Queen Mary
University of London
Science and Engineering

School of Electronic Engineering and Computer Science
QMUL-BUPT Joint Programme

EBU6475 Microprocessor System Design
EBU5476 Microprocessors for Embedded Computing

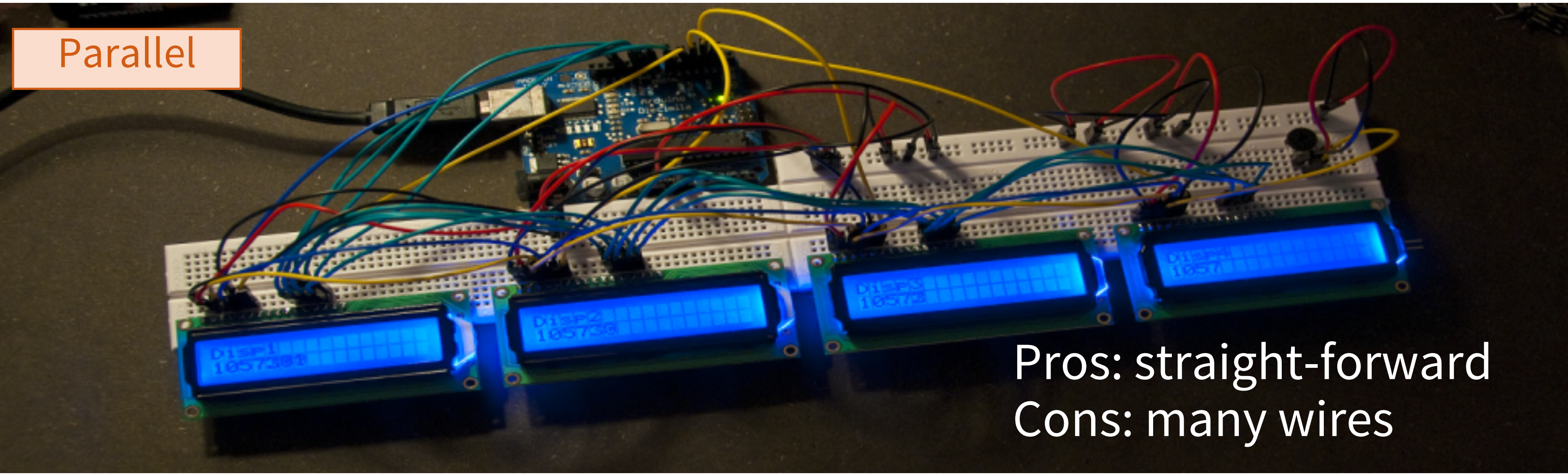
I^{2}C Interface

References:
Chapter 8 (p236-249), Embedded Systems Fundamentals
Chapter 8, STM32F401RE Reference Manual

arm
University Program Education Kits

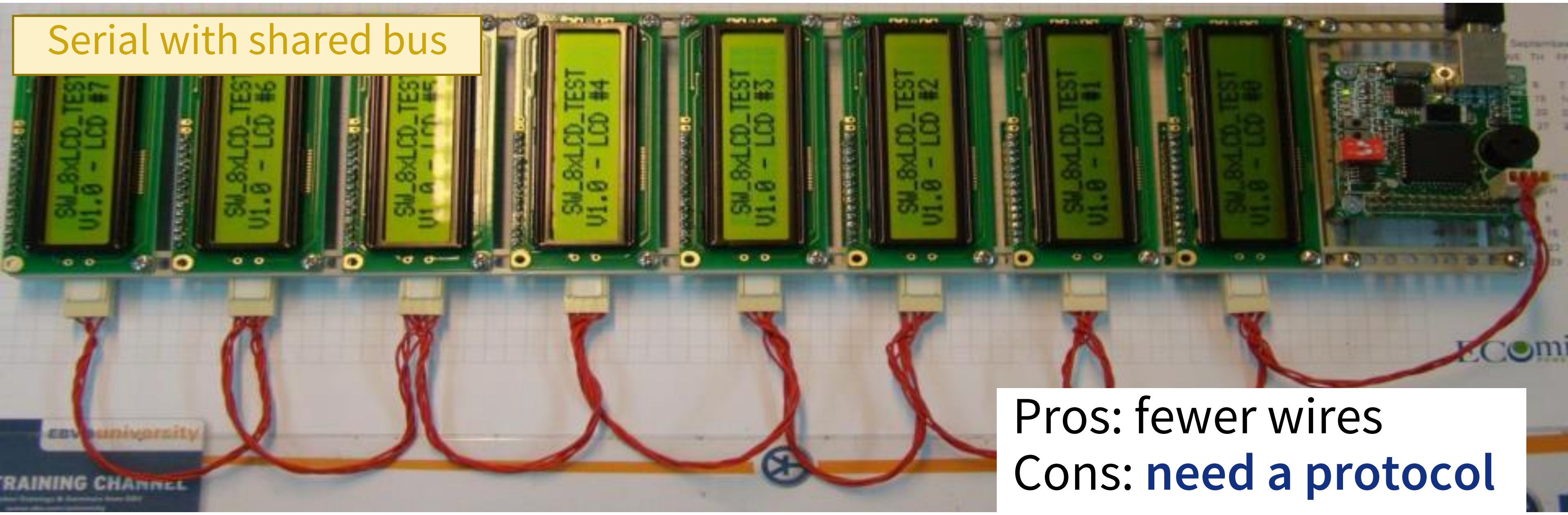
Last updated: 15 May 2020

Suppose you want to connect and control 4-8 LCD displays using only one microcontroller, which way do you prefer?



