

RTL Design Document

Asynchronous FIFO with Credit-Based Flow Control

`async_fifo_credit.sv`

Module `async_fifo_credit`
Version 1.1
Language SystemVerilog (IEEE 1800-2017)
Target ASIC / FPGA
Resets Active-low asynchronous
Date February 15, 2026

Contents

1 Overview	2
2 Architecture	2
2.1 Block Diagram	2
2.2 Module Hierarchy	3
3 Parameters	4
4 Port List	4
4.1 Write Domain	4
4.2 Read Domain	4
5 Depth Formula	5
6 CDC Paths	5
6.1 Summary	5
6.2 Path 1 – Write Pointer to Read Domain	5
6.3 Path 2 – Read Pointer to Write Domain	5
6.4 Path 3 – Credit Toggle to Write Domain	6
6.5 Why Gray Code is Safe for Multi-Bit CDC	6
7 Toggle Frequency Constraint	7
7.1 Failure Mechanism	7
7.2 Constraint Derivation	7
7.3 Constraint Summary	7
7.4 Enforcement	7
8 Functional Blocks	7
8.1 RAM	7
8.2 Write Pointer	8
8.3 Read Pointer and Empty Flag	8
8.4 Full Flag	8
8.5 Credit Toggle and Pulse	8
9 Internal Signal Reference	10
10 Timing and Reset	10
10.1 Reset	10
10.2 STA Constraints	10
11 Metastability	11
11.1 MTBF	11
11.2 Gray Code Guarantee	11
12 Verification	11
12.1 Testbench Scenarios	11
12.2 Formal Properties	11
13 Synthesis Notes	12

1 Overview

`async_fifo_credit` transfers data between two independent clock domains using a credit-based write interface and a standard valid/ready read interface. Gray-coded pointers and 2-FF synchronizers protect every CDC path.

- Source-controlled write side – credit protocol prevents overflow; `wr_full` is a hardware back-stop only
- Gray-coded PTR_W-bit pointers – one bit changes per step, safe for 2-FF CDC
- 2-FF synchronizer on every CDC path – no combinational cross-domain logic
- Depth sized from round-trip latency: $RT_TOTAL=10 \rightarrow DEPTH=16$
- Independent active-low async resets per domain
- Toggle credit return – requires $f_{rd}/f_{wr} < 2$ (back-to-back) or < 4 (≥ 1 idle between reads)

