

An Introduction to Microcontrollers through a Practical Example: A rotating LED Display

Robin Stridh

Teknisk fysik

761228-6919

`rost7041@student.uu.se`

29th April 2002

Abstract

This report introduces the reader to the use of microcontrollers, focusing on a project where a virtual display consisting of only seven rotating light emitting diodes (LED's) was built. It will also give the reader enough information to start using microcontrollers and also provide instructions to build and program the rotating display in the example.

Contents

1	Introduction	2
1.1	The Microcontroller	2
1.2	Programming an Atmel AVR Microcontroller	2
1.3	Digital I/O on an Atmel AVR Microcontroller	3
1.4	Overview of the project	3
2	Functional Description	3
2.1	The Rotating Display	3
2.2	The IR Transmission	4
2.3	Selection of Components	4
2.4	Assembly of the Display	5
3	Programming the Rotating Display	6
3.1	Program 1: <code>hejsan.c</code> (AT90S2313)	6
3.2	Program 2: <code>receiver.c</code> (AT90S2313)	7
3.3	Program 3: <code>sender.c</code> (AT90S8515)	8
3.4	Program 4: <code>pcsender.cc</code> (MS DOS)	8
4	The Resulting Device	8
	References	9
A	C-code	10
A.1	Program 1: <code>hejsan.c</code> (AT90S2313)	10
A.2	Program 2: <code>receiver.c</code> (AT90S2313)	11
A.3	Program 3: <code>sender.c</code> (AT90S8515)	13
A.4	Program 4: <code>pcsender.cc</code> (MS DOS)	15

1 Introduction

Microcontrollers have increased in functionality and dramatically reduced in cost recently, which has led to an explosion in their use. Today you can find them in almost all modern consumer electronics, ranging from cars to refrigerators, but also in cheaper devices like toys.

An electronic device that only a few years ago consisted of tens of integrated circuits and other components, can today often be built with only one single microcontroller and a few other components. All the functionality that earlier had to be implemented by connecting the outputs from some circuits to the input of others by wiring on a circuit board, can nowadays be programmed into the microcontroller. This not only reduces the cost of the components, but also the development cost. If you make a mistake, you do not need to change the physical layout, but can instead change a few lines of code in the microcontroller.

The reader of this text is expected to have a little knowledge of basic electronics and be familiar with C programming, to gain most of its content.

1.1 The Microcontroller

For an easier understanding of the text, the basic concepts of a microcontroller in general are described here very briefly. The specific AVR[®] microcontrollers AT90S2313 and AT90S8515 from Atmel are described in a little more detail, since these are used in the project.

The simplest description of a microcontroller is that it is a complete computer on one chip. Of course it is not a Pentium running Windows, but a greatly scaled-down computer that still contains the most essential parts. Which parts are included differ between different microcontrollers, but some of them are always necessary. These include the central processing unit (CPU), the program memory, the data memory, data busses and in/out (I/O) ports. There may also be an internal clock generator, timers, counters, digital to analogue (DAC) and/or analogue to digital (ADC) converters and much more.

One of the most common applications of microcontrollers is in so-called embedded systems. These are small systems in, for example, cars, refrigerators, CD players or almost anything where a small, independent system is needed to perform a specified task. Since microcontrollers have become very cheap recently, they are often used instead of ordinary electronics. Because the engineering work with microcontrollers is concentrated on programming, it is easy to modify an existing system by modifying the program code instead of the physical construction, and it is therefore much cheaper. The systems can also contain very few components (typically a microcontroller, a power supply, some driving circuits for the microcontroller, some sensors to give input to the microcontroller and the output devices that are to be controlled).

1.2 Programming an Atmel AVR Microcontroller

All Atmel AVR microcontrollers can be programmed from a common PC, both directly in assembler code and in C, using a C compiler adapted to these chips. Every piece of code for this project is written in C and compiled with the free compiler CodeVisionAVR¹. After the code has been compiled, it must be downloaded to the chip and this can be done in several ways. The easiest way is to use a special

¹There is a link to CodeVision's homepage in the Reference section, where the compiler can be downloaded.

hardware (for example Atmel STK500). A cheaper alternative is to build the small needed programming circuit yourself².

