UNIVERSIDADE ESTADUAL DE CAMPINAS

EA871 Laboratório de Programação Básica de Sistemas Digitais

Projeto Final – Osciloscópio Digital

Lucas Alves Martins RA: 158144
Bruno Edson Limeira RA: 141470

Campinas
21/11/2016

Especificação do problema

O problema consiste em realizar a implementação de um Osciloscópio Digital usando o ADC do microcontrolador. Este osciloscópio terá ajuste de escalas verticais e horizontais, comando de trigger pelo teclado para aquisição dos sinais e diversas funções para apresentar as medidas realizadas desde grandezas elétricas até impressão da forma de onda.

Metodologia

Existe um ajuste de escala vertical (tensão) e escala horizontal (tempo) para melhorar a visualização das medidas de acordo com o tipo de sinal medido. Além disso, antes de fazer as medidas 1 e 2 abaixo, um sinal de trigger externo (botão do teclado) deve ser enviado para fazer a leitura da forma de onda e armazená-la na memória.

Lista de medidas realizadas pelo osciloscópio (comandos enviados pelo teclado):

1. **Medidas de parâmetros estático**: Imprimir no terminal o valor corresponde ao comando.
    - Valor máximo
    - Valor mínimo
    - Valor pico a pico
    - Valor médio
    - Valor RMS
2. **Forma de onda**: Imprime no terminal a forma de onda que está armazenada na memória (eixo X = tempo, eixo Y = amplitude).
3. **Lista de valores**: Imprimir no terminal em tempo real o valor correspondente da amplitude.

A frequência do sinal de entrada é limitada de acordo com o Teorema de Nyquist, porém para garantir medidas mais precisas a frequência de amostragem deve ser 10 vezes a frequência máxima do sinal.

A amplitude do sinal de entrada é limitada de acordo com a máxima amplitude suportada pela entrada do ADC no modo single-ended, ou seja, de 0 V a 3,3 V. Não é permitido sinais negativos.

Utilizamos o ADC single-ended 8, que é o pino PTB0 do microcontrolador. Na Figura 1 está representado onde fica este pino na placa 871.

