A Modelling Methodology that Promotes the Rapid Prototyping of Real-Time Image Processing Systems

D. J. Gibson, M. K. Teal,
D. Ait-Boudaoud & M. Winchester*
School of Design, Engineering & Computing, Department of Electronics,
Bournemouth University,
Fern Barrow, Poole, Dorset. UK
e-mail: gibsond@bournemouth.ac.uk
*British Aerospace Dynamic Division, Stevenage, Hertfordshire. UK

Abstract

The foremost objective of system designers is developing hardware with the correct functionality in the shortest possible time. Therefore, they are keen to exploit any methodology that will help reduce development times, increase reuse and make 'right first time' design an achievable target. This paper proposes a method that promotes the rapid prototyping of custom hardware for the implementation of real-time image processing systems. The systems are partitioned into component blocks and then prototyped using a library of generic models. Each component has a standard I/O interface and has been simulated and synthesised to a technology independent level. Systems can be quickly and easily constructed using this library and a model-oriented flow. The technique permits the functionality and performance of the proposed system to be quickly assessed and verified throughout its design cycle. Following simulation the hardware realisation can be considered by synthesising to a technology independent level. This will produce a generic architecture for the system that may then be targeted to a specific technology. Such an integrated approach reduces design times and smoothes the development process. This novel philosophy is demonstrated through the development of an image convolver implementing two low-pass filters.

1. Introduction

Although the principle of rapidly prototyping digital signal processing (DSP) systems using libraries of standard components has been established for a number of years [1], this practice has never been fully exploited for image processing systems. A number of reasons are attributed to this. Firstly, standard structures for many DSP functions are well established, for example, FIR filters. These structures can be easily translated into hardware realisations. Secondly, image processing uses many of the same principles as DSP, but due to the two-dimensional nature of image data the realisation will be far more complex. Thirdly, the real-time requirement and computational rates needed for a hardware implementation of these systems is not easily achieved. In recent years a new trend has been the use of reconfigurable architectures for prototyping systems [2], [3]. These permit the implementation of image processing models on a real-time platform. When the models have been verified they can be retargeted to the desired technology. The major disadvantage of these techniques is that the hardware models have been targeted to the prototyping technology, generally FPGA's. Therefore, the remapping process is extremely difficult and prone to errors. All of these reasons have lead to little work being carried out on the standardisation of image processing systems and their development. Any work that has been undertaken has been specific to its realisation and has tended to rely on DSP development tools [4]. This paper demonstrates an integrated methodology based upon a model-oriented flow technique with a generic library of high-level component models. The high-level design is performed using pre-defined components therefore allowing early characterisation of the hardware.

2. Rapid prototyping design methodology

The design flow adopted during our research has been a library-based scheme that is represented schematically in Figure 2-1.

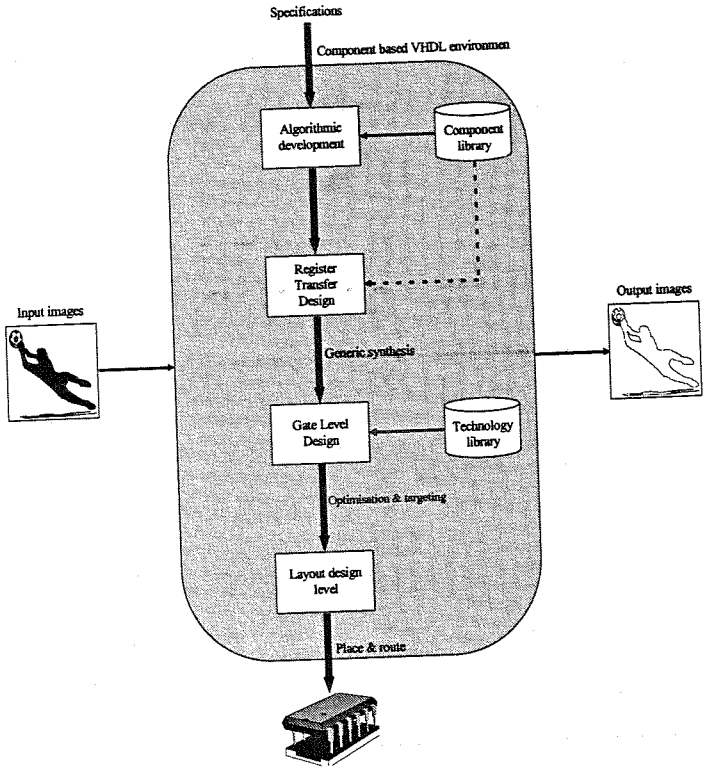

Figure 2-1 Schematic diagram representing the library based VHDL design philosophy

