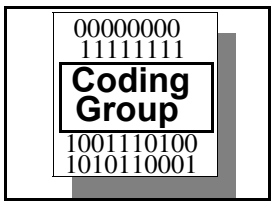
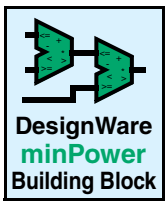




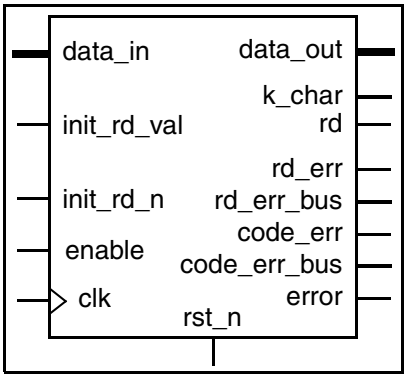
DW_8b10b_dec

8b10b Decoder

Version, STAR and Download Information: IP Directory

Features and Benefits

- Configurable data width
- Configurable simplified Special Character indicator flags (for protocols requiring only the K28.5 special character)
- Synchronous initialization of Running Disparity with design specified value
- All outputs registered
- Provides minPower benefits with the DesignWare-LP license (Get the minPower version of this datasheet.)



Description

DW_8b10b_dec decodes 1 to 16 bytes of data using the 8b10b Direct Current (DC) balanced encoding scheme developed at IBM.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock input
rst_n	1 bit	Input	Asynchronous reset input, active low
init_rd_n	1 bit	Input	Synchronous initialization control input, active low
init_rd_val	1 bit	Input	Value of initial Running Disparity
data_in	*bytes* × 10 bit(s)	Input	Input 8b/10b data for decoding
error	1 bit	Output	Active high, error flag indicating the presence of any type of error (running disparity or coding) in the information currently decoded on `data_out`
rd	1 bit	Output	Current Running Disparity (after decoding data presented at `data_in` to `data_out`)
k_char	*bytes* bit(s)	Output	Special character indicators (one indicator per decoded byte)
data_out	*bytes* × 8 bit(s)	Output	Decoded output data
rd_err	1 bit	Output	Active high, error flag indicating the presence of one or more Running Disparity errors in the information currently decoded on `data_out`

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
code_err	1 bit	Output	Active high, error flag indicating the presence of a coding error in at least one byte of information currently decoded on `data_out`
enable	1 bit	Input	Enables register clocking
rd_err_bus	*bytes* bit(s)	Output	This bus indicates which bytes of the incoming bus have Running Disparity errors (one bit per incoming `data_in` byte with bit 0 corresponding to an error seen in the least significant 10 bits of input data)
code_err_bus	*bytes* bit(s)	Output	This bus indicates which bytes of the incoming bus have Coding errors (one bit per incoming `data_in` byte with bit 0 corresponding to an error seen in the least significant 10 bits of input data).

Table 1-2 Parameter Description

Parameter	Values	Description
bytes	1 to 16 Default: 2	Number of bytes to encode
k28_5_only	0 or 1 Default: 0	Special Character subset control parameter 0 - for all special characters decoded, 1 - for only K28.5 decoded [when k_char = HIGH implies K28.5, all other special characters indicate an error]
en_mode	0 or 1 Default: 0	Enable control 0 - the `enable` input port is not connected (backward compatible with older components) 1 - when `enable` = 0 the decoder is stalled
init_mode	0 or 1 Default: 0	Initialization mode for running disparity 0 - during active `init_rd_n` input, delay `init_rd_val` one clock cycle before applying it to `data_in` input in calculating `data_out` (backward compatible with older components) 1 - during active `init_rd_n` input, directly apply `init_rd_val` to `data_in` input (with no clock cycle delay) in calculating `data_out`
rst_mode	0 or 1 Default: 0	Reset mode: 0 = asynchronous reset 1 = synchronous reset
op_iso_mode	0 to 4 Default: 0	Operand Isolation Mode (controls datapath gating for minPower flow) 0 = DC variable (DW_lp_op_iso_mode) can control the op iso mode 1 = 'none' 2 = 'and' 3 = 'or' 4 = preferred isolation style: 'and'

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Functional Description

All outputs of the DW_8b10b_enc are registered, and as such the decoded results of input data (on `data_in`) are not seen at the output ports, `data_out`, `k_char`, `rd` and `error`, until after the rising edge of the `clk` input. The `init_rd_n` input synchronously initializes the running disparity to the value present on the `init_rd_val` port.