1.3 Digital I/O on an Atmel AVR Microcontroller

AT90S2313 has 15 I/O pins grouped in two ports (PB and PD) with 8 and 7 pins respectively. AT90S8515 has four ports (PA, PB, PC and PD) with 8 pins each. Every pin can be chosen as either input or output. The voltage level on an output pin can be controlled from the microcontroller to either 0 V (logically low or 0) or 5 V (logically high or 1). The microcontroller can also read the voltage level put on an input port and interpret it as either high (above 2.5 V) or low (under 1.5 V) and use this data in the program.

Some ports also have special functions. Some of them are used for example when programming the chip, a few can be used to handle interrupts³ and some are used to handle serial communication. There are also a lot of other special functions that are not used in this project and therefore are not described here.

The most common way to connect a LED to a microcontroller is to connect the anode of the LED to a positive voltage and the cathode in series with a current limiting resistance to an output pin on the microcontroller. The microcontroller can now switch on the LED by sending a low signal to the pin and switch it off by sending a high signal to it. The inverted logic is because the microcontroller can sink⁴ more current than it can produce. The LED's are thus made to light stronger without damaging the microcontroller.

1.4 Overview of the project

This part of the report describes, step by step, the construction of a rotating light emitting diode (LED) display, consisting of only seven LED's. The principle is that the LED's are placed on a rotating vertical arm, and are switched on and off at exactly the right times to convince the eye that you are looking at a seven by 35 dot matrix of LED's forming a text. The LED's are switched on and off by a microcontroller that is rotating along with the LED's.

An extension to the project was to send infrared (IR) signals to the rotating microcontroller to select what text is to be displayed. The reason for using IR signals is that there is no possibility to send the data by a cable, since the construction is rotating.

2 Functional Description

2.1 The Rotating Display

Since the LED's are rotating there was an obvious problem getting power to them. This was solved by placing the microcontroller and a battery on a small circuit board and letting it all rotate along with the LED's.

In the first step everything that was needed for the operation was programmed into the microcontroller, including the text to be displayed. The microcontroller could now switch on and off the LED's, using inverted logic (see above). In order to make the picture still, the LED's had to be switched on and off at exactly the same

²A link to a useful and easy built programmer can be found in the Reference section.

³Interrupts are automatically generated signals to the program, telling it that a certain task has finished, in this case a signal on the pin. The execution will then continue at a different place in the program.

⁴To sink current is to connect the terminal to ground and let the current go into the microcontroller.

positions (or time instances) every turn of the motor. In order to achieve this, the program had to know the speed of the motor, or alternatively get a signal in the same position in every revolution. Since the exact speed of a motor is very hard to measure, and may vary over time, the second approach was chosen. By using an optic sensor, also attached to the rotating circuit board, which gives a signal every time it passes a mark on the stationary part of the motor, the microcontroller could now know exactly when each revolution started. It could then switch on the LED's that should be lit in the first column of the pattern to be displayed. Then it would wait for a short time and switch off the LED's. After another short time it could switch on the LED's forming the second column and then it would repeat this for all 35 columns. By following this pattern on every revolution and always starting exactly when the sensor gave its signal, it was possible to get a steady picture that looked as if it was formed by a 7 x 35 dot matrix of LED's.

2.2 The IR Transmission

In the next step, the device was extended with the possibility of receiving data by infrared transmission from another device, and display this data on the screen. The IR-receiver had to be mounted exactly in the center of rotation to be capable of receiving data while rotating. The data transmission uses a special protocol, described in the programming section later.

2.3 Selection of Components

The following components were used in the project:

- A fast and silent motor. An old computer fan was used in this project, but almost any motor would work.
- A circuit board. Preferably one with holes and parallel copper stripes that can be cut where no connection is wanted.
- An arm to put the LED's on. Here a piece of a coat hanger was used.
- Bolts, nuts and washers for the mounting.
- A microcontroller. Here a AT90S2313 was used since it is small and cheap and has all the functionality needed.
- A clock crystal and two capacitors for the microcontroller to work. 4 MHz, 27 pF and 151 pF were chosen.
- A sensor that gives a signal at a certain position for every revolution. Here a sensor from an old floppy drive was used.
- Seven LED's with current limiting resistors, for example 150 Ω .
- A photodiode with a current limiting resistor.
- A 9 V battery.
- A 5 V voltage regulator (78L05).
- A stationary microcontroller with RS-232 connection. Here an Atmel STK200 programmer with an AT90S8515 attached was used. (This was because it was available in the laboratory, but any Atmel AVR microcontroller together with a RS-232 driving circuit could be used.)
- An IR diode with a current limiting resistor.
- A stationary PC to send the data to the display.

