SEH500 Project Presentation: Assistive Communication Device

Presented by: Hadj Benseddik and Danial Ebadati

Introduction: Proof of Concept

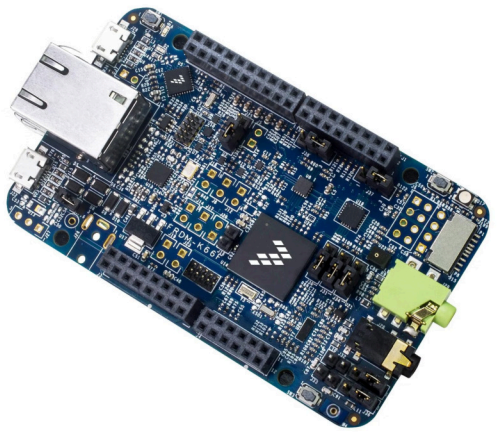
Prototype assistive communication device, designed as a proof of concept for non-verbal patients.

> Allows users to signal basic needs with simplicity and efficiency.

Project Purpose: To demonstrate fundamental embedded systems concepts, including General Purpose Input/Output (GPIO), interrupts, and efficient assembly programming.

Hardware Foundation

Built on the **NXP FRDM-K66F Microcontroller Board**, known for its robust capabilities in embedded applications.



Core Features

Two dedicated buttons: SW2 for water, SW3 for washroom.

Color-coded LED feedback: Green for water, Red for washroom.

Serial communication: Enables keyboard control for versatile interaction.

Future Vision: This prototype lays the groundwork for a battery-powered, remote-like device with customizable buttons, adapting to individual patient needs.

LED Setup Process: Enabling Visual Feedback

Green LED for water requests and a red LED for washroom alerts.

01

Enable Clock

Set the corresponding bit in the SIM_SCGC5 register. For Port E (Green LED), this is bit 13. For Port C (Red LED), it's bit 11.

02

Configure Pin Mode

Adjust the PORT_PCR register to GPIO mode (value 0x0100) for each LED pin, ensuring they function as general purpose I/O.

03

Set as Output

Configure the GPIO_PDDR register for output by setting the relevant bit. This directs the pin to drive output signals.

04

Initialize State

Set the GPIO_PDOR register to ensure LEDs are initially OFF, preventing unintended illumination at startup.

> **Why Assembly?**
>
> Direct register access through assembly programming offers efficiency and precise hardware control, critical for embedded systems.

Enable Port Example:

```
// Enable Port E clock
ldr r1, =0x40048038 // SIM_SCGC5
ldr r0, [r1]
orr r0, r0, #(1<<13) // Enable Port E (bit 13)
str r0, [r1]
```

Assembly Functions for LED Control

Key Functions

- setup_leds(): Initializes both Green and Red LEDs.
- func_green_led_on()/off(): Controls the green LED state.
- func_red_led_on()/off(): Controls the red LED state.

Operational Logic

- Reads current GPIO_PDOR register value.
- Modifies specific bits using **BIC** (Bit Clear) or **ORR** (Bit Set) instructions.
- Writes the modified value back to the register.

Active-Low Principle

A logical '0' (LOW) signal turns the LED ON, while a logical '1' (HIGH) signal turns it OFF.

Read-Modify-Write Safety

Only intended LED pin's state is altered, preserving the status of other pins on the same port.

Turning Green LED ON

```
func_green_led_on:
 ldr r1, =0x400FF100 // GPIOE_PDOR (Port E Data Output Register)
 ldr r0, [r1] // Load current value of GPIOE_PDOR into r0
 bic r0, r0, #(1<<6) // Clear bit 6 (making it LOW = LED ON)
 str r0, [r1] // Store modified value back to GPIOE_PDOR
 bx lr // Return from function
```

Periodic Interrupt Timer (PIT) Configuration for LED Flashing

The Periodic Interrupt Timer (PIT) is used for dynamic visual feedback, specifically the flashing of our alert LEDs. It generates precise, regular interrupts without requiring CPU polling.

1

Default Configuration

Obtain the default PIT configuration parameters to start.

2

Initialize PIT Peripheral

Initialize the PIT module with the retrieved default configuration.

3

Set Timer Period

Configure the timer to trigger an interrupt every 500,000 microseconds (500ms), defining the flashing rate.

4

Enable Interrupts

Activate interrupts for the specific PIT channel (e.g., PIT0).

5

Enable NVIC Interrupt

Enable the corresponding Nested Vectored Interrupt Controller (NVIC) interrupt for PIT0.

