Application Note 55

Title: **EM68xx as I2C Master device**

Product Family: **8-bit Microcontroller**

Part Number: EM6812
Keywords: Microcontroller, i2C
Date: March 6, 2006

Introduction

This application note describes how to implementation software I2C protocol with EM68xx. No hardware I2C peripheral is implemented, so software must be used.
The EM68xx used as the master of the I2C bus cover the following functions:

- Start/stop/acknowledge bit management
- Transmit/Receive byte with error detection and I2C statuts register
- Selectable speed based on CPU speed and can be locally trimmed with a programmable delay

The Application describes an example of I2C bus shared with I2C DAC and I2C E2prom memory. This example covers the following functions:

- Write or Read E2prom (random byte mode or random page mode based on buffer size)
- Write DAC

The software example can be easily used or adapted for a specific application such as addressing other I2C devices, changing the EM68xx I2C pins location, buffer size, changing communication speed …

Communication

Hardware

An i2C bus is composed of 2 lines:

- Serial Clock (SCK) – generated by master only
- Serial Data In/Out (SDA)

These 2 pins need to be configured as floating for the input mode and open drain for the output mode. To allow wired-or some pull-up resistors (typically 22K) must be externally connected to pulls the lines to +5V. Two IOs of the EM68xx will be used. In our example PB6 is SDA, PB7 is SCK.

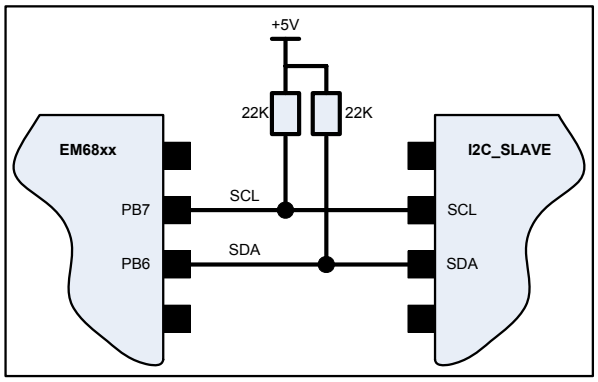

Figure 1: I2C connection

General rules

During normal operation SDA is allow to change only during SCL low state otherwise it generates start/stop conditions.
Every byte must be 8-bits long.
Each byte must be followed by the acknowledge (receiver answer).
Data is transferred MSB first.
To initiate an I2C communication a start condition has to be generated by the master (the bus becomes “busy” until the stop condition). The master will send a first byte including slave address and operation type (Write/read), the acknowledge must be generated by the slave. Depending on the slave IC and the operation type data will be exchanged. Communication will finish when the stop condition will be produced by master.

Start/Stop Condition

To start/stop the communication the master must provide a start/stop condition. These conditions are made with a specific sequence on SCK/SDA (SDA transition during SCK high level state). Only the master is allowed to generate start/stop conditions.

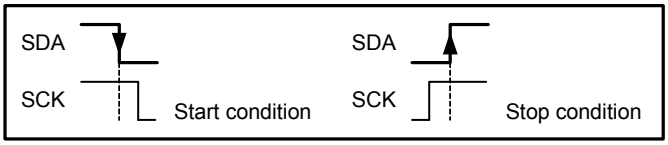

Figure 2: Start/Stop condition

Acknowledge

The master receives the acknowledge after sending an address or a byte to the slave. The master sends the acknowledge after receiving a byte from the slave. Acknowledge is generated on the 9th bit of the trame.

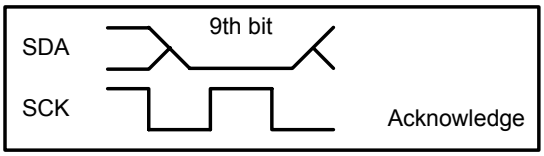

Figure 3: Acknowledge

Trames:

Pattern or communication trames are highly dependant on the I2C parts connected on the bus (E2prom, sensor …). Example of trames is given below.

