Implementation of a Microcode-controlled State Machine and Simulator in AVR Microcontrollers (MICoSS)

S. Korbel, V. Jáneš

This paper describes the design of a microcode-controlled state machine and its software implementation in Atmel AVR microcontrollers. In particular, ATmega103 and ATmega128 microcontrollers are used. This design is closely related to the software implementation of a simulator in AVR microcontrollers. This simulator communicates with the designed state machine and presents a complete design environment for microcode development and debugging. These two devices can be interconnected by a flat cable and linked to a computer through a serial or USB interface.

Both devices share the control software that allows us to create and edit microprograms and to control the whole state machine. It is possible to start, cancel or step through the execution of the microprograms. The operator can also observe the current state of the state machine. The second part of the control software enables the operator to create and compile simulating programs. The control software communicates with both devices using commands. All the results of this communication are well arranged in dialog boxes and windows.

Keywords: state machine, microcontroller, microprogramming, software implementation of a simulator.

1 Introduction to Atmel microcontrollers

The structure of AVR microcontrollers was designed so as to comply with high-level language compilers, namely the widely used C language. Such an optimized core unit having the Harvard architecture bears the main characteristics of microprocessors with a reduced instruction set (RISC). The basic architecture of the AVR microcontrollers is depicted in Fig. 1.

The whole family of AVR microcontrollers can be divided into 3 subgroups: the AT90S, ATtiny and ATmega families. The AT90S series was developed first, and its name suggests that it represents a continuation of the AT89C series. The situation with the other two series is different, since the manufacturer had more experience with the previous series

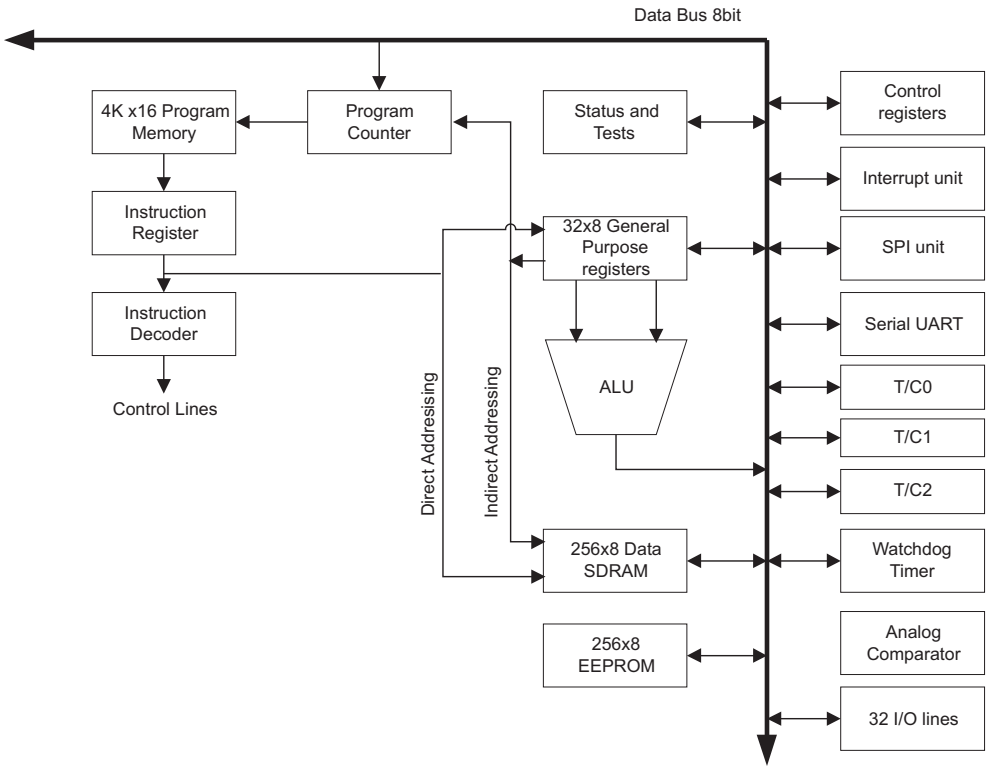

Fig. 1: Architecture of AVR microcontrollers

and designed the names of these two according to their designation.

Representatives of the ATtiny family are designed for smaller, simpler applications, while the ATmega family representatives are designed for complex and more sophisticated applications.

More detailed information concerning the architecture of AVR microcontrollers can be found in [4] and [6].

2 Applications of microcontrollers in the design environment

In the 1970's computer controllers were designed as microcode-controlled state machines. Although we have now switched when designing microcomputers to standard circuit control units, state machines are still important. They are suitable for certain control applications and tutoring. When writing a microprogram we can test the controlling procedures of some of the operations and subsequently use the acquired knowledge in optimizing the microprogram. The correct function of the microprogram and the subsequent optimization is strongly dependent on the development and tuning devices. Thus there is great emphasis on the design environment in which the microprograms are written.

