

Microcontrollers

Team Emertxe



Contents



Microcontrollers

Contents

- Introduction to Embedded Systems
- Introduction to Microcontrollers
- Embedded Programming
- Communication Protocols I



Introduction



Introduction



- What is a Microcontroller
- General Architectures
- μP vs μC
- Choosing a Microcontroller



Introduction

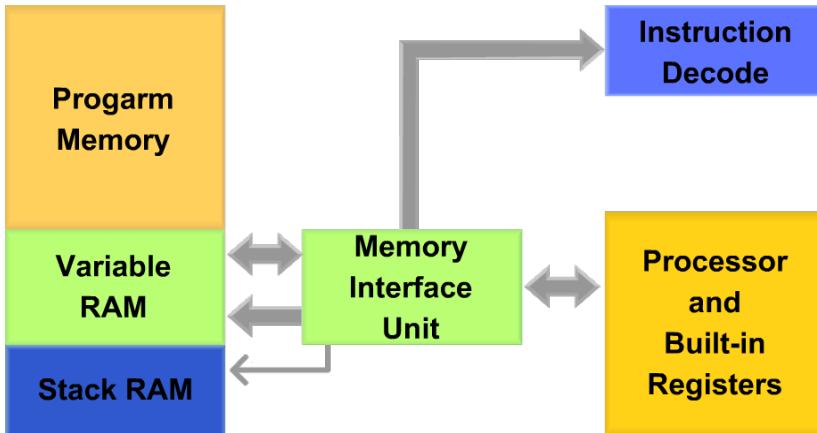
What is a Microcontroller?

- An Integrated Circuit which is capable of being programmed to perform a specific task.
- The design normally has restrictions on its
 - Memory Size
 - I/O Capabilities
 - Peripheral Functions etc.,

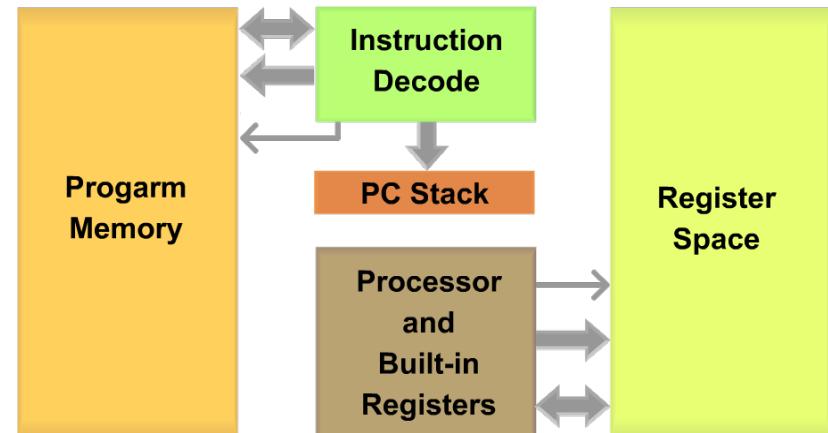


Introduction

General Architectures



Von Neuman Architecture



Harvard Architecture

- Shared signals and memory for code and data

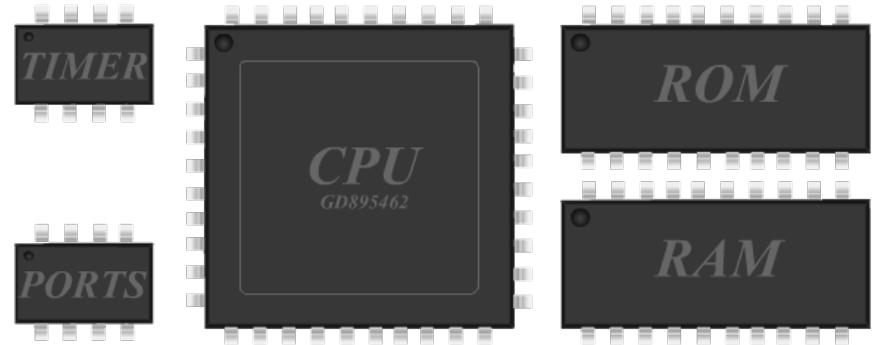
- Physically separate signals and storage for code and data



Introduction

μ P vs μ C - Microprocessors

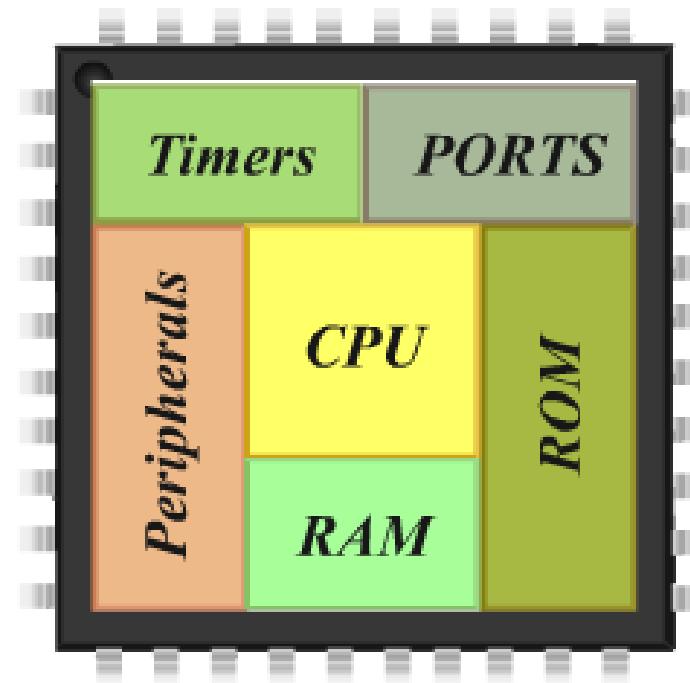
- All separate components
- More flexible
- More design complexity



Introduction

μ P vs μ C - Microcontroller

- All components in single chip
- Less flexible
- Less design complexity



Introduction

Choosing a Microcontroller



- Applications
- Performance
- Price
- Availability
- Availability of Tools
- Special Capabilities



Thank You

Embedded Systems

Introduction

Team Emertxe



Contents



Embedded Systems

Contents

- Introduction to ES
- GPS vs ES
- Real Time Aspects



Introduction to Embedded System

Embedded Systems

Introduction



- What is ES
- Examples
- Categories
- Components
- Requirements
- Challenges
- Trends in Development
- Common Design Metrics



Embedded System

Definition

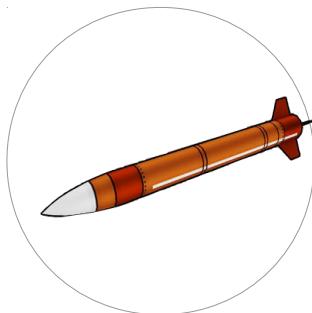
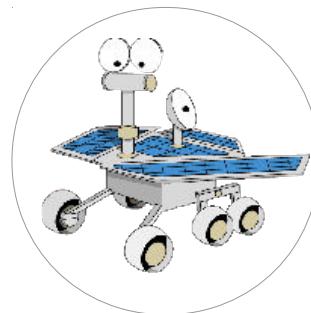
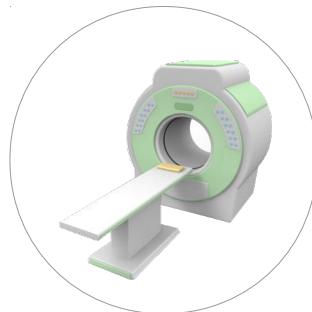


“Any combination of **Hardware** and **Software**
which is intended to do a
Specific Task
can be called as an **Embedded System**”



Embedded Systems

Examples



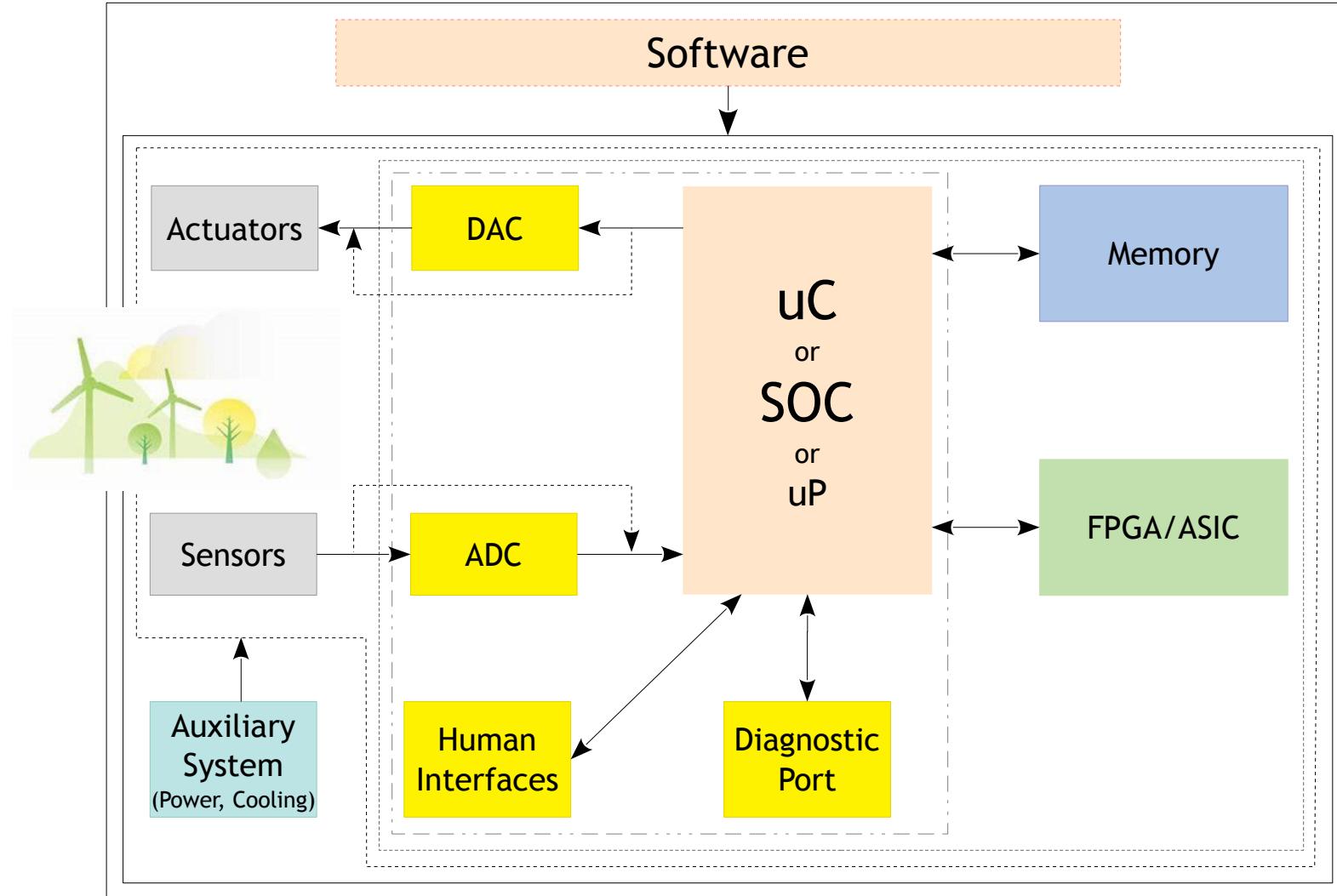
Embedded System

Categories

- Stand-alone
- Real Time
- Networked
- Mobile



Embedded System Components



Embedded System Requirements

- Reliability
- Cost-effectiveness
- Low Power Consumption
- Efficient Usage of Processing Power
- Efficient Usage of Memory
- Appropriate Execution Time



Embedded System Challenges

- Efficient Inputs/Outputs
- Embedding an OS
- Code Optimization
- Testing and Debugging



Embedded System

Trends in Development

- Processors
- Memory
- Operating Systems
- Programming Languages
- Development Tools



Embedded System

Common Design Metrics



- Time to Prototype
- Power
- Performance & Correctness
- Size
- NRE
- Maintainability & Flexibility
- Safety
- Unit Cost
- Time to Market



GPS vs ES



Embedded System

GSP vs ES



- What do you think of your Desktops?
- Does the size matter?
 - Bluetooth Button
 - Industrial Control Systems



Real Time Aspects



Embedded System

Real Time Aspects



- Hard Real Time
 - Should meet its deadline - Life Critical Application
- Firm Real Time
 - Similar to Hard Real Time - Properties
- Soft Real Time
 - Can have tolerance in meeting its deadline



Thank You

Microcontrollers

Empedded Programming - RL-eCee Dev Board

Team Emertxe



Important Terms



Microcontrollers

Important Terms



- Host:

A system which is used to develop the target.

- Target:

A system which is being developed for specific application.

- Cross Compiler:

An application used to generate code for another architecture being in another architecture



Workspace Creation



Microcontrollers

Workspace Creation

- For better data organization lets create some project working directories
- Please follow the below steps

Step 1

```
user@user:~] cd
```

Step 2

```
user@user:~] mkdir -p ECEP/Microcontrollers/ClassWork
```

Step 3

```
user@user:~] cd ECEP/Microcontrollers/ClassWork
```



Lets Start Coding



Microcontrollers

Embedded Programming - Let's Start Coding



- Well, come on lets make our hand a bit dirty in embedded coding with the following code

Example

```
#include <stdio.h>

int main()
{
    int x = 20;

    printf("%d\n", x);

    return 0;
}
```

- Nice, but few questions here
 - Why did you write this code?
Hmm, Just to say hello world to embedded programming
 - Fine, where are you planning to run this code?

Of course on a embedded target!

- Does it have a OS already running?

Ooink, Hmm noo, may be ...



Microcontrollers

Embedded Programming - Let's Start Coding



- So questions raised in the previous slide has to be answered before we can start our code.
- The answers to these questions are little tricky and depends on
 - Complexity of the work you do
 - The requirement of the project and many other factors
- Now the scope of this module is to learn low level microcontroller programming which is non OS (called as bare metal)
- So let's rewrite the same example as shown in the next slide



Microcontrollers

Embedded Programming - Let's Start Coding

Example

```
#include <stdio.h>

void main(void)
{
    int x = 20;

    printf("%d\n", x);
}
```

- The change you observe is **void main(void)**
- Why?
 - As mentioned generally the low end embedded system are non OS based
 - The code you write would be the first piece of code coming to existence
- Now, lets not take this too seriously. This could again depend the development environment
- There could be some startup codes, which would call the main



Microcontrollers

Embedded Programming - Let's Start Coding



Example

```
#include <stdio.h>

void main(void)
{
    int x = 20;

    printf("%d\n", x);
}
```

- The next questions is, where are trying to print? On Screen?
 - Does your target support that?
 - Does your development environment support that?
- Now again, all these are depends on your target board and development environment
- Maaan, So many questions? Well, what should I write then?
 - Well that too depends on your target board!!



Microcontrollers

Embedded Programming - Let's Start Coding



- Well, my principle is simple. No matter on what type of board you work, the first code you write, should give you the confidence that you are on the right path.
- Try to identify the simplest possible interface which can be made work with lesser overhead, so that, we are sure about our setup like
 - Hardware is working
 - Toolchain setup is working
 - Connectivity between the host and target is establishedand so on.



Microcontrollers

Embedded Programming - Let's Start Coding

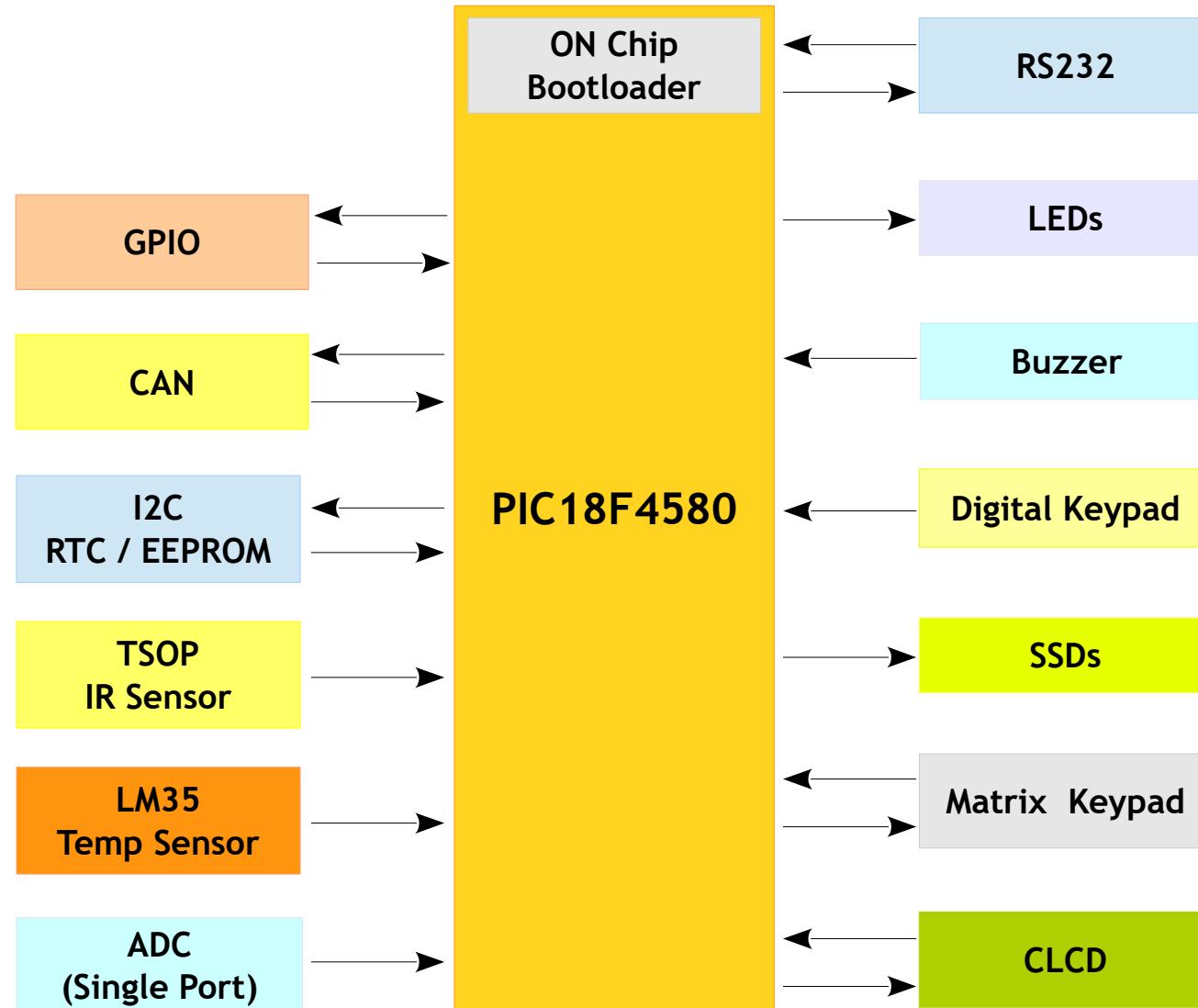


- It is good to know what your target board is, what it contains by its architecture
- Board architecture generally gives you overview about your board and its peripheral interfaces
- In our case we will be using RL-eCee development board which is shown in the next slide



Microcontrollers

EP - Let's Start Coding - RL-eCee Architecture



Microcontrollers

Embedded Programming - Let's Start Coding



- So from the architecture we come to know that the board has few LEDs, So why don't we start with it?

Example

```
#include <stdio.h>

void main(void)
{
    int led;
    led = 0;
}
```

- So simple right? Well I hope you know what's going happen with this code!!
- Any C programmer knows that the **led** is just a integer variable and we write just a value in it, hence no point in this code

Now what should we do?

- Hmm, refer next slide



Microcontrollers

Embedded Programming - Let's Start Coding



- LED is an external device connected to microcontroller **port**
- A **port** is interface between the controller and external peripheral.
- Based on the controller architecture you will have **N** numbers of **ports**
- The target controller in RL-eCee board is **PIC18F4580** from Microchip
- The next question arises is how do I know how many ports my target controller has?
 - From **Microcontroller Architecture** which will be detailed in the data sheet provided by the maker



Microcontrollers

Embedded Programming - Let's Start Coding



- By reading the data sheet you come to know that there are 5 Ports
- Again a question. Where are the LEDs connected. You need the **Schematic** of the target board to know this.
- A **Schematic** is document which provides information about the physical connections on the hardware.
- From the schematic we come to know the the LEDs are connected to PORTB
- Port is a peripheral and we need need to know on how to access and address. This infos will be available in the data sheet in **PORTB** and **Data Memory** sections



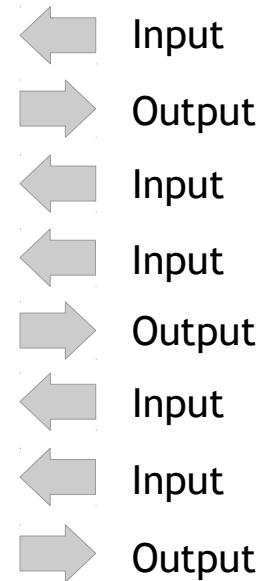
Microcontrollers

Embedded Programming - Let's Start Coding



- From the section of PORTB it clear that there are 2 more registers associated with it named, **TRISB** and **LATB**
- The TRISB register is very important for IO configuration. The value put in this register would decide pin direction as shown below

TRIS Register	PORT Register	Pin Direction
1	TRISx7	?
0	TRISx6	?
1	TRISx5	?
1	TRISx4	?
0	TRISx3	?
1	TRISx2	?
1	TRISx1	?
0	TRISx0	?



Microcontrollers

Embedded Programming - Let's Start Coding



- So from previous slide its clear that we have to use the TRIS register to control the pin direction
- LEDs are driven by external source, so the port direction should be made as output
- In this case the LEDs are connected to the controller and will be driven by it
- Fine, what should write to the port to make it work?
It depends on the hardware design.
- By considering all these point we can modify our code as shown in the next slide



Microcontrollers

Embedded Programming - Let's Start Coding



Example

```
void main(void)
{
    /*
     * Defining a pointer to PORTB data latch register at address 0xF8A,
     * pointing to 8 bit register. Refer data sheet
     */
    unsigned char *latb = (unsigned char *) 0x0F8A;
    /*
     * Defining a pointer to PORTB tri-state register at address 0xF93,
     * pointing to 8 bit register. Refer data sheet
     */
    unsigned char *trisb = (unsigned char *) 0x0F93;

    /* Setting the pin direction as output (0 - output and 1 - Input) */
    *trisb = 0x00;

    /*
     * Writing just a random value on the data latch register where
     * LEDs are connected
     */
    *latb = 0x55;
}
```



Microcontrollers

Embedded Programming - Let's Start Coding



- Hurray!!, we wrote our first Embedded C code for our target board
- Come on let's move forward, how do I compile this code?
- Obviously with a compiler!, Yes but a cross compiler since this code has to run on the target board.
- The target controller, as mentioned, is by Microchip. So we will be using **XC8 (Free Version)**
- You need to download it and install it in your system
 - In Windows you can use MPLABX
 - In Linux, the simplest would be command line



Microcontrollers

Embedded Programming - Let's Start Coding



- There are some assumptions made here like
 - You should know how to interface your board with the Host System
 - How to power up the target board
 - Procedure on how to transfer the code into the target boardand so on...
- All these might be available on the **User Manual** of the target board



Microcontrollers

EP - Let's Start Coding - Compiling



- Assuming you are using Linux it can be compiled as shown below

Compile like

```
user@user:~] xc8 --ROM=0-3000 --chip=18f4580 main.c
```

or

Compile like

```
user@user:~] /<path>/xc8 --ROM=0-3000 --chip=18f4580 main.c
```

- The output of the compilation would be many intermediate files, but we will be interested in file name **main.hex**



Microcontrollers

EP - Let's Start Coding - Hex Download



- As a last step we need to dump the **main.hex** to the target board.
- Sometimes we need a separate hardware to do this (typically depends on the target board design)
- RL-eCee board comes the microcontroller dumped with a serial boot loader in it
- So we need a system with a Serial Port or a USB to serial adapter from hardware prospective
- We will be using tiny boot loader software to transfer the code to the target
- Refer the next slide for the downloading steps



Microcontrollers

EP - Let's Start Coding - Hex Download



Download like

```
user@user:~] tinybldlin.py --port /dev/ttyUSB0 --file main.hex
```

or

Download like

```
user@user:~] <path>/tinybldlin.py --port /dev/<device_file> --file main.hex
```

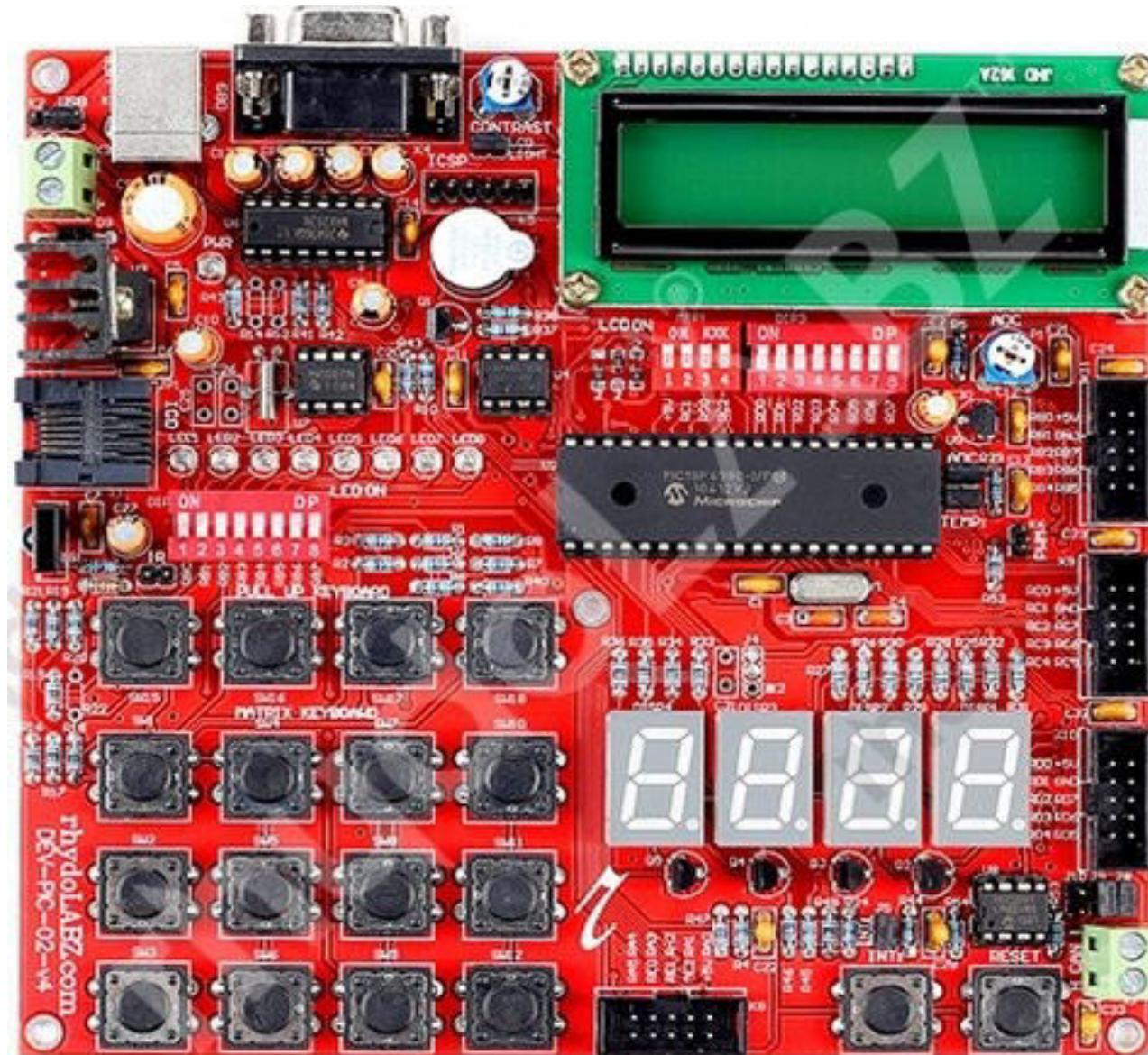
- The LEDs on the board should glow alternately
- Ooops!, where are the LEDs on board?

