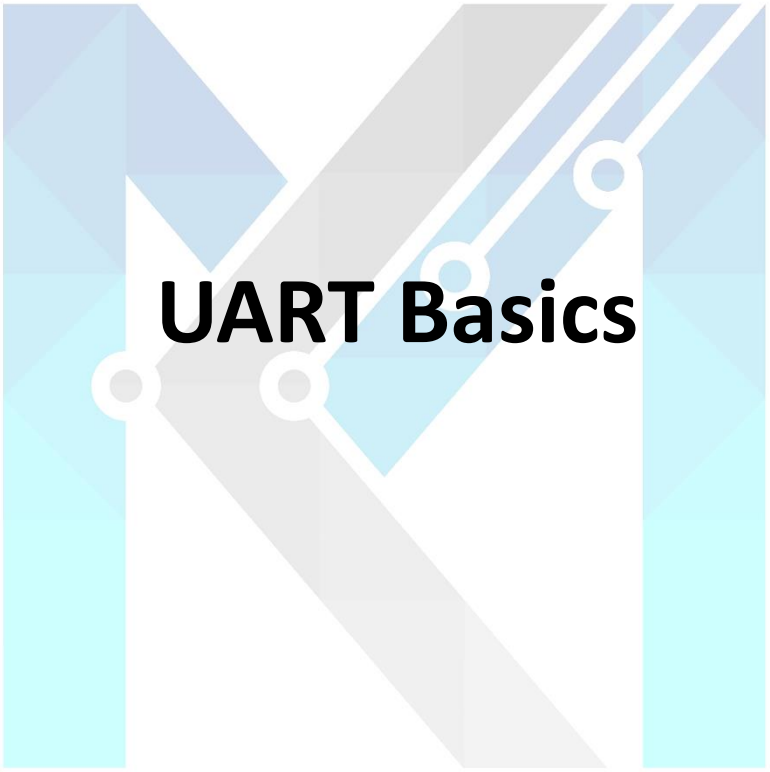
Serial Communication Protocols

Communication Protocols

- **Different communication methods:**
  - Microcontroller to microcontroller
  - Microcontroller to peripherals
- **Types of Communication protocols:**
  - Parallel communication protocols.
  - Serial Communication protocols.

Serial Communication Protocols

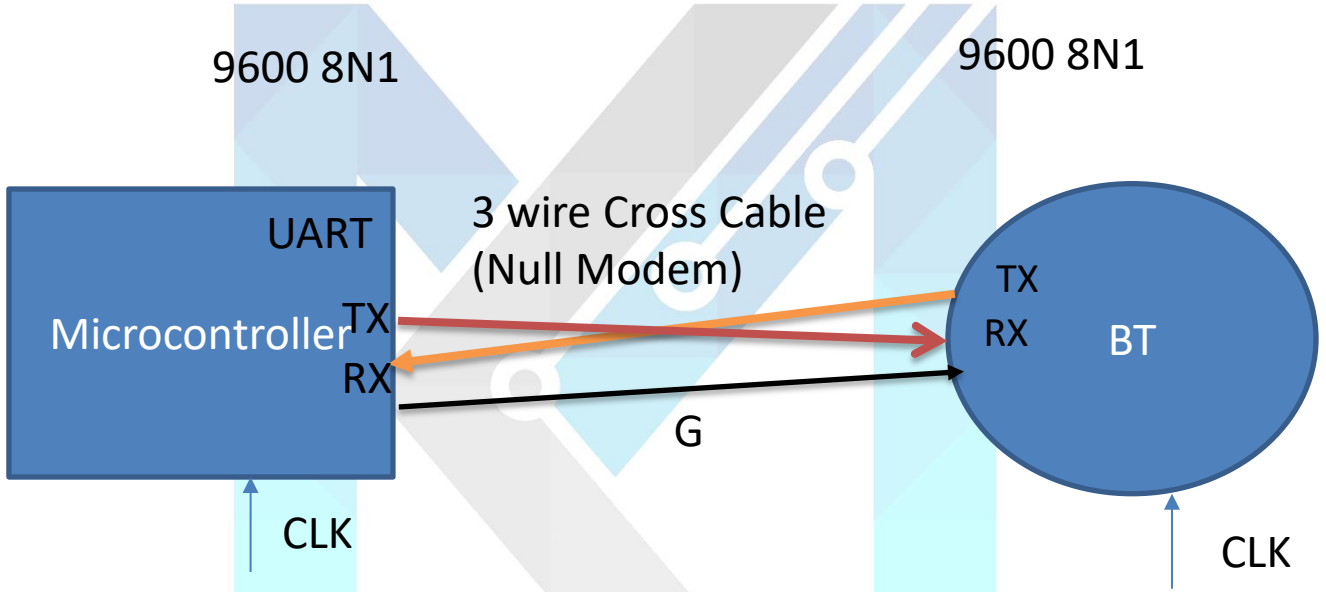
- UART
- I2C
- SPI
- CAN
- RS485



UART Basics

KERNEL MASTERS

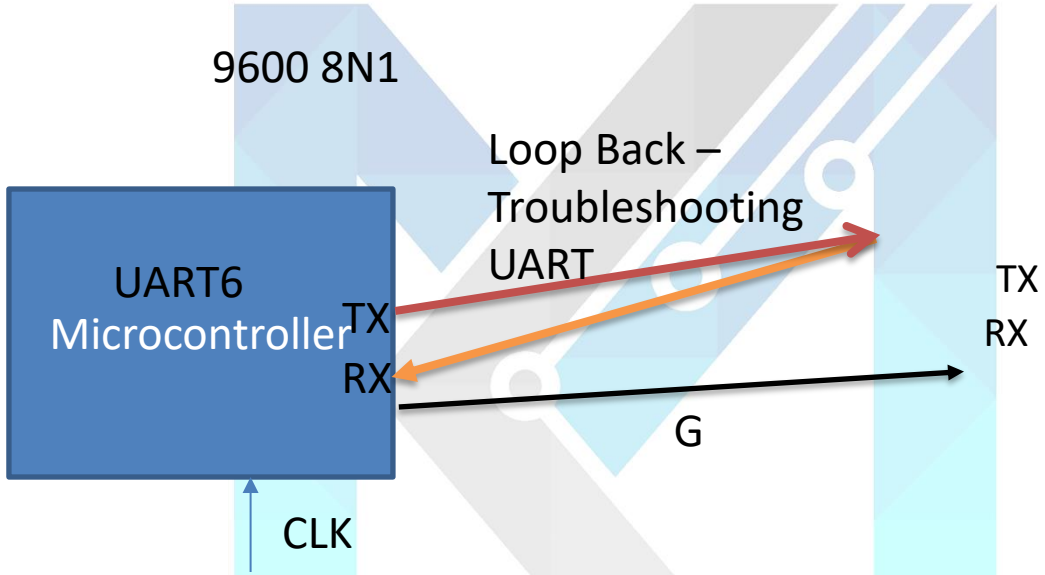
Interfacing peripherals



One to one (Peer to peer communication protocol
3 Wire Serial ASync Comm. Protocol

Word length: 5 bits to 8 bits
Parity:
Start bit
Stop bit
Programmable baud rate (bits/sec) generator

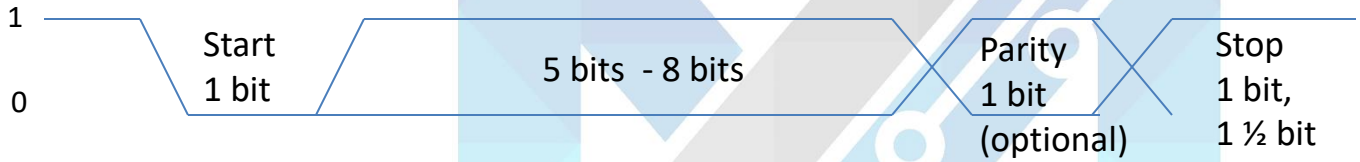
Interfacing peripherals



One to one (Peer to peer communication protocol
3 Wire Serial ASync Comm. Protocol

Word length: 5 bits to 8 bits
Parity:
Start bit
Stop bit
Programmable baud rate (bits/sec) generator

UART Communication protocol format

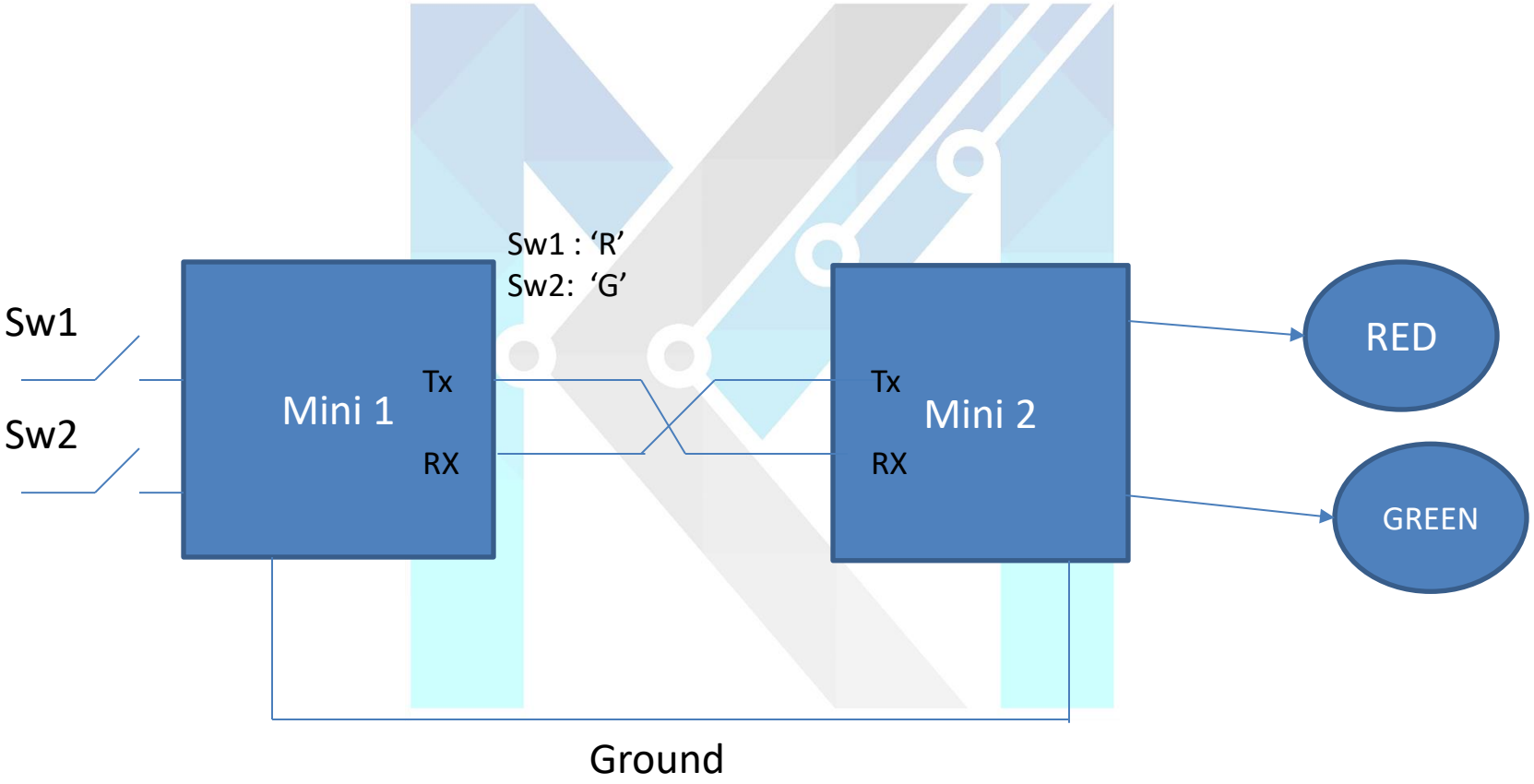


Examples:

