
AVR301: C Code for Interfacing AVR[®] to AT17CXXX FPGA Configuration Memories

Features

- C Routines for Software Two-wire Interface (TWI)
- Example Circuit for AVR Programming FPGA Configuration Memories
- User Programmable Speed
- Supports AT17C/LVXXX and AT24Cxxx Families of EEPROM

Introduction

This application note describes how to In-System-Program (ISP) an Atmel FPGA Configuration memory using an Atmel AVR microcontroller and how to bit bang a Two-wire Interface (TWI) using port pins on an AT90S8515 AVR microcontroller. The AT17C/LVXXX family of FPGA configuration memories, ranging from 512K bits to 1M bit, uses the 2-wire protocol for In-System Programming.

This application note assumes that the user is familiar with the AT90S8515 and AT17 series EEPROMs data sheets and the application note titled "Programming Specification for Atmel's AT17 and AT17A series FPGA Configuration EEPROMs".

This application note is written specifically for the 512K/1M device. C routines to read and write data are included. The code can easily be recompiled for all AVR controllers with 256 bytes SRAM or more.



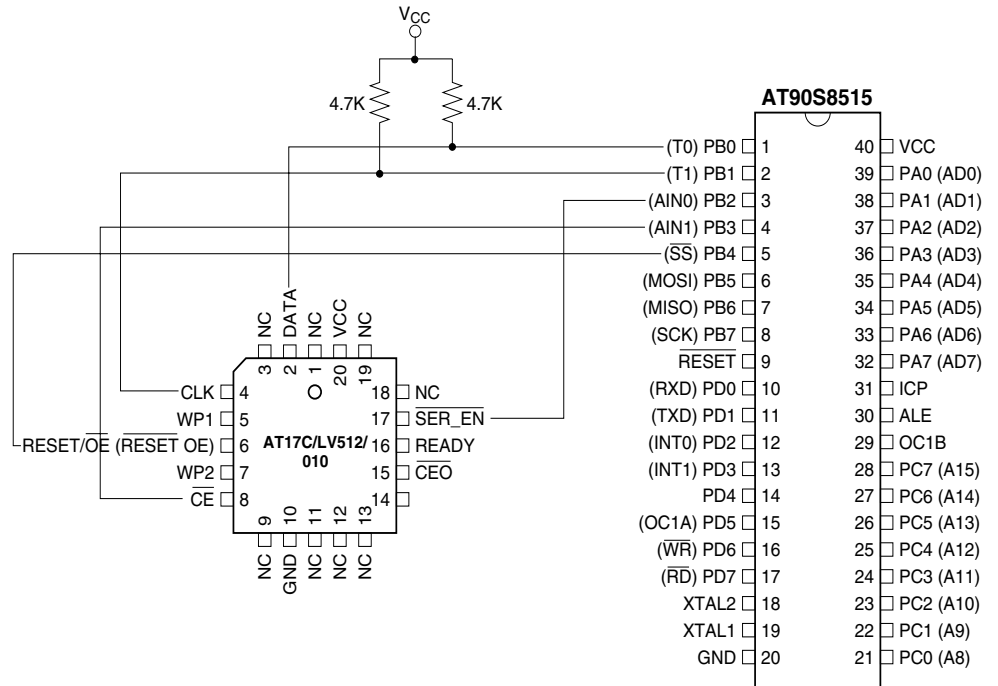
8-bit AVR[®]
Microcontroller

Application
Note

Rev. 1450C-AVR-01/03



Figure 1. Circuit Diagram



Theory of Operation

The TWI is a Two-wire synchronous serial interface consisting of one data (SDA) and one clock (SCL) line. By using open drain/collector outputs, the TWI supports any fabrication process (CMOS, bipolar and more). The TWI is a multi-master bus where one or more devices, capable of taking control of the bus, can be connected. When there is only one Master connected to the bus, the resulting code is much simpler because handling of bus contentions and inter Master access (a Master accessing another Master) is not necessary. Only master devices can drive both the SCL and SDA lines while a Slave device is only allowed to issue data on the SDA line. This application note implements a single Master TWI.

