1 Introduction

The IIC (inter-integrated circuit) protocol is a 2-wire serial communication interface implemented in numerous microcontrollers and peripheral devices. Many microcontroller units (MCUs) do not have an IIC module, yet they must communicate with 2-wire or IIC devices. These MCUs are usually “master on IIC.”

This application note describes a method of communicating on an IIC bus by controlling digital input/output (I/O) pins. This “bit banged” method can execute on any Freescale MCU through the standard digital I/O pins; however, this document uses the MC9RS08KA2 as an example.

2 IIC Overview

IIC is a 2-wire communications link requiring a clock line (SCL) and a data line (SDA) to communicate. The frequency of the IIC clock can go up to 100kHz for standard mode and up to 400kHz for fast mode.
An IIC bus has both a master device and a slave device attached to it. A master initiates a transfer, generates a clock signal (SCL), and terminates the transfer. The master device also addresses a slave device. IIC provides a solution for multiple masters on the same bus. This bus also provides some error checking by using acknowledgment bits during byte transfer.

3 IIC Master Application

The application presented in this document illustrates a basic example of the IIC specification. It is not intended to implement all the features of an IIC bus. It provides only the basic functionality required to transmit from a master device to a slave device through a 2-wire interface. The advantage of this method is it uses standard digital I/O pins available on any Freescale MCU.

This application provides the following functionality:

- 7-bit addressing mode of IIC slave device
- Single master transmitter
- Serial clock frequency of approximately 160 kHz
- Multiple data bytes within a serial transfer
- Acknowledgment polling of error checking

By controlling two digital pins, a designer can simulate an IIC transfer message. These I/O pins should be open drain. If the I/O pins are high-density complementary metal oxide semiconductor (CMOS) and not open drain, some safeguards must be implemented. A series resistor should connect the CMOS output pin and receiver input pin. If the two devices attempt to output conflicting logic levels, this provides some current limiting.

Another consideration is supporting a logic high level for any open-drain receiver pins. You can use a pull-up resistor at the receiver’s open drain pin to passively pull up to the supply voltage when the pin is not actively driven low. Carefully choose this pull-up resistor so that when the master pin drives low, a valid VIL level presents to the IIC receiver pin.

Figure 1 illustrates how to connect digital I/O pins between IIC master and slave devices.
When the SCL and SDA I/O pins of the master and slave devices are open drain, you can omit the resistors R1 and R2.

Specific stages defined by the states of the SCL and SDA wires compose an IIC transfer. The inactive state of the IIC bus happens while both SCL and SDA lines are in the high logic level. Figure 2 shows the timing between the clock (SCL) and data (SDA) lines under the START and STOP conditions; Figure 3 shows the timing between SCL and SDA lines during the data transfer. Figure 4 shows the timing of the acknowledge impulse sent by the slave device after it receives all eight bits of the transferred byte.

4 Basic States

Characteristics of the basic states:

— Falling edge on SDA line while the SCL is held in the high logic level indicates START condition.
— Rising edge on SDA line while the SCL is held in the high logic level indicates STOP condition.
— Data on the SDA line can change only if the SCL line is in the low logic level
— Data on the SDA line is valid and is transferred through the IIC bus between devices when the SCL line is in the high logic level
— Low logic level on the SDA line indicates acknowledge bit (ACK), while the SCL line is the ninth pulse from the byte transfer. The slave device usually generates the ACK bit. The master produces the ACK bit only if the “multiple read function” occurred.

![Figure 2. START and STOP Conditions on IIC Bus](image-url)
5  IIC Bus Transfer

The START condition (START bit) produced by the master device initiates the data transfer through the IIC bus. The device address byte, with its most significant bit (MSB) first, follows the start bit. The least significant bit (LSB) in the device address byte can be high or low, depending on whether it is a “read” or “write” operation.

With all bytes transferred on the IIC bus, a ninth clock cycle gives an acknowledgment. The SDA line is read during this ninth clock cycle by the master and signifies whether the byte is acknowledged. The receiver drives the SDA line low during the ninth clock cycle if it acknowledges the byte transmission.

Any number of data bytes can follow the address byte, each composed of eight data bits and a ninth acknowledge bit. To end a transfer, a STOP condition imposes on the IIC bus. A rising edge on SDA, while SCL line is held high, indicates the Stop condition.

