CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB
FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS
CURSO DE ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA: PROJETO FINAL

ELETROLISADOR MICROCONTROLADO DA ÁGUA

RAFAEL CHAGAS DE MORAES COSTA
RA: 2036757/5

Prof. MSc. Epitácio Pinto Marinho
Orientador

Brasília-DF, dezembro de 2009

RAFAEL CHAGAS DE MORAES COSTA

ELETROLISADOR MICROCONTROLADO DA ÁGUA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/FATECS) como requisito para a obtenção de Certificado de Conclusão do Curso de Graduação de Engenharia de Computação.

Orientador: Prof. MSc. Epitácio Pinto Marinho

Brasília-DF, novembro de 2009

RAFAEL CHAGAS DE MORAES COSTA

ELETROLISADOR MICROCONTROLADO DA ÁGUA

DATA DE APRESENTAÇÃO A BANCA

1 de dezembro de 2009

NOME DOS MEMBROS DA BANCA

Prof. M.C. Epitácio Pinto Marinho
Prof[a]. M.C. Maria Marony Sousa Farias Nascimento
Prof. M.C. Flávio Antônio Klein
Prof. M.C. Luis Cláudio

Agradecimentos

Agradeço a Deus em primeiro lugar pela sua majestosa criação e misericórdia. Agradeço aos meus pais por serem responsáveis pelas grandes conquistas de minha vida. A educação que deles recebi é um imenso legado de sabedoria na qual sempre me conduz ao caminho certo. Não poderia deixar de agradecer profundamente a paciência e compreensão de minha esposa Janaína que sempre esteve ao meu lado nos momentos mais difíceis. Sem a sua compreensão certamente não teria chegado até aqui. Agradeço aos meus irmãos Bruno e Daniel e minha tia Rita pelo apoio sincero que sempre me deram. Não poderia deixar de agradecer aos meus amigos Elves, sua esposa Sandra e Julio pelo apoio que me deram. Agradeço também a todos os professores do UniCeub que me apoiaram ao longo do curso de engenharia da computação.

Dedico este trabalho a minha filha Giovanna cuja presença me enche de graça e faz tudo isso fazer sentido.

Resumo

O projeto Eletrolisador Micro-controlado da Água trata do desenvolvimento de um sistema eletroquímico destinado a produção experimental de gás hidrogênio e oxigênio através do processo de eletrólise da água. Este sistema será utilizado para fazer uma série de análises a respeito da eletrólise convencional. Como complementação, pretende-se através desse trabalho, verificar também algumas relações entre a aplicação de sinais elétricos específicos e a produção de gás associada.

As atividades envolvidas na elaboração desse trabalho consistem em fornecer a fundamentação teórica para o entendimento básico do projeto, apresentar a solução proposta, além de demonstrar as etapas de desenvolvimento e construção dos itens que compõem o sistema Eletrolisador Micro-controlado da Água. Ao final do trabalho será apresentado o conjunto de informações que comprovam a aplicação do projeto proposto.

Palavras-chave: Eletrólise da água, eletrodos, gerador de sinais, micro-controlador.

Abstract

The project Water Electrolyser Microcontrolled is the development of an electrochemical system for experimental production of hydrogen and oxygen gas through the process of electrolysis of water. This system will be used to make a series of analysis about the conventional electrolysis. Also this work is intended to check some relations between the application of specific electrical signals and associated gas production.

The activities involved in the preparation of this work will to provide the theoretical foundation for the basic understanding of the project, submit the proposed solution, and demonstrate the stages of development and construction of items that make up the Water Electrolyser Micro-controlled system. At the end of this work will covered the whole range of information to demonstrate the application of the proposed project.

Keywords: Water Electrolysis, electrodes, signal generator, PIC microcontroller.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES E SIGLAS

CI – Circuito Integrado. Circuito eletrônico dedicado encapsulado em um chip.
CMOS - É uma sigla para *complementary metal-oxide-semiconductor*, i.e., semicondutor metal-óxido complementar.
GPR – Registradores de propósitos Gerais (*General Purpose Register*)
MCU – Unidade Micro-controlada. Sigla em inglês de *Microcontroller Unit*.
SFR – Registradores de funções especiais (*Special Function Registers)*
TTL - A Lógica Transistor-Transistor (Transistor-Transistor Logic ou simplesmente TTL) é uma classe de circuitos digitais construídos de transistores de junção bipolar (BJT), e resistores.

Capítulo 1. Introdução

O processo de geração da eletrólise da água possui um potencial muito grande no que diz respeito às aplicações no uso do gás hidrogênio. Com a maturidade adquirida no desenvolvimento e uso de células de combustível, principalmente aquelas onde o gás utilizado na conversão para energia elétrica é o hidrogênio. Além da possibilidade de produção de energia elétrica através do hidrogênio, esse gás de grande poder de combustão é uma fonte de energia renovável, já que a queima dele resulta novamente em vapor de água. É um ciclo perfeito, utiliza-se energia elétrica na eletrólise para obter hidrogênio (H_2) e oxigênio (O_2) a partir da água (H_2O) e na queima da molécula de hidrogênio com oxigênio é liberado energia em forma de calor e novamente é criada a molécula da água em forma de vapor. A energia em forma de calor produzida pela queima do hidrogênio pode ser utilizada em diversas aplicações residenciais ou industriais.

O projeto Eletrolisador Micro-controlado da Água consiste em um projeto de engenharia cujo foco é o desenvolvimento de um sistema de geração da eletrólise da água. Este projeto engloba conceitos das áreas de eletrônica e computação bem como da física e química. O objetivo deste trabalho é projetar e construir o protótipo de um equipamento eletrônico capaz de gerar a eletrólise da água de forma controlada. Este projeto fornece um conjunto mínimo de informações a respeito do processo convencional de produção dos gases hidrogênio e oxigênio através da eletrólise. Por meio do protótipo construído foram feitas análises a cerca da eletrólise em três diferentes tipos de solução aquosa. Foi analisado o comportamento da eletrólise com a aplicação de sinais elétricos em forma de pulsos nos eletrodos do eletrolisador com as seguintes soluções: água potável; água destilada e por último será analisado o comportamento da técnica em água adicionada de ácido sulfúrico. Ao final do trabalho há a comparação dos resultados da aplicação desse tipo de eletrólise nas diferentes soluções.

1.1. Motivação

A busca por alternativas energéticas menos agressivas ao meio ambiente tem sido preocupação de parcela significativa da população e de muitos governos, a crise do petróleo no século passado não só interferiu na economia global como trouxe novos questionamentos acerca de danos ambientais trazidos pela matriz energética alimentada por combustíveis fósseis. O Brasil naquele momento apresentou alternativa bem menos poluente que acabou se tornando realidade, o álcool passou a integrar o complexo sistema de combustível para automóveis. O investimento em diversificação das fontes energéticas no Brasil tem aumentado significativamente. Combustíveis oriundos de processos renováveis são comuns no país. Destacam-se a produção do álcool e do biodiesel. Outras soluções têm sido experimentadas, mas o mais importante é que o debate está longe de ser encerrado. Um dos combustíveis renováveis muito cotado na atualidade é o gás hidrogênio. O uso desse gás como combustível já é realidade em países desenvolvidos como a Alemanha.

Muito embora seja possível considerar a água a fonte de energia mais limpa existente na terra – e isto é inquestionável tendo em vista a sua estrutura ser composta de hidrogênio e oxigênio – é sabido que o seu estado natural não favorece qualquer tipo de combustão, muito pelo contrário, a associação molecular H_2O (Água pura) é utilizada até mesmo para impedir uma combustão. No entanto, a quebra da molécula da água de maneira apropriada resulta na produção de gases hidrogênio e oxigênio. Estes gases podem ser utilizados juntos ou separadamente. Esses dois gases juntos formam um combustível perfeito.

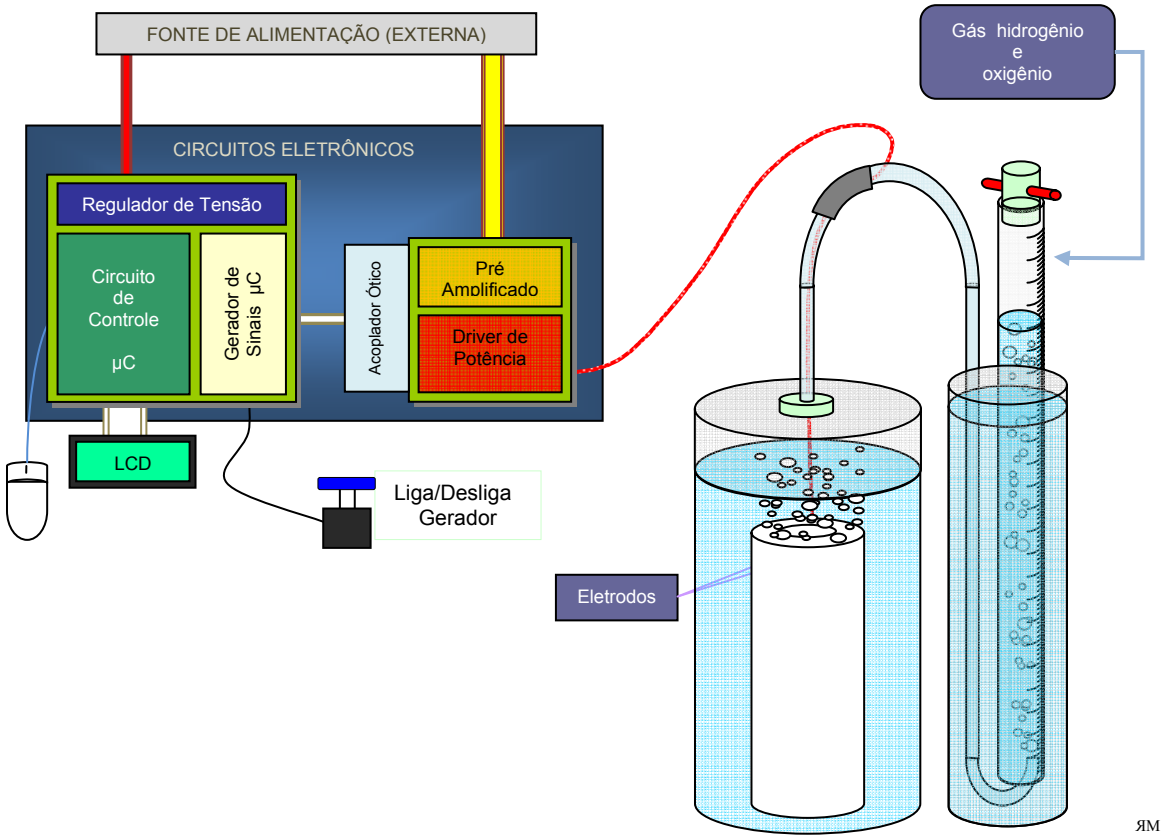
O desenvolvimento desse trabalho foi motivado pelos fatores colocados até aqui e também pela curiosidade a respeito dos processos de quebra da molécula da água para a produção de gás combustível. Além disso, o projeto deverá ser utilizado para verificar se há ou não ganho de produtividade no processo convencional da eletrólise quando da aplicação de sinais elétricos diferenciados.

1.2. Objetivos

O objetivo principal deste trabalho é criar o modelo de um sistema eletrolisador da água, além disso, o projeto conta com a construção de um protótipo eletrônico e mecânico para a realização prática do experimento da eletrólise proposto neste trabalho. O projeto prevê a construção do circuito eletrônico e do artefato físico onde ocorrerá o fenômeno da eletrólise. O circuito eletrônico é capaz de gerar sinais elétricos em forma de pulsos que são introduzidos nos eletrodos do artefato eletrolisador. O projeto foi divido em etapas conforme a descrição abaixo:

- Definição dos componentes do sistema eletrônico
- Definição do programa computacional do sistema
- Construção do esquema eletrônico do eletrolisador micro-controlado
- Construção do artefato eletrolisador
- Montagem do circuito eletrônico em *protoboard*
- Testes e ajustes finais
- Aplicação e Resultados

A imagem abaixo ilustra todo o sistema do Eletrolisador Micro-controlado da Água:



Capítulo 2. Fundamentação Teórica

2.1. A Química e a Eletrolise

O assunto abordado nesse trabalho requer uma revisão de alguns tópicos básicos do estudo da química. A realização do fenômeno da eletrólise é o produto final da aplicação dos estudos aqui tratados. Desta forma, neste tópico serão revisados alguns conceitos básicos sobre os elementos químicos. Para se entender o que ocorre no cerne do Eletrolisador Micro-controlado da Água, é necessário que tais conceitos sejam revisados.

2.1.1. Estrutura Atômica

Na antiguidade havia uma crença de que a matéria poderia ser dividida em pedaços cada vez menores e que em determinado momento essa divisão não poderia mais ocorrer, ou seja, a matéria se tornaria indivisível. Em grego, o nome átomo significa exatamente *indivisível*. Foi daí que surgiu o termo *atomismo* entre os filósofos gregos. Embora muitas teorias tenham sido feitas, na atualidade o modelo aceito para definir a estrutura atômica é o Modelo da Mecânica Quântica ou da Mecânica Ondulatória ou Modelo Orbital ou da Nuvem Eletrônica. [12]

No modelo atômico atual, se sabe que os elétrons possuem carga negativa, massa muito pequena e que se movem em órbitas ao redor do núcleo atômico. O núcleo atômico é situado no centro do átomo e constituído por prótons que são partículas de carga positiva, cuja massa é aproximadamente 1837 vezes superior a massa do elétron, e por nêutrons, partículas sem carga e com massa ligeiramente superior à dos prótons. O átomo é eletricamente neutro, por possuir números iguais de elétrons e prótons. O número de prótons no átomo se chama número atômico, este valor é utilizado para estabelecer o lugar de um determinado elemento na tabela periódica. A tabela periódica é uma ordenação sistemática dos elementos químicos conhecidos. Cada elemento se caracteriza por possuir um número de elétrons que se distribuem nos diferentes níveis de energia do átomo correspondente. Os níveis energéticos ou camadas são denominados pelos símbolos K, L, M, N, O, P e Q. Cada camada possui uma quantidade máxima de elétrons. A camada mais próxima do

núcleo K comporta somente dois elétrons; a camada L, imediatamente posterior, oito, e assim segue em uma seqüência não linear. Os elétrons da última camada (mais afastados do núcleo) são responsáveis pelo comportamento químico do elemento, por isso são denominados elétrons de valência. O número de massa é equivalente à soma do número de prótons e nêutrons presentes no núcleo. O átomo pode perder elétrons, carregando-se positivamente, é chamado de íon positivo (cátion). Ao receber elétrons, o átomo se torna negativo, sendo chamado íon negativo (ânion). O deslocamento dos elétrons provoca uma corrente elétrica, que dá origem a todos os fenômenos relacionados à eletricidade e ao magnetismo. No núcleo do átomo existem duas forças de interação a chamada interação nuclear forte, responsável pela coesão do núcleo, e a interação nuclear fraca, ou força forte e força fraca respectivamente. As forças de interação nuclear são responsáveis pelo comportamento do átomo quase em sua totalidade. As propriedades físico-químicas de determinado elemento são predominantemente dadas pela sua configuração eletrônica, principalmente pela estrutura da última camada, ou camada de valência. As propriedades que são atribuídas aos elementos na tabela se repetem ciclicamente, por isso se denominou como tabela periódica dos elementos. [12]

Íons

Números de massa e as massas atômicas concentram sobre o núcleo do átomo. Pouco se tem dito sobre os elétrons porque a massa dos elétrons é desprezível comparada com a massa dos prótons e nêutrons. Elétrons têm somente cerca de 1/2000 da massa de prótons e nêutrons. Para um átomo permanecer eletricamente neutro, o número de elétrons deve ser igual ao número de prótons. Quando um átomo neutro ganha ou perde elétrons, surge uma partícula carregada chamado de íons. Esse processo é conhecido como **ionização**. Íons positivos são referidos como **cátions**, e íons negativos são chamados **ânions**. Átomos ganham ou perdem um ou mais elétrons para se tornar íons. Os íons são escritos usando o símbolo de elementos e escreve-se a carga usando um sobrescrito. Uma equação simples pode ser escrita para simbolizar o processo de ionização. Por exemplo, quando o lítio perde um elétron para formar Li^+, a equação é:

$$Li \rightarrow Li^+ + e^-$$

Outras equações que representam o processo de ionização são:

$$F + e^- \rightarrow F^-$$

$$Ca \rightarrow Ca^{2+} + 2e^-$$

Átomos não ganham ou perdem elétrons de forma aleatória. As reações químicas envolvem a perda e o ganho de elétrons. Na verdade, o comportamento de todas as substâncias químicas é ditado pela forma como os elétrons das substâncias interagem quando as substâncias estão reunidas.

2.1.2. Eletroquímica

A eletroquímica é uma área da ciência que lida com as interações entre a energia elétrica e a química. A eletroquímica está presente no cotidiano das pessoas embora muitas delas não se dêem conta disso. Um exemplo disso é o uso de baterias em equipamentos portáteis. O uso de baterias em celulares, câmeras e computadores portáteis é indispensável.

Redução e Oxidação

No passado, a combinação de um elemento com o oxigênio foi a maneira tradicional de se definir a reação de oxidação. Esta definição da oxidação foi estendida pela química para incluir reações que não envolvem o oxigênio. Na visão moderna da oxidação, esta ocorre quando uma substância perde elétrons. Toda vez que uma oxidação ocorre e uma substância perde um ou mais elétrons, outra substância deve receber o(s) elétron(s). Quando uma substância ganha um ou mais elétrons, o processo é conhecido como redução. Reações que envolvem a transferência de um ou mais elétrons sempre envolvem ambos a oxidação como a redução. Estas reações são conhecidas como reações de **oxirredução**. [7]

Considere a simples reação do magnésio com o oxigênio para formar o óxido de magnésio:

$$2Mg + O_2 \rightarrow 2MgO$$

Nesta reação, cada um dos dois átomos de magnésio doa dois elétrons aos dois átomos de oxigênio. Neste processo cada átomo de magnésio torna-se Mg^{2+} e cada átomo de oxigênio torna-se O^{2-}

$$Mg \rightarrow Mg^{2+} + 2e^- \qquad O + 2e^- \rightarrow O^{2-}$$

Devido ao balanceamento das equações envolverem dois átomos de magnésio e dois átomos de oxigênio, a equação anterior é mais apropriada sendo escrita como:

$$2Mg \rightarrow 2Mg^{2+} + 4e^- \qquad O_2 + 4e^- \rightarrow 2O^{2-}$$

Neste exemplo, a soma das reações é dada abaixo:

$$2Mg \rightarrow 2Mg^{2+} + 4e^- \qquad \textbf{Oxidação}$$

$$+ O_2 + 4e^- \rightarrow 2O^{2-} \qquad \textbf{Redução}$$

$$2Mg + O_2 + 4e^- \rightarrow 2Mg^{2+} + 2O^{2-} + 4e^- \qquad \textbf{Equação Global}$$

Número de Oxidação

Tomando como exemplo a oxidação do magnésio, cada átomo de magnésio perde dois elétrons e adquire a carga de +2, e cada átomo de oxigênio receber esses dois elétrons e adquire a carga de -2. A carga ou carga aparente que um átomo possui ou adquiriu é chamada de número de oxidação. O número de oxidação do magnésio em MgO é +2, e do oxigênio é -2.

O conceito de oxidação tem sido expandido da simples combinação com o oxigênio para um processo onde ocorre transferência de elétrons. Oxidação não pode ocorrer sem a redução, e os números de oxidação podem ser utilizados para resumir as transferências de elétrons em reações de oxirredução. Este conceito básico pode ser aplicado ao princípio das células eletroquímicas, eletrólise e aplicações da eletroquímica. [7]

Células Eletroquímicas

Ao se colocar uma tira de zinco em um recipiente com água e a solução sulfato de cobre $CuSO_4$ é possível verificar diversas mudanças ocorrendo. O zinco imediatamente se escurece e com o passar do tempo observa-se o aparecimento da cor azul na solução $CuSO_4$. As mudanças observadas resultam de uma reação química espontânea envolvendo a oxidação do zinco e a redução do cobre. O cobre na solução existe como íons Cu^{2+}. Esses íons são reduzidos tornando o cobre sólido e o metal de zinco é oxidado para íons Zn^{2+}. Este experimento simples demonstra uma simples reação redox envolvendo o zinco e o cobre:

$$Zn_{(s)} + CuSO_{4(aq)} \rightarrow Cu_{(s)} + ZnSO_{4(aq)}$$

Na reação acima, a troca de elétrons que ocorre entre o cobre e o zinco é feita diretamente na superfície do metal. Este é o princípio da célula elétrica ou mais conhecida como pilha elétrica. Uma célula eletroquímica é um arranjo nas quais reações redox são utilizadas para a geração de energia elétrica. [7]

Pilha de Volta

A pilha de Volta possui esse nome em honra ao cientista italiano Alessandro Volta (1745-1827). Volta através de seus estudos científicos queria provar que a eletricidade poderia ser produzida com o emprego de metais. Em 1800, Volta construiu um equipamento capaz de produzir corrente elétrica continuamente. Daí surgiu a pilha de Volta. Esta pilha foi construída com discos de zinco e cobre que ficavam empilhados uns em cima dos outros. Estes discos eram separados por pedaços de tecido embebidos em solução de ácido sulfúrico. Ao ligar um fio condutor entre os discos da extremidade da pilha, uma passagem de corrente elétrica ocorria nesse condutor. [12]

Figura 1 – Pilha de Volta.
Fonte: Wikipédia.

Diferença de Potencial (*ddp*)

Diferença de potencial ou **Tensão Elétrica** é a diferença de potencial elétrico entre dois pontos. A unidade de medida utilizada é o Volt, em homenagem ao físico Alessandro Volta. A tensão elétrica é a força eletromotriz, ou seja, a força responsável pela movimentação dos elétrons. O potencial elétrico mede a força que uma carga elétrica experimenta no âmago de um campo elétrico. Esta força é expressa pela lei de Coulomb[1]. [12]

A tensão elétrica entre dois pontos, ou seja, [(+) e (-)] é definida matematicamente como a integral de linha do campo elétrico:

$$V_a - V_b = \int_a^b \mathbf{E} \cdot d\mathbf{l} = \int_a^b E \cos\phi dl.$$

A tensão elétrica também pode ser expressa pela lei de Ohm[2]:

$$U = R \cdot I$$

Onde:

- R = Resistência (Ohms)
- I = Intensidade da Corrente (Ampères)
- U = Diferença de Potencial ou tensão (Volts)

2.1.3. Fundamentos da Eletrólise

Células eletroquímicas produzem energia elétrica através de reações químicas espontâneas. Na eletrólise, o processo é revertido, energia elétrica é utilizada para realizar uma reação química não espontânea. O arranjo de células que executa esse processo é chamado de célula eletrolítica. O processo da eletrólise é uma reação de **oxirredução**. A palavra *eletrólise* é originária dos radicais gregos *eletro* (eletricidade) e *lisis* (decomposição). [12]

[1] **Lei de Coulomb** – trata do princípio fundamental da eletricidade. Em particular, diz-nos que o módulo da força entre duas cargas elétricas puntiformes (q1 e q2) é diretamente proporcional ao produto dos valores absolutos (módulos) das duas cargas e inversamente proporcional ao quadrado da distância r entre eles. [12]

[2] **A Primeira Lei de Ohm**, assim designada em homenagem ao seu formulador Georg Simon Ohm, indica que a diferença de potencial (V) entre dois pontos de um condutor é proporcional à corrente elétrica. [12]

Processo Eletrolítico[3]

Para que o fenômeno da eletrólise possa ocorrer é necessário o uso de uma substância que dissociada ou ionizada, produza íons positivos (cátions) e íons negativos (ânions), pela adição de um solvente ou aquecimento. O nome dado a esta substância é eletrólito. O eletrólito é por si só, condutor de eletricidade. Além do eletrólito, na eletrólise é necessário também o uso de eletrodos para introduzirem a energia elétrica no processo. Dependendo do tipo de eletrodo e do modo de obtenção dos íons que constituem o eletrólito, as reações que ocorrem no processo eletrolítico são diferentes. [12]

Solução Eletrolítica

A solução que contém os íons livres derivados do eletrólito é chamada de *solução eletrolítica*. Quando o eletrólito dissocia parcialmente, estes íons coexistem, em equilíbrio com este eletrólito. Devido à existência de íons livres, a solução eletrolítica tem a capacidade de conduzir a corrente elétrica. [12]

- Solução Eletrolítica Aquosa – É a aquela cujos íons foram solvatados[4] pela água.

- Solução Eletrolítica Ígnea – É aquela cujos íons foram liberados por aquecimento (processo de fusão).

Eletrólitos

Eletrólito Potencial – É aquele eletrólito que não apresenta íons, ou seja, é constituído de unidades estruturais denominadas moléculas que são um agrupamento definido e ordenado de átomos, eletricamente neutro; é a menor partícula dos compostos ou dos elementos simples, que é quimicamente idêntica a substância de que faz parte.

---

[3] **Eletrolítico** – *adj (eletrólito+ico[2])* **1** Concernente à eletrólise. **2** Produzido por eletrólise. *Var: electrolítico* [6]

[4] **Solvatação** – Em química se entende por solvatação o fenômeno que ocorre quando um composto iônico ou polar se dissolve em uma substância polar, sem formar uma nova substância. As moléculas do soluto são rodeadas pelo solvente. A solvatação acontece tanto em soluções iônicas quanto moleculares. [12]

Eletrólito Intrínseco – É o eletrólito que já apresenta íons, porém, fortemente ligados formando um conjunto iônico sólido e cristalino. Os íons são liberados por fusão ou por adição de um solvente polar[5].

Eletrodos na Eletrólise

Eletrodos Inertes – São eletrodos que funcionam apenas como terminais do gerador de eletricidade, cedendo e recebendo elétrons. Este tipo de eletrodo não participa da reação.

Eletrodos reativos – São aqueles eletrodos que sofrem com a oxidação (perda de elétrons). Isto ocorre porque o eletrodo tem mais facilidade para se oxidar que os ânions do eletrólito.

Eletrólise na Prática

Na prática, a eletrólise é utilizada em várias aplicações. Por meio desse processo é possível obter elementos químicos como metais, hidrogênio, oxigênio e cloro. Substâncias como a soda cáustica[6] e água oxigenada (H_2O_2) também são obtidas pelo processo da eletrólise. Com este processo é possível executar a purificação de metais como o cobre e outros.

Por meio de um experimento simples da eletrólise da água é possível obter os gases hidrogênio e oxigênio. Este processo quebra a molécula da água (H_2O) através da reação química de oxirredução. Neste experimento, a reação química da eletrólise ocorre em dois eletrodos. O eletrodo na qual a oxidação ocorre é chamado de ***anodo***; e aquele onde a redução ocorre é chamado de ***catodo***. A eletricidade passa através de um circuito sobre a influência de um potencial ou voltagem, a força motriz do movimento de carga. [4]

[5] **Molécula Polar** – É uma molécula em que as polaridades das ligações individuais não se cancelam. Ex: H_2O. [12]
[6] **Soda Cáustica** – Hidróxido de sódio (NaOH), também conhecido como soda cáustica, é um hidróxido cáustico usado na indústria (principalmente como uma base química) na fabricação de papel, tecidos, detergentes, alimentos e biodiesel. Reage de forma exotérmica com a água e é produzido por eletrólise de uma solução aquosa de cloreto de sódio (salmoura). [12]

Em um recipiente é colocada uma solução de água misturada com ácido sulfúrico. A adição do ácido é necessária para tornar a água condutiva e permitir a ocorrência da reação de oxirredução. Para ocorrer à eletrólise é necessário introduzir dois eletrodos na solução aquosa e inserir uma diferença de potencial nos eletrodos através de uma fonte de energia externa. Com a aplicação de energia nos eletrodos é possível verificar o surgimento de bolhas de gás em torno dos eletrodos. Com o auxilio de duas provetas completas de água e posicionadas cada uma com seu respectivo eletrodo inserido, é possível recolher os gases formados, conforme pode ser visto na Figura 2.

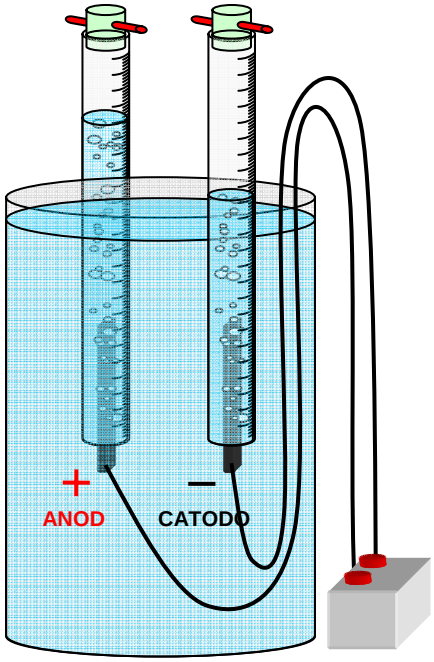


Figura 2 – Eletrólise da Água.
Fonte: Autor

Eletrólise Ígnea

Como visto na página 10, a solução eletrolítica ígnea é aquela onde os íons são liberados por aquecimento. Na eletrólise ígnea se faz o uso desta substância, que é um eletrólito fundido. O termo ígneo vem do latim lat. *Ignèus, a, um* 'ígneo, de fogo, inflamado, ardente'. Na eletrólise ígnea utilizam-se eletrodos inertes que possuam elevado ponto de fusão, comumente são feitos de platina ou grafita. Um exemplo de aplicação da eletrólise ígnea é a sua utilização no processo químico de obtenção do gás cloro[7] e do sódio metálico. Nesta aplicação se utiliza o cloreto de sódio fundido como eletrólito.

[7] **Cloro** – *sm (gr khlorós) Quím* Elemento não metálico, univalente e polivalente, de símbolo Cl, número atômico 17, que pertence aos halogênios. [6]

Eletrólise por Via Aquosa com eletrodos inertes

Na eletrólise por via aquosa com eletrodos inertes, os eletrólitos têm seus íons gerados ou dissociados pela interferência do caráter polar da água. Como a água também se ioniza, ocorre nos eletrodos uma competição na descarga dos elétrons entre os íons do eletrólito e os da água. [12]

Exemplo: Eletrólise do NaCl em solução aquosa. Nessa eletrólise, as equações químicas que ocorrem na reação são as seguinte:

$Na^+ + 1e^- \rightarrow Na$

$Na + H_2O \rightarrow NaOH + H_2$

$2Cl^- - 2e^- \rightarrow Cl_2$

Por meio dessa eletrólise são produzidos o hidrogênio e o cloro e, como subproduto, a soda cáustica (NaOH). Este é um importante processo industrial para a obtenção desses produtos.

Produção e utilização do hidrogênio

Vários estudos apontam para a seguinte conclusão: O gás hidrogênio será o combustível do futuro. Na atualidade, vários aspectos contribuem para esta conclusão; a escassez dos combustíveis fósseis diante da crescente demanda global de energia, a questão sobre a mudança climática e necessidade de diminuição da emissão de gases poluentes na atmosfera, as questões de saúde no que diz respeito à qualidade do ar, etc. Diante desses aspectos muitos estudos têm sido desenvolvidos no sentido de se propor soluções para as questões energéticas do século 21. Um exemplo é o projeto *HyWays* que coordena os principais estudos da economia do hidrogênio na Europa. O centro da questão é, quais combustíveis serão utilizados para substituir os já ultrapassados combustíveis fósseis como o petróleo e o carvão? Uma das promessas de combustíveis renováveis que está em destaque é o hidrogênio. A economia do hidrogênio é uma realidade cada vez mais próxima. Muitos governos e empresas já investem massivamente na tecnologia de produção e utilização de hidrogênio como energia. Exemplos disso são os grandes investimentos em células de combustíveis que geram energia elétrica a partir do gás hidrogênio.

No Brasil já existe uma fábrica de ônibus movidos a este combustível e já existem unidades desse ônibus em circulação na cidade de São Paulo. O Brasil é o quarto país no mundo a deter a tecnologia de fabricação de ônibus de transporte de passageiros movido a hidrogênio (os outros são os EUA, Alemanha e China). [10]

O esquema do uso do hidrogênio como uma fonte prática de energia é ilustrado na Figura 3. A produção de hidrogênio pela eletrólise da água com o uso de fontes energéticas renováveis, como a solar, eólica, biomassa ou hidrelétricas, permite que o hidrogênio seja armazenado, distribuído e convertido em uma fonte de energia limpa e muito útil.

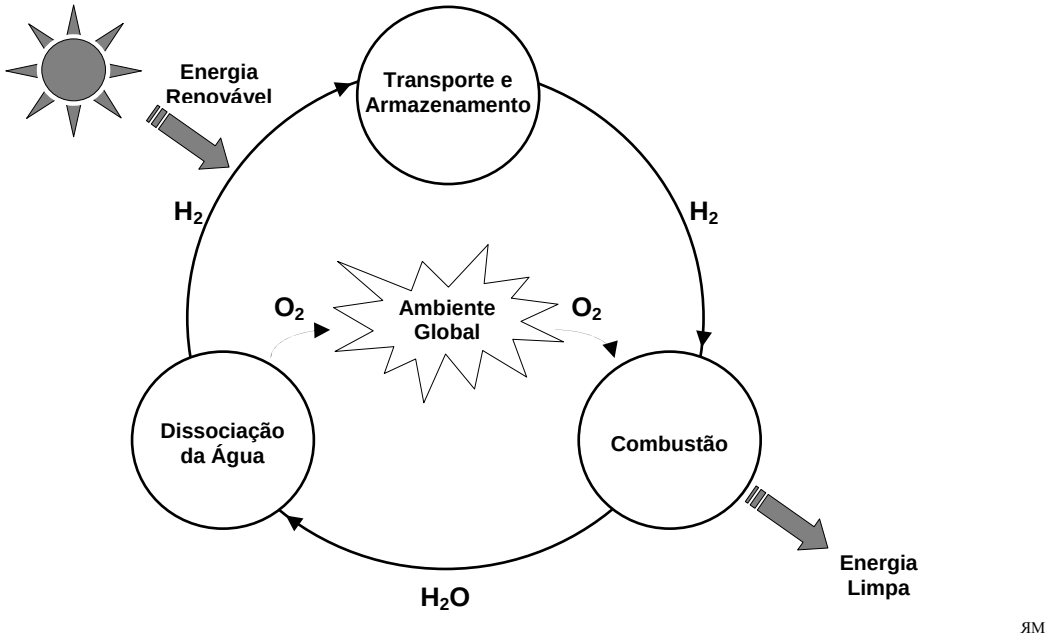


Figura 3 – O ciclo do hidrogênio

A produção de hidrogênio pode ser feita por métodos renováveis ou não renováveis. O uso de hidrogênio como combustível não garante a emissão zero de poluentes na atmosfera, portanto, não garante que seja uma fonte de energia totalmente renovável. Isto se deve ao fato de que, se a energia utilizada para a produção do hidrogênio for de natureza não renovável, o problema persiste. Um exemplo de produção do hidrogênio de forma não renovável é a utilização do método de reforma do gás natural. A equação para este processo é a seguinte:

$$CH_4 + 2H_2O = 4H_2 + CO_2$$

Como pode ser visto na equação, a reforma do gás natural produz hidrogênio e como subproduto obtêm-se o gás carbônico, que é um gás poluente. [14]

Felizmente existem meios de se produzir o hidrogênio de forma renovável. A produção desse gás pode ser feita utilizando um dos seguintes processos:

- *Eletrólise da Água* com uma solução alcalina (30% KOH) e eletrodos asbestos são amplamente utilizados em pequena e média escala (0.5-5.0 MW)

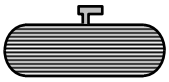
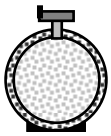
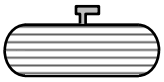
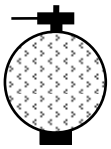
- *Eletrólise da água com eletrólito de Polímero sólido* (*Solid polymer electrolyte water electrolysis*) utiliza uma membrana de polímero sólido como eletrólito (SPE). Esta membrana quando saturada com água torna-se um excelente condutor (resistividade ≤ 15 ohms-cm) e é o único eletrólito requerido. Existem unidades comerciais de até 100KW. A eficiência desse processo fica entre 80 e 90%.

- *Eletrólise de vapor de alta temperatura* explora a diminuição acentuada na tensão de funcionamento de células acima de 700°C. Membranas de condução de íons de oxigênio, operando em 700-1000°C são utilizadas como eletrólito. A água a ser dissociadas entra no lado do cátodo em forma de vapor, levando a uma mistura de vapor de hidrogênio. Como a geração de calor é mais barata do que a geração de energia, os custos de produção com esta opção podem ser mais baixos.

Com o gás produzido é preciso haver mecanismos de armazenamento do gás para viabilizar seu transporte e utilização. A forma mais comum de se armazenar o gás hidrogênio é feita por meio de cilindros de gás de alta pressão com a pressão máxima de 20MPa. Novos cilindros têm sido desenvolvidos cuja pressão pode chegar até 80MPa e a densidade do hidrogênio armazenado pode alcançar 36 kg m^{-3}. As formas de armazenamento e suas características estão descritas na Tabela 1. [14]

Uma vez armazenado, o gás hidrogênio pode ser utilizado de maneira convencional, na forma de geração de energia pela sua combustão. Porém, a forma mais atrativa e que vem sendo massivamente desenvolvida é a utilização do gás hidrogênio para produzir energia elétrica através de dispositivos chamados de célula de combustível. A Figura 4 ilustra uma célula de combustível do tipo PEM (*proton exchange membrane*). [14]

Tabela 1 – Tipos de armazenamento do gás hidrogênio

Tipo de Armazenamento	Diagrama	Volume (massa)	Massa	Pressão	Temperatura
Cilindro composto (Padrão)		Max. 33 kg $H_2 \cdot m^{-3}$	13 %	800 bar	25°C
Hidrogênio Líquido		71 kg $H_2 \cdot m^{-3}$	100%	1 bar	-252°C
Hidretos Metálicos		Max. 150 kg $H_2 \cdot m^{-3}$	2%	1 bar	25°C
Absorção Física		20 kg $H_2 \cdot m^{-3}$	4%	70 bar	-208°C
Hidretos Complexos		150 kg $H_2 \cdot m^{-3}$	18%	1 bar	25°C
Base Alcalina + H_2O		> 100 kg $H_2 \cdot m^{-3}$	14%	1 bar	25°C

Fonte: ***Hydrogen as a Future Energy Carrier.*** **Desenho: Autor**

As células de combustível são utilizadas em diversas aplicações, desde a produção de energia elétrica para o uso industrial até sua utilização em veículos elétricos. Neste último caso, o veículo possui tanques de armazenamento do hidrogênio para sua utilização na geração da energia elétrica que será utilizada na propulsão do veículo, por meio de motores elétricos de alta potência.

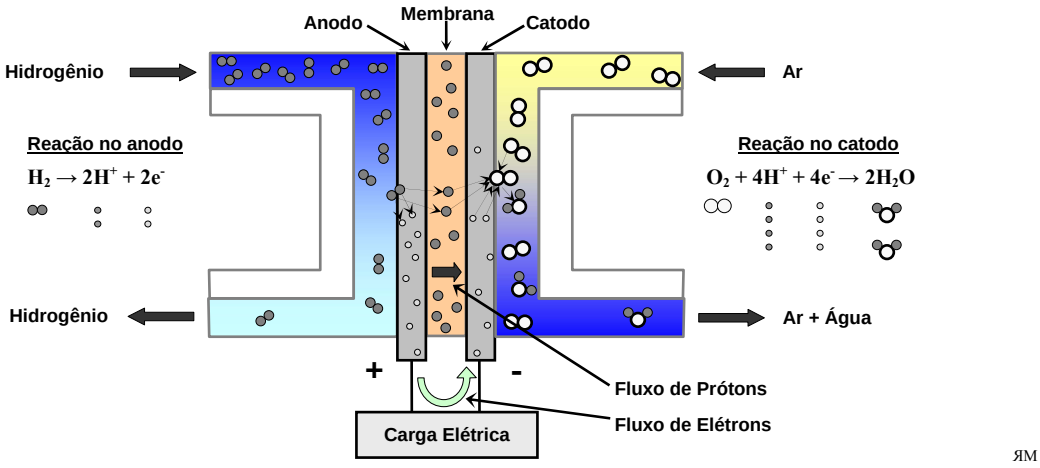


Figura 4 – Célula de combustível do tipo PEM
Fonte: *Hydrogen as a Future Energy Carrier.* Desenho: Autor

2.2. Circuitos Eletrônicos

2.2.1. Circuitos Eletrônicos de Controle

Um circuito eletrônico de controle é todo aquele circuito, que de forma dedicada, coordena a execução de uma determinada atividade. Esta atividade pode ser um controle interno do próprio circuito, como por exemplo, o controle da entrada e saída, bem como pode ser um controle de uma atividade externa ao circuito, como por exemplo, o controle de um motor, sua carga e temperatura. Desde o surgimento da eletrônica, circuitos de controle foram projetados para executarem as mais diversas atividades. Em princípio, com a invenção do transistor, esses circuitos eram fabricados com tecnologia de eletrônica analógica, isto é, circuitos analógicos transistorizados eram desenvolvidos para realizarem operações lógicas e de controle. Esses circuitos eram muito grandes e complexos e também bem eram muito limitados. A quantidade de componentes nesses circuitos era enorme, já que nesses circuitos não havia o emprego de circuitos integrados (CI).

Com o surgimento da eletrônica digital e da fabricação de circuitos integrados digitais, a quantidade de componentes para implementar um circuito de controle diminuiu significativamente. Com estes CIs, ficou bem mais fácil desenvolver circuitos lógicos e de controle, já que foram criados uma variedade de CIs que já implementavam funções lógicas, como as funções E, OU, OU Exclusivo, etc.. A tecnologia CMOS[8] e TTL[9] foram as principais tecnologias utilizadas na fabricação desses circuitos integrados. A família de CIs com o código 74XX ficou bastante conhecida e foram responsáveis pelo avanço da tecnologia de circuitos digitais. Exemplo de circuito integrado dessa família é o 7408 que implementa quatro portas digitais da função lógica E. A Figura 5 (a) ilustra a pinagem, o diagrama lógico e a tabela de função lógica desse CI.

