Bridging the Gap between ASIC and GPP: A High-Performance and C-Programmable ASIP for Image Processing

Mochamad Asri, Hsuan-Chun Liao, Tsuyoshi Isshiki, Dongju Li and Hiroaki Kunieda
Department of Communications and Integrated Systems, Tokyo Institute of Technology, Japan.

Correspondence:
Mochamad Asri
Department of Communications and Integrated Systems, Tokyo Institute of Technology, Japan.
Email: asri@vlsi.ss.titech.ac.jp

Abstract

Emerging digital signal applications nowadays require not only massive computational load but also a significant level of flexibility and programmability. ASIC (Application Specific Integrated Circuit) and GPP (General Purpose Processor) have been widely used as the two main countermeasures to fulfill those requirements. Unfortunately, the gap between these two mainstreams is deliberately big. While ASIC can fulfill the requirement of ultra-high performance, it lacks the flexibility and programmability. On the other hand, the performance of GPP is not competitive despite it offering an excellent level of flexibility and programmability. In this paper, we present ASIP (Application Specific Instruction-set Processor) for image processing as a solution in bridging the gap between ASIC and GPP. The designed ASIP constitutes the appropriate solution for fulfilling both the flexibility and performance constraints of emerging and future image processing applications. Based on the simulation result, we found that our ASIP can achieve 16 times better performance of conventional RISC (Reduced Instruction Set Computing) Processor as well as providing a high level of flexibility and programmability that ASIC lacks.

Keywords: microprocessor, ASIP, embedded system, reconfigurable system, high-level synthesis

1. Introduction

Massive computational workload with high complexity is the state-of-the-art digital signal processing field. Processing requirements are rising sharply and often need to be completed in a very strict timing constraint. Baseband signal processing in 3G cellular base, for instance, requires hundreds of billion operations per second with very limited power budget, an efficiency of about 100 GOPS/W [1].

At the same time, not only high performance efficiency, but programmability is also needed to follow evolving standards. The severe time-to-market constraints and continuously developing new standards and applications in digital applications, make resorting to new design methodologies and the specification changes inevitable. In 3G cellular-base baseband signal processing, for example, flexibility and programmability are needed to support multiple air interfaces, as well as to supply processing resources over different air interfaces.

In fact, conventional signal processing solutions are able to provide high efficiency or programmability. However, unfortunately they lie in extremely different poles. In applications demanding high performance efficiency, a hardwired ASIC (Application Specific Integrated Circuit) has an extremely high amount of efficiency, but it offers little if there are any specification changes. At the other pole, microprocessors are completely programmable but have a less performance efficiency.

In this case, engineers must choose between performance efficiency and programmability. When power budget is considered as the main constraint, efficiency is the choice. ASIC, therefore, would be the main priority, abandoning programmability. This means that only a single design interface or specification can be supported, and a separate ASIC is needed for each design interface and specification.

On the other hand, processor trends seem to bring a new chance for yielding higher performance efficiency. Processor speeds have historically increased over time driven by higher transistor densities. Furthermore, advance techniques such as superpipelining, out-of-order superscalar execution, dynamic scheduling, multilevel memory caching and aggressive speculation have been proposed to tackle the problem. However, with densities approaching atomic levels, these incremental improvements now yield diminishing returns, with instruction-level parallelism reaching its limit, as efficiency improves slower than density [2].

An application Specific Instruction-set Processor (ASIP) emerges as a solution to bridge those two extremes [3]. The concept of ASIP is to get the best out of microprocessor programmability while at the same time trying to offer performance efficiency as high as ASIC's. The key approach is to optimize the microarchitecture design domain by aggressively inserting special instructions (Instruction Set Architecture extensions).

In this paper, we propose a high performance and highly programmable ASIP core for image processing applications that brings together competitive performance efficiency as well as programmability.

This paper is organized in the following manner. First, we present the current image processing demand and its algorithm used in this work. Secondly, we analyze the hotspots of the processing's computational load. Afterwards, we outline our proposed ASIP core based on our novel design methodology. Next, we provide the performance evaluation of our design. Lastly, we present our conclusion regarding the current and future works.

2. To ASIP or Not To ASIP

Over the past decades, ASIC (Application Specific Integrated Circuit) and GPP (General Purpose Processor) have been widely used as the two main countermeasures on Digital Signal Processing field. Unfortunately, the gap between these two mainstreams is considerably big. While ASIC can fulfill the requirement of ultra-high performance, it lacks the flexibility and programmability. On the other hand, the performance of GPP is not competitive despite it offers excellent level of flexibility and programmability. A big architectural spectrum between ASIC and GPP is there. And the question remains the same: What is the best solution to fill this spectrum?

Several studies have been conducted on this hot issue. Engineers are continuously investigating what the most probable and realistic alternative is to realizea a highly efficient, low-power, flexible and programmable engine for computing system. Thanks to their deep and comprehensive analysis, on the latest work regarding this issue both Hameed [4] and Shacham [5] eventually figured out that the smartest and ideal way to cope with this abstruse problem is to design a customized processor for the target application. They came up with a conclusion that the most effective way to be efficient is to find a solution that accomplishes the same task with less work. By tailoring hardware to a specific application,

customization can improve performance efficiencies while at the same time can save energy by performing less work and also maintaining a high level of programmability. The main message is there: The emergence of ASIPs is ineluctable.