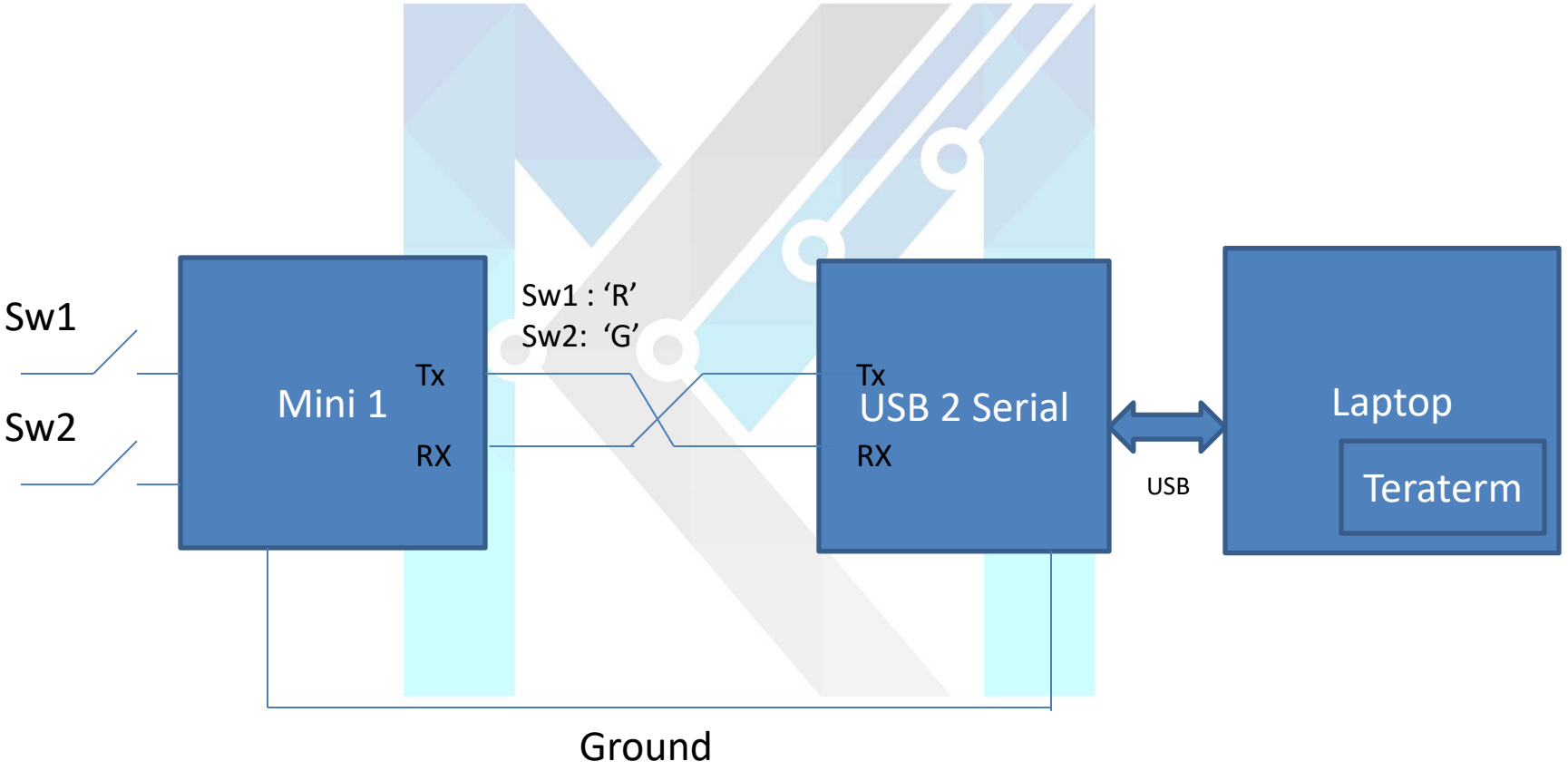
1. **9600(baud rate) 8(word length) N(No parity) 1(stop bits)**
   No. of bits = 1+8+0+1 = 10 bits

1. **115200 5 E 2**
   No. of bits = 1+5+1+2=9 bits
2. 34600 7 O 1
   No. of bits = 1+7+1+1=10 bits

Comm. b/w two boards using UART6 comm. protocol



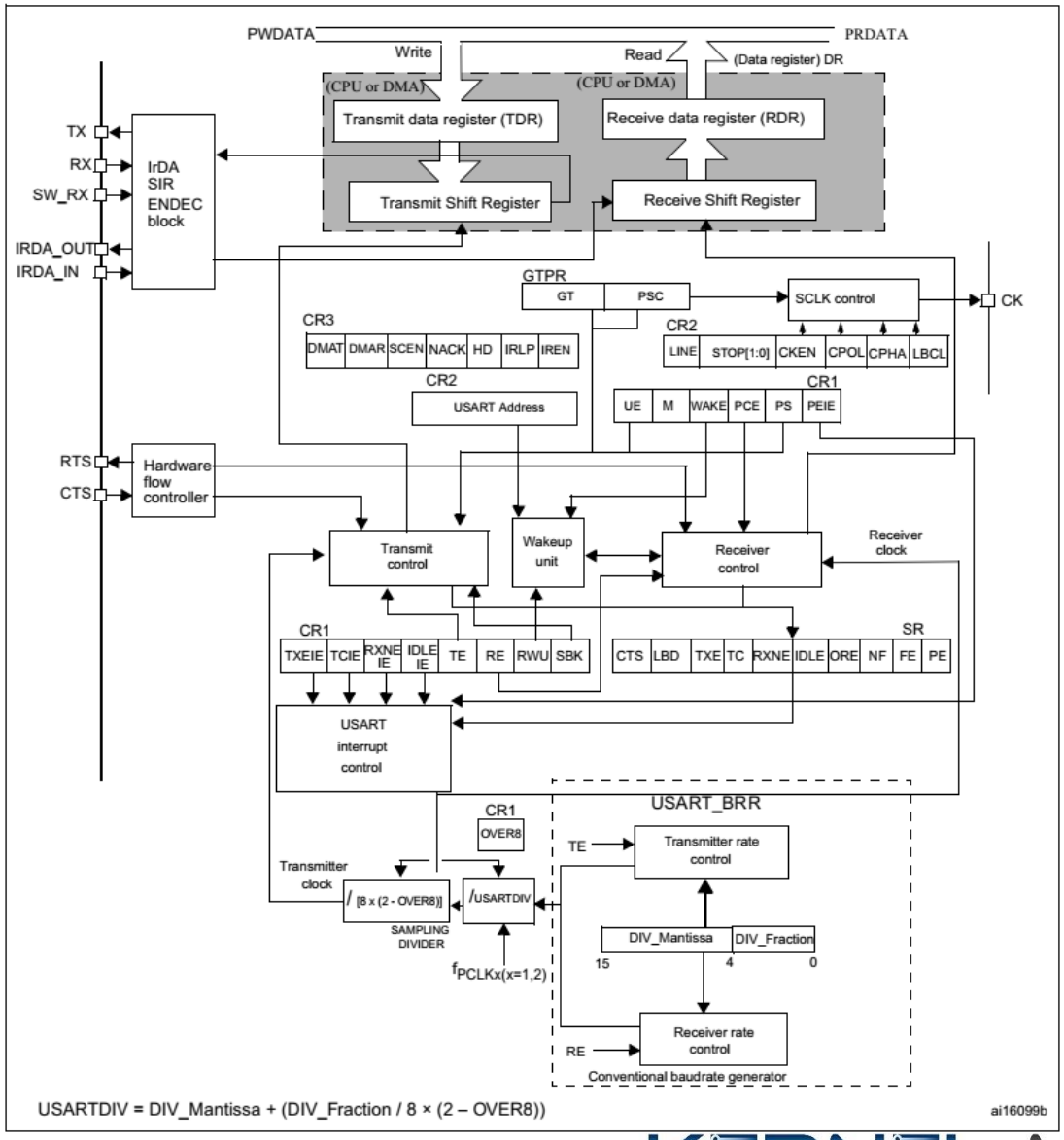
Comm. b/w two boards using UART6 comm. protocol



STM32 UART Features

- Full duplex, asynchronous communications
- Configurable oversampling method by 16 or by 8 to give flexibility between speed and clock tolerance
- Fractional baud rate generator systems
  – Common programmable transmit and receive baud rate (refer to the datasheets for the value of the baud rate at the maximum APB frequency.
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- Separate enable bits for transmitter and receiver

Block Diagram



Basic Operation

- Initialize the UART
  - Enable the UART peripheral, e.g.
  - Set the Rx/Tx pins as UART pins
  - Configure the UART baud rate, data configuration
  - Configure other UART features (e.g. interrupts, FIFO)
- Send/receive a character
  - Single register used for transmit/receive
  - Blocking/non-blocking functions in driverlib:

UART Interrupts

Single interrupt per module, cleared automatically

Interrupt conditions:

- Overrun error
- Break error
- Parity error
- Framing error
- Receive timeout – when FIFO is not empty and no further data is received over a 32-bit period
- Transmit – generated when no data present (if FIFO enabled, see next slide)
- Receive – generated when character is received (if FIFO enabled, see next slide)

Interrupts on these conditions can be enabled individually

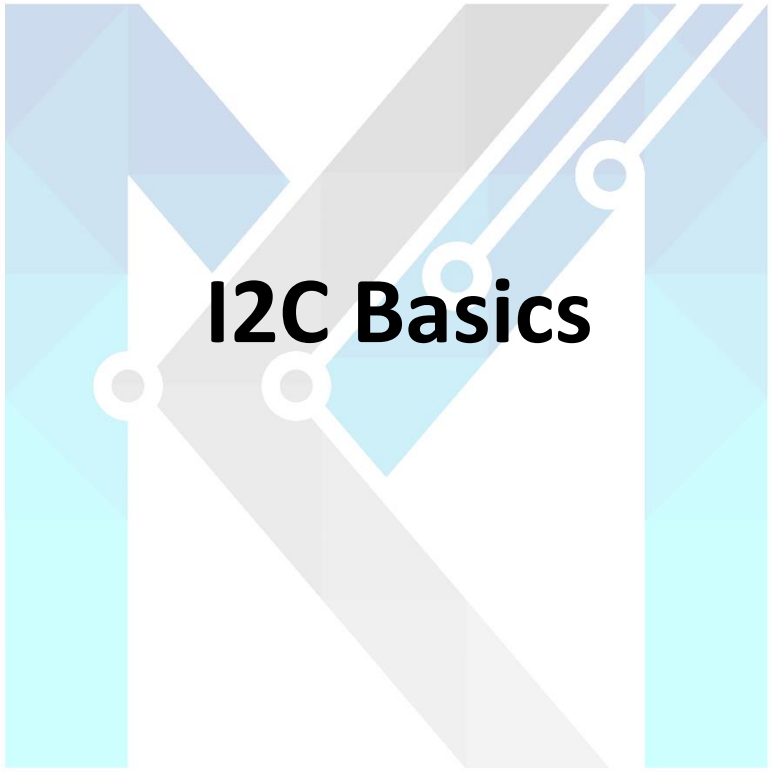
Your handler code must check to determine the source of the UART interrupt and clear the flag(s)

Lab Experiments

Communicate two Launchpad's using UART protocols.

KM_Firmware_Board1 acts as Master and KM_Firmware_Board2 acts as Slave.

KM_Firmware_Board1	**KM_Firmware_Board2**
Sw1 ON & SW2 OFF	RED
Sw1 OFF & Sw2 ON	BLUE
GREEN	SW1 OFF& SW2 ON
PINK	SW1 ON & SW2 OFF



I2C Basics

KERNEL MASTERS

I2C Features

- Philips Semiconductors (now NXP Semiconductors, Qualcomm) developed a simple
  - Serial
  - 8-bit oriented,
  - bidirectional
  - 2-wire
  - synchronous communication protocol.
- Only 2 lines:
  - SDA (Serial Data Line)
  - SCL (Serial Clock Line)
- I2C protocol derived from System Management BUS (SMBUS – developed by INTEL)

