Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

**Desenvolvimento de um Sistema Embarcado, Eletrônico e Computacional para controle remoto de equipamentos de medição**

Gustavo Souto Nóbrega

Orientador: Manoel Eusebio de Lima

Trabalho de Graduação

Recife, Janeiro de 2013

**Resumo**

Este projeto tem o objetivo de detalhar o desenvolvimento de um sistema embarcado, eletrônico e computacional utilizado para armazenamento de informações e controle remoto de equipamentos destinados à medição de dispositivos eletrônicos. O sistema é baseado em um microcontrolador de baixo consumo de potência, o MSP430, controlado por uma plataforma computacional e conectado via interface serial em até dois equipamentos de medição precisos dedicados, do modelo Keithley model 2400[7]. O software embarcado foi escrito em C, e a plataforma foi desenvolvida em Visual C++.

**Lista de Figuras**

Figura 1: Diagrama de blocos do sistema................................................................ 7

Figura 2: Diagrama de bloco funcional do MSP430F2274.................................... 10

Figura 3: Esquemático dos pinos 0 à 3 da porta 1 do MSP430F2274.................... 11

Figura 4: Ciclo de escrita da FIFO.......................................................................... 14

Figura 5: Ciclo de leitura da FIFO.......................................................................... 15

Figura 6: Diagrama de blocos do latch 3-state........................................................ 16

Figura 7: Circuito do conversor MAX232.............................................................. 17

Figura 8: Parte frontal do Keithley model 2400...................................................... 18

Figura 9: Parte traseira do Keithley model 2400..................................................... 19

Figura 10: Diagrama de blocos do circuito.............................................................. 21

Figura 11: Configuração dos pinos das portas do MSP430..................................... 22

Figura 12: Máquina de estado do microcontrolador................................................ 23

Figura 13: Diagrama da máquina de estados escreve serial.................................... 28

Figura 14: Diagrama da máquina de estados escreve serial.................................... 29

Figura 15: Exemplo rotina de escrita e leitura da FTDI pelo computador............. 30

Figura 16: Exemplo rotina de requisição de leitura de um equipamento................ 31

Figura 17: Interface principal desenvolvida............................................................ 32

Figura 18: Interface da aplicação de leitura com variação de configuração............ 33

Figura 19: Interface da aplicação de leitura para transistores.................................. 33

**Lista de Tabelas**

Tabela 1: Funções da FTD2XX.DLL utilizadas no projeto.................................... 15

Tabela 2: Tabela de estados do latch 3-state........................................................... 16

Tabela 3: Funções dos botões da parte frontal do Keithley model 2400................ 18

Tabela 4: Exemplo de mensagens entendidas pelo Keithley model 2400.............. 20

**Sumário**

**1. Introdução......................................................................... 6**

1.1 Contexto e Motivação..................................................................................... 6

1.2 Objetivo.......................................................................................................... 7

1.3 Estrutura do Texto.......................................................................................... 8

**2. Conceitos Básicos............................................................. 9**

2.1 Microcontrolador MSP430............................................................................. 9

2.1.1 MSP430F2274.......................................................................................... 9

2.1.2 Interrupção................................................................................................ 12

2.1.3 Timer......................................................................................................... 12

2.1.4 Comunicação Serial.................................................................................. 12

2.2 FTDI............................................................................................................... 14

2.2.1 FT245R..................................................................................................... 14

2.2.2 FTD2XX................................................................................................... 15

2.3 Latch 3-State................................................................................................... 16

2.4 MAX232......................................................................................................... 17

2.5 Equipamentos Keithley model 2400............................................................... 17

2.5.1 Protocolo de Comunicação....................................................................... 19

**3. Comportamento do Circuito Eletrônico........................ 21**

**4. Programação..................................................................... 23**

4.1 IDE................................................................................................................. 23

4.2 Programação Embarcada................................................................................ 24

4.2.1 Inicializações............................................................................................ 24

4.2.2 Interrupções e Timer................................................................................. 25

4.2.3 Máquina de estados principal................................................................... 26

4.2.4 Máquina de estados envia serial............................................................... 27

4.2.5 Máquina de estados recebe serial............................................................. 28

4.3 Programação alto nível................................................................................... 29

4.3.1 Aplicações desenvolvidas........................................................................ 32

**5. Conclusão.......................................................................... 34**

**6. Referencias Bibliográficas............................................... 35**

**1. Introdução**

**1.1 Contexto e Motivação**

No estudo das propriedades de dispositivos eletrônicos, muitas vezes, é necessário o uso de equipamentos de alta precisão para realizar e gerar condições necessárias para medições de parâmetros, como curvas de comportamento eletrônico em transistores, diodos, etc. Um exemplo desse tipo de equipamento é o analisador de parâmetros elétricos, o Keithley model 2400, utilizado na geração de medidas elétricas, caracterizando propriedades eletrônicas, como corrente e voltagem em dispositivos semicondutores. Esse tipo de equipamento, alvo deste trabalho de graduação, permite a caracterização de vários dispositivos de forma manual, onde o operador calibra, gera sinais e os monitora localmente, ou de forma remota e automática, utilizando sistemas microprocessados, que através de protocolos especiais, permitem monitoração, configurações, armazenamento e tratamento de medições.

Através de uma interface RS-232, e um protocolo de comunicação fornecido pelo fabricante, é possível configurar o equipamento de medição e requisitar informações específicas remotamente. Esse monitoramento é em geral controlado por um computador que trata o protocolo, possibilita a aquisição, o armazenamento e o tratamento adequado dos dados. Essa metodologia viabiliza controle centralizado para múltiplos equipamentos e rápidas solicitações e variações de configurações em série, operação necessária para realizar estudos específicos em componentes e sistemas, como curvas características de dispositivos semicondutores, tais como: curvas de transistores MOS, diodos, ou ainda amplificadores, conversores, etc.

Quando realizadas manualmente, atividades como a descrita acima se tornam árduas para o usuário. Além de tomar nota sem erros a cada medição, caberia a esse, também, a tarefa de configurar cada máquina a cada nova medição, o que toma tempo e cuidados significativos. Realizado por um computador, a árdua tarefa de reduz simplesmente ao trabalho de se ajustar as configurações iniciais e pressionar um botão que abstrairá todas as etapas da operação, obtendo-se rapidamente os resultados.

**1.2 Objetivo**

O sistema abordado nesse projeto foi desenvolvido com a finalidade de controlar e obter informações remotamente de até dois equipamentos medidores Keithley modelo 2400 utilizando apenas um computador dotado de uma interface visual e um módulo eletrônico externo (figura 1).

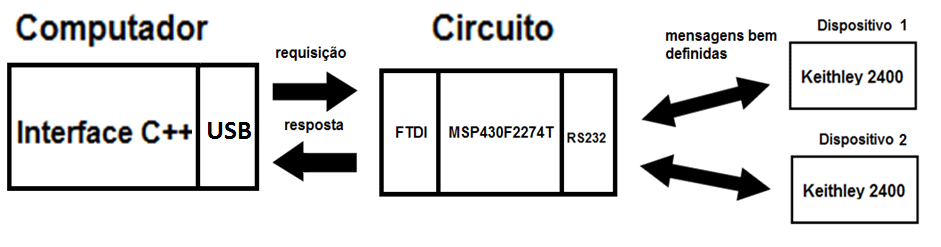


Figura 1: Diagrama de blocos do sistema

O projeto também tem como meta facilitar operações específicas, mostrando o desenvolvimento do controle dos equipamentos pelo computador de maneira singular ou em série, para uma ou ambas as máquinas, podendo-se variar o tipo de solicitação.

**1.3 Estrutura do Texto**

Quanto à disposição desse trabalho, foi adotado que no capítulo 1 teremos a Introdução onde falaremos sobre o contexto e a motivação em realizar esse trabalho, seu objetivo e sua organização. O capítulo 2 expõe os conceitos essenciais para o entendimento e desenvolvimento do projeto. Os capítulos 3 e 4 definem a metodologia do projeto, sendo o primeiro a descrição do módulo eletrônico e o segundo à programações: embarcada e em alto nível. O capítulo 5 é dedicado às conclusões do trabalho, e o 6 às referências bibliográficas.

**2. Conceitos Básicos**

**2.1 Microcontrolador MSP430**

Um Microcontrolador genérico trata-se de um pequeno computador dentro de um circuito integrado contendo um ou mais processadores, memória e dispositivos de entrada e saída programáveis. É um dispositivo que pode ser programado e embarcado no interior de algum outro dispositivo para que possa controlar certas funcionalidades [1].

Particularmente o microcontrolador MSP430 faz parte de uma família de microcontroladores fabricados pela Texas Instruments de 16 bits, voltados para aplicações que requeiram um baixo consumo de energia.

A CPU dos MSP430 possui um conjunto de apenas 51 instruções (27 físicas e 24 emuladas) e um total de 16 registradores de 16 bits. O microcontrolador é nutrido de memória RAM e flash, possui um oscilador interno de baixíssimo consumo, *watchdog timer*, que reinicia o processador se não for periodicamente desarmado, *timers*, *USARTs*, comparador analógico, amplificador operacional, dentre outros recursos, dependendo da família [2].

