Malaysian Journal of Computer Science, Vol. 13 No. 1, June 2000, pp. 65-75

A VHDL MODULE GENERATOR FOR FAST PROTOTYPING OF MULTIMEDIA ASICS

Mohamed Khalil Hani and Kah Hoe Koay
MiCE Department, Faculty of Electrical Engineering
Universiti Teknologi Malaysia
81310 UTM Skudai, Malaysia
Tel.: 607-5505003
Fax: 607-5566272
email: khalil@suria.fke.utm.my
koaykh@tm.net.my

ABSTRACT

This paper presents an electronic design automation (EDA) software for generating synthesizable VHDL modules for hardware applications of multimedia data processing. The tool provides a rapid-prototyping design environment by enabling dynamic storage and retrieval of reusable modules, parameterized design entry, hierarchical design exploration, list-based component interface insertion, and block diagram view of designs. This produces a design environment, which enables fast design development cycle by improving design productivity to meet the fast evolving standards of multimedia formats. A test case design of a JPEG decoder has been built using the tool. Implementation of the design in FPGA has proven the capability of the tool in handling large and complex designs.

Keywords: ***VHDL, module generator, design entry, EDA, CAD, rapid-prototyping, parameterized, JPEG decoder***

1.0 INTRODUCTION

Hardware designers are facing increasingly challenging tasks in developing growing complex systems within shrinking time-to-market windows for shorter product life cycles [1]. In a design process, the design entry stage is time consuming, cumbersome and error prone, which represents a significantly large portion of the overall design cycle. A gap remains between the output of a system-level design tool and the input to the top-down ASIC (Application-Specific Integrated Circuit) design process [1]. The output of the system-level tool is a verified algorithm, and the input to the ASIC design process is a complete structural-level VHDL (VHSIC Hardware Description Language [2]) description of an architecture implementing the algorithm. Design engineers often need to evaluate and choose among different architectural alternatives (for example, parallelism, pipelining, resource sharing and numeric formats) before obtaining the best solution that resolves speed, size and power consumption tradeoffs. Current EDA tools lack this high-level modeling facility for users to evaluate design alternatives in a fast and efficient way.

Another shortcoming in many EDA tools is that they do not support dynamic creation of reusable components, though most provide a component module library in fixed, template or wizard form to avoid designing from scratch. Current EDA software also uses VHDL as text-based design entry representation, which is especially suitable for handling complex digital systems. However, unlike schematic design entry that facilitates pictorial hierarchical design, text-based representation is harder for users to view the whole picture of a design and hence make it more difficult to manipulate the design hierarchy.

2.0 VHDL MODULE GENERATOR

A VHDL Module Generator is proposed to provide a solution to the problems stated. It is a general-purpose software design entry tool that provides a rapid-prototyping design environment. The prototyping framework meets the following design objectives:

1. Improve support of hierarchical and modular design methodologies.
2. Improve support of reuse and incremental development by later-bindings through parameterisation and dynamic module library.
3. Preserve capabilities for automatic synthesis.

2.1 Overview

Fig. 1 shows the conceptual diagram of the software. The main project window is the design hierarchy explorer. As VHDL design entry lacks the convenience for users to explore design hierarchy, the tree view design hierarchy explorer is provided to facilitate hierarchical and modular design methodology. Users can create top-down, bottom-up or mixed design conveniently using the design hierarchy explorer. Each object in the tree view is a design module, which is always associated with a VHDL file. Each object is associated with a parameter list, a port interface list, and a block diagram. The software allows parameterised VHDL entry where users can insert and modify data widths, data types as well as architectural options easily without changing the VHDL code manually but through the parameter editor window. The parameterisable feature enables creation of dynamic module library, where users can create user-defined library modules for future reuse.