[8] **CMOS** – (pronuncia-se "Cê-Mós") é uma sigla para *complementary metal-oxide-semiconductor*, i.e., semicondutor metal-óxido complementar. É um tipo de circuito integrado onde se incluem elementos de lógica digital (portas lógicas, *flip-flops*, contadores, decodificadores, etc.). A principal vantagem dos circuitos integrados CMOS é o baixíssimo consumo de energia, embora não sejam capazes de operar tão velozmente quanto circuitos integrados de outras tecnologias. [12]

[9] **TTL** – A Lógica Transistor-Transistor (*Transistor-Transistor Logic* ou simplesmente TTL) é uma classe de circuitos digitais construídos de transistores de junção bipolar (BJT), e resistores. Isso é chamado lógica transistor-transistor porque ocorrem ambas as funções porta lógica e de amplificação pelos transistores. Estes circuitos têm como principal característica a utilização de sinais de 5 volts para níveis lógicos altos. Seus circuitos integrados são constituídos basicamente de transistores, o que os torna pouco sensíveis à eletricidade estática. [12]

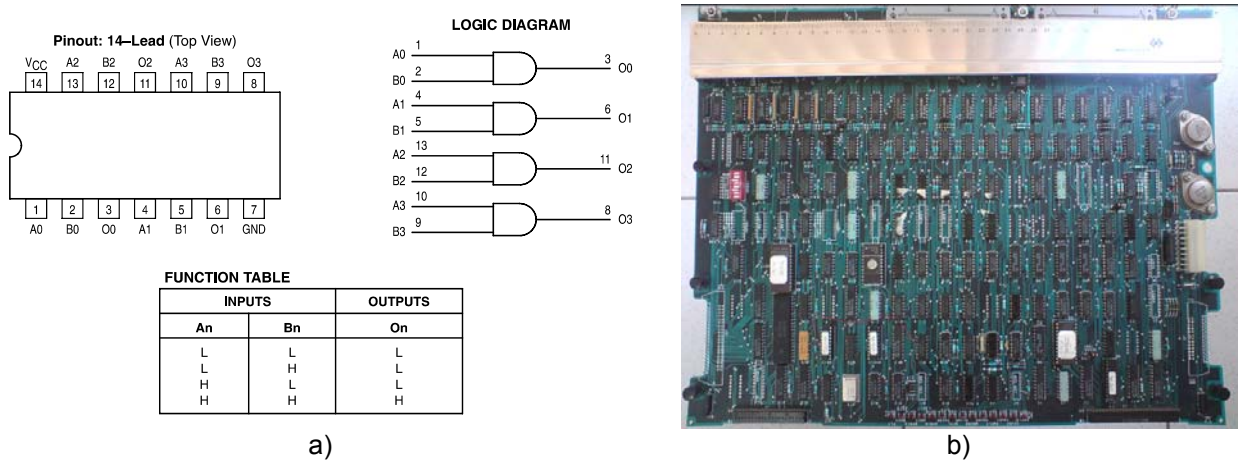


Figura 5 – Circuitos de Controle
a) Pinagem, diagrama e tabela de funções do CI 7408. b) Placa de controle de um mainframe da década de 80 do século XX. Fonte: Autor

Com o avanço da tecnologia eletrônica e o aumento da complexidade de circuitos digitais e circuitos de controle, o uso de CIs digitais que implementavam funções lógicas já não davam mais conta do recado. Eram necessários dezenas e até centenas desses CIs para que funções lógicas complexas pudessem ser implementadas. Exemplo disso era o uso desses CIs na fabricação de computadores *mainframe* dos anos 80 do século passado. Estes *mainframes* eram dotados de dezenas de placas eletrônicas imensas, cada uma com mais de 100 circuitos integrados. A Figura 5 (b) ilustra uma dessas placas.

Como pode ser visto na Figura 5 (b), a quantidade de circuitos integrados para executar tarefas mais complexas era muito grande. O avanço da tecnologia eletrônica permitiu que esse tipo de circuito fosse reduzido drasticamente o que acarretou também na redução do tamanho final dos equipamentos eletrônicos. Atualmente muitos circuitos de controle não contam mais com dezenas ou centenas de CIs. Com o surgimento de *chips* de alta densidade que podem armazenar milhares ou até milhões de transistores em uma única pastilha, foi possível integrar todo o circuito eletrônico como o da Figura 5 (b) em um único *chip*. Um tipo de circuito integrado que agrega internamente vários circuitos e que está sendo muito empregado no momento é o micro-controlador. Este será o assunto do próximo item.

2.2.2. Micro-controladores

Basicamente falando, o micro-controlador ou MCU, sigla em inglês de *Microcontroller Unit*, é sistema computacional completo dentro de um *chip*. Neste *chip* há o processador, a memória, as interfaces de entrada e saída (E/S) e todas as lógicas necessárias para o seu uso, tornando-o um sistema autossuficiente. Os micro-controladores são úteis em diversas aplicações da eletrônica. Na atualidade quase todo equipamento eletrônico possui um micro-controlador embutido. Com este tipo de circuito-integrado é possível reduzir consideravelmente o tamanho de circuitos eletrônicos que desempenham funções lógicas e que não possuam um micro-controlador. Blocos eletrônicos inteiros que antes eram implementados na placa eletrônica foram migrados para o interior do *chip* de um micro-controlador. Exemplo de blocos que são comumente encontrados em MCUs são: memórias FLASH[10] e EEPROM[11], sistemas de relógio, contadores lógicos, conversores de sinais (analógico/digital) (digital/analógico), interfaces de comunicação, entre vários outros.

Semelhante a um sistema computacional convencional, todo micro-controlador possui internamente um estrutura em blocos, onde cada bloco desempenha uma função específica. Alguns blocos são básicos, como o bloco regulador de voltagem e circuito de relógio. Outros blocos mais complexos que todo micro-controlador possui são a unidade central de processamento, o sistema de memória e o sistema de processamento de interrupções. Todos esses blocos se comunicam internamente através de barramentos de dados dedicados ou compartilhados. Além dos blocos indispensáveis mencionados acima, os micro-controladores também podem possuir blocos dedicados a executarem alguma tarefa dedicada como é o exemplo do bloco de entrada e saída E/S de propósito geral. Este bloco gerencia portas de entrada e saída, definindo, por exemplo, se um pino elétrico de determinada MCU deve ser de entrada ou de saída.

[10] **Memória Flash** – Um tipo de chip de memória que retém as informações quando a energia elétrica é interrompida. É uma memória não volátil. Pode ser re-gravada dezenas de milhares de vezes. [3]

[11] **Memória EEPROM** – Sigla de *Electrically-Erasable Programmable Read-Only Memory*. Similar a memória Flash, esta também é não volátil e pode ser apagada eletronicamente. A diferença entre esta e a FLASH é que o processo de escrita na EEPROM é feito bit a bit e na FLASH a escrita é feita bloco a bloco. [12]

MCU - Visão Geral

A maioria dos micro-controladores possui uma estrutura interna semelhante. Os blocos de regulação de voltagem, o sistema de relógio, o circuito de programação, sistema de interrupções, unidade de processamento, memória e portas de entrada e saída são encontrados em todos os micro-controladores. Além disso, alguns recursos adicionais como o Temporizador e o conversor A/D são bem comuns nessa família de circuitos integrados. A Figura 6, ilustra um diagrama em blocos básico de um micro-controlador genérico. [1]

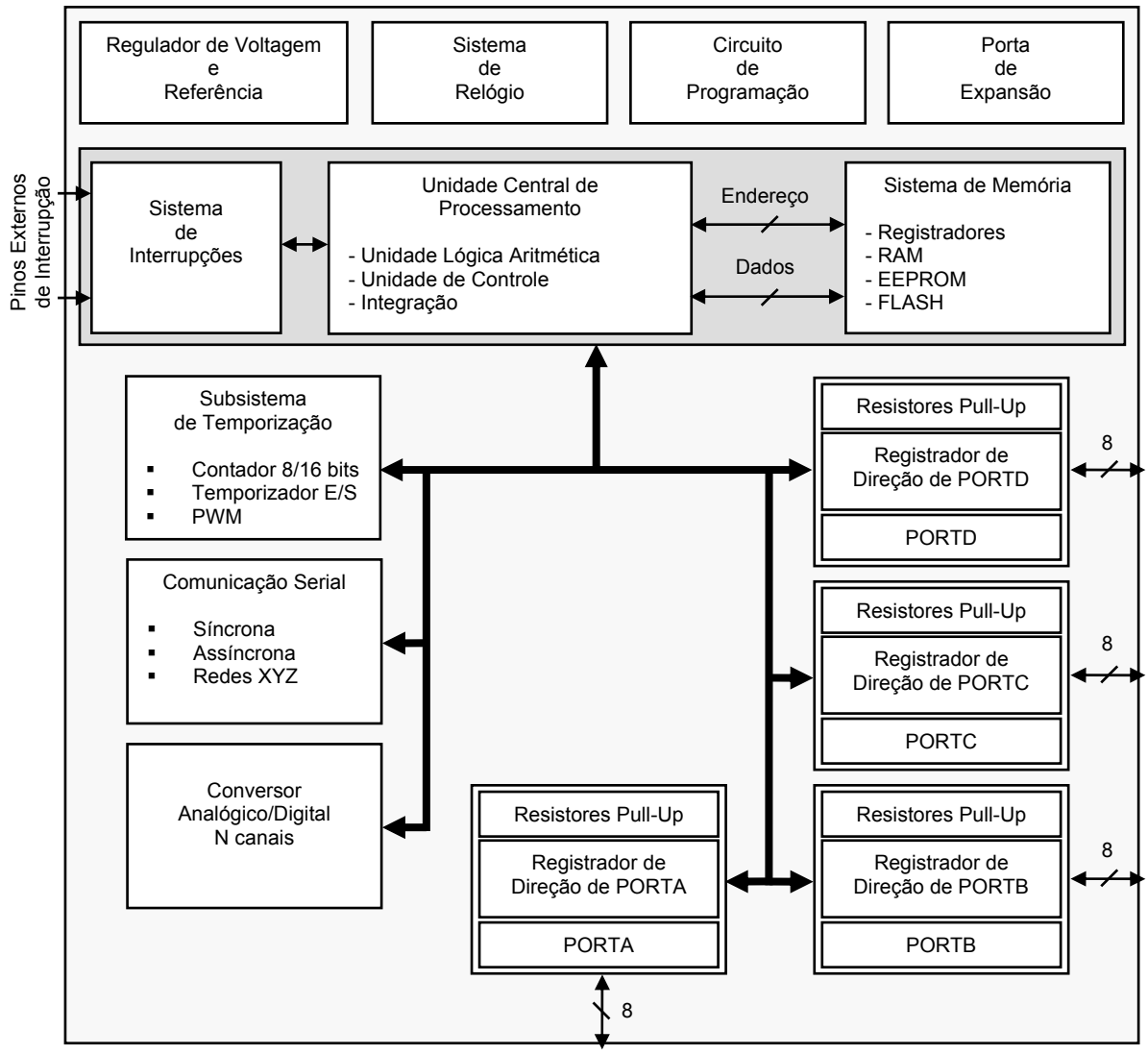
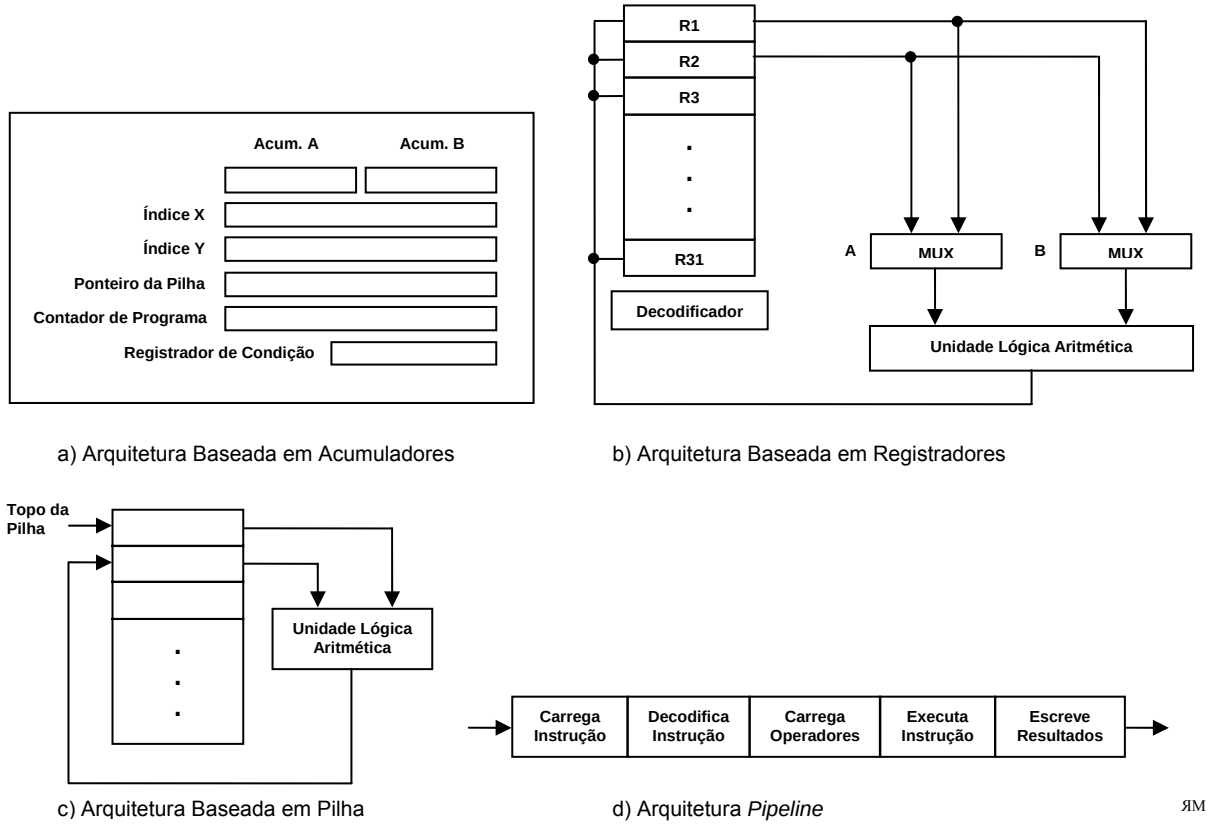


Figura 6 – Diagrama em blocos de um micro-controlador genérico
Fonte: Autor

MCU – Arquitetura Básica

A unidade central de processamento (UCP) de um micro-controlador é um circuito sequencial complexo cuja principal função é a de executar programas que estão armazenados na memória Flash. Um programa é uma série simples de instruções para executar uma tarefa específica. Programas são desenvolvidos por programadores de sistemas micro-controlados que utilizam ferramentas de desenvolvimento de programas. Usualmente, estas ferramentas são utilizadas por meio de um computador pessoal (PC). Uma vez que o programa tenha sido desenvolvido, o programa é transferido para o micro-controlador e este se torna um sistema de processamento autônomo. [1]

A unidade central de processamento é o centro de controle principal de todo o micro-controlador. Enquanto responde a diferentes instruções de programa, a UCP se responsabiliza também pela chamada de subsistemas residentes para executar suas tarefas. A arquitetura básica de uma UCP pode ser de diversos tipos conforme ilustrado na Figura 7. O que deve ser observado, é que uma dada arquitetura não é necessariamente melhor que a outra. Cada uma tem suas próprias vantagens e desvantagens. [1]



a) Arquitetura Baseada em Acumuladores

b) Arquitetura Baseada em Registradores

c) Arquitetura Baseada em Pilha

d) Arquitetura *Pipeline*

Figura 7 – UCP – Arquiteturas
Fonte: *Microcontrollers Fundamentals for Engineers and Scientists*. Desenho: Autor

2.3. O Micro-controlador PIC

2.3.1. Introdução

O item 2.2.2 abordou os conceito básicos sobre micro-controladores. Em resumo, um micro-controlador é um tipo de microprocessador equipado com circuitos periféricos, tudo em um único *chip*. A principal característica de um micro-controlador é a autossuficiência e o baixo custo. Um micro-controlador não foi planejado para ser utilizado como um sistema computacional de forma convencional, isto é, não foi designado para ser uma máquina de processamento de dados e sim para ser um núcleo de inteligência para sistemas dedicados. [11]

Muitas empresas no mundo fabricam e vendem micro-controladores. Estes dispositivos podem variar desde simples *chips* de 8 bits a até complexos micro-controladores de 64 bis. Uma dessas empresas, a Microchip, é uma fabricante estadunidense de circuitos integrados analógicos e digitais. Esta empresa é especializada também em oferecer diversas soluções de micro-controladores. PIC é o nome dado à família de micro-controladores da Microchip. Estes dispositivos foram chamados de PIC devido à expressão em inglês "*Programmable Intelligent Computer*" embora agora sejam associados com "*Programmable Interface Controller*". [11]

Os micro-controladores PIC são divididos em categorias de 8, 16 e 32 bits. Estes dispositivos são agrupados pelo tamanho de suas palavras de instrução. Atualmente existem três famílias de micro-controladores PIC conforme pode ser visto na Tabela 2.

Tabela 2 – Famílias de Micro-controladores PIC

Família	***Instruction Word***
Básico (Baseline PIC Family)	12-bit
Médio Desempenho *(Mid-range PIC Family*)	14-bit
Alto Desempenho (*High Performance PIC Family*)	16-bit

Os micro-controladores PIC possuem muitos recursos e características e falar sobre todos os aspectos desses dispositivos de maneira resumida demandaria um trabalho exclusivo para isso. Portanto, nesta monografia serão apresentados os conceitos básicos dos recursos do PIC utilizados no presente projeto. Este trabalho faz referência à família de médio desempenho.

2.3.2. PIC – *Mid-range Family*

A família PIC de médio desempenho é a família de micro-controladores mais populares e mais utilizados da Microchip. Esta família possui uma larga gama de dispositivos que vão de simples micro-controladores com menos de 1Kbytes de memória de programa até dispositivos mais complexos e completos com 28 Kbytes de memória. Estes micro-controladores são relativamente baratos e simples de se trabalhar.

Arquitetura do PIC

Os micro-controladores PIC possuem características análogas no que diz respeito a sua arquitetura. As qualidades dos micro-controladores PIC podem ser atribuídas às características arquiteturais encontradas em microprocessadores RISC[12] que são implementadas nestes dispositivos. Estas incluem:

- Arquitetura *Harvard*
- Instruções *Long Word*
- Instruções *Single Word*
- Instruções *Single Cycle*
- Tunelamento de Instruções *(Instruction Pipelining)*
- Conjunto de instruções reduzido
- Arquitetura de arquivamento por registradores
- Instruções Ortogonais (Simétricas)

Os micro-controladores PIC utilizam a arquitetura *Harvard*. Esta arquitetura computacional separa as memórias de programa e dados e utiliza-se de barramentos separados para o acesso a estas memórias. Esta técnica melhora a largura de banda em relação a tradicional arquitetura *von Neumann*[13] cuja memória de dados e de programa são dispostas juntas e utilizam o mesmo barramento de acesso. Para executar uma instrução, uma máquina *von Neumann* tem que acessar uma ou mais

[12] **RISC** – *Reduced Instruction Set Computer*. Um tipo de arquitetura de computador que possui poucas instruções e cada instrução executa operações mais elementares. Consequentemente reduz o tamanho da memória utilizada e aumenta a velocidade de execução. [11]

[13] **von Neumann** - A Arquitetura de von Neumann (do matemático John von Neumann), é uma arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital armazenar seus programas no mesmo espaço de memória que os dados, podendo assim manipular tais programas. [12]

vezes (geralmente mais vezes) o barramento de 8 bits para carregar a instrução. Ao mesmo tempo pode haver a necessidade de se carregar ou escrever dados na memória. Desta forma, o barramento pode ser extremamente congestionado. Com a arquitetura *Harvard*, a instrução é carregada em um simples ciclo (todos os 14 bits). Enquanto a memória de programa está sendo acessada, a memória de dados utiliza um barramento independente para leitura e escrita. Estes barramentos separados permitem que uma instrução seja executada ao mesmo tempo em que a próxima instrução esteja sendo carregada. A Figura 8 ilustra a comparação entre as arquiteturas *Harvard* e *von Neumann*. [8]

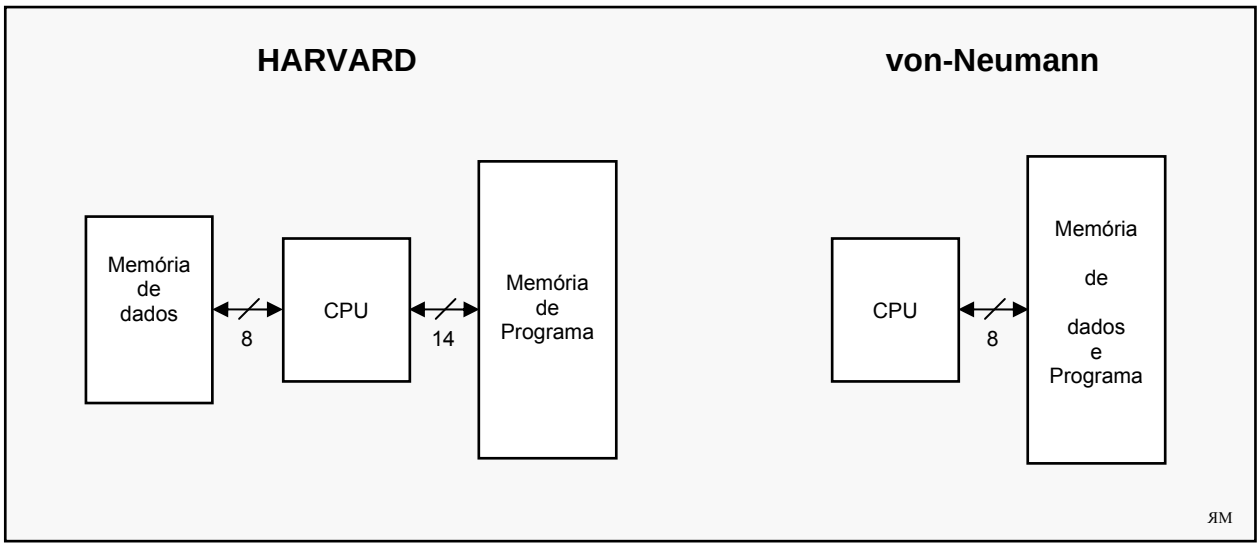


Figura 8 – Arquitetura Harvard versus von-Neumann
Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual*

A arquitetura da UCP dos micro-controladores PIC é do tipo *Pipeline* (Tunelamento de instruções). Esta arquitetura no PIC possui dois estágios na qual o carregamento e a execução das instruções são sobrepostos. Isto significa que uma instrução completa é carregada e outra executada em cada ciclo de máquina, por isso o nome tunelamento de instruções. Na verdade o primeiro ciclo de máquina no PIC apenas carrega uma instrução, porém devido ao tunelamento, a partir daí uma instrução é carregada e outra é executada em apenas um ciclo de máquina. A Figura 9 ilustra este mecanismo.

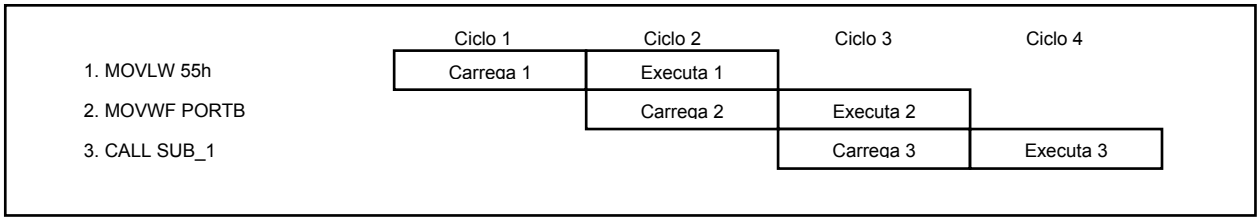


Figura 9 – Tunelamento de Instruções
Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual*

A arquitetura interna geral de todos os micro-controladores PIC da família de médio desempenho é praticamente a mesma. A diferença está na quantidade de portas de entrada e saída e nos módulos periféricos. A Figura 10 ilustra a arquitetura geral da família PIC de médio desempenho. [8]

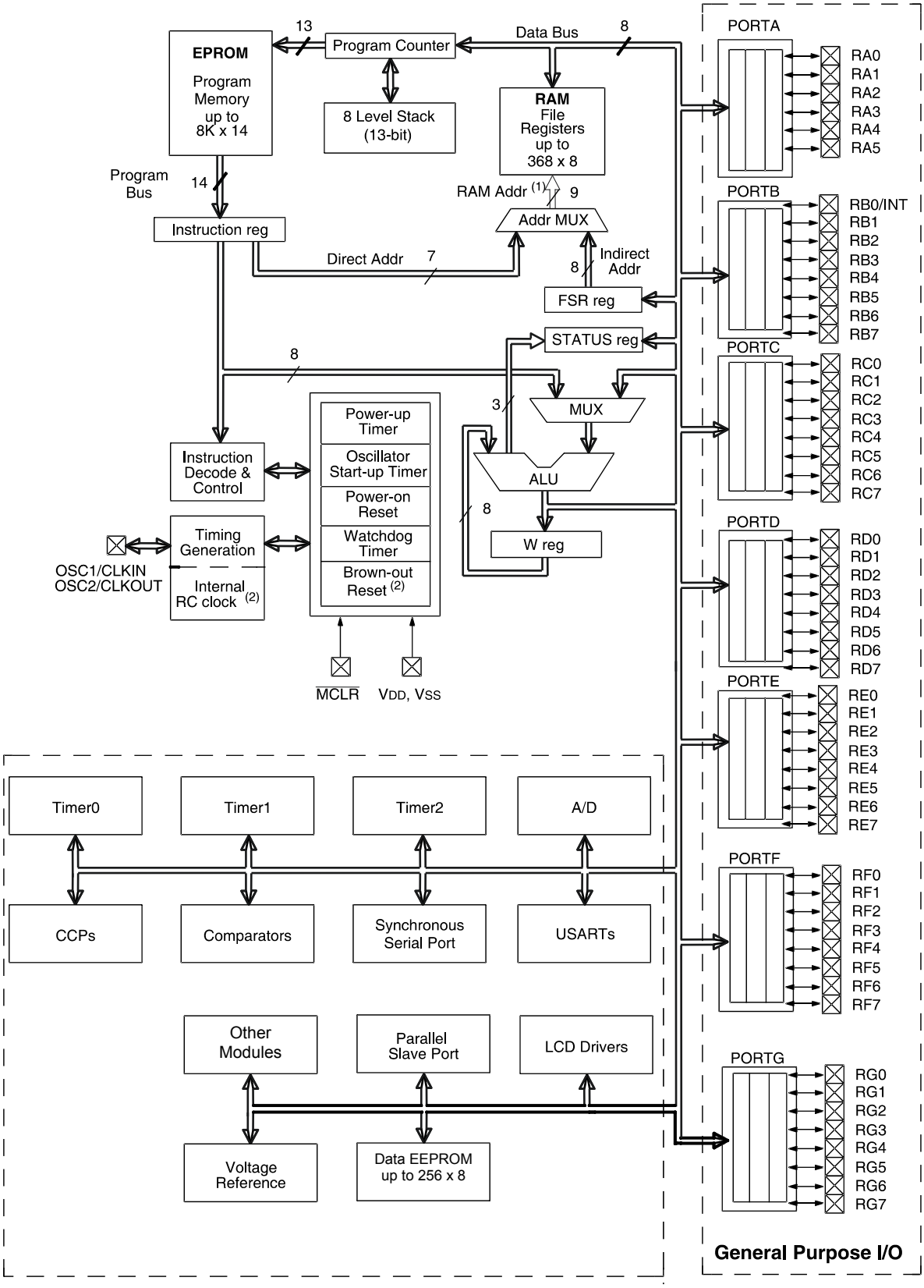


Figura 10 - Arquitetura geral da família PIC de médio desempenho
Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual*

2.3.3. PIC – Memória de Programa e de Dados

Os micro-controladores PIC possuem dois blocos internos de memória; a memória de programa e a memória de dados. Conforme explicado no item 2.3.2, o PIC utiliza a arquitetura *Harvard*, o que significa que cada um desses blocos de memória possui o seu próprio barramento. Desta forma, o acesso a cada bloco pode ser feito durante o mesmo ciclo de máquina. Além disso, a memória de dados é dividida em duas partes; a memória RAM de propósitos gerais (*GPR – General Purpose Registers*) e os registradores de funções especiais (*SFR – Special Function Registers*). Estes registradores especiais são utilizados para controlar o núcleo do micro-controlador, bem como para controlar os módulos periféricos.

Memória de Programa

Os micro-controladores PIC da família de médio desempenho possuem um contador de programa de 13 bits capaz de endereçar um espaço de memória de programa de 8K x 14. Uma vez que toda instrução do PIC (*Mid-Range*) possui o tamanho da palavra de 14 bits, um dispositivo com 8K x 14 de memória de programa tem espaço para 8K de instruções. Este espaço de programa é dividido em quatro paginas de 2K cada (0h – 7ffh, 800h - FFFh, 1000h - 17FFh, e 1800h - 1FFFh). A Figura 11 mostra o mapa da memória de programa, bem como os 8 níveis da pilha. Dependendo do dispositivo, apenas uma parte da memória é implementada. [8]

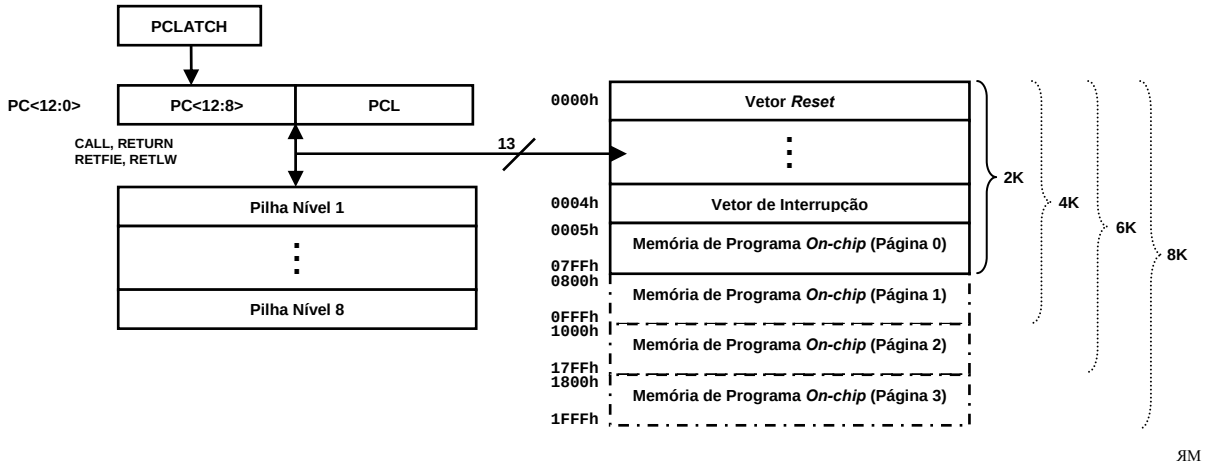


Figura 11 – Arquitetura da Memória de Programa
Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual.* Desenho: Autor

Memória de Dados

A memória de dados dos micro-controladores PIC da família de médio desempenho possuem uma área de registradores de funções especiais (SFRs) e uma área de registradores de propósitos gerais (GPR). Os registradores SFRs controlam as operações do dispositivo, já os GPRs são uma área geral de memória para o armazenamento de dados e operações temporárias. [8]

Registradores de Propósitos Gerais (GPR)

Alguns micro-controladores da família de médio desempenho possuem a memória GPR dividida em bancos. Os GPRs não são inicializados quando o dispositivo é ligado em não são modificados em operações de reinicialização (*reset*). Esta área de memória é referenciada como uma área RAM comum.

Registradores de Funções Especiais (SFR)

Os registrados de funções especiais (SFRs) são utilizados pela UCP e módulos periféricos para o controle de determinadas operações do dispositivo. Estes registradores são implementados como memórias RAM estáticas. Os SFRs são classificados em dois conjuntos, um associado com as funções do núcleo do micro-controlador e outro relacionado as funções dos módulos periféricos.

2.3.4. PIC – Recursos Básicos

Mecanismo de Interrupções

Toda unidade central de processamento de dados (UCP) executa suas instruções de maneira ordenada. Como pôde ser visto na Figura 9, a cada ciclo de máquina uma instrução é carregada e outra é executada pela UCP. Esta seqüência de execução das instruções é ordenada e ocorre sequencialmente do início ao fim do programa instalado na memória de programas. Alguns recursos do PIC podem interromper esta execução e até mesmo modificar a sua ordem. Neste trabalho não serão abordadas todas as formas em que recursos do PIC interrompem a seqüência de execução das instruções,

apenas será tratado o mecanismo de interrupções devido a sua utilização neste projeto.

Um mecanismo de interrupções permite que algum evento externo ou interno ao micro-controlador possa interromper a execução do programa instalado. É claro que esta permissão é dada caso o seu uso seja necessário. Este mecanismo é muito útil porque permite que eventos pré-programados tenham prioridade em sua execução, como por exemplo, o aperto de um botão para acionar uma determinada função. A ocorrência desse evento pré-programado faz com que a execução do programa principal seja interrompida para que o tratamento do evento seja feito o mais rápido possível, isto é, tenha prioridade em sua execução.

Para que este mecanismo funcione conforme o esperado é necessário que o programador defina o modo como essa interrupção deverá ser tratada. Voltando ao exemplo do botão, é necessário programar um algoritmo que tratará o evento do aperto do botão. Este algoritmo poderá ser, por exemplo, o acionamento de uma lâmpada. Se um programa está rodando no micro-controlador estiver executando alguma tarefa, ao se acionar o botão, este programa será interrompido e o mecanismo de interrupções irá carregar o algoritmo de tratamento desse evento que pode ser, por exemplo, ativar um pino com o nível lógico alto para que uma lâmpada seja acesa. Ao terminar a execução desse algoritmo de tratamento do evento, o mecanismo de interrupção retorna a execução do programa que estava sendo executado.

O mecanismo de interrupção dos micro-controladores da família de médio desempenho funcionam de maneira muito semelhante em todos os dispositivos. Esses micro-controladores possuem diversas fontes de interrupção. Geralmente cada módulo periférico do PIC possui uma fonte de interrupção, embora alguns módulos possam gerar múltiplas interrupções. A lista abaixo mostra alguns tipos de interrupções:

- Interrupção externa INT
- Interrupção de temporizadores (*timer0, timer1*)
- Interrupções dos módulos de comunicação (USART, SSP, I^2C)
- Interrupção de conversores A/D
- Interrupção de comparadores

No mecanismo de interrupções do PIC existe ao menos um registrador para controlar as interrupções. Este registrador é o: INTCON. Adicionalmente, se o

dispositivo possuir interrupções de módulos periféricos, então haverá uma série de registradores utilizados para habilitar estas interrupções e registradores para armazenar os bits de estado dessas interrupções (*interrupt flag bits*). Dependendo do dispositivo, estes registradores são: PIE1, PIR1, PIE2, PIR2. [8]

O registrador de controle de interrupções, INTCON, armazena os bits individuais de estado das requisições de interrupções. Este registrador também possui os bits individuais de habilitação das interrupções e o bit de habilitação global das interrupções. A Figura 12 ilustra a estrutura básica do mecanismo de interrupções dos micro-controladores PIC. Como pode ser visto, primeiramente as instruções são controladas pelo registrador **INTCON**. Este registrador pode habilitar ou desabilitar todas as interrupções por meio do pino GIE. O pino PEIE pode habilitar ou desabilitar todas as interrupções dos módulos periféricos. [8]

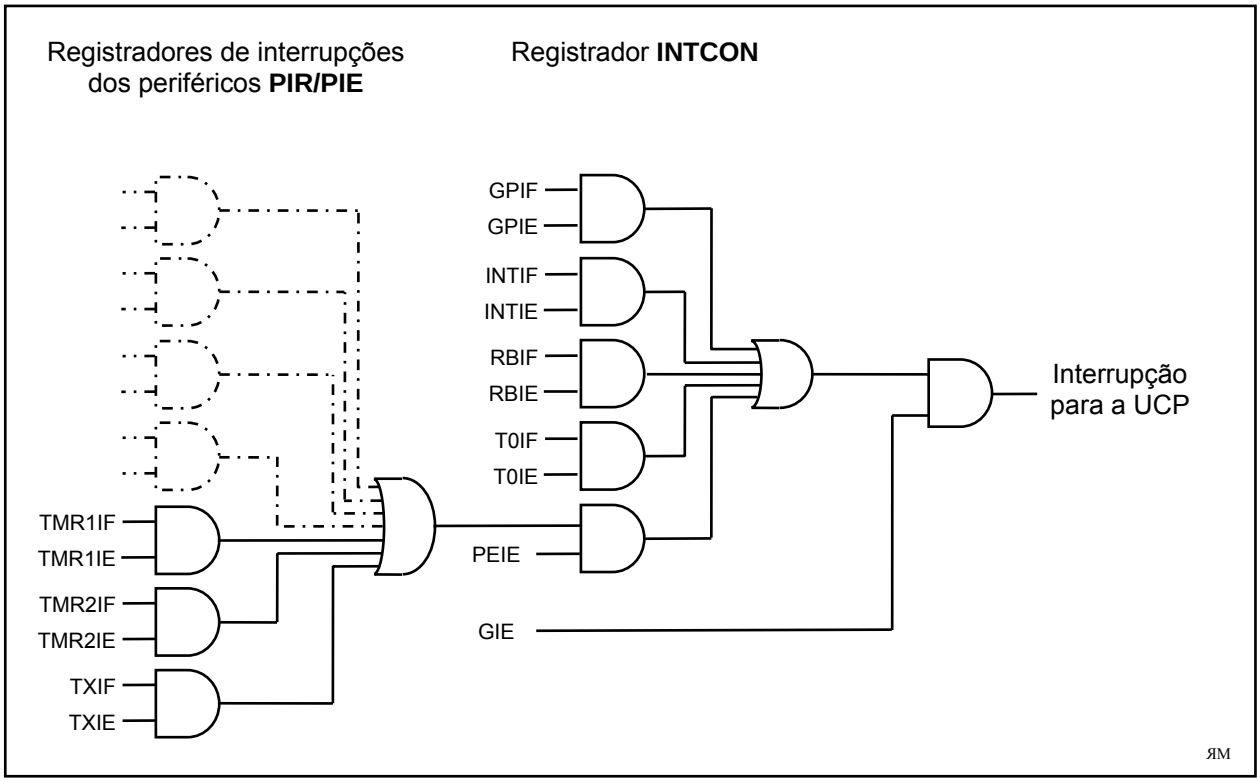


Figura 12 – Estrutura básica do mecanismo de interrupções do PIC
Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual.* Desenho: Autor

Para se utilizar uma interrupção, o programador do dispositivo deve criar um algoritmo de tratamento para a determinada interrupção. Toda vez que uma interrupção válida ocorrer o mecanismo de interrupção interrompe o programa principal, direciona a execução da UCP para o dado algoritmo e quando este tiver sido finalizado, a UCP volta a executar o programa principal de onde havia parado.

