Universidade de São Paulo
Escola de **E**ngenharia de **S**ão **C**arlos
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

TRATAMENTO DE MÚLTIPLOS SINAIS UTILIZANDO SISTEMA RTOS EMBARCADO

Autor:

Vitor Fressatti Mangueira

Orientador:

Prof. Dr. Carlos Dias Maciel

São Carlos, Novembro de 2013

VITOR FRESSATTI MANGUEIRA

TRATAMENTO DE MÚLTIPLOS SINAIS UTILIZANDO SISTEMA RTOS EMBARCADO

Trabalho de Conclusão de Curso
apresentado à Escola de Engenharia de São
Carlos da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Sistemas de Energia e Automação

ORIENTADOR: Prof. Dr. Carlos Dias Maciel

São Carlos
2013

Mangueira, Vitor Fressatti
M277t Tratamento de múltiplos sinais utilizando sistema RTOS embarcado / Vitor Fressatti Mangueira; orientador Carlos Dias Maciel. São Carlos, 2013.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Aquisição de dados. 2. FreeRTOS. 3. Microcontrolador. 4. RTOS. 5. Sistema Operacional Embarcado. I. Título.

FOLHA DE APROVAÇÃO

Nome: Vitor Fressatti Mangueira

Título: "Tratamento de múltiplos sinais utilizando sistema RTOS embarcado"

Trabalho de Conclusão de Curso defendido e aprovado em 19 / 11 / 2013 ,

com NOTA 9,0 (nove, zero), **pela Comissão Julgadora:**

Prof. Associado Carlos Dias Maciel - (Orientador - SEL/EESC/USP)

Mestre Wagner Endo - (Doutorando - SEL/EESC/USP)

Mestre Giovana Yuko Nakashima - (Doutoranda - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

DEDICATÓRIA

Dedico este trabalho
a minha esposa Camila que deu todo o incentivo e apoio possível
assim como suportou todas as fases a que passamos.

AGRADECIMENTOS

Agradeço a todos meus familiares pelo apoio e incentivo na busca de meus objetivos.

Ao meu orientador, Prof. Dr. Carlos Dias Maciel, pela compreensão e apoio durante a elaboração deste trabalho. Também, ao Laboratório de Processamento de Sinais (LPS) da Universidade de São Paulo (USP) pela colaboração durante este período. Em especial a Tales Santini por ceder parte do material usado e pelo auxílio prestado quanto ao conhecimento compartilhado.

A todos os amigos com quem tive o prazer de conviver durante o período da graduação que foram muito importantes e todos que contribuíram de alguma forma direta ou indiretamente, fica aqui minha gratidão.

Sumário

Capítulo 1 - Introdução 15
1.1 Motivação 15
1.2 Objetivos 16
1.3 Novos conhecimentos adquiridos 16
1.4 Corpo do trabalho 17
Capítulo 2 – Referencial Teórico 19
2.1 Sistema Operacional de Tempo Real (RTOS) 19
2.1.1 Definição 19
2.1.2 Gerenciamento de tarefas 20
2.1.3 Criticidade 21
2.1.4 Escalonador 22
2.1.5 Partilha de recursos 24
2.1.5.1 Mascaramento temporário (desabilitar interrupções) 25
2.1.5.2 Semáforos binários 25
2.1.5.3 Mensagens trocadas 26
2.1.6 FreeRTOS 26
2.2 Teorema de Nyquist 28
2.3 Conversores A/D e D/A 31
2.3.1 Conversores A/D 31
2.3.1.1 Paralelo 32
2.3.1.2 Aproximação sucessiva 35
2.3.1.3 Contador 36
2.3.1.4 Integrador 38
2.3.2 Conversores D/A 41
2.3.2.1 Resistor ponderado 41
2.3.2.2 Rede R-2R 42
2.3.2.3 PWM 43
Capítulo 3 - Materiais e métodos 45
3.1 Ferramentas 45
3.1.1 Explorer16^{BR} V1.1 46
3.1.2 ICD2^{BR} V1.1 48
3.1.3 PICTail^{PROTO} V1.0 49
3.1.4 Conversor USB-RS232 50
3.1.5 MPLAB 51
3.1.6 RealTerm 53

3.1.7 LabVIEW 54
3.1.8 Linguagem C 55
3.2 Casos de testes 55
3.2.1 1ª Etapa - Depuração 55
3.2.2 2ª Etapa – Sistema operacional 56
3.2.3 3ª Etapa – Aplicação 56
Capítulo 4 – Resultados 59
4.1 1ª Etapa - Depuração 59
4.2 2ª Etapa – Sistema operacional de tempo real 60
4.3 3ª Etapa – Aplicação 62
Capítulo 5 – Discussão e Conclusão 65
Referências 69

Índice de figuras

Figura 1 - Camadas de interação de um sistema operacional intermediando o hardware do equipamento até a ação final do usuáio 20
Figura 2 - Função do sistema operacional no gerenciamento das tarefas para uso do processamento 21
Figura 3 - Diferença do uso do resultado de uma tarefa após violar o tempo máximo para execução para sistemas de propósito geral, Crítico e Não-Crítico 22
Figura 4 – Atuação do escalonador no gerenciamento da fila de pronto 23
Figura 5 – Conceito básico sobre o Princípio da Amostragm mostrando a formação do sinal discreto gerado a partir de um análogico 28
Figura 6 – Demonstração de efeitos para diferentes frequências de amostragem 30
Figura 7 - Conversor A/D tipo Paralelo ("Flash") 33
Figura 8 - Conversor A/D tipo Aproximação Sucessiva 35
Figura 9 - Conversor A/D tipo contador 37
Figura 10 - Conversor A/D tipo integrador com rampa simples 39
Figura 11 - Conversor A/D tipo integrador com rampa dupla 40
Figura 12 - Conversor D/A de 4 bits a resistores ponderados 42
Figura 13 - Conversor D/A de 4 bits com rede R-2R 42
Figura 14 - Sinal PWM 43
Figura 15 - Kit escolhido: Explorer16^{BR} (A) e o programador ICD2^{BR} (B), ambos da empresa Mosaico®. Conversor USB-RS232 da GigaWare® (C) 46
Figura 16 - Explorer16^{BR} V1.1 47
Figura 17 - Plugin Explorer16BR PIC32MX460F512L-80I/PT USB 47
Figura 18 - Programador ICD2^{BR} V1.1 48
Figura 19 - PICTail^{PROTO} V1.0 49
Figura 20 - Conversor USB-RS232 GIGAWARE 50
Figura 21 - MPLAB IDE v8.92 Utilizado apenas para programar o dispositivo. 51
Figura 22 - MPLAB X IDE v1.90 Utilizado para desenvolver e compilar o código52
Figura 23 - RealTerm: Serial Capture Program 2.0.0.70 53
Figura 24 - Exemplo de telas do LabVIEW 54
Figura 25 - Leitura 25 no A/D, comunicação serial e LED's 59
Figura 26 - Leitura 156 no A/D, comunicação serial e LED's 60
Figura 27 - Menu com FreeRTOS para valor mínimo do A/D 61
Figura 28 - Menu com FreeRTOS para valor de máximo do A/D 62
Figura 29 - Sinal gerado por operação manual de um trimpot 63
Figura 30 - Exemplo de comunicação utilizando o protocolo definido para identificação de múltiplos sinais 63
Figura 31 - Leitura de 3 sinais senoidais defasados entre si 64

RESUMO

MANGUEIRA, V. F. **Tratamento de múltiplos sinais utilizando sistema RTOS embarcado.** 2013. Trabalho de conclusão de curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2013.

Vários aplicações necessitam de uma análise de sinais mais apurada. Este trabalho propõe a elaboração de um dispositivo de aquisição de dados para tratamento de múltiplos sinais. Utilizando-se do sistema operacional FreeRTOS embarcado, novas vantagens são adicionadas aprimorando sua utilização, como: gerenciamento de uso do hardware, excelente algoritmo para o escalonador, facilidade de manutenção e portabilidade, inserção de novas tarefas adicionando recursos ao dispositivo (operações, filtros, transformadas, etc), entre outras. Também o fácil manuseio na definição do número total de sinais a serem analisados e inclusão de protocolos para comunicação serial, se necessário.

Palavras-chave: Aquisição de dados, FreeRTOS, Microcontrolador, RTOS, Sistema Operacional Embarcado.

ABSTRACT

MANGUEIRA, V. F. **Treatment of multiple signals using embedded RTOS system.** 2013. Trabalho de conclusão de curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2013.

Several applications require a more refined signal analysis. This work proposes the development of a data acquisition device for processing multiple signals. Using the embedded operation system FreeRTOS, new benefits are added enhancing their use, such as: managing hardware usage, excellent algorithm for the scheduler, maintainability and portability, inserting new tasks by adding resources to the device (operations, filters, transformed functions, etc.), and more. Also, the easy handling in defining the total number of signals to be analyzed and inclusion of protocols for serial communication, if needed.

Keywords: Data Acquisition, Embedded Operational System, FreeRTOS, Microcontroller, RTOS

Capítulo 1 - Introdução

Várias aplicações se destinam à leitura de sinais de diversos sensores, por exemplo um carro com recursos modernos, que precisa receber múltiplas informações ao mesmo tempo como temperatura e oxigenação do motor, mistura de combustível, rotação das rodas, velocidade, ângulo da direção, freios, entre outras.

Estes sinais podem possuir características diversas como formatos de ondas distintos, amplitude, frequência, protocolos diferenciados, etc. Ao longo do desenvolvimento, durante uma manutenção ou mesmo em casos específicos da aplicação torna-se necessário analisar estes sinais com mais detalhes, criar um *log* (registro ao longo do tempo do histórico de atividades) ou filtrá-los e/ou tratá-los com algum algoritmo determinado.

Um dispositivo de aquisição de dados que consiga ser genérico a ponto de tratar vários sinais independentemente de sua natureza e ainda garantir que, mesmo em situações limite, consiga não ter qualquer tipo de perda pode ter um custo elevado. Isto se dá pelo fato de ser obrigado a possuir um *hardware* poderoso que consiga garantir a aquisição de sinais com frequência muito alta e um armazenamento suficientemente grande. Também deve-se considerar um tempo de desenvolvimento e principalmente testes a ponto de assegurar sua generalização.

1.1 Motivação

Este trabalho propõe-se a elaborar um sistema capaz de capturar e tratar sinais com diferentes características, de forma eficiente, fazendo com que não necessite de um *hardware* muito robusto e que seja de fácil utilização. Desta forma é possível reduzir o custo da ferramenta. Também propõe-se a utilizar plataformas *open source* como base (tanto *hardware* quanto *firmware*) e disponibilizar o código criado para a aplicação visando possíveis futuras alterações ou melhorias no mesmo.

Para auxiliar no desenvolvimento e principalmente na futura melhoria ou utilização, o dispositivo funcionará em cima de um sistema operacional de tempo real

(RTOS). Assim, os recursos do hardware serão administrados pelo sistema operacional assim como a portabilidade será facilitada. A inserção de novas funções será de fácil acesso no código e não implicará em riscos de alterar alguma função já pré-estabelecida.

Para o desenvolvimento deste trabalho será utilizado um *hardware* difundido no mercado e de fácil aquisição, o *kit* PIC Explorer16^{BR} da MOSAICO®, versão nacional e licenciada do *kit* original Explorer16 da Microchip®. Com o foco sendo a aplicação desenvolvida, ou seja, em sua maior parte o código em si, o mesmo terá que possuir a maioria das rotinas sem vínculo ao *hardware*, bastando alterar pontos específicos que os direcionam e configurações gerais.

1.2 Objetivos

Este projeto tem por objetivos principais:

- Criar um sistema de captura dos sinais de modo a armazenar o dado presente em cada sinal analisado de forma cíclica, na maior velocidade necessária, para não haver riscos de perdas.
- Enviar apenas os dados relevantes para um dispositivo externo, como no caso deste trabalho, um computador para análise de dados e geração de gráficos. Porém, estes dados poderão ser enviados a qualquer outro dispositivo desde que atenda o mesmo padrão de comunicação.
- Todo o sistema será baseado no gerenciamento de tarefas pelo sistema operacional presente, o FreeRTOS.

1.3 Novos conhecimentos adquiridos

A conclusão deste trabalho propõe o aprendizado acadêmico e prático de novas habilidades ao autor que poderão trazer muitos benefícios.

Para começar existe toda a parte do desenvolvimento acadêmico com as etapas necessárias ao desenvolvimento do mesmo, envolvendo pesquisa, disciplina,

desenvolvimento técnico, aquisição da prática na escrita para trabalhos acadêmicos e apresentação pública.

Outro ponto de aquisição de conhecimento está em trabalhar com um Sistema Operacional de Tempo Real – RTOS que até então não tinha sido utilizado pelo autor e é um assunto de muito interesse para o mesmo. Também foi acrescido a experiência em se trabalhar com uma nova família de microcontroladores PIC.

1.4 Corpo do trabalho

Este trabalho está dividido em 5 capítulos, sendo que no capítulo 2, discorre-se sobre RTOS (uma das ferramentas principais), Teorema de Nyquist para processamento de amostragem e como é feita a conversão de sinais analógicos para digitais e vice-versa.

No capítulo 3, é descrito todo o ferramental utilizado, desde os equipamentos a softwares e linguagens utilizadas. Também são discutidos os métodos a serem seguidos com a descrição dos testes planejados. No capítulo 4 serão apresentados os resultados obtidos nos testes.

No capítulo 5 é feita a discussão dos resultados, observações e conclusão do trabalho. Na sequência segue as referências.

Capítulo 2 – Referencial Teórico

2.1 Sistema Operacional de Tempo Real (RTOS)

2.1.1 Definição

A definição básica de um Sistema Operacional (SO) é que este tem por finalidade gerenciar os recursos de *hardware* (processamento, memória, entre outros) [1] e intermediar os usos dos mesmos com a camada de aplicação. Esta camada é onde está situado o código que se refere às funcionalidades. Assim, cada sistema possui tarefas específicas e determinadas para o equipamento em questão. Ainda podem existir, em alguns casos, uma camada superior, que é a do usuário, que corresponde à interação e tomada de ações externas pelo mesmo.

Como exemplos temos o próprio sistema operacional mais difundido entre computadores, o Microsoft Windows® e também o MAC OS®. Nestes exemplos temos claramente as camadas, como ilustrado na Figura 1 (a), onde o equipamento corresponde ao *hardware* do computador, o sistema operacional é o Windows ou MAC OS propriamente dito, a aplicação são os *softwares* usados e a última camada são ações tomadas por usuários em sua interação, como digitação, cliques do *mouse*, etc.

Porém, estes tipos de sistemas são desenvolvidos para atender uma demanda muito genérica, ou seja, são otimizados para executarem uma variedade de aplicações (não previstas pelo desenvolvedor) simultaneamente, garantindo que todas tenham seu direito de uso do processador por um tempo determinado. Deste modo, não são os mais indicados para executarem aplicações que necessitem de um desempenho determinístico, ou que ostentem um período maior sem ocorrência de falhas. Por fim, o sistema pode interromper tarefas com alta prioridade para executar novas tarefas mesmo que de baixa prioridade, tornando impossível assegurar um tempo de resposta constante em aplicações críticas. [1]

O fluxograma da Figura 1 (b) mostra a situação proposta por este trabalho no qual será elaborado um sistema que realiza a interface entre o *hardware* e o código de aplicação de forma autônoma e constante, sem contato com ações externas de usuários.

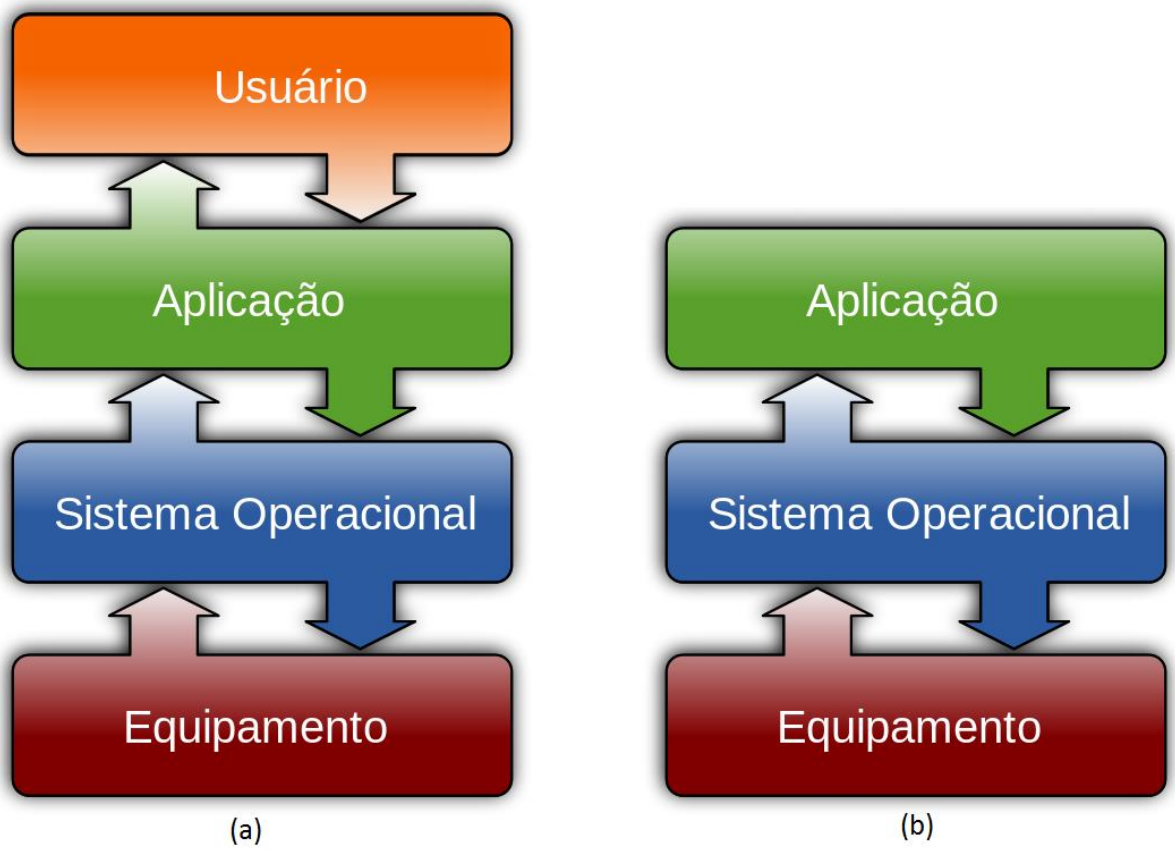


Figura 1 - Camadas de interação de um sistema operacional intermediando o hardware do equipamento até a ação final do usuáio

Fonte: adaptado de [2]

2.1.2 Gerenciamento de tarefas

Um Sistema Operacional de Tempo Real (RTOS) também gerencia múltiplas tarefas, mas é projetado especialmente para rodar aplicações com extrema precisão e alto grau de confiabilidade [1]. A execução do sistema operacional destina-se a múltiplas tarefas onde o tempo é um elemento pré-definido e de alta relevância.

