Universidad del Valle de Guatemala
IE3054 Electrónica Digital 2
Andrés Lemus 21634

Proyecto #3 Tivaware

Introducción

En este último proyecto del curso de Electrónica Digital 2 se tenía como objetivo realizar un parqueo el cual tuviera indicadores visuales y un servidor en la web para poder acceder a él en tiempo real y poder observar el estado del parqueo. El parqueo debía tener 8 lugares disponibles, 4 controlados por un microcontrolador y otros cuatro con otro microcontrolador. En este caso se utilizaron dos kits de evaluación TM4C123G LaunchPad de la empresa de Texas Instruments, que tienen como microcontrolador el TM4C123GH6PM. La información de cada parqueo era enviada a otro microcontrolador que en este caso era un ESP32, que también se utilizo en su forma de DevKit, ya que este servía como servidor para poder realizar la página web. En el ESP32 también se debía tener una pantalla LCD para poder desplegar la información de los parqueos. La información de los parqueos era tomada por sensores de led infrarrojos de proximidad.

Programación y Lógica

En este caso la lógica era bastante sencilla ya que los sensores de led infrarrojos de proximidad trabajan como datos booleanos, es decir, que son o cero o uno. En cada microcontrolador se tenían cuatro sensores y dos leds por sensor. En total ocho leds y cuatro sensores. En este caso para obtener la lectura únicamente se están polleando las salidas de los sensores que son entradas del microcontrolador, para así recopilar los datos. En este caso se hizo una variable booleana por lugar de parqueo, que dependiendo del sensor se haría verdadera si detectaba un objeto el sensor o se haría falsa si no detectaba nada. En total se tenían cuatro variables por lo que para enviar el dato de los cuatro parqueos se realizaron corrimientos de bits para que en un dato de 8 bits cada bit fuera un lugar de parqueo. En este caso: 0000 1110, en este caso los bits del 4 al 7 no se utilizan ya que solo eran cuatros parqueos por microcontrolador, pero los primero cuatro bits significan un lugar de parqueo, esto se logra gracias a los corrimientos. En este caso el lugar 1 (bit 0) está desocupado mientras que el lugar 2 (bit 1), lugar 3 (bit 2) y lugar 4 (bit 3) están ocupados. Este dato de ocho bits se envía por medio de los módulos UART al ESP32 para que este ya sepa que parqueos están ocupados o no. El mismo procedimiento se hizo con los otros cuatro parqueos.

En el ESP32 se tenían activos dos módulos UART para cada uno de los microcontroladores. Para el proyecto se estaban polleando cada uno de los módulos para poder leer los datos de cada uno de los microcontroladores de una manera rápida. En el ESP32 para que ese supiera que parqueos estaban ocupados también se hicieron ocho variables booleanas para cada lugar de parqueo y se hacía un chequeo que básicamente consistía en ver cada uno de los casos, al ser un dato de 4 bits se tenían un total de 16 casos de los parqueos, ya que el dato iba de 0 a 15. En la pantalla también dependía de las ocho variables y solo las imprimía en la pantalla.

El servidor consístia de HTML, CSS y Javascript. En CSS se le dió estilo a los parqueos, en este caso se hicieron dos clases distintas, una para cuando el lugar

estuviera ocupado y otra cuando el lugar estuviera libre.En HTML se crearon divs para cada uno de los lugares. En javascript lo que se hacía era actualizar los lugares de forma automática sin tener que estar refrescando la página por completo. Lo que se hizo fue que en el ESP32 se tenían, aparte de la página root principal (‘/’), otras ocho secciones root, ejemplo (‘/l1), que quiere decir lugar 1. Estos roots chequean el valor de la variable que tenía su valor de los casos recibidos de la tiva e iban a estar enviando texto por medio de javascript. Entonces en HTML se creó un id por lugar de parqueo, en este caso “spot1” (ejemplo), y la función de javascript era llamar a ese identificador, y ejecutar el root de ese lugar (/l1) y dependiendo del valor de la variable el texto que se enviaba era el de “occupied” o “vacant”, entonces en la misma de función de javascript se le asignaba al id creado una de los das clases, o vacant u occupied, donde esas clases de CSS cambiaban el color de fondo del lugar de parqueo.

Funciones Importantes

Chequeo de parqueos en TIVAWARE:

```
if (GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_7) == 0){ //Si el sensor devuelve
un 0 el parqueo está ocupado
            spot1 = 1; //Variable en 1
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_0, 0); //Apagar led verde
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_PIN_1); //Encender
led rojo
        }
        else {
            spot1 = 0; //Variable en 0
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_0, GPIO_PIN_0); //Encender
led verde
            GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_1, 0); //Apagar led rojo
        }
```

Para esta función vemos que lo primero, es preguntar si el pin donde se encuentra el sensor está en 0. El sensor cuando detecta algo envía un 0, si está en 0 pone la variable del parqueo en 1 y luego le indica a cada uno de los leds si se pone encendida o apagada. Si el sensor devuelve un 1 lógico significa que no detectó nada, entonces la variable del lugar se pone en 0 y se encienden los respectivos leds.

Corrimiento de bits y envió por UART:

```
spot2 = spot2 << 1; //Mover bits
        spot3 = spot3 << 2; //Mover bits
        spot4 = spot4 << 3; //Mover bits
        disponibles = spot1 | spot2 | spot3 | spot4; //Realizar un OR para
que cada bit represente un parqueo
        UARTCharPutNonBlocking(UART1_BASE, disponibles); //Enviar por UART1
al esp32
```

Aquí vemos que el lugar 2 hace un corrimiento de un espacio hacia la izquierda, el lugar 3 hace un corrimiento de 2 hacia la izquierda y el lugar 4 hace un corrimiento de

tres hacia la izquierda. Para terminar se hace un OR entre todos los lugares para así convertirlo en un dato de 8 bits y poder enviarlo por Serial, donde cada uno de los primeros cuatro bits significan un lugar de parqueo.

