PC-Interfaced Data Acquisition System with the Atmel AT89C2051 Microcontroller

Introduction

The features of the Atmel AT89C2051 microcontroller offer the user the chance to achieve a very low-cost/performance data acquisition system (DAQS) for low-frequency applications.

The main idea of this application note is to fully use the internal architecture of the microcontroller to obtain the maximum analog-to-digital conversion (ADC) speed, maximum connectivity to **any** IBM-compatible PC, and maximum further development applications.

As it is well known, any stand-alone DAQS must have an ADC suitable for the imposed analog input signals properties, a local memory for temporary data storage and additional hardware for the PC interfacing.

PC interfacing is particularly useful for monitoring analog input signals and data manipulation with a computer program. Furthermore, PC interfacing allows remote control of the DAQS via the Internet.

For slow-varying analog input signals (temperature, pressure, bio-medical signals, speech), a low-speed ADC and, consequently, a low-speed DAQS is needed.

The AT89C2051 was chosen to be the core of such a DAQS because of the following features:

- The internal comparator allows easy implementation of a Successive Approximation ADC (SADC) with an external digital-to-analog converter (DAC) up to 10-bit resolution, if 5V reference voltage is considered.
- Internal Flash memory may store the SADC algorithm and generate the control signals for the DAQS. It may also generate the control signals for the external memory of the DAQS.
- It may also communicate with a host PC where digital data can be easily manipulated and displayed.

This application note will describe a DAQS step-by-step, assuming that the user is familiar with the features of the microcontroller(1), DAC(2), and various logical gates that are used.

Notes: 1. *Programmer's Guide and Instruction Set* – AT89C2051 datasheet (www.atmel.com).
2. Maxim MAX527 datasheet (*www.maxim.com*).
3. Interfacing the Standard Parallel Port document from: *http://www.geocities.com/siliconvalley/bay/8302/parallel.htm*

AT89C2051 Microcontroller

Application Note

Rev. 1489A–MICRO–11/03

The Control Unit of the DAQS

The complete DAQS is illustrated in Figure 1 and any further reference, if not otherwise specified, will be made considering this figure.

The system RESET is used either with the RESET push-button or by the bit D7, through the diode D2 of the host PC's parallel interface (parallel port) DATA_PORT. The group D1, C1, R2 is used for obvious reasons. The system may be manually or automatically restarted.

The useful data and control lines are chosen among the 15 I/O lines of the microcontroller (i.e., 13 out of 15), as shown in Tables 1 and 2.

Figure 1. Data Acquisition System with the Atmel AT89C2051

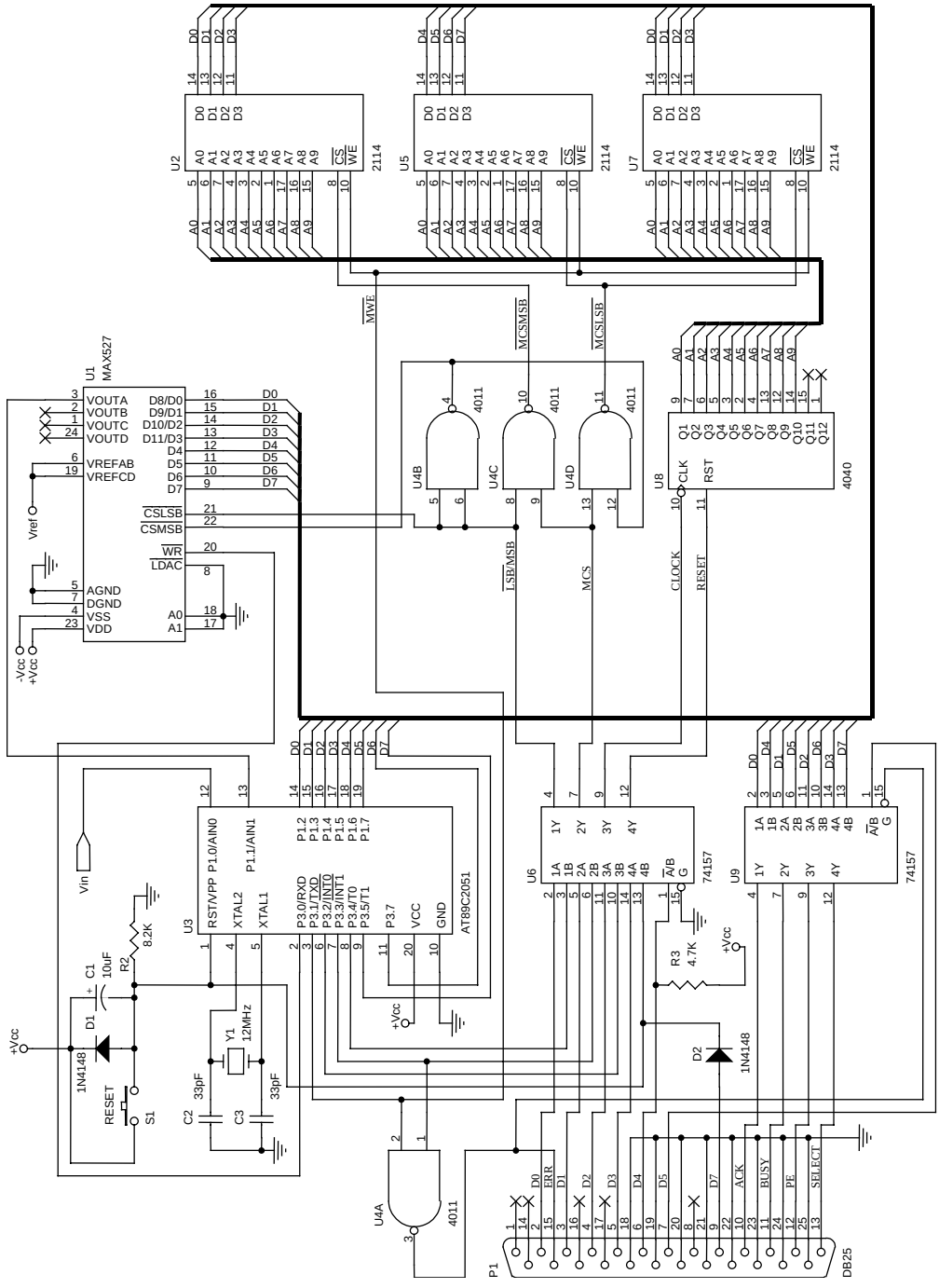


Table 1. Data Line Allocation

I/O Line	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7	P3.7	P3.5
Data Line	D0/D8	D1/D9	D2/D10	D3/D11	D4	D5	D6	D7

