1 Introduction

The IIC slave is a device connected to the IIC master device. The special IIC slaves can be the serial EEPROMs, real time clock modules, GPIO extenders etc. When the MCU is connected as the IIC slave it can serve as the various kind of peripherals in accordance to internal software.

This application note shows how to implement the IIC slave software module on Freescale’s MCU MC9RS08KA2. Then the MCU KA2 can be used as the IIC GPIO extender. Two standard GPIO pins are used as the SDA and SCL pins and the other free pins of the MCU are usable as standard GPIO pins. You can choose which pins will be used as the IIC bus pins. The PTA0 and PTA1 are used in this example.

2 IIC Bus Summary

The IIC bus is a bidirectional, two-wire bus based on the wired AND (open drain) connection between master and...
slave devices. For proper functionality the external pull-up resistors are used. The right values of these resistors are specified in the IIC Bus Specification from Philips and depend on the mode (speed) used. The standard usable values are from 1k5 to 10kΩ. Each transfer is originated by the master and acknowledged or answered by the slave.

2.1 IIC Bus Terminology

- **Transmitter** — device that sends the data to the bus. A transmitter can be either a device that puts data on the bus of its own accord (master transmitter) or a response to a data request from another device (slave transmitter).
- **Receiver** — device that receives the data from the bus.
- **Master** — component that initializes a transfer, generates the clock signal and terminates the transfer. A master can be either a transmitter or a receiver.
- **Slave** — device addressed by the master. Can be receiver or transmitter.
- **Multi-master** — ability for more than one master to co-exist simultaneously on the bus without collision or data loss.
- **Arbitration** — prearranged procedure that authorizes only one master at a time to control the bus.
- **Synchronization** — prearranged procedure that synchronizes clock signals provided by two or more masters.
- **SDA** — data signal line (serial DAta).
- **SCL** — clock signal line (serial CLock).

SDA and SCL are bidirectional lines connected to a positive supply voltage via a current source or pull-up resistor. When the bus is free, both lines are high. Data on the IIC bus can be transferred at rates of up to 100kbit/s in the standard mode, up to 400kbit/s in the fast mode or up to 3.4Mbit/s in the high speed mode. The number of devices connected to the bus depends only on the bus capacitance. The limit is 400pF.

2.2 Bit Transfer

The data on the SDA line must be stable during the clock’s high period. The high or low state of the SDA line can change only when the clock signal on the SCL line is low. See Figure 1.

2.3 START and STOP Conditions

Within the IIC-bus procedure, unique situations, defined as START and STOP conditions, arise (Figure 2). A high-to-low transition on the SDA line while the SCL line is high is the START condition, and a low-to-high transition on the SDA line while SCL is high is the STOP condition. The master always generates START and STOP conditions. The bus is released if both the SDA and SCL remain at high levels after STOP and before the START conditions.
2.4 Bus Communication

The bus transfer protocol is based on the byte transfers followed by the acknowledge bit (ACK). The byte transfer starts with MSM to LSB. Each bit is clocked by SCL positive clock impulse. The ACK bit is clocked by the 9th SCL impulse.

The master device originates every bus transfer, which sends the START condition to the bus. The next consecutive sequence bytes depends on the type of data transferred through the IIC bus.

2.5 Control byte

The START condition is always followed by the control byte. The role of the control byte is to select and activate specific device types on the bus. See Figure 3 for the control byte’s structure. The first four bits represent the control code of the slave device. The next three bits represents the chip selects of the connected devices on the bus. These three bits can be used for multiple device operation.

The 8th bit is the R/W (Read/Write) bit. This bit determines the requested operation for slave device. If this bit equals to “0” this means the “write” operation and “1” means the “read” operation. The last 9th bit is the ACK bit that, in this case, the slave always generates.
2.6 **Acknowledge**

The acknowledge bit (ACK) is the low-level on the SDA line during the 9th pulse on the SCL line. The master always generates the SCL pulses. The receiver generates the ACK impulses. This means that in the write operation the slave device generates the ACK after each byte is transferred. In the READ operation, the slave generates the ACK bit after the master sends the control byte. The master generates the ACK bit after the first byte is transferred from the slave (if the next data bytes are requested by the master). The ACK bit is not generated when the last byte in the “read block” mode was transferred.