2 Architecture

2.1 Block Diagram

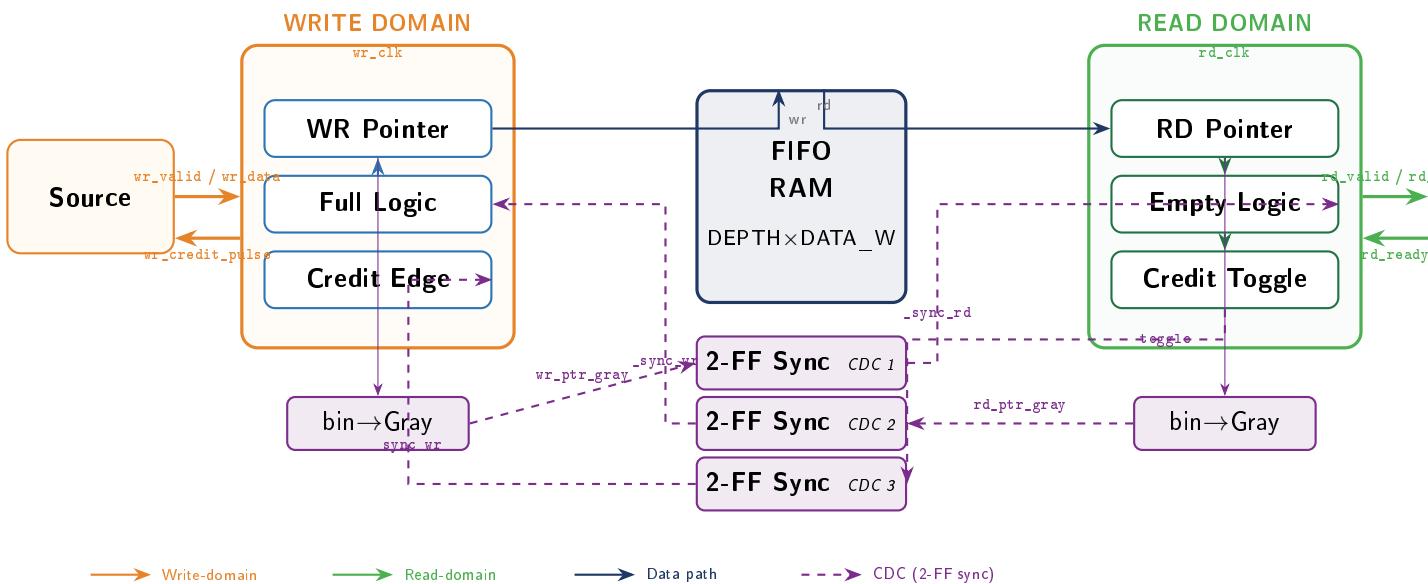


Figure 1: **Source** and **Consumer** are external agents outside their domain boxes. Three dashed CDC paths each pass through a 2-FF synchronizer clocked by the destination domain.

2.2 Module Hierarchy

Module	Instances	Description
cdc_sync	u_sync_wr_ptr u_sync_rd_ptr u_sync_credit	2-FF metastability synchronizer, parameterized width.
async_fifo_cred	top-level	RAM, pointers, full/empty logic, credit toggle, CDC instantiation.

A separate named `cdc_sync` module lets CDC tools (SpyGlass, Questa CDC, JasperGold) identify every crossing by instance name, and lets ASIC flows substitute a hard `sync_cell` macro without touching the top level.

3 Parameters

Parameter	Default	Derived	Description
RT_TOTAL	10	No	Total round-trip latency in wr_clk cycles. Sets minimum DEPTH.
RT_CDC	4	No	CDC round-trip: 2 cycles wr→rd plus 2 cycles rd→wr.
DEPTH	16	No	FIFO depth. Power-of-2 required. Must be \geq RT_TOTAL.
ADDR_W	4	Yes	$\$clog2(DEPTH)$. RAM address bits.
PTR_W	5	Yes	ADDR_W+1. MSB is the wrap bit.
DATA_W	32	No	Data bus width in bits.

4 Port List

4.1 Write Domain

Port	Dir	Width	Description
wr_clk	in	1	Source clock.
wr_rst_n	in	1	Active-low async reset, write domain.
wr_valid	in	1	Source has data. Write accepted on wr_valid && !wr_full.
wr_data	in	DATA_W	Write data bus.
wr_credit_pulse	out	1	One 1-cycle pulse per completed read. Source uses this to replenish its credit counter.

4.2 Read Domain

Port	Dir	Width	Description
rd_clk	in	1	Receiver clock.
rd_rst_n	in	1	Active-low async reset, read domain.
rd_valid	out	1	FIFO non-empty.
rd_ready	in	1	Consumer accepts. Handshake fires on rd_valid && rd_ready.
rd_data	out	DATA_W	Read data, driven combinationally from RAM.

5 Depth Formula

The source pre-loads DEPTH credits and writes one per cycle. Credits cannot return until the full round-trip has elapsed, so the FIFO must absorb DEPTH writes before the first credit returns.

$$D_{\min} = R_{\text{total}} = 10 \text{ cycles} \quad (1)$$

$$\text{DEPTH} = 2^{\lceil \log_2 10 \rceil} = 2^4 = 16 \quad (2)$$

Power-of-2 is required for correct Gray-code pointer arithmetic. The 6 extra entries (16–10) absorb back-pressure, synchronizer jitter, and pipeline registers.