2.4 Assembly of the Display

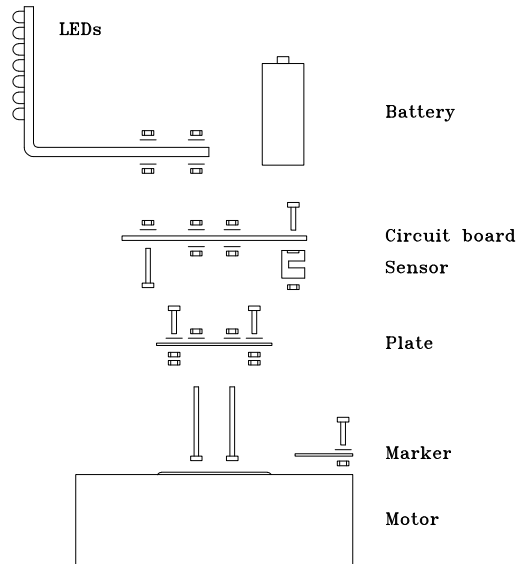


Figure 1: A diagram of the assembly.

This section describes how the device was built. Figure 1 shows a sketch over how the parts were assembled. First a small plate was attached to the fan and three bolts were attached to the plate. On these the circuit board, the battery and the arm with the LED's could be fastened. The sensor was screwed under the circuit board and a small piece of plastic was attached to the stationary foot of the fan, so the sensor would pass it upon every revolution.

The battery was fastened on two of the bolts and connected between input and ground on the voltage regulator. A circuit diagram over the electrical connections can be seen in figure 2. Output and ground from the voltage regulator were connected to the microcontroller. The LED's were mounted on the arm and connected with common anodes to V_{CC} and the cathodes via the current limiting resistors to outputs $PB1$ to $PB7$ on the microcontroller. The crystal was connected between pins $XTAL1$ and $XTAL2$ with capacitors from each pin to ground.

The sensor consists of an IR-diode and a photo transistor with a little gap in between. The common anode was connected to V_{CC} and the cathodes were connected to ground through suitable current limiting resistors (330Ω and $5.1k\Omega$). On the emitter of the photo transistor there would then be a zero when there is something in the gap (the small piece of plastic on the fan) and a one otherwise. This signal was connected to input $INT0$ on the microcontroller, which is capable of generating interrupts.

The photo diode was connected, reverse biased, between V_{CC} and ground through a current limiting resistor ($1\text{ M}\Omega$). There would then be a zero on the cathode when the diode is illuminated (with IR-light of the right frequency) and a one otherwise. This signal was connected to input $PD5$. The photo diode was mounted exactly in the center of rotation to be able to receive a signal even when the motor is rotating.

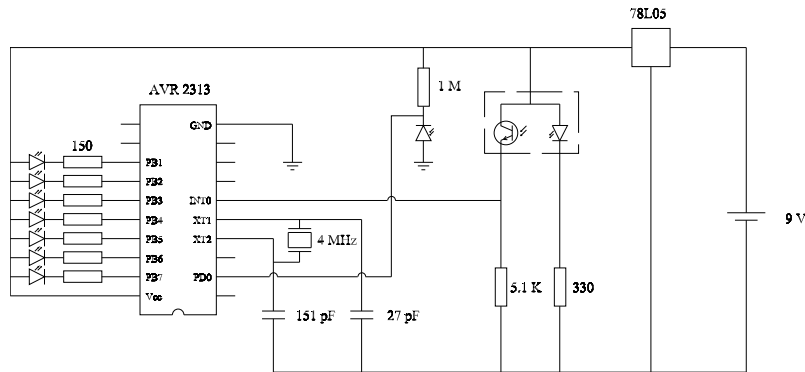


Figure 2: Circuit diagram of the rotating part.

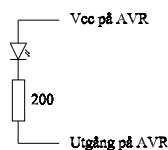


Figure 3: Circuit diagram of the transmitter.

Finally, the small transmitter circuit was built as can be seen in figure 3. The IR-diode was mounted on a support of old Meccano exactly above the photo diode with its anode connected to V_{CC} and its cathode via a current limiting resistor (220Ω) to output $PA0$, both on the stationary microcontroller (AT90S8515). The resulting device can be seen in figure 4, and was now ready for programming and testing.

3 Programming the Rotating Display

This section describes, step by step, how the device was tested and how the programs work, both in the rotating microcontroller (`hejsan.c` and `receiver.c`), the stationary microcontroller (`sender.c`) and in the stationary PC (`pcsender.cc`).