In this work, we propose a high performance and C-programmable ASIP core for image processing applications. We developed a novel ASIP architecture design based on advanced methodology and integrated tools that sufficiently enhance ease-of-development with image processing being our main application target to be accomplished.

3. Image Processing Applications: The Why and The What

In this chapter, the motivational background of targeting image processing application is described first. Then the main two image post-processing stages used in this work will be described.

3.1 Motivational Background

New video formats have been proposed widely as a solution for future video technology. Super Hi-Vision (SHV), for example, has been proposed by Japanese Broadcasting Company (NHK) and has been accepted as a new international standard by the International Standard Union [6]. The resolution of SHV is 16 times Full High Definition format (1980x1080) while the frame rate is 2 times. Because of this new video technology format, the image resolution has increased substantially, which in turn has increased both the image processing computational complexity and flexibility, similar as the previously mentioned 3G baseband technology. Consequently, in order to process such images, much more powerful, while at the same time, flexible processing engines are required. This is the main motivation of our work in choosing image processing as our target application.

3.2 Several Main Types of Image Processing

In general, there are 2 main types of the conventional image processing stages used in this work. We will describe them below.

3.2.1 Image Scaling

This functional block is responsible for scaling of input image to fit the resolution of a display. This block enlarges or shrinks the input image by using Lanczos3 [7] interpolation or pixel averaging. Usually the resolution of display device is different from the resolution of input sources. Therefore, scaling is used to enlarge or shrink the resolution of input image to fit the resolution of a display device. The up-scaling is performed by convoluting several pixels with Lanczos3-windowed function. During the up-scaling process, Lanczos co-efficient and pixels must be loaded from a look up table located in the data memory. Due to the huge amount of load operations during processing of each pixel the total processing time is significantly large.

3.2.2 Image Enhancement

This functional block is responsible for enhancing the quality of the image. In this block a Non-Linear Enhancer (NLE) [7] first uses a high-pass filter on the input image which will be followed by cube operation and then clipping. The processed results will add luminance to the up-scaled image to complete the processing. Linear interpolation techniques (bilinear, Lanczos 3 etc.) for image enhancement cannot reconstruct high frequency components of original images, which are in turn lost. Therefore, NLE is used to restore high frequency components of images, which is based on extending the frequency component of the input image to improve the quality of images.

4. Hotspots of the Processing Algorithm

In order to achieve significant level of performance improvement, the identification of hotspots in program execution is needed. We will explain in detail how we do this.

We first start by creating a sequential application written in C and perform performance analysis of the application. In order to analyze computational intensive areas in the program, we use our previously proposed compiler tool, the TCT Compiler [8]. This compiler tool was developed along with its cycle accurate simulator to measure the cycle count of the specific part of the C code, so that it can provide an performance analysis of the application as well as computational intensive parts in the program. Fig. 1 illustrates our compiler tool, the when performing as a cycle accurate simulator.

In our current implementation the cycle count is calculated based on our custom built 4-stage pipelined, 32-bit RISC Harvard architecture, the TCT Processor [9]. By analysing the program, we found that the double-loop processing part is the hotspot of the program.

After finding hotspots in the application we perform performance estimation with an assumption that the hotspot processing can be completed in "n" cycles. Afterwards, we simulate the program again to obtain the performance improvement estimation that can be achieved if we optimize the microarchitecture by adding special instructions based on the previously stated assumption.

5. Proposed Method

Based on the hotspots analyzed in the previous section, we move to the next step, which is to design ASIP that can process the computationally intensive part in a short time.

To design an ASIP, in general there are 3 main methodologies:

- Design by Hardware Description Language (HDL).
- Design by using template processor and specialized modules.
- Design by Architecture Description Language (ADL).

