MapleDSP

VBuilder: A Schematic to Verilog converter

Operating Manual

Written by:
Martin Alcock, M. Sc

Revision: 0.2

Table of Contentse

List of Tables

List of Figures

Bibliography

[1] gnu.org, "General Public Licence," [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 25th February 2018].

[2] Wikipedia, "XML," [Online]. Available: https://en.wikipedia.org/wiki/XML. [Accessed 27 07 2023].

Document Revision History

Date	Rev	Description
July 2023	0.1	Initial Draft

Table 1 Revision History

Glossary of Terms

FPGA	Field Programmable Gate Array. A general purpose logic device.
Verilog	A language for describing hardware components in an FPGA
Schematic	A diagram illustrating the functionality of a system
Netlist	A list of connections between components
CAD	Computer Automated Design
KiCAD	A schematic entry and PCB layout CAD system
IP	Intellectual Property
XML	eXtensible Markup Language: a set of codes, or tags, that describes the text in a digital document.

Intellectual Property Notice

The PiREx trademark, hardware components and all intellectual property described herein is the exclusive property of Praebius Communications Inc ("the owner"), all rights are reserved.

The owner grants licence to any Amateur for personal or club use, on an as is and where is basis under the condition that its use is for non-commercial activities only, all other usages are strictly prohibited. Terms and conditions are governed by the GNU public licence [1].

No warranty, either express or implied or transfer of rights is granted in this licence and the owner is not liable for any outcome whatsoever arising from such usage.

Disclaimer

This document is a preliminary release for a product still in development and may be subject to change in future revisions.

Introduction

VBuilder is part of a methodology for reusing IP modules written in the Verilog language on a building block basis. Modules are encapsulated with a top level template, which implements a common top level connection methodology. The format and content of this is up to the individual designer.

Each IP module is represented by a schematic symbol, which can be instantiated using a schematic editor, and then wired to other modules at will, representing a pictorial display of a system. A top level component implements the connections to the pins of the FPGA, or to another container module.

This enables a system design methodology that implements a framework that is developed once to interact with devices that are connected to the FPGA, therefore simplifying future design efforts. Processing modules can be then instantiated as needed to fit different system requirements.

VBuilder is a utility program to enable this design philosophy for modules described in a netlist, which can be generated by a schematic capture program. The netlist is generated in XML [2] format, which is read by VBuilder and converted into Verilog code.

Generating the Schematic

The schematic diagram is generated by interconnecting a series of symbols, each representing a Verilog component. To create the component, a library editor is used. Figure 1 illustrates a sample component which is a DTMF decoder. This module takes in an audio source, a clock and reset, and provides a detection output and a connection to an external CPU.

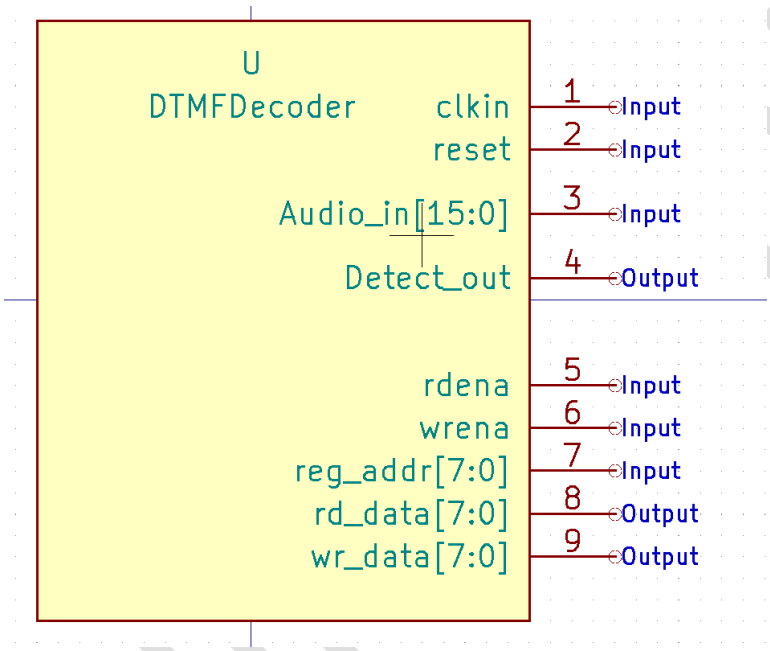

Figure 1. Sample Schematic Component

Signal definitions

Signals names are taken from the pins on the component, which must have a unique designator, and can be numeric or alphanumeric. The pin number is not used in the Verilog generation but is used by the netlist to reference a unique connection on the component.

