Example Design: Generating the ALTECC_ENCODER and ALTECC_DECODER with DCFIFO

This design example uses an DCFIFO to illustrate how the ECC feature can be implemented external to the FIFO. The ALTECC_ENCODER and ALTECC_DECODER IP cores are required as the ALTECC_ENCODER IP core encodes the data input before writing the data into the DCFIFO, while the ALTECC_DECODER IP core decodes the data output from the DCFIFO before transferring the data out to other parts of the logic.

In this design example, the raw data width is 32 bits and is encoded by the ALTECC_ENCODER IP core block to produce a 39-bit width data that is written into the DCFIFO when write-enable signal is asserted.

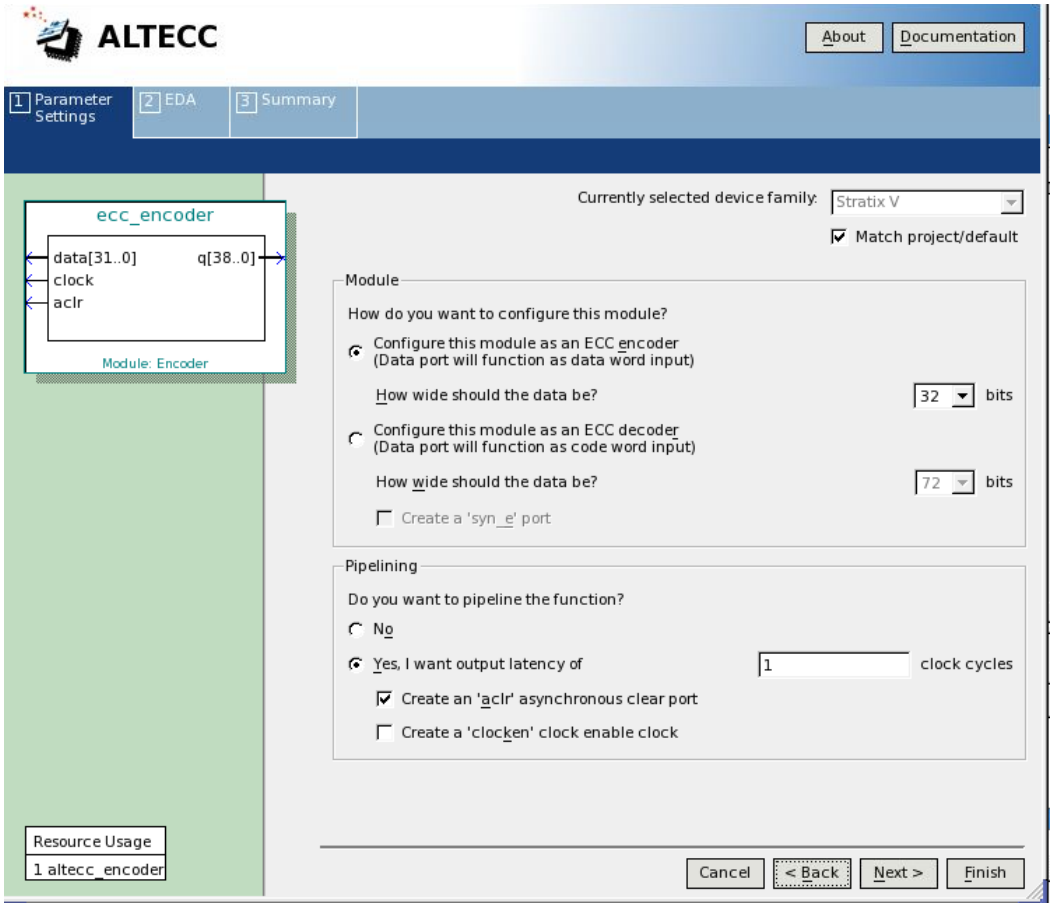
ALTECC_DECODER block is also implemented read port of the DCFIFO. When the read-enable signal is asserted, the encoded data is read from the DCFIFO and decoded by the ALTECC_DECODER block. The decoder shows the status of the data as no error detected, single-bit error detected and corrected, or fatal error (more than 1-bit error).

This example also includes two "corrupt bit" control signals at the DCFIFO. When the signals are asserted, they can change two LSB bits of encoded data before it is written into the DCFIFO. These signals are used to corrupt the two LSB of the data storing and examine the effect of the ECC features.

This design example describes how ECC features can be implemented with the DCFIFO which the ECC is not supported internally by the FIFO in Stratix V. However, the design example might not represent the optimized design or implementation.