Esse componente possui ambientes de desenvolvimento que permitem ao usuário programá-lo a partir de ambientes de programação Open-Source, em linguagens de alto nível, tais como C/C++, ou ainda em Assembly.

A programação desses componentes é simples, necessitando apenas da funcionalidade de alguns sinais de controle em pinos específicos e interfaces padrão USS. Para gravação de seu firmware, sinais especiais devem ser devidamente acionados, tais como:

* DVCC
* DVSS
* TEST
* RESET

**2.1.1 Microcontrolador MSP430F2274**

Neste trabalho, levando-se em conta o baixo consumo de energia, pois todo o sistema será alimentado por uma interface USB do computador de monitoração, e o número de pinos de entradas e saídas necessárias para o projeto, foi escolhido um modelo, dentre aqueles da família MSP430, que possuísse quatro portas de oito pinos cada, o MSP430F2274. A figura 2 mostra os módulos internos do microcontrolador.

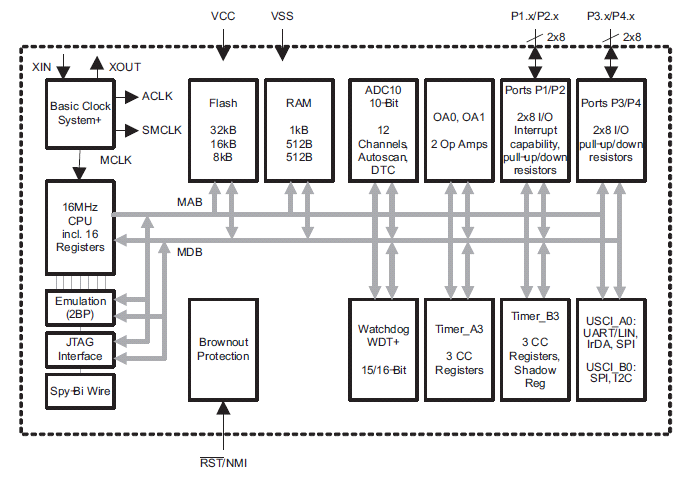


Figura 2: Diagrama de bloco funcional do MSP430F2274.

Dessa forma, as características mais relevantes desse modelo de microcontrolador são: sua baixíssima potência operacional, 594μW, tempo de inicialização menor que 1μs, ciclo de instrução de 62,5ns, clock interno de até 16MHz, 2 Timers de 16 bits, uma interface UART, memória Flash de 32KB para o código, mais 256B livres, 1Kb de memória RAM, programação serial sem necessidade de alimentação externa, emulação no Chip e 4 portas de entrada e saída de 8 pinos cada [3].

Esse modelo oferece diversas funcionalidades padrões de acordo com o pino e da porta. Suas portas I/O podem ser configuradas na programação através de valores específicos em registradores e multiplexadores internos, mostrados na figura 3. Para a aplicação em questão só foi necessário configurar os pinos para funcionar nos modos entrada, saída e interrupção, e se os eventos ocorrem na subida ou descida do nível lógico.

Para programar o pino como modo I/O, configura-se o valor do bit correspondente do registrador de seleção, P1SEL, para 0, o que fará o multiplexador M2 e o módulo U1 isolarem as funcionalidades secundárias do pino, Module X OUT e Module X IN. Além disso, essa configuração permite que o latch F2, responsável por manter a flag de interrupção, disponibilize em sua saída o valor da entrada.

Para configurar o pino como entrada, atribui-se ao bit correspondente do registrador de direção, P1DIR, o valor 0, o que fará os tri-states T1 e T2 isolarem o registrador de saída, P1OUT, e o schmitt trigger ST1 liberar o caminho entre o pino e o registrador de entrada, P1IN. Embora não tenha sido utilizado no projeto, pode-se definir o valor da entrada quando não houver nada conectado ao pino. Isso é feito ativando o resistor pull-up R1, atribuindo ao registrador de ativação do resistor, P1REN, o valor 1, o que fechará a chave C1, sendo definido como valor padrão de entrada DVSS ou DVCC, de acordo com a saída do multiplexador M3 controlado pelo registrador de saída, PIOUT. Além disso, essa configuração evita curto-circuito isolando a entrada de P1OUT utilizando o tri-state T2.

Para configurar o pino como saída, além de atribuir ao bit correspondente de P1REN o valor 0, atribui-se ao bit correspondente do registrador de direção, P1DIR, o valor, 1, o que fará os tri-states T1 e T2 liberarem o caminho entre o registrador de saída, P1OUT, e o pino. Além disso, essa configuração faz o schmitt trigger ST1 ir pro estado alta impedância, isolando o pino do registrador de entrada, P1IN, e do controlador de nível de interrupção.

Para configurar o pino para se fazer uma chamada em uma rotina de interrupção quando ocorrer variação no nível lógico, seja na subida ou descida, atribui-se ao bit correspondente do registrador de ativação de interrupção, P1IE, o valor 1. Para selecionar interrupção na subida, atribui-se 0 ao registrador de seleção de borda, P1IES, ou 1, para selecionar interrupção na descida. Quando houver uma interrupção, a rotina será chamada quando o microcontrolador verificar a flag P1IRQ ativada. O latch F2 irá manter essa flag até o momento em que o registrador de interrupção, P1IFG, for zerado, possibilitando novas chamadas à rotina de interrupção.

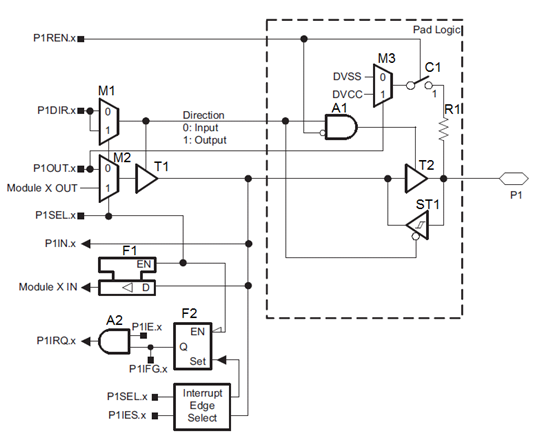


Figura 3: Esquemático dos pinos 0 à 3 da porta 1 do MSP430F2274.

**2.1.2 Interrupção**

Interrupções são atributos essenciais para qualquer microcontrolador. Elas funcionam forçando a execução de determinada rotina quase no momento em que um evento previamente configurado ocorre, sem a necessidade de gastar tempo e poder de processamento verificando se tal evento ocorreu, otimizando o programa.

Embora existam diversas situações em que é necessário o uso de interrupções, nesse projeto foram utilizadas, somente, interrupções de estouro do registrador do Timer e interrupções na variação de nível lógico na entrada de certos pinos.

**2.1.3 Timer**

Em um microcontrolador, a função do Timer é impedir o travamento do processador enquanto se encontra no estado de aguardando para realizar uma determinada função.

Os timers são registradores que servem como contadores em que, a cada pulso de clock, um incremento é realizado nesse registrador. Após o timer ser incrementando até um valor configurado previamente, tendo valor máximo, no caso do microcontrolador em questão, 216 = 65.536, é acionada a função especial programada no firmware na interrupção do timer.

Embora os timers possam ser configurados para terem o clock de incremento através de um cristal externo, no projeto foi utilizado o clock auxiliar interno, o qual pode ter a frequência configurada via programação.

**2.1.4 Comunicação Serial**

A transmissão serial trata-se de um meio de transferência de dados em série. No emissor, ocorre a conversão dos dados em bits, e em seguida os mesmos são transmitidos sequencialmente, passando pelo canal apenas um bit por vez. Os bits então são arranjados no receptor para recompor os dados originais. A vantagem desse tipo de transmissão com relação à transmissão simultânea é o menor custo, tendo desvantagem de ser mais lenta.

A taxa de transmissão serial é medida em baudrate, taxa de envio de bits por segundo no canal. No caso de uma taxa de baudrate de 9600bits/s, quase sempre definido como taxa padrão, o período de cada bit é 1 / 9600 = 104 ms.

A transmissão serial pode ser realizada de duas formas: síncrona ou assíncrona. Na transmissão síncrona, na recepção de um pulso de relógio, o receptor lê o canal de dado e armazena o valor do bit encontrado naquele momento. O canal de dados não é lido novamente até que o próximo pulso de relógio chegue. Como o transmissor é responsável pelos pulsos de dados e de temporização, o receptor irá ler o canal de dados apenas quando comandado pelo transmissor e, portanto, a sincronização é garantida.

