LCD & KEYPAD Non-Blocking Design

Author: Mahmoud Saeed Mowafey.
Reviewers: Sprints/ Expert.
Date: 11/05/2023

1. Project Introduction

Simple LCD & KEYPAD Interfacing

1. Project Introduction

This project involves developing software to interface LCD & Keypad with microcontroller.

2. High Level Design

2.1. System Architecture

2.1.1. Definition

Layered Architecture (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

Microcontroller Abstraction Layer (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

Hardware Abstraction Layer (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

Service Layer is the topmost layer of Basic Software Architecture. The service layer constitutes an operating system, which runs from the application layer to the microcontroller at the bottom.

2.1.2. Layered Architecture

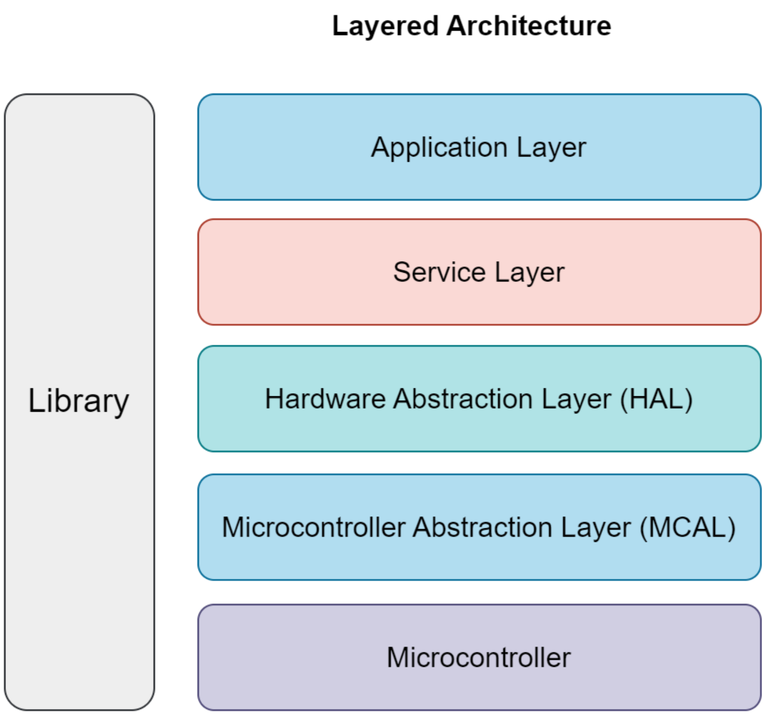
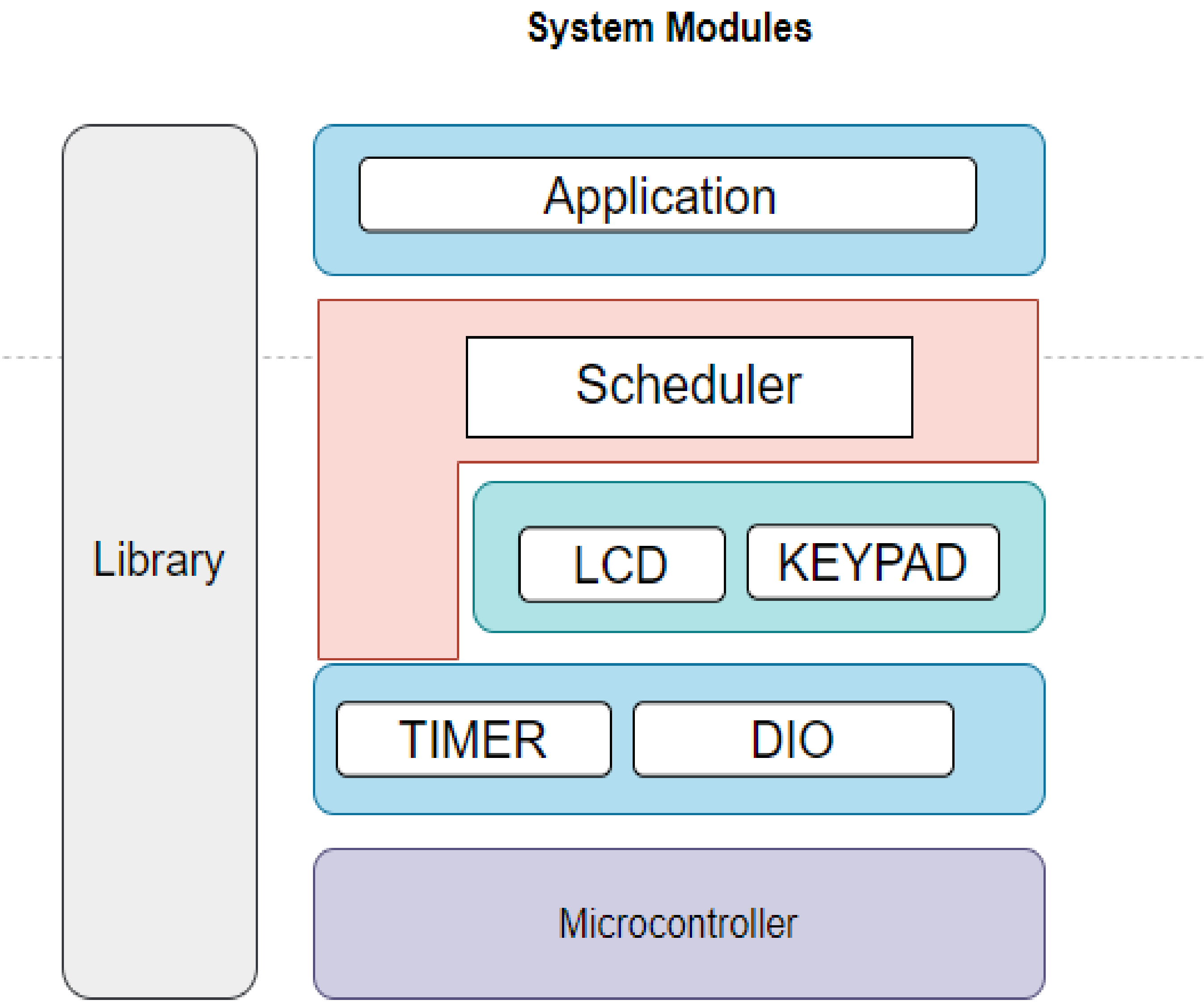


Figure 1. Layered Architecture Design

2.1.3. System Modules



2.2. Modules Description

2.2.1. TIMER Module

The *TIMER* module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs.

2.2.2. KEYPAD Module

Keypad is an analog switching device which is generally available in matrix structure. It is used in many embedded system applications for allowing the user to perform a necessary task. A matrix *Keypad* consists of an arrangement of switches connected in matrix format in rows and columns. The rows and columns are connected with a microcontroller such that the rows of switches are connected to one pin and the columns of switches are connected to another pin of a microcontroller.

2.2.3. LCD Module

LCD stands for "*Liquid Crystal Display*," which is a type of flat-panel display used in electronic devices to display text and graphics. The module is being used in our project to display information about the current and desired temperature of an air conditioning system. We're using the *4-bit* mode to reduce the number of I/O pins needed to interface with the *LCD*. The module includes a controller, a display, and a backlight. By interfacing with the *LCD* module and writing the necessary code.

2.2.4. Scheduler Module

A co-operative scheduler provides a simple, highly predictable environment. The scheduler is written entirely in ‘C’ and becomes part of the application: this tends to make the operation of the whole system more transparent and eases development, maintenance and porting to different environments. Memory overheads are seven bytes per task and CPU requirements (which vary with tick interval) are low.