It is appropriate to use a microcode-controlled state machine structure for developing microprograms (see Fig. 2). The whole structure of a state machine has been implemented into AVR microcontrollers by Atmel Company [3]. The simulation of the behavior of the designed microprogram is carried out on the model of a control system, which is also implemented in the AVR microcontroller. This model is so universal that it can be used for simulation of an arbitrary system.

The design environment **MiCoSS** is composed of two parts: **Mi**crocode-**Co**ntrolled **S**tate machine [1] and **S**imulator of the controlled unit [2]. These devices can be interconnected by a flat cable and linked to a computer through a serial RS-232 or USB interface.

The application software is common to both devices. It allows us to crate and to edit microprograms and to control the whole state machine, i.e., to start, cancel or step the running of the microprograms and to monitor the current state of the state machine. The second part of the simulator software enables an operator to create and compile simulating programs

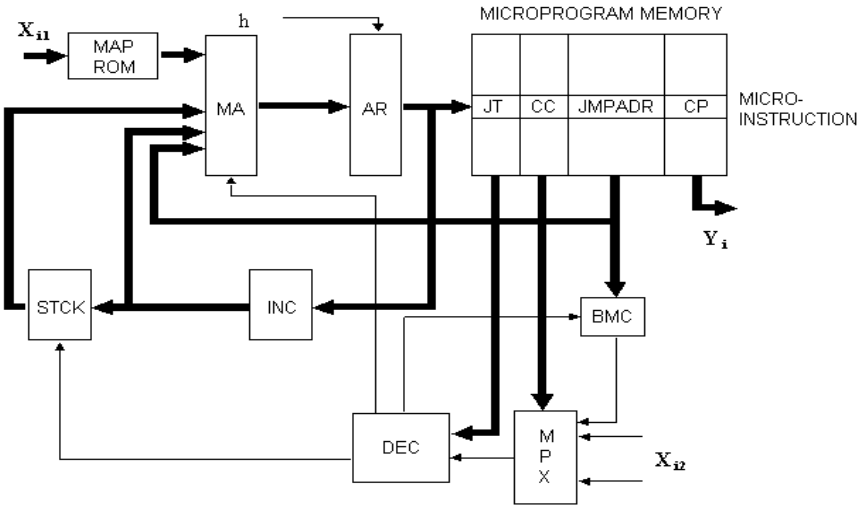

Fig. 2: Structure of a microcode-controlled state machine

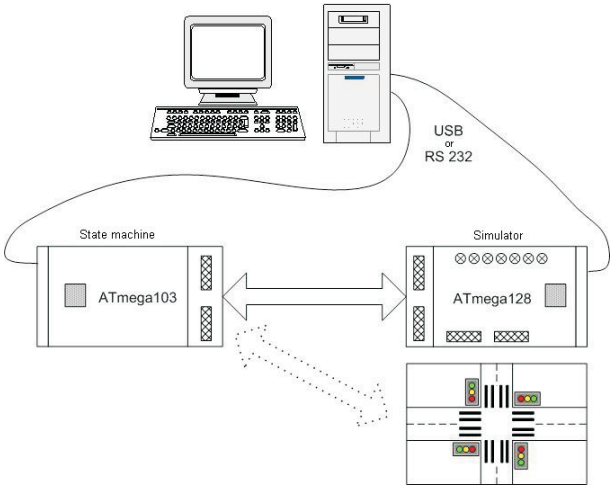

Fig: 3: Design environment scheme

and to control the simulator. A scheme of the design environment is given in Fig. 3.

3 Microcode-controlled state machine

The realized microcode-controlled state machine was developed according to the scheme in Fig. 1, so that its behavior and function would resemble as much as possible the state machine of an AMD2909 chip. For the implementation of the whole core of the state machine, the AVR microcontroller was chosen and designated as ATmega103 [4].

This microcontroller has input/output interfaces by means of which it is possible to communicate with outside parts. A great advantage is the large number of these interfaces, so that they can be used as direct inputs and outputs without any additional multiplexing. Interfaces A, B and C are outputs and interfaces D, E and F are inputs. The lowest two bits from interface E and the lowest two bits from interface F are reserved for communication.

This implies that the micro operational command is 24 bits wide, without the necessary output multiplexing. There are 16 bits connected directly into MPX input and 4 bits into MAPROM memory. The total input is thus 20 bits. This is then reflected in the number of 40 pin connectors on the printed circuit board (20 pins are grounded).

