

COMUNICAÇÃO I2C

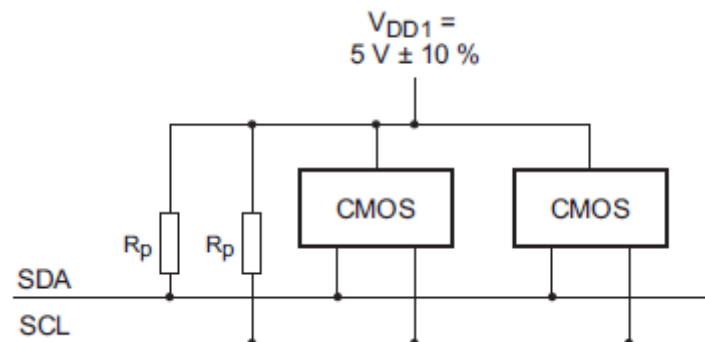
O protocolo de comunicação I2C é um padrão muito utilizado. Tem essa denominação por é constituída de um par de fios (linha), que são SDA (Dados) e SCL (Clock), e que formam o Bus I2C. É uma comunicação serial de 8 bits com transferência de dados bidirecional. Em seu modo padrão de operação pode transferir em até 100 kbits/s ou 128 bytes/s.

A comunicação I2C pode ter várias configurações, como por exemplo: um mestre e um escravo, e mestre com mais de um escravo e outras. Nesse estudo meu foco é a configuração mais simples que é um mestre (Arduino UNO) e um escravo (Display 16x2 I2C interfaceado pelo CI PCF8564), porque o objetivo é entender em detalhes a troca de dados e visualizar o protocolo, e não suas mais variadas configurações.

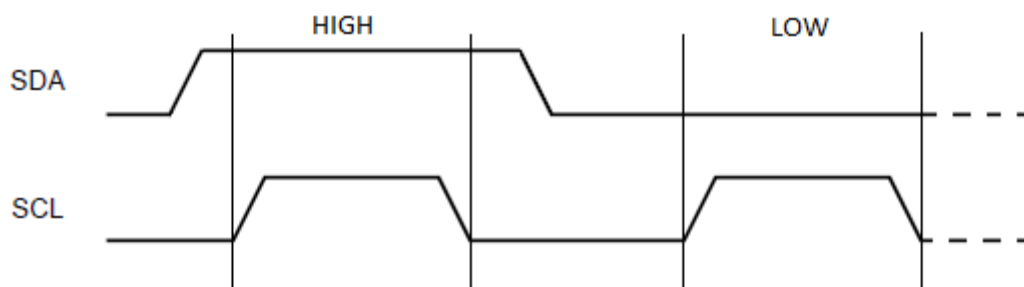
Protocolo I2C

Dois fios, Serial Data (SDA) e Serial Clock (SCL), transferem informações entre os dispositivos conectados ao Bus. Cada dispositivo é reconhecido por um endereço único e pode operar como transmissor ou receptor, dependendo de sua função. Um LCD pode somente receber dados enquanto uma memória pode receber e transmitir dados. O dispositivo mestre é o que inicia a transferência de dados no Bus e gera os sinais de Clock para permitir a transferência. Ao mesmo tempo cada dispositivo endereçado é considerado um escravo.

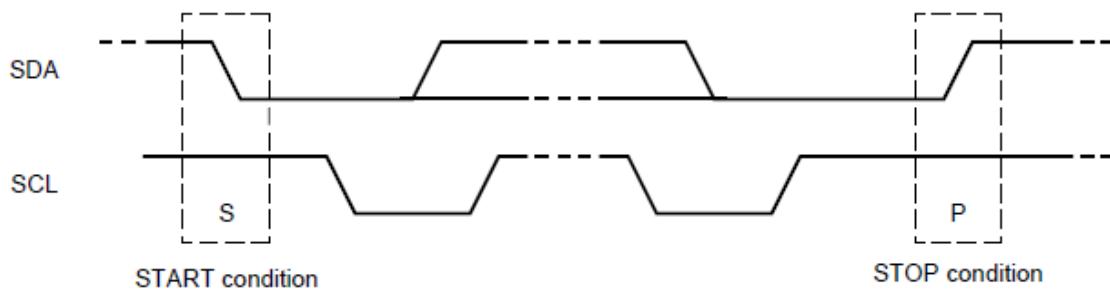
Ambas as linhas SDA e SCL são bidirecionais (Tri-state), conectados ao positivo via resistores de Pull-Up. Quando o Bus estiver livre ambos os sinais são nível HIGH.