The code words generated are either perfectly balanced (that is, they contain the same number of ones as they do zeros) or are unbalanced at 60% or 40% bias. A data byte that does not generate a balanced code generates one of two possible unbalanced code words (60% and 40%). The aggregate DC value is kept balanced by remembering the direction in which the last unbalanced code word was biased and generating the next unbalanced code word in the opposite bias direction. The one bit memory that recalls the bias of the last unbalanced code word is called the Running Disparity. A Running Disparity value of zero is synonymous with negative Running Disparity (-). A Running Disparity value of one is synonymous with positive Running Disparity (+).

So, in decoding 8b/10b data the running disparity is checked along the way as part of validating code words. This is an important aspect in being able to detect shifts in bit transitions that would convert a valid coded character into another valid character while changing the DC balance (thus affecting the running disparity). Such an error may only be detected when a subsequent character that depends on a particular state of the running disparity is received.

Input data presented on the `data_in` port is decoded from the most significant 10-bit byte down to the least significant 10-bit byte. The first bit of serial data that was received in a word is the most significant bit of `data_in` and the last received serial bit is bit zero of `data_in`. The most significant 10-bit byte on `data_in` is decoded to the most significant 8-bit byte on `data_out`. The second most significant 10-bit byte on `data_in` is decoded to the second most significant 8-bit byte on `data_out`. This continues down until finally the least significant 10-bit byte on `data_in` is decoded to the least significant 8-bit byte on `data_out`.

Reset

The `rst_n` input port asynchronously resets the Running Disparity (reflected on the `rd` output port) to zero (-), as the `error` and `rd_error` ports are reset to LOW, while the `data_out` and `k_char` ports are set to all zeros. Initialization of running disparity does not cause the data output registers to be initialized. One interesting thing that `init_rd_n` and `init_rd_val` can be used for is to “disturb” the running disparity in order to force errors to be detected with valid 8B/10B data being processed. One method to force errors is to simply continuously initialize the running disparity while decoding. This causes all 8b/10b words to be decoded with a constant initial running disparity. In systems that tightly control the running disparity during idle times, a single error event can easily be forced by initializing the running disparity to the “wrong” state.

Enable

The synthesis parameter, en_mode, determines the function of the input port, enable. When `en_mode=1`, DW_8b10b_dec will use the input port enable to control clocking of registers in the module. With `en_mode=1`, the module will decode information from `data_in` to `data_out` on every clock cycle for which `enable=1`. No decoding occurs for clock cycles where `enable=0` (with `en_mode=1`). When `en_mode=0` (which is the default used when `en_mode` is not set in the design), the `enable` input is not connected.

Running Disparity Initialization

The `init_rd_n` input is used to synchronously initialize the internal running disparity (reflected on the output port, `rd`) to the value present on the port, `init_rd_val`.

The initialization of the running disparity behaves differently based on the synthesis parameter init_mode. When `init_mode=0` (default and the backward compatible setting) and with init_rd_n asserted, the init_rd_val input is delayed one clock before it is applied to the data_in input in generating the data_out output. Therefore, two clock cycles after asserting init_rd_val the resulting data_out of which init_rd_val is calculated from is available. The rd output reflects the value of init_rd_val (with init_rd_n asserted) with a one-cycle latency (see Figure 1 in Timing Waveforms). When init_mode=1 and with init_rd_n asserted, the init_rd_val input is applied without any clock delay to the data_in input. Thus, the data_out output reflected by init_rd_val being applied to data_in is available on the next clock cycle along with the newly calculated rd output (see Figure 2 in Timing Waveforms).

Outputs

data_out

Data from the `data_in` port is decoded (10 bits per byte symbol) and clocked into an output register that drives the `data_out` output port. For each 10 bits on `data_in` there are 8 bits on `data_out`. The most significant 10 bits of data_in will be decoded and clocked into a register connected to the most significant 8 bits of data_out. Similarly, the least significant 10 bits of `data_in` are decoded and clocked into a register connected to the least significant 8 bits of `data_out`.

k_char

The `k_char` output port indicates whether or not an 8-bit decoded symbol on the `data_out` port is a valid special character. The `k_char` port has 1 bit for each byte in the `data_out` port. The most significant bit of `k_char` indicates whether the most significant byte present on `data_out` (decoded from the most significant 10 bits on `data_in`, one clock earlier) is a control character. Similarly, the least significant bit of `k_char` indicates whether the least significant byte present on `data_out` (decoded from the least significant 10 bits on `data_in`, one clock earlier) is a control character. Refer to Table 1-4 on page 6 for more information about special character decoding.

