# Relatório de comunicação bidirecional do MSP430

Frederico José Dias Möller
Bernardo Lopes Frizero
João Vitor Montessi de Oliveira Amaral
Luiza Bartels de Oliveira

28 Nov 2013

# Sumário

# Lista de Figuras

# Lista de Tabelas

**Resumo**

Nessa etapa da pesquisa, desenvolvemos um algoritmo de comunicação bidirecional entre dois MSPs430 a fim de aplicar esse conhcimento na pesquisa de implementação do protocolo CTCP em uma rede de sensores.

Em primeira instância, invertemos nosso antigo método de comunicação (MSP $\rightarrow$ Computador) e fizemos o computador controlar o acendimento de um led no MSP. Depois realizamso a comunicação bidirecional entre o computador e o microcontrolador. Em seguida implementamos um algoritmo do tipo mestre-escravo em dois MSPs 430 e realizamos uma comunicação monodirecional entre eles para por fim implementarmos a comunicação bidirecional.

Alguns problemas que nas fases anteriores , previíamos que sumiriam com o uso de interrupções por comunicação, como o microcontrolador pular um comando se este vier logo em seguida de outro, acabaram acontecendo, porém não se mostraram um obstáculo para a pesquisa.

# Capítulo 1

# Introdução

Para criar com sucesso uma rede de sensores utilizando microcontroladores é necessário que cada nó da rede receba e transmita dados enquanto executa suas tarefas naturais. Para que isso seja feito com sucesso o uso pleno das funções de interrupção por comunicação devem ser feitas. Até agora, em nossa pesquisa, o MSP enviava dados diretamente ao computador e este não enviada nenhum dado de volta, agora com a comunicação bidirecional, teremos a ferramenta para implementar o sistema de "acks"previstos no protocolo CTCP.

Outra finalidade desse relatório é descrever o processo e os códigos que utilizamos para fazer a comunicação bidirecional, para servir de orientação para pesquisas futuras, de quem quer que seja, que precise de algo do tipo.

## 1.1 As funções de interrupção de comunicação do MSP430

As funções de interrupção, como o nome já diz, interrompem a execução da programação normal do micrcocontrolador, executam um pedaço de código a parte e se for o caso, retornam ao ponto onde a execução normal do programa foi interrompida. Essas funções não são chamadas no escopo do programa principal, mas sim por eventos "externos", como variação de sinal em uma entrada, determinada contagem de um clock, ou, neste caso, quando o MSP430 recebe uma transmissão.

Essa função é importante porque se dependessemos de instruções no programa principal para ler o pino de entrada serial perderíamos muitos pacotes de dados, afinal o programa só leria o pino quando chegasse na instrução que o ordenasse ainda. Mesmo que após cada operação mandassemos o microcontrolador ler tal pino, em metade do tempo perderíamos dados.

A função de interrupção pode ser implementada no msp através do ambiente Grace. Após abrirmos o arquivo main.cfg do projeto usando o code composer studio, devemos ir em **device overview**, clicar no módulo correspondente à comunicação do MSP430 (figura 1.1), caso este ainda não tenha sido habilitado ele deve ser como visto na figura 1.2 e em seguida, após selecionarmos **basic user** devemos escolher a opção UART (figura 1.3). Na parte inferior das opções de **basic user** podemos habilitar as funções de interrupção de transmissão e recepção. Nesse relatório, somente a interrupção por recepção é interessante e somente ela deve ser habilitada (figura 1.4). Um nome deve ser dado à função e é possível escolher o modo de operação após a função ser chamada, no entanto, no nosso caso, preferimos deixar o microcontrolador continuar no regime normal.