Cycle	Event	Credits in flight	FIFO entries
0	Burst begins, DEPTH credits held	0	0
1–10	1 write/cycle, credits in flight	up to 10	1–10
10	First wr_credit_pulse	10	10
11+	Credits return 1/cycle	≤ 10	≤ 10

6 CDC Paths

6.1 Summary

#	Signal	Width	Direction	Sync clock	Purpose
1	wr_ptr_gray	PTR_W	wr→rd	rd_clk	Empty detection
2	rd_ptr_gray	PTR_W	rd→wr	wr_clk	Full detection
3	credit_toggle_rd	1	rd→wr	wr_clk	Credit pulse

6.2 Path 1 – Write Pointer to Read Domain

wr_ptr_bin is Gray-encoded before leaving the write domain. After 2-FF sync into rd_clk:

$$\text{rd_empty} = (\text{rd_ptr_gray} == \text{wr_ptr_gray_sync_rd})$$

All PTR_W bits including the MSB wrap bit must match.

6.3 Path 2 – Read Pointer to Write Domain

Symmetric to Path 1. After sync into wr_clk, Gray is decoded to binary for the full comparison:

$$\text{wr_full} = (\text{addr bits equal}) \wedge (\text{wrap bits differ})$$

Same address slot, different wrap counts – write has lapped read by exactly DEPTH entries.

6.4 Path 3 – Credit Toggle to Write Domain

Every successful read toggles `credit_toggle_rd`. After 2-FF sync into `wr_clk`:

$$\text{wr_credit_pulse} = \text{credit_toggle_sync_wr} \oplus \text{credit_toggle_prev_wr}$$

Why toggle, not pulse?

A single-cycle `rd_clk` pulse may be narrower than one `wr_clk` period and would be missed by the 2-FF sync. A toggle is a persistent level change held for at least 1 full `rd_clk` period – the synchronizer always captures it.

6.5 Why Gray Code is Safe for Multi-Bit CDC

Binary $0111 \rightarrow 1000$ changes four bits at once. A synchronizer sampling mid-transition can land on any of 16 intermediate states. Gray code changes exactly one bit per step – the worst outcome is capturing the old or new valid value, never a phantom between them.

7 Toggle Frequency Constraint

7.1 Failure Mechanism

The 2-FF synchronizer on `wr_clk` needs the toggle stable for **2 wr_clk periods** to guarantee capture. Minimum hold time is **1 rd_clk period** (back-to-back reads). If two reads complete within 2 `wr_clk` periods, the toggle returns to its original value before being sampled – the write domain sees nothing and **one credit is permanently lost**. The FIFO will not overflow, but the source will eventually deadlock.

7.2 Constraint Derivation

$$T_{rd} \geq 2 \times T_{wr} \implies \boxed{\frac{f_{rd}}{f_{wr}} \leq \frac{1}{2}} \quad (\text{back-to-back reads}) \quad (3)$$

With at least one idle cycle guaranteed between every two reads (minimum hold = $2 \times T_{rd}$):

$$\boxed{\frac{f_{rd}}{f_{wr}} \leq 1} \quad (\geq 1 \text{ idle cycle guaranteed}) \quad (4)$$

7.3 Constraint Summary

Consumer read pattern	Max f_{rd}/f_{wr}	Safe?
Back-to-back every cycle	< 2	Yes if ratio < 2
≥ 1 idle cycle guaranteed	< 4	Yes if ratio < 4
f_{wr} faster than f_{rd}	any	Always safe

Constraint violation

If $f_{rd}/f_{wr} \geq 2$ with back-to-back reads, credit pulses are silently dropped. The source stalls permanently even though the FIFO has space.

7.4 Enforcement

The constraint is documented directly on the CDC Path 3 instantiation in the RTL:

```
// CDC PATH 3  credit_toggle -> wr domain
// constraint: f_rd / f_wr < 2 (back-to-back reads)
//             f_rd / f_wr < 4 (>=1 idle cycle between reads guaranteed)
cdc_sync #(.W(1)) u_sync_credit ( ... );
```

8 Functional Blocks

8.1 RAM

Listing 1: Storage and async read

```

1 // storage
2 logic [DATA_W-1:0] mem [0:DEPTH-1];
3
4 // Write Domain
5 always_ff @(posedge wr_clk)
6     if (wr_valid && !wr_full)
7         mem[wr_ptr_bin[ADDR_W-1:0]] <= wr_data;
8
9 assign rd_data = mem[rd_ptr_bin[ADDR_W-1:0]]; // async read

