Embedded Systems Engineering - HAN University of Applied Sciences

MIC5 – RTOS

Dual-core FreeRTOS for Health Concept Lab Healtbot
(Semester 6 project)

Richard Kroesen (1663339)
4-25-2024

Contents

1. Introduction

The purpose of this document is to detail the information related to the final assignment for the course Microcontrollers 5 (MIC5). This course primarily focused on the application of real-time operating systems tailored for embedded devices, with a special emphasis on FreeRTOS. The aim of this document is to thoroughly record the processes and outcomes involved in the implementation of a demonstrative project, which serves to evaluate the students' proficiency in utilizing an embedded real-time operating system (RTOS).

1.1 Project Background

The project selected for this assignment is part of an ongoing initiative at the Health Concept Lab, where the student contributes to the Healthbot project. The overarching goal of this project is to conduct exploratory research in the field of social robotics, examining how embedded technologies can be integrated into healthcare settings to enhance eldery care and support healthcare providers.

The work encapsulated in this assignment represents a distinct and modified segment of the broader Healthbot research project. The emphasis is placed on the practical utilization of embedded RTOS, highlighting advanced techniques discussed throughout the course. Initially, the project builds upon the original codebase, which has been independently expanded to incorporate specific and advanced FreeRTOS features. This adaptation not only demonstrates the technical skills of the students but also showcases his ability to integrate and apply course concepts in a real-world scenario.

2. High-Level Design

On the high-level the abstract functionality is given and the architectural interconnection is illistrated. The objective of this section is to give a global system's overview without all the details.

