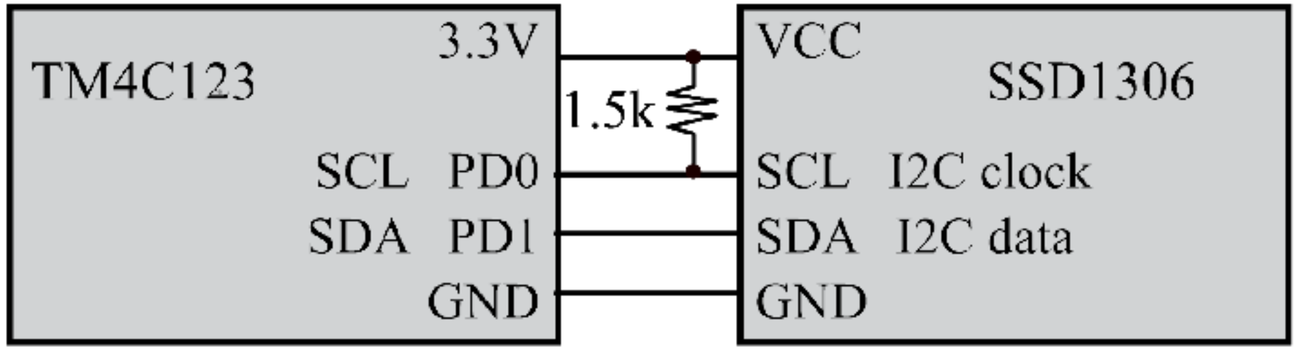
Part A: Build H/W



Test H/W by running the project.

You should test the hardware by running the SSD1306_4C123 starter project.

Part B : I2C Driver

```
; sends two bytes to specified slave
; Input: R0  7-bit slave address
;        R1  first 8-bit data to be written.
;        R2  second 8-bit data to be written.
; Output: 0 if successful, nonzero (error code) if error
; Assumes: I2C3 and port D have already been initialized and enabled
I2C_Send2
; --UUU--
; 1) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 2) write slave address to I2C3_MSA_R,
;   MSA bits7-1 is slave address
;   MSA bit 0 is 0 for send data
; 3) write first data to I2C3_MDR_R
; 4) write 0x03 to I2C3_MCS_R,  send no stop, generate start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 5) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 6) check for errors, if any bits 3,2,1 I2C3_MCS_R is high
;  a) if error set I2C3_MCS_R to 0x04 to send stop
;  b) if error return R0 equal to error bits 3,2,1 of I2C3_MCS_R
; 7) write second data to I2C3_MDR_R
; 8) write 0x05 to I2C3_MCS_R, send stop, no start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 9) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 10) return R0 equal to bits 3,2,1 of I2C3_MCS_R, error bits
;   will be 0 is no error
```

Start Here

i) Load I2C3_MCS_R in reg

ii) Load data at location

iii) Extract bit 0

iv) compare to 0: if not 0 loop back to start

```
; sends two bytes to specified slave
; Input: R0  7-bit slave address
;        R1  first 8-bit data to be written.
;        R2  second 8-bit data to be written.
; Output: 0 if successful, nonzero (error code) if error
; Assumes: I2C3 and port D have already been initialized and enabled
I2C_Send2
; --UUU--
; 1) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 2) write slave address to I2C3_MSA_R,
;   MSA bits7-1 is slave address
;   MSA bit 0 is 0 for send data
; 3) write first data to I2C3_MDR_R
; 4) write 0x03 to I2C3_MCS_R,  send no stop, generate start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 5) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 6) check for errors, if any bits 3,2,1 I2C3_MCS_R is high
;  a) if error set I2C3_MCS_R to 0x04 to send stop
;  b) if error return R0 equal to error bits 3,2,1 of I2C3_MCS_R
; 7) write second data to I2C3_MDR_R
; 8) write 0x05 to I2C3_MCS_R, send stop, no start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 9) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 10) return R0 equal to bits 3,2,1 of I2C3_MCS_R, error bits
;   will be 0 is no error
```

(2)
i) Left shift R0
ii) Make sure bit 0 of R0 is 0
iii) Load MSA_R loc in reg
iv) Store R0 in MSA reg

