- CO3009 -

- Microprocessor and Microcontroller -

Course Introduction

- Instructor
  - **Pham Hoang Anh, Ph.D**
  - Faculty of Computer Science and Engineering
  - HCMC University of Technology
  - Email: anhpham@hcmut.edu.vn
  - Phone: (84)(8) 38647256 (Ext. 5843)
  - Course materials: BKeL

What will you learn in this class?

- Understand microcontroller basics including all common interfaces.
- How to write an efficient firmware for embedded systems using the C programming language.

Why do you study this class?

- It is the implementation basic of all embedded, real time systems.
- This course is part of Computer Engineering program.

anhpham@hcmut.edu.vn

Learning Outcome

- L.O.1 Understanding of the model of systems using Microprocessors - Microcontrollers
  - L.O.1.1 – Fetching and executing an instruction from the system memory.
  - L.O.1.2 – Program counter, instruction decode and arithmetic and logic unit (ALU).
  - L.O.1.3 – The program memory and data memory, the busses.
  - L.O.1.4 – Methods of memory addressing.
  - L.O.1.5 – Conceptions of RISC and CISC.
  - L.O.1.6 – Conceptions of Microprocessors - Microcontrollers
- L.O.2 Understanding of the Microprocessors - Microcontrollers architecture
  - L.O.2.1 – Oscillator configuration.
  - L.O.2.2 – Reset.
  - L.O.2.3 – Memory organization.
  - L.O.2.4 – Reading operands from program memory.
  - L.O.2.5 – Program instruction sequencing.
  - L.O.2.6 – The CPU and its status bits.
  - L.O.2.7 – The special function registers.

anhpham@hcmut.edu.vn

Learning Outcome (cont)

- L.O.3 Understanding of the interrupt and its mechanism
  - L.O.3.1 – Analyze and demonstrate the operation of hardware interrupts.
  - L.O.3.2 – Understand the interrupt service routines (ISRs).
  - L.O.3.3 – Interrupt operations.
- L.O.4 Programing to parallel and serial peripheral interfaces
  - L.O.4.1 – Design basic parallel interface for I/Os.
  - L.O.4.2 – SPI, I2C, UART peripheral interfaces.
- L.O.5 Understand the timer operation and its usages.
  - L.O.5.1 – The timers are used for interval timing and event counting.
  - L.O.5.2 – The timer interrupts.
  - L.O.5.3 – Using timer to implement an embedded operating system.
  - L.O.5.4 – Using timer to implement a co-operative scheduler.

Learning Outcome (cont)

- L.O.6 Special features of Microprocessors - Microcontrollers
  - L.O.6.1 – Analog-to-Digital conversion (ADC).
  - L.O.6.2 – Capture, compare, and pulse width modulation (PWM) functions.
  - L.O.6.3 – Watchdog Timer (WDT).
- L.O.7 Understand a multi-stage system for Microprocessors - Microcontrollers.
  - L.O.7.1 – Design and implement a timed multi-stage system
  - L.O.7.2 – Design and implement a timed/input multi-stage system
- L.O.8 How to choose the right microcontroller?

Tentative Contents

- Microcontroller Basic
- Embedded Software Architectures
- Embedded C Programming
- Adding Structure To Your Code
- Multi-state Systems and Function Sequences
- Creating an Embedded Operating System
- A co-operative scheduler
- Special Functions
- How Do I Choose the Right MCU?

Grading Policy

- Grading
  - Lab: 30%
  - Midterm: 20%
  - Final Exam: 50%
- Presentation
  - Journal Articles
  - Related Topics

https://www.hackster.io/projects?ref=topnav
http://www.electronicshub.org/iot-project-ideas

Textbooks

- Michael J. Pont, Patterns for time-triggered embedded systems, 2001, Addison-Wesley / ACM Press
- Internet

QnA

- CO3009 -

- Microcontroller Basics -

Microprocessors vs. Microcontrollers

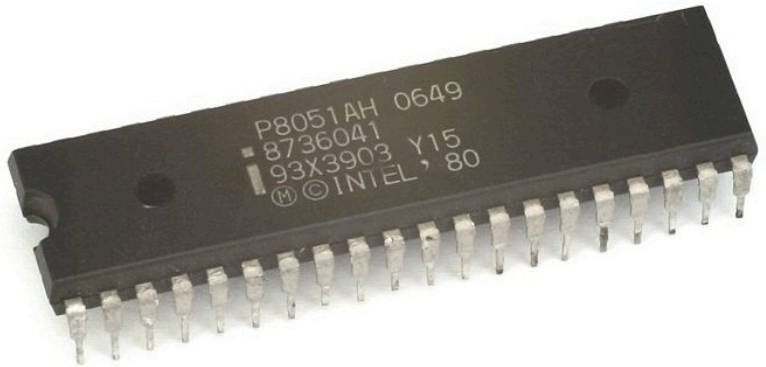
- Microprocessors (µPs)
  - General-purpose compute "engine"
  - External memory and I/O devices
  - Often requires an operating system (OS)

- Microcontrollers (MCUs)
  - Usually chosen for a specific purpose
  - Small packages
  - On-chip memory and peripherals
  - Fast "on time," no BIOS or OS needed

- https://www.youtube.com/watch?v=dcNk0urQsQM

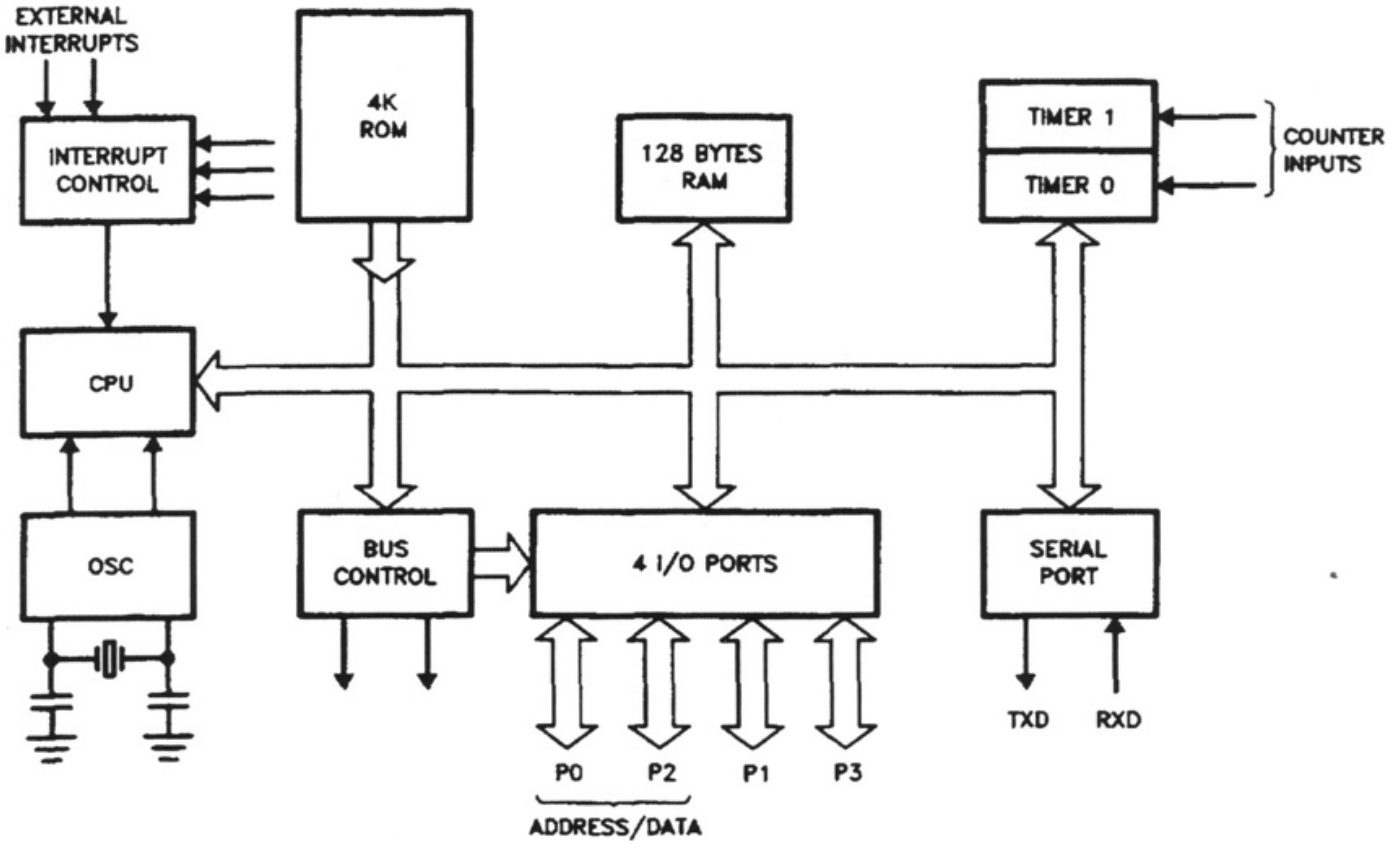
Where Did the MCU Come From?

- Intel introduced the 8051 MCU in 1980
  - Small amount of read-only memory (ROM)
  - External memory expansion if needed
  - Four 8-bit I/O ports
  - Not much different from today's MCUs



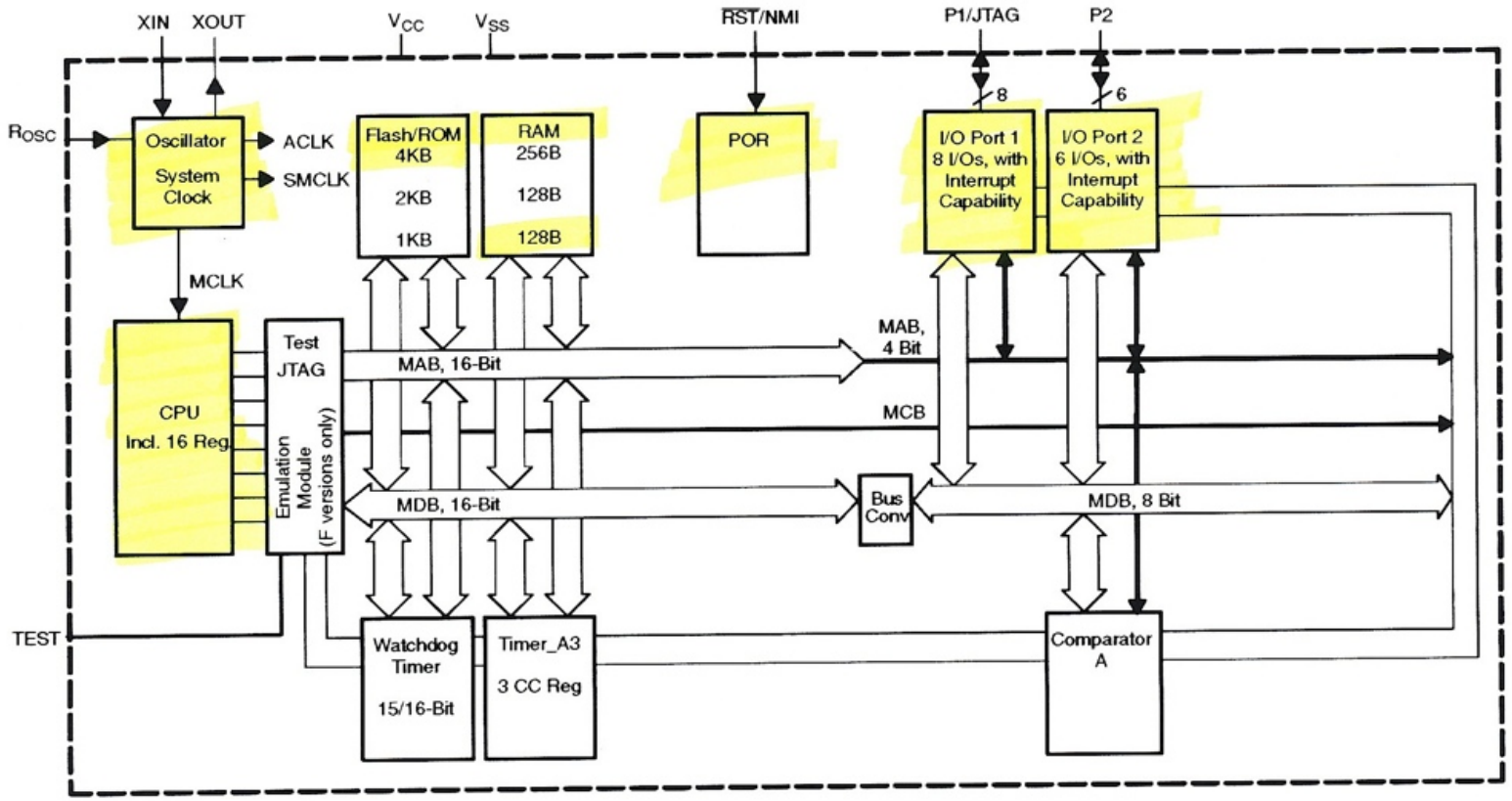
anhpham@hcmut.edu.vn

8051 Architecture (1980)

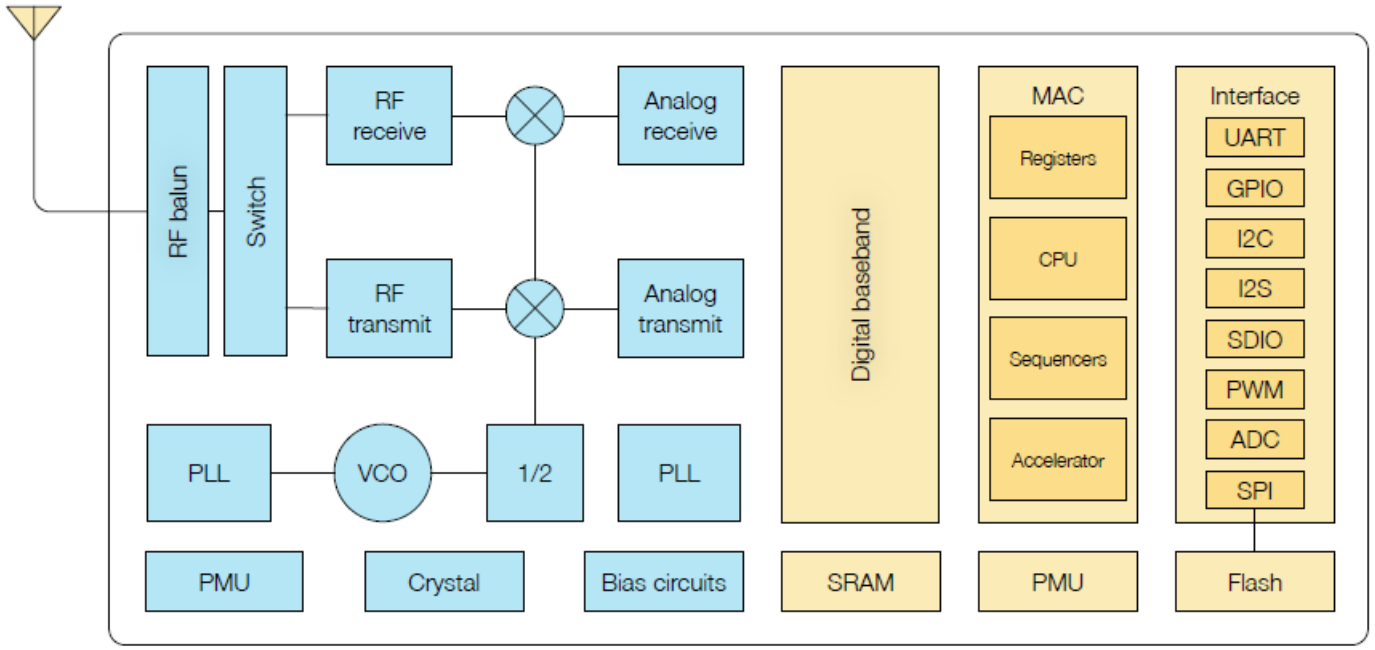


anhpham@hcmut.edu.vn

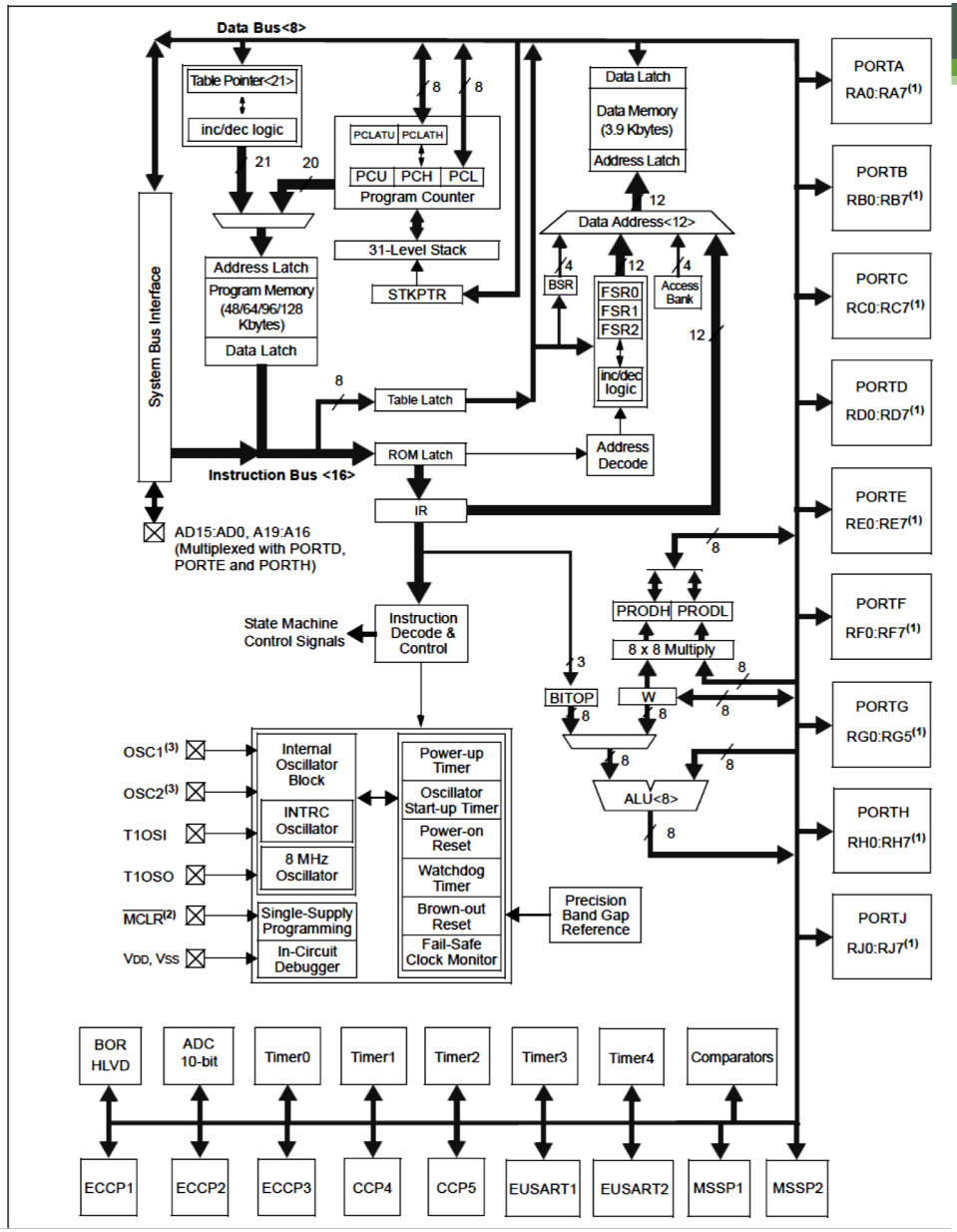
Texas Instruments MSP430 MCU



ESP8266


RF balun
Switch
RF receive
RF transmit
Analog receive
Analog transmit
PLL
VCO
1/2
PLL
Digital baseband
MAC
Registers
CPU
Sequencers
Accelerator
Interface
UART
GPIO
I2C
I2S
SDIO
PWM
ADC
SPI
PMU
Crystal
Bias circuits
SRAM
PMU
Flash

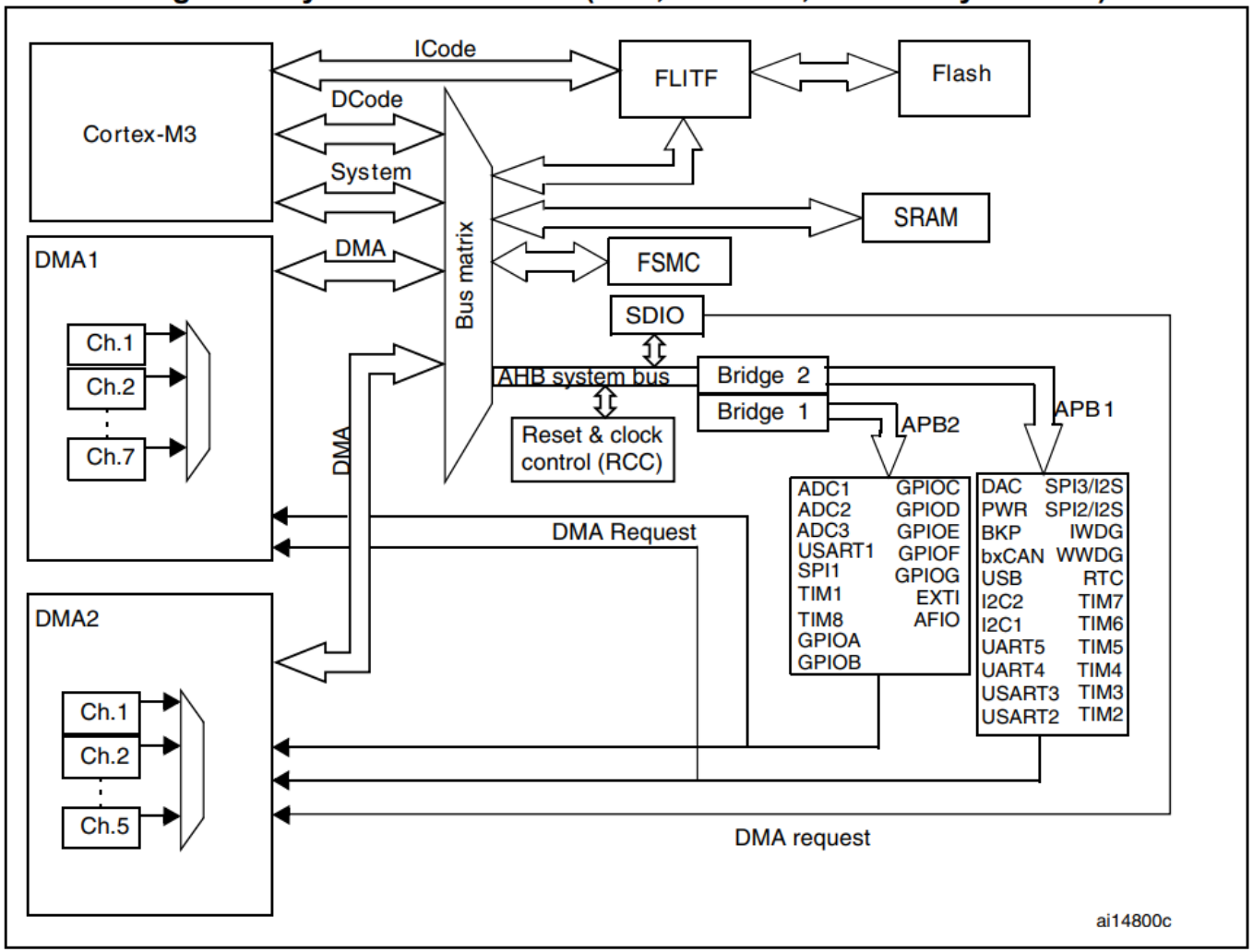
PIC18F8722



anhpham@hcmut.edu.vn

STM32

Figure 1. System architecture (low-, medium-, XL-density devices)



anhpham@hcmut.edu.vn

Why Use an MCU?

- Everything in one small package
- Mix and match peripherals and I/O types
- Lots of memory, flash for code, SRAM for data
- Readily available hardware and software tools
- Helpful support communities and forums
- Reference designs
- Code libraries and examples

What Peripherals Do MCUs Offer?

- Digital I/O -- On or Off
  - Parallel signals
  - Pulse-width-modulated logic signals
  - Counters and timers
- Analog I/O -- Voltages
  - Comparators
  - Analog-to-digital converters (ADCs)
  - Digital-to-analog converters (DACs)
- Communication Devices
  - UART, USART, SPI, I2C
  - CAN, USB, Ethernet
- Interrupts

Peripherals Devices in MCUs

- Parallel I/O Ports
  - Usually 8 or 16 bits for simultaneous control
  - Toggle individual bits
  - Require setup of registers

- Parallel I/O-Port Examples

anhpham@hcmut.edu.vn

How Do I Set Up I/O Ports?

TABLE 2-1: PIC16F631/677/685/687/689/690 SPECIAL FUNCTION REGISTERS SUMMARY BANK 0

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	44,205
01h	TMR0	Timer0 Module Register								xxxx xxxx	81,205
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	44,205
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	36,205
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	44,205
05h	PORTA(7)	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--xx xxxx	59,205
06h	PORTB(7)	RB7	RB6	RB5	RB4	—	—	—	—	xxxx ----	69,205
07h	PORTC(7)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	76,205
08h	—	Unimplemented								—	—
09h	—	Unimplemented								—	—
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter					---0 0000	44,205
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RABIE	T0IF	INTF	RABIF(1)	0000 000x	38,205
0Ch	PIR1	—	ADIF(4)	RCIF(2)	TXIF(2)	SSPIF(5)	CCP1IF(3)	TMR2IF(3)	TMR1IF	-000 0000	41,205
0Dh	PIR2	OSFIF	C2IF	C1IF	EEIF	—	—	—	—	0000 ----	42,205
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	86,205
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	86,205
10h	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	0000 0000	88,205
11h	TMR2(3)	Timer2 Module Register								0000 0000	91,205

Table 59. GPIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOx_CRL	CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]		CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x04	GPIOx_CRH	CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]		CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x08	GPIOx_IDR	Reserved																IDRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx_ODR	Reserved																ODRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_BSRR	BR[15:0]																BSR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	GPIOx_BRR	Reserved																BR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_LCKR	Reserved															LCKK	LCK[15:0]															
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 60. AFIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	AFIO_EVCR	Reserved																									EVOE	PORT[2:0]			PIN[3:0]		
	Reset value																										0	0	0	0	0	0	0
0x04	AFIO_MAPR low-, medium-, high- and XL-density devices	Reserved					SWJ_CFG[2]	SWJ_CFG[1]	SWJ_CFG[0]	Reserved			ADC2_ETRGREG_REMAP	ADC2_ETRGINJ_REMAP	ADC1_ETRGREG_REMAP	ADC1_ETRGINJ_REMAP	TIM5CH4_IREMAP	PD01_REMAP	CAN1_REMAP[1]	CAN1_REMAP[0]	TIM4_REMPAP	TIM3_REMPAP[1]	TIM3_REMPAP[0]	TIM2_REMPAP[1]	TIM2_REMPAP[0]	TIM1_REMPAP[1]	TIM1_REMPAP[0]	USART3_REMAP[1]	USART3_REMAP[0]	USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
	Reset value						0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	AFIO_MAPR connectivity line devices	Reserved	PTP_PPS_REMAP	TIM2ITR1_IREMAP	SPI3_REMAP	Reserved	SWJ_CFG[2]	SWJ_CFG[1]	SWJ_CFG[0]	MII_RMII_SEL	CAN2_REMAP	ETH_REMAP	Reserved				TIM5CH4_IREMAP	PD01_REMAP	CAN1_REMAP[1]	CAN1_REMAP[0]	TIM4_REMPAP	TIM3_REMPAP[1]	TIM3_REMPAP[0]	TIM2_REMPAP[1]	TIM2_REMPAP[0]	TIM1_REMPAP[1]	TIM1_REMPAP[0]	USART3_REMAP[1]	USART3_REMAP[0]	USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
	Reset value		0	0	0		0	0	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	AFIO_MAPR	Reserved					SWJ_CFG[2:0]			Reserved							TIM5CH4_IREMAP	PD01_REMAP	Reserved	TIM4_REMAP	TIM3_REMAP[1:0]		TIM2_REMAP[1:0]		TIM1_REMAP[1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
	Reset value						0	0	0								0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0

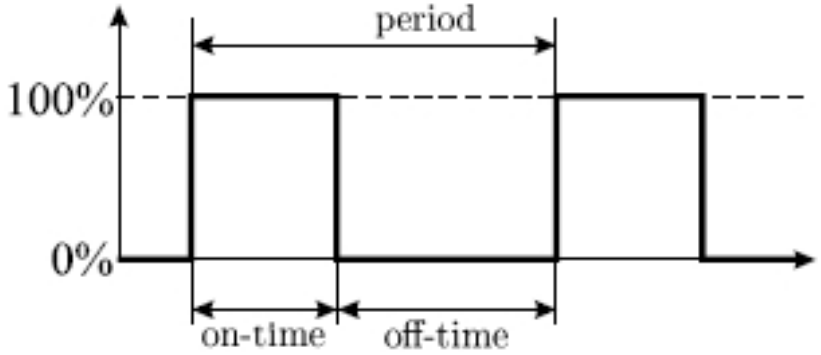
Pulse Width Modulation (PWM)

- **Pulse Width Modulation (PWM)** is a technique that conforms a signal width, generally pulses based on modulator signal information.
- The general purpose of PWM is to control power delivery, especially to inertial electrical devices.
- The on-off behavior changes the average power of signal.
- Output signal alternates between on and off within a specified period.
- If signal toggles between on and off quicker than the load, then the load is not affected by the toggling.

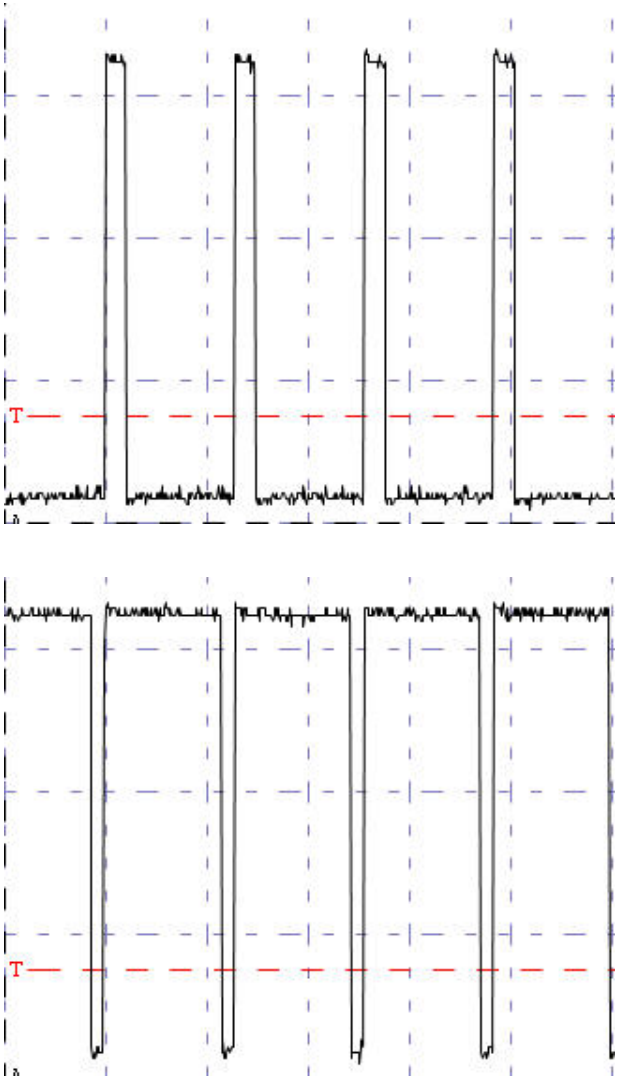
Pulse Width Modulation (PWM)

- Duty Cycle
  - **D = on-time/period**

- V_{LOW} is often zero.



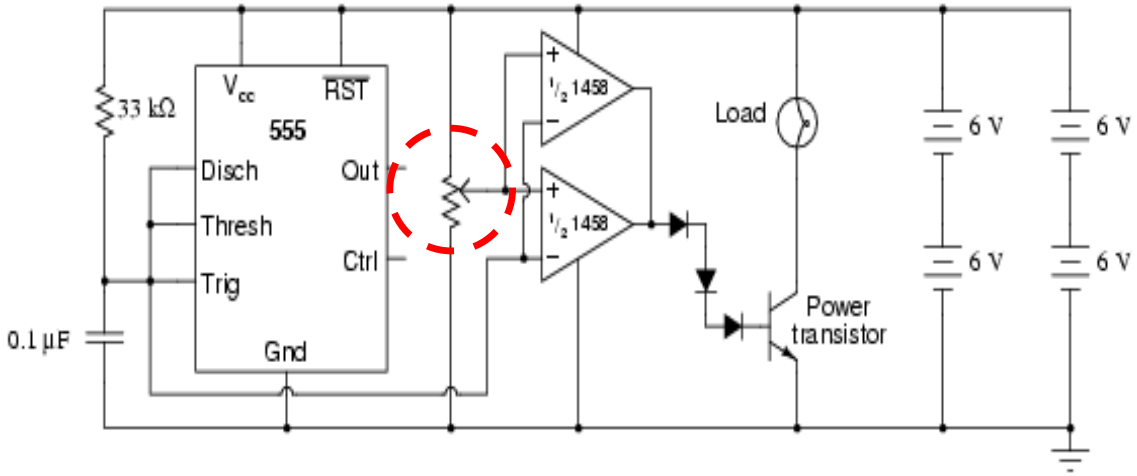
$$V_{AVG} = DV_{HI} + (1 - D)V_{LOW}$$

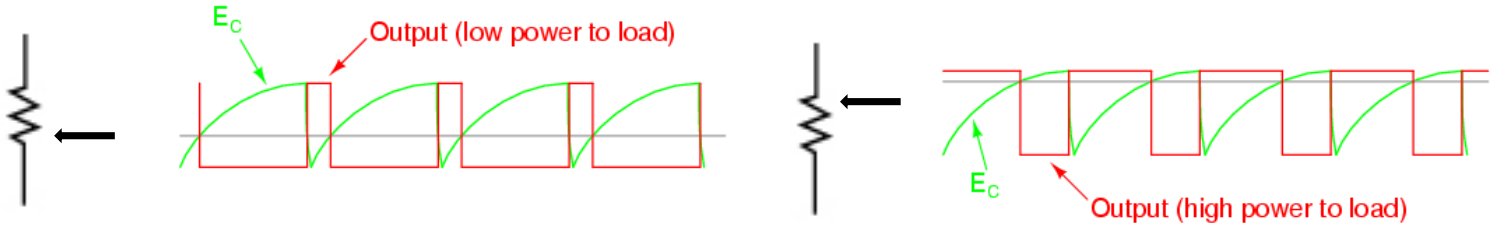


anhpham@hcmut.edu.vn

PWM with 555 Timer

- Potentiometer is used to adjust the duty cycle





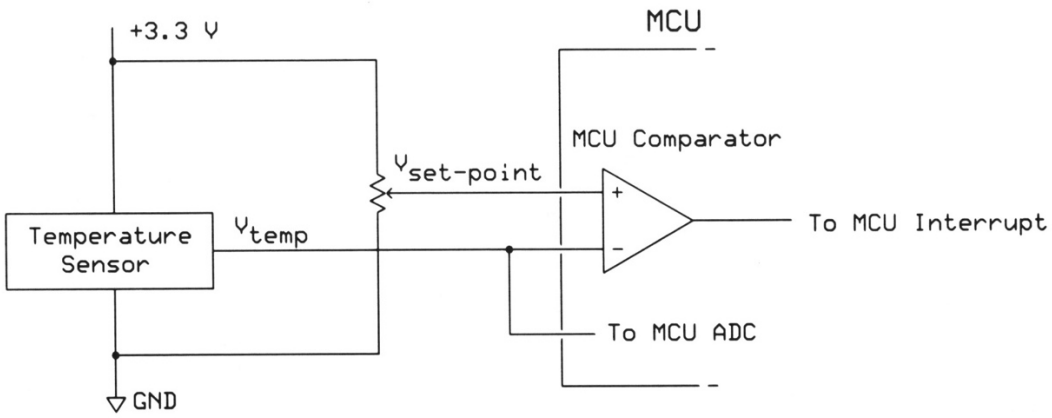
Counters and Timers

- Operate for a specific period or create a delay
- Count external events, count up or down
- Count clock “ticks” between the same or different events
- Choose from various clock sources

anhpham@hcmut.edu.vn

Analog Comparator

- Compare two voltages and then ...
  - Cause a bit to change state
  - Generate an interrupt
  - Wake an MCU from a sleep state

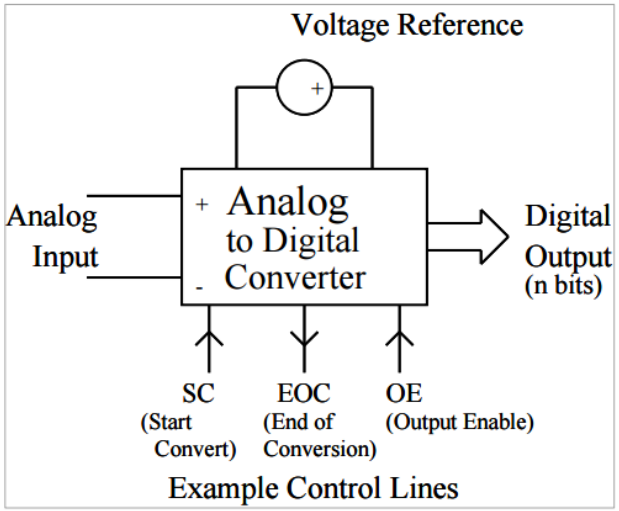


Analog-To-Digital Conversion (ADC)

- Embedded system applications are often required to interface with analog signals.
- They must be able to convert **input analog signals**, for example from microphone or temperature sensor, **to digital data.**
- They must also be able to convert digital signals to analog form, for example if driving a loudspeaker or dc motor
- We will first consider conversion from analog-to-digital, before later looking at digital-to-analog conversion

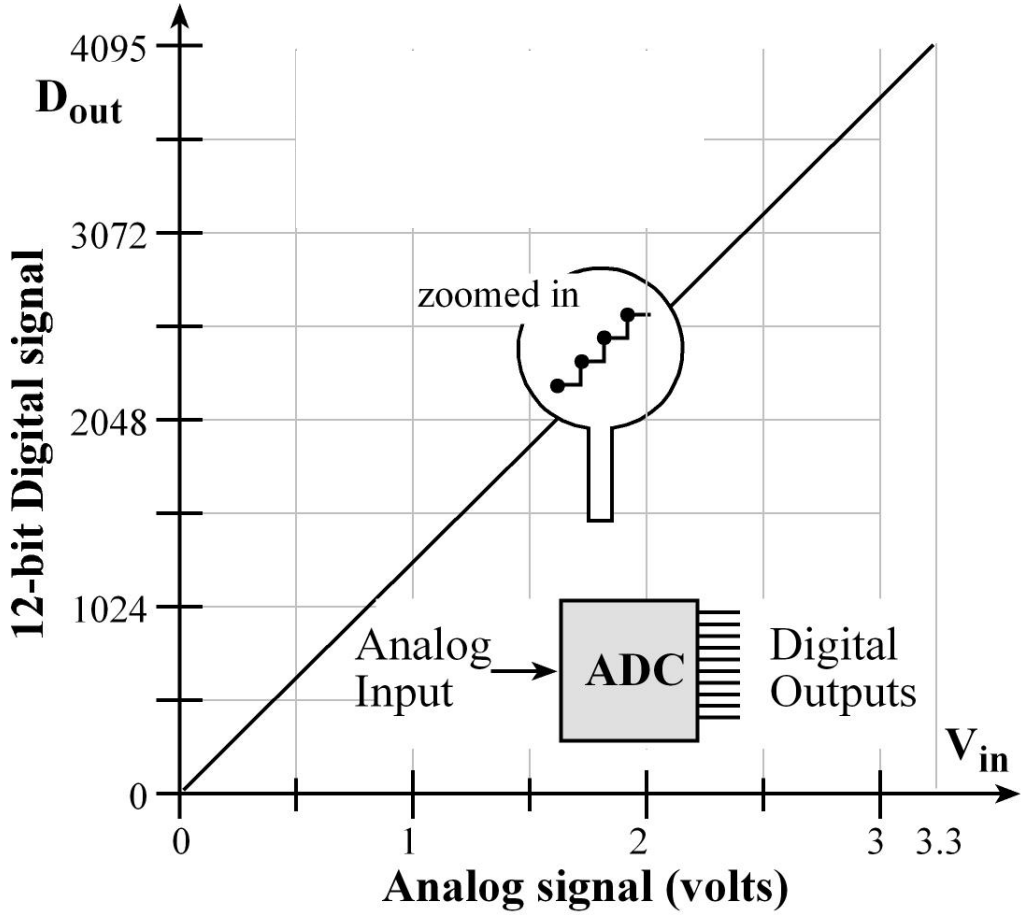
Concepts of Analog-To-Digital Conversion

- An analog-to-digital convertor (ADC) is an electronic circuit whose digital output is proportional to its analog input.
- Effectively it "measures" the input voltage, and gives a binary output number proportional to its size.
- The input range of the ADC is usually determined by the value of a **voltage reference**.



