Formalisation of a Visual Environment for Real Time Image Processing in Hardware (VERTIPH)

C. T. Johnston, D. G. Bailey, P. Lyons, K. T. Gribbon
Institute of Information Sciences and Technology,
Massey University, Private Bag 11222, Palmerston North, New Zealand
C.T.Johnston@massey.ac.nz, D.G.Bailey@massey.ac.nz, P.Lyons@massey.ac.nz

Abstract

We consider the suitability of three types of language for the implementation of image processing algorithms on FPGAs; Hardware Description Languages, Parallel Language Extensions, Serial Language Extensions. We discuss the requirements for a language for this purpose and identify the weaknesses of four specific languages, VHDL, SystemC, Handel-C, SA-C and Match. Finally, we propose VERTIPH, a new multiple-view visual language that should avoid these weaknesses.

Keywords: FPGA, Visual Language, Image processing

1 Introduction

With the continual growth in size and functionality of FPGAs (Field Programmable Gate Arrays) there has been increasing interest in their use as implementation platforms for image processing applications, particularly real time video processing. Due to their structure, FPGAs can exploit the data parallelism found in images. They can either performing all required operations or perform a subset of operations to reduce data before passing processing to a standard DSP or microprocessor.

FPGAs allow increased performance by freeing the implementation from the fixed architecture of standard processors. However, they make the design and implementation of the algorithm more difficult for most image possessing practitioners as they are not familiar with FPGA-related issues of concurrency, pipelining, priming and bandwidth. Offen [1] has stated that the classical serial architecture is so central to modern computing that the architecture-algorithm duality is firmly skewed towards this type of architecture. It is the mapping from a standard) algorithm to a custom architecture which presents the greatest challenge to the novice FPGA designer.

Mapping the algorithm is only part of the problem. Real-time operation places a number of constraints on the design. For example there may be memory bandwidth constraints, concurrency may cause resource conflicts, and large expressions may need to be pipelined to reduce logic depth and prevent the design from failing timing constraints.

The constraints placed on the design depend on the processing model. We believe there are three modes: stream processing, offline and hybrid. In stream processing, data is received from the input device in a raster nature at video rates. Memory bandwidth constraints dictate that as much processing as possible be preformed as the data arrives. In offline processing there is no timing constraint and often has random access to memory containing the image data. This mode is the easiest to program, as a direct mapping from a software algorithm can be used, with the FPGA speed limited in most cases by the memory access speed. The hybrid case is a mixture of stream and offline processing. In this case, the timing constraint is relaxed so the image is captured at a slower rate. While the image is streamed in to an image buffer it can be processed, such as extracting a region of interest. This region of interest can then be processed by an offline stage which would allow random access to the elements in this region. The language should manage these constraints so the designer can focus on the algorithm development.

A high-level language for expression of image processing algorithms in hardware should aim to facilitate this. It should

- allow a mixture of parallel and sequential design
- make it clear to the designer what runs in parallel and what forms part of a pipeline
- be able to detect when concurrent processes may access a shared resource such as a RAM and manage this accordingly, perhaps by informing the designer and giving some suggestions of how to resolve the issue
- be able to handle stream, offline and hybrid processing models
- have some of the common image processing functions and data-types as primitives. Examples include row and pixel buffering, window filters, and look up tables (LUT)
- be intuitive and easy to use
- provide multiple views onto the design

The rest of this paper discusses available languages for implementing image processing algorithms in hardware and builds a case for a new visual language. Section 2 looks at current languages starting with low-level Hardware Description Languages, then high-level languages which have been extended to allow hardware design and finally sequential languages which are converted to FPGA designs. Section 3 introduces our proposed visual design language, VERTIPH, specifying the features it will have and the reasons for including them..

2 Languages

The design of algorithms for FPGAs can occur at a number of different levels of abstraction. At the lowest level is schematic entry – a time consuming and difficult process that involves converting an algorithm to a netlist. Hardware Description Languages (HDLs) are well established, then there are extensions to existing software languages and finally there language to hardware compilers, which convert serial languages to hardware with minimal changes from the standard language.

2.1 Hardware Description Languages

Verilog [2], and VHDL [3] are industry standard HDLs (Hardware Description Languages). They can be thought of as the assemblers of hardware programming, providing great flexibility at several levels from gate level through to RTL (Register Transfer Level). As they offer similar functionality, we will concentrate on VHDL.

