Verification of a FIFO MEMORY

(First In First Out)

Description:

FIFO buffer operates as follows:

Write Operation:

- When writeEn is asserted (set to 1) and the FIFO is not full (full is 0), the data present on writeData is written into the memory array in one clock cycle at the location indicated by wrPtr.
- The write pointer (wrPtr) is then incremented by one in the next clock cycle to point to the next write location.
- If the write pointer reaches the end of the memory array, it wraps around to the beginning, maintaining the circular nature of the FIFO.

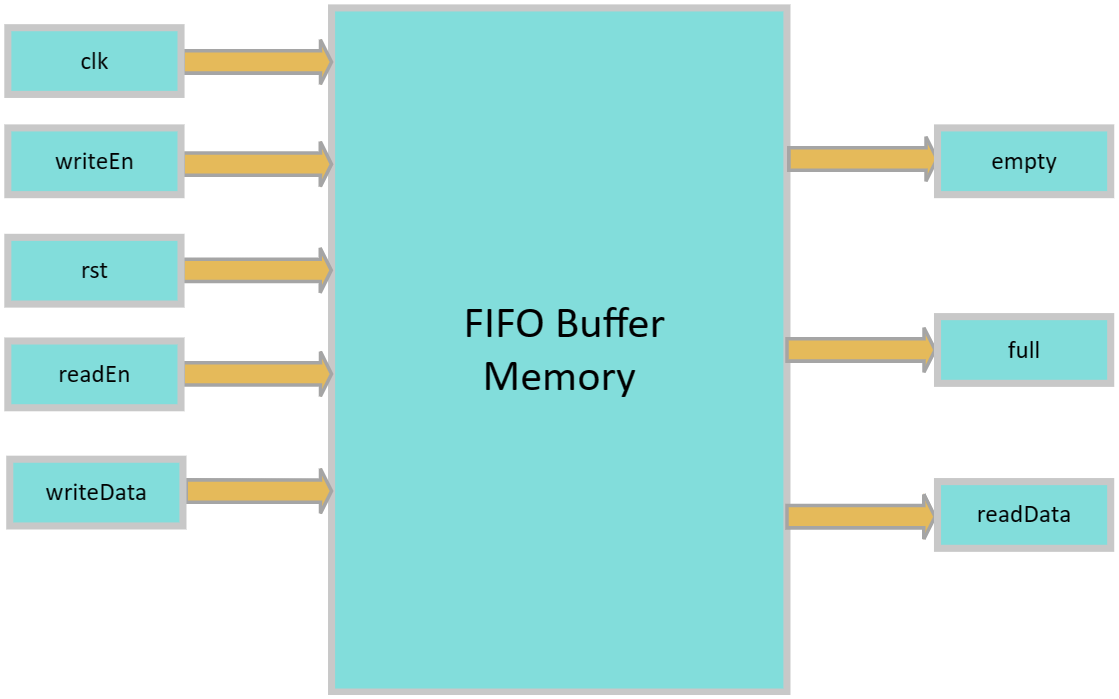
Read Operation:

- When readEn is asserted (set to 1) and the FIFO is not empty (empty is 0), the data at the location indicated by rdPtr is read out and presented on readData in a combinational way using continuous assignment.
- The read pointer (rdPtr) is then incremented by one in the next clock cycle to point to the next read location.
- Like the write pointer, if the read pointer reaches the end of the memory array, it wraps around to the beginning.

Full and Empty Flags:

- Empty Flag (empty): This is one when no write operation was performed and when the same amount of write and read operations was performed.
- Full Flag (full): If the difference between the write and read operations is equal to the memory size, the full flag is active.

Block Diagram:



Input Definitions

- clk: Clock signal.
- rst: Reset signal.
- writeEn: Write enable signal.
- writeData: Data input for writing.
- readEn: Read enable signal.

Output Definitions.

- readData: Data output for reading.
- full: Full flag indicating if the FIFO is full.
- empty: Empty flag indicating if the FIFO is empty.

Internal Components:

- Memory Array: Stores the data elements.
- Write Pointer: Points to the current write location.
- Read Pointer: Points to the current read location.
- Full Flag: Indicates when the FIFO is full.
- Empty Flag: Indicates when the FIFO is empty.

Formal Verification:

Properties

N#	Name	Category	Status
1	full_notWriteEn	Assertion	PASS
2	empty_notReadEn	Assertion	PASS
3	wrPtr_increm_writeEn_on	Assertion	PASS
4	rdPtr_increm_rdEn_on	Assertion	PASS
5	wrPtrNext_increm_writeEn_on	Assertion	PASS
6	rdPtrNext_increm_rdEn_on	Assertion	PASS
7	wr_en_off_wr_ptr_stable	Assertion	PASS
8	rd_en_off_rd_ptr_stable	Assertion	PASS
9	rst_rdPtr_wrPtr_zero	Assertion	PASS
10	~~rst_readEnOff_until_writeEn_on~~	~~Assertion~~	~~CODED~~
11	never_full_and_empty	Assertion	PASS
12	wrPtrNext_maxvalue_reset0	Assertion	PASS
13	rdPtrNext_maxvalue_reset0	Assertion	PASS
14	wrPtr_maxvalue_reset0	Assertion	PASS
15	rdPtr_maxvalue_reset0	Assertion	PASS
16	empty_on_whenreset	Assertion	PASS
17	full_off_whenreset	Assertion	PASS
18	write_correctly	Assertion	PASS
19	read_correctly	Assertion	PASS
20	fifo_stable_when_writeEnoff	Assertion	PASS
1	writeEnoff_rst_on	Assume	PASS
2	readEnoff_rst_on	Assume	PASS
3	readEnoff_empty	Assume	PASS
4	writeEnoff_full	Assume	PASS
1	fifo_full	Cover	PASS
2	fifo_empty	Cover	PASS
3	fifo_notFull	Cover	PASS
4	fifo_notEmpty	Cover	PASS
5	write_all_address	Cover	PASS
6	read_all_address	Cover	PASS
7	writeEn_fifo_full	Cover	X
8	readEn_fifo_empty	Cover	X
9	write_and_read	Cover	PASS
10	write_and_read_mem_full	Cover	X
11	write_and_read_mem_empty	Cover	X
12	fifo_full_no_full	Cover	PASS
13	fifo_empty_no_empty	Cover	PASS

