Application Specific Instruction Set Processor (ASIP) Design in an 8-bit Softcore Microcontroller

Sani Irwan Md Salim, Yewguan Soo, Sharatul Izah Samsudin
Center for Telecommunication Research and Innovation(CeTRI), Faculty of Electronic and Computer Engineering, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia
sani@utem.edu.my

***Abstract*—The microcontroller-based system is currently having a tremendous boost in demand in line with the Industrial Revolution 4.0. Although more applications seem to concentrate on software algorithms and wireless connectivity, the hardware side of the system is still occupied by microcontroller variants. With huge alternatives being offered to setup a microcontroller system, having a softcore microcontroller is extremely beneficial especially when considering the rapid advancement in computer technology. Although the 8-bit microcontroller has less computational capability compare to other high-end microcontroller families, it has an advantage in low code density for I/O application and control. The purpose of this research is to combine the best feature of the 8-bit architecture together with efficient arithmetic operations in the implementation of moving average filter. The modules' integration is implemented using ASIP design without occurring extra board space and is developed using the Field Programmable Gate Array (FPGA) as the single chip solutions. It was found that the revised microcontroller architecture has produced a faster execution time and similar maximum frequency when benchmarked with its predecessor. The overall ASIP design procedures used in this research provides flexibility for further development, either by extending its module to incorporate more complex algorithms or by upgrading current designs of its components.**

***Index Terms*—ASIP; Softcore Microcontroller; Moving Average Filter; FPGA.**

I. INTRODUCTION

Various processor architectures have been developed especially in the past decade that populated the embedded and computing system market. The terms of these processors are also evolving in order to differentiate its functionality and potential in fulfilling broad system requirements. There are microprocessors, digital signal processors, microcontrollers and digital signal controllers that did not just offered different capabilities but also implementing different optimization techniques to suit specific design requirement [1-4].

Across the range of the embedded and computing systems, the processors are classified to their bit architecture that referred to its data bus width. Low-end processors are usually quoted with 4/8-bit architectures while 16-bit architectures are considered mid-range processors. Larger processors, for instance, 32-bit architecture or bigger, are the top-of-the-range processors. The selections of these processors are dependent on several factors such as cost, workload capacity, and energy consumption. The implementations of different processing platforms are also reliant on the type of data processing with regards to the tasks' workload and context. Cravotta [5] has published a useful processor architectural mapping to clearly explain the topographical representation of processor based on the processor's bit width and also classification

There is also the low code density issue which is often associated with the 8-bit microcontroller. In general, code density refers to the combined size of all the instructions needed to perform a particular task. Low code density means that the architecture required more basic instructions repeatedly to execute a task. This assumption is correct only if the 8-bit microcontroller is demanded to perform 32-bit mathematic computations. For control applications, the 8-bit microcontrollers do not suffer from the low-density code due to the offloading techniques for the particular functions from the main processor. Normally, the computationally intensive tasks are offloaded to be performed by a specific hardware module. Some devices also integrate wireless SoC to send/receive signals over Bluetooth or Wi-Fi to eliminate the need to include the code for that function which is a similar concept to the IoT. Because of the limited space, the codes need to be written efficiently and uses the smallest space possible to keep a minimum processing cycle. This is also where the assembly language is the best option for efficient coding. Since 8-bit MCUs have very little overhead code, total code density for control-type functions is higher than equivalent functions implemented on 32-bit MCUs.

As part of the SoC platform, general purpose processor usually has a high degree of flexibility, a friendly design environment, and sufficient design references. This platform is preferred when requirements for power, performance and silicon area are not very critical. When these requirements are strict, ASIP will become a necessity.

Essentially, ASIP is an architecture that includes two parts which are a minimum Instruction Set Architecture (ISA) and a configurable logic which can be used to design a customized instruction set. Thus, it provides relatively high flexibility compared to ASIC and better performance compared to FPGA. The main advantage of the ASIP is the instruction set can be built to meet the system's specific requirement my configuring the ISA.

