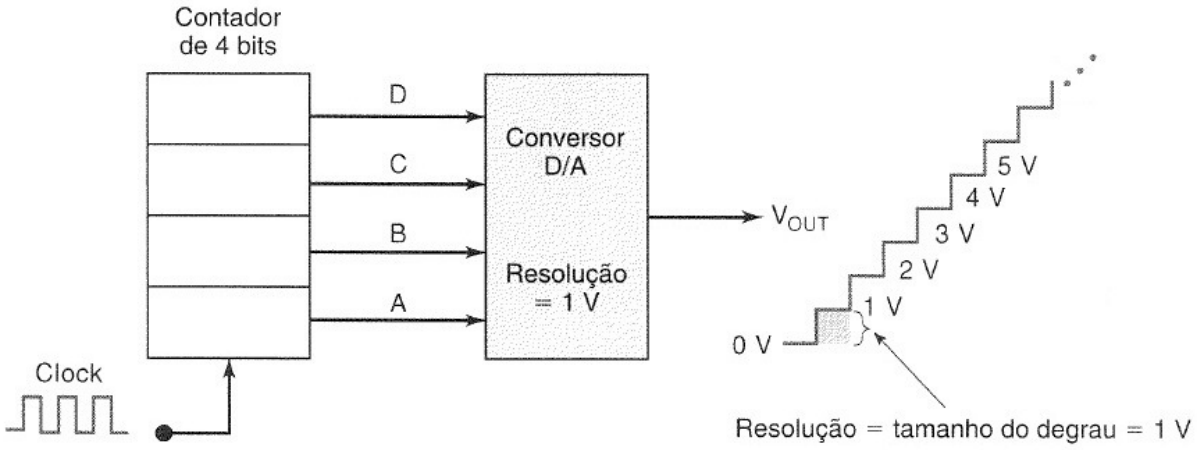


Figura 2-16 – Conversor D/A de 4 bits com resolução de 1V.

Além da resolução existem outros fatores importantes que devem ser levados em consideração para o funcionamento do conversor digital/analógico:

- Faixa dinâmica: é a faixa de amplitude de operação do sinal analógico dentro da região de operação do conversor. O sinal de entrada deve ser condicionado de forma a possibilitar sua máxima utilização dentro dessa faixa dinâmica;
- Tempo de conversão: é o tempo necessário para obter-se o valor na saída do circuito, a partir do momento em que o sinal de entrada foi aplicado e iniciado o processo de conversão. Depende da estrutura do circuito utilizado e da sua resolução. De modo geral, quanto maior a resolução, maior o tempo de conversão. O tempo de conversão é necessário para definir a máxima frequência possível a ser convertida a partir de um sinal de entrada variante no tempo;
- Pico Transitório – É observado na saída do DAC e ocorre quando mais do que um bit muda simultaneamente no código de entrada. Está associado aos atrasos de comutação de nível lógico alto para nível lógico baixo e vice-versa para os diferentes *bit's*;
- Linearidade: expressa o desvio do resultado de conversão de uma reta ideal. A linearidade depende principalmente da precisão dos resistores empregados no conversor, sendo influenciada pela precisão dos mesmos. Podendo ser afetada por mudanças substanciais de temperatura;
- Precisão: É a medida da diferença entre o nível real o ideal de tensão analógico obtido na saída do conversor, podendo também ser afetada diretamente pela falta de linearidade do circuito;

Os dois modos mais comuns de se especificar a precisão em um conversor digital-analógico são através do erro de fundo de escala e erro de linearidade, que normalmente são expressos com uma porcentagem da saída de fundo de escala. O erro de linearidade é o máximo desvio em tamanho do degrau do tamanho de degrau ideal, enquanto que, o erro de fundo de escala é o máximo desvio da saída do conversor em relação ao valor ideal, expresso como uma porcentagem do fundo de escala (Tocci & Widmer, 1998).

Deve-se obervar que a resolução e a precisão do DAC devem ser compatíveis, sendo ilógico haver uma discrepância relevante entre essas duas características.

2.6.1. Conversão DAC através da rede R-2R

A conversão digital/analógica através de resistores é largamente utilizado, pois está implementação necessita somente de componentes com valores de resistência na ordem de grandeza de 2 para 1. Este modelo de conversor recebe esse nome porque forma uma rede com valores intercalados de uma resistência (R) e o dobro dessa resistência (2R) para cada *bit* a ser convertido. Tendo a capacidade de fazer uma conversão de sinal de forma eficiente e precisa, não afetando o comportamento do circuito de modo geral. A Figura 2-16 – Conversor D/A de 4 *bits* com resolução de 1V. A Figura 2-17 mostra o conversor DAC R-2R para 8 *bits*.

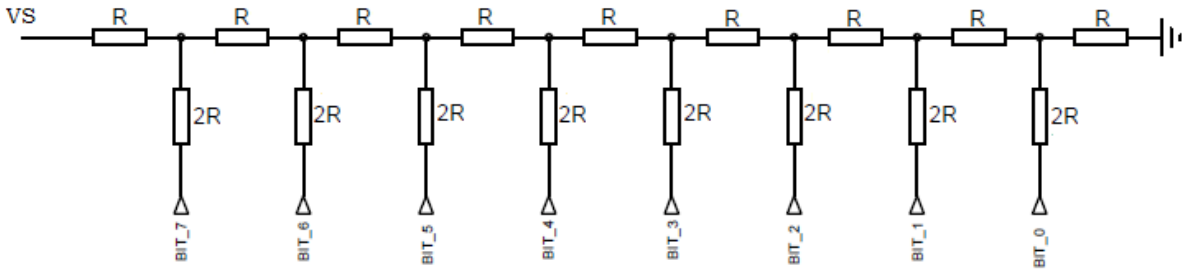


Figura 2-17 - Conversor DAC para 8 *bits* com topologia R-2R

Como esse tipo de conversão é feito de maneira escalar, pode-se aplicar o principio da superposição para o cálculo do valor de saida Vs, pois a tensão de saída de caba *bit* é independente dos outros *bits*. Nesse caso, a tensão total de saída é calculada somando-se todos os bits que estiverem em nível lógico "1" no momento da conversão. Assim, a equação para o cálculo do valor de tensão na saída do circuito é dada por:

$$Vs = V_{lógico} \times \left(\frac{VB_1}{2^1} + \frac{VB_2}{2^2} + \frac{VB_3}{2^3} + \frac{VB_4}{2^4} + \ldots + \frac{VB_n}{2^n} \right) \qquad \text{Equação (2.8)}$$

Onde:

V_S - tensão analógica saída (V);

VB_1 à VB_n - nível lógico em cada um dos bits acionados("0" ou "1");

$V_{lógico}$ - Nível lógico do circuito (V);

A resolução do conversor usando a rede R-2R é obtida a partir da quantidade total de bits do circuito. No caso de um conversor de *bits*, a resolução do mesmo é igual a 2^x, sendo X a quantidade de bits do circuito.

2.7. Comunicação Serial RS-232

A RS-232 (também conhecida por EIA RS-232C) é um padrão para troca de dados binários entre dois pontos. O equipamento que faz o processamento do sinal é chamado DTE (terminal de dados, Data Terminal Equipament), já o equipamento que faz a conexão é denominado de DCE (comunicador de dados, de *Data Comunication Equipament*). Esta comunicação é de grande empregabilidade, sendo utilizada pela maioria dos equipamentos que trabalham com comunicação de dados.

Este padrão foi originalmente usado para conectar um teletipo (equipamento eletromecânico de comunicação assíncrona que usava código ASCII) a um modem. Quando terminais eletrônicos começaram a ser usados, eram projetados para serem intercambiáveis com as "*teletypewriters*", e também suportavam RS-232. A terceira revisão deste padrão (chamada de RS-232C) foi publicada em 1969, em parte para adequar-se às características elétricas destes dispositivos. Deste modo, foi utilizado em diversos tipos de comunicação remota, especialmente por modems. Posteriormente, outros equipamentos começaram a utilizar este padrão para comunicação com equipamentos já existentes. Quando a IBM lançou computadores com uma porta RS-232, esta interface tornou-se realmente onipresente. Por muitos anos o padrão para comunicação serial em quase todos os computadores era algum tipo de porta RS-232. Continuou sendo utilizado em grande escala até o fim dos anos 90. Durante este tempo esta foi a maneira padrão para a conexão de modems (Canzian,1997).

2.7.1. Padronização do Padrão de Comunicação

Com a necessidade de se criar um padrão para as diversas formas de comunicação existentes no mercado, a EIA estabeleceu um padrão para a RS-232-C. A EIA (Aliança das Indústrias Eletrônicas, Eletronic Industries Alliance) é uma parceria de associações de empresas do ramo eletrônico, alta tecnologia, e empresas cuja a missão é promover o desenvolvimento do mercado e competitividades da indústria. Assim, em 1969 ficou definido que:

- Características mecânicas da interface, conectores "plugáveis" e identificação dos pinos;
- Características elétricas como níveis de tensão, taxa de sinalização, taxa de rotação de sinais, nível máximo de tensão, comportamento de curto-circuito e carga máxima de capacitância;

- Funções de cada circuito no conector de interface;
- Subconjuntos padrões de circuitos de interface para aplicações selecionadas de telecomunicação;

Faz mais de 30 anos desde que essa padronização foi desenvolvida, a EIA publicou três modificações. A mais recente, EIA232E, foi introduzida em 1991. Ao lado da mudança de nome de RS232 para EIA232, algumas linhas de sinais foram renomeadas e várias linhas novas foram definidas. Embora tenha sofrido poucas alterações, muitos fabricantes adotaram diversas soluções mais simplificadas que tornaram impossível a simplificação da padronização proposta. As maiores dificuldades encontradas pelos usuários na utilização da interface RS232 incluem pelo menos um dos seguintes fatores (Canzian,1997):

- A ausência ou conexão errada de sinais de controle, resultam em estouro do buffer ou estouro da comunicação;
- Função incorreta de comunicação para o cabo em uso, resultam em inversão das linhas de transmissão e recepção, bem como a inversão de uma ou mais linhas de controle;

Segundo essa padronização, um equipamento que utilize esse padrão de comunicação em sua porta serial pode se comunicar com outro equipamento até uma distância máxima de 15 metros. Para o caso de comunicações de necessitem de maiores distâncias é indicado o uso de outros padrões de comunicação existentes. Além disso, os sinais variam de 3 a 15 volts positivos ou negativos, valores próximos de zero não são sinais válidos. O nível lógico um é definido por ser voltagem negativa, a condição de sinal é chamada marca e tem significado funcional de OFF (desligado). O nível lógico zero é positivo, a condição de sinal é espaço, e a função é ON (ligado). Níveis de sinal +-5, +-10, +- 12 e +-15 são vistos comumente, dependendo da fonte elétrica disponível,

2.7.2. Definição dos Sinais de Conexão

Diversos sinais são necessários para a conexão entre os dispositivos a se comunicar via RS-232, sendo que cada um desses sinais possuem uma função especifica. Os sinais de temporização de transmissão e recepção são utilizados somente quando o protocolo de transmissão utilizado for síncrono. Para protocolos assíncronos, padrão 8 bits, os sinais de temporização externos são desnecessários.

Os nomes dos sinais que implicam em uma direção como "*Transmit Data*" e "*Receive Data*", são nomeados do ponto de vista dos dispositivos DTE. Se a norma

EIA232 for seguida, estes sinais terão o mesmo nome e o mesmo número de pino do lado do DCE. Infelizmente, isto não é feito na prática pela maioria dos engenheiros, provavelmente porque em alguns casos torna-se difícil definir quem é o DTE e quem é o DCE. A figura a seguir apresenta a convenção utilizada para os sinais mais comuns. A Figura 2-18 retrata os sinais básicos para uma comunicação usando RS-232 (Canzian,1997):

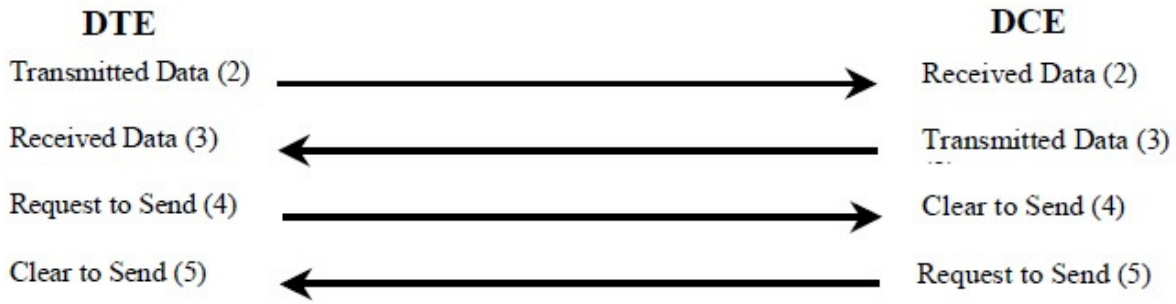


Figura 2-18 - Sinais básicos da comunicação RS-232
Fonte: http://www.professores.aedb.br/arlei/AEDB/Arquivos/rs232.pdf

Cada um dos sinais apresentados na Figura 2-18 tem uma função especifica que deve ser seguida ao se fazer uma comunicação entre dois dispositivos distintos. Os sinais devem seguir um padrão lógico e uma sequência previamente determinada, de modo que ao se fazer uma pergunta usando o sinal *Transmitted Data*, ou seja, o envio de um comando pelo DTE, logo em seguida o DCE deve enviar uma resposta ao DTE, sendo recebida como *Received Data*.

Os principais sinais necessários para a efetiva comunicação de equipamentos usando a RS-232 são apresentados na Tabela 2-1:

Tabela 2-1 - Principais Sinais do Padrão RS-232

Sinal	**Significado**
GND	Terra
TD ou TX	Transmissão de Dados
RD ou RX	Recepção de Dados
DTR	Terminal de Dados Pronto
DSR	Conjunto de Dados Pronto
RTS	Pronto para enviar
CTS	Envie os dados

O padrão desenvolvido pela EIA especifica 20 diferentes sinais de conexão disponíveis, cada um deles com uma funcionalidade específica. Para a comunicação são utilizados conectores machos e fêmeas, há também os chamados "*null modems*" para conectar unidades utilizando-se ambas como terminais de dados. Dentre todos os tipos de conexão disponíveis, o conector comumente usado é o tipo D ou DB9, de nove pinos, sendo apresentado na figura abaixo:

Figura 2-19 - Conector DB9 fêmea e macho

2.7.3. Temporização dos Sinais

Há várias configurações de *software* para conexões seriais. As mais comuns são velocidade e bits de paridade e parada. A velocidade é a quantidade de bits por segundo transmitida de um dispositivo para outro. Taxas comuns de transmissão são 300, 1200, 2400, 9600, 19200, etc. Tipicamente ambos os dispositivos devem estar configurados com a mesma velocidade, alguns dispositivos, porém, podem ser configurados para auto-detectar a velocidade (Canzian,1997).

A paridade é utilizada para a detecção de erros nas transmissões, podendo-se anexar um bit de paridade extra a cada *byte* transmitido pela RS-232, assim um erro pode ser detectado no caso de a paridade do *byte* não coincidir com o bit de paridade. Na paridade par, o bit de paridade deve valer 0 (zero) se houver um número par de uns (1), e com paridade ímpar o bit de paridade deve valer 0 se houver um número ímpar de uns. Ou seja, o bit de paridade se ajusta para que o número total de uns seja ímpar com paridade ímpar e par com paridade par. Ao receber o *byte* o programa verifica se o número de uns coincide com a paridade estabelecida se houver diferença é solicitado o reenvio do dado. Para o caso de paridade nula não existem *bits* sendo enviados ou recebidos.

Bits de parada são enviados no fim de cada *byte* transmitido com o intuito de permitir que o receptor do sinal se sincronize.

3. MATERIAIS E MÉTODOS

3.1. Descrição Geral do Sistema

O sistema consiste em um gerador de funções capaz de gerar sinais de maneira arbitrária usando o *software* desenvolvido para o desenho dos sinais. A geração do sinal ocorre da seguinte maneira: No *software* supervisório é realizada a escolha da forma de onda a ser gerada e a sua frequência. Após a conclusão da primeira etapa, deve ser feito o envio dos dados serialmente para o microcontrolador, aonde são realizados os cálculos dos pontos necessários para a formação do sinal através do algoritmo desenvolvido.

Após a comunicação com o microcontrolador o sinal a ser gerado passa a ser tratado diretamente pelo *hardware*, não dependendo mais do *software* de geração. Os pontos gerados na formação do sinal, são armazenados em memórias externas, e a partir delas são enviados para o circuito de conversão e condicionamento do sinal. A Figura 3-1 retrata a diagrama de blocos do gerador de funções:

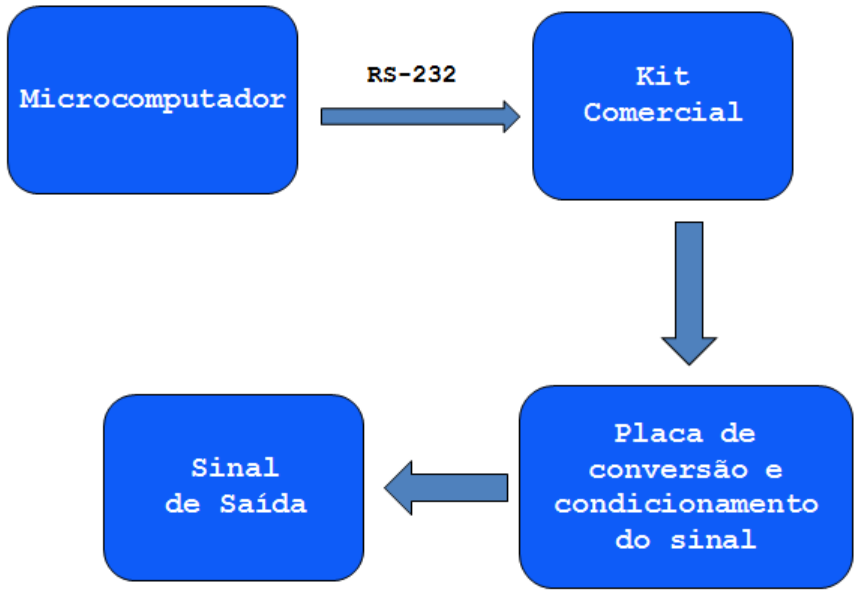


Figura 3-1- Diagrama de blocos do gerador de funções

3.2. *Descrição do Hardware*

O *hardware* é constituído pelas seguintes partes:

- Circuito Microcontrolado;
- Armazenamento do sinal usando memória externa;
- Conversor Digital – Analógico;
- Circuito de condicionamento do sinal a ser gerado;

3.2.1. Kit de Desenvolvimento

Para a implementação do circuito microcontrolado e interface entre o software desenvolvido e placa de condicionamento foi escolhido como plataforma um kit de desenvolvimento baseado em um microcontrolador LPC2378. A Figura 3-2 apresenta o kit de desenvolvimento e a disposição dos conectores:

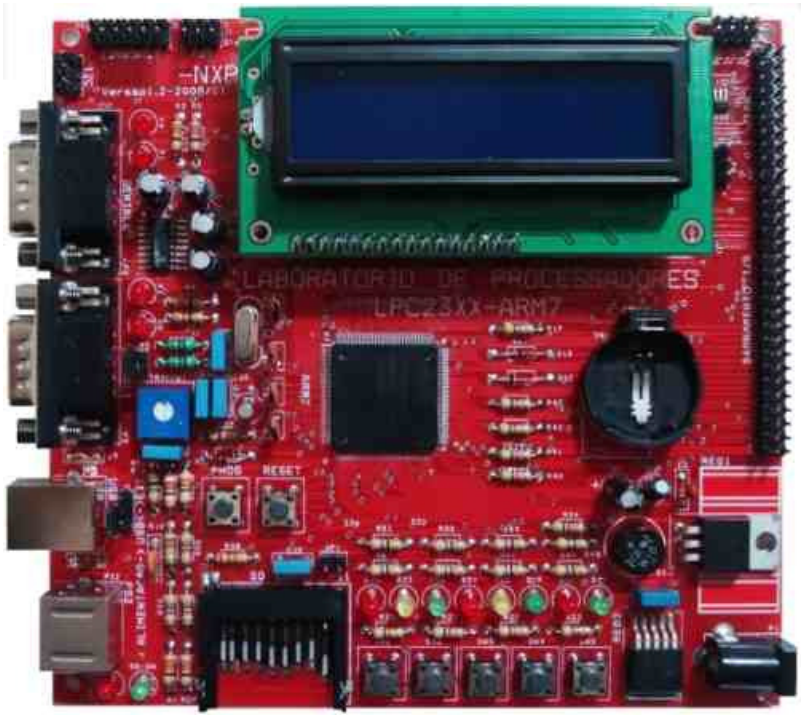


Figura 3-2 - Placa do kit de desenvolvimento.

Esse kit, além de possuir como diferencial um microcontrolador com grande capacidade de processamento e diversas funcionalidades encapsuladas, possui ainda outros periféricos que podem ser acessados pelo usuário. Podendo-se destacar as seguintes funcionalidades:

- Alimentação pela USB ou fonte externa;
- Sensor de temperatura I2C;
- Um conector PS/2;

- Duas portas seriais;
- Uma porta USB *device*;
- Um conector *Smart Card* (cartão incluído);
- Um conector SD *Card* (cartão não incluído);
- Um display de LCD 16x2 fundo azul (não incluído);
- Teclado com cinco botões para utilização do usuário;
- Soquete para pilha CR2032 para o RTC;
- Oito LEDs para utilização do usuário;
- Conector para bateria CR2032 (incluída);
- Diversas barras de pinos com o restante dos periféricos conectados;

No Anexo A é apresentado o esquemático do Kit de desenvolvimento.

3.2.2. Acesso a Memória Externa

Devido à necessidade de se obter uma boa performance, foram usadas memórias externas para o armazenamento dos pontos a serem fornecidos durante a formação do sinal a ser gerado. Dessa forma, após os cálculos dos pontos do sinal, é feito o envio para a memória externa, não exigindo mais toda a capacidade de processamento do microcontrolador. A memória escolhida para a implementação foi a CY7C185 de 8*kbytes* que possuí um alto desempenho, podendo trabalhar com uma velocidade de até 15 ns. A Figura 3-3 retrata o diagrama de blocos da memória CY7C185.

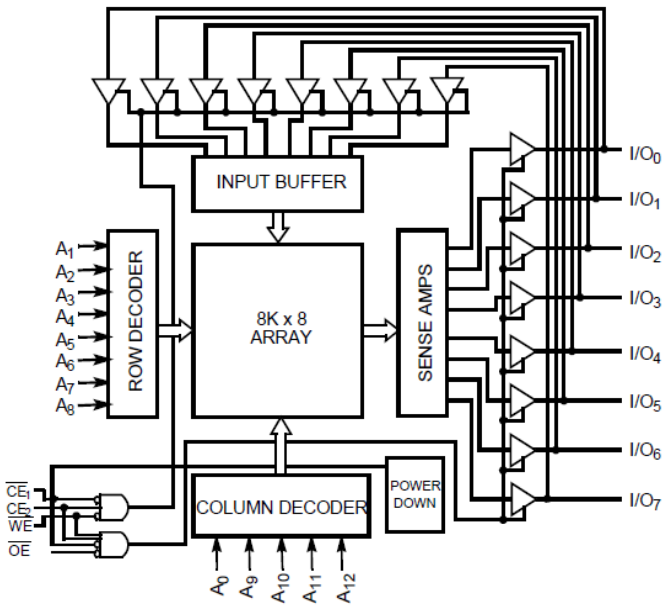


Figura 3-3 - Diagrama de blocos da SRAM CY7C185
Fonte: http://www.cypress.com/?docID=25678

Está memória é uma SRAM (*Static Random Access Memory*), tendo a capacidade de manter os dados armazenados desde que seja mantida sua alimentação. Funciona no modo de escrita quando o pino WE (*write enable*) está em nível lógico baixo e o pino OE (*output enable*) em nível lógico alto. Para o modo de leitura, os sinais acima mencionados devem estar acionados ao contrário, ou seja, WE em nível lógico alto e OE em nível lógico baixo.