The address of microinstruction was chosen to be 15 bits and owing to this, it is possible to develop microprograms up to maximum size 32k (32769 microinstructions).

The above considerations imply that it is possible to determine the size of the individual parts of the microinstruction, whose structure is depicted in Fig. 1. Because 16-bit input was chosen, the condition code (CC) is 5 bits sized. The reason for 5 bits and not 4 bits is as follows: because the input as well as the output from the backward micro cycles counter (BMC) is input into multiplexer MPX, and because constants 1 and 0 are also input here, the total numbers of input bits to input multiplexer MPX are thus 19. From this value it follows that to choose just one input bit from the whole set of 19, the above mentioned 4 bits are not sufficient. The width of the jump type TS was taken from AMD Company, to be precise from chip 29811, i.e. 4 bits. In keeping with the width of the address of microinstruction, the address of the jump ADRSK is 15 bits.

The resultant structure of the microinstruction, including the proposed widths of the individual items, is given in Fig. 4.

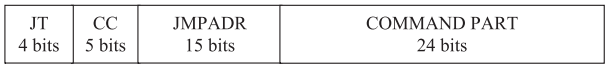

Fig. 4: Resultant microinstruction structure

What remains to be determined is the width of the backward counter of the micro cycles ZPC, the depth of the stack ZAS and the size of the mapping memory MAPROM. The backward counter ZPC is set according to the address of the jump ADRSK from the microinstruction, thus its width will also be 15 bits.

The stack memory (STM) at slices AMD was 4 items deep. If the microinstruction address space were extended up to 32k, the depth of 4 items space would not be sufficient. Therefore the depth of the stack memory was enlarged up to 16 items. Another reason for this enlargement is the fact that, with such a large space, it is possible to create more complex microprograms; consequently the demand for stack memory rises as well.

The width of mapping memory MAPROM is the same as the address of microinstruction, i.e., 15 bits. In order to address this memory 4 input bits will be used; therefore 16 items result from it.

The proposed state machine properties are summarized in the following table.

Table 1: Properties of state machine

Characteristic	Size
CC	5 bits
JT	4 bits
JMPADR	15 bits
COMMAND PART	24 bits
Backward micro cycles counter	15 bits
Input multiplexer	19 bits
Stack size	16 items
MAPRAM size	16 items

3.1 Implementation of a microcode-controlled state machine

The whole design of a microcode-controlled state machine was developed in the AVR Studio version 3.53 [5]. The AVR Studio is a professional development tool for development and debugging of C applications that will be running in AVR microcontrollers.

The implementation of the microcode-controlled state machine itself is depicted in Fig. 1. The individual blocks of the scheme are described as variables and independent functions that perform the required task. All these functions are periodically called in an infinite loop until the running is interrupted. If there is no data on the data link to PC, the application runs autonomously according to the input conditions, and it generates outputs signals. If this is not the case (individual inquiries are being sent from PC), the application performs the requested tasks.

A more detailed description of the implementation and specification of the individual blocks according to Fig. 2 is given in [1].

4 Controlled system simulator

As stated above, a part of the design environment is a simulator of a controlled system. The simulator is a completely universal structure, which enables us to simulate an arbitrary system. To achieve this, there are also external or data inputs and outputs available besides control inputs and outputs. Control and data inputs are chosen by means of multiplexers; conversely, the control and data outputs use latch registers. A total of 16 data inputs and outputs are implemented.

Table 2: Example of simulator commands

Number	Command	Type	Description
3	MAPROM setting	Systemic	Sets 17–20 simulator output bits.
20	Set 1–8 according to input	Command, powerful	Sets output bits 1–8 in accordance with input bits 1-8
63	Set B according to Out (1–8)	Command, powerful	Sets B register according to output bits (1–8)
74	Give back output state (9–16)	Information	Simulator sends outputs bits 9–16 state
83	Wait for bits inverting	Blocking	Simulator waits on chosen input bit settings to inverse state.
166	Give back registers state	Command, powerful	Simulator sends to PC actual content of D, E and F register.

The data inputs can be used to simulate the inputs of variables or to connect an external matrix keyboard. The data outputs can be used for example for connecting of the seven segment LED display. The data inputs and outputs can be used either together or separately by means of a jumper. The general architecture of the device is depicted in Fig. 5.

For the implementation of the simulator, the AVR microcontroller ATmega128 [6] was chosen. This microcontroller is placed on a printed circuit board together with the communication interfaces, data inputs and outputs, display elements and mode jumpers.

