**Content**

1. **Functional Overview**
2. **Interface**
3. **Types of memories**
4. **Block Diagram**
5. **Key Parameters**
6. **Sub-block Description**
   1. **core\_1r1we\_rl2**
   2. **mux**
   3. **t1\_align**
   4. **t1\_stack1**
   5. **t1\_stack2**
   6. **t2\_align**
   7. **t2\_stack**
7. **Formal**
8. **Functional Overview:**

The 1r1w\_a127 IP implements 1R1W (2 port) functionality. The bulk of the memory used in this IP is 1RW or single port memory. In addition there is small amount of 2 port 1R1W memory macro also used in the IP. The 2 port memory is used to store the state and some data. The one port memory is mainly used to store the data.

This IP is implemented to reduce the read latency (Reduced Latency or RL). The target latency for this IP is gated by the latency of reading the data out from the T1 memory. The state lookup and all the relevant decisions are made in parallel and no additional cycle of latency is added on the read data path. This IP supports the following features:

* + ECC options: 1r1W\_A127 supports SECDEC on the state data. Regular data is not ECC protected. The ECC is generated and checked within the IP for the state data. The ECC logic is transparent to the external logic.
  + Flopping options : There are four main flopping options available in this IP – FLOPIN, FLOPOUT, FLOPMEM and FLOPCMD. FLOPIN and FLOPOUT are flop stages to flop the input commands and output data. FLOPMEM flops the read data from the memory macros and the FLOPCMD flops the command to the memory macros. Typically FLOPCMD controls the flop insertion after the ECC logic and FLOPMEM controls the flop insertion right after the MEM data read. Depending on the physical design requirements they can be placed at different stages in the IP logic. For details of where these flops can be placed look at the detailed sub-block description.
  + Sub-packing options: is support by 1r1w\_a127. Using this option memogen can optimize the macro sizes by packing multiple logical words in one physical row. This option enables the platform to choose more optimal physical macros when the width of the required memory is small.
  + Word Write Enable: 1r1w\_a127 is the algorithm that implements the word write enable functionality on top of reduced latency 1R1W algorithm (RL). There are 8 word write enables implemented for this IP. The data width is divided into 8 words and each word write is controlled by the one bit of the write enable primary input.

1. **Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| Pin | Pin Type | Description | Notes |
| Clock and Initialization | | | |
| clk | Input | Clock | Clock input pin |
| rst | Input | Reset | Reset pin |
| refresh | Input | Refresh | Signal to the controller that eDRAM can be refreshed in this cycle. Only needed when IP uses eDRAM. (Optional) |
| ready | Output | Ready | Assertion implies memory core ready for functional operation |
| Read Ports | | | |
| addr\_0 | Input | Read Address | Address bits (width N bits) to denote up to 2N addresses;  N is parameter. Same value for all ports |
| read\_0 | Input | Read Valid | Denotes that a Read command is valid |
| dout\_0 | Output | Read Data | Data (width W) associated with a read address;  W is parameter. Same value for all ports |
| read\_vld\_0 | Output | Read Data Valid | Denotes valid read data |
| read\_serr\_0 read\_derr\_0 | Output | Read Error | Denotes read data error. Data on *dout* pins is invalid |
| paddr\_0 | Output | Physical Address | Physical Address (width P bits) associated with the read. Parameter P depends on the physical address space of the memory core |
| Write Ports | | | |
| addr\_1 | Input | Write Address | Address bits (width N bits). Parameter N is the same as that used for Read address. |
| write\_1 | Input | Write Valid | Denotes that a write command is valid |
| din\_1 | Input | Write Data | Data (width W bits) associated with a write command; Parameter W is same that for Read data. |

1. **Types of memories**

The following psudocode describes the memory types used in the 1r1w\_a127 IP.

# Combined Tag and Cache with 8 Word Enables

1R1W\_A127(f, w, b) [RL]

vars        : n  \_\_ bank\_count(1)

              ws \_\_ words\_array(n,w)

              b' \_\_ bits(b)

formulae    : p  = TAG\_BITS(n+1024)

              w' = ws.max()

macros:

type1(BASE)       : n \* 1RW(f, ws, b')

type2             : 2 \* 1R1W(f, w', b'+p)

The above algorithmic description defines the characteristics (frequency, depth and width) of the memory macros used for 1r1w\_a127. There are two types of memories in the IP – type 1 (T1) and type2 (T2).

