1. What do you understand about an ARM Processor and list out the different features of ARM instruction which makes it suitable for embedded applications?  
     
   Ans: ARM processors are ideal for embedded applications due to their low power consumption, high performance, small size, scalability, low cost, and easy programmability. They are highly customizable and come with integrated peripherals such as timers and serial ports. Overall, the combination of these features makes ARM processors a popular choice for a wide range of embedded applications.
2. What is the ARM Cortex M4 Programming Model? Find out the instructions used. How do you select a specific Cortex-M processor?  
     
   Ans: The ARM Cortex-M4 programming model includes a 32-bit core with a Memory Protection Unit, DSP instructions, and an NVIC. Its instruction set includes general-purpose, memory access, bit manipulation, and DSP instructions. To select a specific Cortex-M processor, consider requirements such as performance, power consumption, available peripherals, cost, and development tools. ARM offers a range of Cortex-M processors with specific features and capabilities. Consult product documentation and development tools to choose the right processor.
3. Study the data sheet of any sensor of your choice and create a write-up explaining the specifications and working of that sensor.  
     
   Ans: LDR or Photoresistor is a resistor that changes resistance based on light. It has no polarity and is breadboard-friendly, making it easy to use. LDRs can sense light and be used with microcontrollers and potential divider circuits. They are a cheap and modest solution for light sensing and can be used in applications such as automatic street lights, day/night detection, and obstacle detection.
4. What are Electric Actuators and how do sensors and actuators are used to interface with microcontrollers in embedded systems?  
     
   Ans: Electric actuators are devices that convert electrical energy into mechanical motion, used for controlling systems in various applications. Sensors are used to detect and measure physical quantities such as temperature, pressure, and position, while actuators are used to control physical processes such as movement or valve opening/closing. In embedded systems, sensors and actuators are interfaced with microcontrollers to collect data and control processes. Microcontrollers process the data from sensors and provide control signals to actuators to perform the desired actions in the system.
5. Measure the temperature of two rooms. To measure the temperature, two different nodes are there, and those two nodes are sending the temperature data to a master node. Implement the system using the I2C protocol. Each slave node consists of one Arduino board and one temperature sensor (TMP36)  
     
   Ans: To implement the I2C protocol for measuring the temperature of two rooms, we can follow these steps: First, connect the TMP36 temperature sensor to each Arduino board and load the I2C slave library. Then, write code on each board to read the temperature data from the TMP36 sensor and send it to the master node using the I2C protocol. Next, connect the two Arduino boards to the master node and load the I2C master library. Finally, write code to request temperature data from each slave node and display it on the master node.
6. Interfacing a 16\*2 LCD with Arduino using I2C protocol  
     
   Ans: To interface a 16\*2 LCD with Arduino using I2C protocol, we need to connect the I2C module to the LCD and the Arduino. We then need to download and include the LiquidCrystal\_I2C library in our Arduino code. We can then use the library's functions to control the LCD display. The library handles the I2C communication and makes it easy to display text and control the cursor on the LCD. We can customize the display with different characters, fonts, and backlight settings. By using the I2C protocol, we can save the number of pins used and simplify the wiring process.
7. I/O circuit diagram of ATmega AVR Microcontroller  
     
   Ans: The I/O circuit diagram of ATmega AVR Microcontroller consists of multiple ports that can be configured as either inputs or outputs. Each port has 8 pins that can be individually programmed to either receive or transmit data. The Microcontroller also has an external interrupt pin that can be used to trigger an interrupt service routine. The I/O circuit diagram also includes a set of registers that can be used to configure the ports and control the input/output operations. These registers include the Data Direction Register (DDR), Port Input Register (PIN), and Port Output Register (PORT). The DDR register is used to set the direction of each pin in a port, while the PIN and PORT registers are used to read and write data to the ports, respectively. Overall, the I/O circuit diagram of ATmega AVR Microcontroller provides flexibility and control in interfacing with various external devices.
8. Write a C program to receive a character from the serial port of ATmega32. If it is “a” to “z”, change it to capital letters and send it back.  
     
   Ans:

#include <avr/io.h>  
#include <util/delay.h>

#define BAUDRATE 9600  
#define BAUD\_PRESCALER ((F\_CPU/(BAUDRATE\*16UL))-1)

void USART\_init(void) {  
UBRRH = (uint8\_t)(BAUD\_PRESCALER>>8);  
UBRRL = (uint8\_t)(BAUD\_PRESCALER);  
UCSRB |= (1<<RXEN) | (1<<TXEN);  
UCSRC |= (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);  
}

unsigned char USART\_receive(void) {  
while(!(UCSRA & (1<<RXC)));  
return UDR;  
}

void USART\_send(unsigned char data) {  
while(!(UCSRA & (1<<UDRE)));  
UDR = data;  
}

int main(void) {  
USART\_init();

while(1) {  
unsigned char receivedChar = USART\_receive();  
if(receivedChar >= 'a' && receivedChar <= 'z') {  
receivedChar -= 32;  
}  
USART\_send(receivedChar);  
}

return 0;  
}

1. Write a C program to transfer the letter 'S' serially at 9600 baud rates repeatedly. Use 8-bit data and 1 stop bit. Find out the memory usage of the code  
     
   Ans:

#include <avr/io.h>  
#include <util/delay.h>

#define F\_CPU 8000000UL  
#define BAUD 9600  
#define UBRR\_VALUE (F\_CPU/16/BAUD-1)

void USART\_Init() {  
/\* Set baud rate \*/  
UBRRH = (unsigned char)(UBRR\_VALUE>>8);  
UBRRL = (unsigned char)UBRR\_VALUE;  
/\* Enable receiver and transmitter \*/  
UCSRB = (1<<RXEN)|(1<<TXEN);  
/\* Set frame format: 8data, 1stop bit \*/  
UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);  
}

void USART\_Transmit(unsigned char data) {  
/\* Wait for empty transmit buffer \*/  
while (!(UCSRA & (1<<UDRE)));  
/\* Put data into buffer, sends the data \*/  
UDR = data;  
}

int main(void) {  
USART\_Init();  
while (1) {  
USART\_Transmit('S');  
\_delay\_ms(100);  
}  
return 0;  
}

Output for memory Usage:   
text   data    bss    dec     hex      filename  
722    0         4       726    2d6       a.out