.UBAfiuba
FACULTAD DE INGENIERÍA

Sistema de Control de Cinta Transportadora de Paquetes

Facultad de Ingeniería, Universidad de Buenos Aires

Taller de Sistemas Embebidos,
1er Cuatrimestre 2025

Ignacio Botbol - Sergio Nahuel Chaparro -
Josefina Maria Duvidovich - Santiago Fontela

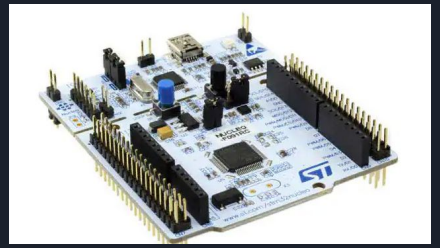
Planteamiento Inicial

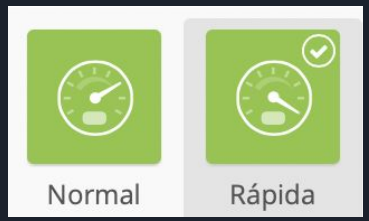
Introducción al Problema

Al comenzar el trabajo práctico, se pensó en el funcionamiento de una cinta transportadora de paquetes dentro de una industria como la que se presenta a continuación.

Objetivo del Proyecto

Implementar un sistema embebido con microcontrolador STM32 F103RB para controlar una cinta transportadora, con operación en distintos modos y velocidades, gestión de fallas y una interfaz de usuario simple y robusta.





Funcionamiento General

Modos de Funcionamiento

→ **Modo Normal (Automático):** Monitorea y controla el ingreso y egreso de paquetes de la cinta transportadora

- Sistema de Control: Supervisa el estado de la cinta y ajusta su funcionamiento según los sensores.
- Modo Seguro: Detiene la cinta ante ausencia de paquetes

→ **Modo Set-Up (Manual):** Configura parámetros de operación del sistema.

- Cambiar la velocidad de la cinta: Nivel 0 (Apagado), Nivel 1 (Normal) y Nivel 2 (Rápido).
- Tiempo de espera para la detención automática.

→ **Modo Falla:** simula una condición de error en el sistema, activando alertas visuales y deteniendo el funcionamiento de la cinta para garantizar la seguridad.

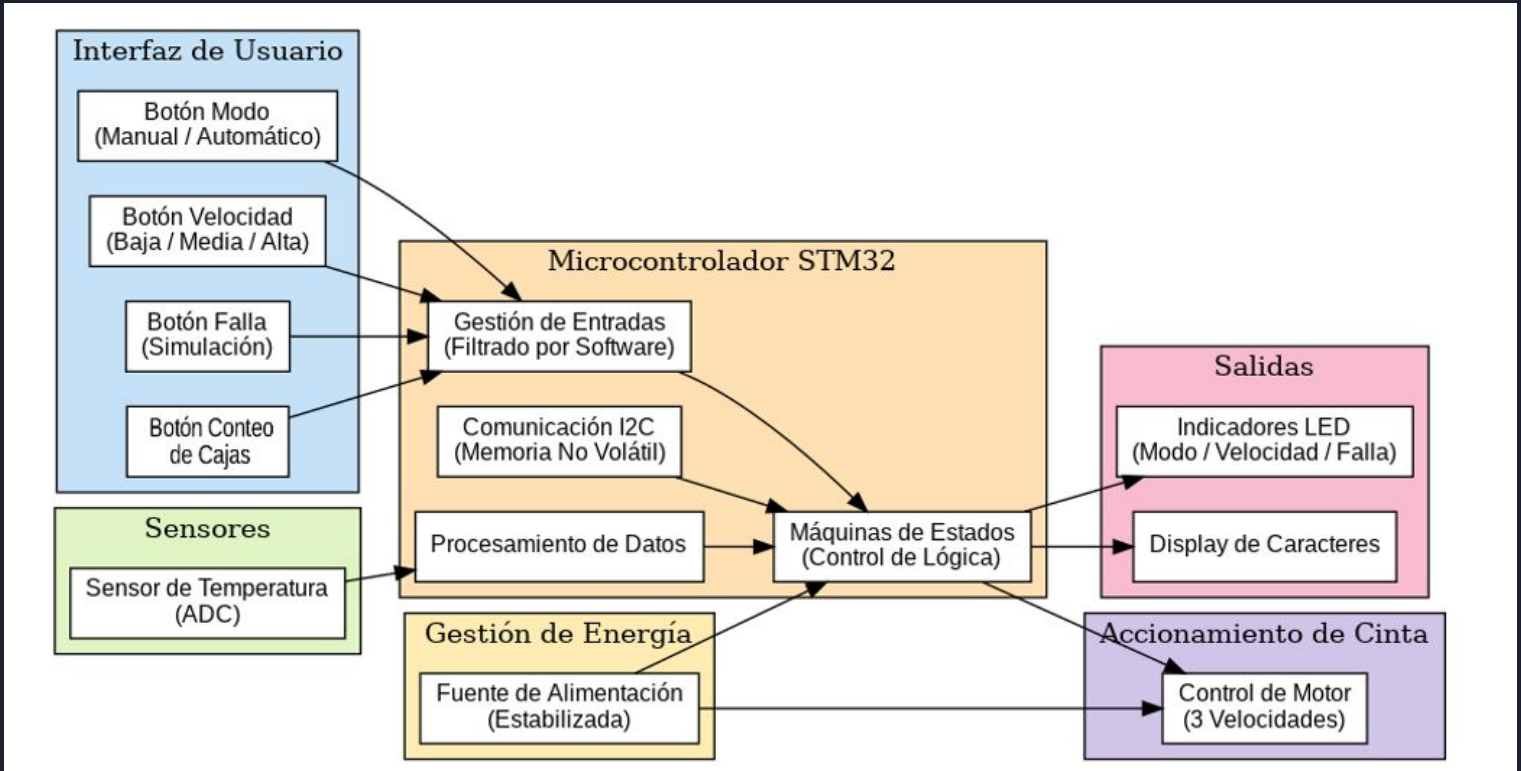
Modos de Funcionamiento

El sistema embebido controla una cinta transportadora con modos Normal y Set-Up, ajuste de velocidad, gestión de fallas, almacenamiento persistente y monitoreo de temperatura.

Se presenta a continuación el diagrama de bloques, elaborado siguiendo la metodología de descripción de sistemas planteada en la Sección 12.1 del libro A Beginner’s Guide to Designing Embedded System Applications on Arm Cortex-M Microcontrollers.

