EE443 - Embedded Systems

Laboratory Note
Atmel Microcontroller Introduction

Contents:

L.1 ATmega328 Device Overview

- **Atmel AVR 8-bit Microcontroller.**

  CPU has 32 registers and all registers can work like an accumulator (operation result is written back to the register).

  Three 16-bit registers can be used for indirect addressing.

  Effective execution time is 1 or 2 clock cycles for most of the instructions.

- **Separate memory blocks for program and data storage (Harvard architecture).**

  32 KByte FLASH program memory: Organized as 16K x 16 bits since all AVR instructions are either 16 or 32 bit wide.

  2 KByte SRAM data memory: 32 CPU registers, 64 I/O registers, and 160 extended I/O registers are mapped to the first 256 bytes of the SRAM address space.

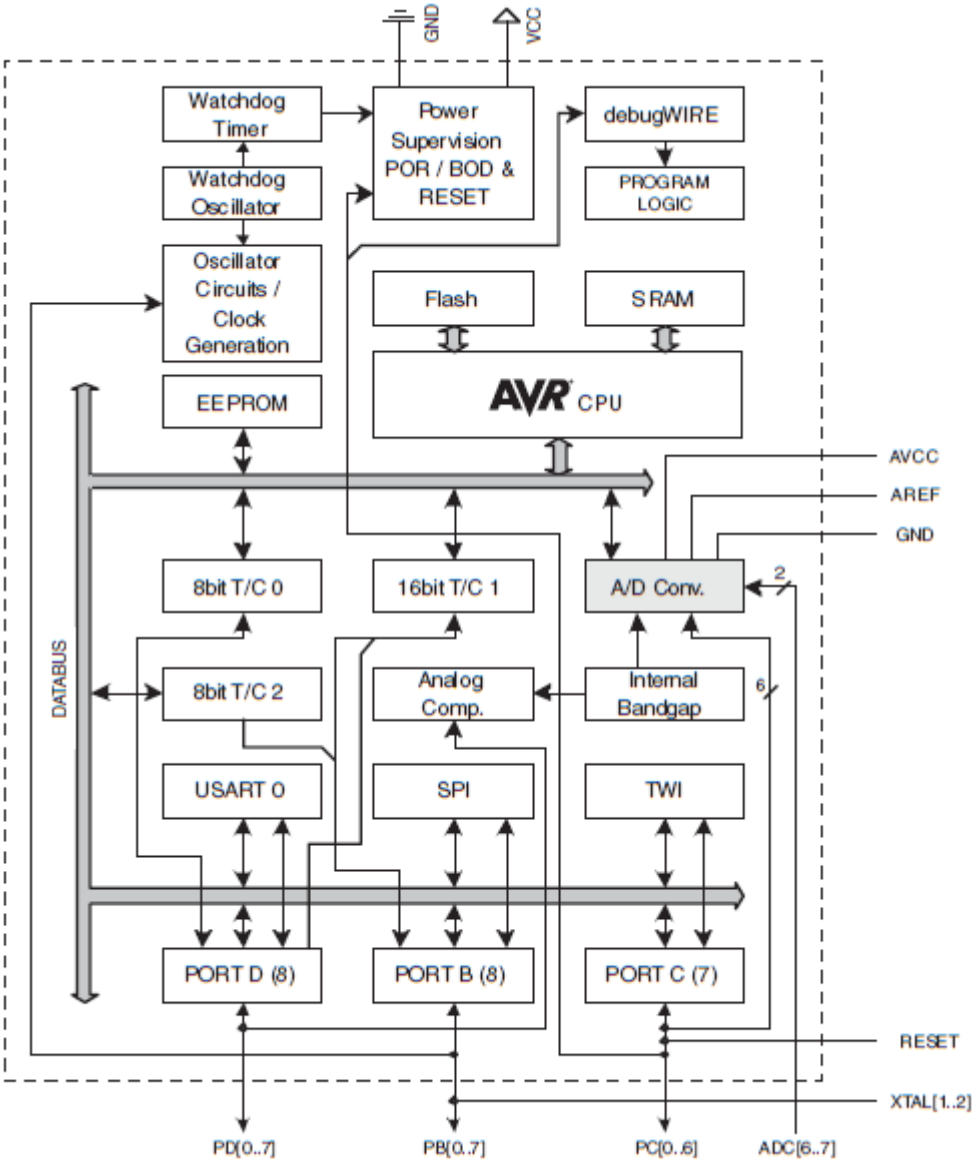
  1 KByte EEPROM: Accessible as a peripheral unit.

