HB0143
Handbook
CoreEDAC v2.10

Microsemi.

Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

50200143. 11.0 10/17

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 11.0

Updated changes related to CoreEDAC v2.10.

1.2 Revision 10.0

Updated changes related to CoreEDAC v2.9.

1.3 Revision 9.0

Updated changes related to CoreEDAC v2.8.

1.4 Revision 8.0

Updated changes related to CoreEDAC v2.7.

1.5 Revision 7.0

Updated changes related to CoreEDAC v2.6.

1.6 Revision 6.0

Updated changes related to CoreEDAC v2.5.

1.7 Revision 5.0

Updated changes related to CoreEDAC v2.4.

1.8 Revision 4.0

Updated changes related to CoreEDAC v2.3.

1.9 Revision 3.0

Updated changes related to CoreEDAC v2.2.

1.10 Revision 2.0

Updated changes related to CoreEDAC v2.1.

1.11 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CoreEDAC v2.0.

2 Overview

CoreEDAC produces Microsemi® field programmable gate array (FPGA)-optimized error detection and correction (EDAC) logic based on user-defined parameters. For ease of use, the core enables generation of logic integrated with on-chip RAM.

In space applications, storage elements such as static random access memory (SRAM) are susceptible to soft (transient) errors caused by heavy ions. Errors can be detected and corrected by employing error correction codes (ECCs). ECCs incorporate redundancy in the user data by forming codewords. With this redundancy, only a subset of all possible codewords contains valid messages. Valid codewords are separated from each other, so that errors are not likely to corrupt one valid codeword into another. The RAM protected by the EDAC stores the codewords. The capacity of the storage device must accommodate the data and the incorporated redundancy.

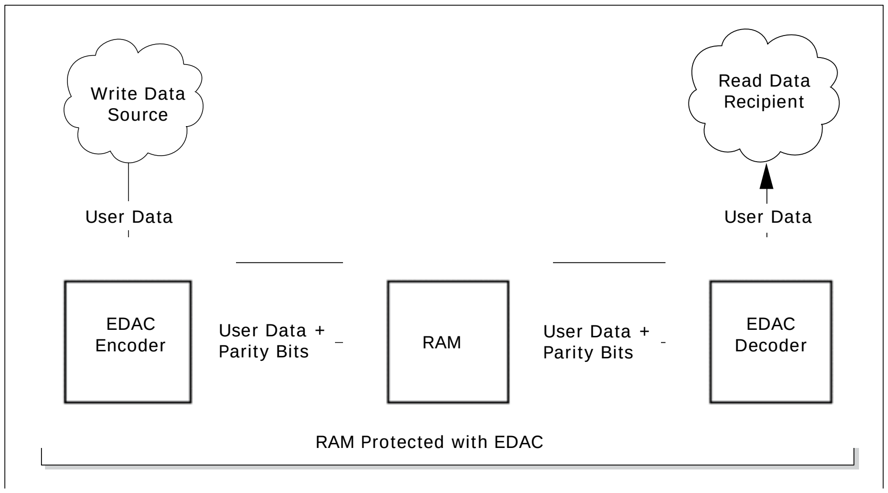
While recovering the data, a decoder first determines if a message read from the RAM is valid. This step is called error detection. If an error is detected, the decoder finds a valid message that is similar to the one that is read and corrects the error.

Theoretically, it is possible to detect and correct an arbitrary number of errors. Practically, a large number of errors require a larger redundant bit number and more complex encoder and decoder circuitry, which results in longer encoder and decoder latency. Latency is a crucial characteristic for the RAM. EDAC circuitry cannot use ECC such as Reed-Solomon, more efficient in terms of redundancy, because it introduces extreme latency. The error correction in RAM traditionally implements the single error correction double error detection (SECDED) technique based on Hamming code. This technique provides the best characteristics in terms of redundancy bits, the die area used, and the shortest encoder or decoder latency.

As memory density increases, a single-ion impact can cause multiple-bit upsets in nearby cells. Though SECDED cannot correct multiple errors, it is quite safe to use it with the Microsemi FPGAs, since these are well protected against multiple-bit upsets by design. Microsemi SRAM blocks are designed so that two bits of the same RAM word are never stored in nearby cells, whether in horizontal or vertical direction. Therefore, the probability of multiple-bit upset in SRAM blocks is negligible, and the SECDED technique proves to be highly efficient.

CoreEDAC is based on a special shortened Hamming code proposed by Hsiao. This code provides better latency and area characteristics than other Hamming codes.

Figure 1 • **Protected RAM Example**



An example of a RAM protected with EDAC is as shown in Figure 1. During user write, user data comes to the EDAC encoder, which calculates the parity bits and appends these to the user data, forming a codeword. The codeword is stored in the RAM.

During user read, the read codeword first comes to the decoder, which detects and corrects errors (if any), discards parity bits, and outputs the corrected user data word.

The core can generate EDAC circuitry for both internal (on-chip) and external RAM blocks. For the internal RAM, CoreEDAC provides an integrated solution that includes the EDAC circuitry, RAM, and all necessary connections between the two. From a user's perspective, the solution looks like a configurable RAM block capable of detecting and correcting errors. This handbook often uses the acronym EDAC to name the EDAC logic, along with the RAM included in the integrated solution. It only differentiates the two wherever it is appropriate.

2.1 Key Features

Following are the key features of CoreEDAC:

- Parameterizable RTL generator
- Modes of operation:
    - EDAC with internal RAM. EDAC RAM generation with optional background scrubbing circuitry
    - EDAC encoder and decoder generation. The mode can be used to apply EDAC encoder and decoder to external memories.
- Flexible user data size from 4 to 64 bits. This corresponds to a codeword size from 8 to 72 bits
- User-defined pipeline options to enhance EDAC throughput
- Parameterizable refresh (scrubbing) rate
- Improved latency and area characteristics
- Correctable and error flags
- Option to suppress write-back during the scrubbing session
- Optional triple EDAC redundancy

2.2 Supported Families

The following families are supported in this version:

- PolarFire™
- RTG4™
- SmartFusion®2
- IGLOO®2
- SmartFusion®
- IGLOO®
- IGLOO®e
- IGLOO® PLUS
- Fusion®
- ProASIC®3
- ProASIC®3E
- ProASIC®3L
- Axcelerator®
- RTAX-S/SL and RTAX-DSP

2.3 Core Version

This handbook supports CoreEDAC v2.10.

2.4 Utilization and Performance

CoreEDAC has been implemented in Microsemi SmartFusion2, RTG4, RTAX-S, and ProASIC3L devices. The core benchmarks are listed in Table 1 to Table 6. Test configurations are listed in Table 8.

Table 1 • **Utilization and Performance for a Common Read and Write Clocks**

	Pipelines		**Cells**					
Data Width	**Decoder**	**RAM**	**Sequential**	**Combinational**	**Total**	**%**	**RAM Blocks**	**Maximum Clock Rate (MHz)**
Device: RTAX1000S, Speed Grade –1								
12	0	0	200	223	432	2.3%	5	77
12	1	1	266	237	503	2.8%	5	124
16	0	0	216	242	458	2.5%	6	77
16	2	0	291	282	573	3.2%	6	109
32	0	0	306	366	672	3.7%	10	66
32	2	0	446	440	886	4.9%	10	86
64	0	0	475	596	1,071	5.9%	18	58
64	2	1	796	776	1,572	8.7%	18	79
Device: A3P1000L, Speed Grade –1								
12	0	0	117	355	472	1.9%	4	71
12	1	1	164	372	536	2.2%	4	97
16	0	0	126	456	582	2.4%	8	50
16	1	1	178	464	642	2.6%	8	92
32	0	0	160	633	793	3.2%	12	51
32	1	1	230	662	892	3.6%	12	80
64	0	0	225	773	998	4.0%	16	49
64	2	1	435	930	1,365	5.5%	16	91

Note: Data is achieved using typical synthesis and layout settings.

Table 2 • **A3P1000L Utilization and Performance for Independent Read and Write Clocks**

	Pipelines		Cells					RAM Blocks	Maximum Clock Rate (MHz)
Data Width	Decoder	RAM	Sequential	Combinational		Total	%		
12	0	0	132	362	494	2.0%	4	142	71
12	1	1	181	384	565	2.3%	4	152	83
16	0	0	141	462	603	2.4%	8	102	60
16	1	1	195	474	669	2.7%	8	134	81
32	0	0	176	640	816	3.3%	12	110	53
32	1	1	248	675	923	3.7%	12	111	80
64	0	0	239	783	1,022	4.1%	16	89	49
64	2	1	455	938	1,393	5.6%	16	93	82

Note: Data is achieved using typical synthesis and layout settings.

Table 3 • **RTAX1000S Utilization and Performance for Independent Read and Write Clocks**

Data Width	Pipelines		Cells				RAM Blocks	Max Clock Rate (MHz)	
	Decoder	RAM	Sequential	Combinatorial	Total	%		Read	Write
12	0	0	289	274	563	3.1	5	79	115
12	1	1	363	321	684	3.8	5	131	131
16	0	0	317	302	619	3.4	6	72	101
16	2	0	410	363	773	4.3	6	119	117
32	0	0	432	431	863	4.8	10	63	78
32	2	0	579	527	1,106	6.1	10	111	93
64	0	0	668	641	1,309	7.2	18	58	68
64	2	1	989	856	1,845	10.2	18	71	69

Table 4 • **Common Read and Write Clocks for M2S025 Device at Speed Grade – 1 and COM conditions**

Data Width	Pipelines			RAM 1K18	Other Resource Utilization		Maximum Clock Rate (MHz)
	Decoder	Encoder	RAM		4-LUT	DFF	
12	1	0	1	1	358	257	375
16	2	0	1	2	432	356	357
32	3	1	1	3	631	526	361
64	3	1	1	4	938	783	350

Note: Data is obtained using High Effort Five Passes Layout configuration options.

Table 5 • **Common Read and Write Clocks for M2S025T Device at Speed Grade – 1 and MIL conditions**

Data Width	Pipelines			RAM 1K18	Other Resource Utilization		Maximum Clock Rate (MHz)
	Decoder	Encoder	RAM		4-LUT	DFF	
12	1	0	1	1	358	257	300
16	2	0	1	2	432	356	297
32	3	1	1	3	631	526	300
64	3	1	1	4	938	783	300

Note: Data is obtained using High Effort Five Passes Layout configuration options.

Table 6 • **Common Read and Write Clocks for RT4G150 Device at Speed Grade – 1 and MIL conditions**

Data Width	Pipelines			RAM 1K18	Other Resource Utilization		Maximum Clock Rate (MHz)
	Decoder	Encoder	RAM		4-LUT	DFF	
12	1	0	1	1	357	252	221
16	2	0	1	2	432	356	221
32	3	1	1	3	615	529	203
64	3	1	1	4	938	782	201

Note: Data is obtained using High Effort Five Passes Layout configuration options.

Table 7 • **Common Read and Write Clocks for MPF300 Device at speed Grade -1 and MIL conditions**

Data Width	Pipelines			RAM 1K18	Other Resource Utilization		Maximum Clock Rate (MHz)
	Decoder	Encoder	RAM		4-LUT	DFF	
12	1	0	1	1	233	187	231.1
16	2	0	1	2	293	283	231.1
32	3	1	1	3	372	390	231.1
64	3	1	1	4	597	614	231.1

Table 8 • **CoreEDAC Test Configuration**

Parameter	**Value**
RAM depth, word	1,024
Scrubbing mode	On
Wirteback mode	On
Number of encoder pipelines	0
Scrubbing period binary divider bit width	10
Scrubbing period arbitrary divider bit width	17
Scrubbing range	0 – 1,023
Generate delayed copy of the read address	Off

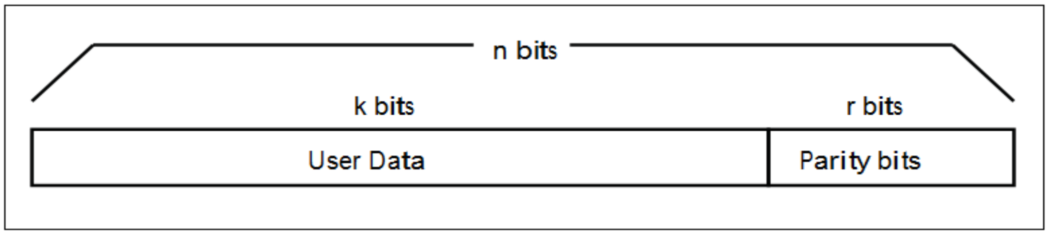
3 Operation

3.1 SECDED Algorithm

Hamming code is a linear error-correcting code. It means that every possible k-bit word can be encoded. Hamming codes can detect and correct single-bit errors. The codeword is guaranteed to be corrected if the Hamming distance[1] between the original and corrupted words is two or more.

