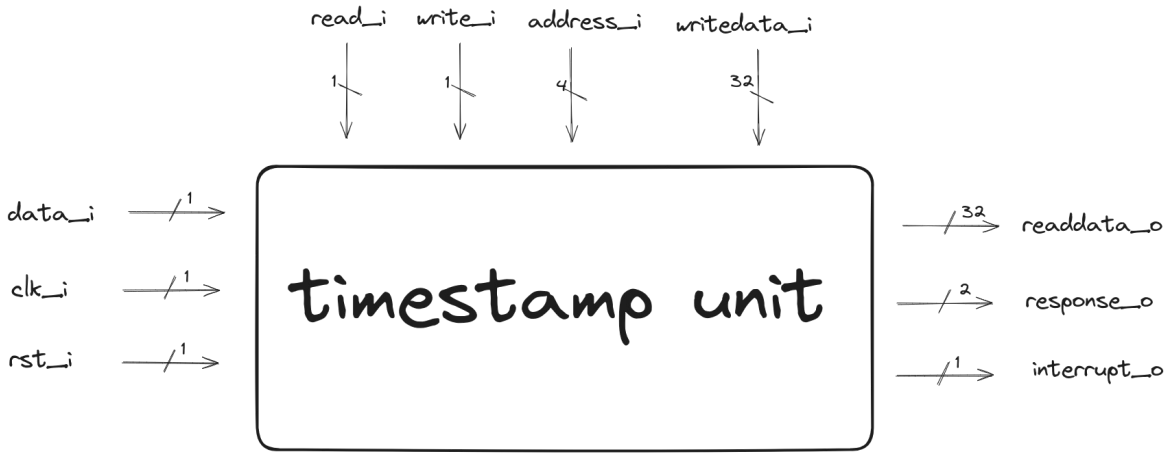
Timestamp unit

Digital System for detection transitions of digital signal

User manual

Table of Contents

1.0 Device Overview



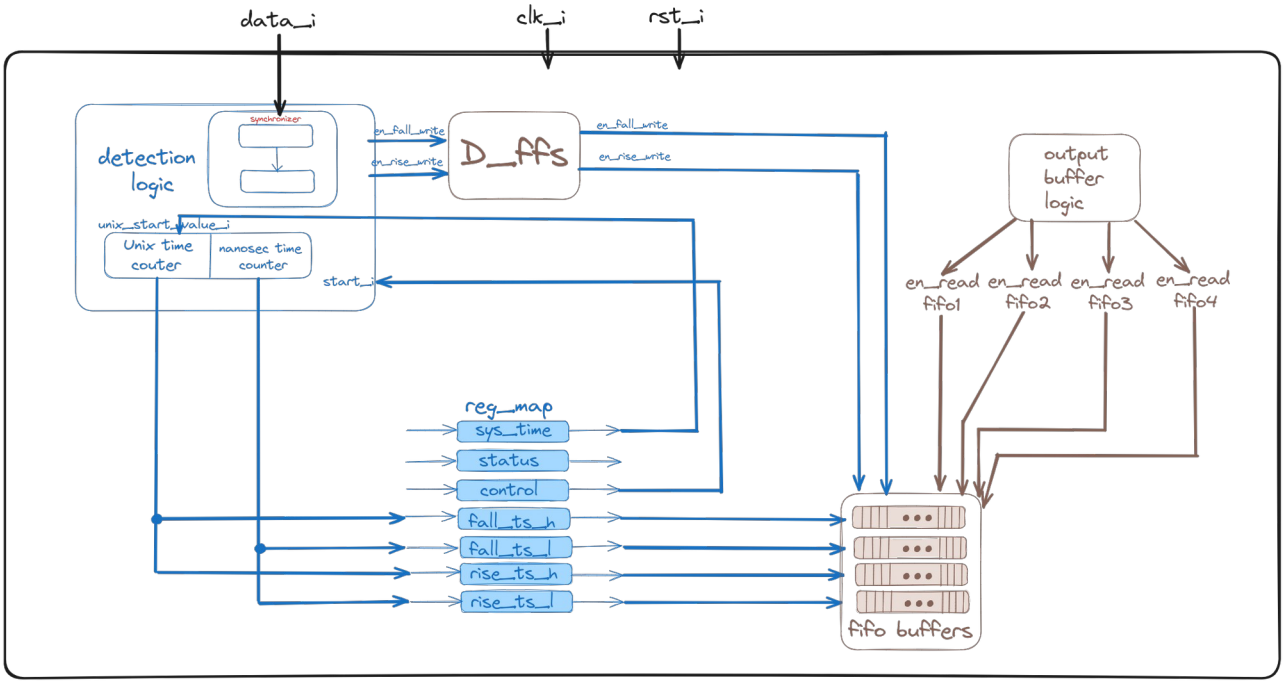
Timestamp unit design has seven inputs and three outputs as presented in the image above.
Inputs :

