UNIVERSIDADE FEDERAL DO ABC

Relatório Prática 06: Segunda Prova do Laboratório

Sistemas Microprocessados

Daniel Ermelino Carvalho	RA: 11092613
Danilo dos Santos Perez	RA: 11116716
Gabriel Gomes de Oliveira	RA: 11108214
Robson Leite da Silva	RA: 11057814
Vanessa Tinami Tamashiro	RA: 11063311

SANTO ANDRÉ

2017

1. Introdução/motivação:

Esta atividade prática tem como principal objetivo fazer com que os alunos consigam desenvolver no microcontrolador STM32F103C8T6 em interface com a placa didática disponibilizada pelo professor, a estrutura de um código em linguagem C que simula o funcionamento de um cronômetro digital e que permite com que duas placas se comuniquem. Para a correta execução de tal atividade todos os pontos vistos tanto nas aulas teóricas quanto nas demais aulas práticas serão de suma importância, tais como a utilização do padrão de comunicação USART, para permitir de forma assíncrona a transmissão de dados serial entre dois microprocessadores, dos timers existentes no próprio microcontrolador a partir da ativação da função Systick em conjunto com a rotina de interrupção (ISR) para fazer com que a contagem do cronômetro funcione em um super loop sem usar tantos recursos da CPU, tornando o circuito mais eficiente e otimizando o processo como um todo, assim como também será necessário utilizar um mecanismo de debouncing para que o botão da placa didática funcione de forma adequada quando for necessário usá-lo para realizar as interrupções durante a verificação dos estados do sistema, que no caso serão três: modo local, no qual será mostrado apenas o cronômetro local; modo externo, no qual é mostrado apenas o cronômetro externo; modo dual, no qual a cada 5 segundos é alternado a visualização de qual cronômetro será mostrado, entre o local e o externo.

2. Fundamentação teórica:

Um protocolo de comunicação pode ser definido como um conjunto de regras que determinam como deve ocorrer o processo de comunicação em uma determinada rede, para que todos os componentes presentes na mesma consigam interagir entre si trocando informação para que o sistema funcione corretamente. Tais protocolos são responsáveis por definir os tipos de sinais de requisição de dados que serão utilizados, o tempo que os sinais ficaram disponíveis na rede, a velocidade (baud-rate) de funcionamento da rede, o tipo de arbitragem que será realizado entre os dispositivos da rede para definir quem tem maior prioridade, dentre outras.

Para a realização deste projeto será utilizado o protocolo de comunicação USART (Universal Synchronous Asynchronous Receiver Transmiter), que se trata de uma interface composta pela combinação de outros dois protocolos, o UART e o SPI. Trata-se de um protocolo bidirecional, ou seja, ele trabalha com um modo de comunicação no qual os dispositivos presentes na rede conseguem enviar e receber informação de forma simultânea

através de dois canais distintos (RX/ TX). Sendo assim deverá ser utilizada uma terminação single-ended entre as placas, na qual precisamos interligar os GNDs das mesmas para que ambas fiquem com o mesmo sinal de referência e não haja diferença de potencial elétrico entre os terras ou ainda para evitar o surgimento de ruídos nos fios de comunicação.

3. Metodologia:

Antes de iniciar a implementação do projeto, houve a necessidade de criação de um protocolo de comunicação para que fosse possível a comunicação entre qualquer dois microprocessadores dos grupos da sala. Sendo assim, utilizando o padrão de comunicação USART, os grupos da sala chegaram a um consenso sobre qual seria o protocolo de comunicação a ser seguido.

O próximo passo foi criar um projeto novo no TrueSTUDIO - usamos o nome de Segunda_prova. Em seguida habilitamos as bibliotecas e GPIO's a serem utilizadas no projeto. No quadro 1 encontra-se o programa principal e no quadro 2 o código do "stm32f1xx_it.c".