Parallel

Pros: straight-forward
Cons: many wires

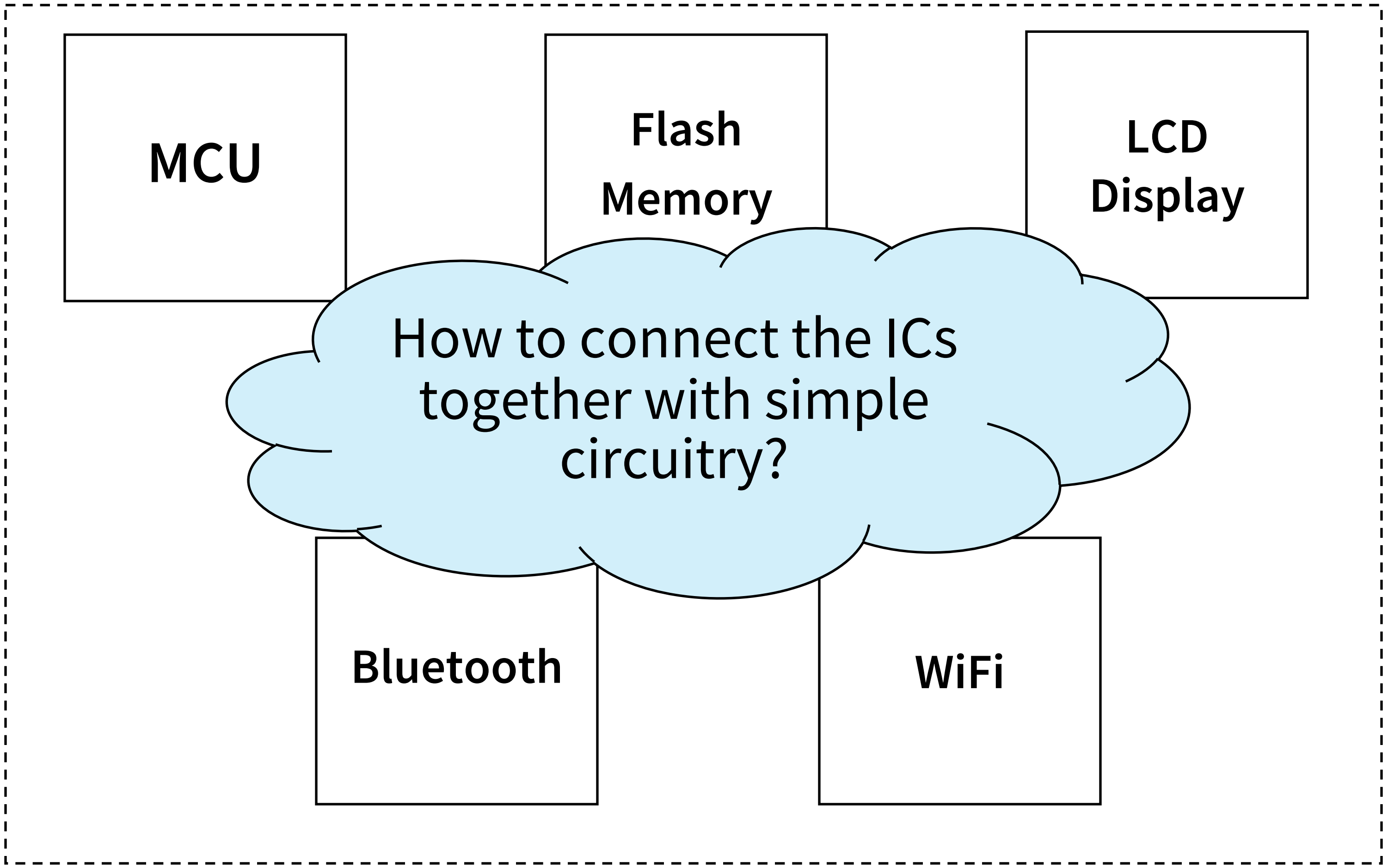

SW_8xLCD_TEST V1.0 - LCD #7
SW_8xLCD_TEST V1.0 - LCD #6
SW_8xLCD_TEST V1.0 - LCD #5
SW_8xLCD_TEST V1.0 - LCD #4
SW_8xLCD_TEST V1.0 - LCD #3
SW_8xLCD_TEST V1.0 - LCD #2
SW_8xLCD_TEST V1.0 - LCD #1
SW_8xLCD_TEST V1.0 - LCD #0

Serial with shared bus

Pros: fewer wires
Cons: **need a protocol**

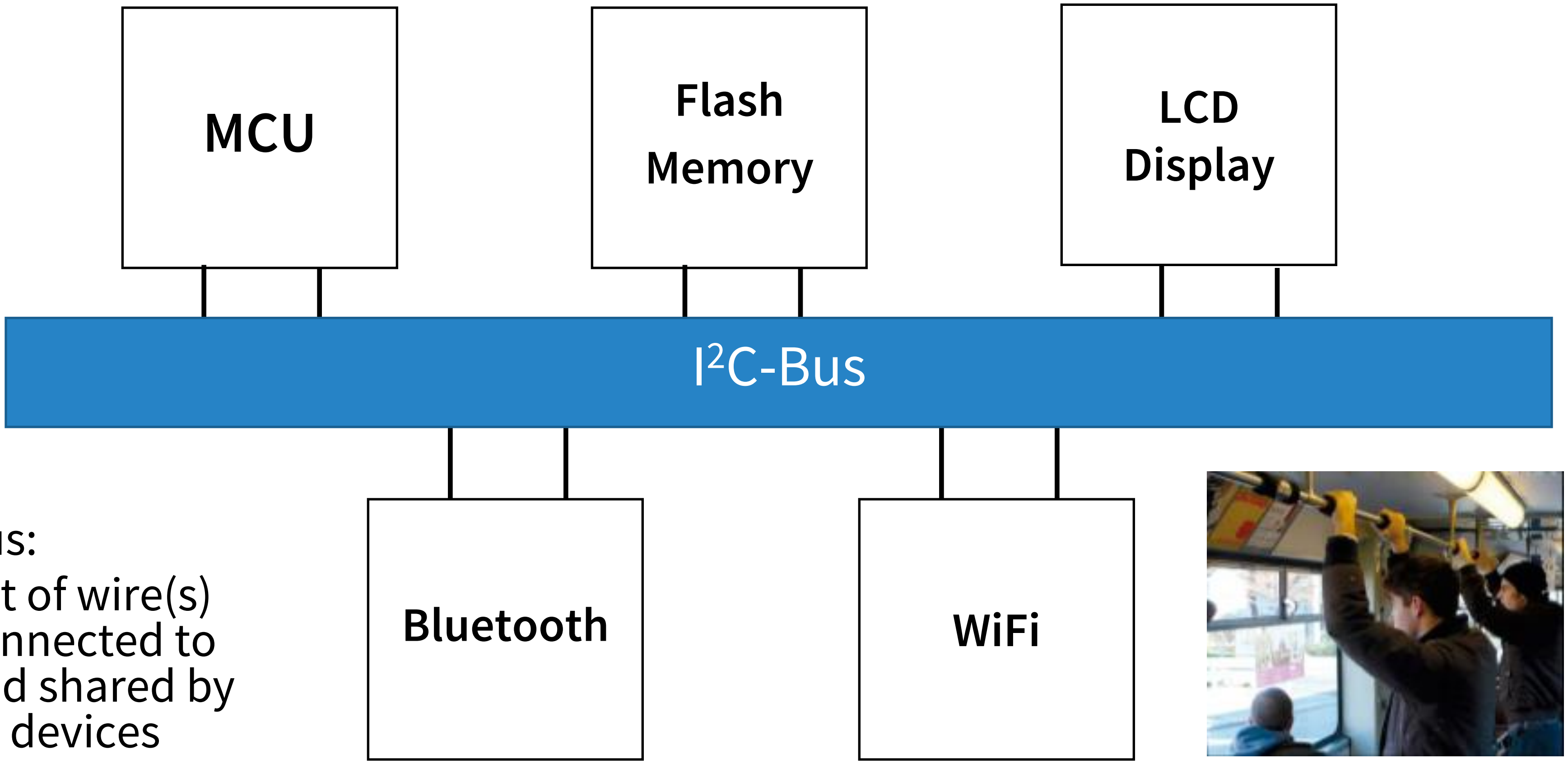
The Device Connection Problem

A small embedded system



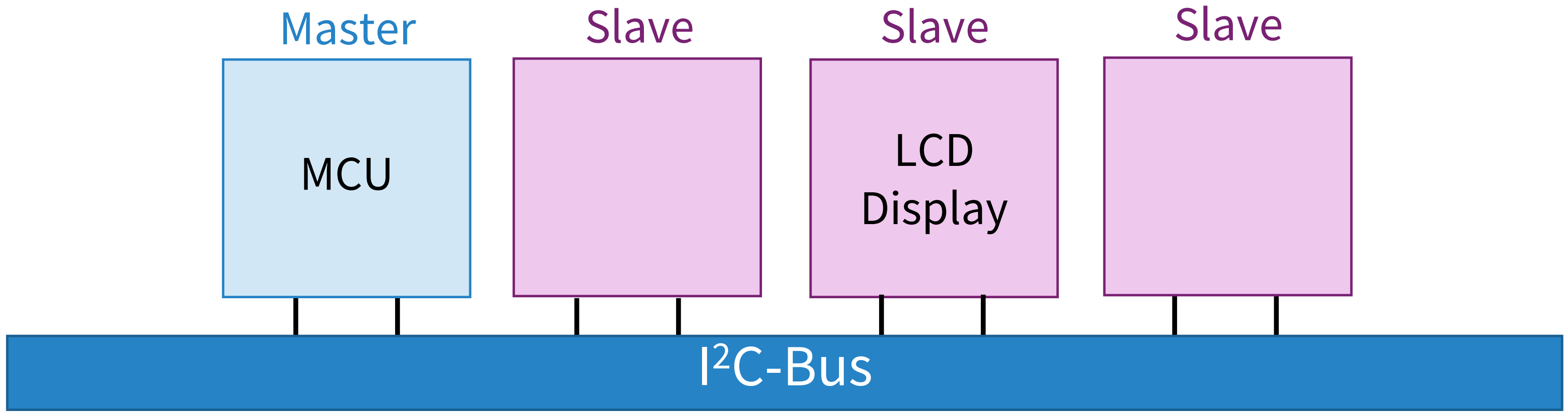
I²C: a Bus-oriented Solution

I²C interface & protocol: make sure messages can be sent from a device to another selected target