For details on how the actual programming of the microcontroller (getting the code into the chip) is performed I refer to Atmel's homepage and the homepage for the home made programmer⁵.

3.1 Program 1: `hejsan.c` (AT90S2313)

The first program was built up step by step, to check that everything worked as expected. First zeros were put on outputs $PB1$ to $PB7$, which, as expected, caused all LED's to be lit. While rotating this was seen as seven horizontal, red lines.

Next the sensor (on $INT0$) was programmed to set the interrupt flag on each revolution. The main program was set to loop until the flag was set and then light all LED's. After a short while (about 100 ns) all LED's were turned off. This made the LED's light for a short while on every revolution and at exactly the same place each time, which as expected, showed as a stable vertical bar.

The program was expanded to repeat the switching of the LED's 35 times. This was displayed as a grid of 7×35 stable red dots.

⁵Links to these pages can be found in the Reference section

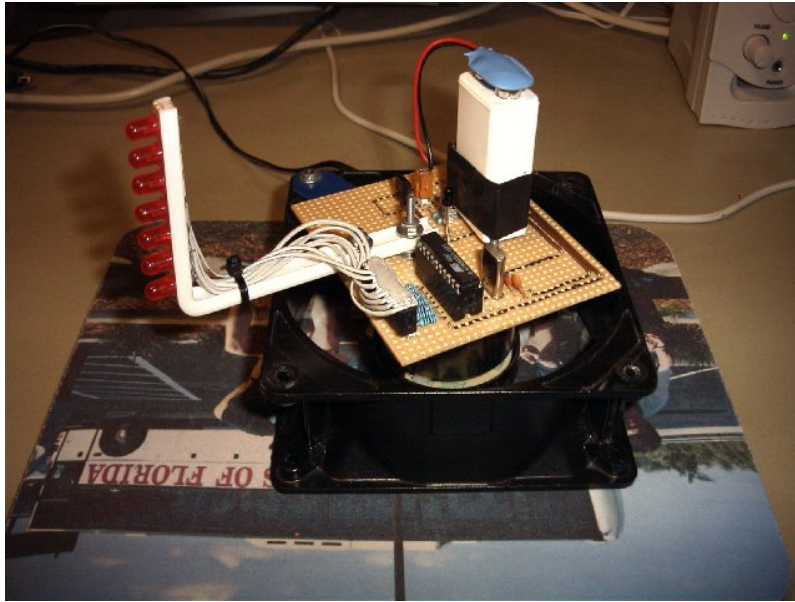


Figure 4: A photo of the assembled display.

Finally the program was set to select which LED's should be lit, from an array in the program and our device displayed the text "HEJSAN" (Swedish for hello).

3.2 Program 2: receiver.c (AT90S2313)

The second program was extended to receive data from a stationary microcontroller, through the photo diode, and then display this text on the LED's. Timer 1 on the AT90S2313 was set to generate interrupts every second ms and then read the value on input *PD0*, which is connected to the photo diode. This correspond to a transfer speed of 500 bits per second (BPS) which means the whole picture can (theoretically) be changed in about half a second.

A simple protocol of our own was developed for the data transfer, where a start sequence of eight ones indicates that some data is being sent. This sequence is unique since only seven bits are used for the LED's and the last bit is always a zero (except in the start sequence). The whole data block (36 bytes, one for each column plus the start sequence) must also end with a zero for the start sequence to be unique.

Upon every interrupt, the program checks if the whole start sequence has been received. If not, it checks if the received bit (*pulse*) is a one, and if so a counter (*ones*) is increased. If the bit is a zero it would mean that the start sequence has not started, and the counter is reset. If the whole start sequence has been received (*ones == 8*) the bit is put into the right position in the bitmap that is to be shown (*image*). This is achieved by putting something in if the bit is a one, (by bit wise OR and left shift) and doing nothing if it is a zero. Therefore each column must be set to zero before a new byte is received.

Viewing of the text is performed as in the first program, using the delay function. A drawback to this method is that the delay will be slightly longer when an interrupt occurs, which can make the image somewhat unsteady. This was compensated for by measuring the time of the shortest and the longest interrupts and adding an extra delay in the shortest, to make them all take the same time and give a more stable picture.

The program initially displays the text “HEJSAN” until a new text is received over the IR link. The latest received text will then be displayed, even if the sender stops sending more information.

3.3 Program 3: sender.c (AT90S8515)

The sender receives information from a stationary PC through the serial port using the usual RS-232 protocol and repeats it to the rotating display using IR communication.

