UART vs SPI para comunicação Arduino-Arduino

Resumo

Comparação entre UART e SPI para comunicação entre Arduino Mega 2560 e Arduino Uno em curtas distâncias

Gabriel Yoshiaki Hotta

Sumário

1 INTRODUÇÃO

Para comunicações em pequenas distancias existem diversas opções. No caso de micro controladores destacam-se a comunicação UART e a SPI. Ambas possuem suporte nativo via hardware na plataforma Arduino, tornando-as opções a serem consideradas ao estabelecer uma comunicação entre dois Arduinos.

Neste documento considerar-se-á a utilização do Arduino Mega2560 recebendo dados de um Arduino Uno. Para comunicação entre outros modelos, pequenas alterações podem ser necessárias.

2 REQUISITOS DO PROJETO

2.1 CARACTERÍSTICAS OBRIGATÓRIAS

- Obter fielmente o número de rotações de um equipamento.
- Enviar as informações para um Arduino

2.2 CARACTERÍSTICAS DESEJÁVEIS

- Alta velocidade de comunicação
- Escalabilidade (possível acréscimo de outros periféricos)

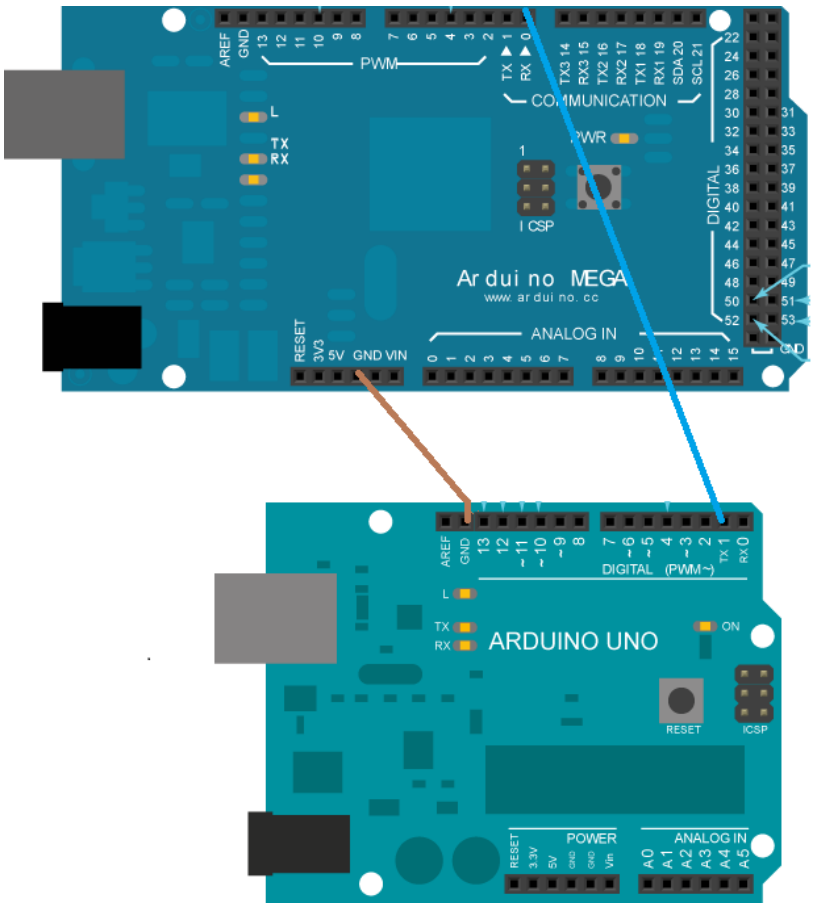
3 UART vs SPI

3.1 UART

A UART (Universal Asynchronous Receiver Transmitter) é um tipo de transmissão serial, conforme o nome diz, assíncrona entre dois dispositivos. Utiliza apenas dois fios, um para recepção e outro para transmissão, por consequência, pode ser utilizada apenas no modo half-duplex, ora um dispositivo pode enviar, ora pode receber, sendo impossível que ambos transmitam e recebam simultaneamente. Também existe uma variação da UART capaz de se comunicar de forma síncrona e no modo full-duplex denominada USART (Universal Sychronous Asynchronous Receiver Transmitter).

