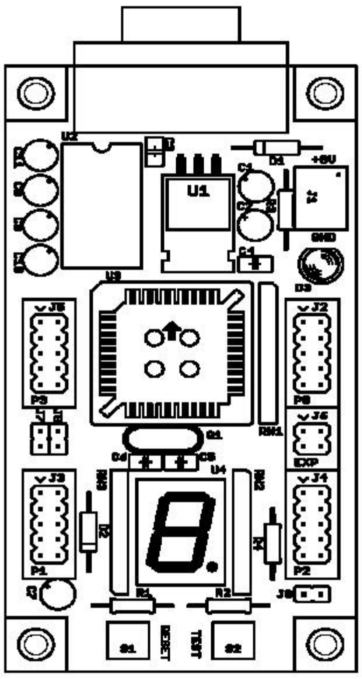
Introduction to microcontrollers and their practical uses.



Embedded Computer Architecture ET4165.
Lab Assignments.

11 November 2007

Harry Broeders (1376934)

Master: Embedded Systems

Introduction.

As part of the Embedded Computer Architecture course ET4165 students need to familiarise themselves to microprocessors and their practical uses. Several aspects of microcontrollers are introduced: I/O ports, serial and parallel communication, timers, interrupts, and interfacing to the outside world by means of several practical assignments. The programming language C is used to program the microcontroller. The microcontroller used in these assignments is the P89V51RD2 [1] an 8-bits microcontroller from Philips based on the well known Intel 8051 microcontroller. Personally I already was familiar with microprocessors and their practical uses, but I didn't use an 8051 derivate before. Therefore it was interesting for me to compare the features of the P89V51RD2 with the features of the two 8-bits microcontrollers I was already familiar with: 68HC11 [2] from Freescale and AVR [3] ATmega16 from Atmel.

Assignment 1.1: A simple square wave signal generator.

Programming a microcontroller in C has the advantage that the programmer doesn't need to learn the specific assembler language used by this microcontroller. In the first assignment an 1 KHz square wave needs to be generated on pin P1.6. The I/O pins of the P89V51RD2 can be programmed by means of the so called special function registers. The Keil C compiler which is used in the lab has a special datatype `sfr` which can be used to access these special function registers. Some special function registers are bit-addressable. This means that there are bit addresses provided which can be used to access a single bit by using special machine code instructions. In this assignment we need to access pin 6 of port 1. This pin can be accessed through bit 6 of the special function register P1 (byte address = 0x90) but also through the bit address 0x96. The Keil C compiler has defined the special datatype `sbit` which makes it possible to use these bit-addressable special function bits in a C program. This `sbit` type provides access to a bit-addressable special function register bit as follows:

```
sbit signal = 0x96;
```

This variable `signal` can now be used to change the value of pin 6 of port 1. For example:

```
signal = 1; /* pin 6 of port 1 becomes high */
```

The AVR and 68HC11 microcontrollers both don't have bit-addressable memory. The programmer needs to use bitwise operators to change a single bit in a special function register. For example: to set pin 6 of PORTA of the AVR microcontroller the programmer needs to use a bitwise or instruction:

```
PORTA = PORTA | 0x40; /* pin 6 of port A becomes high */
```

This bitwise or is necessary because the programmer doesn't want to change the other pins of PORTA.

The bit-addressable special function registers of the P89V51RD2 are easier to use than the byte addressable special function registers of the AVR and 68HC11. The disadvantage is that a C language extension (the `sbit` type) is needed to use these bit addresses in a C program. The C compilers for the AVR and 68HC11 both don't need language extensions.

