Exam Field 2 – sem 1 2012 – 2013

Q1 (a) (i)

Port mapped (PMIO) vs memory mapped (MMIO) I/O

- port map usually smaller
- PMIO needs special pins on CPU.
- PMIO needs special instructions to access ports
- because port map is smaller, addr decoding may be easier
- etc – others possible

4 marks

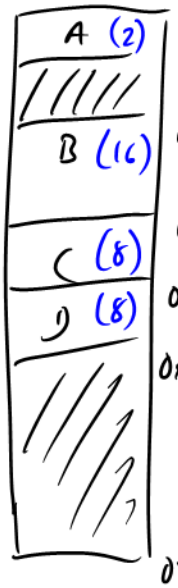
(ii) Device A 2 registers, 8bit Port map ⇒ 256 addresses
B 16 "
C 8 "
D 8 "

Need to choose appropriate addresses carefully

A bad solution – addr dec algebraic expressions would be complex

Good solution

start addr = multiple of size



6 Marks

Q1 (a) (iii)

A

$0b0000\ 0000 = 0x00$

$0b0000\ 0001 = 0x01$

$\overline{CS_A} = \overline{(\bar{A}_7 \bar{A}_6 \bar{A}_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1)}$ (X)

B

$0x10 = 0b0001\ 0000$

$0x1F = 0b0001\ 1111$

$0001\ xxxx$

$\overline{CS_B} = \overline{(\bar{A}_7 \bar{A}_6 \bar{A}_5 A_4)}$

10 marks

C

$0x20 = 0b0010\ 0000$

$0x27 = 0b0010\ 0111$

$0010\ 0xxx$

$\overline{CS_C} = \overline{(\bar{A}_7 \bar{A}_6 A_5 A_4 A_3)}$

D

$0x28 = 0b0010\ 1000$

$0x2F = 0b0010\ 1111$

$0010\ 1xxx$

$\overline{CS_D} = \overline{(\bar{A}_7 \bar{A}_6 A_5 \bar{A}_4 A_3)}$

Q1 (b) (i)

Fault — notes — eg bug in S/W not covered by testing and hasn't run yet

Controlled failures — notes — eg DVD player laser fails (anticipated by developers) — display error message to user

Uncontrolled failures — notes — eg DVD player user places DVD badly such the drawer jams and can't open or shut — engineers didn't anticipate this so no failure recovery or mitigation was designed.

8 marks

(ii) Watchdog — notes

Q2 (a) (i) RC2 low but no key pressed means all row lines to right of resistors are open circuit.

4 marks

Therefore circuit is V_{cc} — resistor — port input line.
Therefore port input is essentially connected to V_{cc} and reads as high. $RC_6 .. RC_3 = 1111$.

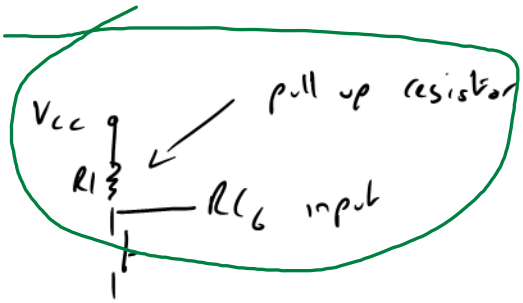
(ii) If 4 key pressed the circuit for RC3 is

$V_{cc} \rightarrow R_2 \rightarrow$ Low : Gnd $\Rightarrow$ RC3 reads low

4 marks

and RC_6 or $RC_3 = 1011$

(iii) Redrawing circuit for RC_6



4 marks

R_1 to R_4 act as pull up resistors when switches are open.

(When switch closed and one of RC_0 to RC_2 are driven low, the entire voltage is dropped across R_1 or R_2 or R_3 or R_4 and corresponding row input reads low.)

The resistor values (100kΩ) ensure that very little current flows when switches are closed. This ensures low power consumption.

