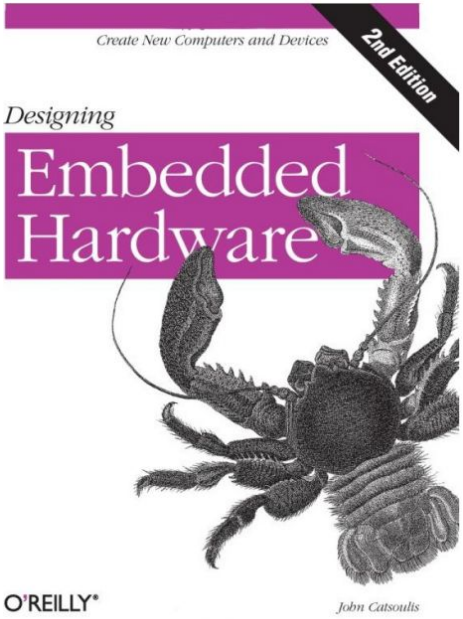
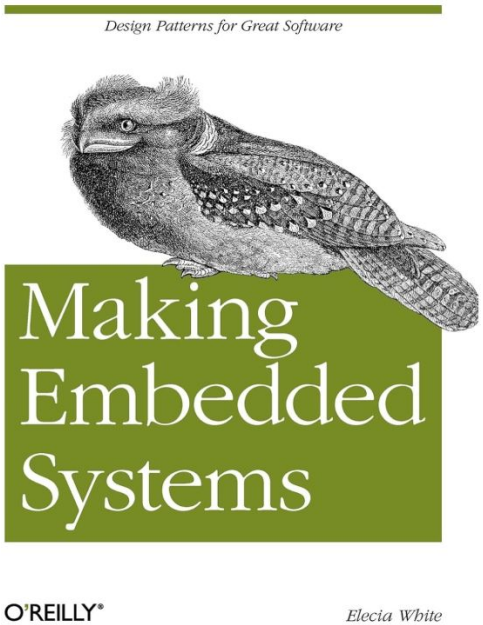
Comunicação Serial

UART I²C SPI

João Felipe Amaral Santiago





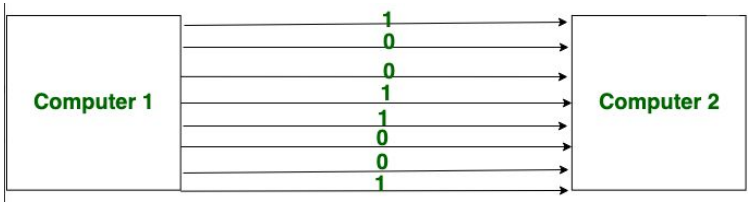
Slides das aulas

https://github.com/JaoIndio/Docencia_Microcontroladores

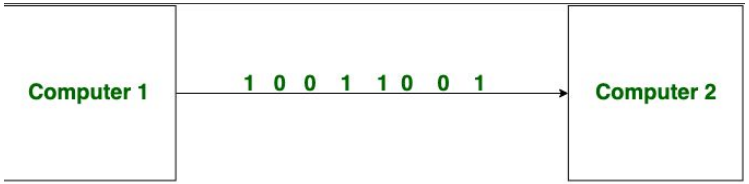
Fundamentos - Serial

Trata-se de um termo extremamente amplo, pois diz-se que uma comunicação é **serial** quando ocorre a transferência de bits ao longo do tempo por meio de um canal ou barramento.

Em contraposição há a comunicação **paralela.** Nela, todos os bits são enviados de uma única vez atrávez de multiplos barramentos ou canais.