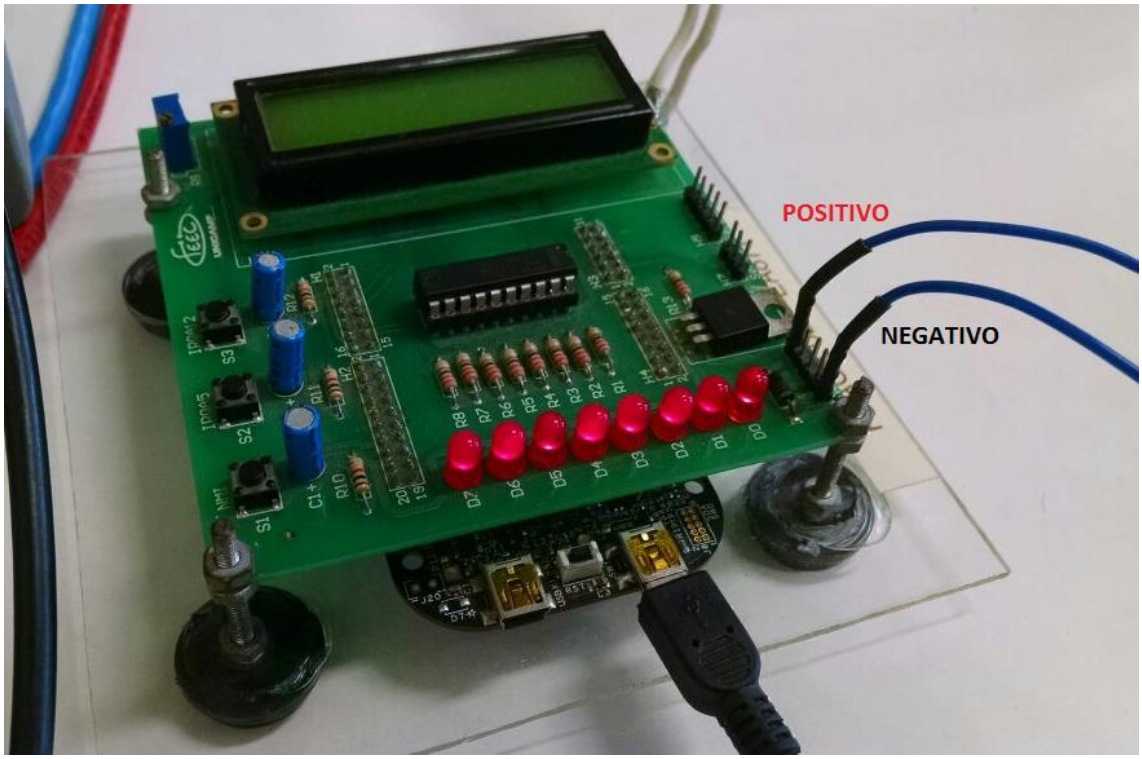

Figura 1: Pinos que devem ser utilizados como entrada do sinal.

Pelo esquemático da placa 871 vemos que estes pinos estão no Header 6, pino 6 para terra e 1 para o PTB0, como mostra a Figura 2.

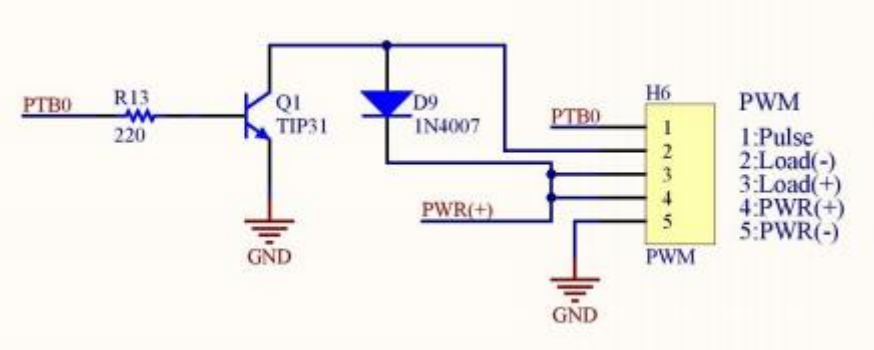

Figura 2: Parte do esquemático que mostra o local do pino PTB0

As configurações dos registradores estão representadas nas figuras seguintes:

```
void initUART(void) {
    /*!
     * Configura&ccedil;&atilde;o de SIM
       */
    SIM_CLKDIV1 &= (uint32_t)~(uint32_t)(SIM_CLKDIV1_OUTDIV4(0x7));  ////< bus_clock = clock/1
    SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK;
    SIM_SCGC4 |= SIM_SCGC4_UART0_MASK;
    SIM_SOPT2 |= SIM_SOPT2_UART0SRC(0x1);   ///< configura a fonte de relógio (20.971520MHz)
    SIM_SOPT2 &= ~SIM_SOPT2_PLLFLLSEL_MASK;
                                            ///< no reset, o valor é zero (FLL)
                                            ///< mas não deixa de ser uma boa
                                            ///< prática de inicializar explicitamente

    /*!
     * Configura&ccedil;&atilde;o dos pinos que servem UART0
       */
    PORTA_PCR1 |= PORT_PCR_MUX(0x2);         ///< UART0_RX
    PORTA_PCR2 |= PORT_PCR_MUX(0x2);         ///< UART0_TX

    /*!
     * Configura&ccedil;&atilde;o do m&oacute;dulo UART0
       */
    UART0_C1 &= ~(UART0_C1_PE_MASK |
            UART0_C1_ILT_MASK |
            UART0_C1_WAKE_MASK |
            UART0_C1_M_MASK |
            UART0_C1_DOZEEN_MASK |
            UART0_C1_LOOPS_MASK );
    UART0_C2 &= ~(UART_C2_RE_MASK |
                UART_C2_TE_MASK);
    UART0_BDH &= ~(UART_BDH_SBNS_MASK |
                UART_BDH_RXEDGIE_MASK |
                UART_BDH_LBKDIE_MASK);
    UART0_C4 |= UART0_C4_OSR(0xF);
    /*!
     * baud rate 19200 (Se&ccedil;&atilde;o 8.3.2 em
     * ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf)
     */
    UART0_BDH |= UART_BDH_SBR(0x00);
    UART0_BDL |= UART_BDL_SBR(0x44);

    UART0_S1 |= (UART0_S1_PF_MASK |              ///< Registradores de estado: w1c
                UART0_S1_FE_MASK |
                UART0_S1_NF_MASK |
                UART0_S1_OR_MASK |
                UART0_S1_IDLE_MASK);

    UART0_C2 |= (UART_C2_RIE_MASK |
        UART_C2_RE_MASK |
        UART_C2_TE_MASK);

}
```