En este diagrama se observan los módulos principales: interfaz de usuario con botones dedicados, sensor de temperatura (ADC), microcontrolador STM32 F103RB con gestión de entradas y filtrado por software, comunicación I2C con memoria no volátil EEPROM, procesamiento de datos y control mediante máquinas de estados. También se incluyen la gestión de energía mediante fuente estabilizada, las salidas visuales (indicadores LED y display de caracteres) y el módulo de accionamiento del motor con control de tres velocidades.

Diagrama de Bloques del Sistema



Implementación

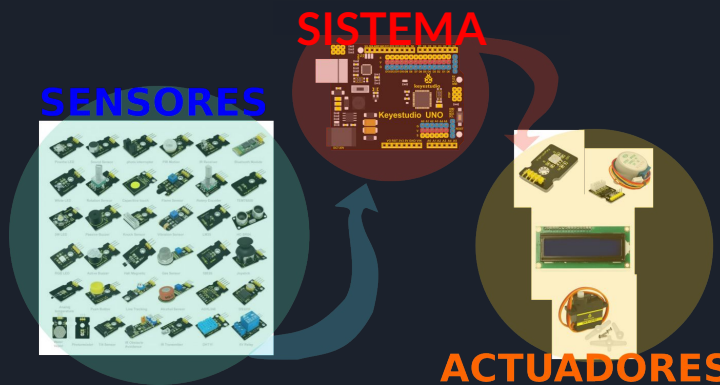
Estructura Interna del Sistema

El sistema diseñado se estructura en tres componentes principales: sensores, sistema y actuadores.

→ **Sensores:** Captan los eventos y los envían al sistema para su procesamiento.

→ **Sistema:** Recibe y procesa los eventos provenientes de los sensores, determinando las acciones a ejecutar.

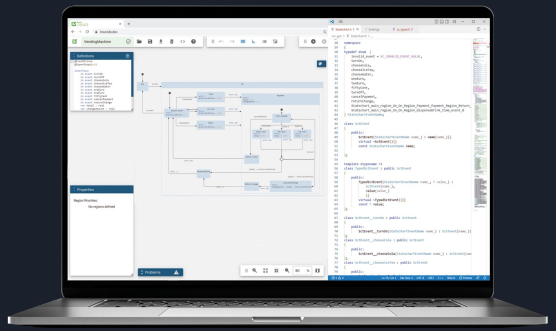
→ **Actuadores:** Ejecutan las acciones definidas por el sistema en respuesta a los eventos captados.



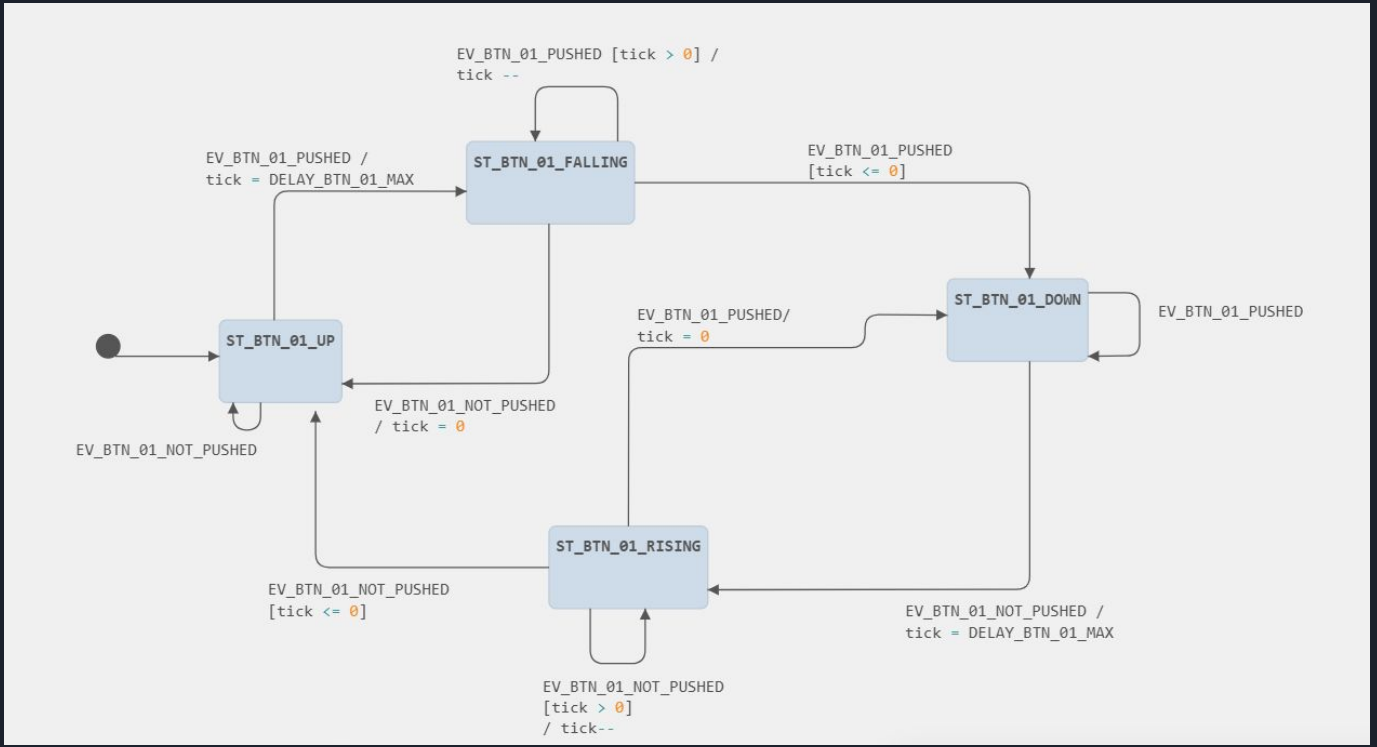
Diagramas de Estados

A partir de la herramiento *Itemis Create*, se diagramaron los Statecharts del sistema. Así, se abstrajo la lógica correspondiente al sistema que se desea modelar, para luego utilizarla en el código.

