PROJECT

Fall - 2022/2023

MKT3811 – Microprocessors and Programming

Project Report

Submitted By: Göktuğ Can Şimay

Lab Partners: Ali Doğan, Basel Hadri

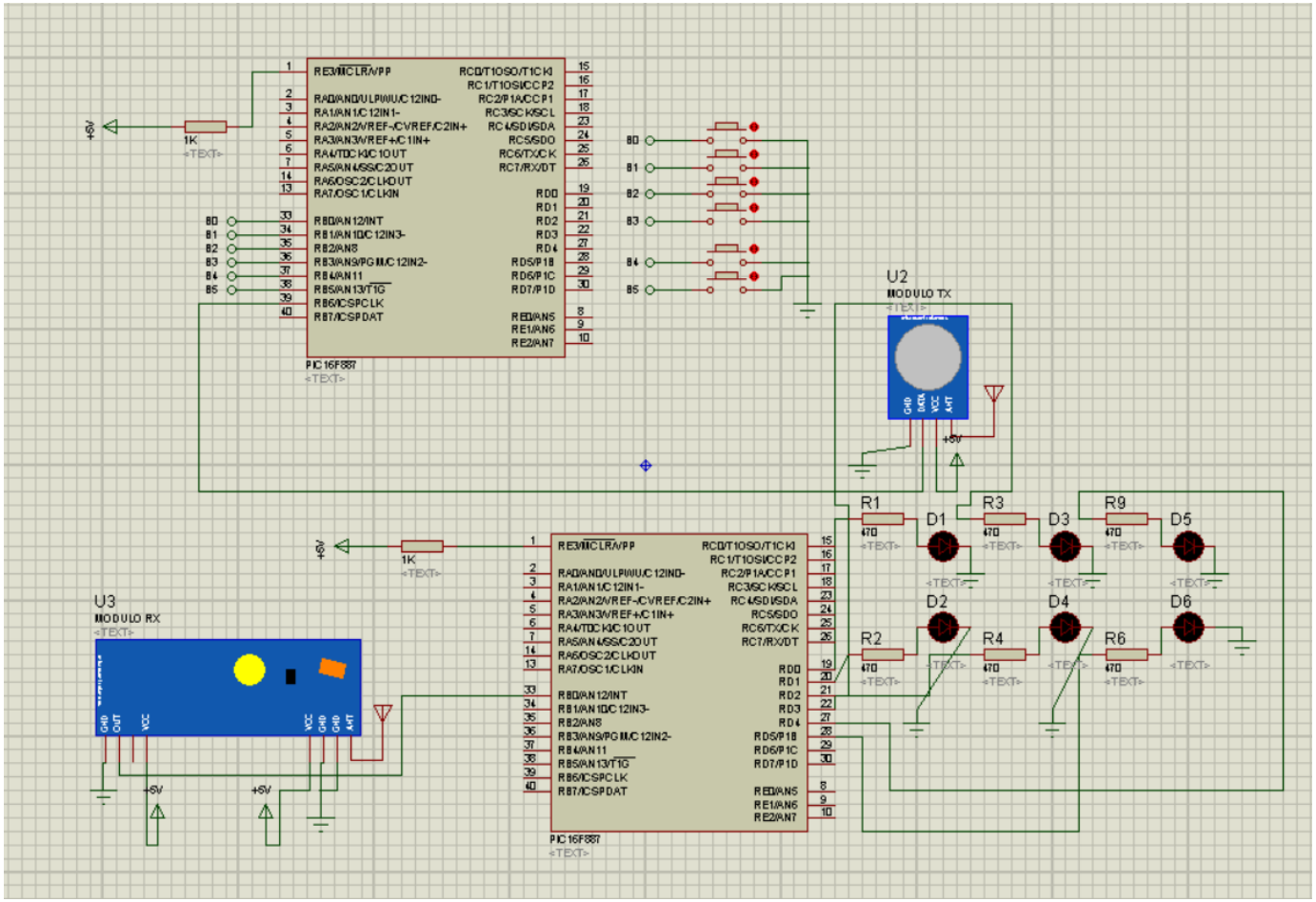
Group Number: 22

Student ID: 22067606

Date: 15.12.2022

Descriptions:

The student is asked to design a controller with 6 buttons on the transmitter side. There should be 6 corresponding LEDs on the receiver side. Each button should light up the corresponding LED. The controller must be in good aesthetic condition.