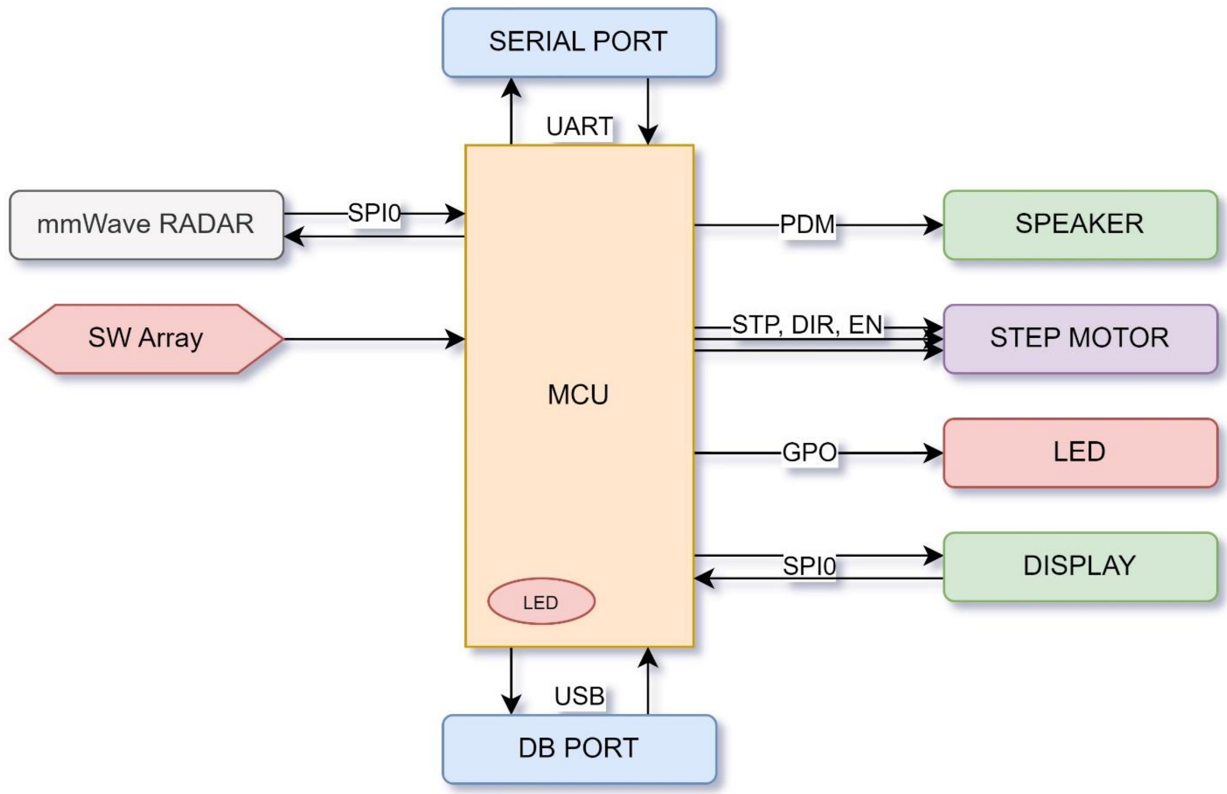


Figure 1 Functional Diagram

The high-level system design diagram illustrates the interconnections between various components and the microcontroller unit (MCU). The MCU serves as the central hub, interfacing with multiple peripherals through distinct communication protocols. The mmWave RADAR connects via the SPI0 interface for raw sensor signal sampling. A switch array inputs directly into the MCU, facilitating user interactions. The serial port utilizes UART for asynchronous communication, while the USB DB port provides additional data transfer capabilities. Audio feedback is managed by the speaker connected through the PDM interface. The step motor, controlled by signals for step, direction, and enable (STP, DIR, EN), allows for precise movement. Visual feedback is provided by the LED, controlled via a general-purpose output (GPO), and the display, also connected through SPI0, acts as the robotic eyes. This configuration ensures efficient communication and control across all system components, enabling comprehensive functionality and performance.

3. Scheduling Description

3.1 FreeRTOS Configuration

The configuration of FreeRTOS within this project is tailored to meet specific demonstrative needs, ensuring optimal functionality and performance balance. Key parameters have been adjusted or specifically chosen based on their critical role in the system's overall operation. These configurations are essential not only for achieving the desired behavior of the embedded system but also for maintaining system stability and efficiency.

This section will detail each configured parameter, highlighting its relevance and impact on the project.

Configuration Title	Parametric Specification	Description (& Argumentation)
Stack Size	128 of uint32_t	The stakc is configured for a minimal of 128x uin32_t. This is a general-used value and is for most tasks sufficient to hold the task's context and items (like local variables, overhead from function calls etcetra).
Total Heap Size	(145 * 1024)	This is enough to run the application stable, there was a need to expand this during the integration of the display. So, 148.480 kB which is below the total ram available of 256 kB is set here.
xPortSysTickHandler	isr_systick	FreeRTOS needs a stable clock source for operation, the system tick ISR is given as the source for this purpose.
Priority Levels	5	Five-level priority system effectively manages task execution. Levels 5 and 4 are reserved for critical functions like emergencies and resets. Level 3 supports essential real-time communications, while levels 2 and 1 handle regular and low-priority tasks, respectively.
Tick Rate (clk frequency)	125MHz	This is the default system tick clock frequency of the Raspberry Pi Pico.
Time-Slicing	True	Time-slicing is allowed, so that when a task is taking to much timing a higher priority task switch is possible inbetween.
Core Numbers	2	Out of curiosity to experiment with dual-core, which is supported on the chosen devkit: RPI-Pico, the system is configured for 2 cores.

Table 1 Configuration Settings

3.2 Tasks Overview

In this paragraph, all tasks which are used for MIC5 assessment are listed in Table 2.

Task Name	Task Functionality	Given Priority (0=lowest, 5=highest).	Core	Expected Stack Load (in n * uint32_t)
mmWave Radar Reader	For the raw sensor's 60GHz mmWave base-band signal sampling, a communication task is created. The external ADC on the sensor is controlled with commands in the registers through SPI protocol.	3	0	400
UART Asynchronous Communication	The standard UART functionality of the used pico SDK is a blocking communication.	RX-task: 3 TX-task: 1	0	400
PWM Audio Control	For UI feedback a PWM-audio output task is used with an audio example.	1	0	600
Display UI Controller	In each robot there will be 2 1.28 inch round displays. These two displays are going to act as controlled eyes.	2	1	900, below this value the display gets stuck.
Stepmotor Controller, by button pressed within time-out	The robots should also contain moveable components. The decision is made to create a which controls the stepper motor. For demonstration a button press counter with time-out task is created.	2	0	400
System Status	For generic administration and status, a system task is created. The system task contains a simple heartbeat LED and a flags checking.	1	1	Minimal (200)

Table 2 Project's Tasks Overview

3.3 Inter-Communication & Scheduling Techniques

The program employs various inter-communication and scheduling techniques to experiment with different FreeRTOS features, focusing on exploration rather than optimizing for the most beneficial methods. Two specific cases are illustrated in a diagram to demonstrate their functionality, but these principles are applied across multiple applications. For example, the mmWave Radar is used for presence detection and triggers a notification to a UART task upon detecting motion. Additionally, there are opportunities for enhancements through the implementation of daemons (background tasks), queues, and messages to further improve system efficiency and responsiveness.

