Embedded Systems Case Sudy

Ilya Dmitrichenko

January 2011

LMU CT3041

Contents

Chapter 1

Emebedded Systems Case Study

For a final year CT3041N module at LMU - a generic project for *Intel* `8051` microcontroller.

Designed using *MCU 8051 IDE (v1.3.11)* by *Martin Ošmera* and *Kara Blackowiak*.

- *MCU 8051 IDE - Project Homepage* - `http://mcu8051ide.sourceforge.net/`
- *Small Device C Compile (SDCC) - Project Homepage* - `http://sdcc.sourceforge.net/`

Foreword Notes

It had been rather difficult to find an appropriate partner for group work, therefore all work was done by one student, Ilya Dmitrichenko.

In order to accomplish the log book component of this assessment unit, revision control software has been utilised. It provides a very appropriate facility for logging the programming activities and keeps track of code modifications. The repository had been also stored on the Internet *GitHub* portal (`http://github.com/errordeveloper/`).

- Repository Homepage
  - `https://github.com/errordeveloper/dev8051-misc-ct3041n/`
- Source Code Tree
  - `https://github.com/errordeveloper/dev8051-misc-ct3041n/tree/master/code/`
- Commit History
  - `https://github.com/errordeveloper/dev8051-misc-ct3041n/commits/master/`

The best attempt was made at documenting the code, however details were omitted for explaining the operation of this very commonly microcontroller used type of circuit.

As mentioned above *MCU 8051 IDE* open-source package had been used to test the code in the simulator. *SDCC* was used to compile the program. Git revision control and Doxygen documentation generator were of great help for the project. *Doxygen* extracts tagged comments from the code.

- *Doxygen Homepage* - `http://www.stack.nl/~dimitri/doxygen/`
- *Git Homepage* - `http://git-scm.com/`

Introduction Notes

This report document details the implementation by describing each of source code files. The description is fallowed by listing of the source code itself. Please note that tagged comments used for description were stripped from the listings. The code under pre-processor condition statements relying on definition of 'DOXYGEN' symbol is not to be used by the compiler.

Please also note that the code assumes C89 or C99 ISO standard compiler.

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 main.c File Reference

Main Program.

```
#include <at89x51.h>
#include "settings.h"
#include "prototypes.h"
```

Functions

- void **main** (void)

3.1.1 Detailed Description

Main Program.

Author

Ilya Dmitrichenko <errordeveloper@gmail.com>

Definition in file **main.c**.

3.1.2 Function Documentation

3.1.2.1 void main (void)

Initialise the DS1620 sensor by writing high and low trigger values and enable MCU mode using **sensor_setup()** (p. 25).

Infinite loop is a common element of an embedded program, it will not exit until the MCU powers-off.

Definition at line **89** of file **main.c**.

3.2 main.c

```
00001
00007 /* Doxygen tagged comments were stripped. */
00008
00080 #ifdef SDCC
00081 #include <at89x51.h>
00082 #else
00083 #include <reg51.h>
00084 #endif
00085
00086 #include "settings.h"
00087 #include "prototypes.h"
00088
00089 void main ( void )
00090 {
00091
00098   sensor_setup();
00099
00106   while(1) {
00107
00108     display_number( sensor_read(), DELAY_CYCLES );
00109
00110     /* READ_DELAY(); */
00111
00112   }
00113
00114 }
```

3.3 prototypes.h File Reference

All function prototypes are in this header.

3.3.1 Detailed Description

All function prototypes are in this header.

Definition in file **prototypes.h**.

3.4 prototypes.h

```
00001
00006 #if !DOXYGEN
00007
00012 void    ts_wait( void );
00013
00014 void    tsc( char byte );
00015
00016 int     tsq( unsigned char mode, int data );
00017
00018 void    sensor_setup( void );
00019
00020 int     sensor_read( void );
00021
00022 void    sensor_test_loop( unsigned char x );
00023
00030 void display_digit( unsigned char d, unsigned char v );
00031
00032 void display_number( int x, unsigned int t );
```

```
00033
00034 void display_test_loop( unsigned char t );
00035
00038 #endif
```

3.5 segment_display.c File Reference

This file implements LED digital display driver.

```
#include <at89x51.h>
#include "settings.h"
#include "prototypes.h"
```

Defines

Software Configuration

These macros can be used to switch different parts of code.

- #define **TESTING_FUNCTIONS** [(0 | 1)]
  *Include **display_test_loop()** (p. 13) function.*
- #define **STANDALONE_TEST** [(0 | 1)]
  *Include **main()** (p. 7) loop for testing.*
- #define **BASIC_METHOD** [(0 | 1)]

LED Interface Pins

These constants define which ports are in use for LED display.

- #define **DIGIT** [(P0 | P1 | P2 | P3)]
- #define **SEGMENT** [(P0 | P1 | P2 | P3)]

- #define **COMMON_PIN** [(ANODE | CATHODE)]
- #define **ANODE** 1
- #define **CATHODE** 0

- #define **SHIFTDIR** [(<< | >>)]
- #define **FIRST_DIGIT** [(0 | 3 | 4 | 7)]

Functions

The Function to Display a Digit

*This function takes the array **figure** (p. 13) and looks-up the code by index to write to pins of **SEGMENT** (p. 12) port. It also selects the right value to pull **DIGIT** (p. 12) pins depending on configuration of **FIRST_DIGIT** (p. 12).*

Parameters

d *digit position (0 to 3)*

v *value of digit (0 to 9)*

- void **display_digit** (unsigned char d, unsigned char v)

The Function to Display a Number

*This function calls **display_digit()** (p. 12)*

Parameters

x *number to display (-999 to 9999)*

t *number of refresh cycles*

It uses an extra array for marking which digit should be OFF in case if it is zero and there no other digit in front of it. It could probably use a 4-bit mask. Bit arrays are not allowed by SDCC.