Figura 3: Configuração dos registradores da UART

```
void InitSysTick (void) {
    /*!
     * core clock: 20.971520MHz
     */
     SYST_RVR = SysTick_RVR_RELOAD(4194);         //!Configura interrupção a cada 10us
     SYST_CVR = SysTick_CVR_CURRENT(2097);
     SYST_CSR |= (SysTick_CSR_CLKSOURCE_MASK);
     SYST_CSR &= ~(SysTick_CSR_ENABLE_MASK | SysTick_CSR_TICKINT_MASK);
}
```

Figura 4: Configuração dos registradores do SysTick

```
void calibraADC (void) {
    uint16_t tmp=0;

    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;      ///< habilita clock do m&oacute;dulo ADC0
    /*! Configura&ccedil;&atilde;o do rel&oacute;gio
     * Os bits de ADC0_CFG1 s&atilde;o resetados em 0:
     * ADLSMP=0 (short sample time)
     */
    ADC0_CFG1 &= (uint32_t)~(uint32_t)(ADC_CFG1_ADLPC_MASK); ///< opera&ccedil;&atilde;o normal
    ADC0_CFG1 |= ADC_CFG1_ADIV(0x3) |              ///< ADCK = (source clock)/8
                 ADC_CFG1_ADLSMP_MASK |            ///< intervalo de amostragem longo
                 ADC_CFG1_MODE(0x3) |             ///< convers&atilde;o em 16 bits
                 ADC_CFG1_ADICLK(0x1);            ///< (source clock) = (bus clock)/2
    /*! Configura&ccedil;&atilde;o dos ciclos adicionais
     * Os bits de ADC0_CFG2 s&atilde;o resetados em 0: MUXSEL = 0 (canais ADxxa)
     */
    ADC0_CFG2 &= (uint32_t)~(uint32_t)(
                  ADC_CFG2_ADACKEN_MASK |     ///< desabilita a sa&iacute;da do sinal de clock
                  ADC_CFG2_ADLSTS(0x3));     ///< 24 (20 extra+4) ADCK ciclos por amostra
    ADC0_CFG2 |=  ADC_CFG2_ADHSC_MASK;       ///< segu&ecirc;ncia de convers&atilde;o em alta velo
    /*! Os bits de ADC0_SC2 s&atilde;o resetados em 0: ACFE=0 (desabilitada fun&ccedil;&atilde;o c
    ADC0_SC2 &= ADC_SC2_REFSEL(0x0);    ///< Tens&otilde;es de refer&ecirc; VREFH e VREFL

    /*! Os bits de ADC0_SC3 s&atilde;o resetados em 0 */
    ADC0_SC3 |= ADC_SC3_AVGE_MASK | ///< habilita fun&ccedil;&atilde;o em hardware
                ADC_SC3_AVGS(0x3);  ///< n&uacute;mero de amostras por m&eacute;dia = 32
    ADC0_SC3 |= (uint32_t)(ADC_SC3_CAL_MASK);  ///< inicializa a calibra&ccedil;&atilde;o

    while (ADC0_SC3 & (uint32_t)(ADC_SC3_CAL_MASK))  {}  ///< aguarda a calibra&ccedil;&atilde;o

    /*! Gere as corre&ccedil;&otilde;es de erros de ganhos, conforme o procedimento
     *  na Se&ccedil;&atilde;o 28.4.6
     */
    tmp = ADC0_CLP0 + ADC0_CLP1 + ADC0_CLP2 + ADC0_CLP3 + ADC0_CLP4 + ADC0_CLPS; ///< soma PS
    tmp = tmp/2;                            ///< divide por 2
    tmp |= (uint16_t)(0x8000);              ///< set o bit mais significativo
    ADC0_PG = tmp;                          ///< plus-side gain
    /*! Somente &uacute;til para modos diferenciais */
    tmp = ADC0_CLM0 + ADC0_CLM1 + ADC0_CLM2 + ADC0_CLM3 + ADC0_CLM4 + ADC0_CLMS; ///< soma PS
    tmp = tmp/2;                            ///< divide por 2
    tmp |= (uint16_t)(0x8000);              ///< set o bit mais significativo
    ADC0_MG = tmp;                          ///< minus side gain
}
```

