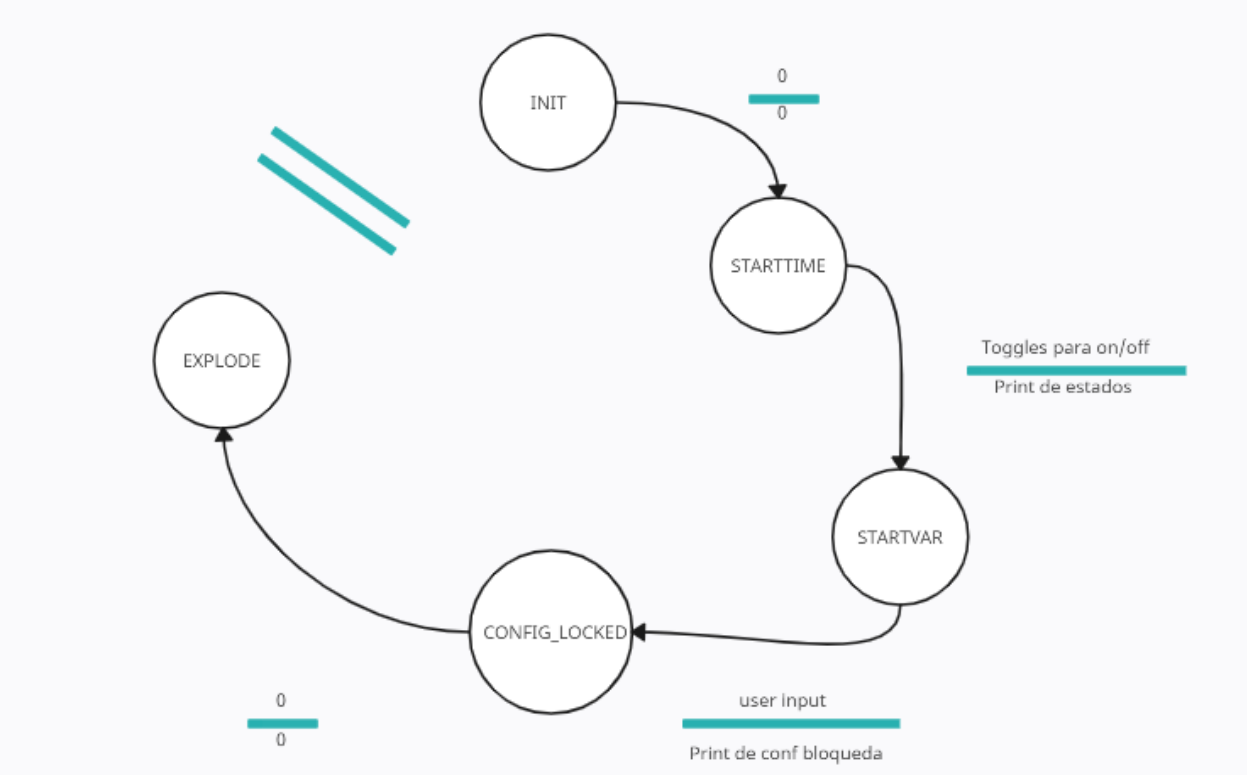
DOCUMENTACIÓN PROYECTO BASE NUCLEAR

Esteban Cardoso, Mateo Jiménez y Mariana Echavarria

Introducción

El objetivo de este proyecto fue desarrollar una versión más elaborada de la experiencia de la Central Nuclear que hicimos al final de la Unidad 1. En esta ocasión realizamos un sistema de aplicaciones interactivas que se comunican entre ellas, una aplicación corriendo en el PC y la otra en el microcontrolador.

Diagrama de Estados:



Para programar el pico:

Primero declaramos las variables globales para controlar el estado de las variables **Temperature**, **Pressure** y **Dynamic**, así como sus valores asociados y un temporizador **Timer**.

```
bool Temperature = false;
bool Pressure = false;
bool Dynamic = false;
int TemperatureValue = 10;
int PressureValue = 10;
int DynamicValue = 10;
int Timer = 60;
```

Después escribimos la función para limpiar el serial buffer (esto se implementa para descartar cualquier dato que esté disponible para ser leído).

```
 void CleanSerialBuffer()
{
  while (Serial.available() > 0)
  {
    char _ = Serial.read();  // Read and discard data until the buffer is empty
  }
}
```

Implementamos máquina de estados y definimos estados posibles:

```
 void task1() {
  enum class Task1States {
    INIT,
    STARTTIME,
    STARTVAR,
    CONFIG_LOCKED,
    INTERVALS,
    EXPLODE  };
```

Declaramos una variable estática **task1State** para mantener el estado actual de la máquina que es **INIT** (para inicializar).

```
static Task1States task1State = Task1States::INIT;
```

Colocamos un switch para navegar entre los diferentes estados.

