FUNCTIONAL SIMULATION IN TEACHING MICROPROCESSORS ARCHITECTURE*

Jacek Majewski** and Włodzimierz Barański**

The paper considers the problem of simulation the microprocessor hardware functions in the IBM PC/AT computer graphics. The simulation has been based on the assumption that the electric hardware details of microprocessors are not important from functional point of view. Control programs, written by the users, for simulated environment and for real hardware must be the same. Of course is very important that simulated environment is invisible for the students. The paper considers as an example the process of preparation a control program for the model of plotter. The control plotter programs are written in 8086 assembler and compiled by real compiler: Borland TASM.

1. Introduction

From the beginning, when microcomputers were invented the most popular idea of microprocessor's communication with outside world is through input/output ports. This idea is a base of constructing more advanced and more complicated microcomputer peripherals as parallel/serial ports, timers/counters, DMA controllers and so on. Generally, internal ports of these peripherals are designated for the following functions:

- DATA ports
- CONTROL ports
- STATUS ports

Idea of communication through input/output ports is also used for such complicated devices as printers/plotters, floppy/hard discs, etc.

More complicated devices like display graphic controller have inside not only ports but also memories or ports mapped into memory.

Complicated external devices have complicated electric schemes. From programmers point of view electric details of controlled devices are not important. For proper control is enough to know what to send to CONTROL/DATA ports and what to watch on STATUS/INPUT ports. Situation is the same even externals are causing interrupts.

At student teaching of microprocessors above idea is very common: students are controlling external simple real devices, connected to the microcomputers (Fig. 1), by writing programs in specified language (assembler, C , Basic). These programs of

* *Invited paper not presented at the Workshop*

** Institute of Engineering Cybernetics, Technical University of Wroclaw, ul. Janiszewskiego 11/17, 50-370 Wroclaw, Poland

course are sending sequence of controls through input/output ports. Instead of real devices let's consider situation of control simulated devices on computer screen (Fig. 2). This aspect is presented below.

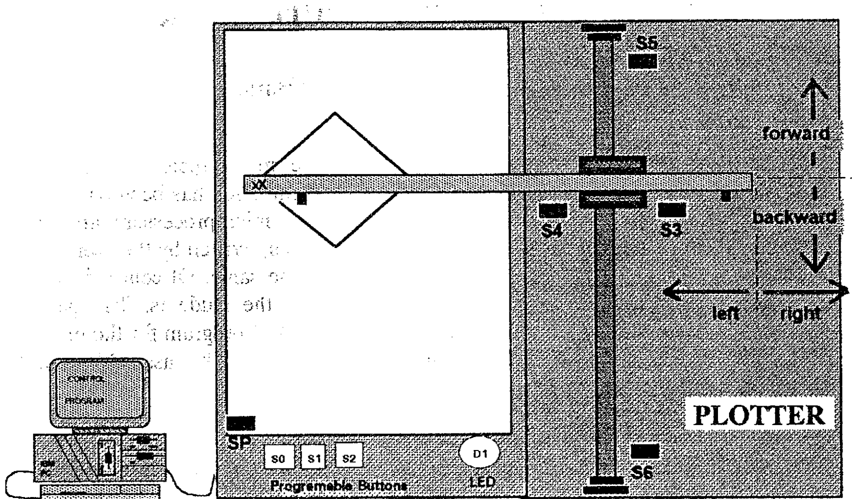

Fig. 1. Real situation: IBM computer controls a plotter.

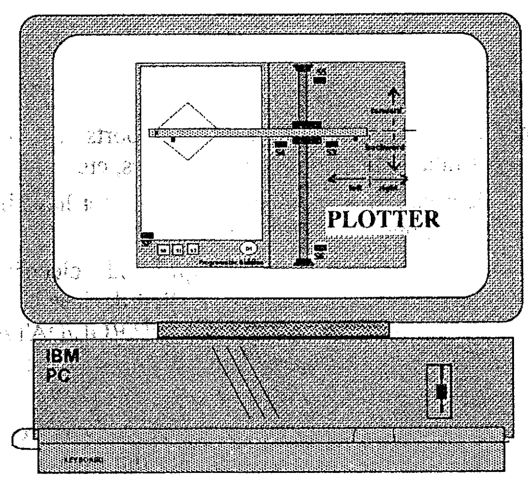

Fig. 2. Simulated environment: a plotter is simulated on the IBM computer's screen.