Bus:
Set of wire(s) connected to and shared by all devices

Typical I²C Configuration



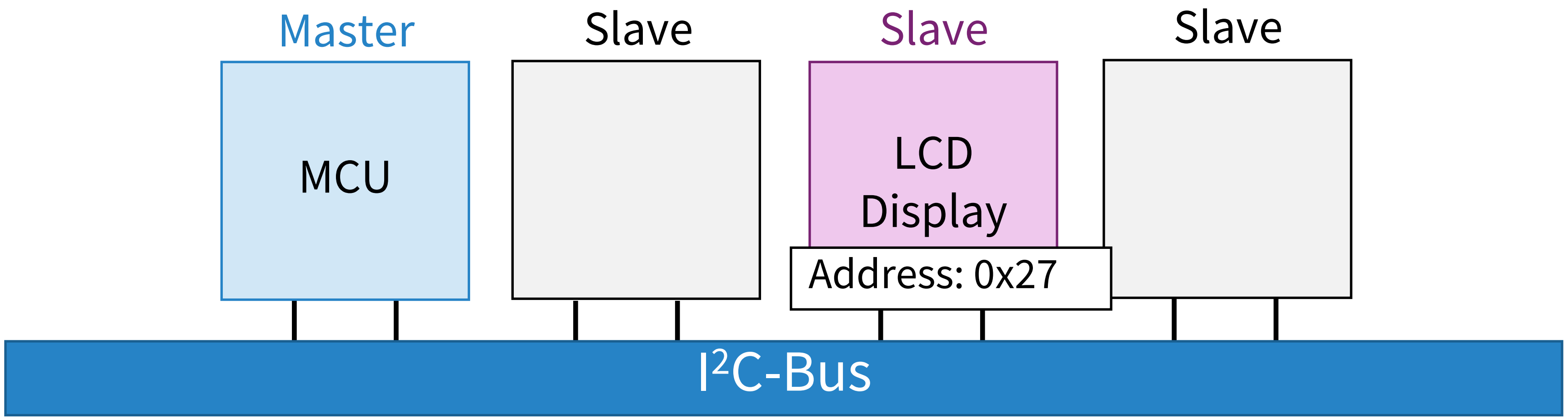
Active Role: Master

- Initializes a transfer
- Generates clock signals
- Send / ask for data
- Terminates a transfer

Passive Role: Slave

- Keep listening to the bus
- Wait to be addressed by the master
- Receive / send data

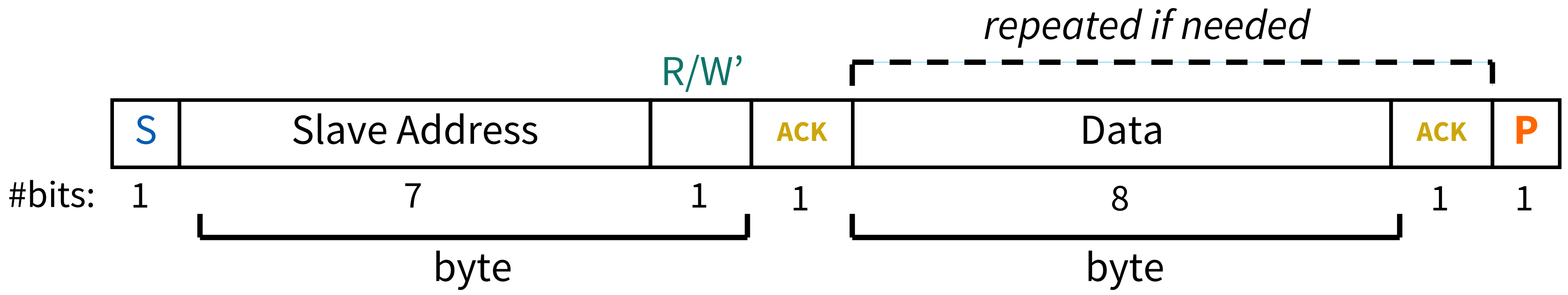
I²C: Transfer Procedure



Assume the LCD display is assigned an address 0x27.
If the microcontroller wants to send data to the LCD Display:

- MCU starts a transfer.
- MCU addresses LCD at 0x27.
- MCU states to LCD that this transfer is a WRITE.
- LCD acknowledges the calling from MCU.
- MCU sends data to LCD (master-transmitter & slave-receiver).
- LCD acknowledges the data received.
- MCU stops the transfer.

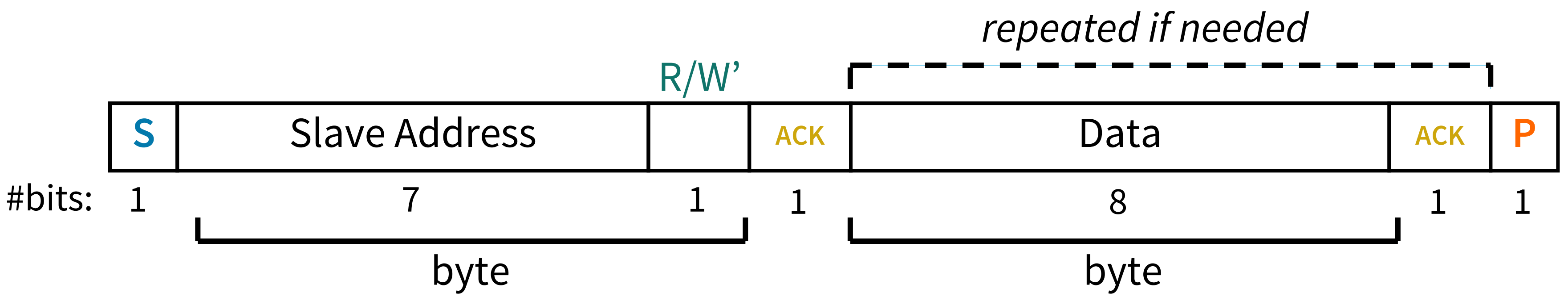
I^{2}C: Transfer Protocol



Assume the LCD display is assigned an address 0x27.
If the microcontroller wants to send data to the LCD Display:

- MCU starts a transfer.
- MCU addresses LCD at 0x27.
- MCU states to LCD that this transfer is a WRITE.
- LCD acknowledges the calling from MCU.
- MCU sends data to LCD (master-transmitter & slave-receiver).
- LCD acknowledges the data received.
- MCU stops the transfer.

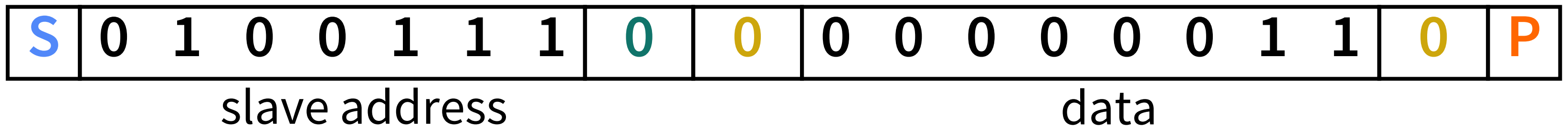
I²C: Transfer Format



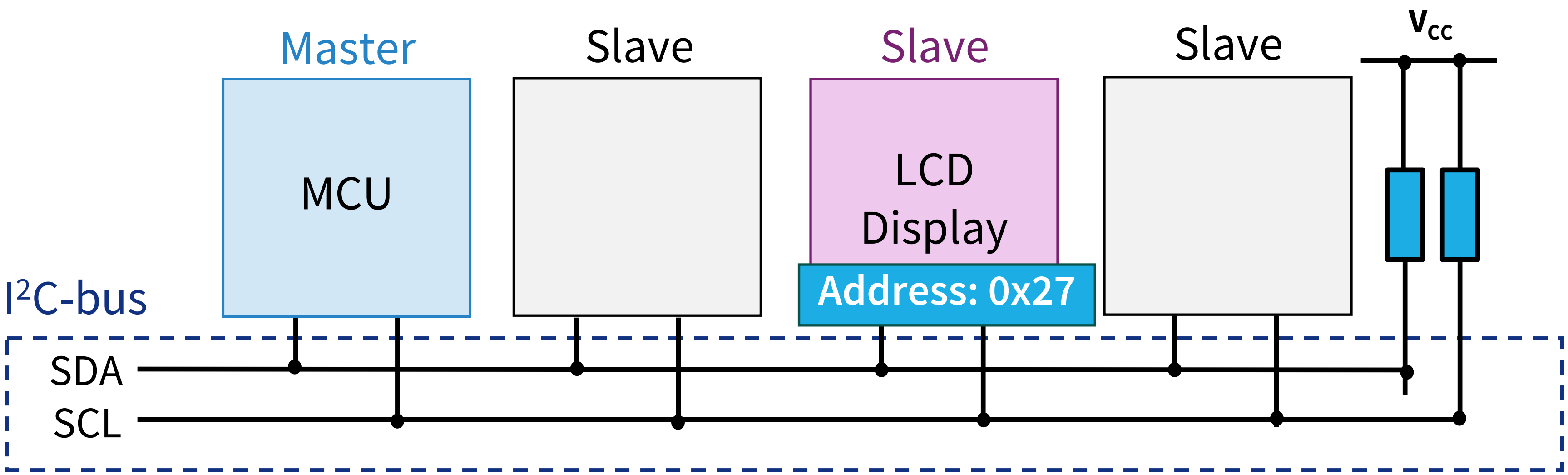
Signal	Meaning	Specification
S	START	1→0
P	STOP	0→1
R/W'	Read / Write	0: Write / 1: Read
ACK	Acknowledge	0: ACK / 1: NACK