Para ser um dado válido, HIGH ou LOW, o nível do dado deve ficar estável durante o nível HIGH do pulso do Clock. Um pulso de Clock é gerado para cada bit a ser transferido.

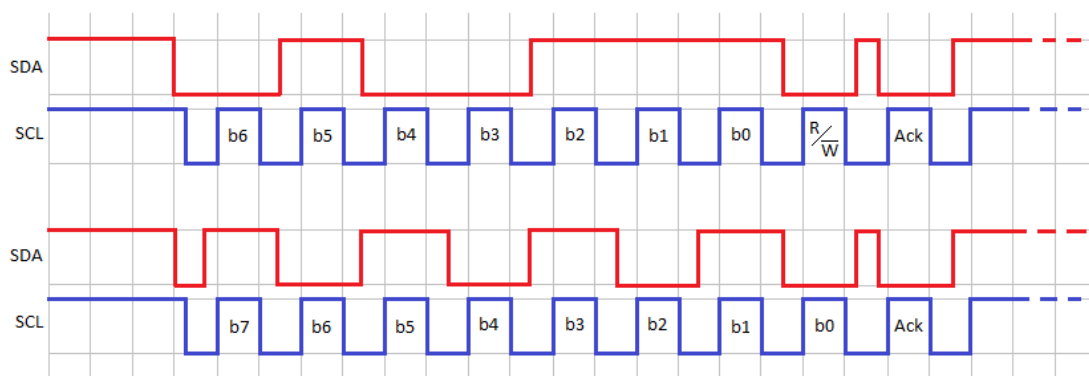


Toda transferência inicia com uma condição de START e termina com uma condição de STOP. Uma Transição de HIGH para LOW da linha de SDA enquanto a linha de SCL é HIGH define uma condição de START. Uma Transição de LOW para HIGH da linha de SDA enquanto a linha de SCL é HIGH define uma condição de STOP. Essas condições são geradas pelo mestre. Após a condição de START o Bus é considerado ocupado, e o Bus é considerado livre novamente depois de um determinado tempo após a condição de STOP.



Cada byte colocado no Bus obviamente deve ter 8 bits de comprimento. A cada byte transferido é seguido por um bit de reconhecimento (Acknowledge), portanto, a cada transferência é gerado nove pulsos de Clock (SCL), onde oito são para os dados e o nono para o bit de reconhecimento. Não há restrição a quantidade de bytes a ser transferido, porém, a transferência é realizada byte a byte. Também vale destacar que a cada transferência de dados o primeiro byte é do endereço e segundo é o dado a ser transferido. O exemplo a seguir um mestre iniciou a comunicação e o endereço do dispositivo escravo é #27 e o dado enviado é #AA.

Note que o primeiro byte, de endereço, tem suas particularidades. Oito bits são transferidos, mas, 7 bits são realmente para endereço e o oitavo bit informa se o mestre quer somente escrever no escravo ou se quer ler um dado do escravo. Nesse caso o bit R/W é LOW, então o mestre somente quer escrever no escravo. Note também que o segundo byte é puramente 8 bits de dados.