- **clk_i** : 50MHz clock input signal – 1bit
- **rst_i** : asynchronous reset input signal – 1bit
- **data_i** : asynchronous input signal which transitions are being detected – 1bit
- **read_i** : control input signal for entering read mode – 1bit
- **write_i** : control input signal for entering write mode – 1bit
- **address_i** : control input signal for addressing certain internal register – 4bit
- **writedata_i** : data to be written into the selected(addressed) internal register – 32bit

Outputs :

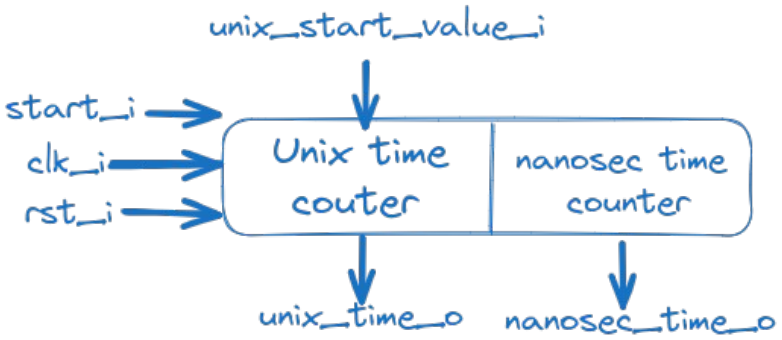
- **readdata_o** : word read from selected(addressed) internal register – 32bit
- **response_o** : notigication output signal
  - response_o = 00 → RESET mode and REGULAR mode
  - response_o = 01 → RESERVED
  - response_o = 10 → ERROR : tried to write into read only register
  - response_o = 11 → ERROR : addressed not-existent memory location
- **interrupt_o** : interrupt logic output (fifo full / empty) – 1bit

2.0 Guidelines through the Timestamp unit architecture



Timestamp unit design is composed of several modules :