Este tempo de resposta a um evento ou duração da resolução da tarefa é chamado de prazo da tarefa (ou em inglês, *deadline*). A perda deste prazo, ou seja,

quando uma tarefa não é concluída no tempo proposto, indica uma falha no sistema, ora crucial, ora tolerável.

Vale ressaltar um equívoco comum onde, pensa-se que um RTOS irá aumentar a velocidade de execução dos programas. Por mais que aconteça em alguns casos, o foco é a temporização precisa e previsível não importando se a velocidade da resposta é elevada ou não.

São ambientes que não tem por prioridade a performance, mas sim o cumprimento das *deadlines*. Porém, mesmo com a ajuda no gerenciamento das tarefas, não necessariamente há garantia de cumprimento. A eficiência de um Sistema de Tempo Real – STR é analisada pela porcentagem de tarefas cumpridas dentro dos seus respectivos prazos determinados, não pela quantidade de dados que processa.

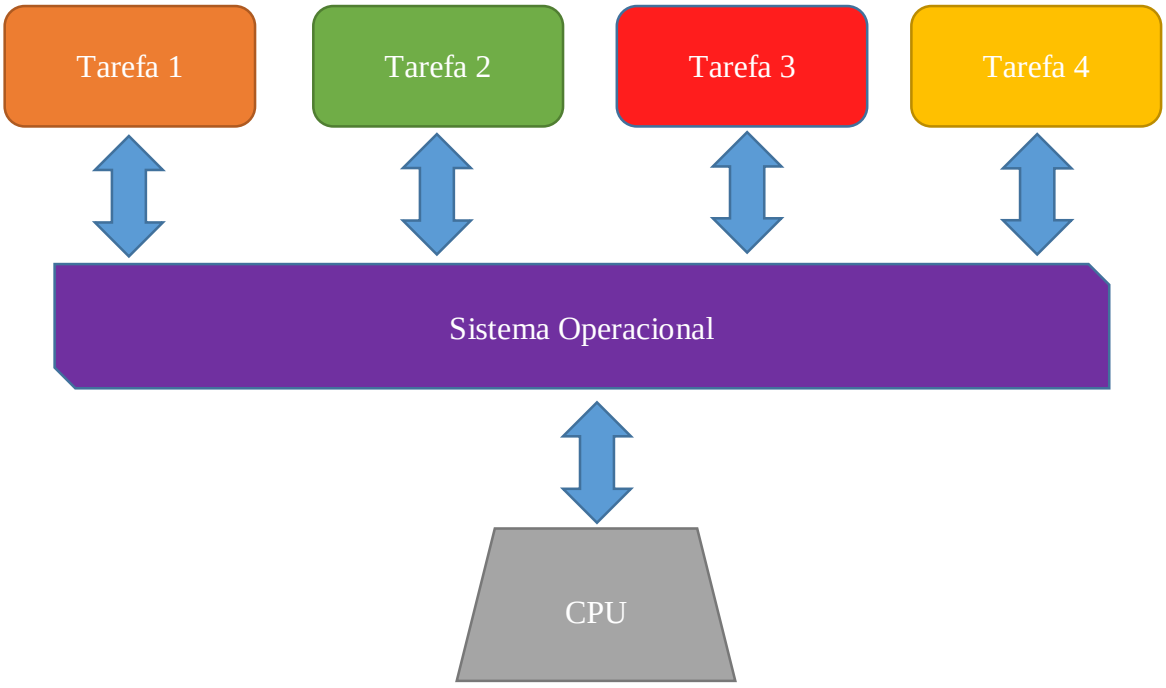


Figura 2 - Função do sistema operacional no gerenciamento das tarefas para uso do processamento

2.1.3 Criticidade

Os STR são classificados, basicamente, em:

- Críticos (*hard* RTS)
- Não-críticos (*soft* RTS)

O STR Crítico é aquele que tem por obrigatoriedade o cumprimento do prazo para execução de uma tarefa (*deadline*). Caso violado este prazo, o resultado obtido

pela tarefa torna-se obsoleto ou até inútil. “**Todas** as tarefas são provadas de executar no prazo” [3]. Como exemplo temos um freio ABS que necessita ser ultraconfiável, caso contrário poderá acarretar em acidentes.

O STR Não-Crítico também tem a atenção focada nos prazos como algo fundamental, porém uma falha é aceitável. O não cumprimento do prazo para finalizar uma tarefa não causa danos irreversíveis, apenas um atraso. Nestes casos, o resultado de uma tarefa não torna-se inútil, mas pode perder sua validade ao longo do tempo. “A **maioria** das tarefas é executada no prazo” [3]. Como exemplo temos um leitor de DVD onde um atraso representa uma demora na aquisição dos dados, mas não necessariamente uma falha crítica.

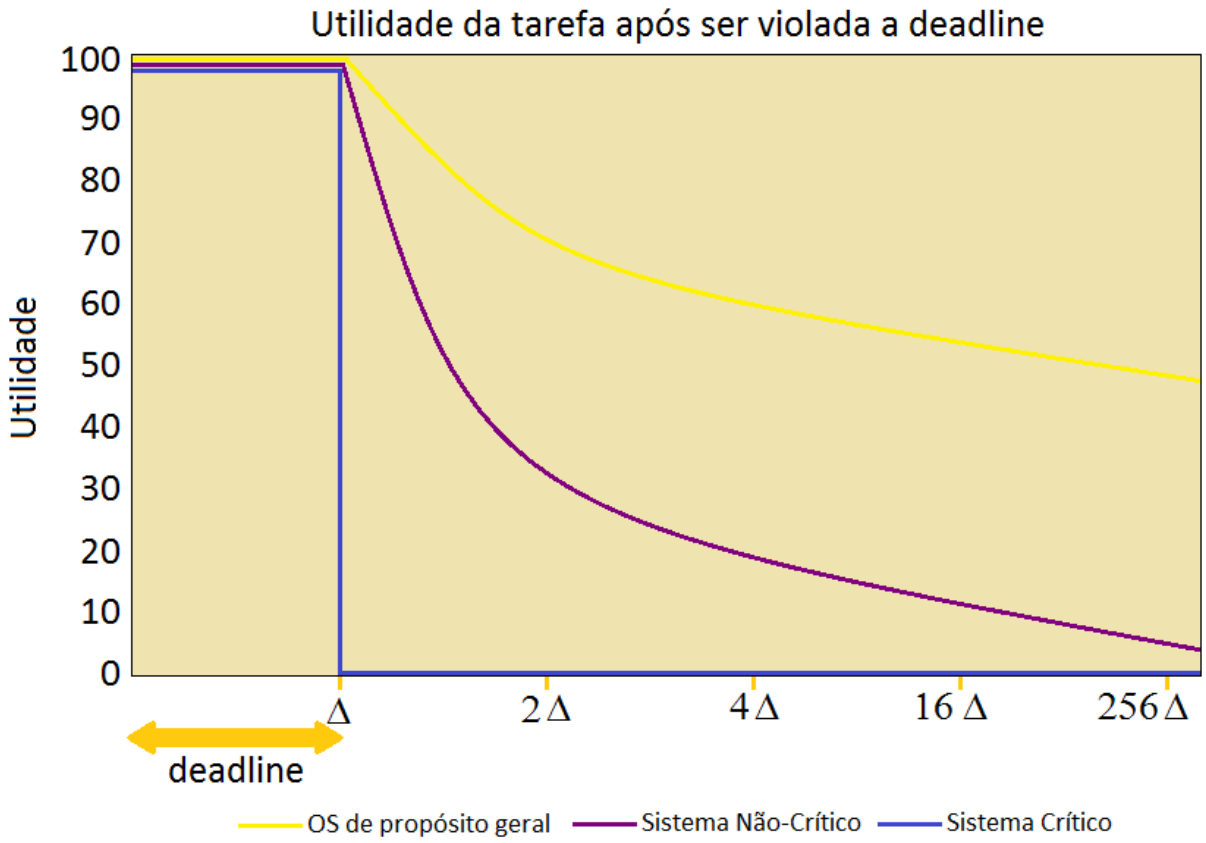


Figura 3 - Diferença do uso do resultado de uma tarefa após violar o tempo máximo para execução para sistemas de propósito geral, Crítico e Não-Crítico

Fonte: traduzido e adaptado de [1]

2.1.4 Escalonador

“O elemento chave que diferencia um RTOS de um OS convencional é o escalonador.” [3]. Existem tipicamente dois tipos:

- Baseado na divisão do tempo, onde as tarefas se alternam de acordo com pulsos de *clock* do processador.
- Baseado em eventos (escalonamento por prioridades), quando as tarefas são alternadas somente quando a necessidade de execução de uma prioridade mais alta sobrepõe a mais baixa. Tal fato denomina-se preemptividade.

Também presente nos demais sistemas operacionais, os RTOS possuem uma fila para ser inseridas as tarefas que se encontram prontas para serem executadas, denominada de fila de prontos. Nota-se que não é executada qualquer tarefa indiscriminadamente, existem requisitos para tal e meios de informar sua prontidão quanto ao atendimento de requisitos.

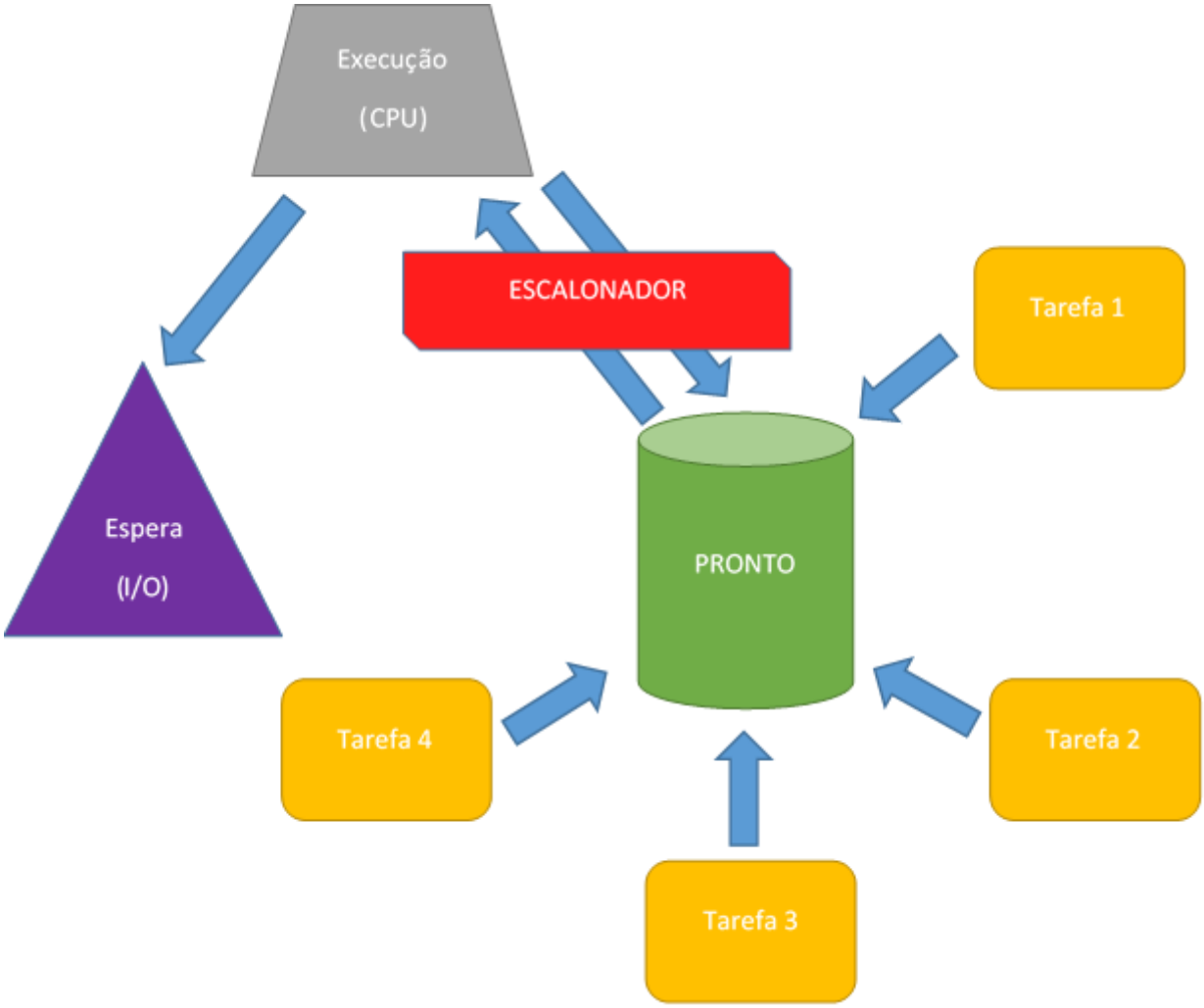


Figura 4 – Atuação do escalonador no gerenciamento da fila de pronto

Fonte: adaptado de [4]

Os algoritmos de escalonamento podem ser classificados como estáticos ou dinâmicos. Estáticos tem como conceito as bases de divisão de tempo. Seus requisitos temporais tem que serem conhecidos previamente. O mais utilizado é o escalonamento por taxas monotônicas (*rate monotonic scheduling* – RMS). Este método é preemptivo e atribui prioridades fixas utilizando-se do número de vezes em um determinado tempo que uma tarefa é acionada, ou seja, quanto maior a frequência de execução, maior a prioridade. Com isto, gera-se um ciclo periódico de processos.

Já os algoritmos dinâmicos não possuem prioridades fixas para as tarefas, alternando conforme a necessidade. O critério para decisão é o tempo de execução. O algoritmo mais comum é o "prazo mais curto primeiro" (*Earliest Deadline First* – EDF) que não necessita de periodicidade, ou seja, a cada ciclo podem haver número de tarefas prontas diferentes.

A maior vantagem do EDF vem do fato de deixar a CPU sempre ocupada, mas em contrapartida torna o desenvolvimento do código mais trabalhoso e complexo. Para aumentar a taxa de sucesso do cumprimento das tarefas, é necessária a otimização dos escalonadores. É comum buscar a utilização direta dos recursos de *hardware* (*clock*, interrupções, etc) assim como dividir as tarefas em ações menores para facilitar a priorização.

2.1.5 Partilha de recursos

Existem recursos do sistema que podem ser utilizados por várias tarefas. Por exemplo, a leitura de um único sensor de temperatura interna de um forno que produz cerâmicas especiais pode ser usado por tarefas como determinar a intensidade do aquecedor ou a velocidade da esteira de produção, entre outras. Ainda mais crítico, aliado a isto, uma tarefa de alarme com alta prioridade que garante a segurança em caso de superaquecimento.

"A partilha de recursos é delicada de se usar num ambiente multitarefas, (…) Quando uma tarefa está a utilizar um dado recurso, a ler, ou a escrever, convém bloquear as outras tarefas de utilizarem esse mesmo recurso" [5]. Existem alguns métodos para serem utilizados a fim de obter uma partilha eficiente dos recursos,

como: Mascaramento temporário (desabilitar interrupções), Semáforos Binários e Mensagens Trocadas.

2.1.5.1 Mascaramento temporário (desabilitar interrupções)

Este método consiste basicamente em desabilitar, temporariamente, os serviços de atendimento às interrupções por parte do processador. Assim, uma tarefa ignora algum estado crítico do sistema até o fim de sua execução, obtendo o monopólio temporário de um determinado recurso.

Este método faz com que as interrupções demorem mais para serem atendidas, o que pode acrescer de um risco de falhas para sistemas que se baseiam no atendimento imediato.

2.1.5.2 Semáforos binários

Semáforos tem por função indicar o bloqueio e desbloqueio de algum determinado recurso. Quando uma tarefa necessita de um recurso que tem chance de ser compartilhado, esta pode bloqueá-lo. Todas as tarefas que se utilizam deste recurso devem primeiro verificar o estado de bloqueio para o mesmo, caso esteja bloqueado, a tarefa pode não se utilizar deste recurso e executar outras funções (caso não seja determinante), não ser executada (o uso do recurso é a função principal da tarefa) ou apenas esperar que o recurso seja desbloqueado.

Alguns problemas podem surgir decorrente do método relacionado a semáforos tais como a prioridade invertida e o entrave.

A prioridade invertida ocorre quando uma tarefa de baixa prioridade é acionada e bloqueia o uso do recurso. Em seguida, uma tarefa de alta prioridade é acionada, porém é obrigada a esperar até que o recurso seja desbloqueado pela tarefa de baixa prioridade. Uma alternativa é atribuir uma prioridade elevada, temporariamente, para a tarefa que está bloqueando o recurso. Isto faz com que a mesma seja resolvida mais

rapidamente, forçando então, o desimpedimento do recurso da forma mais rápida possível.

O outro problema citado, o entrave, acontece quando há uma cadeia de semáforos em tarefas que se utilizam de mais de um recurso compartilhado, criando uma espera infinita pelos desbloqueios dos recursos. “Se uma tarefa A bloquear o semáforo F1 e depois esperar pelo desbloqueio do semáforo F2, e uma tarefa B estiver bloqueada no semáforo F1 para desbloquear o semáforo F2, cria-se um entrave” [5].

2.1.5.3 Mensagens trocadas

Um recurso tem sua gestão feita apenas por uma tarefa capaz de responder a perguntas sobre o estado atual de sua utilização. Quando outras tarefas necessitam do mesmo recurso, são feitas perguntas sobre seu estado atual. As respostas podem ser binárias do tipo ocupado/desocupado ou mais elaboradas com estados intermediários prevendo um possível tempo para liberação das mesmas.

Mensagens também podem ser enviadas solicitando o desbloqueio do recurso, para o caso de uma tarefa de maior prioridade requisitar seu uso. Utilizando de mensagens com indicações intermediárias dos processos, é possível criar um sistema que, através de cálculos estatísticos, preveja se o tempo restante necessário para a liberação do recurso é suficiente para não atrapalhar a *deadline* do processo de maior prioridade, fazendo com que ambas as tarefas sejam atendidas satisfatoriamente.

Através deste método são evitados os problemas de não atendimento de tarefas como no método de mascaramento temporário e também os problemas de entrave que podem surgir com o uso de semáforos, tornando o sistema mais estável.

2.1.6 FreeRTOS

FreeRTOS é o RTOS líder de mercado atualmente e foi desenvolvido pela *Real Time Engineers Ltd*. É um sistema totalmente gratuito que pode ser usado em

produtos comerciais sem a necessidade de expor os códigos fonte proprietários desenvolvidos. Só tem por exigência a presença clara da referência nas documentações, assim segue o site para obtenção do mesmo: FreeRTOS.org.

