

Academy Profession Degree
in
IT Network and Electronics Technology
at
KEA, Københavns Erhvervsakademi,
Copenhagen School of Design & Technology

Subject: Programming and Microprocessor
Semester: 1
Module: 2, Introduction to the microprocessor

Target:

The module will give an introduction to the microprocessor as a general device. The following subjects will be covered:

- The internal design of the microprocessor
- The installation and use of an IDE – Integrated Development Environment
- Design and coding of simple embedded programs
- The compilation, downloading and debugging of programs
- The use of the I/O facilities of the microprocessor
- Construction of external hardware for the processor system
- Programming of state machines

A number of exercises will be made in order to gain understanding of the practical use of the microprocessor. Emphasis will be placed on the understanding of the general development flow from the first idea of a microprocessor-based device to the final operating and debugged version.

Version: July 17th, 2011 by elr@kea.dk

Contents

1	The Microprocessor	3
1.1	Features	4
2	The Atmel ATmega32 kit	6
3	The IDE – Integrated Development Environment	8
4	C coding: Controlling individual bits on the ports.....	11
5	Programming State Machines	12
6	Embedded μ P exercises.....	13
6.1	Embedded Exercise: LED 1	13
6.2	Embedded Exercise: LED 2	13
6.3	Embedded Exercise: I/O	13
6.4	Embedded Exercise: Encoder	13
7	Literature	14

1 The Microprocessor

- A microprocessor (μ P) is a sequential digital circuit.
- A crystal oscillator controls the speed at which the μ P is running = stepping through the sequence. A clock generation circuit will generate the required clocks for the μ P based on the crystal.
- The main purpose of the μ P is to execute a program.
- A program is a collection of instructions to the μ P. These instructions will be executed sequentially in the order they are written.
- The instructions are machine-codes, which the μ P understands (is able to decode).
- A program will always start at the reset-vector. This is the program memory address which is accessed right after leaving the reset state.
- Five groups of instructions:
 - Arithmetic and logic instructions
 - Branch instructions, conditional and unconditional
 - Data transfer instructions
 - Bit- and bit-test instructions
 - μ P control instructions.
- Programs are usually written in a high-level language like C.
- These are translated into machine-code by a compiler.
- The μ P performs three tasks for each machine-code:
 - Fetch the next instruction from the code memory.
 - Decode the instruction
 - Execute the decoded instruction. This involves typically reading of data, some kind of calculation and the writing of the result.
- A μ P will need a bus-interface in order to communicate with external hardware.
- Relevant external hardware:
 - ROM (Read Only Memory) typically used to store code.
 - RAM (Random Access Memory) typically used to store data.
 - I/O ports used for data input/output to/from external circuits.
- The main building blocks of a μ P:
 - ALU (Arithmetic Logic Unit) for data processing.
 - The Program Counter, used for selecting the next code memory address to read an instruction from.
 - Instruction decoding circuit.
 - Working registers for temporary storage of data at high speed.
- Improvements to standard μ P design:
 - Pipeline for parallel code handling
 - Increased bit-width, go from 8 to 16, 32 or 64 bits pr instruction.
 - Internal memory
 - Use RISC design
- The microcontroller (μ C) is a μ P with added on-chip functionality such as:
 - Timers / counters / pulsgenerators / RTC
 - ADC / DAC / analog comparators
 - Communication: UART / SPI / I2C / CAN / LAN / USB
 - Interrupt: internal and external
 - Watchdog

- JTAG interface
- I/O ports, bit and byte wide
- Memory is typically internal
- The μ C facilities are typically handled by SFR (Special Function Registers).
- We are using Atmel ATmega32A as our standard μ C for all exercises.

