AT91M63200: I²C Drivers for AT24C512 Serial EEPROM

AT91 ARM® Thumb® Microcontrollers

Application Note

Introduction

The Inter Integrated Circuit Bus (I²C Bus) is a simple bi-directional 2-wire, Serial Data (SDA) and Serial Clock (SCL) Bus for inter-IC control.

The AT91M63200 does not have dedicated hardware to manage an I²C line, but because of its high processing speed, flexible Timer/Counters (TC) and efficient interrupt management, an effective software implementation can easily be done. This Application Note describes such an implementation compliant with ARM procedure call standard (APCS). It is intended for use with an AT24C512 serial EEPROM.

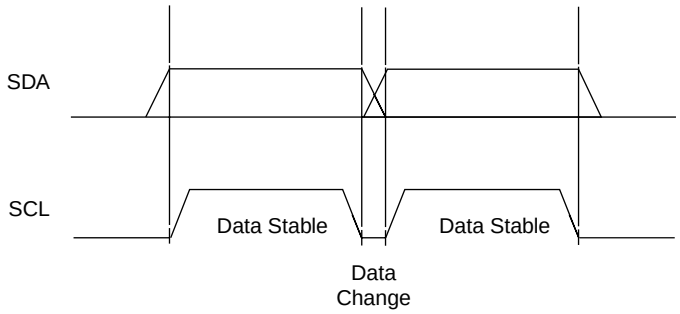
Theory of Operation

The industry-standard interface defined by I²C consists of the following signals:

- GND: common reference
- VCC: power supply
- SCL: serial clocks .The positive edge of the SCL input is used to clock data into the EEPROM device and the negative edge clocks data out of the device.
- SDA: serial data. The SDA pin is bi-directional for serial data transfer.

Waveforms

Figure 1. Clock and Data Transition:



Rev. 1742A–04/01

Figure 2. Start and Stop Condition

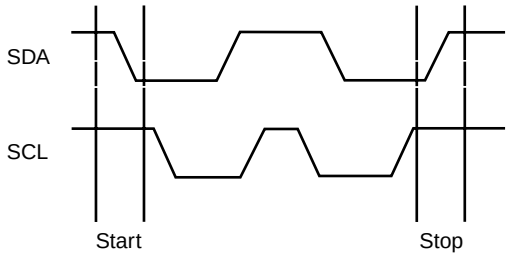


Figure 3. Acknowledge

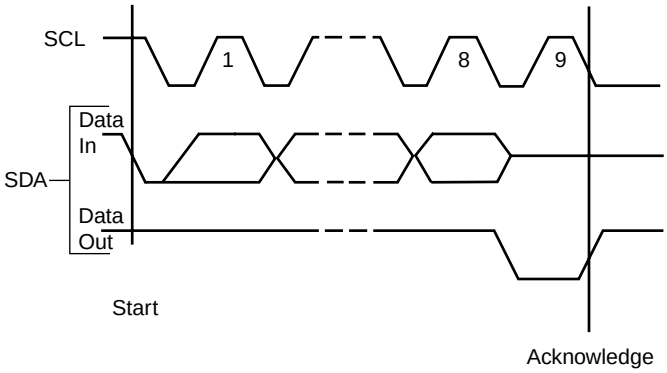


Figure 4. Write Operation

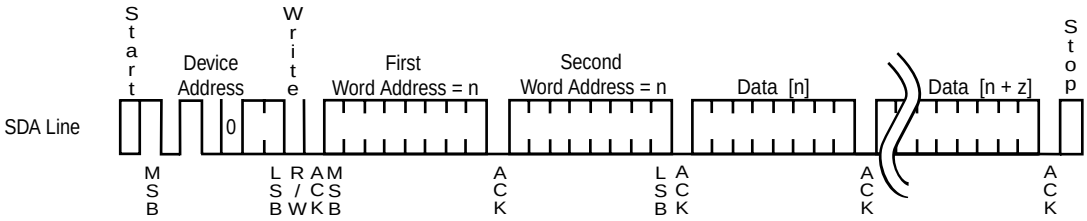
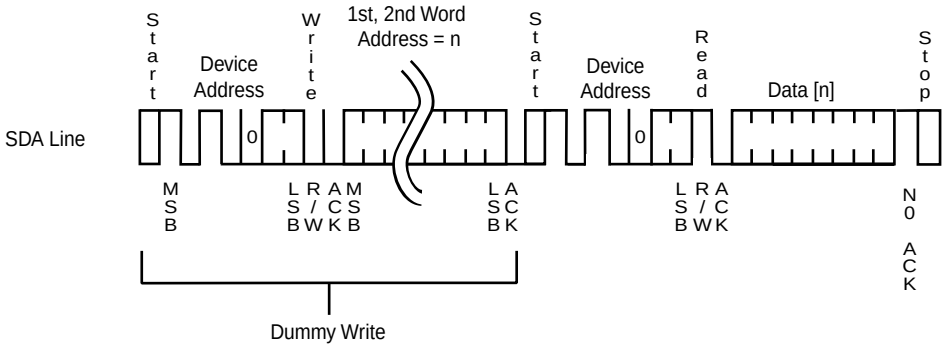


Figure 5. Read Operation



AT91M63200 Implementation

This application note describes how to implement a I²C master device using an AT91M63200 IC. A Timer/Counter channel and two PIO lines are used to manage the SCL and SDA lines.

The bytes from the EEPROM are stored in a reception buffer. The writable bytes come from a transmission buffer.

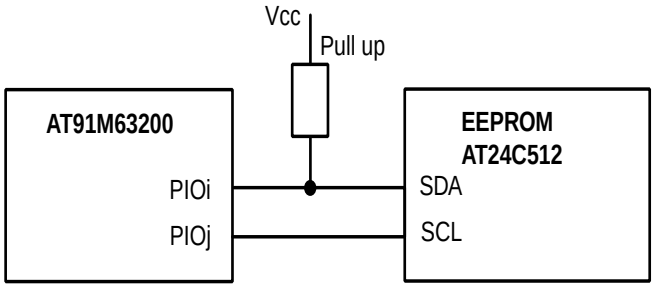
The Timer/Counter is clocked from its internal clock (MCK/2). The bit duration and sampling are generated by programing the register C (RC):

The RC is set with the bit time value. In read and write mode, the counter is reset at each RC Compare event.

Connection

At reset, all PIO pins of the AT91M63200 are programmed as inputs. Therefore, the SDA line must be connected to the supply voltage via a pull-up resistor. (Refer to the data sheet of the AT91M63200 to get more details concerning the value of the pull-up resistors.)

Figure 6. Hardware Connection



Source Files

The following source files are provided to implement the AT91M63200 I²C Driver:
Table 1.

