STM32G491RE: MoT Peripheral Integration Maintainer Document

Author: Andreas Tzitzikas
Version: 1.0
Release Date: 05/19/2025

Introduction

This document provides a comprehensive technical overview of the ENEE440S25 Project, centered around the STM32G491 microcontroller and its integration with the Micro on Tether (MoT) system. It is intended for engineers and developers who may be involved in the future maintenance, modification, or extension of this project.

The following sections detail the system architecture, hardware driver implementations, software design choices, build processes, and testing methodologies. Diagrams, flowcharts, and code structure explanations are included to facilitate a thorough understanding of the project's components and their interactions. The primary goal is to equip maintainers with the necessary knowledge to effectively service and evolve the system.

Table of Contents

Device 3: General Purpose Input/Output (GPIO)

Introduction

Device 3 is responsible for managing General Purpose Input/Output (GPIO) operations within the STM32G491 Micro on Tether (MoT) system. Its primary focus is on controlling and monitoring the state of GPIO pin PA8. Device 3 offers a comprehensive suite of functionalities, including:

- **Digital Output:** Setting PA8 to high or low logic levels, either immediately or after a specified delay.
- **Repetitive Output:** Generating a square wave signal on PA8 with configurable cycle count, on-time, and off-time.
- **Digital Input:** Reading the current logic level of PA8, either immediately or after a delay.
- **Monitored Input:** Continuously monitoring PA8 for changes in its logic state and reporting these changes.
- **External Interrupt (EXTI):** Utilizing PA1 as an external interrupt trigger to initiate a read of PA8's state.

Hardware and Pin Configuration

Device 3 commands call the appropriate PA8_input_init or PA8_output_init before performing an operation to ensure the pin is in the correct mode. The device3_initialize command (Cmd 0x00) calls PA8_init, which sets PA8 to output. However, the device3_demo.txt notes that after initialization, PA8 is configured as input by default. This implies that either the default MoT task for Device 3 or the first input-related command reconfigures it using PA8_input_init.

- **Primary GPIO Pin:** PA8 (Port A, Pin 8) is the main pin for input and output operations.
- **Auxiliary GPIO Pin (EXTI):** PA1 (Port A, Pin 1) is used as an external interrupt input.
- **Clock Enablement:** The GPIOA peripheral clock is enabled via the RCC_AHB2ENR register. For EXTI functionality, the SYSCFG clock is also enabled via RCC_APB2ENR.
- **Pin Initialization:**
    - **PA8_init:** This function performs a general initialization for PA8, configuring it as a general-purpose output, push-pull, with no pull-up or pull-down resistors.
    - **PA8_input_init:** Configures PA8 specifically for input mode, with no pull-up/pull-down.
    - **PA8_output_init:** Configures PA8 specifically for output mode, push-pull, with no pull-up/pull-down.
    - **EXT1_init:** Configures PA1 as an input with a pull-down resistor for EXTI functionality.

Core Functionalities and Implementation

Device 3's functionalities are dispatched via device3_cmdHandler located in device3.S. This handler uses a jump table (device3_cmds) to execute specific command routines.

Initialization

- **Command:** 0x00
- **File:** device3.S

- **Action:** Calls PA8_init to initialize PA8 (as output). Posts an initialization message: device3 has been initialized.
- **Registers:** Modifies GPIOA_MODER, OTYPER, OSPEEDR, OPUPDR via PA8_init. Enables GPIOA clock via RCC_AHB2ENR.

Output Operation

These functions call PA8_output_init before manipulating the pin state.

- **Set PA8 Output Low (Immediate)**
  - **Command:** 0x01 (device_set_low)
  - **File:** device3.S
  - **Action:** Calls PA8_output_init, then PA8_to_low (writes to GPIOA_BSRR to clear PA8). Posts Input is LOW (Note: message might be intended as Output is LOW).
- **Set PA8 Output High (Immediate)**
  - **Command:** 0x02 (device_set_high)
  - **File:** device3.S
  - **Action:** Calls PA8_output_init, then PA8_to_high (writes to GPIOA_BSRR to set PA8). Posts Input is HIGH (Note: message might be intended as Output is HIGH).
- **Set PA8 Output State with Delay**
  - **Command:** 0x03 (device3_scheduled_output)
  - **File:** device3.S
  - **Action:** Calls PA8_output_init. Retrieves a 16-bit delay and a 16-bit state (0 for LOW, non-0 for HIGH) from the command payload using arg_retrieve_half. Calls delay (from Utils.S). Then calls PA8_to_high or PA8_to_low. Posts device3 scheduled a task on.
- **Set PA8 Repetitive Output (Square Wave)**
  - **Command:** 0x04 (device3__repetitive)
  - **File:** device3.S
  - **Action:** Calls PA8_output_init. Retrieves a 16-bit cycle count, 16-bit T_ON, and 16-bit T_OFF from payload using arg_retrieve_half. Stores T_ON/T_OFF in device-specific variables device3_T_ON and device3_T_OFF. Loops for the specified number of cycles, calling delay with T_OFF, PA8_to_high, delay with T_ON, and PA8_to_low. Posts device3 repetitive mode is on.

Input Operation

These functions call PA8_input_init before reading the pin state.

- **Read Immediate Input from PA8**
  - **Command:** 0x05 (device3_input_read)
  - **File:** device3.S, GPIO_input.S
  - **Action:** Calls PA8_input_init, then PA8_read. PA8_read reads GPIOA_IDR and isolates the bit for PA8. Posts Input is HIGH or Input is LOW based on the state.
- **Read Scheduled Input from PA8 (Immediate)**
  - **Command:** 0x06 (device_scheduled_read)
  - **File:** device3.S, GPIO_input.S, Utils.S

    - **Action:** Calls PA8_input_init. Retrieves a 16-bit delay from payload using arg_retrieve_half. Posts Waiting on PA8. Calls delay. Calls PA8_read. Posts Input is HIGH or Input is LOW.
- **Start Monitoring Input on PA8**
    - **Command:** 0x07 (device3_monitored)
    - **File:** device3.S, GPIO_input.S
    - **Action:** Calls PA8_input_init. Reads the initial state of PA8 using PA8_read and stores it in the device-specific variable device3_PREVIOUS. Schedules the monitor_task using MoT_taskUpdate. Posts Monitor on PA8.
    - **monitor_task:** This MoT task periodically executes. It calls delay (100 units), then PA8_input_init and PA8_read to get the current state. It compares the current state with device3_PREVIOUS. If a change is detected, it posts Change on PA8 followed by the new state (Input is HIGH or Input is LOW) and updates device3_PREVIOUS. The task then reschedules itself to continue monitoring.
- **Read and Stop Monitor on PA8**
    - **Command:** 0x09 (device3_stop_monitor)
    - **File:** device3.S
    - **Action:** Updates the current task for Device 3 to device3_skiptask using MoT_taskUpdate, effectively stopping the monitor_task. Posts Monitor Has Stopped. The demo output suggests this command also reports any changes detected *during* the last monitoring cycle before stopping, implying the monitor_task might post its final findings before being fully replaced.

EXTI (External Interrupt) Operation

- **Initialize EXTI on PA1 for Device 3**
    - **Command:** 0x08
    - **File:** device3.S, EXTI1.S
    - **Action:** Calls PA8_input_init (to ensure PA8 can be read). Sets a global variable trigger_state to 8, which the EXTI1_IRQHandler uses to identify that Device 3 is the context for this EXTI event. Calls EXTI1_init to configure PA1 as an EXTI line with a rising edge trigger and enable its NVIC interrupt. Posts Trigger on PA1.
- **Read EXTI Trigger Results for PA8**
    - **Command:** 0x0A
    - **File:** device3.S
    - **Action:** Posts the message Result from Trigger. This command doesn't actively fetch new data but serves as an acknowledgment or a way for the user/PC script to confirm that trigger events (and their corresponding PA8 state reports via the IRQ) have occurred. The actual results are posted asynchronously by the EXTI1_IRQHandler through calls to device3_input_read.

MoT System Integration

- **Command Handling:** device3_cmdHandler in device3.S is the entry point for all Device 3 commands. It uses a Table Branch Byte (tbb) instruction with the device3_cmds jump table for efficient dispatch.
- **Device Core:** The MoT_core_m device3, device3_cmdHandler, device3_skiptask macro instantiation in device3.S defines the core MoT structure for this device, linking its command handler and default task (device3_skiptask).
- **Task Managerment**

    - device3_skiptask: A minimal task that does nothing but pass control to the next device in the MoT task list. This is the default active task.
    - monitor_task: A periodic task scheduled by command 0x07 to monitor PA8. It uses MoT_taskUpdate to reschedule itself.
    - Command 0x09 (device3_stop_monitor) uses MoT_taskUpdate to switch the active task back to device3_skiptask.
- **Messaging:** Device 3 uses MoT_msgPost to send status and data messages to the consoleMsgs queue. Message link structures (e.g., device3_initmsg, device3_input_highmsg) are defined in the .data section of device3.S.

Register Level Details

- **RCC (Reset and Clock Control):**
    - RCC_AHB2ENR: To enable the clock for GPIOA.
    - RCC_APB2ENR: To enable the clock for SYSCFG (for EXTI).
- **GPIOA (General Purpose I/O Port A):**
    - GPIOA_MODER: Configures pin PA8 and PA1 mode (Input, Output, Analog, Alternate Function).
    - GPIOA_OTYPER: Configures PA8 output type (Push-pull).
    - GPIOA_OSPEEDR: Configures PA8 output speed.
    - GPIOA_PUPDR: Configures PA8 and PA1 pull-up/pull-down resistors.
    - GPIOA_IDR: Reads the input state of PA8.
    - GPIOA_BSRR: Atomically sets or resets PA8 output state.
- **SYSCFG (System Configuration Controller):**
    - SYSCFG_EXTICR1: Selects the source input (PA1) for EXTI Line 1.
- **EXTI (External Interrupt/Event Controller):**
    - EXTI_IMR1: Interrupt Mask Register for Line 1 (unmask to enable).
    - EXTI_RTSR1: Rising Trigger Selection Register for Line 1 (enable rising edge).
    - EXTI_PR1: Pending Register for Line 1 (to clear pending interrupt flags).
- **NVIC (Nested Vectored Interrupt Controller):**
    - NVIC_ISER0: Interrupt Set-Enable Register to enable EXTI1_IRQn.
    - NVIC_IPRx: Interrupt Priority Register (implicitly configured, priority not explicitly detailed in assembly but set by EXTI1_init).

Control Flow

- **Command Execution:**
    - A MoT command string is received by the system.
    - The MoT core dispatches to device3_cmdHandler based on Device ID 0x03.
    - device3_cmdHandler uses the command code (second byte of the command) to jump to the specific command implementation function (e.g., device3_set_low, device3_monitored).
    - The command function executes, interacting with GPIO/EXTI registers directly or via helper functions in GPIO_*.S and EXTI1.S.
    - Utility functions from Utils.S (e.g., delay, arg_retrieve_half) are used for timing and argument parsing.
    - Status messages are posted to consoleMsgs via MoT_msgPost.
- **Monitored Input Task Flow:**
    - device3_monitored command schedules monitor_task.
    - monitor_task runs periodically as part of the MoT task list.

- Inside monitor_task: reads PA8, compares to previous state, posts messages on change, and calls MoT_taskUpdate to reschedule itself.

