Universidad del Valle de Guatemala
Facultad de Ingeniería Electrónica
electrónica Digital 2
Catedrático Pablo Mazariegos
Aldo Renato Avila Escobar – 21257
Byron Estuardo Barrientos Perez - 21457

1 de junio del 2023

Entregable Final

Explicación del código

```
25 #define LED_PIN GPIO_PIN_6
26 #define LED_PORT GPIO_PORTA_BASE
27 #define LED_PIN_FULL GPIO_PIN_7
28 #define LED_PORT_FULL GPIO_PORTA_BASE
29 #define PHOTO_PIN GPIO_PIN_4
30 #define PHOTO_PORT GPIO_PORTB_BASE
```

Se declaran los puertos y pines que se usaran para las fotorresistencias y los leds que indicaran si el espacio se encuentra disponible o no. Declararlos de esta forma ayuda a que, si se necesitara hacer un cambio en los pines a usar, solo se tendrá que cambiar en esta sección del código para que sea efectivo. Esta misma declaración se hizo tres veces mas para las otras fotorresistencias.

```
53 #define ADC_SEQ_NUM 0
54 #define ADC_SEQ_NUM2 1
55 #define ADC_SEQ_NUM3 2
56 #define ADC_SEQ_NUM4 3
```

Nos ayuda a saber la secuencia en la que se ejecutara cada canal del ADC.

```
61  // Configurar el sistema
62  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
63
64  // Habilitar los periféricos necesarios
65  SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
66  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
67  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
68  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
69  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
70  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
```

El setup del código comienza con la configuración del oscilador, en este caso se usará uno interno usando el PLL. De igual forma se habilita los tiempos de reloj para cada uno de los puertos a usar.

```
72  // Configurar los pines
73  GPIOPinTypeADC(PHOTO_PORT, PHOTO_PIN);
74  GPIOPinTypeGPIOOutput(LED_PORT, LED_PIN);
75  GPIOPinTypeGPIOOutput(LED_PORT_FULL, LED_PIN_FULL);
```

Se configuran los pines que se usaran como salidas y como ADC. Esto se repite tres veces más para cada uno de los canales ADC.

```
89  // Configurar el ADC
90  ADCSequenceConfigure(ADC0_BASE, ADC_SEQ_NUM, ADC_TRIGGER_PROCESSOR, 0);
91  ADCSequenceStepConfigure(ADC0_BASE, ADC_SEQ_NUM, 0, ADC_CTL_CH10 | ADC_CTL_IE | ADC_CTL_END);
92  ADCSequenceConfigure(ADC0_BASE, ADC_SEQ_NUM2, ADC_TRIGGER_PROCESSOR, 0);
93  ADCSequenceStepConfigure(ADC0_BASE, ADC_SEQ_NUM2, 0, ADC_CTL_CH11 | ADC_CTL_IE | ADC_CTL_END);
94  ADCSequenceConfigure(ADC0_BASE, ADC_SEQ_NUM3, ADC_TRIGGER_PROCESSOR, 0);
95  ADCSequenceStepConfigure(ADC0_BASE, ADC_SEQ_NUM3, 0, ADC_CTL_CH1 | ADC_CTL_IE | ADC_CTL_END);
96  ADCSequenceConfigure(ADC0_BASE, ADC_SEQ_NUM4, ADC_TRIGGER_PROCESSOR, 0);
97  ADCSequenceStepConfigure(ADC0_BASE, ADC_SEQ_NUM4, 0, ADC_CTL_CH2 | ADC_CTL_IE | ADC_CTL_END);
98
99  ADCSequenceEnable(ADC0_BASE, ADC_SEQ_NUM);
100 ADCSequenceEnable(ADC0_BASE, ADC_SEQ_NUM2);
101 ADCSequenceEnable(ADC0_BASE, ADC_SEQ_NUM3);
102 ADCSequenceEnable(ADC0_BASE, ADC_SEQ_NUM4);
```

