ANNALS OF FACULTY ENGINEERING HUNEDOARA
– International Journal of Engineering
Tome XI (Year 2013) – FASCICULE 3 (ISSN 1584 – 2673)

1. Corina Daniela CUNȚAN, 2. Ioan BACIU

EXPERIMENTAL MODEL FOR OPTICAL HIGHLIGHTING OF DATA INPUT IN THE MEMORY OF A MICROCONTROLLER

1-2. UNIVERSITY POLITEHNICA TIMIȘOARA, FACULTY OF ENGINEERING HUNEDOARA, ELECTRICAL ENGINEERING AND INDUSTRIAL INFORMATICS DEPARTMENT, ROMANIA

ABSTRACT: Using microcontrollers in the industrial processes has a high dynamics in the current period. For these reasons, the paper presents a microcontroller programming with optical highlighting of its operation. The application uses an ATmega64 microcontroller, which commands the sequentially entry of 8 bits, and highlights it by optical LED display. The sequentially entry is made in both directions, the entry selection being realized manually, with the possibility to reset at any time the entered sequence. The application is provided with its own voltage stabilizer, to protect the microcontroller and to supply the required power in case of maximum power consumption when setting maximum number of outputs.
KEYWORDS: microcontroller, optical LED display, testing program

INTRODUCTION

A microcontroller is a circuit realized on a single chip, typically containing: the central processing unit, the clock generator (to which a quartz crystal must be added from outside) or, in less demanding applications, a RC circuit), volatile memory (RAM), nonvolatile memory (ROM/PROM/EPROM/EEPROM), I/O serial and parallel devices, interrupt controller, DMA controller, counters / timers, A/D and D/A converters, etc., peripherals.

With a MC, we can realize an integrated controller (Embedded Controller, EC). An embedded controller is part of a system built for a specific purpose, other than usual general calculations. Besides the MC, an embedded controller requires an additional hardware to perform its function.[1]

A microcontroller can be defined based on its simplified representation. (Fig. 1)

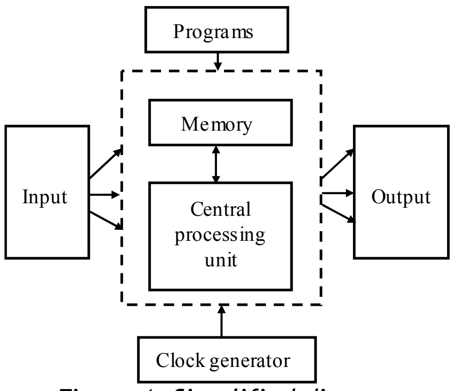

Figure 1. Simplified diagram of a microcontroller

As inputs, we are usually using signals from individual switches or from transducers (temperature, pressure, photo, specialized transducers). The inputs can be digital or analog. The digital inputs convey discrete signals, the information "read" being the information to be sampled when reading that line. The analog inputs convey the information expressed by continuous functions of time. "Reading" them by the microcontroller requires the presence of circuits able to process this information, i.e. analog comparators or analog-to-digital converters, whose outputs are read by MC.

The outputs can be analog, in which case they actually represent outputs of the analog-to-digital converters, or digital, in which case the information is generally stored on them until a new entry is operated by UC at a port of MC. The outputs can control display devices, relays, motors, loudspeakers, etc.[1]

PRESENTATION OF THE PAPER

The paper presents a programming modality of an ATmega64 microcontroller, and a modality to optically view the sequentially entry in its memory.

The application uses an ATmega64 microcontroller that commands the sequentially entry of 8-bit sequences and highlights it by optical LED display. (Fig.2 - Wiring diagram of the microcontroller application).

