Design of User Define Instruction Set using APU

Pratap Khuntia,
Lecturer
E & TC Department
C.V. Raman College of Engineering
Biju Patnaik University of Technology
Bhubaneswar-752054, odisha

Soumyashree Sethy,
M.Tech student
E & TC Department
C.V. Raman College of Engineering
Biju Patnaik University of Technology
Bhubaneswar-752054, odisha

ABSTRACT

This paper includes User Defined Instruction Decoding using the Auxiliary Processor Unit (APU) controller which allows the designer to extend the native PowerPC 405 instruction set with custom instructions that are executed by an FPGA Fabric Co-processor Module (FCM) which accelerate the system performance with the APU Controller, with an aim that Portions of certain software applications that are implemented in software can run faster by moving the implementation into hardware. In a Virtex™-4 FX FPGA, the embedded PowerPC™ 405 (PPC405) processor can run software and offload computations to hardware modules in the FPGA. In such a system, a coprocessor interface known as the Auxiliary Processor Unit (APU) is used to transfer data between the processor and the FPGA. Because certain computations can be done more efficiently in software, and others in hardware, an APU-enhanced system results in a faster overall solution for many digital signal processing (DSP) applications.

Index Terms –

User define instruction, APU, FCM, Co-processor, embedded PowerPC etc.

I. INTRODUCTION

Modern day FPGAs can take the current algorithms from various application fields like scientific calculations, image processing, digital signal processing etc, efficiently convert them into gate logic and run on the hardware . The performance can be improved with inherent parallelism provided by the FPGAs. A co-processor can be used to execute the compute intensive part of the program on hardware. This is achieved by off-loading the processor at a pre-decided location in our software program. The desired location in software program for off-loading the processor is known by the hardware-software partitioning process. In case of PowerPC 405 these co-processing functions are called by either using UDIs or with the pre-defined instructions.The hard core general purpose PowerPC 405 is compatible with the scalable and flexible instruction set and facilitates the implementation of UDIs configured by the user.

The software for the UDIs runs on the PPC 405, which offloads the computation to the hardware modules in the FPGAs. A co-Processor interface known as the auxiliary processor unit controller (APU) is used to transfer the data between the processor and the FCM. Instead of using the bus the APU integration between FCM and the processor pipeline. This technique significantly reduces the delay introduced by the bus signal-switching. The user-defined instructions are decoded by the APU controller as well as by the FCM also. The execution of UDIs takes place in the FCM only.

II. AUXILIARY PROCESSOR UNIT CONTROLLER

The APU controller serves two purposes: to perform clock domain synchronization between the fast PowerPC clock and the slow FCM interface clock, and to decode certain FCM instructions and notify the PowerPC of the PowerPC resources needed by the instruction (for example source data from the PowerPC's GPR). Depending on the FCM application, the APU controller can decode all instructions or no instructions at all, or decode some while the FCM decodes others.

All data is passed to the FCM from the processor through the APU controller and the APU controller knows in advance, when it will obtain the result from the FCM i.e. the time required by the FCM to execute the UDI. The block diagram for the pipeline flow for the PowerPC 405 core is shown in Figure 1. The pipeline for the PowerPC 405 consists of five stages as fetch, decode, execute writeback, and the

writethrough stages. In Figure 1 only three stages decode, execute, and the writeback are shown. The diagram consists of three hardware modules, PowerPC 405 core, APU controller and the FCM. Since the APU controller is available inside the PowerPC 405 core, hence it is also known as the hard coded controller.