Paralelo



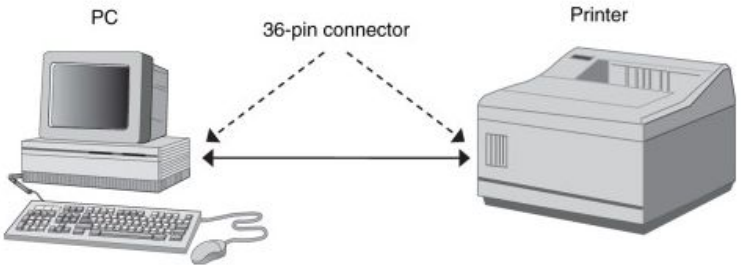
Serial

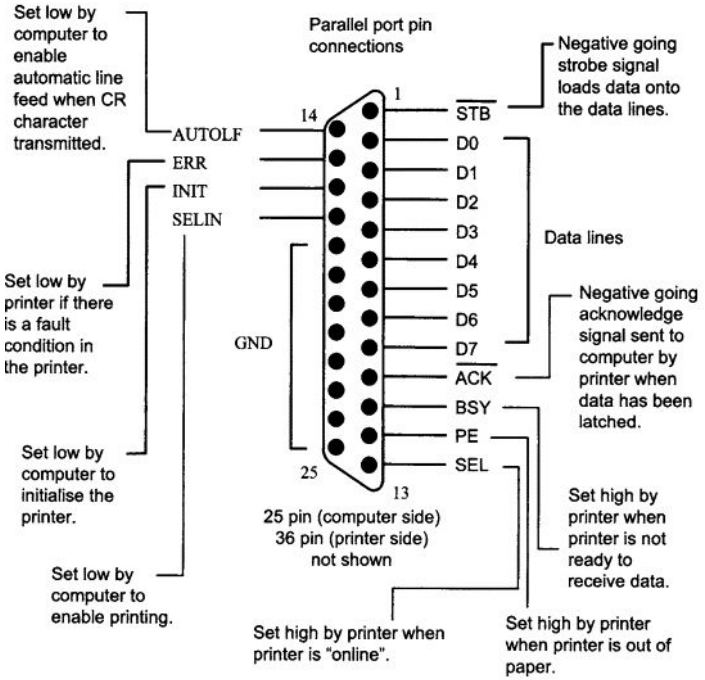
Fundamentos - Serial

Em essência a totalidade das comunicações entre computadores sempre será serial, uma vez que é inviável enviar todas as informações necessárias em um único barramento em um **único ciclo**.

A comunicação entre a CPU e memória é um exemplo de comunicação puramente paralela.

A comunicação com impressoras mais antigas também.



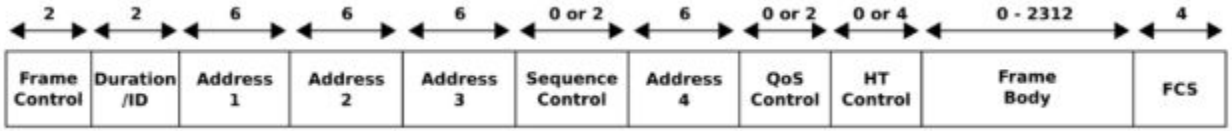


Fundamentos - Serial

Desenvolvida nos anos 80 pela Western Digital, Compaq e IBM, o padrão ATA (*Advanced Technology Attachment*) foi implementado com o propósito de integrar os discos rígidos e demais dispositivos de memória não volátil da época.

Suas variações possuíam de 40 a 80 pinos. No início dos anos 2000 foi substituído pelo padrão SATA (*Serial Advanced Technology Attachment)*.

O protocolo **WiFi** também transmite seus frames serialmente, **apesar** de explorar técnicas de envio muito sofisticadas as quais envolvem por exemplo, a divisão de seu frame em símbolos os quais são enviados em diferentes frequências. Esses símbolos são enviados **simultaneamente,** porém é necessário uma batelada de envios simultâneos para se ter o envio completo de um frame.



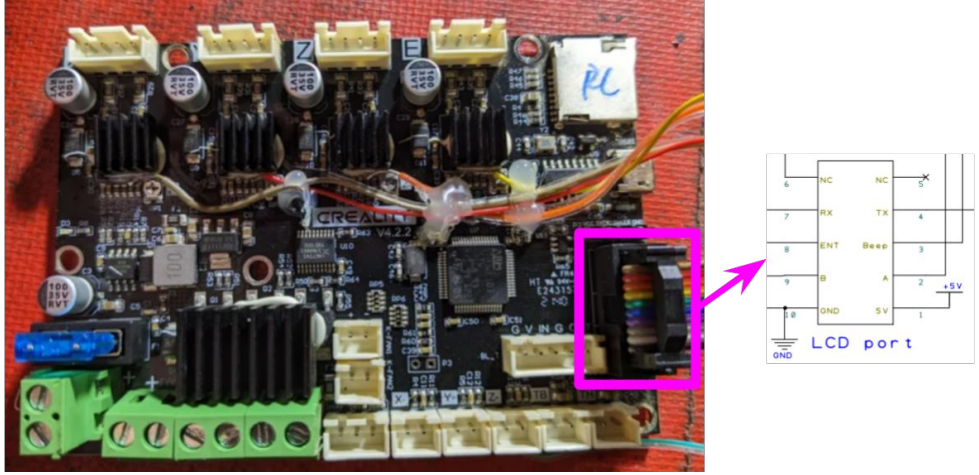
Campos do frame do protocolo IEEE 802.1 (padrão WiFi lançado em 1997)

Fundamentos - Serial

Dentro do escopo de **microcontroladores** e por consequência, sistemas embarcados, os protocolos comumente utilizados nas aplicações são o **UART**, **I²C** e **SPI**.