To highlight the data entry, which is carried out by hand with the buttons B1-B10 connected to the pins 25-34, we connect one LED per output. For each group of 8 outputs, the LEDs are connected with the common cathode, circuit that closes to ground through a resistor whose role is to limit the *output current. Each group of 8 outputs is entered into the microcontroller by means of 2 buttons, one allowing the direct entry and the other one the reverse entry of data.*

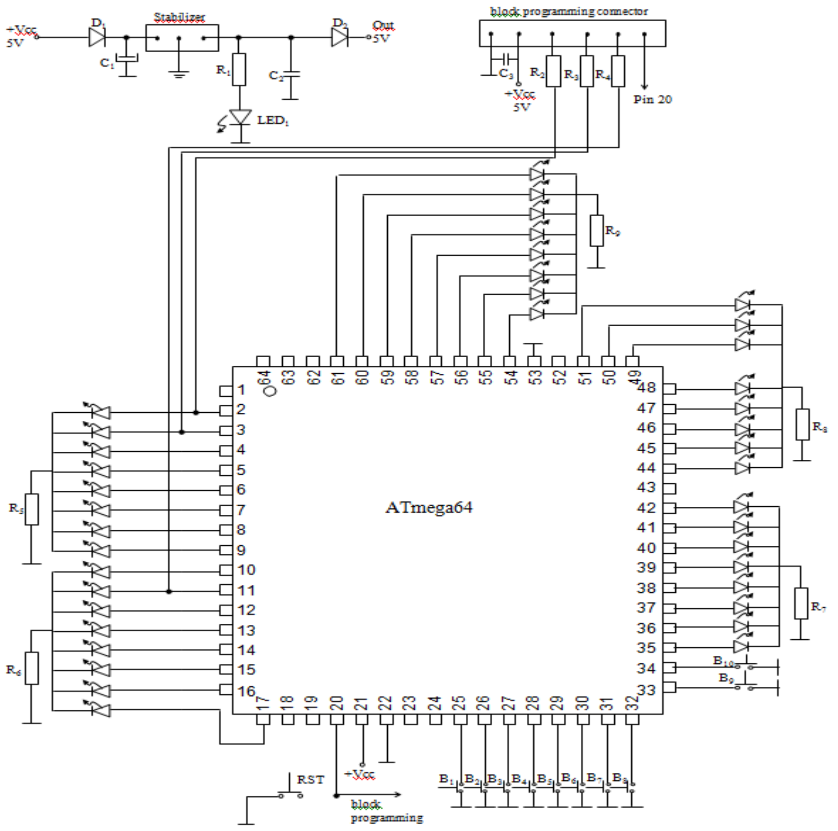

Figure 2. Wiring diagram of the microcontroller application

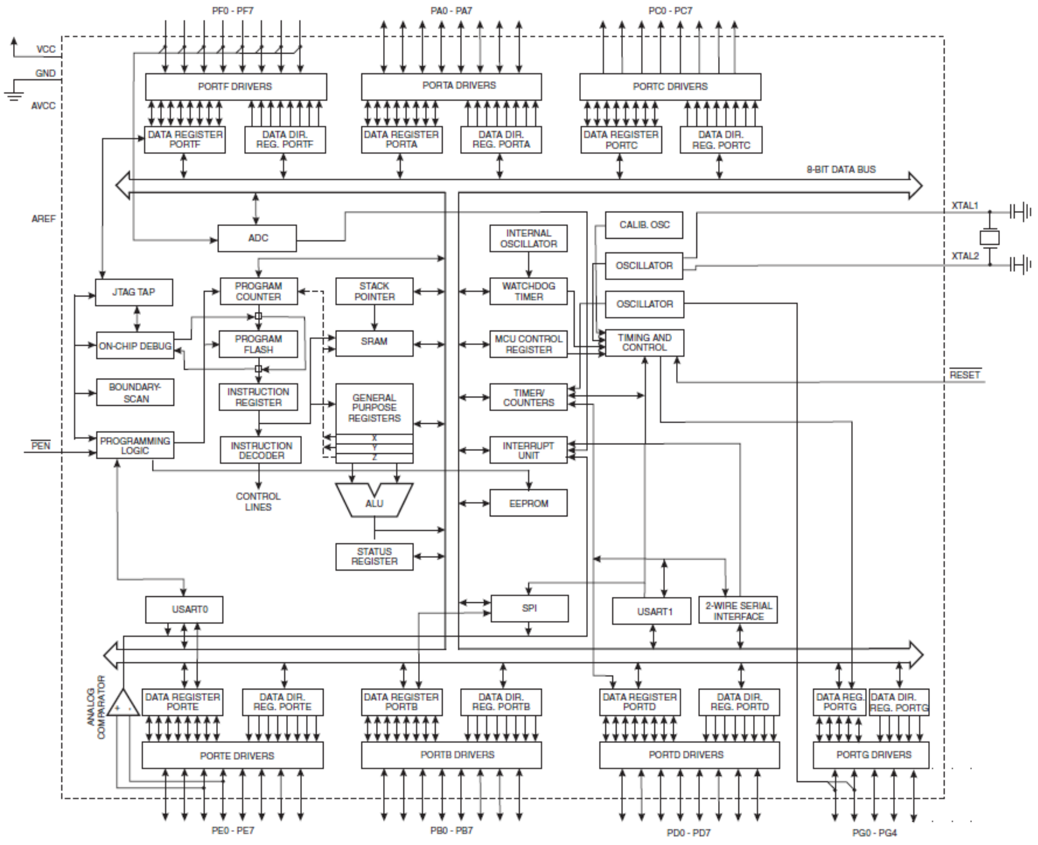

Figure 3. Block diagram of the microcontroller

With the Reset button (BR), we can reset the sequence of data entered up to that point.

To enter the program in the microcontroller memory, it was provided a connector (CP) that connects the above-presented circuitry with the microcontroller programmer.

The application is provided with its own power supply, realized with a voltage stabilizer of 5V, type 7805. The stabilizer is supplied via a rectifier diode used for reverse supply protection.[3]

It is also provided with accidental voltage peaks decoupling capacitors, located at the power source output and the power supply pin of the microcontroller (Pin no. 21).

The microcontroller operation is highlighted by a program written in C language, which is transferred to the microcontroller memory by means of a programming circuit.

The ATmega64 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega64 achieves throughputs approaching 1 MIPS per MHz, allowing the system designer to optimize the power consumption versus the processing speed. [2](Fig. 3)