Output Port Address: 300H:

7	6	5	4	3	2	1	0
X	X	PEN	BW	FW	LF	RT	D1
No meaning		Pen on/off	Backward	Forward	Left	Right	LED on/off
			Controlled by M2 Motor		Controlled by M1 Motor		

Input Port Address: 300H:

7	6	5	4	3	2	1	0
S6	S5	S4	S3	SP	S2	S1	S0
Backward Sensor	Forward Sensor	Right Sensor	Left Sensor	Pen Position Status	Programmable Push Buttons		

Input Port Address: 301H:

7	6	5	4	3	2	1	0
X	X	X	X	X	X	PG2	PG1
No meaning						Pulse Generator of M2 Motor	Pulse Generator of M1 Motor

Fig. 3. Bit assignment of I/O plotter ports (active state is 1).

2. Functional Simulation

Working with simulated devices instead of real environment has some advantages:

- elimination of hardware destruction caused by control errors,
- easy modification of simulated hardware.

Simulation of course is not free of drawbacks. The most important are:

- only selected aspects of hardware can be simulated,
- for complicated simulated hardware speed of working is slower then in a real situations.

As it was mentioned above hardware details are not important from control point of view. Also from user point of view is not important what will be watched: real hardware or hardware simulated by computer. In both situations should be the same reaction for the control. Because only functions of devices are watched, for simulation are important only external reactions -- such accuracy of simulation we will call "functional simulation".

Watching reaction of simulated hardware needs good computer graphics. EGA/VGA IBM computer graphic standard resolution seems to be quite enough for functional simulation purposes.

For better understanding problems of simulation let's consider example of plotter simulation.

3. Simulation Example: Plotter

A plotter low level kinematics scheme is shown on Figure 5. The structure of control and status plotter registers are shown on Figure 3. Two plotter motors are controlled through *LF*, *RT* (left/right) and *BW*, *FW* (backward/forward) bits. Ending sensors for movement at given direction are accessible through bits *S3*, *S4*, *S5* and *S6* of input status port. Movement at vertical and horizontal directions causes step-positioning pulses *PG1* and *PG2*. Plotter pen is controlled through *PEN* bit. There are user push-buttons *S0*, *S1*, *S2* and one LED diode controlled by *D1* bit.

It is worth to mention that plotter model kinematics scheme also describes *two-way positioner* - simplified model of robot arm. Pen bit can be interpreted as "catch" for robot arm.

The Figure 4 shows an *TASM Turbo Assembler* example of controlling the plotter model. Exemplary program draws a diamond shape picture. The size of the picture depends on duration of the *delay* subroutine.

It is important to understand that structures of programs for controlling real devices and simulated devices must be the same. For example, for *TASM* language communication with the real devices is through *in* and *out* microprocessor instructions. When a device is simulated the *INPUT* and *OUTPUT* macro definitions call written for simulation purposes library functions (see include "plptter.mac" statement). *INPUT* and *OUTPUT* macro definitions replace *in* and *out* instructions.

(A) REAL HARDWARE ENVIRONMENT

```
          ideal
          model  large

          jumps
          lf           equ      04h
          rt           equ      02h
          fw           equ      08h
          bw           equ      10h
          pen          equ      20h
          edge         equ      300
          step         equ      200
          repeat_no    equ      7
extrn     _delay:far
stack 100
dataseg

codeseg
macro   make_move action
        mov      dx,300h
        mov      al,action
        OUT      dx,al
        add      cx,step
        call     delay
endm    make_move

proc    delay
        push cx
        call _delay
        pop  cx
        ret
endp

proc    _main    far
        PUSH     DS
        SUB      AX,AX
        push     ax
        mov      cx,edge
        mov      bx,repeat_no
repeat:
        push     bx
        make_move fw+lf+pen
        make_move fw+rt+pen
        make_move bw+rt+pen
        make_move bw+lf+pen
        pop      bx
        dec      bx
        jnz      repeat
        mov      al,00h
        OUT   dx,al

        ret
endp
end     _MAIN
```