Files	Contents
stdio.h	Standard C I/O definitions
lib_m63200.h	AT91M63200 Library include file
reg_m63200.h	AT91M63200 Registers include file
lib_i2c.h	I²C line types and constants
lib_i2c.c	Transmission/Reception interrupt handlers
test_i2c.c	I²C EEPROM drivers
i2c_irq.s	Assembler Interrupt handler

Peripheral Drivers

For each peripheral used (Timer/Counter, advanced interrupt controller, etc.), a structure is defined to provide easy access to the peripheral registers. Constants are also defined to provide easy access to the register fields. These peripherals and constants are defined in lib_m63200.h

Global Variables

Global variables relative to the I²C line are grouped in a structure (I2Cdesc), type defined in `lib_i2c.h`, a portion of which is reproduced below.

Extract From lib_i2c.h

This structure groups timer characteristics, PIO characteristics, buffer pointers, working variables and interrupt handler addresses (AIC handler and C handler).

```
typedef struct I2C
{
//* Buffers
     u_char *RxBase;
     u_char *RxPtr;
    u_char *RxEnd;
    u_char *TxPtr;
    u_char *TxEnd;

    //* work variables
    u_char deviceAddress;
    u_short loadAddress;
    u_char nb_ACK;
    u_int nbI2CByte;
    u_char mode;
    signed char countBit;
    u_char I2CByte;
    u_char  state;

    //* PIO field
    StructPIO *pioa_base;
    const PioCtrlDesc   *pio_ctrl_desc ;
    u_int SDA;
    u_int SCL;

    //* Timer Counter field
    StructTCBlock *TCBase;
    StructTC *timerBase;
    u_char channelId;

    //* IRQ field
    TypeAICHandler *AICHandler;
    void (*I2CTCHandler) (struct I2C *);
} I2Cdesc ;
```

Note: The test_i2c file defines a structure instance for the I²C line: I2C_line
The lib_i2c.c file uses a pointer to this structure (I2C_pt) to allow an access to these variables.

Interrupt Management

Read and write commands are implemented by interupts. The initialization relative to these interrupts (stack,etc.) is not described in this document, but initialization must be done before using functions defined in this application note.

- The Advanced Interrupt Controller (AIC) Source Vector Register (SVR) of the timer used is filled with an assembler interrupt handler.

  This assembler handler (defined in I2c_irq.s) executes the basic interrupt treatments (register saving and restoring, stack management, end of interrupt acknowledgment, etc.) The handler loads the variable structure corresponding to the I2C line descriptor (see Global Variables section) and call the `I2C_lineHandlerInt` C handler with this structure as a parameter.
- `I2C_lineHandlerInt` performs the basic operations to be done for all handlers:
  Read timer status to acknowledge interrupt.
  Call the C handler defined by the variable `I2CTCHandler` field. This handler will perform specific operations according to the machine state.

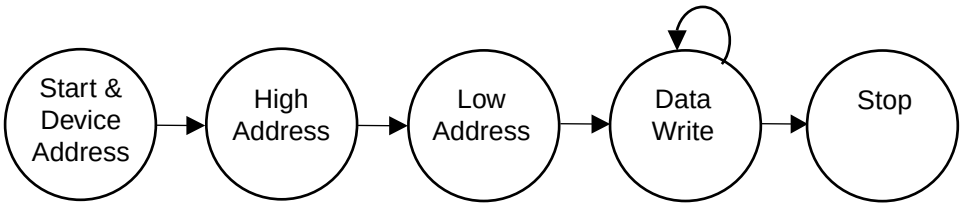
```
Begin
| Acknowledge interrupt status
| Call handler following the machine state
|End
```

I²C Drivers

The I²C drivers have been designed to support multi-byte sequential read and write operations.

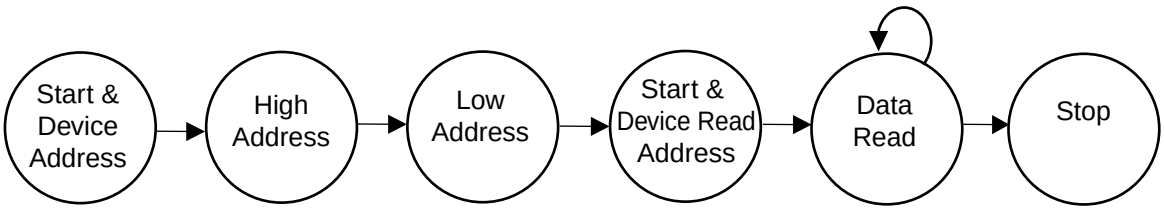
A write operation requires the device address word and acknowledgment, followed by two 8-bit data word addresses, followed by the data sequence. The sequence must begin with a start condition and terminate with a stop condition.

Figure 7. Write Sequence States



A read operation is initiated the same way as a write operation. It requires a dummy write operation in order to set the address. Then the sequence must follow with another start condition and a device address word. The sequence must terminate with a stop condition.

Figure 8. Read Seaquence States



State Machine Management

The file `lib_i2c` provides all the functions required for the management of the two following state machines.

Two counters (byte and bit) and several flags have been defined to manage the change between the different states.

Two functions perform the start of a read or write sequence by initializing the state machine:

`at91_I2CRead`

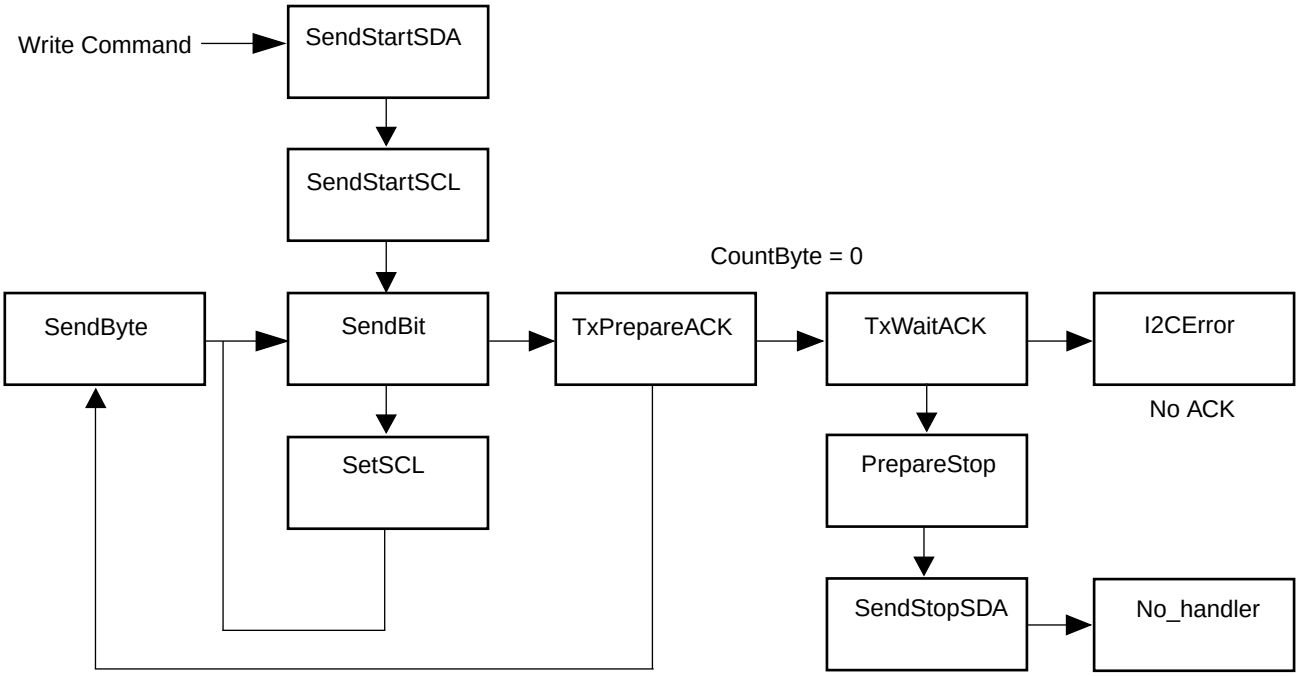
`at91_I2Cwrite`

The other functions are interrupt handlers. There is one interrupt handler for each state of the state machines.