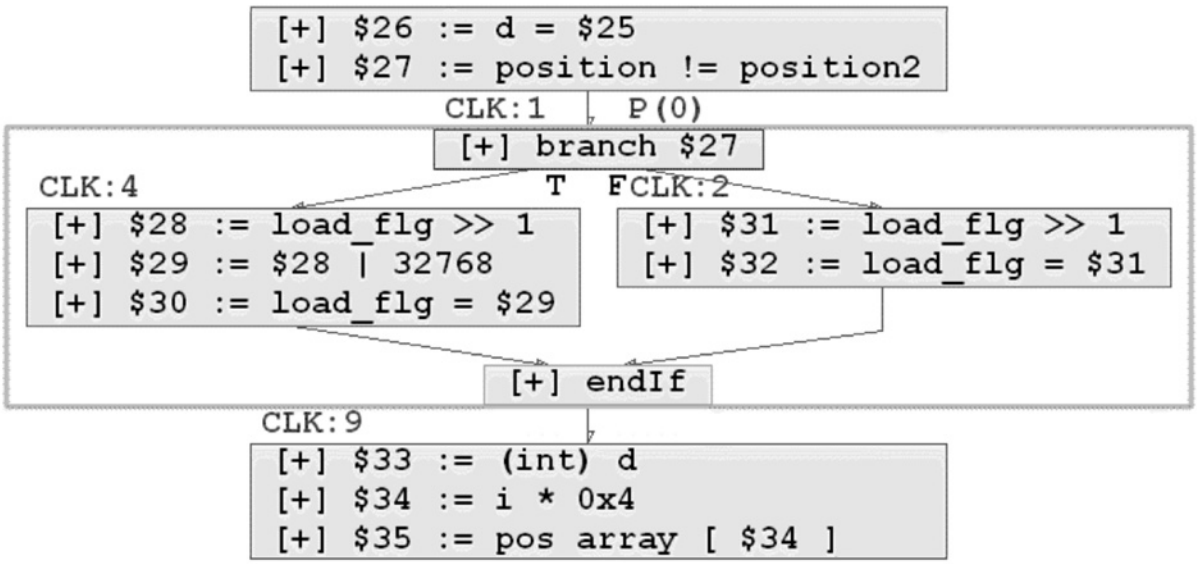

Fig. 1. TCT Compiler Cycle Accurate Simulator.

HDL-based ASIP design may result in an area efficient circuit, but it needs huge design and verification efforts. In template based design, ASIP is designed by combining a template processor with specialized module selected from a library. This method is suitable to solve design complexity and realize short turnaround time. A good example of this approach is MeP [10]. However, one of the drawbacks of the template base design is the design flexibility, because of the fixed base architecture of the template processor and modules available in the library. In ADL-based design, processor architecture and behaviour are described at higher abstraction from which RTL can be automatically generated. Describing processor architecture and behaviour at higher abstraction need less design efforts and makes the process of verification simple. LISA-Language for Instruction Set Architecture [11] and ASIP Meister [12] are good examples of this approach. In this paper we choose ADL-based ASIP design methodology by using the LISA language. We will explain the methodology in details

Fig. 2 shows the general flow of our ASIP design methodology. After figuring out the hotspot by using the TCT Cycle Accurate Simulator, we design a datapath so that it meets the target requirement. Afterwards, we implement the basic architecture of the ASIP using the Instruction Accurate (IA) model on Synopsys Processor Designer [13]. Note that pipeline stages are not considered at this time. This stage aims to verify the implementation and confirm the feasibility of the architecture. Next, after the IA verification is completed, we implement the Cycle Accurate (CA) model. In the CA model each instruction is spread over several pipeline stages. A CA model needs careful scheduling of instructions in each pipeline stage in order to get maximum performance gain. When the verification of the CA model has been performed, we can automatically generate an HDL description of the processor by using Synopsys Processor Designer. This shows the general flow of our design methodology.

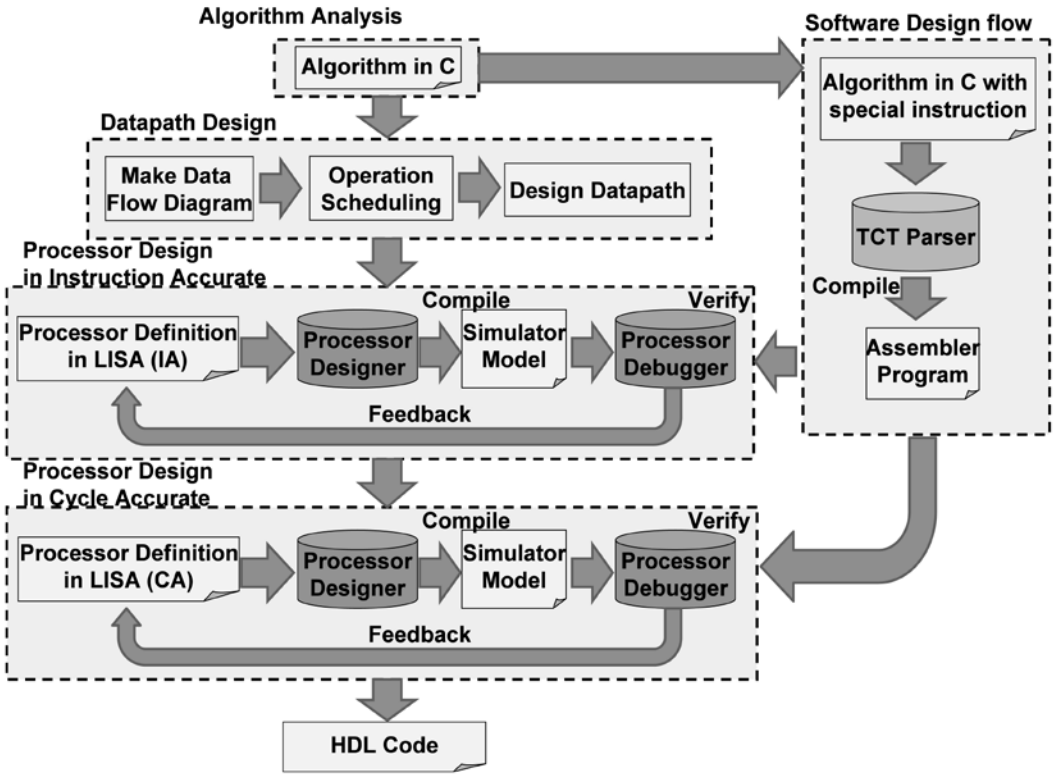

Fig. 2. Proposed ASIP Design Flow.

