Experimento 6 - Comunicação *Bluetooth*

EA076 - Laboratório de Sistemas Embarcados

Turma C - Grupo 6

Gustavo Ciotto Pinton RA 117136

Anderson Une Bastos RA 093392

Campinas, 16 de Maio de 2016

Introdução

O objetivo principal deste experimento foi a integração do módulo *bluetooth* à placa de desenvolvimento. A fim de integrá-lo, foi utilizado o método de comunicação serial assíncrona, que baseia-se em dois sinais, chamados de Rx e Tx. Tais sinais são frequentemente encontrados também em outras placas de desenvolvimento, como o *Arduino*, *Galileo* e *Edison*, e são usados, assim como neste experimento, para transferência de dados e comandos.

O *Bluetooth* é um padrão global de comunicação sem fio e de baixo consumo de energia que permite a troca de informações entre dispositivos, desde que um esteja próximo do outro. A transmissão de dados ocorre por meio de radiofrequência, permitindo que um dispositivo detecte o outro independente de suas posições, sendo necessário apenas que ambos estejam dentro do limite de proximidade.

Dos três métodos de comunicação suportados pela *MCU* - *I2C* (*Inter-Integrated Circuit*), *UART* (*Universal Asynchronous Receiver/Transmitter*) e *SPI* (S*erial Peripheral Interface*) -, aquele que foi escolhido para este experimento foi a comunicação serial assíncrona com os sinais Rx/Tx para a transferência de dados e comandos. O módulo *bluetooth* disponível no almoxarifado pertence à família *HC-06*, ou seja, uma vez configurado seu modo de operação, ele não pode ser modificado. No nosso caso, o módulo foi configurado para o modo escravo portanto, nenhum comando de escolha de modo de operação foi necessária.

Implementação das conexões e componentes **(itens 1 a 4)**

A referência [1] mostra os diversos tipos de módulos *bluetooths* encontrados no mercado, sendo comum a montagem em uma pequena placa com LED de status, regulador de tensão e 4 pinos de conexão [3]. Nos módulos disponíveis no almoxarifado, as ligações destes pinos foram feitas utilizando um conector DB-9, também mostrado na referência [3]. A figura 1, abaixo, representa tais ligações. Estes módulos estabelecem uma conexão do tipo Porta COM virtual com outro dispositivo *bluetooth* (*smartphone, tablet etc*.) e uma conexão serial física (Tx/Rx) com um microcontrolador, através dos pinos de conexão. O roteiro sugere estabelecer uma nomenclatura padronizada, chamando a conexão do módulo com o outro dispositivo *bluetooth* (“sem fio”) de conexão *bluetooth*, e a conexão do módulo com a MCU (“com fio”) de conexão UART.

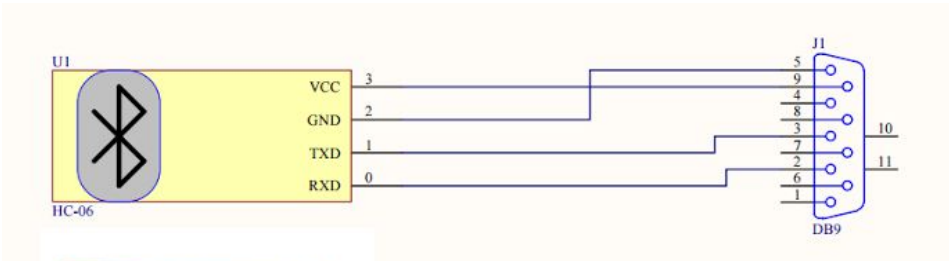

Figura 1: Conexão dos sinais de controle do *bluetooth* ao conector DB-9.

A figura 2, por sua vez, representa as ligações dos pinos do microcontrolador ao conector DB9. Observa-se na tabela da seção 10.3.1 de [4] que o pino PTE0 pode assumir a função de TX, enquanto que PTE1, a função RX. Tais pinos devem ser ligados, respectivamente, às entradas RX e TX do módulo *bluetooth*.

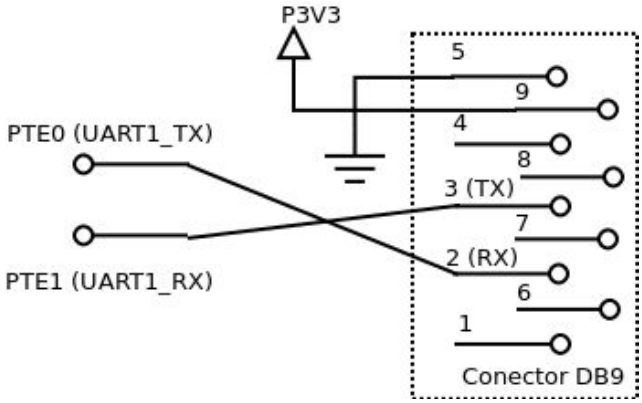

Figura 2: Conexão dos sinais de controle do *microcontrolador* ao conector DB-9.