The high and intermediate levels of design are based around a pre-defined 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. This library of components allows the system engineers to develop the required algorithms from their specifications. This is achieved by creating an abstract '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.

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

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 allows components to be tested immediately [6]. For example, if the architecture of an adder is synthesised to a generic gate-level its functionality can be immediately verified by re-integrating the model into the RT system model.

The aim of this methodology is to produce earlier characterisation of the problem whilst still maintaining a high degree of flexibility. Thus, the development sphere should be able to accept design entry at any level of abstraction and cater for either 'top-down' or 'bottom-up' design flows. The use of a pre-designed library of components permits the rapid creation of the virtual prototype. This can then be used for exploration of the problem domain, speeding up the characterisation of the design. The achievable reduction in 'time to market' resulting from the use of VHDL has been identified and publicised by the RASSP program [7]. It has shown that VHDL can reduce design time by a half when designing these systems. The use of VHDL in the typical design cycle of a signal processing system can be seen as a scarf joint between modelling mediums. This is sown as a solid line in Figure 2-2, which represents the use of modelling mediums in the design of digital hardware.

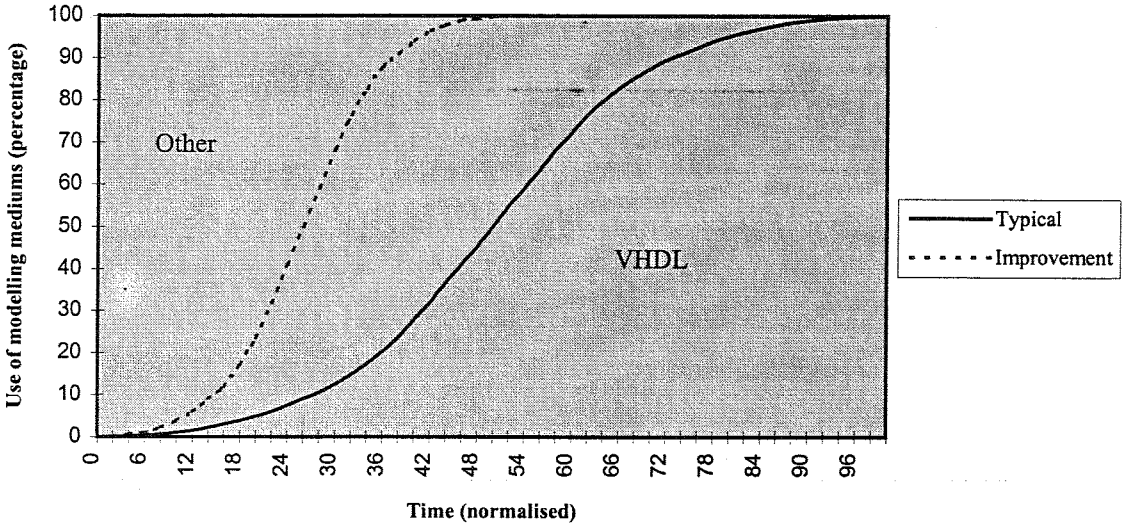

Figure 2-2 Graphical representation of the modelling mediums used in the design of digital hardware

As can be seen at the projects conception VHDL is not used, however throughout the design cycle its use increases in the manner represented until at the completion VHDL is the only medium used. The adoption of this rapid prototyping methodology in conjunction with a component library increases the gradient of this scarf joint. This will result in quicker design times that will reduce the 'time to market' of signal processing hardware. This intended improvement is shown as a dashed scarf joint in Figure 2-2.