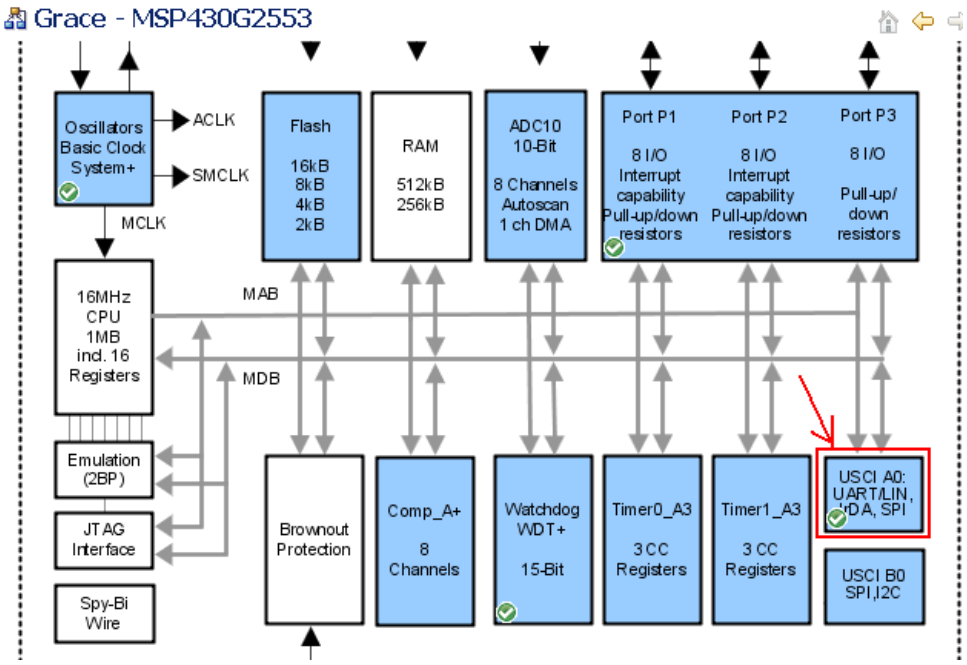


Figura 1.1: Visão geral dos dispositivos do MSP

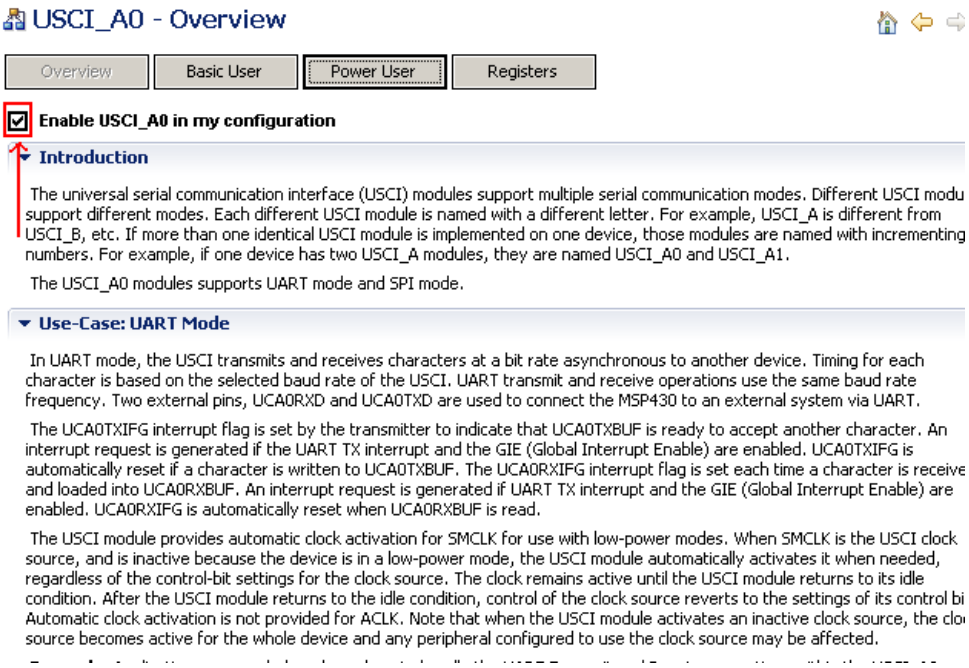


Figura 1.2: Habilitação do módulo de comunicação serial

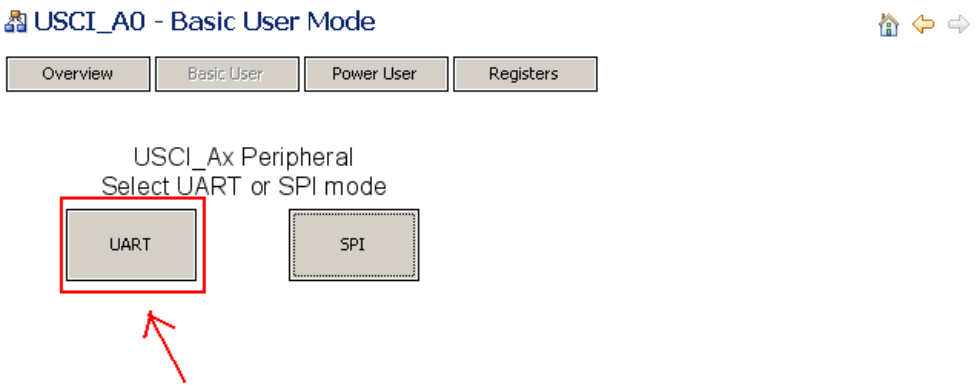


Figura 1.3: Escolha do modo UART

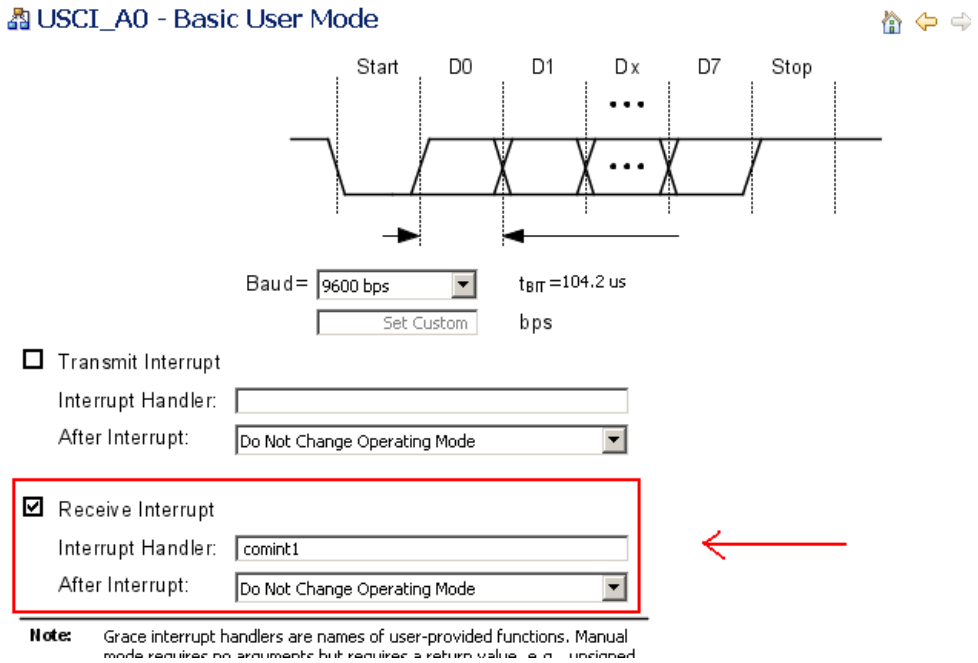


Figura 1.4: Habilitação da função de interrupção por recepção

## 1.2 A configuração utilizada nos módulos xBee

Todos os xBees utilizados seguiram a seguinte configuração:

| Atributo | Valor |
|---|---|
| **DH** | 0 |
| **DL** | 1 |
| **MY** | 1 |
| **BD** | 3 |

Tabela 1.1: Valores dos atributos de configuração dos módulos xBee

Todos os xBee receberam o mesmo ID e o restante dos atributos segue a configuração padrão do xBee 24.

## 1.3 A configuração do Hyperterminal

O hyperterminal foi configurado, nas vezes em que foi utilizado da seguinte forma:

| Atributo | Valor |
|---|---|
| **Bits por segundo** | 9600 |
| **Bits de dados** | 8 |
| **Paridade** | Nenhum |
| **Bits de parada** | 1 |
| **Controle de fluxo** | Nenhum |

Tabela 1.2: Valores dos atributos de configuração do hyperterminal

A conexão foi estabelecida pela porta COM9, que foi a porta atribuida pelo computador para o nosso xBee. Cada máquina pode estabelecer uma porta diferente para o xBee e pode ser que uma máquina atribua portas diferentes ao xBee dependendo do uso do mesmo. Portanto a porta COM9 não é uma regra para o xBee.

Além desses atributos é necessário ir em propriedades, configurações e habilitar as devidas opções em configurações de ASCII, conforme mostrado na figura 1.5.