1.1 **Features**

The following general overview of the ATmega32 is page 1 from the databook:

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single-clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16 MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
 - 32K Bytes of In-System Self-programmable Flash program memory
 - 1024 Bytes EEPROM
 - 2K Byte Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C
 - Optional Boot Code Section with Independent Lock Bits
- In-System Programming by On-chip Boot Program
- True Read-While-Write Operation
 - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Four PWM Channels
 - 8-channel, 10-bit ADC
 - 8 Single-ended Channels
 - 7 Differential Channels in TQFP Package Only
 - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
 - Byte-oriented Two-wire Serial Interface
 - Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
 - External and Internal Interrupt Sources

- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
 - 32 Programmable I/O Lines
 - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
 - 2.7 - 5.5V for ATmega32A
- Speed Grades
 - 0 - 16 MHz for ATmega32A
- Power Consumption at 1 MHz, 3V, 25°C for ATmega32A
 - Active: 0.6 mA
 - Idle Mode: 0.2 mA
 - Power-down Mode: < 1 μ A

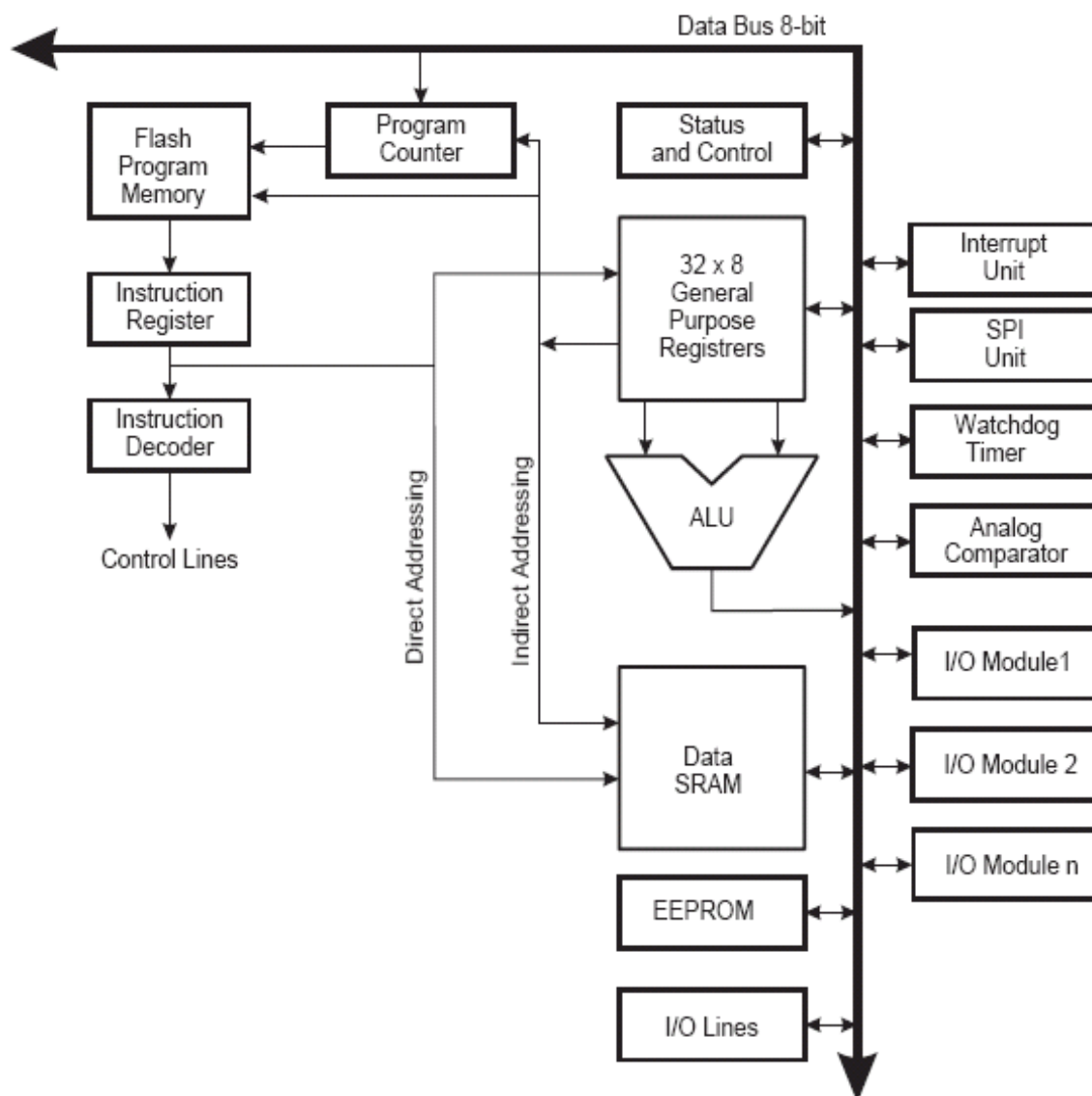


Figure 2, Block diagram of ATmega32 (Atmel figure)

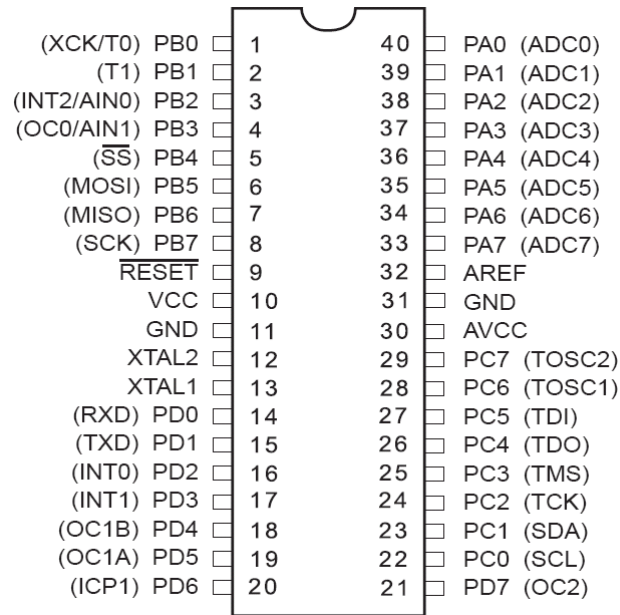