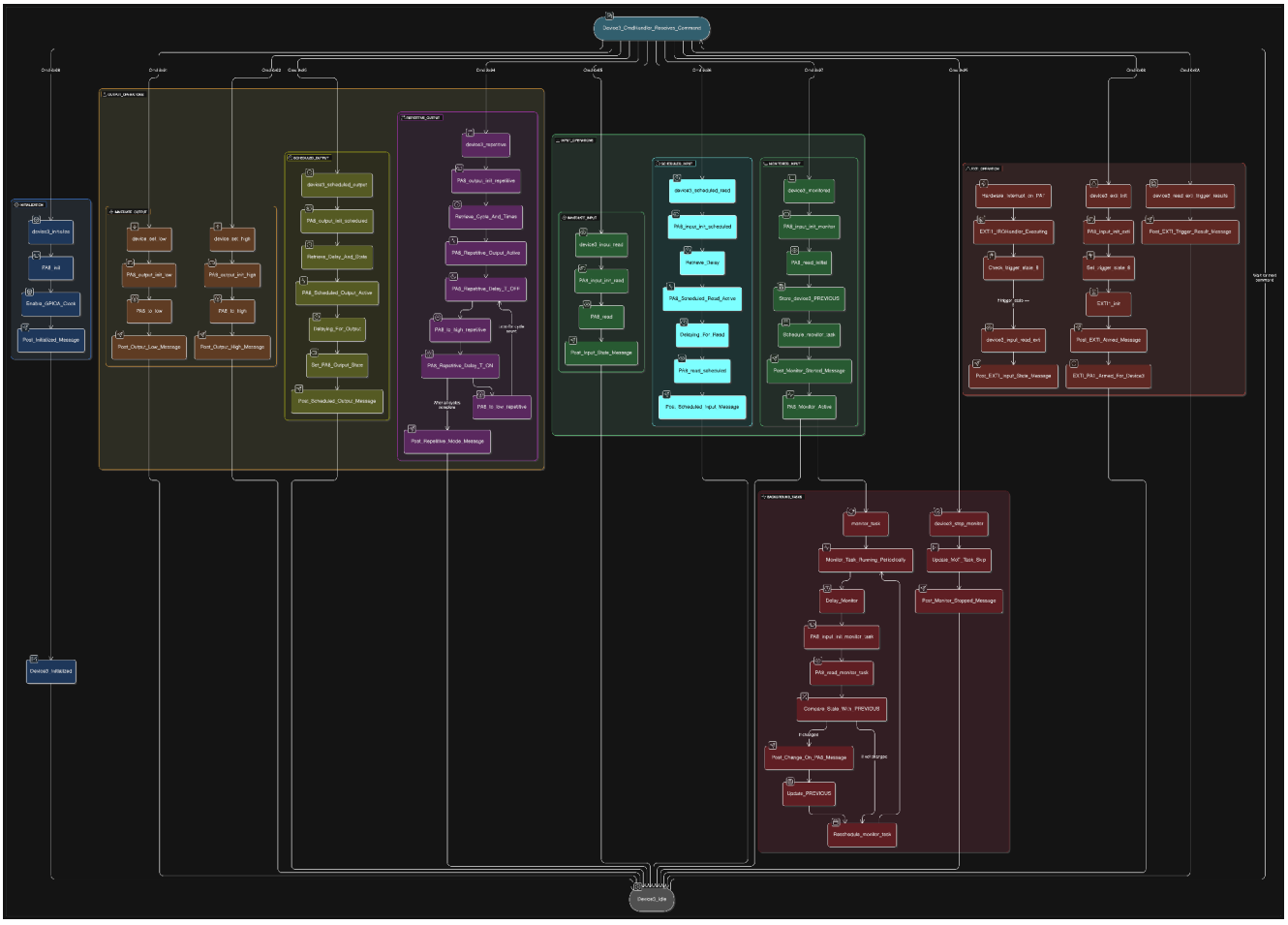
- **EXTI Flow:**
  - device3_trig command configures PA1 as EXTI1 source and enables the interrupt.
  - A rising edge on PA1 triggers the EXTI1_IRQHandler.
  - EXTI1_IRQHandler identifies Device 3 as the context (via trigger_state == 8).
  - It then calls device3_input_read, which reads PA8 and posts its state to the console.

Device Data Storage

Device 3 utilizes device-specific storage allocated by the MoT framework for the following variables:

- **device3_T_OFF and device3_T_ON:** Store the off-time and on-time durations for the repetitive output command (Cmd 0x04). Accessed via rDEVP + offset.
- **device3_PREVIOUS:** Stores the previous state of PA8 for the input monitoring function (Cmd 0x07 and monitor_task). Accessed via rDEVP + offset.

State Diagram



Device 4: Digital-to-Annalog Converter (DAC)

Introduction

Device 4 provides an interface to the STM32G491's Digital-to-Analog Converter (DAC), specifically utilizing DAC1 Channel 2, which outputs on GPIO pin PA5. This device allows for the generation of analog voltage levels based on digital input values. Key functionalities include:

- **Initialization:** Setting up the DAC peripheral and the associated GPIO pin (PA5).
- **Set Constant Output:** Configuring the DAC to output a specific, constant analog voltage. The input is a 12-bit digital value (0x000 to 0xFFF).
- **Periodic Signal Generation:** Producing a periodic waveform by alternating between two voltage levels (VA and VB) with specified time durations (T1 for VA, T2 for VB). This is achieved using SysTick for timing and repeatedly calling the DAC output function.
- **EXTI Triggered Output:** Configuring an external interrupt on PA1 to trigger the DAC to output a pre-defined voltage level.

Hardware and Pin Configuration

- **Primary DAC Output Pin:** PA5 (Port A, Pin 5), connected to DAC1_OUT2.
- **Auxiliary GPIO Pin (EXTI):** PA1 (Port A, Pin 1) is used as an external interrupt input.
- **Clock Enablement:**
    - GPIOA clock is enabled via RCC_AHB2ENR for PA5 and PA1.
    - DAC1 clock is enabled via RCC_AHB2ENR.
    - SYSCFG clock is enabled via RCC_APB2ENR for EXTI functionality.
- **Pin Initialization:**
    - PA5 is configured to Analog mode (GPIOA_MODER).
    - PA5 pull-up/pull-down is disabled (GPIOA_PUPDR).
- **DAC Initialization:**
    - DAC1 peripheral is reset and then enabled.
    - DAC1 Channel 2 is enabled (DAC_CR_EN2).
    - Software trigger for Channel 2 is enabled (DAC_CR_TEN2).
    - Output buffer for Channel 2 is enabled (default, DAC_MCR_MODE2 = 000).

Core Functionalities and Implementation

Device 4 commands are dispatched by device4_cmdHandler in device4.S.

Initialization

- **Command:** 0x00
- **File:** device4.S
- **Action:** Calls DAC_init to configure PA5 and DAC1 Channel 2. Posts device4 has been initialized.
- **Registers:** Modifies RCC_AHB2ENR, GPIOA_MODER, GPIOA_PUPDR, DAC_CR, DAC_MCR.

Output Operation

- **Set DAC Output (Constant Voltage)**
  - **Command:** 0x01 (device4_set_output)
  - **File:** device4.S, Utils.S
  - **Action:** Retrieves a 12-bit value from the command payload using arg_retrieve_half (and swap_low2bytes_r1). Writes this value to DAC_DHR12R2 (DAC1 Channel 2 12-bit Right-aligned Data Holding Register). Triggers a software conversion using DAC_SWTRIGR_SWTRIG2. Posts device4 set to constant voltage.
  - **Registers:** DAC_DHR12R2, DAC_SWTRIGR.
- **Start Periodic Signal**
  - **Command:** 0x02 (device4_periodic_signal)
  - **File:** device4.S, Utils.S
  - **Action:**
    1. Retrieves VA (12-bit), T1 (16-bit), VB (12-bit), T2 (16-bit) from payload using arg_retrieve_half and swap_low2bytes_r1.
    2. Stores these values in device-specific variables: device4_VA, device4_T1, device4_VB, device4_T2.
    3. Schedules the periodic_task_VA using MoT_taskUpdate to start the waveform.
    4. Posts Voltage transitioned (or a similar message indicating the start of the periodic signal).
  - **periodic_task_VA (device4.S):**
    1. Outputs device4_VA to DAC1_CH2.
    2. Calls delay with device4_T1.
    3. Updates the task to periodic_task_VB using MoT_taskUpdate.
    4. Passes control to the next MoT task.
  - **periodic_task_VB (device4.S):**
    1. Outputs device4_VB to DAC1_CH2.
    2. Calls delay with device4_T2.
    3. Updates the task back to periodic_task_VA using MoT_taskUpdate.
    4. Passes control to the next MoT task.
  - **Registers:** DAC_DHR12R2, DAC_SWTRIGR. MoT task registers.
- **Stop Periodic Signal**
  - **Command:** 0x03 (device4_stop_periodic)
  - **File:** device4.S
  - **Action:** Updates the current task for Device 4 to device4_skiptask using MoT_taskUpdate, stopping the periodic waveform generation. Posts device4 has stopped outputing a periodic signal.
  - **Registers:** MoT task registers.

EXTI (External Interrupt) Operation

- **Initialize EXTI on PA1 for Device 4**
  - **Command:** 0x04 (device4_exti_init)
  - **File:** device4.S, EXTI1.S, Utils.S
  - **Action:**
    1. Retrieves a 12-bit voltage value (V_trig) from the payload using arg_retrieve_half. This value is stored in device4_TRIG_VAL (defined in EXTI1.S but used by Device 4 context).

2. Sets the global trigger_state variable to 1 (indicating Device 4 context for EXTI).
3. Calls EXTI1_init to configure PA1 for rising edge interrupt.
4. Posts device4 Trigger Enabled on PA1.

- **EXTI1_IRQHandler (EXTI1.S):**
  1. Clears EXTI1 pending flag.
  2. If trigger_state is 1, branches to device4_EXTI1_IRQHandler.
- **device4_EXTI1_IRQHandler (EXTI1.S):**
  1. Reads the pre-stored device4_TRIG_VAL.
  2. Calls DAC1_CH2_output (from device4.S) to output this voltage.
- **Registers:** SYSCFG_EXTICR1, EXTI_IMR1, EXTI_RTSR1, EXTI_PR1, NVIC_ISER0. DAC_DHR12R2, DAC_SWTRIGR (via DAC1_CH2_output).

MoT System Integration

- **Command Handling:** device4_cmdHandler in device4.S uses a jump table (device4_cmds) for command dispatch.
- **Device Core:** MoT_core_m device4, device4_cmdHandler, device4_skiptask defines the Device 4 core.
- **Task Managerment**
  - device4_skiptask: Default idle task.
  - periodic_task_VA, periodic_task_VB: Scheduled by command 0x02 and reschedule each other to generate the periodic signal.
  - Command 0x03 switches the task to device4_skiptask to stop the periodic signal.
- **Messaging:** Uses MoT_msgPost with messages defined in device4.S (e.g., device4_initmsg, device4_set_const_msg).

Register Level Details

- **RCC (Reset and Clock Control):** RCC_AHB2ENR (GPIOA, DAC1), RCC_APB2ENR (SYSCFG).
- **GPIOA (General Purpose I/O Port A):** GPIOA_MODER (PA5 to Analog, PA1 to Input), GPIOA_PUPDR (PA5 no pull, PA1 pull-down).
- **DAC1:**
  - DAC_CR: Control Register (EN2 for Channel 2 enable, TEN2 for trigger enable).
  - DAC_MCR: Mode Control Register (MODE2 for buffer enable/disable).
  - DAC_DHR12R2: Channel 2 12-bit Right Aligned Data Holding Register.
  - DAC_SWTRIGR: Software Trigger Register (SWTRIG2 to trigger conversion).
- **SYSCFG, EXTI, NVIC:** (As detailed in Device 3, Section 3.4, for EXTI1 on PA1).

Control Flow

- **Command Execution:** Command -> device4_set_output -> arg_retrieve_half -> DAC1_CH2_output (writes to DAC_DHR12R2, then DAC_SWTRIGR).
- **Periodic Signal:**
  1. Command 0x02 -> device4_periodic_signal -> stores VA, T1, VB, T2 -> MoT_taskUpdate schedules periodic_task_VA.
  2. periodic_task_VA runs -> DAC1_CH2_output(VA) -> delay(T1) -> MoT_taskUpdate schedules periodic_task_VB.