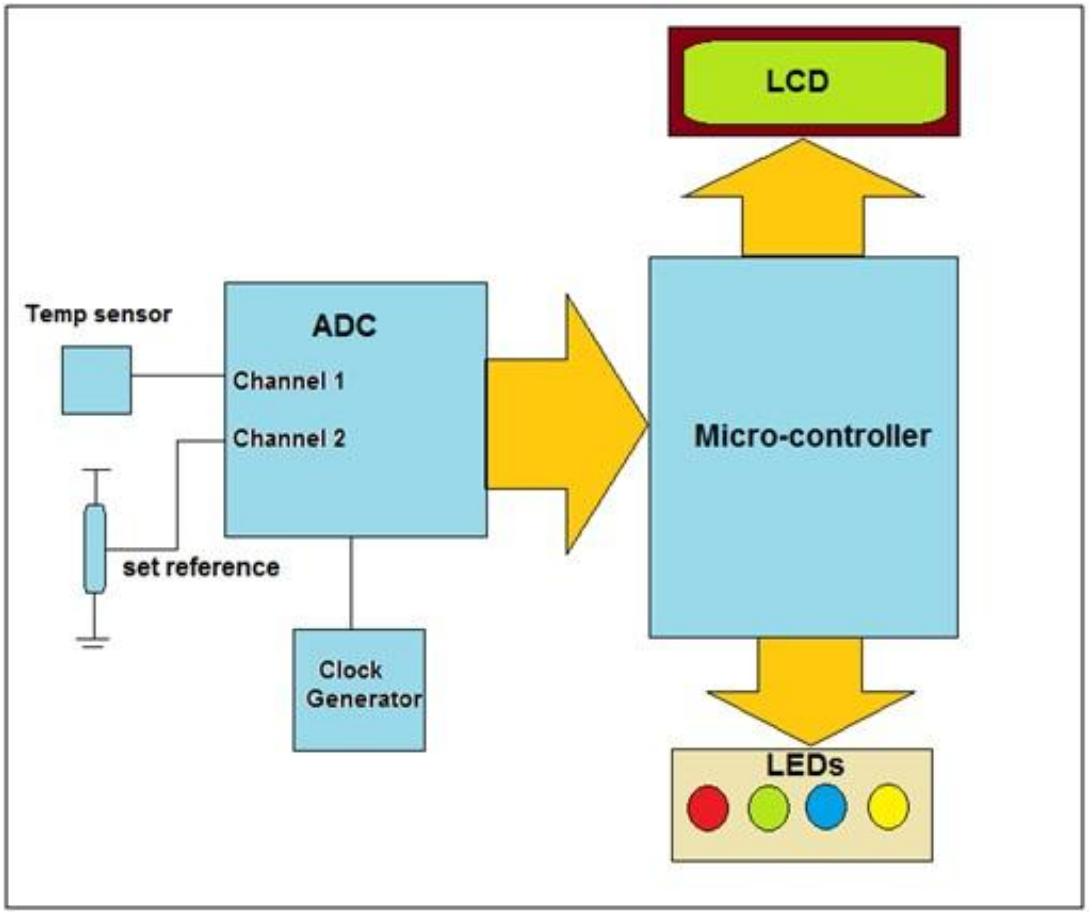
Analog-To-Digital Conversion

- Example



Data Acquisition System

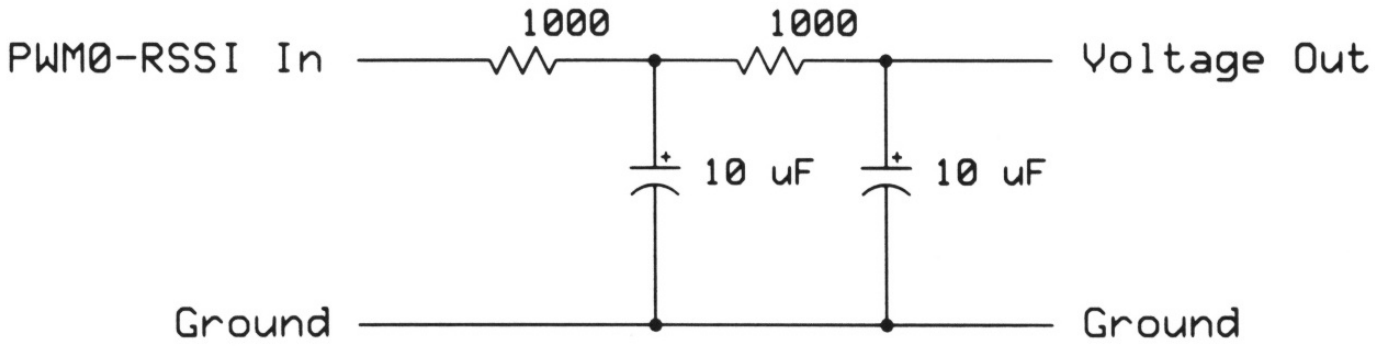
- Example



anhpham@hcmut.edu.vn

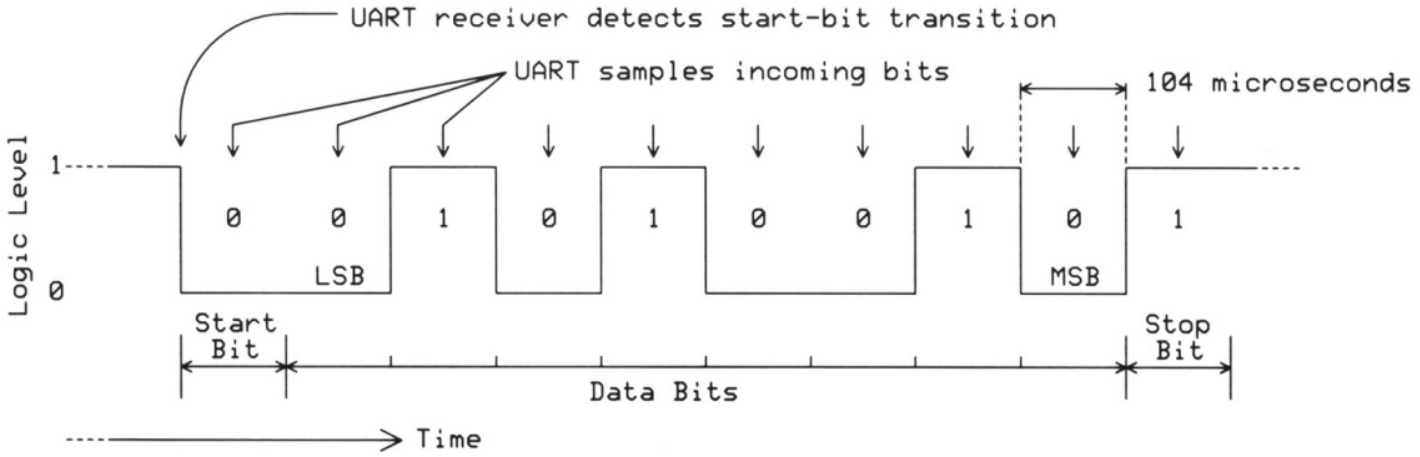
Digital-to-Analog Converter (DAC)

- Unipolar output, might require external offset
- High-impedance output, could require buffering
- 10- and 12-bit DACs common on MCUs
- Filter a PWM output to get an analog voltage

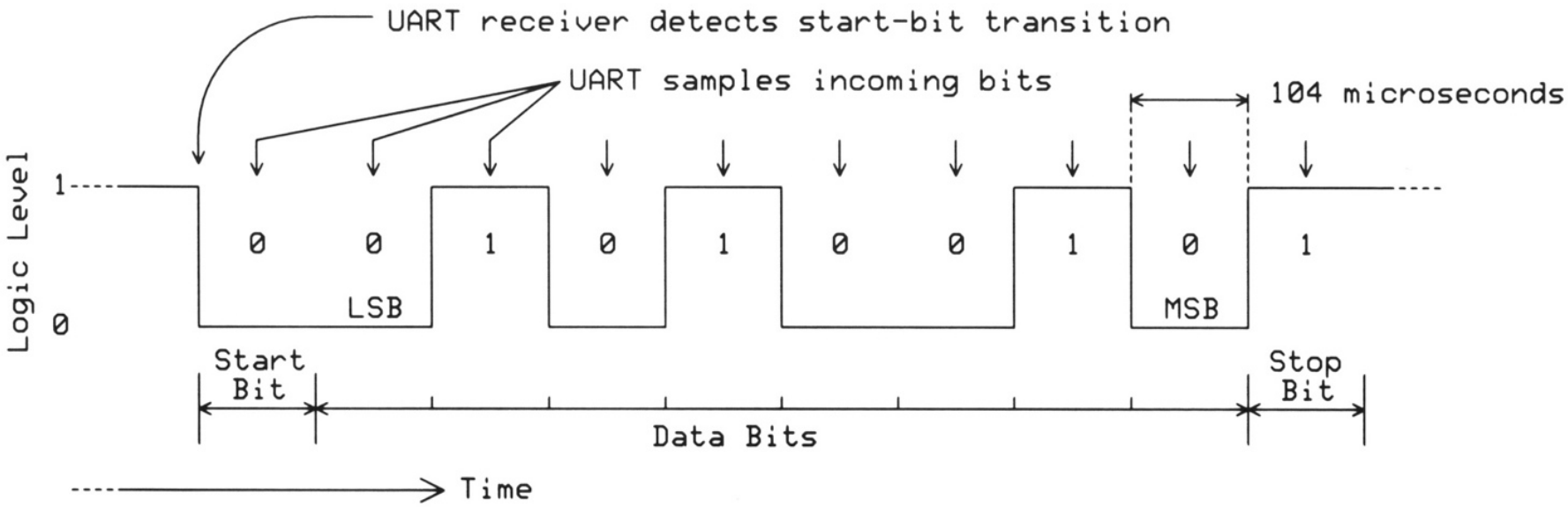


UART Communications

- Universal Asynchronous Receiver-Transmitter (UART)
  - Serial communications
  - Self-timing operations
  - Usually 8-bit transmissions at standard rates
  - Common on most MCUs



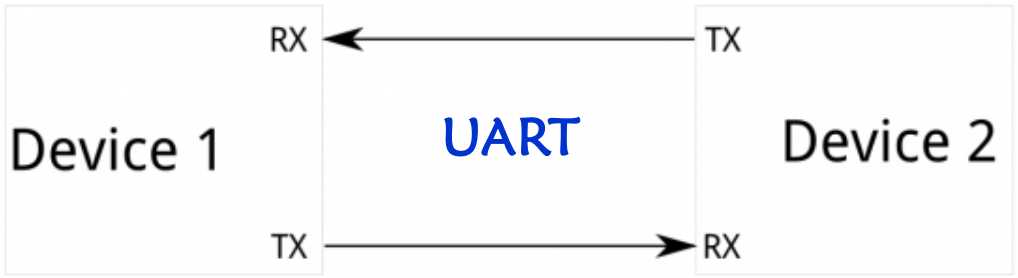
UART Communications Timing



Communications at 9600 bits/sec

What's Wrong with Serial Ports?

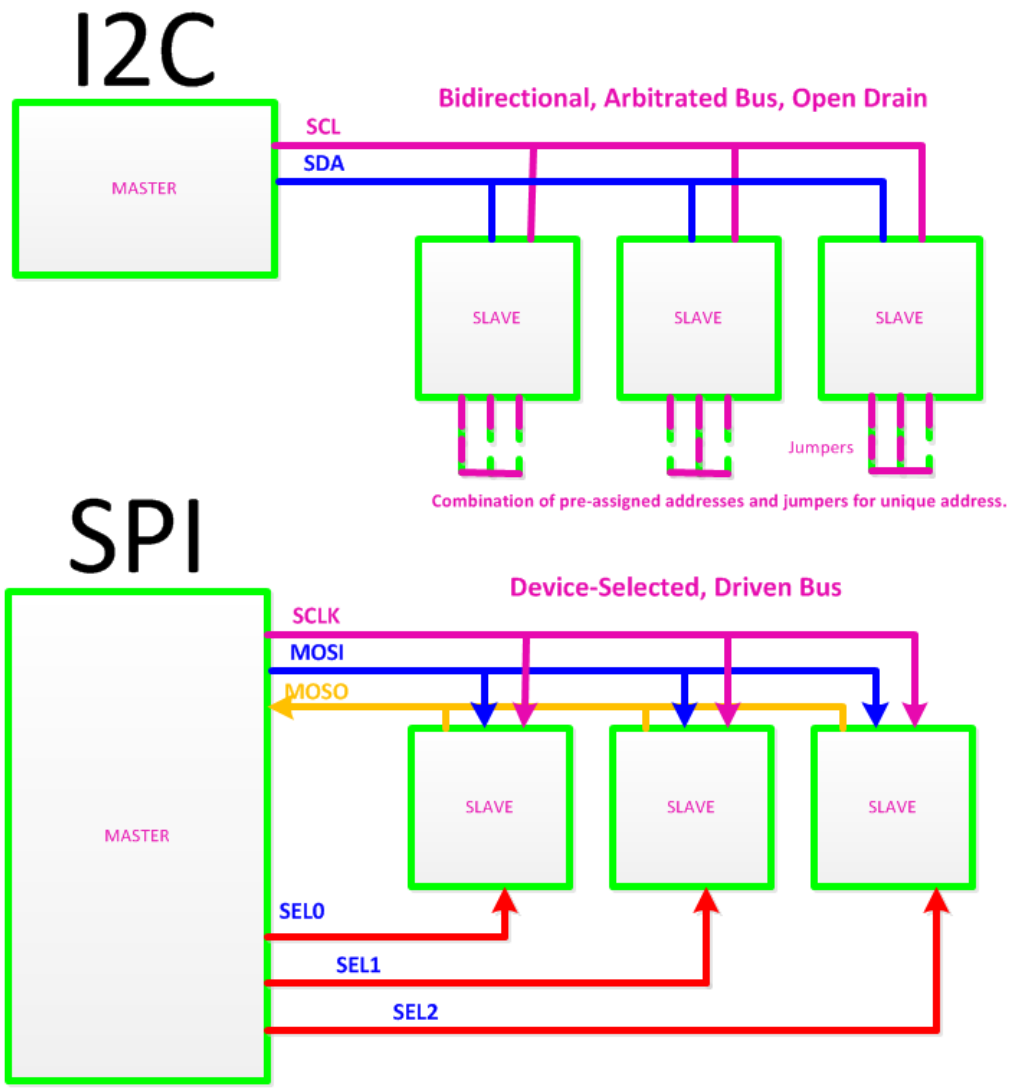
- A common serial port, the kind with TX and RX lines, is called "**asynchronous**" (not synchronous) because there is no control over when data is sent or any guarantee that both sides are running at precisely the same rate.
  - Since computers normally rely on everything being synchronized to a single "clock", this can be a problem when two systems with slightly different clocks try to communicate with each other.



→ Synchronous Serial Communication

anhpham@hcmut.edu.vn

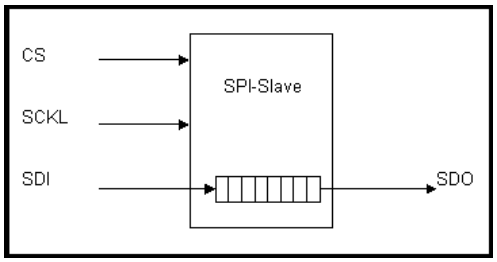
SPI vs. I2C

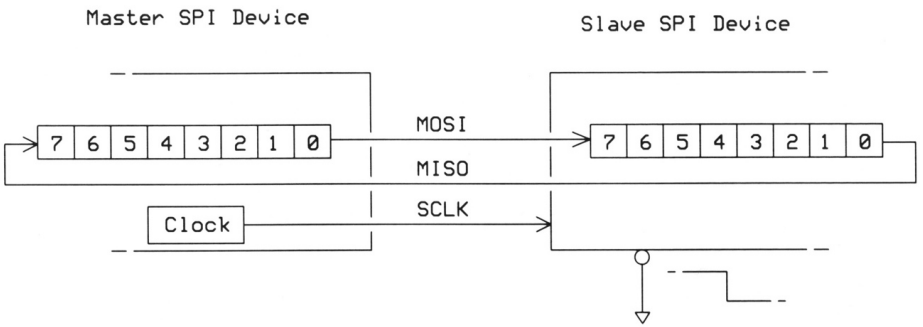


anhpham@hcmut.edu.vn

Serial Peripheral Interface (SPI)

- Developed by Motorola
  - Also known as MicroWire (National Semiconductor), QSPI (Queued), MicrowirePlus.
  - Synchronous Serial Communication.
- Primarily used for **serial communication** between **a host processor and peripherals**.
- Can also connect 2 processors via SPI
- SPI works in a master-slave configuration with the master being the host microcontroller for example and the slave being the peripheral





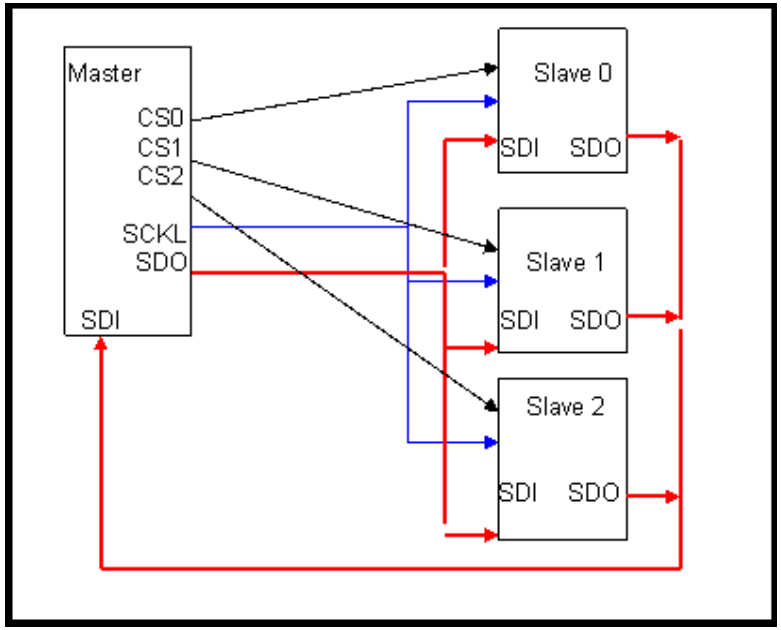
SPI Operation

- For SPI, there are Serial Clocks (SCLK), Chip Select lines (CS), Serial Data In (SDI) and Serial Data Out(SDO).
- There is only one master, there number of slaves depends on the number of chip select lines of the master.
- Synchronous operation, latch on rising or falling edge of clock, SDI on rising edge, SDO on falling edge.
- Master sends out clocks and chip selects. Activates the slaves it wants to communicate with.
- Operates in **1 to 2 MHz** range.

SPI – Master Slave Setup

- In this setup, there are 3 slave devices. The SDO lines are tied together to the SDI line of the master.
- The master determines which chip it is talking to by the CS lines. For the slaves that are not being talked to, the data output goes to a Hi Z state

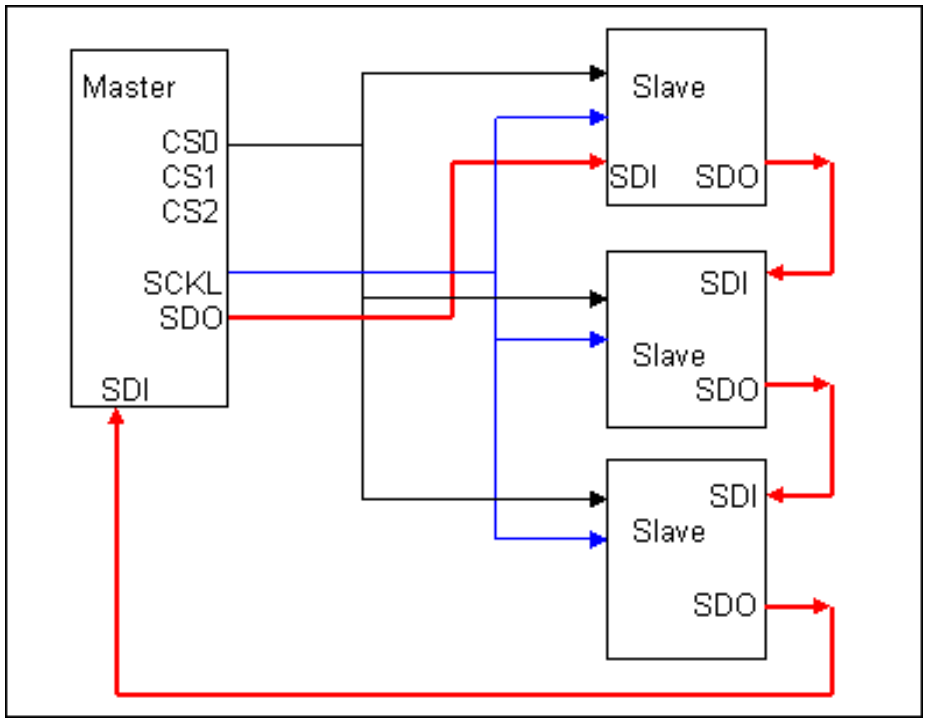
Multiple Independent Slave Configuration



SPI – Master Slave Setup

- In this example, each slave is cascaded so that the output of one slave is the input of another. When cascading, they are treated as one slave and connecting to the same chip select.

Multiple slave cascaded configuration



anhpham@hcmut.edu.vn

SPI Peripherals Types

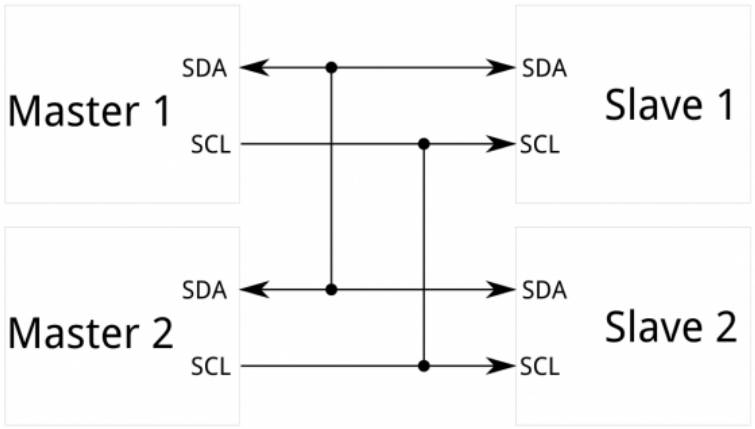
- **Types**
  - Converters (ADC, DAC)
  - Memories (EEPROM, RAM's,Flash)
  - Sensors (Temperature, Humidity, Pressure)
  - Real Time Clocks
  - Misc-Potentiometers, LCD controllers, UART's, USB controller, CAN controller, amplifiers.

- **Vendors that make these peripherals**
  - Atmel –EEPROM, Dig. POT's
  - Infineon- Pressure Sensors, Humidity Sensors
  - Maxim- ADC, DAC, UART,
  - TI- DSP's, ADC, DAC
  - National Semiconductor-Temperature Sensors, LCD/USB controllers

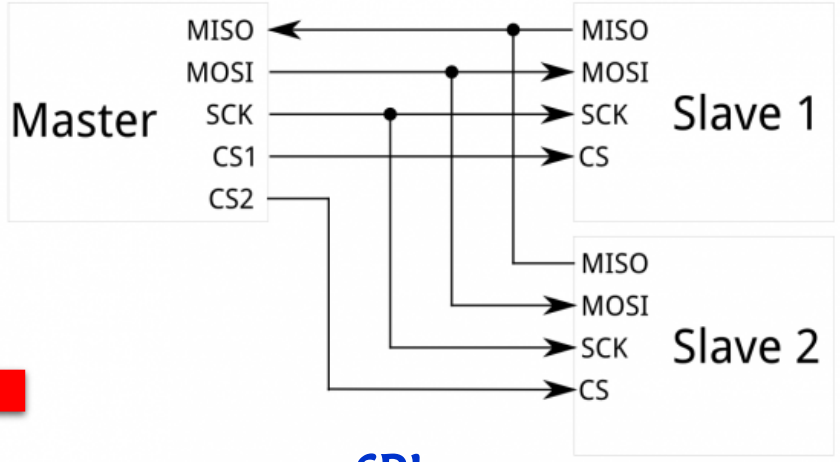
anhpham@hcmut.edu.vn

What's Wrong with SPI?

- The most obvious drawback of SPI is **the number of pins** required. Connecting a single master to a single slave with an SPI bus requires four lines; each additional slave requires one additional chip select I/O pin on the master.
- SPI only **allows one master on the bus**, but it does support an arbitrary number of slaves (subject only to the drive capability of the devices connected to the bus and the number of chip select pins available).



I2C



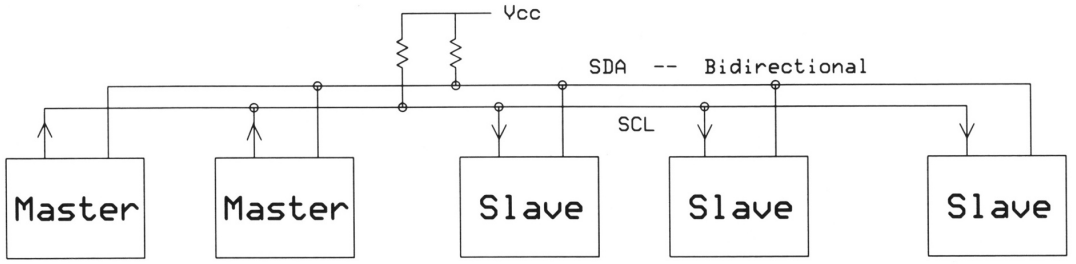
SPI

Inter-Integrated Circuit (I2C)

- I²C requires a mere **two wires**, like asynchronous serial, but those two wires can support up to **1008 slave devices**. Also, unlike SPI, I²C can support a multi-master system, allowing more than one master to communicate with all devices on the bus.

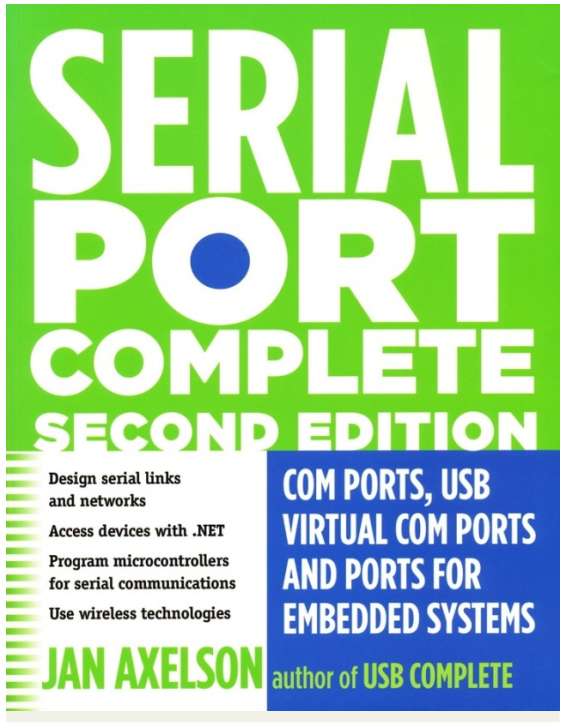
- Data rates fall between asynchronous serial and SPI.
  - Most I²C devices can communicate at **100kHz or 400kHz**.

- There is some overhead with I²C.
  - For every 8 bits of data to be sent, one extra bit of meta data (the "ACK/NACK" bit, which we'll discuss later) must be transmitted.



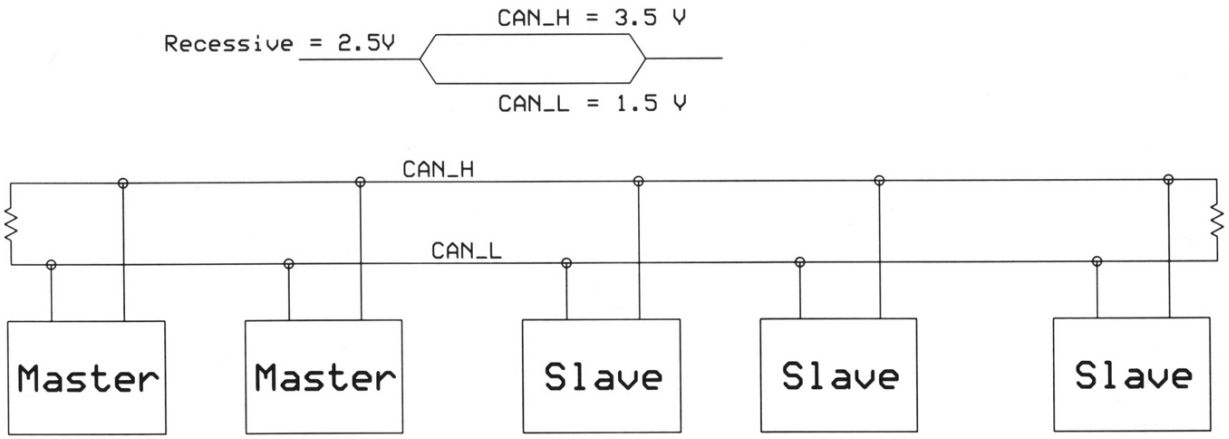
Serial Communications

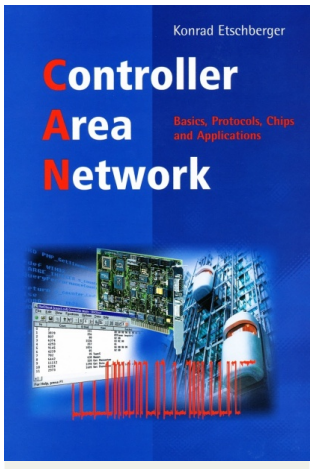
- Further Reading: “Serial Port Complete,” 2nd ed., by Jan Axelson, Lakeview Research, 2007. ISBN: 978-1-931448-06-2.

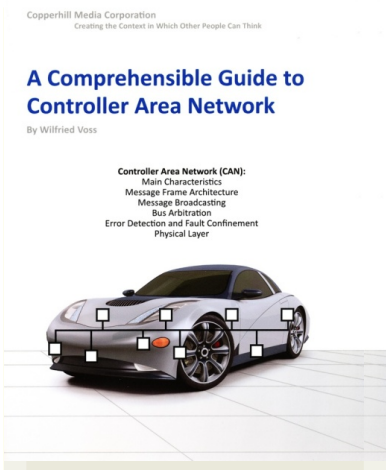


Controller-Area Network (CAN)

- A standard for vehicle equipment
- Uses an ISO-type "stack"
- 2-wire differential bus, no common ground needed
- Uses standardized packets of information







"Controller Area Network," by Konrad Etschberger," IXXAT Automation, 2001. ISB: 978-3-00-007376-0.
"A Comprehensive Guide to Controller Area Network," by Wilfried Voss, Copper Hill Media, 2008. ISBN: 978-0976511601.

