Introduction to I2C & SPI

Chapter 22

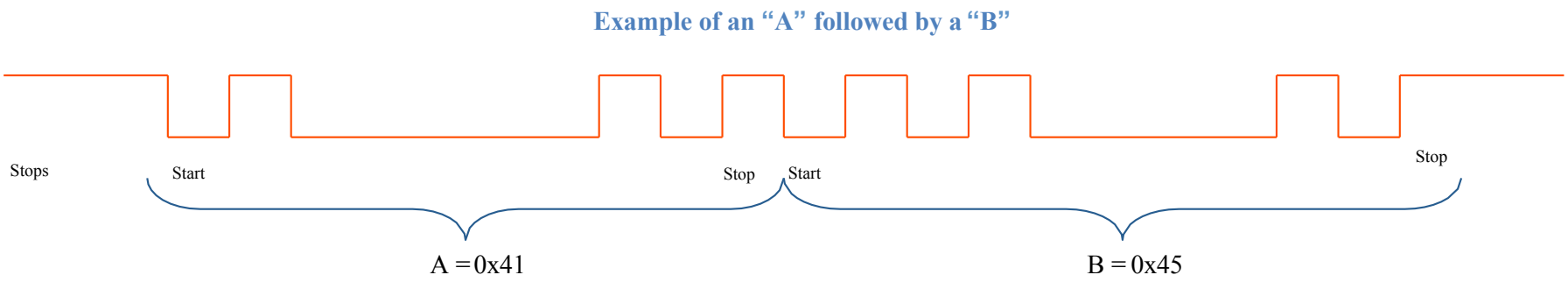
Issues with Asynch. Communication Protocols

Asynchronous Communications

- Devices must agree ahead of time on a data rate
- The two devices must also have clocks that are close to the same rate
- Excessive differences between clock rates on either end will cause garbled data
- Asynchronous serial ports require hardware overhead
- The UART at either end is relatively complex and difficult to accurately implement in software if necessary
- Most UART devices only support a certain set of fixed baud rates, and the highest of these is usually around 230400 bits per second

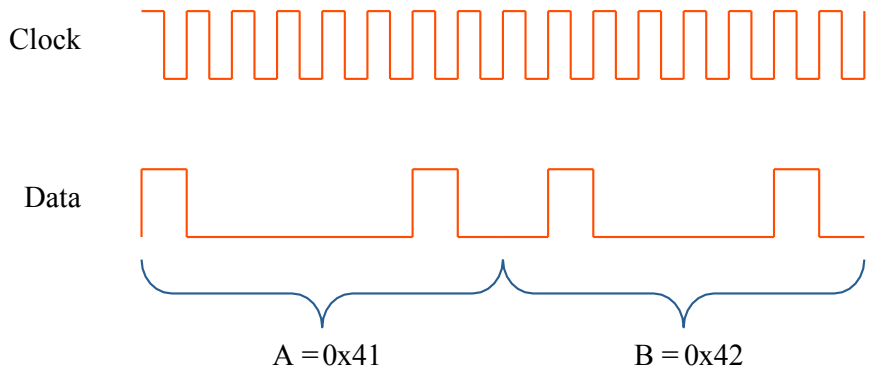
https://learn.sparkfun.com/tutorials/i2c?_ga=1.229206105.432923716.1447312656

Asynchronous Transmission



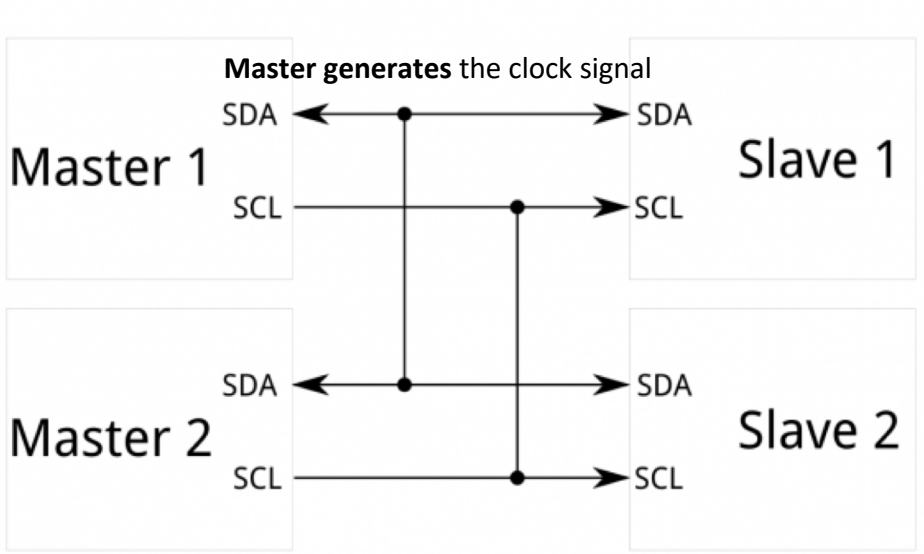
Synchronous Transmission

Example of an "A" followed by a "B"



The Inter-integrated Circuit (I^2C)

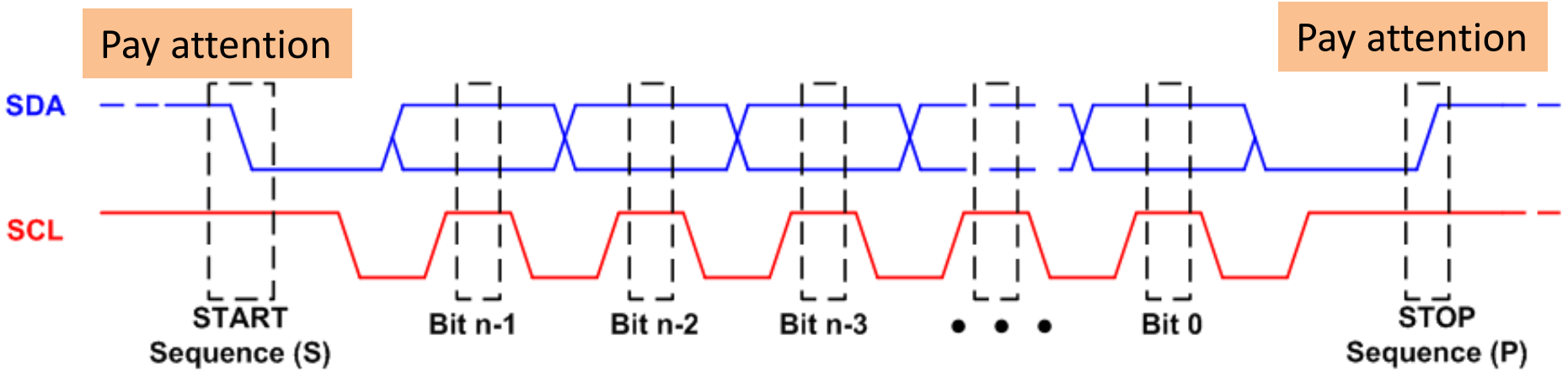
- The Inter-integrated Circuit (I^2C) Protocol is a protocol intended to allow multiple “slave” (or secondary) digital integrated circuits (“chips”) to communicate with one or more “master” chips.
- Multi-master system, allowing more than one master ” (or primary) to communicate with all devices on the bus
- When multiple primary devices are used, the master devices can’t talk to each other over the bus and must take turns using the bus lines.
- In I2C there are **three additional** modes specified: fast-mode plus, at 1MHz; high-speed mode, at 3.4MHz; and ultra-fast mode, at 5MHz.



Characteristics

- Serial, byte-oriented
- Multi-master, multi-slave
- Two bidirectional open-drain lines, plus ground
  - Serial Data Line (SDA)
  - Serial Clock Line (SCL)
  - SDA and SCL need to pull up with resistors

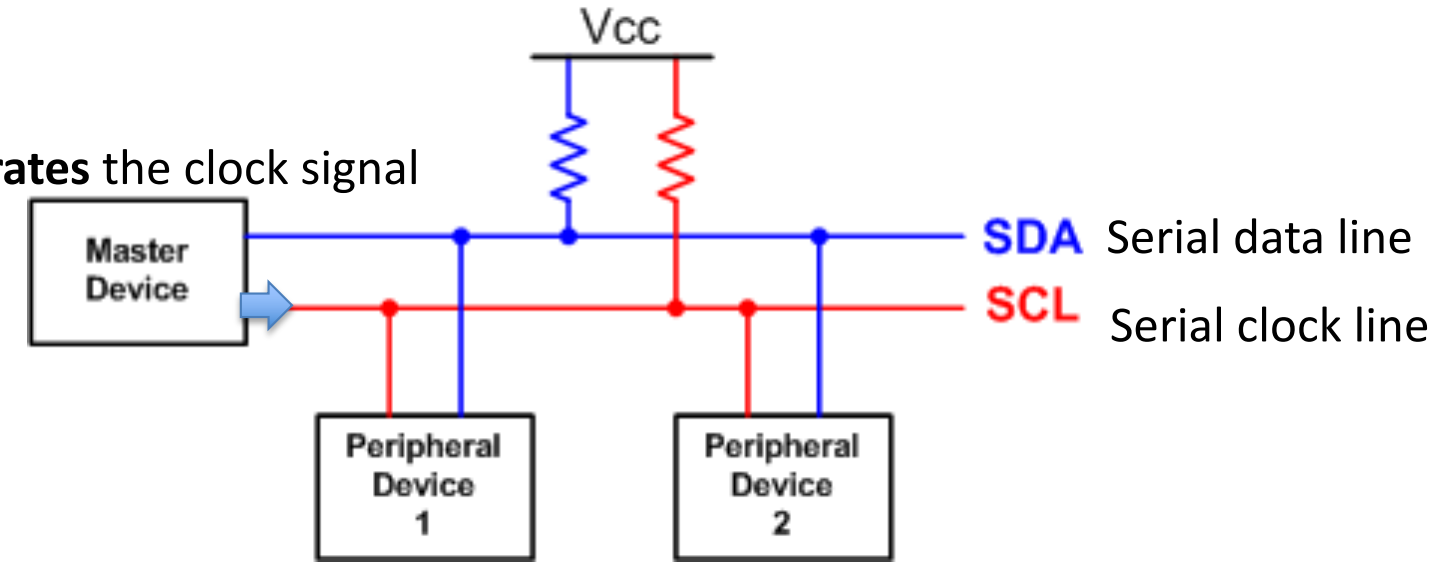
Timing Diagram



- A **START** condition is a high-to-low transition on SDA when SCL is high.
- A **STOP** condition is a low to high transition on SDA when SCL is high.
- The address and the data bytes are sent most significant bit first.
- **Master generates** the clock signal and sends it to the slave during data transfer

Inter-Integrated Circuit (I2C)

Master generates the clock signal

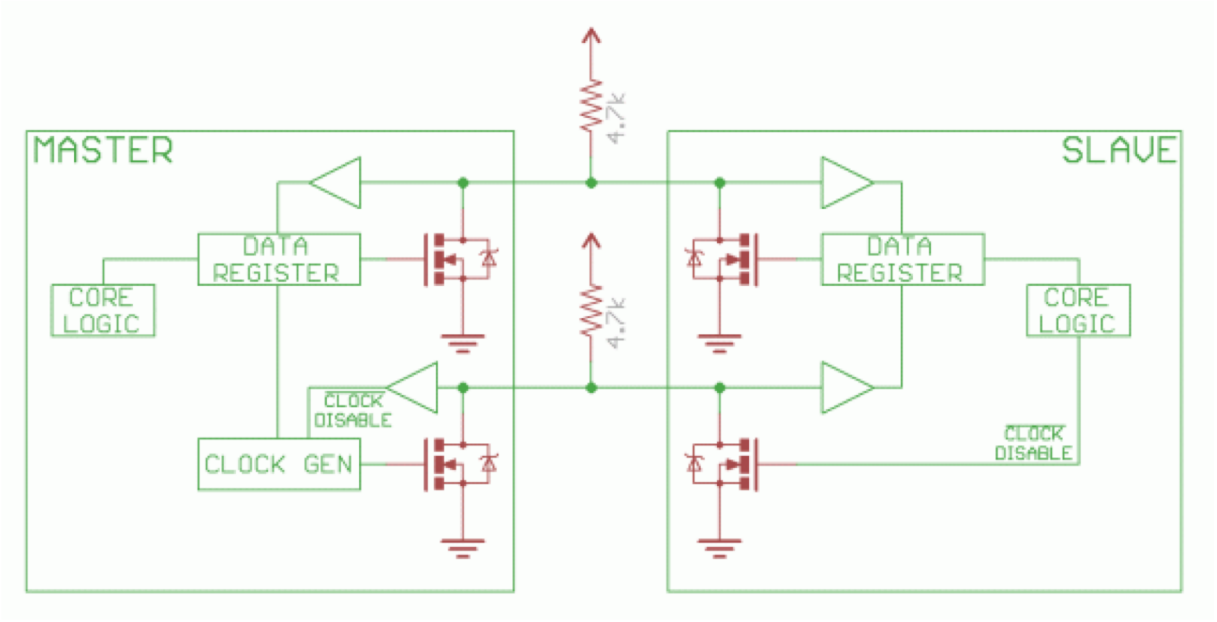


- SDA and SCL have to be **open-drain**
  - Connected to positive if the output is 1
  - In high impedance state if the output is 0
- Each Device has an unique address (7, 10 or 16 bits). Address 0 used for broadcast
- STM32 internal pull-up is too weak (internal 100KΩ)
- External pull-up (4.7 kΩ for low speed, 3 kΩ for standard mode, and 1 kΩ for fast mode).
  - Fast mode refers to fast rise time!

Inter-Integrated Circuit (I2C)

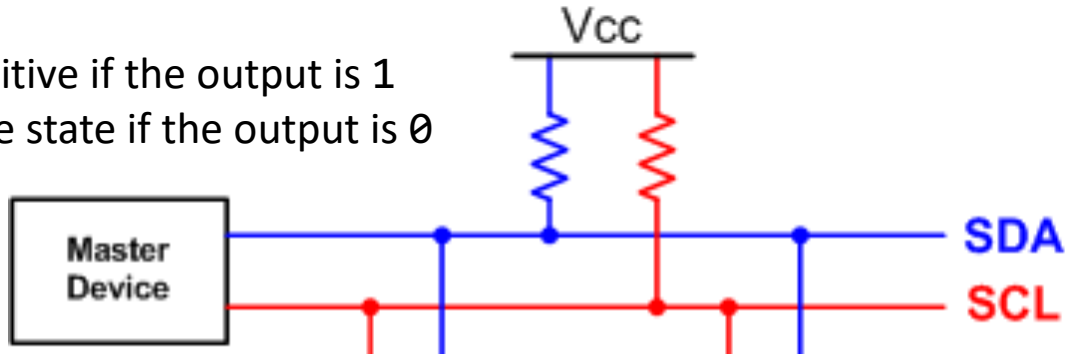
- The I^2C bus drivers are OPEN DRAIN meaning that they can pull the corresponding signal line **low, but cannot drive it high**
- There can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system
- STM32 internal pull-up is too weak (internal 100KΩ)
- External pull-up (4.7 kΩ for low speed, 3 kΩ for standard mode, and 1 kΩ for fast mode – fast rise time!)

"Wired-AND" bus: A sender can pull the lines to low, even if other senders are trying to drive the lines to high



Multiple Masters

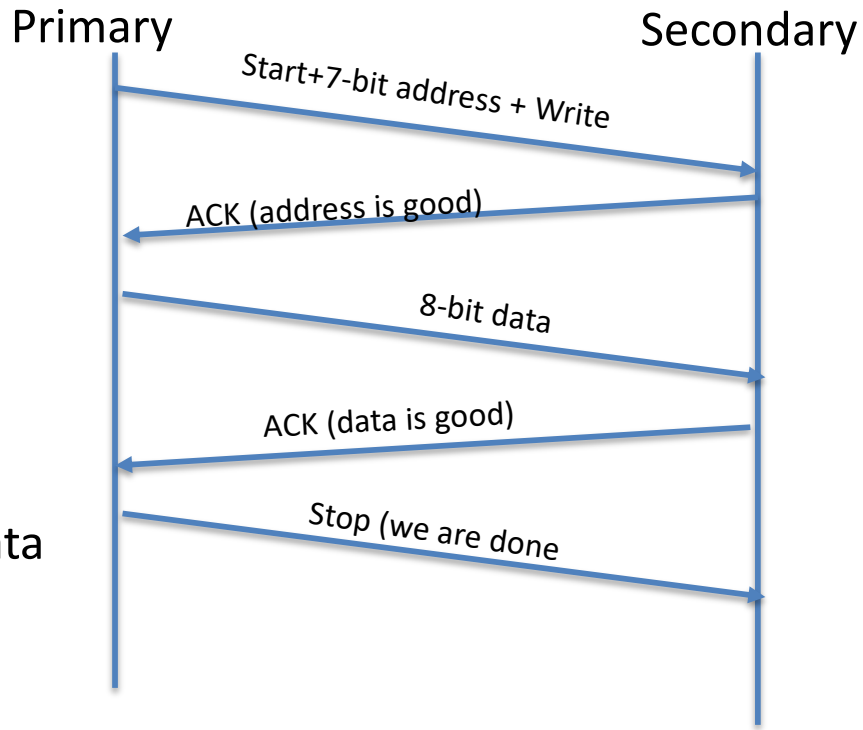
Connected to positive if the output is 1
In high impedance state if the output is 0



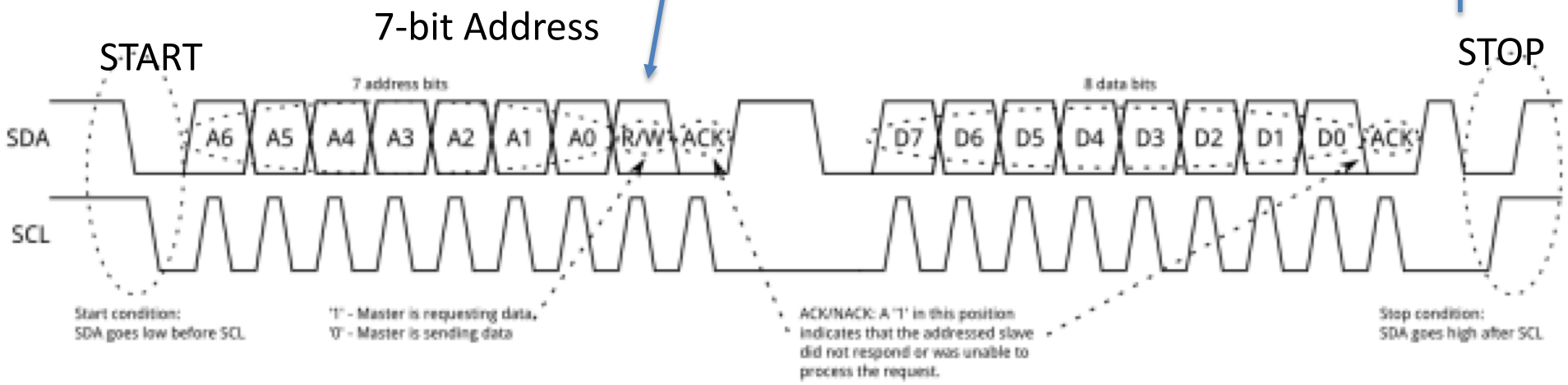
- In single master systems, arbitration is not needed.
- **Arbitration** for multiple masters:
  - During data transfer, the master constantly checks whether the SDA voltage level matches what it has sent.
  - When two masters generate a START setting **concurrently**, the first master which detects SDA low while it has actually intended to set SDA high will **lose the arbitration** and let the other master complete the data transfer.

Basic Protocol (7-bit Addressing)

- Setting the address
- Write the data value
- R/W- : 0 --> Primary sends data

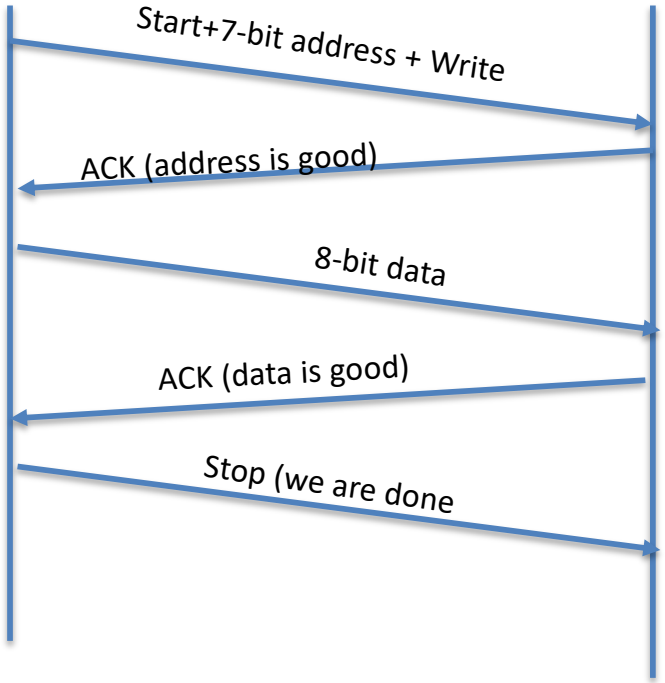


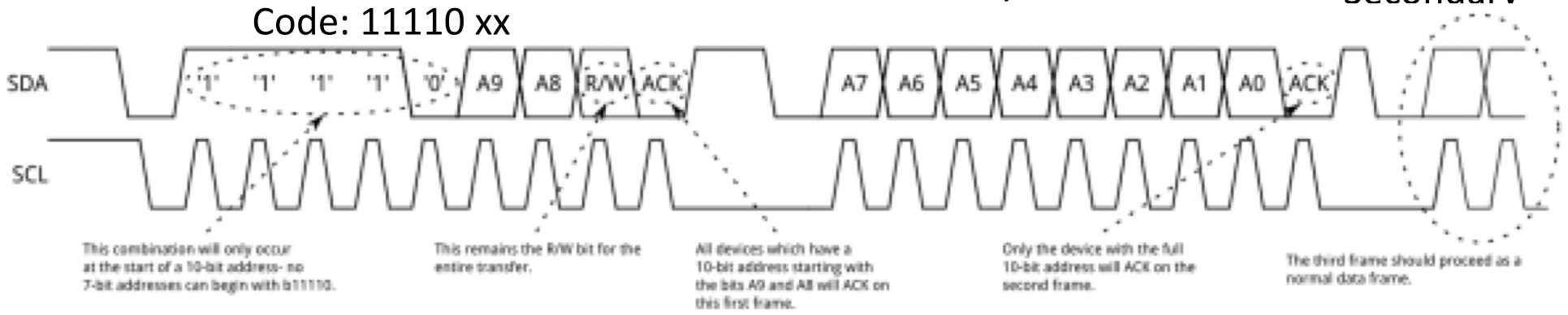
1= Master requires data
0=Master sends data



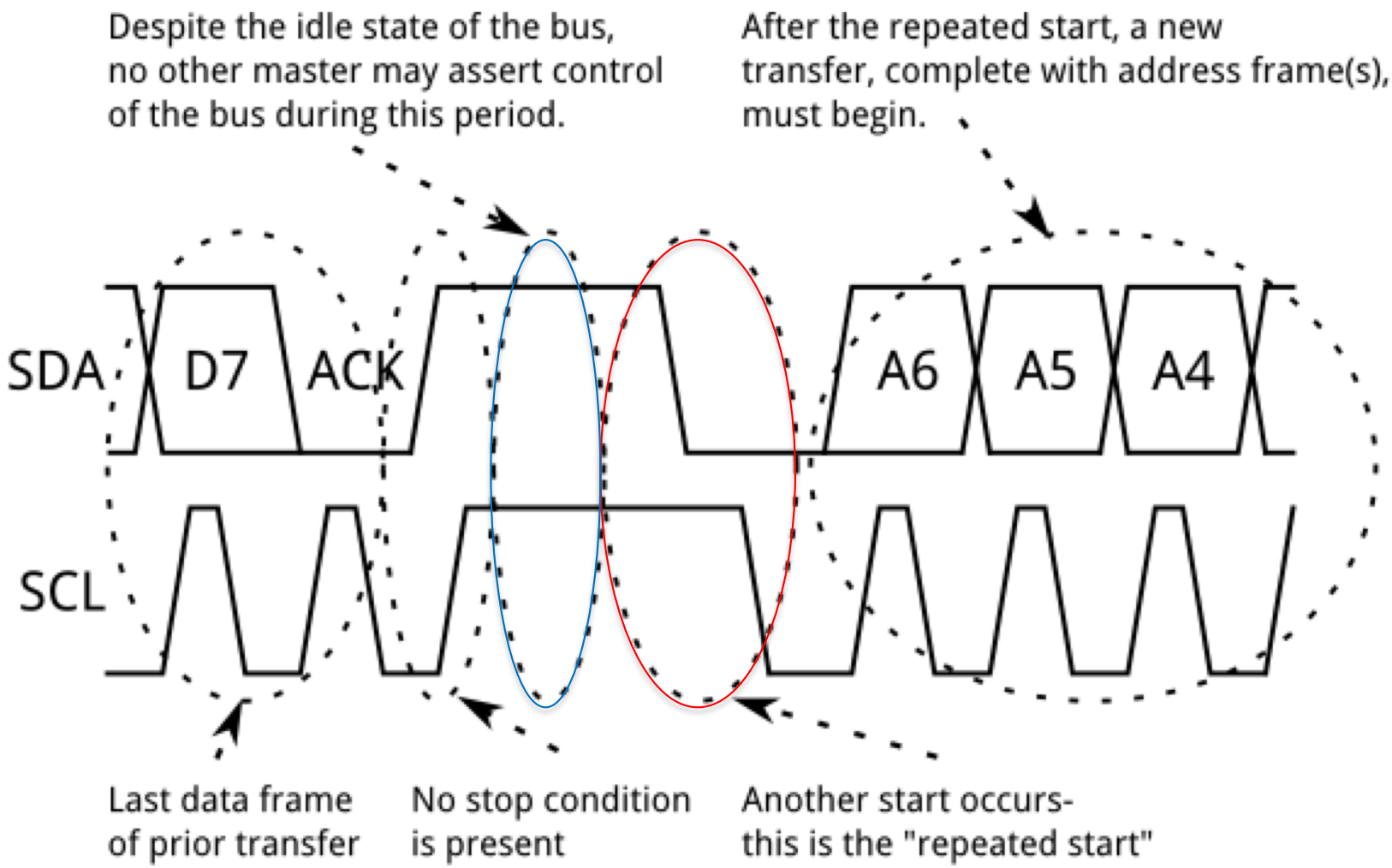
Basic Protocol (10-bit Addressing)

- Setting the address
- Write the data value
- R/W- : 0 --> Primary sends data

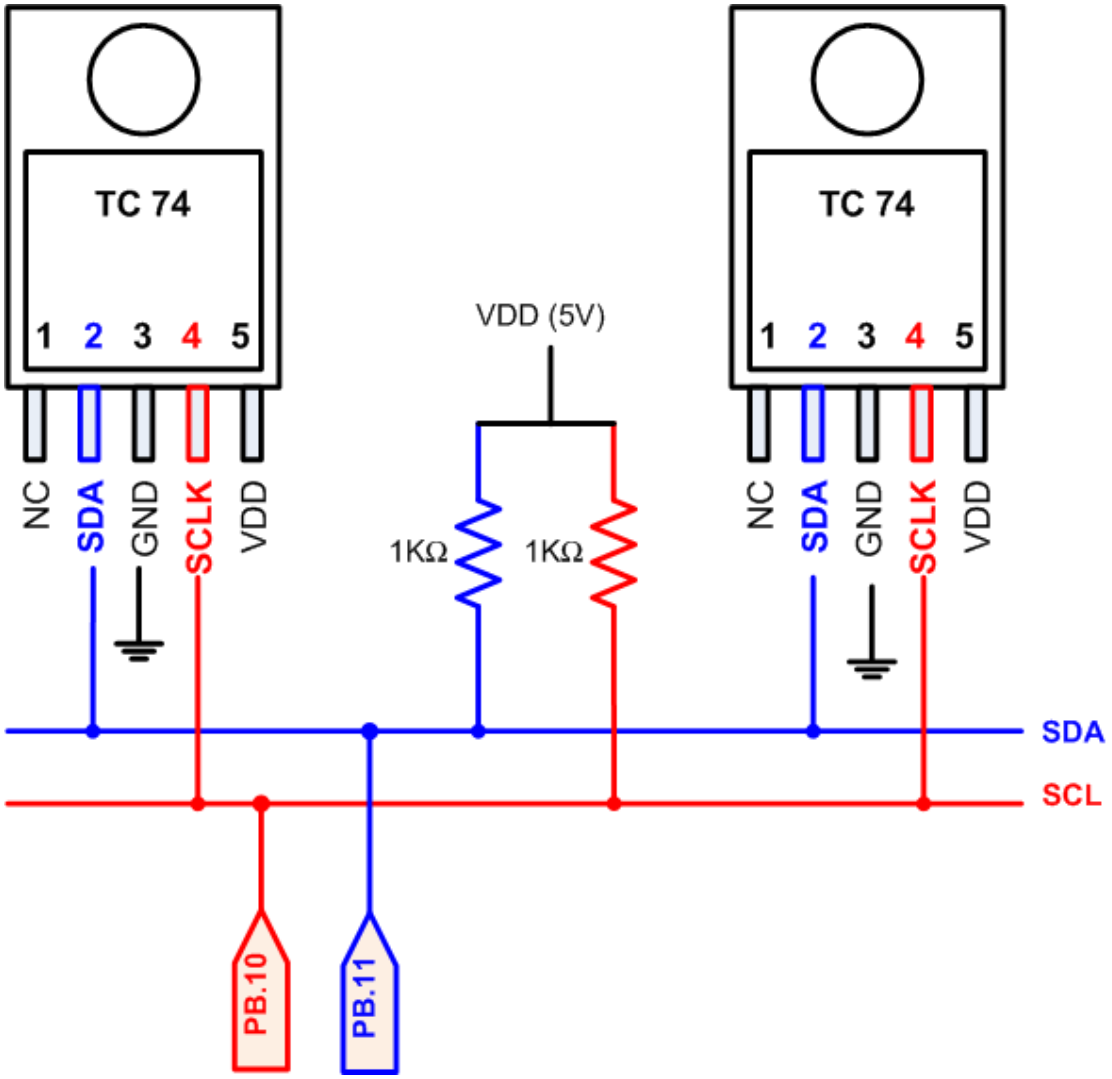




Repeated Starts

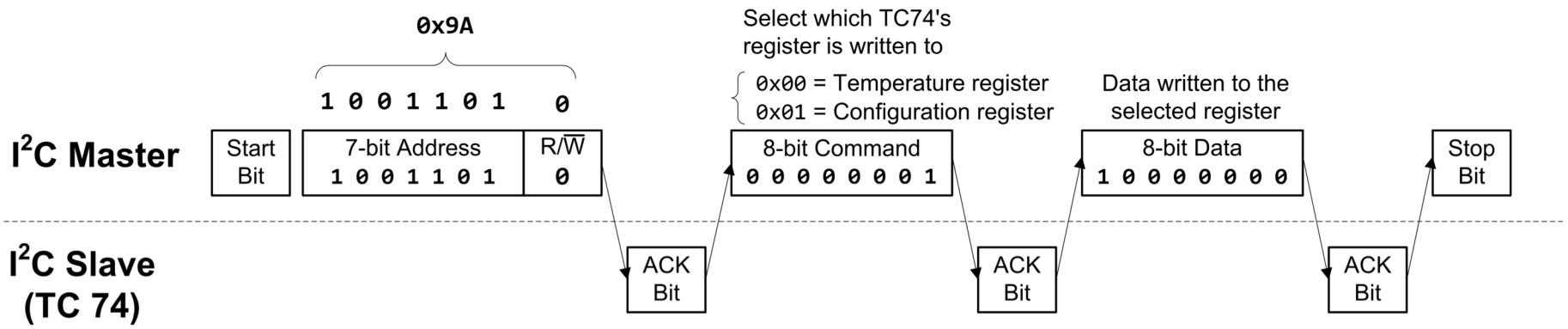


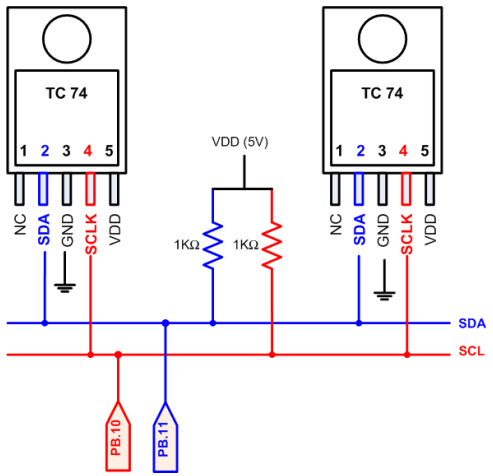
Interfacing Serial Digital Thermal Sensor



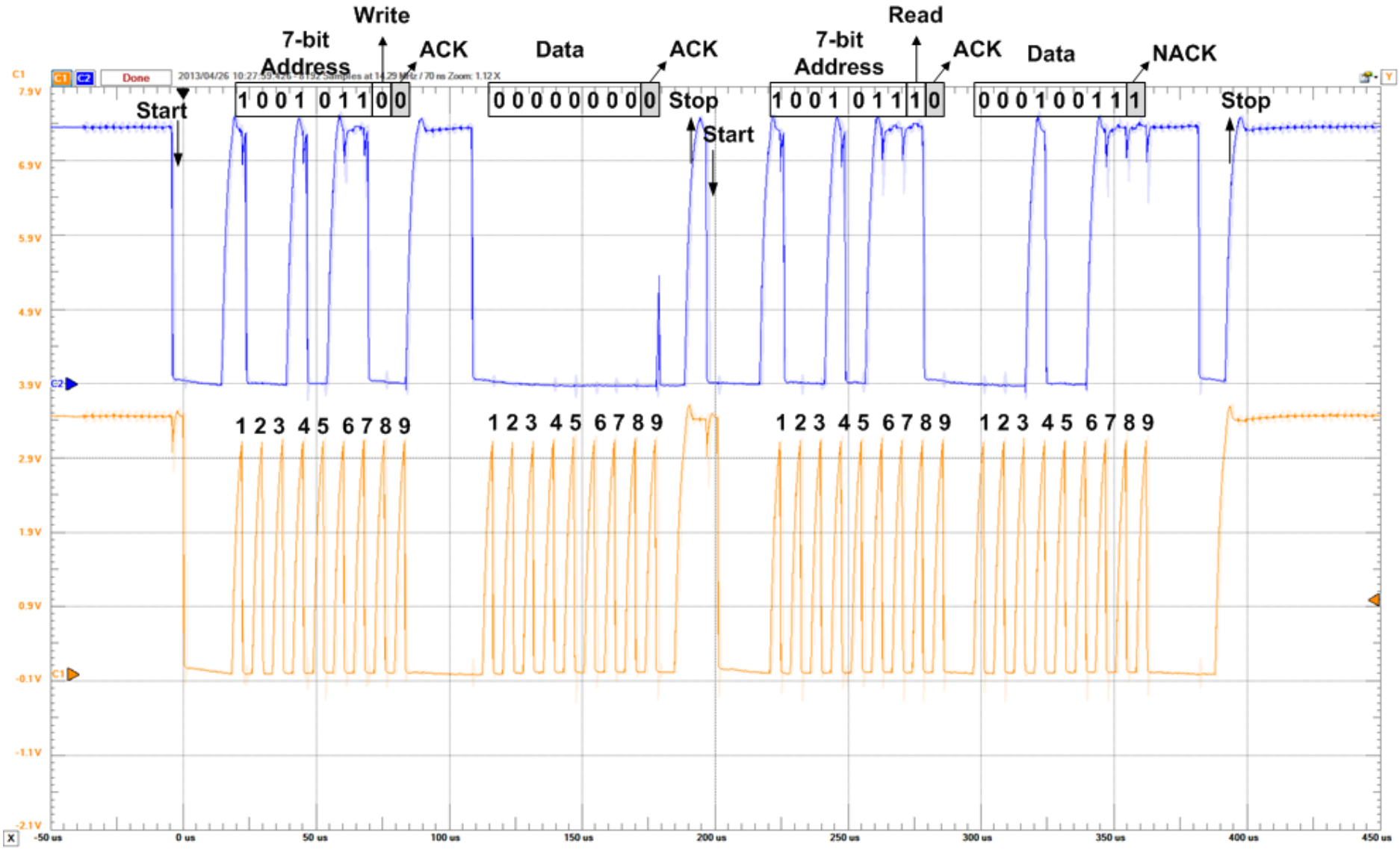
Communicating with TC74

with an address of 0x4D

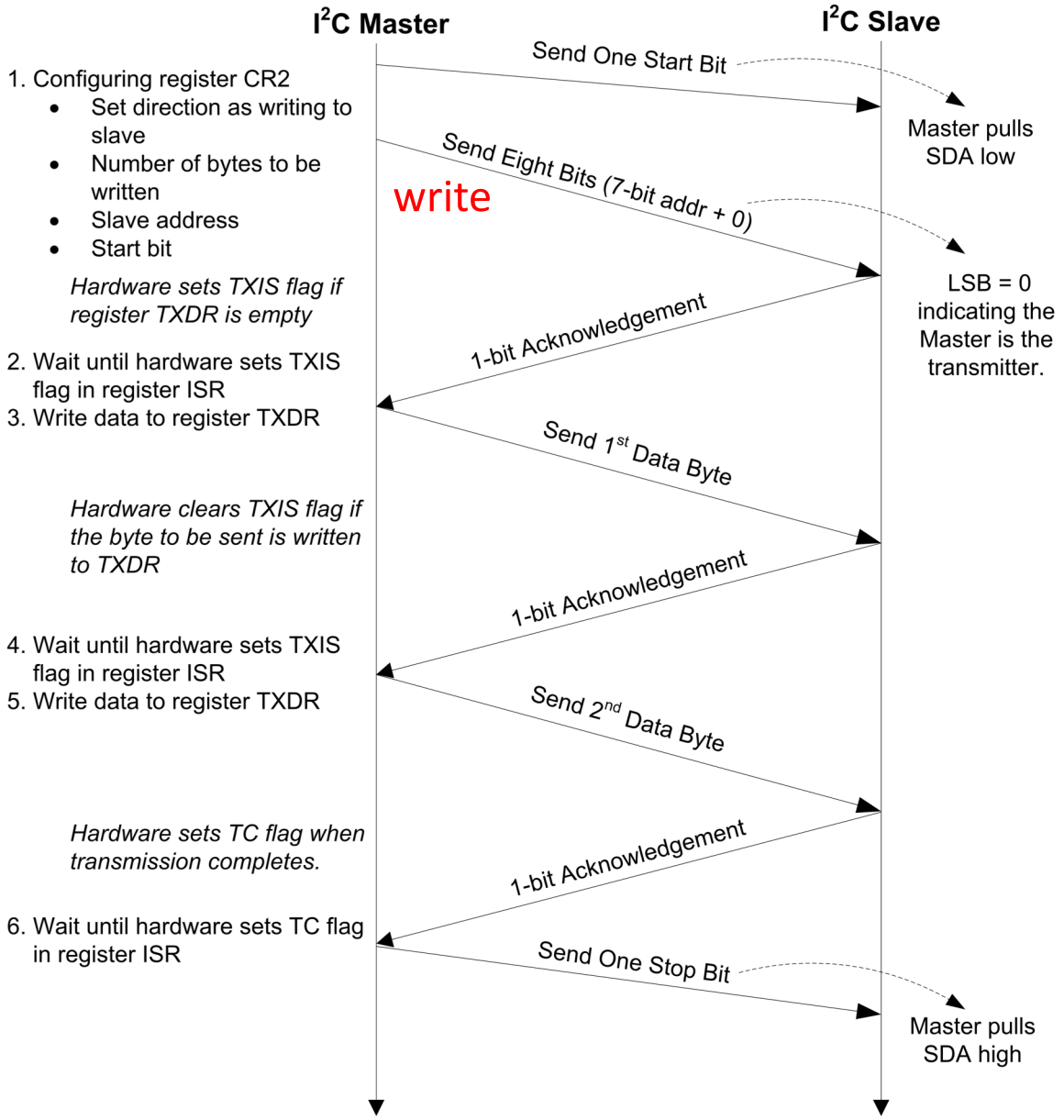




I2C Data


Start
7-bit Address
Write
ACK
Data
ACK
Stop
Start
7-bit Address
Read
ACK
Data
NACK
Stop
1 0 0 1 0 1 1 0 0
0 0 0 0 0 0 0 0 0
1 0 0 1 0 1 1 1 0
0 0 0 1 0 0 1 1 1
1 2 3 4 5 6 7 8 9
7.9V
6.9V
5.9V
4.9V
3.9V
2.9V
1.9V
0.9V
-0.1V
-1.1V
-2.1V
-50 us
0 us
50 us
100 us
150 us
200 us
250 us
300 us
350 us
400 us
450 us

Sending Data to I2C Secondary Via Polling



I2C Initialization

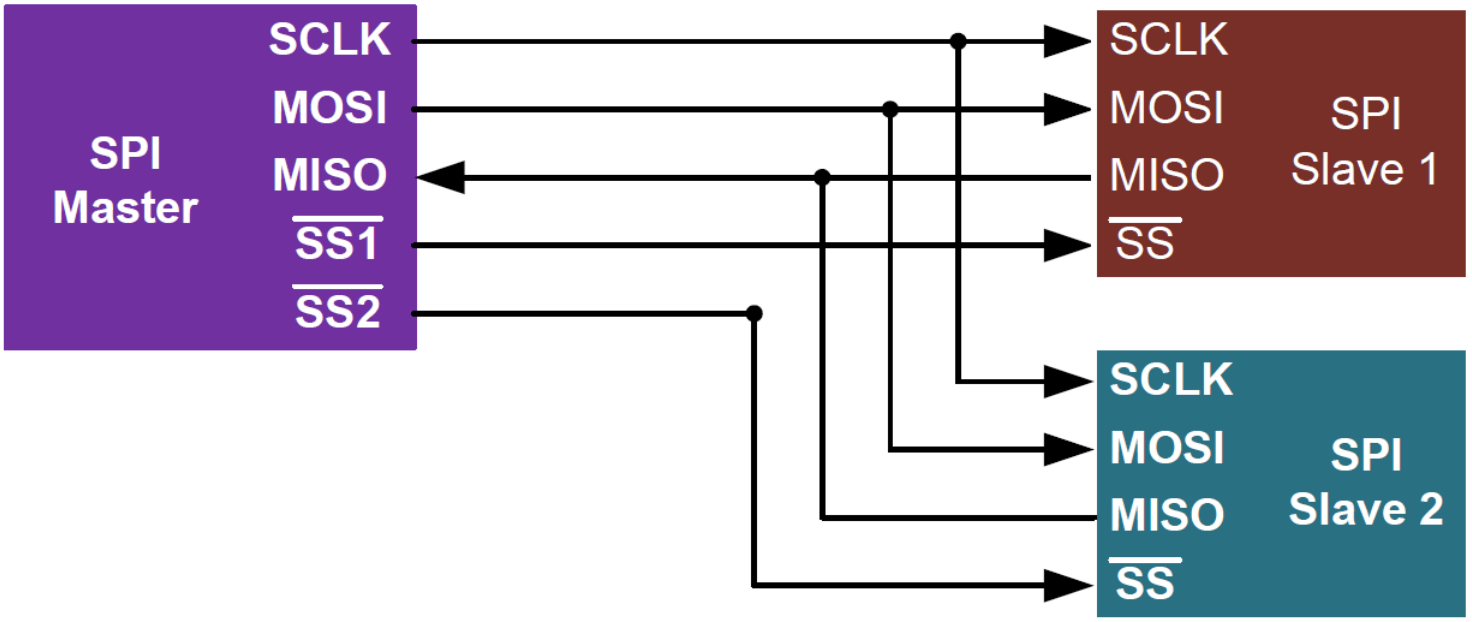
```
void I2C_Initialization(void){
  // Enable the clock of I2C
  RCC->APB1ENR  |= RCC_APB1ENR_I2C2EN;        // I2C 2 clock enable

  // Reset the I2C (set and reset by software)
  RCC->APB1RSTR |= RCC_APB1RSTR_I2C2RST;
  RCC->APB1RSTR &= ~RCC_APB1RSTR_I2C2RST;

  // I2C Control register.  ACK is set and cleared by software, and cleared by hardware when PE=0.
  I2C2->CR1 |= I2C_CR1_ACK;      // Acknowledge after a byte is received
  I2C2->CR1 &= ~I2C_CR1_SMBUS;   // SMBus Mode: 0 = I2C mode;  1 = SMBus mode
  I2C2->CR2 &= ~I2C_CR2_FREQ;    // Clear Peripheral CLK frequency FREQ[5:0]
  I2C2->CR2 |= I2C_CR2_FREQ_1;   // Set Peripheral clock frequency as 2 MHz
  I2C2->CR2 |= I2C_CR2_ITEVTEN;  // Event Interrupt Enable
  I2C2->CR1 &= ~I2C_CR1_PE;      // Disable I2C to configure TRISE
  I2C2->TRISE &= ~I2C_TRISE_TRISE;     // Clear Maximum Rise Time
  I2C2->TRISE |= 17;                   // Set Maximum Rise Time for standard mode

  // I2C own address registers
  // Before the STM32 sends its start sequence it sits listening  to the I2C lines waiting for its address
  // This is helpful if STM32 is used as slave
  I2C2->OAR1 %= ~(0x01);

  I2C2->OAR1 = 0x00;   // Set up the address byte to select the slave device
  I2C2->OAR2 = 0x00;   // Set up the I2C own address2

  // I2C Clock control register
  I2C2->CCR &= ~I2C_CCR_FS;   // 0 = Standard Mode ((up to 100 kHz));
                              // 1 = Fast Mode (up to 400 kHz)
  I2C2->CCR &= ~I2C_CCR_CCR;
  I2C2->CCR |= 0x08;
  I2C2->CR1 |= I2C_CR1_PE;   // Enable I2C1
}
```

A Brief Introduction to SPI

SPI: Serial Peripheral Interface

Serial Peripheral Interface (SPI)

- Synchronous full-duplex communication
- Can have multiple slave (secondary) devices
- No flow control or acknowledgment
- Slave (secondary) cannot communicate with slave directly.



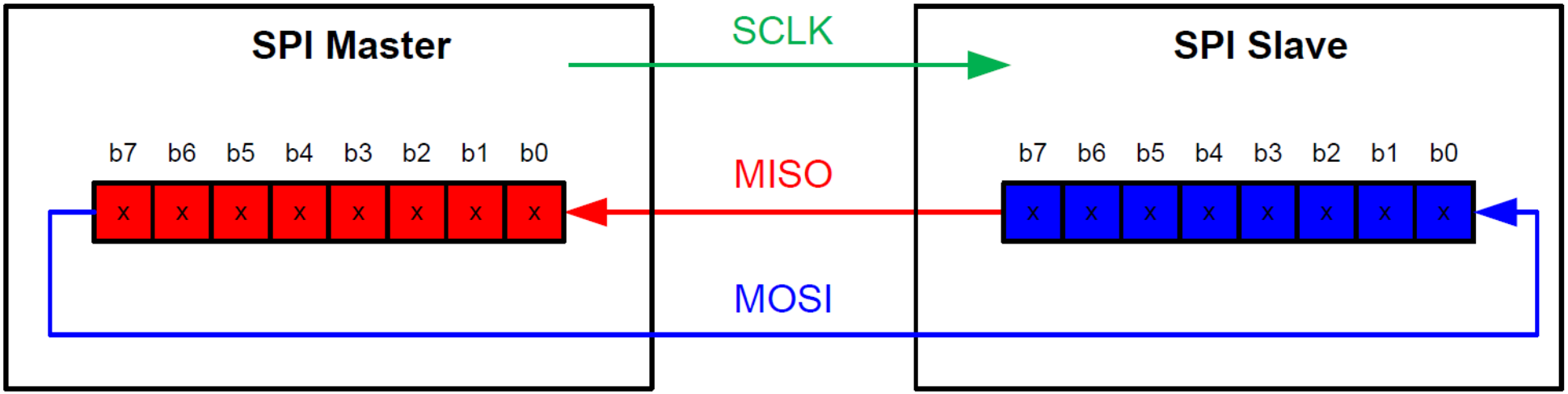
SCLK: serial clock

MOSI: master out slave in

SS: slave select (active low)

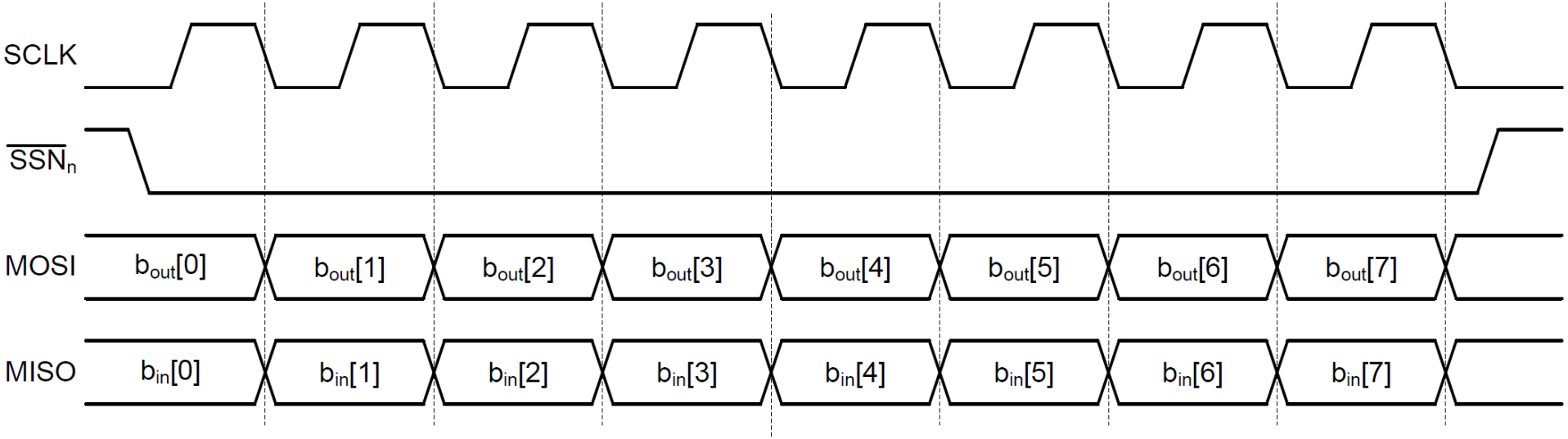
MISO: master in slave out

Data Exchange

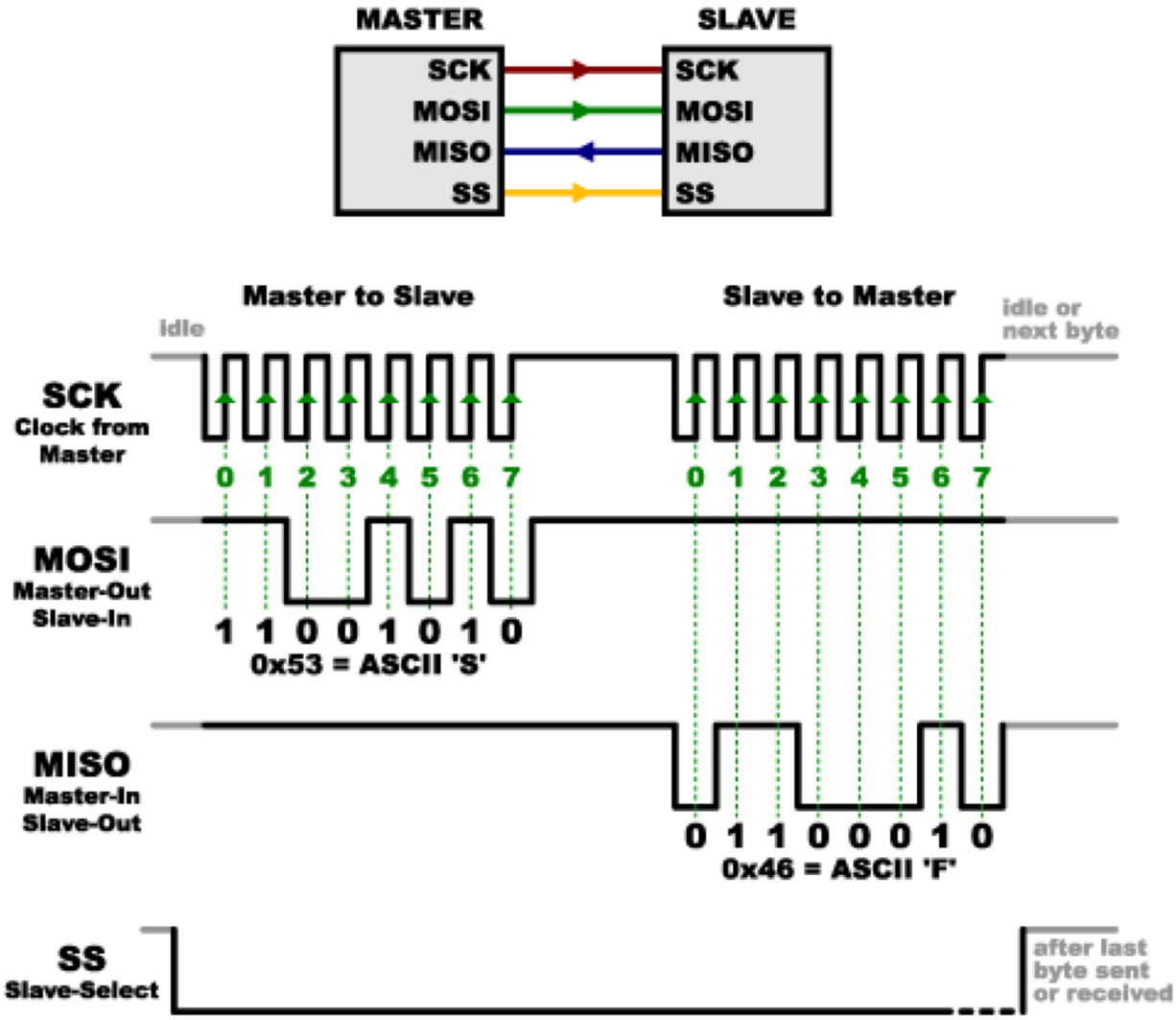


- Master has to provide clock to slave
- Synchronous exchange: for each clock pulse, a bit is shifted out and another bit is shifted in at the same time. This process stops when all bits are swapped.
- Only master can start the data transfer

Clock


SCLK
$\overline{SSN_n}$
MOSI
$b_{out}[0]$
$b_{out}[1]$
$b_{out}[2]$
$b_{out}[3]$
$b_{out}[4]$
$b_{out}[5]$
$b_{out}[6]$
$b_{out}[7]$
MISO
$b_{in}[0]$
$b_{in}[1]$
$b_{in}[2]$
$b_{in}[3]$
$b_{in}[4]$
$b_{in}[5]$
$b_{in}[6]$
$b_{in}[7]$

Read and Write



Multiple Chip Selects