Table 2. Control Line Allocation

I/O Line	Signal	Description
P3.0	$\overline{WR}$	Active-low input. When active, it is written in the input DAC registers.
P3.1	$\overline{MWE}$	Active-low signal. When active and a memory chip is selected, it allows memory writing.
P3.2	CLOCK	Address counter clock signal. Used when a new memory address is accessed.
P3.3	MCS	Memory chip select signal.
P3.4	$\overline{LSB}$/MSB	When low, the least significant byte (D0...D7) is selected and when high, the most significant nibble (D8...D11) is selected.

The Data Conversion Block

The Data Conversion block was created with the help of Maxim MAX527[2] DAC (U1 in Figure 1) and the internal comparator of the microcontoller.

The conversion method (i.e. SADC) is a fixed step number one, in each conversion step one bit being found. The conversion algorithm is stored in the internal Flash memory of the microcontroller and involves 12 steps/conversion. It should be mentioned here that because of the microcontroller internal comparator, a practical 10-bit accuracy ADC can be obtained, the least significant 2 bits being neglected.

The settling time of the DAC is 5 µs and the stored conversion program contains delay loops after any data transmissions to the DAC (see the microcontroller program listing in Appendix A).

The External Memory Block

The external memory block (EMB) used in our DAQS uses three memory devices, 2114-type (1024 x 4 bits) because at the time the system was implemented no other memory devices were available. It should be pointed out that other memory devices could be used with a slight modification of the Flash memory program.

The memory devices are labeled U2, U5 and U7 in Figure 1. In this EMB, 1 kilosample, 12 bits each, can be stored.

The address counter is a standard CMOS 4040 (14-bit resolution, buffered outputs), the address being maintained when the internal memory of the microcontroller is used. It is obvious that, initially, the counter is reset.

Because the data bus is 8 bits wide, the EMB is partitioned into two blocks: 8 bits for the least significant byte (U5 and U7 in Figure 1) with the signal $\overline{MCSLSB}$ and 4 bits for the most significant nibble (U2 in Figure 1) with the signal $\overline{MCSMSB}$.

The EMB control signals are realized with the NAND gates U4B, U4C and U4D in Figure 1, parts of a standard CMOS 4011 chip.

In Table 3, the interconnections between the EMB and the microcontroller data bus are presented.

Table 3. Interconnections between the Microcontroller Data Bus and EMB

Data Bus	D7	D6	D5	D4	D3/D11	D2/D10	D1/D9	D0/D8
U2 I/O Lines	x	x	x	x	D3	D2	D1	D0
U5 I/O Lines	D3	D2	D1	D0	x	x	x	x
U7 I/O Lines	x	x	x	x	D3	D2	D1	D0

PC Interfacing

The DAQS is connected to a host PC with a standard parallel interface (i.e., unidirectional)[2][3].

The data is read through the STATE_PORT of the parallel port (4 out of 5 bits). Two muxes, 74157-type (U6 and U9 in Figure 1), and a NAND gate (U4A in Figure 1) are used for this purpose.

It should be noted that using the standard parallel port allows for PC interfacing with any kind of PC, whether old or new.

The main functions of the PC interfacing are as follows:

1. Establishes who has the control over the EMB – the microcontroller or the PC.
2. Controls the data transfer from the EMB to the PC.

Let us shortly describe these features.

1. The input lines of the mux U6 are grouped into two nibbles: the first one (A inputs) is connected to the DATA_PORT of the parallel port (bits D0...D7) where the host PC EMB bits are generated as shown in Table 4.

Table 4. Mux U6, A Nibble

Mux U6, A Inputs	4A	3A	2A	1A
DATA_PORT Register Bits	D3	D2	D1	D0

The port B of the mux U6 is connected to the control signals generated by the microcontroller, as shown in Table 5.

The 4B bit of the mux U6 is connected to the system RESET signal, as well as the D7 bit of the DATA_PORT of the parallel port, allowing restarting the DAQS by the PC without manual control through “RESET” push-button (see Figure 1).

It is easy to see that, when the system is first started, the microcontroller has the control over the DAQS, the output of the U6 being its input port B, so that the microcontroller has the control of the EMB.

When an acquisition cycle is accomplished, the gate U4A generates a signal read by the PC (i.e., the ERROR bit in the STATE_PORT of the parallel port), the PC program makes D4 = 0 and the mux U6 output bits are its input port A ones. Because the system has a common data bus, the I/O lines of the microcontroller must be put in a neutral state. This can be done by writing a “1” at the port lines.

Table 5. Mux U6, B Nibble

Mux U6, B Inputs	4B	3B	2B	1B
P3 Port Bits	x	P3.2	P3.3	P3.4

When data is read, the host PC commutes the control to the microcontroller.

The output of the mux U6 are given in Table 6.

2. Reading data from the EMB is made up by multiplexing the EMB data nibbles, because of the standard parallel port used, through its STATE_PORT.

In Table 7, the data bus connections to the mux U9 inputs are given, the mux U9 output nibble being connected as in Table 8.