When the reception occurs, the interrupts are disabled and the standard RS-232 protocol for two-way communication is used to receive the data. A transfer rate has of 19200 BPS was chosen, which is the highest possible with a 4 MHz crystal, without getting too big a percentual error in the speed. Each byte is received and stored in the bitmap (`image`) and then returned to the PC for verification.

After the whole data sequence has been received, the interrupts are activated again and timer 1 starts generating interrupts every second millisecond. At each interrupt the program checks if the whole start sequence has been sent and if not, a one is sent on output `PA0`, which is connected to the IR-diode. When the whole start sequence is transmitted the values of the bits in `image` are sent instead. With Interrupts activated, the program waits for the PC to send a next-sign (`'n'`) and at the same time the data in `image` is repeatedly sent to the rotating display. This continues until the next-sign is received. Then the interrupts are disabled and the next data sequence is received from the PC.

3.4 Program 4: pcsender.cc (MS DOS)

The program in the stationary PC lets the user input a text by the keyboard and convert this text to the bit pattern that is to be shown on the display. This data is sent to the sender (the stationary microcontroller) through the serial port. The data is then returned from the sender and displayed on the screen for verification.

Before each new data sequence (except the first) is sent, a next-sign (`'n'`) is sent, to instruct the sender to turn of its interrupts and start receiving the next sequence.

4 The Resulting Device

The resulting device is capable of receiving data from any common PC capable of sending data over its serial port using the standard RS-232 protocol, and the data will almost immediately show up on the rotating display.

The reason why the conversion from text to bitmap picture is put in the PC program even though it demands more data to be transmitted, is that it makes it possible to create new characters, without altering the code in the microcontrollers. It also makes it possible to view any type of graphics that would fit in a 7 x 35 pixel array.

With only a handful of additional components, the somewhat clumsy solution of the stationary AT90S8515 microcontroller on the development board could be avoided and a smaller, cheaper microcontroller like the AT90S2313 could be used as a sender, together with a small RS-232 driver circuit and this could be mounted on the foot of the construction.

In a dim light the rotating display will give quite a spectacular view, since the rotating parts will not be visible and it will give an illusion of the text floating in middle air, as can be seen in figure 5.



Figure 5: A photo of the resulting display in action.

References

1. Robin Stridh, Moses Kullberg: *Roterande LED-display, Projektrapport i Mikrodatorkonstruktion, 2000.*
2. Datasheets of the microcontrollers: www.atmel.com/atmel/products/prod200.htm
3. CodeVision AVR Compiler: infotech.ir.ro
4. SP12 Home made AVR programmer: www.xs4all.nl/~sbolt/e-spider_prog.html
5. Homepage of the project including photos: home.student.uu.se/rost7041

A C-code

Here follows listings of all programs used in the project. Unfortunately the comments are all in swedish, but maybe this will be changed in a future version.

The compilers used are CodeVisionAVR C Compiler v1.2 for the AVR programs (`hejsan.c`, `sender.c` and `receiver.c`) and Borland Turbo C++ v1.01 for the PC program (`pcsender.cc`). (The reason for using such an old compiler as Turbo C++ is that it gives direct access to the port I/O functions under Windows NT, which otherwise demands the implementation of a device driver. Probably any compiler that supports the WIN16 platform would do.)

A.1 Program 1: `hejsan.c` (AT90S2313)

```

/*****
Project : Roterande LED-display
Version : 01
Date   : 12.12.2000
Author : Robin Stridh, Moses Kullberg
Comments: Skriver ut texten "HEJSAN".

Chip type      : AT90S2313
Clock frequency : 4,000000 MHz
*****/
10

#include < 90s2313.h>
#include < delay. h>

#define MAX_INDEX 36 // Antal kolumner som visas.
char col;
// Bitmönstret som ska visas på displayen.
flash const unsigned char image[ MAX_INDEX ] = { 254, 16 , 16 , 16 , 254, 0,
                                                254, 146, 146, 146, 130, 0,
                                                64 , 128, 130, 126, 2 , 0,           20
                                                140, 146, 146, 146, 98 , 0,
                                                252, 18 , 18 , 18 , 252, 0,
                                                254, 8 , 16 , 32 , 254, 0};

void main( void)
{
// Port B är kopplad till lysdioderna.
DDRB = 0xff; // Alla pinnar i port B sätts till utgångar.
PORTB = 0xff; // Släcker alla dioder.

// En flagga i GIFR sätts varje gång sensorn passerar ett märke.
MCUCR = 0x02; // Flaggan sätts på negativ flank.
GIFR = 0x40; // Nollställer flaggan.
30

// Huvudloopen
while( 1){
while(( GIFR & 0x40) == 0) // Så länge som ingen nolla på INTO...
; // ..händer ingenting.
for( col = MAX_INDEX - 1; col >= 0; col--){ // Motorn roterar medsols.
PORTB = ~ image[ col]; // Skickar ut en nolla för varje tänd diod.
delay_us( 200); // Låter dioden lysa en viss tid.
PORTB = 0xff; // Släcker alla dioder.
delay_us( 500); // Ger ett visst avstånd mellan kolumnerna.
40
}
GIFR = 0x40; // Redo att ta emot ny signal från sensorn.
};
}