Proteus Schematic Design

2 tane mikrodenetleyici kullandım. Verici devrede görüldüğü gibi buton bağlantılarını verdim. Port_b_pullups(x) komutunu kullandığım için ekstradan direnç bağlantısı yapmadım. ***RX TX modülleri Proteus ISIS kütüphanesinde yoktu ben de internetten bulup ekledim ve simülasyonumu tıpkı gerçek hayattaymış gibi gerçekleştirdim. Çünkü bu modüller projemde kullandığım 433mHz RF alıcı verici komponentlerle aynı prensipte çalışıyordu.*** *Data pinlerini de kodumda yazdığım şekilde bağlamış oldum.*

```
/* RF transmitter using PIC16F887 microcontroller CCS C code
   This RF transmitter is based on NEC protocol
   Internal oscillator used @ 8MHz
*/
//Her kodu tek tek yorum satırıyla açıklamaya çalıştım.
#include <16F887.h>
#fuses NOMCLR, NOBROWNOUT, NOLVP, INTRC_IO
#use delay(clock = 8MHz)
#use fast_io(B)

void send_signal(unsigned int32 number){
  int8 i;
  // Send 9ms pulse
  output_high(PIN_B6);
  delay_ms(9);
  // Send 4.5ms space
  output_low(PIN_B6);
  delay_us(4500);
  // Send data (32 bits)
  for(i = 0; i < 32; i++){
    // If bit is 1 send 560us pulse and 1680us space
    if(bit_test(number, 31 - i)){
      output_high(PIN_B6);
      delay_us(560);
      output_low(PIN_B6);
      delay_us(1680);
    }
    // If bit is 0 send 560us pulse and 560us space
```

```
          else{
            output_high(PIN_B6);
            delay_us(560);
            output_low(PIN_B6);
            delay_us(560);
          }
        }
        // Send end bit
        output_high(PIN_B6);
        delay_us(560);
        output_low(PIN_B6);
        delay_us(560);
      }
      void main() {
        setup_oscillator(OSC_8MHZ);                    // Set internal
oscillator to 8MHz
        output_b(0);
        set_tris_b(0x3F);                             // Configure RB0,
RB1, RB2, RB3, RB4 and RB5 as inputs
        port_b_pullups(0x3F);                         // Enable internal
pull-ups for pins RB0,RB1,RB2,RB3,RB4 and RB5
        while(TRUE){
          if(!input(PIN_B0)){                          // If RB0 button
is pressed
            send_signal(0x00FF00FF);
            delay_ms(500);
          }
          if(!input(PIN_B1)){                          // If RB1 button
is pressed
            send_signal(0x00FF807F);
```

```
            delay_ms(500);
          }
          if(!input(PIN_B2)){                          // If RB2 button
is pressed
            send_signal(0x00FF40BF);
            delay_ms(500);
          }
          if(!input(PIN_B3)){                          // If RB3 button
is pressed
            send_signal(0x00FF20DF);
            delay_ms(500);
          }
          if(!input(PIN_B4)){                          // If RB4 button
is pressed
            send_signal(0x00FFA05F);
            delay_ms(500);
          }
          if(!input(PIN_B5)){                          // If RB5 button
is pressed
            send_signal(0x00FF609F);
            delay_ms(500);
          }
        }
      }
```

```
/* RF Receiver using PIC16F887 microcontroller CCS C code
   This RF receiver is based on NEC protocol
   Internal oscillator used @ 8MHz
*/
//Her kodu tek tek yorum satırıyla açıklamaya çalıştım.
#include <16F887.h>
#fuses NOMCLR, NOBROWNOUT, NOLVP, INTRC_IO
#use delay(clock = 8MHz)
#use fast_io(D)

short code_ok = 0;
unsigned int8 nec_state = 0, i;
unsigned int32 rf_code;
#INT_EXT                                                    // External
interrupt
void ext_isr(void){
  unsigned int16 time;
  if(nec_state != 0){
    time = get_timer1();                              // Store Timer1
value
    set_timer1(0);                                   // Reset Timer1
  }
  switch(nec_state){
    case 0 :                                        // Start receiving
IR data (we're at the beginning of 9ms pulse)
      setup_timer_1( T1_INTERNAL | T1_DIV_BY_2 );   // Enable Timer1
module with internal clock source and prescaler = 2
      set_timer1(0);                                   // Reset Timer1
value
```

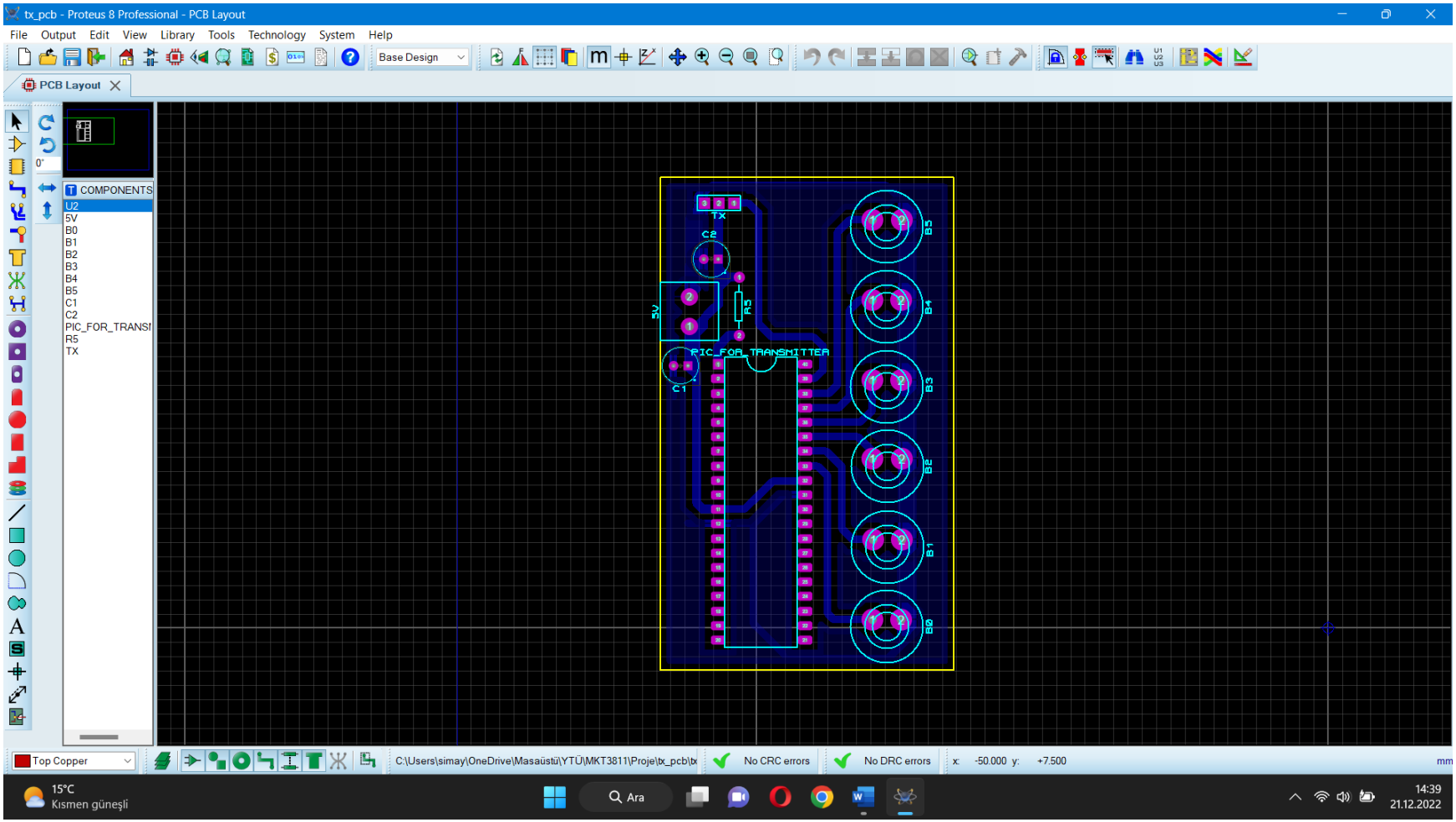
```
            nec_state = 1;                                  // Next state: end
of 9ms pulse (start of 4.5ms space)
            i = 0;
            ext_int_edge( H_TO_L );                         // Toggle external
interrupt edge
            return;
          case 1 :                                            // End of 9ms
pulse
            if((time > 9500) || (time < 8500)){             // Invalid interval
==> stop decoding and reset
              nec_state = 0;                                 // Reset decoding
process
              setup_timer_1(T1_DISABLED);                    // Stop Timer1
module
            }
            else
              nec_state = 2;                                // Next state: end
of 4.5ms space (start of 560µs pulse)
            ext_int_edge( L_TO_H );                         // Toggle external
interrupt edge
            return;
          case 2 :                                             // End of 4.5ms
space
            if((time > 5000) || (time < 4000)){             // Invalid interval
==> stop decoding and reset
              nec_state = 0;                                 // Reset decoding
process
              setup_timer_1(T1_DISABLED);                    // Stop Timer1
module
              return;
            }
            nec_state = 3;                                  // Next state: end
of 560µs pulse (start of 560µs or 1680µs space)
```

```
            ext_int_edge( H_TO_L );                    // Toggle external
interrupt edge
            return;
          case 3 :                                       // End of 560µs
pulse
            if((time > 700) || (time < 400)){          // Invalid interval
==> stop decoding and reset
              nec_state = 0;                            // Reset decoding
process
              setup_timer_1(T1_DISABLED);               // Disable Timer1
module
            }
            else
              nec_state = 4;                            // Next state: end
of 560µs or 1680µs space
            ext_int_edge( L_TO_H );                    // Toggle external
interrupt edge
            return;
          case 4 :                                       // End of 560µs
or 1680µs space
            if((time > 1800) || (time < 400)){         // Invalid interval
==> stop decoding and reset
              nec_state = 0;                            // Reset decoding
process
              setup_timer_1(T1_DISABLED);               // Disable Timer1
module
              return;
            }
            if( time > 1000)                            // If space width
> 1ms (short space)
              bit_set(rf_code, (31 - i));               // Write 1 to bit
(31 - i)
```