> A principal característica é que o FreeRTOS é desenvolvido para ser pequeno o suficiente para rodar em microcontroladores. (…) Estes possuem recursos limitados em um processador que incorpora, em um único chip, o processador em si, memória de somente leitura (ROM – Read-Only Memory ou FLASH) para armazenar os programas que serão executados e memória de acesso aleatório (RAM – Random Access Memory) necessária para execução dos programas. [6 – Tradução direta]

Microcontroladores geralmente são usados em aplicações embarcadas de baixo nível onde o usuário nunca de fato enxerga o processamento ou *softwares* sendo executados. São aplicações normalmente muito específicas.

Aliado a isto e também devido a sua limitação de capacidade, raramente justifica-se a implementação de um sistema de tempo real completo (*full* RTOS) inclusive na maioria dos casos sendo até impossível.

O FreeRTOS fornece o escalonamento de tempo real, comunicação entre as tarefas, temporização e sincronização. Funções adicionais podem ser incluídas através de complementos (*add-on*). O escalonador usado alcança o determinismo permitindo que o usuário atribua a prioridade para cada tarefa.

O código fonte em C do sistema, possui um alto padrão de qualidade, sempre com uma supervisão estrita de seus desenvolvedores mantendo o padrão. Todos os códigos são extremamente testados antes de serem inclusos nos pacotes oficiais para download.

É um sistema preparado para ser multi-plataforma, facilitando sua migração para famílias diferentes de microcontroladores. Assim, diminui os custos com novos desenvolvimentos ou esforço demasiado gasto para migração e mantém o código fonte útil por mais tempo.

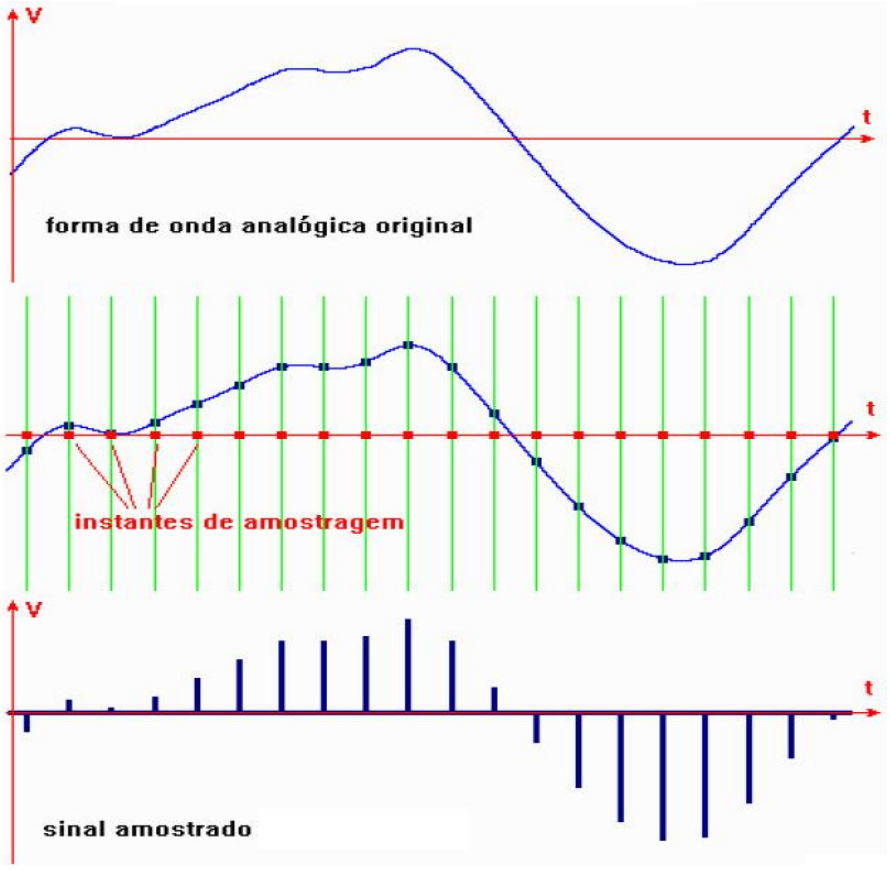
A *Real Time Engineers Ltd*. fornece um suporte gratuito, muita documentação com tutoriais e treinamentos, assim como disponibiliza códigos de exemplos para várias plataformas facilitando muito o início do projeto.

2.2 Teorema de Nyquist

O Teorema de Nyquist é fundamental para a área que atua com teoria de informação, especialmente em processamento de sinais, como por exemplo, codificadores de sinais analógicos e aplicações em telecomunicações. Tal importância se dá por ser um teorema que estabelece o critério adequado para a amostragem dos sinais.

Amostrar significa transformar algum tipo de sinal (contínuo no tempo) em uma sequência numérica (discreto no tempo). O processo de amostragem é realizado medindo-se o valor do sinal contínuo em intervalos fixos de tempo T, que é chamado de intervalo de amostragem. O resultado é uma sequência de números (amostras) que representam o sinal original.

Através do processo de amostragem, obtêm-se na saída um sinal com amplitude igual ao valor instantâneo do sinal de origem, chamados pulsos PAM (pulsos modulados em amplitude).



Fonte: [8]

Figura 5 – Conceito básico sobre o Princípio da Amostragm mostrando a formação do sinal discreto gerado a partir de um análogico

O teorema envolve duas etapas de processamento de sinais. Primeiro tem-se a amostragem para converter o sinal contínuo em discreto. Em seguida o processo de reconstrução do sinal original a partir da recuperação das informações discretas adquiridas na primeira etapa.

> "Seja um sinal, limitado em banda, e seu intervalo de tempo dividido em partes iguais, de forma que se obtenham intervalos tais que, cada subdivisão compreenda um intervalo com período T segundos, onde T é menor do que 1/2*fm, e se uma amostra instantânea é tomada arbitrariamente de cada subintervalo, então o conhecimento da amplitude instantânea de cada amostra somado ao conhecimento dos instantes em que é tomada a amostra de cada subintervalo contém toda a informação do sinal original." [7 – Tradução direta]

De acordo com o Teorema de Nyquist, a quantidade de amostras no tempo (frequência de amostragem) deve ser **maior ou igual** ao dobro da maior frequência contida no sinal a ser amostrado, para que possa ser reproduzido integralmente.

Como não é possível garantir que o sinal não contenha sinais acima da frequência estipulada pelo teorema, é necessário aplicar uma filtragem no sinal com algum filtro do tipo passa-baixa ajustado para a frequência de corte igual (ou menor) que a Frequência de Nyquist.

Todo meio de comunicação possui uma banda específica que por muitas vezes é bem limitada, o que força a transmissão de amostras de sinais ser restrita. Quanto maior for a frequência de amostragem, mais fácil será reproduzir o sinal, porém pode haver um desperdício de banda ocupada sem nenhuma melhoria na qualidade.

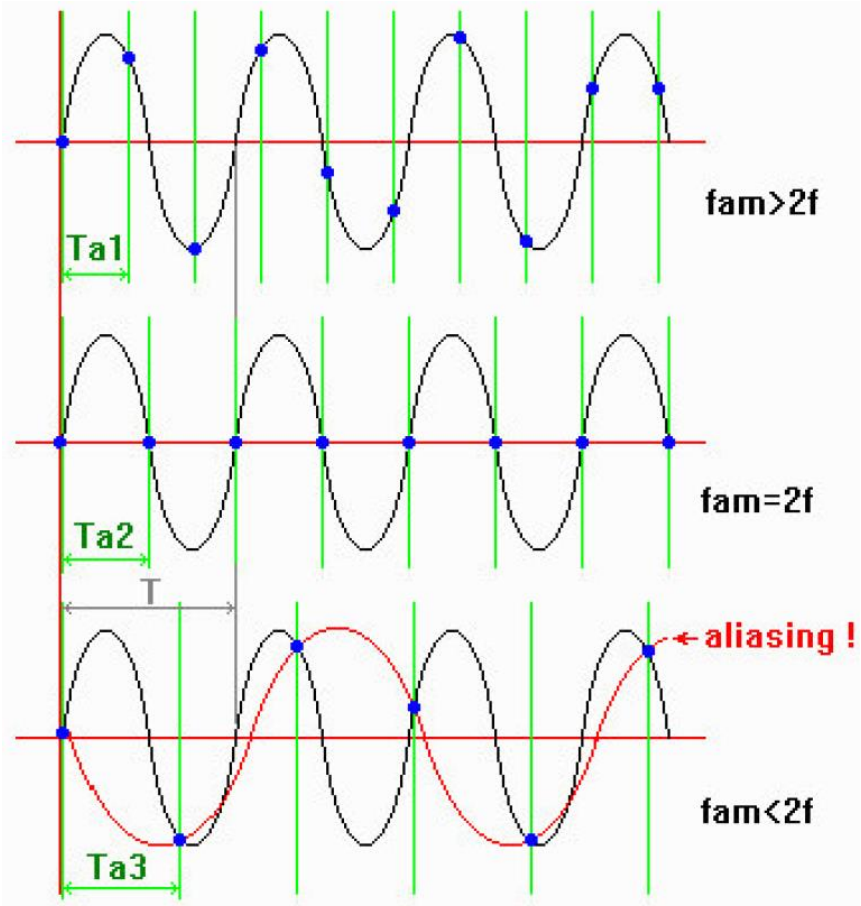


Figura 6 – Demonstração de efeitos para diferentes frequências de amostragem

Fonte: [8]

Na Figura 6 são mostrados os efeitos da amostragem quando se aplica taxas próximas ao limite previsto pelo teorema. Na parte superior a frequência de amostragem (fam) é maior que duas vezes a do frequência do sinal original (f). É possível notar que uma reconstrução ocorre sem erros pois possui amostras suficientes.

No gráfico intermediário a fam é igual a duas vezes a frequência do sinal, tal fato não garante que possa ser realizada a reprodução pois, como no exemplo, existem casos que o sinal PAM vale zero. Esta importante exceção marca a contribuição de Shannon ao estudo que por muitos consideram como Teorema de Nyquist-Shannon. Assim, para Shannon a frequência de amostragem deve ser apenas **maior** que o dobro da maior frequência contida no sinal a ser amostrado

Já na parte inferior, a fam é menor que o dobro de f. A reconstrução do sinal é errônea devido ao insuficiente número de amostras, como mostrado na curva em vermelho. Este erro é causado pelo fenômeno conhecido por *aliasing*.

2.3 Conversores A/D e D/A

Conversor analógico/digital ou ADC (do inglês *analog-to-digital converter*) é um dispositivo eletrônico que, a partir de uma grandeza analógica, consegue gerar um sinal digital que o represente.

Em sentido reverso temos os conversores digital/analógico que recompõem um sinal analógico a partir de um sinal digital discreto em amplitude com 2^N níveis. Esse sinal gerado a partir do sinal discreto é considerado analógico quando tem-se um valor de N muito grande, ou seja, amostras suficientes para não descaracterizar o sinal.

As principais características dos conversores que precisam ser levadas em conta são:

- Tempo de conversão: duração do tempo necessário entre a chegada de um sinal na entrada e seu respectivo valor convertido na saída.
- Taxa de conversão: valor que indica quantas vezes por segundo um sinal é convertido.
- Resolução N: menor variação do sinal de entrada que causa um (de)incremento unitário no valor da saída. Quanto maior for a quantidade de níveis discretos de um sinal, melhor será a reprodução

2.3.1 Conversores A/D

São muito úteis na interface entre dispositivo digitais (microprocessadores, microcontroladores, DSPs, etc) e dispositivos analágicos [9]. Fundamental a importância em um projeto de interfaces para aquisição de dados, como este trabalho se propõe.

Uma grande vantagem do tratamento de sinais na forma digital é a maior imunidade ao ruído, o que também deixa o armazenamento de dados com uma maior facilidade.

O primeiro passo para se realizar a conversão de um sinal analógico para um digital é a amostragem e retenção realizadas por circuitos SH (*sample and hold*). Estes circuitos garantem que a amostra não se altere até o fim da conversão. Vale lembrar que pelo critério de Nyquist, a frequência de amostragem deve ser igual ou maior do que o dobro da banda de frequência do sinal amostrado.

Podem ser classificados em dois grandes grupos que são à Frequência de Nyquist e sobre-amostrados. Os mais utilizados se enquadram no primeiro grupo do qual serão explanados os principais tipos que são o paralelo, aproximações sucessivas, contador e integrador de rampa simples e dupla.

2.3.1.1 Paralelo

Consiste em comparar, simultaneamente, a tensão de entrada analógica com as tensões fixas de referências. O número de comparadores e referências necessárias são (2^N -1), ou seja, para um conversor paralelo de 3 bits, são necessários sete comparadores e tensões de referência.

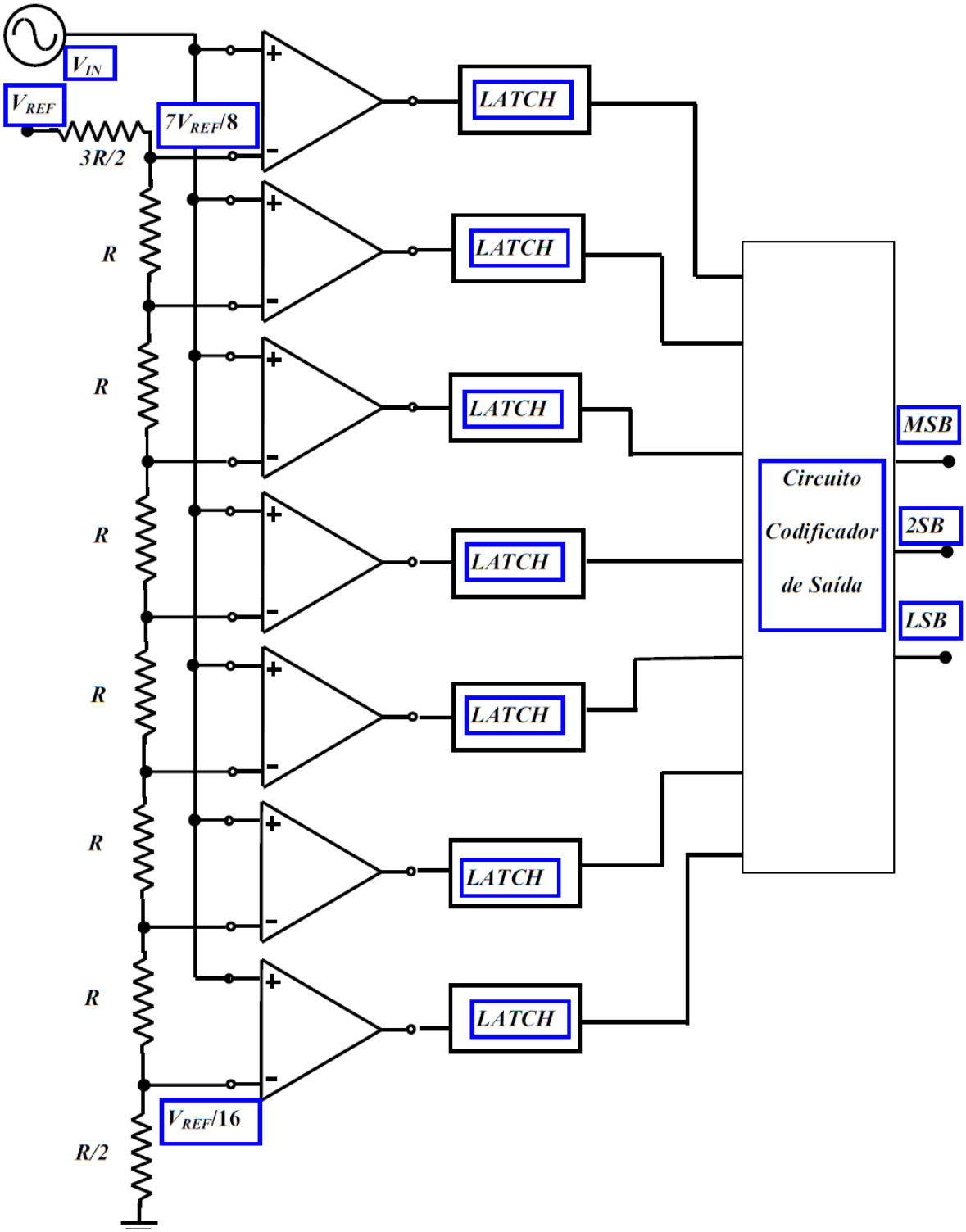


Figura 7 - Conversor A/D tipo Paralelo ("Flash")

Fonte: [10]

Cada comparador usa como entrada a tensão que irá ser convertida e uma tensão de referência única que é obtida a partir de uma malha de resistores. O objetivo

desta malha é determinar vários intervalos de tensões a fim de encontrar em quais o sinal está inserido.

Com um determinado valor obtido do sinal a ser convertido, todos os comparadores que tiverem uma tensão de referência menor que a do sinal de entrada terão como saída um nível baixo, os demais nível alto. Assim temos na saída dos comparadores o chamado código termômetro.

Podemos então utilizar um codificador na saída para obter o código binário correspondente como mostra a Tabela 1.

Tabela 1 - Tabela código termômetro X binário

Nível	Código Termômetro	Código Binário
0	0000000	000
1	0000001	001
2	0000011	010
3	0000111	011
4	0001111	100
5	0011111	101
6	0111111	110
7	1111111	111

Fonte: [10]

A maior vantagem do conversor A/D paralelo é sua rapidez de conversão. O fato de todas as comparações serem feitas simultaneamente o torna o mais rápido de todos os tipos conversores. Para fazer jus a vantagem, normalmente tenta-se sempre o manter atualizado quanto a versões mais novas e rápidas de uma determinada tecnologia.

São comumente utilizados no processamento de sinais de alta frequência, como sinais de vídeo. É possível ter a conversão efetuada em apenas um ciclo de *clock*, porém é habitual efetuar em 2 ciclos, onde em um ciclo é amostrado o sinal, comparado e retido e no segundo ciclo é feita a operação de codificação. De qualquer forma, o conversor é extremamente rápido.

A desvantagem deste conversor paralelo é o aumento do número de comparadores e a complexidade do codificador da saída conforme for necessária uma

resolução maior. O grande número de componentes aumenta o custo, a área de silício e o consumo de potência.

2.3.1.2 Aproximação sucessiva

Este tipo de conversor A/D é muito utilizado comercialmente, pois é o que mais se aproxima da velocidade do tipo paralelo, porém sem suas desvantagens em relação ao aumento de componentes citado anteriormente. Utiliza-se de uma técnica de realimentação para relacionar a entrada analógica a um código digital de saída.