Software Description

The generic TWI routines listed below were compiled with size optimization using IAR's C Compiler Version 1.40. These routines implement a single Master TWI implementation. The AT90S8515 used to perform the master function is driven by an external 7.3728 MHz crystal. The routine BitDelay is executed in 15 clock cycles or a 2.03 μ s period and provides the quarter period bit timing necessary to meet the 3.3V timing specifications found in the above mentioned application note.

This code uses PORTB of the AT90S8515. On power-up, PORTB is initialized to all inputs with the internal pull-ups turned off, the external pull-ups pull the SDA and SCL lines high and the PORTB output latch bits SCL and SDA are initialized to zero. Routines WriteSDA and WriteSCL toggle their respective data direction bit depending on the value of parameter "state". When state is a "1" the port pin is configured as input (external pull-ups pull high). When state is a "0" the port pin is configured as an output and the latch drives the pin low. Table 1 lists the generic TWI routines and the amount of code space used by each routine. WriteSDA and WriteSCL are very simple routines that could be incorporated into their respective calling routines to further reduce the code size.

Table 1 lists the number of clock cycles used while implementing the function. Compiler options were set to generate minimum code.

Table 1. Size and Execution Time for TWI Routines

Routine	Clocks	Bytes
SendStartBit	138	22
SendByte	1050-1054	74
SendStopBit	110	16
BitDelay	15	10
SetSCLHigh	33	38
WriteSCL	12-13	12
WriteSDA	12-13	12
GetByte	1089-1090	66

General Calling Sequence for the TWI Routines

Write:

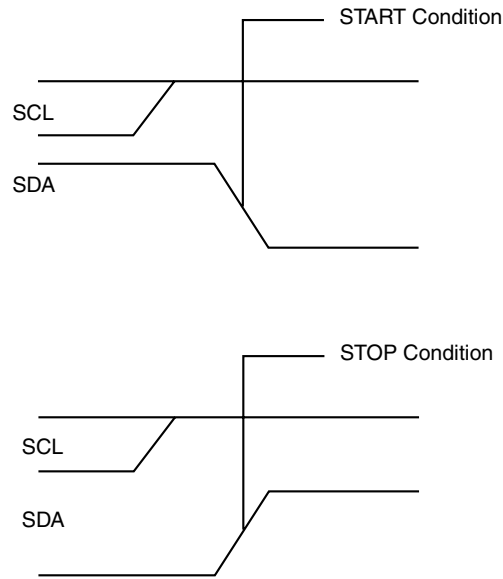
```
SendStartBit();          /*start* /
SendByte(byte,msbfirst); /*send address MSB first*/
SendByte(byte,lsbfirst); /*send data byte to that*/
                          /*address LSB first*/
SendStop():             /*stop*/
```

Read:

```
SendStartBit();          /*start*/
SendByte(byte,msbfirst); /*send address MSB first*/
byte = GetByte(lastbyte); /*read byte from that*/
                          /*address last byte = 1
                          /*for the last byte*/
                          /*in a serial stream*/
SendStop();             /*stop*/
```

The routines SendStartBit, SendByte, and GetByte all leave the SCL signal low on exit, allowing the next routine to write data to SDA (TWI allows changes on SDA only when SCL is low; Otherwise the TWI device will interpret a start or stop condition). SendByte returns a flag indicating a successful write to the TWI Slave, 0x01 signals that the Slave did not acknowledge the transfer and that something is wrong on the TWI bus or with the Slave. WritePage and ProgramResetPolarity use this flag for data polling. Figure 1 shows TWI start and stop bit conditions.