rd

The `rd` output port provides the current state of running disparity after decoding the byte or bytes present on the `data_out` output port. The state of running disparity is normally derived from the data stream being decoded as it indicates the polarity of the last symbol that contained an imbalanced code group.

error

The error output indicates a decoding error was found when the data currently present on the `data_out` port was decoded. The error output is active whenever the decoder detects an error in any byte symbol whose decoded output is present on the data_out port. There are two basic kinds of errors: code errors and running disparity errors. There are also separate outputs for the two types of errors for use in systems that may want to consider the presence of only running disparity errors as a "soft" error. Note that the error port is effectively the logical OR of the ports, `code_err` and `rd_err`.

code_err

The `code_err` output port indicates (when active) that the decoder has detected one or more invalid 10- bit symbols on the `data_in` input port. The `code_err` output is register at the same time as the `data_out` bus so that its status relates directly to the current content on the `data_out` port. Any time a code error is detect, it should be considered a "hard" error and the decoded data on the `data_out` port may be corrupt.

The `code_err` port is also affected by the parameter, k28_5_mode. When k28_5_mode is 1, the only valid special character is the "K28.5" character. So, with the parameter k28_5_mode set to one, all other special characters received will cause the `code_err` output (as well as the error output) to be activated. See Table 1-4 on page 6 for more details.

code_err_bus

The `code_err_bus` output indicates which bytes on the `data_in` port contain a detected Coding error. The `code_err_bus` port has 1 bit for each byte in the `data_in` port containing an error. The most significant bit of `rd_err_bus` indicates whether the most significant byte present on `data_in` is a contains a Coding error. Similarly, the least significant bit of `rd_err_bus` indicates whether the least significant byte present on `data_in` is has an error.

rd_err

The `rd_err` output port indicates (when active) that the decoder has detected one or more valid 10- bit symbols on the `data_in` input port that occurred with the incorrect running disparity. The `rd_err` output is captured by a register at the same time as the `data_out` bus so that its status relates directly to the current content on the `data_out` port. Any time a running disparity error is detected and a code error is not, it may be considered a "soft" error and the decoded data on the data_out port reflects the proper decoding of byte symbols found even in context of reversed running disparity.

rd_err_bus

The `rd_err_bus` output indicates which bytes on the `data_in` port contain a detected Running Disparity error. The `rd_err_bus` port has 1 bit for each byte in the `data_in` port containing an error. The most significant bit of `rd_err_bus` indicates whether the most significant byte present on `data_in` is a contains a Running Disparity error. Similarly, the least significant bit of `rd_err_bus` indicates whether the least significant byte present on `data_in` is has an error.

Special Character Decoding

Each byte of the port `data_in` has an associated special character indicator output on the output port, `k_char`. A LOW on `k_char`[0] indicates that `data_out`, bits 7 through 0, contains one data byte (i.e. is not a valid special character). When `k_char`[0] is HIGH then bits 7 though 0 of data_out represent a valid special

character (as shown in Table 1-4 on page 6). When the parameter `k28_5_only` is 0 and `k_char[`*i*`]` is HIGH, the particular special character that was decoded is distinguished by the value on the corresponding eight bits of the `data_out` port as indicated in Table 1-4, provided that there were no code errors. When `k28_5_only` is 1, then `k_char[`*i*`]` HIGH indicates reception of a valid K28.5 special character (all other special characters received with `k28_5_only` = 1 will be flagged as code errors (`code_err` active) without a `k_char` indicator).

Table 1-4 Special Character Decoding

Character Name	data_out value	k28_5_mode	k_char[*i*]	code_err
K28.0	00011100 (1Ch)	0	1	0
K28.1	00111100 (3Ch)	0	1	0
K28.2	01011100 (5Ch)	0	1	0
K28.3	01111100 (7Ch)	0	1	0
K28.4	10011100 (9Ch)	0	1	0
K28.5	10111100 (BCh)	0	1	0
K28.6	11011100 (DCh)	0	1	0
K28.7	11111000 (FCh)	0	1	0
K23.7	11110111 (F7h)	0	1	0
K27.7	11111011 (FBh)	0	1	0
K29.7	11111101 (FDh)	0	1	0
K30.7	11111110 (FEh)	0	1	0
K28.0	00011100 (1Ch)	1	0	1
K28.1	00111100 (3Ch)	1	0	1
K28.2	01011100 (5Ch)	1	0	1
K28.3	01111100 (7Ch)	1	0	1
K28.4	10011100 (9Ch)	1	0	1
K28.5	10111100 (BCh)	1	1	0
K28.6	11011100 (DCh)	1	0	1
K28.7	11111000 (FCh)	1	0	1
K23.7	11110111 (F7h)	1	0	1
K27.7	11111011 (FBh)	1	0	1
K29.7	11111101 (FDh)	1	0	1
K30.7	11111110 (FEh)	1	0	1

