Politecnico di Torino

1859

Design and Implementation of a Virtual S32K3X8EVB Board in QEMU

Master Degree in Computer Engineering (Embedded Systems)

Operating Systems for Embedded Systems

Authors:
s338247 - Rebecca Burico
s336721 - Yuqi Li

Table of Contents

Project overview

The aim of the project is to create a **virtual model** of the **NXP S32K3X8EVB** board using **QEMU**, and to **test its functionality** by **porting FreeRTOS** and running a demo application.

Key objectives:

- QEMU board emulation
  - Create a custom QEMU version to emulate the **NXP S32K3X8EVB** board following the reference manual
  - Ensure that QEMU emulates the correct CPU, memory mapping and assigned peripherals (LPUART and LPSPI)

Project overview

- FreeRTOS porting
  - Ensure that FreeRTOS runs on the emulated board
- Writing a simple application
  - Writing a simple application implementing different tasks to test the setup
- Documentation and presentation
  - Creating a document with full details of the project and a tutorial on how to run and test the code

Table of Contents

Implementation Detail: Board

```c
static const TypeInfo s32k3x8evb_type = {
    .name = MACHINE_TYPE_NAME("s32k3x8evb"),
    .parent = TYPE_MACHINE,
    .instance_size = sizeof(S32K3X8EVBState),
    .class_init = s32k3x8evb_class_init,
};
```

- Machine is registered in QEMU with the name *s32k3x8evb*

```c
/* Register machine type*/
static void s32k3x8evb_machine_init(void)
{
    qemu_log_mask(CPU_LOG_INT, "Registering S32K3X8EVB machine type\n");
    type_register_static(&s32k3x8evb_type);
}
```

- Board specifications:
  - ARM **Cortex-M7** CPU
  - 8MB **Flash memory**, 768KB **SRAM**, 256KB **DTCM**, and 128KB **ITCM**
  - **NVIC** with **240 interrupts** and **4 priority bits**
  - Multiple peripherals: 16 **LPUART**, 6 **LPSPI**
  - System clock running at 160 MHz

Implementation Detail: Board

- Memory Mapping
  - Create and map memory regions
  - ELF Firmware to Flash *(0x00400000)*
  - Vector Table set to ITCM *(0x00000000)*

memory_region_init_ram() / init_rom()
memory_region_add_subregion()

- Peripherals
  - Create — *qdev_new()*
  - Connect — *sysbus_mmio_map()* *sysbus_connect_irq()*

Table of Contents

Implementation Detail: LPSPI

- 6 LPSPI instances with base addresses:
  - *0x4035C000*
  - *0x40360000*
  - *0x40364000*
  - *0x404BC000*
  - *0x404C0000*

Implementation Detail: LPSPI

- FIFO management:
  - TX and RX FIFOs implemented as circular buffer
  - Watermark level trigger flags in the status register (SR)

fifo_push()/fifo_pop()

- Data transfer (loopback)
  - When writing in TX FIFO, same data in RX FIFO

s32k3x8_lpspi_do_transfer()

- Status and interrupts

s32k3x8_lpspi_update_status()
s32k3x8_lpspi_update_irq()

- Memory-Mapped I/O
  - QEMU reads/writes registers
  - Handles all SPI config and data registers

s32k3x8_lpspi_read()
s32k3x8_lpspi_write()