5.1 ASIP Architecture

As stated in the previous section, an ASIP should fulfill 3 categories in order to accommodate several requirements of target application which are High-Performance, Flexible, and Easily Programmable. Considering these demands, we design an ASIP core with 3 characteristics: High Data-Parallelism, Reconfigurable, and C-Programmable. We will explain in detail below.

5.1.1 High Data-Parallelism Datapath

After hotspot identification in the code we design a datapath using LISA. This data path is added as a special instruction to a simple RISC with some basic load store instructions. Each of the operations Multiply, Add, Shift, and Clip are completed in a single cycle as shown in data flow diagram in Fig 3. As for the filtering operation, we apply 6-tap filtering as the mainly used process in this work. As we can see from Fig. 3, the complete 6-tap filtering operation is scheduled to complete in 6 cycles.

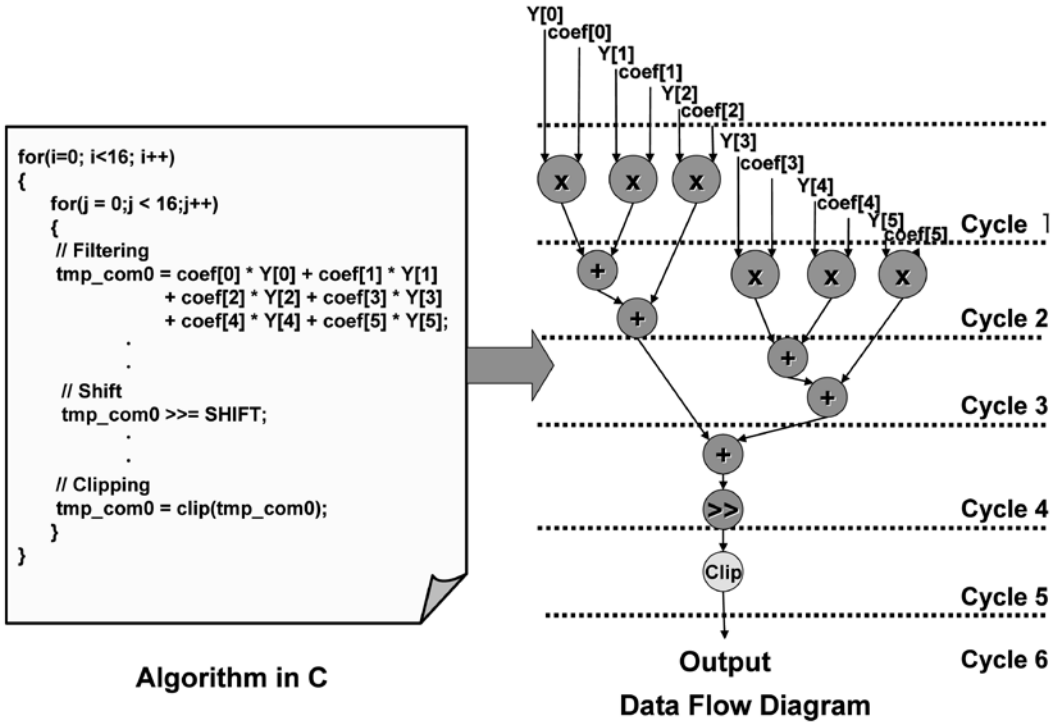

Fig. 3. Data flow Diagram.

Our ASIP core is designed based on the TCT processor. The basic architecture of the TCT processor is shown in Fig. 4 while the extended massively parallel datapath design is illustrated in Fig. 5. The key point of this datapath is the maximum utilization of parallelism. We provide 16 48-bit special registers as well as 3 parallel Multiplier and other Arithmetic Units such as Adder, Shifter, and Clipper in order to make the most of data-parallelism. Furthermore, since there are large number of Load/Store operations used in the execution, Load/Store operations are performed in parallel with other computational operations, to maximize the performance efficiency (VLIW-like behavior).

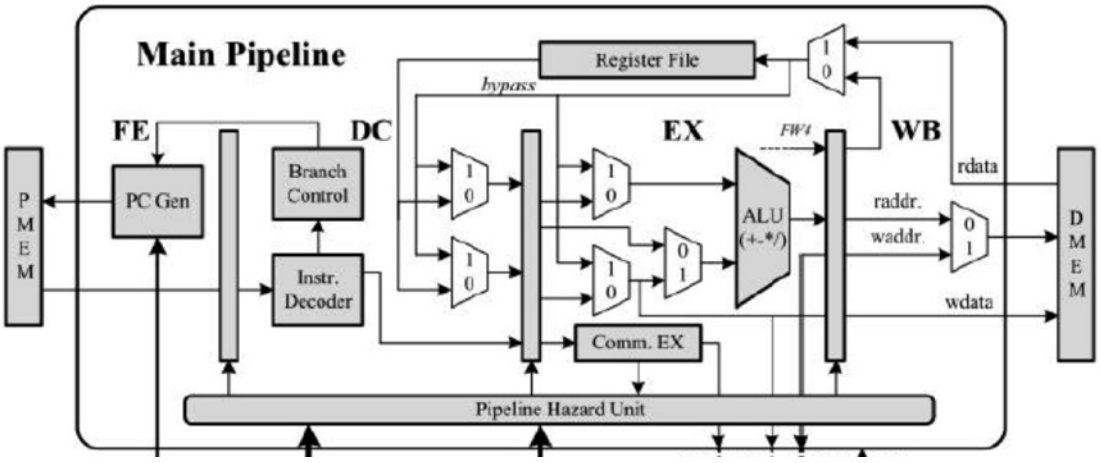

Fig. 4. Basic TCT Architecture.