Note

There is no boundary checking, therefore the value of the argument should be between -999 and 9999. It was considered not appropriate to implement fixed decimal point.

- void **display_number** (int x, unsigned int t)

Display Test Loop

*This is a scrolling loop intended for display hardware tests and basic demo. It can be disabled by setting constant **TESTING_FUNCTIONS** (p. 18) to 0.*

- void **display_test_loop** (unsigned char t)

Standalone Testing Function

*The **main()** (p. 7) function is only compiled when **STANDALONE_TEST** (p. 18) is set to 1.*

- void **main** (void)

Variables

- char **figure** [13]

3.5.1 Detailed Description

This file implements LED digital display driver. The code has been tested with 4-digit 7-segment multiplexed LED display using GUI emulator.

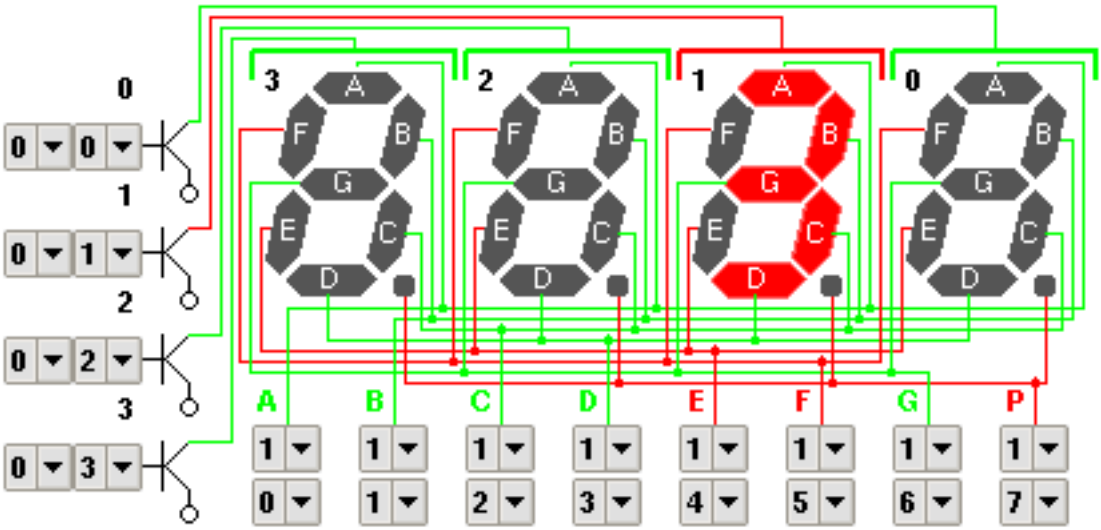


Figure 3.1: MCU 8051 IDE Virtual Multiplexed LED Display

The software is intended to run on MCU with different pin configurations for the connection of LED segments. Using two 8-bit ports up to 8 digits can be driven. There are a few pre-processor macros to configure the topology of pin connection. Common cathode or anode options are included, but reverse segment configuration has not been implemented; though, pins for digit selection may be connected in various ways.

Note

DISPLAY_HOLD() is not implemented. It requires testing with real LED display in order to determin the hold time period, which depends on fade-out time of real LED display.
Therefore the **display_digit()** (p. 12) function turns the segments off in order to prevent effect of moving digit. The function **display_number()** (p. 13) uses second argument to sepcify number of cycles.

Author

Ilya Dmitrichenko <errordeveloper@gmail.com>

Definition in file **segment_display.c**.

3.5.2 Define Documentation

3.5.2.1 #define BASIC_METHOD [(0 | 1)]

There two ways of dealing with the way first digit is connected. One is faster and can be chosen by setting **BASIC_METHOD** (p. 11) to 1. The other method involves an extra bit-shift operation and is more universal. With the second method **SHIFTDIR** (p. 12) needs to be defined only if pins are connected in unusual order. Apply faster code for digit selection.

Definition at line **60** of file **segment_display.c**.

3.5.2.2 #define DIGIT [(P0 | P1 | P2 | P3)]

Set the port for digit selection.

Definition at line **68** of file **segment_display.c**.

3.5.2.3 #define SEGMENT [(P0 | P1 | P2 | P3)]

Set the port for segment selection.

Definition at line **69** of file **segment_display.c**.

3.5.2.4 #define COMMON_PIN [(ANODE | CATHODE)]

LED segment displays can be connected with common cathodes or anodes.

Definition at line **76** of file **segment_display.c**.

3.5.2.5 #define SHIFTDIR [(<< | >>)]

This code can operate in four pin configuration modes, modifications needed for unusual combination of these. When universal method is used this is set to '<<' or '>>'.

Definition at line **90** of file **segment_display.c**.

3.5.2.6 #define FIRST_DIGIT [(0 | 3 | 4 | 7)]

This can be set to 0, 3, 4 or 7. Other values require more changes.

Definition at line **92** of file **segment_display.c**.

3.5.3 Function Documentation

3.5.3.1 void display_digit (unsigned char *d*, unsigned char *v*)

Presuming that the digit 0 is connected to pin 7 of **DIGIT** (p. 12), a clear pattern can be identified:

```
 switch(d)
 {
        case 0:
                DIGIT = 0x80; // 10000000 = 128
        case 1:
                DIGIT = 0x40; // 01000000 = 64
        case 2:
                DIGIT = 0x20; // 00100000 = 32
        case 3:
                DIGIT = 0x16; // 00010000 = 16
        default:
                DIGIT = 0xff; // Off
}
```

It can be expressed as:

```
DIGIT = 128/(2**d)
```

However, the '$**$' operator is not valid in C, and using 'math.h' is not considered appropriate for this small design.

