8B10B Decoder

User Guide

Product Brief

Introduction

The 8b10b line code is widely used to achieve DC-balance and bounded disparity when transmitting serial data over a medium. It ensures enough state transitions to allow the use of a Clock Recovery unit. Thus both data and clock are encoded in the serial data stream.

The RoaLogic 8b10b decoder is a full implementation of the 8b10b scheme described by Widmer and Franaszek, as used by many high-speed protocols. The decoder takes 10bit symbols and decodes them into 8bit data while simultaneously checking for bit errors. The core detects the special comma symbols and automatically detects K28.5.
Two cores can be cascaded to support 16b20b decoding.

Features

- 8b10b decoding
- Can be cascaded for 16b20b decoding
- Supports industry standard comma “K” symbols
- Pipelined design
- Low latency
- Fully synthesizable
- Target technology independent

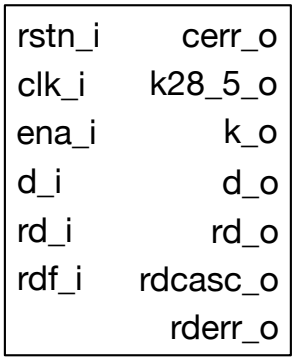


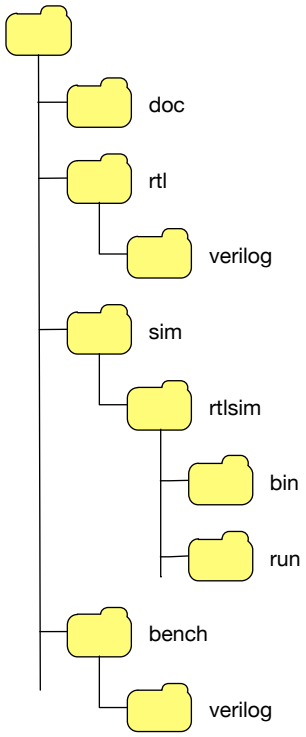
Table of Contents

1. Getting Started

Deliverables

All IP is delivered as a zipped tarball, which can be unzipped with all common compression tools (like unzip, winrar, tar, …).

The tarball contains a directory structure as outlined below.



The *doc* directory contains relevant documents like user guides, application notes, datasheets.

The *rtl* directory contains the actual IP design files. Depending on the license agreement the 8b10b decoder is delivered as either encrypted Verilog-HDL or as plain Verilog-HDL RTL source files. Encrypted files have the extension .enc.v, plain source files have the extension .v. The files are encryption according to the IEEE-P1735 encryption standard. Encryption keys for Mentor Graphics (Modelsim, Questasim, Precision), Synplicity(Synplify, Synplify-Pro), and Aldec (Active-HDL, Riviera-Pro) are provided. As such there should be no issue targeting any existing FPGA technology.
If any other synthesis or analysis tool is used then a plain source RTL delivery may be need. A separate license agreement and NDA is required in these cases.

The *bench* directory contains the (encrypted) source files for the testbench.

The *sim* directory contains the files/structure to run the simulations. See 'running the testbench' for instructions on how to use the makefile.

Running the testbench

The 8b10b decoder IP comes with a dedicated testbench that tests all features of the design and finally runs a full random test. The testbench is started from a Makefile that is provided with the IP.
The Makefile is located in the <*install_dir*>/sim/rtlsim/run directory. The Makefile supports most commonly used simulators; Modelsim/Questasim, Cadence ncsim, Aldec Riviera, and Synopsys VCS.

To start the simulation enter the <*install_dir*>/sim/rtlsim/run directory and type: **make <*simulator*>**. Where simulator is any of: msim (for modelsim/questasim), ncsim (for Cadence ncsim), riviera (for Aldec Riviera-Pro), or vcs (for Synopsys VCS). For example type **make msim** to start the testbench in Modelsim/Questasim.

Self-checking testbench

The testbenches is a self-checking testbench intended to be executed from the command line. There is no need for a GUI or a waveform viewer. Once the testbench completes it displays a summary and closes the simulator.