Quadro 1: Código de principal do programa.

```
/*
Códico do Grupo composto por Daniel Carvalho, Vanessa Tiname, Robson Leite,
Gabriel Gomes e Danilo dos Santos.
*/

#include "stm32f10x.h"
#include "stm32f1xx_it.h"      // prots serv interrupt (ISR, em
stm32f1xx_it.c)

#define FREQ_TICK 1000
#define dt_data 5
#define dt_toggle 5000
#define DT_PC13 50
#define DT_PC14 50


// declação de  ESTRUTURAS de dados: GPIO, EXTI (external int) e NVIC
(contrl vet interrpt)
EXTI_InitTypeDef IntExt_PtB;           // declara estrut IntExt_PtB
NVIC_InitTypeDef NVIC_PtB;             // estrutura NVIC port B
GPIO_InitTypeDef GPIO_PtC;             // estrutura dados config GPIO_C
GPIO_InitTypeDef GPIO_PtB;             // estrutura dados config GPIO_B

uint32_t millis(void);         // prot fn millis (em stm32f1xx_it.c)
uint32_t Ler_Modo_Oper(void);  // prot fn ModoOper (em stm32f1xx_it.c)

void setup_INT_externa(void) //interrupção externa
{
```

```
  // configurar dados estrutura interrupcao
  IntExt_PtB.EXTI_Line = EXTI_Line6;           // qual linha pede
interrupcao
  IntExt_PtB.EXTI_Mode = EXTI_Mode_Interrupt;  // modo interrupcao
  IntExt_PtB.EXTI_Trigger = EXTI_Trigger_Rising;  // dispara no
falling_edge
  IntExt_PtB.EXTI_LineCmd = ENABLE;           // habilita ext_int
  EXTI_Init(&IntExt_PtB);  // chama funÃ§Ã£o que inicializa interrupcao

  // configurar o NVIC (estrutura e funcao no misc.h e misc.c)
  NVIC_PtB.NVIC_IRQChannel = EXTI9_5_IRQn;       // IRQ_ext linha
EXTI9_5_IRQn
  NVIC_PtB.NVIC_IRQChannelPreemptionPriority = 1;// prioridade preempt
  NVIC_PtB.NVIC_IRQChannelSubPriority = 1;       // prioridade 1
  NVIC_PtB.NVIC_IRQChannelCmd = ENABLE;               // habilitada
  NVIC_Init(&NVIC_PtB);    // chama fn que inicializa NVIC
}

void setup_USART(uint32_t baud_rate) {
      GPIO_InitTypeDef GPIO_InitS;
      /* USART */
      //PA2 = USART2 TX
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); // clock do
GPIOA
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);  // clock da
USART
      GPIO_InitS.GPIO_Mode = GPIO_Mode_AF_PP;
      GPIO_InitS.GPIO_Pin = GPIO_Pin_2;
      GPIO_InitS.GPIO_Speed = GPIO_Speed_2MHz;
      GPIO_Init(GPIOA, &GPIO_InitS);
      //PA3 = USART2 RX
      GPIO_InitS.GPIO_Mode = GPIO_Mode_AF_PP;
      GPIO_InitS.GPIO_Pin = GPIO_Pin_3;
      GPIO_Init(GPIOA, &GPIO_InitS);
      /* USART END */

      USART_InitTypeDef USART_InitStruct; // this is for the USART2
initilization
      USART_InitStruct.USART_BaudRate = baud_rate; // the baudrate is set
to the value we passed into this init function
      USART_InitStruct.USART_WordLength = USART_WordLength_8b; // we want
the data frame size to be 8 bits (standard)
      USART_InitStruct.USART_StopBits = USART_StopBits_1;    // we want 1
stop bit (standard)
      USART_InitStruct.USART_Parity = USART_Parity_No;// we don't want a
parity bit (standard)
      USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None; // we don't want flow control (standard)
      USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx; // we
want to enable the transmitter and the receiver
      USART_Init(USART2, &USART_InitStruct); // again all the properties
are passed to the USART_Init function which takes care of all the bit
setting

      /* Here the USART2 receive interrupt is enabled
       * and the interrupt controller is configured
       * to jump to the USART2_IRQHandler() function
```

```
       * if the USART2 receive interrupt occurs
       */
      USART_ITConfig(USART2, USART_IT_RXNE, ENABLE); // enable the USART2
receive interrupt
      NVIC_InitTypeDef NVIC_InitS;
      NVIC_InitS.NVIC_IRQChannel = USART2_IRQn; // we want to configure the
USART2 interrupts
      NVIC_InitS.NVIC_IRQChannelSubPriority = 0; // this sets the
subpriority inside the group
      NVIC_InitS.NVIC_IRQChannelCmd = ENABLE;    // the USART2 interrupts
are globally enabled
      NVIC_Init(&NVIC_InitS);  // the properties are passed to the
NVIC_Init function which takes care of the low level stuff

      // finally this enables the complete USART2 peripheral
      USART_Cmd(USART2, ENABLE);
      usart_counter = 0;
      //USART_ClearITPendingBit(USART2, USART_IT_RXNE);
}

void setup_GPIO(void)  // setup GPIO
{
  // habilitar o clock no barramento das GPIOs B e C
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

  // habilitar o clock do periferico de interrupcao
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

  // setup GPIO B1, B0, B13 - OUTPUT - clock, latch, data
      GPIO_InitTypeDef GPIO_PtB;
  GPIO_PtB.GPIO_Pin = GPIO_Pin_1 + GPIO_Pin_0 + GPIO_Pin_13 +GPIO_Pin_7;
  GPIO_PtB.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_PtB.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(GPIOB, &GPIO_PtB);

  // setup GPIO B6 - interrupção - pushbutton
  GPIO_PtB.GPIO_Pin = GPIO_Pin_6;
  GPIO_PtB.GPIO_Mode = GPIO_Mode_IN_FLOATING; // modo input pino B6
  GPIO_PtB.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(GPIOB, &GPIO_PtB);

  GPIO_InitTypeDef GPIO_PtC;             // estrutura dados config GPIO_C
      // setup dos vals pars da estrutura de dados da GPIO_C e iniciar
      GPIO_PtC.GPIO_Pin = GPIO_Pin_13 + GPIO_Pin_14 ;
      GPIO_PtC.GPIO_Mode = GPIO_Mode_Out_PP;
      GPIO_PtC.GPIO_Speed = GPIO_Speed_2MHz;
      GPIO_Init(GPIOC, &GPIO_PtC);



  // config que GPIOB pino 6 sera usado para gerar EXT INT
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource6);
  EXTI_ClearITPendingBit(EXTI_Line6);

}
```

```
