VINÍCIUS GOMES SERAGUCI

SONDA DE MONITORAMENTO DA QUALIDADE DA ÁGUA: MÓDULOS DE MEDIDA DE pH, OXIGÊNIO DISSOLVIDO E INTERFACEAMENTO I2C COM MICROCONTROLADOR

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

Curso de Engenharia Elétrica com ênfase em Eletrônica

ORIENTADOR: Valentin Obac Roda

São Carlos

2009

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Seraguci, Vinícius Gomes
S481s Sonda de monitoramento da qualidade da água : módulos de medida de pH, oxigênio dissolvido e interfaceamento 12C com microcontrolador / Vinícius Gomes Seraguci ; orientador Valentim Obac Roda. –- São Carlos, 2009.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2009.

1. Hidrologia. 2. Sonda. 3. MSP430.
4. Instrumentação. 5. Oxigênio dissolvido. I. Título.

Aos meus pais, por toda a dedicação, carinho, atenção e apoio durante toda a minha graduação.

À minha falecida avó, pelo apoio e ajuda.

A um grande amigo e companheiro que foi.

AGRADECIMENTOS

Ao Prof. Valentin Obac Roda, professor orientador, pelo auxílio e disponibilização do laboratório de ensino.

Ao CNPq/PIBIC pela bolsa de estudos.

Aos amigos Bruno Crippa, Tiago Spineli, Otto Litjens e Lucas Licursi, pelo apoio e auxílio na solução de problemas.

A Deus, por todos esses anos de graduação e pela realização deste projeto.

RESUMO

Este trabalho teve por objetivo o aperfeiçoamento de alguns dos circuitos de uma sonda para a medida de variáveis físicas e químicas, indicadoras da qualidade da água, a partir do isolamento dos circuitos condicionadores de sinais dos sensores de pH, oxigênio dissolvido, condutividade, temperatura, turbidez e pressão. Os sinais são processados por microcontroladores de baixo consumo que se comunicam através do padrão I2C, isolados opticamente, com um microcontrolador central que gerencia a informação e envia as informações a um microcomputador. Cada sensor junto com o circuito condicionador de sinal e unidade de processamento formam módulos com alimentação independente, a partir da utilização de conversores DC/DC. As informações enviadas ao computador são mostradas em um *software* desenvolvido exclusivamente para a sonda, para a captura dos dados. Testes foram realizados em laboratório com os sensores de pH e o medidor de oxigênio dissolvido, com a possibilidade de verificação das leitura na tela do PC. Os resultados obtidos foram comparados com os valores definidos e com os *buffers* utilizados.

Palavras-chave: Sonda – MSP430 - Instrumentação – Hidrologia – pH – Oxigênio Dissolvido

ABSTRACT

The objective of this work is the improvement of some circuits of a probe for measurement of physical and chemical variables that index water quality, based on isolation of the signal conditioning circuits of pH, dissolved oxygen, conductibility, temperature, turbidity and pressure. The signals are processed by low power microcontrollers that communicate using I2C standard, optically isolated, with a central microcontroller that manages information and send it to a computer. Each sensor with a signal conditioning circuits and a processing unit constitute a module with independent power source, from DC/DC convertors. The information sent to computer is displayed on software developed exclusively for the probe, to capture data. Tests were performed in laboratory with the pH and dissolved oxygen sensors, with the ability of checking the data on PC. The results were compared with defined values and with used buffers.

Keywords: Probe – Microcontroller – Instrumentation – Hydrology – pH – Dissolved Oxygen.

LISTA DE FIGURAS

LISTA DE SIGLAS

ACK	*Acknowledge*
AD	Analógico/Digital
ADC	*Analog-to-Digital Converter*
CI	Circuito Integrado
CPU	*Central Processing Unit*
DC	*Direct Current*
DCO	*Digitally Controlled O*scillator
DMA	*Direct Memory Access*
FET	*Flash* Emulation *Tool*
I2C	*Inter-Integrated Circuit*
JTAG	*Joint Test Action Group*
LCD	*Liquid-Crystal Display*
LQFP	*Low Profile Quad Flat Pack*
MIPS	*Million Instructions per Second*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PWM	*Pulse Width Modulation*
QFN	*Quad Flat No-Lead*
RAM	*Random Acess Memory*
RISC	*Reduced Instruction Set Computer*
SCL	*Serial Clock*
SDA	*Serial Data*
USART	*Universal Asynchronous Receiver/Transmitter*

SUMÁRIO

1. Introdução

O presente projeto consistiu no aprimoramento de alguns dos circuitos utilizados em uma sonda para monitoramento da qualidade da água [1]. O microcontrolador usado anteriormente foi substituído por um modelo da *Texas Instrumentos*, da família MSP430 que deverá simplificar e melhorar os circuitos eletrônicos do instrumento. Alguns dos circuitos que eram utilizados na versão anterior da sonda foram implementados para o microcontrolador MSP430 [2].

O projeto de sonda desenvolvido anteriormente é composto de um único microcontrolador central modelo PIC. Os circuitos condicionadores de sinais estão organizados em duas placas de circuito impresso, com alimentação independente conseguida com um transformador com quatro secundários. Os sinais dos sensores dos circuitos condicionadores são multiplexados e selecionados a partir de chaves analógicas controladas pelo PIC. O sinal antes de ser processado passa por um circuito conversor analógico-digital e então é enviado ao microcontrolador, para este se comunicar com um microcomputador através da comunicação serial.

O atual projeto propõe para evitar a interferência entre os sensores, o desenvolvimento de módulos independentes para cada sensor, com alimentação independente e com os dados isolados opticamente, utilizando um barramento comum de comunicação baseado no padrão I2C.

A arquitetura prevista para os módulos inclui além dos circuitos de condicionamento de sinal, uma alimentação independente (a partir de uma bateria de 3V no próprio circuito ou de um sinal DC isolado das outras fontes do sistema) à sonda com monitoramento da temperatura, pressão, pH, oxigênio dissolvido, turbidez e condutividade.

A sonda tem a finalidade de estudo e verificação das características físicas e químicas da água, com aplicações em diversas áreas como: o controle da qualidade das águas de abastecimento, caracterização dos efluentes líquidos de águas residuárias industriais ou domésticas e caracterização de diferentes corpos de água naturais, como os rios, lagos ou reservatórios.

O trabalho proposto consistiu no projeto, construção e avaliação de uma sonda aquática capaz de medir alguns parâmetros necessários para sua qualidade e controle. Escolheu-se desenvolver instrumentos para medição e visualização de variáveis como pH e oxigênio dissolvido.

1.1. Potencial Hidrogeniônico – pH

É a medida da acidez ou alcalinidade de uma solução. O valor é definido a partir do logaritmo decimal da concentração de íons H^+ em uma solução aquosa, ou seja:

$$pH = \log_{10} \frac{1}{aH} \quad (1.1)$$

onde aH é a concentração de íons de hidrogênio ativos na solução.

O pH pode ser lido a partir de um pHmêtro, que consiste de um eletrodo acoplado a um sistema de medida. O medidor de pH é um voltímetro capaz de converter o valor do potencial do eletrodo em unidades de pH.

O eletrodo normalmente é feito de vidro e do tipo íon seletivo, ou seja, este detecta íons H+ e gera uma tensão proporcional a sua concentração. A partir desta tensão gerada é possível saber o valor do pH da solução.

A impedância de entrada de um circuito de medição de pH deve ser alta, por causa do alto valor da resistência do vidro dos eletrodos tipicamente utilizados. O circuito condicionador de sinal normalmente consiste de um amplificador operacional com configuração inversora de ganho de algumas unidades, pois o sinal captado dos sensores é da ordem de centésimos de volt.

A medição correta de um pH depende de um bom sensor e calibrações periódicas. O valor medido pode variar com o tempo e temperatura, portanto calibrações são sugeridas a cada dia de utilização. As calibrações deverão ser realizadas com a utilização de soluções com valores de pH conhecidos, *buffers*. É recomendado o uso de dois ou mais *buffers* com valores de pH entre 4 e 10.

Fabricantes de sensores para medição de pH recomendam armazenar o eletrodo sempre que não está sendo utilizado, em uma solução ácida ou em água, para casos de emergência, mantê-lo úmido. Constantes exposições do eletrodo a soluções de água destilada ou deionizada podem causar a degradação do mesmo.

1.2. Oxigênio dissolvido

A medida do oxigênio dissolvido é de fundamental importância no estudo das águas, uma vez que o oxigênio é um elemento que interage diretamente com os seres vivos aquáticos. Sua concentração deve ser controlada, pois tanto valores baixos quanto altos, são letais para os organismos.

O conhecimento de sua concentração pode ser útil para detecção de impactos ambientais como eutrofização e poluição orgânica.

A medição de oxigênio dissolvido pode ser realizada a partir de dois métodos:

- **Eletrodo Íon seletivo:** um eletrodo inerte que se comunica com uma solução externa através de uma membrana que permite a passagem de apenas um íon a ser analisado. Este eletrodo pode ser utilizado em conjunto com um eletrodo de referência.
- **Eletrodo de referência:** é um tipo de eletrodo que mantém um potencial constante em relação a um potencial que pode ser medido de uma outra meia-pilha. O eletrodo de referência

ideal apresenta um potencial conhecido e constante em relação ao eletrodo padrão de oxigênio.

Assim como a medição do pH, a leitura do oxigênio dissolvido sofre alterações do meio externo. A temperatura da água e a variação de pressão devido à altitude, influenciam de maneira exponencial no sinal do sensor de oxigênio dissolvido. Por esse motivo calibrações periódicas também são recomendadas.

Normalmente as unidades de medida de oxigênio dissolvido são: ppm, mg/l e porcentagem (%). Para esta última unidade, o instrumento de medida deve ser calibrado em uma solução com 100% e em outra com 0% de oxigênio. Os valores em ppm e mg/l são obtidos a partir da porcentagem de oxigênio e a temperatura ambiente. Uma tabela com a solubilidade do oxigênio para a determinada temperatura deverá ser consultada [3].

1.3. Microcontrolador MSP430

O microcontrolador utilizado neste trabalho, o MSP430, fabricado pela empresa *Texas Instruments* apresenta baixo consumo e alto desempenho [4].

Suas principais características estão listadas a seguir.

- **Baixo consumo:** os MSP430 são *chips* conhecidos pelo seu consumo incrivelmente baixo (da ordem de 0, 1µA para retenção dos dados da RAM, 0, 8µA para funcionamento no modo de relógio de tempo real e cerca de 250µA/MIPS em funcionamento normal). O baixo consumo é obtido graças aos diversos modos de funcionamento da CPU.
- **Baixa tensão de operação:** podem operar com tensões a partir de 1,8V até 3,6V.
- **Alto desempenho:** utilizando um barramento de 16 bits, diversos modos de endereçamento e um conjunto de instruções pequeno, mas muitíssimo poderoso, os MSP430 permitem realizar tarefas complexas com um código bastante pequeno e rápido.
- **Conjunto de instruções ortogonais:** a disponibilidade de qualquer modo de endereçamento para qualquer instrução e qualquer operando permitem que escrevam códigos pequenos e eficientes, facilitando a tarefa dos compiladores de linguagens de alto nível como a linguagem C.
- **Número reduzido de instruções:** arquitetura RISC com apenas 27 instruções físicas (op-codes) e mais 24 instruções emuladas (variações de 27 instruções que utilizam os geradores de constantes), resultando um conjunto de 51 instruções.
- **Grande quantidade de periféricos:** os chips MSP430 contam com um conjunto bastante extenso de periféricos internos, com ênfase especial para conversores A/D de até 16 bits, conversores D/A, comparador analógico, amplificador operacional programável, controladores de DMA, temporizadores com diversos modos de funcionamento (incluindo PWM), controlador de LCD, USARTs com capacidade endereçamento, multiplicador por hardware com capacidade de

executar operações de multiplicação e acumulo, etc.

- **Facilidade de gravação e de depuração:** a utilização da interface JTAG para a gravação e depuração permite que se realize a programação e a depuração de seu *software* diretamente na placa de aplicação, sem a necessidade de utilização de equipamentos dispendiosos como emuladores.
- **Diversos encapsulamentos:** desde o diminuto QFN de 24 pinos e seus 4 x 4 mm. até encapsulamentos LQFP de 100 pinos.

Há a possibilidade do uso de osciladores de alta freqüência, baixa freqüência, oscilador por resistor, e diversas combinações, todas programadas via *software*. Consegue-se operar o dispositivo sem cristais externos, apenas com o oscilador DCO presente nos mesmos. O uso dos modos corretos de oscilador é vital para otimização do consumo de energia.

Vale lembrar também que toda a linha MSP430 utiliza internamente a arquitetura Von-Neumann, isto é, o barramento de dados e o barramento de endereçamento de memória são os mesmos (multiplexado).

1.4. Barramento I2C

A fim de aprimorar a comunicação entre os sensores e o circuito de processamento, foi realizado um estudo para que esta comunicação seja feita através do barramento I2C.

O protocolo de comunicação I2C, sistema multi-mestre escravo, foi desenvolvido pela Philips, e a partir de 1996 seu uso foi liberado ao público. Esta interface foi escolhida devido a sua simplicidade e facilidade de inclusão e exclusão de dispositivos no barramento sem afetar a comunicação com os outros dispositivos. Além de ser um padrão de comunicação síncrono, ou seja, através do sinal de *clock* é possível ter o controle da transferência, permite a comunicação com a presença de somente 2 dutos bi-direcionais: SDA e SCL [5].

Muitas vantagens podem ser atribuídas ao protocolo I2C. Destacam-se entre elas:

- Organização funcional em blocos, providenciando um simples diagrama esquemático final.
- Não há necessidade de se desenvolver interfaces. Todos os dispositivos integram as interfaces "*on-chip*", o que aumenta a agilidade no desenvolvimento.
- Endereçamento e protocolo de transferência de dados totalmente definido via *software*.
- Possibilidade de inclusão ou exclusão de dispositivos no barramento sem afetá-lo ou outros dispositivos conectados a este.
- Diagnóstico de falhas extremamente simples. O mau funcionamento é imediatamente detectado.

- Desenvolvimento simplificado do *software* através do uso de bibliotecas e módulos de *software* reutilizáveis.
- Facilidade no desenvolvimento de placas de circuito impresso, devido à quantidade de interconexões.

O protocolo apresenta as seguintes terminologias:

- *TRANSMISSOR*: circuito integrado (CI) que fornece os dados para o barramento;
- *RECEPTOR*: CI que recebe os dados do barramento;
- *MASTER:* CI que inicia e finaliza a transferência de dados, gera o sinal de *clock*, ou seja controla a transmissão;
- *SLAVE*: CI endereçado pelo *MASTER,* sem o poder de controle da comunicação, é controlado pelo *clock* do MASTER;

Apesar deste protocolo não apresentar complexidade na sua configuração, há muitos parâmetros a serem configurados e o máximo de atenção é requerido para que erros simples não sejam cometidos.

Na figura 1 tem-se a seqüência de quando algum dado deve ser enviado do MASTER ao SLAVE. SDA representa o duto de dados e SCL, a linha do *clock* do padrão de comunicação.