Como o circuito conversor digital/analógico utilizado possuí 16 *bits* de resolução, foi necessária a utilização de duas memórias externas em paralelo, pois as memórias escolhidas trabalham com somente 8 *bits* de dados. Desse modo, para o correto acionamento de cada uma das memórias durante o processo de escrita das mesmas, foi empregado o uso de dois *latches* usando circuitos integrados 74HCT373 para o acionamento das memórias. Assim, era feito o acesso em uma memória de cada vez.

O circuito integrado 74HCT373 é um *latch* octal que possuí dois pinos de controle independentes, além de operar em altas velocidades tem a função de deixar com que o sinal dos dados gerados chegue à memória somente após a sua habilitação, realizada pelo pino de controle chamado de LE (*latch enable*). Nesse caso, o pino OE estará ligado diretamente ao GND (nível lógico 0) para que as suas saídas estejam sempre habilitadas. Já o pino LE é controlado pelo *software* do microcontrolador, habilitando o seu uso somente quando for necessário o envio dos dados a memória correspondente, assim os dados contidos nos pinos de entrada (D0...D7) serão transferidos para os pinos de saída do *latch* (Q0...Q7) . A Figura 3-4 apresenta o diagrama de blocos do 74HCT373:

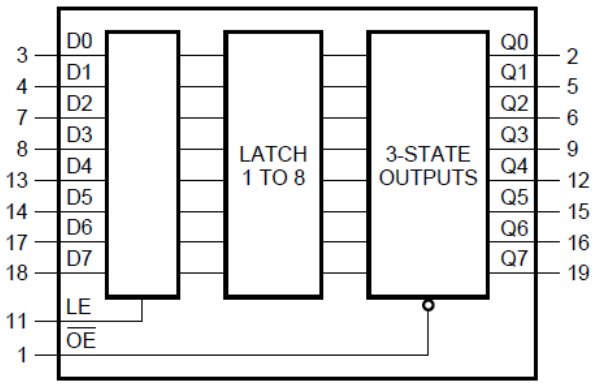


Figura 3-4 - Diagrama de blocos do CI 74HCT373
Fonte**:** http://www.nxp.com/documents/data_sheet/74HC_HCT373.pdf

3.2.5. Circuito de Endereçamento

O endereçamento da memória não é realizado pelos pinos de endereços fornecidos pelo microcontrolador. Nesse caso, para o endereçamento foi empregado o uso de um contador digital de 13 *bits*. Para a implementação deste contador, foram usados dois CI'S 74HC590 em cascata, pois os mesmos conseguem operam com velocidades de até 61MHz, possibilitando assim um endereçamento rápido. A Figura 3-5 mostra o diagrama de blocos do 74HC590:

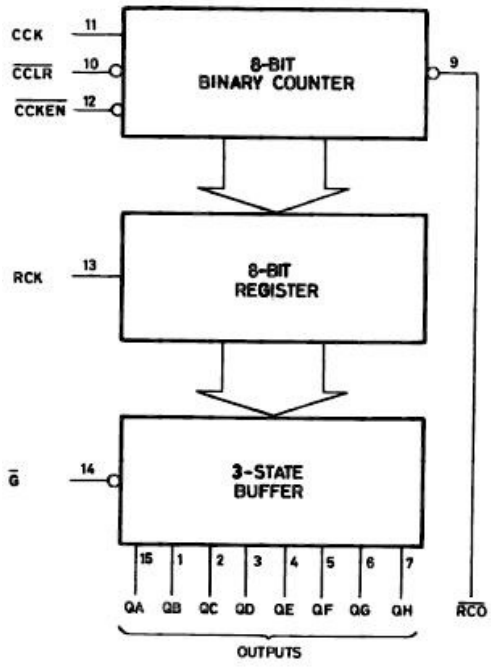


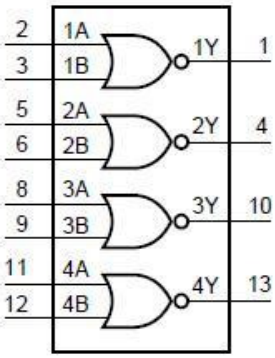
Figura 3-5 - Diagrama de blocos do 74HC590.
Fonte**:** http://www.nxp.com/documents/data_sheet/74HC590.pdf

Para realizar a contagem dos 13 *bits* necessários para o endereçamento da memória externa, foi necessária a ligação do segundo contador (U6) de 8 *bits* após a finalização da contagem do primeiro contador (U5). Para isso, foi colocada a habilitação do U6, pino CCLKEN (*Counter Clock Enable Output*), após o término de contagem pelo U5, nesse caso, o U6 inicia a contagem somente quando ocorre o acionamento do pino RCO (*ripple carry output*).

O reinício do endereçamento, é feito através da ligação do pino 5 (Q5) do U6 ao CCLR (*Counter Clean Input*) dos dois contadores, assim ao térmico da contagem dos treze *bits*, a mesma é reiniciada automaticamente. Porém, como a saída Q5 é ativa em nível lógico alto e o CCLR é ativo em nível lógico baixo, e além disso é necessário que o reset do circuito seja realizado tanto de maneira automática quanto manual, foi necessária a colocação de uma porta lógica não ou (NOR) entre os dois circuitos. Desse modo, a contagem pode ser reiniciada de forma automática,

no caso do estouro de contagem dos 13 *bits*, ou por software, aonde foi desenvolvido um circuito que ao perceber que todos os *bits* do circuito digital estão em nível lógico "1" reseta o contador.

Para fazer a inversão do sinal gerado por ambos os circuitos, foi usado o CI 74HCT02 (U4) que tem por objetivo fazer a lógica combinacional da tensão que lhe foi aplicada em um dos pinos de entrada da porta lógica. Este CI é composto por quatro portas não ou (NOR) independentes. A Figura 3-6 mostra o diagrama de blocos e a tabela verdade do CI 74HCT02:



A	B	Y
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

Figura 3-6 - Diagrama de blocos e tabela verdade do CI 74HCT02
Fonte**:** http://www.nxp.com/documents/data_sheet/74HC_HCT02.pdf

3.2.6. Circuito de Geração de RESET

Como descrito anteriormente, o reset da contagem do endereçamento pode ser realizado de maneira automática no estouro de contagem dos treze *bits* ou pelo *software* de forma intencionada.

O reinicio da contagem, no caso da forma de onda a ser gerada não necessitar de todas as posições possíveis da memória RAM, é feita através do emprego de um circuito lógico combinacional. Este circuito, tem a capacidade de enviar um pulso no caso de todos os 16 *bits* do conversor DAC estarem em nível lógico "1". Desse modo, a delimitação para reinício do endereçamento é feito através do *firmware* do LPC2378, onde sempre é inserido o delimitador com o valor 0x7FFF na próxima posição da memória após o envio do último ponto calculado para a geração do sinal. Como o *bit* mais significativo terá sempre nível lógico "0", foi colocada uma porta lógica inversora, usando o circuito integrado 74HCT04, para mudar o estado lógico do mesmo, sendo isso necessário para evitar que o delimitador seja sempre amostrado juntamente com os demais pontos do sinal gerado. Com essa implementação, é possível deixar o circuito operando de forma automática, não dependendo mais do microcontrolador para a formação do sinal a ser gerado. A Figura 3-7 apresenta 74HCT04.

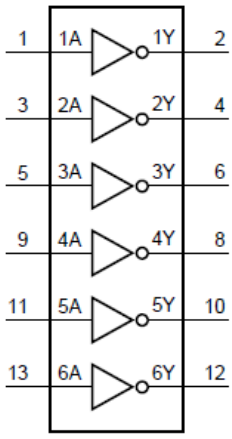


Figura 3-7 - Diagrama de blocos do CI 74HCT04
Fonte**:** http://www.nxp.com/documents/data_sheet/74HC_HCT04.pdf

O circuito integrado que faz a detecção de quando os 16 *bits* do conversor DAC estão em nível lógico "1" é o 74HCT30. Este CI é composto por uma porta lógica Não E (NAND), que realiza a multiplicação e a inversão dos oito sinais aplicados nas suas entradas. A Figura 3-8 apresenta o 74HCT30.

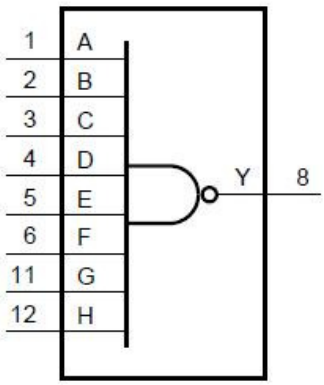
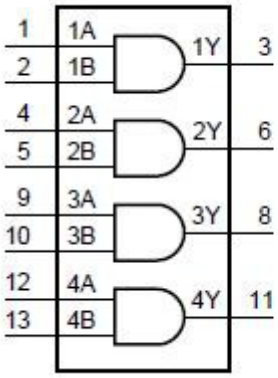


Figura 3-8 - Diagrama de blocos do CI 74HCT30
Fonte: http://www.nxp.com/documents/data_sheet/74HC_HCT30.pdf

Como é necessário que na saída do circuito de reset tenha-se nível lógico alto "1" no caso de detecção de terminação, foi usada uma porta lógica NOR (U4) na saída dos integrados 74HCT30 (U13 e U14). Desse modo, o circuito apresentará nível lógico alto em sua saída somente quando todas as suas entradas estiverem em nível lógico alto.

Todo o circuito de reset pode ser desabilitado ou habilitado a qualquer instante. O mesmo é controlado por uma porta E (AND) conectada a um pino físico do microcontrolador, e desse modo o circuito só entre em uso durante o ciclo de leitura da memória RAM, pois durante a escrita o mesmo não é necessário. Para a

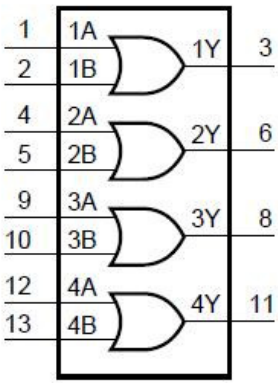
implementação da porta E foi empregado o CI 74HCT08. A Figura 3-9 mostra o CI 74HCT08 e sua tabela verdade.



A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Figura 3-9 - Diagrama de blocos e tabela verdade do CI 74HCT08
Fonte: http://www.nxp.com/documents/data_sheet/74HC_HCT08.pdf

Prevendo a possibilidade de se fazer o reset a qualquer instante durante o período de leitura ou escrita, foi empregada uma porta lógica OR usando o integrado 74HCT32 (U12) com um de seus pinos ligados diretamente em uma das saídas do microcontrolador (I02). A Figura 3-10 apresenta o 74HCT32 e sua respectiva tabela verdade.



A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Figura 3-10 - Diagrama de blocos e tabela verdade do CI 74HCT32
Fonte: http://www.nxp.com/documents/data_sheet/74HC_HCT32.pdf

A Figura 3-11 apresenta o circuito de reset do contador de endereçamento de 13 *bits*:

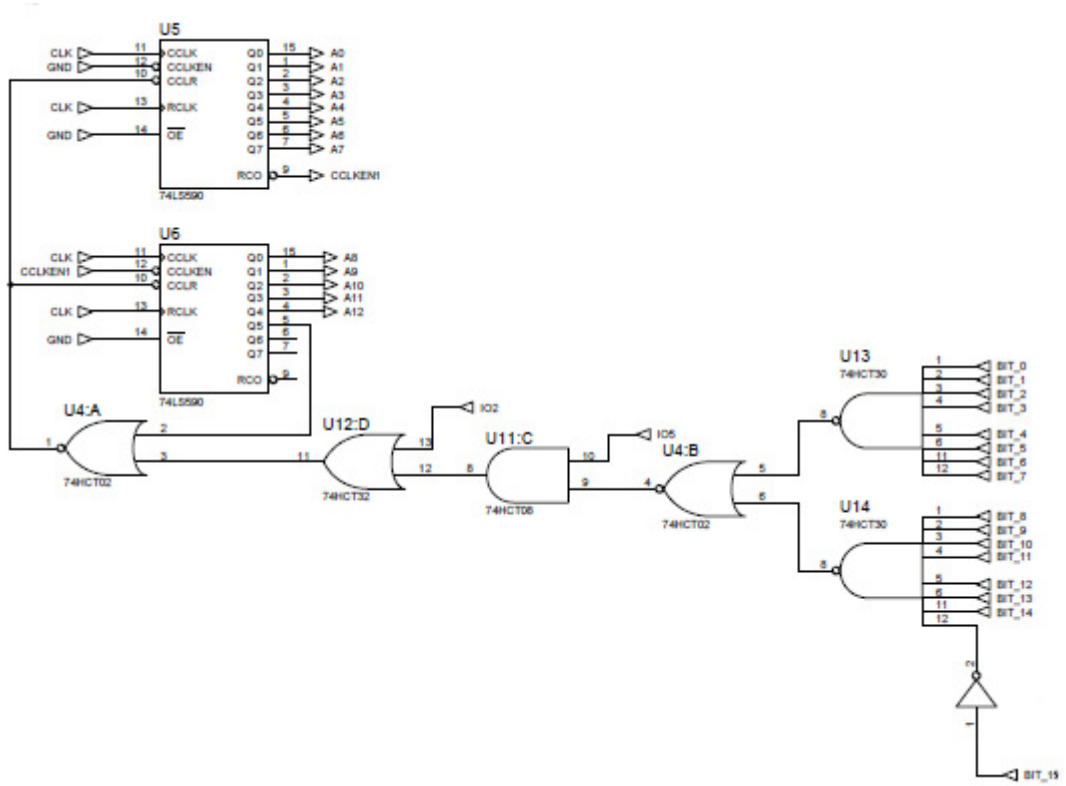


Figura 3-11 - Diagrama elétrico do circuito de endereçamento.

3.2.7. Geração de CLOCK

O circuito de geração do *clock* é constituído por dois osciladores, de 1 e 8 MHz, e portas lógicas para o controle da lógica de acionamento. Os osciladores servem para controlar a frequência de contagem do circuito de endereçamento descrito no tópico anterior. Além disso, foram empregados dessa forma devido à necessidade de se ter uma quantidade de pontos variáveis para cada forma de onda gerada. Assim, ao obter-se o valor de frequência a ser gerada, é realizado o cálculo necessário para definir a quantidade de pontos necessários para a formação da onda. O cálculo de pontos é realizado pela seguinte equação:

$$N = \frac{F_{osc}}{F_{saida}} \qquad \text{Equação (3.1)}$$

Onde:

N - número de pontos necessários para a formação do sinal;

F_{osc} - Frequência do oscilador base (MHz);

F_{saida} - Frequência do sinal a ser gerado (Hz);

Para a geração de ondas com baixa frequência, de 125Hz até 2500Hz, é necessária a utilização do oscilador de 1MHz. Isso ocorre devido ao fato que para

um sinal abaixo da frequência mencionada aumentaria o número de pontos máximos para a geração do sinal, que nesse projeto é de 8192, devido a capacidade máxima da memória SRAM.

No caso de sinais com frequências a partir de 2500Hz é usado o oscilador de 8MHz, podendo gerar sinais de 2501Hz a 1MHz.

3.2.8. Dimensionamento Conversor Digital – Analógico

Para interpretar as palavras digitais geradas durante o ciclo de leitura das memórias, foi utilizado o conversor digital para analógico de 16 *bits*, com uma resolução de 65536 posições. Foram empregados resistores de 10kR e 20kR com precisão de aproximadamente 1%. A Figura 3-12 retrata o conversor:

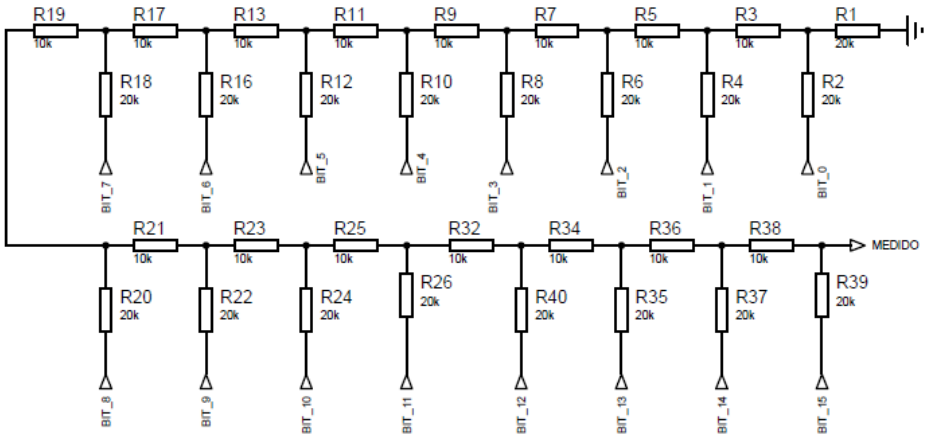


Figura 3-12 - Conversor DAC

Como o sinal de nível lógico alto fornecido pelas memórias é de aproximadamente 3,71V para cada *bit*, após a conversão será fornecido um sinal analógico variando de 0 a 3,71 V na saída do circuito.

3.2.9. Circuito Subtrator

O circuito subtrator tem a finalidade de amplificar a diferença entre as tensões nos seus terminais de entrada. Foi implementado com a função de corrigir o deslocamento atribuído ao sinal durante o cálculo dos pontos no *firmware* do microcontrolador. Esse deslocamento é necessário devido ao circuito digital operar somente com níveis de tensões positivas, assim é deslocado o nível "0" de operação para o nível correspondente a metade do nível de tensão da saída do circuito de conversão, fazendo que o sinal digital seja gerado na sua parte positiva de 0x7FFF a 0xFFFE e na sua parte negativa variando de 0x7FFF à 0x0000.

Este circuito tem a função de corrigir o deslocamento no sinal analógico atribuído no instante da conversão digital-analógica, fazendo assim com que a referência do sinal de saída seja 0V. A Figura 3-13 apresenta o circuito subtrator:

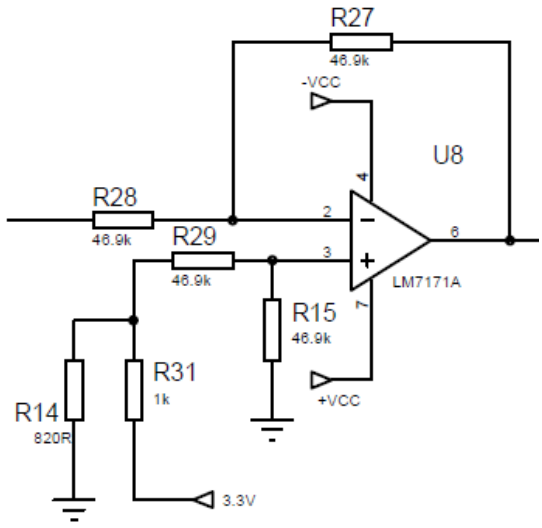


Figura 3-13 - Circuito subtrator

O cálculo da tensão de saída do circuito subtrator é dado pela seguinte equação:

$$V_{saida} = -\frac{49{,}6k\Omega}{49{,}6k\Omega}(V_2 - V_1) \qquad \text{Equação (3.3)}$$

Onde:

V_{saida} - tensão de saída do circuito (V);

$V_{1,}, V_2$ - Níveis de tensão da entrada do circuito (V);

Como o circuito possuí os resistores idênticos, o mesmo torna-se um subtrator puro, onde apenas subtrai e inverte os sinais de suas entradas.

Como a memória SRAM fornece um nível lógico alto equivalente a 3,71V, valor esse correspondente ao nível de tensão V2, foi empregado o divisor resistivo fixo, no caso V1, que descontará a metade desse valor no sinal analógico, sendo obtido pela seguinte equação:

$$V_1 = \frac{3{,}3 \times 1000}{1000 + 820} = 1{,}83V \qquad \text{Equação (3.4)}$$

O amplicador operacional escolhido para as implementações é o LM7171 da *National Semiconductor*. Este é um amplificador de realimentação de tensão de alta velocidade, que tem a característica de rotação de um amplificador de realimentação de corrente; contudo pode ser utilizado com qualquer configuração tradicional de amplificadores. O LM7171 é estável para ganhos baixos como 2 ou -1. Ele fornece uma taxa de variação muito elevada, em torno de 4100V/µs, e uma largura de banda de ganho unitário de 200MHz consumindo apenas 6,5mA de corrente de alimentação. É ideal para vídeo de alta velocidade, e aplicações de processamento, tais como HDSL e amplificadores de pulso. Com corrente de saída de 100mA, o LM7171 pode ser utilizado para distribuição de vídeo, ou como um controlador de diodo laser(National Instruments,2011) .

3.2.10. Circuito de Controle de Amplitude

A amplificação do sinal analógico é necessária para obter-se os ganhos nos sinais a serem gerados, sendo realizada pelo amplificador operacional LM7171 configurado como amplificador-inversor. Como a amplitude do sinal é de até 10 Vpp e seu controle é feito de forma manual, foi empregado o uso de um potenciômetro multivoltas de 10kR para o controle de ganho do circuito. A Figura 3-14 mostra o circuito amplificador completo.

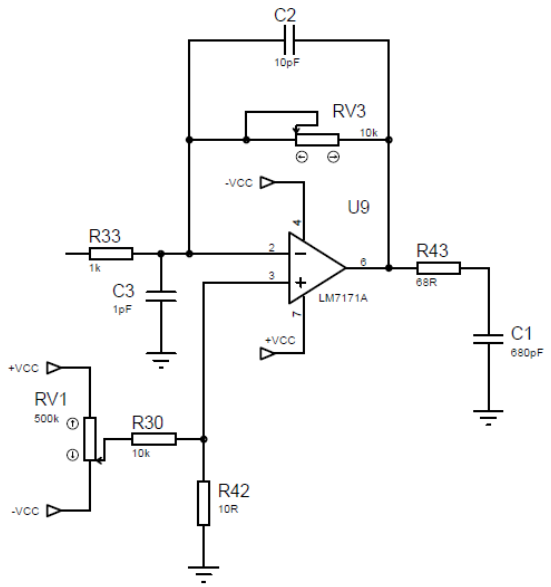


Figura 3-14 - Circuito amplificador

A equação que define a tensão de saída deste circuito é apresentada abaixo.

$$V_{saida} = -\frac{RV3 \times Vi}{1k\Omega} \pm Vcc \times \frac{10k\Omega}{10k\Omega + 10\Omega} \qquad \text{Equação (3.5)}$$

Onde:

V_{saida} - tensão de saída do circuito (V);

$RV3$ - Potenciômento de controle de ganho (ohns);

$V_{1,}$ - Níveis de tensão da entrada do circuito (V);

Vcc - Tensão simétrica de alimentação (V);

Como circuitos amplificadores apresentam constantemente erro residuais em sua configuração, foi empregado um controle de *off-set* para correção destes possíveis erros. Sendo que, os mesmos podem ser eliminados a qualquer instante através de potenciômetro de ajuste.

Devido a picos ou oscilações bruscas no sinal a ser amplificado, é recomendado pelo fabricante do amplificador operacional o emprego de capacitores com a função de cancelar os pólos gerados durante a operação do mesmo. Sendo assim, para resolver esse possível problema é aconselhado que seja colocado um capacitor como filtro na entrada do circuito e outro capacitor em paralelo com o resistor de ganho do circuito(National Instruments,2011).

