GE VERNOVA

Use of Data Quality and Other Metadata for Robust Decision Making in SIPS and WAMPAC Schemes

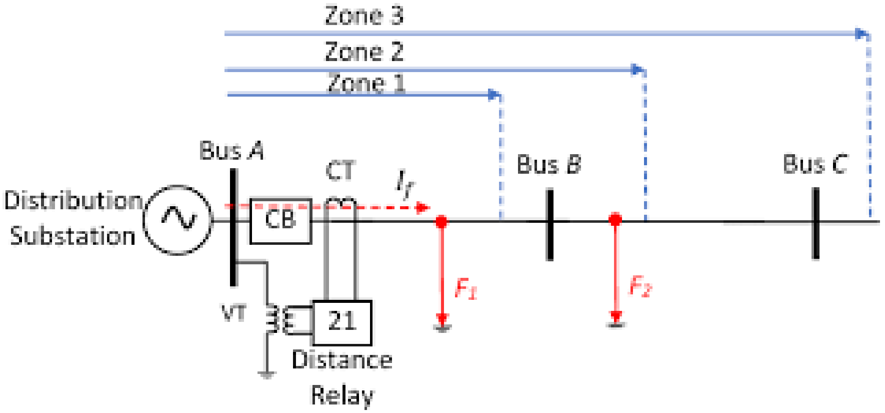
Oleg Bagleybter, Piotr Sawko
GE Vernova, Grid Automation

GE VERNOVA

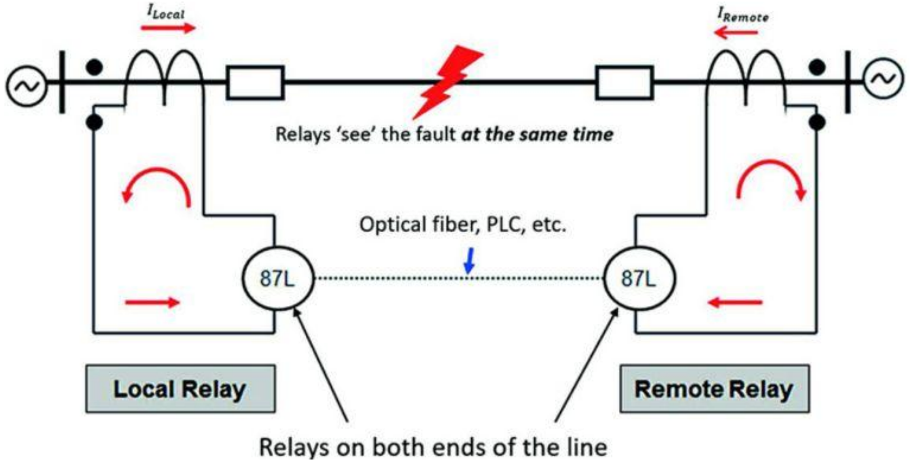
PROBLEM STATEMENT:
WHY WAMPAC AND SIPS REQUIRE SPECIAL TREATMENT FOR DATA QUALITY?

Conventional Protection

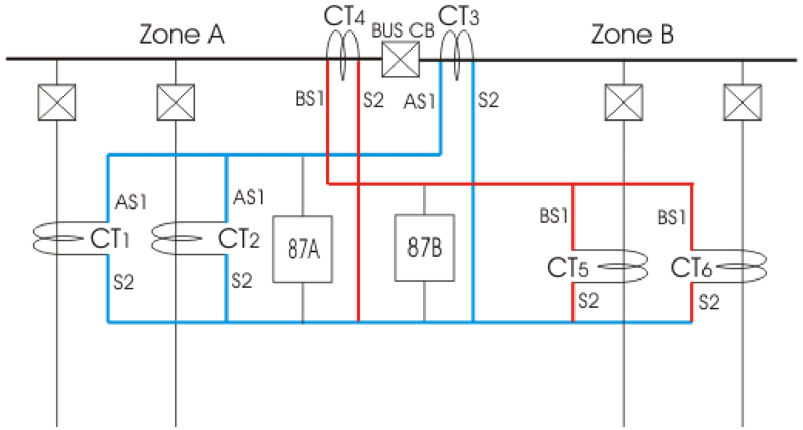
Distance



Line Differential



Busbar Differential



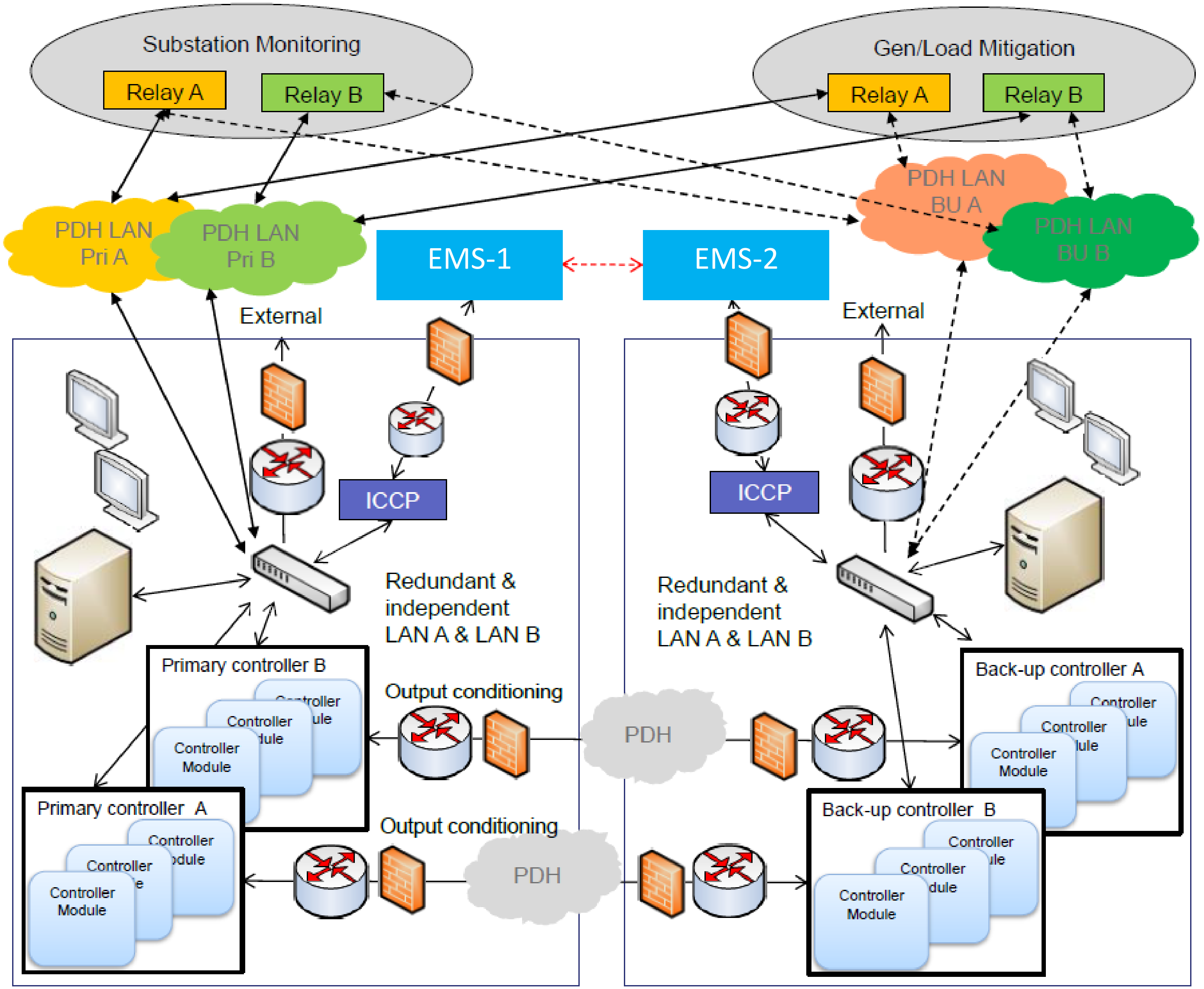
- Limited number of data sources (CT/VT)
- Located physically nearby
- In case of using Sampled Values – short communications links

CT/VT Supervision function => if anything goes wrong, block the whole protection element