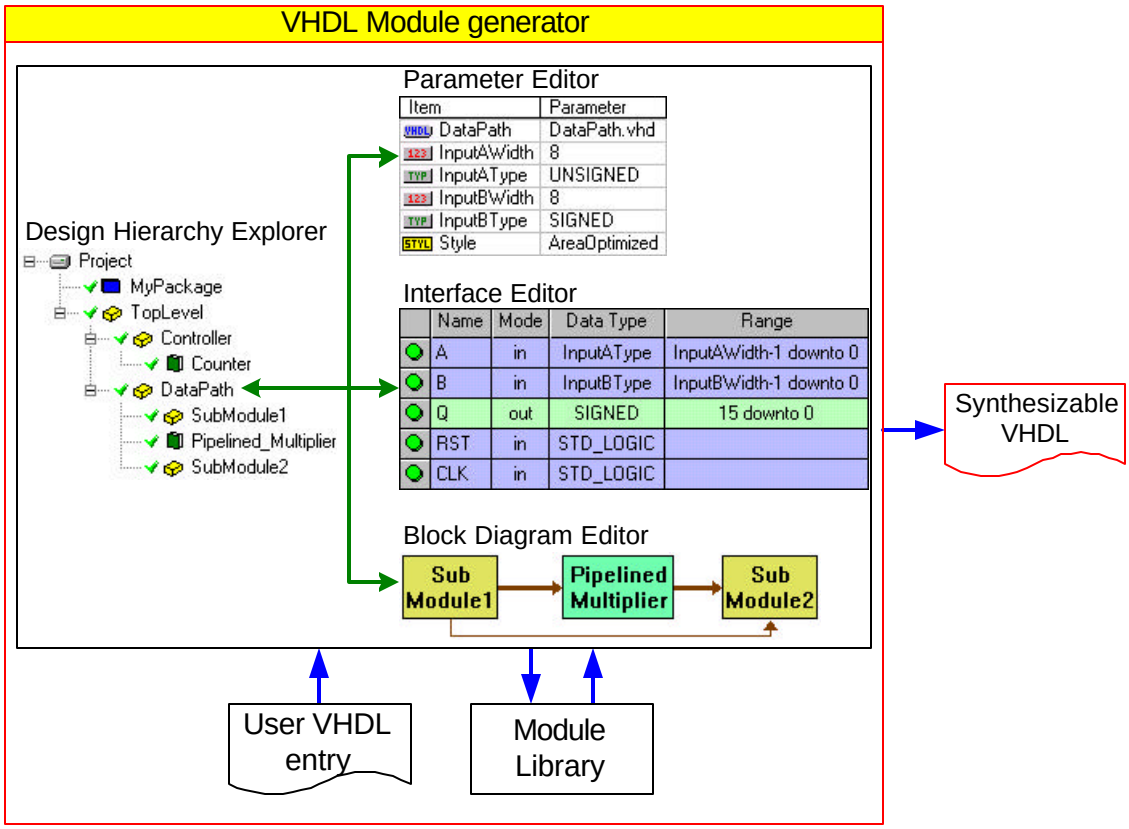

Fig. 1: VHDL Module Generator

The interface editor window enables insertion, modification, and viewing of port interface of VHDL entities. This feature eliminates VHDL text coding of port list. Block diagram editor window provides a more intuitive and realistic pictorial view of designs. The output of the software is synthesisable VHDL code. The software will compile the VHDL source codes to make them synthesisable.

The VHDL Module Generator was developed using Microsoft Visual C++ version 5.0. The software is designed to run in Microsoft Windows 95/98/NT/2000 platforms in x86 PC (Personal Computer) systems. The synthesisable VHDL code output from the VHDL Module Generator is synthesised using FPGA Express [3] from Synopsys Inc. The software is integrated in Xilinx Foundation Series 1.5i implementation tool from Xilinx Inc.

A reconfigurable computing platform is employed to build hardware prototypes, where an FPGA (Field Programmable Gate Array) is mounted on an ISA (Industry Standard Architecture) card (APS-X208 from APS, Inc.) in a PC slot. The FPGA used is XC4052XLA from Xilinx, which has typical logic gate count of 52k. Programming and inter-facing the FPGA with the computer is made seamlessly easy using the platform. This FPGA-based reconfigurable computing platform provides facility to download prototype designs to FPGA, then design verifications can be done in real-time through the established interface between the FPGA and CPU. It eliminates conventional slow verification process, which is done in software simulation.

The following sub-sections describe some of the major features of the VHDL Module Generator, which include parameterised design, dynamic library, hierarchical design exploration, automatic VHDL editing and compilation, interface port editor, and block diagram editor.

2.2 Parameterised Design Entry

Parameterised design entry is one of the main features of the VHDL Module Generator. A number of researchers [1, 4, 5] suggested this. Modules or components can be described in a generalised form and made specific by using certain parameters. Take the case of a simple example like counter. A generalised counter can be made and specified with parameters like the counter width, its initial value, stop value, increment step, and architecture options, such as binary or one-hot encoding.

By using generic clauses, configuration statements, enumerations, etc., generalised components can be described in VHDL code. Furthermore, multiple levels of parameter passing can be done as well. The VHDL Module Generator takes advantage of this feature and further enhances it to windows user interface. The software searches through the VHDL code for all parameterisable

elements and extracts them to a parameter editor, where users can view and edit the parameters in the software without opening the VHDL file. Thus, it makes design work easier and faster for hardware designers. Fig. 2 shows an example of parameter extraction process. On the other way round, users can insert new parameters to a module through the software's user interface, then the software will generate the corresponding VHDL code automatically.

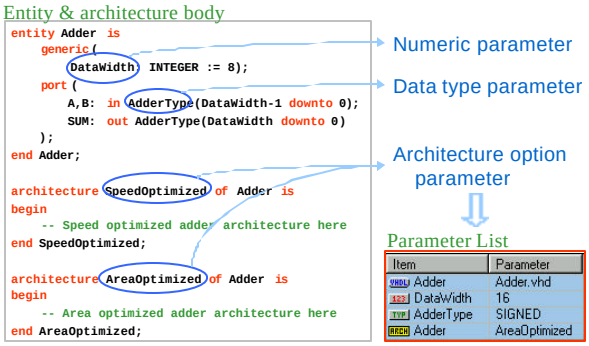

Fig. 2: Parameter extraction

Mathur [6] argued that parameterisation makes the VHDL code more complex and harder to understand. It also takes longer time during the synthesis process comparing to fixed designs. However, considering that a design usually involves repeated modifications, (for example, changing a bus width could involve modifications in several different VHDL entities), designers would have to restudy the codes to find where the changes would have to be made, which could take an even longer time. If designers have created parameterised designs from the beginning, future modification work is relatively easy. In other words, parameterised design entry enables rapid prototyping of designs.

Among supported parameters are numeric parameter, data type parameter, architecture option parameters, and logic parameter. Users can insert parameters either through manual VHDL coding or through the software's user interface. Fig. 3 shows a practical example of parameter list for a data register. The following paragraphs discuss each type of parameter in detail.

Item	Parameter
DataRegister	DataRegister.vhd
Author	Koay Kah Hoe
Description	A set of flip-flops
Documentation	DataRegister.pdf
DataType	SIGNED
DataWidth	8
CLK_EDGE	'1'
RST_ACTIVE	'1'
ChipEnable	DontUse
DataRegister	Asynchronous_Reset

Fig. 3: Parameter list for a data register

Numeric parameter describes data width, number range, or any other numeric values used in VHDL code. Numeric parameters are stated in generic clause at entity declaration part of a component. Only integer numbers are supported. In Fig. 2, a numeric parameter called DataWidth with a default value is stated in the code. Users can modify the DataWidth value in the parameter editor. The user-defined value will be inserted in the component declaration part at its upper design hierarchy. The VHDL synthesiser in use [3] supports this generic mapping.

