IOB-SoC

Tutorial: Create a RISC-V-based System on Chip

IObundle Lda

August 9, 2020

Outline

- Introduction
- Project setup
- Instantiate an IP core in your SoC
- Edit file `firmware.c` to drive the new peripheral
- Simulate IOb-SoC
- Run IOb-SoC in FPGA
- 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 core 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 have push/pull access to github.com using an ssh key
- Clone the repository github.com/IObundle/iob-soc
- Follow the instructions in its README file

Instantiate an IP core in your SoC

- The Timer IP core at `www.github.com/IObundle/iob-timer.git` will be used as an example
- Add the Timer IP core repository as a git submodule of your IOb-SoC repository:

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

- Add the timer IP core to the list of peripherals in the `./system.mk` file:

  ```
  PERIPHERALS:=UART TIMER
  ```

- An IP core can be integrated into IOb-SoC if it provides the following files:
  - hardware/hardware.mk
  - software/software.mk
- Study these files and its references in the Time IP core repository to learn how to make your own peripheral cores.

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

`./software/firmware/firmware.c`

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

int main()
{
  unsigned long long elapsed;
  unsigned int elapsedu;

  //read current timer count, compute elapsed time
  elapsed  = timer_get_count(TIMER_BASE);
  elapsedu = timer_time_us(TIMER_BASE);

  //init uart
  uart_init(UART_BASE, FREQ/BAUD);

  uart_printf("\nHello world!\n");

  uart_txwait();

  uart_printf("\nExecution time: %d clocks in %dus @%dMHz\n\n",
              (unsigned int)elapsed, elapsedu, FREQ/1000000);

  uart_txwait();
  return 0;
}
```

Simulate IOb-SoC

- The following assumes you are using the default Icarus Verilog simulator installed locally (see the README.md file)
- To add your own simulator, add a directory into `./hardware/simulation`, using the existing simulation directories as examples
- To run the simulation with the firmware pre-initialised in the memory: `make sim INIT_MEM=1`
- The firmware and bootloader C codes and the system's Verilog description are compiled as you can see from the printed messages
- During the simulation itself, the following is printed:

```
IOb–SoC Bootloader:

Reboot CPU and run program...

Hello world!

Execution time: 6583 clocks in 66us @100MHz
```

Run IOb-SoC in FPGA

- To compile and run your SoC in one of our FPGA boards, contacts us at info @ iobundle . com. The following assumes your are using our default Intel Cyclone V GT Development Kit
- To add your own FPGA board, add a directory into `./hardware/fpga`, using the existing board directories as examples
- To compile the FPGA design with the firmware pre-initialised in the memory:

  ```
  make fpga INIT_MEM=1
  ```
- To load the hardware design in the FPGA:

  ```
  make fpga-load
  ```
- To run your firmware in the FPGA:

  ```
  make run-hw INIT_MEM=1
  ```
- You may change the firmware afterwards and use the previous command to recompile and reload the firmware via UART

Conclusions

- A tutorial on creating a simple SoC using IOb-SoC has been presented
- The addition of an example peripheral IP core has been illustrated
- A simple software driver for the IP core has been described
- Simulation of IOb-SoC hs been explained
- Compiling and running IOb-SoC in FPGA has been explained