Figura 5: Calibração do ADC

A rotina de calibração foi feita baseando-se no que foi especificado no manual.

```
void configuraADC(void) {
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;      ///< habilita clock do m&oacute;dulo PORTB
    /*! Pino que serve um canal do m&oacute;dulo ADC0 */
    PORTB_PCR0 &= (uint32_t)~(uint32_t)((PORT_PCR_ISF_MASK |        ///< baixa bandeira
                    PORT_PCR_MUX(0x7)));                            ///< fun&ccedil;&atildeo ADC0
    /*!
     * Configura&ccedil&atilde;o do rel&oacute;gio: convers&atilde;o em 16 bits,
     * fregu&ecirc;ncia ADCK = (bus clock)/2*8
     */
    ADC0_CFG1 &= (uint32_t)~(uint32_t)(
              ADC_CFG1_ADLSMP_MASK ) ;   ///< intervalo de amostragem curto
    ADC0_CFG1 |= ADC_CFG1_ADLPC_MASK;    ///< opera&ccedil;&atilde;o low-power
    /*!
     * Configura&ccedil;&atilde;o da velocidade: MUXSEL=0 (canais ADxxa), ADACKEN=0 (desabilitada
     * sa&iacute;da ass&iacute;ncrona, ADLSTS=00 (24 ciclos)
     */
    ADC0_CFG2 &= (uint32_t)~(uint32_t)(
              ADC_CFG2_ADHSC_MASK);      ///< segu&ecirc;ncia de convers&atilde;o normal
    /*!
     * Configura o modo de convers&atilde;o: ADCO=0 (modo não cont&iacute;nuo), AVGE=0
     */
    ADC0_SC3 &= (uint32_t)~(uint32_t)(
                ADC_SC3_AVGS(0x3));    ///<  n&uacute;mero de amostras por m&eacute;dia = 4
    ADC0_SC3 |= ADC_SC3_CALF_MASK;      ///< baixa a bandeira de "calibration failed"

    /*! ADC0_SC1A: COCO=0, DIFF=0 (medida unipolar) e ADCH=0b11111 (canais desabilitados) no reset */
    ADC0_SC1A |= ADC_SC1_AIEN_MASK;           ///< habilita interrup&ccedil;&atilde;o
}
```

Figura 6: Configuração dos registradores do ADC

```
void enableNVIC(void) {

    NVIC_IPR3 = (uint32_t)((NVIC_IPR3 & (uint32_t)~(uint32_t)( ///< NVIC_IPR3: PRI_15=0x80
                  NVIC_IP_PRI_15(0x7F)
                 )) | (uint32_t)(
                  NVIC_IP_PRI_15(0x40)
                 ));
    NVIC_ISER |= NVIC_ISER_SETENA(GPIO_PIN(31-16));   ///< Habilita interrup&ccedil;&atilde;o ADC0
    NVIC_ICPR |= NVIC_ICPR_CLRPEND(GPIO_PIN(31-16));  ///< Limpa as poss&iacute;veis pend&ecirc;ncias do ADC0

    NVIC_ISER |= GPIO_PIN (12);          //!Habilita interrupção da UART
    NVIC_ICPR |= GPIO_PIN (12);
    NVIC_IPR7 |= NVIC_IP_PRI_12(0x80);   //!Configura Prioridade 2 a interrupção do UART0
    UART0_C2 &= ~UART_C2_TIE_MASK;
}
```