void setup_systick(uint16_t f_tick)  // fn que configura o SysTick c/ 1ms
(freq_systema/frequency
{
  RCC_ClocksTypeDef RCC_Clocks;
  RCC_GetClocksFreq(&RCC_Clocks);
  (void) SysTick_Config(RCC_Clocks.HCLK_Frequency / f_tick);
}

void serialdata (int vector[16]) //função para serializar o vetor data
{
      for (int i=0; i<=16; i++)
      {
            if (i==16) //habilita a saída dos registradores (latch)
            {
                  GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_SET);
                  GPIO_WriteBit(GPIOB, GPIO_Pin_0, Bit_RESET);
            }
            else
            {
                  if (vector[i]==0)       //lê os bits do vetor[]
                  {
                        GPIO_WriteBit(GPIOB, GPIO_Pin_13, Bit_RESET);
                  }
                  else
                  {
                        GPIO_WriteBit(GPIOB, GPIO_Pin_13, Bit_SET);
                  }
            }
                  GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_SET); //clock
                  GPIO_WriteBit(GPIOB, GPIO_Pin_1, Bit_RESET);
      }
}


int main(void) // funcao principal do programa
{
      setup_GPIO();                  // setup GPIOs interface LEDs
      setup_INT_externa();                  // setup Interrupcao externa
      setup_systick(FREQ_TICK);             // set timers p/  1 ms  (1000
Hz)++-/
      setup_USART(9600);



      int data[16]; // {DP,G,F,E,D,C,B,A,X,X,X,X,D1,D2,D3,D4} - {display
(MSB - LSB); espaço inútil; escolha do led}
      data[0]=0;
      data[8]=0;
      data[9]=0;
      data[10]=0;
      data[11]=0;

      int uniseg=0, dezseg=0, unimin=0, decseg=0; //variáveis do cronômetro

      //vetores do decoder
      int decsega[]={0,0,0,0,0,0,0,0};
      int unisega[]={0,0,0,0,0,0,0,0};
```

```
      int dezsega[]={0,0,0,0,0,0,0,0};
      int unimina[]={0,0,0,0,0,0,0,0};

      int flag=0; //flag do multiplex temporal
      int prx_data = (millis() + dt_data); //controla a frequencia do
multiplex temporal
      int prx_dt_toggle = (millis() + dt_toggle);
      int prx_PC13 = (millis() + DT_PC13); // calc prox toggle de PC13
      int prx_PC14 = (millis() + DT_PC14);
      int flag_toggle=0;
      int f13=0, f14=0;

      // entra no loop infinito
      while (1)
      {
            cronometro();
            assemblePayload(Cunimin, Cdezseg,Cuniseg, Cdecseg);

//maquina de estado que controla o cronômetro
      switch (Ler_Modo_Oper()) // dependendo do estado da var Modo_Oper
                  {
                      case 0:
                                  uniseg=Cuniseg;
                                  dezseg=Cdezseg;
                                  unimin=Cunimin;
                                  decseg=Cdecseg;


                                  GPIO_WriteBit(GPIOC, GPIO_Pin_13,
Bit_RESET);
                                  GPIO_WriteBit(GPIOC, GPIO_Pin_14,
Bit_RESET);

                            break;

                      case 1:

                            GPIO_WriteBit(GPIOC, GPIO_Pin_13, Bit_SET);
                            GPIO_WriteBit(GPIOC, GPIO_Pin_14, Bit_SET);

                            USART_SendData(USART2, 0xFF);
                            uniseg=UsartUniSeg;
                            dezseg=UsartDezSeg;
                            unimin=UsartUniMin;
                            decseg=UsartDecSeg;
                            break;

                      case 2:

                            if (millis()>=prx_dt_toggle){
                                  prx_dt_toggle = (millis() +
dt_toggle);
                                  if (flag_toggle==1){
                                  flag_toggle=0;
                                  }
                                  else{
                                  flag_toggle=1;
```

```
                                        }

                                    }

                                                if (flag_toggle==1){
                                                      uniseg=Cuniseg;
                                                      dezseg=Cdezseg;
                                                      unimin=Cunimin;
                                                      decseg=Cdecseg;


                                                      GPIO_WriteBit(GPIOC,
GPIO_Pin_14, Bit_RESET);
                                                      if (millis()>=prx_PC13)
                                                            {
                                                              // calc proximo
tempo para PC13...
                                                                    prx_PC13 =
(millis() + DT_PC13);
                                                                    if(f13==0)
// faz toogle LED PC13 entre ligado e desligado
                                                                    {
                                                                        f13 =
1;
GPIO_WriteBit(GPIOC, GPIO_Pin_13, Bit_RESET);  // apaga o LED PC13
                                                                    } else {
                                                                        f13 =
0;
GPIO_WriteBit(GPIOC, GPIO_Pin_13, Bit_SET); // acende o LED PC13
                                                                    }

                                                              }
                                                }

                                                if (flag_toggle==0){