Data type parameter can be used to specify signal types like STD_LOGIC_VECTOR, UNSIGNED, SIGNED, or others. This parameter is needed as there might be different designs using different kind of data types, but the basic components used could be independent of data type and usable in different designs. The software extracts data type parameter from three places in VHDL code: generic clause similar to numeric parameter, port list at entity declaration, and signal declaration at architecture declarative part, as shown in the following listing. For parameterised data type Type2 and Type3 in the example below, the software checks whether they are common standard data types, if not, the software will treat them as parameterised data types.

```
entity Adder is
    generic(
        Type1: type := SIGNED
    );
    port(
        A, B: in Type2(7 downto 0);
        SUM: out Type2(7 downto 0)
    );
end Adder;

architecture Adder_arch of Adder is
    signal C: Type3(7 downto 0);
begin
  :
  :
```

Architecture option parameter provides flexibility for users to choose different kinds of architectures of a component. This permits designers to evaluate different kinds of architectures quickly. In VHDL syntax, an entity body could have several architecture bodies associated with it. Configuration declaration or configuration specification is used to specify which particular architecture to choose. In Fig. 2, SpeedOptimised and AreaOptimised are two architecture options for the entity Adder. They will be extracted as architecture option parameter. The architecture selected by users will be stated as configuration specification at architecture declarative part of the upper design hierarchy. However, VHDL synthesiser used [3] does not recognise configuration specification. To compensate this limitation, during compilation, only selected architecture option is added to the compiled VHDL file. Other architecture options are discarded.

Style parameter is very similar to architecture option parameter. Only one architecture option parameter is allowed in one module, but this is not practical enough as parameterised designs often needs several independent architecture option parameters. The style parameter solves this limitation. As in Fig. 3, the architecture option parameter has been used to select the style of reset signal. The style parameter is used to select the style of chip enable signal. The number of style parameter in a module is unlimited. The style parameter is inserted after port declaration in the entity body. An example is shown in VHDL listing below. Enumeration type is used to define style options. This is followed by a constant declaration of the enumerated type. The constant name is the style parameter name. The constant value denotes the selected style parameter.

```
entity DataRegister is
   generic (
      ...
   );
   port (
      ...
   );
   type ENUM_ChipEnable is
      (HIGH, LOW, DontUse);
   constant ChipEnable: ENUM_ChipEnable
      := DontUse;
end DataRegister;
```

The style parameter can be used in two ways:

- Inside a process statement by using if statement:

```
if ChipEnable=DontUse then
      ...
      ...
end if;
```

- Generate statement:

```
G1: if ChipEnable=DontUse generate
   ...
   ...
end generate;
```

Logic parameter has two possible values only, either '0' or '1'. This kind of parameter is useful in specifying active logic level or edge of control signals, like active clock edge, reset active level, enable active level, etc. In the example code below, a logic parameter called ResetActive with a default value is stated in the generic clause. STD_LOGIC is used as the keyword. The software will extract it as logic parameter. The generic clause with STD_LOGIC type is not supported by VHDL synthesiser used [3]. The software replaces the parameter with the selected logic directly into the compiled VHDL code.

```
entity FlipFlop is
   generic(
      ResetActive: STD_LOGIC := '1'
   );
   port(
        D, RST, CLK: in STD_LOGIC;
        Q: out STD_LOGIC
    );
end FlipFlop;
```

Note parameter is used to write a one line comment or note in the parameter list. An example usage is shown in the second and third parameters in Fig. 3. In VHDL code, the note parameter is expressed in generic clause using the keyword "string" as shown below. The parameter is enclosed with double quotes. When compiling the design, the note parameter is removed.

```
generic (
 Author: string := "Koay Kah Hoe";
 Description: string := "A set of flip-flops";
   ...
);
```

Link parameter is used to provide linkage to other files that are associated with the design. It could be any registered file type. In Fig. 3, the fourth item is link parameter. When users double click on this parameter, the software will invoke default viewer or application for the file depending on its file type. This parameter is useful in associating documentation file to a design. In VHDL code, the link parameter is expressed in generic clause using the keyword "file" as shown below. The parameter is enclosed with double quotes. When compiling the design, the link parameter is removed.

```
generic (
    Documentation: file := "DataRegister.pdf";
    ...
);
```

2.3 Dynamic Module Library

From the fact that most systems share a number of basic components, apart from providing a basic design environment, many EDA tools often come with a set of predefined component libraries [1, 4, 6]. This is very useful for designers to avoid designing from scratch. There are several forms of component libraries used in EDA tools, i.e. fixed, template, or wizard form. Fixed library components have least flexibility among these three methods, as users need to modify the codes to suit the design. Template-based components provide generic VHDL codes. Users can easily modify components by changing available parameters. It has slower synthesis run time and is prone to create redundant hardware. dQdt [1] uses template-based library. Wizard-based library creates VHDL codes after users have specified the parameters, example of this is HDLGen [6]. The resulted VHDL code is clean, because specified parameters have been inserted directly into the VHDL code. However, created code is fixed. If modification is needed, users have to rerun the wizard.

Most EDA tools do not support dynamic library creation of reusable components, only a fixed set of library modules is provided. Like the HDLGen [6], the module generators are compiled objects, which cannot be modified by the user. In practical cases, hardware designers often need to reuse their previous designs in other places. If the design entry tool allows dynamic storage of library modules, future designs could be speeded up by just retrieving previously designed

modules from library. dQdt [1] implemented an expanding library of application-specific and parameterised VHDL models, with model complexity ranging from single arithmetic operators to large core functions to entire VLSI chips.

In the VHDL Module Generator, dynamic module library is empowered by parameterised design entry, as it provides a method to create generalised modules that could be stored as library modules. Thus, any design that is reusable can be stored in module library dynamically.

Actually, modules in library are just a collection of VHDL files. All parameter information is recorded in the VHDL file itself. When a user stores a design into library, the software will store all parameters currently defined in the VHDL library file. If a design consists of several sub-components in several files, when it is stored in library, the software will combine them into one file in ascending order. The next time when user gets the module from library, only one object is shown, the sub-components in lower hierarchy are hidden from users. Expanding the original design hierarchy inside the library components is not allowed.