Figura 7: Configuração do NVIC

Proposta de Solução

O pseudocódigo da rotina main está representado a seguir:

```
main():
        Inicializa UART, ADC, SysTick e NVIC;
        Inicializa os buffers circulares;
        Inicializa o buffer de dados zerado;
        Imprime “Osciloscópio Digital”;
        Início do loop infinito:
                Imprime mensagem e comandos do menu inicial;
                Força o comando para ‘H’;
                Espera até um comando ser digitado;
                Executa uma ação baseada no comando;
Fim
```

A rotina main nada mais é do que um menu inicial que irá aguardar o usuário digitar um comando para executar uma certa tarefa. No caso do menu inicial temos cinco tarefas possíveis:

- **Trigger (comando ‘T’):** Ao digitar este comando o microcontrolador irá fazer a amostragem do sinal que estiver em sua entrada através do ADC, armazenando 1000 pontos em um vetor;
- **Configurar (comando ‘C’):** Ao digitar este comando o usuário poderá fazer configurações de amplitude e período de amostragem;
- **Imprimir (comando ‘I’):** Ao digitar este comando alguns pontos do sinal armazenado serão impressos no terminal, seguindo a configuração feita pelo usuário da tarefa Configurar;
- **Lista Dinâmica (comando ‘L’):** Ao digitar este comando o micro controlador enviará ao terminal constantemente o valor em mV da tensão atual do sinal;
- **Medidas (comando ‘M’):** Ao digitar este comando o usuário poderá escolher fazer algumas medidas do sinal armazenado;

Caso o que foi digitado pelo usuário não equivalha a nenhum dos comandos especificados a mensagem “Comando Inválido” será exibida;

Cada uma dessas tarefas pode ser representada por um pseudocódigo, a da tarefa Trigger é o seguinte:

Trigger():
Imprime frase “Adquirindo dados, agruarde...”;
Configura o tempo de contagem total do Systick baseado no que foi configurado;
Zera o contador do Systick;
Habilita o Systick;
Até que o vetor de dados esteja completamente preenchido:
Espera a flag COUNTFLAG do Systick levantar;
Zera campo do ADC;
Habilita a leitura do ADC;
Espera que a leitura do ADC termine;
Armazena a leitura no vetor de dados;
Desabilita o Systick;
Imprime a frase “Amostragem Concluída”;

Para este programa estamos utilizando um vetor de tamanho 1000 para guardar o sinal amostrado, este vetor está declarado como uma variável global para que possa ser acessado em qualquer parte do programa. A tarefa que armazena os dados no vetor de dados é o Trigger. Como pode ser observado no pseudocódigo utilizamos o Systick como base de tempo entre os intervalos de medições. Como foi explicado anteriormente este tempo deve ser configurado pelo usuário e tem como default 200 us.

A flag do SysTick COUNTFLAG levanta toda vez que o contador vai de 1 para 0, ou seja, sempre que termina uma contagem. O uso do Sysytick se faz necessário pois para gerar um período de amostragem coerente precisamos que a contagem do tempo ocorra em paralelo com o fluxo do programa, assim podemos desconsiderar o tempo de aquisição do ADC.

Para fazer o coreto armazenamento do valor medido utilizamos:

```
data[i]=((float)(value)*(VREFH-VREFL))/MAX;
```

No caso (VREFH – VREF)/MAX calcula o valor em mV equivalente a um bit, quanto multiplicamos isto pelo que foi convertido pelo ADC temos o valor em mV da medida. VREFH e VREFL devem ser os valores em mV da referência que o micro controlador está utilizando, no caso 0 para VREFL e 3300 para VREFH.

O pseudocódigo para a tarefa Configurar é o seguinte:

```
Configurar():
    Enquanto o comando digitado não for um 'S':
        Força comando como 'H';
        Imprime frase e comandos do menu configurar;
        Espera um comando ser digitado;
        Executa o comando referente ao que foi digitado;
```

São quatro comandos que podem ser utilizados: M para ajuste do valor máximo; B para ajuste do valor mínimo; T para medida do tempo de aquisição; e S para voltar ao menu inicial. Os três primeiros comandos mencionados são executados da mesma maneira, mudando apenas onde são armazenadas as configurações:

```
Execução dos comandos():
    Imprime a frase exibindo os limites;
    Lê o número inteiro digitado;
    Se estiver fora dos limites:
        Imprime a frase "Número Fora dos Limites";
        Lê número inteiro digitado;
    Senão:
        Armazena a configuração no local adequado;
```

Ao digitar 'S' o programa volta para a rotina principal.