Know your target board too!!.. Refer next slide



Microcontrollers

EP - Let's Start Coding - RL-eCee Dev Board



Microcontrollers

Embedded Programming - Let's Start Coding



- The thrill of having your first code working is different.
- But, this is just the beginning, you might like to design some good application based on your board
- Proceeding forward, the way how we wrote the code with indirect addressing would require good amount of time
- So it is common to use the definitions and libraries provided by the cross compiler to build our applications **else we end up “Reinventing the Wheel”**
- The same code can be re-written the the way provided in the next slide



Microcontrollers

Embedded Programming - Let's Start Coding



Example

```
#include <xc8.h>

void main(void)
{
    /* Setting the pin direction as output (0 - output and 1 - Input) */
    TRISB = 0x00;

    /*
     * Writing just a random value on the data latch register where
     * LEDs are connected
     */
    LATB = 0x55;
}
```

- So simple. Isn't it?



Project Creation



Microcontrollers

Project Creation - Code Organization

- Please organize the code as shown below to increase productivity and modularity
- Every .c file should have .h file

```
#include "supporting_file.h"  
void f() {  
    void function_1(void)  
    {  
        /* Function Code */  
    }  
}
```

```
#include "main.h"  
  
void init_config(void)  
{  
    /* Initialization Code */  
}  
  
void main(void)  
{  
    init_config();  
  
    while (1)  
    {  
        /* Application Code */  
    }  
}
```

```
#ifndef MAIN_H  
#define MAIN_H  
  
#include <htc.h>  
  
#endif
```

```
#ifndef SUPPORTING_FILE_H  
#define SUPPORTING_FILE_H  
  
#endif
```

modules.c

main.c

main.h

modules.h



Microcontrollers

Project Creation - Code Template

main.c

```
#include "main.h"

void init_config(void)
{
    /* Initialization Code */
}

void main(void)
{
    init_config();

    while (1)
    {
        /* Application Code */
    }
}
```

main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <htc.h>

#endif
```

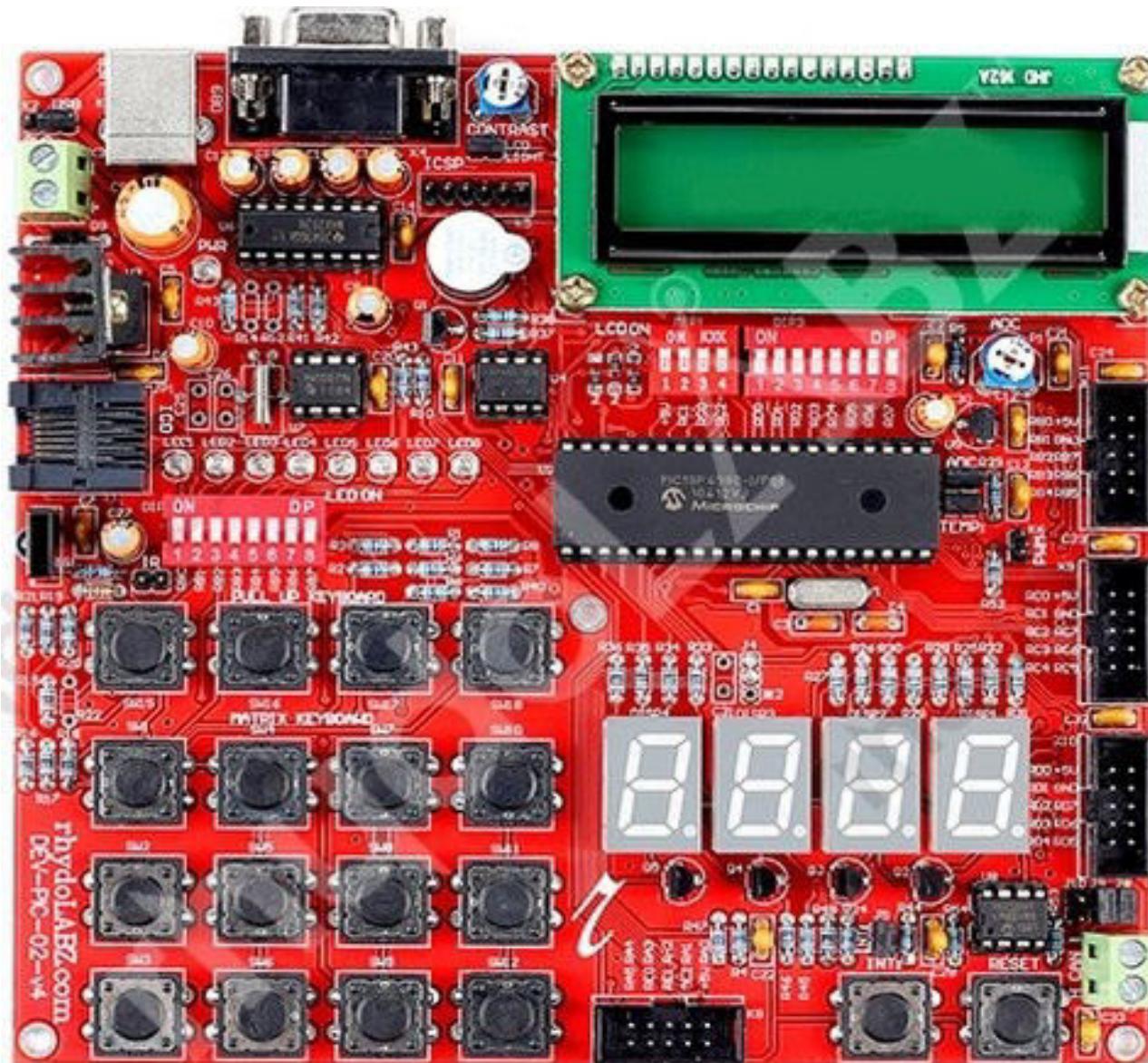


Lets Roll Out on Interfaces



Microcontrollers

Lets Role Out on Interfaces



- LEDs
- Digital Keypad
- Interrupts
- Timers
- Clock I/O
- SSDs
- CLCD
- Matrix Keypad
- Analog Inputs



Light Emitting Diodes



Microcontrollers

Interfaces - LEDs - Introduction

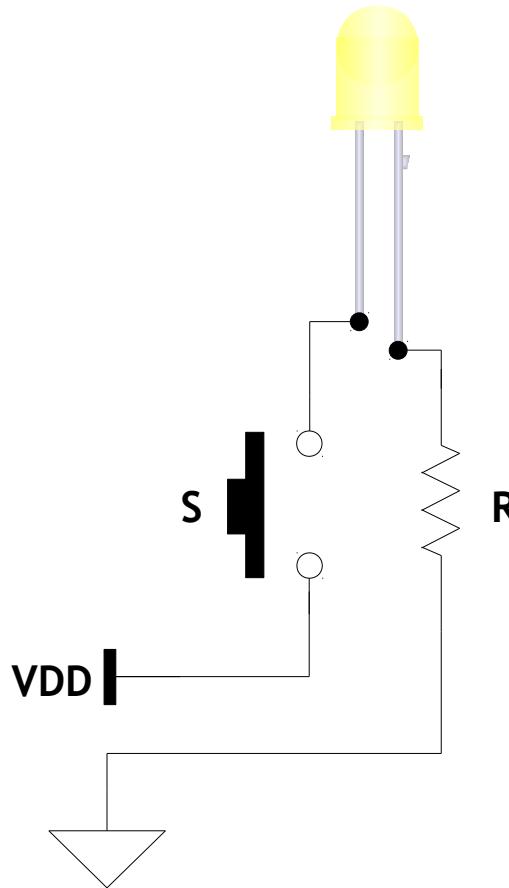


- Simplest device used in most on the embedded applications as feedback
- Works just like diodes
- Low energy consumptions, longer life, smaller size, faster switching make it usable in wide application fields like
 - Home lighting,
 - Remote Controls, Surveillance,
 - Displays and many more!!



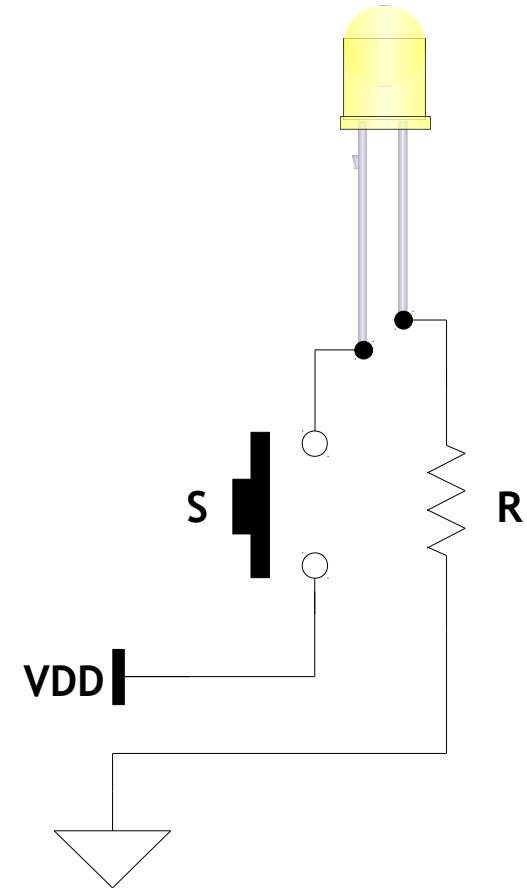
Microcontrollers

Interfaces - LEDs - Working Principle



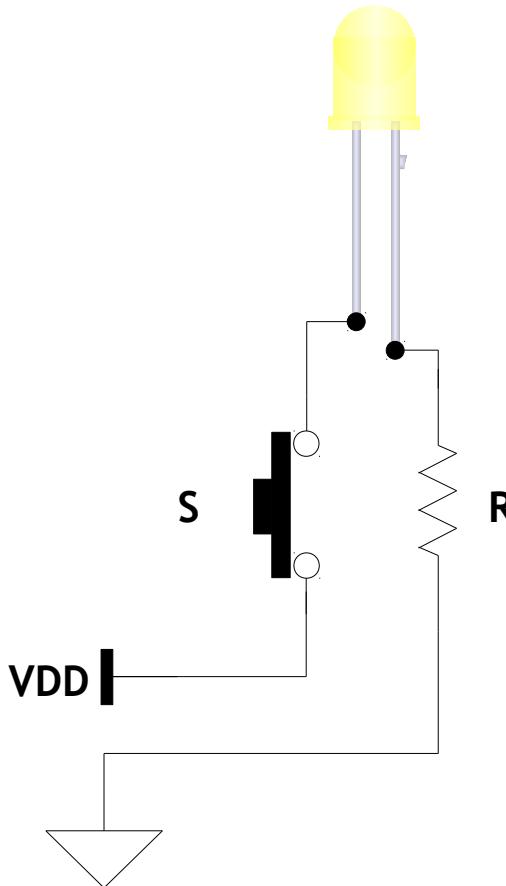
?

Which side will
work?

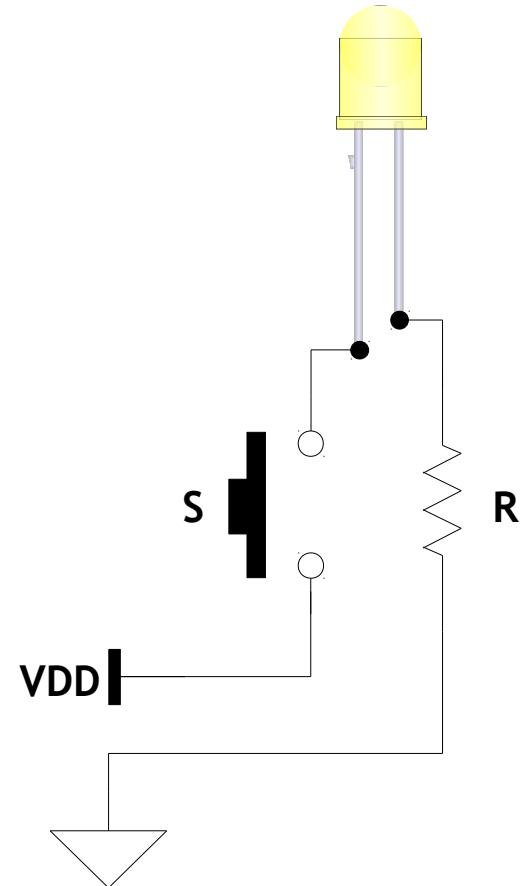


Microcontrollers

Interfaces - LEDs - Working Principle

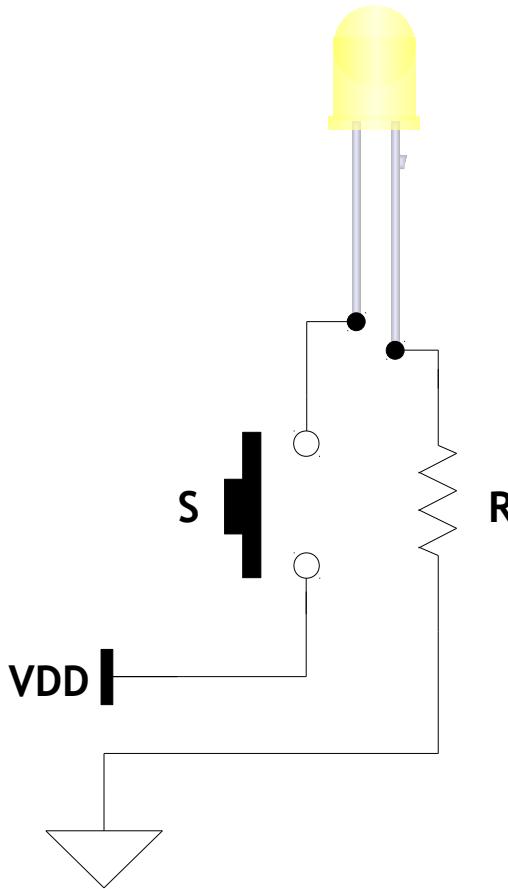


×
Oops, wrong
choice. Can you
explain why?

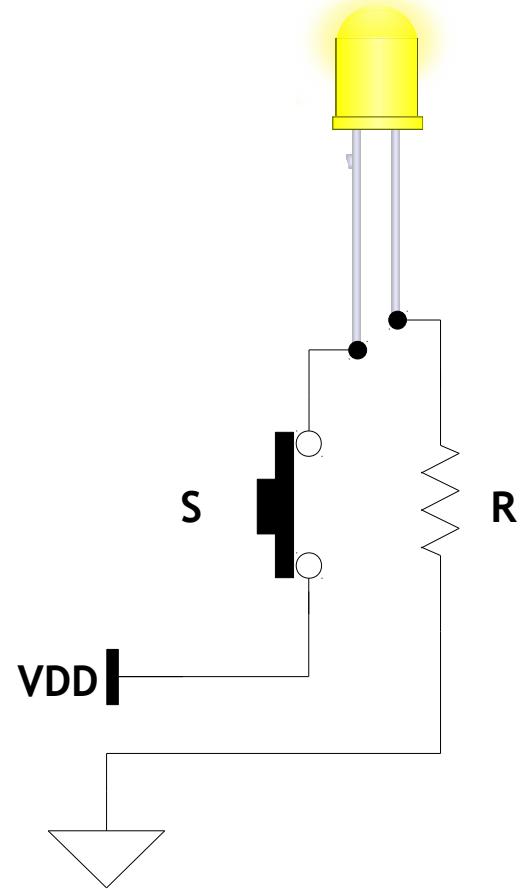


Microcontrollers

Interfaces - LEDs - Working Principle

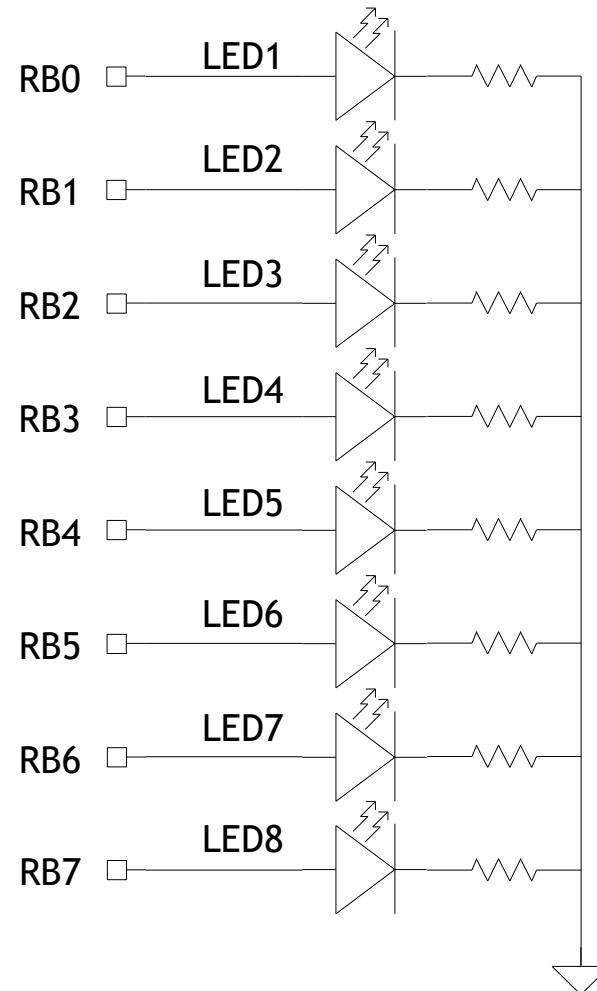


Ooh, looks like
you know the
funda.



Microcontrollers

Interfaces - LEDs - Circuit on Board



Note: Make sure the DP switch its towards LEDs



Digital Keypad



Microcontrollers

Interfaces - Digital Keypad



- Introduction
- Interfacing
- Input Detection