Os valores dos capacitores podem ser obtidos pela seguinte equação:

$$C_f > \frac{R_f \times C_{IN}}{R_i} \qquad \text{Equação (3.6)}$$

Onde:

C_f - Capacitor de ganho (pF);

Rf - Resistor de ganho (ohns);

Ri - Resistor de ganho (ohms);

C_{IN} - Capacitor de filto (pF);

O esquemático completo da placa de armazenamento, conversão e condicionamento é apresentado no apêndice A.

3.3. Descrição dos Softwares

No decorrer desta seção serão apresentadas as características, e o funcionamento do *firmware* do microcontrolador e do desenvolvimento da interface com o usuário em Builder C++.

3.3.1. Software Supervisório

O *software* que gerencia a construção e envio dos sinais foi desenvolvido de maneira funcional, de modo que a interface entre o usuário e o microcontrolador ficasse limpa e sem controles desnecessários. Inicialmente, ao abrir o *software,* é apresentado ao usuário os campos para a colocação manual da frequência do sinal a ser gerado, da forma de onda do sinal, da porta serial disponível, e por último deve ver feita a inicialização da porta e envio dos dados, apertando os botões "Inicializa" e "Envia" respectivamente. A Figura 3-15 apresenta a tela inicial do *software* supervisório com a onda senóide selecionada:

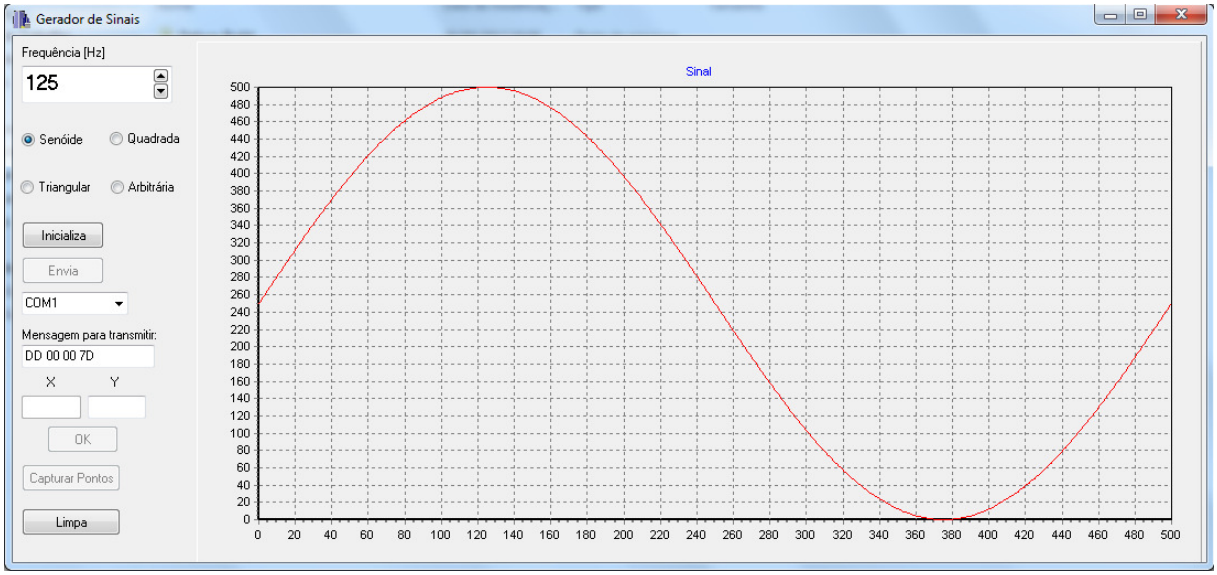


Figura 3-15 - Tela inicial do supervisório para a geração da senóide.

No caso de serem escolhidas as formas de onda "Senóide", "Quadrada" ou "Triangular", não é necessário que seja desenhada a forma de onda para que a mesma venha a ser gerada pelo hardware, pois a mesma é desenhada automaticamente. Somente deve-se escolher o sinal e a frequência desejada. Como o valor mínimo de frequência fornecida neste trabalho é de 125Hz, e seu valor máximo é de 500000, o campo correspondente à frequência já inicializa com esse valor, não sendo possível colocar valores menores que o mínimo exigido.

Se a forma de onda escolhida for a arbitrária, será apresentado um gráfico com a quantidade de pontos correspondentes a frequência anteriormente escolhida no eixo X do gráfico. Já os pontos do eixo Y permanecem sempre com os seus respectivos valores variando de 0 a 500. Assim, para geração do sinal devem ser escolhidos os pontos nas coordenadas X e Y do gráfico, de modo que em cada colocação de pontos deve ser apertado o botão "OK", fazendo com que a coordenada escolhida seja previamente mostrada. O sinal arbitrário digitado deve ser contínuo

durante todo o trajeto do gráfico. A Figura 3-16 mostra o funcionamento do *software* supervisório para a montagem de um sinal arbitrário:

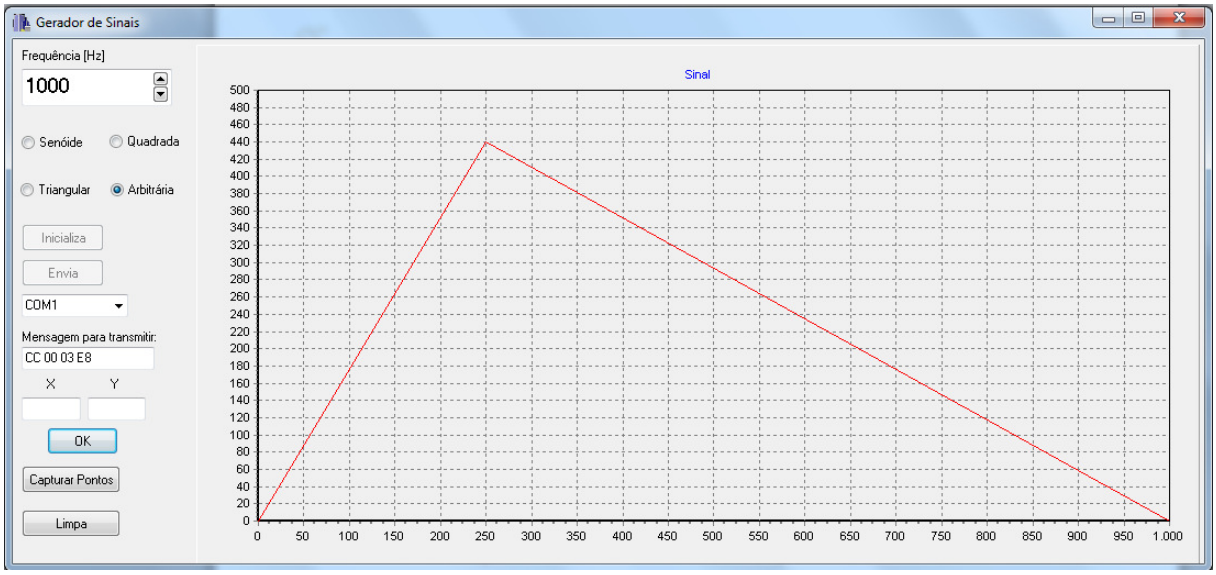


Figura 3-16 - Funcionamento supervisório durante a criação de um sinal arbitrário

Na Figura acima é possível visualizar o funcionamento do supervisório durante a criação do sinal arbitrário com a frequência de 1000Hz. Ao término da montagem do sinal, deve-se apertar no botão “Capturar Pontos”, assim serão coletados todos os valores digitados. Após a conclusão dessa etapa, deve ser feito o envio dos dados de acordo com o processo descrito anteriormente. A fim de facilitar o processo para a geração do sinal foi criado um botão com nome de “Limpa”, que zera todos os valores de coordenadas e apaga todas as opções escolhidas.

No caso de ser digitando algum ponto na coordenada X ou Y fora da faixa de operação durante a montagem do sinal, é apresentada a seguinte mensagem de erro:

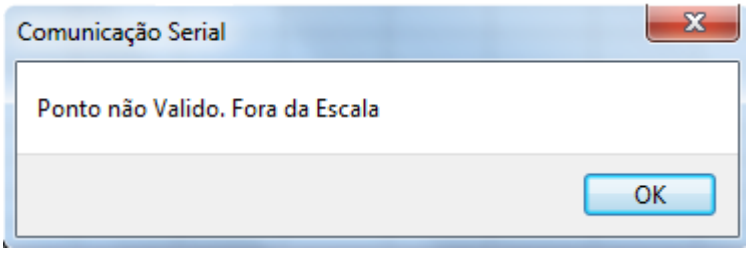


Figura 3-17 – Mensagem de erro para pontos fora da faixa

O código fonte completo é mostrado no apêndice B.

3.3.2. Protocolo de Comunicação

O protocolo de comunicação foi criado para fornecer uma segurança adicional aos dados enviados pela porta serial, de modo que antes dos *bytes* de dados serem enviados, sempre é enviado um *byte* de sincronização que por sua vez aguarda uma resposta por parte do microcontrolador e um *byte* de cabeçalho que também aguarda uma resposta para dar início ao envio dos *bytes* de dados. Após essa etapa são enviados os *bytes* de dados, podendo variar a quantidade dos mesmos de acordo com a forma de onda escolhida.

No envio dos dados das ondas senoidais, triangulares e quadradas são enviados para o microcontrolador um total de 4 *bytes* correspondentes a indicação do sinal a ser criado e do valor da frequência do sinal a ser gerado, além dos *bytes* correspondentes à sincronização e ao cabeçalho. Nesse caso é enviado o seguinte protocolo de dados:

Tabela 3-1 - Protocolo de Comunicação Para Sinais Não Arbitrários

Protocolo	Software	Microcontrolador
Inicia Transmissão	Envia 0xAA	Recebe Valor
Confirma Sincronização	Recebe Valor	Envia 0x55
Envia Cabeçalho	Envia 0xAA	Recebe Valor
Confirma Cabeçalho	Recebe Valor	Envia 0x55
***Byte* 0**	Sinal	Recebe os Dados
***Byte* 1**	Valor da Frequência	
***Byte* 2**		
***Byte* 3**		
Termina Transmissão	Recebe Valor	Envia 0xCC

O *byte* 0 corresponde ao modelo de sinal a ser gerado pelo microcontrolador, sendo que, para cada uma das quatro formas de ondas disponíveis deve ser enviado um valor diferente como parâmetro de diferenciação durante a recepção dos valores por parte do microcontrolador. Os sinais possuem os seguintes valores de codificação:

- Senóide = 0xDD;
- Triangular = 0xEE;
- Quadrada = 0xAA;
- Arbitrária = 0xCC;

Como ocorre uma diferenciação na quantidade de dados entre o envio dos sinais regulares e dos sinais arbitrários, após o envio dos quatro *bytes* de dados é feita a consistência no código fonte do microcontrolador para saber qual é o tipo de sinal que está sendo enviado pelo supervisório. Assim, se o sinal for arbitrário a

comunicação não deve encerrar após a recepção dos primeiros *bytes* mencionados anteriormente, pois devem ser enviados ainda os valores correspondentes às coordenadas capturadas pelo *software* supervisório.

Nessa etapa de envio, o primeiro *byte* corresponde a quantidade de dados que serão enviadas logo a seguir, dessa maneira é possível tornar todo o processo de comunicação síncrono, evitando que o mesmo venha a trancar durante o envio de dados pela serial. Os próximos *bytes* são os valores das coordenadas dos pontos X e Y anteriormente coletadas, sendo os valores são subdivididos em valores de coordenada X e valores de coordenada Y, com cada ponto possuindo 4 *bytes* de tamanho.

Tabela 3-2 - Protocolo de Comunicação Para Sinais Arbitrários

Protocolo	Software	Microcontrolador
Inicia Transmissão	Envia 0xAA	Recebe Valor
Confirma Sincronização	Recebe Valor	Envia 0x55
Envia Cabeçalho	Envia 0xAA	Recebe Valor
Confirma Cabeçalho	Recebe Valor	Envia 0x55
***Byte* 0**	Sinal	Recebe os Dados
***Byte* 1**	Valor da Frequência	
***Byte* 2**		
***Byte* 3**		
Envia Tamanho dos Dados	Qtd_coordenadas	
Envia Coordenadas X	Valores de X	
Envia Coordenadas Y	Valores de Y	
Termina Transmissão	Recebe Valor	Envia 0xCC

Com o término do envio do protocolo apresentado acima, é realizada a contagem do total de *bytes* recebidos e é iniciado o cálculo de pontos no microcontrolador.

3.3.3. Microcontrolador

O *firmware* do microcontrolador foi escrito usando a ferramenta Notepad++ 5.9.3, e suas compilações foram efetuadas através do compilador arm-elf-gcc. Este compilador é de grande utilidade e de fácil aplicação, pois atualmente é um *software* livre e possuí uma grande quantidade de informações e documentações disponíveis.

O código fonte completo desenvolvido em linguagem C é mostrado no apêndice C.

3.3.4. Fluxograma do Firmware do Microcontrolador

O *firmware* desenvolvido foi dividido em diversas funções, sendo elas de inicialização, cálculos dos pontos e geração dos sinais com as escritas na memória SRAM.

Na Figura 3-18 é apresentado o fluxograma da rotina principal do *firmware*. Posteriormente, serão detalhadas as rotinas de cálculo dos pontos para ondas senoidais, triangulares, quadradas e arbitrárias devido ao seu grau de importância.

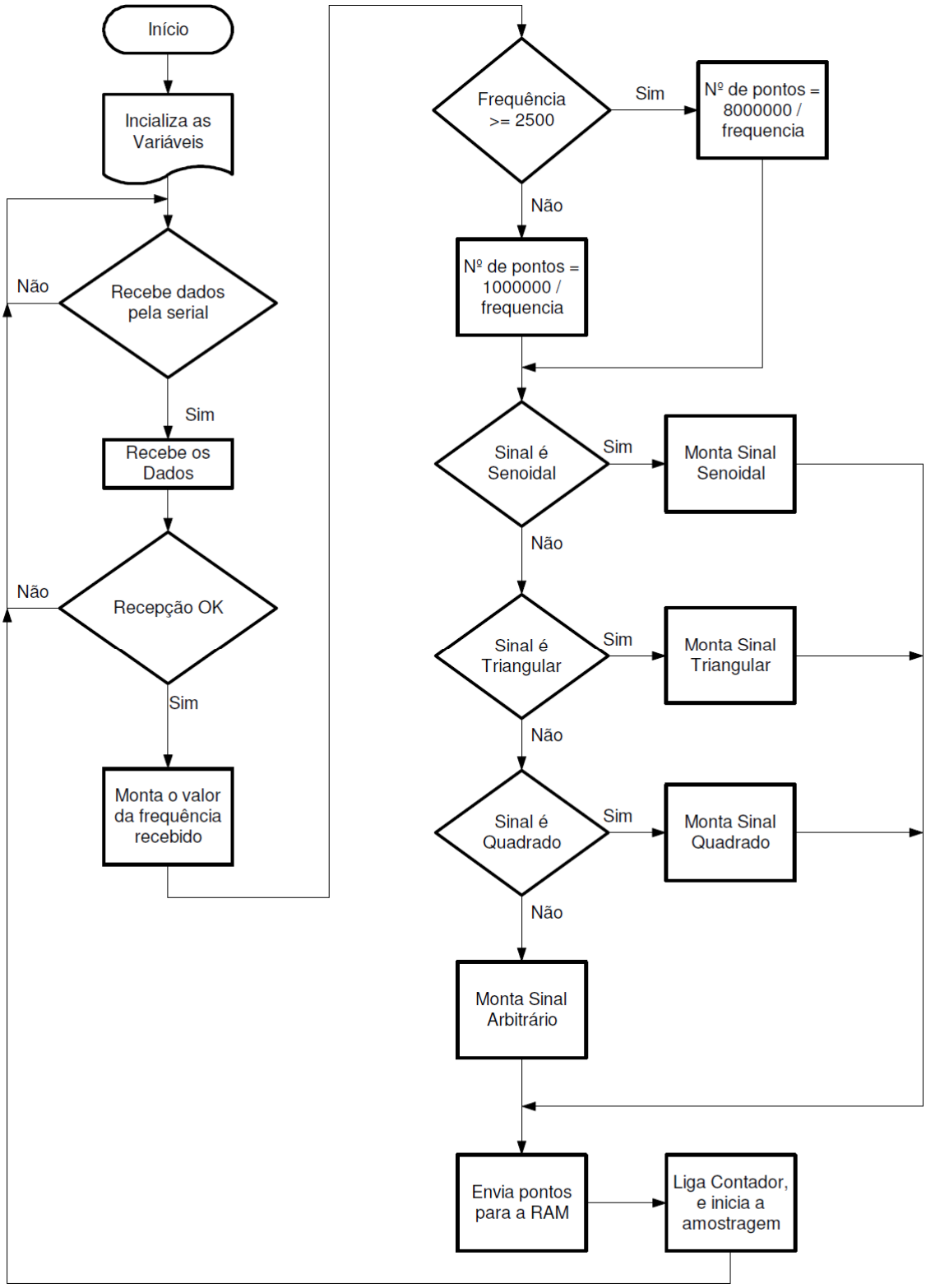


Figura 3-18 - Fluxograma do programa principal

3.3.5. Descrição do Firmware

Ao iniciar o sistema o microcontrolador entra no modo de recepção de dados, e aguardará o início do protocolo de comunicação e o envio dos dados por

parte do supervisório. Depois da etapa de recepção concluída, é iniciada a consistência dos dados recebidos e por sua vez executada a rotina especifica para a geração do sinal informado nos dados recebidos.

Como foi informado na seção referente ao protocolo, durante a recepção dos dados o firmware deve distinguir o tipo de sinal e valores que estão sendo recebidos serialmente, pois essa informação afeta diretamente a quantidade de dados e as informações que serão enviadas ao supervisório. Terminando a recepção dos dados, é feita a montagem do valor correspondente à frequência solicitada. Essa montagem do valor e feita através do rotacionamento de *bytes* e ocorre devido a comunicação de dados ser *byte* a *byte,* e como este valor deve variar de 125 a 1000000 são necessários ao todo três *bytes* para informar o valor correto da frequência.

De posse do valor da frequência, deve ser feito o cálculo dos pontos para o sinal a ser gerado. O cálculo dos pontos é variável de acordo com o valor da frequência recebida e isso ocorre devido a necessidade de haver uma distinção entre o oscilador a ser utilizado. Essas duas informações, frequência e quantidade de pontos, são passadas como parâmetro para as rotinas de geração de pontos e escrita em memória SRAM. A seção 3.2.7 detalha como deve ser feito o cálculo mencionado e a condições para a execução do mesmo.

O reinício da recepção de dados para a formação de uma nova forma de onda é realizado de maneira automático, aonde é necessário somente que os dados sejam enviados através do *software* supervisório.

3.3.5.1. Sinal Senoidal

O sinal senoidal é gerado a partir da seguinte equação:

$$Ponto = \sin\left[\frac{2\times\pi\times número\_ponto}{número\_total\_pontos}\right] \qquad \text{Equação (3.7)}$$

Onde:

π - Valor de PI = 3,141626;

$número\_ponto$ - valor do ponto que está sendo calculado;

$número\_total\_pontos$ - quantidade total de pontos do sinal;

O cálculo usando a equação acima é executado até que o número de pontos calculados seja igual ao número total de pontos do sinal que está sendo gerado nesse instante. Além disso, como o *reset* é dado através de um ponto também escrito na memória do circuito, e os valores devem ser deslocados, pois a escrita de

valores digitais não pode ter valores negativos, é feita a transposição dos pontos dando um ganho para escalonar a amplitude e deslocar o zero para a metade da escala, assim os valores sempre iniciam acima do valor correspondente ao reinicio do ciclo de contagem da memória.

3.3.5.2. Sinal Triangular

Para a montagem do sinal triangular deve-se primeiramente saber o valor do passo correspondente entre os pontos que virão a ser calculados. Esse valor do passo é necessário para que a quantidade de pontos e deslocamentos do sinal durante a formação da onda atenda o valor de frequência que foi passado como parâmetro de cálculo. A equação 3.8 apresenta a forma para cálculo do valor de passo:

$$Passo = \left[\frac{65536}{F_{osc}} \right] \times F_{saida} \qquad \text{Equação (3.8)}$$

Onde:

F_{osc} - Frequência do oscilador base (MHz);

F_{saida} - Frequência do sinal a ser gerado (Hz);

Após o cálculo do passo, são executadas as instruções que fazem a escrita na memória do circuito. Nos valores de subida do sinal triangular, o valor do ponto começa em 0x7FFE, pois dessa forma já o valor já inicia com o deslocamento do sinal, sendo que o próximo valor do ponto será a soma do valor anterior mais o valor passo cálculo, até que a quantidade de pontos de subida chegue na metade da quantidade total de pontos a serem gerados. Já nos valores de descida, ocorre o cálculo contrário, onde a partir do último calculado na subida é feita a subtração do passo correspondente, até que a quantidade de pontos calculados atinja a quantidade total de pontos para a formação do sinal.

3.3.5.3. Sinal Quadrado

Para a geração deste sinal é usada a quantidade total de pontos que devem ser calculados. Assim, é somente executada a divisão deste valor e escrito em memória os valores correspondentes ao nível lógico alto do sinal, que são exatamente a metade da quantidade total de pontos. Já para o nível lógico baixo, é usada a metade restante da quantidade de pontos, sendo que esse valor é igual a 0x7FEE devido ao deslocamento necessário para a lógica digital.

3.3.5.4. Sinal Arbitrário

Para a geração dos sinais arbitrários deve ser feita a interpolação linear usando as coordenadas enviadas pelo *software* supervisório depois da aquisição das mesmas. Essa implementação é necessária devido ao fato de que a onda criada no gráfico é a soma de vários trechos de retas que juntas formam o desenho desejado. Assim, para o cálculo da interpolação são necessárias as coordenadas de ponto inicial e final da reta para que as mesmas formem a reta que terá os pontos intermediários calculados. A Figura 3-19 mostra um exemplo de interpolação linear:

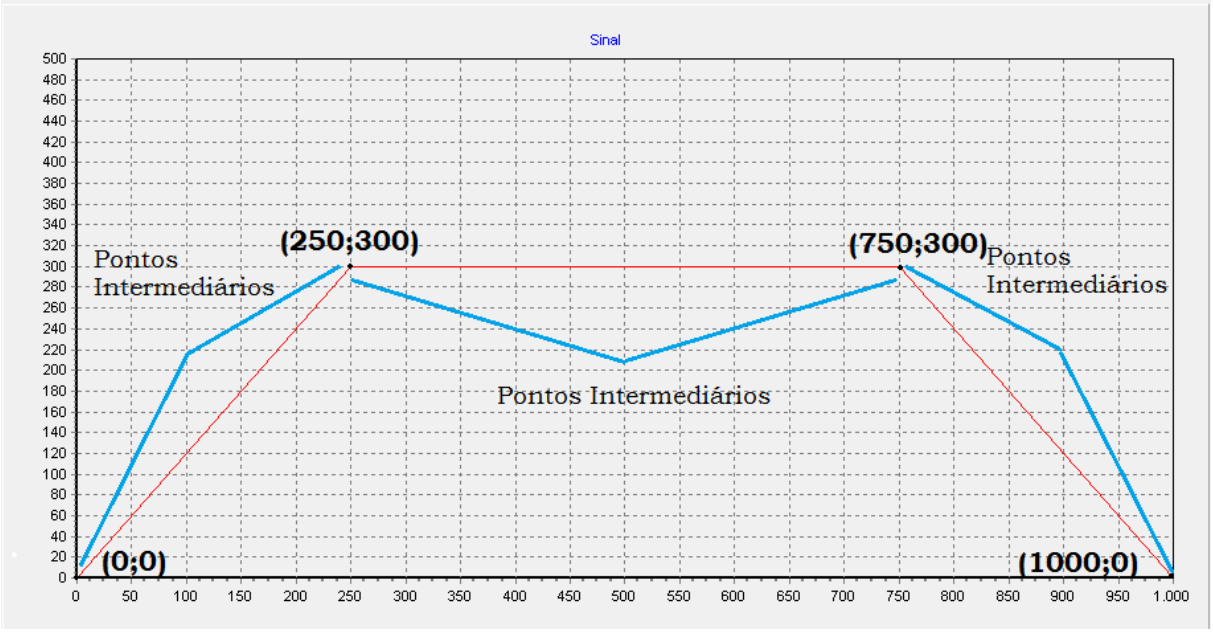


Figura 3-19 - Exemplo de interpolação linear

A figura acima retrata um exemplo de funcionamento da rotina que calcula a interpolação dos pontos capturados. O cálculo inicia usando como referência o primeiro e o segundo ponto da coordenada X e Y, neste caso, 0 e 250 para X e 0 e 300 para Y, respectivamente. Com esses valores, são calculados os pontos intermediários entre as duas coordenadas coletadas usando a equação 2.6, sendo que cada valor intermediário obtido no calculo já é gravado em memória. Esse processo é repetido durante o cálculo dos novos pontos intermediários, agora usando como base os próximos valores de coordenadas X e Y, que são 750 e 250 para X e 300 e 300 para Y. Logo após, o cálculo é ser efetuado, usando os últimos pontos das coordenadas.