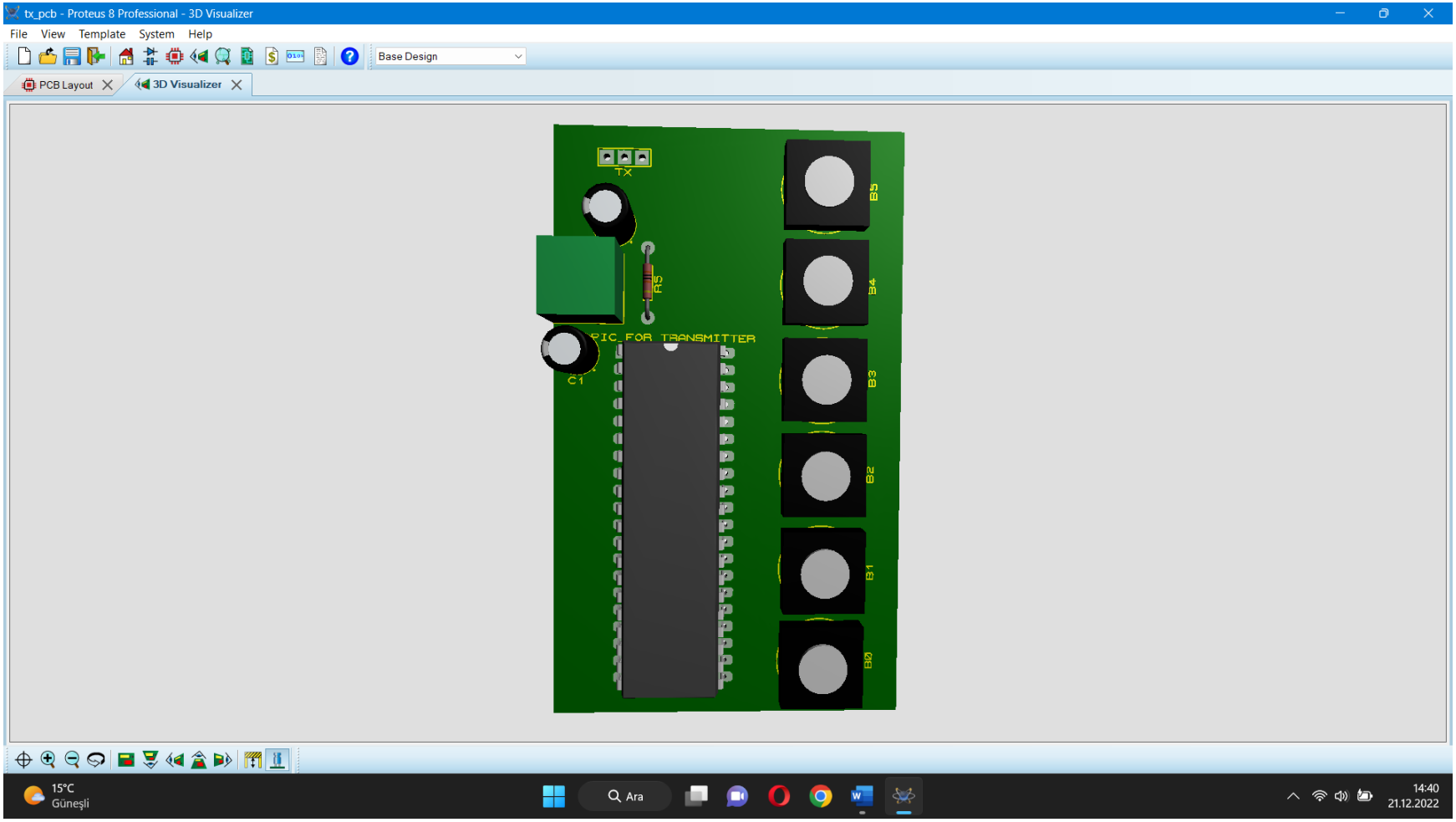
```
            else                                        // If space width
< 1ms (long space)
              bit_clear(rf_code, (31 - i));             // Write 0 to bit
(31 - i)
            i++;
            if(i > 31){                                 // If all bits are
received
              code_ok = 1;                              // Decoding process
OK
              disable_interrupts(INT_EXT);                // Disable the
external interrupt
            }
            nec_state = 3;                              // Next state: end
of 560µs pulse (start of 560µs or 1680µs space)
            ext_int_edge( H_TO_L );                     // Toggle external
interrupt edge
        }
      }
      #INT_TIMER1                                       // Timer1 interrupt
(used for time out)
      void timer1_isr(void){
        nec_state = 0;                                  // Reset decoding
process
        ext_int_edge( L_TO_H );                             // External
interrupt edge from high to low
        setup_timer_1(T1_DISABLED);                     // Disable Timer1
module
        clear_interrupt(INT_TIMER1);                    // Clear Timer1
interrupt flag bit
      }
      void main() {
        setup_oscillator(OSC_8MHZ);                     // Set internal
oscillator to 8MHz
```

```
        output_d(0);                                    // PORTD initial
state
        set_tris_d(0);                                 // Configure PORTD
pins as outputs
        enable_interrupts(GLOBAL);                      // Enable global
interrupts
        enable_interrupts(INT_EXT_L2H);                // Enable external
interrupt
        clear_interrupt(INT_TIMER1);                    // Clear Timer1
interrupt flag bit
        enable_interrupts(INT_TIMER1);                  // Enable Timer1
interrupt
        while(TRUE){
          if(code_ok){                                     // If the mcu
successfully receives NEC protocol message
            code_ok = 0;                                // Reset decoding
process
            nec_state = 0;
            setup_timer_1(T1_DISABLED);                 // Disable Timer1
module
            if(rf_code == 0x00FF00FF)
              output_toggle(PIN_D0);
            if(rf_code == 0x00FF807F)
              output_toggle(PIN_D1);
            if(rf_code == 0x00FF40BF)
              output_toggle(PIN_D2);
            if(rf_code == 0x00FF20DF)
              output_toggle(PIN_D3);
            if(rf_code == 0x00FFA05F)
              output_toggle(PIN_D4);
            if(rf_code == 0x00FF609F)
```