Já na transmissão assíncrona, o transmissor e o receptor devem ter tido, antecipadamente, configurados tanto o oscilador, os quais precisam ser iguais, ou muito próximos, quanto o número de bits a serem transmitidos por pacote, para que a comunicação se estabeleça de forma eficiente. O transmissor deve iniciar a transmissão dos dados enviando, inicialmente, um bit lógico, conhecido como start bit, para sinalizar ao receptor que uma transmissão foi iniciada. O nível lógico desse bit deve ser diferente do nível lógico pré-estabelecido como repouso, momento em que não está ocorrendo transferências. Por exemplo, no protocolo RS232, o estado de repouso é o nível lógico ‘1’, enquanto o start bit representa um nível lógico ‘0’. O start bit inicializa um temporizador interno no receptor avisando que a transmissão começou e que serão necessários pulsos de relógio. Seguido do start bit, os bits que compõem o pacote da mensagem devem ser enviados na taxa de transmissão especificada. O pacote pode ser concluído nessa etapa, ou pode ser incluído, também, um bit de paridade, com o objetivo de impor uma detecção de falha, e um ou mais bits de stop, indicando o término do envio do pacote.

O bit de paridade é adicionado ao pacote de dados com o propósito de detecção de erro. Na convenção de paridade-par, o valor do bit de paridade é escolhido de tal forma que o número total de dígitos ‘1’ dos dados adicionado ao bit de paridade do pacote seja sempre um número par. Na recepção do pacote, a paridade do dado precisa ser recalculada e comparada com o bit de paridade recebido com os dados. Se qualquer bit mudou de estado durante a transferência, a paridade não irá coincidir, e um erro será detectado. Se um número par de bits for trocado, a paridade coincidirá e o dado com erro será validado. Contudo, uma análise estatística dos erros de comunicação de dados tem mostrado que um erro com bit simples é muito mais provável que erros em múltiplos bits na presença de ruído randômico. Portanto, a paridade é um método confiável de detecção de erro [4].

De acordo com o protocolo RS-232 utilizado pelo fabricante do Keithley 2400, neste trabalho foi implementado um protocolo serial assíncrono com baudrate de 9600bits/s, dois bits de stop e sem bit de paridade.

**2.2 FTDI**

Trata-se de um conversor USB-FIFO (paralelo) composto por um módulo em circuito integrado, o FT245R, e um driver especial, denominado FTD2XX.lib.

**2.2.1 FT245R**

Através do módulo FT245R da FTDI, o computador de monitoração pode enviar e receber dados provenientes do circuito de aquisição através de uma interface USB e, dessa forma, se comunicar com as portas do microcontrolador. Esse dispositivo funciona como adaptador USB para interface paralela FIFO, através dos pinos D0 à D7.

Na FTDI, para definir se o dispositivo estará operando no modo conversor serial-paralelo ou paralelo-serial, os pinos de entrada RD# e WR devem ser configurados no momento da conversão. Tanto RD# quanto WR devem ficar sempre em nível lógico alto, exceto no momento da conversão de dados, quando na queda do nível lógico de um desses bits, a FTDI realiza a conversão. Quando RD# vai de alto para baixo nível, a conversão paralelo-serial é realizada, momento em que o computador lê os dados provenientes do microcontrolador. Já quando WR vai de alto para baixo nível, ocorre a conversão serial-paralela, onde o computar escreve os dados, os quais são encaminhados para o microcontrolador. Ambos os pinos são controlados pelo microcontrolador.

O pino TXE# quando em estado alto, não permite a conversão serial-paralela. Quando em baixo estado, permite a conversão do dado serial do USB para os pinos de saída, D0 à D7, na descida de nível de WR. A figura 4 apresenta o ciclo da conversão serial-paralelo, ou escrita no microcontrolador pelo computador de monitoração.

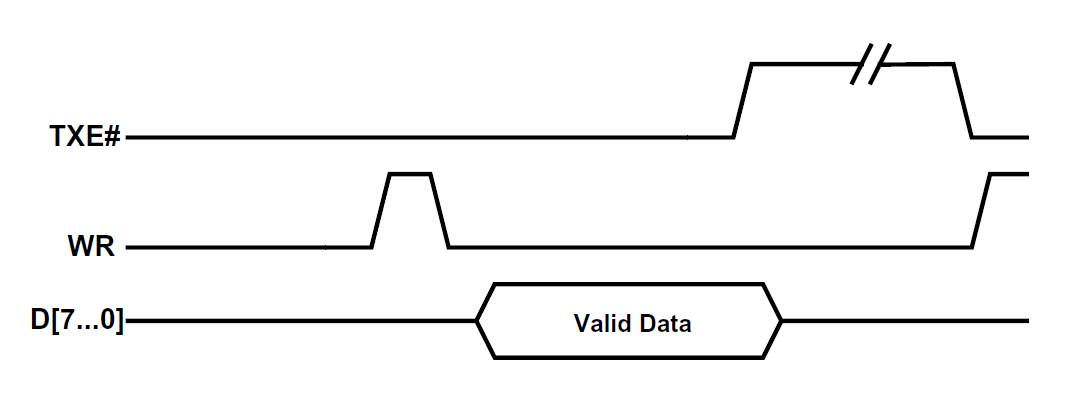


Figura 4: Ciclo de escrita da FIFO.

O pino RXF# funciona de maneira semelhante, não permitindo a conversão paralelo-serial quando em estado alto. Quando em estado baixo, permite a conversão do dado paralelo em seus pinos de D0 à D7 para serial via USB, na descida de nível de RD. A figura 5 apresenta o ciclo da conversão paralelo-serial, ou leitura da FTDI pelo computador de monitoração.

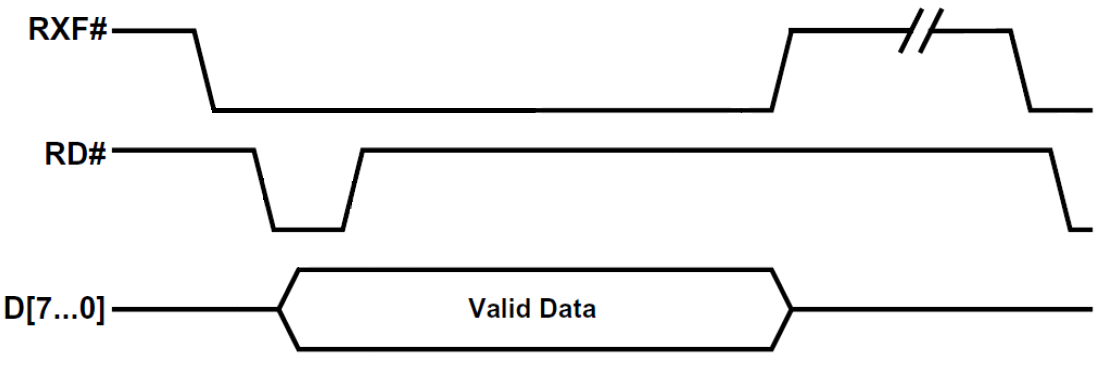


Figura 5: Ciclo de leitura da FIFO.

Os pinos VCC, 3V3OUT e GND aproveitam a alimentação do USB proveniente do computador para alimentar todo o circuito. Para a aplicação em questão o restante dos pinos é aterrado ou deixado em aberto.

**2.2.2 FTD2XX**

O driver *FTD2XX.lib* é uma interface específica dos dispositivos FTDI. Ele permite a comunicação direta do programa nativo em um computador com o módulo FT245R, emitindo e recebendo dados, bem como detectando se os procedimentos no módulo obtiveram sucesso ou não. É necessário, apenas, inclusão no projeto para realização das chamadas das funções desejadas (tabela 1). O driver requer a passagem de parâmetros corretamente, quando necessários.

Tabela 1: Funções da FTD2XX.DLL utilizadas no projeto.

|  |  |
| --- | --- |
| Função | Efeito |
| FT\_Write | Escreve dados do dispositivo |
| FT\_Read | Lê dados do dispositivo |
| FT\_Open | Fecha o dispositivo |
| FT\_Close | Abre o dispositivo |
| FT\_SetTimeouts | Configura o tempo máximo de espera pela leitura |
| FT\_HANDLE (Tipo) | Tipo de variável que armazena o endereço onde o dispositivo está alocado |
| FT\_Status (Tipo) | Tipo de variável que armazena se a função em questão obteve sucesso |

**2.3 Latch 3-State**

Como os pinos das portas do microcontrolador só podem ser programados ou como entrada ou como saída, e a interface com o computador requer ambos os modos, foi adicionado um controle extra para evitar um curto circuito quando há competição no barramento. Trata-se de um latch 3-state, tendo um pino do microcontrolador a função de controle sobre esse componente. Deixando-se sempre o pino LE em nível lógico alto, quando o pino está também em nível lógico alto, a saída que liga o latch com o FTDI vai para alta impedância, isolando aquela porta do circuito, permitindo apenas a ligação da saída da FTDI com a porta de entrada do microcontrolador, o que representa uma leitura do microcontrolador. Contudo, quando o sinal de cai, o latch permite a passagem dos dados da saída do microcontrolador para a porta da FTDI, o que representa uma tarefa de escrita do microcontrolador. Nesse momento, embora as portas de entrada e saída estejam em curto circuito, não ocorre falhas, pois a entrada de dados no nível do programa é ignorada. O comportamento das portas de entrada e saída podem ser visualizadas na figura 6. As configurações dos modos de operação são apresentadas na tabela 2.