- Bouncing Effect
- Circuit on Board



Microcontrollers

Interfaces - Digital Keypad - Introduction



- Provides simple and cheap interface
- Comes in different shapes and sizes
- Preferable if the no of user inputs are less
- Mostly based on tactile switches
- Some common application of tactile keys are
 - HMI
 - Mobile Phones
 - Computer Mouse etc.,

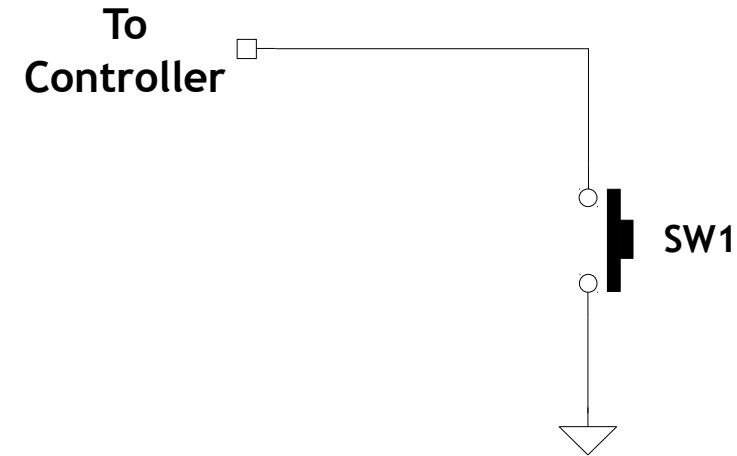


Microcontrollers

Interfaces - Digital Keypad - Tactile Switches



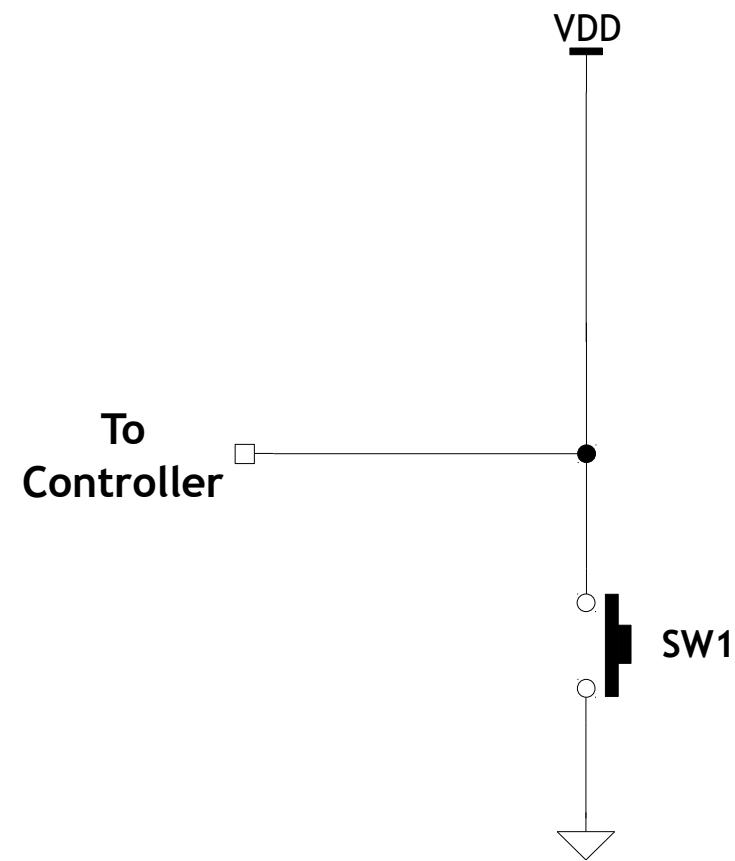
- Considering the below design what will be input to the controller if the switch is pressed?



Microcontrollers

Interfaces - Digital Keypad - Tactile Switches

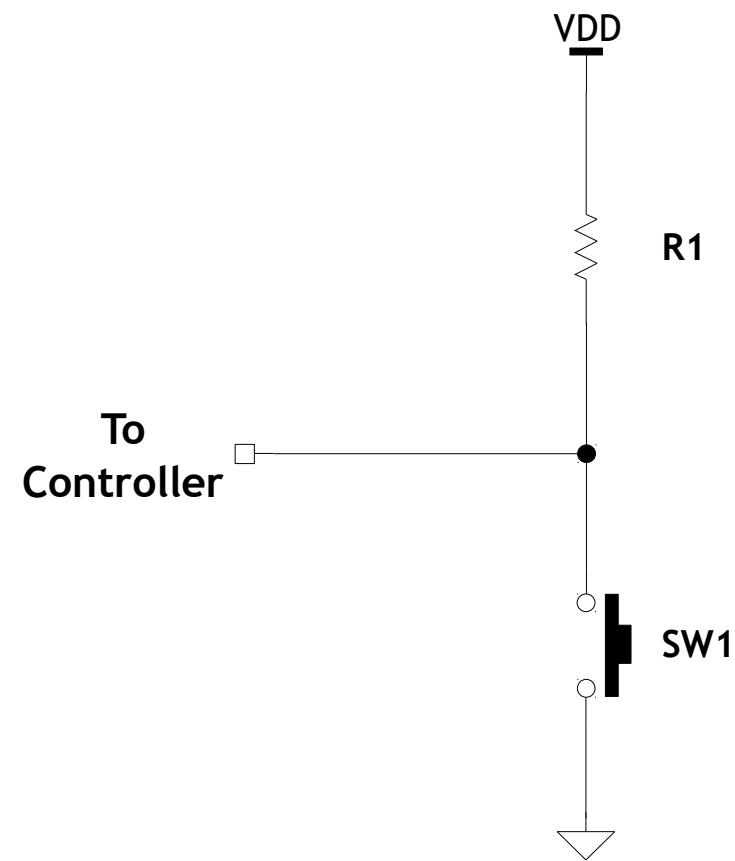
- Will this solve the problem which may arise in the design mentioned in previous slide?



Microcontrollers

Interfaces - Digital Keypad - Tactile Switches

- Now will this solve the problem which may arise in the design mentioned in previous slides?

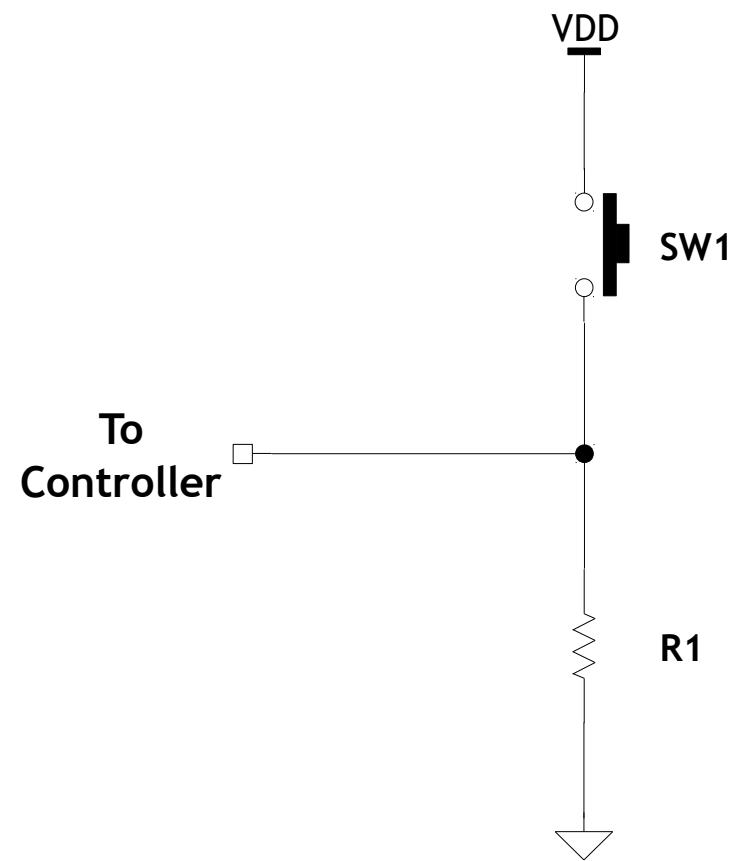


Microcontrollers

Interfaces - Digital Keypad - Tactile Switches



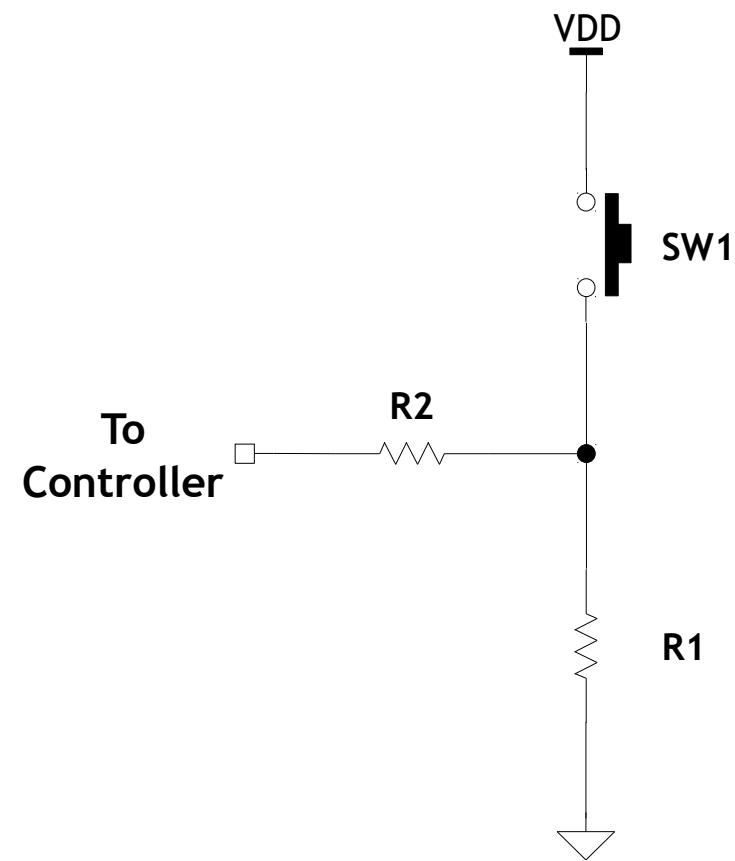
- What would you call this design?
- Is there any potential problem?



Microcontrollers

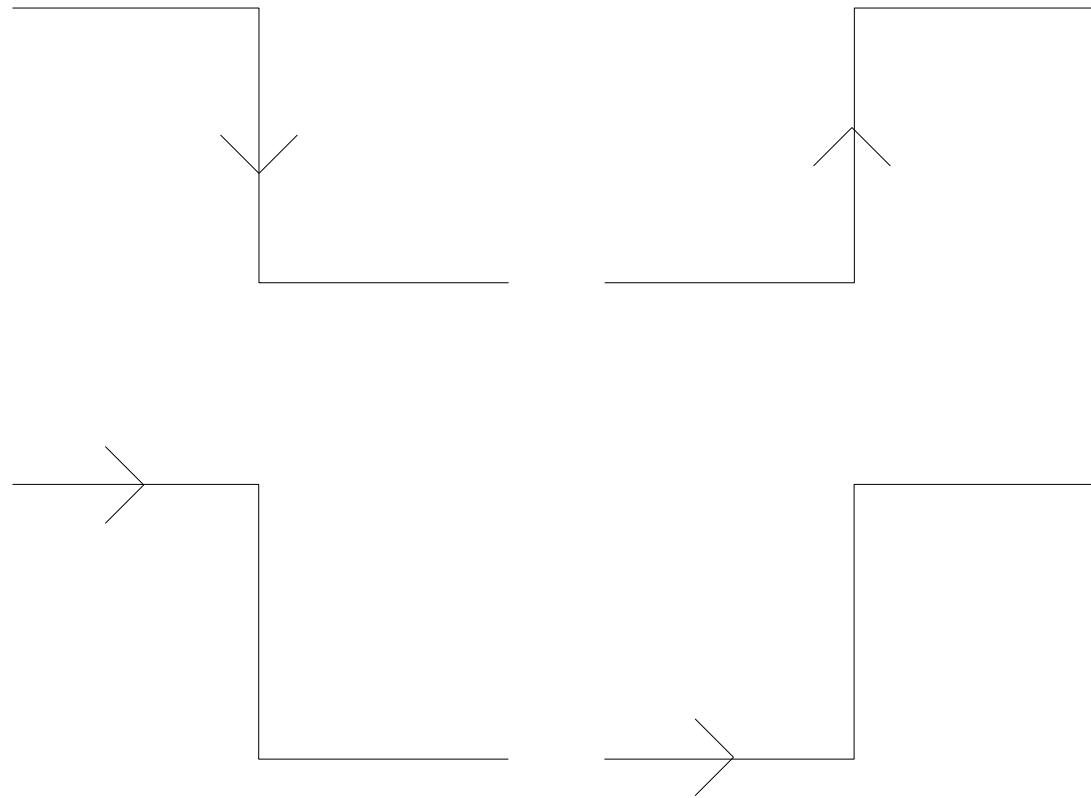
Interfaces - Digital Keypad - Tactile Switches

- What would you call this design?
- Is there any potential problem?



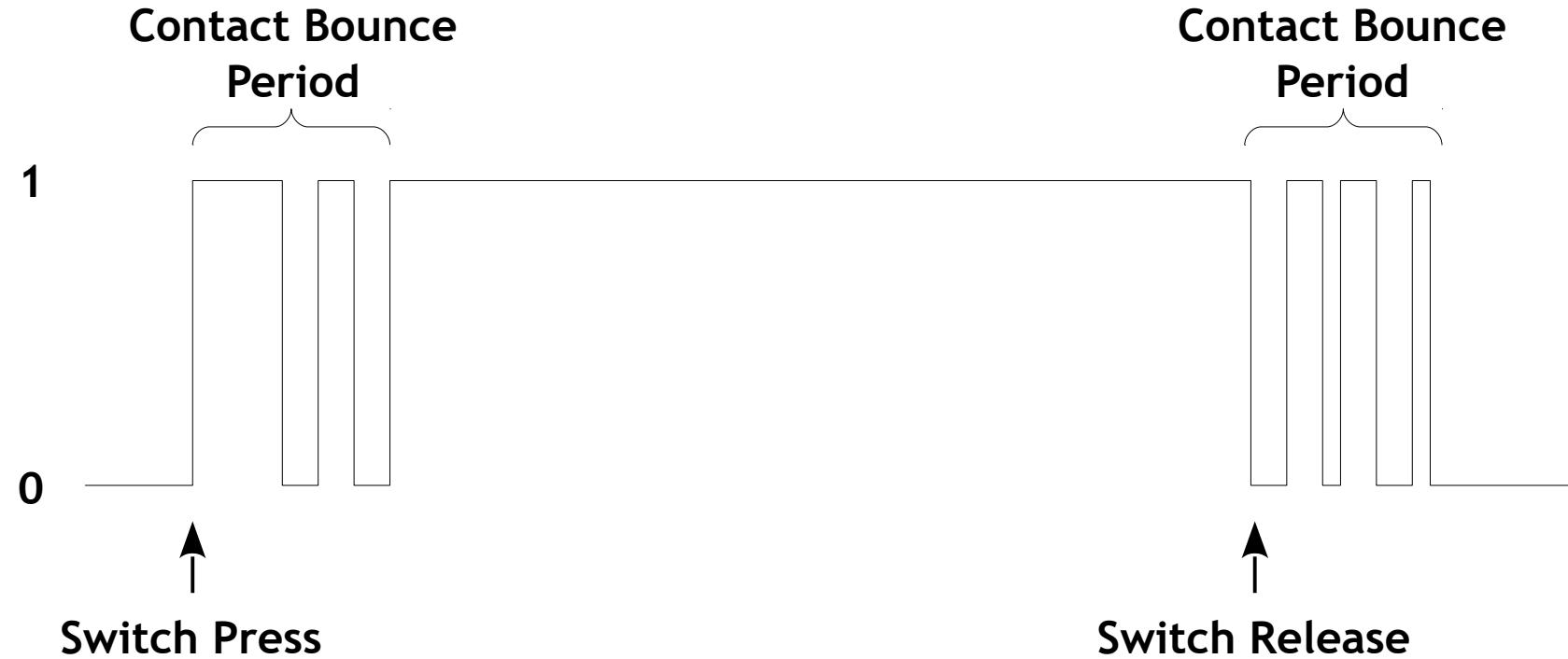
Microcontrollers

Interfaces - Digital Keypad - Triggering Methods



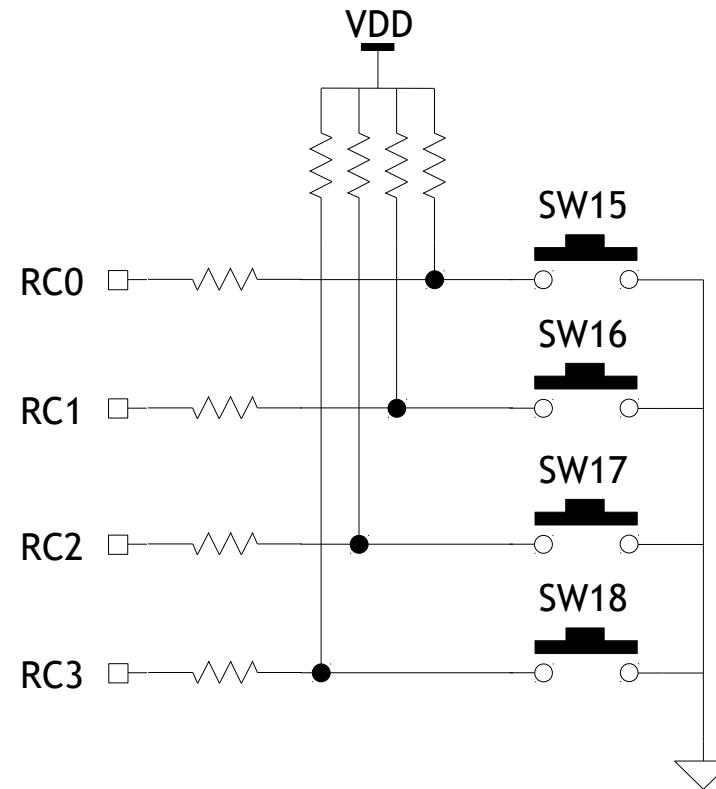
Microcontrollers

Interfaces - Digital Keypad - Bouncing Effects



Microcontrollers

Interfaces - Digital Keypad - Circuit on Board



Interrupts



Microcontrollers

Interrupts



- Basic Concepts
- Interrupt Source
- Interrupt Classification
- Interrupt Handling



Microcontrollers

Interrupts - Basic Concepts



- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
 - An internal or external device requests the MPU to stop the processing
 - The MPU acknowledges the request
 - Attends to the request
 - Goes back to processing where it was interrupted
- Polling



Microcontrollers

Interrupts - Interrupt vs Polling



- Loss of Events
- Response
- Power Management



Microcontrollers

Interrupts - Sources

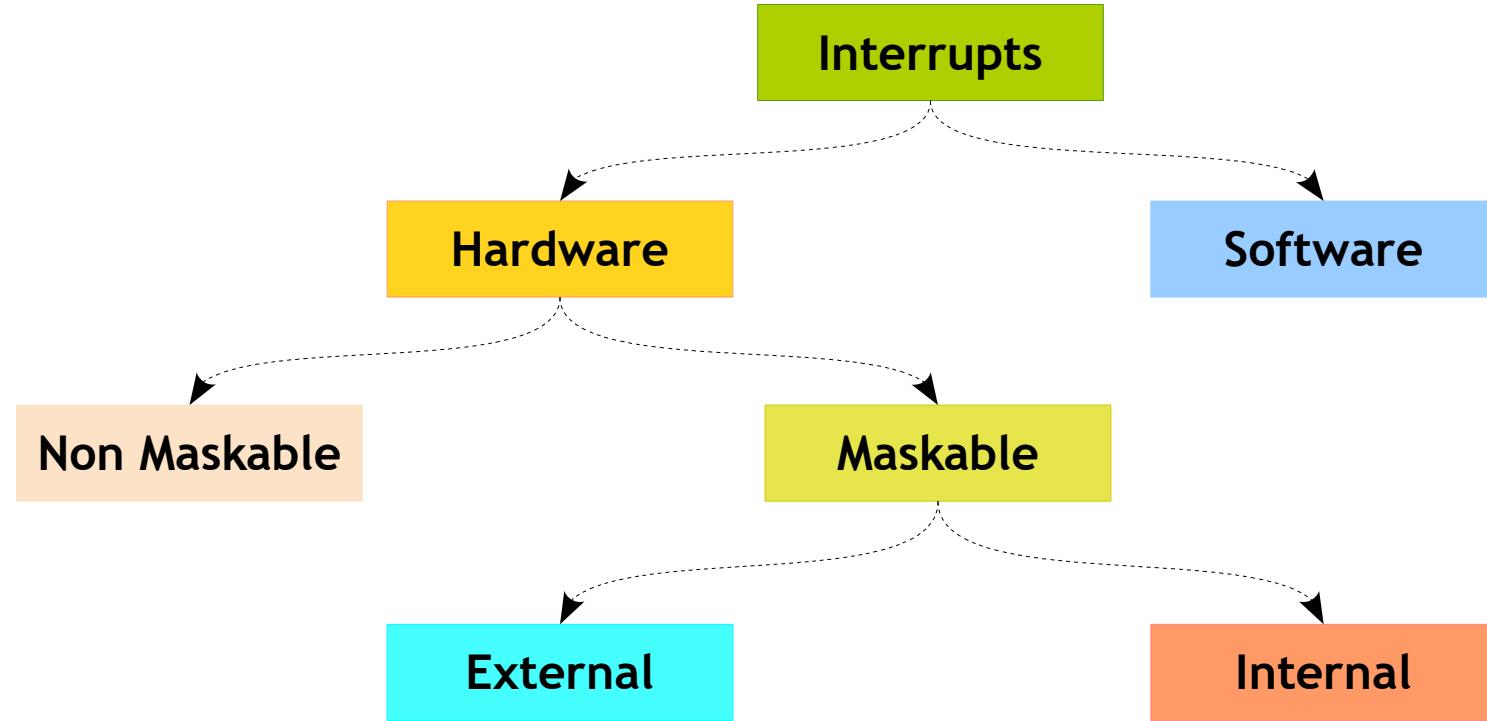


- External
- Timers
- Peripherals



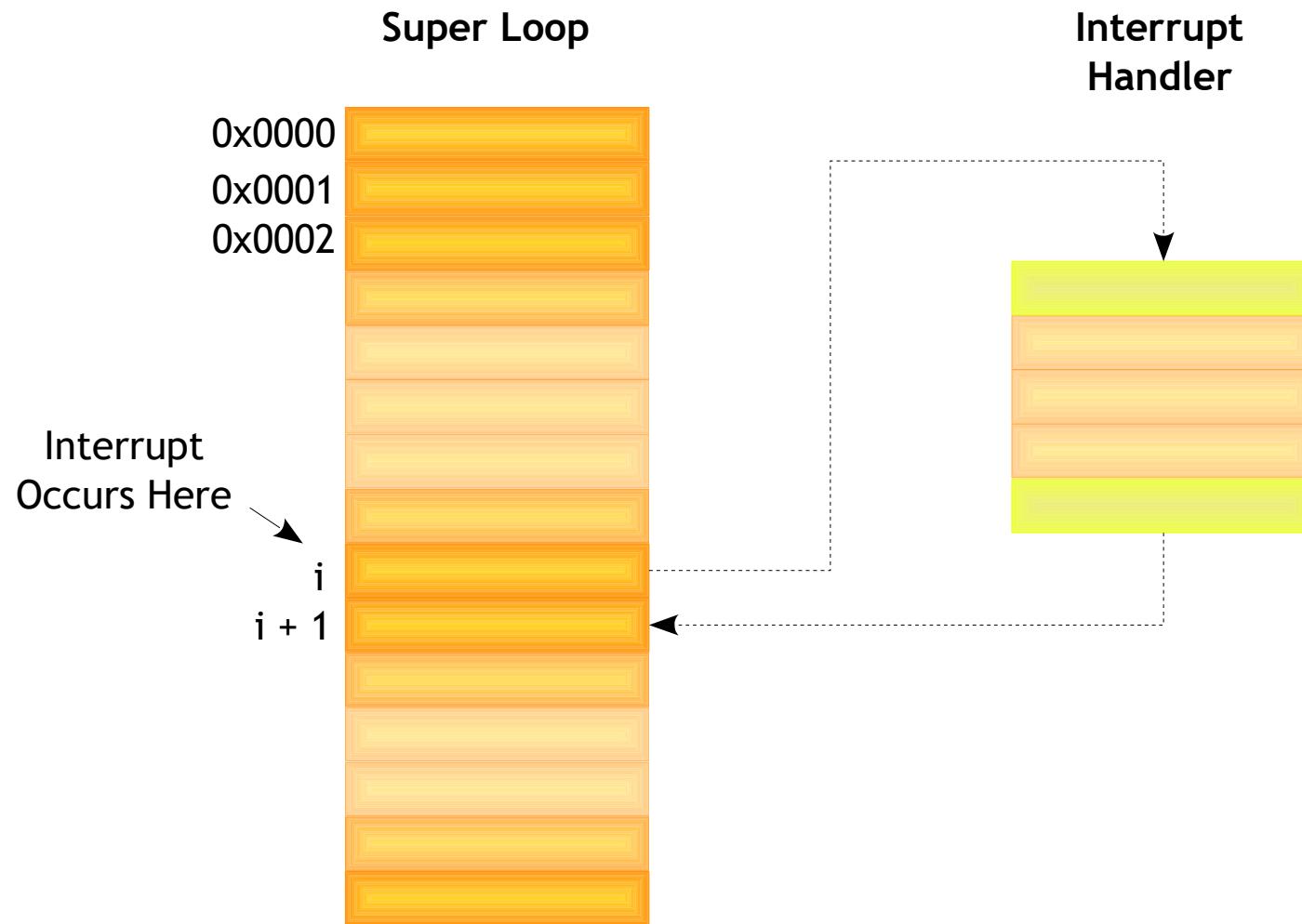
Microcontrollers

Interrupts - Classification



Microcontrollers

Interrupts - Handling



Microcontrollers

Interrupts - Service Routine (ISR)



- Similar to a subroutine
- Attends to the request of an interrupting source
 - Clears the interrupt flag
 - Should save register contents that may be affected by the code in the ISR
 - Must be terminated with the instruction RETFIE
- When an interrupt occurs, the MPU:
 - Completes the instruction being executed
 - Disables global interrupt enable
 - Places the address from the program counter on the stack
- Return from interrupt



Microcontrollers

Interrupts - Service Routine (ISR)



- What / What Not



Microcontrollers

Interrupts - Latency

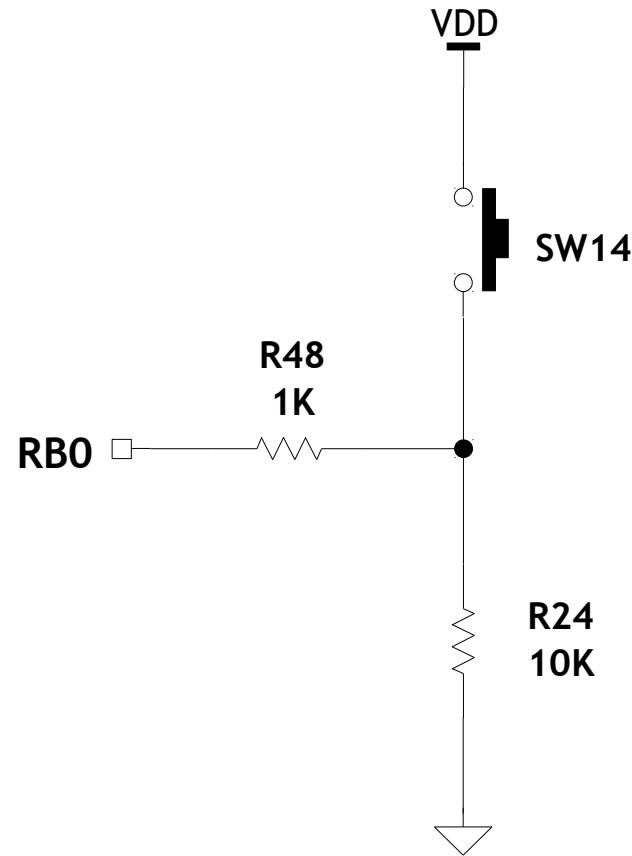


- Latency is determined by:
 - Instruction time (how long is the longest)
 - How much of the context must be saved
 - How much of the context must be restored
 - The effort to implement priority scheme
 - Time spent executing protected code



Microcontrollers

Interrupts - External Interrupt - Circuit on Board



Microcontrollers

Interrupts - External Interrupt - Example

Example