II. UTEMRISC SOFTCORE MICROCONTROLLER

A soft-core is a processor, a microcontroller or a digital signal processor, which is integrated as a virtual unit in an FPGA or ASIC design. The soft-core processor is a terminology that reflects the reprogrammable nature of the processor architecture. Usually implemented in the programmable platform, the soft-core processor is described using hardware description language and implemented using logic synthesis. Soft cores are used in the FPGAs to perform complicated tasks, but at the same time do not place too high demands on speed. On the contrary, the hard-core processor

is defined as processor core that is physically implemented as a structure in the silicon. The hard-core processor in ASIC is also initiated from a soft-core processor in FPGA platform, but during the conversion is from FPGA to ASIC, the core processor is placed as the real hard-core on the chip. The obvious limitation of the soft-core processor when compared to the hard-core processor is the processing speed. Hard-core provide better processing speed due to its cell optimization in the silicon. However, for design flexibility and multi-core processor setup, the soft-core processor is the way to go. Soft-core processors can be easily modified and customized to specific requirements and to include more features. Its core also could be synthesized in the manifold to generate a multi-core processor. Small soft-cores for example, can be placed and used in parallel in the FPGA which would increase the data transfer capacity in certain applications.

UTeMRISC0 is a soft-core processor that is essentially based on the PIC architecture, that is also the fundamental architecture of several other soft-core processors previously mentioned in this thesis. The UTeMRISC0 soft-core processor was first introduced in [6] as a part of the experimental setup to maximise the capability in delivering highest performance achievable by a processor core compared to their physical IC counterpart. Figure 1 shows the insignia of the UTeMRISC softcore families.

Figure 1: Insignia of UTeMRISC processor

Further research has been done by Salim, et al. [7] to improve the architecture of the UTeMRISC0 by eliminating several features that detrimental to the overall performance. For instance, the memory bank structure, which is inherited from the original PIC architecture and maintained in the other PIC-based soft-core processor, is removed. This will omit the need to change memory banks to access a certain range of memory address in the programming code. In place, a new single bank memory structure is introduced with an algorithm is developed to determine the final memory address in the memory space [8-12].

III. METHODOLOGY

A. ASIP Design

Details on the ASIP design methodology has been explored by Liu [13] that includes the element of hardware/software co-design. This element has been neglected or hidden in the other ASIP methodologies due to its complexity in determining the correct stages when the hardware and software parts are designed concurrently. the relation between hardware and software design stages and how it interacted or correlated with each other [13]. In the code synthesis, the instruction set specification is laid out during the instruction set extension should be mirrored to the actual processor architecture, specifically involving the instruction set architecture. To reflect the major changes made to the architecture, the soft-core processor is renamed as UTeMRISC01.

1) *Application Design and Constraints*

The main function of the moving average filter is to determine the windows for buffering and then performing the summation and division. As the summation module is already available in the UTeMRISC01 architecture, that left the division arithmetic that needs to be configured. Creating a general-purpose divider module would be a challenging task and involved another specific algorithm such as Booth's algorithm. Inserting this module also would unnecessarily increase the resource utilization that in the end would enlarge the whole processor architecture size. To keep it simple, the division part is limited to perform power of two division. Effectively, the division by two required the bits to be shifted one place to the right. Normal PIC instruction has the shift instruction in place. However, as the windows buffer getting larger, several bit shift operations are needed to complete the division process. This would affect tremendously the overall instruction cycles. A common solution, which is widely applicable in the higher-end DSC architecture is by employing barrel shifter. In this case, the shift operation could be executed in a single instruction cycle within the destination register.

2) *Processor Architecture Exploration*

As the single-bit shift operation is already available in the UTeMRISC0, the ISA now needs to include the operand that indicates more than single bit shifting to be executed by the ALU module. The bit shift operation in the ALU module is modified to instantly perform the bit shifting as per the operand's input and executed in one clock cycle.

3) *Instruction Set Generation.*

Two new instructions called 'bsrf' and 'bslf' are created that stands for 'barrel shift right (f)' and 'barrel shift left (f)' respectively. The instruction's operand consisted of the address of the target register and the 3 bits that indicates how many times the data would be shifted. A new opcode is also assigned to this instruction.

Table 1
New Instruction for Moving Average Program

CPU Sim		Format	16-bit Opcode
Opcode	**Mnemonics**		
0B	bsr f	6 7 3	0010_11ff_ffff_fbbb
0B	bsl f	6 7 3	0010_11ff_ffff_fbbb

4) *Code Synthesis.*

The new instruction the is simulated using the CPUSim software with the UTeMRISC01 architecture is loaded as the main processor execution. The new instruction's micro-operation is defined in this process to iterate the sequence of the instruction execution. Figure 2 and 3 show the machine instruction edit done in the CPUSim software.