O componente *AsynchroSerial* é um *driver* que permite o envio serial de dados. Ele permite, além da configuração dos pinos que serão utilizados, a modificação de alguns atributos como o *baudrate* e o número de *start e stop bits*. No nosso caso, configuraremos o *baudrate* para 9600 bps, 1 *stop bit* e nenhum *start bit*. Este componente fornece duas funções para o envio e recebimento de *bytes*, sendo elas, respectivamente, `SendChar()` e `RecvChar()`, e duas funções de tratamento de interrupção. A primeira, `OnRxChar`, é ativada quando a UART recebe um caracter e a segunda, `OnTxChar`, quando um *byte* foi corretamente enviado.

Analisando as referências [1] e [2] podemos verificar os comandos básicos para controlar o módulo *bluetooth*. A sintaxe geral destes comandos se dá pela seguinte sequência de caracteres: “AT” para a verificação de comunicação e “AT+*command*”, em que “*command*” é o comando desejado. Por exemplo, inicialmente mandamos o comando “AT” para o módulo *bluetooth* e ele nos fornece a sequência “OK”, indicando que a comunicação foi estabelecida. Em seguida, podemos trocar o nome do módulo através do comando “AT+NAME*name*”, em que “*name*”, no nosso caso, vale “C6” (turma da disciplina e número do grupo, conforme sugerido pelo roteiro). O *baud rate* também pode ser modificado através do comando “AT+BAUD#”, em que “#” é o número que representa o *baud rate* desejado (fornecido na tabela em [2]). Para os propósitos deste experimento foi utilizado o comando “AT+BAUD4” (*baud rate* correspondente ao valor 9600).

Testando a comunicação bluetooth **(item 5)**

As funções de tratamento das interrupções geradas pelo microcontrolador ao receber e ao enviar *bytes* estão representadas nos programas abaixo.

Programa 1: função de tratamento de interrupção que é chamada quando a MCU recebe um *byte*.

```
void Bluetooth_OnRxChar(void) {
    hasReceivedChar = 1;
}
```

Quando um novo *byte* é recebido, a única ação é atualizar uma *flag*, responsável por sinalizar ao programa principal que um novo *byte* pode ser lido. Cabe ao programa principal, lê-lo efetivamente.

Programa 2: função de tratamento de interrupção que é chamada quando a MCU envia um *byte*.

```
void Bluetooth_OnTxChar(void) {
    isReadyToSend = 1;
}
```

Da mesma maneira que no caso anterior, a função de tratamento de interrupção somente atualiza o estado de uma nova *flag,* chamada de `isReadyToSend`. Tal *flag* indica que a UART está pronta para enviar um novo *byte*.

A função `send_string_bluetooth`, contida no trecho abaixo, recebe uma *string* como parâmetro e envia cada um de seus caracteres. A cada envio, a função aguarda que o *flag* `isReadyToSend` seja atualizado novamente para 1 a fim enviar o próximo *byte*.

Programa 3: função que envia uma sequência de *bytes* ao módulo *bluetooth.*

```
int send_string_bluetooth(char* name) {

     int result, i;

     for (i = 0; i < strlen(name); i++) {
         /* Espera interr. indicando que a UART esta pronta para
            enviar novo byte */
          while(!isReadyToSend);

          isReadyToSend = 0;
          result = Bluetooth_SendChar(name[i]);

          if (result != ERR_OK)
               return result;
     }
     return ERR_OK;
}
```

Enfim, para alterar o nome do módulo *bluetooth*, utiliza-se a função `change_bluetooth_name`, representada abaixo. Ela recebe o novo nome como parâmetro e realiza as operações necessárias para que o módulo identifique corretamente tal comando. Por fim, ela espera a resposta `OKsetname` do componente, conforme [2]. Para realizar a mudança de nome, a função `change_bluetooth_name` concatena o comando AT+NAME com o nome desejado e envia para o módulo *bluetooth* através da função `send_string_bluetooth`, explicada anteriormente.

Programa 5: função que envia um comando de mudança de nome ao módulo *bluetooth.*

```
int change_bluetooth_name(char *name) {

     char command[255], response[255];

     memset(command, 0, 255);
     memset(response, 0, 255);
     /* Comando segundo referencia [2] */
     strcpy(command, "AT+NAME");

     strcat(command, name);
     /* Envia comando + novo nome */
     int result = send_string_bluetooth(command);
     if (result == ERR_OK) {

          /* Espera resposta. */
          int tam = 0;
          while (strcmp(response, "OKsetname")) {
```

```
                char rcv_byte;
                if (hasReceivedChar) {
                     Bluetooth_RecvChar(&rcv_byte);

                     hasReceivedChar = 0;

                     response[tam++] = rcv_byte;
                     send_data(rcv_byte);
                }
           }
     }

}
```

Comunicação entre dispositivo *bluetooth* e MCU (item 6)

Nesta seção, a *MCU* foi programada de modo a ecoar todos os dados recebidos através da conexão *bluetooth*. Para tal, ela utiliza a função `read_bluetooth` que verifica a *flag* `hasReceivedChar` e, caso ela seja 1, isto é, a UART recebeu de fato um *byte,* lê o respectivo caracter (através da função `RecvChar`) e o retorna ao programa principal. Caso contrário, ela retorna 0, indicando que nenhum caracter foi lido.