                                                      USART_SendData(USART2,
0xFF);
                                                      uniseg=UsartUniSeg;
                                                      dezseg=UsartDezSeg;
                                                      unimin=UsartUniMin;
                                                      decseg=UsartDecSeg;

                                                GPIO_WriteBit(GPIOC,
GPIO_Pin_13, Bit_SET);
                                                if (millis()>=prx_PC14)
                                                      {
                                                        // calc proximo tempo
para PC13...
                                                              prx_PC14 =
(millis() + DT_PC14);
                                                              if(f14==0)   //
faz toogle LED PC13 entre ligado e desligado
                                                              {
                                                                  f14 = 1;
```

```
GPIO_WriteBit(GPIOC, GPIO_Pin_14, Bit_SET);  // apaga o LED PC13
                                                                } else {
                                                                    f14 = 0;

GPIO_WriteBit(GPIOC, GPIO_Pin_14, Bit_RESET); // acende o LED PC13
                                                                }

                                                            }
                                                        }



                                    break;

                        }

//---fim da máquia de estado



if (request==1){
      GPIO_WriteBit(GPIOB, GPIO_Pin_7, Bit_SET);
}
else{
      GPIO_WriteBit(GPIOB, GPIO_Pin_7, Bit_RESET);
}




//decoder ------------------------------------------

      switch (decseg) // decoder dos décimos de segundos

                    {

                       case 0:

                              decsega[1]=1;

                              decsega[2]=0;

                              decsega[3]=0;

                              decsega[4]=0;

                              decsega[5]=0;

                              decsega[6]=0;

                              decsega[7]=0;

                       break;
```

```
case 1:
      decsega[1]=1;
      decsega[2]=1;
      decsega[3]=1;
      decsega[4]=1;
      decsega[5]=0;
      decsega[6]=0;
      decsega[7]=1;
break;

case 2:
      decsega[1]=0;
      decsega[2]=1;
      decsega[3]=0;
      decsega[4]=0;
      decsega[5]=1;
      decsega[6]=0;
      decsega[7]=0;
break;

case 3:
      decsega[1]=0;
      decsega[2]=1;
      decsega[3]=1;
      decsega[4]=0;
      decsega[5]=0;
      decsega[6]=0;
      decsega[7]=0;

break;
```

```
case 4:
      decsega[1]=0;
      decsega[2]=0;
      decsega[3]=1;
      decsega[4]=1;
      decsega[5]=0;
      decsega[6]=0;
      decsega[7]=1;

break;

case 5:
      decsega[1]=0;
      decsega[2]=0;
      decsega[3]=1;
      decsega[4]=0;
      decsega[5]=0;
      decsega[6]=1;
      decsega[7]=0;

break;

case 6:
      decsega[1]=0;
      decsega[2]=0;
      decsega[3]=0;
      decsega[4]=0;
      decsega[5]=0;
      decsega[6]=1;
      decsega[7]=0;
```

```
break;

case 7:
      decsega[1]=1;
      decsega[2]=1;
      decsega[3]=1;
      decsega[4]=1;
      decsega[5]=0;
      decsega[6]=0;
      decsega[7]=0;
break;

case 8:
      decsega[1]=0;
      decsega[2]=0;
      decsega[3]=0;
      decsega[4]=0;
      decsega[5]=0;
      decsega[6]=0;
      decsega[7]=0;
break;

case 9:
      decsega[1]=0;
      decsega[2]=0;
      decsega[3]=1;
      decsega[4]=0;
      decsega[5]=0;
      decsega[6]=0;
```

```
                        decsega[7]=0;
                    break;

                }

    switch (uniseg) // decoder da unidade de segundos
                            {
                    case 0:
                        unisega[1]=1;
                        unisega[2]=0;
                        unisega[3]=0;
                        unisega[4]=0;
                        unisega[5]=0;
                        unisega[6]=0;
                        unisega[7]=0;
                    break;

                    case 1:
                        unisega[1]=1;
                        unisega[2]=1;
                        unisega[3]=1;
                        unisega[4]=1;
                        unisega[5]=0;
                        unisega[6]=0;
                        unisega[7]=1;
                    break;

                    case 2:
                        unisega[1]=0;
                        unisega[2]=1;
                        unisega[3]=0;
```

```
                unisega[4]=0;
                unisega[5]=1;
                unisega[6]=0;
                unisega[7]=0;
            break;

            case 3:
                unisega[1]=0;
                unisega[2]=1;
                unisega[3]=1;
                unisega[4]=0;
                unisega[5]=0;
                unisega[6]=0;
                unisega[7]=0;

            break;

            case 4:
                unisega[1]=0;
                unisega[2]=0;
                unisega[3]=1;
                unisega[4]=1;
                unisega[5]=0;
                unisega[6]=0;
                unisega[7]=1;

            break;

            case 5:
                unisega[1]=0;
                unisega[2]=0;
```

```
            unisega[3]=1;
            unisega[4]=0;
            unisega[5]=0;
            unisega[6]=1;
            unisega[7]=0;