It is in fact most appropriate to use '$>>$' bit-shift operator:

```
DIGIT = (128 >> d);
```

Though, it turns out that the above values (128, 64, 32, 16) are wrong. We should invert the bits:

```
DIGIT = ~(128 >> d);
```

By experiment, two techniques were found. One uses hard-coded assignments for each particular predefined pin configuration. Setting **BASIC_METHOD** (p. 11) to 1 enables this technique.

More universal but computation-intensive method would be:

```
switch(FIRST_DIGIT)
{
        case LOWER:
        ~( (1<<FIRST_DIGIT) >> d ); break;
        case UPPER:
        ~( (1<<FIRST_DIGIT) << d ); break;
}
```

The code, in fact, takes advantage of pre-processor conditional definitions and uses **SHIFTDIR** (p. 12) set to '$<<$' or '$>>$' depending what the value of **FIRST_DIGIT** (p. 12) is set to. In an unusual case **SHIFTDIR** (p. 12) has to be set manually.

Definition at line **186** of file **segment_display.c**.

3.5.3.2 void display_number (int *x*, unsigned int *t*)

Determin the value of each decimal place by using int division and remainder.

Repeat for t times.

If a digit is zero the it it is not displayed, unless there is a non-zero digit in from of it.

Definition at line **310** of file **segment_display.c**.

3.5.3.3 void display_test_loop (unsigned char *t*)

Scroll full count t times.

Definition at line **382** of file **segment_display.c**.

3.5.3.4 void main (void)

Run **display_test_loop()** (p. 13) 2 times.

Definition at line **416** of file **segment_display.c**.

3.5.4 Variable Documentation

3.5.4.1 char figure[13]

Store a digit code look-up table (array) in ROM, perhaps otherwise an enumeration could be used.

There two option for pre-processor to chose from depending of **COMMON_PIN** (p. 12).

Index 10, 11 and 12 are used for '-', 'C' and 'F'.

Definition at line **121** of file **segment_display.c**.

3.6 segment_display.c

```
00001
00034 #ifdef SDCC
00035 #include <at89x51.h>
00036 #else
00037 #include <reg51.h>
00038 #endif
00039
00040 #include "settings.h"
00041 #include "prototypes.h"
00042
00043 #if DOXYGEN /* These definitions are only for documentation. */
00044
00050 #define TESTING_FUNCTIONS       [( 0 | 1 )]
00051
00052 #define STANDALONE_TEST         [( 0 | 1 )]
00053
00060 #define BASIC_METHOD            [( 0 | 1 )]
00068 #define DIGIT [( P0 | P1 | P2 | P3 )]
00069 #define SEGMENT [( P0 | P1 | P2 | P3 )]
00076 #define COMMON_PIN [( ANODE | CATHODE )]
00077
00078 #endif /* DOXYGEN */
00079
00080 #define ANODE 1
00081 #define CATHODE 0
00082
00084 #if DOXYGEN
00085
00090 #define SHIFTDIR [( << | >> )]
00092 #define FIRST_DIGIT [( 0 | 3 | 4 | 7 )]
00095 #endif
00096
00097 #if !BASIC_METHOD
00098 #warning "Computationally intensive universal method will be used!"
00099   #ifndef SHIFTDIR
00100    #if ( (FIRST_DIGIT == 4) || (FIRST_DIGIT == 0) )
00101    #define SHIFTDIR <<
00102    #elif ( (FIRST_DIGIT == 7) || (FIRST_DIGIT == 3) )
00103    #define SHIFTDIR >>
00104    #else
00105    #warning "In universal method values of first digit other then 7, 4, 3 or 0 re
      quire specific direction!"
00106    #endif
00107   #endif
00108 #endif
00109
00120 #if DOXYGEN
00121 char figure[13] = {
00122 #else
00123 static const char figure[13] = {
00124 #endif
00125
00126 #ifdef REVERSE_SEGMENTS
00127 #error "Reverse connection of segments not implemented!"
00128 #endif
00129
00130 #if COMMON_PIN == ANODE
```

```
00131
00132   0xc0, // = 0
00133   0xf9, // = 1
00134   0xa4, // = 2
00135   0xb0, // = 3
00136   0x99, // = 4
00137   0x92, // = 5
00138   0x82, // = 6
00139   0xF8, // = 7
00140   0x80, // = 8
00141   0x90, // = 9
00142   0xbf, // = -
00143   0xc6, // = C
00144   0x8e  // = F
00145
00146 #else /* COMMON_PIN == CATHODE */
00147
00148   0x3F, // = 0
00149   0x06, // = 1
00150   0x5B, // = 2
00151   0x47, // = 3
00152   0x66, // = 4
00153   0x6d, // = 5
00154   0x75, // = 6
00155   0x07, // = 7
00156   0x7f, // = 8
00157   0x6f, // = 9
00158   0x40, // = -
00159   0x39, // = C
00160   0x71  // = F
00161
00162 #endif /* COMMON_PIN */
00163
00164 #if !DOXYGEN
00165 #define SYM_MINUS 10
00166 #define SYM_C 11
00167 #define SYM_F 12
00168 #endif
00169
00170 };
00186 void display_digit( unsigned char d, unsigned char v )
00187 {
00188
00263   SEGMENT = figure[v];
00264
00265   #if BASIC_METHOD
00266
00267    #if ( FIRST_DIGIT == 7 )
00268    DIGIT = ~( 128 >> d );
00269
00270    #elif ( FIRST_DIGIT == 3 )
00271    DIGIT = ~( 8 >> d);
00272
00273    #elif ( FIRST_DIGIT == 4 )
00274    DIGIT = ~( 16 << d);
00275
00276    #elif ( FIRST_DIGIT == 0 )
00277    DIGIT = ~( 1 << d );
00278
00279    #else
00280    #warning "In basic method values of first digit can be 7, 4, 3 or 0 only!"
00281    #endif
00282
00283   #else /* GOOD_METHOD_IN_THEORY */
00284
00285   DIGIT = ~( (1<<FIRST_DIGIT) SHIFTDIR d );
00286
```

```
00287   #endif
00288
00289 }
00290
00310 void display_number( int x, unsigned int t )
00311 {
00312
00313 #ifndef DOXYGEN
00314 #define REM(X, Y) X/Y; X %= Y
00315 #endif
00316
00317   char n[4], m[4] = {1, 1, 1, 1}, i;
00323   if( x >= 0 ) {
00324
00325   n[3] = REM(x, 1000);
00326   if( n[3] == 0 )
00327    m[3] = 0;
00328
00329   n[2] = REM(x, 100);
00330   if( n[2] == 0 && m[3] == 0 ) m[2] = 0;
00331
00332   n[1] = REM(x, 10);
00333   if( n[1] == 0 && m[2] == 0 ) m[1] = 0;
00334
00335   n[0] = x;
00336
00337   } else {
00338
00339   x *= -1;
00340
00341   n[3] = SYM_MINUS;
00342
00343   n[2] = REM(x, 100);
00344   if( n[2] == 0 ) m[2] = 0;
00345
00346   n[1] = REM(x, 10);
00347   if( n[1] == 0 && m[2] == 0 ) m[1] = 0;
00348
00349   n[0] = x;
00350
00351   }
00353   while(t--) {
00354
00355     for( i = 0; i < 4; i++ ) {
00362       if( m[i] ) display_digit( i, n[i] );
00363       /* DISPLAY_HOLD(); */
00364       DIGIT = 0xff;
00365
00366     }
00367
00368   }
00369
00370 }
00371
00381 #if TESTING_FUNCTIONS || DOXYGEN
00382 void display_test_loop( unsigned char t )
00383 {
00384
00385   short unsigned int i, j;
00386
00389   while(t--) {
00390
00391     for( i=0; i<10; i++ ) {
00392
00393       for( j=0; j<4; j++ ) {
00394
00395         display_digit( j, i );
```

```
00396
00397       }
00398
00399     }
00400
00401   }
00402
00403 }
00404 #endif
00405
00406
00407 #ifndef MAIN_PROGRAM
00408
00415 #if STANDALONE_TEST || DOXYGEN
00416 void main (void)
00417 {
00418
00421   //display_test_loop(2);
00422   //DIGIT = 0xff;
00423
00424   display_number(-768, 20);
00425   display_number(1769, 20);
00426   display_number(2761, 20);
00427
00428 }
00429 #endif
00430 #endif
00431
```