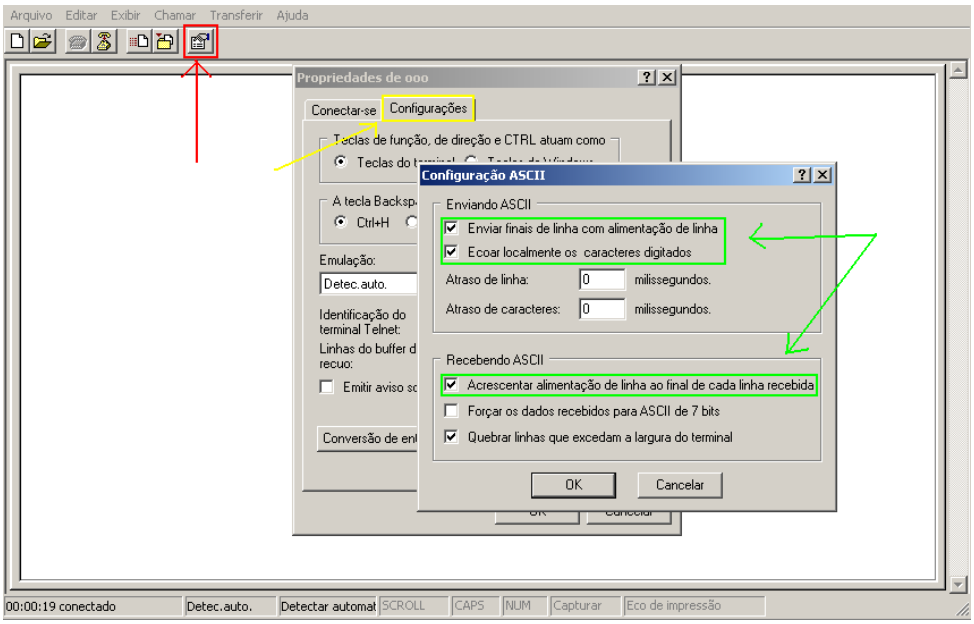


Figura 1.5: Configurações de ASCII do hyperterminal

# Capítulo 2

# Desenvolvimento da comunicação em via dupla

Para o desenvolvimento da comunicação bidirecional, iniciamos com uma comunicação monodirecional, do computador para o MSP, a fim de testar a função de interrupção deste. Em seguida implementamos uma comunicação bidirecional do MSP com o computador, para testar os fundamentos da comunicação bidirecional, após isso, implementamos uma comunicação monodirecional entre dois MSPs para "visualizarmos"uma maneira de testarmos a etapa final, que foi a comunicação bidirecional entre dos MSPs.

A pinagem, para todos esses programas, foi configurada da mesma forma:

| Pino | Valor |
|---|---|
| **P1.0** | OUTPUT |
| **P1.1** | Pino de recepção* |
| **P1.2** | Pino de transmissão* |
| **P1.5** | OUTPUT |
| **P1.6** | OUTPUT |
| **Demais** | Configurações padrão |

Tabela 2.1: Pinagem do MSP430

Obs(*): O grace já seta esses pinos para tais funções assim que você habilita o modo UART.
, E os MSPs configurados da seguinte forma:

| Atributo | Valor |
|---|---|
| **Tensão** | 3,6V |
| **Watchdog Timer** | Desabilitado |
| **Frequência do clock** | 1MHZ |

Tabela 2.2: Configuração dos MSPs

## 2.1 Comunicação monodirecional computador → MSP

Nessa etapa, criamos um programa para o MSP430 no qual, ao enviarmos os caracteres 'w', 's', ou 'd' o microcontrolador alternaria o sinal de saída dos pinos 0, 5, ou 6, aos quais estavam ligados, em cada um, um led.

### 2.1.1 Configuração do MSP430

As configurações de comunicação seguiram como mostrado na introdução. A pinagem foi configurada de acordo com a tabela 2.1.

### 2.1.2 Código utilizado

```
/*
 * base2msp
 * Autor: Frederico José Dias Möller
 * PET Mecatrônica/BSI
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>
#include <string.h>
#include <stdlib.h>
/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>
/*
 *  ======== main ========
 */
int main(void)
{
CSL_init(); // Activate Grace-generated configuration
__bis_SR_register(LPM0_bits + GIE);
return (0);
}
void comint1(void){
if(UCA0RXBUF==’w’){
P1OUT ^= BIT0;
}
else if(UCA0RXBUF==’s’){
P1OUT ^=BIT6;
}
else if(UCA0RXBUF==’d’){
P1OUT ^=BIT5;
}
}
```

Os 3 primeiros comandos se referem à inclusão de bibliotecas utilizadas no código. Em seguida vem o comando de inclusão da biblioteca do grace. A função main é iniciada com o comando que configura o dispositivo com base no que foi feito na intergace grace, em seguida vem o comando que deveria dizer que a partir daquele ponto interrupções por recepção poderiam acontecer. Por fim o comando "return"indica o fim do programa principal.

A parte interessante aqui é a função comint1, que foi nomeada no ambiente grace e é a função de interrupção por recepção. Toda vez que o MSP com esse código recebe um pacote ele para executa essa função, dentro dela existe um conjunto de if-else que lê o buffer de recepção (UCA0RXBUF) e o compara com caracteres pré-determinados. Se algum deles for o 'w', o 's', ou o 'd' ele vai alternar o valor digital de uma das portas de saída (0,6,ou 5).