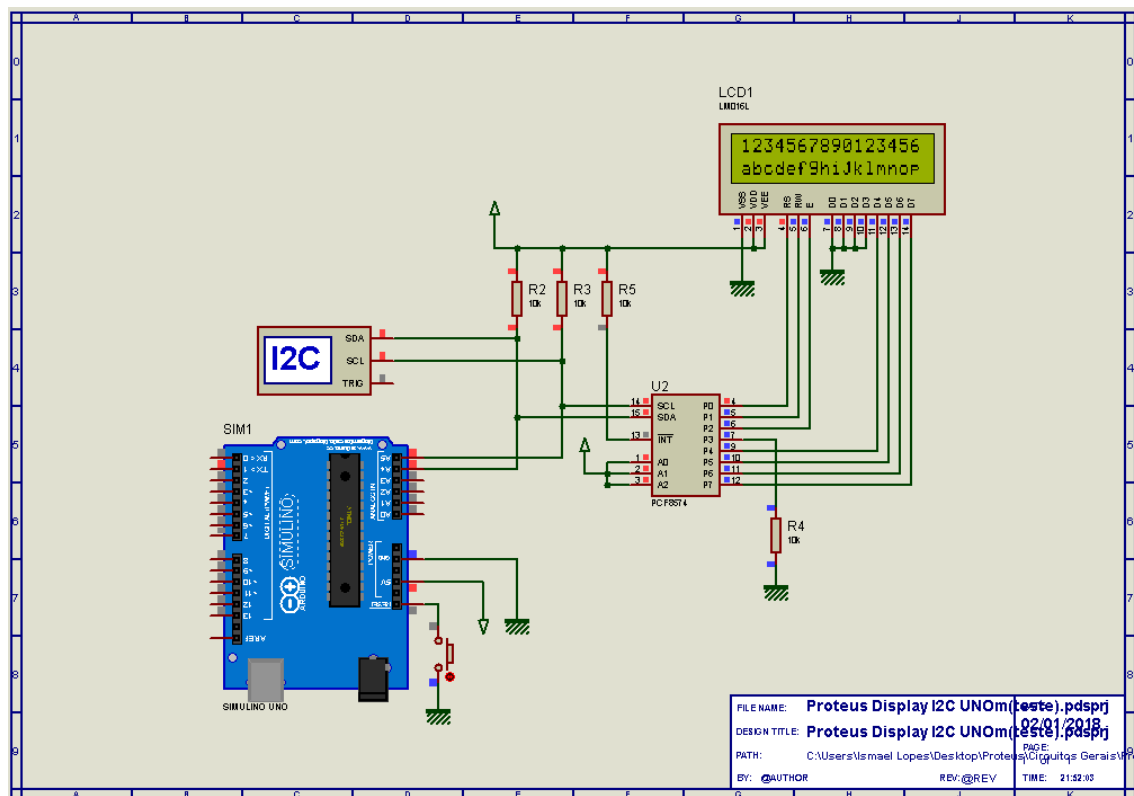
Observe também que o nono pulso de Clock (SCL) é exclusivo para reconhecer que a transferência do byte ocorreu, então, isso se repete a cada byte. O mestre gera todos os pulsos de Clock, inclusive o de reconhecimento (Ack). O reconhecimento ocorre da seguinte forma: o transmissor libera a linha de dado (SDA) durante o Clock de reconhecimento (Ack), então, o

receptor coloca a linha de dado (SDA) em LOW e mantém estável durante o nono pulso, obviamente se recebeu corretamente o byte. Se alguma falha ocorreu o receptor mantém em HIGH a linha de dado (SDA).

Referência bibliográfica usado nesse estudo foi o documento de especificação e manual para o Bus I2C – UM10204 da NXP Semiconductors.

Um Caso Prático

Para complementar esse estudo eu montei no Proteus um circuito para simular uma comunicação simples I2C entre um mestre (Arduino UNO) e um escravo (Display 16x2).



O Proteus tem um Debugger I2C para auxiliar nessa análise, então, foi inserido para obtermos informações do Bus I2C.

Para entender essas informações temos que entender as particularidades dessa aplicação, porque só assim podemos visualizar o funcionamento da comunicação I2C, que é objeto principal desse estudo.

Dados da Aplicação:

Que faz a interface I2C entre o Arduino e o Display é o circuito integrado PCF8574, porque o Display propriamente não comunica em I2C. O PCF8574 tem três pinos de endereçamento, que são: A0, A1 e A2. Todos estão conectados ao HIGH, portanto, seu endereço em hexadecimal é #27.

