Design of an FPGA-based IP Using SPARTAN-3E Embedded system

Sangook Moon, *Member, KIMICS*

***Abstract*— Recent semiconductor design technology has been substantially developed that we can design a micro-system on a chip as well as implementing an application specific IC in an FPGA. SPARTAN-3E developed by Xilinx is equipped with an FPGA that holds as much as 500 thousand transistors connected with MicroBlaze softcore microprocessor bus system. In this paper, we discuss a method of implementing an embedded system using the SPARTAN-3E. We also explain the peripherals and the bus protocols and the expandability of this kind of embedded systems.**

***Index Terms*— MicroBlaze, Embedded system, Platform based design, FPGA.**

I. INTRODUCTION

WITH the development of recent semiconductor technologies, we now can store tens or hundreds the times of capacity of 1990's hard disks into tiny flash memories which is far smaller than a thumbnail. Similarly, we now live an era in which we can build a microprocessor based system as easy as a LEGO toy with an application specific FPGA device with little cost in no time, while we used to build an ASIC (application specific integrated circuit) chip system at the cost of more than tens of thousand dollars. These types of FPGA systems are equipped with microprocessors and the bus systems as shapes of platforms, which require some kind of protocols with external peripherals to acquire both hardware and software verification in a very short time. We suggest a method which guides you through the process of using Xilinx Embedded Development Kit (EDK) software tools, in which this contribution will use the Xilinx Platform Studio (XPS) tool to create a simple processor system and the process of adding a custom OPB (On-chip Peripheral Bus) peripheral (an 32-bit adder circuit) to that processor system by using the functions which the tool provide. In this paper, we suggest a method to develop an embedded system easily in a short time, using the Xilinx SPARTAN-3E FPGA board.

Manuscript received July 20, 2011; revised July 29, 2011; accepted August 3, 2011.

Sangook Moon is with the Department of Electronic Engineering, Mokwon University, Daejeon, 302-729, Korea (Email: smoon @mokwon.ac.kr)

II. SOFTCORE MICROPROCESSOR

Microblaze is a virtual microprocessor core which is automatically synthesized using the special code blocks in the Xilinx FPGA device. One of the advantages of this kind of implementation is that we can duplicate the microprocessor core as many as we need [1].

Microblaze conforms to the 32bit Harvard RISC architecture and optimally designed for automatically synthesizable in the FPGA (Fig. 1). The separate 32bit instruction and data buses can get access to the external memory as well as the internal memory. The architecture has a 3 stage pipeline structure and there are 32 general registers, an ALU, a shifter, two-level interrupts, a barrel shifter, a divider, a multiplier, an FPU, and I/D cache to have an extended structure. This type of flexibility helps designers adjust the tradeoff between the area and the speed [2][3] (Fig. 2).

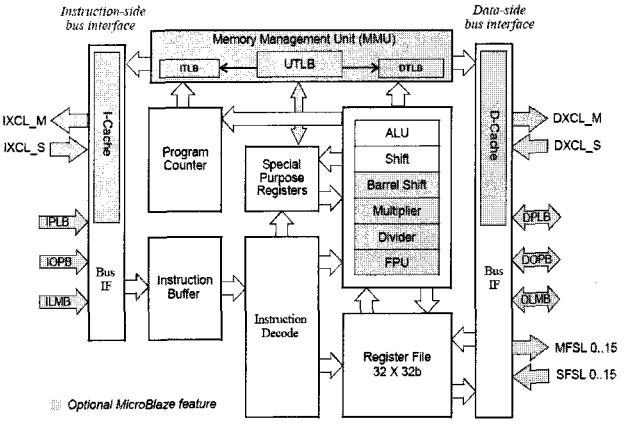

Fig. 1. MicroBlaze core block diagram.

III. OPB (ON-CHIP PERIPHERAL BUS) SYSTEM

In order to develop an application specific embedded system, building of a bus system is required. The Xilinx FPGA EDK (embedded system development kit) supports build either OPB or PLB (processor local bus) using the tool. In this paper, we chose the OPB because the OPB is easier and quicker to implement [4]. Fig. 3 shows the

OPB protocol between the peripheral bus and the softcore Microblaze processor. With this specified protocol, we could achieve fast communication between inside the system.

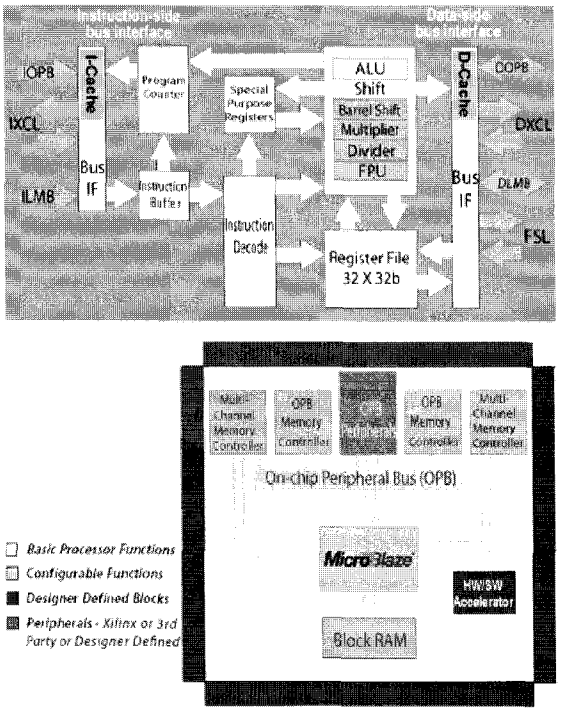

Fig. 2. MicroBlaze system block diagram.