To generate an 1 kHz square wave signal we need to invert the pin every 500 ms. In the first assignment this timing is realised by writing a simple, empty for loop to wait for 500 ms. The value 124 is calculated by inspecting the generated assembler code for the loop.

```
for (i=0; i<124; ++i);
```

This code can be optimised away by an optimising compiler because the code doesn't perform any real action. The gcc compiler for example will remove this code when the –O2 command line option is used. To prevent this the variable `i` is declared as follows:

```
volatile unsigned char i;
```

The `volatile` type qualifier tells the compiler that the variable `i` can change outside the control of the compiler. Therefore any read or write operation to or from this variable has to really access the variable in memory. The compiler is not allowed to cache this variable in a register or to optimise it away. The variable `i` is defined as `unsigned char` instead of `int` because we want to use an 8 bit variable which is most efficient on an 8 bits microcontroller. If we make the compare and increment inside the for statement as efficient as possible then we can make the delay loop as accurate as possible. The C99 standard defines fixed sized integers in the include file <stdint.h> [5]. For example the 8 bits unsigned integer type `uint8_t` would be more appropriate to define the variable `i` instead of the `unsigned char` type because it better shows the intended use of this variable. The C99 standard is not supported by the Cx51 Keil compiler provided in the lab.

Because we used the generated assembler code to calculate the delay time of the delay loop, this delay time is not only dependent on the clock frequency of the microcontroller but also on the specific version of the compiler used or even on the specific compiler directive used. The C compiler used in the lab has an OPTIMIZE(n) directive which can be passed via the command line to specify the level of optimisation performed by the compiler. This can change the code generated for the loop and influence the timing. Therefore it is better to use a hardware timer to create a delay period (see assignment 3). A delay generated by a hardware timer will still be dependent of the clock frequency but will be independent of the compiler version and directives.

Assignment 1.2: A maximum value search program.

In this assignment the 7 segments display which is connected to port P1 of the microcontroller is used. All 8 pins of this port can be accessed at once using a variable of the special type `sfr`:

```
sfr seg7=0x90;
```

The digit 3 can be displayed on the 7 segments display as follows:

```
seg7 = 0xb0;
```

An array filled with 8 bits numbers is used as data input for this assignment. This array is defined as follows:

```
unsigned char xdata table[]={1,2,3,4,5,6,7,8,9,0};
```

The `xdata` type qualifier is a Cx51 language extension. This language extension is needed because the P89V51RD2 has a harvard architecture. This means that the code and data have separate memory spaces. The C language assumes one flat memory space for data and code. The `data` type qualifier can be used to identify the 128 byte internal RAM provided by the original 8051. The `xdata` type qualifier is used to address RAM in the 64K data address space. The name xdata which stands for external data is misleading in this case because the P89V51RD2 has 1Kbyte internal RAM. If we want to place the array outside of the 128 bit direct addressable RAM we need to use the `xdata` qualifier but the data will still be stored inside the microcontroller. The `xdata` qualified data must be accessed using the special assembler instruction `MOVX` therefore data qualified with `xdata` can not be accessed as fast as data qualified with `data`. The AVR microcontroller also has a harvard architecture but the 68HC11 uses a Von Neumann architecture with one unified memory map for code and data. A Harvard architecture is more efficient because code and data can be accessed in parallel in this architecture.

In the assignment a binary value should be displayed on the 7 segments display. If the data only consists of positive numbers smaller than 10 a lookup table can be used to display the value in a readable way. The following display function implements this:

```
void write7seg(unsigned char value) {
    static unsigned char codes[] = {
        0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x98 };
    assert(value >= 0 && value <= 9);
    seg7 = codes[value];
}
```

Assignment 2.1: A triangle wave form generator.

In this assignment port P0 is connected to a digital to analog converter (ADC). The output of the ADC can vary between -5V and +5V and is connected to an oscilloscope which displays a graphical representations of the waveform. If the value of port P0 is incremented from a low value to a high value and decremented again from the high to the low value one period of a triangle wave form is generated. To generate a triangle waveform between -4V and +3V the low value should be 25 and the high value should be 205. The maximum frequency of the signal is about 275 Hz. This is exactly as expected from inspecting the assembler code which is generated by the compiler. This frequency can be decreased by inserting a delay before incrementing or decrementing the value on port P0.

Assignment 2.2: An ellipse generator program.