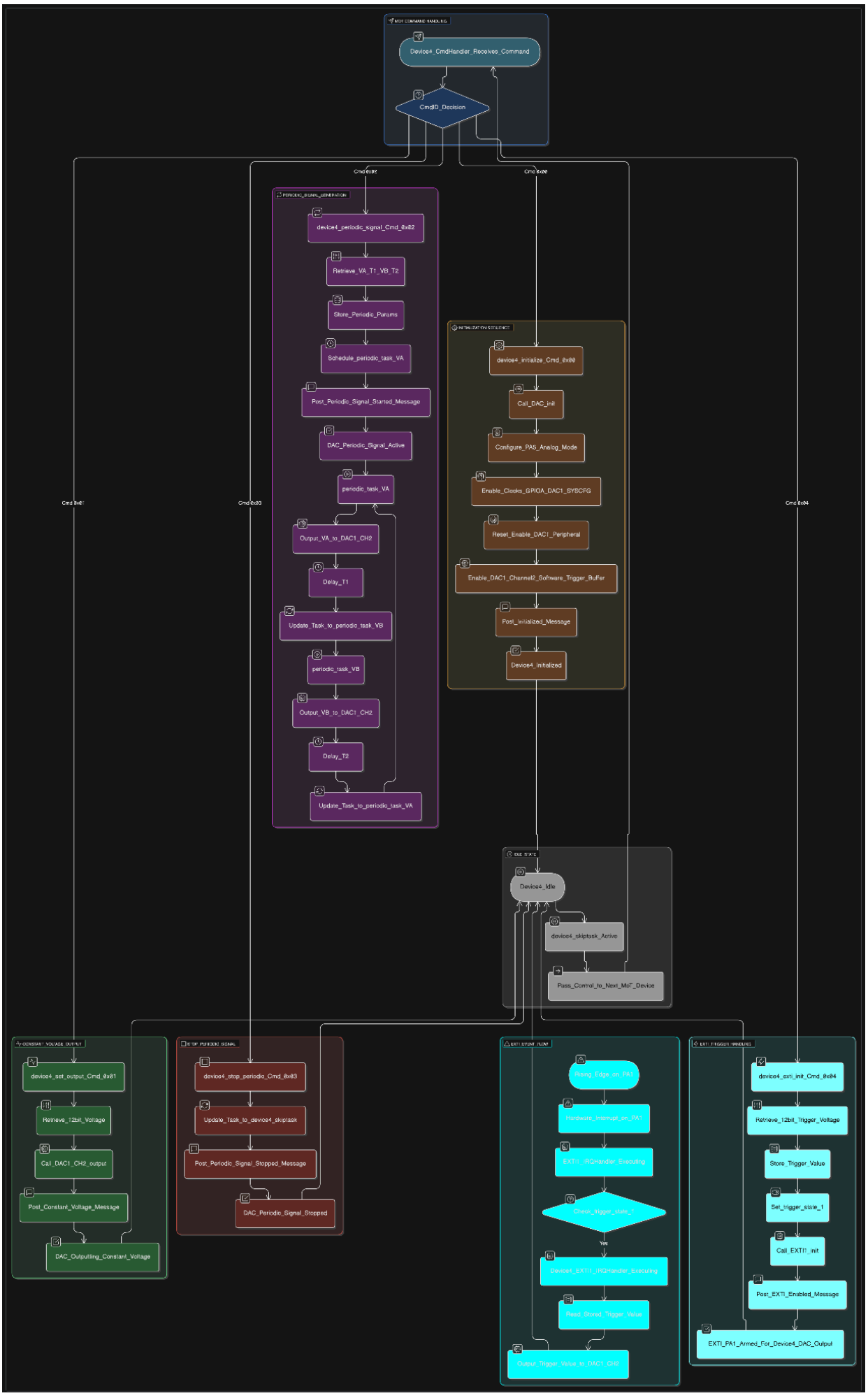
3. periodic_task_VB runs -> DAC1_CH2_output(VB) -> delay(T2) -> MoT_taskUpdate schedules periodic_task_VA.
4. Loop continues until device4_stop_periodic (Cmd 0x03) schedules device4_skiptask.

- **EXTI Triggered Output:**
  1. Command 0x04 -> device4_exti_init -> stores V_trig in device4_TRIG_VAL, sets trigger_state=1 -> EXTI1_init.
  2. PA1 Rising Edge -> EXTI1_IRQHandler -> checks trigger_state, branches to device4_EXTI1_IRQHandler.
  3. device4_EXTI1_IRQHandler -> reads device4_TRIG_VAL -> DAC1_CH2_output.

Device Data Storage

- **device4_VA, device4_T1, device4_VB, device4_T2:** Store parameters for periodic signal generation. Accessed via rDEVP + offset.
- **device4_TRIG_VAL (in EXTI1.S):** Stores the voltage value for EXTI-triggered DAC output.

State Diagram


Device4_CmdHandler_Receives_Command
CmdID_Decision
Cmd 0x02
Cmd 0x00
device4_periodic_signal_Cmd_0x02
Retrieve_VA_T1_VB_T2
Store_Periodic_Params
Schedule_periodic_task_VA
Post_Periodic_Signal_Started_Message
DAC_Periodic_Signal_Active
periodic_task_VA
Output_VA_to_DAC1_CH2
Delay_T1
Update_Task_to_periodic_task_VB
periodic_task_VB
Output_VB_to_DAC1_CH2
Delay_T2
Update_Task_to_periodic_task_VA
device4_initialize_Cmd_0x00
Call_DAC_init
Configure_PA5_Analog_Mode
Enable_Clocks_GPIOA_DAC1_SYSCFG
Reset_Enable_DAC1_Peripheral
Enable_DAC1_Channel2_Software_Trigger_Buffer
Post_Initialized_Message
Device4_Initialized
Cmd 0x01
Cmd 0x03
Cmd 0x04
Device4_Idle
device4_skiptask_Active
Pass_Control_to_Next_MoT_Device
device4_set_output_Cmd_0x01
Retrieve_12bit_Voltage
Call_DAC1_CH2_output
Post_Constant_Voltage_Message
DAC_Outputting_Constant_Voltage
device4_stop_periodic_Cmd_0x03
Update_Task_to_device4_skiptask
Post_Periodic_Signal_Stopped_Message
DAC_Periodic_Signal_Stopped
Rising_Edge_on_PA1
Hardware_Interrupt_on_PA1
EXTI1_IRQHandler_Executing
Check_trigger_state_1
Yes
Device4_EXTI1_IRQHandler_Executing
Read_Stored_Trigger_Value
Output_Trigger_Value_to_DAC1_CH2
device4_exti_init_Cmd_0x04
Retrieve_12bit_Trigger_Voltage
Store_Trigger_Value
Set_trigger_state_1
Call_EXTI1_init
Post_EXTI_Enabled_Message
EXTI_PA1_Armed_For_Device4_DAC_Output

Device 5: Analog-to-Digital Converter (ADC)

Introduction

Device 5 interfaces with the STM32G491's Analog-to-Digital Converter (ADC1), primarily reading analog voltage from pin PA0. It supports:

- **Initialization:** Configuring the ADC peripheral (ADC1) and the analog input pin (PA0).
- **Immediate Read:** Performing a single ADC conversion on PA0 and reporting the 12-bit digital result.
- **Threshold Monitoring:** Continuously monitoring the ADC value on PA0 against user-defined upper (H) and lower (L) thresholds. It reports when the value crosses these thresholds (ABOVE LIMIT, BELOW LIMIT).
- **EXTI Triggered Read:** Using an external interrupt on PA1 to trigger an ADC conversion on PA0 and report the result.

Hardware and Pin Configuration

- **Primary Analog Input Pin:** PA0 (Port A, Pin 0), connected to ADC1_IN1.
- **Auxiliary GPIO Pin (EXTI):** PA1 (Port A, Pin 1) is used as an external interrupt input.
- **Clock Enablement:**
    - GPIOA clock via RCC_AHB2ENR.
    - ADC12 clock via RCC_AHB2ENR (ADCEN).
    - ADC clock source selection via RCC_CCIPR (ADC12SEL).
    - SYSCFG clock via RCC_APB2ENR for EXTI.
- **Pin Initialization:**
    - initPA0 (device5.S): Configures PA0 to Analog mode (GPIOA_MODER) and no pull-up/pull-down (GPIOA_PUPDR).
    - EXTI1_init (EXTI1.S): Configures PA1 as input with pull-down for EXTI.
- **ADC Initialization:**
    1. Calls initPA0.
    2. Enables ADC12 clock (RCC_AHB2ENR_ADCEN).
    3. Selects ADC clock source (e.g., PLL via RCC_CCIPR_ADC12SEL).
    4. Exits deep power-down mode (ADC_CR_DEEPPWD = 0).
    5. Enables ADC voltage regulator (ADC_CR_ADVREGEN = 1) and waits for stabilization.
    6. Performs ADC calibration (ADC_CR_ADCAL = 1) and waits for completion.
    7. Enables the ADC (ADC_CR_ADEN = 1) and waits for it to be ready (ADC_ISR_ADRDY).
    8. Configures ADC Channel 1 (PA0) as the first in the regular sequence (ADC_SQR1).
    9. Sets sample time for Channel 1 (ADC_SMPR1).

Core Functionalities and Implementation

Commands are dispatched by device5_cmdHandler in device5.S.

Initialization

- **Command:** 0x00 (device5_initialize)
- **File:** device5.S
- **Action:**
    1. Calls report function.
    2. report function:
        - Starts ADC conversion (ADC_CR_ADSTART = 1).
        - Waits for End of Conversion (ADC_ISR_EOC).
        - Reads 12-bit data from ADC_DR.
        - Calls pull_up_down which seems to map the 12-bit ADC range to 0 or 4095 based on a threshold (around 2040).
        - Calls insert_hex_into_stopmsg to format the (potentially modified by pull_up_down) ADC value into the Output Vx from ADC 0xNNN message string.
    3. Posts the formatted message.
- **Registers:** ADC_CR, ADC_ISR, ADC_DR.

Input Operation

- **Report ADC Value (Immediate Read)**
    - **Command:** 0x01 (report_ADC_amp)
    - **File:** device5.S
    - **Action:**
        1. Calls report function.
        2. report function:
            a. Starts ADC conversion (ADC_CR_ADSTART = 1).
            b. Waits for End of Conversion (ADC_ISR_EOC).
            c. Reads 12-bit data from ADC_DR.
            d. Calls pull_up_down which seems to map the 12-bit ADC range to 0 or 4095 based on a threshold (around 2040).
        3. Calls insert_hex_into_stopmsg to format the (potentially modified by pull_up_down) ADC value into the Output Vx from ADC 0xNNN message string.
        4. Posts the formatted message.
    - **Registers:** ADC_CR, ADC_ISR, ADC_DR.
- **Report if ADC Value is Above/Below Thresholds (Start Monitoring)**
    - **Command:** 0x02 (device5_checklimits)
    - **File:** device5.S, Utils.S
    - **Action:**
        1. Retrieves V_low (16-bit) and V_high (16-bit) from payload using arg_retrieve_half and swap_low2bytes_r1.
        2. Stores these in device-specific variables device5_BELOW (V_low) and device5_ABOVE (V_high).
        3. Schedules device5_EQUALtask using MoT_taskUpdate.
        4. Posts ADC Range Monitor is On.
    - **device5_EQUALtask (device5.S):**
        1. Calls report to get current ADC value (r0).
        2. Compares r0 with device5_ABOVE. If r0 > device5_ABOVE, posts ABOVE LIMIT and schedules device5_ABOVEtask.

3. Else, compares r0 with device5_BELOW. If r0 < device5_BELOW, posts BELOW LIMIT and schedules device5_BELOWtask.
4. Else (value is within limits), reschedules device5_EQUALtask.