```

A.2 Program 2: receiver.c (AT90S2313)

```
/******  
Project : Roterande LED-display  
Version : 01  
Date : 01.13.2001  
Author : Robin Stridh, Moses Kullberg  
Comments: Tar emot information från fotodiod för att sedan  
          skriva ut på "skärmen".  
  
Chip type      : AT90S2313  
Clock frequency : 4,000000 MHz  
*****/ 10  
  
#include < 90s2313.h>  
#include < delay. h>  
  
#define MAX_INDEX 36 // Antal kolumner som visas.  
//Skriver ut starttext "HEJSAN" om ingen text skickas  
unsigned char image[] = { 254, 16, 16, 16, 16, 254, 0,  
                          254, 146, 146, 146, 146, 130, 0,  
                          64, 128, 130, 126, 2, 0, 20  
                          140, 146, 146, 146, 98, 0,  
                          252, 18, 18, 18, 252, 0,  
                          254, 8, 16, 32, 254, 0};  
  
void main( void){  
  // Lokala variabler.  
  char col;  
  // Port initieringar.  
  // Port B är kopplad till lysdioderna.  
  DDRB = 0xff; // Alla pinnar i port B sätts till utgångar. 30  
  PORTB = 0xff; // Släcker alla dioder.  
  // Port D  
  DDRD = 0x00; // Alla pinnar på port D ingångar.  
  PORTD = 0x00; // Aktiverar ej pull-up.  
  
  // En flagga i GIFR sätts varje gång sensorn passerar ett märke.  
  MCUCR = 0x02; // Flaggan sätts på negativ flank.  
  GIFR = 0x40; // Nollställer flaggan.  
  
  // Timer 0 skalas ned 64 ggr. 40  
  TCCR0 = 0x03; // 00000011  
  
  // Timer 1 genererar avbrott varannan millisekund.  
  TCCR1A = 0x00;  
  TCCR1B = 0x09; // "00001001" dela klockan med 1  
  OCR1 = 8000; // Antal pulser på 2 ms  
  TIMSK = 0x40; // "01000000"  
  #asm("sei"); // Aktiverar interrupt  
  
  // Huvudloop 50  
  while( 1){  
    while(( GIFR & 0x40) == 0) // Så länge som ingen nolla på INT0..  
      ; // ..händer ingenting.  
    for( col = MAX_INDEX - 1; col >= 0; col--){ // Motorn roterar medsols.  
      PORTB = ~ image[ col]; // Skickar ut en nolla för varje tänd diod.  
      delay_us( 100); // Låter dioden lysa en viss tid.  
      // alternativ delay  
      //TCNT0 = 0; // nollställ timer 0;  
      //while(TCNT0 < 7) // 7 * 16 = 112 us 60  
      //;  
      PORTB = 0xff; // Släcker alla dioder.  
      delay_us( 400); // Ger ett visst avstånd mellan kolumnerna.  
      // alternativ delay  
      //TCNT0 = 0;
```


