Group 10: NXP S32K3X8EVB in QEMU

Implementation and FreeRTOS porting

Master’s Degree in Computer Science: Embedded Systems

Francesco Mignone Leonardo Gallina Silvia Bonenti Andrea Baraldi

Lorenzo Parata

Politecnico di Torino

In this presentation, we will discuss the implementation of the NXP S32K3X8EVB board in QEMU and the porting of FreeRTOS to this architecture.

This is a collaborative effort by Francesco Mignone, Leonardo Gallina, Silvia Bonenti, Andrea Baraldi, and Lorenzo Parata as part of the Opereting Systems course.

Table of Contents

1 Project Goals

Project Assignment

1 Project Goals

The main goals of this project are:

- Implement the NXP S32K3X8EVB board in QEMU
  - CPU: ARM Cortex-M7
  - Peripherals: UART, SPI
- Port FreeRTOS to the newly implemented architecture
- Test the implementation with an application that utilizes the board's peripherals
- Document the entire process and provide a comprehensive report

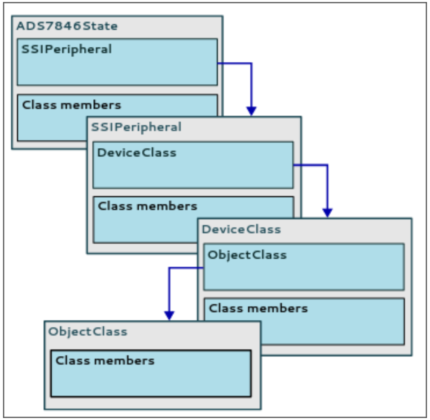
Table of Contents

2 QEMU Structure

Introduction to QEMU structure

Even if QEMU is a complex software, its structure is quite simple and modular:

- Written in C with extra feature that mimics C++ classes
- It is composed by a core and several modules organized in parents and children
- The core provides the main functionalities, while the modules provide support for different architectures and devices
- The main components of QEMU are:
  - CPU emulation
  - Memory management
  - Device emulation
  - I/O handling



Basic QEMU Module

Each module in QEMU has two main phases:

- Initialization: where the module is registered and its properties are set
- Realization: where the module interacts with other components and performs its functions

Table of Contents

3 QEMU Implementation

NXP S32K3X8EVB

The Board has been divided in two components for modularity:

- MCU: with CPU and peripherals (UART, SPI)
- Main Board: mounts the MCU and can be used to attach other components in the future

Board Declaration

The MCU includes:

- ARM Cortex-M7 CPU
- 512KB SRAM
- 1MB Flash memory
- UART peripherals
- SPI peripherals

MCU Initialization

```
struct S32K3x8State{
    SysBusDevice parent_obj;
    ARMv7MState cpu;
    Clock *sysclk;
    uint32_t sram0_size;
    uint32_t flash0_size;
    MemoryRegion sram0;
    MemoryRegion flash0;
    MemoryRegion flash_alias;
    MemoryRegion *board_memory;
    MemoryRegion container;
    S32K3x8UartState uart[NXP_NUM_UARTS];
    S32K3x8SPIState spi[NXP_NUM_SPI];
};
```

UART

The UART peripheral has been implemented with the following features:

- Basic configuration (baud rate, data bits, stop bits, parity)
- Data transmission and reception
- Interrupt handling

All 16 UARTs have been implemented, but only UART0 has been tested, since the code is the same, with base address `0x40328000`

MCU Initialization

```
struct S32K3x8UartState {
    SysBusDevice parent_obj;
    MemoryRegion mmio;
    uint32_t usart_sr;
    uint32_t usart_dr;
    uint32_t usart_brr;
    uint32_t usart_cr1;
    uint32_t usart_cr2;
    uint32_t usart_cr3;
    uint32_t usart_gtpr;
    CharBackend chr;
    qemu_irq irq;
};
```

SPI

The SPI peripheral implementation includes:

- Basic configuration and initialization
- Data transmission and reception
- Interrupt handling

Since there is no effective device at the end of the SPI bus, the implementation is done by saving the written data and returning incremented by one on read.