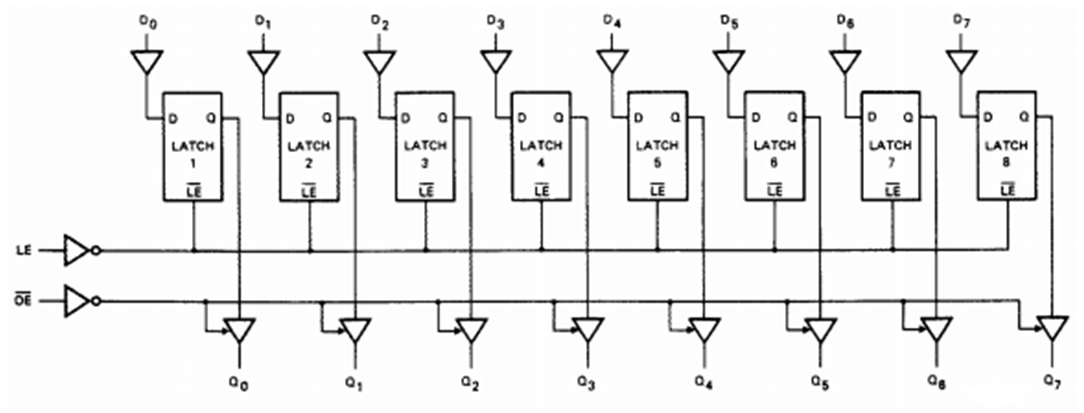


Figura 6: Diagrama de blocos do latch 3-state.

Tabela 2: Tabela de estados do latch 3-state.



**2.4 MAX232**

Embora o microcontrolador emita e receba dados seriais com 0 volt para representar baixo nível lógico e 3 volts para alto nível lógico, os equipamentos medidores em questão utilizam protocolo RS232, onde sinais de -25 à -3 volts representam nível lógico ‘1’, enquanto de +3 à+25 volts representam nível lógico ‘0’. Então um conversor MAX232 (figura 7) foi alocado como interface de protocolo físico entre os dois componentes.

A tensão mais alta é gerada por um circuito inversor interno, que utiliza quatro capacitores externos [6]. Com esse dispositivo, evita-se a necessidade de uma fonte de alimentação com voltagens negativas, como no caso de acopladores óticos conversores de nível RS-232/TTL. O MAX232 já oferece duas portas para TX e duas para RX, o que permitiu o uso de apenas um único dispositivo desse tipo para o projeto em questão.

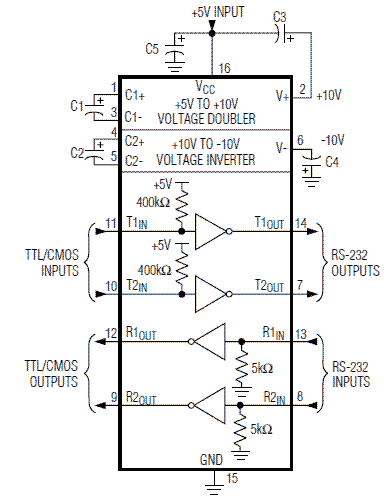


Figura 7: Circuito do conversor MAX232.

**2.5 Equipamentos Keithley model 2400**

O Keithley modelo 2400, figura 8, trata-se de um equipamento de alta precisão ideal no estudo de propriedades de componentes eletrônicos. Combina uma fonte DC altamente estável, precisa e pouco ruidosa com um multímetro de alta impedância, pouco ruidoso e altamente repetível. A mesma opera com uma precisão de 0,012% e resolução de 5½ dígitos, o que indica uma taxa de 520 leituras por segundo, podendo esse valor ser elevado para mais de 2000 leituras por segundo caso amplie a capacidade do seu buffer interno.

Seu alcance de tensão vai de 5µV à 210V, enquanto de corrente, de 50pA à 1.05A. Além disso, esse modelo pode detectar resistências de 100µΩ à 211MΩ e gera potência máxima de 22W[7].

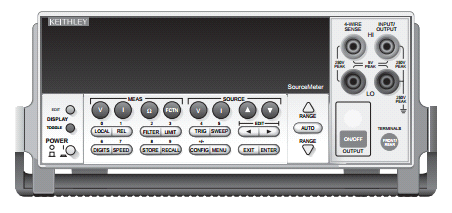


Figura 8: Parte frontal do Keithley model 2400.

A tabela 3 apresenta as funcionalidades do equipamento:

Tabela 3: Funções dos botões da parte frontal do Keithley model 2400.

|  |  |
| --- | --- |
| Tecla | Função |
| V (MEAS) | Medir tensão |
| I (MEAS) | Medir corrente |
| Ω (MEAS) | Medir resistência |
| FCTN (MEAS) | Executar funções matemáticas |
| V (SOURCE) | Configurar tensão |
| I (SOURCE) | Configurar corrente |
| /\ e \/ (SOURCE) | Aumentar/Diminuir valor |
| EDIT | Alterna entre fonte e comprimento |
| TOGGLE | Alterna a posição de amostra da medida e configuração, ou medidas de corrente e tensão |
| LOCAL | Cancela operações remotas |
| REL | Ativa/Desativa leituras da função presente |
| FILTER | Mostra filtros digitais e os ativa/desativa |
| LIMIT | Realiza testes dos limites configurados |
| TRIG | Força a leitura pelo painel frontal |
| SWEEP | Inicia varredura configurada |
| < e > | Move entre os valores dos parâmetros ou seleções dentro de funções e operações |
| DIGITS | Muda o número de dígitos da resolução |
| SPEED | Muda a velocidade de leitura |
| STORE | Configura o tamanho do buffer |
| RECALL | Mostra leituras salvas e a hora/data |
| CONFIG | Configura a função ou operação |
| MENU | Acessa e configura o Menu |
| EXIT | Cancela a operação |
| ENTER | Aceita a seleção |
| /\ e \/ (RANGE) | Move para o próximo intervalo do dígito |
| AUTO | Ativa/Desativa auto intervalo da medida |
| ON/OFF | Liga/Desliga o equipamento |
| FRONT/REAR | Seleciona conexões do painel frontal ou anterior |

A figura 9 apresenta os conectores através dos quais o equipamento se comunica com o mundo externo.

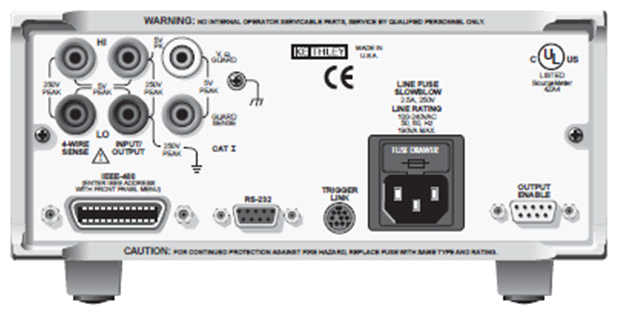


Figura 9: Parte traseira do Keithley model 2400.

Através da interface RS-232, o equipamento pode realizar solicitações de medições e configurações de todas as funções do painel frontal remotamente. Conectado ao MAX232, os dados podem fluir do microcontrolador para o dispositivo, e vice-versa.

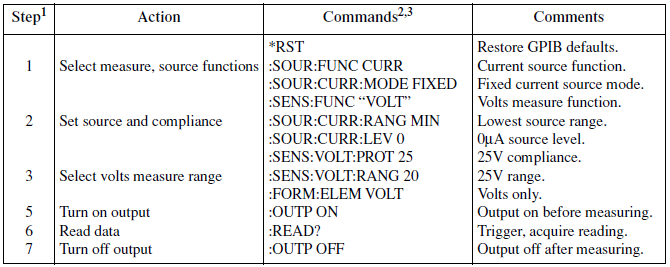
**2.5.1 Protocolo de comunicação**

Com a interface RS-232 do Keithley modelo 2400, o equipamento pode ser controlado remotamente. Ele pode receber dados que podem alterar suas configurações e valores, ou podem requisitar respostas de leituras. Para cada função dos botões da parte frontal da máquina, existe uma mensagem previamente definida pelo fabricante que, depois de recebida na interface serial, concatenada com um parâmetro, caso possua, e o caracter \n, indicando o fim da mensagem, executará a mesma funcionalidade. As mensagens foram elaboradas intuitivamente, formada por uma cadeia de bytes. Cada byte representa um caractere em código ASCII, tendo a mensagem uma aparência de frase, concatenada com o parâmetro numérico, quando necessário. Tal mensagem deve ser transmitida via serial e, por padrão, o baudrate é 9600bits/s com um bit de start e dois bits de stop, sem bit de paridade.

O maior tamanho de uma mensagem entendida pelo equipamento é 40 bytes, o que leva 40 \* 11 \* 0,104 ms = 45,76 ms para ser transmitida.

A tabela 4 mostra exemplos de mensagens utilizadas pelo equipamento.

Tabela 4: Exemplo de mensagens entendidas pelo Keithley model 2400. Mais comandos consulte o manual do fabricante, [7].