### 2.1.3 Observações importantes

A biblioteca string está inclusa aqui por mera facilidade ela não tem utilidade nesse algoritmo. Ela aparece porque usamos o nosso programa de transmissão de série numérica do MSP para o computador como base

para esse e mantivemos ela porque os programas posteriores foram escritos com base nesse e eles precisariam da string.

Apesar de não haver nenhum programa principal rodando e execução da alternância dos leds ser feita por interrupção, o microcontrolador não se comportou bem, ignorando alguns comandos quando estes foram digitados à uma taxa próxima a 3 comandos por segundo.

Não é necessário ler o buffer de recepção na função de interrupção por recepção. Utilizamos porque não só o fato de estar recebendo algo, mas o conteúdo da menssagem também era importante. Poderíamos ter criado uma função separada, para ler o buffer e tomar as decisões e ter chamado ela dentro da função de interrupção.

## 2.2 Comunicação bidirecional computador $\leftrightarrow$ MSP

Nessa seção, programamos o microcontrolador para inicialmente transmitir o valor de um número inteiro "i"que era acrescido de uma unidade ao fim de cada ciclo do programa principal. No entanto o usuário poderia escolher visualizar uma sequência cíclica de dígitos de 0 a 9, de letras minúsculas, ou letras maiúsculas. Para fazer isso, o usuário deve enviar um comando pelo hyperterminal. Dessa forma, nesse programa nós temos o microcontrolador enviando e recebendo menssagens e temos também a interrupção de um programa principal para a execução de uma rotinha pré-estabelecida.

### 2.2.1 Configuração do MSP430

As configurações de comunicação seguiram como mostrado na introdução. A pinagem foi configurada de acordo com a tabela 2.1, com excessão dos pinos 1.0,1.5 e 1.6 que foram mandados para INPUT, já que não seriam utilizados nesse programa.

### 2.2.2 Código utilizado

```
/*
 * 2comp2msp
 * AUTOR: Frederico José Dias Möller
 * PET Mecatrônica/BSI
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>
#include <string.h>
#include <stdlib.h>

/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>

/*
 *  ======== main ========
 */
char canal = 0;
int base = 0;

char* itoa(int n, char str[])
{
  int i = 0;               /* Contador do loop */
  int negative = 0;        /* Indicador de negativo*/
  int length = 0;          /* Tamanho da string*/
  int temp = 0;            /* Armazenamento temporário*/
```

```
  if(negative = (n<0))     /*Testa se negativo*/
    n = -n;                /*Faz positivo*/
  /*Gera os dígitos na ordem inversa*/
  do
  {
    str[i++] = '0'+n%10;    /*Pega o digito mais a direita*/
    n /= 10;                /*Remove*/
  }while(n>0);              /*Continua*/
  if(negative)              /*É nagativo?*/
    str[i++] = '-';         /*Coloca o menos*/
  str[i] = '\0';            /*Coloca o terminador*/
  length = i;               /*salva o tamanho*/
  for(i = 0 ; i<length/2 ;i++)
  {
    temp = str[i];
    str[i] = str[length-i-1];
    str[length-i-1] = temp;
  }
  return str;
}

void println(char *line){
long int a=0;
int i;
while(a<2788){
a++;
}
for(i=0;i<strlen(line);i++){
while (!(IFG2 & UCA0TXIFG));
UCA0TXBUF=line[i];
}
while (!(IFG2 & UCA0TXIFG));
UCA0TXBUF=13;
}

int main(void)
{
    CSL_init();                     // Activate Grace-generated configuration
    char buffer[100], transmit;
    // >>>>> Fill-in user code here <<<<<
while(1){
long int a=0;
if(canal==0){
itoa(base,buffer);
println(buffer);
}
else if(canal==1){
transmit = (base%10)+48;
while (!(IFG2 & UCA0TXIFG));
UCA0TXBUF = transmit;
}
else if(canal==2){
transmit = (base%26)+65;
while (!(IFG2 & UCA0TXIFG));
```

```
UCA0TXBUF = transmit;
}
else if(canal==3){
transmit = (base%26)+97;
while (!(IFG2 & UCA0TXIFG));
UCA0TXBUF = transmit;
}
while(a<2788){
a++;
}
base++;
}
    return (0);
}
void comint1(void){
if(UCA0RXBUF==’a’){
canal = 0;
}
else if(UCA0RXBUF==’b’){
canal = 1;
}
else if(UCA0RXBUF==’c’){
canal = 2;
}
else if(UCA0RXBUF==’d’){
canal = 3;
}
}
```

No código segue inicialmente a inclusão das biblitecas utilizadas e da biblioteca do grace, em seguida as variáveis **canal** e **base** são declaradas. Em seguida as funções **itoa** e **println** são descritas.