O pseudocódigo para obtenção do número inteiro digitado é:

```
Leitura do número digitado():
    Cria uma variável que funcionará de acumulador;
    Enquanto um enter não for digitado:
        Se o buffer do receptor não estiver vazio:
            Se o caractere for um número:
                Escreve o que foi digitado no terminal;
                Multiplica o valor atual do acumulador por 10;
                Soma o valor digitado ao acumulador;
            Senão for um número:
                Remove o caractere do buffer;
```

O pseudocódigo para impressão é o seguinte:

Impressão():
Até que o número total de linhas esteja preenchido:
Imprime “|”
Até que o número total de colunas esteja preenchido:
Se o dado do vetor estiver no intervalo correto:
Imprime ‘*’;
Senão:
Imprime ‘ ‘;
Retorna e pula linha;
Imprime “|”;
Imprime “-“ até o final da coluna;
Retorna e pula linha;

Para calcular se o dado está no intervalo correto usamos a condição:

```
if((data[j*SIZEVEC/COLUNAS]>=step*i+config[1])&&(data[j*SIZEVEC/COLUNAS
]<=step*(i+1)+config[1]))
```

Assim conseguimos varrer os valores possíveis entre o máximo e o mínimo configurado. Como não é possível imprimir 1000 ponto no terminal o programa pega apenas dados espaçados de forma a caber todo o vetor em uma linha;

Na lista dinâmica o controlador faz uma leitura pelo ADC e imprime o valor medido no terminal, um valor por linha. O usuário, para sair desse modo deverá digitar ‘S’.

Para o comando de medidas o pseudocódigo é o mesmo que o do comando configurar, mudando apenas os comandos possíveis e o que eles fazem.

Testes

Durante todo o desenvolvimento o setup de testes estava como mostrado na Figura 8.

