13

9

Scoring Indicators

COURSE NAME : EMBEDDED SYSTEM AND REAL TIME OPERATING SYSTEM

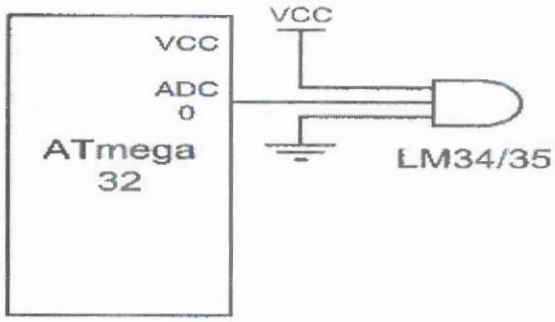
COURSE CODE : 5131 **QID : 2109230283A**

Q No	Scoring Indicators	Split score	Sub Total	Total score
	PART A			9
I. 1	32	1	1	
I. 2	Small scale, medium scale, large/complex embedded systems	1	1	
I. 3	Unsigned char ,char, unsigned int, int, unsigned long,long,float,double (any two)	1	1	
I. 4	Timer 1	1	1	
I. 5	DDRB = 0x00 or DDRB =0	1	1	
I. 6	MAX232	1	1	
I. 7	RS232	1	1	
I. 8	A process is a program in execution.	1	1	
I. 9	FCFS,SJF,Round Robin,Priority,LCFS (any Two).	1	1	
	PART B			24
II. 1	Consumer Electronics, House hold appliances, Home Animation and security systems, Automotive Industry, Telecom, Computer peripherals etc. (any three applications)	1x3	3	
II. 2	PORTx is associated with 3 registers: PORTx.PINx and DDRx where x can be A,B C or D.	1x3	3	
II. 3	Program ROM Space. • Program ROM is a non-volatile flash memory which stores the program/code • As the CPU fetches the opcode from the program ROM, the program counter is incremented to point to the next instruction. • ATMega32 has 32K x 8 Bytes of program ROM. But it is arranged into 16K x 2 Bytes of memory. • Advantages of this arrangement is we can fetch 2 bytes of data from the ROM in a single clock cycle.. •This architecture will speed up the entire program execution		3	

II. 4	#include <avr/io.h>
	int main(void)
	{
	unsigned char temp;
	DDRB =0x00; //Port B is input
	DDRC = 0xFF; //Port C as output
			3	

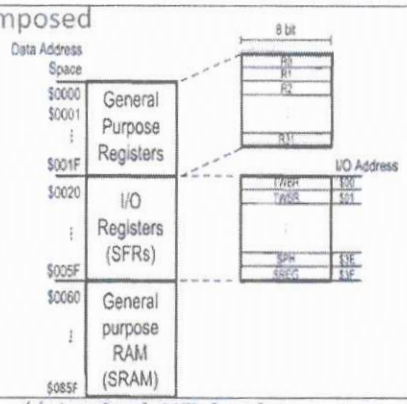
	while(1)
	{
	temp = PINB;
	PORTC=temp;
	}
	return 0;
	}			

II. 5	There are 3 ways to create a time delay in AVR C. 1. Using a simple for loop 2. Using predefined C functions 3. Using AVR timers	1x3	3	
II. 6	PORTB = 0x05 PORTC = 0x6C PORTD = 0x2C	1x3	3	
II. 7	The ADC peripheral of the ATmega32 has the following characteristics: **(Any 3 features)** (a) It is a 10-bitADC. (b) It has 8 analog input channels, 7 differential input channels, and 2 differential input channels with optional gain of I0x and 200x. (c) The converted output binary data is held by two special function registers called ADCL (A/D Result Low) and ADCH (A/D Result High). (d) Because the ADCH:ADCL registers give us 16 bits and the ADC data out is only 10 bits wide, 6 bits of the 16 are unused. We have the option of making either the upper 6 bits or the lower 6 bits unused. (e) We have three options for Vref- Vref can be connected to AVCC (Analog V cc), internal 2.56 V reference, or external AREF pin. (f) The conversion time is dictated by the crystal frequency connected to the XTAL pins and ADPS0:2 bits.	1x3	3	
II. 8	**RS (Register Select)** There are two very important registers inside the LCD. 1. **command code register,** 2. **data register** □ The RS pin is used for their selection as follows. □ If **RS=0**, the instruction **command code register is selected,** allowing the user to send commands such as clear display, cursor at home, and so on. □ If **RS= 1** the **data register** is selected, allowing the user to send data to be displayed on the LCD. **R/W (Read/Write)** □ R/W input allows the user to write information to the LCD or read information from it. □ **R/W=1 when reading; R/W=0 when writing. E (Enable)**	1x3	3	

