Design process

Steps of the design process

- Determine the requirements
- Design the system architecture
- Select the OS
- Choose the processor and peripherals
- Choose the development platform
- Code the application and optimize
- Verify the application on the host system (simulate)
- Verify the application on the target system
Microcontrollers

- **Microcontroller** – a microprocessor with the circuitry added. It’s a specialized form of microprocessor
- **Microprocessor** – a processor built on a single silicon chip. Is typically designed to be general purpose
Microcontrollers

- Program memory
- Data memory
- Microprocessor core
- Internal data & address buses
- Further peripheral
- Further peripheral
- Digital I/O
- Analog I/O
- Counters & timers

- Power
- Reset, Interrupt(s)
- Clock
Microcontrollers

- Microcontrollers are designed to be self-sufficient and cost-effective
- Microcontroller has all the features of microprocessor (CPU, ALU, Registers)
- It has also the additional features, such as RAM, ROM, I\O ports, counter etc.
- Program that is executed by microcontroller is stored in its ROM memory
Microcontrollers - programming

Language choice:

- Assembler
  - Writing takes the longest time
  - Smallest binary code
  - Fastest code

- C
  - Writing takes short time
  - Good size of binary code
  - Fast enough
  - Good trade between coding time and operating speed and binary code

- Basic
  - No reason to use
Microcontrollers - problems

When using two types of memory, designer faces speed issues

- On-chip memory – is fast and easy to use.
- External memory – slower to access

When using two types of memory, designer must decide which elements will be placed in which memory, to optimize the operation speed

- On-chip memory can be used as a cache (data is copied from external memory to on-chip just for the operation time)
Microcontrollers - problems

External components

- Some of them need direct memory access (DMA)
- In such case, the microcontroller does the processing, and the external component processes the specialized functions (such as graphical)
Microcontrollers - problems

Real time environment

- System might not be able to wait for microcontroller resources to be ready
- Requests must be processed right away
- For such critical events, the interrupts should be used
Microcontrollers - problems

Power and resources use

- All resources will be competing for data bus
- All resources will consume energy
- For high power consumption, some specialized units should be used.
Microcontrollers - prototyping

Stabilize 5V using LM7805
Microcontrollers - prototyping

Remember about decoupling:

- 0.1uF capacitors:
  - Between VCC and GND
  - Between AVCC and AGND
  - Place them as close to pins as possible
- Place pull-up resistor on the reset pin (prevents chip from resetting caused by floating voltage value)
Microcontrollers - Arduino

Should not be used for a project, because of things not used further during the operation, such as built in programmer.

You can use the Arduino to program your chip.
Microcontrollers - Coding

Code design – good practices:

- Place a lot of comments in your code
  - Comments should answer question “Why I am doing this” instead of “What I am doing”

```c
/*
 * Detect ringing phone line for 30 seconds
 */

Areg = IPort & 1 /* load accumulator */      /* No, REALLY? */
if(Areg != LastTest){ /* Has the line changed? */
```
Microcontrollers - Coding

Code design – good practices:

- Use conventions:
  - Uppercase for macros, constants
  - Variable names conventions, such as Hungarian Convention
- Don’t use meaningless variable names
- Place rough outline for your program (few lines of comments, describing your program functionality in simple steps)
Code design – good practices:

- Break code into functions
- Design the operation of your program into parts, that:
  - Can be implemented separately
  - Can be tested separately
- Avoid ‘spaghetti plate’ program
Microcontrollers - Coding

Code design – good practices:

- When using debugging code, do not delete if afterwards
  - Comment lines
  - Use `#ifdef`
- Avoid complicated expressions even if they work properly
  - Few weeks from the time you write it, you will not know what that code is doing
Microcontrollers - Coding

Code design – good practices:

- Put all `#define` in one place in the code. Preferably the include file.
- Put function prototypes in a header file. This way you can use functions in any order you like.
  - Function prototype = declaration of the function, but only function name, arguments and return type. No body code.
Microcontrollers - Coding

Code design – good practices:

- Use appropriate operations:

```c
TCCR0 = (1<<FOC0) | (1<<WGM00) | (1<<CS00);
```

TCCR0 would be loaded with 0b1100_0001;
Microcontrollers - Coding

Code design – good practices:

- Use appropriate operations:

```
OCR0 = 0x53; // OCR0 match is at 0x53
```
Microcontrollers - Coding

Code design – good practices:

- Use |= and &= to set certain bits

```c
TIFR |= (1<<OCF0); // clear OCF0 interrupt

TIFR = (1<<OCF0); // clear OCF0 interrupt (OOPS!)
```
Microcontrollers - Coding

Code design – good practices:

- Never hardcode any literals, values in your code (non portable!). Things programmers like to hardcode:
  - File names
  - Paths
  - Host names
  - Port numbers
  - IP addresses
  - URLs
  - parameters
Microcontrollers - Coding

Code design – good practices:

- Keep code nice
  - Use proper indentation
  - Use one operation per line
  - Avoid unnecessary blank lines
  - Avoid unnecessary spaces
Microcontrollers - Coding

Code design – good practices:

- Break code into parts, test these parts thoroughly just after they are finished.
- Do not wait with testing when the bigger part of code is done.
  - Most likely it will not work and you spend time for looking for the error in the big part of code.
  - It is easier to find an error in smaller part of code
  - Finding an error in the big code, that is not divided into the parts is usually impossible
Microcontrollers - Coding

Code design – good practices:

- When using any external components – write code for them first
- Test the code for component, ensure it works fine
- Create functions that will provide access to the component, never access components directly from the main code.
Microcontrollers - Coding

Code design – good practices:

- Stack – its size is limited. Be aware when writing the code that uses the stack.
- Heap – its size is limited. Be careful while allocating the memory.
  - Fragmentation can be an issue for more excessive memory use.
  - Avoid manual memory allocation
Microcontrollers - Coding

Code design – good practices:

- For highest/specific optimization, use assembly inline
- Never use GOTO
```cpp
#include <iostream>

using namespace std;

int f = 100;
int q = 3;
int return_value = 0;
int f1(int, int);
int f2(int, int);

int main() {
    int x = 100;
    int y = 3;

    // output = (x*y) (x+y)
    int output = multiply( multiply( x, y ), add( x, y ) );

    cout << "RESULT = " << output << endl;

    return 0;
}

int multiply(int left, int right) {
    return left * right;
}

int add(int left, int right) {
    return left + right;
}
```