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.[2]

The ATmega64 provides the following features: 64 Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 2 Kbytes EEPROM, 4 Kbytes SRAM, 53 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), four flexible Timer/Counters with compare modes and PWM, two USARTs, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain, programmable Watchdog Timer with internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and the interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.

The ADC Noise Reduction mode stops the CPU and all I/O modules, except the asynchronous timer and ADC, to minimize switching noise during the ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main Oscillator and the asynchronous timer continue to run.[2]

The device is manufactured using Atmel's high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot Program can use any interface to download the Application Program in the Application Flash memory. The software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. [2]

By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega64 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

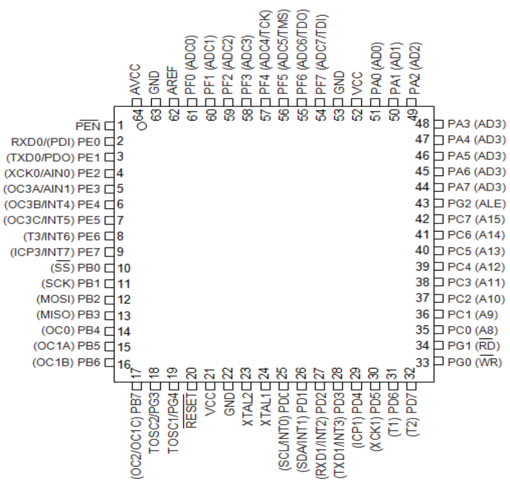

Figure 4. Configuration of the pins

The ATmega64 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/ Simulators, In-Circuit Emulators, and Evaluation kits.

The configuration of the ATmega64 microcontroller pins (Fig. 4) imposes the programming mode.

