Tecnológico Nacional de México
Instituto Tecnológico de Chihuahua

ARQUITECTURA DE PROGRAMACION PARA HARDWARE

Unidad 1 Laboratorio 3.

Alumno: Mario Alejandro Briones Lara (20060767)
Docente: Chacón Aldama Alfredo **Fecha: 13/03/2023**

Objetivo:

1. Defina y ejemplifique el concepto herencia. (1 punto)
2. Defina y ejemplifique el concepto polimorfismo. (1 punto)
3. Realice un programa con el paradigma orientado a objetos (6 puntos).

Genere un programa que simule la configuración de un ADC

3.a Cree un proyecto en Dev C.

Dicho programa debe pedir lo siguiente:

Resolución (8, 10 o 12 bits).

Frecuencia de muestreo.

Cantidad de canales a leer.

Definir cuales canales se va a leer (los canales son AN1, …, AN32).

La configuración se hace solo con una lectura para cada canal.

Una vez configurado el canal ADC, se puede realizar lo siguiente:

- Hacer una lectura de un canal del ADC. Este punto se hace colocando un dato en consola, el cual se introduce con valores de 0 a 3.3 Volts. Una vez que se introduce dicho valor, se genera el dato que se convirtió con base en la resolución.
- Imprimir el valor de la lectura del ADC.
- Si se configuró hacer la lectura con más de un canal, se hace la lectura de los canales activados en la configuración.
- Imprimir el valor de las lecturas de los canales del ADC activos.

3.b Cree un repositorio en GIT para el control de versiones de su proyecto y genere al menos un commit (versión inicial).

4. Cambie el programa para implementar esta parte con polimorfismo y con herencia. (3 puntos)

nota: Deberá añadir un menú al principio del programa para seleccionar esta opción o la del punto3.

Suponga ahora que la configuración de la frecuencia de muestreo se realiza de la siguiente manera:

Fs = f_osc / ack.

Donde fosc es de 8 MHz y ack es un factor que tiene valores de 2, 4, 8, 16, 32 y 64.

Conceptos:

Herencia: En la programación orientada a objetos (POO), la herencia es un mecanismo que permite que una clase adquiera las propiedades y métodos de otra clase. En otras palabras, una clase puede heredar atributos y comportamientos de una clase padre, y además puede agregar nuevos atributos y comportamientos propios.

Ejemplo:

Volvamos a los animales, pensemos en los mamíferos. Todos tienen una serie de características, como meses de gestación en la barriga de la madre, pechos en las hembras para amamantar y luego funcionalidades como dar a luz, mamar, etc. Eso quiere decir que cuando realices la clase perro vas a tener que implementar esos atributos y métodos, igual que la clase vaca, cerdo, humano, etc.

¿Te parecería bien reescribir todo ese código común en todos los tipos de mamíferos, o prefieres heredarlo? en este esquema tendríamos una clase mamífero que nos define atributos como numero_mamas, meses_gestacion y métodos como dar_a_luz(), mamar(). Luego tendrías la clase perro que extiende (hereda) el código del mamífero, así como las vacas, que también heredan de mamífero y cualquiera de los otros animales de esta clasificación.

Otro ejemplo, tenemos alumnos universitarios. Algunos son alumnos normales, otros Erasmus y otros becarios. Probablemente tendremos una clase Alumno con una serie de métodos como asistir_a_clase(), hacer_examen() etc., que son comunes a todos los alumnos, pero hay operaciones que son diferentes en cada tipo de alumno como pagar_mensualidad() (los becarios no pagan) o matricularse() (los Erasmus que son estudiantes de intercambio, se matriculan en su universidad de origen).

Lo que debes observar es que con la herencia siempre consigues clases hijas que son una especialización de la clase padre. Para saber si está correcto emplear herencia entre unas clases y otras, plantéate la pregunta ¿CLASE HIJA es un CLASE PADRE? (por ejemplo, ¿un perro es un mamífero? ¿Un becario es un alumno de universidad?)

