

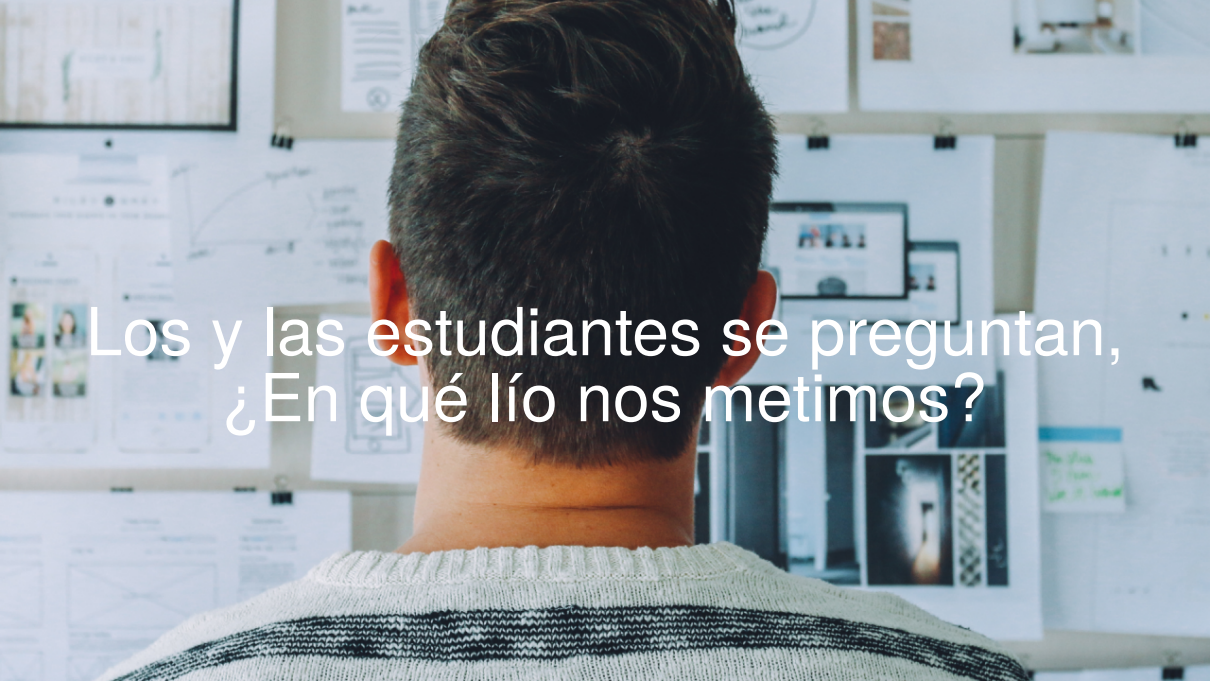


Facultad de Ingeniería, Universidad de Buenos Aires

# **Sistema de Control de Cinta Transportadora de Paquetes**

**Taller de Sistemas Embebidos, 2do C. 2024**

*Tomás Giani - Felipe Perassi -  
Tomás Rabinovich - Maximiliano Rodríguez*



Los y las estudiantes se preguntan,  
¿En qué lío nos metimos?

# **Planteamiento Inicial**

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



# Funcionamiento General

- **Modo Normal:** Monitorea y controla el ingreso y egreso de paquetes en 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:** Configura parámetros de operación del sistema:
  - Cantidad de paquetes para cambiar la velocidad de la cinta.
  - Tiempo de espera para la detención automática.

# Implementación

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

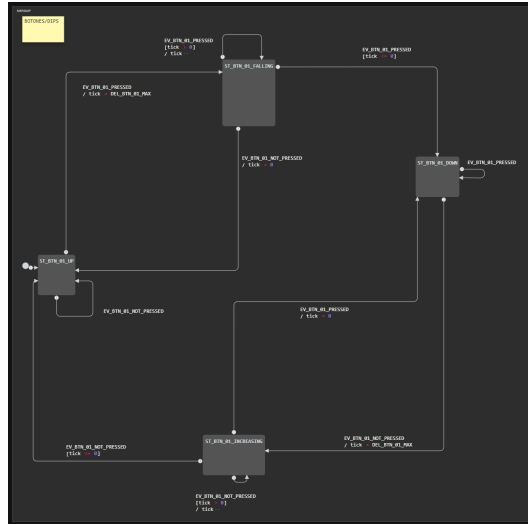
- **Sensores:** Captan los eventos y los envían al sistema para su procesamiento.
- **Sistema:** Recibe y procesa los eventos provenientes de los sensores, determinando las acciones a ejecutar.
- **Actuadores:** Ejecutan las acciones definidas por el sistema en respuesta a los eventos captados.