A função principal do programa, começa com as configurações do grace sendo carregadas e depois com as variáveis **buffer** e **transmit** são declaradas. Em seguida o programa entra em um loop sem fim, a cada começo desse ciclo uma variável inteira **a** é declarada e tem valor zero atribuido, então o programa verifica qual o valor da variável **canal** e de acordo com esse o programa vai enviar transmitir o valor integro da variável **base**, convertendo-a para string pela função itoa e depois transmitindo os caracteres pela função println, o programa pode converter o valor da variável **base** para uma posição de um ciclo de dígitos de 0 a 9, ou de caracteres de A à Z, ou a à z da tabela ASCII e grava-los no buffer de transmissão UCA0TXIFG. Terminado isso o microcontrolador entra em uma espera forçada e então adiciona uma unidade à variável **base**.

Por fim a função de interrupção por recepção é descrita, de acordo com o caracter recebido ela altera o valor da variável **canal**

### 2.2.3 Observações importantes

As variáveis **canal** e **base** são declaradas fora da função main pois são variáveis globais ao programa. A variável canal utilizada dentro da função main é a mesma variável utilizada na função comint1, se fosse tal variável fosse declarada apenas na função main, as alterações que ela sofresse em comint1 não afetaria os eventos em main. A variável **base** até poderia ser declarada dentro de main, desde que fora do loop, pois essa variável só é utilizada nessa função, mas em hipótese nenhuma **canal** poderia ser declarada dentro de qualquer função.

Apesar de ter sido descrito na documentação como um habilitador da interrupção, o comando:

```
__bis_SR_register(LPM0_bits + GIE);
```

Não precisou ser utilizado. Na verdade a inclusão deste, dentro do loop sem fim do programa, fez com que esse não funcionasse devidamente, fazendo com que nenhum dado após esse comando fosse transmitido.

O comando:

```
while (!(IFG2 & UCA0TXIFG));
```

Como descrito em uma publicação dessa pesquisa, realizada na 4ª SECAI, serve para esperar que o buffer de transmissão esteja limpo, antes de executar a próxima tarefa.

## 2.3 Comunicação monodirecional MSP → MSP

O objetivo dessa etapa foi compreender a comunicação entre dois microcontroladore e criar um modelo ao qual pudessemos verificar o bom andamento dessa comunicação sem usar um computador.

Foram criados dois programas, um mestre e um escravo. O MSP mestre alternava o estado de seus leds (acendia o vermelho, apagava, acendia o verde, apagava) e então enviava um sinal para o MSP escravo, que cumpria o mesmo ciclo de atividades com a diferença que, enquanto o mestre alternava os leds por um ciclo de tempo, o escravo alternava somente quando recebesse uma ordem do mestre.

### 2.3.1 Configuração do MSP430

A mesma configuração de comunicação descrita anteriormente e a mesma pinagem da tabela 2.1 com excessão do pino 1.5 que não é utilizado e por razões de segurança deve estar como INPUT.

O MSP mestre deve estar com a interrupção por comunicação **desabilitada**.

### 2.3.2 Código utilizado

Para o mestre:

```
/*
 * msp2msp1master
 * AUTOR: Frederico José Dias Möller
 * PET Mecatrônica/BSI
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>
#include <string.h>
#include <stdlib.h>
/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>
/*
 *  ======== main ========
 */
int main(void)
{
    CSL_init();                     // Activate Grace-generated configuration
    while(1){
long int a=0;
P1OUT ^= BIT0;
while(a<45000){
a++;
}
P1OUT ^= BIT0;
a=0;
```

```
while(a<45000){
a++;
}
P1OUT ^= BIT6;
a=0;
while(a<45000){
a++;
}
P1OUT ^= BIT6;
a=0;
while(a<45000){
a++;
}
while (!(IFG2 & UCA0TXIFG));
UCA0TXBUF='a';
}
return (0);
}
```

Um código bem simples, que começa com a inclusão de bibliotecas gerais e da bilioteca do grace e que não tem nenhuma função descrita, apenas a função principal. Nela, após o comando de configuração do dispositivos pelo grace, é iniciado um ciclo sem fim, dentro desse ciclo, no ínicio, a variável **??** é declarada e tem o valor 0 atribuido. Em seguida temos uma sequência de alternância de estados das portas 1.0 e 1.6, intercaladas com esperas forçadas. Ao fim do circlo o MSP escreve o caracter 'a' no buffer de transmissão (não confundir com a variável **a**), que não tem nada haver com esse caracter.