Table 6. Mux U6 Outputs

U6 Output	1Y	2Y	3Y	4Y
Control Signals	$\overline{\text{LSB}}$/MSB	MCS	CLOCK	RESET

Table 7. Mux U6 Inputs

U9 Inputs	4B	3B	2B	1B	4A	3A	2A	1A
Data Bus	D7	D6	D5	D4	D3/D11	D2/D10	D1/D9	D0/D8

Table 8. Mux U6 Outputs

U6 Outputs	1Y	2Y	3Y	4Y
Control Signals	$\overline{\text{ACK}}$	BUSY	PE	SELECT

The control bits of the mux U9 are connected as follows:

- $\overline{\text{G}}$ is connected at the U4A gate output, meaning that the data is transmitted to the parallel port only when the DAQS cycle is accomplished.
- $\overline{\text{A}}$/B is connected to the D5 bit in the DATA_PORT of the parallel port, controlling the reading process in the PC memory.

PC Program

The PC program has two main features: a data acquisition program and a GUI.

The data acquisition program, written in ANSI C, performs the following functions:

- Loop test to find out when a data acquisition cycle is accomplished
- Read the EMB stored samples
- Address counter reset
- Address counter incrementing

The GUI is implemented through a LabWindows/CVI™ (National Instruments®) medium. Samples of different signals are shown in Figures 2 through 5.

The software can be made available on a web site to allow remote control of the DAQS wherever an Internet connection is available.

In Appendix B, the listing of the entire PC program is given. Because it is well documented, additional comments are unnecessary.