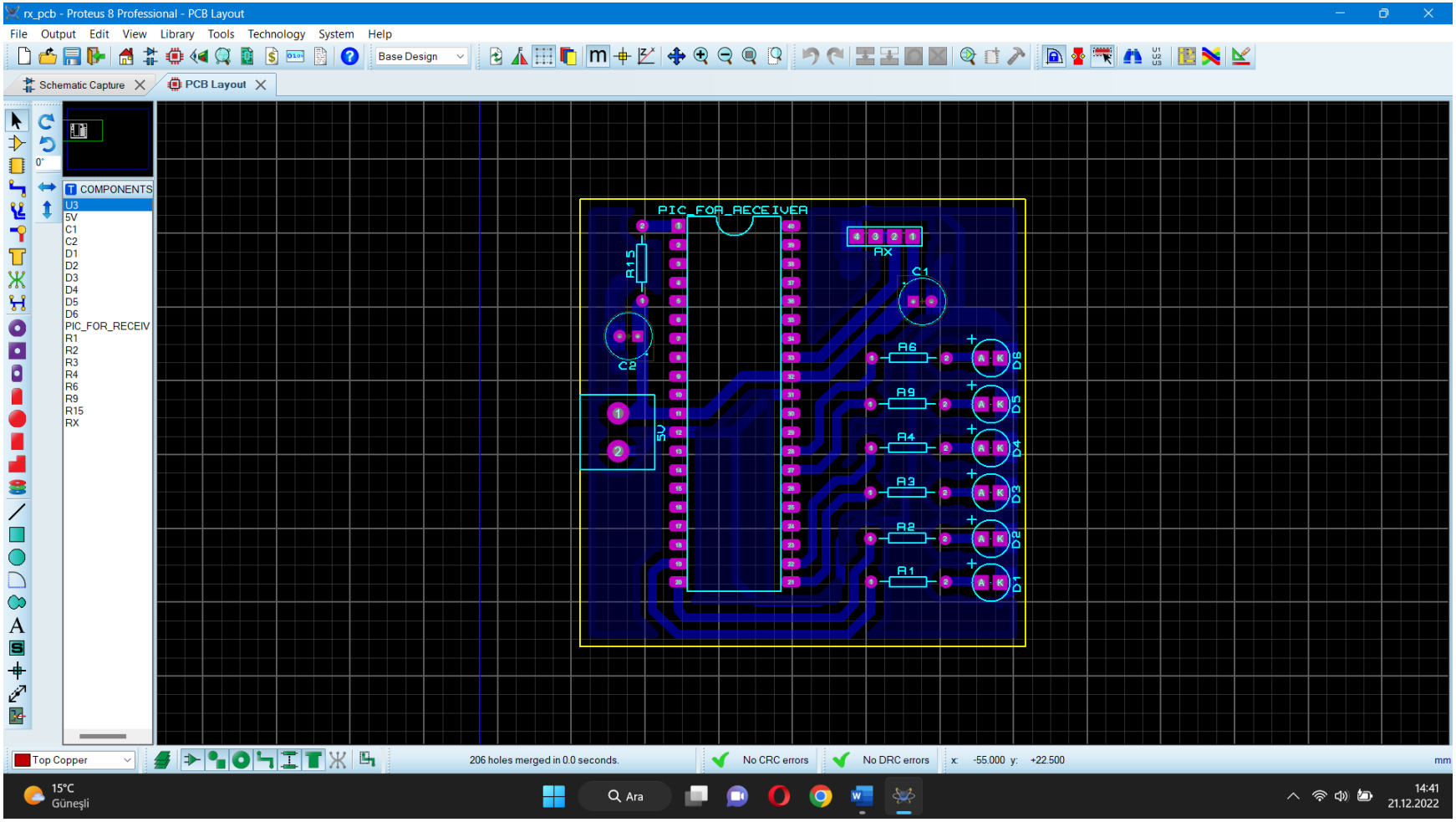
```
            output_toggle(PIN_D5);
          enable_interrupts(INT_EXT_L2H);          // Enable external
interrupt
        }
      }
    }
```



