Maestría en Sistemas Embebidos

Presentación del Trabajo Final

.UBAfiuba
FACULTAD DE INGENIERÍA

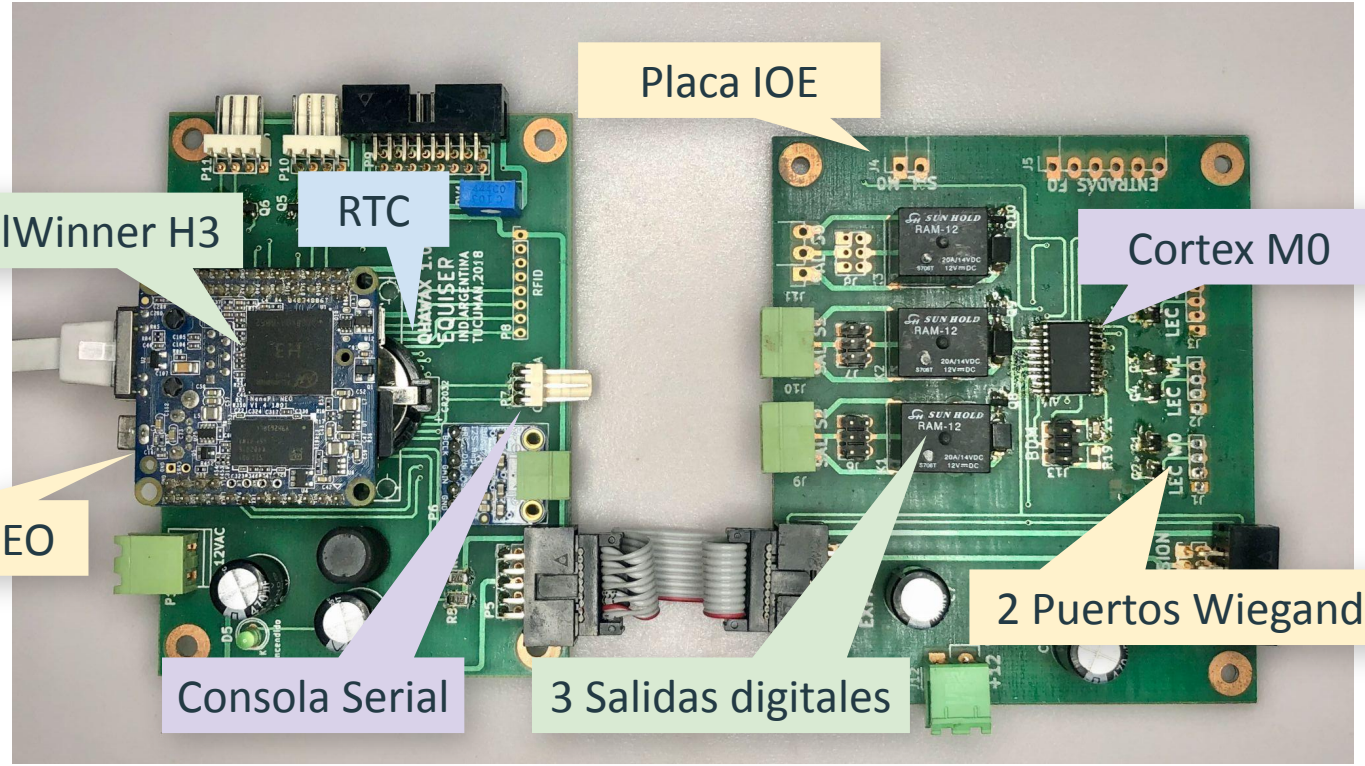
Controlador de dispositivo para placa de expansión con salidas digitales

UNIVERSIDAD NACIONAL DE TUCUMÁN

Universidad Nacional de Tucumán
facet
Facultad de Ciencias Exactas y Tecnología

Autor: Esp.Ing.Esteban Volentini

Placa para control de accesos



Tareas realizadas en el trabajo final

- Se agregó el MCP7940x en el DTS
- Se agregó un servicio que actualiza la hora del sistema a partir de RTC
- Se agregó la placa IOE en el DTS
- Se agregó un dispositivo separado para cada salida y cada lectora
- Se habilitó el módulo nativo de kernel que da soporte al MCP7940x
- Se agregó un módulo propio de kernel para dar soporte a la placa
- Se definieron las operaciones de read y write para las salidas digital y solo la operación de read para las lectoras

Cambios en el archivo DTS

```
&i2c0 {
    status = "okay";
    clock-frequency = <100000>;

    rtc_ext: rtc_ext@68 {
        compatible = "microchip,mcp7940x";
        reg = <0x68>;
    };

    rqwx_ioe: qwx_ioe@50 {
        compatible = "equiser,qwxioe";
        reg = <0x50>;
    };
};
```