Figure 2. TWI Start and Stop Bits



The AT17C/LV512/010 device is programmed/verified on a 128-byte page boundary. During normal FPGA configuration operations the read of the device starts at address 0 and continues until the FPGA has completed its configuration. The routines WritePage and ReadPage write and read 128-byte pages from the configuration memory and use the generic TWI routines to perform this function. WritePage and ReadPage are both called with the page address to write/read to, and a pointer to a 128-byte page buffer. At the end of a page write the data polling method is used to determine the end of the internal page programming cycle. ProgramResetPolarity and VerifyResetPolarity write and read data from specific memory locations 0x2000H - 0x20003H for 512K/1M in effect setting and verifying the reset/ \overline{oe} polarity.

Table 2. Code Size and Execution Time for Page Read/Write

Routines	Cycles	Bytes
WritePage	287872	146
ReadPage	145901	126
ProgramResetPolarity	156104	106
VerifyResetPolarity	157269	98

PORTB on the AT90S8515 is used to communicate with the FPGA configuration memory. Bit assignments are as follows:

```

/*PB0 = SDA*/
/*PB1 = SCL*/
/*PB2 = SER_EN - used to put AT17C010 in I2C mode*/
/*PB3 = CS*/
/*PB4 = RESET/OE*/

```

The routine Init.c initializes the AT90S8515 peripherals. Routines Timer0.c and Timer1.c are general purpose time-out routines. Main is used to call WritePage, ReadPage, ProgramResetPolarity, and VerifyResetPolarity and serves to illustrate proper calling conventions for those routines.

Modifications and Optimizations

Impact on Changing Crystal Frequency

If the user decides to change oscillator frequencies then the following routines would have to be modified:

- BitDelay
- ProgramResetPolarity
- SetSCLHigh
- WritePage

BitDelay uses NOP's to effect a quarter bus period delay. Add or remove NOP's to increase or decrease the delay. In the routines ProgramResetPolarity and WritePage Timer1 is used to generate a Time-out after 20 milliseconds, the programming operation should have completed by then. Timer0 is used in SetSCLHigh to generate a Time-out after 35 microseconds. If the SCL line is not high by then, then something is wrong on the bus.

References

Programming Specification for Atmel's Configuration Memory EEPROMS – Atmel Corporation

AT17C/LV512/010 FPGA Configuration Memory Datasheets – Atmel Corporation

AT90S8515 Datasheet – Atmel Corporation

Codes

Main.c

/* main use the STK200 starter kit and external prototype board to program an Atmel AT17C/LV512/010 device. LEDs on PORTC are used as visual status indicators 4.7k ohm pull-up resistors are used on SDA and SCL */

```
#include "2wi.h"

volatile unsigned char t0_timed_out;
volatile unsigned char t1_timed_out;
tiny unsigned char wrbuf[PAGE_SIZE];
unsigned char rdbuf[PAGE_SIZE];

extern void WritePage(unsigned int address, unsigned char *bufptr);
extern void ReadPage(unsigned int address, unsigned char *bufptr);
void ProgramResetPolarity(unsigned char state);
unsigned char VerifyResetPolarity(void);
extern void Init(void);

void C_task main(void)
{
    unsigned int address = 0;
    unsigned char i;

    Init();

    _SEI(); /* enable interrupts */

    /* init test buffer */
    for (i = 0; i < PAGE_SIZE; i++)
        wrbuf[i] = i;

    while (1)
    {
        PORTC = 0x00; /* LEDs off */

        for (address = 0; address < MAX_PAGES; address++)
        {
            WritePage(address, &wrbuf[0]);
            PORTC = address; /* LEDs on */
            ReadPage(address, &rdbuf[0]);

            /* verify programmed page */
            for (i = 0; i < PAGE_SIZE; i++)
            {
                if (wrbuf[i] != rdbuf[i])
                {
                    PORTC = 0xff; /* LEDs on */
                }
            }
        }
    }
}
```

```

        ProgramResetPolarity(0);
        if (VerifyResetPolarity())
        {
            PORTC = 0xaa;
        }

        ProgramResetPolarity(0xff);
        if (!VerifyResetPolarity())
        {
            PORTC = 0x55;
        }

        /* clear verify buf */
        for (i = 0; i < PAGE_SIZE; i++)
            rdbuf[i] = 0;
    }
} /* while (1) */
} /* main */

/* BitDelay.c for a 3.3v device min clock pulse width low and high is 4 us generate 2us delay for bit timing
using NOP's 7.3728MHz crystal */

#include "2wi.h"

void BitDelay(void)
{
    char delay;

    delay = 0x03;
    do
    {
        while(--delay)
        ;
        _NOP();
    }
    return;
}