5) *Hardware Synthesis.*

The findings from the code synthesis stage are brought forward to the FPGA implementation by replicating the instruction execution sequence, this time in Verilog code. The instruction's opcode and ISA are updated in the instruction decoder and ALU module. Once the architecture is successfully synthesized and implemented, the performance parameters are observed and measured using the logic analyzer.

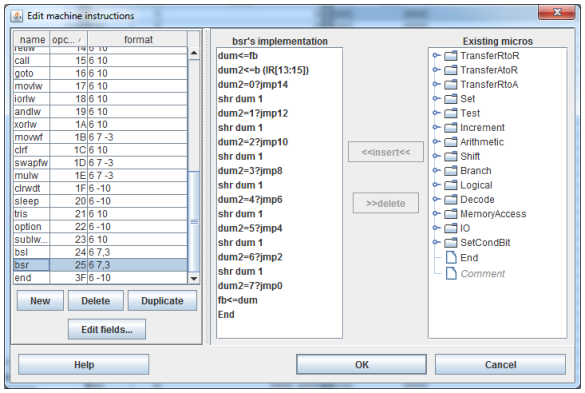

Figure 2: Machine Instruction Edit for 'bsr' in CPUSim

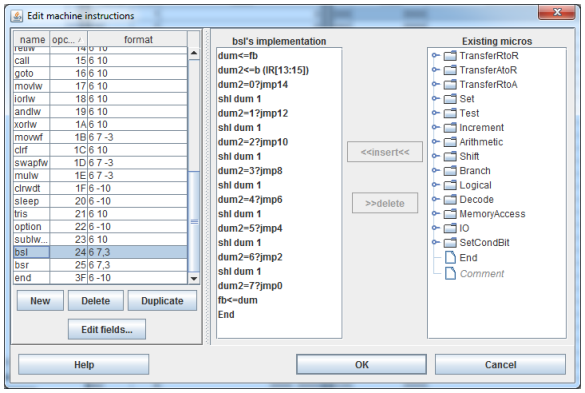

Figure 3: Machine Instruction Edit for 'bsl' in CPUSim

B. Related Tools

The ASIP design is implemented in Xilinx ML605 FPGA board, and the retargetable assembler is developed from the ground up using the Visual Basic software by Microsoft. The complete workflow of ASIP design starts from the initial design phase, behavioral simulations, instruction set simulations, logic synthesis, design implementation, hardware configuration and data collections.

During the early phases of the design, the new instruction is tested and simulated using the CPUSim software. CPU Sim is a Java application that allows users to design simple computer CPUs at the microcode level and to run the machine-language or the assembly-language programs on those CPUs through simulation [14]. It is a very useful tool to simulate a variety of architectures, especially the RISC-like and the accumulator-based architecture that is adopted in this research. Using this software, the current instruction can be modified, and new instruction can be loaded to the existing instruction set. Furthermore, the CPUSim software also assisted in reconfiguring the instruction set architecture (ISA) by providing the ISA structures with the related opcode and operand configurations. This configuration then will be the reference point in designing the retargetable assembler and the hardware module of the ISA itself in HDL.

The tools used in the ASIP hardware implementation are exclusively done on Xilinx environment, including its Integrated Synthesis Environment (ISE) Design Suite. The Xilinx ISE Design Suite is responsible for the bulk of the work by providing the tools to develop HDL modules in the processor architecture. The behavioral simulations are completed by using the ISE simulators, also known as ISim, which its main purpose is to perform functional and timing simulations for Verilog designs. Xilinx ISE will perform the logic synthesis and design implementation to generate the bit file. Using a software called iMPACT (integrated with the Xilinx ISE), the bit file is configured and loaded to the FPGA board, in this case, ML605 FPGA board).

C. Moving Average Filter

In digital signal processing, the moving average is the most common filter largely contributed by its simplicity and easy to understand. Its main task is to reduce random noise while retaining a sharp step response. For time domain encoded signals, the moving average is the easiest digital filter to use.

Fundamentally, the moving average filter operates by averaging a number of points from the input signal to produce each point in the output signal. Figure 4 shows the flowchart of the moving average filter program. The equation for moving average is shown in Equation (1). In this equation, $x[\,]$ is the input signal, $y[\,]$ is the output signal, and M is the number of points used in the moving average. This equation only uses points on one side of the output sample being calculated. Programming is slightly easier with the points on only one side; however, this produces a relative shift between the input and output signal.

$$y[i] = \frac{1}{M}\sum_{j=0}^{M-1} x[i+j] \tag{1}$$

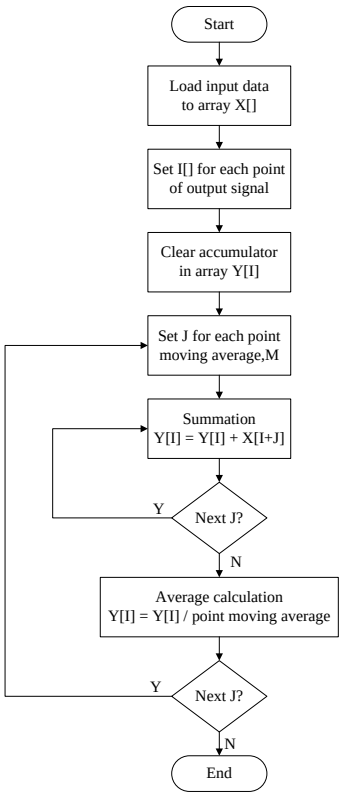

Figure 4: Flowchart of the moving average filter

IV. RESULTS AND DISCUSSION

Moving average filter algorithm required more arithmetic calculation compare to the available instruction in the base processor. Although the base processor is capable of performing operations such as data shifting and division, the lack of dedicated instructions means that those calculations would take several clock cycles to be completed. The operations can only be made using several basic instructions, and additional loops are required to finish the function. In

moving average filter algorithm, new instructions for a faster bit-shifting is created. In term of the processor hardware, barrel shifter module is also created in the architecture to accommodate better division operation, which includes dividing the data set by 2^n.

A. Execution Times

During the architecture implementation for both processors, the execution times are decreasing correspondingly with the increase of the oscillator clock frequency. There is marked improvement in the execution time as the moving average filter finished faster in the UTeMRISC01 processor compare to the UTeMRISC0 processor. Figure 5 shows the execution times for both architecture, with UTeMRISC01's times are 18.6% average faster than the UTeMRISC0's times. This is mainly contributed to the reduction of instruction cycle in the UTeMRISC0 which uses specialized single-cycle bit-shifter. In comparison, the UTeMRISC0 processor needs to perform several loops of single-bit-shifting to achieve the same results.

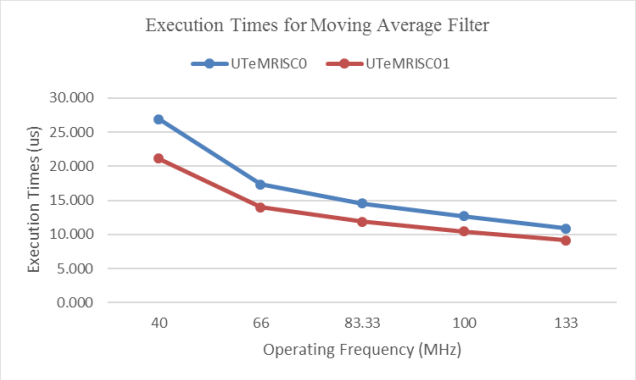

Figure 5: Execution Time for Moving Average Filter

B. Maximum Allowable frequency

Figure 6 shows the maximum allowable operating frequency observed for both processor architecture. The UTeMRISC0's maximum frequency recorded as high as 127.959 MHz, 2% higher than the maximum frequency for the UTeMRISC01 architecture. Further analysis in the slack time also revealed the all timing requirements are met hence the minor increase in the maximum frequency of the UTeMRISC0 is due to the lesser critical path delay when executing the basic instructions. However, the difference is so small that the UTeMRISC01 could achieve or surpass the maximum frequency with further ALU design optimization.

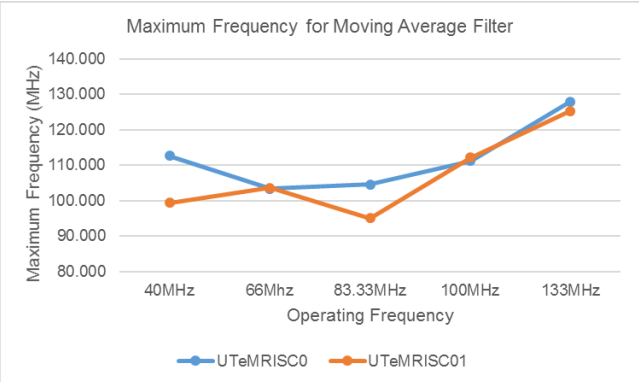

Figure 6: Maximum Operating Frequency of Moving Average Filter

However, as more module and complex calculations are introduced to the system, the maximum frequency for the UTeMRISC01 is slightly lower than the normally-operated UTeMRISC0. The additional shifting operation has caused the critical path to increase, and it is affecting the outcome of the maximum frequency. During this stage, all timing constraints are met without any violations recorded.

V. Conclusion

The impact of having a customized instruction set of specific applications has been demonstrated in this thesis with regards to the moving average filter implementation. Even though the instruction set modification is generally overlooked to avoid compatibility problem on the customer side, there are also performance parameters that can be gained by streamlining the instructions geared towards a specific application. This paper has successfully demonstrated the effectiveness of having an extended instruction set architecture that leads to better optimization of instruction program while at the same time improve the program execution period.

Acknowledgment

The author would like to thank Universiti Teknikal Malaysia Melaka and Ministry of Higher Education Malaysia for the financial support through the research grant number PJP / 2016 / FKEKK-CETRI / S01496.

References

[1] A. Baysal and S. Sahin, "Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors," IACR Cryptology ePrint Archive 2015.

[2] J. Donovan. (2014, 2 July). *Is There a Future for 8-Bit MCUs?* Available: http://www.digikey.com/en/articles/techzone/2014/feb/is-there-a-future-for-8-bit-mcus

[3] J. Ganssle. (2012, 2 July). *Is 8-bits dying?* Available: http://www.embedded.com/electronics-blogs/break-points/4389890/Is-8-bits-dying-

[4] R. Cravotta. (2007, 20 July). *Putting the Squeeze on 16-bit Processors*. Available: http://www.edn.com/design/systems-design/4314333/Putting-the-squeeze-on-16-bit-processors

[5] R. Cravotta. (2012, 11 July). *One Processor to Rule Them All?* Available: http://www.edn.com/design/systems-design/4398890/One-processor-to-rule-them-all

[6] L. E. Yong and A. J. Salim, "Implementation of an 8-bit RISC Microcontroller Chip," in *4th International Symposium on Broadband Communication*, 2010, pp. 1-4.

[7] A. J. Salim, S. I. M. Salim, N. R. Samsudin, and Y. Soo, "Instruction Set Extension Through Partial Customization of Low-End RISC Processor," *Australian Journal of Basic and Applied Sciences*, vol. 7, pp. 678-687, 2013.

[8] A. J. Salim, S. I. Salim, N. R. Samsudin, and Y. Soo, "Educational development tools for software and hardware processor design," in *Proceedings - 8th EUROSIM Congress on Modelling and Simulation, EUROSIM 2013*, 2015, pp. 622-627.

[9] A. J. Salim, S. I. M. Salim, N. R. Samsudin, and Y. Soo, "Customized instruction set simulation for soft-core RISC processor," in *IEEE Control and System Graduate Research Colloquium (ICSGRC)*, 2012, pp. 38-42.

[10] A. J. Salim, S. I. M. Salim, N. R. Samsudin, and Y. Soo, "Conversion of an 8-bit to a 16-bit Soft-core RISC Processor," *International Journal of Electronics Communication and Computer Technology*, vol. 3, pp. 393-397, 2013.

[11] A. J. Salim, N. R. Samsudin, S. I. M. Salim, and S. Yewguan, "Modification of Instruction Set Architecture in a UTeMRISCII Processor," *International Journal of Computer Trends and Technology (IJCTT)*, vol. 4, pp. 1196-1201, 2013.

[12] A. J. Salim, N. R. Samsudin, S. I. M. Salim, and S. Yewguan, "Multiply-accumulate instruction set extension in a soft-core RISC

Processor," in *10th IEEE International Conference on Semiconductor Electronics (ICSE)*, 2012, pp. 512-516.

[13] D. Liu, *Embedded DSP Processor Design, : Application Specific Instruction Set Processors*: Morgan Kaufmann, 2008.

[14] D. Skrien, "CPU Sim 3.1: A Tool for Simulating Computer Architectures for Computer Organization Classes," *Journal on Educational Resources in Computing (JERIC)*, vol. 1, pp. 46-59, 2001.