NOTE:
To avoid unwanted conditions, the software must transition the SDA pin only while the SCK line is held low.
This application requires some software overhead, but is somewhat interruptible as the IIC bus is completely synchronous. A more automated timing source, such as a free-running counter or real-time interrupt, could create an implementation that requires less software overhead.

Appendix A

The code shows how a MC9RS08KA2 microcontroller can connect to an IIC peripheral. The software continuously sends a write command, ramping the digital value from $00 to $FF and back down again. Using the DEMO9RS08KA2 board, the SCL signal is approximately 160kHz.

;*******************************************************************************************
;       RS08IICMaster.ASM
;*******************************************************************************************
; Copyright (C) 2006 Freescale Semiconductor, Inc.
;       All Rights Reserved
;*******************************************************************************************
;
; Description: User Code for MC9RS08KA2
;       Bit Bashed IIC Master for MC9RS08KA2
;       Tested on DEMO9RS08KA2
;
; Engineer : Inga Harris
; Date : 21/06/06
;
; Notes:
;*******************************************************************************************
; * THIS CODE IS ONLY INTENDED AS AN EXAMPLE OF CODE FOR THE
; * CODEWARIOR COMPILER AND HAS ONLY BEEN GIVEN A MIMIMUM
; * LEVEL OF TEST. IT IS PROVIDED 'AS SEEN' WITH NO GUARANTEES
; * AND NO PROMISE OF SUPPORT.
;*******************************************************************************************
;
; Freescale reserves the right to make changes without further notice to any
; product herein to improve reliability, function, or design. Freescale does
; not assume any liability arising out of the application or use of any
; product, circuit, or software described herein; neither does it convey
; any license under its patent rights nor the rights of others. Freescale
; 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 life, or for any other application in
; which the failure of the Freescale product could create a situation where
; personal injury or death may occur. Should Buyer purchase or use Freescale
; products for any such intended or unauthorized application, Buyer shall
; indemnify and hold Freescale 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
; was negligent regarding the design or manufacture of the part. Freescale
; and the Freescale logo* are registered trademarks of Freescale Ltd.

;*******************************************************************************************
; Macro to manage nested Subroutine entry code
;*******************************************************************************************
ENTRY_CODE: MACRO
IIC Master on the MC9S08KA2, Rev. 0

IIC Bus Transfer

SHA
STA pcBUFFER+(2*(\1))
SHA
SLA
STA pcBUFFER+(2*(\1))+1
SLA
ENDM

;**************************************************************************
; Macro to manage nested Subroutine exit code
;**************************************************************************
EXIT_CODE: MACRO
SHA; this line can be removed if Accumulator contents are no longer needed
LDA pcBUFFER+2*(\1)
SHA
SLA; this line can be removed if Accumulator contents are no longer needed
LDA pcBUFFER+2*(\1)+1
SLA
ENDM

; Include derivative-specific definitions
INCLUDE 'derivative.inc'

; export symbols
XDEF _Startup, main
; we export both '_Startup' and 'main' as symbols. Either can
; be referenced in the linker.prm file
MAXlevel EQU 1 ; Nesting depth for subroutine macro
PTAPE2 EQU 2 ; Pull Up Resistor bit location for RESET pin
CLKST EQU 2 ; Bit Location of Clock mode status bit in
; ICS Status and Control Register

;******************************************************************
; Emulated IIC lines on Port A
; Need a clock (SCL) and data (SDA)
;**************************************************************************
SCL EQU 0 ; Bit Location of Serial Clock Pin in PortA
SDA EQU 4 ; Bit Location of Serial Data Pin in PortA

;**************************************************************************
; Variables to be held in RAM
;**************************************************************************
_RAMStart:
BitCounter RMB 1 ; Used to count bits in a Tx
Value RMB 1 ; Used to store data value
Direction RMB 1 ; Indicates increment(1) or decrement(0)
pcBUFFER DS.W MAXlevel ; Buffer for return address of nested subroutine macro

;**************************************************************************
; Start of program code
;**************************************************************************
_Startup:
main:
JSR L0_init
; use following lines if using reset default settings
;mov #HIGH_6_13(SOPT), PAGESEL
;mov $000, MAP_ADDR_6(SOPT) ; disable COP

; initialise variables
CLR BitCounter
CLR Value
CLR Direction ; count down

;Set up parallel ports
LDA #$11 ; PTA0 and PTA4
STA PTAD ; as high
STA PTADD ; and outputs