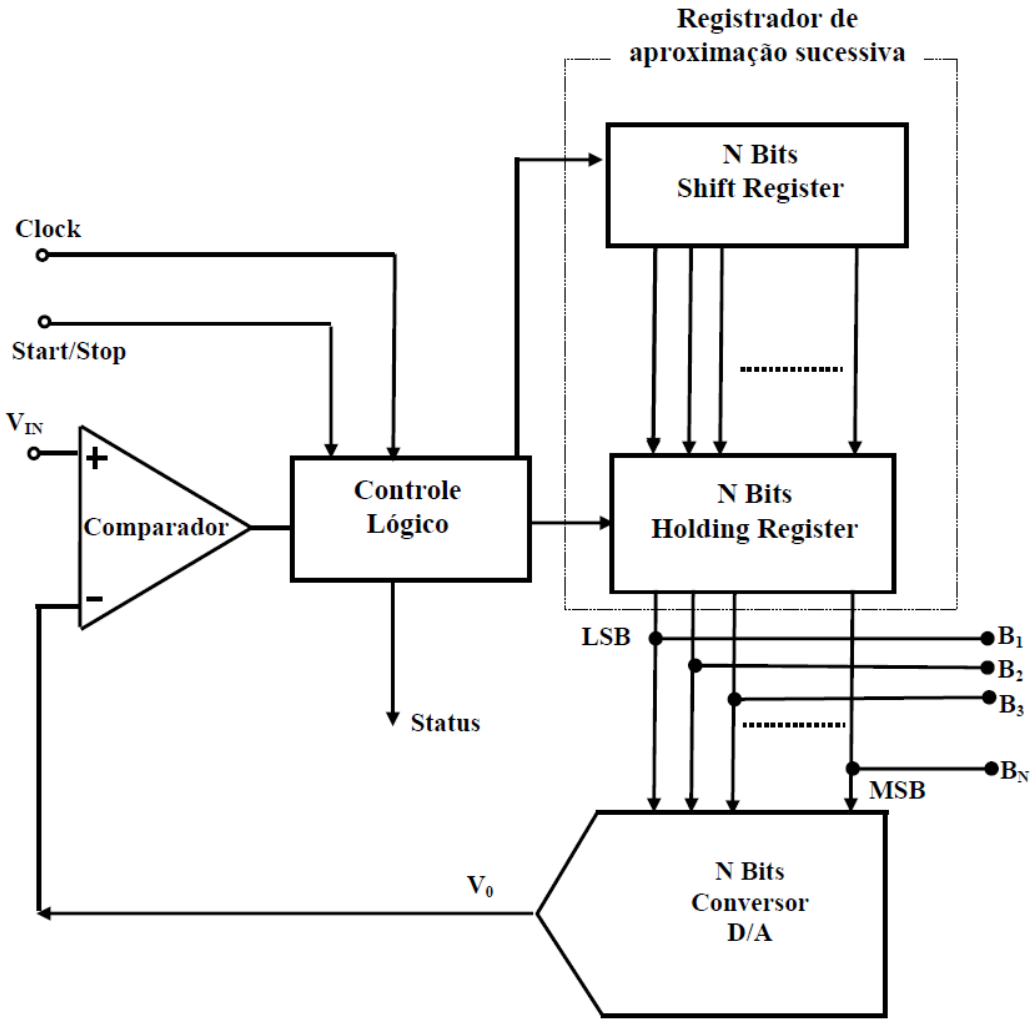


Figura 8 - Conversor A/D tipo Aproximação Sucessiva

Fonte: [10]

No início do processo de conversão o *Shift Register* e o *Holding Register* são zerados. Atribui-se nível lógico 1 apenas no bit mais significativo do *Holding Register* e o restante em nível lógico 0. Realiza-se a comparação com o sinal de entrada. Se a saída ainda for menor que a entrada, este *bit* é mantido em nível lógico 1, caso contrário muda-se para nível lógico 0. O mesmo processo é feito agora para o segundo *bit* mais significativo. O processo continua até que todos os *bits* tenham sido verificados, de acordo com a resolução N desejada.

Apesar de não ser tão rápido quanto a conversão quase instantânea do tipo parapelo, o conversor por aproximação sucessiva também é bem versátil onde o tempo de conversão necessário é fixo e de (N+2) ciclos de *clock* e permite maiores resoluções sem o aumento expressivo dos componentes.

2.3.1.3 Contador

Este tipo de contador é bastante simples, de baixo custo e fácil implementação. Porém exige um número maior de ciclos de *clock* para se chegar ao fim de cada conversão.

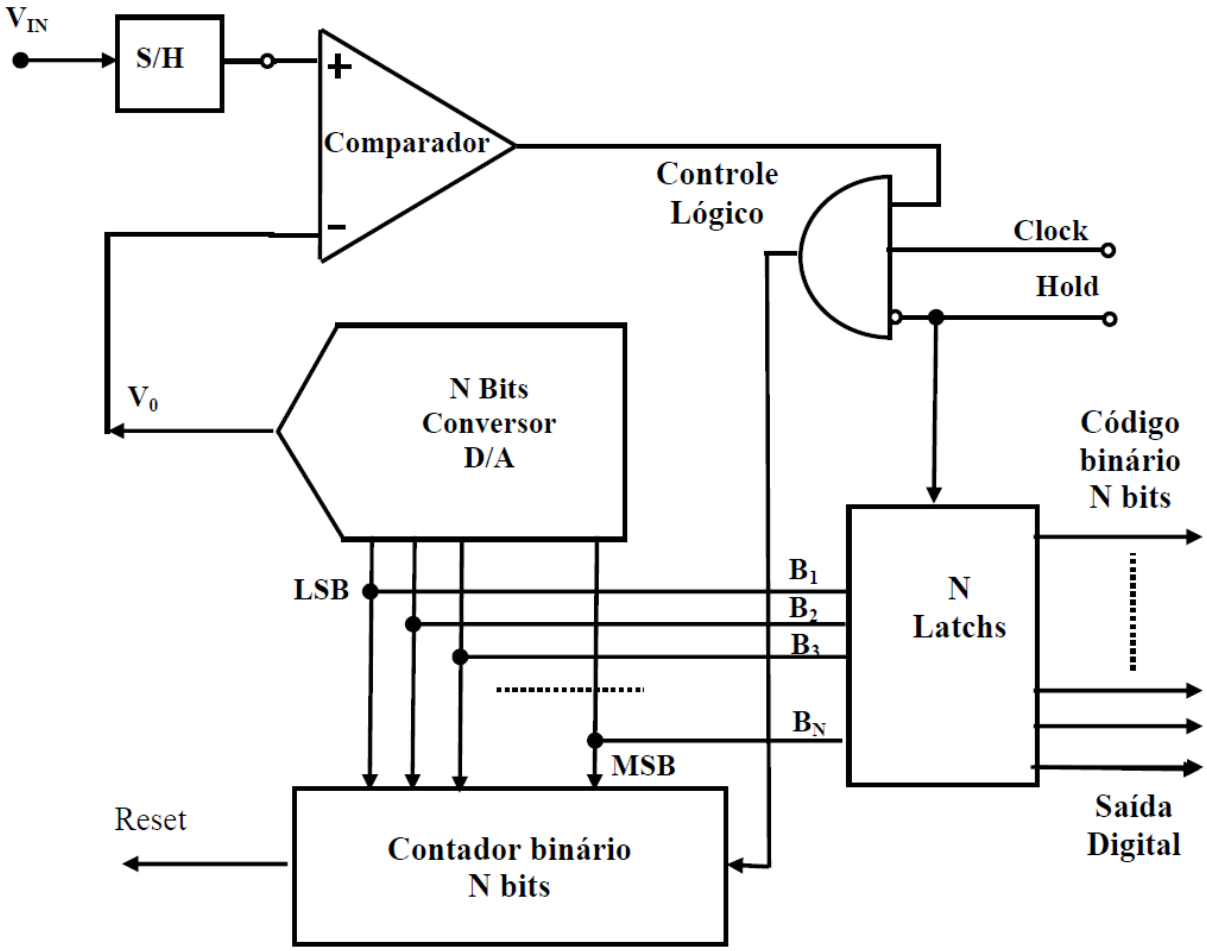


Figura 9 - Conversor A/D tipo contador

Fonte: [10]

A cada conversão, o valor do sinal de entrada armazenado no S/H é comparado com um valor que é incrementado unitariamente até que atinja o mesmo valor do sinal de entrada. O tempo é variável e pode ser demorado caso o valor da entrada esteja na outra extremidade da contagem do incrementador.

Como citado anteriormente, apesar de sua simplicidade, a sua desvatangem encontra-se em ter que varrer um valor incremental a cada nova conversão e ter este tempo variável. Mas uma boa vantagem é a possibilidade de uma alta resolução, assim é indicado quando se precisa relacionar boa resolução com uma taxa de conversão moderada.

2.3.1.4 Integrador

Para este tipo de conversor existem várias formas de implementação do circuito lógico, porém o conceito básico é sempre integrar a entrada e obter como saída o número de ciclos executados pelo processo, onde através de um contador binário consegue a saída digital.

Não é o tipo mais rápido de conversor devido ao tempo necessário para completar a rampa (explicado posteriormente) mas permite uma alta resolução a um baixo custo, com uma característica importante que os diferenciam dos demais tipos de conversores, a boa rejeição a interferências ou ruído.

O funcionamento é relativamente simples, com o contador e integrador zerados, a saída do comparador fica em nível lógico 0, habilitando os pulsos de *clock*. Liberando o pino de reset, o integrador começa a produzir a rampa linear em função de RC, também começa a ser incrementado o contador até que o valor da rampa se iguale a tensão de referência. Neste ponto o contador para de contar e tem-se no *latch* a saída digital correspondente ao número de pulsos acumulados no contador.

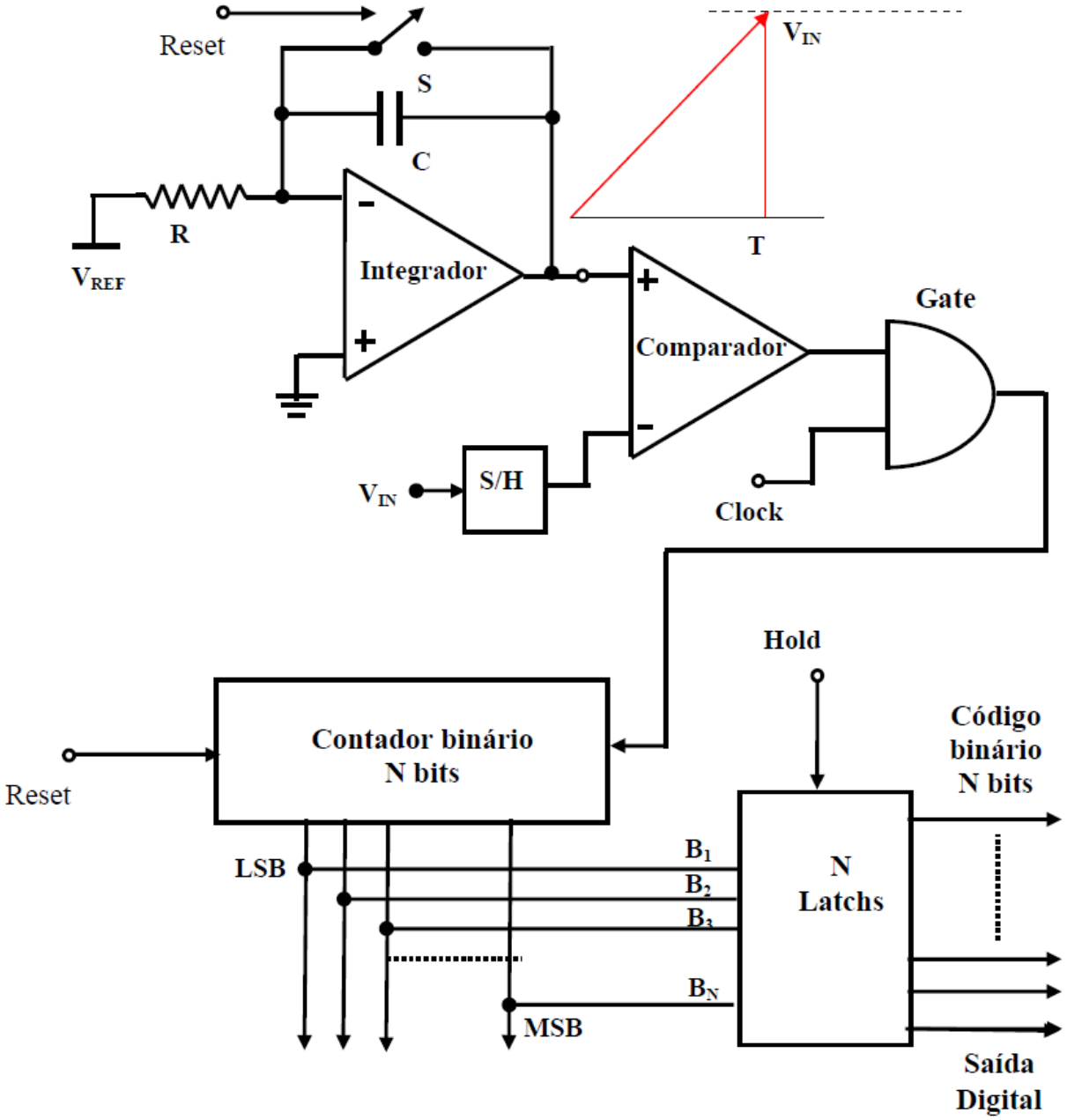


Figura 10 - Conversor A/D tipo integrador com rampa simples

Fonte: [10]

Este formato de rampa simples tem por desvantagem a dependência da constante de tempo RC e o offset do comparador. Para evitar estas desvantagens é utilizada uma aplicação ligeiramente diferente com um integrador de rampa dupla.

O processo de conversão agora possui duas fases. Na primeira, idêntica ao método de rampa simples, o integrador fornece um sinal crescente linear em função de RC. Mas agora, ao atingir o mesmo valor do sinal de referência, o sinal de entrada do integrador passa a ser a referência, não mais o sinal a ser convertido. Assim, o

integrador agora fornece um sinal, também linear, mas desta vez decrescente até este sinal atingir o valor igual a zero.

Com o método da rampa dupla, o tempo medido pelo contador não mais depende da constante de tempo RC do integrador, assim como também não sofre nenhum erro devido a algum *offset* presente.

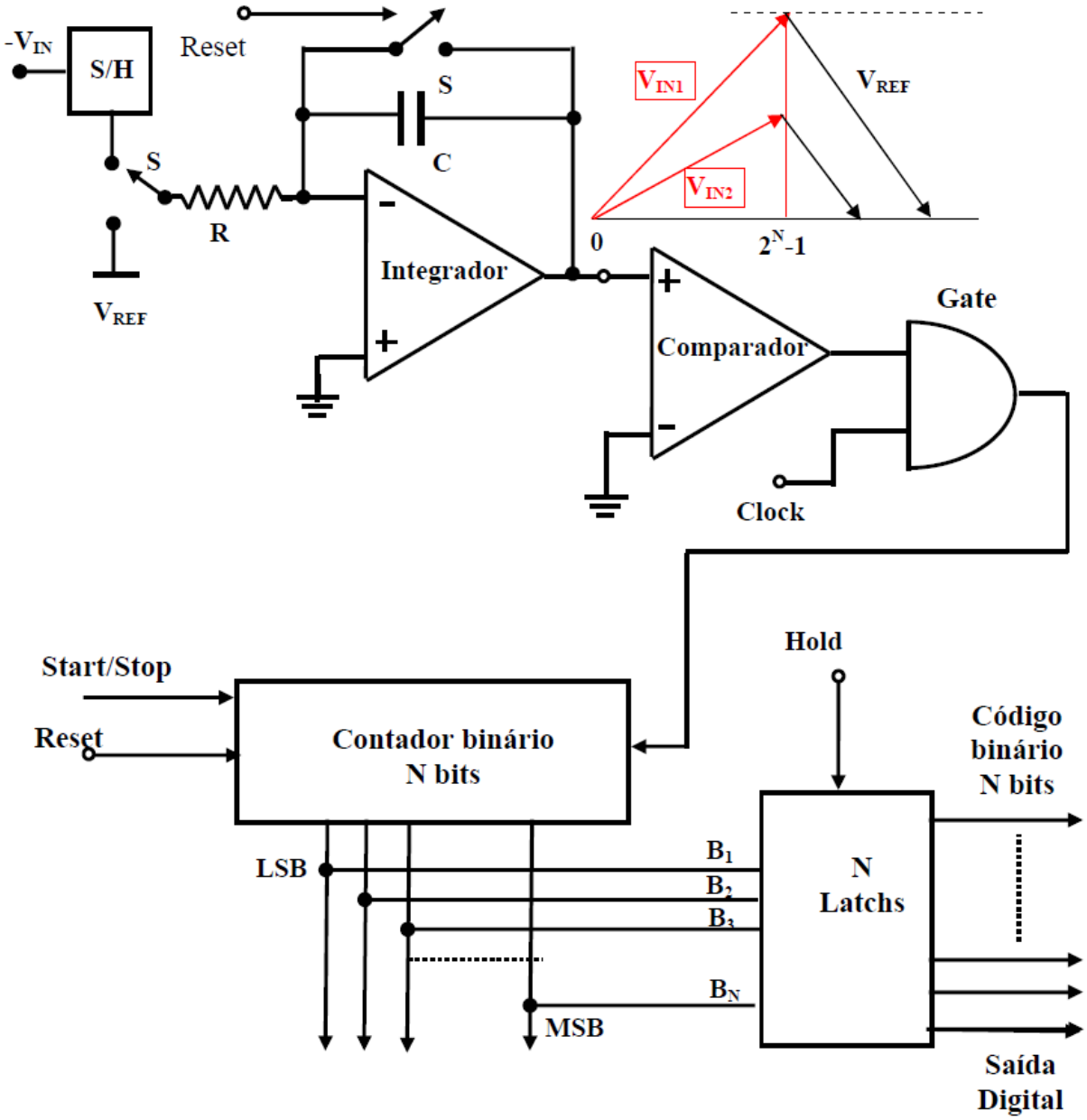


Figura 11 - Conversor A/D tipo integrador com rampa dupla

Fonte:[10]

2.3.2 Conversores D/A

Um sinal sendo originalmente digital ou quando provém de uma conversão prévia, depois de digitalizado, é processado e, na maioria das vezes, será utilizado para atuar sobre o circuito analógico que gerou o sinal original, ou até mesmo sobre outro circuito. Para tal, é necessário que o sinal seja previamente convertido (ou reconvertido) para a forma analógica. Quanto mais bits conter o sinal de entrada digital, ou seja, quanto maior a resolução N, melhor será o resultado do sinal analógico devido a maior precisão.

Como exemplo, temos um tocador de CD que converte as informações digitais contida no disco para a forma analógica gerando o som nos falantes. Ou um sistema de controle realimentado, onde recebe uma informação analógica de um sensor, converte para digital para o processamento e, por fim, é reconvertido em analógico para atuar no mesmo sistema. Há três tipos comuns de conversores D/A: resistor ponderado, rede R-2R e PWM.

2.3.2.1 Resistor ponderado

Cada bit do sinal digital é ligado a uma respectiva chave associada a um resistor ponderado. A soma de todas as chaves acionadas compõe uma tensão proporcional ao seu valor. A ponderação do valor dos resistores equivale a significância de cada bit.