```
#include <xc8>

static void init_config(void)
{
    init_external_interrupt();
}

void main(void)
{
    unsigned char i;
    init_config();

    while (1)
    {
        while (!glow_led)
        {
            if (SWITCH == 1)
                glow_led = 1;

            for (i = 10000; i--; );
        }
        if (glow_led)
            LED = 0;
    }
}
```

```
bit glow_led;

void interrupt external_pin(void)
{
    if (INTFLAG)
    {
        glow_led = 1;
        INTFLAG = 0;
    }
}
```



Timers



Microcontrollers

Timers - Introduction



- Resolution → Register Width
- Tick → Up Count or Down Count
- Quantum → System Clock settings
- Scaling → Pre or Post
- Modes
 - Counter
 - PWM or Pulse Generator
 - PW or PP Measurement etc.,
- Examples



Microcontrollers

Timers - Example



- Requirement – 5 pulses of 8 µsecs
- Resolution – 8 Bit
- Quantum – 1 µsecs
- General



Microcontrollers

Timers - Example

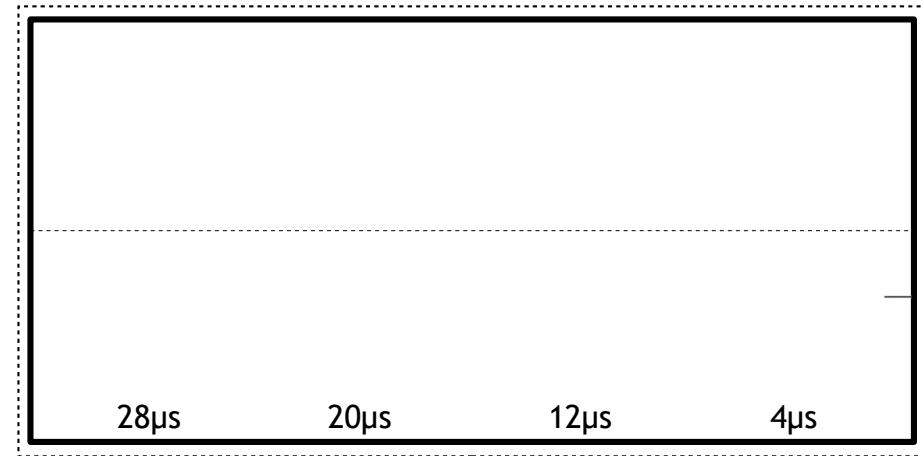


Timer Register

252

Overflows

0



Clock I/O



Microcontrollers

Clock I/O - Introduction



- Most peripheral devices depends on Clocking
- Controllers have to generate clock to communicate with the peripherals
- Some of the Controllers internal peripherals work on external clocking provided at it pins

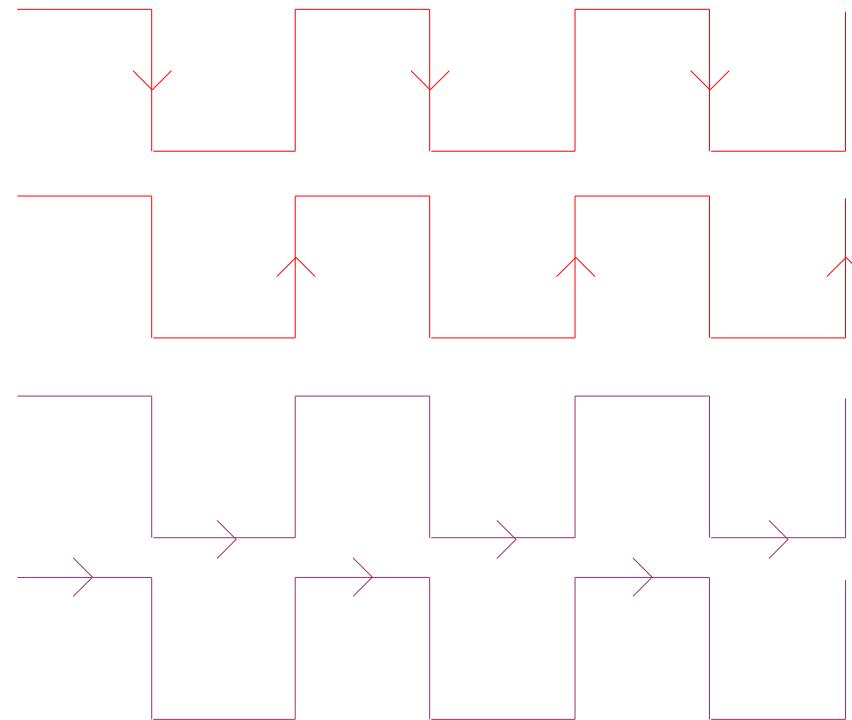


Microcontrollers

Clock I/O - Introduction



- The activity on the devices could be on
 - Edges
 - Levels



Microcontrollers

Clock I/O - PWM

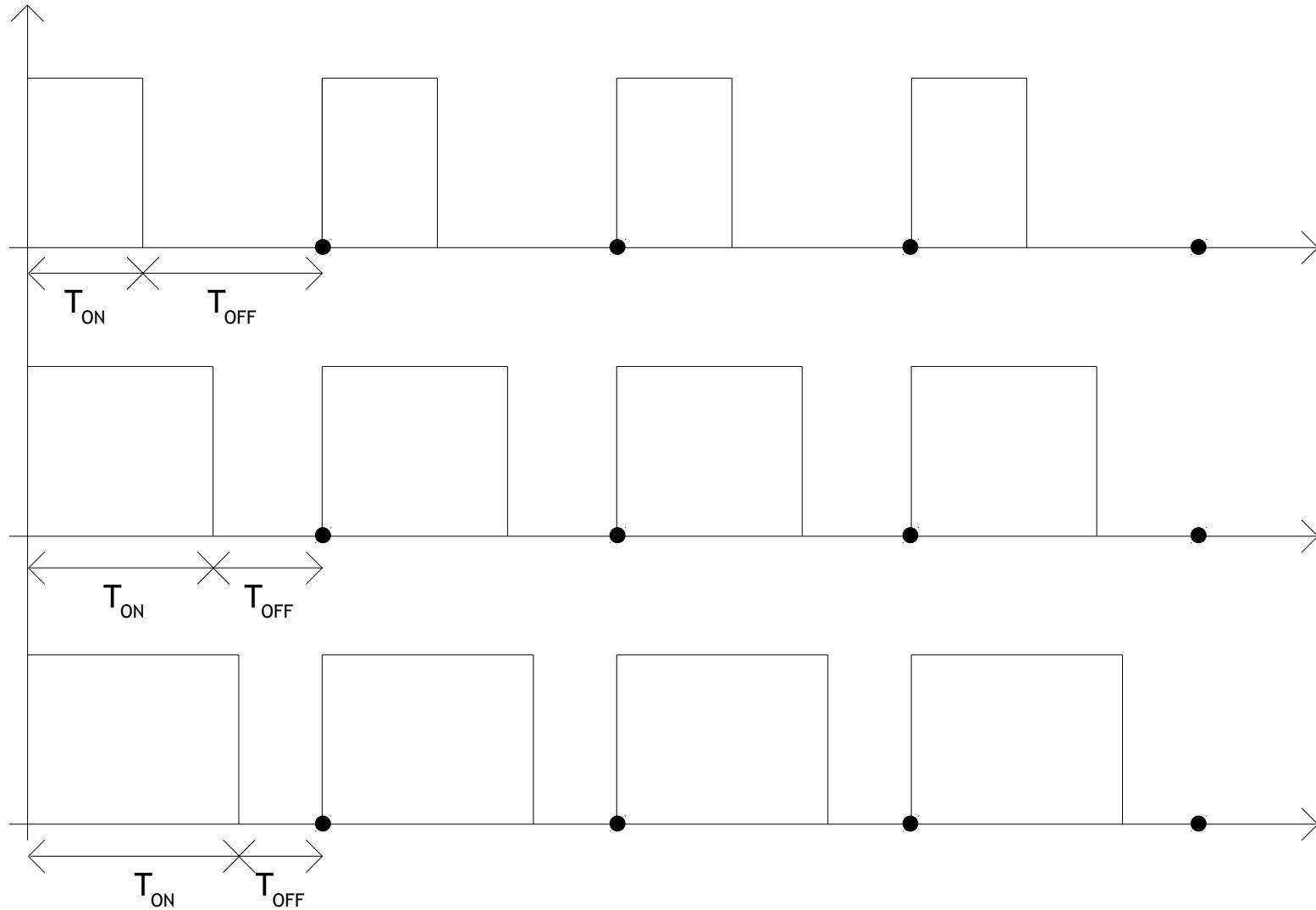


- Sometimes called as PDM (Pulse Duration Modulation)
- A technique used to vary the active time vs inactive time in a fixed period.
- Mostly used to control the average voltage of the Load connected.
- Used in wide applications like Motor Controls, Lamp Dimmers, Audio, Power Controls and many more..



Microcontrollers

Clock I/O - PWM



Seven Segment Displays



Microcontrollers

Interfaces - SSDs - Introduction

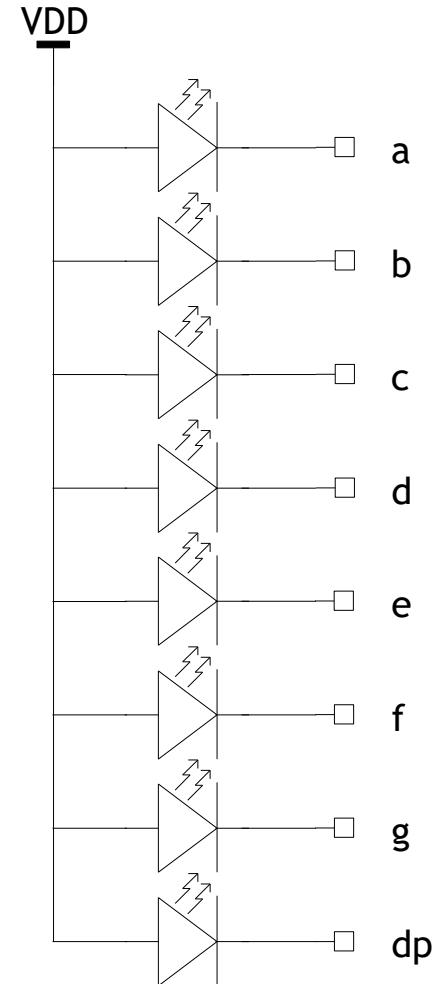
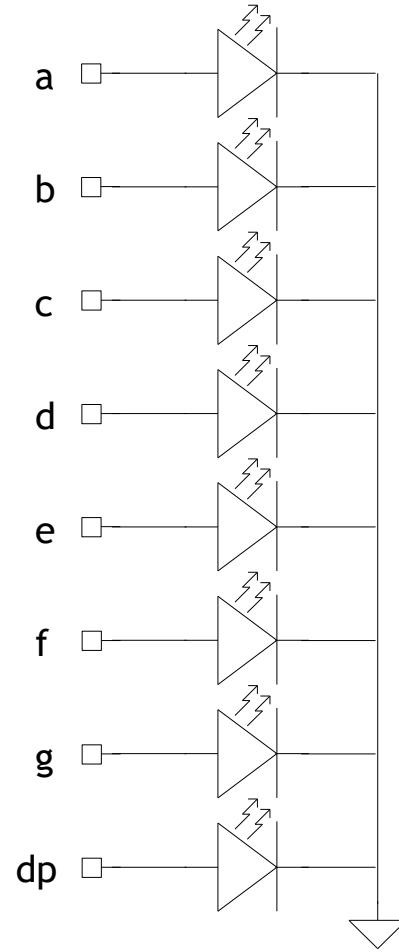


- Array of LEDs connected internally
- Possible configurations are common cathode and common anode
- Very good line of sight
- Used in many applications



Microcontrollers

Interfaces - SSDs - Introduction - Design

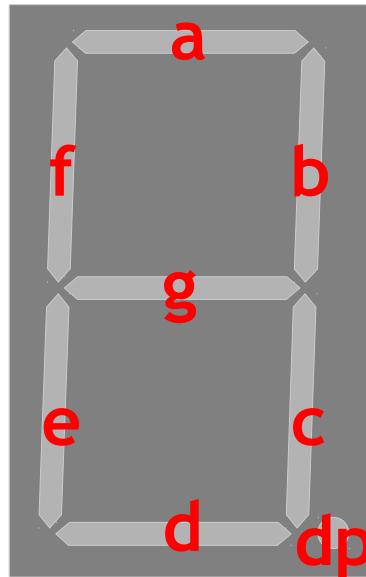


Microcontrollers

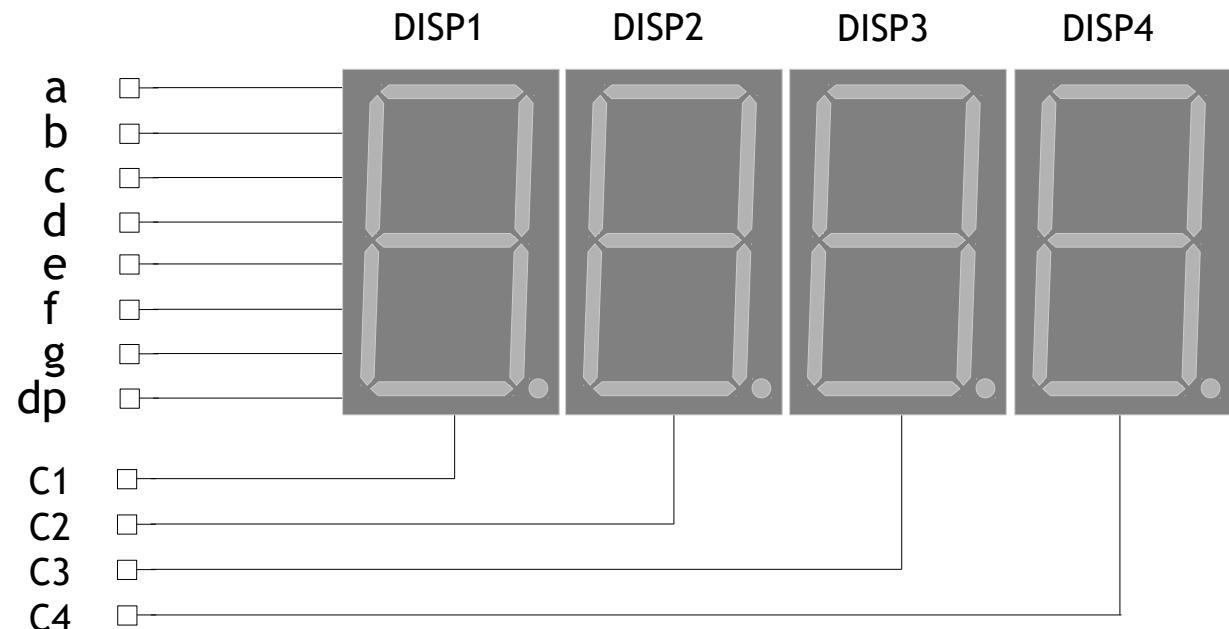
Interfaces - SSDs - Example

- Assuming a common anode display, what will the output for provided table?

Data								Control			
a	b	c	d	e	f	g	dp	C1	C2	C3	C4
0	0	1	0	0	1	0	1	0	1	0	0
0	1	0	0	1	0	0	1	0	0	1	0



Segment Map

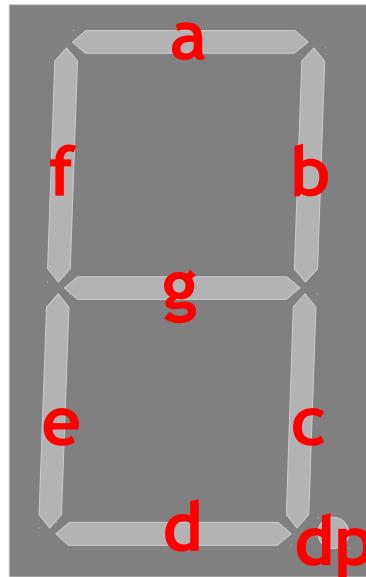


Microcontrollers

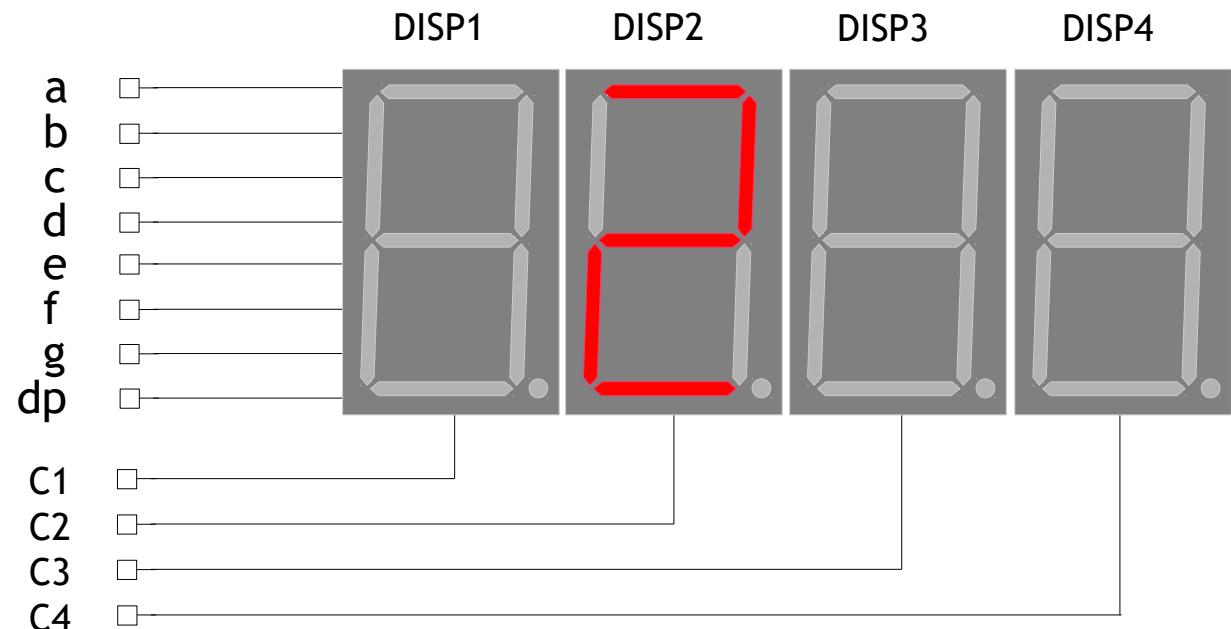
Interfaces - SSDs - Example

- Assuming a common anode display, what will the output for provided table?