The pin type defines the direction of each signal and can be any one of the following as illustrated in Table 2.

Type	Purpose
Input	An input only pin.
Output	An output only pin.
Inout	A bi-directional pin.

Table 2 Allowable pin types

Naming conventions

The signal name can be either in upper or lower case. If a bus is being defined that has a size greater than one, then the name must contain the size in the Verilog convention of [High order bit:Low order bit]. The width of the signal is defined by the formula:

$$Hob - Lob - 1$$

Names with a negated polarity are supported by most CAD systems, by prepending a tilde (~) character to the signal name. This is permissible, but it is renamed internally to have a "_n" appended to it, and the tilde is removed, as an inversion in a signal name is not allowed in Verilog.

A Simple example

A simple example has only one component. To create the top level connections, the signals from modules are exported, by connecting them to a symbol with the keyword 'export' in its name. The pin type and direction on the export symbol is not important, as it is taken from the definition on the component.

Figure 2 illustrates the same component with its signals exported.

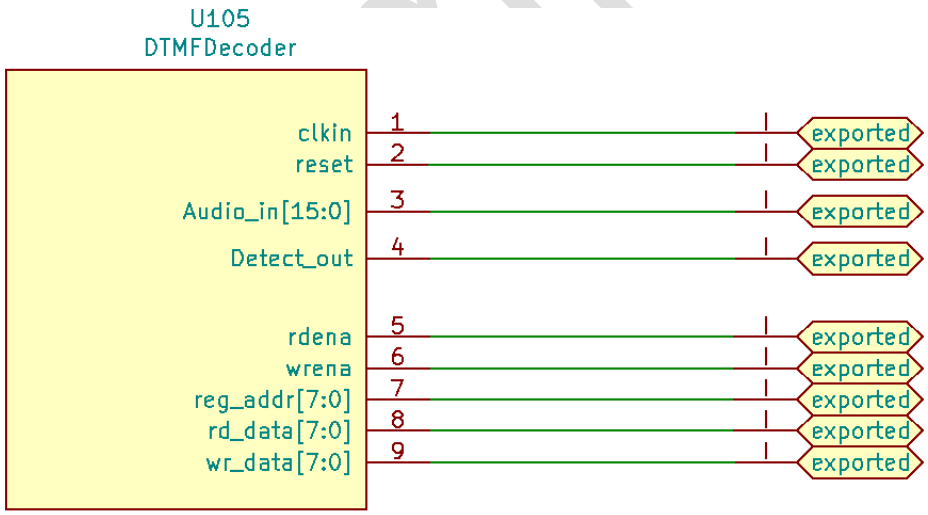

Figure 2 Symbol with exported signals

Sample Output

The code below is a sample generated from the symbol in Figure 2.

```
//---------Module Definition--------

module DTMF_Decoder (
      input       wire              clkin,
      input       wire              reset,
      input       wire  [15:0]      Audio_in,
      output      wire  [15:0]      Audio_out,
      output      wire              Detect_out,
      input       wire              rdena,
      input       wire              wrena,
      input       wire  [7:0]       reg_addr,
      output      wire  [7:0]       rd_data,
      output      wire  [7:0]       wr_data
);

//---------Instantiation of DTMFDecoder---------

DTMFDecoder
      DTMFDecoder_inst
            (
                  .clkin                  (clkin),
                  .reset                  (reset),
                  .Audio_in               (Audio_in),
                  .Audio_out              (Audio_out),
                  .Detect_out             (Detect_out),
                  .rdena                  (rdena),
                  .wrena                  (wrena),
                  .reg_addr               (reg_addr),
                  .rd_data                (rd_data),
                  .wr_data                (wr_data)
            );


endmodule
```

Figure 3 Generated Output

A more complex example

Figure 4 illustrates a more complex example with a second component and named nets.