```

Getbyte.c

```

/* GetByte.c reads a byte from the TWI Slave, lastbyte is used to tell Slave that the read is over */

#include "2wi.h"

extern void WriteSCL(unsigned char state);
extern void WriteSDA(unsigned char state);
extern void BitDelay(void);
extern unsigned char SetSCLHigh(void);

unsigned char GetByte(unsigned char lastbyte)
{

```

```

/* lastbyte == 1 for last byte */

unsigned char i, bit;
unsigned char result = 0;

DDRB &= 0xfe; /* SDA to input */

for (i = 0; i < 8; i++)
{
/* each bit at a time, LSB first */
    SetSCLHigh();
    BitDelay();
    bit = (PINB & 0x01);
    result = (bit << (i)) | result;
    WriteSCL(0);
    BitDelay();
}

/* send ACK */
WriteSDA(lastbyte); /* no ack on last byte ... lastbyte = 1 for the last byte */
BitDelay();
SetSCLHigh();
BitDelay();
WriteSCL(0);
BitDelay();
WriteSDA(1);
BitDelay();

return(result);
}

```

Init.C

```

/* init.c */

#include "2wi.h"

void Init(void)
{
/* P0 = SDA - bidirectional */
/* P1 = SCL - output */
/* P2 = SER_EN - output */
/* P3 = CS - output */
/* P4 = RESET/OE - output */

/* P7, P6, P5, P4, P3, P2, P1, P0 */
/* 0 0 0 0 0 0 0 0 */
/* 1 1 1 1 1 1 1 1 */
DDRB = 0xfc;
PORTB = 0xfc;

```



```

/* Port c used to light leds on prototype board */
DDRC = 0xff;
PORTD |= 0xff;
PORTC = 0x00; /* 0 turns leds off ... really should be the other way around */

/* use pushbutton switches on atstk200 kit to start events */
DDRD = 0x00;

TCCR1A = 0x00; /* timer/counter 1 PWM disable */
TCCR1B = 0x00; /* timer/counter 1 clock disable */
TCNT1H = 0x00;
TCNT1L = 0x00; /* clear counter */
TCNT0 = 0x00;
TCCR0 = 0x00; /* stop the clock */
TIMSK |= 0x82; /* enable timer counter 0 & 1 interrupt on overflow */
}

```

ProgramResetPolarity.c

/* ProgramResetPolarity.c Locations 0x20000H through 0x20003H are used to store the reset/output enable polarity for 512K/1M.
0xff = active low reset and active low output enable.
0x00 = active high reset and active high output enable.
So the memory location values determine the reset polarity.
After programming the data polling method is used to determine the end of the internal programming cycle.
Timer/Counter1 is used to ensure the polling code exits its while loop. */

```

#include "2wi.h"

extern volatile unsigned char t1_timed_out;

extern unsigned char SendByte(unsigned char byte, unsigned char order);
extern void SendStartBit(void);
extern void SendStopBit(void);

void ProgramResetPolarity(unsigned char state)
{
    unsigned char i;
    unsigned char test_ack = 0xff;

    PORTB &= 0xf7; /* bring CS low */
    PORTB &= 0xef; /* bring RESET/OE low */
    PORTB &= 0xfb; /* bring SER_EN low */

    SendStartBit();
    SendByte(AT17 + WRITE,MSB_FIRST); /* send device address byte */
    SendByte(0x02,MSB_FIRST); /* 1st address byte */
    SendByte(0x00,MSB_FIRST); /* 2nd address byte */
    SendByte(0x00,MSB_FIRST); /* 3rd address byte ... most significant byte */
}