Alternatively, Hamming code can detect (but not correct) up to two simultaneous bit errors. Because of the simplicity of Hamming codes, they are widely used in computer memory. Figure 2 shows the Hamming code structure.

Figure 2 • **Hamming Code Structure**



Parameters of the maximal length Hamming code (n, k) relate as follows:

$n = 2^r - 1$

$k = n - r = 2^r - r - 1$

SECDED code has an extra parity bit to increase the minimal Hamming distance from 3 to 4. To support the flexible data bit width, shortened Hamming codes are used in SECDED. A shortened codeword contains fewer data bits than the maximal length code. The shortened codeword keeps the same number of parity bits to correct one and detect two errors. Therefore, the number of data bits in the shortened code is reduced by the same amount as the overall codeword length.

Hamming encoder and decoder implementation uses a number of wide (multi-input) XOR gates. The wide gates not only consume the die area but also introduce a certain amount of latency because they are built as multi-layer tree logic. The gate input count primarily depends on the user data bit width, k. Given the k width, XOR input count can be minimized by a clever selection of shortened Hamming codes. CoreEDAC implements the Hsiao code, optimized for the least XOR input count.

Table 9 shows the Hsiao codec ECC parameters (n and k).

Table 9 • **SECDED (n, k) Code Parameters**

User Data Bitwidth, k	**Parity Bits, n-k**	**Codeword Bitwidth, n**
4	4	8
5–11	5	10–16
12–26	6	18–32
27–57	7	34–64
58–64	8	66–72

Note: The Hamming distance is the number of positions for which the corresponding bits are different.

Table 10 shows the maximum user data widths that can be accommodated by various RAM blocks.

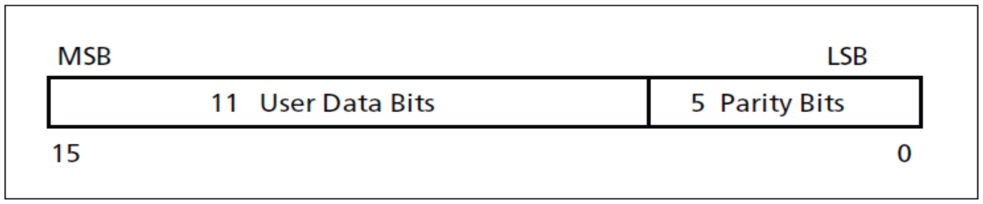
Table 10 • **Maximum User Word Sizes Fault-Tolerant RAM Accommodates**

RAM Bit Width	Maximum User Data Bit Width
9	4
18	12
27	21
36	29
54	47
72	64

Hamming ECC adopted by the core can correct a single-bit error and detect a double-bit error per word. The core generates two flags that identify a number of erroneous bits detected and/or corrected. Once the number exceeds two per word, the algorithm may yield incorrect flags.

Figure 3 shows the arrangement of the parity and data bits in a 16-bit example of the codeword.

Figure 3 • **Parity and Data Bits Arrangement in a Codeword**



3.2 Scrubbing

The section states that the Microsemi RAM design precludes multi-bit upsets from single-ion impact. Still, multiple impacts, given enough time, can cause uncorrectable errors if they hit a nearby area. To prevent the RAM from collecting soft errors over extended time periods, it is important to check each memory location periodically, before the next impact is likely to happen. This is accomplished by using the scrubbing process.

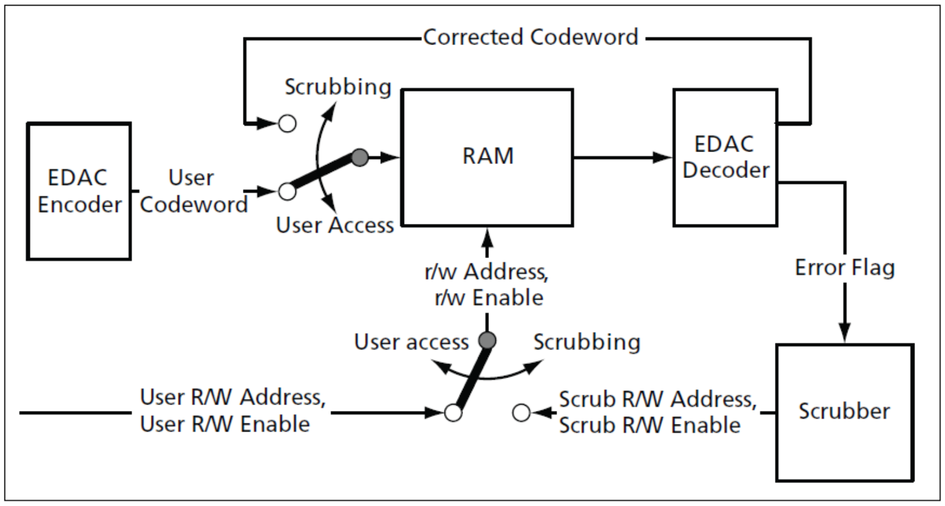
Scrubbing periodically checks every memory location using the ECC decoder. If a particular location contains a corrupted word, the decoder detects and corrects the word. The scrubbing circuitry then writes the corrected word back to the same location.

To provide a normal access to the RAM and prevent decreasing performance, scrubbing is only done during idle periods. Since scrubbing consists of regular read and write operations, it may increase power consumption. Therefore, scrubbing is not done continuously, but periodically. Usually, the scrubbing refresh period—the time interval between two consecutive scrubbing sessions—is much longer than the session itself.

A simplified block diagram of the scrubbing circuitry that complements the EDAC protected memory is shown in Figure 4. Once the scrubbing starts, it acts as another memory user, providing the RAM with read address and read enable signals through multiplexers (MUXes). During the scrubbing process, the MUXes disconnect user signals such as user read and write addresses, user codeword, and user read and write enable. The EDAC decoder reads the codewords from the RAM one by one. If the EDAC decoder detects an error, it corrects the codeword and raises an internal error flag.

As a result, the scrubbing circuitry sets a proper write address and write-enable signals, writing the corrected codeword back to the RAM. Writeback occurs only upon detecting an error.

Figure 4 • **EDAC with Scrubbing**



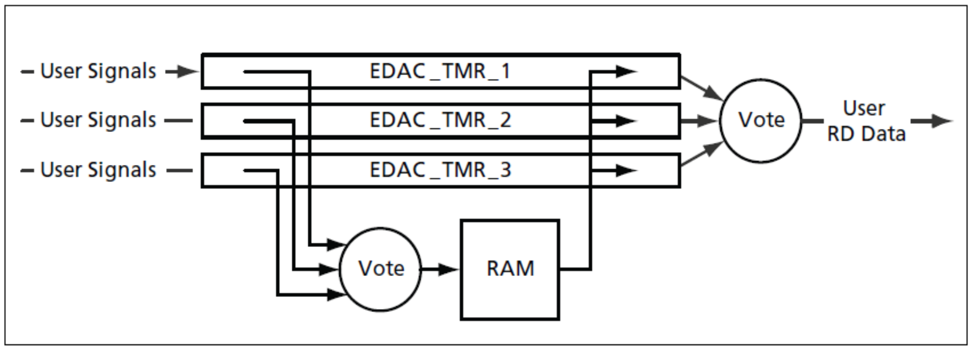
3.3 Triple-Modular Redundancy

As the EDAC circuitry occupies certain die area, the circuitry itself can be hit by a heavy ion and it is susceptible to soft errors. Normally, such errors are not likely to impact the user data, but some of the most demanding applications may require additional protection. CoreEDAC supports optional triple-module redundancy (TMR). TMR is a fault-tolerant technique, in which three parallel systems perform a process and three process results are processed by a voting system to produce a single result. If any one of the three systems fails, the other two can correct and mask the fault.

When the TMR option is enabled, the EDAC encoder, decoder, and scrubber are replicated three times, and appropriate majority vote logic is inserted. Figure 5 shows a simplified block diagram of the EDAC TMR. User signals, such as read or write address, read or write enable, write data, and scrub control, come to the three EDAC instances. Every instance implements the complete EDAC logic. The enabled TMR option triplicates the amount of resources utilized.

Every EDAC instance generates signals to control the RAM block. Voting logic merges the three sets into a single set that actually controls the RAM. The RAM read data output results in the three EDAC instances, where every instance decodes the same data. Voting logic again merges the three decoded results into a single user RD Data set. The TMR option is disabled on RTG4 family where the triple redundancy along with voting logic is built in every FPGA fabric component.

Figure 5 • **EDAC TMR**



3.4 Internal or External RAM

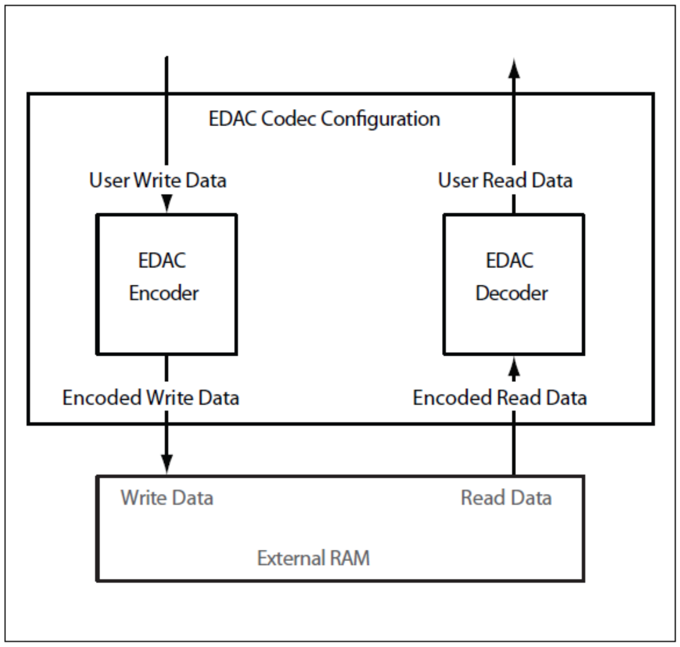
Depending on needs, CoreEDAC can generate two basic EDAC configurations: Protected RAM and EDAC codec. The protected RAM integrated configuration delivers the configurable internal (on-chip) RAM block and connects it to the configurable EDAC circuitry. The latter includes the encoder and decoder, as well as an optional scrubber.

The core supports only a two-port RAM configuration with equal read and write data width. CoreEDAC also supports an optional built-in RAM read pipeline and both common and independent read and write clocks. In independent read and write clock signals, all write operations, including Hamming encoding, are performed in the write clock domain. Read operations, including Hamming decoding, are performed within a read clock domain. Scrubbing uses both the clock domains, as it performs both read and write operations.

Configured as the EDAC codec, the core generates the configurable encoder and decoder only, comprising the ECC codec. It is required to provide other signals necessary to write the encoded data to the external RAM, route the read data from the external RAM to the decoder input, and read the decoded data.

Figure 6 shows a simplified block diagram of the EDAC codec connected to the external RAM.

Figure 6 • **EDAC with External RAM**



3.5 EDAC Pipelines

Hamming encoder and decoder circuitry uses a number of wide (multi-input) XOR gates. While the Hsiao code minimizes the XOR input count, it is still high enough to impact the data rate. To improve the achievable data rate, CoreEDAC offers an optional EDAC logic pipelining. The core also enables you to optionally enable a read data pipeline register built in an on-chip RAM. Every inserted pipeline introduces a latency of one clock cycle, so pipelining can be enabled if a RAM application can tolerate the latency.

Specify the maximum tolerable latencies separately for the encoder and decoder. CoreEDAC analyzes the encoder and decoder structure and the implementation platform (FPGA family selected). The family influences a number of logic tree layers needed to implement a given wide XOR gate. Based on the analysis, CoreEDAC inserts only pipelines that actually improve the data rate.

Therefore, the actual number of pipelines can be either equal to or smaller than the maximum latency specified. CoreEDAC posts the actual latency values through a configuration window.

To achieve the highest data rate, it is necessary to conduct experiments while enabling pipeline insertion at a critical path.

4 Interface Description

4.1 Ports

Figure 7 shows the core I/O ports. The ports shown are a superset of all possible ports. In every given EDAC configuration, only a subset of the ports is used.