Figure 3, Pin configuration for ATmega32 (Atmel figure)

2 The Atmel ATmega32 kit

A detailed schematic and the PCB layout of the kit can be found at the web (or next) page. The components in the schematic are described in this chapter.

The μP (IC1) has 40 pins in the DIL-version. 32 pins are used for port-pins (General Purpose I/O) and different alternative functions. The remaining 8 pins are used for the system. These are:

- Vcc and GND, power supply (2.7 .. 5.5 V) and ground connection.
- AGND, AVCC and AREF are supply pins for the Analog-Digital Converter (ADC).
- XTAL1 and XTAL2 are input and output for the crystal oscillator circuit.
- RESET will keep the μP in the reset condition when the pin is low.

Alternative functions for the portpins:

- Port A (PAX) is used as 8 input channels for the ADC
- Port B (PBx) is used for SPI communication, analog comparator and timer I/O pins
- Port C (PCx) is used for 32 KHz crystal, JTAG and I2C communication
- Port D (PDx) is used for UART, external interrupts and timer I/O pins

The diode bridge will ensure that the power can be connected without worries about polarity.

The capacitors C6 and C7 are used to filter the incoming power.

IC2 is a voltage regulator. It generates a clean 5 V supply (Vcc).

C4, C5 and L2 is a filter circuit used to generate a smooth reference voltage for the ADC.

C8 is a filter capacitor for the μP supply.