O Código utilizado no escrevo foi:

```
/*
 * msp2msp1slave
 * AUTOR: Frederico José Dias Möller
 * PET Mecatrônica/BSI
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>
#include <string.h>
#include <stdlib.h>
/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>
/*
 *  ======== main ========
 */
char contador = 1;
int main(void)
{
CSL_init();                     // Activate Grace-generated configuration
// >>>>> Fill-in user code here <<<<<
while(1){
while(1){
if(contador==0){
break;
}
}
P1OUT ^= BIT0;
```

```
contador=1;
while(1){
if(contador==0){
break;
}
}
contador=1;
P1OUT ^= BIT0;
while(1){
if(contador==0){
break;
}
}
contador=1;
P1OUT ^= BIT6;
while(1){
if(contador==0){
break;
}
}
contador=1;
P1OUT ^= BIT6;
}
return (0);
}
void comint1(void){
if(UCA0RXBUF==’a’){
contador=0;
}
}
```

O código inicial com a inclusão das bibliotecas gerais e a biblioteca do grace. Uma variável global **contador** é declarada e tem valor 1 atribuido em seguida começa a função principal do programa, nela as configurações do grace são carregadas e então começa um ciclo sem fim. No ciclo temos o esquema de alternância de valores lógicos nos pinos 1.0 e 1.6 tal qual no programa mestre, no entanto, ao invés de termos a espera forçada de 45000 ciclos entre eles, temos ciclos sem fim que serão quebrados somente se a variável contador tiver valor 0. Após a quebra de cada um desses ciclos, o programa executa a alternância do valor lógico de um dos pinos, atribui o valor lógico 1 à variável **contador** e volta a entrar em um ciclo sem fim.

A atribuição do valor 0 à variável **contador** ocorre na função de interrupção por comunicação, quando é recebido um caracter ’a’.

### 2.3.3 Observações importantes

A espera forçada, é um comando burro, ele só foi utilizado por ser mais simples de implementar e porque o único objetivo desses programas é visualizar e entender o funcionamento da comunicação entre dois microcontroladores.

No código escravo, ao invés de escrever o código:

```
while(1){
if(contador==0){
break;
}
}
```

Poderia ter escrito:

```
while(contador);
```

De fato foi o que eu fiz e isso era para funcionar (já tendo fucionado em outras ocasiões inclusive), no entanto dessa vez o microcontrolador passou a simplesmente pular esses ciclos. Presumo que seja um erro particular do arquivo no qual está contido o código e que isso não deve acontecer se esse código for re-escrito em outro arquivo.

Por fim, não era necessário o if-else na função de interrupção, poderíamos ter feito o código mandando zerar a variável contador toda vez que uma menssagem fosse recebida, mas para evitar erros de interferência preferimos ter certeza de que era o MSP mestre que estava mandando o a menssagem e por isso a leitura do buffer de recebimento e o teste if-else.

## 2.4 Comunicação bidirecional MSP → MSP

Neste programa cada MSP piscava continuamente um de seus LEDs, inicialmente ambos piscavam o led vermelho (associado ao pino 1.0). Cada MSP contava uma quantidade de piscadas, quando atingia essa quantidade ele enviava uma menssagem para o outro MSP. Toda vez que um MSP recebia uma menssagem ele alternava o led, se estivesse piscando o led vermelho, passaria a piscar o led verde associado ao pino 1.6. Além disso a frequência de piscada de cada MSP era diferente.

### 2.4.1 Configuração do MSP430

A mesma configuração de comunicação descrita no início desse relatório e a mesma pinagem da tabela t1, com o pino 1.5 como INPUT.