Polimorfismo: Se refiere a la capacidad de los objetos de una clase para tomar diferentes formas o comportarse de diferentes maneras. En otras palabras, el polimorfismo permite que un objeto pueda ser tratado como si fuera de varios tipos diferentes. Hay dos tipos principales de polimorfismo: el polimorfismo estático y el polimorfismo dinámico. El polimorfismo estático se refiere al uso de sobrecarga de funciones y plantillas, mientras que el polimorfismo dinámico se logra a través de la herencia y las funciones virtuales.
Ejemplo:
Un ejemplo clásico de polimorfismo es el siguiente. Podemos crear dos clases distintas: Gato y Perro, que heredan de la superclase Animal. La clase Animal tiene el método abstracto makesound() que se implementa de forma distinta en cada una de las subclases (gatos y perros suenan de forma distinta). Entonces, un tercer objeto puede enviar el mensaje de hacer sonido a un grupo de objetos Gato y Perro por medio de una variable de referencia de clase Animal, haciendo así un uso polimórfico de dichos objetos respecto del mensaje mover.

```
class Animal {
  public void makeSound() {
    System.out.println("Grr...");
  }
}
class Cat extends Animal {
  public void makeSound() {
    System.out.println("Meow");
  }
}
class Dog extends Animal {
  public void makeSound() {
    System.out.println("Woof");
  }
}
```

Como todos los objetos Gato y Perro son objetos Animales, podemos hacer lo siguiente

```
public static void main(String[ ] args) {
  Animal a = new Dog();
  Animal b = new Cat();
}
```

```
a.makeSound();
//Outputs "Woof"

b.makeSound();
//Outputs "Meow"
```

Metodología:

Se desarrolló una clase ADC en donde se encuentran los atributos y métodos necesarios para el funcionamiento del programa, estos se determinaron con las especificaciones que pedía el objetivo. Resolución, frecuencia de muestreo, cantidad de canales a leer y los canales del ADC. Los métodos son setup para definir la configuración del ADC, setData para que el usuario ingrese los voltajes y getData para mostrar al usuario los valores del ADC.

Materiales:

- Dev C++
- Laptop
- Github

Desarrollo:

Punto 3: Lo primero que se hizo fue incluir las librerías para el funcionamiento del programa y la creación de la clase “ADC” con los atributos para resolución, frecuencia, cantidad de canales a leer y los canales que va a leer (Puestos en un arreglo de 32).

```
/*****************************************************************************************************************/
#include <stdlib.h>
#include <iostream>
#include <math.h>
using namespace std;
//*****************************************************************************************************************
class ADC{
   int _res;          //Resolución (8, 10 o 12 bits)
   int _freq;         //Frecuencia de muestreo
   int _num_can;      //Cantidad de canales a leer
   int _can_an[32];   //Definir cuales canales se va a leer (los canales son AN1, …, AN32)
public:
   ADC(int _res,int _freq,int _num_can,int _can_an[]);
   void setup();
   int setData(int canal);
   void getData();
};
//*****************************************************************************************************************
```

Los métodos que se utilizan en la clase son:

setup(): aquí encontramos la forma en que se va a configurar el ADC conforme las exigencias del usuario. Véase en la imagen que es aquí donde entran en juego los atributos declarados en la clase, como algunas formas de validar entradas con ciclos.

```
//*************************************************************************************************************************
void ADC::setup(){
    cout << "Configuracion del ADC:" << endl;
    cout << "Resolucion (8, 10, or 12 bits): ";
    cin >> _res;
    while(_res !=8 && _res !=10 && _res !=12 ){
            cout << "Resolucion (8, 10, or 12 bits): ";
            cin >> _res;
        }
    cout << "Frecuencia de muestreo: ";
    cin >> _freq;
    cout << "Cantidad de canales a leer(AN1-AN32): ";
    cin >> _num_can;
    while(_num_can <1 || _num_can>32){
        cout << "Cantidad de canales a leer(AN1-AN32): ";
        cin >> _num_can;
        }
    cout << "Definir cuales canales se va a leer (AN1-AN32): ";
    for (int i = 0; i < _num_can; i++) {
        cin >> _can_an[i];
    }
}
//*************************************************************************************************************************
```

setData(): con este método el usuario ingresa el valor de voltaje que hace en su lectura y se aplica una fórmula para la conversión

```
//*************************************************************************************************************************
int ADC::setData(int canal){
    cout << "Ingrese el voltaje del canal " << canal << " (0-3.3V): ";
    double volt;
    cin >> volt;
    int adcout = volt / (3.3 / pow(2, _res));
    return adcout;
}
//*************************************************************************************************************************
```

getData(): el ultimo método implementado arroja el canal que selecciono el usuario y el valor convertido

```
//*************************************************************************************************************************
void ADC::getData(){
    cout << "Lecturas ADC:" << endl;
    for (int i = 0; i < _num_can; i++) {
        int canal = _can_an[i];
        int adcout = setData(canal);
        cout << "Canal " << canal << ": " << adcout << endl;
    }
}
//*************************************************************************************************************************
```

