R1

1. Design an application that monitors the temperature (T) of the environment using a LM50 sensor (with a Vout=T[°C]*0.01[V/°C]+0.5V response function in the −40°C to +125°C range). The output pin of the LM50 will be connected to an analogue inputs an of the Arduino's ADC (10 bit ADC with 5V reference voltage). The T reading should be triggered every 10 seconds (do not use delay) and perform filtering over 5 consecutive readings. The monitored T should be displayed on a RGB LED (3xLEDs with a common anode and independent cathodes) by the combination of the Red and Blue LEDs (the Green should not be used). For the lower T limit (−40°C) the RGB LED should emit a pure Blue color, for the upper one (125°C) the RGB LED should emit a pure Red color while for any other T in the range the RGB LED should emit a color between the Red ↔ Blue spectrum. Display the numerical value of the T on the LCD shield. Also, drive a fan connected to a DC motor if the T exceeds 25°C with a rotation speed ~ T (increase the rotation speed with 25% for every 5°C increase of T over 25°C). **ToDo:** Explain the design, draw the schematic (highlight the interconnection between all the components implied), draw the flow-chart of the app. and write the corresponding procedure(s) in C. [**Hint:** to generate a color in the Red ↔ Blue spectrum the Red and Blue LEDs should be driven by PWM signals (use: analogWrite(led, DC) and the relation between the Duty Cycles of the LEDs should always be: DCRed + DCBlue = 255; use a 270Ω resistor in series with each LED to limit the current]. (**3 p**)

2. Design a simple application that displays the contents of a software counter (register) as binary number on 8 LEDs using an Atmel MCU. The contents of the counter is incremented or decremented automatically every T seconds depending on the status of a switch (Close = increment / Open = decrement) connected to the MCU. The value of T can be set in the [1s .. 5s .. 1s ...] interval with a 1s step by successively pushing a button. Explain the design, draw the schematic and write a short program (C). Do not use delay(). **(2p)**

3. Design an ***EPROM*** memory interface for an 8086μP (**having a 16 bit data bus**). Total size: **128 KB (128K x 1 byte = 128K x 8 bits)**, mapped in the **lower part** of the memory space. Use ***EPROM* chips: 8K x 2 bits**. The interface design should contain: the memory chips layout, the address decoder, the data, address and control lines interconnection. Explain the design. **(2p)**

4. Explain (using bus-cycle time-diagrams and words) the execution phases of the **MOV MEM_ADD, AX** instruction (memory write), where **MEM_ADD** is an **even** memory address for the **8086** in minimum mode. **(1p)**

5. Explain the principles of the Interrupt driven I/O transfer (Intel 8086). **(1p)**

R2

1. Design a simple backwards parking warning sensor using an Arduino board and a LV EZ0 sonar (sensor_resolution = 0.01V/inch; range between 6 – 254 inches). When measuring the current free distance (D) perform filtering of 10 consecutive readings. The measurement should be restricted to the D = 6 – 80 inch range. Within this range, output the distance using a 4 LEDs bar (VU-meter style) and generate an audio signal with frequency f ~ 1 / D. If the measured distance is over 80 inch, turn off the LEDs and stop any audio signal generation. Also, display the numerical value of D on the LCD shield. **ToDo:** Explain the design, draw the schematic (highlight the interconnection between all the components implied), draw the flow-chart of the app. and write the corresponding procedure(s) in C. [**Hint:** The 4 LED VU-meter should be composed by a Green, Yellow, Orange and Red LEDs (placed linearly in this order: GYOR). For a free distance D > 80 inch all the LEDs are turned off. For D = 61 – 80 inch, lit up the Green LED (G). For D = 41 – 60 inch, lit up the Green and Yellow LEDs (GY), … , for a distance D < 20 inch lit up all the LEDs (GYOR). To generate the audio signal use a speaker connected to one of the digital ports and generate a tone with a f = 100 Hz when D = 61 – 80 inch, … , f = 400Hz when the D < 20 inch (use function tone(pin, f) to start the tone generation for a specific range/frequency and noTone() to stop the tone generation. (**3p**)

2. Design a simple application that repeats indefinitely a (fade-in + fade-out) cycle over an LED. The [fade-in+fade out] time interval T can be incremented/decremented (in the 1s – 10s interval with a 1s step) by 2 push buttons (-/+) using an Atmel MCU. Explain the design, draw the schematic and write a short program (C). Do not use delay().**(2p)**