3.3.5.5. Escrita e Leitura de Memória

A rotina de escrita de memória foi desenvolvida tomando como base as informações fornecidas pelo o fabricante quanto à temporização de pinos de acesso e controle (vide Figura 3-20).

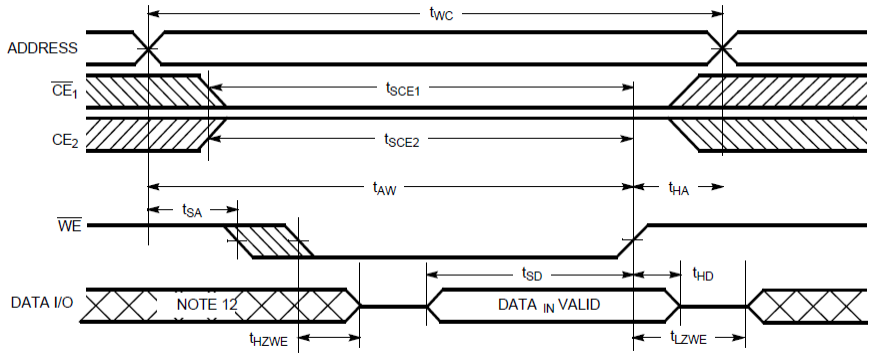


Figura 3-20 - Temporização para Escrita em Memória
Fonte: http://www.cypress.com/?docID=25678

Como o circuito possuí duas memórias trabalhando em paralelo, é feita a escrita primeiramente em uma memória e logo após em outra. Todos os cálculos de valores de pontos são efetuados por variáveis de dois *bytes,* na hora da escrita em memória é realizada uma mascará nos valores de modo que uma das memórias receba a parte alta dos valores do ponto e a outra memória receba a parte baixa do mesmo, de forma intercalada. Dessa forma, durante o processo de leitura, os valores são lidos e enviados para o conversor DAC no mesmo instante.

Como na escrita dos valores não é necessário obter-se velocidade, não foi empregado o uso dos osciladores durante a escrita, sendo que o controle de endereçamento é realizado por um pino do microcontrolador.

O processo de leitura de memória é mais simplificado em relação ao processo de escrita. Para iniciar o processo e por consequência a geração do sinal usando todos os pontos calculados, é necessário alterar a configuração dos pinos da memória, fazendo com que a mesma deixe executar a escrita em seus endereços e passe a executar a leitura dos mesmos. Além disso, deve-se escolher ainda o oscilador que será usado para o endereçamento da memória, essa escolha depende exclusivamente do valor da frequência recebido durante o processo de leitura da porta serial.

4. APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Neste capítulo, apresentam-se os resultados obtidos e as análises realizadas para cada modelo de sinal, através da comparação entre os sinais ideais e reais gerados. Para a obtenção dos resultados e coleta das imagens, foi utilizado um osciloscópio digital de bancada, da marca Tektronix, modelo "TDS-2014C" de 100MHz. O certificado de calibração do osciloscópio encontra-se no anexo B. A Figura 4-1 apresenta o equipamento utilizado para a realização dos testes.

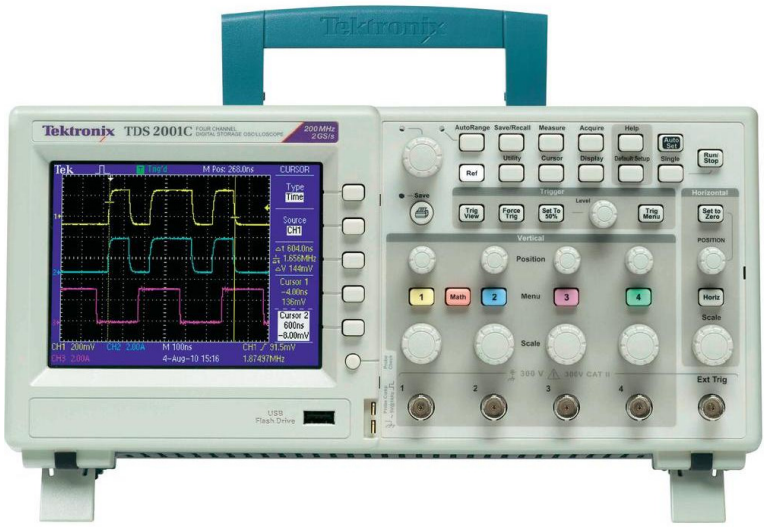


Figura 4-1 - Osciloscópio digital de bancada da marca Tektronix
Fonte: http://www.tek.com/oscilloscope/tds2000-digital-storage-oscilloscope

O processo de testes iniciou através de diversas coletas para cada um dos sinais gerados, onde foram obtidos os valores mínimo, intermediário e máximo de cada sinal. Como houve uma distorção muito grande em frequências altas, pois o circuito possuí a característica de atenuar os sinais devido ao conversor digital-analógico e o circuito de condicionamento operarem como filtros passa-baixa, como a quantidade de pontos diminui consideravelmente com o aumento da frequência foi atribuído um valor máximo de 500kHz para os sinais senoidais, triangular e quadrados, e um valor máximo de 210kHz para o sinal arbitrário.

Para cada coleta obtida, foi realizada a análise com base na comparação entre os pontos gerados durante o cálculo para a formação do sinal, e o sinal gerado pelo circuito desenvolvido, e dessa forma, foi possível estimar através da normalização dos pontos o erro médio eficaz para o somatório das amostras coletadas. Foi escolhido realizar a análise por esse método devido ao valor eficaz ou valor quadrático médio ser uma medida estática de uma variável ou função que se repete periodicamente. A equação que define o erro de valor eficaz é apresentada abaixo:

$$E.Total_{RMS} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}\left(P.ideal_n - P.real_n\right)^2} \qquad \text{Equação (4.1)}$$

Onde:

$E.Total_{RMS}$ - Erro total do sinal coletado;

N - Quantidade total de amostras coletadas;

$P.ideal$ - Valor ideal do ponto gerado pelo circuito;

$P.real$ - Valor real do ponto coletado pelo osciloscópio;

Como nenhum instrumento de medição é absolutamente exato, seu desempenho sofre influência da sua capacidade de efetuar uma medida e torná-la legível, isto é, possibilitar a obtenção de um valor medido com uma incerteza previsível em relação ao valor da grandeza medida. A quantificação dessa incerteza pode ser feita em relação as propriedades mensuráveis que caracterizam o instrumento. Desta forma a medição realizada apresenta uma incerteza ou erro na medição em relação ao valor verdadeiro da grandeza medida. De acordo com o fabricante, o instrumento utilizado para as coletas dos resultados possuí um erro de medição de 3% em relação à medida real. Através disso, e dos resultados de erro obtidos pela equação 4.1 é possível estimar o erro do circuito para o sinal em análise através da seguinte equação:

$$\varepsilon_c = \sqrt{\varepsilon_t^{\,2} - \varepsilon_o^{\,2}} \qquad \text{Equação (4.2)}$$

Onde:

ε_t - Erro Total do Projeto;

ε_o - Erro do Osciloscópio;

ε_c - Erro do Circuito;

Para a análise dos erros, foram coletadas ao todo cinco amostras de cada sinal, sendo de 125Hz, 125kHz, 250kHz, 380kHz e 500KHz para os sinais senoidais, quadrados e triangulares, e 125Hz, 75kHz, 100kHz 150kHz e 210kHz para os sinais arbitrários.

4.1. Análise do Sinal Triangular

Os sinais triangulares atenderam satisfatoriamente a amplitude requerida de 10Vpp. Houve atenuação em frequências altas, entretanto a mesma pode ser corrigida através da variação do potenciômetro de ganho de amplitude.

Como descrito anteriormente, o erro do valor eficaz foi obtido através da obtenção e normalização dos pontos ideais e reais gerados. Os pontos ideais do sinal foram coletados durante o cálculo e gravação dos mesmos na memória do circuito, já os pontos dos sinais reais foram coletados via osciloscópio, através das funções de captura e gravação de imagens. Os resultados obtidos para cada uma das frequências testadas são apresentados no gráfico abaixo:

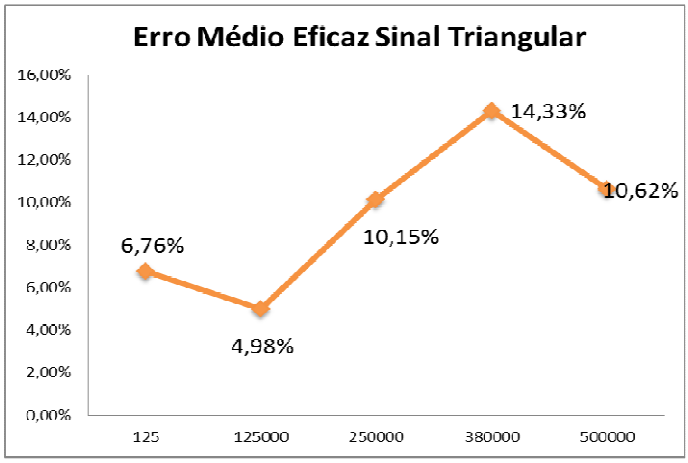


Figura 4-2 – Erro médio eficaz para o sinal triangular

De acordo com os dados apresentados acima, houve uma elevação quantitativa do percentual de erro devido ao aumento das frequências de geração, afetando assim a linearidade e acarretando no aumento das distorções do sinal. Com a obtenção dos valores de erro total do projeto e usando a equação 4.2 são apresentados abaixo os valores percentuais correspondentes ao erro do circuito desenvolvido em função da frequência.

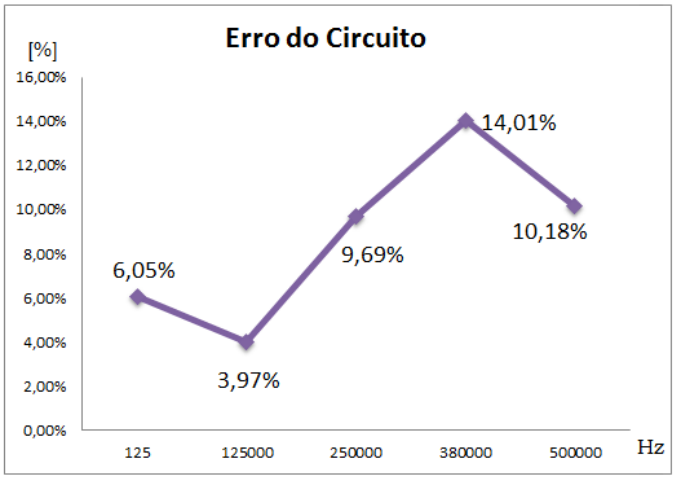


Figura 4-3 - Erro do circuito para o sinal triangular

A Figura 4-4 apresenta graficamente (Valor Normalizado x Quantidade de Amostras) os comparativos entre o sinal triangular real e o ideal normalizado para as frequências coletadas, onde o sinal real é destacado em azul e o ideal em vermelho.

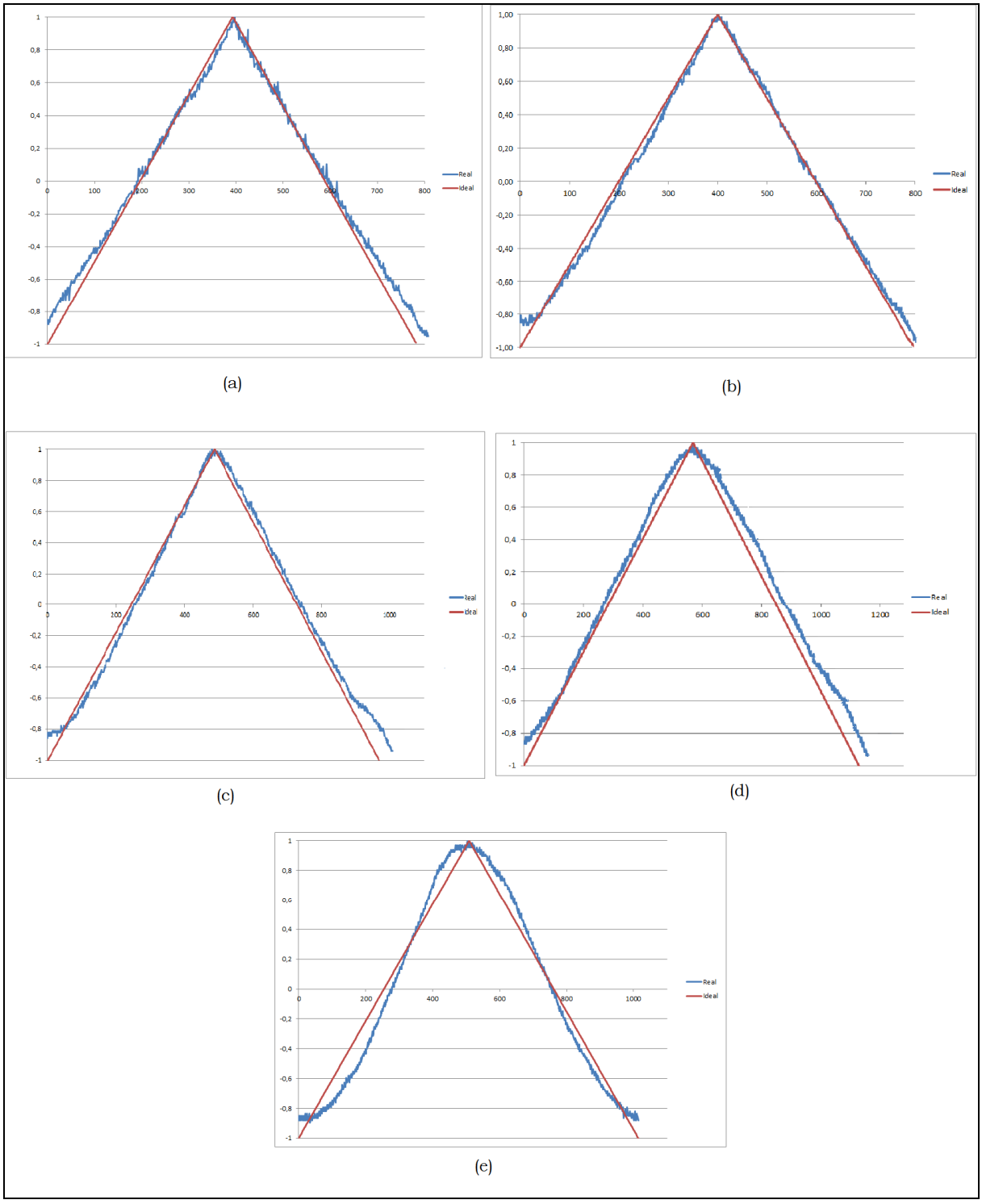


Figura 4-4- (a) Comparativo sinal triangular de 125Hz. (b) Comparativo sinal triangular de 125kHz. (c) Comparativo sinal triangular de 250kHz. (d) Comparativo sinal triangular de 380kHz. (e) Comparativo sinal triangular de 500kHz.

4.2. Análise do Sinal Senoidal

Os sinais senoidais apresentam os melhores resultados entre os sinais analisados. Além de atender a amplitude de 10Vpp, foi obtido um percentual de erro relativamente baixo para as frequências de até 300kHz. Apesar de a análise ter

sido realizada até 500kHz devido as características anteriormente citadas, constatou-se que o circuito tem a capacidade de operar até 700kHz para os sinais senoidais, não sofrendo reduções bruscas em sua amplitude, porém sendo influenciado diretamente pelas distorções harmônicas e quantidade de pontos. Após a máxima frequência citada, no caso frequência de corte, ocorre uma grande atenuação do sinal, chegando assim na frequência de -3dB do circuito.

A Figura 4-5 apresenta o erro médio eficaz para os sinais senoidais, usando como método de cálculo o processo descrito anteriormente.

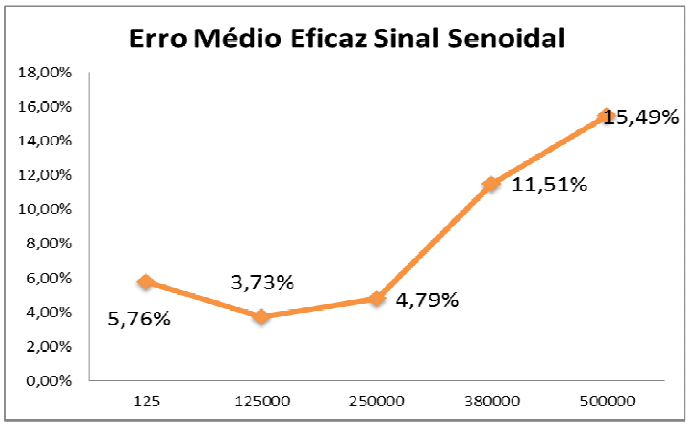


Figura 4-5 - Erro médio eficaz para o sinal senoidal

De acordo com os resultados obtidos acima, observa-se que o erro total do sinal analisado se manteve estável para os sinais de 125Hz, 125kHz e 250kHz, havendo um aumento gradativo do erro para os sinais com a maior frequência. Além disso, é possível constatar que os percentuais de erro são os menores obtidos, influenciando diretamente na qualidade do sinal gerado.

Os erros do circuito em função da frequência acompanham os resultados obtidos no gráfico acima, e são apresentados na Figura 4-6:

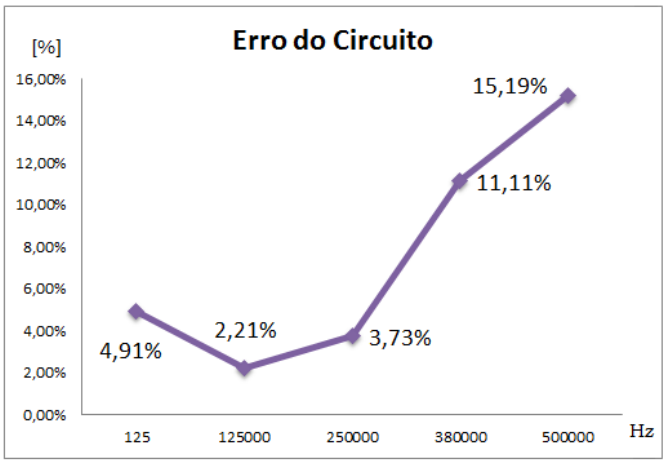


Figura 4-6 - Erro do circuito para o sinal senoidal

A Figura 4-7 apresenta os comparativos realizados entre os sinais ideais e reais gerados para as analises dos sinais senoidais:

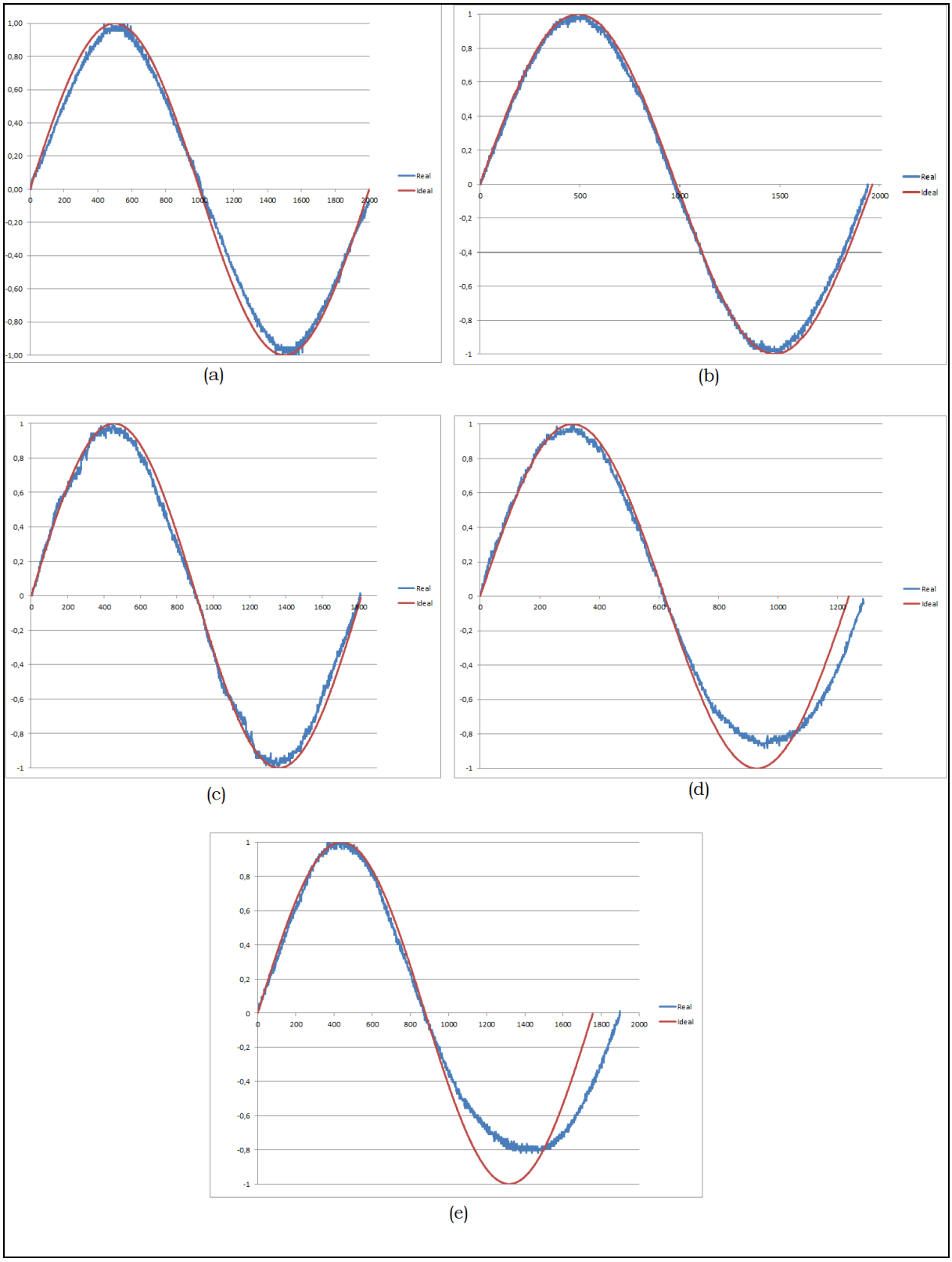


Figura 4-7 - (a) Comparativo sinal senoidal de 125Hz. (b) Comparativo sinal senoidal de 125kHz. (c) Comparativo sinal senoidal de 250kHz. (d) Comparativo sinal senoidal de 380kHz. (e) Comparativo sinal senoidal de 500kHz.