3.3.1 UART Asynchronous Control

The provided diagram illustrates the inter-communication and scheduling techniques used for handling UART0 interrupt requests (IRQs) in a FreeRTOS-based system. The key components and their interactions are as follows:

UART0 IRQ:

TX (Transmit) and RX (Receive) sections handle the transmission and reception of UART data, respectively.

UART-TX-Done-Semaphore:

This semaphore is used to signal the completion of a UART transmission. Once the data is transmitted, the semaphore is released.

vUART-Sending Task:

This task is triggered by the UART-TX-Done-Semaphore. It sends a string via the UART (uart.send(string)). After sending, it prints the string to indicate successful transmission.

UART-RX-Queue:

This queue stores incoming UART data received by the RX section. It buffers the data until it can be processed by the receiving task.

vUART-Receive Task:

This task continuously checks if there is data available in the UART-RX-Queue (uart.checkAvailable()). When data is available, it reads the data from the queue (string = uart.readQueue()). It then sends the received data back via UART (uart.send(string)) to create an echo. Finally, it prints the echoed data to indicate successful reception and processing.

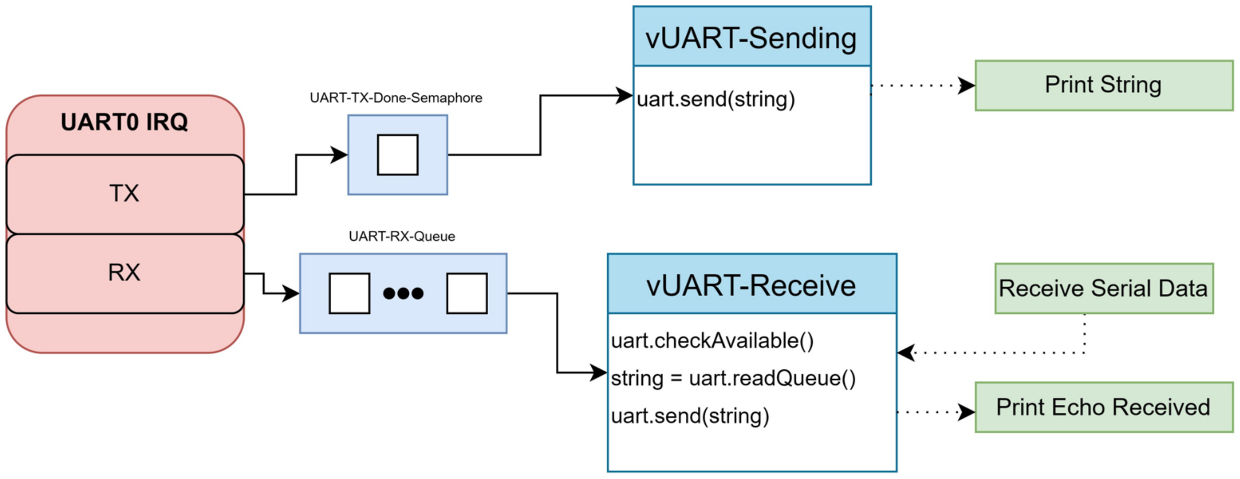


Figure 2 UART Driver RTOS inter-communication diagram

Data is transmitted via UART, a semaphore signals completion, and the sending task prints the transmitted string. Incoming data is queued, the receiving task processes it, sends an echo, and prints the received echo.

This diagram demonstrates efficient inter-task communication and scheduling using semaphores and queues in a FreeRTOS environment.