3.1.1 Ligação UART

Para Arduino Mega **apenas recebendo** dados de Arduino Uno (comunicação simplex) pode-se utilizar a seguinte ligação:



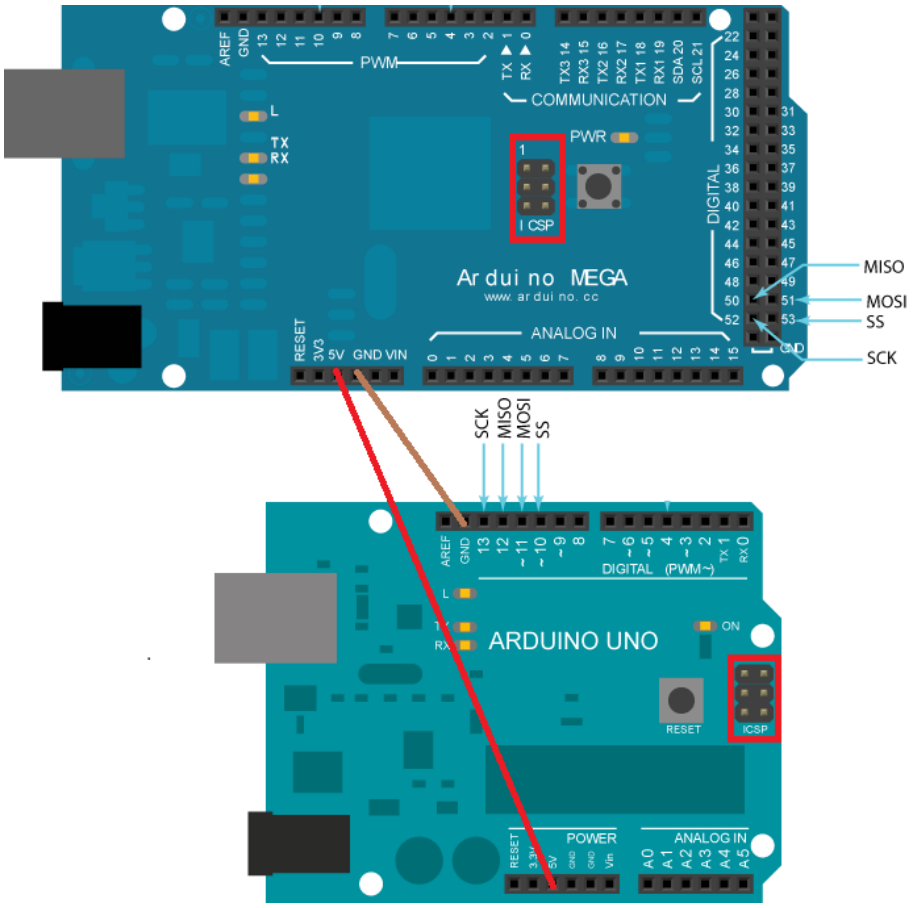
Note que para estabelecer o envio de dados do **Arduino Mega para o Arduino Uno** basta ligar o **Tx do Mega** no **Rx do Uno**

3.2 SPI

A comunicação SPI (Serial Peripheral Interface) é um outro tipo de interface serial bastante utilizada, principalmente entre controladores e periféricos. Utiliza o modelo mestre-escravo, o mestre (habitualmente um controlador) pode enviar dados ou comandos para o escravo (usualmente um periférico). Ao escravo cabe a responsabilidade de utilizar adequadamente os dados ou executar os comandos recebidos. Pode trabalhar com diversos escravos (mas somente um mestre). Utiliza pelo menos 5 fios para comunicação e 1 fio adicional para cada escravo. Em compensação, possui full-duplex, sendo possível enviar e receber dados simultaneamente.

3.2.1 Ligação SPI

Para utilizar SPI, podem ser usados os pinos MISO, MOSI, SS, SCK, GND e 5V ou os conectores ICSP.



3.3 TABELA COMPARATIVA ENTRE SPI E UART

	Quantidade de dispositivos	Facilidade de implementação	Direcionalidade	Quantidade de fios	Distância de comunicação	Velocidade
UART	Um	Fácil	Half-duplex	2	Curta	Média

SPI	Vários	Média	Full-duplex	4 ou mais	Curtíssima	Alta

4 ESCOLHA PARA O PROJETO

Devido a maior flexibilidade, optou-se se por utilizar SPI para comunicação entre os Arduinos. Apesar maior dificuldade de implementação e maior quantidade de fios, a alta velocidade de transmissão de dados em full-duplex e possibilidade de adicionar outros escravos tornam a melhor escolha nesse caso especifico.

5 IMPLEMENTAÇÃO

O código fonte para desenvolvedores encontra-se disponível e bem documentado.