How It Works:

The timer counts down from a pre-set value. Once it reaches zero, it triggers an interrupt and automatically reloads to repeat the cycle, ensuring continuous flashing.

PIT Setup:

```
pit_config_t pitConfig;
PIT_GetDefaultConfig(&pitConfig);
PIT_Init(PIT, &pitConfig);
PIT_SetTimerPeriod(PIT, kPIT_Chnl_0,
USEC_TO_COUNT(500000U,
CLOCK_GetFreq(kCLOCK_BusClk)));
PIT_EnableInterrupts(PIT, kPIT_Chnl_0,
kPIT_TimerInterruptEnable);
EnableIRQ(PIT0_IRQn);
```

Button Setup Process: Enabling User Interaction

The responsiveness of our assistive device relies on correctly configuring button interrupts. This way system reacts instantly to user input —> Seamless experience for non-verbal patients.

Enable Clock

`CLOCK_EnableClock()` for Port D (SW2/Water) and Port A (SW3/Washroom) to power the respective GPIO ports.

Interrupt Type

Configure interrupt detection for falling-edge. This means an interrupt is triggered when the button is pressed (signal goes LOW).

NVIC Enable

Enable the interrupt in the Nested Vectored Interrupt Controller (NVIC) to allow the CPU to respond to button presses.

1

2

Pin Configuration

Set up the pins with a pull-up resistor, configure them for GPIO mode with fast slew rate.

3

4

GPIO Initialization

Initialize the pin as a digital input using `GPIO_PinInit()`, preparing it to receive signals from the button.

5

> **Why Interrupts?**
>
> Interrupts are crucial because they prevent the CPU from constantly polling button states. Instead, the CPU is notified immediately when a button is pressed = much more efficient and responsive system.

Button Setup:

```
CLOCK_EnableClock(kCLOCK_PortD);
PORT_SetPinConfig(PORTD, WATER_BUTTON_PIN,
&portConfig);
PORT_SetPinInterruptConfig(PORTD,
WATER_BUTTON_PIN, kPORT_InterruptFallingEdge);
GPIO_PinInit(WATER_BUTTON_PORT,
WATER_BUTTON_PIN, &gpioConfig);
EnableIRQ(PORTD_IRQn);
```

Interrupt Service Routines (ISRs)

Interrupt Service Routines (ISRs) allow the microcontroller to react to external events (button presses) and internal events (timer expirations) without delay.

PORTD_IRQHandler

Dedicated to handling interrupts from SW2 (water button) connected to Port D. It detects water requests.

PORTA_IRQHandler

Manages interrupts from SW3 (washroom button) connected to Port A, signaling washroom needs.

PIT0_IRQHandler

Handles interrupts from the 500ms Periodic Interrupt Timer, responsible for toggling LED states for flashing alerts.

Handler Process:

Clear Interrupt Flag First:

This critical step prevents the interrupt from being re-triggered immediately after processing, avoiding infinite loops.

Call Appropriate Alert Function:

Based on the interrupt source, the relevant alert handling function is called.

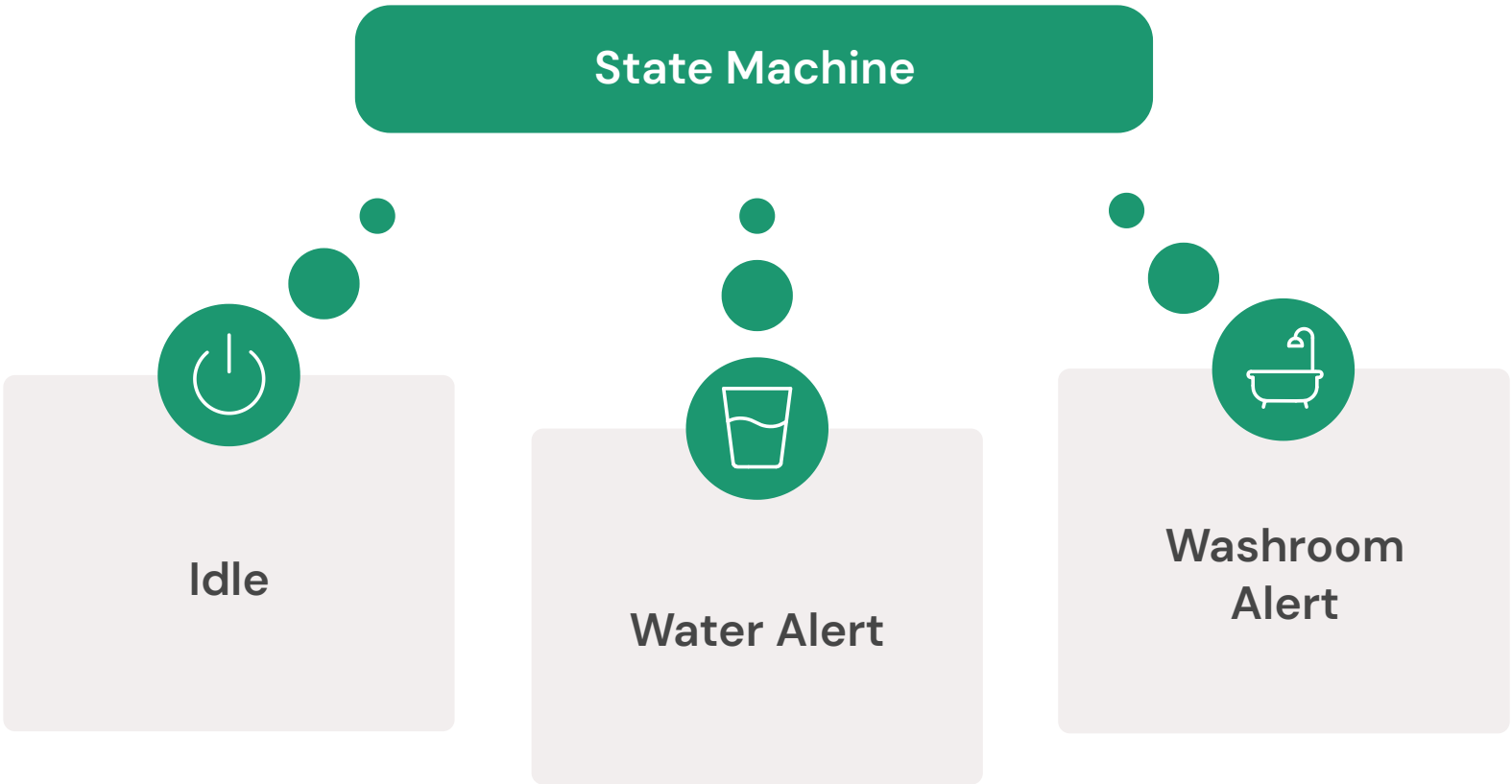
Return from Interrupt:

The CPU resumes normal operation.

Interrupt Handler (PORTD_IRQHandler):

```
void PORTD_IRQHandler(void) {
 GPIO_PortClearInterruptFlags(WATER_BUTTON_PORT, 1U << WATER_BUTTON_PIN);
 PRINTF("[BUTTON PRESS] SW2 pressed!\r\n");
 handle_water_alert();
}
```

State Machine Logic: System State Management



1	2	3
STATE_IDLE	**STATE_WATER_ALERT**	**STATE_WASHROOM_ALERT**
The default state where no active alert is present, and LEDs are off.	Activated when a water request is made. The green LED flashes, indicating an active water alert.	Activated for a washroom request. The red LED flashes, signifying an active washroom alert.

Mutual Exclusivity: A key design principle is that only one alert can be active at any given time for clarity and preventing confusion.

UART Serial Communication: Bidirectional Control and Debugging

Universal Asynchronous Receiver-Transmitter (UART) communication is integrated to provide both user control via a keyboard and essential debugging output.

Transmission

Debug messages and system status are sent to a connected computer, providing real-time insights into device operation.

Reception

The device receives keyboard input: 'W' for water alert and 'T' for washroom alert, allowing for remote or alternative control.

Configuration: Operates at a standard baud rate of 115200 —> reliable data transfer.

UART hardware generates interrupt when character arrives

Interrupt handler reads character and processes command

CPU can sleep between keyboard inputs (no polling needed)

Why Interrupts:

Polling would mean CPU constantly scanning for new characters, wasting CPU cycles. Interrupts allow CPU to sleep until data arrives.

UART Reception:

```
void UART0_RX_TX_IRQHandler(void) {
    if (statusFlags & kUART_RxDataRegFullFlag) {
        uint8_t ch = UART_ReadByte(uartBase);
        if (ch == 'W' || ch == 'w') {
            handle_water_alert();
        }
    }
}
```

System Architecture Overview: Complete Integration

The assistive communication device integrates diverse hardware and software components to form a cohesive, responsive system.

Buttons

User input for triggering requests.

UART

Bidirectional communication for control and debugging.

PIT Timer (500ms)

Generates periodic interrupts for LED flashing.

GPIO Interrupts

Efficiently detect button presses.

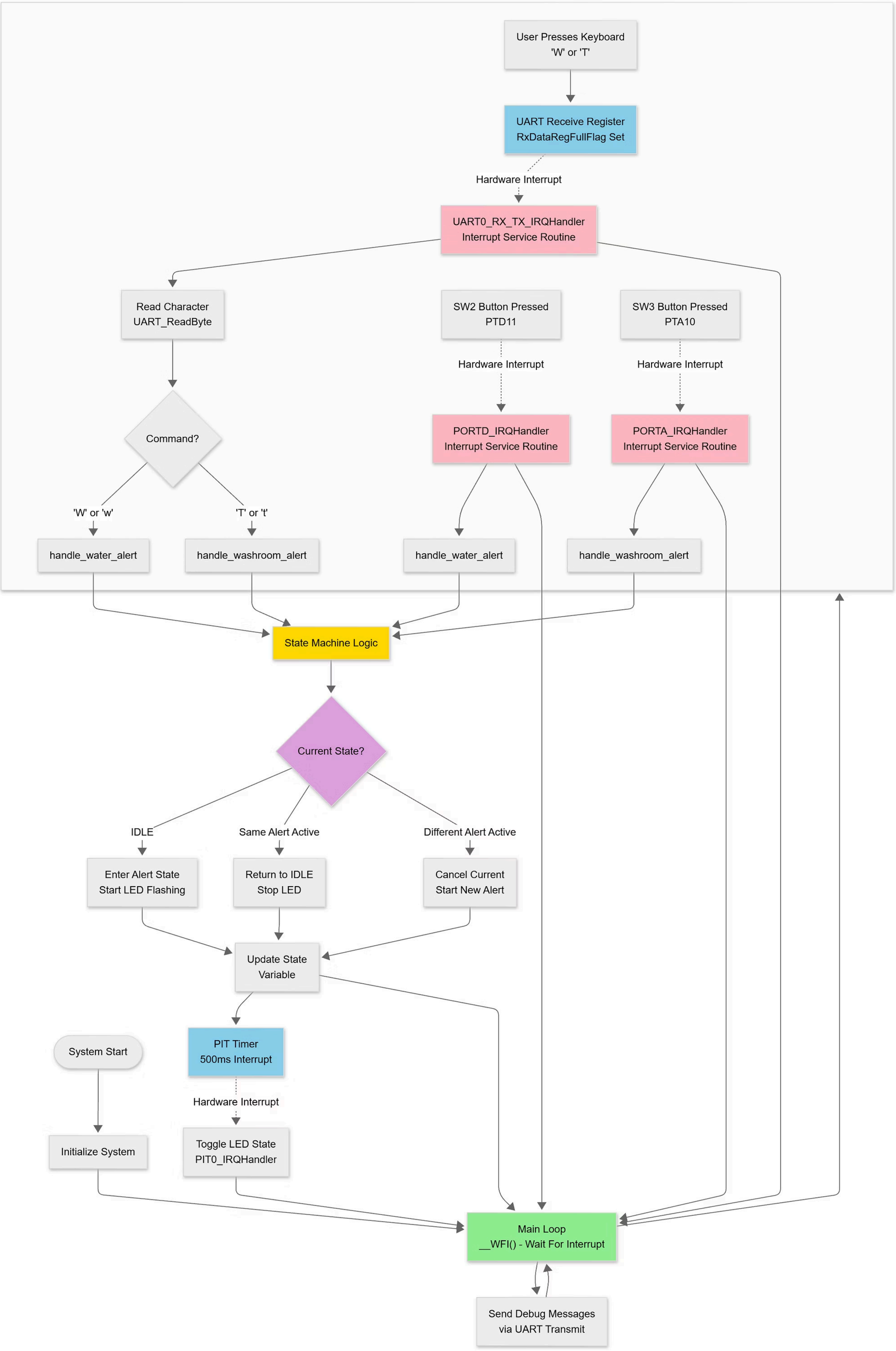
State Machine

Manages system behavior and alert states.

LED Control (Assembly)

Direct and efficient LED manipulation.

Data Flow Sequence:



Conclusion & Key Takeaways

We successfully developed a ***functional proof of concept*** for an assistive communication device, integrating hardware and software principles.

Project Highlights & Achievements

GPIO Configuration

Comprehensive pin setup and control

Interrupt Handling

Efficient interrupt-driven inputs

Assembly Programming

Low-level code >100 lines

State Machine

Predictable system logic design

Bidirectional UART

Control and debugging comms

Future Outlook

This lays a strong foundation for a scalable, portable, and battery-powered device with customizable input options.

> Would be tremendously helpful for non-verbal patient communication.

Find the complete codebase and detailed documentation on our [GitHub repository](#).