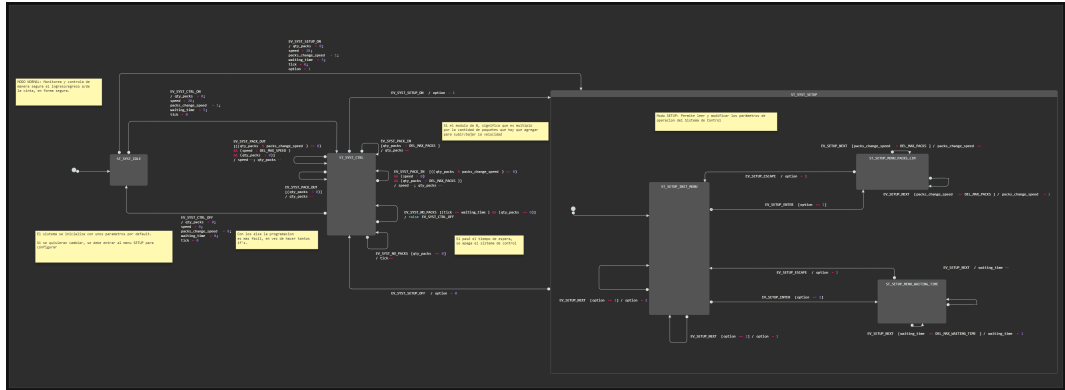
Sensores → Sistema → Actuadores

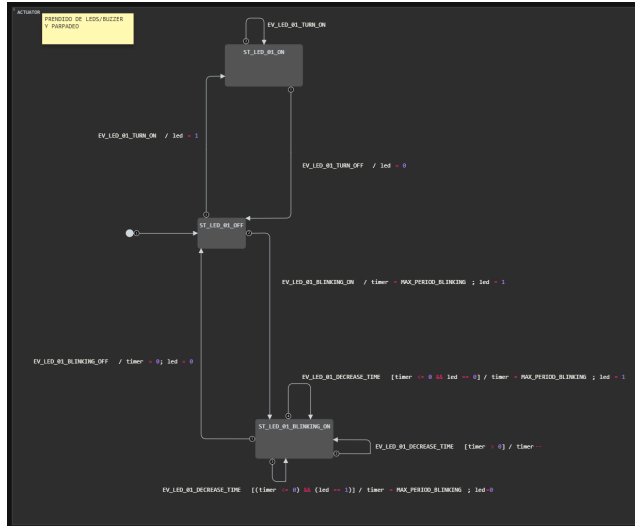


A partir de la herramienta *Itemis Create*, se diagramaron los *Statecharts* del sistema. Así, se abstraigo la lógica correspondiente al sistema que se desea modelar, para luego utilizarla en el código.

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







# Código

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:** 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.

- **task\_temperature.c:** Convierte las muestras captadas por el ADC a su equivalente en temperatura, mide la temperatura del microcontrolador (sensor interno) y la del ambiente (LM35).
- **display.c:** Manejo y actualización de la información visualizada en la pantalla LCD.
- **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.

