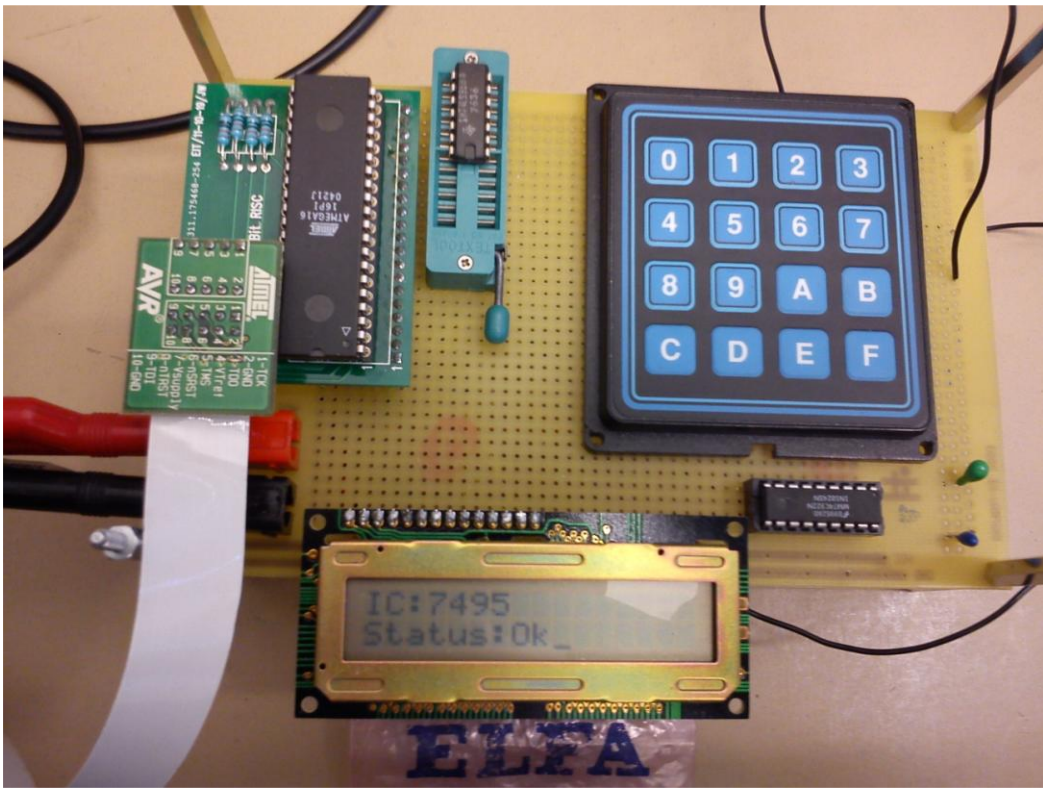
EITF40

# IC tester

## Report

**David Winér, Mattias Olsson**

**3/5/2013**



# Contents

# Introduction

We wanted to build a working IC tester designed to test some commonly used IC in the 74…. Series since we thought it would be interesting to know how to build one, even though today it is probably cheaper to buy a new circuit if you know which IC it is that does not work, and along the way gain some insight in how the ATMEGA16 processor works

# Specification:

Test the logic functions of some 14 and 16 pin logic ICs from 7400 series [3]

- 2, 3, 4-input- AND, NAND, OR, NOR etc. etc.
- SR, D- Flip-flop
- Binary -> BCD, BCD->binary
- Counter, modulo
- 4bit - full adder
- Parity checker/generator
- Shift register
- Mux/ Demux

Modes:

- Read input from keypad and test specific IC.
- Test unknown IC with search mode.

# Realization

## Hardware

### Processor

The Atmega16 processor was chosen because it is a lot simpler to use if there's a need to change the ports between output and input and the software could easily fit in 16kbytes of program memory.