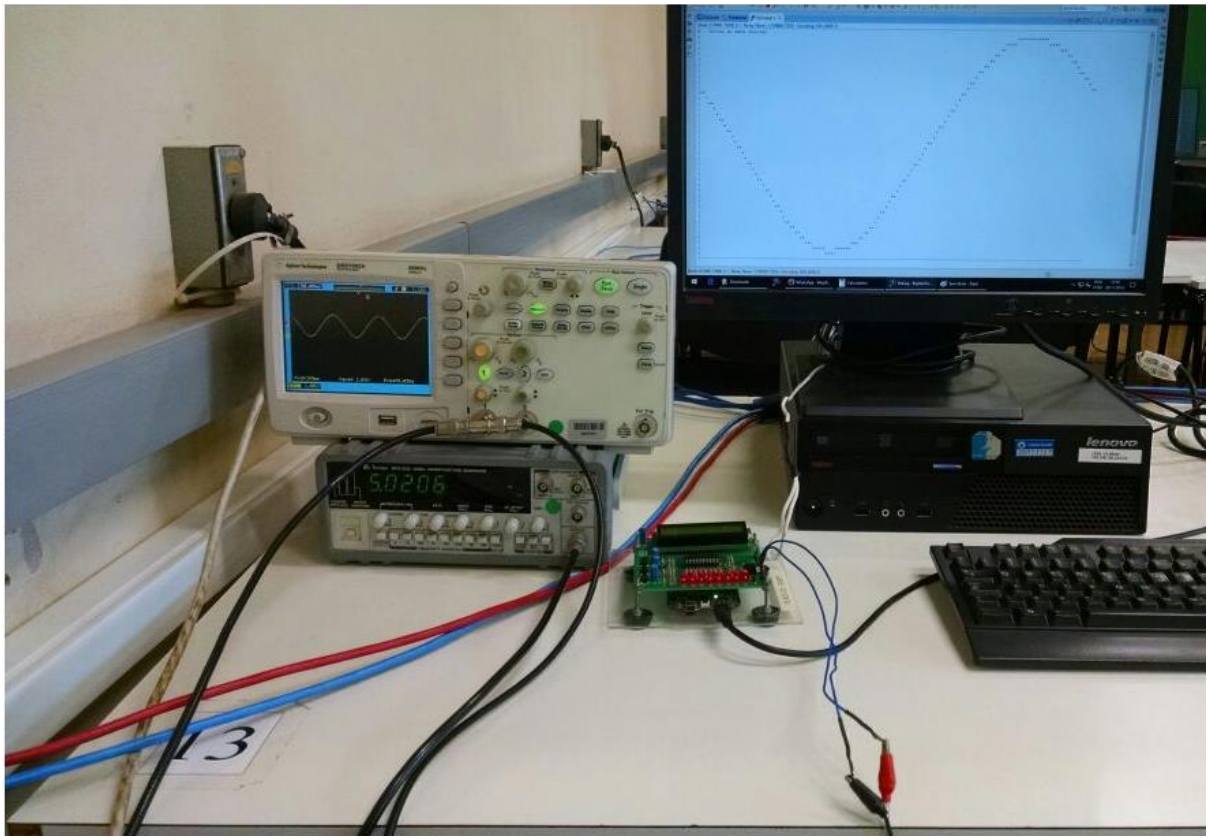

Figura 8: Setup para testes

No caso utilizamos um gerador de frequência para gerar um sinal de teste e o osciloscópio para comparação do que estávamos medindo.

Este projeto foi dividido em etapas, a primeira foi fazer o ADC funcionar. Para isso fizemos um programa que fazia a leitura do ADC e imprimia na tela, uma medida por linha, tivemos algumas dificuldades de interpretação da implementação do ADC, mas no final conseguimos fazer um código que funcionasse adequadamente, este código foi usado em partes no programa final como a função lista dinâmica.

```
1725.4
1640.7
1545.7
1446.1
1339.4
1231.9
1126.6
1013.9
903.4
793.1
699.1
603.6
506.4
410.5
321.3
241.6
174.4
118.4
83.0
57.2
41.9
35.8
47.9
64.9
94.1
133.0
182.2
256.1
344.4
440.1
542.0
639.8
737.4
838.3
939.0
1038.9
1139.8
1252.1
1364.1
1469.5
1574.5
1671.1
1756.0
```

Figura 9: Teste feito de impressão de dados obtidos

O próximo passo foi fazer com que os dados amostrados fossem salvos em um vetor após o envio de comando Trigger, com um tempo de espera entre as aquisições. Para o período de amostragem utilizamos inicialmente uma rotina de delay escrita em assembly depois da amostragem pelo ADC, esta solução não seria viável no programa final pois o tempo programado seria apenas parte do período de amostragem, pois também estaria somado o tempo de leitura do ADC, porém para um intervalo grande entre medições esta solução funciona adequadamente. Fizemos um teste copiando os dados amostrados e plotando no Excel e o resultado foi satisfatório.

Ainda com a aquisição desta forma implementamos as rotinas de medida e impressão no terminal. Levamos algum tempo para pensar na solução que foi implementada no programa final, mas após finalizada conseguimos imprimir uma senóide de 1 Hz na tela do terminal e fazer as medidas corretamente desta forma de onda. Testamos também uma onda quadrada, uma triangular e um dente de serra e todas apresentaram resultados satisfatórios.