The ATmega128 microcontroller also has enough interfaces; moreover it has one extra communication interface G. This is, in the design itself, used to control the input multiplexer and output latches. Two bits from this interface are used for indication LED diodes, which inform about the running and the state of the simulator. The remaining interfaces are used for communication between the simulator and the state machine in the following way; Interfaces A, B and C are used for inputs to the simulator, and interfaces D, E and F are used for outputs. The two lowest bits from interface F are reserved for communication with the computer.

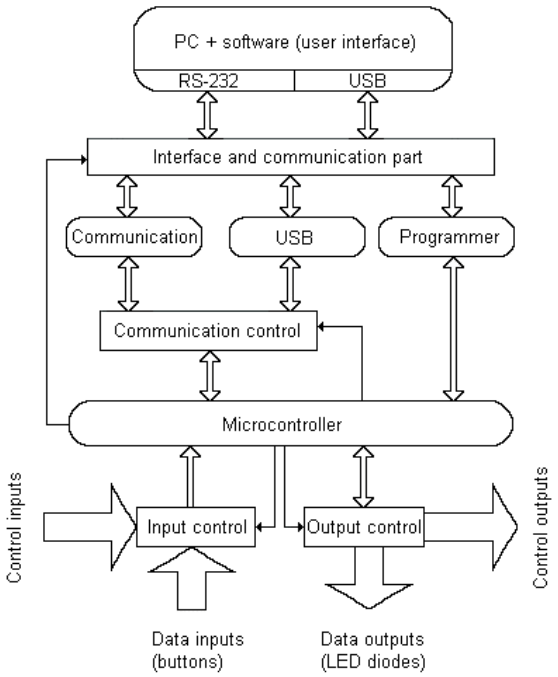

Fig. 5: Architecture of the simulator

A more detailed description of the simulator can be found in [2].

4.1 Implementation of the simulator of controlled system

The whole design was, as in the previous case, performed in the design environment of AVR Studio version 3.53 [5].

The implementation itself was carried out in the following way; the basic communication between an application running in a microcontroller and a computer is accomplished by sending individual commands. Each command has its specific code which determines its meaning. For this reason, each command is assigned one function in the application. These functions are periodically called in an infinite loop according to the corresponding received code. The list of selected codes is given in the Table 2.

A detailed command description is given in [2].

5 Communication software

The proposed software is common to both the microcode-controlled state machine [1] and the controlled system simulator [2]. The software application was written in the program Delphi 6.0 to ensure transferability of the code between the individual platforms. The program may be executed under Windows 95, 98, 2000 and XP.

The design of software for the state machine is maximally accommodated to software for the simulator; it is composed of two-level program units. The units of lower level ensure communication with the device, and high-level units ensure communication with the user.

5.1 Microcode-controlled state machine

Proposed architecture of communication software is depicted in Fig. 6.

5.1.1 Communication with the device (lower level)

Communication with the device is accomplished by a serial UART interface on the part of the microcontroller and a serial RS-232 or USB interface on the part of the computer.

The whole communication is accomplished by sending individual commands to an application in a microcontroller, which performs the required function.

There are two different modes. The first is autonomous running of the state machine. In this mode the running depends only on the inputs (state information), and the outputs (controlling information) correspond to the stored microprogram. In this way the state machine can be used, for example, to control a processing line. The state machine can work quite autonomously and is independent of further commands from the computer.

Selection of the second mode enables the user to influence directly the running of the whole machine. Of course, it is possible to influence the running of the machine even in the previous mode; the state machine can be switched between these two modes.

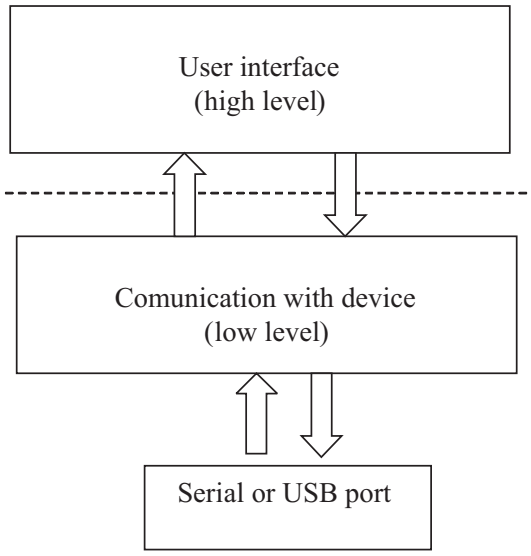

Fig. 6: Software architecture

This choice is accomplished by changing logical ‚0' to logical ‚1' on the line RTS of RS-232. It can be said that this line serves as a switch for the functioning of the whole machine and it switches between the two modes.