The ports were used as follows:
Port A: handles the I/O of the left side of the socket.
Port B: handles the I/O of the right side of the socket.
Port C: 4 pins are used to program the processor, the other 4 pins are used as keypad input through a keypad decoder IC.
Port D: The pins of Port D are used to print out on the display.

The processor is set to run at 1 MHz because there is no need to run the processor faster, since the tests are not that heavy in computation power and the circuits wouldn't work any faster anyway.

### Keypad and decoder

We used a keypad with 16 keys and a decoder to reduce the number of IO pins needed on the Atmega16.

### LCD-Display

We use an alpha-numeric display with 2 rows and 16 characters per row.

# Schematic

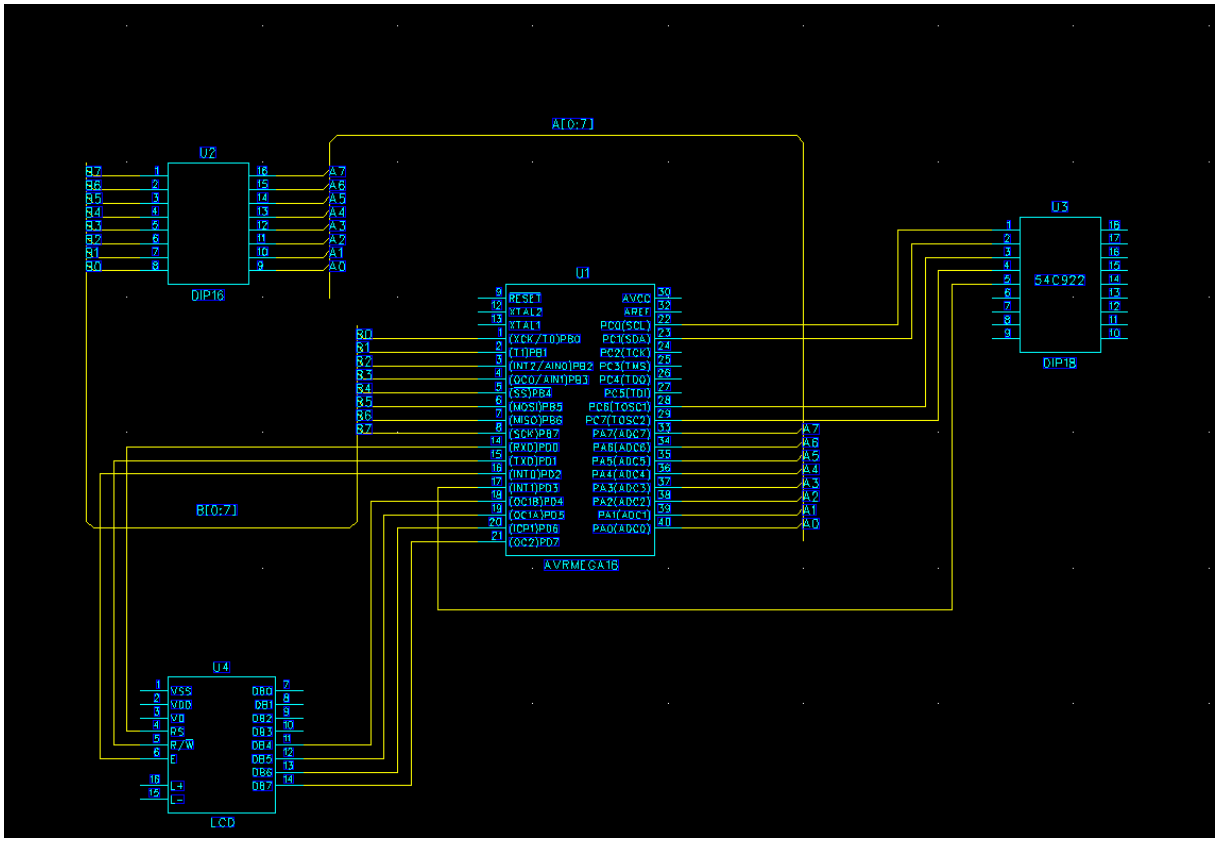



Figure 1: Schematic of the circuit draw in power

# Software

When the power is turned on the LCD, Interrupts and ports are initialized and the device is ready to receive input.