En case **INIT** inicializamos la comunicación serial y mostramos las instrucciones con un mensaje previo de bienvenida. Luego se cambia al estado **STARTTIME**.

```
  case Task1States::INIT: {
    Serial.begin(115200);
// (Se imprimen mensajes de bienvenida e instrucciones)
    Serial.println("Welcome to the nuclear control center!");
    Serial.println("Press 1 to toggle temperature.");
    Serial.println("Press 2 to toggle pressure.");
    Serial.println("Press 3 to toggle dynamic.");
    Serial.println("Press 4 to print current state of all variables.");
    task1State = Task1States::STARTTIME;
    break;
  }
```

En **STARTTIME**, se espera la entrada del usuario y se realiza lo que este ingrese según la opción de su elección. Las acciones incluyen cambiar el estado de las variables y mostrar el estado actual.

```
case Task1States::STARTTIME: {
    if (Serial.available() > 0) {
      int userInput = Serial.read() - '0'; // Convert char to integer
      switch (userInput) {
        case 1:
          Temperature = !Temperature;
          Serial.print("Temperature state: ");
          Serial.println(Temperature ? "On" : "Off");
          break;
        case 2:
```

```
      Pressure = !Pressure;
      Serial.print("Pressure state: ");
      Serial.println(Pressure ? "On" : "Off");
      break;
    case 3:
      Dynamic = !Dynamic;
      Serial.print("Dynamic state: ");
      Serial.println(Dynamic ? "On" : "Off");
      break;
    case 4:
      Serial.println("Current state of all variables:");
      Serial.print("Temperature: ");
      Serial.println(Temperature ? "On" : "Off");
      Serial.print("Pressure: ");
      Serial.println(Pressure ? "On" : "Off");
      Serial.print("Dynamic: ");
      Serial.println(Dynamic ? "On" : "Off");
      Serial.print("Temperature Value: ");
      Serial.println(TemperatureValue);
      Serial.print("Pressure Value: ");
      Serial.println(PressureValue);
      Serial.print("Dynamic Value: ");
      Serial.println(DynamicValue);
      Serial.print("Timer: ");
      Serial.println(Timer);
      break;
    case 5:
      Serial.println("Switching to STARTVAR state. Press 'a', 'b', or 'c' to increment corresponding value.");
      task1State = Task1States::STARTVAR;
      break;
```

```
      default:
        break;
    }
  }
  break;
}
```

En **STARTVAR**, le permitimos al usuario incrementar los valores de las variables **TemperatureValue**, **PressureValue**, **DynamicValue** y **Timer**. (En el caso de que el usuario ingrese una opción incorrecta se le notifica que ha cometido un error y que puede ingresar nuevamente algun input para modificar las variables mostradas).

```
case Task1States::STARTVAR: {
  if (Serial.available() > 0) {
    int userInput = Serial.read();
    switch (userInput) {
      case 'a':
        TemperatureValue += 10;
        Serial.print("Temperature Value incremented to: ");
        Serial.println(TemperatureValue);
        break;
      case 'b':
        PressureValue += 10;
        Serial.print("Pressure Value incremented to: ");
        Serial.println(PressureValue);
        break;
      case 'c':
        DynamicValue += 10;
        Serial.print("Dynamic Value incremented to: ");
        Serial.println(DynamicValue);
        break;
      case 'z':
```

```
        if (Timer < 180) {
          Timer += 10;
          Serial.print("Timer incremented to: ");
          Serial.println(Timer);
        } else {
          Serial.println("Maximum timer reached (180 seconds).");
        }
        break;
      case 'j':
        Serial.println("Configuration locked. Moving to next state.");
        task1State = Task1States::CONFIG_LOCKED;
        break;
      default:
        Serial.println("Invalid input. Please try again.");
        break;
    }
  }
  break;
}
```

En **CONFIG_LOCKED**, se cambia al estado **EXPLODE**.

```
case Task1States::CONFIG_LOCKED: {
  task1State = Task1States::EXPLODE;
  Serial.println("Moving to EXPLODE state.");
  break;
}
```