3. Design a ***SRAM*** memory interface for an 8086μP (**having a 16 bit data bus**). Total size: **256 KB (256K x 1 byte = 256K x 8 bits)**, mapped in the **upper part** of the memory space. Use **SRAM chips: 8K x 8 bits**. The interface design should contain: the memory chips layout, the address decoder, the data, address and control lines interconnection. Explain the design. (**2p**)

4. Explain (using bus-cycle time-diagrams and words) the execution phases of the **IN AX, PORT_ADD** instruction (I/O read), where **PORT_ADD** is an **odd** I/O address for the **8088** in minimum mode. **(1p)**

5. Explain the principles of the Programmed (conditioned) I/O transfer (Intel 8086). **(1p)**

Solutions

For problems 1 and 2 the following design steps should be followed and presented (in this very order):

1. **Schematic**
2. **Problem analysis (words, formulas / equations, time diagrams, …).**
3. **Logic design: flow-chart / logic scheme etc.**
4. **Implementation: code in C (explained with comments).**

In the following the key elements / hints for solving the exam subjects are presented. Solutions are not unique …

R1.1

Blue and Red LEDs should be connected to 2 digital ports pith PWM capabilities (the common anode is connected to Vcc and the independent cathodes are connected to the digital pins through some 270 ohm resistors). This way the LEDs can be turned on using a LOW signal applied to the pin, and OFF using a high signal (the PWM logic will be inverted: DC=0 ⇒ LED is ON (full brightness), while DC=255 ⇒ LED is OFF.

Because Temp. measurement should be triggered every 10 second, a timer can be used:

```
Timer t;
period = 10000 // 10 s
void setup() {
      …
      t.every(period, doProcess);
      …
}
```

The DC motor will be driven also by a PWM signal

- If T<=25 the motor should be stopped: DCmotor = 0
- If 25<T<=30 : DC = 25% (DCmotor = 64-1=63)
- If 30<T<=35 : DC = 50% (DCmotor = 128-1=127)
- If 35<T<=40 : DC = 75% (DCmotor = 192-1=191)
- If T>40 (i.e. 45) : DC = 100% (DCmotor = 256-1=255)

The above conditions can be simply implemented by restricting/saturating the temperature value below 25 and above 40 using the function constrain(val, LowerLimit, UperLimit): https://www.arduino.cc/en/Reference/Constrain and then by using the map function to map the above temperature ranges to digits: 0, 1, 2, 3, 4 and finally multiply these digits with 64 and subtract then with 1 to obtain the final DC:

```
      constrain(T,25, 45);
      digit = map(T,25,45,0,4);
      DCmotor=digit*64-1;
```

Also if-else statements can be used instead.

```
void doProcess(){
      int T = readTempInCelsius(5, pinSensor); // see C7p29

      //Generate LEDs color
      byte DCRed=map(T,-40,125,255,0);
      //DC=0 means full brightness due to the schematic (inverted logic)
      analogWrite(pinRedLED,DCRed);

      byte DCBlue=map(T,-40,125,0,255); // or DCBlue=255-DCRed;
      analogWrite(pinBlueLED,DCBlue);

      //display T on the LCD ….

      // Drive the motor
      constrain(T,25, 45);
      digit = map(T,25,45,0,4);
      DCmotor=digit*64-1; // DC for the Enable pin of the H bridge that drives the motor
      analogWrite(pinMotor,DCmotor); // https://www.arduino.cc/en/Reference/analogWrite
}
```

R1.2
LEDs should be connected to a digital ports (i.e. port A - for simplicity the anodes can be connected do the pins and the cathodes to the GND through some resistors). This way the LEDs can be turned on using a HIGH signal applied to the pin.

BTN will be connected to pin D0. In order to handle the button press event, interrupts should be used (the interrupt can be configured to be triggered on the RISING or FALLING edge – whatever).
Successive pushes of the button should change the value of T in the following sequence: 1,2,3,4,5,4,3,2,1,2,3, ..

The SW will be connected to pin D1. For the open/close event of the SW we can use an interrupt (configured to be triggered on CHANGE) or we can simply read the status of the pin (PIND). The second option seems much simpler and will be used.