3.7 sensor_interface.c File Reference

This file implements DS1620 digital temperature sensor driver.

```
#include <at89x51.h>
#include "settings.h"
#include "prototypes.h"
```

Defines

DS1620 Commands

- #define **READ_TEMP** 0xAA
  Read temperature value from the register.
- #define **WRITE_TH** 0x01
  Write to the TH (Trigger High) register.
- #define **WRITE_TL** 0x02
  Write to the TL (Trigger Low) register.
- #define **READ_TH** 0xA1
  Read the TH (Trigger High) register.
- #define **READ_TL** 0xA2
  Read the TL (Trigger Low) register.
- #define **READ_COUNT** 0xA0

Read the value of the counter byte.

- #define **READ_SLOPE** 0xA9
  Read the value of the slope counter.

- #define **RUN_CONV** 0xEE
  Start temperature conversion.

- #define **END_CONV** 0x22
  Stop temperature conversion.

- #define **WRITE_CONF** 0x0C
  Write configuration register.

- #define **READ_CONF** 0xAC
  Read configuration register.

- #define **TEST** 0x00
  Software testing only.

DS1620 Constants

- #define **WRITE_DELAY** 20
  Time to wait (20 ms) after a `EEPROM` *write.*

- #define **TS_CONF** 10
  Enables CPU-mode and disables single-shot-mode.

Software Configuration

These macros can be used to switch different parts of code.

- #define **TESTING_FUNCTIONS** [(0 | 1)]
  Include ***sensor_test_loop()*** *(*p. *25**) function.*

- #define **STANDALONE_TEST** [(0 | 1)]
  Include ***main()*** *(*p. *7**) loop for testing.*

DS1620 Interface Pins

3-wire connection to the MCU Port

- #define **TS_DATA** [(P0 | P1 | P2 | P3)]
  Data pin for DS1620 sensor chip.

- #define **TS_CLOCK** [(P0 | P1 | P2 | P3)]
  Clock pin for DS1620 sensor chip.

- #define **TS_RESET** [(P0 | P1 | P2 | P3)]
  Enable pin for DS1620 sensor chip.

Macros

Very simplistic handy functions are implemented as macros.

- #define **TS_TICK**() TS_CLOCK = 0; TS_CLOCK = 1
  *Positive pulse on **TS_CLOCK** (p. 18).*
- #define **TS_STOP**() TS_RESET = 0
  *Pull **TS_RESET** (p. 18) low.*
- #define **TS_START**() TS_RESET = 0; TS_CLOCK = 1; TS_RESET = 1
  *Pull **TS_RESET** (p. 18) low, **TS_CLOCK** (p. 18) high, and then **TS_RESET** (p. 18) high.*
- #define **TS_SETUP**()
  Basic DS1620 setup macro example.