Figure 2. Sine Wave Input Signal Recovered from its Samples

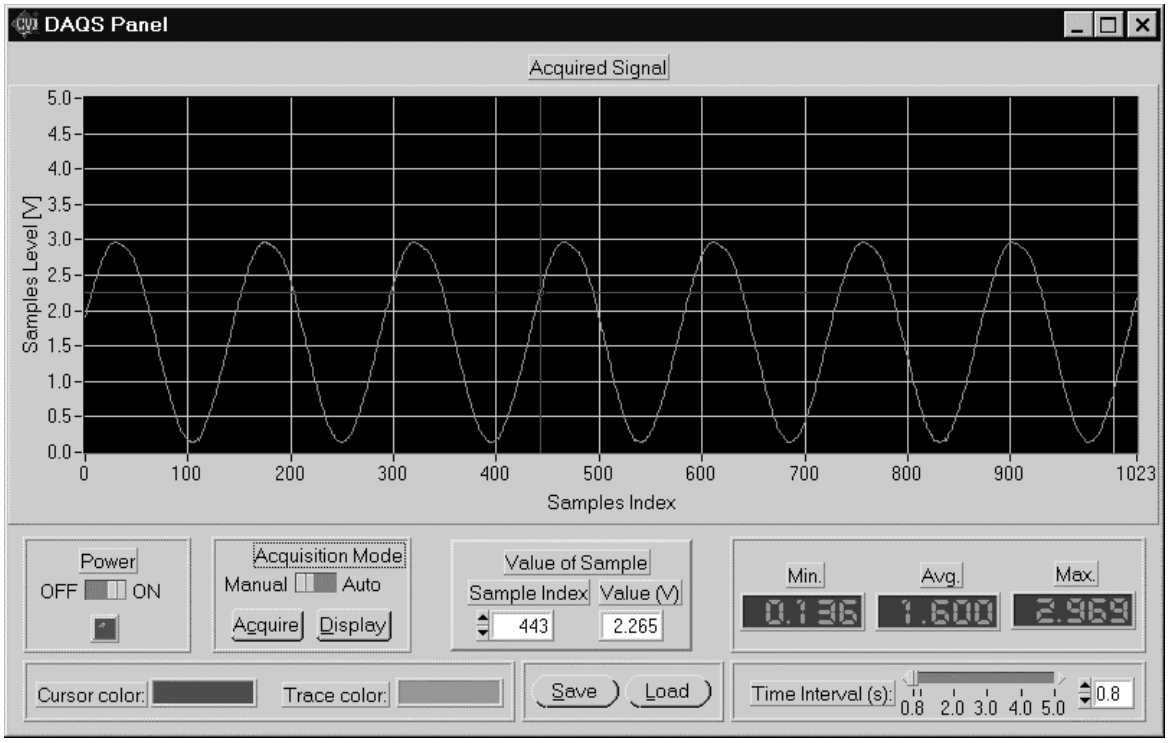


Figure 3. Input Integrated Square Wave Signal Recovered from its Samples