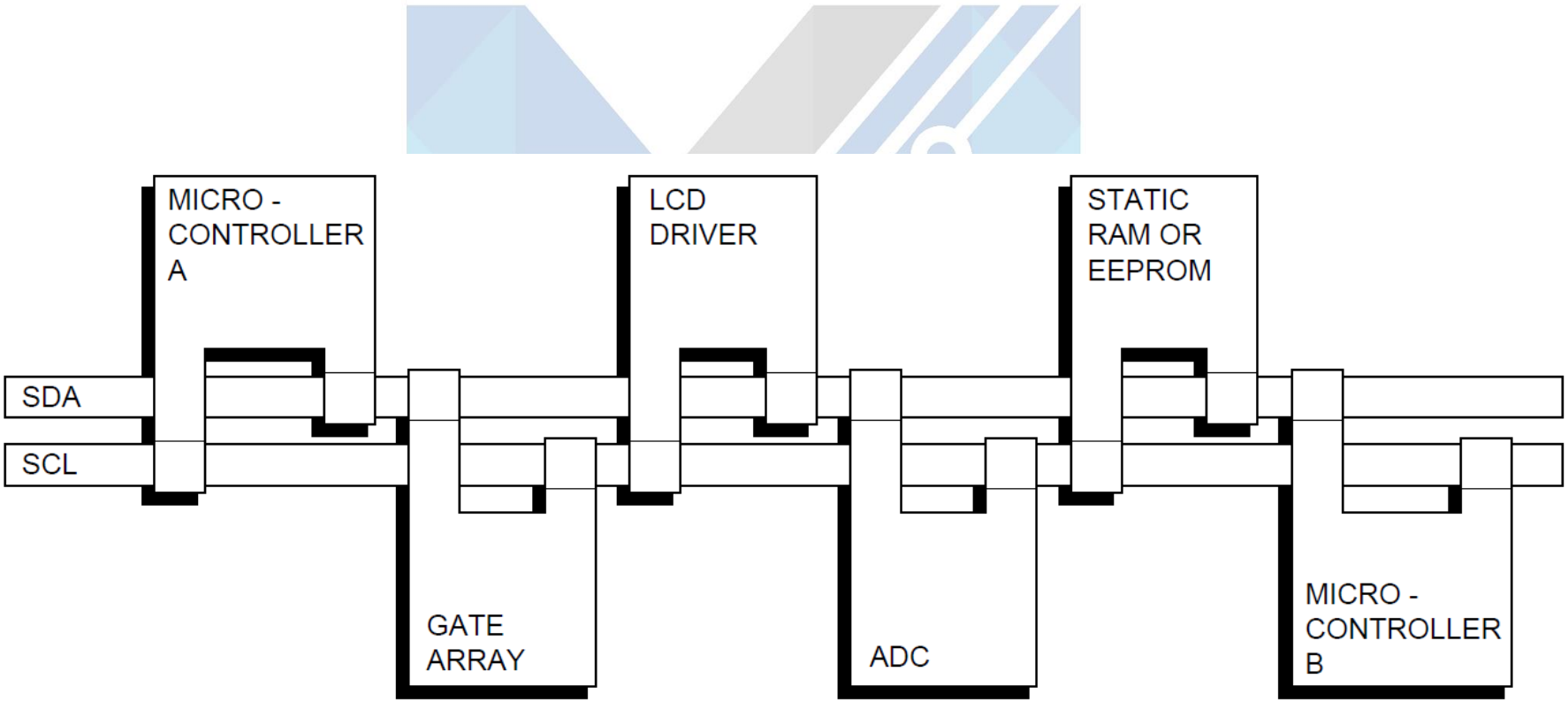
I2C Features

- I2C various Data transfer modes:
  - Standard-mode: 100kbit/s
  - Fast-mode: 400kbit/s
  - Fast-mode Plus (Fm+): 1Mbit/s
  - High-speed mode: 3.4 Mbit/s
  - Ultra Fast-mode: 5 Mbit/s (Unidirectional mode)
- Each device connected to the bus is software addressable by a unique address.
- simple master/slave relationships.
- masters can operate as master-transmitters or as master-receivers

I2C Terminology

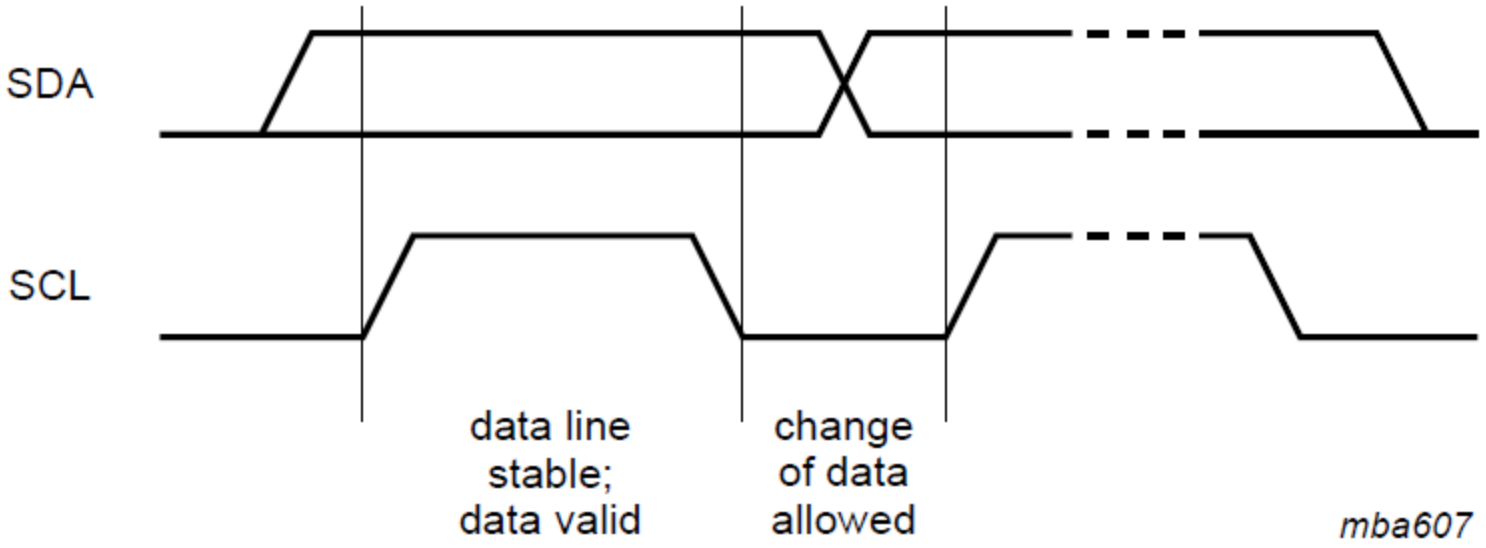
Term	Description
Transmitter	The device which sends data to the bus
Receiver	The device which receives data from the bus
Master	the device which initiates a transfer, generates clock signals and terminates a transfer
Slave	the device addressed by a master
Multi-Master	more than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted

I2C Bus Configuration



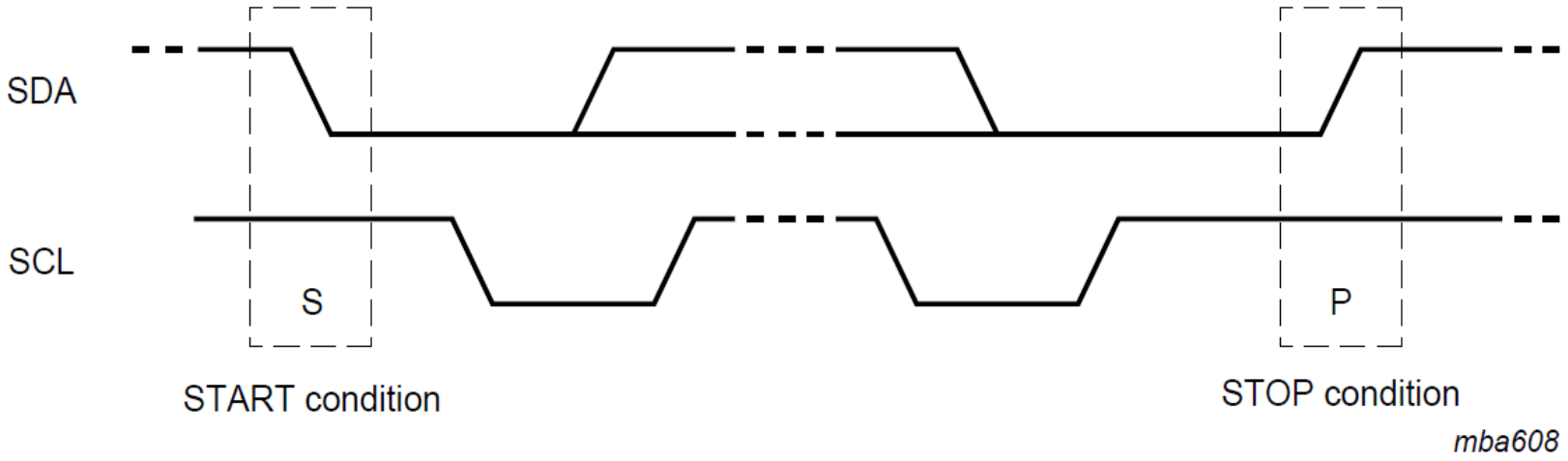
Data Validity

- The data on the SDA line must be stable during the HIGH period of the clock.
- The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Figure).
- One clock pulse is generated for each data bit transferred



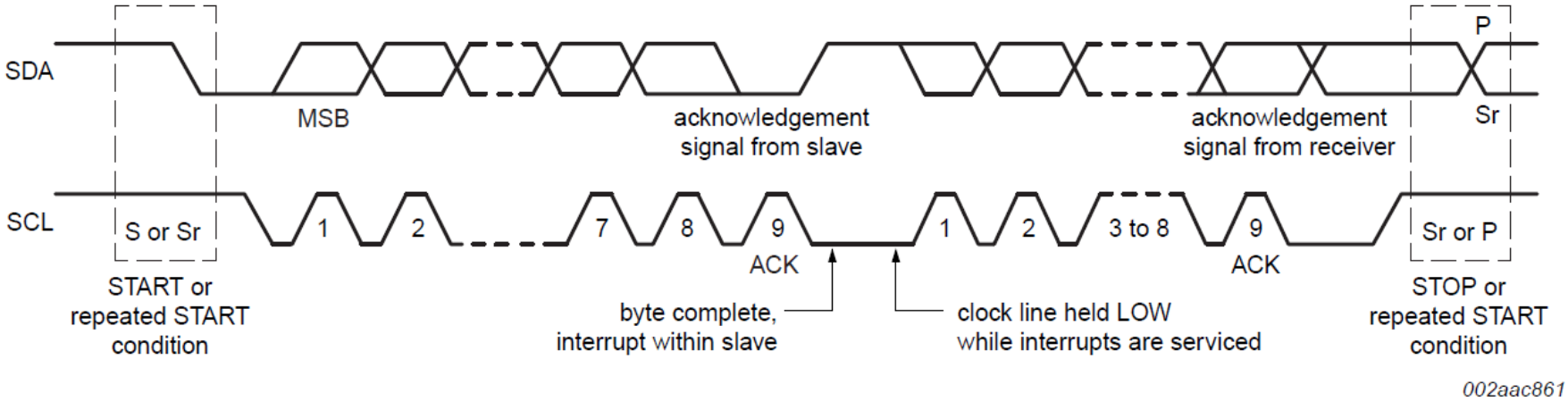
Start and Stop Conditions

- A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition.
- A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.
- START and STOP conditions are always generated by the master.



Byte Format

Every byte put on the SDA line must be eight bits long. The number of bytes that can be transmitted per transfer is unrestricted.
Each byte must be followed by an Acknowledge bit.
Data is transferred with the Most Significant Bit (MSB) first.



Acknowledge (ACK) and Not Acknowledge (NACK)

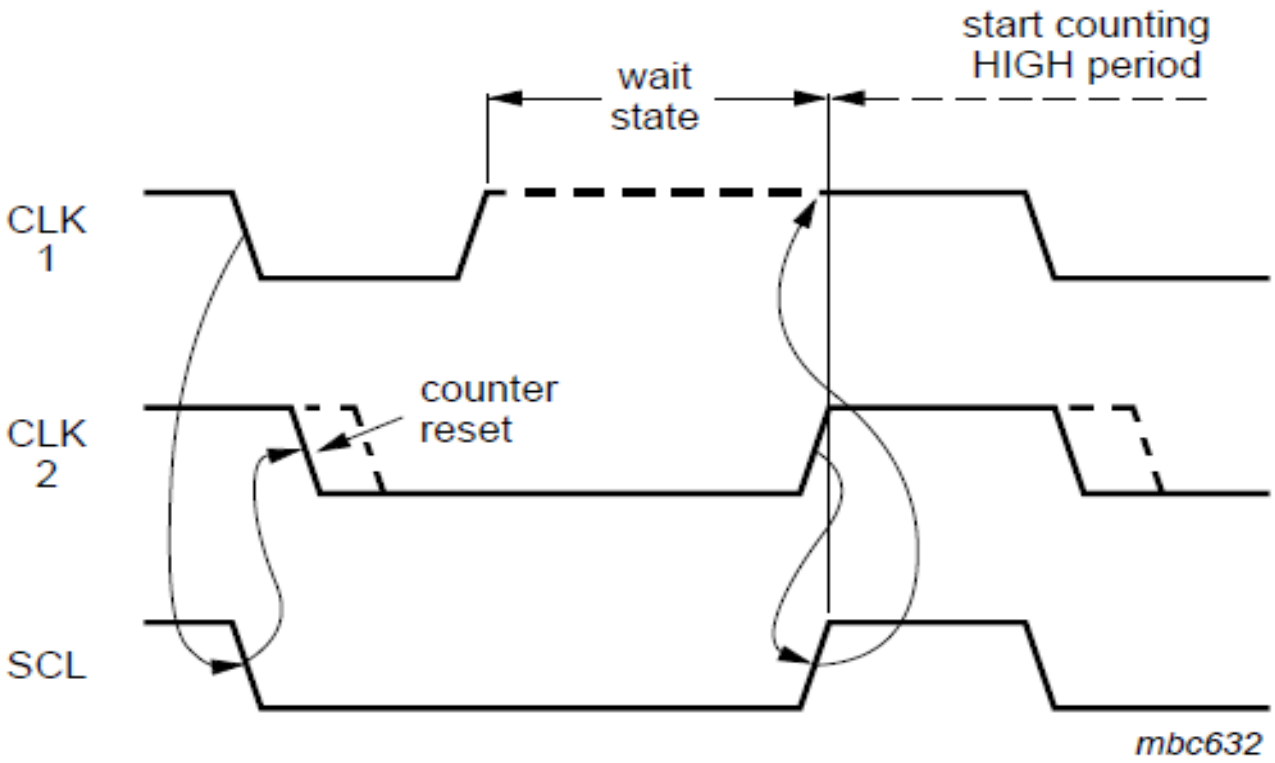
- **ACK Defines:** the transmitter releases the SDA line during the acknowledge clock pulse so the receiver can pull the SDA line LOW and it remains stable LOW during the HIGH period of this clock pulse.
- **NACK Defines:** When SDA remains HIGH during this ninth clock pulse, this is defined as the Not Acknowledge signal.