3. Two dimensional convolution

One of the most widely used techniques for the low-level manipulation of images is two dimensional convolution. This is a moving neighbourhood, filtering operation that is used to enhance desired features within images. Image convolution may be expressed mathematically as Equation 3-1.

$$H(t1,t2) * I(m1,m2) = Q(j1,j2) = \sum_{n1=0}^{L-1} \sum_{n2=0}^{L-1} H(n1,n2).I(j1-n1,j2-n2)$$

Equation 3-1

Where: $T(t_1,t_2)$ is a filter template of size $(L \times L)$, $I(m_1,m_2)$ is the input image of $(N \times N)$ and $Q(j_1,j_2)$ is the output image with size $(M \times M)$. Note, this representation assumes that the input

and output images and the template are square. However, the formula may be easily modified if this is not the case.

The filter template is an array of coefficients. This template is moved over the image in steps of one pixel, at each new position an image window is created the same size as the template. Each filter coefficient is then multiplied with the grey scale value of the corresponding image pixel. The results of these multiplications are then summed to give a single value. This value is taken to be the grey scale of the corresponding pixel within the new image. As this operation is repeated over the whole image a resulting image is created. The result from the convolution operation is only valid when the filter mask is fully on the input image. That is, if the mask overhangs the image the result is not valid. Therefore, the resulting image from the convolution is smaller than the original. However, for the designers of image processing systems it is desirable to have input and output images of the same size. One of the most popular methods for eliminating this problem is called *centred, zero boundary superposition* [8]. This technique calculates the convolution when the filter mask is fully over the image and any other pixels are zeroed. Thus, the size of the images remains constant. This is shown as a diagram in Figure 3-1 and is expressed mathematically in Equation 3-2.

$$H(t1,t2) * I(m1,m2) = Q(j1,j2) = \sum_{n1=0}^{L-1} \sum_{n2=0}^{L-1} H(n1,n2).I(j1-n1-Lc, j2-n2-Lc)$$

Equation 3-2

Where:

$$Lc = \frac{L-1}{2}$$

Equation 3-3

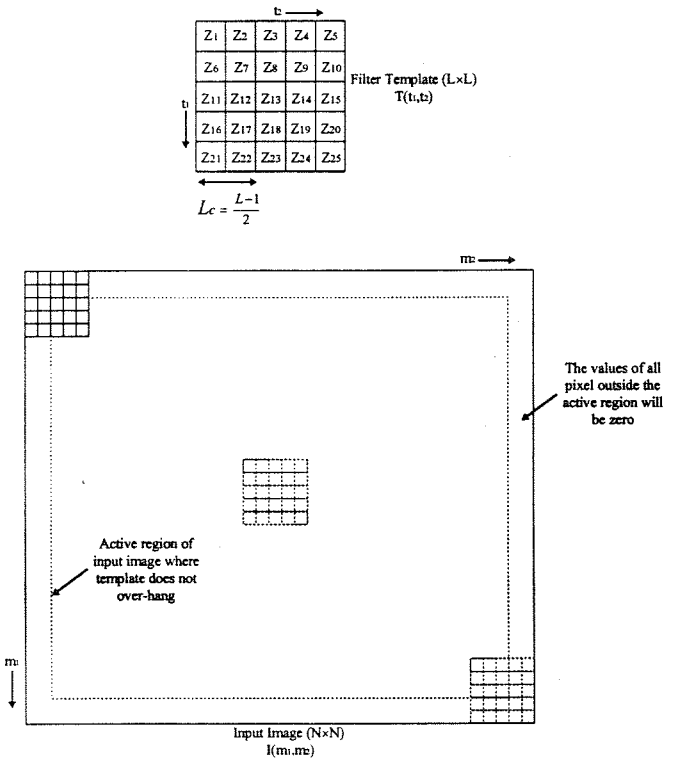

Figure 3-1 Schematic showing the relationship between the input image and the filter mask

The characteristics of the resulting image is dependent on the template's coefficient values. The values of these coefficients is derived from the impulse response of the filter, this is often referred to as the filter's operator. Thus, convolution can produce a wide variety different outputs depending on the operator used. Many filter operators are already well defined and can be used to produce some of the most fundamental image processing operations. These include edge detection, high and low pass filtering. For example, a simple low pass filter may have the filter operator $H(t_1,t_2)$ as shown Equation 3-4.

$$H(t_1,t_2) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Equation 3-4

Note that the filter operator normally consists of two parts, an array of coefficients and a normalisation factor. These two separate parts may then be substituted into the *centred, zero boundary superposition* formula, Equation 3-2. This will produce a general form as shown in Equation 3-5.