- **All peripherals are accessible through Special Function Registers (SFRs).** Atmel datasheet uses the term ***"I/O register"*** instead of *"Special Function Register"*.

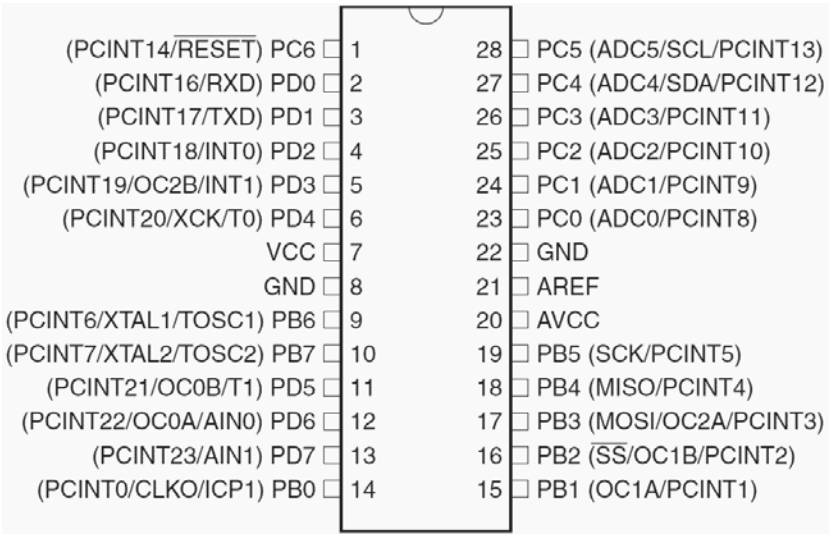
Timers: Two 8-bit and one 16-bit timers with compare and capture modes, programmable watchdog timer.

Serial I/O: Programmable USART, master/slave SPI, and two-wire (I^2C) serial interface.

Analog interface: 10-bit ADC with 6 input channels, an analog comparator, and six PWM outputs.



As usual, multiple functions are assigned to all I/O pins. For example, if a stable crystal oscillator is required then the port-B pins 6 and 7 cannot be used for any other purpose.



L.2 Memory Organization

L.2.1 Program Memory

Program Memory: 16 K x 16-bit, total of 16 K program words in the address range 0x0000 through 0x3FFF.

Reset Vector: at address 0x0000 (target address after a reset condition)

Interrupt Vector: 25 interrupt targets starting at 0x0002 (see page-65 in the datasheet for the interrupt list). Interrupts at lower target addresses have higher priority.

A small section (256, 512, 1024, or 2048 words) at the end of the program memory is reserved for a boot-loader program.

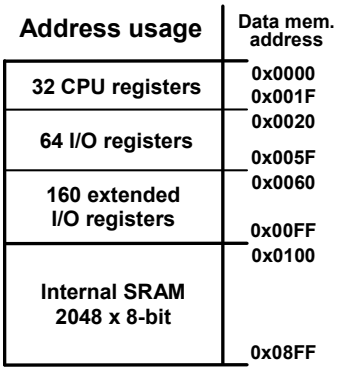
Rest of the program memory can be used for the application program.

The boot-loader program is necessary for in-system programming. The boot-loader reads the new machine code from a serial port and it replaces the application code in the FLASH memory.

Program Counter (PC): 14-bit (to access 16 KWord program memory)

L.2.1 Data Memory

The first 256 bytes (low addresses, 0x0000-0x00FF) of the memory address space are mapped to the 32 CPU registers, 64 I/O registers, and 160 extended I/O registers. The 2 KByte SRAM address range is 0x0100 through 0x08FF. In reality, CPU and I/O registers are all independent of the SRAM module. An address decoder checks the address specified by the CPU and determines the source or target of data read/write operations.