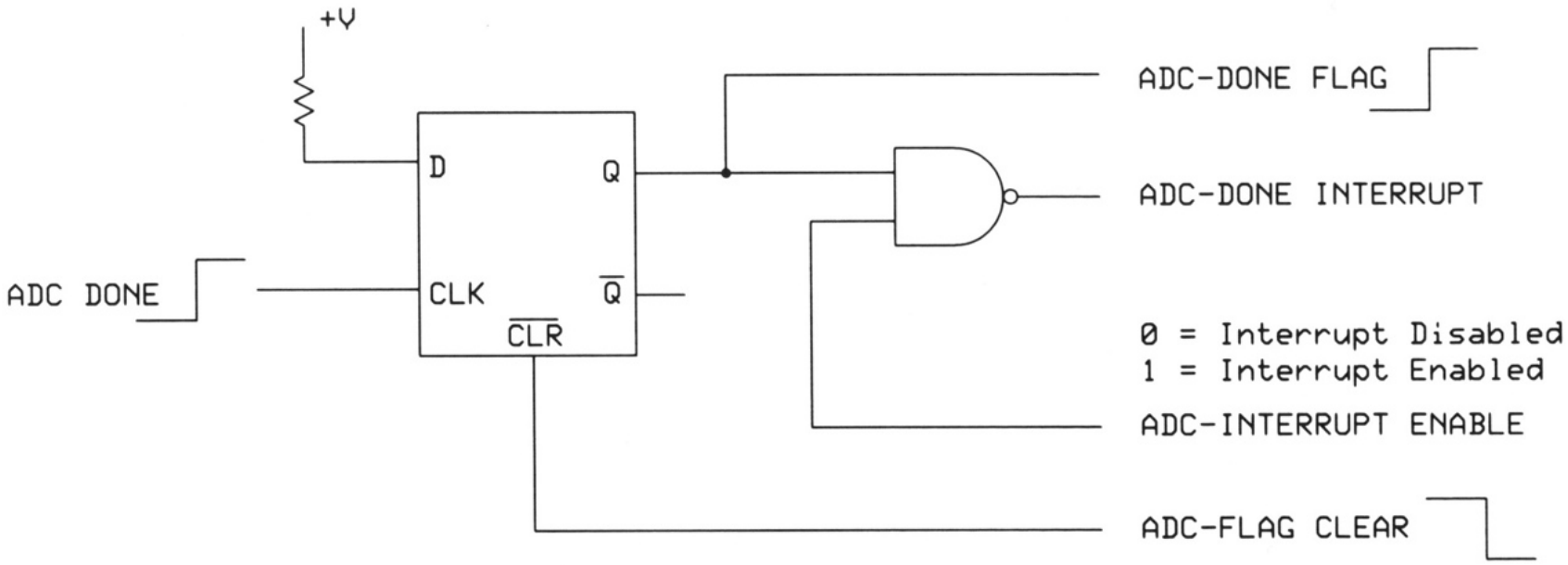
Ethernet and USB

- Governed by standards
- Require a software “stack”
  - Purchase, license, or create one yourself
  - MCU vendors might have stacks
- Some MCUs include everything except the physical interface (PHY)
- Can demand considerable memory
- Start with a development kit or reference design

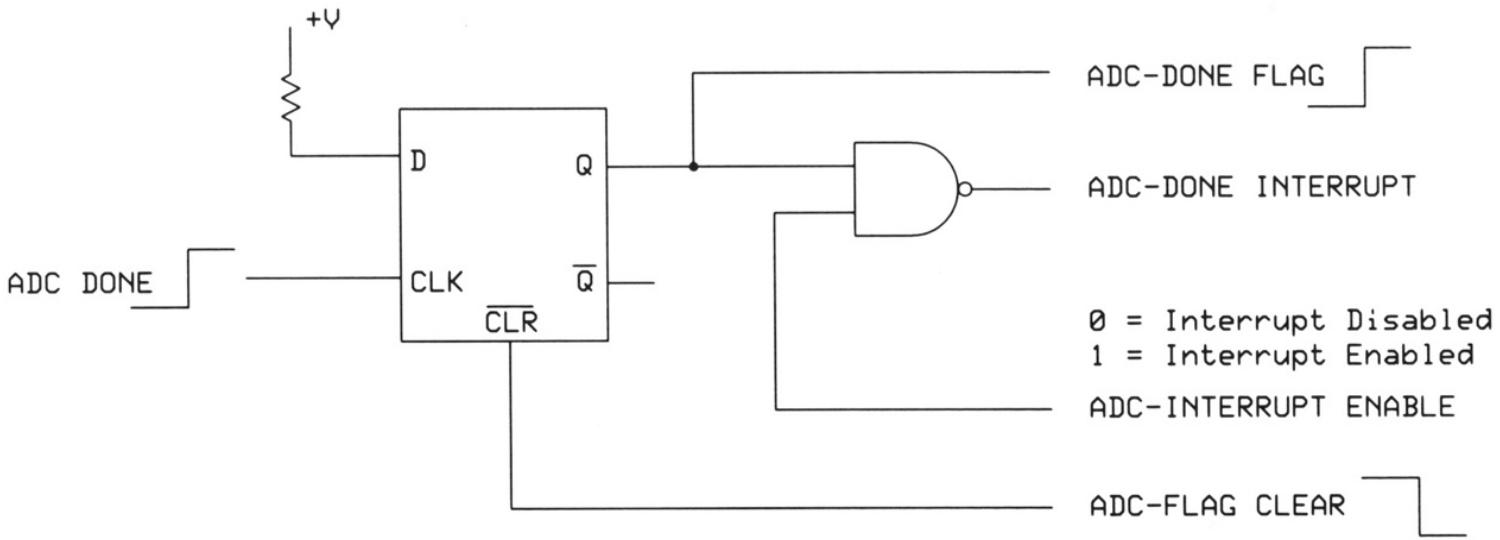
Interrupts

- Cause immediate action
- Internal and external hardware and software sources
- Two types of action -- one or many vectors
- Can present debug challenges

An ADC Interrupt

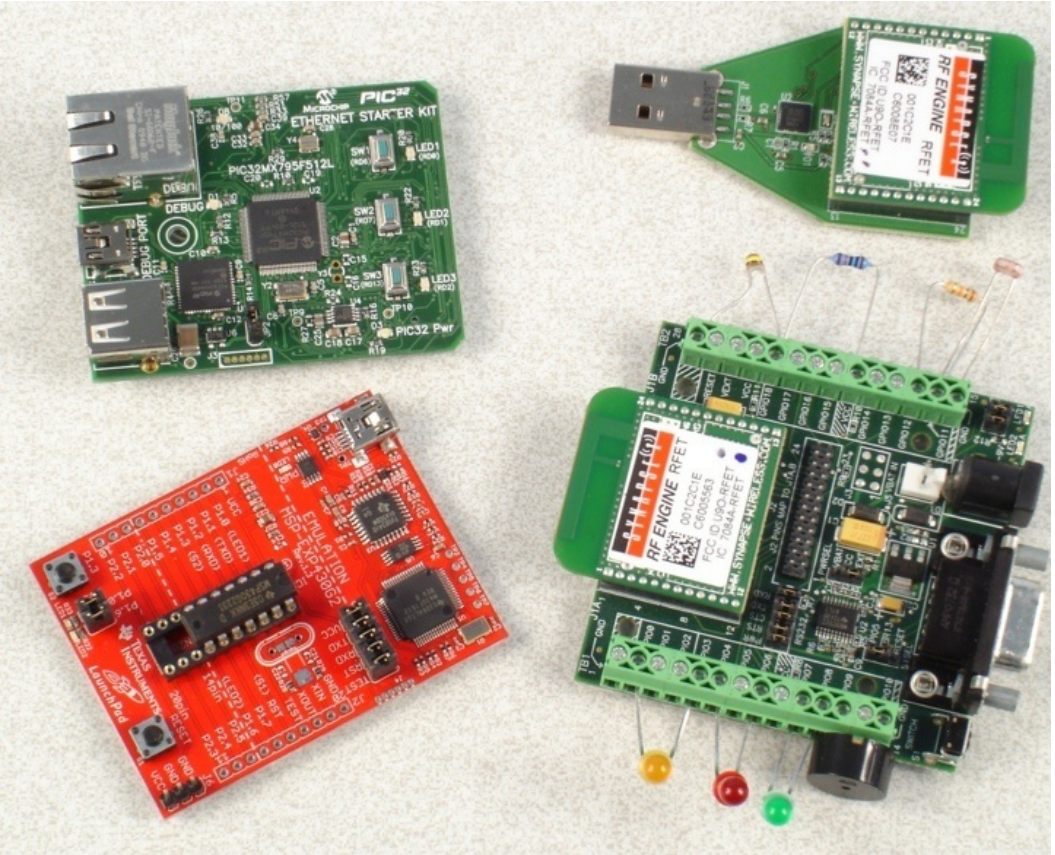


An ADC Interrupt



anhpham@hcmut.edu.vn

How Do I Get a Quick Start?



anhpham@hcmut.edu.vn

How Do I Get Started?

- CO3009 -

- Software Architectures -

Embedded Software Architectures

- Round Robin
- Round Robin with Interrupts
- Function Queue Scheduling
- Real-time Operating System

anhpham@hcmut.edu.vn

Round Robin

- No interrupts
- Main loop checks each of the I/O device in turn and services any that need service
- No priorities

```
void main(void){
        while (TRUE){
                if (!! I/O Device A needs service) {
                        !! Take care of I/O Device A
                        !! Handle data to or from I/O Device A
                }
                if(!! I/O Device B needs service) {
                        !! Take care of I/O Device B
                        !! Handle data to or from I/O Device B
                }
                etc.
                etc.
                if(!! I/O Device Z needs service) {
                        !! Take care of I/O Device Z
                        !! Handle data to or from I/O Device Z
                }

        }
}
```


Round Robin – Digital Multimeter

```
void vDigitalMultiMeterMain(void){
        enum {OHMS_1, OHMS_10, …, VOLTS_100} eSwitchPosition;
        while (TRUE){
                eSwitchPosition  = !! Read the position of the switch;
                switch (eSwitchPosition) {
                        case OHMS_1:
                                !! Read hardware to mearsure ohms
                                !! Format result
                        break;
                        case OHMS_10:
                                !! Read hardware to mearsure ohms
                                !! Format result
                        break;
                        .
                        .
                        .
                        case VOLTS_100 :
                                !! Read hardware to mearsure ohms
                                !! Format result
                        break;
                }
                !! Write result to display
        }
}
```

anhpham@hcmut.edu.vn

Round Robin

- If any one device needs response in less time that it takes the MCU to get around the main loop in the worst-case scenario, then the system won't work.

- For example, if device Z can wait no longer than 7 milliseconds for service and if the pieces of code that service devices A and B take 5 milliseconds each, then the processor won't always get to device Z quickly enough

Round Robin

- Even if none of the required response times are absolute deadlines, the system may not work well if there is any lengthy processing to do.

- For example, if one of the cases were to take, say, 3 seconds, then the system's response to the rotary switch may get as bad as 3 seconds. This may not quite meet the definition of "not working", but it would probably not be a system that anyone would be proud to ship

Round Robin

- This architecture is fragile.
- Even if you manage to tune it up so that the MCU gets around the loop quickly enough so satisfy all the requirements, a single additional device or requirement may break everything.
- This architecture is probably suitable only for very simple devices.

Round Robin with Interrupts

- Interrupt routines deal with the very urgent needs of the hardware and the set flags.
- The main loop polls the flags and does any follow-up processing required by the interrupts.

Round Robin with Interrupts

```
BOOL fDeviceA = FALSE;
BOOL fDeviceB = FALSE;
…
BOOL fDeviceZ = FALSE;
void INTERRUPT vHandleDeviceA (void){
        !! Take care of I/O Device A
        fDeviceA = TRUE
}
void INTERRUPT vHandleDeviceB (void){
        !! Take care of I/O Device B
        fDeviceB = TRUE
}
…
void INTERRUPT vHandleDeviceZ (void){
        !! Take care of I/O Device Z
        fDeviceZ = TRUE
}
```

```
void main (void) {
        while(TRUE){
            if (fDeviceA) {
                    fDeviceA = FALSE;
                    !! Handle data to or from device A
            }
            if (fDeviceB) {
                    fDeviceB = FALSE;
                    !! Handle data to or from device B
            }
            ….
            if (fDeviceZ) {
                    fDeviceZ = FALSE;
                    !! Handle data to or from device Z
            }
        }
}
```


Round Robin with Interrupts

- This architecture gives you a bit more control over priorities.
- The interrupt routines can get good response, because the hardware interrupt signal causes the microcontroller to stop whatever it is doing in the main function and execute the interrupt routine instead.
- All of the processing that you put into the interrupt routines has a higher priority than the task code in the main routine.
- Since you can usually assign priorities to the various interrupts in your system, you can control the priorities among the interrupt routines as well.

Function-Queue Scheduling

- The interrupt routines add function pointers to a queue of function pointers for the main function to call.

- The main routine just reads pointers from the queue and calls the functions.

Function-Queue Scheduling

```
!! Queue of function pointers;
void interrupt vHandleDeviceA (void){
        !! Take care of I/O Device A
        !! Put function_A on queue of function poitners
}
void interrupt vHandleDeviceB (void){
        !! Take care of I/O Device B
        !! Put function_B on queue of function poitners
}
void function_A (void){
       !! Handle actions required by device A
}
void function_B (void){
      !! Handle actions required by device B
}
```

```
void main (void){
     while(TRUE){
          while (!! Queue of function pointer is empty);
          !! Call first function on queue
     }
}
```


Function-Queue Scheduling

- No rule says main has to call the functions in the order that the interrupt routines occurred.
- It can call them based on any priority scheme that suits your purposes.

Real-Time Operating Systems

- The interrupt routines take care of the most urgent operations.
- They signal that there is work for the task code to do.

```
void interrupt vHandleDeviceA (void) {
        !! Take care of I/O Device A
        !! Set signal X
}
void interrupt vHandleDeviceB (void) {
        !! Take care of I/O Device B
        !! Set signal B
}
void Task1 (void){
        while(TRUE){
                !! Wait for signal X
                !! Handle data to or from I/O device A
        }
}
void Task2 (void){
        while(TRUE){
                !! Wait for signal Y
                !! Handle data to or from I/O device B
        }
}
```


Real-Time Operating Systems (RTOS)

- The differences between RTOS and the previous architectures are that:
  - The necessary signaling between the interrupt routines and the task code is handled by the RTOS.
  - No loop in our code decides what needs to be done next. Code inside the RTOS decides which of the task code function should run.
  - The RTOS knows about the various task-code subroutines and will run whichever of them is more urgent at any given time.
  - The RTOS can suspend one task code subroutines in middle of its processing in order to run another.

Summary

- Response requirements most often drive the choice of architecture.
- Generally, you will be better off choosing a simpler architecture.
- Hybrid architecture can make sense for some systems.

anhpham@hcmut.edu.vn

Summary

	Priorities Available	Worst Response Time for Task code	Stability of Response when the code changes	Simplicity
Round robin	None	Sum of all task code	Poor	Very simple
Round robin with interrupts	Interrupt routines in priority order, then all task code at the same priority	Total of execution time for all task code (plus execution time for interrupt routines)	Good for interrupt routines, poor for task code.	Must deal with data shared between interrupt routings and task code.
Function-Queue scheduling	Interrupt routines in priority order, then task code in priority order	Execution time for the longest function (plus execution timer for interrupt routines)	Relatively good	Must deal with shared data and must write function queue code
RTOS	Interrupt routines in priority order, then task code in priority order	Zero (plus execution time for interrupt routines)	Very good	Most complex (although much of the complexity is inside the OS itself.

- CO3009 -

- Embedded C Programming -

A Simple C program

```
void FSM();
void main(void){
    // initialize the device
    System_Initialization();
    while (1) {
            FSM();
   }
}
/**
  * @brief  This function handles TIM interrupt request.
  * @param  None
  * @retval None   */
void TIM3_IRQHandler(void){
      HAL_TIM_IRQHandler(&TimHandle);
}
```

```
/**
  * @brief  Period elapsed callback in non blocking mode
  * @param  htim : TIM handle
  * @retval None  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
*htim) {
      SCH_Update();
}
```


A main() Function

- System initialization
  - Oscillator
  - Input/Output
  - Special peripherals/modules: Timers, CCP, ADC
  - Disable/Enable interrupt

- A super loop while(1)

- One or more finite state machine (FSM)

System Initialization

```
35 enum InitState initState = HAL_INIT;
36
37 void System_Initialization(void)
38 {
39     while(initState != MAX_INIT_STATE){
40         switch (initState) {
41         case HAL_INIT:
42             HAL_Init();
43             break;
44         case SYSTEM_CLOCK_INIT:
45             SystemClock_Config();
46             break;
47         case UART_INIT:
48             UART3_Init();
49             UART1_Init();
50             DEBUG_INIT(UART3_SendToHost((uint8_t*)"UART_INIT - Done \r\n"));
51             break;
52         case GPIO_INIT:
53             MX_GPIO_Init();
54             DEBUG_INIT(UART3_SendToHost((uint8_t*)"GPIO_INIT - ADC_DMA_Init - Done \r\n"));
55             break;
56         case LED_DISPLAY_INIT:
57             Led_Display_Init();
58             DEBUG_INIT(UART3_SendToHost((uint8_t*)"LED_DISPLAY_INIT - Done \r\n"));
59             break;
60         case RELAY_INIT:
61             Relay_Init();
62             DEBUG_INIT(UART3_SendToHost((uint8_t*)"RELAY_INIT - Done \r\n"));
63             break;
64         case FLASH_INIT:
65             DEBUG_INIT(UART3_SendToHost((uint8_t*)"FLASH_INIT - Done \r\n"));
66             break;
67         case TIMER_INIT:
68             Timer_Init();
69             DEBUG_INIT(UART3_SendToHost((uint8_t*)"TIMER_INIT - Done \r\n"));
70             break;
71         case SPI_INIT:
72             SPI1_Init();
73             SPI2_Init();
74             DEBUG_INIT(UART3_SendToHost((uint8_t*)"SPI_INIT - Done \r\n"));
75             break;
76         case SPI_25LCXXX_INIT:
77             Eeprom_Initialize();
78             DEBUG_INIT(UART3_SendToHost((uint8_t*)"SPI_25LCXXX_INIT - Done \r\n"));
79             break;
80         case I2C_INIT:
81             I2C_Init();
82             DEBUG_INIT(UART3_SendToHost((uint8_t*)"I2C_Init \r\n"));
83             PCF_Init();
```

anhpham@hcmut.edu.vn

HAL_Init

- This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it performs the following:
  - Configure the Flash prefetch.
  - Configures the SysTick to generate an interrupt each 1 millisecond, which is clocked by the HSI (at this stage, the clock is not yet configured and thus the system is running from the internal HSI at 16 MHz).
  - Set NVIC Group Priority to 4.
  - Calls the **HAL_MspInit()** callback function defined in user file "**stm32f1xx_hal_msp.c**" to do the global low level hardware initialization

  - **@note** SysTick is used as time base for the **HAL_Delay()** function, the application need to ensure that the SysTick time base is always set to **01 millisecond** to have correct HAL operation.
  - **@retval** HAL status

HAL_Init

```
142 HAL_StatusTypeDef HAL_Init(void)
143 {
144   /* Configure Flash prefetch */
145 #if (PREFETCH_ENABLE != 0)
146 #if defined(STM32F101x6) || defined(STM32F101xB) || defined(STM32F101xE) || defined(STM32F101xG) || \
147     defined(STM32F102x6) || defined(STM32F102xB) || \
148     defined(STM32F103x6) || defined(STM32F103xB) || defined(STM32F103xE) || defined(STM32F103xG) || \
149     defined(STM32F105xC) || defined(STM32F107xC)
150
151   /* Prefetch buffer is not available on value line devices */
152   __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
153 #endif
154 #endif /* PREFETCH_ENABLE */
155
156   /* Set Interrupt Group Priority */
157   HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
158
159   /* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) */
160   HAL_InitTick(TICK_INT_PRIORITY);
161
162   /* Init the low level hardware */
163   HAL_MspInit();
164
165   /* Return function status */
166   return HAL_OK;
167 }
```


SystemClock_Config

```c
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
    /** Initializes the CPU, AHB and APB busses clocks
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
    RCC_OscInitStruct.HSEState        = RCC_HSE_OFF;
    RCC_OscInitStruct.LSEState        = RCC_LSE_OFF;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;

    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK){
        Error_Handler();
    }
    /** Initializes the CPU, AHB and APB busses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK){
        Error_Handler();
    }
    PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
    PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
    if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK){
        Error_Handler();
    }
}
```


Clocks

Three different clock sources can be used to drive the system clock (SYSCLK):

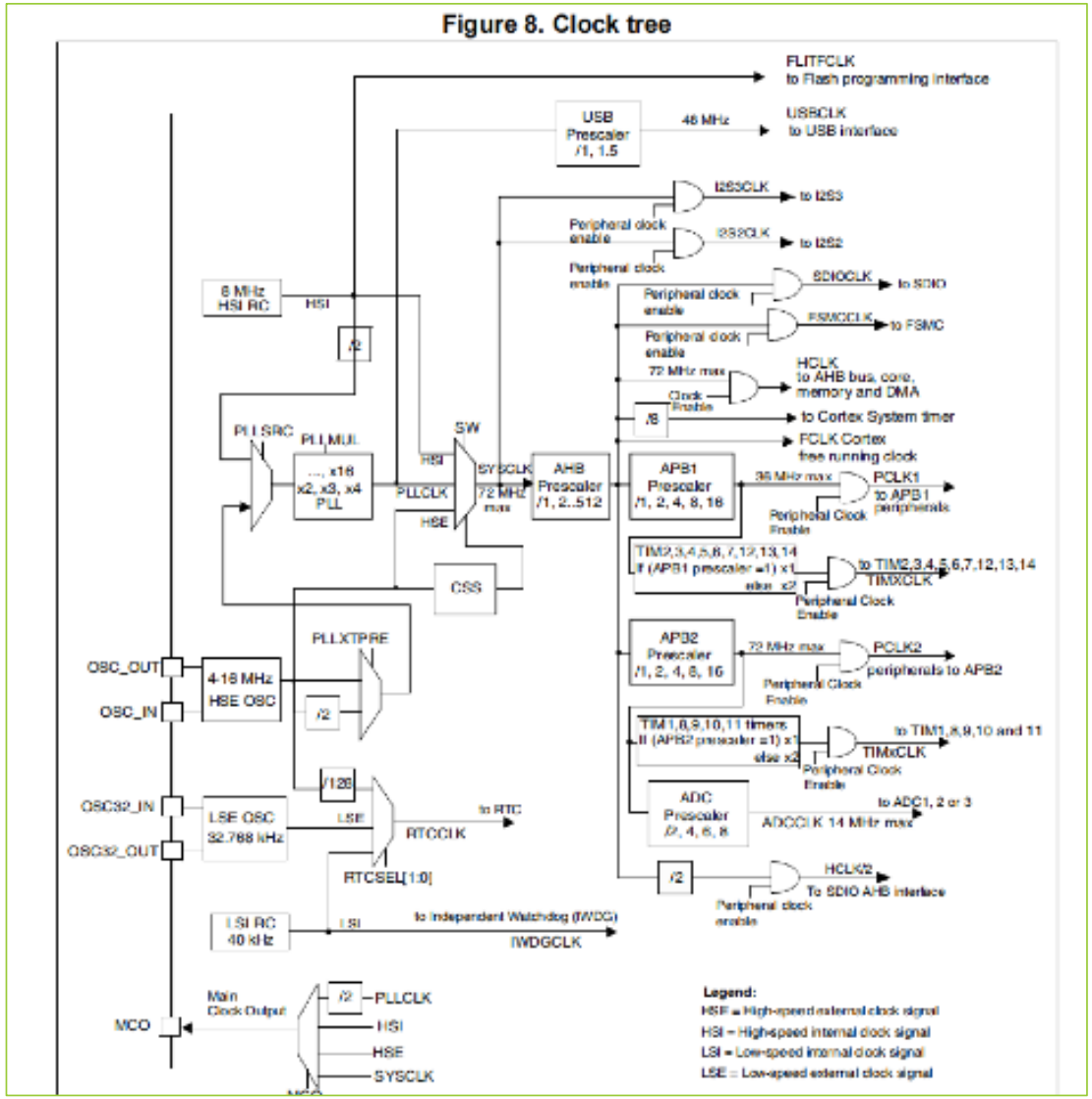
- HIS (High-Speed Internal) oscillator clock
- HSE (High-Speed External) oscillator clock
- PLL (Phase-Locked Loop) clock

The devices have the following two secondary clock sources:

- 40 kHz low speed internal RC (LSI RC), which drives the independent watchdog and optionally the RTC used for Auto-wakeup from Stop/Standby mode.
- 32.768 kHz low speed external crystal (LSE crystal), which optionally drives the real-time clock (RTCCLK)

Each clock source can be switched on or off independently when it is not used, to optimize power consumption.

STM32F103 Family Clockc Diagram



anhpham@hcmut.edu.vn

MX_GPIO_Init

```
14  /**
15    * @brief GPIO Initialization Function
16    * @param None
17    * @retval None
18    */
19  void MX_GPIO_Init(void)
20  {
21  //  GPIO_InitTypeDef GPIO_InitStruct = {0};
22
23      /* GPIO Ports Clock Enable */
24      __HAL_RCC_GPIOC_CLK_ENABLE();
25      __HAL_RCC_GPIOD_CLK_ENABLE();
26      __HAL_RCC_GPIOA_CLK_ENABLE();
27      __HAL_RCC_GPIOB_CLK_ENABLE();
28
29  //  /*Configure GPIO pin Output Level */
30  //  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
31  //
32      LED_Init();
33      GPIO_Relay_Init();
34      Buzzer_Init();
35      SPI_CS_Init();
36      ZeroPoint_Detection_Pin_Init();
37  }
```

anhpham@hcmut.edu.vn

Input Output Pins Initialization

```
38
39  void LED_Init(void){
40
41      GPIO_InitTypeDef GPIO_InitStruct = {0};
42
43      GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
44      GPIO_InitStruct.Pull  = GPIO_PULLUP;
45      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
46      GPIO_InitStruct.Pin = LED2_PIN;
47      HAL_GPIO_Init(LED2_GPIO_PORT, &GPIO_InitStruct);
48
49  }
50
```

```
102
103 void ZeroPoint_Detection_Pin_Init(void){
104     GPIO_InitTypeDef GPIO_InitStruct = {0};
105
106     GPIO_InitStruct.Pin = ZERO_POINT_DETECTION_PIN;
107     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
108     GPIO_InitStruct.Pull = GPIO_NOPULL;
109     HAL_GPIO_Init(ZERO_POINT_DETECTION_PORT, &GPIO_InitStruct);
110
111 #if(VERSION_EBOX == 2)
112     /* EXTI interrupt init*/
113       HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
114       HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
115 #else
116       HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
117       HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
118 #endif
119 }
120
```

```
54  void GPIO_Relay_Init(void){
55      GPIO_InitTypeDef GPIO_InitStruct = {0};
56
57      GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
58      GPIO_InitStruct.Pull  = GPIO_PULLUP;
59      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
60
61      GPIO_InitStruct.Pin = RELAY_PIN_0;
62      HAL_GPIO_Init(RELAY_PORT_0, &GPIO_InitStruct);
63
64      GPIO_InitStruct.Pin = RELAY_PIN_1;
65      HAL_GPIO_Init(RELAY_PORT_1, &GPIO_InitStruct);
66      GPIO_InitStruct.Pin = RELAY_PIN_2;
67      HAL_GPIO_Init(RELAY_PORT_2, &GPIO_InitStruct);
68      GPIO_InitStruct.Pin = RELAY_PIN_3;
69      HAL_GPIO_Init(RELAY_PORT_3, &GPIO_InitStruct);
70      GPIO_InitStruct.Pin = RELAY_PIN_4;
71      HAL_GPIO_Init(RELAY_PORT_4, &GPIO_InitStruct);
72      GPIO_InitStruct.Pin = RELAY_PIN_5;
73      HAL_GPIO_Init(RELAY_PORT_5, &GPIO_InitStruct);
74      GPIO_InitStruct.Pin = RELAY_PIN_6;
75      HAL_GPIO_Init(RELAY_PORT_6, &GPIO_InitStruct);
76      GPIO_InitStruct.Pin = RELAY_PIN_7;
77      HAL_GPIO_Init(RELAY_PORT_7, &GPIO_InitStruct);
78      GPIO_InitStruct.Pin = RELAY_PIN_8;
79      HAL_GPIO_Init(RELAY_PORT_8, &GPIO_InitStruct);
80      GPIO_InitStruct.Pin = RELAY_PIN_9;
81      HAL_GPIO_Init(RELAY_PORT_9, &GPIO_InitStruct);
82  #if(VERSION_EBOX == 2)
83      GPIO_InitStruct.Pin = PD2_RELAY_ENABLE_PIN;
84      HAL_GPIO_Init(PD2_RELAY_ENABLE_PORT, &GPIO_InitStruct);
85  #endif
86  }
```


UART3_Init

```
78 /**
79   * @brief USART3 Initialization Function
80   * @param None
81   * @retval None
82   */
83 void UART3_Init(void)
84 {
85     Uart3Handle.Instance = USART3;
86     Uart3Handle.Init.BaudRate = 115200;
87     Uart3Handle.Init.WordLength = UART_WORDLENGTH_8B;
88     Uart3Handle.Init.StopBits = UART_STOPBITS_1;
89     Uart3Handle.Init.Parity = UART_PARITY_NONE;
90     Uart3Handle.Init.Mode = UART_MODE_TX_RX;
91     Uart3Handle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
92     Uart3Handle.Init.OverSampling = UART_OVERSAMPLING_16;
93   if (HAL_UART_Init(&Uart3Handle) != HAL_OK) {
94     Error_Handler();
95   }
96 }
```


Timer_Init

Configure the TIM peripheral

In this example TIM3 input clock (TIM3CLK) is set to APB1 clock (PCLK1) x2, since APB1 prescaler is set to 4 (0x100).

TIM3CLK = PCLK1*2

- PCLK1 = HCLK/2
- => TIM3CLK = PCLK1*2 = (HCLK/2)*2 = HCLK = SystemCoreClock
- To get TIM3 counter clock at 10 KHz, the Prescaler is computed as following:
- Prescaler = (TIM3CLK / TIM3 counter clock) - 1
- Prescaler = (SystemCoreClock /10 KHz) - 1

Note:

- SystemCoreClock variable holds HCLK frequency and is defined in **system_stm32f1xx.c** file.
- Each time the core clock (HCLK) changes, user had to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.
- This variable is updated in three ways:
  1) by calling CMSIS function SystemCoreClockUpdate()
  2) by calling HAL API function HAL_RCC_GetSysClockFreq()
  3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency

An FSM Function

```
enum {St_Intro, St_Voltmeter, St_Temperature, St_Clock,} State_Machine;
unsigned char State = St_Intro;
void FSM(){
	switch (State)
	{
		case St_Intro:
			Intro();
			break;
		case St_Voltmeter:
			Voltmeter();
			break;
		case St_Temperature:
			Temperature();
			break;
		case St_Clock:
			Clock();
			break;
	}
}
```


- CO3009 -

- Switches and Buttons -

Introduction

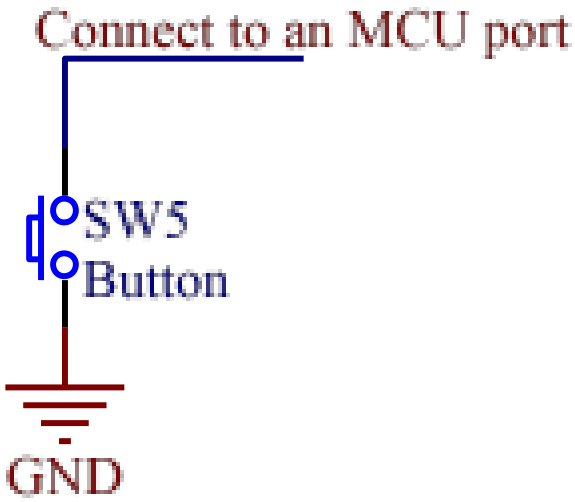
- Embedded systems usually use switches/buttons as part of their user interface.
- This general rule applies from the most basic remote-control system for opening a garage door, right up to the most sophisticated aircraft autopilot system.
- Whatever the system you create, you need to be to create a reliable switch interface.

What we will do in this lecture

- How we can read inputs from mechanical switches in an embedded application

anhpham@hcmut.edu.vn

The need for pull-up resistors



- The hardware operates as follows:
  - When the switch is open, it has no impact on the port pin. An internal resistor on the port 'pull up' the pin to the supply voltage of the MCU (typically 5V or 3V3). If we read the pin, we will see the value '1'
  - When switch is closed (pressed), the pin voltage will be 0V. If we read the pin, we will see the value '0'

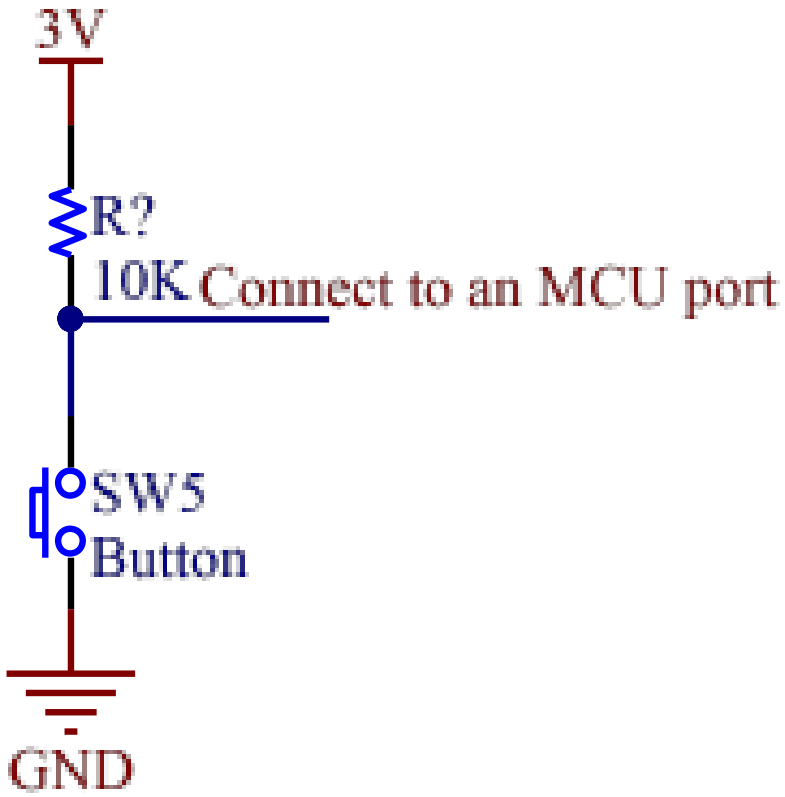
The need for pull-up resistors

With pull-ups:

Without pull-ups:

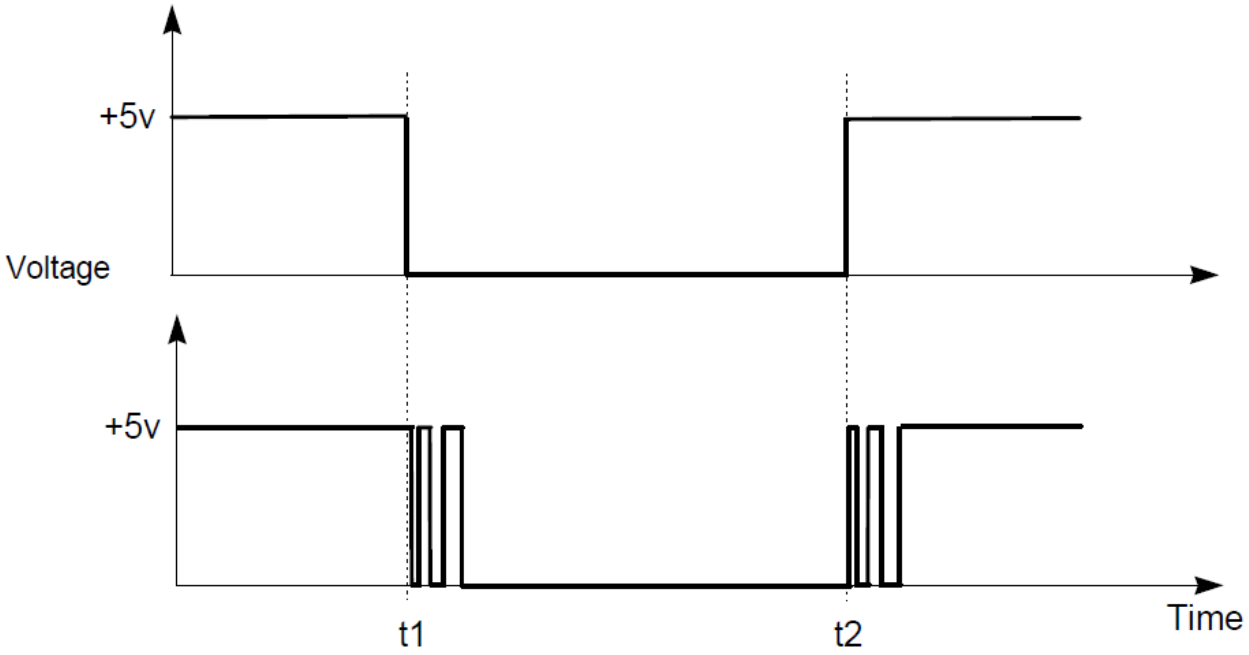
anhpham@hcmut.edu.vn

The need for pull-up resistors



Dealing with switch bounce

- In practice, all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for short period of time) after the switch is closed or opened.



Dealing with switch bounce

- As far as the MCU concerns, each “bounce” is equivalent to one press and release of an “ideal” switch.

- Without appropriate software design, this can give several problems
  - Rather than reading ‘A’ from a keypad, we may read ‘AAAAA’
  - Counting the number of times that a switch is pressed becomes extremely difficult.

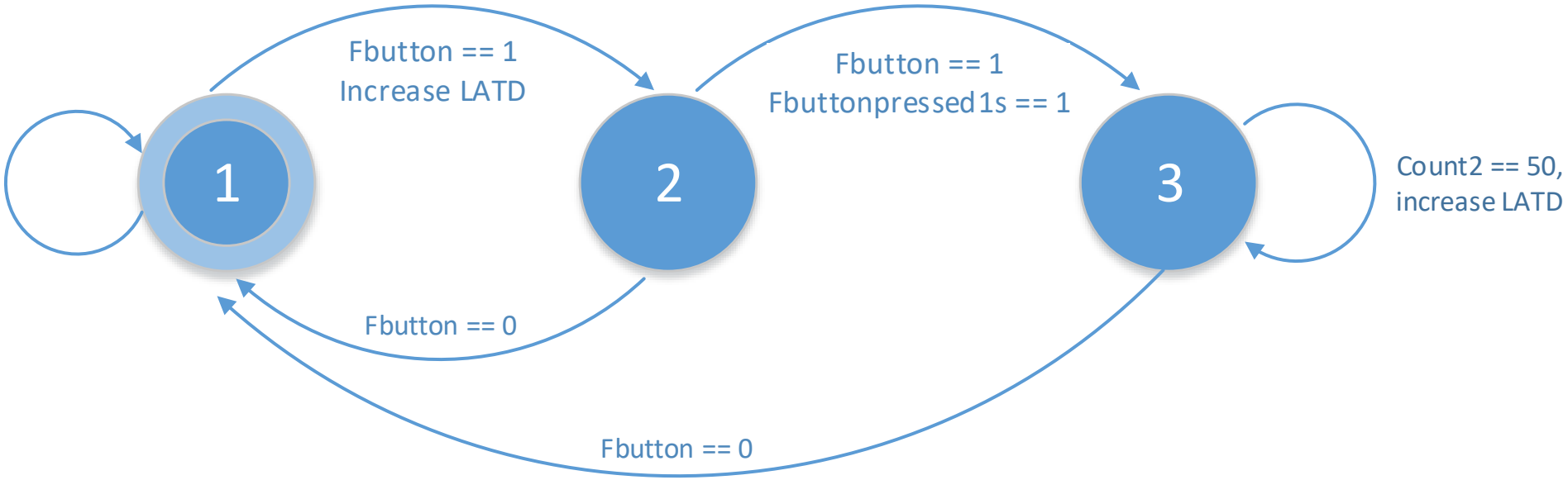
Dealing with switch bounce

- Creating a simple software to check for a valid switch input is straightforward:
  - Read the relevant port pin
  - If we think we have detected a switch depression, we wait for 20ms and then read the pin again
  - If the second reading confirms the first reading, we assume the switch really has been depressed.
- Note that the figure of '20ms' will depend on the switch used.

Round robin with timer interrupts example

- **W**rite a program that
  - Has a timer which has an interrupt in every 10 milliseconds.
  - Reads values of button **RA5** every 10 milliseconds.
  - Increases the value of LEDs connected to **PORTD** when the button **RA5** is pressed.
  - Increases the value of **PORTD** automatically in every 0.5 second, if the button RA5 is pressed in more than 1 second.

Finite State Machine

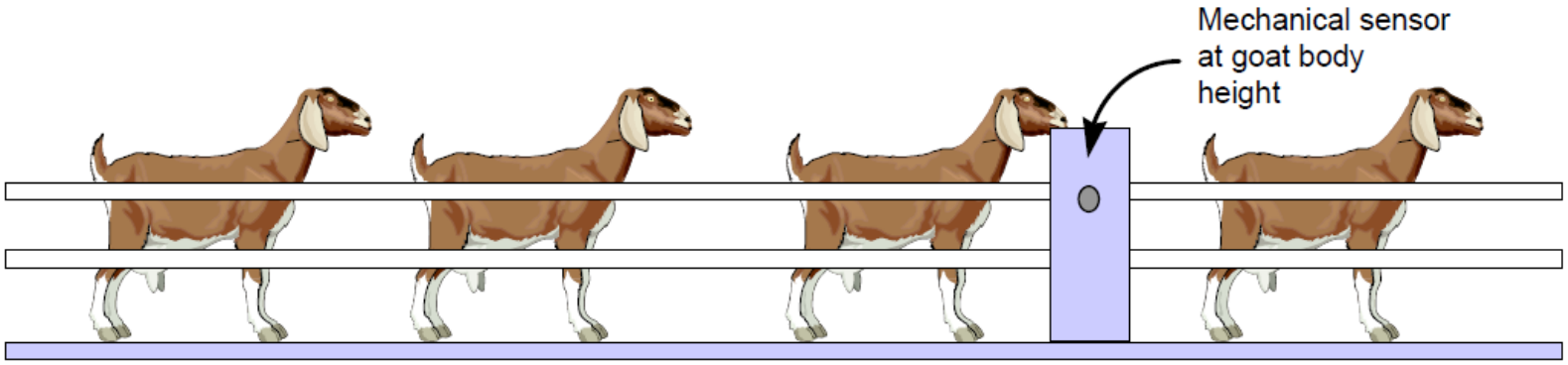


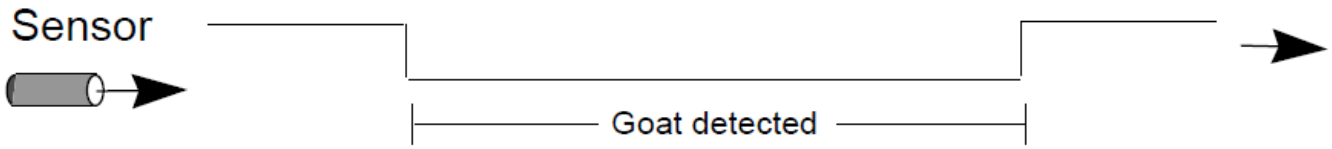
Example Code

```
void main (void) {
    enum {STATE1, STATE2, STATE3} eState;
    while (TRUE) {
        switch (eState){
          case STATE1:
            if (fbutton) {
                increase LATD;
                eState = STATE2;
            }
          break;
          case STATE2:
            if (fbutton == 0) eState = STATE1;
            else if (fbuttonpress1s) eState = STATE3;
          break;
          case STATE3:
            if (normalCountUpFlag) {
                increase LATD;
                normalCountUpFlag = 0;
             }
            if (fbutton == 0) eState = STATE1;
                break;
          }
        }
}
```

```
void Readbutton(void) {
    firstReadRA5 = secondReadRA5;
    secondReadRA5 = digitalRead(BUTTON_1);
    if (secondReadRA5 == firstReadRA5) {
        if (firstReadRA5) {
            fbutton = 1;
            count1++
            if (count1 >= 100){
                fbuttonpressed1s = 1;
                count2 ++;
                if (count2 >= 50){
                    normalCountUpFlag = 1
                    count2 = 0;
                }
            }
        } else {
            fbutton = 0;
            fbuttonpressed1s = 0;
            count1 = 0;
            count2 = 0;
        }
    }
}
```


Exercise: Counting Goats





- CO3009 -

- Adding Structure To Your Code -

What we will do in this lecture

- Describe how to use an object-oriented style of programming with C programs
  - allowing the creation of libraries of code that can be easily for use in different embedded projects
- Describe how to create and use a 'Project Header' file.
  - This file encapsulates key aspects of the hardware environment, such as the type of processor to be used, the oscillator frequency, and the number of oscillator cycles required to execute each instructions.
  - This helps to document the system, and make it easier to port the code to a different processor
- Describe how to create and use a 'Port Header' file.
  - This brings together all details of the port access from the whole system. Like project header, this helps during porting and also serves as a means of documenting important system features

Object Oriented Programming with C

Language generation	Example languages
-	Machine Code
First-Generation Language (1GL)	Assembly Language.
Second-Generation Languages (2GLs)	COBOL, FORTRAN
Third-Generation Languages (3GLs)	C, Pascal, Ada 83
Fourth-Generation Languages (4GLs)	C++, Java, Ada 95

Graham notes[1]:

> *"[The phrase] 'object-oriented' has become almost synonymous with modernity, goodness and worth in information technology circles."*

Jalote notes[2]:

> *"One main claimed advantage of using object orientation is that an OO model closely represents the problem domain, which makes it easier to produce and understand designs."*

---

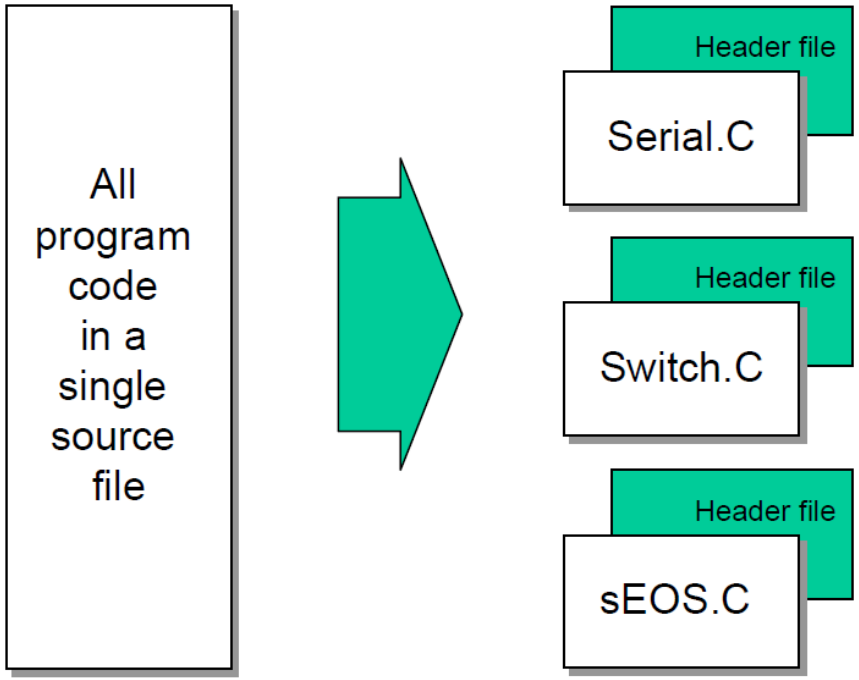
1 **Graham, I.** (1994) *"Object-Oriented Methods,"* (2nd Ed.) Addison-Wesley. Page 1.

2 **Jalote, P.** (1997) *"An Integrated Approach to Software Engineering"*, (2nd Ed.) Springer-Verlag. Page 273.

O-O languages are not readily available for small embedded systems, primarily because of the overheads that can result from the use of some of the features of these languages.

Object Oriented Programming with C

It is possible to create 'file-based classes' in C without imposing a significant memory or CPU load.



C Program Layout

.c files

1. Header comment
2. #included files
3. #defines
4. local struct typedefs
5. local prototypes
6. global vars
7. main function (if present)
8. local functions

.h files

1. Header comment
2. #ifndef guard
3. #included files
4. #defines
5. struct typedefs
6. prototypes
7. (extern) global vars

Example of 'O-O C' – PC_IO.h

```
/*-------------------------------------------------------------*-

   PC_IO.H (v1.00)

  ----------------------------------------------------------

   - see PC_IO.C for details.

-*-------------------------------------------------------------*/

#ifndef _PC_IO_H
#define _PC_IO_H

/* ------ Public constants ---------------------------------- */

/* Value returned by PC_LINK_Get_Char_From_Buffer if no char is
   available in buffer */
#define PC_LINK_IO_NO_CHAR 127

/* ------ Public function prototypes ------------------------ */

void PC_LINK_IO_Write_String_To_Buffer(const char* const);
void PC_LINK_IO_Write_Char_To_Buffer(const char);

char PC_LINK_IO_Get_Char_From_Buffer(void);

/* Must regularly call this function... */
void PC_LINK_IO_Update(void);

#endif

/*-------------------------------------------------------------*-
  ---- END OF FILE ------------------------------------------
-*-------------------------------------------------------------*/
```


```
/*-------------------------------------------------------------*-

   PC_IO.C (v1.00)

  ------------------------------------------------------------

   [INCOMPLETE - STRUCTURE ONLY - see EC Chap 9 for complete library]

-*-------------------------------------------------------------*/

#include "Main.H"
#include "PC_IO.H"

/* ------ Public variable definitions ------------------------ */

tByte In_read_index_G;     /* Data in buffer that has been read */
tByte In_waiting_index_G;  /* Data in buffer not yet read */

tByte Out_written_index_G; /* Data in buffer that has been written */
tByte Out_waiting_index_G; /* Data in buffer not yet written */

/* ------ Private function prototypes ------------------------ */

static void PC_LINK_IO_Send_Char(const char);

/* ------ Private constants ---------------------------------- */

/* The receive buffer length */
#define RECV_BUFFER_LENGTH 8

/* The transmit buffer length */
#define TRAN_BUFFER_LENGTH 50

#define XON  0x11
#define XOFF 0x13

/* ------ Private variables ---------------------------------- */

static tByte Recv_buffer[RECV_BUFFER_LENGTH];
static tByte Tran_buffer[TRAN_BUFFER_LENGTH];
```

PC_IO.c

```
/*-------------------------------------------------------------*/
void PC_LINK_IO_Update(...)
   {
   ...
   }


/*-------------------------------------------------------------*/
void PC_LINK_IO_Write_Char_To_Buffer(...)
   {
   ...
   }

/*-------------------------------------------------------------*/
void PC_LINK_IO_Write_String_To_Buffer(...)
   {
   ...
   }

/*-------------------------------------------------------------*/
char PC_LINK_IO_Get_Char_From_Buffer(...)
   {
   ...
   }

/*-------------------------------------------------------------*/
void PC_LINK_IO_Send_Char(...)
   {
   ...
   }
```


Project Header File

- Includes all header files that are used
- Defines operating frequency
- Depends on the project

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"
#include "stm32f1xx_nucleo.h"

#include "stdio.h"
#include "string.h"

typedef enum
{
  ABNORMAL = 0,
  NORMAL = !ABNORMAL
} WorkingStatus;

#define DEBUG_INIT(X)                   //X


/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* Private defines -----------------------------------------------------------*/
#define     VERSION_EBOX              1
#define     INTERRUPT_TIMER_PERIOD    10 //ms
#define     WATCHDOG_ENABLE           1
//#define       SIM5320                   1
#define     SIM7600                   1

#define LED2_PIN                       GPIO_PIN_2
#define LED2_GPIO_PORT                 GPIOB

//LED output control signals
#define LED_SDI                        GPIO_PIN_6
#define LED_SDI_PORT                   GPIOC
#define LED_SCK                        GPIO_PIN_3
#define LED_SCK_PORT                   GPIOC
#define LED_LE                         GPIO_PIN_4
#define LED_LE_PORT                    GPIOC
#define LED_OE                         GPIO_PIN_5
#define LED_OE_PORT                    GPIOC

//BUZZER
#define PB5_BUZZER_PIN                 GPIO_PIN_5
#define PB5_BUZZER_PORT                GPIOB
#define BUZZER_PIN                     PB5_BUZZER_PIN
#define BUZZER_PORT                    PB5_BUZZER_PORT

//SPI CS pin
#define SPI_CS_PIN                     GPIO_PIN_12
#define SPI_CS_PORT                    GPIOB

```


Project Header File

```
#ifndef _MAIN_H
#define _MAIN_H

/*-------------------------------------------------------
   WILL NEED TO EDIT THIS SECTION FOR EVERY PROJECT
  ------------------------------------------------------- */

/* Must include the appropriate microcontroller header file here */
#include <reg52.h>

/* Oscillator / resonator frequency (in Hz) e.g. (11059200UL) */
#define OSC_FREQ (12000000UL)

/* Number of oscillations per instruction (12, etc)
   12 - Original 8051 / 8052 and numerous modern versions
    6 - Various Infineon and Philips devices, etc.
    4 - Dallas 320, 520 etc.
    1 - Dallas 420, etc. */
#define OSC_PER_INST (12)

/* -------------------------------------------------------
   SHOULD NOT NEED TO EDIT THE SECTIONS BELOW
  ------------------------------------------------------- */

/* Typedefs (see Chap 5)    */
typedef unsigned char tByte;
typedef unsigned int  tWord;
typedef unsigned long tLong;

/* Interrupts (see Chap 7)    */
#define INTERRUPT_Timer_0_Overflow 1
#define INTERRUPT_Timer_1_Overflow 3
#define INTERRUPT_Timer_2_Overflow 5

#endif
```


Common Data Types

```
typedef unsigned char tByte;
typedef unsigned int tWord;
typedef unsigned long tLong;
```

In C, the typedef keyword allows us to provide aliases for data types: we can then use these aliases in place of the original types.

For example, we use `tWord Temperature;` to declare a variable rather than use `unsigned int Temperature;`

Common Data Types

- The main reason for using **typedef** keyword is to simplify and promote the use of unsigned data types.
  - Most MCUs do not support signed arithmetic and extra code is required to manipulate signed data → this reduces your program speed and increases the program size.
  - Use of bitwise operators generally makes sense only with unsigned data types: use of **typedef** variable reduces the likelihood that programmers will inadvertently apply these bitwise operators to signed data.
  - Use of the **typedef** keyword can make it easier to adapt your code for use on a different processor.

Why use the Project Header?

- Use of **Project header** can help to make your code more readable
  - Anyone using your projects knows where to find key information, such as the model of microcontroller and the oscillator frequency required to execute the software.
- The use of a project header can help to make your code more easily portable, by placing some of the key MCU-dependent data in one place
  - If you want to change the processor of the oscillator used, then in many cases- you will need to make changes only to the Project Header

The Port Header File

- Consider, for example, that we have three files in a project (A, B and C), each of which require access to one or more port pins, or to a complete port
- All the port access requirements are spread over multiple files
- It is better to intergrade all port access in single port header file

File A may include the following:

```
/* File A */
sbit Pin_A = P3^2;
...
```

File B may include the following:

```
/* File B */
#define Port_B = P0;
...
```

File C may include the following:

```
/* File C */
sbit Pin_C = P2^7;
```


Port Header File

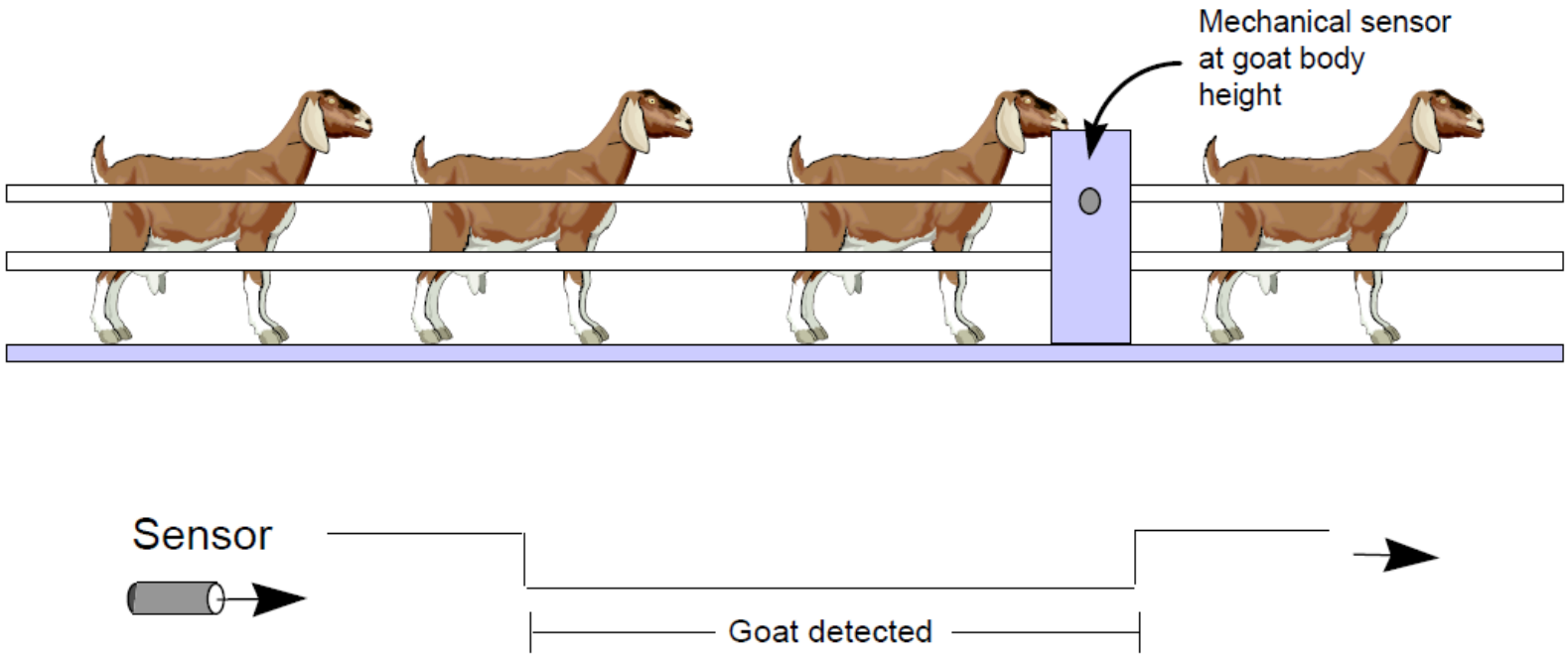
- Includes all Pins that are used
- Depends on the project and the MCU used

```
265 //Relay control pins and ports
266 #define PA11_OUT0                   GPIO_PIN_11
267 #define PA11_OUT0_PORT              GPIOA
268 #define PA12_OUT1                   GPIO_PIN_12
269 #define PA12_OUT1_PORT              GPIOA
270 #define PB8_OUT2                    GPIO_PIN_8
271 #define PB8_OUT2_PORT               GPIOB
272 #define PB9_OUT3                    GPIO_PIN_9
273 #define PB9_OUT3_PORT               GPIOB
274 #define PA15_OUT4                   GPIO_PIN_15
275 #define PA15_OUT4_PORT              GPIOA
276
277 #define PC10_OUT5                   GPIO_PIN_10
278 #define PC10_OUT5_PORT              GPIOC
279 #define PC11_OUT6                   GPIO_PIN_11
280 #define PC11_OUT6_PORT              GPIOC
281 #define PC12_OUT7                   GPIO_PIN_12
282 #define PC12_OUT7_PORT              GPIOC
283
284 #define PB3_OUT8                    GPIO_PIN_3
285 #define PB3_OUT8_PORT               GPIOB
286
287 #define PB4_OUT9                    GPIO_PIN_4
288 #define PB4_OUT9_PORT               GPIOB
```

```
193
194 /* Definition for SPIx Pins */
195 #define SPI2_NSS_PIN                GPIO_PIN_12
196 #define SPI2_NSS_GPIO_PORT          GPIOB
197
198 #define SPI2_SCK_PIN                GPIO_PIN_13
199 #define SPI2_SCK_GPIO_PORT          GPIOB
200 #define SPI2_MISO_PIN               GPIO_PIN_14
201 #define SPI2_MISO_GPIO_PORT         GPIOB
202 #define SPI2_MOSI_PIN               GPIO_PIN_15
203 #define SPI2_MOSI_GPIO_PORT         GPIOB
204
205
```