The module library could be categorised as template-based library, but with some differences from conventional template-based library. Conventional template-based library modules are used directly in the VHDL synthesiser, whereas in this software, the design VHDL source code is different from the output (synthesisable) VHDL code. The software will remove parameterised elements from the code and replace them with user-defined parameters during compilation process. Only numeric and style parameters stay parameterised with some insignificant synthesis run time tradeoffs.

2.4 Hierarchical Design Exploration

Visualisation of the design hierarchy is implicitly provided with schematic design entry. In contrast, VHDL-based design entry lacks a convenient way for designers to visualise the system structure, as all descriptions are written in text. Some synthesis tools can create a hierarchical view of design entities after elaborating a design, including FPGA Express [3] and Active-HDL from Aldec Inc. However, the view is shown only after designers have completed their designs; even that, only when the design is elaborated successfully. The VHDL source files are not inserted in hierarchy order as the design, but just in a flat file list. Clearly, it is not an intuitive way of organising design sources.

The VHDL module generator overcomes the stated weaknesses. In the software, the VHDL source files are organised and displayed in the actual design hierarchy through a window called design hierarchy explorer. Users can view and manipulate the design hierarchy right from the beginning of the design process using the design hierarchy explorer.

Fig. 4 shows an example view of the design hierarchy explorer. The first icon on the top of the view shows the subfolder name (Test Project) where all the source files reside. Except for the first icon, each icon in the view, which is called module, represents a VHDL source file. The module with the name ThisPackage in the example contains a VHDL package. The filename always has the same name with the package name, entity name, or component name. TopLevel, Controller and DataPath in the example are VHDL entities with architecture bodies. Only one entity body is allowed inside each module of this type.

Fig. 4: Example design hierarchy view

Counter and Pipelined_Multiplier in the example are library modules. Inside each library module, there could be more than one-entity bodies. The module name is the top-level entity name of the library module. The internal entities of library modules are hidden from users. RAM16X1 and BUFGP are predefined modules. Predefined module can either be a primitive component predefined in the synthesis tool or an external component that is defined using other representation, like schematic or netlist.

Among available functions of the design hierarchy explorer includes add new, insert module, cut, copy, paste, move, rename, and remove. By using these functions, users can create top-down, bottom-up or mixed design methodologies conveniently. All these functions are done in the graphical user interface of the design hierarchy explorer. The functions make design work much easier to manage.

2.5 VHDL Editing Automations

The VHDL Module Generator also provides automatic editing of VHDL codes. When adding a new object using the design hierarchy explorer, the software automatically creates the basic entity body and architecture body for users. When a new module is added under another module, the software will automatically create a component declaration and an instantiation example at the architecture body of the upper hierarchy module. When some interface ports of a component are changed, the corresponding component declaration must be changed, too. This is a tedious work if it is done manually. If a component is

removed using the design hierarchy explorer, its component declaration is automatically removed by the software. When renaming an object, the software automatically changes its entity name, component declaration, and all occurrences of the old name to the new name.

By having these automations, designers do not have to do all these tedious works manually, and thus, can focus on the actual design itself. The designers need only to make interconnections among components and create actual VHDL coding.

2.6 Interface Port Editing

The VHDL Module Generator provides a user-friendly environment to create interface ports of modules. Fig. 5 shows an example of the interface port editor window. The editor is divided into 4 columns. The "Name" column specifies the port name of a signal. The software checks for correct naming of signal names according to the VHDL identifier naming standard. In Fig. 5, the "_RESET" signal name is not allowed, thus the software displays a red knob at the first column to denote wrong syntax.

	Name	Mode	Data Type	Range
●	D	in	SIGNED	7 downto 0
●	Q	out	SIGNED	7 downto 0
●	IO	buffer	SIGNED	7 downto 0
●	LOAD	in	STD_LOGIC	7 downto 0
●	_RESET	in	STD_LOGIC	
●	CLK	in	STD_LOGIC	
●				

Fig. 5: Interface port editor

The "Mode" column denotes signal's port mode, there are five kinds of modes: in, out, inout, buffer, and linkage. The software creates an in-place list box for users to choose the mode when the column is activated. Also, the software uses colours as well to differentiate the modes. Users can easily identify the mode of signals.

The "Data Type" column denotes the type of signal. It could be standard data types like SIGNED, UNSIGNED, STD_LOGIC, STD_LOGIC_VECTOR, or user defined type. The software creates a combo box (a list box and an edit box) when it is activated. Users can choose standard data types from the list box or key in user defined type as well. The software also checks for correct naming of type according to standard identifier naming.

The "Range" column denotes the range of signal, if applicable, depends on the data type. Data types like SIGNED, UNSIGNED, INTEGER, etc. can be specified with range, whereas other data types like STD_LOGIC and BIT has no range. The software checks for standard data types that allow range or not. For example, the LOAD signal in Fig. 5 should not has range, thus a red knob is shown.

If the syntax is correct for the whole row of port declaration, the software will show a green knob. If the port declaration is incomplete, the software will show an orange knob. Common editing functions including cut, copy, paste, move, insert, and remove are available to aid users in creating or modifying interface ports.

2.7 Block Diagram Exploration

Design hierarchy is best viewed in tree view form, but for design modules under the same hierarchy, tree view cannot give any indication of how are the interconnections among the modules. Instead, block diagram can best describe the interrelations among these modules. The VHDL Module Generator provides a block diagram editor for the purpose. The software creates module blocks automatically which are directly under the currently selected module. The module blocks are coloured according to the type of modules. Users can resize and move the blocks accordingly. Connection lines can then be made by selecting the first block and then point to the second block using mouse. Fig. 6 to Fig. 9 are examples of block diagrams created by the software.