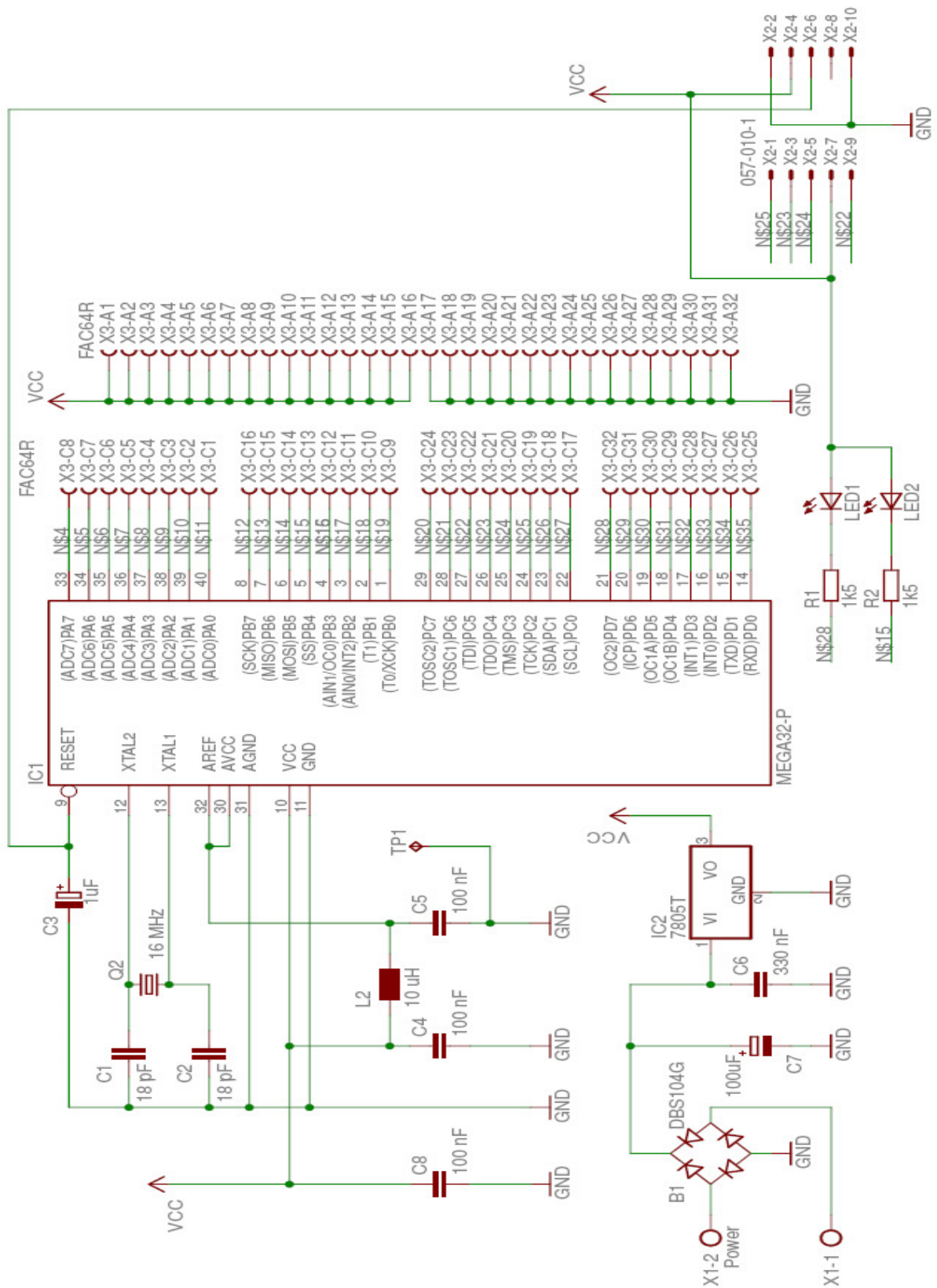


Figure 4, The kit

Q2, C1 and C2 are external components used by the oscillator circuit to generate a 16 MHz clock.