Quando a mensagem :READ?\n é recebida pelo equipamento, o mesmo devolve uma mensagem de resposta, obedecendo ao protocolo serial RS-232 com dois bits de stop. Essa resposta contém 13 bytes que correspondem à voltagem, 13 bytes relativos à corrente e outros 13 bytes para a representação de valores de resistência. Outros bytes de informação também são fornecidos, porém não são relevantes neste projeto.

No caso das medidas desejadas, o primeiro byte corresponde ao sinal da medida, o segundo a casa decimal mais significativa, o terceiro ao caractere ‘.’, o quarto ao nono as casas decimais depois da vírgula, o décimo ao caractere ‘E’, que representa expoente na base 10, o décimo primeiro ao sinal do expoente, e os dois últimos ao valor do expoente.

**3. Comportamento do Circuito Eletrônico**

O circuito do sistema funciona recebendo os dados do computador e em seguida repassando para o controlador. Após o processamento, ele encaminha os dados para o equipamento previamente selecionado. O mesmo também faz a transmissão inversa, viabilizando o tráfego da resposta do equipamento para o microcontrolador e do microcontrolador para o computador. O diagrama de blocos da figura 10 apresenta as conexões entre os módulos do projeto.

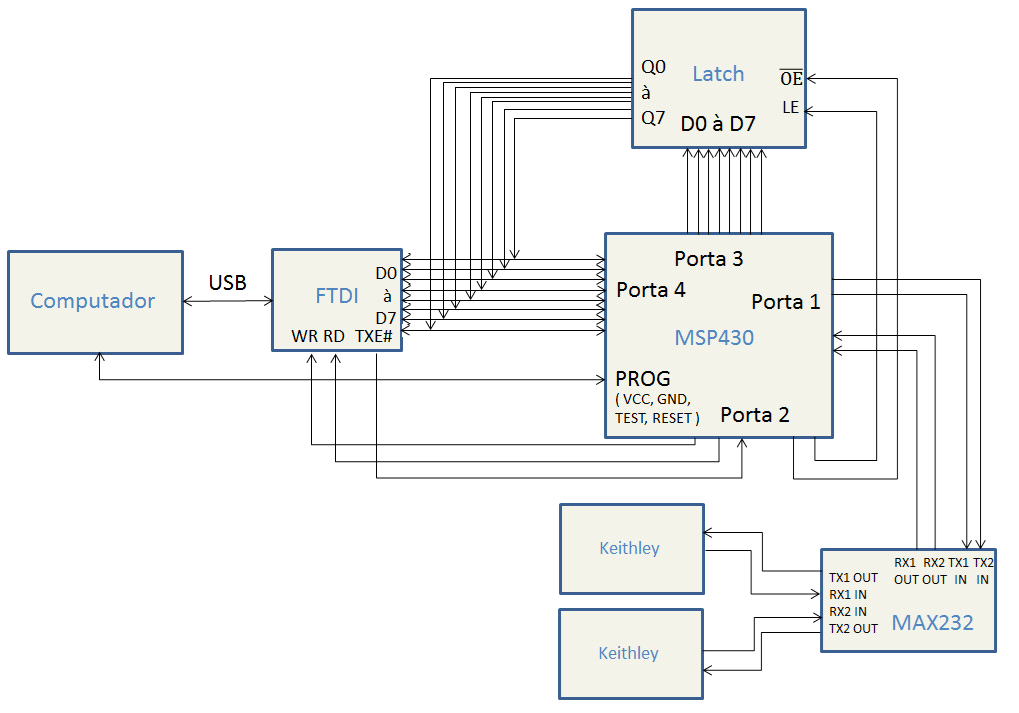


Figura 10: Diagrama de blocos do circuito.

Uma chave, PROG, foi incluída no circuito com o objetivo de facilitar a reprogramação do microcontrolador, alternando a configuração dos fios entre os modos programação e execução.

No projeto desenvolvido, a porta 1 ficou responsável pela comunicação com os dispositivos medidores. Para essa porta, os pinos 0 e 1 funcionam como saída para os pinos TX1 e TX2, respectivamente, os quais são os pinos de entrada serial dos equipamentos 1 e 2. Já os pinos 2 e 3 funcionam como entrada dos pinos RX1 e RX2, respectivamente, os quais emitem os dados seriais dos equipamentos 1 e 2. A porta 2 ficou dedicada ao controle da FTDI e do latch. Para essa porta, o pino 0, 1 e 2 destina-se ao controle dos pinos da FTDI, WR, RD e TXE, respectivamente. Os dois primeiros são saídas, enquanto o ultimo é entrada, o qual indica que novos dados estão prontos para serem escritos no microcontrolador. Já os pinos de saída 3 e 4 são responsáveis pelos pinos do latch, e LE, respectivamente. A porta 3, saída, ficou responsável pelo envio de dados para o FTDI e a porta 4, entrada, pela recepção de dados da FTDI.

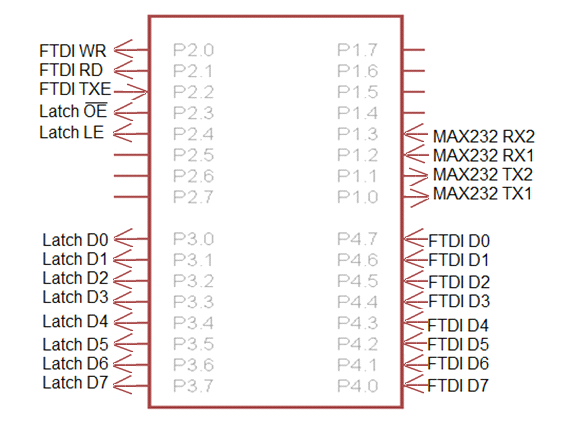


Figura 11: Configuração dos pinos das portas do MSP430.

Inicialmente o microcontrolador está em estado de ócio, esperando dados da FTDI. Isso só ocorrerá quando o usuário do sistema desejar fazer uma requisição ao Keithley model 2400, pressionando uma tecla do programa no computador, que chamará a função da biblioteca FTD2XX.DLL, que se conecta diretamente à FTDI via interface USB. A FTDI, então, manda um sinal ao microcontrolador que está pronta para enviar dados, abaixando o nível lógico do pino TXE#. O microcontrolador, ao receber esse nível lógico baixo, prepara a rotina para receber o byte paralelo também da FTDI pela da porta 4, fazendo o pino RD cair o nível lógico a cada byte recebido. Ao receber um byte específico, pré-definido como \n, indicando fim de transmissão, o microcontrolador repassa os bytes recebidos de forma serial pelo pino TX1 ou TX2 para o MAX232, dependendo para qual equipamento se destina os dados, o que se é definido no primeiro byte recebido. Caso o equipamento devolva alguma resposta, dependendo dos dados previamente recebidos, o microcontrolador os recebe também do MAX232, de forma serial pelo pino RX1 ou RX2, dependendo de qual equipamento a resposta se originou. Em seguida, entra na rotina de envio para a FTDI, baixando tanto o nível do pino , liberando a saída do latch, quanto o nível de WR para cada byte emitido pela porta 3. Após esse processo o microcontrolador entra no estado, novamente, de espera por novos dados da FTDI. A figura 12 mostra a maquina de estado de controle da monitoração dos equipamentos.

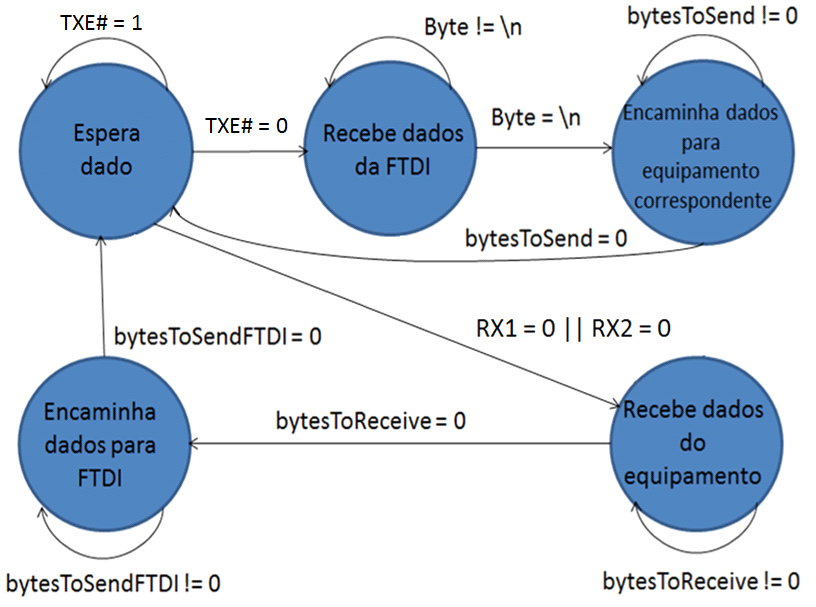


Figura 12: Máquina de estado do microcontrolador.

