

AN1820

Software I²C Communications

By Brad Bierschenk MMD Applications Engineering Austin, Texas

Introduction

The I²C (inter-integrated circuit) protocol is a 2-wire serial communications interface, implemented on numerous microcontrollers and peripheral devices. Many MCUs (microcontroller units) do not have an I²C module, yet they are required to communicate to 2-wire, or I²C, devices.

This application note describes a method of communicating on an I²C bus by controlling digital input/output (I/O) pins. This "bit-banged" method can be implemented on any Freescale MCU.

I²C Overview

I²C is a 2-wire communications link, requiring a clock line (SCK) and a data line (SDA) to communicate. The frequency of the I²C clock can go up to 100 Kbits per second for standard mode, and up to 400 Kbits per second for fast mode.

An I²C bus has both master devices and slave devices attached to it. A master is defined as a device which initiates a transfer, generates clock signals, and terminates a transfer. A slave device is simply a device

© Freescale Semiconductor, Inc., 2004. All rights reserved.





Application Note

addressed by a master. I²C provides for multiple masters on the same bus. The I²C also provides some error checking by acknowledgment bits during byte transfers.

The application presented in this document illustrates a limited version of the I²C specification. It is not intended to implement all the features of an I²C bus. It only provides the basic functionality required to transmit as a master device to slave devices through a 2-wire interface. The advantage of this method is it uses standard digital input/output pins available on any Freescale MCU.

The application presented here provides the following functionality:

- 7-bit addressing
- Single master transmitter
- Multiple data bytes within a serial transfer
- Serial clock frequency of approximately 28 kHz (arbitrary)
- Acknowledgment polling for error checking

By controlling two digital I/O pins, one can simulate an I²C transfer. When the I/O pins are CMOS and not open-drain, some safegaurds have to be implemented. A series resistor should be used between the CMOS output pin and the receiver's input pin. This will provide some current limiting should the two devices attempt to output conflicting logic levels.

The other consideration is supporting a logic high for any open-drain receiver pins. A pullup resistor can be used at the receiver's open-drain pin to passively pullup to the supply voltage, when the pin is not being actively driven low. This pullup resistor should be carefully chosen, so that when the master pin drives low, a valid $V_{\rm IL}$ level is presented to the $\rm I^2C$ receiver's pin.

The diagram shown in **Figure 1** illustrates a way to connect digital I/O pins to an external I²C receiver device. In this case, a MC68HC705J1A microcontroller is connected to a Maxim MAX517 DAC (Digital-to-Analog Converter). The MAX517 has a 2-wire interface that is I²C compatible. The MC68HC705J1A has CMOS bidirectional input/output

AN1820



Application Note I²C Overview

pins. When connected as shown, successful I²C communications can be made to the external IC.

An I²C transfer is composed of specific stages, defined by the states of the two wires. **Figure 2** shows the timing between the clock and data lines. To signal the beginning of a transmission, a START condition is presented to the bus. This START condition is indicated by a falling edge on SDA, while SCK is held high.

Once the START condition has been driven, the master device places a 7-bit address on the bus, with its most significant bit first. This address corresponds to the address of the I²C device the transfer is intended for. The eighth bit following the 7-bit address can be high or low, depending on whether it is a "read" or "write" operation.

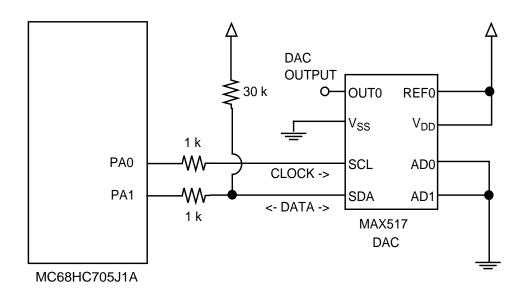


Figure 1. Hardware Diagram

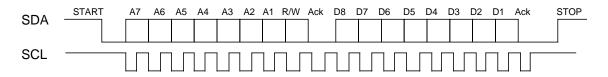


Figure 2. Example of I²C Transfer Timing

AN1820



Application Note I²C Overview

As with all bytes transferred on the I²C bus, a ninth clock cycle is used as an acknowledgment. The SDA line is read during this ninth clock cycle and signifies whether or not the byte is acknowledged. The receiver will drive 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 is imposed on the I²C bus. The STOP condition is indicated by a rising edge on SDA, while the SCK line is held high.