        break;

        case 6:
            unisega[1]=0;
            unisega[2]=0;
            unisega[3]=0;
            unisega[4]=0;
            unisega[5]=0;
            unisega[6]=1;
            unisega[7]=0;

        break;

        case 7:
            unisega[1]=1;
            unisega[2]=1;
            unisega[3]=1;
            unisega[4]=1;
            unisega[5]=0;
            unisega[6]=0;
            unisega[7]=0;

        break;

        case 8:
```

```
                unisega[1]=0;
                unisega[2]=0;
                unisega[3]=0;
                unisega[4]=0;
                unisega[5]=0;
                unisega[6]=0;
                unisega[7]=0;

            break;

            case 9:
                unisega[1]=0;
                unisega[2]=0;
                unisega[3]=1;
                unisega[4]=0;
                unisega[5]=0;
                unisega[6]=0;
                unisega[7]=0;

            break;

                }

    switch (dezseg) // decoder da dezena de segundos
            {
                case 0:

                if(unimin==0 && dezseg==0)
                {
                    dezsega[1]=1;
                    dezsega[2]=1;
                    dezsega[3]=1;
```

```
                dezsega[4]=1;
                dezsega[5]=1;
                dezsega[6]=1;
                dezsega[7]=1;
            }
            else {
            dezsega[1]=1;
            dezsega[2]=0;
            dezsega[3]=0;
            dezsega[4]=0;
            dezsega[5]=0;
            dezsega[6]=0;
            dezsega[7]=0;
            }
        break;

        case 1:
            dezsega[1]=1;
            dezsega[2]=1;
            dezsega[3]=1;
            dezsega[4]=1;
            dezsega[5]=0;
            dezsega[6]=0;
            dezsega[7]=1;
        break;

        case 2:
            dezsega[1]=0;
            dezsega[2]=1;
```

```
                dezsega[3]=0;
                dezsega[4]=0;
                dezsega[5]=1;
                dezsega[6]=0;
                dezsega[7]=0;
            break;

            case 3:
                dezsega[1]=0;
                dezsega[2]=1;
                dezsega[3]=1;
                dezsega[4]=0;
                dezsega[5]=0;
                dezsega[6]=0;
                dezsega[7]=0;

            break;

            case 4:
                dezsega[1]=0;
                dezsega[2]=0;
                dezsega[3]=1;
                dezsega[4]=1;
                dezsega[5]=0;
                dezsega[6]=0;
                dezsega[7]=1;

            break;

            case 5:
                dezsega[0]=1;
```

```
                dezsega[1]=0;
                dezsega[2]=0;
                dezsega[3]=1;
                dezsega[4]=0;
                dezsega[5]=0;
                dezsega[6]=1;
                dezsega[7]=0;

            break;

            case 6:
                dezsega[1]=0;
                dezsega[2]=0;
                dezsega[3]=0;
                dezsega[4]=0;
                dezsega[5]=0;
                dezsega[6]=1;
                dezsega[7]=0;

            break;

            case 7:
                dezsega[1]=1;
                dezsega[2]=1;
                dezsega[3]=1;
                dezsega[4]=1;
                dezsega[5]=0;
                dezsega[6]=0;
                dezsega[7]=0;

            break;
```

```
            case 8:
                  dezsega[1]=0;
                  dezsega[2]=0;
                  dezsega[3]=0;
                  dezsega[4]=0;
                  dezsega[5]=0;
                  dezsega[6]=0;
                  dezsega[7]=0;

            break;

            case 9:
                  dezsega[1]=0;
                  dezsega[2]=0;
                  dezsega[3]=1;
                  dezsega[4]=0;
                  dezsega[5]=0;
                  dezsega[6]=0;
                  dezsega[7]=0;

            break;

               }

      switch (unimin)  // decoder para unidade de minutos
               {
                  case 0:
                        unimina[0]=1;
                        unimina[1]=1;
                        unimina[2]=1;
                        unimina[3]=1;
```

```
                unimina[4]=1;
                unimina[5]=1;
                unimina[6]=1;
                unimina[7]=1;
            break;

            case 1:
                unimina[0]=0;
                unimina[1]=1;
                unimina[2]=1;
                unimina[3]=1;
                unimina[4]=1;
                unimina[5]=0;
                unimina[6]=0;
                unimina[7]=1;
            break;

            case 2:
                unimina[0]=0;
                unimina[1]=0;
                unimina[2]=1;
                unimina[3]=0;
                unimina[4]=0;
                unimina[5]=1;
                unimina[6]=0;
                unimina[7]=0;
            break;

            case 3:
                unimina[0]=0;
```

```
            unimina[1]=0;
            unimina[2]=1;
            unimina[3]=1;
            unimina[4]=0;
            unimina[5]=0;
            unimina[6]=0;
            unimina[7]=0;

        break;

        case 4:
            unimina[0]=0;
            unimina[1]=0;
            unimina[2]=0;
            unimina[3]=1;
            unimina[4]=1;
            unimina[5]=0;
            unimina[6]=0;
            unimina[7]=1;

        break;

        case 5:
            unimina[0]=0;
            unimina[1]=0;
            unimina[2]=0;
            unimina[3]=1;