Verify the correct write operation.

Properties defined to cover this spec:

Assumptions:

- Assume write enable is not active when full is active: writeEnoff_full.
- Assume write enable is not active when rst is active: writeEnOff_rst_on.

Assertions:

- The property assures that if FIFO is full then write enable signal must not be active: full_notWriteEn.
- This property verifies writeData was written correctly when the writeEn is activated: write_correctly.
- The property assures that write pointer increments when a write operation happens: wrPtr_increm_writeEn_on.
- The property assures that wrPtrNext increments when a write operation happens: wrPtrNext_increm_writeEn_on.
- The property assures that when wrPtr reaches max value wraps around to 0: wrPtr_maxvalue_reset0.
- The property assures that when wrPtrNext reaches max value wraps around to 0: wrPtrNext_maxvalue_reset0.
- The property assures that FIFO memory value is stable if writeEn is not active: fifo_stable_when_writeEnoff.
- The property assures that wrPtr is stable if a write doesn't occur: wr_en_off_wr_ptr_stable.
- The property assures that after reset the read and write pointers must have the same value and be 0: rst_rdPtr_wrPtr_zero.

Covers:

- All the memory was written: write_all_address.
- What if writeEn is active while FIFO is full: writeEn_fifo_full.

Verify the correct read operation:

Properties defined to cover this spec:

Assumptions:

- Assume write enable is not active when full is active: readEnoff_empty.
- Assume write enable is not active when rst is active: readEnOff_rst_on.

Assertions:

- The property assures that if FIFO is empty then read enable signal must not be active: empty_notReadEn.
- This property verifies readData was read correctly when the readEn is activated: read_correctly.
- The property assures that read pointer increments when a read operation happens: rdPtr_increm_readEn_on.
- The property assures that rdPtrNext increments when a read operation happens: rdPtrNext_increm_readEn_on.
- The property assures when rdPtr reaches max value wraps around to 0: rdPtr_maxvalue_reset0.
- The property assures that when rdPtrNext reaches max value wraps around to 0: rdPtrNext_maxvalue_reset0.
- The property assures that rdPtr is stable if a read doesn't occur: rd_en_off_rd_ptr_stable.
- After reset is active read enable is off until a write operation happens: This property is evaluated through other assertions, we make sure that reading never happens when the memory is empty, and also make sure that the empty flag is initialized high when resetting.

Covers

- All the memory was read: read_all_address.
- What if readEn is active while FIFO is empty: readEn_fifo_empty.

Verify the correct full signal:

Assertions:

- The property assures that full signal is off after reset: full_off_whenreset.
- The full and empty flags can never be active at the same time: never_full_and_empty.

Covers:

- Cover that the FIFO becomes full: fifo_full.
- Cover that the write enable signal is asserted when the FIFO is not full: fifo_notFull.
- Cover a sequence when FIFO becomes full and then no full: fifo_full_no_full.

Verify the correct empty signal:

Assertions:

- The property assures that empty signal is active after reset: empty_on_whenreset.

Covers:

- Cover that the FIFO becomes empty: fifo_empty.
- Cover that the read enable signal is asserted when the FIFO is not empty: fifo_notEmpty.
- Cover a sequence when FIFO becomes empty and then no empty: fifo_empty_no_empty.

Verify the correct simultaneous write and read operation:

Properties defined to cover this spec:

Covers:

- Read and write at the same time: write_and_read.
- Read and write at the same time while the memory is full: write_and_read_mem_full.
- Read and write at the same time while the memory is empty: write_and_read_mem_empty.

Results

Time and bound results were extracted from the formal verification tool. The values of these data were recorded in the tables below and then the time taken for each assertion and cover was plotted. The property number corresponds to those assigned in the section of properties . The verification was performed in 3 cases with different memory size parameters:

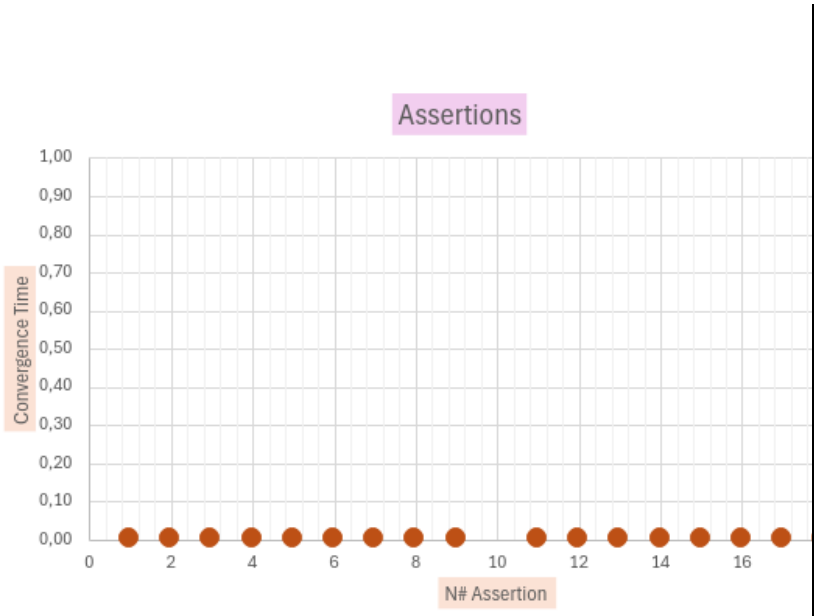
- Depth: 8 – DataWidth: 32,
- Depth: 128 – DataWidth: 64,
- Depth: 256 – DataWidth: 64

In all cases tested the assertions passed and only the coverage properties that are

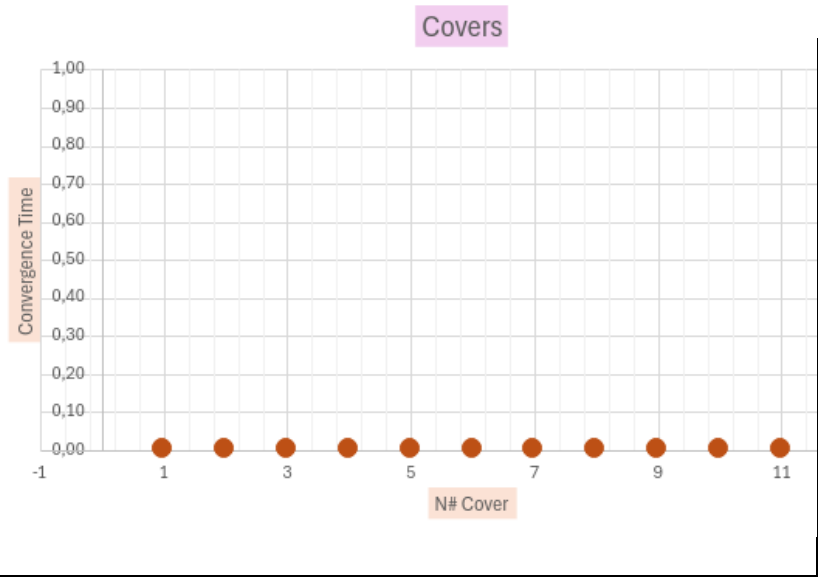
limited the assumes were not met.

8 x 32 memory:

N # Assert	Time	Bound
1	0.00	Infinite
2	0.00	Infinite
3	0.00	Infinite
4	0.00	Infinite
5	0.00	Infinite
6	0.00	Infinite
7	0.00	Infinite
8	0.00	Infinite
9	0.00	Infinite
11	0.00	Infinite
12	0.00	Infinite
13	0.00	Infinite
14	0.00	Infinite
15	0.00	Infinite
16	0.00	Infinite
17	0.00	Infinite
18	0.00	Infinite
19	0.00	Infinite
20	0.00	Infinite

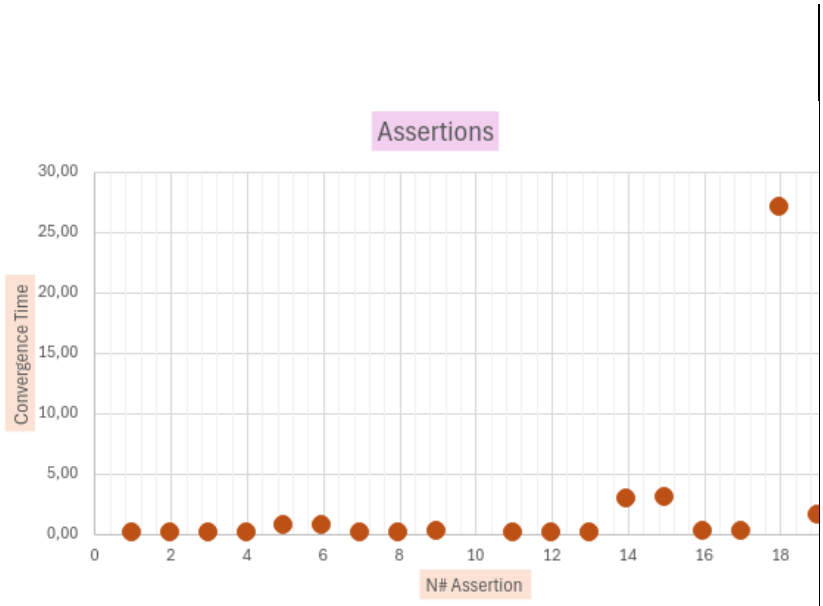


N # Cover	Time	Bound
1	0.00	2 – 10
2	0.00	1
3	0.00	2
4	0.00	2 - 3
5	0.00	2
6	0.00	2 - 3
7	0.00	Infinite
8	0.00	Infinite
9	0.00	2 - 3
10	0.00	Infinite
11	0.00	Infinite
12	0.00	11
13	0.00	2 - 4