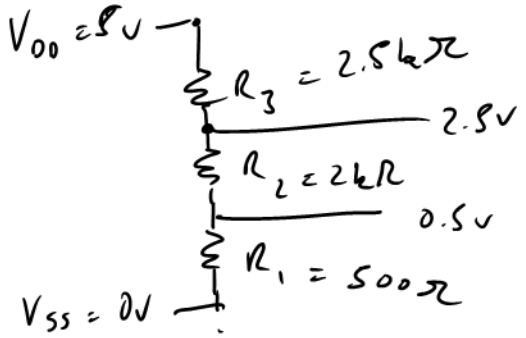
Q2 (b) something like..

```
scanKeys():
    for col = 0 to 2:
        portc bits 2:0 = col
        rowKeys = portC bits 6:3
        if (rowKeys == 1110) then row = 3, return // 0 based indexing
        else if (rowKeys == 1101) then row = 2, return
        " " " " = 1011 " " = 1, return
        " " " " = 0111 " " = 0, return
    // otherwise repeat for next column
col = NONE, row = NONE // no valid found
```

Q2 (c) (i) Num bits/sample — determines the resolution (number of voltage steps) in the input signal

Voltage reference levels — determines min and max voltages that can be sensed and hence the range and hence the size of each step

(ii) $V_{ref\,min} = 0.5V$, $V_{ref\,max} = 2.5V$ — using these we can sense entire range of input signal



Q 2 (c) (cont.)

(iii) 10 bit ADC $\Rightarrow 2^{10} = 1024$ steps

voltage range $= V_{ref_{max}} - V_{ref_{min}} = 2V$

$\Rightarrow$ step size (resolution) $= \frac{2}{1024} = 1.953mV$

(iv) Range is 0.5 - 2.5V. Step size = 1.953mV

$$num_V = 2^B \frac{V - V_{min}}{V_{max} - V_{min}} \Rightarrow num_2 = 2^{10} \frac{(2 - 0.5)}{(2.5 - 0.5)}$$

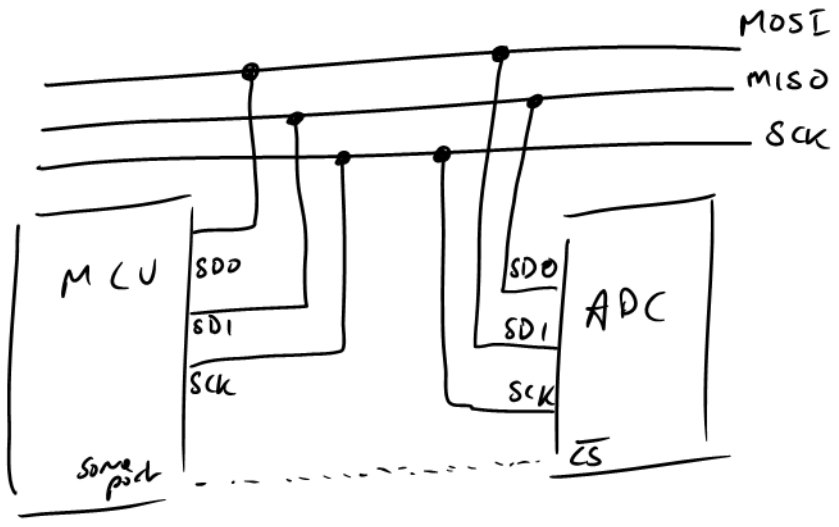
$$= 768$$

Q3 (a)

```
const T_BIT_US = 1000000 / 2400

init():
    configure port for 1 output, called tx
    set tx = MARK        // high

transmitChar(ch)
    set tx = SPACE       // start bit
    delay T_BIT_US
    for i: 0 to 7
        set tx = bit i of ch    // data, LSB first
        delay T_BIT_US
    // to parity, 2 stop bits
    set tx = MARK        // stop bit 1
    delay T_BIT_US
    set tx = MARK // stop bit 2
    delay T_BIT_US
```

Q3 (b) (i)