3.3.2 Event Driven switches to LED array

The provided diagram illustrates the process of handling GPIO interrupts and controlling an LED matrix based on the state of multiple buttons. The key components and their interactions are as follows:

GPIO IRQ:

5xBT-A State: This section monitors the state of five buttons (BT-A). When a button state changes, it triggers a GPIO interrupt request (IRQ).

BT-States-EventGroup:

This event group captures the state changes of the buttons. The interrupt handler sets bits in this event group to represent the current state of each button.

vSwitchHandler Task:

This task is responsible for handling the events from the BT-States-EventGroup. It sets the LEDs according to the bits in the event group, which reflect the current states of the buttons.

LEDS Matrix ON/OFF:

The vSwitchHandler task controls the LED matrix based on the button states. It turns the LEDs on or off as determined by the event group bits.

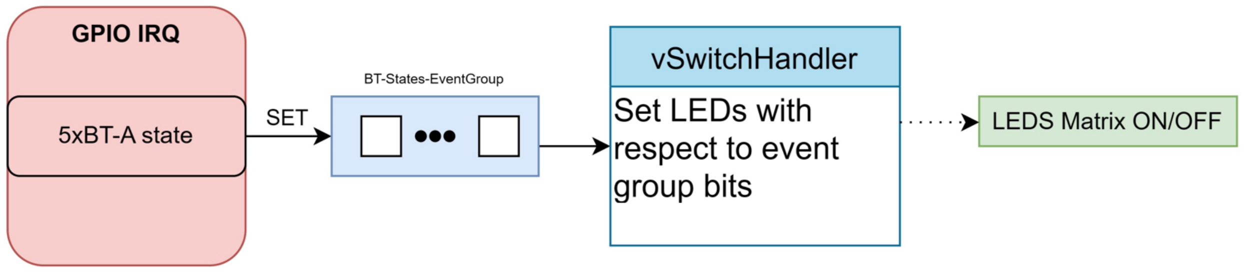


Figure 3 LED array RTOS event group demo

Changes in button states trigger GPIO interrupts. The GPIO IRQ sets bits in the BT-States-EventGroup to indicate button states. The vSwitchHandler task reads the event group bits and adjusts the LED matrix accordingly, turning LEDs on or off based on the button states.

This diagram demonstrates the use of event groups to manage button states and control an LED matrix in a FreeRTOS environment, showcasing effective inter-task communication and event handling.

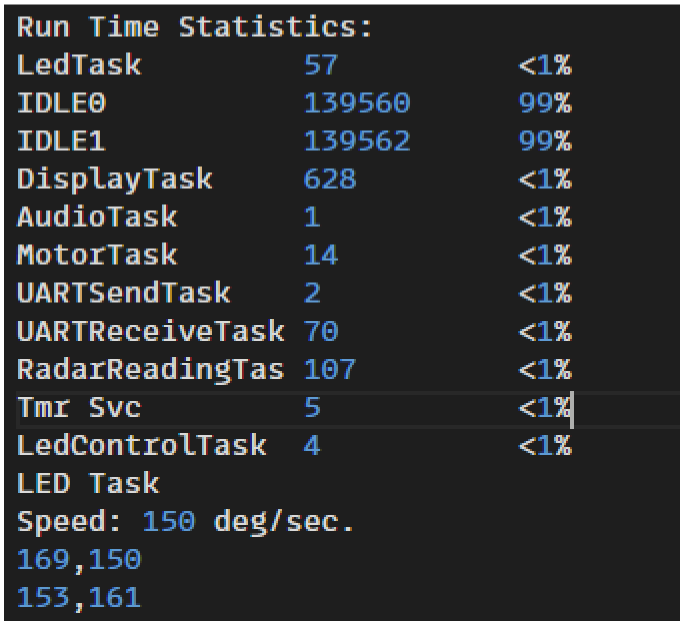
4. Verification & Validation

The verification and validation process is designed to be brief and practical. Due to time constraints imposed by other projects and the lack of interest in a comprehensive real-system validation, since this is a demo without valuable combined features a minimalistic approach has been adopted. Despite its simplicity, this approach aims to provide significant and indicative results. The focus is on ensuring that the system demonstrates the intended functionalities effectively, using essential checks and tests to validate key aspects of the implementation.

4.1 Run-Time Statistics

The program is configured to create run-time statistics, which give good indications in the system performance.