Functions

Wait for 20 ms

This function is require when writing data to the `EEPROM` of the `DS1620` sensor chip.

Implemented using either nested `for-loop` or inline assembly function (quote below). The MCU 8051 IDE provides delay loop calculator, which was used to generate the assembly code.

The default behaviour is to use a nested for-loop. This behaviour can be changed using pre-processor constant USE_ASM_TS_WAIT. Skelleton code had been also wirtten with different C compiler in mind.

```
; START: Wait loop, time: 20 us
; Clock: 14745.6 kHz (12 / MC)
; Used registers: R0
; Rest: -889.6 ns
LOOP:
       MOV     R0, #007h
       NOP
       DJNZ    R0, $
       NOP
       NOP
```

- void **ts_wait** (void)

Temperature Sensor Command

Parameters

byte *- 8-bit command to write.*

*Writes a command byte (bit-by-bit) to the **TS_DATA** (p. 18) pin, applying **TS_TICK()** (p. 19) after each bit. Code line below performs the `"shift-right-AND-mask"` operation.*

```
TS_DATA = ( bytes >> n ) & 0x01;
```

- void **tsc** (char byte)

Temperature Sensor Query

*This function implements all-in-one DS1620 sensor control. This is probably the most straight- forward way of communicating with the sensor. Having multiple function is not considered to be of any particular use. The only helper function that is needed is **tsc()** (p. 23).*

Parameters

mode *- the command to operate (sets the mode)*

data *- value to write, pass '0' when reading*

- int **tsq** (unsigned char mode, int data)

Temperature Sensor Setup

Basic setup function

- void **sensor_setup** (void)

Temperature Sensor Read

Basic read function for data acquisition.

- int **sensor_read** (void)

Sensor Test Loop

This is a loop intended for hardware tests. It can be disabled by setting constant ***TESTING_FUNCTIONS*** *(p. 18) to 0.*

- void **sensor_test_loop** (unsigned char x)

Standalone Testing Function

The ***main()*** *(p. 7) function is only compiled when* ***STANDALONE_TEST*** *(p. 18) is set to 1.*

- void **main** (void)

Variables

- short int **temp** = 0

3.7.1 Detailed Description

This file implements DS1620 digital temperature sensor driver. The information had been extracted from datasheets and code examples. The objective is to implement 3-wire communication protocol that uses 8-bit commands to query for measurement and configuration data, or storing new configuration. The data is of 9-bit length, with 9th bit to represent the sign, therefore some manipulation routines are required.

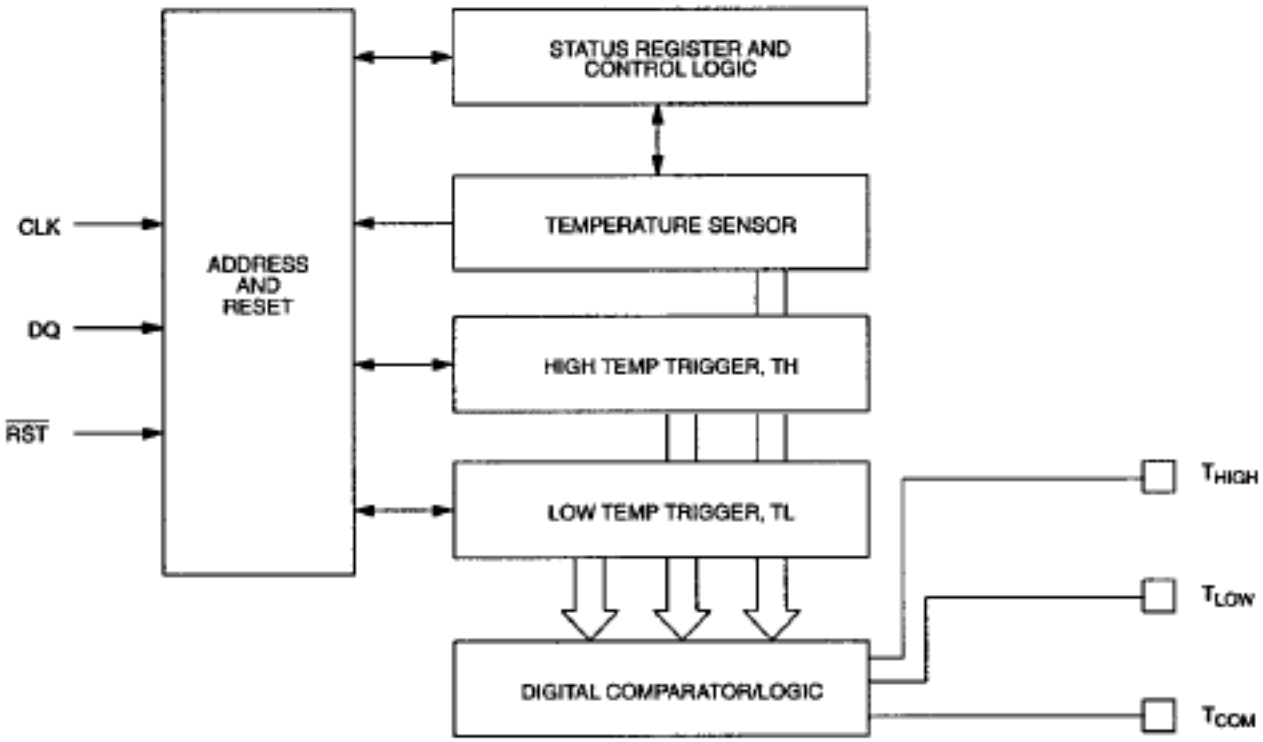


Figure 3.2: DS1620 Digital Thermometer and Thermostat (function block diagram)

