# Secure Timeout System NXP S32K3X8EVB

## Beamer for the CAOS Project

**Andrea Botticella**

**Fabrizio Emanuel**

**Elia Innocenti**

**Renato Mignone**

**Simone Romano**

February 5, 2025

Politecnico di Torino

# Table of Contents

# Project Overview



- The assignment consists of FOUR parts:
  - **Part 1: QEMU board emulation**
    - Generating a custom QEMU version to emulate the NXP S32K3X8EVB board.
    - Ensuring that QEMU emulates the proper CPU, memory map, and the peripherals assigned.
  - **Part 2: FreeRTOS porting**
    - Ensuring that FreeRTOS runs on the emulated board.
  - **Part 3: Writing a simple application**
    - Writing a simple application implementing different tasks to test the setup.

# Project Overview



- The assignment consists of FOUR parts:
  - **Part 4: Documentation and presentation**
    - Creating a tutorial to run and test your code.
    - Documentation of the code.
- What we've actually done:
  - *Secure Timeout System* application on the **NXP S32K3X8EVB board** using **FreeRTOS**, emulated with **QEMU**.
  - Refer to the dedicated markdown files in the repository: `README.md` and `GUIDE.md`. These files contain all the implementation details and the tutorial to replicate the project.

# Table of Contents

2 QEMU Board Emulation

# Custom QEMU Version



- Emulate the NXP S32K3X8EVB board, which is not natively supported by QEMU.
- Ensure proper emulation of the CPU, memory map, and peripherals.

# Resources Used



- Adding a new architecture to QEMU.
- Previous projects and repositories for reference.

# Implementation Details

- Board Initialization and Configuration:
  - Implement functions to load firmware, initialize memory regions, and handle hardware components.
  - Set up and configure hardware components like NVIC, LPUART, and PIT timers.
  - Manage system clocks and interrupts.

# Memory Regions Initialization



- Flash Memory:
  - Block0: Base Address: 0x00400000, Size: 2 MB
  - Block1: Base Address: 0x00600000, Size: 2 MB
  - Block2: Base Address: 0x00800000, Size: 2 MB
  - Block3: Base Address: 0x00AD0000, Size: 2 MB
  - Block4: Base Address: 0x10000000, Size: 128 KB
  - Utest: Base Address: 0x18000000, Size: 8 KB
- SRAM Memory:
  - Block0: Base Address: 0x20400000, Size: 256 KB
  - Block1: Base Address: 0x20440000, Size: 256 KB
  - Block2: Base Address: 0x20480000, Size: 256 KB
- DRAM Memory:
  - Base Address: 0x30000000, Size: 1 MB

# Hardware Components Setup



- NVIC (Nested Vectored Interrupt Controller):
  - Configured with 32 IRQs and 4 priority bits.
  - Connected to system clock and reference clock.
- LPUART (Low Power UART):
  - Base Address: 0x4006A000
  - Connected to NVIC and system clock.
- PIT Timers (Periodic Interrupt Timer):
  - Timer1: Base Address: 0x40037000
  - Timer2: Base Address: 0x40038000
  - Connected to NVIC and system clock.

# System Clocks and Interrupts



- System Clock:
  - Created clock object with 7.14ns period (140MHz frequency).
- Interrupt Handling:
  - Configured NVIC to handle interrupts.
  - Linked NVIC's memory access to system memory.

# Firmware Loading



- Function: `s32k3x8_load_firmware`
- Parameters:
  - `cpu`: The ARM CPU instance.
  - `ms`: The machine state.
  - `flash`: The memory region representing the flash memory.
  - `firmware_filename`: The filename of the firmware to be loaded.
- Functionality:
  - Reads the firmware file and loads its contents into the specified flash memory region.

# Initialization Functions



- `s32k3x8_initialize_memory_regions`:
  - Initializes flash, SRAM, and DRAM memory regions.
- `s32k3x8_init`:
  - Initializes the system, including memory regions, NVIC, LPUART, and PIT timers.

# Table of Contents

3 FreeRTOS Porting

- To test the **FreeRTOS Porting** on **QEMU**, a very simple application was created.
- The application runs a basic task that prints a message every second.
- If everything works correctly, it means that the **FreeRTOS Porting** has been successfully implemented.

# Setting Up FreeRTOS

1. **Cloning** the FreeRTOS repository.
2. Creating the directory **structure**: `App/` and `App/Peripherals/`.
3. Creating and implementing the following files in the `App/` directory:
   - `s32_startup.c`, `s32_linker.ld`
   - `FreeRTOSConfig.h`
   - `Makefile`
   - `main.c`
   - `Peripherals/`: `uart.c`, `printf-stdarg.c` with their respective header files

- main.c:

```c
xTaskCreate(vTask1, "Task1", configMINIMAL_STACK_SIZE, NULL,
    mainTASK_PRIORITY, NULL);

void vTask1(void *pvParameters)
{
    (void) pvParameters;

    for (;;)
    {
        printf("Task1 is running...\n");
        vTaskDelay(1000);
    }
}
```

- Run the **Test**:
  - `cd App && make run`


Figure: FreeRTOS Porting Test.