2.3. Drivers' Documentation (APIs)

2.3.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

2.3.2. MCAL APIs

2.3.2.3. TIMER Driver

```
| Initializes timer0 at normal mode
|
| This function initializes/selects the timer_0 normal mode for the
| timer, and enables the ISR for this timer.
| Parameters
|           [in] en_a_interrputEnable value to set the interrupt
|                                     bit for timer_0 in the TIMSK reg.
|            [in] **u8_a_shutdownFlag double pointer, acts as a main switch for
|                                     timer0 operations.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|           the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|           otherwise)
|
EN_TIMER_ERROR_T TIMER_timer0NormalModeInit(EN_TIMER_INTERRPUT_T
en_a_interrputEnable, u8 ** u8_a_shutdownFlag);



| Creates a delay using timer_0 in overflow mode
|
| This function Creates the desired delay on timer_0 normal mode.
| Parameters
|           [in] u16_a_interval value to set the desired delay.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|           the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|           otherwise)
|
EN_TIMER_ERROR_T TIMER_delay_ms(u16 u16_a_interval);
```

```
| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_0.
| Parameters
|             [in] u16_a_prescaler value to set the desired prescaler.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|             the operation
|              (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_timer0Start(u16 u16_a_prescaler);


| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_0.
|
| Return
|      void
|
void TIMER_timer0Stop(void);


| Initializes timer2 at normal mode
|
| This function initializes/selects the timer_2 normal mode for the
|  timer, and enables the ISR for this timer.
| Parameters
|             [in] en_a_interrputEnable value to set
|             the interrupt bit for timer_2 in the TIMSK reg.
|
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|             the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|             otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2NormalModeInit(EN_TIMER_INTERRPUT_T);


| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_2.
| Parameters
|             [in] void.
| Return
```

```
|     void
|
void TIMER_timer2Stop(void);


| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_2.
| Parameters
|            [in] u16_a_prescaler value to set the desired prescaler.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|            the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|            otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2Start(u16 u16_a_prescaler);

| Creates a timeout delay in msy using timer_2 in overflow mode
|
| This function Creates the desired delay on timer_2 normal mode.
| Parameters
|            [in] u16_a_interval value to set the desired delay.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|            the operation
|             (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_intDelay_ms(u16 u16_a_interval);



| Set callback function for timer overflow interrupt
|
| Parameters
|            [in] void_a_pfOvfInterruptAction Pointer to the function to be
|                                        called on timer overflow interrupt
| Return
|     EN_TIMER_ERROR_T Returns TIMER_OK if callback function is set
|                        successfully, else returns TIMER_ERROR
|
EN_TIMER_ERROR_T TIMER_ovfSetCallback(void
(*void_a_pfOvfInterruptAction)(void));



| Interrupt Service Routine for Timer2 Overflow.
|      This function is executed when Timer2 Overflows.
|      It increments u16_g_overflow2Ticks counter and checks whether
|      u16_g_overflow2Numbers is greater than u16_g_overflow2Ticks.
```

```
|       If true, it resets u16_g_overflow2Ticks and stops Timer2.
|       It then checks whether void_g_pfOvfInterruptAction is not null.
|       If true, it calls the function pointed to by
|      void_g_pfOvfInterruptAction.
|
| Return
|      void
|
ISR(TIMER2_ovfVect);
```

2.3.3. HAL APIs

2.3.3.1. LCD APIs

```
| Initializes the LCD module.
|
| This function initializes the LCD module by configuring the data port,
| configuring the LCD to 8-bit mode, setting the display to on with cursor
| and blink, setting the cursor to increment to the right, and clearing
| the display.|
| Return
|     void
|
void LCD_init(void);
```

```
| Sends a command to the LCD controller
|
| Sends the upper nibble of the command to the LCD's data pins, selects
|  the command register by setting RS to low,
| generates an enable pulse, delays for a short period, then sends the
|  lower nibble of the command and generates
| another enable pulse. Finally, it delays for a longer period to ensure
|  the command has been executed by the LCD
| controller.
|
| Parameters
|            [in] u8_a_cmd The command to be sent
|
void LCD_sendCommand(u8 u8_a_cmd);
```

```
| Sends a single character to the LCD display
|
| This function sends a single character to the LCD display by selecting
|  the data register and sending the
| higher nibble and lower nibble of the character through the data port.
| The function uses a pulse on the enable pin to signal the LCD to read
|  the data on the data port.
| The function also includes delays to ensure proper timing for the LCD
|  to read the data.
|
| Parameters
|            [in] u8_a_data single char ASCII data to show
|
void LCD_sendChar(u8 u8_a_data);
```

```
| Displays a null-terminated string on the LCD screen.
|
| This function iterates through a null-terminated string and displays it
| on the LCD screen. If the character '\n' is encountered, the cursor is
| moved to the beginning of the next line.
|
| Parameters
|             [in]u8Ptr_a_str A pointer to the null-terminated string to be
|                           displayed.
| Return
|      void
|
void LCD_sendString(u8 * u8Ptr_a_str);

| Clears the LCD display
void LCD_clear(void);
```

2.3.3.2. KPD APIs

```
| This function reads input from a keypad and returns the pressed key value after
| debouncing.
|
| Parameters
|  -          [in] pu8_a_returnedKeyValue Pointer to a u8 variable that will hold the
|                  value of the pressed key.
| Return
|      STD_OK if successful, otherwise STD_NOK
|
u8 KPD_getPressedKey(u8 *pu8_a_returnedKeyValue);
```

2.3.4. Service Layer APIs

2.3.4.1. Scheduler APIs

```
| When function is due to run This function will run it
| it must be called repeatedly from the main loop
| Parameters
|  -          void
| Return
|             void
void CSCH_dispatcher (void);


| Create a task to be executed at regular intervals
| | Parameters
|             [in] *pTask : pointer to task.
|                   delay : interval before task is executed.
|                   period : periodic time to call this task again.
| Return
|            void
void CSCH_addTask (void (void * pTask ), const u32 u32_l_delay,
                    const u32    u32_l_period);


| Remove task from the scheduler
| | Parameters
|             [in] u32_l_taskIndex: task index provided by CSCH_addTask().
|
| Return
|             EN_CSCH_errorState.
EN_CSCH_errorState CSCH_deletTask (const u32 u32_l_taskIndex);




| Enter idle mode
| | Parameters
|             [in] void.
|
| Return
|             void
void CSCH_goToSleep (void);
```

2.3.4. LCD & KEYPAD APP APIs

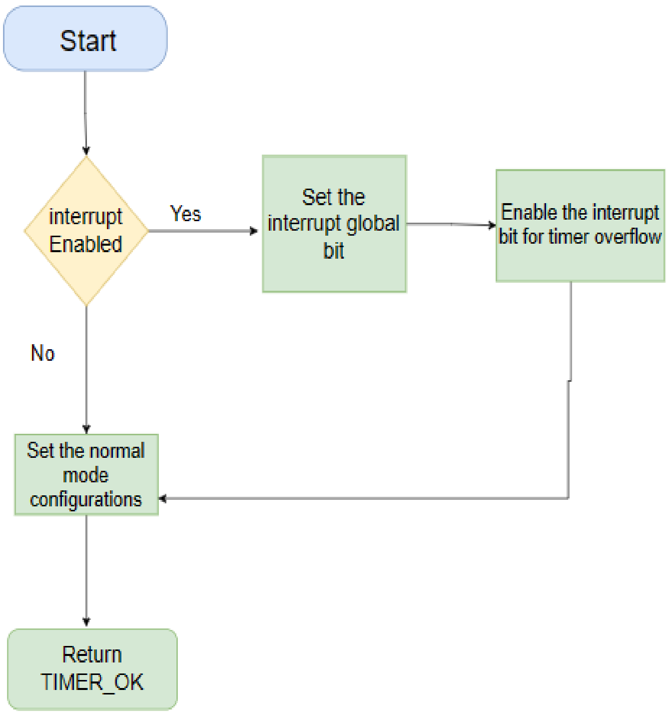
```
| This function initializes various MCAL and HAL components.
void APP_initialization(void);

| The function starts the program and checks for data in memory before switching
| between programmer mode, user mode, and check mode based on the current app mode.
void APP_startProgram(void);
```