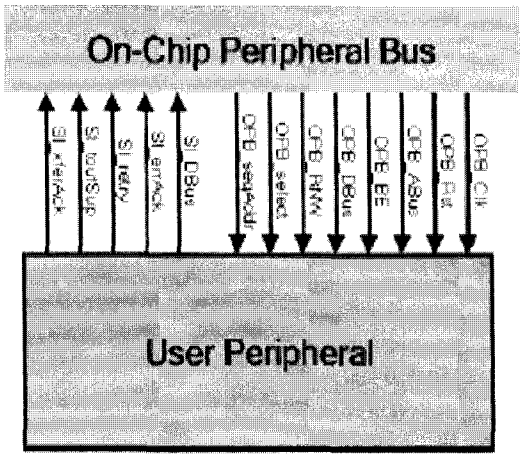

Fig. 3. The OPB bus protocol.

IV. TEST IP DESIGN

For the purpose of easy test, we created a custom 32-bit adder peripheral. We added the following code to user_logic.vhd file.

We firstly placed the code below to designate the signals for user logic slave model s/w accessible register example. This is declaring signals to be used in the custom 32-bit adder circuit.

```
signal sum          : std_logic_vector(0 to C_DWIDTH-1);
signal c            : std_logic_vector(0 to 32);
```

Secondly, we added the signal sum to the SLAVE_REG_READ_PROC and change the signal slv_reg2 to sum, as shown below. This will let the user to read the sum by reading the third register located at the base address of custom_logic_adder + 0x2.

```
SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0, slv_reg1, slv_reg2, sum ) is
 begin
  case slv_reg_read_select is
   when "100" => slv_ip2bus_data <= slv_reg0;
   when "010" => slv_ip2bus_data <= slv_reg1;
   when "001" => slv_ip2bus_data <= sum;
   when others => slv_ip2bus_data <= (others => '0');
  end case;
 end process SLAVE_REG_READ_PROC;
```

Finally, all the vhdl code below is needed to create the custom adder peripheral, a 32-bit ripple carry adder circuit that needs to be placed after the end process SLAVE_REG_READ_PROC. This is implemented in a process that will execute every time that the value of slv_reg0 or slv_reg1 or if both of them change. The 32-bit ripple carry adder circuit is implemented using a for loop instead of of a vhdl port map method because of the fact of not being able to get the vhdl port map method to work with the EDK.

```
adder_PROC : process(slv_reg0, slv_reg1) is

 begin
   c(32) <= '0';          -- Let c(32) = cin of the first LSB full adder
   -- Create 32 full adder using a for loop
   for i in 31 downto 0 loop

   sum(i) <= slv_reg0(i) xor slv_reg1(i) xor c(i+1);
   c(i) <= (slv_reg0(i) and slv_reg1(i)) or (slv_reg0(i) and c(i+1)) or (slv_reg1(i) and c(i+1));

   end loop;
end process adder_PROC;
```

V. IMPLEMENTATION AND CONCLUSION

We first create the MHS (microprocessor hardware specification) file and the MSS (microprocessor software specification) file in the EDK to define the hardware and the software specifications. After that, we set the operation clock speed and the size of the caches, the OPB bus system, and the other peripherals with the base system builder conforming to the OPB bus protocol. We inserted using the JTAG port interface to confirm the contents of the internal registers. As an application specific device, we used VHDL (very high speed hardware description language) to produce a 32-bit custom booth multiplier.

A rapid prototype of a 32-bit multiplier was described in VHDL and we loaded the HDL to the OPB bus system in the XPS. We put the protocol together and automatically synthesized the hardware image and transferred to the SPARTAN-3E board. Then we also transferred the software test program and confirmed the operation through the terminal via the UART. Fig. 4 shows the target board of SPARTAN-3E. We used the PC and a USB cable to transfer the hardware and the software image files, and we used the XMD debugger to verify the process of the result of the test programs. For the verification of the embedded system, we used the C program result and the hardware multiplier result to compare the functionality.

We suggested a method of rapidly implementing an embedded system using the Xilinx SPARTAN-3E development board. In this way, we can quickly produce an application specific embedded system with well made specification of software and hardware.

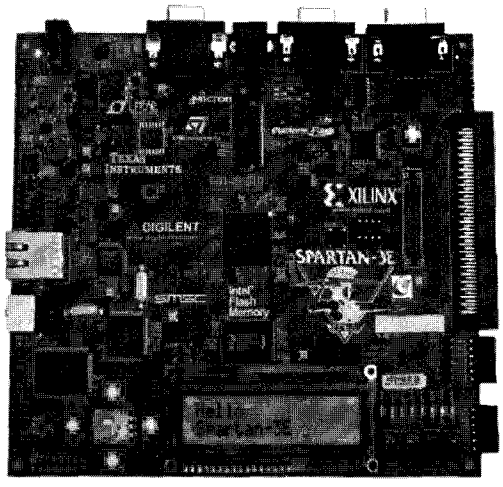

Fig. 4. SPARTAN-3E target board.

REFERENCES

[1] "Xilinx UG230 SPARTAN-3E Starter Kit Board User Guide", http://www.xilinx.com
[2] "Microblaze tutorial", http://www.xilinx.com
[3] "Microblaze microprocessor reference guide", http://www.xilinx.com
[4] http://ecasp.ece.iit.edu

Sangook Moon was born in Korea, in 1971. He received the B.S., M.S. and Ph.D. degree in electronic engineering from Yonsei University, Korea in 1995, 1997, and 2002 respectively. After the graduation, he had been working at Hynix Semiconductor as a senior engineer. In 2004, he joined the department of Electronic Engineering at Mokwon University, where he is currently an associate professor. His current research interests include VLSI, crypto-processors, computer arithmetic, SoC, and embedded systems.