Data								Control			
a	b	c	d	e	f	g	dp	C1	C2	C3	C4
0	0	1	0	0	1	0	1	0	1	0	0



Segment Map

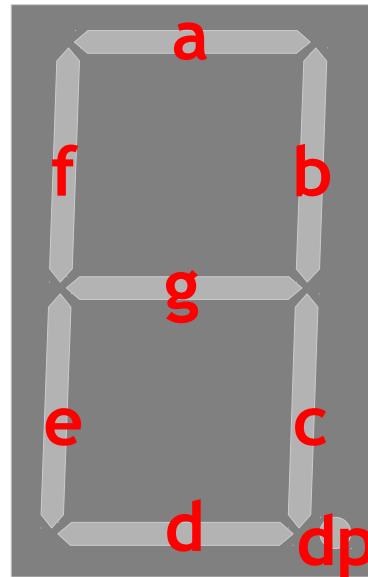


Microcontrollers

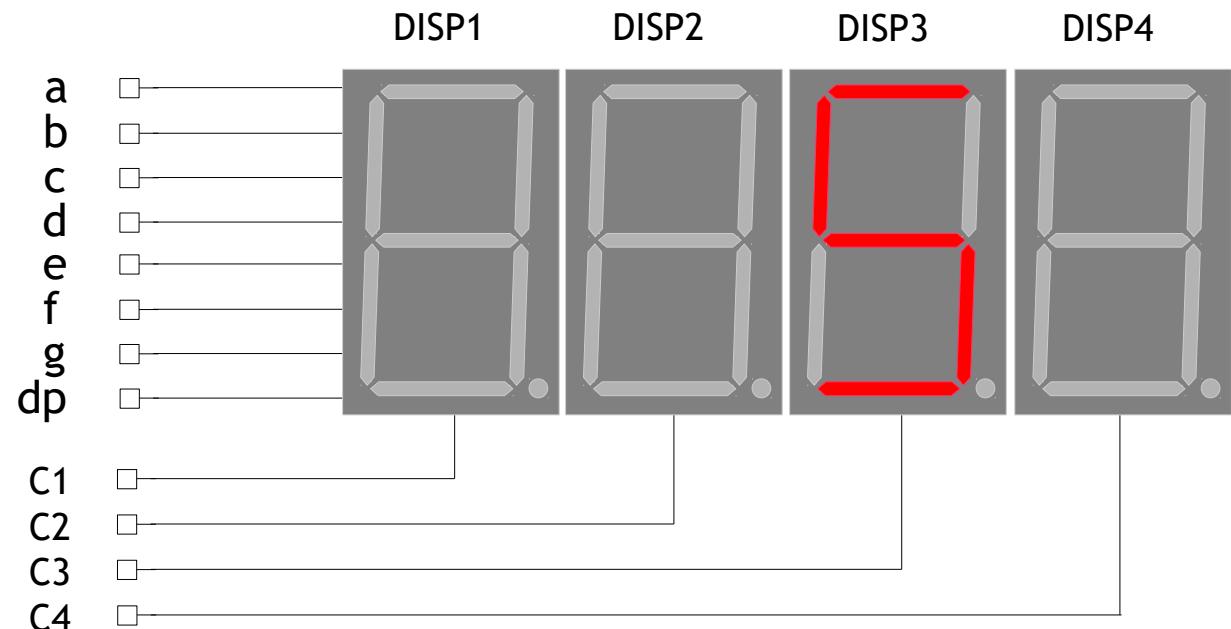
Interfaces - SSDs - Example

- Assuming a common anode display, what will the output for provided table?

Data								Control			
a	b	c	d	e	f	g	dp	C1	C2	C3	C4
0	1	0	0	1	0	0	1	0	0	1	0

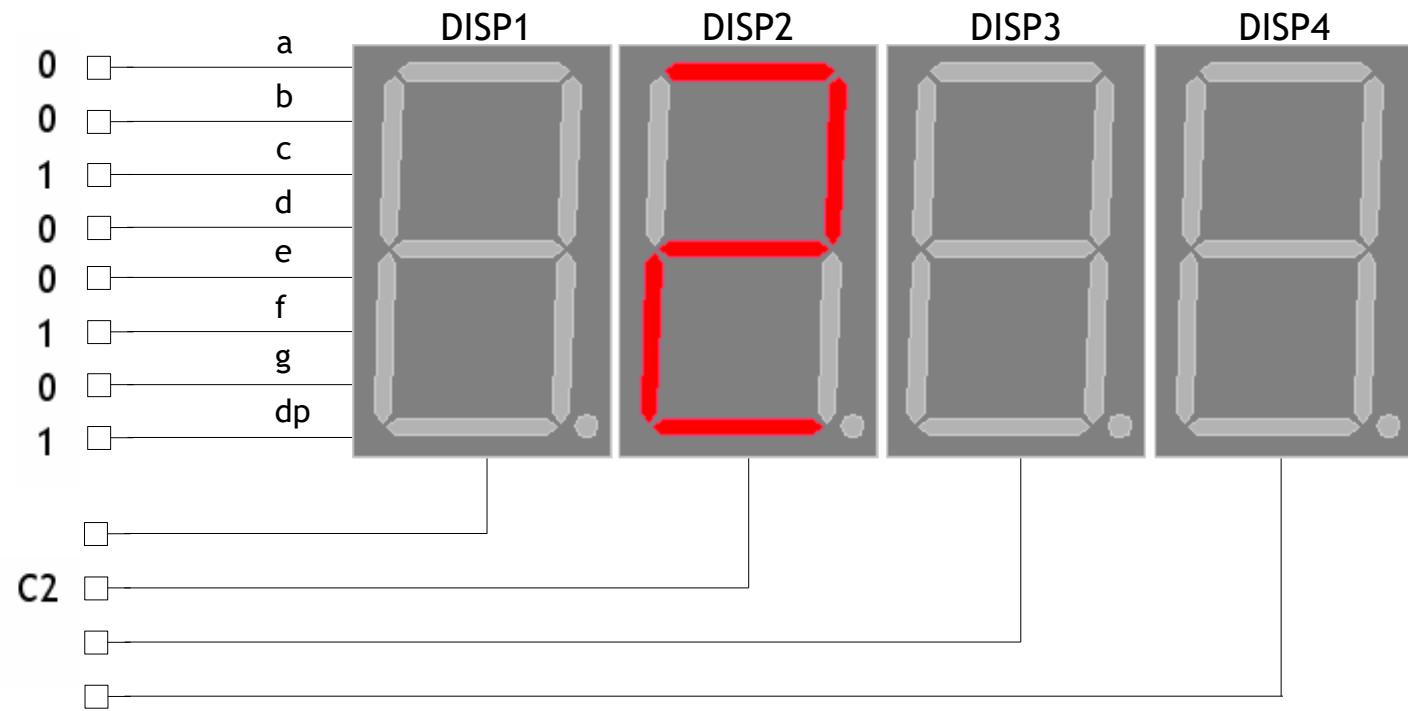


Segment Map



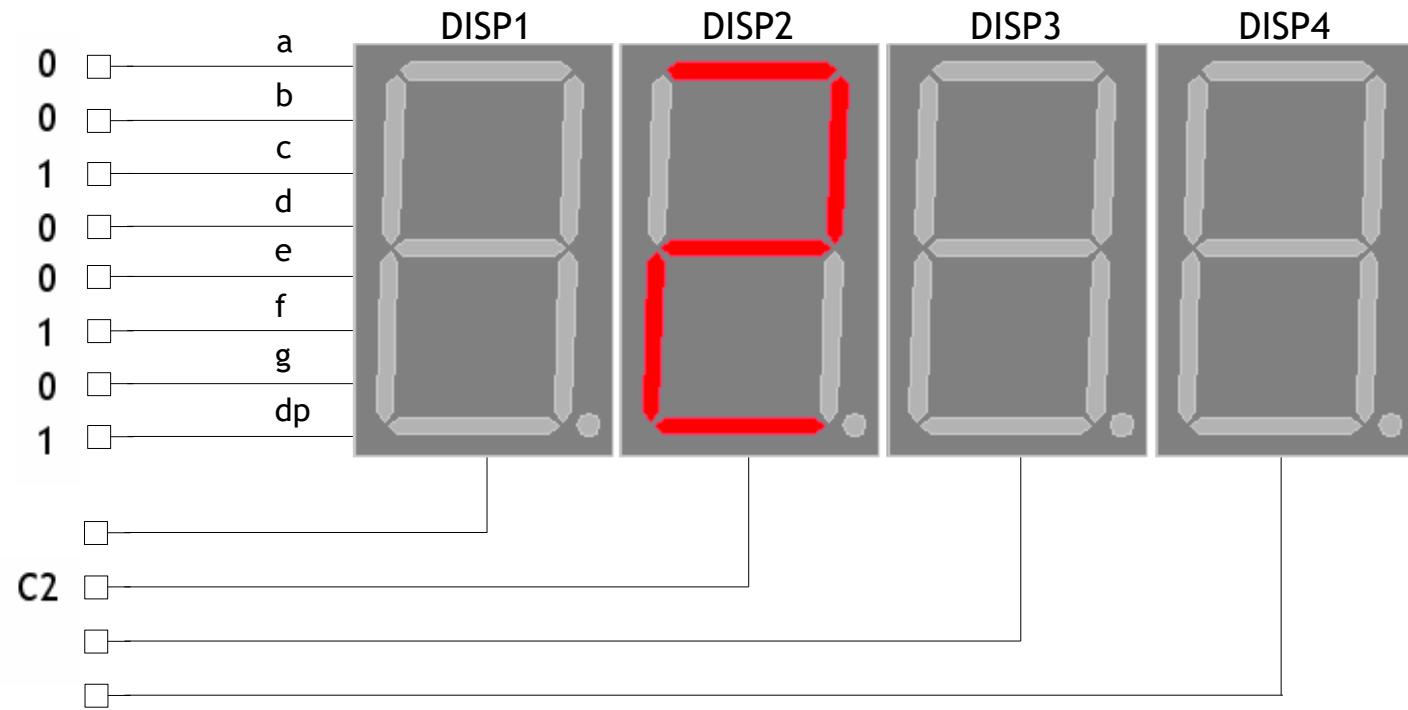
Microcontrollers

Interfaces - SSDs - Multiplexing



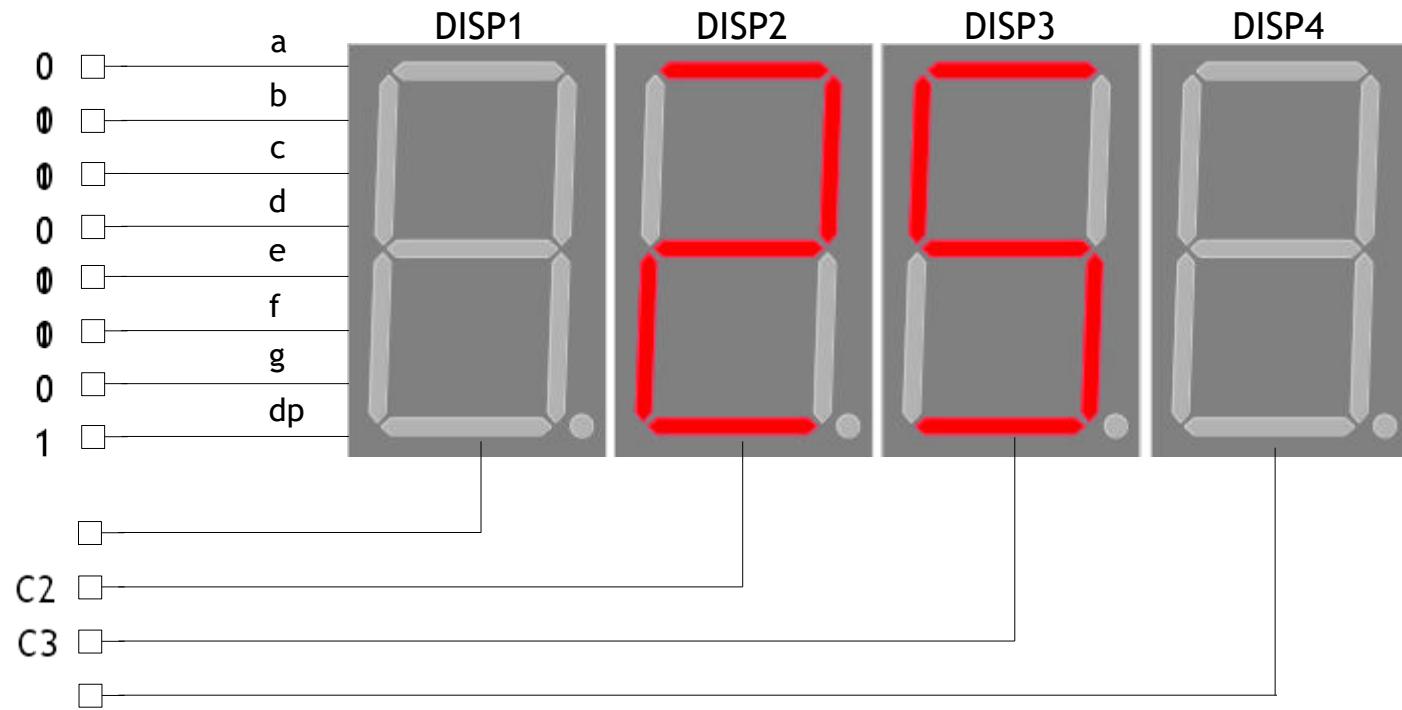
Microcontrollers

Interfaces - SSDs - Multiplexing



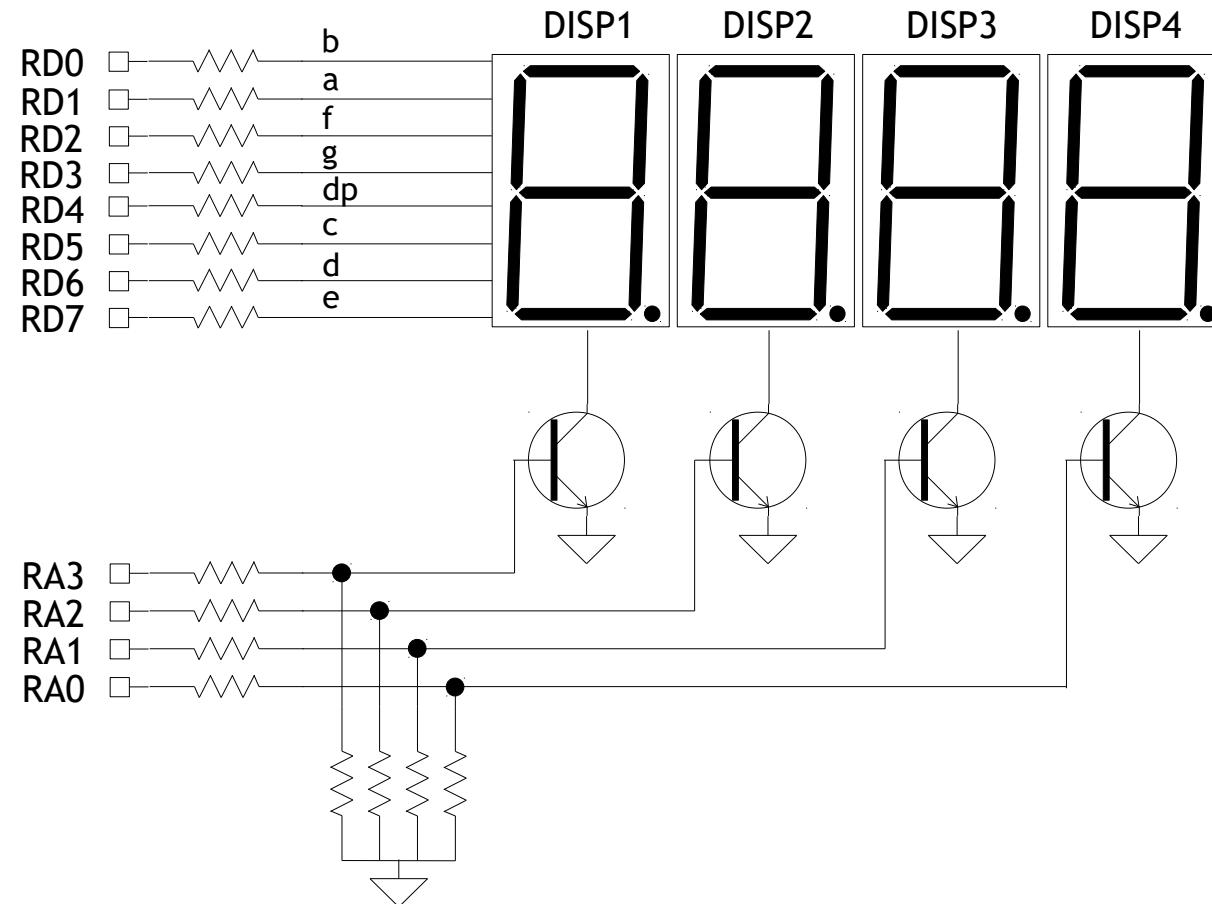
Microcontrollers

Interfaces - SSDs - Multiplexing



Microcontrollers

Interfaces - SSDs - Circuit on Board

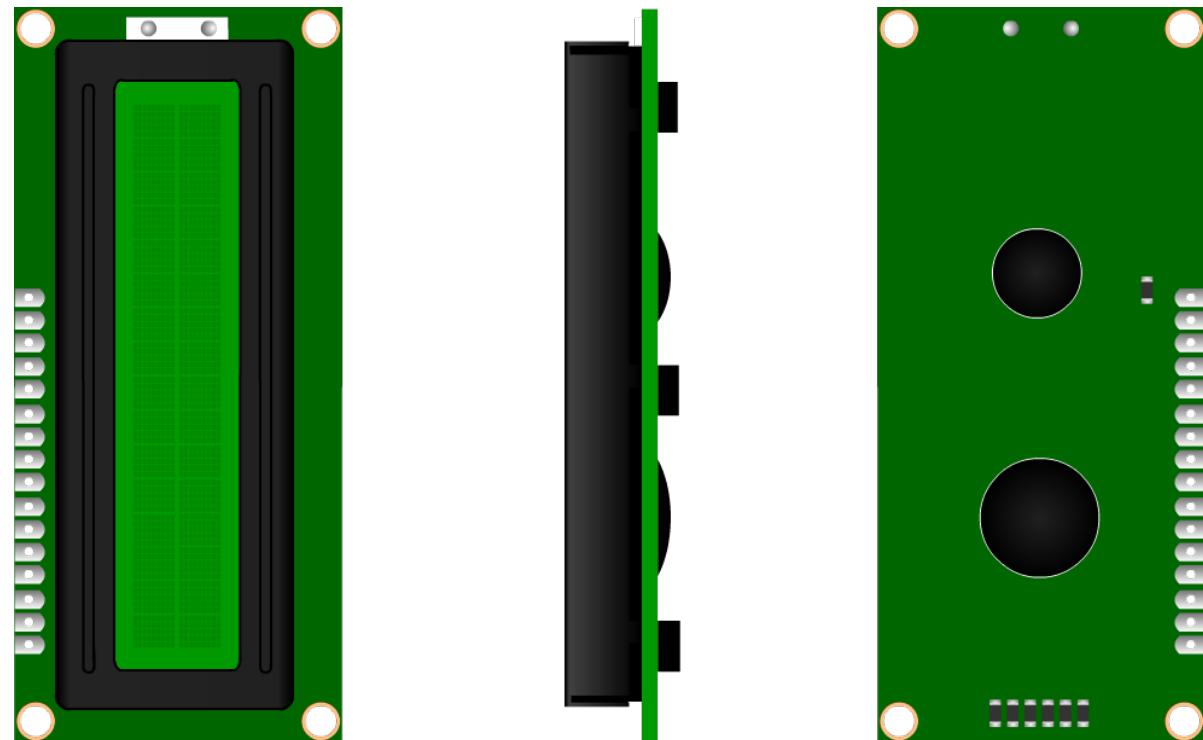


Character Liquid Crystal Display

Microcontrollers

Interfaces - CLCD - Introduction

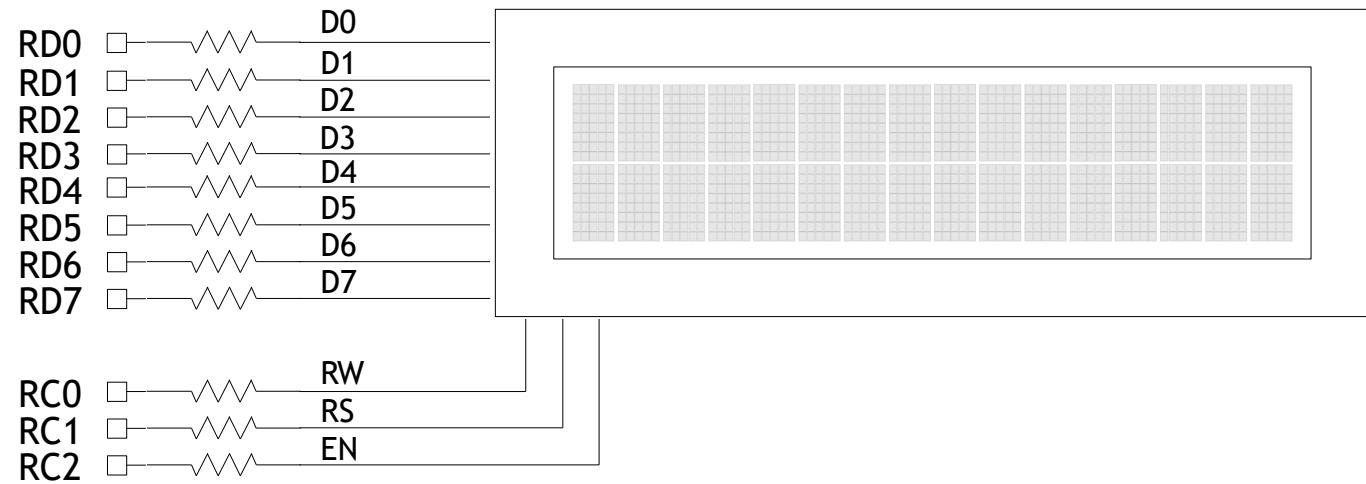
- Most commonly used display ASCII characters
- Some customization in symbols possible
- Communication Modes
 - 8 Bit Mode
 - 4 Bit Mode



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Vdd	Vdd	Vo	RS	R/W	E	D0	D1	D2	D3	D4	D5	D6	D7	A	K

Microcontrollers

Interfaces - CLCD - Circuit on Board



Matrix Keypad



Microcontrollers

Interfaces - Matrix Keypad - Introduction

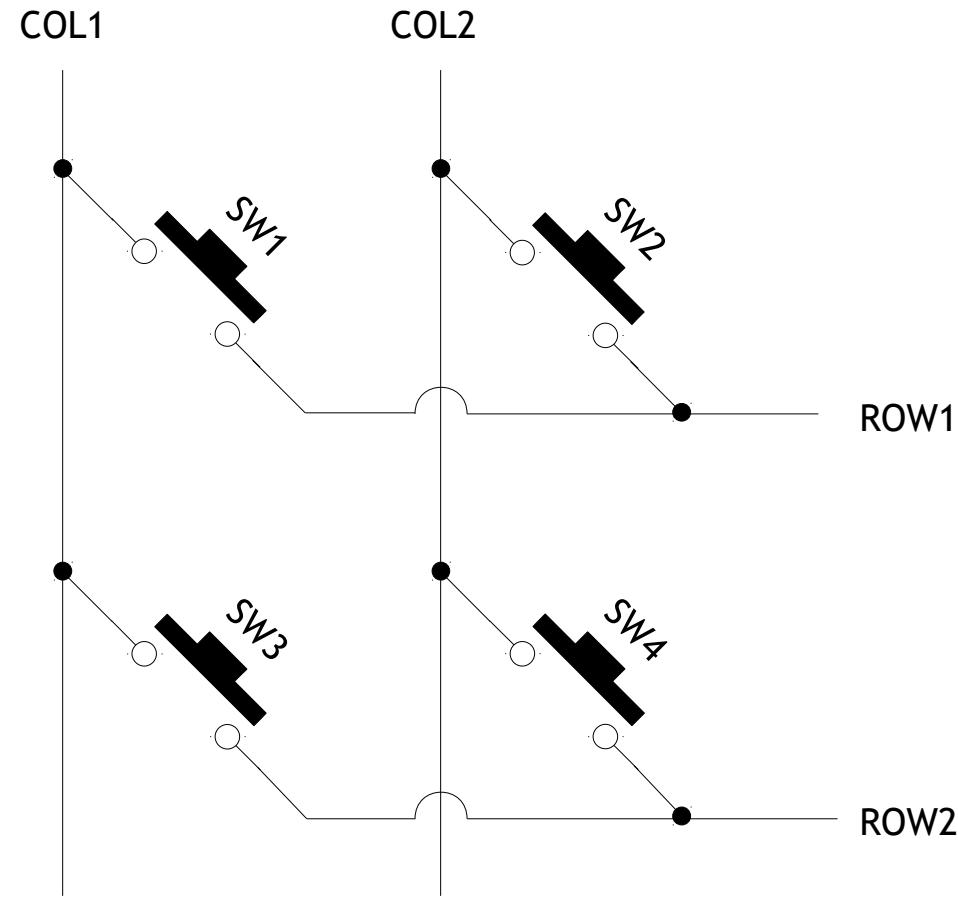


- Used when the more number of user inputs is required and still want to save the some controller I/O lines
- Uses rows and columns concept
- Most commonly used in Telephonic, Calculators, Digital lockers and many more applications



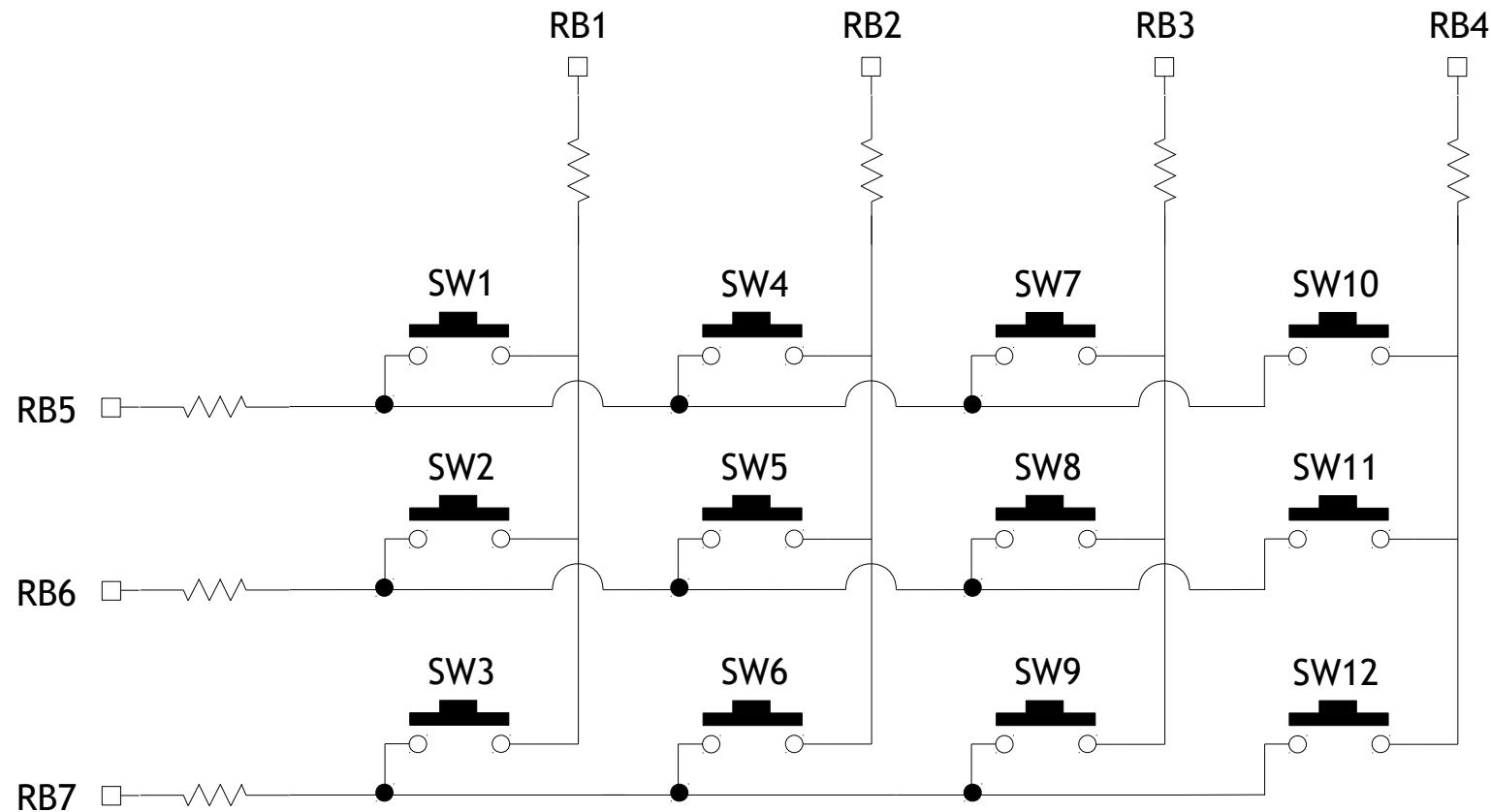
Microcontrollers

Interfaces - Matrix Keypad - Example



Microcontrollers

Interfaces - Matrix Keypad - Circuit on Board



Analog Inputs



Microcontrollers

Analog Inputs - Introduction

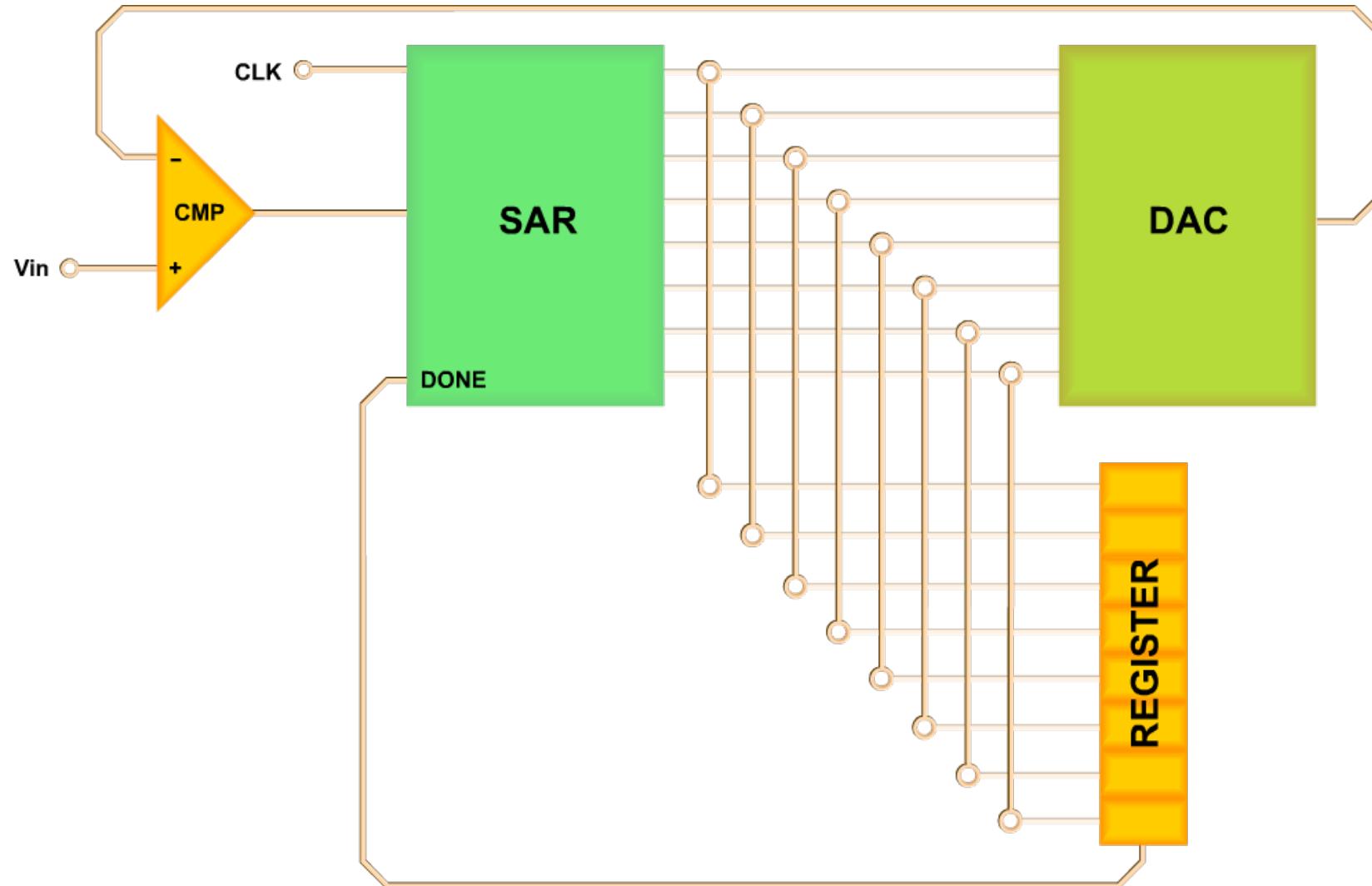


- Very important peripheral in embedded systems for real time activities
- Multiplexed with normal GPIO
- Comes with different resolutions



Microcontrollers

Analog Inputs - SAR



Data Storage



Microcontrollers

Data Storage - Introduction



- Mostly used in every Embedded System
- The size and component (memory) of storage depends upon the requirements
- Modern controllers has built in data storage
 - EEPROM
 - Data Flash etc.



Thank You

Communication Protocols I

Team Emertxe



Communication Protocols I

- Introduction
- UART
- SPI
- I²C
- CAN



Introduction



Introduction



- What do mean by Communication?
- Mode of Communications
- Type of Communications
- Why Protocols?



UART



UART



- Introduction
- Interface
- Hardware Configurations
- Frame Format



UART

Introduction

- Asynchronous
