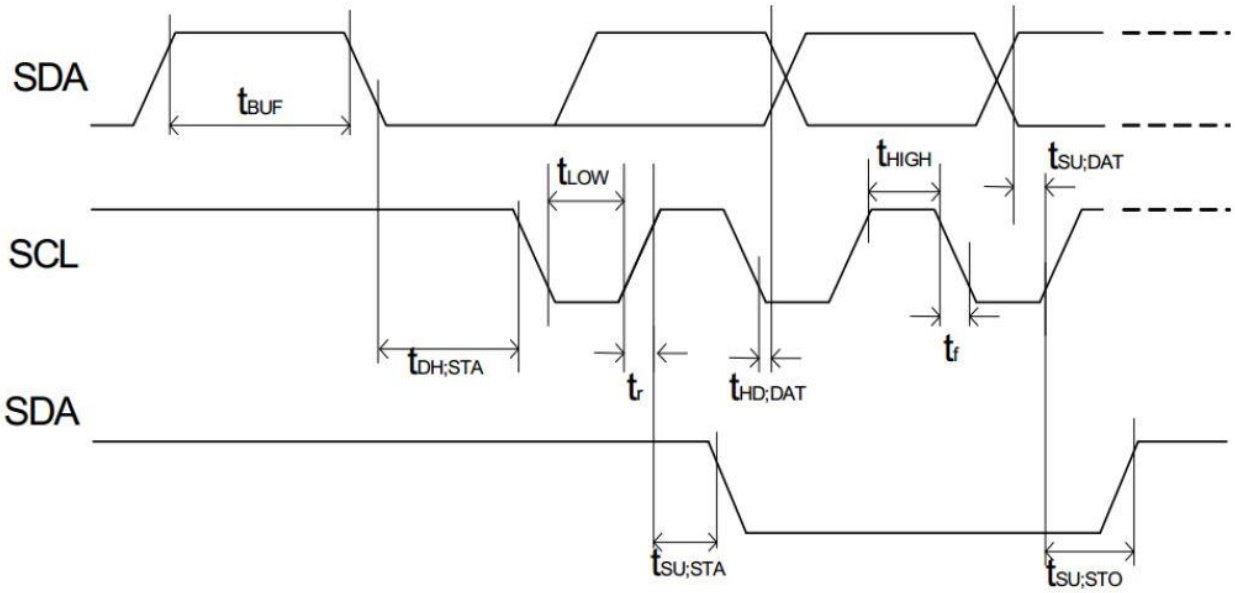
I2C DRIVER FOR NEWHAVEN DISPLAY



Artem Kulakevich

Douglas Hall

Portland State University

ECE372 – 21 March 2019

I2C Part 1 – Polling

TASK LIST

-Go through I2C guide in reference manual

-Find necessary registers for initialization

-Find necessary bits for initialization

-Go through LCD Control module

-Find necessary words for initialization

-Implement polling

Initialization Data (Shared for Part 1 and 2):	Data Send Data
00h: Array Place holder 00h: Command Send 38h: Function Set 39h: Function Set 14h: Bias Set 78h: Contrast Set, 5Eh: Power/Icon contrast 6Dh: Follower Control 0Ch: Display ON 01h: Clear Display 06h: entry mode set	00h: Array Place Holder 40h: Data Send ASCII 'A' ASCII 'r' ASCII 't' ASCII 'e' ASCII 'm'

High Level Algorithm

MAINLINE

1. Remap mux pins for I2C1
2. Enable I2C1 modules
3. Soft reset I2C1 for troubleshooting
4. Setup prescaler value to divide 48Mhz clock
5. Setup tLOW value for SCLL
6. Setup tHIGH value for SCLH
7. Configure own address
8. Take the module out of reset and enable for transmit
9. Setup slave address
10. Load data send counter with value for initialization
11. Branch to PollBB()
12. Assert Start and future stop condition
13. Branch to send_init()

14. Load data send counter with new value
15. Branch to PollBB()
16. Assert Start and future stop condition
17. Branch to send_data()

SEND_INIT

1. Load initialization data to be sent into array
2. Load array pointer/counter
3. Repeat until counter equals “data send counter” value from earlier
    a. Branch to pollXDRY()
    b. Load data from memory into I2C1 data buffer
    c. Branch to delay()
    d. Increment pointer/counter

SEND_DATA

1. Load data to be sent into array
2. Load array pointer/counter
3. Repeat until counter equals “data send counter” value from earlier
    a. Branch to pollXDRY()
    b. Load data from memory into I2C1 data buffer
    c. Branch to delay()