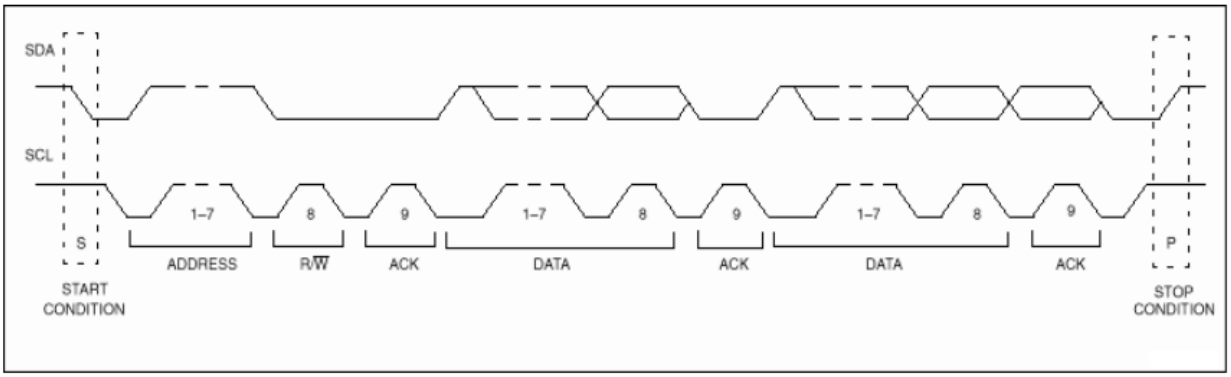


Figura 1 - Seqüência de Controle do barramento.

O padrão de comunicação é composto da seguinte seqüência para envio de dados: primeiro o MASTER deve enviar um sinal de *start* para sinalizar o inicio da transmissão. Este sinal de inicio deve ser da maneira ilustrada na figura 1: o sinal do *clock* deve estar em nível alto e então deve-se alternar o sinal de dados (SDA) para nível baixo. Após o envio do *start,* o MASTER deve mandar o endereço do dispositivo ao qual deseja se comunicar, seguido do bit que indica se ocorrerá leitura ou escrita no dispositivo (R/W).

O bit seguinte é o sinal de reconhecimento: o MASTER envia o sinal de *clock* e aguarda por um tempo definido a resposta do SLAVE ao qual se deseja estabelecer a comunicação. O SLAVE deverá enviar para nível alto o sinal do SDA, a seqüência descrita é entendida que a

comunicação está ocorrendo normalmente e deve continuar. Caso o SLAVE não responda ao sinal de *clock* do MASTER, a comunicação deverá retornar erro e ser encerrada.

Caso a opção do MASTER seja de escrita, como descrito no exemplo da figura 1, este microcontrolador deve enviar em seguida os dados desejados, respeitando a regra de que a cada um *byte* enviado, um sinal de reconhecimento (ACK) deverá ser realizado. Se a escolha do MASTER foi de receber dados do SLAVE, o MASTER deve gerar o sinal de *clock* para que o SLAVE envie suas informações sincronizadas, mantendo a regra de um sinal de ACK a cada *byte* enviado.

O encerramento da comunicação é realizado pelo MASTER, com o envio do sinal de STOP, onde o SCL deve ser levado para nível alto e em seguida, fazer o mesmo com o SDA.

A configuração elétrica do método é simples e está esquematizada na figura 2.

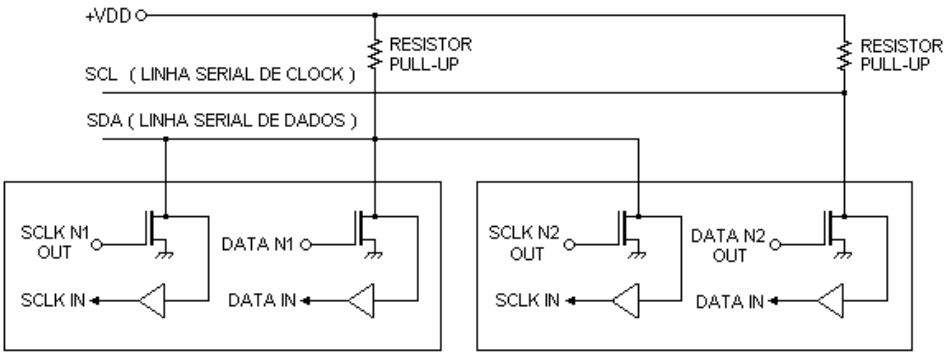


Figura 2 - Configuração elétrica do I2C.

Ambas as linhas SDA e SCL são bidirecionais e estão conectadas à alimentação via resistor PULL-UP (figura acima). Quando a barra está livre, ambas as linhas permanecem em nível alto. O estágio de saída do dispositivo conectado à barra deve possuir um coletor aberto ou um dreno aberto, para executar a função AND, como descrito na figura 2.

2. Materiais e Métodos

2.1 Circuitos condicionadores de sinal: Medidor de pH

Para a realização da medida do pH utilizou-se um sensor comercial fabricado pela empresa *Cole Parmer*, figura 3. O sensor fornece uma diferença de potencial gerada por uma célula galvânica (eletrodo + eletrodo de referência).

Figura 3: Sensor para medição de pH.

Para a confecção do circuito condicionador de sinal, obteve-se com o fabricante a resposta teórica do sensor integrado, para uma temperatura de 25°C. O gráfico obtido está representado na figura 4.

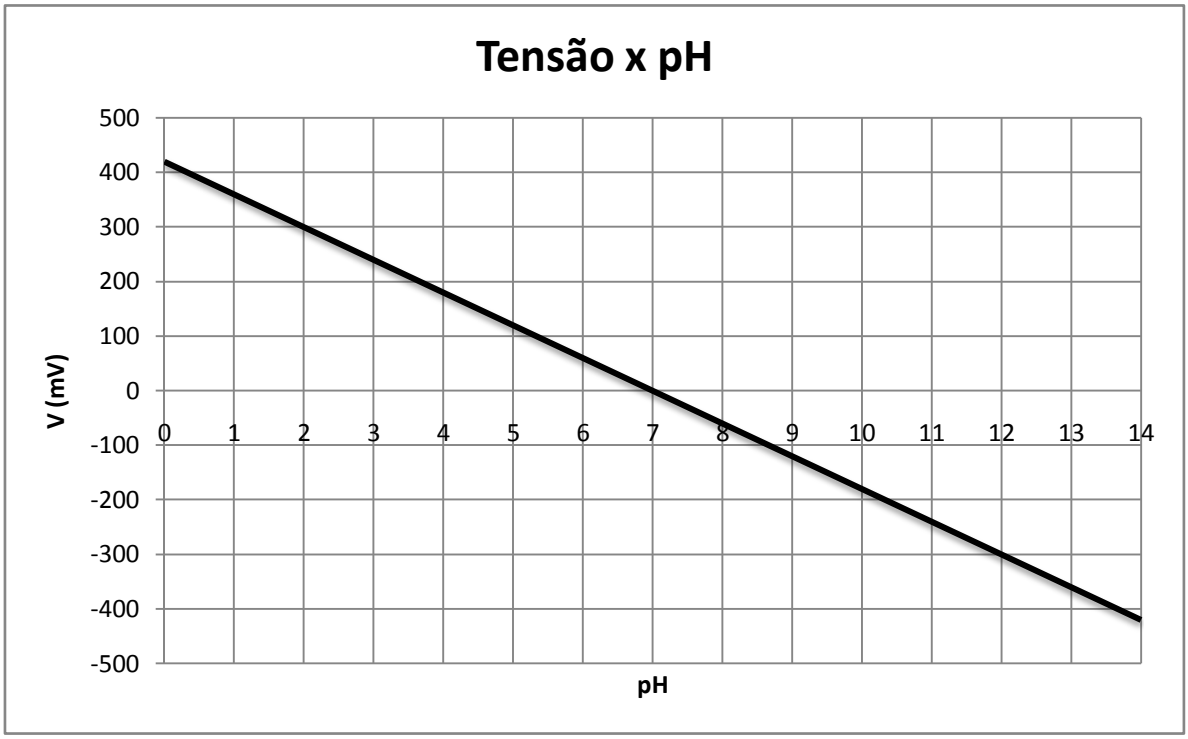


Figura 4: Resposta do Sensor a 25ºC.

O conversor AD utilizado é um comparador de 10bits presente no microcontrolador MSP 430F1232, e este aceita tensões entre 0 e VDD, 3,3V. Através da curva do sensor observa-se que a tensão produzida está compreendida aproximadamente entre -420mV e +420mV, portanto duas modificações no sinal devem ser realizadas: torná-lo apenas positivo e amplificá-lo para faixa de tensão do conversor.

A figura 5 ilustra o circuito construído com a finalidade de condicionar o sinal do sensor de pH. Como mencionado anteriormente, estes eletrodos apresentam altas resistências internas criando a necessidade de se utilizar um amplificador operacional seguidor de tensão, que apresenta uma alta impedância de entrada e permite uma baixa corrente de polarização. O segundo amplificador operacional proporciona um ganho ao sinal e soma uma componente de tensão contínua e fixa, para então gerar um sinal dentro da faixa de medida do conversor A/D do microcontrolador [6].

A tensão fixa gerada para o segundo amplificador operacional é feita com um diodo *Zenner* de 5,5V, seguido de um divisor de tensão.

O sensor de pH é conectado nos terminais Elet_pH1(fio vermelho da figura 3) e Elet_pH2 (fio preto da figura 3). O terminal S_ph, representa a saída do sinal de pH que deve ser lida pelo conversor do MSP430.

Como pode ser observada na figura 5, a alimentação do circuito apresenta o índice 2, isto foi realizado para criar a isolação na alimentação do módulo.

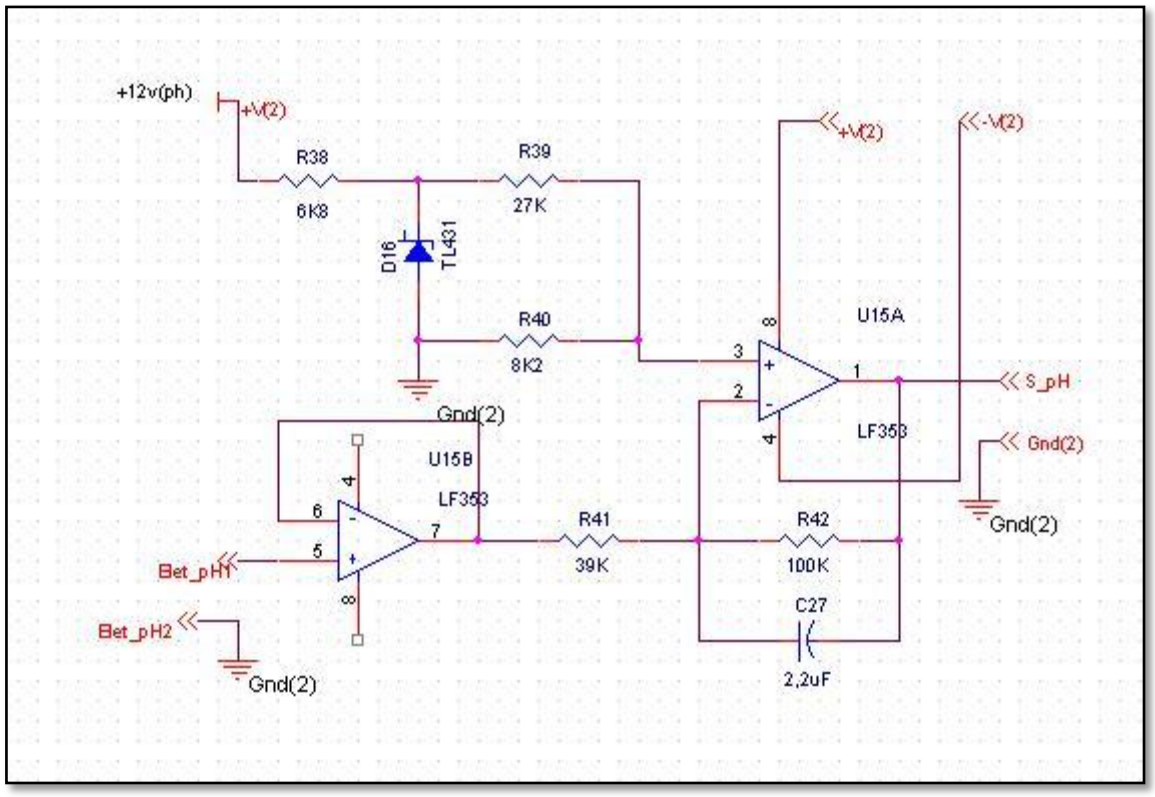


Figura 5 – Circuito condicionador de sinal: Sensor de pH.

2.2 Circuitos condicionadores de sinal: Oxigênio dissolvido

Para o presente trabalho utilizou-se o método do eletrodo de referência, como mencionado na introdução. A quantidade de oxigênio dissolvida é baseada na corrente elétrica gerada devido à diferença de potencial entre dois eletrodos imersos em uma solução eletrolítica.

Os eletrodos utilizados são feitos de um metal nobre e um deles é polarizado negativamente com relação ao outro eletrodo de referência, de prata/cloreto de prata. Ambos estão imersos em uma solução de cloreto de potássio isolada por uma membrana semipermeável. No primeiro eletrodo acontece uma redução do oxigênio e uma corrente elétrica é gerada proporcional a esta redução.

A corrente elétrica produzida com a diferença de potencial entre os eletrodos se mantém constante para valores de tensão de polarização entre 0,6 a 0,8V, portanto com a fixação deste potencial para um valor próximo de 0,7V é possível encontrar a concentração de oxigênio a partir da leitura da corrente, sendo esta proporcional a quantidade de íons reduzidos.

Na figura 6 está representado o sensor para medição do oxigênio dissolvido utilizado.

Figura 6: Sensor para medição do Oxigênio Dissolvido.

Na Figura 7 está representado o circuito condicionador do sinal; este está dividido em duas partes: à esquerda, um circuito responsável por fornecer a tensão de referência constante de 0,7V, e à direita, um conversor corrente-tensão seguido de um amplificador de sinal. Esta tensão de referência é obtida a partir de um *Zenner* de 5V e um circuito divisor de tensão. A tensão de 0,7V é responsável por criar a situação desejada, uma corrente proporcional a quantidade de íons de oxigênio produzidos.

O sensor é conectado entre a referência de 0,7V, (Elet_ox1) e a entrada inversora do primeiro amplificador operacional (Elet_ox2).

O sinal representando a porcentagem de oxigênio dissolvido é lido no pino S_ox. De acordo com o tratamento de sinal realizado, o nível de tensão na saída do circuito para uma solução com 100% de oxigênio fica em torno de 2V. Assim como encontrado no circuito condicionador de sinal do pH, este apresenta a isolação na alimentação, como pode ser visto com o índice 3.

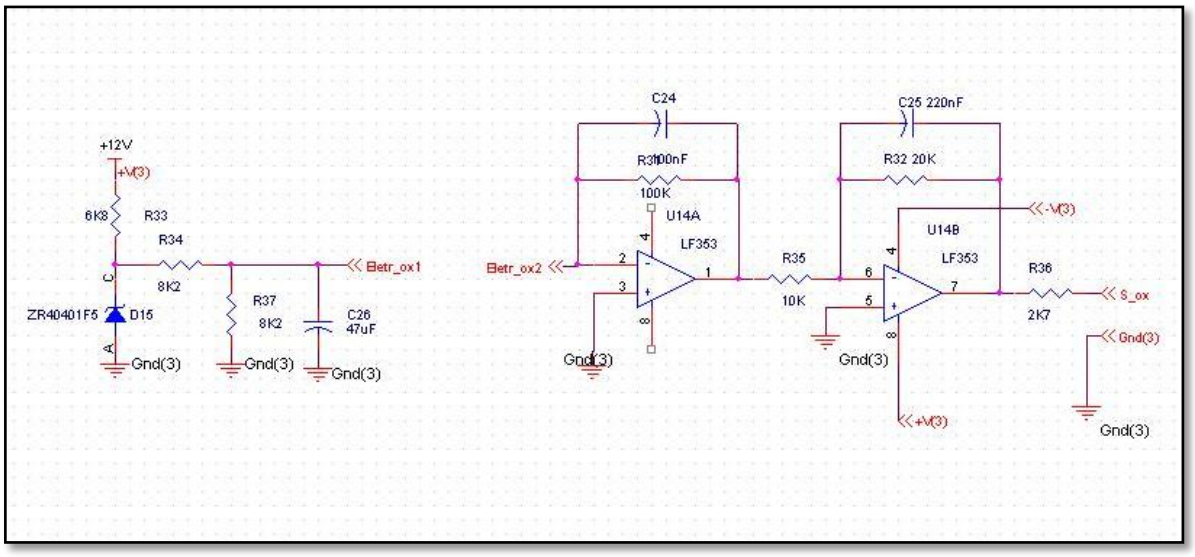


Figura 7 – Circuito condicionador de sinal do sensor de oxigênio dissolvido.