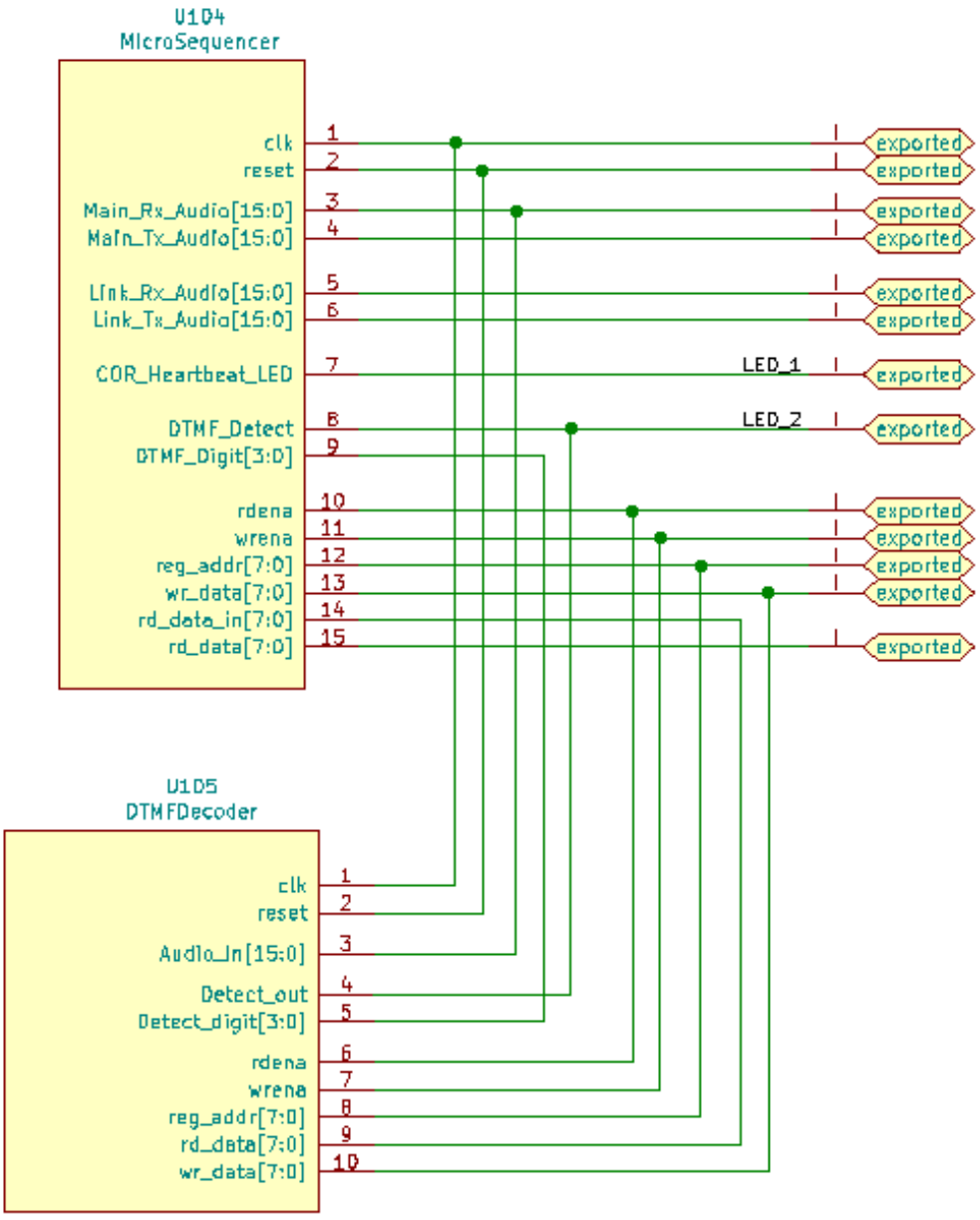

Figure 4 A more complex example

A second component is instantiated in this example, which shares nets with the first component. The exported net name is taken from one of two criteria:

1. If the net contains a label, then the exported variable name is taken from the label. Label text can be upper or lower case.
2. The pin name of the first non-export component found in the netlist.

To ensure that a specific export name is used, a label on the exported net should be used.