2.7 **Read/Write format**

They are two types of the data transfer on the IIC bus - read from the slave and write to the slave.

When the read operation is executed, the first part of the communication protocol the master writes is the control byte with the R/W bit equal to 1 to the slave. Then the slave sends the ACK to the master. The next byte is the data byte sent by the slave. Then the master does not send the ACK bit (ACK=1); it does generate the STOP bit. Finally, the master and the slave release the bus. See Figure 4 for the read-operation structure.

When the write operation is executed, the data byte follows the control byte. This data byte contains the actual data which will be written to the slave. The slave generates both ACK bits. See Figure 5.
2.8 IIC software for MCU

The main goal is to reach the highest possible communication speed by IIC; therefore, the whole software code is written in assembler.

2.9 Initialization of the IIC

The main task is to set the right speed of the SCL clock signal. These clocks are based on the bus frequency. The internal ICS module can run at 16MHz to achieve the maximal bus frequency of 8MHz when the internal reference is trimmed. So the internal bus clocks is at 8MHz. The main state when the MCU is not active is WAIT state. The MCU leaves from this state by KBI interrupt. So the KBI interrupt on the “SCL” pin is activated before entering to WAIT state. After wake-up, the KBI interrupt is disabled.

2.10 WRITE function

The write to slave will be discussed first. This will be done by send the start bit followed by the slave control byte with the R/W bit equal to “0”. The meaning of this byte’s contents and bits is discussed in the previous section. When the master sends the byte, the slave sends the ACK bit during the 9th SCL impulse. The master recognizes this ACK bit as the slave is ready to receive next byte. The next byte (second byte) is the data byte. The data will be written to the slave. This data byte is followed by the ACK sent by the slave. When the master receives the ACK bit, it generates the STOP bit. At this moment, the slave disconnects internally from the IIC bus, processes the write operation, then writes the transferred data to port A. The whole data byte is transferred, but only several bits are possible to write to the whole port; this depends on the GPIO pins configuration. Port A consists from PTA0-PTA5 and the IIC bus uses two bits of them.

2.11 READ function

The read function is similar to the write function. The master sends the control byte but the R/W bit must be equal to 1. This is the read-from-slave function. The slave answers by ACK. The master must consecutively switch to receive mode and generate SCL pulses for the slave to receive data. When the master receives the whole byte (8 bits), it generates the 9th SCL pulse with ACK = 1 followed by the STOP bit. After the STOP bit is on the bus, the master and slave release the bus. The SDA & SCL lines stay inactive, which equals 1.

The read portA instruction on the RS08KA2 occurs when the first (control) byte with R/W = 1 is received by the slave. The actual portA data is read immediately and sent to the master as a data byte.
3 Conclusion

The goal of this note is to explain how to implement the IIC slave module in Freescale’s MCU MC9RS08KA2. The RS08KA2 MCU serves in this case as the IIC slave 4-bit GPIO port extender. The whole software code example is also available on the MCU web page.

The whole application is tested with the MCU MC9S12DP256B with the internal IIC module configured for a speed of 100 kHz.
Appendix A

/*******************************************************************************
*                                                                             *
* Freescale Semiconductor Inc.                                               *
* (c) Copyright 2004-2006 Freescale Semiconductor, Inc.                       *
* ALL RIGHTS RESERVED.                                                        *
*                                                                             *
*******************************************************************************/

; ; Example for RS08KA2 - IIC Slave; max speed 100kHz
; * can serve as IIC 4-bit GPIO port
; ; Include derivative-specific definitions
INCLUDLe 'derivative.inc'

; export symbols

XDEF _Startup
ABSENTRY _Startup

SCL_PIN equ PTAD_PTAD0          ; SCL = PTA0;
SDA_PIN equ PTAD_PTAD1          ; SDA = PTA1;