A desvantagem deste conversor é a dificuldade em encontrar os valores ideais dos resistores, necessitando por várias vezes realizar associações dos resistores para alcançar os valores desejados. Isto faz com que não seja muito usado.

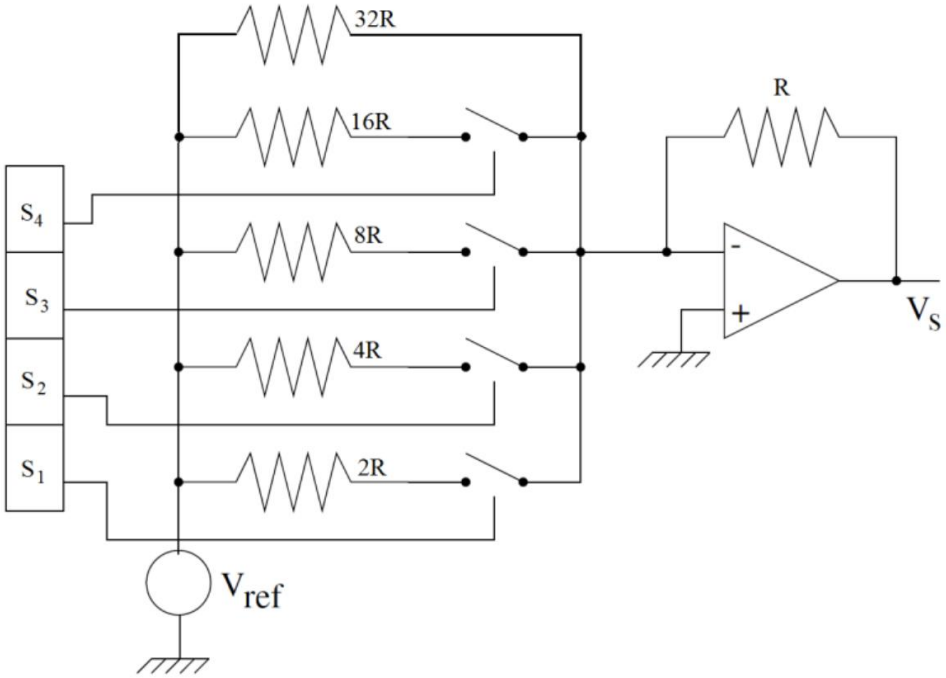


Figura 12 - Conversor D/A de 4 bits a resistores ponderados

Fonte: [11]

2.3.2.2 Rede R-2R

Diferente dos resistores ponderados, este tipo utiliza apenas dois valores de resistores, ou até apenas um usando de uma associação se for possível. Tal fato torna este conversor bem versátil. A lógica de operação é semelhante ao tipo resistor ponderado.

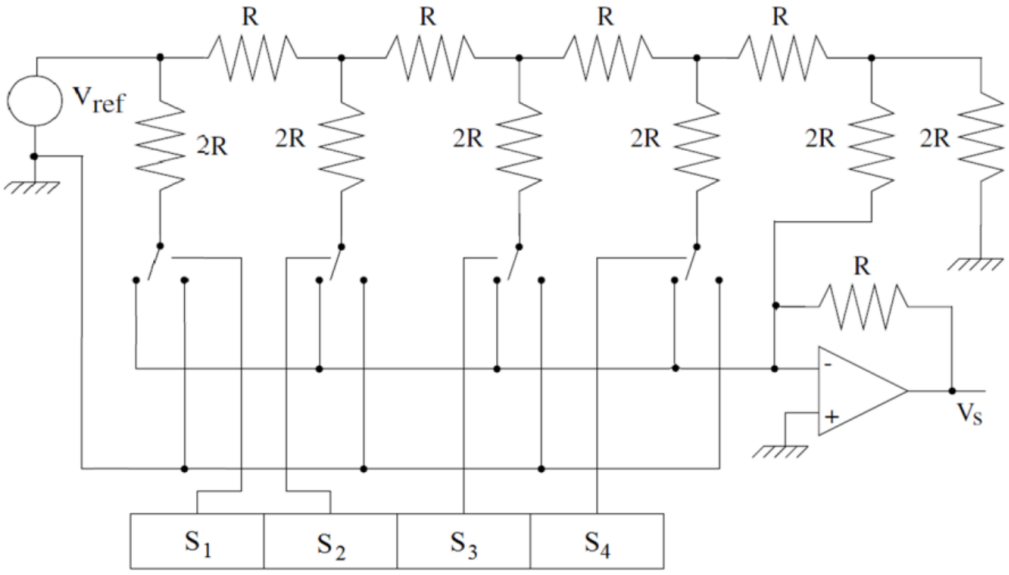


Figura 13 - Conversor D/A de 4 bits com rede R-2R

Fonte: [11]

2.3.2.3 PWM

São bastante usados em microcontroladores de fácil implementação por código alterando-se portas com pouca complexidade. Basicamente, gera-se um sinal com período constante T, amplitude constante E e duração de pulso programável digitalmente τ.

O valor de tensão médio é correspondente ao valor digital programado em τ. A desvantagem é o tempo de resposta pois necessita sempre do período completo para se obter um valor médio.

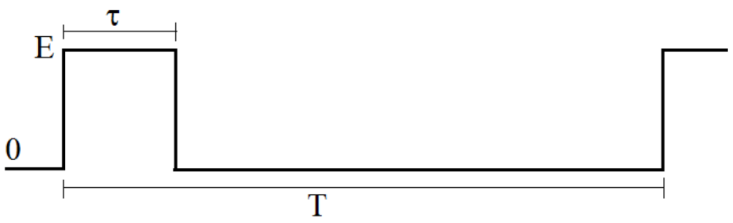


Figura 14 - Sinal PWM

Fonte: [11]

Capítulo 3 - Materiais e métodos

3.1 Ferramentas

A fim de economizar tempo de projeto, custo de produção e esforço para aprovar o pleno funcionamento do *hardware*, evitou-se desenvolver um projeto de específico para o trabalho. Além de contar com possíveis mudanças e/ou melhorias ao longo do trabalho que poderiam acarretar em novos tempos de produção de placas.

Visto isto, foi escolhido o uso de um kit de desenvolvimento microcontrolado a fim de tornar o trabalho bem versátil e facilitar algumas aplicações pois disponibiliza de forma bem acessível a maioria dos recursos do microcontrolador, como por exemplo LED's para facilitar a sinalização durante o desenvolvimento, botões, saída serial ou leitura de um canal de conversão análogico-digital (AD).

Assim, há opções prontas e consolidadas de mercado que, durante o processo de aprendizado e desenvolvimento, tornam mais fáceis executar as primeiras configurações e testes de algo que será útil para aplicação final.

O kit foi escolhido por ser bem completo, possuir boa documentação e haver alguns membros do laboratório com experiência na família de microcontroladores que poderiam auxiliar o autor quando necessário.

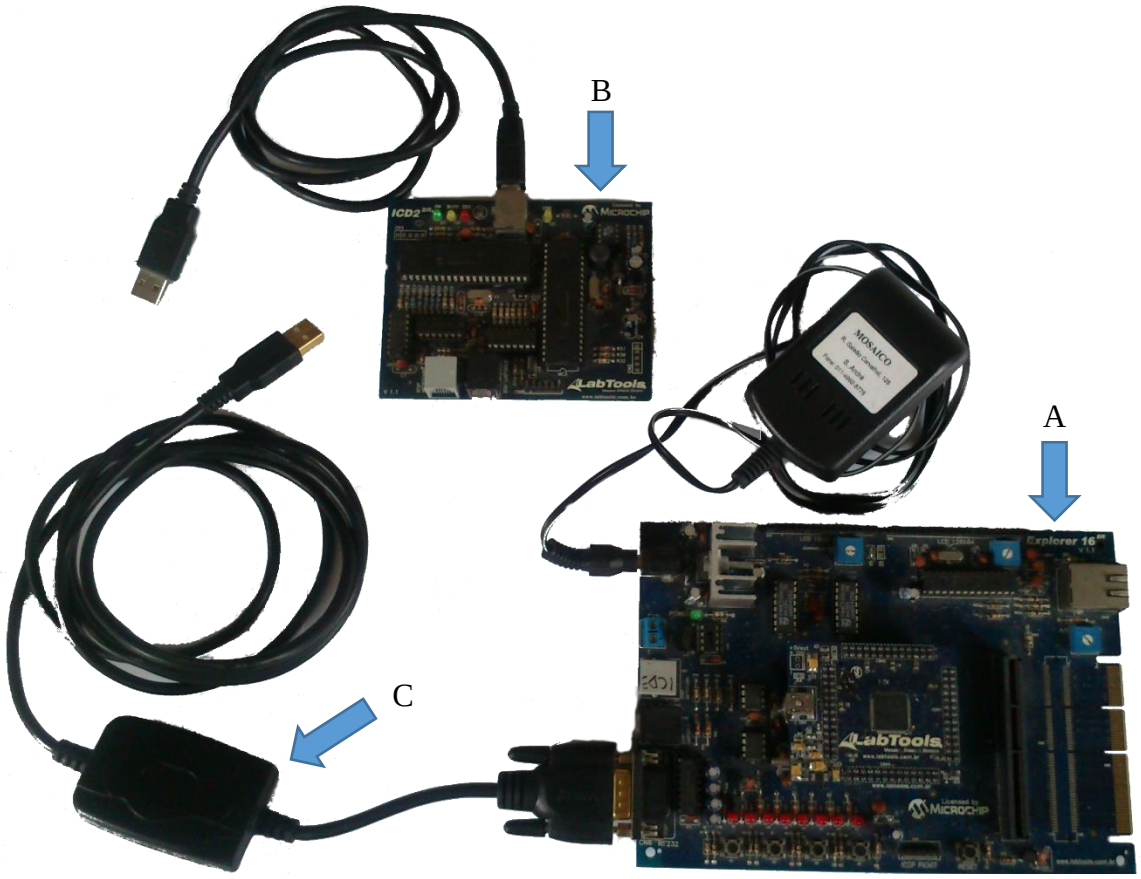


Figura 15 - Kit escolhido: Explorer16^{BR} (A) e o programador ICD2^{BR} (B), ambos da empresa Mosaico®. Conversor USB-RS232 da GigaWare® (C)

OBS: A impressão LabTools® nas placas representam a divisão de produtos didáticos da empresa Mosaico®, que atualmente foi novamente integrada junto com Hiware® (divisão tecnológica) e operadas somente por Mosaico®. Todas as placas são ferramentas licenciadas pela Microchip®, garantindo assim a compatibilidade com qualquer outro Plugin ou periférico produzido pela mesma.

3.1.1 Explorer16^{BR} V1.1

O kit é composto de uma placa-mãe que recebe o nome de Explorer16^{BR} V1.1, versão nacional da empresa Mosaico® para a placa Explorer16 original produzida pela empresa Microchip®. Também faz parte do kit a placa-filha, denominada de “plugin”, onde está de fato o microcontrolador.

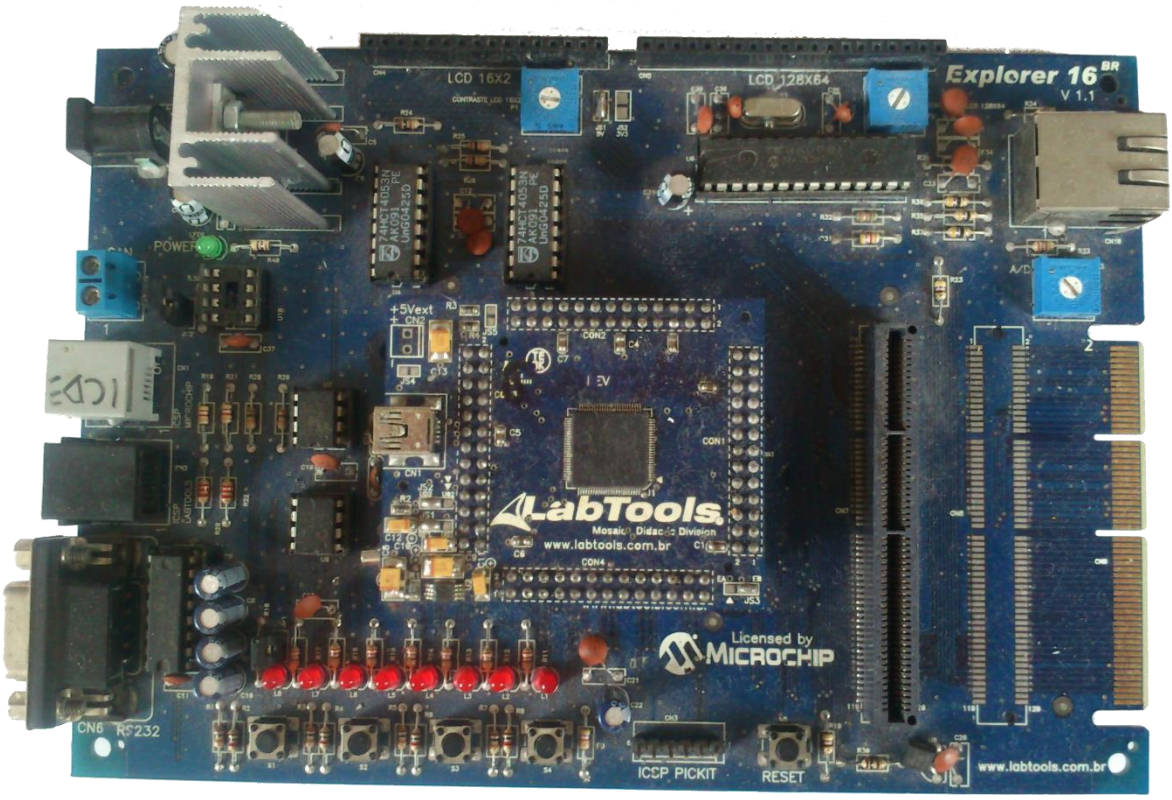


Figura 16 - Explorer16^{BR} V1.1

Existem alguns modelos de plug-ins disponíveis, o escolhido foi o “Plugin Explorer16BR PIC32MX460F512L-80I/PT USB” da Mosaico®. Este placa conta com um microcontrolador de 32 bits PIC, modelo PIC32MX460F512L. Ideal para resolução de problemas complexos como execução de RTOS [12], que é o escopo deste trabalho.

Figura 17 - Plugin Explorer16BR PIC32MX460F512L-80I/PT USB

Fonte: [12]

O microcontrolador presente nesta CPU é o PIC32MX460F512L que pode trabalhar até 80Mhz/105GMIPS. Possui 512K de memória Flash e 32K de RAM. Uma característica importante são os rápidos e precisos conversores analógico-digital com 16 canais de 10-bit.

3.1.2 ICD2^{BR} V1.1

O programador que compõe o kit utilizado é o ICD2^{BR} V1.1 da Mosaico® que segue o mesmo padrão das placas anteriores que é uma nova versão nacional gerada baseada na original ICD2 da Microchip®. Comunica-se com o computador através de uma conexão USB.

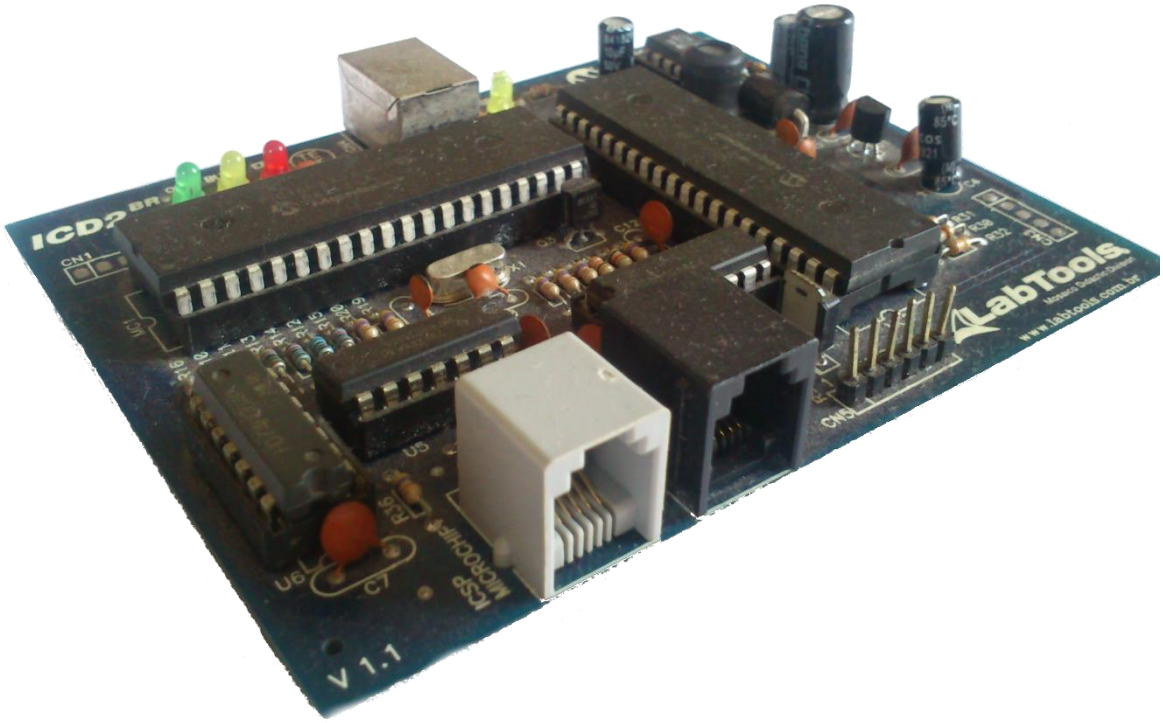


Figura 18 - Programador ICD2^{BR} V1.1

3.1.3 PICTail^{PROTO} V1.0

Esta é uma placa expansora que dá acessos a quase todas as portas do microcontrolador liberando seu uso independente do sistema didático ligado a mesma presente na placa-mãe. Assim, por mais que alguma porta esteja ligado a um LED na placa-mãe, por exemplo, é possível desativá-lo e ter livre acesso para qualquer outro fim que queira usando esta PICTail.

O modelo utilizado foi o PICTail ^{PROTO} V1.0 também da Mosaico® licenciada pela Microchip®. Junto à pinagem, que é disponibilizada para acesso às portas, há também, uma área de pontos de encaixe e solda semelhantes a uma protoboard, facilitando pequenas montagens de circuitos que possam ser necessárias, evitando, assim, elaboração de placas externas ou outras protoboards interligadas.

