QEMU Support for NXP S32K3X8EVB-Q289

Developed as part of the **Computer Architectures and Operating Systems** course (Cybersecurity MSc) at Politecnico di Torino by **group 27**

Motivation

- Testing on physical boards is **time-consuming and costly**.
- QEMU can provide a **fast, flexible, and scalable** testing environment.
- Developers **must** be able to run **real-world ELF binaries** developed using official NXP-provided tools on QEMU, so there is **no need to modify existing code**.

Technical specifications & Quality requirements

- **Board:** NXP S32K3X8EVB-Q289, an evaluation board for the S32K3 family designed for automotive applications.
- **MCU:** NXP S32K358 (32-bit Arm® Cortex®-M7)
- **Peripherals and requirements:**
  - **UART** support
  - **DMA** integration
  - **Memory Protection Unit** (MPU), with support for 16 regions
  - Must work with **official libraries** using FW developed with the S32 Design Studio IDE provided by NXP
  - Must run **FreeRTOS** and AUTOSAR-based projects

Board and SoC Overview

- **Clocks**: static configuration, with `sysclk` at 160MHz, `refclock` at 20MHz and LPUART clocks described in the UART section.
- **Memory regions**: as per the S32K3 reference manual
- **Interrupts and priorities**: Support for **240 interrupt channels** with **16 priority levels** (4-bit).
- **Memory Protection Unit** (MPU): Full support for **16 MPU regions**, as per the S32K3 reference manual, instead of the default 8 regions.
- **Peripherals**: Support for all **16 LPUART channels** and **eDMA** (NXP's enhanced DMA) for data transfers with **32 channels**.
- **Documentation**: Board definition, Board implementation, SoC definition, SoC implementation

UART Support

- Officially named **LPUART** (Low Power UART), it is a **low-power, high-performance** serial communication module.
- All 16 channels are supported (`LPUART0` to `LPUART15`), with the related registers and fixed clock configurations (`AIPS_PLAT_CLK` for `LPUART{0, 1, 8}` and `AIPS_SLOW_CLK` for the others).
- **Baud rate** configuration based on each channel's clock.
- Standard **UART operations** for data transmission and reception (i.e. handling of the `DATA` register).
- **Interrupt handling** for UART communication.
- **Documentation**: Definition, Implementation

DMA Integration

- **eDMA** (NXP's enhanced DMA) support for data transfers:
  - **Memory-to-memory** transfers configured using **TCD** (transfer control descriptor) registers.
  - **Interrupts** for transfer completion (`INTMAJOR`, i.e. `CITER==0`) and for half transfer (`INTHALF`, i.e. `CITER >= BITER/2`) completion.
  - Partial support for some complex transfer types too (e.g. **scatter-gather**) but no implementation of SMLOE, DMLOE, BWC or other features.
- On the real board, enables **efficient peripheral communication** by offloading data transfer tasks from the CPU.
- **Documentation**: eDMA definition, TCD Definition, Implementation

Compatibility

- Designed for **FreeRTOS** and **AUTOSAR-based projects**.
- Works with projects created using **NXP S32 Design Studio**.
- Compatible with **NXP's official libraries** and tested with the following:
    - Basic components: **BaseNXP**, **Siul2_Port**
    - DMA: **Dma_Ip**
    - UART: **Lpuart_Uart**
    - Interrupts: **IntCtrl_Ip**
    - OS: **FreeRTOS** package provided by NXP
- **No need to modify** linker scripts, startup files or IDE generated code, which means that the **existing codebase** can be used without any changes, provided that it leverages implemented peripherals and features only.

Test Firmware

- **Demo application** showcasing:
  - **MPU support** (implicit, activated by defining `MPU_ENABLE`).
  - **DMA and UART functionality**.
  - **Interrupt handling** for both DMA (explicit, showing an ISR implementation) and UART (implicit, managed by NXP's libraries).
  - **FreeRTOS scheduling**.
- Entirely based on the previously mentioned **NXP libraries**.
- Functionality: creation of **multiple tasks**, of which one generates data, puts it in a buffer and **triggers a DMA transfer**, while another task waits for the transfer to complete (using a **semaphore**, unlocked by the **DMA ISR**), verifies that the buffers are identical and logs information using **UART**.

Demonstration

- Live demo of **compiling an ELF binary** in S32 Design Studio and **running it** in QEMU.
- Example UART communication output.
- DMA transfer simulation results.

Benefits for Developers

- **Accelerates testing** without needing the physical board.
- Enables **CI/CD pipelines** for embedded development.
- Supports **early software development** before hardware availability.

Conclusion & Next Steps

- Our QEMU implementation can significantly **improve the development workflow**.
- Since our code is modular and adheres to the **QEMU coding standards**, leveraging the QEMU Object Model, it can be **easily extended** and support for missing features and peripherals can be added.
- Would you consider using our version of QEMU for your projects?

Let’s discuss how we can integrate this into your workflow!

Thank you!

CAOS group 27

C. F. Vescovo, C. Sanna, F. Stella