VHDL has constructs that enable the concurrent or sequential behaviour of a digital system to be expressed with or without timing [4]. In VHDL, the basic abstraction for a digital circuit is called an *entity*. An entity comprises a declaration (which describes how it is accessed using ports) and an architecture body (which specifies the operation to be performed). This specification can be a behavioural model that describes the entity as a set of sequentially executed statements, the structure of the entity is not specified, just the function. Alternatively, the entity may be specified as a structure where a set of interconnected components are used. A dataflow representation uses concurrent signal assignments, where when any signal changes the expression is revaluated. Finally a mixture of methods can be used. VHDL [4] is highly explicit and this makes it a very verbose language. Even for a very simple operation a full entity needs to be declared including its ports and then the architecture of the actual operation.

VHDL's constructs are at a very low level, making it a poor choice for implementing complex image-processing algorithms. As a general purpose language it provides no image-processing- specific operations. HDLs in general are very powerful and flexible but can be difficult to program.

2.2 Parallel Language Extensions

Many software languages have been extended to allow the implementation of hardware in a higher level language. We will discuss SystemC [5] and Handel-C [6].

In most conventional programming languages, statements are executed sequentially following the order of assignment statements, and branches specified by flow-of-control statements (**`while-`**, **`if-`**statements, etc). In general, they do not offer the ability to run processes in parallel, although some may support process threads. The lengths of data types are defined by either the fixed architecture of the processor or by the language (ANSI C [7] or Java).

There are four main areas in which conventional programming languages need to be extended in order to support hardware design. It should be possible:

- to specify that operations occur concurrently and to specify the timing or clock speed of processes
- to define communication between processes running at different speeds
- to define data types in terms of their bit length as there is no fixed architecture to conform to
- to build architectural components such as RAMs, ROMs, WOMs, channels,
- to create low level structures such as wires along with bit level operations such as bit concatenation.

Both Handel-C and SystemC offer this type of functionality.

2.2.1 SystemC

SystemC is a modelling language based on C++. It is intended to enable system-level design at multiple levels of abstraction [5]. It extends C++ with classes for concurrency, hardware data types, clocks, reactive behaviour and communication. SystemC's main difference from Handel-C is that like VHDL, it includes the concept of modules and ports for transferring data into and out of modules. And as in VHDL, modules contain processes where functionality is described. Processes run concurrently, but code inside a process is sequential [8].

SystemsC's processes are categorised by the way they are invoked. *Methods* are sensitive to changes in a set of signals or ports and will run when these change. Methods cannot be suspended. *Threads* are similar to methods, but as in VHDL, they can be suspended.

Cthreads are a special case of thread process that are sensitive to an edge of one clock and are invoked by the active edge of that clock [8]. SystemC also provides a rich set of data types, allowing the user to set widths and to define fixed point data types for DSP operations. Cthreads are a special case of thread process that are sensitive to an edge of one clock and

are invoked by the active edge of that clock [8]. SystemC also provides a rich set of data types allowing widths to be set and fixed point data types for DSP operations.

As SystemC is a general purpose hardware design language, it does not provide any specific support for image processing. Many of the data types and structures available are not suited to image processing. The way that ports, methods and processes need to be defined is too closely related to hardware and it is not intuitive for someone used to writing C++ code. That said, SystemC provides excellent tools for the simulation of systems at both high-level and RTL levels.

2.2.2 Handel-C

Handel-C is a language that compiles algorithms written in a high-level C-like language directly into gate-level netlists. It is based on a subset of ANSI-C with syntax extensions for hardware design such as variable data widths, parallel processing and channel communication between parallel processing blocks. The language is designed to allow software engineers to express an algorithm without any knowledge of the underlying hardware[9].

Assignment statements in Handel-C take exactly one clock cycle. All other language statements, such as control logic constructs, add zero additional clock cycles although they can increase combinatorial delays to the extent that the system clock cycle may need to be lengthened [9].

Apart from the introduction of architectural constructs and bit level operations the only main differentiation from ANSI-C and Handel-C is the introduction of the **`par`** construct. All statements within a **`par`** block runs in parallel. In Handel-C the default is sequential operation.

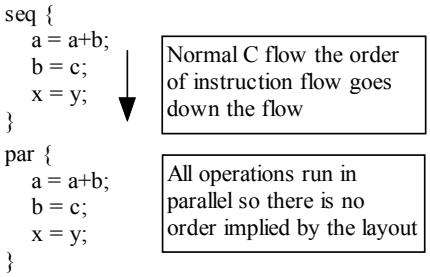

Figure 1: Logical flow of instructions

Handel-C provides a good level of abstraction from hardware design However, as Figure 1 shows, there is almost no textual difference between sequential and parallel code.

The increased level of abstraction from a hardware definition to a high level language enables the developer to concentrate on algorithm development. Overall both SystemC and Handel-C are an improvement over hardware description languages. However they do not have any features which aid the designer to visualise the effects of concurrency.

2.3 Serial Language Extensions

There are several systems which take "standard" software code and convert them to hardware. Normally there is a need to have some compiler directives for data type lengths. Two of these, SA-C (Single-Assignment C, from the Cameron project [10, 11]) and Match [12-15] (derived from MatLab) have been aimed at image processing.

Match needs to convert from the vector operation in Matlab to **`for`** loops unless there is an optimized core which can be used for the operation. It then converts complex expressions to single operation statements, in a process called *levelization*. Pipelining is done by finding data dependences within the loops and loop unrolling.

SA-C is aimed at image processing and makes some changes to the normal C model. The language does not include pointers but it does incorporate common image processing functions such as array summing for histograms, and window loops. The first step in converting to hardware is to unroll loops. Then, where possible, consecutive loops are combined into one. This is followed by standard CSE (common sub-expression elimination) and temporal CSE (replacing a computation in one loop iteration with a result computed in a previous iteration). It is through the loop unrolling that parallelism is exploited. In SA-C, if timing needs to be improved, a further step of breaking up complex expressions into pipelined sub-expressions can be invoked through compiler options.

With both Match and SA-C the designer only needs very limited understanding of the underlying hardware that implements the algorithm. Converting the algorithm from software to hardware results in significant speed improvements. However only some of the potential benefits are realised; MatchC and SAS-C can only express algorithms sequentially, where there may be a more efficient concurrent algorithm. This type of design environment also limits the designer to working in an offline mode where the desired image is loaded into memory and then processed until finished.

3 VERTIPH:A New Visual Language

There have been a number of different visual image processing languages for use on a serial computer including Khoros [16] and OpShop [17]. There are also several general purpose visual languages which can be used for image processing including LabView and Simulink.. Khoros, LabView and Simulink now have extensions that allow them to be used for FPGA design, though this was not their original purpose. Khoros offers a high level view for algorithm

development, but it was not designed to support the implementation of novel image processing operations, so it does not include lower level design capabilities.

We propose VERTIPH - Visual Environment for Real Time Image Processing in Hardware - a visual design language that aims to improve on present languages for image processing design on FPGAs. It will use multiple views of the algorithm and resources required for data storage, capture and processing.

3.1 Top Level View

A top level view will make it possible to visualise the overall flow of image processing operations. Figure 2 is a top level architectural view showing the separate components of a barrel distortion algorithm [18].

3.2 Timing View

Developers who never design their own algorithms, can assemble pre-defined library modules into a high-level overview like the one shown in Figure 2.

However, to allow the developer to design their own operations and help with buffering, pipeline priming and synchronisation, a lower level timing view is needed. To accomplish this we have modified the Gantt chart notation [19]. In this notation, time flows from left to right, so in Figure3(a), operation **`x`** is followed by operation **`y`**, which is followed by operation **`z`**. In Figure 3(b), the operations occur concurrently, and in Figure 3(c) they are pipelined. This representation is an abbreviation of Figure 3(d) which explicitly shows the parallel repeating processes that feed data from one to the other, and that each process is active in succeeding phases.

Of course, these basic types can be used together as is shown in Figure 4 which is the pipeline for the barrel distortion algorithm. This figure also shows VERTIPH's **`if-`** and **`while-`** control structures provided by the language which are based on the control structures used in Nassi-Shneiderman diagrams [20]. This pipeline view conveys to the developer graphically the time required to prime and flush the pipeline.

Operations can be registered or unregistered with unregistered operations needing feed to a register before a clock cycle can finish. To save space on the screen only the operation or register name is shown, a operations key has the instructions for the block in a Handel-C type syntax. This view shows the same information as a textual language but the layout makes the structure of the algorithm easier to visualise. For example it is immediately obvious that the x value must be offset by 3 before it is used in the calculation of the undistorted x value.