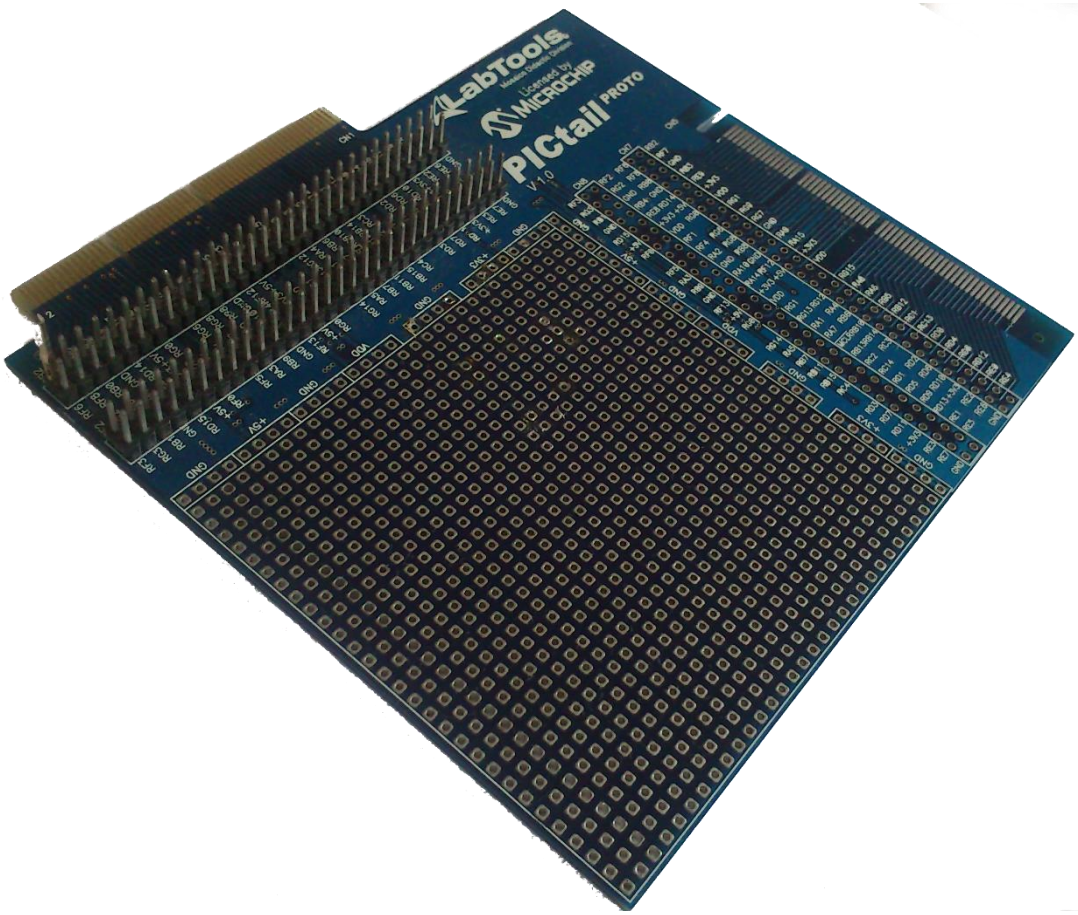


Figura 19 - PICTail^{PROTO} V1.0

3.1.4 Conversor USB-RS232

Esta é uma ferramenta útil para usos em que , por exemplo, o computador não possui uma porta serial com conector DB9. O kit Explorer16^{BR} conta com uma porta serial RS232 que é usada no projeto para comunicar com o computador que irá montar os sinais. Para tal, utiliza-se este conversor para acessar a porta USB do computador.

O modelo usado foi do fabricante GigaWare que possui chipset Prolific PL2303. É importante citar que houve uma certa dificuldade em encontrar os drivers corretos e fazer o dispositivo funcionar com o Windows 8 da Microsoft®, sistema operacional utilizado pelo autor.

Figura 20 - Conversor USB-RS232 GIGAWARE

3.1.5 MPLAB

Software indicado e desenvolvido pela Microchip® para programação e compilação de microcontroladores PIC. Inicialmente foi utilizado a versão MPLAB IDE v8.92 por ser compatível com os dispositivos utilizados. Assim, neste software é possível compilar o projeto e já programar o dispositivo ou depurar o código diretamente pelo mesmo.

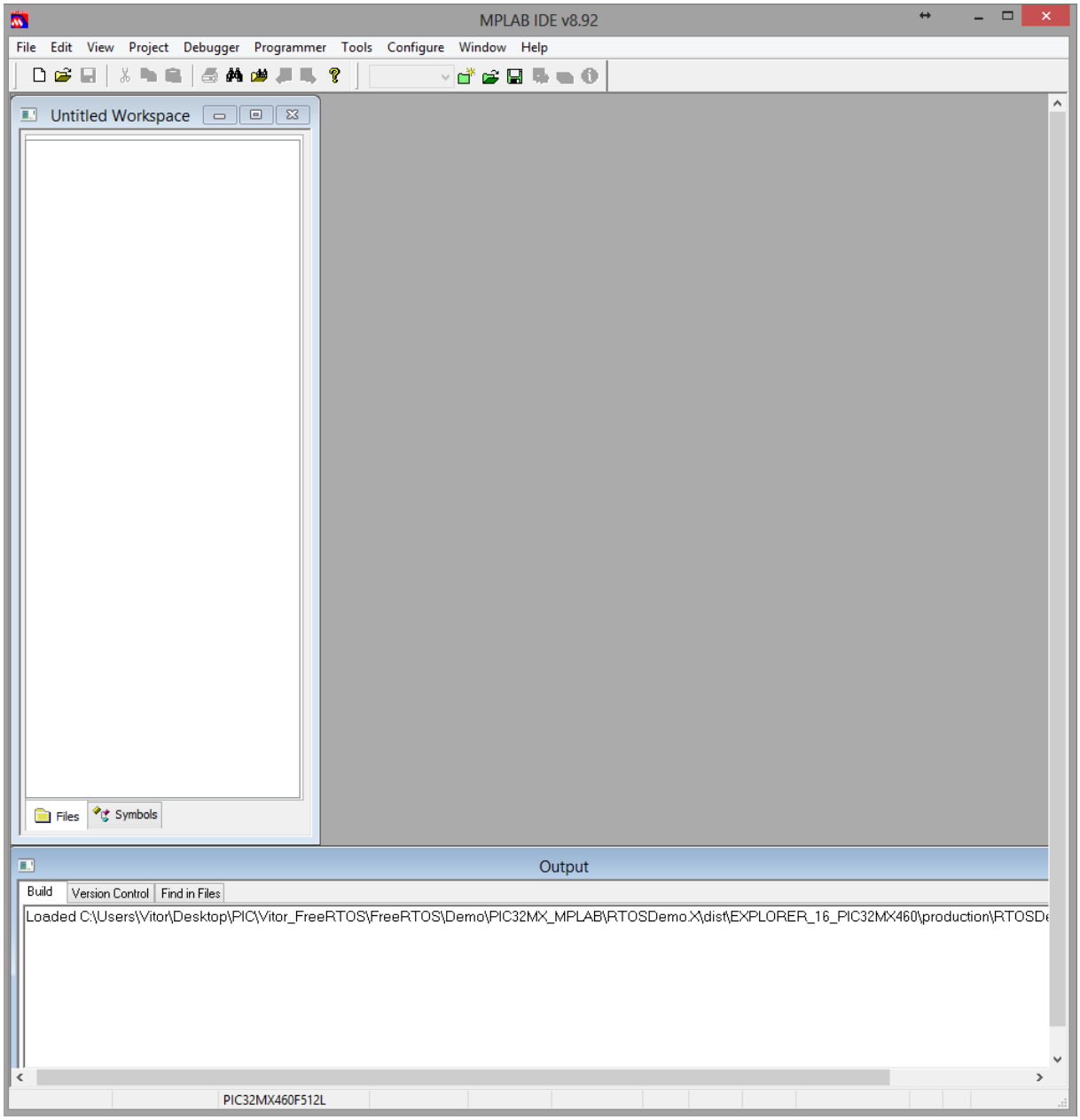


Figura 21 - MPLAB IDE v8.92 Utilizado apenas para programar o dispositivo

Posteriormente, foi necessário a migração para a versão MPLAB X IDE v1.90, por ser recente e possuir vários novos recursos que auxiliam no desenvolvimento, como uma interface remodelada, recursos de interação com o desenvolvedor e possibilidade de inserção de plug-ins e novos motores de compilação. A necessidade surgiu ao tentar rodar códigos de demonstrações do sistema operacional que exigiam esta nova versão.

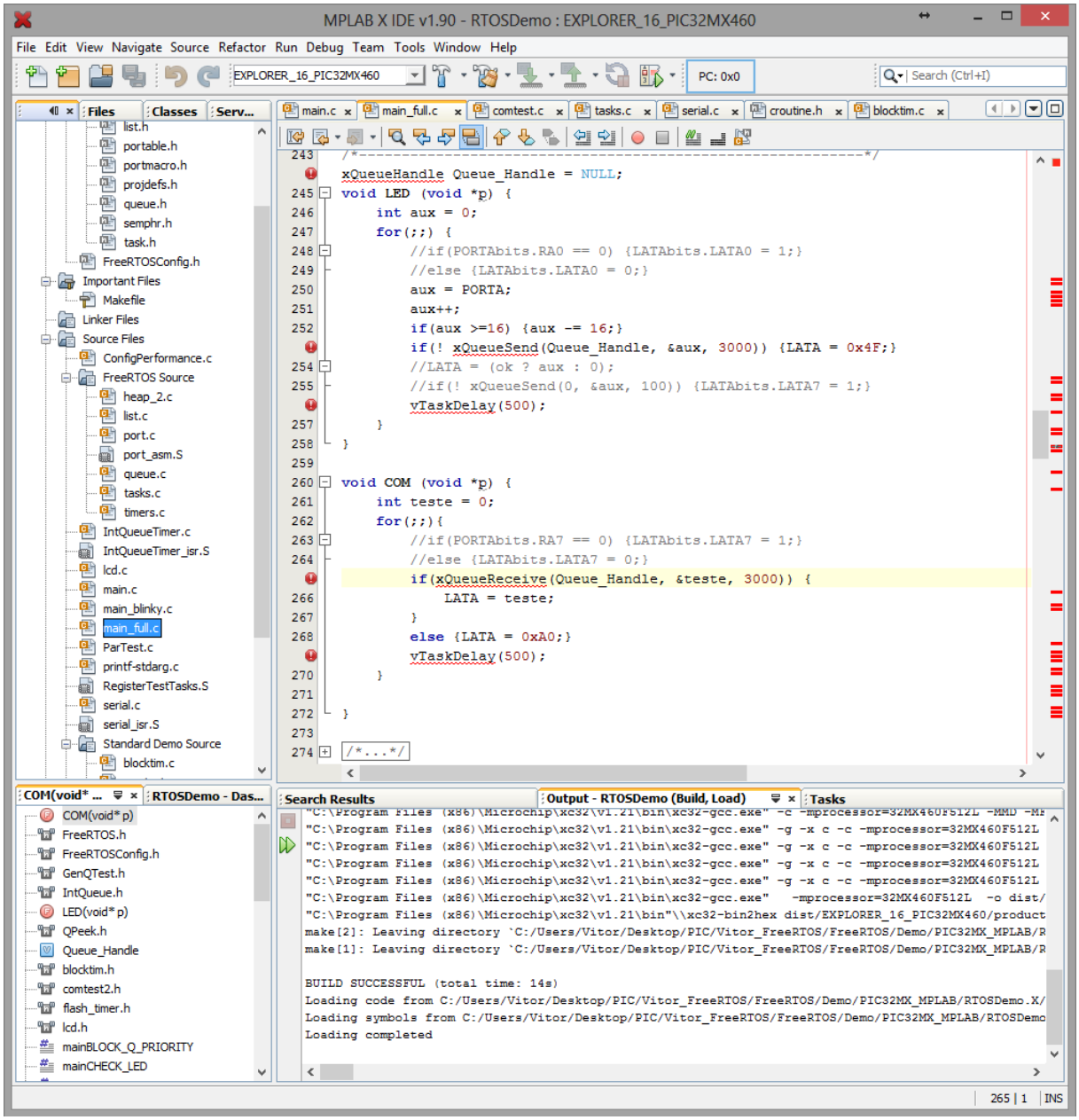


Figura 22 - MPLAB X IDE v1.90 Utilizado para desenvolver e compilar o código

Porém, essa versão não é compatível com o programador ICD2[BR] utilizado. Por consequência, foi preciso gerar um arquivo de saída no formato hexadecimal pelo compilador MPLAB X IDE 1.90 (mais novo). Após compilado, acessar o software MPLAB IDE v8.92 (mais antigo), importar este arquivo hexadecmal e gravar no dispositivo.

3.1.6 RealTerm

É um software que atua como um terminal para monitoramento de comunicações seriais. É possível escolher quais das portas disponíveis serão monitoradas e mostrar as informações de diversas formas como binário, hexadecimal ou ASCII, por exemplo.

Também é possível enviar dados para o dispositivo conectado e criar logs. Este software foi bastante utilizado para testar toda a comunicação serial do projeto. A versão utilizada foi: RealTerm Serial Capture Program v2.0.0.70.

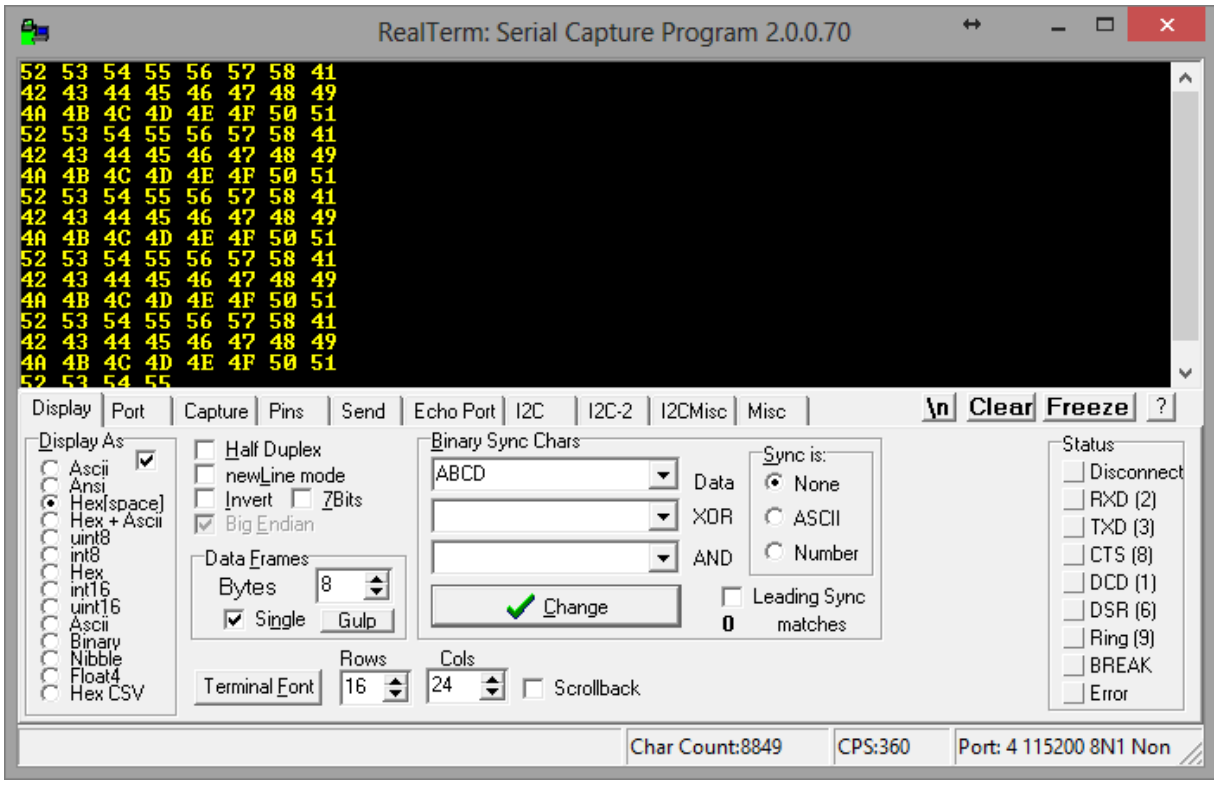


Figura 23 - RealTerm: Serial Capture Program 2.0.0.70

3.1.7 LabVIEW

LabVIEW (do inglês *Laboratory Virtual Instrument Engineering Workbench*) é uma plataforma de projeto gráfico e desenvolvimento para sistemas de medição, teste e controle, produzido pela empresa National Instruments. A programação é feita seguindo o modelo de fluxo de dados.

Os “programas” em LabVIEW são denominados de instrumentos virtuais, ou VI’s (do inglês *Virtual Instruments*). Seu ambiente de desenvolvimento inclui o painel frontal, que contém a interface visível de operação do usuário, e o diagrama de blocos, onde é elaborado o código gráfico do programa por meios de blocos de funções e ligações. A linguagem gráfica do LabVIEW é conhecida como “G”.

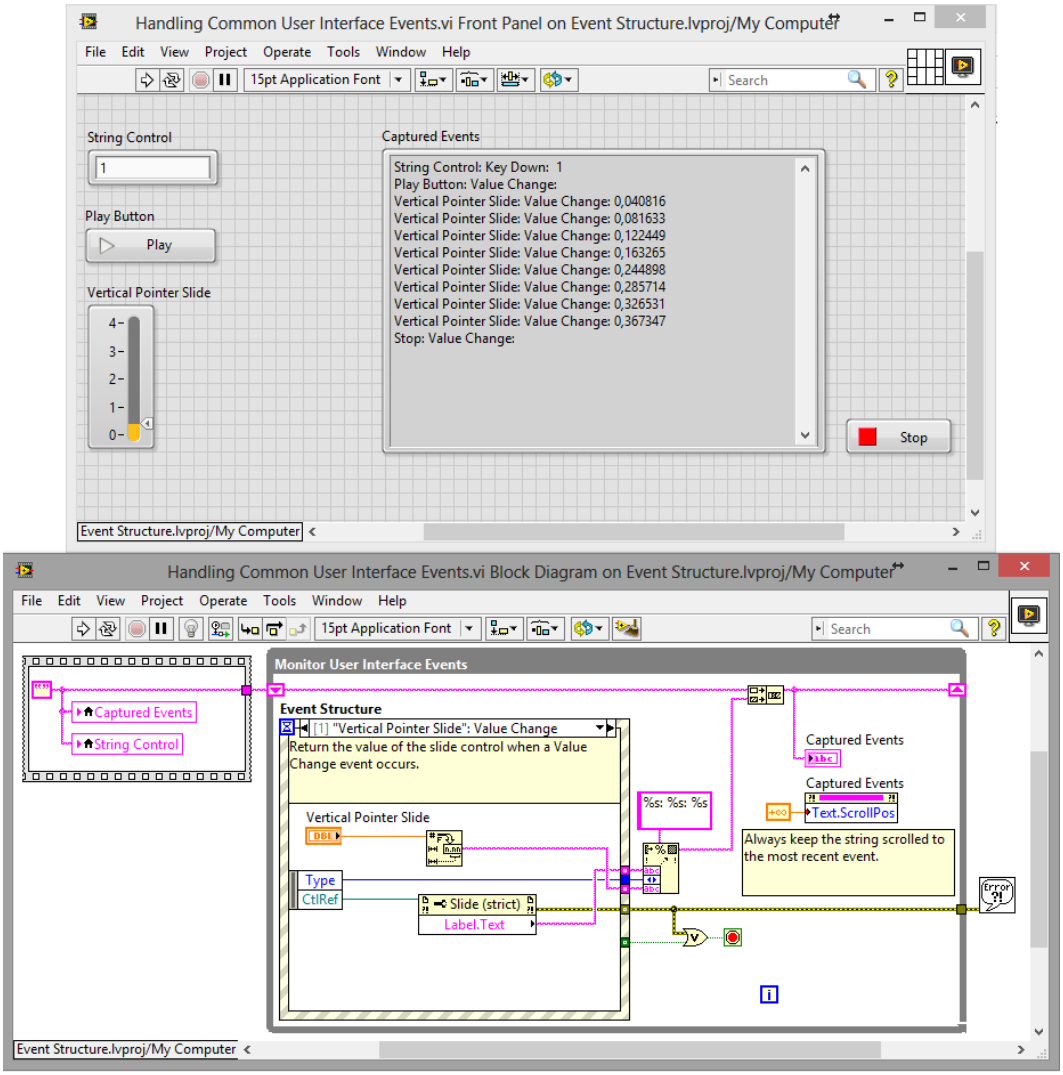


Figura 24 - Exemplo de telas do LabVIEW

3.1.8 Linguagem C

A linguagem C é extremamente difundida quando se diz respeito a programação de microcontroladores. Isto se dá por ser uma linguagem de propósito geral que facilita interpretação e portabilidade dos códigos para diferentes plataformas, bastando apenas usar o compilador para cada caso, que será responsável por gerar o código adequado para cada dispositivo.