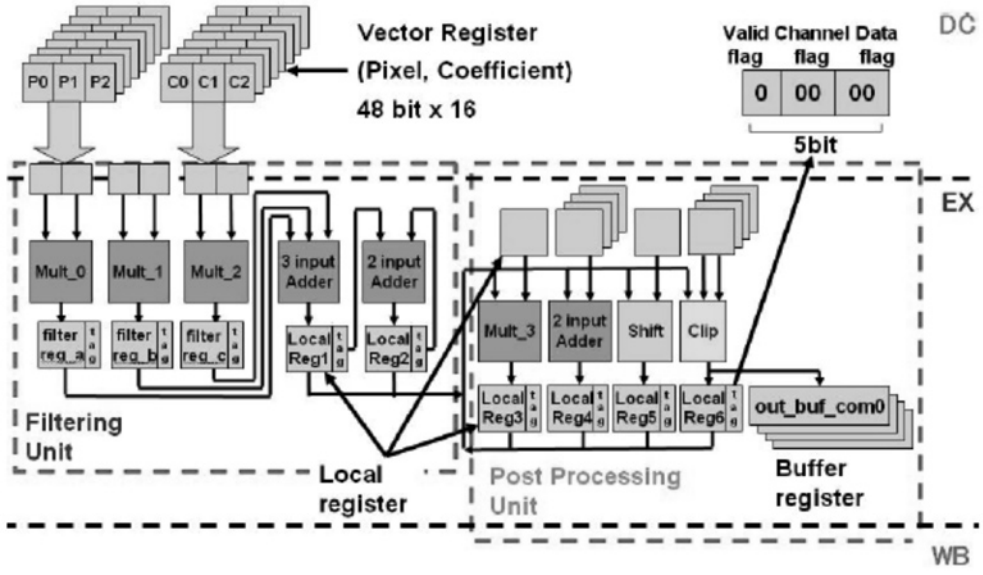

Fig. 5. Extended Parallel Datapath.

Meanwhile, we add additional 5-bit to each filtering unit registers and local registers in order to perform operation scheduling. This 5-bit of tag information will control and schedule the data flow to make sure the data order are correctly calculated in the desired way.

5.1.2 Reconfigurable Datapath

Another characteristic of our novel designed processor is its reconfigurable datapath. As stated previously, since we are aiming not only high-performance ASIP but also high level of flexibility and programmability, we designed the datapath so that it can process all the previously mentioned types of image processing. In other word, several processes can be completed using the same datapath.

The datapath can perform not only different processing, but also use a different tap number. In the image processing field, since there is high possibility of configurational change, the demand of flexibility becomes higher. Consequently, the datapath needs to be designed so that it can provide certain level of flexibility to support different tap-number processing as well as different image applications.

The datapath can be configured based on the processing requirements. Input source and operating mode of arithmetic units (adder, multiplier etc.) are programmable. There are four selections for each arithmetic unit. For instance, the shifter right shifts 8 bits in mode 0, 16 bits in mode 1, 24 bits in mode 2, and shifts based on special register bits in mode 3.

In order to improve the flexibility of the datapath, intermediate registers which are used to memorize the computing results of each arithmetic unit, are combined with a set of tag registers. These 5 bits tag registers represent valid field (bit 4), which is used to determine the status of the current value in the intermediate register, channel selection field (bit 3 and bit 2), and the mode of arithmetic unit field (bit 1 and bit 0).

In the designed datapath, by just setting up the type of desired image processing as well as tap number with a special instruction in the C code, a programmer can freely and easily implement several images, processing with varying tap-numbers, on the same datapath. Afterwards, the control signal of processor will manage the data flow and execution based on the information stated by the programmer in the C code.

5.1.3 C-Programmable Environment

The designed ASIP is completely C-Programmable. Unlike other common ASIP design methodology which rarely support a C-Programmable environment, our ASIP excels in programmability in the way that it offers C-Programmable application development.

Fig. 6 shows the sample of C-coded application program developed using our ASIP. As illustrated in Fig. 6, the massively used special instruction of the ASIP core is fully developed and written in C. A programmer can state the special instruction as if they are writing a C-function. Instruction intrinsic code of a special instruction will be stated in the C-code 1 time, then, our compiler tool will generate the object code. Moreover, in order to further optimize the application, our compiler also supports the direct assignment of a register. In Fig. 6, for example, R16 is set to the value of 255, and then afterwards MV_LOCAL_REG instruction is defined so that it will take an R16 value as an operand. By offering this C-level direct register assignment, the image processing program can be further optimized to achieve fast-speed processing while at the same time the programmability aspect is very much maintained.

```
//LD_VEC_REG(src1, vecdest_vhflag, data_pos, data_mask, mode)
        R20 = LD_VEC_REG(R20, 8, 1, 3, 0);
        R20 = LD_VEC_REG(R20, 8, 3, 3, 0);

//MV_LOCAL_REG(src1, tag, destreg)
        R16 = 255;
        MV_LOCAL_REG(R16, 16, 14);
        R16= 0;
        MV_LOCAL_REG(R16, 16, 15);
        R16 = 15;
        MV_LOCAL_REG(R16, 16, 7);

//FILTER(src1, vecsrc1_channel, vec_src2, mem, pixflag) pixel 2
        R18 = FILTER_EXCEPT(R18, 0, 14, 1, 0);
        R19 = FILTER(R19, 2, 15, 3, 0);
        R18 = FILTER(R18, 8, 14, 1, 0);
        R20 = FILTER(R20, 10, 15, 3, 0);
        R18 = FILTER(R18, 16, 14, 1, 0);
        r21 = FILTER(r21, 19, 15, 3, 0);
```

Fig. 6. C-Programmable Application Development.