Makefile setup

The simulator is executed in its associated directory. Inside this directory is another Makefile that contains simulator specific commands to start and execute the simulation. The <*install_dir*>/sim/rtlsim/run/Makefile enters the correct directory and calls the simulator specific Makefile.
For example modelsim is executed in the <*install_dir*>/sim/rtlsim/run/msim directory. Typing **make msim** loads the main Makefile, which then enters the msim sub-directory and calls its Makefile. This Makefile contains commands to compile the RTL and testbench sources with Modelsim, start the Modelsim simulator, and run the simulation.

Makefile backup

The <*install_dir*>/sim/rtlsim/bin directory contains backups of the original Makefiles. It may be desirable to modify or extend the Makefiles or to completely clean the run directory. Use the backups to restore the original setup.

No Makefile

For users unfamiliar with Makefiles or those on systems that do not natively support make (e.g. Windows) a run.do file is provided that can be used with Modelsim/Questasim and Riviera-Pro.

2. Specifications

Functional Description

The 8b10b decoder translates 10bit symbols into 8bit data. The symbols are generated using an 8b10b encoder, like Roa Logic's 8b10b Encoder IP. The 8b10b decoder translates the encoded symbols back to the original data. It extracts both data Dx.y and comma Kx.y symbols, checks for errors in both data and comma symbols, and checks for disparity errors.
The table below shows the supported comma symbols. Any other comma symbol triggers an error.

K-code	8bit equivalent code
K23.7	8'b111_10111
K28.0	8'b000_11100
K28.1	8'b001_11100
K28.2	8'b010_11100
K28.3	8'b011_11100
K28.4	8'b100_11100
K28.5	8'b101_11100
K28.6	8'b110_11100
K28.7	8'b111_11100
K29.7	8'b111_11101
K30.7	8'b111_11110
10B_ERR	8'b111_11111

Table 1: Symbol Codes

Latency

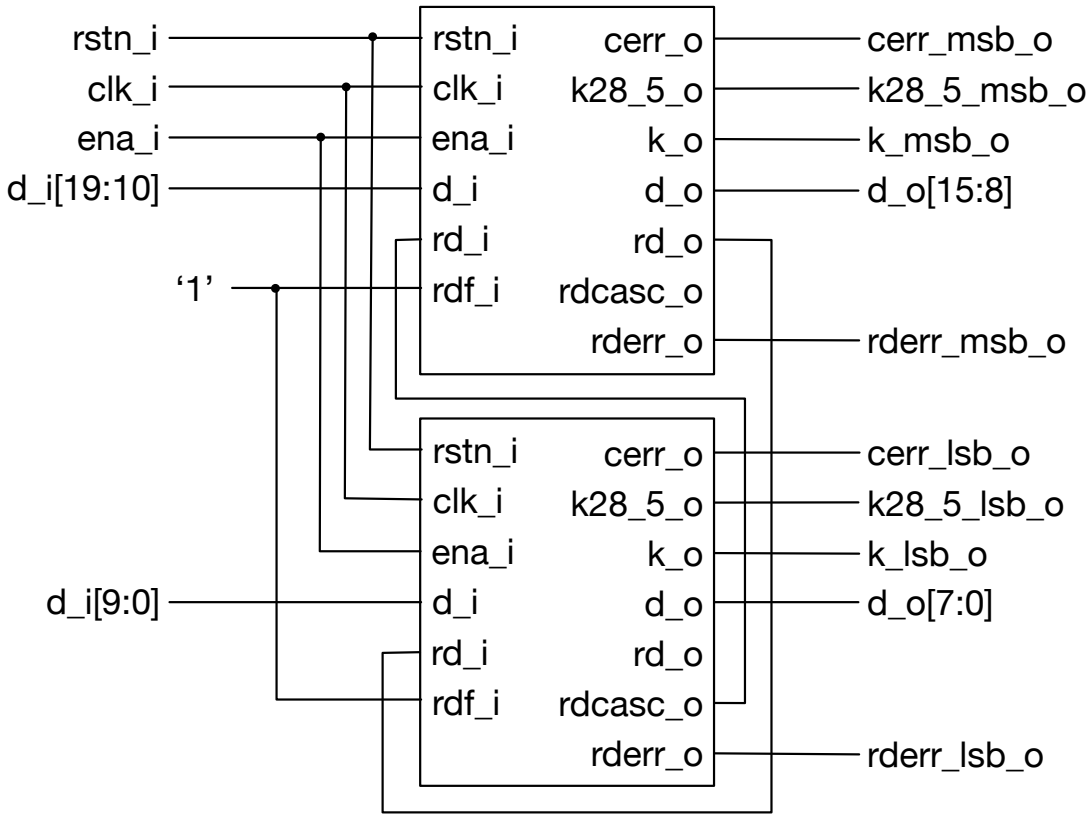
The core has a latency of two (2) clock cycles. This means that any 10bit symbol presented at the *d_i* input appears after two clock cycles as a decoded 8bit byte at the *d_o* output. A clock cycle is a rising edge of *clk_i* while *ena_i* is asserted ('1'). Negating ('0') *ena_i* retains the core in its current state and does not change the outputs; the same *d_o* value retains while *ena_i* is negated ('0').