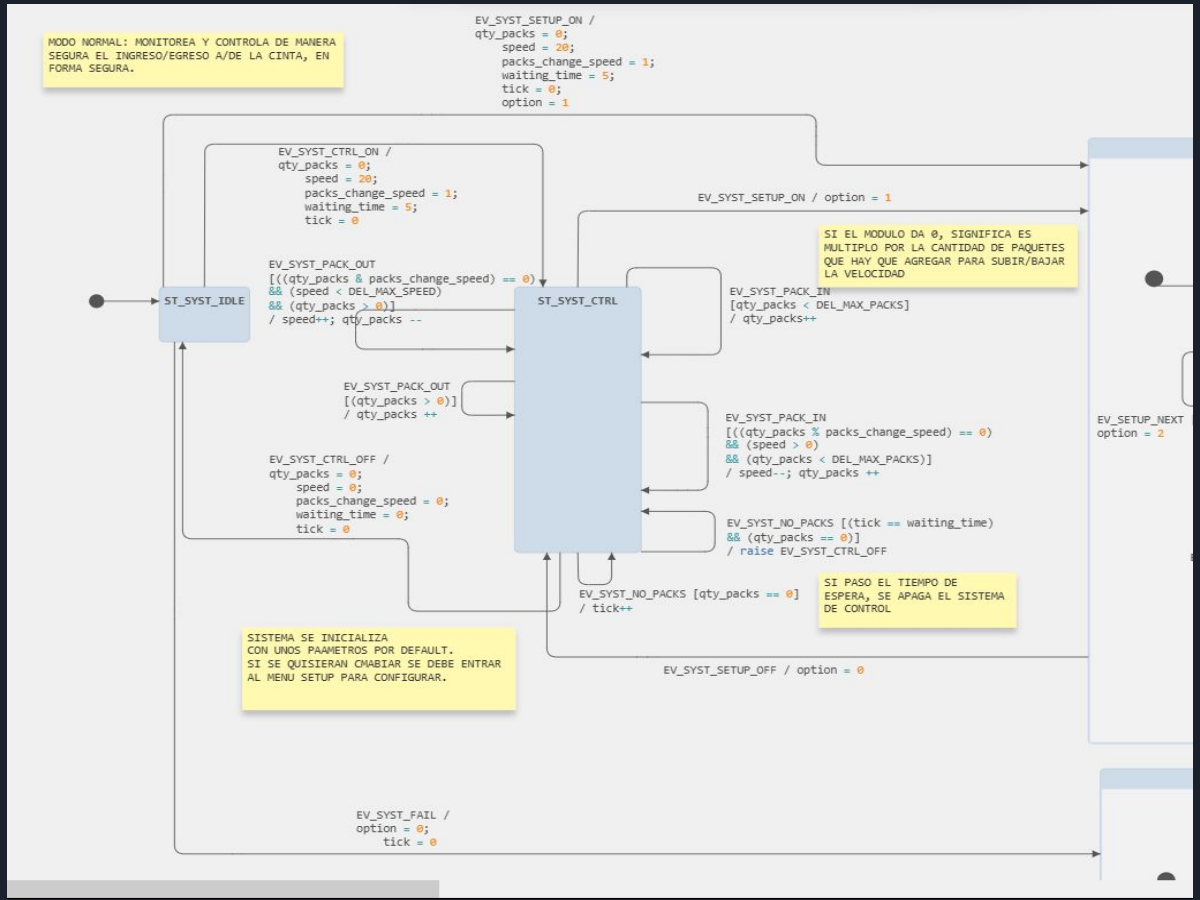
Además, se desarrolló la lógica correspondiente para los sensores y los actuadores, integrandolos al funcionamiento global del sistema



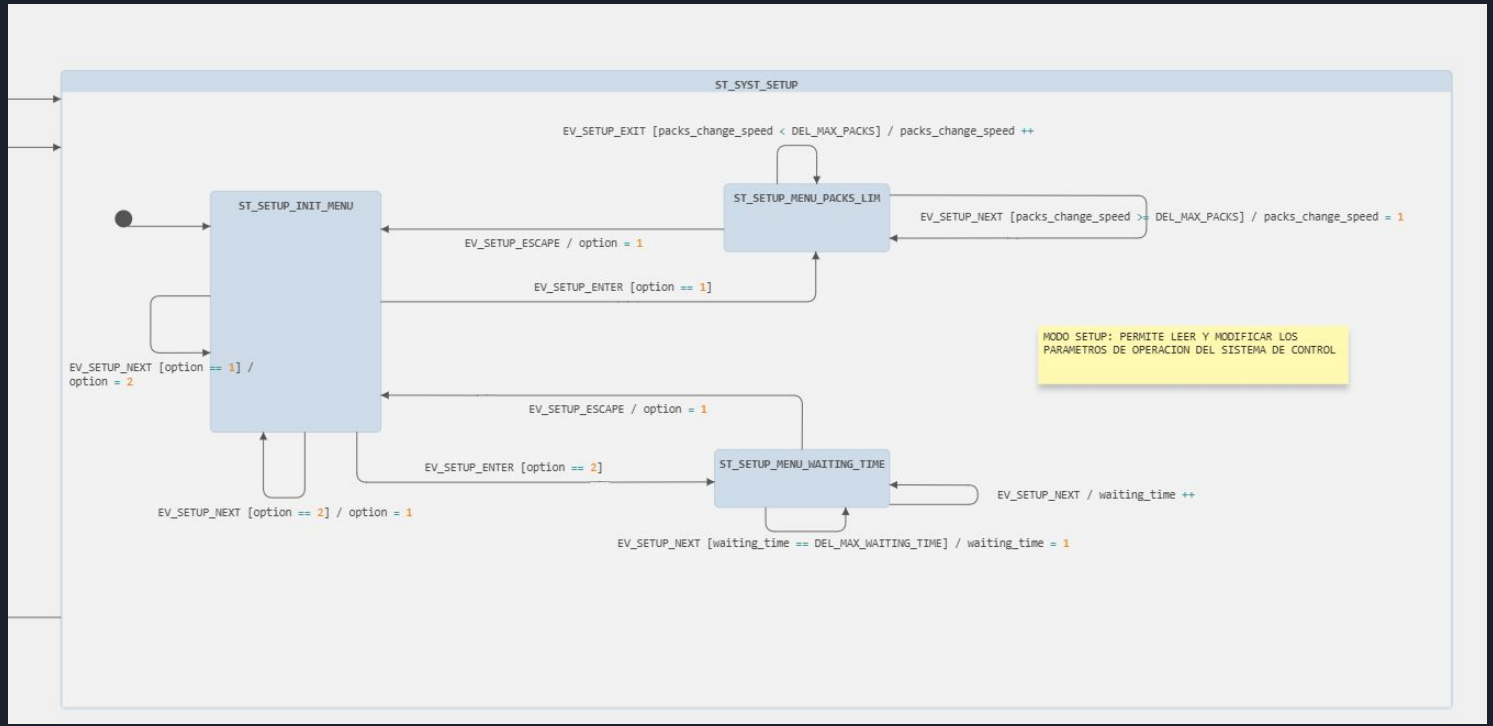
Statechart - Sensores



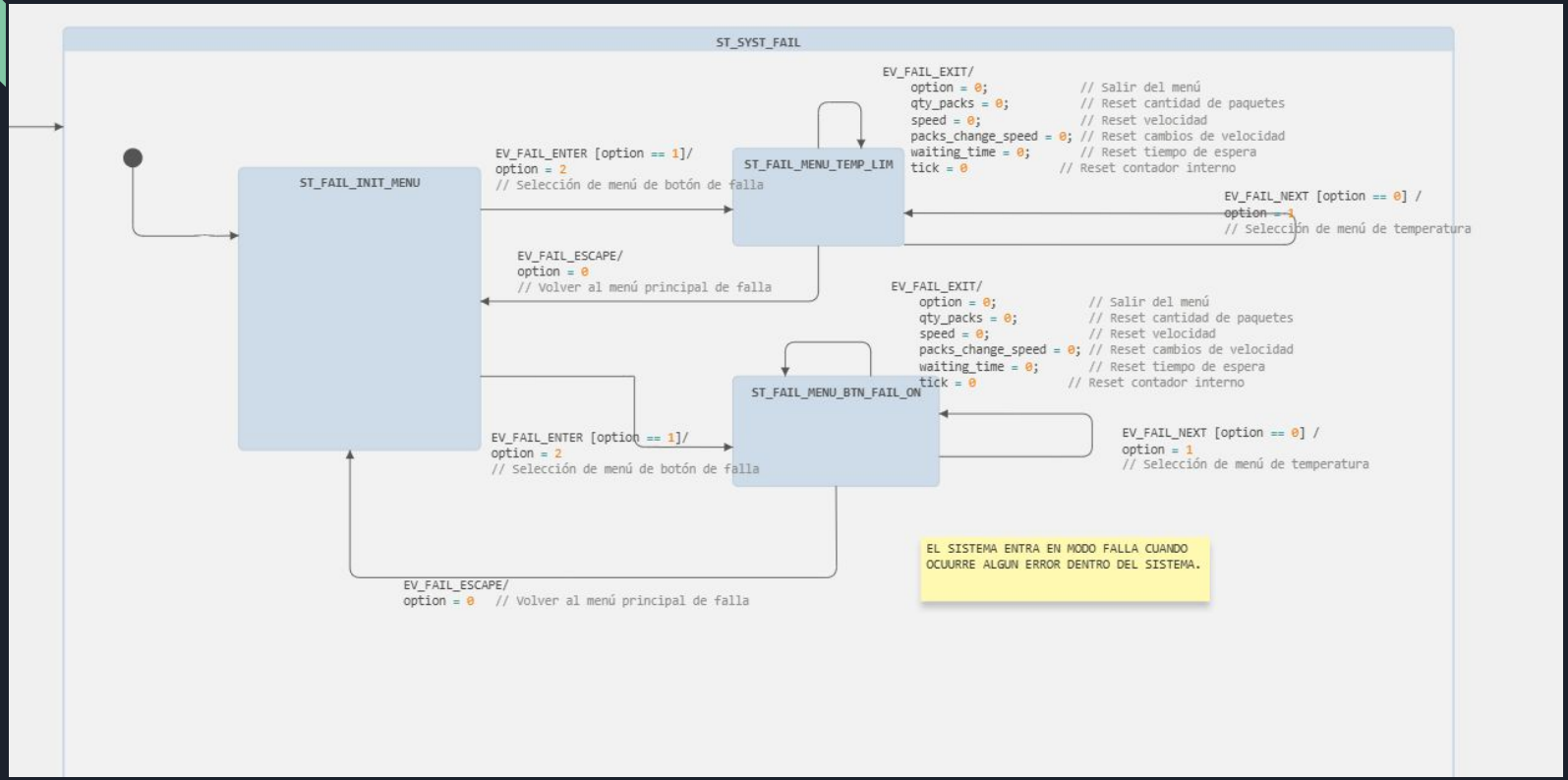
Statechart - Sistema



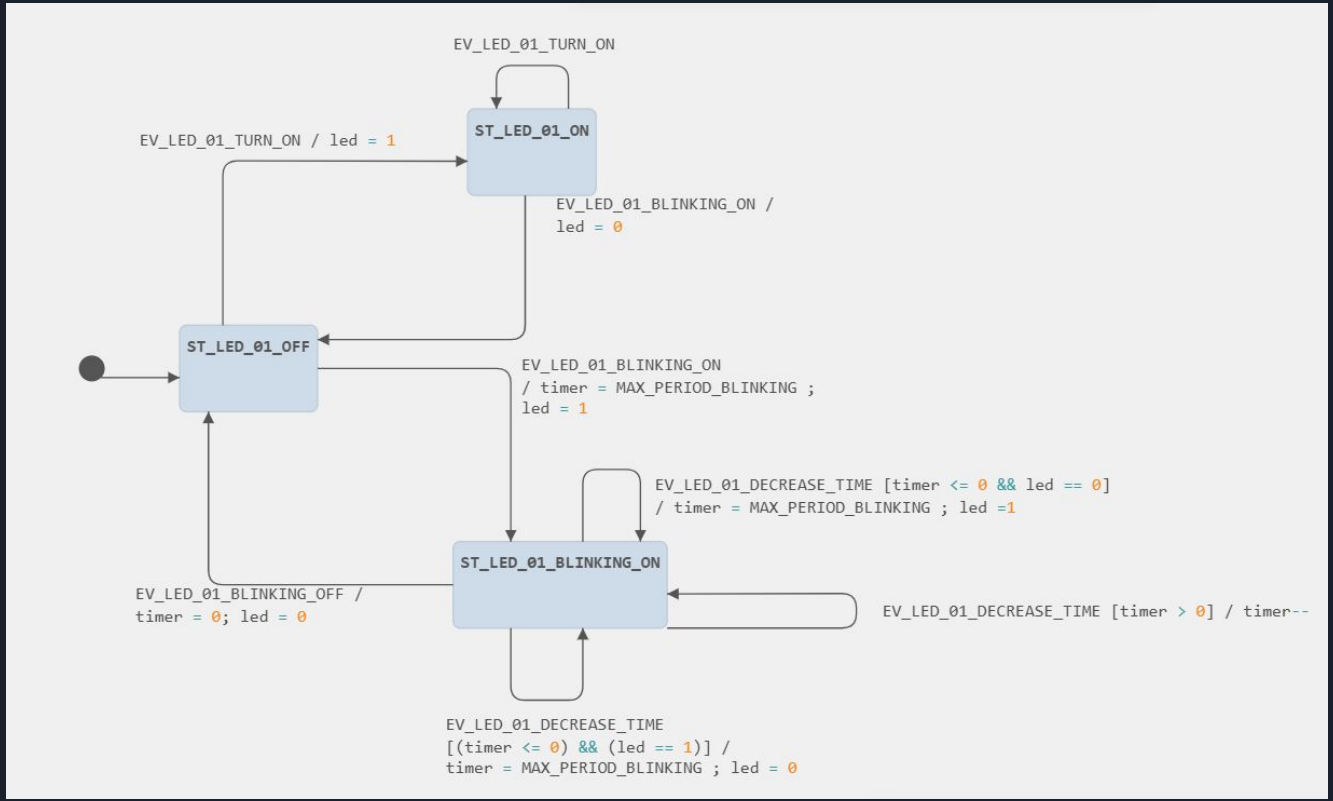
Statechart - Sistema



Statechart - Sistema



Statechart - Actuadores




Código

Estructura del Código I

Teniendo en cuenta los diagramas de estado anteriormente dispuestos, se genera el código correspondiente siguiendo esta lógica.