Analisando os comparativos realizados e os valores de erros obtidos, é possível observar que até a frequência de 250kHz o circuito se comportou de forma esperada, pois não houve deformações significativas no sinal. Porém, as respostas para as frequências de 380 e 500kHz sofreram grande deformação em seu pico negativo, de modo que o sinal não realiza a envoltória correta durante a sua amostragem. Esse comportamento afeta diretamente na qualidade e na resposta em frequência do sinal, fazendo com que o mesmo também não atenda de forma satisfatória a amplitude máxima requerida.

4.3. *Análise do Sinal Quadrado*

Quanto a amplitude, a onda quadrada atendeu o objetivo de operar em até 10Vpp. Entretanto, houve um aumento percentual de erro em operação nas frequências altas, semelhante ao sinal triangular.

O tempo de subida e descida dos sinais analisados tiveram pouca variação. Para o tempo de subida, as melhores e piores respostas foram de 328ns para o sinal de 125Hz e 403ns para sinal de 380Khz, respectivamente. Já o tempo de descida teve uma resposta de 349ns para o sinal de 500kHz e 386ns no sinal de 125Hz.

Para a coleta das amostras foi realizado o procedimento idêntico à coleta das ondas triangulares. O Erro médio eficaz para os sinais quadrados são apresentados abaixo:

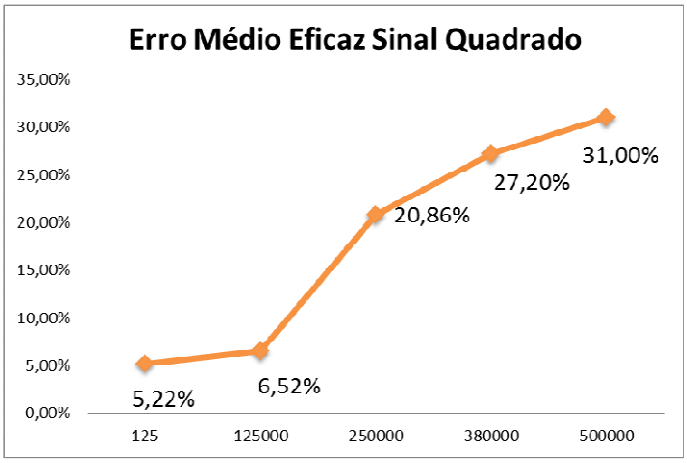


Figura 4-8 - Erro médio eficaz para o sinal quadrado

O Erro do circuito em função da frequência para a geração do sinal de onda quadrada é apresentada abaixo:

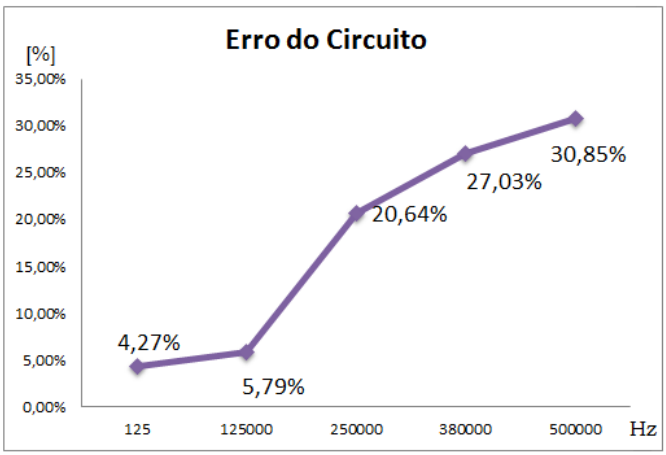


Figura 4-9 - Erro do circuito para o sinal quadrado

A Figura 4-10 apresenta os comparativos dos sinais coletados com os sinais ideais para a forma de onda quadrada. É possível perceber que com o aumento gradual da frequência de operação ocorre uma distorção no sinal, fazendo com que os mesmos apresentem arredondamento durante a transição de nível lógico alto para nível lógico baixo.

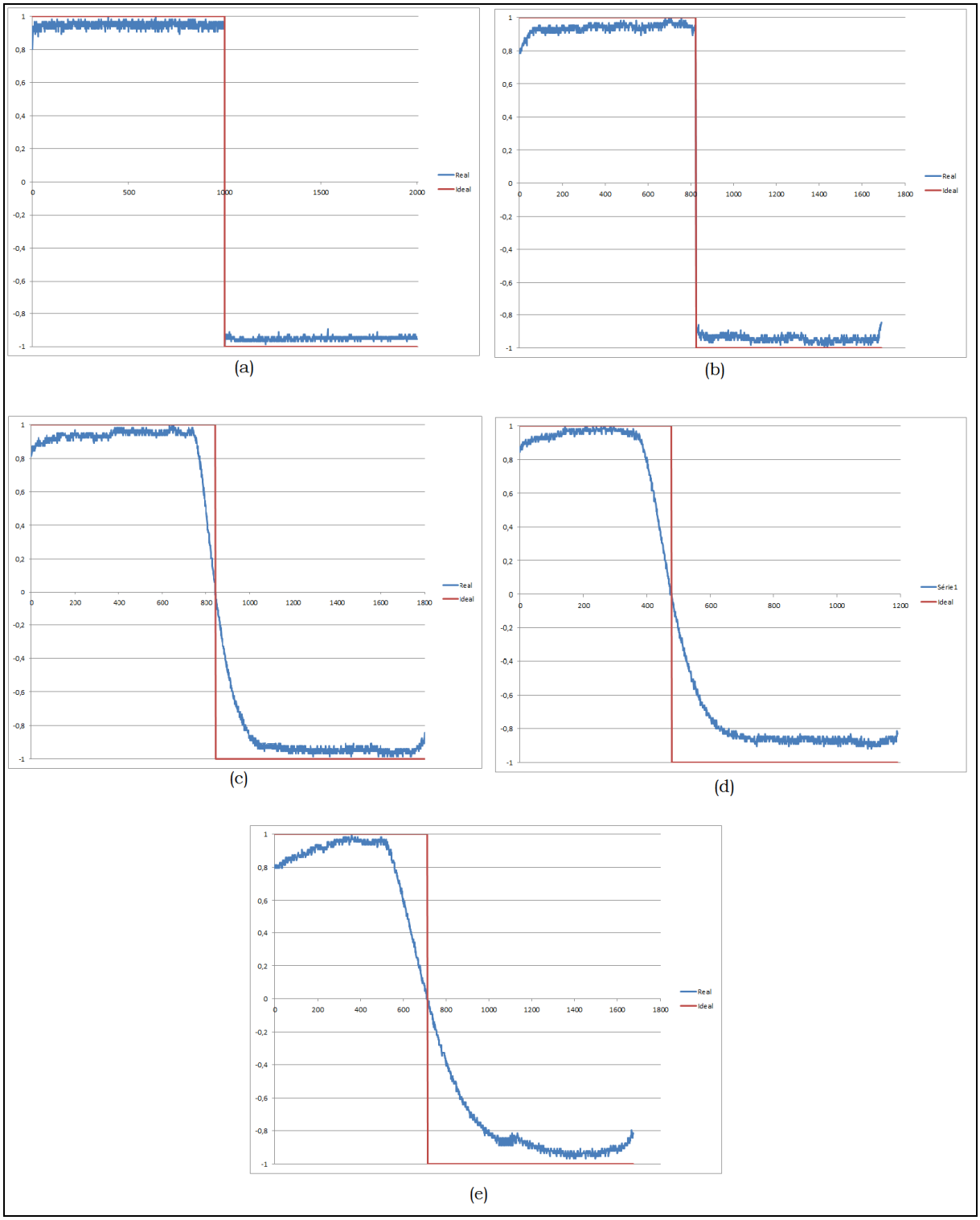


Figura 4-10 - (a) Comparativo sinal quadrado de 125 Hz. (b) Comparativo sinal quadrado de 125 kHz. (c) Comparativo sinal quadrado de 250 kHz. (d) Comparativo sinal quadrado de 380 kHz. (e) Comparativo sinal quadrado de 500 kHz.

Além das características anteriormente citadas de comportamento do circuito e do erro associado a medição, a onda quadrada também sofre a influência da velocidade de resposta do amplificador operacional utilizado no desenvolvimento do circuito. Como pode ser visualizada nos sinais de 250kHz, 380kHz e 500kHz o

tempo de transição de nível lógico alto para nível lógico baixo tende a ser a mesmo para todos os casos.

4.4. Análise do Sinal Arbitrário

Os sinais arbitrários tiveram os seus resultados abaixo do esperado, pois não foi possível obter sinais satisfatórios para as frequências altas apresentadas nas formas de onda anteriores. Estes sinais sofreram grande influência de desempenho devido os problemas de linearidade apresentados na onda triangular, e os problemas de arredondamento apresentados na onda quadrada. Devido a esse motivo, optou-se por realizar a análise de erro para frequências menores que as analisadas anteriormente. Mesmo assim, houve um aumento do percentual de erro em relação aos números apresentados nas análises anteriores.

A Figura 4-11 apresenta o erro médio eficaz para os sinais arbitrários.

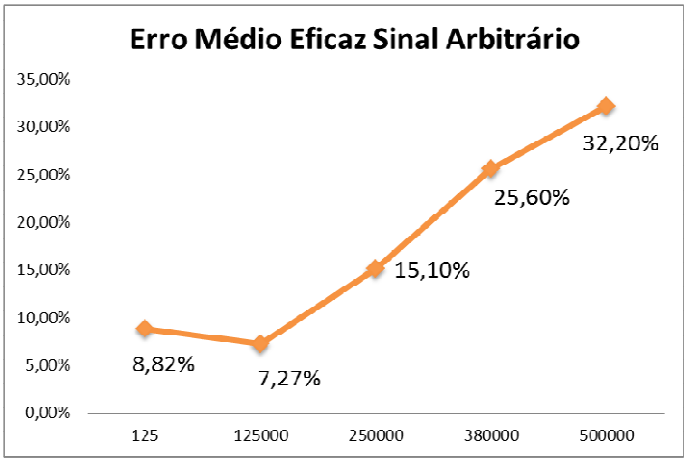


Figura 4-11 - Erro de valor eficaz para o sinal arbitrário

A Figura 4-12 apresenta os erros do circuito em função da frequência.

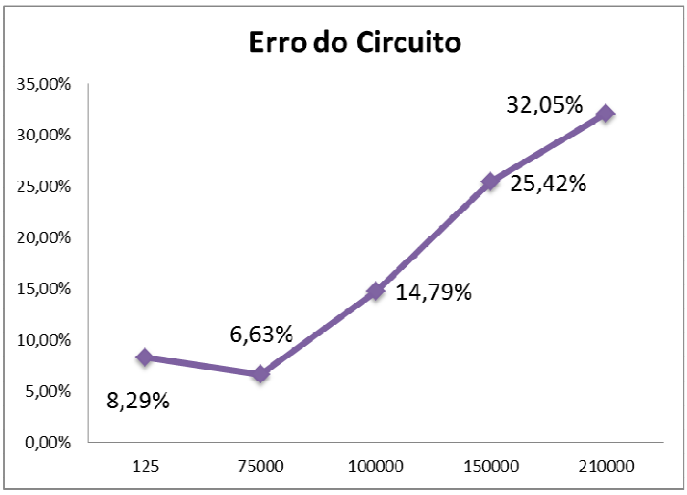


Figura 4-12 - Erro do circuito para o sinal arbitrário

Com os resultados obtidos é possível dizer que os sinais arbitrários tiveram o pior resultado dentre os quatro tipos de sinais analisados. Diferentemente dos outro sinais, este sinal apresenta um erro relativamente grande já no sinal de 150kHz, enquanto que, para os outros sinais, o erro torna-se muito grande somente na frequência de 380kHz.

Nas Figura 4-13 a Figura 4-22 são apresentados os comparativos realizados para o cálculo de erro do circuito, além disso, são mostrados também as formas de onda desenhadas no *software* supervisório.