All of these functions and handlers are described after the descriptions of the two state machines.

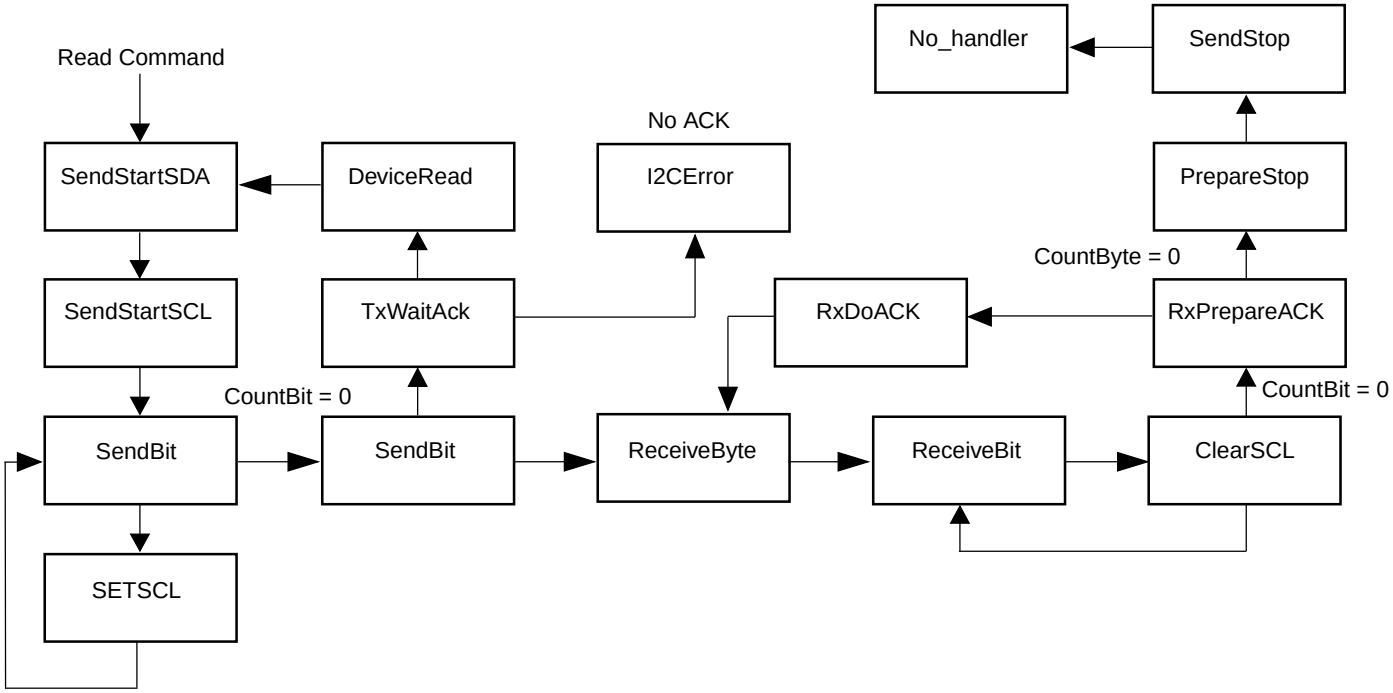
Write Sequence

Figure 9. Write Sequence State Machine



Read Sequence

Figure 10. Read Sequence State Machine



User Interface Layer

I2C_lineOpen

The `I2C_lineOpen` function configures the PIO lines and the Timer/Counter channel to be used.

```
Begin
| Set SDA and SCL as output
| Set SDA and SCL
| Open the clock of the timer
| Initialize the mode of the timer
| Initialize the RC Register
| Enable timer clock
End
```

I2C_lineClose

The `I2C_lineClose` function disables interrupts on the timer, allowing it to be used by another task.

```
Begin
| Disable all interrupts of the current timer
End
```

at91_I2CWrite

The `at91_I2CWrite` function initializes the write sequence by setting up the address of the slave device, the number of bytes to write and the address of the first byte. It sets the next mode flag at high_address_write value. It also enables the timer interrupt and changes the interrupt handler to `at91_I2CTxSendStartSDA`.

```
Begin
| Initialize state
| Set SDA and SCL as an output
| Set SDA and SCL
| Get device address
| Get address where to write
| Set next mode High_address_write
| Get byte counter
| Initialize bit counter
| Change IT handler to at91_I2CTxSendStartSDA
| Enable interrupt on RC Compare event
End
```

at91_12CRead

The `at91_12CRead` function initializes the read sequence by setting up the address of the slave device, the number of bytes to read and the address of the first byte. It sets the next mode flag at high_address_read value. It also enables the timer interrupt and changes the interrupt handler to `at91_I2CTxSendStartSDA.`

```
Begin
| Initialize state
| Set SDA and SCL as an output
| Set SDA and SCL
| Get device address
| Get address where to read
| Set next mode High_address_read
| Get byte counter
| Initialize bit counter
| Change IT handler to at91_I2CTxSendStartSDA
| Enable interrupt on RC Compare event
End
```

Interrupt Interface Layer

at91_I2CDeviceRead

The `at91_I2CDeviceRead` handler configures the read mode. It sets the next mode flag at data_read value and changes the interrupt handler to `at91_I2CTxSendStartSDA`.

```
Begin
| Set SDA and SCL as an output
| Set SDA and SCL
| Configure device address for read mode
| Set next mode Data_read
| Initialize bit counter
| Change IT handler to at91_I2CTxSendStartSDA
End
```

at91_I2CTxSendByte The `at91_I2CTxSendByte` handler configures the SDA line as an output at high level and, depending on the current mode flag, sets up the byte to write and the next mode flag. It changes the interrupt handler to `at91_I2CTxSendBit`.

```
Begin
| Set SDA and SCL as an output
| Set SDA and SCL
| | if mode == Data
| | | get byte from the buffer
| | | decrement counter byte
| | | if byte counter == 0
| | | | next mode = Stop
| | if mode == High_address_write
| | | get high address byte
| | | | next mode = Low_address_write
| | if mode == Low_address_write
| | | get low address byte
| | | | next mode = Low_address_write
| | if mode == High_address_read
| | | get high byte address
| | | | next mode = Low_address_read
| | if mode == Low_address_read
| | | get high byte address
| | | | next mode = Device_Read
| Initialize bit counter
| Change IT handler to at91_I2CTxSendBit
End
```