El código del proyecto se organiza en los siguientes archivos principales.

→ **task_sensor.c** : Captura y procesamiento de datos provenientes de los sensores, enviando la información al sistema de control.

→ **task_system.c + cinta.c** : Procesamiento de la lógica principal del sistema, coordinando las tareas y gestionando los eventos recibidos.

→ **task_actuator.c** : Control y activación de los actuadores en función de la lógica implementada y las decisiones tomadas por el sistema.

Estructura del Código II

→ **task_sen_temperature.c** : Convierte las muestras captadas por el ADC a su equivalente en temperatura, mide la temperatura del microcontrolador (sensor interno) y simula la del ambiente (potenciómetro).

→ **display.c :** Manejo y actualización de la información visualizada en la pantalla LCD.

→ **eeprom.c:** Guardar configuraciones o variables en memoria no volátil.

→ **app.c :** Inicializa y actualiza las tareas del proyecto.

→ **main.c** : Inicializa los periféricos y recursos del sistema. Además implementa el bucle principal para la ejecución cíclica de tareas.

Estructuras Implementadas

En esta sección se describen las estructuras utilizadas en el proyecto para modelar la lógica del sistema. Se definieron las estructuras de datos, así como los eventos y los estados correspondientes a los sensores, el sistema y los actuadores, con el objetivo de representar e implementar el comportamiento del sistema de control de manera clara y organizada.

task_sensor.h

```c
/* Identificadores de botones */
typedef enum {
    BTN_1 = 0,  // Contador Automático (PB15)
    BTN_2,      // Modo Falla (PB14)
    BTN_3,      // Cambio Manual / Automático (PB13)
    BTN_4,      // Cambio Velocidad (PB12)
    BTN_QTY
} task_sensor_id_t;

/******************** Estructuras ***************************************/
typedef struct {
    task_sensor_id_t id;
    GPIO_TypeDef *port;
    uint16_t pin;
    GPIO_PinState pressed_level; // Nivel lógico presionado
    uint32_t tick_max;           // Tiempo de debounce
} task_sensor_cfg_t;

typedef struct {
    uint32_t tick;
    task_sensor_state_t state;
    bool is_pressed;
} task_sensor_dta_t;
```

Typedef y Estructuras

→ **task_sensor_id_t:** Enumera y asigna un identificador único a cada sensor del sistema.
– Ejemplos: BTN_1 (Contador Automático), BTN_2 (Modo Falla), BTN_3 (Cambio Normal/Setu-Up), BTN_4 (Cambio Velocidad).

→ **task_sensor_cfg_t:** Contiene la configuración estática del sensor.

- *id:* Identificador del botón (tipo task_sensor_id_t).
- *port*: Puerto GPIO al que está conectado.
- *pin:* Número de pin dentro del puerto.
- *pressed_level*: Nivel lógico que indica pulsación (alto o bajo).
- *tick_max*: Tiempo máximo de debounce.

Typedef y Estructuras

→ **task_sensor_dta_t:** Almacena el estado dinámico del sensor durante la ejecución.

- *tick:* Contador de tiempo para control y debounce.
- *state:* Estado actual del botón (tipo task_sensor_state_t).
- *is_pressed:* Indica si el botón está presionado (true / false).

task_actuator.h

```c
/* Eventos que controlan la FSM del actuador */
typedef enum task_actuator_ev {
    EV_LED_XX_TURN_OFF,       // Apagar LED
    EV_LED_XX_TURN_ON,        // Encender LED
    EV_LED_XX_BLINKING_OFF,   // Detener parpadeo
    EV_LED_XX_BLINKING_ON     // Iniciar parpadeo
} task_actuator_ev_t;

/* Estados posibles de cada actuador */
typedef enum task_actuator_st {
    ST_LED_XX_OFF,    // Apagado
    ST_LED_XX_ON,     // Encendido fijo
    ST_LED_XX_BLINK   // Parpadeando
} task_actuator_st_t;

/* Identificadores de actuadores (en este caso 3 LEDs) */
typedef enum task_actuator_id {
    LED_1 = 0,
    LED_2,
    LED_3,
    LED_QTY           // Cantidad total de LEDs
} task_actuator_id_t;
```

task_actuator.h

```c
/********************* Estructuras ***************************************/
/* Configuración estática del actuador */
typedef struct {
    task_actuator_id_t identifier;
    GPIO_TypeDef *gpio_port;     // Puerto del GPIO
    uint16_t pin;                // Pin del GPIO
    GPIO_PinState led_on;        // Nivel lógico para encendido
    GPIO_PinState led_off;       // Nivel lógico para apagado
    uint32_t tick_blink;         // Tiempo base para parpadeo (en ticks)
} task_actuator_cfg_t;

/* Datos dinámicos (estado actual del actuador) */
typedef struct {
    uint32_t tick;               // Contador para manejo de tiempo en parpadeo
    task_actuator_st_t state;    // Estado actual (OFF, ON, BLINK)
    task_actuator_ev_t event;    // Último evento a procesar
    bool flag;                   // ON/OFF interno en modo BLINK
} task_actuator_dta_t;
```

Typedef y Estructuras

→ **task_actuator_ev_t:** Representa los eventos que controlan la FSM (Máquina de Estados Finita) del actuador.

→ **task_actuator_st_t:** Modela los estados posibles del actuador.

→ **task_actuator_id_t:** Asigna un identificador único a cada actuador (en este caso, LEDs).

→ **task_actuator_cfg_t:** Contiene la configuración estática del actuador.

- *identifier:* Identificador del actuador (tipo task_actuator_id_t).
- *gpio_port:* Puerto GPIO al que está conectado.
- *pin:* Pin dentro del puerto.

Typedef y Estructuras

- *led_on:* Nivel lógico para encendido.
- *led_off:* Nivel lógico para apagado.
- *tick_blink:* Tiempo base para parpadeo (en ticks).

→ **task_actuator_dta_t:** Contiene el estado dinámico del actuador durante la ejecución.

- *tick:* Contador para manejo de tiempo en parpadeo.
- *state:* Estado actual del actuador (OFF, ON, BLINK).
- *event:* Último evento a procesar.
- *flag:* ON/OFF interno en modo BLINK.

task_temperature.c

Tanto para la temperatura interna como para la externa, se utilizaron las siguientes expresiones que realizan el pasaje de las mediciones a grados centígrados del sensor interno.