When the linearly incremented and decremented value, called x, from assignment 2.1 is used to lookup the sine of the value x·2·p / 256 and this looked up value is written to the ADC then a sine wave is generated. If we generate 2 sine waves, by using 2 ports connected to 2 separate ADC’s, one signal can be used to control the X-channel of the oscilloscope and the other signal can be used to control the Y-channel of the oscilloscope. If the phase between the 2 signals is 45 degrees an ellipse is shown on the oscilloscope. When the phase between the 2 signals is made adjustable by 8 switches which are connected to port P1 the length of one diagonal of the ellipse can be adjusted. The 2 extreme figures are a line (phase is 0, 180, or 360 degrees) and a circle (phase is 90 or 270 degrees).

If the frequency of one of the signals is made N times higher than the frequency of the other signal even more interesting patterns can be generated as can be seen in figure 1. These patterns are called Lissajous patterns.

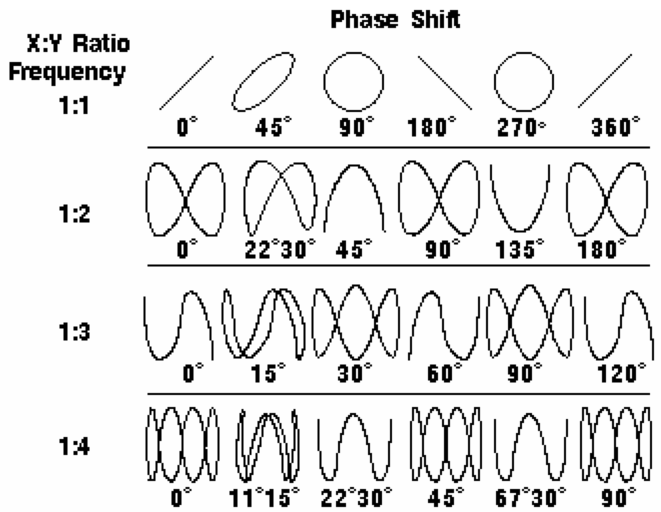


Figure 1. Lissajous patterns.

The amount of memory is restricted in most microcontroller applications. Therefore it is good practice to look for ways to minimalise the amount of memory needed by a program. In this assignment a lookup table with all sine values for a full period of a sine in 256 steps is used. Using the symmetrical properties of the sine function we can simple reduce the memory for this table with a factor of 4 using the following trigonometric formulas: sin(a) = sin(p-a) and sin(-a) = -sin(a). Only the first quarter of the sine period has to be saved in memory. If for example the first quarter of the sine period (from x = 0 up to x = 0.5·p is saved in a table called lookup containing 64 steps then y = sin(x·2·p / 256) can be found as follows:

• y = lookup[x]	for 0 <= x < 64.
• y = lookup[127 - x]	for 64 <= x < 128.
• y = - lookup[x-128]	for 128 <= x < 192.
• y = - lookup[255-x]	for 192 <= x < 256.

Assignments 3: Timers.

As already mentioned in assignment 1.1 a delay loop can be made independent of the compiler version and directives by using a hardware timer. The P89V51RD2 has several hardware timers build in. These timers can be used in several modes. The mode used in this assignment is called auto reload mode (mode 2). This mode configures the timer special function register TL0 as an 8-bit counter with automatic reload. Overflow from TL0 sets bit TF0, and also reloads TL0 with the contents of special function register TH0. The reload leaves TH0 unchanged. The timers of the P89V51RD2 are very similar to the timers of the AVR microcontroller. This microcontroller also has an auto reload mode. The 68HC11 has a more simple timer system. The 68HC11 timers need to be reloaded by software every time an overflow occurs.

Using the auto reload mode it is very easy to generate events with a constant timing interval. The timer is first initialised in such a way that it overflows after the desired interval. The program now just waits until bit TL0 is set, generates one event, resets TL0 and waits until TL0 is set again.

Assignments 4: Timers and interrupts.

In assignment 3 we polled the TF0 bit in software to find out when the time period elapsed. Alternatively an interrupt can be used. When an interrupt is used the program which is running is suspended when the TF0 bit is set. A special function called interrupt handler is executed. After performing the interrupt handler the processor continues whatever it was doing when it was suspended. The TF0 bit is cleared automatically when the interrupt handler is called.