When a key is pressed it is added to the vector containing the device number if it's not on the lowest row. Keys C-F has special functions that control the IC-tester. C and E erase the stored device number, D starts the search mode and F starts the specific test of the stored device number. When the test is complete the results are written to the display, or if the specified device number does not exist an error message is displayed.

# Conclusion

Some problems were encountered one of them being the power supply to the device under test. Because the device is driven from the microcontroller it did not receive a stable 5V supply needed to operate at high frequency.
Other problems include missing detail in the datasheets.

The project have gone smooth, we had realistic goals given the time we had. Some minor improvements were not made, we hoped to make the tests able to identify which parts of the device that is not working. There is an issue with the search mode if this is implemented in that it wouldn't be able to separate certain circuits from each other.

The tester works on the circuits in our list as far as we can tell since all the circuits were not available to test. We are quite satisfied with the result and have learned a great deal about the processor and optimization, the code written takes 46% of the memory with maximum optimization active and without it takes 698,7%.

# Reference:

[1] ATmega16 - Dokumentation för ATmega16 från Atmel 2003

[2] AVR-bibliotek - http://www.nongnu.org/avr-libc/

[3] http://www.eit.lth.se/index.php?fsw=Datablad&dir=Logik/

# Appendix source code

```
//Short version of main.c
#define F_CPU 1000000UL
#define maxsize 5

#include <string.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "LCD.h"
#include "test.h"

int size = 0; // the number of numbers in nbr[]
char nbr[maxsize];

//sets the keypad and lcd ports
void PORT_init()
//sets the needed interrupts
void INT_init()
//runs all tests until match or not found
void search()
//runs the needed functions based on inputs
ISR(SIG_INTERRUPT1)

int main()
{
    PORT_init();
    INT_init();
    LCD_init();
    LCD_clr();
    while(1)
    {
        PORTA = 0;
        PORTB = 0;
    }
    return 0;
}
```

```
//test.h
char test7400(int ic);
```

```
//Really short version of test.c
switch(ic)
{
       // case 7400-74280 [49]
       case 7430:
       {
           #define a30vcc   1
           #define a30h     3
           #define a30g     4
           #define a30y     7

           #define b30a     1
           #define b30b     2
           #define b30c     3
           #define b30d     4
           #define b30e     5
           #define b30f     6
           #define b30gnd   7

           DDRA = (char) ~(1<<a30y | 1<<2 | 1<<5 | 1<<6);
           DDRB = 0xFF;

           PORTA = 1<<a30vcc;
           PORTB = 0;

           if((PINA & 1<<a30y) == 0)
              return 1;

           for(PORTB = 0;PORTB < 128; PORTB++)
           {
              if((PINA & 1<<a30y) == 0)
                 return 1;
              _delay_us(10);
           }

           PORTB = (char)~(1<<b30gnd);

           PORTA |= 1<<a30h;
           if((PINA & 1<<a30y) == 0)
              return 1;

           PORTA |= 1<<a30g;

           if((PINA & 1<<a30y) != 0)
              return 0;

           break;
       }
}

//LCD.h
#define LCD_RS      0
#define LCD_RW      1
#define LCD_E       2
#define LCD_HOME    1
#define LCD_CLEAR   0
```

```
#define LCD_D4      4
#define LCD_D5      5
#define LCD_D6      6
#define LCD_D7      7

#include <avr/io.h>
#include <util/delay.h>

void LCD_write_char(char c);
void LCD_write_int(int c);
void LCD_command(char c);
void LCD_write_hex(unsigned char c);
void LCD_write_byte(char c);
void LCD_write_string(char* ch);

void LCD_init();
void LCD_clr();
void LCD_home();
void LCD_nextline();
```