0046

The Open University of Sri Lanka
Faculty of Engineering Technology
Department of Electrical and Computer Engineering

Study Programme	:	Bachelor of Science Honours in Engineering
Name of the Examination	:	Final Examination
Course Code and Title	:	**EEX5346 Embedded system**
Academic Year	:	2024/25
Date	:	21st February 2026
Time	:	0930-1230h
Duration	:	3 hours

General instructions

1. Read all instructions carefully before answering the questions.
2. This question paper consists of **Four (4)** questions in **Six (6)** pages.
3. Answer **all** questions.
4. Refer to the Annexure of the ESP32 power modes given on page Four (4) to calculate power consumption, if required.
5. Answer for each question should commence on a new page.
6. Refer to the Annexure of the Arduino programming syntax given on page Five(5) to write any Arduino programming code.
7. This is a Closed Book Test (CBT).
8. Answers should be in clear handwriting.
9. Do not use Red colour pen.

The following section is taken from the article "*Urban transportation system development project for Colombo metropolitan region and suburbs*". Your task is to analyse the following specifications and design the Smart Traffic Control System (**STCS**) by providing IoT solutions for a smart city, by answering the questions.

This article was published on the "*Ministry of Transport, Highway, Port and Civil Aviation Transport Division*" website.

https://www.transport.gov.lk/web/images/downloads/F-CoMTrans_Main_S.pdf

"At many intersections in the Colombo Municipal Council(CMC), traffic demand exceeds the capacity of the intersection in peak hours. The majority of traffic signals at these intersections are standalone; therefore, the signals do not coordinate with each other. The existing signal applies a pattern control unit for the day of week and for time of

day; therefore, the existing signal system does not provide efficient traffic control."

To address these challenges, the report recommends the implementation of traffic-actuated signal systems equipped with vehicle detectors. These systems would dynamically adjust signal timings based on real-time traffic conditions to enhance throughput. Proposed features include adaptive timing based on approaching vehicle queues, predictive green light control, pedestrian request integration, emergency vehicle prioritisation, night mode operation, and manual override capabilities.

Question 01

a. Draw a diagram and briefly explain the IoT architecture of the proposed **STCS** solution. Clearly indicate what the edge devices, gateways, connectivities, cloud services, and applications are. (13 marks)

b Traffic light controllers are grouped into wireless cluster networks based on traffic flow density. Within each cluster, the traffic signals communicate with one another to coordinate their control operations. Draw and briefly explain the proposed connectivity architecture for the **STCS** and justify your design. (Hint: Consider the short-range long range wireless communication technologies and their range, power, bandwidth, and throughput, etc.) (8 marks)

c. Draw block diagrams of the hardware architecture of the traffic light controller, cluster sync controller, and the sensor nodes and show how to integrate each input and output of each module/unit in the proposed system. (Hint: Consider the interfacing techniques in master/slave devices, arbitration, transactions, priorities, etc.) (8 marks)

d. Briefly describe the Watchdog timer roles related to the proposed **STCS** features. (7 marks)

Question 02

c. Briefly describe the three (03) main challenges you would have to face when applying the IoT concept to the **STCS** solution. (8 marks)

d. List three (3) cloud tools available for data visualisation for **STCS** IoT applications. (6 marks)

Question 03

a. Draw a Software architecture for the traffic light controller in **STCS** and briefly explain each operation. (10 marks)
(Hint: Show how Interrupt Service Requests (**ISR**) and tasks interact)

b. Draw an event-driven state machine diagram for a 3-way traffic light controller in **STCS** tasks and describe how they interact with the states of an **RTOS**. (10 marks)

c. Refer to the Arduino programming instructions and write a program for an event-driven state machine in Q3.(b). State the comments where necessary. (10 marks)

Question 04

a. Considering power optimisation, compute the energy consumption for the **traffic light controller in STCS** over one month. Assume that the microcontroller module undergoes an immediate transition from sleep to wake, transmitting data within 650ms. Each sensor requires 25ms for a single measurement and consumes 15mA during typical operation. Sensing and transmission are done **once every 1 second** using an ESP32 single-core module. (8 marks)
(Hint: E= sleep energy x sleep time + active energy x active time)

b. Draw a diagram and briefly describe how you performed the power-performance tradeoff evaluation in the traffic controller for the proposed **STCS** (Hint: The processor runs at three different voltages, 2.5V, 4.0V, and 5.0V. In addition, the maximum clock frequencies are 25MHz, 40MHz, and 50MHz for each voltage, respectively. Assume energy consumption is 40nJ per cycle at 5.0V, and tasks need to execute 10^9 cycles within 25s. Power consumption is zero when the processor is shut down.) (12 marks)

All Rights Reserved

Annexure

ESP32 Power Modes Figures Referenced from the ESP32 Series Datasheet

Table 8: Power Consumption by Power Modes

Power mode	Description			Power Consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 17 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered on.	240 MHz*	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz*	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP coprocessor is powered on.			150 μA
	ULP sensor-monitored pattern			100 μA @1% duty
	RTC timer + RTC memory			10 μA
Hibernation	RTC timer only			5 μA
Power off	CHIP_PU is set to low level, the chip is powered off.			1 μA

Table 9: Association of sleep pattern

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Table 17: RF Power-Consumption Specifications