```

```

for (i = 0; i < 4; i++)
    SendByte(state,LSB_FIRST);

SendStopBit();

t1_timed_out = FALSE; /* set in timer counter 0 overflow interrupt routine */

/* 20 milli-second timeout */
/* 7.3728MHz / 1024 = 7200 Hz */
/* 7200 Hz = 138.8 us */
/* 20 ms / 138.8 us = 144.09 */
/* 65536 - 144 = 65392 = ff70 */
/* interrupt on ffff to 0000 transition */
TCNT1H = 0xff;
TCNT1L = 0x70; /* load counter */
TCCR1B = 0x05; /* timer/counter 1 clock / 1024 */

/* continue sending start bit and device address until we get an ack back */
/* data poll to program complete ... time out for error */
while (test_ack && !t1_timed_out)
{
    SendStartBit();
    test_ack = SendByte(AT17 + WRITE,MSB_FIRST); /* send device address byte */
}
SendStopBit();

PORTB |= 0x04; /* bring SER_EN high */
PORTB |= 0x10; /* bring RESET/OE high */
PORTB |= 0x08; /* bring CS high */
}

```

ReadPage.c

/* ReadPage.c Read PAGE_SIZE (128) bytes at address into bufptr Starts reading at address 0 within the page
Please refer to the application note titled: "Programming Specification for Atmel's AT17 and AT17A series FPGA
Configuration EEPROMS" found at www.atmel.com for detailed device address decoding and page address
formatting */

```

#include "2wi.h"

extern void BitDelay(void);
extern unsigned char SendByte(unsigned char byte, unsigned char order);
extern unsigned char GetByte(unsigned char lastbyte);
extern void SendStartBit(void);
extern void SendStopBit(void);

void ReadPage(unsigned int address, unsigned char *bufptr)
{
    unsigned char i;
    unsigned char addr1;
    unsigned char addr2;
    unsigned char addr3;

```

```

PORTB &= 0xf7; /* bring CS low */
PORTB &= 0xef; /* bring RESET/OE low */
PORTB &= 0xfb; /* bring SER_EN low */

BitDelay();

addr1 = (unsigned char)(address >> 9);
addr2 = (unsigned char)(address >> 1);
addr3 = (unsigned char)(address << 7);

SendStartBit();
SendByte(AT17 + WRITE,MSB_FIRST);          /* send device address byte */
SendByte(addr1,MSB_FIRST);                 /* 1st address byte */
SendByte(addr2,MSB_FIRST);                 /* 2nd address byte */
SendByte(addr3,MSB_FIRST);                 /* 3rd address byte */

SendStartBit();
SendByte(AT17 + READ,MSB_FIRST);          /* send device address byte with read bit */

for (i = 0; i < (PAGE_SIZE-1); i++)
    bufptr(PAGE_SIZE-1) = GetByte(0);

bufptr[PAGE_SIZE-1] = GetByte(1); /* 1 signals last byte of read sequence */

SendStopBit();

PORTB |= 0x04; /* bring SER_EN high */
PORTB |= 0x10; /* bring RESET/OE high */
PORTB |= 0x08; /* bring CS high */
}

```

SendByte.c

/* SendByte.c send a byte of address or data to the TWI Slave parameter order used to select between sending LSB or MSB first returns a 1 if the Slave didn't ack and a 0 if the Slave did */

```

#include "2wi.h"

extern void WriteSCL(unsigned char state);
extern void WriteSDA(unsigned char state);
extern void BitDelay(void);
extern unsigned char SetSCLHigh(void);

unsigned char SendByte(unsigned char byte, unsigned char order)
{
    unsigned char i;
    unsigned char error;

    for (i = 0; i < 8; i++)
    {