(B)SIMULATED ENVIRONMENT

```
          ideal
          model    large
          INCLUDE "PLOTTER.MAC"
          jumps
           lf              equ      04h
           rt              equ      02h
           fw              equ      08h
           bw              equ      10h
           pen             equ      20h
           edge            equ      300
           step            equ      200
           repeat_no       equ      7
extrn     _delay:far
stack     100
dataseg

codeseg
macro   make_move action
        mov      dx,300h
        mov      al,action
        OUTPUT   dx,al
        add      cx,step
        call     delay
endm    make_move

proc    delay
        push cx
        call _delay
        pop  cx
        ret
endp

proc _main far
        CALL     _START

        mov      cx,edge
        mov      bx,repeat_no

repeat:
        push     bx
        make_move fw+lf+pen
        make_movefw+rt+pen
        make_move bw+rt+pen
        make_move bw+lf+pen
        pop      bx
        dec      bx
        jnz      repeat
        mov      al,00h
        OUTPUT   dx,al
        CALL     _STOP
        ret
 endp
end
```

Fig. 4. Exemplary program, written in 8086 assembler language, for drawing a rhomboidal spiral.

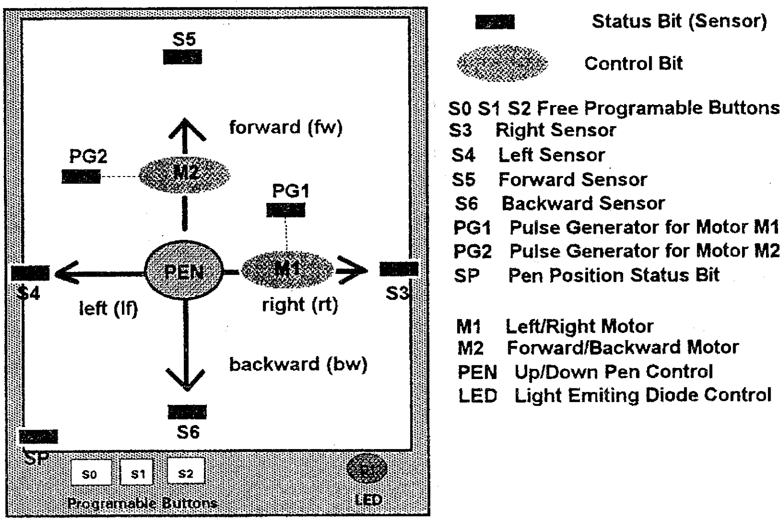

Fig.5. Kinematics scheme of the plotter.

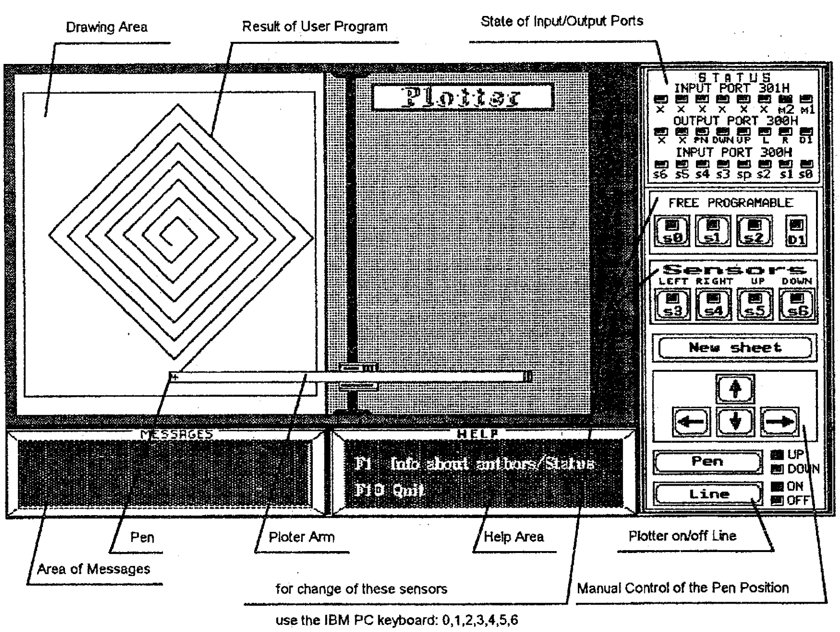

Fig. 6. The plotter simulated on the IBM PC computer screen: all details are the same like for the real plotter.

4. Simulation Difficulties

Problems of simulating real devices on a computer functionally can be classified as follows:

SOUND: working, stopping, destroying actions are using the sound generated by computer; sound generation by IBM computers is very bounded.

VISION: static displaying, on/off permanent fragments of a picture, motion on the screen: object rotation, object panning; there are difficulties of simultaneous simulation of many moving objects on IBM computers.