Esta linguagem surgiu no centro de Pesquisas da Bell Laboratories criada por Dennis Ritchie em 1972 com o fim de reescrever o Sistema Operacional UNIX, que na época foi elaborado em Assembly. [13]

3.2 Casos de testes

Os testes planejados para o desenvolvimento do projeto vão desde simples códigos para familiarização com o ambiente e depuração do kit, passam por testes que serão base do funcionamento futuro até testes finais para validação da aplicação.

3.2.1 1ª Etapa - Depuração

Primeiramente, como já foi dito, serão realizados alguns testes básicos para familiarizar-se com o novo ambiente que envolve plataforma do microcontrolador (lógica interna do CI, forma de configuração dos registradores, etc), gravação do código, adaptação ao estilo da documentação, compilador, entre outros.

Para os testes, será necessária instalação do ambiente de desenvolvimento e de todos os *drivers* relativos aos equipamentos. O primeiro passo é a configuração do microcontrolador e o teste inicial com LED's, para verificar a configuração do timer e operação das portas.

Após, configura-se uma leitura simples de um conversor AD ligado a um *trimpot*, presente no kit. Atribui-se o valor dos bits menos significativos a cada LED

(total de 8 disponíveis) verificando a contagem binária mostrada ao operar o trimpot, concluindo o sucesso da leitura analógica.

O próximo passo será a comunicação serial, onde deve-se criar um menu que será enviado a um computador (utilizando o conversor USB-Serial) com o fim de receber comandos do mesmo. Quando solicitado deverá enviar mais informações contendo o valor atual da leitura AD previamente desenvolvida.

Assim, tem-se os primeiros passos concluídos que, mesmo parecendo simples, são importantes pelo processo de familiarização, garantia de funcionamento do kit e preparação para a próxima etapa.

3.2.2 2ª Etapa – Sistema operacional

Nesta segunda etapa, deverá ser feita a implementação do FreeRTOS. Seguindo toda a documentação e com a junção de trechos de alguns exemplos, o microcontrolador deverá rodar alguma aplicação genérica que demostre o funcionamento correto do OS. Por exemplo, temporização visualizada por LED's, resposta de botões, comunicação serial desde que garantida a funcionalidade advinda do OS.

Com o FreeRTOS em pleno funcionamento, precisarão ser criadas tarefas relativas ao funcionamento antes do OS como na 1ª etapa, ou seja, temporizadores, LED's, leitura AD e comunicação serial.

3.2.3 3ª Etapa – Aplicação

Para a terceira e última etapa, é necessário o pleno funcionamento de todos os testes anteriores. Neste ponto será criada a aplicação em si, pois serão executadas através do RTOS as tarefas de leitura múltipla de canais de AD, tarefas de cálculo para a otimização, tarefas de tratamento de código e, por fim, tarefas de envio destas leituras para o computador.

A validação desta etapa será complexa, pois envolverá teste com sinais diversos provindos de geradores de função, juntamente com análise das posições de memória e estudo minucioso da comunicação serial, para verificar se, de fato, a mesma está reduzida o suficiente sem repetir informações desnecessárias.

Para analisar os sinais, deverá ser realizada a integração com o software LabVIEW. Os grupos de sinais a serem analisados seguem um padrão, como ter a forma de uma senoide ou aleatório, possuir baixa frequência (ordem de kHz) ou alta frequência (ordem de MHz), individuais ou simultâneos em canais diferentes. Devem ser analisados:

- Uma senoide pura de baixa frequência➔verificar sua replicação no computador;
- Uma senoide pura com alta frequência➔ verificar o limite de frequência em que acontecem perdas;
- Várias senoides puras iguais com baixa frequência➔verificar leitura e armazenamento dos sinais;
- Várias senoides puras iguais com alta frequência➔verificar o limite de frequência em que acontecem perdas;
- Várias senoides puras diferentes com baixa frequencia➔ validação do método otimizado de armazenamento e leitura.

Capítulo 4 – Resultados

4.1 1ª Etapa - Depuração

Após um tempo dedicado em aprender as novas formas de programação da família PIC, foi possível realizar a programação necessária para concluir a primeira etapa.

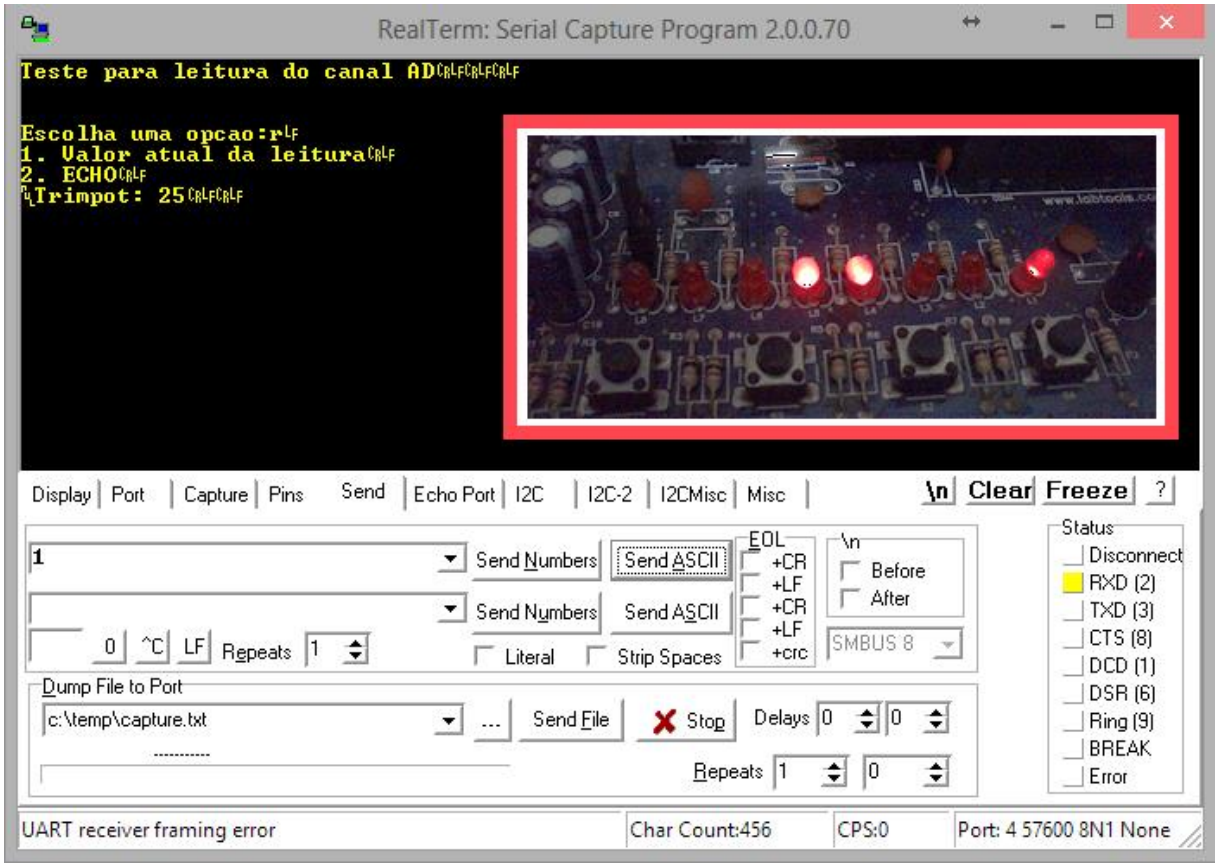


Figura 25 - Leitura 25 no A/D, comunicação serial e LED's

Como mostra a Figura 25, tem-se o valor da variável obtida pela leitura do AD visualizada direto do terminal serial RealTerm. Nota-se que o valor mostrado é 25, o que em binário fica 00011001. Observando os LED's do kit, acendem respectivamente de acordo com este valor. LED's a direita são menos significativos e acendem com nível lógico alto.

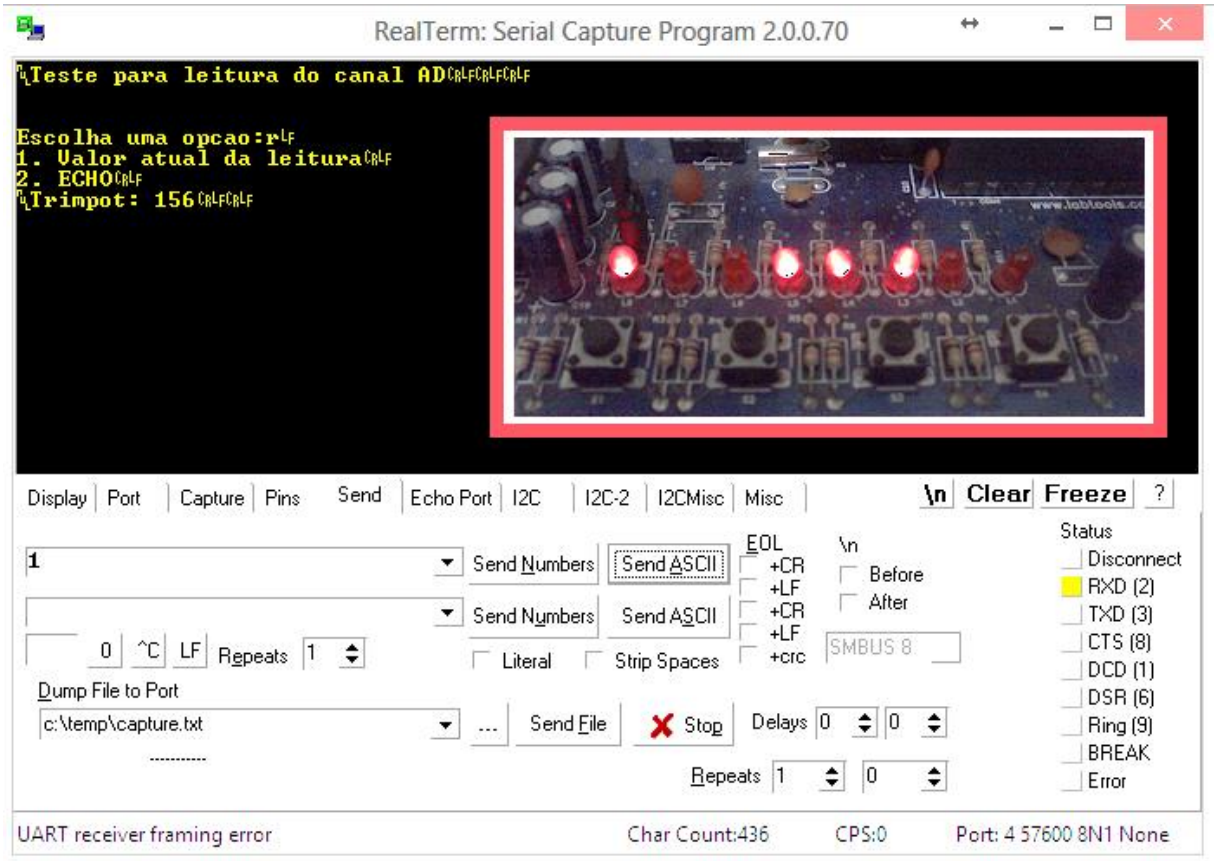


Figura 26 - Leitura 156 no A/D, comunicação serial e LED's

Seguindo o mesmo raciocínio, temos outro exemplo para o valor 156, que em binário fica 10011100. É importante dizer que, para a obtenção do valor, é preciso enviar a escolha do menu, representado pela opção 1. Somente ao receber esta opção é mostrado o valor atual.

4.2 2ª Etapa – Sistema operacional de tempo real

Esta etapa já era esperada que fosse a mais trabalhosa de todas e de fato, exigiu bastante empenho. Houve uma pesquisa muito grande sobre o entendimento de um RTOS, as particularidades do FreeRTOS e, por fim, o trabalho para funcionar no kit.

Foi necessário buscar informações em vários fóruns, estudar muitos exemplos (a maioria para outros microcontroladores) para entender como é organizado o

sistema operacional e assistir a muitos vídeos tutoriais sobre tarefas, semáforos, filas, etc.

Por fim, com o sistema operacional funcionando no kit, foram desenvolvidas todas as tarefas para cumprir as mesmas funções da 1ª etapa. Nas figuras seguintes temos a mesma operação de leitura do A/D (agora com resolução total disponível de 10 bits), comunicação serial com resposta de comando. As capturas de tela foram para os valores extremos da variável A/D.

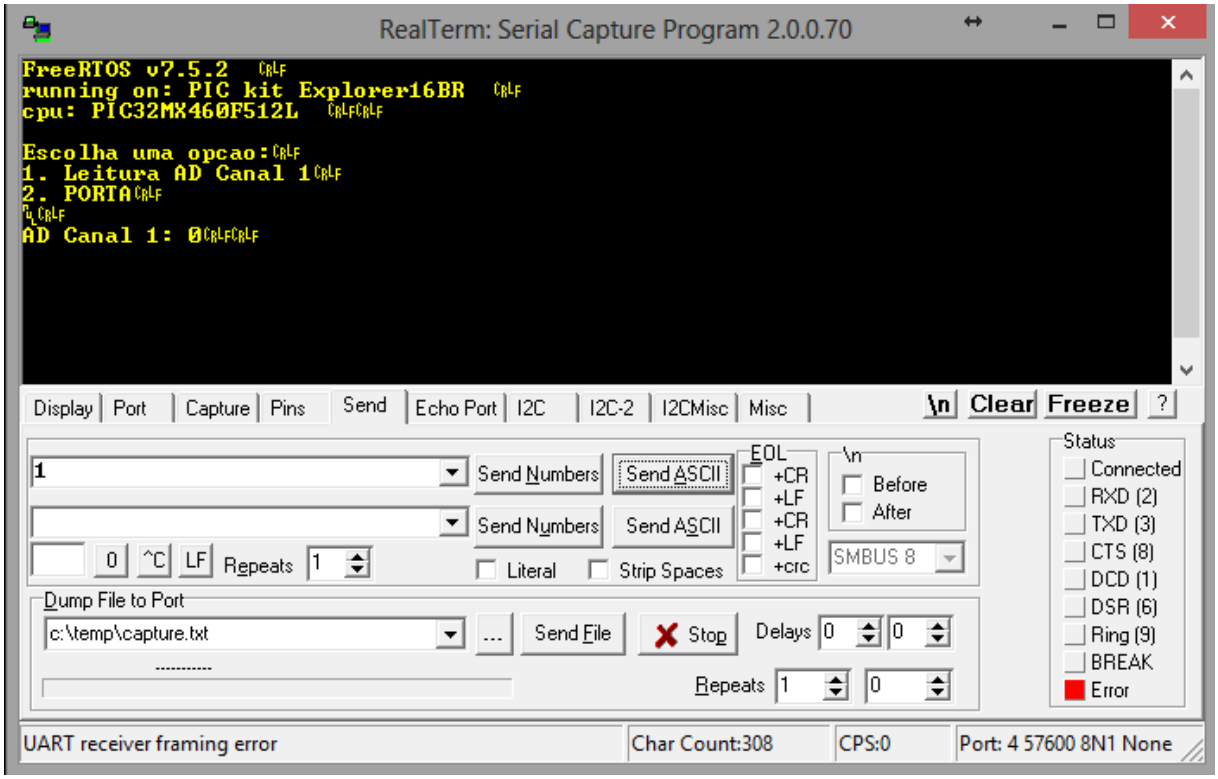


Figura 27 - Menu com FreeRTOS para valor mínimo do A/D

Nota-se que estes valores foram adquiridos utilizando um trimpot, presente no kit, ligado ao canal 1 do conversor A/D. Como a resolução é de 10 bits, teoricamente o valor da variável deveria excursionar entre 0 e 1023 (2^{10} = 1024), mas na medida real alcançou apenas 1015, devido a resistência do próprio trimpot e limite de excursão física do mesmo.

Da mesma forma da etapa anterior, o valor da variável solicitada da leitura do A/D só é mostrada após o envio do comando “1”. Uma observação a ser feita é sobre os símbolos CrLf que aparecem na mensagem que correspondem a *Carriage Return*

e *Line Feed*, responsáveis por retornar ao início da linha e ir para a próxima linha, respectivamente.