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 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.



```
1  typedef enum task_sensor_ev {EV_BTN_01_PRESSED,  
2                               EV_BTN_01_NOT_PRESSED} task_sensor_ev_t;  
3  
4  /* States of Task Sensor */  
5  typedef enum task_sensor_st {ST_BTN_01_UP,  
6                               ST_BTN_01_FALLING,  
7                               ST_BTN_01_DOWN,  
8                               ST_BTN_01_INCREASING} task_sensor_st_t;  
9  
10 /* Identifier of Task Sensor */  
11 typedef enum task_sensor_id {ID_BTN_PACK_IN,  
12                              ID_BTN_PACK_OUT,  
13                              ID_DIP_NORMAL_OR_SETUP,  
14                              ID_DIP_INFRARED,  
15                              ID_DIP_CTRL_SYST_ON,  
16                              ID_BTN_ENTER,  
17                              ID_BTN_NEXT,  
18                              ID_BTN_ESCAPE} task_sensor_id_t;  
19  
20 typedef struct  
21 {  
22     task_sensor_id_t  identifier;  
23     GPIO_TypeDef *    gpio_port;  
24     uint16_t          pin;  
25     GPIO_PinState     pressed;  
26     uint32_t          tick_max;  
27     task_sensor_ev_t  signal_up;  
28     task_sensor_ev_t  signal_down;  
29 } task_sensor_cfg_t;  
30  
31 typedef struct  
32 {  
33     uint32_t          tick;  
34     task_sensor_st_t  state;  
35     task_sensor_ev_t  event;  
36 } task_sensor_dta_t;
```

- **task\_sensor\_ev\_t**: Representa los eventos que pueden ser detectados en los sensores.
- **task\_sensor\_st\_t**: Modela los estados posibles del sensor.
- **task\_sensor\_id\_t**: Asigna un nombre único a cada sensor.
- **task\_sensor\_dta\_t**: Contiene datos del sensor durante la ejecución.
  - *tick*: Contador de tiempo.
  - *state*: Estado actual del sensor.
  - *event*: Último evento detectado.

```
1  /* Events to excite Task System */
2  typedef enum task_system_ev {
3      EV_SVST_IDLE,
4      EV_SVST_CTRL_ON,
5      EV_SVST_CTRL_OFF,
6      EV_SVST_PACK_IN,
7      EV_SVST_PACK_OUT,
8      EV_SVST_NO_PACKS,
9      EV_SVST_SETUP_ON,
10     EV_SVST_SETUP_OFF,
11     EV_SETUP_ENTER,
12     EV_SETUP_ESCAPE,
13     EV_SETUP_NEXT,
14 } task_system_ev_t;
15
16
17 /* State of Task System */
18 typedef enum task_system_st {ST_SVST_IDLE,
19                             ST_SVST_CTRL,
20                             ST_SVST_SETUP} task_system_st_t;
21
22 /* Composed State of Task System */
23 typedef enum task_system_composed_st {ST_SETUP_INIT_MENU,
24                                       ST_SETUP_MENU_PACKS_LIM,
25                                       ST_SETUP_MENU_WAITING_TIME} task_system_composed_st_t;
26
27 typedef struct
28 {
29     uint32_t      tick;
30     uint32_t      speed;
31     uint32_t      qty_packs;
32     uint32_t      pack_rate;
33     uint32_t      waiting_time;
34     uint32_t      option;
35     task_system_st_t state;
36     task_system_composed_st_t composed_state;
37     task_system_ev_t event;
38     bool          flag;
39     bool          apagado_por_waiting_time;
40     bool          cambio_de_estado;
41 } task_system_data_t;
```

- **task\_system\_ev\_t**: Representa los eventos que pueden activar o cambiar el estado del sistema.
- **task\_system\_st\_t**: Modela los estados posibles del sistema.
- **task\_system\_composed\_st\_t**: Define los estados compuestos dentro del sistema.
- **task\_system\_dta\_t**: Contiene datos del sistema durante la ejecución.
  - *tick*: Contador de tiempo.
  - *speed*: Velocidad de operación del sistema.
  - *qty\_packs*: Cantidad de paquetes.