Temporizador *timer1*

Quase todos os micro-controladores da família PIC de médio desempenho possuem pelo menos um temporizador. O temporizador básico dos micro-controladores PIC chama-se *timer0*. O *timer0* pode funcionar tanto como temporizador como contador de pulsos externos ao PIC. Este é um temporizador/contador de 8 bits. O temporizador descrito nesta seção será o *timer1*, um temporizador/contador também, porém com 16 bits. O *timer1* possui dois registradores de 8 bits (TMR1H e TMR1L). Operações de escrita e leitura podem ser feitas nestes registradores. O par de registradores TMR1 (TMRH:TMRL) pode ser incrementado de 0000h a FFFFh e tornar a 0000h. /a interrupção do *timer1*, se habilitada, é gerada toda vez que o contador estourar (*overflow*), isto é, toda vez que o contador girar de FFFFh para 0000h. Esta interrupção pode ser habilitada ou desabilitada pela configuração do bit TMR1IE. [8]

O *timer1* pode operar em um dos três modos:

- Como um temporizador síncrono
- Como um contador síncrono
- Como um contador assíncrono

O modo de operação é determinado pelo bit de seleção do relógio (*clock select*), TMR1CS (T1CON<1>), pelo bit de sincronização $\overline{\text{T1SYNC}}$. No modo temporizador, o *timer1* é incrementado a cada ciclo de instrução. Em modo contador, o incremento ocorre a cada borda de subida na entrada do *clock* externo (pino T1CKI). [8]

O temporizador *timer1* pode ser ativado ou desativado por meio do bit de controle TMR1ON (T1CON<0>). A Figura 13 ilustra o diagrama em blocos do *timer1*. [8]

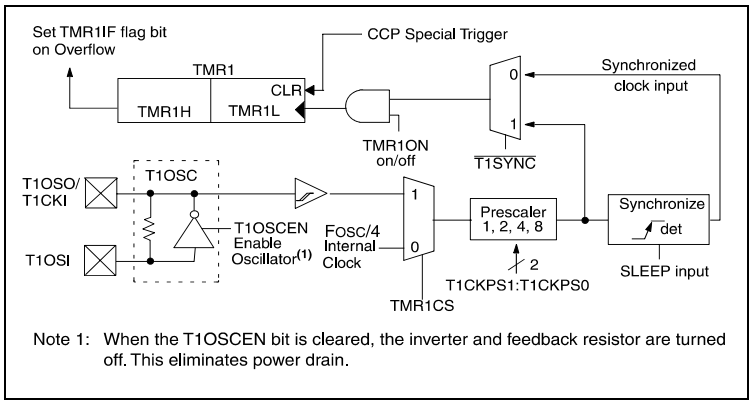


Figura 13 – Diagrama em blocos do temporizador *timer1*.
Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual*

2.3.5. PIC – Módulo de Comunicação Serial SSP

O módulo SSP (*Sinchronous Serial Port*) é uma interface serial muito útil para comunicação com outros dispositivos periféricos ou micro-controladores. Estes dispositivos periféricos podem ser memórias EEPROMs, registradores de deslocamento, *drivers* de displays, conversores A/D, etc. O módulo SSP pode operar em um dos dois modos:

- SPI – Interface Serial Periférica (*Serial Peripheral Interface*)
- I^2C – *Inter-Integrated Circuit*
  - Modo escravo
  - Modo mestre ou multi-mestre

Barramento de dados I^2C

I²C é um barramento em série. Este nome é a sigla de *Inter - Intergrated Circuit*. A versão 1.0 data de 1992 e a versão 2.1 do ano 2000. Esta tecnologia foi desenvolvida pela Philips. A velocidade normal de operação é de 100Kb/s sendo que é possível atingir até 3.4 Mb/s. É um barramento muito utilizado na indústria, principalmente para comunicar micro - controladores e seus periféricos em sistemas embarcados. Pode também ser utilizado para comunicar circuitos integrados entre si que normalmente se encontram em um mesmo circuito. [5]

I^2C é um barramento serial baseado em uma relação mestre-escravo entre os nós. O mestre controla toda a utilização do barramento. O I^2C utiliza apenas duas linhas para a interconexão, a linha de dados SDA (*serial data*) e a linha de relógio SCL (s*erial clock*). As duas linhas de interconexão possuem cada uma um resistor *pull-up*[14]. Ambas as linhas são bi-direcionais, mas o sinal de relógio SCL é sempre gerado pelo mestre corrente. A Figura 14 ilustra a estrutura básica de uma interconexão I^2C. [13]

[14] **Resistores pull-up** – São resistores usados no projeto de circuitos lógicos eletrônicos para garantir que entradas para sistemas lógicos se ajustem em níveis lógicos esperados se dispositivos externos são desconectados. Eles também podem ser usados na interface entre dois diferentes tipos de dispositivos lógicos, possivelmente operando em tensões diferentes. [12]

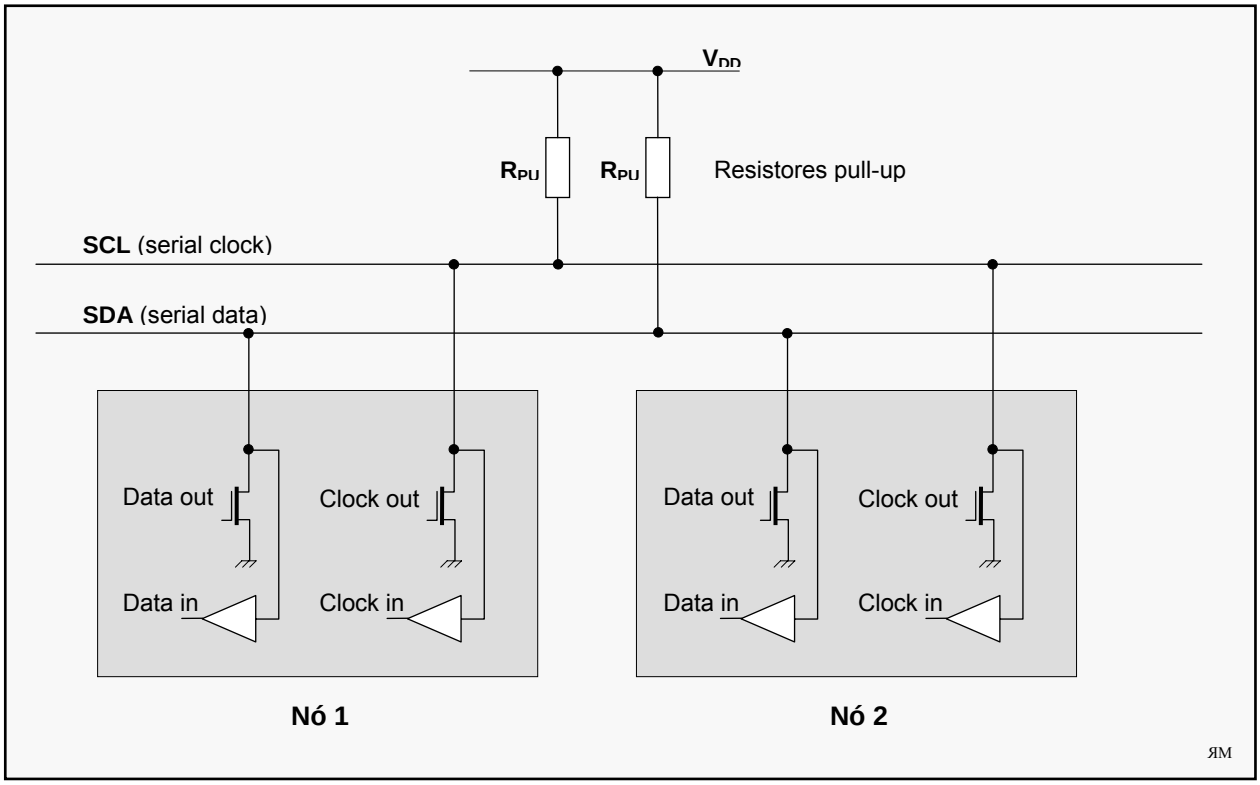


Figura 14 – Estrutura básica de uma interconexão I^2C
Fonte: ***Designing embedded systems with PIC microcontrollers.*** **Desenho: Autor**

Módulo I^2C no micro-controlador PIC

O micro-controlador PIC possuí um periférico *on-chip* chamado MSSP. A sigla advém de *Master Synchronous Serial Port*. Este módulo implementa a interface serial SSP. Por meio desse módulo é possível criar uma rede de comunicação I^2C para interconexão entre dispositivos. O módulo MSSP em modo I^2C implementa todas as funções para operar o micro-controlador como mestre ou escravo (Incluindo suporte a chamadas comuns). Este módulo também fornece interrupções em hardware para os bits START e STOP. O módulo MSSP implementa as especificações padrões, bem como o endereçamento de 7 e 10 bits.

I2C - Modo Mestre

A operação em modo mestre é suportada pela geração de interrupção na detecção das condições de START e STOP. No modo mestre, as linhas SCL e DAS são manipuladas pelo hardware MSSP.

Os seguintes eventos colocarão o bit SSPIF em estado alto:

- Condição de START
- Condição de STOP
- Transferência de dados (*transmitted/recevied*)
- Transmissão do ACK
- Repetição da condição de START

Os diagramas em blocos do módulo I^2C estão ilustrados no Apêndice D. A Tabela 3 ilustra todos os registradores envolvidos nas operações deste módulo. Esta tabela foi extraída do manual de referência dos micro-controladores PIC da família de médio desempenho. [8]

Tabela 3 – Registradores utilizados pelo módulo I^2C

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on: MCLR, WDT
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Dh	PIR2	—	(2)	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	-r-0 0--0
8Dh	PIE2	—	(2)	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0	-r-0 0--0
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	0000 0000
93h	SSPADD	I^2C Slave Address/Master Baud Rate Register								0000 0000	0000 0000
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the SSP in I^2C mode.
Note 1: These bits are reserved on PIC16F873/876 devices; always maintain these bits clear.
2: These bits are reserved on these devices; always maintain these bits clear.

Fonte: MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual*

Capítulo 3. Resultados

A execução desse trabalho gerou os seguintes resultados práticos: o protótipo do eletrolisador micro-controlado da água, que engloba os circuitos eletrônicos de controle, geração de sinais, pré-amplificação e *driver* de potência, bem como o *software* computacional da interface de controle. Além disso, o protótipo também conta com o artefato físico construído para a execução da eletrólise da água. O projeto do protótipo será o assunto do próximo item.

Com o protótipo fabricado, foram feitos também os experimentos de análise do comportamento da eletrólise quando da aplicação de determinados pulsos elétricos nos eletrodos do eletrolisador. Estes experimentos foram feitos com três tipos de solução aquosa e serão descritos no item 3.2.

3.1. Protótipo do Eletrolisador Micro-controlado da Água

O projeto Eletrolisador Micro-controlado da água foi dividido em duas partes. A primeira parte trata-se do equipamento eletrônico. Este equipamento foi concebido para executar as funções de controle e geração de sinais elétricos responsáveis pela geração da eletrólise da água. A segunda parte é o artefato físico onde ocorre a eletrólise.

A principal função do Eletrolisador é produzir o gás hidrogênio e oxigênio através da eletrólise da água. Neste método é necessária a utilização de um artefato físico onde ocorrerá o fenômeno da eletrólise. O artefato utilizado no projeto é composto basicamente por um recipiente de vidro para o armazenamento da água, quatro pares de eletrodos metálicos em forma de tubos e os acessórios necessários para a fixação dos eletrodos, a vedação do recipiente, os cabos e as conexões elétricas dos eletrodos, as mangueiras de transmissão dos gases, entre outros. Este artefato é descrito por completo no item 3.1.5.

Na eletrólise convencional da água, um potencial elétrico fixo é aplicado em eletrodos mergulhados em uma solução aquosa. A quebra da molécula da água ocorre com mais ou menos intensidade dependendo da condutibilidade da água e do nível do potencial elétrico aplicado. O Eletrolisador aqui descrito é capaz de gerar sinais

elétricos com formas de onda diferenciadas. A forma como a eletrólise se comporta com a aplicação desses sinais diferenciados será o objeto de estudo deste trabalho.

Para produzir os sinais elétricos diferenciados foi desenvolvido um sistema eletrônico capaz de gerar pulsos elétricos de forma controlada. O sistema foi desenvolvido para gerar uma seqüência de pulsos elétricos em períodos definidos. Esses pulsos elétricos são amplificados e aplicados aos eletrodos do Eletrolisador. A forma como estes pulsos são gerados será descrita no Módulo Gerador de Sinais, item 3.1.2.

No sistema eletrônico, o controle e a geração dos sinais elétricos são definidos através de uma interface de controle. Nesta interface o usuário do sistema poderá configurar os parâmetros de controle para a geração dos sinais elétricos do Eletrolisador. Uma vez definido os parâmetros na interface de controle, pode-se iniciar a produção do gás através da aplicação dos sinais elétricos nos eletrodos do eletrolisador.

A visão geral do protótipo do eletrolisador é ilustrada na Figura 15.

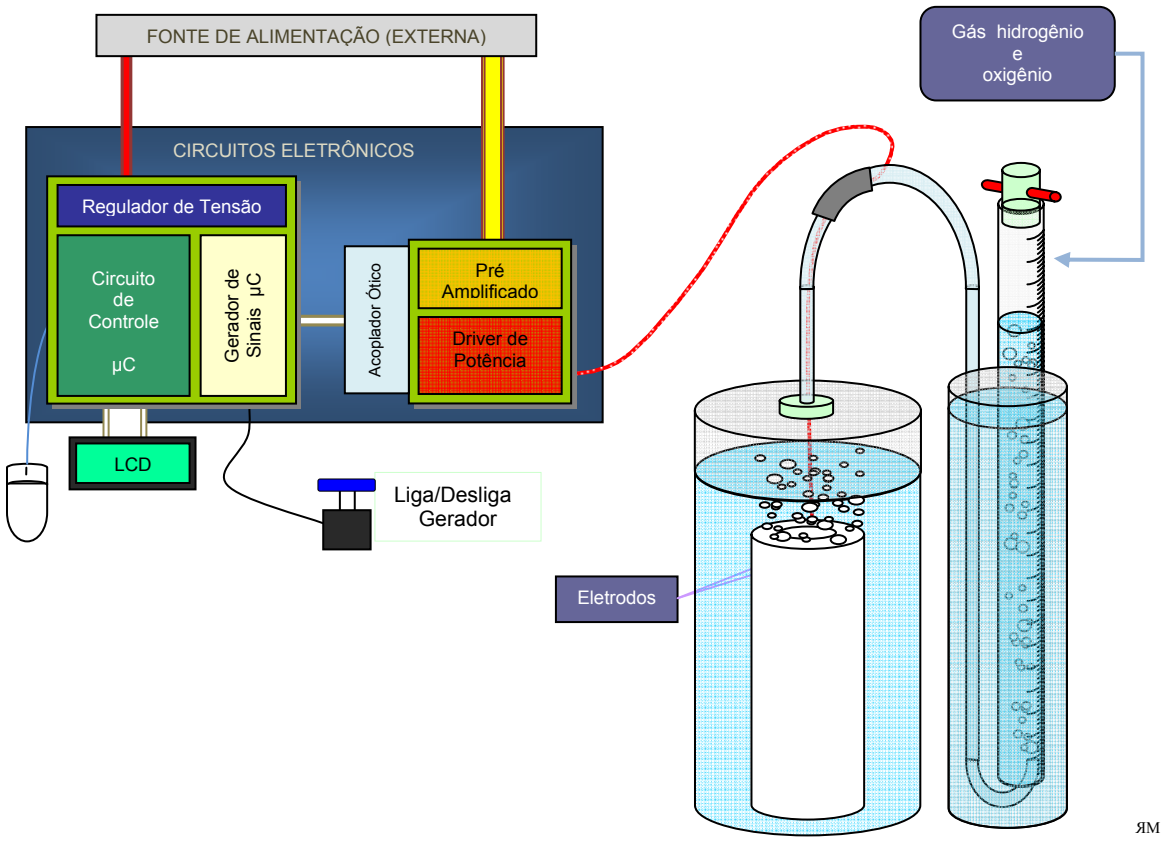


Figura 15 – Visão Geral do Eletrolisador Micro-controlado da Água.
Fonte: Autor

3.1.1. Interface de Controle

Todo sistema necessita de uma interface de interação entre o usuário e o sistema. No sistema do eletrolisador essa interação é feita através de um programa de computador desenvolvido em linguagem C embarcado em um circuito eletrônico micro-controlado. A interação entre o usuário e o programa embarcado é feita através de um *mouse* adaptado e um botão liga/desliga como entrada. A interface de saída do programa é feita através de três *leds* de sinalização e um *display* LCD. O programa embarcado é descrito em detalhes no item 3.1.3.

Funções da Interface de Controle

Para controlar a produção de gás no eletrolisador, o usuário deve operar o sistema por meio da interface de controle. Nesta interface, os parâmetros de configuração do gerador de sinais do eletrolisador podem ser alterados através do programa de computador embarcado no sistema. Para isso, o usuário contará com um conjunto de funções disponíveis no sistema.

As funções disponíveis na Interface de Controle são as seguintes:

- Definição do relógio do sistema
- Definição da quantidade de pulsos da seqüência de pulsos
- Configuração do período do pulso ligado
- Configuração do período do pulso desligado

Conforme é explicado no item 3.1.2, existem dois módulos eletrônicos micro-controlados. No primeiro módulo, o de controle, está instalado o programa da Interface de controle. Este programa executa as funções da interface de controle. A estrutura dos menus de funções do programa está ilustrada na Figura 16.

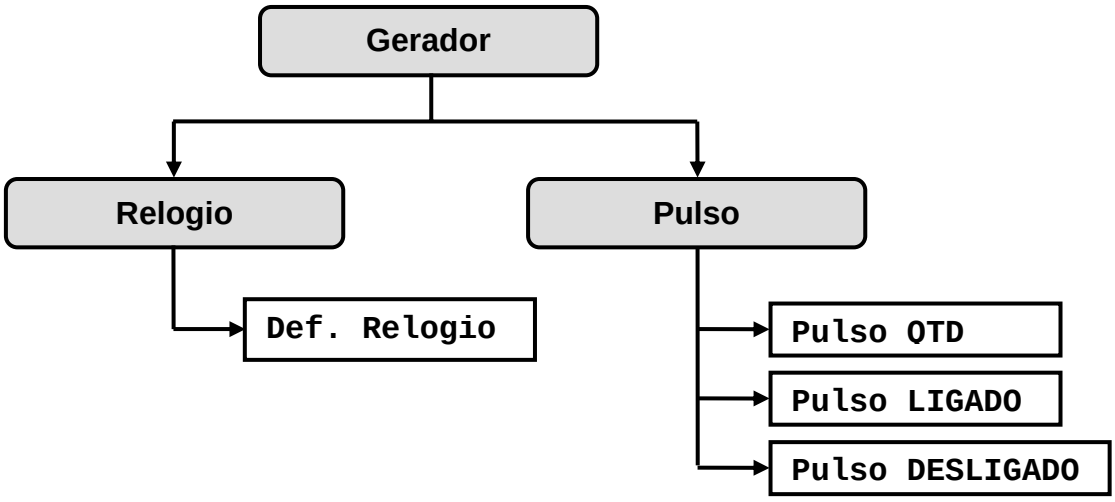


Figura 16 - Estrutura dos menus de funções da Interface de Controle

Função de definição do relógio (Def. Relogio)

O relógio do sistema define o período que o sinal permanecerá em nível baixo entre o fim de uma seqüência de pulsos e o início da seqüência subsequente. Este período é definido em micro segundos e por questões de calibração pode variar entre 133 µs e 11122 µs.

Função de definição da quantidade de pulsos (Pulso QTD)

Nesta função o usuário define a quantidade de pulsos que serão gerados a cada sequência de pulsos. Essa quantidade pode variar entre 1 e 65355.

Função de definição da largura do pulso ligado (Pulso LIGADO)

A definição da largura do pulso em nível alto é dada pela função Pulso LIGADO. Nesta função, o usuário pode definir o tempo em que cada pulso da seqüência de pulsos irá permanecer no nível alto.

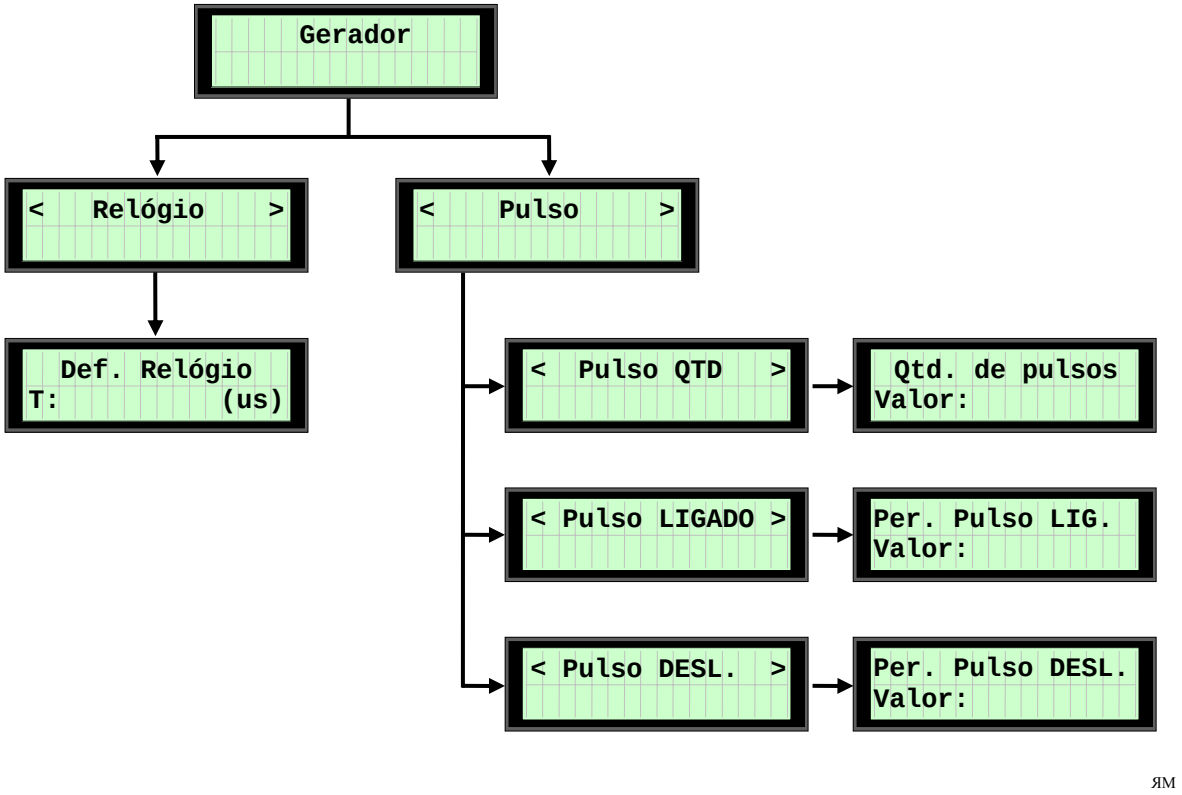
Função de definição da largura do pulso desligado (Pulso DESLIGADO)

A definição da largura do pulso em nível baixo é dada pela função Pulso DSLIGADO. Nesta função, o usuário pode definir o tempo em que cada pulso da seqüência de pulsos irá permanecer no nível baixo.

Definição das Telas da Interface de Controle

O programa embarcado foi desenvolvido para contemplar as funções da interface de controle apresentadas anteriormente. Para contemplar essas funções foi definido o formato e as expressões utilizadas nas telas de interação com o usuário. A definição das telas foi feita com base na estrutura já apresentada na Figura 16.

As telas do programa são apresentadas na Figura 17.



ЯМ

Figura 17 – Telas da Interface de Controle
Fonte: Autor

Modo de Navegação da Interface de Controle

A navegação nos menus de funções da Interface de Controle é feita por meio de um *mouse* de três botões com a função *roller* no botão central. O botão esquerdo é utilizado para entrar em um menu selecionado ou para definir um parâmetro em uma função. O botão direito serve para alternar entre os menus. O botão central é utilizado para retornar ao menu anterior. O *roller* é utilizado para definir os valores a serem configurados nas funções.

3.1.2. Módulos Eletrônicos

Os módulos eletrônicos do projeto são responsáveis pelo controle e geração dos pulsos elétricos que serão utilizados para a produção do gás. Observando a Figura 15, no painel dos circuitos eletrônicos, é possível identificar dois blocos distintos. No primeiro bloco, à esquerda, encontram-se três módulos eletrônicos distintos, o regulador de tensão, o circuito de controle e o circuito gerador de sinais. O circuito de controle e o circuito gerador de sinais são micro-controlados.

O segundo bloco, localizado à direita do painel dos circuitos eletrônicos, ilustrado na Figura 15, contém os módulos de pré-amplificação e *driver* de potência. Além disso, há nesse bloco um circuito de acoplamento ótico para isolar o primeiro bloco, o de controle e geração de sinais, do segundo bloco. Este isolamento é necessário devido ao segundo bloco trabalhar com altos níveis de tensão e corrente. A isolação evita que qualquer surto de corrente ou de tensão inesperados, possa danificar os circuitos de controle e geração de sinais.

Bloco de Controle e Geração de Sinais

Conforme descrito anteriormente, o sistema eletrônico do Eletrolisador Micro-controlado da Água foi dividido em dois blocos e os módulos eletrônicos de controle e geração de sinais se encontram no primeiro bloco. Este bloco pode ser caracterizado com sendo um bloco de eletrônica digital, isto é, trabalha com níveis de tensão que representam os estados lógicos zero e um. Neste bloco, os níveis de tensão e corrente são baixos e a potência consumida nos circuitos de controle e geração de sinais são ínfimas em comparação com o segundo bloco do sistema.

Este bloco é ilustrado na Figura 18.

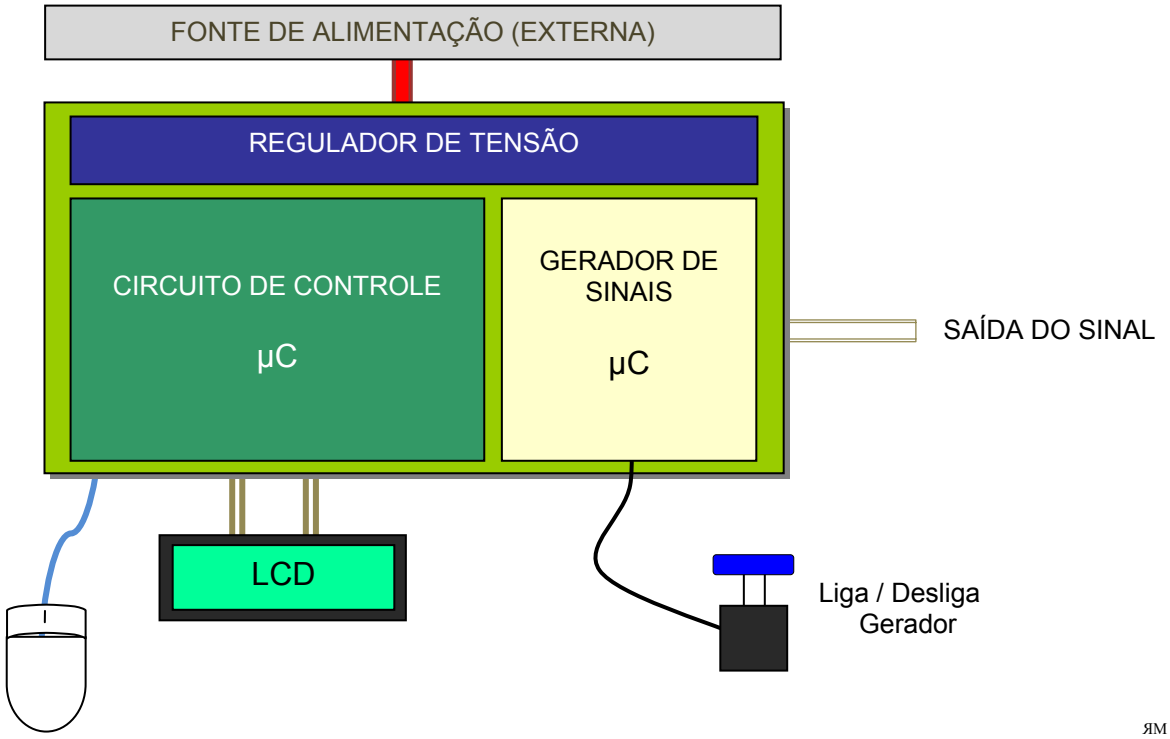


Figura 18 – Bloco de Controle e Geração de Sinais
Fonte: Autor

Regulador de Tensão

Como pode ser visto na Figura 18, os módulos eletrônicos de controle e de geração de sinais são alimentados por uma fonte de alimentação externa. A fonte de alimentação externa do módulo de controle pode variar entre 7 e 12 volts e deve ser capaz de fornecer uma corrente de até 300mA. No protótipo montado para a demonstração do projeto foram utilizadas duas baterias de celular que fornecem juntas 7,5 volts a 380mA. O bloco que contém os módulos de controle e gerador de sinais é um circuito de eletrônica digital. Para que os circuitos digitais sejam alimentados corretamente é necessário adequar a tensão da fonte de alimentação externa para o nível de tensão aceito pelos circuitos digitais. Para executar esta tarefa, foi utilizado um circuito regulador de tensão. No mercado existem diversos circuitos integrados para esta finalidade, sendo que os mais conhecidos são os reguladores lineares da família 78XX.

O CI utilizado no projeto para regular a tensão de alimentação do bloco digital é o LM7805 da fabricante *National Semiconductors*. Este regulador é configurado de fábrica para regular a tensão na saída em 5 volts. O modo de utilização desse componente eletrônico é bem simples conforme pode ser visto na Figura 19.

Basicamente, apenas dois componentes externos são necessários, sendo eles o capacitor de entrada e o capacitor de saída.

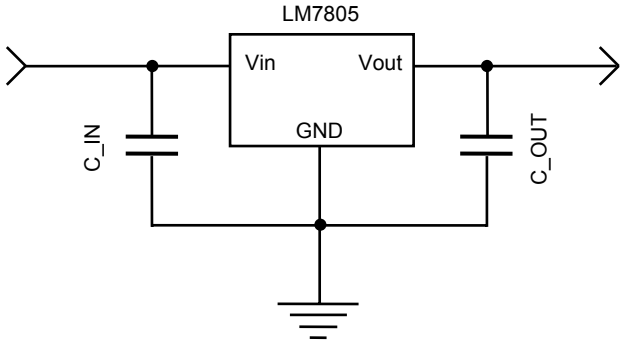


Figura 19 – Regulador de Tensão LM7805
Fonte: Autor

Módulo Eletrônico de Controle

O módulo eletrônico de controle é responsável pela execução da interface de controle descrita no item 3.1.1. Este módulo também faz o controle da comunicação de dados entre o módulo de controle e o módulo gerador de sinais. O núcleo deste módulo é o micro-controlador PIC16F877. Este micro-controlador é bastante poderoso e será utilizado para controlar a entrada e saída do sistema, bem como, para definir os parâmetros de configuração do eletrolisador. Para executar as funções de controle, um programa de computador *(firmware)* é instalado nesse micro-controlador. Este programa esta descrito no item 3.1.3. A escolha do PIC16F877 para ser o micro-controlador principal do protótipo do Eletrolisador foi feita devido à complexidade e à quantidade de tarefas que deve ser executadas no módulo de controle. Este módulo é o próprio sistema operacional do eletrolisador. Ele é responsável pelo controle das principais atividades do sistema. Estas atividades estão descritas abaixo:

- Execução da Interface de Controle
- Leitura dos dispositivos de entrada
- Controle dos dispositivos de saída
- Controle do sistema de comunicação
- Controle parcial do módulo gerador de sinais

Para que as operações de controle sejam feitas pelo usuário do sistema, foram implementados mecanismos de entrada e saída que permitisse tanto ao usuário interagir com o sistema como a possibilidade do sistema responder a esta interação.

No módulo eletrônico de controle, o dispositivo de entrada é um *mouse* convencional de três botões com a função *roller* no botão central. O dispositivo de saída deste módulo é um *display* LCD de dezesseis colunas por duas linhas (16x2). Além disso, há também um *led* para sinalizar a transmissão de dados entre o módulo de controle e o módulo do gerador. O *firmware* instalado no micro-controlador foi desenvolvido para que o módulo de controle faça a leitura dos dispositivos de entrada, bem como a execução da escrita dos dados de saída ao LCD e ao *led*. O micro-controlador também é responsável pela comunicação de dados entre o módulo de controle e o módulo gerador de sinais. Esta comunicação é feita através de uma rede I^2C, sendo que micro-controlador do módulo de controle opera como mestre e o micro-controlador do módulo gerador de sinais opera como escravo.

A Figura 20 ilustra as entradas e saídas do módulo de controle. As setas indicam o sentido da comunicação. As setas que se direcionam ao PIC correspondem ao fluxo de entrada de dados e as que se direcionam para fora do PIC correspondem ao fluxo de saída de dados. Observe que na comunicação entre o PIC do módulo de controle e o módulo gerador de sinais as setas são de entrada e saída, isto significa que essa comunicação é bidirecional, o PIC tanto envia dados ao gerador como recebe dados deste através da rede I^2C.

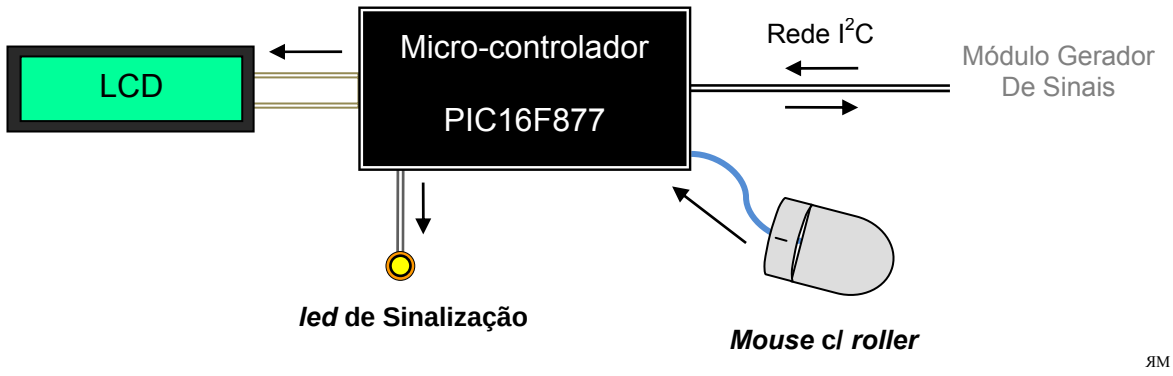


Figura 20 – Entradas e saídas do módulo de controle
Fonte: Autor

O módulo de controle foi primeiramente concebido como um circuito eletrônico. Desta forma, um esquema eletrônico foi criado para definir os componentes utilizados neste circuito e suas respectivas conexões. O diagrama em blocos deste esquema eletrônico é ilustrado na Figura 21. Como pode ser visto nesta figura, as linhas de entrada e saída são ilustradas como barramento, isto é, possuem um traço na diagonal com o número de vias daquele barramento. Exemplo: O barramento de dados entre o

PIC e o LCD possui quatro vias de dados, isto é, quatro pinos de uma determinada porta do micro-controlador são utilizados neste barramento. O esquema eletrônico completo do módulo de controle está ilustrado no Apêndice C.

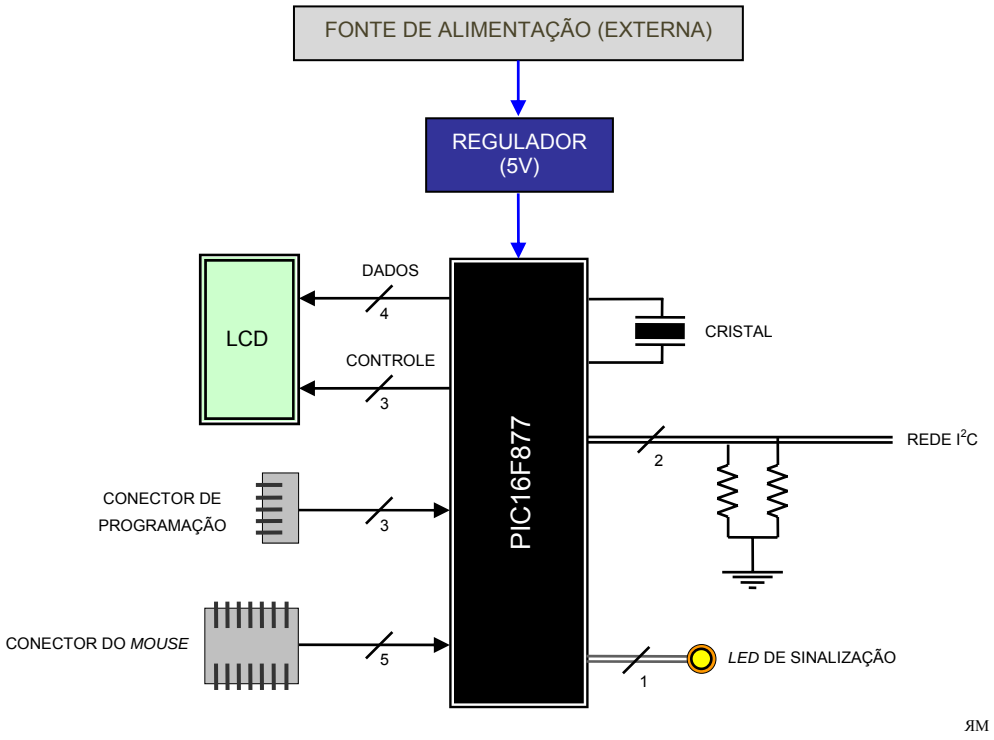


Figura 21 – Diagrama em blocos do esquema eletrônico do módulo de controle
Fonte: Autor

Módulo Gerador de Sinais

O módulo gerador de sinais é o circuito eletrônico micro-controlado responsável pela geração dos pulsos elétricos utilizados no eletrolisador. Este módulo foi concebido separadamente do módulo de controle para que sua execução seja exclusivamente a geração de sinais. O módulo de controle poderia gerar os sinais do eletrolisador, porém como neste módulo há outras atividades como a leitura da entrada e a escrita da saída, a geração de sinais seria interrompida em toda operação de E/S tornando o sinal elétrico não sincronizado, o que dificultaria significativamente a análise dos resultados do projeto. Desta forma o módulo gerador de sinais é um circuito dedicado exclusivamente às atividades de geração de sinais do eletrolisador.

O micro-controlador utilizado no módulo gerador de sinais é o PIC16F88. Este micro-controlador é mais simples do que aquele utilizado no módulo de controle. Isto se deve ao fato de que o módulo gerador de sinais possui como atividade principal apenas a geração dos sinais do eletrolisador. Como atividade secundária este micro-

controlador recebe do módulo de controle, pela rede I²C, os comandos de controle. Além disso, este dispositivo faz a leitura de um botão de entrada. A Figura 22 ilustra as entradas e saídas do módulo gerador de sinais.

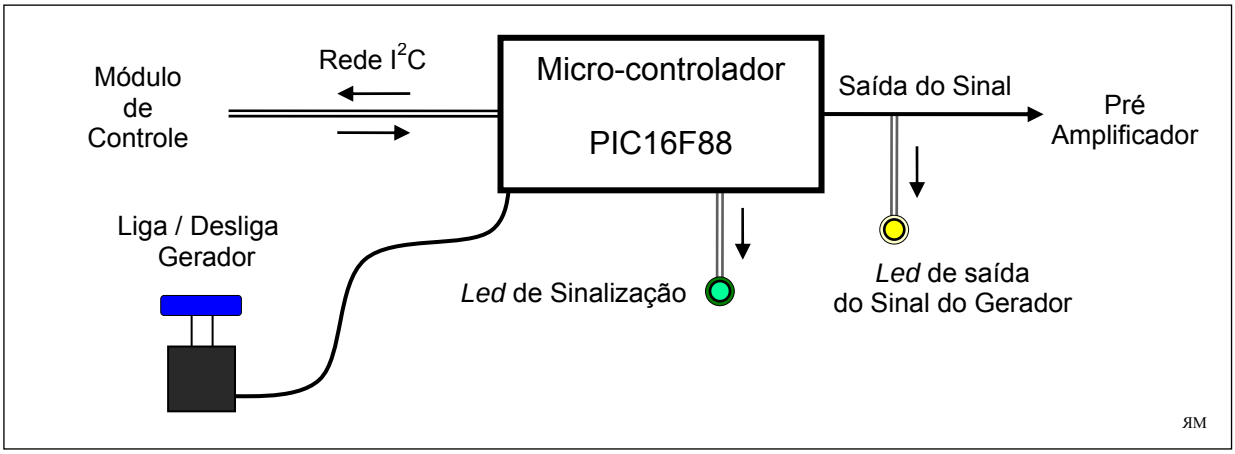


Figura 22 – Entradas e saídas do módulo gerador de sinais.
Fonte: Autor

No módulo de geração de sinais, o dispositivo de entrada é um botão do tipo *push-buttom* utilizado para ligar ou desligar a geração do sinal de saída. Este botão é útil para fazer os experimentos de produção dos gases. Com um cronômetro é possível iniciar e parar a geração do sinal e consequentemente a produção em um tempo específico. O diagrama em blocos do esquema eletrônico do módulo gerador de sinais é ilustrado na Figura 23.

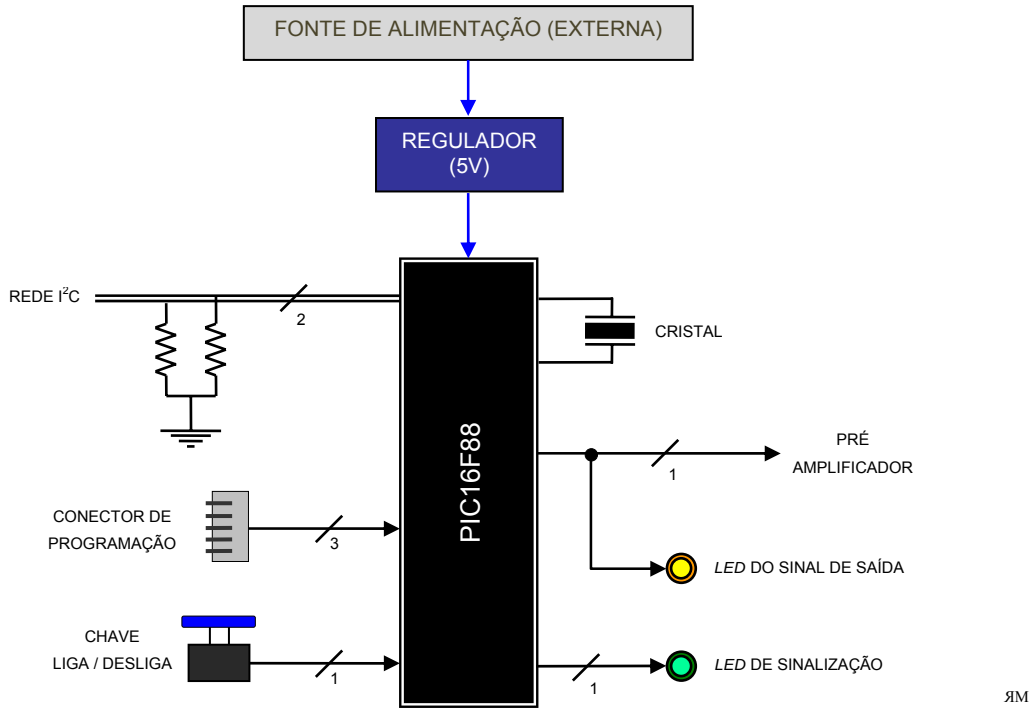


Figura 23 – Diagrama em blocos do esquema eletrônico do módulo gerador de sinais
Fonte: Autor

O sinal gerado por este módulo é basicamente uma sequência de pulsos. Conforme pode ser visto no item 3.1.1 que trata das funções da interface do módulo de controle, existem quatro funções disponíveis na interface de controle que servem para definir os parâmetros de configuração da geração dos sinais do eletrolisador. Estas funções servem para definir os seguintes parâmetros: a quantidade de pulsos que serão produzidos em cada seqüência de pulsos, o período de cada pulso ligado, o período de cada pulso desligado e o período entre o fim de uma seqüência de pulsos e o início da outra. Esta seqüência é produzida constantemente enquanto o gerador estiver ligado. Conforme pode ser vista na Figura 24, esses parâmetros definem a forma de onda do sinal que será gerado na saída do módulo gerador de sinais.