(3)
i) load MDR_R loc in reg
ii) Store R1 in MDR reg

(4)
i) mov 0x03 in reg
ii) load MCS_R loc in reg
iii) Store reg in MCS reg

```
; sends two bytes to specified slave
; Input: R0  7-bit slave address
;        R1  first 8-bit data to be written.
;        R2  second 8-bit data to be written.
; Output: 0 if successful, nonzero (error code) if error
; Assumes: I2C3 and port D have already been initialized and enabled
I2C_Send2
; --UUU--
; 1) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 2) write slave address to I2C3_MSA_R,
;   MSA bits7-1 is slave address
;   MSA bit 0 is 0 for send data
; 3) write first data to I2C3_MDR_R
; 4) write 0x03 to I2C3_MCS_R,  send no stop, generate start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 5) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 6) check for errors, if any bits 3,2,1 I2C3_MCS_R is high
;  a) if error set I2C3_MCS_R to 0x04 to send stop
;  b) if error return R0 equal to error bits 3,2,1 of I2C3_MCS_R
; 7) write second data to I2C3_MDR_R
; 8) write 0x05 to I2C3_MCS_R, send stop, no start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 9) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 10) return R0 equal to bits 3,2,1 of I2C3_MCS_R, error bits
;   will be 0 is no error
```

(4) iv) NOP (x4)

(5) Same as (1)

(6) i) use MCS loaded in (5)
ii) Right shift by 1
iii) Extract bit 2,1,0 (0x07)
iv) Compare with 0
if not 0, Error
else continue

(Error) : i) move error code to R0
ii) move 0x04 to reg
iii) load mcs_R loc in reg
iv) Store 0x04 to mcs reg
v) return (BX LR)

```
; sends two bytes to specified slave
; Input: R0  7-bit slave address
;        R1  first 8-bit data to be written.
;        R2  second 8-bit data to be written.
; Output: 0 if successful, nonzero (error code) if error
; Assumes: I2C3 and port D have already been initialized and enabled
I2C_Send2
; --UUU--
; 1) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 2) write slave address to I2C3_MSA_R,
;   MSA bits7-1 is slave address
;   MSA bit 0 is 0 for send data
; 3) write first data to I2C3_MDR_R
; 4) write 0x03 to I2C3_MCS_R,  send no stop, generate start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 5) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 6) check for errors, if any bits 3,2,1 I2C3_MCS_R is high
;  a) if error set I2C3_MCS_R to 0x04 to send stop
;  b) if error return R0 equal to error bits 3,2,1 of I2C3_MCS_R
; 7) write second data to I2C3_MDR_R
; 8) write 0x05 to I2C3_MCS_R, send stop, no start, enable
; add 4 NOPs to wait for I2C to start transmitting
; 9) wait while I2C is busy, wait for I2C3_MCS_R bit 0 to be 0
; 10) return R0 equal to bits 3,2,1 of I2C3_MCS_R, error bits
;   will be 0 is no error
```

(7) → Same as (3)

(8) → Same as (4) but write (0x05)

(9) → Same as (5)

(10) → Same as (6)
if no error
mov R0 in RC
return

Part C: Out string fn

```
void SSD1306_OutString(char *ptr){
}
```

→ null terminated array of char

null = '\0'

ptr to a string

int i as index

loop(i=0 → ptr[i]==='\0'; i=i+1)

→ SSD1306_OutChar(ptr[i]);

Part D: IO Functions

```
#define PF2 (*((volatile unsigned long *)0x40025010)) // PF2
#define PF4 (*((volatile unsigned long *)0x40025040))  // PF4

void IO_Init(void) { volatile uint32_t delay;
 // --UUU-- Code to initialize PF4 and PF2
  SYSCTL_RCGCGPIO_R =            //1) activate clock for Port
 delay = SYSCTL_RCGCGPIO_R;      // allow time for clock to start
  GPIO_PORTF_DIR_R =             // 2) PF4 in, PF2 out
  GPIO_PORTF_PUR_R =             // 3) Set pullup register for PF4
  GPIO_PORTF_DEN_R =             // 4) enable digital I/O on PF4,2
}
```

→ Set bit 5

→ Set for output (=1)
Reset for input (=0)

→ Set bit 4

→ Set for bit 2,4

```
uint8_t led = 0x04;
void IO_HeartBeat(void) {
 // --UUU--
    led ^= ;
    PF2 = ;
    Clock_Delay1ms(100);
}
```