A.3 Program 3: sender.c (AT90S8515)

```

/*****
Project : Roterande LED-display
Version : 01
Date : 01.13.2001
Author : Robin Stridh, Moses Kullberg
Comments: Tar emot information från PC via RS-232 för att sedan sända
vidare till den roterande displayen via IR-diod.

Chip type : AT90S8515
Clock frequency : 4,000000 MHz
*****/

#include < 90s8515.h>
#include < stdio. h>

// Globala variabler
#define MAX_INDEX 36
char image[ MAX_INDEX];

void main( void)
{
// Lokala variabler
char k; // Startbyte från PC.
char index; // Kolumnräknare.

// Port initieringar.
// PORT A
DDRA= 0xff; // Alla pinnar utgångar.
PORTA= 0x00; // Sätter "nollor" på alla utgångar.

// Timer 1 genererar avbrott varannan millisekund.
TCCR1A= 0x00;
TCCR1B= 0x09; // "00001001" dela klockan med 1
OCR1A= 8000; // Antal pulser på 2 ms
TIMSK= 0x40; // "01000000"

// UART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// UART Receiver: On
// UART Transmitter: On
UCR= 0x18;
// UART Baud rate: 19200
UBRR= 0x0C;

// Huvudloop.
while ( 1) {
for( index = 0; index < MAX_INDEX; index++) {
k = getchar(); // Läser in data från Pc.
image[ index] = k; // Läger data i image.
putchar( k); // Returnerar data till PC för verifiering.
}
asm("sei"); // Slår på avbrott.
while(( k = getchar()) != 'n')// lyssnar efter startbyte från PC.
;
asm("cli"); // Slår av avbrott.
putchar( k); // Returnerar startbyte till PC.
};
}

interrupt[ TIM1_COMPA] void t2ms( void) {
static char ones = 0; // Startbiträknare
static char index = 0; // Kolumnräknare.
static char bit_in_char = 0; // Biträknare.
char pulse; // Bit som skickas.

```

```

if ( ones < 8) { // Generering av startsekvens.
    pulse = 1;
    ones++;
}
else { // Läger in "rätt" bit i pulse. 70
    pulse = image[ index] & ( 1 << bit_in_char);
    bit_in_char++;
    if ( bit_in_char == 8) {
        bit_in_char = 0;
        index++;
        if ( index >= MAX_INDEX) {
            index = 0;
            ones = 0;
        }
    }
} // Skickar data. 80
PORTA. 0 = pulse;
}

```

A.4 Program 4: pcsender.cc (MS DOS)

```

/*****
Project : Roterande LED-display
Version : 01
Date   : 01.13.2001
Author : Robin Stridh, Moses Kullberg
Comments: Skickar information från tangentbordet på en PC via RS-232 till
stationär mikrodator för att sända vidare till den roterande
displayen via IR-diod.
*****/

#include < dos.  h>
#include < stdio. h>
#include < conio. h>
#include < iostream. h>

/*****
* Communication settings
*****/
#define PORT1 0x2F8
/* Serial Ports Base Address */
/* COM1 0x3F8 */
/* COM2 0x2F8 */
/* COM3 0x3E8 */
/* COM4 0x2E8 */

int main() {
    outportb( PORT1 + 1 , 0); /* Turn off interrupts - Port1 */
    outportb( PORT1 + 3 , 0x80); /* SET DLAB ON */
    outportb( PORT1 + 0 , 0x06); /* Set Baud rate - Divisor Latch Low Byte */
    /* Default 0x03 = 38,400 BPS */
    /* 0x01 = 115,200 BPS */
    /* 0x02 = 57,600 BPS */
    /* 0x06 = 19,200 BPS */
    /* 0x0C = 9,600 BPS */
    /* 0x18 = 4,800 BPS */
    /* 0x30 = 2,400 BPS */
    outportb( PORT1 + 1 , 0x00); /* Set Baud rate - Divisor Latch High Byte */
    outportb( PORT1 + 3 , 0x03); /* 8 Bits, No Parity, 1 Stop Bit */
    outportb( PORT1 + 2 , 0xC7); /* FIFO Control Register */
    outportb( PORT1 + 4 , 0x0B); /* Turn on DTR, RTS, and OUT2 */

/*****
*
*****/

const char MAX_INDEX = 36; // Antalet kolumner.
// Omvandlingstabell från bokstäver till bitmönster.
const unsigned char Alphabet[ ][ 5] = {{ 124, 162, 146, 138, 124}, // 0
    { 0 , 132, 254, 128, 0 }, // 1
    { 132, 194, 162, 146, 140}, // 2
    { 66 , 130, 138, 150, 98 }, // 3
    { 48 , 40 , 36 , 254, 32 }, // 4
    { 78 , 138, 138, 138, 114}, // 5
    { 120, 148, 146, 146, 96 }, // 6
    { 2 , 226, 18 , 10 , 6 }, // 7
    { 108, 146, 146, 146, 108}, // 8
    { 12 , 146, 146, 82 , 60 }, // 9
    { 0 , 0 , 0 , 0 , 0 }, // (:)
    { 0 , 0 , 0 , 0 , 0 }, // (;)
    { 0 , 0 , 0 , 0 , 0 }, // (<)
    { 0 , 0 , 0 , 0 , 0 }, // (=)
    { 0 , 0 , 0 , 0 , 0 }, // (>)
    { 0 , 0 , 0 , 0 , 0 }, // (?)
    { 0 , 0 , 0 , 0 , 0 }, // (
    { 252, 18 , 18 , 18 , 252}, // A