**4. Programação**

Uma vez que o microprocessador permite o uso da linguagem C, essa foi escolhida por abstrair funções de baixo nível e por ser facilmente legível ao programador.

**4.1 IDE**

O ambiente de programação do microcontrolador escolhido para esse projeto foi o IAR Embedded Workbench IDE. Além de ser grátis para códigos abaixo de 4KB, o mesmo permite programação para várias famílias do MSP430 em C/C++, possibilitando depuração por simulação e tempo real.

A programação do MSP430 é simples, necessitando apenas a ligação do USB do computador com quatro pinos específicos do microcontrolador, VCC, TEST, RESET e GND, para a gravação do programa na memória flash. Após realizadas as respectivas ligações, e compilado o programa desenvolvido, basta baixar pela IDE o código no MSP430. A partir daí, pode-se executar, pela IDE, linha a linha do código, permitindo depurações dos registradores e variáveis no controlador, ou selecionar execução direta, fazendo o programa rodar livremente.

**4.2 Programação Embarcada**

A fim de impedir o travamento, o programa do microcontrolador foi desenvolvido com máquinas de estado, onde os loops e espera por eventos são administrados dando oportunidade para outras rotinas serem chamadas.

**4.2.1 Inicializações**

Inicialmente, inclui-se a biblioteca do microcontrolador, io430.h, a qual contém os endereços de todos os registradores, pinos e definições de constantes nativas. Para facilitar o projeto, foram definidas constantes para acessar diretamente certos pinos. Para esse modelo, isso é feito especificando um nome para o conjunto: porta em questão, pino desejado e de que forma se comporta: entrada ou saída. Por exemplo, para o pino 0 da porta 1, que é uma saída, TX1, a seguinte linha especifica o nome que será utilizado quando se desejar trabalhar com esse pino: #define TX1 P1OUT\_bit.P0. Também foram definidos TX2: P1OUT\_bit, RX1: P1IN\_bit.P2, RX2: P1IN\_bit.P3, WR: P2OUT\_bit.P0, RD: P2OUT\_bit.P1, OE: P2OUT\_bit.P3, LE: P2OUT\_bit.P4.

No projeto, optou-se por desabilitar o watchdog, timer que reinicia o processador se não for periodicamente desarmado. Isso é feito atribuído ao código a seguinte linha: WDTCTL = WDTPW + WDTHOLD.

Em seguida, os pinos das portas devem ser configurados para funcionar como entrada ou saída e se ocorrerá interrupções na subida ou na descida de nível deles. Os registradores que guardam informações dos pinos guardam oito bits, onde cada bit representa um pino daquela porta, sendo o bit menos significativo a informação do pino 0, enquanto o mais significativo o do pino 7.

PXSEL, onde X representa o número da porta, ao receber o valor ‘0’ em determinado bit, configura o pino correspondente para se trabalhar no modo entrada e saída. Caso recebesse ‘1’, configuraria tal pino para se trabalhar em outra função do microcontrolador, dependendo do pino. Como no projeto em questão só se deseja trabalhar nos pinos no modo entrada e saída, todos os quatro registradores PXSEL receberam o valor 0x00, que representa o número em hexadecimal, que transformado para binário representa oito zeros, atribuindo ‘0’ a todos os pinos.

PXDIR configura os pinos da porta X para se trabalhar ou como entrada, valor ‘0’, ou como saída, valor ‘1’. Para o projeto, de acordo com a figura 11, esses registradores foram configurados da seguinte forma: P1DIR = 0x03, P2DIR = 0x1B, P3DIR = 0xFF, P4DIR = 0x00.

PXIE configura os pinos da porta X para executar interrupções. Se receber ‘1’ no bit correspondente, aquele pino estará configurado para encaminhar o programa para uma rotina de tratamento da interrupção. PXIES determina se a interrupção ocorrerá na subida, valor ‘0’ ou descida de nível, valor ‘1’. No projeto, as interrupções da porta 1 ocorrem na descida de nível de RD1 e RD2, enquanto que na porta 2 ocorre na descida de nível de TXE#, então tanto para PXIE quanto para PXIES as configurações para porta 1 ficaram: 0x0C, e para porta 2: 0x04. PXIFG armazena as flags dos pinos de interrupção. Sempre que ocorre uma interrupção, o valor ‘1’ é armazenado no bit correspondente ao pino em PXIFG, não permitindo que a mesma interrupção ocorra até esse valor ser novamente zerado. A habilitação das interrupções é feita configurando o registrador de status com o seguinte código: \_\_bis\_SR\_register(GIE), onde o parametro GIE representa interrupções gerais habilitadas.

O MSP430 tem é dotado de osciladores que podem gerar o sinal de clock. Para o projeto em questão, foi escolhida a frequência de 8 MHz, a qual já possui constates definidas na biblioteca io430.h que configuram o clock auxiliar com os seguintes comandos: DCOCTL = CALDCO\_8MHZ e BCSCTL1 = CALBC1\_8MHZ. DCOCTL configura a frequência e o tipo de modulação do oscilador interno e BCSCTL1 configura o clock auxiliar. Outras frequências podem ser configuradas de acordo com [8].

Das variáveis internas do microcontrolador, configuram-se seus valores das seguintes maneiras: como os equipamentos Keithley trabalham no protocolo serial RS232, as saídas TX1 e TX2 recebem o valor ‘1’, pois representa o nível lógico que o protocolo opera quando não se está recebendo dados. Dos pinos que vão para FTDI, WR fica em ‘0’, enquanto RD em ‘1’. Já para os que ligam o Latch, LE em ‘0’ e OE em ‘1’. Inicialmente a flag ocupado está em 0, pois representa que o microcontrolador não está ocupado. A forma como se armazenará os bytes recebidos será através de um buffer que será controlado através das variáveis indexBuffer, que indicará a posição de envio ou recepção em questão, e tamBuffer que indicará o final do mesmo. Inicialmente ambas são configuradas como ‘0’.

**4.2.2 Interrupções e Timer**

No MPS430 as rotinas de interrupção das portas são declaradas da seguinte maneira: #pragma vector = PORTX\_VECTOR, onde X representa a porta em questão, e em seguinda a função da interrupção: \_\_interrupt void função (void) {}. A interrupção só estará novamente habilitada após se zerar a flag PXIFG.

Já a rotina do Timer é declarada da seguinte forma: #pragma vector = TIMERA1\_VECTOR, e em seguida a rotina: \_\_interrupt void rotinaTimer(void){}. No MSP430 o Timer pode operar em até três modos [9], mas para o projeto em questão foi selecionado o modo UP, onde o timer contará até um valor determinado, que ao alcançá-lo, chama a rotina. Tal valor fica armazenado no registrador TACCR0. Para habilitar a contagem do Timer nesse modo, atribui-se ao registrador TACTL, responsável pelo controle do Timer, as constantes mascaradas: TASSEL\_2 (clock interno auxiliar) + ID\_3 (divisor do clock, 3 representa /4) + MC\_1 (modo UP) + TAIE (habilita interrupção timer) + TACLR (zera o registrador que guarda o valor dos incrementos do Timer, TAR). No estouro do timer, o mesmo é desabilitado atribuindo-se ‘0’ ao registrador TACTL.

Quatro interrupções foram configuradas para ocorrerem no projeto. Uma se trata da recepção de dados vindos da FTDI, pelo pino TXE#. Duas outras se referem à chegada de dados do equipamento, pelos pinos RD1 e RD2. A quarta é o timer, que será disparado quando o tempo de um bit da recepção ou emissão for computado.

Nas três primeira interrupções, ao serem detectadas, a flag ocupado é levantada, indicando à máquina de estados o ocorrido. Para as duas segundas, em uma variável, é atribuído um valor que representa qual equipamento devolveu a resposta. Além disso é ativado o timer, com valor de TACCR0 representando período de 104µs. Para se chegar nesse valor, faz-se aproximação por cima de: frequência do clock (= 8MHz) / (frequência do timer ( = 1/ 104 µs) \* divisor (= 4) ) = 70. No estouro do Timer, uma variável é incrementada, indicando a o fim do tempo do bit recebido ou transmitido.

**4.2.3 Máquina de estados principal**

A máquina de estados principal, representada na figura 12, inicia no estado esperando a flag ocupado ser levantada. Isso só ocorrerá nas rotinas de interrupção do pino TXE#, RD1 e RD2, o que para esses dois últimos só ocorrerão no caso de um envio de dados prévio requisitar retorno de informações. Uma vez detectado levantamento de ocupado, com RD1 e RD2 intactos, o estado muda para recebe dados da FTDI. Já se ocupado for levantado com RD1 ou RD2 abaixado, representa que um equipamento está devolvendo uma resposta, e o estado muda para recebe dados do equipamento.