6. Results

In this section, we will show several results we obtained from the proposed ASIP.

6.1 Performance Comparison

Before describing the performance of ASIPs, we would like to define our implementation conditions and some of the processing requirements. The working frequency of ASIP is set at 500MHz. The maximum allowable processing cycles for each pixel can be calculated as

$$Cycles\_per\_Pixel = \frac{Working\ frequency(MHz)}{Output\ resolution * Frame\ rate} \quad (1)$$

The required performance for different video standards is shown in Table 1. ASIPs which are able to support a specific video standard have to meet the required processing budget. The ASIP (TCT-PE-1D) is designed to be able to process scaling and edge enhancement with only software modification. In the horizontal scaling experiment, we use a 1D (1-Dimention) ASIP to enlarge a 8x16 pixel block to 16x16, and then enlarge a 16x8 pixel block to 16x16 in the vertical scaling experiment. In horizontal and vertical edge enhancement, a 16x16 block is processed in horizontal and vertical directions separately. The performance improvement of the ASIP and the increase in area are shown in Table 2.

In the horizontal scaling experiment, basic TCT-PE uses 133.37 cycles to complete one pixel processing, but our ASIP uses only 7.99 cycles. In the vertical scaling experiment, TCT-PE uses 142.16 cycles to complete one pixel processing, but our ASIP uses only 8.35 cycles. In horizontal and vertical edge enhancement, TCT-PE uses 92.63 cycles and 92.90 cycles, respectively, but a 1D ASIP only uses 5.74 cycles and 5.672 cycles respectively. The speed-up of ASIP in these experiments is 16.68, 17.01, 16.13 and 16.36, respectively. Our ASIP can achieve the 8 cycles per pixel design target which is required to support HD1080 applications, except for vertical scaling processing.

Table 1 Performance budget of video standards.

Video Standard	Resolution	Frame rate (fps)	Cycle per pixel
HD 1080	1920x1080	30	8.04
HD 720	1280x720	30	18.08
WVGA	854x480	30	40.69
VGA	640x480	30	54.25

Table 2 Performance of ASIP Compared to TCT-PE.

Processing	Processor	Processing
Horizontal scaling	TCT-PE	133.37
	TCT-PE-1D	7.99
Vertical scaling	TCT-PE	142.16
	TCT-PE-1D	8.35
Horizontal edge enhancement	TCT-PE	92.63
	TCT-PE-1D	5.74
Vertical edge enhancement	TCT-PE	92.90
	TCT-PE-1D	5.67

In order to provide a more comprehensive discussion, we also conducted processing on the designed ASIP as well as a state-of-the-art DSP Video Processor. For the DSP video processor, we apply TMS320DM6437 video processor of Texas Instruments Corp [14]. We performed cycle-accurate simulation by using CCStudio V3.3 Compiler + Board Evaluation (Cycle-Accurate) with maximum level of optimization (O3). Meanwhile, as for our designed ASIP, we use Synopsis Corp. Processor Debugger in order to perform a cycle accurate simulator.

The processing time required by each core will be then compared. We use 4 targeted applications which are horizontal/vertical scaler, horizontal/vertical, and non-linear enhancer, as the benchmarks.

Table 3 shows the throughput of the conventional general processor (TCT-RISC), DSP and the proposed ASIP on different benchmarks. As shown in Table 3, as expected, the general purpose processor TCT-RISC can not achieve competitive performance due to processing inefficiencies. On the other hand, the DSP TMS320DM6437 outperformed TCT-RISC processors excellently, up to 10 times faster. Yet, our proposed ASIP can perform computations, 2 up to 3 times, faster than the DSP Core. Hence, we believe that we accomplished the target of designing high-performance ASIP.

6.2 Resource Comparison

In order to provide a comprehensive discussion, we also investigated the area comparison between TCT-RISC, DSP TMS320DM6437, and our proposed ASIP. Table 4 illustrated the resource comparison between different cores.

Table 3 Throughput on Different Benchmarks.

Throughput (cycles/pixel)	TCT-RISC	TCT-ASIP	TMS320 DM64x
Horizontal Scaling	133.37	7.99	**15.20**
Vertical Scaling	142.16	8.35	**16.30**
Horizontal Enhance	92.63	5.74	**12.02**
Vertical Enhance	92.90	5.67	**12.11**

Table 4 Resource Comparison on Different Cores.