* 1. **T1 Memory:** This memory is the bulk of the memory used in the 1r1w\_a127 and is used for the primary data storage. The size of this memory is equal total capacity requirement of the algorithmic memory. The depth of the memory is equal to the depth of required memory and the width is equal to the width of the IP. In 1r1re\_a127 1RW (single port memory) is used as physical memory for T1 memory.
  + f – target frequency of T1 memory. This parameter is user defined.
  + ws – The optimal depth of the T1 memory macros. For IP sizes which are non-power-of-two the tool can optimize the area using different size T1 macros. This parameter is Platform optimized.
  + b’ – data with of the IP. This parameter is user defined.
  + n – is the optimal number of T1 macros. This parameter is Platform optimized.
  1. **T2 Memory:** This memory is used for two types of data – regular data that cannot be written in the T1 banks due to collision read and write collision and the state. The regular data is referred to as the cache data and the state data is referred to as the tag data. Reads are given higher priority on T1 reads and on collision cases write data is written to the T2 memory. The tag for 1r1w\_a127 consists of the bank number and the write enable signals associated with the writes. Note that this structure is protected by ECC (SECDEC).
  + f – target frequency of T1 memory. This parameter is user defined.
  + w’ – The maximum depth of the T1 memory macros. For IP sizes which are non-power-of-two the tool can optimize the area using different size T1 macros. w’ would be the depth of the deepest t1 macro used. This parameter is Platform optimized.
  + b’ – data with of the IP. This parameter is user defined.
  + p – is the width of the tag, the write enable ECC protected.

1. **Block Diagram**

**Figure 1 Picture.**

An algo\_1r1we\_rl2\_pseudo\_top instantiates:

1. For a1\_loop (algo) : An instance of algo\_1r1we\_rl2\_pseudo. This module consists of the two modules – core\_1r1we\_rl2 and mux. The core consists of the bulk of the algorithm and the mux translates the data from the logical p-interfaces and s-interfaces to physical T1 interfaces and T2 interfaces.
2. For t1\_loop (t1 memories): An instance of infra\_align\_bw\_pseudo, an instance of infra\_stack\_1r1w, and NUMXBNK (NUMVBNK/4) instances of infra\_stack\_pseudo. NUMVBNK is a top level parameter which is optimized by the memogen. It defines the number of T1 banks. The infra\_align\_bw\_pseudo module talks to the T1 memory macros and enables the word write functionality.
3. For t2s\_loop (t2 state memories): Two instances of infra\_align\_ecc\_1r1w and two instances of infra\_stack\_1r1w. Each instance controls the access to one of the two T2 memory macros.
4. For t2d\_loop (t2 data memories): Two instances of infra\_align\_bw\_1r1w and two instances of infra\_stack\_1r1w. The two instances sit closest to the T2 memory macros.
5. **Key Parameters:**
6. **Block descriptions**
   1. **core\_1r1w\_rl2**
7. **Block Diagram:**

****

1. **Interfaces:** There are three key interface of the core\_1r1w\_rl2 block – v-interface, p-interface and s-interface.

**Virtual interface (v-interface):** The primary input and output interface of the 1R1W\_RL2 IP is visible to the 1R1W\_rl2 core logic. This interface is driven by and used by the application logic to get the data in and out of the IP. For 1R1W\_RL2 core there is a write port and a read port. The read and write ports have the width defined by the parameter ‘WIDTH’. On the read-side the interface has the read data interface, read data forward interface and the read error interface {vread\_serr, vread\_derr, vread\_padr}. The read data forward signal is asserted if the data has been forwarded by an older write operation in the pipe instead of the data from the T1 bank. The error signals signal ecc error information to the user logic. These signals are only valid when the IP has enabled ECC generation and checking. vread\_padr communicates the physical bank where the data came from (including T1 and T2 banks).

input vwrite;

input [BITADDR-1:0] vwraddr;

input [WIDTH-1:0] vdin;

input vread;

input [BITADDR-1:0] vrdaddr;

output vread\_vld;

output [WIDTH-1:0] vdout;

output vread\_fwrd;

output vread\_serr;

output vread\_derr;

output [BITPADR-1:0] vread\_padr;