at91_I2CTxSendByte The `at91_I2CTxSendByte` handler configures the SDA line as an input, clears the SCL line and decrements the number of bytes to read. If there are no more bytes to read, it sets the next mode flag at stop value. It changes the interrupt handler to `at91_I2CrxReceiveBit`.

```
Begin
| Set SDA as an input
| Clear SCL
| Initialize bit counter
| Decrement byte counter
| Next mode = STOP if byte counter == 0
| Change IT handler to at91_I2CRxReceiveBit
End
```

at91_I2CTxSendBit

The at91_I2CTxSendBit handler clears the SCL line and decrements the number of bits to send. If all the bits have been sent, it sets the SDA line as an input for the acknowledgment and changes the interrupt handler to at91_I2CTxPrepareACK. Otherwise, it sets the SDA line depending on the value of the bit to write, and changes the interrupt handler to at91_I2CTxSetSCL.

```
Begin
| Clear SCL
| Decrement bit counter
| if bit counter != 0
| | Get bit to write
| | Change IT handler to at91_I2CTxSetSCL
| else
| | Set SDA as an input
| | Change IT handler to at91_I2CTxPrepareACK
End
```

at91_I2CRxReceiveBit

The at91_I2CRxReceiveBit handler sets SCL line and decrements the number of bits to read. If all the bits have been read, it stores the byte in the reception buffer. Otherwise, it builds the byte depending on the value of the bit. It changes the interrupt handler to at91_I2CTxClearSCL.

```
Begin
| Set SCL
| Decrement bit counter
| get bit
| if bit counter > 0
| | update working byte
| else
| | write byte in the buffer
| Change IT handler to at91_I2C_RxClearSCL
End
```

at91_I2CTxsendstart SDA

The at91_I2CTxSendStartSDA handler clears the SDA line and changes the interrupt handler to at91_I2CTxStartSCL.

```
Begin
| Clear SDA
| Change IT handler to at91_I2CTxStartSCL
End
```

at91_I2CTxSendStart SCL

The at91_I2CTxSendStartSCL handler clears the SCL line and changes the interrupt handler to at91_I2CTxSendBit.

```
Begin
| Clear SCL
| Change IT handler to at91_I2CTxSendBit
End
```

at91_I2CTxPrepare Stop

The at91_I2CTxPrepareStop handler sets the SDA line as an output, clears the SDA line and sets the SCL line. It changes the interrupt handler to at91_I2CTxSendStopSDA.

```
Begin
| Set SDA as an output
| Clear SDA
| Set SCL
| Change IT handler to at91_I2CTxSendStopSDA
End
```

at91_I2CTxSendStop SDA

The at91_I2CTxSendStopSDA handler sets the SDA line and releases the state of the line. It disables the interrupt and changes the handler to no_handler_tc.

```
Begin
| Set SDA
| Change state
| Disable RC compare interrupt
| Change IT handler to no_handler_tc
End
```

at91_I2CTxPrepare ACK

The at91_I2CTxPrepareACK handler sets the SCL line and changes the interrupt handler to at91_I2CTxWaitACK.

```
Begin
| Set SCL
| Change IT handler to at91_I2CTxWaitACK
End
```

at91_I2CTxWaitACK

The at91_I2CTxWaitACK handler checks the level of the SDA line. If the level is low, it clears the SCL and changes the interrupt handler, depending on the current mode flag . Otherwise, it decrements the error counter and changes the interrupt handler to at91_I2CError when the counter reaches 0.

```
Begin
| if SDA == 0
| | if mode == Stop
| | | Clear SCL
| | | Change IT handler to at91_I2CTxPrepareStop
| | if mode == Device_read
| | | Clear SCL
| | | Change IT handler to at91_I2CDeviceRead
| | if mode == Data_read
| | | Change IT handler to at91_I2CRxReceiveByte
| | else
| | | Clear SCL
| | | Change IT handler to at91_I2CtxSendByte
| else
| | decrement error counter
| | if error counter == 0
| | | Change IT handler to at91_I2CError
End
```

at91_I2CRxPrepare ACK

The at91_I2CRxPrepareACK handler sets the SCL line and changes the interrupt handler to at91_I2CTxPrepareStop, if the current mode flag is set with stop value. Otherwise, it changes the interrupt handler to at91_I2CRxDoACK.

```
Begin
| Set SCL
| If mode == STOP
| | Change IT handler to at91_I2CTxPrepareStop
| else
| | Change IT handler to at91_I2CRxDoACK
| End
```

at91_I2CRxDoACK

The at91_I2CRxDoACK handler clears the SCL line and changes the interrupt handler to at91_I2CRxReceiveByte.

```
Begin
| Clear SCL
| Change IT handler to at91_I2CRxReceiveByte
End
```

at91_I2CTxSetSCL

The at91_I2CTxSetSCL handler sets the SCL line and changes the interrupt handler to at91_I2CTxSendBit.

```
Begin
| Set SCL
| Change IT handler to at91_I2CTxSendBit
End
```

at91_I2CRxSetSCL

The at91_I2CRxSetSCL handler sets the SCL line and changes the interrupt handler to at91_I2CRxReceiveBit.

```
Begin
| Set SCL
| Change IT handler to at91_I2CRxReceiveBit
End
```

at91_I2CTxClearSCL

The at91_I2CTxClearSCL handler clears the SCL line, sets up the SDA line as an input if the bit counter is equal to 0 and changes the interrupt handler to at91_I2CTxSendBit.

```
Begin
| Clear SCL
| If bit counter == 0
| | Set SDA as an input
| Change IT handler to at91_I2CTxSendBit
End
```

at91_I2CRxClearSCL

The `at91_I2CRxClearSCL` handler clears the SCL line, sets up the SDA line as an output, clears the SDA line and changes the interrupt handler to `at91_I2CRxPrepareACK` if the bit counter is equal to 0. Otherwise, it changes the interrupt handler to `at91_I2CRxReceiveBit`.

```
Begin
| Clear SCL
| If bit counter == 0
| | Change IT handler to at91_I2CRxPrepareACK
| | Set SDA as an output
| | Clear SDA
| else
| | Change IT handler to at91_I2CRxReceiveBit
End
```

at91_I2CError

The **at91_I2CError** handler implements a forever loop.

```
Begin
| forever loop
End
```

EEPROM Drivers

The file: test_i2c.c, implements the management of the EEPROM. It consists of:

- A driver for write access
- A driver for read access

NVMWrite

The `NVMWrite` function has three arguments :

1. number of bytes to write
2. first byte load address
3. address of the buffer containing the bytes to write

```
Begin
| Set number of page to program
| for each page
| | set the pointer at the beginning of the buffer to program
| | call at91_I2Cwrite function
| | wait the end of programmation of the page
| if number of byte to program < number of byte per page
| | set the pointer at the beginning of the buffer to program
| | call at91_I2CWrite function
| | wait the end of programmation of the bytes
End
```

NVMRead

The NVMRead function has three arguments :