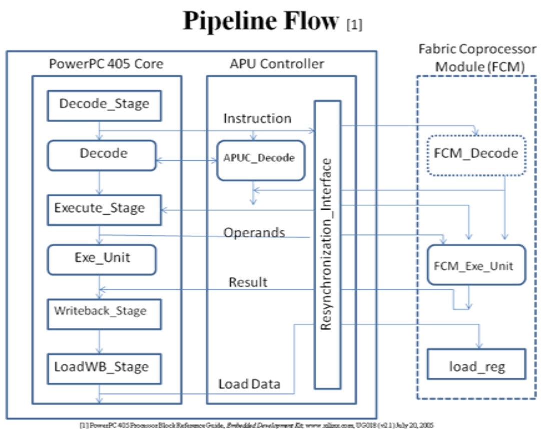


Figure 1. Pipeline flow for PowerPC 405 Core

III. WHY APU USEFUL?

It can be explained by making comparison between an APU implementation versus a bus peripheral implementation (shown below) in which there are same computations to offload to hardware. Typically, we would use a bus peripheral going through the PLB and taking a latency hit from bus arbitration and interface logic. With the APU we can remove the bus and most of the interface logic and go directly to the fabric. This saves logic resources and cycles and reduces the amount of software code

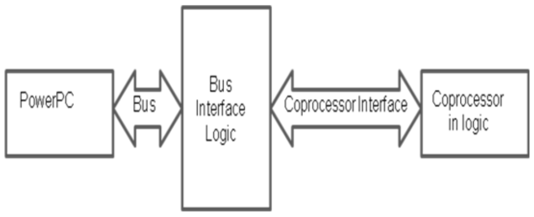


Figure 2: Convectional approach vertex-II pro FPGA platform

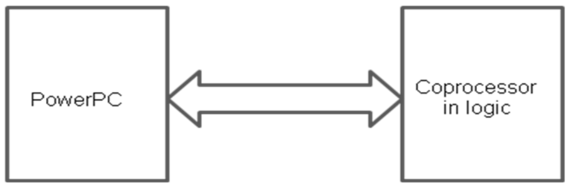


Figure 3: APU-centric approach vertex-4 FPGA platform

IV. FABRIC CO-PROCESSOR MODULE (FCM)

All the modules implemented in the FPGA logic, which interact with the APU controller represents the FCM. Most of the time the FCM runs at a lower clock frequency than that of the processor and APU. But, it never runs on the clock frequency greater than that of the processor and APU. The ratio between processor clock to the FCM clock can be in integer multiples e.g. 1:1, 2:1, 3:1,…, 16:1.During the operation, the FCM can also signal an exception to the processor

When the APU passes an instruction to the FCM, the FCM must decode and execute it [11]. This is because the APU decoding does not give the information about the operation to be performed by the instruction.

V. INSTRUCTION FORMAT

The general format for the user-defined as well as pre-defined instruction is shown below in Figure 4

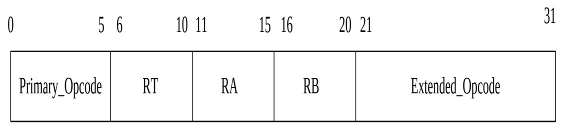


Figure 4: Instruction Format Showing the Various Instruction Fields [7].

The instruction format consists of five fields namely Primary_Opcode, RT (Target register, RA (Base Register), RB (Offset Register), and the Extended Opcode. RA and RB specifies the operand registers. As described earlier, the instruction can be decoded by the processor, APU controller, and the FCM. The PowerPC make use of both Primary_Opcode and Extended_Opcode to recognize the capability of the FCM instructions. In case of APU controller decoding as well as FCM decoding, to identify the specific FCM instructions the op codes are decoded [7].

In case of pre-defined instruction the length of all the five fields in the instruction format are fixed. But in case of UDI, the register fields i.e. RT, RA and RB can be configured to interpret these fields. The bit values in these fields can represent address as well as immediate values also.

The largest field in the instruction format is Extended_Opcode containing 10 bits. The bitwise description of this field is shown in Figure 5. To load the RA register with the effective address the bit-21 is used. The bit 22 is used to distinguish between the load/store operations in case of pre-defined instructions.. This information can be decoded with the help of bits 23-to-25 in the Extended_Opcode. The next three bits 24, 25, and 26 bits are used to know which operation the instruction will perform. The next two bits 27 and 28 are used to set the priviledge mode.