Programa 6: função que lê um caracter da UART.

```
char read_bluetooth() {

     char c = 0;
     /* Verifica se UART recebeu novo caracter */
     if (hasReceivedChar) {
          Bluetooth_RecvChar(&c);
          hasReceivedChar = 0;
     }
     return c;
}
```

O programa principal, por sua vez, está representado abaixo. Caso o retorno de `read_bluetooth` não seja nulo, o programa aguarda que a *flag* `isReadyToSend` seja atualizado e, em seguida, envia o caracter ao módulo *bluetooth* por meio da função `SendChar`, implementada no *driver AsychroSerial*.

Programa 7: trecho da função *main* responsável por 'ecoar' os *bytes* recebidos.

```
int main() {
     (...)
     for (;;) {

      /* Verfica se ha caracter a ser recebido */
      if (hasReceivedChar) {
           char rcv_byte;
           Bluetooth_RecvChar(&rcv_byte);

           send_data(rcv_byte); /* Escreve no LCD */
```

```
            hasReceivedChar = 0;

           /* Espera flag ser atualizado */
            while(!isReadyToSend);
            isReadyToSend = 0;
            Bluetooth_SendChar((byte) rcv_byte);

       }
     }
}
```

Complemento do programa do roteiro 4 (item 7)

Nesta última seção, extendemos a aplicação implementada no relatório 4 para enviar ao módulo *bluetooth* as amostras presentes na memória EEPROM. Foram adicionadas, portanto, mais duas condições: caso o caracter R seja recebido pela UART, envia-se um relatório contendo os valores máximo, mínimo e médio ao *bluetooth,* e caso L seja recebido, todas as amostras contidas na memória EEPROM são transmitidas ao mesmo. Tais condições estão representadas abaixo.

Programa 8: trecho da função que manipula a memória EEPROM extendida ao bluetooth.

```
(...)
char_pressed = read_bluetooth();

/* Envia media, max. e min. */
if (char_pressed == 'R') {

     char buffer[150];
     memset(buffer, 0, 150);

     /* Calcula partes decimal e fracionaria */
     float decimal_max = max - (int) max,
            decimal_min = min - (int)min ,
            decimal_avg  = sum/count - (int)(sum/count);

     int decimal_max_i = (int) (100*decimal_max),
          decimal_min_i = (int) (100*decimal_min),
          decimal_avg_i = (int) (100*decimal_avg);

     sprintf(buffer, "\n\rRELATORIO:\n\rMEDIA:
                     %d.%02d\n\rMAX.:%d.%02d\n\rMIN.:%d.%02d\n\r",
                     (int) (sum/count),  decimal_avg_i,
                      (int)max, decimal_max_i, (int)min, decimal_min_i);
     send_string_bluetooth(buffer);
}

/* Envia todas amostras presentes na memoria EEPROM */
if (char_pressed == 'L') {

     EE241_Address aux_address = 0x0;

     send_string_bluetooth("\n\rValores na memoria:\n\r");
```

```
     /* Percorre todas as posicoes de memoria */
     while (aux_address <= initial_address) {

          union Temperatura temp;
          char buffer_data[6], buffer_address[6];

          memset(buffer_data, 0, 6);
          memset(buffer_address, 0, 6);

          temp.temp_as_float = 0;
          memset(temp.temp_as_bytes, 0, sizeof(float));
          EE241_ReadBlock(aux_address, temp.temp_as_bytes,
                         sizeof(float));

          float decimal = temp.temp_as_float - (int) temp.temp_as_float;
          sprintf(buffer_data, "%d.%02d",
                 (int) temp.temp_as_float, (int) (decimal * 100));

          sprintf(buffer_address, "0x%x - ", aux_address);

          send_string_bluetooth(buffer_address);
          send_string_bluetooth(buffer_data);
          send_string_bluetooth("\n\r");

          aux_address += sizeof(float);
     }
}
(...)
```

Referências

[1] HC Serial Bluetooth Products – User Instructional Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/datasheet/hc_hc-05-user-instructions-bluetooth.pdf

[2] Using the HC-06 Bluetooth Module
http://mcuoneclipse.com/2013/06/19/using-the-hc-06-bluetooth-module/

[3] Esquemático do módulo bluetooth com o conector DB-9.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/complementos/Bluetooth.pdf

[4] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf