Rapid prototyping & modelling of real-time image convolvers

D. J. Gibson, M. K. Teal,
D. Ait-Boudaoud & M. Winchester*
School of Design, Engineering & Computing, Department of Electronics,
Bournemouth University.
e-mail: gibsond@bournemouth.ac.uk

Abstract
The foremost objective of designers is developing hardware with the correct functionality in the shortest possible time. Therefore, they are keen to exploit any methodology that will help to reduce development times, increase reuse and make 'right first time' design an achievable target. This paper details such a method that allows rapid prototyping of custom hardware for the implementation of real-time image convolvers. The technique relies on the partitioning of a convolver template into standard components. A library containing VHDL component models and various architectural implementations has been constructed. Using this and an integrated design cycle, convolver models can be quickly and easily constructed. As a standard has been established for the partitioning of the convolver and its components, design times are significantly reduced. Since all the library component models have been previously tested, time consuming development problems will be reduced. This novel philosophy is presented in this paper and conclusions are made upon its use.

Keywords rapid prototyping, system design, image processing, VHDL

1. Introduction

The implementation of real-time image processing systems is generally performed on very high speed processing units. However, before the image information is in a form that is acceptable for processing it will often require some form of low-level manipulation. These low-level or pixel-based operations tend to be realised in custom hardware as the algorithms are executed on all the pixels within an image, and as such are computationally intensive. One of the most widely used techniques for pixel-based manipulation is two dimensional image convolution. This operation may be used to produce a wide variety of different results depending on the filter coefficients chosen.
Although an image convolver performs the same basic operation for each filter implementation, the design of its hardware is nontrivial. This is due to the wide range of filter masks that can be realised. For example, the first mask shown in Figure 1-1 can be used to implement a low-pass filter on an image convolver, whereas the other mask will give a high-pass operation [1]. Although the mask change is a minor algorithmic difference the impact on the convolver's architecture will be substantial for two reasons. First, the low-pass filter will not require any multipliers as all the filter coefficients have a value of one. However, the high-pass coefficient values imply that a multiplication stage is required, even if this is a minor 'power of two' implementation. Secondly, the high-pass mask possesses both positive and negative coefficients, meaning the convolver's hardware will require a two's complement architecture. Therefore, although the convolution is a simple 'multiply & accumulate' operation the hardware realisations vary greatly.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \textbf{Low-pass filter mask} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \textbf{High-pass filter mask}$$

Figure 1-1 Two examples of 3×3 image convolution masks

* British Aerospace Dynamic Division, Stevenage, Hertfordshire.

The use of Hardware Descriptive Languages (HDL's) has revolutionised design techniques for digital hardware. Over the last decade VHDL has become one of the most popular standards for digital design entry. It allows hardware to be modelled at various levels of abstraction and these models can be simulated and synthesised to a particular implementation technology. Although VHDL has been used for the design entry of pixel-based systems it has not been used for algorithmic development or the modelling of generic classes [2]. This paper will describe how a class of image convolvers may be rapidly designed and developed using a parameterised library of component models and a standard convolver configuration.

1.1 Traditional system design techniques

Although convolution is a simple 'multiply & accumulate' operation, the design of a hardware realisation is not easy. In the past, these systems have tended to be designed from scratch using a top-down design flow [3], [4]. A typical design flow and testing philosophy for image processing systems is shown in Figure 1-2.

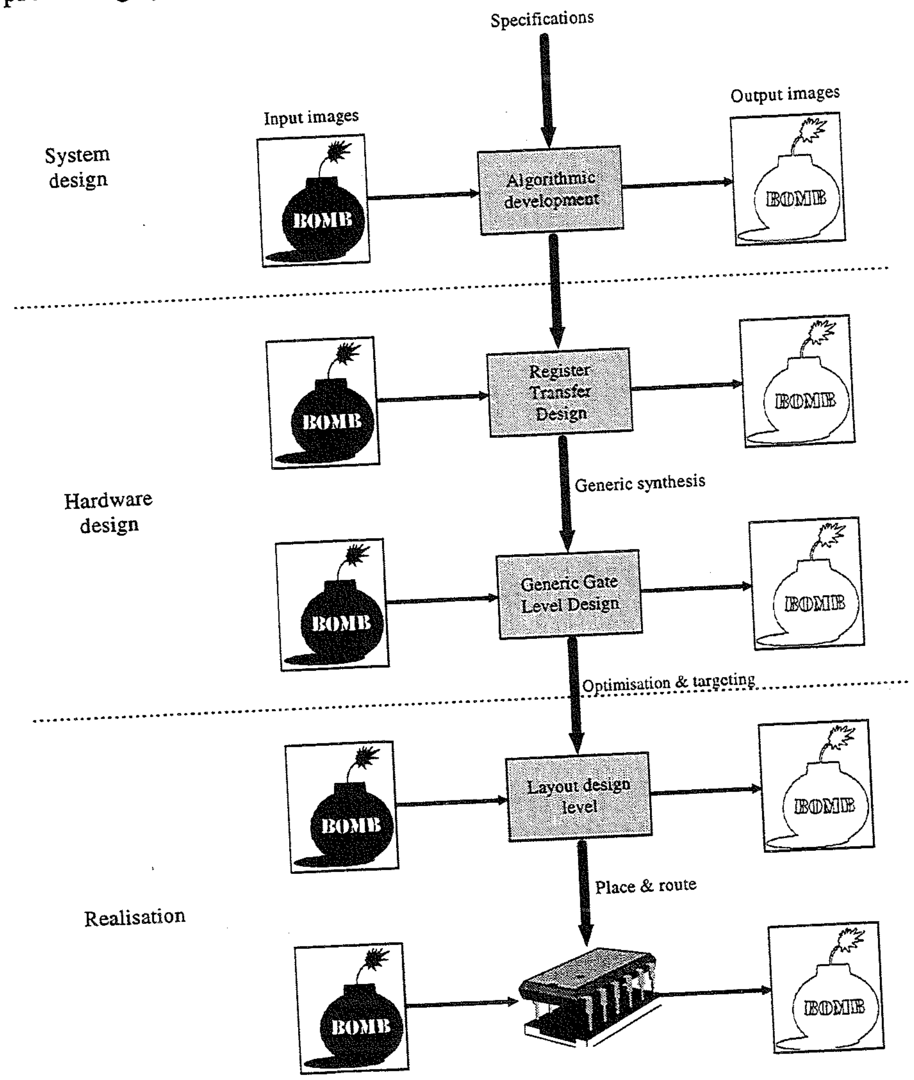

Figure 1-2 Schematic diagram representing a typical design flow and testing philosophy for image processing systems

From the initial specifications the system developers will design an algorithmic realisation of the system. This is generally performed using high-level programming languages that are very versatile and ideal for computationally intensive tasks, thus virtually any form of image processing algorithms can be implemented. This algorithmic design is then verified using test vectors and input image sequences. The results obtained at this stage will become benchmark results that will be used to determine if the subsequent development has the correct functionality.

Following the algorithmic development the design is passed to the hardware engineers, this change over phase is fraught with danger. For a start the hardware development is performed using Hardware Descriptive Languages (HDL's), therefore the hardware must be designed from scratch so that it has the same functionality as the algorithmic implementation. This can be difficult as many image processing algorithms can not be realised easily in digital hardware. Close liaison between the system designers and the hardware engineers will be required to ensure that the realisation of the chosen algorithms can be achieved in hardware. The design entry for the hardware will be at Register Transfer (RT) level. The functionality of this design stage is verified by using the same test vectors that were used during algorithmic development and comparing the results.

The RT design is then translated into a gate-level realisation in terms of generic gates. The translation or synthesis stage is a well automated process and can be performed using many different tools. As with the previous stages the gate-level performance can be verified by using the same test vectors. After the functionality of the hardware model has been verified it can be targeted to a specific technology. First optimisation and targeting tools are used to convert the design's models from the generic gate-level to a technology dependent layout level. Next the place and route tools are used to map the design to the hardware technology. Finally, the functionality of this stage is confirmed against the previously used test vectors.

Although this design strategy is fine for either small systems or ones containing a low number of functional blocks, it does have a number of faults. First, the translation between the system and hardware phases is prone to the introduction of errors. This is because translation is performed manually between two completely different modelling mediums, thus errors can be inadvertently introduced. Also the system designers may not be fully aware of the constraints that digital hardware has upon their chosen algorithms. Secondly, when the hardware designers begin their task they will first decompose the systems algorithms into functional blocks. These blocks will then be created as separate modules, this means that the overall functionality of the RT system model cannot be verified until all the modules have been created at this level. This process generally leads to errors that hamper the debugging of the system model. Finally, this strategy despite the use of HDL's does not encourage reuse of designs and tends to rely greatly on the experience and expertise of the engineers.

Our aim is to target these problems and create a design philosophy that will allow the rapid prototyping of image processing systems. This paper documents how such a philosophy has been applied to the design of image convolvers.

2. Rapid prototyping design philosophy

The design cycle adopted for our research is a library-based scheme that is represented schematically in Figure 2-1. The entire design cycle is based around a VHDL component library. This library contains models of basic arithmetic components such as adders and multipliers, as well as components that are specifically required for image processing systems. These include edge counters, thresholds and pipelines. All the components contained within the library possess a standard interface that must be maintained during the entire design cycle. With this pre-designed and simulated library of components the system engineers can develop the required algorithms from the specifications. This is achieved by creating a 'virtual prototype' of the system purely from the components contained in the library [5]. As far as the system designers are concerned each of the components can be considered as a functional 'black box'. In this way the system engineers can design and verify the relevant algorithms.

The design is then passed to the hardware developers, this changeover will be a much smoother process as the modelling medium that both sets of engineers are using is the same. In addition, the required algorithms and hence the design has already been partitioned into functional blocks. The hardware engineers have to take each component and specify any architectural constraints [5]. For example, the system engineer may have specified an 8-bit addition, the hardware engineer will need to determine the architectural implementation, i.e. ripple carry, carry look-ahead, carry save etc.

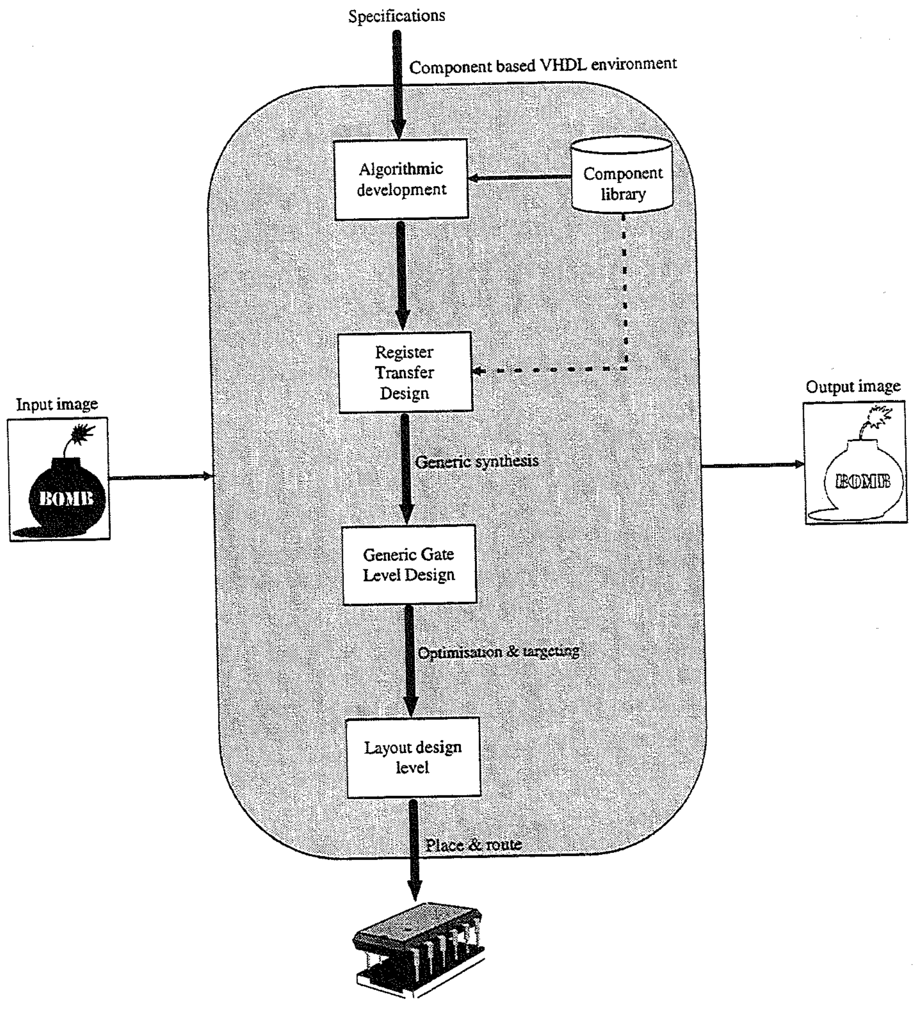

Figure 2-1 Schematic diagram representing the library based VHDL design philosophy

As each component model is modified to account for its hardware realisation, its performance can be verified by integrating the modified component back into the system model and re-simulating. This integrated testing strategy is based upon work undertaken at the University of Dortmund and allows components to be tested immediately [6]. This will identify errors earlier in the design cycle as models at any level can be simulated after modification within the previous system model. For example, if the architecture of an adder component is synthesised to a generic gate-level it can be immediately re-integrated into the RT system model and its performance verified.