3. Low Level Design

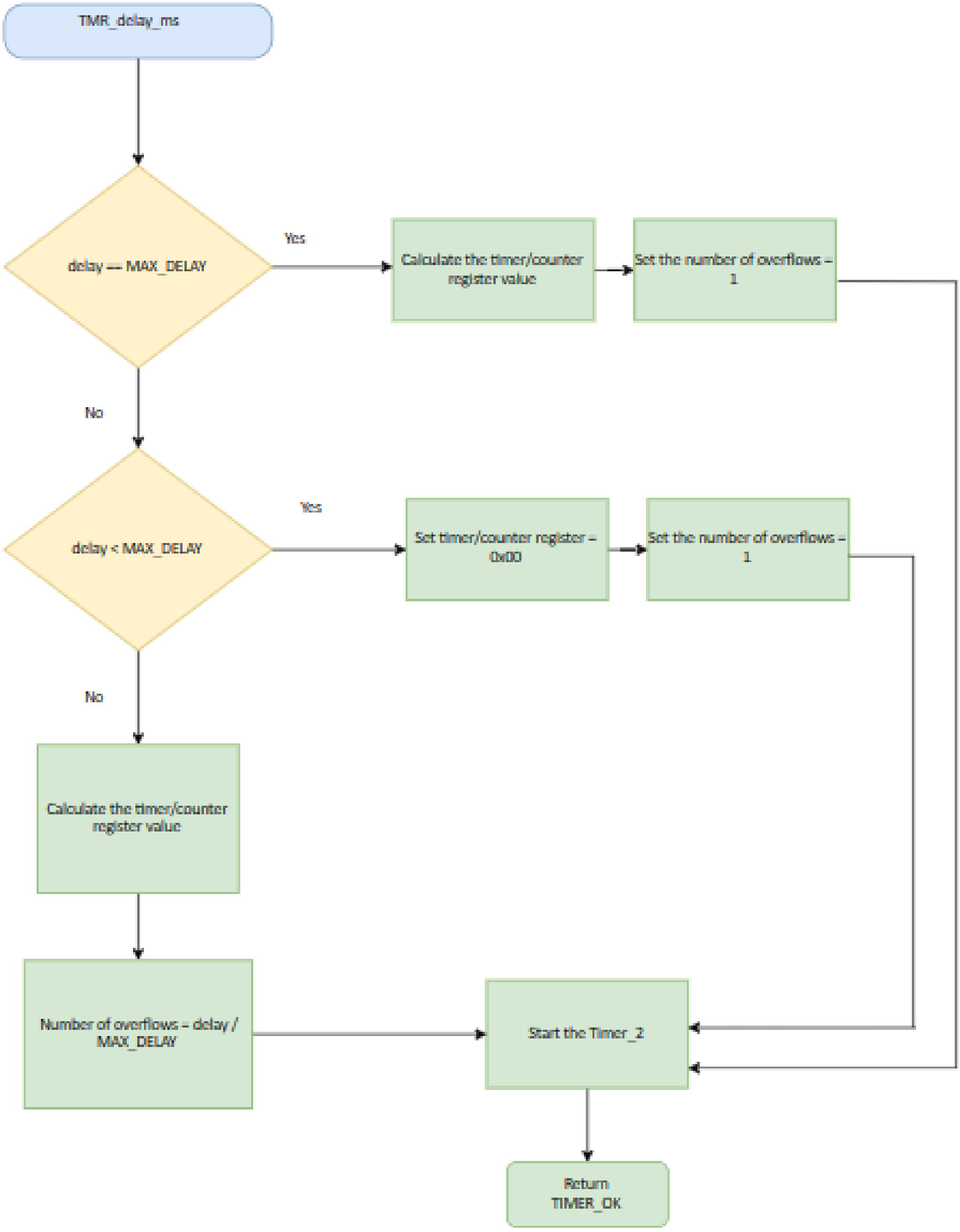
3.1. MCAL Layer

3.1.1. Timer Module

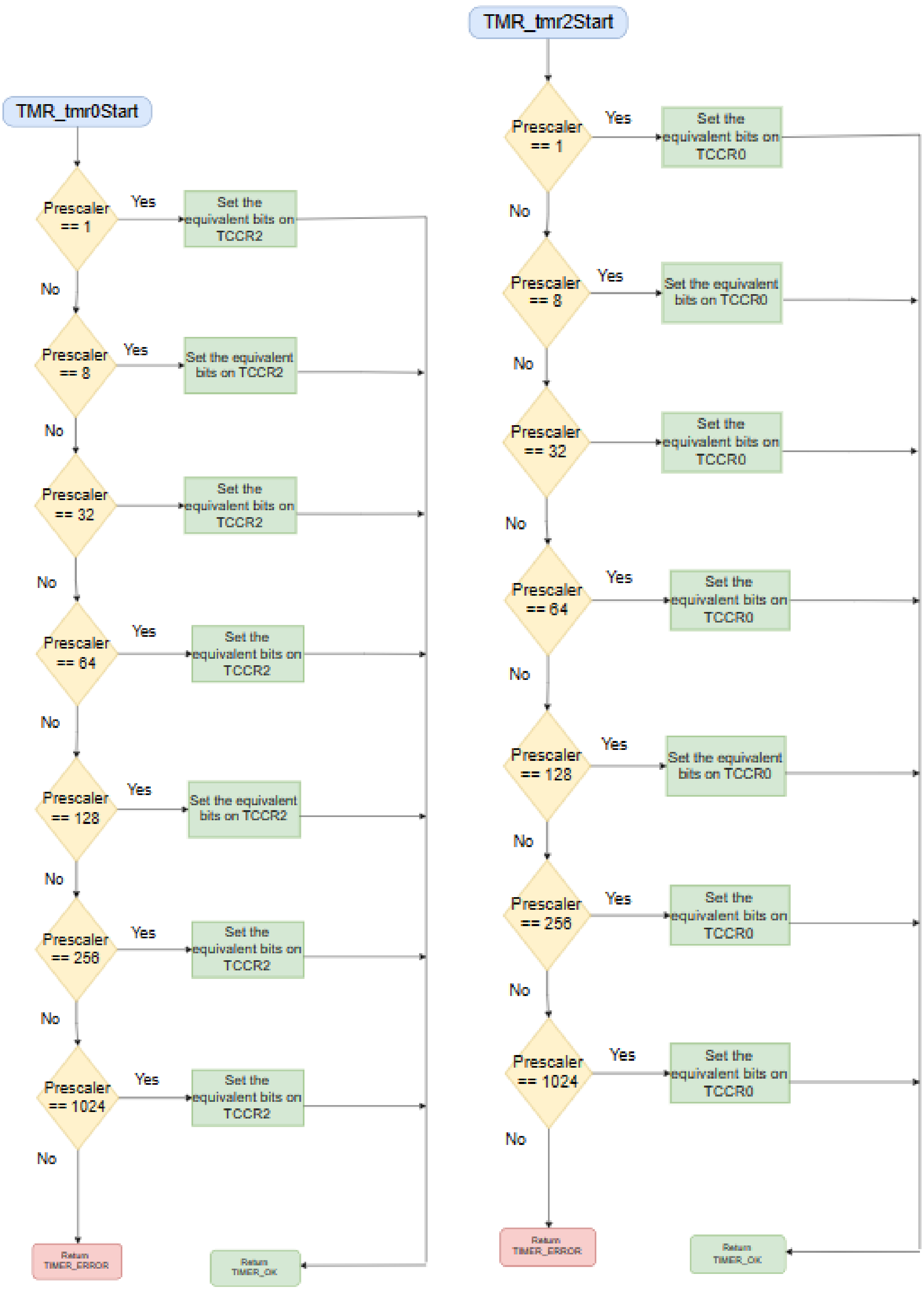
3.1.1.1. TMR_tmr0NormalModeInit / TMR_tmr2NormalModeInit