NOTE: To avoid unwanted START or STOP conditions, the software must transition the SDA pin only while the SCK line is held low.

A listing of assembly code that shows a specific implementation of I²C in software follows this text. This application does require some software overhead, but is somewhat interruptible as the I²C bus is completely synchronous. An implementation that requires less software overhead could be created using a more automated timing source, such as a free-running counter or real-time interrupt.

The code shows how a MC68HC705J1A microcontroller can be connected to an I²C peripheral, in this case a Maxim MAX517 DAC. The software continuously sends a write command to the DAC, ramping the digital value for the DAC from \$00 to \$FF and back down again. This creates a triangular wave at the output of the DAC.

The point is not to show a completely useful DAC application, but to illustrate the use of digital input/output pins as an I²C master device.



Application Note Code Listings

Code Listings

```
* TRIANGLE.ASM
* Purpose: Test of I2C bit-banging using the J1A
* Target: 705J1A
* Author: Brad Bierschenk, MMD Applications
* Tested using Maxim I^2C DAC IC, MAX517
* Has a "2-wire interface" (another word for I^2C)
* This code continuously sends 8-bit data to the
* Digital to Analog IC, incrementing from $00 to
* $FF, and back down again. This creates a
* triangular waveform at the output of the DAC chip.
* The SCL frequency is approximately 28 kHz. This is
* completely arbitrary.
* Assembler Equates
* -----
RAMSPACE EOU $C0
                         ;RAM start address
ROMSPACE EQU $300
PORTA EQU $00
                         ;EPROM start address
                          ;Port A
PORTB EQU $01
DDRA EQU $04
DDRB EQU $05
                         ;Port B
                         ;Data direction A
                         ;Data direction B
* Emulated I2C lines on Port A pins
* Need a clock (SCL) and data (SDA)
SCL EQU 0
                     ;Serial clock
SDA EQU
         1
                          ;Serial data
DACADDR EQU $2C
                          ;Slave address of DAC
* RAM Variables
* -----
        ORG RAMSPACE
BitCounter RMB 1
                          ;Used to count bits in a Tx
Value RMB 1
                          ;Used to store data value
Direction RMB 1
                          ; Indicates increment or
                          ;decrement
* Start of program code
* ______
     ORG ROMSPACE
                          ;Start of EPROM
Start:
     ;Initialize variables
                         ;Clear all RAM variables
     CLR Value
          BitCounter
     CLR
          Direction
     CLR
     ;Setup parallel ports
                          ;PAO and PA1 as outputs
     LDA #$03
     STA
          PORTA
                          driven high to start
         DDRA
     STA
```