El constructor de objetos es el siguiente, el ciclo for es para la construcción de los 32 canales

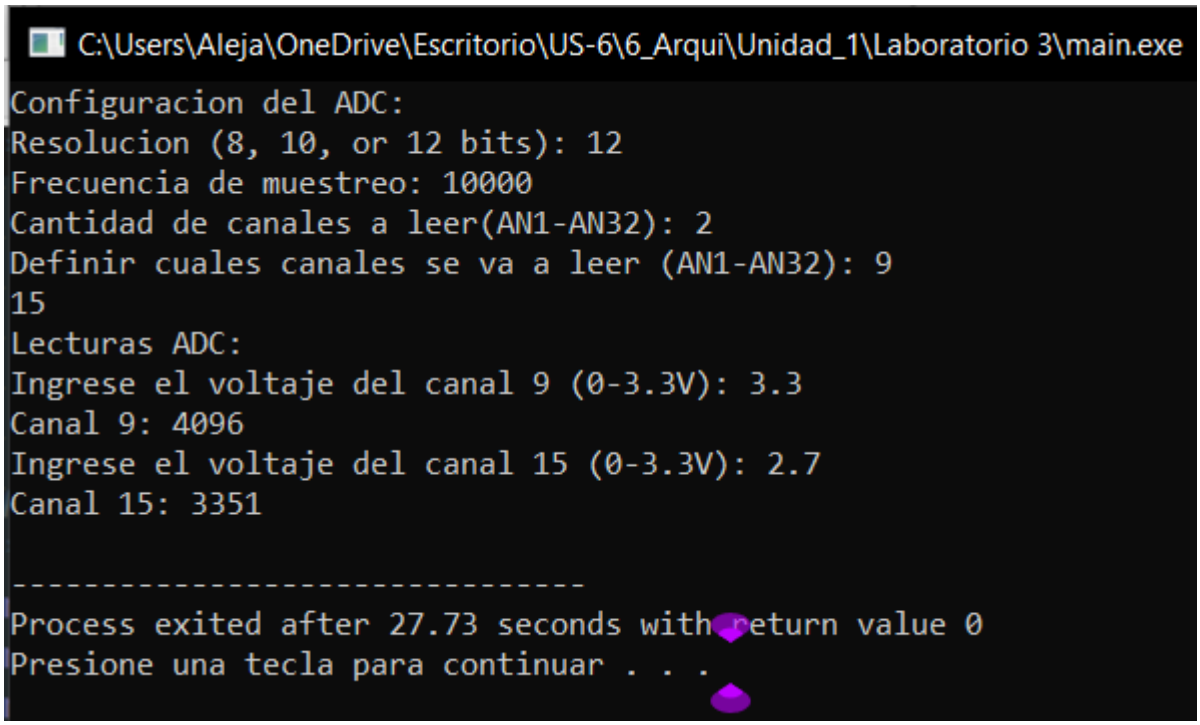
```
//*************************************************************************************************************************
ADC::ADC(int res,int freq,int num_can,int can_an[]){
    _res = res;
    _freq = freq;
    _num_can = num_can;
    for(int i = 0; i < num_can ;i++){
        _can_an[i] = can_an[i];
    }
}
//*************************************************************************************************************************
```

Por último, el main

```
//*************************************************************************************************************************
int main() {
    int canales[]={0};
    ADC adc(0, 0, 0, canales);
    adc.setup();
    adc.getData();
    return 0;
}
```

Resultados:

Punto 3:



Punto 4:

Conclusiones:

Para el laboratorio se siguen usando las herramientas anteriores y las nuevas, permitiendo realizar trabajos más limpios en cuanto a código, la programación orientada a objetos nos permite manejar módulos haciendo más fácil el entendimiento del código, veo una gran utilidad en programas de mayor tamaño donde el manejo de datos aumenta considerablemente, por otro lado, no quita la complejidad que fue entender el funcionamiento de un ADC.

Referencias:

Libro de FOO