C3 is part of an RC charge circuit. It will ensure a stable reset period at power-on.

R1, R2, LED1 and LED2 are two low-current active low LED's that can be controlled by the μ P at port-pins PD7 and PB4.

The small connector is used for a 10 lead ribbon cable to a PC for JTAG interface.

The big connector is used for general interface to extension boards. Portpin PA0 is near pin 40 of the μ P – connector pin C1. Connector row A is near the PCB edge.

In order to use the kit, the following connections must / can be made:

- Connect approximately 10 V DC to the power connector. Polarity is irrelevant.
- Connect the USB-to-JTAG interface to the 10 pin connector.
- Connect your extension board to the big connector. Always observe your current consumption at power up. Is it realistic? Expected current consumption:
 - Kit only: 20 mA
 - Kit and JTAG interface: 50 mA
- Download and run your code from the AVR studio program on your PC.

3 The IDE – Integrated Development Environment

The software development package from Atmel consists of two programs:

- AVR studio
- GNU C compiler

The GNU compiler is an open source project mainly targeted towards Linux-oriented command-prompt-loving programmers. It is definitely not user-friendly and it takes a long time to learn mastering it. Atmel expanded their programming interface – the AVR studio – to include a windows-based graphical user-interface – a true IDE as a front-end to the GNU compiler. The result is that the project manager, the editor, the compiler, the downloader and the debugger are all included in the same very intuitive and user-friendly interface. The entire package is available for free download from Atmel.

Installation


You must install two different programs: The AVR studio and the compiler. It must be done in this order: First install AVR studio, then the compiler.

- The AVR studio: Get latest version here: http://atmel.com/dyn/products/tools_card.asp?tool_id=2725. This document describes version AVR Studio 4.14 (build 589). Later versions might have an updated user interface. Use default values for everything during installation.
- The compiler: Get latest version here: <http://sourceforge.net/projects/winavr>. This document describes version 20071221. Later versions might have an updated user interface.
- Using Vista or Windows 7? You might get USB problems. Get the latest USB driver for the JTAG ICE here: <http://www.ftdichip.com/Drivers/VCP.htm>, select and run: [setup executable](#).

Creating your first simple project – or a guided tour through the IDE

1. Make a folder, e.g. C:\AVR_Projects for your programs.
2. Start AVR studio. Close any pop-up windows.
3. Select Tools / Options / Editor. Mark “Replace tabs with space” and set Tabwidth = 3.
4. Select Project / New Project.
 - Choose AVR GCC – the alternative is used for assembly language projects.
 - Write a project name. Only English letters, no spaces.
 - Mark “Create initial file” and “Create folder”.
 - Choose a location for the project – the folder you made in step 1.
 - Press Next.
 - Select “Debug platform = JTAG ICE” and “Device = ATmega32”.
 - Press “Finish”.
5. Select: Project / Configuration options / General. Change “Optimization” to “-O0”. Press “OK”. This means that the compiler will not optimize your code.
6. Step #4 and #5 must be done every time you start a new project.
7. An editor window opens with the same name as your project. Type the following program:

```
#include <avr/io.h>
int main (void)
{
    DDRD = 0xFF;           // Set all pins of port D as output
    while (1)
    {
        PORTD = 0x00;      // LED on
        PORTD = 0xFF;      // LED off
    }
}
```





8. Compile the project by pressing the compile button  or press F7.
9. Check the Build window at the bottom of the screen. Any red dots? They indicate detected errors in your program. Fix the errors and try again.
10. Select: File / Save All.
11. Verify that your project is saved in your project folder. The *.APS file is the project file.

