IOB-SoC

A RISC-V-based System on Chip

José T. de Sousa

IObundle Lda

June 26, 2020

Outline

- Introduction
- Project setup
- Project editing guidelines
- Getting started with the timer IP hardware design
- Edit the hardware definitions file `system.vh`
- Instantiate the timer IP in file `rtl/src/system.v`
- Add the timer IP firmware
- Edit file `firmware.c` to drive the timer IP
- Edit the Makefile to compile the timer IP firmware
- Setup RTL simulation using the Icarus Verilog simulator
- Simulate the system
- Conclusions and future work

Introduction

- Building processor-based systems from scratch is challenging
- The IOB-SoC template eases this task
- Provides a base Verilog SoC equipped with
  - a RISC-V CPU
  - a memory system including boot ROM, RAM and AXI4 interface to DDR
  - a UART communications module
- Users can add IP cores and software to build more complex SoCs
- Here, the addition of a timer IP and its software driver is exemplified

Project setup

- Use a Linux machine or VM
- Install the latest stable version of the open source Icarus Verilog simulator (`iverilog.icarus.com`)
- Make sure you can access `github.com` using an ssh key
- At `github.com` create your SoC repository (*my-soc*) using `github.com/IObundle/iob-soc` as a template
- Follow the instructions in the README file to clone the repository in your Linux machine

Create an IP core to instantiate in your SoC

- Create a timer IP core or alternatively use a ready-made one and iclude it as a git submodule: at the `iob-soc` root type

  ```
  git submodule add git@github.com:IObundle/iob-timer.git submodules/iob-timer
  ```

- Edit the following lines in the `./system.mk` file to configure the system as explained in the next slide

Edit the ./system.mk configuration file to declare the new peripheral

```
#FIRMWARE
FIRM_ADDR_W:=13

#SRAM
SRAM_ADDR_W=13

#DDR
USE_DDR:=0
RUN_DDR:=0
DDR_ADDR_W:=30

#BOOT
USE_BOOT:=0
BOOTROM_ADDR_W:=12

#Number of peripherals
N_SLAVES:=2

#Peripheral IDs (assign serially: 0, 1, 2, etc)
UART:=0
TIMER:=1

#Peripheral paths
UART_DIR=$(SUBMODULES_DIR)/iob-uart
TIMER_DIR=$(SUBMODULES_DIR)/iob-timer
```

Edit the `hardware.mk` and `software.mk` configuration files to import the peripheral's hardware and software

`./hardware/hardware.mk`

```
include $(ROOT_DIR)/system.mk

# submodules
include $(TIMER_DIR)/hardware/hardware.mk
include $(CPU_DIR)/hardware/hardware.mk
...
```

`./software/software.mk`

```
define:=-D
include $(ROOT_DIR)/system.mk

#submodules
include $(TIMER_DIR)/software/software.mk
include $(INTERCON_DIR)/software/software.mk
```

Instantiate the timer IP in file `rtl/src/system.v`

```
`timescale 1ns/1ps
`include "system.vh"

module system (

    ...

    time_counter #(.COUNTER_WIDTH(32))
    timer (
            rst(reset_int),
            clk(clk),
            .addr(m_addr[2]),
            .data_in(m_wdata),
            .data_out(s_rdata[`TIMER_BASE]),
            .valid(s_valid[`TIMER_BASE]),
            .ready(s_ready[`TIMER_BASE])
     );

    ...
endmodule
```

Edit the `firmware.c` file to drive the new peripheral

`./software/firmware/firmware.c`

```
#include "system.h"
#include "iob-uart.h"
#include "iob_timer.h"

#define UART (UART_BASE<<(DATA_W-N_SLAVES_W))
#define SOFT_RESET (SOFT_RESET_BASE<<(ADDR_W-N_SLAVES_W))
#define TIMER (TIMER_BASE<<(ADDR_W-N_SLAVES_W))

int main()
{

  //read timer cycle count
  int cycles = timer_get_count(TIMER_BASE);

  uart_init(UART,UART_CLK_FREQ/UART_BAUD_RATE);

  uart_printf("Hello world!\n");

  uart_txwait();

  //read current timer count and compute elapsed time in clock cycles
  cycles =  timer_get_count(TIMER) - cycles;

  //print the elapsed time and clock frequency
  uart_printf("Execution time: %dus @%dMHz,115200BAUD\n",(time*1000000)/UART_CLK_FREQ);

  uart_txwait();
  return 0;
}
```

Simulate the system

- To simulate the system just type
  `make`
- The firmware, bootloader and system verilog description are compiled as you can see from the printed messages
- The last prints should look like the following

```
IOb-SoC Bootloader:

Reboot CPU and run program...

Hello world!
Total execution time: 1262 us @100MHz
```

- Congratulations! You have created your first RISC-V system using IOB-SoC.

Main memory options

- The main memory type options are enabled by defining or undefining the `USE_DDR` and `USE_BOOT` macros
- Option 1: place the firmware image in FPGA memory during design compilation
  - Do not define either macro: not reprogrammable
  - This option is only valid for FPGA: needs recompilation if firmware changes
- Option 2: place the firmware in internal RAM
  - Define macro `USE_BOOT` only: reprogrammable
  - This option is valid for FPGA and ASIC
  - Firmware can be (re)loaded via UART
- Option 3: place the firmware in external DDR memory
  - Define both macros: reprogrammable
  - This option is valid for FPGA and ASIC
  - Firmware can be (re)loaded via UART
  - Third party DDR controller IP core is required

Conclusions and future work

- Conclusions
  - Presented project and editing guidelines
  - Designed of a peripheral IP (timer)
  - Instantiated peripheral in the system
  - Designed a simple software driver for the peripheral
  - Compiled the software
  - Simulated the system's RTL code running the software in the memories
  - Presented options for the main memory
- Future work
  - Non volatile (flash) external memory support
  - Real Time Operating System (RTOS)