Design of a SIMD Multimedia SoC Platform

Guang-Huei Lin 1, Ya-Nan Wen 1, Xiao-Long Wu 2, Sao-Jie Chen 1,2, and Yu-Hen Hu 3

1Graduate Institute of Electrical Engineering, National Taiwan University
Taipei, Taiwan, ROC
2Graduate Institute of Electronics Engineering, National Taiwan University
Taipei, Taiwan, ROC
3Department of Electrical and Computer Engineering, University of Wisconsin, Madison
Madison, WI53706, USA

ABSTRACT

PLX is a native SIMD instruction set architecture, which integrates scalar instructions into a multimedia extension SIMD core to run OS and multimedia functions. It changes the typical CPU-DSP dual core architecture to a SIMD-I/OCTRL architecture. We have developed a PLX-based system developing platform, including a PLX core, a RISC-like 8051 as I/O controller, an H.264 accelerator hardware, a SystemC system level virtual prototype for verification and OS development at early stage, and a SIMD code generation methodology for software to better exploit the 64-bit SIMD computation power.

I. INTRODUCTION

The idea of SIMD (single instruction multiple data) instruction set architecture (ISA) is to parallel process low-precision data, such as 16-bit audio samples and 8-bit pixel components. On video processing, similar operation is often applied on all pixels, if the eight 8-bit pixels can be parallel processed in a 64-bit processor, the performance can be enhanced eight times. MAX-1 of HP PA-RISC [1] is the first SIMD ISA, other famous ISA examples are MMX/SSE in Intel IA-32/64, VMX/AltiVec in IBM PowerPC, 3DNow! in AMD, VIS in SUN SPARC, and MDMX in MIPS processors. All of them are implemented as co-processor.

PLX [2] is a native SIMD ISA, developed in Princeton University. Its SIMD function unit supports 8/16/32/64 subword width, and its scalar function unit is the 32-bit subword subset in a SIMD unit. All PLX softwares are SIMD. Usually, processor with a CPU and a DSP core has been used for portable productions to reduce cost. Being a native SIMD design, PLX can replace DSP. With its 64-bit processing capability, two 32-bit scalar operations can be executed simultaneously, which compensates the high computation time of many DSP functions.

Real-time is a critical concern on multimedia system. While integrating DSP functions into PLX, CPU becomes more busy and may delay real-time response. To avoid real-time violation, we move some work to I/O controller, which is implemented with an 8-bit RISC-like 8051 MCU, responsible for disk management and input image filter. For examples, when an input image is obtained from CMOS sensor, it should be filtered by AWB (auto white balance) and AE (auto exposure) to balance its color. And if an input image is captured from the VCD analog signal, de-interlace filter should be applied to smooth the image. These works are suitable to be performed by an I/O controller.

II. PLX PROCESSOR DESIGN

The PLX core is designed as a low power processor. As we know, the basic dynamic power consumption equation is shown as:

\[ \text{Power}_{\text{max}} = \alpha \times C \times V_{cc}^2 \times f_{\text{max}} \]  (1)

where \( \alpha \) is an activity factor, \( C \) is load capacitance, \( V_{cc} \) is supply power, and \( f_{\text{max}} \) is clock frequency. The maximum clock frequency is approximately proportional to \( V_{cc} \) in operation range [3] as:

\[ f_{\text{max}} = K \times V_{cc} \]  (2)

By the above equations, \( \text{Power}_{\text{max}} \) is proportional to \( V_{cc}^3 \). That is, while \( V_{cc} \) is reduced by 1%, \( f_{\text{max}} \) will be reduced by 1%, but \( \text{Power}_{\text{max}} \) reduced by 3%. In other words, using lower \( V_{cc} \) to save power, we have to design a CPU with \( f_{\text{max}} \) far higher than the normal specification required.
Multiplier always forms the critical path in an ALU. Since in multimedia applications, 16-bit multiplication is used more often, PLX thus supports only 16-bit multiplication to reduce the critical path delay. Though we need more instructions to perform a 32-bit multiplication but since it is seldom used, the trade-off is worthy.