```
215
216 /* Definition for I2Cx Pins */
217 #define I2C1_SCL_PIN                GPIO_PIN_6
218 #define I2C1_SCL_GPIO_PORT          GPIOB
219 #define I2C1_SDA_PIN                GPIO_PIN_7
220 #define I2C1_SDA_GPIO_PORT          GPIOB
221
```


Restructuring the Goat-counting Example



- CO3009 -

- Working with Time: Interrupts, Counters, and Timers -

Timer in Stm32F103

- Advanced-control timers (TIM1 and TIM8)
- General-purpose timers (TIM2 to TIM5)
- Basic timers (TIM6 and TIM7)
- General-purpose timers (TIM9 to TIM14)
- Real-time clock (RTC)
- Independent watchdog (IWDG)
- Window watchdog (WWDG)

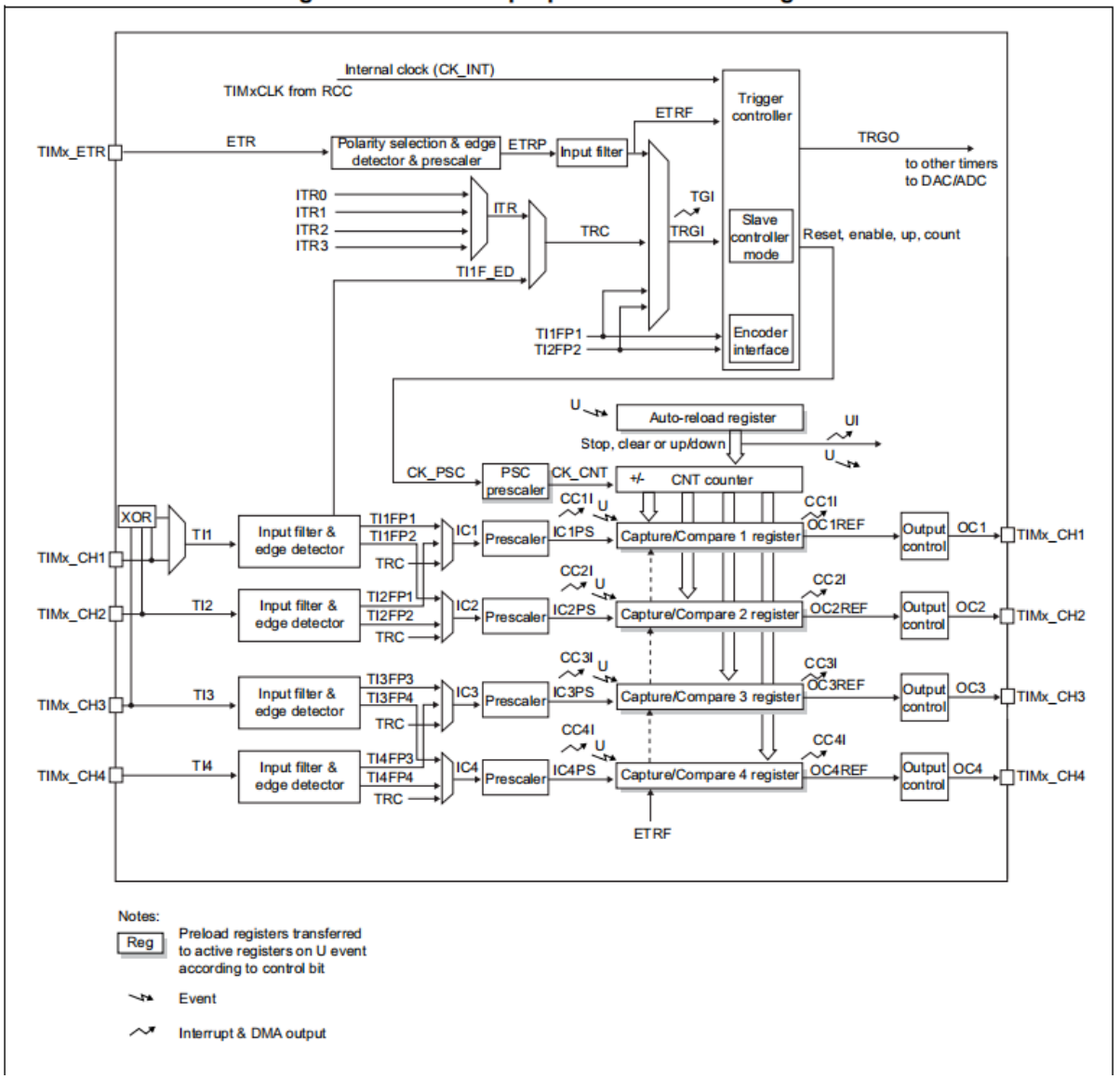
General-purpose Timers (TIM2 to TIM5)

- The general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.
- They may be used for a variety of purposes, including:
  - measuring the pulse lengths of input signals (input capture) or
  - generating output waveforms (output compare and PWM).
- Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.
- The timers are completely independent, and do not share any resources.

General-purpose TIMx Timer Features

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65536.
- Up to 4 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge- and Center-aligned modes)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 100. General-purpose timer block diagram



TIMx Functional Description

- Time-base unit
  - Prescaler description
- Counter modes
  - Upcounting mode
  - Downcounting mode
  - Center-aligned mode (up/down counting)

https://www.youtube.com/watch?v=VfbW6nfG4kw&t=637s

https://drive.google.com/file/d/1IPCnbnfwLIAEaNPWvPxO6MSJ1Psvruva/view?usp=sharing

Figure 101. Counter timing diagram with prescaler division change from 1 to 2

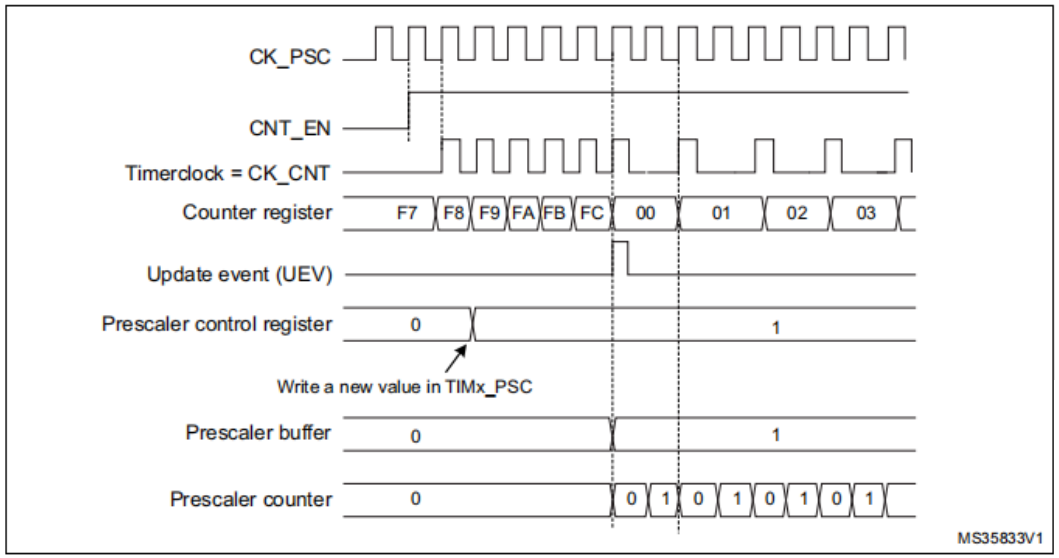


Figure 102. Counter timing diagram with prescaler division change from 1 to 4

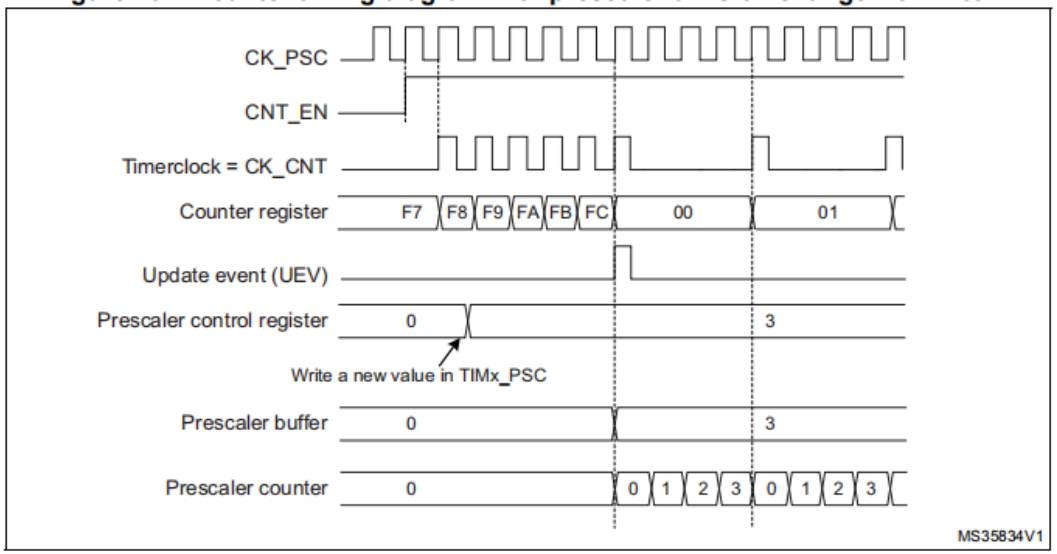


Figure 103. Counter timing diagram, internal clock divided by 1

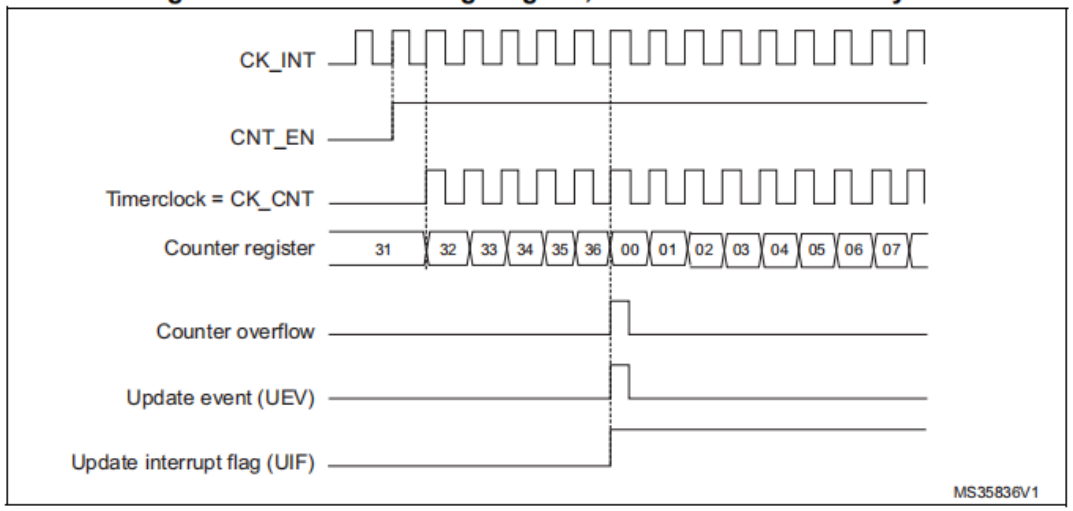
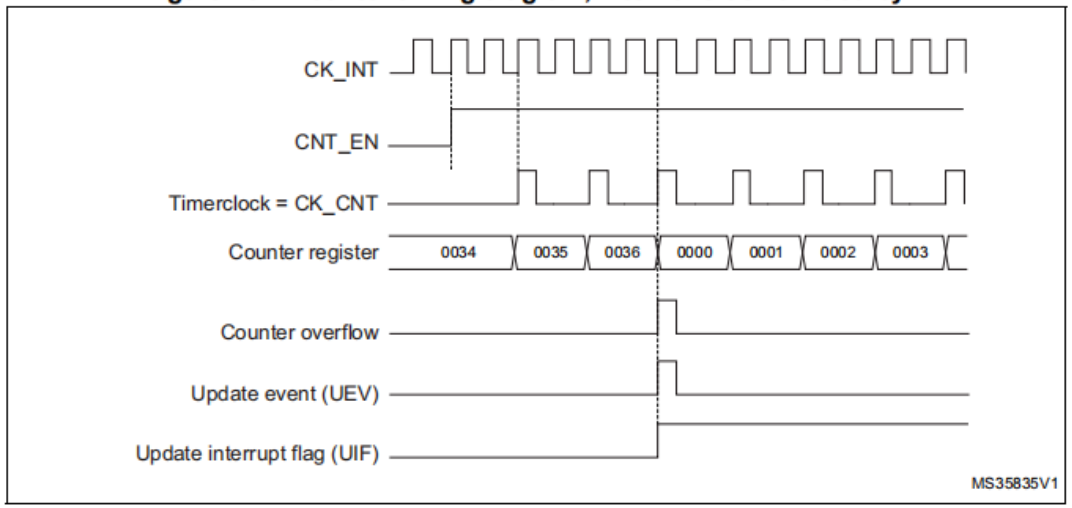


Figure 104. Counter timing diagram, internal clock divided by 2



Clock Selection

- The counter clock can be provided by the following clock sources:
  - Internal clock (CK_INT)
  - External clock mode1: external input pin (TIx)
  - External clock mode2: external trigger input (ETR).
  - Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, Timer1 can be configured to act as a prescaler for Timer 2.

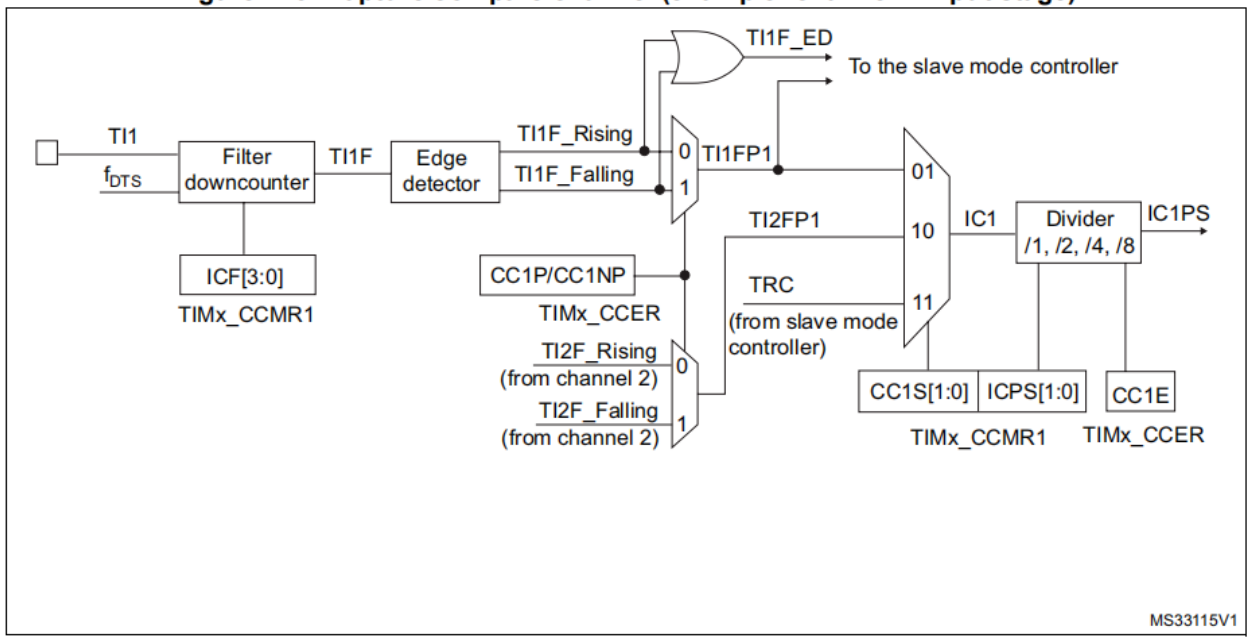
Questions?

- How to configure the upcounter to count in response to a rising edge on the TI2 input?
- How to configure the upcounter to count each 2 rising edges on ETR?

Capture/Compare Channels

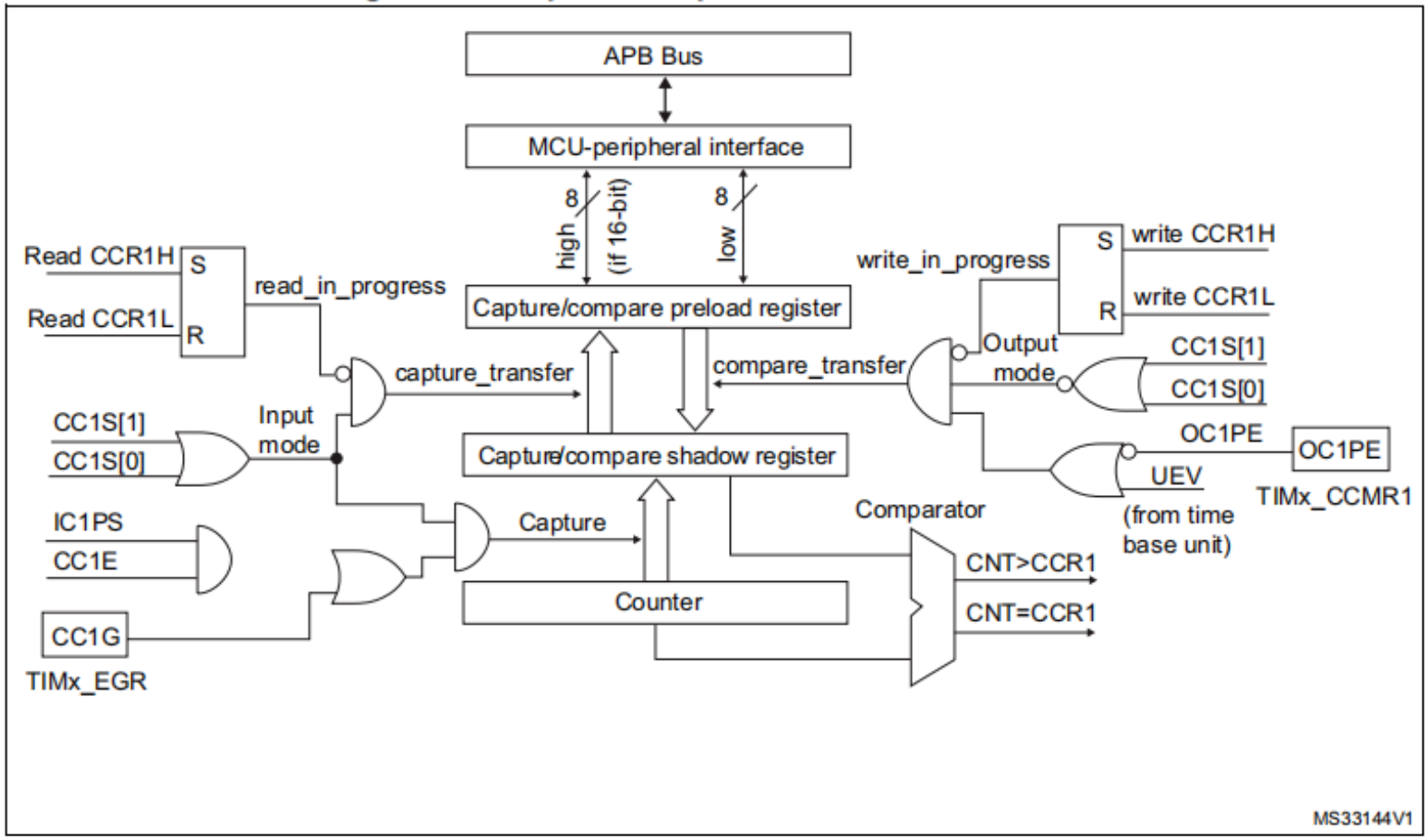
- Each Capture/Compare channel is built around a capture/compare register (including a shadow register),
- An input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control)

Figure 125. Capture/compare channel (example: channel 1 input stage)



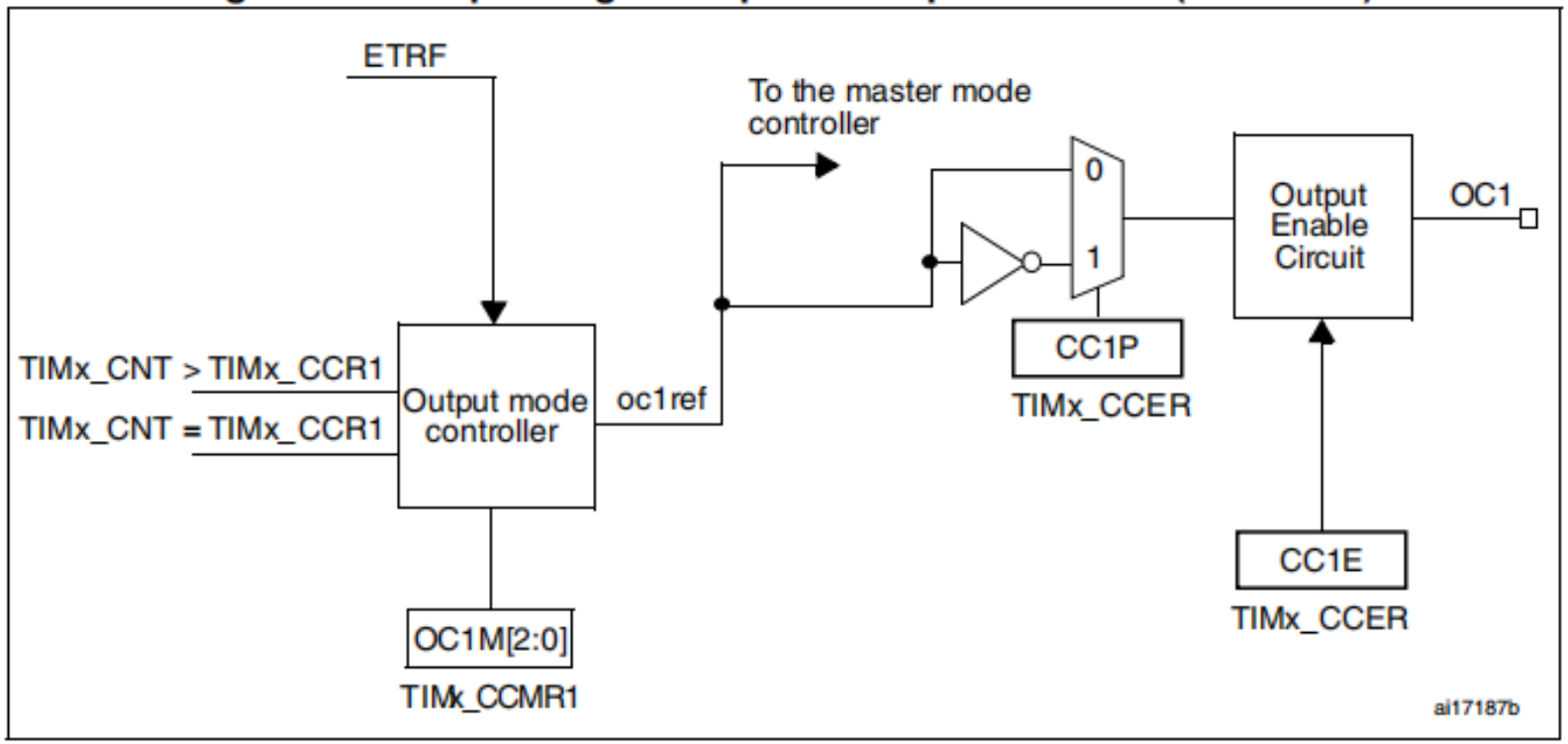
Capture/Compare Channels

Figure 126. Capture/compare channel 1 main circuit



Capture/Compare Channels

Figure 127. Output stage of capture/compare channel (channel 1)



Input/Output Capture/Compare Mode

- Input Capture Mode
  - How to capture the counter value in TIMx_CCR1 when TI1 input rises?
  - What happens when an input capture occurs?

https://www.youtube.com/watch?v=RZ0uszK2PMo

- Output Compare Mode
  - This function is used to control an output waveform or indicating when a period of time has elapsed.

https://drive.google.com/file/d/1F50A8NR4AR2zPZc7wvRDJZjfTO18WId1/view?usp=sharing

PWM Input Mode

- This mode is a particular case of input capture mode. The procedure is the same except:
  - Two ICx signals are mapped on the same TIx input.
  - These 2 ICx signals are active on edges with opposite polarity.
  - One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

Questions

- How to measure the period and the duty cycle of a PWM signal?
- How to measure the period and the duty cycle of the PWM applied on TI1?

https://drive.google.com/file/d/17RvIlt_PaihrqqBR-7AkB0DLXMltwL_b/view?usp=sharing

PWM Mode

- Pulse width modulation mode allows generating a signal with:
  - A frequency determined by the value of the TIMx_ARR register and
  - A duty cycle determined by the value of the TIMx_CCRx register.

- The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or ‘111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register.

- The user must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx_CR1 register.

https://www.youtube.com/watch?v=92_CWBWXPw0

Exercises

- Write a program to generate a square wave on pin PD5 (1 kHz)
  - Using **HAL_Delay(uint32_t ms)** function
- Write a program using Timer3 to generate a square wave on pin PD5 (1 kHz)
- Generate square waves on pins PD1 (10 Hz) and PD7 (100 Hz) respectively.

Exercises (cont)

- Use Timer2 and Timer3 interrupts to generate square waves on pins PD1 (10 Hz) and PD7 (100 Hz) respectively.
- Use Timer3 interrupts to generate square waves on pins PD1 (10 Hz) and PD7 (100 Hz) respectively.
- Write a program to count up PD[0-7] every 1 second using TIMER3 interrupt.

- CO3009 -

- Basic Hardware Foundations -

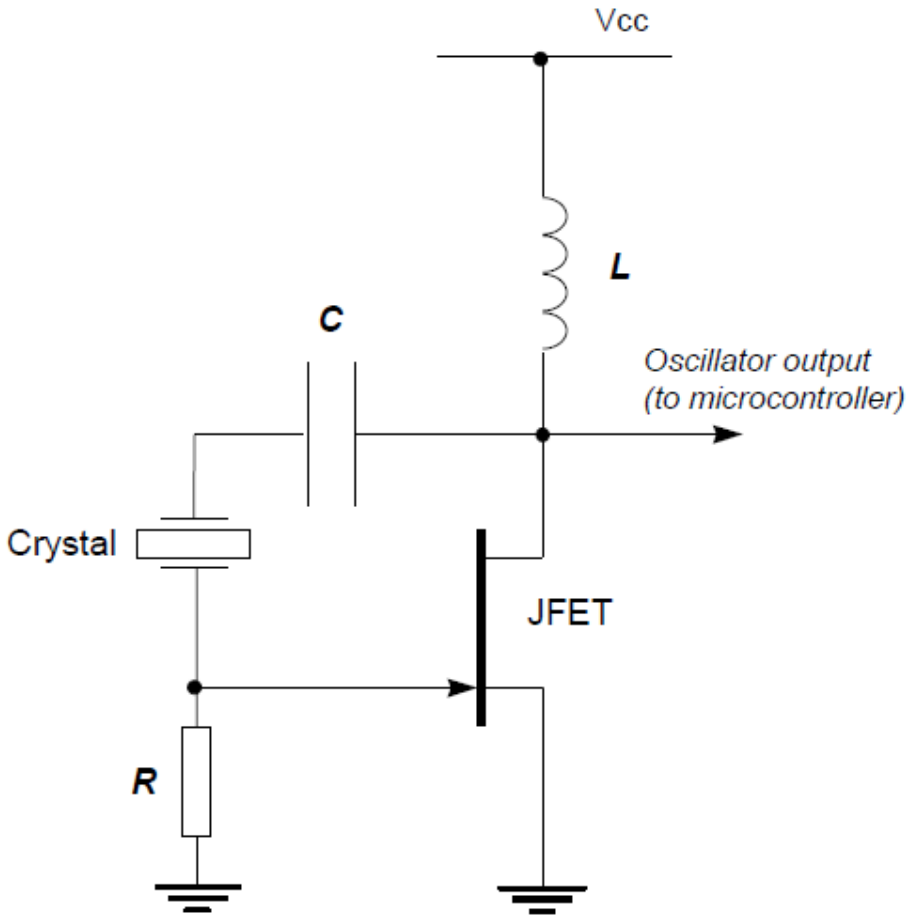
Contents

- Oscillator circuits
- Reset circuits

Oscillator Circuits

- All digital computer systems are driven by some form of oscillator circuits
- This circuits is the heartbeat of the system and is crucial to correct operations
- For example:
  - If the oscillator fails, the system will not function at all
  - If the oscillator runs irregularly, any timing calculation performed by the system will be inaccurate.

Crystal Oscillator

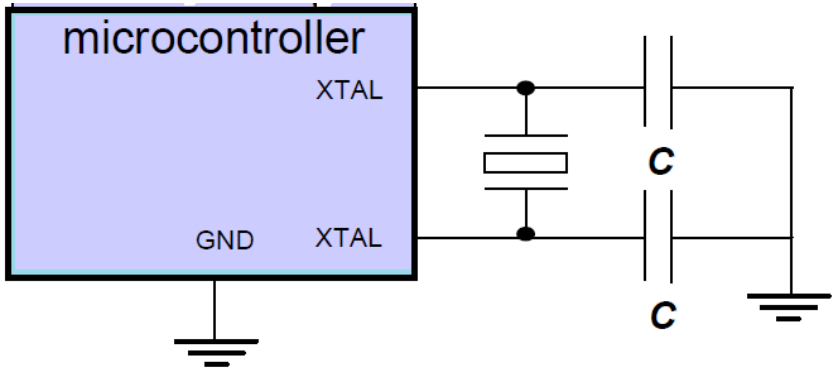


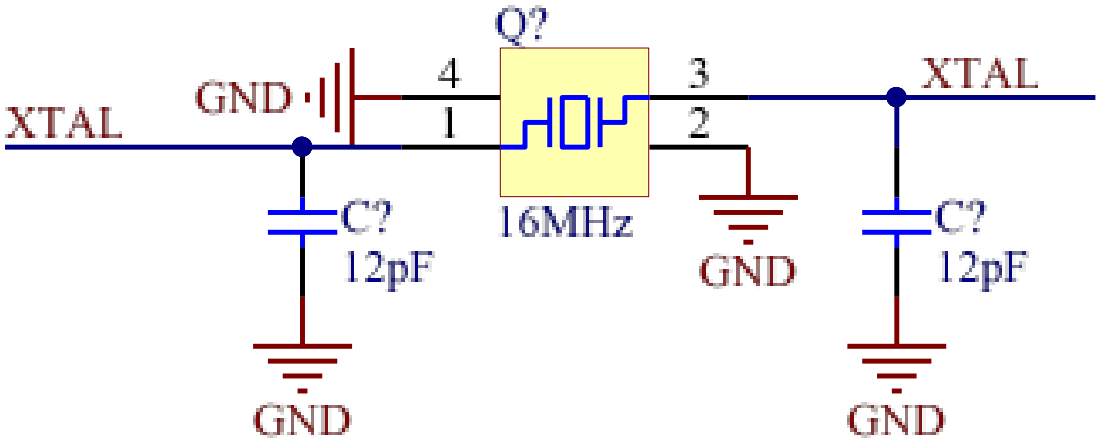
Crystals may be used to generate a popular form of oscillator circuit knows as a **Pierce oscillator**.

Crystal oscillator

- A variant of the Pierce oscillator is common in the MCU.
- To create such an oscillator, most of the components are included on the MCU itself.
- The user of the MCU must generally only supply the crystal and two small capacitors to complete the oscillator implementation.

How to connect a crystal to an MCU





Oscillator Frequency and Machine Cycle Period

- In the original members of the 8051 family, the machine cycle takes 12 oscillator periods.

- Infineon C515C: 6 periods

- Dallas 89C420: 1 period

- P18F8722: 4 periods

Keep the Clock Frequency as Low as Possible

- Many developers select an oscillator/resonator frequency that is at or near the maximum value supported by a particular device
- This can be a mistake:
  - Many application do not require the levels of performance that an MCU can provide.
  - The electromagnetic interference (EMI) generated by a circuit increases with clock frequency
  - In most modern MCUs, there is almost linear relationship between the oscillator frequency and the power-supply current. As a result, by using the lowest frequency necessary it is possible to reduce the power requirement: this can be useful in many applications.
  - When accessing low-speed peripherals (such as slow memory or LCD displays), programming and hardware design can be greatly simplified, and the cost of peripherals components can be reduced.

Keep the Clock Frequency as Low as Possible

In general, you should operate at the lowest possible oscillator frequency compatible with the performance needs of your application.

Stability Issues

- A key factor in selecting an oscillator for your system is the issue of oscillator stability.
- In most cases, oscillator stability expressed in figures such as '+/- 20 ppm': 20 parts per million
- To see what this mean in practice, consider that there are approximately 32 million seconds in a year. In every million seconds, your crystal may gain (or lose) 20 seconds. Over a year, a clock based on a 20-ppm crystal may gain (or lose) about 32x20 second, or around 10 minutes

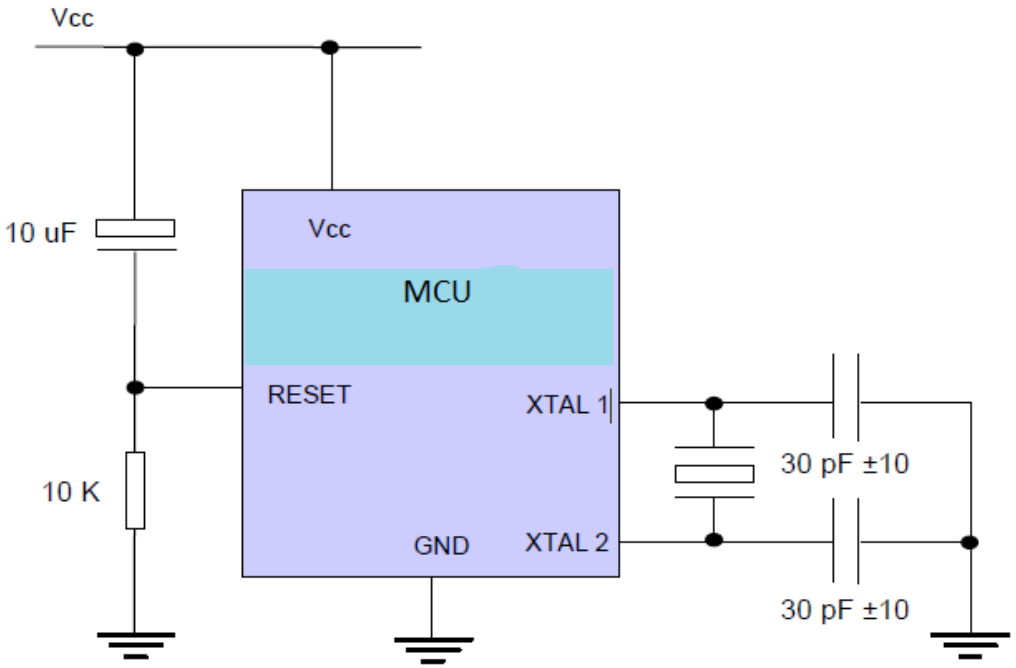
Improving the Stability of A Crystal Oscillator

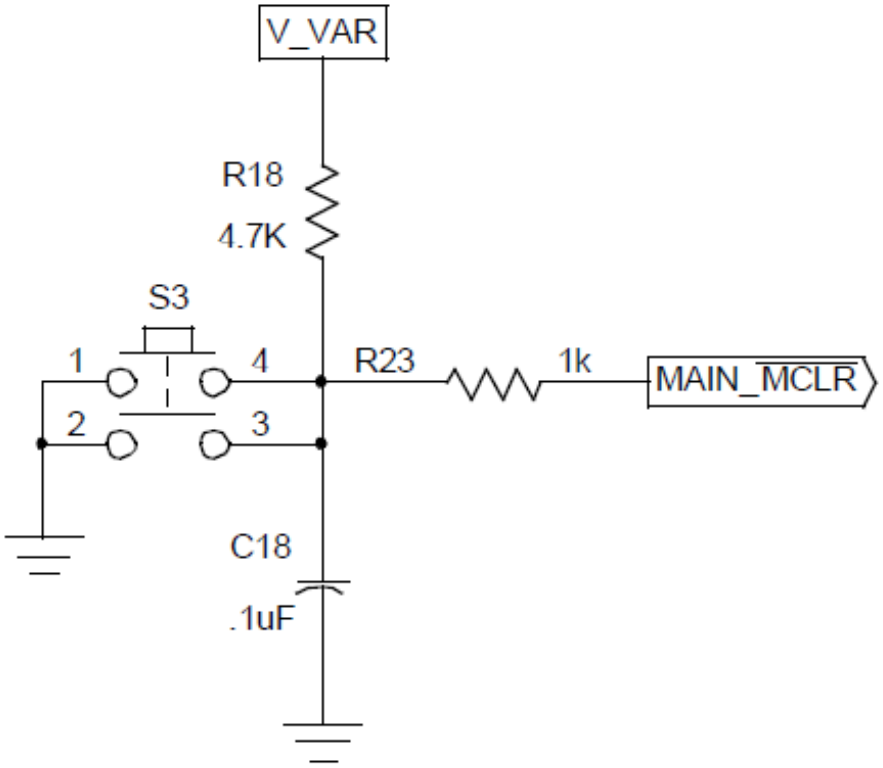
- If you want a general crystal-controlled embedded system to keep accurate time, you can choose to keep the device in an oven (or fridge) at a fixed temperature, and fine-tune the software to keep accurate time → this is impractical
- Temperature compensated crystal Oscillator (TCXO) are available that provide - in an easy to use package - a crystal oscillator and circuitry that compensates for changes in temperature
  - +/- 0.1 ppm – no more than 1 minute every 20 years
  - TCXOs can cost in excess of $100 per unit.
- One practical alternative is to determine the temperature-frequency characteristics for your chosen crystal, and include this information in your application.

Reset Circuits

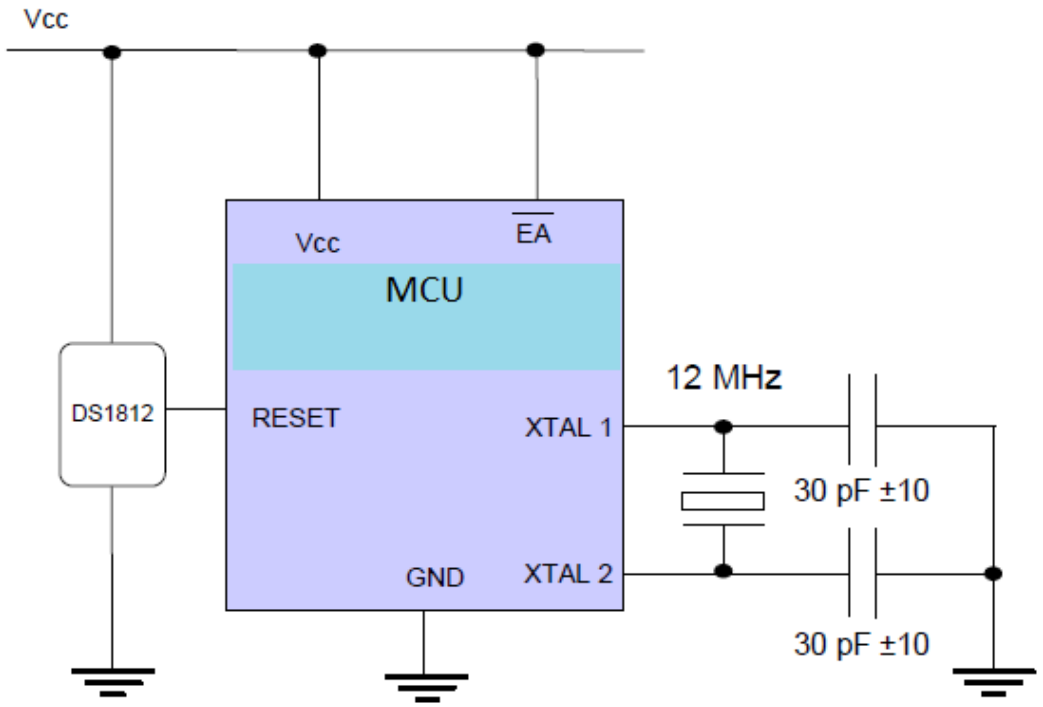
- The process of starting any MCU is a non-trivial one.
- The underlying hardware is complex and a small, manufactured-defined 'reset routine' must be run to place this hardware into an appropriate state before it can begin executing the user program. Running this reset routine takes time and requires that the MCU's oscillator is operating.
- An RC reset circuit is usually the simplest way of controlling the reset behavior.

Reset Circuits





More Robust Reset Circuit



- CO3009 -

- Multi-State Systems and Function Sequences -

Introduction

- Two broad categories of multi-state systems

- Multi-State (Timed)
  - The transition between states will depend only on the passage of time

- Multi-State (Input/Timed)
  - More common form of system
  - Transition between states and behavior in each state will depend both on the passage of time and on system inputs

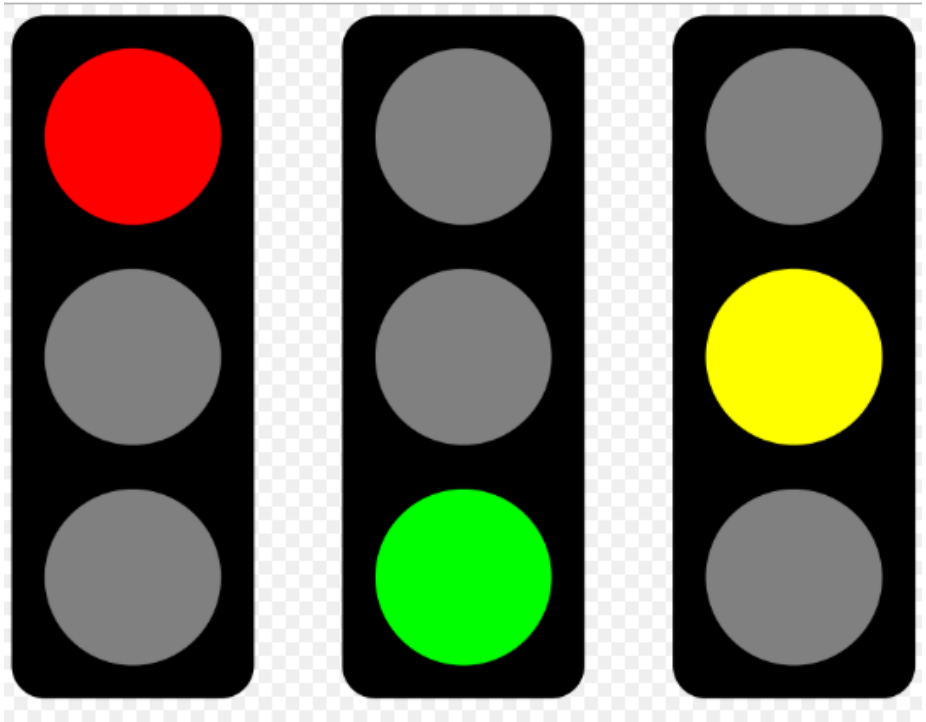
Multi-state (Timed) Systems

- The system will operate in two or more states.
- Each state may be associated with one or more function calls.
- Transitions between states will be controlled by the passage of time.
- Transitions between states may also involve function calls.

- Please note that, in order to ease subsequent maintenance tasks, the system states should not be arbitrarily named, but should – where possible – reflect a physical state observable by the user and /or developer.

- Please also note that the system states will usually be represented by means of a switch statement

Traffic Light



Traffic Light

- The various states are easily identified
  - RED
  - GREEN
  - AMBER

- In the code, these states can be represented as follows:

```
/* Possible system state */
typedef enum {RED, GREEN, AMBER} eLightState
```


Traffic Light

- The time to be spent in each state will be stored as follows:

```
/* Times are in second*/
#define RED_DURATION          20
#define GREEN_DURATION        10
#define YELLOW_DURATION       3
```


Traffic Light

- In this simple case, we do not require function calls from or between system states.

- The required behavior will be implemented directly through control of the three port pins which - in the final system – would be connected to appropriate bulbs.

Main.c

```
void main(void){
    SYSTEM_Initialize();
    INTERRUPT_GlobalInterruptEnable();
    INTERRUPT_PeripheralInterruptEnable();
   // Prepare to run traffic sequence
   Traffic_Light_Init(RED);
   //Set up 50ms ticks
   Init_Timer0(50);
   while(1) {  //Supper loop
          Go_To_Sleep();
    }
}
```


Traffic_Light.h

```
#ifndef _T_LIGHT_H
#define _T_LIGHT_H
//Possible system states
typedef enum {RED, GREEN, AMBER } eLightState;

//public function prototype
void Traffic_Light_Init(const eLightState);
void Traffic_Light_Update(void);

#endif
```


Traffic_Light.c

- Working in group of four students
- What if the Traffic_Light_Update function is called in timer interrupt service routine of 50 ms.
- What if the Traffic_Light_Update function is called in main function.

TRAFFIC_LIGHTS_Update()
Must be called once per second.

```
void TRAFFIC_LIGHTS_Update(void){
    static tWord Time_in_state;
    switch (Light_state_G){
        case Red:
                Red_light = ON;
                Amber_light = OFF;
                Green_light = OFF;
                    if (++Time_in_state == RED_DURATION){
                        Light_state_G = Green;
                        Time_in_state = 0;
                     }
                 break;
```

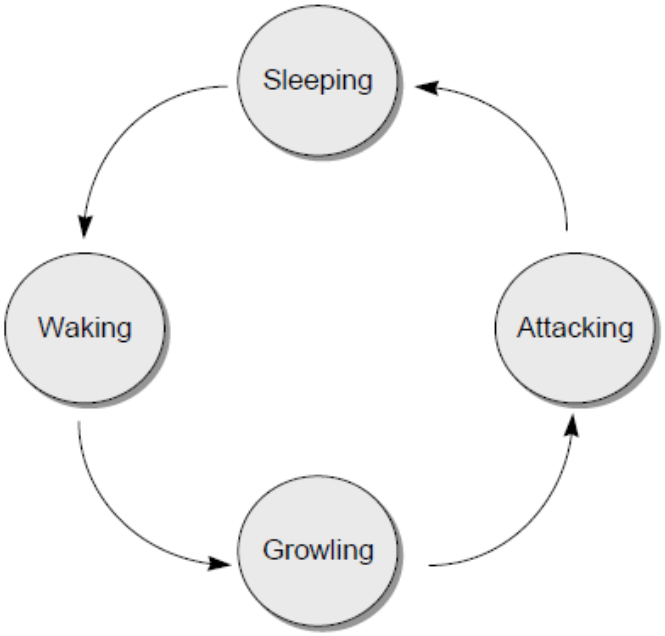

TRAFFIC_LIGHTS_Update()
Must be called once per second.

```
case Green:
    Red_light = OFF;
    Amber_light = OFF;
    Green_light = ON;
     if (++Time_in_state == GREEN_DURATION){
        Light_state_G = Amber;
        Time_in_state = 0;
    }
    break;
```


TRAFFIC_LIGHTS_Update()
Must be called once per second.

```
case Amber:
    Red_light = OFF;
    Amber_light = ON;
    Green_light = OFF;
     if (++Time_in_state == AMBER_DURATION){
        Light_state_G = Red;
        Time_in_state = 0;
    }
    break;
```


Example: Animatronic dinosaur



The System States

- **Sleeping**
  - The dinosaur will be largely motionless, but will be obviously “breathing”. Irregular snoring noises, or slight movement during this time will add interest for the audience.
- **Waking**
  - The dinosaur will begin to wake up. Eyelids will begin to flicker. Breathing will become more rapid
- **Growling**
  - Eyes will suddently open, and the dinosaur will emit a very loud growl. Some further movement and growling will follow.
- **Attacking**
  - Rapid ‘random’ movements towards the audience. Lots of noise (you should be able to hear this from the next floor in the museum)

A Multi-State (Input/Timed) System

- The system will operate in two or more states
- Each states may be associated with one or more function calls
- Transitions between states may be controlled by the passage of time, by system inputs or a combination of time and inputs
- Transition between states may also involve function calls

Implementing State Timeouts

- Consider the following informal system requirements
  - The pump should be run for 10 seconds.
  - If, during this time, no liquid is detected in the outflow tank, the pump should be switched off and ‘low water’ warning should be sounded.
  - If liquid is detected, the pump should be run for a further 45 seconds, or until the ‘high water’ sensor is activated (whichever is first)

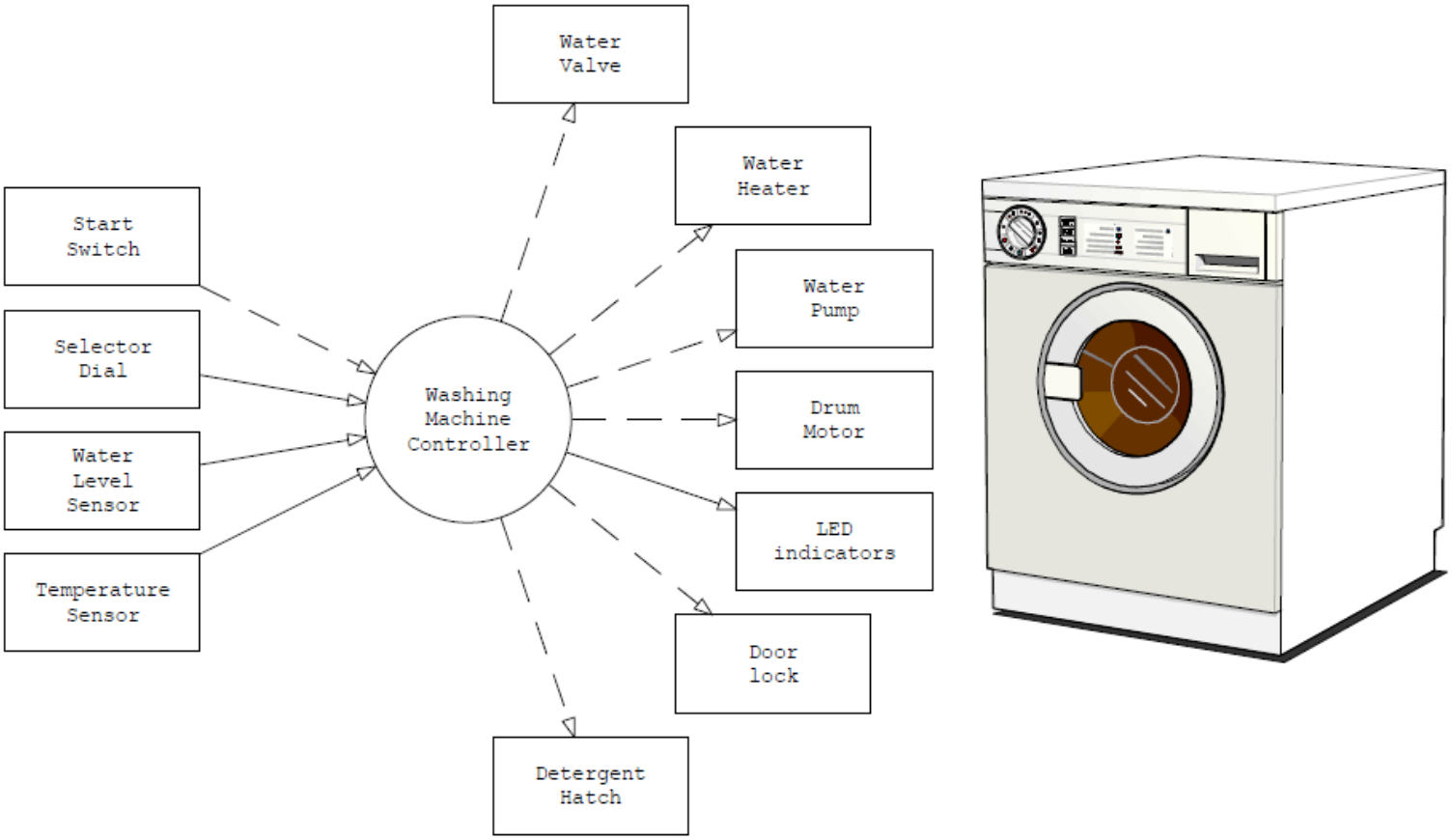
Implementing State Timeouts

- After the front door is opened, the correct password must be entered on the control panel within 30 seconds or the alarm will sound.

Implementing State Timeouts

- To meet this type of requirements, we can do two things
  - Keep track of the time in each system state;
  - If the time exceeds a predetermined error value, then the system should move to a different state

Example: Washing Machine Controller



System Operations

1. The user selects a wash program (e.g. ‘Wool’, ‘Cotton’) on the selector dial.
2. The user presses the ‘Start’ button.
3. The door lock is engaged.
4. The water valve is opened to allow water into the wash drum.
5. If the wash program involves detergent, the detergent hatch is opened. When the detergent has been released, the detergent hatch is closed.
6. When the ‘full water level’ is sensed, the water valve is closed.
7. If the wash program involves warm water, the water heater is switched on. When the water reaches the correct temperature, the water heater is switched off.
8. The washer motor is turned on to rotate the drum. The motor then goes through a series of movements, both farward and reverse (at verious speeds) to wash the clothes. The precise set of movements carried out depends on the wash program the user has selected.) At the end of the wash cycle, the motor is stopped.
9. The pump is switched on to drain the drum. When the drum is empty, the pump is swithed off.

Washer_Update()

```
void WASHER_Update(void) {
    static tWord Time_in_state;
   switch (System_state_G) {
       case START:
           // Lock the door
             WASHER_Control_Door_Lock(ON);
           // Start filling the drum
             WASHER_Control_Water_Valve(ON);
             // Release the detergent (if any)
          if (Detergent_G[Program_G] == 1) {
                 WASHER_Control_Detergent_Hatch(ON);
          }
             // Ready to go to next state
          System_state_G = FILL_DRUM;
          Time_in_state_G = 0;
     break;
```