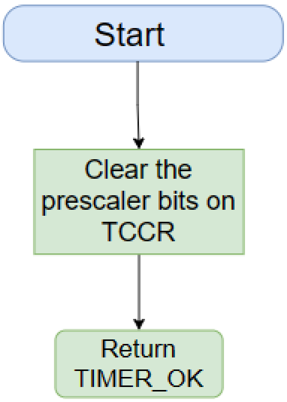
3.1.1.2. TMR_tmr0Delay / TMR_tmr2Delay



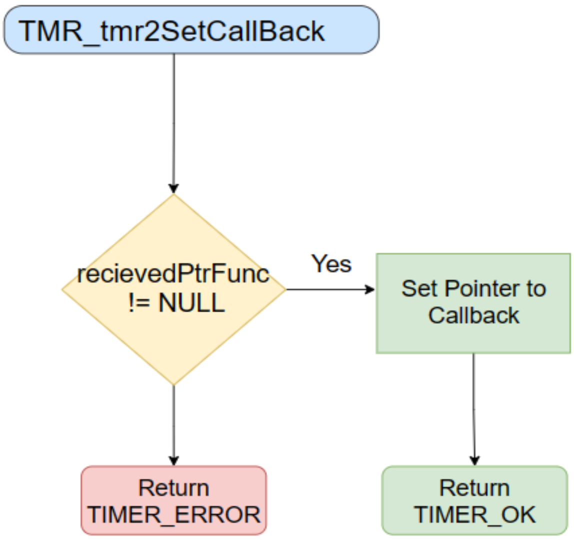
3.1.1.3. TMR_tmr0Start / TMR_tmr2Start



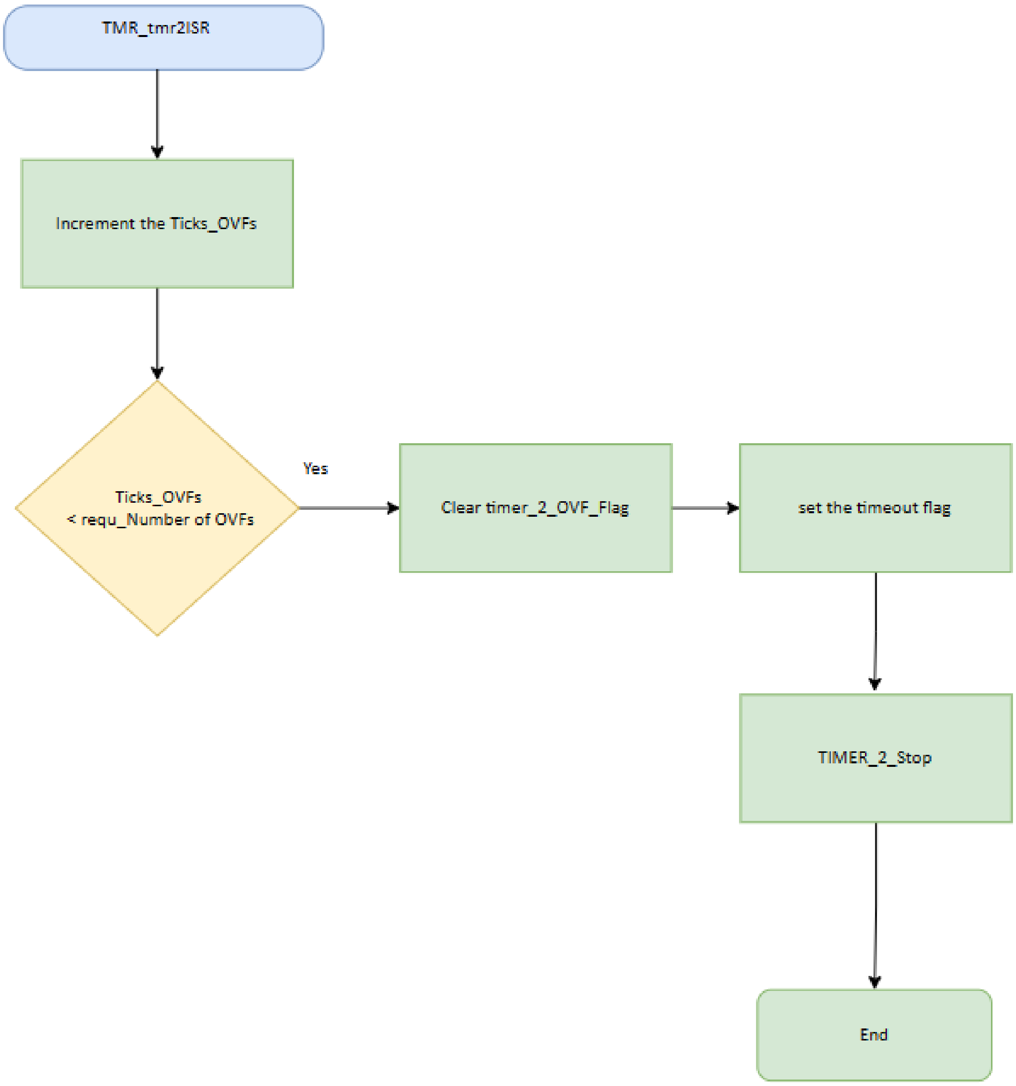
3.1.1.4. TMR_tmr0Stop / TMR_tmr2Stop