- *pack\_rate*: Cada cuantos paquetes cambia la velocidad.
- *waiting\_time*: Tiempo de espera del sistema al no detectar paquetes.
- *option*: Opción seleccionada en el menú de configuración.
- *state*: Estado actual del sistema.
- *composed\_state*: Estados compuestos dentro del Setup.
- *event*: Último evento detectado.
- *flag*: Indica que se realizó un evento.
- *apagado\_por\_waiting\_time*: Indica si el sistema se apagó por esperar demasiado tiempo.
- *cambio\_de\_estado*: Marca si hubo un cambio de estado.

```
1  /* Events to excite Task Actuator */
2  typedef enum task_actuator_ev {EV_LED_XX_TURN_OFF,
3                                EV_LED_XX_TURN_ON,
4                                EV_LED_XX_BLINKING_OFF,
5                                EV_LED_XX_BLINKING_ON} task_actuator_ev_t;
6
7  /* States of Task Actuator */
8  typedef enum task_actuator_st {ST_LED_XX_OFF,
9                                ST_LED_XX_ON,
10                               ST_LED_XX_BLINK} task_actuator_st_t;
11
12  /* Identifier of Task Actuator */
13  typedef enum task_actuator_id {ID_LED_MAX_SPEED,
14                                ID_LED_MIN_SPEED,
15                                ID_LED_CTRL_SYST,
16                                ID_LED_CTRL_SYST_IDLE,
17                                ID_BUZZER} task_actuator_id_t;
18
19  typedef struct
20  {
21      task_actuator_id_t  identifier;
22      GPIO_TypeDef *      gpio_port;
23      uint16_t            pin;
24      GPIO_PinState       led_on;
25      GPIO_PinState       led_off;
26      uint32_t            tick_blink;
27  } task_actuator_cfg_t;
28
29  typedef struct
30  {
31      uint32_t            tick;
32      task_actuator_st_t  state;
33      task_actuator_ev_t  event;
34      bool                flag;
35  } task_actuator_dta_t;
```

- **task\_actuator\_ev\_t**: Representan los eventos que activan o cambian el estado de los actuadores.
- **task\_actuator\_st\_t**: Modela los estados posibles del actuador.
- **task\_actuator\_id\_t**: Asigna un nombre único a cada led o buzzer.
- **task\_actuator\_cfg\_t**: Estructura que define la configuración de cada actuador.
  - *identifier*: Identificador del actuador.
  - *gpio\_port*: Puerto GPIO al que está conectado.
  - *pin*: Pin del GPIO al que está asociado.

- *led\_on*: Estado de encendido del LED.
- *led\_off*: Estado de apagado del LED.
- *tick\_blink*: Tiempo para el parpadeo del LED.
- **task\_actuator\_dta\_t**: Contiene datos del actuador durante la ejecución.
  - *tick*: Contador de tiempo.
  - *state*: Estado actual del actuador.
  - *event*: Último evento detectado.
  - *flag*: Indica que está prendido.



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

- **LM35:**

$$\frac{\text{medición} \cdot 0.452 \cdot V_{CC}}{10 \text{ mV} \cdot 2^{12}}$$

- **Sensor Interno:**

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

con  $V_{25} \approx 1.43 \text{ V}$  y  $\text{Avg\_slope} \approx 4.3 \frac{\text{mV}}{C}$

- **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 envío de información a través del bus de datos.

- **displayCharPositionWrite:**

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



```
1 displayCharPositionWrite(0, 0);  
2 displayStringWrite("CONTROL SYST OFF");  
3 displayCharPositionWrite(0, 1);  
4 displayStringWrite("PRESS BUTTON 1 ");
```

# Hardware

A partir del archivo *.ioc* del *IDE*, se configuraron los diferentes pines de propósito general (GPIO) y el periférico ADC.

- **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:** El *ADC1* está configurado para la lectura de señales provenientes del sensor de temperatura interno, mientras que el *ADC2* para la lectura del LM35. En cada caso, se configuró el tiempo de muestreo adecuado para garantizar mediciones precisas y estables.

El sistema se compone de los siguientes elementos principales:

- **Microcontrolador:** STM32F103RB, encargado del procesamiento y control central del sistema.
- **Sensores:** Botones y *dip switches* utilizados para captar eventos e interactuar con el sistema.
- **Actuadores:** LEDs para proporcionar indicaciones visuales y un buzzer para generar alertas sonoras.
- **Display:** Pantalla LCD utilizada para mostrar información relevante sobre los parámetros del sistema.