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

Equation 3-5

Where $T(n1,n2)$ is the template of filter coefficients and $\frac{1}{N}$ is the normalisation factor.

4. Generic model for image convolution

Using the general form of the *centred, zero boundary superposition* formula (Equation 3-5) derived in Section 3 it is possible to construct a standard model template of the required hardware structure. Before the component models can be created it is necessary to establish which algorithmic parameters should be quantified using generics. The easiest to spot are the image size and mask size. Both of these can be changed and doing so will drastically affect the systems hardware. Not so easy to spot is the range of grey scales within the image, this determines the data width of the arithmetic hardware. Finally, the range of the operators' coefficients will also affect the hardware architecture. Thus, a model of a two dimensional image convolver can now be constructed in terms of these parameters. Therefore, this model may be used for any square image size, any odd sized square mask (that is, 3, 5, 7, 9, etc.), any power of grey scale range (that is, 1, 2, 4, 8, 16, 32, etc.) and any filter operator that is made up of positive, power of two numbers. With the generic parameters established it is now possible to decompose the algorithm into its constituent parts. A block diagram of the convolvers' components is shown in Figure 4-1.

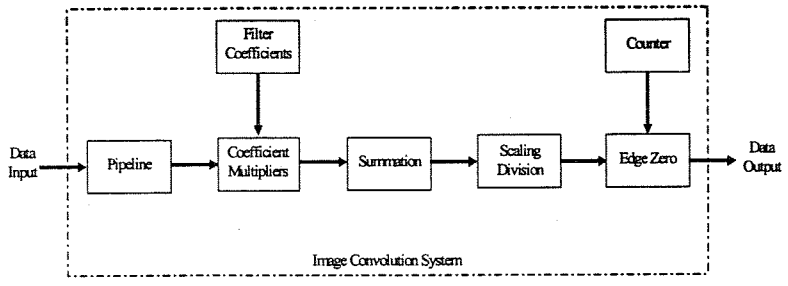

Figure 4-1 Block diagram of the top-level image convolver model

5. Modelling image convolvers

Each of the component models constructed have been given an asynchronous reset and a clock input. Therefore, a top-level model of the image convolver can constructed using the component models created and the hardware template. A block diagram of the top-level image convolver model is shown in Figure 5-1. This model of the image convolver has been built by instantiating all the required components. Each of the components was given a generic map that supplied the individual entities with the current algorithmic values for the convolution system.

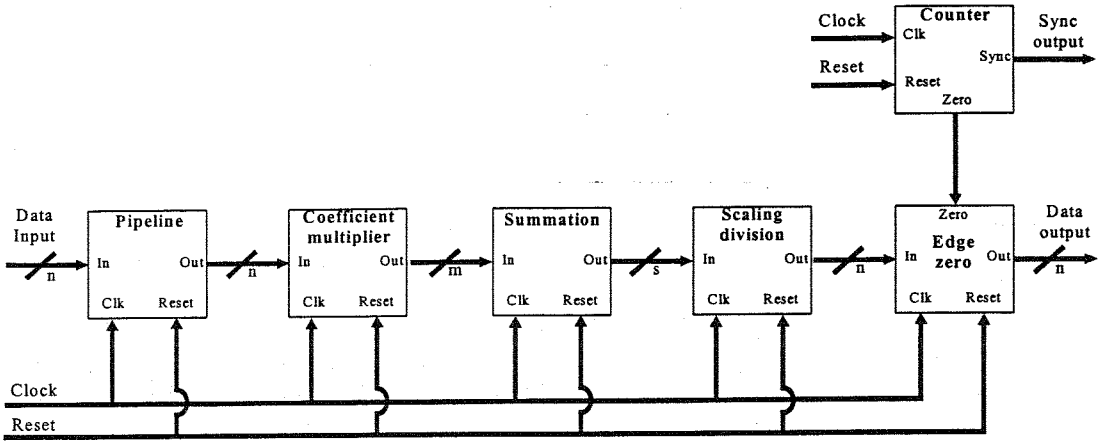

Figure 5-1 Block diagram of the top-level image convolver model

Using the convolver model it now possible to perform functional simulation for the desired parameters. For example, the convolver's performance and hardware structure were accessed with the following parameters:

Image size = 128×128 pixels
Mask size = 3×3
Grey scales = 256 levels
Coefficient range = 1 to 4
Scaling division = 16

$$\text{Convolution mask} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The results from the functional simulation of the convolver with these parameters are shown in Figure 5-2.