If logical "1"is chosen in the RTS line, the serial interface UART of the AVR microcontroller plays an important role. Through UART, individual controlling bytes are sent to a microcontroller. Each transferred byte represents either a command (selected command) or transferred data. After logical "1" has been set on the RTS line, a further, important choice follows. The microcontroller awaits which byte will be accepted by UART. A transferred byte (0x10) specifies the choice of microprogram memory – work with its contents, while byte (0x20) signifies the choice of MAPROM memory – also handling its contents; byte (0x30) signifies monitoring the state of the machine, changes of contents in individual registers, setting breakpoint and stepping mode. After this choice, mere transfer of data to the microcontroller follows.

Communication between a computer and a microcontroller on a low level is solved by the above described procedure. To illustrate the mutual communication, see Fig. 7, which deals with the contents of the memory of the microprograms.

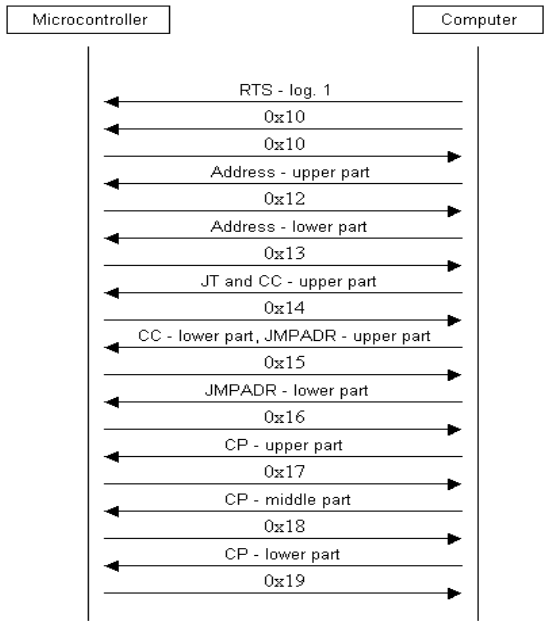

Fig. 7: Low-level communication between devices

5.1.2 User interface (higher level)

The user interface enables users to use the following functions:

- Testing of the connected device
- Editing of MAPROM memory and microprogram memory
- Saving and reopening of both memories
- Monitoring the state of the machine:
  - Microprogram stepping, register state watching, microinstruction address (RA) stack top and backward counter DC.
  - Microprogram running with possible stopping and breakpoint address definition.
  - Microinstruction address (RA) top of STCK and backward counter DC value changing.

The whole application can be operated using a keyboard and a mouse (like any application running in Windows), and there are also shortcuts for majority operations.

After starting application atas.exe (main file application), the basic application window is displayed (see Fig. 8). There is a main menu in the upper part. The menu contains the following items: Project, Edit, Mode, Run, Window and Help. Below the menu there are many buttons for quick choice: Create new project, Open existing project, Save current project, buttons for text editing (Cut, Copy, Paste), state machine and Simulator. A detailed description of menu and buttons can be found in [2].

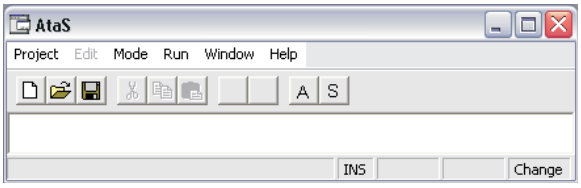

Fig. 8: Basic application window

After pushing button A in the main window or after clicking on the item State machine in menu Mode, the new, so-called state machine window is opened (Fig. 8). This window is absolutely independent of the main window. In the upper part there is the main menu with the following items: File, Microprogram, Run, Device and Help. The specification of other buttons listed in the main menu can be found in [1].

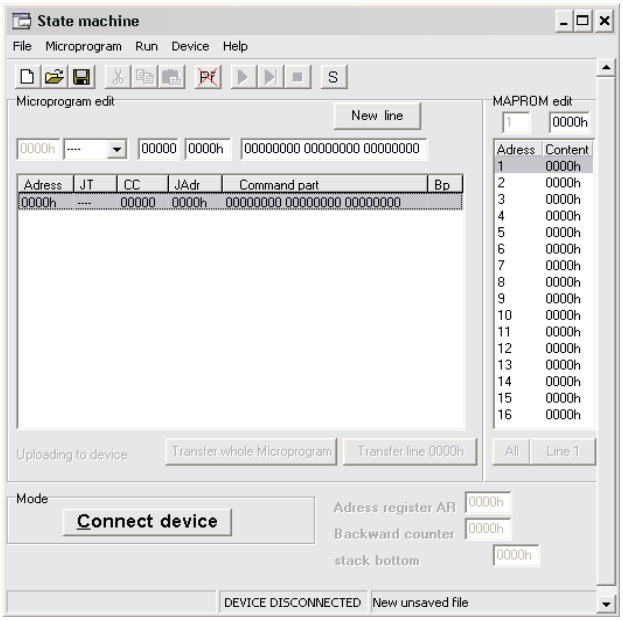

Fig. 9: State machine – disconnected device

As indicated in Fig. 9, within this window it is possible to input the individual microinstructions and to edit the content of MAPROM. All this can be done without the necessity to link the device. At the moment when you need to test the microprogram, you must take the following steps:

- Connect the device to the computer
- Choose the communication interface
- Choose the communicating speed.