```

```

if (order)
{
    WriteSDA(byte & 0x80); /* if > 0 SDA will be a 1 */
    byte = byte << 1; /* send each bit, MSB first for address */
}
else
{
    WriteSDA(byte & 0x01); /* if > 0 SDA will be a 1 */
    byte = byte >> 1; /* send each bit, LSB first for data */
}

BitDelay();
SetSCLHigh();
BitDelay();
WriteSCL(0);
BitDelay();
}

/* now for an ack */

/* Master generates clock pulse for ACK */
WriteSDA(1); /* release SDA ... listen for ACK */
BitDelay();
SetSCLHigh(); /* ACK should be stable ... data not allowed to change when SCL is high */

/* SDA at 0 ?*/
error = (PINB & 0x01); /* ack didn't happen if bit 0 = 1 */

WriteSCL(0);
BitDelay();

return(error);
}

```

SendStartBit.c

/* SendStartBit.c generates an TWI start bit start bit is a 1 to 0 transition on SDA while SCL is high

```

      _____
      /
SCL  ___/

      _____
      \
SDA  \____
*/

```

```
#include "2wi.h"
```

```
extern void WriteSCL(unsigned char state);
extern void WriteSDA(unsigned char state);
```

```

extern void BitDelay(void);
extern unsigned char SetSCLHigh(void);

void SendStartBit(void)
{
    WriteSDA(1);
    BitDelay();
    SetSCLHigh();
    BitDelay();
    WriteSDA(0);
    BitDelay();
    WriteSCL(0);
    BitDelay();
}

```

SendStopBit.c

/* SendStopBit.c generates an TWI stop bit assumes SCL is low
stop bit is a 0 to 1 transition on SDA while SCL is high

```

      _____
     /
SCL  ___/

      _____
     /
SDA  _____/

```

*/

```
#include "2wi.h"
```

```

extern void WriteSCL(unsigned char state);
extern void WriteSDA(unsigned char state);
extern void BitDelay(void);
extern unsigned char SetSCLHigh(void);

```

```

void SendStopBit(void)
{
    WriteSDA(0);
    BitDelay();
    SetSCLHigh();
    BitDelay();
    WriteSDA(1);
    BitDelay();
}

```

SetSCLHigh.c

/* SetSCLHigh.c Det SCL high, and wait for it to go high. Returns the value of t0_timed_out. If 0xff then we
timed out before SCL went high and should be used to indicate an error to the caller

Crystal frequency is 7.3728 MHz */

```

#include "2wi.h"

extern volatile unsigned char t0_timed_out;

extern void WriteSCL(unsigned char state);

unsigned char SetSCLHigh(void)
{
    WriteSCL(1); /* release SCL*/

    /* set up timer counter 0 for timeout */
    t0_timed_out = FALSE; /* will be set after approximately 34 us */
    TCNT0 = 0; /* clear counter */
    TCCR0 = 1; /* ck/1 .. enable counting */

    /* wait till SCL goes to a 1 */
    while (!(PINB & 0x02) && !t0_timed_out)
        ;

    TCCR0 = 0; /* stop the counter clock */

    return(t0_timed_out);
}

```

Timer0.c

```

/* timer0 */

#include "2wi.h"

extern volatile unsigned char t0_timed_out;

interrupt [TIMER0_OVF0_vect] void TIMER0_OVF0_interrupt(void)
{
    t0_timed_out = TRUE;
}

```

Timer1.c

```

/* timer 1 timer */

#include "2wi.h"

extern volatile unsigned char t1_timed_out;

interrupt [TIMER1_OVF1_vect] void TIMER1_OVF1_interrupt(void)
{
    t1_timed_out = TRUE;
}

```

VerifyResetPolarity.c

```

/* VerifyResetPolarity.c 4 bytes are read from locations 0x20000H through 0x20003H for 512k/1M.
The bytes are verified to be of all the same value. If they are then the value is returned.
return value = 0xff = active low reset and active low output enable.
return value = 0x00 = active high reset and active high output enable.
If they aren't then 0xaa is returned to signal an error condition. */