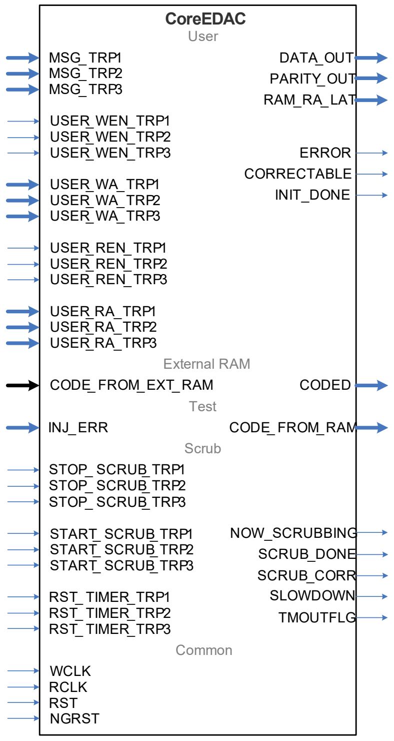
Some input ports are tripled to support the EDAC TMR. When the TMR option is disabled, input signals are to be supplied to the input pins with names ending in_TRP1. This section describes port connections with the TMR option disabled. The functionality of the complementary TMR ports with names ending in_TRP2 or _TRP3 is the same as that of the _TRP1 ports. These input ports provide access to the two redundant instances of the EDAC circuitry when the TMR option is enabled. For the RTG4 family, the TMR option is not available as the triple redundancy is built in every FPGA fabric component. Accordingly, for the RTG4 family all the ports ending in _TRP2 and _TRP3 are not available as well.

The write data, write address, and write enable come to the MSG_TRP1, USER_WA_TRP1, and USER_WEN_TRP1 ports respectively. The read address and read enable signal come to USER_RA_TRP1 and USER_REN_TRP1, respectively. Corrected read data appears at the DATA_OUT port and corrected parity bits appear on the PARITY_OUT port. The core provides optional matching delay for the user read address USER_RA_TRP1. With this option enabled, the core delays the read address so that it appears at the RAM_RA_LAT port, along with user read data DATA_OUT and PARITY_OUT.

CoreEDAC generates two flags, ERROR and CORRECTABLE, to report detected and corrected errors. The INIT_DONE flag marks completion of the optional RAM initialization.

When the core is used with external RAM, the user write data is supplied on the same MSG_TRP1 port. After encoding, the user data codeword appears at the CODED port of the core. CODE_FROM_EXT_RAM accepts codewords coming from the external RAM for decoding. The decoded data appear at the DATA_OUT port, while the parity bits appear on the PARITY_OUT port.

Figure 7 • **CoreEDAC Pinout**



Two ports, INJ_ERR and CODE_FROM_RAM, are provided to test the EDAC ability to detect and correct the errors. The artificial errors can be injected through the INJ_ERR port in the data path between the encoder and the RAM. By using the CODE_FROM_RAM output port, the codewords can be accessed directly from the RAM, bypassing the decoder. The testbench uses the direct RAM output to verify scrubbing if it is capable of correcting the errors.

The STOP_SCRUB_TRP1 input signal transfers full control over the RAM to the user access. The active STOP_SCRUB_TRP1 signal prevents scrubbing from start or interrupts the scrubbing that has already started.

START_SCRUB_TRP1 starts the scrubbing session, provided STOP_SCRUB_TRP1 is inactive. This optional signal can be used to launch an unscheduled scrubbing session. RST_TIMER_TRP1 resets a scrubbing timer that sets timing intervals between the scrubbing sessions. The signal can be used to resynchronize scheduled scrubbing refresh periods. The output flags, NOW_SCRUBBING, SCRUB_DONE, SLOWDOWN, and TMOUTFLG, provide feedback on the scrubbing status.

Finally, write and read clock signals are supplied on WCLK and RCLK. If a common read and write clock is used, this clock connects to the RCLK input port. The RST and NGRST ports accept synchronous and negative global asynchronous resets.

Table 11 contains more information on the EDAC ports.

Note: The ports ending in _TRP2 or _TRP3 are not available for the RTG4 family.

Table 11 • **CoreEDAC Ports**

Signal	Relevant TMR Inputs	In/Out	Description
MSG_TRP1	MSG_TRP2, MSG_TRP3	In	Data input word to be encoded and written in the RAM. The input is k bits wide.
USER_WEN_ TRP1	USER_WEN_ TRP2, USER_WEN_TRP3	In	Write enable signal. It marks a clock cycle when a user data is ready to be written. Signal polarity is positive.
USER_WA_ TRP1	USER_WA_ TRP2, USER_WA_ TRP3	In	Write address. The input is ceil(log$_2$(RAM depth)) bits wide. For example, at a RAM depth of 1200 words, the write address equals 11 bits. Write address bit width must be equal to the read address bit width.
USER_REN_ TRP1	USER_REN_ TRP2, USER_REN_TRP3	In	Read enable signal. Signal polarity positive.
USER_RA_ TRP1	USER_RA_ TRP2, USER_RA_ TRP3	In	Read address. The input is ceil(log$_2$(RAM depth)) bits wide. For example, at RAM depth of 1200 words, the read address equals 11 bits. Read address bit width must be equal to the write address bit width.
CODE_FROM_EXT_RAM	–	In	Optional read data input from the external RAM. The input is n bits wide. Used only with external RAM to decode the external RAM data.
INJ_ERR	–	In	Error input used for the test purpose. The input is n bits wide. Error code present on this input are XORed with the encoded data prior to be written to the RAM. The parameter/generic TEST has to be set to 1 to enable the port.
STOP_SCRUB_TRP1	STOP_SCRUB_TRP2, STOP_SCRUB_TRP3	In	Input to control scrubbing process. User asserts the signal when accessing the RAM. Scrubbing can only run when the signal is deasserted. Active high. **Note:** Valid only when SCRUB_ON=1. This should be set to 0 when it is not used.

Table 11 • **CoreEDAC Ports**

START_ SCRUB_TRP1	START_ SCRUB_TRP2, START_ SCRUB_TRP3	In	Optional input to start scrubbing process. Active high. **Note:** Valid only when SCRUB_ON=1. This should be set to 0 when it is not used.
RST_TIMER_ TRP1	RST_TIMER_ TRP2, RST_TIMER_ TRP3	In	Optional input to reset scrubbing timer. Active high. **Note:** Valid only when SCRUB_ON=1. This should be set to 0 when it is not used.
WCLK	–	In	Write clock signal. It is used when the core is configured for independent read/write clocks. If both read and write sides utilize a single-clock, the RCLK port is used to input the single-clock signal.
RCLK	–	In	Read clock signal. If both read and write sides utilize a single-clock, the RCLK is used as the read and write clock signal.
RST	–	In	Optional synchronous reset signal. Active high. The signal puts design in a valid initial state. In particular, all the generated flags become valid after the RST or NGRST signals are asserted and de-asserted. If the encoder or decoder pipeline registers are enabled, then the RST signal keeps the encoder or decoder output signals in state 0(zero). If independent read and write clocks are used, then the signal should last long enough to cover at least one period of both clocks. The length of the RST signal of two or more slower clock periods satisfies the condition. After deactivating the RST signal, the core takes up to 6 clock cycles to return to normal functionality. Actions of the RST and NGRST signals are the same except for an auto RAM initialization, which starts on NGRST when proper configuration (INIT_RAM=1) is selected. The RST signal does not start the auto initialization.
NGRST	–	In	Asynchronous reset. Active low. Once the asynchronous reset is complete, CoreEDAC state machine is set to a valid initial state. In particular, all the generated flags become valid after the RST or NGRST signals are asserted and de-asserted. The signal is also used as a sign of powering up the FPGA device. It is the only signal that starts the RAM initialization process when the core is configured for automatic RAM initialization (INIT_RAM = 1). You must use the NGRST signal when the core is configured for auto initialization mode. If the auto initialization is not required, then you can use the RST signal instead of NGRST.
DATA_OUT	–	Out	Corrected read data output. It is k bits wide. The output is synchronous with RCLK.
PARITY_OUT	–	Out	Parity bits of the corrected codeword. It is r bits wide. The output is synchronous with RCLK.

Table 11 • **CoreEDAC Ports**

CODE_FROM_RAM	–	Out	Direct RAM output is used to test scrubbing results. It is n bits wide. The k most significant bits (MSBs) of the signal are the decoded (corrected) data, the r least significant bits (LSBs) are the corrected parity bits. The output is synchronous with RCLK.
CODED	–	Out	Encoded user data to be written to the external RAM. The output is n (user data width+ parity width) bits wide.
RAM_RA_LAT	–	Out	Optional read address signal. It replicates the user read address delayed to match read data delay caused by ECC decoder and RAM output pipeline register. The signal bit width equals the bit width of the user_rA_trp1 signal. The output is synchronous with RCLK.
ERROR	–	Out	Optional error flag. Active high. The signal flags uncorrectable double errors detected during user read access. The signal is not registered. If a glitch-free flag is required, the signal needs to be registered at RCLK positive edge.
CORRECTABLE	–	Out	Optional correctable error flag. Active high. It flags correctable errors detected during user read access. The signal is not registered. If a glitch-free flag is required, the signal needs to be registered at RCLK positive edge.
INIT_DONE	-	Out	Optional flag. Active high. When INIT_RAM mode is enabled, the flag indicates completion of the initialization process. The flag is synchronous with WCLK.
NOW_SCRUBBING	-	Out	Optional flag. Active high. Indicates the scrubbing is in progress. The flag is synchronous with RCLK.
SLOWDOWN	–	Out	Optional flag. Active high. The core raises the flag when a scrubbing session is due, but has not completed by the time a scheduled scrubbing refresh period timed out. This happens due to extensive user access to the RAM, and the flag indicates the user should slow down user access. The flag is synchronous with RCLK.
TMOUTFLG	–	Out	Optional scrub timeout flag. Active high. It signals the scheduled scrubbing refresh period is about to expire and the scrubbing session is due. The output is synchronous with RCLK.
SCRUB_DONE	–	Out	Optional flag signal. Indicates a scrubbing session has ended. Active high. The output is synchronous with RCLK.
SCRUB_CORR	-	Out	Optional flag. Active high. Marks time intervals when the scrubber writes a corrected code back to the RAM; that is, actually corrects RAM contents. The flag is synchronous with WCLK.

4.2 Interface Examples

Figure 8 shows an example of an EDAC configuration that includes the RAM and the scrubber, but not TMR. Optional in/out signal names are in grey in Figure 8.

Figure 8 • **Protected RAM Interface Example**

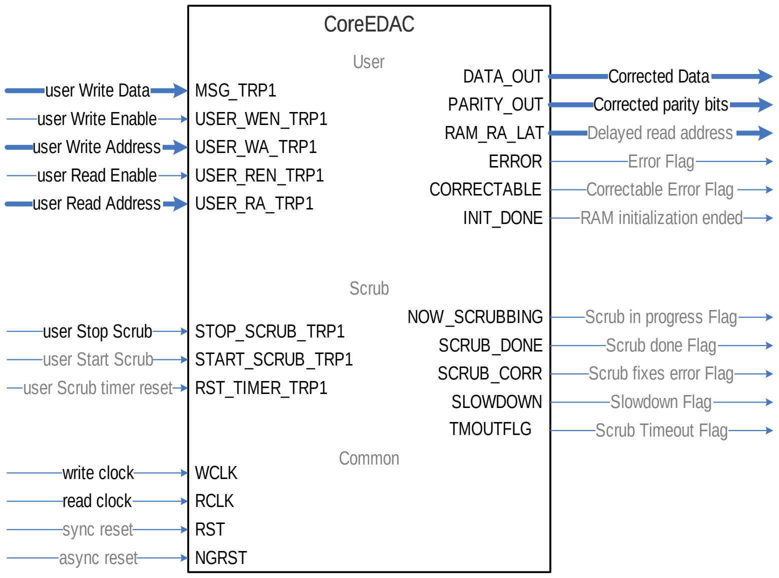


Figure 9 shows another interface example where the core is configured as the EDAC codec to protect the external RAM. TMR is not enabled.

Figure 9 • **EDAC Codec Interface Example**

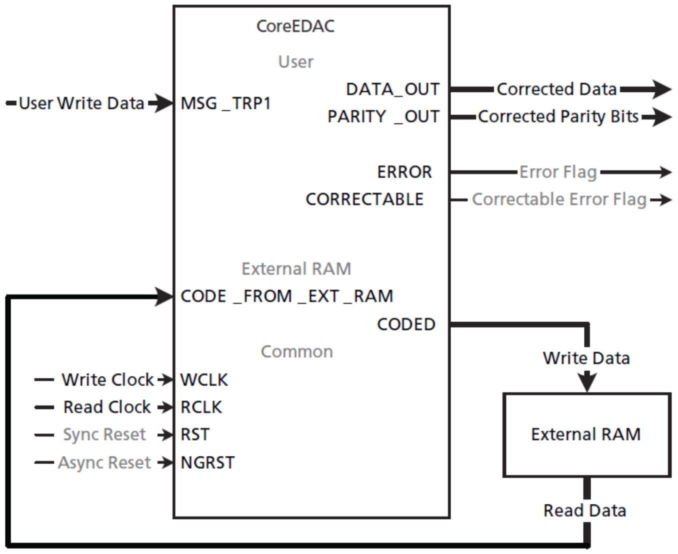


Figure 10 shows an example of an EDAC configuration utilizing TMR. The core input triplets are connected, so that every user input signal drives all the three redundant EDAC instances.