No estado recebe dados da FTDI, uma vez que TXE# foi sinalizado, indicando que a FTDI tem bytes para escrever no microcontrolador, o mesmo abaixa o nível de RD, fazendo a FTDI liberar o byte em questão nos pinos de saída, levantando em seguida. O byte é capturado pela porta 4 no buffer no índice indexBuffer, sendo esta incrementada posteriormente. O primeiro byte recebido representa para qual equipamento se destina os bytes, tendo valor ‘2’, caso seja para o equipamento um, ou ‘4’, caso seja para o equipamento 2. Caso o primeiro byte venha diferente desse valor, ou o espaço total do buffer seja excedido, qualquer byte recebido é ignorado até que uma nova mensagem que contradiga esses termos seja recebida. Essa operação é realizada tantas vezes até se receber no buffer o valor 10, que representa o caracter ‘\n’, que indica término do envio de bytes, fazendo o estado mudar para encaminhar para equipamento.

Todos os bytes recebidos no estado anterior, com exceção do primeiro que representa o equipamento, serão enviados via serial no estado encaminhar para equipamento. Enquanto o indexBuffer, que agora tem valor 1 para se excluir o primeiro byte, não chegar no tamanho total do buffer, uma nova máquina de estados é chamada, a máquina envia serial. Após o envio total dos bytes, a máquina principal terá o flag ocupado zerado, e voltará para o estado esperando a flag ocupado ser levantada.

No estado recebe dados do equipamento, onde uma resposta está querendo ser escrita no microcontrolador pelo equipamento em questão, uma nova máquina de estados é chamada, a máquina recebe serial. O estado só mudará para envia para FTDI quando os 90 bytes que compõem a resposta forem recebidos no buffer.

Já no estado envia para FTDI, enquanto os 90 bytes não forem enviados, cada byte do buffer é atribuído à porta 3, em seguida é atribuído ao pino OE o valor ‘0’, permitindo que o dado dessa porta passe pelo latch e chegue na FTDI, tendo agora o microcontrolador a tarefa de levantar WR indicando à FTDI que dados estão disponíveis para serem escritos. No término, o estado volta a ser espera flag ocupado, voltando OE ter valor ‘1’ e ocupado ‘0’.

**4.2.4 Máquina de estados envia serial**

Para cada byte, o estado inicial dessa máquina de estados é start bit. Nesse estado, é conferido o primeiro byte armazenado no buffer. Se o valor for igual a 2, é atribuído a TX1 o valor ‘0’, que será o start bit do protocolo serial, e habilitado o timer para estourar em 104 µs, período de um bit. De modo semelhante, se o valor for igual a 4, é atribuído a TX2 o valor ‘0’. Após isso, o estado muda para espera período.

No espera período, é esperado a interrupção do Timer, onde uma variável timer é incrementada. Quando isso ocorre, o Timer é novamente reativado e é conferido se a variável timer é maior que oito, o que representa fim de envio do byte. Em caso negativo, é verificado, mais uma vez, para qual equipamento se destina a mensagem e, em seguida, atribuído ao TX correspondente o bit do byte em questão. Isso é feito com a seguinte lógica: TX = buffer[indexBuffer] & 0x01, deslocando o buffer um bit para direita para a próxima interação. Em seguida a máquina continua no mesmo estado esperando novamente o timer estourar. Caso a variável timer seja maior que oito, indica que o byte foi totalmente enviado, então atribui-se ao TX correspondente o valor de ‘0’ por dois períodos de bit, indicando dois stop bit. Em seguida a variável indexBuffer é incrementada e o estado muda para start bit, para enviar o próximo byte. Quando essa variável é maior que o tamanho do buffer, o microcontrolador finaliza as chamadas dessa máquina.

A figura 13 representa o diagrama da máquina de estados envia serial.

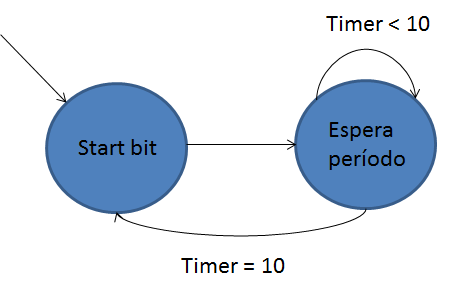


Figura 13: Diagrama da máquina de estados escreve serial.

**4.2.5 Máquina de estados recebe serial**

Quando se inicia as chamadas para essa máquina de estado, significa que o microcontrolador recebeu uma interrupçãono pino RD1 ou RD2, indicando que o equipamento em questão deseja enviar uma resposta. Se a recepção foi em RD1, é atribuído ao buffer na posição 0 o valor 2. Caso tenha sido de RD2, o valor é 4. Em seguida, ainda na interrupção, o timer é acionado para estourar no período de um bit e meio. A flag de habilitação de interrupção para esses dois pinos só será zerado, permitindo nova interrupção, no final da recepção serial.

No estado inicial, espera período, o microcontrolador ficará esperando o tempo de um bit e meio estourar no timer, o que o fará ignorar o bit de start do protocolo serial e, posteriormente, coletar o bit na metade do pulso. Quando o timer estoura, o estado muda para bit válido.

Nesse estado, é atribuído ao buffer o valor do bit, seja RD1 ou RD2, sendo este deslocado um valor menor que o número de incrementos do timer: byte[indexBuffer] = byte[indexBuffer] + (RX << (timer-1)). Em seguida é conferido se a variável timer é maior que sete, o que representa fim da recepção do byte. Em caso negativo, o timer é habilitado para estourar no período de um bit. No caso positivo, a variável timer é zerada, o indexBuffer é incrementado e o timer é habilitado para estourar no período de três bits, o que o fará ignorar dois bits de stop e um de start. No final, independente do caso anterior ter sido positivo ou negativo, o estado passa a ser espera período. Quando o indexBuffer é maior que 90, significa que a mensagem foi totalmente recebida e o microcontrolador finaliza as chamadas dessa máquina.

A figura 14 representa o diagrama da máquina de estados recebe serial.

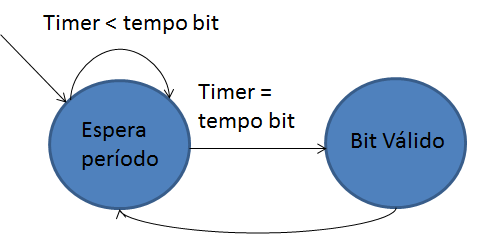


Figura 14: Diagrama da máquina de estados recebe serial.

**4.3 Programação alto nível**

Um programa escrito em Visual C++ é a interface do sistema com o usuário. Utilizando chamadas às funções do driver FT2XX.DLL, é possível ter total controle sobre cada equipamento, o que ocorre quando se pressiona um botão da interface. A partir dessa ação, os dados fluem do computador para a FTDI via interface USB, chegando, posteriormente, ao microcontrolador.

As chamadas às funções dessa biblioteca sempre retornam variável do tipo FT\_STATUS, que se obtiver conteúdo igual à FT\_OK, indica que o procedimento obteve sucesso, retornando conteúdo diferente deste, caso tenha ocorrido alguma falha.

Antes de qualquer chamada às funções da FTDI, é necessário testar e ligar a conexão do driver com o módulo externo FT245R. Isso é feito chamando a função FT\_Open, que localiza os módulos FTDI. A mesma recebe como parâmetros o index do dispositivo desejado na ordem dos FTDI mapeados no computador, começando de 0, e um registrador do tipo FT\_HANDLE, que é um ponteiro para o endereço ao qual o FT245R está alocado.

De modo semelhante, após o término do uso do programa, é feita a chamada à função que encerra a conexão: FT\_CLOSE, que recebe como parâmetro o endereço ao qual o FT245R está alocado.

Quando se deseja fazer uma escrita na FTDI, é feita a chamada à função que escreve bytes na FTDI: FT\_WRITE, que recebe como parâmetros o endereço ao qual o FT245R está alocado, a variável que guarda os bytes a serem enviados, o número de bytes a se enviar e um registrador ao qual estará o número de bytes transmitidos com sucesso.

De modo semelhante, quando se deseja fazer uma leitura da FTDI, é feita a chamada à função que lê bytes da FTDI: FT\_Read, que recebe como parâmetros o endereço ao qual o FT245R está alocado, a variável que guardará os bytes recebidos, o número de bytes a se ler e um registrador ao qual estará o número de bytes recebidos com sucesso.

O tempo máximo de espera pelos bytes é definido fazendo-se chamada à função que configura o timeout: FT\_SetTimeouts, recebe como parâmetros o endereço ao qual o FT245R está alocado, o tempo máximo de espera em milissegundos para a leitura e o tempo máximo de espera para a escrita.

A figura 15 mostra um exemplo de como se utilizar as funções da FTDI explicadas acima.