L.3 Instruction Set

Atmel AVR processors have a **RISC** (Restricted Instruction Set Computer) architecture. Instruction set is ***"restricted"***, mainly because the ALU instructions for arithmetic and logic operations cannot specify a memory address. Operands are first loaded into the CPU registers, and then ALU operations are executed on the CPU registers. ALU instructions can specify two registers as operands. For example:

```
ADD  Rd, Rr  // Add Rd and Rr and put the result into Rd
```

Once the data is ready in the CPU registers, execution of ALU operations is very efficient. In a way, the 32 CPU registers work like a small cache memory.

L.3.1 Data Transfer Instructions

These are the LOAD and STORE instructions for data transfers between the CPU registers and the SRAM module using direct and indirect addressing modes. A couple of MOVE instructions are provided to copy data between the CPU registers.

ATmage328 supports immediate, direct, and indirect addressing modes. Direct addressing is restricted to two instructions:

```
STS  <addr>, Rr  // Store Direct to SRAM
LDS  Rd, <addr>  // Load Direct from SRAM
```

Six of the 32 CPU registers are used for indirect addressing. These registers pair up to form three 16-bit index registers labeled as **X**, **Y**, and **Z** registers. There are four different implementations of indirect addressing in load and store instructions:

1. **Indirect:** Load/store data from/to the address given in **X**, **Y**, or **Z** index register.
2. **Indirect with post-increment:** Increment the index register after the load/store operation (useful for quick access to arrays).
3. **Indirect with pre-decrement:** Decrement the index register before the load/store operation (useful for quick access to arrays).
4. **Indirect with displacement:** Add the offset value specified in the instruction to the address given in the index register (useful for access to arrays of structured data blocks).

Z register has an additional capability to address the data stored in the FLASH program memory with **`LPM`** and **`SPM`** instructions. This feature is utilized for in-system programming and to access the permanent tables and other data stored in the FLASH.

Any part of the 2 KByte SRAM data memory can be used for PUSH and POP stack operations. The Stack Pointer is accessed as two 8-bit registers in the I/O space.

L.3.2 Arithmetic and Logic Instructions

Arithmetic operations: Negation (1's or 2's complement), addition, subtraction, and multiplication.
Logic operations: AND, OR, and exclusive OR.
Bit manipulation: Logical shift left/right, arithmetic shift right (keep the sign bit), rotate left/right through carry, set/clear single bit in I/O registers.

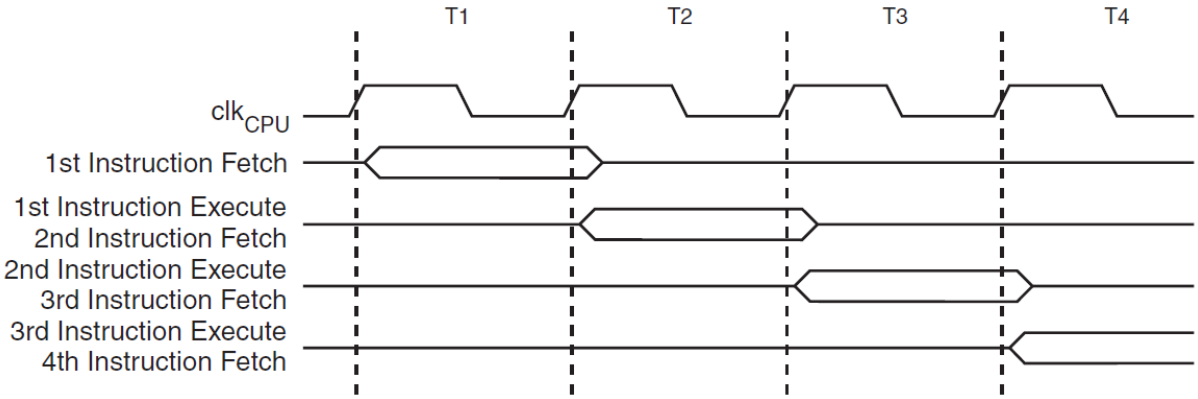
Immediate operands can be used with addition, subtraction, and logical AND/OR operations. **Add/subtract-with-carry** instructions support arithmetic operations on 16-bit and longer operands.

