UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 6 Examination in Engineering: May 2026

Module Number: EC6303 **Module Name: Embedded System Design(C-23)**

[Three Hours]

[Answer all questions]

Q1 a) Compare the differences in between 16F84A and 16F877A microcontrollers [2 Marks]

b) Briefly explain in few words the purpose of the address 0x0004 in the program memory of the PIC 16 series microcontrollers. [2 Marks]

c) What is meant by the statement; PIC 16 series microcontrollers are 8-bit microcontrollers. [2 Marks]

d) State the difference between a microprocessor and a microcontroller. [2 Marks]

e) Describe the advantages of in-circuit programming and debugging over the corresponding conventional development process. [2 Marks]

f) State the advantages of flash ROM, compared with other memory types [1 Mark]

g) State the function of the EQU directive. [1 Mark]

Q2 A matrix keypad consists of matrix switches (one switch for each key) as shown in Figure Q2 (a). In order to scan the keypad, each column is pulled to logic one in turn whilst each raw is read in turn. Figure Q2 (b) demonstrates the flow chart of the scanning process. Once the key is identified it should be displayed on a 7 segment LED which is connected to a Microcontroller. The task of the program is to produce software which will scan the keypad and determine which key is pressed. Refer the table Q2 for coding.

a) Draw a complete flow chart to demonstrate the overall operation of the keypad. [3 Marks]

b) Write an algorithm to demonstrate the scanning and identification process. [3 Marks]

c) Draw a complete circuit diagram with Microcontroller. [4 Marks]

d) Write a program in assembly language to execute the given task. [5 Marks]

Q3 You are developing firmware for a smart temperature-controlled ventilation system using the ATmega32. An LM35 temperature sensor with a sensitivity of 10 mV/°C connected to ADC0 (PA0) is used to monitor the room temperature. The system uses the 10-bit Successive Approximation ADC of the ATmega32 with a reference voltage of 5 V.

The system periodically reads the sensor value and controls the speed of a DC fan connected to the OC0 pin using PWM. A push button connected to INT0 enables or disables the cooling system. Timer0 is configured for PWM generation, while ADC conversion is performed inside the main program loop. You are given a partially completed Embedded C program in Figure Q3.1.

a) i) Define polling in embedded systems and briefly state one major disadvantage of polling. Identify where polling is used in the given code and specify the corresponding line. [1 Mark]

ii) Explain the sequence of operations carried out by the microcontroller when the INT0 external interrupt is triggered. [1Mark]

iii) Complete **Line 10** of the given Embedded C program in Figure Q3.1. to toggle the variable *system_on* whenever the external interrupt occurs. [1Mark]

iv) With reference to the INT0 configuration statements given in Figure Q3.1, identify the external interrupt triggering mode currently configured for the INT0 interrupt. [1Mark]

b) i) Determine the approximate 10bit digital output value corresponding to a room temperature of $45°C$. Clearly show all calculation steps. [1 Mark]

ii) Based on the ADC configuration used in Figure Q3.1, determine whether the ADC result is left-adjusted or right-adjusted. Justify your answer briefly. [1 Mark]

iii) Although the datasheet of the LM35 temperature sensor specifies a temperature range of $\mathbf{-55°C}$ **to** $\mathbf{150°C}$, the given ATmega32-based ADC system can directly measure only the range from $\mathbf{0°C}$**to** $\mathbf{150°C}$. Briefly explain the reason for this limitation and state one method that can be used to measure negative temperatures. [2 Marks]

c) i) Compare Fast PWM mode and Phase Correct PWM mode with respect to waveform generation and practical applications. [1Mark]

ii) Using the given Timer0 configuration details and register configuration tables, complete **Line 39** of the Embedded C program given in Figure Q3.1. to configure Timer0 in:

- Fast PWM mode
- Non-inverting mode
- Prescaler = 64. [1 Mark]

Q4. Modern embedded systems are increasingly integrated into smart and connected technologies such as autonomous vehicles, wearable healthcare devices, industrial automation systems, smart agriculture, and IoT-based monitoring applications. These systems are expected to provide reliable communication, real-time responsiveness, low-power operation, and dependable performance while operating under limited hardware and energy resources.