Kx.y and Idle Detection

The core asserts ('1') k_o when it detects a comma symbol. The associated data byte is presented on d_o.
In addition the core has an **early** idle (K28.5) symbol indication. It asserts ('1') k28_5_o one cycle before it asserts ('1') k_o and set d_o to 8'b101_11100. This is useful for automatically removing idle bytes from the data stream.

Cascading Decoders

Two 8b10b decoders can be cascaded to create a 16b20b decoder. The cores must be connected as shown below.



Core Ports

Signal	Size	Direction	Description
rstn_i	1	Input	Asynchronous active low reset
clk_i	1	Input	Clock input
ena_i	1	Input	Clock enable input
d_i	10	Input	Symbol input
rd_i	1	Input	
rdf_i	1	Input	
d_o	8	Output	Decoded data output
k_o	1	Output	Comma symbol
k28_5_o	1	Output	Idle symbol
cerr_o	1	Output	Symbol error
rdcasc_o	1	Output	
rd_o	1	Output	
rderr_o	1	Output	Running disparity error

rstn_i
When the active low reset input is asserted ('0'), the core is put into its initial reset state.

clk_i
The entire decoder operates at the system clock driven into the clk_i port.

ena_i
The core progresses a state when the clock enable input is asserted ('1'). When the clock enable input is negated ('0') all inputs are ignored and the core retains its current state.

d_i
The to-be-decoded 10bit symbol is presented on the data input

rd_i
The running disparity input presents a new current running disparity to the core. When the force running disparity (*rdf_i*) input is asserted the value of *rd_i* is taken as the new running disparity. A zero ('0') on *rd_i* generates a positive or neutral disparity. A one ('1') on *rd_i* generates a negative or neutral disparity. The *rd_i* input is also used when cascading encoders.

rdf_i
When asserted ('1'), the force running disparity input forces the core to use the disparity input (*rd_i*) as the current running disparity, instead of the internally

calculated one. This can be used to force synchronization patterns or manually introduce disparity errors.
The *rdf_i* input is also used when cascading encoders.

d_o
The data output (*d_o*) contains the decoded 8bit data.

k_o
The comma (Kx.y) output is asserted ('1') when the core decodes a comma symbol. When k_o is asserted ('1') the data output (*d_o*) contains the valid 8bit comma data.

k28_5_o
When the core detects a K28.5 symbol, the idle symbol *k28_5_o* output is asserted ('1') one cycle before *k_o* and *d_o*.

cerr_o
The character error output is asserted ('1') when an illegal 10bit symbol or 10BERR is detected.

rdcasc_o
The running disparity cascade output is used when cascading encoders.

rd_o
The running disparity output is used when cascading encoders.

rderr_o
The running disparity error output is asserted ('1') when the core detect a disparity error.

3. Resources

Below are some example implementations for various platforms.
All implementations are push button, no effort has been undertaken to reduce area or improve performance.

Platform	DFF	Logic Cells	Memory	Performance (MHz)
lfxp3c-5	34	103	0	226MHz

4. Revision

Date	Rev.	Comments
19-dec-2014	1.0	Official release