

IOb-SoC

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

IOb-SoC

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

IObundle Lda.

April 6, 2022



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 on an FPGA board
- Conclusion



Introduction

- Building processor-based systems from scratch is challenging
- The IOb-SoC template eases this task by providing a base Verilog SoC equipped with
 - a RISC-V CPU
 - a memory system including boot ROM, RAM, 2-level cache system and an AXI4 interface to external memory (DDR)
 - a UART communications module
 - an example firmware program
- 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 (see the README file to download a VM)
- Make sure a **stable** version of the open source Icarus Verilog simulator (iverilog.icarus.com) is installed locally or on some remote server
- Make sure you have FPGA build tools installed locally or on some remote server
- Make sure you have an FPGA board attached to your Linux machine or to some remote server
- Set up **ssh** access key to GitHub (github.com) (using https will ask for your password many times)
- Follow the instructions in the IOb-SoC 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
```

- Update the Timer IP core submodules:

```
git submodule update --init --recursive
```

- Add the Timer IP core to the list of peripherals in the ./config.mk file:

```
PERIPHERALS:=UART TIMER
```

- Include the Timer IP core hardware.mk file in ./hardware.mk file:

```
include $(TIMER_DIR)/hardware/hardware.mk
```

- Include the Timer IP core embedded.mk file in ./firmware/Makefile file:

```
include $(TIMER_DIR)/software/embedded/embedded.mk
```



Instantiate an IP core in your SoC

- Include the Timer IP core `pc.mk` file in `./pc-emul/Makefile` file:

```
include $(TIMER_DIR)/software/pc-emul/pc.mk
```

- An IP core can be integrated into IOB-SoC if it provides the following files:
 - `CORE_REPO/hardware/hardware.mk`
 - `CORE_REPO/software/embedded/embedded.mk`
 - `CORE_REPO/software/pc-emul/pc.mk`
- Study these files and its references in the Timer IP core repository.



Write the software to drive the new peripheral

Edit the software/firmware/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);

    uart_finish();
}
```



Simulate IOb-SoC

- Run the simulation with the firmware pre-initialised in the memory:
`make sim`
- The printed messages show that the firmware and bootloader C files are compiled
- The whole system's Verilog files are also compiled
- Finally the simulation is started and the following is printed:

```
TESTBENCH: connecting..IOb-Bootloader: connected!  
IOb-Bootloader: Restart CPU to run user program...
```

```
Hello world!
```

```
Execution time: 4356 clock cycles
```

```
Execution time: 44us @100MHz
```

```
TESTBENCH: exiting
```



Run IOb-SoC on an FPGA board (1)

- 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/<toolchain>`, where `<toolchain>` is either `*vivado*` or `*quartus*`. Use 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



Run IOb-SoC on an FPGA board (2)

- When running IOb-SoC on an FPGA with the default settings and the firmware pre-initialised in the memory, the following should be printed:

```
+-----+  
|               IOb-Console               |  
+-----+
```

```
BaudRate = 115200  
StopBits = 1  
Parity    = None
```

```
IOb-Console: connecting ...
```

```
IOb-Bootloader: connected!
```

```
IOb-Bootloader: Restart CPU to run user program...
```

```
Hello world!
```

```
Execution time: 114466 clock cycles
```

```
Execution time: 1145us @100MHz
```

```
IOb-Console: exiting ...
```



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
- RTL simulation of the system running the firmware has been demonstrated
- FPGA board running of the system running the firmware has been demonstrated