Download and debug

In order to download and execute your program, the following steps must be followed:

1. Power off your µP kit.
2. Connect the JTAG interface between the kit and the PC with the USB cable.
3. Apply power (+10-12 V) to your kit. Monitor the current consumption while turning the power on. Switch off immediately if the power is too high (> 100 mA).
4. Do the following once for each kit:
 - Select Tools / Options / Number of Comm ports = 16. If you have problems connecting, you should force your PC to use COM port 1.
 - Select Tools / Program AVR / Connect.
 - Select JTAG ICE + Auto.
 - In the pop-up window: Select Fuses / CKSEL / last option. Click “Program”. Close window. Problem here? Most likely your USB-driver. Install the update.

5. AVR-studio will either be in edit-mode or debug-mode. Use edit mode to edit and compile your code and use the debug mode to run and test your programs.

6. Press “Build and Run”  or Compile  and Start Debugging . Your program has now been compiled and downloaded into your kit. The yellow arrow  shows you the present position of the Program Counter. You are now ready to start using the different debug commands:



Stop debugging: Stop the debugging and return to the editor for changing the code.



Run (F5): Run the program at full speed (16 MHz).



Break (Ctrl + F5): Stop the program but remain in the debugger.



Reset (Shift + F5): Stop the program and jump to line 1 in the main function.



Step over (F10): Execute one C-instruction. Stop at the next instruction.



Step into (F11): Execute one C-instruction. Step through all the sub-instructions, e.g. each count in a for-loop.



Auto step (Alt + F5): The debugger will step through the program at low speed. You can follow the program execution on the screen.





Run to cursor (Ctrl + F10): Place the cursor somewhere in the code while the debugger is in “break”-mode. Use “Run to Cursor” and the program will run at full speed until it meets the code-line where you placed the cursor. Then the debugger will break.

7. Use “Step into” (F11) a number of times. Watch the screen and the kit. This is called single-stepping through the program.
8. Use “AutoStep” (Alt + F5). Watch the screen and the kit.
9. Press “Break” and “Run”. The μ P is now running at full speed (16 MHz). Notice the dimmed light from the LED.
10. Press “Break” and “Stop Debugging”. Expand the code:

```
#include <avr/io.h>
int main(void)
{
    int i = 0;
    DDRD = 0xFF;           // Set all pins of port D as output
    while (1)
    {
        i++;
        PORTD = 0x00;      // LED on
        PORTD = 0xFF;      // LED off
    }
}
```

11. Press “Build and Run” and enter the Debug Mode.

12. If you don't see the watch window, turn it on by using  "Toggle Watch Window".

Highlight the variable "i" and use  "Quickwatch". Click "Add to Watch", and you will see the variable in the Watch window. Select "AutoStep" and notice the action in the watch window.

4 C coding: Controlling individual bits on the ports

The individual bits on the ports of the μ P can not be controlled directly. Whenever you want to change the value of a port-bit, you must write a full byte to the port. If you only want to change the value of one bit on a port, you must make sure that the seven other bits are left unchanged. An example: We want to pull bit 2 on port A low.

Port A:	x	x	x	x	x	x	x	x
Control value:	1	1	1	1	1	0	1	1
	<u>AND</u>							
Result:	x	x	x	x	x	0	x	x

The code: `PORTA = PORTA & 0xFB;`

When you AND something with the value 0, it will go low. If you AND something with the value 1, it will remain unchanged. Next example: Pull bit 5 on Port B high.

Port B:	x	x	x	x	x	x	x	x
Control value:	0	0	1	0	0	0	0	0
	<u>OR</u>							
Result:	x	x	1	x	x	x	x	x

The code: `PORTB = PORTB | 0x20; or PORTB |= 0x20;`

When you OR something with the value 1, it will go high. If you OR something with the value 0, it will remain unchanged.

You can invert a bit by using an exclusive-OR operator. Let us invert bit 5 on port D:

Port D:	x	x		x	x	x	x	x
High to invert:	0	0	1	0	0	0	0	0
	<u>XOR</u>							
Result:	x	x	!x	x	x	x	x	x

Code: `PORTD = PORTD ^ 0x20;`

If you want to write a group of bits to a port, the same method is applied. An example: Write the 4-bit value 1011 (placed as bit 3..0 in variable 'x') to bit 5..2 on port A.