3.1.1.5. TMR_ovfSetCallback



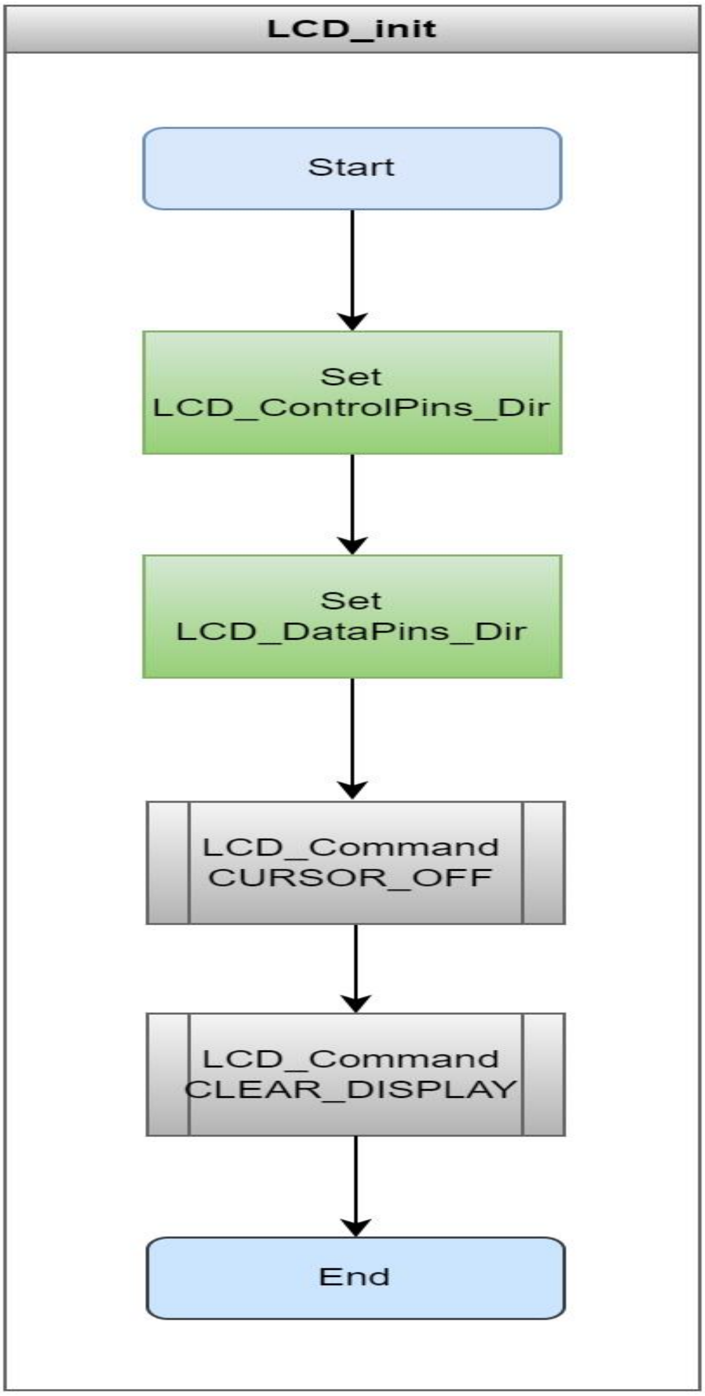
3.1.1.6. TMR2_ovfVect



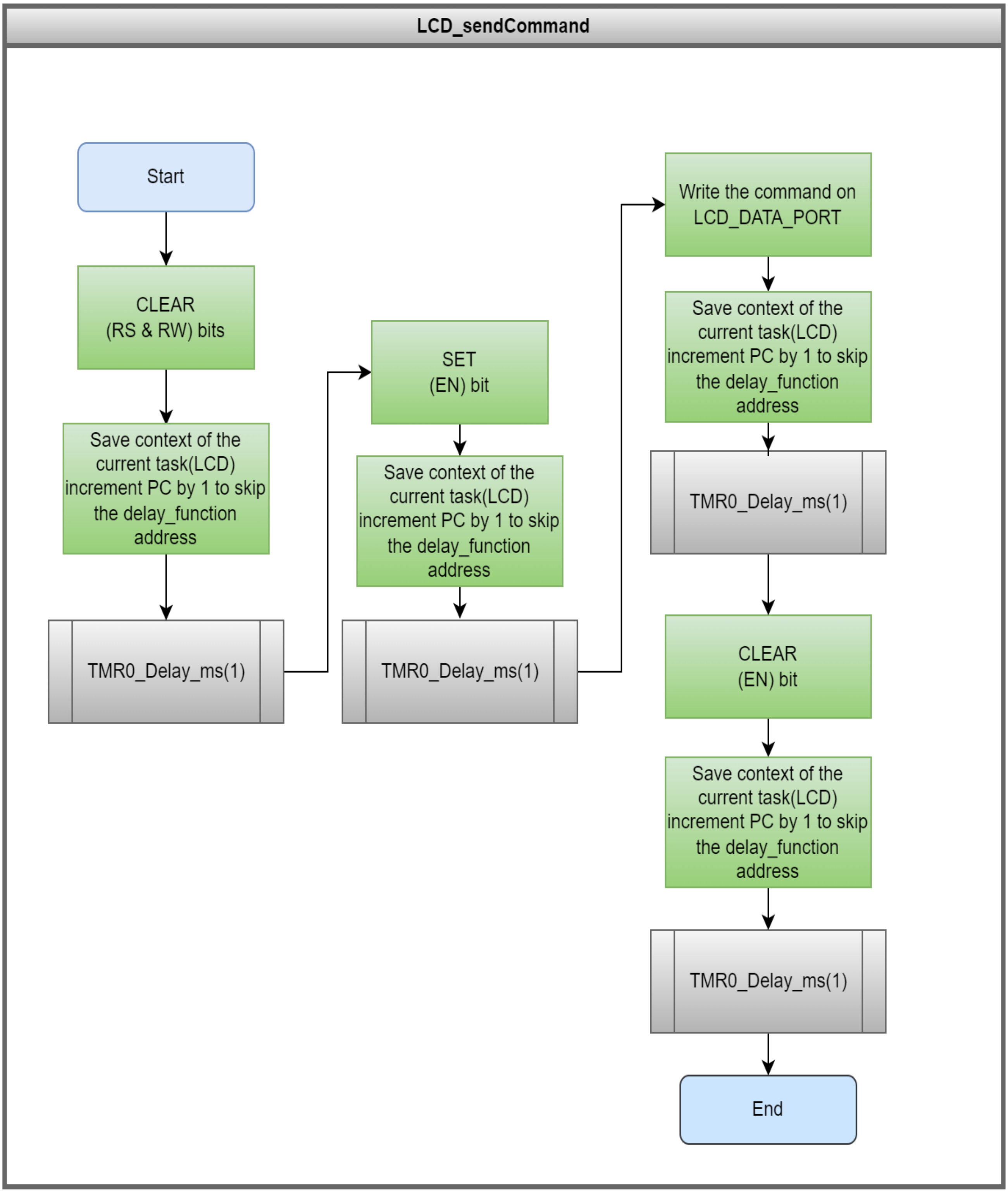
3.2. HAL Layer

3.2.1. LCD Module

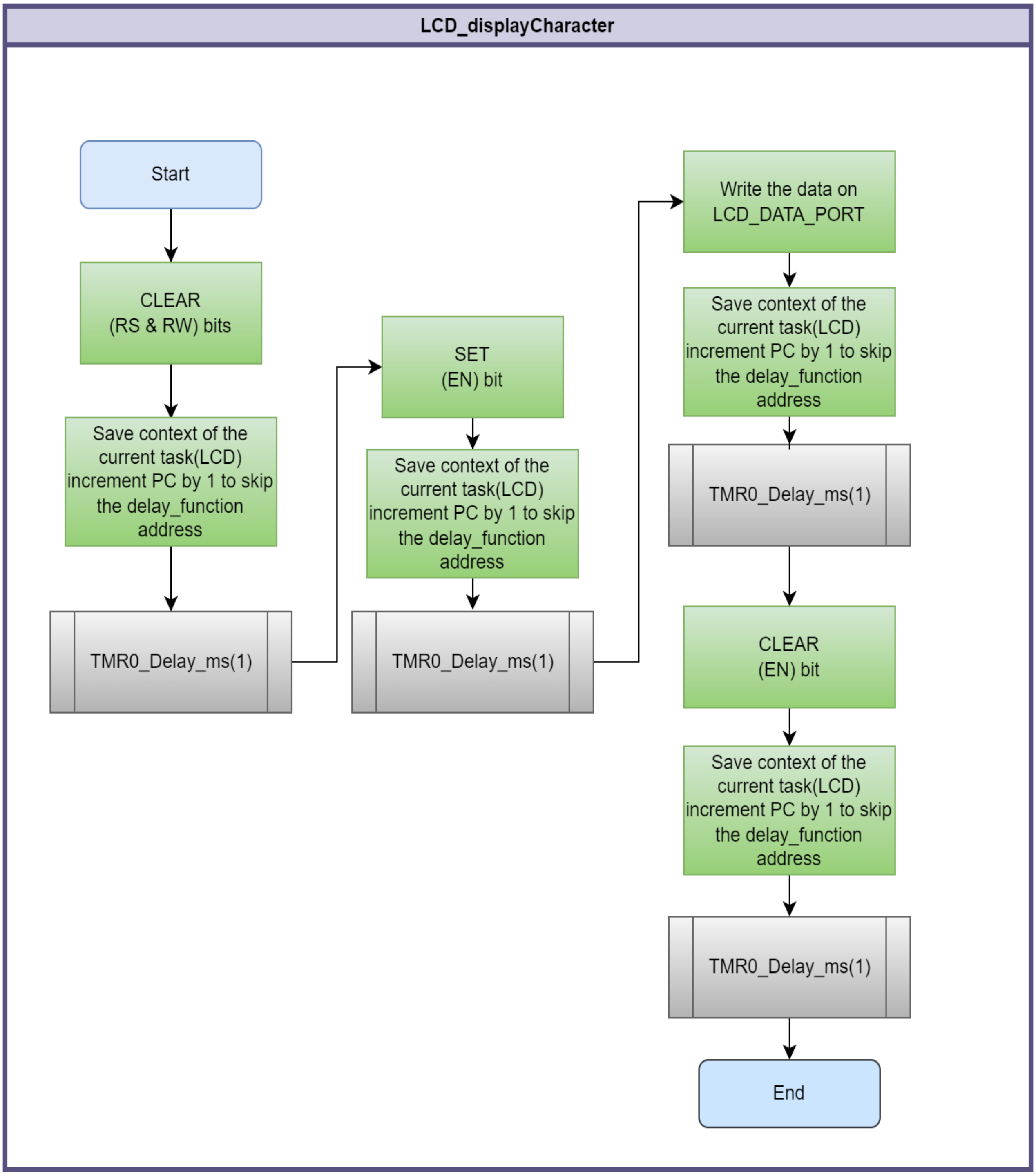