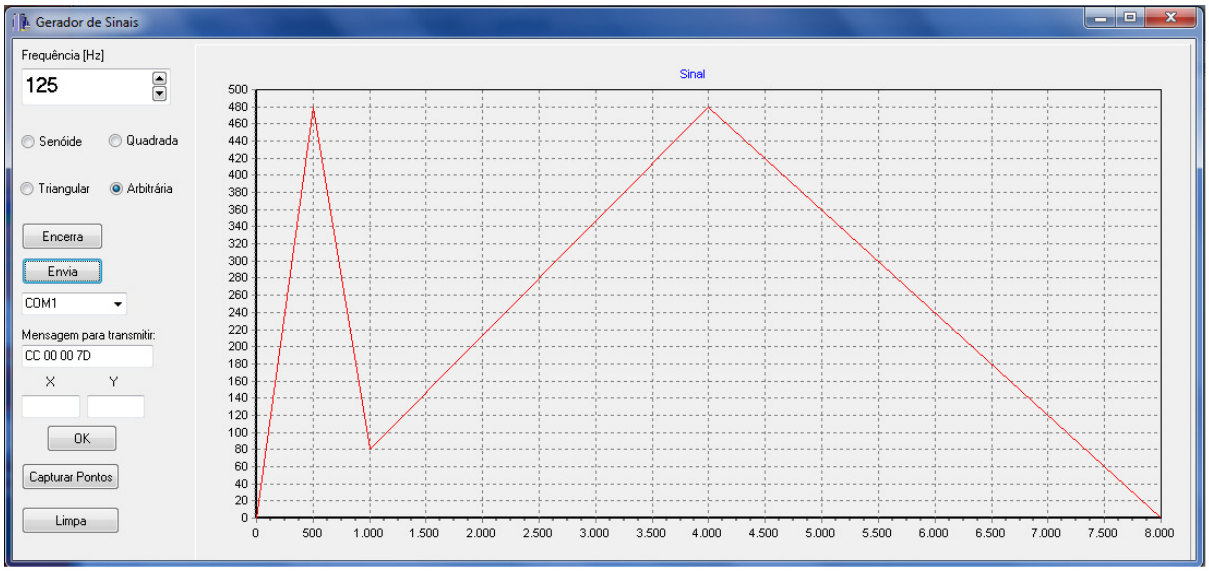


Figura 4-13 - Sinal desenhado para frequência de 125Hz

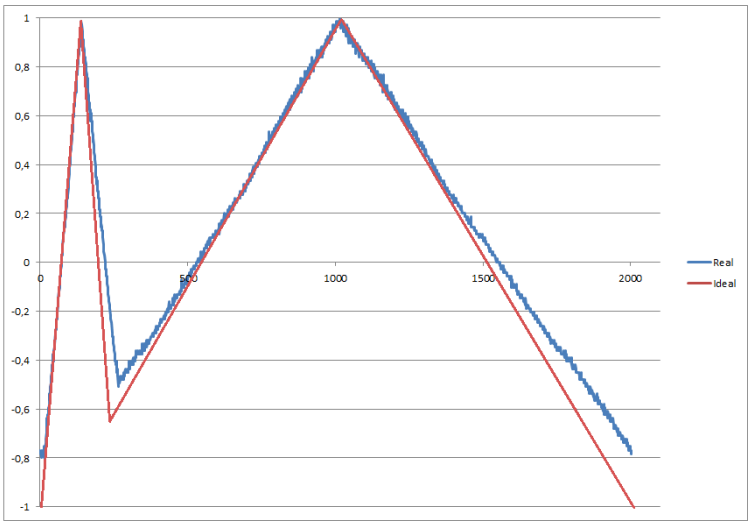


Figura 4-14 - Comparativo sinal arbitrário de 125Hz

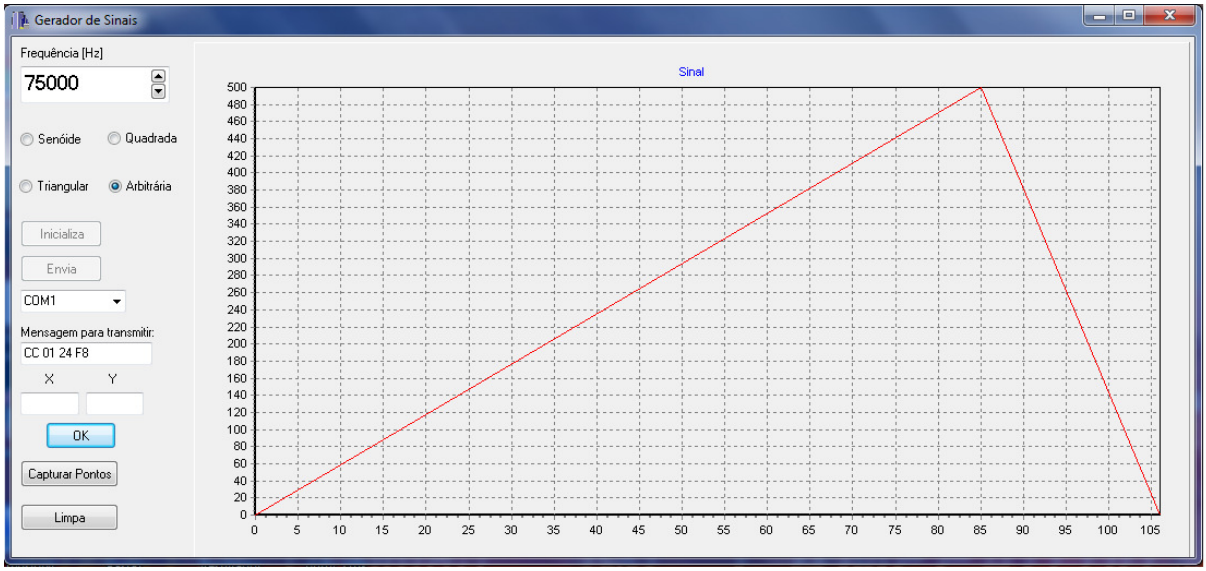


Figura 4-15 - Sinal desenhado para frequência de 75kHz

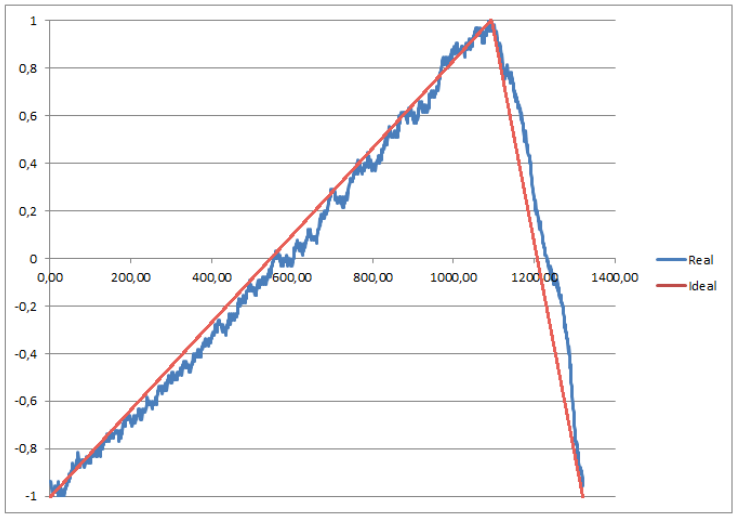


Figura 4-16 - Comparativo sinal arbitrário de 75kHz

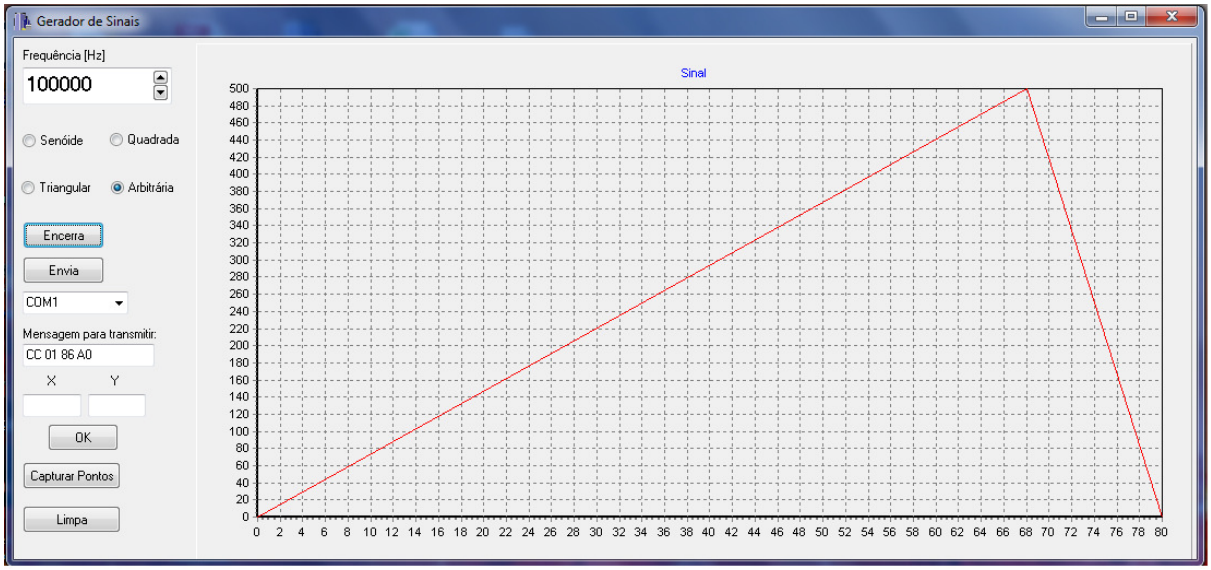


Figura 4-17 - Sinal desenhado para frequência de 100kHz

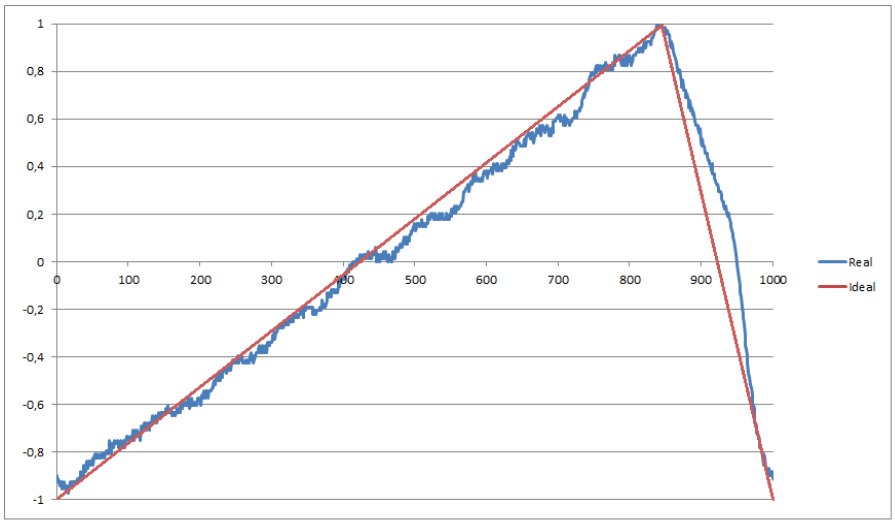


Figura 4-18 - Comparativo sinal arbitrário de 100kHz

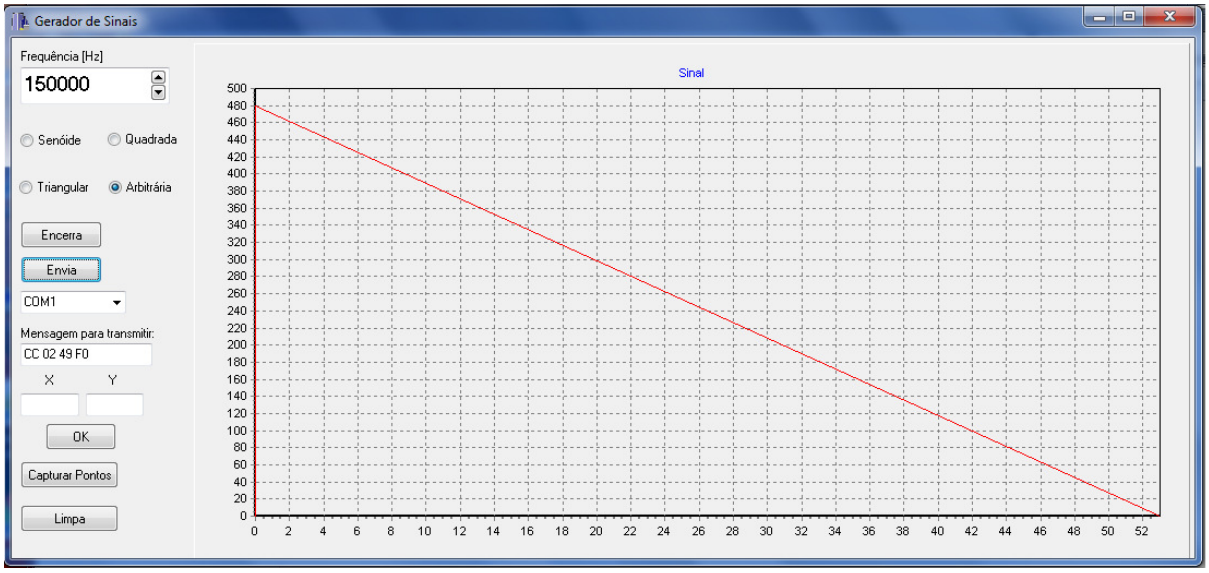


Figura 4-19 - Sinal desenhado para frequência de 150kHz

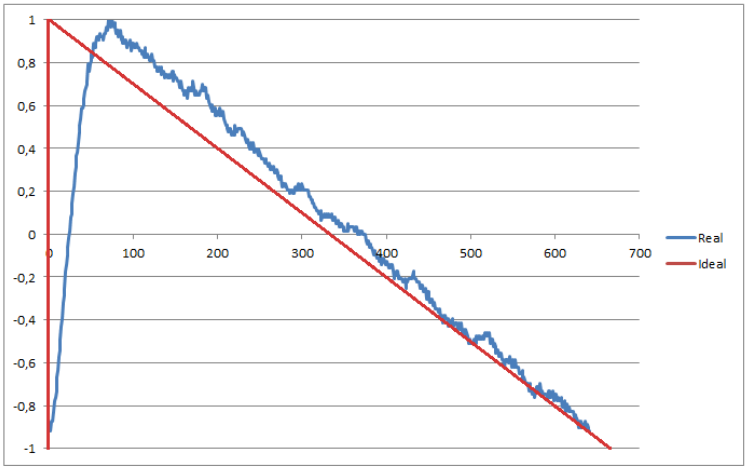


Figura 4-20 - Comparativo sinal arbitrário de 150kHz

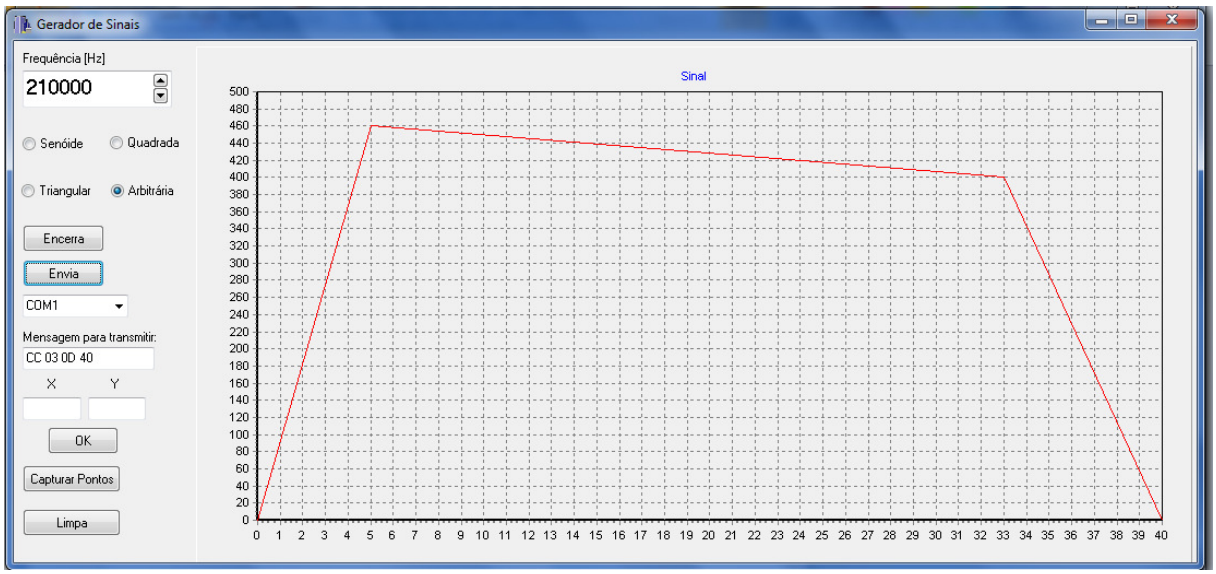


Figura 4-21 - Sinal desenhado para frequência de 210kHz

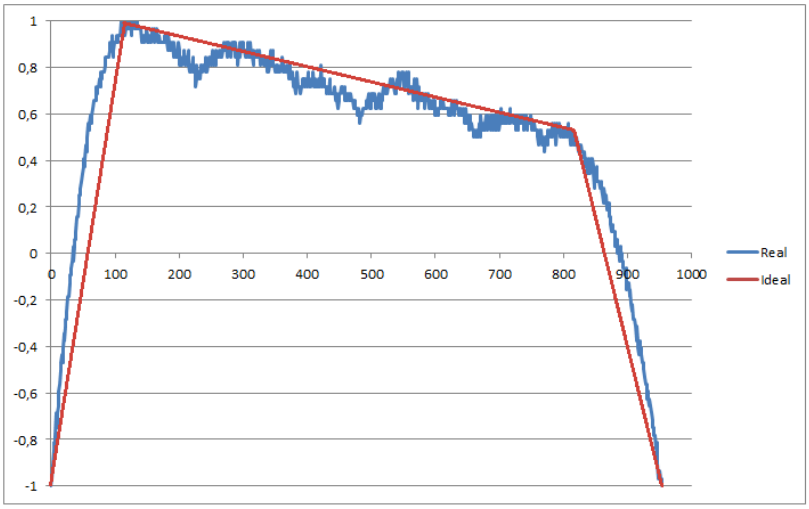


Figura 4-22 - Comparativo sinal arbitrário de 210kHz

De acordo com as imagens acima, viu-se que o circuito se comportou de forma linear para as frequências de 100kHz, apesar de haver uma distorção no sinal da transição de nível lógico alto para nível lógico baixo. Como esperado, ocorreu uma grande distorção nos sinais de 150 e 210kHz. Outro ponto negativo foi que o sinal de 125Hz não atendeu à amplitude máxima por apresentar problemas em seu pico negativo, entretanto, esse sinal foi o que apresentou os melhores resultados quanto a linearidade.

5. CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi desenvolver uma ferramenta de geração de sinais arbitrários, controlados por software, empregando o método de síntese digital direta. Nesse método, o sinal gerado na saída do circuito é influenciado diretamente pela precisão do cristal oscilador utilizado.

O projeto apresentou um melhor comportamento para os sinais senoidais e triangulares. Obtiveram-se respostas de até 250kHz de forma bastante linear, havendo pouca interferência na aparência do sinal. Entretanto, os sinais quadrados perder qualidade a partir da frequência de 125kHz e os arbitrários a partir de 100kHz. Essas limitações ocorreram devido às restrições apresentadas no circuito analógico do projeto, sendo constatado que os amplificadores operacionais introduziram ruídos nos sinais gerados.

Além disso, as limitações apresentadas também ocorreram devido a pouca quantidade de pontos na formação dos sinais acima dos valores mencionados. No entanto, os circuitos digitais de endereçamento de memória, geração de *reset* e geração de *clock* mostraram-se eficientes em todas as condições de funcionamento, não havendo ruídos, nem outras variáveis espúrias que afetassem seus comportamentos.

Os resultados obtidos mostram que os objetivos foram atingidos parcialmente. Inicialmente, este projeto tinha como objetivo obter sinais com amplitude de até 10Vpp e frequência de até 1MHz. Todavia, com os resultados apresentados acima e os percentuais de erros obtidos, foi possível chegar ao resultado esperado na amplitude do sinal.

Na frequência, os resultados fugiram do esperado devido aos problemas anteriormente relatados. Dessa forma, é aconselhável que o emprego do gerador fique limitado aos valores de frequência de 250kHz para os sinais senoidais e triangulares, 125kHz para os sinais quadrados e 100kHz para os sinais arbitrários conforme já mencionados.

Em relação a futuras melhorias, pode ser realizada uma mudança no *layout* na placa de conversão e condicionamento de sinal para minimizar a susceptibilidade da mesma a ruídos na parte analógica do circuito. Quanto ao desempenho, é recomendável uma mudança no circuito de conversão digital-analógico, fazendo com que o sinal convertido não seja interferido pela amostragem dos pontos que estão sendo lidos da memória externa. Além disso, a fabricação de uma placa de circuito impresso com diversos *layers* possibilitará uma melhoria no sinal gerado.

Como sugestão para trabalho futuros, está a implementação do campo de defasagem de fase na onda senoidal no *software* supervisório e o aumento dos osciladores para a geração de sinais com maiores quantidade de pontos nos sinais com frequência elevada.

6. REFERÊNCIAS

[1] MAITELLI, A. L. – Apostila de Controladores Lógicos Programáveis – UFRN, Natal-RN, 2003.

[2] Direct-Digital Frequency Synthesis; a basic tutorial; Osicom Technologies Inc, 1983. Disponível em <http://web.itu.edu.tr/~pazarci/ddstutor.html>. Acesso em: 14 de Março de 2012.

[3] GUSSOW, Milton. Eletricidade básica. São Paulo : McGraw-Hill do Brasil,1985.

[4] A Technical Tutorial on Digital Signal Synthesis, Tutorial, Analog Devices Inc.,1999. Disponível em: <http://www.analog.com/static/imported-files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf>. Acesso em: 17 de Março de 2012.

[5] CLÁUDIO, Dálcidio M. e MARINS Jussara M. Cálculo Numérico e Computacional - Teoria e Prática, São Paulo: Ed: Atlas, 2000.

[6] TOCCI, Ronaldo J. WIDMER, Neal S. SISTEMAS DIGITAIS: Princípios e Aplicações. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A, 1998.

[7] CANZIAM, Edmur. Comunicação Serial – RS232. Cotia, 2006. Disponível em <http://www.professores.aedb.br/arlei/AEDB/Arquivos/rs232.pdf>. Acesso em: 31 de Março de 2012.

[8] NXP – LPC2378, Data Sheet. Disponivel: http://www.nxp.com/documents/data_sheet/LPC2377_78.pdf. Acesso em: 04 de Março de 2012.

[9] UNICID. Linguagens de Programação. São Paulo, 2007. Disponível em: <www.dee.feis.unesp.br/.../c_04_linguagem_de_programacao.pdf >. Acesso em: 10 de Março de 2012.

[10] 64-Kbit (8 K × 8) Static RAM CY7C185, Data Sheet. Disponível:

http://www.cypress.com/?docID=25678. Acesso em: 30 de Abril de 2012.

[11] CYPRESS – CY7C185, Data Sheet. Disponível: http://www.cypress.com/?docID=25678. Acesso em: 30 de Abril de 2012.

[12] NXP – 74HCT373, Data Sheet. Disponível: http://www.nxp.com/documents/data_sheet/74HC_HCT373.pdf. Acesso em: 30 de Abril de 2012.

[13] NXP – 74HC590, Data Sheet. Disponível: http://www.nxp.com/documents/data_sheet/74HC590.pdf. Acesso em: 30 de Abril de 2012.

[14] NXP – 74HCT02, Data Sheet. Disponível: http://www.nxp.com/documents/data_sheet/74HC_HCT02.pdf. Acesso em: 30 de Abril de 2012.

[15] NXP – 74HCT02, Data Sheet. Disponível: http://www.nxp.com/documents/data_sheet/74HC_HCT30.pdf. Acesso em: 15 de Maio de 2012.

[16] NXP – 74HCT32, Data Sheet. Disponível: http://www.nxp.com/documents/data_sheet/74HC_HCT32.pdf. Acesso em: 15 de Maio de 2012.

[17] NATIONAL INSTRUMENTS – LM7171, Data Sheet. Disponível: http://www.ti.com/lit/ds/symlink/lm7171.pdf. Acesso em: 20 de Junho de 2012.

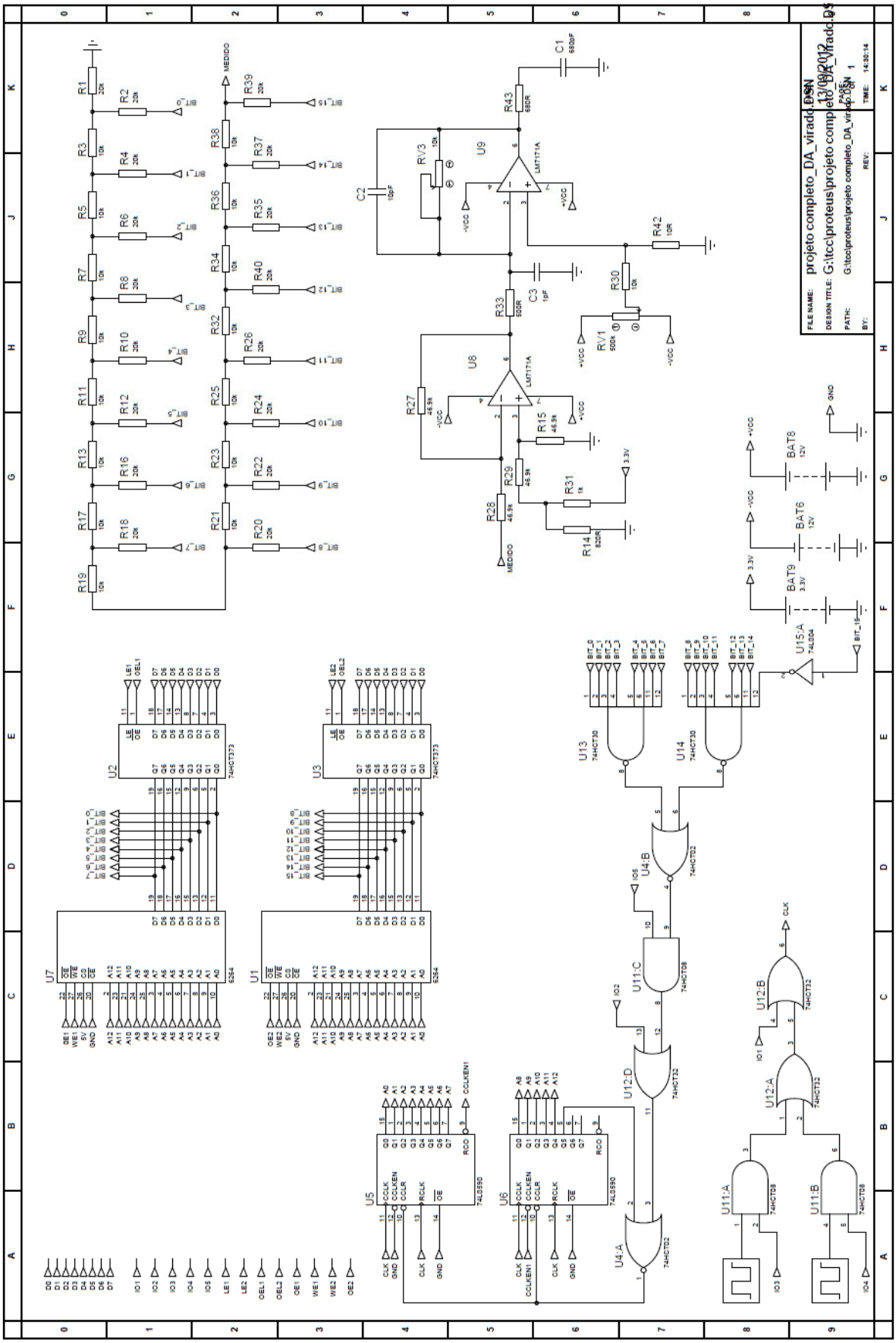
7. OBRAS CONSULTADAS

COCIAN, LUIS F. E. – Manual da Linguagem C – 1ª Ed. – Canoas: Editora da Ulbra, 2004.

BOYLESTAD, R.; NASHELSKY, L. – Dispositivos Eletrônicos e Teoria de Circuitos - 3ª Ed. – Rio de Janeiro: PHB, 1982.

PERTENCE, ANTONIO JR – Amplificadores Operacionais e Filtros Ativos – 6 ª Ed. – São Paulo: Makron Books, 2003.

APÊNDICE A – ESQUEMÁTICO DA PLACA DE ARMAZENAMENTO, CONVERSÃO E CONDICIONAMENTO DO SINAL


FILE NAME: projeto completo_DA_virado.DSN
DESIGN TITLE: G:\tcc\proteus\projeto completo_DA_virado.DSN
PATH: G:\tcc\proteus\projeto completo_DA_virado.DSN
BY:
REV: 1
TIME: 14:36:14
13/08/2012
MEDIDO
+VCC
-VCC
GND
3.3V
U1
U2
U3
U4:A
U4:B
U5
U6
U7
U8
U9
U11:A
U11:B
U11:C
U12:A
U12:B
U12:D
U13
U14
U15:A
RV1
RV3
BAT6
BAT8
BAT9

APÊNDICE B – SOFTWARE SUPERVISÓRIO

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "UPrincipal.h"
#include "Serial.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CSPIN"
#pragma resource "*.dfm"

        TfrmPrincipal *frmPrincipal;
        AnsiString sPorta;
        int freq_maxima;  //variavel que guarda a frequencia maxima de cada ponto
        unsigned char vetor_serial[3]; //vetor do frame que sera enviado pela serial

        AnsiString teste;
        char buffer[2];

        int x = 0;
        int nome[3];

        int contador_pontos = 0;        // conta da quantidade de pontos
        int valor_ponto_x,valor_ponto_y;

        unsigned short int xs[8010];
        unsigned short int ys[8010];

        char qtd_coordenadas;

        unsigned char valor_baixo;
        unsigned char valor_alto;
        unsigned short int valor_ponto;
        unsigned short int vetor_onda[128];
        float n_pontos;
        unsigned short int teste_ponto;
//---------------------------------------------------------------------------
__fastcall TfrmPrincipal::TfrmPrincipal(TComponent* Owner)
  : TForm(Owner)
{
  Chart1->Visible = false;
  lt_pontos->Visible = false;
  bt_limpa->Visible = false;
  frmPrincipal->Width = 180;
  frmPrincipal->Height = 346;
  frmPrincipal->Left = 130;
```

```
  Chart1->Series[0]->AddXY(0,0,"",clRed); // define ponto inicial (0;0).
  Chart1->Series[0]->AddXY(500,0,"",clRed); // define ponto final (127;0).

  for (int y = 0; y <= 10; y++) {
        vetor_serial[y] = 0;
  }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::FormCreate(TObject *Sender)
{
   // Inicializa porta serial padrão: COM1.
    sPorta = ComboBox1->Items->Strings[0];
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnInicializaSerialClick(TObject *Sender)
{

if(btnInicializaSerial -> Caption == "Inicializa")
   if (bInicializaSerial(sPorta.c_str())== ERRO_ABERTURA_PORTA_COMUNICACAO )
      {
      ShowMessage("Erro na Abertura da Porta");
      }
   else
     {
      btnInicializaSerial -> Caption = "Encerra";
      Timer1 -> Enabled = True;
            btnEnvia -> Enabled = True;
      }
else
   {
    bEncerraSerial();
    btnInicializaSerial -> Caption = "Inicializa";
    btnEnvia -> Enabled = False;
    Timer1 -> Enabled = False;
    }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::rdgPortaSerialClick(TObject *Sender)
{
   sPorta = ComboBox1->Items->Strings[ComboBox1->ItemIndex];
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::btnEnviaClick(TObject *Sender)
{
        unsigned char *handshake, val, vetor;
        int x;
        int qtde = 0;
        BYTE aux;

        val = 0xAA;

        bTransmiteMensagem(&val);

        Sleep(3);

        do
        {
                aux = bRecebeMensagem(buffer);
        }
```

```
while (aux == SS_SERVICO_EM_ANDAMENTO);
Sleep(3);
if(buffer[0] == 'U')
{
        val = 0xAA;
        bTransmiteMensagem(&val);

        Sleep(20);
        do
        {
                aux = bRecebeMensagem(buffer);
        }
        while (aux == SS_SERVICO_EM_ANDAMENTO);
        Sleep(20);

        if(buffer[0] == 'U')
        {
                for (int i = 0; i < 4; i++) {
                        if (vetor_serial[i] == 0xFF) {  // ajusta valor devido a problema
                                vetor_serial[i] = 0xFE;    // na serial
                        }
                        bTransmiteMensagem(&vetor_serial[i]);
                        Sleep(20);
                }
                do
                {
                        aux = bRecebeMensagem(buffer);
                }
                while (aux == SS_SERVICO_EM_ANDAMENTO);
                Sleep(20);

                if (buffer [0] == '»')
                {
                        qtd_coordenadas = lt_pontos->Items->Count;
                        qtd_coordenadas *=4;
                        bTransmiteMensagem(&qtd_coordenadas);
                        Sleep(20);                              // manda cabeçalho
                        bTransmiteMensagem(0x00);

                        for (int i = 0; i < lt_pontos->Items->Count; i++)
                        {                                       // de X
                                valor_ponto = xs[i];
                                valor_baixo = valor_ponto & 0x00FF;
                                valor_alto = (valor_ponto & 0xFF00)>>8;

                                if (valor_alto == 0xFF){
                                        valor_alto = 0xFE;    // na serial
                                }
                                if (valor_baixo == 0xFF){
                                        valor_baixo = 0xFE;    // na serial
                        }
                        bTransmiteMensagem(&valor_alto);
                        Sleep(20);

                        bTransmiteMensagem(&valor_baixo);
                        Sleep(20);
                }
                Sleep(20);
                bTransmiteMensagem(0x00);
```

```
                        for (int i = 0; i < lt_pontos->Items->Count; i++)
                        {                                       // de Y
                                valor_ponto = ys[xs[i]];
                                valor_baixo = valor_ponto & 0x00FF;
                                valor_alto = (valor_ponto & 0xFF00)>>8;

                                if (valor_alto == 0xFF){  // ajusta valor devido a problema
                                        valor_alto = 0xFE;    // na serial
                                }
                                if (valor_baixo == 0xFF){  // ajusta valor devido a problema
                                        valor_baixo = 0xFE;    // na serial
                                }

                                bTransmiteMensagem(&valor_alto);
                                Sleep(20);

                                bTransmiteMensagem(&valor_baixo);
                                Sleep(20);
                         }
                           ShowMessage("Envio OK");
                        }
                        else
                           ShowMessage("Envio OK");
                }
                else
                ShowMessage("Erro de Envio");
        }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::ComboBox1Change(TObject *Sender)
{
   sPorta = ComboBox1->Items->Strings[ComboBox1->ItemIndex];
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::bt_senoideClick(TObject *Sender)
{
        btnInicializaSerial -> Enabled = true;

        Chart1->Visible = false;
        frmPrincipal->Width = 180;
        frmPrincipal->Height = 346;

        lt_pontos->Visible = false;
        bt_limpa->Visible = false;

        Chart1->Series[0]->Clear();      //zera valores
        freq_maxima = frmPrincipal->sp1_frequencia->Value;