Port A:	x	x	x	x	x	x	x	x
Clear the 4 bits:	1	1	0	0	0	0	1	1
	<u>AND</u>							
Result (A2):	x	x	0	0	0	0	x	x

Shift 'x' two places left: `00001011 → 00101100 (x2)`

A2:	x	x	0	0	0	0	x	x
x2:	0	0	1	0	1	1	0	0
New Port A:	x	x	1	0	1	1	x	x

C code for the last example: `PORTA = (PORTA & 0xC3) | (x << 2);`

5 Programming State Machines

A computer program can either be sequential or event driven. Sequential programs are those that perform a number of tasks without being controlled by any inputs. An event driven program is controlled by inputs from the outside world, either from a user or from electronic devices connected to input pins.

A state machine is a programming method where the program can remain in one of a number of states. The program remains in a specific state while or until a condition is met, after which it moves on to another state. These conditions are typically external inputs to the μ P (keyboards, sensors, analog values, etc) or internally generated events (communication interrupts, timers, etc).

The state machines are documented by using a state diagram – see details in your compendium regarding sequential circuits. The code is written using a switch statement to determine the active state. The switch will have one case for each relevant state. The state machine can almost be designed as a small operating system. In that case – if it controls the overall program-flow – it will reside in the main function. Otherwise it will be placed in a relevant function. It could look like this:

```
char input, state = 0;           // initial state

while (1)
{
    input = GetInput();          // read input for the statemachine
    switch (state)
    {
        case 0:
            if (input == 'A')
                state = 3;
            if (input > 'D')
                state = 5;
            break;
        case 1:
            // ... etc
    }
}
```

6 Embedded μ P exercises

6.1 *Embedded Exercise: LED 1*

- This is a pure software exercise that will run on the μ P kit.
- Set the two LED port pins as outputs.
- Let one LED flash three times as often as the other.
- Make a reasonable delay between each change of state by means of a for-loop.

6.2 *Embedded Exercise: LED 2*

- This is a pure software exercise that will run on the μ P kit.
- Set the two LED port pins as outputs.
- Let one LED flash three times as often as the other.
- The delay between each change of state is made by calling a function: `void delay (unsigned integer ms).`
- The delay is made in the function by a calibrated for-loop.

6.3 *Embedded Exercise: I/O*

- Target: Make a counter in a 7-segment display
- Design: Mount a 7-segment common-anode display and a connector on a PCB. Remember current limiting resistors. Connect the 7 segments and the decimal point to a port on the processor.
- Draw a detailed schematic.
- Make a truth table showing which segments to turn on for each number 0..9.
- Write a program which repeatedly counts from 0..9 using realistic delays.
- Test the program and document the work (hardware and software) in a report.

6.4 *Embedded Exercise: Encoder*

- Target: Add a 2-bit rotary encoder to the hardware from the last exercise. Design and code a state-machine. The input from the encoder will control a binary counter made with the 7-segment display.
- Mount the encoder. Connect it to two input pins.
- Update your schematic from the last exercise.
- Measure and draw a pulse-plan for the behaviour of the encoder.
- Draw the state-diagram showing the functionality of your program.
- Write the software:
 - Initialize the I/O
 - Make a 5 ms delay between each call to the state-machine.
 - Make a state-machine based on a switch command.
- The counting frequency of the display will increment at clock-wise rotation and decrement at counter-clock-wise rotation.
- Test the program and document the work in a report.

7 Literature

A few suggestions:

- The reference book concerning the C language is Kernighan & Ritchie: “The C Programming Language”, ISBN 0-13-110362-8. This book is highly recommended.
- AVR libraries: Fronter → [avr-libc-user-manual.pdf](#)
- AVR mega32 databook: Fronter → [mega32A databook.pdf](#)