Aquí se configura el ADC, cuando es que se va a dar la lectura y que canales son los que se van a utilizar. En la parte de abajo se habilita la secuencia en la que se estarán realizando las lecturas.

```
104 // se habilita y configura el UART
105 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
106 SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
107 GPIOPinConfigure(GPIO_PB0_U1RX);
108 GPIOPinConfigure(GPIO_PB1_U1TX);
109 GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0|GPIO_PIN_1);
110 UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8|UART_CONFIG_STOP_ONE|UART_CONFIG_PAR_NONE));
111 UARTEnable(UART1_BASE);
```

Se habilita y configura el UART a utilizar, en este caso se utilizará el UART1, se configuran los pintes 0 y 1 del puerto b como RX y TX respectivamente. Se tuvo que especificar que estos pines se iban a usar para el UART1 debido a que este UART esta multiplexado con otro par de pines. No se utilizó una interrupción ya que solo se iban a estar enviando datos, con un baudrate de 115200 sin paridad.

```
125     // Leer el valor de la fotoresistencia
126     ADCProcessorTrigger(ADC0_BASE, ADC_SEQ_NUM);
127     while(!ADCIntStatus(ADC0_BASE, ADC_SEQ_NUM, false)) {}
128     uint32_t photo_value;
129     ADCSequenceDataGet(ADC0_BASE, ADC_SEQ_NUM, &photo_value);
130
131     ADCProcessorTrigger(ADC0_BASE, ADC_SEQ_NUM2);
132     while(!ADCIntStatus(ADC0_BASE, ADC_SEQ_NUM2, false)) {}
133     uint32_t photo_value2;
134     ADCSequenceDataGet(ADC0_BASE, ADC_SEQ_NUM2, &photo_value2);
135
136     ADCProcessorTrigger(ADC0_BASE, ADC_SEQ_NUM3);
137     while(!ADCIntStatus(ADC0_BASE, ADC_SEQ_NUM3, false)) {}
138     uint32_t photo_value3;
139     ADCSequenceDataGet(ADC0_BASE, ADC_SEQ_NUM3, &photo_value3);
140
141     ADCProcessorTrigger(ADC0_BASE, ADC_SEQ_NUM4);
142     while(!ADCIntStatus(ADC0_BASE, ADC_SEQ_NUM4, false)) {}
143     uint32_t photo_value4;
144     ADCSequenceDataGet(ADC0_BASE, ADC_SEQ_NUM4, &photo_value4);
```

En el loop principal se llevan a cabo las lecturas de cada uno de los canales del ADC0 indicados. Mientras se realiza la lectura, el código se queda en ciclo while, para después el dato ser guardado en la variable indicada, como por ejemplo “photo_value”.

```
146     // Encender la LED si la luz es suficiente
147     if (photo_value > 1500) {
148         GPIOPinWrite(LED_PORT, LED_PIN, LED_PIN);
149         GPIOPinWrite(LED_PORT_FULL, LED_PIN_FULL, 0);
150         UARTCharPut(UART1_BASE, 'a');
151     } else {
152         GPIOPinWrite(LED_PORT_FULL, LED_PIN_FULL, LED_PIN_FULL);
153         GPIOPinWrite(LED_PORT, LED_PIN, 0);
154         UARTCharPut(UART1_BASE, 'b');
155     }
```

Para cada una de las variables se realiza una comparación. En caso de que la lectura sea mayor a 1500, la LED que indica que el espacio esta libre, estará encendida. Esto también quiere decir que la fotorresistencia se encuentra recibiendo algún tipo de luz. Además de encender la LED, la tiva enviara un carácter ‘a’ por UART para indicarle al ESP que este espacio esta libre. Si la lectura es menor a 1500 el LED que indica que el espacio está ocupado se encenderá y se enviará un carácter ‘b’ para indicarle al ESP que el espacio está ocupado. Cabe resaltar que esto ocurrirá cuando la fotorresistencia no este recibiendo algún tipo de luz.

```
 8   #define UART1_BAUDRATE 115200
 9   #define UART1_TX_PIN 2
10   #define UART1_RX_PIN 4
11
12   #define UART2_BAUDRATE 115200
13   #define UART2_TX_PIN 17
14   #define UART2_RX_PIN 16
15
16   HardwareSerial SerialPort1(1);
17   HardwareSerial SerialPort2(2);
```