- Slave address is 7 bits long.
- Theoretically, at most 128 devices can be attached to the same bus.
- 8-bit oriented transfer

Example: MCU sends a byte 0x03 to LCD (slave address: 0x27)



I^2C Wires: SDA & SCL

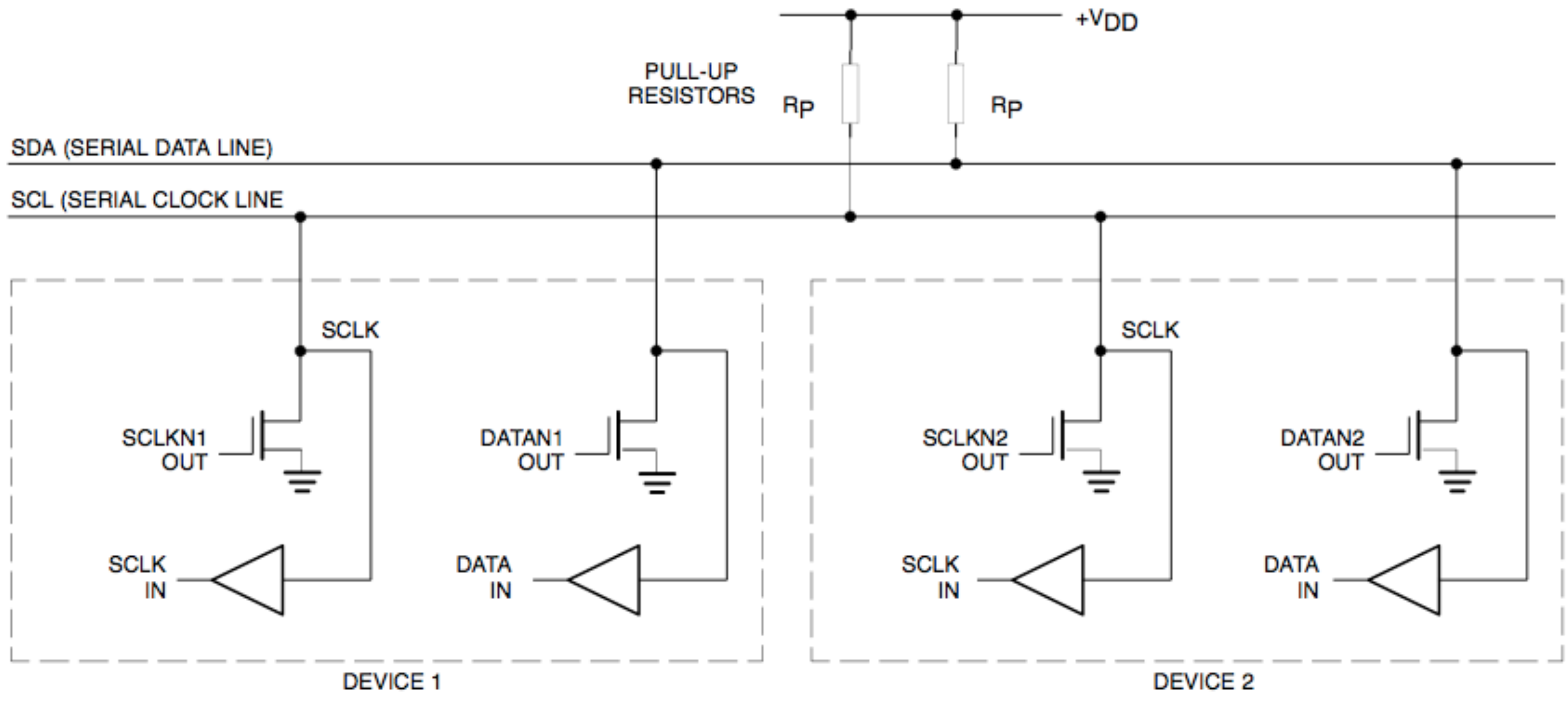


- Serial communications - ONE bit is sent at a time.
- There are 3 elements in serial comm.: START, BIT, STOP
- SDA: Serial data - transmits actual address/data bits.
  SCL: Serial clock - synchronises the data.
  - Both are pulled up to supply V_{CC} ('1') through resistors.
  - Can be pulled down to Gnd ('0') by the devices.

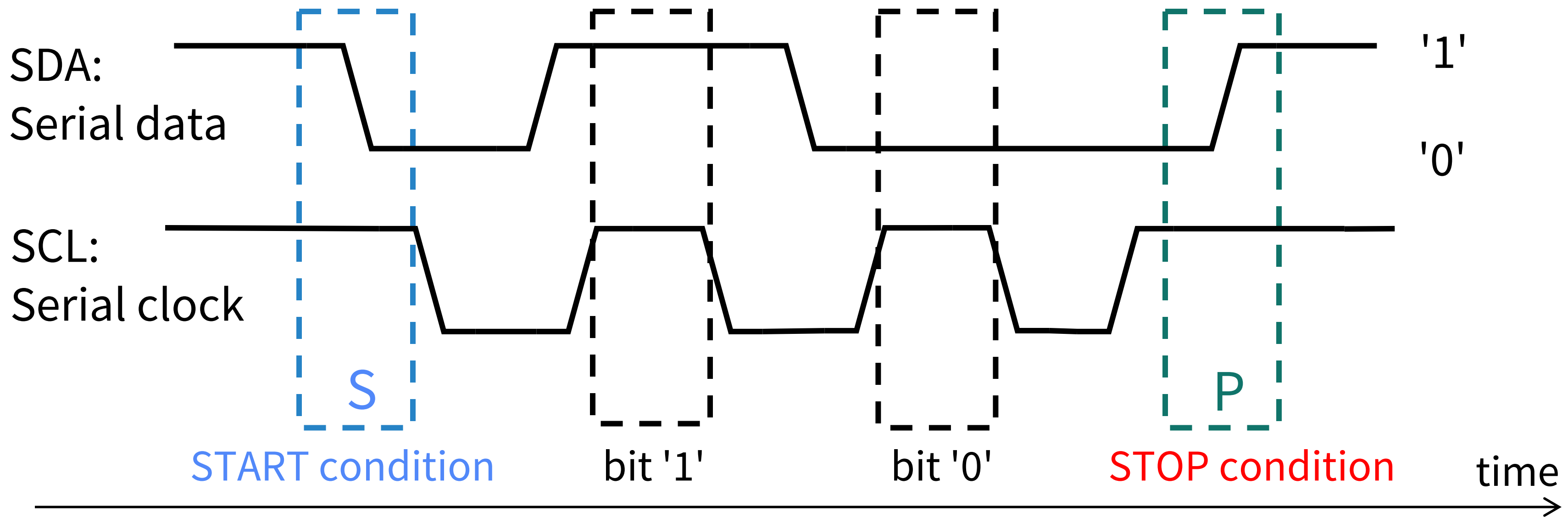
I^{2}C Electrical Details

Both SDA and SCL are bi-directional.

- Uses open collector/drain outputs
- Lines float HIGH unless an output goes LOW.