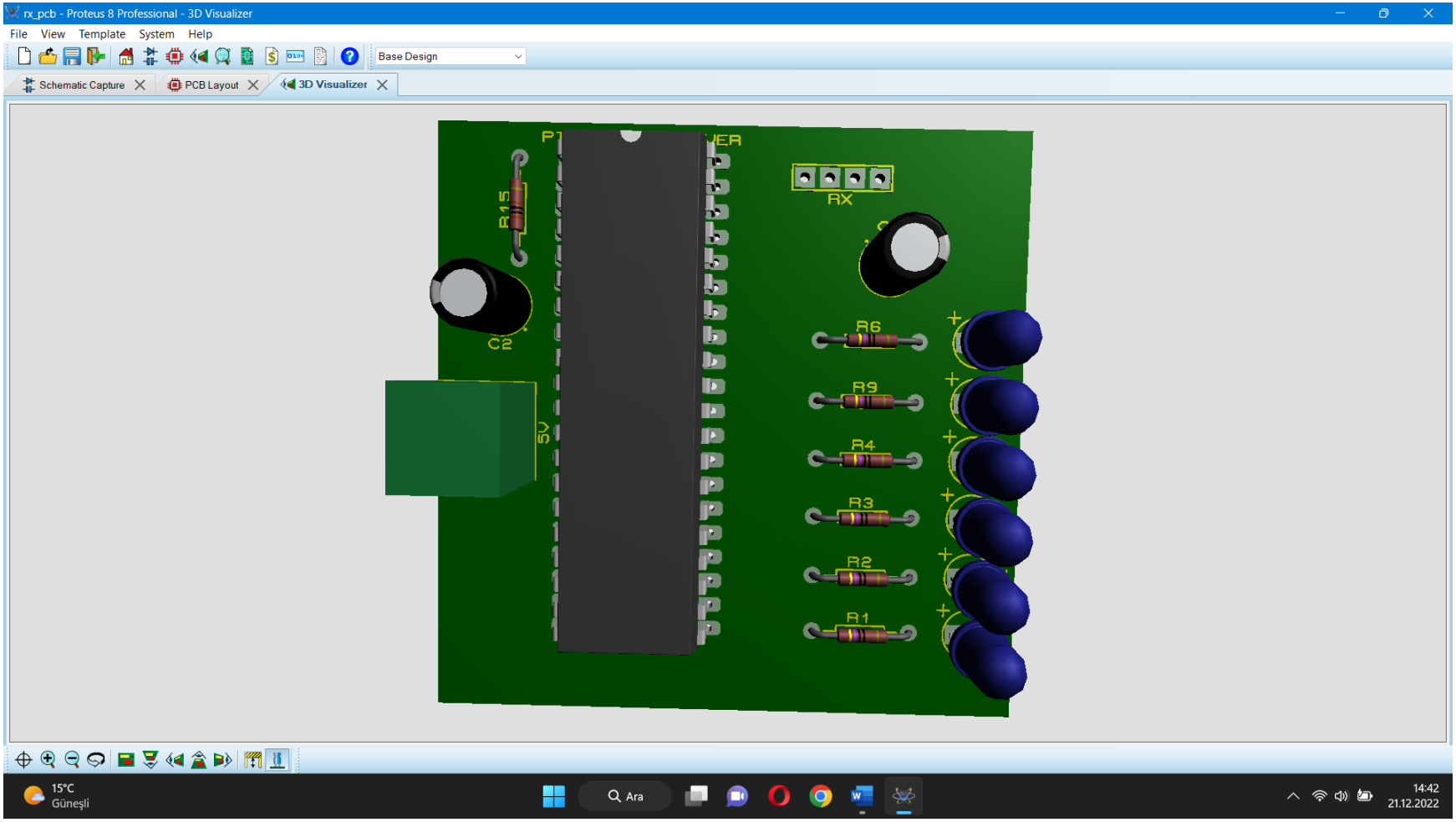
Proteus ARES Dijital Devre Baskı Çizimi - Verici Devre