- **device5_ABOVEtask (device5.S):**
  1. Calls report.
  2. If ADC value <= device5_ABOVE:
     a. If ADC value < device5_BELOW, posts BELOW LIMIT, schedules device5_BELOWtask.
     b. Else (back in range), schedules device5_EQUALtask.
  3. Else (still above), reschedules device5_ABOVEtask.
- **device5_BELOWtask (device5.S):**
  1. Calls report.
  2. If ADC value >= device5_BELOW:
     a. If ADC value > device5_ABOVE, posts ABOVE LIMIT, schedules device5_ABOVEtask.
     b. Else (back in range), schedules device5_EQUALtask.
  3. Else (still below), reschedules device5_BELOWtask.
- **Registers:** DAC_DHR12R2, DAC_SWTRIGR. MoT task registers.

- **Stop Monitor Task / Clear Status**
  - **Command:** 0x03 (device5_stop_monitor)
  - **File:** device5.S
  - **Action:** Schedules device5_skiptask using MoT_taskUpdate. Posts Monitor Has Stopped. The demo output shows multiple Output Vx from ADC... messages before Monitor Has Stopped. This implies that when the monitor is stopped, it might report the current ADC value several times, or the test script is sending multiple read commands after stopping the monitor. The device5_stop_monitor function itself doesn't appear to actively report values.

EXTI (External Interrupt) Operation

- **Initialize EXTI on PA1 for Device 5**
  - **Command:** 0x04 (device5_trig)
  - **File:** device5.S, EXTI1.S
  - **Action:**
    1. Sets global trigger_state to 2 (Device 5 context for EXTI).
    2. Calls EXTI1_init to configure PA1 for rising edge interrupt.
    3. Posts device5 Trigger Enabled on PA1.
  - **EXTI1_IRQHandler (EXTI1.S):**
    1. Clears EXTI1 pending flag.
    2. If trigger_state is 2, branches to device5_EXTI1_IRQHandler.
  - **device5_EXTI1_IRQHandler (EXTI1.S):** Calls report_triggered.
  - **report_triggered (EXTI1.S):** This function simply calls report_ADC_amp from device5.S. So, on a PA1 trigger, an ADC conversion on PA0 is performed, and the result is reported to the console.

MoT System Integration

- **Command Handling:** device5_cmdHandler in device5.S uses a jump table.
- **Device Core:** MoT_core_m device5, device5_cmdHandler, device5_skiptask.
- **Task Managerment**
  - device5_skiptask: Default idle task.

    - device5_EQUALtask, device5_ABOVEtask, device5_BELOWtask: Mutually schedule each other for threshold monitoring.
- **Messaging:** Uses MoT_msgPost with messages defined in device5.S. The insert_hex_into_stopmsg function formats the ADC value into the device5_voltage_reporttxt string before posting.

Register Level Details

- **RCC (Reset and Clock Control):** RCC_AHB2ENR (GPIOA, ADC12), RCC_CCIPR (ADC12 clock source), RCC_APB2ENR (SYSCFG).
- **GPIOA (General Purpose I/O Port A):** GPIOA_MODER (PA0 to Analog, PA1 to Input), GPIOA_PUPDR (PA0 no pull, PA1 pull-down).
- **ADC1:**
    - ADC_CR: Control (DEEPPWD, ADVREGEN, ADCAL, ADEN, ADSTART).
    - ADC_ISR: Status (ADRDY, EOC).
    - ADC_DR: Data Register.
    - ADC_SQR1: Regular Sequence Register 1 (channel selection).
    - ADC_SMPR1: Sample Time Register 1.
- **SYSCFG, EXTI, NVIC:** (As detailed in Device 3, Section 3.4, for EXTI1 on PA1).

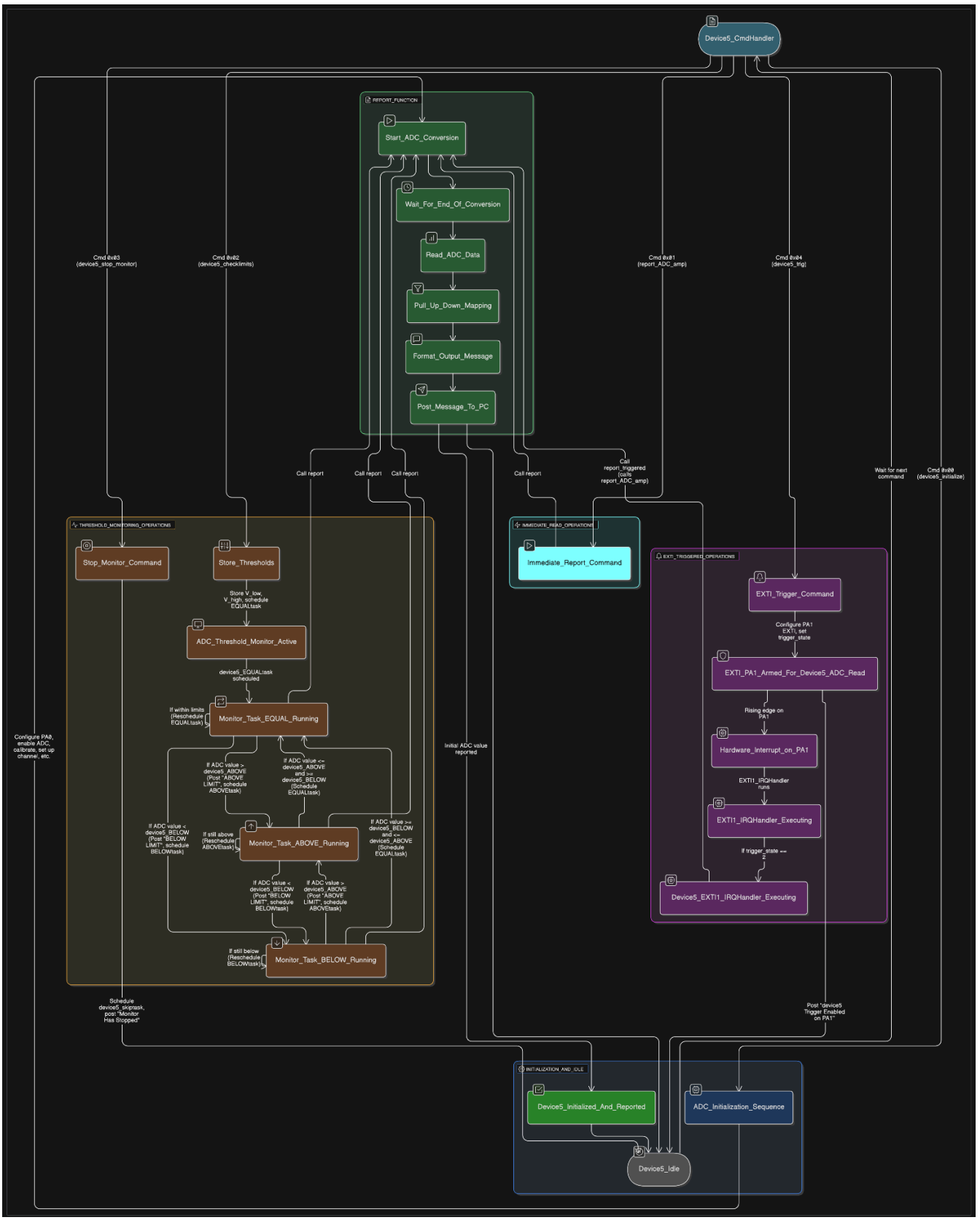
Control Flow

- **Immediate Read:** Cmd 0x01 -> report_ADC_amp -> report (starts conversion, waits, reads ADC_DR, calls pull_up_down) -> insert_hex_into_stopmsg -> MoT_msgPost.
- **Threshold Monitoring:**
    1. Cmd 0x02 -> device5_checklimits -> stores thresholds -> schedules device5_EQUALtask.
    2. device5_EQUALtask (or _ABOVEtask, _BELOWtask) runs -> report -> compares with stored thresholds -> posts message if crossed -> schedules appropriate next monitoring task.
- **EXTI Triggered Read:**
    1. Cmd 0x04 -> device5_trig -> sets trigger_state=2 -> EXTI1_init.
    2. PA1 Rising Edge -> EXTI1_IRQHandler -> checks trigger_state, branches to device5_EXTI1_IRQHandler.
    3. device5_EXTI1_IRQHandler -> report_triggered -> report_ADC_amp (which reads ADC and posts).

Device Data Storage

- **device5_BELOW (aliased device5_cyclecount):** Stores the lower ADC threshold.
- **device5_ABOVE (aliased device5_reload):** Stores the upper ADC threshold. Accessed via rDEVP + offset.

State Diagram


Device5_CmdHandler
REPORT_FUNCTION
Start_ADC_Conversion
Wait_For_End_Of_Conversion
Read_ADC_Data
Pull_Up_Down_Mapping
Format_Output_Message
Post_Message_To_PC
Cmd 0x03 (device5_stop_monitor)
Cmd 0x02 (device5_checklimits)
Cmd 0x01 (report_ADC_amp)
Cmd 0x04 (device5_trig)
Call report
Call report
Call report
Call report
Call report_triggered (calls report_ADC_amp)
Wait for next command
Cmd 0x00 (device5_initialize)
THRESHOLD_MONITORING_OPERATIONS
Stop_Monitor_Command
Store_Thresholds
Store V_low, V_high, schedule EQUALtask
ADC_Threshold_Monitor_Active
device5_EQUALtask scheduled
If within limits (Reschedule EQUALtask)
Monitor_Task_EQUAL_Running
If ADC value > device5_ABOVE (Post "ABOVE LIMIT", schedule ABOVEtask)
If ADC value <= device5_ABOVE and >= device5_BELOW (Schedule EQUALtask)
If ADC value < device5_BELOW (Post "BELOW LIMIT", schedule BELOWtask)
If still above (Reschedule ABOVEtask)
Monitor_Task_ABOVE_Running
If ADC value >= device5_BELOW and <= device5_ABOVE (Schedule EQUALtask)
If ADC value < device5_BELOW (Post "BELOW LIMIT", schedule BELOWtask)
If ADC value > device5_ABOVE (Post "ABOVE LIMIT", schedule ABOVEtask)
If still below (Reschedule BELOWtask)
Monitor_Task_BELOW_Running
Configure PA0, enable ADC, calibrate, set up channel, etc.
Initial ADC value reported
Schedule device5_skiptask, post "Monitor Has Stopped"
IMMEDIATE_READ_OPERATIONS
Immediate_Report_Command
EXTI_TRIGGERED_OPERATIONS
EXTI_Trigger_Command
Configure PA1 EXTI, set trigger_state
EXTI_PA1_Armed_For_Device5_ADC_Read
Rising edge on PA1
Hardware_Interrupt_on_PA1
EXTI1_IRQHandler runs
EXTI1_IRQHandler_Executing
If trigger_state == 2
Device5_EXTI1_IRQHandler_Executing
Post "device5 Trigger Enabled on PA1"
INITIALIZATION_AND_IDLE
Device5_Initialized_And_Reported
ADC_Initialization_Sequence
Device5_Idle

Device 6: Timer (TIM2)

Introduction

Device 6 provides an interface to the STM32G491's Timer 2 (TIM2) peripheral, outputting signals on GPIO pin PA5 (TIM2_CH1). This device allows for various timer-based signal generation modes:

- **Initialization:** Setting up TIM2, PA5 for alternate function, and related clocks.
- **Pulse Width Modulation (PWM):** Generating a PWM signal with configurable period (ARR) and duty cycle (CCR1).
- **Pulse Frequency Modulation (PFM):** Generating a signal with a fixed 50% duty cycle at a target frequency. (Note: The implementation seems to use PWM mode with calculated ARR for the target frequency and a fixed 50% duty cycle).
- **Simple Pulse (SP):** Generating a single pulse of a specified width. This is implemented using TIM2's One-Pulse Mode (OPM).
- **Pulse Width (PW / OPM):** Generating a single pulse with a specified initial delay and a specified pulse width, using OPM.
- **Pulse Frequency (PF):** Generating a single cycle of a square wave at a specified target frequency (50% duty), then stopping. This also uses OPM.
- **Triggered Control:** Arming TIM2 to generate one of the above signals (PWM, PFM, SP, PW, PF) upon an external trigger event on PA1 (EXTI1).
- **Timer Off:** Disabling the currently active timer function.