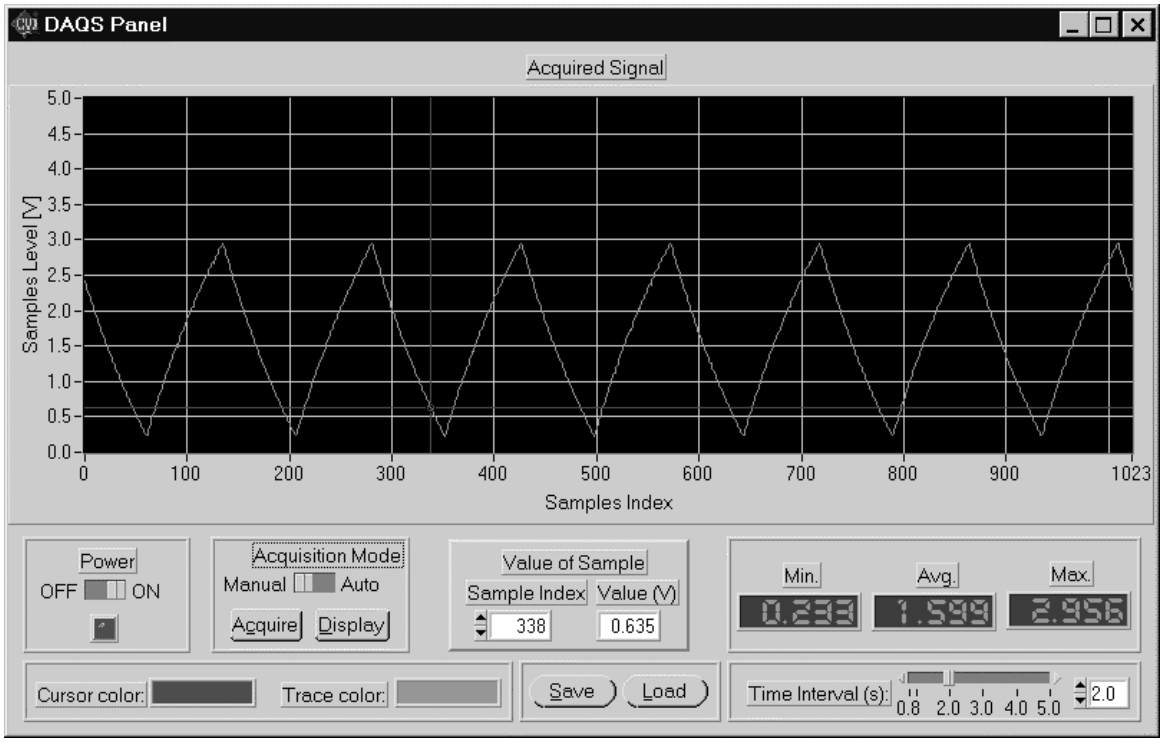


Figure 4. Sample Software Screen

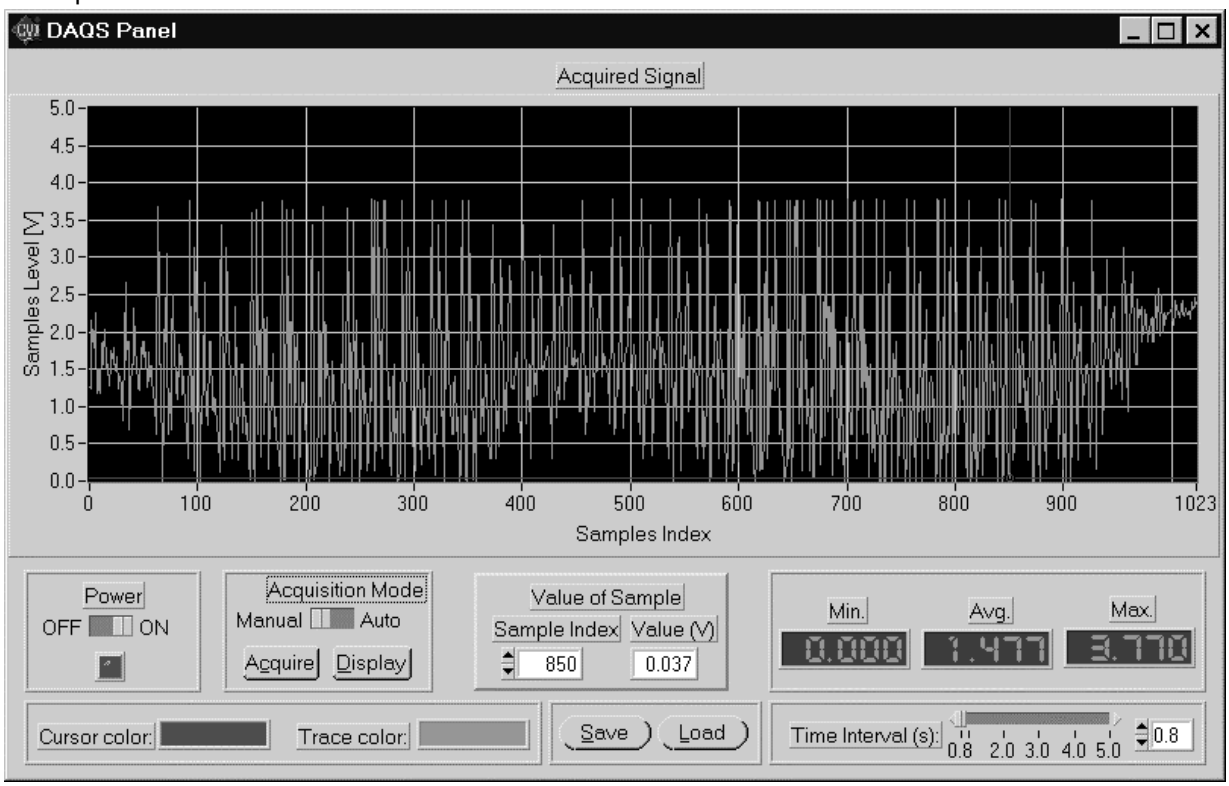
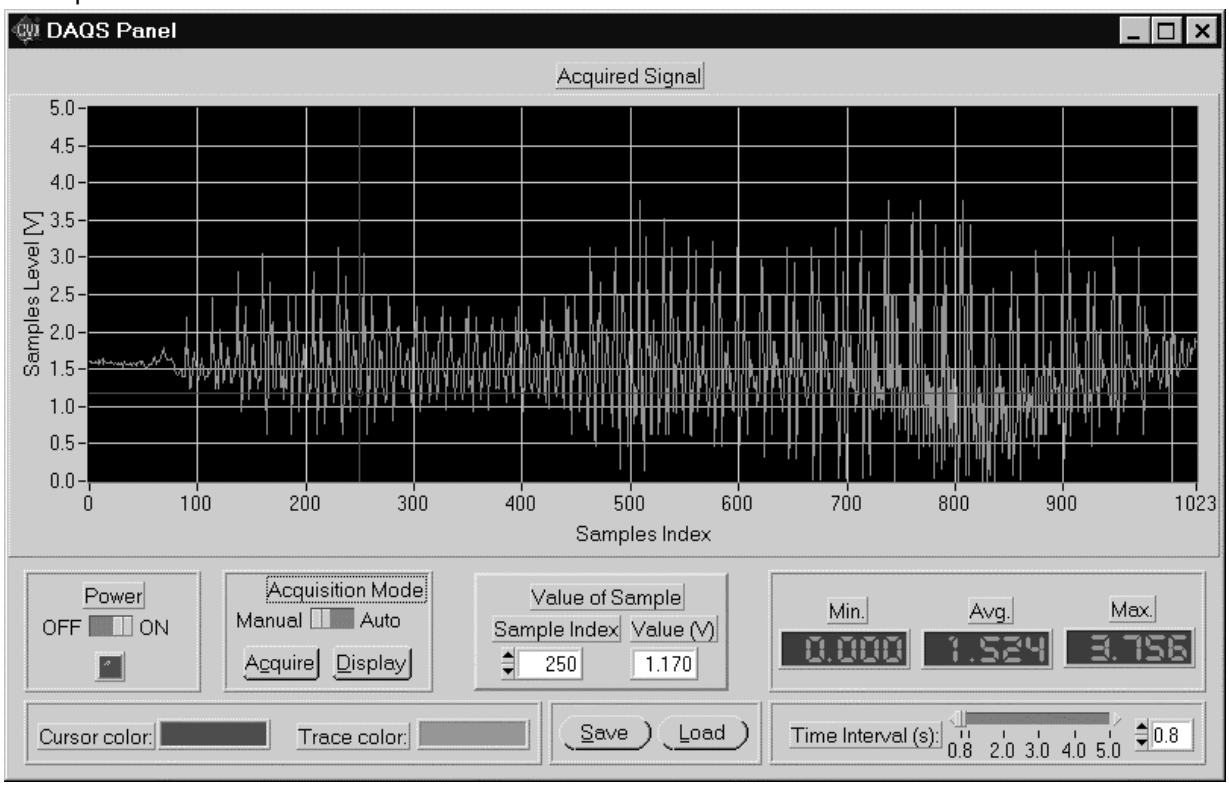