Large-scale WAMPAC/SIPS project

50+ substations

200+ PMUs

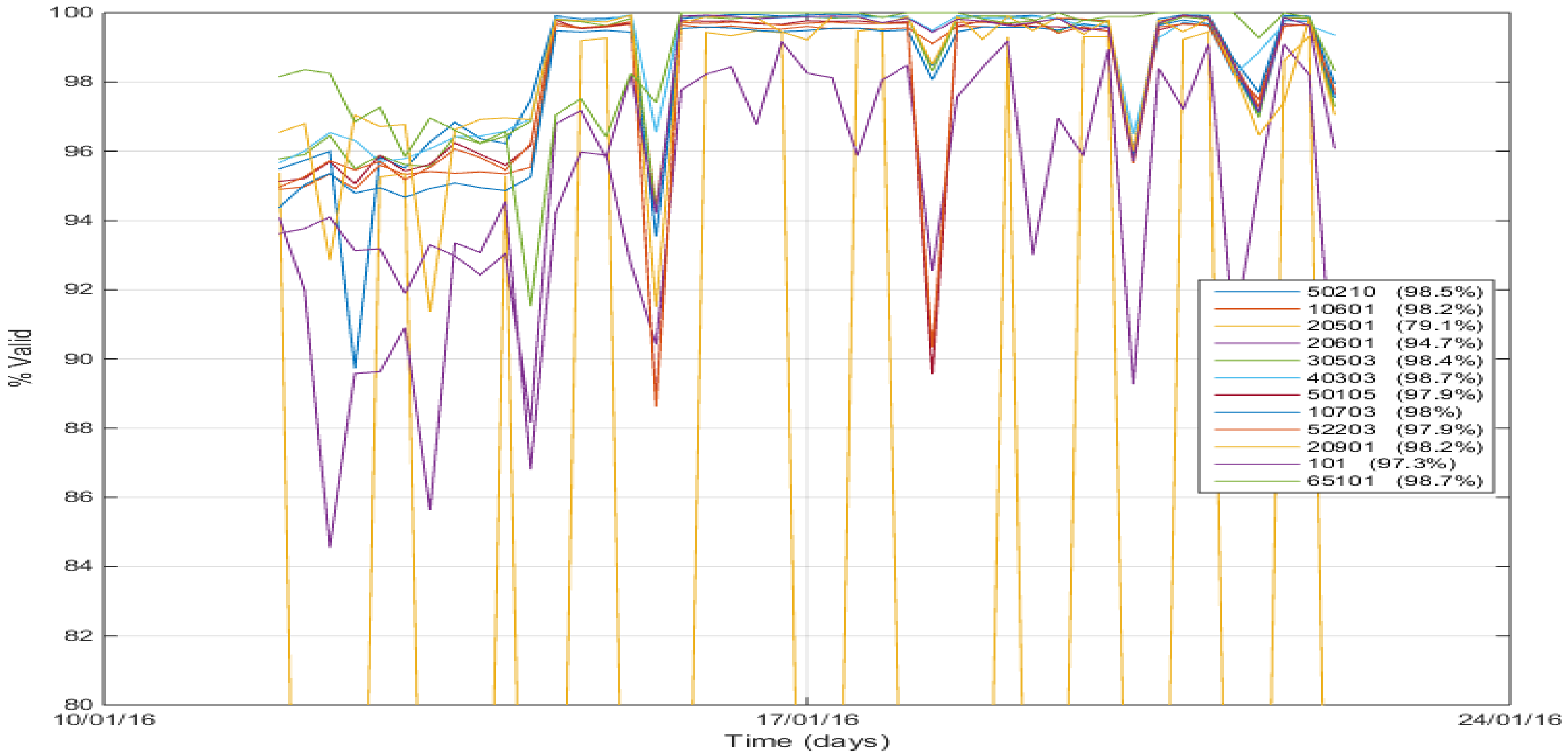


12x SIPS controllers

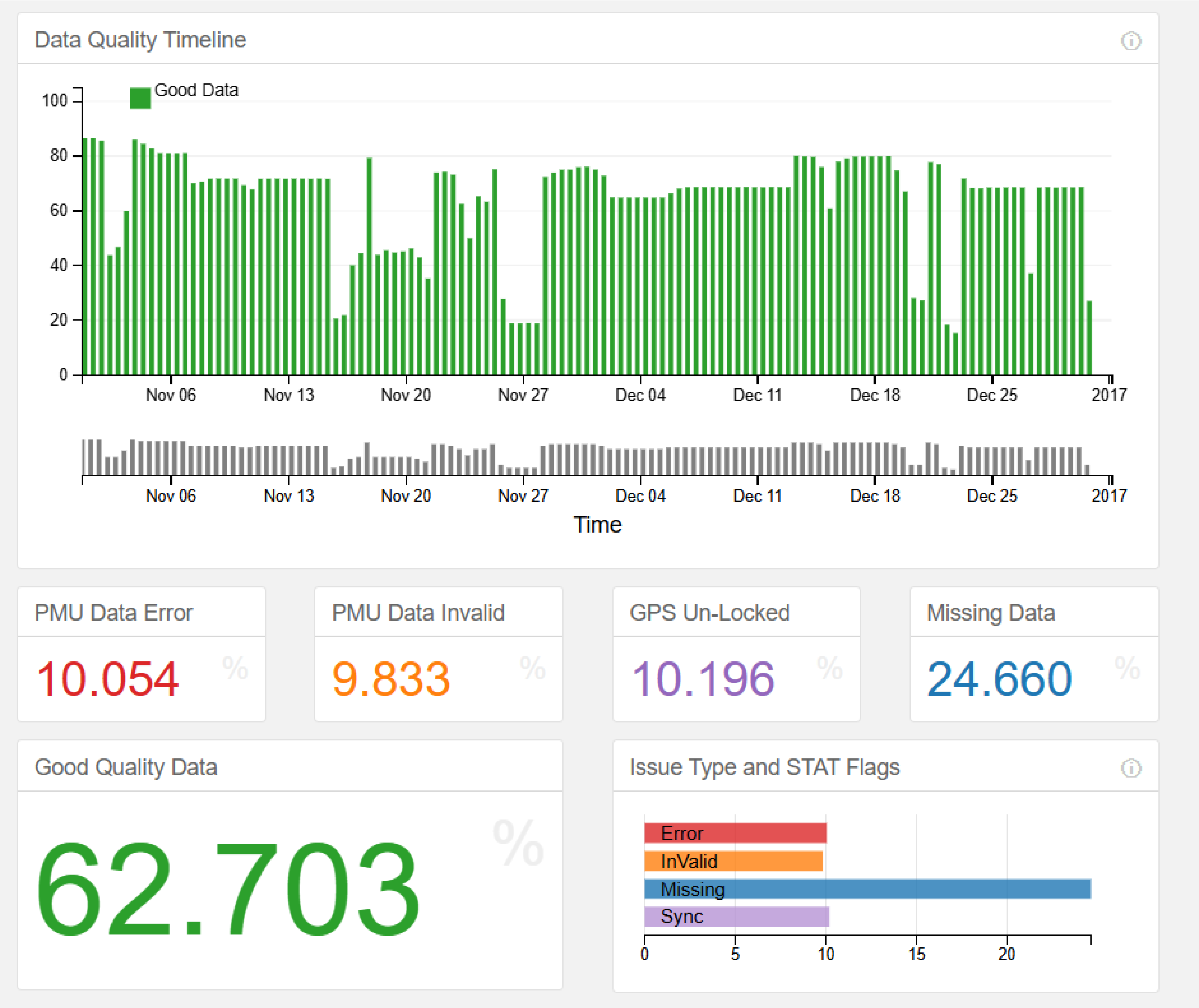
2x 2x 3x redundancy

If one or two PMU inputs are missing or have low quality – cannot afford to block everything