Acknowledge (ACK) and Not Acknowledge (NACK)

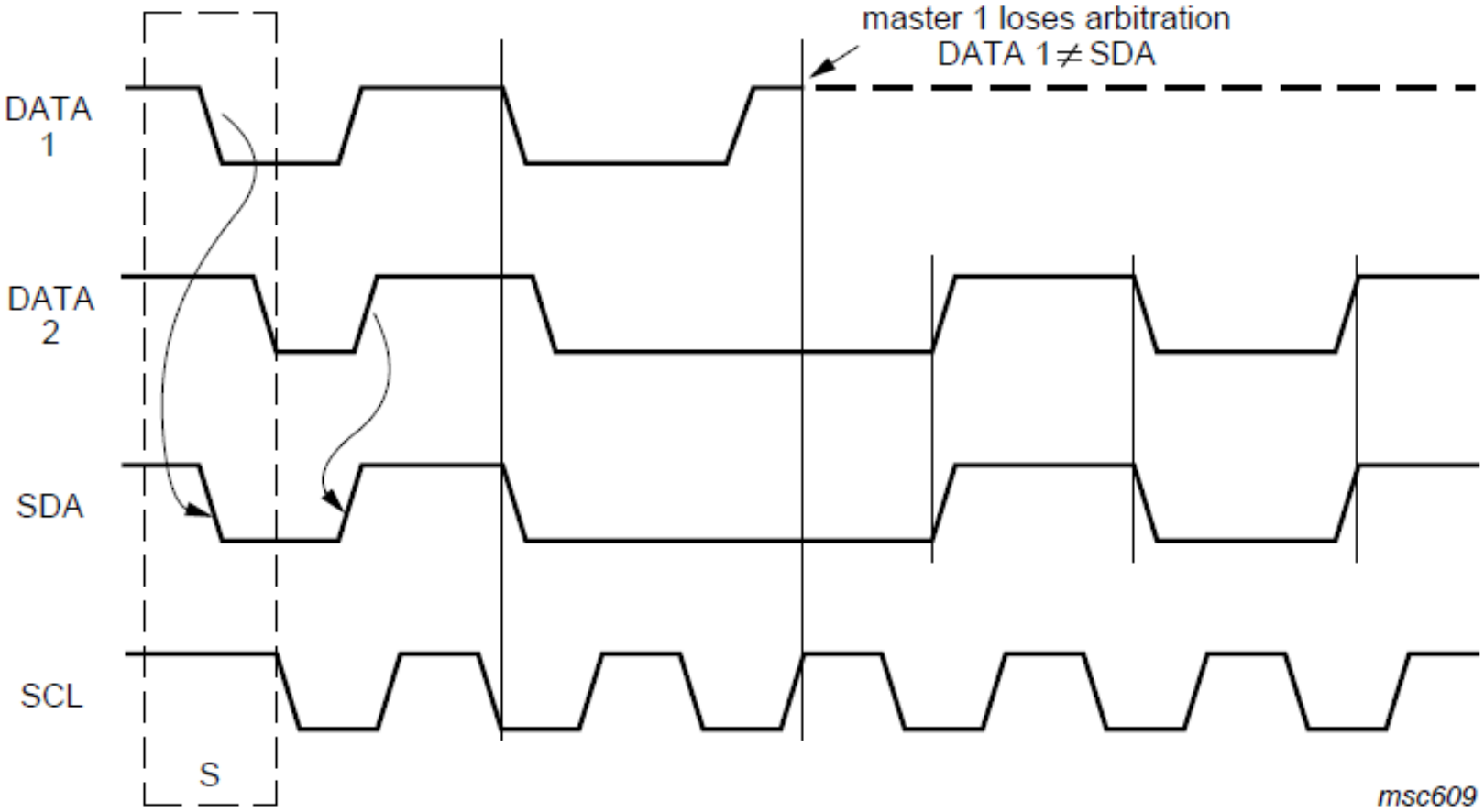
There are five conditions to the generation of a NACK:

1. No receiver is present on the bus with the transmitted address so there is no device to respond with an acknowledge.
2. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the master.
3. During the transfer, the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any more data bytes.
5. A master-receiver must signal the end of the transfer to the slave transmitter.

Clock Synchronization



Arbitration



I2C FAQ

Q. What happens if I omit the pullup resistors on I2C lines?

A. There will be no communication on the I2C bus. At all. The MCU will not be able to generate the I2C start condition. The MCU will not be able to transmit the I2C address.

Q. The lack of pullups is likely to damage any of those two ICs in my board?

A. Even without the internal pull-ups, a lack of any pull-ups will not damage either IC. The internal build of i2c device SCl and SDA lines are like NPN transistors. They are Open Collectors, essentially current controlled/switched diodes.

.

I2C FAQ

- **Why need pull up resistors in I2C?**
- Generally you will need to have the pullup resistors for an I2C interface circuit. If the interface is truly a full spec I2C on both ends of the wires then the signal lines without the resistors will never be able to go to the high level.
- They may remain low or go to some intermediate level determined by the leakage current in the parts at each end.
- The reason for this is because true I2C is an open drain bus.
- I2C is a TTL-logic protocol; so your data and clock lines are open-drain. In other words, the I2C hardware can only drive these lines low; they are left floating when not a zero. That's where the pull-up resistors come in.
- Some devices may actually have on-chip pullup resistors in the 20K to 100K ohm range just to hold the interface pins at a high inactive level when the I2C interface on the part is not in use. For simple and short interfaces these pullup resistors may be just enough to provide the current needed to pull the lines high while clocks and/or data is being signaled.
- i2c pull up resistors allows for features like concurrent operation of more than one I2C master (if they are multi-master capable) or stretching (slaves can slow down communication by holding down SCL).

Bit Banging

- **I2C Bit Banging:**

It is hard to tell from your schematic but in some instances I2C interfaces are implemented using general purpose I/O port pins and then bit banged in software. Sometimes the implementer may not operate the I/O pins in this configuration using an open drain methodology and this may play a factor on why an interface without pullup resistors may seem to work.

STM32
I2C Controller

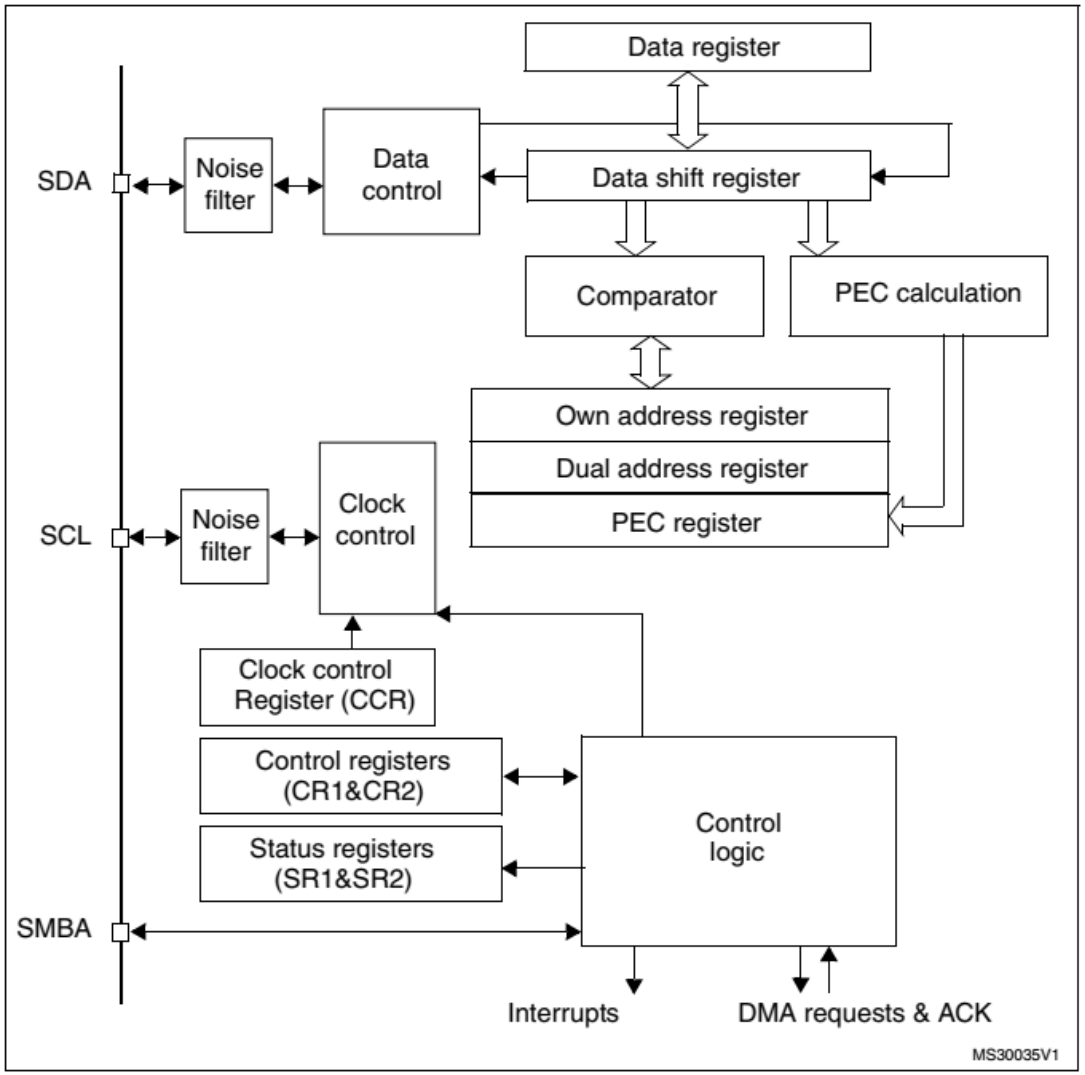
KERNEL MASTERS

STM32 I2C Controller

Specifications:

- Multi master capability: the same interface can act as Master or Slave
- I2C Master features:
  – Clock generation
  – Start and Stop generation
- I2C Slave features:
  – Programmable I2C Address detection
  – Dual Addressing Capability to acknowledge 2 slave addresses
  – Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and General Call
- Supports different communication speeds:
  – Standard Speed (up to 100 kHz)
  – Fast Speed (up to 400 kHz)
  – The I2C bus frequency can be increased up to 1 MHz. For more details about the complete solution, please contact your local ST sales representative
- **Status flags:**
  – Transmitter/Receiver mode flag
  – End-of-Byte transmission flag
  – I2C busy flag
- **Error flags:**
  – Arbitration lost condition for master mode
  – Acknowledgment failure after address/ data transmission
  – Detection of misplaced start or stop condition
  – Overrun/Underrun if clock stretching is disabled
- **2 Interrupt vectors:**
  – 1 Interrupt for successful address/ data communication