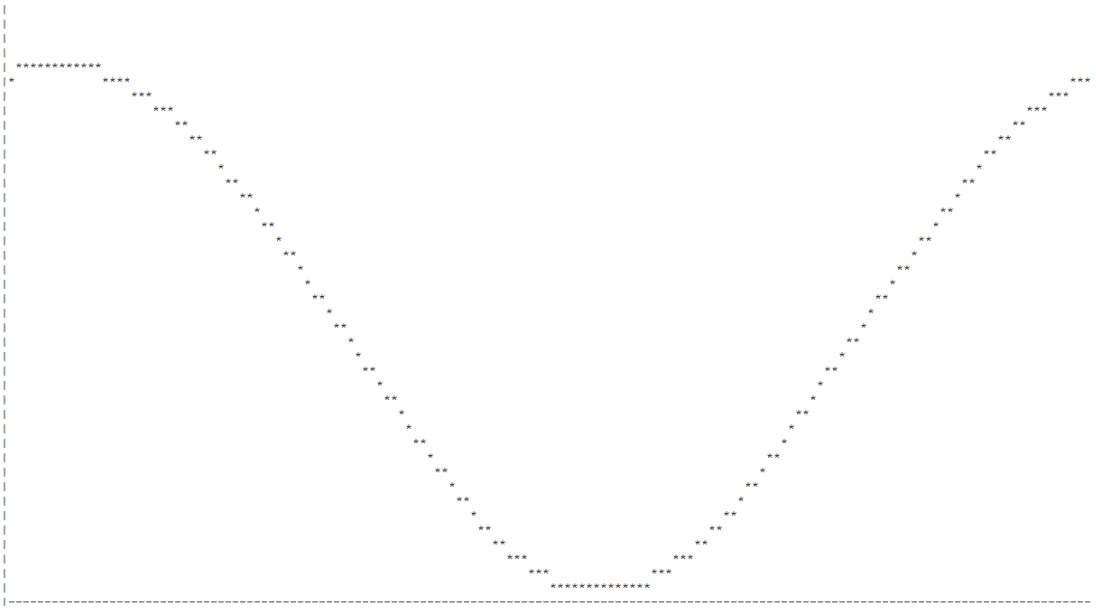

Figura 10: Impressão da forma de onda no terminal

```
DIGITE UM COMANDO
1 - Valor Máximo
2 - Valor Mínimo
3 - Pico a pico
4 - Valor Médio
5 - Valor RMS
S - Voltar ao menu inicial
Máximo = 1960.4mV
DIGITE UM COMANDO
1 - Valor Máximo
2 - Valor Mínimo
3 - Pico a pico
4 - Valor Médio
5 - Valor RMS
S - Voltar ao menu inicial
Mínimo = 31.2mV
DIGITE UM COMANDO
1 - Valor Máximo
2 - Valor Mínimo
3 - Pico a pico
4 - Valor Médio
5 - Valor RMS
S - Voltar ao menu inicial
Pico a Pico = 1929.1mV
DIGITE UM COMANDO
1 - Valor Máximo
2 - Valor Mínimo
3 - Pico a pico
4 - Valor Médio
5 - Valor RMS
S - Voltar ao menu inicial
Média = 1017.2mV
DIGITE UM COMANDO
1 - Valor Máximo
2 - Valor Mínimo
3 - Pico a pico
4 - Valor Médio
5 - Valor RMS
S - Voltar ao menu inicial
RMS = 1215.0mV
```

Figura 11: Medidas feitas de uma forma de onda

Finalmente implementamos o tempo de aquisição com o Systick. Inicialmente utilizamos um tempo de aquisição de 100us, porém notamos pela impressão da forma de onda no terminal que o período de aquisição estava maior que isto, por isso aumentamos o tempo mínimo para 200us. Testamos novamente várias formas de onda e os resultados foram satisfatório até para ondas de maior frequência.

Conclusão

Este projeto foi desafiador, porém bastante gratificante de desenvolver. Tivemos que aprender a utilizar o conversor AD, que não tínhamos tido contato ainda, e aprender a fazer um código do zero.

Tivemos algumas dificuldades ao longo do desenvolvimento, porém com a ajuda da professora e com o estudo mais avançado dos manuais fomos capazes de superar os desafios e gerar um projeto final que consideramos satisfatório.

Tivemos muitas ideias de implementação adicional ao código, como auto scale do osciloscópio, impressão dinâmica da forma de onda no terminal e até FFT do sinal, porém não tivemos tempo de fazer estas implementações. Ao idealizar o projeto pensamos em muitas coisas, porém durante o desenvolvimento notamos que coisas que achávamos que seria simples acabaram tomando bastante tempo.

Referências

KL25 Sub-Family Reference Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf

Wu Shin-Ting e A.A.F. Quevedo. Ambiente de Desenvolvimento – Hardware
http://www.dca.fee.unicamp.br/cursos/EA871/2s2016/UW/apostila/cap1.pdf

Wu Shin-Ting. Modelo de Relatório
http://www.dca.fee.unicamp.br/cursos/EA871/2s2016/UW/roteiros/relatorio_template.tx

Código-exemplo
http://www.dca.fee.unicamp.br/cursos/EA871/1s2016/codes/exp9.c

Manual: Osciloscópio Digital