1. number of byte to read
2. first byte read address
3. address of the storing buffer

```
Begin
| set the pointer at the beginning of the buffer
| call the function at91_I2CRead with the number of byte and the
| address
End
```

Code

test_i2c.c

```
//*----------------------------------------------------------------------------
//*      ATMEL Microcontroller Software Support  -  ROUSSET  -
//*----------------------------------------------------------------------------
//* The software is delivered "AS IS" without warranty or condition of any
//* kind, either express, implied or statutory. This includes without
//* limitation any warranty or condition with respect to merchantability or
//* fitness for any particular purpose, or against the infringements of
//* intellectual property rights of others.
//*----------------------------------------------------------------------------
//* File Name           : test_i2c.c
//* Object              : I2C
//*
//* 1.0 29/05/00  EL    : Creation
//*----------------------------------------------------------------------------

#include     <stdio.h>
#include    "drivers/lib_i2c/lib_i2c.h"
#include    "parts/m63200/lib_m63200.h"
#include    "parts/m63200/reg_m63200.h"


#define PAGE_LENGTH 128
#define NB_PAGE 4096
#define EPROM_SIZE  NB_PAGE*PAGE_LENGTH
#define NB_PAGE_TEST 10

extern void tc0_interrupt_handler(void);


void no_handler_tc (I2Cdesc *I2C_pt)
{
}

void at91_I2CError(I2Cdesc *I2C_pt)
{
    printf("I2C EPROM ERROR !\n");
    while(1);
}

I2Cdesc I2C_line;

u_char txBuffer[128];
u_char rxBuffer[128];
u_int i;
int line_status = OK;
```

```
u_char NVMWrite(u_short address, u_char *data_array, u_int num_bytes)
{
    u_short num_page = num_bytes / PAGE_LENGTH;
    u_short cpt;
    u_int j;


  for (cpt=0; cpt<num_page; cpt++)
  {
      I2C_line.TxEnd = data_array;
      at91_I2CWrite(&I2C_line,address+cpt*PAGE_LENGTH,PAGE_LENGTH);
      while(I2C_line.state != OK);
      for(j=0;j<20000;j++);
  }

  if((num_page*PAGE_LENGTH) < num_bytes)
  {
      I2C_line.TxEnd = data_array;
      at91_I2CWrite(&I2C_line,address+cpt*PAGE_LENGTH,
    (num_bytes-PAGE_LENGTH*cpt));
      while(I2C_line.state != OK);
      for(j=0;j<20000;j++);
  }
    return(1);
}


u_char NVMRead(u_short address, u_char *data_array, u_int num_bytes)
{
     I2C_line.RxEnd = data_array;

     at91_I2CRead(&I2C_line,address,num_bytes);
     return(1);
}

int main(void)
{
 u_char j;

  //* PIO field
  I2C_line.pio_ctrl_desc = &PIOA_DESC;
  I2C_line.pioa_base = PIOA_BASE;
        I2C_line.SDA = PA21;
        I2C_line.SCL = PA20;

  // EPROM AT24C512 address
  I2C_line.deviceAddress = 0;

  //* Timer Counter field
  I2C_line.TCBase = TCB0_BASE;
```

```
    I2C_line.timerBase = (StructTC *) TCB0_BASE;
    I2C_line.channelId = TC0_ID;

    //* IRQ field
    //* disable interrupt
    I2C_line.timerBase->TC_IDR = 0x1FF;
    I2C_line.AICHandler = tc0_interrupt_handler;
    I2C_line.I2CTCHandler = no_handler_tc;

    //* Buffers
    I2C_line.RxBase = rxBuffer;
    I2C_line.RxPtr  = rxBuffer;
    I2C_line.RxEnd  = rxBuffer;
    I2C_line.TxPtr  = txBuffer;
    I2C_line.TxEnd  = txBuffer;

    //* 220 = 28Khz MAX
    I2C_lineOpen(&I2C_line, 250);

    //* Trig the timer
    I2C_line.timerBase->TC_CCR = TC_SWTRG;

    for(i=0;i<128;i++)
        txBuffer[i] =  (u_char) i;

    printf("I2C EPROM TEST\n");
    printf("WRITING EPROM %d pages\n",NB_PAGE_TEST);

         NVMWrite(0,txBuffer,PAGE_LENGTH*NB_PAGE_TEST);
    printf("READING EPROM\n");
    for(i=0; i<NB_PAGE_TEST; i++)
    {
        NVMRead(i*PAGE_LENGTH,rxBuffer,PAGE_LENGTH);
        while(I2C_line.state != OK);
        I2C_line.RxEnd = I2C_line.RxPtr;
        I2C_line.TxEnd = I2C_line.TxPtr;
        for(j = 0; j<PAGE_LENGTH; j++)
        {
      if( *I2C_line.RxEnd++ != *I2C_line.TxEnd++)
      {
          printf("Error page %d\n",i);
          printf("TEST EPROM FAILED\n");
          while(1);
      }
        }
    }
    printf("%d pages : OK\n", NB_PAGE_TEST);
    printf("TEST EPROM OK\n");
    while(1);
}
```

i2c_irq.s

```
;-------------------------------------------------------------------------------
;-  ATMEL Microcontroller Software Support  -   ROUSSET -
;-------------------------------------------------------------------------------
;- File source: i2c_irq.s
;- Librarian: Not applicable
;- Translator: ARM Software Development Toolkit V2.11
;-
;- Treatment: Assembler Timer Counter Interrupt Handler.
;- Imported Resources:
;-    I2C_lineHandlerInt
;-    i2C_line
;-
;- Exported Resources:
;-    tc0_interrupt_handler
;-
;- 1.0
;-
;-------------------------------------------------------------------------------
; Function :
; The handlers for each Timer Counter are stored in the table <TCHandlerTable>.
; When an interrupt occurs, the core branches the channel corresponding
; assembler handler which manage the system ( core and AIC ) and branch to
; the corresponding C handler. This last can be defined by the C function
; <init_timer_irq>.
; Note in thumb mode, handler addresses are saved with bit 0 set. Then,
; you don't need to set it before execution of bx.
;-------------------------------------------------------------------------------
      AREA    AT91Lib, CODE, READONLY, INTERWORK


            INCLUDE     periph/aic/aic.inc
            INCLUDE     periph/arm7tdmi/arm.inc

            IMPORT      PtAICBase
       EXPORTtc0_interrupt_handler
      IMPORTI2C_line
      IMPORTI2C_lineHandlerInt


tc0_interrupt_handler

;- Adjust and save LR_irq in IRQ stack
            sub         r14, r14, #4
            stmfd       sp!, {r14}

;- Write in the IVR to support Protect Mode
;- No effect in Normal Mode
;- De-assert the NIRQ and clear the source in Protect Mode
            ldr         r14, =AIC_BASE
            str         r14, [r14, #AIC_IVR]
```

```
;- Save SPSR and r0 in IRQ stack
            mrs         r14, SPSR
            stmfd       sp!, {r0, r14}

;- Enable Interrupt and Switch in SYS Mode
            mrs         r0, CPSR
            bic         r0, r0, #I_BIT
            orr         r0, r0, #ARM_MODE_SYS
            msr         CPSR_c, r0

;- Save scratch/used registers and LR in User Stack
            stmfd       sp!, { r1-r3, r12, r14}


      ldr   r0,=I2C_line
;- Load the address of the Timer Counter Handler Table
      ldr   r1, =I2C_lineHandlerInt

      mov   r14, pc
      bx    r1
;- Restore scratch/used registers and LR from User Stack
            ldmia       sp!, { r1-r3, r12, r14}

;- Disable Interrupt and switch back in IRQ mode
            mrs         r0, CPSR
            bic         r0, r0, #ARM_MODE_SYS
            orr         r0, r0, #I_BIT:OR:ARM_MODE_IRQ
            msr         CPSR_c, r0

;- Mark the End of Interrupt on the AIC
            ldr         r0, =AIC_BASE
            str         r0, [r0, #AIC_EOICR]

;- Restore SPSR_irq and r0 from IRQ stack
            ldmia       sp!, {r0, r14}
            msr         SPSR_cxsf, r14

;- Restore adjusted  LR_irq from IRQ stack directly in the PC
            ldmia       sp!, {pc}^


      END


END
```