L.3.3 Branch Instructions

Direct, relative, and indirect (through Z register) jump/call instructions are available. A set of relative branch instructions are provided for conditional branching.

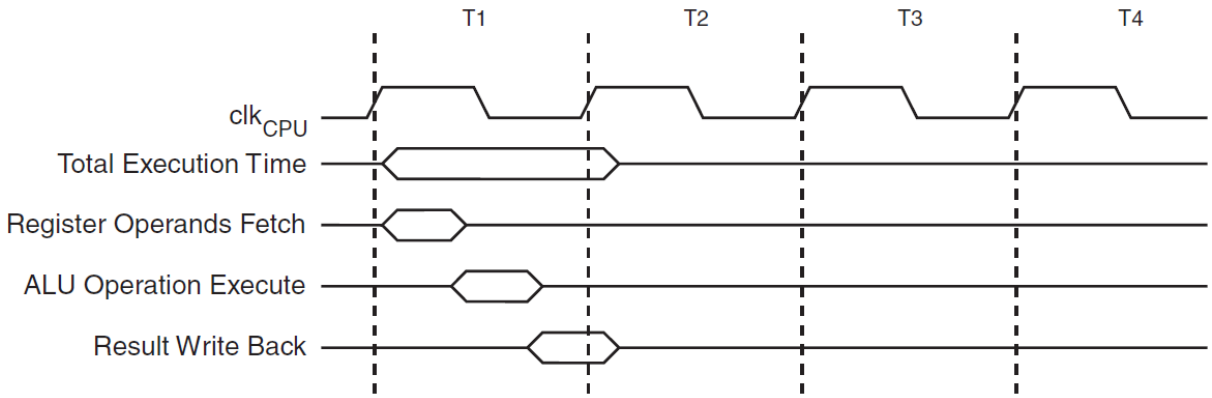
L.3.4 Instruction Timing

Most of the instructions are executed in 2 or 3 clock cycles. The effective execution time is reduced to 1 or 2 clock cycles as a result of two-level pipelining. The next instruction is fetched while the last loaded instruction is being executed.



See the instruction set summary in the datasheet to find out effective execution time for all instructions.

ALU operations are completed in a single clock cycle:


T1
T2
T3
T4
clk CPU
Total Execution Time
Register Operands Fetch
ALU Operation Execute
Result Write Back

L.5 Programming Support

You need to obtain the machine code of your program before you can execute it on actual hardware or simulate it on a design tool such as Proteus. The most common file format for microcontrollers is the Intel HEX format. First, you should setup a development environment following the steps given below. It may be helpful if you read the section titled "Tips to Avoid Trouble" at the end of this laboratory note before you start.

1. Get The Compiler:

WinAVR is a collection of open source software development tools for the Windows platform. WinAVR includes the GNU GCC compiler for C and C++ supporting Atmel AVR series processors. All development tools, including the Arduino software, rely on the WinAVR compiler. You can download WinAVR at:

`http://winavr.sourceforge.net/`

2. Get The Development Environment:

Code::Blocks is an open-source cross-platform Integrated Development Environment (IDE) that supports multiple compilers. Its installation package contains the GCC compiler for the Windows platform. You can immediately start writing programs that will run on your PC. Code::Blocks can also be used for creating programs and applications for AVR, ARM, and many other platforms. Code::Blocks provides:

- A text editor with syntax highlighting for C and C++ programs.
- A project management tool that keeps track of the source files and other resources in a software development project.
- An easy interface to the WinAVR compiler. This interface activates the GCC AVR compiler specifying all source components of a project and the target locations and file names for the object code and executable files that will be generated by the compiler.