POLLBB

1. Test BB bit of I2C1
    a. If HIGH, then test again
    b. If LOW, then leave (Bus is not busy)

POLLXDRY

1. Test XDRY bit of I2C1
    a. If HIGH, then leave (Bus is ready for more data)
    b. If LOW, then test again

DELAY

1. Setup counter register
2. If counter register < final counter value
    a. Increment by 1
    b. Else leave

Low Level Algorithm

MAINLINE

1. Remap mux pins for I2C1
    a. **Store 0x7A in conf_spi0_cs0 register at 0x44E10000+0x958**
    b. **Store 0x7A in conf_spi_d1 register at 0x44E10000+0x95C**
2. Enable I2C1 modules
    a. **Store 0x02 in CM_PER_CLOCK_I2C1 at 0x44E00000+0x48**
3. Soft reset I2C1 for troubleshooting
    a. **Store 0x2 in I2C1_SYSC at 0x4802A000+0x10**
4. Setup prescaler value to divide 48Mhz clock
    a. **Store 0x3 in I2C1_PSC at 0x4802A000+0xB0**
5. Setup tLOW value for SCLL
    a. **Store 0x35 in I2C1_SCLL at 0x4802A000+0xB4**
6. Setup tHIGH value for SCLH
    a. **Store 0x37 in I2C1_SCLH at 0x4802A000+B8**
7. Configure own address
    a. **Store 0x00 in I2C1_OA at 0x4802A000+A8**
8. Take the module out of reset and enable for transmit
    a. **Store 0x8600 in I2C1_CON at 0x4802A000+0xA4**
9. Setup slave address
    a. **Store 0x3C in I2C1_SA at 0x4802A000+0xAC**
10. Load data send counter with value for initialization
    a. **Store 0xA in I2C1_CNT at 0x4802A000+0x98**
11. Branch to PollBB()
12. Assert Start and future stop condition
    a. **Store 0x8603 in I2C1_CON at 0x4802A0000xA4**
13. Branch to send_init()
14. Load data send counter with new value
    a. **Store 0x6 in I2C1_CNT at 0x4802A000+0x98**
15. Branch to PollBB()
16. Assert Start and future stop condition
    a. **Store 0x8603 in I2C1_CON at 0x4802A000+0xA4**
17. Branch to send_data()

SEND_INIT

4. Load initialization data to be sent into array
5. Load array pointer/counter
6. Repeat until counter equals “data send counter” value from earlier
    a. Branch to pollXDRY()
    b. Load data from memory into I2C1 data buffer

        i. **Store data in I2C1_DATA at 0x4802A000+0x9C**
    c. Branch to delay()
    d. Increment pointer/counter

SEND_DATA

4. Load data to be sent into array
5. Load array pointer/counter
6. Repeat until counter equals "data send counter" value from earlier
    a. Branch to pollXDRY()
    b. Load data from memory into I2C1 data buffer
        i. **Store data in I2C1_Data at 0x4802A000+0x9C**
    c. Branch to delay()

POLLBB

2. Test BB bit of I2C1
    a. **Test bit 12 (0x1000) in I2C1_IRQSTATUS_RAW at 0x4802A000+0x24**
    b. If HIGH, then test again
    c. If LOW, then leave (Bus is not busy)

POLLXDRY

2. Test XDRY bit of I2C1
    a. **Test bit 4 (0x10) in I2C1_IRQSTATUS_RAW at 0x4802A000+0x24**
    b. If HIGH, then leave (Bus is ready for more data)
    c. If LOW, then test again

DELAY

3. Setup counter register
4. If counter register < final counter value
    a. Increment by 1
    b. Else leave

I2C Part 2 – Interrupts

Task List

-Modify polling I2C program to allow for interrupts

-Go through datasheet and find initialization interrupts

High Level Algorithm

MAINLINE

1. Remap mux pins for I2C1
2. Enable I2C1 modules
3. **MOD:** Unmask bit for MIR_Clear for I2C1INT (71)
4. Soft reset I2C1 for troubleshooting
5. Setup prescaler value to divide 48Mhz clock
6. Setup tLOW value for SCLL
7. Setup tHIGH value for SCLH
8. Configure own address
9. **MOD**: Enable interrupt mask
10. Setup slave address
11. Load data send counter with value for initialization
12. Branch to PollBB()
13. Assert Start and future stop condition
14. **MOD:** Enable IRQ input by clearing bit 7
15. Wait for interrupt

INT_HANDLER

1. Save register on stack
2. Test bit for I2C1INT in MIR
    a. If bit is set, test for xdry
        i. If set, then branch to XDRY_int()
        ii. If clear, then branch to pass_on()
    b. If bit is clear, then branch to pass_on()

XDRY_INT

1. Reset I2C1INT interrupt
2. Test current_state variable
    a. If current_state = 1, branch to send_init()
    b. If current_state = 2, branch to send_data()

    c. If current_state = 3, disable interrupts
3. Branch to pass_on()

SEND_INIT()

1. Setup data array for initialization
2. Setup static pointer for array
3. Send out data for I2C1 data buffer, increment pointer
4. If pointer = data counter value
    a. Write 2 to current_state variable
    b. Write next DCOUNT value to I2C1 counter register
    c. Branch to PollBB()
    d. Assert start condition
5. pass_on()