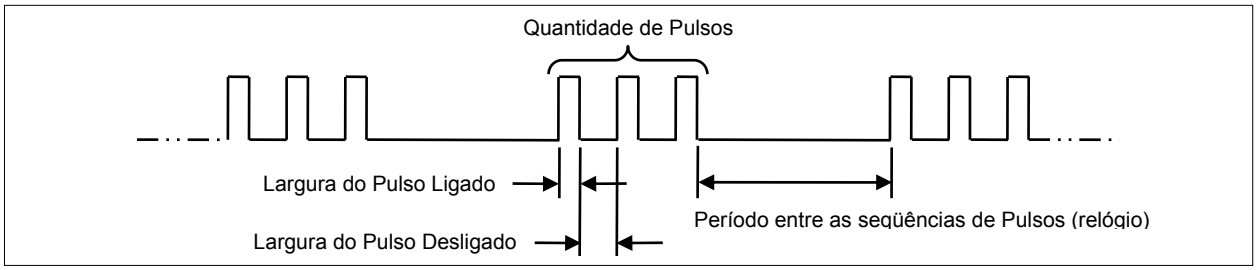


Figura 24 – Forma de onda do sinal gerado pelo Módulo Gerador de Sinais. Fonte: Autor

O *firmware* do módulo gerador de sinais implementa um algoritmo para a geração do sinal com o auxilio do módulo periférico *timer1* do micro-controlador PIC16F88. O parâmetro *relógio* é derivado do temporizador *timer1*.

Como visto no item 3.1.2, o circuito micro-controlado do módulo de controle e o circuito do módulo gerador de sinais estão agrupados em um bloco de eletrônica digital, cuja função é lidar com sinais digitais. Este bloco trabalha com baixas magnitudes de tensão e corrente devido a este ser um circuito exclusivamente digital. Devido a esta característica, foi criado um isolamento entre o bloco digital e o bloco de pré-amplificação e *driver* de potência. Este isolamento evita que algum transiente ou ruído oriundo do segundo bloco possa danificar os circuitos do primeiro bloco, tendo em vista que os componentes do bloco digital são bem mais sensíveis que os componentes do bloco de potência. A Figura 25 ilustra o isolamento.

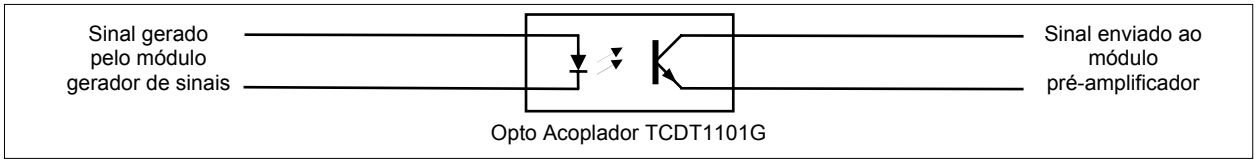


Figura 25 – Isolamento ótico entre o bloco digital e o bloco de potência. Fonte: Autor

Módulos de Pré-amplificação e *Driver* de Potência

O sinal gerado pelo módulo gerador de sinais define a forma de onda básica que será pré-amplificada e enviada ao *driver* de potência. O *driver* de potência é o módulo de amplificação final do sinal gerado. Como a corrente necessária para alimentar corretamente o *driver* de potência é superior a corrente máxima suportada pela porta do micro-controlador PIC16F88 que gera o sinal digital, é necessário um circuito de pré-amplificação desse sinal antes que o mesmo seja enviado a entrada do *driver* de potência.

O *driver* de potência utiliza como amplificador um conjunto de quatro transistores FET de potência, logo é necessário que o circuito de pré-amplificação seja apropriado para atender as especificações de corrente de entrada desses transistores. Para esta finalidade, foi utilizado o circuito integrado MC34151 da fabricante *On Semiconductor*. Este CI é um *driver* invertido de alta velocidade especialmente desenvolvido para aplicações que necessitem que sinais digitais de baixa corrente alimentem grandes cargas capacitivas como é o caso da porta de entrada de transistores FET de potência.

A Figura 26 ilustra o diagrama em blocos do módulo de pré-amplificação e *driver* de potência.

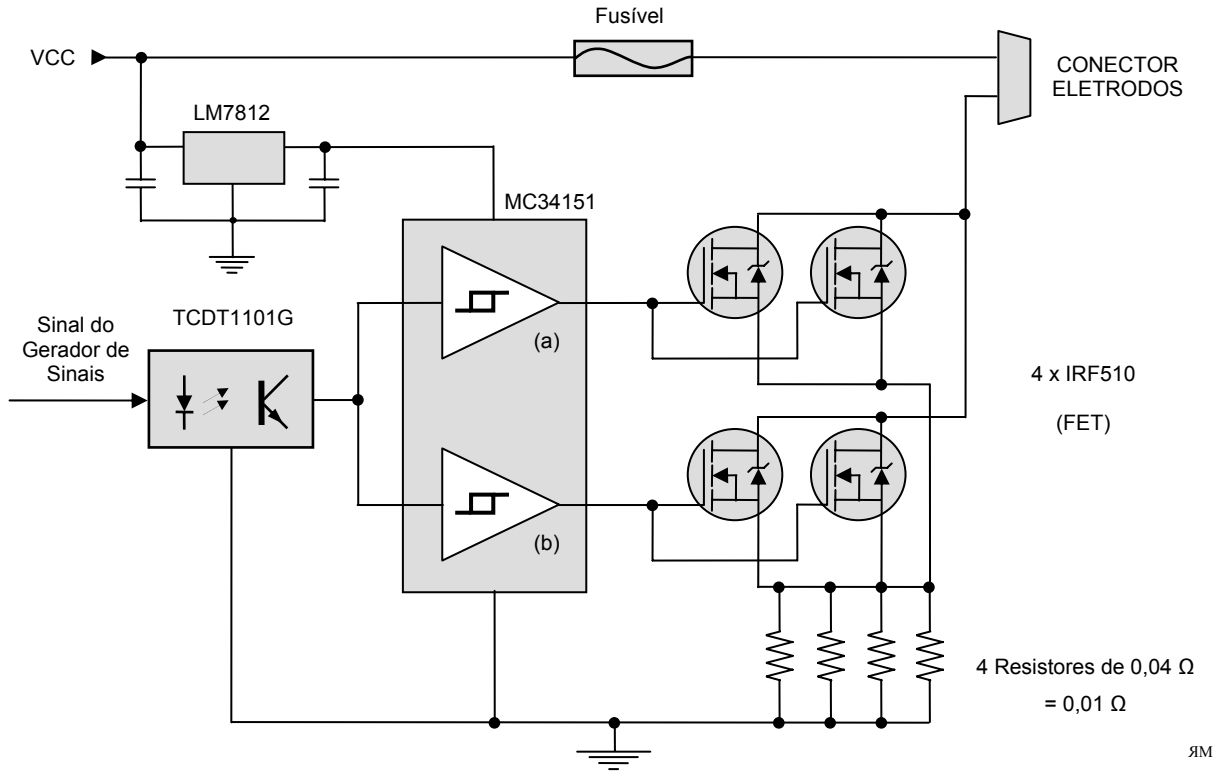


Figura 26 – Diagrama em blocos do esquema eletrônico do módulo de pré-amplificação e *driver* de potência
Fonte: Autor

3.1.3. Programa Embarcado (*Firmware*)

Conforme descrito anteriormente, o projeto eletrolisador micro-controlado da água possui dois módulos eletrônicos micro-controlados. Para cada um dos micro-controladores foi criado um projeto de *software* para contemplar os códigos fontes dos *firmwares* do micro-controlador PIC16F877 (módulo de controle) e do PIC16F88(módulo gerador de sinais).

O projeto do micro-controlador PIC16F877 que contempla o programa da interface de controle possui a seguinte estrutura de arquivos:

main.c – Arquivo principal contendo todo o programa da interface de controle e rotinas para a comunicação com o módulo gerador de sinais bem como as rotinas de controle das entradas e saídas.
main.h – Arquivo *header* do programa principal. Contém as declarações dos recursos utilizados pelo programa principal.
util.c – Contém algumas rotinas úteis como rotinas para conversão de dados.
util.h – Arquivo *header* de *util.c*.
lcd.c – Arquivo contendo todas as rotinas para o controle do LCD.
lcd.h – Arquivo *header* de *lcd.c*.
i2c_drv.c – Rotinas para o uso da rede de comunicação I^2C em modo mestre.

O projeto do micro-controlador PIC16F88 que contempla o programa do módulo gerador de sinais possui a seguinte estrutura de arquivos:

main.c – Arquivo principal contendo todo o programa do módulo de geração de sinais e rotinas para a comunicação com o módulo de controle bem como as rotinas de controle das entradas e saídas.
main.h – Arquivo *header* do programa principal. Contém as declarações dos recursos utilizados pelo programa principal.
i2c.h – Rotinas para o uso da rede de comunicação I^2C em modo escravo.

3.1.4. O Circuito em *protoboard*

Na construção do protótipo do Eletrolisador Micro-controlador da Água, todos os circuitos eletrônicos foram montados em um *protoboard*, espécie de matriz de contatos com milhares de furos e conexões condutoras para a montagem de circuitos elétricos e eletrônicos experimentais. O projeto eletrônico do eletrolisador, como dito anteriormente, foi dividido em dois blocos eletrônicos. O primeiro bloco de eletrônica digital e o segundo bloco de eletrônica de potência. Por este motivo, foram utilizados dois *protoboards*, um para cada bloco eletrônico. A Figura 27 mostra a foto do *protoboard* com o circuito montado do primeiro bloco, detalhando os módulos eletrônicos.

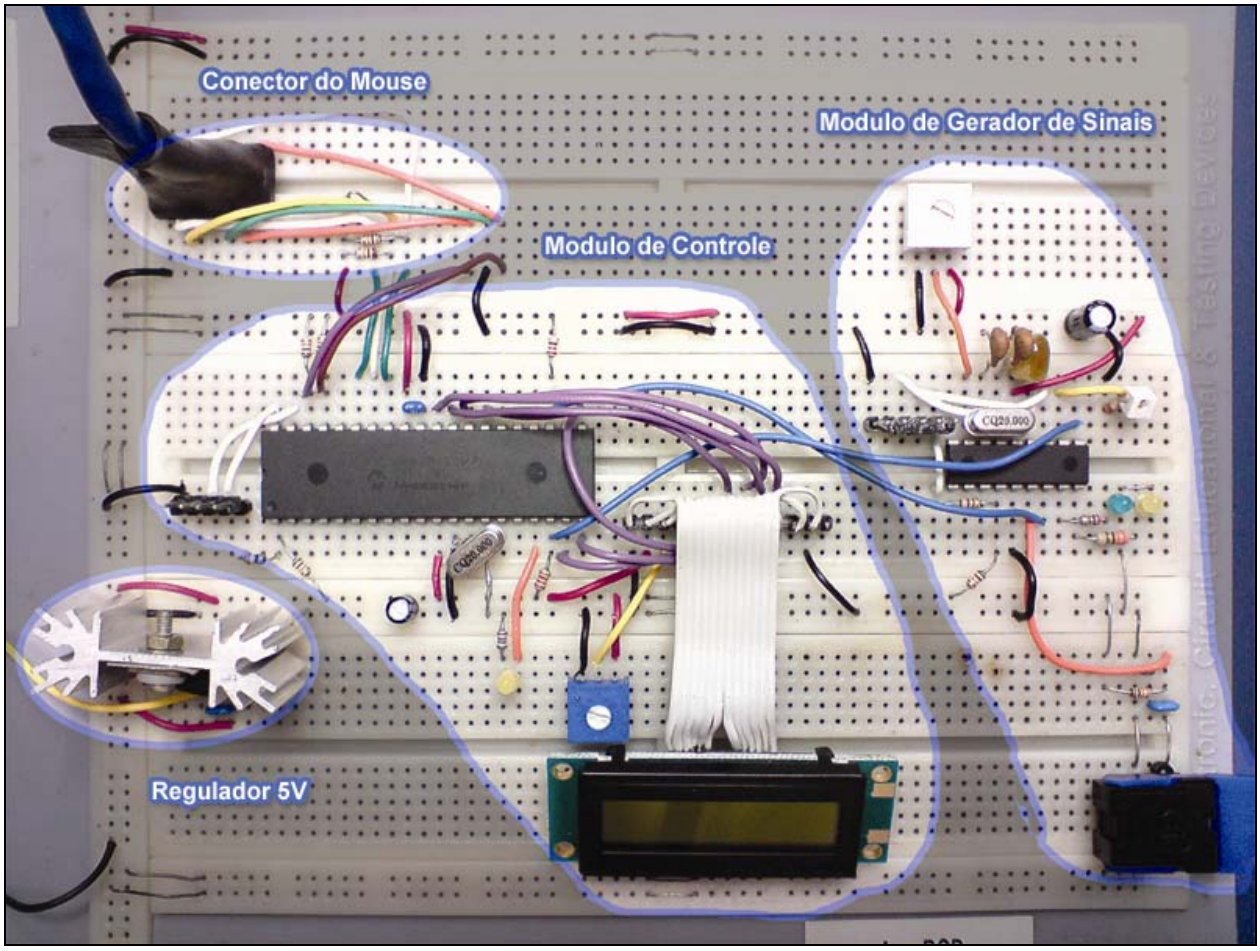


Figura 27 – *Protoboard* dos circuitos eletrônicos do bloco digital
Fonte: Autor

A Figura 28 ilustra o *protoboard* com os circuitos do bloco de potência, contendo os circuitos de acoplamento ótico, pré-amplificação e *driver* de potência.

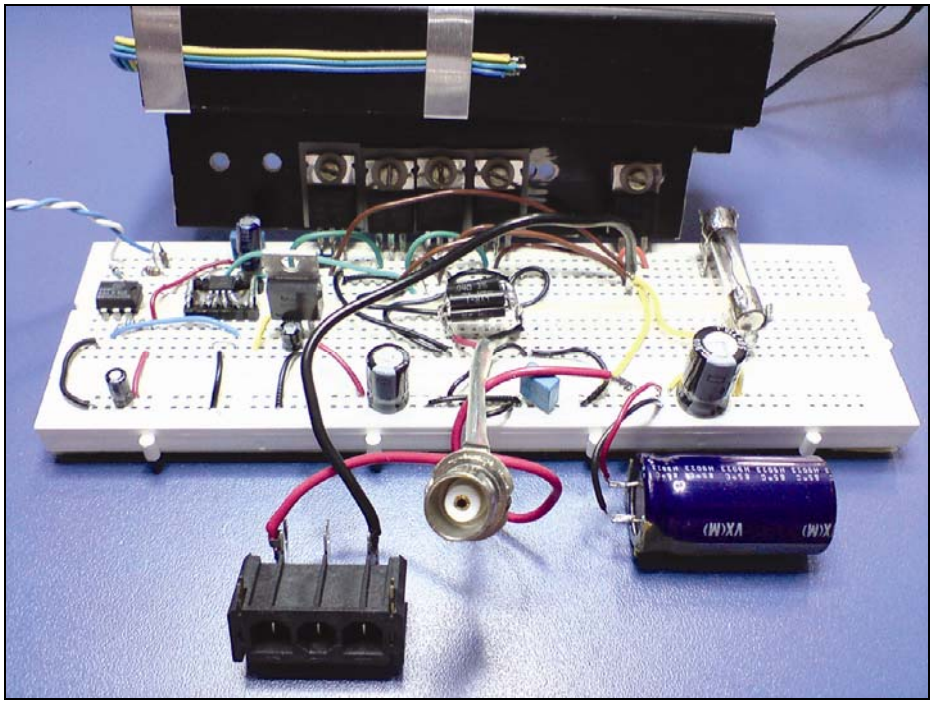
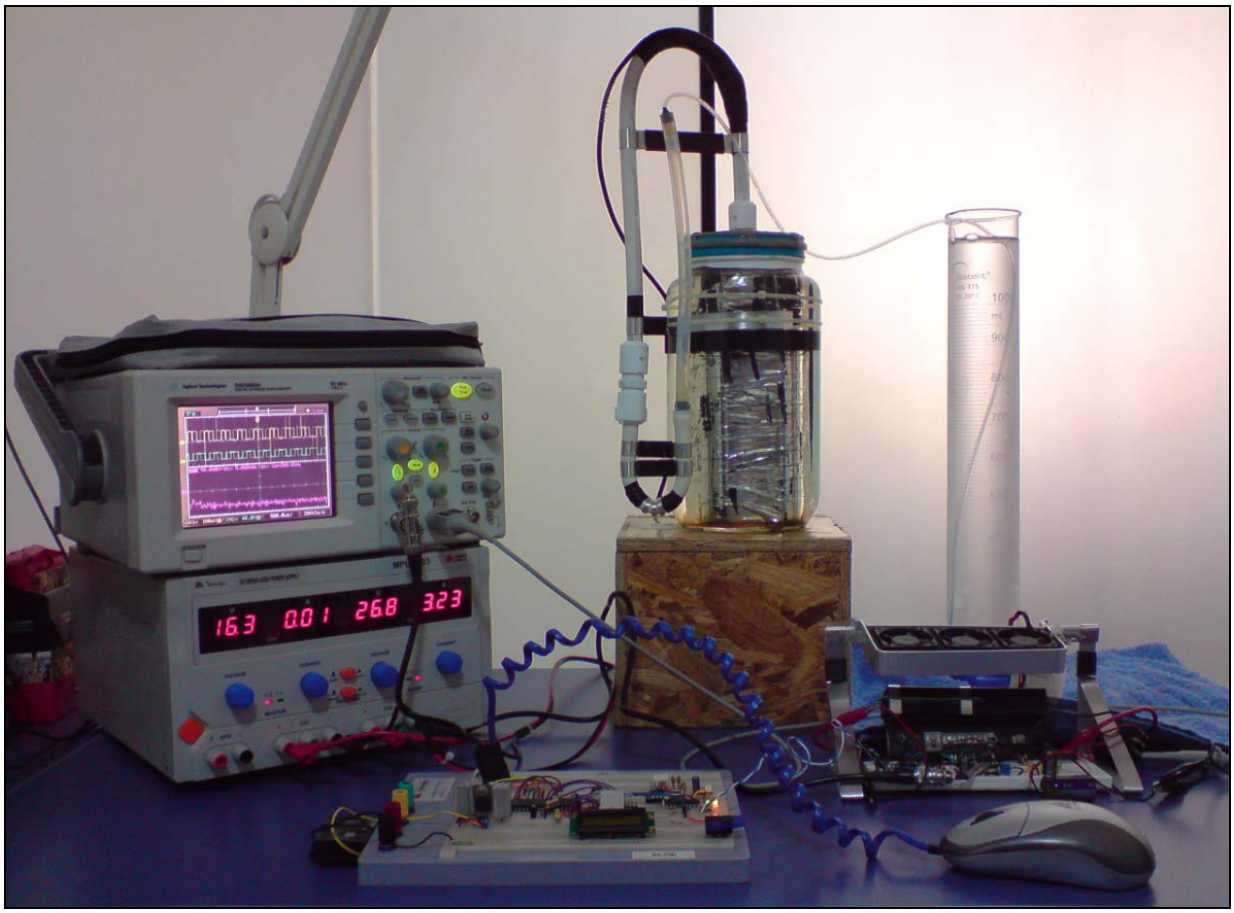


Figura 28 – *Protoboard* dos circuitos eletrônicos do módulo de potência
Fonte: Autor

A imagem abaixo mostra todo o sistema montado e em operação:



Fonte: Autor

3.1.5. O Artefato de geração da eletrólise

Na aplicação convencional da eletrólise da água são utilizados eletrodos metálicos mergulhados em solução eletrolítica para permitir que o fenômeno ocorra. No projeto eletrolisador micro-controlado da água o eletrodo utilizado foi fabricado a partir de tubos de aço inox de uma categoria especial denominada T-304. Este tipo de aço é bastante resistente à corrosão e suas características físicas são ideais para o uso experimental da eletrólise. Ao todo foram fabricados quatro pares de eletrodos para serem utilizados no artefato eletrolisador.

Cada par de eletrodos utilizados no projeto foram construídos com as seguintes dimensões:

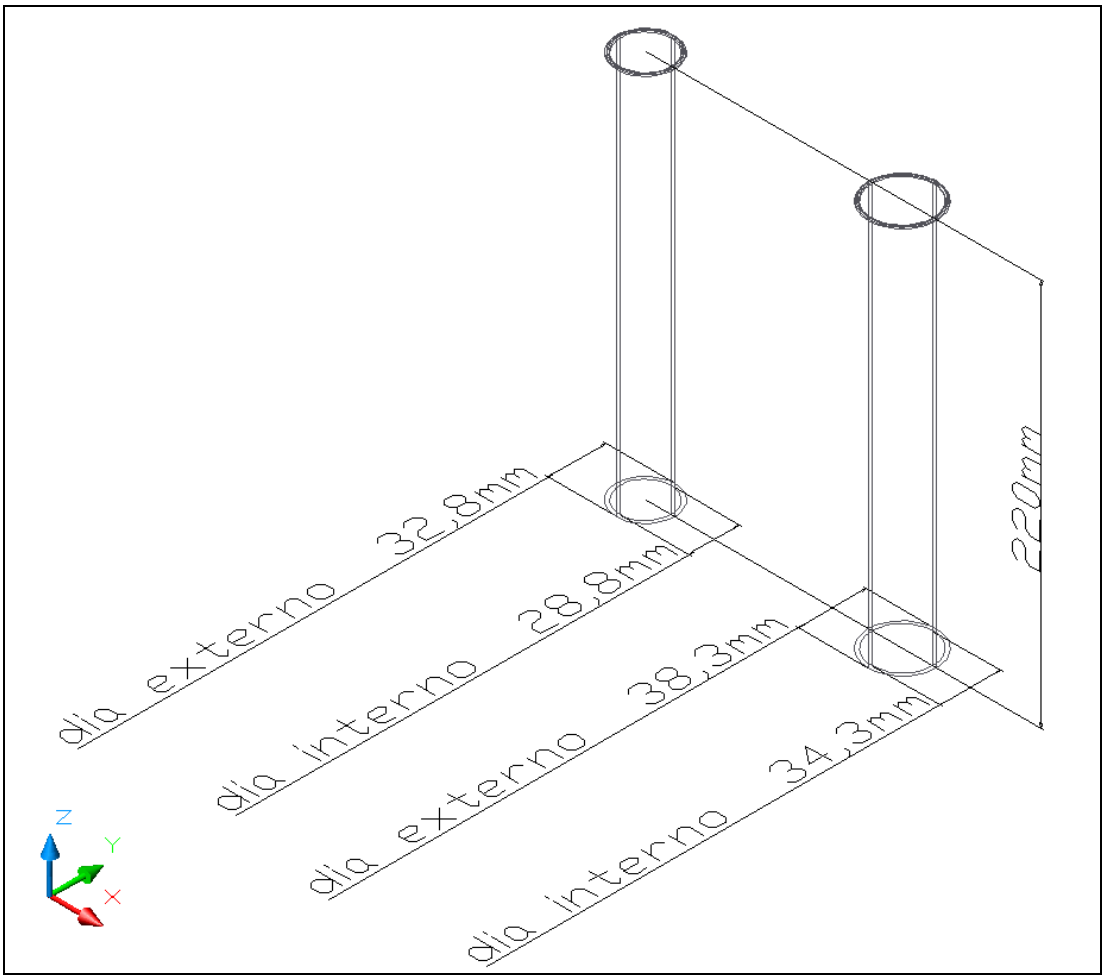
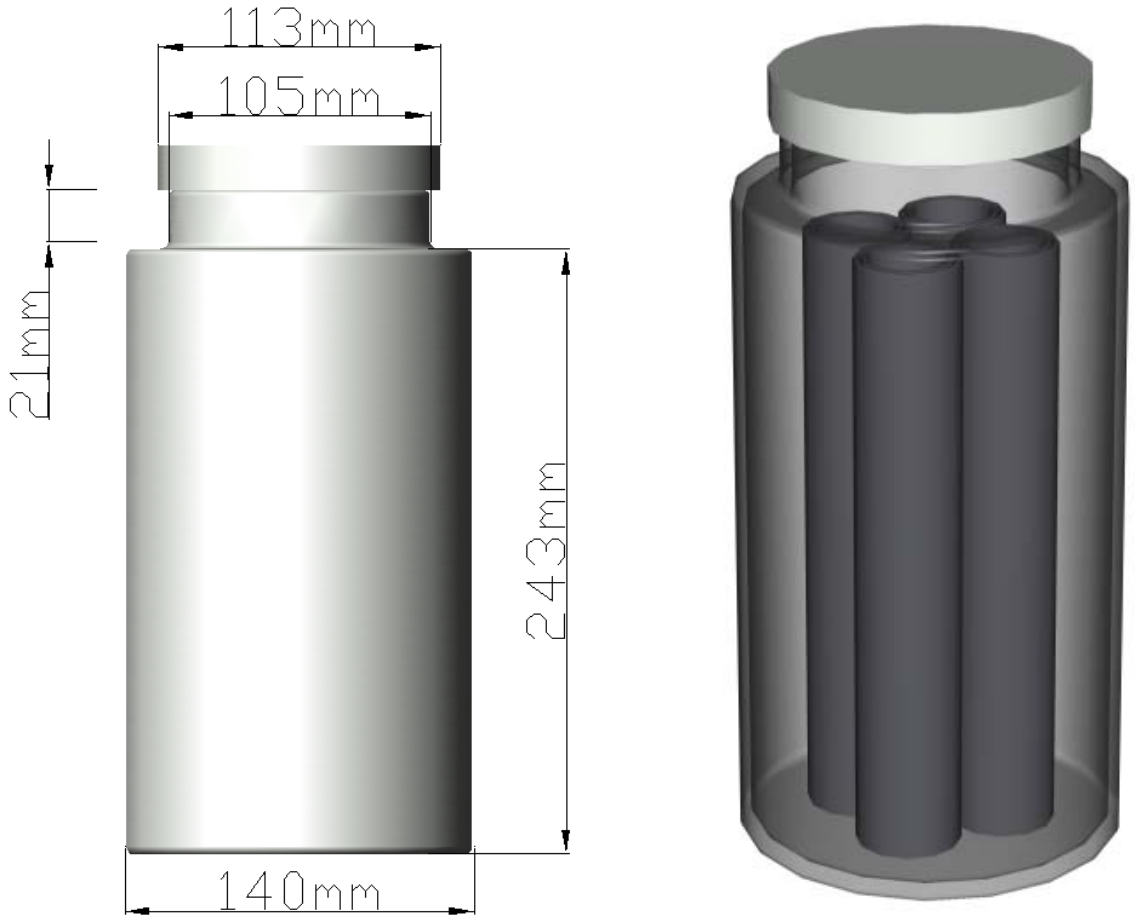


Figura 29 – Dimensões dos eletrodos interno e externo.
Fonte: Autor

O artefato físico onde ocorre o fenômeno da eletrólise foi construído utilizando um recipiente de vidro. A figura abaixo ilustra as seguintes dimensões desse recipiente e a disposição dos eletrodos dentro do recipiente.



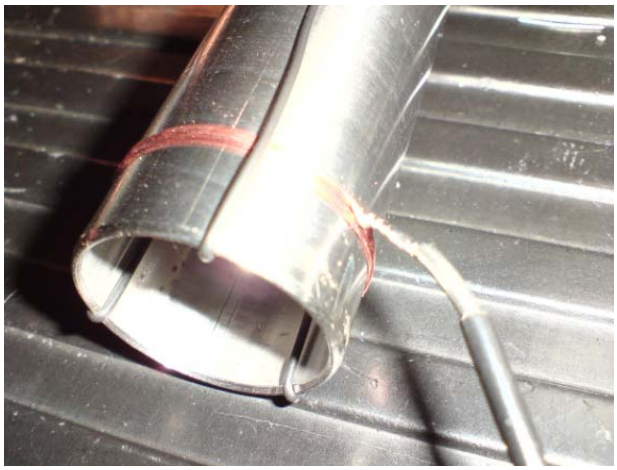
Fonte: Autor

Para que a eletricidade seja aplicada aos eletrodos foram fixados cabos elétricos aos eletrodos da seguinte maneira:

1. Todos os tubos **internos** dos quatro pares de eletrodos foram interconectados entre si e soldados ao cabo de alimentação positiva dos eletrodos.

2. Todos os tubos **externos** dos quatro pares de eletrodos foram interconectados entre si e soldados ao cabo de alimentação negativa dos eletrodos.

O artefato eletrolisador possui uma tampa para o seu fechamento. Um reforço de **araldite®** e **durepox®** foi feito para dificultar a quebra da tampa. Esta tampa possui uma rosca adaptada para permitir sua abertura e fechamento sem prejudicar as instalações. Nesta rosca é conectada a mangueira de saída do gás produzido. Este mesmo orifício onde saem os gases foi utilizado para a passagem do cabo elétrico dos eletrodos, facilitando a instalação como um todo. O cabo elétrico sai do artefato através de um orifício feito na mangueira de saída. Este furo foi utilizado para a passagem do cabo e foi vedado com silicone de alta resistência. Para finalizar a construção do artefato eletrolisador foi criado um mecanismo antichama feito de mangueira de silicone. As fotos abaixo ilustram o processo de fabricação dos eletrodos e do artefato eletrolisador.



Fixação do fio elétrico ao eletrodo

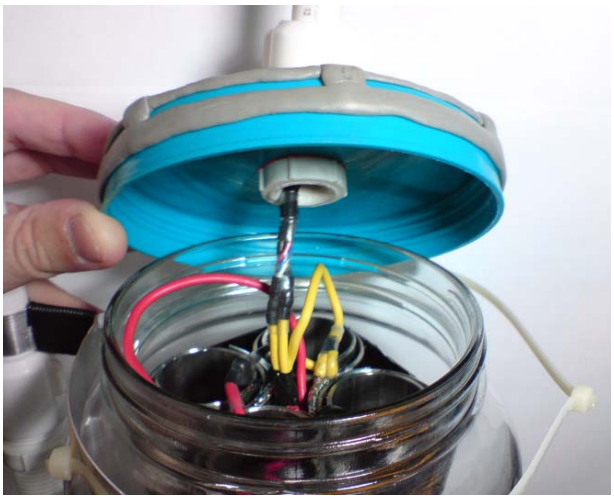
Eletrodos externos

Tampa do recipiente com rosca

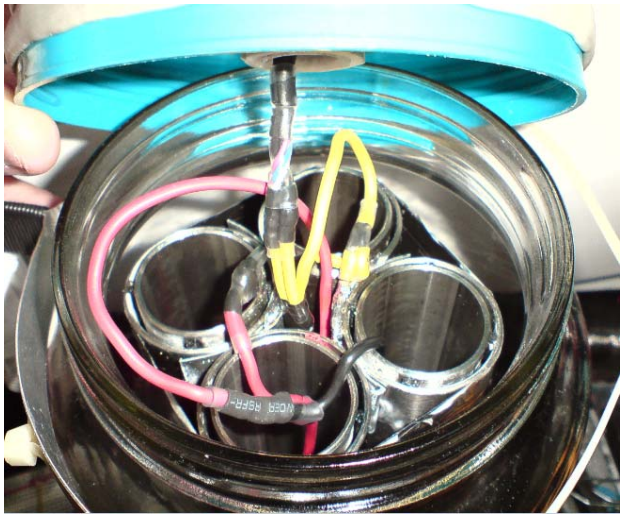
Tampa – Vista interna

Fonte: Autor

Tampa – Detalhe do reforço feito



Detalhe das instalações - I



Detalhe das instalações - II

Artefato montado

Detalhe do local onde o cabo elétrico sai da mangueira de saída do gás.

Fonte: Autor

3.2. Experimentos Realizados

O projeto Eletrolisador Micro-controlado da água foi dividido em duas partes. A primeira parte trata-se do protótipo do equipamento eletrônico. Este equipamento foi concebido para gerar os sinais de alta potência que serão utilizados para a geração da eletrólise. Além da construção do protótipo do eletrolisador, a análise da eletrólise controlada também faz parte do objetivo desse trabalho. Esta análise visa entender as relações entre a aplicação de pulsos elétricos diferenciados e a quantidade de gás produzido pelo processo de eletrólise proposto neste projeto. Os experimentos foram feitos em três soluções aquosas: água potável, água destilada e água adicionada de ácido sulfúrico.

A quantidade inicial de água utilizada em todos os experimentos foi de 3 litros. A solução de água potável utilizada foi a do tipo água de torneira filtrada. A solução de água destilada é do tipo água destilada uma única vez. A solução de água adicionada de ácido sulfúrico é composta por 3 litros de água destilada misturada com 2 ml de ácido sulfúrico.

3.2.1. Metodologia

Para alcançar o objetivo de se relacionar os pulsos elétricos gerados pelo eletrolisador com a quantidade de gás produzido, foi necessário definir uma metodologia a ser aplicada nos experimentos da eletrólise. Esses experimentos feitos em laboratório foram executados de forma coordenada e em condições semelhantes para que os resultados fossem alcançados com uma precisão satisfatória. Para executar tais experimentos, as seguintes considerações foram levadas em conta:

- A sequência de experimentos para cada tipo de solução aquosa foi feita sem interrupções para que condições como a temperatura e a pressão atmosférica não interferissem no experimento;
- A quantidade de água no artefato eletrolisador foi exatamente igual para os três tipos de solução aquosa, bem como a forma de obtenção dos dados;
- Os dados a serem analisados foram capturados da mesma forma em todos os experimentos.

Dados Analisados

Em todos os experimentos os seguintes dados foram obtidos para análise:

1. Corrente aplicada no *driver* de potência;
2. Tensão aplicada nos eletrodos;
3. Tempo de captura em segundos;
4. Quantidade de gás produzido;

Com a corrente e a tensão capturada e normalizada o seguinte dado também é composto na análise: Potência aplicada no sistema;

Parâmetros de configuração

Em cada experimento feito, uma série de parâmetros foi definida como sendo o conjunto de variáveis que serão modificadas para que a analise comparativa fosse feita. Esses parâmetros ou variáveis estão descritos abaixo:

Variáveis do gerador de sinais:

- Relógio (µs)
- Quantidade de pulsos
- Valor do Pulso LIGADO
- Valor do Pulso DESLIGADO

Aquisição dos dados

Os dados referentes a cada experimento foram capturados por meio de um osciloscópio digital da marca *Agilent* da série 3000 e de seu *software* de captura. A captura desses dados foi feita de forma discreta em períodos definidos de acordo com os parâmetros de configuração. A quantidade de amostras na aquisição foi fixada em 1200 pontos para todos os experimentos. O gás produzido em cada experimento foi armazenado em um tubo de vidro graduado. O tempo de produção desse gás foi fixado em 60 segundos, ou seja, foi tomado como dado a quantidade de gás produzido nesse período. Essa quantidade foi medida com o auxilio do tubo graduado, que possui a graduação em mililitros.

Em todos os experimentos, os dados capturados foram aqueles definidos pelo projeto, ou seja, a corrente a tensão aplicada nos eletrodos. Como esta corrente e tensão não são contínuas, isto é, o sinal aplicado não é contínuo e sim em forma de pulsos, foi utilizado o osciloscópio para capturar a forma de onda de cada sinal. No canal 1 do osciloscópio foi capturado o sinal da corrente aplicada em ampéres. Este sinal é adquirido por meio de um resistor de baixíssima resistência (0,01 ohms) que fica entre o terra o pino *source* dos transistores FET de potência. O sinal de tensão aplicado nos eletrodos é adquirido com a ponta de prova entre o terra e o pino negativo do conector dos eletrodos. A Figura 30 ilustra as conexões das pontas de prova.

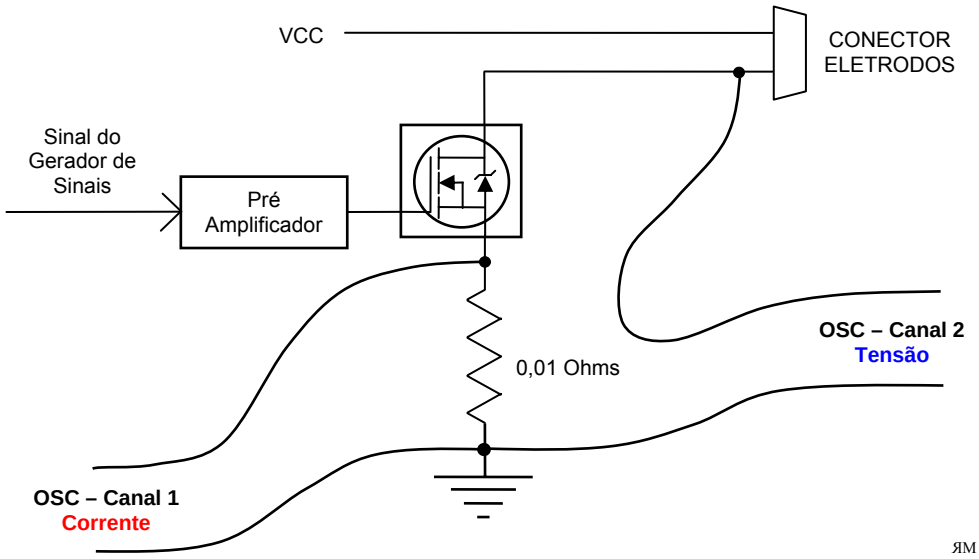


Figura 30 – Conexões das pontas de prova
Fonte: Autor

Estes sinais são capturados pelo programa de capturas de dados do osciloscópio digital. A tela do programa é ilustrada na Figura 31.

Figura 31 – Programa de captura de dados do osciloscópio digital

Conforme pode ser visto na tela de trabalho do programa de captura do osciloscópio digital, além dos controles encontrados a esquerda da tela, há três janelas de armazenamento dos dados e uma janela que é o painel virtual do osciloscópio. Por meio desse painel é possível controlar completamente o osciloscópio, já que há uma conexão de dados entre o programa e o instrumento por meio de um cabo *usb*. A tela desse painel virtual é mostrada na Figura 32.

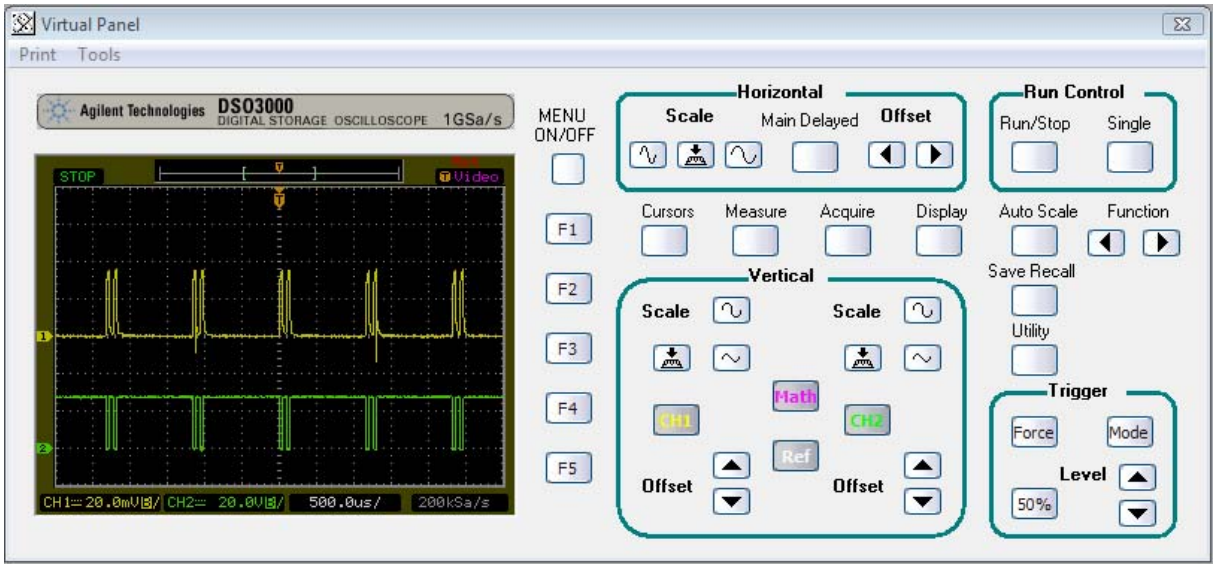


Figura 32 – Painel Virtual do Osciloscópio Digital

As janelas de armazenamento de dados são: *measurement, data* e *waveform*. A janela utilizada no experimento para capturar os dados foi a janela *data*. Esta janela faz a captura de 1200 amostras do sinal que foi lido pelo osciloscópio em um dado instante. Este instante especificamente é selecionado após a estabilização do sinal aplicado. Como em todas as experiências o tempo de produção foi de 60 segundos, o sinal capturado foi feito no tempo médio, isto é, 30 segundos após o início da produção do gás. Uma vez tendo capturado o sinal, através da tela *data* é possível exportar os dados de cada canal. Feito isso, esses dados são inseridos em uma planilha de dados para a geração dos resultados e análise.

Interpretação dos dados

Conforme descrito na página 55, os dados analisados foram a corrente aplicada no *driver* de potência, a tensão aplicada nos eletrodos, o tempo de aquisição em segundos e a quantidade de gás produzido. Os dados foram capturados em 1200 amostras do canal 1 e 1200 amostras do canal 2 do osciloscópio. Estes dados foram colocados em uma planilha para que fossem interpretados. As planilhas contendo os dados das amostras estão descritas no anexo I.