        Chart1->Series[0]->AddXY(0,250,"",clRed); // define ponto inicial (0;0).
        Chart1->Series[0]->AddXY(250,250,"",clRed); // define ponto final (freq_maximo;0).

        vetor_serial[0] = 0xDD;
        vetor_serial[1] = (freq_maxima >> 16);
        vetor_serial[2] = (freq_maxima >> 8);
        vetor_serial[3] = freq_maxima;

        teste = ""; // zera a string antes de montar o frame

        AnsiString tmp = ""; //variavel temporaria que guarda o valor da conversão
```

```
        for (int i = 0; i < 4; i++) {
        tmp.sprintf("%02X ", vetor_serial[i]); // for que monta o frame da string
        teste += tmp;
        }
        edtMensagemTransmitir->Text = teste;

        valor_seno_x = 0;
        valor_seno = 0;

        for (contador_pontos = 0; contador_pontos <= 500 ; contador_pontos++)
        {
          valor_seno = ((2 * PI * contador_pontos)/500);
          valor_seno = sin(valor_seno);
          valor_seno = valor_seno * (500/2)+250;

          Chart1->Series[0]->AddXY(valor_seno_x,valor_seno,"",clRed);
          valor_seno_x++;
        }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::bt_quadradaClick(TObject *Sender)
{
        btnInicializaSerial -> Enabled = true;

        Chart1->Visible = false;
        frmPrincipal->Width = 180;
        frmPrincipal->Height = 346;

        lt_pontos->Visible = false;
        bt_limpa->Visible = false;

        Chart1->Series[0]->Clear();      //zera valores
        freq_maxima = frmPrincipal->sp1_frequencia->Value;

        vetor_serial[0] = 0xAA;
        vetor_serial[1] = (freq_maxima >> 16);
        vetor_serial[2] = (freq_maxima >> 8);
        vetor_serial[3] = freq_maxima;

        teste = ""; // zera a string antes de montar o frame

        AnsiString tmp = ""; //variavel temporaria que guarda o valor da conversão

        for (int i = 0; i < 4; i++) {
                tmp.sprintf("%02X ", vetor_serial[i]); // for que monta o frame da string
                teste += tmp;
        }
        edtMensagemTransmitir->Text = teste;

        for (contador_pontos = 0; contador_pontos < 500 / 2; contador_pontos++)
        {
                valor_ponto_y = 500;                    // salva na variavel base valor do ponto
                Chart1->Series[0]->AddXY(contador_pontos,valor_ponto_y,"",clRed);
        }

        // for que escreve os valores negativos da onda
        for (contador_pontos = 500 / 2; contador_pontos < 500; contador_pontos++)
        {
                valor_ponto_y = 0;                      // salva na variavel base valor do ponto
                Chart1->Series[0]->AddXY(contador_pontos,valor_ponto_y,"",clRed);
```

```
        }
        valor_ponto_y = 500;                            // salva na variavel base valor do ponto
        Chart1->Series[0]->AddXY(contador_pontos,valor_ponto_y,"",clRed);
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::bt_triagularClick(TObject *Sender)
{
        btnInicializaSerial -> Enabled = true;

        Chart1->Series[0]->Clear();      //zera valores
        frmPrincipal->Width = 180;
        frmPrincipal->Height = 346;
        Chart1->Visible = false;

        lt_pontos->Visible = false;
        bt_limpa->Visible = false;

        freq_maxima = frmPrincipal->sp1_frequencia->Value;

        vetor_serial[0] = 0xEE;
        vetor_serial[1] = (freq_maxima >> 16);
        vetor_serial[2] = (freq_maxima >> 8);
        vetor_serial[3] = freq_maxima;

        teste = ""; // zera a string antes de montar o frame

        AnsiString tmp = ""; //variavel temporaria que guarda o valor da conversão

        for (int i = 0; i < 4; i++) {
                tmp.sprintf("%02X ", vetor_serial[i]); // for que monta o frame da string
                teste += tmp;
        }
        edtMensagemTransmitir->Text = teste;

        valor_ponto = 0;
        for (contador_pontos = 0; contador_pontos <= 500 / 2; contador_pontos++)
        {
                valor_ponto++;                  // salva na variavel base valor do ponto
                Chart1->Series[0]->AddXY(valor_ponto,valor_ponto*2,"",clRed);
        }

                // for que escreve os valores na descida da onda
        for (contador_pontos = 0; contador_pontos <= 500 / 2; contador_pontos++)
        {
                valor_ponto--;                  // salva na variavel base valor do ponto
                Chart1->Series[0]->AddXY(valor_ponto,valor_ponto*2,"",clRed);
        }
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::bt_limpaClick(TObject *Sender)
{
         lt_pontos->Clear();
         Chart1->Series[0]->Clear();
         btnEnvia -> Enabled = False;
         btnInicializaSerial -> Enabled = False;

         ed_X->Enabled = false;
         ed_Y->Enabled = false;
         OK->Enabled = false;
```

```
        Chart1->Series[0]->AddXY(0,0,"",clRed);
        Chart1->Series[0]->AddXY(n_pontos,0,"",clRed);

        frmPrincipal->sp1_frequencia->Value = 125;

        RadioButton2->Checked = false;
        RadioButton3->Checked = false;
        RadioButton4->Checked = false;
        bt_arbitraria->Checked = false;
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::bt_arbitrariaClick(TObject *Sender)
{
        lt_pontos->Clear();
        btnInicializaSerial -> Enabled = False;
        btnEnvia -> Enabled = False;

        ed_X->Enabled = true;
        ed_Y->Enabled = true;
        OK->Enabled = true;

        Chart1->Visible = true;
        frmPrincipal->Width = 1100;
        frmPrincipal->Height = 515;

        lt_pontos->Visible = false;
        lt_pontos->Visible = true;
        bt_limpa->Visible = true;

        Chart1->Series[0]->Clear();      //zera valores

        freq_maxima = frmPrincipal->sp1_frequencia->Value;

        if (freq_maxima >= 2500) {
                n_pontos = 8000000 / freq_maxima;
        }
        else{
                n_pontos = 1000000 / freq_maxima;
        }

        lt_pontos->Items->Add("00;00");
        lt_pontos->Items->Add((AnsiString)n_pontos+";00");

        Chart1->Series[0]->AddXY(0,0,"",clRed); // define ponto inicial (0;0).
        Chart1->Series[0]->AddXY(n_pontos,0,"",clRed); // define ponto final(freq_maximo;0).

        vetor_serial[0] = 0xCC;
        vetor_serial[1] = (freq_maxima >> 16);
        vetor_serial[2] = (freq_maxima >> 8);
        vetor_serial[3] = freq_maxima;

        teste = ""; // zera a string antes de montar o frame
        AnsiString tmp = ""; //variavel temporaria que guarda o valor da conversão

        for (int i = 0; i < 4; i++) {
                tmp.sprintf("%02X ", vetor_serial[i]); // for que monta o frame da string
                teste += tmp;
        }
        edtMensagemTransmitir->Text = teste;
}
```

```
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::CompilaClick(TObject *Sender)
{
        qtd_coordenadas = lt_pontos->Items->Count;

        if (qtd_coordenadas >=3) // tests a qtd minina de pontos
        {
                int cont = 0;
        for(int i=0 ; i < lt_pontos->Items->Count; i++)        //copia os valores dos pontos
                {
                        AnsiString linha = lt_pontos->Items->Strings[i];
                        AnsiString x = linha.SubString(1,linha.AnsiPos(";")-1);
                        AnsiString y = linha.SubString(linha.AnsiPos(";")+1,3); string(x;y).
                        ys[StrToInt(x)] = StrToInt(y); //Salva o valor y na posicao x do vetor
                        xs[cont++] = StrToInt(x); // Salva em um vetor os valores de x.
                        }

                        for(int j=0;j<(lt_pontos->Items->Count)-1;j++)
                                if(xs[j] > xs[j+1])                    //testa se o ponto anterior
                        {
                                        unsigned short int temp = xs[j];
                                        xs[j] = xs[j+1];
                                        xs[j+1] = temp;
                        }
                        ShowMessage("Captura de Pontos Realizada com Sucesso");
                        btnInicializaSerial -> Enabled = true;
                        btnEnvia -> Enabled = true;
                }
                else
                  ShowMessage("A quantidade de coordenadas deve ser maior ou igual a 3");
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::OKClick(TObject *Sender)
{
                if (StrToInt (ed_X->Text) <= n_pontos && StrToInt (ed_Y->Text) <= 500 ) {
// testa se esta nas margens
                  Chart1->Series[0]->AddXY(StrToInt(ed_X->Text),StrToInt(ed_Y-
>Text),"",clRed);
                // Adiciona os pontos digitados no gráfico.
                if (teste_ponto == StrToInt (ed_X->Text))
                {
                        teste_ponto = teste_ponto + 1;
                        lt_pontos->Items->Add((AnsiString)teste_ponto+";"+ed_Y->Text);
                }
                else
                {
                        lt_pontos->Items->Add(ed_X->Text+";"+ed_Y->Text);
                }
                teste_ponto = StrToInt(ed_X->Text);
                // Adiciona os pontos digitados no Listbox.
                ed_X->Text = "";
                        ed_Y->Text = "";
                }
                else
       ShowMessage("Ponto não Valido. Fora da Escala");
}
//---------------------------------------------------------------------------
```

APÊNDICE C – FIRMWARE DO MICROCONTROLADOR

```
//*****************************************************************************
//*
//*  PROGRAMA:    Software Executivo Gerador de Funcoes
//*
//*
//*  DESCRICAO:   Modulo principal de geracao dos sinais arbitrarios e
//*                  sinais triangulares, senoidais e quadrados, alem disso
//*                  comunica com o supervisorio
//*
//*  PROGRAMADOR: Rafael da Silva Barboza
//*
//*  MICROPROCESSADOR: LPC2378
//*
//*****************************************************************************

// INCLUDES
#include <arch/nxp/lpc23xx.h>
#include "uart.h"
#include "math.h"
#include <stdlib.h>

// DEFINES
#define TRUE          1
#define FALSE         0
#define TAM           4
#define TAM_2         256

// VARIAVEIS GLOBAIS
int cont = 0;
int frequencia;
int estado = 0;
unsigned short int val_medido;              //variavel que salva os valores lidos na tabela
unsigned char val_baixo;                    //guarda os valores baixos do calculo 0x00XX
unsigned char val_alto;                     //guarda os valores altos do calculo 0xXX00
unsigned short int n_pontos;                // guarda a quantidade de pontos do sinal
unsigned short int n_passo;                 // guarda o passo entre cada ponto
unsigned short int valor_calculado = 0;     // recebe o valor do ponto calculado
unsigned short int valor_ponto;             //variavel que salva os valores lidos na tabela
unsigned short int valor_ponto_x[100];      //variavel valores das coordenadas em X
unsigned short int valor_ponto_y[100];      //variavel  valores das coordenadas em y

int flag_oscilador = 0;                     // guarda o valor do oscilador a ser usado
float frequencia_angular;                   // guarda frequencia angular

unsigned char buffer_serial[TAM];
unsigned char buffer_arbitraria[TAM_2];
int sucesso_recepcao = 0;
```

```
int main (void)
{
    int cpuclk;
    char buf[32];
    int pllm=12,cpud=12; //cpuclock = 12MHz*(pllmulti/cpudiv)
    char var;

    FIO0DIR = 0xFFFFFFFF;   /* porta 0 ligado como saida*/
    FIO1DIR = 0xFFFFFFFF;   /* porta 1 ligado como saida*/
    FIO3DIR = 0xFFFFFFFF;   /* porta 3 ligado como saida*/

    //incia timer 0, contador de milisegundos
    init_timer0();

    /* Configura o sistema de clock */
    cpuclk=PLL_Init(pllm,cpud);

    /* Configura a porta serial UART0 */
    UART0_Init(BAUDRATE);

    //desabilita latch's
    FIO1SET = (1 << 28);                    //liga OEL1
    FIO1SET = (1 << 29);                    //liga OEL2

    //desabilitacao da RAM 1
    FIO0SET = (1 << 17);                    //liga WE1
    FIO0SET = (1 << 14);                    //liga OE1

    //desbilitacao da RAM 2
    FIO0SET = (1 << 18);                    //liga OE2
    FIO0SET = (1 << 31);                    //liga WE2

    while(1)
    {
        if (sucesso_recepcao == FALSE){     // entra aqui ate a recepacao OK
                lerSerial();
        }

        if (sucesso_recepcao == TRUE)
        {
                var = buffer_serial[0]; // pega a onda

                frequencia = (buffer_serial[1] << 16) + (buffer_serial[2] << 8) +  buffer_serial[3];
                // monta valor da frequencia

                if (frequencia >= 2500)             // testa o oscilador q sera usado
                {
                        flag_oscilador = 20;
                        n_pontos = 8000000 / frequencia;
                }
                else
                {
                        flag_oscilador = 1;
                        n_pontos = 1000000 / frequencia;
                }
                switch(var)
                {
                        case 0xEE:
                                FIO1CLR = (1 << 26);        //seta reset manual(IO5)
                                monta_onda_triagular(frequencia,n_pontos);
```

```
                        FIO1SET = (1 << 19);          //seta reset manual(IO2)
                        FIO1CLR = (1 << 19);          //zera reset manual(IO2)
                        le_ram();       // le os valores em RAM e joga na saida

                    case 0xAA:
                        FIO1CLR = (1 << 26);          //seta reset manual(IO5)
                        monta_onda_quadrada(frequencia,n_pontos);

                        FIO1SET = (1 << 19);          //seta reset manual(IO2)
                        FIO1CLR = (1 << 19);          //zera reset manual(IO2)

                        le_ram();       // le os valores em RAM e joga na saida

                    case 0xDD:
                        FIO1CLR = (1 << 26);          //seta reset manual(IO5)
                        monta_onda_senoide(frequencia,n_pontos);

                        FIO1SET = (1 << 19);          //seta reset manual(IO2)
                        FIO1CLR = (1 << 19);          //zera reset manual(IO2)

                        le_ram();// le os valores em RAM e joga na saída

                    case 0xCC:
                        FIO1CLR = (1 << 26);          //seta reset manual(IO5)

                        monta_onda_arbitraria();

                        FIO1SET = (1 << 19);          //seta reset manual(IO2)
                        FIO1CLR = (1 << 19);          //zera reset manual(IO2)

                        le_ram();       // le os valores em RAM e joga na saida
                    }
                }
            }
}

////////////////////////////////////////////////////////////////////////
/**
 * Rotina que monta o sinal senoide, recebendo como paramentro a frequencia a
 * ser gerada ja escrevendo em RAM os pontos calculados
 * Autor:
 */
////////////////////////////////////////////////////////////////////////
int monta_onda_senoide(int freq,int numero_pontos )
{
float contador_pontos_seno;
float variavel = 722;

    do
     {
        variavel++;
        frequencia_angular = ((2 * 3.1416 * variavel) / numero_ponto;
        frequencia_angular = sin(frequencia_angular);

        frequencia_angular = (frequencia_angular * (0x7FFE / 2))+0xBB79;

        valor_ponto = frequencia_angular;

     }while (valor_ponto >= 32768);
```

```
      for(contador_pontos_seno = variavel; contador_pontos_seno <= (variavel
      +numero_pontos); contador_pontos_seno++)
      {
         frequencia_angular = ((2 * M_PI * contador_pontos_seno) / numero_pontos);
         frequencia_angular = sin(frequencia_angular);
         // calcula o seno de cada ponto
         frequencia_angular = (frequencia_angular * (0x7FFE / 2))+0xBB79;
         valor_ponto = frequencia_angular;                    // ajusta o offset
         escreve_ram();
       }
      valor_ponto = 0x7FFF;
      escreve_ram();                                  // escreve valor do terminador ram
}
////////////////////////////////////////////////////////////////////////////
/**
 * Rotina que monta o sinal trangular, recebendo como paramentro a frequencia a
 * ser gerada ja escrevendo em RAM os pontos calculados
 * Autor:
 */
////////////////////////////////////////////////////////////////////////////
int monta_onda_triagular(int freq,int numero_pontos )
{
int contador_pontos = 0;                              // conta da quantidade de
pontos
float n_ptos;
float maximo = 65536;
float oscila_1mega = 1000000;
float oscila_20mega = 8000000;

        if (freq <= 2500)                             // testa se calcularos pontos para o
        {                                             // oscilador e 1 ou 20 mega.
                n_ptos = maximo / oscila_1mega;       // calcula o passo de cada ponto
                n_ptos *= freq;
                n_passo = n_ptos;
        }
        else
        {
                n_ptos = maximo / oscila_20mega;
                n_ptos *= freq;
                n_passo = n_ptos;
        }
        // for que escreve os valores na subida da onda
        for (contador_pontos = 0; contador_pontos < numero_pontos / 2; contador_pontos++)
        {
                valor_ponto = valor_calculado + 0x7FFE ;
                escreve_ram();                        // escreve valor na ram
                valor_calculado = (valor_calculado + n_passo);// da o passo para o proximo
        }

        // for que escreve os valores na descida da onda
        for (contador_pontos = 0; contador_pontos < numero_pontos / 2; contador_pontos++)
        {
                valor_ponto = valor_calculado + 0x7FFE ;
                escreve_ram();                                // escreve valor na ram
                valor_calculado =  (valor_calculado - n_passo);
        }
        valor_ponto = 0x7FFF;
        escreve_ram();                                // escreve valor do terminador ram
}
```

```
////////////////////////////////////////////////////////////////////////
/**
 * Rotina que monta o sinal quadrado, recebendo como paramentro a frequencia a
 * ser gerada ja escrevendo em RAM os pontos calculados
 * Autor:
 */
////////////////////////////////////////////////////////////////////////
int monta_onda_quadrada(int freq,int numero_pontos)
{
        int contador_pontos = 0;                        // conta da quantidade de pontos

        // for que escreve os valores positivos da onda
        for (contador_pontos = 0; contador_pontos < numero_pontos / 2; contador_pontos++)
        {
                valor_ponto = 0xFEEE;                   // salva na variavel base valor do ponto
                                                        // a ser enviado a ram
                escreve_ram();                          // escreve valor na ram
        }

        // for que escreve os valores negativos da onda
        for (contador_pontos = 0; contador_pontos < numero_pontos / 2; contador_pontos++)
        {
                valor_ponto = 0x7FFE;                   // salva na variavel base valor do ponto

                                                        // a ser enviado a ram
                escreve_ram();                          // escreve valor na ram

        }
        valor_ponto = 0x7FFF;
        escreve_ram();                                  // escreve valor do terminador ram
}

////////////////////////////////////////////////////////////////////////
/**
 * Rotina que monta o sinal arbitrario, recebendo como paramentro a frequencia a
 * ser gerada ja escrevendo em RAM os pontos calculados
 * Autor:
 */
////////////////////////////////////////////////////////////////////////
int monta_onda_arbitraria(void)
{
        int contador_pontos = 0;                        // conta da quantidade de pontos
        int cont = 0;
        int i;
        unsigned short int x1,x0,y1,y0,z;
        float a,b;


        for (contador_pontos = 0; contador_pontos < buffer_arbitraria[0] /
4;contador_pontos++)
        {
                valor_ponto_x[contador_pontos] = (buffer_arbitraria[cont = cont + 1] << 8) +
                                                 (buffer_arbitraria[cont = cont + 1]);

        }
        cont = buffer_arbitraria[0] / 2;

        /* pega o restante dos bytes no buffer de recepcao*/
        for (contador_pontos = 0; contador_pontos < buffer_arbitraria[0] / 4;
contador_pontos++)
```

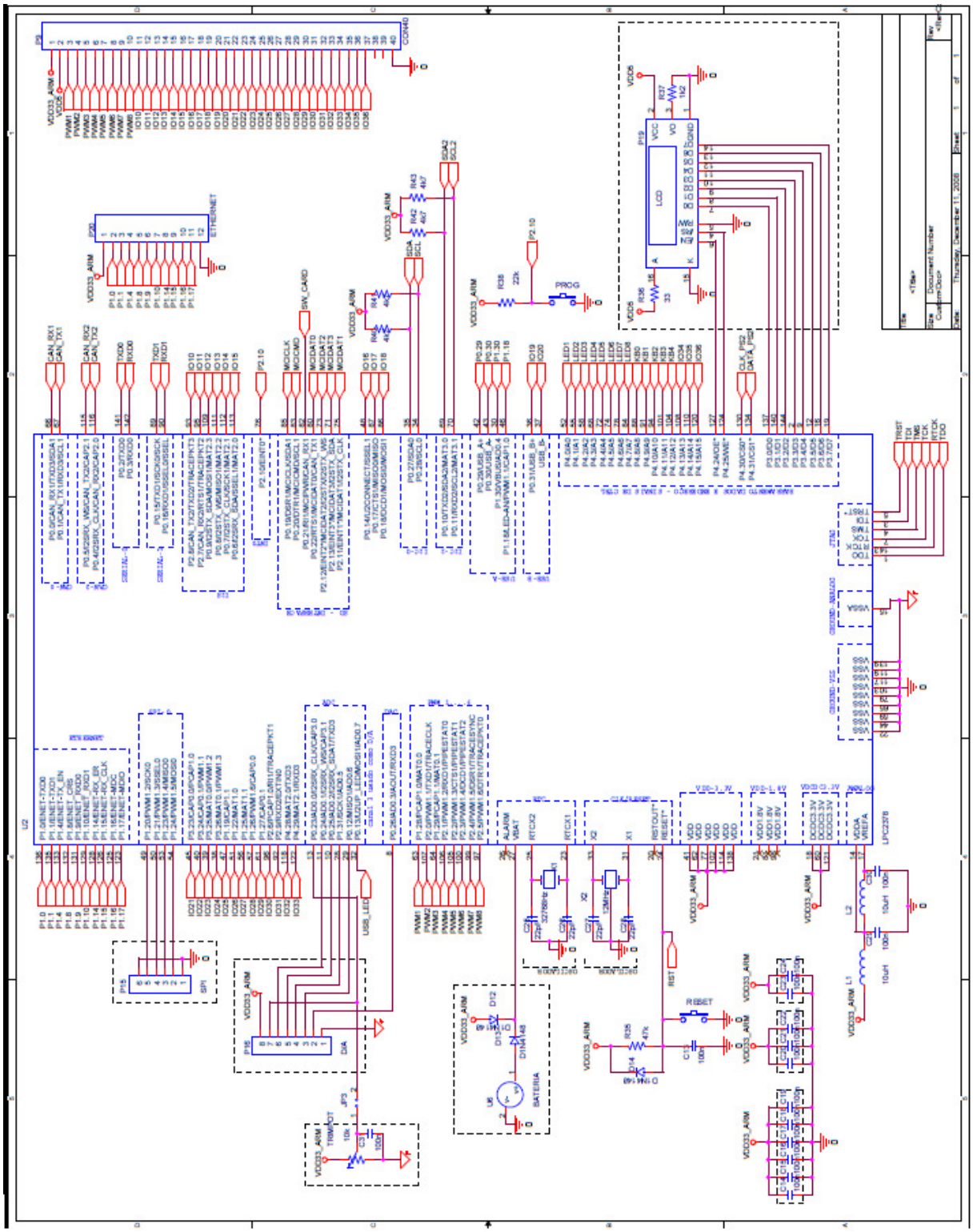
```
        {
                valor_ponto_y[contador_pontos] = (buffer_arbitraria[cont = cont + 1] << 8) +
                                                 (buffer_arbitraria[cont = cont + 1]);

                // ajusta o dado recebido para o valor de geracao
                valor_ponto_y[contador_pontos] = (valor_ponto_y[contador_pontos] *
                                                 (32767 / 500) + 32767);
        }

        for(i = 0; i < (buffer_arbitraria[0] / 4) - 1; i++)          //interpolacao
        {
                x0=valor_ponto_x[i];
                y0=valor_ponto_y[i];                    //Define os pontos limites das retas

                x1=valor_ponto_x[i+1];
                y1=valor_ponto_y[i +1];

                a=(float)(y1-y0)/(x1-x0);
                b=(float)y1-(a*x1);              //Cálculo dos pontos intermediários da reta.