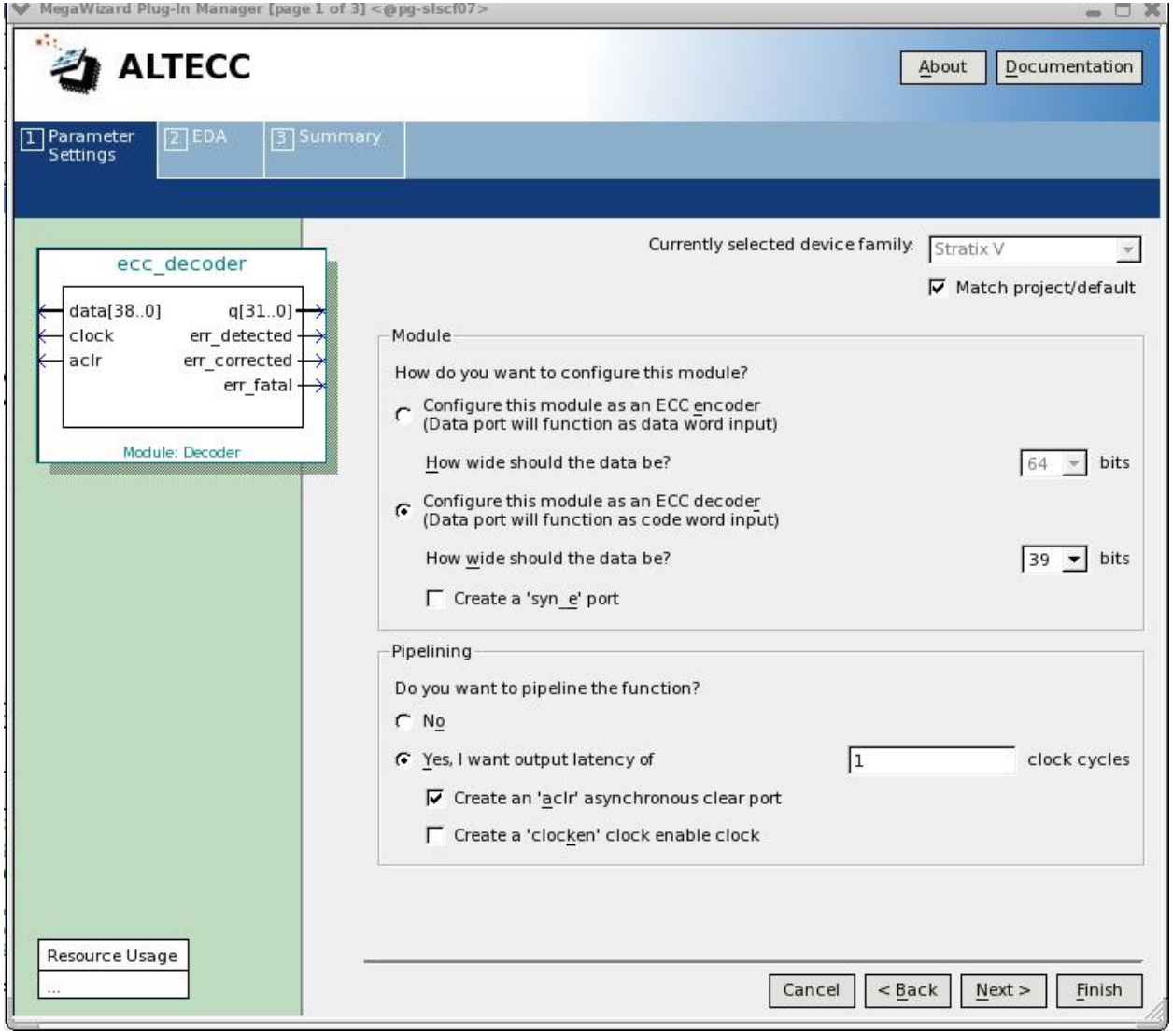
To generate the ALTECC_ENCODER and ALTECC_DECODER with the DCFIFO IP core, follow these steps:

1. Open the DCFIFO_ECC_ED.zip file and extract dcfifo_ecc_top_ed.qar.
2. In the Quartus II software, open the dcfifo_ecc_top_ed.qar file and restore the archive file into your working directory.
3. In the IP Catalog (Tools > IP Catalog), locate and double-click the ALTECC IP core. The parameter editor appears.
4. Specify the following parameters:



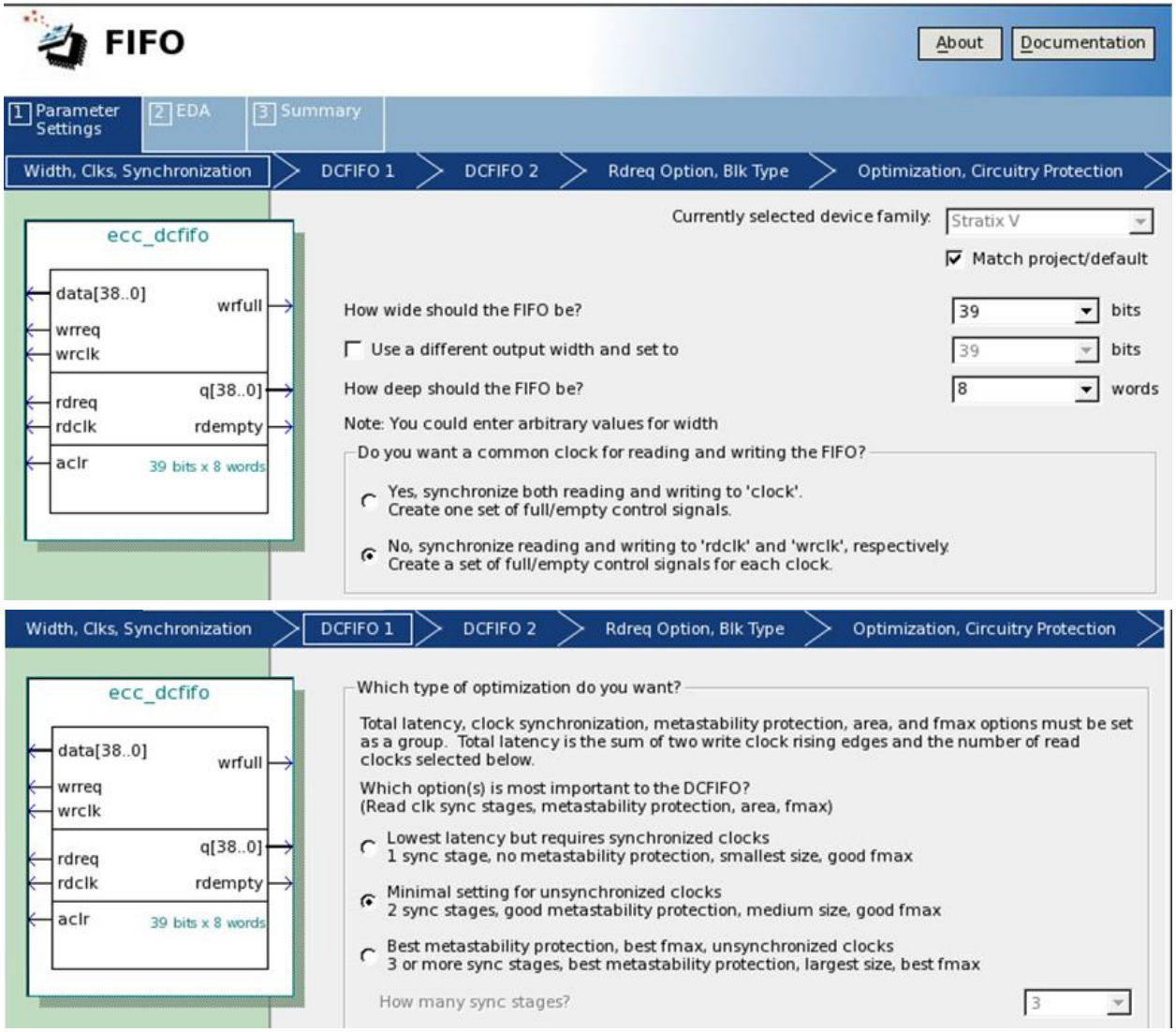
5. Click Finish. The ecc_encoder.v module is built.