A forma como os dados capturados foram interpretados é dada logo abaixo:

1. O sinal da corrente foi capturado através da tensão sobre um resistor de 0,01Ω. Pela primeira lei de ohm, a corrente sobre um resistor é dada pelo valor da divisão entre a tensão e a resistência, conforme equação abaixo:

$$V = R \cdot I \qquad (1)$$

   Desta forma, o valor capturado pelo canal 1 do osciloscópio foi colocado em uma coluna da planilha de analise de dados chamada **C(Canal 1)**. Para adequar o valor capturado ao valor correto de corrente sobre o resistor foi então aplicado à equação (1) em uma segunda coluna chamada de **C(A)**. Nesta coluna estão os dados reais da corrente medida.

2. O sinal da tensão lida foi feito entre o terra e o pino negativo dos eletrodos. Isso se deve ao fato de que se o sinal fosse medido entre o pino negativo e o pino positivo dos eletrodos, haveria um curto no sistema devido ao estágio de potência fazer o chaveamento do sinal pelo pino negativo (opção pelo uso de transistores FET de canal N). Devido a este motivo, o sinal capturado do canal 2, que corresponde a tensão lida sobre os eletrodos é capturada de forma inversa. Além disso, por se tratar da aplicação de sinais em eletrodos imersos em solução ácida, uma diferença de potencial (tensão) ocorre nos eletrodos e isto modifica o sinal capturado pelo osciloscópio. O valor dessa tensão deve ser subtraído do valor capturado para se achar a tensão correta sobre os eletrodos. Portanto o seguinte procedimento foi adotado para normalizar o sinal de forma a poder utilizá-lo nas análises:

a) O sinal capturado pelo canal 2 foi colocado em uma coluna chamada **T (Canal 2)**.

b) Foi introduzida uma tabela com os valores de mínimo e de máximo encontrados nas 1200 amostras e a respectiva diferença entre esses dois valores. Esta diferença representa a tensão presente nos eletrodos que deve ser subtraída do valor capturado e foi chamada de ***dif***. O valor mínimo e máximo foram chamados de ***min*** e ***max***.

c) Foi criada uma coluna com o nome de **Tensão(INV)** para armazenar o valor da tensão já normalizado. A fórmula para encontrar o valor correto da tensão foi a seguinte:

SE ((***dif*** – **VALOR_LIDO**) < 2) : **Tensão(INV) =** 0**;**
SENÃO : **Tensão(INV) =** ***dif*** – **VALOR_LIDO;**

Obs.: O valor 2 utilizado na fórmula foi colocado para eliminar pequenos ruídos do sinal.

3. O sinal da potência foi colocado em uma coluna chamada **Potência** e é derivado diretamente do produto da corrente (coluna **C(A)**) com a tensão normalizada (**Tensão(INV)**) segundo a equação abaixo:

$$P = V \cdot I \qquad (2)$$

4. O tempo gasto na captura dos 1200 pontos é colocado em uma coluna chamada **Tempo(s)**.

5. Como o sinal aplicado não é contínuo, a forma utilizada para de se relacionar a quantidade produzida com o sinal aplicada foi a integração numérica do sinal de potência pelo método dos trapézios. Como o sinal é capturado de forma discreta (não contínua), isto é, em períodos de tempo definidos, a aplicação da integração numérica do sinal pelo método dos trapézios é aplicada em sua forma mais simples, conforme descrita abaixo na equação (3). [2]

$$I = (h/2) \cdot (y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n) \qquad (3)$$

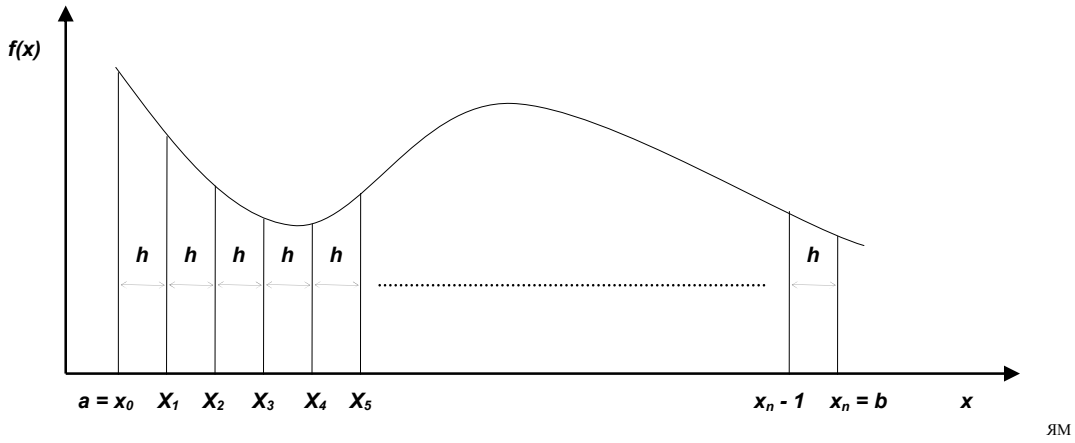


Figura 33 – Aplicação sucessiva da regra dos trapézios

O cálculo da integral foi feito em cima do sinal da potência sobre os 1200 pontos da coluna **Potência**. Uma coluna chamada **P*2** foi criada para formar os valores $2 \cdot y_n$ utilizado na equação (3). Com a integral calculada, o valor do sinal em watts por segundo foi dado pelo produto da integral pelo tempo total da captura do sinal. Este valor está expresso na célula **Consumo**.

6. Todos os experimentos foram realizados com o tempo de produção do gás estipulado em 60 segundos. Desta forma foi anotada a quantidade em ml de gás produzido por meio do tubo graduado de armazenamento do gás.

7. Com a quantidade de gás produzido em ml foi então calculado quantos mililitros por watt são produzidos com o dado sinal elétrico aplicado. Este dado está expresso na coluna ***η***.

A Figura 34 ilustra uma das tabelas contendo o resultado de um dado experimento.

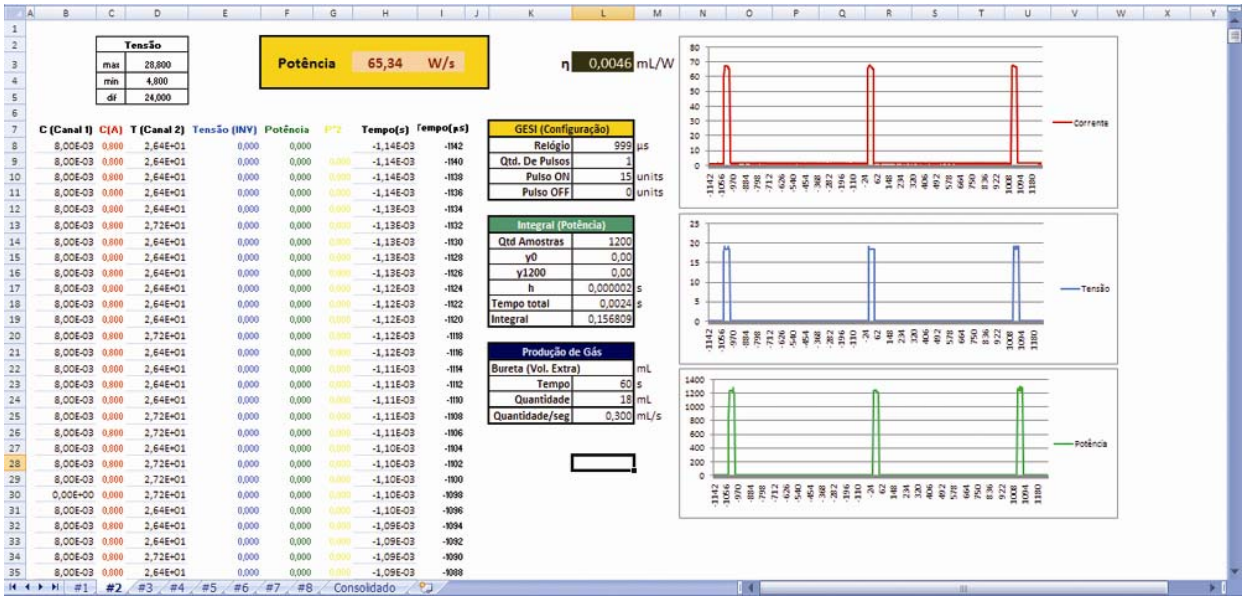


Figura 34 – Planilha de captura de dados de um experimento.
Fonte: Autor

3.2.2. Consolidação dos Resultados dos Experimentos

Ao todo foram feitos mais de 40 experimentos de produção dos gases hidrogênio e oxigênio com o eletrolisador projetado. De acordo com os resultados ficou evidente que a produção mais intensa de gás foi alcançada com a solução aquosa misturada com ácido sulfúrico. Desta forma, foram realizados dois experimentos com oito sessões de captura cada na solução de água destilada misturada com ácido sulfúrico. Os dados consolidados destes dois experimentos serão apresentados em detalhes nos próximos itens. As planilhas e gráficos de cada sessão desses experimentos se encontram no **Apêndice A**.

Experimento I

No experimento I a eletrólise foi feita em uma solução eletrolítica composta de 3 litros de água destilada adicionada de 2 ml de ácido sulfúrico. Neste experimento, a quantidade de pulsos gerados na sequência de pulsos foi fixada em apenas um pulso. O parâmetro do gerador de sinais variado foi a largura de pulso ligado. Os parâmetros relógio e pulso desligado ficaram estáticos assim com a quantidade de pulsos. A Tabela 4 detalha a configuração utilizada nesse experimento:

Tabela 4 – Configuração dos parâmetros do gerador de sinais – Experimento I

Configuração dos Parâmetros do Gerador de Sinais	
Parâmetro	**Valor**
Relógio	999
Quantidade de Pulsos	1
Pulso LIGADO	**X (Variável)**
Pulso DESLIGADO	0

Conforme pode ser visto na tabela, o parâmetro variável foi o pulso ligado. O valor deste parâmetro foi variado entre 10 e 45 unidades em múltiplos de 5, totalizando 8 variações desse parâmetro. O objetivo desse experimento é verificar se a sequência da alteração do parâmetro influencia na produção do gás. O resultado consolidado desse experimento está ilustrado na Tabela 5. O gráfico desse resultado é dado na Figura 35.

Tabela 5 – Resultado consolidado do experimento I

Variações	Variável (Pulso LIGADO)	ml/KW
#1	10	3,482
#2	15	4,592
#3	20	4,374
#4	25	4,316
#5	30	4,464
#6	35	4,272
#7	40	4,516
#8	45	4,513

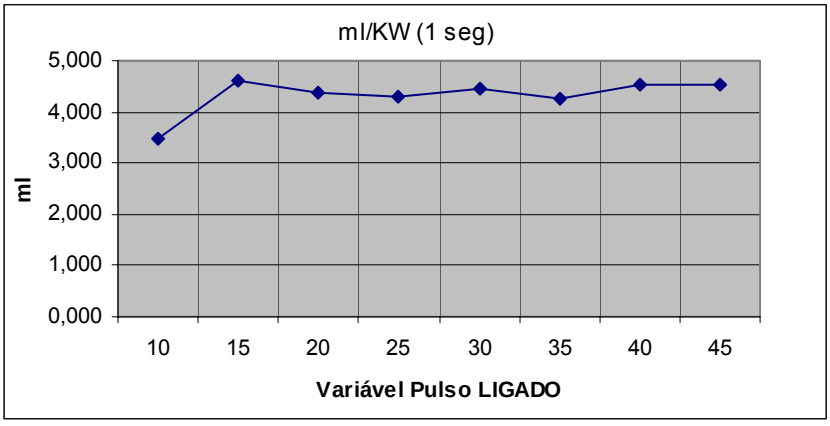


Figura 35 – Gráfico do resultado consolidado do experimento I

Experimento II

No experimento II a eletrólise foi feita com a mesma solução eletrolítica do experimento I, isto é, em uma solução eletrolítica composta de 3 litros de água destilada adicionada de 2 ml de ácido sulfúrico. Neste experimento, o parâmetro que foi variado foi a quantidade de pulsos gerados na sequência de pulsos. Os parâmetros relógio, pulso ligado e pulso desligado ficaram estáticos. A Tabela 6 detalha a configuração utilizada nesse experimento:

Tabela 6 – Configuração dos parâmetros do gerador de sinais – Experimento II

Configuração dos Parâmetros do Gerador de Sinais	
Parâmetro	**Valor**
Relógio	999
Quantidade de Pulsos	**Variável de 1 a 8**
Pulso LIGADO	20
Pulso DESLIGADO	30

Conforme pode ser visto na tabela, o parâmetro variável foi a quantidade de pulsos. O valor deste parâmetro foi variado entre 1 e 8. O objetivo desse experimento é verificar se a sequência da alteração do parâmetro irá influenciar na produção do gás. O resultado consolidado desse experimento está ilustrado na Tabela 7. O gráfico desse resultado é dado na Figura 36.

Tabela 7 – Resultado consolidado do experimento II

Variações	Variável (Quantidade de Pulsos)	ml/KW
#1	1	2,797
#2	2	5,351
#3	3	4,974
#4	4	4,445
#5	5	4,728
#6	6	3,952
#7	7	4,018
#8	8	3,965

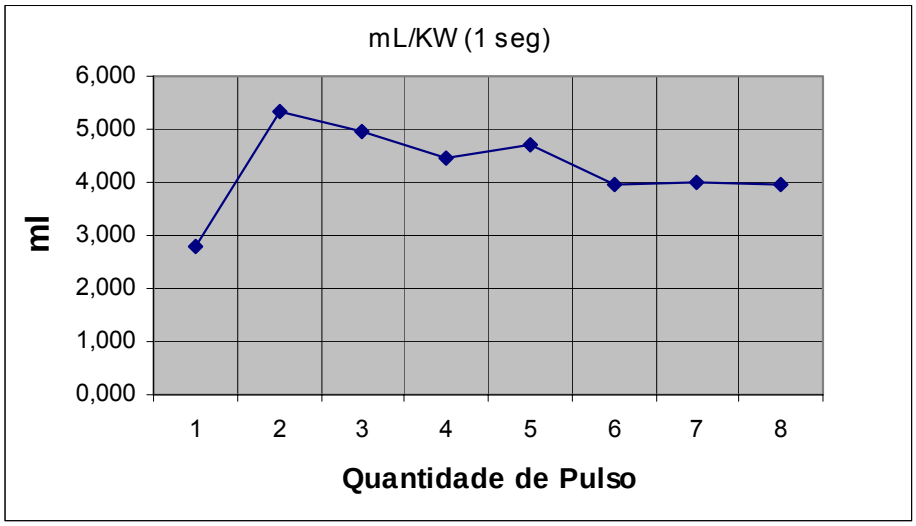


Figura 36 – Gráfico do resultado consolidado do experimento II

Conclusão

De acordo com os resultados obtidos nos experimentos, ficou ilustrado que dependendo do sinal aplicado nos eletrodos do eletrolisador e da solução eletrolítica utilizada, não há uma alteração significativa no volume de gás produzido pelo processo da eletrólise. Segundo a teoria básica do processo da eletrólise, o volume de produção de gás neste processo é proporcional a quantidade de carga inserida nos eletrodos. Os experimentos realizados comprovam esta teoria. As pequenas variações apresentadas nos resultados podem ser atribuídas aos possíveis erros de leitura e as características dos componentes eletrônicos utilizados no projeto. Também deve ser levada em consideração que o protótipo eletrônico foi montado em *protoboard* e devido à natureza desse tipo de montagem e por se tratar de um equipamento que trabalha com pulsos elétricos de alta magnitude, a presença de ruídos pode interferir significativamente no resultado final das medições. Este aspecto pode ser alvo de aprimoramento em futuros projetos.

O equipamento desenvolvido neste trabalho mostrou ter a capacidade de se controlar a produção dos gases hidrogênio e oxigênio através do controle do sinal elétrico introduzido nos eletrodos do eletrolisador. Desta maneira como sugestão, a continuidade desse trabalho pode ser feita no sentido de se explorar as aplicações da eletrólise, seja na produção de gases, aplicação da oxirredução para a composição ou modificação de materiais metálicos ou até mesmo na geração de energia por meio de células de combustível. Há uma infinidade de tecnologias sendo desenvolvidas explorando o conceito da eletrólise. Espera-se que este trabalho possa servir de contribuição aos estudos dessa técnica.

Referências

[1] BARRET, Steven F.; PACK, Daniel J., *Microcontrollers Fundamentals for Engineers and Scientists*, 2006.

[2] BARROSO, Leônidas; BARROSO, Magali; CAMPOS, Frederico; CARVALHO, Márcio; MAIA, Miriam, *Cálculo Numérico*, 2ª edição, 1987.

[3] LEITE, Rogério E., *Glossário Telecomunicações e Informática*, *Babylon.*

[4] GOLDBERG, David, *Beginning Chemistry*, 2005.

[5] I^2C – Especificação do Barramento.
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

[6] Michaelis Moderno Dicionário da Língua Portuguesa.

[7] MYERS, Richard, *The Basics of Chemistry*, 2003.

[8] MICROCHIP, *PICmicro™ Mid-Range MCU Family Reference Manual.*
http://ww1.microchip.com/downloads/en/DeviceDoc/33023a.pdf

[9] Portal e-física – USP.
http://efisica.if.usp.br/eletricidade/basico/eletrolise/eletrolise/

[10] Portal Inovação Tecnológica.
http://www.inovacaotecnologica.com.br

[11] SANCHEZ, Julio; CANTON, Maria P., *Microcontroller Programming. The Microchip PIC®*, 2007.

[12] Wikipédia – A enciclopédia livre.
http://pt.wikipedia.org

[13] WILMSHURST, Tim, *Designing embedded systems with PIC microcontrollers: principles and applications*, 2007.

[14] ZÜTTEL, Andreas; BORGSCHULTE, Andreas; SCHLAPBACH, Louis, 2008, Hydrogen as a Future Energy Carrier.

Apêndice A – Dados dos Experimentos

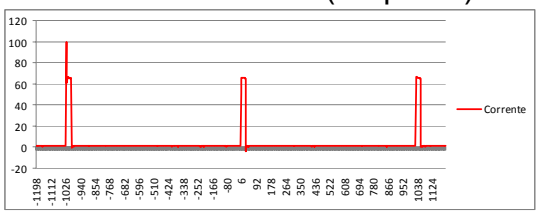
1. Experimento I

No experimento I a eletrólise foi feita em uma solução eletrolítica composta de 3 litros de água destilada adicionada de 2 ml de ácido sulfúrico. Neste experimento, a quantidade de pulsos gerados na sequência de pulsos foi fixada em apenas um pulso. O parâmetro do gerador de sinais variado foi a largura de pulso ligado. Os parâmetros relógio e pulso desligado ficaram estáticos assim com a quantidade de pulsos. As oito sessões de captura de dados com cada uma das variações do parâmetro estão detalhadas abaixo.

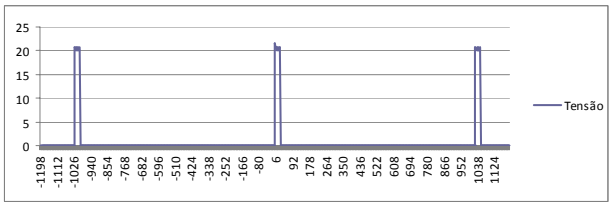
a. Experimento I – Sessão #1

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	10	
Pulso DESLIGADO	0	

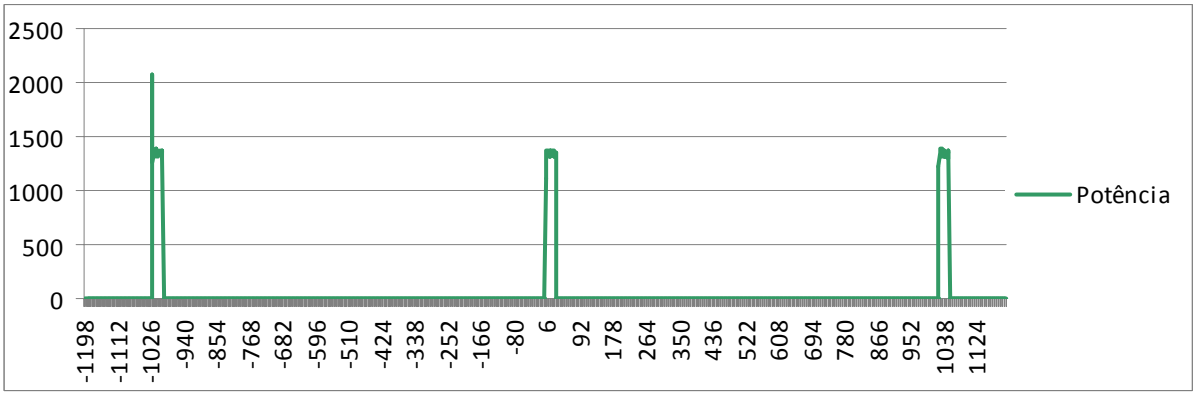
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,1160256**	

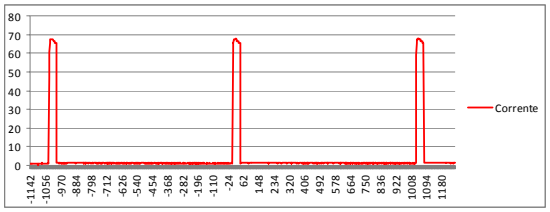
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	10,1	ml

Rendimento		
Potência Aplicada	**48,34**	W/s
Quantidade Produzida em W/s	**0,0035**	ml

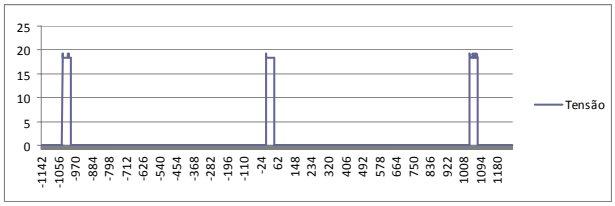
b. Experimento I – Sessão #2

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	15	
Pulso DESLIGADO	0	

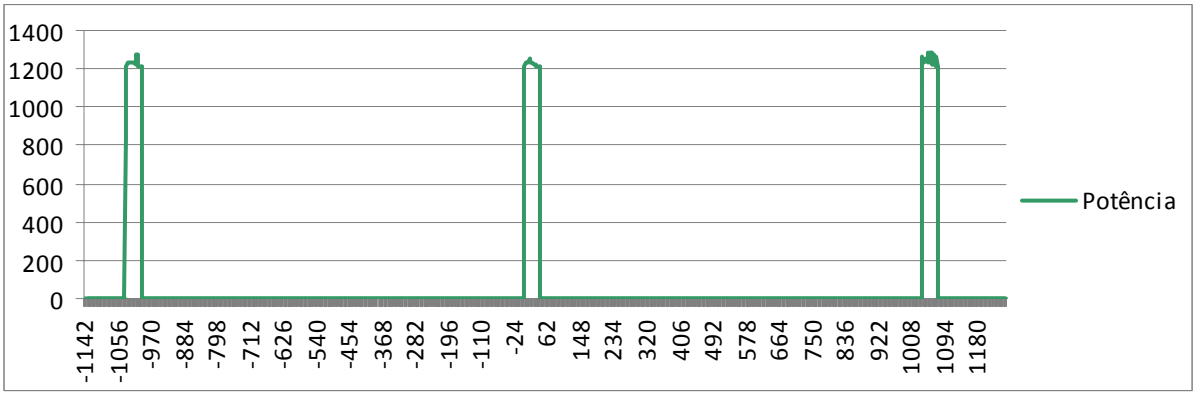
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,15680896**	

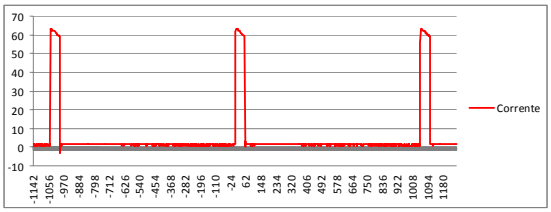
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	18	ml

Rendimento		
Potência Aplicada	**65,34**	W/s
Quantidade Produzida em W/s	**0,0046**	ml

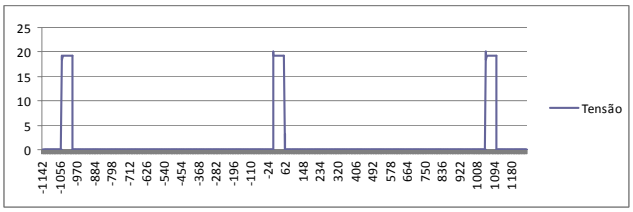
c. Experimento I – Sessão #3

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	20	
Pulso DESLIGADO	0	

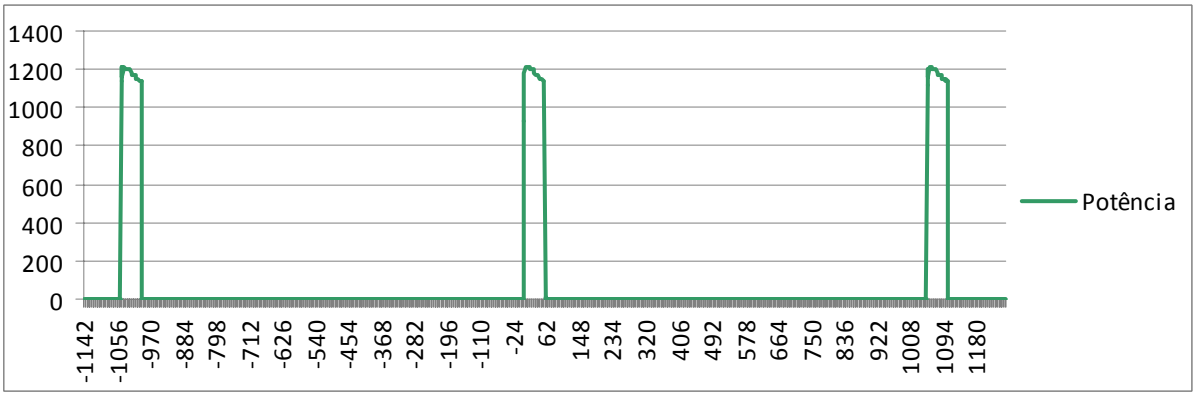
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,1947776**	

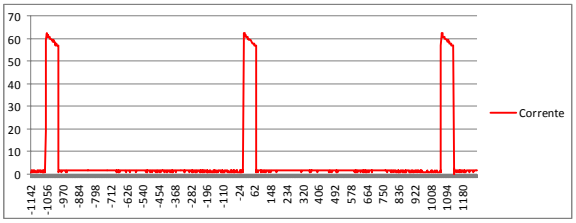
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	21,3	ml

Rendimento		
Potência Aplicada	**81,16**	W/s
Quantidade Produzida em W/s	**0,0044**	ml

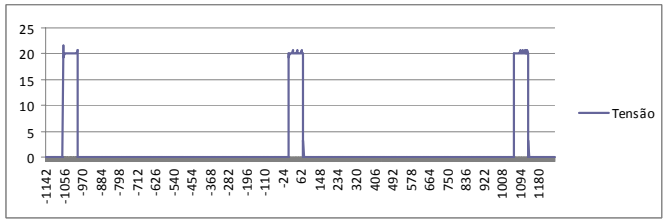
d. Experimento I – Sessão #4

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	25	
Pulso DESLIGADO	0	

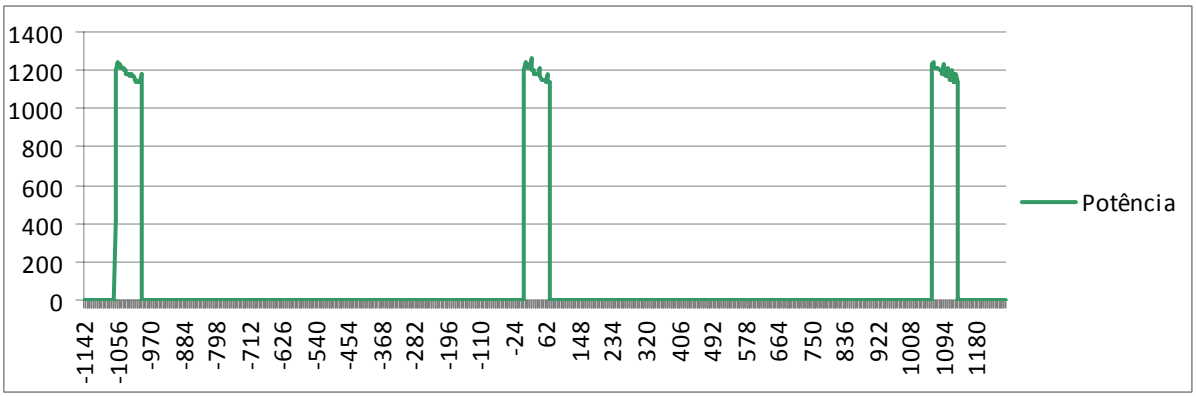
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,24279808**	

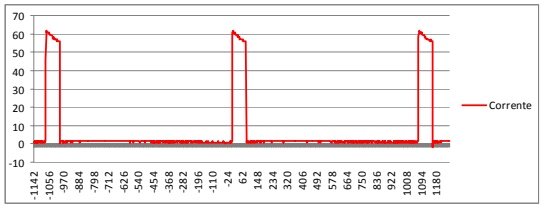
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	26,2	ml

Rendimento		
Potência Aplicada	**101,17**	W/s
Quantidade Produzida em W/s	**0,0043**	ml

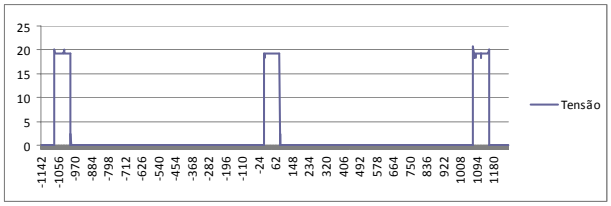
e. Experimento I – Sessão #5

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	30	
Pulso DESLIGADO	0	

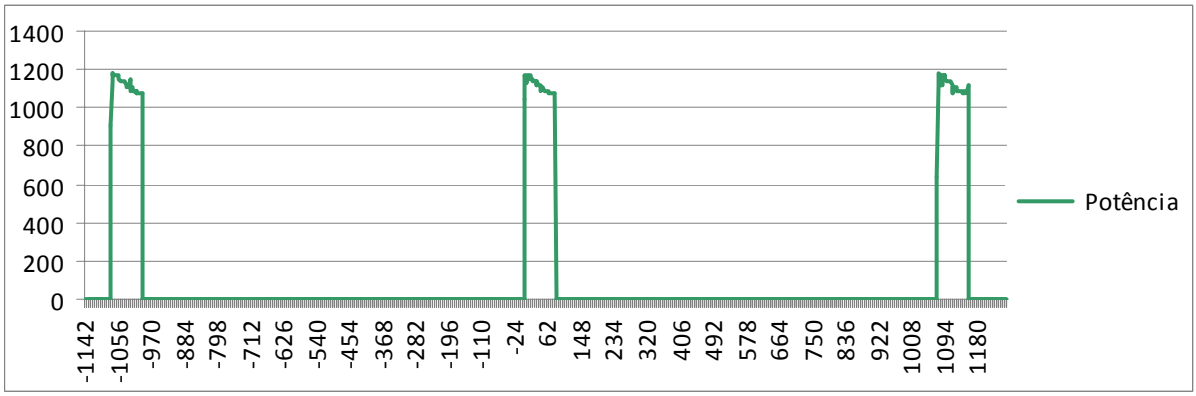
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,27329152**	

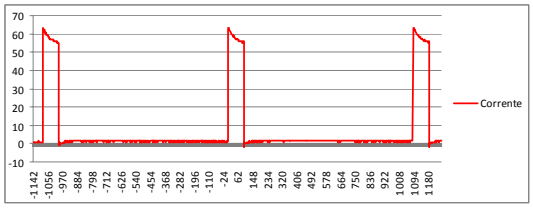
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	30,5	ml

Rendimento		
Potência Aplicada	**113,87**	W/s
Quantidade Produzida em W/s	**0,0045**	ml

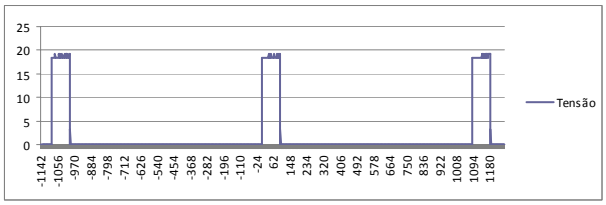
f. Experimento I – Sessão #6

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	35	
Pulso DESLIGADO	0	

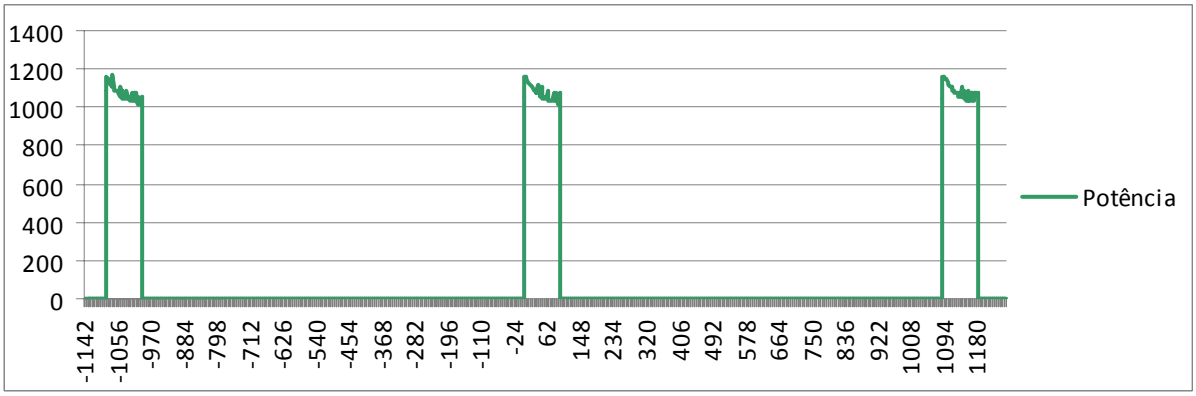
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,30428416**	

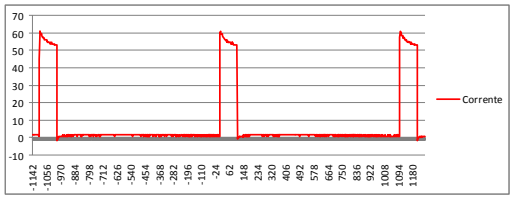
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	32,5	ml

Rendimento		
Potência Aplicada	**126,79**	W/s
Quantidade Produzida em W/s	**0,0043**	ml

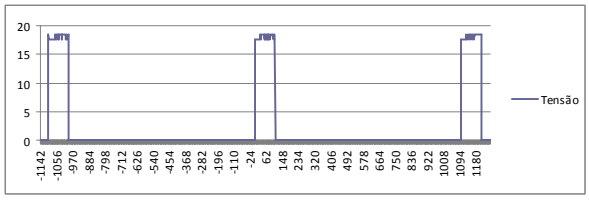
g. Experimento I – Sessão #7

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	40	
Pulso DESLIGADO	0	

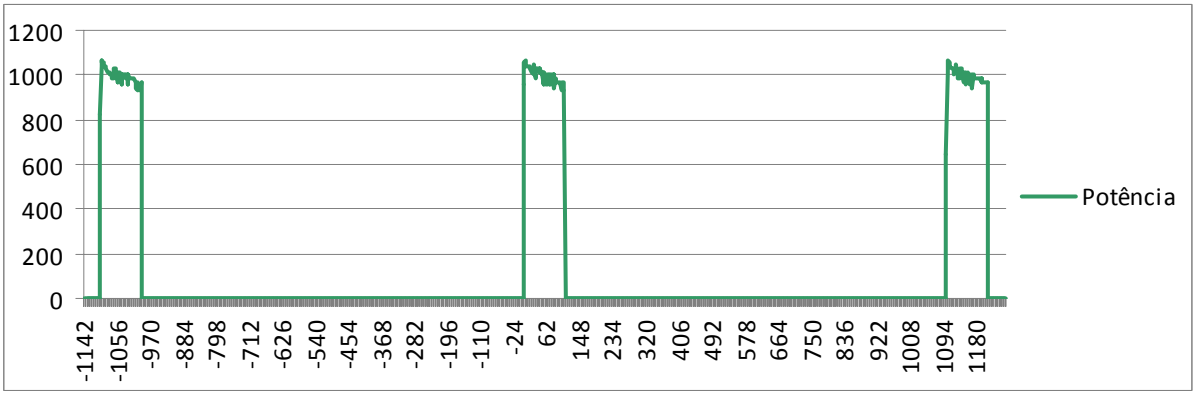
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,32238464**	

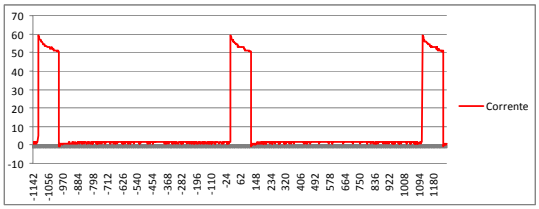
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	36,4	ml

Rendimento		
Potência Aplicada	**134,33**	W/s
Quantidade Produzida em W/s	**0,0045**	ml

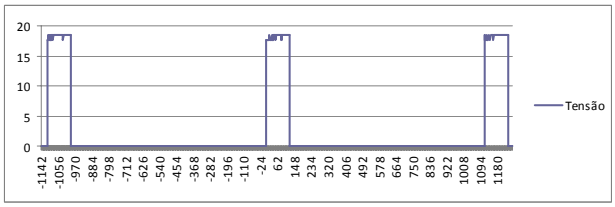
h. Experimento I – Sessão #8

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	999	µs
Quantidade de Pulsos	1	
Pulso LIGADO	45	
Pulso DESLIGADO	0	

Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)

Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000002	segundos
Tempo Total	0,0024	segundos
Integral	**0,3500992**	

Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	39,5	ml

Rendimento		
Potência Aplicada	**145,87**	W/s
Quantidade Produzida em W/s	**0,0045**	ml

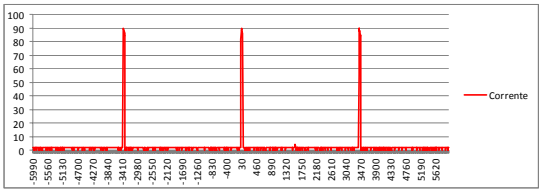
2. Experimento #2

No experimento II a eletrólise foi feita com a mesma solução eletrolítica do experimento I, isto é, em uma solução eletrolítica composta de 3 litros de água destilada adicionada de 2 ml de ácido sulfúrico. Neste experimento, o parâmetro que foi variado foi a quantidade de pulsos gerados na sequência de pulsos. Os parâmetros relógio, pulso ligado e pulso desligado ficaram estáticos. As oito sessões de captura de dados com cada uma das variações do parâmetro estão detalhadas abaixo.