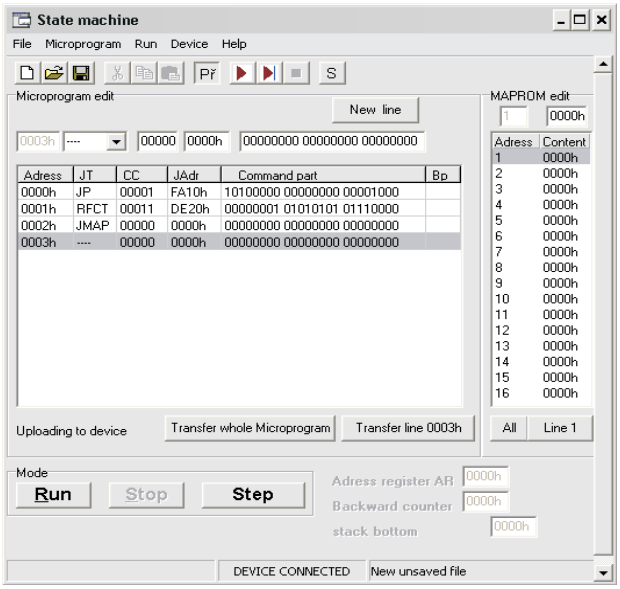

Fig. 10: State machine window – connected device

Then you must press the "Connect device". At this moment, the application tests the device. If everything is correct, three other buttons controlling the operation of the device appear. These are the "Run", "Step" and "Stop" buttons. Fig. 10 describes the situation after the device is connected. If the connection fails, an error message is displayed.

A detailed description of communication with the state machine can be found in [1].

5.2 Simulator of a controlled system

The simulator window contains a text editor for recording simulating programs (see Fig. 11). These programs are written in a special simulating language: "S-language".

In terms of control, the structure of the simulator is simple, and it copies its hardware structure. All simulating language commands, are related to specific basic elements (control and data registers, stack, and so on). Commands are divided into system, command, blocking, and information. A detailed description is given in [2].

Simulating programs can be saved and reloaded by commands from the File menu, or directly by pressing corresponding icons from control panel. The saved programs have a *sim* extension. Simulating programs are saved in text mode in this file. Therefore they can be written simply in any text editor, saved with a *sim* extension and then opened in a simulator program window. An important choice is represented by the "Compile" button. This button begins the compilation into internal form. During compilation, the correctness of the program from the semantic and syntactic point of view is checked. The syntactic analyzer repeatedly calls the lexical analyzer, which returns the appropriate lexical elements. If there are lexical or syntactic errors, the compilation is stopped and the error must be corrected.

It is necessary to connect the simulator device to start the simulating program. When the "Connect device" button is pressed, the program starts communication with the device. After confirmation of the connection by the device, the items for starting the simulation programs are made accessible.

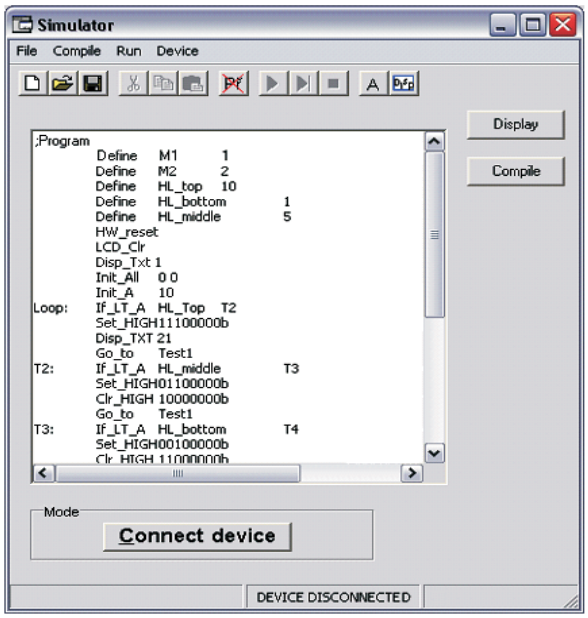

Fig. 11: Window of simulator

It is possible to choose between continuous running or stepping of the simulation program. The current program line is highlighted on the screen. A special mode is stepping with delay. This choice is suitable if it is necessary to debug the application and to follow the processes running in the simulator. The simulator or stepping program can be started from both the menu and icons (tool panel). The "Stop" button stops simulation. After starting or stepping simulation, a new Display window appears (see Fig. 12).