Figure 5. Sample Software Screen



Appendix A: Microcontroller Program

PC – Interfaced Data Acquisition System with Atmel AT89C2051 Microcontroller

Design Engineers:

Nicos A. Zdukos, Undergraduate Student

Cristian E. Onete, Associate Professor

Project Manager:

Cristian E. Onete, Associate Professor
"Gh.Asachi" Technical University
Computer Sciences Department
P.O. Box 1355 Iasi 6
Iasi 6600, Romania
e-mail: conete@cs.tuiasi.ro
Tel.: +40-32-213749

```
; R7,R6 - counters for the number of samples
; R5+R4 - successive approximation register
;
$MOD52
ORG 00H
scriere_DAC MACRO
       CPL P3.0       ; nWR=0
       NOP
       CPL P3.0              ; nWR=1
ENDM
SJMP start
; Delay Routine
intrz:   MOV R1,#2             ; 8us delay
bucla:   DJNZ R1,bucla
         RET
;
start:  MOV R7,#04H
        MOV R6,#0FFH ; 4*255=1020 samples
        MOV P3,#11100111B ; initial conditions
ach:    CALL achiz
        DJNZ R6,ach
        MOV R6,#0FFH
        DJNZ R7,ach
        MOV R6,#04H ; 4+1020=1024 samples
ach1:   CALL achiz
        DJNZ R6,ach1 ; mission accomplished 1024 samples
        MOV P1,#0FFH ; D0-D5=1
        ORL P3,#10100010B ; D6,D7=1, /MWE=1
        SETB P3.3       ; MCS=1
                        ; acquisition accomplished PC takes command

gata:   SJMP gata
;
; One sample acquisition and its storage
achiz:  MOV A,#0
        MOV P1,#03H
        ANL P3,#01011111B       ; D0-D7=0
        scriere_DAC
        CPL P3.4        ; /LSB/MSB=1
        CPL P1.5        ; bit 11=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc1
        CPL P1.5        ; bit 11=0
        XRL A,#00001000B
etc1:   XRL A,#00001000B
        CPL P1.4        ; bit 10=1
```

```
        scriere_DAC
        CALL intrz
        JB P3.6,etc2
        CPL P1.4        ; bit 10=0
        XRL A,#00000100B
etc2:   XRL A,#00000100B
        CPL P1.3        ; bit 9=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc3
        CPL P1.3        ; bit 9=0
        XRL A,#00000010B
etc3:   XRL A,#00000010B
        CPL P1.2        ; bit 8=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc4
        CPL P1.2        ; bit 8=0
        XRL A,#00000001B
etc4:   XRL A,#00000001B
        MOV R5,A        ; in R5 there are the first 4 bits of the sample
scriere_DAC
        CPL P3.4        ; /LSB/MSB=0
        MOV A,#0
        MOV P1,#03H
        ANL P3,#01011111B
        CPL P3.5        ; bit 7=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc5
        CPL P3.5        ; bit 7=0
        XRL A,#10000000B
etc5:   XRL A,#10000000B
        CPL P3.7        ; bit 6=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc6
        CPL P3.7        ; bit 6=0
        XRL A,#01000000B
etc6:   XRL A,#01000000B
        CPL P1.7        ; bit 5=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc7
        CPL P1.7        ; bit 5=0
        XRL A,#00100000B
etc7:   XRL A,#00100000B
        CPL P1.6        ; bit 4=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc8
        CPL P1.6        ; bit 4=0
        XRL A,#00010000B
etc8:   XRL A,#00010000B
        CPL P1.5        ; bit 3=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc9
        CPL P1.5        ; bit 3=0
        XRL A,#00001000B
etc9:   XRL A,#00001000B
        CPL P1.4        ; bit 2=1
```

```
        scriere_DAC
        CALL intrz
        JB P3.6,etc10
        CPL P1.4        ; bit 2=0
        XRL A,#00000100B
etc10:  XRL A,#00000100B
        CPL P1.3        ; bit 1=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc11
        CPL P1.3        ; bit 1=0
        XRL A,#00000010B
etc11:  XRL A,#00000010B
        CPL P1.2        ; bit 0=1
        scriere_DAC
        CALL intrz
        JB P3.6,etc12
        CPL P1.2        ; bit 0=0
        XRL A,#00000001B
etc12:  XRL A,#00000001B
        MOV R4,A        ; in R4 there are the last 8 bits of the sample
; Writing data in the EMB
        CLR P3.1        ; /MWE=0
        NOP
        SETB P3.3       ; MCS=1
        NOP
        XRL P3,#00001010B ; MCS=0,nMWE=1
        CPL P3.4        ; /LSB/MSB=1
        MOV A,R5
        RL A
        RL A
        ANL P1,#00000011B
        ORL P1,A
        CLR P3.1        ; /MWE=0
        NOP
        SETB P3.3       ; MCS=1
        NOP
        XRL P3,#00001010B ; MCS=0,nMWE=1
        CPL P3.4        ; /LSB/MSB=0
        CPL P3.2
        NOP
        CPL P3.2        ; address counter incrementing
        RET
;
END
```

Notes: 1. NOP instructions are only for testing purposes being removed from the final version of the program.