PMU Data Quality Examples



PMU Data Quality Examples



Data Quality Timeline

PMU Data Error 10.054 %

PMU Data Invalid 9.833 %

GPS Un-Locked 10.196 %

Missing Data 24.660 %

Good Quality Data 62.703 %

Issue Type and STAT Flags

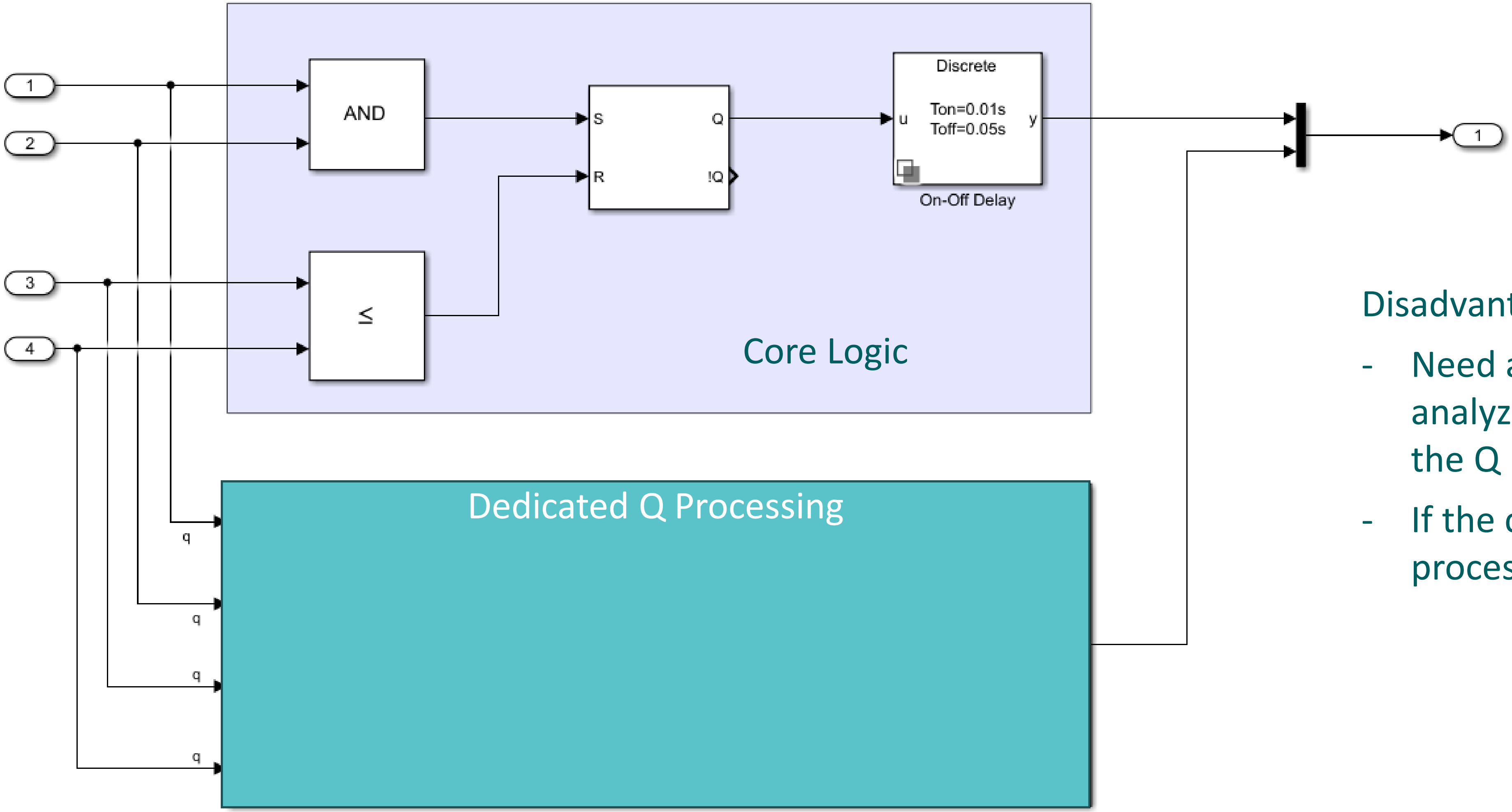
Conclusion:
WAMS/WAMPAC data are never perfect

GE VERNOVA

PROPOSED SOLUTION:

PROCESS AND PROPAGATE QUALITY ATTRIBUTES OF THE DATA THROUGH THE WAMPAC/SIPS ALGORITHMS AND LOGIC

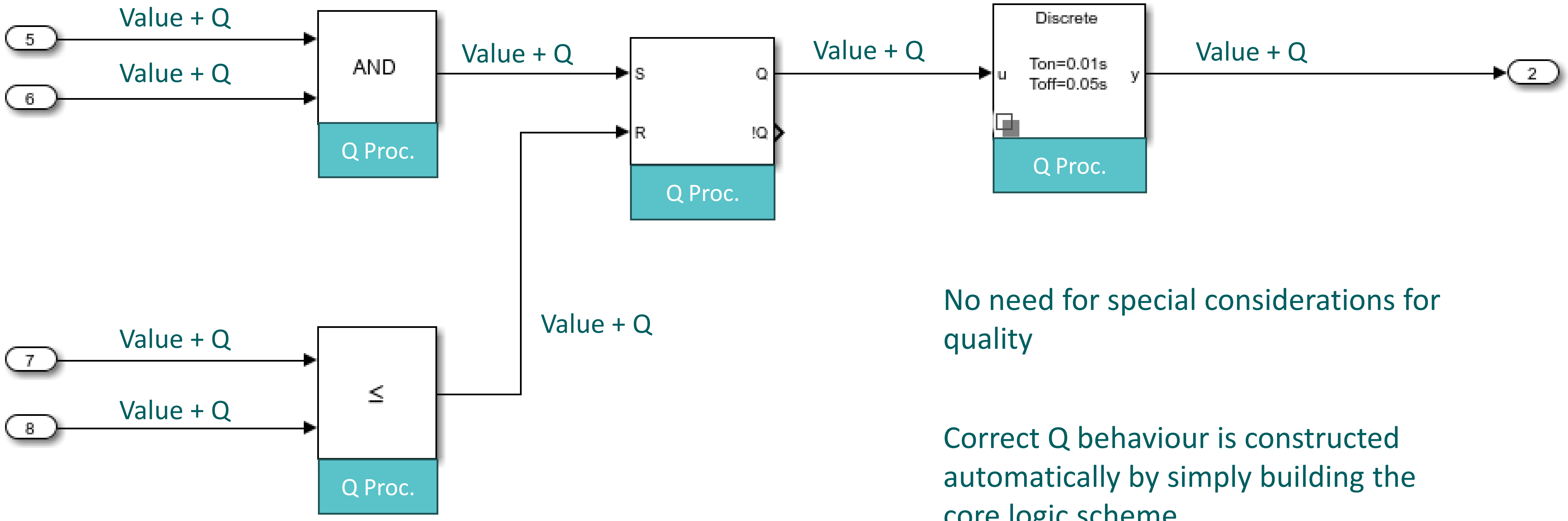
Option 1: Quality processing built “by hand”, abstracted from the main part of the logic



Disadvantage:

- Need an experienced designer to analyze the core logic and build the Q processing accordingly
- If the core logic changes, the Q processing needs to change too

Option 2: Quality built into each primitive logic element



No need for special considerations for quality

Correct Q behaviour is constructed automatically by simply building the core logic scheme

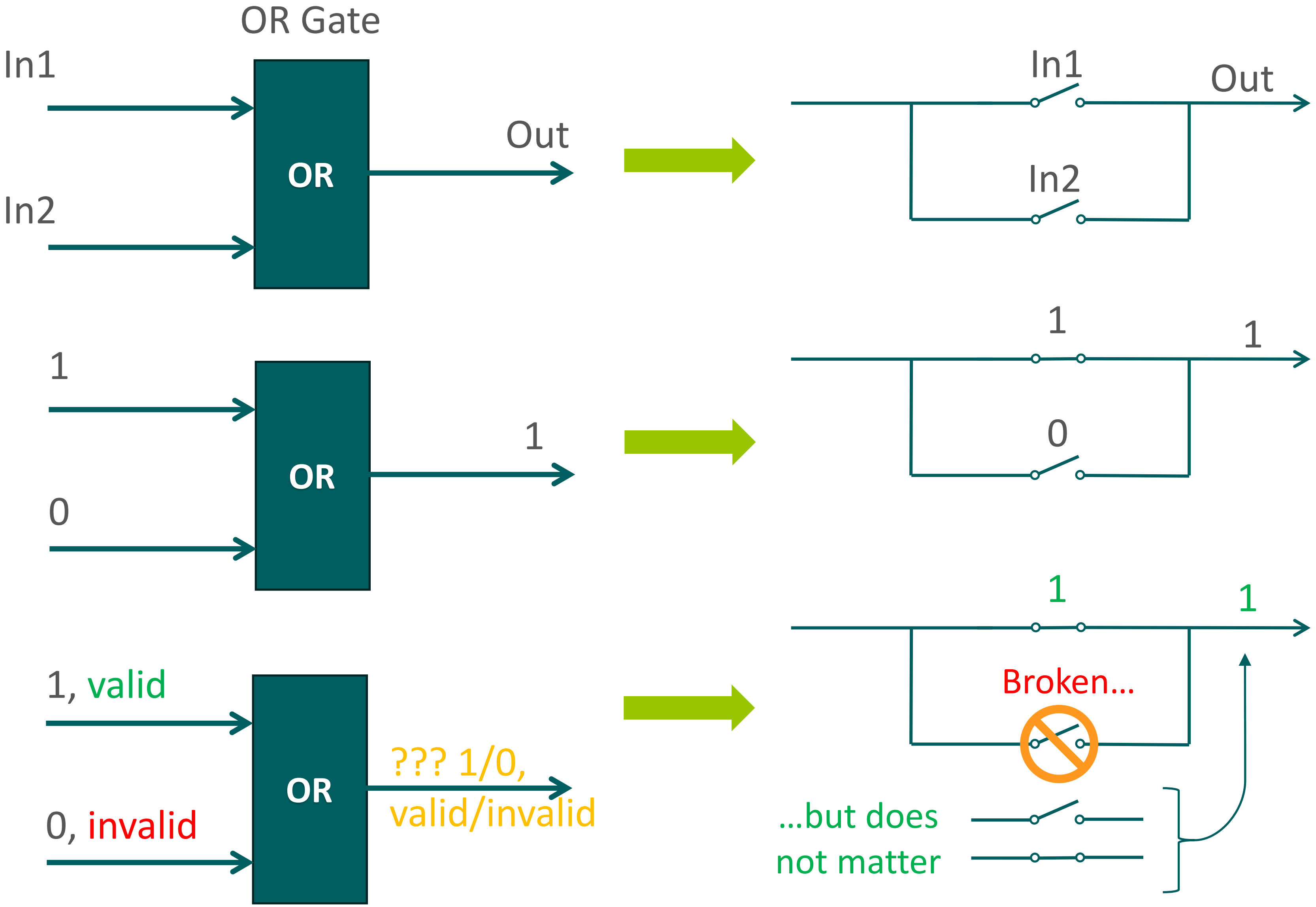
General principles of quality processing

1. If the data is **invalid**, the real value of the signal is presumed unknown. We consider that it can be either TRUE or FALSE for Boolean signals, or any value for analog signals.

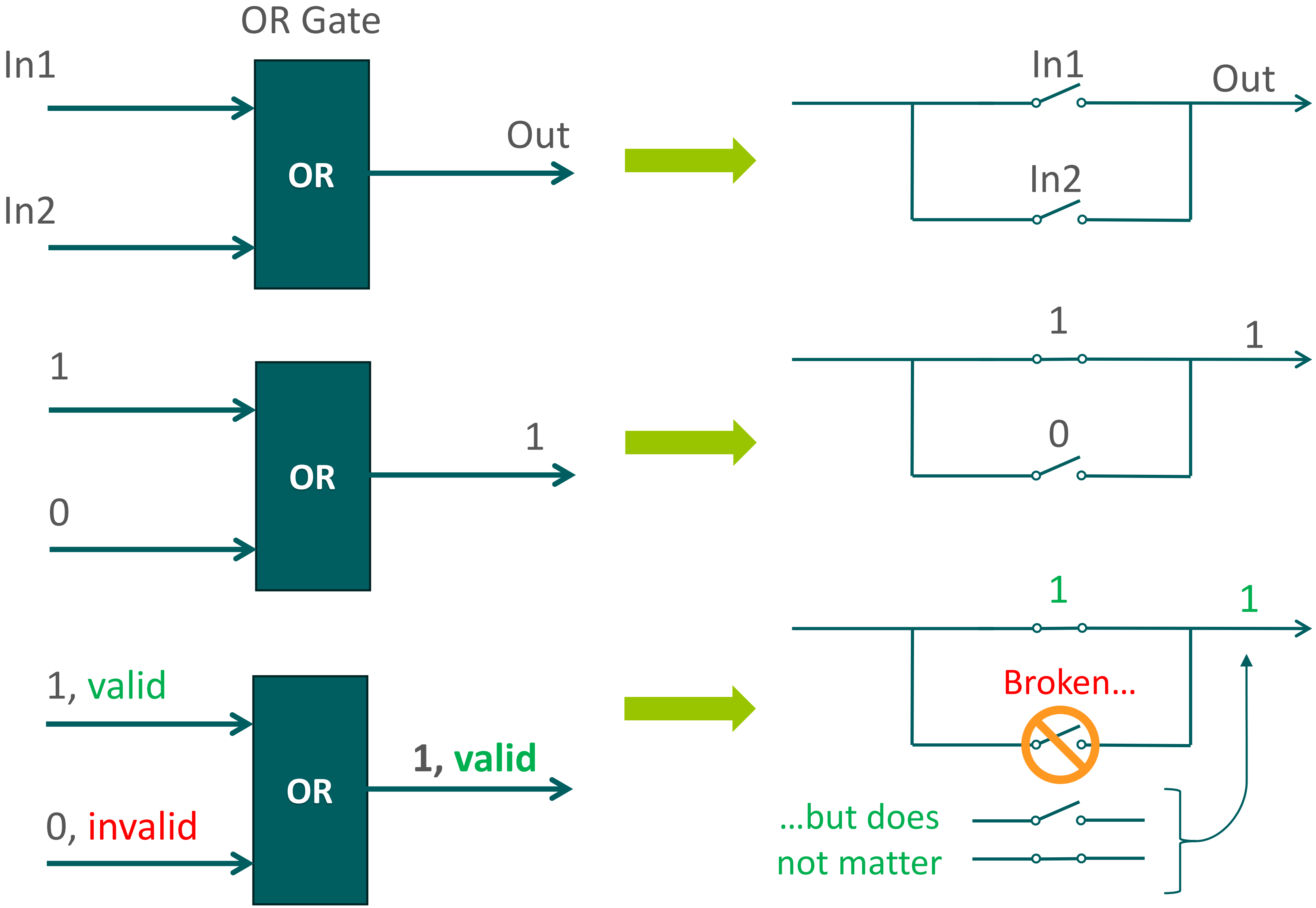
2. If we cannot decide – UNAMBIGUOUSLY – what the value of the function output is, then we declare it **invalid**. Otherwise, we declare the output **valid,** even if one or more of the inputs are **invalid.**

Let’s apply these principles to a simple OR gate.

Validity processing for an OR gate



Validity processing for an OR gate



OR Gate

In1

In2

OR

Out

In1

In2

Out

1

0

OR

1

1

0

1

1, valid

0, invalid

OR

1, valid

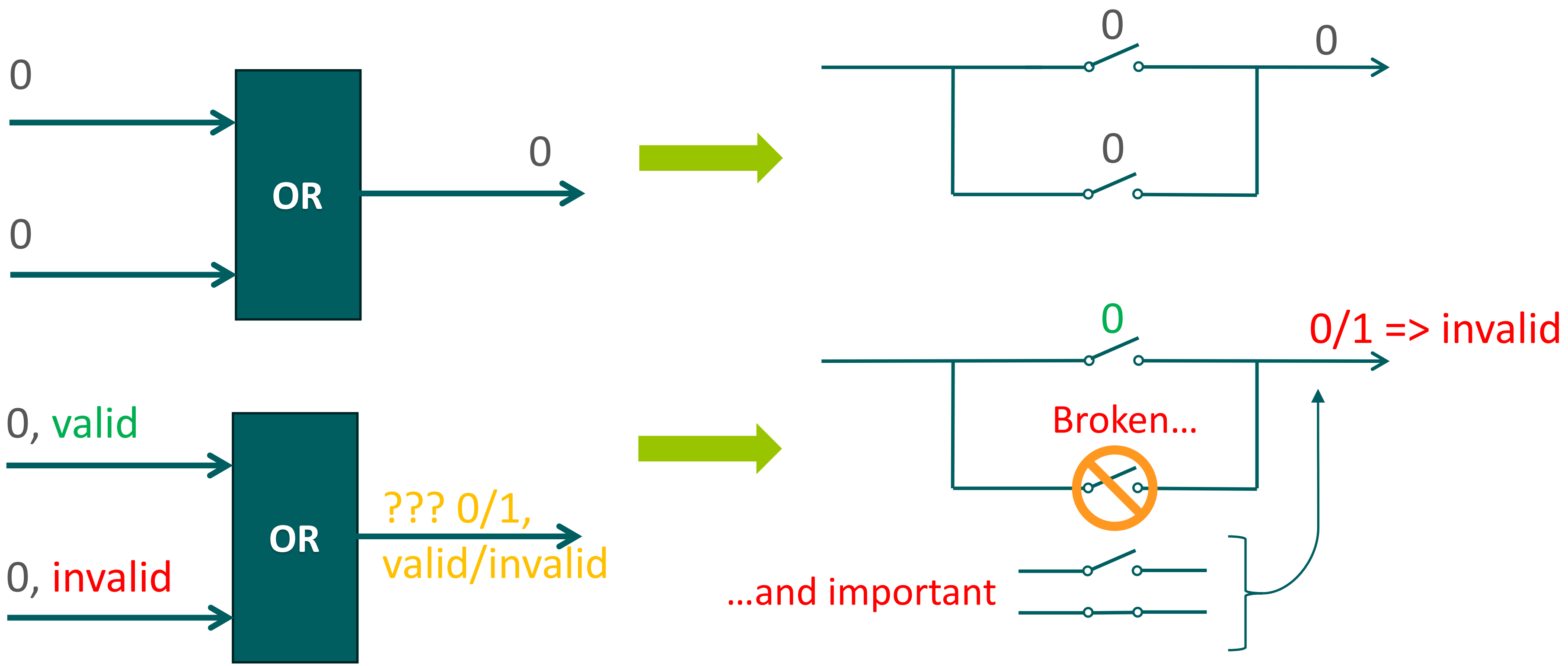
1

1

Broken…

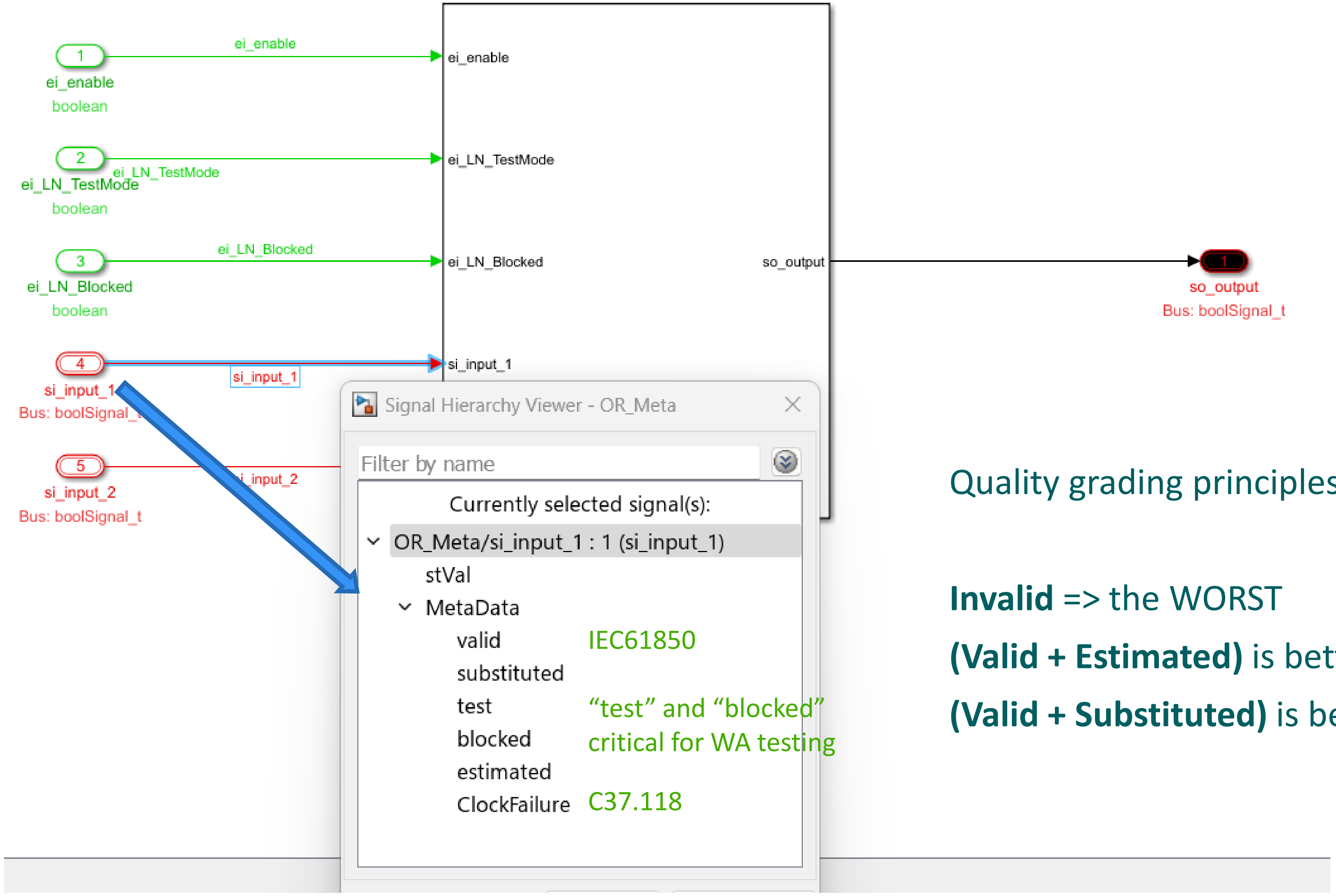
…but does not matter

Validity processing for an OR gate



The fundamental approach to processing degraded data is based on interpreting “invalid” signals as “unknown”. This is directly derived from the *Stephen Cole Kleene's "strong logic of indeterminacy“.*

Additional quality information – in addition to VALIDITY



Currently selected signal(s):

- OR_Meta/si_input_1 : 1 (si_input_1)
  - stVal
  - MetaData
    - valid
    - substituted
    - test
    - blocked
    - estimated
    - ClockFailure

IEC61850

"test" and "blocked" critical for WA testing

C37.118

Quality grading principles:

Invalid => the WORST

(Valid + Estimated) is better than **(Valid + Substituted)**

(Valid + Substituted) is better than **(Valid + ClockFailure)**

Detailed quality processing principles

OR gate:

- If one or more inputs are TRUE, pick the one with the best quality and pass it through to the output (this input is considered to “drive” the output directly).

- If all inputs are FALSE, the quality of the output is derived as the worst-case combination of the quality of all inputs (since all of the FALSE inputs are “driving” the output together).

AND gate: inverse of the OR gate approach, i.e. all TRUE inputs “drive” the output together and their quality needs to be “worst-case-combined”, while a FALSE input with the best quality would “drive” the output directly.

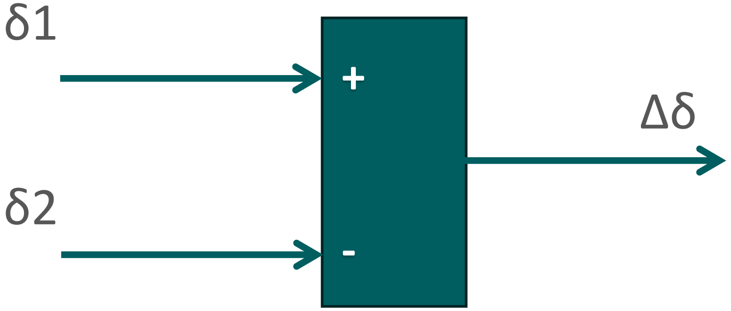
Analog operations on multiple inputs (ADD, SUBTRACT, PRODUCT, COMPARE etc.):

- All inputs are equally important, they “drive” the output together, hence their quality needs to be “worst-case-combined”.

- Exceptions are also possible that have increased tolerance to input data degradation, for example an AVERAGE function on 50 inputs that can tolerate the loss (invalidity) of up to 20% of all inputs – in this case the AVERAGE will still work on the remaining inputs, unless the loss tolerance limit is exceeded.

Additional considerations for phasor angles

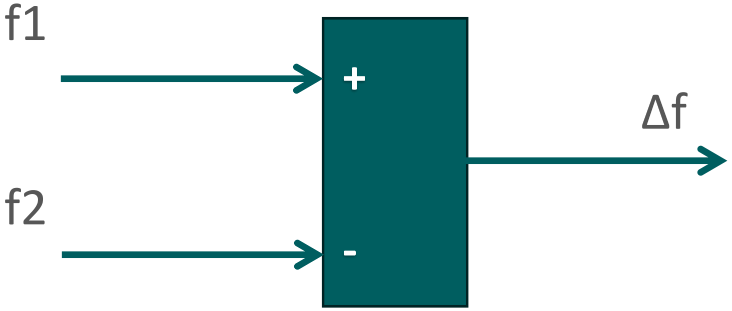
Angle Difference



1ms clock drift = 18° error

If any angle is not GPS-locked, the output is INVALID

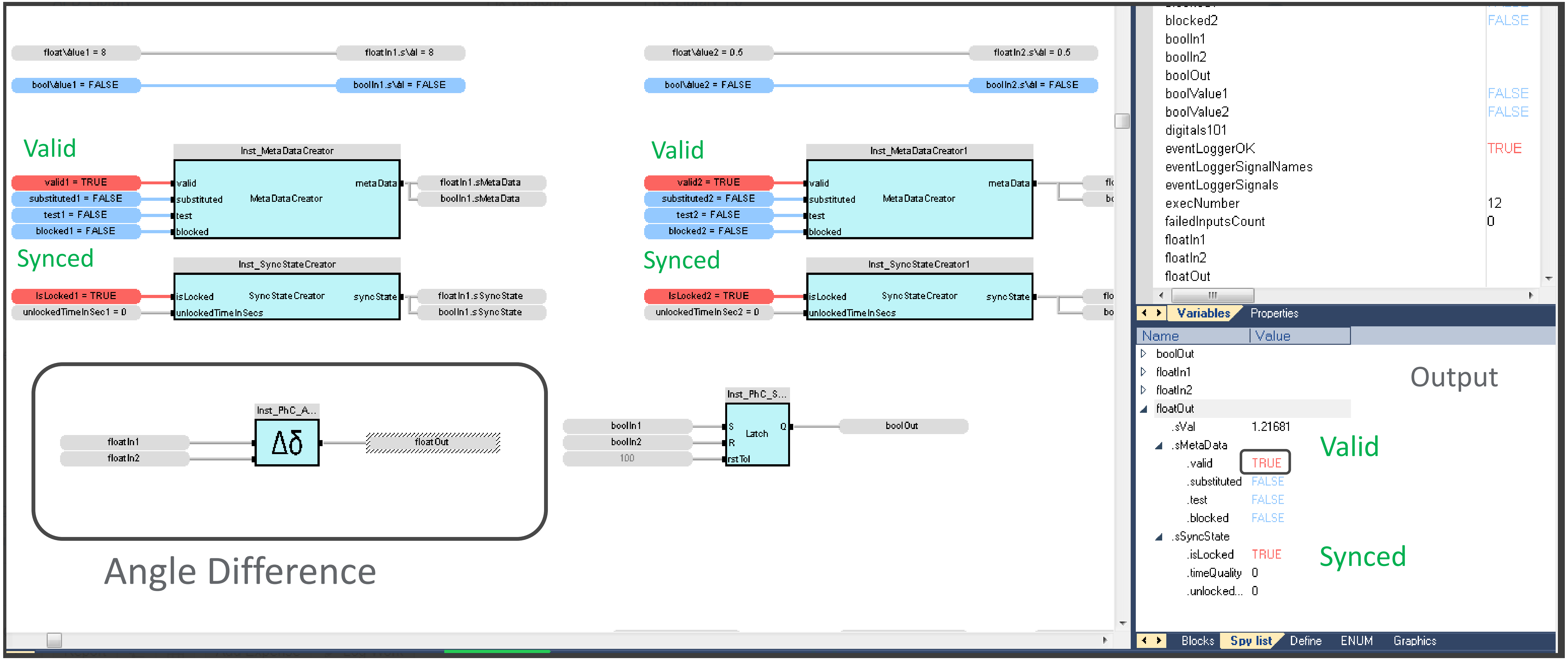
Frequency (or other) Difference



Frequency in the system does not change quickly, PMUs also apply filtering to frequency signal.

Even if one or both of frequencies is not GPS-locked, the output is VALID

Angle Difference block Demonstration



Valid

Synced

Valid

Synced

Angle Difference

blocked2	FALSE
boolIn1	
boolIn2	
boolOut	
boolValue1	FALSE
boolValue2	FALSE
digitals101	
eventLoggerOK	TRUE
eventLoggerSignalNames	
eventLoggerSignals	
execNumber	12
failedInputsCount	0
floatIn1	
floatIn2	
floatOut	

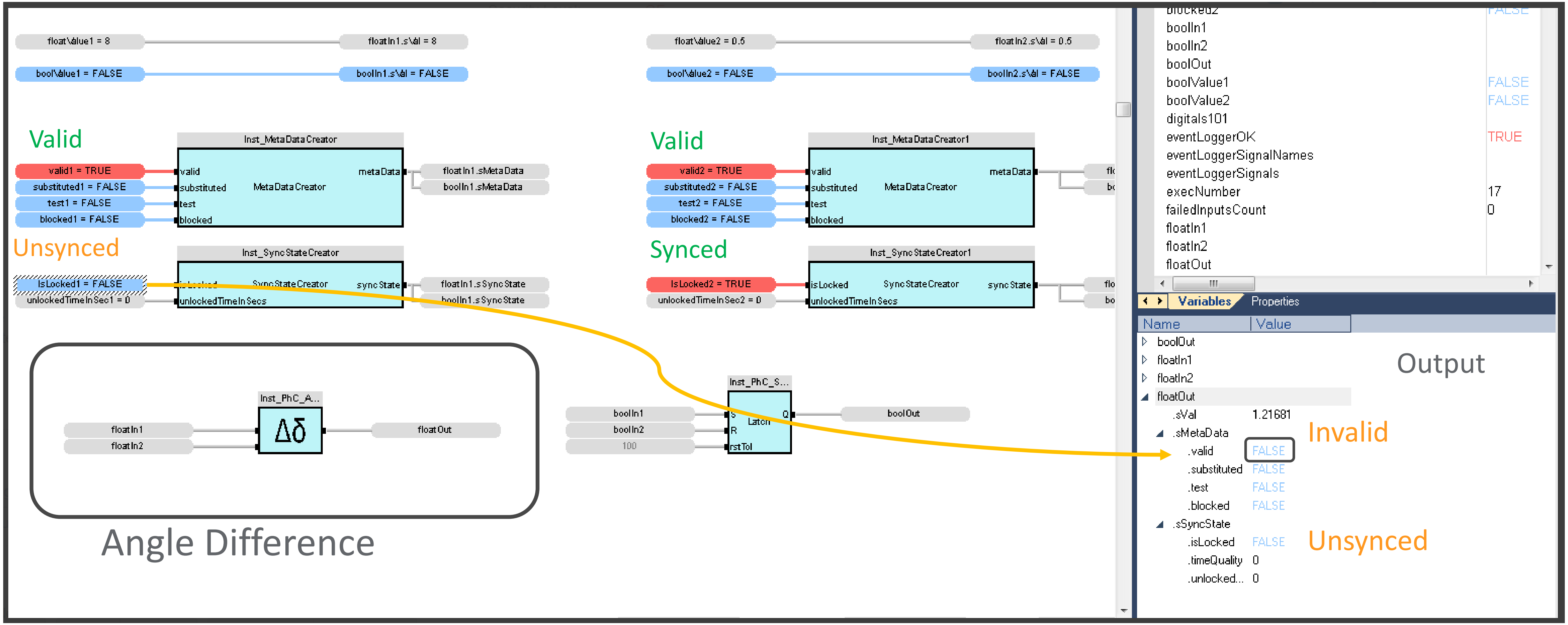
Name	Value
boolOut	
floatIn1	
floatIn2	
floatOut	
.sVal	1.21681
.sMetaData	
.valid	TRUE
.substituted	FALSE
.test	FALSE
.blocked	FALSE
.sSyncState	
.isLocked	TRUE
.timeQuality	0
.unlocked...	0

Output

Valid

Synced

Angle Difference block Demonstration



Valid

Unsynced

Valid

Synced

Angle Difference

Output

Invalid

Unsynced

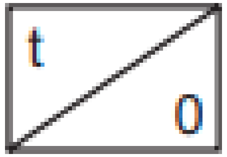
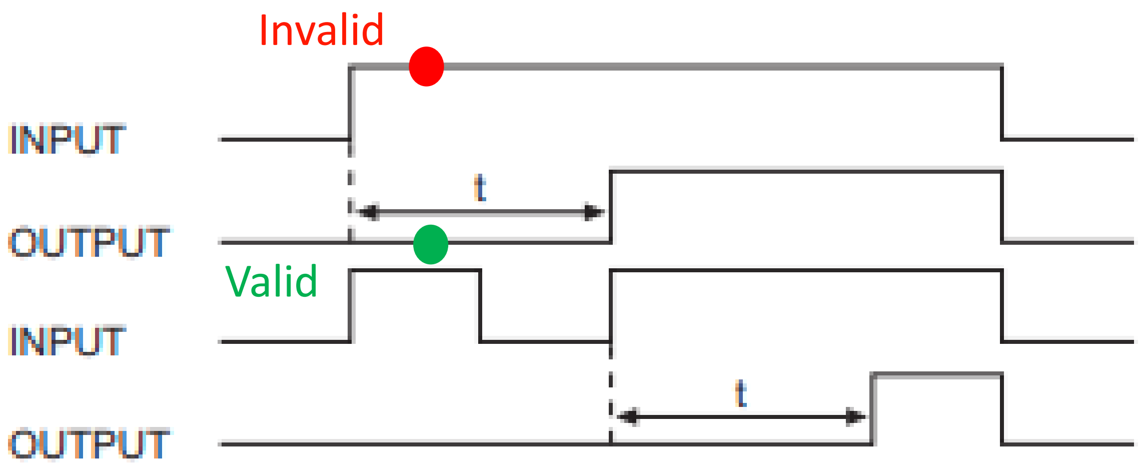
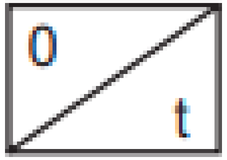
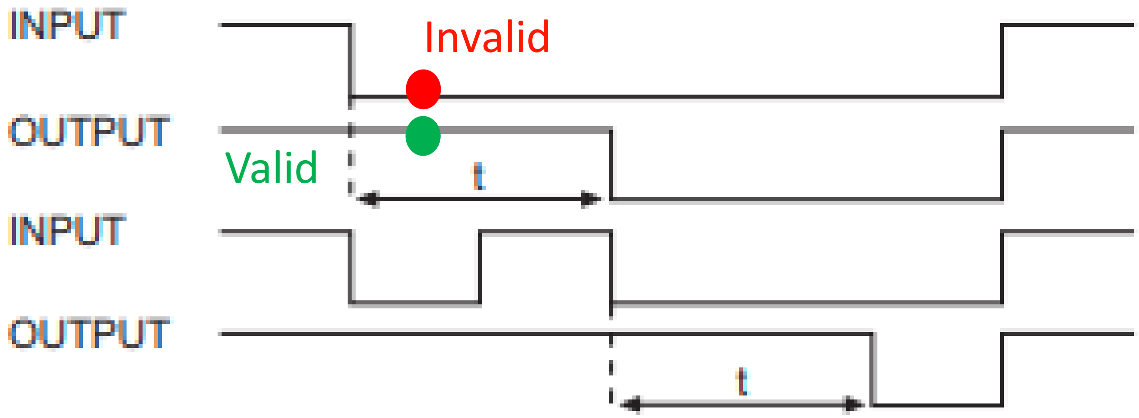
Name	Value
blocked2	FALSE
boolIn1	
boolIn2	
boolOut	
boolValue1	FALSE
boolValue2	FALSE
digitals101	
eventLoggerOK	TRUE
eventLoggerSignalNames	
eventLoggerSignals	
execNumber	17
failedInputsCount	0
floatIn1	
floatIn2	
floatOut	

Name	Value
boolOut	
floatIn1	
floatIn2	
floatOut	
.sVal	1.21681
.sMetaData	
.valid	FALSE
.substituted	FALSE
.test	FALSE
.blocked	FALSE
.sSyncState	
.isLocked	FALSE
.timeQuality	0
.unlocked...	0

STATEFUL FUNCTION: TIMERS, DELAYS, FILTERS (IIR AND FIR)

WORK STILL ONGOING

Pickup and Dropoff Timers

	Delay on pick-up timer, t	 Invalid INPUT t OUTPUT Valid INPUT t OUTPUT
	Delay on drop-off timer, t	 INPUT Invalid OUTPUT Valid t INPUT OUTPUT t

Quality Considerations for Timers

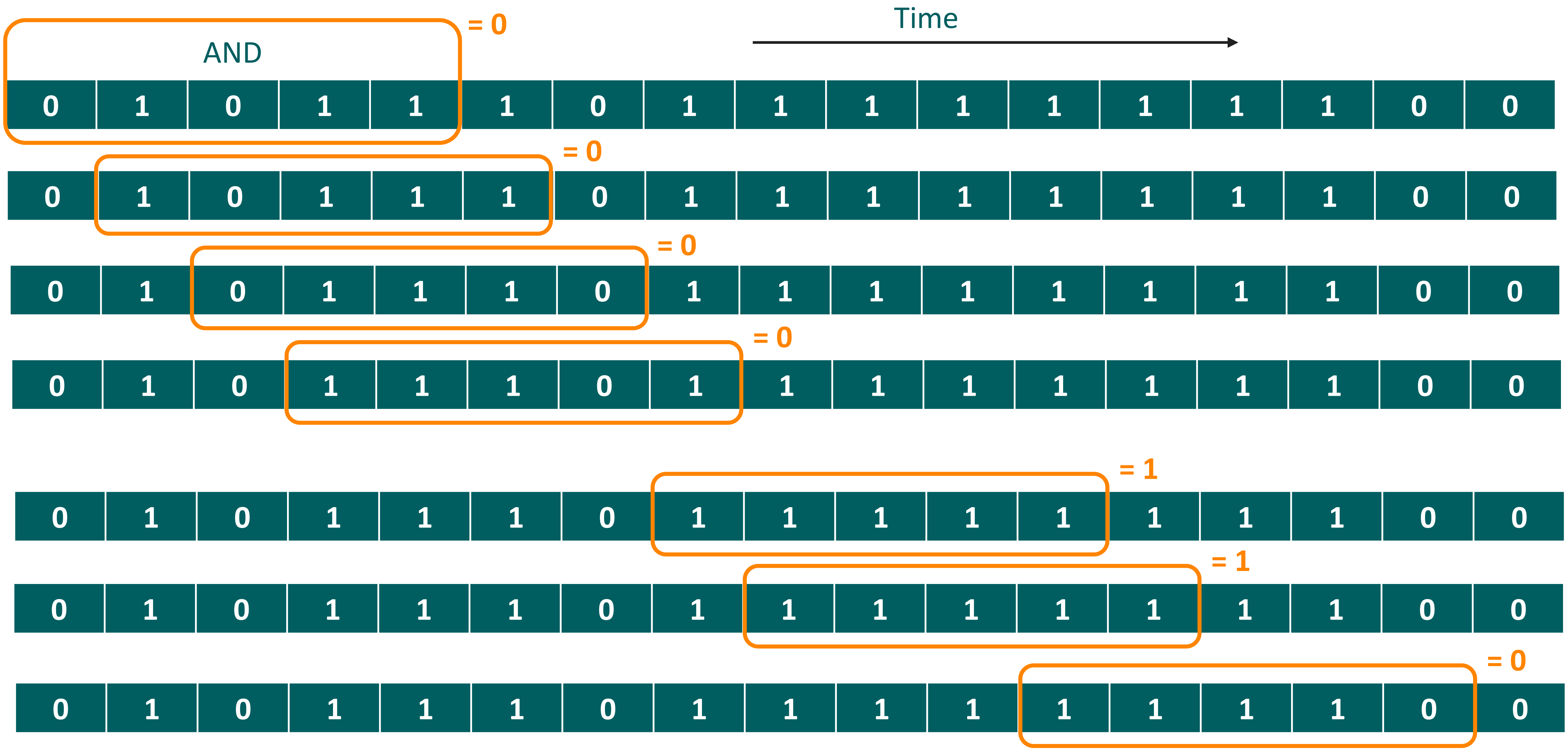
Intuitively, it was obvious that there were moments when the **current** state of the timer output is independent from the state (and hence validity) of the input. The output must then be considered **valid** even if the input is **invalid**.

This consideration was, unfortunately, not rigorous enough – it was not clear how to deal with other quality attributes. A more robust mathematical formulation was needed.

Proposed approach: Express all timers through AND and OR logic gates applied to buffered input values, with the length of the buffer equal to the timer delay. We already have a rigorous solution for quality processing in logic gates.

Quality Considerations for Timers

PU Timer Delay = 5 samples



0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

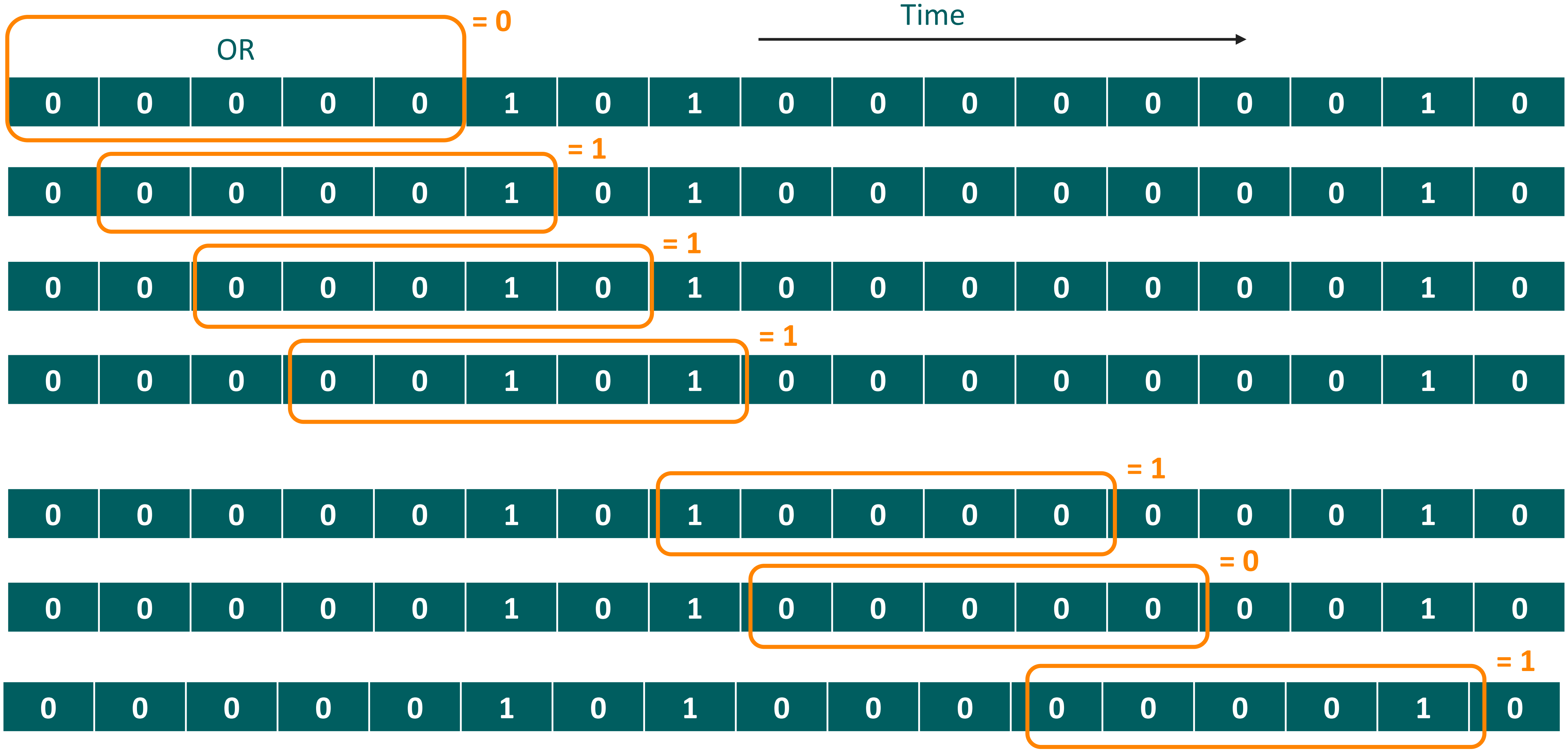
0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0

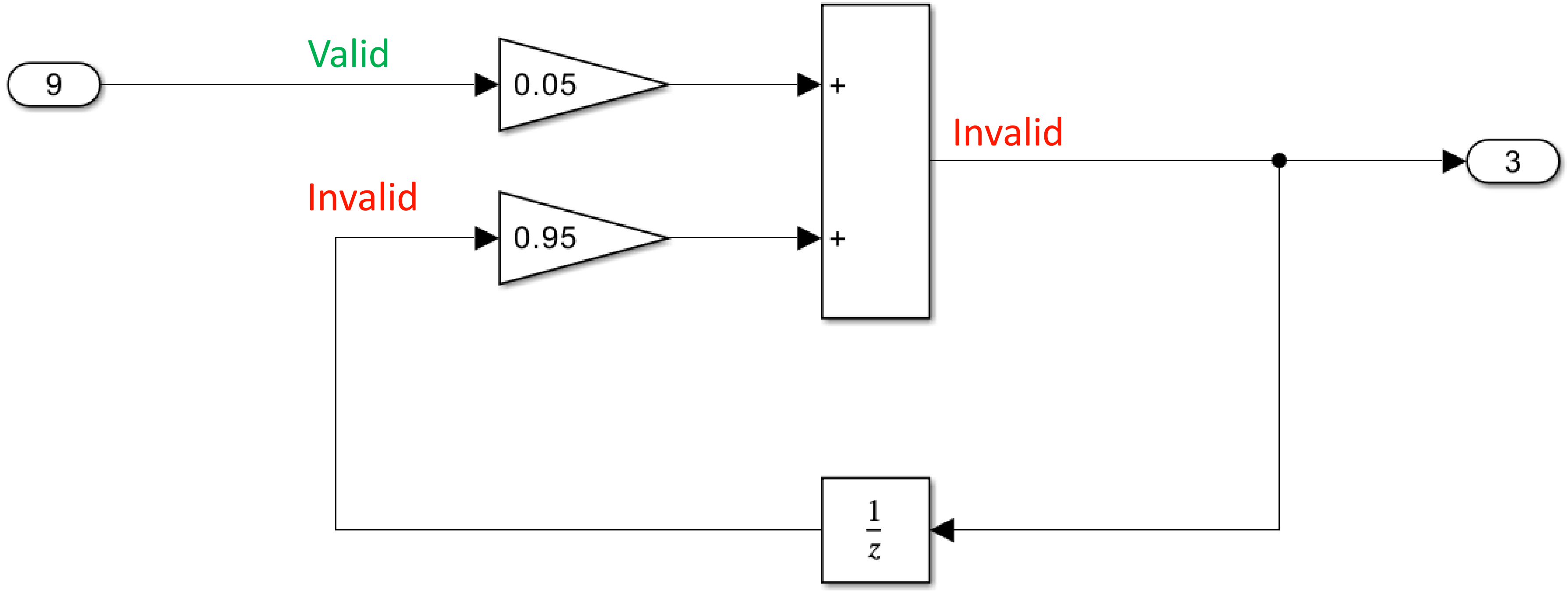
Quality Considerations for Timers

DO Timer Delay = 5 samples



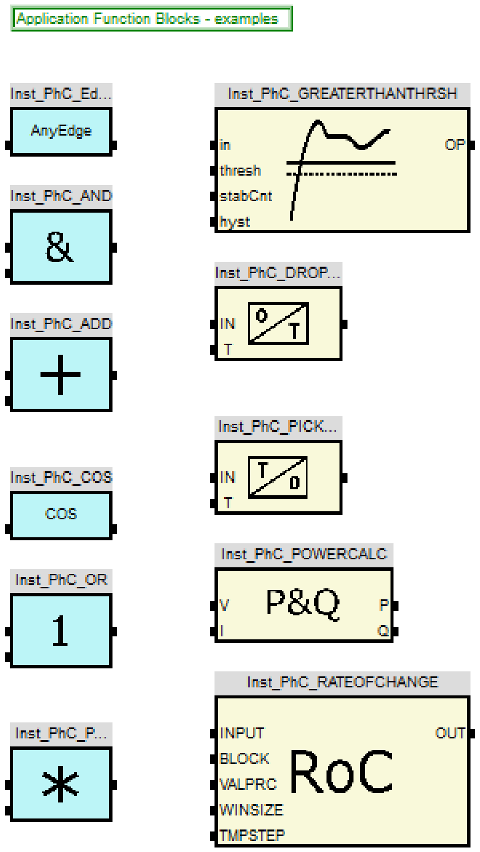
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0

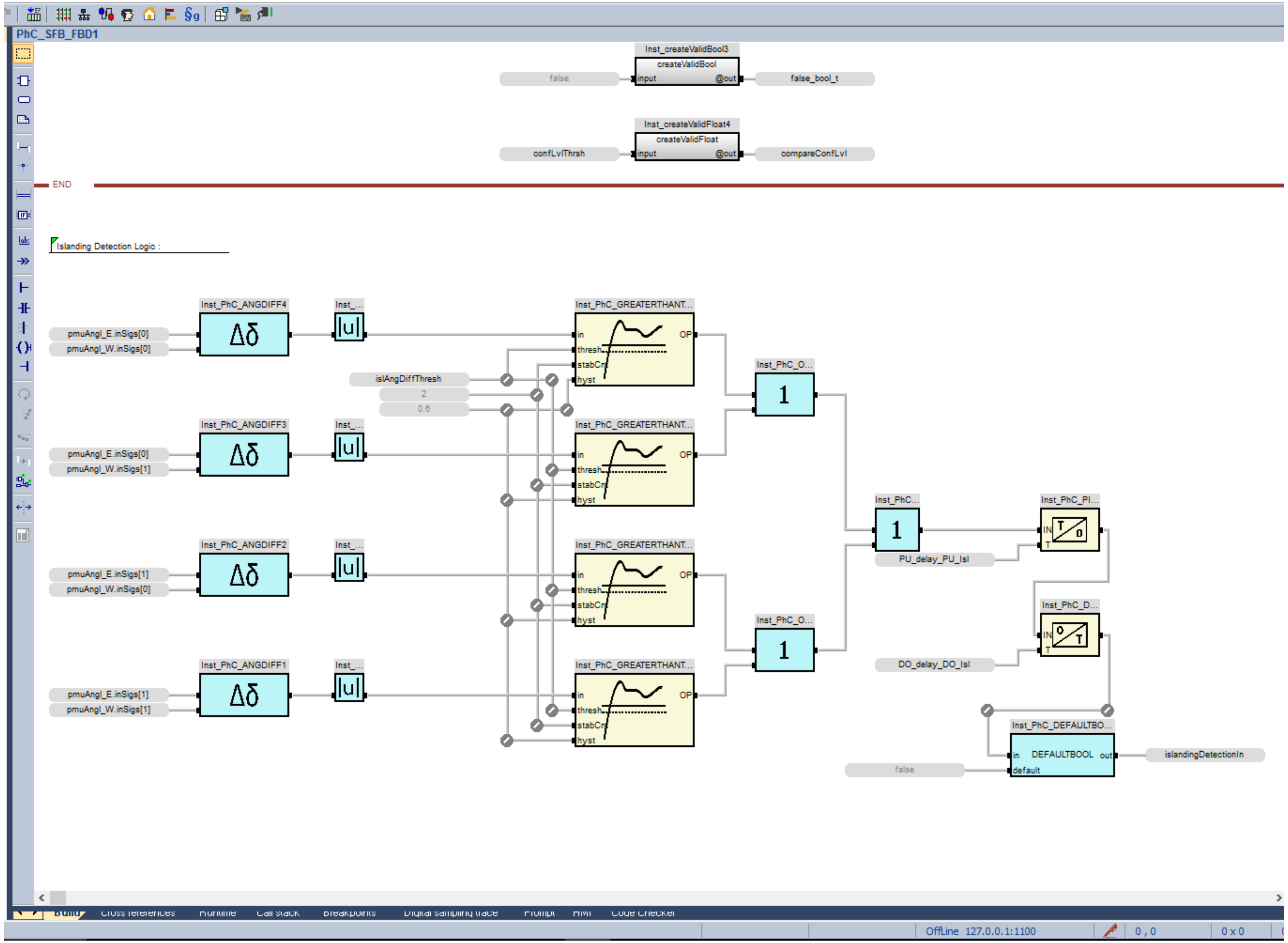
Quality Considerations for IIR Filter



If the input gets **invalid** once, the output becomes **invalid** and can never become **valid** again. No rigorous mathematical solution found so far. The engineering approach is to introduce a configurable timer that will reset the output to **valid** after the input has been valid for a certain duration of time.

Using Quality Aware Function Blocks in real projects





GE VERNOVA