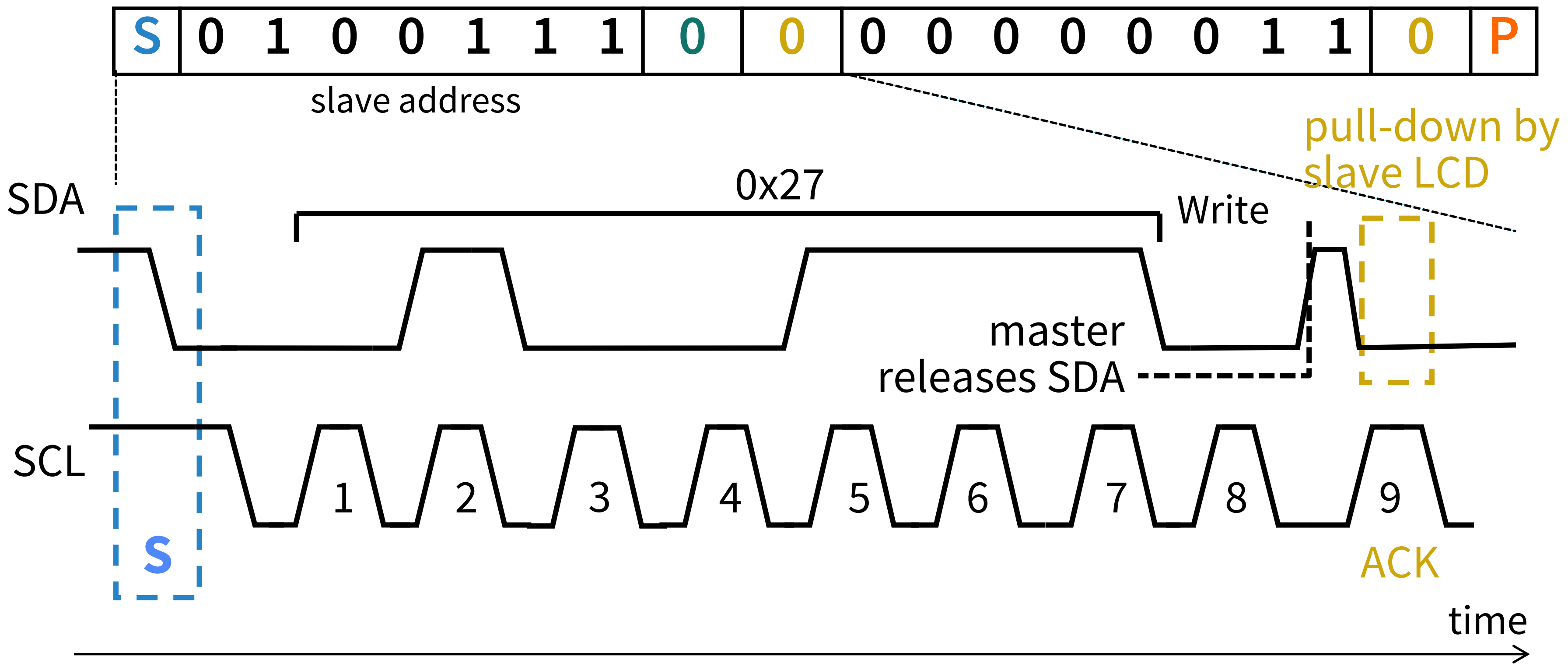
SDA & SCL: START, BIT, STOP



- START and STOP must be generated by the master.
- Data bit is valid only when SCL is '1'.
- SCL is also generated by the master.
  - The clock frequency determines the data transfer rate.

Signals on SDA & SCL

Example: MCU sends a byte 0x03 to LCD (slave address: 0x27)



Would you be able to continue this timing diagram for the second half of this transfer?

Acknowledge Signal (ACK)

- ACK takes place after every byte is sent.
  - It signals to the master/transmitter that the byte was successfully received.
- Transmitter releases the SDA line so the receiver can pull the SDA line LOW ('0').
- If SDA remain '1', it is a Not acknowledge signal (NACK). Possible scenarios:
  - No receiver on the bus
  - The receiver is unable to receive or transmit (not ready)
  - During the transfer, the receiver does not understand the data / cannot receive any more data bytes.
  - A master-receiver uses NACK to signal the end of the transfer to the slave-transmitter (then generates a STOP).

Recap - Features of I^{2}C

- All devices attached to the same bus, taking roles of either master or slave.
- A serial, 8-bit oriented, bi-directional (read/write) data transfer protocol is defined.
- Each device is assigned a 7-bit unique address.
- There are only two bus lines - SDA & SCL.

These features enable designers to construct a single-master system with several fast slaves.

However, can we do even better?

Advanced Features in I^{2}C

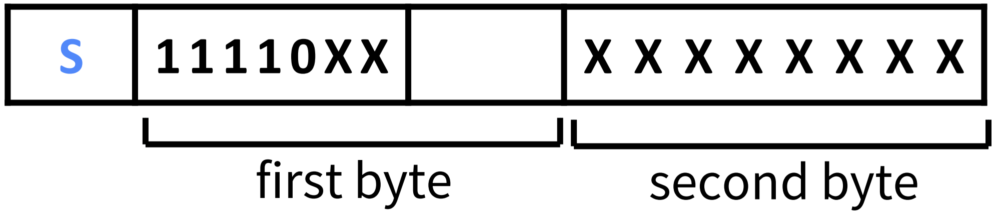
I^{2}C: Pros / Cons

- ☑ Simple and easy to implement
- ☑ ICs can be attached or detached without affecting other circuits.
- ☑ There are fewer I/O pins & fewer PCB tracks.

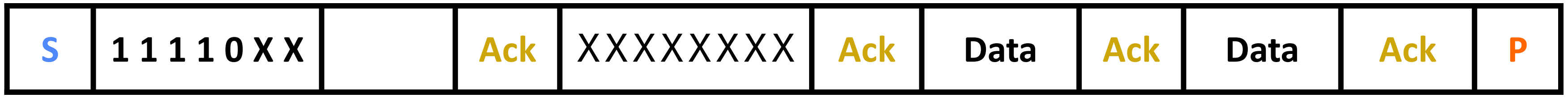
- ☒ The slowest I^{2}C device dominates your bus performance, especially clock is often stretched.
- ☒ The 7-bit slave addresses allows (theoretically) at most 128 devices on the same bus.
- ☒ Since I^{2}C is a shared bus, a faulty device hangs the entire bus. *e.g.* it keeps pull SDA LOW ('0') and no START/STOP can be generated by the master.

10-bit Addressing

- 10-bit addressing expands the possible addresses.
- Devices using 7-bit or 10-bit addressing can be mixed on the same bus.
- The 10-bit slave address is formed from the first two bytes following a START condition (S) or a repeated START condition (Sr).



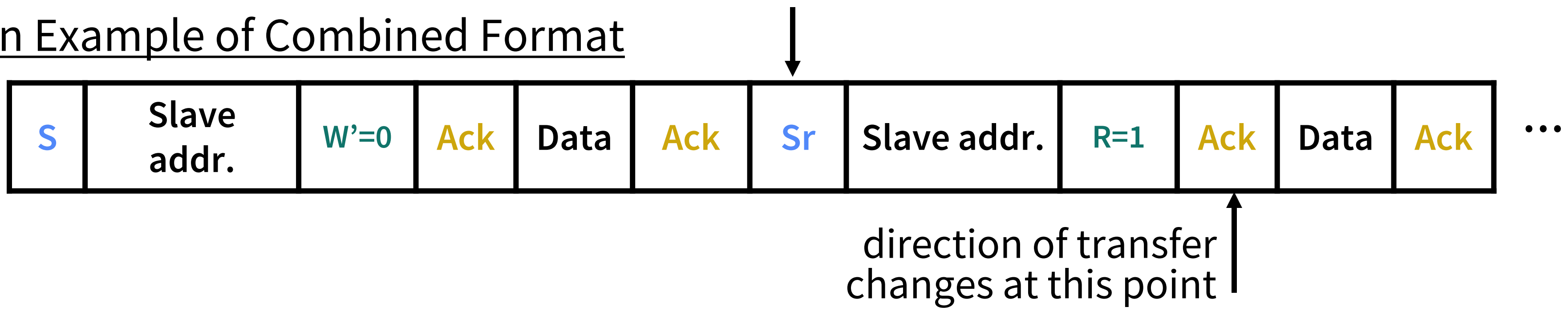
Example: a master writing data to a slave in 10-bit addressing



Combined Format & Repeated Start

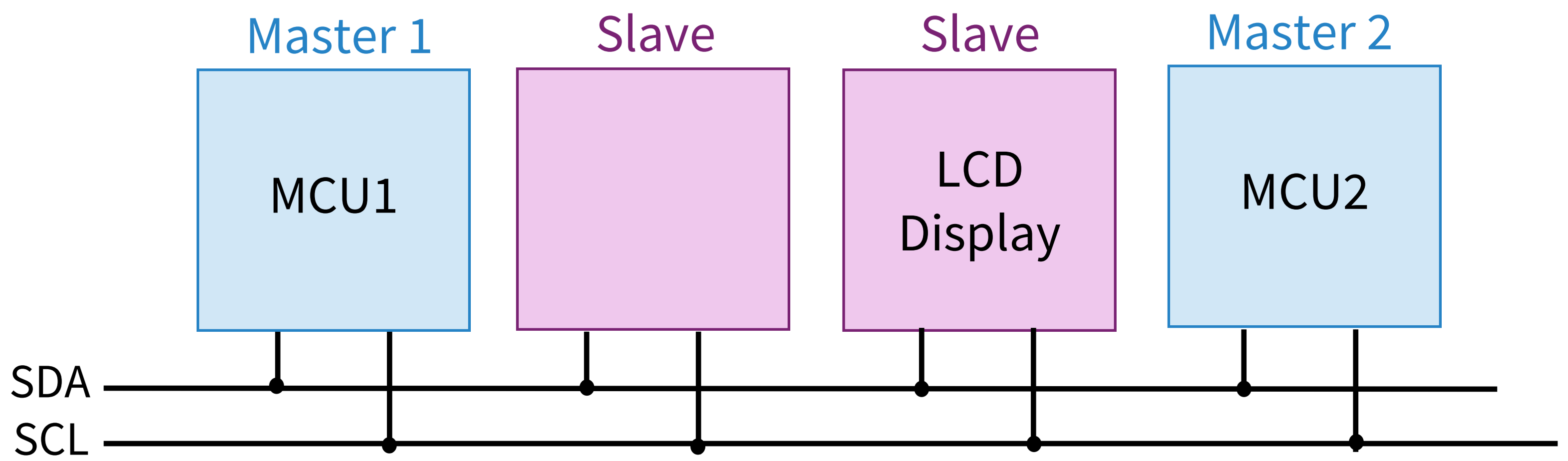
- If the master wants to change the direction of transfer, then it simply addresses the slave again using a new R/W' with another START signal. (instead of a STOP).
- This is usually called the "repeated START" (Sr).

