IOB-SoC

A RISC-V-based System on Chip

José T. de Sousa

IObundle Lda

June 28, 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 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 repository or, alternatively, use the one at `github.com:IObundle/iob-timer.git`
- An IP core can be integrated in an IOb-SoC if it provides the following 2 files:
  1. hardware/hardware.mk
  2. software/embedded/embedded.mk
- Add the IP core repository as a git submodule of your IOb-SoC repository:
  `git submodule add git@github.com/IObundle/iob-soc submodules/TIMER`
- To configure the system to host the IP core, edit the `./system.mk` file as 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

#Peripheral list (must match respective submodule name)
PERIPHERALS:=UART TIMER
```

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"

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;
}
```

Run the firmware in internal SRAM

1. Run the firmware in internal RAM and disable (re)programming
   - Assign `USE_DDR=0` and `USE_BOOT=0`
   - Loading programs after the FPGA is programmed is disabled: if the firmware is modified the FPGA must be recompiled
   - This option is only valid for FPGA which permits memory initialisation
2. Run the firmware in internal RAM and enable (re)programming
   - Assign `USE_DDR=0` `USE_BOOT=1`
   - Loading programs after the FPGA is programmed is enabled
   - This option is valid for FPGA and ASIC
   - Firmware is (re)loaded via UART

Run the firmware in external DDR

1. Run the firmware in external DDR and disable (re)programming
   - Assign `USE_DDR=1` and `USE_BOOT=0`
   - This option is only allowed in simulation which permits memory initialisation
   - An FPGA or ASIC implementation will not work
2. Run the firmware in external DDR memory and enable (re)programming
   - Define `USE_DDR=1 USE_BOOT=1`
   - This option is valid for FPGA and ASIC
   - Firmware is (re)loaded via UART
   - Third party DDR controller IP core is required

Simulate and implement 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
```

- To implement in your chosen FPGA just type `make fpga`
- To implement in your chosen ASIC just type `make fpga`
- To load the firmware in the hardware just type `make load-firmware`

Conclusions and future work

- Conclusions
  - A tutorial on SoC creation using IOb-SoC is presented
  - The addition of a peripheral IP core (timer) is illustrated
  - A simple software driver for the IP core is exemplified
  - How to compile and run the system explained
  - Options for implementing the main memory are presented
- Future work
  - Non volatile (flash) external memory support
  - Real Time Operating System (RTOS)