En la parte del ESP32 se comienza declarando que pines serán usados para cada UART y cual es el RX y TX. También los baudrate, en este caso es de 115200 para coincidir con el colocado en la Tiva C.

```
24  const char* ssid = "iPhonedeAldo";  // Enter your SSID here
25  const char* password = "Jamondepavo03";  //Enter your Password here
26
27  const int segmentPins[] = {13, 14, 12, 5, 18, 19, 27, 26};
28
29  int currentDigit = 0;
30
31  // Números en código de segmento común
32  const byte digitPatterns[] = {
33    B00111111,  // 0
34    B00000110,  // 1
35    B01011011,  // 2
36    B01001111,  // 3
37    B01100110,  // 4
38    B01101101,  // 5
39    B01111101,  // 6
40    B00000111,  // 7
41    B01111111,  // 8
42    B01101111   // 9
43  };
```

Aquí se declaran las variables a usar, en este caso también se coloca el nombre de la red wifi a usar con su contraseña. También se declaran los pines a usar para el display de 7 segmentos con el contador que nos dirá cual es el numero de parqueos disponibles. También está la lista con los bits que nos indicaran que pines se tendrán que encender para que se despliegue el numero deseado. Cabe resaltar que estos están en el orden contrario que los puertos por como se iba asignando cada valor a su respectivo pin.

```
46  char parqueo1;
47  char parqueo2;
48  char P1;
49  char P2;
50  char P3;
51  char P4;
52  char P5;
53  char P6;
54  char P7;
55  char P8;
```

```
57  int FlagP1;
58  int FlagP2;
59  int FlagP3;
60  int FlagP4;
61  int FlagP5;
62  int FlagP6;
63  int FlagP7;
64  int FlagP8;
65
66  WebServer server(80);
```

Se declaran las variables en las cuales se guardaran los valores obtenidos del UART. La variable parqueo1 obtendra los datos del UART1 y la variable parqueo2 los datos del UART2. Las variables que comienzan con Flag nos ayudaran con el contador, serviran de antirrebote. El webserver esta por defaul el 80.

```
72  SerialPort1.begin(UART1_BAUDRATE, SERIAL_8N1, UART1_RX_PIN, UART1_TX_PIN);
73  SerialPort2.begin(UART2_BAUDRATE, SERIAL_8N1, UART2_RX_PIN, UART2_TX_PIN);
74  // Inicializa la comunicación serial en el puerto UART1 y UART2 con las velocidades de baudios y pines especificados
75  // SERIAL_8N1 indica 8 bits de datos, sin paridad y 1 bit de parada
76  // Los pines especificados se utilizarán para la comunicación UART1 y UART2
77  for (int i = 0; i < 8; i++) {
78    pinMode(segmentPins[i], OUTPUT);
79  }
```

Se configuran e inician los UART a utilizar, también se colocan los pines a utilizar en el display de 7 segmentos como salidas.

```
81    Serial.begin(9600);
82    Serial.println("Try Connecting to ");
83    Serial.println(ssid);
84
85    // Connect to your wi-fi modem
86    WiFi.begin(ssid, password);
87
88    // Check wi-fi is connected to wi-fi network
89    while (WiFi.status() != WL_CONNECTED) {
90      delay(1000);
91      Serial.print(".");
92    }
93    Serial.println("");
94    Serial.println("WiFi connected successfully");
95    Serial.print("Got IP: ");
96    Serial.println(WiFi.localIP());  //Show ESP32 IP on serial
97
98    server.on("/", handle_OnConnect); // Directamente desde e.g. 192.168.0.8
99
100   server.onNotFound(handle_NotFound);
101
102   server.begin();
103   Serial.println("HTTP server started");
104   delay(100);
```

