MEC302: Embedded Computer Systems

Theme II: Design of Embedded Computer Systems
Lecture 6 – Input, output and peripheral devices

Dr. Timur Saifutdinov

Assistant Professor at EEE, SAT

Email: Timur.Saifutdinov@xjtlu.edu.cn

Outline

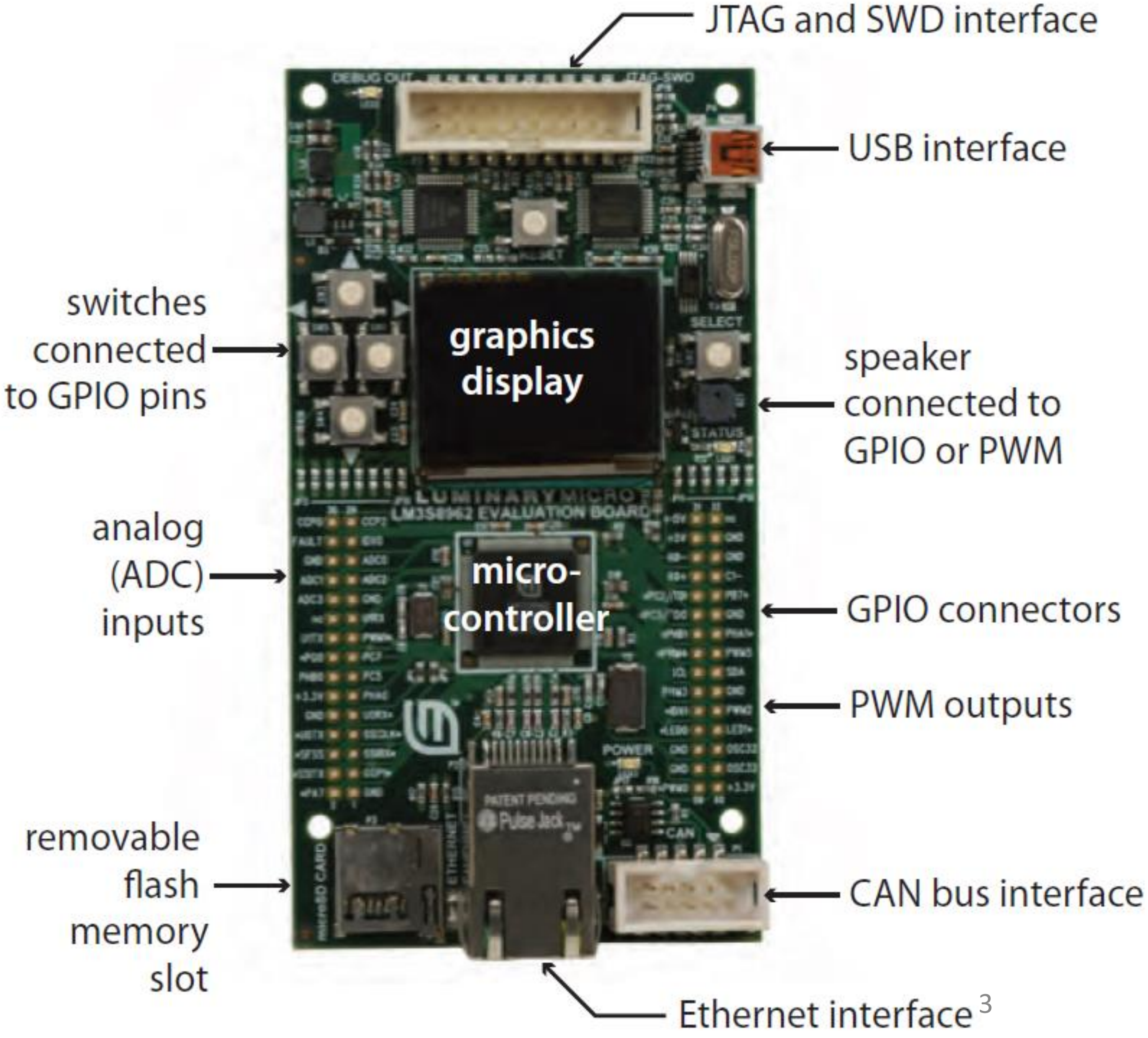
- Input and Output (I/O) Hardware
  - General-purpose digital I/O
  - Pulse width modulation
  - Data interfaces: serial, parallel and buses
- Mechanisms to interact with the external world
  - Interrupts and exceptions
  - Interrupt controllers
  - Interrupt models
- 8051 Architecture (to be used in Lab sessions and assignments)
  - Technical specification/Programming environment/Edsim51 emulator

I/O Hardware

For the controller to operate in the physical world, it needs:

1. Read instructions from the physical memory
   - Flash memory, USB interface, etc.;
2. For inputs:
   - Distinguish discrete voltage levels
     - Convert analog signals to a binary sequence of data (ADC);
     - General purpose input and output (GPIO)
     - Data transfer interfaces (USB, Eth., CAN)
3. For outputs:
   - Provide binary output via digital GPIO
   - Provide non-binary output through
     - Sequence of bits (GPIOs, DAC);
     - One bit output varying in time (eg, PWM);
     - Data transfer interfaces (USB, Eth., CAN).

#Stellaris LM3S8962 evaluation board:



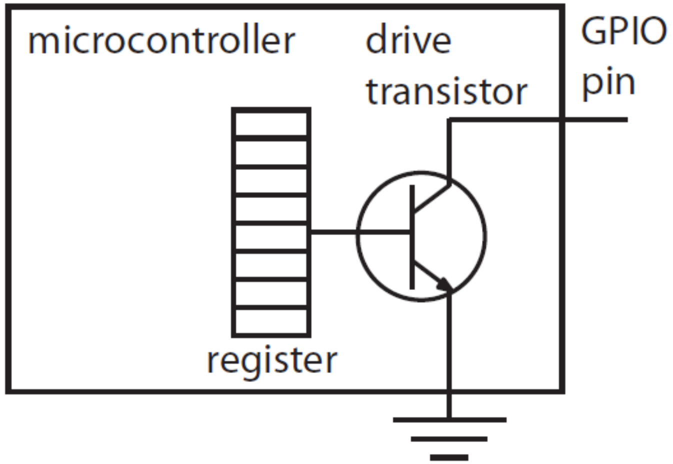
I/O Hardware

General-Purpose Digital I/O

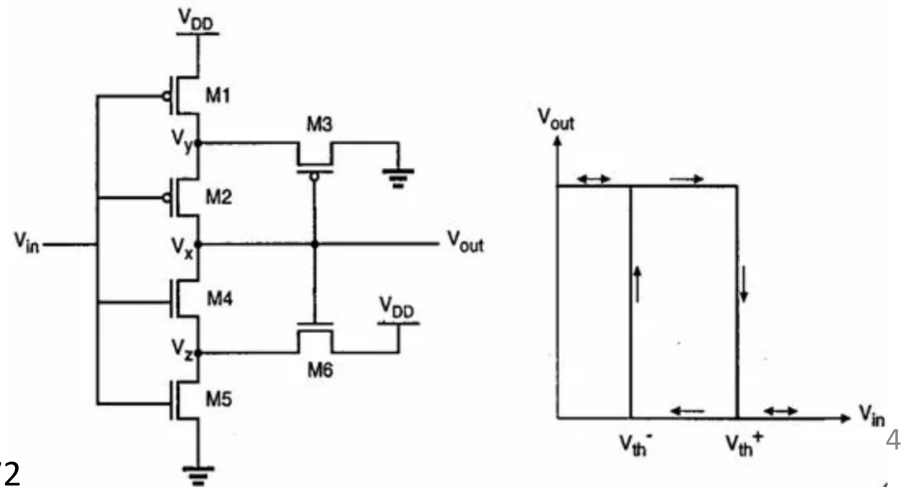
- Represent logical 0 or 1 with an actual voltage level being either 0 or +V (Low or High) – not necessarily respectively;
- When configured as output
  - “Write” to a **Memory-Mapped Register (MMR)** controls the pin voltage from software;
- When configured as input
  - External signal (voltage) “writes” to a register through a circuitry, e.g., **Schmitt trigger** for voltage compatibility and to avoid chattering

Input and output current needs to be maintained (eg, via resistors, amplifiers and isolators).

#GPIO output configuration (ie, open collector):



GPIO input configuration (ie, Schmitt trigger) [1]:



[1] Sequential CMOS and NMOS Logic Circuits, https://slideplayer.com/slide/3942472

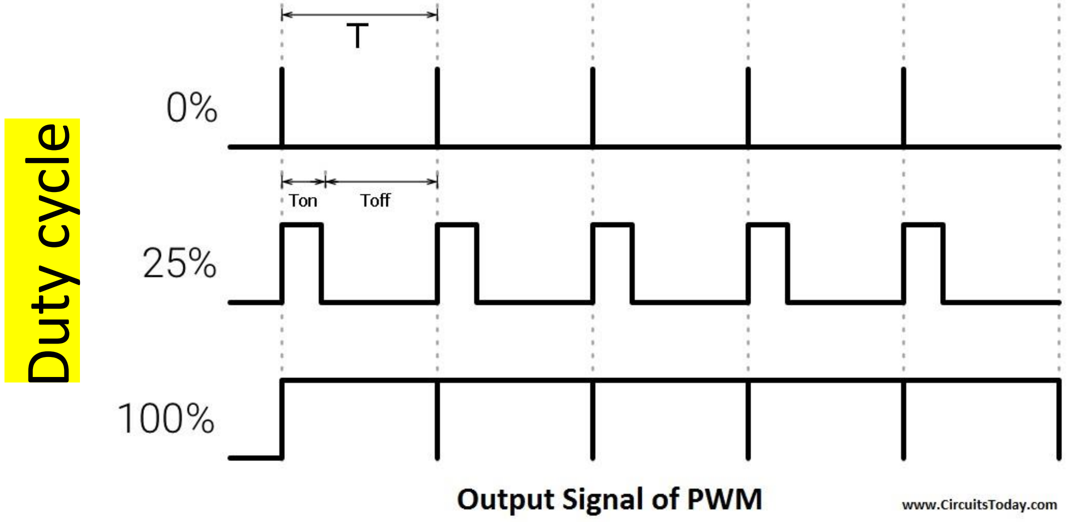
PWM信号是一种方波信号，具有固定的周期和可变的脉冲宽度。
脉冲宽度（也称为占空比）是指一个周期内高电平持续的时间与整个周期的时间之比。通过调整脉冲宽度，可以改变PWM信号的平均电压，从而实现对输出设备的控制

I/O Hardware

Pulse width modulation (PWM) – binary output periodic signal of variable duration;

- The main PWM characteristics are:
  - Output frequency;
  - Duty cycle.
- Technique for delivering a variable amount of power to control external devices, e.g.:
  - Speed of electric motors;
  - Brightness of LED and incandescent light;
  - Temperature of a heating element;
  - Other devices that tolerate rapid changes in voltage and current.

PWM signal:



#Consider a resistive heating element.

The amount of power supplied by PWM:

$$P = \frac{T_{Duty}}{T} \cdot \frac{V^2}{R},$$

where T_{Duty} can be flexibly varied.

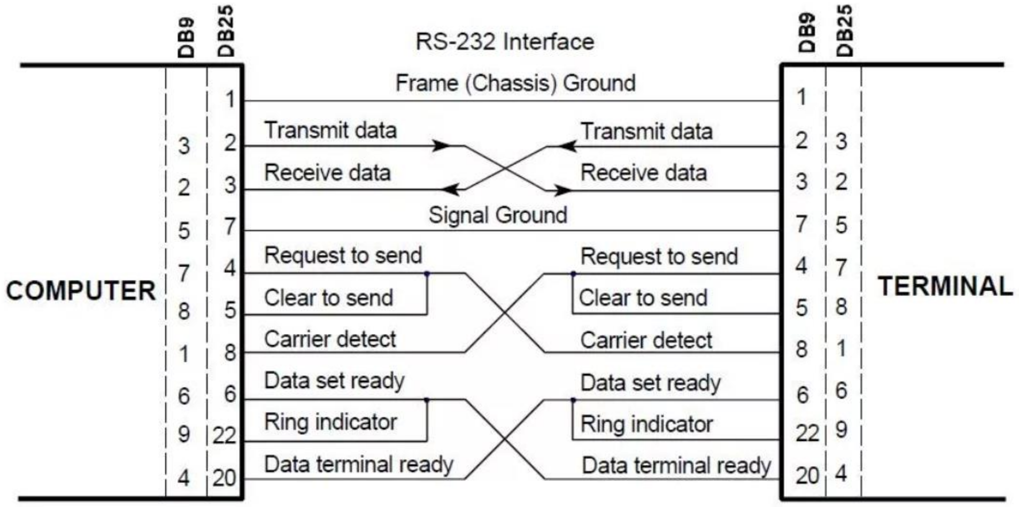
I/O Hardware

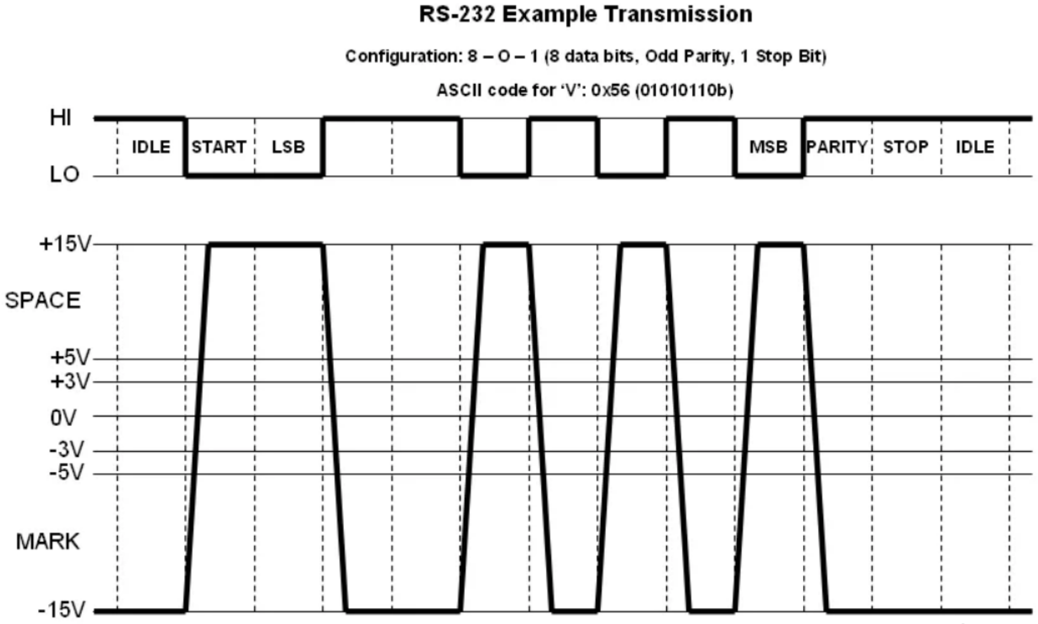
Data interfaces:

- **Serial interface** sends information (i.e., a byte) point-to-point as a sequence of bits.
  - E.g., RS-232 – is EIA* standard for asynchronous data transfer between two devices:
    - **Universal Asynchronous Receiver/Transmitter (UART)** is a hardware device on a controller to convert 8-bit register into a sequence of bits.

      #C program to send a sequence of 10 bytes:

      ```
      1.  for (i = 0; i < 10; i++) {
      2.      while(!(UCSR0A & 0x20));
      3.      UDR0 = x[i];
      4.  }
      ```





- Other serial interfaces are:
  - RS-422/423, 485;
  - Universal Serial Bus (USB);
  - Many others: I2C, PCI, FireWire, MIDI,…

* - Electronics Industries Association

I/O Hardware

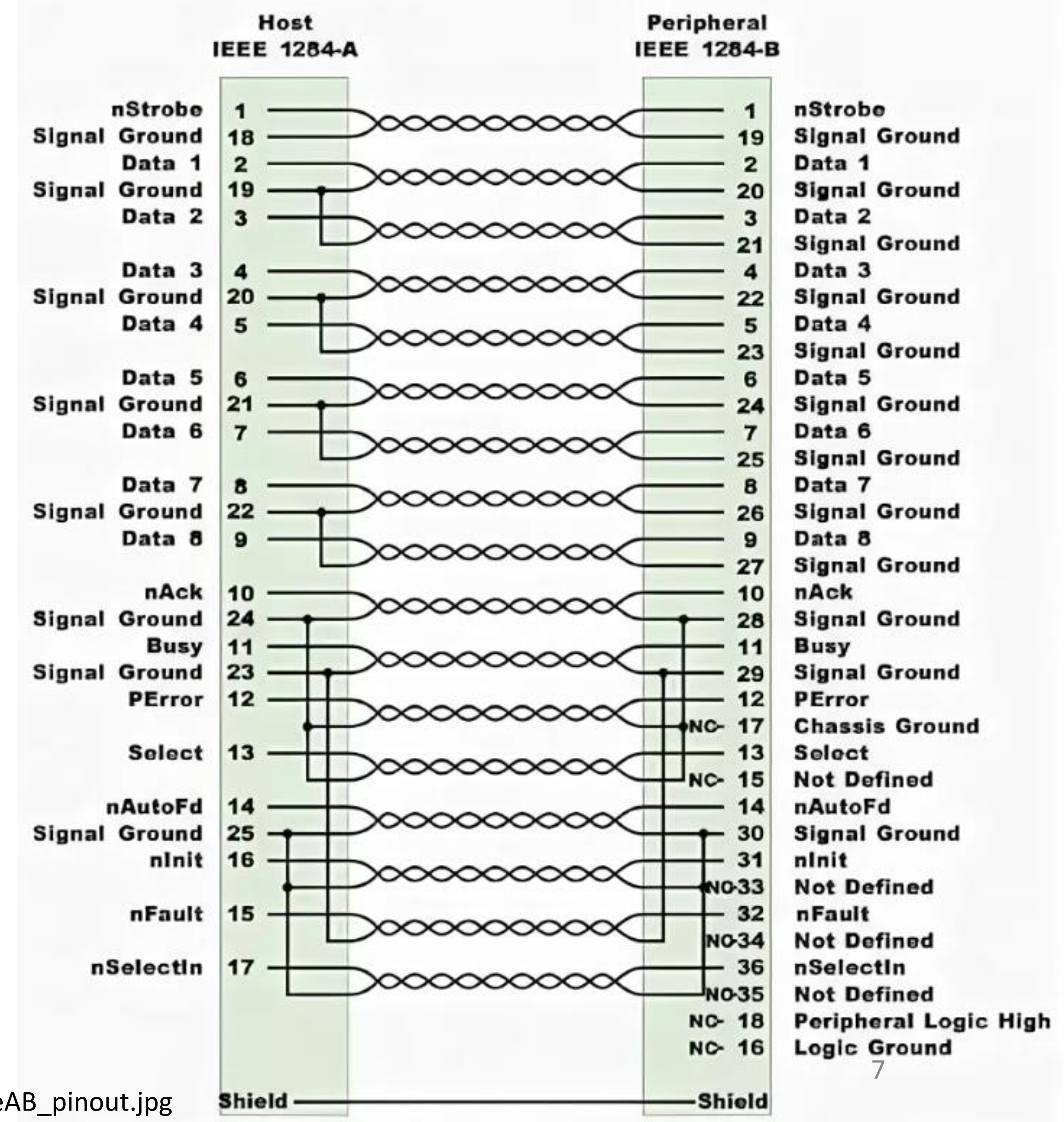
Data interfaces:

- **Parallel interface** uses multiple data lines to simultaneously send bits from point to point
  - IEEE-1284 printer port

They did not find spread use in embedded systems. Compared to serial interfaces, parallel are:

- Conceptually, faster, but not in practice;
- More prone to noise;
- Challenging to maintain synchrony across wires;
- Bulkier, heavier and more expensive.

#IEEE-1284 printer port wiring [4]:



[4] Wikipedia, IEEE1284 Printer Pinout, https://de.wikipedia.org/wiki/Datei:IEEE1284-PrinterCableAB_pinout.jpg

CAN（Controller Area Network）是一种用于汽车和工业控制领域的高可靠性、实时性的通信协议。它提供了一种高效、低成本的方法来实现设备间的通信和数据交换。CAN通信网络由多个节点（CAN Node）组成，这些节点可以是微控制器、传感器、执行器或其他嵌入式设备，它们通过CAN总线连接在一起。

I/O Hardware

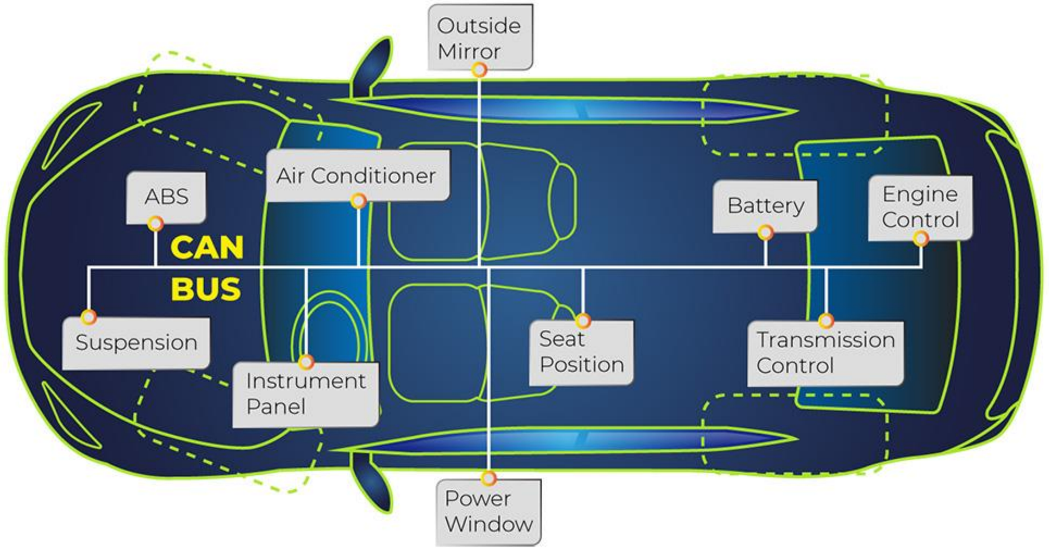
Data interfaces:

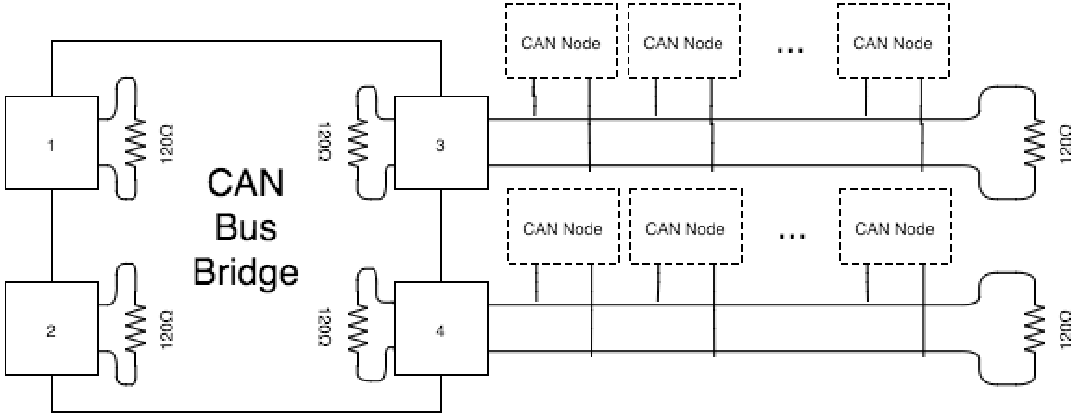
- **Bus** is an interface shared among multiple devices (in contrast to a point-to-point connection):
  - **Memory bus** connects **cache** with external **RAM**;
  - **Peripheral buses** (**SCSI, ISA, PCI, ATA**) in computers connect video cards, sound cards and disk drives;
  - Industry buses (Ethernet, PROFIBUS, CAN, etc.).

Communication protocols provide rules for data transmission between devices:

- **Media-access control (MAC)** protocol to arbitrage access to the bus:
  - Master-slaves technology;
  - Time-triggered bus;
  - Token ring
  - Bus arbiter;
  - Carrier Sense Multiple Access (CSMA).

CAN bus illustration [5]:





[5] Can bridge, https://github.com/russery/can_bridge/blob/master/README.md

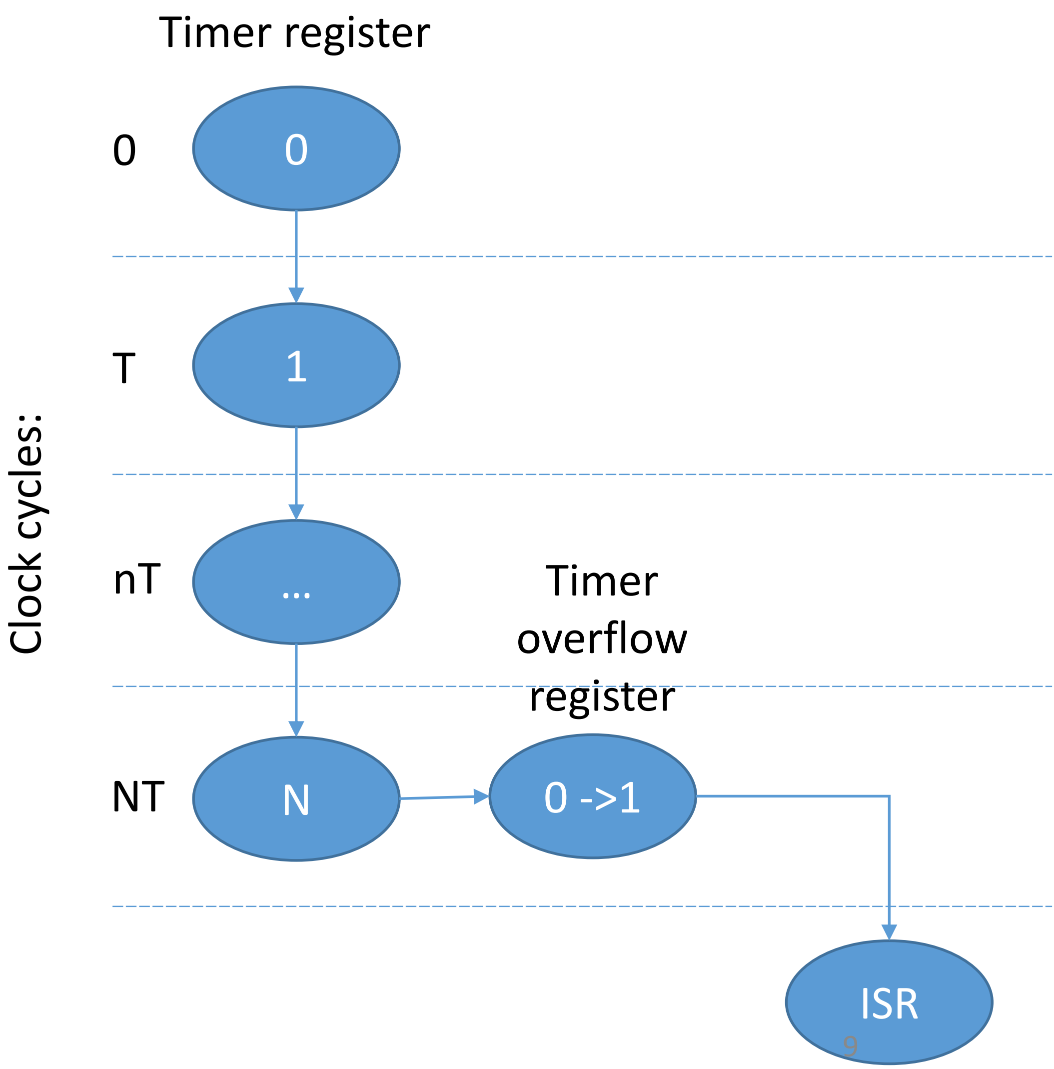
Mechanisms to interact with the external world

Timing and reaction are vital for most embedded applications!

Most (if not all) controllers provide hardware and software mechanisms for immediate reaction to an external signal and precise timing:

- **Interrupts** pause execution of a program for a pre-defined code sequence, i.e., **Interrupt Service Routine (ISR)**. Three kinds of events to invoke ISR:
  - Software – called within currently executed software and/or timer-counter by writing to a MMR;
  - Hardware – called by external signal through an interrupt request line (i.e., specific pin).
  - Exception – internal hardware detects fault (eg, segmentation fault).
- eg, every 12 cyc e as in 8051 architecture);
- **Counters** count external events (i.e., voltage level change on a specific pin) and increment or decrement its MMR.

Interrupt triggered by the **timer**:



Timers and counters

Hardware and software implementation of timers is up to a manufacturer. For details, users are referred to µC documentation.

#Let’s consider the manual for a 8051µC [6]:

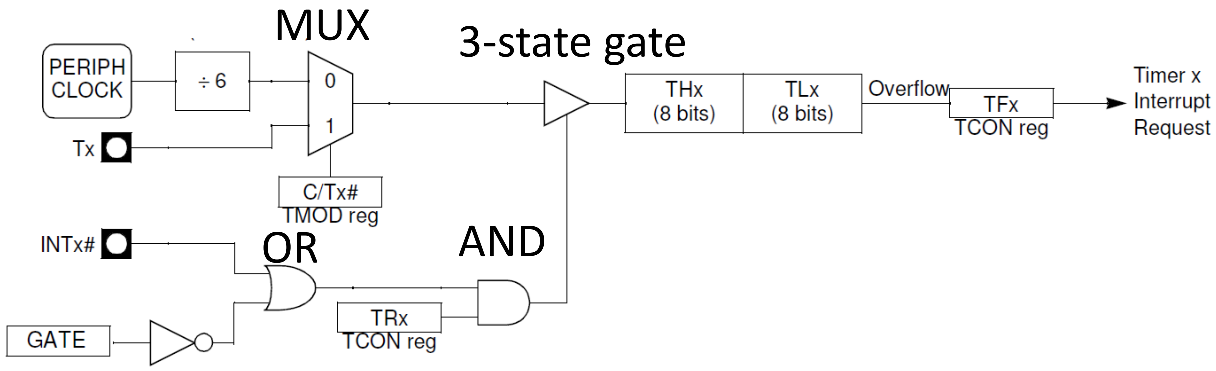
- Two GP 16-bit timers/counters (T0 and T1):

TH0/1	TL0/1

- Each of them has 4 modes (**M0-3**)
- They are configured with the register **TMOD**:

(MSB)			Bits				(LSB)
7	6	5	4	3	2	1	0
Gate	C/T	M1	M0	GATE	C/T	M1	M0
Timer/Counter 1				Timer/Counter 0			

Timer/Counter Mode 1 operation logic [6]:



#Configure and run timers and counters in C:

1. TMOD = 0x61; // Set Timer0 to M1, Counter1 to M2
2. TH0 = 0x3C; TL0 = 0xAF; // Set initial value to 15535;
3. TR0 = 1; // Start Timer0
4. TR1 = 1; // Start Counter1

- Controlled with the register **TCON**:

(MSB)			Bits				(LSB)
7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer/ Counter 1		Timer/ Counter 0		Interrupts control			

[6] Atmel 8051 Microcontrollers Hardware Manual, https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/UserGuides/doc4316.pdf

Interrupt controllers

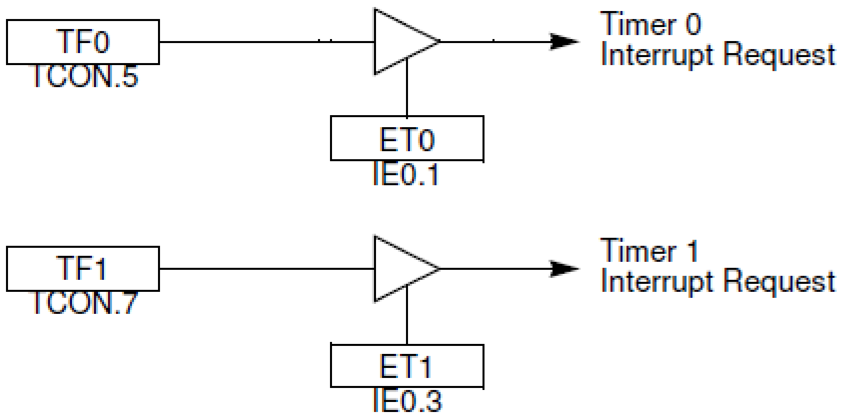
Continuing with the manual for a 8051µC [6], each timer/counter handles one interrupt.

- Interrupts controlled with four **LSB** bits in **TCON** (prev. slide) – internal/external (on/off);
- Interrupts are configured with the register **IE**:

(MSB)				Bits			(LSB)
7	6	5	4	3	2	1	0
EA	X	X	ES	ET1	EX1	ET0	EX0

where EA enables interrupts, ES serial port interrupt, ETx set interrupt by timer x, EXx set external interrupt.

Timer interrupt operation logic [6]:



Configure interrupt for Timer0 in C:

```
…
void main(void){
  IE = 0x82; // Enable and set interrupt for Timer0
  while(1){ // Do something loop
     …}}

void ISR_name(void) interrupt 1{
  TH0 = 0x3C; TL0 = 0xAF; // Reset timer init value;
  … // Other instructions of ISR
  }
```


[6] Atmel 8051 Microcontrollers Hardware Manual, https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/UserGuides/doc4316.pdf

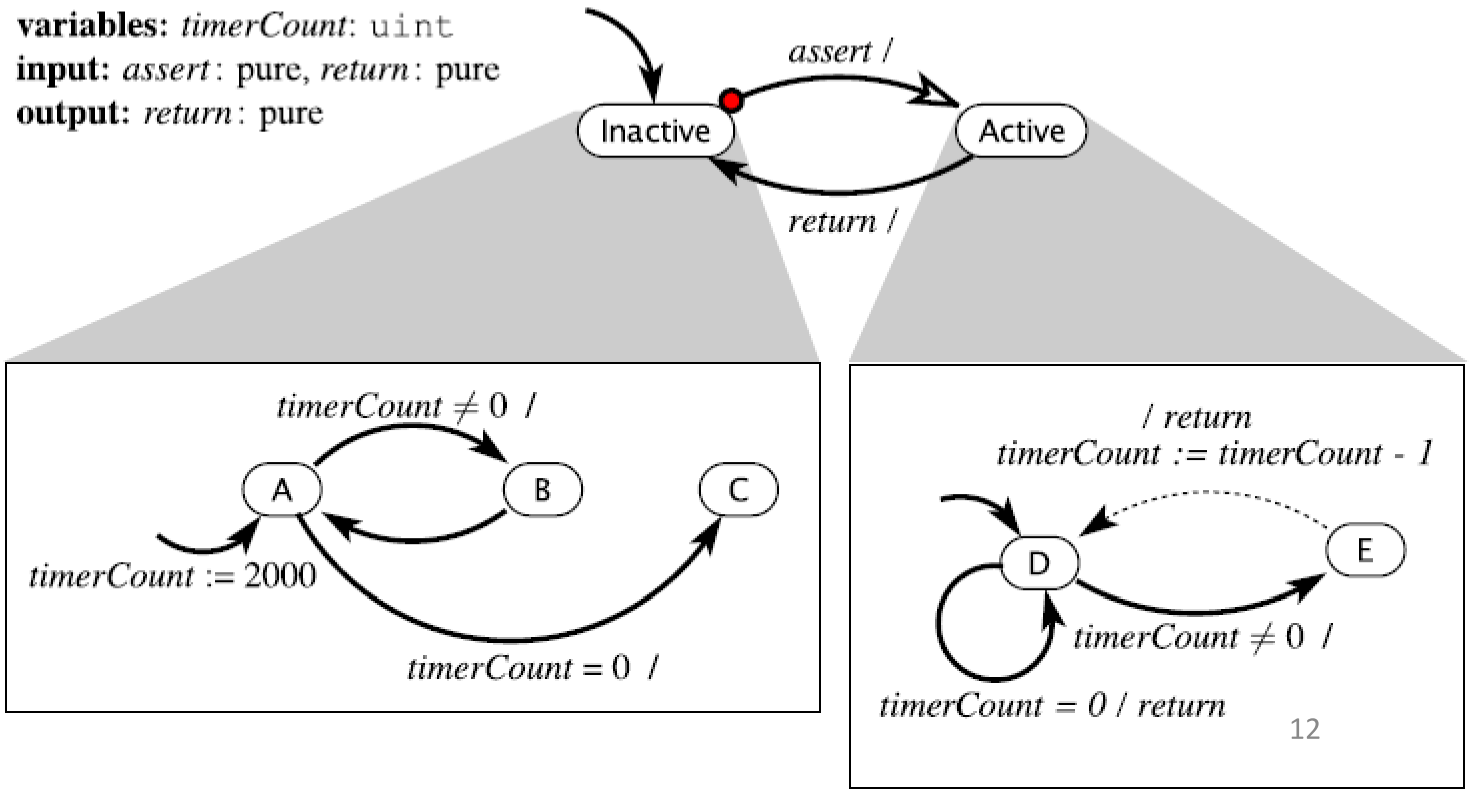
Modelling interrupts

To avoid unexpected behavior - it is important to think about atomicity of the instructions that could be interrupted and variables shared between functions!

Interaction between an ISR and the main program can be represented with Finite State Machines (FSM).

#Consider the following code and its hierarchical FSM:

```
    volatile uint timerCount = 0;
    void ISR(void) {
        … disable interrupts
D→      if(timerCount != 0) {
E→          timerCount--;
        }
        … enable interrupts
    }
    int main(void) {
        // initialization code
        SysTickIntRegister(&ISR);
        … // other init
A→      timerCount = 2000;
B→      while(timerCount != 0) {
         … code to run for 2 seconds
        }
    }
C→  … whatever comes next
```



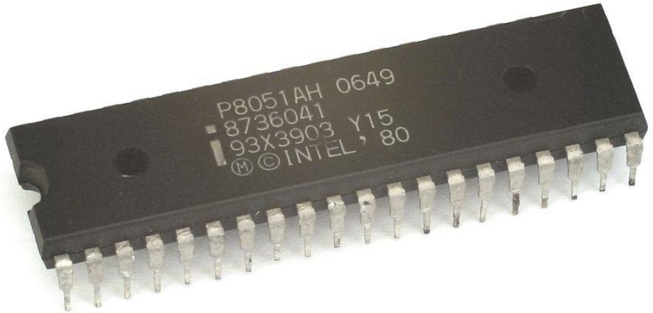
8051 Architecture µC

Technical specification:

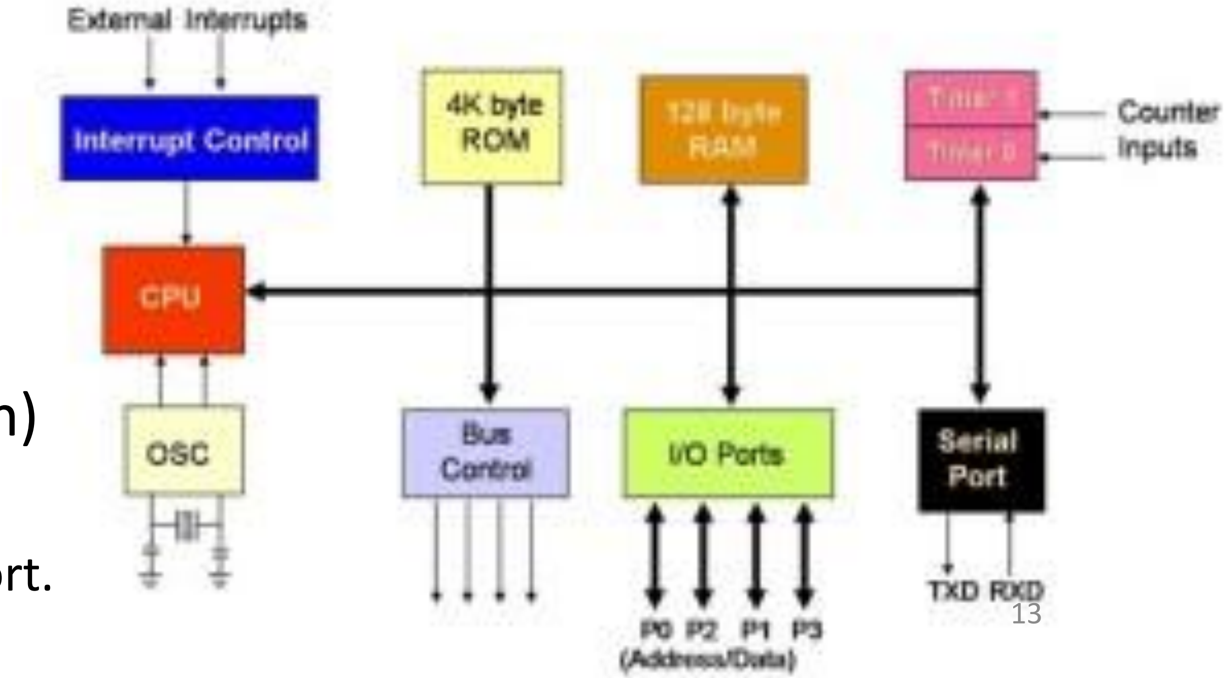
- 8-bit CPU with two Registers;
- On-chip oscillator – 12 MHz clock;
- 32 GP I/O (4 groups of 8 pins – P0-P3);
- 4KB ROM for program instructions;
- 128B RAM:
  - 64B special registers (i.e., I/O, timers and peripherals);
  - 64B for operational needs;
- Two 16-bit timers/counters:
  - Timers increment every 12th clock cycle;
- Serial port: Tx and Rx (duplex transmission)
- Five interrupts:
  - 2x peripheral (ext.); 2x timer (int.); 1x data port.

五个中断源：两个外部中断，两个定时器中断，一个串行通信中断

Physical realization – 40-pins DIP:



Block diagram of 8051 µC:



8051 Architecture µC

Programming 8051µC:

- Software (IDE) – **Keil uVision 5**:
  - Officially available online for download (Windows only);
  - In dedicated computer rooms on campus (SD546, SD554, SC375, CBG13);
- Language – Embedded C:
  - Uses syntax and semantics of the C language / global variables not allowed;
  - Additional support: fixed-point arithmetic; named address spaces; I/O hardware addressing.
  - Every program starts with the declaration of 8051µC registers: `#include <REG51F.H>`
  - Most of the programs (i.e., main functions) run in infinite loop:

```
void main(void){
    while(1){…
}}
```

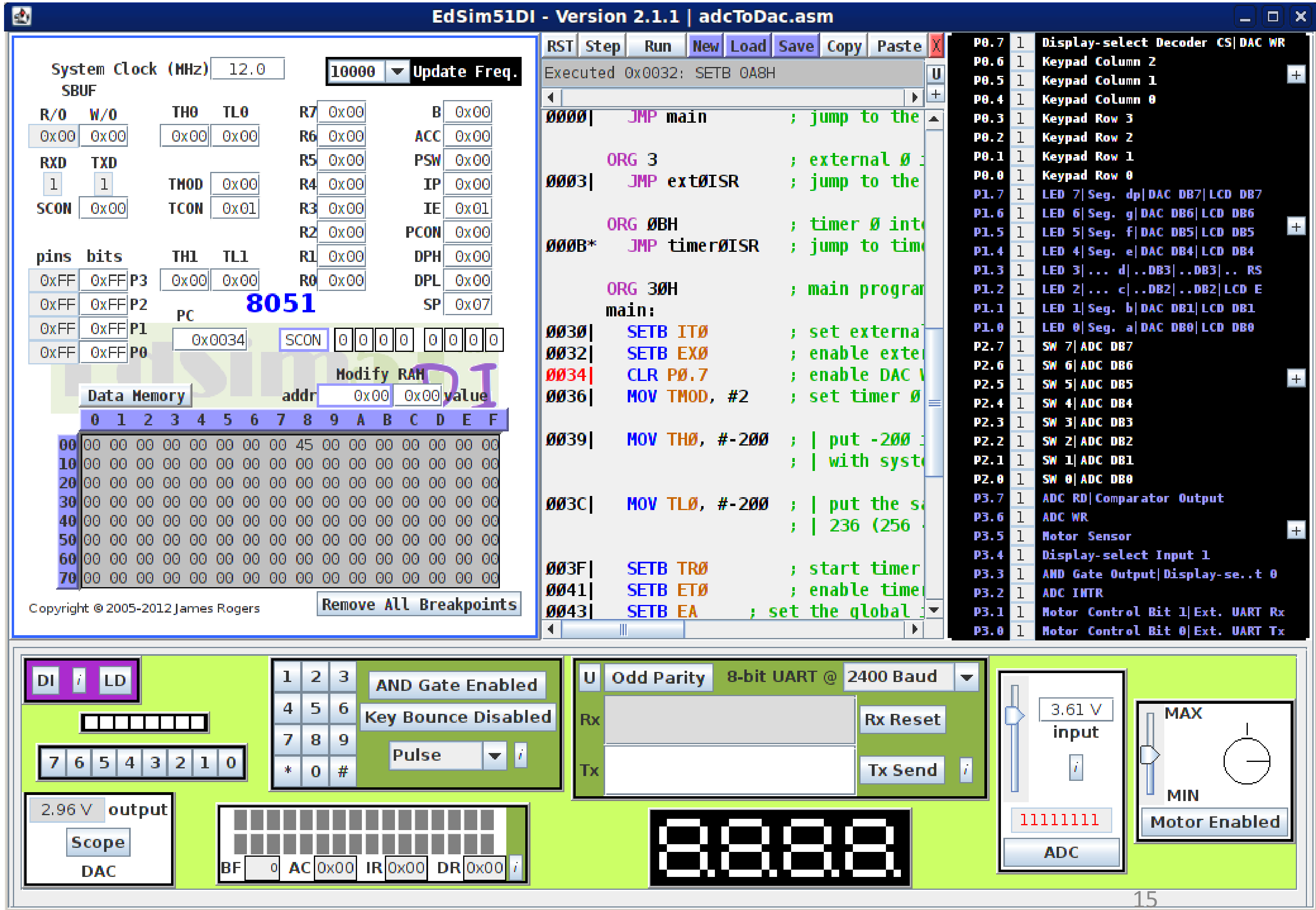
  - Custom functions can be called from the main function or by the interrupts:

```
void func_name(void) interrupt 1{
    …
}}
```

8051 Architecture µC

8051µC emulator:

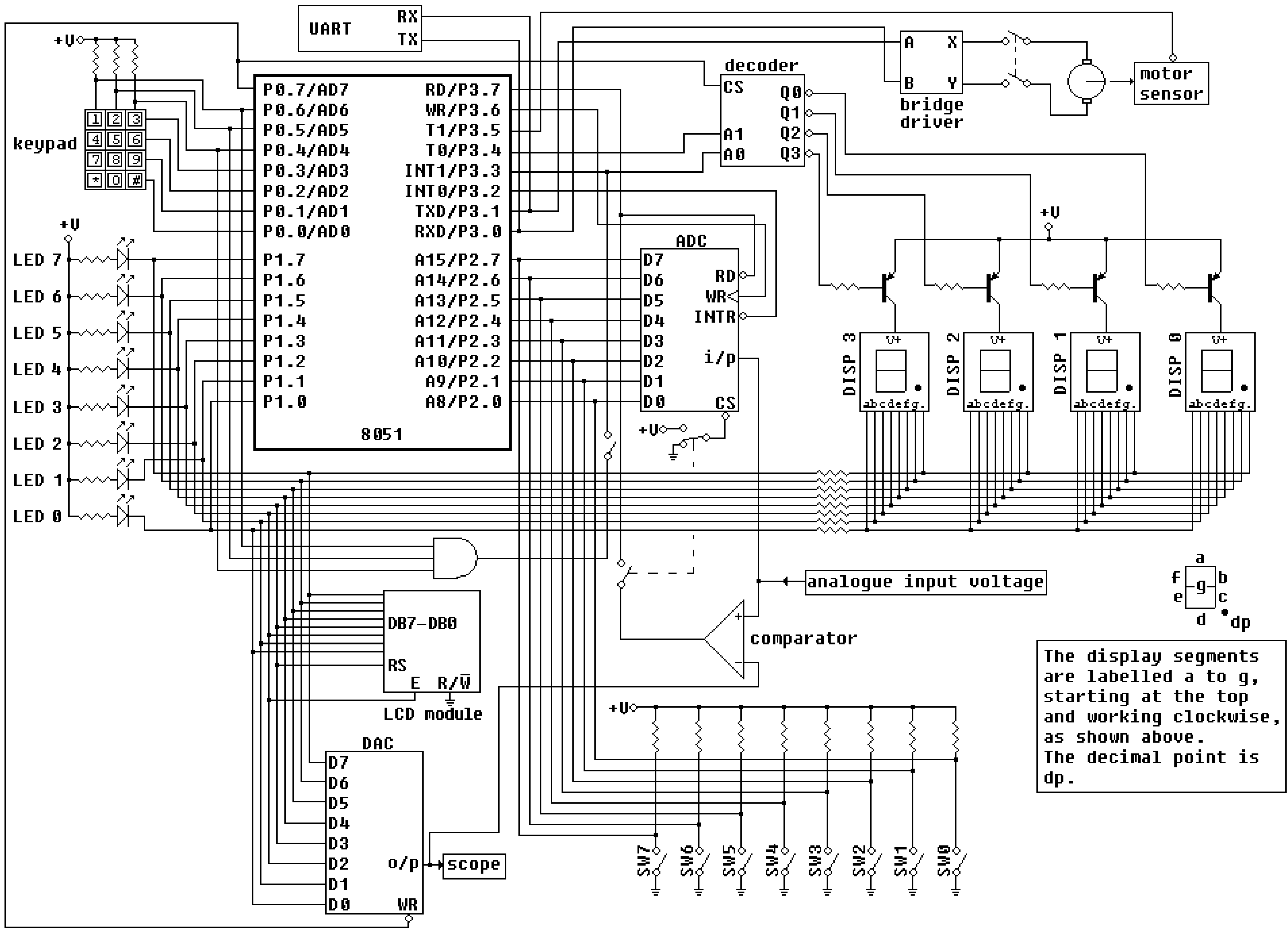
- Software – EdSim51di:
  - Officially available online for [download](#) (Windows only) – Java program;
  - In dedicated computer rooms on campus – can downloaded from the link above.



8051 Architecture µC

EdSim51di peripheral logic diagram:

- Analogue-to-Digital Converter (ADC)
- Comparator
- UART
- 4 Multiplexed 7-segment Displays
- 4 X 3 Keypad
- 8 LEDs
- DC Motor
- 8 Switches
- Digital-to-Analogue Converter (DAC) - displayed on oscilloscope



To sum up

- I/O hardware bridges physical and cyber (computer) worlds;
- µCs mix and match numerous types of I/O hardware:
  - **General-Purpose Digital I/O (GPIO)** – vast application including all below;
  - **Pulse width modulation (PWM)** – deliver variable value (i.e., amount of power) with just a binary state of a signal varying in time;
  - **Serial and parallel interfaces** – provide point-to-point data transfer;
  - **Bus interfaces** – connect multiple devices for data transfer.
- µCs provide hardware and software tools for precise timing and execution of tasks:
  - **Timers** measure internal events (i.e., cycles) for precise timing;
  - **Counters** count the number of external events (i.e., state change of a designated pin);
  - **Interrupts** (almost) instantly response to internal events (timer overflow) or external stimuli to execute predetermined **Interrupt Service Routine (ISR)**.
- **Finite State Machines** can be useful to analyze interactions of the main program and ISR.
- **8051 architecture** accommodates most of the mechanisms available in modern µCs.
- 8051µC can be programmed using **Keil uVision IDE** and tested on **EdSim51di** emulator.

The end!

See you next time – April 3* (optional – mini lecture on Embedded C)

April 10 (next real lecture).

* - Week 8 – reading week.