Quando comparados com outros protocolos cabeados seriais (CAN, ModBus, USB, PCIe, SATA, HDMI, Ethernet, etc), esses três são os mais simples, possuem menor complexidade portanto, sua implementação tende a ter custo reduzido.

São fortemente empregados para conectar e configurar RTCs (Real-Time Clock), memórias não voláteis (flash, EEPROM, etc), interfacemanto com sensores, atuadores e vários outros tipos de periféricos.

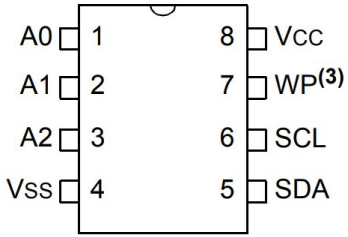


MICROCHIP

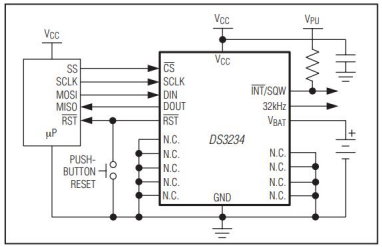
24AA00/24LC00/24C00 24AA01/24LC01B
24AA014/24LC014 24C01C/24C02C
24AA02/24LC02B 24AA025/24LC025
24AA024/24LC024 24AA08/24LC08B
24AA04/24LC04B 24AA32A/24LC32A
24AA16/24LC16B 24AA128/24LC128/24FC128
24AA64/24LC64/24FC64 24AA512/24LC512/24FC512
24AA256/24LC256/24FC256 24AA1025/24LC1025/24FC1025

I²C™ Serial EEPROM Family Data Sheet

PDIP/SOIC



Typical Operating Circuit



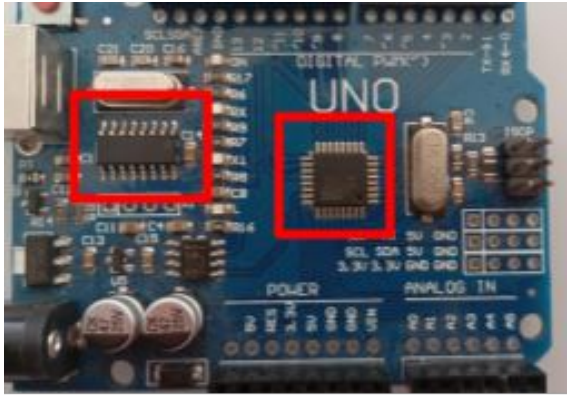
maxim integrated

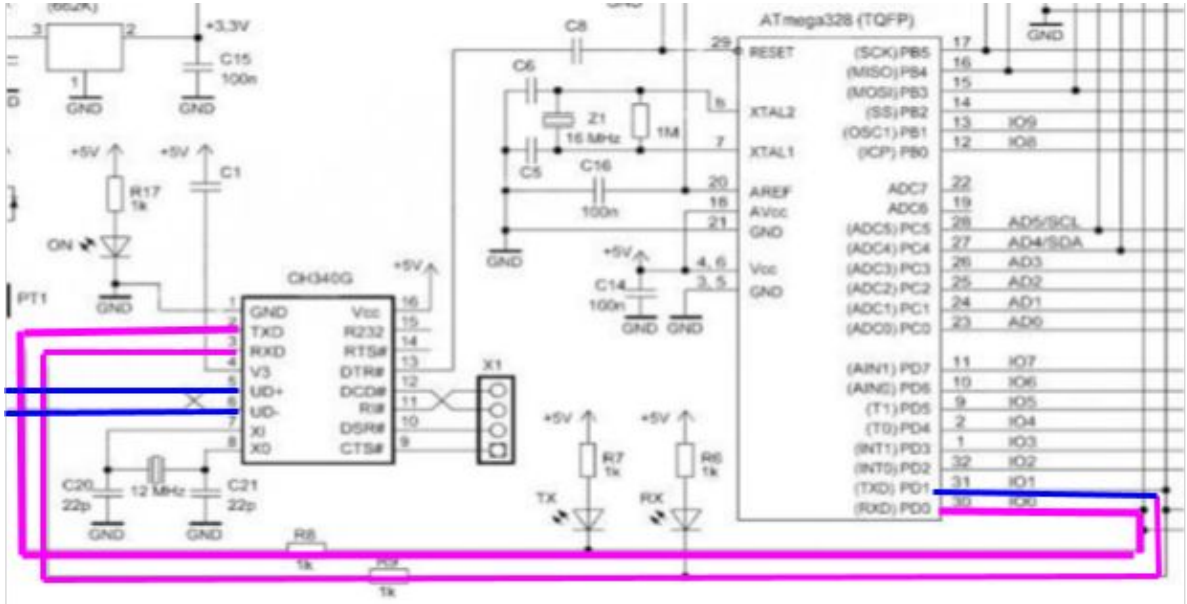
Extremely Accurate SPI Bus RTC with Integrated Crystal and SRAM