3.2.1.1. LCD_init



3.2.1.2. LCD_sendCommand

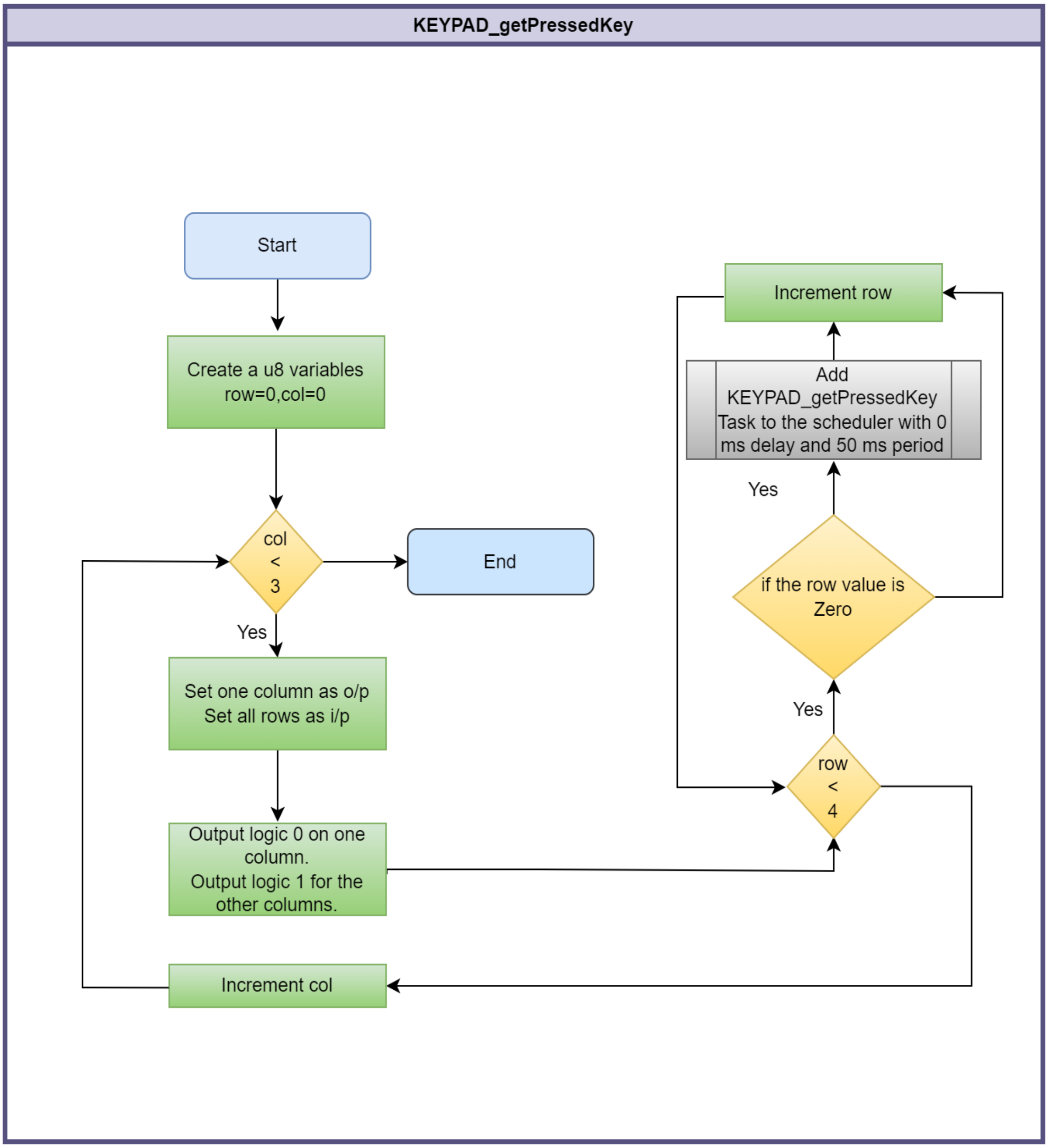


3.2.1.3. LCD_displayCharacter



3.2.2. KPD Module

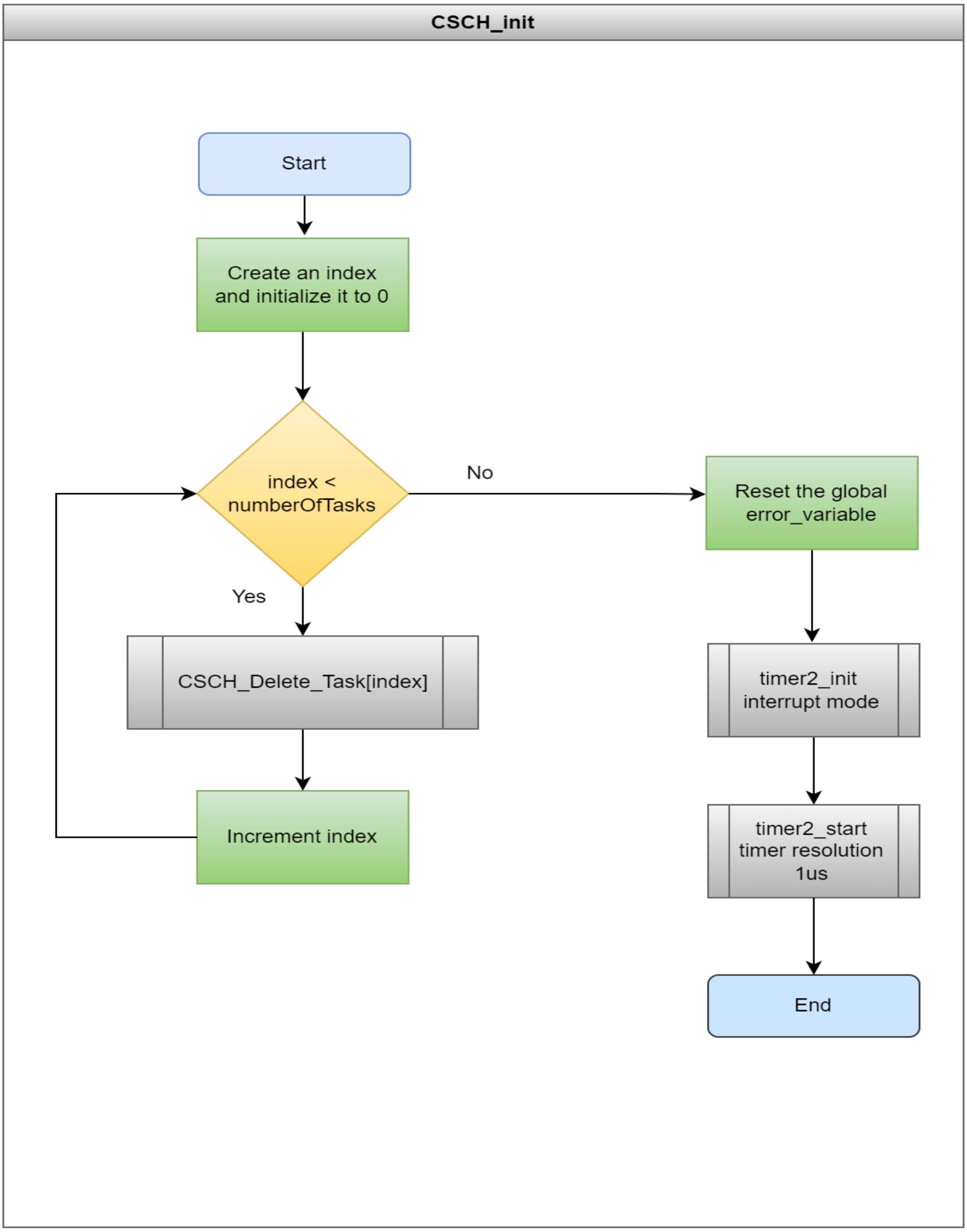
3.2.2.1. KPD_getPressedKey



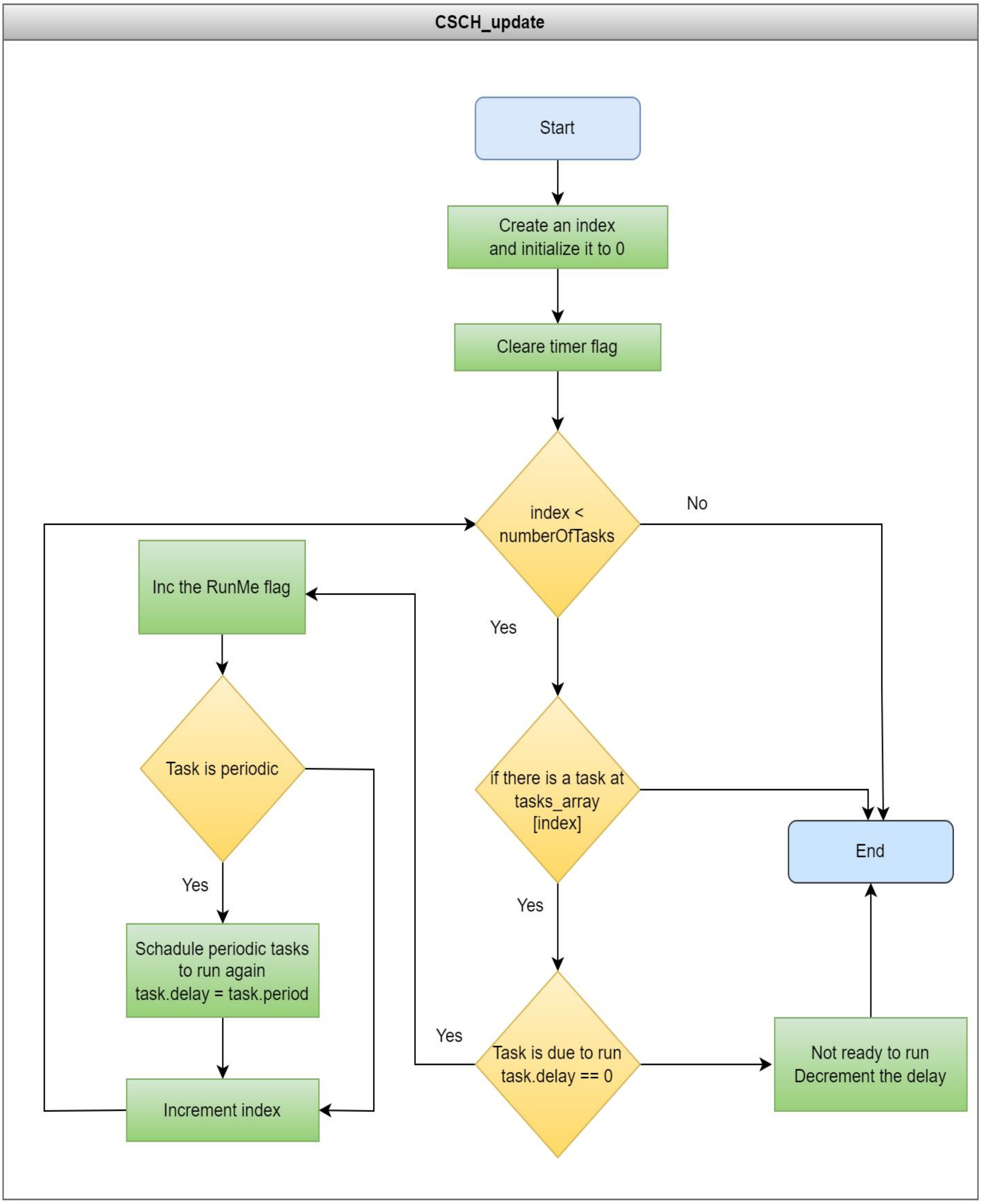