The display window substitutes the simulator display and other signaling elements. It clearly displays the stack register contents and the data input and output logical values. The command code or each single command can be sent directly to simulator, irrespective of the executed program.

A detailed description of communication with the simulator can be found in [2].

5.3 Intercommunication of devices

As indicated above, the microcode-controlled state machine device is connected to a controlled system simulator device by flat cables. The input signal cable has 40 pins and the output signal cable has 50 pins – micro operation sign MOZ (command part-CP). The reason for using this cable is that for each of signal wires it is also necessary to have a ground wire. The signal input is 20 bits, 20 bits are the ground wires – a 40 wire flat cable.

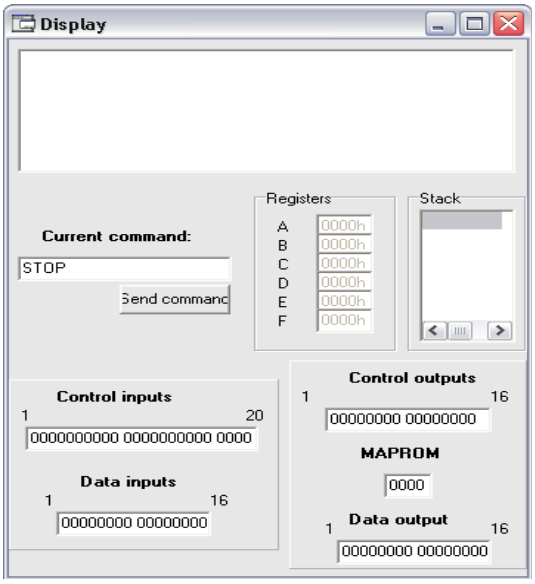

Fig. 12: Display window

Mutual synchronization of the two devices has not been explicitly solved. It is thus necessary to implement this synchronization directly in the running application, which is being tested on the devices. A possible solution is to reserve one signal bit in both directions and control the running of the whole state machine according to the state of this signal wire. The second choice for mutual synchronization of the devices, which is made possible due to common connection of both devices to a single computer, is to control the running of the state machine on the basis of the input of the control byte from the PC. This would define the exact moment at which the state machine would start operation. In addition to positive considerations accompanying this synchronization, there would also be some negative considerations. An example is the delay resulting from waiting for the input of a starting byte.

Synchronization could also be achieved by creating a clock – a cycle on the device of a microprogrammable state machine. "Clocks" generated in this way would enable us to control the running of the whole state machine, and at same time the "clocks" could be joined to the top of the output connector. This would ensure control of the simulator from a single time base. Such a generator would use for instance a counter with a preset initial value. In the event of downward counting, an interruption would occur and thus a clock pulse would be generated. It is obvious that the value set in the counter would have to correspond to the longest executed part of the program. However, if speed is emphasized, this method would cause a delay in the running of machine.

6 Example

Let us assume we have a reservoir with two pumps and an outflow in our house. The reservoir supplies the whole house with drinking water. There are three sensors in the reservoir. If the water level is between sensors H1 and H2, both pumps are inactive. If the level sinks below sensor H1, pump no. 1 begins to fill the reservoir. When the level is above sensor H2, pump no.1 is stopped. Conversely, if the level sinks below the sensor H0, the pump no. 2 also begins to fill the reservoir. If the level is above sensor H2, both pumps are stopped.

Buttons no. 1 and no. 2 simulate the requirements on the amount of water. The whole reservoir with pumps and buttons is depicted in Fig. 13.

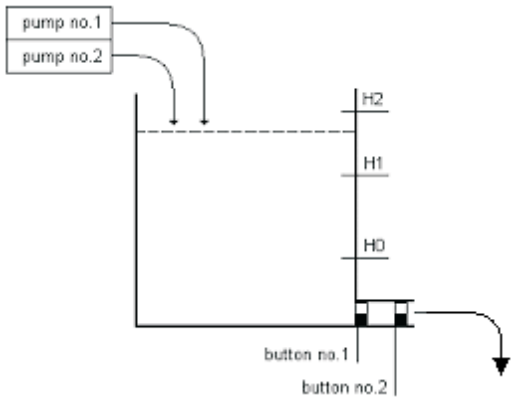

Fig. 13: A reservoir with pumps and sensors

The microprogram and a model of the controlled system are shown in Fig.14 and Fig.15. The simulator program with a detailed description is shown in Table 3. Information on the