lib_i2c.c

```
//*----------------------------------------------------------------------------
//*      ATMEL Microcontroller Software Support  -  ROUSSET  -
//*----------------------------------------------------------------------------
//* The software is delivered "AS IS" without warranty or condition of any
//* kind, either express, implied or statutory. This includes without
//* limitation any warranty or condition with respect to merchantability or
//* fitness for any particular purpose, or against the infringements of
//* intellectual property rights of others.
//*----------------------------------------------------------------------------
//* File Name           : lib_i2c.c
//* Object              : I2C driver functions
//*
//* 1.0 29/05/00  EL    : Creation
//* 2.0 09/07/00  EL    : Clean up
//*----------------------------------------------------------------------------

#include    "periph/timer_counter/lib_tc.h"
#include    "lib_i2c.h"
#include    "periph/power_saving/lib_power_save.h"



extern void no_handler_tc(I2Cdesc *I2C_pt);
extern void at91_I2CError(I2Cdesc *I2C_pt);

//*----------------------------------------------------------------------------
//* Function Name       : at91_I2CWrite (I2Cdesc *I2C_pt, u_short loadAddress,
//*u_char nbByte)
//* Object              : I2C
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//*   : loadAddress
//*   : nbByte
//* Output Parameters   : none
//*----------------------------------------------------------------------------
void at91_I2CWrite (I2Cdesc *I2C_pt, u_short loadAddress, u_int nbByte)
//* Begin
{
    //* initialize state
    I2C_pt->state = NOK;

    //* set SDA line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

    //* set SCL line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SCL;

    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* set SDA line
```

```
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;

    //* get byte to transmit
    I2C_pt->I2CByte = I2C_pt->deviceAddress | WRITE_BIT | 0xA0;

    //* get address
    I2C_pt->loadAddress = loadAddress;

    //* next mode
    I2C_pt->mode = HIGH_ADDRESS_WRITE;

    //* get nb byte to transmit
    I2C_pt->nbI2CByte = nbByte;

    //* initialize countBit
    I2C_pt->countBit = 8;

    //* initialize nb_ACK
    I2C_pt->nb_ACK = 10;

    //* change interrupt handler to at91_I2CTxSendStartSDA
    I2C_pt->I2CTCHandler = at91_I2CTxSendStartSDA;

     //* Enable RC compare interrupt
    I2C_pt->timerBase->TC_IER = TC_CPCS;

//* End
}
//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CRead (I2Cdesc *I2C_pt, u_short loadAddress,
// *u_char nbByte)
//* Object              : I2C
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//*   : loadAddress
//*   : nbByte

//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CRead (I2Cdesc *I2C_pt, u_short loadAddress, u_int nbByte)
//* Begin
{

    //* initialize state
    I2C_pt->state = NOK;

    //* set SDA line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

    //* set SCL line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SCL;
```

```
    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* set SDA line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;

    //* get byte to read
    I2C_pt->I2CByte = I2C_pt->deviceAddress | WRITE_BIT | 0xA0;

    //* get address
    I2C_pt->loadAddress = loadAddress;

    //* next mode
    I2C_pt->mode = HIGH_ADDRESS_READ;

    //* get nb byte to transmit
    I2C_pt->nbI2CByte = nbByte;

    //* initialize countBit
    I2C_pt->countBit = 8;

    //* initialize nb_ACK
    I2C_pt->nb_ACK = 10;

    //* change interrupt handler to at91_I2CTxSendStartSDA
    I2C_pt->I2CTCHandler = at91_I2CTxSendStartSDA;

     //* Enable RC compare interrupt
    I2C_pt->timerBase->TC_IER = TC_CPCS;

//* End
}

//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2C (I2Cdesc *I2C_pt, u_short loadAddress,
//* u_char nbByte)
//* Object              : I2C
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//*   : loadAddress
//*   : nbByte
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CDeviceRead (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SDA line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

    //* set SDA line
```

```
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;

    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* get byte to transmit
    I2C_pt->I2CByte = I2C_pt->deviceAddress | READ_BIT | 0xA0;

    //* next mode
    I2C_pt->mode = DATA_READ;

    //* initialize countBit
    I2C_pt->countBit = 8;

    //* initialize nb_ACK
    I2C_pt->nb_ACK = 10;

    //* change interrupt handler to at91_I2CTxSendStartSDA
    I2C_pt->I2CTCHandler = at91_I2CTxSendStartSDA;

//* End

}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSendByte (I2Cdesc *I2C_pt)
//* Object              : I2C
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxSendByte (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SDA line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

    //* set SDA line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;


    //* get byte to transmit
    switch(I2C_pt->mode)
    {
  case DATA :
      I2C_pt->I2CByte = *(I2C_pt->TxEnd++);

      I2C_pt->nbI2CByte--;
      if(I2C_pt->nbI2CByte == 0)
          //* next mode
```

```
                I2C_pt->mode = STOP;
        break;

    case HIGH_ADDRESS_WRITE :
        I2C_pt->I2CByte = (u_char) (I2C_pt->loadAddress >> 8);

              //* next mode
              I2C_pt->mode = LOW_ADDRESS_WRITE;
        break;

    case LOW_ADDRESS_WRITE :
        I2C_pt->I2CByte = (u_char) I2C_pt->loadAddress;

        //* next mode
        I2C_pt->mode = DATA;
        break;

    case HIGH_ADDRESS_READ :
        I2C_pt->I2CByte = (u_char) (I2C_pt->loadAddress >> 8);

              //* next mode
              I2C_pt->mode = LOW_ADDRESS_READ;
        break;

    case LOW_ADDRESS_READ :
        I2C_pt->I2CByte = (u_char) I2C_pt->loadAddress;

        //* next mode
        I2C_pt->mode = DEVICE_READ;
        break;
      }

      //* initialize countBit
      I2C_pt->countBit = 8;

      //* initialize nb_ACK
      I2C_pt->nb_ACK = 10;

      //* change interrupt handler to at91_I2CTxSendBit
      I2C_pt->I2CTCHandler = at91_I2CTxSendBit;


//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSendBit (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
```

```
//* Output Parameters   : none
//*----------------------------------------------------------------------------
void at91_I2CTxSendBit (I2Cdesc *I2C_pt)
//* Begin
{
    u_char bit_val;

     //* Clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    //* decrement countBit
    I2C_pt->countBit--;

    //* if bit counter = 0
    if(I2C_pt->countBit >= 0)
    {
  //* get bit to transfert
  bit_val = (I2C_pt->I2CByte >> I2C_pt->countBit) & 1;

  //* Send 0 or 1
  if (bit_val)
       //* Set SDA line
      I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;

        else
      //* Clear SDA line
      I2C_pt->pioa_base->PIO_CODR = I2C_pt->SDA;

  //* change interrupt handler to at91_I2CTxSetSCL
  I2C_pt->I2CTCHandler = at91_I2CTxSetSCL;
    }
    else
    {
   //* set SDA line as an input
  I2C_pt->pioa_base->PIO_ODR = I2C_pt->SDA;

  //* change interrupt handler to at91_I2CTxPrepareACK
  I2C_pt->I2CTCHandler = at91_I2CTxPrepareACK;
    }
//* End
}


//*----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSendStartSDA (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*----------------------------------------------------------------------------
void at91_I2CTxSendStartSDA (I2Cdesc *I2C_pt)
```

```
//* Begin
{
    //* clear SDA line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SDA;

    //* change interrupt handler to at91_I2CTxSendStartSCL
    I2C_pt->I2CTCHandler = at91_I2CTxSendStartSCL;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSendStart (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxSendStartSCL (I2Cdesc *I2C_pt)
//* Begin
{
    //* clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    //* change interrupt handler to at91_I2CTxSendBit
    I2C_pt->I2CTCHandler = at91_I2CTxSendBit;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxPrepareSTOP (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxPrepareSTOP (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SDA line as an output