```

8.2 Write Pointer

Listing 2: Write pointer and bin → Gray

```

1 // Write Domain
2 always_ff @(posedge wr_clk or negedge wr_rst_n)
3     if (!wr_rst_n) wr_ptr_bin <= '0;
4     else if (wr_valid && !wr_full) wr_ptr_bin <= wr_ptr_bin + 1'b1;
5
6 assign wr_ptr_gray = wr_ptr_bin ^ (wr_ptr_bin >> 1); // bin -> Gray

```

8.3 Read Pointer and Empty Flag

Listing 3: Read pointer, empty flag, bin → Gray

```

1 // Read Domain
2 assign rd_empty = (rd_ptr_gray == wr_ptr_gray_sync_rd);
3 assign rd_valid = !rd_empty;
4 assign rd_data = mem[rd_ptr_bin[ADDR_W-1:0]];
5
6 always_ff @(posedge rd_clk or negedge rd_rst_n)
7     if (!rd_rst_n) rd_ptr_bin <= '0;
8     else if (rd_valid && rd_ready) rd_ptr_bin <= rd_ptr_bin + 1'b1;
9
10 assign rd_ptr_gray = rd_ptr_bin ^ (rd_ptr_bin >> 1); // bin -> Gray

```

8.4 Full Flag

Listing 4: Gray-to-binary decode and full comparison

```

1 // FULL FLAG (wr_clk)
2 // full: lower bits equal (same slot) AND wrap bits differ (wr lapped rd
3 always_comb begin : gray2bin_rd_in_wr
4     rd_ptr_bin_wr[PTR_W-1] = rd_ptr_gray_sync_wr[PTR_W-1];
5     for (int i = PTR_W-2; i >= 0; i--)
6         rd_ptr_bin_wr[i] = rd_ptr_bin_wr[i+1] ^ rd_ptr_gray_sync_wr[i];
7 end
8
9 assign wr_full = (wr_ptr_bin[ADDR_W-1:0] == rd_ptr_bin_wr[ADDR_W-1:0])
10    && (wr_ptr_bin[ADDR_W] != rd_ptr_bin_wr[ADDR_W]);

```

8.5 Credit Toggle and Pulse

Listing 5: Credit toggle (rd_clk) and edge-detect pulse (wr_clk)

```

1 // CREDIT TOGGLE (rd_clk)
2 always_ff @(posedge rd_clk or negedge rd_rst_n)
3     if (!rd_rst_n) credit_toggle_rd <= 1'b0;
4     else if (rd_valid && rd_ready) credit_toggle_rd <= ~credit_toggle_rd
5         ;
6
7 // CDC PATH 3 credit_toggle -> wr domain
8 // constraint: f_rd / f_wr < 2 (back-to-back reads)
9 //           f_rd / f_wr < 4 (>=1 idle cycle between reads guaranteed)
10 cdc_sync #(.W(1)) u_sync_credit (
11     .clk      (wr_clk),
12     .rst_n   (wr_rst_n),
13     .d       (credit_toggle_rd),
14     .q       (credit_toggle_sync_wr)
15 );
16
17 always_ff @(posedge wr_clk or negedge wr_rst_n)
18     if (!wr_rst_n) credit_toggle_prev_wr <= 1'b0;
19     else           credit_toggle_prev_wr <= credit_toggle_sync_wr;
20
21 assign wr_credit_pulse = credit_toggle_sync_wr ^ credit_toggle_prev_wr;