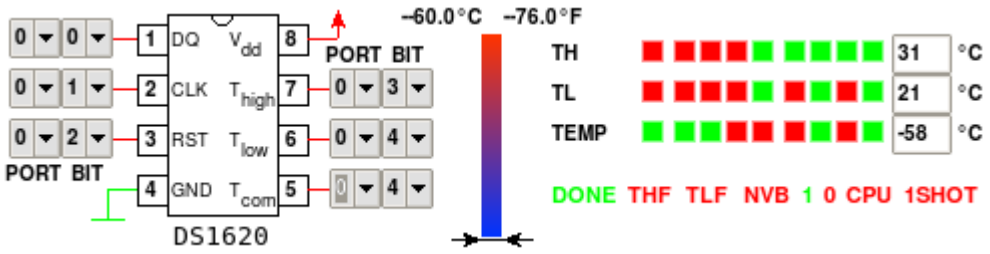


Figure 3.3: Virtual DS1620 in MCU 8051 IDE

Some code has been borrowed from `DS1620` Arduino C++ Library:

- `https://github.com/thinkhole/Arduino-DS1620/blob/master/ds1620.cpp`

However, after some consideration of how it would be used in the full context, a single multi-purpose function had been developer.

For datasheet information see Dallas Semiconductors, Maxim Integrated Products:

- `http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2735`
- `http://datasheets.maxim-ic.com/en/ds/DS1620.pdf`

Author

Ilya Dmitrichenko <errordeveloper@gmail.com>

Definition in file **sensor_interface.c**.

3.7.2 Define Documentation

3.7.2.1 #define WRITE_DELAY 20

Time to wait (20 ms) after a EEPROM write.

This constant is not in use, see **ts_wait()** (p. 22) .

Definition at line **87** of file **sensor_interface.c**.

3.7.2.2 #define TS_CONF 10

Enables CPU-mode and disables single-shot-mode.

See also

DS1620 datasheet for details

Definition at line **90** of file **sensor_interface.c**.

3.7.2.3 #define TS_SETUP( )

Value:

```
tsq(WRITE_TL, 15); \
tsq(WRITE_TH, 30); \
tsq(WRITE_CONF, TS_CONF); \
tsq(RUN_CONV, 0)
```

Basic DS1620 setup macro example.

Definition at line **134** of file **sensor_interface.c**.

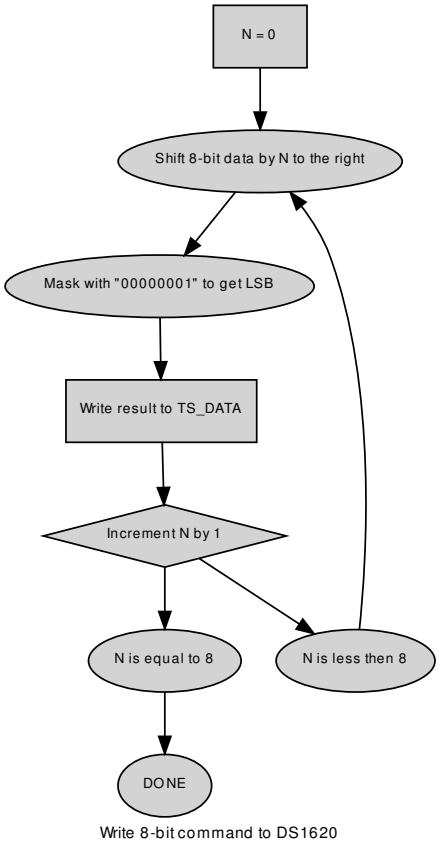
3.7.3 Function Documentation

3.7.3.1 void ts_wait (void)

Total run of nested for loops will be 50.

Definition at line **175** of file **sensor_interface.c**.

3.7.3.2 void tsc (char *byte*)

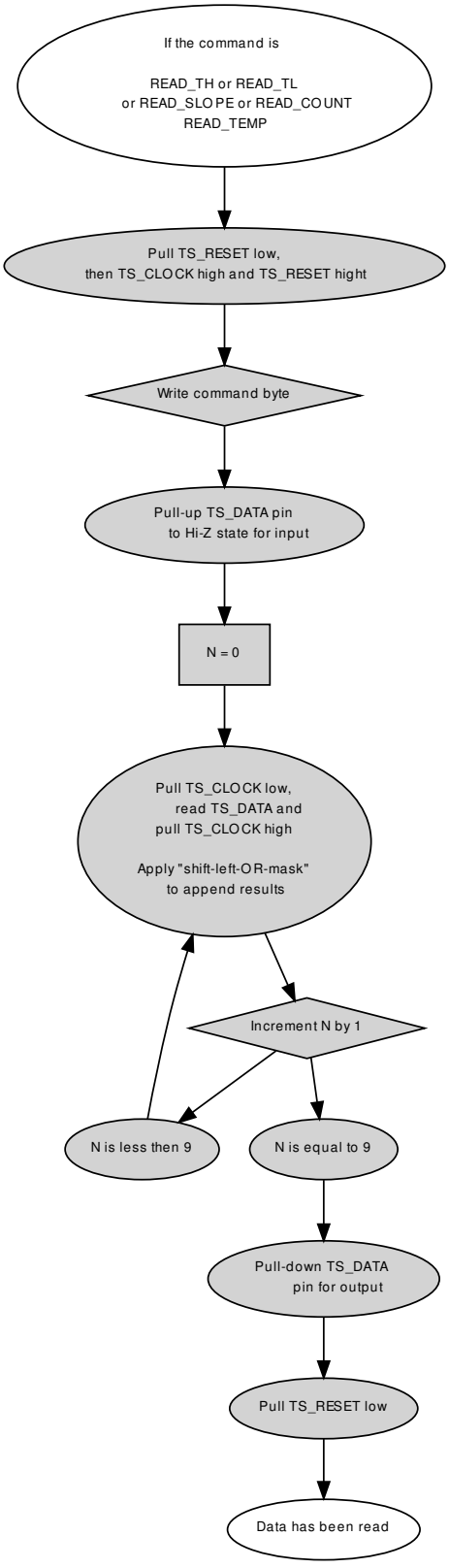


Definition at line **222** of file **sensor_interface.c**.

