ECE 3730 Principles of Embedded Systems:
Serial Communications

Overview,

Generic Serial Communications

Overview

- Asynchronous Serial Communications
  - RS-232
  - USB

- Synchronous Serial Communications
  - Generic
  - SPI
  - I^2C

Generic Synchronous Serial Communications
Inter-Connection Architecture

- Two or more devices may be inter-connected using five wires:
  - Request to Send (RTS)
  - Clear to Send (CTS)
  - Serial Data Out (SDO)
  - Serial Data In (SDI)
  - Serial Clock (SCK)

Generic Synchronous Serial Communications

Directionality

- Full Duplex
  - Data may be transmitted simultaneously in both directions
  - Not a requirement, but the option of simultaneous bi-directional transmission is supported

- Half Duplex
  - Allows data to be transmitted in both directions, but not simultaneously, i.e., simultaneous bi-directional transmission is not supported

Generic Synchronous Serial Communications

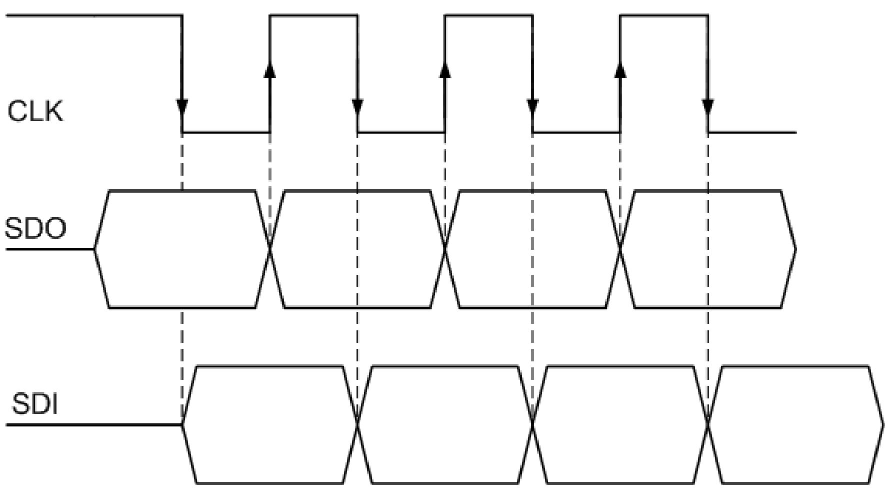
Hardware Handshaking Protocol

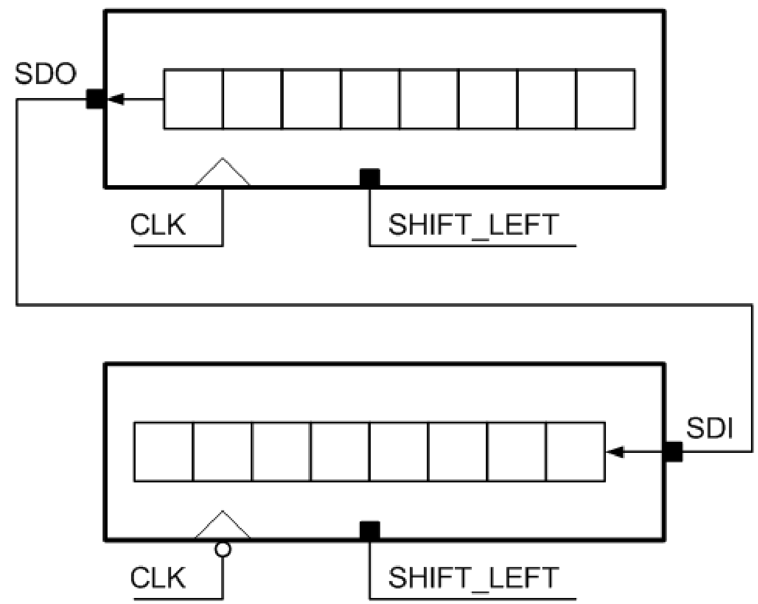
- RTS and CTS are hardware handshaking control lines
- Example: Device 1 requests to send to Device 2:
  - D1 requests to send by asserting D1 RTS
    - D1 RTS is connected to D2 CTS
    - D2 senses D2 CTS asserted, and if it wants to grant the request, D2 will assert D2 RTS
    - D2 RTS is connected to D1 CTS
    - D1 senses D1 CTS asserted, and at this point knows that D2 has granted its request