```

```

{ 254, 146, 146, 146, 108}, // B
{ 124, 130, 130, 130, 68 }, // C
{ 254, 130, 130, 130, 124}, // D
{ 254, 146, 146, 146, 130}, // E
{ 254, 18 , 18 , 18 , 2 }, // F      70
{ 124, 130, 146, 146, 244}, // G
{ 254, 16 , 16 , 16 , 254}, // H
{ 0 , 130, 254, 130, 0 }, // I
{ 64 , 128, 130, 126, 2 }, // J
{ 254, 16 , 40 , 68 , 130}, // K
{ 254, 128, 128, 128, 128}, // L
{ 254, 4 , 24 , 4 , 254}, // M
{ 254, 8 , 16 , 32 , 254}, // N
{ 124, 130, 130, 130, 124}, // O
{ 254, 18 , 18 , 18 , 12 }, // P      80
{ 124, 130, 162, 66 , 188}, // Q
{ 254, 18 , 50 , 82 , 140}, // R
{ 140, 146, 146, 146, 114}, // S
{ 2 , 2 , 254, 2 , 2 }, // T
{ 126, 128, 128, 128, 126}, // U
{ 62 , 64 , 128, 64 , 62 }, // V
{ 126, 128, 112, 128, 126}, // W
{ 198, 40 , 16 , 40 , 198}, // X
{ 14 , 16 , 224, 16 , 14 }, // Y
{ 194, 162, 146, 138, 134}, // Z      90
{ 240, 76 , 74 , 76 , 240}, // Å (I)
{ 240, 74 , 72 , 74 , 240}, // Ä (Λ)
{ 112, 138, 136, 138, 112}, // Ö (J)
{ 0 , 0 , 0 , 0 , 0 }, // (˘)
{ 0 , 0 , 0 , 0 , 0 }, // (˘)
{ 0 , 0 , 0 , 0 , 0 }, // (˘)
{ 64 , 168, 168, 168, 240}, // a
{ 254, 144, 136, 136, 112}, // b
{ 112, 136, 136, 136, 64 }, // c
{ 112, 136, 136, 144, 254}, // d      100
{ 112, 168, 168, 168, 48 }, // e
{ 16 , 252, 18 , 2 , 4 }, // f
{ 24 , 164, 164, 164, 124}, // g
{ 254, 16 , 8 , 8 , 240}, // h
{ 0 , 136, 250, 128, 0 }, // i
{ 64 , 128, 136, 122, 0 }, // j
{ 0 , 254, 32 , 80 , 136}, // k
{ 0 , 130, 254, 128, 0 }, // l
{ 248, 8 , 48 , 8 , 240}, // m
{ 248, 16 , 8 , 8 , 240}, // n      110
{ 112, 136, 136, 136, 112}, // o
{ 248, 40 , 40 , 40 , 16 }, // p
{ 16 , 40 , 40 , 48 , 248}, // q
{ 248, 16 , 8 , 8 , 16 }, // r
{ 144, 168, 168, 168, 64 }, // s
{ 8 , 126, 136, 128, 64 }, // t
{ 120, 128, 128, 64 , 248}, // u
{ 56 , 64 , 128, 64 , 56 }, // v
{ 120, 128, 96 , 128, 120}, // w
{ 136, 80 , 32 , 80 , 136}, // x      120
{ 24 , 160, 160, 160, 120}, // y
{ 136, 200, 168, 152, 136}, // z
{ 64 , 172, 170, 172, 240}, // å (f)
{ 64 , 170, 168, 170, 240}, // ä (l)
{ 96 , 146, 144, 146, 96 }; // ö (j)

const unsigned char Space = 0; // Bitmönsternellanrumbyte.
int first = 1; // Startteckenflagga.
char ch; // Bokstavsräknare för den inlästa texten.
char text[ 20]; // Inläst textsträng.
unsigned char image[ MAX_INDEX]; // Bitmönstret innan det skickas.
unsigned char back[ MAX_INDEX]; // Det returnade bitmönstret.

```