3.7.3.3 int tsq (unsigned char *mode,* int *data*)

Switch operation modes, depending on command:

- Start or Stop conversion:
  - Only send the command, namely **RUN_CONV** (p. 18) or **END_CONV** (p. 18)
  - Will always return zero
- Read 9-bit data for the fallowing commands:
  - **READ_SLOPE** (p. 18) or **READ_COUNT** (p. 17)
  - **READ_TH** (p. 17) or **READ_TL** (p. 17)
  - **READ_TEMP** (p. 17)
  - *The sequence is illustrated in the flow diagram*



Read 9-bit data from DS1620

As it can be see from the flow digaram, the procedure underneath this function is quite linear. Further bit manipulation is required for handling of negative value representation:

```
temp = data & 0x00ff;

temp = 0x0100 - temp;

temp *= (short)-1;

return ( temp / 2 );
```

The idea behind writing **tsq()** (p. 23) as one long function is to demonstrate the linearity and general nature of operation, as well optimising the size on the chip.

- Write 9-bit data for the **WRITE_TL** (p. 17) or **WRITE_TH** (p. 17)
  - Send command using tsc(mode)
  - Multiply the data value by 2
  - Write bits using "shift-right-AND-mask" operation
- Write 8-bit data to the configuration register
- Test case: return the data

Definition at line **279** of file **sensor_interface.c**.

3.7.3.4 void sensor_setup (void)

Use **TS_SETUP()** (p. 22) macro.

Definition at line **541** of file **sensor_interface.c**.

3.7.3.5 int sensor_read (void)

Use **tsq()** (p. 23) to send READ_TEMP command and 0 as the data value to specify 'read' mode.

Definition at line **556** of file **sensor_interface.c**.

3.7.3.6 void sensor_test_loop (unsigned char *x*)

Use **TEST** (p. 18) command to check basic functionality.

Definition at line **577** of file **sensor_interface.c**.

3.7.3.7 void main (void)

Run **sensor_test_loop()** (p. 25) 2 times.

Definition at line **600** of file **sensor_interface.c**.

3.7.4 Variable Documentation

3.7.4.1 short int temp = 0

Gloabal temperature variable.

Definition at line **145** of file **sensor_interface.c**.