The software can divide the text inside the block intelligently to multiple lines when the block width is not enough to contain the entire text in a single line. The path of connection line between two blocks is created by the software automatically after a user has made the link between two blocks. The software finds the easiest path for the user. The user can still modify the connection path as needed. There are three line thickness options available. It is used to roughly indicate the bus width of the signal in the connection. Arrows at line ends can also be created to indicate the flow of signal.

Users can insert text blocks and custom blocks to build a complete block diagram. The software automatically searches for interface ports' name among the text blocks inserted. If a text block matches an interface port name, the text block is changed to interface port shape corresponding to the port's mode.

The block diagram is also another mean of design hierarchy explorer. By double clicking on a block, block diagram of the module under the clicked block will be opened. To go up a level, simply double click on empty space in the block diagram. By having the block diagram editor, users can create documentation diagrams easily.

2.8 Compilation to Synthesisable VHDL

Compilation process converts the VHDL source files to synthesisable VHDL file that is recognisable by the VHDL synthesizer. Source VHDL files has certain parameterised entries that are not recognised by VHDL synthesiser. During compilation, unrecognised parameterised entries are removed and replaced with the actual parameters. Synthesisable VHDL is created by selecting an object in the

design hierarchy explorer and then invoking the compilation function. The output VHDL file will have the same name with the selected object name. If the selected object has child objects under it, the software will compile all the child objects and combine all objects in one file in hierarchical order. The output VHDL file is saved in a destination folder specified at the parameter list associated with the first object at the design hierarchy explorer. The synthesisable file can then be analysed by the VHDL synthesiser.

3.0 TEST CASE: A JPEG DECODER

To show the capability of the VHDL Module Generator in handling large and complex designs, a baseline gray-scale JPEG (Joint Photographic Experts Group) [7] decoder has been designed exploiting the full functionality of the tool. The test case was accomplished successfully. It contains 99 VHDL entity bodies, or 140k-bytes of VHDL source code, which takes about 85k logic gates when implemented in Xilinx FPGA, or 1793 CLBs.

Due to the enormous processing time required to simulate large digital systems, executing a VHDL model with a representative data set even on a fast workstation is not practical. A reconfigurable computing platform was employed to enable rapid prototyping on it. Reconfiguration from one processing task to another does not require physical changes but is accomplished by downloading a hardware personalisation database to a reconfigurable computing platform within seconds. A designer with this capability has a means for evaluating the performance of experimental algorithm or architecture, and a working component that can be used in the development and testing of a much larger system [8].

The JPEG decoder has been prototyped and verified in a reconfigurable computing platform (APS-X208 from Associated Professional Systems, Inc. with FPGA XC4052XLA-08). The VHDL coding is basically optimised for Xilinx FPGA architecture [9]. Nevertheless, it can be easily retargeted to other implementation platforms.

3.1 Design Architecture

Fig. 6 shows a simplified block diagram of the baseline gray-scale JPEG decoder. Table entries and configuration signals for Huffman table, quantisation table, and DPCM (Differential Pulse Code Modulation) selector have been omitted in the diagram to avoid complexity.

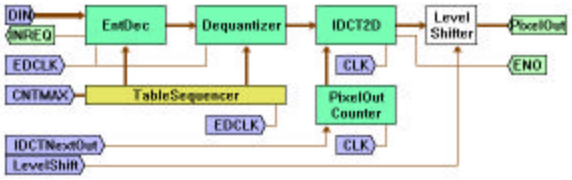

Fig. 6: Block diagram of a JPEG decoder

EntDec in the diagram is entropy decoder, which contains a Huffman decoder and post-entropy decoder. Inside the post-entropy decoder are a DPCM decoder and a run-length decoder for decoding DC coefficient and AC coefficients respectively. Fig. 7 shows the block diagram of the entropy decoder. A parallel architecture is employed in the entropy decoder. The JPEG bitstream input, DIN is 32-bit wide, as the maximum allowable width of a Huffman code plus amplitude length is 32 bits. 4 cycles are needed to fill in the pipeline, after that, the entropy decoder produces coefficient outputs in zigzag order at every clock cycle.

After the entropy decoder, the coefficient outputs are dequantised using the quantisation table at the dequantiser. Fig. 8 shows the block diagram of the dequantiser. Inside the Dequantiser path is just a pipelined multiplier. The latency of the dequantiser is 4. The dequantiser produces dequantised coefficients in zigzag order at every clock cycle.

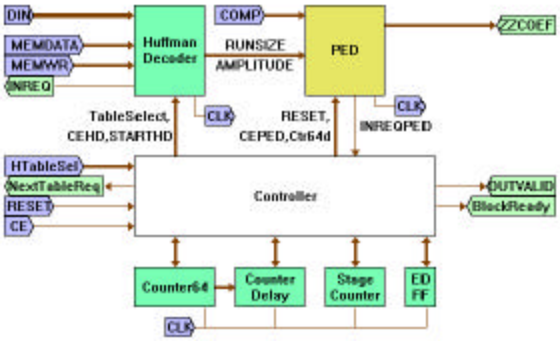

Fig. 7: Entropy decoder

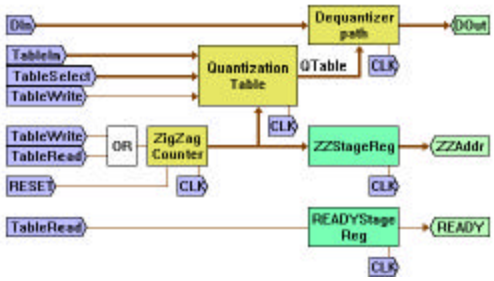

Fig. 8: Dequantiser

In JPEG bitstream, multiple colour components can be stored in interleaved order in a scan. Each component could have its own Huffman table and dequantisation table. Also, each component uses its own DPCM predictor. The table sequencer shown in Fig. 6 is used to provide table IDs and DPCM predictor selector for the Huffman decoder, dequantiser, and DPCM decoder in order while decoding data units. The sequence is depended on the number of components and sampling factors of the components.