; MACRO Definitions:

SDA0: MACRO
        bclr   SDA_PIN,PTAD
ENDM

SDA1: MACRO
        bset   SDA_PIN,PTAD
ENDM

SDA: MACRO
        brclr SCL_PIN,PTAD,\1
        brclr SDA_PIN,PTAD,\2
ENDM

SCL0: MACRO
        bclr PTAD_PTAD0,PTAD
        bset PTADD_PTADD0,PTADD
ENDM

SCL1: MACRO
        bclr SCL_PIN,PTADD
ENDM

IIC Slave on RS08KA2, Rev. 0
IIC Slave on RS08KA2, Rev. 0

Conclusion

WSCL1: MACRO
    brclr SCL_PIN,PTAD,\1
ENDM

WSCL0: MACRO
    brset SCL_PIN,PTAD,\1
ENDM

TX0: MACRO
    bclr SDA_PIN,PTAD
    bset SDA_PIN,PTADD
ENDM

; variable/data section

ORG    RAMStart         ; Insert your data definition here
My_IIC_Addr: DS.B   1
r_w     DS.B   1
RX_data: DS.B   1
TX_data: DS.B   1
bit_cnt_RX: DS.B   1
bit_cnt_TX: DS.B   1

; code section

ORG    ROMStart

;-----------------------------------------------------------------------
_Startup:

; CONFIGURES SYSTEM CONTROL
    mov #HIGH_6_13(SOPT), PAGESEL
    mov #$02, MAP_ADDR_6(SOPT)        ; Disables COP;

; CONFIGURES CLOCK (FEI Operation Mode)
    mov #HIGH_6_13(NV_ICSTRM),PAGESEL
    lda MAP_ADDR_6(NV_ICSTRM)
    sta ICSTRM           ; Sets trimming value
    clr ICSC1           ; Selects FLL as clock source
    ; and disables it in stop mode
    clr ICSC2           ; ICSOUT = DCO output frequency

wait_clock:
    brset 2,ICSSC,wait_clock  ; Waits until FLL is engaged

; CONFIGURES PORT A
    clr PTADD           ; PTA as inputs (OR outputs);
    clr PTAD
; INITIALIZE VARIABLES

init var:
clr a
sta RX_data
sta TX_data
sta bit_cnt_RX
sta bit_cnt_TX
sta r_w
lda #$B0 ; own device IIC address
          ; in accordance to your requirements;
          sta My_IIC_Addr

;-----------------------------------------------------------------------

mainLoop:

feed_watchdog
lda PTAD ; wait for IIC bus not active,
          ; then go to WAIT mode;
          and #$03
          cmp #$03
          bne mainLoop

; CONFIGURES KEYBOARD INTERRUPT MODULE

bclr 1,KBISC             ; Disable KBI interrupt;
mov #HIGH_6_13(PTAPE), PAGESEL
mov #$03, MAP_ADDR_6(PTAPE) ; Enables internal Pulling device
          ; on PTA0 & PTA1;
bset KBIPE_KBIPE0,KBIPE    ; Enable KBI0 pin;

bset KBISC_KBIE,KBISC     ; Enable KBI interrupt;
bset KBISC_KBACK,KBISC    ; Clear the flag;

wait              ; MCU in low power mode
          ; will wake-up by KBI interrupt;
          ; instruction needs to be disabled for debug option;
mov #HIGH_6_13(SIP1), PAGESEL
brset 4,MAP_ADDR_6(SIP1),Kboard ; Branch if KB interrupt pending

Kboard:

bclr KBISC_KBIE,KBISC    ; Disable KBI interrupt;
bset KBISC_KBACK,KBISC   ; Clear the flag;

feed_watchdog
lda PTAD
          ; START condition occured;
          and #$03
          tsta
          bne Kboard
          bne Kboard