```

9 Internal Signal Reference

Signal	Dom	Width	Type	Description
wr_ptr_bin	wr	PTR_W	reg	Binary write pointer. [ADDR_W-1:0] = RAM addr; [ADDR_W] = wrap.
wr_ptr_gray	wr	PTR_W	comb	Gray write pointer. Source of CDC Path 1.
rd_ptr_gray_sync_wr	wr	PTR_W	reg	rd Gray ptr after 2-FF sync into wr domain.
rd_ptr_bin_wr	wr	PTR_W	comb	Decoded rd ptr in wr domain. Full comparison.
wr_full	wr	1	comb	Full flag. Gates pointer increment and RAM write.
credit_toggle_sync_rd	wr	1	reg	credit_toggle_rd after 2-FF sync.
credit_toggle_prev_wr	wr	1	reg	Previous sync'd toggle value for XOR edge detect.
rd_ptr_bin	rd	PTR_W	reg	Binary read pointer.
rd_ptr_gray	rd	PTR_W	comb	Gray read pointer. Source of CDC Path 2.
wr_ptr_gray_sync_rd	rd	PTR_W	reg	wr Gray ptr after 2-FF sync into rd domain.
wr_ptr_bin_rd	rd	PTR_W	comb	Decoded wr ptr in rd domain. Occupancy monitor only.
rd_empty	rd	1	comb	Empty flag. <code>rd_valid = !rd_empty</code> .
credit_toggle_rd	rd	1	reg	Toggles once per accepted read. Source of CDC Path 3.
mem	—	D×W	mem	Dual-port FIFO RAM. Sync write, async read.

10 Timing and Reset

10.1 Reset

Each domain has its own independent `rst_n`. The two resets need not be released simultaneously – both domains see empty on release. The RAM is not reset; its contents are never read before a write advances the pointer past them.

10.2 STA Constraints

Listing 6: SDC / XDC false-path constraints

```
# CDC Path 1: wr_ptr_gray -> rd_clk
set_false_path -from [get_cells {wr_ptr_gray_reg[*]}] \
                  -to  [get_cells {u_sync_wr_ptr/s1_reg[*]}]

# CDC Path 2: rd_ptr_gray -> wr_clk
```

```

set_false_path -from [get_cells {rd_ptr_gray_reg[*]}] \
                  -to  [get_cells {u_sync_rd_ptr/si_reg[*]}]

# CDC Path 3: credit_toggle -> wr_clk
set_false_path -from [get_cells {credit_toggle_rd_reg}] \
                  -to  [get_cells {u_sync_credit/si_reg}]

```

11 Metastability

11.1 MTBF

$$\text{MTBF} = \frac{e^{T_r/\tau}}{f_{\text{src}} \cdot f_{\text{dst}} \cdot C} \quad (5)$$

T_r = resolution time, $\tau \approx 30$ ps at 28 nm, $C \approx 10^{-10}$. At 500 MHz in 28 nm, MTBF exceeds 10^9 years. For higher frequencies, increase `cdc_sync` to a 3-stage chain.

11.2 Gray Code Guarantee

A metastable capture on the one transitioning Gray bit results in either the old or new valid pointer – never a phantom value between them. This eliminates the catastrophic pointer corruption possible with raw binary pointers.

12 Verification

12.1 Testbench Scenarios

Scenario	wr_clk	rd_clk	Back-pressure	Words
wr 2× faster	10 ns	20 ns	No	12
rd 2× faster	20 ns	10 ns	No	12
Equal clocks	15 ns	15 ns	No	14
Equal + back-pressure	10 ns	10 ns	Random	10
Async 7:11	7 ns	11 ns	No	12

Each scenario checks data ordering, credit pulse count equals write count, and no underflow.

12.2 Formal Properties

Listing 7: SVA assertions – bind to `async_fifo_credit`

```

1 // wr_full is a backstop: credit protocol should prevent writes to full
   FIFO
2 property p_no_overflow;
3   @ (posedge wr_clk) disable iff (!wr_rst_n)
4     wr_full | -> !wr_valid;
5 endproperty
6 assert property (p_no_overflow);
7
8 // rd_data stable while rd_valid holds and rd_ready not yet asserted

```

```
9 property p_rd_stable;
10    @ (posedge rd_clk) disable iff (!rd_rst_n)
11      (rd_valid && !rd_ready) |=> (rd_valid && $stable(rd_data));
12 endproperty
13 assert property (p_rd_stable);
```

13 Synthesis Notes

FPGA — `mem` infers block RAM in SDP mode. (`* ASYNC_REG = "TRUE" *`) co-locates `s1/s2` in the same slice and adds them to Vivado's CDC report automatically. Apply `set_false_path` on all three CDC paths in XDC.

ASIC — Replace each `cdc_sync` instance with the foundry-characterised `sync_cell` hard macro. Replace `mem` with a RAM macro for area/power. No latches are inferred; all `always_comb` blocks are fully specified.