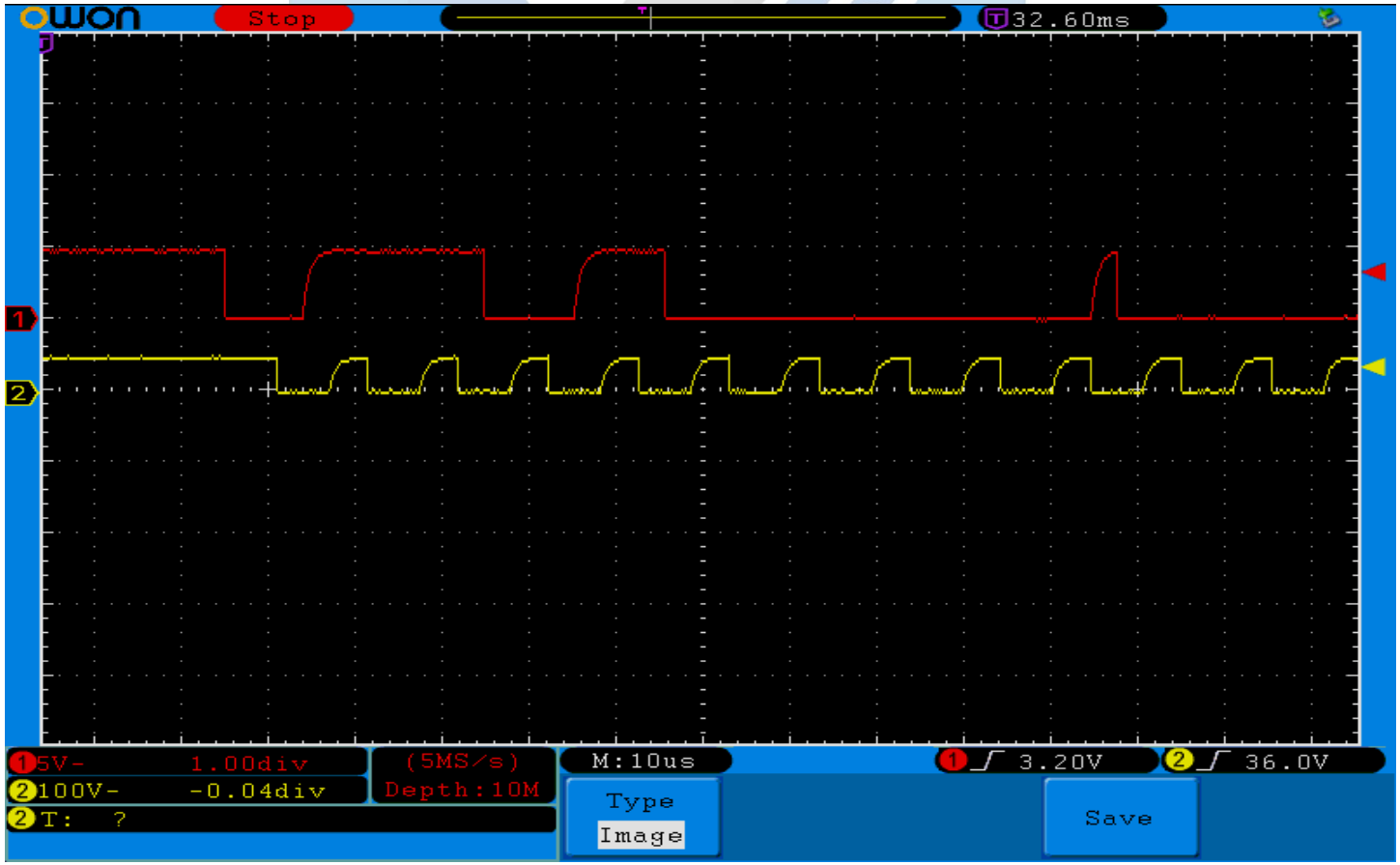
STM32 I2C Controller Block Diagram



I2C Waveform

I2C_Send2(0x68,0x00,15);

<S><Slave Address><W><ACK> <Sec Reg Address><ACK> <Data><ACK>

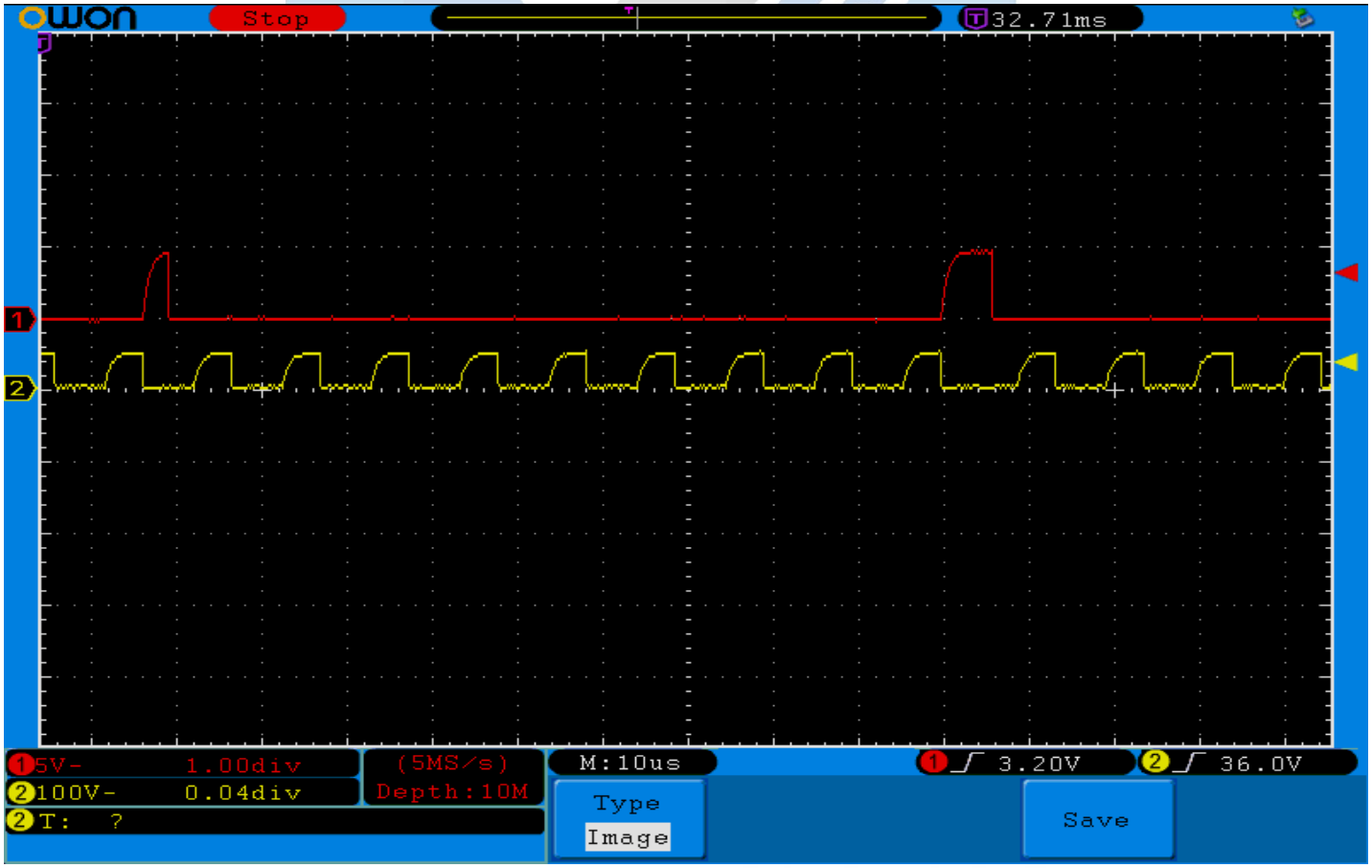


KERNEL MASTERS

I2C Waveform

I2C_Send2(0x68,0x00,15);

<S><Slave Address><W><ACK> **<Sec Reg Address><ACK>** <Data><ACK>

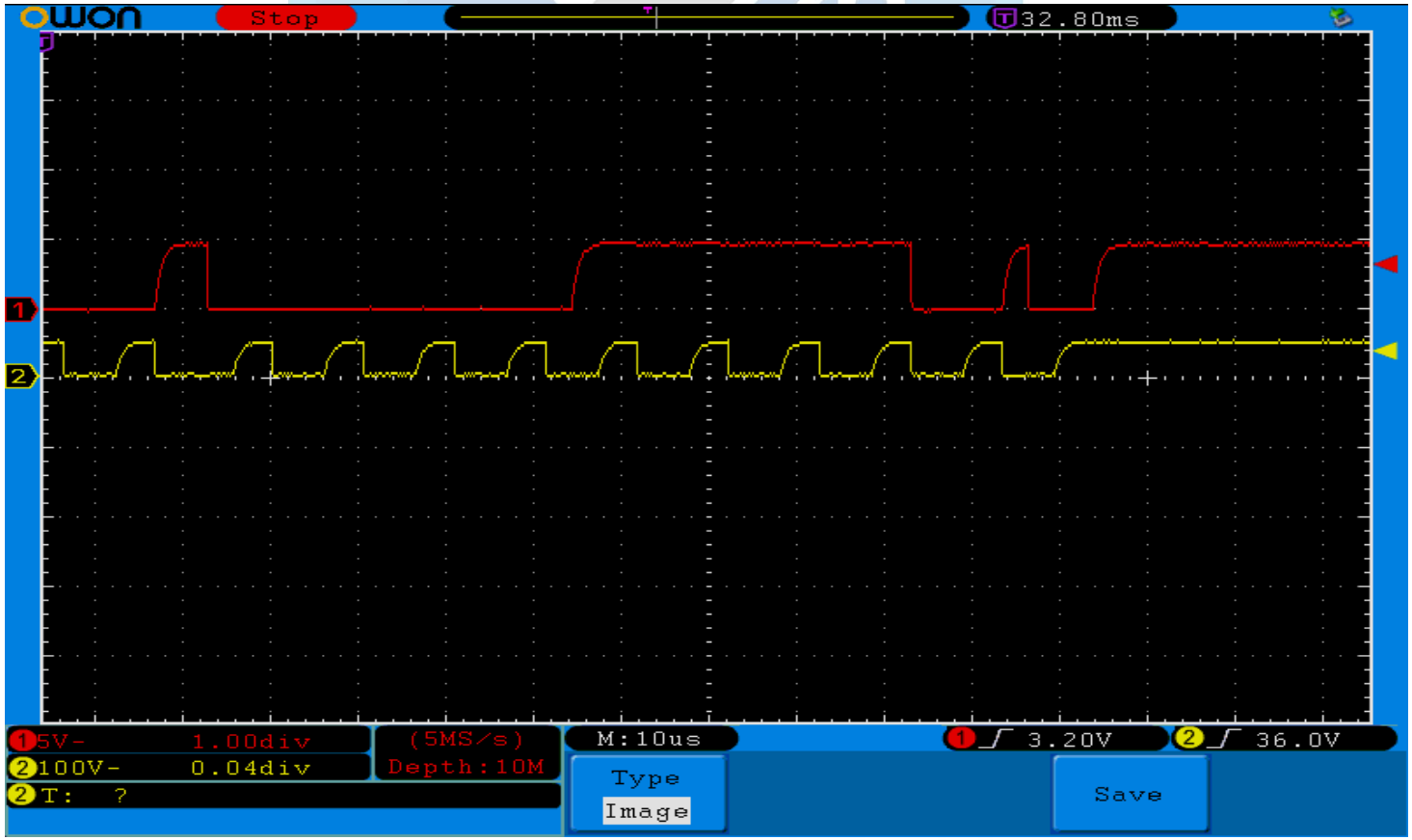


KERNEL MASTERS

I2C Waveform

I2C_Send2(0x68,0x00,15);

<S><Slave Address><W><ACK> <Sec Reg Address><ACK> **<Data><ACK>**



KERNEL MASTERS

SPI/SSI Basics

KERNEL MASTERS

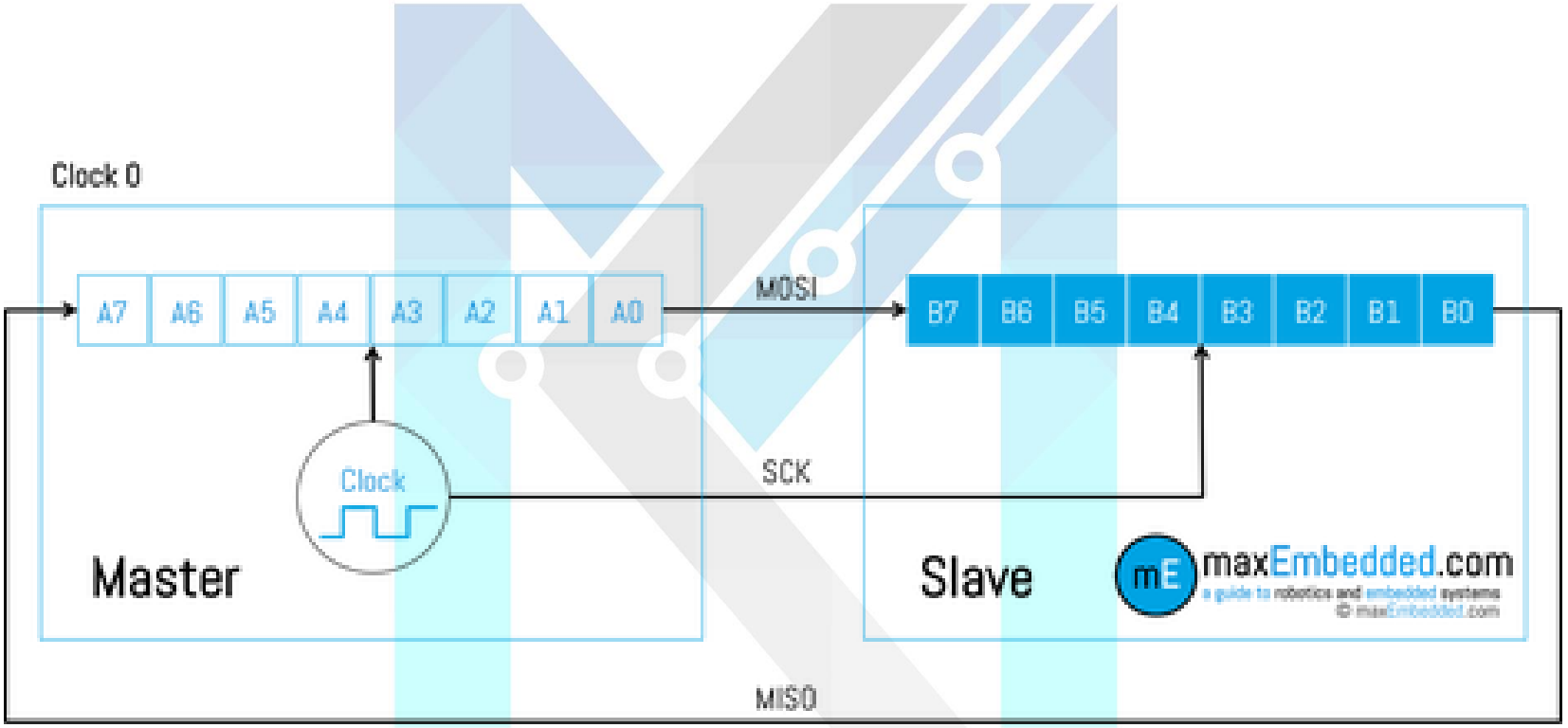
SPI Features

- SPI is a
  - 4 wire
  - full-duplex
  - synchronous serial data transfer protocol.
- SPI bus consists of four wires/signals – MOSI, MISO, SCK and SS'.
- Maximum speed up to 20Mbps
- Doesn't support Multi master mode. Max. No of masters: 1
- Max. No of Slaves: Theoretically Unlimited.