→ Initialize a variable

→ toggle value

→ store value to port

→ use this function for a delay

```
//------------IO_Touch------------
// wait for release and press of
the switch
// Input: none
// Output: none
// 1) wait for release;
// 2) delay for 5ms;
// 3) wait for press; and then
// 4) delay for another 5ms
```

→ wait while PF4 is 0

→ use delay func

→ wait while PF4 is 1

→ use delay func

Part E: LCD Driver (OutDec)

N EQU 4 ← The number to get units place for
CNT EQU 0 ← Count of numbers
FP [RN 1] ← Frame pointer
(on top of code)

Read bits:

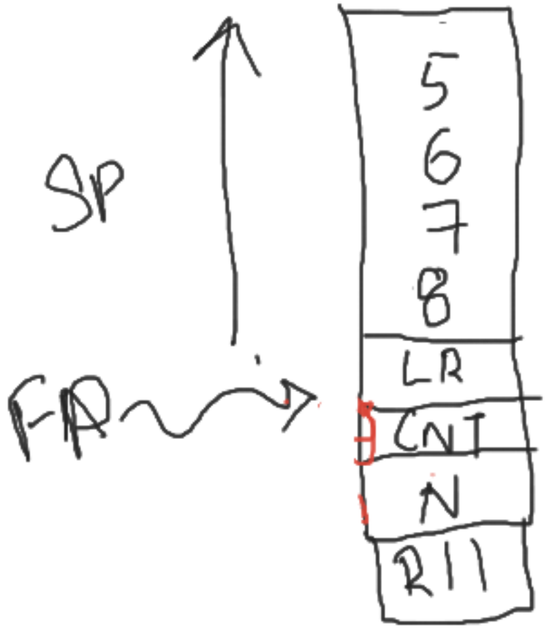
Init

N = 5678 → 567 → 56 → 5 → 0

CNT = 0 → 1 → 2 → 3 → 4

[SP] = 8 → 7 → 6 → 5

Check for N >= 0



→ Write Char

[SP] = 5 → 6 → 7 → 8

CNT = 4 → 3 → 2 → 1 → 0

Check CNT >= 0

Exit

OutDec Algorithm: (Using recursion)

LCD_OutDec
- Store R11 on stack
- Store number N on Stack
- make space for count (subtract SP by 4) ← CNT
- make SP your frame pointer

(INIT 1)

INIT
- store LR on Stack
- store CNT as 0
- load a reg with #10 for division

How to access a var?

```
LDR R0,[FP,#CNT]
STR R0,[FP,#CNT]
```

← CNT is defined as 0 in last slide

Read loop

① – Load CNT to reg
– add 1 to reg
– Store reg as CNT

load CNT, add 1, store

---

② – Load current value of N to reg1
– make a copy in reg2
– Unsigned divide reg1 by 10 in reg1
– Store this reg1 as new N
– multiply this reg1 to 10 in reg1
– Substract reg1 from reg2 (reg2 - reg1)
– Store this difference to stack

load N
N1, N2.
N1/10 → new N
extracted digit
push digit on stack

③ - Load new N
- Check if N is 0
- NO? Loop back to read loop
YES? continue

Write loop

④ - POP value from Stack
- Add to 0x30 and store in R0
- Branch to SSD1306_OutChar {BL}

⑤ - Load CNT to reg1
- Decrement
- Store CNT from reg1

⑥ - Check if CNT (reg1) greater than 0
- YES? Loop back to write loop
- NO? continue

⑦ - Store LR from stack
- ADD #8 to SP
- POP R11 from Stack
- Return BX LR

LCD_OUTFIX

- INIT 1 same as OUTDEC
- INIT2 same as OUTDEC

① - Check if N is less than 1000 (BLO)
- Yes? Jump to INRANGE
NO? continue

② - move 0x2A to R0 (*)
- Branch SSD1306_OutChar

③ - move 0x2E to R0
- Branch SSD1306_OutChar

④ - DO ②

⑤ - DO ②
- Jump to Exit

INRANGE
⑥ - same as ① in OUTDEC

⑦ - same as ② in OUTDEC

⑧ - Load CNT to reg
- Compare CNT to 3
- Less than? Loop to INRANGE
NO? Continue

---

⑨ - Same as ④ in OutDec
- Same as ② in OutFix
- Same as ④ in OutDEc
- Same as ④ in OutDEC

⑩
EXIT
- Store LR from stack
- ADD #8 to SP
- Store R11 from stack
- return (BX LR)