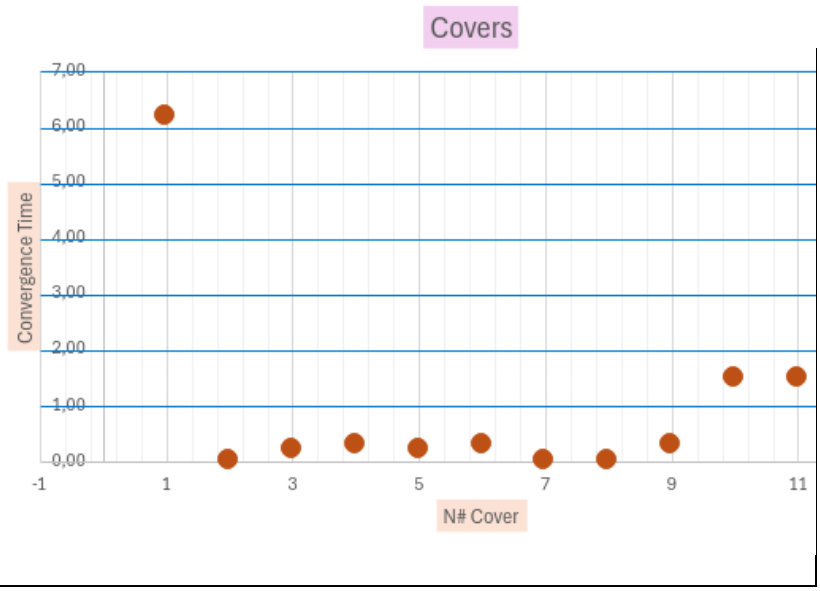


128 x 64 memory:

N # Assert	Time	Bound
1	0.00	Infinite
2	0.00	Infinite
3	0.00	Infinite
4	0.10	Infinite
5	0.60	Infinite
6	0.60	Infinite
7	0.00	Infinite
8	0.00	Infinite
9	0.20	Infinite
11	0.00	Infinite
12	0.00	Infinite
13	0.00	Infinite
14	2.90	Infinite
15	3.00	Infinite
16	0.20	Infinite
17	0.20	Infinite
18	27.10	Infinite
19	1.50	Infinite
20	27.10	Infinite

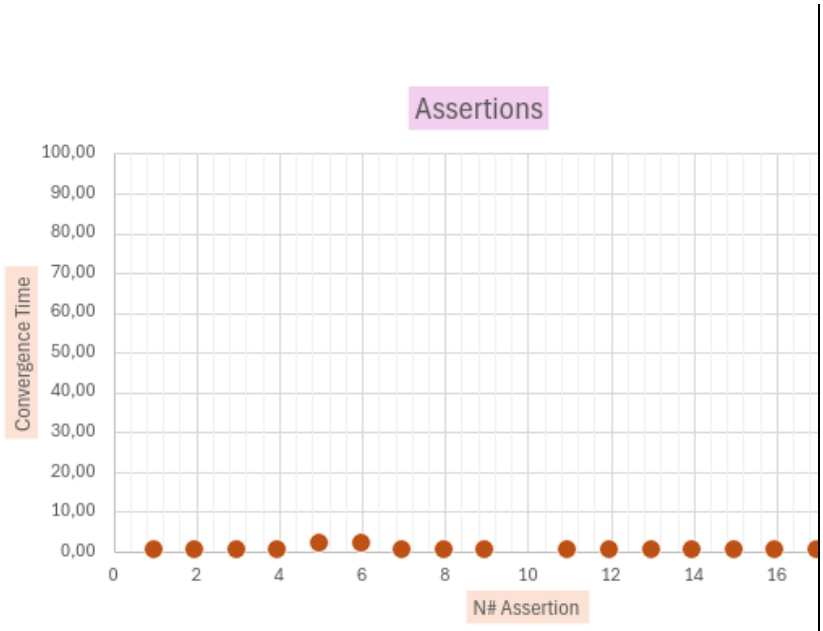


N # Cover	Time	Bound
1	6.20	41 – 130
2	0.00	1
3	0.20	2
4	0.30	3
5	0.20	2
6	0.30	3
7	0.00	Infinite
8	0.00	Infinite
9	0.30	3
10	1.50	Infinite
11	1.50	Infinite
12	5.50	131
13	0.40	4

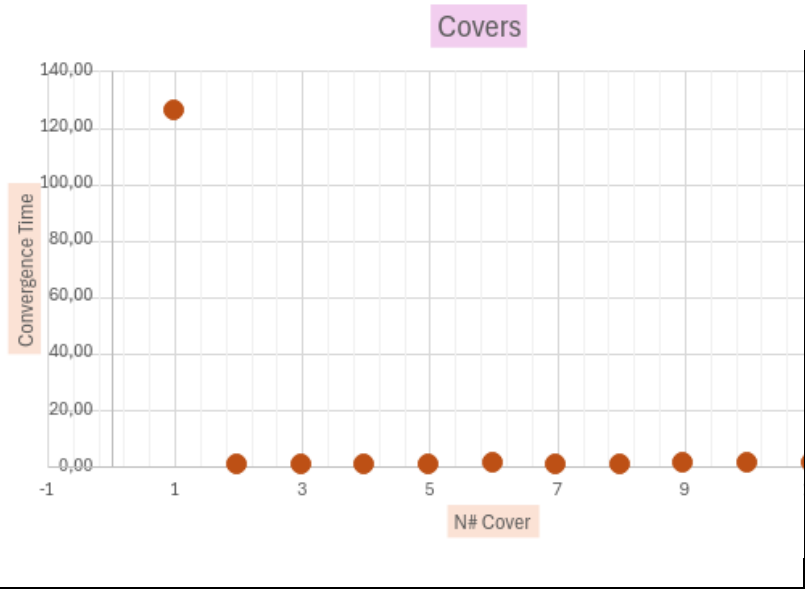


256 x 64 memory:

N # Assert	Time	Bound
1	0.00	Infinite
2	0.00	Infinite
3	0.10	Infinite
4	0.10	Infinite
5	1.60	Infinite
6	1.80	Infinite
7	0.10	Infinite
8	0.10	Infinite
9	0.30	Infinite
11	0.00	Infinite
12	0.10	Infinite
13	0.10	Infinite
14	0.10	Infinite
15	0.10	Infinite
16	0.30	Infinite
17	0.30	Infinite
18	94.60	Infinite
19	5.20	Infinite
20	94.60	Infinite



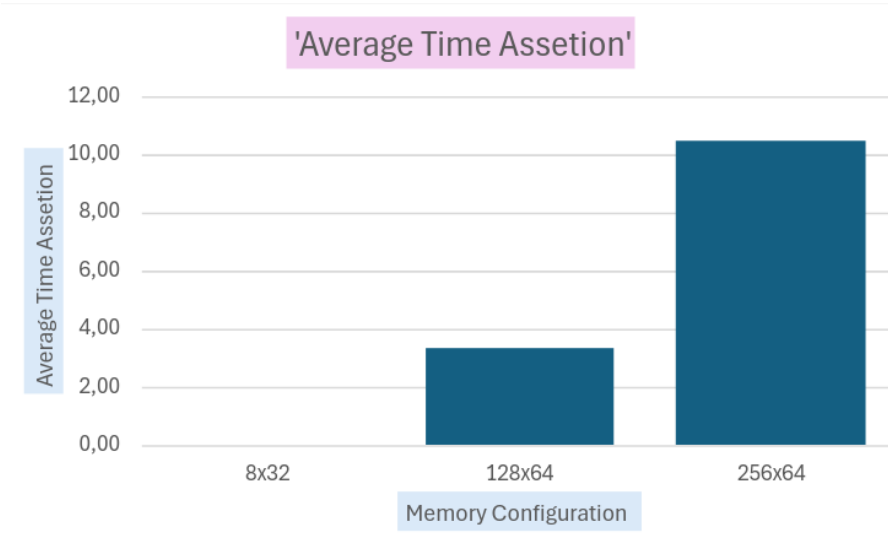
N # Cover	Time	Bound
1	125.70	258
2	0.00	1
3	0.20	2
4	0.50	3
5	0.40	2
6	0.60	3
7	0.00	Infinite
8	0.00	Infinite
9	0.70	3
10	0.80	Infinite
11	0.80	Infinite
12	126.00	259
13	0.8	4

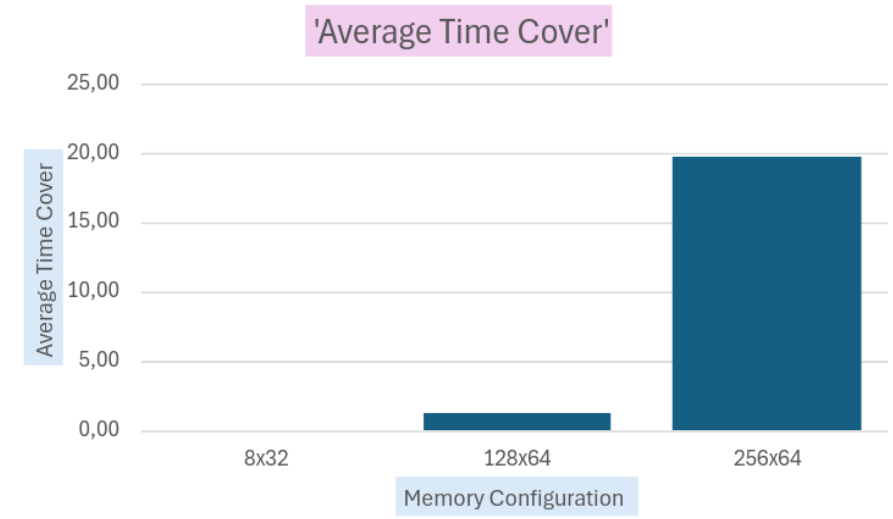


Time average:

Memory Config	Average Time Assertion
8x32	0,00
128x64	3,34
256x64	10,50

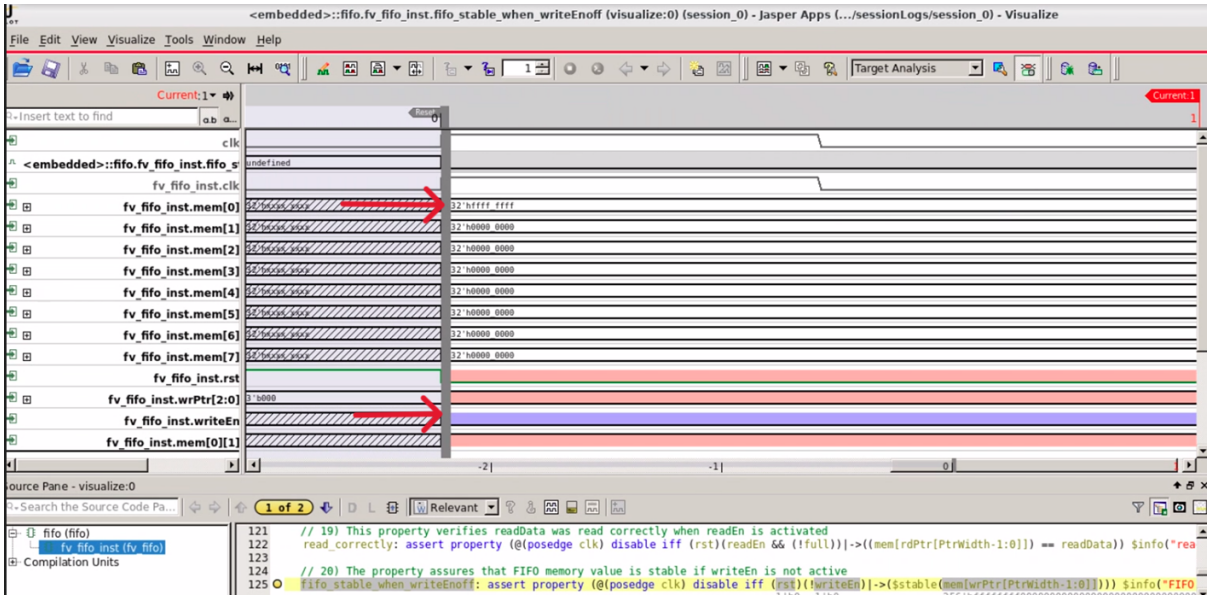
Memory Config	Average Time Cover
8x32	0,00
128x64	1,26
256x64	19,73