3.3. Service Layer

3.3.1. Scheduler Module

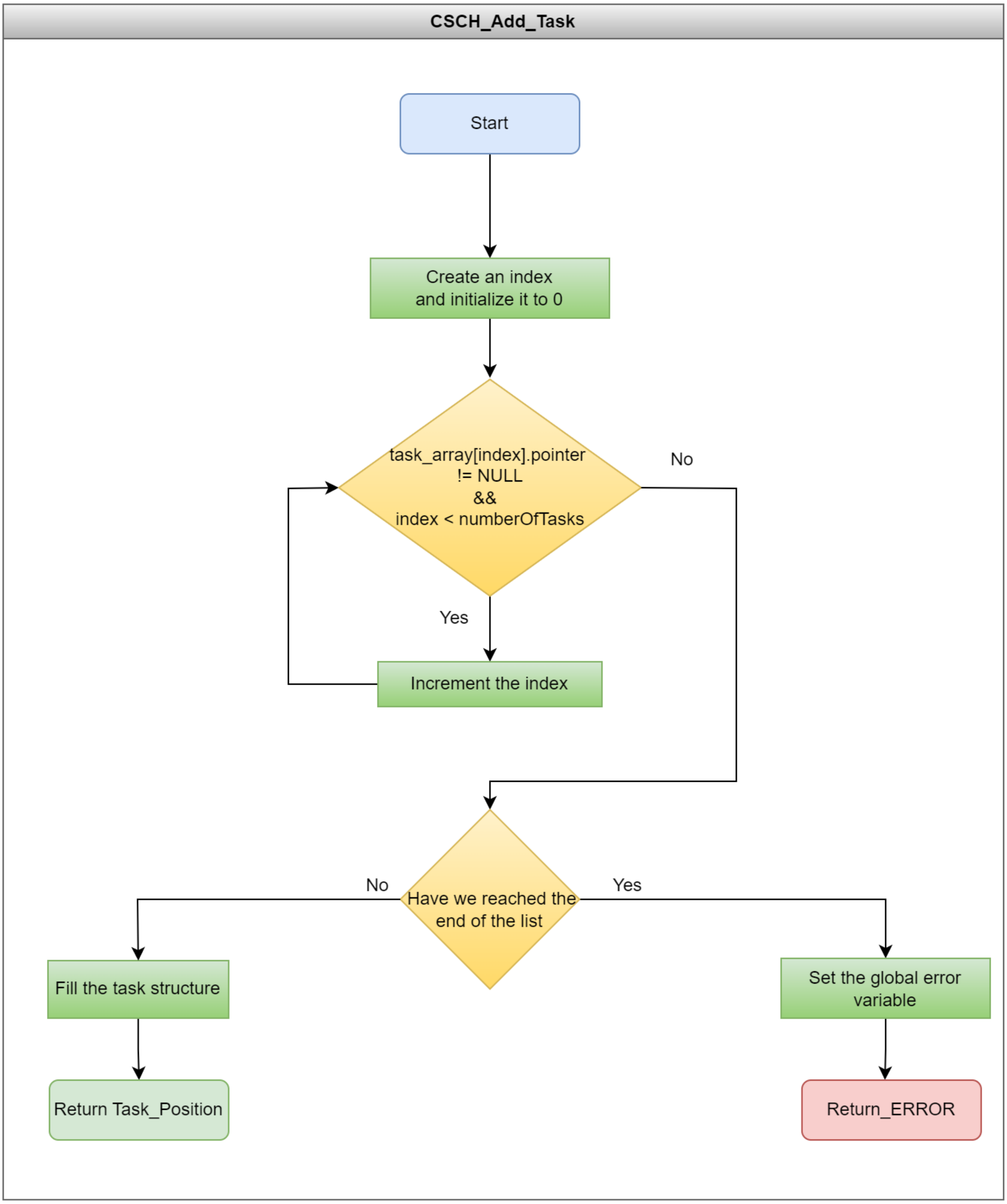
3.3.1.1 CSCH_init



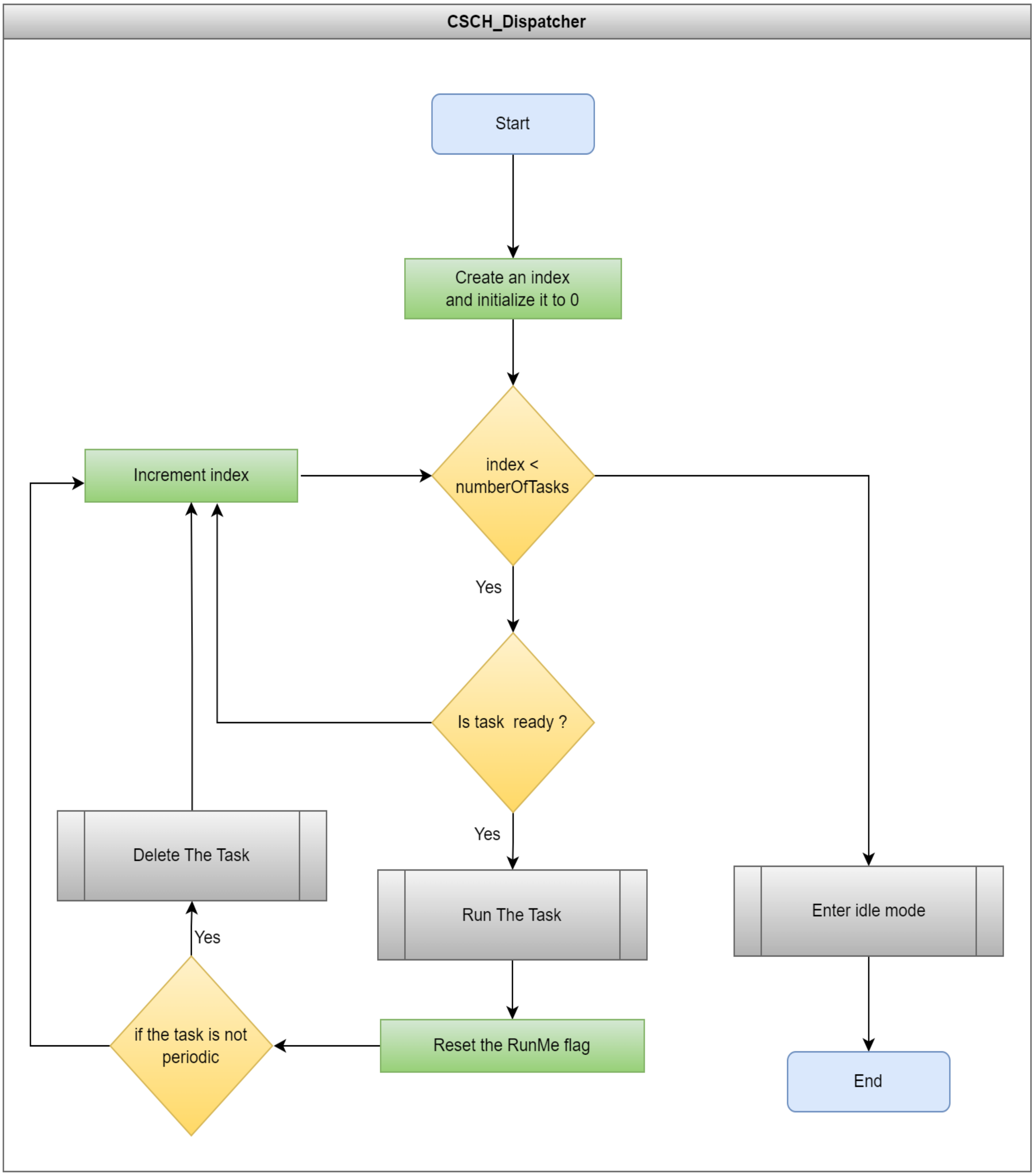
3.3.1.2 CSCH_update



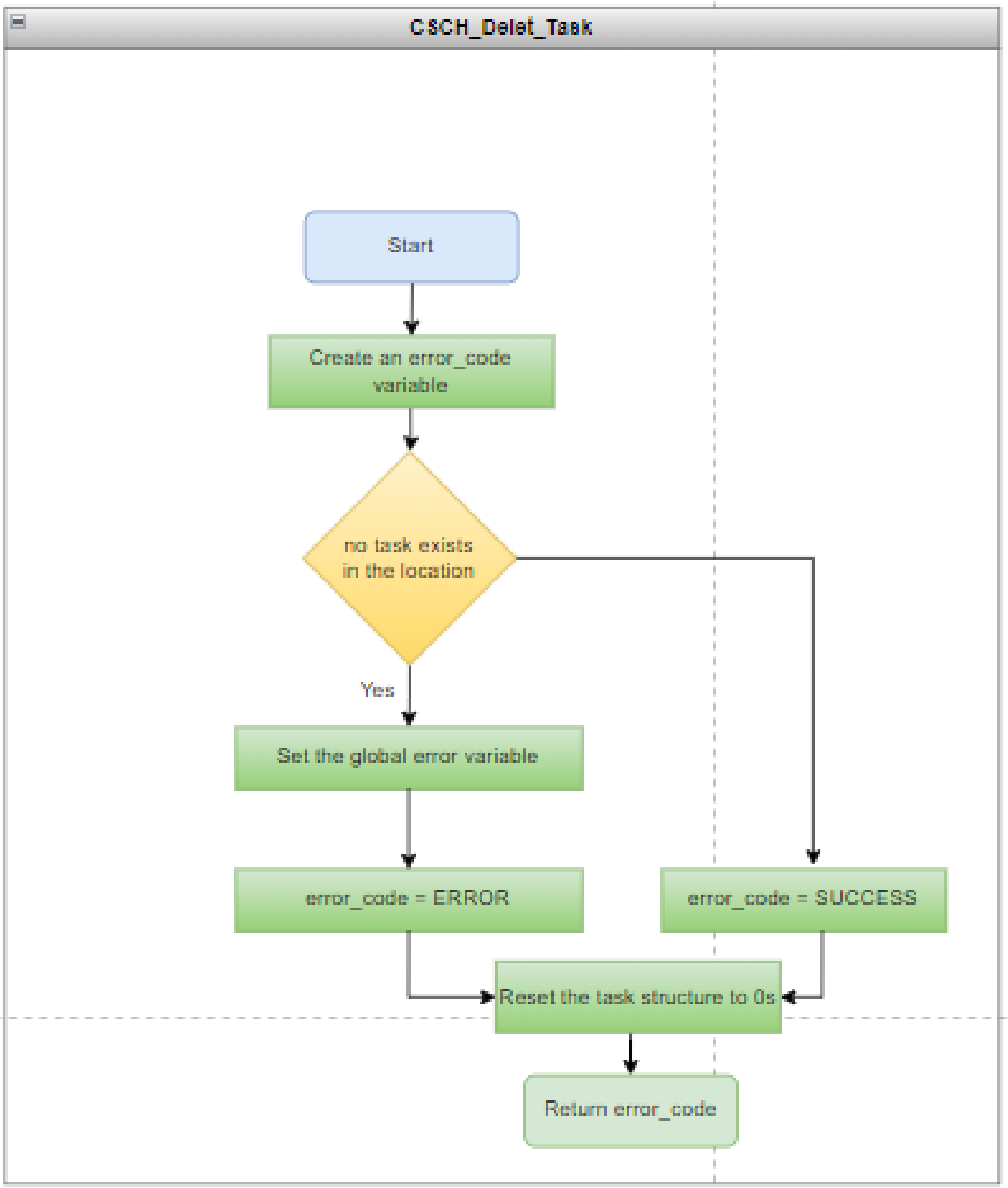