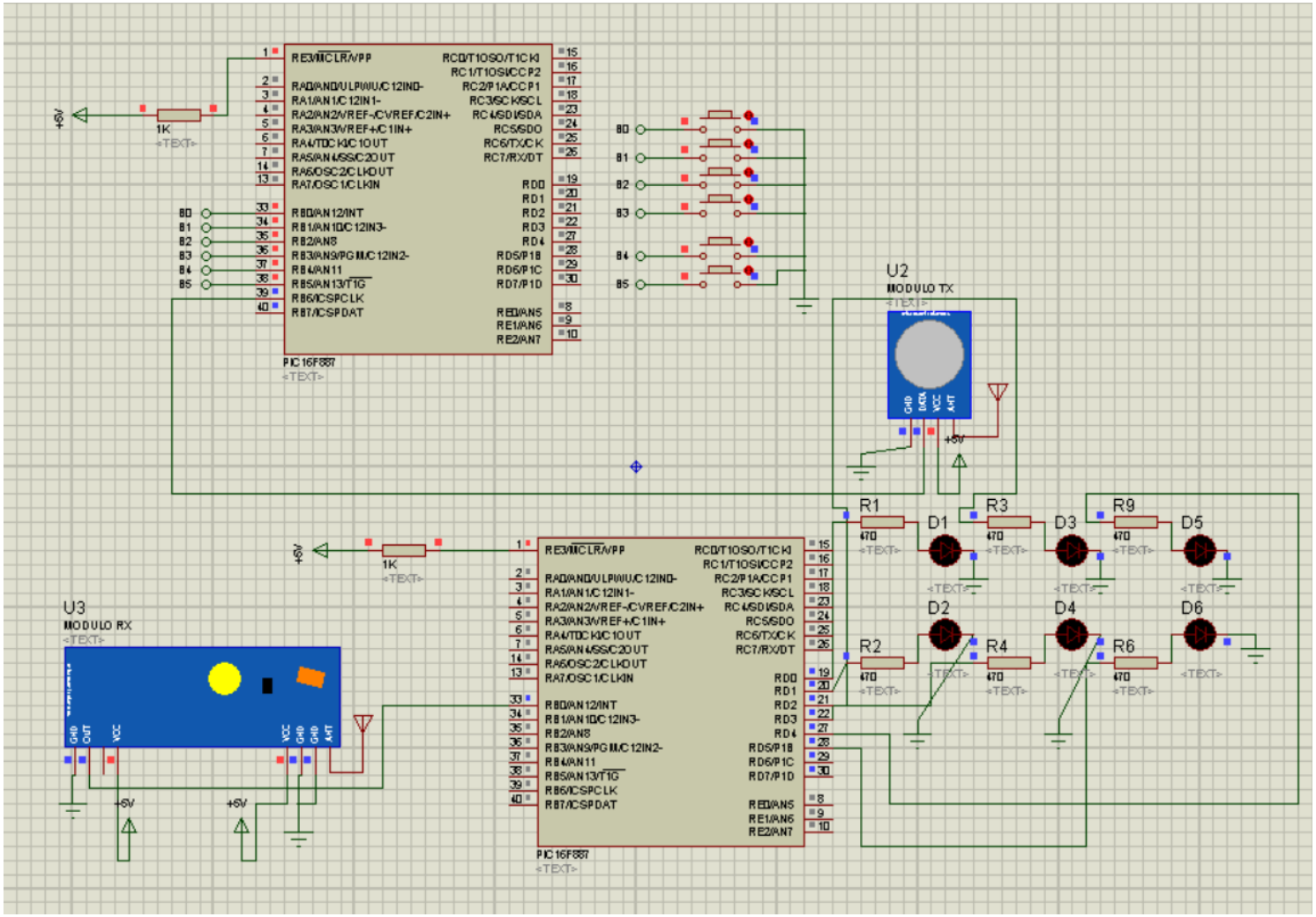
Proteus ARES Dijital Devre Baskı Çizimi – PCB 3 Boyutlu Görseli