2. The simulated DAQS shows that the acquisition time/sample for a 12 MHz clock frequency is:

   243.738 cycles/1024 (i.e., 4201 Hz) in most favorable case

   268.314 cycles/1024 (i.e., 3816 Hz) in worst case

   when the acquisition cycles are not balanced in microcontroller program. A new release of our microcontroller program was realized where these times are balanced, but the price paid for this balancing is speed reduction.

   The conversion speed can be increased using higher speed microcontroller (higher clock frequencies).

Appendix B: PC Program Listing

PC – Interfaced Data Acquisition System with Atmel AT89C2051 Microcontroller

Design Engineers:

Nicos A. Zdukos, Undergraduate Student

Cristian E. Onete, Associate Professor

Project Manager:

Cristian E. Onete, Associate Professor
"Gh.Asachi" Technical University
Computer Sciences Department
P.O. Box 1355 Iasi 6
Iasi 6600, Romania
e-mail: conete@cs.tuiasi.ro
Tel.: +40-32-213749

```
#include <utility.h>
#include <cvirte.h>
#include <userint.h>
#include <formatio.h>
#include "prj.h"
#define NrEsant 1024 // Number of samples
#define RDate 0x378
#define RStare RDate+1
#define RControl RDate+2
#define SADCtrlMask 0x10
#define SADResetMask 0x80
#define ResetMask 0x08
#define ClockMask 0x04
#define McsMask 0x02

static int panelHandle;
static int plotHandle=0;
int SwitchVal=0,Switch_2Val=0;
unsigned char ctrl;
float vsemnal[NrEsant];
float Vref=5.0;
int nr_esant;
int culoare_trasare=VAL_GREEN;
static char proj_dir[MAX_PATHNAME_LEN];
static char file_name[MAX_PATHNAME_LEN];

void TestSADLiber(void);
void ResetSAD(void);
void Achizitie(void);
void ResetNumarator(void);
void IncrNumarator(void);
unsigned int PreluareDate(void);
void Indicatoare(void);
void DezactivareControale(int);
void ActiveazaCursor(void);
void DezactiveazaCursor(void);

int main (int argc, char *argv[])
{
      if (InitCVIRTE (0, argv, 0) == 0)
            return -1;
      if ((panelHandle = LoadPanel (0, "prj.uir", PANEL)) < 0)
            return -1;
      SuspendTimerCallbacks ();
      GetProjectDir (proj_dir);
```

```
      DisplayPanel (panelHandle);
      ctrl=0x14;
      outp(RDate,ctrl); // Initial Conditions
      SetGraphCursor (panelHandle, PANEL_GRAPH, 1, 0, 0.0);
      RunUserInterface ();
      return 0;
}

int CVICALLBACK PanelCall (int panel, int event, void *callbackData,
            int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_GOT_FOCUS:
                  break;
            case EVENT_LOST_FOCUS:
                  break;
            case EVENT_CLOSE:
                  if(!SwitchVal)
                        QuitUserInterface (0);
// Panel closes only if "Power" button is OFF
                  break;
      }
      return 0;
}

// This function is called when the "Power" switch is selected "ON/OFF"
int CVICALLBACK SwitchCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (panelHandle, PANEL_BINARYSWITCH, &SwitchVal);
                  SetCtrlVal (panelHandle, PANEL_LED, SwitchVal);
                  if(SwitchVal)
                  {
                        DezactivareControale (0);
                  }
                  else
                  {
                        SuspendTimerCallbacks ();
                        if(plotHandle > 0)
                        {
                              DeleteGraphPlot (panelHandle, PANEL_GRAPH,
                                  plotHandle, VAL_IMMEDIATE_DRAW);
                              plotHandle=0;
                        }
                        DezactivareControale (1);
                        SetCtrlVal (panelHandle, PANEL_BINARYSWITCH_2, 0);
                        SetCtrlVal (panelHandle, PANEL_METER, 0.0);
                        SetCtrlVal (panelHandle, PANEL_METER_2, 0.0);
                        SetCtrlVal (panelHandle, PANEL_METER_3, 0.0);
                        nr_esant=0;
                        SetCtrlVal (panelHandle, PANEL_NUMERIC, 0);
                        etCtrlVal (panelHandle, PANEL_NUMERIC_2, 0.0);
                        SetCtrlVal (panelHandle, PANEL_NUMERICSLIDE, 0.8);
                        SetCtrlAttribute (panelHandle, PANEL_TIMER,
ATTR_INTERVAL, 0.800);
                        SetGraphCursor (panelHandle,PANEL_GRAPH, 1,0,0.0);
                        ActiveazaCursor ();
                  }
                  break;
      }
      return 0;
}
```

```
// The function is called when the "Acquisition Mode" switch is selected
int CVICALLBACK Switch_2Call (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
            GetCtrlVal (panelHandle, PANEL_BINARYSWITCH_2, &Switch_2Val);
            if(Switch_2Val)
                  {
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON, ATTR_DIMMED, 1);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_2, ATTR_DIMMED, 1);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_3, ATTR_DIMMED, 1);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_4, ATTR_DIMMED, 1);
      DezactiveazaCursor ();
      ResumeTimerCallbacks ();
                  }
                  else
                  {
      SuspendTimerCallbacks ();
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON, ATTR_DIMMED, 0);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_2, ATTR_DIMMED, 0);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_3, ATTR_DIMMED, 0);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_4, ATTR_DIMMED, 0);
      ActiveazaCursor ();
      SetGraphCursor (panelHandle, PANEL_GRAPH, 1, nr_esant,
       vsemnal[nr_esant]);
                  }
                  break;
      }
      return 0;
}

// This function is called when the "Display" button is pressed
int CVICALLBACK ButtonCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                        if(plotHandle > 0)
                        DeleteGraphPlot (panelHandle, PANEL_GRAPH,
                          plotHandle, VAL_IMMEDIATE_DRAW);
                        plotHandle = PlotY (panelHandle, PANEL_GRAPH,
                          vsemnal, 1024, VAL_FLOAT,
VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, culoare_trasare);
                        SetGraphCursor (panelHandle, PANEL_GRAPH,
                          1, nr_esant, vsemnal[nr_esant]);
                        Indicatoare ();
                        SetCtrlVal (panelHandle, PANEL_NUMERIC_2,
                          vsemnal[nr_esant]);
                  break;
     }
      return 0;
}

// This function is called when the "Acquisition Time (s)" button is pressed
// This is the time interval in between two successive automatic acquisitions
(PC control mode)
int CVICALLBACK Button_2Call (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  SetWaitCursor (1);
                  ResetSAD ();
```

```
                  TestSADLiber ();
                  Achizitie ();
                  SetWaitCursor (0);
                  break;
      }
      return 0;
}

// This function is calld on timer's pulse (i.e. each time interval)
int CVICALLBACK TimerCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_TIMER_TICK:
                  SetWaitCursor (1);
                  ResetSAD ();
            TestSADLiber ();
            Achizitie ();
            SetWaitCursor (0);
                  if(plotHandle > 0)
                  DeleteGraphPlot (panelHandle, PANEL_GRAPH, plotHandle,
                     VAL_IMMEDIATE_DRAW);
                  plotHandle = PlotY (panelHandle, PANEL_GRAPH, vsemnal,
                     1024, VAL_FLOAT, VAL_THIN_LINE,
VAL_EMPTY_SQUARE, VAL_SOLID, 1, culoare_trasare);
                  Indicatoare ();
                  SetCtrlVal (panelHandle, PANEL_NUMERIC_2,
                     vsemnal[nr_esant]);
                  break;
      }
      return 0;
}

// This function is called when "Sample Index" is selected
// A number in between 0 and 1023
int CVICALLBACK NumCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (panelHandle, PANEL_NUMERIC, &nr_esant);
                  SetCtrlVal (panelHandle, PANEL_NUMERIC_2,
                     vsemnal[nr_esant]);
                  SetGraphCursor (panelHandle, PANEL_GRAPH, 1, nr_esant,
                     vsemnal[nr_esant]);
                  break;
      }
      return 0;
}

// This function is called on "Cursor color" selection
int CVICALLBACK CursorColorCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int culoare_cursor;