	☐ The enable pin is used by the LCD **to latch information presented** to its data pins.			
II.9			3	

II.10	Real-Time Operating Systems Systems that strictly adhere to the timing constraints for a task is referred as Real-Tune' systems. 1 .A **Hard Real-Time system** must meet the deadlines for a task without any slippage. Missing any deadline may produce catastrophic results for Hard Real-Time Systems, including permanent lose and irrecoverable damages to the system/users. Hard Real-Time systems emphasize the principle 'A late answer is a wrong answer'. 2.**Soft Real Time Systems** Real-Time Operating System that does not guarantee meeting deadlines, but offer the best effort to meet the deadline are referred as 'Soft Real-Time' systems. Missing deadlines for tasks are acceptable for a Soft Real-time system	Listing-1 1 1	3	
	PART C			42
III. 1		1x7	7	7

	General purpose computer	Microcontroller
Purpose	Multipurpose	Single function
Constraint	Low or no resource constraint	Size, power, cost, memory, real time
Performance	Faster and better	Fixed runtime requirement
User Interface	Can have keyboard, display, mouse, touch screen	Integrated into the real world with buttons, sensors, Leads, LCDs, Bluetooth system
Access Time	Higher accessing time required	Low accessing time
Size	Systems become bulkier and expensive	Make the system simple, economic and compact
Example	INTEL 8086,	INTEL 8051

IV.	The data memory is composed of three parts: • GPRs (general purpose registers), • Special Function Registers (SFRs), and • Internal data SRAM. 	Listing of parts-1 Figure-3 Explanation-3	7	7

V.	`#include <avr/io.h>` //standard AVR header
	`int main(void){` //the code starts from here
	`unsigned char myList[] = "012345ABCD";`
	`unsigned char z;`
	`DDRB = 0xFF;` //PORTB is output
	`for(z=0; z<10; z++)` //repeat 10 times and increment z
	`PORTB = myList[z] ;` //send the character to PORTB
	`while(1);` //needed if running on a trainer
	`return 0;`
	`}`
		7	7	7

VI.	**Steps in executing an interrupt** 1. It finishes the instruction it is currently executing a n d saves the address of the next instruction (program counter) on the stack. 2. It jumps t o a fixed location in memory called the *interrupt vector table.* The Interrupt vector table directs the microcontroller to the address of the interrupt service routine (ISR). 3. The microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI(return from interrupt). 4. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted .First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.	7	7	7

VII.

```
#include <avr/io.h>          // standard AVR header
int main(void){
   unsigned char  x, y;
   unsigned  char mybyte = 0x29;
   DDRB = DDRC = 0xFF;     // make  Ports  B  and C  output
   x = mybyte & 0x0F;      // mask upper   4  bits
   PORTB  =  x | 0x30;     // make  it ASCII
   y = mybyte & 0xF0;      // mask  lower  4  bits
   y =  y >> 4;            // shift   it   to   lower   4   bits
   PORTC  =  y | 0x30;     // make  it ASCII
   while(1);               // stay here
   return  0;
}
```

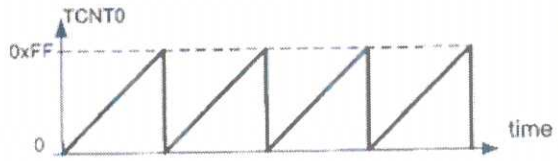
7 | 7 | 7

VIII.

Modes of Operation in Timer0

1. Normal mode of operation

- In this mode, the content of the timer/counter increments with each clock.
- It counts up until it reaches its max of 0xFF.
- When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0(Timer Over flow).