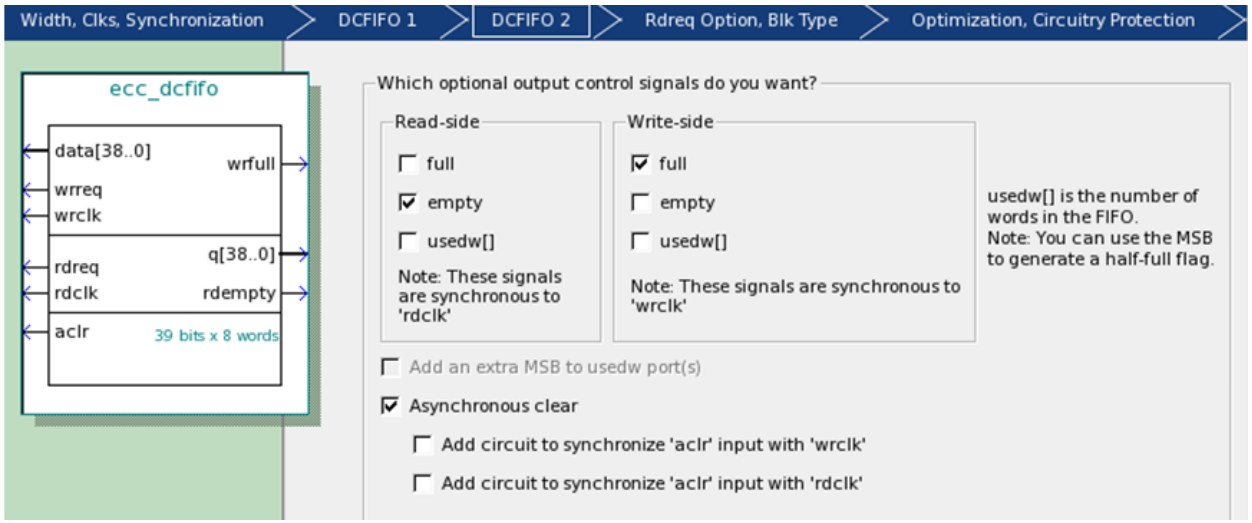
6. In the IP Catalog double-click the ALTECC IP core. The parameter editor appears.
7. Specify the following parameters:

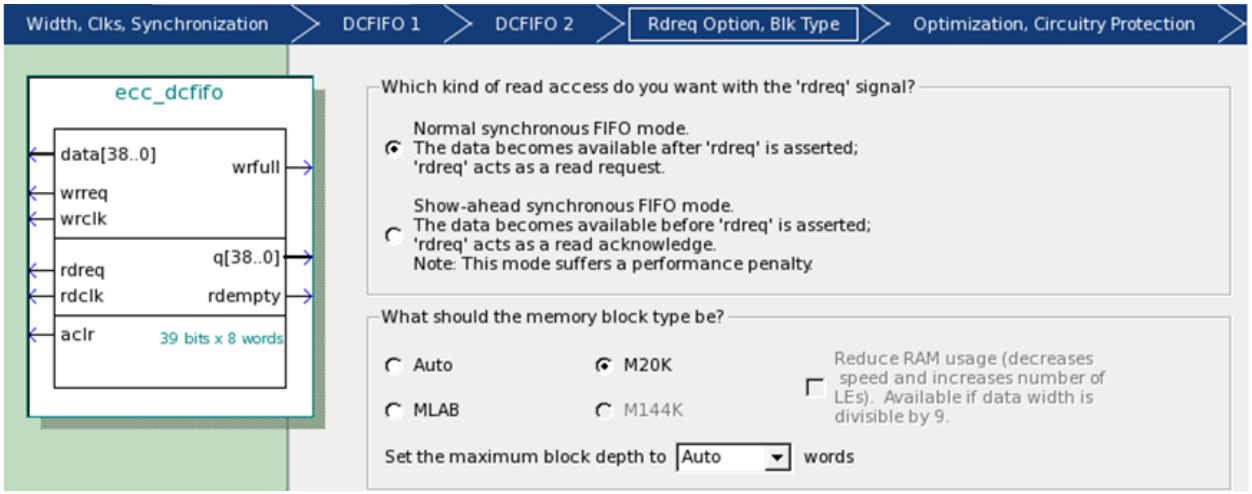


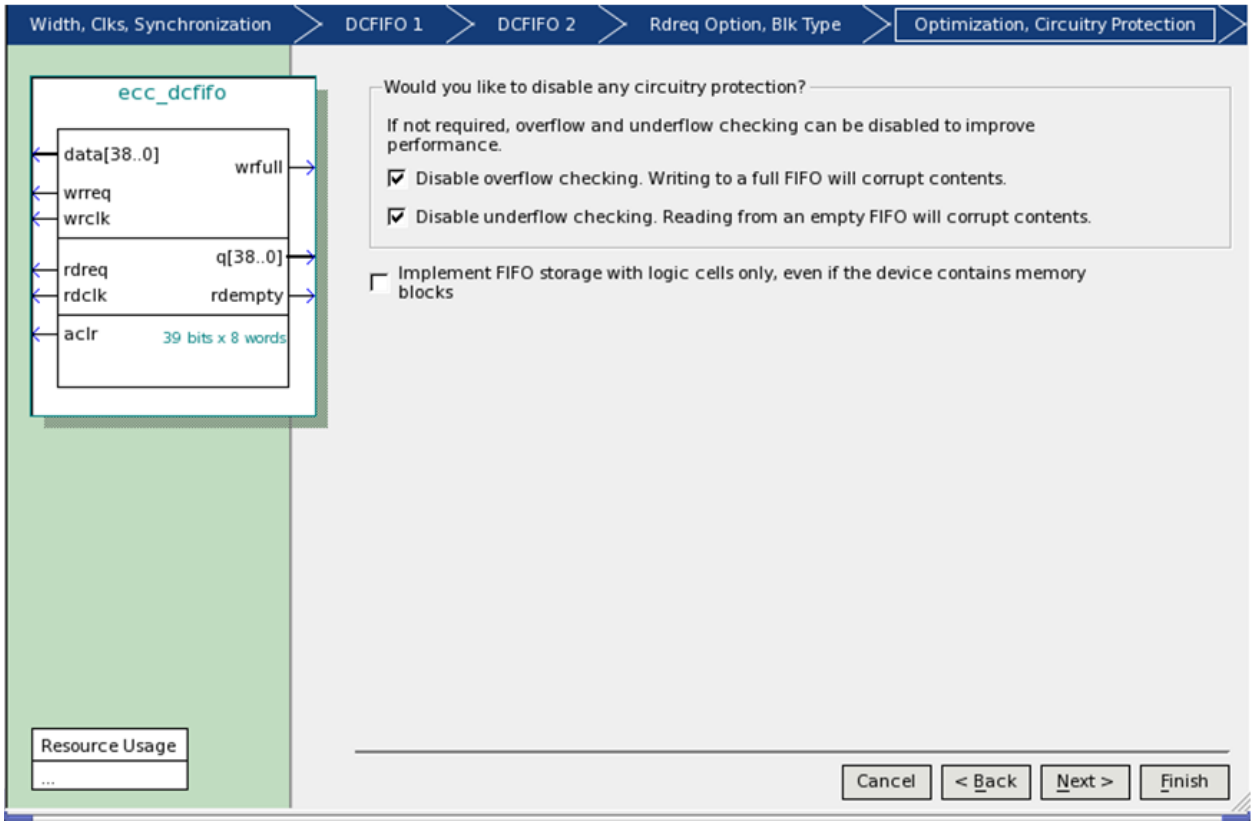
8. Click Finish. The ecc_decoder.v module is built.

9. In the IP Catalog double-click the FIFO IP core. The parameter editor appears.
10. Specify the following parameters:




Width, Clks, Synchronization
DCFIFO 1
DCFIFO 2
Rdreq Option, Blk Type
Optimization, Circuitry Protection
ecc_dcfifo
data[38..0]
wrfull
wrreq
wrclk
q[38..0]
rdreq
rdclk
rdempty
aclr
39 bits x 8 words
Which optional output control signals do you want?
Read-side
full
empty
usedw[]
Note: These signals are synchronous to 'rdclk'
Write-side
full
empty
usedw[]
Note: These signals are synchronous to 'wrclk'
usedw[] is the number of words in the FIFO.
Note: You can use the MSB to generate a half-full flag.
Add an extra MSB to usedw port(s)
Asynchronous clear
Add circuit to synchronize 'aclr' input with 'wrclk'
Add circuit to synchronize 'aclr' input with 'rdclk'