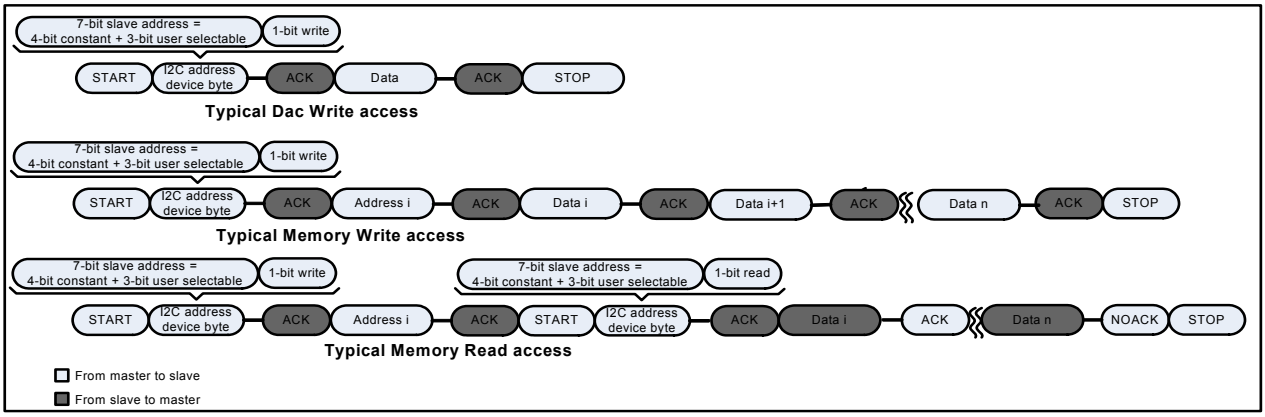

Figure 4: Pattern Examples

Buffer definitions:

This example works with buffers.

In case of Write access, the buffer is filled with the necessary information such as I2C device address, data or memory address etc …

In case of Read access the buffer is filled with the necessary information's for device addressing (I2C device address, memory address) and the remaining of the buffer will be loaded with the receipted bytes.

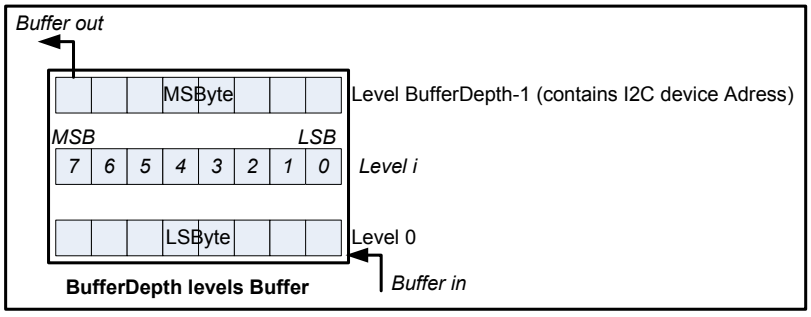

Figure 5: Buffer view

For our example the buffers are based on the following:

DAC access	Level	Write memory byte access	Level	Write memory page access	Level
I2C Dac access	1	I2C Memory access	2	I2C Memory access	9
Dac Data	0	Memory access i	1	Memory access i	8
		Data memory	0	Data memory i	7
				…	…
				Data memory i+7	0

(Level 9 will be sent two times)

Read memory byte access	Level	Write memory page access	Level
I2C Memory access	2	I2C Memory access	9
Memory access i	1	Memory access i	8
To be filled with Data memory i	0	To be filled with Data memory i	7
		…	…
		To be filled with Data memory i+7	0

(Level 2 will be sent two times)

(Level 9 will be sent two times)