- Duplex - Any
- Master / Slave



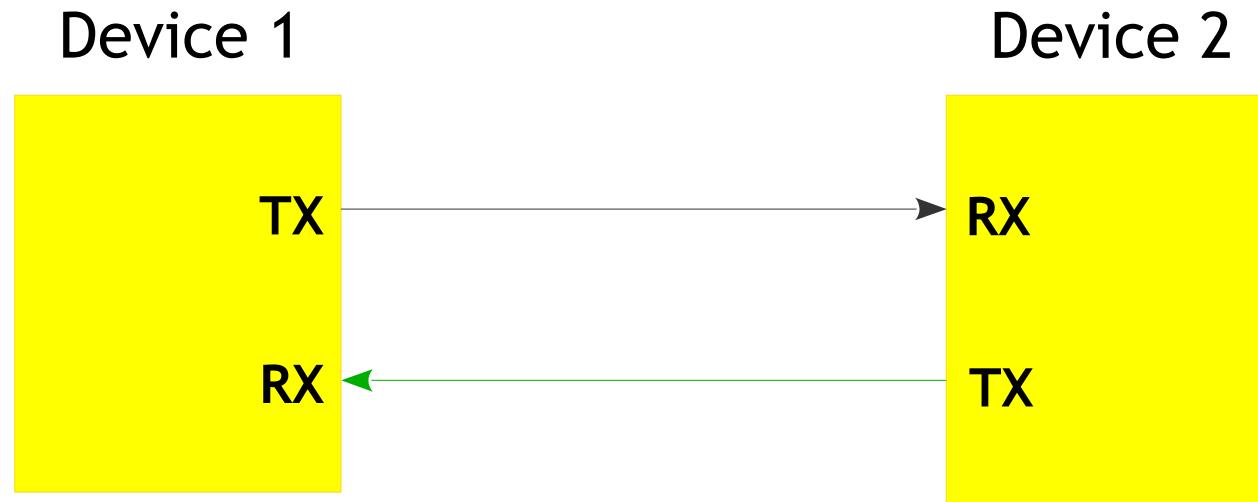
UART Interface

- RX
- TX



UART

Hardware Configuration



UART

Frame Format



- Data part can be 5 to 9 bits
- Stop could be 2 bits
- Parity could be 0 or 1 bit

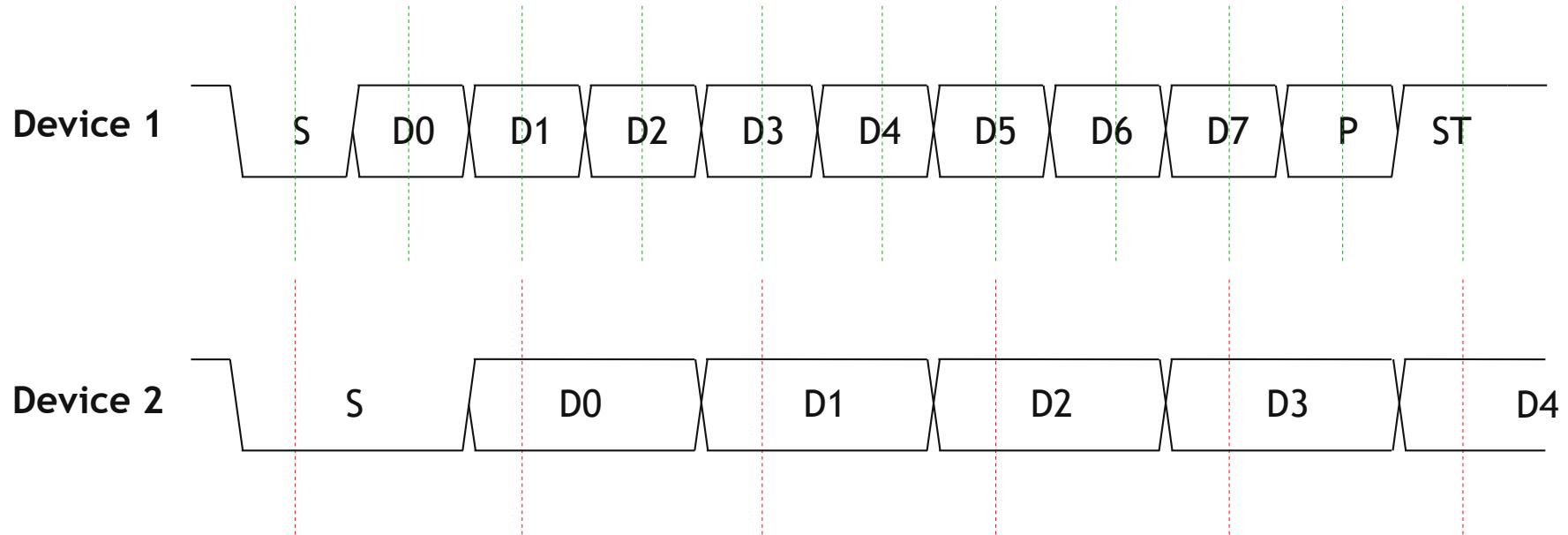


- Number of symbols per second (In this context the a symbol is a bit)
- So, sometimes referred as Bit Rate (No of bits per second)
- The frequency of the data transfer
- Both transmitter and receiver has to agree upon a common frequency for data integrity



UART

Baud Rate

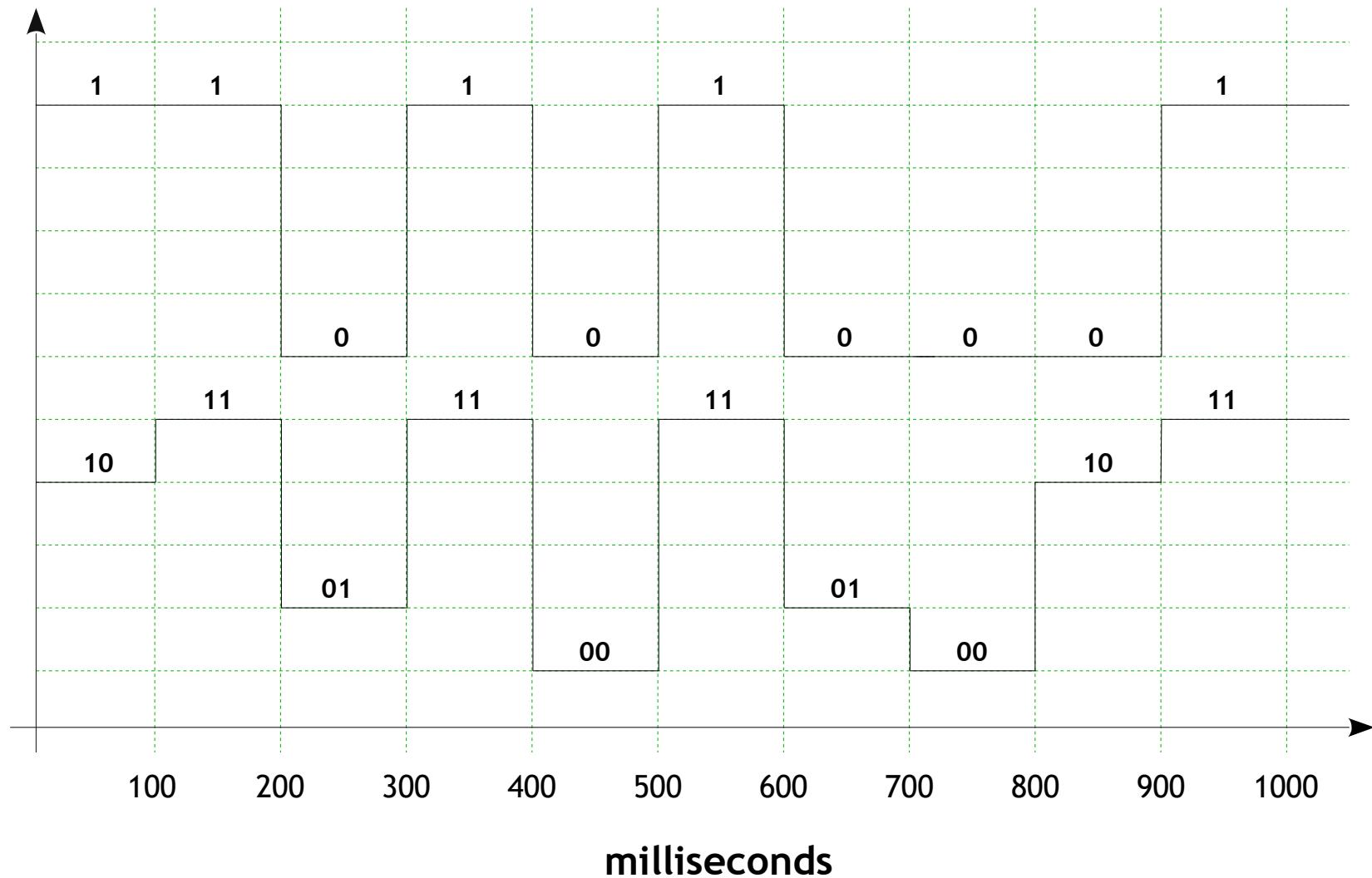


— Transmitter Sample Frequency
— Receiver Sample Frequency



UART

Baud Rate vs Bit Rate



Serial Peripheral Interface



Serial Peripheral Interface



- Introduction
- Interface
- Hardware Configurations
- Data Transmission
 - Data Validity



SPI

Introduction

- Synchronous
- Full Duplex
- Master / Slave



SPI Interface

- SCLK
- MOSI
- MISO
- nSS



SPI

Hardware Configuration

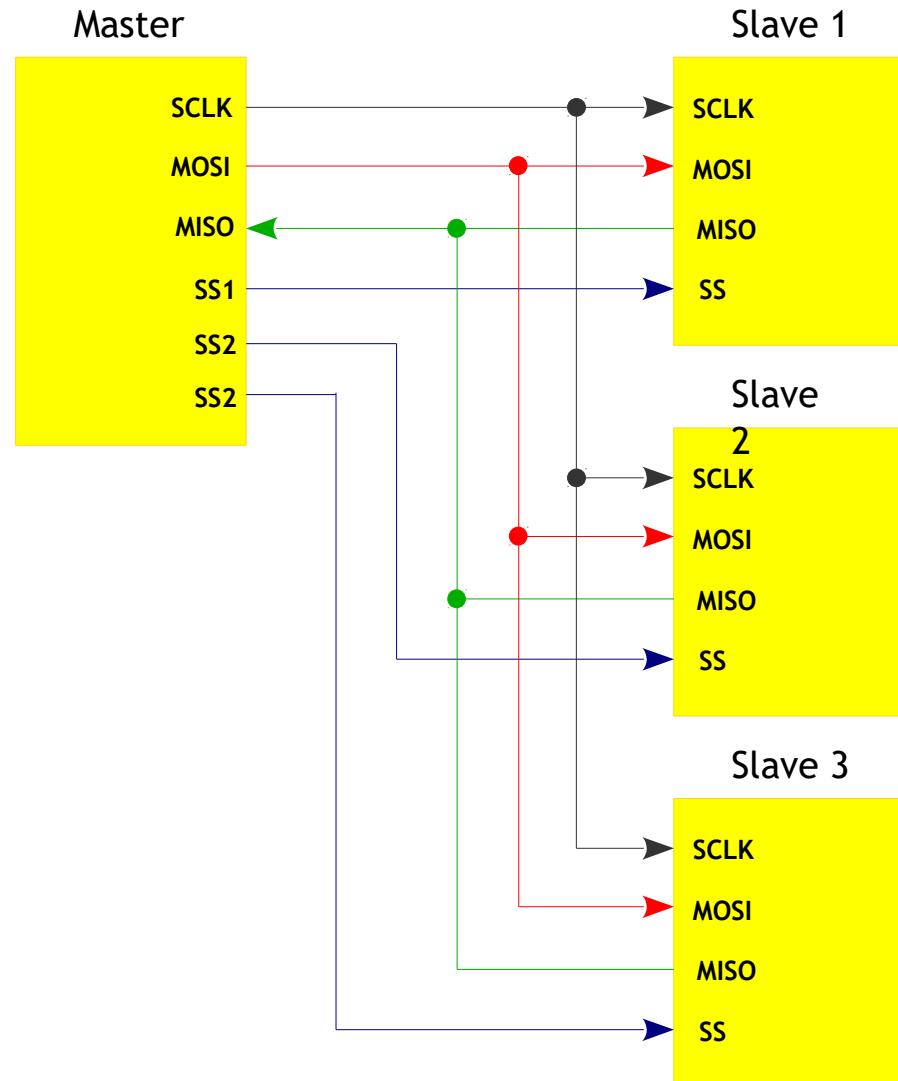


Single Master and Single Slave



SPI

Hardware Configuration

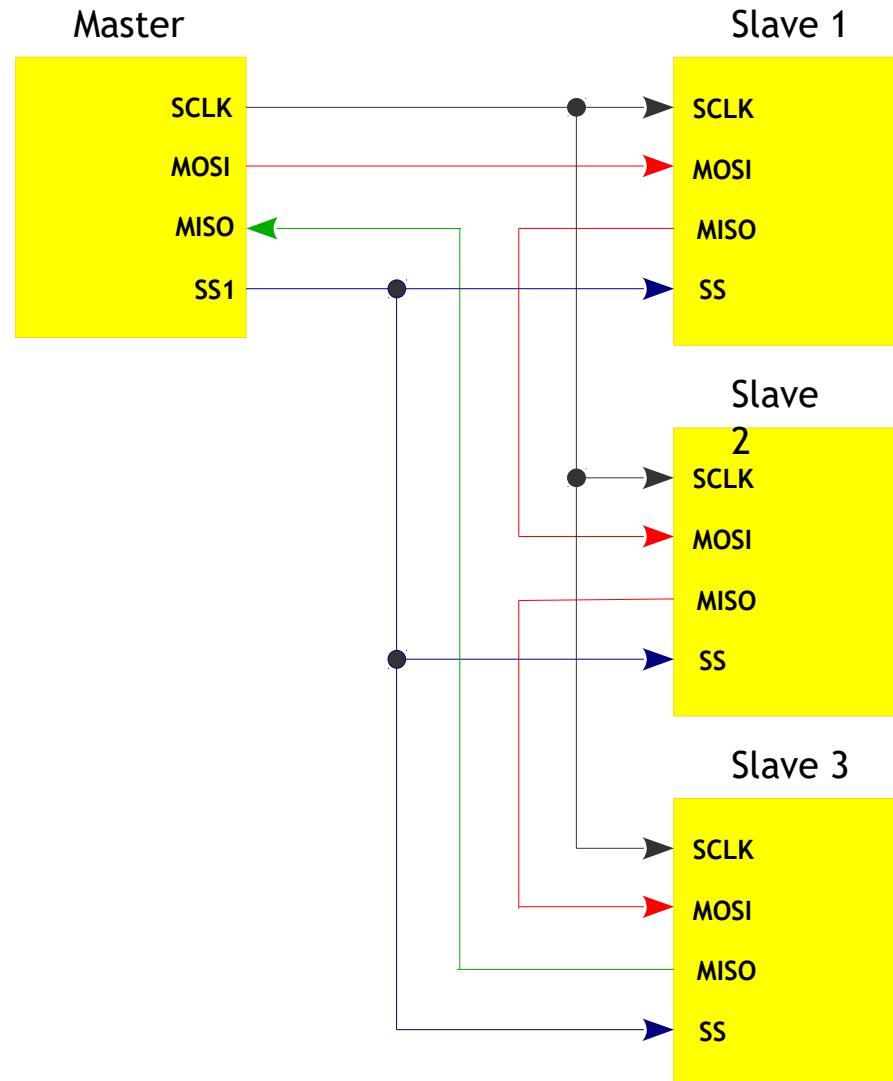


Single Master and Three Slaves



SPI

Hardware Configuration

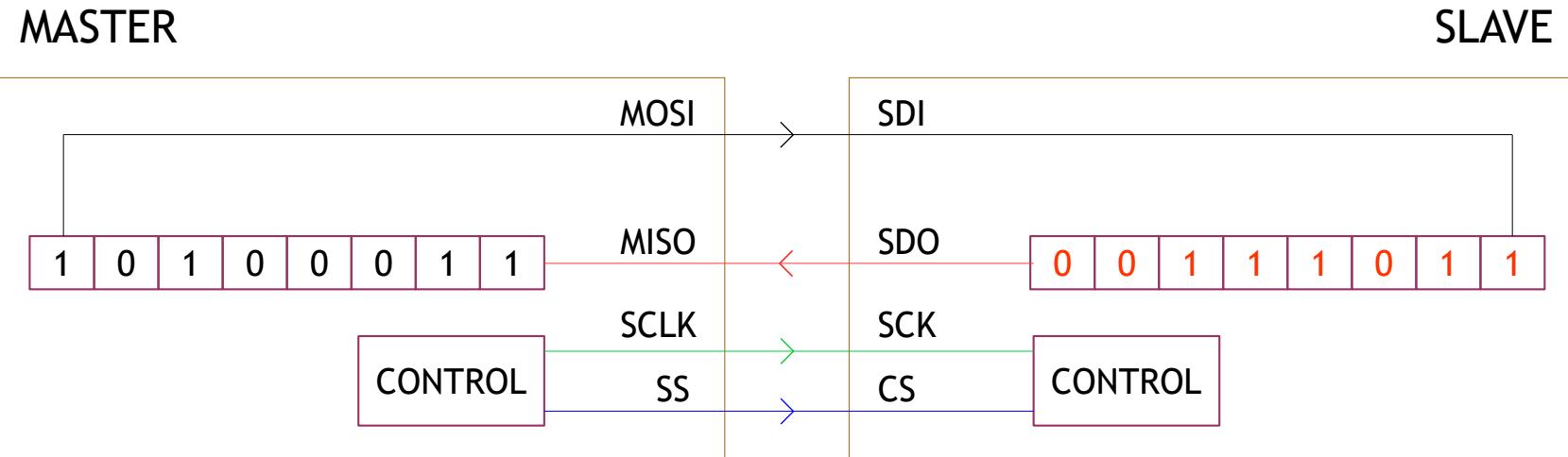


Single Master and Three Daisy Chained Slaves



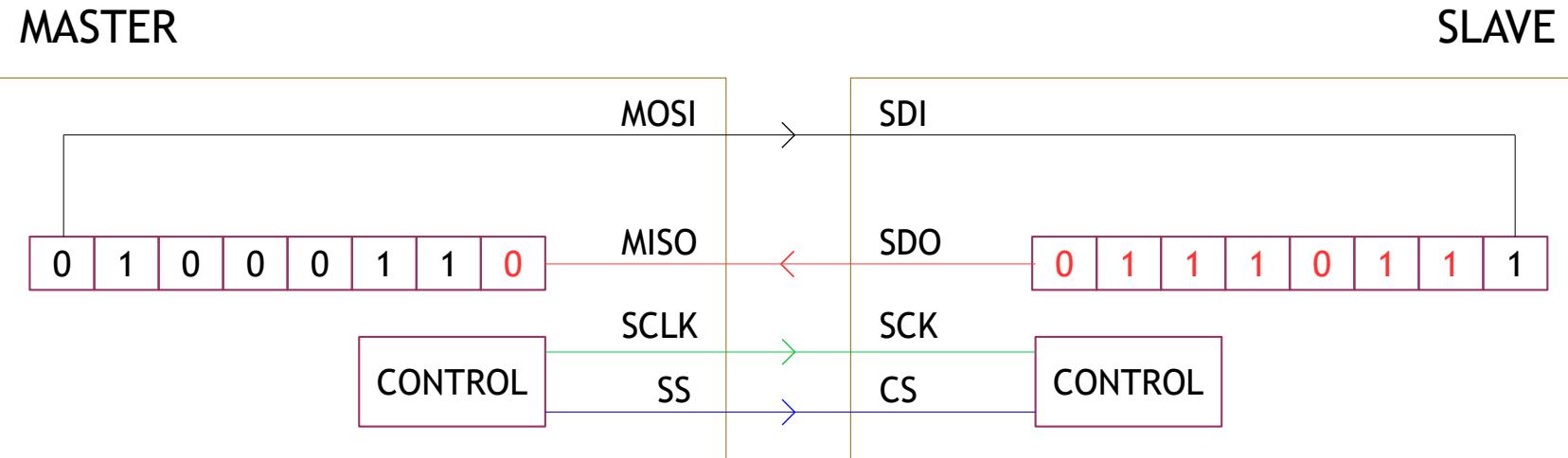
SPI

Data Transmission



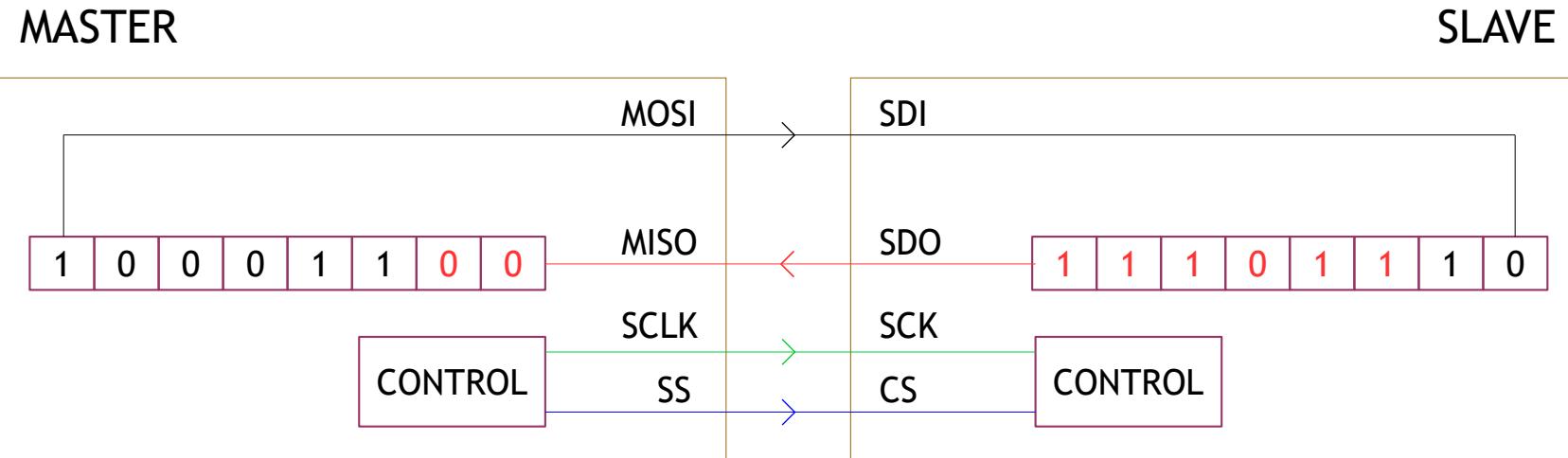
SPI

Data Transmission



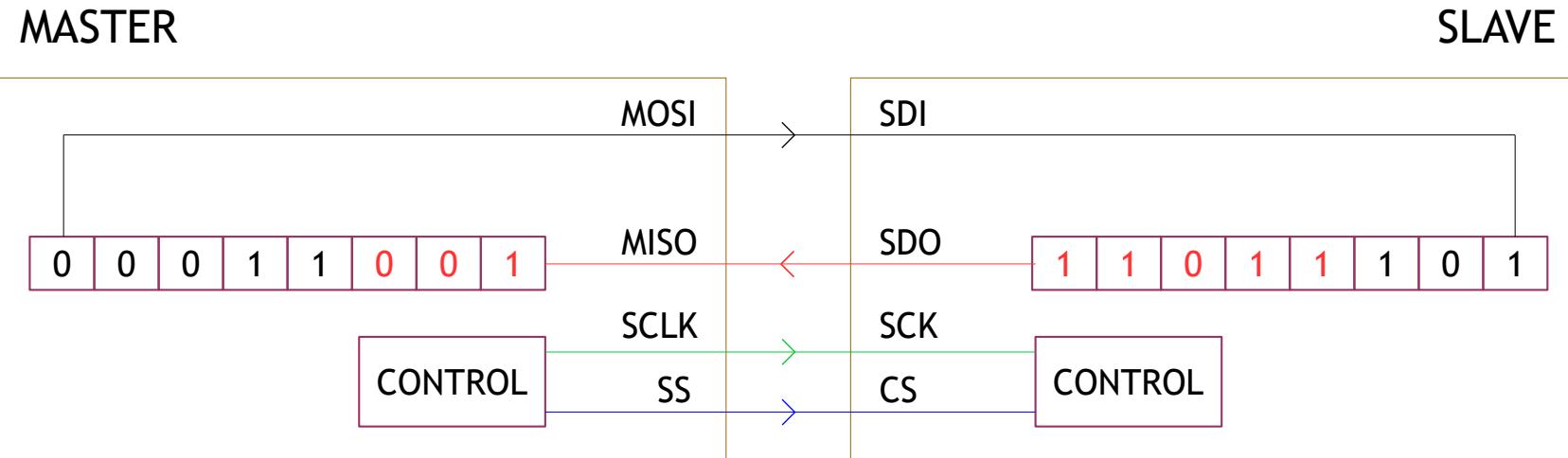
SPI

Data Transmission



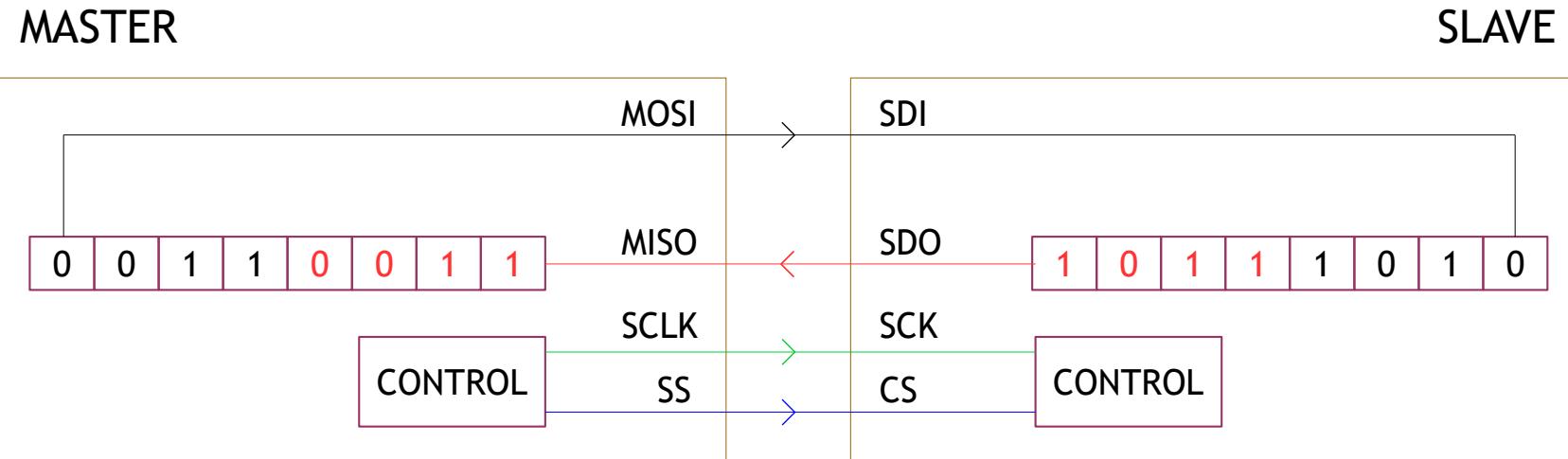
SPI

Data Transmission



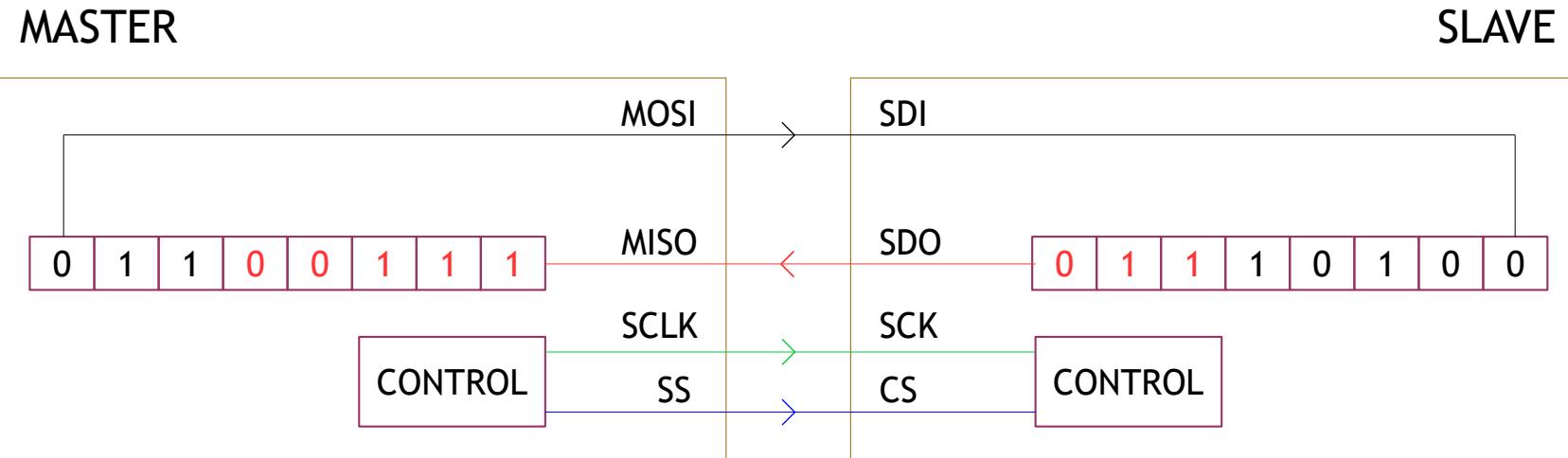
SPI

Data Transmission



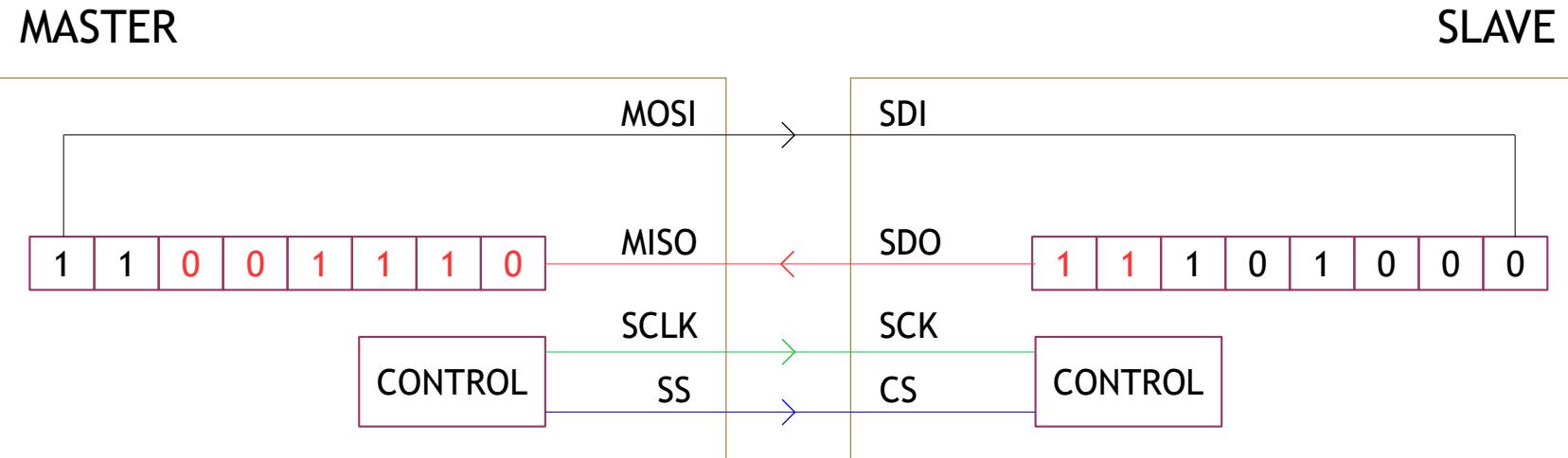
SPI

Data Transmission



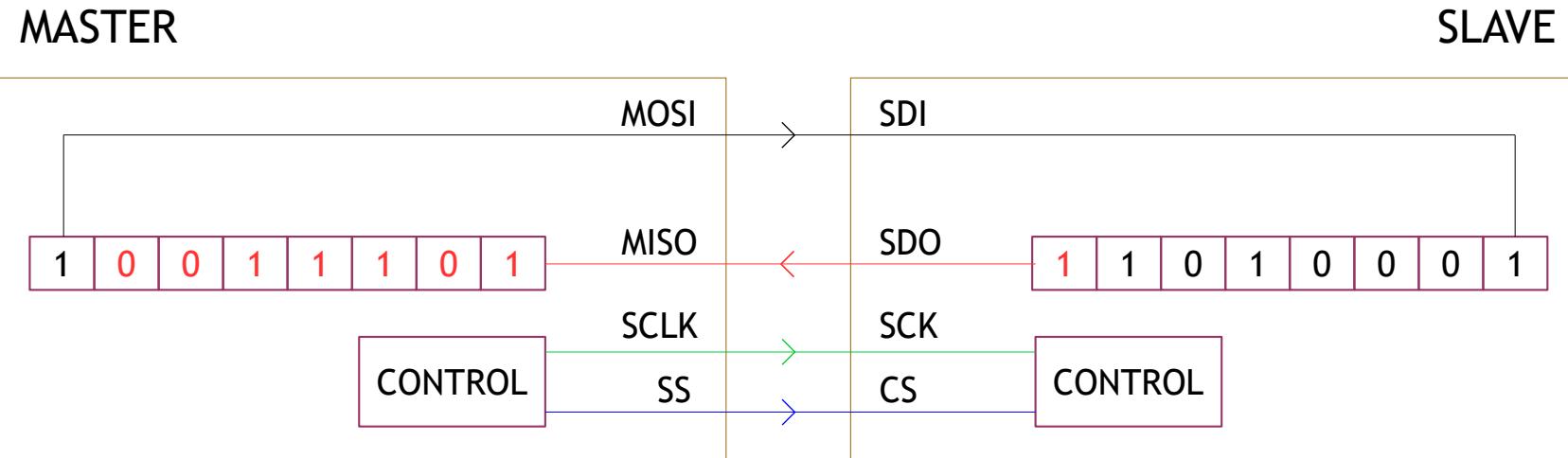
SPI

Data Transmission



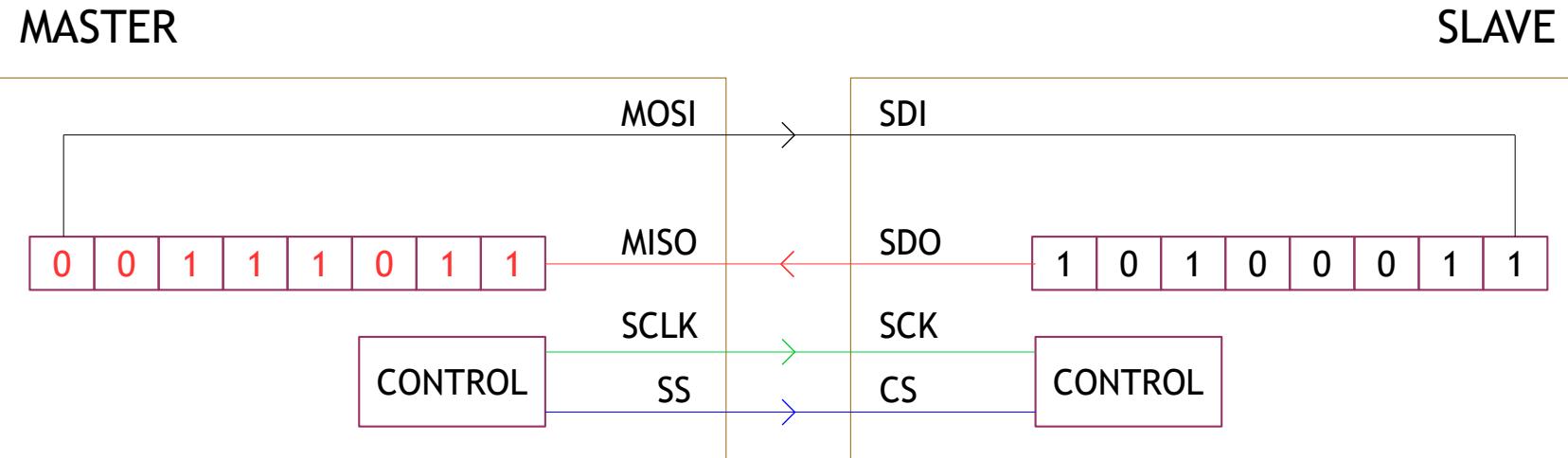
SPI

Data Transmission



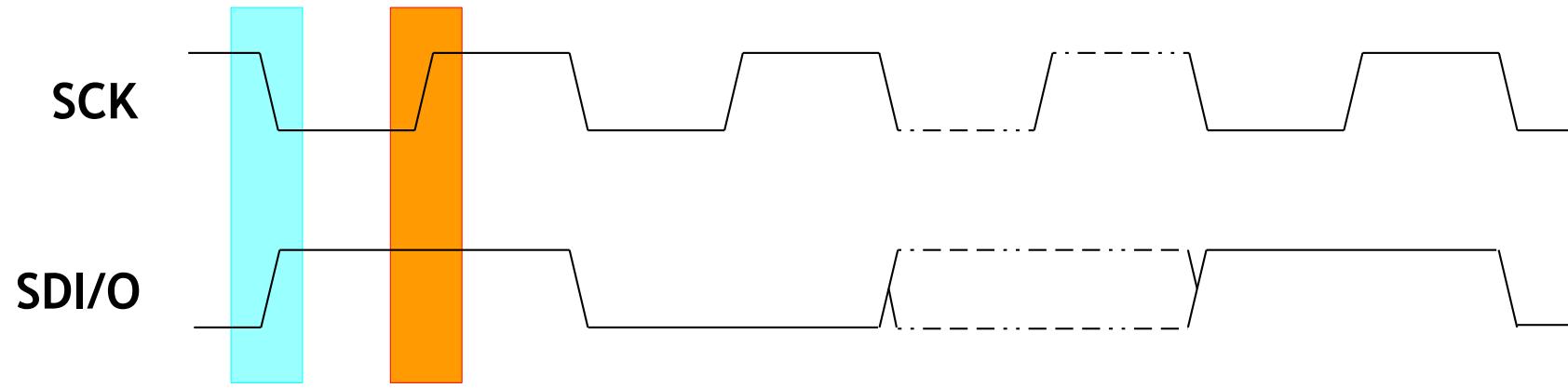
SPI

Data Transmission



SP

Data Validity



Data Write



Data Read



Inter Integrated Circuits



Inter Integrated Circuits

- Introduction
- Bus Features
- The Protocol
- Bus Speeds



I²C

Introduction

- Synchronous
- Half Duplex
- Multi Master / Slave



I²C

Bus Features



- Two Line Interface
- Software Addressable
- Multi Master with CD
- Serial, 8 bit Oriented, Bidirectional with 4 Modes
- On Chip Filtering



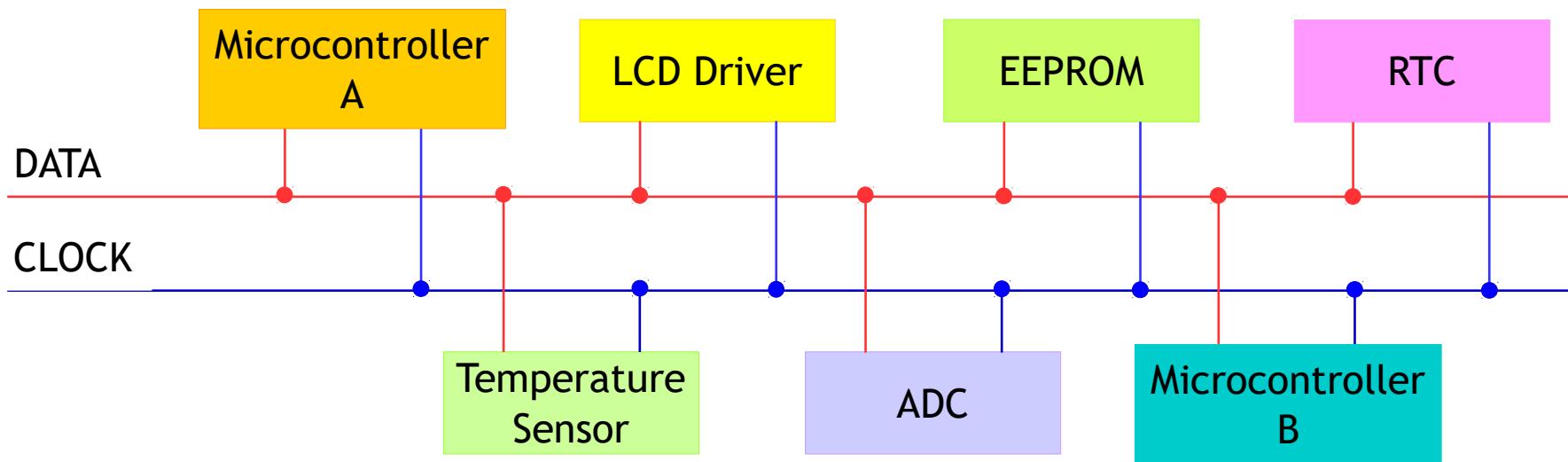
I²C Protocol

- Example
- Signals
- A Complete Data Transfer



I²C

Example



I²C Signals

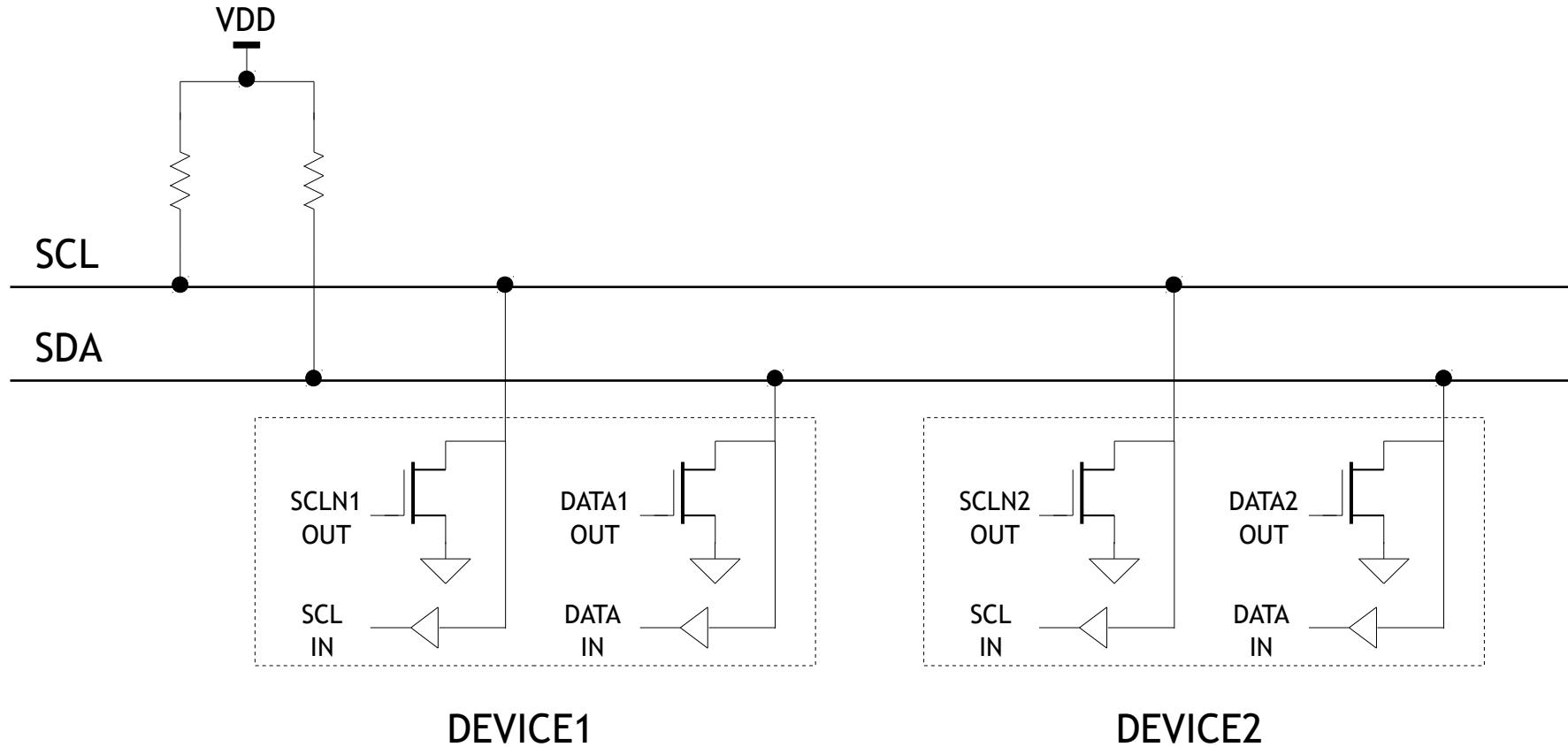


- Two-wired Interface
 - SDA
 - SCL
- Wired-AND
- Conditions and Data Validity
- Transmission



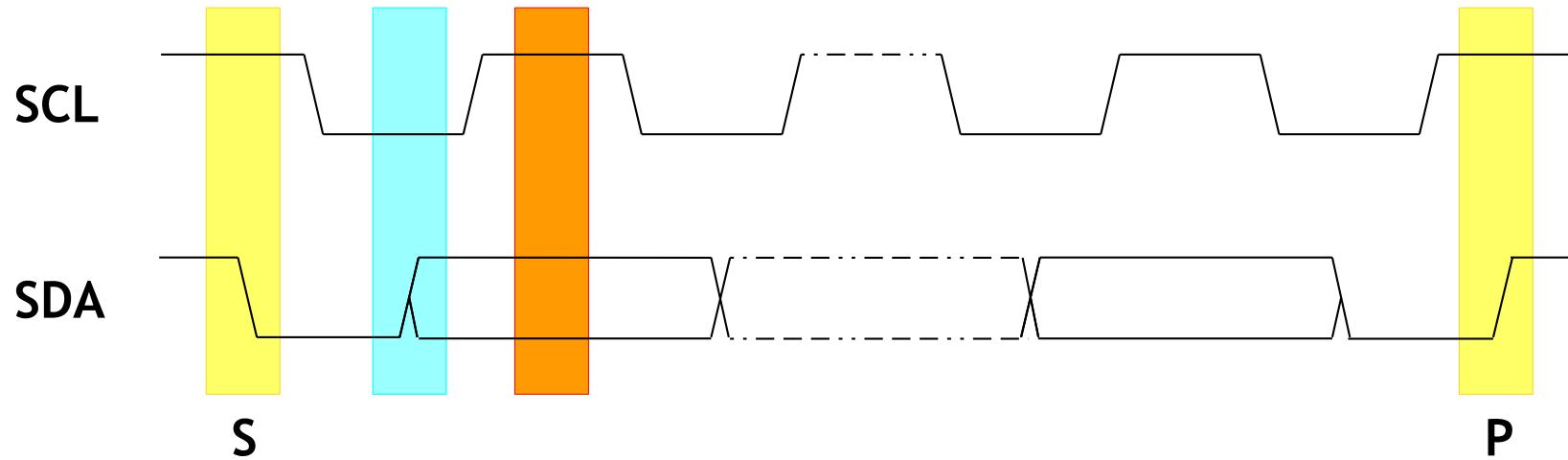
I²C

Signals - Wired-AND



I²C

Signals - Conditions and Data Validity



Data Write



Data Read