a) i) Differentiate between bit rate and baud rate. [1Mark]

ii) Briefly explain why RS232 signals cannot be directly interfaced with TTL-level microcontrollers. . [1Mark]

ii) Sketch the complete frame formation in the serial line when the ASCII character 'x' (0x78) is transmitted using 8 data bits, even parity, and 1 stop bit. Clearly label each bit. . [2 Marks]

iii) Figure Q4(a) shows an AVR USART initialization function. Calculate the configured baud rate. [2 Marks]

b) Explain the difference between hard real-time systems and soft real-time systems by giving one suitable embedded-system example for each. [1 Mark]

c) Briefly explain Dynamic Voltage Scaling (DVS) and how it helps reduce power consumption in CMOS-based embedded systems. [2 Marks]

d) State two reasons why re-configurability is important in modern Networked Embedded Systems. [1 Mark]

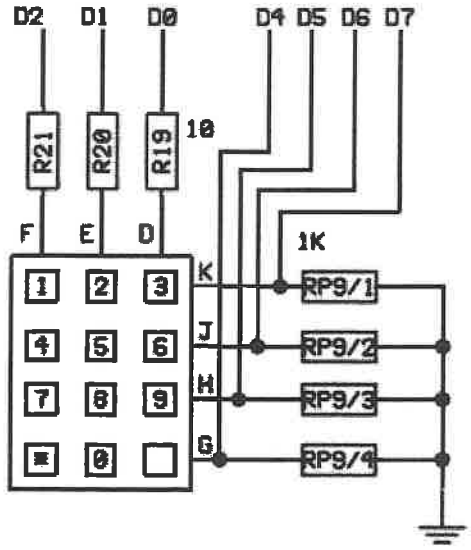


Figure Q2.(a)

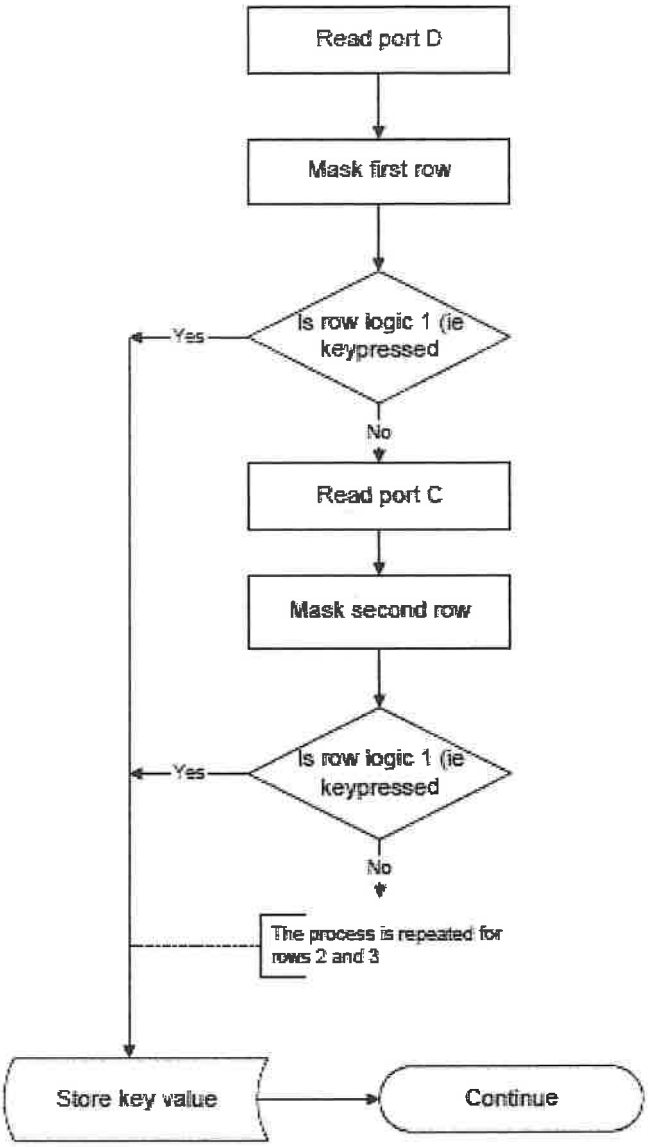


Figure Q2.(b)

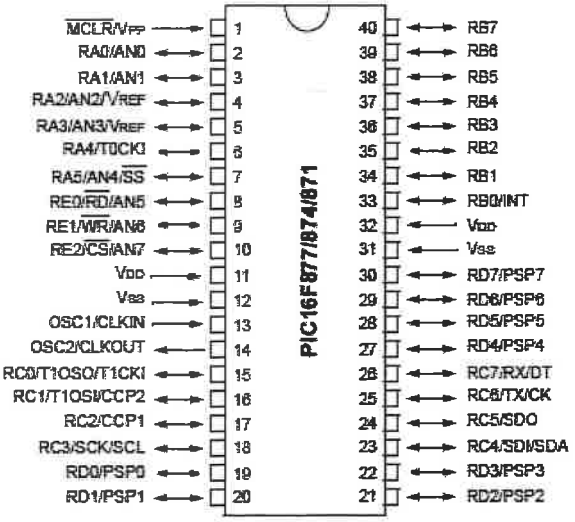


Figure Q2 (c): Pin diagram of 16F8XX.