Resource Comparison	TCT-RISC	TCT-ASIP	TMS320 DM64x
Technology	180 nm	90 nm	90 nm
Frequency	100 Mhz	500 Mhz	600 Mhz
Normalized Area	0.123	0.292	**34.5**

From Table 4, our designed ASIP consumed the smallest area among the above cores. This is very understandable since the ASIP was developed from a simple basic RISC Processor with Harvard Architecture, the TCT Processor. Compared to the TCT Processor, the area indeed increased as we extended and customized the datapath. In 90 nm technology, our ASIP consumed 125% larger area than a basic TCT Processor. However, looking at the bigger picture, our ASIP can run up to 16 times faster that a TCT a processor, meaning that the area efficiency is about 8 times higher than a TCT processor.

6.3 Development Time and RTL Efficiency

We use Synopsys Processor Designer to implement an ASIP core. This tool enables us to shorten the total development time for an ASIP by describing the architecture in high level architecture description language (LISA). Thanks to this powerful tool, the design of our ASIP can be completed in an average of two weeks, which is a very significant level in terms of short development time. As for RTL code of the design, it is generated automatically by using the same tool. However, the drawback is that extra area usage is unavoidable. Manually coded TCT-PE with special instructions and a communication module only used 37,707 gates, but when we implemented it in Processor Designer, it used 39,216 gates. However, while the gate count increases by about 4% with TSMC 180 nm technology, it is acceptable since the development time can be shortened to a great extent.

7. Discussion: Room for Improvement

We accomplished our target in designing a high performance and programmable ASIP. However, we believe that there are still some room for improvement in order to realize a highly optimize ASIP for image processing. We list some aspects that can be improved in the future below.

7.1 ALU Occupancy for Different Tap

We get the most performance optimization by appending the extended datapath specialized for image processing into the core architecture. By utilizing this technique, we succeeded in achieving a high level of optimization on both data and instruction level parallelism. However, probing more thoroughly, we found that the extended datapath occupancy for different tap numbers still can be improved.

Table 5 shows the occupancy level of the multipliers on the filtering unit (FU) of the extended datapath. In this datapath, as shown in Fig. 5, the 3 multipliers are placed in parallel. Thanks to this technique, we can get optimal processing on performing the targeted work with fewer cycles. For a number which is the factor of 3 it worked excellently indeed, however, if the tap number is not the factor of 3, then the occupancy rate of these multipliers is deteriorated. As a result, the performance for a number which is not the factor of 3 will be limited and forced to have about the same cycles with the closest bigger factor of 3-tap number. On the above case, we can see clearly that, while in an ideal case, the cycles required for 4-tap and 5-tap processing should be less than the cycles required for 6 tap, they need approximately the same required cycles as 6-tap processing.

This deficiency comes from the different multiplier occupancy rate from different tap number. As for the tap number which is a factor of 3, the multiplier occupancy rate will be 100%, meaning that all multipliers on FU will be busy all the time. This brings out the most optimal performance. However, for a tap number which is not the factor of 3, the processing will share only 33% or 66% of multiplier occupancy level. This means that out of 3 maximum multipliers, one or two multipliers will be idle since they have to wait, in order to match the scheduling mechanism of the current datapath.

We believe that we can improve this issue by applying several alterations. If we apply a more occupancy-aware scheduling and data-flow mechanism, we can heighten the level of ALU occupancy and have optimal performance. If such an improvement is employed, than the required cycles on different tap numbers will exactly correspond, when the tap number is a factor of 3 or not. This will be one of our future works.

7.2 Evaluation on DCT Benchmark

Moreover, to provide further comprehensive discussion and clear performance evaluation about the proposed ASIP, we conducted another experiment by comparing required cycles of the proposed ASIP with TI corp. Digital Signal Processing (DSP) TMDSVDP6437 with the Discrete Cosine Transform (DCT) benchmark. DCT is a widely used essential algorithm for image processing. Hence, we want to see how far our ASIP can cope with DCT although it is beyond our target application. As for DSP, we use the tuned library file provide by TI, which is highly optimized and hand-written assembly code.

Table 5 ALU Occupancy on Different Tap.

	3-Tap	**4-Tap**	**5-Tap**	**6-Tap**
Cycles	1100	1987	2005	2045
Multipliers Occupancy	100%	33%	66%	100%

Fig. 8 shows the required cycles of 64-point 2D DCT. As shown in Fig. 8, we can see that our ASIP is only 2 times faster compared to conventional RISC, and 10 times slower than the DSP engine. This is because, our ASIP datapath can not leverage massive parallelism calculation flow in the most widely used DCT algorithm, which is the Chen algorithm [15]. While the Chen algorithm butterfly calculations are excessively performed, our implementation implements DCT from a primitive definition which uses $O(N^2)$ computations, while the Chen algorithm in DSP uses $O(N/2 \log_2 N)$.

Considering this, further improvement of datapath is needed in order to make the ASIP core process a more diverse and wider scope of image processing programs. If such a datapath can be improved, then we expect our ASIP to perform near the DSP performance level, with the advantages of lesser circuit area, and higher flexibility and programmability. This will be one of our future works.

8. Conclusion

In this paper, we proposed novel ASIP methodology and architecture for image processing. Based on the simulation result, the proposed ASIP overwhelmed conventional RISC General Purpose Processors by 16 times. Moreover, the designed ASIP can perform 2~3 times faster than DSP cores, with almost 100 times lesser area than DSP cores. Our ASIP also can provide certain level of flexibility in order to process several image processing algorithms and different tap-number using the same core. Our advanced and integrated design methodology also brings out the short development period of ASIP, resulting in an average of two weeks for design time.