Hardware and Pin Configuration

- **Primary Timer Output Pin:** PA5 (Port A, Pin 5), configured as TIM2_CH1 alternate function.
- **Auxiliary GPIO Pin (EXTI):** PA1 (Port A, Pin 1) is used as an external interrupt input for triggered timer operations.
- **Clock Enablement:**
    - GPIOA clock via RCC_AHB2ENR for PA5 and PA1.
    - TIM2 clock via RCC_APB1ENR1.
    - SYSCFG clock via RCC_APB2ENR for EXTI functionality.
- **Pin Initialization (TIM2.S - device6_timer_base_init):**
    - PA5 is configured for Alternate Function mode (AF1 for TIM2_CH1), Push-Pull, High Speed, No Pull-up/pull-down.
- **Timer Initialization (TIM2.S - device6_timer_base_init):**
    - TIM2 clock is enabled.
    - TIM2 is initially disabled (TIMx_CR1_CEN = 0).
    - Channel 1 output is initially disabled (TIMx_CCER_CC1E = 0).
- **Default Timer Settings:**
    - Prescaler (TIMx_PSC) is set to achieve a TARGET_TIM_CLOCK_HZ (e.g., 1MHz from a PCLK1 of 170MHz).
    - ARR, CCR1 are set by specific mode functions.
    - Up-counting mode, Edge-aligned.

Core Functionalities and Implementation

Device 6 commands are dispatched by device6_cmdHandler in device6.S.

Initialization

- **Command:** 0x00 (device6_initialize)
- **File:** device6.S, TIM2.S
- **Action:** Calls device6_timer_output_disable (to reset TIM2 state) and device6_timer_base_init (to setup PA5 and TIM2 clocks/basic config). Posts device6 is Initialized.
- **Registers:** Modifies RCC_AHB2ENR, RCC_APB1ENR1, GPIOA_MODER, GPIOA_OTYPER, GPIOA_OSPEEDR, GPIOA_PUPDR, GPIOA_AFRL. Resets and configures TIM2_CR1, TIM2_CCER, TIM2_CCMR1, etc.

Input Operation

- **PWM On**
  - **Command:** 0x0B (11 decimal) (device6_off)
  - **File:** device6.S, TIM2.S
  - **Action:** Calls device6_timer_output_disable. Posts device6 Fucntion OFF.

Output Operation

- **PWM On**
  - **Command:** 0x01 (device6_pwm_on)
  - **File:** device6.S, TIM2_PWM.S, Utils.S
  - **Action:** Retrieves ARR (32-bit) and Duty (16-bit, percentage) from payload. Stores them in global variables pwm_arr_val and pwm_arr_duty. Calls device6_pwm_enable. Posts device6 PWM ON.
  - **device6_pwm_enable (TIM2_PWM.S):**
    1. Sets TIM2_PSC based on ACTUAL_PCLK1_HZ and TARGET_TIM_CLOCK_HZ.
    2. Sets TIM2_ARR from pwm_arr_val.
    3. Calculates CCR1 = (pwm_arr_duty * ARR) / 100. Sets TIM2_CCR1.
    4. Configures TIM2_CCMR1 for PWM Mode 1 (TIMx_CCMR1_OC1M_PWM1) and enables preload (TIMx_CCMR1_OC1PE).
    5. Configures TIM2_CCER to enable Channel 1 output (TIMx_CCER_CC1E) with active high polarity.
    6. Enables ARR preload (TIMx_CR1_ARPE).
    7. Generates an update event (TIMx_EGR_UG) to load shadow registers.
    8. Enables the timer counter (TIMx_CR1_CEN).
- **PFM On**
  - **Command:** 0x02 (device6_pfm_on)
  - **File:** device6.S, TIM2_PWM.S, Utils.S
  - **Action:** Retrieves target frequency (32-bit) from payload. Calls freq_to_arr_ticks (from Utils.S) to convert frequency to an ARR value (for TARGET_TIM_CLOCK_HZ). Stores this ARR in pwm_arr_val and sets pwm_arr_duty to 50. Calls device6_pwm_enable. Posts device6 PFM ON.
- **Simple Pulse On (SP)**
  - **Command:** 0x04 (device6_pulse_on)
  - **File:** device6.S, TIM2_Pulse.S, Utils.S

    - **Action:** Retrieves pulse width (32-bit ticks) from payload. Stores it in simple_pulse_width_ticks_val. Calls device6_simple_pulse_enable. Posts device6 Pulse ON.
    - **device6_simple_pulse_enable (TIM2_Pulse.S):**
        1. Loads simple_pulse_width_ticks_val into opm_width_ticks_val.
        2. Sets opm_delay_ticks_val to a small fixed value (e.g., 10 ticks).
        3. Calls device6_opm_enable.
- **Pulse Width On (PW / OPM)**
    - **Command:** 0x04 (device6_pulse_width_on)
    - **File:** device6.S, TIM2_Pulse.S, Utils.S
    - **Action:** Retrieves delay (32-bit ticks) and width (32-bit ticks) from payload. Stores them in opm_delay_ticks_val and opm_width_ticks_val. Calls device6_opm_enable. Posts device6 Pulse Width ON.
    - **device6_opm_enable (TIM2_Pulse.S):**
        1. Sets TIM2_PSC.
        2. Sets TIM2_CCR1 to opm_delay_ticks_val.
        3. Sets TIM2_ARR to opm_delay_ticks_val + opm_width_ticks_val.
        4. Configures TIM2_CCMR1 for PWM Mode 2 (0b111 - inactive then active) and enables preload.
        5. Configures TIM2_CCER for CH1 output, active high.
        6. Sets TIM2_CR1 for One-Pulse Mode (TIMx_CR1_OPM) and ARR preload (TIMx_CR1_ARPE).
        7. Generates update event and enables counter.
- **Pulse Frequency On (PF)**
    - **Command:** 0x05 (device6_pulse_frequency_on)
    - **File:** device6.S, TIM2_Pulse.S, Utils.S
    - **Action:** Retrieves target frequency (32-bit Hz) from payload. Stores it in single_freq_cycle_target_hz_val. Calls device6_single_freq_cycle_enable. Posts device6 Pulse Frequency ON.
    - **device6_single_freq_cycle_enable (TIM2_Pulse.S):**
        1. Sets TIM2_PSC.
        2. Calculates ARR = (TARGET_TIM_CLOCK_HZ / target_freq) - 1. Sets TIM2_ARR.
        3. Calculates CCR1 = (ARR + 1) / 2 (for 50% duty). Sets TIM2_CCR1.
        4. Configures TIM2_CCMR1 for PWM Mode 1 and preload.
        5. Configures TIM2_CCER for CH1 output, active high.
        6. Sets TIM2_CR1 for OPM and ARPE.
        7. Generates update event and enables counter.

MoT System Integration

- **Command Handling:** device6_cmdHandler in device6.S uses a jump table.
- **Device Core:** MoT_core_m device6, device6_cmdHandler, device6_skiptask.
- **Task Management:** Device 6 primarily uses direct command execution. No persistent MoT tasks are scheduled by Device 6 itself for its timer operations, as timer operations are hardware-driven once configured or run in OPM. The device6_skiptask is the default.
- **Messaging:** Uses MoT_msgPost with messages defined in device6.S.

EXTI (External Interrupt) Operation

- **Initialize Trigger for TIM2 on PA1**
  - **Command:** 0x06 (device6_triggered_init)
  - **File:** device6.S, EXTI1.S
  - **Action:** Sets global trigger_state to 0 (or a default indicating TIM2 context for EXTI). Calls EXTI1_init to configure PA1 for rising edge interrupt. Posts device6 Trigger Enabled.
- **Arm Triggered TIM2 Function**
  - **Command:** 0x07 (device6_triggered)
  - **File:** device6.S, TIM2_Trig.S, Utils.S
  - **Action:** Calls prep_trig. Posts device6 Trigger Armed With Given Command.
  - **prep_trig (TIM2_Trig.S):**
    1. Reads a sub-command byte from payload (1=PWM, 2=PFM, 3=SP, 4=PW, 5=PF).
    2. Sets trigger_state to a value corresponding to the sub-command (e.g., 3 for PWM, 4 for PFM, etc.).
    3. Based on sub-command, calls the appropriate store_<type>_cmd function (e.g., store_pwm_cmd) to parse further arguments from payload and store them in temporary global variables (e.g., temp_pwm_arr_val, temp_opm_delay_ticks_val).
- **EXTI1_IRQHandler (EXTI1.S):**
  1. Clears EXTI1 pending flag.
  2. If trigger_state matches one of the TIM2 armed states (3-7):
     a. Calls device6_timer_output_disable.
     b. Branches to a specific handler (e.g., device6_cmd_1_EXTI1_IRQHandler for PWM).
  3. Device 6 specific EXTI Handlers (e.g., device6_cmd_1_EXTI1_IRQHandler in EXTI1.S):**
     a. Copy parameters from temporary global variables (e.g., temp_pwm_arr_val) to actual operational global variables (e.g., pwm_arr_val).
     b. Call the corresponding enable function (e.g., device6_pwm_enable).

Register Level Details

- **RCC:** RCC_AHB2ENR (GPIOA), RCC_APB1ENR1 (TIM2), RCC_APB2ENR (SYSCFG).
- **GPIOA (for PA5):** GPIOA_MODER, GPIOA_OTYPER, GPIOA_OSPEEDR, GPIOA_PUPDR, GPIOA_AFRL.
- **TIM2:**
  - TIMx_CR1: Control Register 1 (CEN, OPM, ARPE, DIR, CMS).
  - TIMx_CR2: Control Register 2.
  - TIMx_SMCR: Slave Mode Control Register.
  - TIMx_DIER: DMA/Interrupt Enable Register.
  - TIMx_SR: Status Register.
  - TIMx_EGR: Event Generation Register (UG bit).
  - TIMx_CCMR1: Capture/Compare Mode Register 1 (OC1M for PWM mode, OC1PE for preload).
  - TIMx_CCER: Capture/Compare Enable Register (CC1E for CH1 enable, CC1P for polarity).

    - TIMx_CNT: Counter.
    - TIMx_PSC: Prescaler.
    - TIMx_ARR: Auto-Reload Register.
    - TIMx_CCR1: Capture/Compare Register 1.
- **SYSCFG, EXTI, NVIC:** (As detailed in Device 3, Section 3.4, for EXTI1 on PA1).