Table Q2: Instruction set

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

```
1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3
4  volatile uint8_t system_on = 0;
5
6  // External Interrupt Service Routine
7  ISR(INT0_vect)
8  {
9      // Line 10: Toggle cooling system state
10     ------------------------------;
11 }
12
13 // ADC Initialization
14 void ADC_Init(void)
15 {
16     ADMUX = (1 << REFS0); // AVcc as reference, ADC0 selected
17
18     ADCSRA = (1 << ADEN) |
19              (1 << ADPS2) |
20              (1 << ADPS1); // ADC Prescaler
21 }
22
23 // ADC Read
24 uint16_t ADC_Read(void)
25 {
26     ADCSRA |= (1 << ADSC); // Start conversion
27
28     while (ADCSRA & (1 << ADSC)); // Wait for conversion
29
30     return ADC;
31 }
32
33 // PWM Initialization
34 void PWM_Init(void)
35 {
36     DDRB |= (1 << PB3); // OC0 as output
37
38     // Line 39:
39     // Configure Timer0 for:
40     // Fast PWM mode
41     // Non-inverting mode
42     // Prescaler = 64
43     TCCR0 = ----------------------------------------;
44 }
```

```
int main(void)
{
    uint16_t adc_value;

    DDRD &= ~(1 << PD2); // INT0 as input

    MCUCR |= (1 << ISC01); // Falling edge trigger
    MCUCR &= ~(1 << ISC00);

    GICR |= (1 << INT0); // Enable INT0

    sei(); // Enable global interrupts

    ADC_Init();
    PWM_Init();

    while (1)
    {
        adc_value = ADC_Read();

        if (system_on)
        {
            if (adc_value < 300)
            {
                OCR0 = 60;
            }
            else if (adc_value < 600)
            {
                OCR0 = 150;
            }
            else
            {
                OCR0 = 240;
            }
        }
        else
        {
            OCR0 = 0;
        }
    }
}
```

Figure Q3.1: Partially Completed Embedded C Program

Table Q3.1: MCUCR -- MCU Control Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

Table Q3.2: INT0 Interrupt Sense Control

ISC01	ISC00	INT0 Trigger Condition
0	0	Low level
0	1	Any logical change
1	0	Falling edge
1	1	Rising edge

Table Q3.3: GICR – General Interrupt Control Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INT1	INT0	INT2	–	–	–	IVSEL	IVCE

Table Q3.4: ADMUX – ADC Multiplexer Selection Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Table Q3.5: GICR – ADC Result Adjustment Selection

ADLAR	ADC Result Format
0	Right-adjusted Result
1	Left-adjusted Result

Table Q3.6: TCCR0 – Timer/Counter0 Control Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Table Q3.7: Timer0 Waveform Generation Mode

COM01	COM00	OC0 Operation
0	0	Normal port operation
1	0	Non-inverting mode
1	1	Inverting mode

Table Q3.8: Compare Output Mode for Fast PWM

WGM01	WGM00	Timer0 Mode
0	0	Normal
0	1	PWM, Phase Correct
1	0	CTC
1	1	Fast PWM

Table Q3.9: Timer0 Clock Select

CS02	CS01	CS00	Clock Source
0	0	0	No clock source
0	0	1	clk/1
0	1	0	clk/8
0	1	1	clk/64
1	0	0	clk/256
1	0	1	clk/1024
1	1	0	External clock on T0, falling edge
1	1	1	External clock on T0, rising edge

```
#define F_CPU 8000000UL

#include <avr/io.h>

void USART_Init(void)
{
    /* Baud Rate Configuration */

    UBRRH = 0x01;
    UBRRL = 0x9F;

    /* Enable Receiver and Transmitter */

    UCSRB = (1<<RXEN) | (1<<TXEN);

    /* USART Frame Format:
       8 data bits, 1 stop bit
    */

    UCSRC = (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);
}
```

Figure Q4.a: AVR USART initialization function