Proteus ARES Dijital Devre Baskı Çizimi - Alıcı Devre

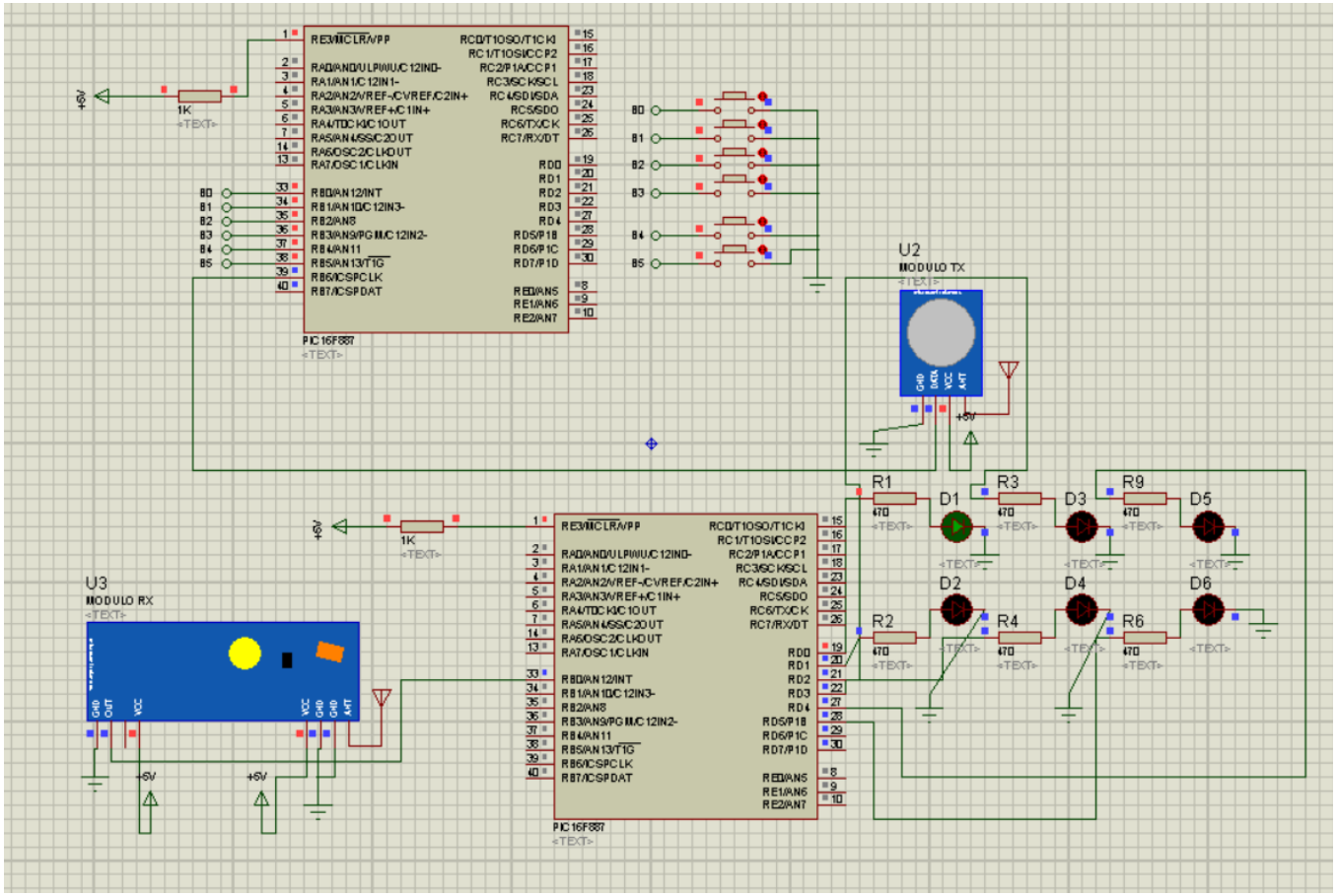


Proteus ARES Dijital Devre Baskı Çizimi – PCB 3 Boyutlu Görseli



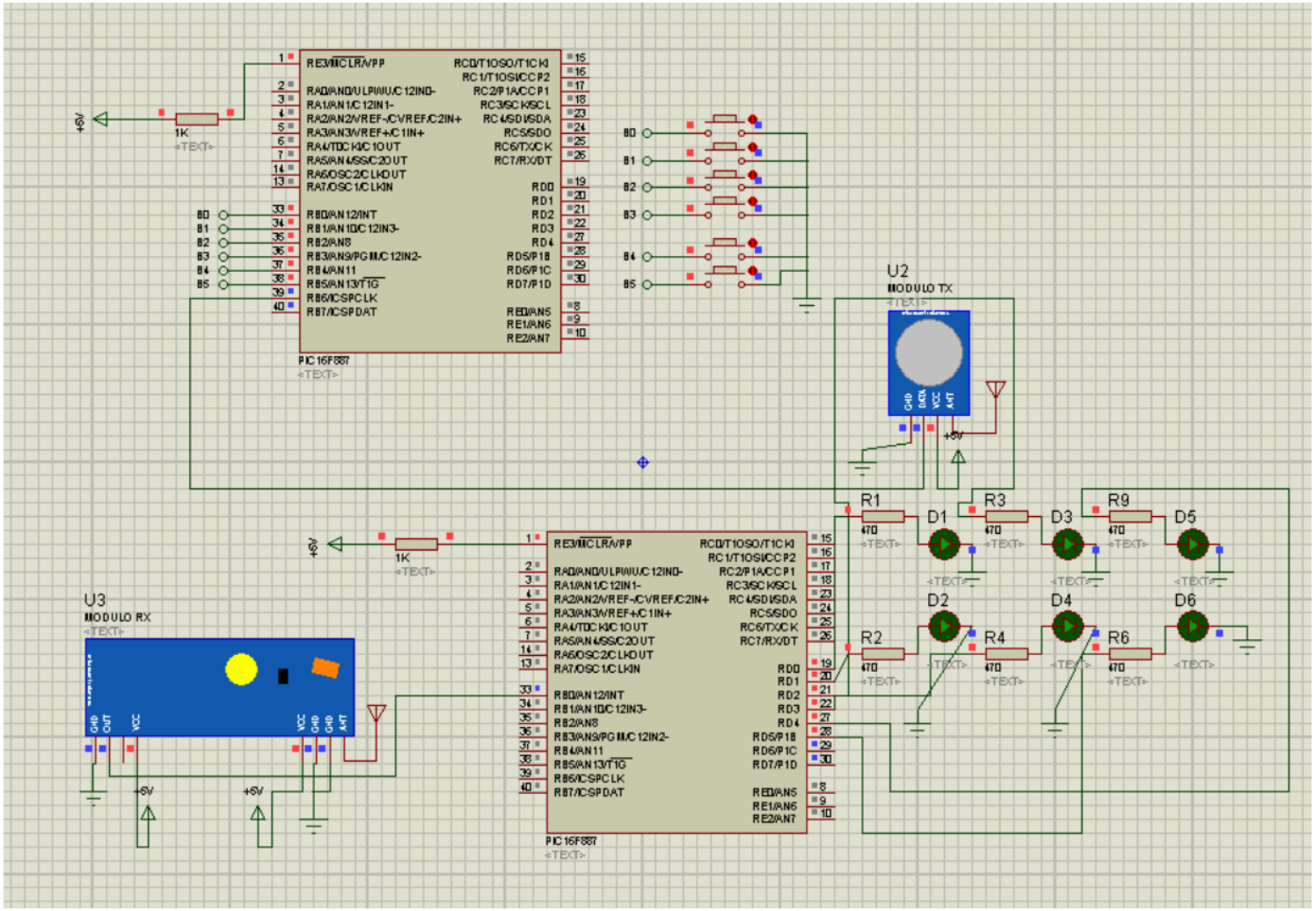
Proteus Simulation Part 1

Simülasyonu başlatınca I/O'lar daha net görünüyor.