You can easily introduce the WinAVR compiler to Code::Blocks (if it is not done automatically) that supports program development for Atmel AVR processors. If you need an ARM compiler in the future, then you can download the GCC ARM compiler and use Code::Blocks as a convenient interface. You can download Code::Blocks at:

`http://www.codeblocks.org/downloads`

A similar open-source cross-platform IDE is **Orwell Dev-C++**, that can be downloaded at:

`http://sourceforge.net/projects/orwelldevcpp/`

3. Setup The Development Environment:

Install **WinAVR** and **Code::Blocks** on your computer. The default compiler of Code::Blocks is the GNU GCC Compiler that can be used for the programs written for

Windows environment. You will need to follow a few configuration steps to add WinAVR as a compiler option for your projects.

Adding WinAVR compiler:

1. Select the **Compiler...** option under the **Settings** menu.

2. Use the **Global compiler settings** option on the left.

3. Select the **GNU AVR GCC Compiler** in the pull-down list.

4. Click on the **Toolchain executables** tab

5. Under the Compiler's installation directory select the directory where **WinAVR** is installed.

6. In the **Program Files** tab, make sure that the following executables are listed:

C compiler: **`<WinAVR installation dir>\bin\avr-gcc.exe`**

C++ compiler: **`<WinAVR installation dir>\bin\avr-g++.exe`**

Linker for dynamic libs: **`<WinAVR installation dir>\bin\avr-g++.exe`**

Linker for static libs: **`<WinAVR installation dir>\bin\avr-ar.exe`**

Debugger: **`GDB/CDB debugger : default`**

Resource compiler: *`leave empty, no need for a resource compiler`*

Make program: **`<WinAVR installation dir>\utils\bin\make.exe`**

7. In the **Additional Paths** tab, add the following directory where make.exe is located.

`<WinAVR installation dir>\utils\bin`

8. Click on the **OK** button.

4. Create an AVR project in Code::Blocks IDE:

1. Click on the **Create a new project** option.

2. Select the **AVR Project** option.

3. Enter the **project title** and select the folder where the project folder will be created.

4. Uncheck the **Create "Debug" configuration** box.

5. Choose the processor, **atmega328p** (or **atmega2560** for Arduino Mega), and define **F__CPU** as **8000000UL** for Proteus simulations (or **16000000UL** for Arduino Uno and Mega boards). The CPU clock frequency is critical for proper execution of the time delay macros (i.e. `_delay_us(...)` defined in **<util/delay.h>**). Make sure that "**Use external memory**" box is unchecked and "**Create hex files**" box is checked.

The executable code generated by WinAVR should be written into an output file with the "**hex**" extension. This output file is a plain text file that contains the byte-by-byte hexadecimal programming data in Intel hex format.

6. Click on the **Finish** button.

7. Add/remove source files using the **Project** menu. You should keep the source file, **fuse.c**, just to avoid build errors. The fuse settings generated by the compiler

will be ignored in Proteus and programming of fuse bits on the Arduino board is not possible through USB interface. In Proteus, CPU clock and the other necessary parameters for simulation can be entered as device properties on the schematic.

L.5 Connecting With Hardware

Peripheral units are accessible through the I/O registers. The following program initializes all pins of Port-B of ATmega328 for output and then toggles the output pins at every 500 ms.

```
#include <avr/io.h>
#include <util/delay.h>  // required for _delay_ms(.) macro

int main(void)
{
DDRB = 0xFF;  // set all pins of PORTB for output
while(1)         // repeat forever
{ PORTB = 0x0F;  // set PB[3:0] high
  _delay_ms(500);
  PORTB = 0xF0;  // set PB[7:4] high
  _delay_ms(500);
};
return 0;
}
```

In this program, **DDRB** is the Port-B Data Direction Register and **PORTB** is the Port-B Data Register as they are described in the ATmega328 datasheet. The WinAVR compiler comes with an include file that defines the necessary address constants and macro operations for every AVR processor model. The include file used for ATmega328 is **iom328p.h**. This support provided by the compiler saves a lot of time while programming a specific processor model.

At the beginning, it may seem overwhelming to manage a microcontroller with 100 special function registers. After you gain a little experience, handling SFR access becomes a straightforward task.