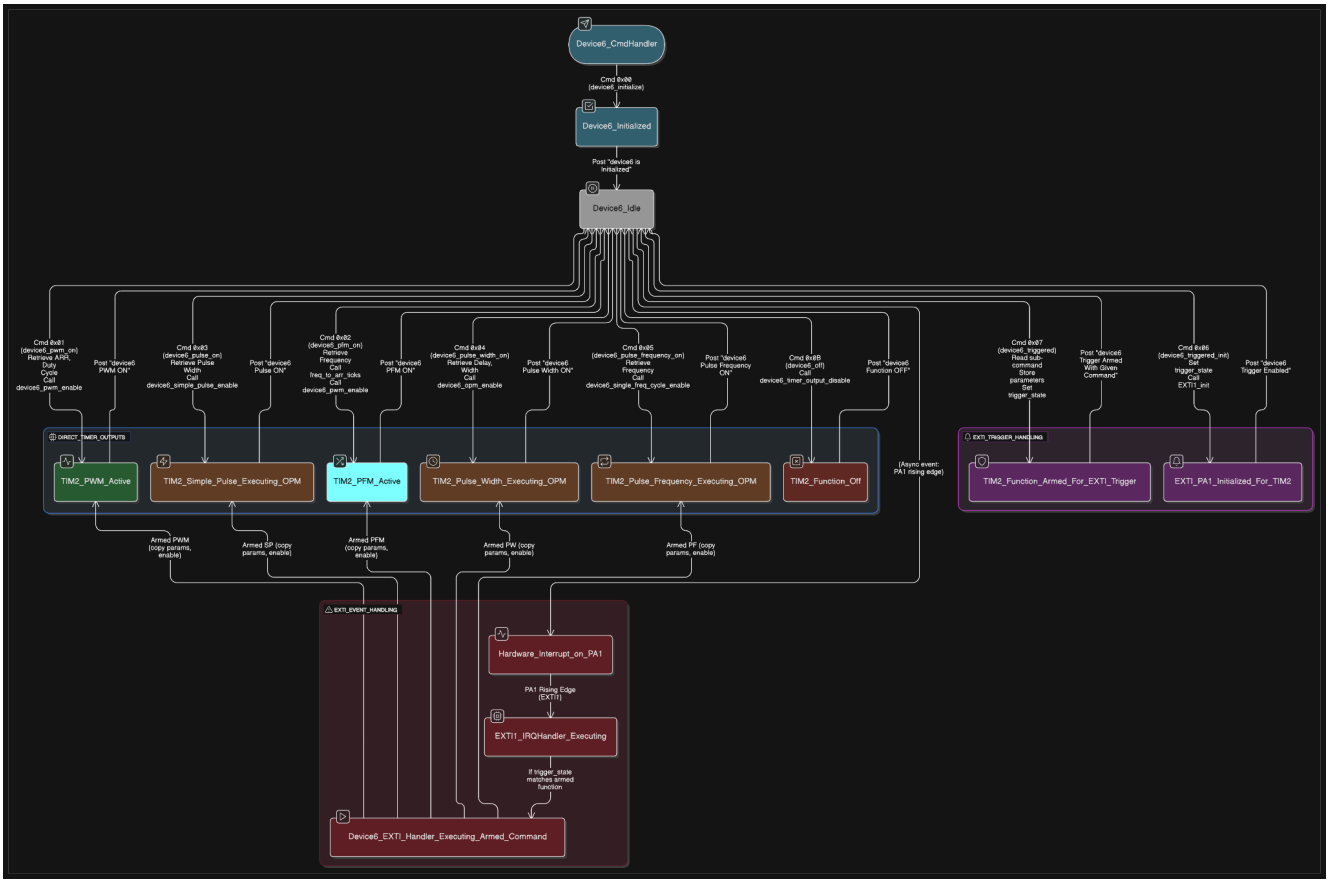
Control Flow

- **Direct Timer Commands (PWM, PFM, SP, PW, PF):**
    1. Command received -> device6_cmdHandler dispatches.
    2. Specific command function (e.g., device6_pwm_on) is called.
    3. Payload arguments are parsed using arg_retrieve* and stored in global variables (e.g., pwm_arr_val).
    4. The corresponding device6_<mode>_enable function is called (e.g., device6_pwm_enable).
    5. This function configures TIM2 registers (PSC, ARR, CCR1, CCMR1, CCER, CR1) and starts the timer.
    6. Confirmation message is posted.
- **Triggered Timer Commands:**
    1. device6_triggered_init (Cmd 0x06) -> sets trigger_state=0 (default for TIM2), calls EXTI1_init.
    2. device6_triggered (Cmd 0x07) -> prep_trig -> parses sub-command and parameters, stores params in temp_* globals, sets trigger_state (3-7).
    3. PA1 Rising Edge -> EXTI1_IRQHandler -> checks trigger_state.
    4. If trigger_state is 3-7, branches to specific handler in EXTI1.S (e.g., device6_cmd_1_EXTI1_IRQHandler).
    5. Specific handler copies params from temp_* to operational globals (e.g., pwm_arr_val) -> calls the relevant device6_<mode>_enable function.
- **Timer Off:** Cmd 0x0B -> device6_off -> device6_timer_output_disable (resets TIM2 registers).

Device Data Storage

- **Operational Parameters (global variables in device6.S):**
    - pwm_arr_val, pwm_arr_duty
    - opm_delay_ticks_val, opm_width_ticks_val
    - pfm_fixed_pulse_width_ticks_val, pfm_current_arr_val (Note: PFM seems implemented via PWM, so pfm_current_arr_val might map to pwm_arr_val and pfm_fixed_pulse_width_ticks_val might not be directly used by device6_pwm_enable if PFM uses 50% duty of the calculated ARR).
    - single_freq_cycle_target_hz_val
    - simple_pulse_width_ticks_val
- **Temporary Parameters for Triggered Operations (global variables in TIM2_Trig.S):**
    - temp_pwm_arr_val, temp_pwm_arr_duty
    - temp_opm_delay_ticks_val, temp_opm_width_ticks_val
    - temp_pfm_fixed_pulse_width_ticks_val, temp_pfm_current_arr_val
    - temp_single_freq_cycle_target_hz_val
    - temp_simple_pulse_width_ticks_val
- **trigger_state (global, in the main file):** Used to coordinate EXTI events with the intended device and sub-function.

State Diagram



Device 7: SPI Loopback Testing

Introduction

Device 7 provides functionality for testing the Serial Peripheral Interface (SPI), specifically SPI2, in a loopback configuration. This means the MOSI (Master Out Slave In) pin is connected externally to the MISO (Master In Slave Out) pin. This allows data sent by the SPI master to be immediately received back, verifying the basic operation of the SPI peripheral and data paths. Device 7 supports:

- **Initialization:** Configuring the SPI2 peripheral and associated GPIO pins for communication.
- **SPI Write & Loopback Read:** Writing a specified number of bytes to the SPI bus and simultaneously capturing the data received on MISO into an internal buffer.
- **SPI Read from Buffer:** Reading the contents of the internal receive buffer (data captured during previous write operations) and displaying it.
- **Clear Buffers:** Resetting internal read and write software buffers.

Hardware and Pin Configuration

- **SPI Peripheral:** SPI2.
- **GPIO Pins (Port B):**
    - PB12: NSS (Network Slave Select) - configured as GPIO Output, manually controlled.
    - PB13: SCK (Serial Clock) - Alternate Function AF5.
    - PB14: MISO (Master In Slave Out) - Alternate Function AF5.
    - PB15: MOSI (Master Out Slave In) - Alternate Function AF5.
- **Clock Enablement (device7_enable_clocks in device7.S)::**
    - GPIOB clock via RCC_AHB2ENR_GPIOBEN.
    - SPI2 clock via RCC_APB1ENR1_SPI2EN.
    - SPI2 peripheral is also reset via RCC_APB1RSTR1_SPI2RST.
- **Pin Initialization (device7_configure_gpios in device7.S):**
    - PB12 (NSS): GPIO Output, Push-Pull, Very High Speed, No Pull-up/down. Initialized High (de-asserted).
    - PB13 (SCK), PB14 (MISO), PB15 (MOSI): Alternate Function (AF5), Push-Pull, Very High Speed, No Pull-up/down.
- **SPI Peripheral Configuration (device7_configure_spi_peripheral in device7.S):**
    - SPI disabled (SPI_CR1_SPE = 0) before configuration.
    - Master mode (SPI_CR1_MSTR = 1).
    - Baud rate: PCLK/16 (SPI_CR1_BR_DIV16). (PCLK is APB1 clock).
    - CPOL=0, CPHA=0 (SPI Mode 0).
    - Software Slave Management enabled (SPI_CR1_SSM = 1).
    - Internal Slave Select set high (SPI_CR1_SSI = 1) to ensure master operation with SSM.
    - Data size: 8-bit (SPI_CR2_DS_8BIT).
    - RXNE event on 8-bit reception (SPI_CR2_FRXTH = 1).
    - SPI enabled (SPI_CR1_SPE = 1) after configuration.

Core Functionalities and Implementation

Device 7 commands are dispatched by device7_cmdHandler in device7.S.

Initialization

- **Command:** 0x00 (device7_initialize)
- **File:** device7.S
- **Action:** Calls device7_enable_clocks, device7_configure_gpios, and device7_configure_spi_peripheral. Posts device7: SPI2 initialized (PB12-15).
- **Registers:** Modifies RCC, GPIOB, and SPI2 registers as detailed in the initialization functions.

Input Operation

- **SPI Write (and Loopback Read to Buffer)**
    - **Command:** 0x01 (device7_write)
    - **File:** device7.S
    - **Action:**
        1. Reads a count byte (N) from the command payload.
        2. Calls store_message_bytes_into_write_arr to copy N bytes from the command payload into the write_arr software buffer.
        3. Loops N times, calling device7_write_byte_action in each iteration.
        4. Posts device7: SPI2 write complete.
    - **store_message_bytes_into_write_arr (device7.S):** Copies bytes from the MoT command buffer (r0) to the global write_arr.
    - **device7_write_byte_action (device7.S):**
        1. Calls read_byte_from_write_arr to get the next byte to send from write_arr (uses/updates pointer_write).
        2. Asserts NSS (PB12 LOW).
        3. Waits for SPI TXE (Transmit Buffer Empty) flag (SPI_SR_TXE).
        4. Writes the byte to SPI2_DR.
        5. Waits for SPI RXNE (Receive Buffer Not Empty) flag (SPI_SR_RXNE).
        6. Reads the received byte from SPI2_DR.
        7. Calls write_byte_to_read_arr to store the received byte into the read_arr software buffer (uses/updates pointer_read).
        8. Waits for SPI BSY (Busy) flag to clear.
        9. De-asserts NSS (PB12 HIGH).
    - **Registers:** SPI2_CR1, SPI2_CR2, SPI2_SR, SPI2_DR, GPIOB_BSRR.
- **Clear SPI Buffers**
    - **Command: 0x03 (device7_clr_read_arr)**
    - **File:** device7.S
    - **Action:** Calls arr_clear. Posts device7: Read and Write Buffers CLEAR.
    - **arr_clear (device7.S):**
        1. Resets pointer_read and pointer_write to 0.
        2. Fills the global read_arr and write_arr with zeros up to MAX_ARR_SIZE.

Output Operation

- **SPI Read (from Buffer)**
    - **Command:** 0x02 (device7_read)
    - **File:** device7.S, Utils.S (for nibble_to_ascii)
    - **Action:**
        1. Calls device7_read_byte_action.

2. Posts the content of device7_read_txt (which was populated by device7_read_byte_action).

- **device7_read_byte_action (device7.S):**
  9. Iterates up to MAX_ARR_SIZE times.
  10. For each byte in the global read_arr:
      a. Converts the byte to two ASCII hex characters using nibble_to_ascii.
      b. Writes these characters into the device7_read_txt message string buffer at the appropriate offset.

MoT System Integration

- **Command Handling:** device7_cmdHandler in device7.S uses a jump table.
- **Device Core:** MoT_core_m device7, device7_cmdHandler, device7_skiptask.
- **Task Managerment:** Device 7 operations are command-driven and synchronous. The device7_skiptask is the default MoT task.
- **Messaging:** Uses MoT_msgPost with messages defined in device7.S.

Register Level Details

- **RCC (Reset and Clock Control):**
  - RCC_AHB2ENR: RCC_AHB2ENR_GPIOBEN for GPIOB clock.
  - RCC_APB1ENR1: RCC_APB1ENR1_SPI2EN for SPI2 clock.
  - RCC_APB1RSTR1: RCC_APB1RSTR1_SPI2RST for SPI2 reset.
- **GPIOB (for PB12-PB15):**
  - GPIOB_MODER: Mode (Output for NSS, AF for SCK, MISO, MOSI).
  - GPIOB_OTYPER: Output type (Push-Pull).
  - GPIOB_OSPEEDR: Output speed (Very High Speed).
  - GPIOB_PUPDR: Pull-up/Pull-down (No pull).
  - GPIOB_BSRR: For NSS control.
  - GPIOB_AFRH: Alternate Function Register High (AF5 for SPI2).
- **SPI2:**
  - SPI_CR1: Control Register 1 (SPE, MSTR, BR, CPOL, CPHA, SSM, SSI).
  - SPI_CR2: Control Register 2 (DS, FRXTH).
  - SPI_SR: Status Register (TXE, RXNE, BSY).
  - SPI_DR: Data Register.