To increment/decrement the software counter, we can use either a timer function (i.e. every) or using the millis() based scheme (C5a, page 5). In the following the timer based approach will be presented.

```
#include Timer.h
volatile byte T;  // T [s];
volatile byte inc; // used to toggle the value of T by button press
volatile myTimer; // timer object
volatile int ID;  // ID of the timer 'every'
byte counter;     // The software counter

void setup(){
      DDRA=0xFF; // output
      DDRD=0x00; // input
      MCUCR &= 0b11101111; //pull-up enable
      PORTD=0x03; // activate pull-up for D0 and D1
      counter = 0;
      T = 1;
      inc = 1;
      ID=myTimer.every(1000*T, changeCounterAndDsiplay); // initialize the timer for 1s
      attachInterrupt(D0, BTN1_ISR, FALLING);
      …
}

BTN1_ISR(void)
{     if ( T==5) inc=-1;
      if ( T==1) inc=1;
      T= T+inc;
      myTimer.stop(ID); //stop the current timer
      ID=myTimer.every(1000*T, changeCounterAndDsiplay); //re-start with the new T

}

void changeCounterAndDsiplay {
      byte statusSW = PIND & 0x02; // mask the bit corresponding to D1 (SW pin)
      if (statusSW == 0) // (0 = close / pull-up logic) - > increment
            counter ++;
      else //statusSW == 1 (1 = open / pull-up logic) - > decrement
            counter --;
      // because counter is of type byte, we do not need to bother with its overflow
      // its value will cycle

      PORTA = counter; // data persistency principle: is enough to do it only here
}

void loop(){
      myTimer.update();
}
```

R1.3 (only brief explain)
64 chips (8 columns and 8 rows; chips on a row connected in series; chips on a column connected in parallel)

- A1-13 are used for byte/word selection inside the chip

- A14-16 are used as inputs for the 3:8 decoder used to select each row (from 0 to 7). Each output of the decoder is used as a CS for the chips on the corresponding row.
- A17-19 are used as inputs for a "0"detect logic circuit (ex. OR gate) to enable the 3:8 DCD, thus mapping the memory block into the lower part of the memory space
- ~MRDC = (!M or ~RD) is connected to the ~OE of each chip (to control the memory read operations)
- No ~WR control signal is used

R1.4
The solution is similar to the one in C13p3

R1.5
The solution is in C11p15-16

R2.1

LEDs should be connected to 4 digital pins (for simplicity the anodes can be connected do the pins and the cathodes to the GND through some resistors). This way the LEDs can be turned on using a HIGH signal applied to the pin.

Option 1:
```
void loop {
      doProcess();
}
```

Option2 (does not keep the processor busy with our measurement & display)
```
Timer t;
period = … //  few ms
void setup() {
      …
      t.every(period, doProcess)
      …
}

void doProcess(){
      int distance = readDistance(10, pinSensor); // see C7p30
      if (distance >80)
            // all LEDS off and noTone
      if (distance <=80 && distance >=61 )
            // G is ON, YOR are off, Tone(pinSPK,100);
      if (distance <=60 && distance >=41 )
            // GY are ON, OR are off, Tone(pinSPK,200);
      if (distance <=40 && distance >=21 )
            // GYO are ON, R is off, Tone(pinSPK,300);
      if (distance <20)
            // GYOR are ON, Tone(pinSPK,400);

      //display distance on the LCD …

}
```

R2.2

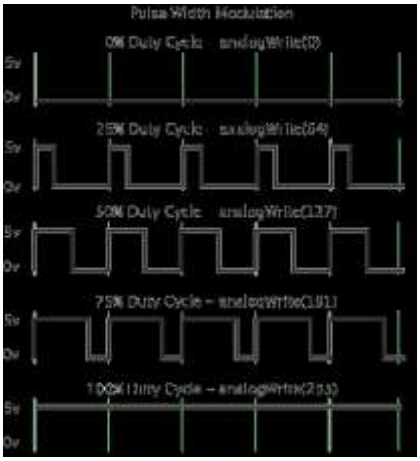
T = T_fadein + T_fadeout = 1 .. 10 s.
Where: T_fadein = T_fadeout=T/2

T is set by buttons, and shoud be done in ISRs associated to each button:
```
volatile int T; // T [s]; in the setup is initialized with 1

BTN1_ISR(void)
{     if ( T<10)  T++; }

BTN2_ISR(void)
{     if ( T>1)   T--; }
```