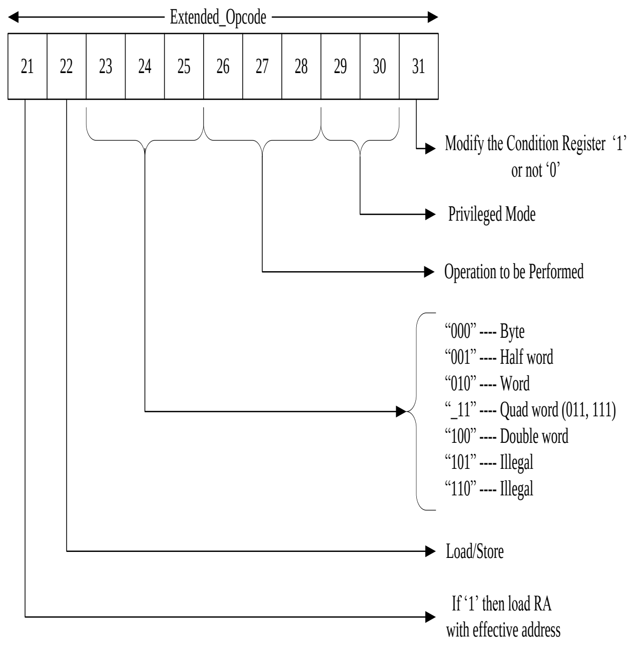


Figure 5: Bitwise Description for Extended_Opcode

VI. INTEGRATING HARDWARE AND SOFTWARE FLOW

After developing the hardware and software platform for our system, both are merged together to make a complete system

The compiled ELF and Bit files are loaded into the memory and then this combined image is loaded into the FPGA containing the PowerPC inside. The software is then executed on to the custom hardware inside the FPGA as shown above in Figure 6

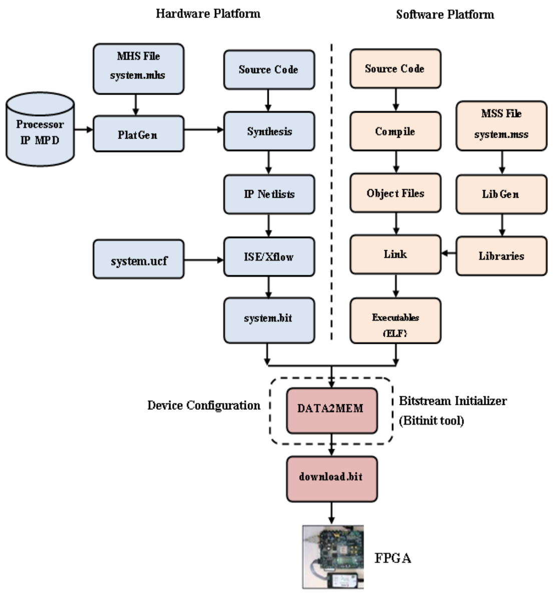


Figure 6. Embedded system development flow

VII. HARDWARE PLATFORM DEVELOPMENT FOR UDI

The hardware platform was developed using the Xilinx tool known as XPS. The wizard Base System Builder was used to develop the hardware system. The wizard automatically generates the required range of addresses and connects the ports of various peripherals to the processor through the buses. The hardware platform is also shown with the help of generated block diagram also. After making the complete hardware system it consists of the following IPs:

THE CUSTOM HARDWARE CREATION

Once our hardware platform is completed, next step is to create the custom hardware functionality in a hardware description language. For this purpose the VHDL was used. It should be remembered that the custom hardware to be defined in the HDL should be able to communicate with the PowerPC through the APU controller. For this reason, we create a state machine that controls the data communication between the custom functionality represented by the HDL module and the PowerPC through the APU controller. The FSM was also written in the HDL as a Moore machine.

The behavior of this state machine can be chosen according to the various instruction types. Various state machine were designed in different classes. At the same time, it should also be noticed that the instruction decoding is performed by the APU controller or the FCM.