# Table of Contents

# Secure Timeout System Application



- The application is a simple implementation of a *Secure Timeout System*.
- It consists of **multiple tasks** that simulate events, monitor activities, and handle alerts.
- **Hardware timers** are used to generate **periodic interrupts** for activity detection.

# Task Implementation



- **Event Task:**
  - Periodically generates events that can be either user activities or suspicious activities.
  - Uses a pseudo-random number generator to decide the type of event.
  - Logs the generated event and updates the respective counters.

```
[EVENT SIMULATOR] ──── New Cycle Started ──────────────
[EVENT SIMULATOR] Generated: User Activity    | Count: 1
```

```
[EVENT SIMULATOR] ──── New Cycle Started ──────────────
[EVENT SIMULATOR] Generated: Security Event   | Count: 1
```

Figure: Generation of a user activity and a suspicious activity.

# Hardware Timer Initialization



- **Timer 0:**
  - Configured to generate periodic interrupts.
  - Interrupt handler checks for **user activities** and sets the user activity detection flag.
- **Timer 1:**
  - Configured to generate periodic interrupts.
  - Interrupt handler checks for **suspicious activities** and sets the suspicious activity detection flag.

# Task Implementation



- **Monitor Task:**
  - Checks for user activity detection.
  - Logs the status of user activity.
  - Resets the user activity detection flag after logging.
- **Alert Task:**
  - Checks for suspicious activity detection.
  - Logs the status of the system security.
  - Initiates security protocols if suspicious activity is detected.
  - Resets the suspicious activity detection flag after logging.

- **Global Variables:**
  - Four main **flags**:
    - `userActivity`, `userActivityDetection`, `suspiciousActivity`, `suspiciousActivityDetection`
- **Task Priorities:**
  - Event Task has the highest priority to ensure timely event generation.
  - Monitor Task and Alert Task have lower priorities.
- **Timer Frequency:**
  - Timer 0 and Timer 1 are configured to generate periodic interrupts at a frequency of 2 Hz.

# Implementation Details

- **Task Priorities:**

```c
// filepath: /App/SecureTimeoutSystem/secure_timeout_system.c
#define MONITOR_TASK_PRIORITY (tskIDLE_PRIORITY + 2)
#define ALERT_TASK_PRIORITY   (tskIDLE_PRIORITY + 3)
#define EVENT_TASK_PRIORITY   (tskIDLE_PRIORITY + 4)
```

- **Timer Frequency:**

```c
// filepath: /App/Peripherals/IntTimer.c
#define tmrTIMER_0_FREQUENCY (2UL)
#define tmrTIMER_1_FREQUENCY (2UL)
```

# Run Example

## 4 Test Application

```
[EVENT SIMULATOR] ────── New Cycle Started ──────────────────
[EVENT SIMULATOR] Generated: Security Event   | Count: 1

Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
[SECURITY ALERT] Suspicious activity detected | Status: ALARM
[SECURITY ALERT] Initiating security protocols...
[USER MONITOR] No activity                    | Status: IDLE
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
[SECURITY ALERT] Suspicious activity detected | Status: ALARM
[SECURITY ALERT] Initiating security protocols...
[USER MONITOR] No activity                    | Status: IDLE
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
[SECURITY ALERT] Suspicious activity detected | Status: ALARM
[SECURITY ALERT] Initiating security protocols...
[USER MONITOR] No activity                    | Status: IDLE
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
[SECURITY ALERT] Suspicious activity detected | Status: ALARM
[SECURITY ALERT] Initiating security protocols...
[USER MONITOR] No activity                    | Status: IDLE
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
[SECURITY ALERT] Suspicious activity detected | Status: ALARM
[SECURITY ALERT] Initiating security protocols...
[USER MONITOR] No activity                    | Status: IDLE
Timer 0 Interrupt: looking for user activities...
Timer 1 Interrupt: looking for suspicious activities...
```

# Table of Contents

# Overview

## 5 Memory Protection Unit (MPU) Implementation

- text

# Table of Contents

# Conclusion



- The `s32k3x8evb_board.c` file plays a crucial role in the emulation of the **NXP S32K3X8EVB board** within **QEMU**.
- It provides the necessary functions to load firmware, initialize memory regions, set up hardware components, and manage system clocks and interrupts.
- All the implementations and detailed information about the project are contained in the repository.
- Repository link: `https://baltig.polito.it/caos2024/group2.git`

*Thank you for listening!*
*Any questions?*