Conditions



MERTXE

I²C

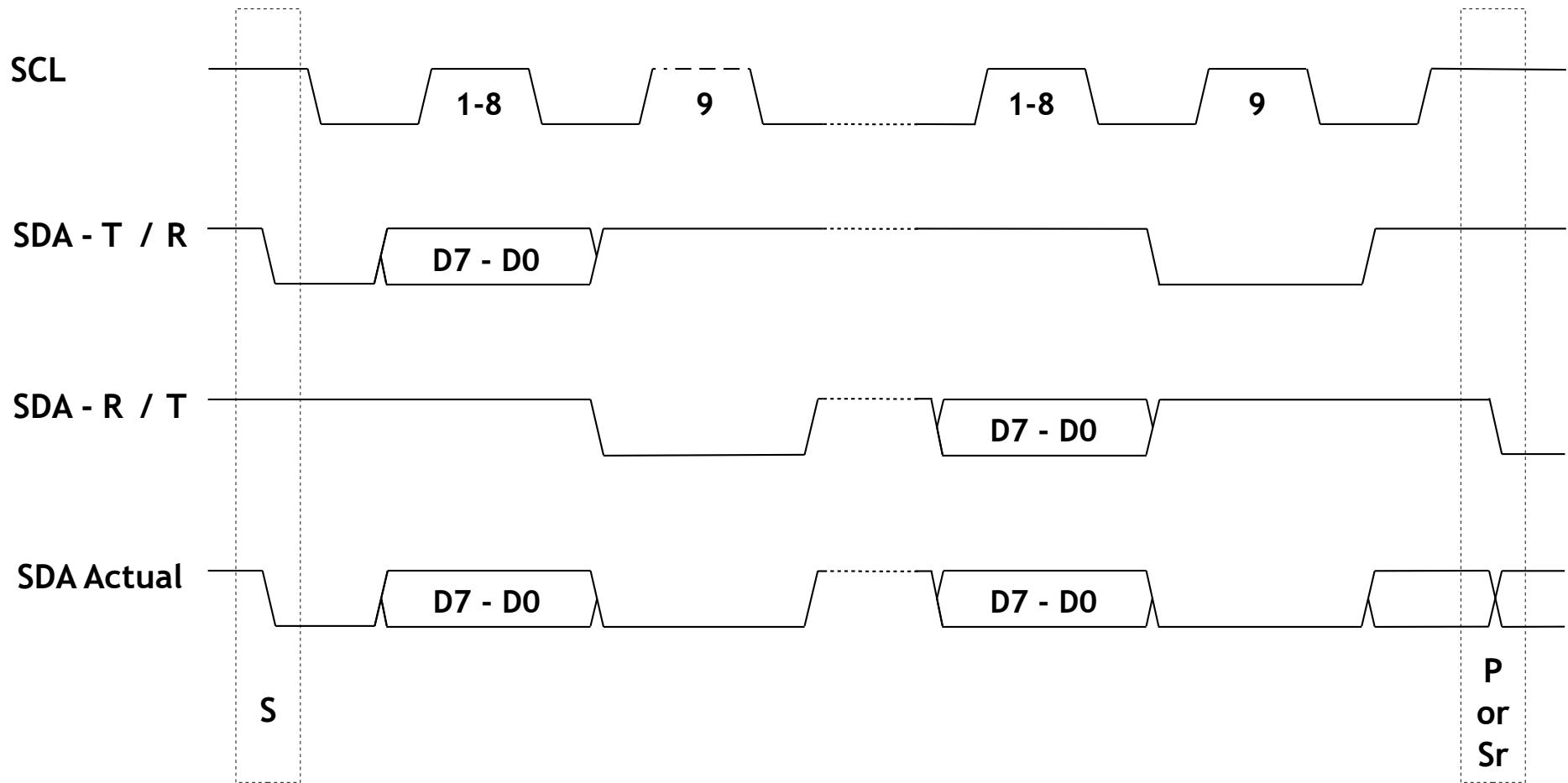
Signals - Transmission

- Data on SDA
- Clocking on SCL
- Clock Synchronization
- Data Arbitration



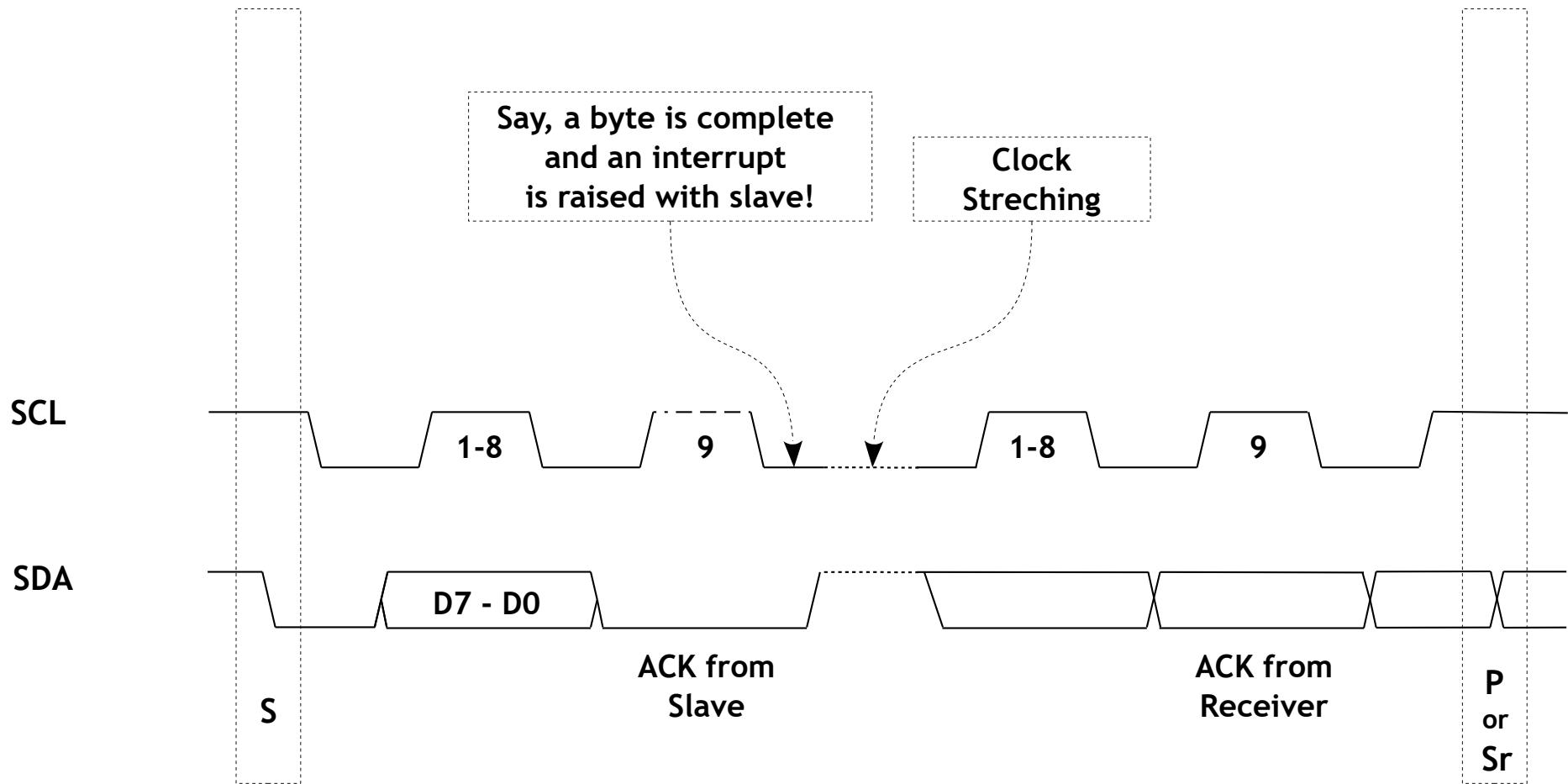
I²C

Signals - Data on SDA



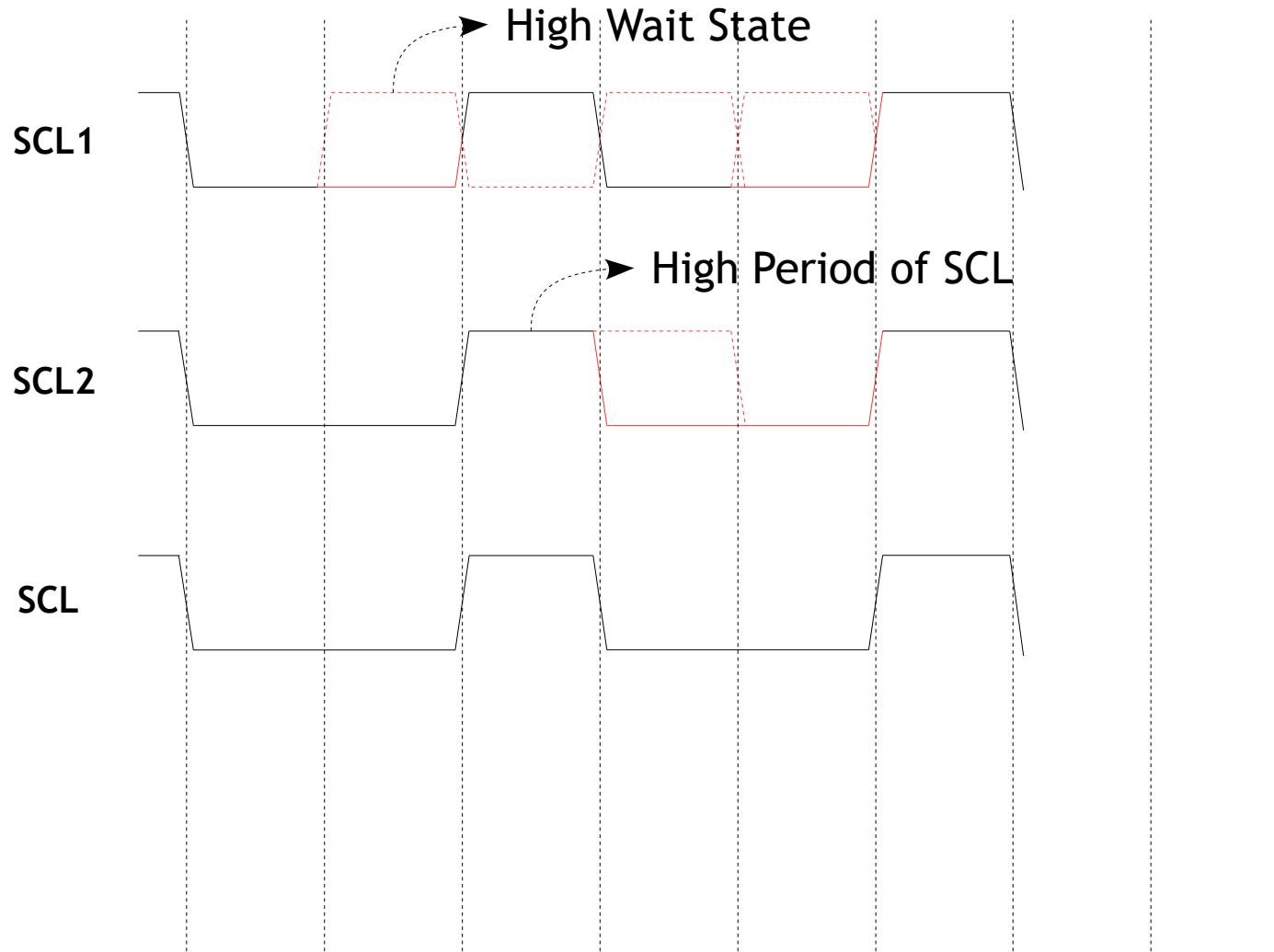
I²C

Signals - Clocking on SCL



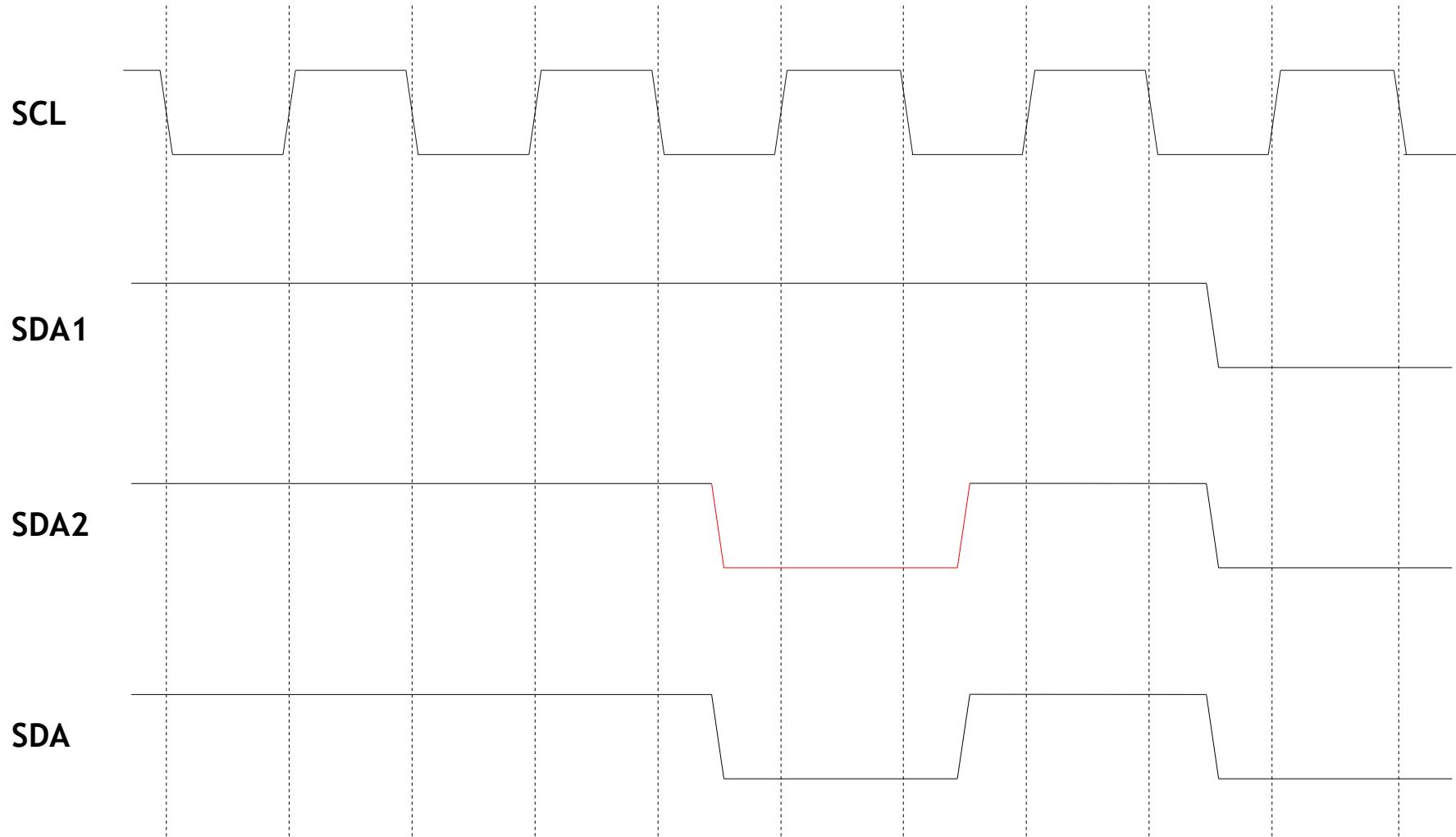
I²C

Signals - Clock Synchronization



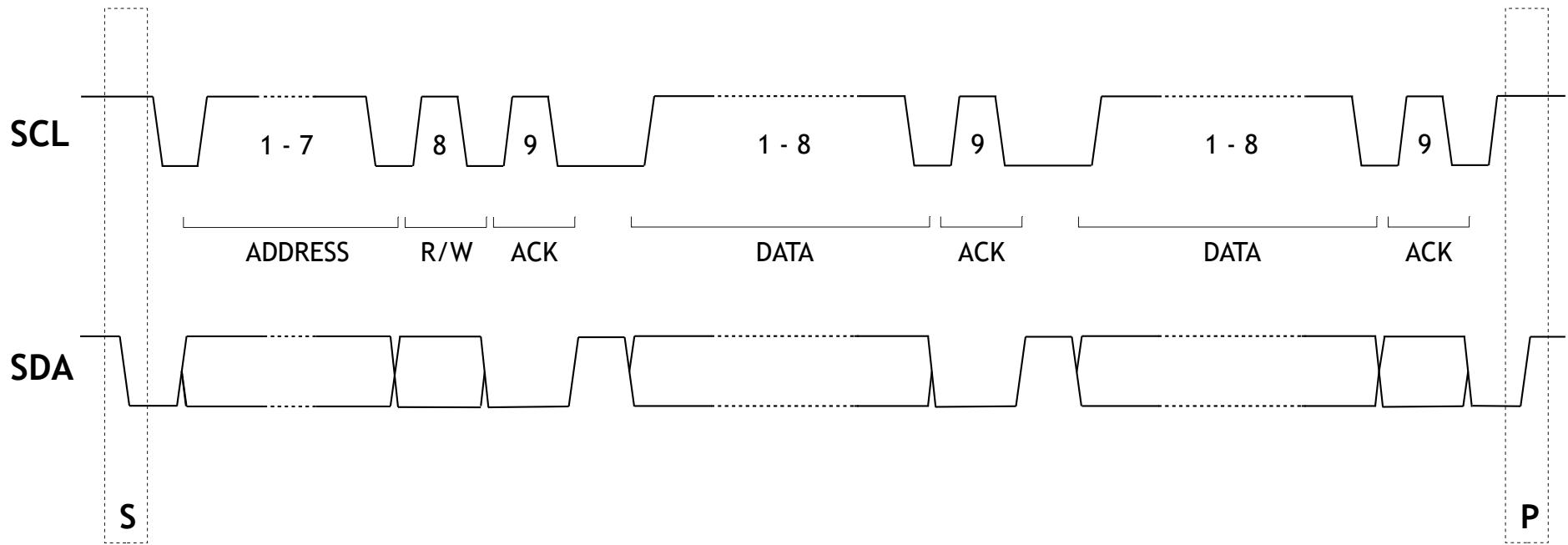
I²C

Signals - Data Arbitration



I²C

A Complete Data Transfer



I²C

Bus Speeds



- Bidirectional Bus
 - Standard Mode - 100 Kbit/s
 - Fast Mode - 400 Kbits/s
 - Fast Mode Plus - 1 Mbits/s
 - High Speed Mode - 3.4 Mbits/s
- Unidirectional Bus
 - Ultra Fast Mode - 5 Mbits/s
 - Uses Push-Pull Drivers (No Pullups)



Controller Area Network



Controller Area Network



- Introduction to CAN
- Basic Concepts
- Message Transfer
- Error Handling
- Fault Confinement



CAN

Introduction



- Asynchronous
- Half Duplex
- Multi Master / Slave

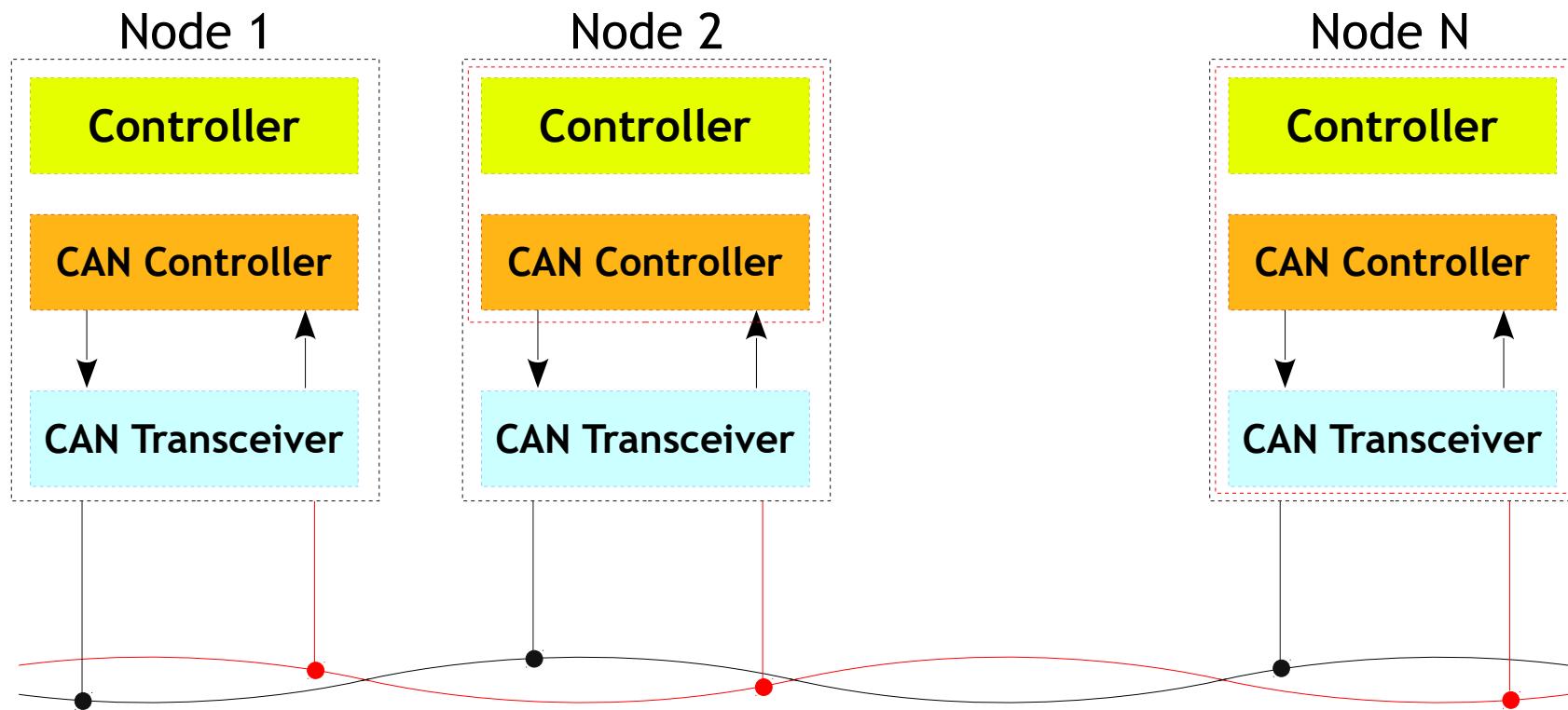


CAN

Basic Concepts



- Example
- Versions
- Absence of node addressing
 - Message identifier specifies contents and priority
 - Lowest message identifier has highest priority
- Non-destructive arbitration system by CSMA with collision detection
- Simple Transmission Medium
 - Twisted pair - CAN H and CAN L
- Properties
- Layered Architecture



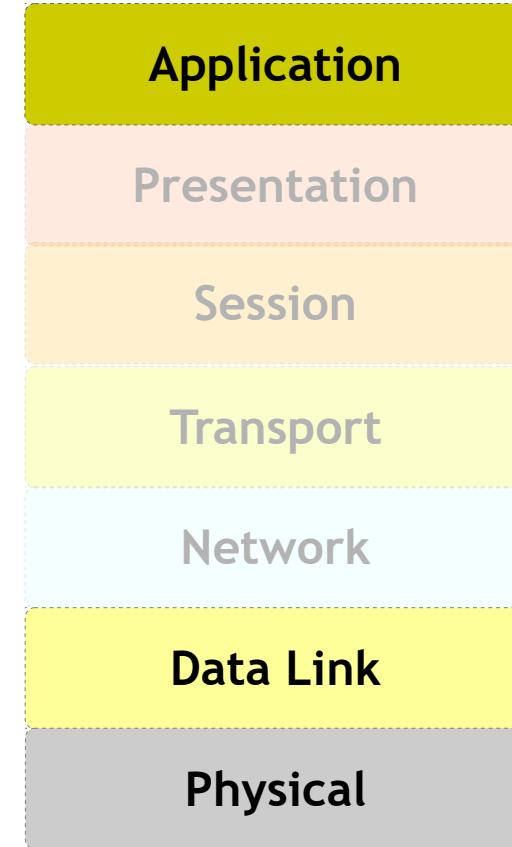
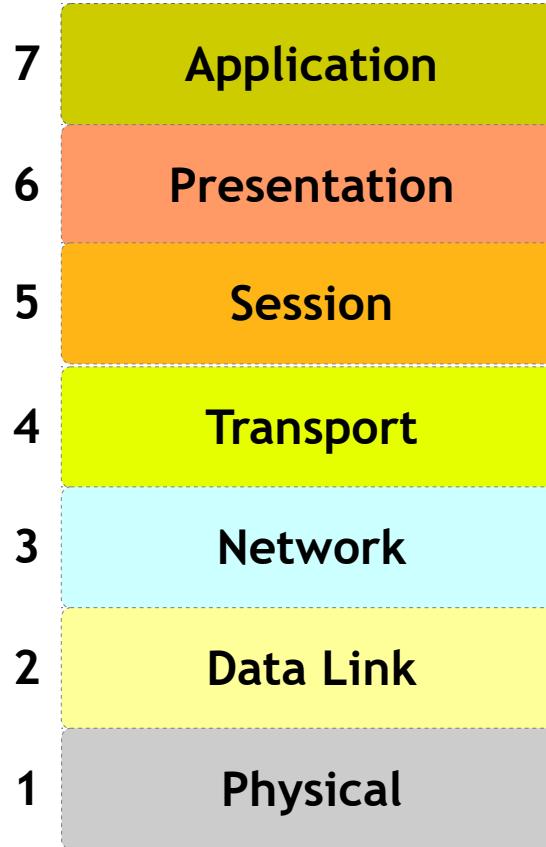
CAN

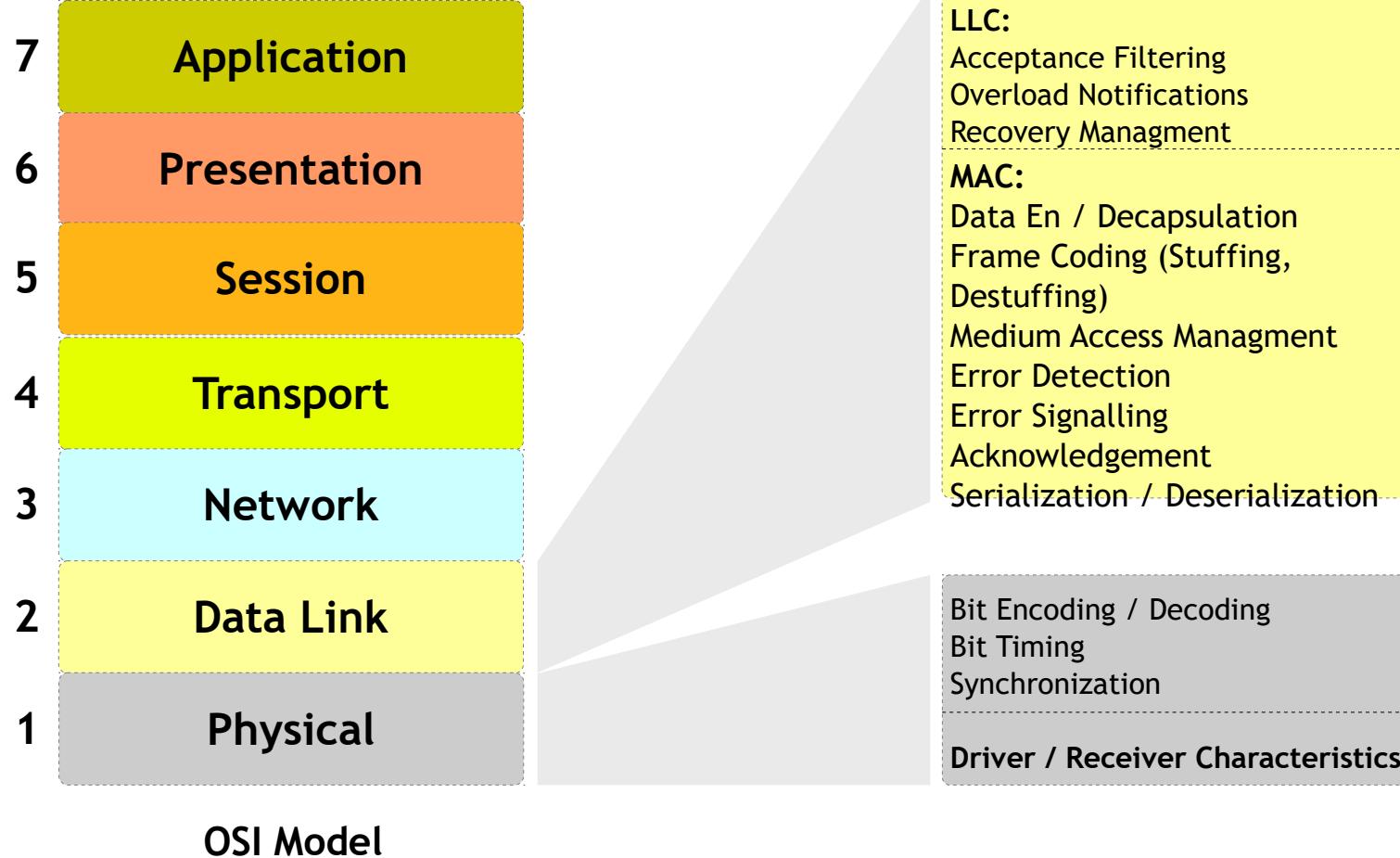
Basic Concepts - Versions



NOMENCLATURE	STANDARD	MAX SIGNALING RATE	IDENTIFIER
Low Speed CAN	ISO 11519	125 kbps	11 bit
CAN 2.0A	ISO 11898:1993	1 Mbps	11 bit
CAN 2.0B	ISO 11898:1995	1 Mbps	29 bit

- Prioritization of Messages
- Guarantee of Latency Times
- Configuration Flexibility
- Multicast Reception with Time Synchronization
- System wide Data Consistency
- Multi master
- Error Detection and Error Signaling
- Automatic Retransmission
- Distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

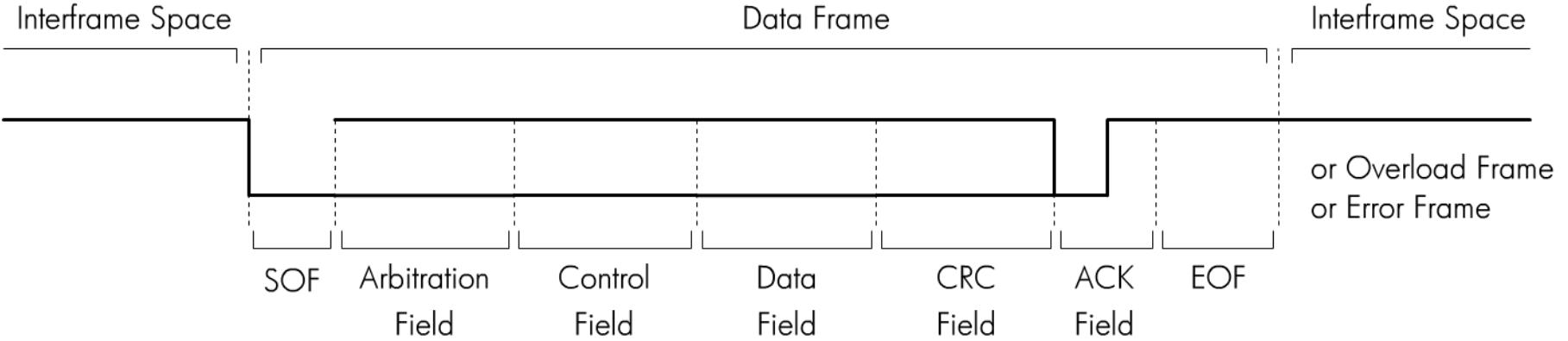




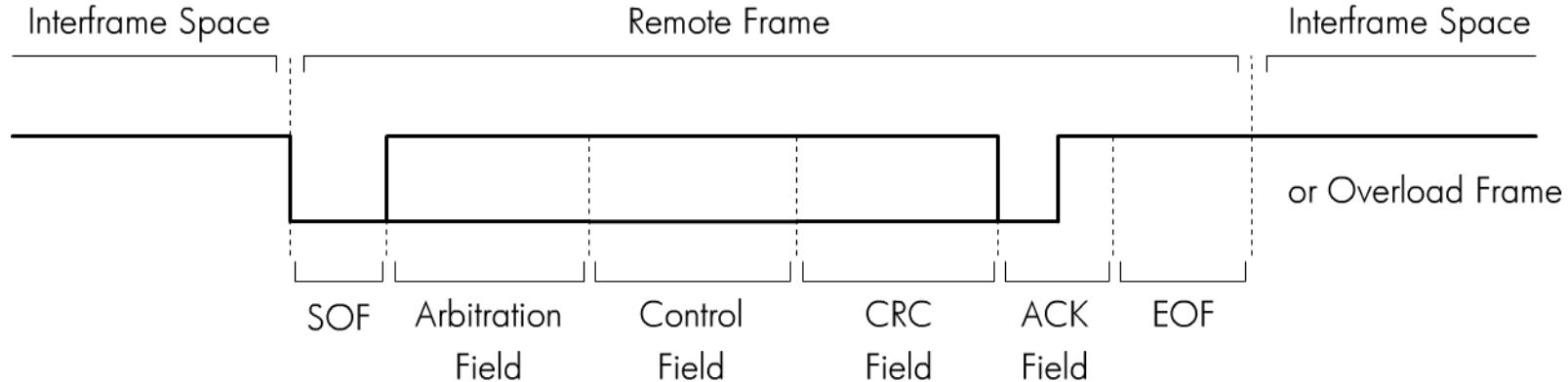


- Frame Formats
 - Standard Frame - 11 bits Identifiers
 - Extended Frame - 29 bits Identifiers
- Frame Types
 - Data Frame
 - Remote Frame
 - Error Frame
 - Overload Frame
- Frame Fields

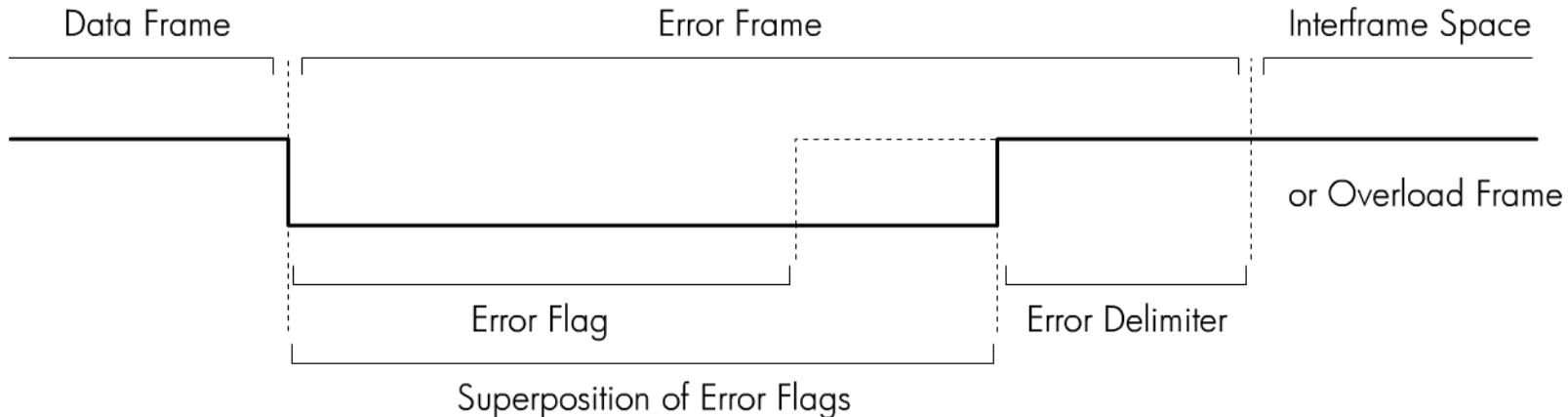




- A data frame consists of seven fields: start-of-frame, arbitration, control, data, CRC, ACK, and end-of-frame.

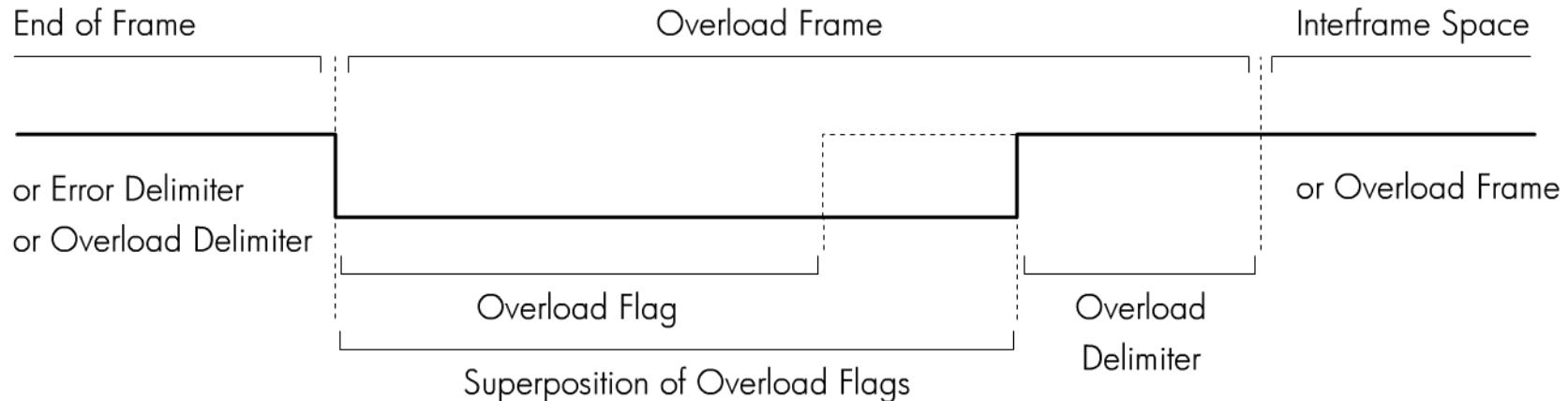


- Used by a node to request other nodes to send certain type of messages
- Has six fields as shown in above figure
 - These fields are identical to those of a data frame with the exception that the RTR bit in the arbitration field is **recessive** in the remote frame.