This is because in different cases the state machine consists of different states and then different signals get important in a particular state. The custom functionality that represents the user- defined operation to be performed by the FCM was written in VHDL and instantiated in the FSM as a component. This module 'X' in Figure 6 represents the custom functionality. This module 'X' is used for computation in a particular state as defined by the instruction behavior in a particular class. These instruction classes are defined with timing relationships in detail in [7].

VIII. THE CUSTOM IP CONNECTION

When the IP is created in the VHDL, then it should be included in the system. We can include our IP in the system in two ways

A. USING THE DIRECT INTERFACE

In this approach no bus was used to connect the custom IP named as testipand to the APU controller's ports. The 32-bit and function was used as the 'X' module in the FCM, and instantiated as component in the Moore FSM.

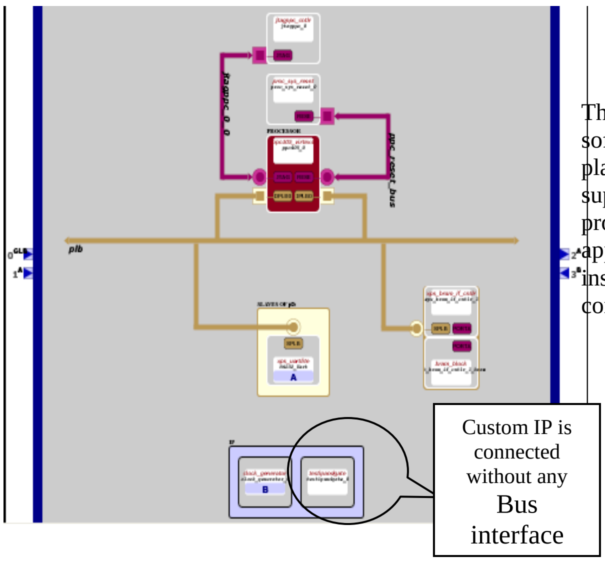


Figure 7: Block Diagram for Custom IP connection using the Direct Interface

B. USING THE FCB INTERFACE

This interface provides the alternative for the designer to the direct interface. This is the dedicated bus that connects the custom IP directly to the PowerPC through the APU controller. This option is provided in the XPS under the IP catalog. Thus, there is no need to create the interface from scratch. This technique was also successfully used to connect the custom IP with the PowerPC through the APU controller. The block diagram generated by the XPS is shown below in Figure .8.

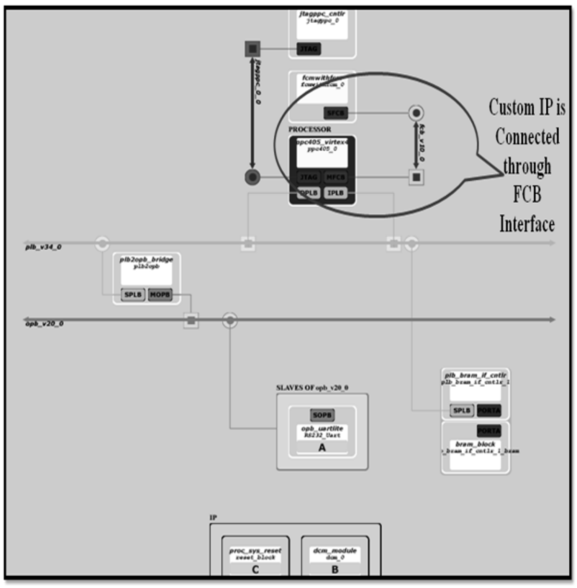


Figure 8: Block Diagram showing the Custom IP Connection using the FCB Interface

IX. SOFTWARE PLATFORM DEVELOPMENT FOR UDI

The second important task in UDI formation is to develop a software platform for the UDI. For developing the software platform we use C as the software language. This is because it is supported by the Xilinx EDK tools. We write an application program for our already developed hardware platform. This application program was needed to transfer the control of instruction execution from PowerPC 405 pipeline to the APU controller.