;set up interrupt for Wait instruction
BSET KBISC_KBIE,KBISC ; enable KBI
BSET KBIPE_KBIPE1,KBIPE ; on pin 7, SW0, falling edge

;*******************************************************************
; Loop to ramp up and down the data value sent
;*******************************************************************
DataLoop:
WAIT ; Wait command for debugging purposes
to allow user to see the code working a KBI on SW0 is used
to each loop iteration
BSET KBISC_KBACK,KBISC ; clear flag
LDA Direction ; inc or dec
BEQ GoUp

;GoDown:
LDA Value
BNE GoDown2 ; decrement
CLR Direction ; change direction if needed
BRA SendIICStartBit

GoDown2:
DEC Value ; decrement data value
BRA SendIICStartBit

GoUp:
LDA Value
CMP #$FF ; increment
BNE GoUp2 ; change direction if needed
BRA SendIICStartBit

GoUp2:
INC Value ; increment data value
BRA SendIICStartBit

;*******************************************************************
; Main Loop to send the message
; Start Bit + Address + Command + Data + Stop Bit + Wait for Ack
;*******************************************************************
SendIICStartBit: ; Send Start Condition
BCLR  SDA,PTAD
JSR   IICBitDelay ; Start Condition is defined as a falling egde
BCLR  SCL,PTAD ; on SDA while SCL is high
SendAddress:
LDA #$78                  ; Arbitrary Slave Device address (7bit)
ASLA                      ; Align address
JSR L0_IICTxByte          ; send the 8 Bit Address
LDA #$00                  ; Command Byte for Slave Device
JSR L0_IICTxByte          ; Send the 8-bit Command
LDA Value                 ; Send value to Slave Device
JSR L0_IICTxByte          ; Send the Value
JSR IICStopBit            ; Send Stop Condition
JSR IICBitDelay           ; Wait a Bit
BRA DataLoop              ; Repeat with next Value

;*******************************************************************************************
; Peripheral Initialization
;*******************************************************************************************
L0_init:
ENTRY_CODE 0
;CONFIGURES SYSTEM CONTROL
MODE:         EQU 0                 ; MODE=0 Background Mode, MODE=1 Run Mode
IFNE  MODE
    mov #HIGH_6_13(SOPT), PAGESEL
    mov #$01, MAP_ADDR_6(SOPT)        ; Disables COP and enables RESET (PTA2) pin
    mov #$38, PTADD                   ; Configures PTA3, PTA4 and PTA5 as output
ELSE
    mov #HIGH_6_13(SOPT), PAGESEL
    mov #$03, MAP_ADDR_6(SOPT)        ; Disables COP and enables BKGD (PTA3) and RESET (PTA2) pins
    mov #$30, PTADD                   ; Configures PTA4 and PTA5 as output
ENDIF

;CONFIGURES CLOCK (FEI Operation Mode)
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 CLKST,ICSSC,wait_clock      ; Waits until FLL is engaged

;CONFIGURES TIMER
mov #$70, MTIMSC                  ; Enables interrupt, stops and resets timer counter
mov #$05, MTIMCLK                 ; Selects fBUS as reference clock (8 MHz)
    ; with prescaler = 32 (increments timer counter every 4 us)

;CONFIGURES ACMP
clr ACMPSC                        ; Selects analog comparator between ACMP+ and ACMP-

;CONFIGURES I/O CONTROL PORT
mov #HIGH_6_13(PTAPE), PAGESEL
bset PTAPE2, MAP_ADDR_6(PTAPE)    ; Enables pullup in RESET (PTA2) pin
clr PTAD                          ; Clears PTA port
EXIT_CODE 0
rts
IIC Bus Transfer

; IICTxByte
; Transmit the byte in A to the SDA pin
; A not restored on return
; Must be careful to change SDA values only when SCL is low
; otherwise a stop or start could be implied
;****************************************************************************

L0_IICTxByte:
ENTRY_CODE 0
;initialise variable
LDX #$08
STX BitCounter

IICNextBit:
ROLA
; Shift MSB in to carry
BCC SendLow
; Send low or high bit

SendHigh:
BSET SDA,PTAD
; Set the data bit value
JSR IICSetUpDelay
; Give some time for data setup
BSET SCL,PTAD
; Clock it in
JSR IICBitDelay
; Wait
BRA IICTxCont
; Continue