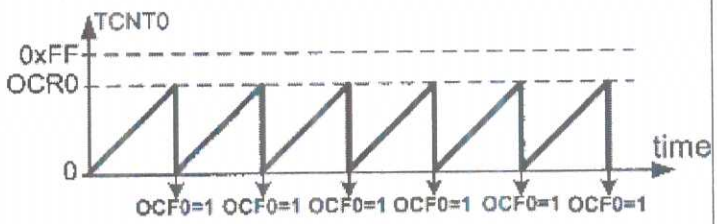
3.5

2. **compare match (CTC) mode programming**

- The OCR0 register is used with CTC mode.
- In the CTC mode, the timer is incremented with a clock.
- But it counts up until the content of the TCNT0 register becomes equal to the content of OCR0 (compare match occurs); then, the timer will be cleared and the OCF0 flag will be set(OCT0=1) when the next clock occurs.
- The OCF0 flag is located in the TIFR register.

3.5

Timer/counter0 CTC mode :



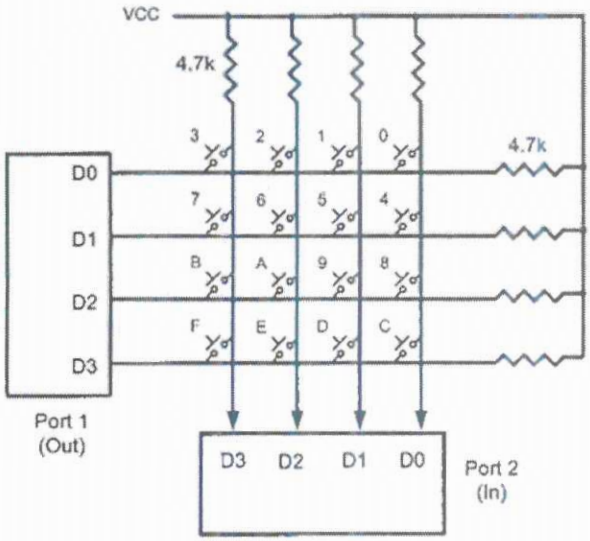
7 | 7

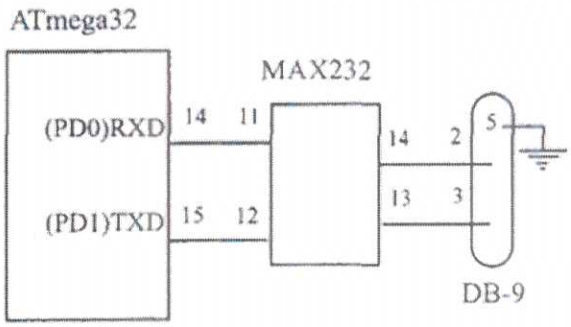
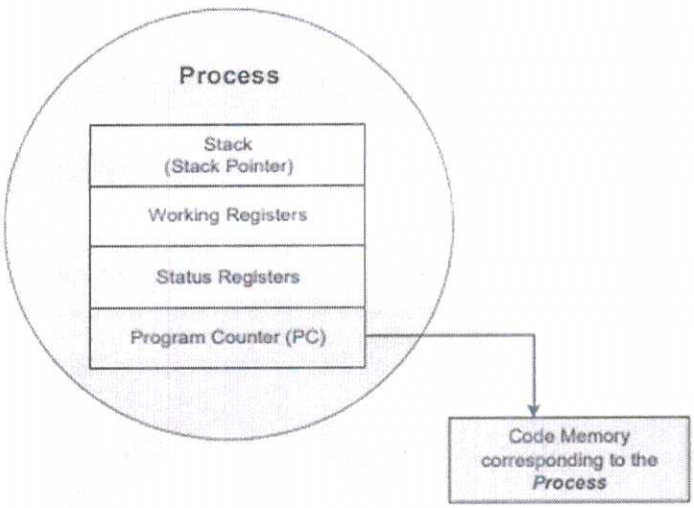
IX

Interfacing keyboard with AtMega32

- Keyboards are organized in a matrix of rows and columns.
- The CPU accesses both rows and columns through ports.
- Therefore, with two 8-bit ports, an 8x8 matrix of keys can be connected to a microcontroller.