```

```
#include "2wi.h"
```

```

extern void BitDelay(void);
extern unsigned char SendByte(unsigned char byte, unsigned char order);
extern unsigned char GetByte(unsigned char lastbyte);
extern void SendStartBit(void);
extern void SendStopBit(void);

```

```

unsigned char VerifyResetPolarity(void)
{
    unsigned char loc_1;
    unsigned char loc_2;
    unsigned char loc_3;
    unsigned char loc_4;
    unsigned char value;

    PORTB &= 0xf7; /* bring CS low */
    PORTB &= 0xef; /* bring RESET/OE low */
    PORTB &= 0xfb; /* bring SER_EN low */
    BitDelay(); /* for good measure */

    SendStartBit();
    SendByte(AT17 + WRITE,MSB_FIRST); /* send device address byte */
    SendByte(0x02,MSB_FIRST);          /* 1st address byte ... most significant byte first */
    SendByte(0x00,MSB_FIRST);          /* 2nd address byte ... most significant byte first */
    SendByte(0x00,MSB_FIRST);          /* 3rd address byte ... most significant byte first */

    SendStartBit();
    SendByte(AT17 + READ,MSB_FIRST); /* send device address byte with read */

    loc_1 = GetByte(0);
    loc_2 = GetByte(0);
    loc_3 = GetByte(0);
    loc_4 = GetByte(1);
    SendStopBit();

    PORTB |= 0x04; /* bring SER_EN high */
    PORTB |= 0x10; /* bring RESET/OE high */
    PORTB |= 0x08; /* bring CS high */

    if ((loc_1 == loc_2) && (loc_2 == loc_3) && (loc_3 == loc_4))
        value = loc_1; /* valid reset/oe polarity */
    else

```

```

    value = 0xaa; /* error */

    return(value);
}

```

WritePage.c

/* WritePage.c Writes PAGE_SIZE (128)bytes at address from bufptr Starts writing at address 0 within the page Please refer to the application note titled: "Programming Specification for Atmel's AT17 and AT17A series FPGA Configuration EEPROMS" found at www.atmel.com for detailed device address decoding and page address formatting After programming the data polling method is used to determine the end of the internal programming cycle. Timer/Counter1 is used to ensure the polling code exits its while loop. */

```

#include "2wi.h"

extern volatile unsigned char t1_timed_out;

extern unsigned char SendByte(unsigned char byte, unsigned char order);
extern void SendStartBit(void);
extern void SendStopBit(void);
extern void BitDelay(void);

void WritePage(unsigned int address, unsigned char *bufptr)
{
    unsigned char i;
    unsigned char addr1;
    unsigned char addr2;
    unsigned char addr3;
    unsigned char test_ack = 0xff;

    PORTB &= 0xf7; /* bring CS low */
    PORTB &= 0xef; /* bring RESET/OE low */
    PORTB &= 0xfb; /* bring SER_EN low */
    BitDelay();

    addr1 = (unsigned char)(address >> 9);
    addr2 = (unsigned char)(address >> 1);
    addr3 = (unsigned char)(address << 7);

    SendStartBit();
    SendByte(AT17 + WRITE,MSB_FIRST); /* send device address byte with write bit */
    SendByte(addr1,MSB_FIRST); /* 1st address byte */
    SendByte(addr2,MSB_FIRST); /* 2nd address byte */
    SendByte(addr3,MSB_FIRST); /* 3rd address byte */

    for (i = 0; i < PAGE_SIZE; i++)
        SendByte(bufptr[i],LSB_FIRST);

    SendStopBit();

    t1_timed_out = FALSE; /* set in timer counter 0 overflow interrupt routine */
}