2.3. Comunicação entre os sistemas

Após a escolha do microcontrolador e do sistema de comunicação a ser utilizado, MSP430 e padrão I2C, desenvolveu-se o sistema de comunicação deste sistema ao microcomputador.

A ligação entre os microcontroladores foi realizada conforme definido pelo padrão I2C especificado pela PHILIPS [5]. Optou-se por utilizar um valor de VDD de 3,3V, e resistores de *pull-up*, 10kOhm.

A visualização dos dados dos sensores deve ser feita em um computador. Para isto é necessário o desenvolvimento de um *software* capaz de se comunicar com o MSP430. Optou-se por utilizar a comunicação serial, pela sua facilidade em configuração no microcontrolador e as opções de desenvolvimento de *software* para o PC.

Para a realização da comunicação do microcontrolador MASTER com o PC, utilizou-se o integrado MAX232, este proporciona um tratamento do sinal do microcontrolador para os níveis de tensão aceitos por qualquer computador que apresente uma porta comum serial. O sinal a ser enviado pelo MSP para o computador é lido através de dois pinos do microcontrolador e então passa pelo CI MAX232. A configuração do integrado foi feita a partir da sua folha de dados. Na figura 8 estão as ligações realizadas. Tx representa o duto de transmissão, e Rx, duto de recepção dos dados.

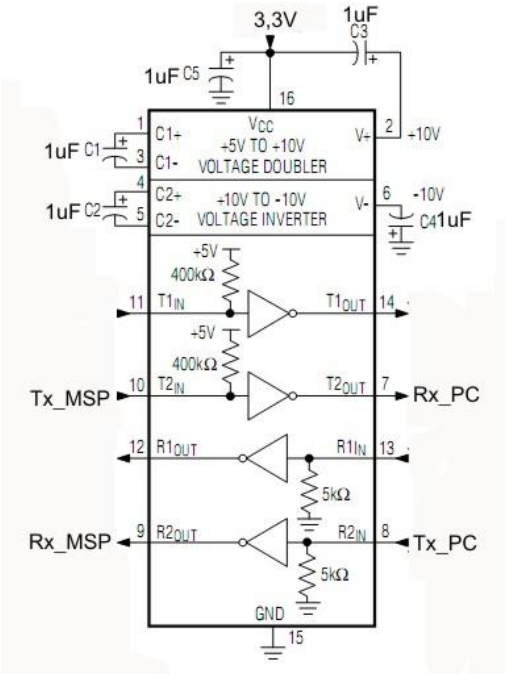


Figura 8: Ligações MAX232.

Na figura 9 está representado o esquemático de como são realizadas as ligações entre os microcontroladores e o computador, com especificação dos tipos de comunicação compartilhada entre cada sistema. O SLAVE 1 e SLAVE 2, representam microcontroladores responsáveis por receber e converter os dados dos sensores de pH e oxigênio dissolvido [7]. O MASTER é o responsável por juntar os dados dos dois sensores e enviá-los ao PC serialmente.

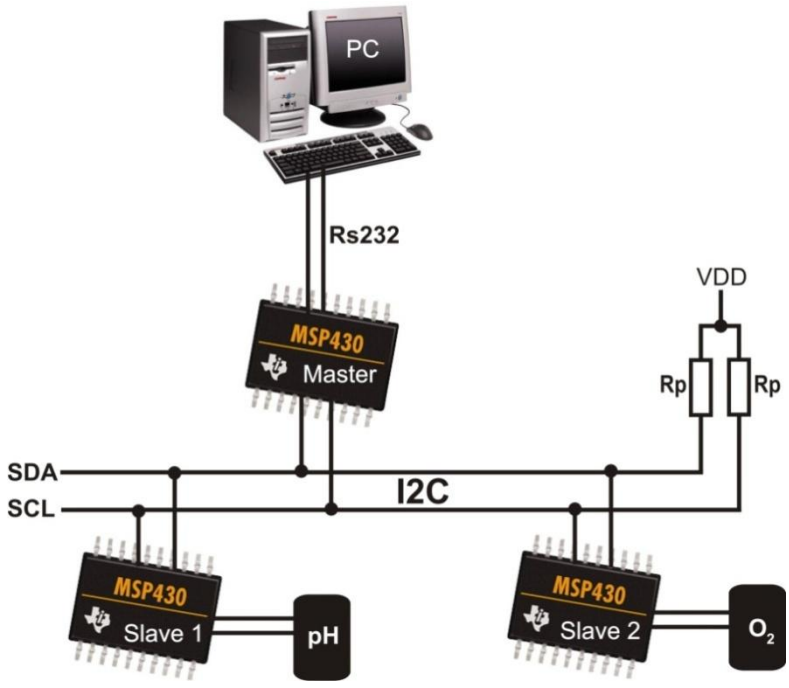


Figura 9 - Diagrama do projeto.

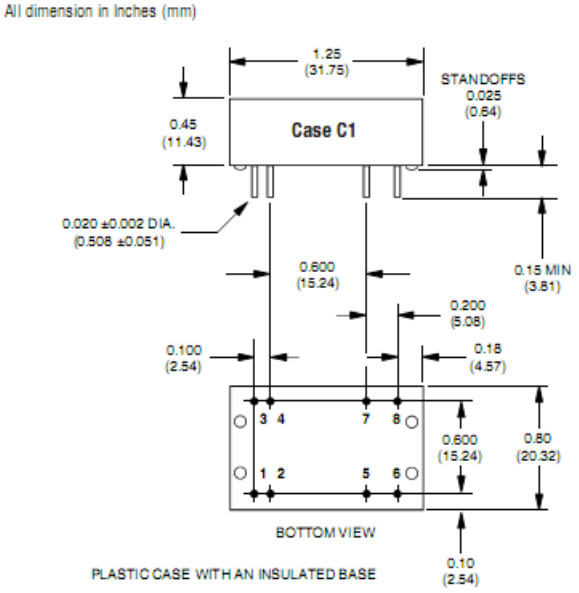
2.4 Métodos de Isolação

- **Alimentação**

Os sensores utilizados apresentam certa sensibilidade a interferências entre os sinais. Um método de isolação é necessário para que se capture os sinais desejados. A isolação de cada módulo de sensor poderá ser realizada com a utilização de conversores DC/DC para a alimentação e isoladores ópticos na comunicação I2C entre os microcontroladores. A isolação entre a alimentação dos circuitos condicionadores de sinal foi possível através da utilização de conversores DC/DC da *DATEL*.

Os conversores da *DATEL* são uma grande solução para fontes de alimentação de amplificadores, pois eles apresentam duas saídas de tensão, uma positiva e outra negativa com valores entre 5 e 15 volts dependendo do modelo escolhido. No projeto utilizou-se o modelo para saída de 15V, *BWR-15/100-D12*. Outra vantagem do circuito integrado é seu tamanho reduzido, que é de extrema importância para o projeto da sonda, pois a parte mecânica não apresenta tamanho elevado.

Na figura 10 é apresentado o conversor *BWR*.



I/O Connections	
Pin	Function P2
1	+Input
2	+Input
3	−Input
4	−Input
5	Common
6	+Output
7	Common
8	−Output

Figura 10 - *BWR-15/100-D12.*

Como a alimentação dos MSP430 é próxima de 3,3V, foram utilizados reguladores de tensão LM317, com ajustes da tensão de saída para 3,3V, a partir das tensões de entrada geradas pelos conversores DC/DC de 15V.

Na figura 11 encontra-se o esquemático da fonte feita no *software OrCAD* com os conversores DC/DC. A figura além de apresentar o sinal de alimentação dos circuitos dos sensores de pH e oxigênio dissolvido, apresenta mais dois conversores. O esquemático foi confeccionado para fornecer a alimentação dos outros circuitos condicionadores da sonda: condutividade, temperatura, pressão e turbidez. Nele pode-se observar a alimentação que irá para cada módulo de processamento. As tensões com índices ‘1’ são para a alimentação do circuito condicionador de sinal do condutivimetro, 2 são para o circuito de medida do pH, 3 para o módulo do oxigênio dissolvido, e G, para os sensores que não sofrem interferências significativas como os demais: sensor de turbidez, temperatura e pressão.

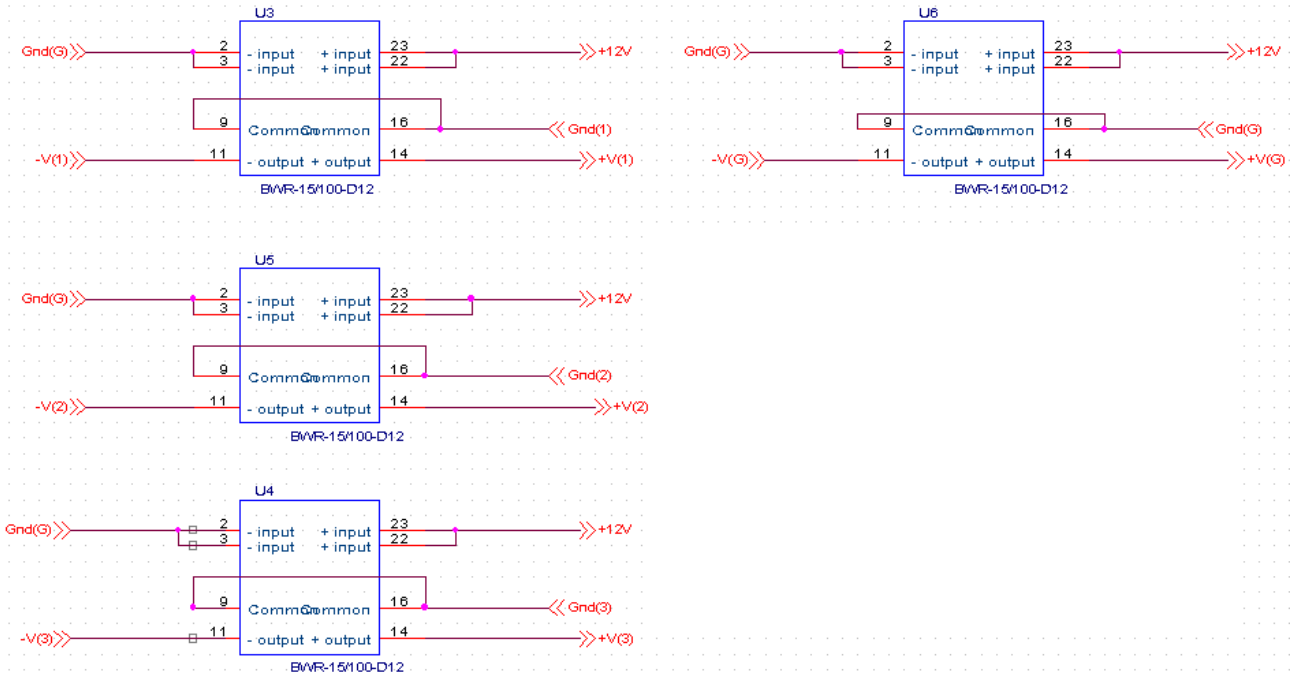


Figura 11 – Esquemático da placa de alimentação dos circuitos.

O circuito de redução de tensão para alimentação dos MSP430, está representado na figura 12. Com um sinal de entrada próximo de 15V, a saída do circuito é aproximadamente 3,3V.

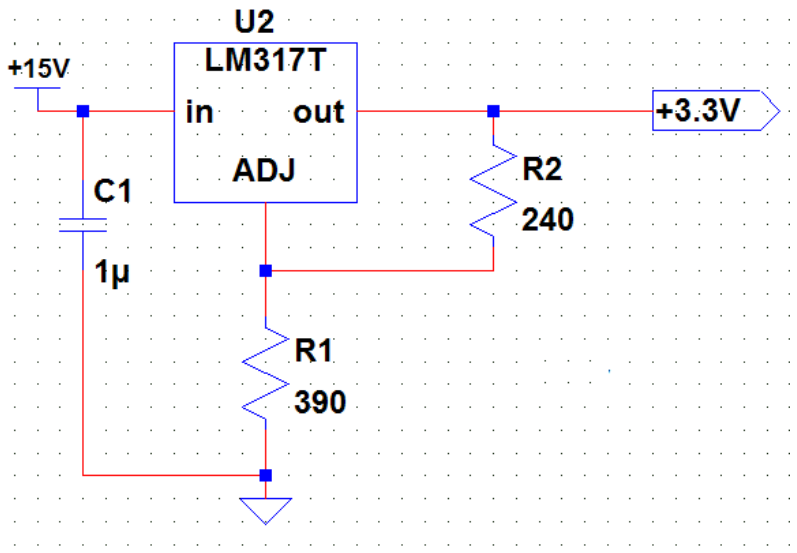


Figura 12: Circuito Regulador de 3,3V.

- **Barramento I2C**

Foi realizado um estudo a respeito de como realizar a isolação no barramento I2C, e deste modo completar a isolação de cada módulo de sensor. Há um integrado fabricado pela própria PHILIPS, criadora do barramento I2C, para uso indicado na comunicação I2C, este circuito integrado, o *P82B96,* funciona como um *buffer* para o sistema de comunicação que é bidirecional. O CI elimina o problema de *latching* e cria uma relação bidirecional entre o barramento I2C e outras configurações de barramento [5]. A transmissão dos sinais *SDA/SCL* através das linhas equilibradas de transmissão pode ser realizada com a isolação galvânica, por acoplamento óptico, de forma simples pois os sinais direcionais de transmissão (*Tx*) e de recepção (*Rx*) são fornecidos separados. Os sinais de *Tx* e de *Rx* podem diretamente ser conectados, sem causar *latching*, para fornecer uma linha de sinal bidirecional alternativa com as propriedades de I2C.

A figura 13 mostra a especificação do circuito integrado utilizado para isolamento na comunicação I2C.

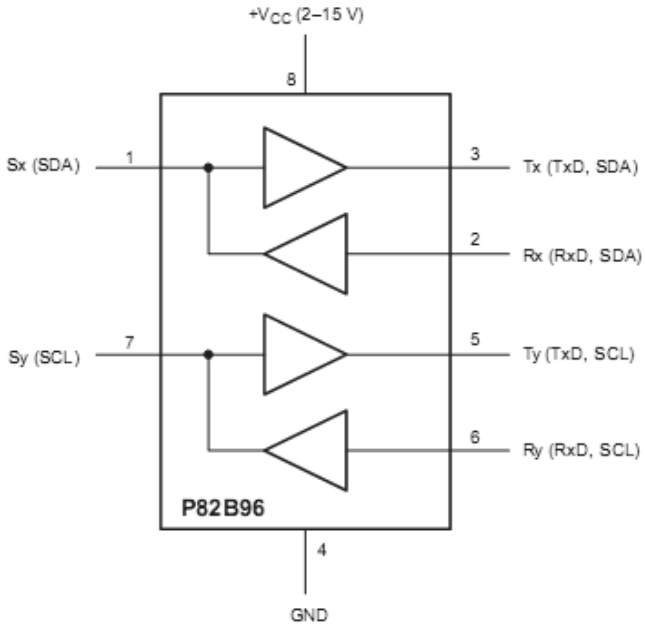


Figura 13 – CI P82B96.

Na ilustração 14 está um exemplo de como deve ser ligado o *buffer* para o caso de se utilizar o método de isolação por acoplamento óptico. No esquemático da figura pode-se observar como é realizada a ligação para se obter um duto bi-direcional.