      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (panelHandle, PANEL_COLORNUM, & culoare_cursor);
                  SetCursorAttribute (panelHandle, PANEL_GRAPH, 1,
ATTR_CURSOR_COLOR, culoare_cursor);
                  break;
      }
      return 0;
}
```

```
// This function is called on "Trace color" selection
int CVICALLBACK TrasareColorCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (panelHandle, PANEL_COLORNUM_2, &
                   culoare_trasare);
                  if(Switch_2Val==0 && plotHandle > 0)
                  {
                  DeleteGraphPlot (panelHandle, PANEL_GRAPH, plotHandle,
                   VAL_IMMEDIATE_DRAW);
                  plotHandle = PlotY (panelHandle, PANEL_GRAPH, vsemnal,
                   1024, VAL_FLOAT, VAL_THIN_LINE,
VAL_EMPTY_SQUARE, VAL_SOLID, 1, culoare_trasare);
                  }
                  break;
      }
      return 0;
}

// This function is called on "Time Interval" selection
int CVICALLBACK TCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      float timp;
      switch (event) {
            case EVENT_COMMIT:
            GetCtrlVal (panelHandle, PANEL_NUMERICSLIDE, & timp);
            SetCtrlAttribute (panelHandle, PANEL_TIMER, ATTR_INTERVAL, timp);
            break;
      }
      return 0;
}

// This function is called when the "Save" button is pressed
int CVICALLBACK SaveCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  if (FileSelectPopup (proj_dir, "*.sad", "*.sad", "Save
                   File", VAL_OK_BUTTON, 0, 1, 0, 1,
                        file_name) > 0);
                  {
                  ArrayToFile (file_name, vsemnal, VAL_FLOAT, 1024, 1,
                   VAL_GROUPS_TOGETHER,
VAL_GROUPS_AS_COLUMNS, VAL_CONST_WIDTH, 8, VAL_BINARY, VAL_TRUNCATE);
                  }
                  break;
      }
      return 0;
}

// This function is called when the "Load" button is pressed
int CVICALLBACK LoadCall (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  if (FileSelectPopup (proj_dir, "*.sad", "*.sad", "Load
                     File", VAL_OK_BUTTON, 0, 1, 0, 1,
                  file_name) == 1)
                  {
```

```
                  FileToArray (file_name, vsemnal, VAL_FLOAT, 1024, 1,
                     VAL_GROUPS_TOGETHER,
VAL_GROUPS_AS_COLUMNS, VAL_BINARY);
                  if(plotHandle > 0)
                  DeleteGraphPlot (panelHandle, PANEL_GRAPH, plotHandle,
                     VAL_IMMEDIATE_DRAW);
                  plotHandle = PlotY (panelHandle, PANEL_GRAPH, vsemnal,
                       1024, VAL_FLOAT, VAL_THIN_LINE,
VAL_EMPTY_SQUARE, VAL_SOLID, 1, culoare_trasare);
                  SetGraphCursor (panelHandle, PANEL_GRAPH, 1, nr_esant,
                       vsemnal[nr_esant]);
                  Indicatoare ();
                  SetCtrlVal (panelHandle, PANEL_NUMERIC_2,
                       vsemnal[nr_esant]);
                  }
                  break;
      }
      return 0;
}
// Testing if DAQS has finished data acquisition (1024 samples)
void TestSADLiber(void)
{
      while((inp(RStare)&0x08)!=0)
      {
      SyncWait (Timer(),0.3);
    }
}

// Rests DAQS
void ResetSAD(void)
{
      ctrl=ctrl ^ SADResetMask;
      outp(RDate,ctrl);
      SyncWait (Timer(),0.05);
      ctrl=ctrl ^ SADResetMask;
      outp(RDate,ctrl);
      SyncWait (Timer(),0.4); // Wait for DAQS to acquire data
}

// Controls EMB reading by the host
void Achizitie(void){
      int index;
      ctrl=ctrl ^ SADCtrlMask;
      outp(RDate,ctrl); // DAQS Gain Control
      SyncWait (Timer(),0.001);
      ResetNumarator(); // Reset the address counter
      for(index=0; index < NrEsant; index++)
      {
            ctrl=ctrl ^ McsMask;
            outp(RDate,ctrl); // MCS=1
            vsemnal[index]=(PreluareDate()/4095.0)*Vref;
            ctrl=ctrl ^ McsMask;
            outp(RDate,ctrl); // MCS=0
            IncrNumarator();
      }
      ResetNumarator();
      ctrl=ctrl ^ SADCtrlMask;
      outp(RDate,ctrl); // Give back control to system
}

// Reset address counter
void ResetNumarator(void)
{
      ctrl=ctrl ^ ResetMask; // RESET=1
      outp(RDate,ctrl);
```

```
      SyncWait (Timer(),0.001);
      ctrl=ctrl ^ ResetMask; // RESET=0
      outp(RDate,ctrl);
      SyncWait (Timer(),0.001);
}

// Current address EMB reading
unsigned int PreluareDate(void)
{
      unsigned char temp,lsb,lsbl,lsbh,msb;
      unsigned int data;
      temp=(inp(RStare) ^ 0x80); // BUSY bit is hardware inverted
      lsbl=((temp & 0xc0)>>6) | ((temp & 0x10)>>1) | ((temp & 0x20)>>3);
                  // lsbl contains the bits: 0 0 0 0 D3 D2 D1 D0
      ctrl=ctrl ^ 0x20;
      outp(RDate,ctrl);
      temp=(inp(RStare) ^ 0x80);
      lsbh=((temp & 0xc0)>>2) | ((temp & 0x10)<<3) | ((temp & 0x20)<<1);
                  // lsbh contains the bits: D7 D6 D5 D4 0 0 0 0
      ctrl=ctrl ^ 0x21;
      outp(RDate,ctrl);
      temp=(inp(RStare) ^ 0x80);
      msb=((temp & 0xc0)>>6) | ((temp & 0x10)>>1) | ((temp & 0x20)>>3);
                 // msb contains the bits: 0 0 0 0 D11 D10 D9 D8
      ctrl=ctrl ^ 0x01;
      outp(RDate,ctrl);
      lsb=lsbl | lsbh;
      data=msb*256+lsb;
      return (data);
}
// Address counter incrementing
void IncrNumarator(void)
{
      ctrl=ctrl ^ ClockMask; // CLOCK=1-->0
      outp(RDate,ctrl);
      ctrl=ctrl ^ ClockMask; // CLOCK=0-->1
      outp(RDate,ctrl);
}
// Data samples statistics of the whole EMB content (1024 samples):
//                minimum ("Min."), average ("Avg."), maximum ("Max.")
void Indicatoare(void)
{
      int index;
      float suma=0.0,minim=5.0,maxim=0.0;
      for(index=0; index < NrEsant; index++)
      {
            suma+=vsemnal[index];
            if (vsemnal[index] < minim)
                  minim=vsemnal[index];
            if (vsemnal[index] > maxim)
                  maxim=vsemnal[index];
      }
      SetCtrlVal (panelHandle, PANEL_METER_2, minim);
      SetCtrlVal (panelHandle, PANEL_METER, suma/1024.0);
      SetCtrlVal (panelHandle, PANEL_METER_3, maxim);

}
// Activate/deactivate panel controls (buttons, switches, graph)
void DezactivareControale(int dimm)
{
      SetCtrlAttribute (panelHandle, PANEL_GRAPH, ATTR_DIMMED, dimm);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON, ATTR_DIMMED, dimm);
      SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_2,
        ATTR_DIMMED, dimm);
```

```
        SetCtrlAttribute (panelHandle, PANEL_BINARYSWITCH_2,
          ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_METER, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_METER_2, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_METER_3, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_NUMERIC, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_NUMERIC_2, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_TEXTMSG, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_COLORNUM, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_COLORNUM_2, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_NUMERICSLIDE, ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_3,
          ATTR_DIMMED, dimm);
        SetCtrlAttribute (panelHandle, PANEL_COMMANDBUTTON_4,
          ATTR_DIMMED, dimm);
}
// Graph cursor activating
void ActiveazaCursor(void)
{
        SetCursorAttribute (panelHandle, PANEL_GRAPH, 1, ATTR_CROSS_HAIR_STYLE,
           VAL_LONG_CROSS);
        SetCursorAttribute (panelHandle, PANEL_GRAPH, 1,
           ATTR_CURSOR_POINT_STYLE, VAL_EMPTY_CIRCLE);
}
// Graph cursor deactivating
void DezactiveazaCursor(void)
{
        SetCursorAttribute (panelHandle, PANEL_GRAPH, 1, ATTR_CROSS_HAIR_STYLE,
           VAL_NO_CROSS);
        SetCursorAttribute (panelHandle, PANEL_GRAPH, 1,
           ATTR_CURSOR_POINT_STYLE, VAL_NO_POINT);
}
```

Program Code Authorship

Design Engineers:

Nicos A. Zdukos, Undergraduate Student

Cristian E. Onete, Associate Professor

“Gh.Asachi” Technical University

Project Manager:

Cristian E. Onete, Associate Professor
“Gh.Asachi” Technical University
Computer Sciences Department
P.O. Box 1355 Iasi 6
Iasi 6600, Romania
e-mail: conete@cs.tuiasi.ro
Tel.: +40-32-213749

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe

Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia

Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan

9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

1489A–MICRO–11/03 xM