Based on this ASIP development experience, we believe that as long as we can tailor both the ASIP Core architecture and its interconnects easily in a designer-friendly environment with powerful tools support, An ASIP is promising and has significant potential to emerge as an ideal solution in future computing systems. It can readily be said that tailoring hardware to a specific application is the most effective arsenal that we have in order to compute efficiently while at the same time maintaining flexibility and programmability.

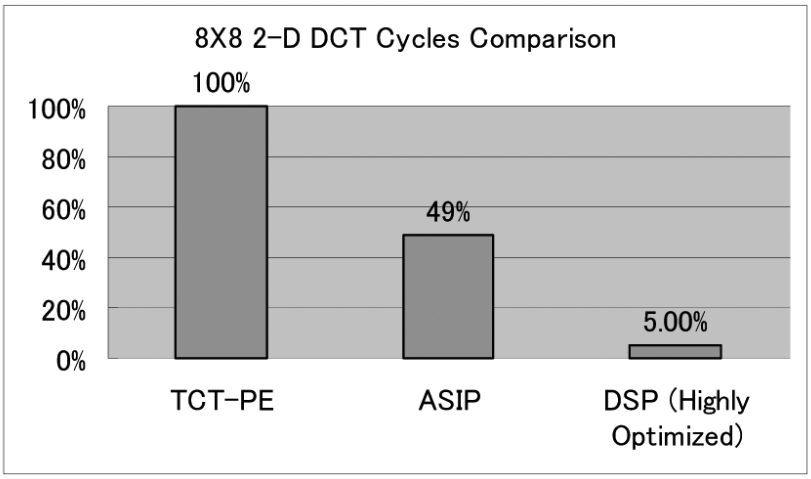

Fig. 8. DCT Performance Comparison.

9. References

[1] William Dally et al. "Stream Processors: Programmability with Efficiency" ACM Queue, pp. 52 - 62, March 2004.

[2] V. Agarwal, M.S. Hrishikesh, S. W. Keckler, and D.Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures," In the 27th Annual International Symposium on Computer Architecture, May 2000.

[3] G. Goosens, J.V. Praet, D. Lanneera and W. Geurts, "Ultra-low Power? Think Multi-ASIP SOC," In the 7th International Forum on Embedded MPSoC and Multicore, June 2007.

[4] R.Hameed, W.Qadeer, M.Wachs, O.Azizi, A.Solomatnikov, B.C. Lee, S. Richardson, C. Kozyrakis, M. Horowitz, "Understanding sources of inefficiency in general-purpose chips" In the 37th Annual International Symposium on Computer Architecture, May 2010.

[5] O. Shacham, O. Azizi, M. Wachs, W. Qadeer, Z. Asgar, K. Kelley, P. Stevenson, A. Solomatnikov, A. Firoozshahian, B. C. Lee, S. Richardson, M. Horowitz, Why design must change: Rethinking digital design, IEEE Micro, (2010).

[6] TU (International Telecommunication Union) recommendation ITU-R BT.1769, www.itu.int/md/R07-SG06-C-0060.

[7] S. Gohshi, M. Teragawa, H. Mikami, S. Imai, The Novel Signal Processing Method for Super Resolution, BCT ITE-BT, Vol. 33, No. 25, BCT2009-57, pp. 9 - 13, June 2009.

[8] M. Z. Urfianto, T. Isshiki, A. U. Khan, D. Li, H. Kunieda, Decomposition of Task-Level Concurrency on C Programs Applied to the Design of Multiprocessor SoC. IEICE Trans 91-A(7), pp. 1748 - 1756, 2008.

[9] M. Z. Urfianto, T. Isshiki, A. U. Khan, D. Li, H. Kunieda, A Multiprocessor SoC Architecture with Efficient Communication Infrastructure and Advanced Compiler Support for Easy Application Development, IEICE Trans. 91-A(4), pp. 1185 - 1196, 2008.

[10] A. Misuno, K. Kohno, R. Ohyama, T. Tokuyoshi, H. Uetani, H. Eichiel, T. Miyamori, N. Matsumoto, M. Matsui, Design Methodology and System for a Configurable Media Embedded Processor Extensible to VLIW Architecture, In Proc. ICCD'02, pp. 2 - 7, 2002.

[11] O. Schliebusch, A. Hoffmann, A. Nohl, G. Braun, H. Meyr, Architecture Implementation Using the Machine Description Language LISA, In Proc. ASP-DAC 2002, pp. 239 - 244, 2002.

[12] M. Imai, Y. Takeuchi, A. Shiomi, J. Sato, A. Kitajima, An Application Specific Processor Development Environment: ASIP Meister, ICICE technical report ICD, Vol. 102, No. 401, pp.39-44, October 2002.

[13] Processor Designer/Processor Debugger/Design compiler, http://www.synopsys.com.

[14] DM6437 Digital Video Development Platform, http://www.ti.com/tool/tmdsvdp6437.

[15] W. H. Chen and S. C. Fralick, Image enhancement using cosine transform Fltering, Image Sci. Math. Symposium., Monterey, CA, November 1976.