KEYBOARD/MOUSE: asynchronous using of keyboard/mouse should give chance to change states of simulated environment (e.g. push-button pressing simulation, state changing of sensors by force).

Solving of keyboard problems and simultaneous object animation are forcing multiprogramming. Multiprogramming means using computer interrupts, like keyboard/mouse or real-time clock interrupt at low-level programming in assemblers. Second way of multiprogramming is using multi-thread model programming in languages like MODULA, Top-Speed C. Multi-thread programming uses low-level real-time clock interrupts. Of course the second way of writing simulation environment libraries is easier but consumes more time, what causes slower simulation.

5. Simulation of the Plotter

How does simulated plotter works? For better understanding a *TASM* assembler exemplary program will be considered. General structure of user assembler programs is as follows:

```
include "plotter.mac"
...
call _START()
...
OUTPUT dx,al
...
INPUT al,dx
...
call _STOP
```

The statement *include "plotter.mac"* includes macro definitions for simulated environment.

_START, _STOP functions are for creating/removing the plotter picture on/from the screen as a background.

The function *OUTPUT* moves the plotter arm with the pen at given in output port direction. When the arm reaches the edge of the sheet a sound is produced. The sound means that moving mechanics are destroying.

```
EXTRN    _OUTPUT:FAR,_INPUT:FAR,_STOP:FAR,_START:FAR,_MESSAGE:FAR
EXTERN   _MAIN

MACRO    OUTPUT      DDD1,DDD2    ;DDD1 <-- DX REG. / DDD2 <-- AL REG.
         PUSH        BP
         PUSH        AX
         PUSH        BX
         PUSH        CX
         PUSH        DX
         PUSH        SP
         PUSH        SI
         PUSH        DI
         PUSH        ES
         IFIDNI      <DDD1>,<DX>

          IFIDNI <DDD2> ,<AL>
                 PUSH   AX
                 PUSH   DX
                 CALL   _OUTPUT   ;CALL EXTERNAL PROCEDURE
                 ADD    SP,4
           ELSE
                 MOV    AX,5
                 PUSH   AX
                 CALL   _MESSAGE  ;IF NOT CORRECT CALL ERROR
                 POP    AX
          ENDIF
         ELSE

           MOV    AX,5
           PUSH   AX
           CALL   _MESSAGE          ;IF NOT CORRECT CALL ERROR
           POP    AX
         ENDIF

         POP         ES
         POP         DI
         POP         SI
         POP         SP
         POP         DX
         POP         CX
         POP         BX
         POP         AX
         POP         BP
ENDM                              ;END OF MACRO


MACRO    INPUT       DDD1,DDD2    ;DDD1 <-- DX REG. / DDD2 <-- AL REG.
•••                               ; THE SAME LIKE FOR OUTPUT !!!!!
ENDM                              ;END OF MACRO
```

Fig. 7. Contents of the PLOTTER.MAC macro definition file.

The function *INPUT* gives the state of sensors. There are two situations when state of sensor is changed. First, state of sensors is changed as a result of movement, when a user program is executed. Second, at any moment when a user program is executed the user has possibility to change states of sensors "manually", by pressing keys on IBM PC keyboard. For example press the key *"4"* changes the state of sensor *S4*. In this way the user has chance to "cheat" - is possible to make active left sensor (*LF*) before the plotter arm reaches the left edge of drawing sheet.

States of all sensors and control bits are visible on the computer's screen (Fig. 6).

Above description shows that only functions of the plotter are simulated, not electrical or mechanical details.

All functions of simulated environment are hidden in above described macro definitions. The simplified structure of *plotter.mac* file, where all plotter definitions are placed, is shown on Figure 7.

6. Conclusion

The above presented ideas were applied and checked in writing control programs for the following devices:

- traffic lights control simulation
- stepper motor control simulation
- control of industrial lifts and conveyor belts simulation
- dynamic control of 8-segment LED display and keyboard simulation
- plotter simulation.

The *Microprocessors Architecture Laboratory*, designed for didactic purposes, uses above exercises: instead of real environment students are trained on unreal, simulated devices.

Reference

Baranski W. and Majewski J. (1991): *Functional Simulation.*- Scientific Papers of the Institute of Engineering Cybernetics of the Technical Univ. of Wroclaw, No.89, pp. 7-12.

Borland International Inc. (1988): *Turbo Assembler, User's Guide.*

Nicoud J.D (1991): *Dedicated Tools for Microprocessor Education.*- IEEE Micro.