            unimina[4]=0;
            unimina[5]=0;
            unimina[6]=1;
```

```
                unimina[7]=0;

            break;

            case 6:
                unimina[0]=0;
                unimina[1]=0;
                unimina[2]=0;
                unimina[3]=0;
                unimina[4]=0;
                unimina[5]=0;
                unimina[6]=1;
                unimina[7]=0;

            break;

            case 7:
                unimina[0]=0;
                unimina[1]=1;
                unimina[2]=1;
                unimina[3]=1;
                unimina[4]=1;
                unimina[5]=0;
                unimina[6]=0;
                unimina[7]=0;

            break;

            case 8:
                unimina[0]=0;
                unimina[1]=0;
                unimina[2]=0;
```

```
                                    unimina[3]=0;

                                    unimina[4]=0;

                                    unimina[5]=0;

                                    unimina[6]=0;

                                    unimina[7]=0;


                              break;


                              case 9:

                                    unimina[0]=0;

                                    unimina[1]=0;

                                    unimina[2]=0;

                                    unimina[3]=1;

                                    unimina[4]=0;

                                    unimina[5]=0;

                                    unimina[6]=0;

                                    unimina[7]=0;


                              break;


                           }

                                                                        //fim
do decoder
-------------------------------------------------------------------------
-------


                              //multiplex temporal
                                          if (millis()>=prx_data)
                                          {
                                                prx_data = (millis() +
dt_data);

                                                switch (flag)
                                                {
                                                      case 0:
                                                            data[0]=1;

data[1]=decsega[1];
```

```
data[2]=decsega[2];

data[3]=decsega[3];

data[4]=decsega[4];

data[5]=decsega[5];

data[6]=decsega[6];

data[7]=decsega[7];
                                                        data[12]=0;
                                                        data[13]=0;
                                                        data[14]=0;
                                                        data[15]=1;
                                                        flag=1;
                                                  break;

                                                  case 1:
                                                        data[0]=0;

data[1]=unisega[1];

data[2]=unisega[2];

data[3]=unisega[3];

data[4]=unisega[4];

data[5]=unisega[5];

data[6]=unisega[6];

data[7]=unisega[7];
                                                        data[12]=0;
                                                        data[13]=0;
                                                        data[14]=1;
                                                        data[15]=0;
                                                        flag=2;
                                                  break;
                                                  case 2:
                                                        data[0]=1;

data[1]=dezsega[1];

data[2]=dezsega[2];

data[3]=dezsega[3];

data[4]=dezsega[4];

data[5]=dezsega[5];

data[6]=dezsega[6];

data[7]=dezsega[7];
                                                        data[12]=0;
```

```
                                                                    data[13]=1;
                                                                    data[14]=0;
                                                                    data[15]=0;
                                                                    flag=3;
                                                              break;

                                                              case 3:

data[0]=unimina[0];

data[1]=unimina[1];

data[2]=unimina[2];

data[3]=unimina[3];

data[4]=unimina[4];

data[5]=unimina[5];

data[6]=unimina[6];

data[7]=unimina[7];
                                                                    data[12]=1;
                                                                    data[13]=0;
                                                                    data[14]=0;
                                                                    data[15]=0;
                                                                    flag=0;
                                                              break;

                                                              default:
                                                              flag=0;
                                                              break;

                                                        }

                                                        serialdata(data); //código
para serializar o vetor
                                                  }

                                    // fim do multiplex temporal

      } // fim do loop

} // fim da função principal
```