An Example of Combined Format



A master-transmitter addresses and sends a byte to a slave-receiver.
Then it generates a repeated START and change the direction.
The slave then becomes a transmitter and sends data afterwards.

More Practical Issues in I²C



Q1: What if the slave is not fast enough to process the data and want to ask the master to wait until it is ready?

Q2: What if two (or more) masters (M1 & M2) begin a transfer at the same time? Which can get the I2C bus access?

Advanced I^{2}C Features - Solutions

Q1: What if the slave is not fast enough to process the data and want to ask the master to wait until it is ready?

So far there is no protocol allowing a slave to ask the master to wait or pause a transfer.
Solution: clock stretching

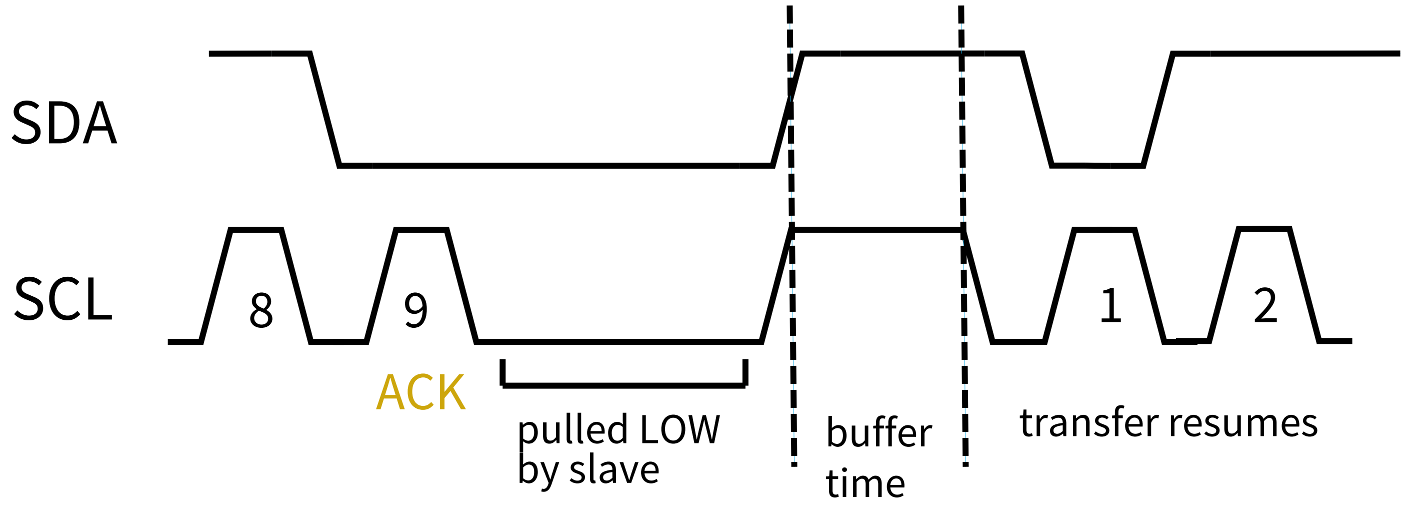
Q2: What if two (or more) masters (M1 & M2) begin a transfer at the same time? Which can get the I2C bus access?

The bus is even more usable if we can put several masters together, controlling a set of slaves.
Solution: collision detection & bus arbitration

Clock Stretching

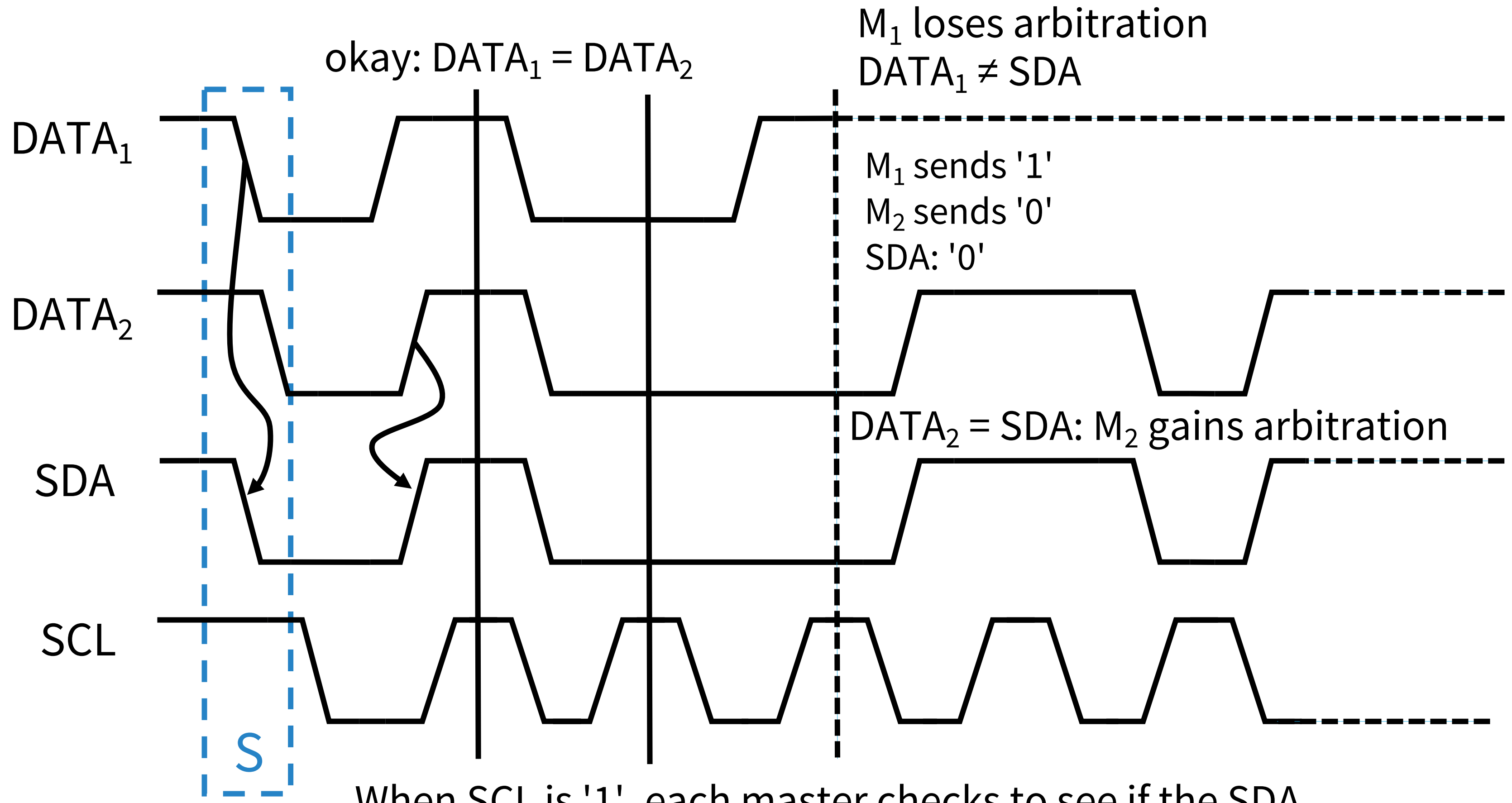
Q1: What if the slave is not fast enough to process the data and want to ask the master to wait until it is ready?

Clock stretching: slave pulls down SCL to '0' while it is not ready for more data.
Master has to wait until SCL = '1' again, and continues to send data after an additional minimum buffer time (say 4 µs).



Bus Arbitration

Q2: What if two (or more) masters (M1 & M2) begin a transfer at the same time? Which can get the I2C bus access?



When SCL is '1', each master checks to see if the SDA level matches what it has sent.

REFERENCE

Brief History of I^{2}C

1982: Philips inverted the original 100-kHz I^{2}C.

1992: rev. 1.0 released. 400-kHz Fast-mode & 10-bit addr added.

mid-1990s: Various non-Philips I^{2}C products appeared on the market

1998-2000: rev. 2.0 & 2.1 released. 3.4-MHz High-speed mode added.

2006: no licensing fees are required to implement the I^{2}C protocol

2007: rev. 3.0 released. 1-MHz Fast mode plus added.

2012: rev. 4.0 & 5.0 released. 5-MHz Ultra-fast mode added.

REFERENCE

Reserved Address

The following addresses are reserved:

Slave Address	bit	Description
0000 000	0	General call address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	reserved for different bus format
0000 011	X	reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	1	device ID
1111 0XX	X	10-bit slave addressing

If it is known that the reserved address is never going to be used for its intended purpose, a reserved address can be used for a slave address.

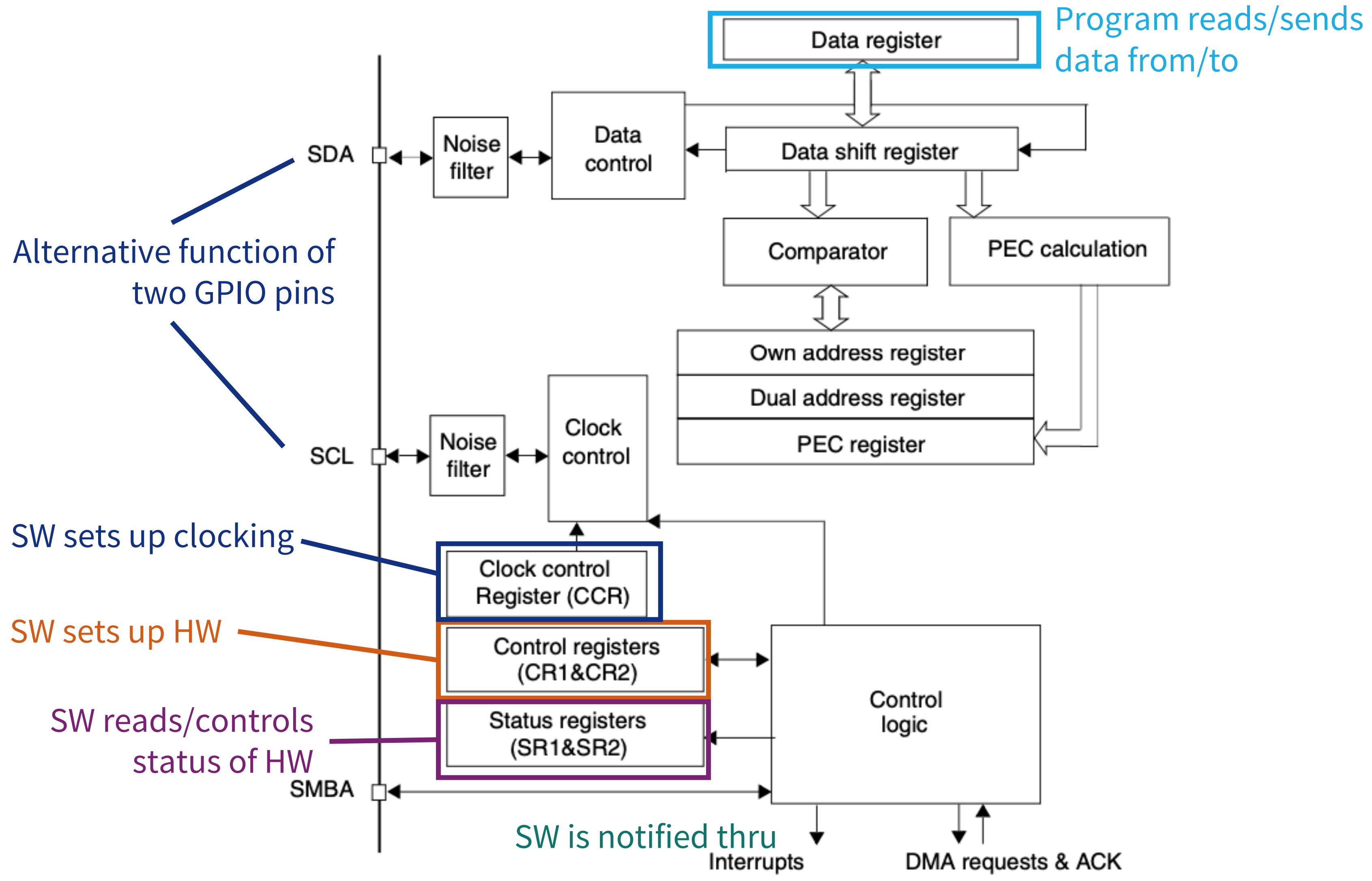
REFERENCE

General Call

- Address "00000000" is reserved for general call: broadcasts to all devices connected to the bus.
- A slave device may:
  - ignore this general call with NACK (do nothing)
  - respond to the call and acknowledge, and become a slave-receiver.
- If more than one devices acknowledge, the master doesn't know the total number. (wired-AND)
- The second byte in this general call determines the action requested by the master, including
  Software reset, START byte (for MCU transfer), Bus clear, Device ID

Programming I^{2}C in Cortex M3/M4

I^{2}C Block Diagram

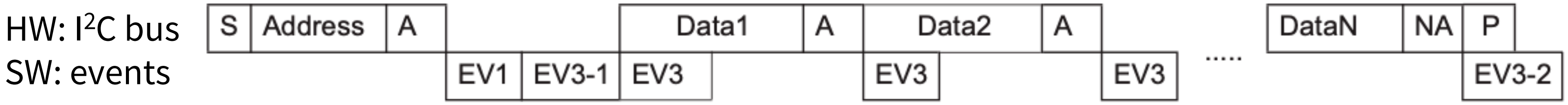


Slave Mode

- On default, the I^{2}C interface works in slave mode.
- The peripheral input clock must be programmed in the I2C_CR2 register in order to generate correct timings.
  - The peripheral input clock frequency must be at least 2 MHz in standard mode (up to 100 kHz SCL)
- The hardware interface automatically waits for a START signal and address from the master.
- It match the slave address in OAR1. When matched it generates:
  - an acknowledge pulse if the ACK bit is set
  - the ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.
- The TRA bit indicates whether the slave is in Receiver or Transmitter mode.

Slave Mode - Transmitter

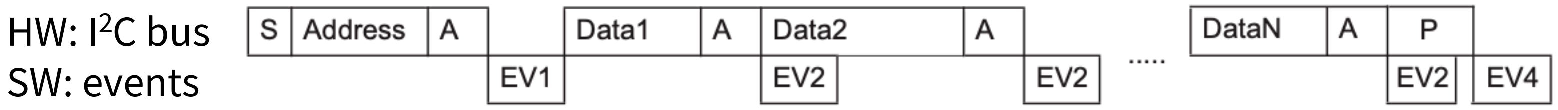
- The HW sends bytes from the DR register to the SDA line via the internal shift register.
- It stretches SCL low until ADDR is cleared and DR filled with the data to be sent (during EV1 and EV3-1 below).
    - This forces the master to wait for the data.
- When acknowledge pulse (from master) is received:
    - The TxE bit is set by hardware with an interrupt if the ITEVFEN and the ITBUFEN bits are set.



EV1: ADDR=1 – Slave address matched
EV3-1: TXE=1 – DR transmit empty, SW writes Data1 into DR
EV3: TxE=1 – DR transmit empty, shift register transmitting
EV3-2: AF=1 – Acknowledge failed, transmission ended

Slave Mode - Receiver

- The HW receives bytes from the SDA line into the DR register via the internal shift register.
- After each byte the interface generates in sequence:
  - An acknowledge pulse if the ACK bit is set
  - The RxNE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bit is set.
- If RxNE is set and the data in the DR register is not read before the end of the next data reception, the BTF bit is set and the HW waits until BTF is cleared by a read from the I2C_DR register, stretching SCL low.



EV1: ADDR=1 – Slave address matched
EV2: RxNE=1 – DR receiver not empty (i.e. data received)
EV4: STOPF=1 – Slave stopped detected (i.e. no more data)

After the last data byte, a Stop Condition is generated by the master.
The HW detects this condition and sets the STOPF bit and generates an interrupt if the ITEVFEN=1.

Master Mode

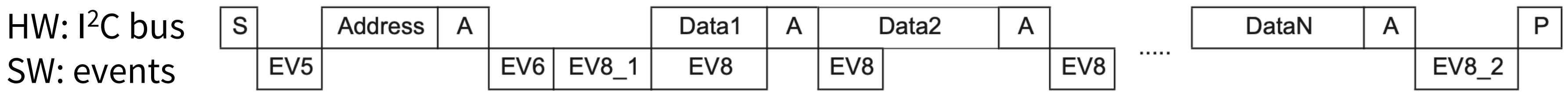
- HW initiates a data transfer and generates the clock signal, with an initialisation sequence:
    - Program the peripheral input clock in I2C_CR2
    - Configure the clock control registers
    - Configure the rise time register
    - Program I2C_CR1 register to enable
    - Set the START bit in I2C_CR1 (see below)
- HW checks the SCL for any stretching.
- Setting the START bit causes HW to generate a Start (a ReStart) and switch to Master mode.
    - SB bit is set by HW and an interrupt is generated if ITEVFEN=1.

Master Mode (Cont')

- Then it waits for a read of the SR1 (from SW) and then a write to DR with the slave address.
    - HW enters transmitter mode if R/W' (LSB in DR) = 0;
    - HW enters receiver mode otherwise
- As soon as the address byte is sent, the ADDR bit is set by hardware and an interrupt is generated if ITEVFEN=1.

Master Mode - Transmitter

- SW clears ADDR and load data into DR.
- HW then sends bytes to SDA.
- When slave sends an ACK, TxE bit is set by hardware and an interrupt is generated if ITEVFEN=ITBUFEN=1.
- SW sets STOP bit to generate a STOP signal.
- HW then goes back to slave mode.

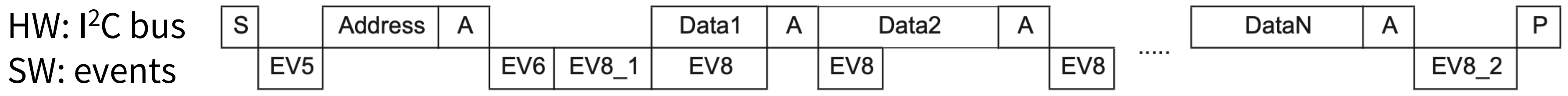


EV5: SB=1 – master mode selected
EV6: ADDR=1 - master /**transmitter** mode selected
EV8_1: TxE=1 - DR transmitter empty, SW writes Data1 in DR
EV8: TxE=1 – DR transmitter empty (shift register transmitting)
EV8_2: TxE=1, BTF=1 - master byte transmitted.

Master: i2c_write()

Study the basic polling version of the master transmitter in C code using a number of functions from CMSIS. This assumes I2C1 is used.

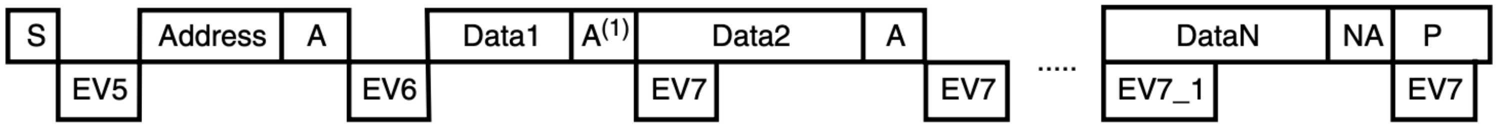
```c
void i2c_write(uint8_t address, uint8_t *buffer, int buff_len) {
    int i = 0;
    // Send in sequence: Start bit, Contents of buffer 0..buff_len, Stop
    while (((I2C1->SR2>>1)&1)); // wait until I2C1 is not busy anymore
    I2C_GenerateSTART(I2C1, ENABLE); // Send I2C1 START condition
    // wait for I2C1 EV5 --> Slave has acknowledged start condition
    while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));
    // Send slave Address for write then wait for EV6
    I2C_Send7bitAddress(I2C1, address, I2C_Direction_Transmitter);
    while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
    while (i < buff_len){
        I2C_SendData(I2C1, buffer[i]); // send data then wait for EV8_2
        while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
        i++;
    }
    I2C_GenerateSTOP(I2C1, ENABLE);    // send stop bit
}
```



Master Mode - Receiver

- HW receives bytes from the SDA line into DR via the internal shift register.
- After each byte the interface generates in sequence:
  - An acknowledge pulse if the ACK bit is set
  - The RxNE bit is set and an interrupt is generated if ITEVFEN=ITBUFEN=1.
- Waits for reads from SW: if RxNE=1 and the data in the DR register is not read, the BTF bit is set and the interface waits until BTF is cleared by a read in the DR register, stretching SCL low.

HW: I^{2}C bus
SW: events



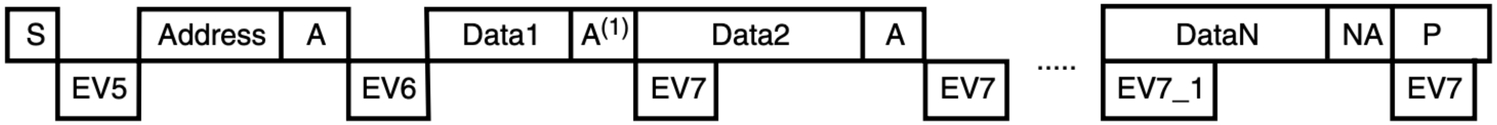
EV5: SB=1 - master mode selected
EV6: ADDR=1 – master **receiver**/transmitter mode selected
EV7: RxNE=1 – data register receiver not empty (byte received)
EV7_1: RxNE=1 - master byte received. SW programs NACK and STOP request.

Master: i2c_read()

Study the basic polling version of the master receiver in C code using a number of functions from CMSIS.

```c
void i2c_read(uint8_t address, uint8_t *buffer, int buff_len) {
    int i = 0;
    //  Start bit, Contents of buffer from 0..buff_len, sending a NACK
    //  for the last item and an ACK otherwise, Stop bit
    I2C_GenerateSTART(I2C1, ENABLE);
    while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT)); //EV5
    // Send slave Address for write then wait for EV6
    I2C_Send7bitAddress(I2C1, address, I2C_Direction_Receiver);
    while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
    I2C_AcknowledgeConfig(I2C1, ENABLE);    // going to send ACK
    while (i < buff_len - 1){
        while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)); //EV7
        buffer[i] = I2C_ReceiveData(I2C1); // get data byte
        i++;
    }
    I2C_AcknowledgeConfig(I2C1, DISABLE);  // going to send NACK
     while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED)); //EV7
    buffer[i] = I2C_ReceiveData(I2C1);     // get the last byte
    I2C_GenerateSTOP(I2C1, ENABLE);        // send stop
}
```

HW: I^{2}C bus
SW: events



Error Conditions

A quick summary on possible errors raised by I^{2}C interface HW.

Error	Code	Causes
Bus error	BERR	HW detects an external Stop or Start condition during an address or a data transfer.
Acknowledge failure	AF	HW detects a non-acknowledge bit.
Arbitration lost	ARLO	HW detects an arbitration lost condition.
Overrun/underrun error	OVR	An overrun error can occur in slave mode when clock stretching is disabled and the I^{2}C interface is receiving/transmitting data.

In all cases, the corresponding error bit is set and an interrupt will be generated (if interrupt error enable: ITERREN=1).

I^{2}C Interrupts

This table gives the list of I2C interrupt requests.

Event	Flag	Enable Control Bit
Start bit sent	SB	ITEVFEN
Address sent/matched	ADDR	
Stop received	STOPF	
Data byte transfer finished	BTF	
Received buffer not empty	RxNE	ITEVFEN and ITBUFEN
Transmit buffer empty	TxE	
Errors	BERR/ARLO/AF/OVR	ITERREN

Event interrupt

Error interrupt

References

- UM10204 I2C-bus specification and user manual (Rev 5.0) http://www.nxp.com/documents/user_manual/UM10204.pdf
- Wikipedia: http://en.wikipedia.org/wiki/I%C2%B2C
- http://www.i2c-bus.org/
- http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

Sources of photos/images:
http://www.logotypes101.com/free_vector_logo/34736/i2c_Bus
http://www.jechavarria.com/wp-content/uploads/2013/03/DSC01259.jpg
http://www.hackmeister.dk/wp-content/uploads/2010/08/quad_lcd_arduino.png
http://1.bp.blogspot.com/_NpINLHeo8rM/TBxsL86jCCI/AAAAAAAAzpw/I4K9ZaTWQxI/s1600/1.jpg