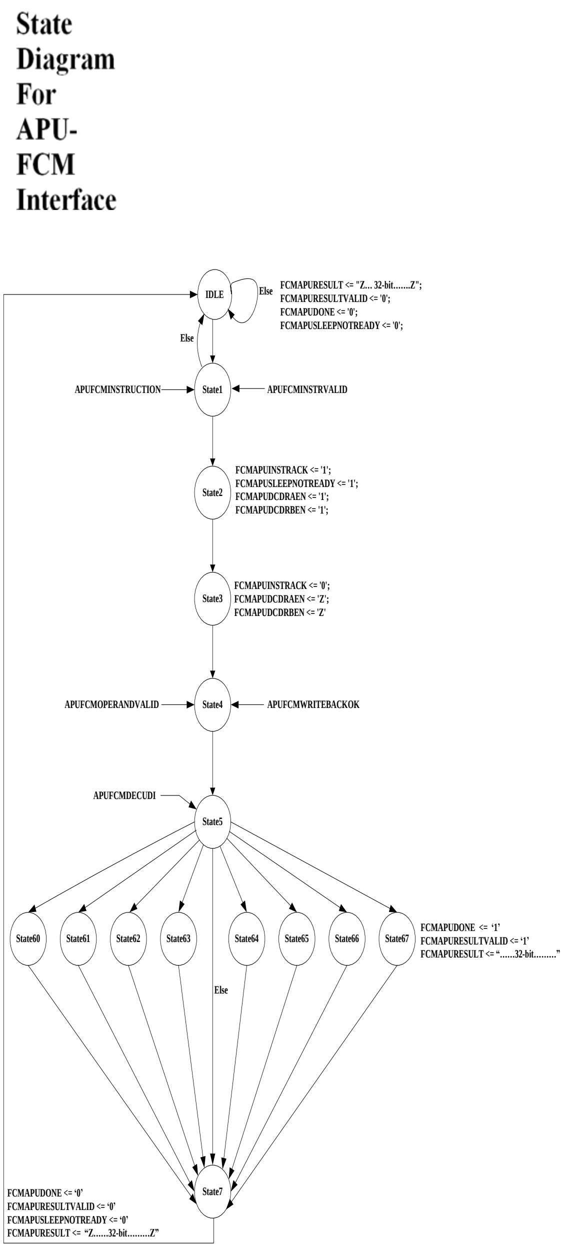


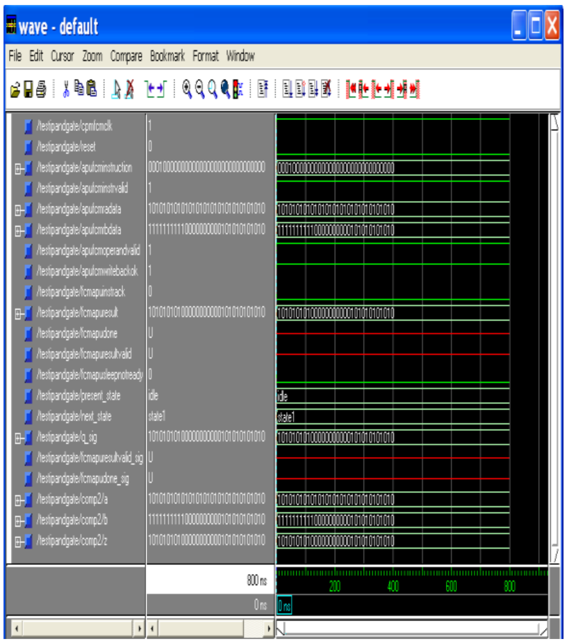
Figure 9.State Diagram for APU-FCM Interface

X. RESULTS AND DISCUSSION

In this chapter the simulation and synthesis results for the UDIs with different classes are shown. For simulation purpose the ModelSim 6.0d and for synthesis the Xilinx's ISE

9.1i tools were used. The simulation results are based on the Moore state machines.