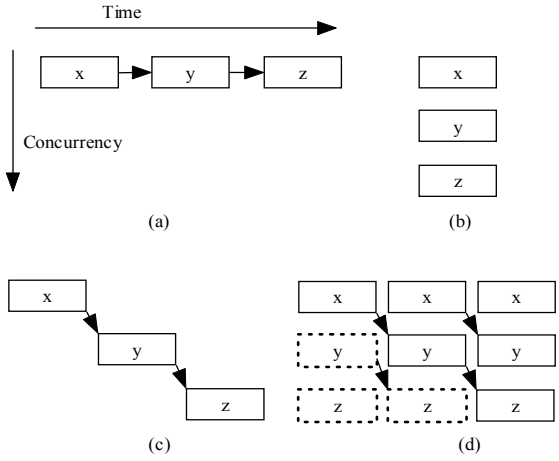

Figure 3: Process representations: (a) Sequential, (b) Parallel, (c) Pipelined, (d) actual pipeline structure

The language should automatically generate structures to handle pipeline priming, stalling and flushing and it should prompt the developer when their design might be using values from a different stage of the pipeline.

3.3 Scheduling and Resource Usage View

To help the designer to avoid resource conflicts such as two parallel processes accessing an external RAM at the same time, a resource usage view should be incorporated. This should work like standard Gantt software packages and identify when resources are used more than once in a time period. It can then suggest changes in the ordering of events. In the case of a multi process design, this would involve either modifying start conditions for processes (to ensure they do not run together) or the use of semaphores to block access to the resource. For a time-critical design such as stream processing from a video camera, the blocking approach is not desirable as it can cause data to be lost, such as when writing from a pixel stream to a frame buffer. Fortunately, blanking periods can often be used to allow changes in the scheduling of competing processes. This view can also help in the scheduling of processes which run only at specific times, such as when a new frame is received, or for identifying where caching of pixels would be more appropriate than memory access, such as when a RAM access occurs when another process is using it and no rescheduling is possible.

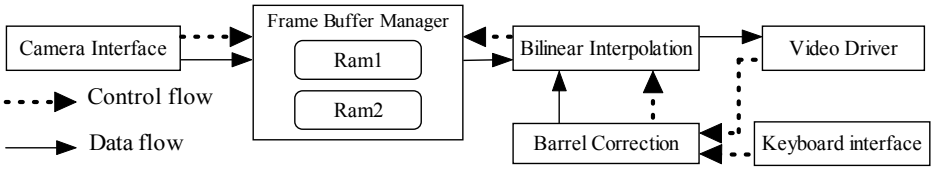

Figure 2 : Architectural view of a Barrel distortion correction system showing: components, control and data flows

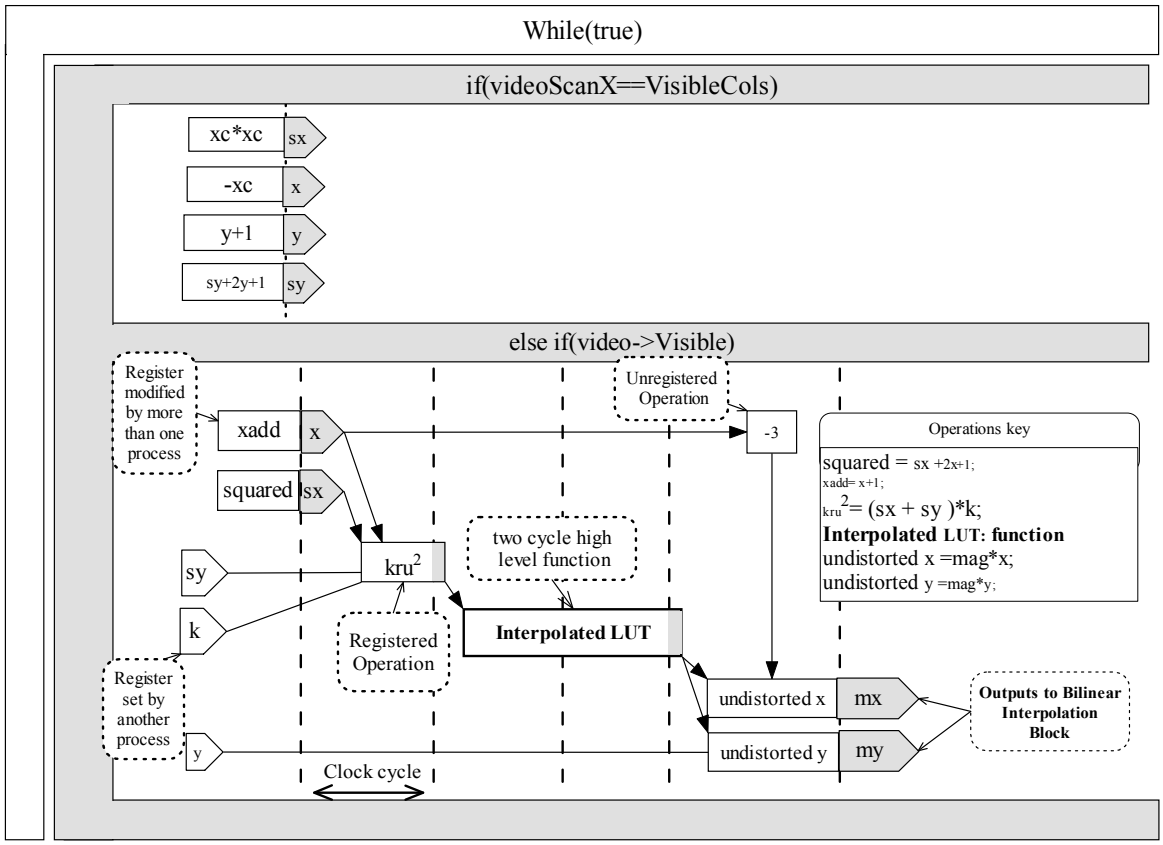

Figure 4 : Low level view of Barrel distortion block showing control functions, timing and operation representation

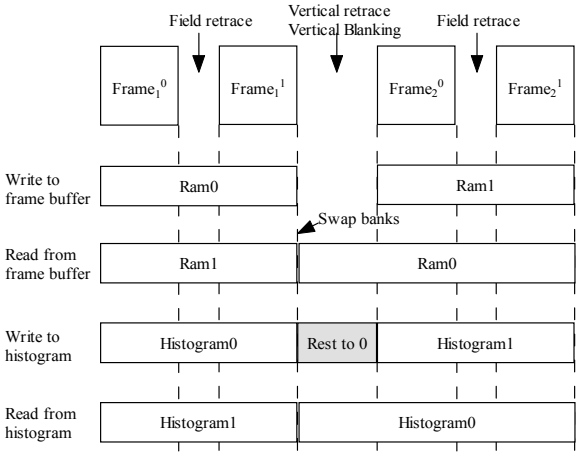

Figure 4: Timing of processes and resources used for a streamed histogram function

This type of resource conflict can occur when a histogram is being constructed and displayed, for example. It is desirable to construct the histogram while the video stream is buffered into one RAM. At the same time, in a different clock domain, both the last full image and its histogram are being displayed. Keeping one of these processes from trying to write to one RAM while the other is reading can be accomplished with a simple condition test. The problem occurs due to the need to reset the histogram values in each bin before the histogram construction algorithm is run, as shown below in Figure 5. This requires a more complex passing of control of resources from process to process, which can lead to error.

When a number of conditions have to be meet before a process can execute, the nested logic can become difficult to interpret. This is especially true with expressions which become true when several counter or external events become true. It can be easier for the designer to have a flag which is set when the conditions are meet. Such flags may be used to give the conditions for the control structures used in Figure 4.

3.4 Simulation and Implementation

A design system should have a simulator for debugging the algorithm before it is implemented in hardware. This is a significant challenge in itself as a simulator's design has conflicting constraints; it must accurately simulate the behaviour of the hardware but it also needs to run with in a reasonable period of time. At this stage little work has been done on the simulation requirements for VERTIPH but it will become an important consideration as the language's functionality and structure are finalised.

The implementation of a design will be done by converting the visual language to a HDL. Handel-C would be the preferred language for this conversion. However an option for conversion to VHDL might be

implemented for developers with out access to the Celoxica Design Suite.