Flowchart

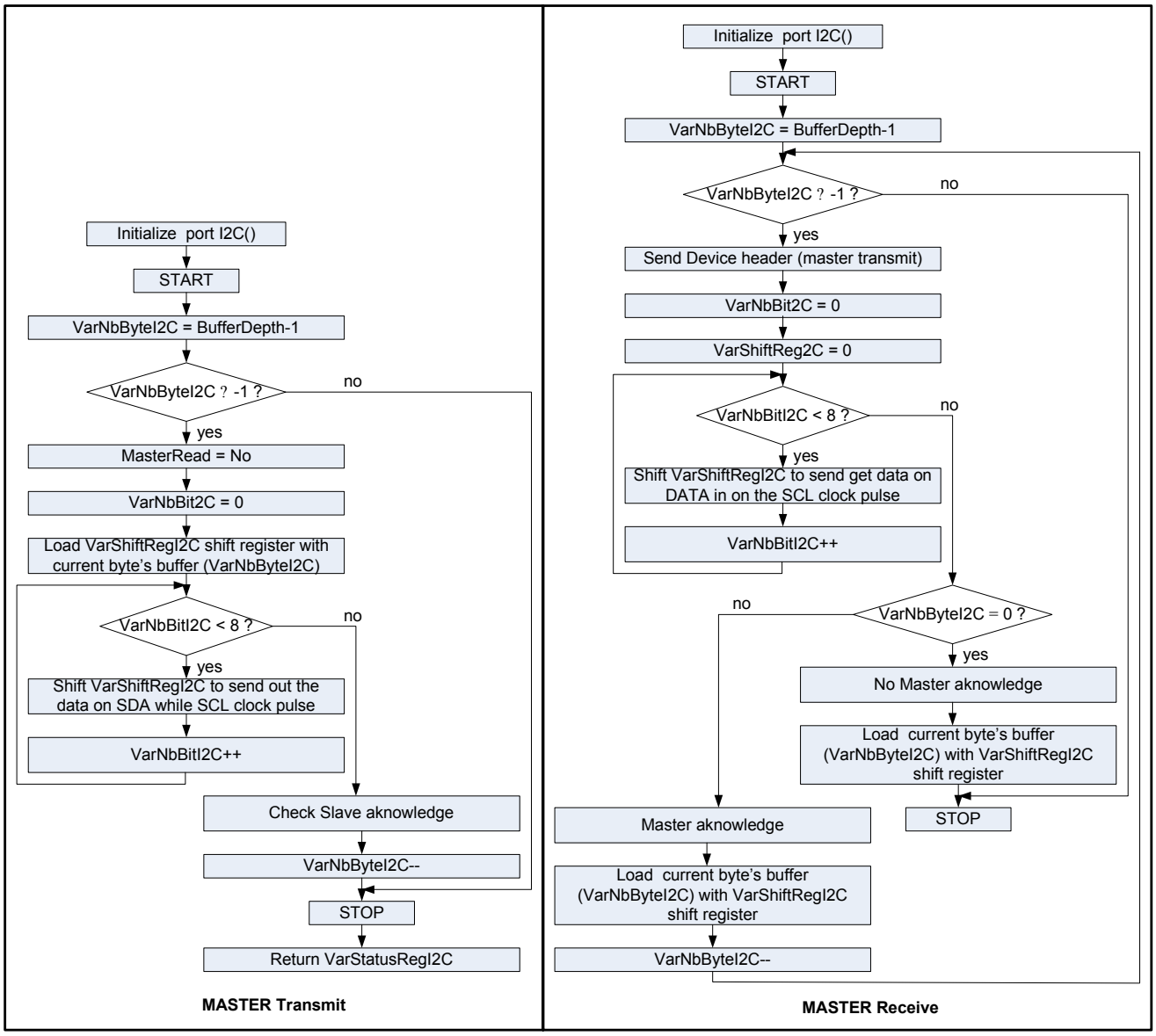

Figure 6: Transmit / Receive flowchart

Status and error

The Status register contain the following information:

Error during I2C communication	BIT7
Reception (‘1’ forRx, ‘0’ for Tx)	BIT1
Acknowledge present (‘1’ if present)	BIT0

This status register is returned after each transmission/reception

Software

The complete corresponding project software is available on our website. It is also available in Appendix A of this document.