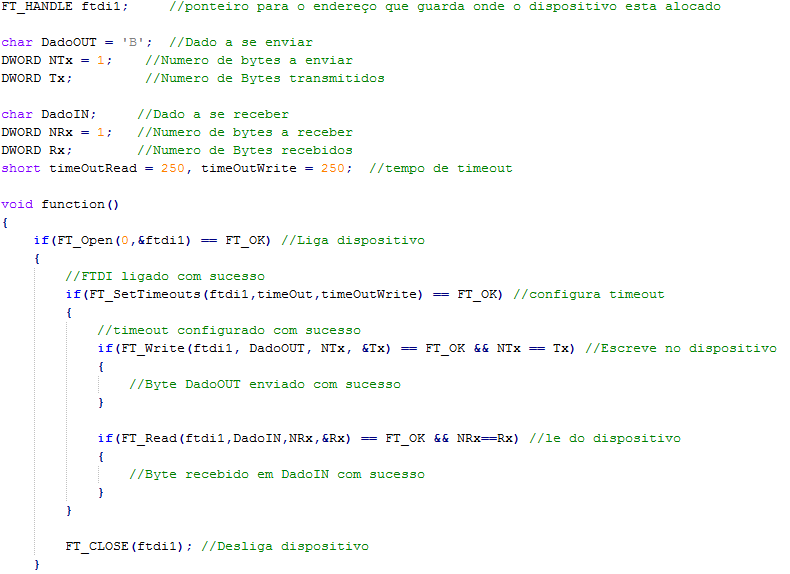


Figura 15: Exemplo rotina de escrita e leitura da FTDI pelo computador.

No projeto, o microcontrolador foi programado para interpretar que ao receber o primeiro byte como tendo valor ‘2’, a mensagem se destina ao dispositivo um. Se tiver valor ‘4’, a mensagem se destina ao dispositivo dois. Então, para cada mensagem emitida pelo computador, ida do FTDI para o microcontrolador o programa em alto nível envia como primeiro byte o identificador do dispositivo de acordo com a solicitação do usuário.

Após essa etapa, cada byte da mensagem que define a requisição, exemplificadas na tabela 4, é enviado. Em outras palavras, cada letra da requisição é enviada ao microcontrolador pela FTDI. No término disso, uma função envia o caractere ‘\n’ como sinal ao microcontrolador que a mensagem já foi totalmente emitida.

No caso da requisição solicitar resposta, o programa entra em estado de espera pelos 90 bytes. Como a transmissão da maior mensagem entendida pelo equipamento leva 45,76 ms, e a recepção serial pelo microcontrolador tem bit de start e dois bits de stop, com um byte válido e baudrate de 9600bits/s, o tempo mínimo de recepção será: ( 45,76 ) + 90 \* 11 \* 0,104 ms = 148,72 ms. No programa em alto nível foi dado um tempo de espera de 200 milissegundos, tempo suficiente os dados trafegarem indo e voltando todo o circuito, pois o tempo de conversão nos hardwares influencia muito pouco no tempo total. O timeout foi definido maior que o tempo total de tráfego para considerar o tempo de processamento dos equipamentos das mensagens recebidas.

A figura 16 representa exemplo das rotinas de escrita e leitura de dados pela FTDI.

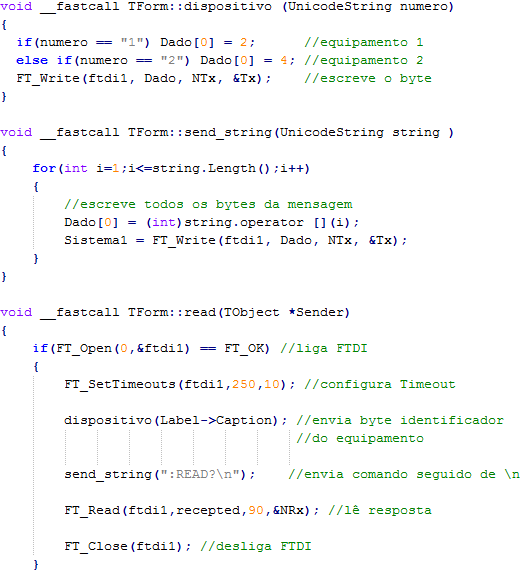


Figura 16: Exemplo rotina de requisição de leitura de um equipamento.

**4.3.1 Aplicações desenvolvidas**

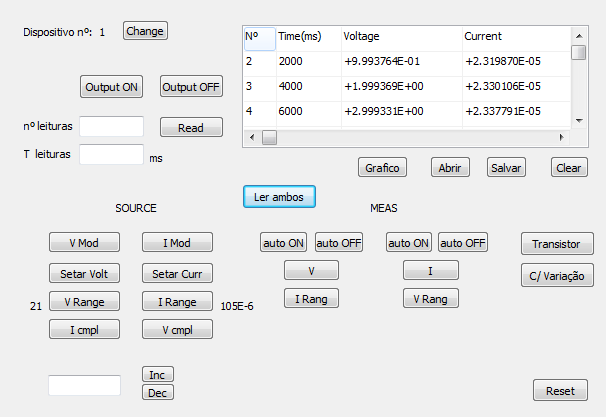


Figura 17: Interface principal desenvolvida.

Além do desenvolvimento das principais funcionalidades dos botões da tabela 3, seguindo as mensagens como na tabela 4, que correspondem basicamente à configurações e solicitações de medições, outras aplicações foram desenvolvidas para facilitar o estudo de certos componentes, como os transistores. As figuras 17, 18 e 19 mostram interfaces de aplicações desenvolvidas utilizando-se o sistema.

Uma aplicação é a leitura em série. Depois de configurado o número de leituras desejadas e o tempo de espera entre elas, as solicitações de leitura são realizadas e armazenadas numa tabela, onde posteriormente podem ser salvas em outro formato ou utilizadas para gerar um gráfico, também desenvolvido em tempo real pela interface. Também existe a opção de se ler com o mesmo número de séries e tempo de intervalo para ambos os equipamentos simultaneamente.

Outra aplicação é a leitura em série das medidas com variação de configurações simultaneamente. Depois de efetuada a configuração, o valor inicial da configuração, o número de medições, a taxa de variação da configuração e o tempo entre as leituras, cada medida é realizada seguida de posterior variação da configuração. Os dados são armazenados numa tabela permitindo exportação para outros arquivos e formatos, além de poderem ser transformados em gráficos pela própria interface.

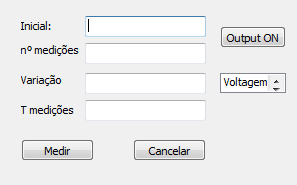


Figura 18: Interface da aplicação de leitura com variação de configuração.

Para o estudo de transistores em particular, uma aplicação desenvolvida foi a leitura da corrente entre coletor, emissor, variando-se a tensão entre esses terminais e a corrente na base. A tarefa de medir e variar a tensão cabe ao equipamento um, enquanto o papel de variar a corrente cabe ao dispositivo dois. As medições são armazenadas na tabela, permitindo, também, serem salvas e exportadas, e plotadas em gráfico na interface.

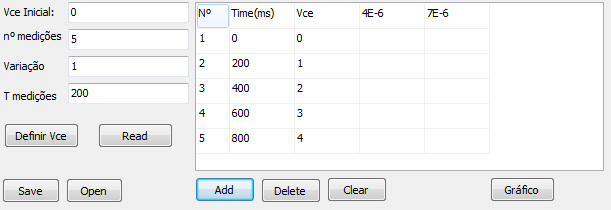


Figura 19: Interface da aplicação de leitura para transistores.

**5. Conclusão**

O sistema desenvolvido torna o árduo trabalho de configurar, medir, tomar notas e processar os dados manualmente de equipamentos do tipo Keithley model 2400 à simples tarefa de solicitá-los computacionalmente e programar novas funcionalidades.

Toda a parte de comunicação é abstraída, permitindo o usuário à comunicação direta com os diversos equipamentos do laboratório utilizando somente um computador com o programa e o módulo eletrônico externo desenvolvido.

Gráficos, tabelas, relatórios e requisições alternadas e em série são exemplos de funcionalidades que podem ser facilmente implementadas com a interface desenvolvida.

**6. Referencias Bibliográficas**

[1] en.wikipedia.org/wiki/Microcontroller

[2] sites2.jf.ifsudestemg.edu.br/sites/default/files/Minicurso%20-%20MSP430.pdf

[3] www.ti.com/lit/ds/symlink/msp430f2274.pdf

[4] [www.capriconsultorios.com/Aula4-Comun\_serial.pdf](http://www.capriconsultorios.com/Aula4-Comun_serial.pdf)

[5] www.ftdichip.com/Support/Documents/ProgramGuides/D2XX\_Programmer%27 s\_Guide%28FT\_000071%29.pdf

[6] [pt.wikipedia.org/wiki/MAX232](http://pt.wikipedia.org/wiki/MAX232)

[7] intro.phys.psu.edu/class/p457/experiments/html/keithley\_2400\_sourcemeter\_full. pdf

[8] www.feng.pucrs.br/~stemmer/labproc1/apostila430/msp04.html

[9] [www.ccs.neu.edu/home/noubir/Courses/CSU610/S07/MSP430-Clock-Timers.pdf](http://www.ccs.neu.edu/home/noubir/Courses/CSU610/S07/MSP430-Clock-Timers.pdf)

[10] [homepages.ius.edu/RWISMAN/C335/HTML/msp430Timer.HTM](http://homepages.ius.edu/RWISMAN/C335/HTML/msp430Timer.HTM)

[11] [www.atmel.com/Images/7799s.pdf](http://www.atmel.com/Images/7799s.pdf)

[12] www.rogercom.com