Quadro 2: Código do “stm32f1xx_it.c” onde fora implementado as funções do SysTick, millis, a interrupção com ajuste debouncing e a Ler_Modo_Oper.

```
/* Includes
------------------------------------------------------------------*/
#include "stm32f1xx_it.h"
```

```
#include "stdlib.h"

/** @addtogroup IO_Toggle
  * @{
  */

/* Private typedef
-----------------------------------------------------------*/
/* Private define
-----------------------------------------------------------*/
/* Private macro
-------------------------------------------------------------*/
/* Private variables
---------------------------------------------------------*/
/* Private function prototypes
-----------------------------------------------*/
/* Private functions
---------------------------------------------------------*/

/******************************************************************************
****/
/*            Cortex-M Processor Exceptions Handlers
*/
/******************************************************************************
****/

/**
  * @brief  This function handles NMI exception.
  * @param  None
  * @retval None
  */
void NMI_Handler(void)
{
}

/**
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @retval None
  */
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
```

```
  {
  }
}

/**
  * @brief  This function handles Bus Fault exception.
  * @param  None
  * @retval None
  */
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Usage Fault exception.
  * @param  None
  * @retval None
  */
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{
}

/**
  * @brief  This function handles Debug Monitor exception.
  * @param  None
  * @retval None
  */
void DebugMon_Handler(void)
{
}

/**
  * @brief  This function handles PendSVC exception.
  * @param  None
  * @retval None
  */
void PendSV_Handler(void)
{
}
```

```
// define a var "ticks"  (contador atualizado a cada 1ms)
volatile uint32_t ticks;

// rotina de tratamento da interrupcao (ISR) to timer SysTick
void SysTick_Handler(void)
{
  ticks++;
}

// fn que qpenas retorna o valor de "ticks" quando desejado
uint32_t millis(void)
{
  return ticks;
}

/*******************************************************************************/
/*                 STM32F1xx Peripherals Interrupt Handlers                    */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f10x_md.s).                                             */
/*******************************************************************************/

// ISR rotina de servico de IRQ linha 1  (Joao Ranhel - 09/2017 )

#define delta_t 500
volatile uint32_t Modo_Oper = 0;    // define var Modo_Oper
volatile uint32_t prox_t =delta_t;

void EXTI9_5_IRQHandler(void)
{

      if (EXTI_GetITStatus(EXTI_Line6)!=RESET && millis()>=prox_t)
      {
      prox_t=(delta_t + millis());
                  Modo_Oper++;          // cada vez que atender ISR inc modo operacao
                  if (Modo_Oper > 2)
                  {
                        Modo_Oper = 0;    // se >2 volta modo operacao 0
                  }
      }
      EXTI_ClearITPendingBit(EXTI_Line6);
}

// fn que qpenas retorna o valor de "rdt_A E rdt_B" quando desejado
uint32_t Ler_Modo_Oper(void)
{
  return Modo_Oper;
}
volatile uint8_t usart_buffer[6] = {0, 0, 0, 0, 0, 0};
volatile uint8_t usart_counter = 0;
```

```
#define dt_crono 100
uint32_t prx_valor= 0;
uint32_t Cuniseg=0, Cdezseg=0, Cunimin=0, Cdecseg=0; //variáveis do
cronômetr
uint32_t UsartUniMin = 0, UsartDezSeg = 0, UsartUniSeg = 0, UsartDecSeg =
0;
uint8_t payload[4];
uint8_t request=0;
uint32_t prx_request=0;

#define dt_request 2000
void cronometro(void){

      if (millis()>=prx_valor)
                  {
                        prx_valor = (millis() + dt_crono);
                        Cdecseg++;
                        if(Cdecseg==10)
                        {
                              Cdecseg=0;
                              Cuniseg++;
                              if(Cuniseg==10)
                              {
                                    Cuniseg=0;
                                    Cdezseg++;
                                    if(Cdezseg==6)
                                    {
                                          Cdezseg=0;
                                          Cunimin++;
                                          if(Cunimin==10)
                                          {
                                                Cdezseg=0;
                                                Cunimin=0;
                                          }
                                    }
                              }
                        }
                        if (request==1){
                        USART_SendBytes(payload, 4); // mandando sempre ,
                        }
                  }
      for (int i = 0; i < 5; i++) {
                  uint8_t digito = (usart_buffer[i] & 0xF0) >> 4;
                  uint8_t valor = usart_buffer[i] & 0x0F;

                  for (int i = 0; i < 5; i++){
                              if(usart_buffer[i]==0xFF){
                                    usart_buffer[i]=0x00;
                                    request=1;
                                    prx_request = (millis() + dt_request);
                              }
                  }
                  if(millis()>=prx_request){
                  request = 0;
                  }

                  switch (digito) {
```

```
                  case 1:
                        UsartDecSeg = valor;
                        break;
                  case 2:
                        UsartUniSeg = valor;
                        break;
                  case 3:
                        UsartDezSeg = valor;
                        break;
                  case 4:
                        UsartUniMin = valor;
                        break;
                  default:
                        break;
                  }
            }
}

void USART_SendBytes(uint8_t *data, int count) {
      int i;
      for (i = 0; i < count; i++) {
            USART_SendData(USART2, *(data + i));
            while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
      }
}

void assemblePayload(uint8_t min1_p, uint8_t seg1_p, uint8_t seg2_p,
uint8_t decseg_p) {
      payload[0x0] = (0x1 << 4) + (decseg_p & 0x0F);
      payload[0x1] = (0x2 << 4) + (seg2_p & 0x0F);
      payload[0x2] = (0x3 << 4) + (seg1_p & 0x0F);
      payload[0x3] = (0x4 << 4) + (min1_p & 0x0F);
}


void USART2_IRQHandler(void) {

      // check if the USART2 receive RCC_APB1Periph_USART2interrupt flag
was set
      if (USART_GetITStatus(USART2, USART_IT_RXNE)) {

                  usart_buffer[usart_counter] = USART2->DR;
                  usart_counter = (usart_counter + 1) % 5;

            USART_ClearITPendingBit(USART2, USART_IT_RXNE);
      }
}
```