Timing Waveforms

Figure 1-1 Running Disparity Initialization when init_mode=0

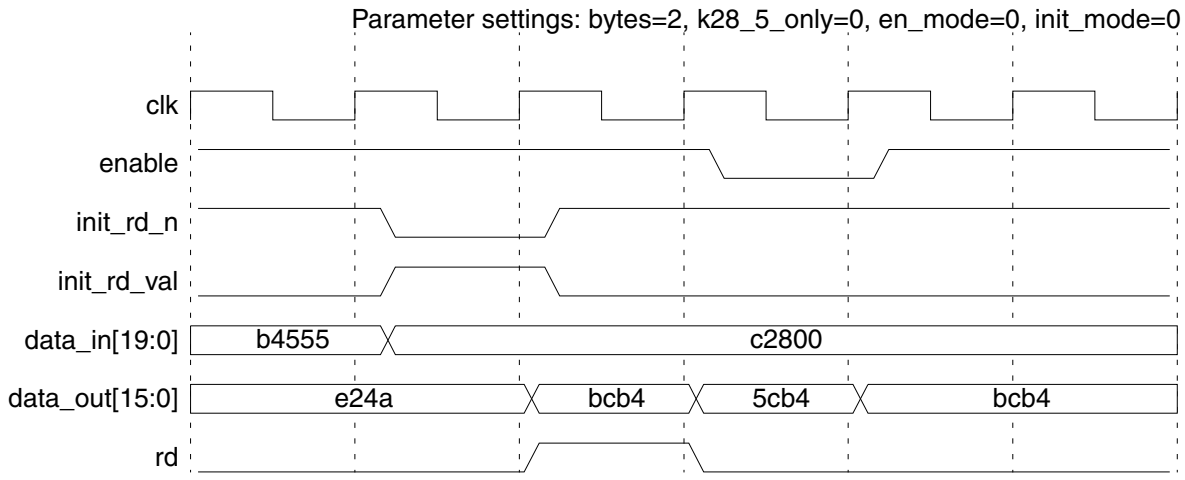
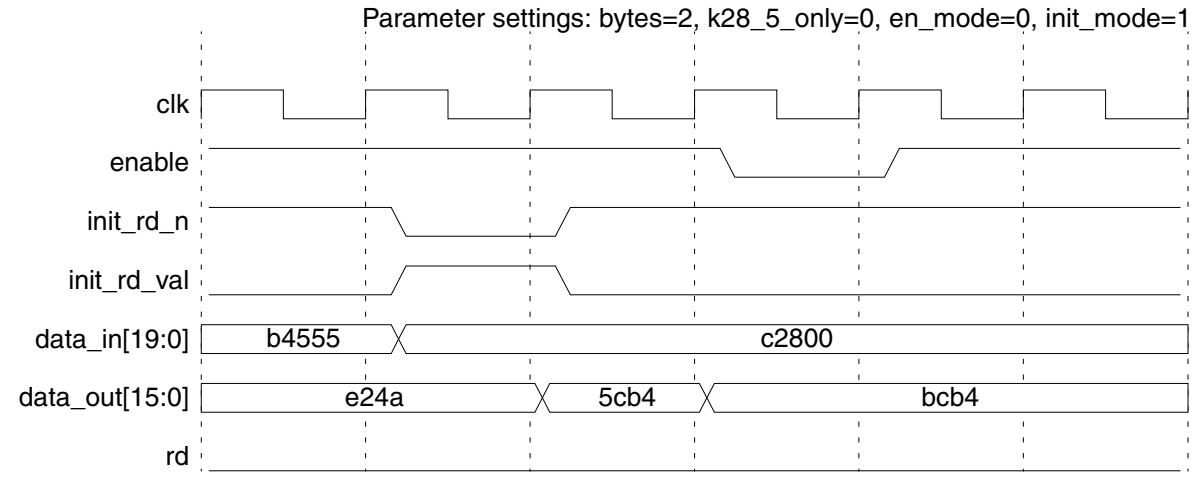