Reference

Philips I2C bus specification, Version 2.1 January 2000

Appendix A

File I2c.h

```
/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o File              o I2c.h
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Author            o EM Microelectronic-Marin SA
o Company           o EM Microelectronic-Marin SA
o Project           o skeleton EM6812
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Comments          o Some I2cs usefull definition - dependencie file
o                   o
o                   o
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Version           o 1.0
o Date              o 10.03.06
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/

// Definitions section
#ifndef __I2c__
#define __I2c__

// Include section
#include "Types.h"

/*** usefull I2C definitions ***/
// Define I2C bus on a standard Port IO - Bit 6/7
#define SdaI2C                      BIT7
#define SclI2C                      BIT6
#define RegOpenDrainI2C             RegOpenDrainPB
#define RegIOSelI2C                 RegIOSelPB
#define RegPullDownI2C              RegPullDownPB
#define RegPullUpI2C                RegPullUpPB
#define RegOutI2C                   RegOutPB
#define RegInI2C                    RegInPB
#define CheckComingAckI2C           if ((RegInI2C & SdaI2C) == SdaI2C) { (VarStatusRegI2C |= ErrorStatusI2C); (VarStatusRegI2C &=
~AckStatusI2C); return;}
#define SetAckStatusI2C             VarStatusRegI2C |= AckStatusI2C
#define ClrAckStatusI2C             VarStatusRegI2C &= ~AckStatusI2C
#define SetRxModeStatusI2C          VarStatusRegI2C |= RxModeStatusI2C
#define ClrRxModeStatusI2C          VarStatusRegI2C &= ~RxModeStatusI2C
#define CheckAckStatusI2C           ((VarStatusRegI2C & AckStatusI2C) == AckStatusI2C)
#define CheckErrorStatusI2C         ((VarStatusRegI2C & ErrorStatusI2C) == ErrorStatusI2C)
#define CheckRxModeStatusI2C        ((VarStatusRegI2C & RxModeStatusI2C) == RxModeStatusI2C)
#define ErrorStatusI2C              BIT7
#define RxModeStatusI2C             BIT1
#define AckStatusI2C                BIT0
#define MsbI2C                      BIT7
#define TestMsbTxI2C                ((VarShiftRegI2C & MsbI2C) == MsbI2C)
#define ShiftLeftVarShiftRegI2C     (VarShiftRegI2C = (VarShiftRegI2C << 1))
#define ReadSlaveI2C                1
#define TestSclI2CHigh              ((RegInI2C & SclI2C) == SclI2C)
#define TestSdaI2CHigh              ((RegInI2C & SdaI2C) == SdaI2C)
#define LoadOneVarShiftRegI2C       VarShiftRegI2C |= 1
#define LoadZeroVarShiftRegI2C      VarShiftRegI2C |= 0
#define ReleaseI2C                  RegIOSelI2C &= ~(SdaI2C | SclI2C)   // both pins not driven
#define ForceZeroI2C                RegIOSelI2C |= (SdaI2C | SclI2C)   // both pins to '0'
#define ReleaseSclI2C               RegIOSelI2C &= ~SclI2C // SclI2C not driven
#define ForceZeroSclI2C             RegIOSelI2C |= SclI2C  // SclI2C to '0'
#define ReleaseSdaI2C               RegIOSelI2C &= ~SdaI2C // SdaI2C not driven
#define ForceZeroSdaI2C             RegIOSelI2C |= SdaI2C  // SdaI2C to '0'
#define DeviceMemoryI2C             YES
#define CheckDeviceMemoryI2C        (DeviceTypeI2C & DeviceMemoryI2C) == DeviceMemoryI2C
#define YES                         1
#define NO                          0
#define ResultOperationKO           ((ResultOperation & ErrorStatusI2C) == ErrorStatusI2C)

/*** I2C functions ***/
```

```
void WaitTimeI2C(UINT8 delay_size);
void InitPortI2C(void);
void StartConditionI2C(void);
void StopConditionI2C(void);
void CheckSlaveAcknowledgeI2C(void);
void NoMasterAcknowledgeI2C(void);
void MasterAcknowledgeI2C(void);
void SendBitI2C(void);
void GetBitI2C(void);
unsigned char SendBufferI2C(unsigned char* pBufferOut, unsigned char BufferDepth, unsigned char DeviceTypeI2C);
unsigned char GetBufferI2C(unsigned char* pBufferIn, unsigned char BufferDepth);

#endif /* __I2c__ */
```