**Physical interface (p-interface):** This p-interface is used by the core logic to read and write data from the T1 memories. The address is broken into the row and the bank address. On the read-side the interface gets the read data along with a read data forward signal and the read error interface {pread\_serr, pread\_derr, pread\_padr}. The read data forward signal is asserted if the data has been forwarded by an older write operation in the pipe (from the stage of logic ahead of the core logic) instead of the data from the T1 bank. The error signals signal ECC error information to the user logic. These signals are only valid when the IP has enabled ECC generation and checking. pread\_padr communicates the physical bank where the data came from (including T1 and T2 banks)

output pwrite;

output [BITVBNK-1:0] pwrbadr;

output [BITVROW-1:0] pwrradr;

output [WIDTH-1:0] pdin;

output pread;

output [BITVBNK-1:0] prdbadr;

output [BITVROW-1:0] prdradr;

input [WIDTH-1:0] pdout;

input pdout\_fwrd;

input pdout\_serr;

input pdout\_derr;

input [BITPADR-BITPBNK-1:0] pdout\_padr;

**State interface (s-interface):** This s-interface is used by the core logic to read and write data from the T2 memories. T2 memories contain the cache data and the cache tag. The cahce data width is the same as the width of the T1 memory banks. The cache tag contains the T1 bank number whose data is sitting in the cache. This is a critical state for the IP and this data is duplicated and ECC protected to acieve 2 bit ECC correction. The data is broken down into ddin and sdin. ddin is the cache data and ddin is the tag data. The cache has the same number of rows as a physical bank and hence the s-interface addresses address only VROW depth (equivalent to depth of one T1 bank). There are two read interfaces for the T2 memory. Each one has the cache data and tag data interface (ddout, sdout). T2 memories are acced by both read and write operations and therefore there are two read interfaces. On both read-side interfaces gets the read data along with a read data forward signal and the read error interface {ddout\_serr, ddout\_derr, ddout\_padr}. The read data forward signal is asserted if the data has been forwarded by an older write operation in the pipe (from the stage of logic ahead of the core logic) instead of the data from the T2 bank. The error signals signal ECC error information to the user logic.

parameter SDOUT\_WIDTH = 2\*(BITVBNK+1)+ECCBITS;

output swrite;

output [BITVROW-1:0] swrradr;

output [SDOUT\_WIDTH-1:0] sdin;

output [WIDTH-1:0] ddin;

output sread1;

output [BITVROW-1:0] srdradr1;

input [SDOUT\_WIDTH-1:0] sdout1;

input [WIDTH-1:0] ddout1;

input ddout1\_fwrd;

input ddout1\_serr;

input ddout1\_derr;

input [BITPADR-BITPBNK-1:0] ddout1\_padr;

output sread2;

output [BITVROW-1:0] srdradr2;

input [SDOUT\_WIDTH-1:0] sdout2;

input [WIDTH-1:0] ddout2;

input ddout2\_fwrd;

input ddout2\_serr;

input ddout2\_derr;

input [BITPADR-BITPBNK-1:0] ddout2\_padr;

1. **Key Parameters:**

The following are the key parameters used in the core logic:

1. parameter WIDTH : is the data width supported by the IP
2. parameter NUMADDR : is the total address space for the IP
3. parameter NUMVROW : is the number of rows in the T1 banks
4. parameter NUMVBNK: is the number of T1 banks
5. parameter BITPADR : is the number of bit sufficient for all the banks in the design (T1 and T2 banks), used for physical address interface.
6. parameter SRAM\_DELAY: is the delay to access T1 banks
7. parameter DRAM\_DELAY: is the delay to access T2 banks
8. parameter FLOPIN : is the parameter that controls the flopping option on the command signals of the core.
9. parameter FLOPOUT : is the parameter that controls the flopping option on the output signals of the core.
10. parameter ECCBITS : is the parameter that controls the ECC bits in the state memory (T2 memory)
11. **Conflict Resolution:**

Every write launches a read operation of the cache and tag data as soon as it enters the core logic on the sread2 interface. It takes SRAM delay amount of latency for the cache send back the data to the core logic. Till than the write command sits in the write command FIFO. When the cache data gets back to the core logic the decision to launch the write to the T1 or the T2 memory is made. The decision is based on the new read command being launched in this cycle. The Writes to the T1 and T2 are also dependent on the state of the tag data received by the core logic. The following logic describes the conflict resolution logic in the core\_1r1w\_rl2.