Note: The ports ending in _TRP2 or _TRP3 are not available for the RTG4 family.

Figure 10 • **TMR Interface Example**

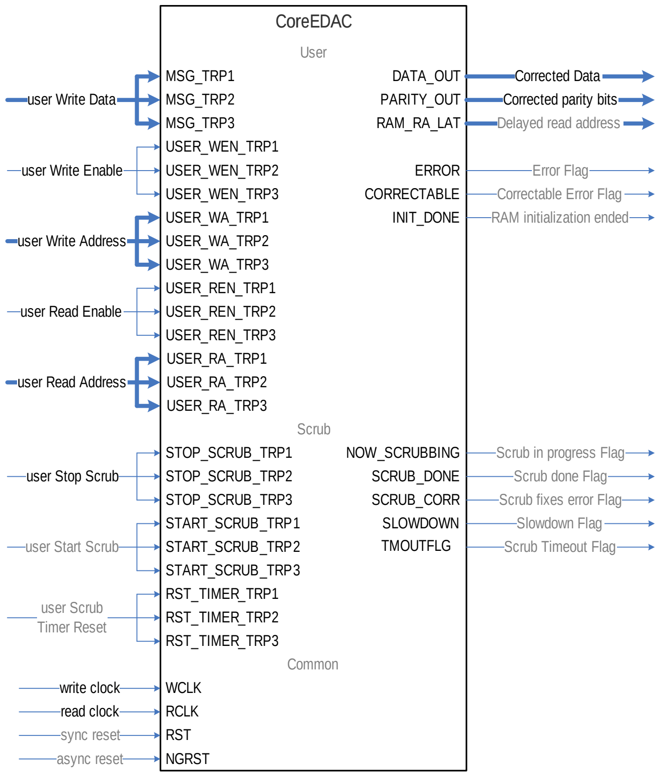
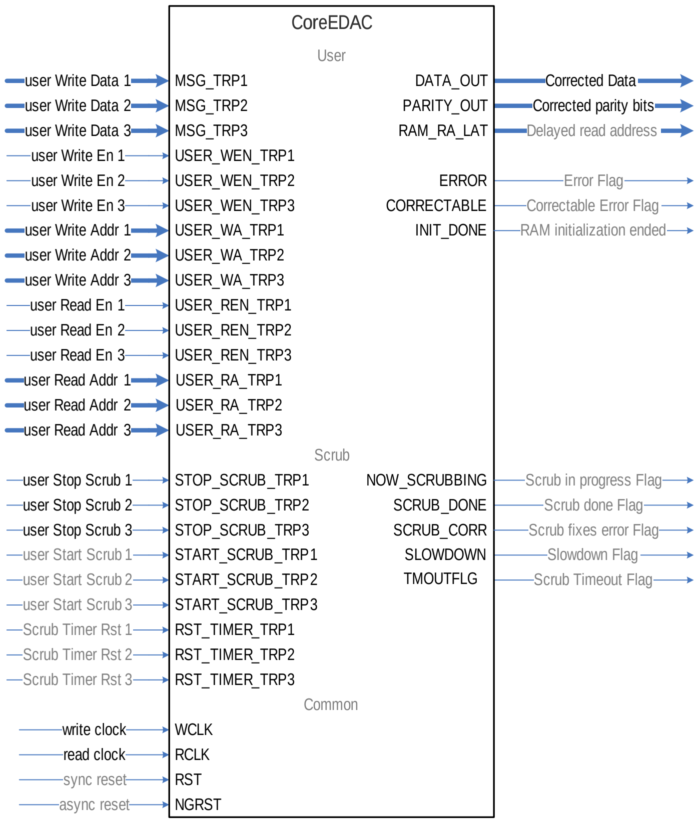


Figure 11 shows another example of using TMR. The user circuitry can employ TMR as well. Thus, the whole logic around the RAM is protected by TMR.

Note: The example is not applicable to the core implementation on the RTG4 family.

Figure 11 • **EDAC TMR Interface to TMR User Circuitry**



4.3 Parameters or Generics

CoreEDAC parameters or generics are shown in Table 12.

Table 12 • **Parameters or Generics**

No.	Name	Valid Range	Description
1	MODE	0, 1	The parameter sets either ECC Codec mode (0) or Protected RAM mode (1). The latter configuration is available only when using the internal (on-chip) RAM.
2	TMR	0, 1	TMR disabled (0) or enabled (1). Once enabled, the core generates three independent sets of the EDAC circuitry and majority vote logic to protect the EDAC itself from soft errors. When the core is instantiated on the RTG4 FPGA family, the TMR is always disabled, TMR=0. RTG4 FPGA design provides the triple redundancy at the silicon level.
3	DAT_WIDTH	4–64	User data bit width, *k*
4	SINGLECLK	0, 1	The parameter sets either independent write or read clocks (0), or a common single-clock (1).

Table 12 • **Parameters or Generics**

5	RAM_DEPTH	8– 543744	Depth (capacity) of the RAM block expressed in words. The maximal RAM_DEPTH value depends on the FPGA device selected, which defines available RAM capacity and the User data width DAT_WIDTH, which defines encoded word bit width. The core automatically calculates the valid upper limit of the RAM_DEPTH based on the FPGA device and DAT_WIDTH selected. The parameter makes sense in the protected RAM configuration only when MODE is equal to 1. Otherwise, it does not affect the generated RTL.
6	USER_ENC_PIPE	0–2	Maximum number of pipelines allowable for the encoder. The actual pipeline count can be equal to or less than the USER_ENC_PIPE. Every pipeline inserted introduces the encoder latency by 1 clock cycle.
7	USER_DEC_PIPE	0–3	Maximum number of pipelines allowable for the decoder. The actual pipeline count can be equal to or less than the USER_DEC_PIPE. Every pipeline inserted introduces the decoder latency by 1 clock cycle.
8	RAM_PIPE	0, 1	Disable (0) or enable (1) the RAM read data output pipeline. The Pipeline is a part of the RAM hard macro. The pipeline being enabled increases the overall read data latency by one clock cycle. The option is available in the protected RAM configuration only when MODE = 1. Otherwise, it does not affect the generated RTL. **Note:** On AX or RTAX-S/SL/DSP devices, the hard RAM macro pipeline is always bypassed. When the RAM_PIPE option is set on the core configuration window, the core implements the read data pipeline as an FPGA fabric register
9	DLY_RD_A_ON	0, 1	Disable (0) or enable (1) the RAM read data matching delay. Once enabled, the option delays the user read address by the overall read data delay imposed by the decoder and optional RAM output pipeline. As a result, the read data and corresponding read address appear at the outputs simultaneously. The option is available in the protected RAM configuration only when MODE is 1. Otherwise, it does not affect the generated RTL.
10	SCRUB_ON	0, 1	Disable (0) or enable (1) scrubbing. The option is available in the protected RAM configuration only when MODE is 1. Otherwise, it does not affect the generated RTL.
11	WRBK_ON	0, 1	Disable (0) or enable (1) writing the corrected word back to RAM during the scrubbing session. When disabled, scrubbing only detects errors but does not correct them. The option is available in the protected RAM configuration if scrubbing is enabled; that is, when MODE is 1 and SCRUB_ON is 1. Otherwise, it does not affect the generated RTL.
12	SCRUB_AMIN	0 to RAM_DEPTH-2	Lower limit of the RAM location range allocated for scrubbing. The parameter makes sense in the protected RAM configuration if scrubbing is enabled; that is, when MODE is 1 and SCRUB_ON is 1. Otherwise, it does not affect the generated RTL. At the very first session after nGrst, the scrubbing starts from address 0.

Table 12 • **Parameters or Generics**

13	SCRUB_AMAX	1 to RAM_DEPTH-1	Upper limit of the RAM location range allocated for scrubbing. The parameter makes sense in the protected RAM configuration if scrubbing is enabled; that is, when MODE is 1 and SCRUB_ON is 1. Otherwise, it does not affect the generated RTL. If SCRUB_AMAX is lower than RAM_DEPTH, a scrubber looks at a few RAM cells above SCRUB_AMAX. Once it detects errors, it raises the ERROR and/or CORRECTABLE flags, but it does not attempt to correct these cells.
14	DIV_WDTH	1–31	Binary divider bit width of the scrubbing refresh period timer. The parameter makes sense in the protected RAM configuration if scrubbing is enabled; that is, when MODE is1 and SCRUB_ON is 1. Otherwise, it does not affect the generated RTL.
15	TMOUT_SET	2-1,000,000	Scrubbing interval timeout setting. The parameter makes sense in the protected RAM configuration if scrubbing is enabled; that is, when MODE is 1 and SCRUB_ON is 1. Otherwise, it does not affect the generated RTL.
16	URAM	0-1	The parameter defines which kind of on-chip RAM hard blocks is going to be used to build the protected RAM. The parameter makes sense on SmartFusion2 family only. If URAM is 0, the Large SRAM is used, otherwise micro RAM.
17	INIT_RAM	0-1	Initialize RAM. If the parameter is set to be 1, upon power-on the core fills out the RAM with all 0s, which are valid Hamming codes. The RAM addresses from SCRUB_AMIN to SCRUB_AMAX are initialized. The option is available when SCRUB_ON is 1.
18	TEST	0-1	Creates a test error input INJ_ERR used for the test purpose.
19	FAMILY	10-26	The target FPGA family numerical value. The parameter is set through the Libero® Project settings dialog and automatically transfers to the core. If the Libero device selection changes, you must invoke the core configuration interface and regenerate RTL.

5 Core Description

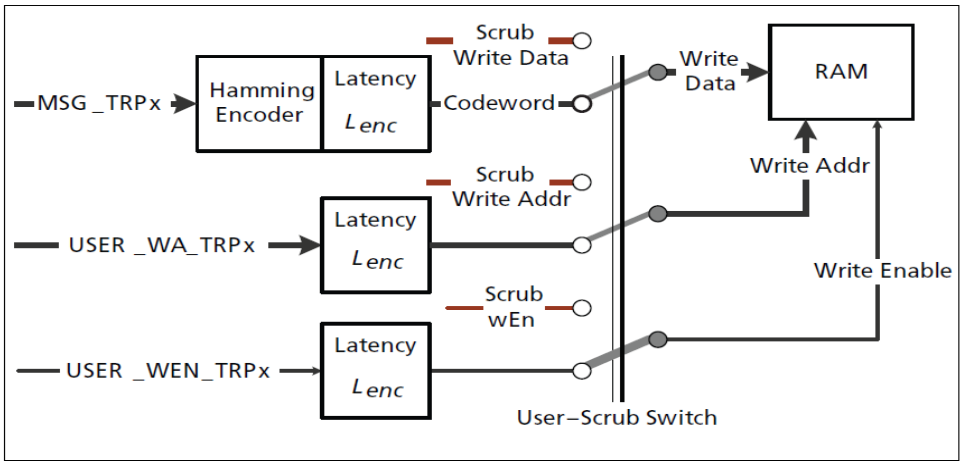
5.1 User Access

5.1.1 User Write Mode

This section describes the user modes when the EDAC is in protected RAM mode.

Figure 12 shows a functional block diagram of EDAC in Write mode. In this mode, the user-scrub switch is in the position shown in Figure 12. The Hamming encoder receives a user data word on the MSG_TRPx input[1], calculates parity bits, and creates the codeword, appending the parity bits to the user word. The encoder generally introduces the latency Lenc of 0 to 2 clock cycles, depending on the user configuration. Figure 12 shows the latency as a delay behind the encoder. It reflects the functionality of the encoder, but in reality, the pipeline delays are evenly distributed over the encoder logic. The core circuitry automatically delays the user write address USER_WA_TRPx and the user write enable signal USER_WEN_TRPx, by the same amount of time of Lenc to compensate for the encoder latency. The user encoded data word, address, and write enable signal reach the RAM at the same time.

Figure 12 • **EDAC Write Mode**



1.TRPx refers to TRP1, TRP2, or TRP3. With the TMR option disabled, TRPx = TRP1.