2.1 The component model library

Before this rapid prototyping methodology can be adopted it is necessary to create a library containing the relevant component models. The components that will be required can be ascertained by decomposing the algorithm into functional blocks. For each block a component will require creation with a standard interface. As the component model's internal structure may be subsequently modified with hardware details the interface must be in terms of hardware. For example, Figure 2-2 contains the interface requirements for an 8-bit adder. As can be seen the interface is defined in terms of the IEEE standard multivalued logic system for VHDL model interoperability (std_logic_1164) [7]. Ports *A*, *B* and *Result* have been declared as type *std_logic_vector* and *Clk*, *Reset* and *Cout* as *std_logic*. This means that although the system engineers are only concerned with the development of the algorithms they must use this style of logic interface between the components.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY adder IS
    GENERIC ( bus_width : INTEGER := 8;
              granularity : INTEGER := 2);
    PORT ( A : IN std_logic_vector ((bus_width - 1) DOWNTO 0);
           B : IN std_logic_vector ((bus_width - 1) DOWNTO 0);
           Clk, Reset : IN std_logic;
           Cout : OUT std_logic;
           Result : OUT std_logic_vector ((bus_width - 1) DOWNTO 0));
END adder;
```

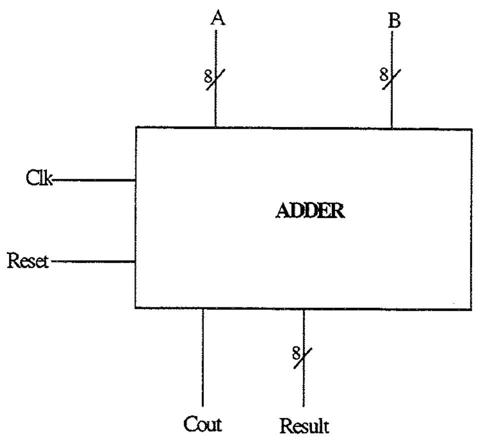

Figure 2-2 Example of component interface for an adder

All the components contained in the library have been made fully generic in size. This allows one model to be used for the realisation of any component width. This can be seen in Figure 2-2, where the width of the adder is controlled by the generic *bus_width*. Due to the high throughput nature of real-time applications it is difficult to predict the level of pipelining that a component will require. Moreover, the degree of pipelining required is partly dependent on the implementation technology chosen for the system. At the modelling stage these details may not be known. Thus, the 'granularity' or pipelining of each component is also controlled with generic constructs [8]. In addition, any algorithmic parameters that can be varied in the components will also be controlled with generic constructs [9].

The abstraction level that the VHDL component models are created is dependent on the functionality of the component. For example, a storage device is easily constructed at a RT level, whereas an adder may be first designed at a functional level and then have implementation details added later during the design cycle. This is shown more clearly in Figure 2-3, where as an example an adder at different stages of the design cycle is shown. As can be seen the adder starts off as a purely functional model and evolves into a full hardware model. At the RT level a carry look-ahead architecture is added to the model, this can be achieved in one of two ways. Either incorporate a pre-designed ripple carry architecture from the library or modify the adder's existing architecture. Following the model's synthesis it is determined that the adder requires a *granularity* of four. Finally, during the layout it is discovered that the routing has incurred larger delays than expected and a *granularity* of two is required.

At any time during the design cycle the adder model should have the same functionality. As the system engineers purely regard the component models as 'black boxes' they can take any model during the projects design cycle and verify its functionality. This is simply achieved by incorporating the component back into the overall system model.

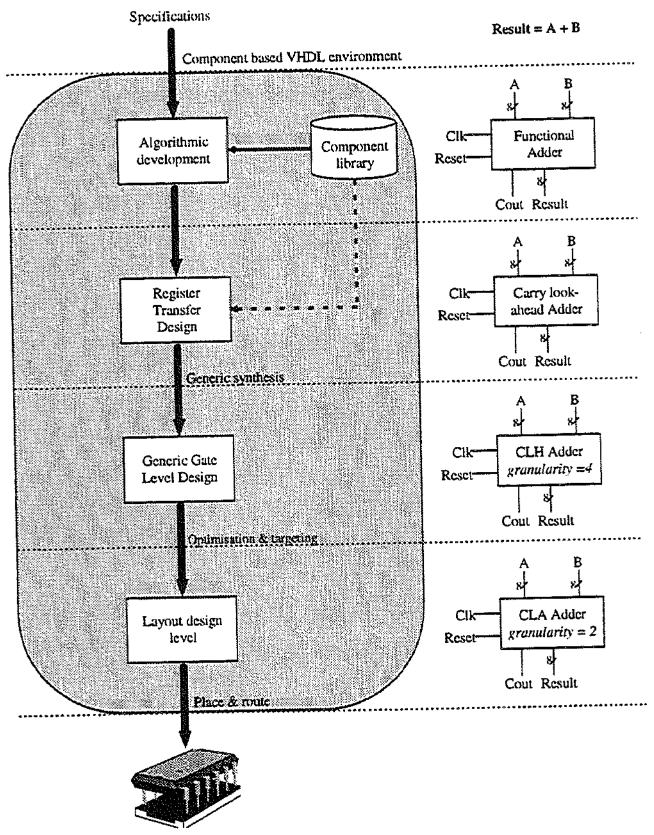

Figure 2-3 An example of an adder model throughout the rapid prototyping design cycle

3. Rapid prototyping of image convolvers