```
case FILL_DRUM: // Remain in this state until drum is full
    // NOTE: Timeout facility included here
    if (++Time_in_state_G>= MAX_FILL_DURATION){// Should have filled the drum by now
        System_state_G = ERROR;
    }
    // Check the water level
     if (WASHER_Read_Water_Level() == 1) {// Drum is full
         // Does the program require hot water?
         if (Hot_Water_G[Program_G] == 1) {
             WASHER_Control_Water_Heater(ON);
              // Ready to go to next state
             System_state_G = HEAT_WATER;
             Time_in_state_G = 0;
        }  else {  // Using cold water only
             // Ready to go to next state
              System_state_G = WASH_01;
              Time_in_state_G = 0;
         }
     }
break;
```


```
case HEAT_WATER:
        // Remain in this state until water is hot
        // NOTE: Timeout facility included here
        if (++Time_in_state_G >= MAX_WATER_HEAT_DURATION) {
        // Should have warmed the water by now...
                System_state_G = ERROR;
        }
        // Check the water temperature
        if (WASHER_Read_Water_Temperature() == 1) {
        // Water is at required temperature
        // Ready to go to next state
                System_state_G = WASH_01;
                Time_in_state_G = 0;
        }
break;
```


```
            case WASH_01:
                // All wash program involve WASH_01
                // Drum is slowly rotated to ensure clothes are fully wet
                WASHER_Control_Motor(ON);
                if (++Time_in_state >= WASH_01_DURATION) {
                        System_state_G = WASH_02;
                        Time_in_state = 0;
                }
            break;
            // REMAINING WASH PHASES OMITTED HERE ...
            case WASH_02:
            break;
            case ERROR:
            break;
        }
}
```


- CO3009 -

- UART, SPI & I2C -

UART - Universal Asynchronous Receiver Transmitter

- Universal Asynchronous Receive Transmit (UART) or Serial communication is one of the most simple communication protocols between two devices.
- It transfers data between devices by connecting two wires between the devices, one is the transmission line while the other is the receiving line.
- The data transfers bit by bit digitally in form of bits from one device to another.
- The main advantage of this communication protocol is that its not necessary for both the devices to have the same operating frequency.
  - For example, two microcontrollers operating at different clock frequencies can communicate with each other easily via serial communication.
  - However, a predefined bit rate that is referred to as baud rate usually set in the flash memory of both microcontrollers for the instruction to be understood by both the devices.

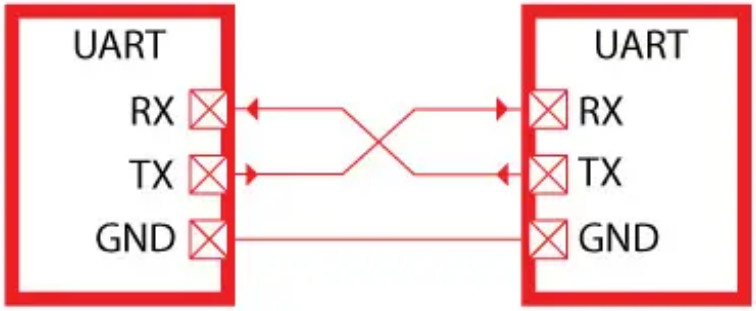
Transmitting and Receiving Serial Data

- The transmitting UART takes bytes of data and transmits the bits in a sequential form.

- Communication between two UART devices may be simplex, full-duplex or half-duplex.
  - Simplex communication is a one-direction type of communication where the signal moves from one UART to another. It doesn't have provision for the receiving UART to send back signals.
  - A full-duplex is when both devices can transmit and receive communications at the same time.
  - A half-duplex is when devices take turns to transmit and receive.

How Two Devices Communicate through UART

- It takes two UART's to communicate directly with each other.
- On one end the transmitting UART converts parallel data from a CPU into serial form then transmits the data in serial form to the second UART which will receive the serial data and convert it back into parallel data.
- This data can then be accessed from the receiving device.



UART in STM32

- Set up UART with receiving and transmitting interrupt mode
- UART Initialization

```
37 void UART1_Init(void){
38     Uart1Handle.Instance          = USART1;
39
40     Uart1Handle.Init.BaudRate     = 115200;
41     Uart1Handle.Init.WordLength   = UART_WORDLENGTH_8B;
42     Uart1Handle.Init.StopBits     = UART_STOPBITS_1;
43     Uart1Handle.Init.Parity       = UART_PARITY_NONE;
44     Uart1Handle.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
45     Uart1Handle.Init.Mode         = UART_MODE_TX_RX;
46
47     if(HAL_UART_DeInit(&Uart1Handle) != HAL_OK){
48         Error_Handler();
49     }
50     if(HAL_UART_Init(&Uart1Handle) != HAL_OK){
51         Error_Handler();
52     }
53     if(HAL_UART_Receive_IT(&Uart1Handle, (uint8_t *)aUART_RxBuffer, RXBUFFERSIZE) != HAL_OK){
54         return HAL_ERROR;
55     }
56     return HAL_OK;
57 }
```


UART PIN and Interrupt Initialization

```
51 void HAL_UART_MspInit(UART_HandleTypeDef *huart)
52 {
53     GPIO_InitTypeDef  GPIO_InitStruct;
54     if(huart->Instance == USART1){
55         /*##-1- Enable peripherals and GPIO Clocks #################################*/
56         /* Enable GPIO TX/RX clock */
57         USART1_TX_GPIO_CLK_ENABLE();
58         USART1_RX_GPIO_CLK_ENABLE();
59         /* Enable USARTx clock */
60         USART1_CLK_ENABLE();
61 
62         /*##-2- Configure peripheral GPIO ##########################################*/
63         /* UART TX GPIO pin configuration  */
64         GPIO_InitStruct.Pin       = USART1_TX_PIN;
65         GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
66         GPIO_InitStruct.Pull      = GPIO_PULLUP;
67         GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_HIGH;
68 
69         HAL_GPIO_Init(USART1_TX_GPIO_PORT, &GPIO_InitStruct);
70 
71         /* UART RX GPIO pin configuration  */
72         GPIO_InitStruct.Pin = USART1_RX_PIN;
73         GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
74         HAL_GPIO_Init(USART1_RX_GPIO_PORT, &GPIO_InitStruct);
75         /*##-3- Configure the NVIC for UART ########################################*/
76         /* NVIC for USART */
77         HAL_NVIC_SetPriority(USART1_IRQn, 0, 1);
78         HAL_NVIC_EnableIRQ(USART1_IRQn);
79     } else if(huart->Instance == USART2){
```


UART Interrupt Service Routine

```c
/* stm32f1xx_it.c */

void USART1_IRQHandler(void)
{
    HAL_UART_IRQHandler(&Uart1Handle);
}
```

```c
void HAL_UART_IRQHandler(UART_HandleTypeDef *huart)
{
  uint32_t isrflags   = READ_REG(huart->Instance->SR);
  uint32_t cr1its     = READ_REG(huart->Instance->CR1);
  uint32_t cr3its     = READ_REG(huart->Instance->CR3);
  uint32_t errorflags = 0x00U;
  uint32_t dmarequest = 0x00U;

  /* If no error occurs */
  errorflags = (isrflags & (uint32_t)(USART_SR_PE | USART_SR_FE | USART_SR_ORE | USART_SR_NE));
  if (errorflags == RESET)
  {
    /* UART in mode Receiver ----------------------------------------------------*/
    if (((isrflags & USART_SR_RXNE) != RESET) && ((cr1its & USART_CR1_RXNEIE) != RESET))
    {
      Custom_UART_Receive_IT(huart);
      return;
    }
  }
```

anhpham@hcmut.edu.vn

UART Transmit Data

```
135 void UART1_Transmit(uint8_t * buffer){
136     uint8_t buffer_len = GetStringLength((uint8_t*)buffer);
137     if(buffer_len == 0) {
138         return;
139     } else {
140         if(HAL_UART_Transmit_IT(&Uart1Handle, (uint8_t*)buffer, buffer_len)!= HAL_OK){
141             Error_Handler();
142         }
143         UartTransmitReady = RESET;
144     }
145     return;
146 }
11 uint8_t GetStringLength(uint8_t* buffer){
12     uint8_t len = 0;
13     if(*buffer == 0){
14         return 0;
15     }
16     while(*buffer != 0){
17         len ++;
18         if(*buffer == '\n') break;
19         buffer ++;
20 
21     }
22     return len;
23 }
```


UART Receive Data

```
162 HAL_StatusTypeDef Custom_UART_Receive_IT(UART_HandleTypeDef *huart)
163 {
164   /* Check that a Rx process is ongoing */
165   if (huart->RxState == HAL_UART_STATE_BUSY_RX)
166   {
167       huart->ErrorCode = HAL_UART_ERROR_NONE;
168       huart->RxState = HAL_UART_STATE_BUSY_RX;
169       aUART_RxBuffer[receiveBufferIndexHead] = (uint8_t)(huart->Instance->DR & (uint8_t)0x00FF);
170       receiveBufferIndexHead = (receiveBufferIndexHead + 1) % RXBUFFERSIZE;
171       return HAL_OK;
172   } else {
173     return HAL_BUSY;
174   }
175 }
177 uint8_t Uart1_Received_Buffer_Available(void){
178     if(receiveBufferIndexTail != receiveBufferIndexHead){
179         return 1;
180     } else {
181         return 0;
182     }
183 }
184 uint8_t Uart1_Read_Received_Buffer(void){
185     uint8_t buffer[2];
186     if(receiveBufferIndexTail == receiveBufferIndexHead) return 0xff;
187     uint8_t ch = aUART_RxBuffer[receiveBufferIndexTail];
188     receiveBufferIndexTail = (receiveBufferIndexTail + 1) % RXBUFFERSIZE;
189     return ch;
190 }
```


Processing UART Received Data

```
593 void FSM_Process_Data_Received_From_Sim3g(void){
594     static uint8_t readCharacter = 0;
595     static uint8_t preReadCharacter = 0;
596 
597     switch(processDataState){
598     case CHECK_DATA_AVAILABLE_STATE:
599         if(Uart1_Received_Buffer_Available()){
600             Clear_Sim3gDataProcessingBuffer();
601             processDataState = DETECT_SPECIAL_CHARACTER;
602             isReceiveDataFromServer = SET;
603         }
604         break;
```


Processing UART Received Data

```
605     case DETECT_SPECIAL_CHARACTER:
606         if(Uart1_Received_Buffer_Available()){
607             preReadCharacter = readCharacter;
608             readCharacter = Uart1_Read_Received_Buffer();
609             if(readCharacter == '>'){
610                 processDataState = PREPARE_FOR_SENDING_DATA;
611             }else if(preReadCharacter == '\r' && readCharacter == '\n'){
612                 processDataState = PREPARE_PROCESSING_RECEIVED_DATA;
613             } else if((preReadCharacter == SUBSCRIBE_RECEIVE_MESSAGE_TYPE)
614                     && (readCharacter == LEN_SUBSCRIBE_RECEIVE_MESSAGE_TYPE1
615                     || readCharacter == LEN_SUBSCRIBE_RECEIVE_MESSAGE_TYPE2)){
616                 processDataState = PROCESSING_RECEIVED_DATA;
617                 Clear_Sim3gDataProcessingBuffer();
618             }
619             else if((preReadCharacter == SUBSCRIBE_RECEIVE_RETAINED_MESSAGE_TYPE)
620                     && (readCharacter == LEN_SUBSCRIBE_RECEIVE_RETAINED_MESSAGE_TYPE
621                 processDataState = PROCESSING_RETAINED_DATA;
622                 Clear_Sim3gDataProcessingBuffer();
623             }
624             else {
625                 Sim3gDataProcessingBuffer[sim3gDataProcessingBufferIndex++] = readCh
626             }
627         }
628         break;
```


Processing UART Received Data

```
case PREPARE_FOR_SENDING_DATA:
    isReadyToSendDataToServer = SET;
    processDataState = CHECK_DATA_AVAILABLE_STATE;
    break;
case PREPARE_PROCESSING_RECEIVED_DATA:
    if(isReceivedData((uint8_t *)PB_DONE)){
        isPBDoneFlag = SET;
    } else if(isReceivedData((uint8_t *)STIN25)){
        isStin25 = SET;
    } else if(isReceivedData((uint8_t *)OK)){
        isOKFlag = SET;
    } else if(isReceivedData((uint8_t *)ERROR_1)){
        isErrorFlag = SET;
        UART3_SendToHost((uint8_t*)"aNg");
    } else if(isReceivedData((uint8_t *)IP_CLOSE)){
        isIPCloseFlag = SET;
    } else if(isReceivedData((uint8_t *)RECV_FROM)){
        isSendOKFlag = RESET;
        isRecvFromFlag = SET;
    } else if(isReceivedData((uint8_t *)Send_ok)){
        isSendOKFlag = SET;
    } else if(isReceivedData((uint8_t *)ISCIPSEND)){
        isCipsend = SET;
    }
    processDataState = CHECK_DATA_AVAILABLE_STATE;
    break;
```


Processing UART Received Data

```
655     case PROCESSING_RECEIVED_DATA:
656         if(Uart1_Received_Buffer_Available()){
657             preReadCharacter = readCharacter;
658             readCharacter = Uart1_Read_Received_Buffer();
659             if((preReadCharacter == '\r' && readCharacter == '\n')){
660                 Processing_Received_Data((uint8_t*)SUBSCRIBE_TOPIC_1, Get_Box_ID());
661                 processDataState = CHECK_DATA_AVAILABLE_STATE;
662             } else if(preReadCharacter == SUBSCRIBE_RECEIVE_MESSAGE_TYPE
663                     && (readCharacter == LEN_SUBSCRIBE_RECEIVE_MESSAGE_TYPE1
664                     || readCharacter == LEN_SUBSCRIBE_RECEIVE_MESSAGE_TYPE2)){
665                 processDataState = PROCESSING_RECEIVED_DATA;
666                 Processing_Received_Data((uint8_t*)SUBSCRIBE_TOPIC_1,  Get_Box_ID());
667                 Clear_Sim3gDataProcessingBuffer();
668             } else {
669                 Sim3gDataProcessingBuffer[sim3gDataProcessingBufferIndex++] = readCharac
670                 if(sim3gDataProcessingBufferIndex >= DATA_RECEIVE_LENGTH){
671                     Processing_Received_Data((uint8_t*)SUBSCRIBE_TOPIC_1, Get_Box_ID());
672                     processDataState = CHECK_DATA_AVAILABLE_STATE;
673                 }
674             }
675         }
676         break;
```

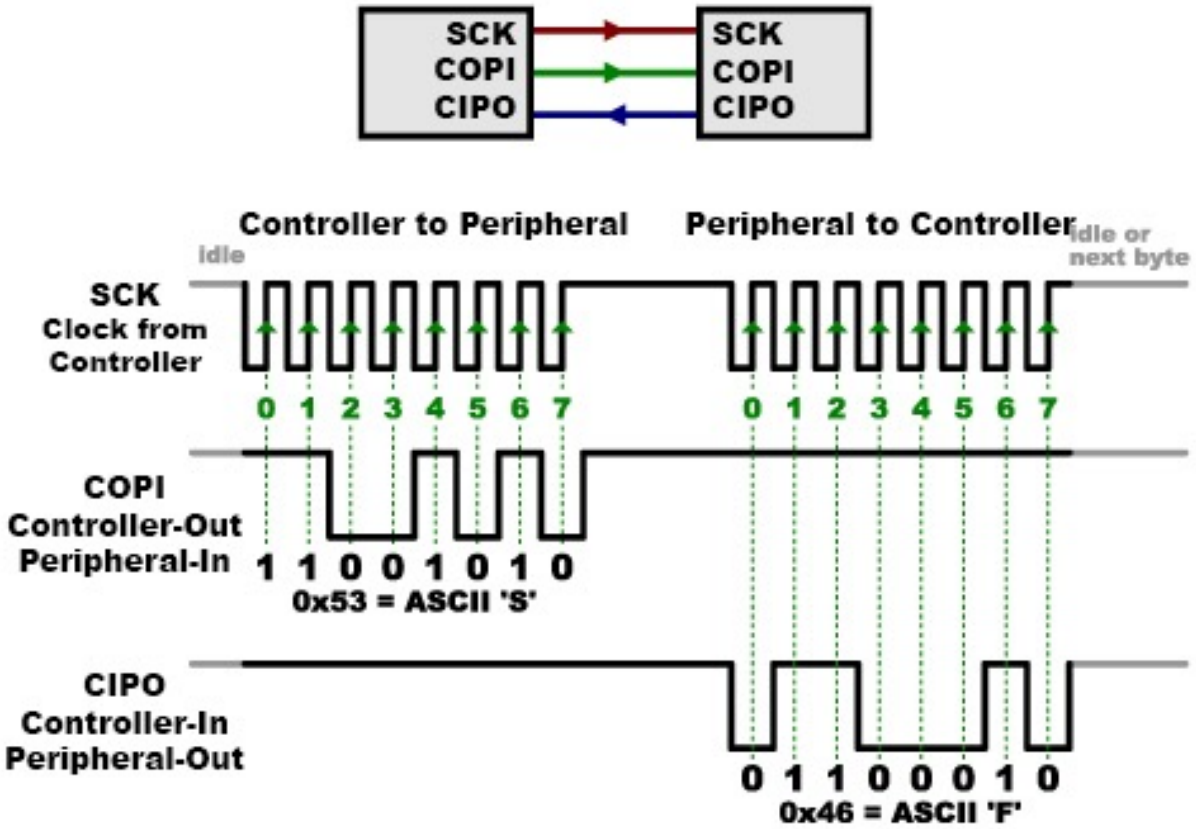

Processing UART Received Data

```
677     case PROCESSING_RETAINED_DATA:
678             if(Uart1_Received_Buffer_Available()){
679                 preReadCharacter = readCharacter;
680                 readCharacter = Uart1_Read_Received_Buffer();
681                 if((preReadCharacter == '\r' && readCharacter == '\n')){
682                     processDataState = CHECK_DATA_AVAILABLE_STATE;
683                 }
684                 else {
685                     Sim3gDataProcessingBuffer[sim3gDataProcessingBufferIndex++] = readChara
686                     if(sim3gDataProcessingBufferIndex >= DATA_RETAINED_MESSAGE_LENGTH){
687                         Processing_Received_Data_From_Retained_Message((uint8_t*)SUBSCRIBE_
688                         processDataState = CHECK_DATA_AVAILABLE_STATE;
689                     }
690                 }
691             }
692             break;
693     default:
694         break;
695     }
```

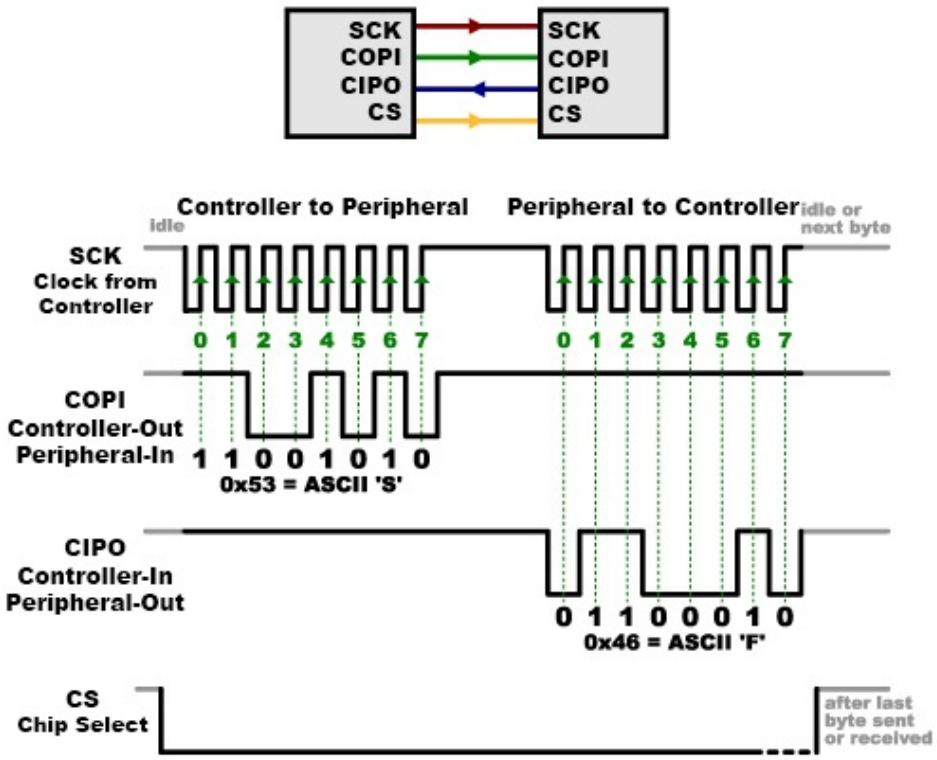

SPI - Serial Peripheral Interface

- Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards.
- It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.
- In SPI, only one side generates the clock signal (usually called CLK or SCK for Serial Clock).
- The side that generates the clock is called the "controller", and the other side is called the "peripheral".
- There is always only one controller (which is almost always your microcontroller), but there can be multiple peripherals

SPI in STM32 and IC 25LC512

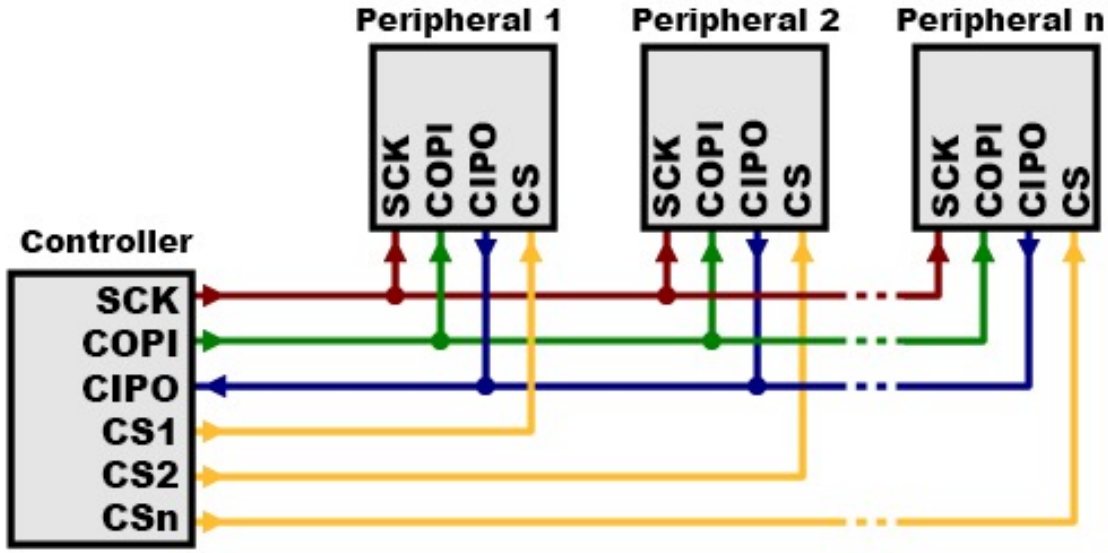


Chip Select (CS)



- There's one last line you should be aware of, called CS for Chip Select.
- This tells the peripheral that it should wake up and receive / send data and is also used when multiple peripherals are present to select the one you'd like to talk to.

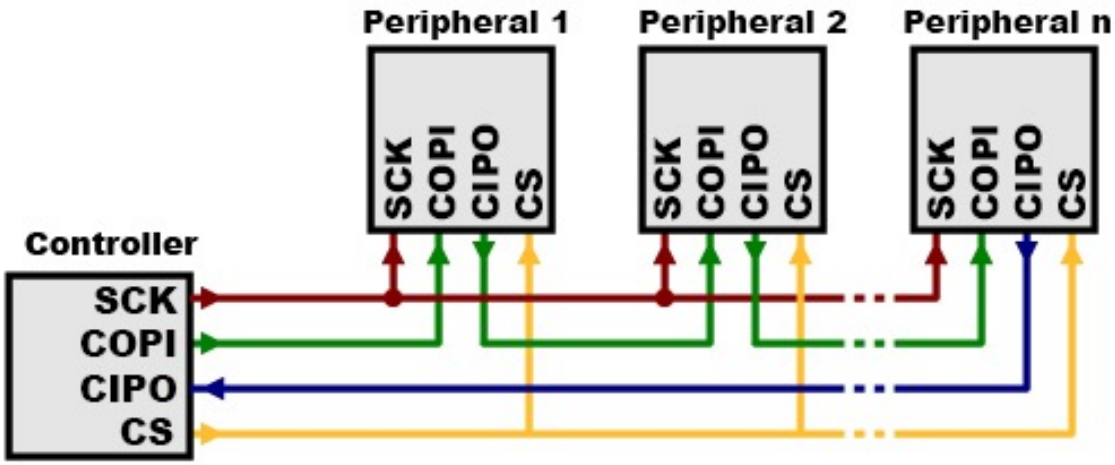
Multiple Peripherals with Multiple CS



- In general, each peripheral will need a separate CS line. To talk to a particular peripheral, you'll make that peripheral's CS line low and keep the rest of them high.

- Lots of peripherals will require lots of CS lines; if you're running low on outputs, there are binary decoder chips that can multiply your CS outputs.

Multiple Peripherals with Single CS



- On the other hand, some parts prefer to be daisy-chained together, with the CIPO (output) of one going to the COPI (input) of the next.
- In this case, a single CS line goes to all the peripherals.
- Once all the data is sent, the CS line is raised, which causes all the chips to be activated simultaneously.
- This is often used for daisy-chained shift registers and addressable LED drivers.

SPI in STM32 and IC 25LC512

```
24 void SPI2_Init(void){
25     /*##-1- Configure the SPI peripheral #######################################*/
26           /* Set the SPI parameters */
27           Spi2Handle.Instance               = SPI2;
28           Spi2Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
29           Spi2Handle.Init.Direction         = SPI_DIRECTION_2LINES;
30           Spi2Handle.Init.CLKPhase          = SPI_PHASE_1EDGE;
31           Spi2Handle.Init.CLKPolarity       = SPI_POLARITY_LOW;
32           Spi2Handle.Init.DataSize          = SPI_DATASIZE_8BIT;
33           Spi2Handle.Init.FirstBit          = SPI_FIRSTBIT_MSB;
34           Spi2Handle.Init.TIMode            = SPI_TIMODE_DISABLE;
35           Spi2Handle.Init.CRCCalculation    = SPI_CRCCALCULATION_DISABLE;
36           Spi2Handle.Init.CRCPolynomial     = 7;
37 //        Spi2Handle.Init.NSS               = SPI_NSS_SOFT;
38           Spi2Handle.Init.NSS               = SPI_NSS_HARD_OUTPUT;
39
40           Spi2Handle.Init.Mode = SPI_MODE_MASTER;
41           if(HAL_SPI_Init(&Spi2Handle) != HAL_OK)
42           {
43             /* Initialization Error */
44             Error_Handler();
45           }
46           __HAL_SPI_ENABLE(&Spi2Handle);
47 }
```


PIN Intialization

```
313 void HAL_SPI_MspInit(SPI_HandleTypeDef *hspi)
314 {
315 GPIO_InitTypeDef  GPIO_InitStruct;
316   if(hspi->Instance == SPI2){
317     /*##-1- Enable peripherals and GPIO Clocks #################################*/
318     /* Enable GPIO TX/RX clock */
319     SPI2_SCK_GPIO_CLK_ENABLE();
320     SPI2_MISO_GPIO_CLK_ENABLE();
321     SPI2_MOSI_GPIO_CLK_ENABLE();
322     /* Enable SPI clock */
323     SPI2_CLK_ENABLE();
324     /*##-2- Configure peripheral GPIO ##########################################*/
325     /* SPI SCK GPIO pin configuration  */
326     GPIO_InitStruct.Pin       = SPI2_SCK_PIN;
327     GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
328     GPIO_InitStruct.Pull      = GPIO_PULLDOWN;
329     GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_LOW;
330     HAL_GPIO_Init(SPI2_SCK_GPIO_PORT, &GPIO_InitStruct);
331     /* SPI MISO GPIO pin configuration  */
332     GPIO_InitStruct.Pin = SPI2_MISO_PIN;
333     HAL_GPIO_Init(SPI2_MISO_GPIO_PORT, &GPIO_InitStruct);
334     /* SPI MOSI GPIO pin configuration  */
335     GPIO_InitStruct.Pin = SPI2_MOSI_PIN;
336     HAL_GPIO_Init(SPI2_MOSI_GPIO_PORT, &GPIO_InitStruct);
337     /* SPI NSS GPIO pin configuration  */
338     GPIO_InitStruct.Pin = SPI2_NSS_PIN;
339     HAL_GPIO_Init(SPI2_NSS_GPIO_PORT, &GPIO_InitStruct);
340   }
341 }
```


Preparation

```
49 void ResetChipSelect(void){
50     HAL_GPIO_WritePin(SPI2_NSS_GPIO_PORT,SPI2_NSS_PIN, GPIO_PIN_RESET);
51 }
52
53 void SetChipSelect(void){
54     HAL_GPIO_WritePin(SPI2_NSS_GPIO_PORT,SPI2_NSS_PIN, GPIO_PIN_SET);
55 }
24 void MC25LC512_CS(uint8_t CS_Status)
25 {
26     // For Cs of the EEprom
27     if(CS_Status == EEPROM_CS_PIN_RESET){
28         ResetChipSelect();
29     } else {
30         SetChipSelect();
31     }
32 }
33 void MC25LC512_WriteEnableOrDisable(uint8_t EnableOrDisable)
34 {
35     uint8_t SendOneByte = 0;
36     MC25LC512_CS(EEPROM_CS_PIN_RESET);// Reset The spi Chip //Reset means Enable
37     for(uint16_t i = 0; i < 10; i ++);
38     if(EnableOrDisable==EEPROM_Enable)  {
39             SendOneByte = MC25LCxxx_SPI_WREN;
40     } else {
41             SendOneByte = MC25LCxxx_SPI_WRDI;
42     }
43     HAL_SPI_Transmit(&Spi2Handle , &SendOneByte, 1, 200) ;
44     for(uint16_t i = 0; i < 10; i ++);
45     MC25LC512_CS(EEPROM_CS_PIN_SET);// Set The spi Chip //Set means Disable
46 }
```


Release Deep Power Mode IC 25LC512

```
49 uint8_t MC25LC512_ReleaseDeepPowerDownMode(void)
50 {
51     uint8_t SendOneByte;
52     uint8_t RecieveByteOfReleaseDeepPowerMode = 0;
53     SendOneByte = MC25LCxxx_SPI_RDID;
54     MC25LC512_CS(EEPROM_CS_PIN_RESET);// Reset The spi Chip //Reset means Enab
55     for(uint16_t i = 0; i < 1000; i ++);
56     HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);
57     HAL_SPI_Receive(&Spi2Handle , &RecieveByteOfReleaseDeepPowerMode, 1,200) ;
58     HAL_SPI_Receive(&Spi2Handle , &RecieveByteOfReleaseDeepPowerMode, 1,200) ;
59     HAL_SPI_Receive(&Spi2Handle , &RecieveByteOfReleaseDeepPowerMode, 1,200) ;
60     for(uint16_t i = 0; i < 1000; i ++);
61     MC25LC512_CS(EEPROM_CS_PIN_SET);// Set The spi Chip //Set means Disable
62     return RecieveByteOfReleaseDeepPowerMode;
63
64 }
```


25LC512 Initialization

```
66 void MC25LC512_Initialize(void)
67 {
68     MC25LC512_CS(EEPROM_CS_PIN_SET);// Reset The spi Chip //Reset means Enable
69     MC25LC512_ReleaseDeepPowerDownMode();
70     MC25LC512_ReadStatusRegister();
71     MC25LC512_WriteEnableOrDisable(EEPROM_Enable);
72 }
```


Write a Buffer of Bytes to 25LC512

```
73  void MC25LC512_Write_Bytes(uint16_t AddresOfData, uint8_t *WriteArrayOfEEProm, uint16_t SizeOfArray){
74      uint8_t SendOneByte;
75      MC25LC512_WriteEnableOrDisable(EEPROM_Enable);
76      for(uint16_t i = 0; i < TIME_DELAY; i ++);
77      MC25LC512_CS(EEPROM_CS_PIN_SET);
78      MC25LC512_CS(EEPROM_CS_PIN_RESET);// Reset The spi Chip //Reset means Enable
79      for(uint16_t i = 0; i < TIME_DELAY; i ++);
80      SendOneByte = MC25LCxxx_SPI_WRITE;
81      HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);
82      SendOneByte = AddresOfData>>8;
83      HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);//High byte of address
84      SendOneByte = AddresOfData & 0x00FF;
85      HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);//Low byte of address
86      HAL_SPI_Transmit(&Spi2Handle, WriteArrayOfEEProm, SizeOfArray, SizeOfArray*50) ;
87      for(uint16_t i = 0; i < TIME_DELAY; i ++);
88      MC25LC512_CS(EEPROM_CS_PIN_RESET);
89      MC25LC512_CS(EEPROM_CS_PIN_SET);// Reset The spi Chip //Reset means Enable
90      for(uint16_t i = 0; i < 10000; i ++);
91      for(uint16_t i = 0; i < 10000; i ++);
92  }
```

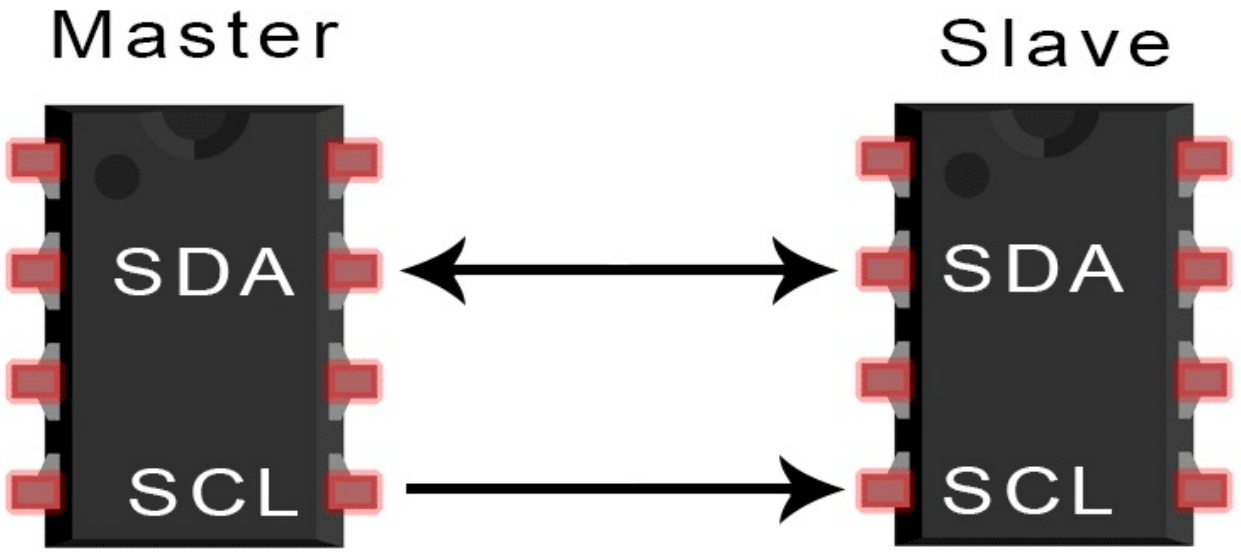

Read a Buffer of Byte From 25LC512