                for(z = x0+1 ; z < x1 ; z++)
                {
                        valor_ponto=(a*z+b);            // monta o vetor com os pontos
                        escreve_ram();
                }
         }
        valor_ponto = 0x7FFF;
        escreve_ram();                                  // escreve valor do terminador ram
}
////////////////////////////////////////////////////////////////////////////
/**
 * Rotina que escreve em RAM os valores dos pontos ja calculador
 * Autor:
 */
////////////////////////////////////////////////////////////////////////////
int escreve_ram(void)
{
        //zera osciladores
        FIO0CLR = (1 << 9);                             //zera oscilador 1M
        FIO0CLR = (1 << 8);                             //zera oscilaodor 20M

        val_baixo = valor_ponto & 0x00FF;               //mascara para somente os valores
baixos
        val_alto = (valor_ponto & 0xFF00) >> 8;         // mascara para pegar somente os altos

        FIO3MASK = 0x000000FF;                          // nao mudar status de D0 a D7
        //pulsa o clock
        FIO3SET = (1 << 24);                            //liga I01 do hardware
        FIO3CLR = (1 << 24);                            //desliga I01 do hardware

        //habilitacao da RAM 1
        FIO0CLR = (1 << 17);                            //zera WE1
        FIO0SET = (1 << 14);                            //liga OE1

        //habilita latch 1
        FIO0SET = (1 << 7);                             //liga LE1
        FIO1CLR = (1 << 28);                            //zera OEL2

        FIO3MASK = ~0x000000FF;                         // escrita apenas nos dados na porta
        FIO3PIN = val_baixo;
```

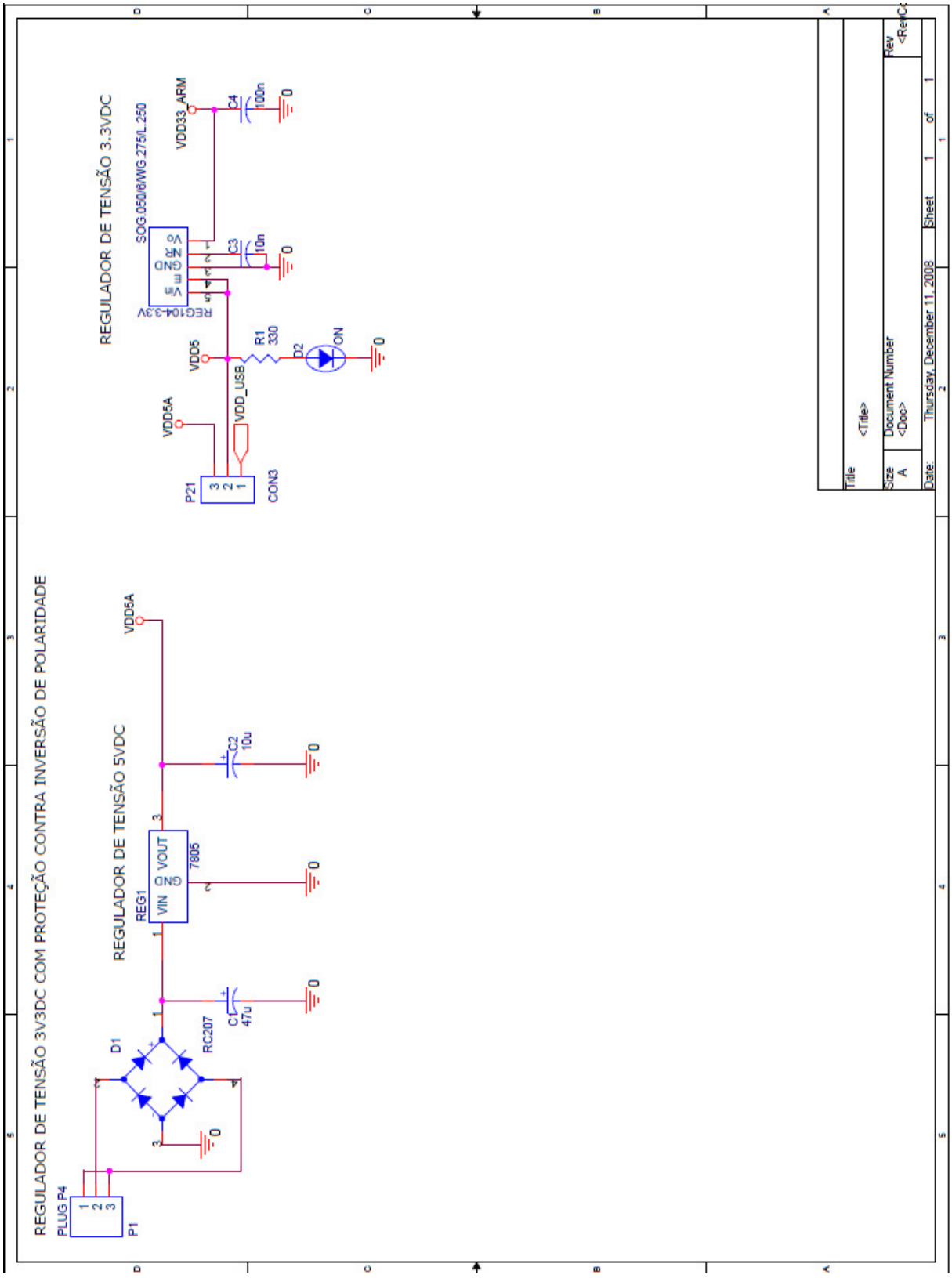
```
	//desabilitacao da RAM 1
	FIO0SET = (1 << 17);			//liga WE1
	FIO0SET = (1 << 14);			//liga OE1

	//desabilita latch 1
	FIO0CLR = (1 << 7);			//zera LE1
	FIO1SET = (1 << 28);			//liga OEL1

	//habilitacao da RAM 2
	FIO0SET = (1 << 18);			//liga OE2
	FIO0CLR = (1 << 31);			//zera WE2

	//habilita latch 2
	FIO0SET = (1 << 6);			//liga LE2
	FIO1CLR = (1 << 29);			//zera OEL2

	FIO3PIN = val_alto;			// Permite escrita apenas na porta

	//desbilitacao da RAM 2
	FIO0SET = (1 << 18);			//liga OE2
	FIO0SET = (1 << 31);			//liga WE2

	//desabilita latch 2
	FIO0CLR = (1 << 6);			//zera LE2
	FIO1SET = (1 << 29);			//liga OEL2
}
////////////////////////////////////////////////////////////////////////////
*/
/**
 * Rotina que le a RAM os valores dos pontos ja calculados
 * Autor:
 */
////////////////////////////////////////////////////////////////////////////
int le_ram(void)
{
	//	int contador;
	FIO3MASK = ~0x000000FF;
	FIO3PIN = 0x00;				// Permite escrita apenas na porta
	FIO3MASK = 0x000000FF;			// Nao permite escrita

	//desabilita latch's
	FIO1SET = (1 << 28);			//liga OEL1
	FIO1SET = (1 << 29);			//liga OEL2

	//habilitacao da RAM 1
	FIO0SET = (1 << 17);			//liga WE1
	FIO0CLR = (1 << 14);			//zera OE1

	//habilitacao da RAM 2
	FIO0CLR = (1 << 18);			//zera OE2
	FIO0SET = (1 << 31);			//liga WE2

	FIO1SET = (1 << 26);			//seta reset manual(IO5)

	// testa se deve setar o oscilador de 1 ou 20 mega.
	if (flag_oscilador == 1)
	{
		FIO0SET = (1 << 9);		//liga oscilador 1M
	}
```

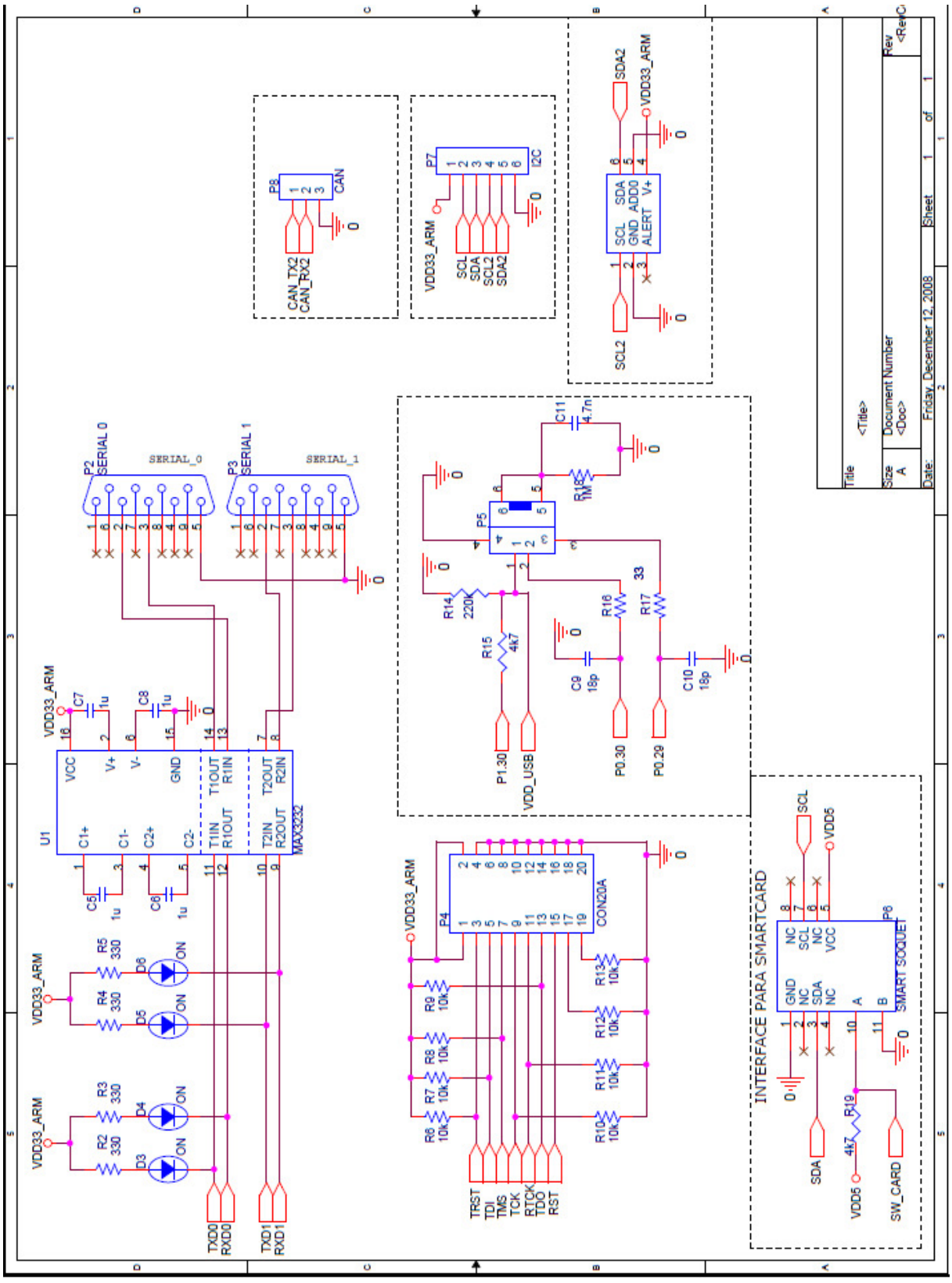
```
        else
        {
                FIO0SET = (1 << 8);                     //liga oscilaodor 20M
        }
        // le a ram initerruptamente
        while(1);
}

void init_timer0(void) //funcao para inicializar o timer 1.
{
        T0TCR =         0;                  /* Desabilita T0 */
        /* Prescaler para incrementar o T0TC 1000 vezes por segundo */
        T0PR = CRYSTALFREQ/8- 1;
        T0TCR =         2;                  /* Inicializa T0 */
        T0TCR =         1;                  /* Habilita T0 */
}
////////////////////////////////////////////////////////////////////////////
*/
/**
 * Rotina que le os dados recebidos da serial
 * Autor:
 */
////////////////////////////////////////////////////////////////////////////
void lerSerial(void)
{
        char lido;
        lido = UART0getchar();
        if(estado == 0 && lido == 0xAA)             // primeiro byte cabecalho
        {
                UART0putchar(0x55);                 // resposta
                estado = 1;
                sucesso_recepcao = FALSE;
        }
        else if((estado == 1) && (lido == 0xAA))    // segundo byte cabecalho
        {
                UART0putchar(0x55);                 // resposta
                estado = 2;
                cont = 0;
                sucesso_recepcao = FALSE;

                FIO0CLR = (1 << 9);                 //desliga oscilador 1M
                FIO0CLR = (1 << 8);                 //desliga oscilaodor 20M
        }
        else if(estado == 2)                        // recebe o frame de 4 bytes, aonde:
        {                                           // 0 : forma de onda
                if (lido != 0xFF)
                {
                        buffer_serial[cont] = lido; // 1,2 e 3 = valor da frequencia
                        cont++;
                        sucesso_recepcao = FALSE;

                        if(cont == TAM)             // testa  4 bytes das ondas simples
                        {
                                if (buffer_serial[0] == 0xCC)  // teste se a onda e arbitraria
                                {
                                        estado = 3;     // se for cai no proximo estado
                                        sucesso_recepcao = FALSE;
                                        cont = 0;
                                        lido = 0xBB;            // mandar o terminador
```

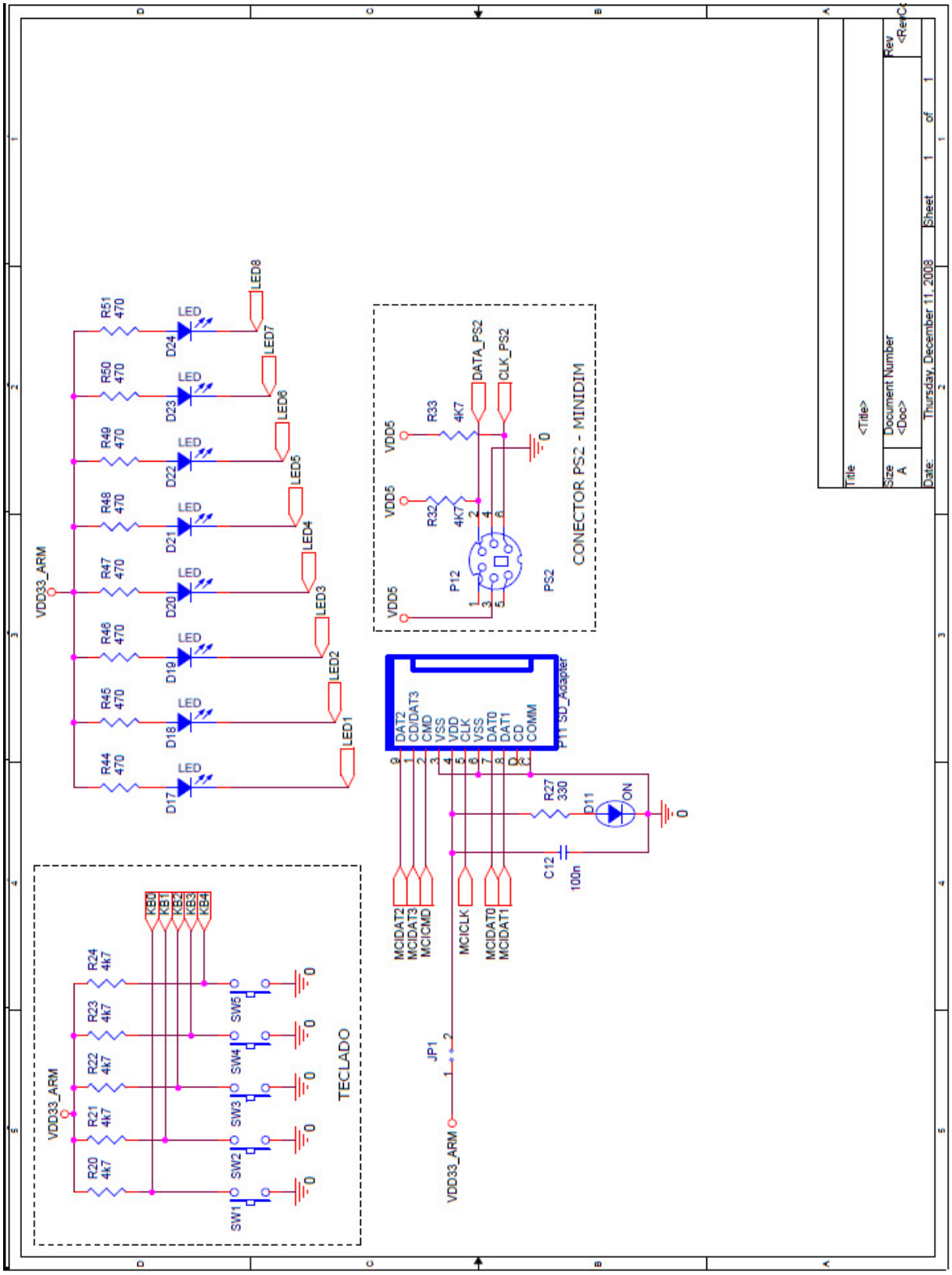
```
                                        UART0putchar(lido);
                                }
                                else
                                {
                                        estado = 0;             // se nao gera os sinais
                                        sucesso_recepcao = TRUE;
                                        lido = 0xCC;            // mandar o terminador

                                        UART0putchar(lido);
                                }
                        }
                }
        }
        else if(estado == 3 )
        {
                if (lido != 0xFF)
                {
                        buffer_arbitraria[cont] = lido; // recebe o valor da onda

                        cont++;
                        if(cont == buffer_arbitraria[0]) // testa se atingiu a quantidade
                        {
                                estado = 0;                             // fim de recepcao
                                sucesso_recepcao = TRUE;
                                lido = 0xCC;            // mandar o terminador
                                UART0putchar(lido);
                        }
                }
        }
}

/* Estas rotinas sao chamados pelo crt.s.
Devem existir, mas ainda nao estao sendo usadas */
void UNDEF_Routine(){}
void SWI_Routine(){}
void FIQ_Routine(){}
```

ANEXO A – ESQUEMÁTICO DO KIT DE DESENVOLVIMENTO




REGULADOR DE TENSÃO 3V3DC COM PROTEÇÃO CONTRA INVERSÃO DE POLARIDADE
PLUG P4
P1
D1
RC207
C1
47u
REGULADOR DE TENSÃO 5VDC
REG1
VIN
GND
VOUT
7805
C2
10u
VDD5A
REGULADOR DE TENSÃO 3.3VDC
SOG.050/6/WG.275/L.250
REG104-3.3V
Vo
NR
GND
Vi
C3
10n
C4
100n
VDD33_ARM
VDD5
R1
330
D2
ON
VDD_USB
P21
CON3
Title
<Title>
Size
A
Document Number
<Doc>
Rev
<RevC
Date:
Thursday, December 11, 2008
Sheet
1
of
1


P8
CAN
CAN_TX2
CAN_RX2
P7
I2C
VDD33_ARM
SCL
SDA
SCL2
SDA2
SDA2
VDD33_ARM
SCL
SDA
ADD0
V+
GND
ALERT
SCL2
P2
SERIAL 0
SERIAL_0
P3
SERIAL 1
SERIAL_1
C11
4.7n
R18
1M
P5
R14
220k
R15
4k7
R16
33
R17
33
C9
18p
C10
18p
P1.30
VDD_USB
P0.30
P0.29
VDD33_ARM
C7
1u
C8
1u
U1
VCC
V+
V-
GND
T1OUT
R1IN
T2OUT
R2IN
C1+
C1-
C2+
C2-
T1IN
R1OUT
T2IN
R2OUT
MAX3232
C5
1u
C6
1u
R5
330
R4
330
R3
330
R2
330
D6
D5
D4
D3
ON
TXD0
RXD0
TXD1
RXD1
P4
CON20A
R9
10k
R8
10k
R7
10k
R6
10k
R13
10k
R12
10k
R11
10k
R10
10k
TRST
TDI
TMS
TCK
RTCK
TDO
RST
INTERFACE PARA SMARTCARD
SCL
VDD5
NC
SCL
NC
VCC
GND
NC
SDA
NC
P6
SMART SOQUET
A
B
R19
4k7
SDA
VDD5
SW_CARD
Title
<Title>
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Friday, December 12, 2008
Sheet
1
of
1


VDD33_ARM
R51 470
R50 470
R49 470
R48 470
R47 470
R46 470
R45 470
R44 470
LED
D24
D23
D22
D21
D20
D19
D18
D17
LED8
LED7
LED6
LED5
LED4
LED3
LED2
LED1
CONECTOR PS2 - MINIDIM
VDD5
R33 4K7
R32 4K7
DATA_PS2
CLK_PS2
P12
PS2
P11 SD_Adapter
DAT2
CD/DAT3
CMD
VSS
VDD
CLK
VSS
DAT0
DAT1
CD
COMM
R27 330
D11
ON
C12 100n
MCIDAT2
MCIDAT3
MCICMD
MCICLK
MCIDAT0
MCIDAT1
JP1
TECLADO
KB0
KB1
KB2
KB3
KB4
R24 4k7
R23 4k7
R22 4k7
R21 4k7
R20 4k7
SW6
SW4
SW3
SW2
SW1
Title
<Title>
Size A
Document Number
<Doc>
Rev
<RevCo
Date: Thursday, December 11, 2008
Sheet 1 of 1

ANEXO B – CERTIFICADO DE CALIBRAÇÃO DO OSCILOSCÓPIO TDC-2014C

Tektronix (China) Co. Ltd.
1227 Chuan Qiao Road
Pudong New District, Shanghai, China 201206

TRACEABLE CALIBRATION
TDS2014C, C014057
Cal Date: 06-JAN-2012
Cal Due: Lab: Tek
Tektronix

Tektronix

CERTIFICATE OF TRACEABLE CALIBRATION

Certificate No: PCXPS8Q2WW **Revision:** 00
Manufacturer: Tektronix **Model:** TDS2014C **Serial No:** C014057
Description: Oscilloscope; 100 MHz, 4 Channel **Temperature:** 24 °C **Humidity:** 22 %
Cal Date: 06-JAN-2012 **Date Placed In Service:** _____* **Due Date:** _____*

*Optional customer entry fields: The "Due Date" may be established by adding the calibration interval to the "Date Placed In Service." Recommended calibration intervals, for Tektronix products, are available on the Tektronix website at: http://www.tek.com/service/product-support/

INSTRUMENT CONDITION:

Received: Not applicable, initial calibration
Returned: In tolerance, within published measurement specification

Tektronix certifies the above instrument has been calibrated using standards traceable to the PRC National Institute of Metrology (NIM) and/or other National Metrology Institutes (NIST, NPL, PTB) that are linked to the international system of units (SI). The policies and procedures used for the calibration of this product are based on ISO/IEC 17025:2005.
This certificate shall not be reproduced, except in full, without the written approval of the calibration facility.

CALIBRATION PROCEDURE:

MANIFEST:Product_Oscilloscopes_TDS1K2K-SC_Full VERSION:81

CALIBRATION EQUIPMENT USED:

MANUFACTURER/MODEL	MODEL DESCRIPTION	ID NUMBER	DUE DATE
Fluke 9100	Calibrator	CT0038	29-Jan-2012
Keithley 2000	Digital Multimeter	CT0008	17-Jun-2012

Issued By: **Certified By:** Zhengxing Wang

Quality Director: Cui, Hu Wa **Date Issued:** 06-JAN-2012

ISO 9001 Registered Quality System

TDS2014C C014057 Page 1 of 1 001-1387-01