Serial Bit Transmission/Reception

Transmission/Reception Synchronized by Clock

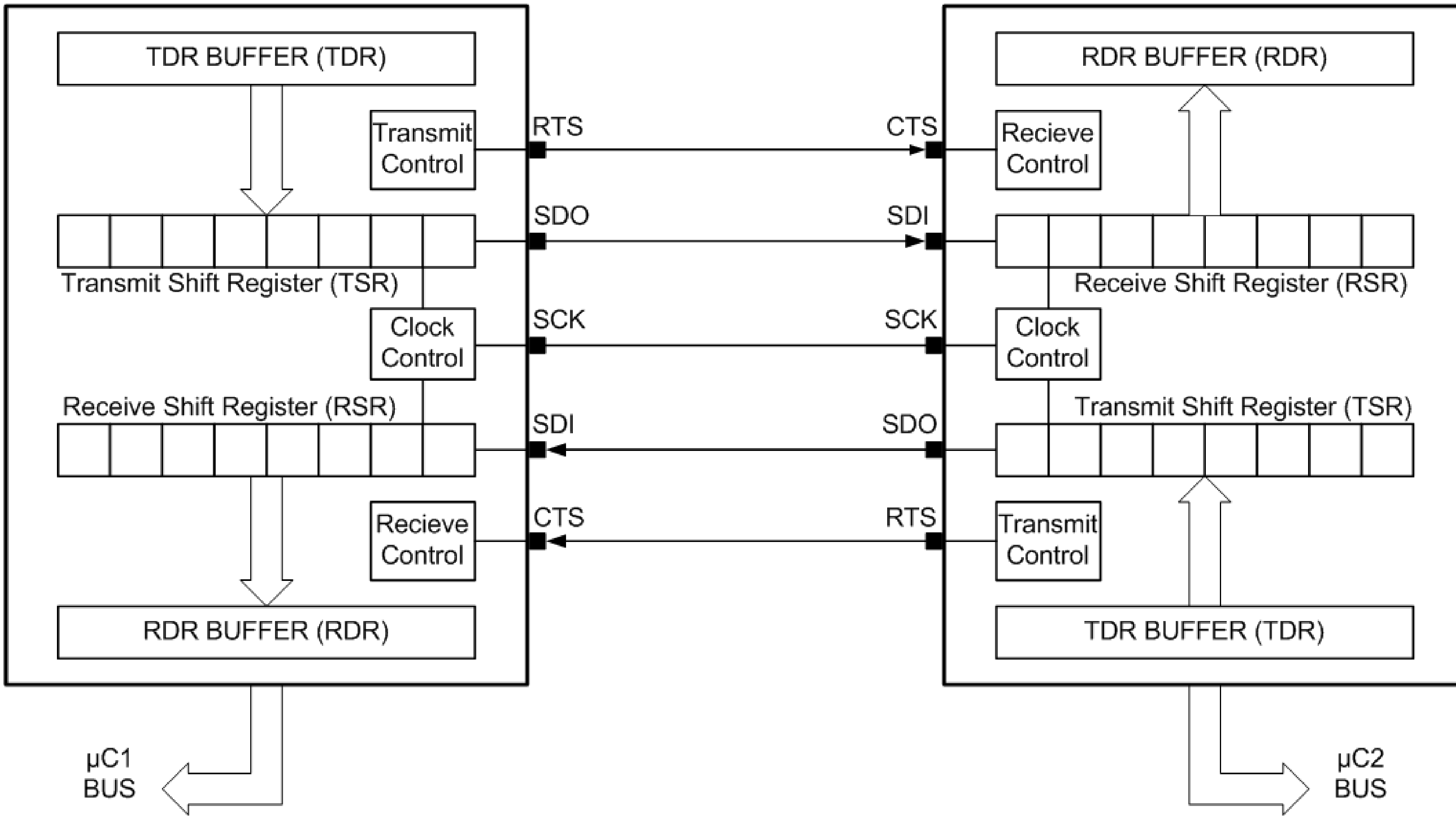
- Data is transmitted from one device to the other serially, i.e., one bit at a time
- Transmission of the data is synchronized with the clock
- The transmitter sends a bit at the first edge of the clock period, and the receiver receives the bit at the second edge of the clock period.