SendLow:
BCLR SDA,PTAD
; Set the data bit value
JSR IICSetUpDelay
; Give some time for data setup
BSET SCL,PTAD
; Clock the line to get Ack
JSR IICBitDelay
; Wait

IICTxCont:
BCLR SCL,PTAD
; Restore Clock to Low State
DEC BitCounter
; Decrement the bit counter
BEQ IICAckPoll
; Last Bit?
BRA IICNextBit

IICAckPoll:
BSET SDA,PTAD
; set SDA as input
BCLR SDA,PTADD
; Look for Ack from Slave device
BSR IICNoAck

IICAck:
BCLR SCL,PTAD
; Restore clock line
BSET SDA,PTADD
; SDA back as output
EXIT_CODE 0
RTS

;****************************************************************************
; No Ack received from slave device
; Some error action can be performed here
; For this example we just restore the bus
;****************************************************************************

IICNoAck:
Appendix B

The code also allows an introduction on how to write nested subroutines with the stack-less RS08 core using macros and careful “level control.” A quick tutorial on the necessity of this and how it works appears in a little more detail than in the RS08QRUG.

The RS08 core has no stack and therefore has lost the ability to inherently cope with nested subroutine calls. The addition of the Shadow Program Counter (SPC) enables software to overcome this issue.

Table 1. Subroutine Command

<table>
<thead>
<tr>
<th></th>
<th>HC(S)08</th>
<th>RS08</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR Function</td>
<td>Push current PC on to stack</td>
<td>Store PC in SPC</td>
</tr>
<tr>
<td>JSR Function</td>
<td>Load PC with address of function</td>
<td>Load address of function into PC</td>
</tr>
<tr>
<td>RTS</td>
<td>Pull return address from stack and set PC</td>
<td>Copy SPC to PC</td>
</tr>
</tbody>
</table>
Negligible difference occurs in calling a subroutine on HC(S)08 and RS08, but the lack of stack causes problems when calling a subroutine from a subroutine. (See Figure 5)

When calling a subroutine, the PC saves to the SPC before the jump to subroutine executes. On returning from a subroutine the PC loads from the saved return address on the SPC. However, when calling a subroutine from a subroutine, the current PC stores in the SPC overwriting the PC, that would have returned the program to the initial entry point. This means that the CPU can no longer go back to the main program as illustrated in the first program-flow diagram. By using a macro and a clear subroutine level system in your software to manage subroutine nesting, this issue overcomes as shown in the second program-flow diagram.

![Figure 5. Nested Subroutine on RS08](image)

**Assembly - Macro Code**

```
;********************************************************************
; Macro to manage nested Subroutine entry code
;********************************************************************
ENTRY_CODE: MACRO
    SHA
    STA pcBUFFER+(2*(\1))
    SHA
    SLA
    STA pcBUFFER+(2*(\1))+1
    SLA
ENDM
```

**IIC Master on the MC9RS08KA2, Rev. 0**
EXIT_CODE: MACRO
SHA  ; this line can be removed if Accumulator contents are no longer needed
LDA pcBUFFER+2*(\1)
SHA
SLA  ; this line can be removed if Accumulator contents are no longer needed
LDA pcBUFFER+2*(\1)+1
SLA
ENDM

The macro swaps the high-byte and then low-byte of the SPC with the accumulator. If the accumulator contents are not needed in the exit macro, you can skip this step. The macro then stores (entry) or loads (exit) the accumulator value into or from the pcBUFFER located in RAM in the position governed by the subroutine level. The SPC returns to its original state with another swap command.

Figure 6. Entry Code Macro
Example of subroutine using the macro

L0_IICTxByte:
  ENTRY_CODE 0
  ;Initialise variable
  LDX #$08
  STX BitCounter

IICNextBit:
  ROL A                      ;Shift MSB in to carry
  BCC SendLow               ;Send Low or high bit
  .
  .
  .
  .
  BSR IICNoAck

IICAck:
  BCLR SCL, PTAD            ;Restore clock line
  BSET SDA, PTADD           ;SDA back as output
  EXIT_CODE 0
  RTS

Label the subroutine with the level so you can manage the level rules easily;

- Main program can call level 0,1,2…n
  - Level 0 can call level 1,2…n
  - Level 1 can call level 2,3…n
- Subroutines which do not call other subroutines do not need to use level macro
- Calling subroutines at same level would destroy the link
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.