5 marks



Q3 (b) (ii) $T_{CLK} = 625ns$

$T_{sample} = 2 \times 8 \times T_{bit} + T_{CLK}$

in SPI $T_{bit} = T_{CLK}$

$\Rightarrow T_{sample} = 17\, T_{CLK} = 10625ns$

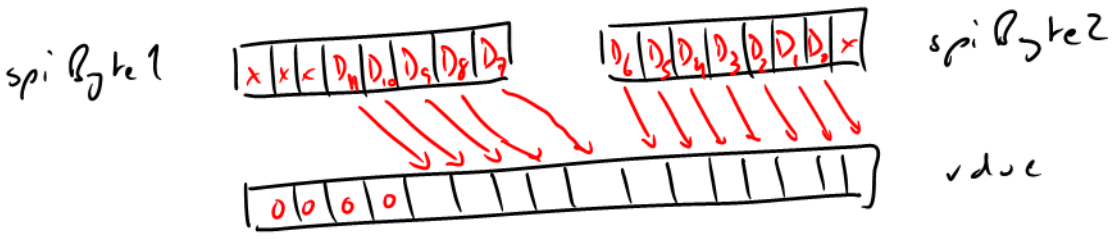
3 marks

Sample rate $= \frac{1}{T_{sample}} \approx 94118$ Samples/s

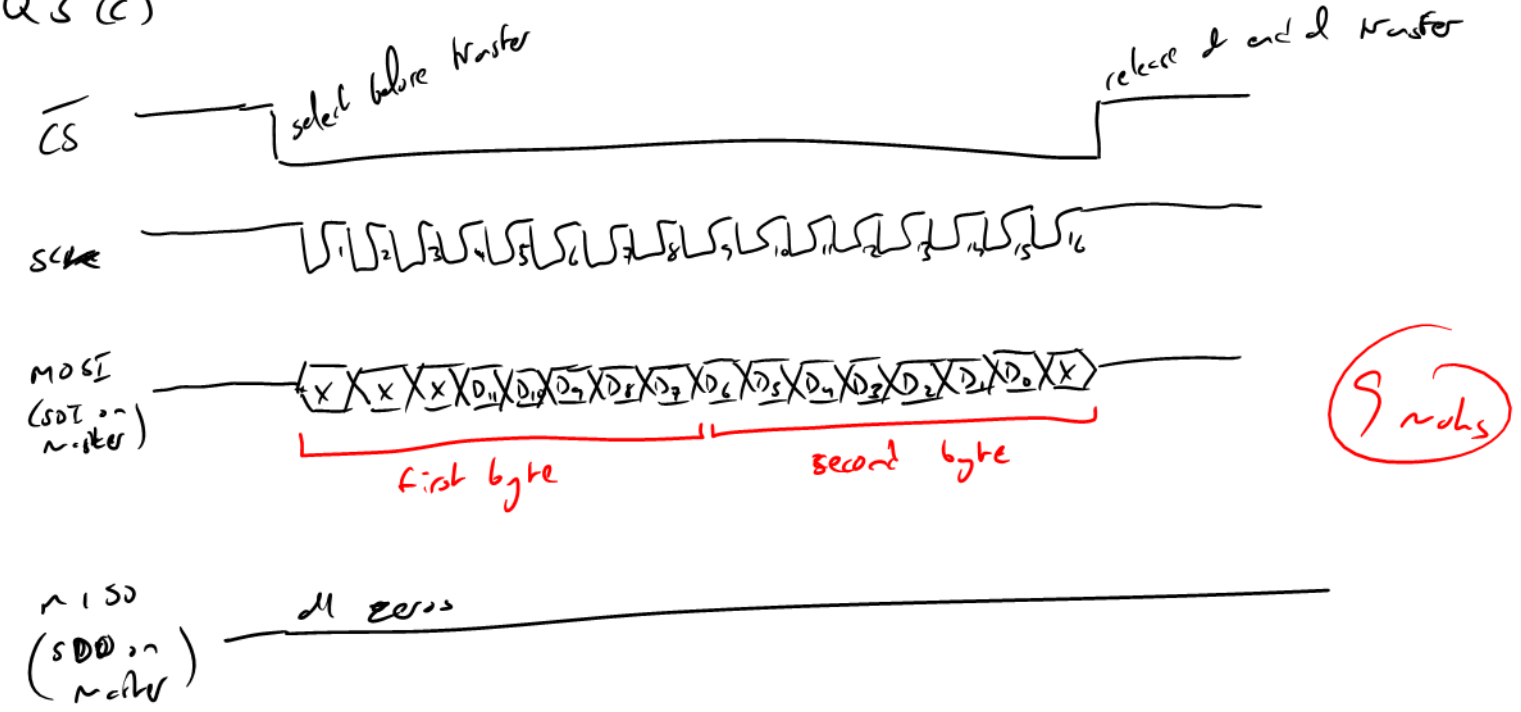
(iii)

```
int16 value;

value = spiByte1 & 0b000 11111; // grab 5 MSb (D11..D7) of data
                                // which are in bits 4:0 of spiByte1
value = value << 7; // left shift 7 places so that
                    bits 4..0 become bits 11..7

tmp = spiByte2 >> 1; // grab 7 LSb (D6..D0) of data
                     // which are in bits 7:1 of spiByte2
value = value | tmp; // value now has all bits D11..D0 of data
```