;************************************************************************

;---------- start receive first byte ------------------------
clr a
Conclusion

;---------- receive bit7 ---------------------

bit17:
  SDA bit17,bit17a ; wait for SCL=1, then read SDA
  add #$01         ; read SDA = 1
bit17a:
  lsla             ; read SDA = 0
w17:
  WSCL0 w17        ; wait for SCL = 0;

;---------- receive bit6 ---------------------

bit16:
  SDA bit16,bit16a
  add #$01         ; read SDA = 1
bit16a:
  lsla             ; read SDA = 0
w16:
  WSCL0 w16

;---------- receive bit5 ---------------------

bit15:
  SDA bit15,bit15a
  add #$01         ; read SDA = 1
bit15a:
  lsla             ; read SDA = 0
w15:
  WSCL0 w15

;---------- receive bit4 ---------------------

bit14:
  SDA bit14,bit14a
  add #$01         ; read SDA = 1
bit14a:
  lsla             ; read SDA = 0
w14:
  WSCL0 w14

;---------- receive bit3 ---------------------

bit13:
  SDA bit13,bit13a
  add #$01         ; read SDA = 1
bit13a:
  lsla             ; read SDA = 0
w13:
  WSCL0 w13

;---------- receive bit2 ---------------------

bit12:
  SDA bit12,bit12a
  add #$01         ; read SDA = 1
bit12a:
  lsla             ; read SDA = 0
w12:
WSCL0 w12
;---------- receive bit1 ---------------------

bit11:
SDA bit11, bit11a
add #$01 ; read SDA = 1

bit11a:
lsla ; read SDA = 0

w11:
WSCL0 w11
;---------- receive bit0 ---------------------

bit10:
SDA bit10, bit10a
inc r_w ; read SDA = 1 -> bit RW = 1;

bit10a: ; read SDA = 0 -> bit RW = 0;

w10:
WSCL0 w10
;---------- send SL_ACK ----------------------

TX0

sl_ack_1a:
WSCL1 sl_ack_1a
cmp My_IIC_Addr ; compare valid own IIC address;
bne go_back ; go to start if not match;

sl_ack_1b:
WSCL0 sl_ack_1b
bclr SDA_PIN, PTADD ; SDA = PTA1 as input;
;---------- read the PTA port ---------------------
lda PTAD ; read portA
and #$30 ; mask bits 4 & 5;
sta TX_data ; save data;
;---------- end of read the PTA port ---------------------
bne go_back ; go to start if not match;
brc lr 0, r_w, gol ; go to receive next byte (RX_data);
bra go_read_port ; go to send (TX_data = PTA) to Master;

;---------- end of receive first byte ---------------------

go_back:
clr
sta PTADD
jmp init_var ; not valid device address -> go to WAIT state;

go_read_port:
jmp read_port ; read port A and send data to master;
;******************************************************************************;
;---------- start receive second byte ----------------------------------------

gol:
clra
              ;-------- receive bit7 ---------------------
bit27:
      SDA bit27,bit27a ; wait for SCL=1, then read SDA
      add #$01        ; read SDA = 1
bit27a:
      lsla            ; read SDA = 0
w27:
      WSCL0 w27      ; wait for SCL = 0;
              ;-------- receive bit6 ---------------------
bit26:
      SDA bit26,bit26a
      add #$01       ; read SDA = 1
bit26a:
      lsla           ; read SDA = 0
w26:
      WSCL0 w26      ; receive bit5 ---------------------
bit25:
      SDA bit25,bit25a
      add #$01       ; read SDA = 1
bit25a:
      lsla           ; read SDA = 0
w25:
      WSCL0 w25      ; receive bit4 ---------------------
bit24:
      SDA bit24,bit24a
      add #$01       ; read SDA = 1
bit24a:
      lsla           ; read SDA = 0
w24:
      WSCL0 w24      ; receive bit3 ---------------------
bit23:
      SDA bit23,bit23a
      add #$01       ; read SDA = 1
bit23a:
      lsla           ; read SDA = 0
w23:
      WSCL0 w23      ; receive bit2 ---------------------
bit22:
      SDA bit22,bit22a
      add #$01       ; read SDA = 1
bit22a:
      lsla           ; read SDA = 0
w22:
WSCL0 w22
;---------- receive bit1 ---------------------