Figure 13 shows an example of Write mode timing when the actual encoder latency L_{enc} equals two write clock cycles. Write mode lasts as long as the write enable signal is active. Write mode operates in the write clock domain.

Figure 13 • **User Write Mode Timing**

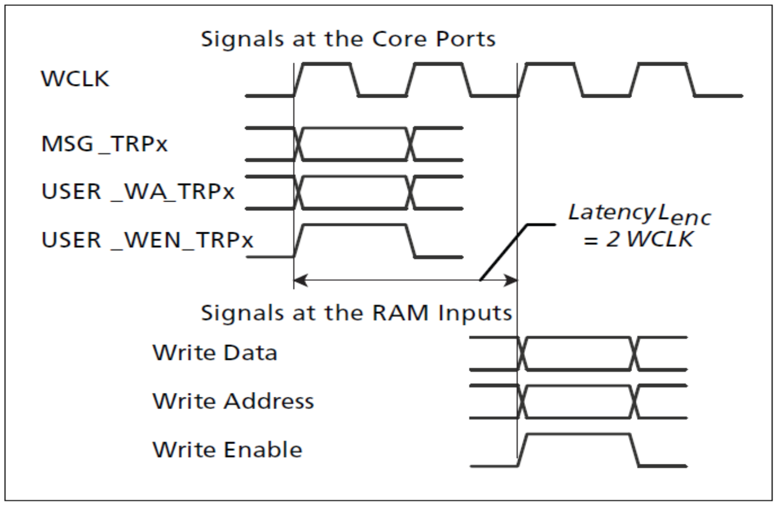


Figure 14 • **User Write Mode Timing when Actual Encoder Latency is One Clock Cycle**

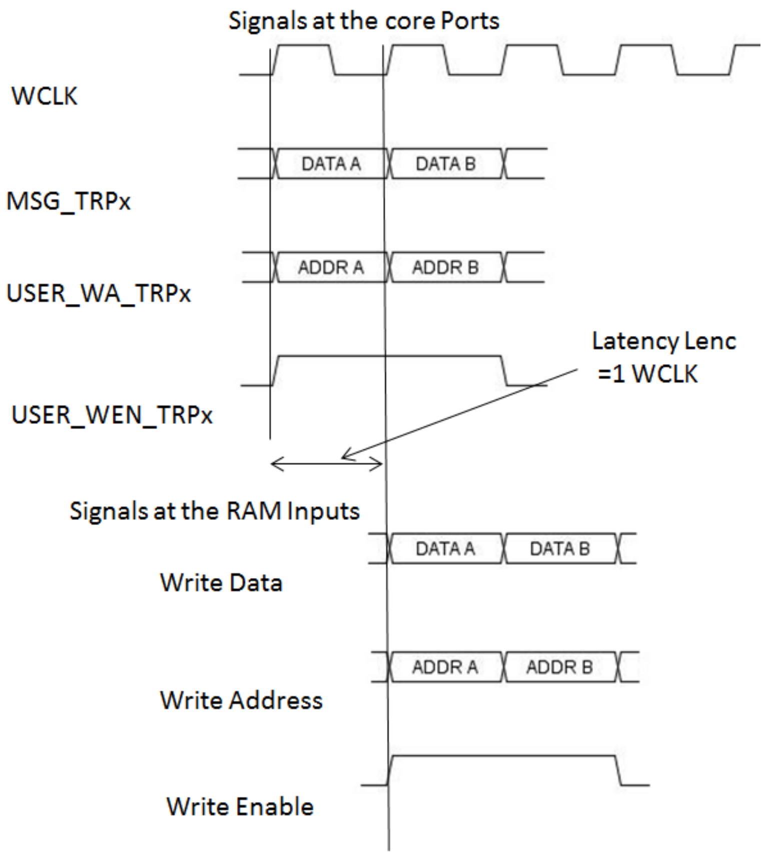
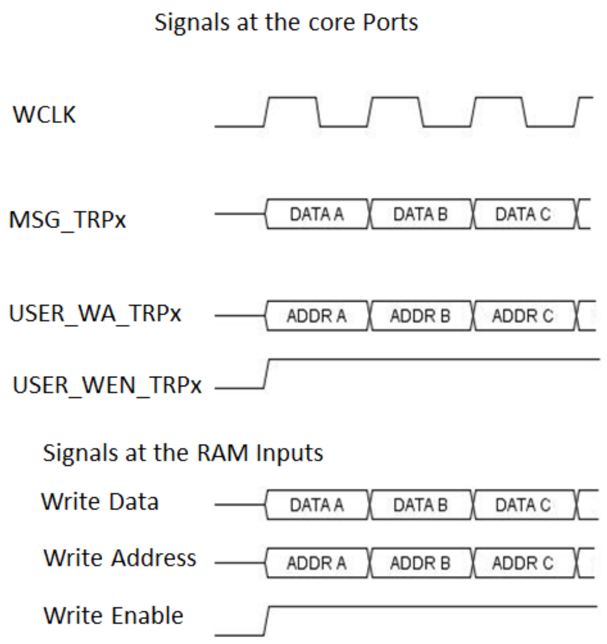


Figure 15 • **User Write Mode when Encoder Latency is Zero**

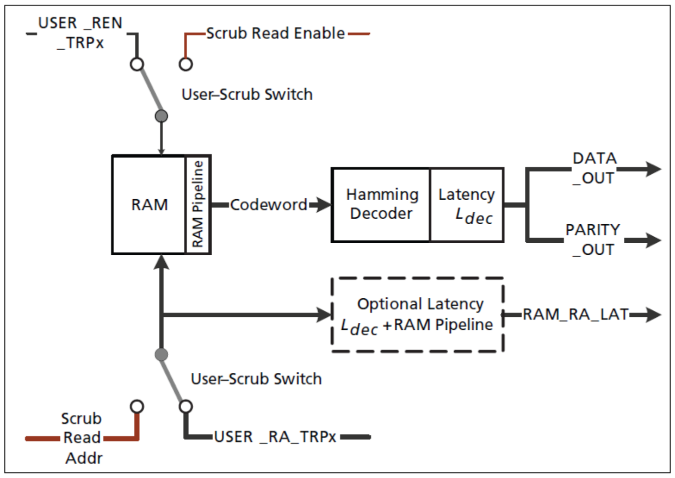


5.1.2 User Read Mode

The EDAC Read mode is shown in Figure 16. The user read address, USER_RA_TRPx, and read enable signal, USER_REN_TRPx, go directly to the RAM, through a user-scrub switch. However, the corresponding codeword appears at the RAM array output one clock cycle later. The optionally enabled RAM pipeline adds one more clock cycle to the delay, and then the codeword goes to the decoder that corrects possible errors and outputs the corrected user read data DATA_OUT, along with the corrected parity bits, PARITY_OUT. The decoder introduces yet another delay, L_{dec}, of 0 to 3 clock cycles, depending on the user configuration. The overall delay between the instance (clock cycle) when the user read address and read enable are issued and the point when the user read data is available can range from 1 to 5 clock cycles. The delay equals 1 + L_{dec} + RAM_PIPE. The latter parameter or generic can take a value of 0 (RAM pipeline disabled) or 1 (RAM pipeline enabled).

The core provides optional matching delay for the user read address. With the option enabled (parameter or generic DLY_RD_A_ON = 1), the core delays the read address so that the user read data and corresponding read address appear at the core ports simultaneously. Read mode uses the RCLK signal.

Figure 16 • **EDAC Read Mode**



An example of Read mode timing is shown in Figure 17. For this example, the RAM pipeline is enabled (RAM_PIPE = 1), and the actual decoder latency, L_{dec}, equals 3 read clock cycles.

The overall delay = 1 + L_{dec} + RAM_PIPE = 5 rClk.

Figure 17 • **User Read Mode Timing**

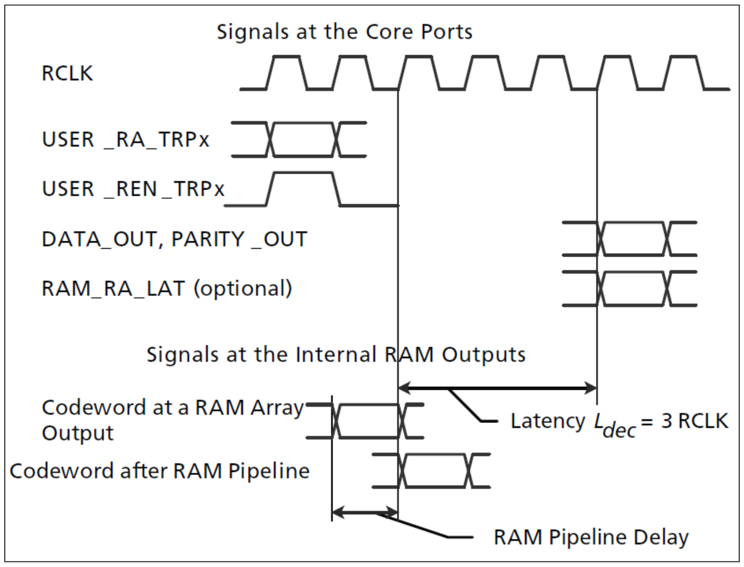
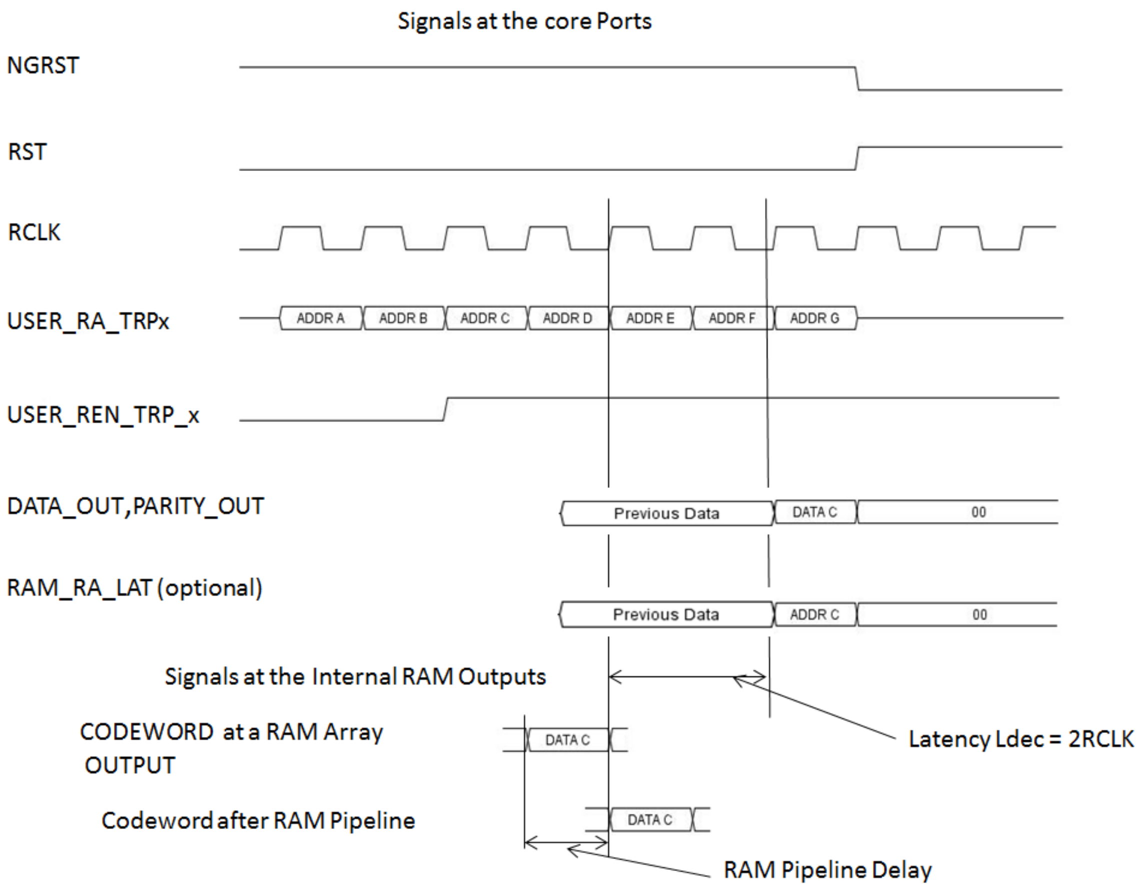

Signals at the Core Ports
RCLK
USER _RA_TRPx
USER _REN_TRPx
DATA_OUT, PARITY _OUT
RAM_RA_LAT (optional)
Signals at the Internal RAM Outputs
Codeword at a RAM Array Output
Codeword after RAM Pipeline
Latency Ldec = 3 RCLK
RAM Pipeline Delay

Figure 18 • **User Read Mode Timing**



In Figure 18, the RAM pipeline is enabled (RAM_PIPE = 1), and the actual decoder latency Ldec equals 2 read clock cycles.