File I2c.c

```
/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o File                  o I2c.c
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Author                o EM Microelectronic-Marin SA
o Company               o EM Microelectronic-Marin SA
o Project               o i2C
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Comments              o I2c - C file
o                       o
o                       o
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Version               o 1.0
o Date                  o 10.03.06
o Change                o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/

// Definitions section
#ifndef __I2c_c__
#define __I2c_c__
// Include section
#include "Header.h"

// usefull I2C variables
unsigned char           VarNbByteI2C = 0;
unsigned char           VarNbBitI2C = 0;
unsigned char           VarShiftRegI2C = 0;
unsigned char           VarStatusRegI2C = 0;

// Function declarations

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o WaitTimeI2C
o Comments          o This function generate a delay time based on Nop
o                   o operation. The parameter delay_size introduce the
o                   o number of occurences.
o                   o Delay = (10xFcpu + (6*Fcpu)*delay_size)
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void WaitTimeI2C(UINT8 delay_size)
{
  while (delay_size != 0)
  {
    Nop;
    delay_size--;
  }
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o InitPortI2C
o Comments          o This function set PB6 and PB7 in a compatible I2C
o                   o configuration: opendrain + pull-up + default input
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void InitPortI2C(void)
{
  RegOpenDrainI2C |= (SdaI2C | SclI2C);   // active open drain
  ReleaseI2C;      // input mode
  RegPullDownI2C &= ~(SdaI2C | SclI2C);   // no pull-down
  RegPullUpI2C = (SdaI2C | SclI2C);       // pull-up
  RegOutI2C &= ~(SdaI2C | SclI2C);        // 0 as default output reg
}
```

```
/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o StartConditionI2C
o Comments          o This function generate a start condition
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void StartConditionI2C(void)
{
  ReleaseI2C;   // input mode to force high-level
  WaitTimeI2C(10);
  ForceZeroSdaI2C; // low transition on SdaI2C
  WaitTimeI2C(4);
  ForceZeroSclI2C; // force to low SclI2C
  WaitTimeI2C(6);
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o StopConditionI2C
o Comments          o This function generate a start condition
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void StopConditionI2C(void)
{
  ForceZeroI2C;   // force low level
  WaitTimeI2C(10);
  ReleaseSclI2C; // force SclI2C high
  WaitTimeI2C(4);
  ReleaseSdaI2C; // force SdaI2C high
  WaitTimeI2C(6);
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o CheckSlaveAcknowledgeI2C
o Comments          o This function wait and check the acknowledge
o                   o generated by slave.
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void CheckSlaveAcknowledgeI2C(void)
{
  unsigned char i;
  ForceZeroSclI2C;  // force low level on SclI2C
  ReleaseSdaI2C; // release SdaI2C to let the slave drive 0
  for (i=0; i<8; i++)     // first step of test for acknowledge
    {
      WaitTimeI2C(10);
      CheckComingAckI2C;
    }
  ReleaseSclI2C; // start generating clock pulse
  for (i=0; i<3; i++)     // second step of test for acknowledge
    {
      WaitTimeI2C(1);
      CheckComingAckI2C;
    }
  ForceZeroSclI2C; // end generating clock pulse
  SetAckStatusI2C;       // register the acknowledge in status register
  WaitTimeI2C(1);
  ForceZeroSdaI2C; // force low level on Sda
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o MasterAcknowledgeI2C
o Comments          o This function generate master acknowledge
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
```

```
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void MasterAcknowledgeI2C(void)
{
   ForceZeroSdaI2C; // force low level on Sda
   WaitTimeI2C(10);
   ReleaseSclI2C; // start generating clock pulse
   WaitTimeI2C(10);
   ForceZeroSclI2C;  // force low level on SclI2C
   WaitTimeI2C(5);
   ReleaseSdaI2C; // release SdaI2C
   SetAckStatusI2C;  // register the acknowledge in status register
}
/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o NoMasterAcknowledgeI2C