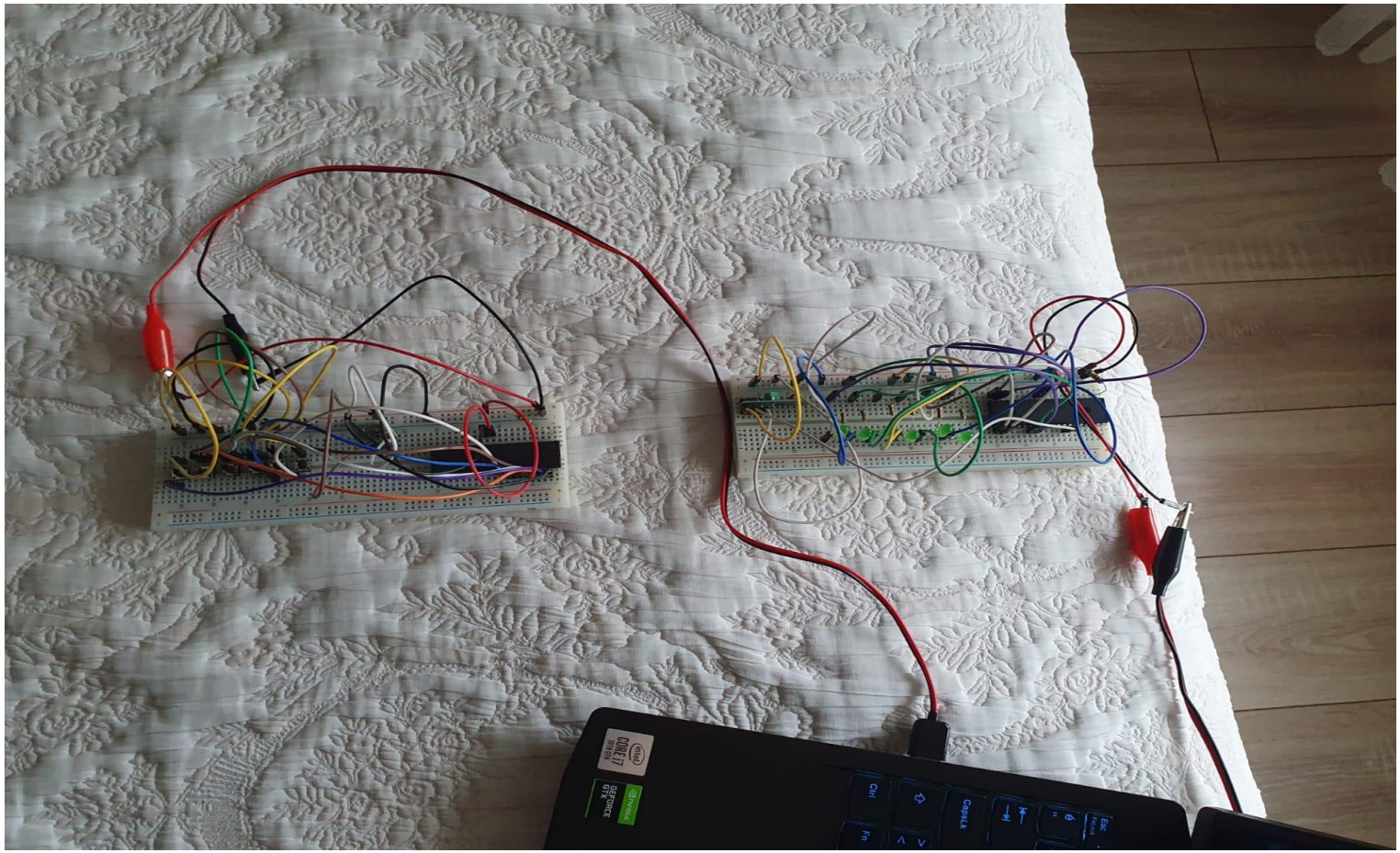
Proteus Simulation Part 2

İlk butonum RB0 bağlı olan, birinci sıradaki ledimi yakıyor.



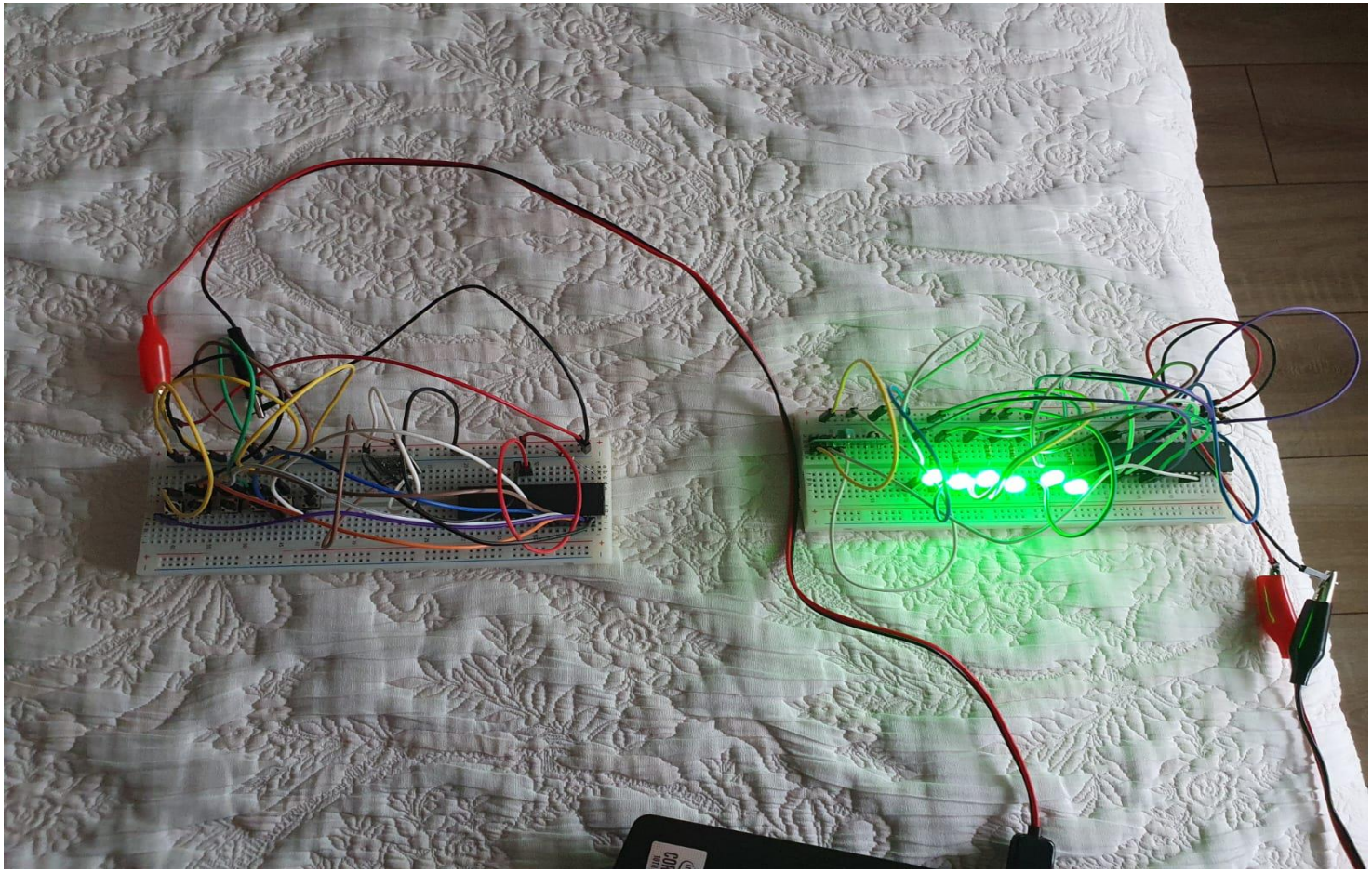
Proteus Simulation Part 3

Bütün butonlara basınca her led yandı. Söndürmek için tekrar basmamız gerekir. Çünkü output toggle ederken while değil if içine alarak yazdım.



Real Life Breadboard Testing Part 1

PCB yapmadan önce kodumu bir de gerçek hayatta test etmek için breadboard üzerine kurdum ve sorunsuz çalıştı.



Real Life Breadboard Testing Part 2

Dijital devre baskısı yapmadan önce kodumu bir de gerçek hayatta test etmek için breadboard üzerine kurdum ve sorunsuz çalıştı.