The overall delay = 1 + Ldec + RAM_PIPE = 4 rClk.

Figure 19 • **User Read Mode Timing**

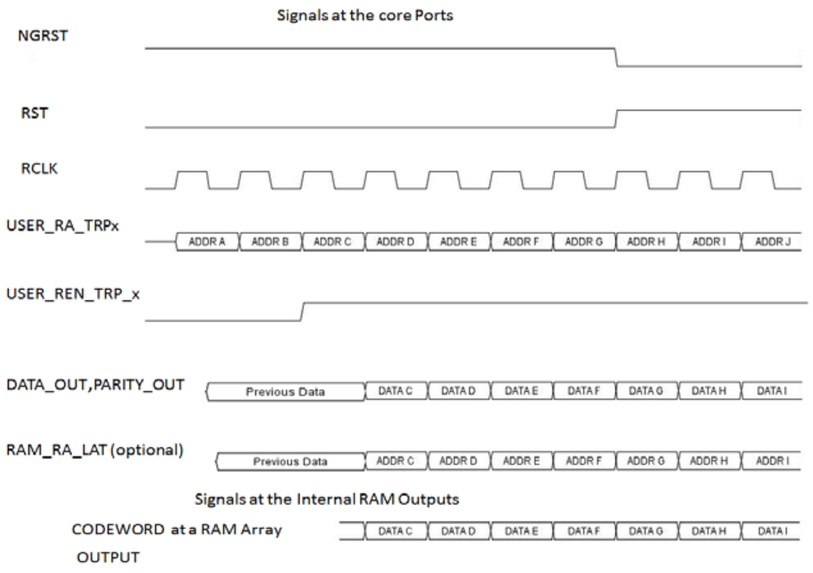


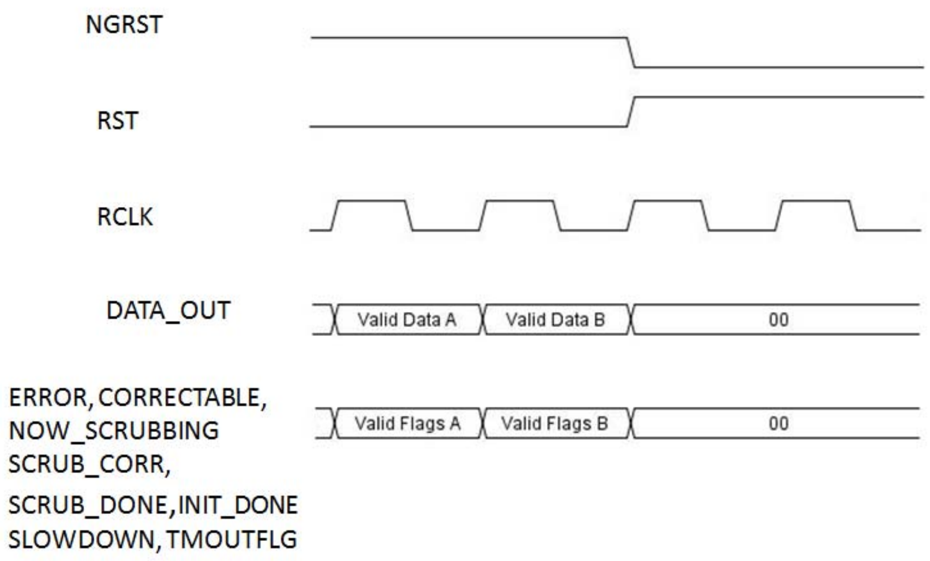
Figure 19 shows the user read mode timing when the actual decoder latency Ldec is zero and RAM_PIPE = 0, the overall delay = 1 + L_{dec} + RAM_PIPE = 1 rclk

Note: In decoder latency zero mode, even after asserting reset, the output data comes directly from RAM as the RAM does not have any reset. Hence, user need to control read enable and resets.

5.1.3 Flags Generation

Figure 20 shows the valid flag generation. All the valid flags goes to default state when RST and NGRST is asserted.

Figure 20 • **Valid Flag Generation**

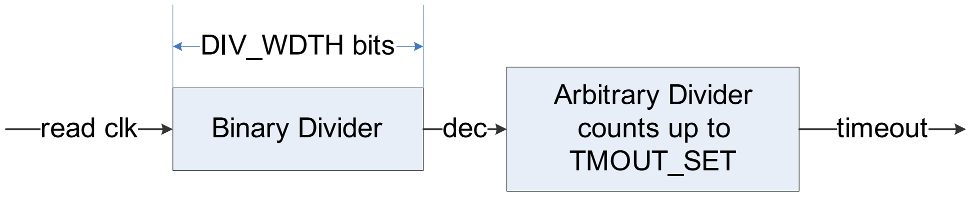

NGRST
RST
RCLK
DATA_OUT
Valid Data A
Valid Data B
00
ERROR, CORRECTABLE, NOW_SCRUBBING SCRUB_CORR, SCRUB_DONE, INIT_DONE SLOWDOWN, TMOUTFLG
Valid Flags A
Valid Flags B
00

5.2 Scrubbing Mode

5.2.1 Scrubbing Refresh Period

The refresh period defines how often the scrubbing sessions run. The block diagram of the refresh period timer is shown in Figure 21. The timer is driven by the RCLK signal. The binary divider that has a configurable bitwidth of DIV_WDTH, generates a relatively slow signal, *dec*. The frequency of the *dec* signal equals the frequency of the read clock divided by 2DIV_WDTH. The *dec* signal serves as an input to the configurable arbitrary divider. It divides *dec* frequency by arbitrary number TMOUT_SET. As a result, the circuitry generates a timeout output signal once per TMOUT_SET × 2DIV_WDTH RCLK periods. The Refresh Period Setup Examples section provides a detailed explanation of setting the refresh period timer.

Figure 21 • **Refresh Period Timer**



The refresh period must be more than ten times the scrubbing time that is, DIV_WDTH and TMOUT_SET parameters must satisfy the following condition:

$\text{TMOUT\_SET}*2^{\text{DIV\_WDTH}} > 10*(\text{SCRUB\_AMAX} - \text{SCRUB\_AMIN})$

Normally, the timeout signal initiates another scrubbing session. As the user access takes priority over scrubbing, there might be an exception. The Flags section covers details of the exception.

5.2.2 RAM Initialization

SECDED Hamming code successfully deals with up to two erroneous bits per word. If the number of errors happens to be more than two, the code is useless. Neither it is able to signal such event took place. Therefore, it is very important to make sure the protected RAM initially contains valid encoded words. At power-on, the RAM contains garbage until some meaningful data are written in the RAM. If a scrubbing session ran over the garbage contents, it would try to fix the garbage data. Every such attempt to correct a data word slows down the scrubbing process and raises meaningless flags until the valid codes are written in the scrubbed space.

To avoid this, Microsemi recommends filling the RAM with valid codes prior to running the scrubbing session. You can initialize the RAM with valid codes or you can make the core do this by selecting the **Initialize RAM with 0** check box on the core UI. Scrubbing space (from Start Address to End Address) is initialized, if the option is enabled. The initialization period starts automatically after the NGRST signal is de-asserted. It is assumed that the NGRST signal follows a beginning of powering up an FPGA device. During the initialization period, the RAM is not available for writing user data. Upon completion of the initialization period, the core generates the INIT_DONE flag. The initialization option is available when scrubbing is enabled.

5.2.3 User Access and Scrubbing

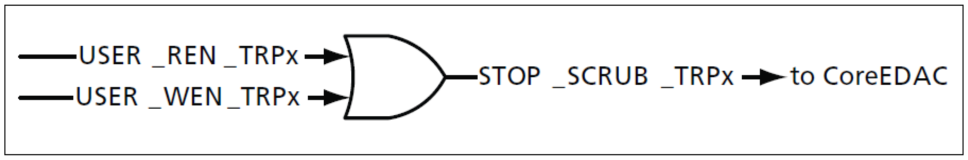
The scrubbing process utilizes time intervals when the RAM is free of user access. It is necessary to allocate enough time for scrubbing to prevent collecting the soft errors. The scrubber needs at least SCRUB_AMAX – SCRUB_AMIN + 5 read clock periods to scan through the RAM addresses designated for scrubbing. If the scrubber detects an error, it takes several additional slower[1] clock intervals to complete each writeback.

User access takes priority over scrubbing. While user access remains active, scrubbing does not start even if timeout occurs. Upon user access completion, the core starts the scrubbing session. In this case, the core generates the SLOWDOWN flag. The flag tells the user that user access is too extensive for the scrubbing to occur with the predefined timeout period. If user access starts in the middle of the scrubbing session, the latter stops and the user gets the access. The interrupted scrubbing waits until the user access is over to complete a session. It is highly desirable for the user to allocate a long enough solid time interval for the whole scrubbing session. It is a bad practice to interrupt the scrubbing session often, as it heavily extends the scrubbing time. Once the user access interrupts the scrubbing, it takes several clock cycles to return to the scrubbing mode. If gaps between user accesses are too short, the scrubbing does not start or resume until the gaps are longer.

User access manifests itself as the active STOP_SCRUB_TRPx signal.

Figure 22 shows the recommended STOP_SCRUB_TRPx generation, in case of a common read or write clock.

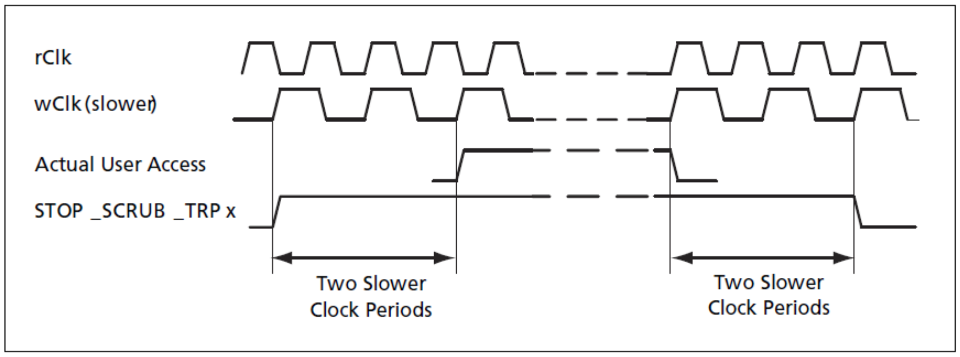
Figure 22 • **Recommended STOP_SCRUB_TRPx for the Common Read or Write Clock**



For independent read or write clocks, the STOP_SCRUB_TRPx signal must be activated prior to the actual user access to provide sufficient transition time. The transition period should not be less than USER_ENC_PIPE + 2 slower clock periods. The same transition period must apply after the user transaction by keeping the STOP_SCRUB_TRPx signal active after the actual user access ends. An example of the timing diagram (encoder latency = 0) is shown in Figure 23.

The slower of the two independent clocks, rClk and wClk, which happens to be the write clock, wClk, in Figure 23, determines the margin intervals. The STOP_SCRUB signal is two slower clock intervals wider on each end than the actual user access.

Figure 23 • **Recommended STOP_SCRUB Generation for Independent Clocks**



The slower clock period (read or write) is the larger one; or a clock period of the clock with the lower frequency. If the write clock frequency is lower than the read clock frequency, the slower clock period is a period of the write clock.

5.3 Flags

When CoreEDAC detects an error, it raises either ‘correctable’ or ‘error’ internal flags. Table 13 shows meaning of the internal flags.

Table 13 • **Correctable and Error Internal Flags**

Errors Detected	Correctable	Error
No errors	0	0
Single error detected and corrected	1	0
Double error detected	0	1
Multiple errors present	Undefined	Undefined

Internal flags are generated solely based on a codeword analysis. Depending on configuration, the core can apply additional logic prior to generating the user flags CORRECTABLE and ERROR. Table 14 explains the logic for the most common core configurations. The user flags appear at the core outputs simultaneously with the relevant read data and parity bits.