To reuse low precision subword addition for a high precision subword sum, carry-select adder as shown in Fig 1 is adopted, where eight 8-bit units work in parallel. When a higher resolution sum is required, the carry select path is opened to perform a 16-, 32-, or 64-bit addition operation. The delay of a PLX 64-bit addition is the delay of an 8-bit adder plus those of 7 multiplexers, which is far faster than a normal 16-bit multiplication.

By the low critical path delay, \( f_{\text{max}} \) will be far higher than we need. Thus we have the room to reduce both \( f_{\text{max}} \) and Vcc to get more power gain.

Figure 2 shows the PLX core block diagram. It is a 5-stage pipelining RISC architecture. The execution stage contains a Load/Store unit and a subword-parallel ALU. The RegFile contains 32 64-bit registers. The ALU result will be forwarded to OPFETCH when a read-after-write dependency occurs. Load result may be delayed to store back into the RegFile when ALU is writing, this reduces the requirement of a RegFile write port. DCACHE is implemented as Write Through that allows H.264 module to get consistent data.

### III. RISC-LIKE 8051 DESIGN

On our architecture, a high performance I/O controller is required to share some real-time work from PLX. We implement a RISC-like 8051 for this purpose. 8051 is an 8-bit CISC microcontroller developed by Intel in 1970s. Most CISC architecture contains a long critical path between ALU and memory, which restricts clock frequency enhancement. The idea of utilizing RISC technology for CISC processor is brought from the Intel P6 processor project [4].

The RISC philosophy can be simplified as: (1) build hardware in which almost all of the operations are simple and fundamental operations, (2) operate on simple data kept in registers plus load and store operations.

The RISC conversion design is shown in Fig 3. Our designed 8051 is a 3-way superscalar architecture, which instruction length is 1 to 3 bytes. The codes are fetched from the 32-bit embedded ROM, and stored in IBUFF. The ALIGN stage extracts 3 instructions from IBUFF with the aid of Length Decode Chain. Then, every instruction is translated into 1 to 3 \( \mu \)Ops which follows RISC philosophy. The following stages of OPFETCH, EXECUTION and STORE are the same as the RISC ones.

To reduce pipeline invalidation penalty, a SKIP mechanism is designed to emulate the conditional execution that most RISC cores use. When a jump occurs and the destination is short in forward direction, this jump is recognized as an if-else block, and a SKIP flag is set instead of a jump assertion. During SKIP set, all \( \mu \)Ops in the execution unit are replaced by NOPs. When the program counter contains a target jump destination, the SKIP flag is cleared and the \( \mu \)OP executed normally. This mechanism reduces a branch pipeline invalidation penalty from 7 to 4 cycles.

![Fig 1. 64-bit subword-parallel carry-select ALU.](image1.png)

![Fig 2. PLX core block diagram.](image2.png)

![Fig 3. RISC-like 8051 decode stage.](image3.png)
IV. SIMD CODE GENERATION

The challenge of running applications on the PLX platform is to exploit its SIMD capability. Most multimedia functions cannot be optimized by just using a vectorizing compiler [5]. The following example shows how to parallel get two 4x4 SAD (summation of absolute difference) results for motion estimation using PLX [6]: load a row (8 pixels) from current frame and reference frame into two 64-bit registers, calculate their difference and absolute in parallel, sum with next three rows, finally sum the low 4-bytes and high 4-bytes sequentially. This flow is shown in Code 1. Without SIMD, it takes 4x8x4+8x4+4=168 instructions. With SIMD, the loop of index col is removed by SIMD parallel execution, the number of instructions becomes 4x4+4+4+4=28, which owns a 6-times speedup.

Typical SAD processing flow calculates two 4x4 SADs sequentially, vectorizing innermost loop only cannot deliver good speedup.

To improve software performance, we developed an 8-step code generation methodology based on abstraction and reconstruction techniques to exploit the SIMD capability when transforming the source codes to PLX codes.

Step 1, divide complex equation into one-operation codes. A complex equation which has many operations is usually partially parallelizable. The if-compare contains a compare operation and an if-judgment operation. The compare operation should be independent to save to an additional Boolean local variable.

Step 2, unroll all loops.

Step 3, build the data dependency graph.

Step 4, expand local variable to remove write-after write and write-after-read data dependencies. This is an important step to make more code parallelizable.