Output of the second example

Figure 5 illustrates the output of the second example. Note that the top level wire names reflect the labels on the schematic.

```
//---------Module Definition--------

module DTMF_Decoder (
      input wire              clk,
      output      wire        LED_1,
      output      wire        LED_2,
      input  wire [15:0]      Link_Rx_Audio,
      output wire [15:0]      Link_Tx_Audio,
      input  wire [15:0]      Main_Rx_Audio,
      output wire [15:0]      Main_Tx_Audio,
      output wire [7:0]       rd_data,
      input  wire             rdena,
      input  wire [7:0]       reg_addr,
      input  wire             reset,
      input  wire [7:0]       wr_data,
      input  wire             wrena
);

//---------Local Wires--------

wire  [7:0]             LocalWire_Net_10;
wire  [3:0]             LocalWire_Net_7;
```

```
//---------Instantiation of DTMFDecoder---------

DTMFDecoder
      DTMFDecoder_inst
            (
                  .clk                          (clk),
                  .reset                        (reset),
                  .Audio_in                     (Main_Rx_Audio),
                  .Detect_out                   (LED_2),
                  .Detect_digit                 (LocalWire_Net_7),
                  .rdena                        (rdena),
                  .wrena                        (wrena),
                  .reg_addr                     (reg_addr),
                  .rd_data                      (LocalWire_Net_10),
                  .wr_data                      (wr_data)
            );


//---------Instantiation of MicroSequencer---------

MicroSequencer
      MicroSequencer_inst
            (
                  .clk                          (clk),
                  .reset                        (reset),
                  .Main_Rx_Audio                (Main_Rx_Audio),
                  .Main_Tx_Audio                (Main_Tx_Audio),
                  .Link_Rx_Audio                (Link_Rx_Audio),
                  .Link_Tx_Audio                (Link_Tx_Audio),
                  .COR_Heartbeat_LED            (LED_1),
                  .DTMF_Detect                  (LED_2),
                  .DTMF_Digit                   (LocalWire_Net_7),
                  .rdena                        (rdena),
                  .wrena                        (wrena),
                  .reg_addr                     (reg_addr),
                  .wr_data                      (wr_data),
                  .rd_data_in                   (LocalWire_Net_10),
                  .rd_data                      (rd_data)
            );


endmodule
```

Figure 5 Two component example generated code

Generating the Netlist

The netlist is exported from the CAD program in XML format. There are three XML tags that are parsed, as listed in

XML Tag	Purpose
<components>	Contains a list of components used in the design
<libparts>	Describes library components implemented in the design
<nets>	Contains a list of all interconnections in the design

Table 3 XML tags used in processing netlist

Component Tag

The component tag contains one entry for each component in the design. The tags processed are show in Table 4.

Tag	Attribute	Contents
<comp>	-	Encapsulates a component entry
	ref	Contains the reference designator for this component
<value>	-	Contains the component name as a text field

Table 4 Component definition XML tags

Libparts Tag

The libparts tag contains an entry for each unique part in the design. A part only appears once in this section, although it may be used more than once in the component section.

Tag	Attribute	Comments
<pins>		Encapsulates the component pin entries
<pin>	num	Pin number, can be alphanumeric
	name	Name of the pin
	type	Type as per Table 7.

Table 5 Library parts definition XML tags

Nets tag

The connections net definitions are show in Table 6.

Tag	Attribute	Comments
<nets>		Encapulates the netlist entries
<net>	code	Defines a specific net
	name	Name of the net, either machine generated, or one listed in Table 7
<node>	ref	Defines the component to which this net is attached
	pin	Defines the pin to which this net is attached.

Table 6 Net definition XML tag

Specialized Nets

There are two specialized nets that are supported, a shown in Table 7. These do not connect to any outside component.

Net Name	Purpose
VCC	Ties a signal, or bus, to a logic one
GND	Ties a signal or bus to a logic zero.

Table 7 Specialized nets

Running the software

The software is packaged as a Java jar file, the command line is show below:

```
java -jar VBuilder.jar -o <module> -X <netlist>
```

Where:

Switch	Use
-o	Defines the output module name. The file name is the same with '.v' at the end
-X	Defines the XML input file. The entire name must be specified

Table 8 Command line switches

Processing the netlist

The netlist is processed in the following order:

1. Components. The component in the design are read to build a symbol table.
2. Library Parts. The library parts are read to define the pins in each component.
3. Netlist. The netlist is read to define the interconnections between component pins.

After processing the netlist, the following is then performed:

- Export symbols are processed to promote their connections to the top level, they are then removed from the symbol table.
- VCC and GND nets are processed and removed.
- A list of internal wires is generated from the netlist, except those at the top level

When completed, the module definition is emitted, followed by the internal wire definitions, followed by the component instantiations.