Implementation Detail: LPSPI

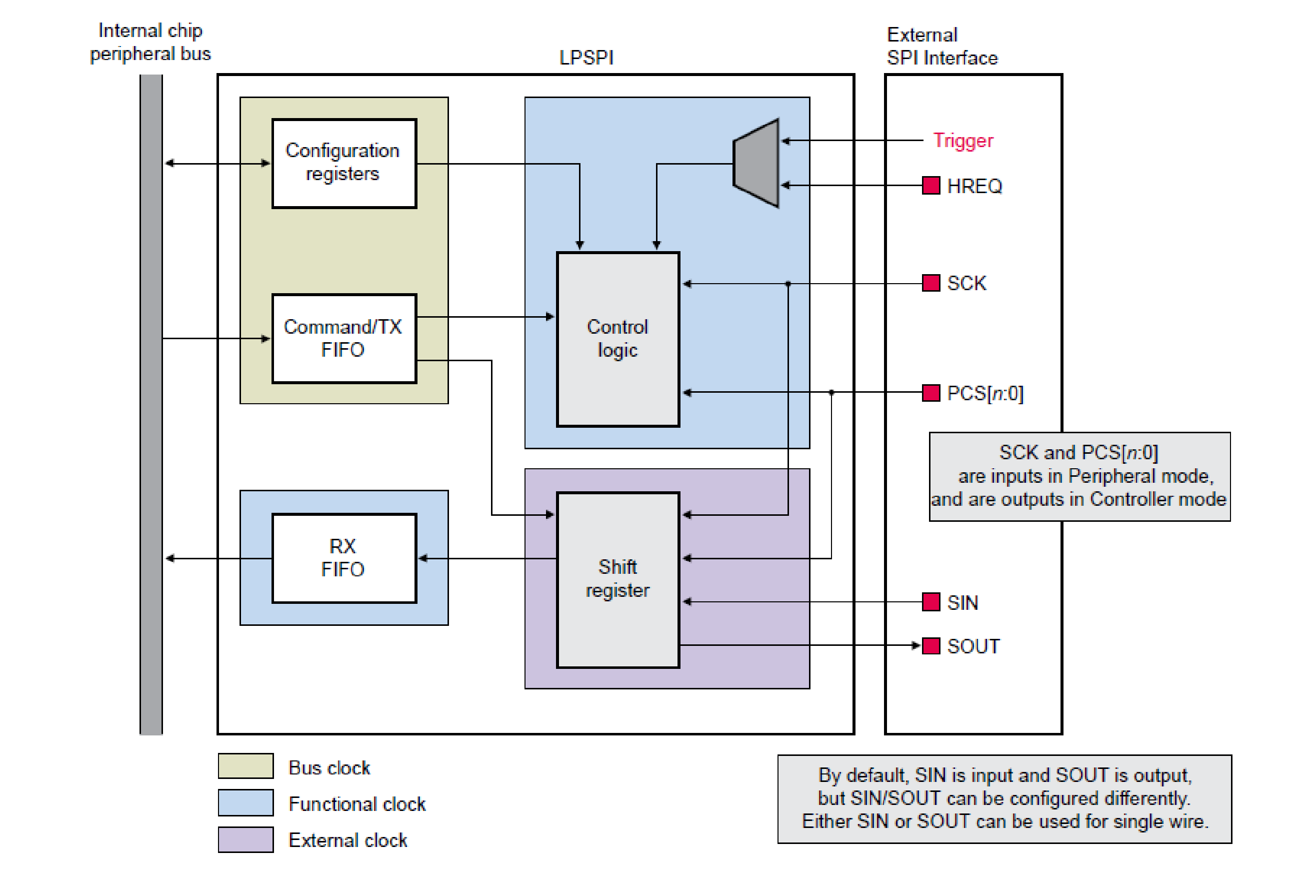

Table of Contents

Implementation Detail: LPUART

- LPUART instances
  - Number: 16
  - Base address: 0x40328000

- LPUART 4 parts
  - Mapping I/O
  - FIFO management
  - I/O data transmission method
  - Character backend integration

MemoryRegionOps

fifo_push()/fifo_pop()...