*If (Write Operation has a bank conflicts with read operation) {*

*Write operation is launched on the swrite interface;*

*If (Data in the write address row of the cache is valid)*

*Flush is launched on the pwrite interface;*

*} else {*

*Write operation is launched on the pwrite interface;*

*If (cache data is for the same address as the new write address)*

*Clear the data from the cache - clear on swrite interface;*

*}*

1. **Microarchitecture Description:**

The v-interface of the core\_1r1w\_rl2 is driven by the primary read and write commands to the IP. This block is supposed to transform these commands into corresponding reads and writes to the T1 and T2 memories. The T1 memories are single port memories and theT2 memories have 2 read and 1 write ports. The core logic is supposed to resolve the conflicting access to the T1 memories.

* 1. **The Read operation:** The read operation launches a read command to the T1 and T2 memory banks in parallel. Depending on the tag value the data from the T1 or T2 command is forwarded to the requester after DRAM\_DELAY delay. Every read command can get the data from three sources – T1 or T2 memory or from an earlier write that has not made it to the T1 or T2 memories.
  2. **The Write operation:** Every write operation needs to check the state of the tag T2 memory and resolve any T1 bank conflict with the current read operation before posting the data to the T1 or T2 memory. Every write launches a read operation of the cache and tag data as soon as it enters the core logic on the sread2 interface. It takes SRAM delay amount of latency for the cache send back the data to the core logic. Till than the write command sits in the write command FIFO. When the cache data gets back to the core logic the decision to launch the write to the T1 or the T2 memory is made. The decision is based on the new read command being launched in this cycle. The Writes to the T1 and T2 are also dependent on the state of the tag data received by the core logic. The logic described above is used to resolve conflicts between read and write commands.

The read and write addresses on the v-interfaces are broken into a bank and row address depending on the number of logical banks used in the T1 memory. This logic also enables the use of non-power- of- two bank logical bank sizes to be used in the T1 memory. The decoded address along with the read and write commands sit in two command FIFO’s – vread\_reg\* and vwrite\_reg\*. The depth of the write command FIFO is SRAM\_DELAY or the read delay of the T2 memory and the depth of the T1 command FIFO is DRAM\_DELAY or the delay for the read from the T1 memory.

**The snoop logic:** There are three more fifos on the sread and pread interfaces –

**The forwarding logic:**

1. **Formal**

**Figure 2: Diagram of all the elements.**

* 1. **Coverage Matrix:**  The coverage matrix lists each instance and the assertions proved for each instance. ….

|  |  |  |  |
| --- | --- | --- | --- |
|  | **dout\_check** | **mem\_check / port check** | **mem side port check** |
| **a1 loop** | pdout,sdout,ddout  *pdout\_check* |  |  |
| **t1\_loop.align** | dout,fwrd,padr  *t1\_align\_dout\_check* | mem  *t1\_1\_mem\_check* | mem\_wr\_range, rd\_wr\_pseudo  *t1\_align\_check* |
| **t1\_loop.stack1** | dout,fwrd,padr,derr  *t1\_1\_stack\_dout\_check* | mem, wr\_range  *t1\_2\_mem\_check, t1\_1\_port\_check* | mem\_wr\_range, rd\_wr\_pseudo  *t1\_1\_stack\_check* |
| **t1\_loop.stack2** | dout,fwrd,padr,derr  *t1\_2\_stack\_dout\_check* | wr\_range  *t1\_2\_port\_check* | mem\_rw\_range,  *t1\_2\_stack\_check* |
| **t2s\_loop.align** | dout,fwrd,padr  *t2s\_align\_dout\_check* | mem, wr\_range, mem\_chk  *t2s\_mem\_check, t2s\_port\_check* | mem\_wr\_range, rd\_wr\_pseudo  *t2s\_align\_check* |
| **t2s\_loop.stack** | dout,fwrd,padr,derr  *t2s\_stack\_dout\_check* | wr\_range  *t2s\_port\_check* | mem\_wr\_range, rd\_wr\_pseudo  *t2s\_stack\_check* |
| **t2d\_loop.align** | dout,fwrd,padr  *t2s\_align\_dout\_check* | mem, wr\_range  *t2d\_mem\_check, t2d\_port\_check* | mem\_wr\_range, rd\_wr\_pseudo  *t2d\_align\_check* |
| **t2d\_loop.stack** | dout,fwrd,padr,derr  *t2s\_stack\_dout\_check* | wr\_range  *t2d\_port\_check* | mem\_wr\_range, rd\_wr\_pseudo  *t2d\_stack\_check* |