Función para recibir e interpretar datos de la TIVA:

```
//Función para chequear la información de la Tiva1
void check_tiva1(){
  tiva1 = Serial2.read(); //Leer de UART2

  //Chequear para los parqueos
  if (tiva1 == 0){
    parqueo1 = false;
    parqueo2 = false;
    parqueo3 = false;
    parqueo4 = false;
  }
  else if (tiva1 == 1){
    parqueo1 = true;
    parqueo2 = false;
    parqueo3 = false;
    parqueo4 = false;
  }
  else if (tiva1 == 2){
    parqueo1 = false;
    parqueo2 = true;
    parqueo3 = false;
    parqueo4 = false;
  }
  else if (tiva1 == 3){
    parqueo1 = true;
    parqueo2 = true;
    parqueo3 = false;
    parqueo4 = false;
  }
  else if (tiva1 == 4){
    parqueo1 = false;
    parqueo2 = false;
    parqueo3 = true;
    parqueo4 = false;
  }
  else if (tiva1 == 5){
    parqueo1 = true;
    parqueo2 = false;
    parqueo3 = true;
    parqueo4 = false;
  }
  else if (tiva1 == 6){
    parqueo1 = false;
    parqueo2 = true;
    parqueo3 = true;
    parqueo4 = false;
```

```
}
else if (tiva1 == 7){
  parqueo1 = true;
  parqueo2 = true;
  parqueo3 = true;
  parqueo4 = false;
}
else if (tiva1 == 8){
  parqueo1 = false;
  parqueo2 = false;
  parqueo3 = false;
  parqueo4 = true;
}
else if (tiva1 == 9){
  parqueo1 = true;
  parqueo2 = false;
  parqueo3 = false;
  parqueo4 = true;
}
else if (tiva1 == 10){
  parqueo1 = false;
  parqueo2 = true;
  parqueo3 = false;
  parqueo4 = true;
}
else if (tiva1 == 11){
  parqueo1 = true;
  parqueo2 = true;
  parqueo3 = false;
  parqueo4 = true;
}
else if (tiva1 == 12){
  parqueo1 = false;
  parqueo2 = false;
  parqueo3 = true;
  parqueo4 = true;
}
else if (tiva1 == 13){
  parqueo1 = true;
  parqueo2 = false;
  parqueo3 = true;
  parqueo4 = true;
}
else if (tiva1 == 14){
  parqueo1 = false;
  parqueo2 = true;
  parqueo3 = true;
  parqueo4 = true;
}
else if (tiva1 == 15){
  parqueo1 = true;
  parqueo2 = true;
  parqueo3 = true;
```

```
    parqueo4 = true;
  }
}
```

Esta función lo que hace es pollear la lectura del módulo 2 de serial del ESP32 para que este constantemente leyendo el valor de los parqueos enviado por la TIVA. Dependiendo del valor que lea es un caso diferente de los parqueos, como se había el bit 0 representa el lugar y así sucesivamente, por lo que por ejemplo si todos están ocupados, todos los cuatro bits deberían de estar en 1, y al ser de cuatro bits eso significa que es 15, y como se puede ver cuando se recibe un 15 todas las variables de los parqueos están en true.

Actualizar parqueos sin actualizar la página entera:

```
  ptr +="<script>\n";
  ptr +="function updateColor1() {\n";
  ptr +="var xhttp = new XMLHttpRequest();\n";
  ptr +="xhttp.onreadystatechange = function() {\n";
  ptr +="if (this.readyState == 4 && this.status == 200) {\n";
  ptr +="document.getElementById('spot1').className = this.responseText;\n";
  ptr +="}\n";
  ptr +="};\n";
  ptr +="xhttp.open('GET', '/l1', true);\n";
  ptr +="xhttp.send();\n";
  ptr +="setTimeout(updateColor1, 500);}\n";
```

En esta función lo que se hace es que primer se crea una función que se estará ejecutando constantemente. Primero se crea una variable para cuando se ejecute una request de tipo XML. Luego se crea una función para manejar la request realizada, donde lo primero es chequear que el si el ready state es 4 y el status es 200 se proceda. Esto indica que la request fue completada (status == 4) y exitosa (status == 200). Lo siguiente es actualizar el HTML con el id ‘spot1’ y asignarle la clase que devuelva el texto basado en la respuesta de la request. Por último, asíncronamente se está haciendo una request a la dirección /l1, y enviando la respuesta de está dirección. En la dirección l1 se envía el texto para asignarle la clase al identificador específico.

Direcciones para enviar respuesta a la request:

```
void handleColor1(){
  if (parqueo1 == false) {
    server.send(200, "text/plain", "parking-spot vacant"); //Si el parqueo
esta vacío enviar un texto a javascript con la clase de css en HTML para
asignarle al id correspondiente
  } else {
    server.send(200, "text/plain", "parking-spot occupied"); //Si el parqueo
esta ocupado enviar un texto a javascript con la clase de css en HTML para
asignarle al id correspondiente
  }
}
server.on("/l1", handleColor1);
```

Como vemos chequea primero la variable del parqueo para si está ocupado o no. Esta función se ejecuta cuando se recibe el request a la dirección /l1. Si está ocupado el lugar envía al servidor un texto que dice “parking-spot ocuppied” esta es la clase de CSS que se le asgina al HTML para representar que está ocupado, básicamente es poner el fondo en rojo. Cuando se encuentra libre se envía “parking-spot vacant”. Este es el texto que recibe la función de javascript donde se busca el id del HTML y se le asigna la clase de CSS a ese identificador de HTML.