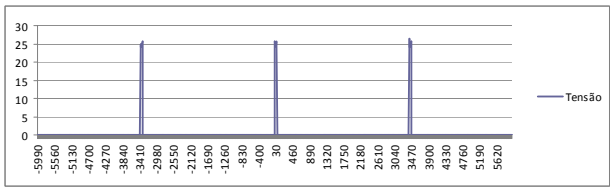
a. Experimento II – Sessão #1

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	1	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

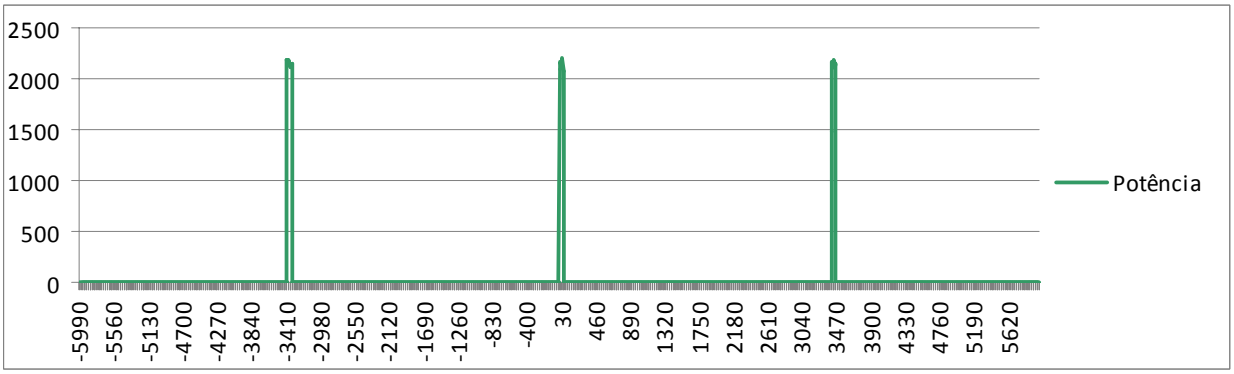
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**0,37904**	

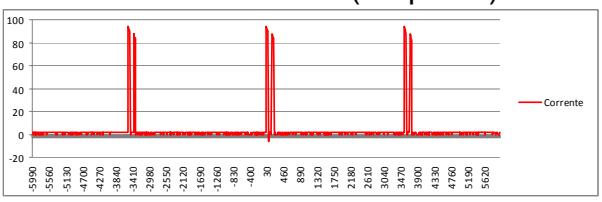
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	5,3	ml

Rendimento		
Potência Aplicada	**31,59**	W/s
Quantidade Produzida em W/s	**0,0028**	ml

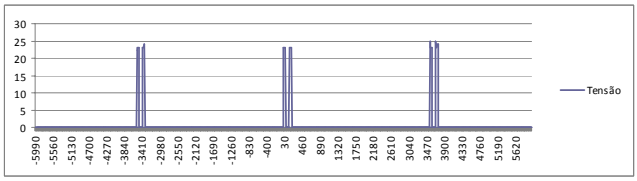
b. Experimento II – Sessão #2

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	2	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

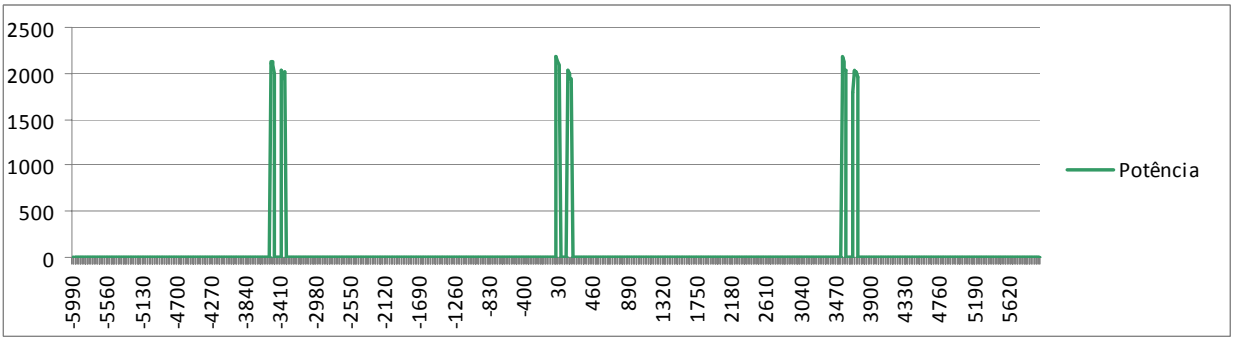
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**0,67272**	

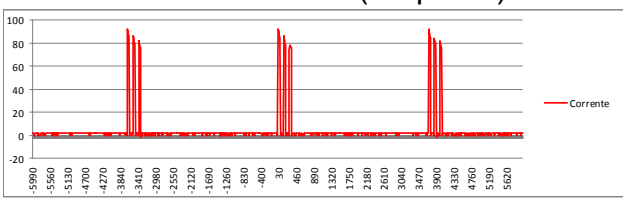
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	18	ml

Rendimento		
Potência Aplicada	**56,06**	W/s
Quantidade Produzida em W/s	**0,0054**	ml

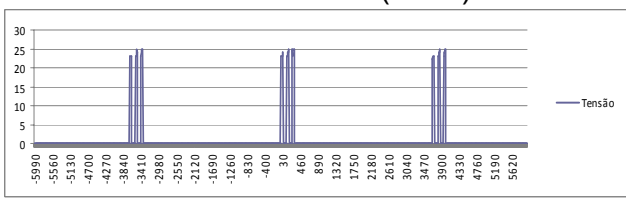
c. Experimento II – Sessão #3

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	3	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

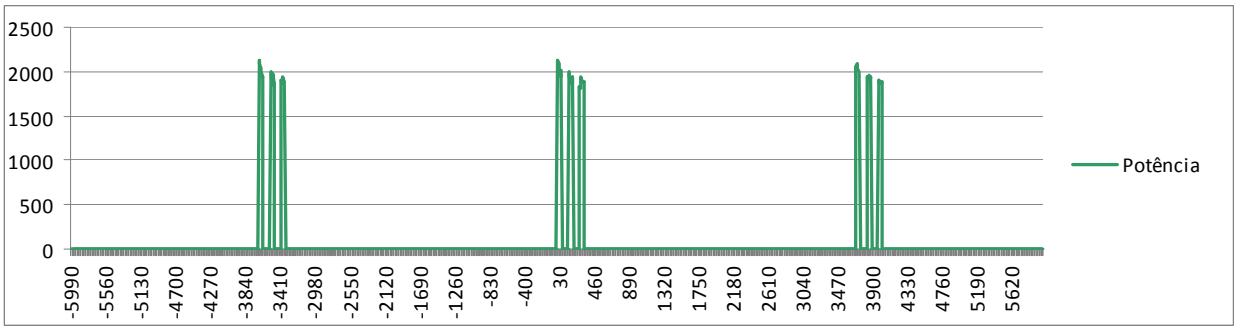
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**0,93688**	

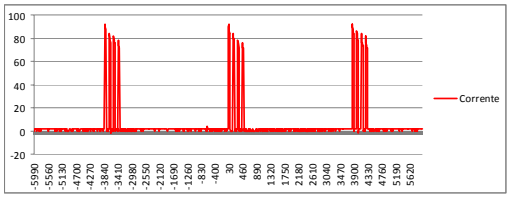
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	23,3	ml

Rendimento		
Potência Aplicada	**78,07**	W/s
Quantidade Produzida em W/s	**0,0050**	ml

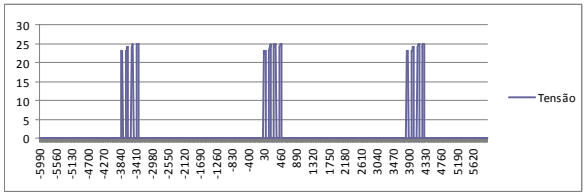
d. Experimento II – Sessão #4

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	4	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

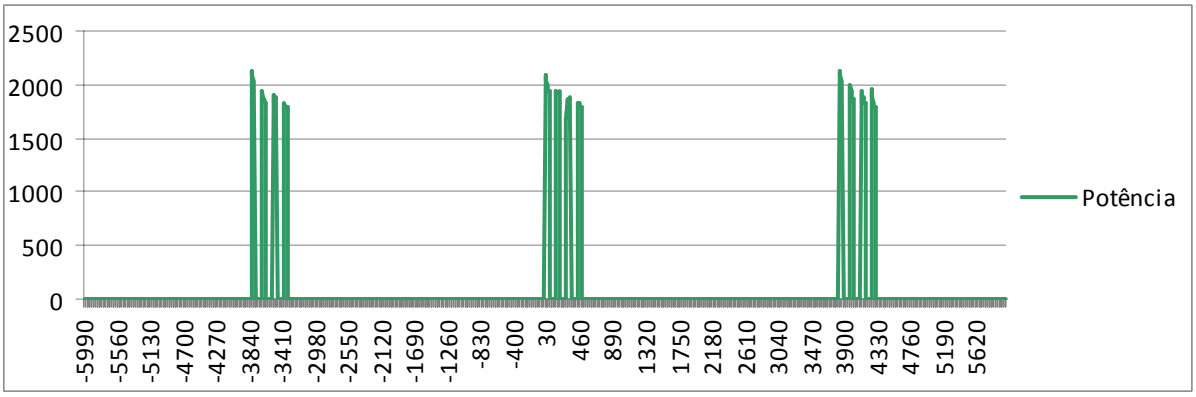
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**1,232736**	

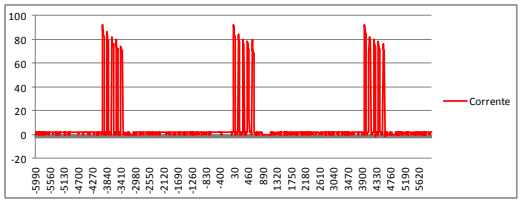
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	27,4	ml

Rendimento		
Potência Aplicada	**102,73**	W/s
Quantidade Produzida em W/s	**0,0044**	ml

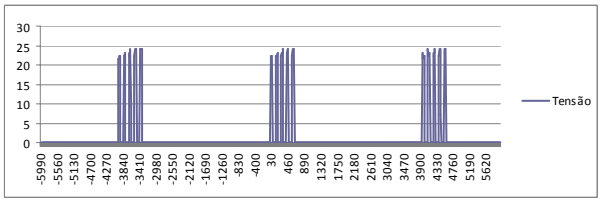
e. Experimento II – Sessão #5

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	5	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

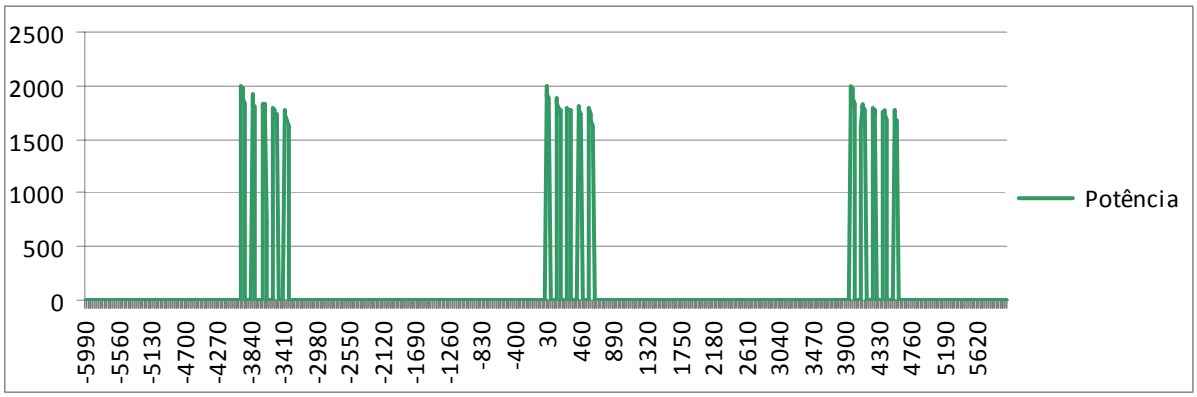
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**1,433904**	

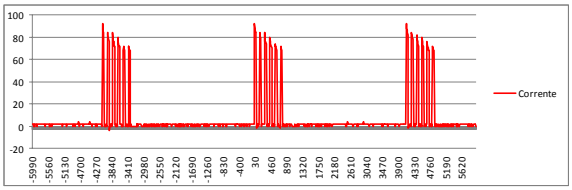
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	33,9	ml

Rendimento		
Potência Aplicada	**119,49**	W/s
Quantidade Produzida em W/s	**0,0047**	ml

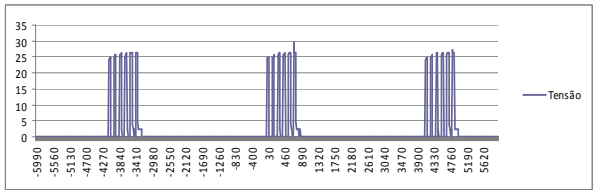
f. Experimento II – Sessão #6

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	6	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

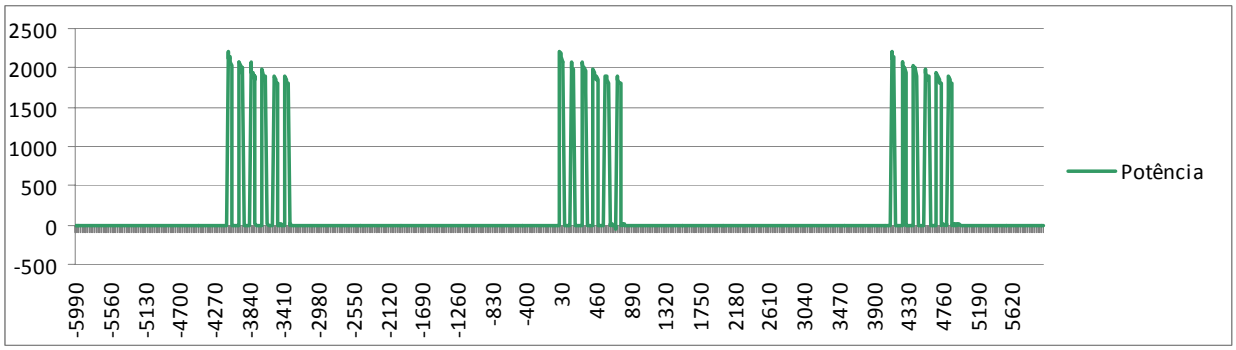
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**1,913136**	

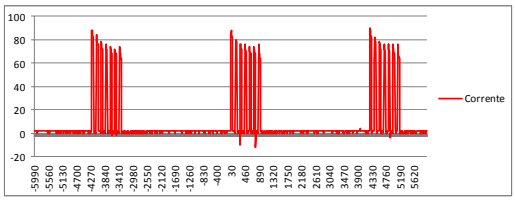
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	37,8	ml

Rendimento		
Potência Aplicada	**159,43**	W/s
Quantidade Produzida em W/s	**0,0040**	ml

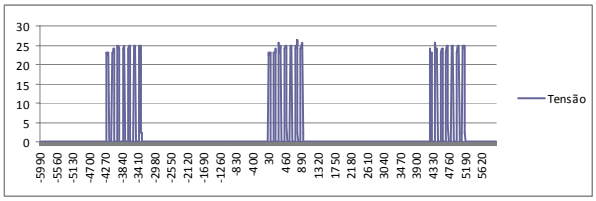
g. Experimento II – Sessão #7

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	7	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

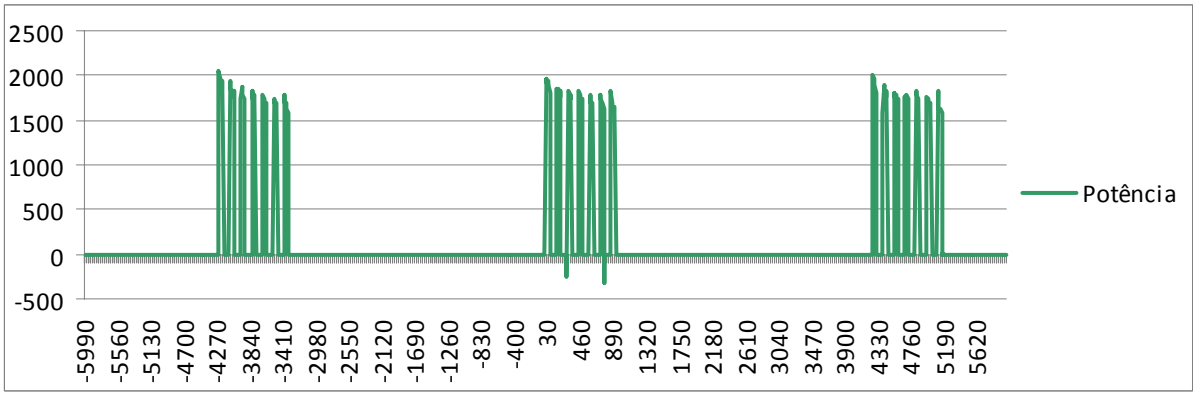
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**2,045584**	

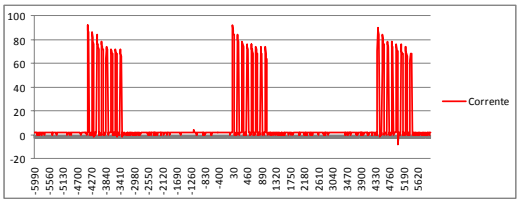
Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	41,4	ml

Rendimento		
Potência Aplicada	**170,47**	W/s
Quantidade Produzida em W/s	**0,0040**	ml

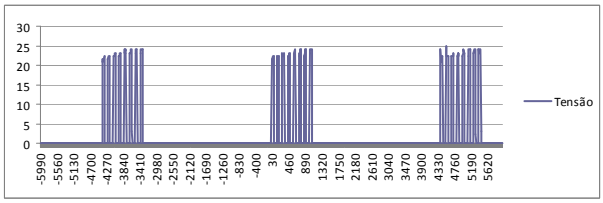
h. Experimento II – Sessão #8

Gerador de Sinais		
Parâmetro	**Valor**	**Unidade**
Relógio	3330	µs
Quantidade de Pulsos	8	
Pulso LIGADO	20	
Pulso DESLIGADO	30	

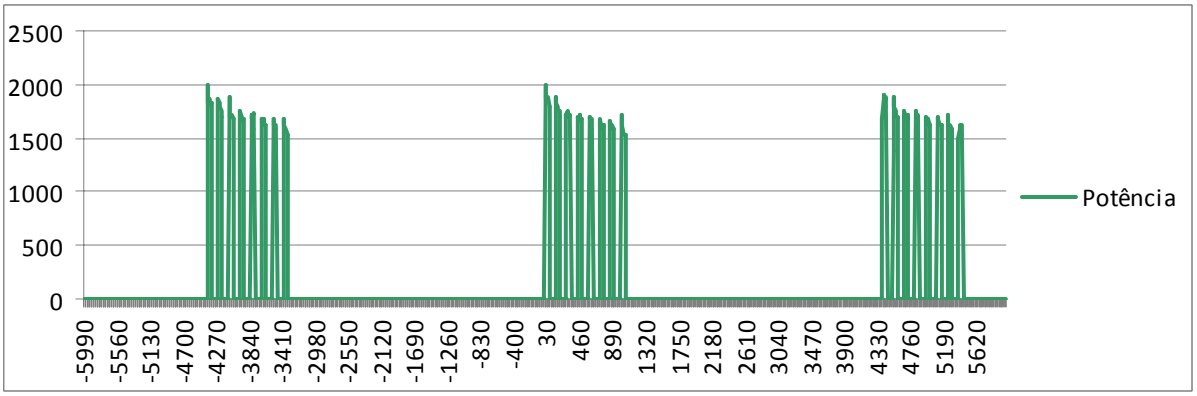
Sinal da Corrente (Ampéres)



Sinal da Tensão (Volts)



Sinal da Potência (Watts)



Cálculo de Integral da Potência		
Parâmetro	**Valor**	**Unidade**
Quantidade de Amostras	1200	
dx	0,000010	segundos
Tempo Total	0,012	segundos
Integral	**2,23432**	

Produção de Gás		
Parâmetro	**Valor**	**Unidade**
Tempo de Aplicação do Sinal	60	segundos
Quantidade Produzida	44,3	ml

Rendimento		
Potência Aplicada	**186,19**	W/s
Quantidade Produzida em W/s	**0,0040**	ml

Apêndice B – Código Fonte

1 – Código Fonte do Programa do **Módulo de Controle**

main.c

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#include <system.h>
#include "main.h"
#include "lcd.h"
#include "adc.h"
#include "i2c_com.h"
#include "util.h"

#pragma CLOCK_FREQ 20000000

#pragma DATA 0x2007, _HS_OSC & _WDT_OFF & _LVP_OFF & _PWRTE_ON & _DEBUG_OFF

void main()
{
      config();

      menuSelect = REF_MENU_PRINCIPAL;

      while(1)
      {
            switch(menuSelect)
            {
                  case REF_MENU_PRINCIPAL:           menuPrincipal();
            break;

                  case REF_MENU_GERADOR:             menuGerador();
            break;

                  case REF_MENU_GRELOGIO:            menuGRelogio();
            break;

                  case REF_MENU_GPULSO:              menuGPulse();
            break;

                  case REF_MENU_GPULSO_QTD:          menuGPulseQtd();
            break;

                  case REF_MENU_GPULSO_LIGADO:       menuGPulseOn();
            break;
                  case REF_MENU_GPULSO_DESLIG:  menuGPulseOff();
            break;
            }

            lcd_clearDisplay();
            delay_ms(1);
            clear_wdt();
```

```
	}
}

void interrupt(void) {

	if(l_rbif)  {

		l_rbif = 0;

		char temp = portb & 00110000b;
		new_val = temp >> 4;

		if(new_val != last_val) {

			if(new_val == 0) {
				if(last_val == 0x03 && roll < 10000) ++roll;
				else if (roll > 0) --roll;
			}
			else if(new_val == 0x03) {
				if(last_val == 0 && roll > 0) --roll;
				else if (roll < 10000) ++roll;
			}
			else {
				if(last_val == 0 && roll < 10000) ++roll;
				else if (roll > 0) --roll;
			}

			last_val = new_val;
		}
	}
}

void menuPrincipal() {

	lcd_printf(E_GERADOR, 1, 1);

	while(1) {

		lerTeclado(&tecla);

		if(tecla == CHAVE_ESQ) {
			menuSelect = REF_MENU_GERADOR;
			break;
		}

		delay_ms(10);
		clear_wdt();
	}
}

void menuGerador() {

	lcd_printf(E_RELOGIO, 1, 1);

	setaLinha = 1;

	while(1)
	{
		lerTeclado(&tecla);

		if(tecla == CHAVE_ESQ) {

			switch(setaLinha) {
				case 1: menuSelect = REF_MENU_GRELOGIO; break;
				case 2: menuSelect = REF_MENU_GPULSO;    break;
```

```
                }

                break;
            }
            else if(tecla == CHAVE_DIR) {
                if(setaLinha < 2) ++setaLinha;
                else setaLinha = 1;
                delay_ms(150);
                clear_wdt();
            }
            else if(tecla == CHAVE_MID) {
                menuSelect = REF_MENU_PRINCIPAL;
                break;
            }

            switch(setaLinha) {
                case 1:
                    lcd_printf(E_RELOGIO, 1, 1);
                    break;
                case 2:
                    lcd_setPos(1,1);
                    lcd_printf(E_PULSO, 1, 1);
                    break;
            }

            delay_ms(10);
            clear_wdt();
        }
}

void menuGRelogio() {

    lcd_printf(E_RELOGIO_DEF, 1, 1);
    lcd_printf(E_RELOGIO_DEF_C, 2, 1);

    gRelogio = lerEscravo(CMD_GET_RELOGIO);

    roll = gRelogio;
    rollerOn;

    char c = 0;
    while(1)
    {
        lerTeclado(&tecla);

        if(tecla == CHAVE_ESQ) {
            rollerOff;
            ledOn;
            sendMcpCommand(slaveAddress, CMD_SET_RELOGIO, gRelogio);
            ledOff;
            delay_ms(50);
            rollerOn;
        }
        else if(tecla == CHAVE_DIR) {
            rollerOff;
            menuSelect = REF_MENU_GERADOR;
            break;
        }

        if(++c == 10)
        {
            lcd_setPos(2,5);
            if(roll < GRELOGIO_MIN) roll = GRELOGIO_MIN;
            else if(roll > GRELOGIO_MAX) roll = GRELOGIO_MAX;
            gRelogio = roll;
```

```
                  unsigned long grelLong = (unsigned long) gRelogio+2; //
##### (+2) Calibração
                  grelLong = grelLong * 111;  // Calibração 111
                  grelLong = grelLong / 10;
                  unsigned short gRel = (unsigned short) grelLong;
                  lcd_print_int((gRel), 0, 5);
                  c = 0;
            }

            delay_ms(1);
            clear_wdt();
      }
}

void menuGPulse() {

      lcd_printf(E_PULSO_QTD, 1, 1);

      setaLinha = 1;

      while(1)
      {
            lerTeclado(&tecla);

            if(tecla == CHAVE_ESQ) {

                  switch(setaLinha) {
                        case 1: menuSelect = REF_MENU_GPULSO_QTD;break;
                        case 2: menuSelect = REF_MENU_GPULSO_LIGADO; break;
                        case 3: menuSelect = REF_MENU_GPULSO_DESLIG; break;
                  }

                  break;
            }
            else if(tecla == CHAVE_DIR) {
                  if(setaLinha < 3) ++setaLinha;
                  else setaLinha = 1;
                  delay_ms(150);
                  clear_wdt();
            }
            else if(tecla == CHAVE_MID) {
                  menuSelect = REF_MENU_GERADOR;
                  break;
            }


            switch(setaLinha) {
                  case 1:
                        lcd_printf(E_PULSO_QTD, 1, 1);
                        break;
                  case 2:
                        lcd_printf(E_PULSO_LIGADO, 1, 1);
                        break;
                  case 3:
                        lcd_printf(E_PULSO_DESLIG, 1, 1);
                        break;
            }

            delay_ms(10);
            clear_wdt();
      }
}

void menuGPulseQtd() {
```

```
	lcd_printf(E_PULSO_QTD_DEF, 1, 1);
	lcd_printf(E_VALOR, 2, 1);

	gPulseQtd = lerEscravo(CMD_GET_PULSO_QTD);

	roll = gPulseQtd;
	rollerOn;

	char c = 0;
	while(1)
	{
		lerTeclado(&tecla);

		if(tecla == CHAVE_ESQ) {
			rollerOff;
			ledOn;
			sendMcpCommand(slaveAddress, CMD_SET_PULSO_QTD, gPulseQtd);
			ledOff;
			delay_ms(50);
			rollerOn;
		}
		else if(tecla == CHAVE_DIR) {
			rollerOff;
			menuSelect = REF_MENU_GPULSO;
			break;
		}

		if(++c == 10)
		{
			lcd_setPos(2,8);
			gPulseQtd = roll;
			lcd_print_int(gPulseQtd, 0, 5);
			c = 0;
		}

		delay_ms(1);
		clear_wdt();
	}
}


void menuGPulseOn() {

	lcd_printf(E_PULSO_LIGADO_DEF, 1, 1);
	lcd_printf(E_VALOR, 2, 1);

	gPulseOn = lerEscravo(CMD_GET_PULSO_ON);

	roll = gPulseOn;
	rollerOn;

	char c = 0;
	while(1)
	{
		lerTeclado(&tecla);

		if(tecla == CHAVE_ESQ) {
			rollerOff;
			ledOn;
			sendMcpCommand(slaveAddress, CMD_SET_PULSO_ON, gPulseOn);
			ledOff;
			delay_ms(50);
			rollerOn;
		}
		else if(tecla == CHAVE_DIR) {
```

```
                rollerOff;
                menuSelect = REF_MENU_GPULSO;
                break;
            }

            if(++c == 10)
            {
                lcd_setPos(2,8);
                gPulseOn = roll;
                lcd_print_int(gPulseOn, 0, 5);
                c = 0;
            }

            delay_ms(1);
            clear_wdt();
        }
    }

    void menuGPulseOff() {

        lcd_printf(E_PULSO_DESLIG_DEF, 1, 1);
        lcd_printf(E_VALOR, 2, 1);

        gPulseOff = lerEscravo(CMD_GET_PULSO_OFF);

        roll = gPulseOff;
        rollerOn;

        char c = 0;
        while(1)
        {
            lerTeclado(&tecla);

            if(tecla == CHAVE_ESQ) {
                rollerOff;
                ledOn;
                sendMcpCommand(slaveAddress, CMD_SET_PULSO_OFF, gPulseOff);
                ledOff;
                delay_ms(50);
                rollerOn;
            }
            else if(tecla == CHAVE_DIR) {
                rollerOff;
                menuSelect = REF_MENU_GPULSO;
                break;
            }

            if(++c == 10)
            {
                lcd_setPos(2,8);
                gPulseOff = roll;
                lcd_print_int(gPulseOff, 0, 5);
                c = 0;
            }

            delay_ms(1);
            clear_wdt();
        }
    }

    void lerTeclado(char *data) {

        if          (botEsq == 0) *data = CHAVE_ESQ;
        else if     (botMid == 0) *data = CHAVE_MID;
        else if     (botDir == 0) *data = CHAVE_DIR;
```

```
	else *data = 0;
}


void printEspaco(char qtd)
{
	for(;qtd > 0; qtd--)
		lcd_print_char(' ');
}



void sendMcpCommand(unsigned char mcp, unsigned char command, unsigned short
param)
{
	txShortBuffer[0] = command;

	txShortBuffer[1] = (unsigned char) param;
	param = param >> 8;
	txShortBuffer[2] = (unsigned char) param;

	bp = txShortBuffer;
	writeStringMcp(mcp, bp, N_TX_SHORT_MX);
}

void sendMcpCommand(unsigned char mcp, unsigned char command, unsigned char
param1, unsigned char param2)
{
	txShortBuffer[0] = command;
	txShortBuffer[1] = param1;
	txShortBuffer[2] = param2;
	bp = txShortBuffer;
	writeStringMcp(mcp, bp, N_TX_SHORT_MX);
}

unsigned short lerEscravo(unsigned char command)
{
	sendMcpCommand(slaveAddress, command, 0, 0);
	delay_ms(5);
	readStringMcp(slaveAddress, rxShortBuffer, 2);
	charToShort(&shortTemp, rxShortBuffer[0], rxShortBuffer[1]);
	return shortTemp;
}

void config(void) {

	// Configura as portas
	trisa = 0x00;
	trisb = 0x00;
	trisc = 0x00;
	trisd = 0x00;
	trise = 0x00;

	porta = 0x00;
	portb = 0x00;
	portc = 0x00;
	portd = 0x00;
	porte = 0x00;

	// Configura o tipo de acesso a EEPROM para acesso a memória de dados
	clear_bit(eecon1, EEPGD);

	// Habilita pull-ups em PORTB
	//clear_bit(option_reg, NOT_RBPU);
```

```
	// Configura recursos periféricos
	//adc_init();
	lcd_init();
	i2c_init(0x31);

	// Configura os botões
	botDir_tris = 1;
	botMid_tris = 1;
	botEsq_tris = 1;

	// Configura o recurso interrupt-on-change em PORTB
	rb4_tris = 1;
	rb5_tris = 1;
	set_bit(intcon, GIE);
	clear_bit(intcon, RBIE);
	roll = 0;
}
```

1 – Código Fonte do Programa do **Módulo de Controle**

main.h

```
/**********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                      *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS   *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                               *
 *                                                                    *
 *    $Author: Rafael Chagas $                                        *
 *    $Date: 2009/09/29 22:07:00 $                                    *
 *    $Revision: 1.0 $                                                *
 *                                                                    *
 **********************************************************************/

#ifndef _MAIN_H_
#define _MAIN_H_

#include <system.h>

volatile bit led        @ PORTC.2;
#define ledOn     led = 1
#define ledOff    led = 0

#define CH1             1

// PORTB Interrupt-on-change
volatile bit l_rbif                 @     INTCON.RBIF;

volatile bit botDir                 @     PORTB.0;
volatile bit botDir_tris            @     TRISB.0;

volatile bit botMid                 @     PORTB.1;
volatile bit botMid_tris            @     TRISB.1;

volatile bit botEsq                 @     PORTB.2;
volatile bit botEsq_tris            @     TRISB.2;

volatile bit rb4                    @     PORTB.4;
volatile bit rb4_tris               @     TRISB.4;

volatile bit rb5                    @     PORTB.5;
volatile bit rb5_tris               @     TRISB.5;
```

```
// Recursos para o controle do roller
#define rollerOn        set_bit(intcon, RBIE)
#define rollerOff       clear_bit(intcon, RBIE)

unsigned char encoder;
unsigned char new_val;
unsigned char last_val;

#define     ROLL_MAX    10000
unsigned short roll;

/*********************************************************************/
/* Constantes e variáveis utilizadas na comunicação i2c entre os MCPs */
/*********************************************************************/
unsigned char mcpAddress;
#define slaveAddress          0x02 // 00000000

#define     N_RX_SHORT_MX     2
#define     N_TX_SHORT_MX     3
char *bp;
char rxShortBuffer[N_RX_SHORT_MX];
char txShortBuffer[N_TX_SHORT_MX];

/*********************************************************************/
/* Constantes que armazenam o endereço das frases gravadas na EEPROM */
/*********************************************************************/
//#define ADR_as_Cargas       0x00 // "as Cargas"


/*********************************************************************/
/* Constantes que armazenam as frase dos Menus que não estão na EEPROM. */
/*********************************************************************/
//const char* STR_CONFIG                = "Configuracoes";

/*******************************************************/
/* Constantes que armazenam as referências dos Menus. */
/*******************************************************/
#define REF_MENU_PRINCIPAL                                      0x00
#define REF_MENU_GERADOR                                        0x10
#define REF_MENU_GRELOGIO                                       0x11
#define REF_MENU_GPULSO                                         0x12
#define REF_MENU_GPULSO_QTD                                     0x13
#define REF_MENU_GPULSO_LIGADO                                  0x14
#define REF_MENU_GPULSO_DESLIG                                  0x15

// Expressões utilizadas no programa
//                                  "1234567890123456"
const char* E_GERADOR               = "    Gerador     ";

const char* E_RELOGIO               = "<   Relogio    >";
const char* E_RELOGIO_DEF           = "Def. Relogio    ";
const char* E_RELOGIO_DEF_C         = "T:          (us)";

const char* E_PULSO                 = "<    Pulso     >";
const char* E_PULSO_QTD             = "<  Pulso QTD   >";
const char* E_PULSO_QTD_DEF         = "Qtd. de pulsos  ";
const char* E_PULSO_LIGADO          = "< Pulso LIGADO >";
const char* E_PULSO_LIGADO_DEF      = "Per. Pulso LIG. ";
const char* E_PULSO_DESLIG          = "<  Pulso DESL. >";
const char* E_PULSO_DESLIG_DEF      = "Per. Pulso DESL.";

const char* E_VALOR                     = "Valor: ";

/*****************************************************/
/* Constantes que armazenam as referências das Teclas. */
```

```
/*******************************************************/
#define CHAVE_ESQ       1
#define CHAVE_MID       2
#define CHAVE_DIR       3


/*******************************/
/* Funções de Utilização Geral */
/*******************************/
void config(void);
unsigned short hexToVolt(unsigned short num);
void escreverDataEepromToLcd(char adr);
void escreverDataEeprom(char addr, char data);
void printEspaco(char qtd);

void menuPrincipal();

void menuGerador();
void menuGRelogio();
void menuGPulse();
void menuGPulseQtd();
void menuGPulseOn();
void menuGPulseOff();

void menuPotencia();

void lerTeclado(char *data);

void mostrarSinalRemoto();
void sendMcpCommand(unsigned char mcp, unsigned char command, unsigned short
param);
void sendMcpCommand(unsigned char mcp, unsigned char command, unsigned char
param1, unsigned char param2);
unsigned short lerEscravo(unsigned char command);

void setLed(char ch, bool set);
void efeitoU(char ch);
void efeitoD(char ch);
void efeito(char ch, bool b);

/*********************************/
/* Variáveis de Utilização Geral */
/*********************************/
unsigned short shortTemp;
char menuSelect;
char setaLinha;
char tecla;

// Recursos de controle do gerador
#define GRELOGIO_MIN    10
#define GRELOGIO_MAX    1000
unsigned short gRelogio;

#define GPULSO_MIN      1
#define GPULSO_MAX      1000
unsigned short gPulseQtd;
unsigned short gPulseOn;
unsigned short gPulseOff;


// Comandos disponíveis

#define CMD_SET_RELOGIO       0x10
#define CMD_GET_RELOGIO       0x11
```

```
#define CMD_SET_PULSO_QTD     0x20
#define CMD_GET_PULSO_QTD     0x21
#define CMD_SET_PULSO_ON      0x22
#define CMD_GET_PULSO_ON      0x23
#define CMD_SET_PULSO_OFF     0x24
#define CMD_GET_PULSO_OFF     0x25


#endif // _MAIN_H_
```

1 – Código Fonte do Programa do **Módulo de Controle**

util.c

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#include <system.h>
#include "util.h"

void charToShort(unsigned short *dst, char lobyte, char hibyte)
{
      *dst = (unsigned short)hibyte;
      *dst = *dst << 8;
      *dst |= lobyte;
}

void delaySSwdt(char seg)
{
      int i=0, j;
      for(; i<seg; i++)
      {
            for(j=0; j<4; j++)
            {
                  delay_ms(250);
                  clear_wdt();
            }
      }
}
```

1 – Código Fonte do Programa do **Módulo de Controle**

util.h

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#ifndef _UTIL_H_
#define _UTIL_H_

void charToShort(unsigned short *dst, char lobyte, char hibyte);
void delaySSwdt(char seg);

#endif // _UTIL_H_
```

1 – Código Fonte do Programa do **Módulo de Controle**

lcd.c

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#include "lcd.h"

// Local variable
bit lcd_found = 0;

//////////////////////////////////////////////////////////
// Drive a certain nibble on the LCD data pins, the nibble
// is taken from the lsb's of the input char
//////////////////////////////////////////////////////////
void lcd_clock_nibble(char nibble){

      // Since there seems to be no way to extract a bit from a byte in BoostC
-> do it in asm.
      // It is replicated 4 times here, since a function cannot return a bit
eiter.

      // Set the d4 depending on the state of nibble bit 0
      asm {
      btfss _nibble, 0
      goto  clear_lcd4
      bsf     _lcd_d4, F
      goto  done_lcd4
```

```
      clear_lcd4:
      bcf     _lcd_d4, F
      done_lcd4:
      }

      // Set the d5 depending on the state of nibble bit 1
      asm {
      btfss _nibble, 1
      goto  clear_lcd5
      bsf     _lcd_d5, F
      goto  done_lcd5
      clear_lcd5:
      bcf     _lcd_d5, F
      done_lcd5:
      }

      // Set the d6 depending on the state of nibble bit 2
      asm {
      btfss _nibble, 2
      goto  clear_lcd6
      bsf     _lcd_d6, F
      goto  done_lcd6
      clear_lcd6:
      bcf     _lcd_d6, F
      done_lcd6:
      }

      // Set the d7 depending on the state of nibble bit 3
      asm {
      btfss _nibble, 3
      goto  clear_lcd7
      bsf     _lcd_d7, F
      goto  done_lcd7
      clear_lcd7:
      bcf     _lcd_d7, F
      done_lcd7:
      }

      lcd_toggle_e;

      return;
}

////////////////////////////////////////////////////////////
// Read the busy flag of the LCD and wait to return until
// the LCD is ready.
// Skipped when running in debug mode.
////////////////////////////////////////////////////////////
void lcd_wait_busy(){

      // Skip this function if we are running in debug mode.
#ifdef DEBUG
      return;
#endif

      // If we have not detected an LCD in a previous call to this function,
      // then we just skip it this time, since else the program will hang
later
      // on in this function.
      if (!lcd_found){
            return;
      }

      // Save the current LCD mode
      bit mode = lcd_rs;
```

```
	// The data lines are inputs
	lcd_tris_d4 = 1;
	lcd_tris_d5 = 1;
	lcd_tris_d6 = 1;
	lcd_tris_d7 = 1;

	// Enter command mode
	lcd_cmd_mode;
	lcd_rw      = 1;

	// Put lcd_e high
	lcd_e_hi;

	char counter = 0;

	// Wait for completion of the operation, with a timeout of ~.5 seconds
	// LCD d7 is high if the operation is complete.
	while (lcd_d7 && counter < 0xFF){
		lcd_e_lo;
		lcd_e_hi;
		lcd_e_lo;
		lcd_e_hi;
		delay_ms(2);
		counter++;
	}

	// Check if the previous loop timed out
	if (counter == 0xFF) {
		// If it was a timeout -> disable the flag.
		lcd_found = 0;
	}

	// And put the lcd_lo again
	lcd_e_lo;

	// Restore the TRIS
	lcd_tris_d4 = 0;
	lcd_tris_d5 = 0;
	lcd_tris_d6 = 0;
	lcd_tris_d7 = 0;

	// Restore LCD RS mode.
	lcd_rs = mode;
	lcd_rw = 0;

	return;
}

//////////////////////////////////////////////////////////
// Send a byte to the LCD. Don't forget to set the mode
// (data or cmd) before calling this function.
//////////////////////////////////////////////////////////
void lcd_send_byte(char data){

	char temp = data >> 4;

	lcd_rw = 0; // Write mode

	// Clock the high nibble
	lcd_clock_nibble(temp);

	// Clock the low nibble
	lcd_clock_nibble(data);
```

```
	// Wait until the LCD is finished
	lcd_wait_busy();

}

//////////////////////////////////////////////////////////
// Initialisation sequence of the LCD
//////////////////////////////////////////////////////////
void lcd_init(){

	// Wait for internal reset
	delay_ms(15);

	// Set all pins as output
	lcd_tris_rs = 0;
	lcd_tris_rw = 0;
	lcd_tris_e  = 0;
	lcd_tris_d4 = 0;
	lcd_tris_d5 = 0;
	lcd_tris_d6 = 0;
	lcd_tris_d7 = 0;

	lcd_rw = 0;
	lcd_cmd_mode;

	// Init sequence (see datasheet)
	/// attention (clock 0x03 twice)
	lcd_clock_nibble(0x03);
	delay_ms(5);
	lcd_toggle_e;
	delay_10us(15);
	//delay_ms(1);
	/// 4bit mode (clock 0x03 and 0x02)
	lcd_toggle_e;
	delay_ms(5);
	lcd_d4 = 0;
	lcd_toggle_e;
	delay_10us(15);
	//delay_ms(1);

	// Assume we have an LCD attached
	lcd_found = 1;

	/// We're in 4 bit mode now, program general settings
	lcd_send_byte(FUNCTION_SET);
	lcd_send_byte(DISP_OFF);
	lcd_send_byte(DISP_ON);
	lcd_send_byte(ENTRY_INC);
	lcd_send_byte(DISP_CLR);
	lcd_send_byte(LINE1);

	// Init done, you can start writing characters now
	lcd_data_mode;

	return;

}

//////////////////////////////////////////////////////////
// Send a command to the LCD
//////////////////////////////////////////////////////////
void lcd_send_cmd(char command){
	lcd_cmd_mode;
	lcd_send_byte(command);
}
```

```
//////////////////////////////////////////////////////////
// Send a line to the LCD
//////////////////////////////////////////////////////////
void lcd_printf(const char* line){
    char i = 0;
    lcd_data_mode;
    while(line[i] != 0)
        lcd_send_byte(line[i++]);
}

// Send a line to the LCD on position defined by row and col.
void lcd_printf(const char* line, char row, char col)
{
      lcd_setPos(row,col);
      lcd_printf(line);
}

//////////////////////////////////////////////////////////
// Print a character to the LCD
//////////////////////////////////////////////////////////
void lcd_print_char(char value){
      lcd_data_mode;
      lcd_send_byte(value);
}


//////////////////////////////////////////////////////////
// Print a hex value to the LCD
//////////////////////////////////////////////////////////
void lcd_print_hex(char value){
      lcd_data_mode;

      char hexChar;
    char i;

    for(i = 0; i < 2; i++)
    {
        if(i == 0)
            hexChar = value >> 4;
        else
            hexChar = value & 0x0F;
        if(hexChar < 10)
            hexChar = hexChar + '0';
        else
            hexChar = hexChar + ('A' - 10);
        lcd_send_byte(hexChar);
    }
}

//////////////////////////////////////////////////////////
// Print a 16-bit hex value to the LCD
//////////////////////////////////////////////////////////
void lcd_print_hex_s(short value){
      lcd_data_mode;

      char hexChar;
    char i;

    char value1 = (char)((value >> 8) & 0x00FF);

    for(i = 0; i < 2; i++)
    {
        if(i == 0)
            hexChar = value1 >> 4;
```

```
        else
            hexChar = value1 & 0x0F;
        if(hexChar < 10)
            hexChar = hexChar + '0';
        else
            hexChar = hexChar + ('A' - 10);
        lcd_send_byte(hexChar);
    }

    char value0 = (char)(value & 0x00FF);

    for(i = 0; i < 2; i++)
    {
        if(i == 0)
            hexChar = value0 >> 4;
        else
            hexChar = value0 & 0x0F;
        if(hexChar < 10)
            hexChar = hexChar + '0';
        else
            hexChar = hexChar + ('A' - 10);
        lcd_send_byte(hexChar);
    }
}

void lcd_setPos(char linha, char pos)
{
      if(linha<1 || linha>4 || pos<1 || pos>20) return;
      else
      {
            pos -= 1;

            if(linha == 1) lcd_send_cmd(0x80 + pos);
            else if(linha == 2) lcd_send_cmd(0x80 + 0x40 + pos);
            else if(linha == 3) lcd_send_cmd(0x80 + 0x14 + pos);
            else lcd_send_cmd(0x80 + 0x54 + pos);

            delay_ms(1);
      }
}

void lcd_clearDisplay()
{
      char i;
      lcd_setPos(1,1);
      for(i=0; i<80; i++)
      {
            lcd_print_char(' ');
      }
}

void lcd_print_int(unsigned short num, char pt, char qtdChar)
{
      char nn = 0;
      unsigned short shortValue;
      bool zeroEsq = true;

      shortValue = num / 0x2710;
      nn = verificaNShort(&shortValue, nn);
      if(nn) shortValue == 0 ? lcd_print_char(' ') : lcd_print_char( '0' +
shortValue);
      if(pt == 1) lcd_print_char('.');
      if(shortValue != 0) zeroEsq = false;

      if(qtdChar > 1)
```

```
	{
		shortValue = (num / 0x03E8) % 0x000A;
		nn = verificaNShort(&shortValue, nn);
		if(nn) {
			if(zeroEsq && shortValue == 0) lcd_print_char(' ');
			else lcd_print_char('0' + shortValue);
		}
		if(pt == 2) lcd_print_char('.');
		if(zeroEsq && shortValue != 0) zeroEsq = false;
	}

	if(qtdChar > 2)
	{
		shortValue = (num / 0x0064) % 0x000A;
		nn = verificaNShort(&shortValue, nn);
		if(nn) {
			if(zeroEsq && shortValue == 0) lcd_print_char(' ');
			else lcd_print_char('0' + shortValue);
		}
		if(pt == 3) lcd_print_char('.');
		if(zeroEsq && shortValue != 0) zeroEsq = false;
	}

	if(qtdChar > 3)
	{
		shortValue = (num / 0x000A) % 0x000A;
		nn = verificaNShort(&shortValue, nn);
		if(nn) {
			if(zeroEsq && shortValue == 0) lcd_print_char(' ');
			else lcd_print_char('0' + shortValue);
		}
		if(pt == 4) lcd_print_char('.');
		if(zeroEsq && shortValue != 0) zeroEsq = false;
	}

	if(qtdChar > 4)
	{
		shortValue = num % 0x000A;
		nn = verificaNShort(&shortValue, nn);
		if(nn) lcd_print_char( '0' + shortValue);
	}
}

void lcd_print_long(long num, char pt)
{
	char nn = 0;
	long longValue;

	/*
	long longValue = num / 0x3B9ACA00;
	if((longValue == 0 && pt == 1) || longValue > 0) lcd_print_char( '0' +
longValue);
	if(pt == 1) lcd_print_char('.');
	*/

	longValue = ((num / 0x05F5E100) % 0x000A);
	nn = verificaNLong(&longValue, nn);
	if(nn) lcd_print_char( '0' + longValue);
	if(pt == 1) lcd_print_char('.');

	longValue = ((num / 0x00989680) % 0x000A);
	nn = verificaNLong(&longValue, nn);
	if(nn) lcd_print_char( '0' + longValue);
	if(pt == 2) lcd_print_char('.');
```

```
        longValue = ((num / 0x000F4240) % 0x000A);
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);
        if(pt == 3) lcd_print_char('.');

        longValue = ((num / 0x000186A0) % 0x000A);
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);
        if(pt == 4) lcd_print_char('.');

        longValue = ((num / 0x00002710) % 0x000A);
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);
        if(pt == 5) lcd_print_char('.');

        longValue = ((num / 0x000003E8) % 0x000A);
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);
        if(pt == 6) lcd_print_char('.');

        longValue = ((num / 0x00000064) % 0x000A);
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);
        if(pt == 7) lcd_print_char('.');

        longValue = ((num / 0x0000000A) % 0x000A);
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);
        if(pt == 8) lcd_print_char('.');

        longValue = num % 0x0000000A;
        nn = verificaNLong(&longValue, nn);
        if(nn) lcd_print_char( '0' + longValue);

}

char verificaNLong(long *longValue, char nn)
{
        if(nn) return true;
        else
        {
                if(*longValue > 0) return true;
                else return false;
        }
}

char verificaNShort(unsigned short *shortValue, char nn)
{
        if(nn) return true;
        else
        {
                if(shortValue > 0) return true;
                else return false;
        }
}
```