4 Summary

In this paper, we have presented a number of the difficulties in designing image processing algorithms for hardware. We have then evaluated a range of the languages available for designing algorithms for hardware, commenting on their benefits and limitations in terms of image processing implementations and how they aid the developer. We then presented a preliminary design for VERTIPH, a language designed to simplify the switch from a serial design paradigm to a concurrent hardware model by providing a developer with a design environment with multiple views for expressing and visualising an algorithm. Instead of hiding all concurrency from the developer, this allows for the design and implementation of novel algorithms that would be difficult or impractical to implement in conventional languages. VERTIPH is also flexible enough to allow the development of FPGA hardware which is a direct representation of a standard algorithm when the timing constraints are limited to offline processing. VERTIPH encourages developers to design in a way appropriate for the FPGA architecture, with the aim of breaking the architecture-algorithm duality imposed by the standard von Neumann computer model.

5 Acknowledgements

The Celoxica University Programme for generously providing the DK2 Design Suite.

References

[1] R. J. Offen, "VLSI Image Processing," 1 ed. London: Collins, 1985, pp. 326.

[2] IEEE, "IEEE Standard Verilog Hardware Description Language," IEEE,2004, http://www.verilog.com/IEEEVerilog.html, (August 2004).

[3] Accellera, "EDA Industry Working Groups - VHDL," Accellera,2004, http://www.vhdl.org/, (August 2004).

[4] J. Bhasker, *A VHDL Primer*, Third ed. New Jersey: Prentice-Hall, 1999.

[5] S. Swan, "An Introduction to System Level Modeling in SystemC 2.0," Open SystemC Initiative,2001, www.systemc.org, (August 2004).

[6] Celoxica, "Handel-C Software-Compiled System Design," Celoxica,2004, http://www.celoxica.com/methodology/handelc.asp, (August 2004).

[7] H. Schildt, *Programmer's Refeence C/C++*. Berkeley: Osborne McGraw-Hill, 1997.

[8] M. Ganguly, "SystemC Overview," Synopsys Inc.,2001, www.systemc.org, (August 2004).

[9] I. Alston and B. Madahar, "From C to netlists: hardware engineering for software engineers?," *Electronics & Communication Engineering Journal*, vol. 14, pp. 165-173, 2002.

[10] R. Rinker, J. Hammes, W. A. Najjar, W. Bohm, and B. Draper, "Compiling image processing applications to reconfigurable hardware," *Proceedings. IEEE International Conference on Application-Specific Systems, Architectures, and Processors*,pp. 56-65, 2000.

[11] J. Hammes, B. Rinker, W. Bohm, W. Najjar, B. Draper, and R. Beveridge, "Cameron: high level language compilation for reconfigurable systems," *Proceedings.International Conference on Parallel Architectures and Compilation Techniques*,pp. 236-244, 1999.

[12] M. Haldar, A. Nayak, N. Shenoy, A. Choudhary, and P. Banerjee, "FPGA hardware synthesis from MATLAB," *Fourteenth International Conference on VLSI Design*,pp. 299-304, 2001.

[13] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "A system for synthesizing optimized FPGA hardware from Matlab(R)," *International Conference on Computer Aided Design, ICCAD*,pp. 314-319, 2001.

[14] P. Banerjee, "An overview of a compiler for mapping MATLAB programs onto FPGAs," *Proceedings of the ASP-DAC, Asia and South Pacific Design Automation Conference*,pp. 477-482, 2003.

[15] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "Automated synthesis of pipelined designs on FPGAs for signal and image processing applications described in MATLAB(R)," *Proceedings of the ASP-DAC, Asia and South Pacific Design Automation Conference*,pp. 645-648, 2001.

[16] C. S. Williams and J. R. Rasure, "A visual language for image processing," *Proceedings of the IEEE Workshop on Visual Languages*,pp. 86-91, 1990.

[17] P. M. Ngan, "*The Development of a Visual Language for Image Processing Applications*,"PhD Thesis in Computer Science, Massey University, Palmerston North, 1992.

[18] K.T. Gribbon, C.T. Johnston, and D. G. Bailey, "A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation," *Image and Vision Computing New Zealand,IVCNZ*, Massey University, Palmerston North, New Zealand,pp. 408-413, 2003.

[19] J. R. Schermerhorn, *Management*, Sixth ed. New York: John Wiley & Sons, 2001.

[20] Nassi I and Shneiderman B, "Flowchart techniquies for structured programming," *ACM SIGPLAN Notices*, vol. 8, pp. 12-26, 1973.