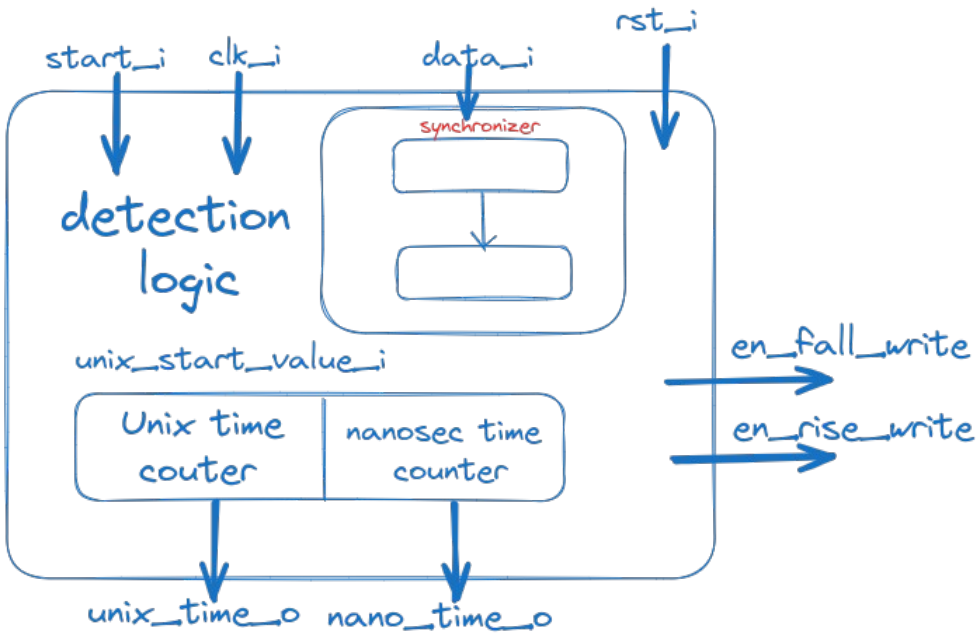
1. **counter module**



Essentially, there are 2 counters:
→ One that counts the number of nanoseconds
→ The other counts Unix time based on those nanoseconds

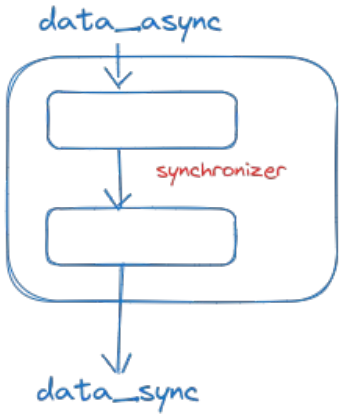
The frequency of the clock signal is defined as 50MHz.
Based on that, the step increment for time is 20ns. The nanosecond counter increments itself by 20 every clock cycle. When it reaches 1000000000, one second has passed in real time, and the Unix counter gets incremented by one, and the nanoseconds counter gets reset

2. detection logic module



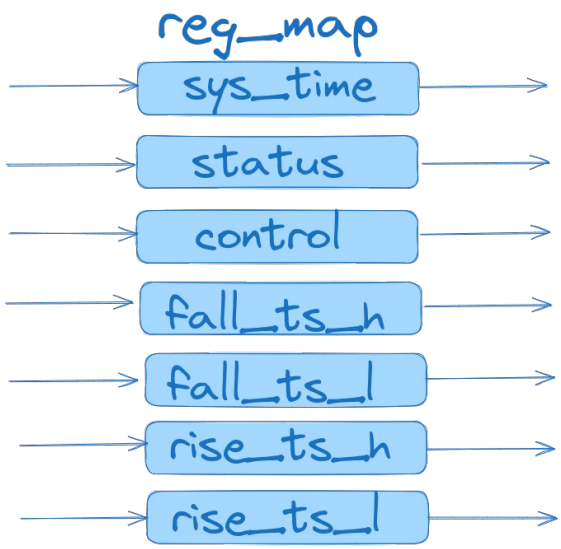
An input data_i single-bit signal which transitions are gonna be detected. Each time transition is detected, 2 outputs (unix_time_o & nano_time_o) are gonna be updated according to the exact time when it's happened (both Unix time moment and nanosec time moment). When transition occurs, 2 enable_write signal are gonna be HIGH in order to write these times into the buffers.

data_i signal is asynchronous, so we use synchronizer in order to achieve full synchronous design.



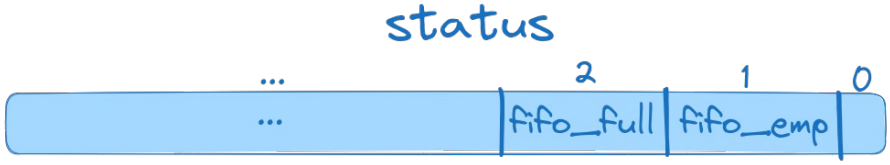
start_i signal enables counting mechanism. If it is LOW – count mechanism is disabled (counter values stay frozzen until HIGH start_i signal occurs).

3. register map



Simple register map consists of 7 registers :

- **sys time** : storing system time (Unix time)
- **status** : fifo_full & fifo_empty flags
- **control** : software reset, interrupt enable, timestamp enable
- **fall_ts_h** : unix time for falling edge
- **fall_ts_l** : nano time for falling edge
- **rise_ts_h** : unix time for rising edge
- **rise_ts_l** : nano time for rising edge



4. fifo buffer

data_i → Data to be written into buffer
wren_i → Enable writing into fifo buffer
rden_i → Enable reading from fifo buffer
data_o → Output data (pull)
rdvalid_o → Data valid output
empty_o → signalizing empty buffer
full_o → signalizing full buffer
fill_count_o → number of elements in buffer

Buffer is size configurable. By default, both number of memory slots and size of each memory slot (in bits) is 32.

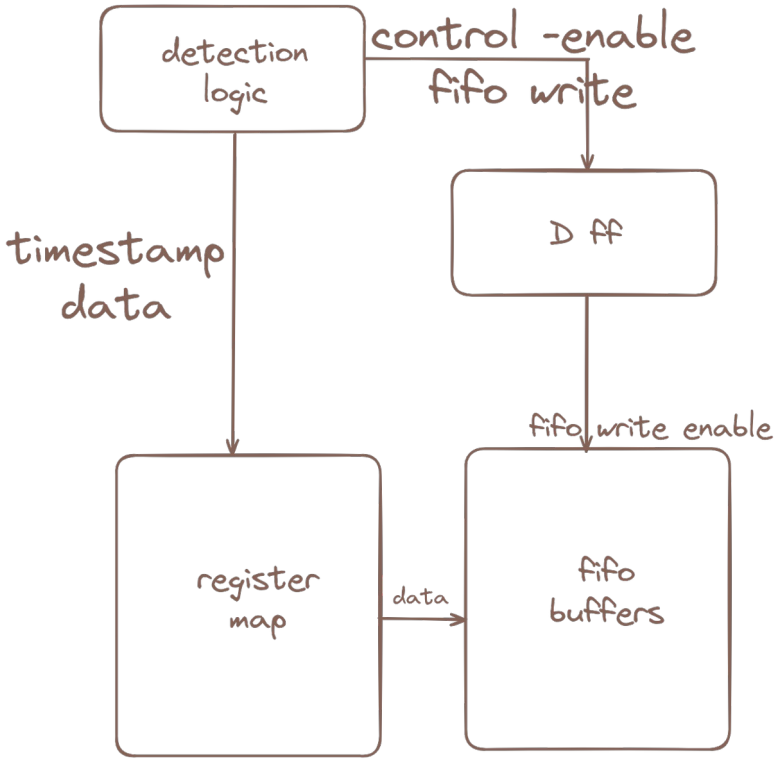
Process for incrementing the head and tail of the buffer :

wren_i = 1 and fifo not full → head gets incremented
wren_i = 1 and fifo full → head gets incremented & old values get overwritten

rden_i = 1 and fifo not empty → tail gets incremented & rdvalid_o = 1 (element is successfully popped out of buffer)