1 – Código Fonte do Programa do **Módulo de Controle**

lcd.h

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#ifndef _LCD_H_
#define _LCD_H_

#include <system.h>

// Define the LCD connections here
#define LCD_E_PORT                  PORTD
#define LCD_RS_PORT                 PORTD
#define LCD_RW_PORT                 PORTD
#define LCD_DATA4_PORT        PORTD
#define LCD_DATA5_PORT        PORTD
#define LCD_DATA6_PORT        PORTD
#define LCD_DATA7_PORT        PORTD

#define LCD_E_TRIS                  TRISD
#define LCD_RS_TRIS                 TRISD
#define LCD_RW_TRIS                 TRISD
#define LCD_DATA4_TRIS        TRISD
#define LCD_DATA5_TRIS        TRISD
#define LCD_DATA6_TRIS        TRISD
#define LCD_DATA7_TRIS        TRISD

#define LCD_E_PIN             2
#define LCD_RS_PIN                  0
#define LCD_RW_PIN                  1
#define LCD_DATA4_PIN         4
#define LCD_DATA5_PIN         5
#define LCD_DATA6_PIN         6
#define LCD_DATA7_PIN         7

////////////////////////////////////////////////////////////////
// Don't change below this line
volatile bit lcd_e                  @ LCD_E_PORT     . LCD_E_PIN;
volatile bit lcd_rs                 @ LCD_RS_PORT    . LCD_RS_PIN;
volatile bit lcd_rw                 @ LCD_RW_PORT    . LCD_RW_PIN;
volatile bit lcd_d4                 @ LCD_DATA4_PORT . LCD_DATA4_PIN;
volatile bit lcd_d5                 @ LCD_DATA5_PORT . LCD_DATA5_PIN;
volatile bit lcd_d6                 @ LCD_DATA6_PORT . LCD_DATA6_PIN;
volatile bit lcd_d7                 @ LCD_DATA7_PORT . LCD_DATA7_PIN;

volatile bit lcd_tris_e       @ LCD_E_TRIS     . LCD_E_PIN;
volatile bit lcd_tris_rs      @ LCD_RS_TRIS    . LCD_RS_PIN;
volatile bit lcd_tris_rw      @ LCD_RW_TRIS    . LCD_RW_PIN;

bit lcd_tris_d4   @ LCD_DATA4_TRIS . LCD_DATA4_PIN;
bit lcd_tris_d5   @ LCD_DATA5_TRIS . LCD_DATA5_PIN;
bit lcd_tris_d6   @ LCD_DATA6_TRIS . LCD_DATA6_PIN;
bit lcd_tris_d7   @ LCD_DATA7_TRIS . LCD_DATA7_PIN;
```

```
// LCD command set
#define     LINE1             0x80        // set display to line 1 character 0
#define     LINE2           0xC0    // set display to line 2 character 0
#define     FUNCTION_SET      0x28        // 4 bits, 2 lines, 5x7 Font
#define     DISP_ON         0x0C    // display on
#define     DISP_ON_C       0x0E    // display on, Cursor on
#define     DISP_ON_B       0x0F    // display on, Cursor on, Blink cursor
#define     DISP_OFF        0x08    // display off
#define     DISP_CLR        0x01    // clear the Display
#define     ENTRY_INC       0x06    // increment-mode, display shift OFF
#define     ENTRY_INC_S     0x07    // increment-mode, display shift ON
#define     ENTRY_DEC       0x04    // decrement-mode, display shift OFF
#define     ENTRY_DEC_S     0x05    // decrement-mode, display shift ON
#define     DD_RAM_ADDR     0x80    // Least Significant 7-bit are for address

// Function definitions
void lcd_init();
void lcd_send_byte(char data);
void lcd_data_mode();
void lcd_cmd_mode();
void lcd_send_cmd(char command);
void lcd_printf(const char* line);
void lcd_printf(const char* line, char row, char col);
void lcd_print_char(char value);
//void lcd_print_char_on(const value, char row, char col);
void lcd_print_hex(char value);
void lcd_print_hex_s(short value);
void lcd_setPos(char linha, char pos);

void lcd_clearDisplay();

void lcd_print_int(unsigned short num, char pt, char qtdChar);
void lcd_print_long(long num, char pt);
char verificaNLong(long *longValue, char nn);
char verificaNShort(unsigned short *shortValue, char nn);

// Macro's
#define lcd_data_mode lcd_rs = 1
#define lcd_cmd_mode  lcd_rs = 0
#define lcd_e_hi    lcd_e  = 1
#define lcd_e_lo    lcd_e  = 0
#define lcd_toggle_e  lcd_e_hi ; lcd_e_lo

#endif // _LCD_H_
```

1 – Código Fonte do Programa do **Módulo de Controle**

i2c_com.h

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#ifndef _I2C_COM_H_
#define _I2C_COM_H_

#include <system.h>
#include "i2c_drv.h"

void writeCharMcp(unsigned char mcp, char data);
void readCharMcp(unsigned char mcp, char *data);
void writeStringMcp(unsigned char mcp, char *s, char cc);
void readStringMcp(unsigned char mcp, char *data, char cc);

#endif // _I2C_COM_H_
```

1 – Código Fonte do Programa do **Módulo de Controle**

i2c_drv.c

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#ifndef _I2C_DRV_H_
#define _I2C_DRV_H_

#include <system.h>

volatile bit ledVm @ PORTC.1;

////////////////////////////////////////////////////////////////////
// i2c hardwareware implementation template arguments
////////////////////////////////////////////////////////////////////
#define i2c_ARGS  e_SCL_BIT, PORTC, TRISC, e_SDA_BIT, PORTC, TRISC, e_SSPCON1,
e_SSPCON2, \
                            e_SSPSTAT, e_SSPBUF, e_SSPIF_BIT, e_SSPIF_PIR,
           \
                            e_BCLIF_BIT, e_BCLIF_PIR, e_SMP_BIT, e_SSPADD,
(i2c_reset_wdt | i2c_SMP | i2c_HW | i2c_400KHz)
```

```
// variables cannot be passed as template arguments. The following constants
map to
// the PIC registers and PIC's i2c register locations. These constants are
// then used by the templated functions.

// Todas as variáveis abaixo foram alteradas para o PIC16F877
#define PORTC           0x0007
#define TRISC           0x0087
#define e_SSPCON1 0x0014
#define e_SSPCON2 0x0091
#define e_SSPSTAT 0x0094
#define e_SSPADD  0x0093
#define e_SSPBUF  0x0013
#define e_SSPIF_PIR     0x000c
#define e_BCLIF_PIR     0x000d
#define e_SSPIF_BIT     3
#define e_BCLIF_BIT     3
#define e_SCL_BIT 3
#define e_SDA_BIT 4
#define e_SMP_BIT 7


////////////////////////////////////////////////////////////////////////
// Define the common i2c template structure
////////////////////////////////////////////////////////////////////////
#define _I2C_TEMPL      template <unsigned char T_SCL_BIT, unsigned short
T_SCL_PORT, \
                                unsigned short T_SCL_TRIS, unsigned char
T_SDA_BIT,              \
                                unsigned short T_SDA_PORT, unsigned short
T_SDA_TRIS,             \
                                unsigned short T_i2c_SSPCON1, unsigned short
T_i2c_SSPCON2,    \
                                unsigned short T_i2c_SSPSTAT, unsigned short
T_i2c_SSPBUF,           \
                                unsigned char T_i2c_SSPIF_BIT, unsigned short
T_i2c_SSPIF_PIR,  \
                                unsigned char T_i2c_BCLIF_BIT, unsigned short
T_i2c_BCLIF_PIR,  \
                                unsigned char T_i2c_SMP_BIT, unsigned short
T_i2c_SSPADD,           \
                                unsigned char T_MODE>

////////////////////////////////////////////////////////////////////////
// Define the common i2c template parameters
////////////////////////////////////////////////////////////////////////
#define _I2C_TEMPL_ARGS T_SCL_BIT, T_SCL_PORT, T_SCL_TRIS, T_SDA_BIT,
      \
                                T_SDA_PORT, T_SDA_TRIS, T_i2c_SSPCON1,
T_i2c_SSPCON2,    \
                                T_i2c_SSPSTAT, T_i2c_SSPBUF,
T_i2c_SSPIF_BIT,              \
                                T_i2c_SSPIF_PIR, T_i2c_BCLIF_BIT,
T_i2c_BCLIF_PIR,        \
                                T_i2c_SMP_BIT, T_i2c_SSPADD, T_MODE

////////////////////////////////////////////////////////////////////////
// Helpers that hide template arguments
////////////////////////////////////////////////////////////////////////
#define i2c_init  i2c_INIT<i2c_ARGS>
#define i2c_start i2c_START<i2c_ARGS>
#define i2c_restart     i2c_RESTART<i2c_ARGS>
#define i2c_stop  i2c_STOP<i2c_ARGS>
#define i2c_read  i2c_READ<i2c_ARGS>
#define i2c_write i2c_WRITE<i2c_ARGS>
```

```
////////////////////////////////////////////////////////////////////////
// I2C Control Status Bits - Emulates the PIC18F hardware I2C implementation
////////////////////////////////////////////////////////////////////////
// define I2C i2C_SSPCON1 control bits
#define     i2c_WCOL    7
#define     i2c_SSPOV   6
#define     i2c_SSPEN   5
#define     i2c_CKP           4
#define     i2c_SSPM3   3
#define     i2c_SSPM2   2
#define     i2c_SSPM1   1
#define     i2c_SSPM0   0

// define I2C SSPCON2 control bits
#define     i2c_GCEN    7
#define     i2c_ACKSTAT 6
#define     i2c_ACKDT   5
#define     i2c_ACKEN   4
#define     i2c_RCEN    3
#define     i2c_PEN           2
#define     i2c_RSEN    1
#define     i2c_SEN           0



// define I2C SSPSTAT status bits
#define     i2c_DA            5
#define     i2c_P       4
#define     i2c_S       3
#define     i2c_RW            2
#define     i2c_UA            1
#define     i2c_BF            0

////////////////////////////////////////////////////////////////////////
// I2C Control Flag Bits
////////////////////////////////////////////////////////////////////////
// define I2C Mode bits
#define     i2c_HW                  0x01
#define     i2c_400KHz        0x02  // 100KHz or 400KHz I2C clock (set =
400KHz)
#define     i2c_reset_wdt     0x04
#define i2c_SMP               0x80

////////////////////////////////////////////////////////////////////////
// I2C software constants
////////////////////////////////////////////////////////////////////////
#define     dly 10      // number of 1us delay increments

////////////////////////////////////////////////////////////////////////
// Generates the I2C Bus Start Condition
////////////////////////////////////////////////////////////////////////
_I2C_TEMPL
void i2c_START(void)
{
      ledVm = 1;

      // Initiate the I2C START condition
      volatile bit l_scl@T_SCL_PORT.T_SCL_BIT, l_sda@T_SDA_PORT.T_SDA_BIT;
      volatile bit l_scl_tris@T_SCL_TRIS.T_SCL_BIT,
l_sda_tris@T_SDA_TRIS.T_SDA_BIT;
      volatile bit l_sspif@T_i2c_SSPIF_PIR.T_i2c_SSPIF_BIT,
l_bclif@T_i2c_BCLIF_PIR.T_i2c_BCLIF_BIT;
      volatile bit l_rw@T_i2c_SSPSTAT.i2c_RW, l_s@T_i2c_SSPSTAT.i2c_S,
l_sen@T_i2c_SSPCON2.i2c_SEN;
```

```
      volatile bit l_rcen@T_i2c_SSPCON2.i2c_RCEN, l_pen@T_i2c_SSPCON2.i2c_PEN;
      volatile bit l_rsen@T_i2c_SSPCON2.i2c_RSEN,
l_acken@T_i2c_SSPCON2.i2c_ACKEN;

      delay_us(dly);
      l_bclif = 0; // initialise the collision flag for this command
      l_sspif = 0;

      //    Hardware I2C implementation
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      l_sen = 1; // initiate START condition

      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw ||
!l_sspif)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

}
////////////////////////////////////////////////////////////////////////
// Generates the I2C Bus Restart Condition
////////////////////////////////////////////////////////////////////////
_I2C_TEMPL
void i2c_RESTART(void)
{
      // Initiate the I2C RESTART condition
      volatile bit l_scl@T_SCL_PORT.T_SCL_BIT, l_sda@T_SDA_PORT.T_SDA_BIT;
      volatile bit l_scl_tris@T_SCL_TRIS.T_SCL_BIT,
l_sda_tris@T_SDA_TRIS.T_SDA_BIT;
      volatile bit l_sspif@T_i2c_SSPIF_PIR.T_i2c_SSPIF_BIT,
l_bclif@T_i2c_BCLIF_PIR.T_i2c_BCLIF_BIT;
      volatile bit l_rw@T_i2c_SSPSTAT.i2c_RW, l_s@T_i2c_SSPSTAT.i2c_S;
      volatile bit l_rcen@T_i2c_SSPCON2.i2c_RCEN, l_pen@T_i2c_SSPCON2.i2c_PEN,
l_sen@T_i2c_SSPCON2.i2c_SEN;
      volatile bit l_rsen@T_i2c_SSPCON2.i2c_RSEN,
l_acken@T_i2c_SSPCON2.i2c_ACKEN;

      delay_us(dly);
      l_bclif = 0; // initialise the collision flag for this command
      l_sspif = 0;

      //    Hardware I2C implementation
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      l_rsen = 1; // initiate RESTART condition

      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw ||
!l_sspif)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();
}

////////////////////////////////////////////////////////////////////////
// Generates the I2C Bus Stop Condition
////////////////////////////////////////////////////////////////////////
_I2C_TEMPL
void i2c_STOP(void)
{
      volatile bit l_scl@T_SCL_PORT.T_SCL_BIT, l_sda@T_SDA_PORT.T_SDA_BIT;
      volatile bit l_scl_tris@T_SCL_TRIS.T_SCL_BIT,
l_sda_tris@T_SDA_TRIS.T_SDA_BIT;
```

```
      volatile bit l_sspif@T_i2c_SSPIF_PIR.T_i2c_SSPIF_BIT,
l_bclif@T_i2c_BCLIF_PIR.T_i2c_BCLIF_BIT;
      volatile bit l_rw@T_i2c_SSPSTAT.i2c_RW, l_s@T_i2c_SSPSTAT.i2c_S,
l_p@T_i2c_SSPSTAT.i2c_P;
      volatile bit l_rcen@T_i2c_SSPCON2.i2c_RCEN, l_pen@T_i2c_SSPCON2.i2c_PEN,
l_sen@T_i2c_SSPCON2.i2c_SEN;
      volatile bit l_rsen@T_i2c_SSPCON2.i2c_RSEN,
l_acken@T_i2c_SSPCON2.i2c_ACKEN;

      l_bclif = 0; // initialise the collision flag for this command
      l_sspif = 0;

      //    Hardware I2C implementation
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      l_pen = 1; // initiate STOP condition on the I2C bus

      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw ||
!l_sspif)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      ledVm = 0;
}

////////////////////////////////////////////////////////////////////////
// Generates the I2C Bus Write Condition
////////////////////////////////////////////////////////////////////////
_I2C_TEMPL
unsigned char i2c_WRITE(unsigned char i2c_data)
{
      volatile unsigned char i2c_SSPBUF@T_i2c_SSPBUF;
      volatile bit l_scl@T_SCL_PORT.T_SCL_BIT, l_sda@T_SDA_PORT.T_SDA_BIT;
      volatile bit l_scl_tris@T_SCL_TRIS.T_SCL_BIT,
l_sda_tris@T_SDA_TRIS.T_SDA_BIT;
      volatile bit l_bf@T_i2c_SSPSTAT, l_ackdt@T_i2c_SSPCON2.i2c_ACKDT;
      volatile bit l_sspif@T_i2c_SSPIF_PIR.T_i2c_SSPIF_BIT,
l_bclif@T_i2c_BCLIF_PIR.T_i2c_BCLIF_BIT;
      volatile bit l_rw@T_i2c_SSPSTAT.i2c_RW,l_wcol@T_i2c_SSPCON1.i2c_WCOL;
      volatile bit l_rcen@T_i2c_SSPCON2.i2c_RCEN, l_pen@T_i2c_SSPCON2.i2c_PEN,
l_sen@T_i2c_SSPCON2.i2c_SEN;
      volatile bit l_rsen@T_i2c_SSPCON2.i2c_RSEN,
l_acken@T_i2c_SSPCON2.i2c_ACKEN;

      char BitMask;
      bit local_ack;

      l_bclif = 0; // initialise the collision flag for this command
      l_sspif = 0; // clear the operation completed

      //    Hardware I2C implementation
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      l_wcol = 0; // clear write collision flag
      i2c_SSPBUF = i2c_data;

      // test if a write collision occurred
      if (l_wcol)
            return (1); // error exit

      // wait until MSSP Tx register is empty
```

```
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw ||
!l_sspif)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      return (0); // successful exit
}

////////////////////////////////////////////////////////////////////////
// Generates the I2C Bus Read Condition
////////////////////////////////////////////////////////////////////////
_I2C_TEMPL
unsigned char i2c_READ(char ack_status)
{
      volatile unsigned char i2c_SSPBUF@T_i2c_SSPBUF;
      volatile bit l_scl@T_SCL_PORT.T_SCL_BIT, l_sda@T_SDA_PORT.T_SDA_BIT;
      volatile bit l_scl_tris@T_SCL_TRIS.T_SCL_BIT,
l_sda_tris@T_SDA_TRIS.T_SDA_BIT;
      volatile bit l_bf@T_i2c_SSPSTAT.i2c_BF, l_ackdt@T_i2c_SSPCON2.i2c_ACKDT;
      volatile bit l_sspif@T_i2c_SSPIF_PIR.T_i2c_SSPIF_BIT,
l_bclif@T_i2c_BCLIF_PIR.T_i2c_BCLIF_BIT;
      volatile bit l_rw@T_i2c_SSPSTAT.i2c_RW,l_wcol@T_i2c_SSPCON1.i2c_WCOL;
      volatile bit l_rcen@T_i2c_SSPCON2.i2c_RCEN, l_pen@T_i2c_SSPCON2.i2c_PEN,
l_sen@T_i2c_SSPCON2.i2c_SEN;
      volatile bit l_rsen@T_i2c_SSPCON2.i2c_RSEN,
l_acken@T_i2c_SSPCON2.i2c_ACKEN;

      char BitMask;
      char i2c_data;
      bit local_ack;

      l_bclif = 0; // initialise the collision flag for this command
      l_sspif = 0; // clear the operation completed
      l_wcol = 0; // clear write collision flag

      //    Hardware I2C implementation
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      // enable master for 1 byte reception
      l_rcen = 1;

      // wait until byte received
      while(!l_sspif || !l_bf)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      // read the byte from the Rx register
      i2c_data = i2c_SSPBUF;

      // wait until the bus is idle
      while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw)
            if (T_MODE & i2c_reset_wdt)
                  clear_wdt();

      if (ack_status)
            l_ackdt = 1; // preset ack bit
      else
            l_ackdt = 0; // preset ack bit

      l_sspif = 0;
      l_acken = 1; // acknowledge sequence enable
```

```
	while (l_acken || l_rcen || l_pen || l_rsen || l_sen || l_rw ||
!l_sspif)
		if (T_MODE & i2c_reset_wdt)
			clear_wdt();

	return(i2c_data);
}

////////////////////////////////////////////////////////////////////////
// Generates the I2C Bus Initialization
////////////////////////////////////////////////////////////////////////
_I2C_TEMPL
void i2c_INIT(unsigned char i2c_divisor)
{
	volatile unsigned char i2c_SSPADD@T_i2c_SSPADD,
	i2c_SSPSTAT@T_i2c_SSPSTAT;
	volatile unsigned char
	i2c_SSPCON1@T_i2c_SSPCON1,i2c_SSPCON2@T_i2c_SSPCON2;

	volatile bit l_scl@T_SCL_PORT.T_SCL_BIT, l_sda@T_SDA_PORT.T_SDA_BIT;
	volatile bit l_scl_tris@T_SCL_TRIS.T_SCL_BIT,
	l_sda_tris@T_SDA_TRIS.T_SDA_BIT;
	volatile bit l_sspif@T_i2c_SSPIF_PIR.T_i2c_SSPIF_BIT,
	l_bclif@T_i2c_BCLIF_PIR.T_i2c_BCLIF_BIT;
	volatile bit l_sspen@T_i2c_SSPCON1.i2c_SSPEN,
	l_smp@T_i2c_SSPSTAT.T_i2c_SMP_BIT;

	l_sda_tris = 1;
	l_scl_tris = 1;

	i2c_SSPCON1 = 0x00; // initialise the I2C control register (mirrors HW
SSPCON1)
	i2c_SSPADD = i2c_divisor; // get the I2C baud rate divisor
	i2c_SSPCON1 = 0x08; // initialise the I2C control register (mirrors HW
SSPCON1)
	i2c_SSPCON2 = 0x00; // initialise the I2C control register (mirrors HW
SSPCON2)

	i2c_SSPSTAT = 0x00; // initialise the I2C status register (mirrors HW
SSPSTAT)

	if (T_MODE & i2c_SMP)
		l_smp = 1;

	l_sspif = 0; // initialise the I2C SSP interrupt status
	l_bclif = 0; // initialise the I2C BCL interrupt status
	l_sda = 0;
	l_scl = 0;
	l_sspen = 1; // enable I2C

	i2c_STOP<_I2C_TEMPL_ARGS>();
}

#endif // _I2C_DRV_H_
```

2 – Códigos Fonte do Programa do **Módulo Gerador de Sinais**

main.c

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#include <system.h>
#include "main.h"
#include "i2c.h"
#include "adc.h"

#pragma CLOCK_FREQ 20000000

#pragma DATA 0x2007, _WDT_ON & _LVP_OFF & _PWRTE_ON & _HS_OSC & _CCP1_RB3 &
_DEBUG_OFF & _MCLR_OFF

char I2cTxRx;           // flag utilizado na recepçãp I2C (Tx ou Rx)

void main()
{
      config();

      char c = 0;
      unsigned short temp = 0;
      while(1)
      {
            if(procComando) {

                  switch(comando) {

                        case(CMD_SET_RELOGIO):

                              charToShort(&param, rxShortBuffer[1],
rxShortBuffer[2]);

                              if(param <= RELOGIO_MAX) {
                                    ledI2c_on;
                                    rel = param;
                              }

                              break;

                        case(CMD_SET_PULSE_ON):

                              charToShort(&param, rxShortBuffer[1],
rxShortBuffer[2]);

                              if(param <= PULSE_CC_MAX) {
                                    ledI2c_on;
                                    pcc_on = param;
                              }

                              break;
```

```
                        case(CMD_SET_PULSE_OFF):

                              charToShort(&param, rxShortBuffer[1],
rxShortBuffer[2]);

                              if(param <= PULSE_CC_MAX) {
                                    ledI2c_on;
                                    pcc_off = param;
                              }

                              break;

                        case(CMD_SET_PULSE_QTD):

                              charToShort(&param, rxShortBuffer[1],
rxShortBuffer[2]);

                              if(param <= PULSE_QTD_MAX) {
                                    ledI2c_on;
                                    pcc_qtd = param;
                              }

                              break;
                  }

                  delay_ms(10);
                  startGINT;
                  startRecur();

                  comando = CMD_OCIOSO;
                  procComando = false;
            }

            delay_ms(1);
            clear_wdt();

            ledI2c_off;
      }
}

void config() {

      // Configuração do oscilador

      // Se comentar o código abaixo o pic usará o clock primário
      /*osccon = 0x7E;  // Configura OSC interno
                        // 8Mhz
                        // INTRC
                        // Modo do oscilador definido em FOSC<2:0>
      */

      // Configuração do WDT
      wdtcon = 00010111b;

      // Configuração de I/O
      trisa = 0x00;
      porta = 0x00;
      trisb &= 00010011b;     // Configura os pinos SDA e SCL e INT
      portb = 0x00;

      ledPwr_tris = 0;
      ledSig_tris = 0;
      ledI2c_tris = 0;

      // Configuração da comunicação SSP - I2C
```

```
	startGINT;				// Liga as interrupções
	set_bit(intcon, PEIE);		// Set PEIE - Liga interrupções periféricas
	set_bit(pie1, SSPIE);		// Set SSPIE - Liga interrupção do módulo SSP
	clear_bit(pir1, SSPIF ); // Limpa interrupções pendentes

	sspadd = deviceAddress;  // Configura o endereço para o módulo SSP

	// ,--------> WCOL      - Write Collision Detect bit
	// |,-------> SSPOV     - Receive Overflow Indicator bit
	// ||,------> SSPEN     - Synchronous Serial Port Enable bit
	// |||,-----> CKP       - Clock Polarity Select bit
	// ||||,,,,-> SSPM<3:0> - Synchronous Serial Port Mode Select bits
	// ||||||||
	// 00110110 Habilita o módulo SSP - Slave 7 bits

	sspcon = 0x36;

	// Ativa o conversor AD
	adc_init();

	// Ativa o timer1
	set_bit(pie1, TMR1IE);			// Set TMR1IE - Liga interrupção do
módulo TIMER1
	clear_bit(pir1, TMR1IF);		// Limpa interrupções pendentes
	ledPwr = 0;

	// Ativa a interrupção externa
	startInt;
	clear_bit(option_reg, INTEDG);	// A interrupção ocorre na borda de
descida


	// comparator voltage reference module
	/*cmcon &= 11111010b;
	cvrcon &= 11110011b;
	set_bit(cvrcon, CVROE);
	set_bit(cvrcon, CVREN);*/
}

void interrupt(void)
{
	if(l_sspif)
	{
		stopRecur();

		l_sspif = 0;

		if(l_sspov) {
			l_sspov = 0;
			if(l_bf) rxShortBuffer[countRx] = sspbuf;
		}
		else
		{
			if(l_da) // Data
			{
				if(!I2cTxRx) // Modo Rx
				{
					if(countRx < N_RX_SHORT_MX)
					{
						rxShortBuffer[countRx] = sspbuf;
						countRx++;

						if(countRx == N_RX_SHORT_MX)
						{
							comando = rxShortBuffer[0];
```

```
                                    processarComando();
                              }
                        }
                  }
                  else // Modo Tx
                  {
                        if(countTx < N_TX_SHORT_MX)
                        {
                              sspbuf = txShortBuffer[countTx];
                              countTx++;
                        }
                        else
                        {
                              sspbuf = 0;
                        }

                        l_ckp = 1;
                  }
            }
            else // Address
            {
                  if(l_rw) // Modo de transmissão
                  {
                        I2cTxRx = 1;
                        countTx = 1;
                        sspbuf = txShortBuffer[0];
                        l_ckp = 1;
                  }
                  else // Modo de recepção
                  {
                        I2cTxRx = 0;
                        countRx = 0;
                        rxShortBuffer[countRx] = sspbuf; // Leitura
fictícia
                  }
            }
      }
}
else {

      if(l_intf) {

            l_intf = 0;

            if(ledPwr) {
                  stopTimer;
                  ledSig_off;
                  ledPwr_off;
            }
            else {
                  startTimer;
                  ledPwr_on;
            }
      }
      else if(l_tmr1if) {

            l_tmr1if = 0;
            stopTimer;
            tmr1h = 0xFF;           // Configura o estouro a cada 20us
            tmr1l = 0xFF;

            ++contador;

            if(contador == rel) {
```

```
                        sendPulses();
                        contador = 0;
                  }

                  startTimer;
            }
      }
}

void startRecur() {
      if(ledPwr) startTimer;
      startInt;
}

void stopRecur() {
      stopTimer;
      stopInt;
      l_intf = 0;
      l_tmr1if = 0;
}

void cleanI2cRecur() {
      l_sspif = 0;
      countRx = 0;
      countTx = 0;
}

void sendPulses(void) {
      char i;
      for(i=0; i<pcc_qtd; i++) {
            ledSig_on;
            forDelay(pcc_on);
            ledSig_off;
            if(i<(pcc_qtd-1)) forDelay(pcc_off);
      }
}

////////////////////////////////////////////////////////////////////////
//    A chamada da função em sí leva 6,8 us e cada giro leva 5,6 us
////////////////////////////////////////////////////////////////////////
void forDelay(unsigned short cc) {
      unsigned short i;
      for(i=0; i<cc; i++) {
            i=i;
      }
}

void processarComando()
{
      switch(comando)
      {
            case CMD_GET_RELOGIO:

                  shortToBuffer(rel, txShortBuffer);

                  break;

            case CMD_GET_PULSE_ON:

                  shortToBuffer(pcc_on, txShortBuffer);

                  break;

            case CMD_GET_PULSE_OFF:
```

```
                  shortToBuffer(pcc_off, txShortBuffer);

                  break;

            case CMD_GET_PULSE_QTD:

                  shortToBuffer(pcc_qtd, txShortBuffer);

                  break;

            case CMD_SET_RELOGIO:
                  break;
      }

      stopGINT;
      stopRecur();

      procComando = true;
}

void shortToBuffer(unsigned short shortValue, unsigned char *buffer)
{
      buffer[0] = (unsigned char) shortValue;
      shortValue = shortValue >> 8;
      buffer[1] = (unsigned char) shortValue;
}

void charToShort(unsigned short *dst, char lobyte, char hibyte)
{
      *dst = (unsigned short)hibyte;
      *dst = *dst << 8;
      *dst |= lobyte;
}
```

2 – Códigos Fonte do Programa do **Módulo Gerador de Sinais**

main.h

```
/*********************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                     *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS  *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                              *
 *                                                                   *
 *    $Author: Rafael Chagas $                                       *
 *    $Date: 2009/09/29 22:07:00 $                                   *
 *    $Revision: 1.0 $                                               *
 *                                                                   *
 *********************************************************************/

#ifndef _MAIN_H_
#define _MAIN_H_

// Definições das utilizações das portas
//////////////////////////////////////////////////////////////////////
#define LED_PWR_PORT          PORTB
#define LED_PWR_TRIS          TRISB
#define LED_PWR_PIN                 2
volatile bit ledPwr                 @ LED_PWR_PORT    . LED_PWR_PIN;
volatile bit ledPwr_tris      @ LED_PWR_TRIS    . LED_PWR_PIN;

#define LED_SIG_PORT          PORTB
#define LED_SIG_TRIS          TRISB
```

```
#define LED_SIG_PIN                3
volatile bit ledSig                @ LED_SIG_PORT    . LED_SIG_PIN;
volatile bit ledSig_tris      @ LED_SIG_TRIS    . LED_SIG_PIN;

#define LED_I2C_PORT          PORTB
#define LED_I2C_TRIS          TRISB
#define LED_I2C_PIN                5
volatile bit ledI2c                @ LED_I2C_PORT    . LED_I2C_PIN;
volatile bit ledI2c_tris      @ LED_I2C_TRIS    . LED_I2C_PIN;

// Controle das interrupções
////////////////////////////////////////////////////////////////////////////
#define startGINT set_bit(intcon, GIE);
#define stopGINT  clear_bit(intcon, GIE);

// Configuração do temporizador timer1
////////////////////////////////////////////////////////////////////////////
volatile bit l_tmr1if   @     PIR1.TMR1IF;
#define enableTimer set_bit(pie1, TMR1IE)
#define disableTimer clear_bit(pie1, TMR1IE)
#define startTimer enableTimer; set_bit(t1con, TMR1ON)
#define stopTimer disableTimer; clear_bit(t1con, TMR1ON)
unsigned short contador = 0;
unsigned short pcc_on = 100;
unsigned short pcc_off = 100;
unsigned short pcc_qtd = 4;
unsigned short rel = 100;

#define RELOGIO_MAX           1000
#define PULSE_CC_MAX    500
#define PULSE_QTD_MAX   50

// Configuração da interrupção externa
////////////////////////////////////////////////////////////////////////////
volatile bit l_intf           @     INTCON.INTF;
#define startInt set_bit(intcon, INTE)
#define stopInt clear_bit(intcon, INTE)

// Macros
////////////////////////////////////////////////////////////////////////////
#define ledPwr_on ledPwr = 1
#define ledPwr_off      ledPwr = 0

#define ledSig_on ledSig = 1
#define ledSig_off      ledSig = 0

#define ledI2c_on ledI2c = 1
#define ledI2c_off      ledI2c = 0

// Recursos para a máquina de estado
////////////////////////////////////////////////////////////////////////////
unsigned char estado = 0;

#define     T_OCIOSO    0
#define     T_SW_PRESS  1

// Funções
////////////////////////////////////////////////////////////////////////////
void config();
void startRecur();
void stopRecur();
void cleanI2cRecur();

void sendPulses(void);
void forDelay(unsigned short cc);
```

```
void processarComando();
void shortToBuffer(unsigned short shortValue, unsigned char *buffer);
void charToShort(unsigned short *dst, char lobyte, char hibyte);

// Comunicação i2c entre os MCPs
//////////////////////////////////////////////////////////////////////////
#define     N_RX_SHORT_MX     3
#define     N_TX_SHORT_MX     2
unsigned char rxShortBuffer[N_RX_SHORT_MX];
unsigned char txShortBuffer[N_TX_SHORT_MX];
unsigned char comando = 0;
bool procComando = false;
unsigned short param;
unsigned char countRx;
unsigned char countTx;

unsigned short mediaValor;


// Comandos disponíveis
//////////////////////////////////////////////////////////////////////////
#define CMD_OCIOSO                  0x00

#define CMD_SET_RELOGIO       0x10
#define CMD_GET_RELOGIO       0x11

#define CMD_SET_PULSE_QTD     0x20
#define CMD_GET_PULSE_QTD     0x21
#define CMD_SET_PULSE_ON      0x22
#define CMD_GET_PULSE_ON      0x23
#define CMD_SET_PULSE_OFF     0x24
#define CMD_GET_PULSE_OFF     0x25

#endif // _MAIN_H_
```

2 – Códigos Fonte do Programa do **Módulo Gerador de Sinais**

i2c.h

```
/*****************************************************************
 *    CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB                 *
 *    FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS    *
 *    CURSO DE ENGENHARIA DA COMPUTAÇÃO                          *
 *                                                               *
 *    $Author: Rafael Chagas $                                   *
 *    $Date: 2009/09/29 22:07:00 $                               *
 *    $Revision: 1.0 $                                           *
 *                                                               *
 *****************************************************************/

#include <system.h>

// Todas as variáveis abaixo são mapeadas para o PIC16F88
#define deviceAddress 0x02

#define e_SSPCON        0x0014
#define e_WCOL_BIT            7
#define e_SSPOV_BIT           6
#define e_SSPEN_BIT           5
#define e_CKP_BIT             4
```

```
#define e_SSPM3_BIT          3
#define e_SSPM2_BIT          2
#define e_SSPM1_BIT          1
#define e_SSPM0_BIT          0

#define e_SSPSTAT       0x0094
#define e_SMP_BIT            7
#define e_CKE_BIT            6
#define e_DA_BIT             5
#define e_P_BIT              4
#define e_S_BIT              3
#define e_RW_BIT             2
#define e_UA_BIT             1
#define e_BF_BIT             0

#define e_PIR1               0x000c
#define e_PIE1               0x008c
#define e_SSPIF_BIT          3
#define e_SSPIE_BIT          3

volatile bit l_sspif    @     e_PIR1.e_SSPIF_BIT;
volatile bit l_da       @     e_SSPSTAT.e_DA_BIT;
volatile bit l_rw       @     e_SSPSTAT.e_RW_BIT;
volatile bit l_bf       @     e_SSPSTAT.e_BF_BIT;
volatile bit l_sspov    @     e_SSPCON.e_SSPOV_BIT;
volatile bit l_ckp           @     e_SSPCON.e_CKP_BIT;
```