→ **Sensor Interno:**

$$\frac{V_{25} - \frac{\text{medición}}{2^{12}}}{\text{Avg\_slope}} + 25$$

con V25 ≅ 1.43 V y Avg_slope ≅ 4.3 mV/C

task_temperature.c

→ Sensor externo (potenciómetro en PA0):

- El ADC mide el voltaje que entrega el potenciómetro, obteniendo un valor digital de 0 a 4095 (12 bits). Ese valor se convierte proporcionalmente a una escala de 0 a 100, representando el “grado” de posición del potenciómetro:
- No son grados Celsius reales, sino una escala normalizada para interpretar fácilmente la posición del sensor.

$$valor = \frac{lectura\ ADC \cdot 100}{4095}$$

task_display.c

→ **displayInit:**

- Inicializa el display en modo de comunicación de 4 bits.
- Configura parámetros iniciales como modo de visualización, control de cursor y comportamiento de entrada.

→ **displayCodeWrite:**

- Escribe comandos o datos en el display LCD.
- Gestiona la selección de pines y el envio de informacion a traves del bus de datos.

→ **displayCharPositionWrite:**

- Mueve el cursor a una posición específica en el LCD.

Ejemplo de Implementación

```
1    displayCharPositionWrite(0, 0);
2    displayStringWrite("CONTROL SYST OFF");
```

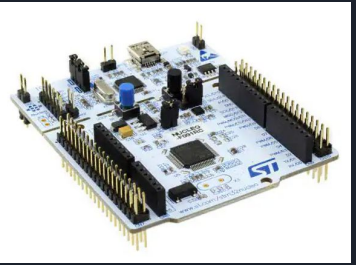

Hardware

Configuración

A partir del archivo .ioc del *IDE*, se configuraron los diferentes pines de propósito general (GPIO: *General-Purpose Input/Output*) y el periférico ADC (*Analog-to-Digital Converter*).

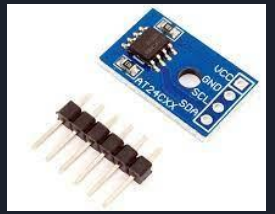
- **GPIO:** Se asignaron pines como entradas digitales para la lectura de los sensores y como salidas digitales para el control de los actuadores. Los pines configurados como entradas utilizan resistencias de pull-up internas.
- **ADC:** Está configurado para la lectura de señales provenientes del sensor de temperatura interno y para la lectura de la temperatura externa simulada a partir de un potenciómetro. En cada caso, se configuró el tiempo de muestreo adecuado para garantizar mediciones precisas y estables.

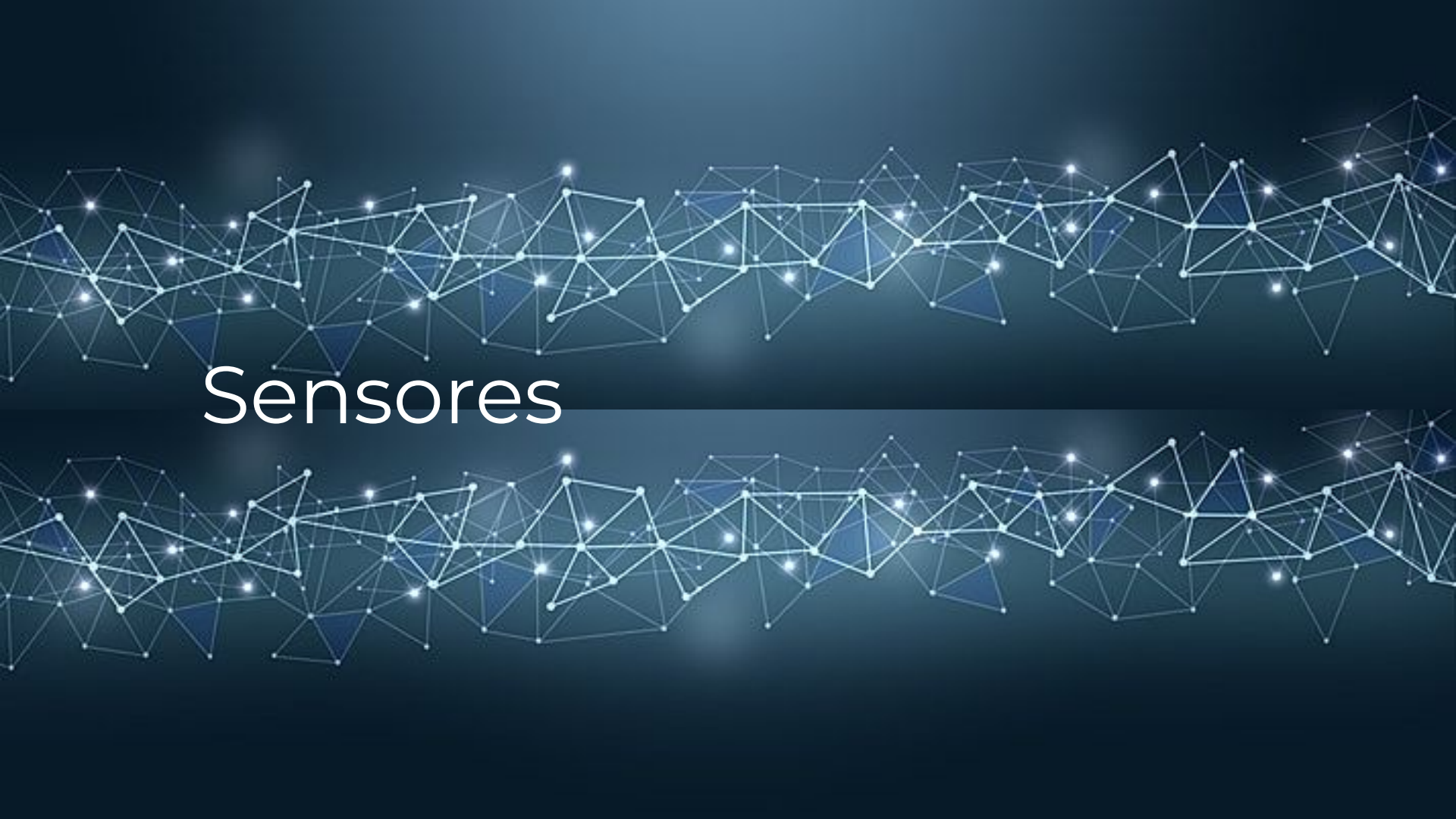
Componentes Utilizados



El sistema se compone de los siguientes elementos principales:

- **Microcontrolador:** STM32 F103RB, encargado del procesamiento y control central del sistema.
- **Sensores:** Botones utilizados para captar eventos e interactuar con el sistema.
- **Actuadores:** LEDs para proporcionar indicaciones visuales.
- **Display:** Pantalla LCD utilizada para mostrar información relevante sobre los parámetros del sistema.