Figure 1-2 Running Disparity Initialization when init_mode=1



Related Topics

- Coding Group – Coding Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_8b10b_dec_inst is
  generic (inst_bytes       : integer := 2;    inst_k28_5_only : integer := 0;
           inst_en_mode     : integer :=0 ;    inst_init_mode : integer := 1;
           inst_rst_mode    : integer := 0;    inst_op_iso_mode : integer := 0 );
  port (inst_clk         : in std_logic;
        inst_rst_n       : in std_logic;
        inst_init_rd_n   : in std_logic;
        inst_init_rd_val : in std_logic;
        inst_data_in     : in std_logic_vector(inst_bytes*10-1 downto 0);
        error_inst       : out std_logic;
        rd_inst          : out std_logic;
        k_char_inst   : out std_logic_vector(inst_bytes-1 downto 0);
        data_out_inst : out std_logic_vector(inst_bytes*8-1 downto 0);
        rd_err_inst      : out std_logic;
        code_err_inst    : out std_logic;
        inst_enable      : in std_logic;
        rd_err_bus_inst  : out std_logic_vector(inst_bytes-1 downto 0);
        code_err_bus_inst: out std_logic_vector(inst_bytes-1 downto 0) );
end DW_8b10b_dec_inst;

architecture inst of DW_8b10b_dec_inst is
begin

  -- Instance of DW_8b10b_dec
  U1 : DW_8b10b_dec
    generic map (bytes => inst_bytes,   k28_5_only => inst_k28_5_only,
                 en_mode => inst_en_mode,   init_mode => inst_init_mode,
                 rst_mode => inst_rst_mode, op_iso_mode => inst_op_iso_mode )
    port map (clk => inst_clk,   rst_n => inst_rst_n,
              init_rd_n => inst_init_rd_n,   init_rd_val => inst_init_rd_val,
              data_in => inst_data_in,   error => error_inst,  rd => rd_inst,
              k_char => k_char_inst,   data_out => data_out_inst,
              rd_err => rd_err_inst,   code_err => code_err_inst,
              enable => inst_enable, rd_err_bus => rd_err_bus_inst,
           code_err_bus => code_err_bus_inst );
end inst;

-- Configuration for use with VSS simulator
-- pragma translate_off
configuration DW_8b10b_dec_inst_cfg_inst of DW_8b10b_dec_inst is
  for inst
  end for; -- inst
```

```
end DW_8b10b_dec_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_8b10b_dec_inst(inst_clk, inst_rst_n, inst_init_rd_n,
                         inst_init_rd_val, inst_data_in, error_inst,
                         rd_inst, k_char_inst, data_out_inst,
                         rd_err_inst, code_err_inst, inst_enable,
                  rd_err_bus_inst, code_err_bus_inst );
  parameter inst_bytes = 2;
  parameter inst_k28_5_only = 0;
  parameter inst_en_mode = 0;
  parameter inst_init_mode = 1;
  parameter inst_rst_mode = 0;
  parameter inst_op_iso_mode = 0;

  input inst_clk;
  input inst_rst_n;
  input inst_init_rd_n;
  input inst_init_rd_val;
  input [inst_bytes*10-1 : 0] inst_data_in;
  output error_inst;
  output rd_inst;
  output [inst_bytes-1 : 0] k_char_inst;
  output [inst_bytes*8-1 : 0] data_out_inst;
  output rd_err_inst;
  output code_err_inst;
  input inst_enable;
  output [inst_bytes-1 : 0] rd_err_bus_inst;
  output [inst_bytes-1 : 0] code_err_bus_inst;

  // Instance of DW_8b10b_dec
  DW_8b10b_dec #(inst_bytes, inst_k28_5_only, inst_en_mode,
                 inst_init_mode, inst_rst_mode, inst_op_iso_mode)
    U1 (.clk(inst_clk),   .rst_n(inst_rst_n),   .init_rd_n(inst_init_rd_n),
        .init_rd_val(inst_init_rd_val),   .data_in(inst_data_in),
        .error(error_inst),   .rd(rd_inst),   .k_char(k_char_inst),
        .data_out(data_out_inst),   .rd_err(rd_err_inst),
        .code_err(code_err_inst),   .enable(inst_enable),
     .rd_err_bus(rd_err_bus_inst), .code_err_bus(code_err_bus_inst) );
endmodule
```

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com