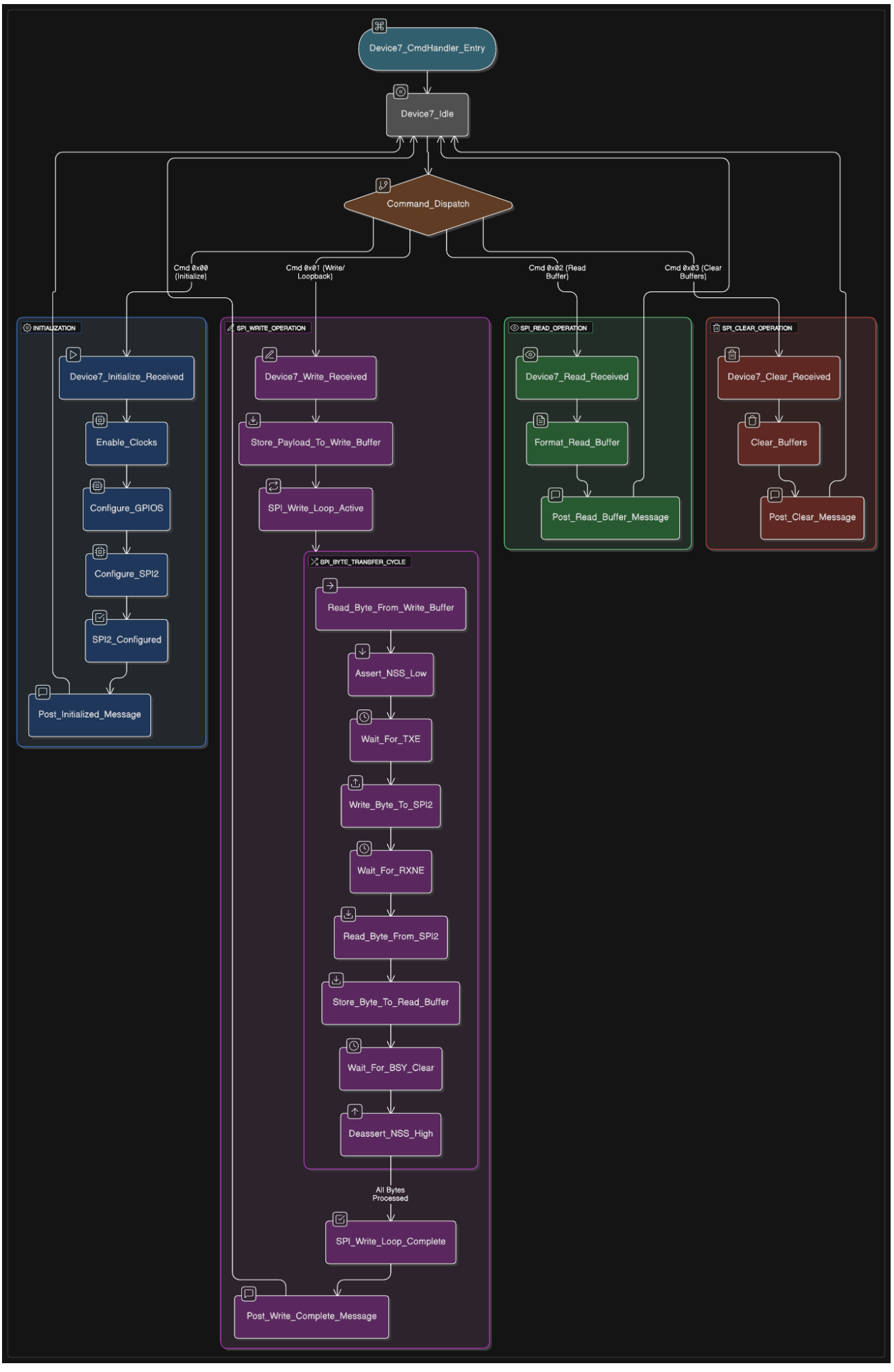
Control Flow

- **Initialization (Cmd 0x00):**
  - device7_initialize -> device7_enable_clocks -> device7_configure_gpios -> device7_configure_spi_peripheral.
- **Write (Cmd 0x01):**
  - device7_write -> parses count -> store_message_bytes_into_write_arr (payload to write_arr) -> loop N times: device7_write_byte_action.
  - device7_write_byte_action -> read_byte_from_write_arr -> SPI send/receive -> write_byte_to_read_arr (SPI data to read_arr).
- **Read (Cmd 0x02):**
  - device7_read -> device7_read_byte_action (formats read_arr into device7_read_txt) -> MoT_msgPost (device7_read_txt).

Device Data Storage

- **read_arr (device7.S, .data section):** Software buffer of MAX_ARR_SIZE (10 bytes) to store data received from SPI MISO.
- **write_arr (device7.S, .data section):** Software buffer of MAX_ARR_SIZE (10 bytes) to store data to be sent via SPI MOSI.
- **pointer_read (device7.S, .data section)**: Byte variable, acts as an index for the read_arr.
- **pointer_write (device7.S, .data section):** Byte variable, acts as an index for the write_arr.

State Diagram


Device7_CmdHandler_Entry
Device7_Idle
Command_Dispatch
Cmd 0x00 (Initialize)
Cmd 0x01 (Write/ Loopback)
Cmd 0x02 (Read Buffer)
Cmd 0x03 (Clear Buffers)
INITIALIZATION
Device7_Initialize_Received
Enable_Clocks
Configure_GPIOS
Configure_SPI2
SPI2_Configured
Post_Initialized_Message
SPI_WRITE_OPERATION
Device7_Write_Received
Store_Payload_To_Write_Buffer
SPI_Write_Loop_Active
SPI_BYTE_TRANSFER_CYCLE
Read_Byte_From_Write_Buffer
Assert_NSS_Low
Wait_For_TXE
Write_Byte_To_SPI2
Wait_For_RXNE
Read_Byte_From_SPI2
Store_Byte_To_Read_Buffer
Wait_For_BSY_Clear
Deassert_NSS_High
All Bytes Processed
SPI_Write_Loop_Complete
Post_Write_Complete_Message
SPI_READ_OPERATION
Device7_Read_Received
Format_Read_Buffer
Post_Read_Buffer_Message
SPI_CLEAR_OPERATION
Device7_Clear_Received
Clear_Buffers
Post_Clear_Message

Device 8: W25QXX Flash Memory Storage

Introduction

Device 8 provides an interface to an external W25QXX series SPI Flash memory chip (e.g., W25Q128FV). This allows for non-volatile storage of data, managed through the MoT system. Key functionalities include:

- Initializing the SPI2 peripheral and the W25QXX flash memory.
- Reading the JEDEC identification data from the flash chip.
- Erasing a sector of the flash memory.
- Inputting data from the MoT command payload into an internal RAM buffer.
- Copying (writing) data from the RAM buffer to the flash memory.
- Verifying data in the flash memory against the RAM buffer.
- Outputting (reading) data from the flash memory to the console.

Communication with the W25QXX chip is performed using the SPI2 peripheral.

Hardware and Pin Configuration

- **SPI Peripheral:** SPI2 is used for communication with the W25QXX flash memory.
- **GPIO Pins (Port B):**
    - PB12: NSS (Chip Select) - configured as GPIO Output, manually controlled by w25q128_assert_cs and w25q128_deassert_cs.
    - PB13: SCK (Serial Clock) - Alternate Function AF5.
    - PB14: MISO (Master In Slave Out) - Alternate Function AF5.
    - PB15: MOSI (Master Out Slave In) - Alternate Function AF5.
- **Clock Enablement:**
    - GPIOB clock is enabled via RCC_AHB2ENR_GPIOBEN bit in RCC_AHB2ENR.
    - SPI2 clock is enabled via RCC_APB1ENR1_SPI2EN bit in RCC_APB1ENR1.
    - The SPI2 peripheral is reset via RCC_APB1RSTR1_SPI2RST bit in RCC_APB1RSTR1 during initialization (device7_enable_clocks called by w25q128_reset_init).
- **SPI Configuration (performed by device7_configure_spi_peripheral called during w25q128_reset_init):**
    - Master mode (SPI_CR1_MSTR = 1).
    - Baud rate: PCLK1/16 (APB1 clock divided by 16, SPI_CR1_BR_DIV16).
    - CPOL=0, CPHA=0 (SPI Mode 0).
    - 8-bit data size (SPI_CR2_DS_8BIT).
    - Software Slave Management enabled (SPI_CR1_SSM = 1).
    - Internal Slave Select set high (SPI_CR1_SSI = 1) to ensure master operation with SSM.
    - RXNE event generated when 8 bits are received (SPI_CR2_FRXTH = 1).
    - SPI enabled (SPI_CR1_SPE = 1) after configuration.

Core Functionalities and Implementation

Device 8 commands are dispatched by device8_cmdHandler in device8.S. This handler uses a jump table device8_cmds to execute specific command routines. Helper functions for W25QXX operations are primarily located in W25QXX_init_and_helper.S, W25QXX_read_ident.S, W25QXX_copy_erase.S, W25QXX_read.S, and W25QXX_verify.S.

Initialization

- **Command:** 0x00 (device8_initialize)
- **File:** device8.S, W25QXX_init_and_helper.S
- **Action:**
    1. Calls w25q128_reset_init.
        a. w25q128_reset_init first calls SPI and GPIO setup functions: device7_enable_clocks, device7_configure_gpios, and device7_configure_spi_peripheral (these are defined, shared with Device 7, to configure SPI2 and GPIOB pins PB12-PB15 as detailed above).
        b. Asserts CS (PB12 LOW) via w25q128_assert_cs.
        c. Sends W25QXX Enable Reset command (W25_CMD_ENABLE_RESET = 0x66) using spi_send_receive_byte.
        d. De-asserts CS (PB12 HIGH) via w25q128_deassert_cs.
        e. Introduces a short delay using short_delay.
        f. Asserts CS.
        g. Sends W25QXX Reset Device command (W25_CMD_RESET_DEVICE = 0x99) using spi_send_receive_byte.
        h. De-asserts CS.
        i. Waits for W25QXX reset recovery time (t_RST) using short_delay.
    2. Posts the confirmation message device8 has been initialized on W25Q128 using MoT_msgPost with device8_initmsg.

Initialization

- **Erase Sector**
    - **Command:** 0x02 (device8_erase)
    - **File:** device8.S, W25QXX_copy_erase.S, W25QXX_init_and_helper.S
    - **Action:**
        1. Sets r0 to the flash address 0x000000 (hardcoded).
        2. Calls w25q128_erase_sector with the address.
            a. Calls w25q128_write_enable to send the Write Enable command (0x06) to the flash.
            b. Asserts CS.
            c. Sends the Sector Erase command (W25_CMD_SECTOR_ERASE = 0x20).
            d. Sends the 24-bit sector address (from r0).
            e. De-asserts CS.
            f. Calls w25q128_wait_for_write_complete which polls Status Register-1 (SR1) by repeatedly calling w25q128_read_status_register1 until the BUSY bit (SR1_BUSY_BIT) is clear.
        3. Posts device8 erased directed block using device8_erase_blockmsg.