MCU Initialization

```
struct S32K3x8SPIState {
    SysBusDevice parent_obj;
    MemoryRegion mmio;
    uint32_t spi_cr1;
    uint32_t spi_cr2;
    uint32_t spi_sr;
    uint32_t spi_dr;
    uint32_t spi_crcpr;
    uint32_t spi_rxcrcr;
    uint32_t spi_txcrcr;
    uint32_t spi_i2scfgr;
    uint32_t spi_i2spr;
    uint32_t test_var;
    qemu_irq irq;
    SSIBus *ssi;
};
```

Table of Contents

4 FreeRTOS Porting

FreeRTOS Porting

The porting of FreeRTOS to the NXP S32K3X8EVB architecture involved several steps:

- Setting up the development environment
- Configuring FreeRTOS for the ARM Cortex-M7 architecture
- Testing and debugging the ported FreeRTOS on the QEMU-emulated board

Development Environment

The development environment was set up with the following tools:

- GCC toolchain for ARM
- FreeRTOS source code
- Makefile for building and linking the application

Peripheral test: UART

The test application for the UART peripheral includes:

- Initialization of the UART peripheral
- Sending data
- Printing messages to the terminal

The application is structured as a FreeRTOS task that runs independently.

MCU Initialization

```
int main(int argc, char **argv) {
...
    UART_init();
...
}
...
void UART_test(void *pvParameters) {
(void)pvParameters;
uart_printf("Starting UART Test\n");
uart_printf("Hello from UART of Group10!\n"
uart_printf("Ending UART Test\n");
vTaskDelete(NULL);
}
```

Peripheral test: SPI

The test application for the SPI peripheral includes:

- Initialization of the SPI peripheral
- Sending and receiving data
- Printing messages to the terminal

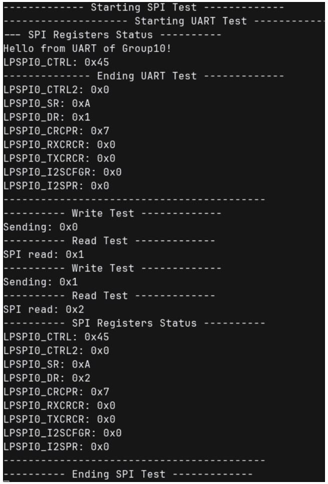
Also this application is structured as a FreeRTOS task that runs independently.

MCU Initialization

```
int main(int argc, char **argv) {
    SPI_init();
}
void SPI_test(void *pvParameters) {
  (void)pvParameters;
  SPI_status();
  uint32_t pippo = 0x00;
  uint32_t pluto = 0x00;
  for (int i=0; i<10; ++i) {
    uart_printf("Sending: %x\n", pippo);
    SPI_write((uint8_t)pippo);
    SPI_get((uint8_t *)&pluto);
    pippo = pluto;
    uart_printf("SPI read: %x\n", pluto);
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
```

Teast Output

4 FreeRTOS Porting



```
------------- Starting SPI Test -------------
------------------- Starting UART Test ----------
--- SPI Registers Status ----------
Hello from UART of Group10!
LPSPI0_CTRL: 0x45
------------- Ending UART Test -------------
LPSPI0_CTRL2: 0x0
LPSPI0_SR: 0xA
LPSPI0_DR: 0x1
LPSPI0_CRCPR: 0x7
LPSPI0_RXCRCR: 0x0
LPSPI0_TXCRCR: 0x0
LPSPI0_I2SCFGR: 0x0
LPSPI0_I2SPR: 0x0
------------------------------------------
---------- Write Test -------------
Sending: 0x0
---------- Read Test -------------
SPI read: 0x1
---------- Write Test -------------
Sending: 0x1
---------- Read Test -------------
SPI read: 0x2
---------- SPI Registers Status ----------
LPSPI0_CTRL: 0x45
LPSPI0_CTRL2: 0x0
LPSPI0_SR: 0xA
LPSPI0_DR: 0x2
LPSPI0_CRCPR: 0x7
LPSPI0_RXCRCR: 0x0
LPSPI0_TXCRCR: 0x0
LPSPI0_I2SCFGR: 0x0
LPSPI0_I2SPR: 0x0
------------------------------------------
---------- Ending SPI Test -------------
```

Table of Contents

5 Summary

Conclusions

5 Summary

In conclusion, the implementation of the NXP S32K3X8EVB board in QEMU and the porting of FreeRTOS to this architecture were successful. The main achievements include:

- Successful implementation of the NXP S32K3X8EVB board in QEMU, including CPU and peripherals (UART, SPI)
- Porting of FreeRTOS to the ARM Cortex-M7 architecture
- Development and testing of applications that utilize the board's peripherals
- Comprehensive documentation of the entire process

Future work could involve adding more peripherals, improving the existing implementations, and exploring additional features of FreeRTOS.

Group 10: NXP S32K3X8EVB in QEMU

Thank you for your attention!