Se inicia nuestro servidor WIFI en el ESP32 y pedimos que mientras se conecta a la red se mantenga en el ciclo while para asegurar la conexión. De igual forma pedimos que imprima en la consola la dirección IP de la pagina para poder accederla luego.

```
113   if (SerialPort1.available()) {
114     parqueo1 = SerialPort1.read();
115     Serial.print("UART1: ");
116     Serial.println(parqueo1);
117   }
118
119   if (SerialPort2.available()) {
120     parqueo2 = SerialPort2.read();
121     Serial.print("UART2: ");
122     Serial.println(parqueo2);
123   }
```

Se leen los datos del puerto UART y se almacenan en su respectiva variable.

```
125   if (parqueo1 == 'a'){
126     P1 = 'a';
127     if (FlagP1 == 0 && currentDigit < 8){
128       currentDigit++;
129       FlagP1 = 1;
130     }
131   }
132   if (parqueo1 == 'b'){
133     if (FlagP1 == 1 && currentDigit > 0){
134       currentDigit--;
135       FlagP1 = 0;
136     }
137     P1 = 'b';
138   }
```

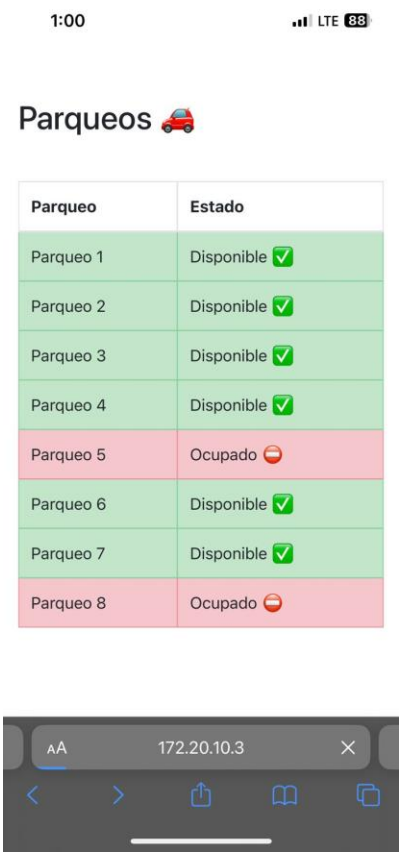
Esta será una comparación que se de para cada uno de los parqueos, así aseguramos que el programa tenga una variable por parqueo. En esta verificamos el carácter que recibe el ESP, y si es igual al que se encuentra comparando en el if, lo guardara en la variable respectiva al parqueo; hay desde P1 hasta P8. También están las comparaciones que sumaran o restaran al contador, junto on la bandera que sube o baja dependiendo de si el parqueo esta ocupado o no, esta nos ayudara a que no se este sumando constantemente cada vez que entre al if.

```
239   displayNumber(currentDigit);
240   server.handleClient();
```

En esta parte del código se entrará a una función la cual coloque en el display de 7 segmentos el valor de la variable currentDigit.

```
290   if (P1 == 'a'){
291   pag += "<tr class=table-success>\n";
292   pag += "<td>Parqueo 1</td>\n";
293   pag += "<td>Disponible &#9989</td>\n";
294   pag += "</tr>\n";
295   }
296
297   if (P1 == 'b'){
298   pag += "<tr class=table-danger>\n";
299   pag += "<td>Parqueo 1</td>\n";
300   pag += "<td>Ocupado &#9940</td>\n";
301   pag += "</tr>\n";
302   }
```

En esta parte del código ya se esta creando el servidor en base al String SendHTML(). Estas comparaciones se realizan para que en la tabla se muestre si se encuentra libre o no el espacio.



Esta sería la página resultante del servidor en la cual se observa que espacios se encuentran libres y cuales ocupados.

Links:

https://youtu.be/oUMaKEWcxOY

https://github.com/Aldoa912/Proyecto_parqueo.git