# Sensores

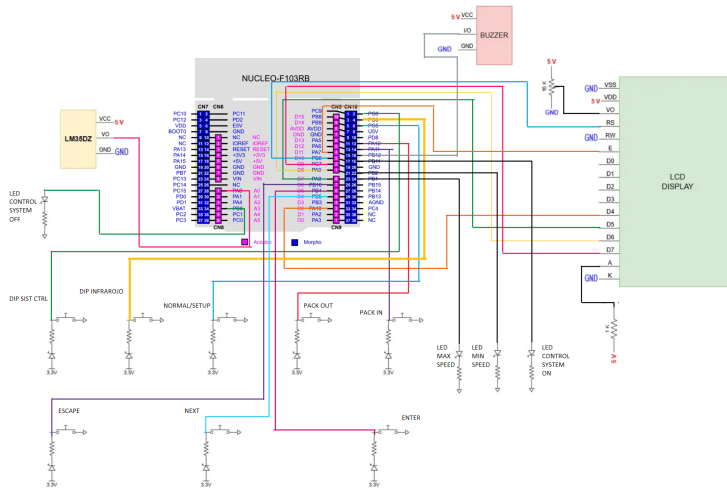
- **Paquete Ingresa** (Pulsador): Detecta un paquete más en la cinta.
- **Paquete Egres**a (Pulsador): Detecta un paquete menos en la cinta.
- **Barrera Infrarroja** (Dip Switch): Detecta si no hay paquetes en la cinta.
- **Activar/Desactivar Sistema** (Dip Switch): Activa o desactiva el Sistema de Control.
- **Configuración/Enter/Next/Escape** (Pulsadores): Configura parámetros del sistema.
- **Sensores de Temperatura** (LM35 y sensor interno del uC): Miden la temperatura ambiente y del microcontrolador.



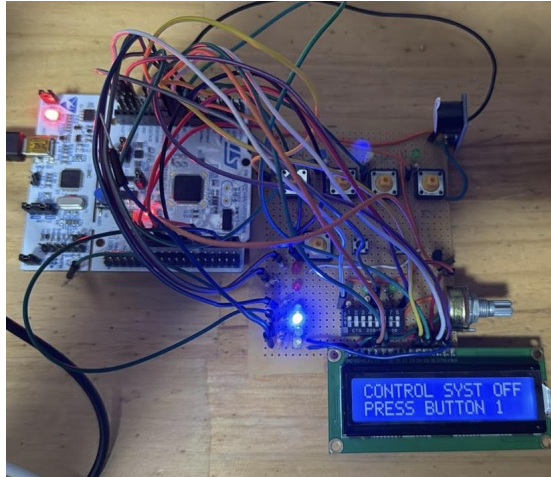
# Actuadores

- **LED de máxima velocidad:** Indica que el motor mueve la cinta a máxima velocidad.
- **LED de mínima velocidad:** Indica que el motor mueve la cinta a mínima velocidad.
- **LED titilante:** Indica que el Sistema de Control está activado.
- **LED de desactivación:** Indica que el Sistema de Control está desactivado.
- **Sirena ululante:** Indica que el sistema detuvo la cinta.

# Esquema Eléctrico



# Modelo Final



# Consumo Eléctrico

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

- Sensores activos.
- Actuadores en operación (LEDs y Buzzer).
- Consumo del microcontrolador.



Los resultados obtenidos fueron los siguientes:

- **Consumo promedio: 37.56 mA.** Calculado considerando:
  - Sensor LM35DZ (**60  $\mu$ A**).
  - 3 LEDs activos (**22.5 mA**).
  - Buzzer al **50%** de su ciclo de trabajo.
- **Consumo máximo estimado: 112.56 mA.** Corresponde a todo el sistema activo (incluyendo consumo del microcontrolador y pérdidas):
  - Sensor LM35DZ (**60  $\mu$ A**).
  - 11 LEDs activos (**82.5 mA**).
  - Buzzer al **100%** de su ciclo de trabajo (**30 mA**).

# **Medición del WCET y Factor de Uso de CPU**

Se midió el **Worst-Case Execution Time (WCET)** de cada tarea del sistema y el uso de GPIOs, obteniendo los siguientes valores:

Tarea	WCET [ms]
task_sensor	12.6
task_system	545.3
task_actuator	4.2

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]
94474	11.8

# 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 garantizó un diseño optimizado y organizado.
- 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 operación normales.



**¡Gracias!**