Apêndice C – Esquemas Eletrônicos

CIRCUITO DE CONTROLE

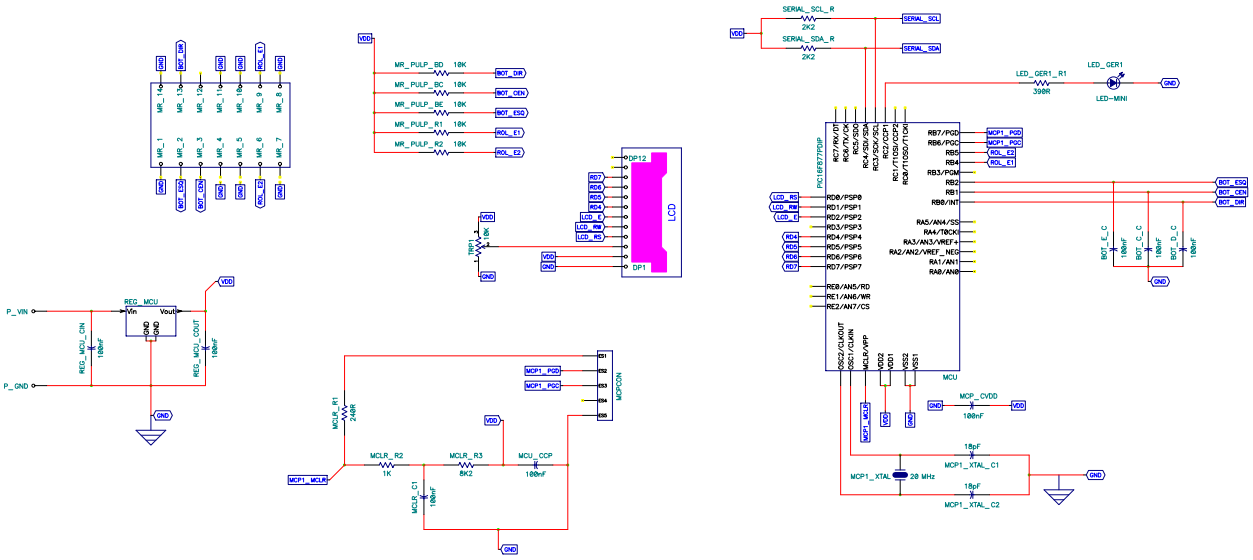


Figura 37 – Esquema eletrônico do módulo de controle

CIRCUITO DE GERAÇÃO DE SINAIS

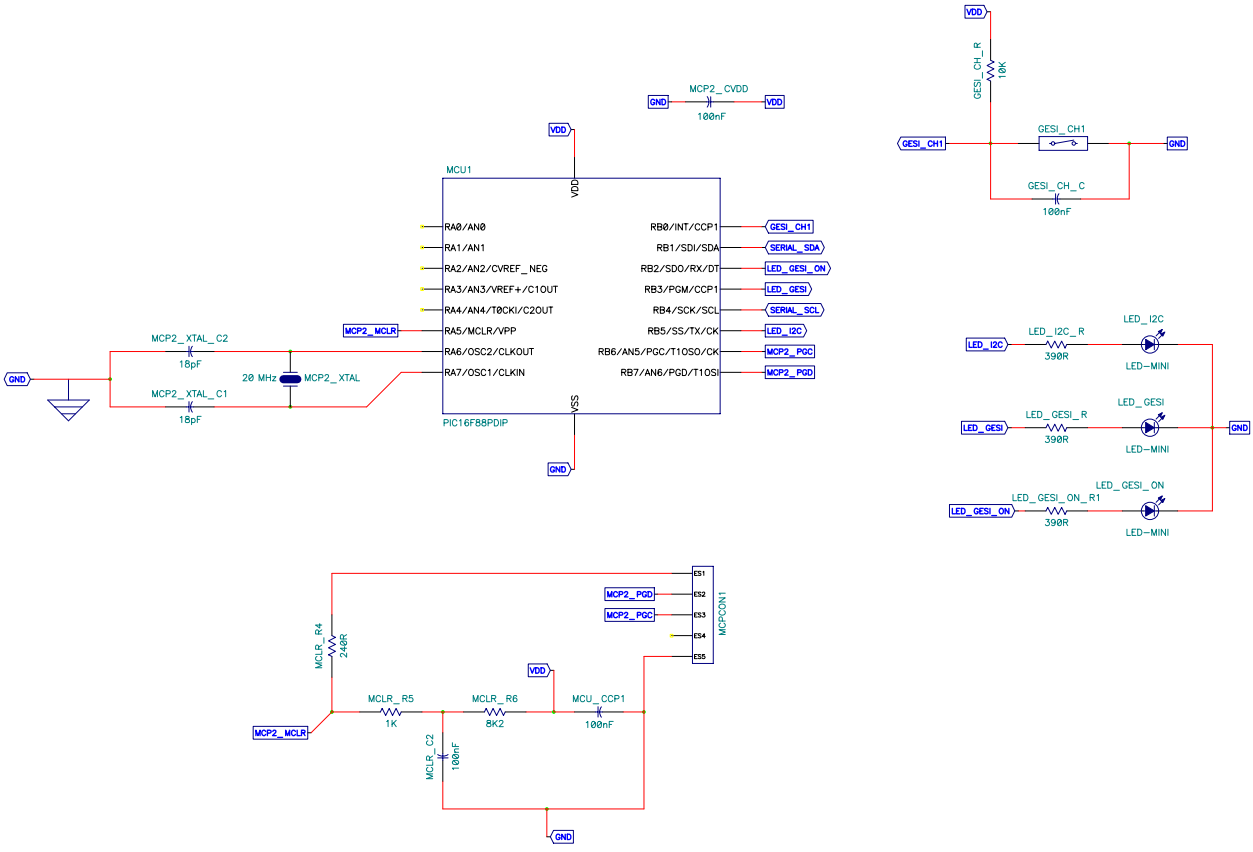


Figura 38 - Esquema eletrônico do módulo gerador de sinais

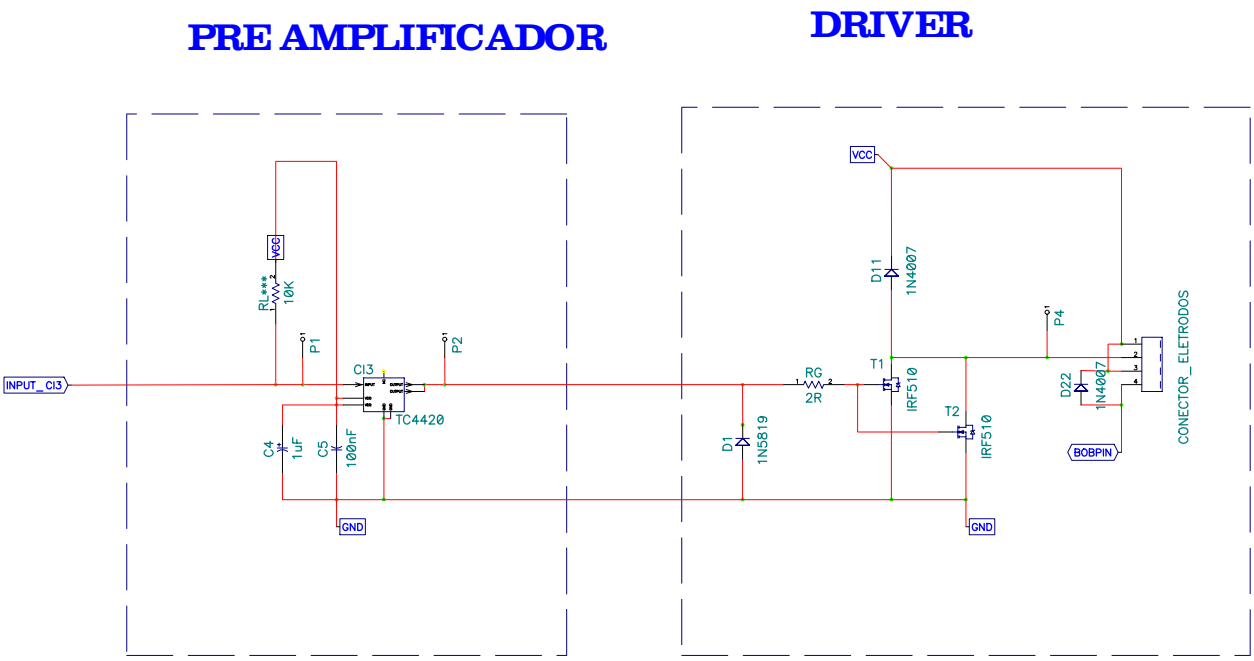


Figura 39 - Esquema eletrônico do módulo de potência

Apêndice D – Diagramas do módulo I²C

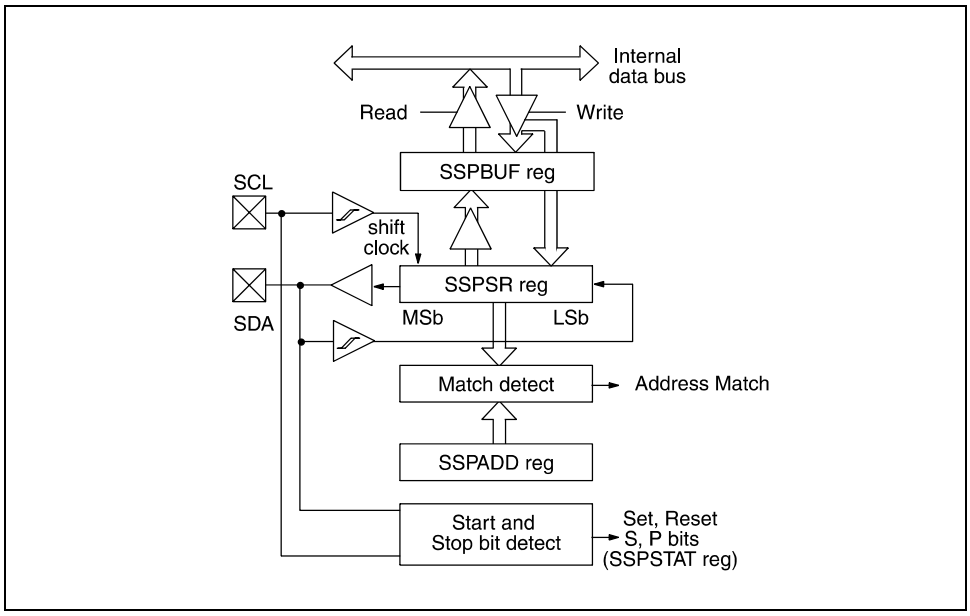


Figura 40 – Módulo I²C (modo ESCRAVO) – Diagrama em blocos

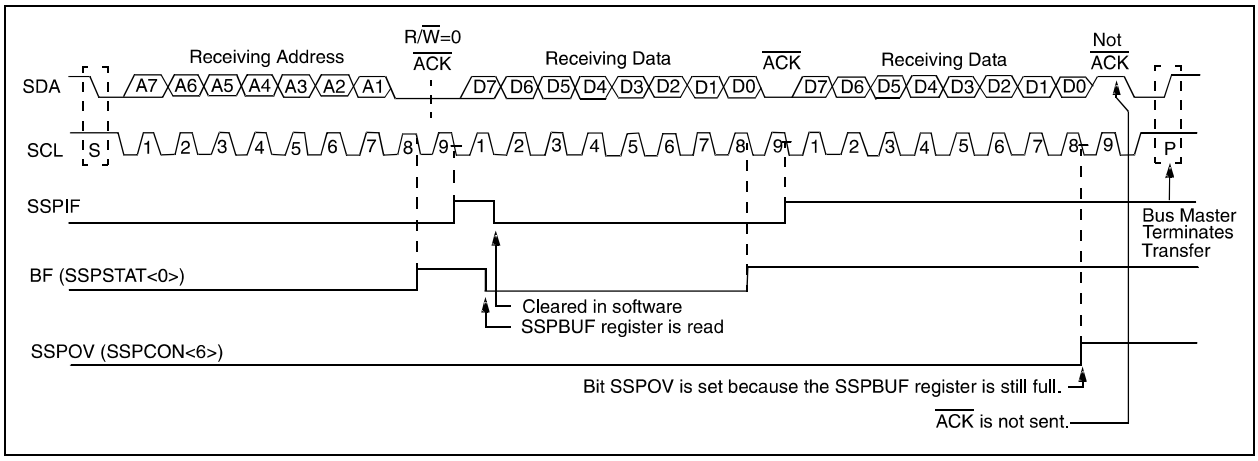


Figura 41 – Módulo I²C (modo ESCRAVO) – Forma de onda – RECEPÇÃO (Endereço de 7 bits)

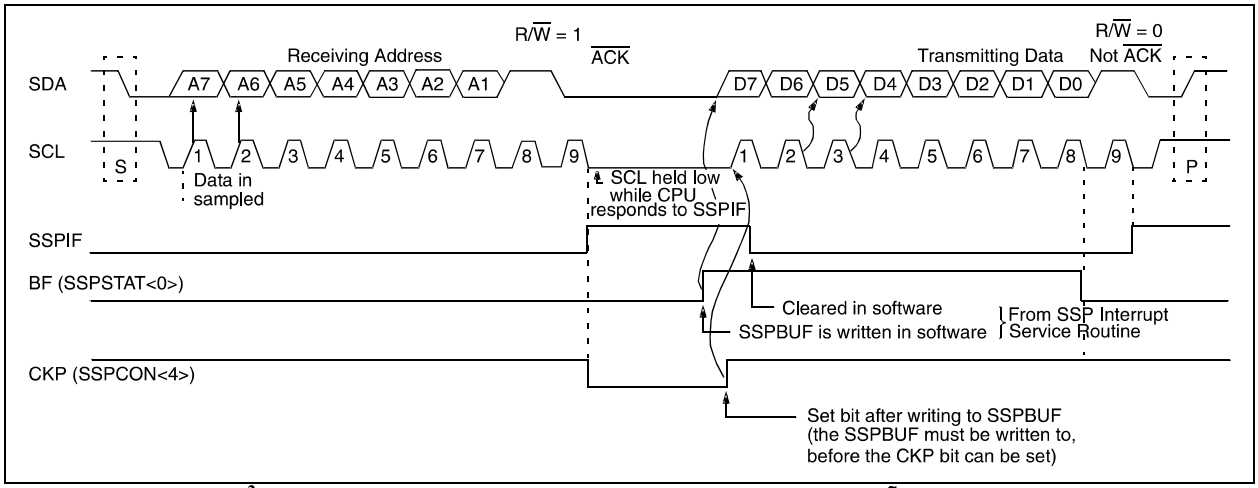


Figura 42 – Módulo I²C (modo ESCRAVO) – Forma de onda – TRANSMISSÃO (Endereço de 7 bits)

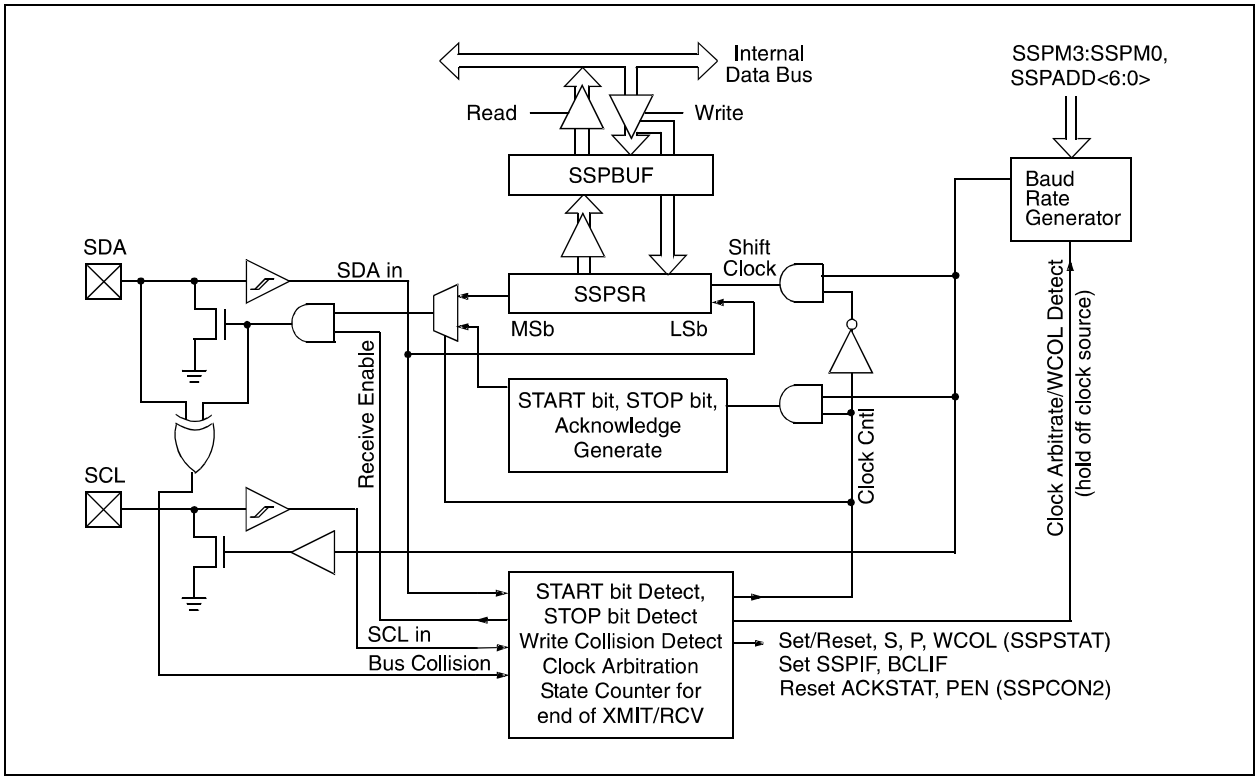


Figura 43 – Módulo I²C (modo MESTRE) – Diagrama em blocos

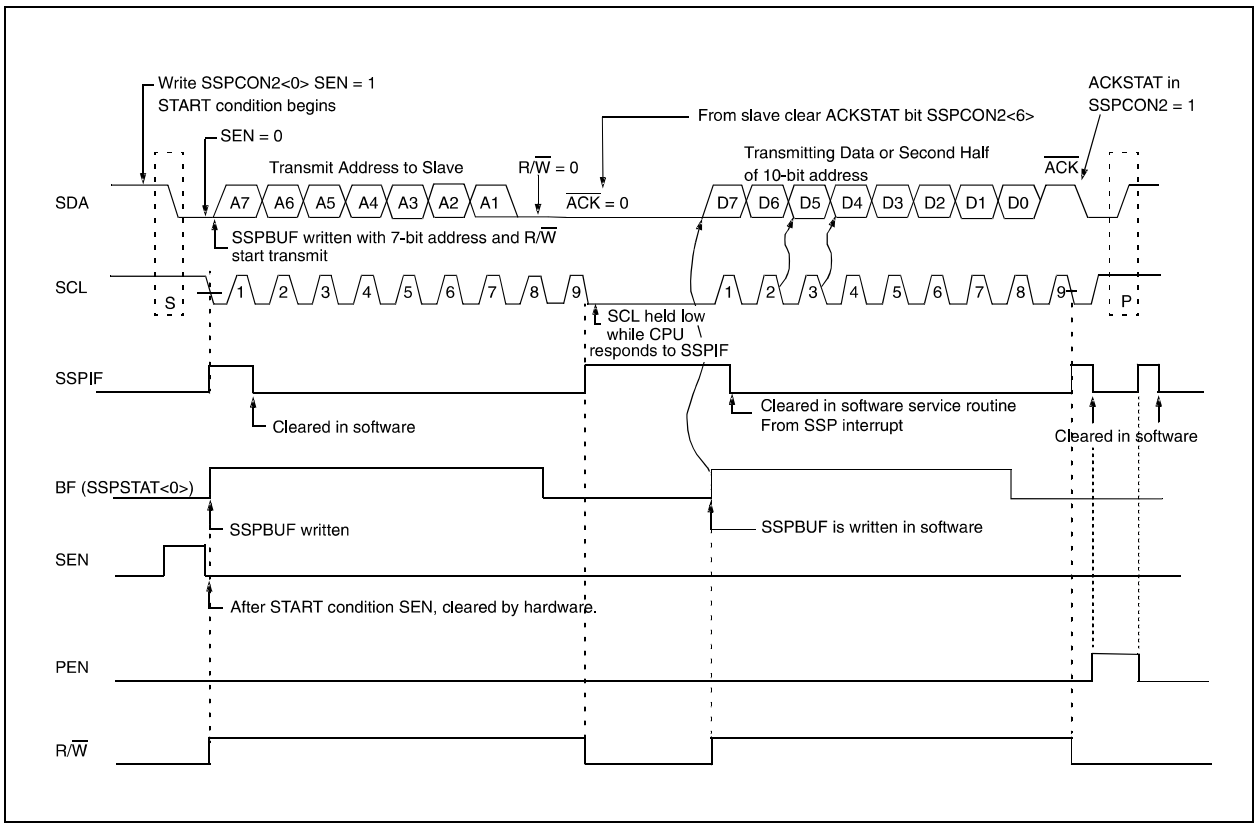


Figura 44 – Módulo I²C (modo MESTRE) – Forma de onda – TRANSMISSÃO (Endereço de 7 ou 10 bits)

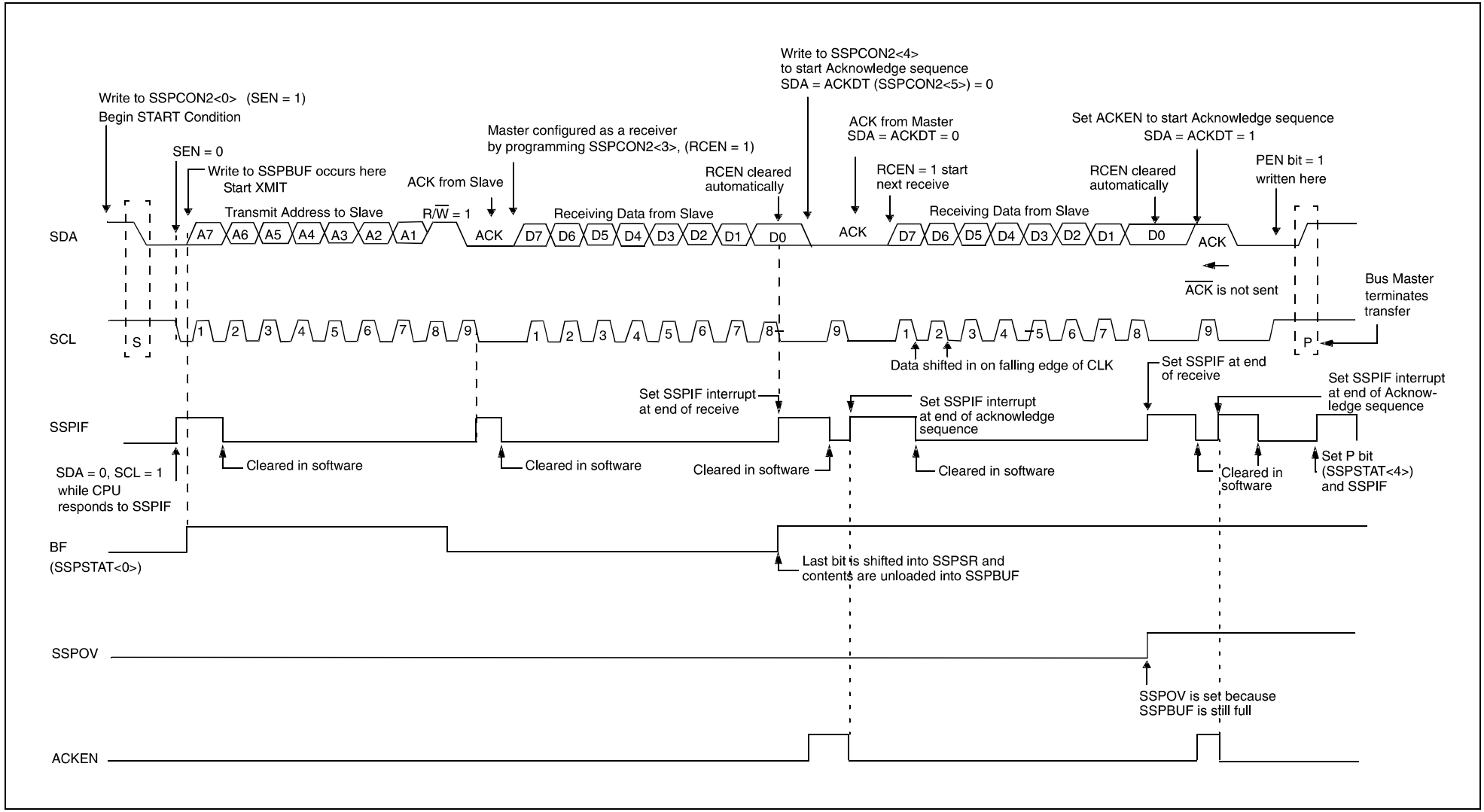
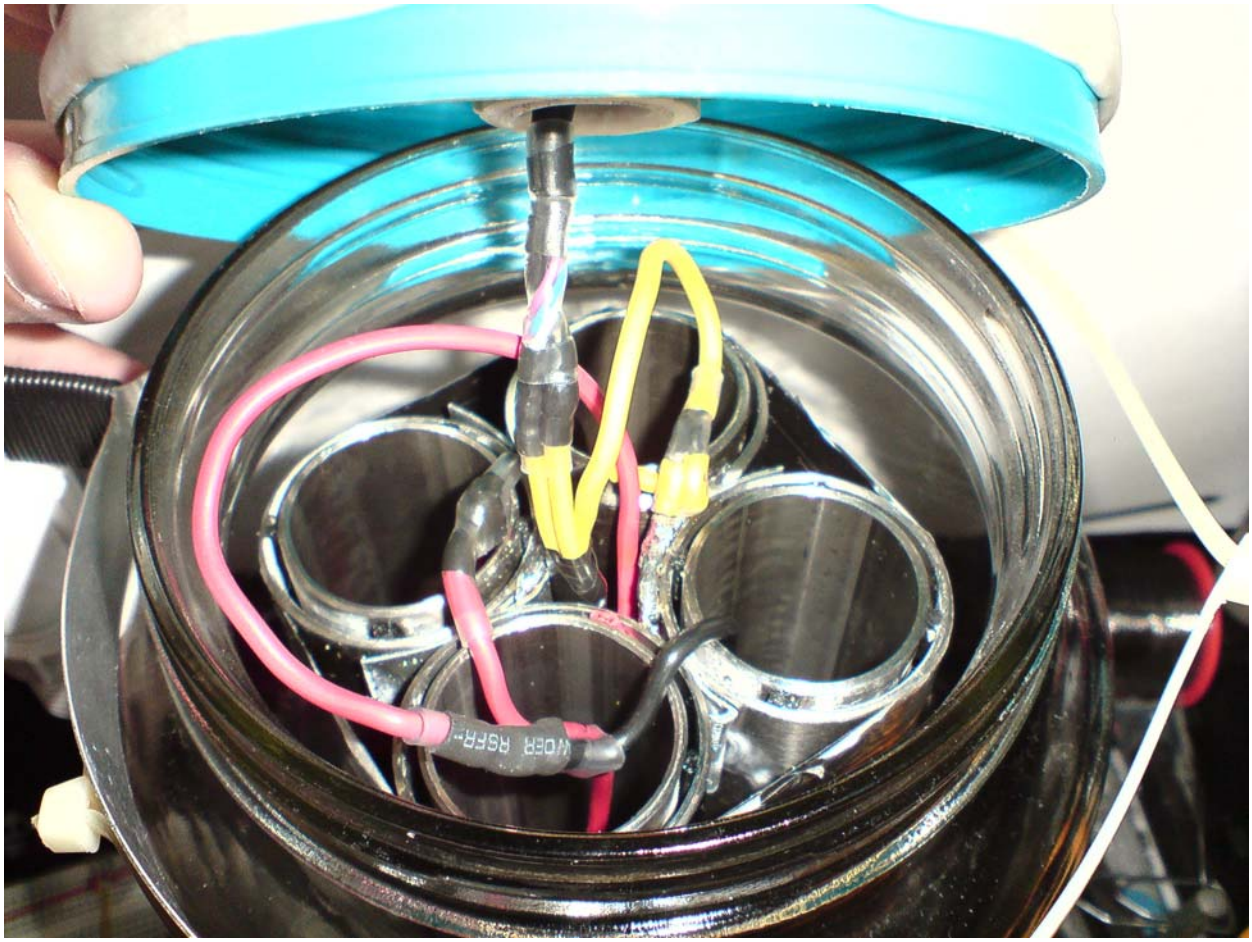
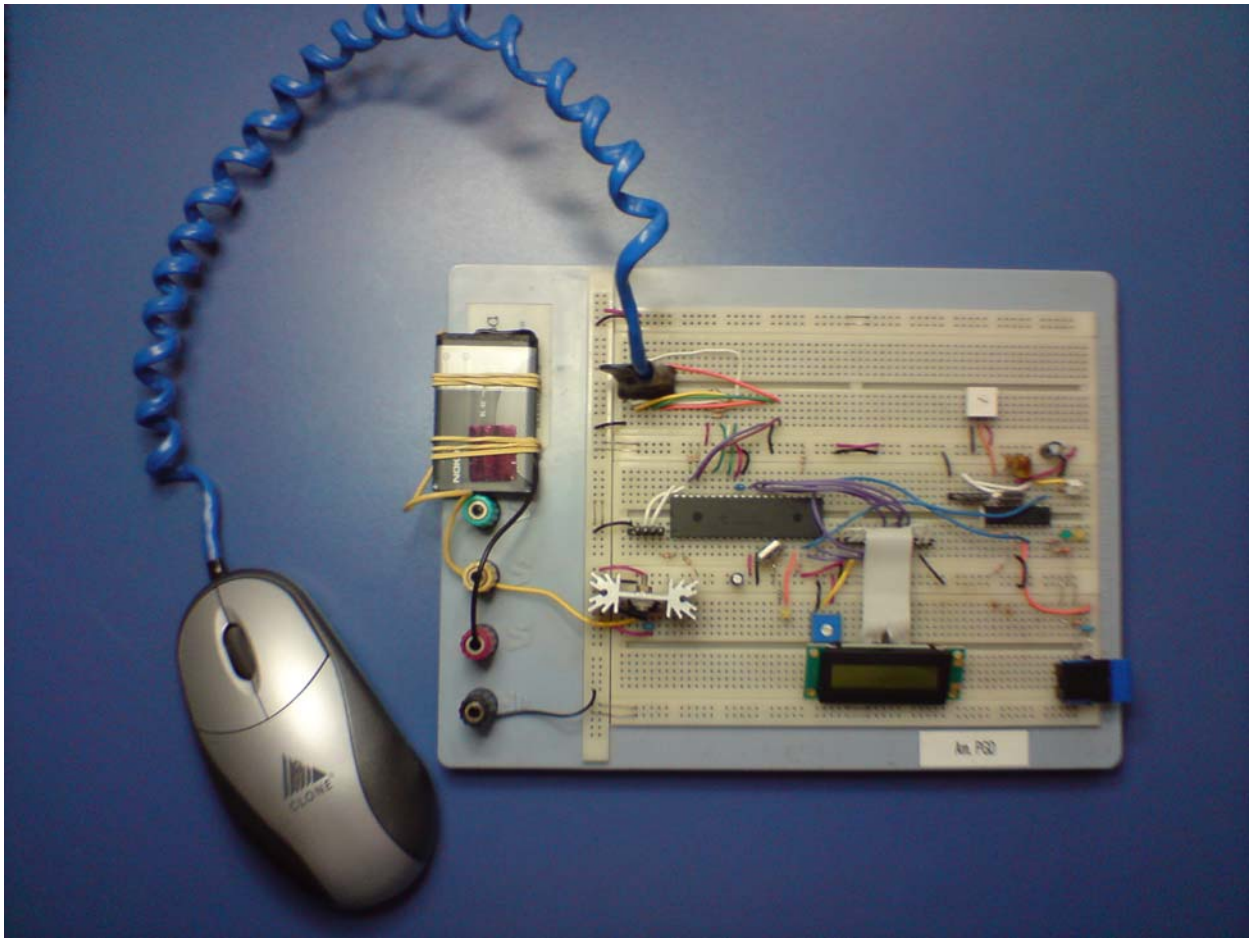
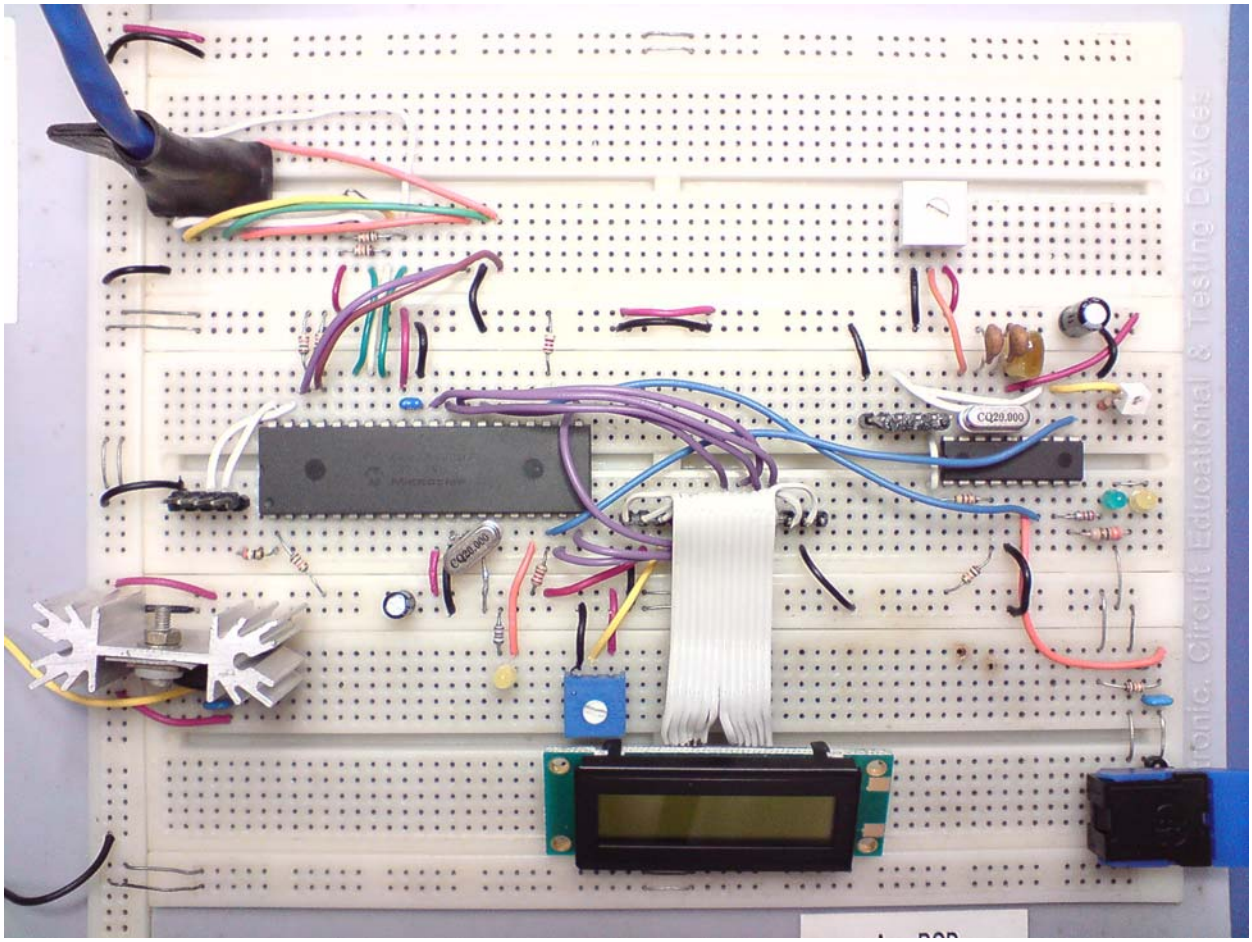


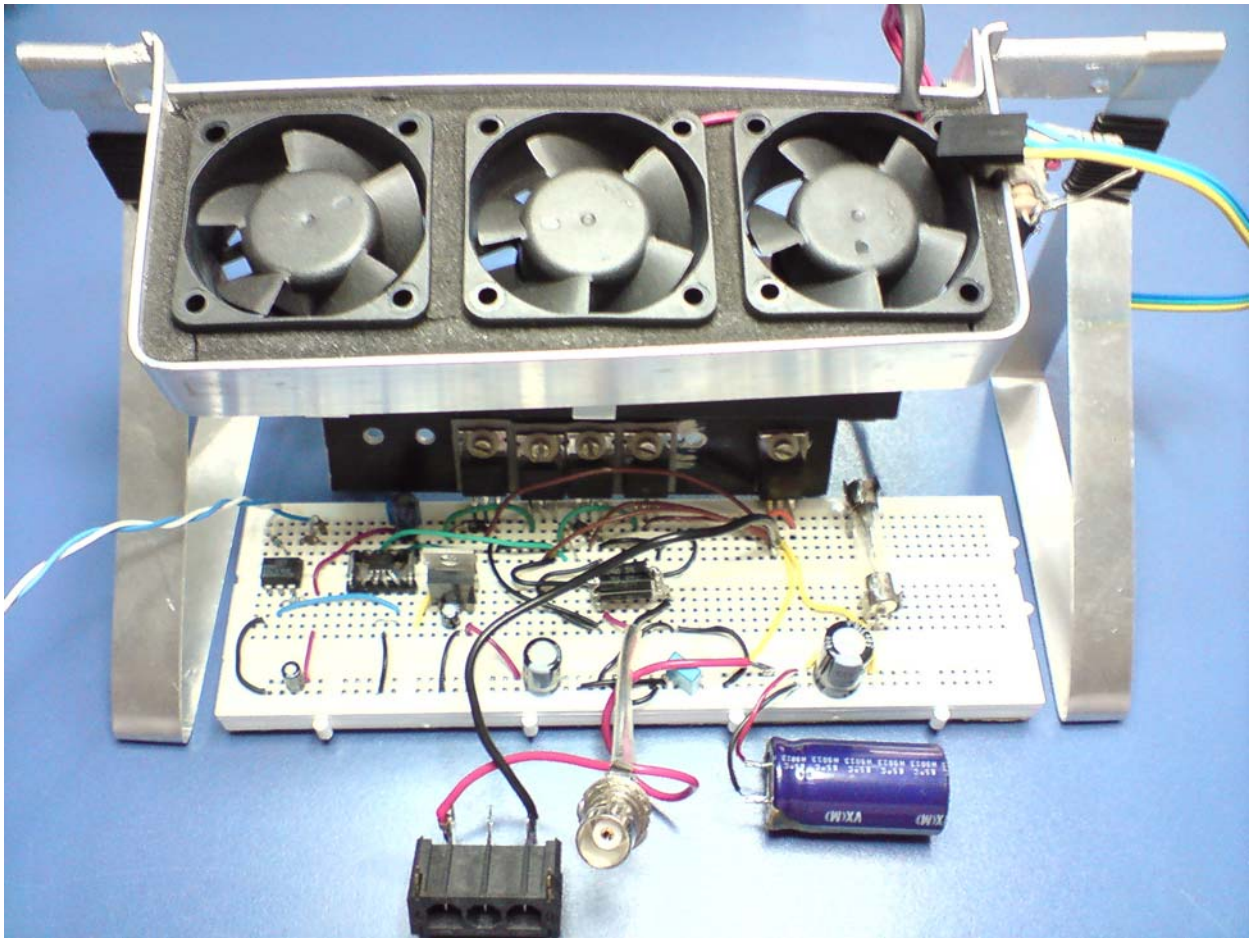
Figura 45 – Módulo I²C (modo MESTRE) – Forma de onda – RECEPÇÃO (Endereço de 7 bits)

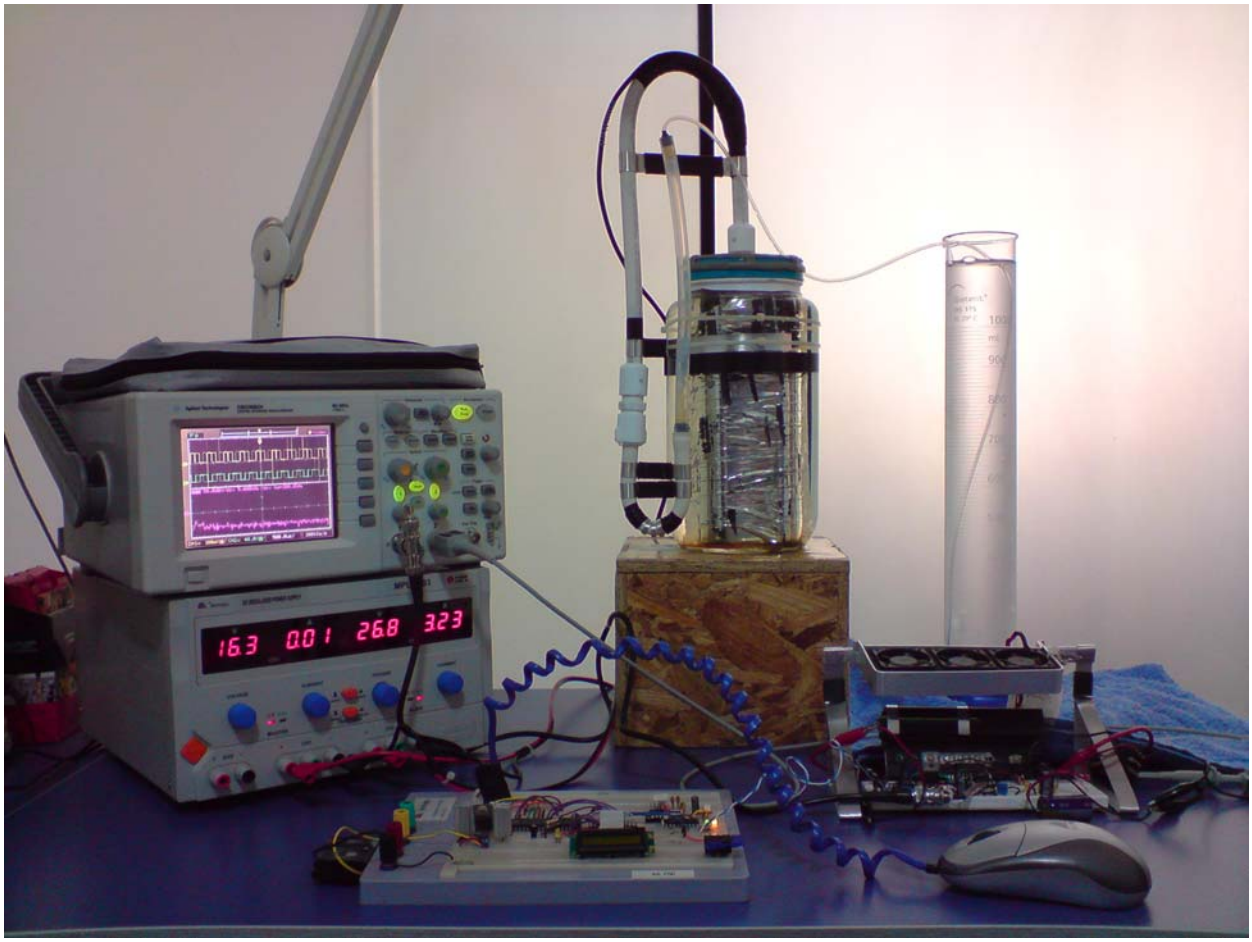
Anexo I – Fotos do Projeto






CQ20.000
Tonic Circuit Educational & Testing Devices




16.3 0.01 268 323