Bug fixing

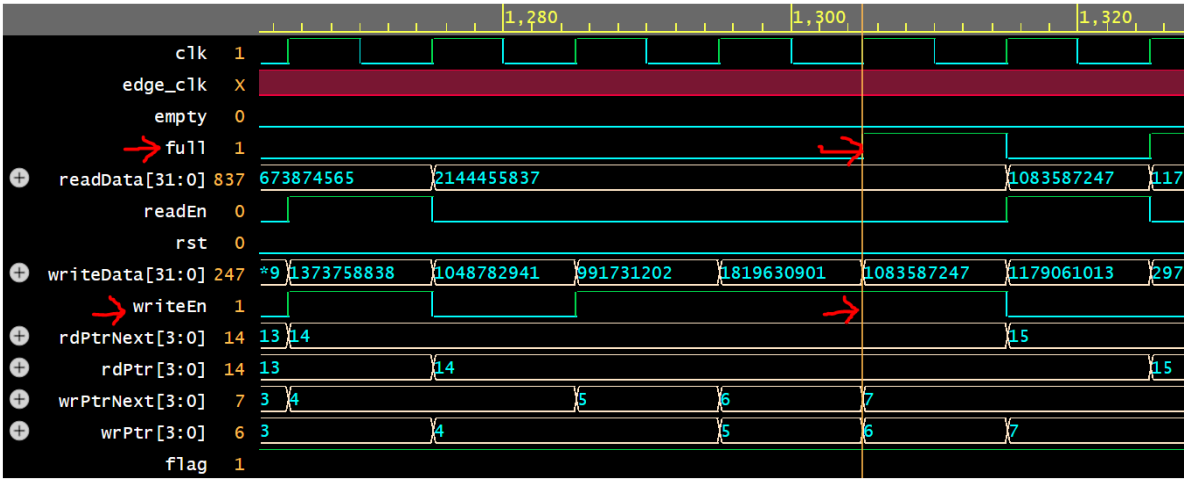
There was a bug in the RTL behavior that was found during verification when the fifo_stable_when_writeEnoff property failed.



The next feature is in the specification:

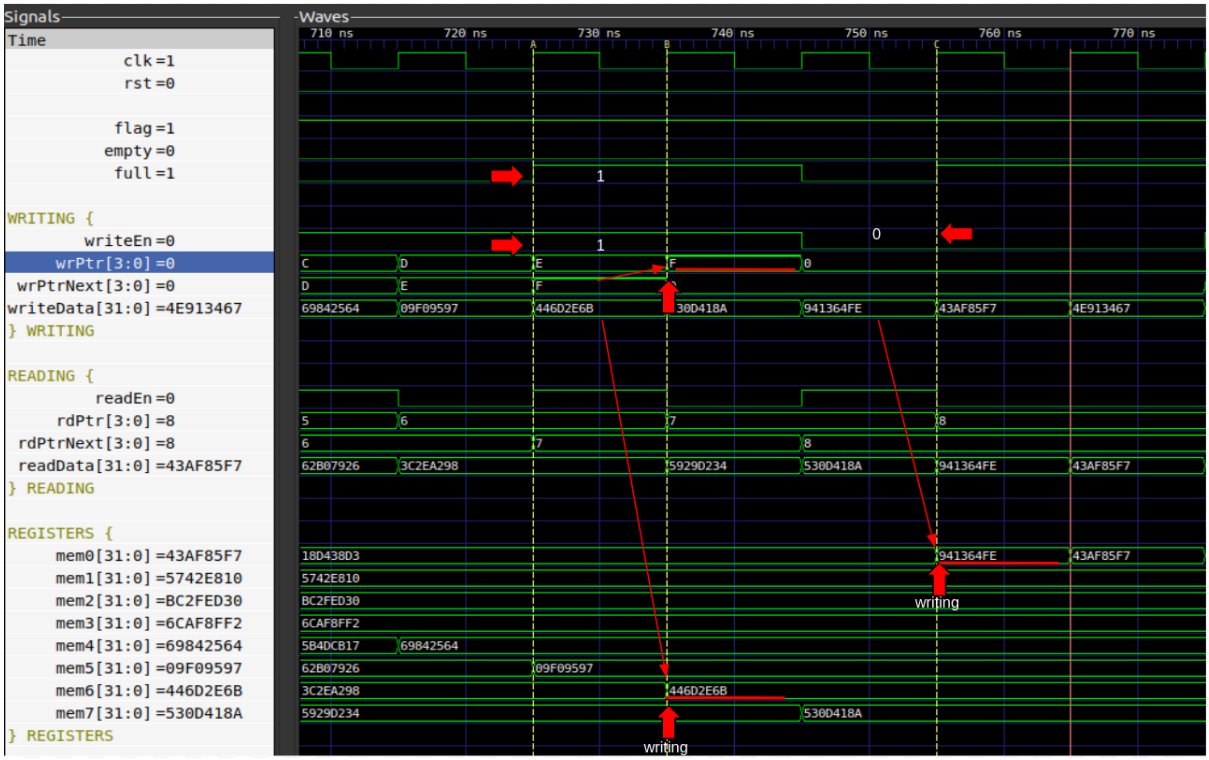
- When writeEn is asserted (set to 1) and the FIFO is not full (full is 0), the data present on writeData is written into the memory array in one clock cycle at the location indicated by wrPtr.