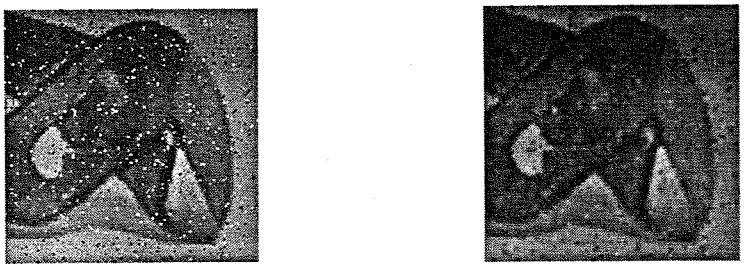

Figure 5-2 Simulation results obtained from a model of a convolution low pass filter

Using exactly the same model it is possible to verify the functionality of a different algorithmic implementation of a low pass filter. This is demonstrated below as the convolver model has had its generic parameters modified so that the performance of following system could be established.

Image size = 512×512 pixels
Mask size = 3×3
Grey scales = 256 levels
Coefficient range = 0 to 1
Scaling division = 8

$$\text{Convolution mask} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The results from the functional simulation of this convolver with these new parameters are shown in Figure 5-3.

Figure 5-3 Simulation results obtained from the second low pass model

These changes were implemented quickly and easily by simply modifying the relevant generic values in the top-level model and changing the mask constant in the filter function. Once these changes had been made the convolver was re-simulated. The simulation of the model allows the convolvers performance to be compared with the pervious algorithm, for a similar input image. Thus, it is instantly possible to assess if the algorithmic changes have had the desired effect on the convolver's performance.

6. Conclusions

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 convolvers top-level template. Functional testing at this level can be performed directly upon the convolver model with real input image sequences. Simulation at this level is only to establish the performance of the chosen algorithms against real images as all the components have been previously tested. With a satisfactory variation of the algorithm chosen, hardware

implementation details can be modified or added to the component models. For example, an FPGA implementation of the convolver will require modification to the arithmetic components and an external realisation of the pipeline. This architectural targeting is dependent on the chosen algorithms and the target application of the system. However, using the component based design environment the design flow is greatly eased. 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.

Although the high-level designer considers the component models purely as functional 'black boxes', each is actually an abstraction of a hardware module. Therefore, without a conscious effort the systems engineer will be considering the high-level design in terms of hardware. In addition, the hand-over of the design between system developers and 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. Moreover, this methodology will encourage early characterisation of the design and allow rapid exploration of the problem domain. This in turn will speed up the integration of VHDL into the design flow and therefore reduce the 'time to market' of products developed in this way.

References

[1] Salvela, V., P. Järvinen, A. Nummela, J. Keskinen & J. Nurmi, 'Utilization of a VHDL-based ASIC-Realizable Filter Architecture Library in DSP System Design.' *Proceedings of the International User's Forum, VIUF-Fall'94: Component Modelling,* 1994.

[2] Athanas, P., & A. Lynn Abbott, 'Real-Time Image Processing on a Custom Computing Platform.' *IEEE Computer Journal,* Vol. 28, No. 2, February 1995.

[3] Quenot, G., C. Coutelle, J. Serot & B. Zavidovique, 'Implementing Image Processing Applications an a Real-Time Architecture.' *Proceedings of the Computer Architectures for Machine Perception,* 1993.

[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] Hein, C., 'Exploiting VHDL design in RASSP.' *Proceeding of the VIUF Conference*

[8] Pratt, W., *Digital Image Processing.* New York: John Wiley & Sons, 1991.