By successively pressing the B1 button connected to the pin no. 25, we can set the sequential outputs 10-17, and by successively pressing the B2 button we can set, in reverse order, the same outputs. In the same way, by pressing the buttons B3 and B4 we set the outputs 2-9, the buttons 5 and 6 set the outputs at the pins 54-61, by means of the buttons 7 and 8 we set the outputs 44-51, and with the buttons B9 and B10 we set the outputs 35-42. This order is provided by the C programming language so that, by pressing the button, the LED afferent to the selected output lights up. [2][4]

```
#include <avr/io.h>
int main (void)
{   int x,y, yy;
  DDRA |= 0xFF; // Set LED as output
  DDRB |= 0xFF; // Set LED as output
  DDRC |= 0xFF; // Set LED as output
  DDRE |= 0xFF; // Set LED as output
  DDRF |= 0xFF; // Set LED as output
  DDRD =  0b00000000;
  PORTD = 0b11111111;
  DDRG =  0b00000000;
  PORTG = 0b00000011;
  TCCR1B |= ((1 << CS10) | (1 << CS11)); // Set up timer at Fcpu/64
  for (;;)
  {   x=PIND; y=PING;
          // Check timer value in if statement, true when count matches 1 second
    if ((TCNT1 >= 16000))
    { yy = y | 0b11111101;  yy ^=    0b11111101;
          if(yy==0b00000000)
                  {PORTC = PORTC << 1;
                   //PORTC |= 0b10000000; }
                  yy = y | 0b11111110;  yy ^=    0b11111110;
                  if(yy==0b00000000)
                  { PORTC = PORTC >> 1; PORTC |= 0b10000000; }
                  yy = x | 0b01111111;  yy ^=    0b01111111;
                  if(yy==0b00000000)
                  { PORTA = PORTA << 1;
                    //PORTA |= 0b10000000; }
                    yy = x | 0b10111111;  yy ^=    0b10111111;
                  if(yy==0b00000000)
                  { PORTA = PORTA >> 1; PORTA |= 0b10000000; }
                  yy = x | 0b11011111;  yy ^=    0b11011111;
                  if(yy==0b00000000)
                  { PORTF = PORTF << 1;
                    //PORTF |= 0b10000000; }
                  yy = x | 0b11101111; yy ^=    0b11101111;
                  if(yy==0b00000000)
                  { PORTF = PORTF >> 1; PORTF |= 0b10000000; }
                  yy = x | 0b11110111;  yy ^=    0b11110111;
                  if(yy==0b00000000)
                  { PORTE = PORTE << 1;
                    //PORTE |= 0b10000000; }
                  yy = x | 0b11111011; yy ^=    0b11111011;
                  if(yy==0b00000000)
                  {PORTE = PORTE >> 1;
                    PORTE |= 0b10000000; }
                  yy = x | 0b11111101; yy ^=    0b11111101;
                  if(yy==0b00000000)
                  { PORTB = PORTB << 1;
                   // PORTB |= 0b10000000; }
                  yy = x | 0b11111110; yy ^=    0b11111110;
                  if(yy==0b00000000)
                  { PORTB = PORTB >> 1; PORTB |= 0b10000000; }
                 TCNT1 = 0; // Reset timer value }}}
```

CONCLUSIONS

In this paper, we highlighted the possibility of data entry into the microcontroller memory. The data entry is realized sequentially, bit-by-bit, from the control buttons, each output being optically visualized with a LED corresponding to the set bit.

The entry of information in the microcontroller memory is performed on each byte, with the possibility to change the direction of entry by means of two buttons for each set of 8 bits.

The circuit is provided with a button for general reset of the presented application and of the programmer used to enter the program in the microcontroller memory.

For additional protection of the microcontroller, it was provided its own power supply circuit containing decoupling capacitors on the supply pins of the microcontroller.

REFERENCES

[1] http://www.unitbv.ro/faculties/biblio/interfete_specializate/curs.pdf
[2] http://www.atmel.com/Images/Atmel-2490-8-bit-AVR-Microcontroller-ATmega64-L_datasheet.pdf
[3] Ciugudean M., Voltage Stabilizers with linear integrated circuits. Sizing, Publisher: Editura de Vest, Timişoara, 2001.
[4] http://www.circuitstoday.com/avr-microcontroller-tutorial