    I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

    //* clear SDA line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SDA;

    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;
```

```
    //* change interrupt handler to at91_I2CTxSendStopSDA
    I2C_pt->I2CTCHandler = at91_I2CTxSendStopSDA;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSendStopSCL (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxSendStopSCL (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* change interrupt handler to at91_I2CTxSendStopSDA
    I2C_pt->I2CTCHandler = at91_I2CTxSendStopSDA;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSendStopSDA (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxSendStopSDA (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SDA line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;

    I2C_pt->state = OK;

    //* Disable RC compare interrupt
    I2C_pt->timerBase->TC_IDR = TC_CPCS;

    //* change interrupt handler to
    I2C_pt->I2CTCHandler = no_handler_tc;


//* End
}
```

```
//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxPrepareACK (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxPrepareACK (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* change interrupt handler to at91_I2CTxWaitACK
    I2C_pt->I2CTCHandler = at91_I2CTxWaitACK;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CRxPrepareACK (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------

void at91_I2CRxPrepareACK (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    if(I2C_pt->mode == STOP)
        //* change interrupt handler to
        I2C_pt->I2CTCHandler = at91_I2CTxPrepareSTOP;

    else
        //* change interrupt handler to at91_I2CRxDoACK
        I2C_pt->I2CTCHandler = at91_I2CRxDoACK;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxWaitACK (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
```

```
//*-----------------------------------------------------------------------------
void at91_I2CTxWaitACK (I2Cdesc *I2C_pt)
//* Begin
{
    //* if ACK (SDA = 0)
    if((I2C_pt->pioa_base->PIO_PDSR & I2C_pt->SDA) == 0)
    {

  switch(I2C_pt->mode)
  {
      case STOP :
    //* clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

          //* change interrupt handler to
          I2C_pt->I2CTCHandler = at91_I2CTxPrepareSTOP;
    break;

      case DEVICE_READ :
    //* clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    //* change interrupt handler to
          I2C_pt->I2CTCHandler = at91_I2CDeviceRead;
    break;


      case DATA_READ :
    //* change interrupt handler to
          I2C_pt->I2CTCHandler = at91_I2CRxReceiveByte;
    break;

      default :
    //* clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    //* change interrupt handler to
    I2C_pt->I2CTCHandler = at91_I2CTxSendByte;
    break;
  }
    }
    else
    {
        I2C_pt->nb_ACK--;
  if(I2C_pt->nb_ACK == 0)
      //* change interrupt handler to at91_I2C
      I2C_pt->I2CTCHandler = at91_I2CError;
   }
//* End
}
```

```
//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CRxDoACK (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CRxDoACK (I2Cdesc *I2C_pt)
//* Begin
{

    //* clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    //* change interrupt handler to
    I2C_pt->I2CTCHandler = at91_I2CRxReceiveByte;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxSetSCL (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxSetSCL (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* change interrupt handler to at91_I2CTxSendBit
     I2C_pt->I2CTCHandler = at91_I2CTxSendBit;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CTxClearSCL (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CTxClearSCL (I2Cdesc *I2C_pt)
//* Begin
{
```

```
    //* Clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    if(I2C_pt->countBit == 0)
    //* set SDA line as an input
  I2C_pt->pioa_base->PIO_ODR = I2C_pt->SDA;

    //* change interrupt handler to at91_I2CTxSendBit
    I2C_pt->I2CTCHandler = at91_I2CTxSendBit;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CRxSetSCL (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CRxSetSCL (I2Cdesc *I2C_pt)
//* Begin
{
    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

    //* change interrupt handler to at91_I2CRxReceiveBit
    I2C_pt->I2CTCHandler = at91_I2CRxReceiveBit;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CRxClearSCL (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CRxClearSCL (I2Cdesc *I2C_pt)
//* Begin
{
    //* Clear SCL line
    I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

    if(I2C_pt->countBit == 0)

{
      //* change interrupt handler to at91_I2CI2CRxPrepareACK
      I2C_pt->I2CTCHandler = at91_I2CRxPrepareACK;
```

```
            //* set SDA line as an output
        I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

        //* clear SDA line
        I2C_pt->pioa_base->PIO_CODR = I2C_pt->SDA;
        }
      else
    //* change interrupt handler to at91_I2CRx
    I2C_pt->I2CTCHandler = at91_I2CRxReceiveBit;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : at91_I2CRxReceiveByte (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void at91_I2CRxReceiveByte (I2Cdesc *I2C_pt)
//* Begin
{
  //* set SDA line as an input
  I2C_pt->pioa_base->PIO_ODR = I2C_pt->SDA;

  //* clear SCL line
  I2C_pt->pioa_base->PIO_CODR = I2C_pt->SCL;

  //* initialize countBit
  I2C_pt->countBit = 8;

        I2C_pt->nbI2CByte--;
        if(I2C_pt->nbI2CByte == 0)
            //* next mode
            I2C_pt->mode = STOP;

  //* change interrupt handler to at91_I2C
  I2C_pt->I2CTCHandler = at91_I2CRxReceiveBit;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : u_char at91_I2CRxReceiveBit (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
```

```
//*-----------------------------------------------------------------------------
void at91_I2CRxReceiveBit (I2Cdesc *I2C_pt)
//* Begin
{
  u_char bit_val;

        //* set SCL line
        I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;

  //* decrement countBit
  I2C_pt->countBit--;

  //* get bit
  if((I2C_pt->pioa_base->PIO_PDSR & I2C_pt->SDA) == I2C_pt->SDA)
      bit_val = 1;
  else
      bit_val = 0;

  //* if bit counter > 0
  if(I2C_pt->countBit > 0)
      //* update working byte
      *I2C_pt->RxEnd |= (bit_val << I2C_pt->countBit);

  else
       *(I2C_pt->RxEnd++) |= (bit_val << I2C_pt->countBit) ;

    //* change interrupt handler to at91_I2CRxClearSCL
    I2C_pt->I2CTCHandler = at91_I2CRxClearSCL;

//* End
}


//*-----------------------------------------------------------------------------
//* Function Name       : u_char at91_I2CRxReceiveBit (I2Cdesc *I2C_pt)
//* Object              :
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void I2C_lineHandlerInt (I2Cdesc *I2C_pt)
{
u_char dummy;

    //* acknowledge interrupt status
    dummy = I2C_pt->timerBase->TC_SR;

    //* call automate state handler*/
    (*(I2C_pt->I2CTCHandler))(I2C_pt);

}
```

```
//*-----------------------------------------------------------------------------
//* Function Name       :  (I2Cdesc *I2C_pt)
//* Object              : I2C
//* Input Parameters    : <I2C_pt> = I2C Peripheral Descriptor pointer
//* Output Parameters   : none
//*-----------------------------------------------------------------------------
void I2C_lineOpen (I2Cdesc *I2C_pt, u_int RCValue)
//* Begin
{
    //* line
    I2C_pt->state = OK;

    //* set SDA line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SDA;

    //* set SDA line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SDA;

    //* set SCL line as an output
    I2C_pt->pioa_base->PIO_OER = I2C_pt->SCL;

    //* set SCL line
    I2C_pt->pioa_base->PIO_SODR = I2C_pt->SCL;


    at91_clock_open(I2C_pt->pio_ctrl_desc->periph_id);

    //* TIMER configuration
    //* Set the mode of the timer
    //* Open the clock of the timer
    at91_clock_open(I2C_pt->channelId);

    I2C_pt->timerBase->TC_CMR = 0x0000C400;
    I2C_pt->timerBase->TC_RC = RCValue;

    at91_irq_open(I2C_pt->channelId, 7, AIC_SRCTYPE_INT_EDGE_TRIGGERED, I2C_pt-
>AICHandler);

    //* Enable the clock
    I2C_pt->timerBase->TC_CCR = TC_CLKEN;


//* End
}
```