5. additional supporting modules

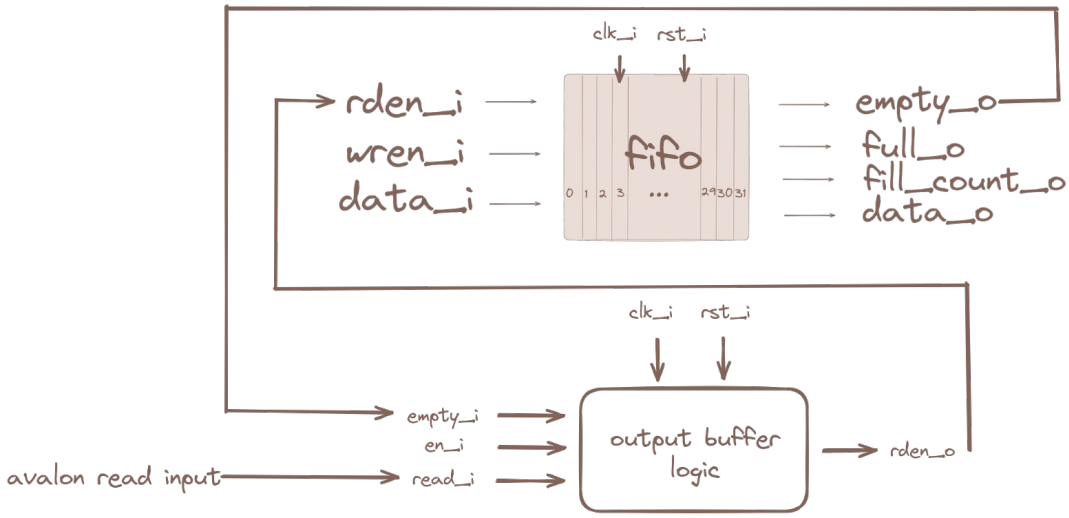
5.1. D flip flop



D ff is delay element in this case. The request for fifo write will occure at the same time as data we want to write into the buffer.

Via D ff, fifo write enable signal are carried over.

5.2. Output Buffer Logic

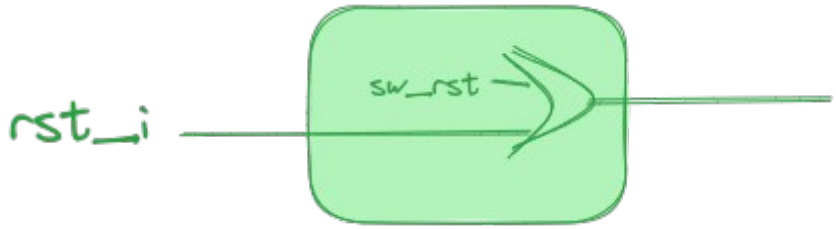


Output buffer logic is nearly related with the fifo design. Two of them work together in order to accomplish push – pop functionality.

If read mode is choosen (avalon read intput → 1) and fifo buffer isn't empty, then the **rden_o** signal from output_buffer_logic is set to 1 which means that **rden_i** of fifo is also 1. Now the next element is being popped from the fifo buffer.

Via Output Buffer Logic, fifo read enable signal are carried over.

3.0 Reset



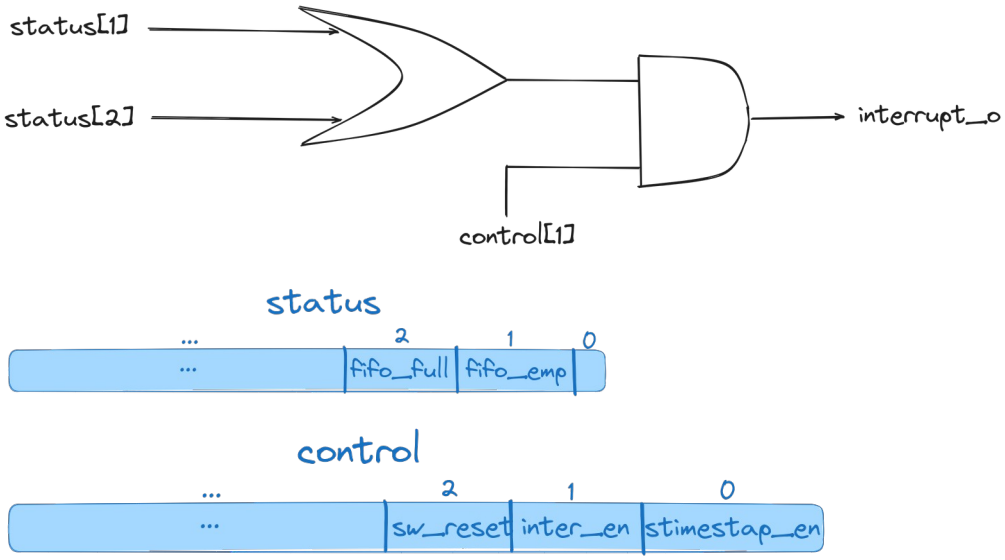
Reset combines hardware and software reset. Hardware reset is external reset signal sent via one of the design's ports. Software reset is control bit from control register (register map).

If at least one of them is 1, the system is going to be reseted.