```
160 uint8_t MC25LC512_Read_Bytes(uint16_t AddresOfData, uint8_t *DataArrayOfEEProm, uint16_t SizeOfArray){
161     uint8_t SendOneByte;
162     MC25LC512_CS(EEPROM_CS_PIN_RESET);// Reset The spi Chip //Reset means Enable
163     for(uint16_t i = 0; i < TIME_DELAY; i ++);
164     SendOneByte = MC25LCxxx_SPI_READ;//Config the Device
165     HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);
166
167     SendOneByte= AddresOfData >> 8;
168     HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);//High byte of address
169     SendOneByte= AddresOfData & 0x00FF;
170     HAL_SPI_Transmit(&Spi2Handle, &SendOneByte, 1, 200);//Low byte of address
171
172     HAL_SPI_Receive(&Spi2Handle, DataArrayOfEEProm, SizeOfArray, SizeOfArray*30) ;//Receive Amount of  Data
173     for(uint16_t i = 0; i < TIME_DELAY; i ++);
174     MC25LC512_CS(EEPROM_CS_PIN_SET);// Reset The spi Chip //Reset means Enable
175     return 0;
176 }
```


I2C- Inter Interconect Communication

- I2C combines the best features of SPI and UARTs.
- With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves.
- This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

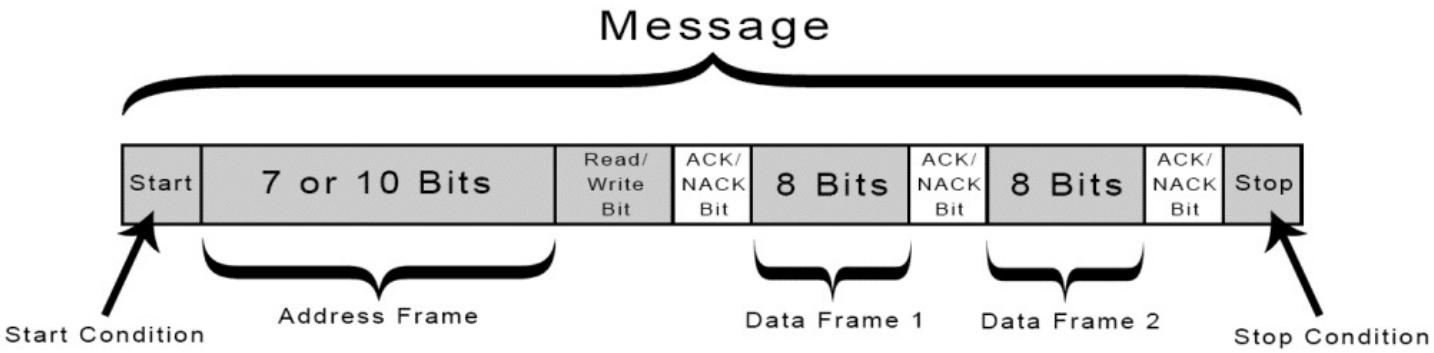
I2C- Inter Interconect Communication



- SDA (Serial Data) – The line for the master and slave to send and receive data.
- SCL (Serial Clock) – The line that carries the clock signal.

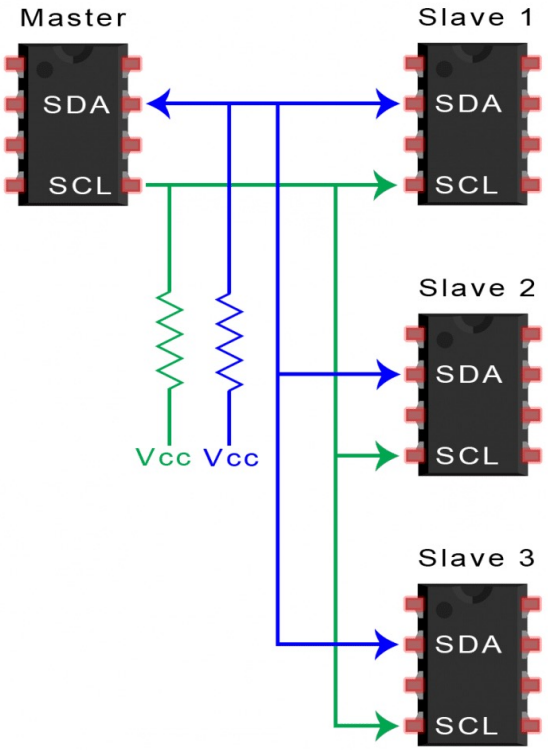
How I2C Works

- With I2C, data is transferred in messages. Messages are broken up into frames of data.
- Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted.
- The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:



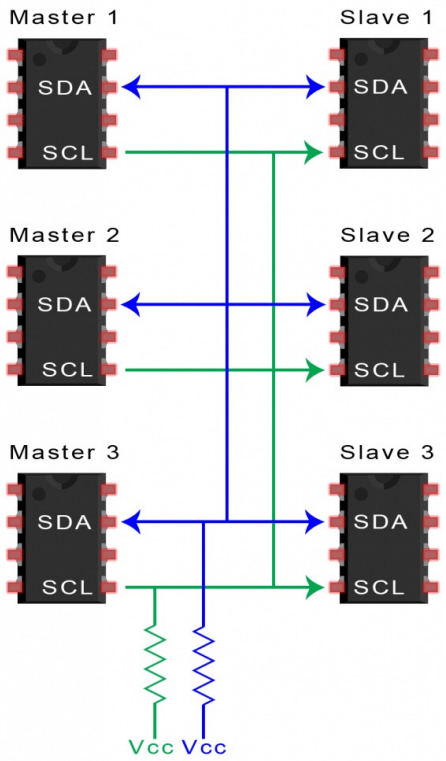
Single Master with Multiple Slaves

- Because I2C uses addressing, multiple slaves can be controlled from a single master.
- With a 7 bit address, 128 (27) unique address are available.
- To connect multiple slaves to a single master, wire them like this, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc.



Multiple Masters with Multiple Slaves

- Multiple masters can be connected to a single slave or multiple slaves.
- The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line.
- To solve this problem, each master needs to detect if the SDA line is low or high before transmitting a message.
- If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message.
- If the SDA line is high, then it's safe to transmit the message.
- To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc.



I2C Initialization

```
23 void I2C_Init(void){
24     /*##-1- Configure the I2C peripheral ######################################*/
25       I2cHandle.Instance             = I2C1;
26       I2cHandle.Init.ClockSpeed      = I2C_SPEEDCLOCK;
27       I2cHandle.Init.DutyCycle       = I2C_DUTYCYCLE;
28       I2cHandle.Init.OwnAddress1     = I2C_ADDRESS;
29       I2cHandle.Init.AddressingMode  = I2C_ADDRESSINGMODE_7BIT;
30       I2cHandle.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
31       I2cHandle.Init.OwnAddress2     = 0xFF;
32       I2cHandle.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
33       I2cHandle.Init.NoStretchMode   = I2C_NOSTRETCH_DISABLE;
34
35       if(HAL_I2C_Init(&I2cHandle) != HAL_OK)
36       {
37         /* Initialization Error */
38         Error_Handler();
39       }
40 }
41
```


PIN Initialization

```
376 void HAL_I2C_MspInit(I2C_HandleTypeDef *hi2c)
377 {
378   GPIO_InitTypeDef  GPIO_InitStruct;
379   /*##-1- Enable peripherals and GPIO Clocks #################################*/
380   /* Enable GPIO TX/RX clock */
381   I2C1_SCL_GPIO_CLK_ENABLE();
382   I2C1_SDA_GPIO_CLK_ENABLE();
383   /* Enable I2Cx clock */
384   I2C1_CLK_ENABLE();
385   /*##-2- Configure peripheral GPIO ##########################################*/
386   /* I2C TX GPIO pin configuration  */
387   GPIO_InitStruct.Pin       = I2C1_SCL_PIN;
388   GPIO_InitStruct.Mode      = GPIO_MODE_AF_OD;
389   GPIO_InitStruct.Pull      = GPIO_PULLUP;
390   GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_HIGH;
391   HAL_GPIO_Init(I2C1_SCL_GPIO_PORT, &GPIO_InitStruct);
392   /* I2C RX GPIO pin configuration  */
393   GPIO_InitStruct.Pin       = I2C1_SDA_PIN;
394   HAL_GPIO_Init(I2C1_SDA_GPIO_PORT, &GPIO_InitStruct);
395 }
```


PCF8574 Set Input Pin

```
146 void Set_Input_PCF_Pins(void){
147     uint8_t initData[1] = {0xff};
148     HAL_I2C_Master_Transmit(&I2cHandle, (uint16_t)PCF_WRITE_ADDRESS_1, (uint8_t *) initData, 1, 0xffff);
149     HAL_Delay(100);
150     HAL_I2C_Master_Transmit(&I2cHandle, (uint16_t)PCF_WRITE_ADDRESS_2, (uint8_t *) initData, 1, 0xffff);
151     HAL_Delay(100);
152     HAL_I2C_Master_Transmit(&I2cHandle, (uint16_t)PCF_WRITE_ADDRESS_3, (uint8_t *) initData, 1, 0xffff);
153     HAL_Delay(100);
154     HAL_I2C_Master_Transmit(&I2cHandle, (uint16_t)PCF_WRITE_ADDRESS_4, (uint8_t *) initData, 1, 0xffff);
155     HAL_Delay(100);
156 }
```


PCF8574

```
99  void PCF_read(void){
100     static uint8_t pcfReadState = 0;
101     switch(pcfReadState){
102     case 0:
103         HAL_I2C_Master_Receive(&I2cHandle, PCF_READ_ADDRESS_1, (uint8_t*) I2CReceiveBuffer, 2, 0xffff);
104         pcfData.bytePCFData[pcfReadState] = I2CReceiveBuffer[0];
105         pcfReadState = 1;
106         break;
107     case 1:
108         HAL_I2C_Master_Receive(&I2cHandle, PCF_READ_ADDRESS_2, (uint8_t*) I2CReceiveBuffer, 2, 0xffff);
109         pcfData.bytePCFData[pcfReadState] = I2CReceiveBuffer[0];
110         pcfReadState = 2;
111         break;
112     case 2:
113         HAL_I2C_Master_Receive(&I2cHandle, PCF_READ_ADDRESS_3, (uint8_t*) I2CReceiveBuffer, 2, 0xffff);
114         pcfData.bytePCFData[pcfReadState] = I2CReceiveBuffer[0];
115         pcfReadState = 3;
116         break;
117     case 3:
118         HAL_I2C_Master_Receive(&I2cHandle, PCF_READ_ADDRESS_4, (uint8_t*) I2CReceiveBuffer, 2, 0xffff);
119         pcfData.bytePCFData[pcfReadState] = I2CReceiveBuffer[0];
120         pcfReadState = 0;
121         break;
122     default:
123         pcfReadState = 0;
124         break;
125     }
126 }
```


- CO3009 -

- How to Choose the Right MCU -

Questions to Answer

- Do you have the needed time, talent, and budget for this project?
- What happens if you cannot find an "ideal" MCU?
- Are you already familiar with an MCU family and associated hardware and software tools?
- Does your company dictate approved vendors? You might start with a limited group of MCU choices.
- What are your product restraints, requirements, and specifications?

anhpham@hcmut.edu.vn

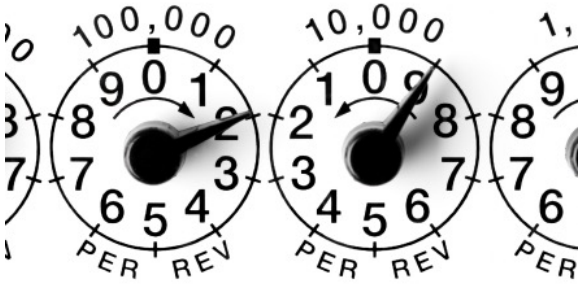
What Will Your Product Need? (1)

- **Human interface**: Keyboard, display, annunciator, motor, solenoid..?
  - Does the display need a touch-control overlay? MCUs have built-in touch-control electronics
  - Some MCUs can directly drive LCD segments.
  - Do you need individual pixel control?
  - Build or buy an off-the-shelf LCD?
  - How many I/O pins will the human interface need?

anhpham@hcmut.edu.vn

What Will Your Product Need? (2)

- **Control-and-measurement capabilities?**
  - Will standard digital, analog, and communication ports suffice? Beware of pin conflicts!
  - Must the MCU measure analog signals?
    - How many analog inputs and outputs?
    - What resolution and accuracy do you require?
    - Do you need signal conditioning circuits?
  - Do you need interrupts?
  - Do you need PWM outputs?



What Will Your Product Need? (3)

- **Do you need real-time or near real-time performance?**
  - You might need a real-time operating system, or a "scheduler" for your application.
  - Ethernet and USB communications require real-time response from an MCU.
  - Something you cannot determine an MCU's real-time capability until you create some code and test it.
  - Find out what MCU vendor software libraries offer.

anhpham@hcmut.edu.vn

What Will Your Product Need? (4)

- **What's your power budget?**
  - Line power
  - Battery power
  - Alternate energy source
  - MCU low-power and sleep modes
  - Take advantage of power- measuring development software tools
  - Consider system power needs and control

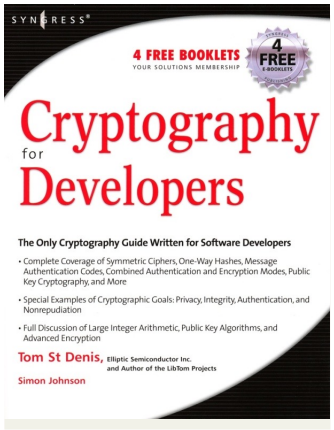
What Will Your Product Need? (5)

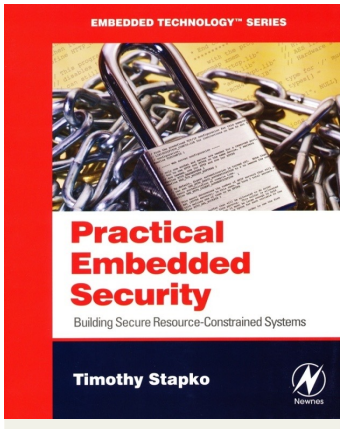
- **What types of communications does your product need?**
  - Chip-level: SPI, I2C, I2S?
  - System-level: CAN, UART, Ethernet..?
  - Wireless communications?
    - Standard or proprietary?
    - DIY or drop-in module?
    - FCC or agency approvals?
    - Range and data rate?
    - Regional limits?

What Will Your Product Need? (6)

- **Do you need math operations?**
  - Some MCUs have a multiplier-accumulator, which helps speed FFTs, FIR filter routines, etc.
  - Floating point or fixed point?
    - Single- or double-precision math?
    - IEEE 754?
  - Can MCU vendors supply a math library?
  - Use a math add-on chip:
    - Micromega 28-pin IC
    - www.micromegacorp.com

What Will Your Product Need? (7)





- **Does your product require security?**
  - How much security do you need?
  - Must you protect code, data, or both?
  - Investigate “secure” MCUs.
  - Will the Advanced Encryption Standard (AES) suffice?
  - Look for an MCU that includes an AES “engine.”
  - Buy or license encryption-decryption libraries.

  - **Security-related references:**
    - "Practical Embedded Security," by Timothy Stapko, Newnes. ISBN: 978-0-7506-8215-2.
    - ”Cryptography for Developers," Tom St. Denis and Simon Johnson, Syngress. ISBN: 9781597491044

ATM Approach to Security

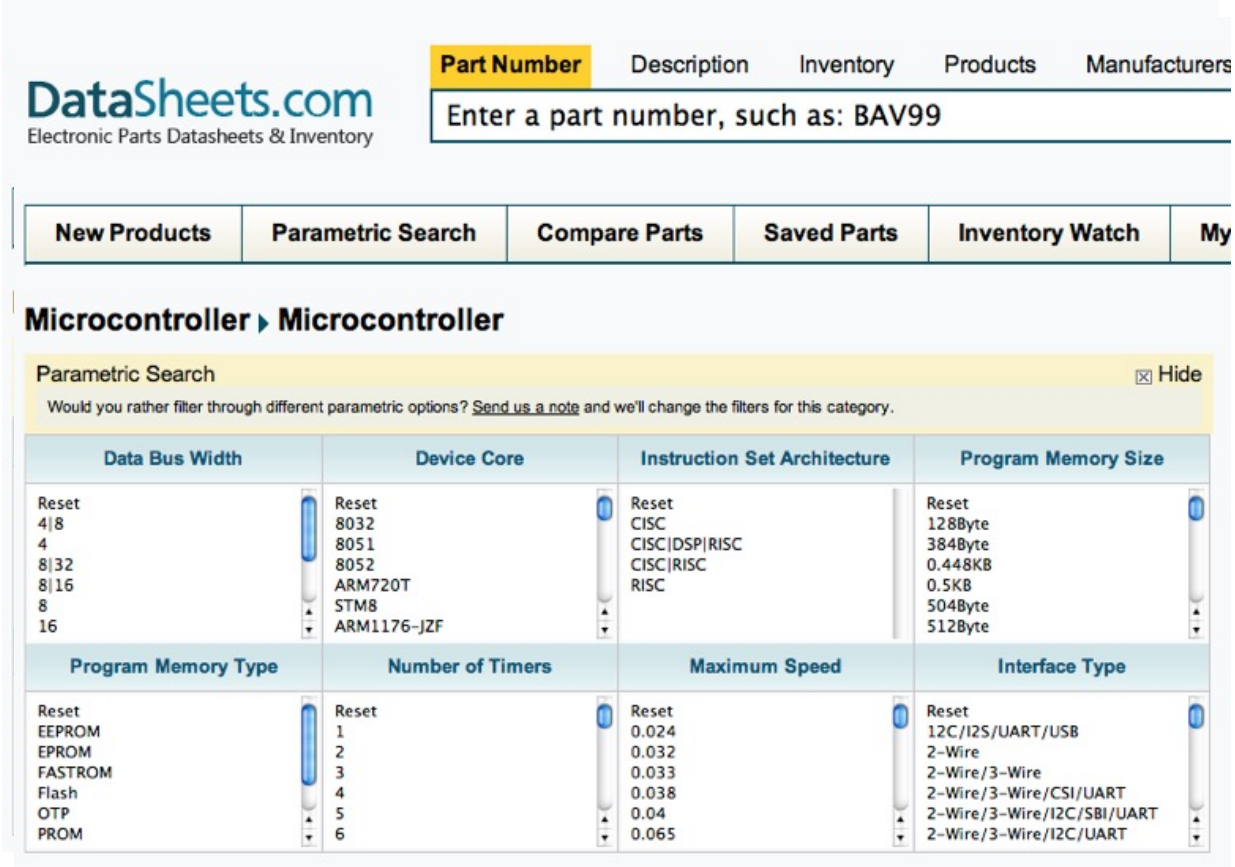
anhpham@hcmut.edu.vn

What Will Your Product Need? (8)

- **Will your product involve safety of humans?**
  - Learn about safety standards from FDA, FAA, FCC, etc.
  - Ask software companies for information about tools that measure compliance with standards: MISRA C, DO-178, etc.

Use MCU Selection Tools

- Web sites let you choose selection criteria and find MCUs:
  - datasheets.com
  - gruntwareinc.com
  - Vendor sites
  - Distributor sites

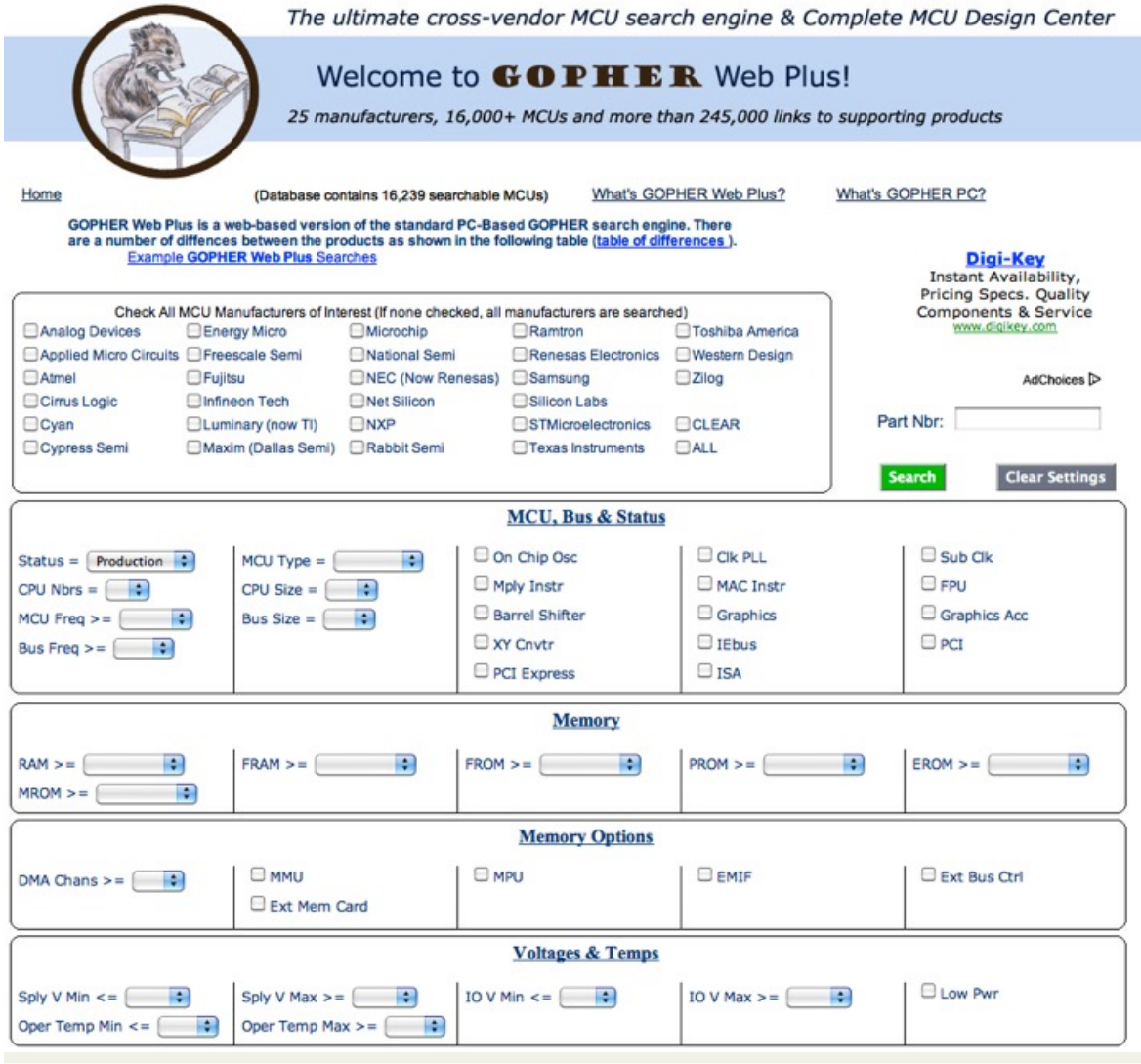


Showing 1- 15 of 24 search results for Microcontroller Prev Next | Page 1

Compare Save

Part Number	Manufacturer	Description	Inventory	Quantity	Buy Now
MSC1200Y2PFBT	Texas Instruments	Register or login	Digi-Key More Distributors	241	Buy now More
MSC1200Y3PFBT	Texas Instruments	Register or login	Digi-Key More Distributors	224	Buy now More
MSC1201Y2RHHT	Texas Instruments	Register or login	More Distributors		More
MSC1201Y3RHHT	Texas Instruments	Register or login	Digi-Key More Distributors	477	Buy now More

gruntwareinc.com



anhpham@hcmut.edu.vn

Manuf	PartNbr	ProdLine	Status	MCU Type	CPU	CPU Nbrs	CPU Size	MCU Freq	Bus Size	Bus Freq	On Chip Osc	Clk PLL	Sub Clk	Mply Instr	MAC Instr	FPU	Barrel Shifter	Graphics	Graphics Acc	XY Cnvtr	IEbus	PCI	E
MIC	PIC24HJ128GP202	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	
MIC	PIC24HJ128GP204	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	
MIC	PIC24HJ128GP206	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	
MIC	PIC24HJ128GP210	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	
MIC	PIC24HJ128GP306	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	
MIC	PIC24HJ128GP310	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	
MIC	PIC24HJ128GP502	PIC24H	Production	MCU	PIC24H	1	16	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	

GOPHER Links

Manufacturer: **Microchip**
Part Number: **PIC24HJ128GP202**
Pricing and Availablility: Octopart
Datasheet and/or Hardware Manual
App notes, eval boards, software, etc

There are currently **292,471** links to support products by other manufacturers in the Gopher database. The following links came from reference material provided by those manufacturers indicating that the products listed below support the MCU you selected: **PIC24HJ128GP202**

Hardware Cables/Adapters

Company	List Price	Part number	Description
CCS	$29.95	Tag-Connect	ICSP Programming Cable

Programmer/Debugger

Company	List Price	Part number	Description
CCS	$75	ICD-U64	In-Circuit Programmer/Debugger
CCS	$75	ICD-S40	In-Circuit Debugger/Programmer

Hardware Programmer

Company	List Price	Part number	Description
CCS	$199	MachX	Mach X Programmer
CCS	$199	LoadnGo	Load-n-Go Handheld Programmer
CCS	$899	PRIME8	Production Programmer

digikey.com

ırer	Core Processor	Core Size	Speed	Connectivity
'p	–	–	–	–
	ACE1001	4-Bit	30/20MHz	ACCESS.Bus (2-Wire/I²C, SMBus), CAN, Microwire/Plus (SPI), UART/USART
	ACE1202	8-Bit	40/20MHz	ACCESS.Bus (2-Wire/I²C, SMBus), Microwire/Plus (SPI)
	ACE1502	8/16-Bit	40/30MHz	ASC, CAN, EBI/EMI, I²C, SSC, UART/USART
	ARM Cortex-A8	16-Bit	60/30MHz	ASC, CAN, EBI/EMI, MLI, MSC, SSC
	ARM Cortex-M0	16/32-Bit	40kHz	ASC, CAN, EBI/EMI, SSC
	ARM Cortex-M4	32-Bit	625kHz	ASC, CAN, MLI, MSC, SSC
	ARM® Cortex™ - R4F	32-Bit Dual-Core	1MHz	ATA, Audio, CAN, EBI/EMI, I²C, IDE, MMC/SD, SPI, UART/USART, USB OTG
rica Inc	ARM® Cortex™-M4/M0		1.2MHz	ATA, Compact Flash, EBI/EMI, Memory Stick, MMC, SCI, SD, Smart Media, U
& Precision Sensors	ARM® Cortex-M3™		1.6MHz	ATAPI, Ethernet, I²C, SCI, SSI, USB

digikey.com

Part Number	Manufacturer Part Number	Description	Series	Manufacturer	Core Processor	Core Size	Speed	Connectivity	Peripherals	Number of I /O	Program Memory Size	Program Memory Type	EEPROM Size	RAM Size
-I/OTTR-ND	PIC10F200T-I/OT	IC PIC MCU FLASH 256X12 SOT23-6	PIC® 10F	Microchip Technology	PIC	8-Bit	4MHz	-	POR, WDT	3	384B (256 x 12)	FLASH	-	16 x 8
-I/OTCT-ND	PIC10F200T-I/OT	IC PIC MCU FLASH 256X12 SOT23-6	PIC® 10F	Microchip Technology	PIC	8-Bit	4MHz	-	POR, WDT	3	384B (256 x 12)	FLASH	-	16 x 8
-I/OTDKR-ND	PIC10F200T-I/OT	IC PIC MCU FLASH 256X12 SOT23-6	PIC® 10F	Microchip Technology	PIC	8-Bit	4MHz	-	POR, WDT	3	384B (256 x 12)	FLASH	-	16 x 8

ti.com

Add/Hide Parameters	Status	SubFamily	Max Speed (MHz)	CAN	ADC Channels	Motion PWM	SSI/SPI	I2C	Timers	Maximum 5-V Tolerant GPIOs	Dedicated 5-V Tolerant GPIOs	Watchdog Timers
Total Parts: 31 Matching Parts: **31** Reset	☐ ACTIVE ☐ PREVIEW	☐ 9000 Series	☐ 80	☐ 2 ☐ 3	☐ 16	☐ 0 ☐ 6 ☐ 8	☐ 2	☐ 2	☐ 5	☐ 60 ☐ 65 ☐ 72	☐ 0	☐ 2
Highlight Differences												
☐ LM3S9B81 - Stellaris Microcontroller	ACTIVE	9000 Series	80	3	16	0	2	2	5	65	0	2
☐ LM3S9B90 - Stellaris Microcontroller	ACTIVE	9000 Series	80	2	16	0	2	2	5	60	0	2
☐ LM3S9B92 - Stellaris Microcontroller	ACTIVE	9000 Series	80	2	16	8	2	2	5	65	0	2
☐ LM3S9B95 - Stellaris Microcontroller	ACTIVE	9000 Series	80	2	16	8	2	2	5	65	0	2
☐ LM3S9B96 - Stellaris Microcontroller	ACTIVE	9000 Series	80	2	16	8	2	2	5	65	0	2
☐ LM3S9BN2 - Stellaris Microcontroller	ACTIVE	9000 Series	80	2	16	8	2	2	5	72	0	2

anhpham@hcmut.edu.vn

Investigate MCU HW Tools

- **What hardware tools does an MCU vendor offer?**
  - Evaluation boards and kits
  - Development boards and kits
  - Reference designs
    - Schematic diagrams
    - PCB files
    - Bill of materials (BOM)
  - Code
    - Download and review manuals and user guides

Investigate MCU SW Tools

- **What software tools does an MCU vendor offer?**
  - Complete set of coding tools
    - IDE (Integrated Development Environment)
    - Editor, assembler, compiler, debugger
    - Project-management software
  - Technical support and user forums
  - Application notes
  - Software examples
  - Libraries
  - More about software in the next session

Dig Deeper

- **Locate more information and tools**
  - Run a Web search to find independent information about hardware, software, tools, and problems and accolades.
  - Look at third-party development hardware and software.
  - Examine information about your chosen MCU's "family."
  - Ask vendors about their MCU "roadmap."
  - Try demo software before you buy.
  - …

Further Reading

- https://www.youtube.com/watch?v=R7IPEeUyNNA
- https://www.youtube.com/watch?v=_Y211pA0ITM
- https://www.youtube.com/watch?v=vbaFMT9LXmg

- CO3009 -

- A Scheduler -

What is A Co-operative Scheduler?

- A **co-operative scheduler** provides a **single-tasking system** architecture

Operation:

- Tasks are scheduled to run at specific times (either on a periodic or one-shot basis)
- When a task is scheduled to run, it is added to the waiting list
- When the CPU is free, the next waiting task (if any) is executed
- The task runs to completion, then returns control to the scheduler

Implementation:

- The scheduler is simple and can be implemented in a small amount of code
- The scheduler must allocate memory for only a single task at a time
- The scheduler will generally be written entirely in a high-level language (such as 'C')
- The scheduler isn't a separate application; it becomes part of the developer's code

Performance:

- Obtaining rapid responses to external events requires care at the design stage

Reliability and safety:

- Co-operate scheduling is simple, predictable, reliable and safe

Introduction

- In this lecture, we discuss technique for creating a co-operative scheduler suitable for use in single processor environments.

- These provide a very flexible and predictable software platform for a wide range of embedded applications, from the simplest consumer gadget up to and including aircraft control systems.

Introduction

Context

- You are developing an embedded application using MCUs.
- The application is to have a time-triggered architecture, constructed using a scheduler.

Problem

- How do you create and use a co-operative scheduler?

Background - Function Pointer

- Just as we can determine the starting address of an array of data in memory ➔ we can also find the address in memory at which the executable code for a particular function begins.

- This address can be used as a 'pointer' to the function; most importantly, **it can be used to call the function.**

Background - Function pointer

- Used with care, function pointers can make it easier to design and implement complex programs.
- For example, suppose we are developing a large, safety-critical, application, controlling an industrial plant.
  - If we detect a critical situation, we may wish to shut down the system as rapidly as possible.
  - However, the appropriate way to shut down the system will vary, depending on the system state.
  - What we can do is create a number of different recovery functions and a function pointer. Every time the system state changes, we can alter the function pointer so that it is always pointing to the most appropriate recovery function.
  - In this way, we know that – if there is ever an emergency situation – we can rapidly call the most appropriate function, by means of the function pointer.

```
void Square_Number(int, int*);
int main(void) {
        int a = 2, b = 3;
        // pFn is a pointer with int and int pointer parameters (return void)
        void (* pFn)(int, int*);
        int Result_a, Result_b;
        pFn = Square_Number; // pFn holds address of Square_Number
        printf("Function code starts at address: %u\n", (tWord) pFn);
        printf("Data item a starts at address: %u\n\n", (tWord) &a);
        // Call 'Square_Number' in the conventional way
        Square_Number(a, &Result_a);
        // Call 'Square_Number' using function pointer
        (*pFn)(b, &Result_b);
        printf("%d squared is %d (using normal fn call)\n", a, Result_a);
        printf("%d squared is %d (using fn pointer)\n", b, Result_b);
        while(1);
}
```


```
void (*TMR0_InterruptHandler)(void);

void TMR0_SetInterruptHandler(void (* InterruptHandler)(void)){
    TMR0_InterruptHandler = InterruptHandler;
}

void TMR0_Initialize(void){
    TMR0_SetInterruptHandler(Test);
}
void TMR0_DefaultInterruptHandler(void){
    // add your TMR0 interrupt custom code
    // or set custom function using TMR0_SetInterruptHandler()
}

void TMR0_ISR(void) {
    if(TMR0_InterruptHandler) {
        TMR0_InterruptHandler();
    }
}
```


A Scheduler

- A scheduler has the following key components:
  - The scheduler data structure.
  - An initialization function.
  - A single interrupt service routine (ISR), used to update the scheduler at regular time intervals.
  - A function for adding tasks to the scheduler.
  - A dispatcher function that causes tasks to be executed when they are due to run.
  - A function for removing tasks from the scheduler (not required in all applications).

Overview

- Before discussing the scheduler components, we consider how the scheduler will typically appear to the user.

- To do this we will use a simple example
  - A scheduler is used to flash a single LED on and off repeatedly: on for one second, off for one second etc.

```
void main(void) {
        // Set up the scheduler
        SCH_Init_T2();
        // Prepare for the 'Flash_LED' task
        LED_Flash_Init();
        // Add the 'Flash LED' task (on for ~1000 ms, off for ~1000 ms)
        // - timings are in ticks (1 ms tick interval)
        // (Max interval / delay is 65535 ticks)
        SCH_Add_Task(LED_Flash_Update, 0, 1000);
        // Start the scheduler
        SCH_Start();
        while(1)  {
                SCH_Dispatch_Tasks();
        }
}

void SCH_Update(void)  {    }
```


1. The LED is switched on and off by means of a 'task' **LED_Flash_Update()**. Thus, if the LED is initially off and we call **LED_Flash_Update()** twice, we assume that the LED will be switched on and then switched off again.

2. To obtain the required flash rate, the scheduler calls **LED_Flash_Update()** every second.

3. We prepare the scheduler using the function **SCH_Init_T2()**.

4. After preparing the scheduler, we add the function **LED_Flash_Update()** to the scheduler task list using the **SCH_Add_Task()** function. At the same time we specify that the LED will be turned on and off at the required rate.

5. The timing of the **LED_Flash_Update()** function will be controlled by the function **SCH_Update(),** an interrupt service routine triggered by the overflow of Timer 2.

6. The 'Update' ISR does not execute the task: it calculates when a task is due to run and sets a flag. The job of executing **LED_Flash_Update()** falls to the dispatcher function (SCH_Dispatch_Tasks()), which runs in the main ('super') loop.

The scheduler data structure and task array

At the heart of the scheduler is the scheduler data structure: this is a user-defined data type which collects together the information required about each task.

```
typedef data struct {
// Pointer to the task (must be a 'void (void)' function)
        void ( * pTask)(void);
// Delay (ticks) until the function will (next) be run
        tWord Delay;
// Interval (ticks) between subsequent runs.
        tWord Period;
// Incremented (by scheduler) when task is due to execute
        tByte RunMe;
} sTask;

#define SCH_MAX_TASKS (10)

// The array of tasks
sTask SCH_tasks_G[SCH_MAX_TASKS];
```

anhpham@hcmut.edu.vn

The Size of the Task Array

- You **must** ensure that the task array is sufficiently large to store the tasks required in your application, by adjusting the value of **SCH_MAX_TASKS**.

- For example, if you schedule three tasks as follows:
  - SCH_Add_Task(Function_A, 0, 2);
  - SCH_Add_Task(Function_B, 1, 10);
  - SCH_Add_Task(Function_C, 3, 15);

Then SCH_MAX_TASKS must have a value of three (or more) for correct operation of the scheduler.

- Note also that, if this condition is not satisfied, the scheduler will generate an error code.

The Initialization Function

- Like most of the tasks we wish to schedule, the scheduler itself requires an initialization function.
- This performs various important operations – such as preparing the scheduler array and the error code variable.
- The main purpose of this function is to set up a timer that will be used to generate the regular 'ticks' that will drive the scheduler.

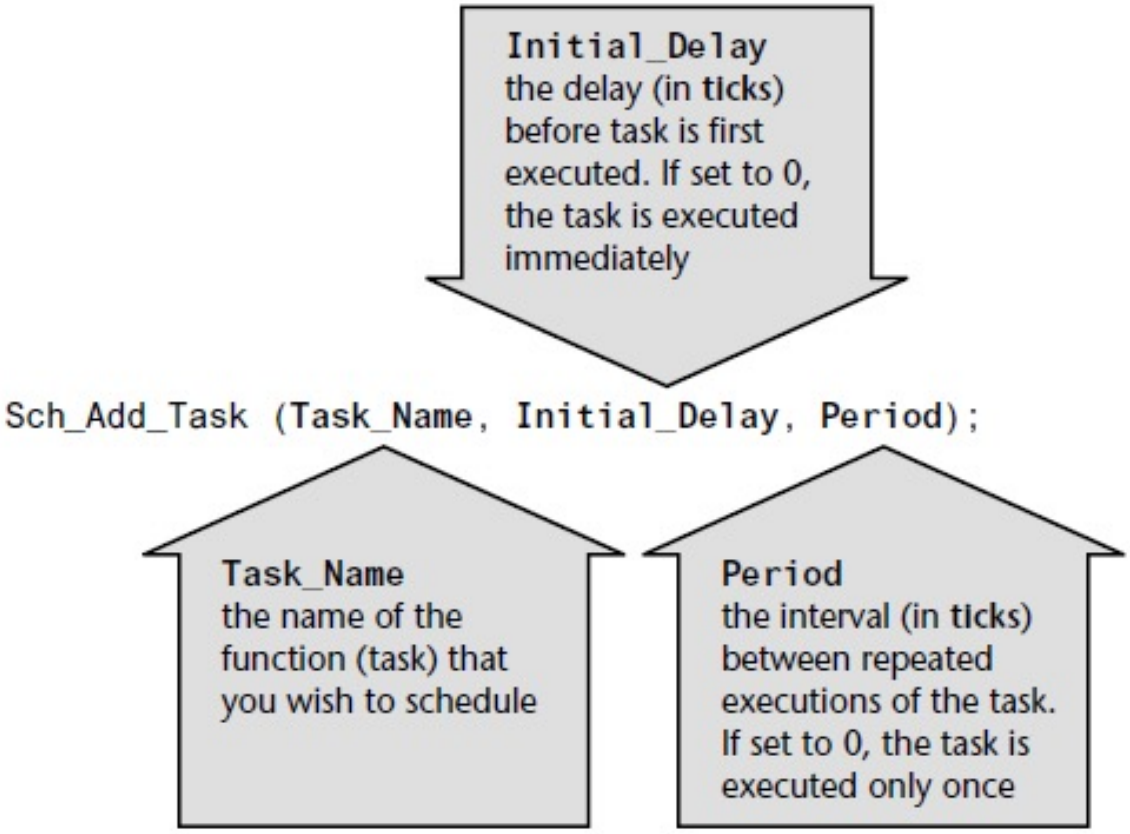
The 'Update' function

- The 'Update' function is the scheduler ISR. It is invoked by the overflow of the timer.
- Like most of the scheduler, the update function is not complex.
- When it determines that a task is due to run, the update function increments the RunMe field for this task: the task will then be executed by the dispatcher.