Notice that the JPEG decoder has two separate clocks, the entropy decoder, dequantiser, and table sequencer use the same clock (EDCLK), while the IDCT (Inverse Discrete Cosine Transform) uses another clock. The reason is, the Huffman decoder has a long combinational logic which requires a longer clock period. However, the IDCT requires short but many clock cycles to complete its computation. Thus, two clocks are used to avoid compromising the slow clock rate requirement at Huffman decoder and degrading the whole system performance.

After dequantisation, coefficient data are sent to the IDCT. The IDCT converts coefficient data from spatial frequency domain back to original space domain in 8x8 pixel data units. This is the most computationally intensive part of JPEG decoder. The two-dimensional IDCT can be obtained by performing one-dimensional IDCT by rows and then another 1-D IDCT by columns. In this implementation, due to the hardware area limitation, the 1-D IDCT for the first stage and the second stage is sharing the same piece of hardware. The block diagram of the 2-D IDCT is shown in Fig. 9. Data from dequantiser is stored in the input buffer (InBuf). During the first stage, the multiplexer (Mux) selects data from InBuf. The result of the first stage 1-D IDCT is stored in the transposition RAM (TRAM). The TRAM transposes data from row sequence to column sequence. After 8 rows of input data being processed, the Mux will select input from TRAM to perform second stage 1-D IDCT. The computed result is stored in the output buffer (OutBuf). Each 1-D IDCT operation takes 17 cycles. There are a total of 16 1-D IDCT operations, plus some initial and transition states, the whole 2-D IDCT process takes 283 clock cycles.

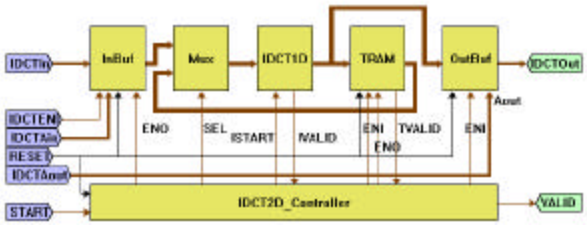

Fig. 9: 2-D IDCT

After the IDCT operation, the data is level shifted from signed values to unsigned pixel data. For gray-scale image, the result is decoded image, which can be displayed on the screen. The decoding capability of full colour image was not implemented in the hardware. This was due to hardware area limitation of the FPGA in used. To decode colour images, there should be additional Huffman tables, quantisation tables, and an additional YCbCr (luminance-blueness-redness) to RGB (red-green-blue) colour space converter module, which are too large to fit in the FPGA. Thus, colour image decoding was disabled.

3.2 Hardware Test Result

The JPEG decoder has been implemented in XC4052XLA-08 on APS-X208 reconfigurable computing platform successfully. It has been tested with a number of JPEG files generated from commercial and non-commercial software, including Microsoft Photo Editor in Microsoft Office, LView Pro, as well as JPEG encoder from IJG (Independent JPEG Group).

A wide variety of quality factors and test patterns of JPEG files has been tested with the JPEG decoder, including photo pictures, line-art graphics, and several extreme cases like all white, all black, chess pattern, black and white texts, etc. All the tests were passed with no visually perceptible errors comparing with software JPEG viewer (Microsoft Photo Editor). The tests were performed by saving reconstructed images into bitmap files for both hardware and software JPEG decoders, then the two versions of bitmap files were compared. Table 1 shows comparisons

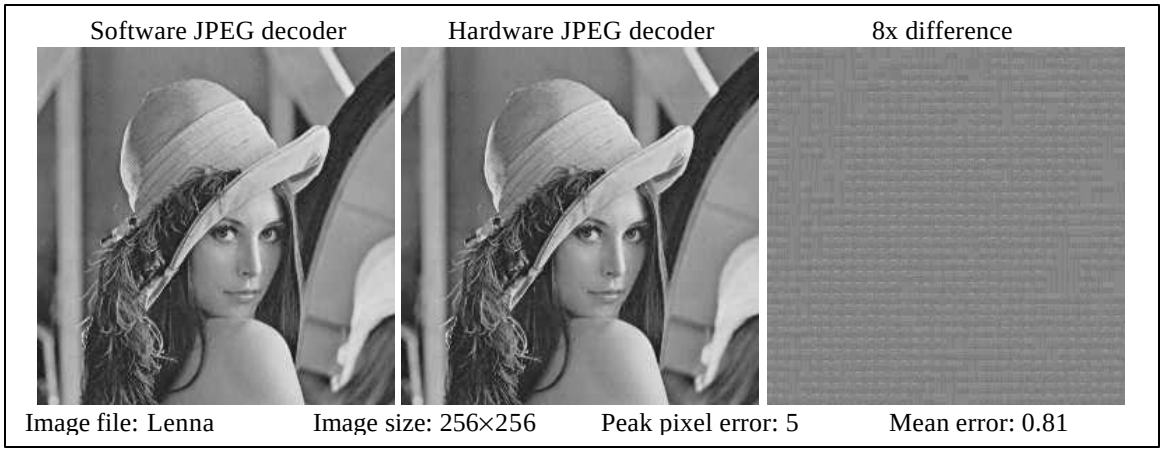

Fig. 10: Comparison of hardware and software JPEG decoders

made on different types of images, while Table 2 shows comparisons made on different quality factors of JPEG files. In the tables, the peak pixel error is the maximum pixel difference between hardware and software versions of reconstructed image. Mean error is the sum of pixel errors divided by the total number of pixels. Fig. 10 shows a test image of Lenna, where the first image is JPEG file reconstructed using software decoder and the second one is hardware-reconstructed image. The third image is the differential image, where errors are enlarged 8 times to be more visible and normalised at neutral gray.