O Display pode operar no modo 8 bits de dados ou 4 bits de dados. Internamente o Display trabalha com 8 bits, então, se operamos no modo 4 bits precisaremos de dois ciclos de

3 de janeiro de 2018.

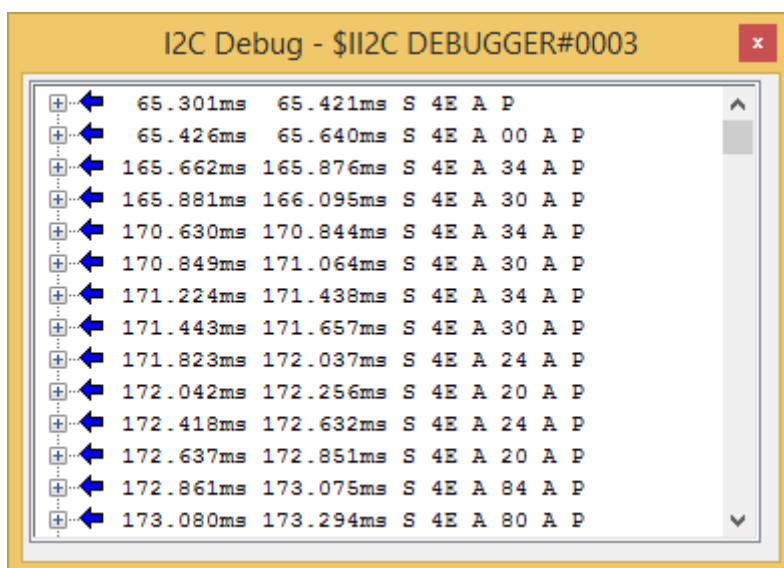
Ismael Lopes

instrução para formar os 8 bits para o Display. Observe que nossa aplicação é configurada para operar no modo 4 bits de dados (P4...P7). Os pinos de P0...P3 são usados diretamente nos pinos de controle do Display, mesmo que efetivamente usamos os pinos P0...P2. O pino P3 não usamos nessa aplicação.

Também vale destacar que o Display recebe bytes de comando e bytes de dados, sendo esse ultimo os caracteres que desejamos mostrar no visor. Para facilitar o entendimento dessa aplicação imprimimos duas sequencias, uma em cada linha do Display, porque na tabela ASCII os códigos hexadecimais são sequenciais. Para a sequência de '0' à '9', os códigos são #30, #31, #32, #33, #34, #35, #36, #37, #38 e #39. Para a sequência de 'a' à 'p', os códigos são #61, #62, #63, #64, #65, #66, #67, #68, #69, #6A, #6B, #6C, #6D, #6E, #6F e #70.

Análise do Debugger:

Agora vamos analisar as informações obtidas pelo Debugger I2C. Segue o primeiro trecho das informações coletadas:



Cada linha é uma transferência de dados, onde a primeira linha somente identifica que o escravo esta presente, e nas demais linhas dois bytes são transferidos de cada vez, mas, sempre o primeiro byte é o especial, contendo endereço e bit de R/W.

Analisar a primeira linha –

Observe que essa linha pode ser expandida para mais detalhes, porém, como esta já nos passa muita informação. As duas primeiras colunas são informações de tempo (duração). Depois vem um 'S' indicando uma condição de START. Depois vem 4E que esta em hexadecimal, portanto, seria b(01001110) em binário, onde o bit 0 é o R/W e do bit 1 ao bit 7 representa #27 em hexadecimal que é o endereço o escravo. Depois vem um 'A' que indica um reconhecimento do escravo e por ultimo um 'P' que indica uma condição de STOP.

Analisar a segunda linha –

As duas primeiras colunas são informações de tempo (duração). Depois vem um 'S' indicando uma condição de START. Depois vem 4E que indica o bit especial (endereço + R/W). Depois

vem um 'A' que indica um reconhecimento do escravo. Depois vem um dado em hexadecimal #00. Depois vem um 'A' que indica um reconhecimento do escravo e por último um 'P' que indica uma condição de STOP.