o Comments          o This function does not generate master acknowledge
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void NoMasterAcknowledgeI2C(void)
{
  ClrAckStatusI2C; // unregister the acknowledge in status register
  ReleaseSdaI2C; // release SdaI2C to go high
  ForceZeroSclI2C;  // force low level on SclI2C
  WaitTimeI2C(10);
  ReleaseSclI2C; // start generating clock pulse
  WaitTimeI2C(10);
  ForceZeroSclI2C;  // force low level on SclI2C
  WaitTimeI2C(5);
  ClrAckStatusI2C; // unregister the acknowledge in status register
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o SendBitI2C
o Comments          o This function send one bit
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void SendBitI2C(void)
{
  ForceZeroSclI2C; // force low level on SclI2C
  if (!CheckErrorStatusI2C) // no error
    {
     if (TestMsbTxI2C) // Msb in shift reg is '1'
      ReleaseSdaI2C; // release SdaI2C to go high
     else
      ForceZeroSdaI2C; // force low level on Sda
     ShiftLeftVarShiftRegI2C; // shift the Tx buffer
     ReleaseSclI2C; // release SclI2C to go high
     WaitTimeI2C(10);
    }
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o GetBitI2C
o Comments          o This function get one bit
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
void GetBitI2C(void)
{
  if (!CheckErrorStatusI2C) // no error
    {
      ShiftLeftVarShiftRegI2C;  // shift the Rx buffer
      ReleaseSclI2C; // release SclI2C to go high
      while (!TestSclI2CHigh) {}; // Wait until SclI2C fully high
      if (TestSdaI2CHigh)
        LoadOneVarShiftRegI2C;
```

```
    else
      LoadZeroVarShiftRegI2C;
    WaitTimeI2C(10);
    ForceZeroSclI2C; // force low level on SclI2C
  }
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o SendBufferI2C
o Comments          o This function send the buffer content
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
unsigned char SendBufferI2C(unsigned char* pBufferOut, unsigned char BufferDepth, unsigned char DeviceTypeI2C)
{
  unsigned char MasterRead;
  InitPortI2C();
  StartConditionI2C();
  for (VarNbByteI2C = BufferDepth-1; VarNbByteI2C!= 0xFF; VarNbByteI2C--)
    {
      MasterRead = NO;
      if ((VarNbByteI2C == BufferDepth-3) && (CheckRxModeStatusI2C) && (CheckDeviceMemoryI2C))
        {// in case of reading memory access the Memory adress must be put again after the
          StartConditionI2C();// header
          VarNbByteI2C = BufferDepth-1; // -> resend the Memory device adress
          MasterRead = YES;
        }
      VarNbBitI2C = 0;
      VarShiftRegI2C = pBufferOut[VarNbByteI2C]; // load shift reg. TX with the buffer
      if (MasterRead == YES) VarShiftRegI2C |= ReadSlaveI2C; // in this case force R_nW = Read !
      while (VarNbBitI2C < 8) // Send the 8 bits
        {
          SendBitI2C();
          VarNbBitI2C++;
        }
      CheckSlaveAcknowledgeI2C();
      if (MasterRead == YES) return VarStatusRegI2C; // Receive coming byte ... from GetBufferI2C
    }
    StopConditionI2C();
    return VarStatusRegI2C;
}

/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Function name     o GetBufferI2C
o Comments          o This function load the buffer with the coming byte
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/
unsigned char GetBufferI2C(unsigned char* pBufferIn, unsigned char BufferDepth)
{
  InitPortI2C();
  StartConditionI2C();
  for (VarNbByteI2C = BufferDepth-1; VarNbByteI2C!=0xFF; VarNbByteI2C--)
    {
      SetRxModeStatusI2C;
      SendBufferI2C(pBufferIn, BufferDepth, DeviceMemoryI2C); // send header
      if (CheckErrorStatusI2C) return VarStatusRegI2C;; // if error return to main program !
      for (VarNbByteI2C = BufferDepth-3; VarNbByteI2C!= 0xFF; VarNbByteI2C--) // header already sent
        {
          VarNbBitI2C = 0;
          VarShiftRegI2C = 0;
          ReleaseSdaI2C;
          while (VarNbBitI2C < 8) // Send the 8 bits
            {
              GetBitI2C();
              VarNbBitI2C++;
            }
          if (VarNbByteI2C == 0x00)
            {
              NoMasterAcknowledgeI2C();
```

```
            StopConditionI2C();
            pBufferIn[VarNbByteI2C] = VarShiftRegI2C;
            return VarStatusRegI2C;
          }
         else
          {
            MasterAcknowledgeI2C();
            pBufferIn[VarNbByteI2C] = VarShiftRegI2C;
          }
      }
   }
  return VarStatusRegI2C;
}


#endif /* __I2c_c__ */
```