Table 1: Decoded results on different image types

Image type	Peak pixel error	Mean error
Photo picture	6	0.85
Graphic	5	0.84
Line-art	5	0.51
Black & white text	5	0.50

Table 2: Decoded results on different quality factors

Quality factor	Peak pixel error	Mean error
100%	6	0.86
75%	5	0.82
50%	5	0.51
25%	5	0.51

From the performance aspect, about 310~370μs is consumed to decode an 8x8 block depending on the quality factor of the JPEG source image. This is due to the bottleneck at the transmission path between the CPU and the FPGA through ISA interface. The reconfigurable computing platform provides 8-bit data bus only and does not support DMA (Direct Memory Access). For the JPEG decoder itself, if it is run in optimum condition, the actual hardware computation requires 283 clock cycles per 8x8 block. If the decoder runs at 20MHz, the decoding takes 14.15μs per block.

4.0 RESULT

The success in creating a JPEG decoder using the VHDL Module Generator, which has a total of 99 VHDL entities, or 140k-bytes of VHDL source code, or 85k logic gate counts, has proven the tool's capability in handling large and complex designs. Logically, the tool could handle even larger and more complex designs without problem.

From portability aspect, designs creation were based on compatibility with the FPGA Express version 3.2 from Synopsys [3]. Porting designs to a VHDL simulator, the Active-HDL version 3.5 from Aldec, has been tested with no problem except additional libraries has to be added for device specific primitives. However, porting designs to MAX+plus II version 8.3 from Altera has shown some incompatibility problems. This is due to lack of support on certain VHDL syntaxes.

From design productivity aspect, it has been proven that component reuse can shorten design time. Parameterisation is the key factor that enables component reuse. First time design of a parameterised module would take longer design time than fixed module. After the first time, reusing the parameterised module is simply modifications of parameters. For the case of reusing fixed module, designers have to restudy the whole architecture of the module and make necessary changes at several places, which is a tedious, unproductive, and repetitive work. Even worst when reusing a module previously done by other designers. Previous works by other researchers have also shown the same result. In cases of reusing components, only 10% to 35% of design time is needed [5].

Parameterised module has longer VHDL source code, which means a longer synthesis run time is required to translate the VHDL code. Nevertheless, synthesis time is insignificant if compared with the whole design cycle. Another draw back is, parameterised design could impose redundant hardware because of generalisation of design. This could reduce the hardware speed and increase unnecessary hardware area. By careful design of parameterised module, this problem could be avoided.

4.1 Comparison with Other EDA Tools

In the object-based HDL generator proposed by Mathur, et. al [6], C++ classes were used instead of HDL templates to generate synthesisable VHDL or Verilog codes. The authors claimed their object-based HDL generation to have advantages of optimised output HDL with removal of parameterised elements, flattening of loops, removal of overloaded functions and parameter size checking. The authors described the template based HDL codes to have disadvantages of complex HDL models due to parameterised coding, unnecessary levels of hierarchy made synthesis process inefficient and possibility to infer unnecessary conditional hardware.

However, the C++ module generators are compiled objects, which cannot be modified by users. This result in a fixed set of library modules where users cannot expand the library themselves. Design reuse cannot be facilitated. Even though if the C++ module generation were user-modifiable, designers would need to additionally learn techniques to create HDL using C++. In contrast to the VHDL Module Generator, all parameterisation elements of the VHDL Module Generator are written in VHDL code itself. A uniform design language is utilised. Although parameterisation of VHDL code increases synthesis run times, it merely represents an insignificant fraction of the entire design time. Redundancy in hardware generation could be resolved by careful design of parameterised module.

In the reuse scenario proposed by Preis, et. al [5], three main types of parameters were identified: component structure (width), logic functionality (level) and component functionality (function). The VHDL Module Generator does provide these three types of parameters, namely numeric parameter (width), logic parameter (level), and architecture option or style parameter (function). In addition, the VHDL Module Generator provides data type parameter.

Powell and Cesear [1] proposed a design methodology based on parameterised VHDL model generators. The key feature of this design methodology is support for rapid exploration of area, speed, power and functional tradeoffs, when combined with existing VHDL simulation tools, logic synthesis tools and switch-level implementation tools. The VHDL Module Generator contains the major feature of the design methodology proposed by Powell and Cesear [1].

Several commercial EDA tools are compared with the VHDL Module Generator. The Xilinx Foundation Series 2.1i from Xilinx, Inc. provides HDL design entry with design wizard (creation of VHDL skeleton with entity name, architecture name and interface ports), HDL editor with language assistant (a set of code templates), and state machine editor (conversion from state machine to HDL code). The FPGA Advantage from Mentor Graphics Corporation provides better VHDL design environment with state machine entry, block diagram entry, truth table entry, flow chart entry and VHDL textual entry itself. Other tools including Active-HDL from Aldec, Inc. and Visual HDL from Innoveda, Inc. also provide similar intuitive visual design entries. These commercial EDA tools cover design process from design entry, logic synthesis, design verification with back annotation, to design implementation. In contrast, the VHDL Module Generator only covers the design entry process without tight integration to downstream design process, but provides links to launch downstream tools.

Nevertheless, the VHDL Module Generator shows its advantages in providing useful features not offered by these commercial EDA products. Especially, the design-time hierarchical design exploration and manipulation in graphical user interface. The other useful feature is the design parameterisation in graphical user interface, which facilitates design reuse.

4.2 Novelty of VHDL Module Generator

Schematics design provides noble visualisation of design and handles hierarchical design proficiently, but it is cumbersome when handling complex digital circuit with massive wire routing. VHDL design allows higher level of description abstraction, but it does not provide visualisation of design. The VHDL Module Generator, predominantly based on VHDL entry, encapsulates advantages of both schematics and textual design entries to offer a rapid-prototyping design environment.