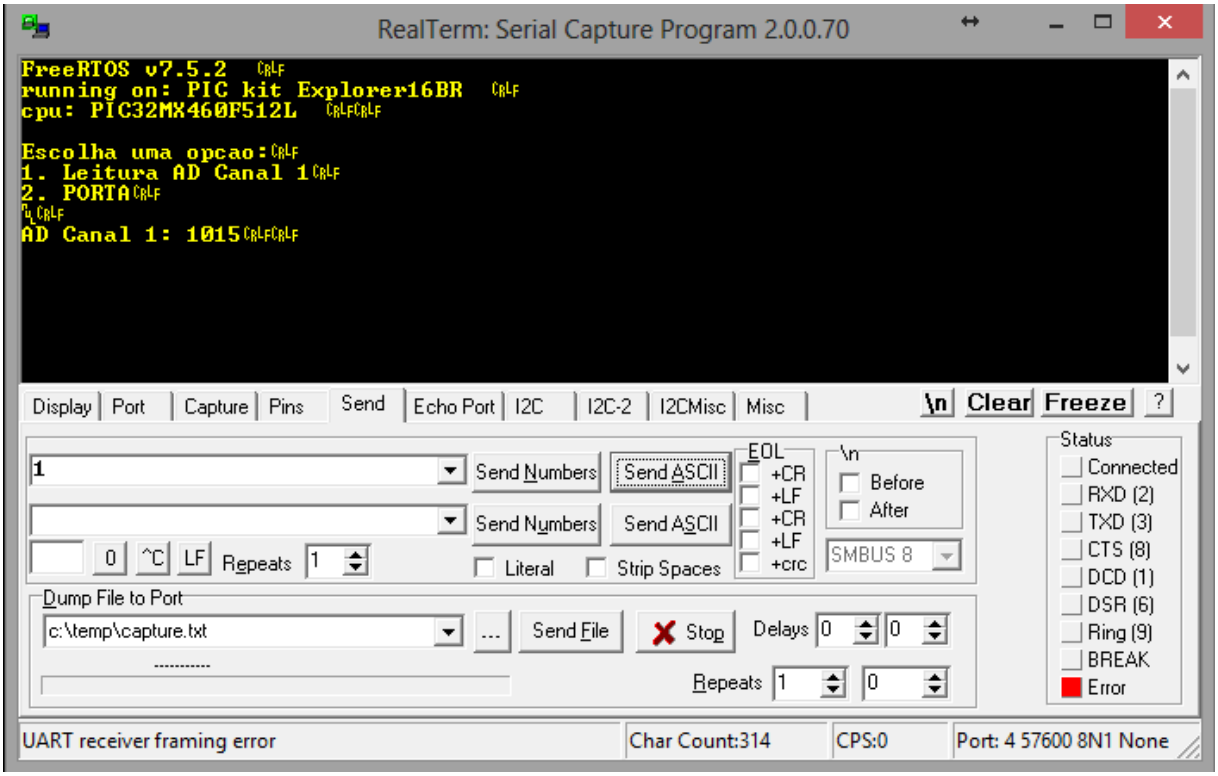


Figura 28 - Menu com FreeRTOS para valor de máximo do A/D

4.3 3ª Etapa – Aplicação

Nesta etapa, foi realizado, primeiramente, a integração com o sistema LabVIEW. Assim, o primeiro teste realizado foi a captura de tempo real do valor do *trimpot* ligado ao canal 1 do conversor A/D (presente no kit) sendo operado de forma manual.

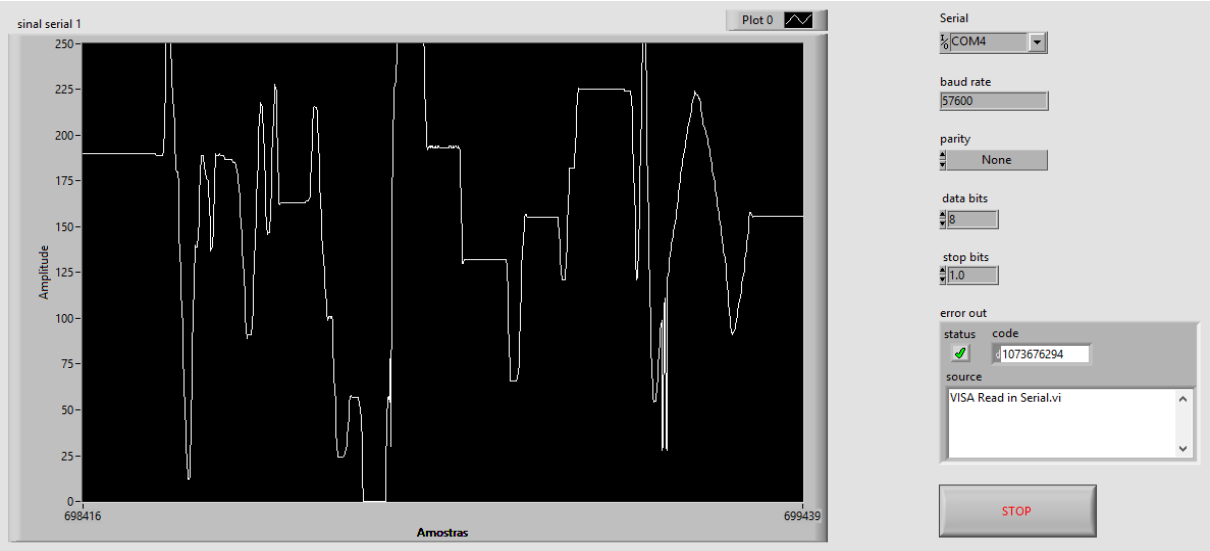


Figura 29 - Sinal gerado por operação manual de um trimpot

Na sequência foi realizada a inserção de 3 sinais senoidais defasados de aproximadamente 120º entre si. Para evitar erros e organizar a transmissão, foi determinado um protocolo simples para poder identificar os sinais enviados.

Seguindo o padrão da Figura 30 onde tem-se um cabeçalho para início de mensagem com valores que não poderão ser coincidentes em outros pontos da comunicação, seguido do *byte* identificador do sinal e por fim os dados.

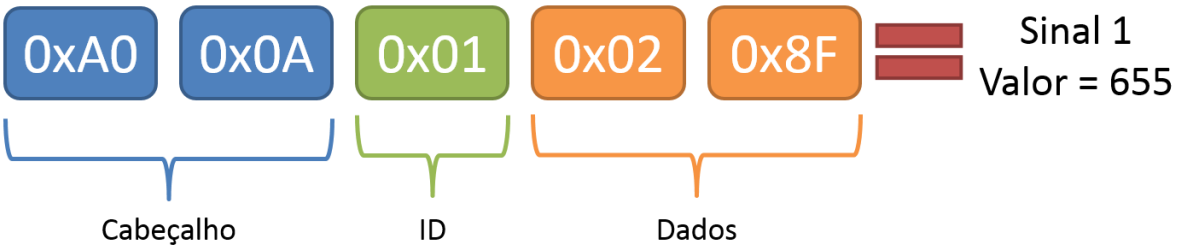


Figura 30 - Exemplo de comunicação utilizando o protocolo definido para identificação de múltiplos sinais

Nota-se que ao lado de cada gráfico da Figura 31 é mostrado o valor calculado para as fases, onde os valores obtidos são bem próximos dos desejados (0,0º;112,7º;236,9º). Foi tracejado a posição dos picos da onda para evidenciar a defasagem.

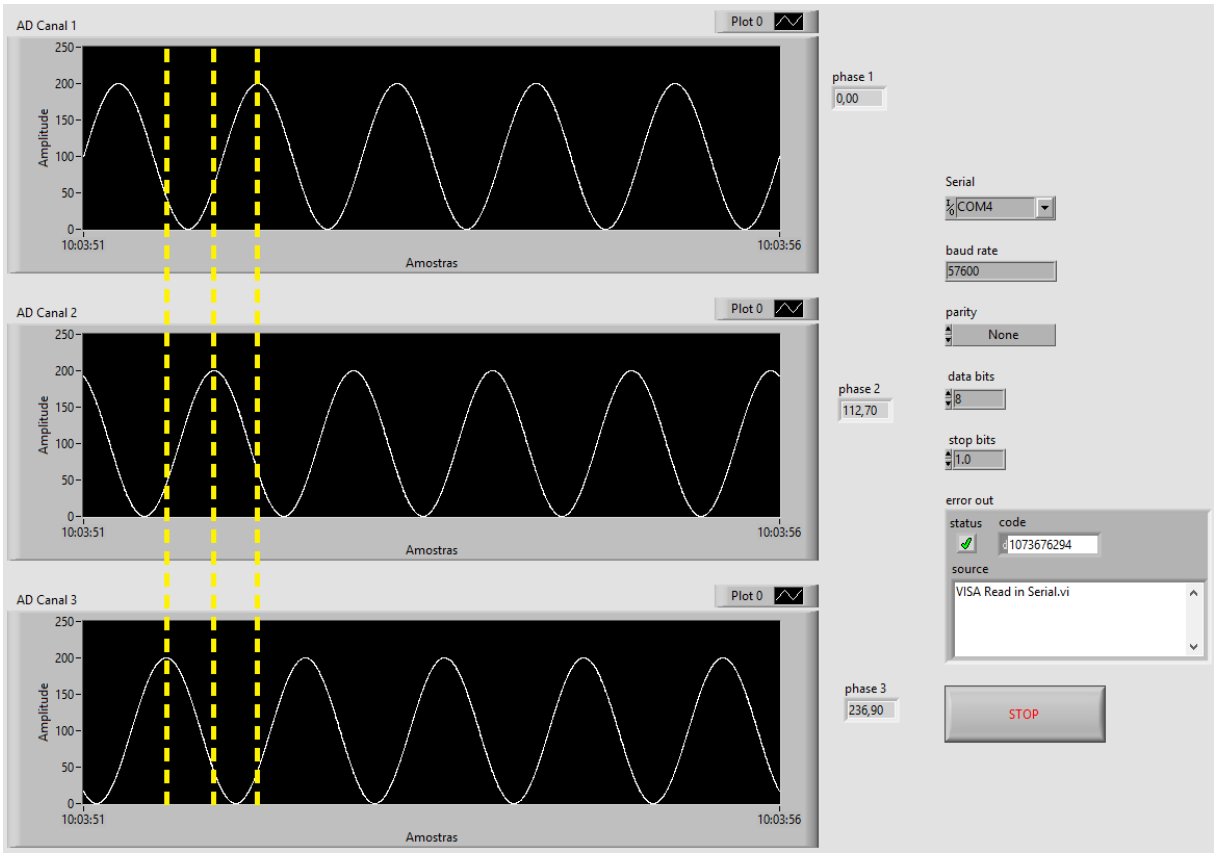


Figura 31 - Leitura de 3 sinais senoidais defasados entre si

Devido a imprevistos ao longo do desenvolvimento deste trabalho, e intensa dedicação exigida para a implementação do sistema operacional, os demais testes previstos para conclusão e validação da aplicação proposta ficarão para um trabalho posterior, já em fase de planejamento, a ser desenvolvido pelo mesmo autor.

Importante ressaltar que uma parte essencial para a continuação deste trabalho está concluída. Esta que será usada como base para o desenvolvimento futuro do algoritmo de otimização das posições de memória e os testes finais.

Capítulo 5 – Discussão e Conclusão

A conclusão da 1ª etapa demonstra a viabilidade do projeto, pois, assim, foi garantida a parte correspondente do hardware escolhido para ser utilizada no projeto, assim como todas as ferramentas adjacentes (como por exemplo, ambiente de desenvolvimento, drivers compatíveis, etc).

Nessa foi possível verificar aplicação dos conhecimentos prévios sobre microcontroladores necessários para se pensar no desenvolvimento de um projeto com tais características.

Com as consultas a *datasheets* e documentação, junto com exemplos fornecidos pela Microchip® foi possível realizar a configuração do conversor A/D e comunicação serial sem muitos problemas, realizando as leituras e envio sem erros.

O teste realizado consistiu em ler o valor atual de um canal do conversor A/D. Foi atribuído, também, seu valor binário a alguns LED's presentes no kit, para confirmação visual junto com o valor mostrado no terminal que captava a comunicação serial. Com isto, foram válidos os testes previstos para a conclusão desta 1ª etapa.

Para a 2ª etapa a fim de obter o efetivo funcionamento do FreeRTOS neste kit com este microcontrolador, foram necessárias várias tentativas e significativo tempo dedicado à pesquisa através de vídeos tutoriais, fóruns e a documentação fornecida pelo desenvolvedor do sistema operacional.

Existem muitos conceitos prévios relacionados ao modo de operação deste sistema. Muitas regras precisam ser cumpridas ao se desenvolver as tarefas, classificar as prioridades, enviar parâmetros para outras tarefas, manusear a fila, etc.

Porém, uma vez feitas essas configurações adequadamente, operar com um sistema operacional facilita aplicações complexas. São notáveis as facilidades que um escalonador próprio traz, não havendo a necessidade de se preocupar com temporização exata para captura de sinais, fila de envios na serial entre outras utilidades.

Durante os testes, foi possível perceber um efeito negativo quanto ao escalonador ser preemptivo, exigiu atenção maior para classificar as prioridades das tarefas. O cuidado de não permitir certas tarefas de interromper outras é fundamental quando se trata de aquisição de dados.

Como exemplo temos a seguinte *string* capturada (em hexa): “0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 30 36 3B 41 46 4B 51 56 5B 61 66 6C 71 76 7C 81 86 8C 91 97 9C A1 A7 AC B2 B7 BC C2 C7 CC D2 D7 DB D6 D1 CB C6 C0 BB B6 B0 AB A6 A0 9B 95 90 8B 85 80 7B 75 70 6A 65 60 5A 55 4F 4A 45 3F 3A 35 2F 2A 24 1F 19 14 0F 0A”

Trata-se de um sinal triangular, com a rampa ajustada para condizer com as amostras a cada 1µs. A intenção foi capturar um sinal perfeito com incrementos unitários até o topo e depois decrementos unitários. A variação seria entre 10 (0A em hexa) e 220 (DC em hexa).

No começo a captura está correta até o ponto 2D em hexa, onde devido a interrupções para envio e/ou conversões, a tarefa de captura começa a ser colocada em espera. Esta tarefa não era ativada por interrupção, logo, precisava obedecer a fila imposta pelo sistema operacional. A comunicação só era feita quando existia *byte* a ser enviado, o qual era obtido após a realização da tarefa de captura.

Deste modo, com a tarefa em espera, acabava por ocorrer saltos nos valores, um vez que o sinal externo continuava sua excursão independentemente. Ao priorizar a tarefa de captura, não mais ocorreu este problema, pois, quando necessário era a tarefa de envio que aguardava. Como possui os bytes armazenados em um vetor de *buffer*, não era uma tarefa crítica e não ocorriam perdas.

Observa-se que a espera da comunicação serial só foi possível porque não existia no teste um protocolo definido, onde um receptor aguarda por um tempo limite a recepção do próximo *byte* para validar a mensagem, assim como a transmissão não é síncrona, evitando qualquer disparidade com outro sinal ou com o receptor.

Para a 3ª etapa houve a necessidade de se trabalhar com o *software* LabVIEW® que é uma ferramenta projetada para facilitar o desenvolvimento por parte do usuário. Porém, por possuir inúmeras ferramentas completas com o intuito de contemplar aplicações complexas, necessita de uma preparação prévia para seu uso. Diversos problemas surgiram durante a elaboração do diagrama de blocos e instalação de drivers até chegar ao funcionamento desejado.

Em seguida, foi possível capturar sinais vindos da serial do kit através do conversor USB-RS232 e gerar gráficos dinâmicos em tempo real para análise. Antes, somente era capturado o valor instantâneo do conversor A/D ligado a um trimpot. Com este passo a mais realizado, foi possível visualizar em tempo real os valores do manuseio do trimpot em forma de um gráfico atualizado continuamente.

O grande destaque ao analisar este passo, se dá pelo fato do aprendizado sucinto porém efetivo do *software* e também por visualizar o poder desta nova ferramenta conhecida.

A partir deste ponto, foram incluídas nas tarefas de captura, uma lógica de organização do vetor de armazenamento para envio pela serial, de forma que o valor de cada sinal fosse enviado sequencialmente. Sendo Vx o valor capturado do sinal x, o envio da serial conteria o padrão V1 V2 V3 V1 V2 V3 V1…e assim sucessivamente.

De volta ao LabVIEW®, foi necessário criar uma lógica de blocos para separar estes *bytes* e enviá-los a seus respectivos gráficos. O sinal foi gerado a partir de uma única fonte e defasados utilizando capacitores e os gráficos mostraram que todo o processo de captura e envio dos 3 sinais estava funcionando.

Assim, verificou-se a configuração de leitura de múltiplos canais no conversor A/D, os ajustes corretos quanto às prioridades das tarefas no FreeRTOS e também a criação dos blocos necessários para geração dos gráficos no LabVIEW®.

Com tudo isto pronto, as próximas fases serão a criação das tarefas para identificar características dos sinais e o algoritmo de otimização da leitura das posições de memória. Na sequência a realização dos testes indicados, como por exemplo, a verificação da frequência máxima do sinal suportada sem perdas.

Neste trabalho, não foi possível a realização destas últimas fases devido a imprevistos nas questões de tempo. No entanto, este trabalho deixa uma plataforma com toda base preparada para futuros desenvolvimentos de aplicações. O sistema está validado, o FreeRTOS está funcionando e as tarefas básicas estão criadas.

Por fim, chega-se à conclusão que a ideia disposta neste trabalho terá grande utilidade. É possível ver a potencialidade desta ferramenta para aquisição de dados e análise dos sinais. Vale lembrar que com um sistema operacional, como o FreeRTOS, em funcionamento, é possível inserir novas tarefas que aprimorem o código já feito, assim como algoritmos de tratamento e quaisquer outras funções que venham a ser necessárias. Por exemplo, pode-se criar tarefas com uma Transformada de Fourier aplicando apenas em 2 canais, antes de ser enviado a outro dispositivo.

Referências

[1] Revista Mecatrônica Atual. **O que é um sistema operacional de tempo real (RTOS)?**. São Paulo: Editora Saber, n.60, ano 11, Jan/Fev, 2013.

[2] Wikimedia Commons. (Consultado em Outubro, 2013). **File:Operating system placement-pt.svg** [Online]. Disponível em: http://commons.wikimedia.org/wiki/File:Operating_system_placement-pt.svg?uselang=pt

[3] COVACECIVE, A.V.T. **Sistemas Operacionais de Tempo-Real**. UNICAMP. Out, 2007.

[4] GONÇALVES, L.R.O. (Consultado em Outubro, 2013). **Apostila de SO On-line** [Online]. Disponível em: http://lrodrigo.lncc.br/index.php/Apostila_de_SO_On-line

[5] TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª ed. São Paulo, Pearson Prentice Hall, 2003.

[6] FreeRTOS.org (Consultado em Outubro, 2013). **What is an RTOS/FreeRTOS?** [Online]. Disponível em: http://www.freertos.org/about-RTOS.html

[7] OPPENHEIM, A., SCHAFER R., and BUCK, J. **Discrete-Tme Signal Processing**. 3ª ed. Prentice Hall. 1999.

[8] TATEOKI, G.T. (Consultado em Outubro, 2013). **Teorema da Amostragem** [Online]. Disponível em: http://getulio.eng.br/meusalunos/sad/Teorema_da_Amostragem.pdf

[9] SICA, C. **Sistemas Automáticos com Microcontroladores 8031/8051**. Editora Novatec. 2006

[10] FERREIRA, E.C. **Conversão AD e DA – Técnicas**. UNICAMP. 2009.

[11] FREIRE, R.C.S. **Conversão A/D e D/A**. UFPI. 2010

[12] Mosaico Produtos (Consultado em Outubro, 2013). Plugin Explorer16BR PIC32MX460F512L-80I/PT USB [Online]. Disponível em: http://www.mosaico.com.br/?canal=5&pg=showProduto&path=produtos&id=108

[13] SILVA, N.C. **Introdução à linguagem C**. 2ª ed. Centro de Computação UNICAMP. 2011.