UART

Conecta o sistema embarcado ao mundo externo. Realiza a transferência de dados a partir do uso de algum outro protocolo físico (RS-232, RS-232C, RS-422, RS-485, USB, etc).

No Arduino Uno o micro ATmega328P liga os pinos TX e RX ao ci CH340G, que por sua vez é um conversor UART/USB.





UART - Universal Asynchronous Receiver Transmitter

É um dos protocolos mais simples. O termo assíncrono é atribuído a ele pois não há nenhum barramento responsável por ser o clock. Nesse sentido o recebedor deve ser capaz de detectar os bits individualmente sem o auxílio de sinais para sincronização.

Além dos barramentos TX/RX o UART fornece informações relacionadas ao status das informações, tais como se o buffer do receptor está cheio, ou o buffer de transmissão está vazio.

UARTs actually predate semiconductor-based computers. In the early days of electrical communication, UARTs were mechanical devices with cogs, relays, and electromechanical shift registers. To adjust a UART's settings, you first picked up a wrench!

UART

Uma de suas grandes desvantagens é a dificuldade na reconstrução dos dados recebidos. Uma dos maiores desafios de sua operação é a detecção da separação entre um bit e outro.

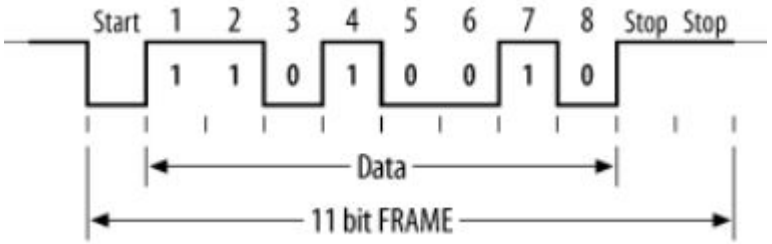
Por exemplo, se um barramento permanece em nível baixo por um dado período de tempo, o dispositivo receptor deve ser capaz de identificar se conjunto representa ‘00’ ou ‘000’. Ou seja, é necessário saber onde um bit termina e o próximo começa.

A solução mais prática para esse problema é o uso de um clock comum. Portanto em comunicações seriais **síncronas**, transmissor e receptor compartilham o mesmo clock. No caso de protocolos **assíncronos** os participantes da comunicação possuem seus próprios clocks. Consequentemente os dispositivos precisam necessariamente estarem configurados para operar na exata mesma frequência

UART

O formato de sua transmissão usa um bit de início no começo e um ou dois bits para indicar o fim. Uma vez identificado o bit de início, o receptor amostra 7 ou 8 bits (depende de sua configuração).

Uma vez recebido o *stop* bit, caso a sequência correta tenha sido enviada, o receptor irá supor que um caracter válido foi enviado e irá disponibilizado no buffer RX.



UART

Slides das aulas

https://github.com/JaoIndio/Docencia_Microcontroladores

Atividade - Material Complementar.

Implementar um echo entre dois Arduino Mega

Serial Monitor (COM1) -> Arduino 1 -> Arduino 2 -> Serial Monitor (COM2)

https://www.youtube.com/watch?v=P0O9sVRe15I
https://docs.arduino.cc/language-reference/en/functions/communication/serial/

https://docs.arduino.cc/micropython/communication/uart/

https://docs.arduino.cc/learn/communication/uart/ ←

https://docs.arduino.cc/built-in-examples/communication/ASCIITable/

https://docs.arduino.cc/built-in-examples/communication/SerialCallResponseASCII/