6 marks



Q3 (c)


CS
select before transfer
release at end of transfer
SCLK
MOSI
(SDI on
master)
X X X D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0 X
first byte
second byte
9 clks
MISO
(SDO on
master)
all zeros

Q4

(a) (i) — see notes 4.20, slide 15 5 marks

(ii) polled interrupts (as used by the PIC MCU)
has a single interrupt service routine and interrupt
priority is implemented in software by the order in
which we check devices.

```
interrupt isr():
    if (device1 interrupt flag)
        handle device1 interrupt()
    else if (device2 interrupt flag)
        handle device 2 interrupt
    else if (device 3 interrupt flag)
        handle device 3 interrupt
```

4 marks

(iii) Any 2 of these:

polling: - all I/O transfers controlled by software in the "foreground"
- simple to implement

2 marks

- usually software checks the hardware device at
regular intervals

polled interrupts:
- I/O transfers occur when device interrupts to ask for
service (in "background")
- more efficient than polling
- requires implementation of a single ISR
and may need care with variables shared
by ISR and superloop

Q4(b) (i) For max timeout, use max prescaler value

$F_{osc} = 10MHz \Rightarrow T_{osc} = \frac{1}{10MHz} = 100ns$

$\Rightarrow T_{CY} = 4 \times T_{osc} = 400ns$

with 8x prescaler, timer1 increments every $8 \times 400 = 3200ns$

max timeout for 16 bit timer is $2^{16} = 65536$ ticks

$= 65536 \times 3200ns$

$\approx 209.7ms$

4 marks

timer resolution is 1 tick = 3200 ns

timer settings : 8x prescaler

timer initial value = 0

(ii) TaskA 10 Hz $\Rightarrow$ 100ms period

TaskB 8 Hz $\Rightarrow$ 125ms period

3 marks common denominator time = 25ms — superloop tick time

$\Rightarrow$ TaskA = 4 ticks

TaskB = 5 ticks

(iii) We want a timeout of 25ms

Again $T_{CY} = 400ns$, max timer1 ticks is $2^{16} = 65536$

4 marks $\Rightarrow$ 1x prescaler : max timeout is $65536 \times 400 \times 1 \approx 26ms$ ✓

(this is bigger than 25ms so ok)

We want timer to expire in 25ms = $25000000^{ns}/_{400_{ns}} = 62500$ timer1 ticks

initial timer value = $2^B - N = 65536 - 62500 = 3036$

Q 4 (b) (iv) from part (iii) taskA needs to do its real work only every 4th superloop (because its period is 4 superloop ticks where each superloop tick is 25 ms)

7 mrks

```
taskA():
    static countdown = TASKA_TICKS // ie. 4 ticks

    decrement countdown
    if (countdown is 0):
        set countdown = TASKA_TICKS
        mainBodyOfTaskA()
```