- This frame consists of two fields.
 - The first field is given by the superposition of error flags contributed from different nodes.
 - The second field is the error delimiter.
- Error flag can be either active-error flag or passive-error flag.
 - Active error flag consists of six consecutive dominant bits.
 - Passive error flag consists of six consecutive recessive bits.
- The error delimiter consists of eight recessive bits.

Message Transfer - Overload Frame



- Consists of two bit fields: overload flag and overload delimiter
- Three different overload conditions lead to the transmission of the overload frame:
 - Internal conditions of a receiver require a delay of the next data frame or remote frame.
 - At least one node detects a dominant bit during intermission.
 - A CAN node samples a dominant bit at the eighth bit (i.e., the last bit) of an error delimiter or overload delimiter.
- Format of the overload frame is shown in above fig
- The overload flag consists of six dominant bits.
- The overload delimiter consists of eight recessive bits.

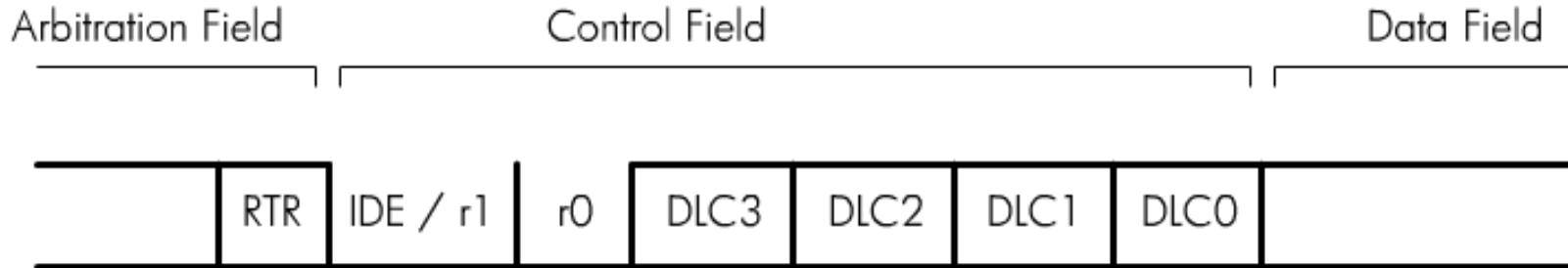
CAN

Message Transfer - Frame Fields



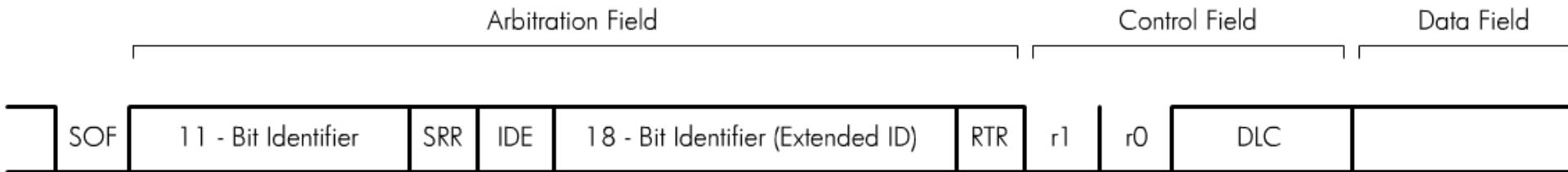
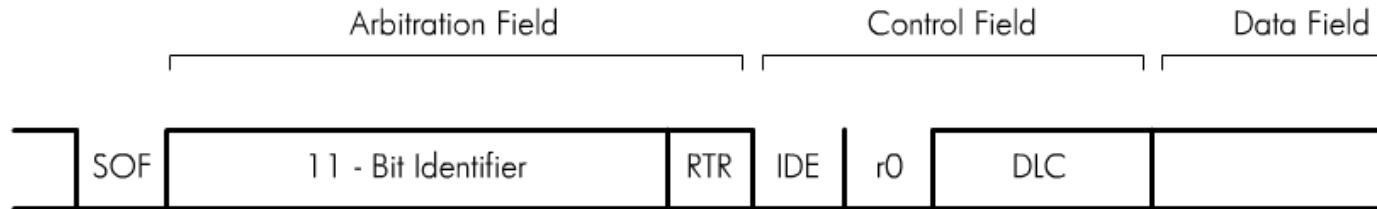
- Control Field
- Arbitration Field
- Data Field
- CRC Field
- ACK Field





- The first bit is IDE bit for the standard format but is used as reserved bit r1 in extended format.
- r0 is reserved bit.
- DLC3...DLC0 stands for data length and can be from 0000 (0) to 1000 (8).

Frame Fields - Arbitration Field



- The identifier of the standard format corresponds to the base ID in the extended format.
- The RTR bit is the remote transmission request and must be 0 in a data frame.
- The SRR bit is the substitute remote request and is recessive.
- The IDE field indicates whether the identifier is extended and should be recessive in the extended format.
- The extended format also contains the 18-bit extended identifier.

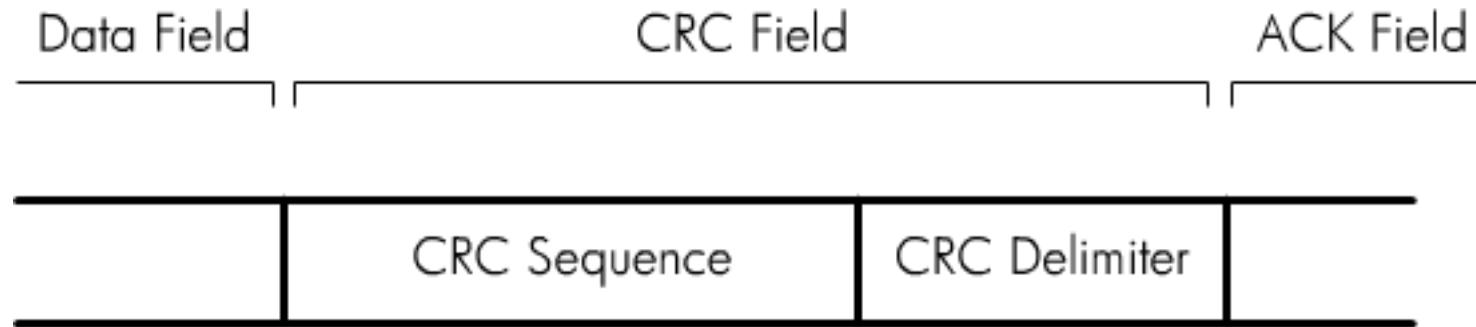
CAN

Frame Fields - Data Field



- May contain 0 to 8 bytes of data





- It contains the 16-bit CRC sequence including CRC delimiter.
- The CRC delimiter is a single **recessive bit**.

- Consists of two bits
- The first bit is the **acknowledgement bit**.
- This bit is set to recessive by the transmitter, but will be reset to dominant if a receiver acknowledges the data frame.
- The second bit is the **ACK delimiter** and is recessive.

CAN

Error Handling

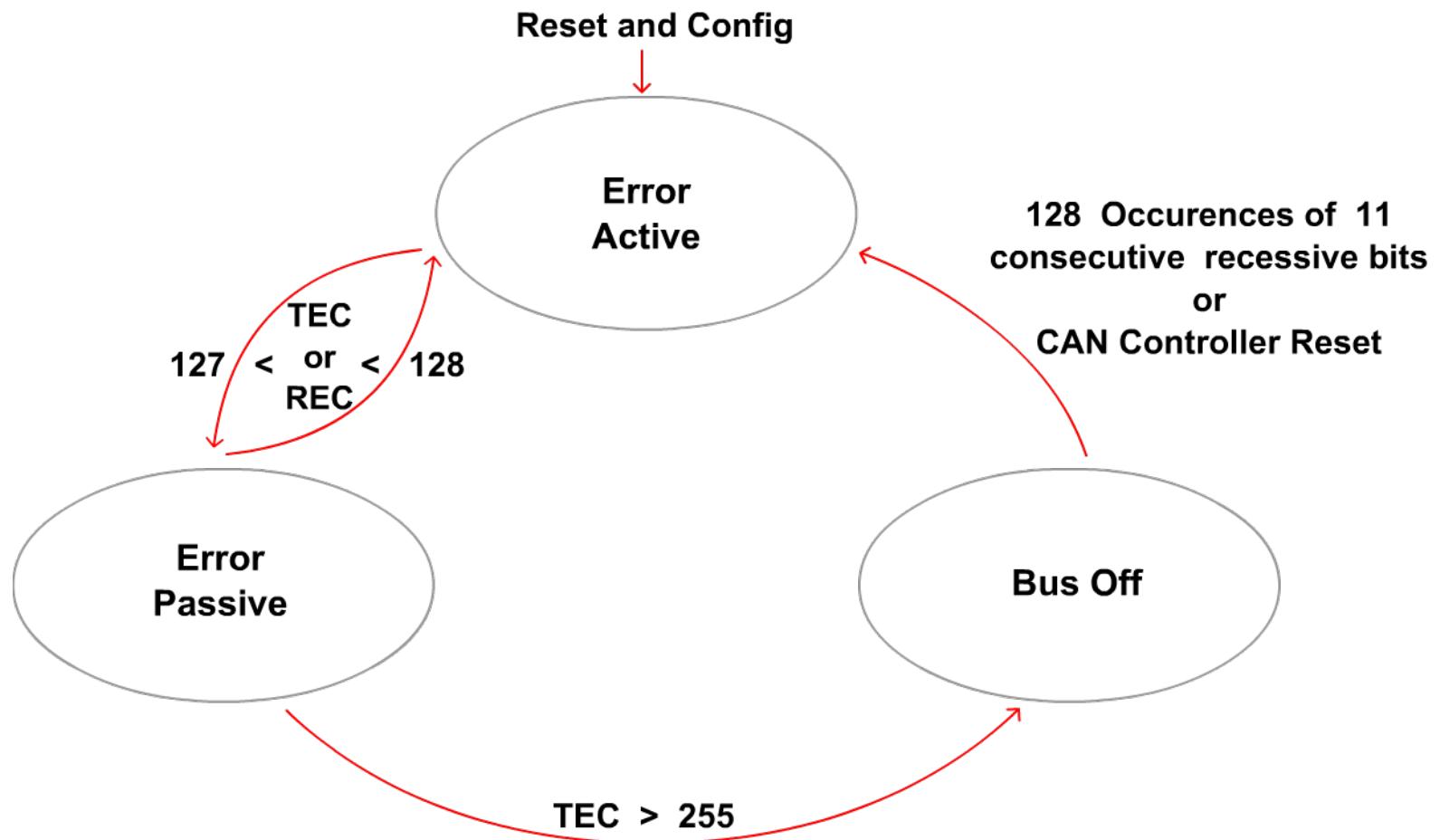


- Error Detection
 - Bit Error
 - Stuff Error
- Error Signaling
 - CRC Error
 - Form Error
 - Acknowledgment Error





- Counters
 - Transmit Error Counter & Receive Error Counter



Thank You