Therefore, we know that writing must occur only when writeEn is asserted and the full flag isn't enabled. This behavior isn't shown in the screenshot below. We can see how the full flag and writeEn signal are activated at the same time and then, on the next clock posedge, writePtr increments and the writeData that shouldn't have been written is read.



This information was reported as an issue in the project repository so that the designer could fix it.

The first step as the designer was run an own simulation to find this bug. In the screenshot below some examples of spec violations are shown:



We can observe the same case that the verification engineer reported in the issue: a write operation is performed when the full flag is active. However, there is also an example of the fifo_stable_when_writeEnoff property violation: a write operation is performed when the writeEn signal isn't active. Although, in this last case the pointer does not increment, this behavior is not expected according to the specification.

Once the wrong behavior is identified, it is necessary to find the code in the RTL that is responsible for controlling this operation. The following code fragment shows this:

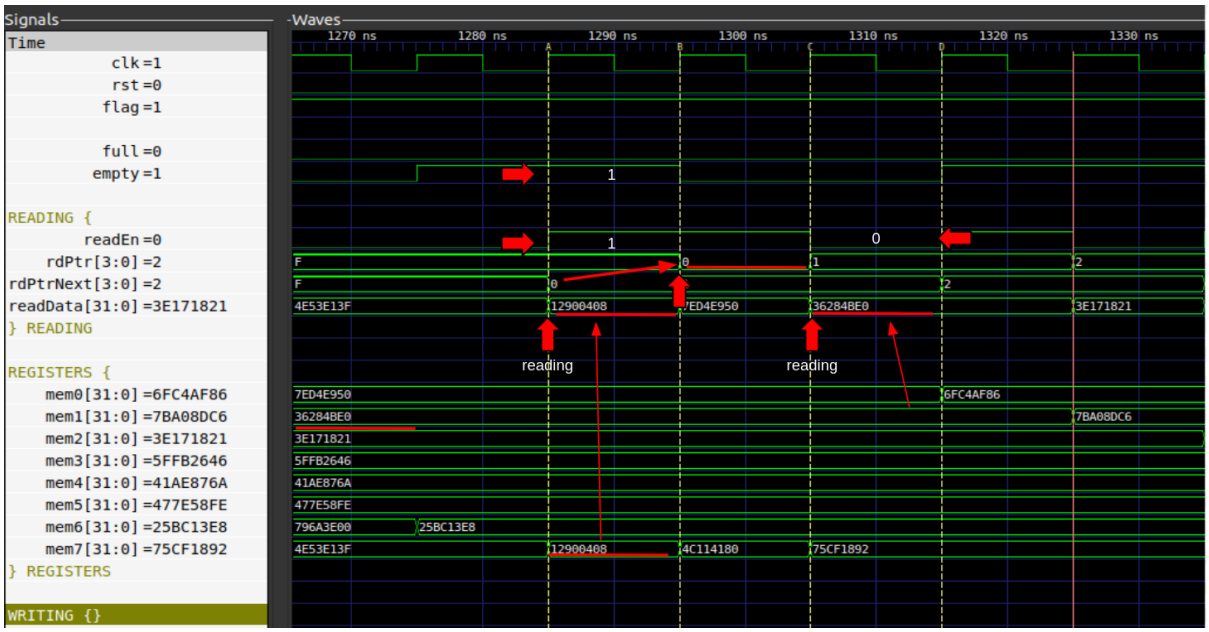
```
always_comb begin
  wrPtrNext = wrPtr;
  rdPtrNext = rdPtr;
  if (writeEn) begin
    wrPtrNext = wrPtr + 1;
  end
  if (readEn) begin
    rdPtrNext = rdPtr + 1;
  end
end
```

```
always_ff @(posedge clk or posedge rst) begin
  if (rst) begin
    wrPtr <= '0;
    rdPtr <= '0;
  end else begin
    wrPtr <= wrPtrNext;
    rdPtr <= rdPtrNext;
  end

   mem[wrPtr[PtrWidth-1:0]] <= writeData;
  end

  assign readData = (mem[rdPtr[PtrWidth-1:0]]);
```

Based on the code, we can expect the same erroneous behavior in the read operation. For this reason, this case was also tested by the designer and is shown in the following screenshot of the simulation:



As expected, reading is also performed even when the empty flag is set and when readEn is disabled.

Solution:

To solve both problems, modifications were made to the RTL.