En **EXPLODE**, mostramos el estado actual de todas las variables y colocamos un temporizador antes de "explotar".

```
case Task1States::EXPLODE: {
```

```
        Serial.println("Running configuration:");
        Serial.print("Temperature: ");
        Serial.println(Temperature ? "On" : "Off");
        Serial.print("Pressure: ");
        Serial.println(Pressure ? "On" : "Off");
        Serial.print("Dynamic: ");
        Serial.println(Dynamic ? "On" : "Off");
        Serial.print("Temperature Value: ");
        Serial.println(TemperatureValue);
        Serial.print("Pressure Value: ");
        Serial.println(PressureValue);
        Serial.print("Dynamic Value: ");
        Serial.println(DynamicValue);
        Serial.print("Timer: ");
        Serial.println(Timer);
while (Timer > 0) {
        delay(1000);
        Timer--;
        Serial.print("Timer: ");
        Serial.println(Timer);
      }



      Serial.println("Explosion");
      task1State = Task1States::INTERVALS; // Change this to the appropriate next state
      break;
    }
  }
}
```

Para finalizar en **setup** llamamos una vez a task1 para inicializar y en **loop** lo hacemos infinitamente para que el programa continue en bucle.

```
void setup() {
  task1();
}
void loop() {
  task1(); }
```

Entrada de Usuario	Descripción y Uso
1	Toggle para cambiar el estado de la temperatura (ON/OFF).
2	Toggle para cambiar el estado de la presión del refrigerante (ON/OFF).
3	Toggle para cambiar el estado del nivel de agua (ON/OFF).
4	Mostrar el estado actual de todas las variables (temperatura, presión, nivel de agua, valor inicial y tiempo).
5	Cambiar al estado **STARTVAR** para permitir incrementar los valores de las variables.
a	Incrementar el valor de la temperatura en 10 unidades.
b	Incrementar el valor de la presión del refrigerante en 10 unidades.
c	Incrementar el valor del nivel de agua en 10 unidades.
z	Incrementar el temporizador en 10 segundos, si no se ha alcanzado el máximo (180 segundos).
j	Cambiar al estado **CONFIG_LOCKED** y luego al estado **EXPLODE**.

Para la interfaz gráfica:

Descripción General

En Unity nuestro script se encarga de establecer la comunicación serial (a través del puerto serial COM6) con el pico y enviar comandos basados en las teclas presionadas por el usuario; también mostrar la respuesta del dispositivo en la interfaz gráfica de usuario utilizando **TextMeshProUGUI**.

Componentes Principales

Enumerador TaskState:

Define los estados posibles para la máquina de estados.

Estados: INIT y CHECK_INPUTS.

Variables Privadas:

taskState: Almacena el estado actual de la tarea.

_serialPort: Objeto SerialPort para la comunicación serial.

buffer: Arreglo de bytes para almacenar datos recibidos del dispositivo.

Variables Públicas:

TextMeshProUGUI: Variables públicas para mostrar información en la interfaz de usuario.

Métodos

Start():

Inicializa el objeto SerialPort con la configuración necesaria.

Abre el puerto serial.

Inicializa el arreglo buffer.

CleanBuffer():

Limpia el arreglo buffer.

Update():

Método principal que se ejecuta en cada frame de Unity.

Maneja la máquina de estados para determinar las acciones a realizar según el estado INIT` o CHECK_INPUTS.

Funcionalidades Principales

1. Estado INIT:

- Cambia el estado a CHECK_INPUTS y muestra un mensaje de espera.

2. Estado CHECK_INPUTS:

- Detecta las teclas presionadas (KeyCode) para enviar comandos al dispositivo mediante el puerto serial.
- Lee y muestra la respuesta del dispositivo en los objetos TextMeshProUGUI correspondientes.
- Limpia el buffer después de leer datos para evitar acumulación.

Uso

1. Configuración:

- Asignar los objetos TextMeshProUGUI en el Editor de Unity para mostrar la información.

2. Interacción:

- Presionar las teclas asignadas (a, b, c, z, j) para enviar comandos al dispositivo.
- Observar la respuesta del dispositivo en los objetos TextMeshProUGUI correspondientes en la interfaz de usuario.

Ejemplo de Uso

1. Iniciar el script.

2. Presionar la tecla 1 para enviar el comando 1 al dispositivo y mostrar la respuesta en VariableToggleA.

3. Presionar otras teclas asignadas para enviar diferentes comandos y observar las respuestas correspondientes.