4. Conclusões:

Com a realização deste projeto foi possível verificar a importância de se utilizar adequadamente um protocolo de comunicação para a realização de troca de dados entre diferentes placas, visto que caso houvesse divergência na forma como cada dispositivo iria enviar os seus dados poderia ocorrer falha na comunicação e os dados seriam corrompidos, assim impossibilitando que os mesmos fossem mostrados no display da placa didática UFABC.

Além disso também foi possível observar que o fato de cada grupo programar a sua placa de forma diferente gerou pequenas falhas, principalmente com relação ao tratamento dos dados externos que deveriam ser mostrados quando as placas estivessem rodando no modo externo e no modo dual, como por exemplo o fato do display de 7 segmentos priorizar o acionamento de determinado dígito em detrimento dos demais, dentre outros. No entanto em relação aos testes executados com a nossa placa tais falhas não ocorreram, sendo que a única alteração que poderia ser feita a fim de otimizar o seu funcionamento seria de alterar o código para que enviasse os dados para a outra placa apenas em períodos pré-programados e não todo o tempo (*full time*), porém tal procedimento não interferiu de forma crucial no correto funcionamento da mesma.

5. Referências:

RANHEL, João. **Sistemas Microprocessados:** Apostila com práticas e foco nos processadores ARM CORTEX. 2017. Disponível em: <https://sites.google.com/view/ufabc-sismicrop-2017-3/lab>. Acesso em: 10 nov. 2017.