The first one was to condition the increase of the pointers to the state of the full and empty flags respectively. This change is shown below:

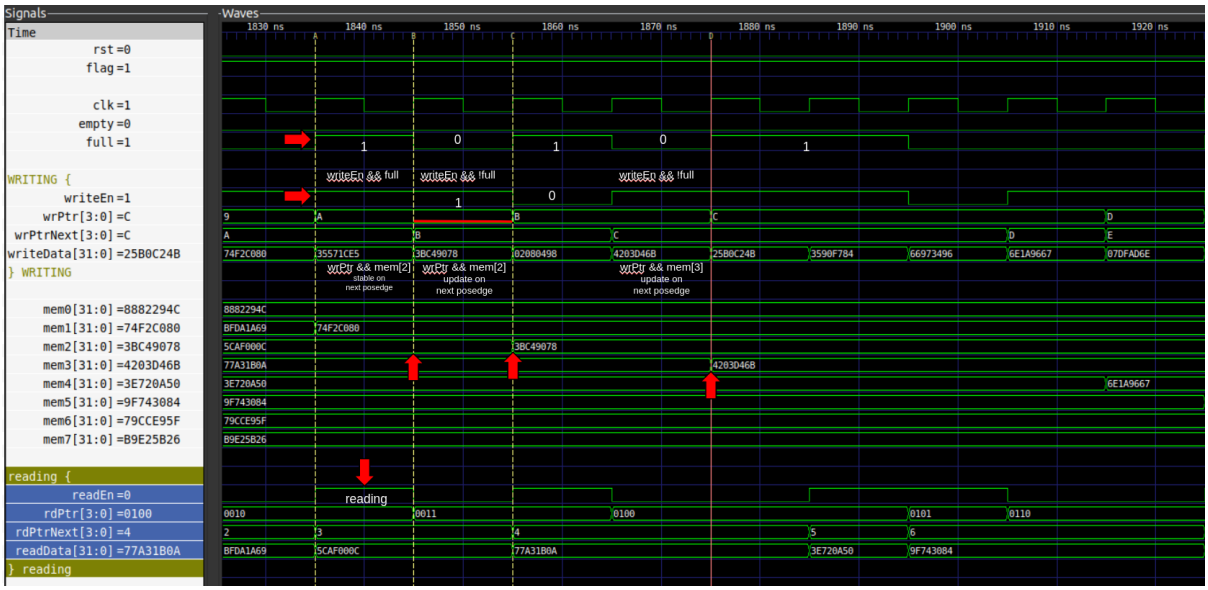
```
always_comb begin
        wrPtrNext = wrPtr;
        rdPtrNext = rdPtr;
        if (writeEn && (!full)) begin
                wrPtrNext = wrPtr + 1;
        end
        if (readEn && (!empty)) begin
                rdPtrNext = rdPtr + 1;
        end
end
```

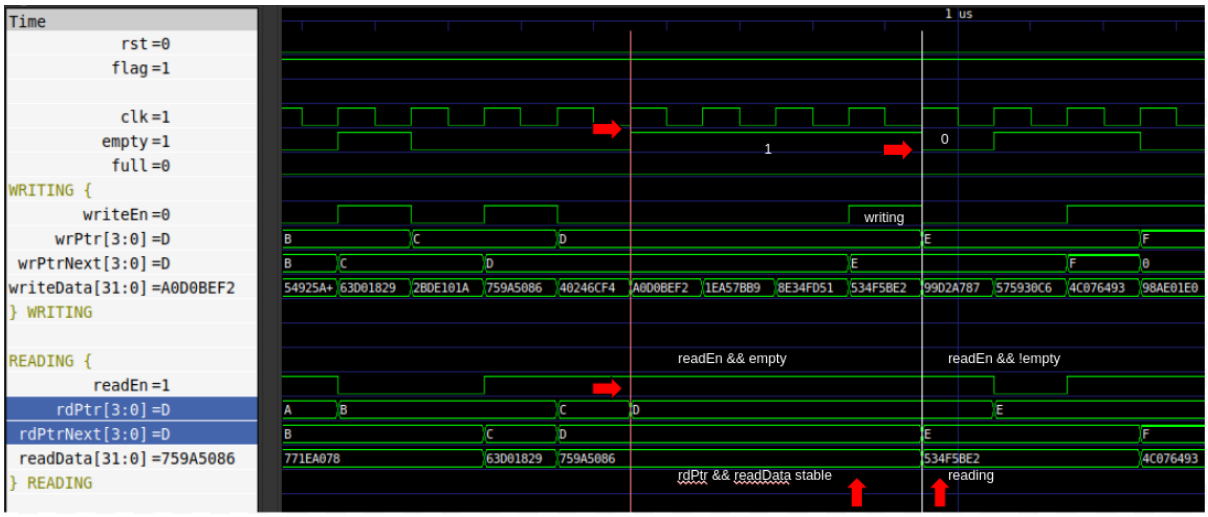
Then, the write and read operations were put in individual process (always) and both were condition to the full/empty flags and the write/read enables respectively.

```
always_ff @(posedge clk or posedge rst) begin
        if(rst) begin
                mem[wrPtr[PtrWidth-1:0]] <= mem[wrPtr[PtrWidth-1:0]];
        end else begin
                if (writeEn && !full) mem[wrPtr[PtrWidth-1:0]] <= writeData;
        end
end

always_comb begin
        if(readEn && (!empty)) readData = (mem[rdPtr[PtrWidth-1:0]]);
        else readData =(readData);
end
```

Once the changes were made to the code, random cases were simulated and in the following waveforms the correct behavior of the desing is observed in these situations previously presented.





It was also verified that the property passed in the formal tool.