Out of the run-time statistics results the conclusion could be made that the system is efficient, even though there are various different features operational.



```
Run Time Statistics:
LedTask         57          <1%
IDLE0           139560      99%
IDLE1           139562      99%
DisplayTask     628         <1%
AudioTask       1           <1%
MotorTask       14          <1%
UARTSendTask    2           <1%
UARTReceiveTask 70          <1%
RadarReadingTas 107         <1%
Tmr Svc         5           <1%
LedControlTask  4           <1%
LED Task
Speed: 150 deg/sec.
169,150
153,161
```

Figure 4 13524-Logfile

Tasks are not all running on one core, only the IDLE state is given in the repsective cores. The following tables gives an overview of which tasks run on which core: see Table 2 Project's Tasks Overview.

4.2 Multiple Tasks Analysis

This section provides a concise summary of the analysis of multiple task switches. Figure 5 displays four distinct signals. The red signal represents a pulse-density-modulated 8-bit signal with an 11 kHz sampling rate, used to play a repetitive sample of music audio for demonstration purposes. The lowest signal is an enable line for the motor step driver hardware module.

The brown-colored pulses indicate the microstep rotations used by the hardware module (8 microsteps per motor step), requiring a minimum of 20 µs pulse duration and a 20 µs pause. In the implemented FreeRTOS system, these are configured for a pulse duration of 35 µs and a period of 250 µs. The configuration has been measured and verified using a logic analyzer.

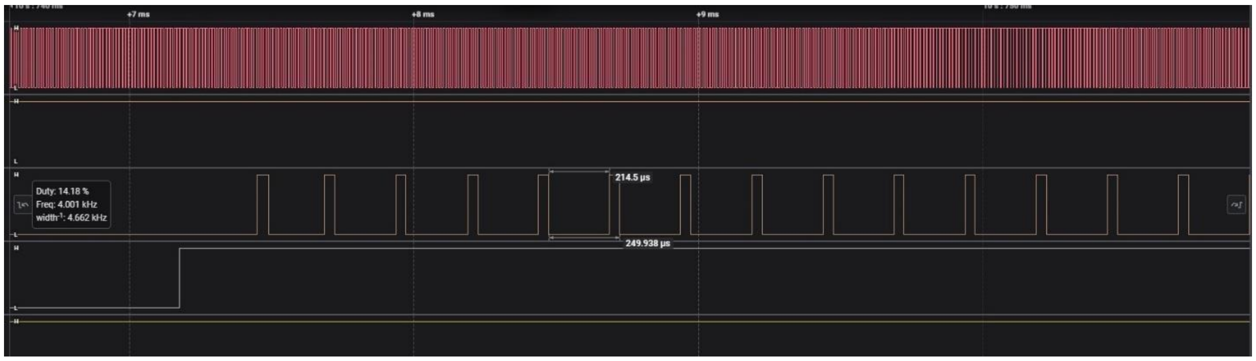


Figure 5 Logic-Analyzer measurement plot

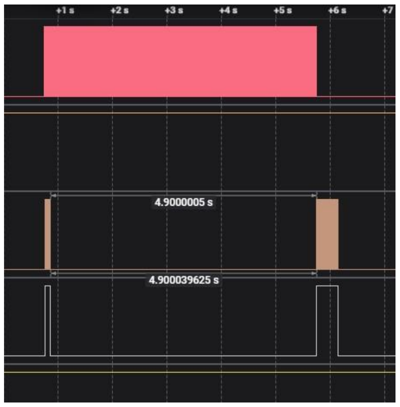
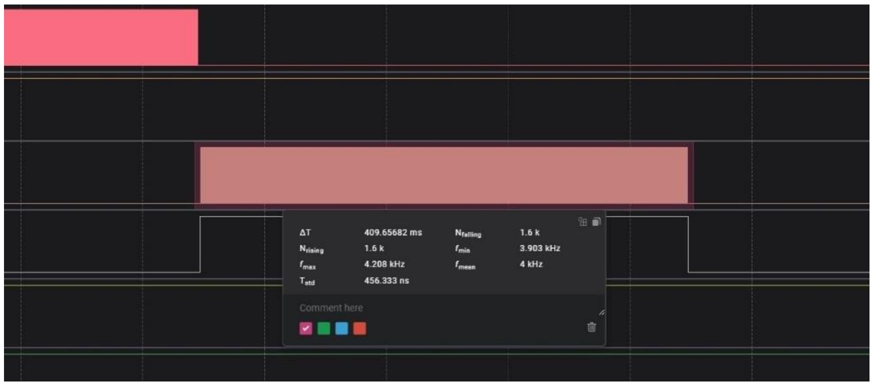


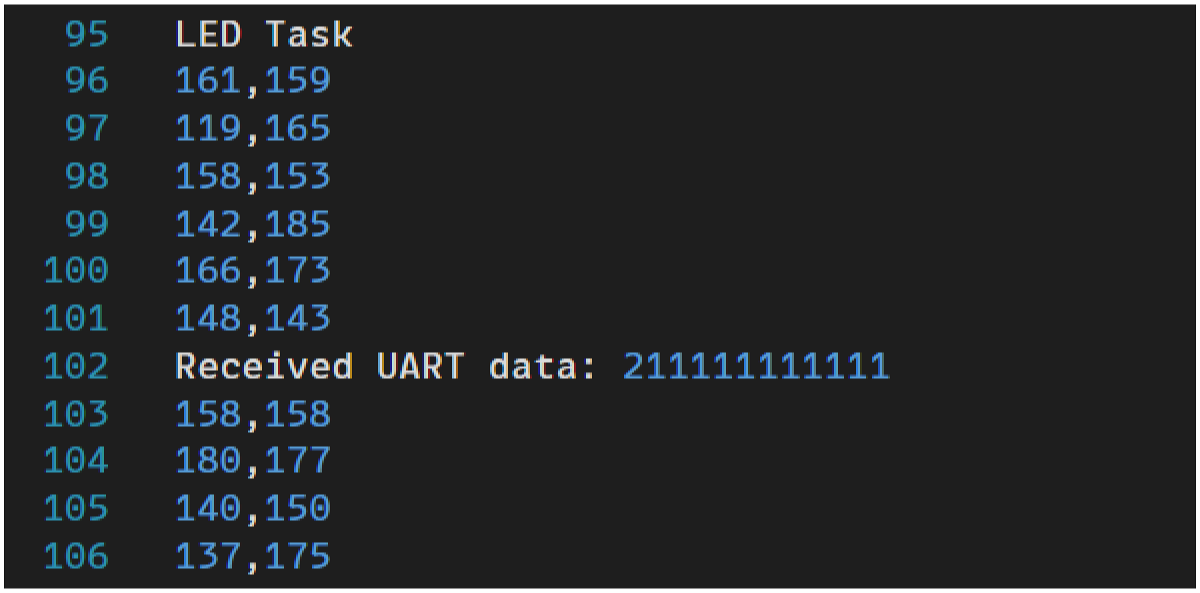
Figure 7 Timing between motion control



ΔT	409.65682 ms	$N_{falling}$	1.6 k
N_{rising}	1.6 k	f_{min}	3.903 kHz
f_{max}	4.208 kHz	f_{mean}	4 kHz
T_{std}	456.333 ns		

Figure 6 Steps validation

The motion control is configured to alternate between turning 90 degrees and 360 degrees, demonstrating the convenience of changing directions. Other task switches were verified by monitoring the serial output. Note that the saved log files lack timestamps; however, during monitoring, the serial terminal generated timestamps, which were used to determine if there were unacceptable time differences between tasks.



```
LED Task
161,159
119,165
158,153
142,185
166,173
148,143
Received UART data: 211111111111
158,158
180,177
140,150
137,175
```

Figure 8 Example of the serial monitor, .txt log export file

The provided serial log captures various tasks and data relevant to the MIC5 assessment. The log begins with the LED Task, which is responsible for generic administration and status, including a simple heartbeat LED and statistics printing. The subsequent lines capture task switches with numerical values, representing raw I & Q components of the mmWave radar. Additionally, the log shows received UART data, `211111111111`, indicating successful data reception through the UART Asynchronous Communication task.

This data, along with other task logs, provides essential information for evaluating task performance and system functionality, ensuring all tasks, such as the mmWave Radar Reader, PWM Audio Control, Display UI Controller, Stepmotor Controller, and System Status, operate within their expected parameters.