File main.c

```
/*
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o File              o main.c
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Author            o EM Microelectronic-Marin SA
o Company           o EM Microelectronic-Marin SA
o Project           o skeleton EM6812
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Comments          o Main C file
o                   o
o                   o
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
o Version           o 1.0
o Date              o 10.11.04
o Change            o Initial
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
*/

// Definitions section
#define MAIN_FILE

// Include section
#include "Header.h"

// Function section

int main(void)
{
/*Examples of I2C devices:*/
// Max5382Leuk
#define AdI2CMAX5382LEUK 0x60
#define DataMAX5382LEUK 0x41
// AT24C02 with A2...A0 tied to '0'
#define AdI2CAT24C02 0xA0
#define AdAT24C02 0x00

/*Examples of buffer devices*/
// buffer Access DAC (MAX5382 LEUK)

// buffer Access memory (AT24C02}
unsigned char Memory1[10] = {'M','E',' ', 'O','L','L','E','H', AdAT24C02, AdI2CAT24C02}; // Buffer W access
unsigned char Memory2[10] = {' ',' ',' ', ' ',' ',' ',' ',' ', AdAT24C02, AdI2CAT24C02}; // Buffer R access
unsigned char TempDac2[2] = {0x80, AdI2CMAX5382LEUK};
unsigned char TempDac[2]; // MSB byte first ... 0x60 = Device address
unsigned char ResultOperation = 0;


unsigned char i;

  DisWdog();
  ResultOperation = SendBufferI2C(Memory1, 10, YES);
  if (!ResultOperationKO)  ResultOperation = GetBufferI2C(Memory2, 10); else Halt;
  if (!ResultOperationKO)
    {
      TempDac[1] = AdI2CMAX5382LEUK;
      for (i=0; i<10; i++)
        {
          TempDac[0] += Memory2[i];
        }
      for (i=0; i<50; i++)
        {
          ResultOperation = SendBufferI2C(TempDac, 2, NO);
          if (!ResultOperationKO) ResultOperation = SendBufferI2C(TempDac2, 2, NO); else Halt;
        }
    }
  else
    Halt;
  while (1) {};
};
```

THIS APPLICATION NOTE IS AN EXAMPLE OF I2C MASTER IMPLEMENTATION: EM MICROELECTRONIC ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCE OF USE OF THESE INFORMATIONS.
NO LIABILTY IS ASSUMED BY EM MICROELECTRONIC WITH RESPECT TO THE ACCURACY OR USE OF SUCH INFORMATIONS. THIS APPLICATION NOTE AND CORRESPONDING CODE IS SUBJECT TO CHANGE WITHOUT ANY NOTICE.