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

Syntax of selected instructions of the Arduino programming

Structure & Flow

Basic program structure

```
void setup () {
    // Runs once when sketch starts
}
void loop () {
   //Runs repeatedly
}
```

Control Structures

```
If (x < 5) {……} else {……}
while (x < 5) {……}
for ( int i = 0; i< 10; i++) {……}
break;       // Exit a loop immediately
continue;    // Go to next iteration
switch (var) {
case 1:
  ……
break;
   case 2:
 ……
break;
   default:
 ……
}
return x;    // x must match return type
return;      // For void return type
```

Function Definitions

<ret. type><name> (params) {……}
e.g. int double (int x) {return x*2;}

Operators

General operators

= assignment
+ add - subtract
* multiply / divide
% modulo
== equal to != not equal to
<less than > greater than
<= less than or equal to
>= greater than or equal to
&& and || or
! not

Compound operators

++ increment
-- decrement
+= compound addition
-= compound subtraction
*= compound multiplication
/= compound division
&= compound bitwise and
|= compound bitwise or

Bitwise Operators

& bitwise and | bitwise or
^ bitwise xor ~ bitwise not
<< shift left >> shift right

Pointer Access

& reference: get a pointer
* dereference: follow a pointer

Variables, Arrays, and Data

Data types

bool	true	|	false
char	-128	-	127, 'a' '$' etc.
unsigned char	0	–	255
byte	0	–	255
int	-32768	-	32767
unsigned int	0	–	65535
word	0	–	65535
long	-2147483648	–	2147483647
unsigned long	0	–	4294967295
float	-3.4028e+38	–	3.4028e+38

double currently same as float
void return type: no return value

Strings

```
char str1[8] =
    {'A', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
    // Includes \0 null termination
char str2[8] =
    {'A', 'r', 'd', 'u', 'i', 'n', 'o', };
    //Compiler adds null termination
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numeric Constants

123 decimal
0b01111011 binary
0173 octal – base 8
0x7B hexadecimal – base 16
123U force unsigned
123L force long
123UL force unsigned long
123.0force floating point
1.23e61.23*10^6 = 1230000

Qualifiers

Static persists between calls
volatilein RAM (nice for ISR)
const read-only
PROGMEMin flash

Arrays

```
byte mypins [] = {2, 4, 8, 3, 6};
int myInts [6];        // Array of 6 ints
myInts[0] = 42;        // Assigning first
                       // Index of myInts
myInts[6] = 12;        // ERROR! Indexes
                       // are 0 though 5
```

Built – in Functions

Pin Input / Output
Digital I/O – pins 0 – 13 A0 – A5
PinMode(pin,
{ INPUT | OUTPUT | INOUT_PULLUP})
int **digitalRead**(pin)
digitalWrite(pin, {HIGH | LOW})

Analog In – pins A0 – A5
Int **analogRead** (pin)
analogReference(
{DEFAULT | INTERNAL | EXTERNAL})

PWM Out – pins 3 5 6 9 10 11
analogWrite(pin, value) // 0-255

Advanced I/O
tone(pint, freq_Hz, [duration_msec])
noTone(pin)
shiftOut(dataPin, clockPin,
{MSBFIRST | LSBFIRST}, value)
shiftIn(dataPin, clockPin,
{MSBFIRST | LSBFIRST})
unsigned long pulseIn(pin,
{HIGH | LOW}, [timeout_usec])

Time
unsigned long millis()
// overflows at 50 days
unsigned long micros()
// overflow at 70 minutes
delay(msec)
delayMicroseconds(usec)

Math
min(x, y)**max**(x, y) **abs**(x)
sin(rad) **cos**(rad) **tan**(rad)
sqrt(x) **pow**(base, exponent)
constain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)

Random Numbers
randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)

Bits and Bytes
lowByte(x) **highByte**(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn) // bitn: 0=LSB 7=MSB

Type Conversions
char(val)**byte**(val)
int(val) **word**(val)
long(val) **float**(val)

External Interrupts
attachInterrrupt(interrupt, func,
{LOW | CHANGE | RISING | FALLING})
detachInterrupt(interrupt)
interrupts()
noInterrupts()

Libraries

serial – comm.with pc or via RX/TX
begin(long speed) // up to 115200
end()
int **available()** // #bytes available
int **read()**// -1 if none available
int **peek()** // Read w/o removing
flush()
print(data)println(data)
write(byte) write(char*string)
write(byte*data, size)
serialEvent() // called if data rdy

softwareSerial.h – comm. on any pin
softwareSerial (rxpin, txpin)
begin(long speed) // up to 115200
listen() // only 1 can listen
isListening() // at aitme.
read, peek, print, println, write
// Equivalent to serial library

EEPROM.h – access non-voLatile memory
byte read(addr)
write(addr, byte)
EEPROM[index] // Access as array

Servo.h – control servo motors
attach(pin, [min_usec, max_usec])
write(angle)// 0 to 180
writeMicroseconds(us)
// 1000-2000; 1500 is midpoint
int read()// 0 to 180
bool attached()
detach()

wire.h – I C communication
begin() // Join a master
begin(addr) // Join a salve @ addr
requestFrom(address, count)
beginTransmission(addr) // step 1
send(byte) // step 2
send(char*string)
send(byte*data, size)
endTransmission() // step 3
int avaialable() // #bytes available
byte receiver() // get next byte
onReceive(handler)
onRequest(handler)