Analisar a terceira linha –

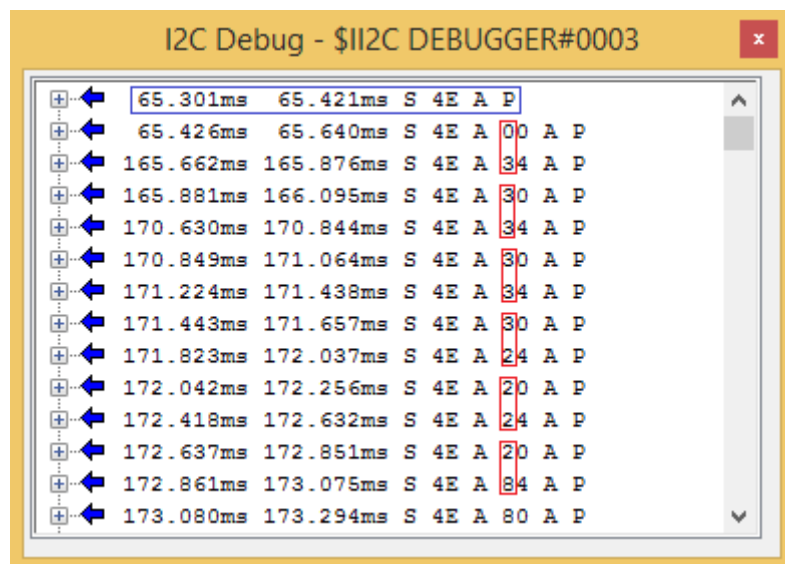
As duas primeiras colunas são informações de tempo (duração). Depois vem um 'S' indicando uma condição de START. Depois vem 4E que indica o bit especial (endereço + R/W). Depois vem um 'A' que indica um reconhecimento do escravo. Depois vem um dado em hexadecimal #34. Depois vem um 'A' que indica um reconhecimento do escravo e por último um 'P' que indica uma condição de STOP. Já os quatro bits menos significativos dos dados

Por enquanto vamos parar nossa análise de linhas e eu quero destacar que os dados da segunda e terceira linhas nessa aplicação são divididos em dois grupos de 4 bits para formar 8 bits internamente no controlador do Display, conforme expliquei sobre essa aplicação.

Os quatro bits mais significativos de cada linha formam o dado #03 em hexadecimal que é um código para inicializar o Display. Os bits quatro bits menos significativos dos dados formam os sinais de controle do Display (RS, RW e EN), portanto, vamos focar mais nos 4 bits mais significativos dos dados.

