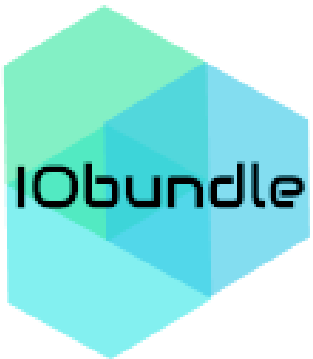
IOb-SoC

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

IObundle Lda.

July 1, 2021



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, cache system 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 **ssh** push/pull access to `github.com` (htps access is not enough) key
- Visit the repository at `github.com/IObundle/iob-soc`
- Follow the instructions in its README file to clone the repository ans install the tools

Instantiate an IP core in your SoC

- The Timer IP core at `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/embedded.mk
- Study these files and its references in the Timer IP core repository.

Edit the `firmware.c` file to look as follows and drive the new peripheral

`./software/firmware/firmware.c`

```
#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

- The following assumes the Icarus Verilog simulator is installed locally and that the SIMULATOR variable is set as SIMULATOR=icarus (see the README.md file)
- Run the simulation with the firmware pre-initialised in the memory: `make sim`
- The firmware and bootloader C files, and the system's Verilog files are compiled as you can see from the printed messages
- During the simulation itself, the following is 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 in FPGA

- To compile and run your SoC in one of our FPGA boards, contacts us at info@iobundle.com.
- To add your own FPGA board, add a directory into `./hardware/fpga`, using the existing board directories as examples
- To compile and run the FPGA design for a set of parameters:
  `make run INIT_MEM=0 RUN_EXTMEM=1`
  This will compile the software and the hardware, produce an FPGA image, load the hardware and firmware to the board, and direct the standard output to your PC.
- To recompile and sent only the firmware to the board via UART:
  `make run INIT_MEM=0 RUN_EXTMEM=1`

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 has been explained
- Compiling and running IOb-SoC on FPGA has been explained