SEND_DATA()

1. Setup data array for character display
2. Setup static pointer for array
3. Send out data for I2C1 data buffer, increment pointer
4. If pointer = data counter value
    a. Write 3 to current_state variable
    b. Disable interrupts
5. pass_on

PASS_ON

1. Reset register for NEWIRQ generation
2. Restore registers from stack and SUBS #4 outta there

POLLBB

1. Test BB bit of I2C1
    a. If HIGH, then test again
    b. If LOW, then leave (Bus is not busy)

Low Level Algorithm

MAINLINE

16. Remap mux pins for I2C1
    a. **Store 0x7A in conf_spi0_cs0 register at 0x44E10000+0x958**
    b. **Store 0x7A in conf_spi_d1 register at 0x44E10000+0x95C**
17. Enable I2C1 modules
    a. **Store 0x02 in CM_PER_CLOCK_I2C1 at 0x44E00000+0x48**
18. **MOD:** Unmask bit for MIR_Clear for I2C1INT (71)
    a. **Store 0x80 in INTC_MIR_CLEAR2 at 0x48200000+0xC8**
19. Soft reset I2C1 for troubleshooting
    a. **Store 0x2 in I2C1_SYSC at 0x4802A000+0x10**
20. Setup prescaler value to divide 48Mhz clock
    a. **Store 0x3 in I2C1_PSC at 0x4802A000+0xB0**
21. Setup tLOW value for SCLL
    a. **Store 0x35 in I2C1_SCLL at 0x4802A000+0xB4**
22. Setup tHIGH value for SCLH
    a. **Store 0x37 in I2C1_SCLH at 0x4802A000+B8**
23. Configure own address
    a. **Store 0x00 in I2C1_OA at 0x4802A000+A8**
24. **MOD**: Enable interrupt mask
    a. **Store 0x10 in I2C1_IRQENABLE_SET at 0x4802A000+0x2C**
25. Setup slave address
    a. **Store 0x3C in I2C1_SA at 0x4802A000+0xAC**
26. Load data send counter with value for initialization
    a. **Store 0xA in I2C1_CNT at 0x4802A000+0x98**
27. Branch to PollBB()
28. Assert Start and future stop condition
    a. **Store 0x8603 in I2C1_CON at 0x4802A000+0xA4**
29. **MOD:** Enable IRQ input by clearing bit 7
    a. **BIC #0x80**
30. Wait for interrupt

INT_HANDLER

3. Save register on stack
4. Test bit for I2C1INT in MIR
    a. **Test bit 7 (0x80) in INTC_PENDING_IRQ2 at 0x48200000+0xD8**
    b. If bit is set, test for xdry
        i. **Test bit 4 (0x10) in I2C1_IRQSTATUS at 0x4802A000+0x28**
        ii. If set, then branch to XDRY_int()
        iii. If clear, then branch to pass_on()

c. If bit is clear, then branch to pass_on()

XDRY_INT

4. Reset I2C1INT interrupt
    a. **Store 0x10 in I2C_IRQSTATUS at 0x4802A000+0x28**
5. Test current_state variable
    a. If current_state = 1, branch to send_init()
    b. If current_state = 2, branch to send_data()
    c. If current_state = 3, disable interrupts
6. Branch to pass_on()

SEND_INIT()

6. Setup data array for initialization
7. Setup static pointer for array
8. Send out data for I2C1 data buffer, increment pointer
    a. **Store data value in I2C1_DATA at 0x4802A000+9C**
9. If pointer = data counter value
    a. Write 2 to current_state variable
    b. Write next DCOUNT value to I2C1 counter register
        i. **Store 0x6 in I2C1_CNT at 0x4802A000+0x92**
    c. Branch to PollBB()
    d. Assert start condition
10. pass_on()

SEND_DATA()

6. Setup data array for character display
7. Setup static pointer for array
8. Send out data for I2C1 data buffer, increment pointer
    a. **Store data value in I2C1_DATA at 0x4802A000+9C**
9. If pointer = data counter value
    a. Write 3 to current_state variable
    b. Disable interrupts
        i. **Store 0x10 in I2C1_IRQENABLE_CLEAR at 0x4802A000+0x30**
10. pass_on

PASS_ON

3. Reset register for NEWIRQ generation
    a. **Store 0x01 at INTC_Contol at 0x48200000+0x48**
4. Restore registers from stack and SUBS #4 outta there

POLLBB

2. Test BB bit of I2C1
    a. **Test bit 12 (0x1000) in I2C1_IRQSTATUS_RAW at 0x4802A000+0x24**
    b. If HIGH, then test again
    c. If LOW, then leave (Bus is not busy)

I developed and wrote this program by myself with no help from anyone except the instructor and/or the T.A. and that I did not give any help to anyone else.

X_____