In this assignment we need an interval timer with an interval of 50000 µs. This interval can not be generated by an 8 bit counter. Therefore we use timer T0 in mode 1. In this mode the two 8 bits special function registers TH0 and TL0 are cascaded and function as one 16 bits counter. Overflow from TL0 increments TH0 and overflow from TH0 sets bit TF0. In this mode there is no auto reload so the TL0 and TH0 registers must be reloaded by the software in the interrupt handler.

The interrupts of the P89V51RD2 are very similar to the interrupts of the AVR microcontroller. This microcontroller also automatically clears the flag which caused the interrupt when the interrupt handler is called. The 68HC11 has a more simple interrupt system. When using the 68HC11 the flag which causes the interrupt must be reset by software in the interrupt handler.

Assignments 5: Serial I/O.

The P89V51RD2 has an on chip Universal Asynchronous Receiver Transmitter (UART). This device can be used to generate an asynchronous serial signal. Timer T1 is used in 8 bit auto reload mode to generate the baud rate clock for the UART. When TH1 is loaded with 0x100-13 the generated baud rate is 2400 Bd. Higher baud rates can not be used when timer T1 is used as baud rate generator because the derivation from the intended baud rate becomes to high. Because the serial communication is asynchronous the PC uses its own clock. When the baud rate expected by the PC derivates to much from the baud rate of the signal send by the microcontroller the PC will receive incorrect data. Timer T2 of the P89V51RD2 has a special mode in which it can be used as baud rate generator. When timer T2 is used as baud rate generator higher baud rates can be used e.g. 19.2 kBd.

The UART of the P89V51RD2 is quite similar to the UART found in the AVR and 68HC11 microcontrollers.

In this assignment the program keeps track of the time by updating global variables `tenths`, `seconds`, `minutes`, and, `hours`. These variables are updated in the timer T0 interrupt handler. The time is send to the PC via the serial connection when the PC sends the character '*' to the microcontroller via the serial connection. This can cause the following problem: Assume that the time is 00:15:59:00 and the * character is received. Just after sending 00:15 the variable `seconds` rolls over from 59 to 00. The send time will be 00:15:00:00 instead of 00:16:00:00. So the time send to the PC is wrong.

This problem can be prevented as follows. When the * character is received the TF0 interrupt is temporally disabled by writing 0 to the ET0 (Timer 0 Overflow Interrupt Enable) bit. The global time variables are copied to local time variables. After that the TF0 interrupt is enabled again by writing 1 to the ET0 bit. Then the local time variables are send to the PC. This prevents that the variables are changed in the interrupt handler when the time is send to the PC. If an TF0 interrupt occurs when the interrupt is disable it is temporally hold until the interrupt is enabled again.

Of course, when the timer T0 overflows twice when the TF0 interrupt is disabled one interrupt is lost and the time lags behind. Therefore the interrupt should only be disabled as short as possible. In this case only four 8 bits variables have to be copied when the interrupt is disabled.

Conclusion.

It was interested to see that the P89V51RD2 which is an Intel 8051 derivate is quite similar to other 8 bits microcontrollers I encountered before: the Atmel AVR ATmega16 and the 68HC11. Especially the UART, timers, and interrupt systems are very similar. The main difference is the bit-addressable special function registers, a neat feature of the P89V51RD2 that is not present in the AVR nor in the 68HC11.

Literature.

[1] Information about the microcontroller we used in the lab can be found here: http://www.standardics.nxp.com/products/80c51/p89v51rx2/

[2] Information about 8-bits microcontrollers from Freescale can be found here: http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=0162468449

[3] Information about the AVR 8-bits microcontrollers from Atmel can be found here: http://www.atmel.com/products/AVR/overview.asp

[4] Keil Cx51 User's Guide: http://www.keil.com/support/man/docs/c51/c51_intro.htm

[5] Information about the fixed sized integer types defined in the C99 standard ISO/IEC 9899 can be found in chapter 7.18 of: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf