Digital Filters Bank

Design Activities

In the digital world, signals are constantly being transmitted and processed. These signals often come with unwanted baggage - noise, interference and distortions that can affect their quality. Digital filters are the tools we use to clean up these signals, removing the unwanted parts and extracting the information we need. From simple low-pass filters that remove high-frequency noise to complex adaptive filters that adjust to changing signals, digital filters play a crucial role in many modern technologies.

The proposed application is focused on a simple noise reduction filter that can be configured using a parallel synchronous interface. The application is parametrizable, N being the number of inputs that can be filtered in parallel. The top-level module will contain **N** identical filters, **N** identical synchronizer blocks, one for each asynchronous input, and will have different number of registers (the address bus size will be affected) based on **N**.

The proposed architecture for the digital filters bank is described in Figure1.

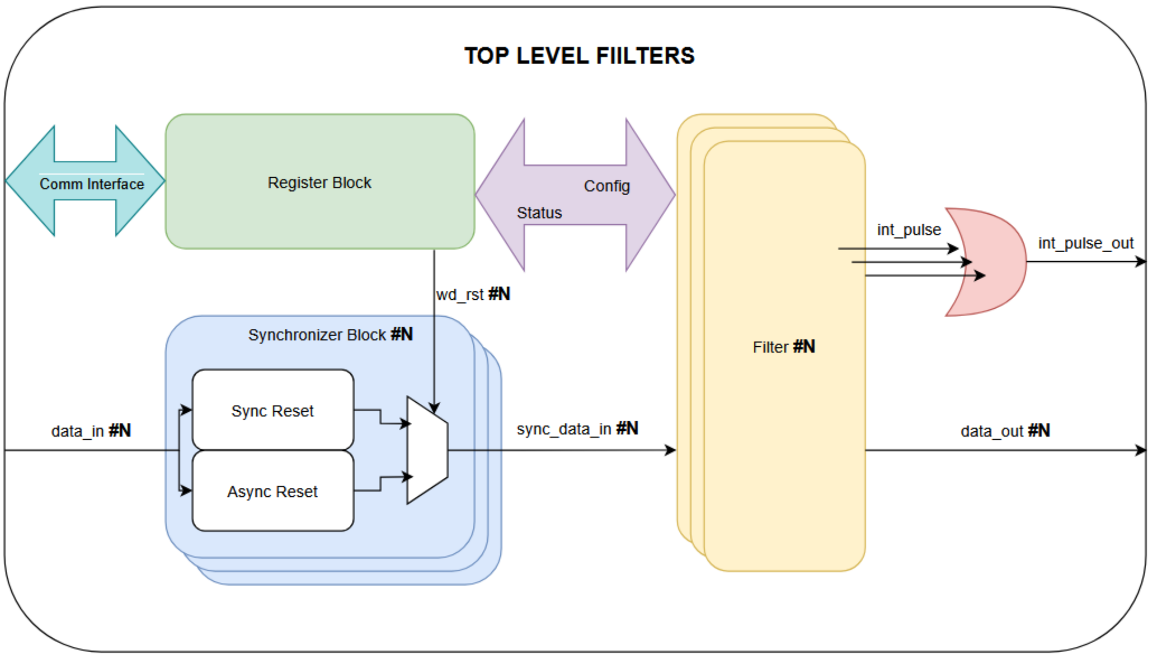


Figure1. Top Level Architecture

The top module should contain:

- A synchronous parallel interface used to configure the filters,
- A register block which contains all the configuration and status bits (the parallel interface with be part of the register block, no individual module needed),
- Synchronizer block for the asynchronous inputs (instantiated N times for each input),
- The same filter (one design) instantiated N time for each unique input

The interface of the top-level module is described in the table below. (**Follow the naming conventions for easy debugging!!!)**

Port name	Size	Direction	Clock domain	Description
clk_i	1	input	n.a	Clock signal with target frequency @TODOMhz
rstn_i	1	input	asynchronous	Asynchronous reset, active low
acc_en_i	1	input	clk_i	Access enable for synchronous transactions
wr_en_i	1	input	clk_i	Write enable. When set, a write is performed, when reset, a read is performed
addr_i	**addr_size***	input	clk_i	Address accessed during a transaction
wdata_i	8	input	clk_i	Data to be written during write access
rdata_o	8	output	clk_i	Data read during read access
data_in	N	input	asynchronous	Inputs that need to be filtered
data_out	N	output	clk_i	Filtered signals
int_pulse_out	1	output	clk_i	Interrupt pulse active for one clock pulse when any filtering active as interrupt source is done

*addr_size should be calculated based on the number of filters N.

Communication Interface

A protocol is a set of rules that define how data is transmitted and received over a network or communication channel. Protocols define the format, timing, sequencing, and error control for data communication. They also specify the types of messages that can be exchanged between devices, and how those messages are interpreted. Depending on the application for each they were defined, protocols are very different and can be grouped in different categories.

In this application, we will use a protocol with the following attributes:

- Synchronous
- Parallel

Synchronous communication requires that the clocks in the transmitting and receiving devices are synchronized – running at the same rate – so the receiver can sample the signal at the same time intervals used by the transmitter. Therefore, synchronous protocols always have an extra wire for the external clock signal.
In parallel communication, multiple bits of data are sent simultaneously, as oppose to serial communications which involves sending data one bit at a time.
The timing for write and read commands is shown in Figure2 and Figure3.

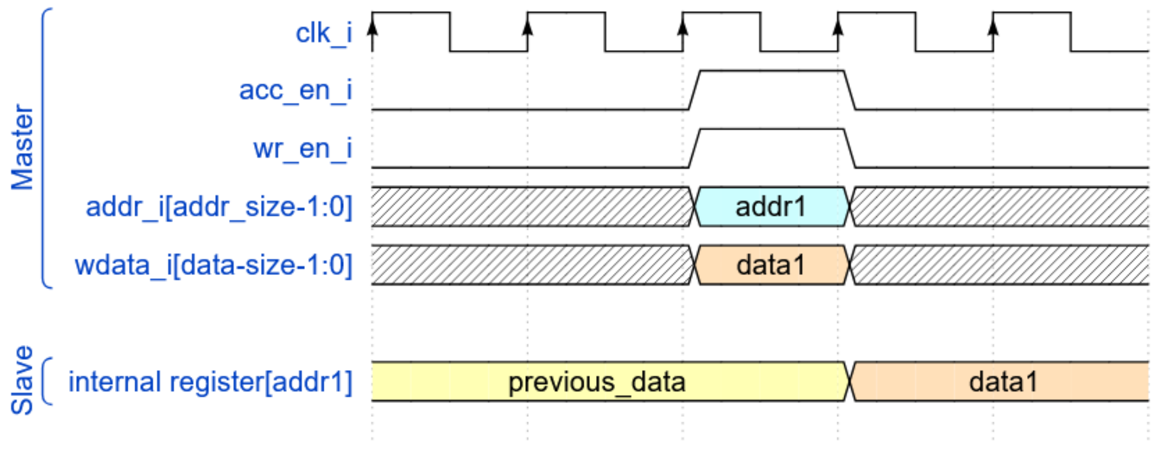


Figure2. Write access

Each access takes one clock cycle. Each register will have a particular access type. Register that can be read will have the attribute R, those that can be written will be the attribute W. Apart from the usual R/W (read/write) accees type, there are also RC registers. For those registers, performing a read will also perfom a clear (write with 0) on its content. Performing a write on a RC register will have no effect (will not modify the content of the register). A read on a RC register in illustread in Figure4.

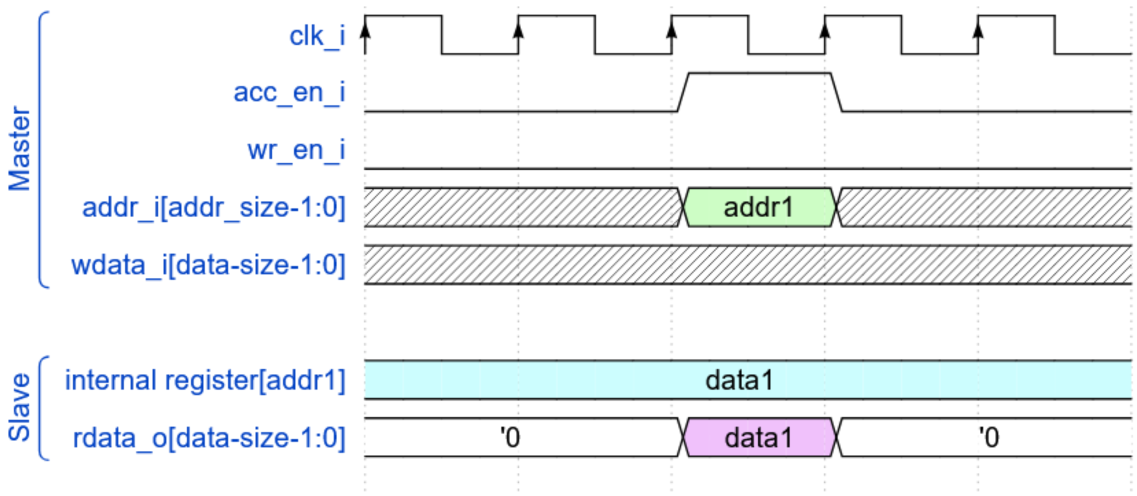


Figure3. Read access

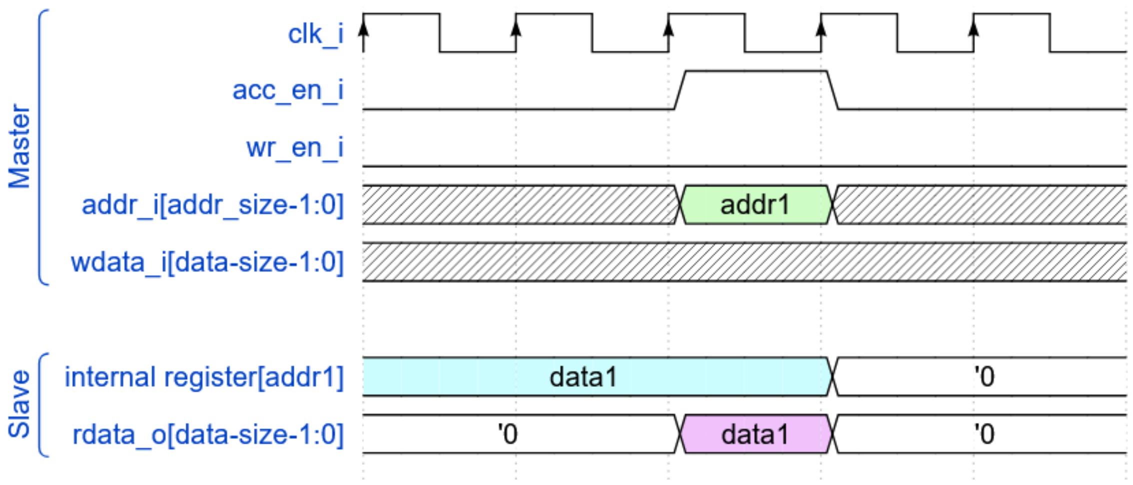
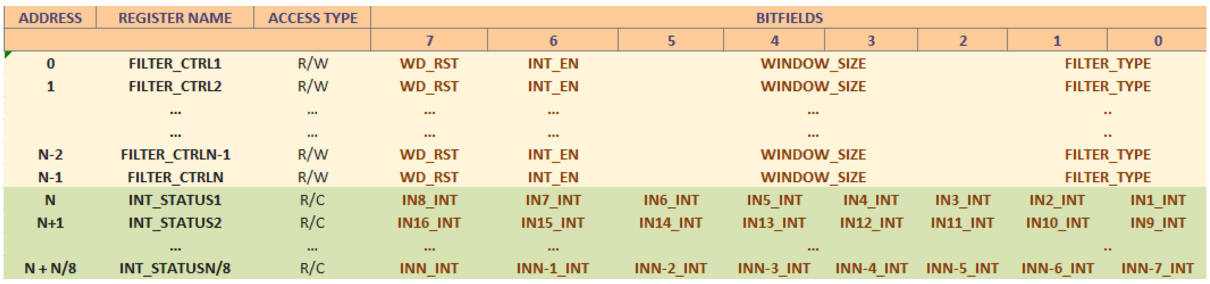


Figure4. Read clear access

Configurability – register block

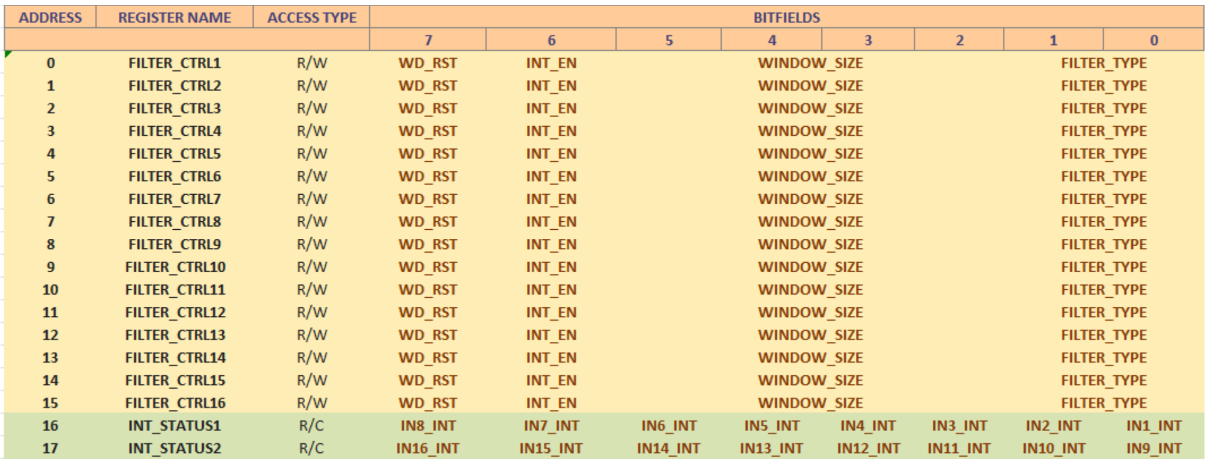
The register block will contain the configuration bits of the filters and will facilitate the access to the status of interrupt line. The module should implement the register map described in Figure5.



ADDRESS	REGISTER NAME	ACCESS TYPE	BITFIELDS							
			7	6	5	4	3	2	1	0
0	FILTER_CTRL1	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
1	FILTER_CTRL2	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
	...	...	...	...	...				..	
	...	...	...	...	...				..	
N-2	FILTER_CTRLN-1	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
N-1	FILTER_CTRLN	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
N	INT_STATUS1	R/C	IN8_INT	IN7_INT	IN6_INT	IN5_INT	IN4_INT	IN3_INT	IN2_INT	IN1_INT
N+1	INT_STATUS2	R/C	IN16_INT	IN15_INT	IN14_INT	IN13_INT	IN12_INT	IN11_INT	IN10_INT	IN9_INT
	...	...	...	...		...			..	
N + N/8	INT_STATUSN/8	R/C	INN_INT	INN-1_INT	INN-2_INT	INN-3_INT	INN-4_INT	INN-5_INT	INN-6_INT	INN-7_INT

Figure5. Register map

For a better understanding of the register map, an example for N=16 is provided in Figure6.



ADDRESS	REGISTER NAME	ACCESS TYPE	BITFIELDS							
			7	6	5	4	3	2	1	0
0	FILTER_CTRL1	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
1	FILTER_CTRL2	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
2	FILTER_CTRL3	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
3	FILTER_CTRL4	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
4	FILTER_CTRL5	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
5	FILTER_CTRL6	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
6	FILTER_CTRL7	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
7	FILTER_CTRL8	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
8	FILTER_CTRL9	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
9	FILTER_CTRL10	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
10	FILTER_CTRL11	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
11	FILTER_CTRL12	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
12	FILTER_CTRL13	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
13	FILTER_CTRL14	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
14	FILTER_CTRL15	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
15	FILTER_CTRL16	R/W	WD_RST	INT_EN	WINDOW_SIZE				FILTER_TYPE	
16	INT_STATUS1	R/C	IN8_INT	IN7_INT	IN6_INT	IN5_INT	IN4_INT	IN3_INT	IN2_INT	IN1_INT
17	INT_STATUS2	R/C	IN16_INT	IN15_INT	IN14_INT	IN13_INT	IN12_INT	IN11_INT	IN10_INT	IN9_INT

Figure6. Register map for N=16

Each register is completely described in the following section. The default(reset) value is underlined in each register.

FILTER_CTRL			
Bit #	Bit field name	Access	Description
[1:0]	Filter_type	R/W	These bits are used to enable the filter and set its operation mode: 2'b00 – filter disabled -corresponding output will be 0 2'b01 – rise filter 2'b10 – fall filter 2'b11 – rise and fall filter
[5:2]	Window_size	R/W	These bits are used to set the size of the filtering window:

			4’b0000 – 4 4’b0001 – 8 4’b0010 – 16 4’b0011 – 32 4’b0100 – 48 4’b0101 – 64 4’b0110 – 128 4’b0111 – 256 4’b1000 – 512 4’b1001 – 640 4’b1010 – 768 4’b1011 – 896 4’b1100 – 1024 4’b1101 – 1280 4’b1110 – 1536 4’b1111 – 2048 Note: The size is specified in clock cycles.
6	Int_en	R/W	This bit is used for interrupt enable. When the filter finishes counting it will generate a pulse that will be mapped on the interrupt output if this bit is enabled: 1’b0 – this filter will not generate an interrupt 1’b1 – this filter will generate an interrupt
7	Wd_rst	R/W	This bit is used to select the window reset type: 1’b0 – asynchronous reset 1’b1 – synchronous reset

INT_STATUS			
Bit #	Bit field name	Access	Description
[7:0]	InX_INT	R/C	Each bit represents the interrupt status (interrupt done) of each individual input X.

Filtering types

Rise Type

When rise type filtering is selected in FILTER_CTRL[filter_type] only the transition from 0 -> 1 will be filtered. The output will transition to high level only if the input had a stable high level for “window size” clock cycles. Filtered output will be reset as soon as the synchronized input is 0. Examples of rise filters with window size 4 and synchronous window reset are shown in Figure7 and Figure8.

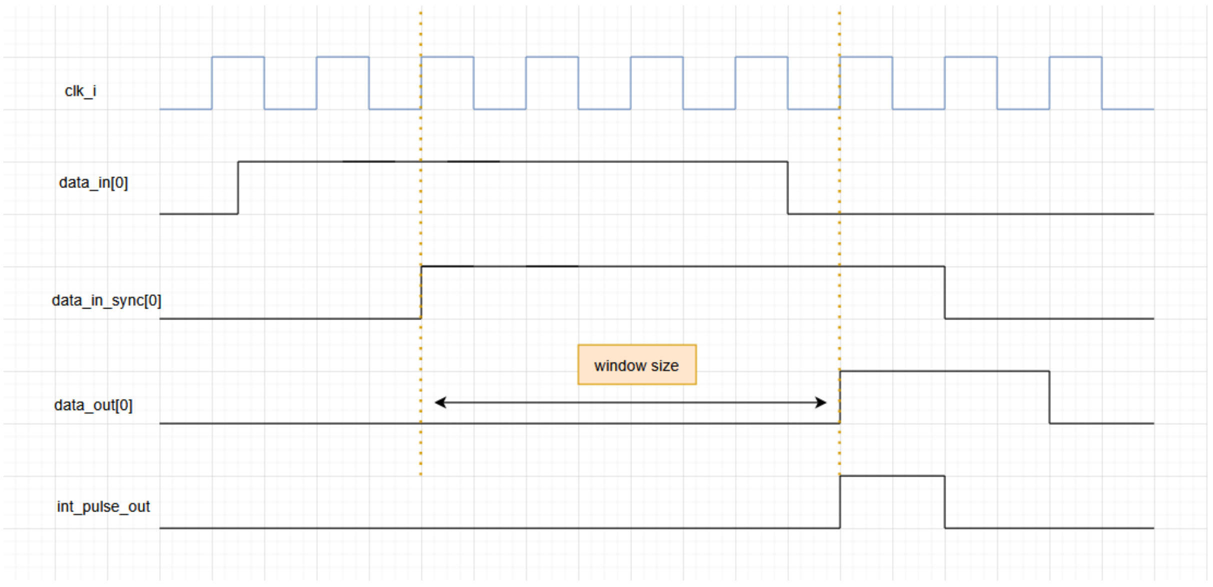


Figure7. Rise Filter example 1

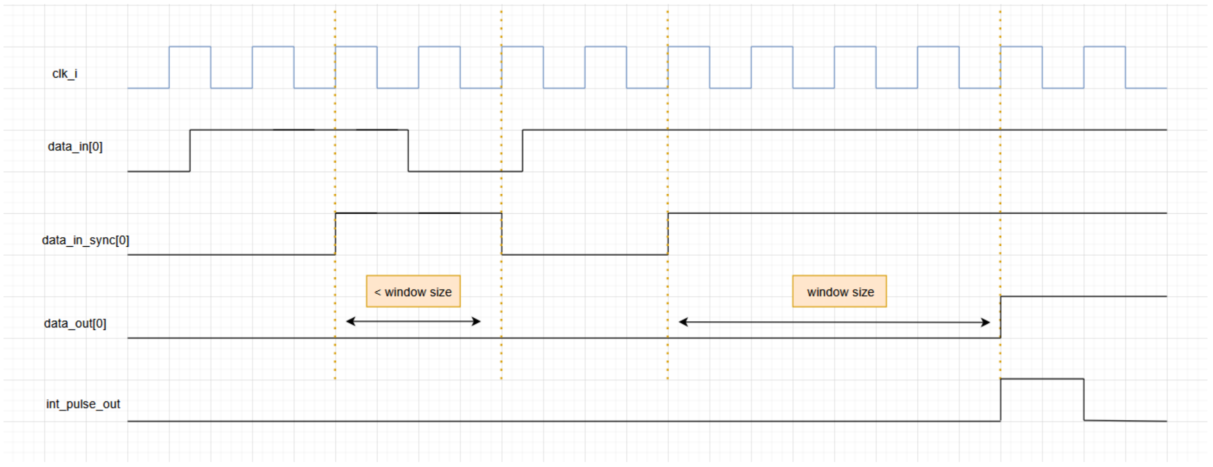


Figure8. Rise Filter example 2

Fall Type

When fall type filtering is selected in FILTER_CTRL[filter_type] only the transition from 1 -> 0 will be filtered. The output will transition to low level only if the input had a stable low level for "window size" clock cycles. Filtered output will be set as soon as the synchronized input is 1. Examples of fall filters with window size 4 and synchronous window reset are shown in Figure9 and Figure10.

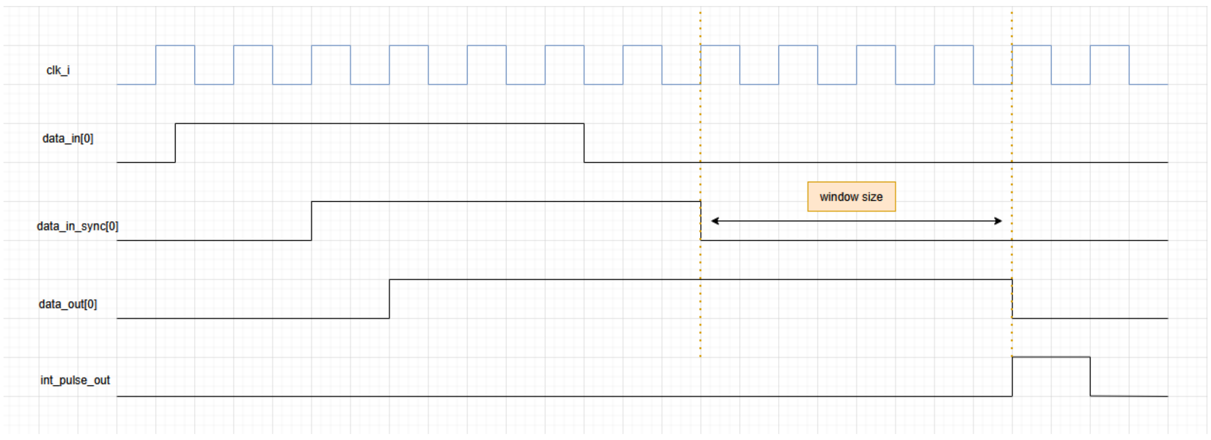


Figure9. Fall Filter example 1

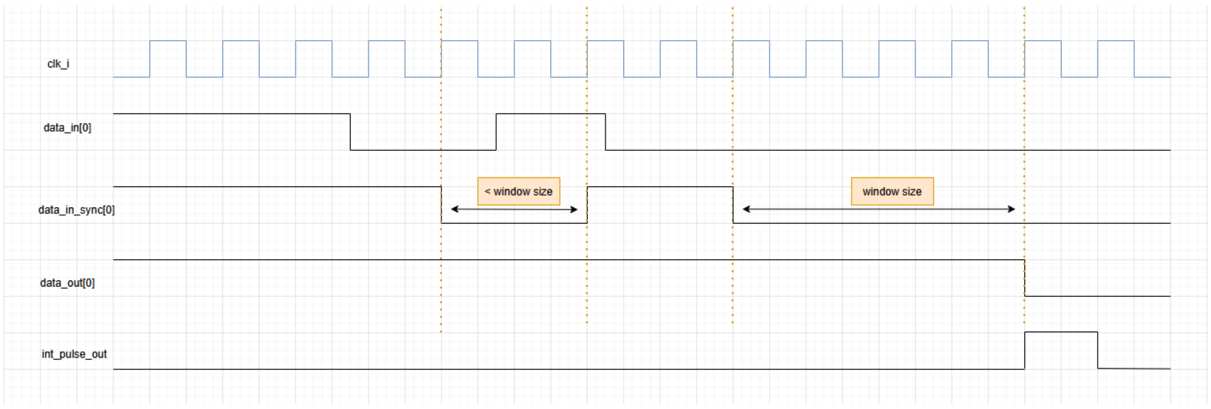


Figure10. Fall Filter example 2

Rise and Fall Type

When rise and fall type filtering is selected in FILTER_CTRL[filter_type] both transitions will be filtered. The output will transition to a new level only if the input had that stable level for "window size" clock cycles. An example of rise and fall filter with window size 4 and synchronous window reset is shown in Figure11.

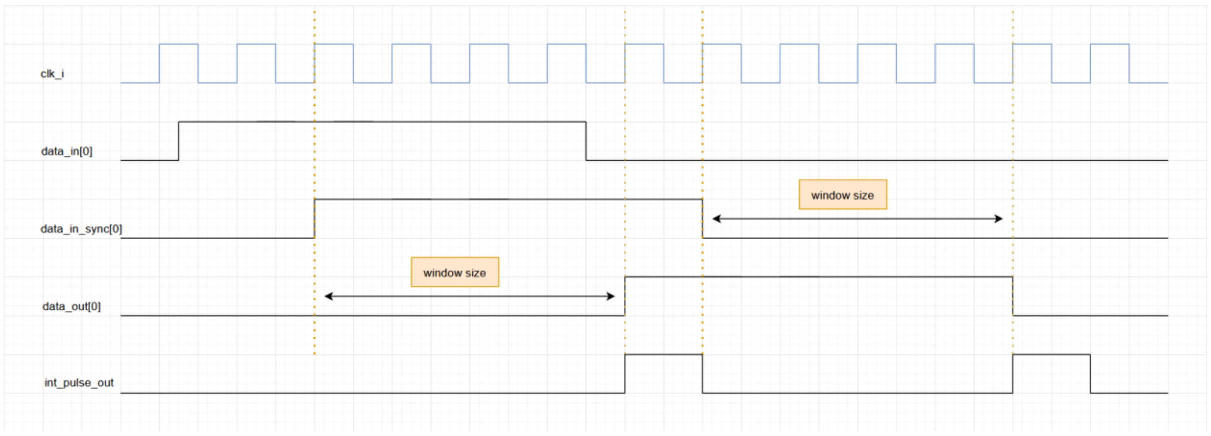


Figure11. Rise and Fall Filter example

Window reset types

Synchronous Type

Filters with synchronous reset synchronizer has the downfall of accumulating "errors" as shown in Figure12. Toggles between the rising edges of the clock signals cannot be detected with these filters.

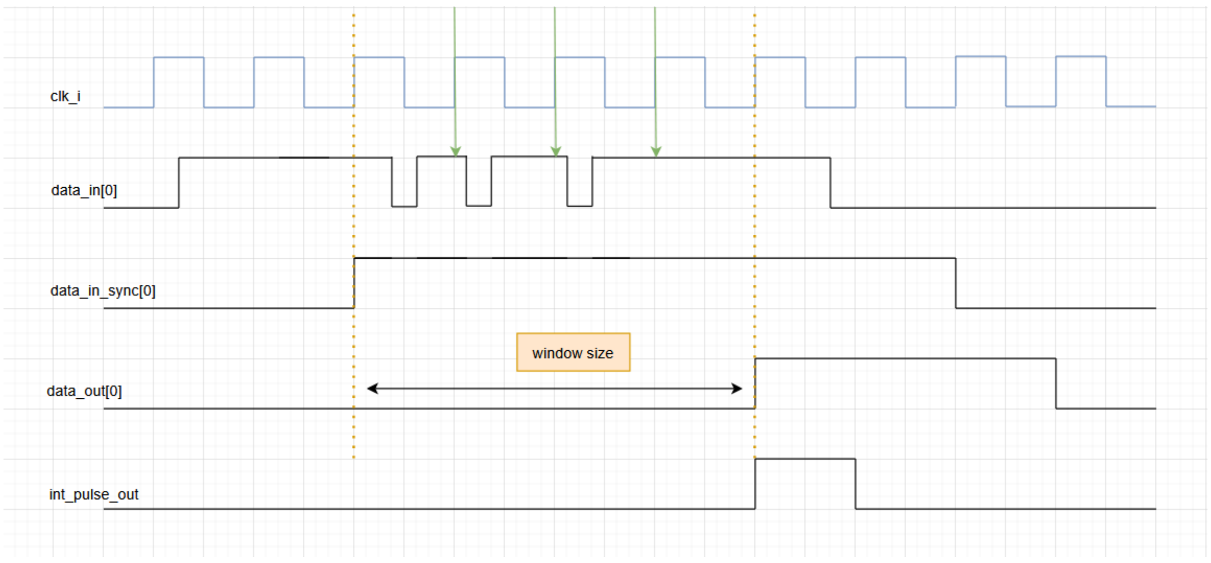


Figure12. Synchronous Window Reset Type Downfall

Asynchronous Type

Filters with asynchronous reset synchronizer can be used to detect any change of the input level, even those smaller than a clock period. These filters will change their output level only if the input was stable for the filtering window size. In case the input is an error signal, this avoids signalizing false errors when the input is toggling at higher duty cycle than the clock signal. An example for this filter type is shown in Figure13.

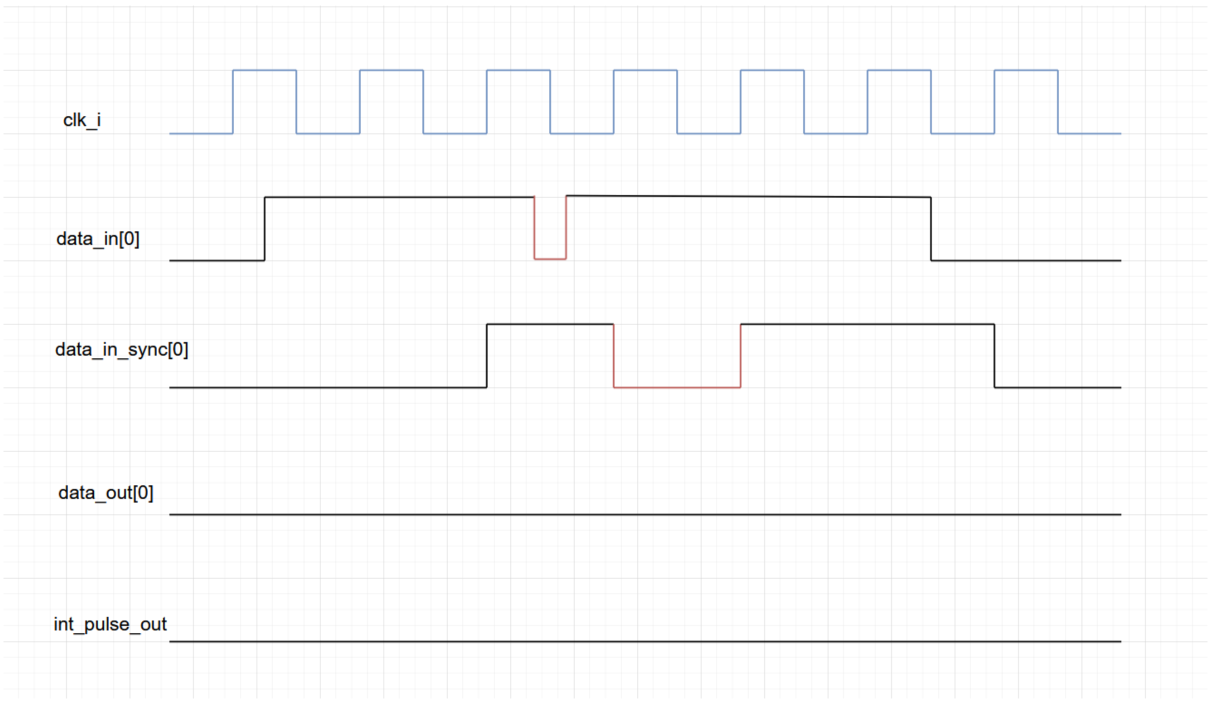


Figure13. Asynchronous Window Reset Type

Note: Only small pulses of low(0) level can be detected with this method.

Interrupt Line

An interrupt line, also known as an interrupt request (IRQ) line, is a signal line that allows a hardware device to interrupt the normal execution of a program or process on a processor. An interrupt is a signal to the processor that an event has occurred and requires immediate attention. This event can be a hardware event, such as a key press, or a software event, such as a timer expiration.

In this application, an interrupt pulse of one clock cycle will be generated when a filter finished counting and it is enabled as an interrupt source in FILTER_CTRL[int_en]. Each filter can generate interrupts, so the output int_pulse_out is an **OR** between the interrupts of all the filters. Examples are shown in Figures14 and Figure15.

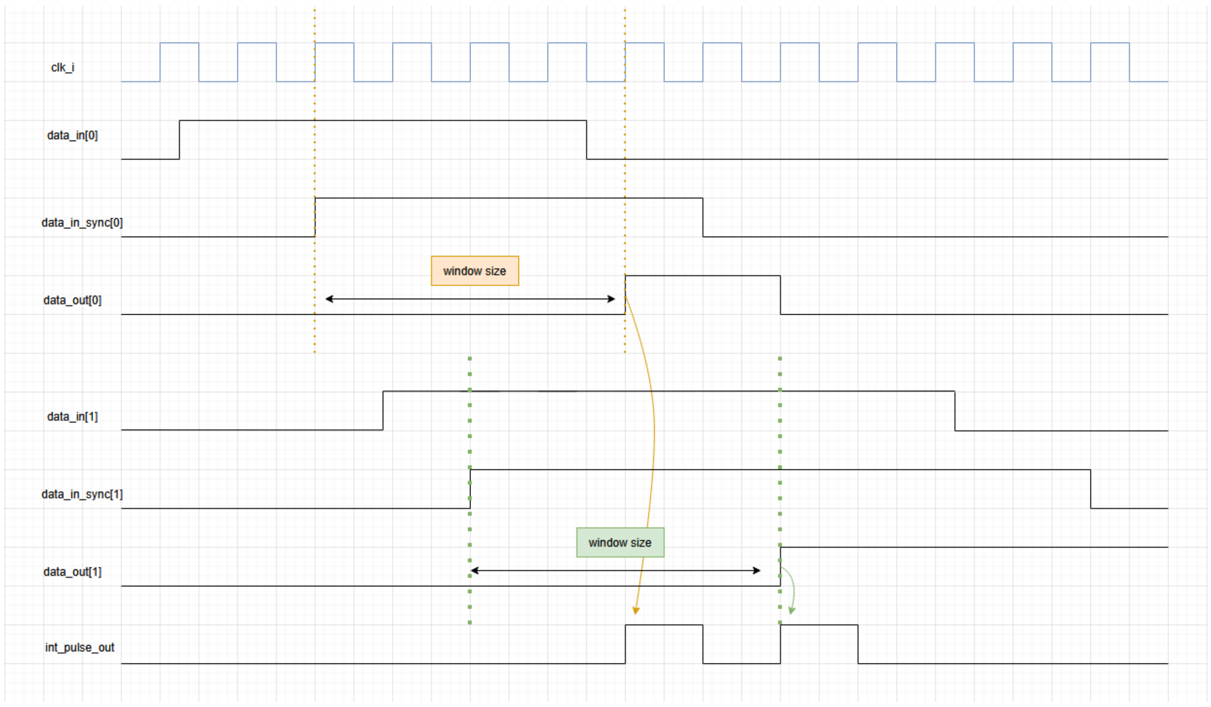


Figure14. Interrupt pulse example1

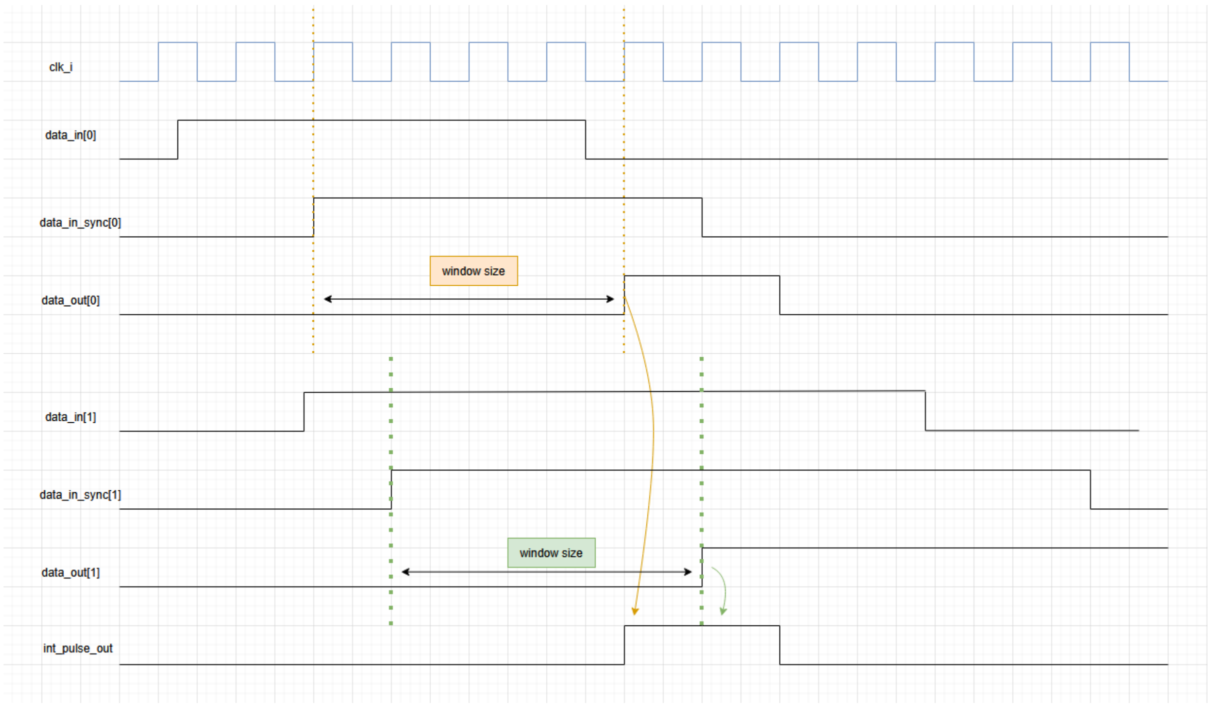


Figure15. Interrupt pulse example2

Task scheduler

Day 1:

- *Digital Design Introduction, HW vs SW, programming languages vs hardware description languages (mentors)*
- *Synchronization block(all)*
- *Testbench for existing blocks (students)*

Day 2:

- *Synchronous parallel protocol explanations & discussions (all)*
- *Interface + register block implementation (students)*
- *Testbench for existing blocks (students)*

Day 3:

- *Filter block implementation (all)*
- *Testbench for existing blocks(students)*

Day 4:

- *Top level instantiations & generation(all)*
- *Testbench for existing blocks using different N (students)*

Day 5:

- *Tests, Debugging(all)*
- *Digital Design Backend, RTL -> chip (mentors)*