### 2.4.2 Código utilizado

```
/*
 * ======== Standard MSP430 includes ========
 */
#include <msp430.h>
#include <string.h>
#include <stdlib.h>
/*
 * ======== Grace related includes ========
 */
#include <ti/mcu/msp430/csl/CSL.h>
/*
 *  ======== main ========
 */
char verde = 0;
int main(void)
{
CSL_init();                     // Activate Grace-generated configuration
// >>>>> Fill-in user code here <<<<<
long int a =0;
int cont = 0;
while(1){
while(a<60000){
a++;
}
if(!verde){
P1OUT ^= BIT0;
}
else{
```

```
P1OUT ^= BIT6;
}
a=0;
cont++;
if(cont>=6){
cont=0;
while (!(IFG2 & UCA0TXIFG));
UCA0TXBUF='a';
}
}
return (0);
}
void comint1(void){
if(UCA0RXBUF=='b'){
verde^=1;
}
}
```

O código inicia com a inclusão das bibliotecas gerais e da biblioteca do grace, segue a declaração da variável **verde**, que diz se é ou não hora de piscar o led de tal cor.

Na função principal, temos o carregamento das configurações do grace, declaração das variáveis **a** e **cont** e em seguida um ciclo sem fim, dentro dele temos uma espera forçada, seguida de um teste if-else, que vai determinar qual pino e por consequência, qual led deve ter seu estado alternado, depois a variável **cont** tem seu valor acrescido de uma unidade e por fim um outro teste, depois de uma determinada contagem de alternância de estados, a variável **cont** vai à zero, espera-se o buffer de transmissão estar limpo e então se transmite um caracter.

Após a função principal temos a função de interrupção de comunicação. Nela temos um teste if-else que, ao receber um determinado caracter, alterna o estado da variável **verde**

### 2.4.3 Observações importantes

Cada MSP possuia uma quantidade de ciclos diferente na espera forçada. No código é mostrado o MSP que tinha uma espera de 60000 ciclos, o outro tinha uma espera de 45000 ciclos.

Outra diferença é na contagem antes de transmitir o caracter, no código mostrado temos uma contagem de 6, que equivale a 3 piscadas de led. O outro MSP tinha uma contagem de 10, ou seja, 5 piscadas.

Devido a essa proposital diferença de ciclos e contagens, acontecia as vezes de um MSP enviar o caracter enquanto o outro ainda não havia apagado o LED, com isso o MSP preservava o led aceso e ia alternando o estado dos outro led, isso não é um erro e em nada interfere na pesquisa.

Por último, para garântir que é um MSP falando com outro, fizemos com que um MSP transmitisse o caracter 'a' e só agisse se recebesse o caracter 'b' (mostrado no código), já o outro MSP fazia o contrário, recebia 'a' e transmitia 'b'

# Capítulo 3

# Análise dos resultados

Durante o processo de desenvolvimento foi verificado se houve ocorrência de atrasos perceptíveis e de comandos transmitidos e ignorados. Também foi analizado a possibilidade de todos os programas na rede de sensores serem exatamente iguais.

Foi observado que quando vários comandos seguidos eram enviados com um intervalo de tempo curto entre eles (cerca de 3 comandos por segundo), alguns desses comandos eram perdidos.

Nenhum atraso relevante foi observado entre o envio e o cumprimento da tarefa. O atraso verificado era o do tempo do microcontrolador receber o comando, ler o buffer de recebimento e executar as instruções correspondentes.

Na maioria dos programas, o MSP executava um programa diferente do seu interlocutor, no entanto, na etapa final, os dois MSPs executaram o mesmo programa, apenas com alguns valores de variáveis diferentes.

## 3.1 Utilidade no projeto de pesquisa de implemetação do protocolo CTCP em uma rede de sensores

O fato de alguns comandos serem perdidos não constitui impecílio para a implementação do CTCP, na verdade esse "problema"acaba sendo até interessante para a pesquisa. Essa perda deve ocorrer pelo fato do buffer do xBee e do MSP terem pouca capacidade, assim ficam cheios logo e com isso existe uma perda de pacotes. Uma vez que o objetivo do CTCP é reduzir essas perdas em uma rede, esse "problema"pode ser explorado no resultado final.

A forma que o programa que rodará nos MSPs na fase de implementação do CTCP dependerá muito mais do resultado da comunicação multidirecional de 3 MSPs do que dessa fase, no entanto a possibilidade de todos os MSPs rodarem o mesmo programa em uma rede se mostrou viável.

## 3.2 Proposta para a próxima fase

A próxima etapa é começar a implementar a comunicação multidirecional em 3 nós. Seria interessante começar a próxima etapa usando dois MSPs com o programa 2msp2msp e um computador. Outros pontos que devem ser explorados é os MSPs usarem o mesmo caracter para transmitir e o mesmo para receber.

# Capítulo 4

# Conclusão

A comunicação bidirecional entre dois MSPs foi feita com sucesso, os erros apresentados no processo não compromentem a pesquisa. A comunicação entre três MSPs pode começar a ser implementada.