L.6 Proteus Overview

Proteus design suite is an electronic design tool that provides three main features for the development of electronic hardware:

1. Schematic Capture With ISIS:

You can draw circuit schematics using the ISIS design environment. There are a wide variety of circuit components available in the libraries provided with ISIS. You can place the selected components on the schematic sheet and make the simple electrical connections. If necessary, new components can be defined.

2. Circuit Simulation:

Circuit behavior can be observed on the graphs or animated display components such as LEDs. Ideal DC power sources or signal sources can be placed on the schematic to support the simulation process. SPICE algorithm is used to simulate the analog circuit behavior. Functional models are also available for simulation of complex digital components such as microcontrollers and programmable digital circuits.

3. Printed Circuit Board (PCB) Layout:

PCB layout can be made based on the circuit schematic and the component footprints specified for each component in the schematic. Unlike the schematic drawings that have simple pins, components placed on the PCB layout has pads with geometric shapes selected according to the device package.

Installing Proteus Software

1. First install version 7.8 by running **setup78.exe**.
2. Copy the license file, **licence.lxk** into the Proteus installation directory.
3. Run **setup710.exe** to upgrade to version 7.10.
4. Install the service-pack-1 by running **update710_SP1.exe**.

Following are the usage tips that will be helpful in creating simple circuit schematics and displaying simulation results.

Place a component:

Right click > Place > Component > ...
Menu: Library-> Pick Device/Symbol

Place a ground terminal:

Right click > Place > Terminal > Ground

Place a power terminal:

Right click > Place > Terminal > Power

Place a voltage/current source:

Right click > Place > Generator > ...

Place a voltage/current probe:

Right click > Place > Voltage Probe or Current Probe

Display simulation results on a graph:

Right click > Place > Graphs > ...
Right click on Graph > Edit Graph > Enter stop time
Right click on Graph > Add Traces > ... (one node at a time)
Right click on Graph > Simulate Graph

Notes on Using Proteus:

- The display of animated components on the screen may not reflect the actual execution speed. Use the simulation time index provided at the bottom of the design window for time measurements, or plot the waveforms on graphs.
- Check the messages after assembly process or simulation. If the number of messages increases continuously during simulation, then this indicates repeated

error or warning messages. The simulation efficiency drops if a large number of messages are created and stored by Proteus.

L.7 Tips to Avoid Trouble

Following is a list of suggestions that will help you keep track of your files.

1. Do not use any special characters (blank, dot, dash, semicolon, etc.) in directory and file names even if they are allowed by the operating system. English letters, digits, and underscore (_) are the valid characters. Other characters may cause problems when you try to access a design tool or a project file using Windows command interface. For example, the complete file path cannot be identified if a directory or file name contains a blank, unless the entire file path is enclosed between quotation marks. Turkish letters (ç, Ç, ğ, Ğ, ı, İ, ö, Ö, ş, Ş, ü, Ü) may cause trouble when you copy or send files to another computer although they look fine on your computer.

2. Do not install the design and programming tools in the "Program Files" directory because its name contains a blank. Make a new directory such as "EEtools" and install all professional software in that directory avoiding any special characters.

3. Do not place your project on your desktop (masaüstü), in "My Documents", or any similar directory that is actually located somewhere deep in the C: drive. The actual path information will be too long and it will contain blank characters. A directory path, such as

"C:\Users\<user name>\My Documents\EExxx\Project1\" or

"C:\Documents and Settings\<user name>\Desktop\EExxx\Project1\"

is not completely visible in window titles or project file listings. If you cannot identify the complete file path, then you may end up editing a different set of files while the compiler or the design tool uses the files in another directory. You should go to your user partition and make an easily accessible directory, such as

"D:\EExxx\Project1\"

4. Make at least two partitions when you initialize your hard disk next time. One partition is for installation of operating system and other executable programs and the second partition is for user files (pictures, music, homework, projects, etc.). Most of the malicious programs, such as viruses and spyware, invade the system partition. Next time your computer gets an infection; you can format the system partition only, and leave your user partition intact. It is advisable to have a third partition to keep the backup copies of setup/installation files of the design and programming tools and other application programs. You can use these backup copies to install the applications easily, without downloading them again.