Table 14 • **CORRECTABLE and ERROR User Flags**

CoreEDAC Configuration	**Configuration Parameter**			**Flag Logic**
	MODE	**SCRUB_ON**	**INIT_RAM**	
ECC Codec only	0	-	-	No additional logic. The user flags are equal to the internal ones, that is CORRECTABLE=Correctable, ERROR=Error. The flags become valid only after the RAM is initialized with valid encoded data. Make sure that the flags are only used after the initialization.
Protected RAM, scrubber disabled	1	0	-	The core holds flags low from power-on time when NGRST=0 until the USER_REN_TRPx signal(s) is asserted. Keep the USER_REN_TRPx signal(s) low until after the RAM initialization is completed and a guard period is expired. The guard period starts at the clock period when the last RAM cell is initialized. The guard period lasts for 1 + USER_ENC_PIPE clock cycles if common write and read clock is used, that is, SINGLECLK=1. If independent clocks, the guard period equals (1+USER_ENC_PIPE) write clock periods + 2 read clock periods.
Protected RAM, scrubbing on, custom RAM initialization	1	1	0	The core separates flags into two groups: ERROR and CORRECTABLE are the flags generated during user access and SCRUB_CORR, SCRUB_DONE, NOW_SCRUBBING, SLOWDOWN, and TMOUTFLAG are the flags set during scrubbing session. The core holds the user flags CORRECTABLE and ERROR low from power-on time when NGRST=0 until the USER_REN_TRPx signal is asserted. Keep the USER_REN_TRPx signal(s) low until the RAM is initialized and a guard period is expired. The guard period starts at the clock period when the last RAM cell is initialized. The guard period lasts for 1 + USER_ENC_PIPE clock cycles if common write and read clocks are used (SINGLECLK=1). If independent clocks, the guard period equals (1+USER_ENC_PIPE) write clock periods + 2 read clock periods.
Protected RAM, scrubbing on, automatic RAM initialization	1	1	1	The core separates flags generated during the user access and the flags set during scrubbing sessions. The core holds the user access flags low from power-on time when NGRST=0 until the initialization is completed and the flags become valid.

The core also generates flags that indicate the scrubbing process status.

During the scrubbing session, CoreEDAC does not generate CORRECTABLE or ERROR flags. Instead, the core generates the SCRUB_CORR flag. The flag goes high when the correctable one-bit error is detected, corrected and the corrected codeword writes back to the RAM.

During the scrubbing session, the core also generates NOW_SCRUBBING flag. It marks the time intervals when actual scrubbing takes place. As the scrubbing session can be interrupted by user accesses, the flag goes low during the interrupts and additional time necessary to resume the scrubbing session. The flag is always low outside of the scrubbing session.

The SCRUB_DONE flag goes active when the scrubbing session is over. It stays active for one clock cycle when the SINGLECLK = 1 or for two RCLK cycles when independent read or write clocks are used.

TMOUTFLG goes high on the scrubbing timer timeout. It signals the scrubbing refresh period has expired and it is time to start another scrubbing session. Starting from that moment, the core gives the scrubber a grace period to complete scrubbing. The grace period lasts 2*(SCRUB_MAX-SCRUB_MIN) clock intervals, which should be sufficient to complete scrubbing. However, there is a possibility that the scrubbing session does not have enough time to complete due to the user activity. As a result, the core raises the SLOWDOWN flag. This flag signals that the user should "slow down" the access to the RAM to free up necessary time for the scrubbing to complete. The SLOWDOWN flag stays active until the scrubbing session ends.

The flag INIT_DONE is described at the RAM Initialization section.

6 Tool Flows

6.1 License

CoreEDAC is included free in the Libero catalog and does not require a separate license to be instantiated and used in the Microsemi devices. Complete source code and a testbench are provided for the core.

6.1.1 RTL

Complete RTL source code is provided for the core.

6.2 SmartDesign

CoreEDAC is available for download to the Libero IP catalog through the web repository. Once it is listed on the catalog, the core can be instantiated using the SmartDesign flow. For information on using the SmartDesign to configure, connect, and generate cores, refer to the Libero online help.

Figure 24 • **SmartDesign CoreEDAC Instance View**

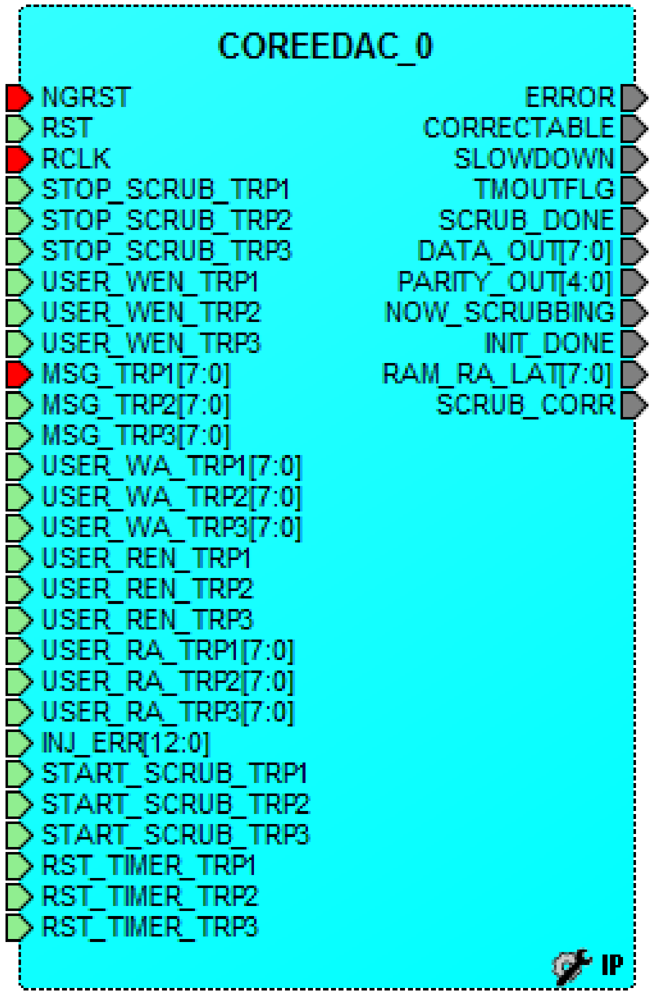


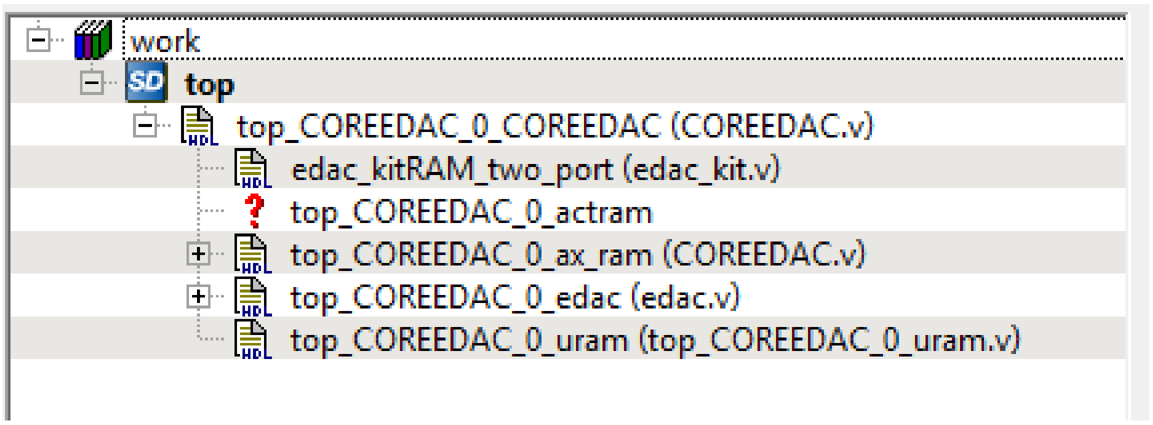
Figure 28 shows the CoreEDAC configuration window. The user interface is context-sensitive, thus its details may look different for a particular configuration. The window displays the encoded word width, which equals a user word width plus the parity bits. The field **Max RAM depth** to the right of the user selected parameter **Depth (words)** indicates potential FPGA capacity for the RAM depth. It is provided as an estimate since Libero determines actual depth limit at the compilation stage. The window also displays the actual encoder and decoder latencies. Refer to the EDAC Pipelines section for more information. After configuring and generating the core instance, basic functionality can be simulated using the testbench supplied with the core. The testbench parameters automatically adjust to the core configuration.

Note:

1. For RTG4, SmartFusion2, and IGLOO2 families, the core provides one extra parameter to select large RAM or micro RAM. Depending upon which RAM has been selected, that RAM gets generated and '?' is observed for the other one in Libero design hierarchy.

Example: Figure 25 shows the Libero design hierarchy when micro RAM is selected in the configurator.

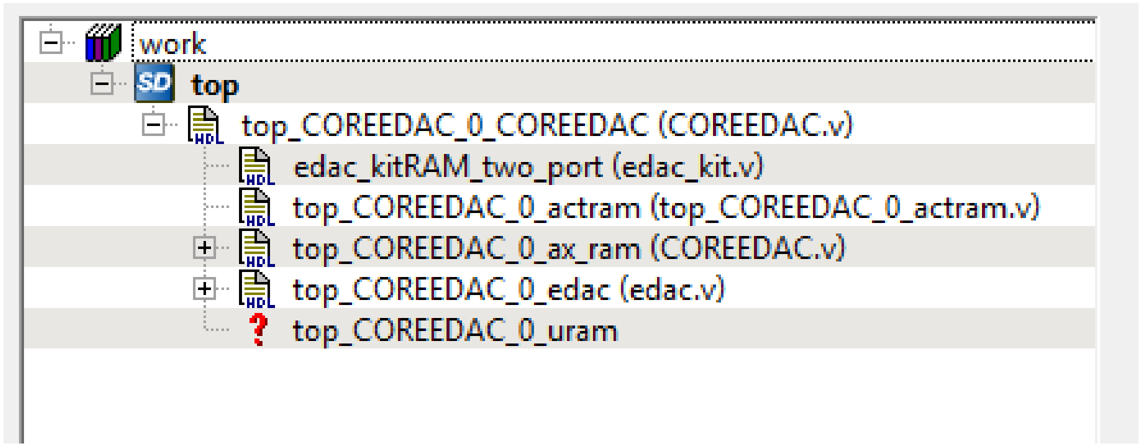
Figure 25 • **Libero Design Hierarchy view when Micro RAM is Selected**



```
work
  top
    top_COREEDAC_0_COREEDAC (COREEDAC.v)
      edac_kitRAM_two_port (edac_kit.v)
      top_COREEDAC_0_actram
      top_COREEDAC_0_ax_ram (COREEDAC.v)
      top_COREEDAC_0_edac (edac.v)
      top_COREEDAC_0_uram (top_COREEDAC_0_uram.v)
```

Figure 26 shows the design hierarchy when Large RAM is selected.

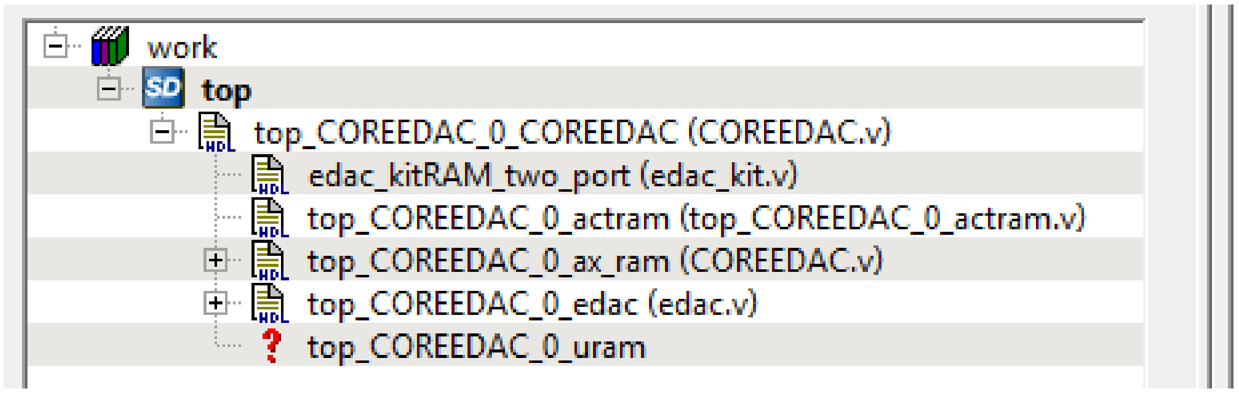
Figure 26 • **Libero Design Hierarchy view when Large RAM is Selected**



```
work
  top
    top_COREEDAC_0_COREEDAC (COREEDAC.v)
      edac_kitRAM_two_port (edac_kit.v)
      top_COREEDAC_0_actram (top_COREEDAC_0_actram.v)
      top_COREEDAC_0_ax_ram (COREEDAC.v)
      top_COREEDAC_0_edac (edac.v)
      top_COREEDAC_0_uram
```

2. For families other than RTG4, SmartFusion2, and IGLOO2, the large RAM gets generated and '?' is observed for Micro RAM in design hierarchy, as shown in Figure 27.