In the following we need to generate a signal with variable DC (duty cycle) and the increasing/decreasing of the DC should take place in T/2 s.
The simplest way to generate a signal with variable DC is by using analogWrite(pin, DC) function (https://www.arduino.cc/en/Reference/analogWrite: *After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal on most pins is approximately **490 Hz***).
So analogWrite generates a signal with a fixed frequency f≈500 Hz (signal period dt =1/f = 2 ms) and variable DC:



For the case T=1s (T/2=500ms) we can use a simple approach like that:

- We cycle 250 times (250 x 2 ms = 500ms) and in each cycle we increase DC from 0 .. 249 for fade in
- We cycle 250 times (250 x 2 ms = 500ms) and in each cycle we decrease DC from 249 .. 0 for fade out

And the code would be:

```
void loop {
// fade-in
for (int DC =0; DC< 250; DC++) // this cycle takes 250 x 2 ms = 500 ms
      analogWrite(pinLED,DC);
// fade-in
for (int DC =249; DC>=0; DC--) // this cycle takes 250 x 2 ms = 500 ms
      analogWrite(pinLED,DC);
}
```

How can we cope with the general case of T = 1 … 10s. The solution would be to insert a delay *delta* in the fade-in / fade-out cycles, after each call of analogeWrite(), thus increasing the duration of the PWM signal generation with the (DC fixed), from 1 clock period (2ms) to several ones (ex: 10 clocks for T=10s).

How we compute delta?

For T = 1s (T/2 = 500 ms) …. delta = 2ms
For T=10 s (T/2 = 5000 ms) … delta = 5000 ms / no_cycles = 5000/250 = 20 ms
(ex: for T=10s we generate for 20 ms a pwm signal with dt = 2ms and DC = i, i= 0 .. 250)

So, in order to get the value *delay* as a function of T we can use the following analytical formula:
delta = 2*T [ms]

```
void loop {
// fade-in
for (int DC =0; DC< 250; DC++) // this cycle takes 250 x delta ms
      {
      analogWrite(pinLED,DC);
      my_delay(2*T);
      }
// fade-out
for (int DC =249; DC>=0; DC--) // this cycle takes 250 x 2 ms = 500 ms
      {
      analogWrite(pinLED,DC);
      my_delay(2*T);
      }
}
```

```
void my_delay(unsigned long delta)
{
      unsigned long currentMillis = previousMillis = millis();  // current time
      while (currentMillis - previousMillis < delta)
                  previousMillis = currentMillis;
}
```

Another solution (by Bucur Andrei):

```
void loop()
{
      current = previous = millis();
      time = current-previous;
      while (time <= 1000*T/2) {
            fade=map(time, 0, 1000*T/2, 0, 255);
            analogWrite(pinLED, fade);
            current = millis();
            time = current – previous;
            }
      current = previous = millis();
      time = current-previous;
      while (time <= 1000*T/2) {
            fade=map(time, 0, 1000*T/2, 255, 0);
            analogWrite(pinLED, fade);
            current = millis();
            time = current – previous;
            }
}
```

Or a similar one (by Zaharia Teofil):

```
byte currentState = 0;
void halfFade(minPWM, maxPWM) {
      unsigned long t0, t1, deltat;
      t0 = millis();
      do {
            t1 = millis();
            deltat = t1-t0;
            curentState = map(deltat, 0,1000*T/2, minPWM, maxPWM);
            analogWrite(pinLED, currentState);
      while (delta <1000*T/2)
      }
}
void loop () {
      halfFade(0, 255); //fade in
      halfFade(255, 0); //fade in
}
```

R2.3 (only brief explain)

32 chips (2 columns and 16 row, chips on arrow connected in series, chips on a column connected in parallel)

- A1-13 are used for byte/word selection inside the chip
- A14-17 are used as inputs for the 4:16 decoder used to select each row (from 0 to 15).. Each output of the decoder is used as a CS for the chips on the corresponding row.
- A18-19 are used as inputs for a “1”detect logic circuit (ex. NAND gate) to enable the 4:16 DCD, thus mapping the memory block into the upper part of the memory space
- ~HWR = (~MWRC or ~BHE) is used to control the ~WE on the chips from the column connected to the High data bus (D8-15)
- ~LWR = (~MWRC or A0) is used to control the ~WE on the chips from the column connected to the Low data bus (D0-7)
- ~MRDC = (!M or ~RD) is connected to the ~OE of each chip (to control the memory read operations)

R2.4

The solution is similar to the one in C13p2

R2.5

The solution is in in C11p15-16