Real-time image convolvers tend to be difficult to design and implement due to high throughput and computational rates. In addition, a minor change at the algorithmic level can translate into major implementation differences. For these reasons the design times for such systems tend to be long. Hence, engineers are keen to adopt techniques that will speed up their 'time to market'. The design philosophy outlined in Section 2 has been applied to the development of such systems and will be outlined in this section.

3.1 Component library and template for image convolvers

Before image convolvers can be modelled it is first necessary to establish what components are required. This is achieved by decomposing the algorithms into functional blocks. The *centred, zero boundary superposition* convolution algorithm that has been applied to our rapid prototyping philosophy is shown below in Equation 3:1 [1].

$$H(t_1,t_2) * I(m_1,m_2) = Q(j_1,j_2) = \frac{1}{N}\sum_{n1=0}^{L-1}\sum_{n2=0}^{L-1} T(n_1,n_2).I(j_1 - n_1 - L_c, j_2 - n_2 - L_c)$$

Equation 3:1

This equation can be decomposed into six components. A schematic diagram representing these components and their interaction is shown in Figure 3-1. The first component, the pipeline is simply a storage device. This is required as the image pixels are supplied in a raster scan form, the pipeline stores the pixels so that at any time the current pixels required for the convolution operation are available. The coefficient multiplier component performs multiplication between the current pixel values and the corresponding filter coefficients. The results from the multiplication's are then summed. Next the scaling division normalises the current value. Finally, the last two components are included for the synchronisation of the system. The counter keeps track of the current position and synchronises the convolver with any external devices, the edge zero component sets the value of any edge pixel outside the convolution range to zero.

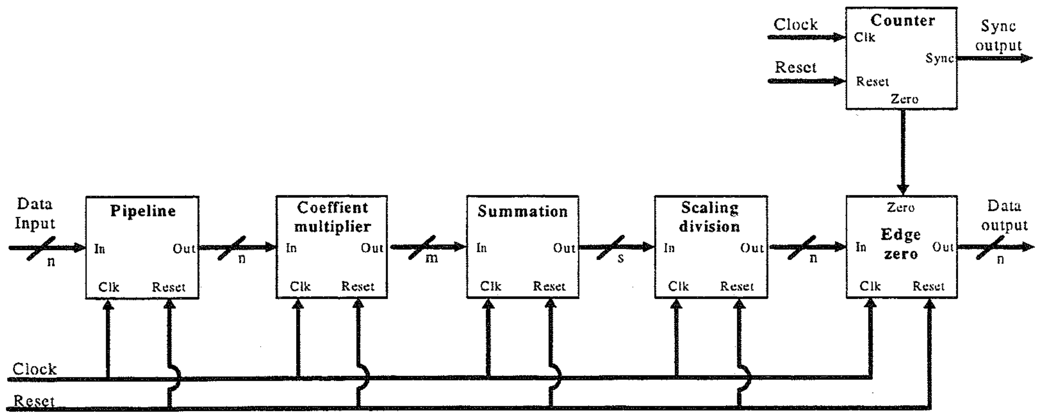

Figure 3-1 Schematic diagram representing the components required to implement an image convolver

This configuration now becomes the standard template for the construction of convolvers from the component library. Therefore, the interface for each component must be kept the same throughout the design. VHDL models for each of the components have been constructed and tested. Each model has been made fully generic in terms of size, pipelining and the actual filter operations that the convolver can perform. Hence, the component library can be used to model single convolution systems. In addition, extra architectures have been included in the library for arithmetic operations. These include: ripple carry addition, carry look-ahead addition and Wallace tree multiplication.
All the library's component models and the convolver template have been fully tested, this allows development of systems without having to worry about the configuration of the convolver.

3.2 Rapid design and simulation of image convolvers

With the component library and the convolver template in place it is possible to design and simulate image convolvers very quickly. From the specifications of the system the generic parameters for each component can be established and entered into the convolver template. This is the algorithmic development phase of the design philosophy as depicted in Figure 2-1. Functional testing at this level can be performed directly upon the convolver with real input image sequences. This is possible as each component and the system template have been previously proven to function correctly. Hence, simulation at this level is only to establish the performance of the chosen algorithms against real input images.
With a satisfactory derivative of the algorithm chosen, hardware implementation details can be modified or added to the component models dependent on the application and realisation technology. For example, an FPGA implementation of the convolver will require modification to the arithmetic components and an external realisation of the pipeline. As each component model is modified its functionality can be immediately verified by re-simulating the convolver. In this way the architectural implementation of each component can be verified in turn and tested immediately.