Address	JT	CC	JmpAdr	CP
000	CJP	4	000	0x000000
001	CJP	5	000	0x000001
002	CJP	5	001	0x000001
003	CJP	5	000	0x000003
004	JP	0	003	0x000003

Fig. 14: Microprogram of the reservoir

Table 3: The simulator program of the water reservoir

```
;Program "The water reservoir"
            Define      M1              1       ;the pump no.1
            Define      M2              2       ;the pump no.2
            Define      HL_top          10      ;the sensor H2
            Define      HL_bottom       1       ;the sensor H0
            Define      HL_middle       5       ;the sensor H1
            HW_reset
            LCD_Clr
            Disp_Txt 1  'The simulation of the reservoir'
            Init_All    0 0                     ;initialization
            Init_A      10                      ;the level of water
Loop:       If_LT_A     HL_Top          T2      ;if the level of water is below the sensor H2 then jump to T2
            Set_HIGH    11100000b               ;set all sensors to 1
            Disp_TXT 21 'The reservoir is full !'
            Go_to       Test1
T2:         If_LT_A     HL_middle       T3      ;if the level of water is below the sensor H1 then jump to T3
            Set_HIGH    01100000b               ;the level of water is between sensors H1 and H2
            Clr_HIGH    10000000b               ;set the sensor H2 to 0
            Go_to       Test1
T3:         If_LT_A     HL_bottom       T4      ;if the level of water is below the sensor H0 then jump to T4
            Set_HIGH    00100000b               ;the level of water is between sensors H0 and H1
            Clr_HIGH    11000000b               ;set the sensors H1 and H2 to 0
            Go_to       Test1
T4:         Clr_HIGH    0ffH                    ;the level of water level is below the sensor H0
            Init_A      0
            Disp_TXT 21 'The reservoir is empty !'
Test1:      If_CLR_IN   M1              Test2   ;if the pump no.1= 0 then jump
            Inc A                               ;the level of water level is raised
Test2:      If_CLR_IN   M2              TestTL  ;if the pump no.2 = 0  then jump
            Inc A                               ;the level of water level is raised
TestTL:     If_CLR_TL1                  Tlac2   ;the test of pressing button no.1
            Dec_A                               ;the level of water level is reduced
Tlac2:      if_CLR_TL2                  Loop    ;the test of pressing button no.2
            Dec_A                               ;the button no.2 is pressed
            Dec_A                               ;fast draining
            Go_to                       Loop
            END
```

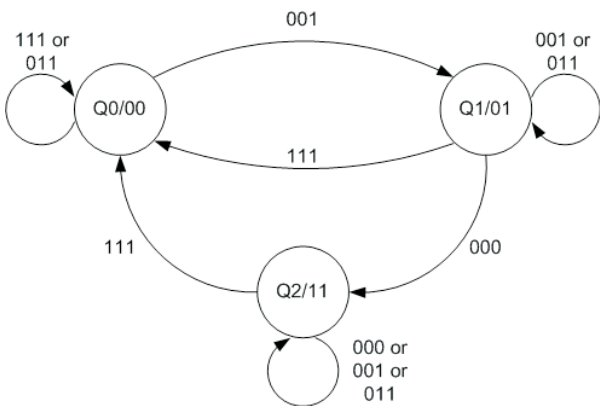

Fig. 15: The state machine model

amount of water is stored in register A and it is shown on the PC monitor.

7 Conclusions

A complete design environment for microcode development and debugging is a demonstration of systems implementation in AVR microcontrollers by Atmel Company. This design environment is suitable both for the construction of simulator of controlled object and for the design of the microprogram of a state machine. The final behavior of the designed state machine can be tested.

All the components are freely available on the market and thus this environment can be mass-produced. The universality of the environment offers the possibility of further experimentation in programming and the design of various structures with AVR microcontrollers.

References

[1] Korbel, S.: *Design and Implementation of a Microcode-Controlled State Machine*. Diploma thesis, FEE CTU Prague 2003 (in Czech).

[2] Klíma, D.: *Controlled System Simulator*. Diploma thesis, FEE CTU Prague 2003 (in Czech).

[3] Atmel: http://www.atmel.com/

[4] ATmega103: http://www.atmel.com/dyn/products/devices.asp?family_id=607#760

[5] AVR Studio: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

[6] ATmega128: http://www.atmel.com/dyn/products/product_card.asp?part_id=2018

Ing. Stanislav Korbel
e-mail: korbels@fel.cvut.cz

Doc. Ing. Vlastimil Jáneš, CSc.
e-mail: janes@fel.cvut.cz

Department of Computer Science and Engineering

Czech Technical University
Karlovo nám. 13,
121 35 Prague 2, Czech Republic

