Alumno: Gonzalo Vila

Plataforma embebida: STM32 NUCLEO-F429ZI

Aplicación:
La aplicación forma parte del trabajo práctico final del alumno. Este consiste en el desarrollo de un software para verificar prototipos de hardware antes de que sean entregados al cliente, quien programará el firmware.

Para el trabajo final de esta materia, se desarrollará un módulo de software que se desplegará sobre el hardware embebido y permanecerá en modo de "escucha" hasta recibir un lote de comandos (simulados mediante la presión del botón de usuario).

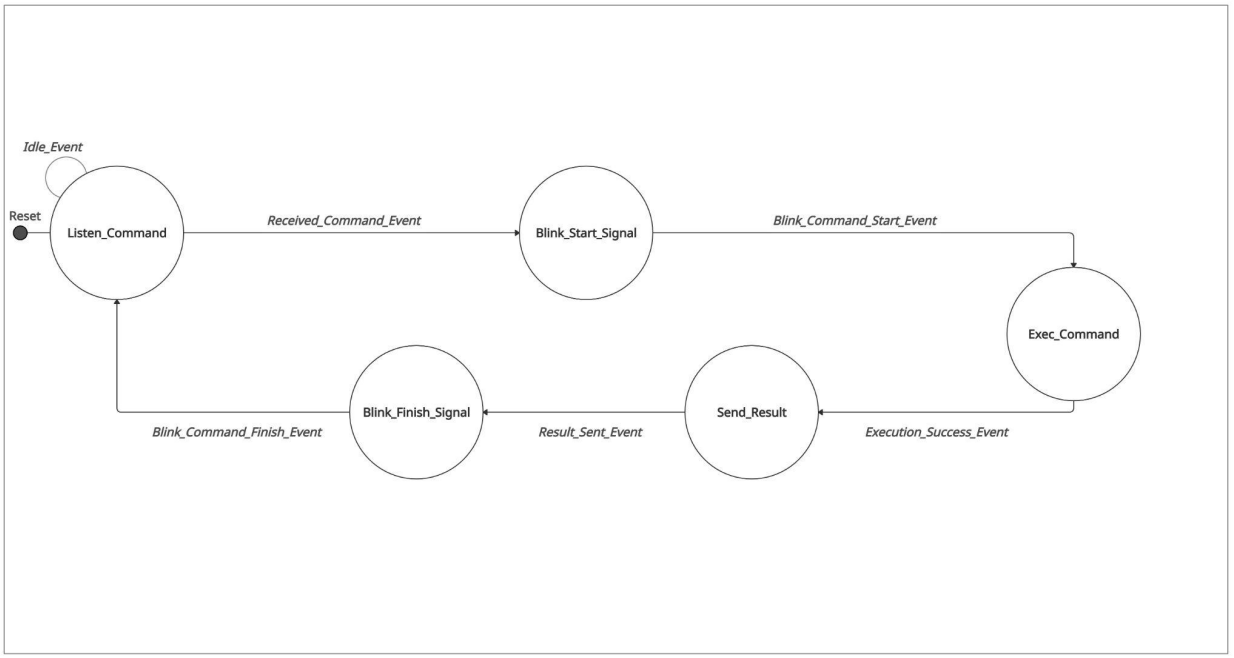
El lote de comandos debe entrar en ejecución, y los procesos deben ser gestionados de forma individual. Se debe notificar al usuario que la ejecución del lote ha comenzado mediante el encendido de la luz verde de la placa. El resultado de la ejecución del comando debe retornar a una consola en la PC de destino a través de la UART. Al terminar se apaga el LED verde.

Los comandos disponibles retornarán información acerca de su estado (processing/finished).

Periféricos (1 ó 2):
UART

Botón de usuario

Diagrama de estado de MEF



Breve descripción de cada estado

Listen_Command: Implementa un ciclo que escucha la llegada de comandos a través de la pulsación del botón de usuario en la placa. Determina cuándo liberar el lote de comandos a las fases siguientes.

Blink_Start_Signal: Enciende el LED verde de la placa para notificar al usuario que la ejecución de un comando ha comenzado.

Exec_Command: Ejecuta los comandos. Cada uno de ellos tiene una duración, tras la cual se considera finalizado. Antes de ello estará en running.

Send_Result: Envía la salida del comando de verificación de hardware a través de la UART hacia la consola en la PC.

Blink_Finish_Signal: Apaga el led verde para señalizar que el lote de comandos ha terminado.

Módulos de software

EDTF_CommandServer.h: Archivo de cabecera que expone los prototipos de las funciones para inicializar y controlar el servidor de comandos.

EDTF_CommandServer.c: Realiza la gestión del servidor de comandos y gestiona un buffer con los procesos.

EDTF_API_uart.h: Archivo de cabecera que expone los prototipos de las funciones para controlar el envío de datos a través de la UART.

EDTF_API_uart.c: Contiene las funciones que inicializan y gestionan la comunicación de datos a través de la UART.

EDTF_API_led.h: Archivo de cabecera que expone los prototipos de las funciones para controlar los LEDs del board.

EDTF_API_led.c: Contiene las funciones que inicializan y controlan los LEDs del board.

EDTF_API_delay.h: Librería para utilizar retardos no bloqueantes.

EDTF_API_delay.c: Contiene las funciones que inicializan y gestionan los retardos no bloqueantes.

Prototipos de las principales funciones

EDTF_CommandServer.h

```
void initCommandServer (void);

void updateCommandServerState (void);
```

EDTF_CommandServer.c

```
typedef uint8_t serverpid_t;

typedef enum {
        Listen_Command,
        Blink_Start_Signal,
        Exec_Command,
        Send_Result,
        Blink_Finish_Signal,
} CommandServerState_t;


typedef struct {
        serverpid_t pid;
        bool_t   finishedProcessing;
}process_t;


typedef struct {
```

```
        process_t       processes [EXEC_BUFFER_MAX_COMM_NUMBER];
        bool_t          isbufferReadyForProcessing;
    }processBuffer_t;
    static processBuffer_t processBuffer;
    static CommandServerState_t currentCommandServerState;
```

EDTF_API_uart.h

```
    bool_t uartInit(void);
    void uartSendString(uint8_t *);
```

EDTF_API_uart.c

```
    void uartSendString(uint8_t *pstring)
```

EDTF_API_led.h

```
    typedef enum {
        Green_Led,
        Blue_Led,
        Red_Led
    }BoardLed_t;

    typedef enum {
        On,
        Off
    }BoardLedState_t;
```

EDTF_API_led.c

```
    void initiateLed (BoardLed_t led)
    void turnLed(BoardLed_t led, BoardLedState_t state)
    void toggleLed(BoardLed_t led)
```

EDTF_API_debounce.h

```
    bool_t debounceInit(void);
    void debounceUpdate(void);
```

```
bool_t consumeSinglePressEvent(void);
```

EDTF_API_delay.h

```
typedef struct{
   tick_t startTime;
   tick_t duration;
   bool_t running;
} delay_t;

void delayInit (delay_t *, tick_t);
bool_t delayRead (delay_t *);
void delayWrite (delay_t *, tick_t);
void configureDelayErrorHandler (void (*p)(void));
```