Figure 27 • **Libero Design Hierarchy View**



```
work
  top
    top_COREEDAC_0_COREEDAC (COREEDAC.v)
      edac_kitRAM_two_port (edac_kit.v)
      top_COREEDAC_0_actram (top_COREEDAC_0_actram.v)
      top_COREEDAC_0_ax_ram (COREEDAC.v)
      top_COREEDAC_0_edac (edac.v)
      top_COREEDAC_0_uram
```

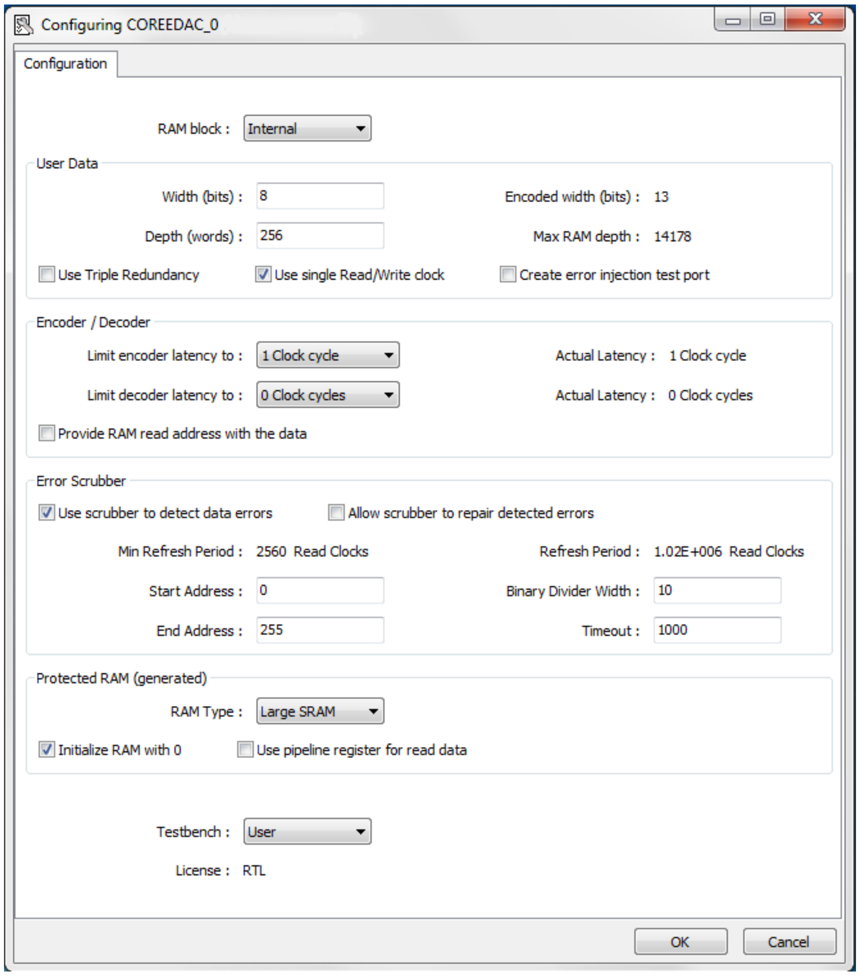
6.3 Simulation Flows

To run simulations, select the user testbench in the core configuration window. After generating the core, the Libero software installs the pre-synthesis testbench HDL files.

Consider an example of instantiating CoreEDAC as an IP component named *tst_edac*. To run the testbench, set the Libero software design root to the core instance **tst_edac_tst_edac_0_COREEDAC**, and run pre-synthesized design simulation.

Note: The RAM output gives 'X' or invalid data even after reset is de-asserted. Read enable should be asserted to get the valid output.

Figure 28 • **CoreEDAC Configuration Window**



Configuring COREEDAC_0

Configuration

RAM block : Internal

User Data

Width (bits) : 8
Encoded width (bits) : 13
Depth (words) : 256
Max RAM depth : 14178
☐ Use Triple Redundancy ☑ Use single Read/Write clock ☐ Create error injection test port

Encoder / Decoder

Limit encoder latency to : 1 Clock cycle
Actual Latency : 1 Clock cycle
Limit decoder latency to : 0 Clock cycles
Actual Latency : 0 Clock cycles
☐ Provide RAM read address with the data

Error Scrubber

☑ Use scrubber to detect data errors ☐ Allow scrubber to repair detected errors
Min Refresh Period : 2560 Read Clocks
Refresh Period : 1.02E+006 Read Clocks
Start Address : 0
Binary Divider Width : 10
End Address : 255
Timeout : 1000

Protected RAM (generated)

RAM Type : Large SRAM
☑ Initialize RAM with 0 ☐ Use pipeline register for read data

Testbench : User
License : RTL

6.4 Synthesis in Libero

To run synthesis on the CoreEDAC, set the design root to the IP component instance and click on Synthesize in Libero Design Flow pane. This will invoke Synplify Pro and automatically runs the synthesis.

6.5 Place-and-Route in Libero

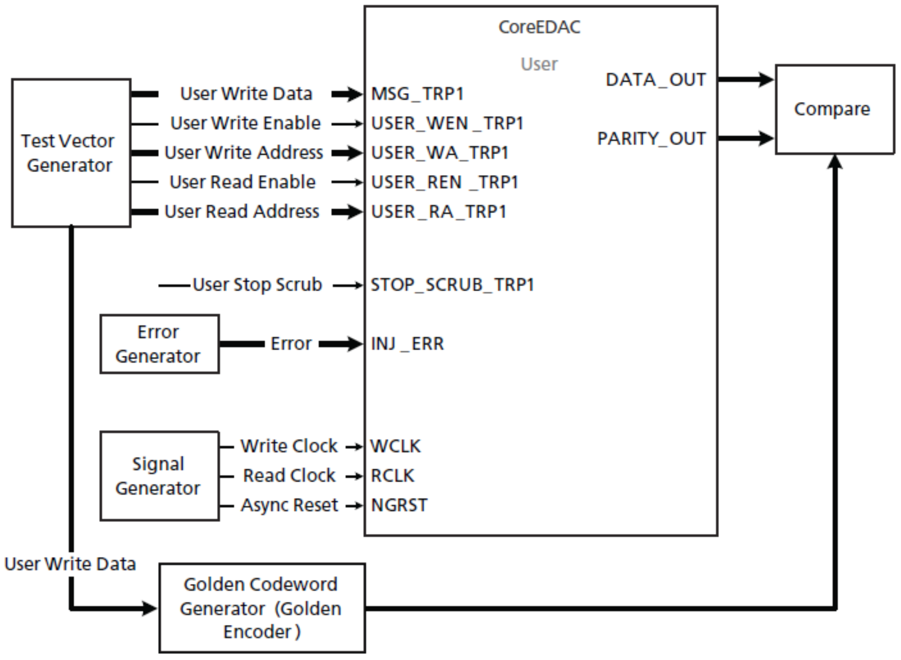
After the design has been synthesized, run **Compile** and then place-and-route tools.

7 User Testbench Operation and Modification

This release of CoreEDAC includes a user testbench that verifies operation of the CoreEDAC engine. The user testbench tests the protected RAM configuration of the EDAC. The TMR option is disabled.

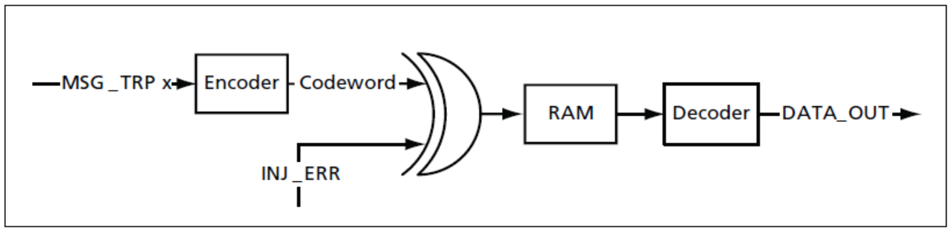
A simplified block diagram of the user testbench is shown in Figure 29. The user testbench instantiates the EDAC engine configured, as well as behavioral non-synthesizable models of an input test vector generator, golden behavioral encoder (codeword generator), comparator, and a signal generator that provides necessary clock, reset, and other signals.

Figure 29 • **User Testbench**



The testbench injects single errors in the codewords at the core encoder output by XORing the codewords with an error vector INJ_ERR (Figure 30). The testbench then compares the actual EDAC output data DATA_OUT and parity bits PARITY_OUT against the golden codeword vector.

Figure 30 • **Testbench Injects Errors in Codewords**



CoreEDAC automatically generates the Verilog or VHDL testbench behavioral code based on the selected core language.

8 Refresh Period Setup Examples

The refresh period setup can be explained using the following examples.

8.1 Example 1

In this example, the desired refresh period is 10 hours. It can tolerate a limited resolution of one minute (the refresh period = 10 hours ± 1 minute.). Read clk frequency f_{read} is 17.895697 MHz. This is a good number to start with, but the next example demonstrates that any number is OK.

Set the DIV_WDTH parameter to 30. The binary divider divides the read clk frequency by 2^{DIV_WDTH} (in this example, it divides by 2^{30} = 1073741824). The frequency at the binary divider output equals 17.895697 MHz/1073741824 to 1/60 Hz. In other words, the binary divider output signal *dec* toggles once a minute.

10 hours contain 10 times 60 minutes, which equals to 600 minutes. It requires 10 bits to be presented. Set the TMOUT_SET to 600 to make the arbitrary divider divide the frequency by 600. The TIMEOUT signal shows at an interval of one minute, times 600, which is equal to 10 hours.

8.2 Formal Algorithm

A formal algorithm to select the two parameters, DIV_WDTH and TMOUT_SET is described below:

1. Check if the quotient $2^{31/f}{}_{read}$ does not exceed the resolution expressed in seconds.

   For example, at $^{f}{}_{read}$ = 100 MHz, the quotient

$$\frac{2^{31}}{fread} \cong \frac{2.5 \times 10^9}{10^8} \cong < 21.5 < 60\ seconds$$

2. Reduce DIV_WDTH by 1 iteratively, until the quotient $2^{DIV\_WDTH/f}{}_{read}$ is less than the resolution expressed in seconds. The goal is to get a period of the binary divider output signal *dec* as large as possible, but not to exceed the refresh period resolution. For example, at $^{f}{}_{read}$ = 10 MHz, the quotient.

$$\frac{2^{29}}{fread} \cong \frac{5.37 \times 10^8}{10^7} = 53.7 < 60\ seconds$$

   Then set DIV_WDTH to 29.

   Calculate TMOUT_SET = ceil (required refresh period × $^{f}{}_{read}$ / 2^{DIV_WDTH}).

   For example, with DIV_WDTH = 29, $^{f}{}_{read}$ = 10 MHz, and required refresh period = 10 hours.

$$\text{TMOUT\_SET} = ceil(required\ refresh\ period/(\frac{2^{\text{DIVT\_WDTH}}}{fread})) = ceil(10 \times \frac{3{,}600}{53.7}) = ceil(670.4) = 671$$

Verify that the actual timeout interval is what is expected. There are two consecutive counters: the binary divider, and the arbitrary divider with a total division coefficient of 2^{29} × 671.

$$Timeout\ Interval = 2^{29} \times \frac{671}{10\ MHz})) = 36{,}024\ seconds = 10\ hours\ and\ 24\ seconds$$

8.3 Example 2

This example, applies the algorithm above to the following conditions:

- Required refresh period = 24 hours
- Acceptable resolution = 15 minutes
- Read clock frequency = 50 MHz

Step 1 of the algorithm yields DIV_WDTH = 31 as the quotient:

$$\frac{2^{31}}{f_{read}} \cong \frac{2.15 \times 10^9}{50 \times 10^7} = 43 < (15 \times 60\ seconds)$$

Step 2 yields:

$$\text{TMOUT\_SET} = ceil\left(required\ refresh\ period \times \frac{f_{read}}{2^{\text{DIVT\_WDTH}}}\right)$$

$$= ceil\left(24 \times 3600 \times \frac{50{,}000{,}000}{2^{31}}\right) = ceil(2{,}011.7) = 2{,}012$$

The actual Timeout interval should be verified. The two consecutive counters provide a total division coefficient of $2^{31} \times 2012$.

$$Timeout\ Interval = 2^{31} \times \frac{2{,}012}{50\ MHz} = 86{,}415\ seconds = 24\ hours\ and\ 15\ second$$