The simulation results for "and_gate" used as the 'X' module is shown below in case of blocking transaction. After receiving the APUFCMWRITEBACKOK signal the FCM perform the "and" operation on the available bits and gives the result back to the PowerPC through the APU controller. The signals FCMAPUDONE etc are also activated in the same cycle by the FCM.



```
/testipandgate/cpmfcmclk              1
/testipandgate/reset                  0
/testipandgate/apufcminstruction      00010000000000000000000000000000
/testipandgate/apufcminstrvalid       1
/testipandgate/apufcmradata           10101010101010101010101010101010
/testipandgate/apufcmrbdata           11111111110000000000101010101010
/testipandgate/apufcmoperandvalid     1
/testipandgate/apufcmwritebackok      1
/testipandgate/fcmapuinstrack         0
/testipandgate/fcmapuresult           10101010100000000000101010101010
/testipandgate/fcmapudone             U
/testipandgate/fcmapuresultvalid      U
/testipandgate/fcmapusleepnotready    0
/testipandgate/present_state          idle
/testipandgate/next_state             state1
/testipandgate/q_sig                  10101010100000000000101010101010
/testipandgate/fcmapuresultvalid_sig  U
/testipandgate/fcmapudone_sig         U
/testipandgate/comp2/a                10101010101010101010101010101010
/testipandgate/comp2/b                11111111110000000000101010101010
/testipandgate/comp2/z                10101010100000000000101010101010
                                800 ns
                                  0 ns
```

Figure 10: FCM Decoding for UDI with Blocking Transaction using "and_gate" as the 'X' Module

The synthesis results for Figure above are listed in Table below. The optimization goal was speed and the delay produced in the synthesis that is 1.393ns. Total memory used for the design is 233716KB. The target device used for the design was xc4vfx20 and the pin package was selected as ff672 with a speed grade of 10 for every case. The direct interface between the APU controller and the FCM was used.

Table 1: Synthesis Results for APU Decoding for UDI with Non-Autonomous Transaction

Logic Utilization	**Used**	**Available**	**Utilization**
No. of Slices	25	8544	0%
No. of 4 – Input LUTs	44	17088	0%
No. of IOBs	18	320	5%

XI. CONCLUSIONS

The proposed work entitled "Design of User-Defined Instruction Set" with EDK for FPGA Based Embedded System has been completed within the specified time. The Hardware-Software co-design approach was used to develop the complete system for the target device xc4vfx20 and pin package ff672 with a speed grade of 10 in Virtex-4 FPGA board. The PowerPC 405 was chosen as the target processor. The HDL code for the internal FCM logic was developed in VHDL. The ModelSim 6.0d tool was used to simulate the developed HDL codes and the synthesis was done with Xilinx's ISE 9.1i tool. The software language C was used to write application program for the complete system.

REFERNCES

[1] Bryce Mackin and Nathan Woods, "FPGA Acceleration in High Power Computing: A Case Study in Financial Analytics", XtremeData, Inc., Version 1.0, November 2006.

[2] Harn Hua Ng and Dan Issacs, "Closely Coupled Co-processors for Algorithmic Acceleration", FPGA and Programmable Logic Journal, Techfocus Media, Inc., 2004.

[3] "IBM PowerPC 405 Embedded Core", @ http://www.ibm.com, 2006.

[4] "Accelerating PowerPC Software Applications", Embedded Magazine, Issue 2, September 2005.

[5] Seyed H. Roosta, "Parallel Processing and Parallel Algorithms", Springer-Verlag, New York, Inc, 2000.

[6] "PowerPC 405 Processor Block Reference Guide", @ http://www.xilinx.com.

[7] "Power PC Instruction Set Extension Guide", @ http://www.xilinx.com.

[8] David Hultan, "High Speed Computing & Co-processing with FPFAs", Dachboden Labs & Pico Computing, Inc, December 2004.

[9] A. Ansari, P. Ryser, and D. Isaacs, "Accelerating System Performance with APU-Enhanced Processing", Xcell Journal, First Quarter2005.

[10] Greg Lara, "Vertex-4: Breakthrough Performance at the Lowest Cost", Xilinx, Inc., January 2005.

[11] "Accelerated System Performance with the APU Controller and Xtreme DSP Slices", xapp717 (v1.1.1) Sept. 29, 2005, @ www.xilinx.com.