Step 5, reschedule the instructions. All instructions are rescheduled to the earliest time it can execute by the data dependency graph. If no dependency between two operations, they will list in the same time slot and able to execute parallel.

Step 6, abstraction. If an operation is applied on all the elements of a matrix, we can found all the matrix indices listed on the same rescheduled time slot. For example, in the reschedule of Code1, time slot 1 contains the difference operations in all row and col, and time slot 2 contains the abs operations in all row and col. Then we know it is an absolute of difference operation applied on the whole matrix. Summation can be abstracted in sequential time slot.

Step 7, reconstruction. By the abstracted function, we can apply pre-defined rules to reconstruct them. The rules depend on the function belonged to video processing (macro-block based), image processing (2-D matrix operation) or communication (1-D cyclic buffer).

Step 8, adjust memory alignment. Streaming shift is inserted to avoid miss-alignment memory access.

For OS and non-matrix operations, since we cannot abstract their function, the reconstruction step becomes useless. But they mostly use 32-bit integers, and PLX is a 64-bit design, we can combine two operations into one SIMD instruction to improve performance. The optimization is performed by a heuristic [7] to check all possible combinations in every code block and unroll loops to have a larger solution space.

V. SYSTEMC VIRTUAL PROTOTYPE DESIGN

SystemC is a class library under C++ compiler. The class library can emulate hardware event trigger and parallel behavior, which can be traced in a C++ debug environment. This helps designer to co-design hardware and software at system level. System level design is especially important to verify new architecture.

At system level, we need only to take care of chip interface and bus transaction and ignore the logic block complexity. The PLX core is implemented as a cycle-accurate instruction set simulator (ISS). This ISS can execute instruction code, handle interrupt and memory access at the same timing as in hardware. H.264 modules are implemented on transaction level, the hardware behavior is implemented by optimized C code to reduce simulation loading, but keeps the bus
transaction timing accurate. Real-time OS can then be developed and debugged on this cycle-accurate environment easily.

Using SystemC, we can build a virtual prototype in a PC. Simulation based design verification needs to build many simulation patterns and feed them into the ISS simulator. But patterns are always not enough for a complex system, especially for real-time applications. We thus have to verify the design by FPGA prototype. But FPGA prototype can work only after the detailed logic design is done, which is too late for system level verification. Since SystemC can easily cooperate with other C++ threads in the same environment, we can use these threads and PC resources to emulate peripherals [8]. The transaction to LCD interface can be captured and displayed on a window, and the TCP/IP packet can be passed to other computer through network card on PC. This PC-based system-level virtual prototype can be used to verify application under real-time condition before any hardware designed.

VI. SYSTEM INTEGRATION

Figure 4 shows our system block diagram. The PLX, H.264 and 8051 chips are independently developed based on SystemC system level specification. These chips are verified on a FPGA reconfigurable platform before integrated into an SoC. The kernel in FPGA is a PCI/AHB switch. Since the Ethernet, LCD and SDRAM on the FPGA development board all work on FPGA local bus, the PCI/AHB switch is used to handle bus protocol conversion. The bus switch is configured to allow two bus accesses at the same time. For example, when H.264 hardware is accessing main memory, the input image from 8051 or packet from Ethernet can be stored on SRAM3/DRAM4. The H264 chip implements 1-D motion estimation and plays as an AHB master. It is started when the CtrlReg is set by PLX. Then it asserts memory request and loads current frame and reference frame pixels from main memory to its local memories SRAM1 and SRAM2. The pixels are loaded one macroblock at a time. During motion estimation processing the macroblock on local memory, the bus is not occupied such that PLX and 8051 can use it.

The designed 8051 chip contains a USB2.0 transceiver, a compact flash interface and some serial I/O interfaces. The USB is configured as a bulk-only device, used to upload pictures saved on compact flash disk when PLX system plays as digital camera. An 8-bit local bus is used to communicate with FPGA. It is translated into AHB by the 8-bit bridge on FPGA.

VII. CONCLUSION

We have built a PLX-core based multimedia SoC development platform, from system level virtual prototype, SIMD code generation to chip design. The PLX platform can be used in small portable devices by its tiny core and high-performance in processing multimedia functions.

REFERENCES