Application Note

```
______
* This main loop just ramps up and down the data
* value that is sent to the DAC chip.
TxLoop:
                               ;Increment or decrement?
      LDA
            Direction
      BEQ
            GoUp
GoDown:
      LDA
            Value
                               ;Decrement
                               ; Change direction if needed
      BNE
            GD2
      CLR
            Direction
      BRA
            SendIt
GD2:
      DEC
            Value
                               ;Decrement the data value
            SendIt
      BRA
GoUp:
      LDA
            Value
                               ;Increment
      CMP
            #$FF
                               ; Change direction if needed
      BNE
            GU2
      TNC
            Direction
                               ; Increment the data value
            SendIt
      BRA
GU2:
      INC
            Value
* Send the I<sup>2</sup>C transmission, including START, address,
* data, and STOP
SendIt:
      ;START condition
                               ; Give START condition
            I2CStartBit
      ;ADDRESS byte, consists of 7-bit address + 0 as LSbit
                              ;Slave device address
      LDA
            #DACADDR
      ASLA
                               ; Need this to align address
      JSR
            I2CTxByte
                               ;Send the eight bits
      ;DATA bytes
            #$00
                               ;$00 is command byte for DAC
      LDA
      JSR
            I2CTxByte
                               ;Send the 8 bits
      LDA
            Value
                               ; Value is value to set DAC
                               ;Send it
      JSR
            I2CTxByte
      ;STOP condition
            I2CStopBit
                               ; Give STOP condition
      JSR
            I2CBitDelay
                               ;Wait a bit
      BRA
            TxLoop
                               ;Repeat
; I2CTxByte
; Transmit the byte in Acc to the SDA pin
; (Acc will not be restored on return)
; Must be careful to change SDA values only while SCL is low,
; otherwise a STOP or START could be implied
I2CTxByte:
      ;Initialize variable
            #$08
      LDX
      STX
            BitCounter
```

AN1820



Application Note Code Listings

```
I2CNextBit:
      ROLA
                              ;Shift MSbit into Carry
      BCC
            SendLow
                              ;Send low bit or high bit
SendHigh:
      BSET
            SDA, PORTA
                              ;Set the data bit value
            I2CSetupDelay
                              ; Give some time for data
      JSR
setup
      BSET
            SCL, PORTA
                              ;Clock it in
      JSR
            I2CBitDelay
                              ;Wait a bit
                              ;Continue
      BRA
            I2CTxCont
SendLow:
      BCLR
            SDA, PORTA
            I2CSetupDelay
      JSR
      BSET
            SCL, PORTA
            I2CBitDelay
      JSR
I2CTxCont:
                              ; Restore clock to low state
      BCLR
            SCL, PORTA
      DEC
            BitCounter
                              ;Decrement the bit counter
      BEQ
            I2CAckPoll
                              ;Last bit?
      BRA
            I2CNextBit
I2CAckPoll:
      BSET
            SDA, PORTA
      BCLR
            SDA, DDRA
                              ;Set SDA as input
      JSR
            I2CSetupDelay
      BSET
            SCL, PORTA
                              ;Clock the line to get ACK
      JSR
            I2CBitDelay
      BRSET
            SDA, PORTA, I2CNoAck
                               ;Look for ACK from slave
                               ;device
      BCLR
                               ;Restore clock line
            SCL, PORTA
      BSET
            SDA, DDRA
                               ;SDA back as output
      RTS
      ; No acknowledgment received from slave device
      ;Some error action can be performed here
      ;For now, just restore the bus
I2CNoAck:
      BCLR
            SCL, PORTA
      BSET
            SDA, DDRA
      RTS
; A START condition is defined as a falling edge
; on SDA while SCL is high
I2CStartBit:
      BCLR
            SDA, PORTA
      JSR
            I2CBitDelay
      BCLR
            SCL, PORTA
      RTS
; A STOP condition is defined as a rising edge
; on SDA while SCL is high
I2CStopBit:
      BCLR
            SDA, PORTA
      BSET
            SCL, PORTA
      BSET
            SDA, PORTA
      JSR
            I2CBitDelay
      RTS
```



Application Note

```
; Provide some data setup time to allow
; SDA to stabilize in slave device
; Completely arbitrary delay (10 cycles)
I2CSetupDelay:
    NOP
    NOP
    RTS
; Bit delay to provide (approximately) the desired
; SCL frequency
; Again, this is arbitrary (16 cycles)
I2CBitDelay:
    NOP
    NOP
    NOP
    NOP
    NOP
    RTS
 Vector Definitions
         $07FE
    ORG
                        ;Reset vector
    FDB
         Start
```

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 which 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.