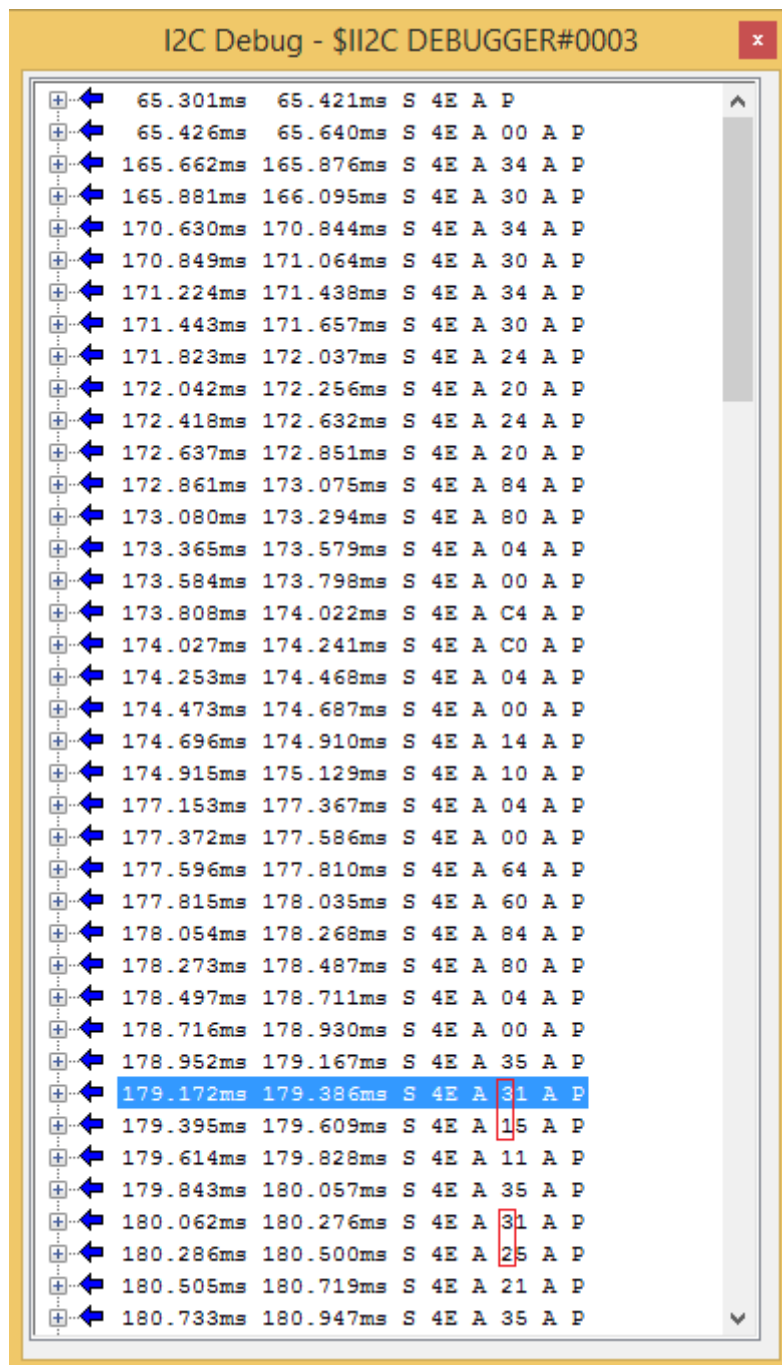
Novamente destacando a cada duas transferências de dados o Display utiliza os 4 bits mais significativos para compor um dado interno de 8 bits. Vários dados iniciais são para inicializar e configurar o Display, até mesmo para definir que ele trabalhará no modo 4 bits de dados externos.

Ainda não chegamos na transferência propriamente dos caracteres que desejamos visualizá-los no LCD. Segue novamente o primeiro trecho com alguns destaques que já fizemos.



Time (ms)	Address	Direction	Data	Control
65.301ms	65.421ms	S	4E	A P
65.426ms	65.640ms	S	4E	A 00 A P
165.662ms	165.876ms	S	4E	A 34 A P
165.881ms	166.095ms	S	4E	A 30 A P
170.630ms	170.844ms	S	4E	A 34 A P
170.849ms	171.064ms	S	4E	A 30 A P
171.224ms	171.438ms	S	4E	A 34 A P
171.443ms	171.657ms	S	4E	A 30 A P
171.823ms	172.037ms	S	4E	A 24 A P
172.042ms	172.256ms	S	4E	A 20 A P
172.418ms	172.632ms	S	4E	A 24 A P
172.637ms	172.851ms	S	4E	A 20 A P
172.861ms	173.075ms	S	4E	A 84 A P
173.080ms	173.294ms	S	4E	A 80 A P

Agora vamos abrir um trecho de visualização do Debugger para avançar na análise. Trinta (30) linhas de dados foram enviadas para compor quinze (15) bytes de comandos de inicialização do LCD. A partir daí inicia a transferência dos caracteres ASCII.



Mesmo ao iniciar a transferência dos dados que representam os caracteres a serem visualizados no LCD, entre cada caráter tem um byte de comando para o controlador do LCD mudar de linha, como pode ser visto o código #13 em hexadecimal entre o primeiro e segundo caráter.

Acho que os detalhes foram passados porque não é necessário analisar até o final dos dados do Debugger porque a sequencia de repete entre dado caráter e dado de comando do LCD.

Atenciosamente, Ismael Lopes.