5.1 UTILIZANDO UART

O projeto “speedracer” realiza a comunicação entre dois Arduinos via UART e contém dois subprojetos:

- Projeto “racerx”, a ser executado no Arduino Mega, trabalha como Rx, recebendo os dados enviados pelo Arduino Uno.
- Projeto “trixie”, a ser executado no Arduino Uno, trabalha como Tx, transferindo periodicamente os dados referentes ao número de voltas contabilizadas.

5.2 UTILIZANDO SPI

O projeto “wackyraces” realiza a comunicação entre dois Arduinos via SPI e contém dois subprojetos:

- Projeto “dastardly”, a ser executado no Arduino Mega, é o Mestre, sendo responsável por requisitar os dados provenientes do Escravo e trata-los da forma mais adequada.
- Projeto “muttley”, a ser executado no Arduino Uno, é o Escravo, sendo responsável por contar o número de voltas e enviar os dados quando requisitado pelo Mestre.

Alguns conceitos e técnicas utilizados serão abordados neste documento.

5.3 INTERRUPÇÕES NO ARDUINO

A aquisição de dados do contador de voltas ocorre através de interrupção, isso significa que a cada sinal recebido a execução normal é interrompida, o contador é incrementado, e retorna-se a execução normal.

Por segurança o controlador desabilita interrupções ao executar uma rotina de interrupção, entretanto, em alguns casos pode ser necessário que uma interrupção seja executada mesmo que uma outra rotina de interrupção esteja sendo executada.

O caso do contador de voltas enquadra-se nesse cenário, pois mesmo que esteja realizando a rotina de comunicação o controlador jamais deve perder a contagem de uma volta.

Para o compilador avr-libc utiliza-se a seguinte declaração (em outros compiladores pode ser necessário alterar):

```
ISR (SPI_STC_vect, ISR_NOBLOCK) {
  /* Manipula os dados SPI */
}
```

É importante ressaltar que interrupções aninhadas podem acarretar problemas se mal utilizadas, sendo imperativo muita cautela e uso apenas quando estritamente necessário.

5.4 REGISTRADOR SPI

A interface SPI utiliza o registrador SPCR

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SPIE – Habilita interrupção SPI quando 1.

SPE – Habilita SPI quando 1.

DORD – Envia LSB primeiro quando 1, MSB primeiro quando 0.

MSTR – Arduino em modo Mestre quando 1, Escravo quando 0.

CPOL – Sets the data clock to be idle when high if set to 1, idle when low if set to 0

CPHA – Dados na borda de descida quando igual a 1, dados na borda de subida quando igual a 0.

SPR1 e SPR0 – Velocidade do SPI, 00 o mais rápido (4MHz), 11 o mais lento (250KHz).

5.5 PROTOCOLO DE RECEBIMENTO DE DADOS

A implementação de SPI no Arduino é capaz de transferir apenas 1 byte por vez, para transferir um long int (4 bytes no Arduino) é necessária a transmissão individual de cada byte utilizando sucessivas requisições (aprimorar futuramente utilizando somente um comando [???]):

```
#define GET_COUNTER 0x0f
#define REQ_BYTE0 0x10
#define REQ_BYTE1 0x11
#define REQ_BYTE2 0x12
#define REQ_BYTE3 0x13
#define REQ_CHECKSUM 0x14
#define END_DATA 0xFF
```

```
transferAndWait(GET_COUNTER); //commando para pegar valor
transferAndWait(REQ_BYTE3); //requisitar byte3
byte byte3 = transferAndWait(REQ_BYTE2); //receber byte3 e requisitar byte2
byte byte2 = transferAndWait(REQ_BYTE1); //receber byte2 e requisitar byte1
byte byte1 = transferAndWait(REQ_BYTE0); //receber byte1 e requisitar byte0
byte byte0 = transferAndWait(REQ_CHECKSUM); //receber byte0 e pedir checksum
byte checksumReceived = transferAndWait(END_DATA); //finalize
```

Note que as requisições aproveitam o aspecto full-duplex da SPI.

Há ainda o checksum para ser comparado e garantir que o valor recebido está correto.

6 CONCLUSÃO

Para aplicações simples a UART é uma boa escolha, pois permite uma rápida implementação. Entretanto, quando consideramos projetos utilizando dispositivo próximo, com possível demanda de expansão na quantidade de dispositivos e alta velocidade, torna-se evidente a necessidade de utilizar uma solução mais elaborada, sendo, neste caso, o uso da SPI o mais recomendado.