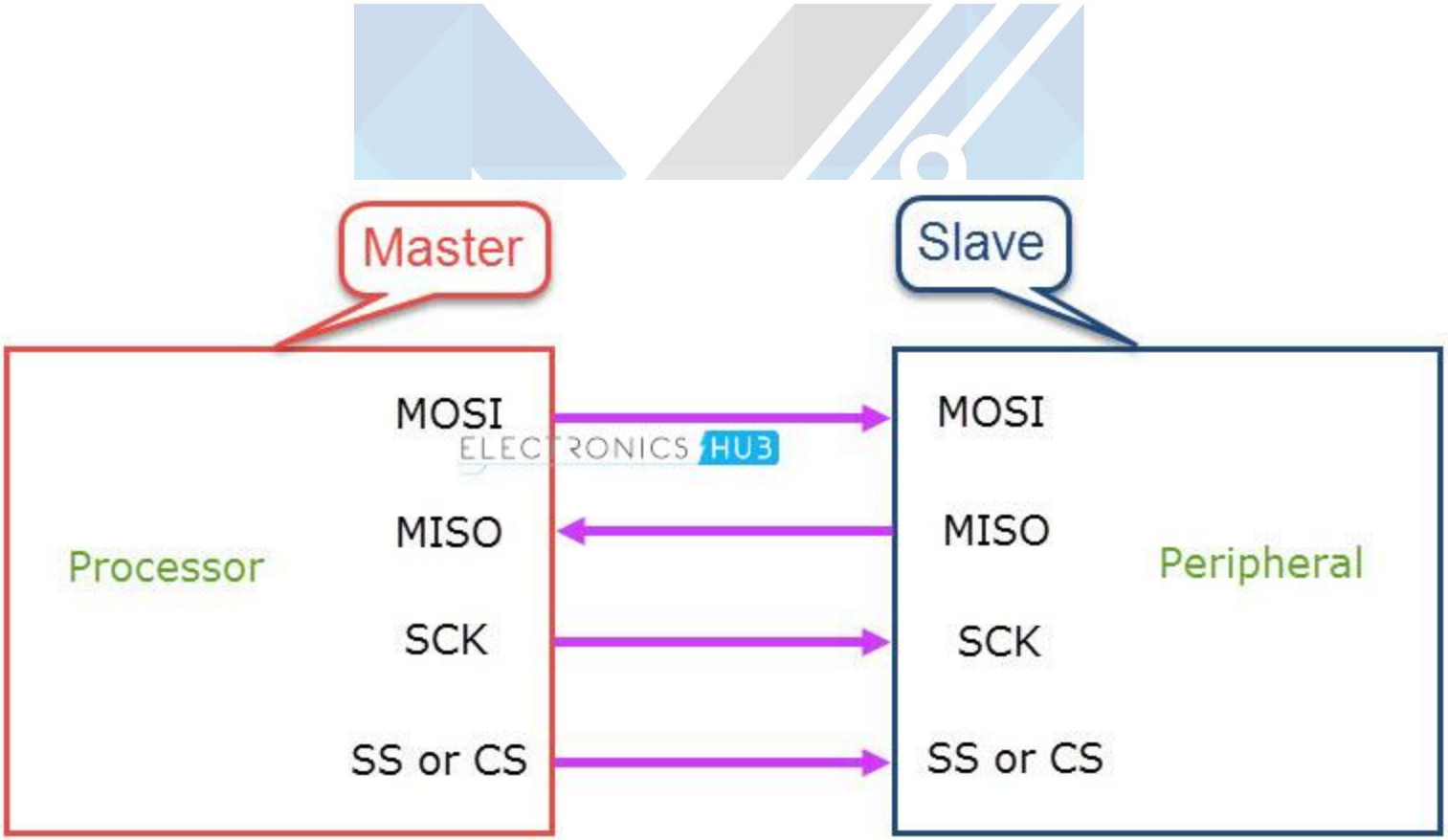
SPI Features

- Data transfer takes place in between *Master* and *Slave* devices.
- Each *Master/Slave* device has an internal 8 bit shift register, which is connected to other devices so as to form a circular/ring buffer.
- At each clock pulse, data gets right shifted in the circular/ring buffer.
- After 8 clock pulses, data is completely exchanged in between devices.
- When we connect more than one *Slave* devices, then we choose them using the SS' signal.

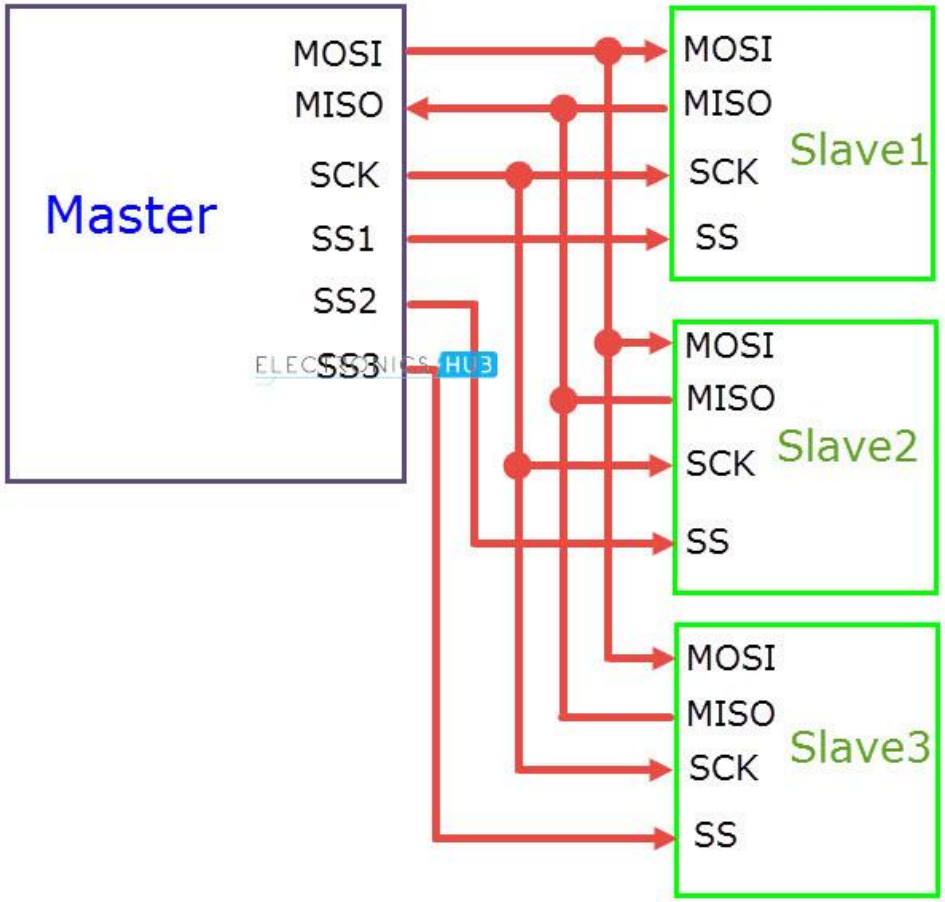
SPI Features



Single master, single slave SPI



Single master, Multiple slave SPI



SPI Modes

- **CPOL – Clock Polarity:** This determines the base value of the clock i.e. the value of the clock when SPI bus is idle.
  - When CPOL = 0, base value of clock is zero i.e. SCK is LOW when idle.
  - When CPOL = 1, base value of clock is one i.e. SCK is HIGH when idle.
- **CPHA – Clock Phase:** This determines the clock transition at which data will be sampled/captured.
  - When CPHA = 0, data is sampled at clock's rising/leading edge.
  - When CPHA = 1, data is sampled at clock's falling/trailing edge
- CPOL and CPHA must be set so that *Master* and *Slave* devices sync properly.

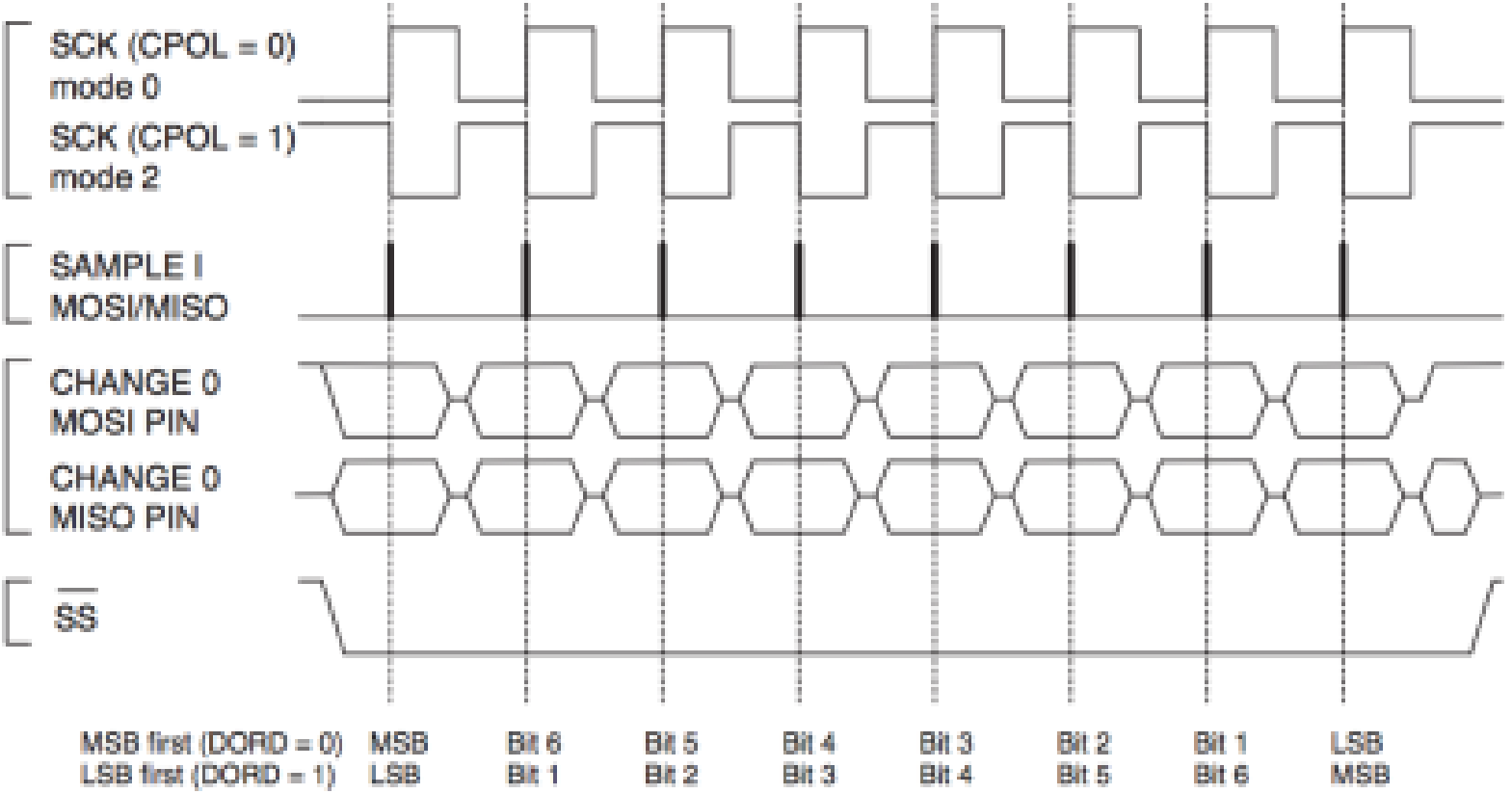
SPI Modes

Table 59. CPOL and CPHA Functionality

	Leading Edge	Trailing Edge	SPI Mode
CPOL = 0, CPHA = 0	Sample (Rising)	Setup (Falling)	0
CPOL = 0, CPHA = 1	Setup (Rising)	Sample (Falling)	1
CPOL = 1, CPHA = 0	Sample (Falling)	Setup (Rising)	2
CPOL = 1, CPHA = 1	Setup (Falling)	Sample (Rising)	3

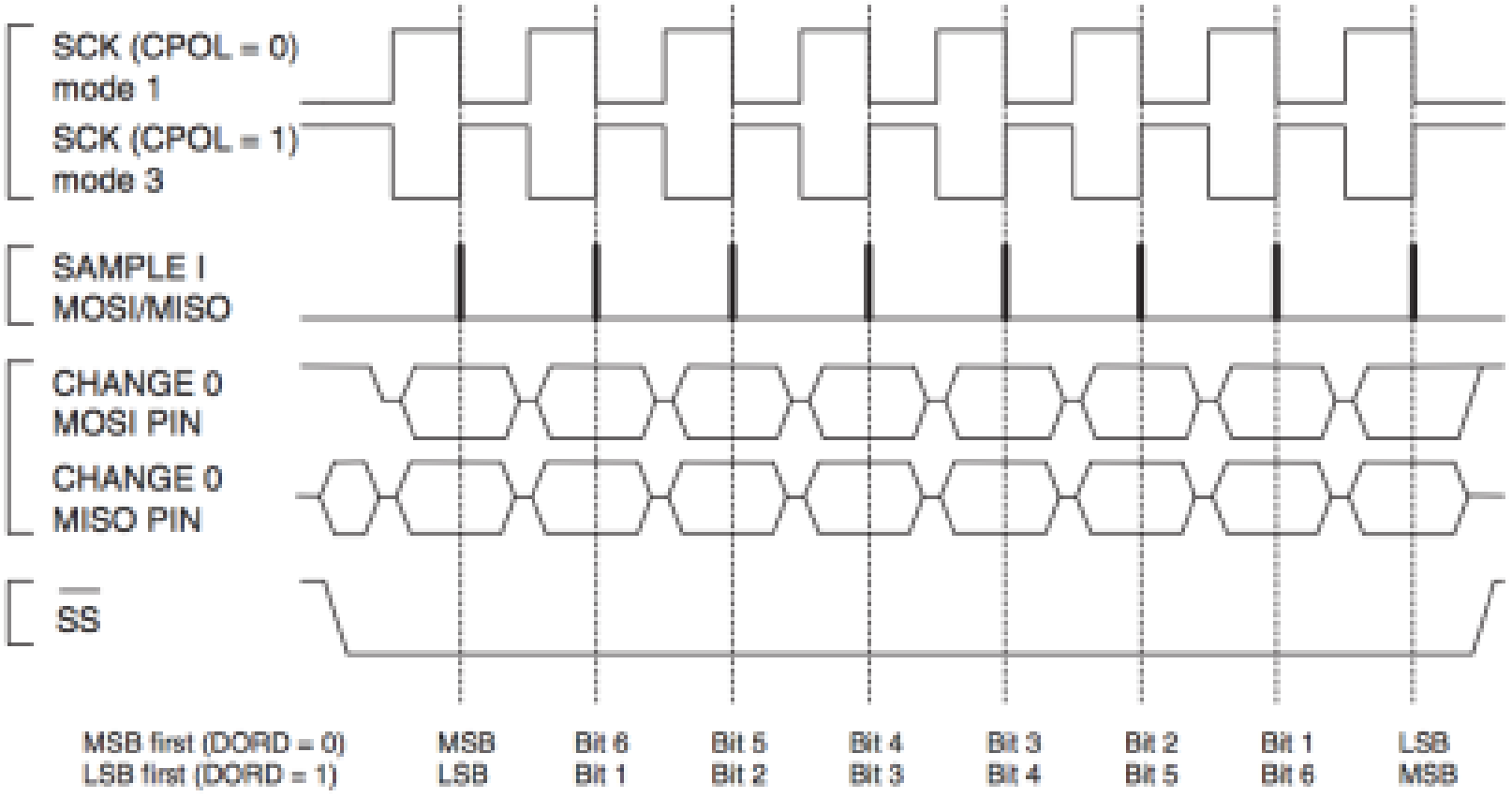
SPI Modes

Figure 67. SPI Transfer Format with CPHA = 0



SPI Modes

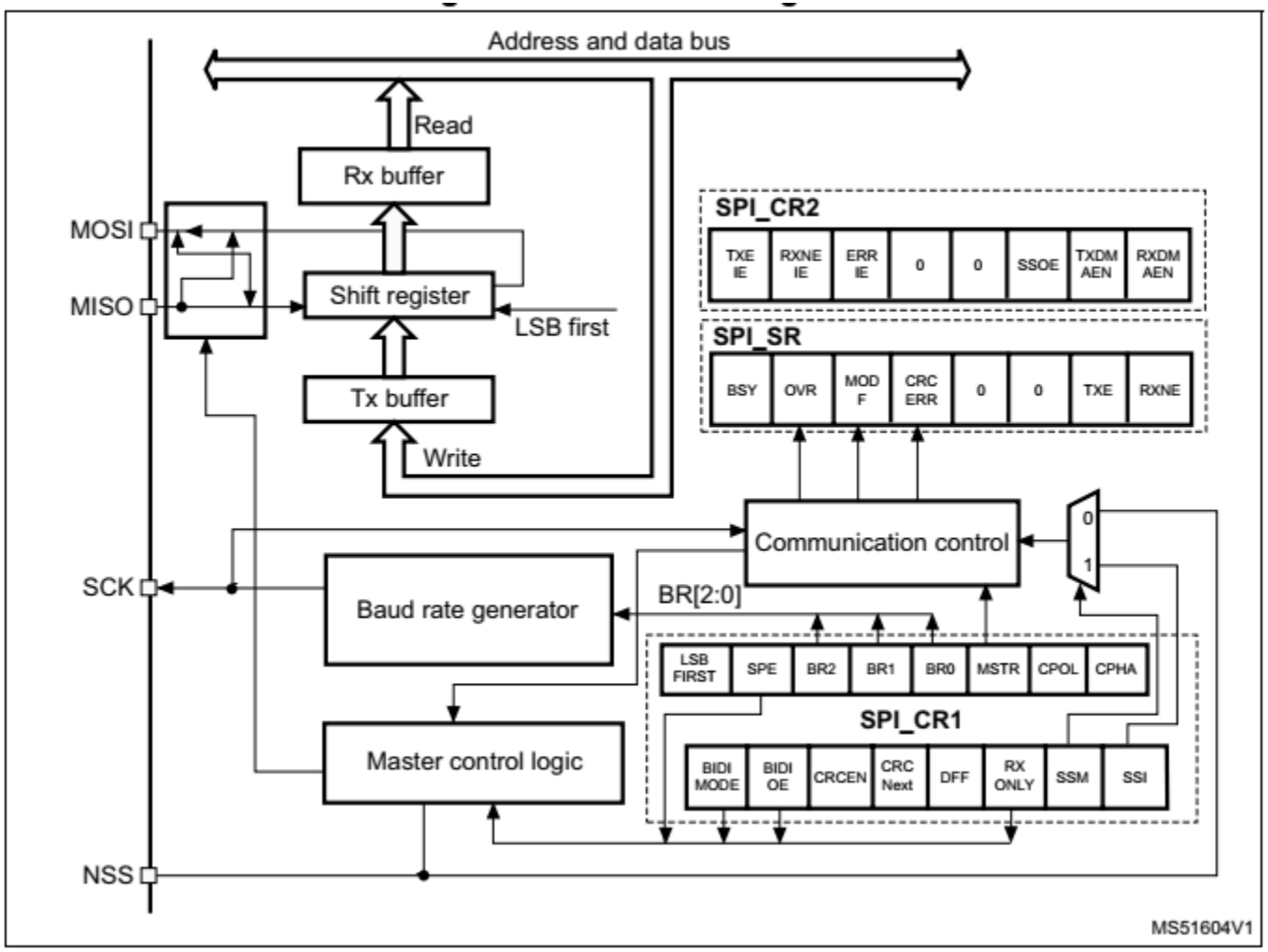
Figure 68. SPI Transfer Format with CPHA = 1



STM32 SPI Features

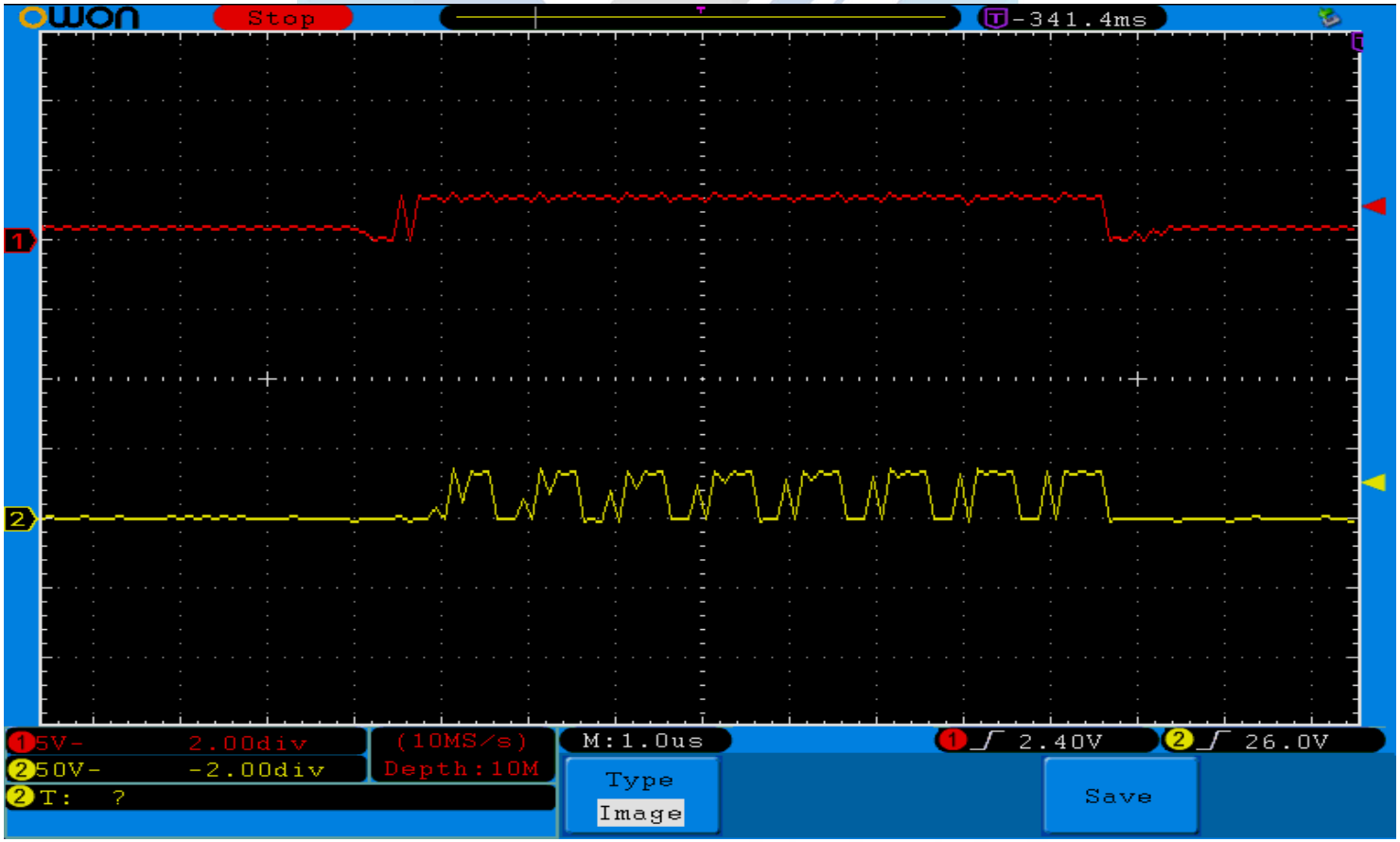
- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line.
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- Slave mode frequency (fPCLK/2 max)
- Faster communication for both master and slave
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- SPI TI mode
- Hardware CRC feature for reliable communication:
  – CRC value can be transmitted as last byte in Tx mode
  – Automatic CRC error checking for last received byte
- Master mode fault, overrun and CRC error flags with interrupt capability
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests.

STM32 SPI Block Diagram



SPI Waveforms

Port_Out(0xFF);



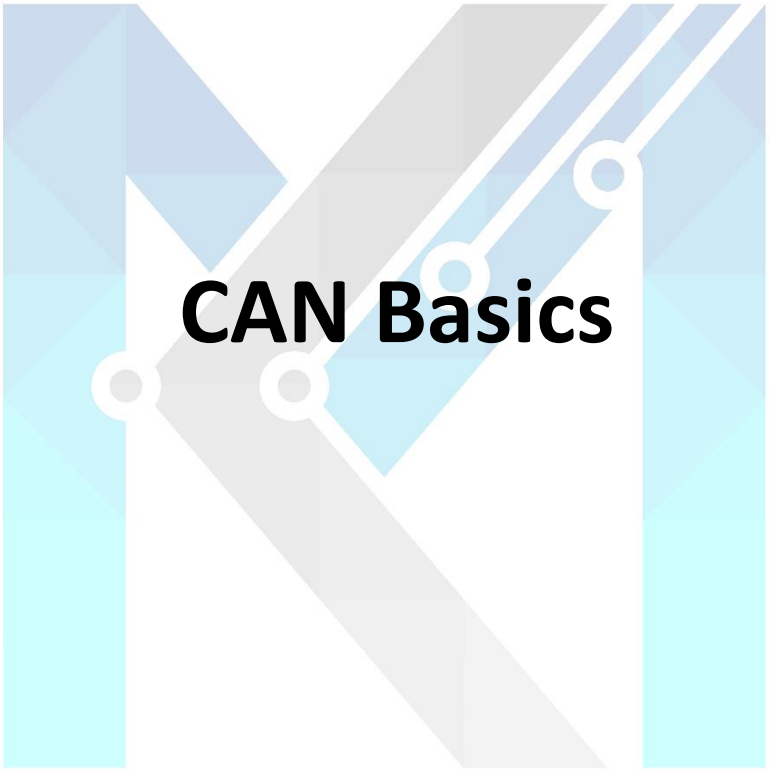
KERNEL MASTERS

Lab Experiments

Communicate two Launchpad's using SPI protocol.

LP1 acts as Master and LP2 acts as Slave.

Launchpad1	**Launchpad2**
Sw1 ON & SW2 OFF	RED
Sw1 OFF & Sw2 ON	BLUE
GREEN	SW1 OFF& SW2 ON
PINK	SW1 ON & SW2 OFF



CAN Basics

KERNEL MASTERS

CAN Features

- Controller Area Network is a
  - 2 wire (CANH & CANL)
  - serial data communication bus.
- CAN was originally developed by **Robert Bosch** for use in automobiles, and is now extensively used in industrial automation and control applications.
- Data rate up to 1 Mbits/sec, having excellent error detection.
- CAN protocol developed by international standard for serial communication, specifically **the ISO 11989**.

CAN Features

Four Component in CAN system

1. **CAN bus** consisting of 2 wires (CANH, CANL) with 120 ohm termination resistors on each end.
2. **Transceiver:** Which handles the voltage levels and interfacing the separate receive (RXD) and transmit (TXD) signals on the CAN bus.
3. **CAN Controller:** Which is hardware built in to the microcontroller, and it handles message timings, priority, error detection and retransmission.
4. **Software** that handles the high level functions of generating data to transmit and processing data received from other nodes.

CAN Features

- CAN is based on the **"broadcast communication mechanism",** which follows a message –based transmission protocol rather than an address-based protocol.

CAN provides two communication services:

1. Sending of a message [Data Frame Transmission]
2. Requesting of a message [Remote Transmission Request]

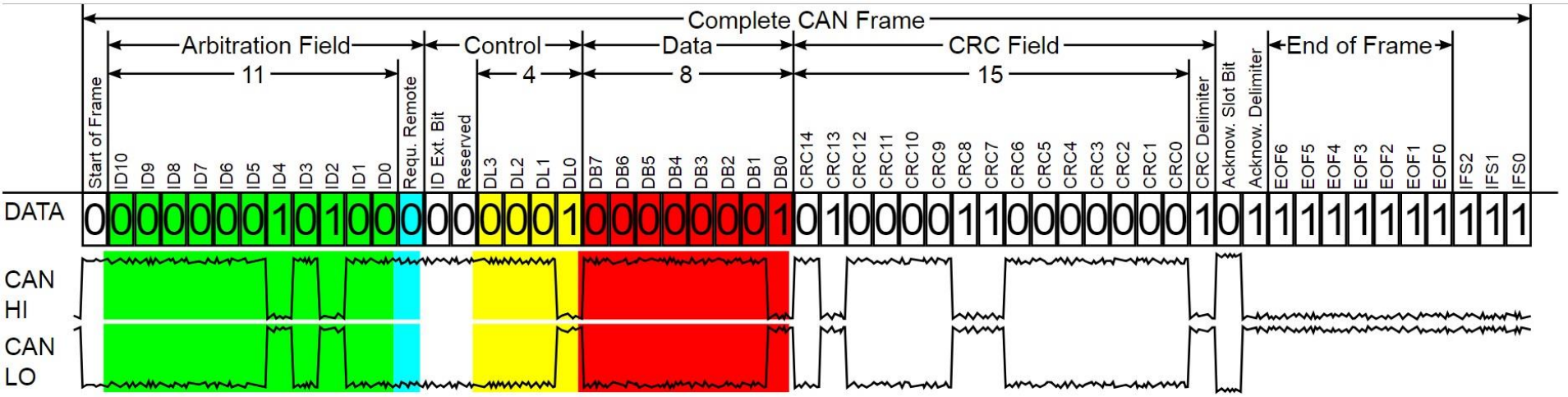
CAN Features

- In CAN system, messages are identified by their contents rather by address.
- Each message sent on the bus has a unique identifier, which defines both content and the priority of the message.

CAN has four frame types:

- **Data Frame:** a frame containing node data for transmission
- **Remote Frame:** a frame requesting the transmission of a specific identifier
- **Error Frame:** a frame transmitted by any node detecting an error
- **Overload Frame:** a frame to inject a delay between data or remote frame

CAN Standard Format Data Frame



There are two message formats:

Base frame format [CAN 2.0A]: with 11 identifier bits

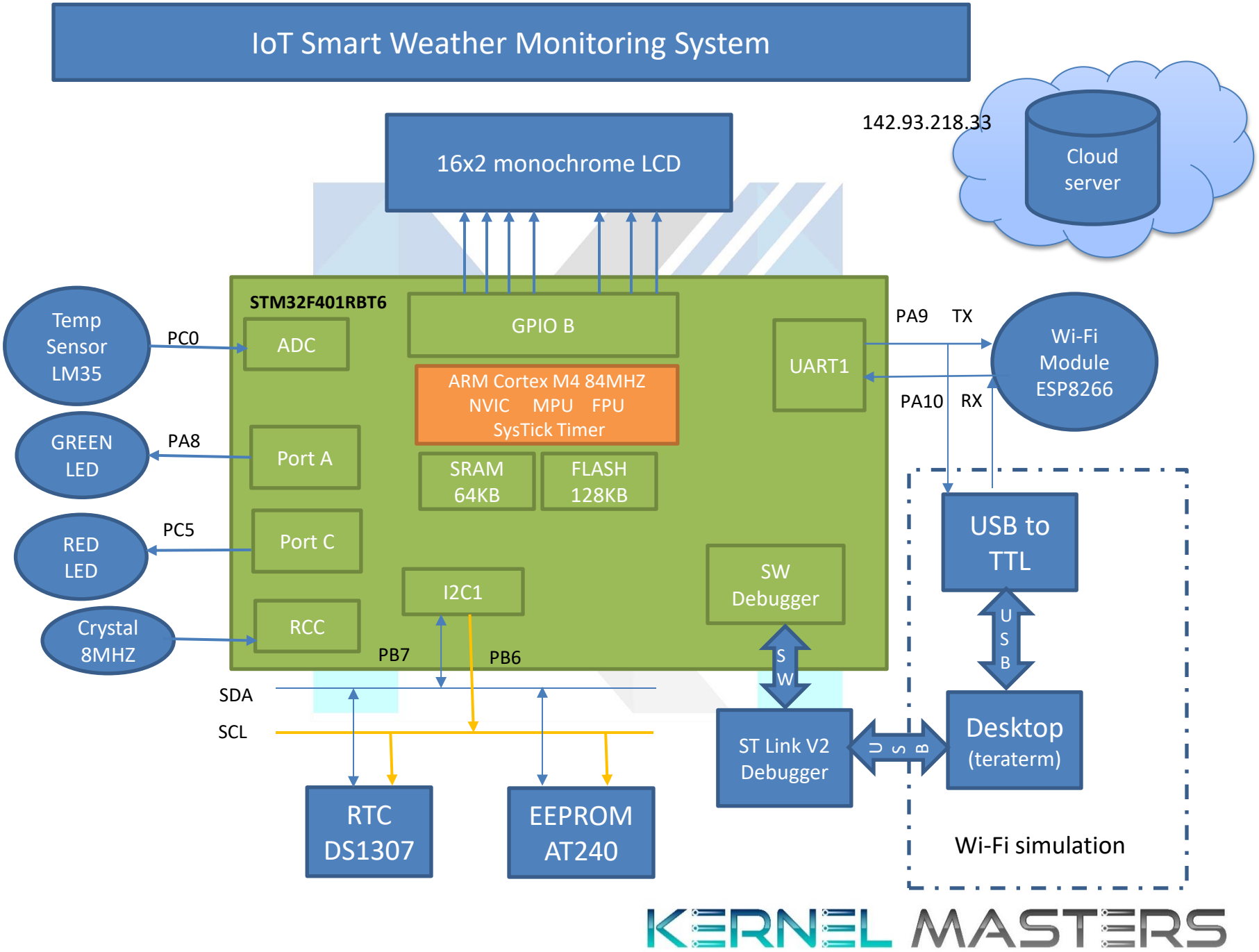
Extended frame format [CAN 2.0B]: with 29 identifier bits

CAN Standard Format Data Frame

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier (green)	11	A (unique) identifier which also represents the message priority
Remote transmission request (RTR) (blue)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames (see Remote Frame, below)
Identifier extension bit (IDE)	1	Must be dominant (0) for base frame format with 11-bit identifiers
Reserved bit (r0)	1	Reserved bit. Must be dominant (0), but accepted as either dominant or recessive.
Data length code (DLC) (yellow)	4	Number of bytes of data (0–8 bytes)[a]
Data field (red)	0–64 (0-8 bytes)	Data to be transmitted (length in bytes dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

UART, CAN, I2C, SPI

Comm. method	Shares clock	Num. of wires	Speed	Dist	Pros	Cons
UART	No	2	115Kbits/sec max	Medium, long	Simple; Widely supported; Large range of physical standard interfaces (TTL, RS-232, RS-422, RS-485);	It's asynchronous; Requires reasonable clock accuracy at both ends;
CAN	No	3	1 Mbits/sec	Long: 40m (1Mbit/sec) up to 10km (5Kbits/sec)	Highly reliable; Reduces amount of wiring; Multi-master capability;	Complex;
I2C	Yes	2	100Kbits/sec 400Kbits/sec fast mode	Short, medium (< 6")	Simple; Multi-master capability; Only 2 wires to support multiple devices; Robust in noisy or power-up/down situations;	More complex protocol than SPI; Harder to level-shift or optoisolate due to bidirectional lines; Need for pull-up resistors can reduce power efficiency in some cases;
SPI	Yes	4	10-20Mbits/sec	Short	Fast, easy, simple; A lot of support; Self clocking; Flexible data word sizes;	Multiple devices need multiple select lines; No acknowledgement ability; No inherent arbitration ; No flow control; Single master only;


IoT Smart Weather Monitoring System
142.93.218.33
Cloud server
16x2 monochrome LCD
STM32F401RBT6
GPIO B
ADC
ARM Cortex M4 84MHZ
NVIC MPU FPU
SysTick Timer
SRAM 64KB
FLASH 128KB
Port A
Port C
RCC
I2C1
UART1
SW Debugger
Temp Sensor LM35
PC0
GREEN LED
PA8
RED LED
PC5
Crystal 8MHZ
PA9 TX
PA10 RX
Wi-Fi Module ESP8266
USB to TTL
USB
SW
PB7
PB6
SDA
SCL
ST Link V2 Debugger
USB
Desktop (teraterm)
RTC DS1307
EEPROM AT240
Wi-Fi simulation
KERNEL MASTERS