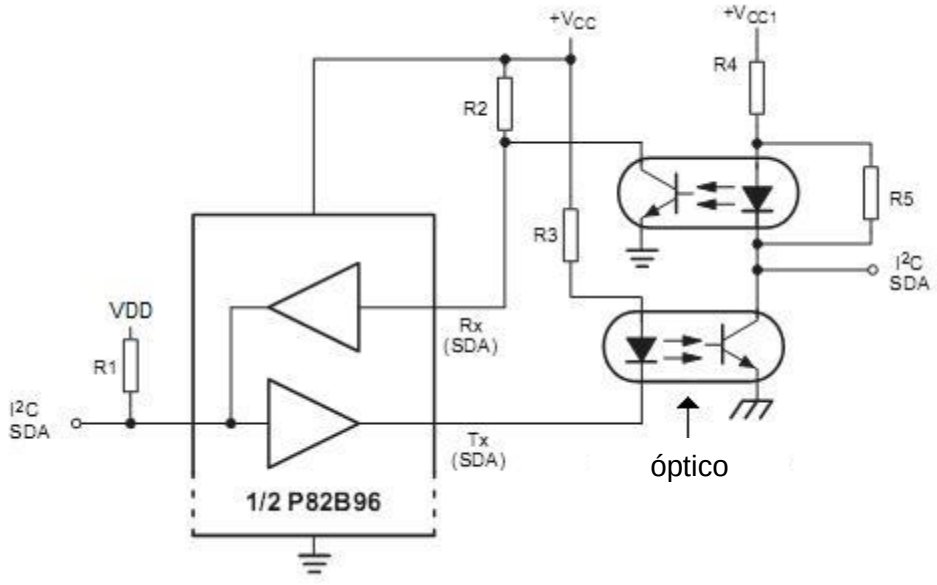


Figura 14 – Isolação do I2C via acoplamento óptico.

2.5. Confecção e montagem das placas de circuito impresso

No desenvolvimento do *hardware* foram utilizados os *softwares OrCAD*, que permitiu o desenho esquemático dos circuitos impressos e a geração dos PCB, para posterior uso no

software CircuitCam, que gera o arquivo utilizado na fresa do Departamento de Engenharia Elétrica da EESC, para confecção das placas.

Após desenho do esquema elétrico no *OrCAD Capture*, o passo seguinte foi reunir todos os circuitos numa placa e posicionar todos os componentes. A partir do *OrCAD Layout*, foi criada a forma que a placa deveria ter e também definida suas dimensões. A forma e a dimensão da placa são de extrema importância devido ao restrito espaço disponível, um dos maiores problemas encontrados. Apesar de a sonda apresentar dimensões circulares, a forma escolhida foi a de uma meia lua, pois há a parte interna dos sensores que dificulta a colocação de placas circulares. As figuras 15 e 16 ilustram a parte mecânica da sonda.

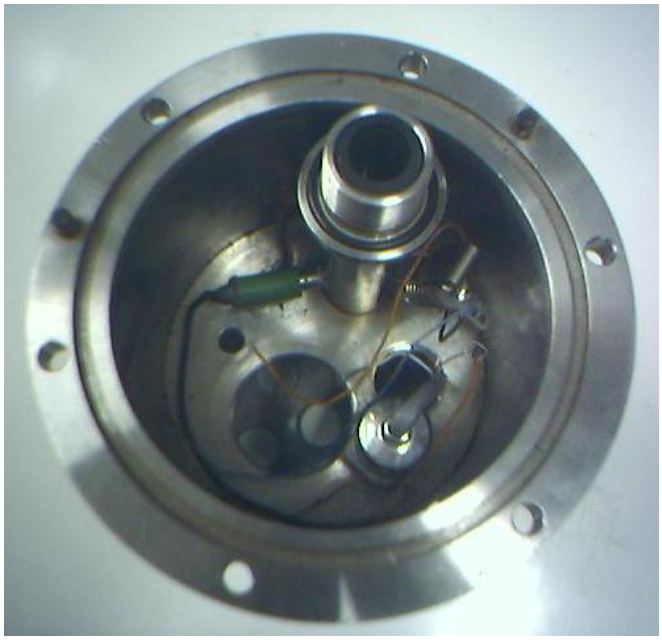


Figura 15 – Interior da sonda, vista superior.

Figura 16 – Vista externa da sonda.

A confecção das placas foi realizada durante o período de iniciação cientifica. Na época da confecção, a formação de módulos microprocessados para cada sensor não havia sido planejada, portanto a placa criada apresenta além do circuito condicionador de sinal de pH e oxigênio dissolvido, circuitos dos sensores de temperatura, pressão e turbidez. Estes circuitos condicionadores poderão no futuro ser incorporados ao barramento I2C através da utilização de microcontroladores.

As placas confeccionadas apresentam dupla face e alimentação independente entre cada circuito condicionador. Na figura 17 está o *layout* da placa confeccionada.

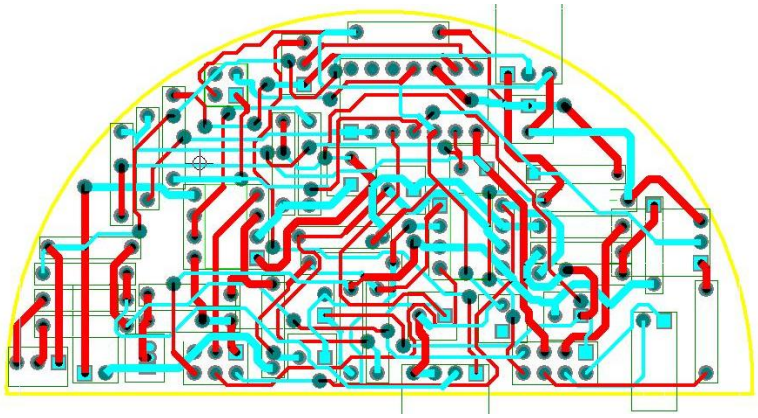


Figura 17– Layout da placa 1 – Circuitos condicionadores de sinal.

O projeto foi realizado para se ter um total de quatro placas, com a primeira contendo todos os sensores exceto o condutivímetro, este apresenta seu espaço reservado na segunda placa, por completo. A terceira placa foi destinada para a unidade de microprocessamento da sonda, como o presente projeto terá processamento isolado para cada sensor, a utilização da terceira placa foi descartada. Por fim a quarta placa que foi criada, apresenta as fontes de alimentação com os conversores DC/DC, dos circuitos condicionadores de sinal.

Para o maior aproveitamento do espaço interno da sonda, planejou-se a conexão entre as placas de maneira que cada uma pudesse ser conectada sobre a outra, por meio de *jumpers*.

Na figura 18 está a outra placa confeccionada, com as fontes de alimentação independentes para os módulos dos sensores.

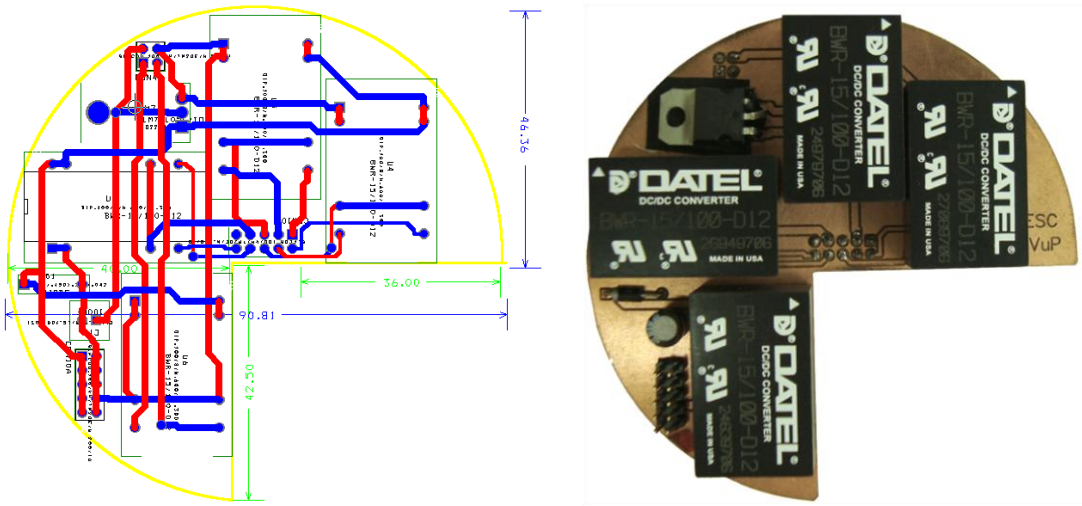


Figura 18 – Placa de alimentação dos circuitos.

A figura 19 demonstra como é realizada a conexão entre as placas.

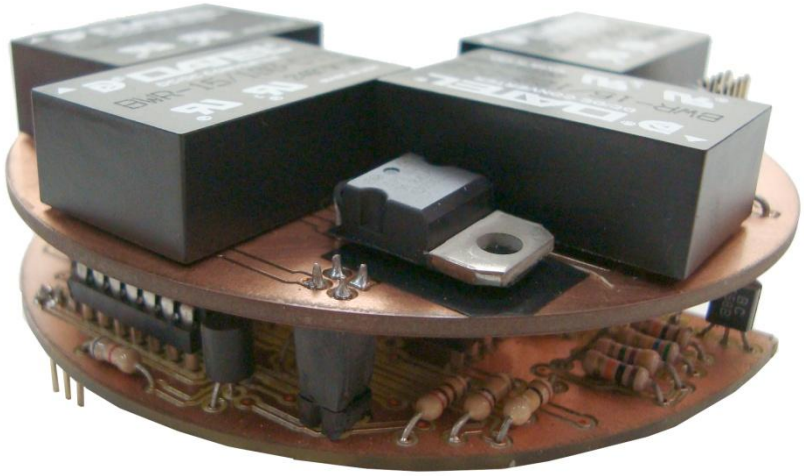


Figura 19 – Conexão entre as placas.

2.6. Programação dos Microcontroladores

Concluído o entendimento sobre o barramento I2C, começaram-se os testes com os microcontroladores. Um dos modelos escolhido foi o *MSP4301F1232*, presente no kit da *Texas*, *FET*[8]. O kit é bem simples, não apresenta nenhuma interface de interação com o programador exceto o cabo JTAG para programação e emulação do *software* com o computador. Com estas ferramentas o uso da plataforma *IAR,* que permite emular e *debugar* o *software* desenvolvido, foi muito importante. Na figura 20 esta o esquemático do kit utilizado.

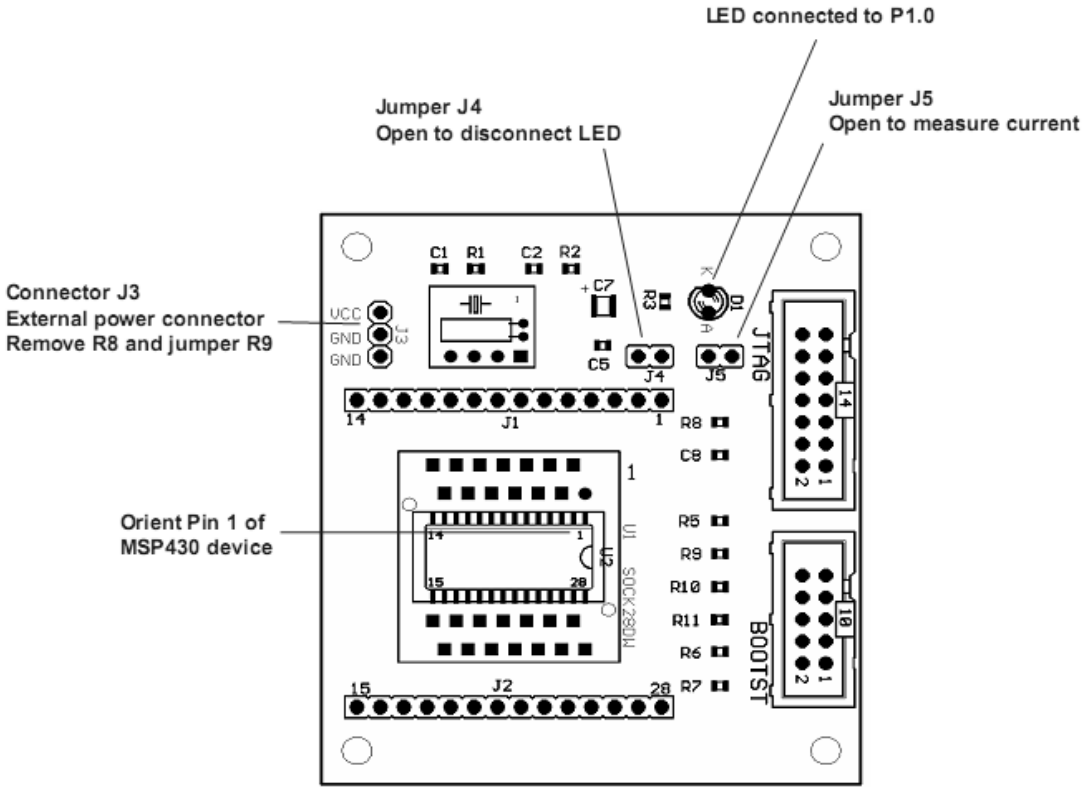


Figura 20 – Visão geral do kit MSP-TS430DW28 [9].

Através do IAR Embedded Workbench foi possível executar o programa e ao mesmo tempo visualizar no computador todas as portas e registradores do microcontrolador. A emulação permitiu também a execução do programa passo a passo e ainda a colocação de breakpoints, o qual favorece a localização de pontos críticos do algoritmo. Essa ferramenta foi muito importante para o desenvolvimento do programa, testes e correções de erros.

A programação do MSP foi feita em linguagem C com a criação de funções básicas para a comunicação I2C. O objetivo de se programar desse modo é obter uma organização do programa e facilidade de se encontrar as partes que realizem certas funções no *software*.

O teste feito com o barramento I2C foi com a implementação de um *software* em um MSP fazendo-o se comunicar com outro microcontrolador da mesma família, um MSP430F169. Esse teste foi útil para o aprendizado do I2C na prática e uma familiarização com o MSP430. Aproveitou-se a oportunidade para a realização da configuração da comunicação serial, e através do *software* livre *Tera Term Pro* foi possível enviar e receber dados para o microcontrolador MASTER.

O microcontrolador MSP430F169 utilizado para a realização dos testes está presente em um kit FET, semelhante ao anterior, não apresenta interface com o usuário, somente o acesso aos pinos do microcontrolador [10].

Os testes foram úteis para se ter uma base de como será a versão final de cada microcontrolador. A figura 21 apresenta uma noção de como os testes foram feitos.

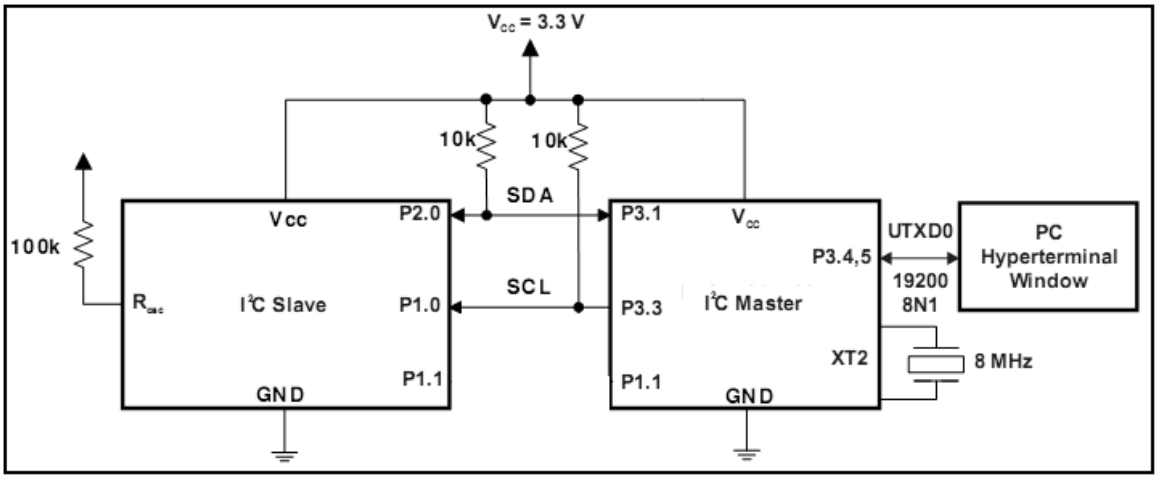


Figura 21 – Esquemático dos testes.