3.8 sensor_interface.c

```
00001
00035 /* Doxygen tagged comments were stripped. */
00036
00037 #ifdef SDCC
00038 #include <at89x51.h>
00039 #else
00040 #include <reg51.h>
00041 #endif
00042
00043 #include "settings.h"
00044 #include "prototypes.h"
00045
00052 #define READ_TEMP  0xAA
00053
00054 #define WRITE_TH   0x01
00055
00056 #define WRITE_TL   0x02
00057
00058 #define READ_TH    0xA1
00059
00060 #define READ_TL    0xA2
00061
00062 #define READ_COUNT 0xA0
00063
00064 #define READ_SLOPE 0xA9
00065
00066 #define RUN_CONV 0xEE
00067
00068 #define END_CONV  0x22
00069
00070 #define WRITE_CONF 0x0C
00071
00072 #define READ_CONF  0xAC
00073
00074 #define TEST 0x00
00075
00087 #define WRITE_DELAY 20
00088
00090 #define TS_CONF 10
00091
00093 #if DOXYGEN
00094
00100 #define TESTING_FUNCTIONS       [( 0 | 1 )]
00101
00102 #define STANDALONE_TEST         [( 0 | 1 )]
00103
00111 #define TS_DATA  [( P0 | P1 | P2 | P3 )]
00112
00113 #define TS_CLOCK [( P0 | P1 | P2 | P3 )]
00114
00115 #define TS_RESET [( P0 | P1 | P2 | P3 )]
00116
00118 #endif
00119
```

```
00125 #define TS_TICK() TS_CLOCK = 0; TS_CLOCK = 1
00126
00127 #define TS_STOP() TS_RESET = 0
00128
00131 #define TS_START() TS_RESET = 0; TS_CLOCK = 1; TS_RESET = 1
00132
00133 //#if DOXYGEN
00134 #define TS_SETUP() \
00135 tsq(WRITE_TL, 15); \
00136 tsq(WRITE_TH, 30); \
00137 tsq(WRITE_CONF, TS_CONF); \
00138 tsq(RUN_CONV, 0)
00139 //#endif
00140
00141 /* #define TS_WAIT() */
00142
00145 short int temp = 0;
00175 void ts_wait( void )
00176 {
00177 #ifdef USE_ASM_TS_WAIT
00178
00179 #ifdef SDCC
00180 _asm
00181         MOV     R0, #007h
00182         NOP
00183         DJNZ    R0, $
00184         NOP
00185         NOP
00186 _endasm;
00187 #else /* !SDCC */
00188 INLINE_ASM_BEGIN_KEYORD
00189         MOV     R0, #007h
00190         NOP
00191         DJNZ    R0, $
00192         NOP
00193         NOP
00194 INLINE_ASM_END_KEYWORD
00195 #endif /* SDCC || !SDCC */
00196
00197 #else /* !USE_ASM_TS_WAIT */
00198
00199   unsigned char i, j;
00200
00202   for( i = 0; i < 10; i++ ) {
00203
00204     for( j = 0; j < 5; j++ ) {;}
00205
00206   }
00207
00208 #endif /* USE_ASM_TS_WAIT */
00209 }
00210
00222 void tsc( char byte )
00223 {
00224
00254   char n;
00255
00256   for( n = 0; n < 8; n++ ) {
00257
00258     TS_DATA = ( byte >> n ) & 0x01;
00259
00260     TS_TICK();
00261
00262   }
00263
00264 }
00265
```

```
00279 int tsq( unsigned char mode, int data )
00280 {
00281
00282   char n; int read = 0;
00283
00308
00313   switch( mode ) {
00314
00320     case RUN_CONV:
00321     case END_CONV:
00322
00323       TS_START();
00324
00325       tsc(mode);
00326
00327       TS_STOP();
00328
00329       break;
00330
00406     case READ_SLOPE:
00407     case READ_COUNT:
00408     case READ_TEMP:
00409     case READ_TH:
00410     case READ_TL:
00411
00412
00413       TS_START();
00414
00415       /* Send the command byte. */
00416       tsc(mode);
00417
00418       /* Pull-up to Hi-Z state for input. */
00419       TS_DATA = 1;
00420
00421       /* Read data with negative clock pulse. */
00422       for( n = 0; n < 9; n++ ) {
00423
00424         TS_CLOCK = 0;
00425
00426         read = TS_DATA;
00427
00428         TS_CLOCK = 1;
00429
00430         data = data | read << n;
00431
00432       }
00433
00434       /* Pull-down the pin state for writing. */
00435       TS_DATA = 0;
00436
00437       TS_STOP();
00438
00439       if( mode == READ_TL || \
00440           mode == READ_TH || \
00441           mode == READ_TEMP ) {
00442
00443         /* Check for negative temperature reading. */
00444         if(( mode == READ_TEMP ) \
00445           && ( data & 0x0100 )) {
00446
00447             temp = data & 0x00ff;
00448
00449             temp = 0x0100 - temp;
00450
00451             temp *= (short)-1;
00452
00453             return ( temp / 2 );
```

```
00454
00455           } else {
00456
00457             return ( data / 2 );
00458
00459           }
00460
00461       } else {
00462
00463         return ( data );
00464
00465       }
00466
00467       /* break; */
00468
00476     case WRITE_TH:
00477     case WRITE_TL:
00478
00479       TS_START();
00480
00481       tsc(mode);
00482
00483       if( mode == WRITE_TL || mode == WRITE_TH ) {
00484
00485         data *= 2;
00486
00487       }
00488
00489       /* Write data with negative clock pulse. */
00490       for( n = 0; n < 9; n++ ) {
00491
00492         TS_DATA = ( data >> n ) & 0x01;
00493
00494         TS_TICK();
00495
00496       }
00497
00498       ts_wait();
00499
00500       TS_STOP();
00501
00502       break;
00503
00506     case WRITE_CONF:
00507
00508       TS_START();
00509
00510       tsc(mode);
00511
00512       tsc(data);
00513
00514       ts_wait();
00515
00516       TS_STOP();
00517
00518       break;
00519
00521     case TEST:
00522
00523         TS_START();
00524
00525         TS_STOP();
00526
00527         return ( data );
00528
00529         /* break; */
00530
```

```
00531   }
00532
00533   return 0;
00534 }
00535
00541 void sensor_setup( void )
00542 {
00543
00546   TS_SETUP();
00547
00548 }
00549
00556 int sensor_read( void )
00557 {
00558
00564   return tsq(READ_TEMP,0);
00565
00566 }
00567
00576 #if TESTING_FUNCTIONS || DOXYGEN
00577 void sensor_test_loop( unsigned char x )
00578 {
00579
00580   while(x--) {
00581
00584     tsq(TEST, 100);
00585
00586   }
00587
00588 }
00589 #endif
00590
00591 #ifndef MAIN_PROGRAM
00592
00599 #if STANDALONE_TEST || DOXYGEN
00600 void main (void)
00601 {
00602
00605   sensor_test_loop(2);
00606
00607 }
00608 #endif
00609 #endif
```

3.9 settings.h File Reference

This header is to be included in all source code C files.

3.9.1 Detailed Description

This header is to be included in all source code C files. All documentation is included in the driver source code C files. This header is for software configuration at the compile time.

Author

Ilya Dmitrichenko <errordeveloper@gmail.com>

Definition in file **settings.h**.

3.10 settings.h

```
00014 #if !DOXYGEN
00015
00016 #define MAIN_PROGRAM
00017
00018 /* Configure some options to compile.
00019  * Including optimisation and debug.
00020 */
00021 #define TESTING_FUNCTIONS       1
00022 #define STANDALONE_TEST         1
00023
00024
00025 /* Cofiguration of segment_display.c
00026 */
00027
00028 #define BASIC_METHOD            0
00029
00030 /* Configure which ports LED display is connected to.
00031  * Use (P0), (P1), (P2) or (P3).
00032 */
00033 #define DIGIT           (P0)
00034 #define SEGMENT         (P1)
00035
00036 /* Set the pin number to which the first
00037  * digit (least significant) is connected.
00038 */
00039 #define FIRST_DIGIT             (0)
00040
00041 /* Change the common pin setting.
00042  * Use (ANODE) or (CATHODE).
00043 */
00044 #define COMMON_PIN              ANODE
00045
00046
00047 /* Cofiguration of sensor_interface.c
00048 */
00049
00050 /* Configure which ports DS1620 sensor is connected to.
00051  * Use (P0), (P1), (P2) or (P3).
00052 */
00053 #define TS_DATA         (P2_0)
00054 #define TS_CLOCK        (P2_1)
00055 #define TS_RESET        (P2_2)
00057 #endif
```

Index