7 | 7

	• When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns. Figure shows the matrix keyboard connection to ports:  • The rows are connected to an output port and the columns are connected to an input port. • Columns and rows are all connected to high(VCC). • It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed. How this is done is explained next. 1. To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch , and then it reads the columns. 2. If the data read from the columns is D3-D0= 1111,no key has been pressed and the process continues until a key press is detected. If one of the column bits has a zero, this means that a key press has occurred. 3. After a key press is detected, the microcontroller will go through the process of identifying t h e key. 4. Starting with the top row, the microcontroller grounds it by providing a low to row D0 only; then it reads the columns. 5. If the data read is all 1s,no key in that row is activated and the process is moved to the next row. 6. It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified. 7. After identification of the row in which the key has been pressed ,the next task is to find out which column the pressed key belongs to. 8. After identifying the row and column, the microcontroller will identify which key has been pressed.	FIG:3 Explanation:4		
X,	ATMEGA32 CONNECTION TO RS232 RX and TX pins in the ATmega32 • The ATmega32 has two pins that are used specifically for transferring and receiving data serially.	Explanation : 4	7	7

	•These two pins are called TX and RX and are part of the PortD group (PD0 and PD1) of the 40-pinpackage. •Pin15 of the ATmega32 is assigned to TX and pin 14 is designated as RX. •These pins are TTL compatible; therefore , they require a line driver to make them RS232 compatible. •One such line driver is the MAX232chip. MAX232 • Because the RS232 is not compatible with today's microprocessors and microcontrollers, we need a line driver (voltage converter) to convert the RS232's signals to TTL voltage levels that will be accept able to the AVR's TX and RX pins. 	Diagram:3		
XI	**The Structure of a Process** A process mimics a processor in properties and holds a set of registers, process status, a Program Counter (PC) to point to the next executable instruction of the process, a stack for holding the local variables associated with the process and the code corresponding to the process. This can be visualized as shown below:  The 'Stack' memory holds all temporary data such as variables local to the process. Data memory holds all global data for the process. The code memory contains the program code (instructions) corresponding to the process.	Explanation – 4 fig: 3	7	7

XII.	**First-Come-First-Served (FCFS)/FIFO Scheduling** • As the name indicates, the First- Come-First-Served (FCFS) scheduling algorithm allocates CPU time to the processes based on the order in which they enter the 'Ready' queue. • The first entered process is serviced first. •FCFS scheduling is also known as First In First Out (FIFO) where the process which is put first into the 'Ready' queue is serviced first. **Priority based scheduling** Priority based non-preemptive scheduling algorithm ensures that a process with high priority is serviced at the earliest compared to other low priority processes in the 'Ready' queue. The priority of a task/process can be indicated through various mechanisms. The Shortest Job First (SJF) algorithm can be viewed as a priority based scheduling where each task is prioritised in the order of the time required to complete the task. The lower the time required for completing a process the higher is its priority in SJF algorithm. Another way of priority assigning is associating a priority to the task/process at the time of creation of the task/process. The priority is a number ranging from 0 to the maximum priority supported by the OS. The maximum level of priority is OS dependent.	Explanation-2 Eg:1.5 Explanation-2 Eg:-1.5	7	7
XIII.	**Deadlock** In its simplest form 'deadlock' is the condition in which a process is waiting for a resource held by another process which is waiting for a resource held by the first process To elaborate: Process A holds a resource x and it wants a resource *y* held by Process B. Process B is currently holding resource *y* and it wants the resource *x* which is currently held by Process A. Both hold the respective resources and they compete each other to get the resource held by the respective processes. The result of the competition is 'deadlock'. **Necessary conditions to hold a deadlock** **Mutual Exclusion:** The criteria that only one process can hold a resource at a time. **Hold and Wait:** The condition in which a process holds a shared resource and waiting for additional resources held by other processes. **No Resource Preemption:** The criteria that operating system cannot take back a resource from a process which is currently	2 Listing-1 Explanation-4x1	7	7

	holding it and the resource can only be released voluntarily by the process holding it. **Circular Wait:** A process is waiting for a resource which is currently held by another process which in turn is waiting for a resource held by the first process. In general, there exists a set of waiting process P0, P1 ... Pn with PO is waiting for a resource held by P1 and P1 is waiting for a resource held by PO, Pn is waiting for a resource held by PO and PO is waiting for a resource held by Pn and so on... This forms a circular wait queue.			
XIV.	**Functional Requirements** • Processor support • Memory requirement. • Real-Time capabilities. • Kernel and Interrupt Latency. • Process communication and task synchronization. • Modularization Support. • Support for Networking and communication. • Development Language Support.	7x1=7	7	7