Após êxito na comunicação I2C e serial, os *software*s dos microcontroladores foram estruturados e definidos.

Realizou-se primeiramente a finalização dos *software*s dos MSP SLAVE 1 (pH) e SLAVE 2 (oxigênio dissolvido). Os microcontroladores são responsáveis por ler os dados do sensor por uma porta analógica, e realizar a conversão com um ADC de 10bits. E então enviar as informações dos sensores quando o microcontrolador MASTER solicitar.

A leitura e conversão dos dados dos sensores são realizadas de maneira a aproveitar toda a resolução do conversor, para a faixa de tensão entre 0 a 3,3V. O endereço de cada microcontrolador foi definido de acordo com a tabela 1:

Tabela 1: Endereços para comunicação I2C.

Microcontrolador	Endereço I2C
pH Slave	A9
Ox Slave	A0

O *software* SLAVE 1 e o SLAVE 2 são muito semelhantes, apresentam a única diferença de endereço na comunicação I2C.

A adequação do valor convertido para o aproveitamento de toda a resolução dos conversores AD, é realizada da seguinte forma:

```
Conv = Conv*255/1024;          // Ajuste valor lido ADC (0 -> 3,30V)->(0 -> 255)
```

O valor 1024 representa a resolução do conversor, e o 255, o parâmetro para restringir o tamanho da conversão em um *byte*.

Tanto o SLAVE 1 quanto o SLAVE 2 contém as seguintes funções, além da função principal:

- **Delay** – função responsável por criar pausas no *software* e temporizar a comunicação I2C;
- **Read*Byte*** – função que recebe um *byte* da comunicação I2C;
- **Read*Byte*Aux** – auxilia na recepção de um *byte*, com o tratamento bit a bit;
- **Send*Byte* –** função que envia um *byte* da comunicação I2C;
- **Send*Byte*Aux –** auxilia no envio de um *byte*, fazendo-o bit a bit;
- **ACK –** função que realiza o envio do sinal de reconhecimento, indicando êxito na comunicação I2C;
- **Interrupt** – tratamento da interrupção do conversor AD;

Os fluxogramas dos *software*s do SLAVE 1 e 2 estão nas figuras 22 e 23.

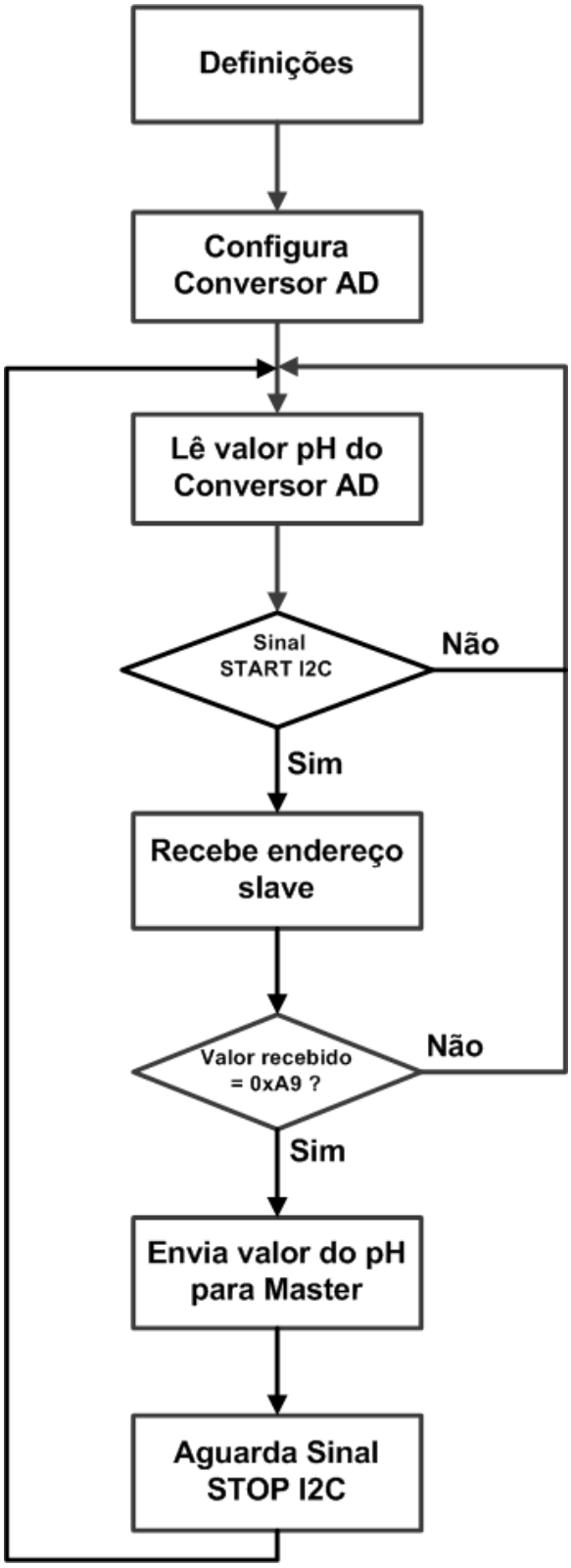


Figura 22: Fluxograma do microcontrolador SLAVE 1 - pH.

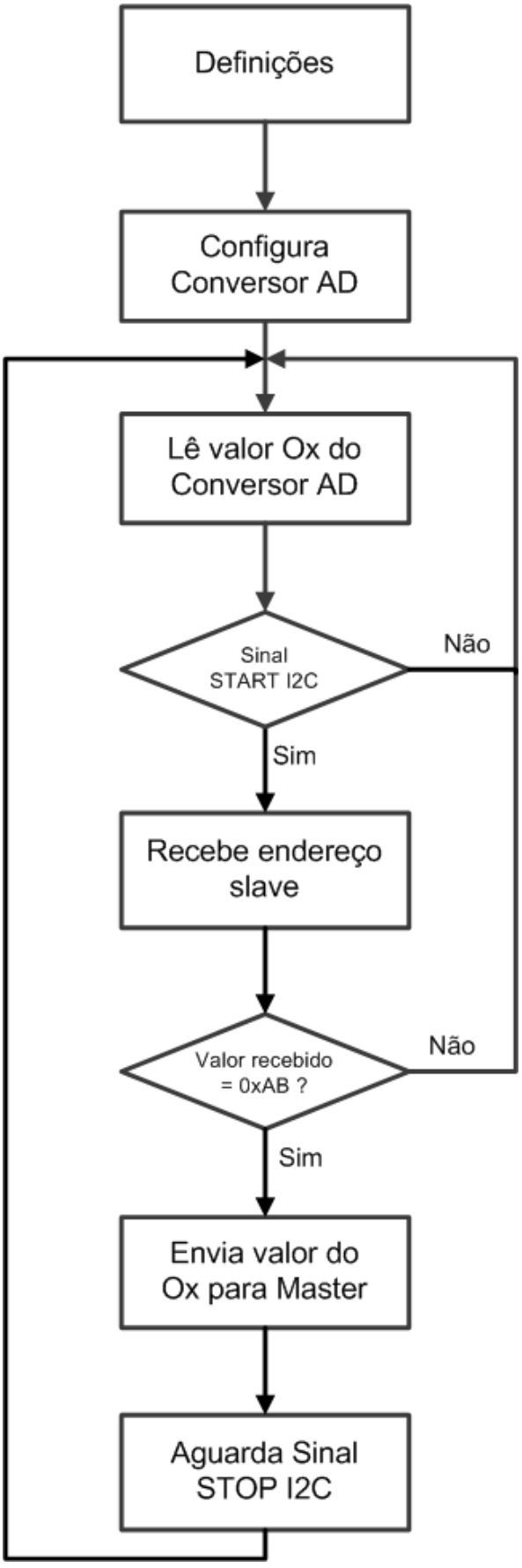


Figura 23: Fluxograma do microcontrolador SLAVE 2 - Ox.

O microcontrolador MASTER tem por objetivo realizar a comunicação com os *slaves* e com um microcomputador via serial. Seu *software* foi feito para começar a operar após um sinal do *software* do computador. A partir de um sinal de start (“V”, ou 0x56), o MASTER envia uma solicitação de leitura ao SLAVE 1 e em seguida ao SLAVE 2. Após recebimento dos dados com segurança, estes são enviados ao *software* do computador, para então serem exibidos na tela.

O sinal transmitido serialmente para o microcomputador é composto primeiramente do sinal do sensor de pH, e em seguida o sinal do sensor de oxigênio dissolvido é enviado.

O *software* do MASTER apresenta as seguintes funções, além da função principal:

- **InitUART0** – configuração da comunicação serial com a seleção de pinos a serem utilizados, *baud-rate* e habilitação de interrupção para quando ocorrer sinal no RX (pino de recebimento da serial);
- **SendStop** – envia o sinal de STOP: subida de borda do SDA com o SCL em alta;
- **__interrupt void usart0_rx** – tratamento da interrupção da comunicação serial;
- **Delay** – função responsável por criar pausas no *software* e temporizar a comunicação I2C;
- **InitI2C** – Inicialização da comunicação I2C com o envio de um sinal de STOP e configuração dos pinos para o SDA e SCL;
- **Read*Byte*** – função responsável por enviar o sinal de START da comunicação I2C, o endereço do microcontrolador selecionado, e receber em uma variável de saída a resposta de um dos slaves.
- **SendStart** – envia o sinal de START: descida de borda do SDA com o SCL em alta;
- **Send*Byte*Aux –** auxilia no envio de um *byte*, fazendo-o bit a bit. Utilizada para envio do endereço do slave;
- **ACK –** função que realiza o envio do sinal de reconhecimento, indicando êxito na comunicação I2C;
- **Read*Byte*Aux** – auxilia na recepção de um *byte*, com o tratamento de bit a bit;

Na figura 24 está ilustrado o fluxograma do *software* do Master.

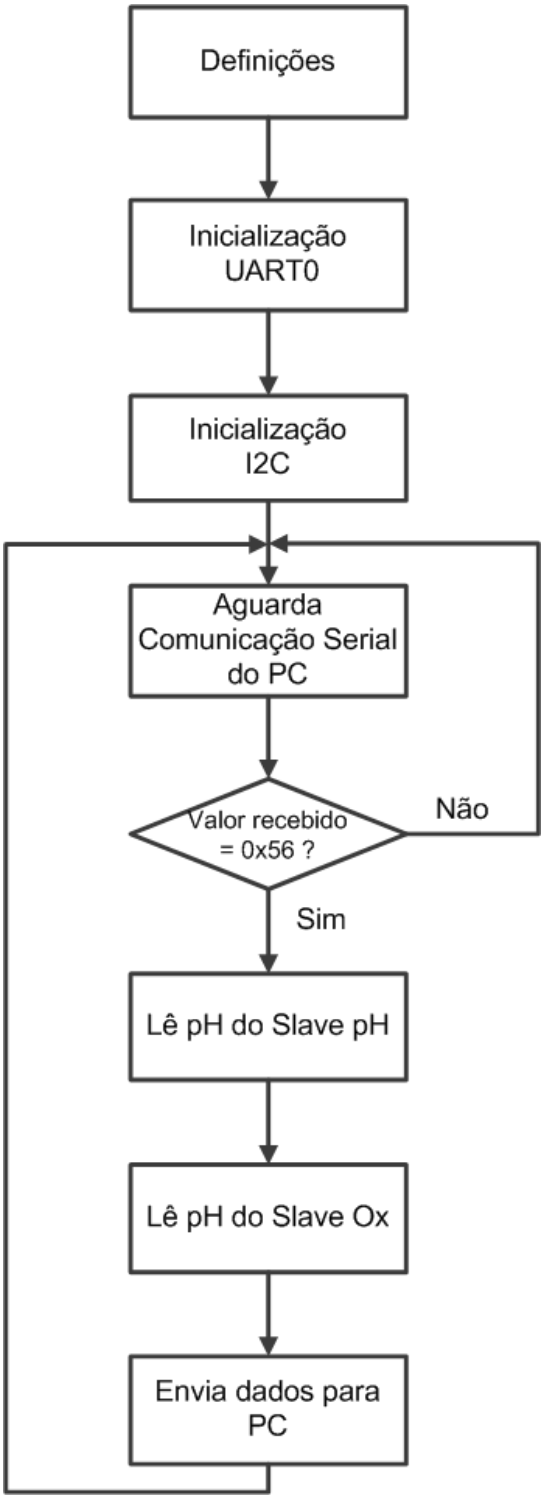


Figura 24: Fluxograma do microcontrolador Master.

2.7. Interface Microcontrolador-PC

Após leitura e tratamento dos sinais dos sensores, deve-se ter uma interface de visualização dos dados captados. Optou-se por utilizar um microcomputador para esta tarefa. Através da comunicação serial, o MSP430 transmite seus dados ao PC, e este os mostra em um

display através de um programa especificamente desenvolvido para esta aplicação.

Todo o controle da comunicação é desenvolvido pelo *software* presente no computador. Este envia um pedido ao microcontrolador MASTER a cada segundo, e o MSP430 MASTER, configurado para interrupção da porta serial, ao receber a solicitação, inicia o processo de pedido dos sinais dos sensores, enviando primeiramente para o microcontrolador do sensor de pH, SLAVE 1, e posteriormente para o medidor de oxigênio dissolvido, SLAVE 2. Após recebimento dos dados dos sensores, o microcontrolador MASTER envia as informações ao *software* do PC.

O *software* do computador é capaz de receber serialmente os dados do microcontrolador, e apresentá-los ao usuário do sistema, com a possibilidade de escolha de quando ligar/desligar a recepção dos dados dos sensores.

O *software* para recebimento e visualização dos dados dos sensores no microcomputador foi implementado com o auxílio do *software Borland Delphi 7* [11]. O programa desenvolvido foi estruturado para apresentar além da tela de trabalho, com a possibilidade de visualização dos dados e controle da comunicação serial com o MASTER, uma tela para calibração do pH e outra do oxigênio dissolvido. Há também a opção de salvar ou carregar uma calibração realizada.

O *software* Monitor foi desenvolvido em linguagem pascal, onde para cada objeto/menu presente, há um procedimento para tratamento da ação dependendo do evento ocorrido. Por exemplo, quando se executa o botão *Button1*, este troca seu nome de exibição, com a intercalação entre “Conectar” e “Desconectar, altera a mensagem na barra de *status* da janela que indica a situação da comunicação, e habilita ou desabilita o *timer* responsável por controlar a comunicação serial com o microcontrolador MASTER.

A comunicação serial é configurada com o auxílio de um objeto próprio para se trabalhar com a porta COM do microcomputador. O objeto *ComPort* facilita a configuração da velocidade de comunicação, seleção da porta utilizada, abertura/fechamento da COM, configuração do bit de paridade, *start* e *stop* bit. É neste objeto que pode ser visualizado a *flag* de recebimento, *buffer* de dados, e os eventos possíveis com a comunicação serial.

Com a utilização de um *timer* é possível realizar a configuração para que o *software* realize a solicitação dos dados dos sensores automaticamente a partir de um tempo estipulado.

O Delphi apresenta a característica de permitir a criação de uma interface amigável e prática ao usuário, permitindo o fácil manuseio das ações. A possibilidade de criação de menu deixa-o com semelhanças ao Windows [12].

O *software* Monitor pode ser executado em qualquer microcomputador, foi criado um arquivo executável que permite sua execução com a única restrição de que o PC deve apresentar pelo menos uma porta COM para a comunicação com o microcontrolador Master.

Na figura 25 esta ilustrada a tela de trabalho do *software* Monitor.

Figura 25: Tela de trabalho do *software Monitor*.

O menu “Arquivo” do Monitor contém três opções: Abrir Calibração, Salvar Calibração, e Fechar o programa. No menu “Calibração” é possível acessar os formulários para a calibração do pH e do oxigênio dissolvido.

Tanto a calibração do pH quanto do oxigênio dissolvido podem ser realizadas da seguinte maneira:

- pH – Dois buffers com pHs conhecidos têm seus valores de tensões lidos, e através destes pontos, uma equação de reta é definida para que se possa determinar qualquer valor de pH possível, equação 2.1. Para quando somente um único ponto de calibração é realizado, utiliza-se uma constante angular de m = 1/7 para a equação, obtida a partir do gráfico da figura 15, e o pH é encontrado a partir da equação 2.2.

$$pHx = \frac{Vyp(pH2 - pH1) - V1(pH2 - pH1) + (Vp2 - Vp1)pH1}{Vp2 - Vp1} \qquad (2.1)$$

$$pHx = \frac{Vy - Vp1 + pH1}{1/7} \qquad (2.2)$$

Onde:

- pHx = valor do pH desejado;
- Vyp = valor da tensão lida para o pH desejado;
- pH1 e pH2 = valores de pH conhecidos, *buffers*;
- Vp1 e Vp2 = valores de tensão para pH1 e pH2;

Casos onde não há calibração de nenhum ponto, a equação 2.2 é utilizada e com os valores para Vp1 = 2,15V e pH1 = 7.

- Oxigênio dissolvido – É realizada a leitura de tensão para uma solução de água destilada, esta com aproximadamente 0% de oxigênio dissolvido. Uma solução com água mineral deverá ser borbulhada com ar, com a utilização de uma bomba de aquário por exemplo, e seu sinal no sensor capturado. Com as duas leituras anteriores determina-se a equação (2.3) de reta e estabelece-se uma relação entre a porcentagem de oxigênio dissolvido e o valor de tensão lido.

$$\boldsymbol{Ox = \frac{Vyo(Ox2 - Ox1) - Vo1(Ox2 - Ox1) + (Vo2 - Vo1)Ox1}{Vo2 - Vo1}} \qquad \boldsymbol{(2.3)}$$

Onde:

- Ox = porcentagem do oxigênio dissolvido;
- Vyo = valor da tensão lida para a porcentagem do oxigênio desejada;
- Ox1 e Ox2 = valores de oxigênio dissolvido conhecidos;

Vo1 e Vo2 = valores de tensão para Ox1 e Ox2;

O sensor de oxigênio dissolvido é muito sensível e apresenta variação de leitura, portanto é necessário realizar a calibração do mesmo, ou no mínimo utilizar uma calibração realizada anteriormente para melhor leitura.

Nas figuras 26 e 27 estão as duas telas para calibração dos dados lidos dos sensores.

Figura 26: Tela de calibração do pH.

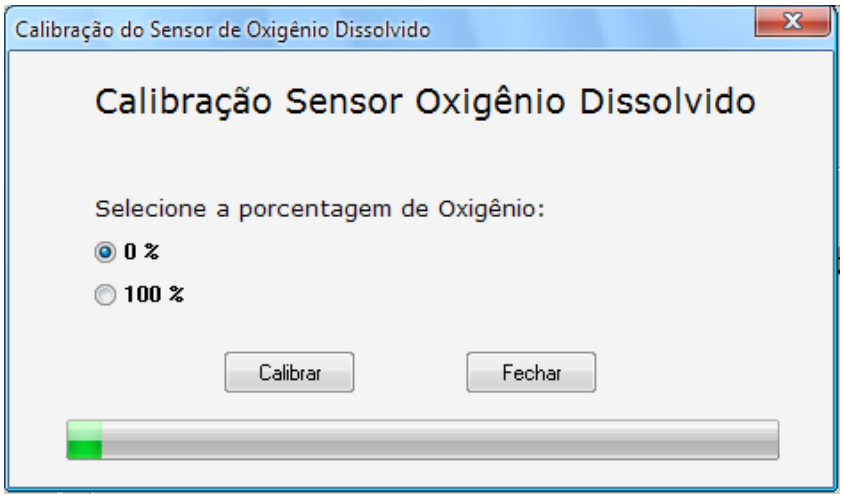


Figura 27: Tela de calibração do Oxigênio Dissolvido.

As rotinas de calibração são executadas após abertura de alguma das telas de calibração, deve-se selecionar o *buffer* utilizado. Para a calibração do sensor de pH, o valor do *buffer* deve ser digitado, com uma casa decimal e com a utilização da vírgula para separação, na caixa de texto presente. Feita a seleção do *buffer*, é necessário executar o botão 'Calibrar' para que o *buffer* escolhido seja lido. Após a barra ser totalmente carregada, pode-se selecionar o outro *buffer* a ser calibrado, e repetir os procedimentos anteriores. Para finalizar a calibração, deve-se executar o botão 'Fechar'.

Quando se utiliza o botão 'Calibrar', o *software* atribui valor '1' para as *frags* de referência para cada sensor, habilita a comunicação serial e o *timer,* que receberá os dados da porta serial automaticamente.

No menu 'Arquivo', é possível carregar uma calibração realizada anteriormente ou salvar uma calibração, como a realizada acima. A calibração é salva em um arquivo texto ".txt", cuja composição é ilustrada na tela da figura 28:

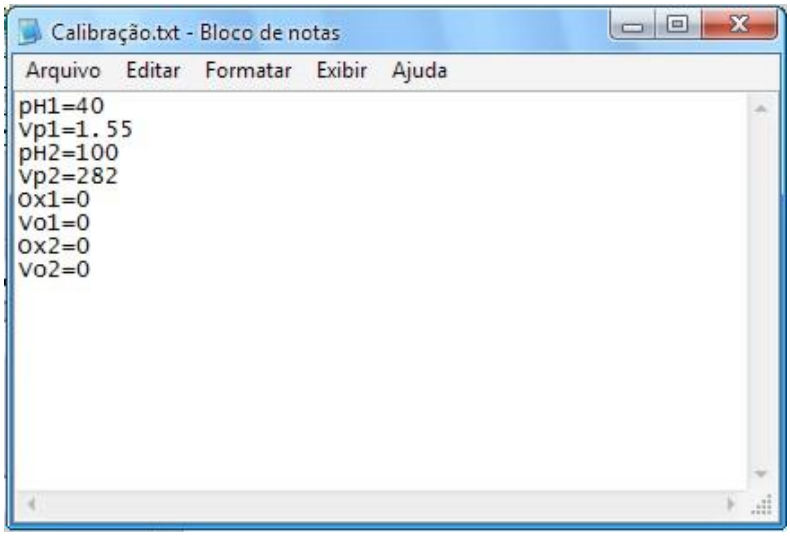


Figura 28: Tela com dados de calibração.

O programa tem seu início de operação quando o usuário executa o botão 'Conectar/Desconectar', então o *timer* responsável pela comunicação serial é habilitado e a cada segundo, o sinal com a solicitação dos dados dos sensores é enviado à porta serial. Na caixa de texto, com o valor "COM1", da figura 29, é possível escolher qual porta serial do computador será utilizada na comunicação.

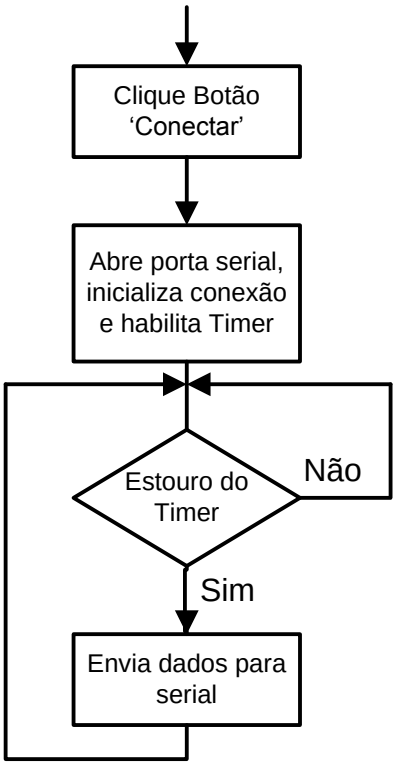


Figura 29: Fluxograma do envia de dados para porta serial.

Após o envio da solicitação para a porta serial, o programa aguarda até a recepção. Quando esta ocorrer, é verificado se é o primeiro *byte*, caso positivo o valor da tensão é carregado e a partir do número de pontos de calibração existentes, uma das equações (2.1 ou 2.2) é utilizada, então o valor do pH é exibido no display. Caso seja o segundo *byte* recebido pela comunicação serial, o valor recebido representará a porcentagem de oxigênio dissolvido, e através da equação 2.3, o *display* é atualizado com o valor corrigido.

Para encerrar a leitura dos sensores, é necessário executar o botão 'Desconectar', e então o *timer* é desabilitado, a porta é fechada e a conexão serial encerra-se. Na figura 30 está o fluxograma da recepção dos dados.

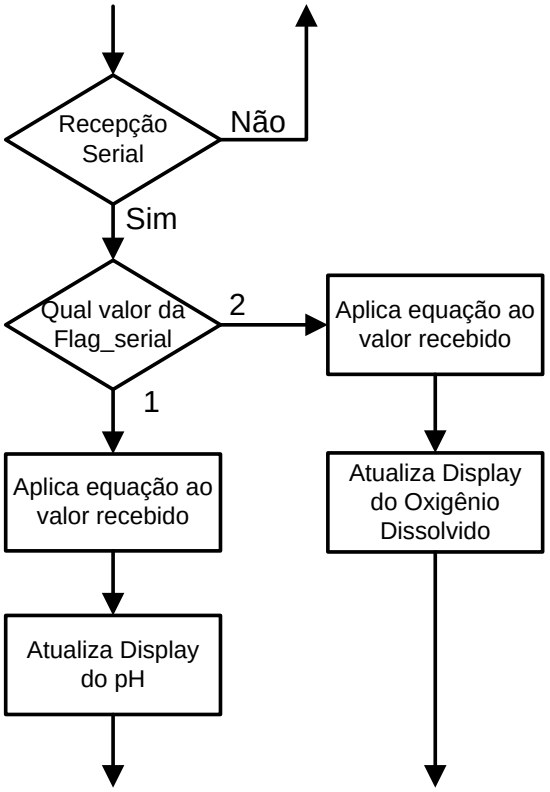


Figura 30: Fluxograma da recepção de dados da serial.

3. Resultados e Conclusões

3.1 Testes dos Circuitos

Para testar de forma eficiente e definitiva, os circuitos condicionadores de sinais foram montados em placas de teste, *protoboards*. Através delas pode-se observar como realmente os circuitos funcionam, assim como quais as dimensões que eles assumiriam. O aspecto da dimensão do circuito é de suma importância, já que o espaço que há disponível no interior da sonda é estritamente pequeno.

Durante a montagem dos circuitos foram feitas algumas alterações para torná-los fidedignos ao resultado desejado. Depois de montados, os circuitos foram conectados aos respectivos sensores. Os sensores por sua vez foram imersos em soluções com características pré-estabelecidas. De acordo com a solução utilizada, foi esperado um sinal elétrico que estivesse dentro de um patamar também pré-estabelecido. Os sinais elétricos em geral foram medidos através de multímetros digitais disponíveis no laboratório.

Com os testes foi possível avaliar a qualidade dos sinais obtidos e outras de suas características, como por exemplo, os tempos necessários para suas estabilizações.

A montagem dos circuitos condicionadores de sinais nas *protoboards* foi bem sucedida; esta atingiu os resultados esperados e mostrou diversas características do sistema que não puderam ser visualizadas nas simulações e cálculos.

O sinal de pH, após receber o tratamento do circuito condicionador, ficou compreendido entre a faixa de 1V a 3,3V, para valores de pH entre 1 e 14. Testes realizados com o circuito e o sensor permitiram coletar dados suficientes para elaboração do gráfico da figura 31, onde está representada a resposta final do circuito que deve ser convertida pelo comparador do microcontrolador SLAVE 1. Para os testes, foram lidos pH de soluções *buffers* presentes no laboratório de pesquisa, pH com valores de 4, 7 e 10.

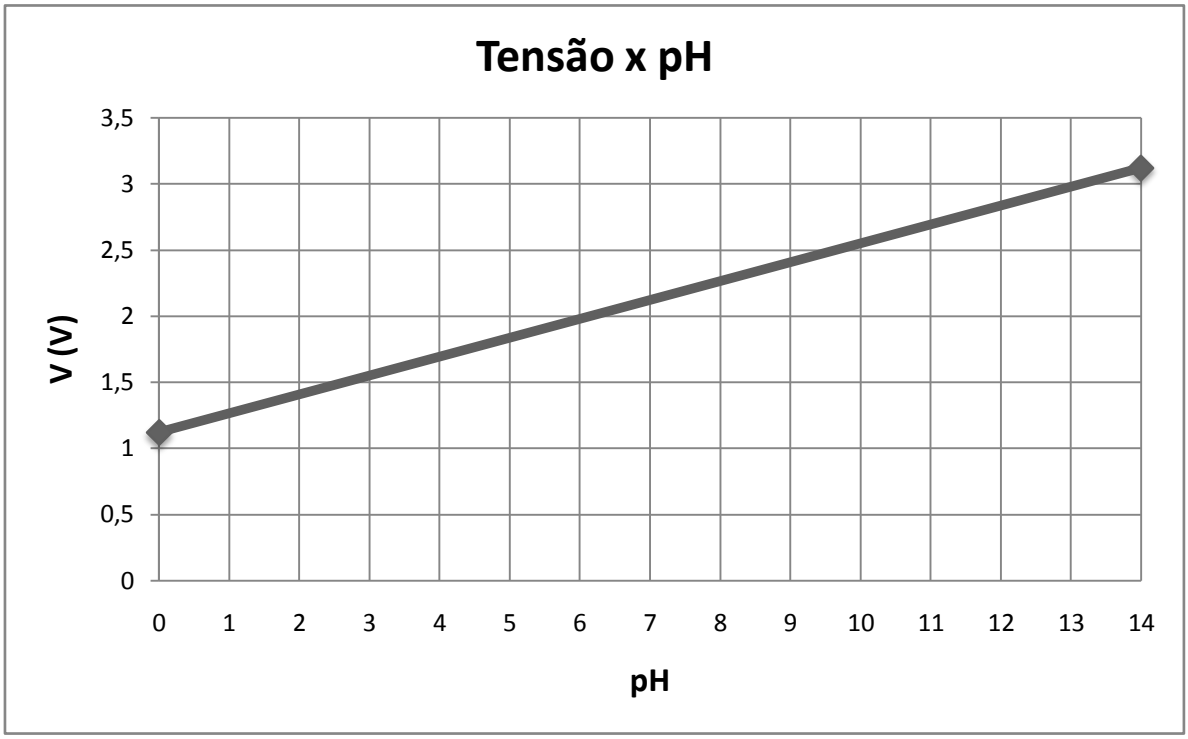


Figura 31: Sinal de pH condicionado.

O teste para o circuito condicionador do oxigênio dissolvido foi realizado de maneira similar ao sensor de pH, porém com soluções de acordo com a tabela 2.

Tabela 2: Testes com o sensor de Oxigênio Dissolvido.

Tensão de saída	Solução
0,55V	Água destilada em repouso.
2,15V	Água destilada após ar ser borbulhado por 10 minutos.

O sucesso nos testes dos circuitos permitiu a criação das placas de circuito impresso.

3.2. Testes dos Microcontroladores

Realizaram-se testes para a comunicação I2C com apenas dois microcontroladores de cada vez, um *slave* e outro *master*, pois se dispunha de apenas dois MSP. Para que todos os *software*s fossem testados, primeiro um dos microcontroladores recebeu o *software* MASTER e o outro, o *software* do SLAVE 1 (pH). A parte referente à solicitação dos dados ao sensor de oxigênio dissolvido foi comentada. Com este teste se comprovou a eficiência da comunicação I2C com o entendimento da mesma por parte de ambos os *softwares.*

Em seguida realizou-se o teste com o SLAVE 2 (Ox) e o MASTER, resultados comprovaram que a comunicação foi bem sucedida e que o MASTER recebeu o valor esperado.

O método de isolação da comunicação I2C não foi testado por falta do integrado responsável por auxiliar na isolação, o P82B96.

O *software* Monitor não funcionou de maneira adequada, onde não foi possível realizar a comunicação com o microcontrolador MASTER. Porém mesmo com o não funcionamento do *software*, o sistema com os microcontroladores e os sensores foi testado, através da utilização do programa *Tera Term Pro*, que permite o envio e recepção de dados pela porta serial do computador.

3.3. Conclusões

Ao final do projeto foi possível ter uma noção sobre a criação de um projeto desde seu planejamento até sua conclusão. Foram vivenciadas as dificuldades presentes em um trabalho com várias etapas de desenvolvimento e a necessidade de um prazo a cumprir.

Através do trabalho com circuitos condicionadores de sinais de sensores, houve uma grande familiarização e desenvolvimento da capacidade de criação desses circuitos. Pode-se compreender o funcionamento dos sensores

O grande número de circuitos impressos e suas restrições para confecções ajudaram a desenvolver um bom conhecimento sobre o assunto e experiência para criação de qualquer outro quando for necessário.

Foi possível desenvolver um bom conhecimento com relação à microcontroladores da família MSP430, que estão presentes no mercado com grande expressividade. Houve um aperfeiçoamento da capacidade de programação com a criação de *software*s mais complexos e a implementação de protocolos e microcontroladores não vistos anteriormente nas salas de aula.

Adquiriu-se conhecimento de *software*s que auxiliam no desenvolvimento de trabalhos com o uso de microcontroladores. Com o *Embedded Workbench IAR* foi possível *debugar* um programa criado e através da emulação, encontrar erros mais facilmente que a maneira habitual.

O protocolo de comunicação I2C foi desenvolvido na prática e testado suas facilidades para quando se desejar realizar uma comunicação entre circuitos de maneira organizada e totalmente controlada por *software*. Foi nesta configuração de comunicação que se obteve maior dificuldade na programação dos microcontroladores, porém com a ajuda da disciplina SEL373 – PROJETOS DE SISTEMS DIGITAIS, esta pode ser superada com maior facilidade.

O projeto foi de extrema importância não somente para a vida acadêmica do aluno, mas também para seu desenvolvimento profissional. Obteve-se a experiência de se trabalhar como engenheiro e conhecimento de suas dificuldades.

Parte do trabalho realizado foi apresentada no SIICUSP – Simpósio Internacional de Iniciação Cientifica da Universidade de São Paulo, em Nov. 2008, onde foram demonstrados os

circuitos condicionadores de sinal, as placas confeccionadas e o estudo sobre os microcontroladores MPS430. Durante o período de Ago 2007 a Jul 2008, houve um auxílio com bolsa CNPq/PIBIC para o desenvolvimento do projeto.

A programação em DELPHI foi inédita ao aluno, para sua realização foi necessário um estudo do *software* e busca de ferramentas para auxiliarem no desenvolvimento do Monitor. Porém a interface amigável e o conhecimento da linguagem Pascal, permitiram que o trabalho proposto fosse realizado. Exceto pela comunicação serial, onde os erros encontrados não foram corrigidos.

De acordo com o que foi realizado, trabalhos futuros podem ser implementados. A utilização do barramento I2C permite que módulos dos outros sensores com sinais não tratados no presente projeto possam ser incorporados ao sistema. Para isto basta algumas alterações no *software* MASTER e adições de rotinas para o *software* Monitor. Verificações de erros na comunicação podem ser realizadas, tanto para o *software* do microcontrolador quanto para o do computador.

O método de isolação da comunicação I2C pode ser implementado facilmente e deste modo garantir a total isolação de cada módulo de sensor.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Bruno, Ronaldo, “Pesquisa em Instrumentação Eletrônica Microprocessada para a Medida de Parâmetros Físicos e Químicos da Água”, Dissertação de mestrado defendida no Instituto de Física de São Carlos/USP, 1997.

[2] Pereira, Fábio; “MICROCONTROLADORES FAMÍLIA MSP 430 – TEORIA E PRÁTICA”, 2004.

[3]Lee, Young H. Tsao, George T., “Dissolved Oxygen Electrodes”. Volume 13, Philadelphia-USA 1979.

[4] Press, N. York, 1998,Pereira, Fábio; “MICROCONTROLADORES FAMÍLIA MSP 430 – TEORIA E PRÁTICA”, 2004.

[5] Philips Semiconductor, APPLICATION NOTE I2C BUS. Disponivel em: www.nxp.com/acrobat_download/applicationnotes/AN10216_1.pdf. Ultimo acesso em 10 de out. 2009.

[6] Dailey, D.J. “Operational amplifiers: design and applications”. McGraw-Hill, 1995.

[7] Gonçalves, Vitor M.S.; “SISTEMAS ELETRÔNICOS COM MICRO-CONTROLADORES”, 2a. Edição.

[8] Texas Instruments. DATASHEET MSP430F12X2. Disponível em: http://www.ti.com/lit/gpn/msp430f1232 . Último acesso em: 05 de nov. 2009.

[9] Texas Instruments. DATASHEET FET TS430DW28. Disponível em: http://focus.ti.com/lit/ug/slau278b/slau278b.pdf. Último acesso em: 05 de maio 2009.

[10] Texas Instruments. DATASHEET MSP430F16X. Disponível em: http://www.ti.com/lit/gpn/msp430f169. Último acesso em: 05 de nov. 2009.

[11] Leão, Marcelo; “BORLAND DELPHI 7 CURSO COMPLETO”, 2003.

[12] Anselmo, Fernando;”BORLAND DELPHI – DESVENDANDO O CAMINHO DAS PEDRAS”, 1997.

Apêndice A – Código Fonte SLAVE 1 - pH

```
#include  <msp430x12x2.h>
#define       bitset(var,bitno) ((var) |= 1 << (bitno))
#define       bitclr(var,bitno) ((var) &= ~(1 << (bitno)))

#define       SDA             BIT2&P3IN       //SDA P3.2
#define       SCL             BIT3&P3IN       //SCL P3.3
#define       _1ms            2               //2 cycles *12 + 9 = 801 / 801*0.305us = 1ms ver isso
#define       SDA_ON          bitset(P3OUT,2)
#define       SDA_OFF         bitclr(P3OUT,2)
#define       SCL_ON          bitset(P3OUT,3)
#define       SCL_OFF         bitclr(P3OUT,3)
#define       MREAD           0xA9          //endereço do MSP p/ comunicação I2C 10101001

//Declaração de funções
void Delay(unsigned int a);
void InitUART0(void);
void InitI2C(void);
void SendStart(void);
void SendStop(void);
void SendByteAux(unsigned char *dado);
void ReadByteAux(unsigned char dado[8]);
unsigned char Ack(void);
unsigned char SendByte(unsigned char *valor);
void ReadByte(void);

unsigned char envia;

void main(void)
{
  unsigned char Conv=0;
  envia=0;

  WDTCTL = WDTPW + WDTHOLD;                         // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE;       // ADC10ON, interrupt enabled
  ADC10AE |= 0x01;                  // P2.0 ADC option select
  P1DIR |= 0x01;                    // Set P1.0 to output direction
  P3DIR &= ~0x04;                   // Pino SDA como entrada
  P3DIR &= ~0x08;                  // Pino SCL como entrada


        while (1)                       //repetir sempre
         {
        ADC10CTL0 |= ENC + ADC10SC;         // Sampling and conversion start
         _BIS_SR(CPUOFF + GIE);             // LPM0, ADC10_ISR will force exit

         //Ajustar valor da conversao p/ envio ao MASTER
         Conv = ADC10MEM;                                   //valor lido de 0 a 1024 - 0 a 3V
        Delay(10);
        Conv = Conv*255/1024;               // Ajuste valor lido ADC (0 -> 3,00V)->(0 -> 255)

         //testar se codigo msp p/ I2C
         if (!SDA & SCL)          //condição de START verifica se houve tentativa de comunicação
        {
                Delay(12);
                if(!SCL)        //antigo Rec_flag==0
        {
         ReadByte();            //leitura do código do SLAVE
        if (envia == 1)         //Se código do SLAVE
                {
                 Delay(6);
                SendByte(&Conv);          //Envia dados do conversor AD
                Delay(13);

                 if(SCL & !SDA)             //Sinal de STOP
                 Delay(12);
```

```
                        if(SCL & SDA) envia=0;   //Final transmissão
                        }
         }
         }
        }
 }

//Função que gera um tempo de 9+a*ciclos de clock (32kHz)
void Delay(unsigned int a)
{
  unsigned int k;
  for (k=0 ; k != a; ++k);
}

// Comunicaçao I2C
/*****************************************************************************************************
//Inicialização da comunicação
void InitI2C(void)
{
  P3DIR |= 0x07;                        //P3.2 = SDA I2C/ P3.3 SCL I2C
  SendStop();
  Delay(150);
}

void SendStart(void)                     //envio de um START
{
  SDA_OFF;
  Delay(10);
  SCL_OFF;
}

void SendStop(void)                      //envio de um STOP
{
  SDA_OFF;
  Delay(10);
  SCL_ON;
  Delay(10);
  SDA_ON;
}

//funcao responsavel pelo envio de um byte
void SendByteAux(unsigned char *dado)      //mudar essa
{
  //unsigned char bit=7;
  unsigned char i;
  i=0;
  Delay(10);

   for(i=7;i>=0;i--)
    {
            Delay(2);
            while(SCL)
                        {
                        //if(BITi&dado) //MSB
                        if(dado[i])
              SDA_ON;
                        else
                          SDA_OFF;
                        Delay(10);
                        }
   }
}

void ReadByteAux(unsigned char dado[8])     //unsigned char *dado
{
  int i;
```

```
  P3DIR &= ~0x04;         //Pino do SDA como entrada
  //ver tempo

  Delay(2);
  for(i=7;i>=0;i--)
    {
      while(SCL);
       {
         //Delay(10);
         // if(SCL);              //checar
         //Delay(10);
       if(SDA) bitset(*dado,i);     //se high, 1

       else
            bitclr(*dado,i);     // se low, 0

          //SCL_OFF;
      Delay(20);
      }
    }
}


//funcao de reconhecimento da comunicação
unsigned char Ack(void)
{
  int contador=0;       //contador responsavel por esperar ate o reconhecimento, se cheio, nao houve Ack
  //SDA_ON;
  bitclr(P3DIR,2);      //pino do SDA como entrada
  Delay(2);
  //SCL_ON;
  while(SDA && (contador<20))
            contador++;
            SDA_OFF;
  //bitset(P3DIR,2);      //pino do SDA como saida
  if(contador>19)
    return 0;
  return 1;
}

//criar função enviar ACK

//funcao responsavel por enviar um bloco de informaçoes
unsigned char SendByte(unsigned char *valor)   //revisar funcao
{
  Delay(10);
  //if(!Ack())
    //return 0;
  //Delay(10);
  SendByteAux(valor);
  if(!Ack())
    return 0;
  Delay(10);
  //Delay(250);
  return 1;
}

//funcao responsavel por leitura um bloco de informaçoes
void ReadByte(void)
{
  unsigned char valor;
  Delay(8);
  ReadByteAux(&valor);

  if(valor == MREAD)      //se eh esse msp
    {
      Delay(15);
      if(SDA)           //ack
```

```
      {
        P3DIR |= 0x04;       //volta do pino do SDA como saida
        Delay(5);
        SDA_OFF;
        P3DIR &= ~0x04;       //Pino do SDA como entrada
        envia = 1;
      }
    }
}



  /*
  SendByteAux(0xA0);
  if(!Ack())
    return 0;
  SendByteAux(0x00);
  if(!Ack())
    return 0;
  Delay(10);
  SDA_ON;
  SCL_ON;
  SendStart();
  Delay(10);
  SendByteAux(0xA1);
  if(!Ack())
    return 0;
  SDA_ON;
  ReadByteAux(valor);
  SendStop();
  */

//*************************************************************************************************


// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
   _BIC_SR_IRQ(CPUOFF);                  // Clear CPUOFF bit from 0(SR)
}
```

Apêndice B – Código Fonte MASTER

```
#include  <msp430x12x2.h>
#include "stdlib.h"
#include "stdio.h"
#include "string.h"

#define bitset(var,bitno) ((var) |= 1 << (bitno))        //atribuir True para bit
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))     //atribuir False para bit

#define         SDA             BIT2&P3IN           //SDA P3.2
#define         SCL             BIT3&P3IN           //SCL P3.3
#define         COD1             0x0A               // codigo `ENTER`
#define         SDA_ON           P3OUT |= 0x04       //P3OUT_bit.P3OUT_2 = 1
#define         SDA_OFF          P3OUT &= ~0x04      //P3OUT_bit.P3OUT_2 = 0
#define         SCL_ON           P3OUT |= 0x08       //P3OUT_bit.P3OUT_3 = 1 //bitset(P3OUT,3)
#define         SCL_OFF          P3OUT &= ~0x08      //P3OUT_bit.P3OUT_3 = 0
#define         pHREAD           0xA9               //endereço p/ leitura do MSP p/ I2C pH - 10101001
#define         OxREAD           0xAB               //endereço p/ leitura do MSP p/ I2C Ox - 10101011
```

```
//variavies globais
unsigned char Start_Trans=0;

//Declaração de funções
void Delay (unsigned int a);
void InitUART0(void);
void InitI2C(void);
void SendStart(void);
void SendStop(void);
void SendByteAux(unsigned char dado);
void ReadByteAux(unsigned char *dado);
void ReadByte(int *valor, unsigned char address);
unsigned char Ack(void);
unsigned char SendByte(unsigned char valor, unsigned char address);


//programa principal
void main(void)
{
  int ConvpH;
  int ConvOx;


  InitUART0();                  //Inicialização comunicação serial
  InitI2C();                    //Inicialização comunicação I2C

// loop infinito
//*****************************************************************************************************
  while(1)
  {
            _BIS_SR(LPM3_bits + GIE);           // Enter LP3, interrupts enaabled
            if (!(IFG2 & UTXIFG0));             // USART0 TX buffer ready?
          if (RXBUF0 == 0x56)
            {
                      ReadByte(&ConvpH,pHREAD);     //enviar e recebe sinal slaves 1
                      ReadByte(&ConvOx,OxREAD);     //enviar e recebe sinal slaves 2

                      //envio serial
                      TXBUF0 = ConvpH;              // envio pH
                      Delay(50);
                      TXBUF0 = ConvOx;              // envio oxigênio dissolvido
            }
  }
//*****************************************************************************************************
}

void InitUART0(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P3SEL |= 0x30;                        // P3.4,5 = USART0 TXD/RXD
  ME2 |= UTXE0 + URXE0;                     // Enabled USART0 TXD/RXD
  UCTL0 |= CHAR;                        // 8-bit character
  UTCTL0 |= SSEL0;                      // UCLK = ACLK
  UBR00 = 0x0D;                         // 32k/2400 - 13.65
  UBR10 = 0x00;
  UMCTL0 = 0x6B;                        // Modulation
  UCTL0 &= ~SWRST;                          // Initalize USART state machine
  IE2 |= URXIE0;                        // Enabled USART0 RX interrupt
}

// UART0 RX ISR will for exit from LPM3 in Mainloop
#pragma vector=USART0RX_VECTOR
__interrupt void usart0_rx (void)
{
  _BIC_SR_IRQ(LPM3_bits);                   // Clear LPM3 bits from 0(SR)
}
```

```
//Função que gera tempo de 9+a*12 ciclos de clock
void Delay (unsigned int a)
{
  unsigned int k;
  for (k=0 ; k != a; ++k);


// Comunicaçao I2C
//*************************************************************************************************
//Inicializa I2C
void InitI2C(void)
{
  P3DIR |= 0x07;                        // P3.2 = SDA I2C/ P3.3 SCL I2C/ P3.4 USART0 TXD/ P3.5 USART0 RXD
  SendStop();
  Delay(150);
}

//Sinal de START I2C
void SendStart(void)
{
  SDA_OFF;
  Delay(10);
  SCL_OFF;
}

//Sinal de STOP I2C
void SendStop(void)
{
  SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SDA_ON;
}

//funcao responsavel pelo envio de um byte, bit a bit
void SendByteAux(unsigned char dado)
{
  Delay(5);
  if(BIT7&dado) //MSB
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
  if(BIT6&dado) //BIT 6
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
  if(BIT5&dado) //BIT 5
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
```

```
  if(BIT4&dado) //BIT 4
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
  if(BIT3&dado) //BIT 3
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
  if(BIT2&dado) //BIT 2
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
  if(BIT1&dado) //BIT 1
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
  if(BIT0&dado) //BIT 0
    SDA_ON;
  else
    SDA_OFF;
  Delay(5);
  SCL_ON;
  Delay(10);
  SCL_OFF;
  Delay(5);
}

//funcao responsavel pelo recebimento de um byte, bit a bit
void Read*Byte*Aux(unsigned char *dado)
{
  int i;
  //SDA_ON;
  P3DIR &= ~0x04;         //Pino do SDA como entrada
  for(i=7;i>=0;i--)
  {
    Delay(10);
    SCL_ON;
    Delay(5);
    if(SDA)                       //Se 1, seta bit
      bitset(*dado,i);
    else                //Se 0, clear bit
      bitclr(*dado,i);
    Delay(5);
    SCL_OFF;
  }
  Delay(10);
  P3DIR |= 0x04;        //volta do pino do SDA como saida
  SDA_OFF;
```

```
}

//funcao de reconhecimento da comunicação, Acknowlegde
unsigned char Ack(void)
{
  int contador=0;       //contador responsavel por esperar ate o reconhecimento, se cheio, nao houve Ack
  SDA_ON;
  bitclr(P3DIR,2);      //pino do SDA como entrada
  Delay(5);
  SCL_ON;
  while(SDA && (contador<20))
    contador++;
  SCL_OFF;
  bitset(P3DIR,2);      //pino do SDA como saida
  if(contador>19)
    return 0;
  return 1;
}

//funcao responsavel por enviar um bloco de informaçoes
unsigned char Send*Byte*(unsigned char valor, unsigned char address)
{
  Delay(10);
  SendStart();                  //Sinal de START
  //Delay(10);
  Send*Byte*Aux(address); //Envio endereço
  if(!Ack())                    //Sinal de reconhecimento
    return 0;                   //Retorna 0 se não há reconhecimento
  Delay(10);
  Send*Byte*Aux(valor);         //Envia os dados da variavel valor
  if(!Ack())
    return 0;
  Delay(10);
  SendStop();                   //Sinal de STOP da comunicação
  Delay(250);
  return 1;
}

//funcao responsavel por receber um bloco de informaçoes
void Read*Byte*(int *valor, unsigned char address)
{
  unsigned char MemChar1=0; //, MemChar2=0;
  Delay(10);
  SendStart();                  //Sinal de START
  Delay(10);
  Send*Byte*Aux(address);  //Endereço do SLAVE para solicitação dos dados
  if(!Ack());                   //Sinal de reconhecimento
  else
  {
            Read*Byte*Aux(&MemChar1);                   //Leitura dos dados do SLAVE
            //Delay(10);
            //Read*Byte*Aux(&MemChar2);
            Delay(10);
            //SDA_ON;
            SendStop();                         //Sinal de STOP da comunicação
            *valor= MemChar1;
  }
//*************************************************************************************************
}
```

Apêndice C – Código Fonte Programa Monitor

Unidade Principal

```
unit Monitor;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, XPMan, Menus, ExtCtrls, StdCtrls, ComCtrls, ToolWin, Buttons,
  JvExControls, JvSimIndicator, JvAutoComplete, JvComponentBase,
  JvSystemPopup, JvgDigits, ActnMan, ActnColorMaps, CPort, CPortCtl;

type
  TForm1 = class(TForm)
    XPManifest1: TXPManifest;
    MainMenu1: TMainMenu;
    Arquivo1: TMenuItem;
    SalvarCalibrao2: TMenuItem;
    Fechar1: TMenuItem;
    Fechar2: TMenuItem;
    Configurao1: TMenuItem;
    SelecionarPorta1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Panel1: TPanel;
    StatusBar1: TStatusBar;
    Timer1: TTimer;
    GroupBox1: TGroupBox;
    Button1: TButton;
    ComboBox1: TComboBox;
    AbrirCalibrao1: TMenuItem;
    Panel2: TPanel;
    AbrirCalibrao2: TMenuItem;
    Label4: TLabel;
    JvLookupAutoComplete1: TJvLookupAutoComplete;
    JvgDigits1: TJvgDigits;
    Label5: TLabel;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    ComPort1: TComPort;
    ComDataPacket1: TComDataPacket;
    ComLed1: TComLed;
    JvgDigits2: TJvgDigits;
    Label6: TLabel;
    Timer2: TTimer;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Fechar2Click(Sender: TObject);
    procedure Sobre1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure SelecionarPorta1Click(Sender: TObject);
    procedure AbrirCalibrao1Click(Sender: TObject);
    procedure SalvarCalibrao2Click(Sender: TObject);
    procedure AbrirCalibrao2Click(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure Timer2Timer(Sender: TObject);
    procedure ComPort1RxChar(Sender: TObject; Count: Integer);
    procedure ComboBox1Select(Sender: TObject);


  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  {Variaveis de calibração}
  pH1:integer;
  pH2:integer;
  Vp1:integer;
  Vp2:integer;
  Ox1:integer;
  Ox2:integer;
  Vo1:integer;
  Vo2:integer;
  flag_serial:integer;

implementation

uses Sobre, Calph, CalOx, Sair, Salvar;

{$R *.dfm}

procedure TForm1.Fechar2Click(Sender: TObject);
begin
  Form5.ShowModal;
end;

{Procedimento para abrir Formulário Sobre}
procedure TForm1.Sobre1Click(Sender: TObject);
begin
  Application.CreateForm(TForm2, Form2);
  Form2.ShowModal;
end;

{Data e hora na barra de status}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    StatusBar1.Panels[2].Text:= 'Hora: ' +TimeToStr(Time)+ '  Data: ' + DateToStr(Date);
    if (Button1.Caption= 'Conectar')  then
        StatusBar1.Panels[1].Text:= 'Desconectado'
    else
        StatusBar1.Panels[1].Text:=  'Conectado';

end;

{mudar nome botão quando clicar nele}
procedure TForm1.Button1Click(Sender: TObject);
begin

    if (Button1.Caption= 'Conectar')  then
      begin
        flag_serial:= 1;          //flag recepção
        flag_cal:=0;              //não entrar em calibração
        ComPort1.Connected := True;
        ComPort1.Open;
        Timer2.Enabled := True;
        Button1.Caption:='Desconectar';
        Button1.Hint:= 'Encerrar comunicação serial com a Sonda.'
      end
    else
      begin
        Timer2.Enabled := False;
        ComPort1.Connected := False;
```

```
        ComPort1.Close;
        Button1.Caption:='Conectar';
        Button1.Hint:= 'Iniciar comunicação serial com a Sonda.'
      end;
end;


procedure TForm1.SelecionarPorta1Click(Sender: TObject);
begin
   Form3.ShowModal;
end;

{formulário calibração oxigênio dissolvido}
procedure TForm1.AbrirCalibrao1Click(Sender: TObject);
begin
    Form4.ShowModal;
end;

{Salvar arquivo de calibração}
procedure TForm1.SalvarCalibrao2Click(Sender: TObject);

var
  arq: TStringlist;
begin
  if SaveDialog1.Execute then
  begin
    arq := TStringlist.Create;
    try
      arq.values['pH1'] := FloatToStr(pH1);
      arq.values['Vp1'] := FloatToStr(Vp1);
      arq.values['pH2'] := FloatToStr(pH2);
      arq.values['Vp2'] := FloatToStr(Vp2);
      arq.values['Ox1'] := FloatToStr(Ox1);
      arq.values['Vo1'] := FloatToStr(Vo1);
      arq.values['Ox2'] := FloatToStr(Ox2);
      arq.values['Vo2'] := FloatToStr(Vo2);
      arq.SaveToFile(SaveDialog1.FileName);
    finally
      arq.free;
    end;
  end;
end;

{Abrir arquivo de calibração}
procedure TForm1.AbrirCalibrao2Click(Sender: TObject);
var
  arq: TStringlist;
begin
  if OpenDialog1.Execute then
  begin
    arq := TStringlist.Create;
    try
      arq.LoadFromFile(OpenDialog1.FileName);
      pH1:= StrToint(arq.Values['pH1']);
      Vp1:= StrToint(arq.Values['Vp1']);
      pH2:= StrToint(arq.Values['pH2']);
      Vp2:= StrToint(arq.Values['Vp2']);
      Ox1:= StrToint(arq.Values['Ox1']);
      Vo1:= StrToint(arq.Values['Vo1']);
      Ox2:= StrToint(arq.Values['Ox2']);
      Vo2:= StrToint(arq.Values['Vo2']);
    finally
      arq.free;
    end;
  end;
end;

//Quando sair
```

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
   Form5.ShowModal;
end;

//Timer de 1s
procedure TForm1.Timer2Timer(Sender: TObject);

begin
   Edit2.Text:= BoolToStr(ComPort1.Connected);
   if ComPort1.Connected then
     begin
      Comport1.WriteStr('V');    //envia caracter ASCII V
      if (flag_serial=3) then flag_serial :=1;
     end;
end;

//Recepção pela serial
procedure TForm1.ComPort1RxChar(Sender: TObject; Count: Integer);
var
  Str: String;
  valor: integer;
  i: integer;
begin
   ComPort1.ReadStr(Str, Count);     //leitura do buffer
   valor:=StrToInt(Str);
   //valor:= valor and (0x00FF);
   Str:= FloatToStr(valor*3.3/255);           //adequação para tensão
   valor:= StrToInt(Str);

   //recepçao pH ************************************************************
   if ((flag_serial=1) and (flag_cal=0)) then
   begin
      //com calibração de 2 pontos
      if ((Vp1<>0) and (Vp2<>0)) then JvgDigits1.Value:=((valor/10)*(pH2-pH1)-Vp1*(pH2-pH1)+(Vp2-Vp1)*pH1)/(Vp2-
Vp1);
      //com calibração de 1 ponto
      if ((Vp1<>0) and (Vp2=0)) then JvgDigits1.Value:= ((valor/10)-Vp1+pH1)/(1/7);
      //sem calibração
      if ((Vp1=0) and (Vp2=0)) then JvgDigits1.Value:= (valor/10-2.15+7)/(1/7)
   end;

   //recepçao pH na calibração ***************************************************
   if ((flag_serial=1) and (flag_cal=1)) then
      begin
         if(Form3.RadioButton1.Checked = True) then
            begin
            Vp1:= valor;
            pH1:= StrToInt(Form3.Edit1.Text);
            for i:=0 to 100 do
              begin
              Form3.ProgressBar1.Position:= Form3.ProgressBar1.Position + 1;
              end;
            end;
         if(Form3.RadioButton2.Checked= true) then
            begin
            Vp2:= valor;
            pH2:= StrToInt(Form3.Edit1.Text);
            for i:=0 to 100 do
              begin
              Form3.ProgressBar1.Position:= Form3.ProgressBar1.Position + 1;
              end;
            end;
         Timer2.Enabled := False;
         ComPort1.Close;
       end;
```

```
   //recepção ox ************************************************************
   if ((flag_serial=2) and (flag_cal=0)) then JvgDigits2.Value:=((valor/10)*(Ox2-Ox1)-Vo1*(Ox2-Ox1)+(Vo2-
Vo1)*Ox1)/(Vo2-Vo1);

   //recepção ox calibração *****************************************************

   //segunda passagem
   if ((flag_serial=2) and (flag_cal=2)) then
   begin
        if(Form4.RadioButton1.Checked) = True then
           begin
           Vo1:= valor;
           Ox1:= 0;
           for i:=0 to 100 do
             begin
             Form3.ProgressBar1.Position:= Form3.ProgressBar1.Position + 1;
             end;
           end;
        if(Form3.RadioButton2.Checked= true) then
           begin
           Vo2:= valor;
           Ox2:= 100;
           for i:=0 to 100 do
             begin
             Form3.ProgressBar1.Position:= Form3.ProgressBar1.Position + 1;
             end;
           end;
        Timer2.Enabled := False;
        ComPort1.Close;
      end;
   //primeira passagem
   if ((flag_serial=2) and (flag_cal=1)) then flag_cal:=2;

   //incremento p/ leitura do oxigênio dissolvido
   if (flag_cal<>1)then flag_serial:= flag_serial + 1;
end;

//Seleção da porta serial
procedure TForm1.ComboBox1Select(Sender: TObject);
begin
   if ComboBox1.Text='COM1' then ComPort1.Port:= 'COM1';
   if ComboBox1.Text='COM2' then ComPort1.Port:= 'COM2';
   if ComboBox1.Text='COM3' then ComPort1.Port:= 'COM3';
   if ComboBox1.Text='COM4' then ComPort1.Port:= 'COM4';
end;

end.
```

Unidade de Calibração do pH

```
unit Calph;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, ExtCtrls,Monitor;

type
  TForm3 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    ProgressBar1: TProgressBar;
    Button2: TButton;
```

```
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    Button3: TButton;
    Edit1: TEdit;
    Label3: TLabel;
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form3: TForm3;
  flag_cal:integer;

implementation

{$R *.dfm}

procedure TForm3.Button2Click(Sender: TObject);
begin
    flag_cal:= 0;
    close;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
    Form3.ProgressBar1.Position:= 0;
    flag_serial:= 1;          //flag recepção
    flag_cal:= 1;
    Form1.ComPort1.Open;
    Form1.ComPort1.Connected := True;
    Form1.Timer2.Enabled := True;
end;

end.
```

Unidade de Calibração do Oxigênio Dissolvido

```
unit CalOx;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, Monitor;

type
  TForm4 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    Button1: TButton;
    Button3: TButton;
    ProgressBar1: TProgressBar;
    procedure Button3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
```

```
    { Public declarations }
  end;

var
  Form4: TForm4;

implementation

uses Calph;

{$R *.dfm}

procedure TForm4.Button3Click(Sender: TObject);
begin
  flag_cal:=0;
  close;
end;

procedure TForm4.Button1Click(Sender: TObject);
begin
    Form3.ProgressBar1.Position:= 0;
    flag_serial:= 2;          //flag recepção
    flag_cal:= 1;
    Form1.ComPort1.Open;
    Form1.ComPort1.Connected := True;
    Form1.Timer2.Enabled := True;
end;

end.
```