Interrupt mode and polling mode

CharBackend chr docking QEMU character device

Implementation Detail: LPUART

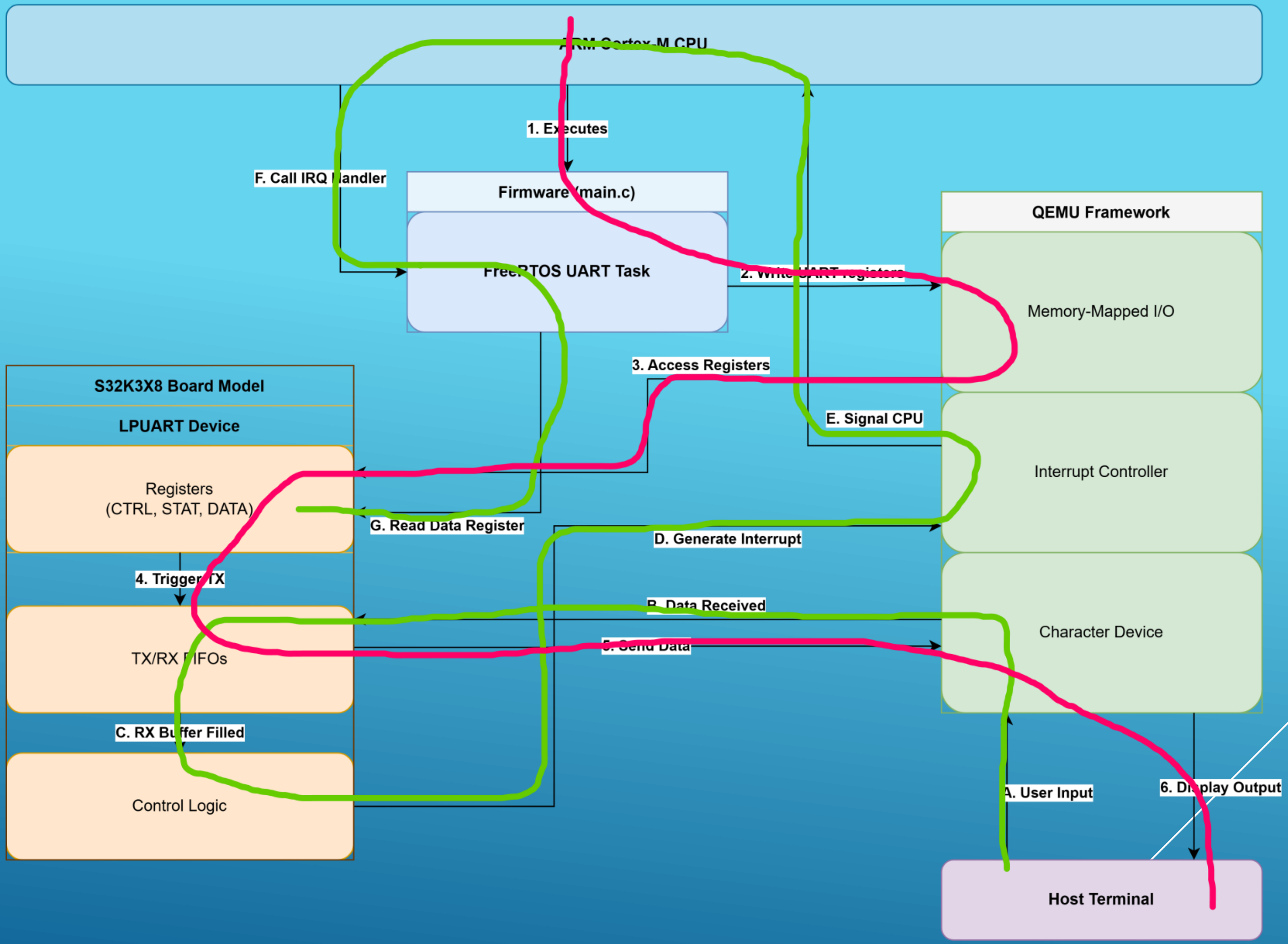

Table of Contents

FreeRTOS Porting and Demo

- FreeRTOS porting
  - FreeRTOS interrupt functions in startup_ARMCM7.c
    - SVCall_Handler = vPortSVCHandler
    - PendSV_Handler = xPortPendSVHandler
    - SysTick_Handler = xPortSysTickHandler
  - configUSE_PREEMPTION = 1
  - configCPU_CLOCK_HZ = 160MHz
  - configTICK_RATE_HZ = 1000
  - configPRIO_BITS = 4
  - configKERNEL_INTERRUPT_PRIORITY = 240

First task run

vTaskStartScheduler();

Task Switching

Time will not move forward

vTaskDelay(1000);

FreeRTOS Porting and Demo

- **Demo**
  - **TxTask1 & TxTask2** – basic communication tasks

```
>>> Starting scheduler...
 Hello task2 from FreeRTOS running in QEMU on S32K3X8!
 Hello task1 from FreeRTOS running in QEMU on S32K3X8!
```

  - **UartStatusTask** – Uart registers status monitoring

```
>>> UART Status Check Task Started
|======================================================|
 UART Status Check #1 STAT Register: 0x00C00000
 Status Bits:
  TDRE: SET (TX Ready)
  TC: SET (TX Complete)
  RDRF: CLEAR
 FIFO Register: 0x00C00088
|------F UART status check!----------------------------|
```

Bit[23] TXEMPT = 1: TX FIFO Empty
Bit[22] RXEMPT = 1: RX FIFO Empty

110000000000000 10001000

$0x88 = 10001000_2$
TXFE: TX fifo enable
RXFE: RX fifo enable

TX System Ready
-Can accept new data now
-Previous send completed
-Hardware buffer clean

RX System Idle
-No data backlog
-Buffer space available
-Ready for new data

FreeRTOS Porting and Demo

- **Demo**
  - **SimpleSpiTestTask**
    - Register initial value read(SPI Status check)

```
s->verid = 0x02000004;      /* Version ID */
s->param = 0x00040202;      /* Parameters: 4 PCS, 4-word FIFO */
s->cr = 0x00000000;
s->sr = 0x00000001;         /* TDF=1 */
```

    - Status Flags

```
/* TDF: Transmit Data Flag */
if (s->tx_fifo.level <= txwater) {
    s->sr |= LPSPI_SR_TDF;
} else {
    s->sr &= ~LPSPI_SR_TDF;
}
```

You can write more

    - SPI Loopback Test Start

Send data == Receive data → Show "OK“
Send data != Receive data → Display "FAIL"

```
|=======================================================|
| SPI TEST #1 |
|=======================================================|
 SPI Status Check
VERID:   0x02000004
PARAM:   0x00040202
CR:      0x00000001 (ENABLED)
SR:      0x00000001
Status Flags:
  TDF: SET (TX Ready)
  RDF: CLEAR
  WCF: CLEAR
  FCF: CLEAR
  TCF: CLEAR
  MBF: CLEAR
CFGR1:   0x00000001 (MASTER)
TCR:     0x00000007
FSR:     0x00000000

SPI Loopback Test Start
Test 1: TX=0xAA RX=0xAA PASS
Test 2: TX=0x55 RX=0x55 PASS
Test 3: TX=0x12 RX=0x12 PASS
Test 4: TX=0x34 RX=0x34 PASS
Test 5: TX=0x56 RX=0x56 PASS
Test 6: TX=0x78 RX=0x78 PASS
Test 7: TX=0x9A RX=0x9A PASS
Test 8: TX=0xBC RX=0xBC PASS
:Test Results: 8 PASS, 0 FAIL
|------F SPI test !-----------------------------------|
```

Table of Contents

Conclusions

- Successfully implemented a **virtual S32K3X8EVB board** in QEMU and tested its correct functioning by porting FreeRTOS and running a simple application
- Our virtual board is now a practical tool for real-time software prototyping and testing.

Thank you for listening!