Sensores

Sensores

→ **Paquete Ingresa** (Pulsador): Detecta un paquete más en la cinta.

→ **Velocidad** (Pulsador): Configura los parámetros de la velocidad en modo Set-Up.

→ **Barrera Infrarroja** (Pulsador): Detecta si ocurre un error en el Sistema de Control, entra en modo falla.

→ **Configuración Modos** (Pulsador): Ajusta los modos del Sistema de Control, modo Normal o Set-up.

→ **Sensores de Temperatura** (Potenciómetro y sensor interno del µC): Miden la temperatura externa y del microcontrolador.

Actuadores

Actuadores

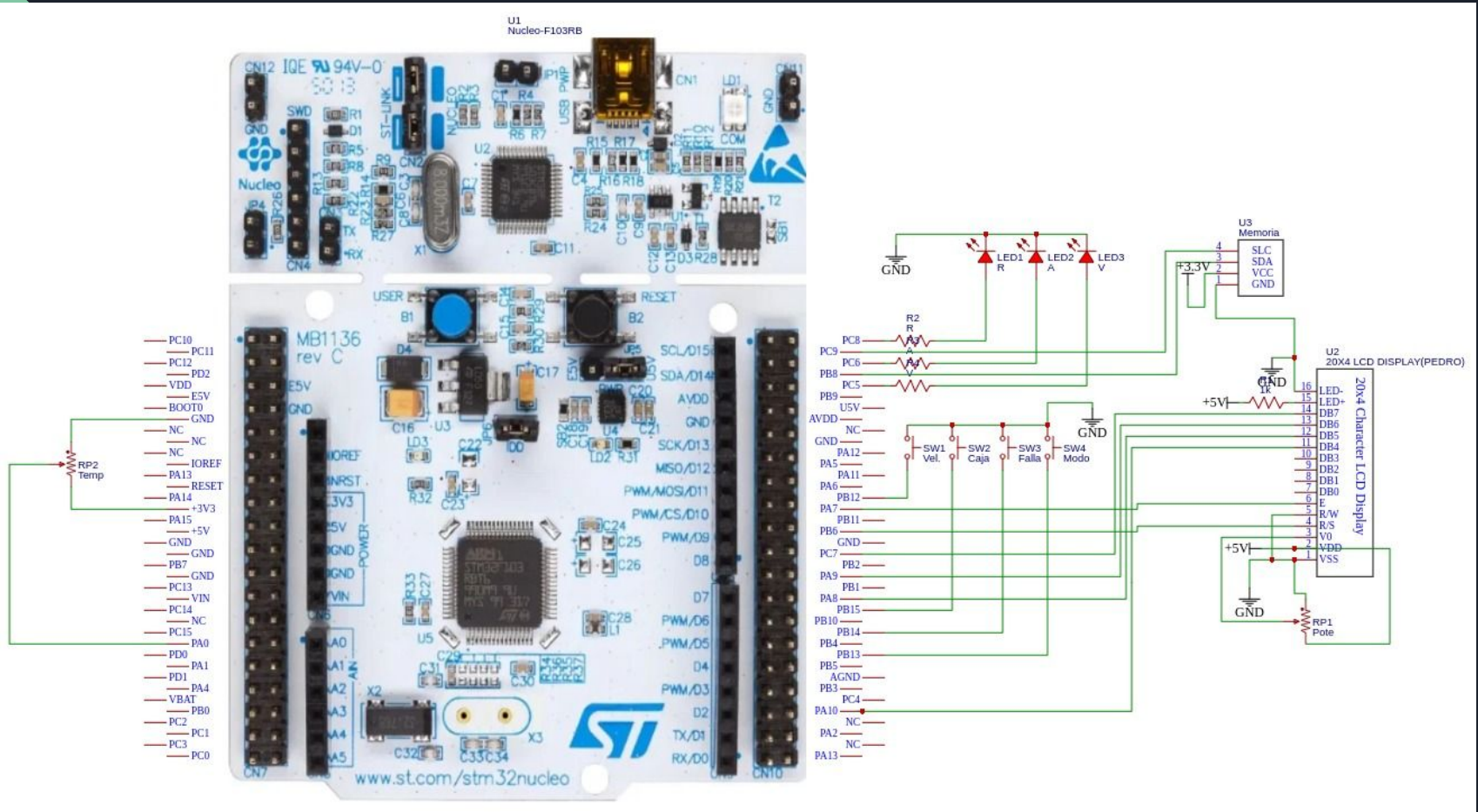
→ **LED Verde:** Indica que el Sistema de Control está activado.

→ **LED Amarillo:** Indica que el Sistema de Control está detenido.

→ **LED Rojo:** Indica que la cinta se encuentra en modo falla.

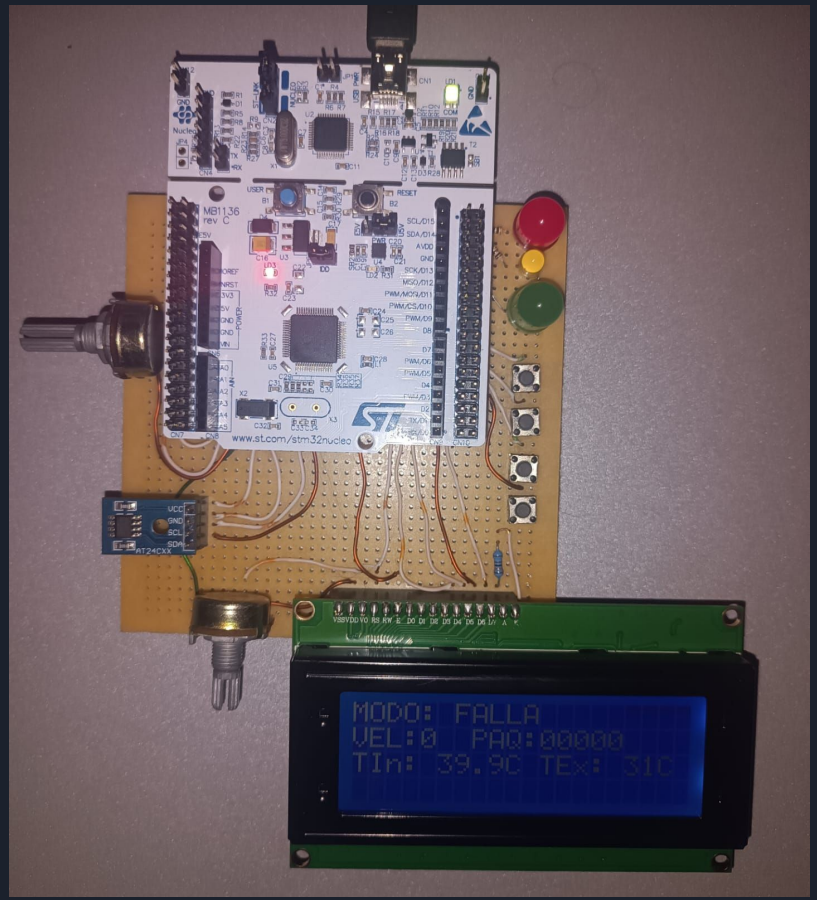
Esquema Eléctrico

Esquema Eléctrico




Modelo Final

Vista del Modelo Terminado



Consumo Eléctrico

Cálculo del Consumo Eléctrico

Se realizó un análisis del consumo eléctrico del sistema, teniendo en cuenta:

- Sensores Activos
- Actuadores en operación (LEDS)
- Consumo del microcontrolador

Se obtuvo el siguiente resultado utilizando un Tester USB:

→ Consumo promedio: 47 mA

→ Consumo promedio: 234 mWatts

Medición del WCET y Factor de Uso de CPU

Medición del WCET

Se midió el **Worst-Case Execution Time (WCET)** de cada tarea del sistema y el uso de GPIOs, teniendo en cuenta lo siguiente:

- Sensores = task_sensor.c + task_sen_temperature.c
- Sistema = display.c + eeprom.c
- Actuadores = task_actuator.c

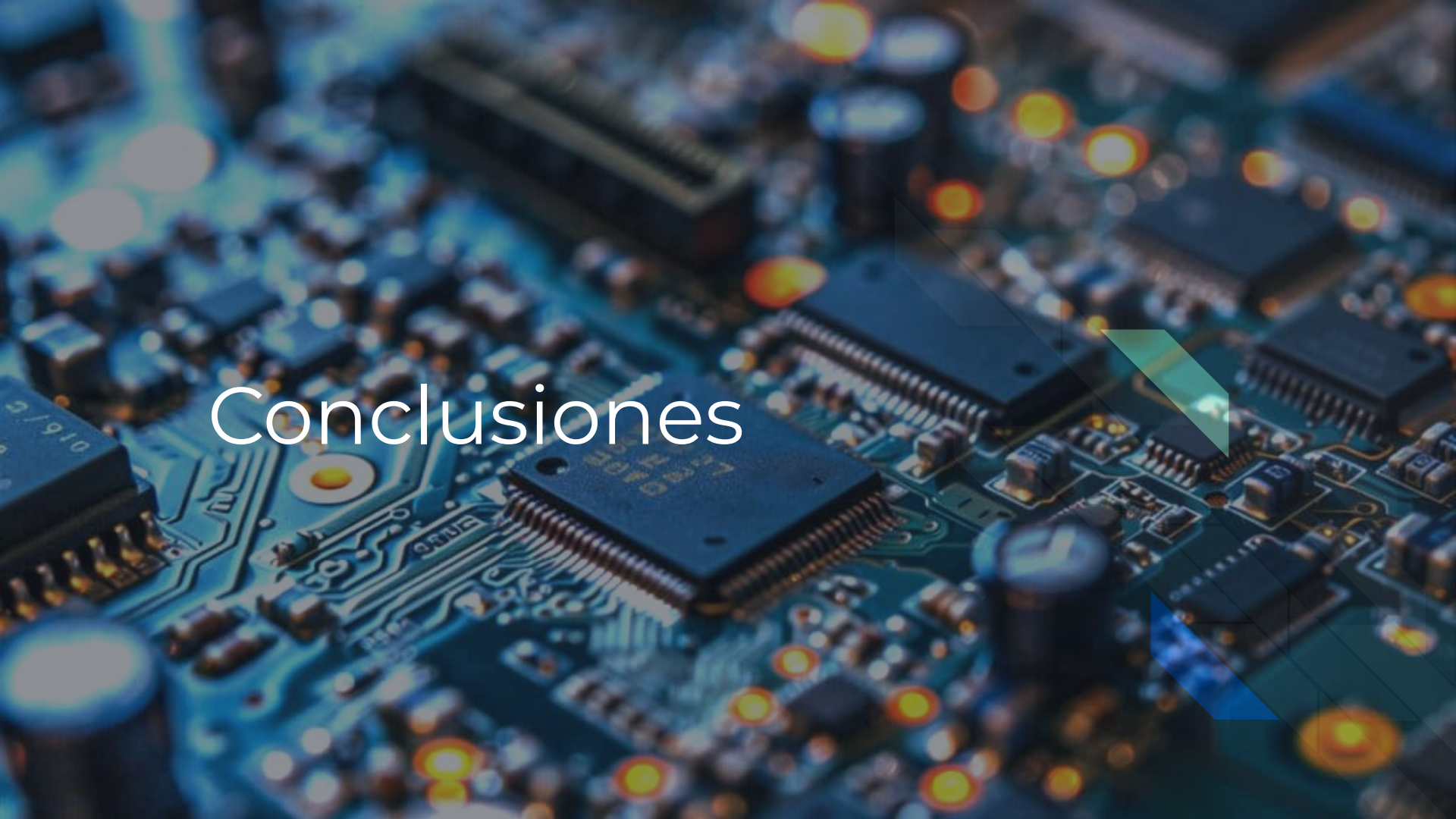
TAREA	WCET [µs]
Sensores	100
Sistema	12073
Actuadores	34

Factor de Uso de CPU

El factor de uso de CPU se midió a partir de los ciclos y el tiempo de ejecución, pasando por todas las tareas.

Los resultados obtenidos son los siguientes:

CICLOS	TIEMPO [ms]
3506	0.4



Conclusiones

Conclusiones

El desarrollo del sistema permitió aplicar los conceptos de la materia. A lo largo del proyecto, se alcanzaron los siguientes objetivos:

→ Se diseñó e implementó un sistema eficiente y funcional capaz de operar en modos Normal y Set Up, permitiendo flexibilidad y seguridad en la operación de la cinta transportadora.

→ Se resolvieron desafíos técnicos, como la integración de múltiples dispositivos y el modelado preciso de tareas mediante diagramas de estado, lo que garantizo un diseño optimizado y organizado.

→ Uno gran desafíos fue la programación de la memoria EEPROM, debido a la necesidad de gestionar correctamente la comunicación y el almacenamiento persistente de datos. Logrando la implementación adecuada de los protocolos de comunicación.

→ Se llevaron a cabo cálculos detallados, como el consumo eléctrico y el Worst-Case Execution Time (WCET), permitiendo validar el rendimiento del sistema en condiciones de operaciones de operación normales.

¡Gracias!