4.0 Start

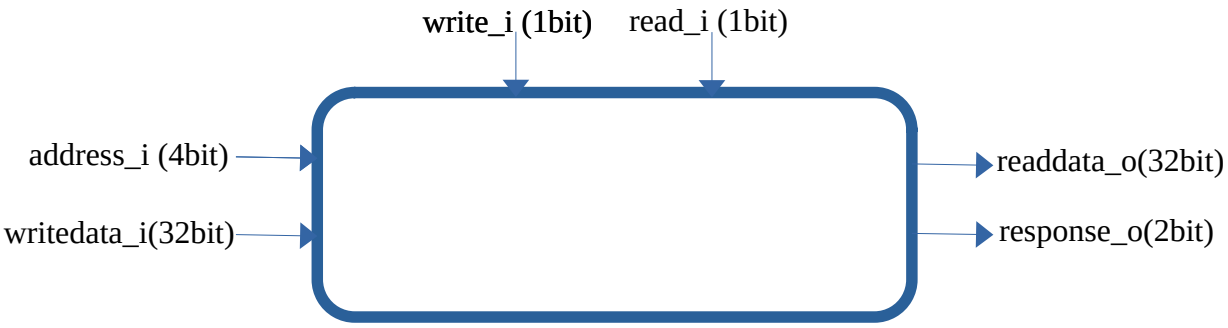
Start signal is not an external signal but internal register flag bit (control[0]). If control[0] is set to HIGH, the counting mechanism is enabled so we have time counting. If it is LOW, counting mechanism is disabled, counter's registers are frozzen.

5.0 Interrupt logic



If at least one of the buffers is full / empty and interrupt signal is enabled, then interrupt_o signal of timestamp design is HIGH. If interrupt signal is disabled, interrupt_o will be LOW no matter fifo fulliness or emptyness.

6.0 Timestamp unit design with Avalon-MM



read_i	Avalon-MM input read signal 1bit
write_i	Avalon-MM input write signal 1bit
address_i	Avalon-MM input address signal 4bit
writedata_i	Avalon-MM input signal for data to be written into slave 32bit
readdata_o	Avalon-MM output signal for data to be read from slaves 32bit
response_o	Avalon-MM output signal for handling bus errors 2bit

	write_i = 1	description
address_i: 0000	ALLOWED	sys_time REGISTER ← ***writedata_i***
address_i: 0001	ALLOWED	status REGISTER ← ***writedata_i***
address_i: 0010	ALLOWED	control REGISTER ← ***writedata_i***

* read_i : not important
* rst_i ≠ 1 & control register(2) ≠ 1 (no hardware no software reset)

	write_i ≠ 1 & ***read_i*** = 1	description
address_i: 0000	ALLOWED	***readdata_o*** ← unix time value
address_i: 0001	ALLOWED	***readdata_o*** ← status register
address_i: 0010	ALLOWED	***readdata_o*** ← control register

* rst_i ≠ 1 & control register(2) ≠ 1 (no hardware no software reset)

	read_i = 1	description
address_i: 0011	ALLOWED	***readdata_o*** ← fall_ts_h
address_i: 0100	ALLOWED	***readdata_o*** ← fall_ts_l
address_i: 0101	ALLOWED	***readdata_o*** ← rise_ts_h
address_i: 0110	ALLOWED	***readdata_o*** ← rise_ts_l
address_i: 1000	ALLOWED	***readdata_o*** ← fifo_fall_ts_h
address_i: 1001	ALLOWED	***readdata_o*** fifo_fall_ts_l
address_i: 1010	ALLOWED	***readdata_o*** ← fifo_rise_ts_h
address_i: 1011	ALLOWED	***readdata_o*** ← fifo_rise_ts_l

* write_i : not important
* rst_i ≠ 1 & control register(2) ≠ 1 (no hardware no software reset)

	read_i ≠ 1 & ***write_i*** = 1	description
address_i: 0011	PERMISSION DENIED	***response_o*** ← "10"
address_i: 0100	PERMISSION DENIED	***response_o*** ← "10"
address_i: 0101	PERMISSION DENIED	***response_o*** ← "10"
address_i: 0110	PERMISSION DENIED	***response_o*** ← "10"
address_i: 1000	PERMISSION DENIED	***response_o*** ← "10"
address_i: 1001	PERMISSION DENIED	***response_o*** ← "10"
address_i: 1010	PERMISSION DENIED	***response_o*** ← "10"
address_i: 1011	PERMISSION DENIED	***response_o*** ← "10"