```



```

/* 20 milli-second timeout */
/* 7.3728MHz / 1024 = 7200 Hz */
/* 7200 Hz = 138.8 us */
/* 20 ms / 138.8 us = 144.09 */
/* 65536 - 144 = 65392 = ff70 */
/* interrupt on ffff to 0000 transition */
TCNT1H = 0xff;
TCNT1L = 0x70; /* load counter */
TCCR1B = 0x05; /* timer/counter 1 clock / 1024 */

/* continue sending start bit and device address until we get an ack back */
/* data poll to program complete ... time out for error */
while (test_ack && !t1_timed_out)
{
    SendStartBit();
    test_ack = SendByte(AT17 + WRITE,MSB_FIRST); /* send device address byte */
}
TCCR1B = 0x00; /* disable timer/counter 1 clock */
SendStopBit();

PORTB |= 0x04; /* bring SER_EN high */
PORTB |= 0x10; /* bring RESET/OE high */
PORTB |= 0x08; /* bring CS high */
}

```

WriteSCL.c

```

/* WriteSCL.c */

#include "2wi.h"

void WriteSCL(unsigned char state)
{
    if (state)
        DDRB &= 0xfd; /* input ... pullup will pull high or Slave will drive low */
    else
        DDRB |= 0x02; /* output ... port latch will drive low */
}

```

WriteSDA.c

```

/* WriteSDA.c */

#include "2wi.h"

void WriteSDA(unsigned char state)
{
    if (state)
        DDRB &= 0xfe; /* input ... pullup will pull high or Slave will drive low */
    else
        DDRB |= 0x01; /* output ... port latch will drive low */
}

```

```
}

```

2wi.h

```
/* 2wi.h */

#include "io8515.h"
#include "ina90.h"
#pragma language=extended

#define MSB_FIRST          0xff
#define LSB_FIRST          0x00
#define READ                0x01
#define WRITE               0x00
#define AT17                0xa6
#define TRUE                0xff
#define FALSE               0x00
#define MAX_PAGES           1024 /*The total number of pages for 1M is 1024; for 512, it is 512 */
#define PAGE_SIZE           128 /*Page size = 128 bytes for 512k and 1M */

```

8515int.xlc

```
-! XLINK command file for AT90S8515. 512 bytes data address
    space and 8 Kbytes program address space. -!

-! Define CPU type (AVR) -!

-ca90

-! Define reset and interrupt vector segment, requires 28(dec) locations -!

-Z(CODE)INTVEC=0-1B

-! Define segments in flash memory -!

-Z(CODE)RCODE,CDATA0,CDATA1,CCSTR,SWITCH,FLASH,CODE=1C-1FFF

-! Define segments in RAM -!
-! The registers are in addresses 0-1F and memory mapped I/O in addresses 20-5F, built-in SRAM in
    addresses 60-25F. Data stack(CSTACK) size is 60 bytes(hex), return stack(RSTACK) size is 20 bytes(hex)-!

-Z(DATA)IDATA0,UDATA0,RSTACK+20,IDATA1,UDATA1,ECSTR,CSTACK+60=60-25F

-! Select reduced "printf" support to reduce library size.
    See the configuration section of the IAR C-compiler Users Guide concerning use of printf/sprintf. -!

-e_small_write=_formatted_write
-e_small_write_P=_formatted_write_P

-! Disable floating-point support in "scanf" to reduce library size.
    See the configuration section of the IAR C-compiler Users Guide concerning use of scanf/sscanf -!

```

```
-e_medium_read=_formatted_read
-e_medium_read_P=_formatted_read_P

    -! Suppress one warning which is not relevant for this processor -!

-w29

    -! Load the 'C' library -!

c11s
```



Atmel Headquarters

Corporate Headquarters

2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe

Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia

Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan

9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory

2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/ High Speed Converters/RF Datacom

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

e-mail

literature@atmel.com

Web Site

<http://www.atmel.com>

© Atmel Corporation 2003.

Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

ATMEL® and AVR® are the registered trademarks of Atmel.

Other terms and product names may be the trademarks of others.



Printed on recycled paper.