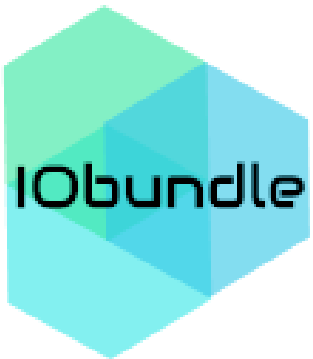
IOb-SoC

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

IObundle Lda.

July 2, 2021



Outline

- Introduction
- Project setup
- Instantiate an IP core in your SoC
- Write the software to drive the new peripheral
- Simulate IOb-SoC
- Run IOb-SoC in an FPGA board
- Conclusion

Introduction

- Building processor-based systems from scratch is challenging
- The IOb-SoC template makes it easy by providing a base Verilog SoC equipped with
  - a RISC-V CPU
  - a memory system including boot ROM, RAM, 2-level cache system and AXI4 interface to external memory (DDR)
  - a UART communications module
  - an example firmware
- Users can add IP cores and software to build their SoCs
- This tutorial exemplifies the addition of a timer IP core and the use of its software driver

Project setup

- Use a Linux real or virtual machine
- Install the latest **stable** version of the open source Icarus Verilog simulator (`iverilog.icarus.com`)
- Set up **ssh** access to Github (`github.com`) (https access requires password many times) key
- Follow the instructions in this repository's README file to clone the repository and install the tools

Instantiate an IP core in your SoC

- The Timer IP core at github.com/IObundle/iob-timer.git is used here as an example
- Add the Timer IP core repository as a git submodule of your IOb-SoC clone 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:
  - CORE_REPO/hardware/hardware.mk
  - CORE_REPO/software/embedded.mk
- Study these files and its references in the Timer IP core repository.

Write the software to drive the new peripheral

Edit the `firmware.c` file to look as follows

```
#include "system.h"
#include "periphs.h"
#include "iob-uart.h"
#include "printf.h"

#include "iob_timer.h"

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

  //init timer and uart
  timer_init(TIMER_BASE);
  uart_init(UART_BASE, FREQ/BAUD);

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

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

  printf("\nExecution time: %d clock cycles\n", (unsigned int) elapsed);
  printf("\nExecution time: %dus @%dMHz\n\n", elapsedu, FREQ/1000000);
}
```

Simulate IOb-SoC

- Run the simulation with the firmware pre-initialised in the memory: `make sim`
- You should see from the printed messages that the firmware and bootloader C files, and the system's Verilog files are being compiled
- Then the simulation is started and the following should be printed:

```
IOb-Bootloader: USE_DDR=0 RUN_EXTMEM=0
IOb-Bootloader: Restart CPU to run user program...

Hello world!

Execution time: 2190 clock cycles

Execution time: 23us @100MHz
```

Run IOb-SoC on an FPGA board

- To compile and run your SoC in one of our FPGA boards, contacts us at info@iobundle.com.
- To compile and run your SoC on your FPGA board, add a directory into `./hardware/fpga`, using the existing board directories as examples
- Then issue the following command:
  `make run BOARD=<board_dir_name> INIT_MEM=0`
  This will compile the software and the hardware, produce an FPGA bitstream, load it to the device, load the firmware binary using the UART (`INIT_MEM=0` prevents the FPGA memory initialisation), start the program and direct the standard output to your PC terminal.
- If you change only the firmware and repeat the above command, only the firmware will be recompiled, reloaded and rerun

Conclusion

- 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 firmware that uses the IP core driver functions has been explained
- IOb-SoC has been simulated at the Verilog level while running the firmware
- IOb-SoC has been compiled and the firmware run on an FPGA board