Conventional VHDL design entry tools enter and organise design modules in flat manner. Design hierarchy could not be visualised graphically until the whole design is synthesised successfully. The VHDL Module Generator provides design-time visualisation of design hierarchy by using graphical tree view.

VHDL coding automation, including generation of VHDL module skeleton and automatic insertion of component declaration, eliminates cumbersome coding to fulfill semantic requirements. Hence, designers can concentrate on the actual coding of design.

Conventional VHDL design entry tools does not provide facility to move design module's hierarchy. Manual coding to remove and insert component declaration is needed. With the VHDL Module Generator, manipulation of design module hierarchy can be done using clipboard functions at the design hierarchy tree view seamlessly.

Top-down, bottom-up, and combinational design methodologies can be applied visually using the design hierarchy tree view as well.

Parameterisation of design module enables component reuse. The VHDL Module Generator makes use of existing generic VHDL coding and adds several more useful types of parameters. In addition, parameter insertion and modification are done through graphical user interface, instead of textual coding. Design engineers often need to evaluate and choose among different architectural alternatives (for example, parallelism, pipelining, resource-sharing, and numeric formats) before obtaining the best solution that resolves speed, size and power consumption tradeoffs. Parameterisation feature enables fast evaluation of design alternatives. Multiple level of parameter passing is supported to enhance the parameterisation capability of design.

Interface ports of design modules can be entered, viewed, and modified from the graphical user interface of the VHDL Module Generator. Fully bi-directional updating from interface port editor to VHDL file, and in reverse is supported.

The VHDL Module Generator provides block diagram view of design. The feature further visualises the design. Apart from design visualisation and documentation, the block diagram editor can also serve as design entry for entering interface ports and sub-modules.

Many EDA tools do not support dynamic creation of reusable components. The VHDL Module Generator has a dynamic module library supporting creation of reusable components. Modules in library are organised in groups for easier search and retrieval.

5.0 CONCLUSIONS

Hierarchical organisation of design modules through GUI (Graphical User Interface), fast evaluation on design alternatives through parameterised design entries, dynamic management of reusable design modules, and automatic insertion of VHDL codes are among the key features of the VHDL Module Generator that permits rapid prototyping of designs. The following checklist highlights the features offered by the VHDL Module Generator:

- ☑ Hierarchical design entry
  - ☑ Design-time hierarchy visualisation
  - ☑ Various design methodologies
  - ☑ GUI design hierarchy manipulation
  - ☑ Design documentation
- ☑ Design parameterisation
  - ☑ Parameterised design entry
  - ☑ GUI parameter insertion and modification
  - ☑ Design space exploration
  - ☑ Multiple level parameter passing
- ☑ Module library
  - ☑ Parameterised (template-based) module library
  - ☑ Dynamic: new library module creation
  - ☑ Predefined module library
- ☑ Block diagram visualisation
  - ☑ Design documentation
  - ☑ Design hierarchy exploration
- ☑ VHDL coding automations
  - ☑ New VHDL module skeleton generation
  - ☑ Automatic component declaration
  - ☑ Automatic component renaming
- ☑ Interface port editing
  - ☑ Fully bi-directional updating (VHDL ↔ GUI)

FPGA has become the best rapid prototyping platform for hardware designers. By actual hardware implementation on reconfigurable devices, design verification can be done at actual or near hardware speed.

One of the fast-evolving applications is multimedia data compression/decompression. Transferring real-time high quality multimedia data is in demand. The key technology that enables this is compression technology. Abundance of new standards and new design alternatives has been developed. The VHDL Module Generator is an EDA design tool meant to meet the design challenges.

The VHDL Module Generator is obtainable from the following web site:
http://www.fke.utm.my/courseware/sew4274

REFERENCES

[1] S. R. Powell and T. M. Cesear "Rapid Design and Exploration of Signal Processing Systems Using a VHDL Model Generator Based Paradigm", in *2nd Annual RASSP Conference*, 1995.

[2] J. R. Armstrong, *Chip-Level Modeling with VHDL*. Singapore, Prentice Hall, 1989.

[3] FPGA Compiler II/FPGA Express VHDL Reference Manual, Synopsys, Inc., 1999.

[4] S. Mohanty, et al., "SCUBA: An HDL Data-Path/Memory Module Generator for FPGAs", in, *Proceedings of VHDL International Users' Forum, 19-22 October 1997*, pp. 135-142.

[5] V. et al. Preis, "A Reuse Scenario for the VHDL-Based Hardware Design Flow", in, *Proceedings of EURO-DAC'95, 18-22 September 1995*, pp. 464-469.

[6] A. Mathur, et. al., "HDL Generation from Parameterized Schematic Design System" in, *Proceedings of ASIC Conference and Exhibit, Tenth Annual IEEE International, 7-10 September 1997*, pp. 130-134.

[7] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York, Van Nostrand Reinhold, 1993.

[8] P. M. Athanas and A. L. Abbott, "Real-Time Image Processing on a Custom Computing Platform". *Computer*, Vol. 28, No. 2, February 1995, pp. 16-25.

[9] Synopsys (XSI) Synthesis and Simulation Design Guide, Xilinx, Inc., 1998.

BIOGRAPHY

Mohamed Khalil Hani obtained his B.Eng. in Communication from University of Tasmania, M. Eng. in Computer Engineering from Florida Atlantic University, and Ph.D. in Digital System and Computer Engineering from Washington State University. He is currently the Vice-Dean at Faculty of Electrical Engineering, Universiti Teknologi Malaysia. His expertise includes VLSI architecture, artificial intelligence, digital design automation, and computer-aided digital design.

Kah Hoe Koay obtained his first degree with first class honour in Electrical Engineering from Universiti Teknologi Malaysia in 1998. Currently, he has completed his masters degree in Electronic Engineering at the same institution. His research interest includes software engineering, artificial intelligence and computer engineering. He is an affiliate member of IEEE Computer Society since 1997 and member of Institute of Engineer Malaysia since 1997 as well.