lib_i2c.h

```
//*----------------------------------------------------------------------------
//*         ATMEL Microcontroller Software Support  -  ROUSSET  -
//*----------------------------------------------------------------------------
//* The software is delivered "AS IS" without warranty or condition of any
//* kind, either express, implied or statutory. This includes without
//* limitation any warranty or condition with respect to merchantability or
//* fitness for any particular purpose, or against the infringements of
//* intellectual property rights of others.
//*----------------------------------------------------------------------------
//* File Name           : lib_i2c.h
//* Object              : I2C driver Function Prototyping File.
//*
//* 1.0 25/05/00 EL     : Creation
//*----------------------------------------------------------------------------

#ifndef lib_i2c_h
#define lib_i2c_h

#include    "periph/pio/lib_pio.h"

#include    "periph/timer_counter/lib_tc.h"

#define I2C_MASTER 0
#define I2C_SLAVE  1

#define WRITE_BIT    0
#define READ_BIT     1
#define STOP       2
#define DATA         3
#define LOW_ADDRESS_WRITE  4
#define HIGH_ADDRESS_WRITE 5
#define LOW_ADDRESS_READ   6
#define HIGH_ADDRESS_READ  7
#define DEVICE_READ    8
#define DATA_READ    9

#define OK  0
#define NOK 1

typedef struct I2C
{
   //* Buffers
    u_char *RxBase;
    u_char *RxPtr;
    u_char *RxEnd;
    u_char *TxPtr;
    u_char *TxEnd;

    //* work variables
    u_char deviceAddress;
```

```
    u_short loadAddress;
    u_char nb_ACK;
    u_int nbI2CByte;
    u_char mode;
    signed char countBit;
    u_char I2CByte;
    u_char  state;

    //* PIO field
    StructPIO *pioa_base;
    const PioCtrlDesc   *pio_ctrl_desc ;
    u_int SDA;
    u_int SCL;

    //* Timer Counter field
    StructTCBlock *TCBase;
    StructTC *timerBase;
    u_char channelId;

    //* IRQ field
    TypeAICHandler *AICHandler;
    void (*I2CTCHandler) (struct I2C *);


} I2Cdesc ;

extern void  I2C_lineHandlerInt(I2Cdesc *I2C_pt);
extern void  I2C_lineOpen(I2Cdesc *I2C_pt, u_int RCValue);

extern void at91_I2CWrite (I2Cdesc *I2C_pt, u_short loadAddress, u_int nbByte);
extern void at91_I2CRead (I2Cdesc *I2C_pt, u_short loadAddress, u_int nbByte);

extern void  at91_I2CTxSendByte (I2Cdesc *I2C_pt);
extern void  at91_I2CTxSendDeviceAddress (I2Cdesc *I2C_pt);
extern void  at91_I2CTxSendBit (I2Cdesc *I2C_pt);
extern void  at91_I2CRxReceiveByte (I2Cdesc *I2C_pt);
extern void  at91_I2CRxReceiveBit (I2Cdesc *I2C_pt);

extern void  at91_I2CTxSendStartSDA (I2Cdesc *I2C_pt);
extern void  at91_I2CTxSendStartSCL (I2Cdesc *I2C_pt);
extern void  at91_I2CTxPrepareSTOP (I2Cdesc *I2C_pt);
extern void  at91_I2CTxSendStopSCL (I2Cdesc *I2C_pt);
extern void  at91_I2CTxSendStopSDA (I2Cdesc *I2C_pt);
extern void  at91_I2CTxPrepareACK (I2Cdesc *I2C_pt);
extern void  at91_I2CTxWaitACK (I2Cdesc *I2C_pt);
extern void  at91_I2CRxDoACK (I2Cdesc *I2C_pt);

extern void  at91_I2CTxSetSCL (I2Cdesc *I2C_pt);
extern void  at91_I2CTxClearSCL (I2Cdesc *I2C_pt);
extern void  at91_I2CRxSetSCL (I2Cdesc *I2C_pt);
```

```
extern void  at91_I2CRxClearSCL (I2CdescC *I2C_pt);


#endif /* lib_i2c_h */
```

Atmel Headquarters

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

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

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

Japan
Atmel Japan K.K.
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

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

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-357-000
FAX (44) 1355-242-743

Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

1742A 03/01