```
void SCH_Update(void)  {
    tByte Index;
    // NOTE: calculations are in *TICKS* (not milliseconds)
    for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
    // Check if there is a task at this location
        if (SCH_tasks_G[Index].pTask) {
            if (SCH_tasks_G[Index].Delay == 0) {
            // The task is due to run
                        SCH_tasks_G[Index].RunMe += 1; // Inc. the 'RunMe' flag
                        if (SCH_tasks_G[Index].Period) {
                                    // Schedule periodic tasks to run again
                                    SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].Period;
                        }
            } else {
                        // Not yet ready to run: just decrement the delay
                        SCH_tasks_G[Index].Delay -= 1;
            }
        }
    }
}
```


The 'Add Task' Function

The 'Add Task' function is used to add tasks to the task array, to ensure that they are called at the required time(s).



- **SCH_Add_Task(Do_X, 1000, 0);**
  - This set of parameters causes the function Do_X() to be executed once after 1,000 scheduler ticks
- **Task_ID = SCH_Add_Task(Do_X, 1000, 0);**
  - This does the same, but saves the task ID (the position in the task array) so that the task may be subsequently deleted, if necessary (see SCH_Delete_Task() for further information about the removal of tasks from the task array)
- **SCH_Add_Task(Do_X, 0, 1000);**
  - This causes the function Do_X() to be executed regularly every 1,000 scheduler ticks; the task will first be executed as soon as the scheduling is started.
- **SCH_Add_Task(Do_X, 300, 1000);**
  - This causes the function Do_X() to be executed regularly every 1,000 scheduler ticks; task will be first executed at T = 300 ticks, then 1,300, 2,300 etc

```
tByte SCH_Add_Task(void (code * pFunction)(), const tWord DELAY, const tWord PERIOD) {
    tByte Index = 0;
   // First find a gap in the array (if there is one)
   while ((SCH_tasks_G[Index].pTask != 0) && (Index < SCH_MAX_TASKS)) {
        Index++;
    }

   if (Index == SCH_MAX_TASKS) {// Have we reached the end of the list?
        // Task list is full, Set the global error variable
      Error_code_G = ERROR_SCH_TOO_MANY_TASKS;
        return SCH_MAX_TASKS; // Also return an error code
   }

   // If we're here, there is a space in the task array
   SCH_tasks_G[Index].pTask = pFunction;
   SCH_tasks_G[Index].Delay = DELAY;
   SCH_tasks_G[Index].Period = PERIOD;
   SCH_tasks_G[Index].RunMe = 0;

   return Index; // return position of task (to allow later deletion)
}
```


The 'Dispatcher'

- The 'Update' function does not execute any tasks.
- The tasks that are due to run are invoked through the 'Dispatcher' function.

```
void SCH_Dispatch_Tasks(void) {
   tByte Index;

 // Dispatches (runs) the next task (if one is ready)
   for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
      if (SCH_tasks_G[Index].RunMe > 0) {
         (*SCH_tasks_G[Index].pTask)(); // Run the task
        SCH_tasks_G[Index].RunMe -= 1; // Reset / reduce RunMe flag
        // Periodic tasks will automatically run again
        // - if this is a 'one shot' task, remove it from the array
        if (SCH_tasks_G[Index].Period == 0) SCH_Delete_Task(Index);
     }
   }

   SCH_Report_Status(); // Report system status
   SCH_Go_To_Sleep(); // The scheduler enters idle mode at this point
}
```


Do We need a Dispatch Function?

- The use of both the 'Update' and 'Dispatch' functions may seem a rather complicated way of running the tasks. Specifically, it may appear that the Dispatch function in unnecessary and that the Update function could invoke the tasks directly.

- However, the split between the Update and Dispatch operations is necessary, to maximize the reliability of the scheduler in the presence of long tasks.

- Suppose we have a scheduler with a tick interval of 1ms and, for whatever reason, a scheduled task sometimes has a duration of 3ms.
  - If the Update function runs the functions directly then – all the time the long task is being executed – the tick interrupts are effectively disabled. Specifically, two 'ticks' will be missed. This will mean that all system timing is seriously affected and may mean that two (or more) tasks are not scheduled to execute at all.
  - If the Update and Dispatch function are separated, system ticks can still be processed while the long task is executing. This means that we will suffer task 'jitter' (the 'missing' tasks will not be run at the correct time), but these tasks will, eventually, run.

The 'Delete Task' Function

- When tasks are added to the task array, **SCH_Add_Task()** returns the position in the task array at which the task has been added:
  - Task_ID = SCH_Add_Task(Do_X,1000,0);

- Sometimes it can be necessary to delete tasks from the array. To do so, **SCH_Delete_Task()** can be used as follows:
  - SCH_Delete_Task(Task_ID);

```
bit SCH_Delete_Task(const tByte TASK_INDEX) {
    bit Return_code;
     if (SCH_tasks_G[TASK_INDEX].pTask == 0) {
           // No task at this location...
          // Set the global error variable
          Error_code_G = ERROR_SCH_CANNOT_DELETE_TASK;
           // ...also return an error code
          Return_code = RETURN_ERROR;
     } else {
          Return_code = RETURN_NORMAL;
     }
      SCH_tasks_G[TASK_INDEX].pTask = 0x0000;
      SCH_tasks_G[TASK_INDEX].Delay = 0;
      SCH_tasks_G[TASK_INDEX].Period = 0;
      SCH_tasks_G[TASK_INDEX].RunMe = 0;
      return Return_code; // return status
}
```


Reporting Errors

- Hardware fails; software is never perfect; errors are a fact of life.
- To report errors at any part of the scheduled application, we use an (8-bit) error code variable Error_code_G, which is defined as follows:
  - tByte Error_code_G = 0;
- To record an error we include lines such as:
  - Error_code_G = ERROR_SCH_TOO_MANY_TASKS;
  - Error_code_G = ERROR_SCH_WAITING_FOR_SLAVE_TO_ACK;
  - Error_code_G = ERROR_SCH_WAITING_FOR_START_COMMAND_FROM_MASTER;
  - Error_code_G = ERROR_SCH_ONE_OR_MORE_SLAVES_DID_NOT_START;
  - Error_code_G = ERROR_SCH_LOST_SLAVE;
  - Error_code_G = ERROR_SCH_CAN_BUS_ERROR;
  - Error_code_G = ERROR_I2C_WRITE_BYTE_AT24C64;

```
void SCH_Report_Status(void) {
#ifdef SCH_REPORT_ERRORS // ONLY APPLIES IF WE ARE REPORTING ERRORS
// Check for a new error code
if (Error_code_G != Last_error_code_G) {
    // Negative logic on LEDs assumed
    Error_port = 255 - Error_code_G;
    Last_error_code_G = Error_code_G;
    if (Error_code_G != 0) {
        Error_tick_count_G = 60000;
   } else {
       Error_tick_count_G = 0;
   }
} else {
    if (Error_tick_count_G != 0) {
        if (--Error_tick_count_G == 0) {
           Error_code_G = 0; // Reset error code
       }
   }
}
#endif
}
```


What does that error code mean?

- The forms of error reporting discussed here are low-level in nature and are primarily intended to assist the developer of the application or a qualified service engineer performing system maintenance.

- An additional user interface may also be required in your application to notify the user of errors, in a more user-friendly manner.

Reliability and Safety Implications

- Make sure the task array is large enough
- Take care with function pointers
- Dealing with task overlap
  - Suppose we have two tasks in our application (Task A, Task B). We further assume that Task A is to run every second and Task B every three seconds. We assume also that each task has a duration of around 0.5ms.
  - Suppose we schedule the tasks as follows (assuming a 1ms tick interval):
    - SCH_Add_Task(TaskA,0,1000);
    - SCH_Add_Task(TaskB,0,3000);

Reliability and Safety Implications

- Dealing with task overlap
  - In this case, the two tasks will sometimes be due to execute at the same time. On these occasions, both tasks will run, but Task B will always execute after Task A. This will mean that if Task A varies in duration, then Task B will suffer from ‘jitter’: it will not be called at the correct time when the tasks overlap.
  - Alternatively, suppose we schedule the tasks as follows:
    - SCH_Add_Task(TaskA, 0, 1000);
    - SCH_Add_Task(TaskB, 5, 3000);
  - Now, both tasks still run every 1,000 ms and 3,000 ms (respectively), as required. However, Task A is explicitly scheduled always to run 5 ms before Task B. As a result, Task B will always run on time.

Guidelines for Predictable and Reliable Scheduling

1. For precise scheduling, the scheduler tick interval should be set to match the 'greatest common factor' of all the task intervals.

2. All tasks should have a duration less than the schedule tick interval, to ensure that the dispatcher is always free to call any task that is due to execute. Software simulation can often be used to measure the task duration.

3. In order to meet Condition 2, all tasks must 'timeout' so that they cannot block the scheduler under any circumstances. Note that this condition can often be met by incorporating, where necessary, a **LOOP TIMEOUT** or a **HARDWARE TIMEOUT** in scheduled tasks.

4. Please remember that this condition also applies to any functions called from within a scheduled task, including any library code provided by your compiler manufacturer. In many cases, standard functions (like printf()) do not include timeout features. They must not be used in situations where predictability is required.

5. The total time required to execute all the scheduled tasks must be less than the available processor time. Of course, the total processor time must include both this 'task time' and the 'scheduler time' required to execute the scheduler update and dispatcher operations.

6. Tasks should be scheduled so that they are never required to execute simultaneously: that is, task overlaps should be minimized. Note that where all tasks are of a duration much less than the scheduler tick interval, and that some task jitter can be tolerated, this problem may not be significant.

Overall Strengths and Weaknesses

- The scheduler is simple and can be implemented in a small amount of code.
- The applications based on the scheduler are inherently predictable, safe and reliable.
- The scheduler is written entirely in 'C': it is not a separate application but becomes part of the developer's code.
- The scheduler supports team working, since individual tasks can often be developed largely independently and then assembled into the final system.
- Obtain rapid responses to external events requires care at the design stage.
- The tasks cannot safely use interrupts: the only interrupt that should be active in the application is the timer-related interrupt that drives the scheduler itself.

Further Reading

- FreeRTOS
  - https://www.youtube.com/watch?v=F321087yYy4
  - https://www.youtube.com/watch?v=JIr7Xm_riRs
  - https://www.youtube.com/watch?v=CdpgqpuPSyQ

- CO3009 -

- Efficient C Programming & Additional Exercise -

Problems with #define

```
#define PI_PLUS_1        3.14 + 1
.
.
x = 5 * PI_PLUS_1;   // The compiler sees the statement as
                     // x = 5 * 3.14 + 1
                     //So, it will be resolved as follows:
                     // x = (5 * 3.14) + 1
                     //Which is not what we want!
                     // Solution: #define PI_PLUS_1  (3.14 + 1)
                     //Moral: Beware of the "()" while dealing
                     //with the #define statement
```

anhpham@hcmut.edu.vn

Problem with Macros (1)

```
#define ADD(a,b) a + b
.
.
c = 2 + ADD(1,2);    // Result is 5 → Correct

d = 2 * ADD(1,2);    // Result is 4 → Incorrect
```

```
#define ADD(a,b) (a + b)
.
.
c = 2 * ADD(1,2);    // Result is 6 → Correct
```

Moral: *Again, beware of the "()" while dealing with the #define statement*

Problem with Macros (2)

```
#define MULT(a,b)  (a * b)
.
.
c = 3 + MULT(1,2);          // Result is 5 → Correct

d = 3 + MULT(1+1,2+2);      // Result is 8 → Incorrect
```

```
#define MULT(a,b)  ((a) * (b))
.
.
d = 3 + MULT(1+1,2+2);      // Result is 11 → Correct
```

Moral: I told you! Beware of the "()" while dealing with the #define statement

Playing around with Increment

- Example 1:

```
a = 2;
b = a++;
//Values after:      a = 3, while b = 2
```

- Example 2:

```
a = 2;
b = ++a;
//Values after:      a = 3, while b = 3
```

- Example 3:

```
a = 5;
b = 2;
c = a+++b;
//Values after:      a = 6, b = 2, while c = 7
```


Bit Manipulation (1)

- Detect if two integers have opposite signs:

```
int x, y;              // input values to compare signs bool
f = ((x ^ y) < 0);     // true iff x and y have opposite signs
```

- Determine if an unsigned integer is zero or a power of 2:

```
unsigned int v;      // we want to see if v is zero or a power of 2
bool f;              // the result goes here
f = (v & (v - 1)) == 0;
```

- Determine if an unsigned integer is a power of 2:

```
f = v && !(v & (v - 1));
```


Bit Manipulation (2)

- Merge bits from two values according to a mask:

```
unsigned int a;     // value to merge in non-masked bits unsigned
int b;              // value to merge in masked bits unsigned
int mask;       // 1 where bits from b should be selected; 0 where from a.
unsigned int r;     // result of (a & ~mask) | (b & mask) goes here
r = a ^ ((a ^ b) & mask);
```

- Counting bits set:

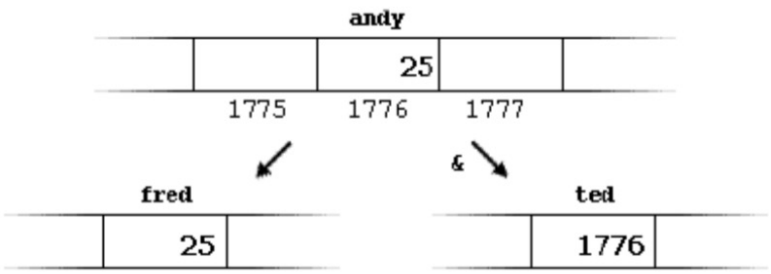
```
unsigned int v; // count the number of bits set in v
unsigned int c; // c accumulates the total bits set in v
for (c = 0; v; v >>= 1)
        c += v & 1;
```


Pointers

- Reference to a data object or a function
- Helpful for "call-by-reference" functions and dynamic data structures implementations
- Very often the only efficient way to manage large volumes of data is to manipulate not the data itself, but pointers to the data

Example:

```
andy = 25;
fred = andy;
ted = &andy;
```



anhpham@hcmut.edu.vn

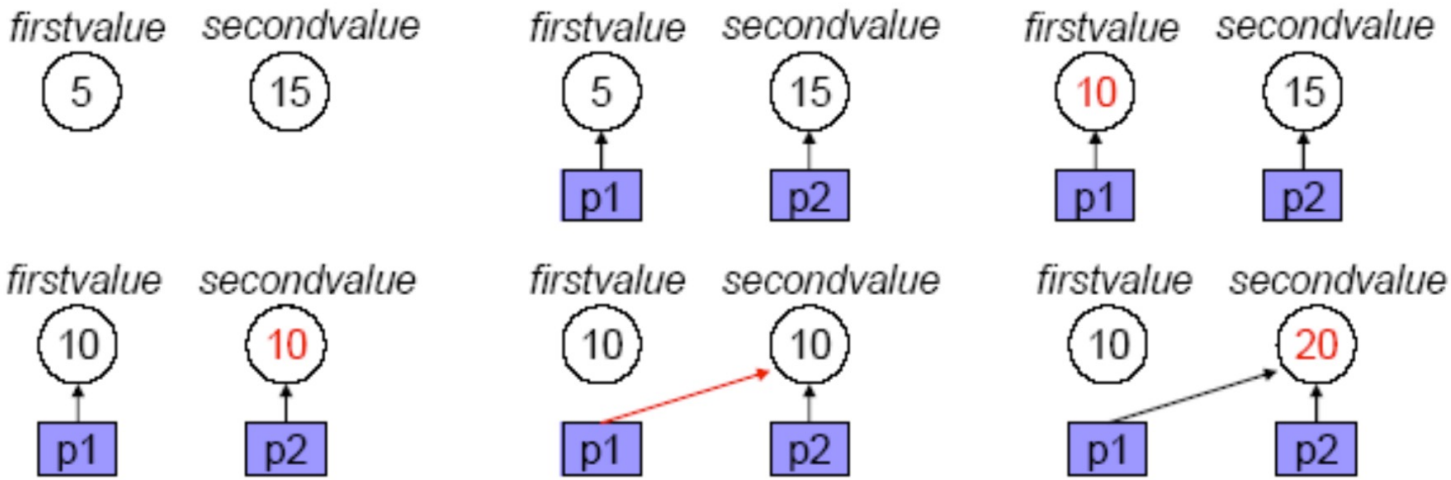
Pointers Example

```
int firstvalue = 5, secondvalue = 15;
int * p1, * p2;

p1 = &firstvalue;  // p1 = address of firstvalue
p2 = &secondvalue; // p2 = address of secondvalue
*p1 = 10;          // value pointed by p1 = 10
*p2 = *p1;         // value pointed by p2 = value pointed by p1
p1 = p2;           // p1 = p2 (value of pointer is copied)
*p1 = 20;          // value pointed by p1 = 20
```

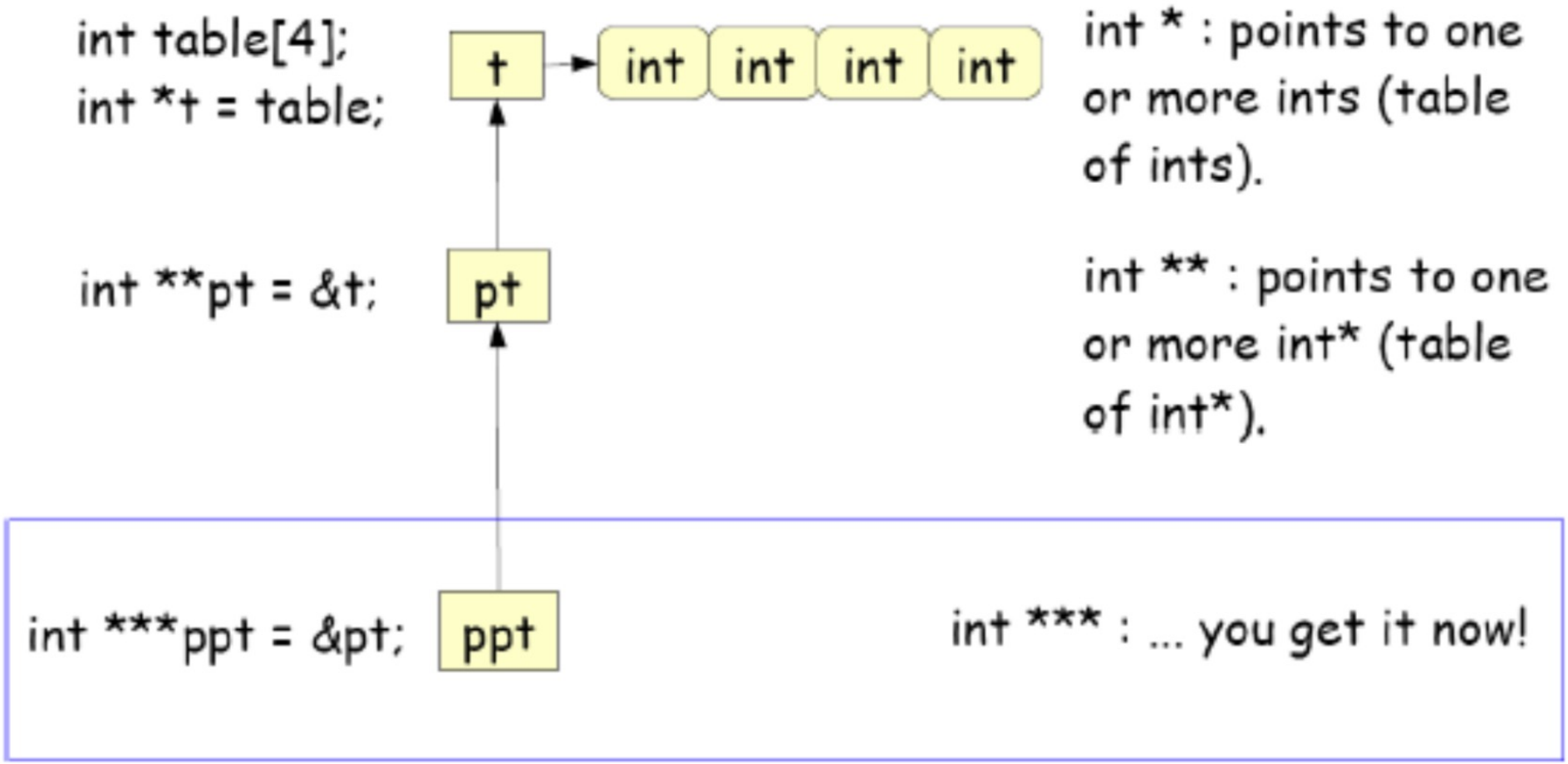
firstvalue = ?
secondvalue = ?

Pointers Example

```
int firstvalue = 5, secondvalue = 15;
int * p1, * p2;

p1 = &firstvalue;  // p1 = address of firstvalue
p2 = &secondvalue; // p2 = address of secondvalue
*p1 = 10;          // value pointed by p1 = 10
*p2 = *p1;         // value pointed by p2 = value pointed by p1
p1 = p2;           // p1 = p2 (value of pointer is copied)
*p1 = 20;          // value pointed by p1 = 20
```



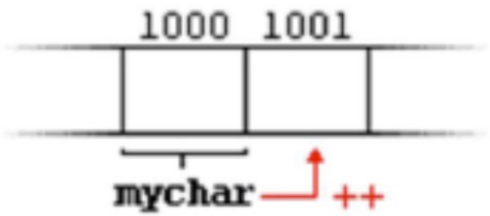
More Pointers Fun

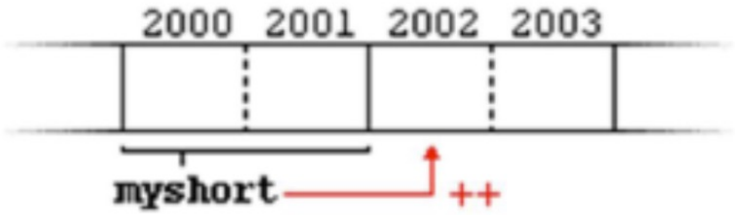


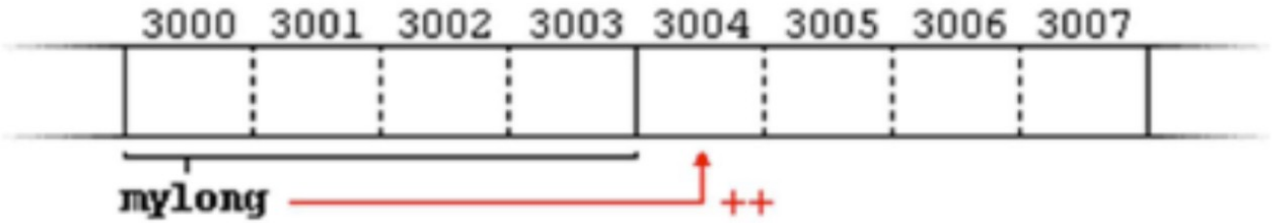
Pointers are Typed

```
char *mychar;
short *myshort;
long *mylong;
```

```
mychar++;
myshort++;
mylong++;
```





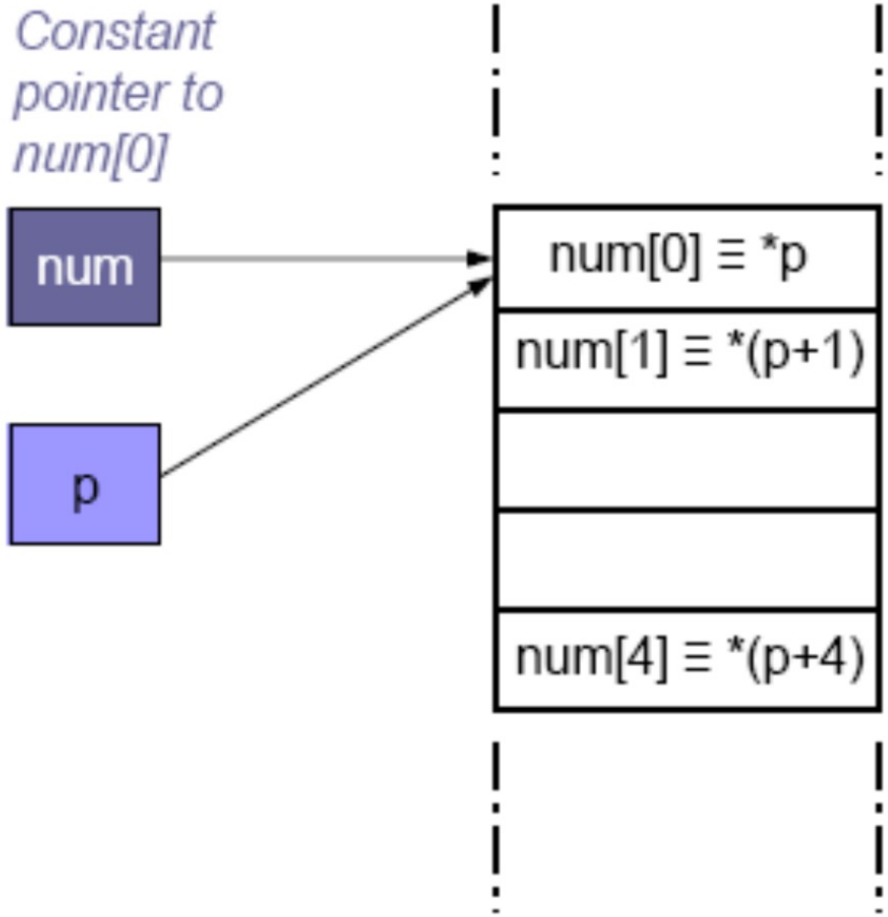


Pointers and Array

- *int num[5];*
- *int *p;*
- *p = num;*

~~*num = p;*~~

An array is a constant pointer



Pointers Precedence Issues

- *(array+i) ≡ array[i]
- *array+i ≡ array[0] + i
- *p++ ≡ *(p++)
- Notice the difference with (*p)++
- Better use parentheses to prevent mistakes
- int * ptr1, ptr2; Vs int * ptr1, * ptr2;

Efficient C Programming

- How to write C code in a style that will compile efficiently (**increased speed and reduced code size**) on ARM architecture?
  - How to use data types efficiently?
  - How to write loops efficiently?
  - How to allocate important variables to registers?
  - How to reduce the overhead of a function call?
  - How to pack data and access memory efficiently?

anhpham@hcmut.edu.vn

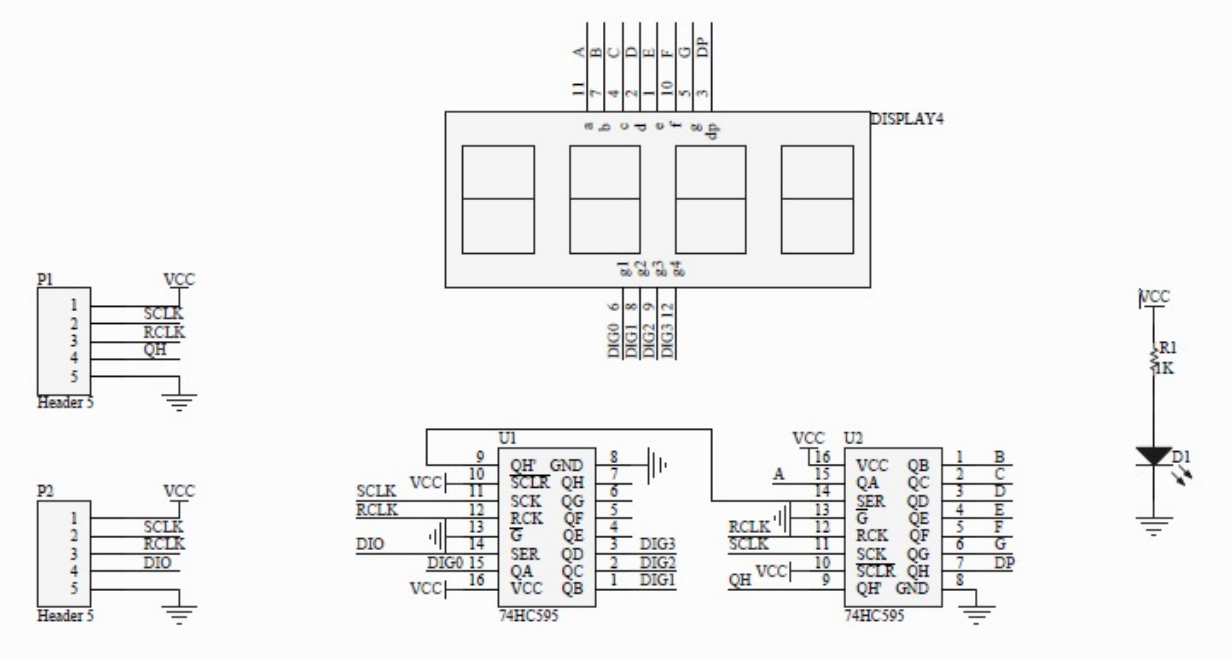
References

- A.N. Sloss, D. Symes, and C. Wright, “ARM System Developers Guide”

- CO3009 -

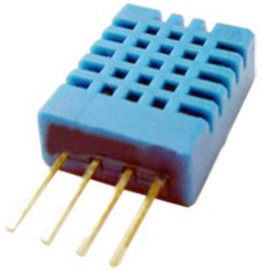
- Additional Exercises -

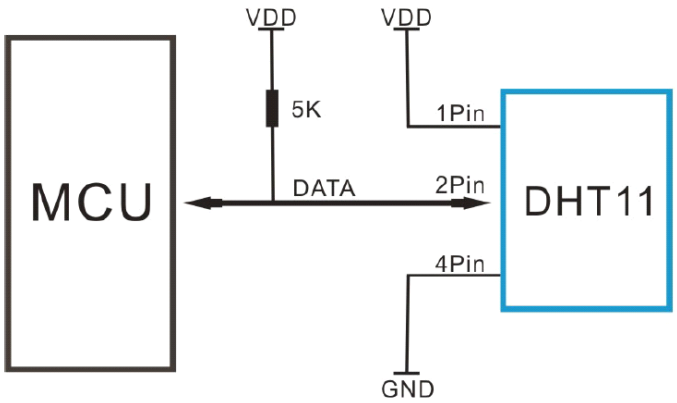
Four 7-Segment LEDs using 74HC595



DHT11 Humidity & Temperature Sensor

- Your task is to read Humidity &Temperature data from DHT11
- Communication Process
  - Serial Interface (Single-Wire Two-Way)





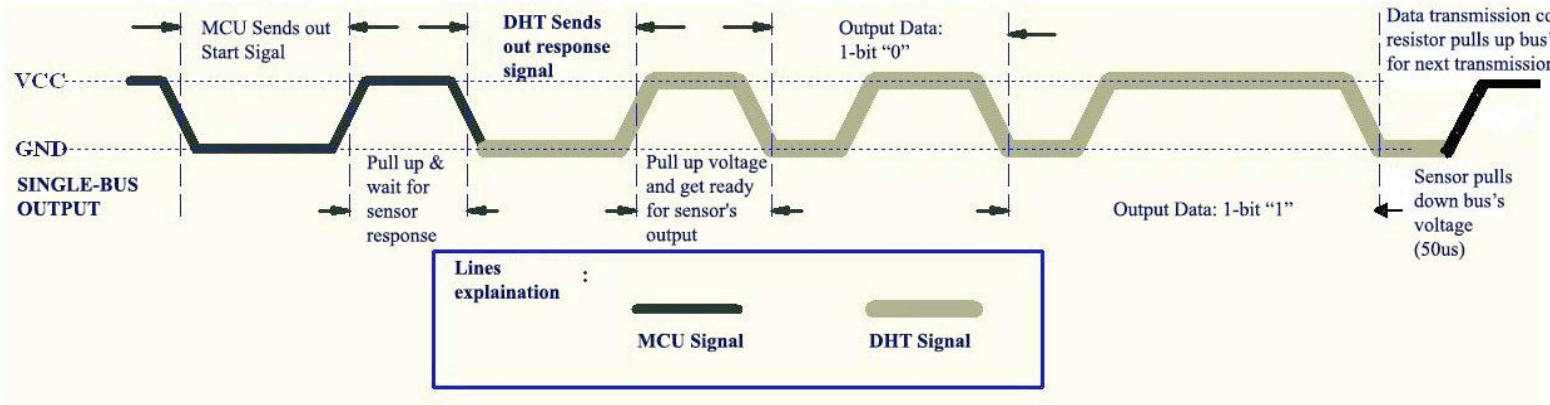
Single-Wire Two-Way (DHT11)

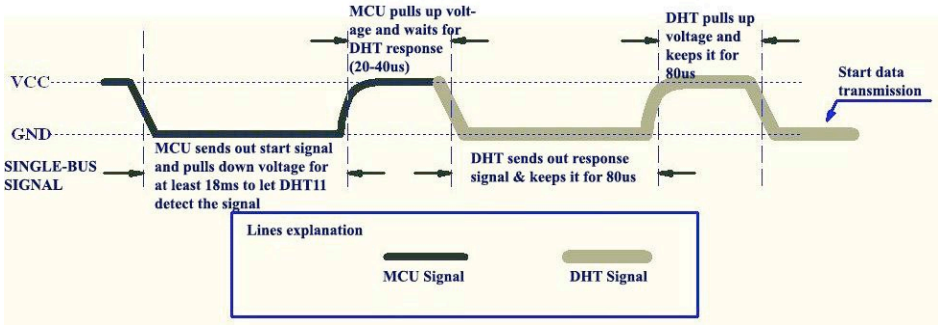
- Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor.
- One communication process is about 4ms.
- Data consists of decimal and integral parts.
- A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.
  - 8 bit integral humidity data
  - 8 bit decimal humidity data
  - 8 bit integral temperature data
  - 8 bit decimal temperature data
  - 8 bit check sum.
    - If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

Overall Communication Process

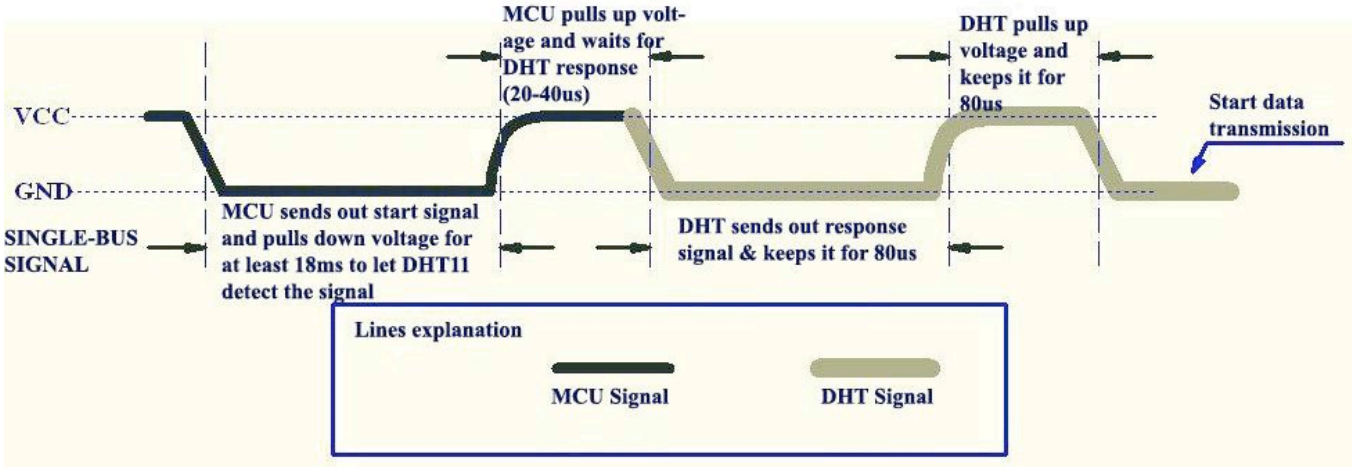
- When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal.
- Once it is completed, DHT11 sends a response signal of **40-bit data** that include the relative humidity and temperature information to MCU.
- Users can choose to collect (read) some data.
- Without the start signal from MCU, DHT11 will not give the response signal to MCU.
- Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

MCU Sends out Start Signal to DHT

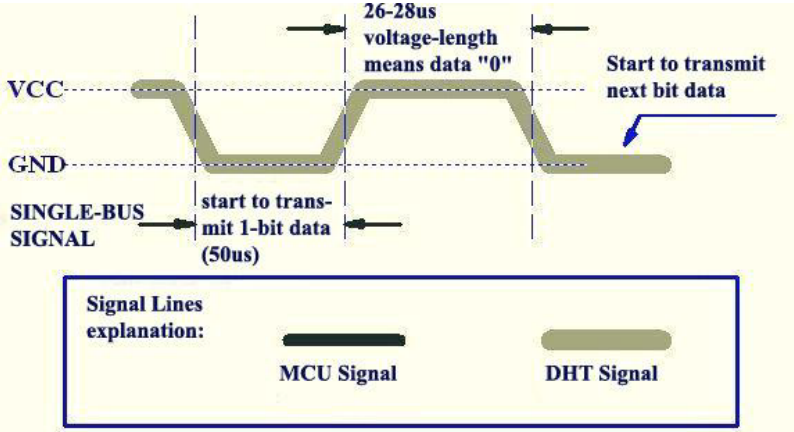




MCU Sends out Start Signal to DHT



DHT Responses to MCU



0 indication

1 indication