Following this design philosophy through to the hardware implementation speeds up the design cycle and can lower the number of errors introduced into the system. This will reduce the 'time to market' for such products and gives greater flexibility during the design to make algorithmic alterations.

4. Conclusions on rapid prototyping philosophy

With the components contained in the library and the convolver template established a number of different convolution systems have been designed and modelled. These include Laplacian, Box filter, Gaussian, Roberts and Sobel.

The initial construction of an algorithmic model for such systems is dependent on the constraints set by the customer's specifications. Therefore, it is difficult to compare results obtained from a contrived example with those expected from a real project. However, it was found that using the convolver template and the component library lead to the chosen algorithms being modelled in a matter of hours. This is much quicker than trying to verify the algorithms using a high-level programming language as this will have to be constructed from scratch. Moreover, although the system designers consider the component models as purely functional 'black boxes', each is actually an abstraction of a hardware module. Therefore, without a conscious effort the systems engineers will be considering the high-level design in terms of hardware. In addition, the hand-over of the design between the system developers to the hardware engineers will be much smoother as both design teams will be using the same modelling medium. This promotes closer liaison and communication between the teams making it easier to iron out errors and problems.

When the required components and templates generic configuration has been established the hardware designers can concentrate on fitting specific architectures to each component. This architectural targeting is dependent on the chosen algorithms and the application of the system. However, using the component based design environment the design flow is greatly eased as it allows the individual testing of new component architectures with previously verified system models, thus, highlighting architectural problems earlier in the design and therefore speeding up the development time.

The overall affect of this integrated design philosophy is that it forces the early partitioning and establishment of the components, thus promoting the re-use of designs. This combined with the integrated testing of the components highlight mistakes earlier and reduce design times. All of these factors will help to reduce the 'time to market' for products designed using this strategy.

[1] Pratt, W. K., *Digital Image Processing, Second edition.* John Wiley & Sons, 1991.

[2] Schumacher. G., W. Nobel, W. Putzke & M. Wilmis 'Applying Object-Oriented Technologies to Hardware Modelling-A Case Study.' *Proceedings SIG-VHDL Working Conference*, Spring 1996.

[3] Valle, M., G. Nateri, D. Caviglia & L. Briozzo, 'An ASIC Design for Real-Time Image Processing in Industrial Applications.' *Proceedings of the European Design & Test Conference*, 1995.

[4] Shewring, I. W., M. A. Wahab 'An Integrated Approach to the Design and Implementation of Image Filters.' *Proceedings IEE 15th SARAGA Colloquium on Digital and Analogue Filters and Filtering Systems,* 1995.

[5] Vahey, M., D. Bushman, S. DaBell, T. Ennis & P. Kalutkiewicz, 'A Virtual Prototype VHDL Development Methodology.' *Proceeding of the VIUF Conference, VHDL: Champions of the Second Generation.* Spring 1995.

[6] Schwoerer, L., M. Lück & H. Schröder, 'Integration of VHDL into a System Design Enviroment.' *Proceedings of the European Design Automation Conference,* Sep 1995.

[7] IEEE Std 1164-1993, *IEEE Standard Multivalued Logic System for VHDL Model Interoperability (Std_logic_1164).*

[8] Dadda, L., & V. Piuri, 'Pipelined Adders.' *IEEE Transactions on Computers,* Vol. 45, No. 3, 1996.

[9] Joyce, J., & J. Van Tassel, 'Fully Generic Descriptions of Hardware in VHDL.' *Computer Hardware Description Languages and their Applications,* Elsevier Science Publishers, 1991.