3.3.1.3 CSCH_addTask



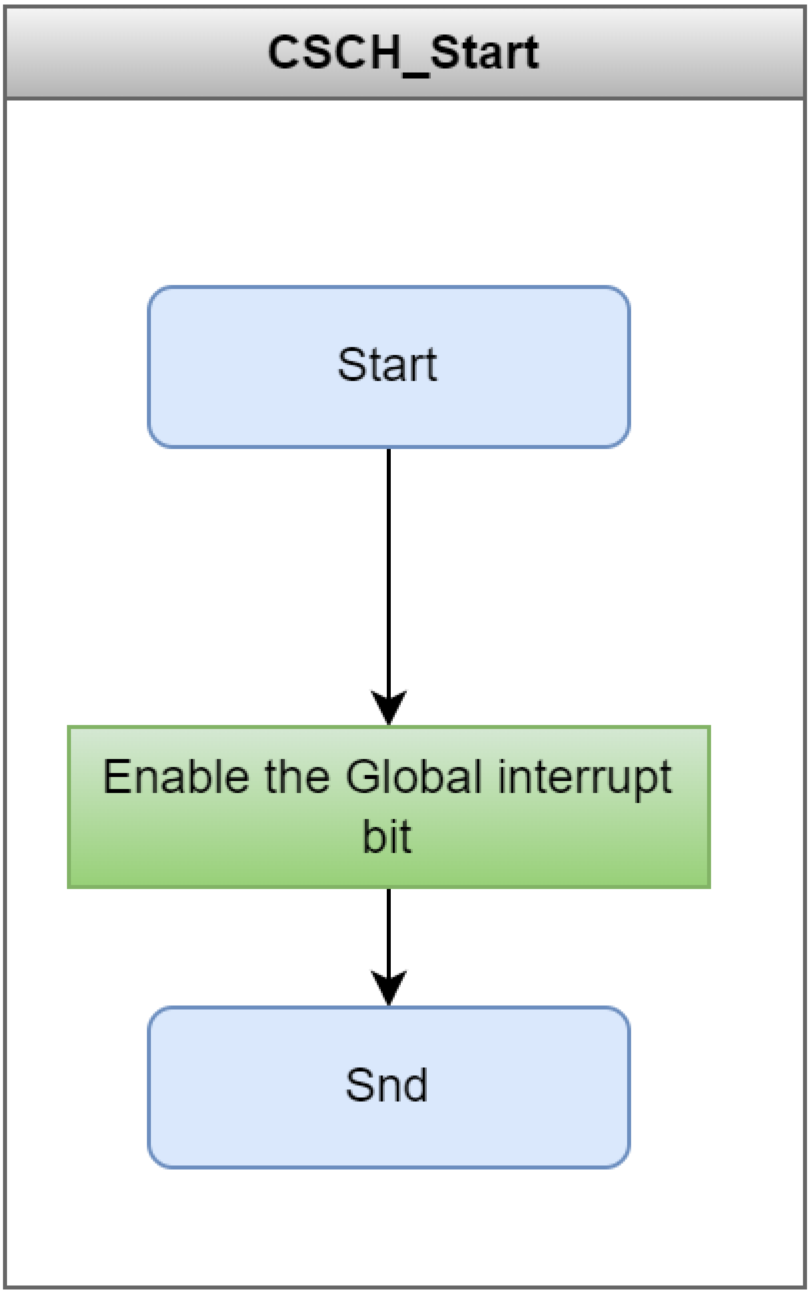
3.3.1.4 CSCH_dispatcher



3.3.1.5 CSCH_deleteTask

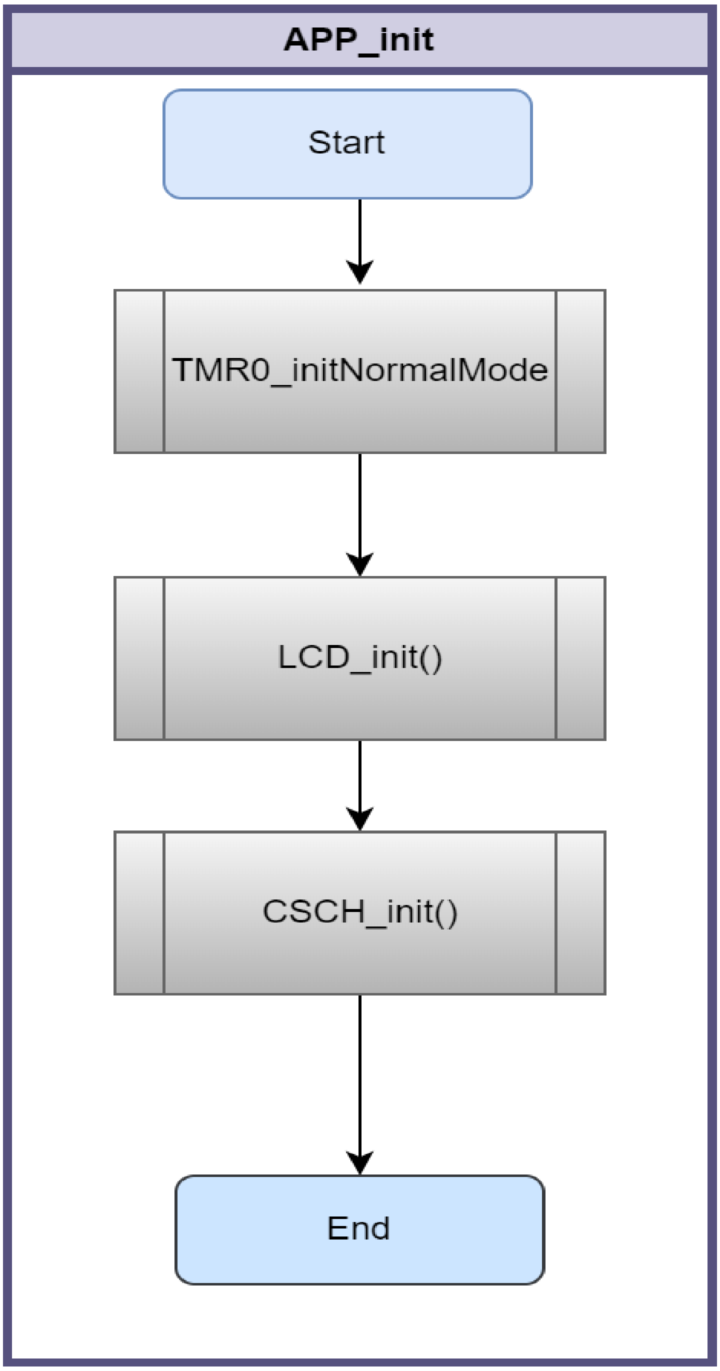


3.3.1.6 CSCH_start



3.3. APP Layer

3.3.1 APP_init



3.3.2 APP_start