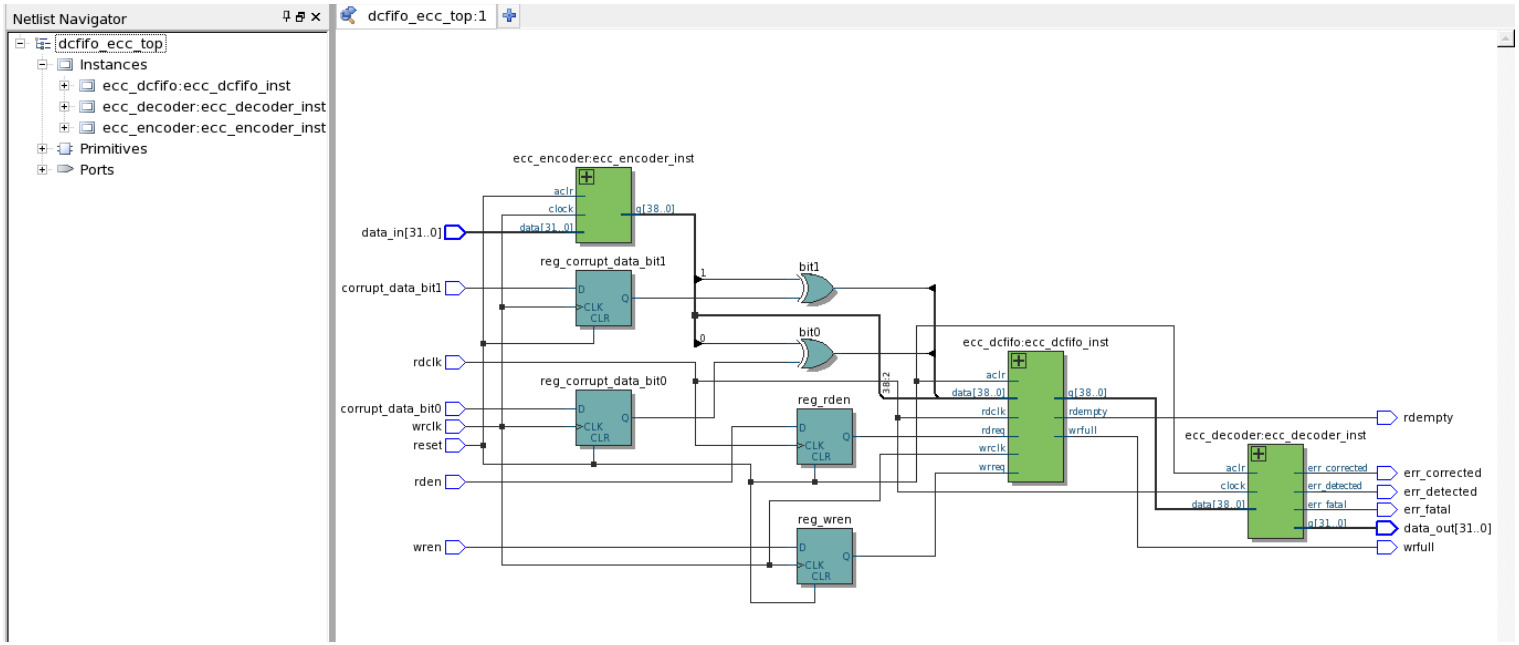

Width, Clks, Synchronization
DCFIFO 1
DCFIFO 2
Rdreq Option, Blk Type
Optimization, Circuitry Protection
ecc_dcfifo
data[38..0]
wrfull
wrreq
wrclk
q[38..0]
rdreq
rdclk
rdempty
aclr
39 bits x 8 words
Which kind of read access do you want with the 'rdreq' signal?
Normal synchronous FIFO mode.
The data becomes available after 'rdreq' is asserted;
'rdreq' acts as a read request.
Show-ahead synchronous FIFO mode.
The data becomes available before 'rdreq' is asserted;
'rdreq' acts as a read acknowledge.
Note: This mode suffers a performance penalty.
What should the memory block type be?
Auto
M20K
MLAB
M144K
Reduce RAM usage (decreases speed and increases number of LEs). Available if data width is divisible by 9.
Set the maximum block depth to
Auto
words



11. Click Finish. The ecc_dcfifo.v module is built.

The dcfifo_ecc_top.v is the top level file that instantiates a encoder, a DCFIFO and a decoders. An error injection logic has been included in this design to corrupt a single or double data bits. This is to examine the of ECC feature for error correction and detection. Altera's ALTECC_DECODER megafunction can perform Single Error Correction Double Error Detection (SECDED). To simulate the design, a testbench, tb.v is created for you to run in the ModelSim-Altera software.

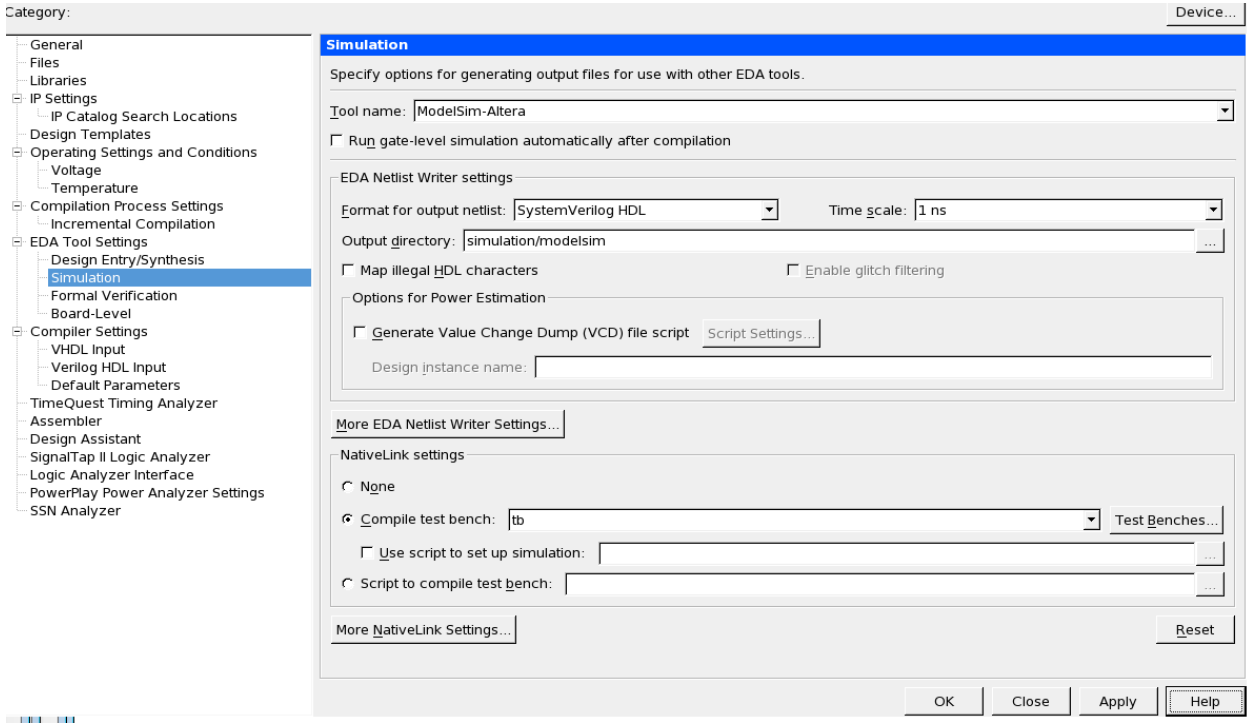
RTL Design:



Simulating the Design

To simulate the design in the ModelSim-Altera software, follow these steps:

1. Go to Assignment → Settings. Under EDA Tool Settings → Simulation, configure the parameter settings as below:



2. Perform the simulation run: Tools → Run Simulation Tool → RTL Simulation

Simulation Result:

WRCLK: Freq = 200MHz, Clock Period = 5ns

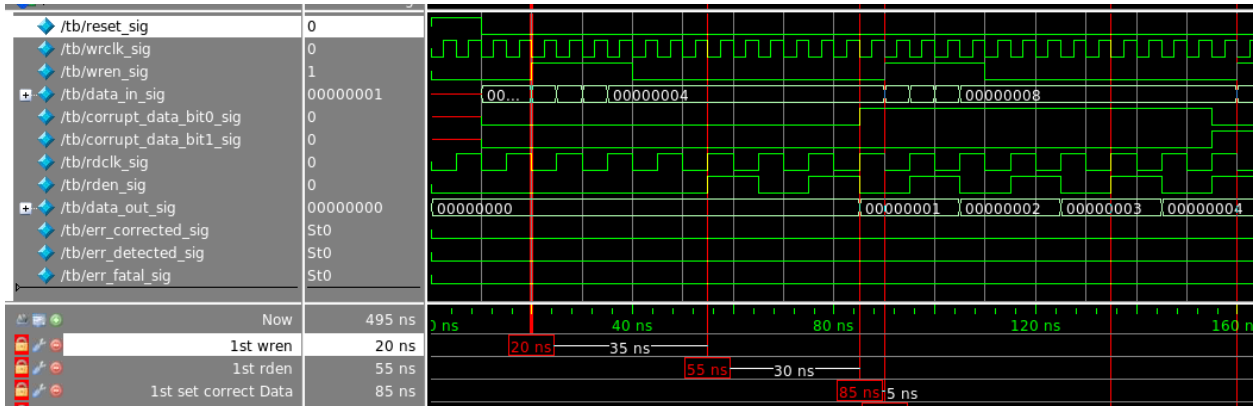
RDCLK: Freq = 100MHz, Clock Period = 10ns

10ns: reset signal de-asserts

1st Wr/Rd: No data corrutption.

20ns: wren signal asserts, writing 1st set of data: 0x00000001, 0x00000002, 0x00000003, 0x00000004.

55ns: rden signal asserts, reading first set of data. No error is detected.

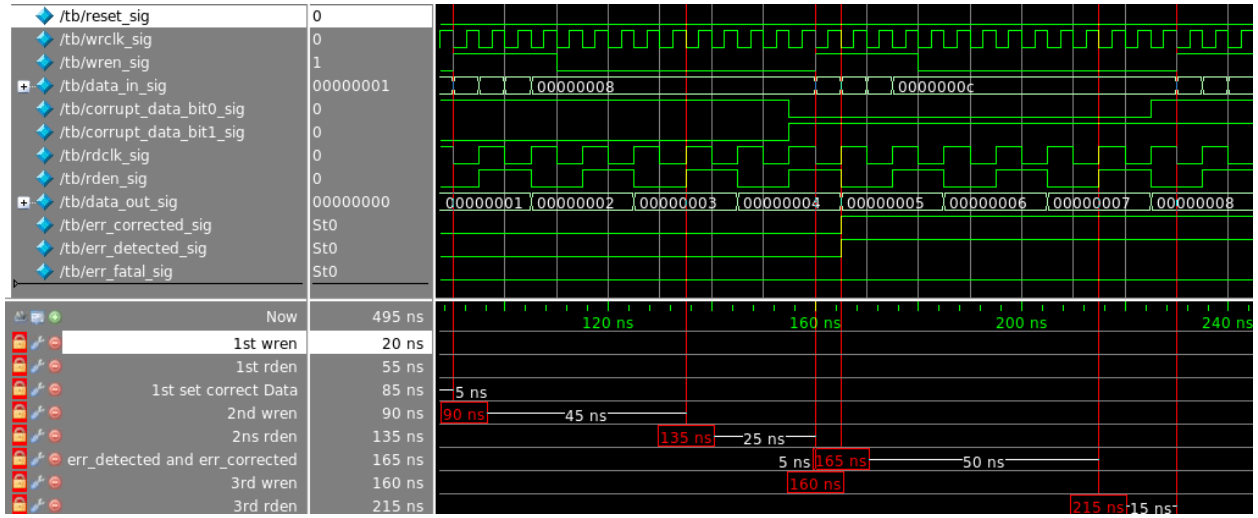


2nd Wr/Rd: Bit [0] of the data is corrupted.

90ns: wren signal asserts, writing 2nd set of data: 0x00000005, 0x00000006, 0x00000007, 0x00000008. Bit[0] of the data is corrupted.

135ns: rden signal asserts, reading second set of data.

165ns: err_detected and err_corrected signals assert. Data_out show corrected data.

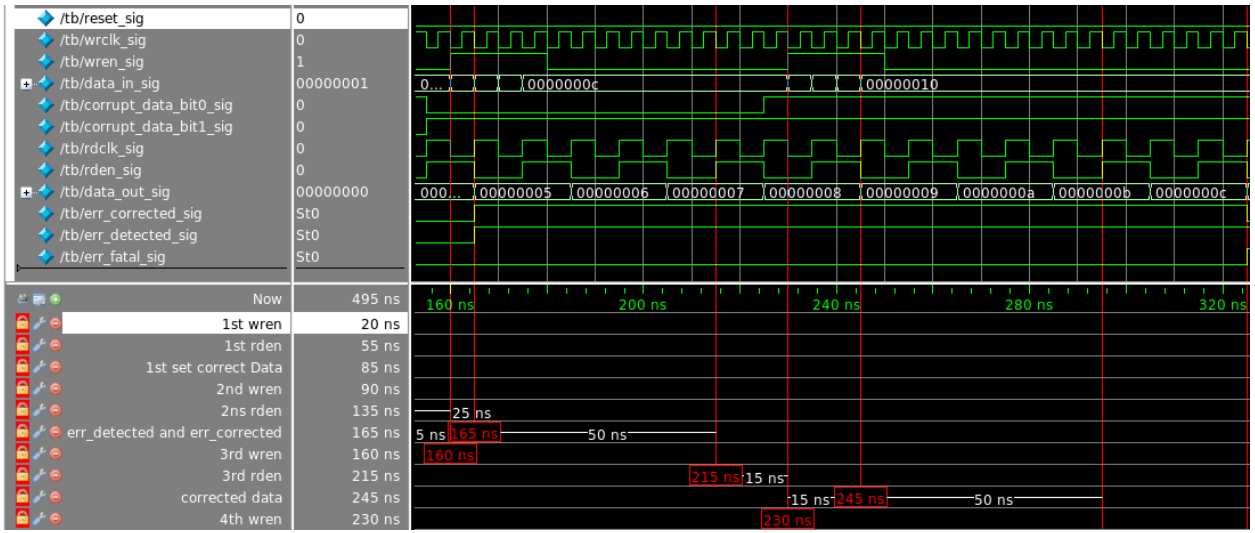


3rd Wr/Rd: Bit[1] of the data is corrupted.

160ns: wren signal asserts, writing 3rd set of data: 0x00000009, 0x0000000a, 0x0000000b, 0x0000000c. Bit[1] of the data is corrupted.

215ns: rden signal asserts, reading third set of data.

245ns: err_detected and err_corrected signals assert. Data_out show corrected data.



4th Wr/Rd: Both bit[0] and bit[1] of the data are corrupted.

230ns: wren signal asserts, writing 4th set of data: 0x0000000d, 0x0000000e, 0x0000000f, 0x00000010.

295ns: rden signal asserts, reading forth set of data.

325ns: err_detected and err_fatal signals assert. Data_out show uncorrected data.