Se agregó el MCP7940x

Se agregó la placa IOE

Estructura privada en el controlador

```c
//! Estructura con la informacion del dispositivo correspondiente a la placa de expansion
struct expansion_dev {
    struct i2c_client *client;
    struct miscdevice outputs[OUTPUTS_COUNT];
    struct miscdevice readers[READERS_COUNT];
    char name[I2C_NAME_SIZE];
    int device;
};

//! Estructura con la implementacion las operaciones de archivos en salidas digitales
static const struct file_operations outputs_fops = {
    .owner = THIS_MODULE,
    .read = output_read,
    .write = output_write,
};

//! Estructura con la implementacion las operaciones de archivos en lectoras de rfid
static const struct file_operations readers_fops = {
    .owner = THIS_MODULE,
    .read = reader_read,
};
```

Se definirá un dispositivo para salida digital

Se definirá un dispositivo para lectora

Se definen las operaciones de read y write para las salidas digitales

Se define la operación de read para las lectoras

Código de la función probe

```
static int probe(struct i2c_client *client, const struct i2c_device_id *id)  {
    struct expansion_dev *device;
    int error, output, reader, index;

    if (client->addr == 0x50) {
        device = devm_kzalloc(&client->dev, sizeof(struct expansion_dev), GFP_KERNEL);
        snprintf(device->name, I2C_NAME_SIZE, "/exp0");
    } else {
        pr_err("No se reconoce la direccion del dispositivo");
        return error;
    }
    device->client = client;
    i2c_set_clientdata(client, device);

    for(output = 0; output < OUTPUTS_COUNT; output++) {
        error = add_output(device, output);
        if (error != 0) {
            pr_err("No se pudo registrar el dispositivo %s/s%d", device->name, output);
            for(index = 0; index < output; index++) {
                misc_deregister(&device->outputs[index]);
            }
            return error;
        }
    }
```

Se valida la dirección

Se asigna memoria

Se asigna el nombre

Se agrega cada puerto como un dispositivo independiente

Se revierten los cambios en caso de un error de registro

Código para agregar un puerto

```
int add_output(struct expansion_dev *device, unsigned short int output_number) {
    struct miscdevice *output = &device->outputs[output_number];
    char *name;

    name = devm_kzalloc(&device->client->dev, I2C_NAME_SIZE, GFP_KERNEL);
    snprintf(name, I2C_NAME_SIZE, "%s/s%d", device->name, output_number);
    output->name = name;
    output->minor = MISC_DYNAMIC_MINOR;
    output->fops = &outputs_fops;

    return misc_register(output);
}
```

Se selecciona el descriptor

Se asigna memoria para el nombre

Se define el nombre como un archivo dentro de una carpeta

Se registra como dispositivo

Se asignan las operaciones

Código para la función de lectura

```
static ssize_t output_read(struct file *file, char __user *buffer, size_t count, loff
    unsigned short int output = file->f_path.dentry->d_name.name[1] - '0';
    struct expansion_dev * device = container_of(file->private_data, struct expansion_dev, outputs[output]);
    char data[3] = "0\n";
    char address, respo

    if (*f_pos == 0) {
        address = 0x70 + output;
        i2c_master_send(device->client, &address, sizeof(address));

        i2c_master_recv(device->client, &response, sizeof(response));
        data[0] += response;

        count = sizeof(data);
        if (copy_to_user(buffer, data, count)) {
            return -EFAULT;
        }

        *f_pos += count;
        return count;
    }
    return 0;
}
```

Se obtiene el puerto a partir del nombre del archivo

Se obtiene el puntero al dispositivo

Se envía el comando para seleccionar el registro a leer

Se lee el valor del registro que informa el estado de una salida

Se actualiza la respuesta

Se copia la respuesta al espacio de memoria del usuario

Se actualiza la posición del archivo

Código para la función de escritura

```
static ssize_t output_write(struct file *file, const char __user *buffer, size_t len
    unsigned short int output = file->f_path.dentry->d_name.name[1] - '0';
    struct expansion_dev * device = container_of(file->private_data, struct expansion_dev, outputs[output]);
    char data[2];
    char response;

    if (len == 0) {
        return 0;
    }
    if (copy_from_user(&response, buffer, 1)) {
        return -EFAULT;
    }

    if (response == '1') {
        data[0] = 0x71;
    } else {
        data[0] = 0x70;
    }

    data[1] = output;
    i2c_master_send(device->client, data, 2);

    return len;
}
```

Se obtiene el puerto a partir del nombre del archivo

Se obtiene el puntero al dispositivo

Se copian los datos al espacio de memoria del kernel

Se prepara la dirección del registro a escribir según el estado

Se envia el numero de salida como dato a escribir en el registro

Se confirma que se escribieron todos los datos del usuario

El sistema completo funcionando

Demostración

Muchas gracias por su atención

¿Preguntas?