bit21:
SDA bit21,bit21a
add #$01 ; read SDA = 1

bit21a:
rla ; read SDA = 0

w21:
WSCL0 w21
;---------- receive bit0 ---------------------

bit20:
SDA bit20,bit20a
add #$01 ; read SDA = 1

bit20a:

w20:
WSCL0 w20
;---------- end of receive second byte ---------------------

;---------- send SL_ACK ---------------------
TX0 ; put 0 to SDA -> SL_ACK;

sl_ack_2a:
WSCL1 sl_ack_2a
sta RX_data ; save received data byte;

sl_ack_2b:
WSCL0 sl_ack_2b
bclr SDA_PIN,PTADD ; SDA = PTA1 as input;

w_stop:
feed_watchdog
lda PTAD
and #$03
cmp #$01 ; test for SDA = 0, SCL = 1;
bne w_stop

fin_stop:
feed_watchdog
lda PTAD
and #$03
cmp #$03
bne fin_stop ; STOP condition occurred / SDA = SCL = 1 /

;---------- end of IIC receive procedure ---------------------
lda RX_data
and #$30
sta PTAD
lda #$30
sta PTADD ; write received data to PortA;
jmp init_var ; go to WAIT state;

;******************************************************************************
;---------- Send data to Master -------------------------------------
read_port:
bset SDA_PIN,PTADD ; SDA = PTA1 as output;
 ;---------- send bit7 ------------------------
brclr 7,TX_data,tx07
SDA1

tx07:
WSCL1 tx07

tx07a:
WSCL0 tx07a
SDA0
;---------- send bit6 ------------------------
brclr 6,TX_data,tx06
SDA1

tx06:
WSCL1 tx06

tx06a:
WSCL0 tx06a
SDA0
;---------- send bit5 ------------------------
brclr 5,TX_data,tx05
SDA1

tx05:
WSCL1 tx05

tx05a:
WSCL0 tx05a
SDA0
;---------- send bit4 ------------------------
brclr 4,TX_data,tx04
SDA1

tx04:
WSCL1 tx04

tx04a:
WSCL0 tx04a
SDA0
;---------- send bit3 ------------------------
brclr 3,TX_data,tx03
SDA1

tx03:
WSCL1 tx03

tx03a:
WSCL0 tx03a
SDA0
;---------- send bit2 ------------------------
brclr 2,TX_data,tx02
SDA1
tx02:
   WSCL1 tx02
tx02a:
   WSCL0 tx02a
SDA0
   ;---------- send bit1 ------------------------
   brclr 1,TX_data,tx01
SDA1
tx01:
   WSCL1 tx01
tx01a:
   WSCL0 tx01a
SDA0
   ;---------- send bit0 ------------------------
   brclr 0,TX_data,tx00
SDA1
tx00:
   WSCL1 tx00
tx00a:
   WSCL0 tx00a
SDA0
   ;--------- end of send data -----------------------
   bclr SDA_PIN,PTADD ; SDA as input;
   ;--------- MS_ACK Time gap -----------------------
ms_ack_rda:
   WSCL1 ms_ack_rda
ms_ack_rdb:
   WSCL0 ms_ack_rdb
   ;--------- end of SL_ACK ------------------------
w_stop1:
   feed_watchdog
   lda PTAD
   and #$03
   cmp #$01 ; test for SDA = 0, SCL = 1;
   bne w_stop1
   fin_stop1:
   feed_watchdog
   lda PTAD
   and #$03
   cmp #$03
   bne fin_stop1
   ;--------- end of IIC communication ----------------
   jmp init_var ; go to WAIT state;

******************************************************************************************

;******************************************************************************
;*                Startup Vector                                              *
;******************************************************************************

ORG $3FFD
JMP _Startup ; Reset
How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2006. All rights reserved.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to http://www.freescale.com/epp.

Document Number: AN3304
Rev. 0
09/2006