Generic Synchronous Serial Communications

Detailed Inter-Connection Architecture



Generic Synchronous Serial Communications

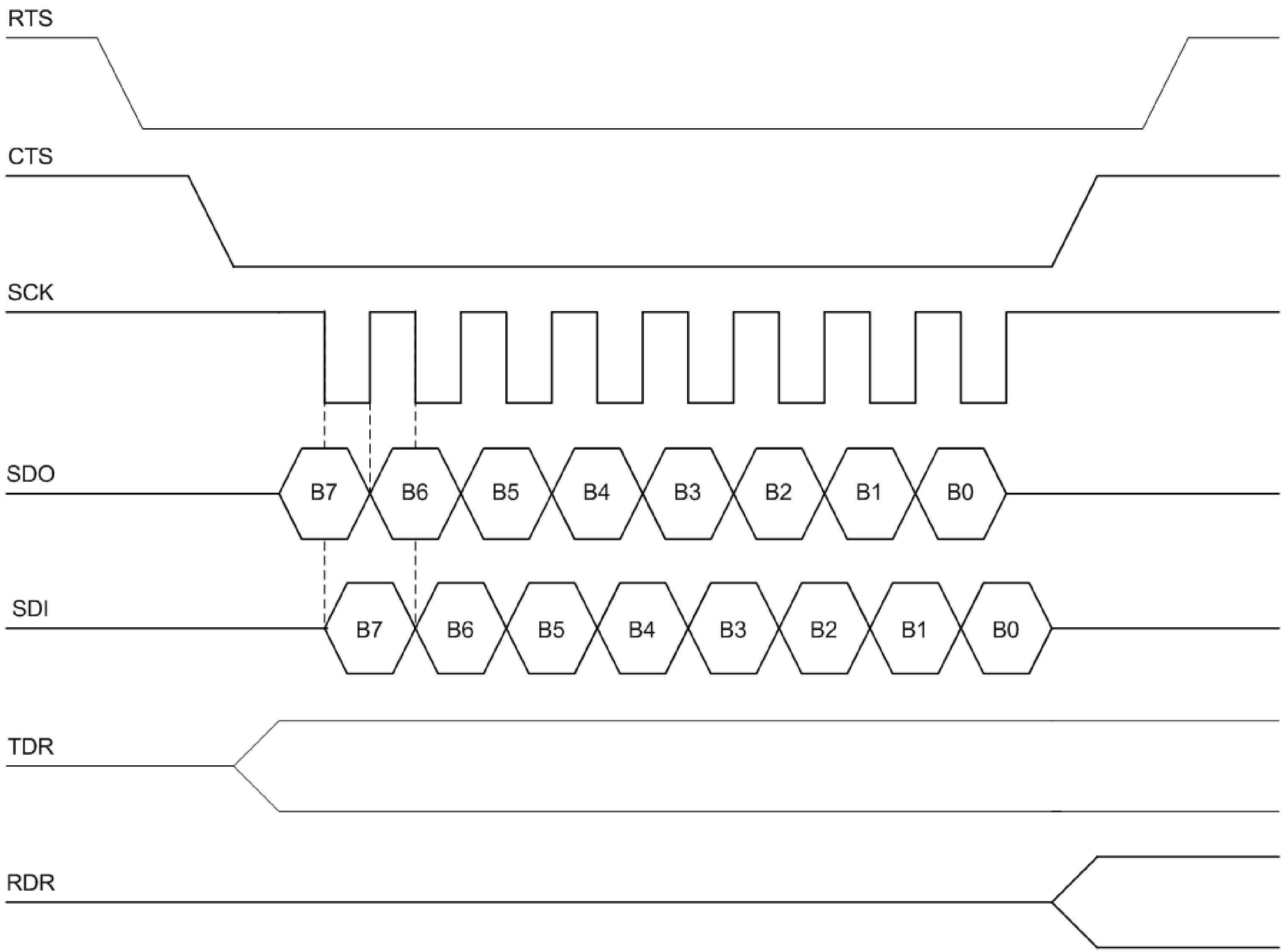
Transmit and Receive Operation

- Transmit Operation
  - μC1 asserts Request to Send (RTS), and then waits for Clear To Send (CTS)
  - μC1 writes a byte of data to the Transmit Data Register (TDR)
  - TDR is copied to the Transmit Shit Register (TSR)
  - Data is shifted out onto the SDO pin, one bit at a time
    - Each bit is sent on the first edge of the clock (SCK)

- Receive Operation
  - μC2 detects Request to Send (RTS), and then sends Clear To Send (CTS)
  - Data is shifted into the Receive Shift Register (RSR), one bit at a time
    - Each bit is received on the second edge of the clock (SCK)
  - When a complete byte has been received, the RSR is copied into the Received Data Register (RDR)
  - μC2 reads the data from the RDR

Generic Synchronous Serial Communications

Transmit and Receive Timing



Generic Synchronous Serial Communications

Bit-Banging Implementation

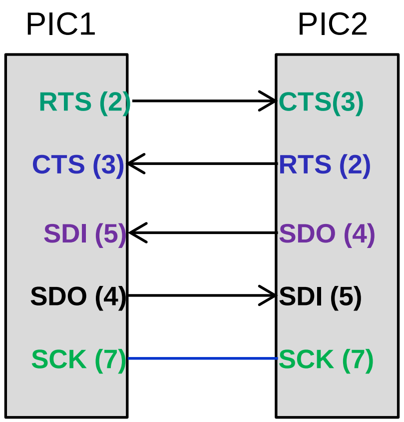
- Port pins are used to implement hardware handshaking RTS and CTS
  - RTS port pin is configured as an output
    - Choose RA0
  - CTS port pin is configured as an input
    - Choose RA1

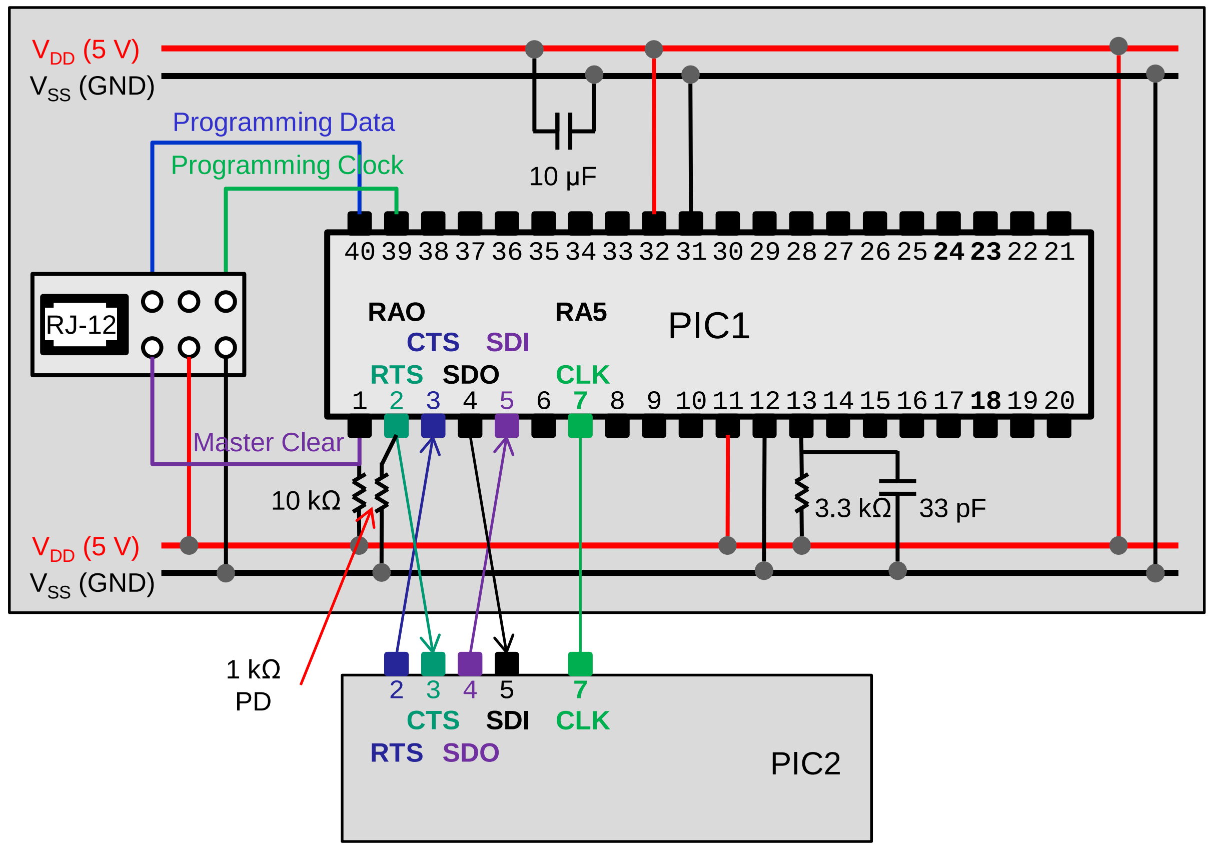
- Port pins are used to implement the serial data input and output lines
  - SDO port pin is configured as an output
    - Choose RA2
  - SDI port pin is configured as an input
    - Choose RA3

- Port pin may be used to implement the serial clock (SCK)
  - SCK port pin is configured as an output
    - Choose RA5

Generic Synchronous Serial Communications

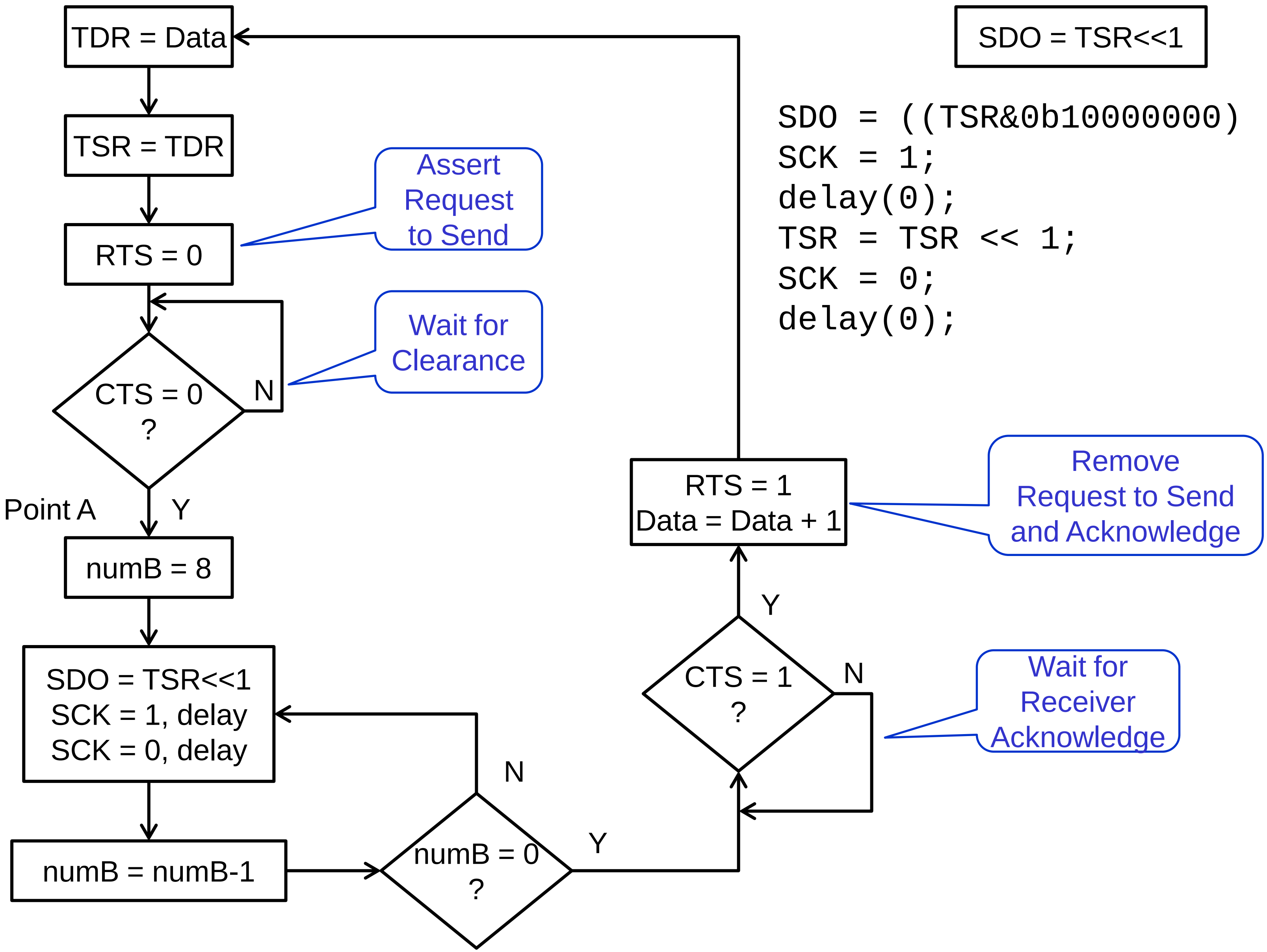
Bit-Banging Schematic





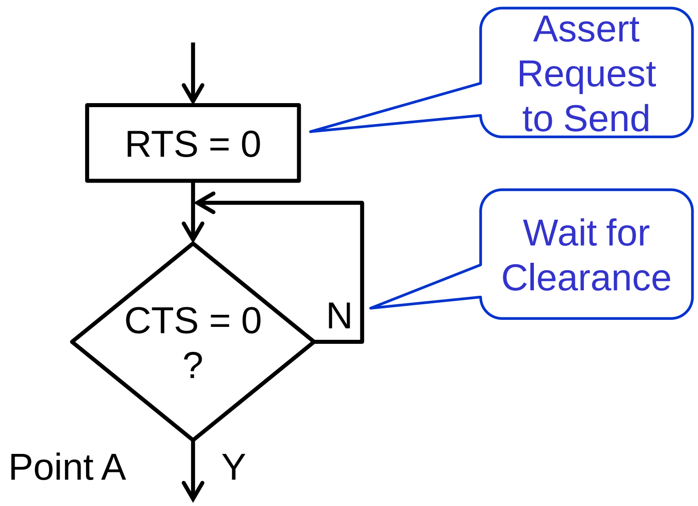
Generic Synchronous Serial Communications

Bit-Banging Transmitter Flowchart: Example of sending 0..255 continually



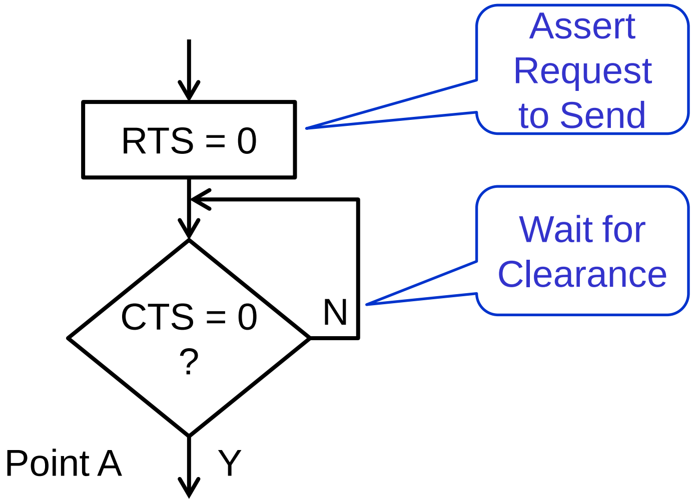

```
SDO = ((TSR&0b10000000) != 0);
SCK = 1;
delay(0);
TSR = TSR << 1;
SCK = 0;
delay(0);
```

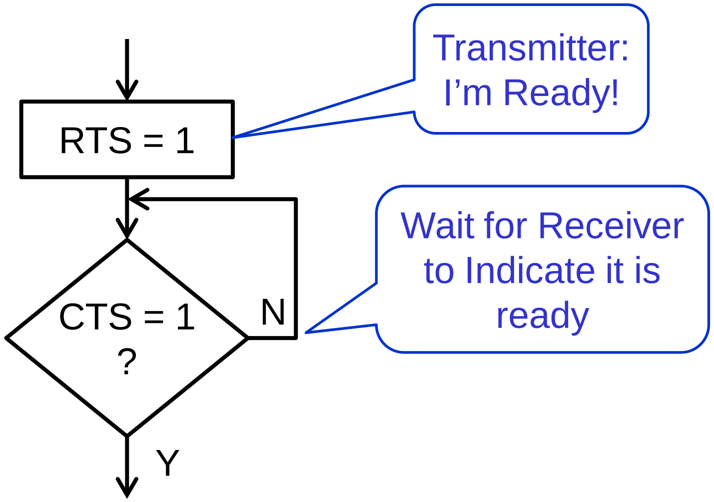
Hardware Handshaking Problem



- Port pins will be floating when no power is applied or when the device has not yet configured the pin as output
- A port pin that is floating may be read and interpreted as either logic 1 or logic 0.
- Therefore, the transmitter may flow past point A even before the receiver's power has switched on

Solution to the Hardware Handshaking Problem





- A solution is to use a pull-down (PD) resistor
  - When the power is off, the pin would be connected to system ground through the PD resistor
  - It would be "electrically" connected to ground even when power is off
- In addition, a check must be added prior to entering the data transmission phase because the true condition of the receiver is CTS = 0, which is the default case when no power is applied
  - The addition is to prepend a Read! Handshake
  - The receiver's power must be on to make the transmitter's CTS=1

```
SDO = ((TSR&0b10000000) != 0);
```

- TSR&0b10000000 performs a bitwise AND of TSR and 0b10000000
- The 8-bit result of TSR&0b10000000 is stored in temp1
- If temp1 is 0b00000000, which means TSR<7>=0, then (temp1 != 0) is false and so SDO = 0;
- If temp1 is 0b10000000, which means TSR<7>=1, then (temp1 != 0) is true and so SDO = 1;

MPLAB Project

Bit-Banging Serial Transmitter

Main.c

```
#include <htc.h>
#include "ProcessorConfiguration.h"
#include "functionPrototypes.h"
#include "definitions.h"

void delay(unsigned char loops) {
    unsigned char i;
    for(i=0; i<loops;i++){}
}

void main(void) {
    unsigned char dataToSend=0, TDR, TSR, numB;

    portInit();

    RTS = 1;        //Transmitter says I'm alive
    while(CTS == 0){} //Wait for Receiver Alive
```

Main.c (Continued)

```
    while(1){
        RTS = 0;           //Make a request to send
        while(CTS == 1){} //Wait for clearance
        TDR = dataToSend;
        TSR = TDR;
        for(numB = 0;numB < 8;numB++){
            SDO = ((TSR&0b10000000) != 0);
            SCK = 1;
            delay(0);
            TSR = TSR << 1;
            SCK = 0;
            delay(0);
        }
        while(CTS == 0){} //Wait for Ack
        RTS = 1;           //Acknowledge
        dataToSend++;
    }
}
```

MPLAB Project

Bit-Banging Serial Transmitter

Defintions.h

```
#define RTS                    RA0
#define CTS                    RA1
#define SDO                    RA2
#define SDI                    RA3
#define SCK                    RA5
```

FunctionPrototypes.h

```
void main(void);
void portInit(void);
void delay(unsigned char);
```

ProcessorConfiguration.h

```
__CONFIG (FOSC_EXTRC & WDTE_OFF & LVP_OFF &
    DEBUG_ON);
// Set configuration bits (See pic16f877a.h)
```

PortInitialization.c

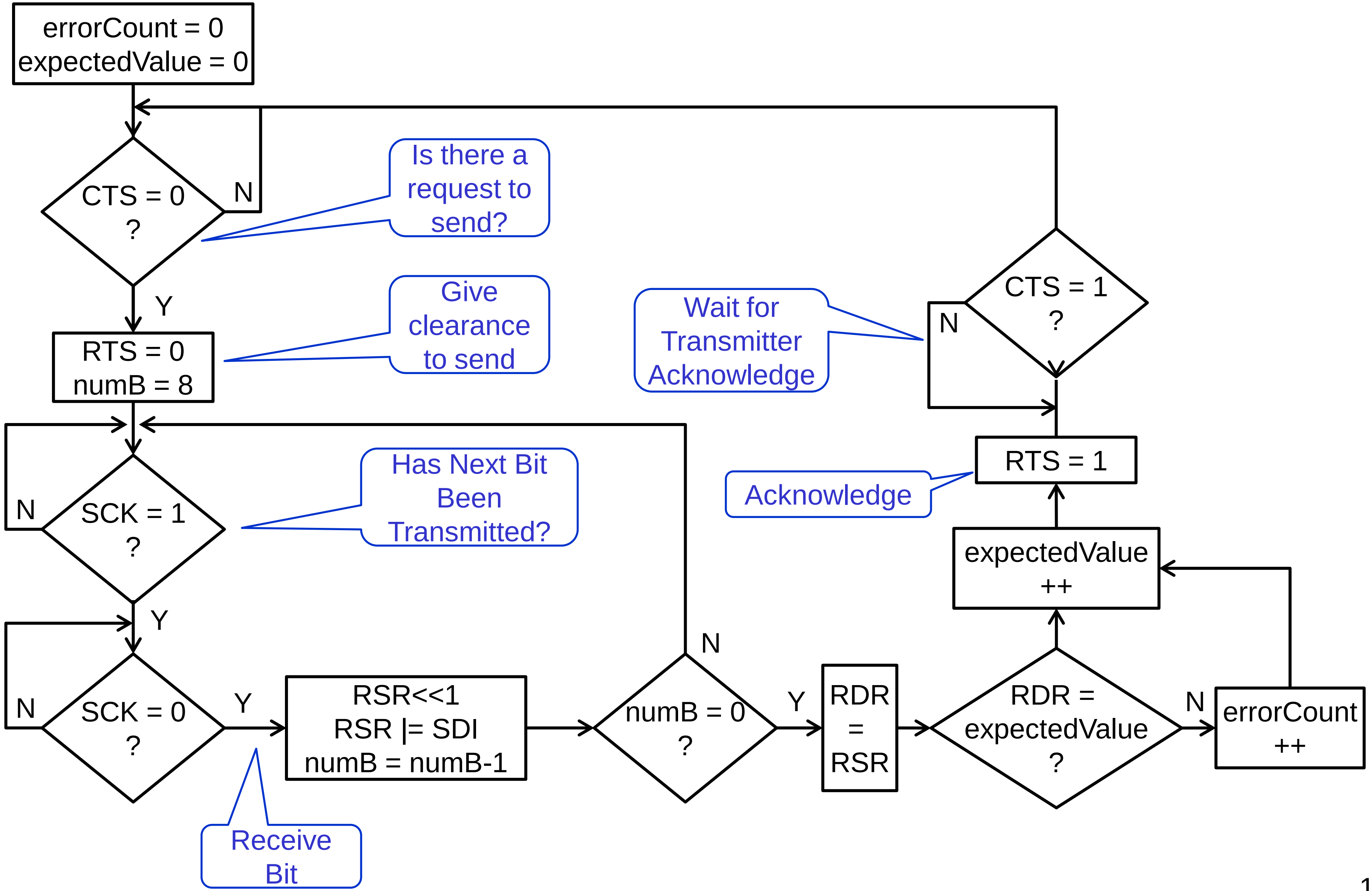
```
#include <htc.h>
#include "definitions.h"

void portInit(void) {

  PCFG3 = 0; // Configure PORTA digital I/O
  PCFG2 = 1;
  PCFG1 = 1;
  TRISA0 = 0; //SDO=A0
  TRISA1 = 1; //SDI=A1
  TRISA2 = 0; //RTS=A2
  TRISA3 = 1; //CTS=A3
  TRISA5 = 0; //SCK=A5
  SCK = 0;    // Initial value of SCK
}
```

Generic Synchronous Serial Communications

Bit-Banging Receiver Flowchart: Verifying Data 0..255



MPLAB Project

Bit-Banging Serial Receiver

Main.c

```
#include <htc.h>
#include "ProcessorConfiguration.h"
#include "functionPrototypes.h"
#include "definitions.h"

void main(void) {
   unsigned char expectedValue=0, errorCnt=0, numB;
   unsigned char RSR = 0, RDR = 0;
   portInit();
```

Main.c (Continued)

```
   RTS = 1;          //Receiver says I'm alive
   while(CTS == 0){}//Wait for Transmitter Alive
   while(1){
      while(CTS == 1){} //Wait for RequestToSend
      RTS = 0;          //Give clearance
      for(numB = 0;numB < 8;numB++){
         while(SCK == 0){} //Wait for SCK=1
         while(SCK == 1){} //Wait for SCK=0
         RSR = RSR << 1;
         RSR |= (bit)SDI;
      }
      RDR = RSR;
      if(RDR!=expectedValue)
         errorCount++;
      expectedValue++;
      RTS = 1;          //Acknowledge
      while(CTS == 0){} //Wait for Ack
   }
}
```

MPLAB Project

Bit-Banging Serial Receiver

Defintions.h

```
#define RTS                  RA0
#define CTS                  RA1
#define SDO                  RA2
#define SDI                  RA3
#define SCK                  RA5
```

FunctionPrototypes.h

```
void main(void);
void portInit(void);
```

ProcessorConfiguration.h

```
__CONFIG (FOSC_EXTRC & WDTE_OFF & LVP_OFF &
    DEBUG_ON);
// Set configuration bits (See pic16f877a.h)
```

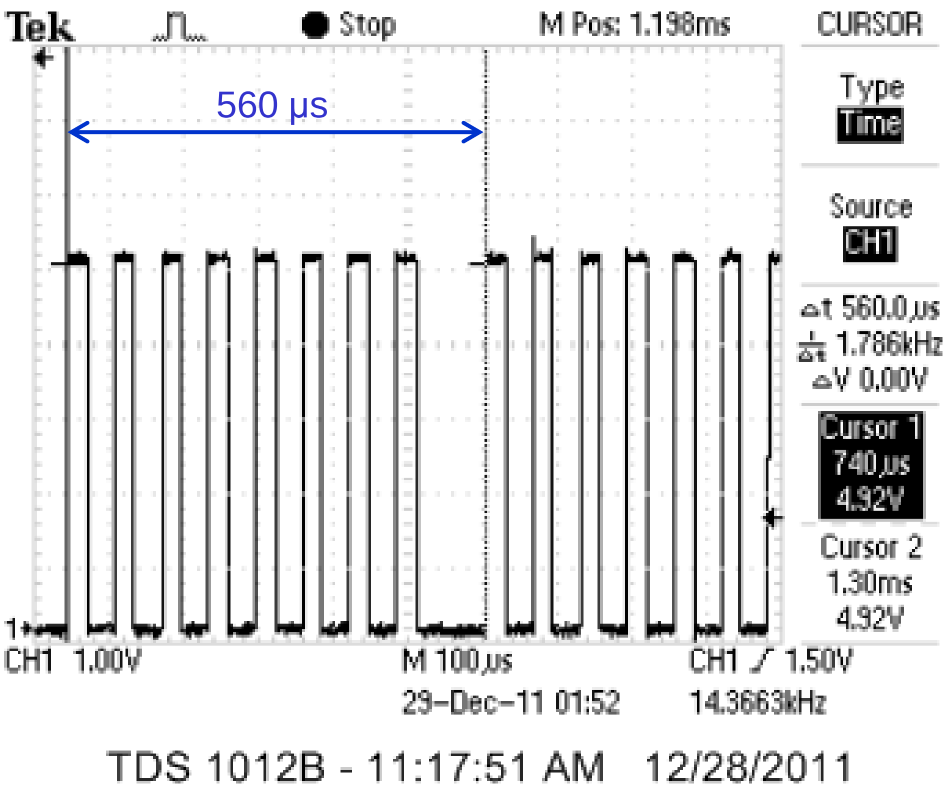
PortInitialization.c

```
#include <htc.h>
void portInit(void) {

  PCFG3 = 0; // Configure PORTA digital I/O
  PCFG2 = 1;
  PCFG1 = 1;
  TRISA0 = 0; //RTS=A0
  TRISA1 = 1; //CTS=A1
  TRISA2 = 0; //SDO=A2
  TRISA3 = 1; //SDI=A3
  TRISA5 = 1; //SCK=A5, input for receiver
}
```

Generic Synchronous Serial Communications

Bit-Banging Results



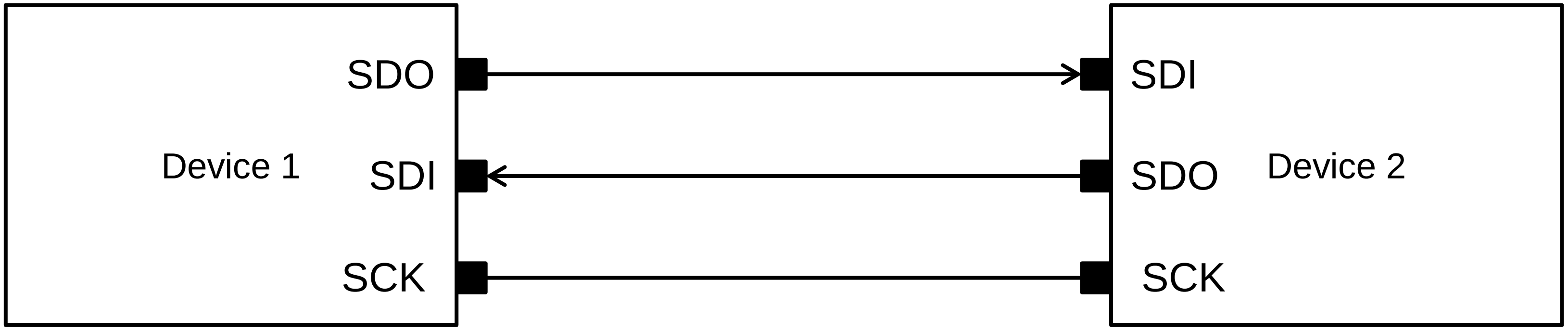
- Transfer rate: 1/(560 µs/8 bits) = 14.3 kbps

Generic Synchronous Serial Communications

Inter-Connection Architecture, Reduced wires

- Two or more devices may be inter-connected using three wires:
  - Serial Data Out (SDO)
  - Serial Data In (SDI)
  - Serial Clock (SCK)

- Unless both devices are ready at the same time, this method is impossible to implement without loosing data



Xon Xoff Protocol

- Instead of hardware handshaking, a software protocol may be designed and implemented to control the transfer of data
  - Xon - Xoff Protocol
    - Transmitter continually transmits the code for ENQ (0x05), meaning that is requesting to send
    - Transmitter waits for an acknowledgment from the receiver ACK (0x06)
    - When the transmitter receives and ACK from the receiver, the transmitter thane waits for Xon (0x11) from the receiver, which means the receiver is ready to receive data.
    - If at any time the receiver want to pause the transmission, receiver send Xoff (0x13)
- Problem – Xon and Xoff codes cannot appear in the data transmission, otherwise they will be interpreted to pause transmission (Xoff), for example
    - Solution, encode the data in larger number of bits so that Xon and Xoff have code that will never appear in the data.
    - For example, encode all data as 0 DDDD DDDD, and encode control codes as 1 CCCC CCCC