- **Copy RAM Buffer to Flash**
  - **Command:** 0x04 (device8_copy)
  - **File:** device8.S, W25QXX_copy_erase.S, W25QXX_init_and_helper.S
  - **Action:**
    1. Loads ram_buffer address into r0.
    2. Sets destination flash address to 0x000000 (hardcoded) in r1.
    3. Calls w25q128_write_ram_buffer with RAM buffer address and flash address.
       a. Iterates in chunks of PAGE_SIZE (256 bytes) until RAM_BUFFER_SIZE bytes are written.
       b. For each page:
          i. Calls w25q128_write_enable.
          ii. Asserts CS.
          iii. Sends Page Program command (W25_CMD_PAGE_PROGRAM = 0x02).
          iv. Sends the 24-bit flash page address.
          v. Sends PAGE_SIZE bytes of data from the RAM buffer.
          vi. De-asserts CS.
          vii. Calls w25q128_wait_for_write_complete.
    4. Posts device8 copy complete using device8_copy_donemsg.
- **Verify Flash against RAM Buffer**
  - **Command:** 0x05 (device8_verify)
  - **File:** device8.S, W25QXX_read.S, W25QXX_verify.S, W25QXX_init_and_helper.S
  - **Action:**
    1. Calls device8_read (internal helper function in device8.S).
       a. device8_read loads received_buffer address into r1 and flash address 0x000000 into r0.
       b. Calls w25q128_read_ram_buffer to read RAM_BUFFER_SIZE bytes from flash (address 0x000000) into received_buffer.
          i. w25q128_read_ram_buffer asserts CS, sends Read Data command (0x03), sends 24-bit address, reads data by sending dummy bytes, then de-asserts CS.
    2. Loads ram_buffer address into r0 and received_buffer address into r1.
    3. Calls w25q128_verify_ram_buffer.
       a. Compares ram_buffer and received_buffer byte-by-byte for RAM_BUFFER_SIZE bytes.
       b. Returns 1 in r0 for success (match), 0 for failure (mismatch).
    4. Based on the return value from w25q128_verify_ram_buffer:
       a. If success (1), posts device8 verification successful using device8_verify_successmsg.
       b. If failure (0), posts device8 verification failed using device8_verify_failedmsg.
  - **Data Storage:** ram_buffer, received_buffer.

Input Operation

- **Input Data to RAM Buffer**
  - **Command:** 0x05 (device8_verify)
  - **File:** device8.S, W25QXX_read.S, W25QXX_verify.S, W25QXX_init_and_helper.S

- **Action:**
    1. Calls device8_read (internal helper function in device8.S).
        a. device8_read loads received_buffer address into r1 and flash address 0x000000 into r0.
        b. Calls w25q128_read_ram_buffer to read RAM_BUFFER_SIZE bytes from flash (address 0x000000) into received_buffer.
            i. w25q128_read_ram_buffer asserts CS, sends Read Data command (0x03), sends 24-bit address, reads data by sending dummy bytes, then de-asserts CS.
    2. Loads ram_buffer address into r0 and received_buffer address into r1.
    3. Calls w25q128_verify_ram_buffer.
        a. Compares ram_buffer and received_buffer byte-by-byte for RAM_BUFFER_SIZE bytes.
        b. Returns 1 in r0 for success (match), 0 for failure (mismatch).
    4. Based on the return value from w25q128_verify_ram_buffer:
        a. If success (1), posts device8 verification successful using device8_verify_successmsg.
        b. If failure (0), posts device8 verification failed using device8_verify_failedmsg.
- **Data Storage:** ram_buffer, received_buffer.

Output Operation

- **Output Data from Flash**
    - **Command:** 0x06 (device8_output)
    - **File:** device8.S
    - **Action:**
        1. Calls clear_output_msg to fill the data portion of device8_outputtxt with '0' characters.
        2. Reads a count byte from the MoT command payload (how many bytes to display).
        3. Loads received_buffer address into r3 (this buffer should contain data read from flash, by a preceding Verify or an explicit Read operation targeting address 0x000000).
        4. Loads the starting address within device8_outputtxt (after device8 output from flash 0x) into r2.
        5. Loops 'count' times:
            a. Reads a byte from received_buffer.
            b. Converts the byte to two ASCII hex characters using nibble_to_ascii.
            c. Writes these characters into device8_outputtxt.
        6. Posts the updated device8_outputmsg to the console.
    - **Data Source:** received_buffer (populated with data from flash address 0x000000).

MoT System Integration

- **Command Handling:** device8_cmdHandler in device8.S is the entry point. It uses a Table Branch Byte (tbb) instruction with the device8_cmds jump table for dispatching based on the command ID.

- **Device Core:** The MoT_core_m device8, device8_cmdHandler, device8_skiptask macro instantiation in device8.S defines the core MoT structure for this device.
- **Task Management:** device8_skiptask: A minimal task that does nothing but pass control to the next device. This is the default active task. Device 8 operations are primarily command-driven and synchronous.
- **Messaging:** Device 8 uses MoT_msgPost to send status and data messages to the consoleMsgs queue. Message link structures (e.g., device8_initmsg, device8_read_identmsg, device8_outputmsg) are defined in the .data section of device8.S.
    - device8_read_identtxt is dynamically updated by print_ident.
    - device8_outputtxt is dynamically updated by device8_output after being cleared by clear_output_msg.

Register Level Details

- **RCC (Reset and Clock Control):**
    - RCC_AHB2ENR: RCC_AHB2ENR_GPIOBEN (bit 1) to enable GPIOB clock.
    - RCC_APB1ENR1: RCC_APB1ENR1_SPI2EN (bit 14) to enable SPI2 clock.
    - RCC_APB1RSTR1: RCC_APB1RSTR1_SPI2RST (bit 14) to reset SPI2 peripheral.
- **GPIOB (General Purpose I/O Port B for PB12-PB15):**
    - GPIOB_MODER: Configures pins for Output (PB12-NSS) or Alternate Function (PB13-SCK, PB14-MISO, PB15-MOSI).
        - PB12 (NSS): Output mode (0b01).
        - PB13, PB14, PB15: Alternate Function mode (0b10)
    - GPIOB_OTYPER: Output type (Push-Pull for all).
    - GPIOB_OSPEEDR: Output speed (Very High Speed for all).
    - GPIOB_PUPDR: Pull-up/Pull-down (No pull for all).
    - GPIOB_BSRR: Used to set/reset PB12 (NSS) for w25q128_assert_cs and w25q128_deassert_cs.
    - GPIOB_AFRH: Configures Alternate Function AF5 for PB13, PB14, PB15.
- **SPI2 (Serial Peripheral Interface 2):**
    - SPIx_CR1: Control Register 1 (SPE, MSTR, BR, CPOL, CPHA, SSM, SSI).
    - SPIx_CR2: Control Register 2 (DS, FRXTH).
    - SPIx_SR: Status Register (TXE, RXNE, BSY).
    - SPIx_DR: Data Register (for sending commands/data and receiving data).

Control Flow

- **Initialization (Cmd 0x00):**
    - device8_cmdHandler -> device8_initialize -> w25q128_reset_init (configures SPI2/GPIOB, sends W25QXX reset commands 0x66, 0x99 via spi_send_receive_byte) -> MoT_msgPost.
- **Read JEDEC ID (Cmd 0x01):**
    - device8_cmdHandler -> device8_read_ident -> w25q128_read_jedec_id (sends 0x9F, reads 3 ID bytes) -> print_ident (formats IDs into device8_read_identtxt) -> MoT_msgPost.
- **Erase Sector (Cmd 0x02):**

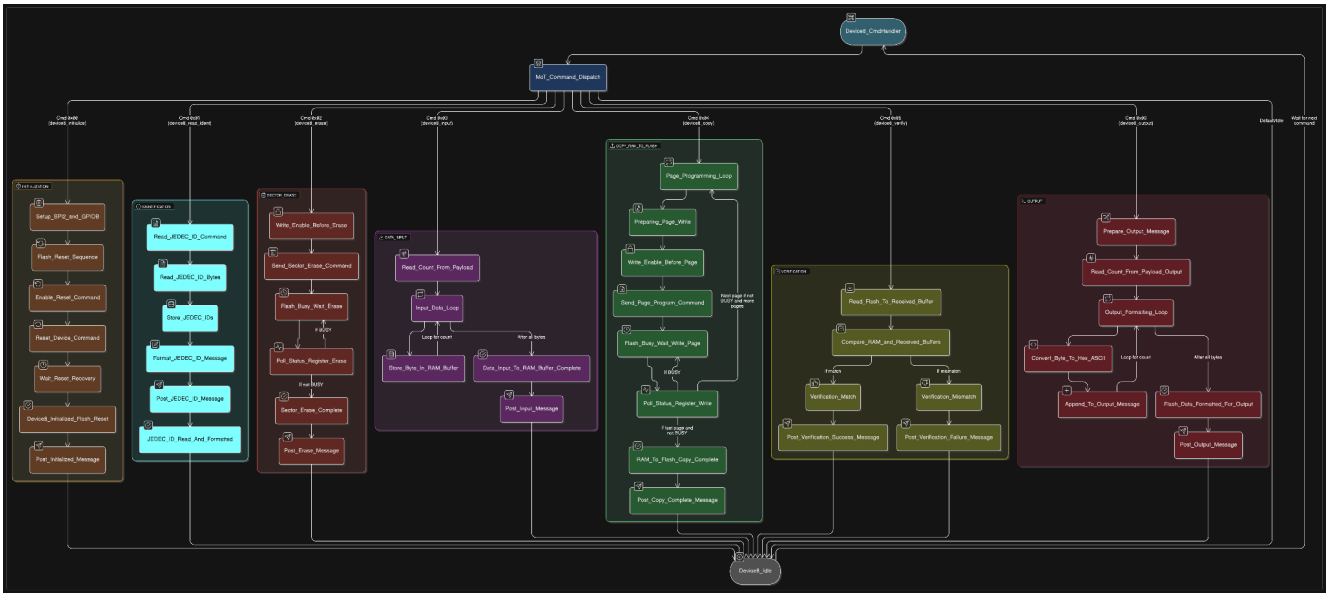
- device8_cmdHandler -> device8_erase -> w25q128_erase_sector (addr 0x000000; calls w25q128_write_enable, sends 0x20 + address, then w25q128_wait_for_write_complete) -> MoT_msgPost.

- **Input Data to RAM (Cmd 0x03):**
  - device8_cmdHandler -> device8_input (reads count and data from MoT payload into ram_buffer) -> MoT_msgPost.
- **Copy RAM to Flash (Cmd 0x04):**
  - device8_cmdHandler -> device8_copy -> w25q128_write_ram_buffer (from ram_buffer to flash addr 0x000000; page-by-page: w25q128_write_enable, sends 0x02 + addr + data, w25q128_wait_for_write_complete) -> MoT_msgPost.
- **Verify Flash (Cmd 0x05):**
  - device8_cmdHandler -> device8_verify -> device8_read (helper, calls w25q128_read_ram_buffer to read flash 0x000000 into received_buffer) -> w25q128_verify_ram_buffer (compares ram_buffer and received_buffer) -> MoT_msgPost (success/fail).
- **Output Data from Flash (Cmd 0x06):**
  - device8_cmdHandler -> device8_output -> clear_output_msg -> reads count from MoT payload -> formats specified number of bytes from received_buffer into device8_outputtxt using nibble_to_ascii -> MoT_msgPost. (Data in received_buffer is from the last read operation, by the one in device8_verify).

Device Data Storage

- **w25q128_manufacturer_id:** .byte 0 - Stores the manufacturer ID from JEDEC.
- **w25q128_device_type_id:** .byte 0 - Stores the device type ID from JEDEC.
- **w25q128_capacity_id:** .byte 0 - Stores the capacity ID from JEDEC.
- **received_buffer:** .space RAM_BUFFER_SIZE (256 bytes) - Used to store data read from flash memory, primarily for verification and output.
- **ram_buffer:** .space RAM_BUFFER_SIZE (256 bytes) - Used to store data input by the user (via Cmd 0x03) before it's written to flash, and as the reference for verification.

State Diagram



End of document