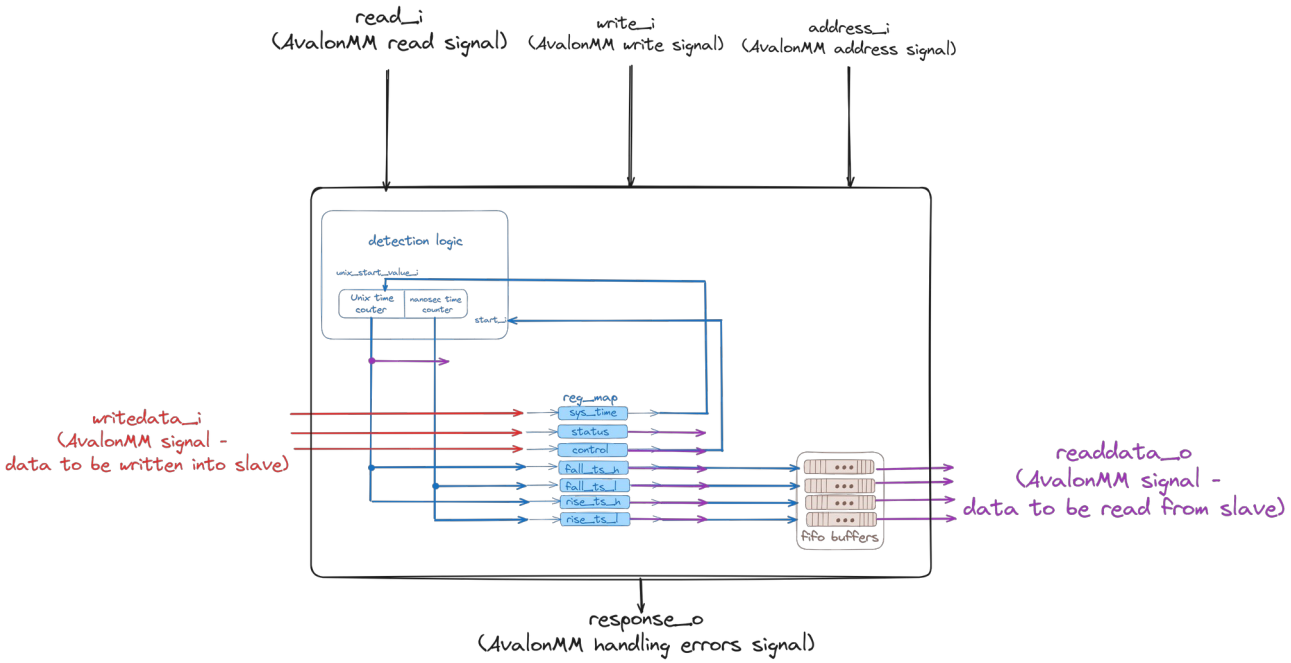
* rst_i ≠ 1 & control register(2) ≠ 1 (no hardware no software reset)

	read_i = 1 V ***write_i*** = 1	description
address_i: ?	PERMISSION DENIED	***response_o*** ← "11"

* rst_i ≠ 1 & control register(2) ≠ 1 (no hardware no software reset)

	rst_i = 1 V ***control register(2)*** = 1	description
address_i: ?	RESET	***response_o*** ← "00"

*__write_i__ : not important
*__read_i__ : not important


read_i
(AvalonMM read signal)
write_i
(AvalonMM write signal)
address_i
(AvalonMM address signal)
detection logic
unix_start_value_i
Unix time counter
nanosec time counter
start_i
reg_map
sys_time
status
control
fall_ts_h
fall_ts_l
rise_ts_h
rise_ts_l
fifo buffers
writedata_i
(AvalonMM signal -
data to be written into slave)
readdata_o
(AvalonMM signal -
data to be read from slave)
response_o
(AvalonMM handling errors signal)