Architecture Design Principles for the Integration of Synchronization Interfaces into Network-on-Chip Switches

Daniele Ludovici – Alessandro Strano[‡] – Davide Bertozzi[‡]

Computer Engineering lab – TUDelft - NL

[‡] *MPSoC research group – UNIFE – Italy*

daniele@ce.et.tudelft.nl

OUTLINE

- ❑ GALS Network-on-Chip design paradigm
- ❑ Different synchronization models
  - ✓ Methodology towards a synchronizer integration
- ❑ Tightly Coupled Mesochronous synchronizer
- ❑ Tightly Coupled Dual-clock FIFO
- ❑ Results
  - ✓ Performance, area overhead, power consumption
- ❑ Conclusions

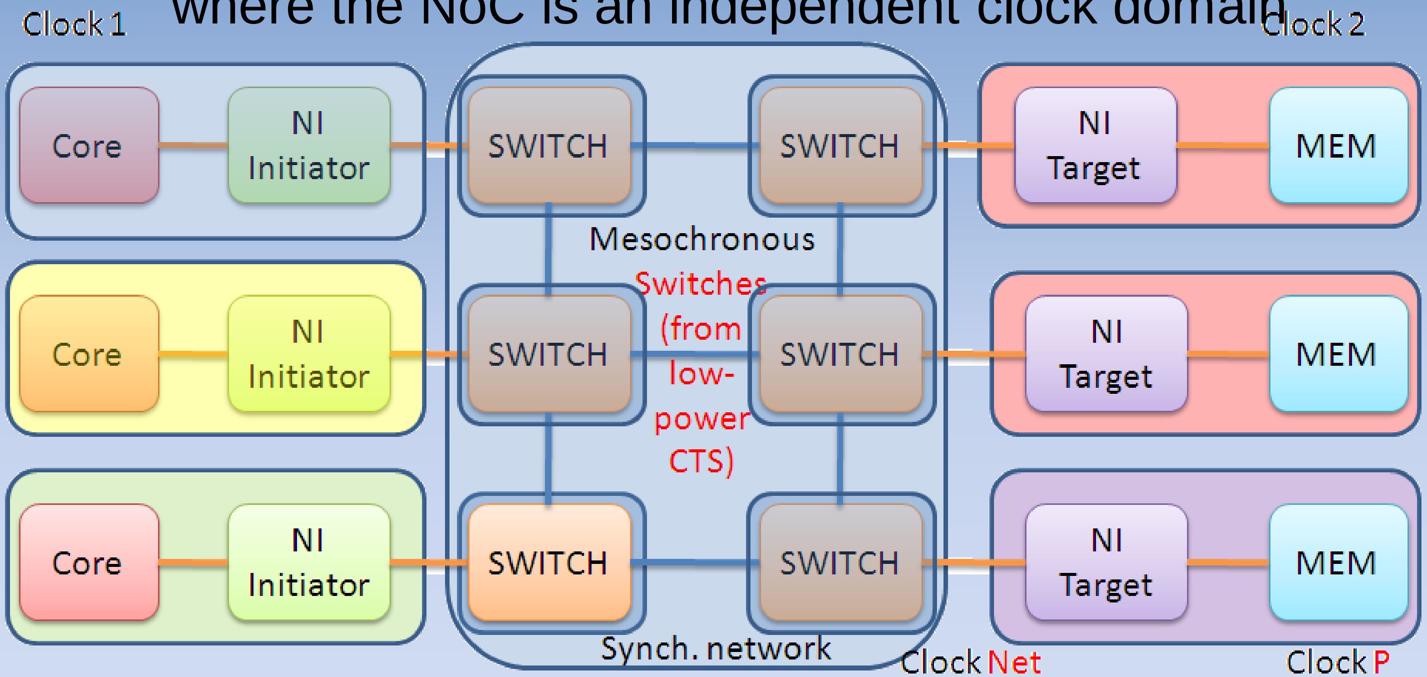
MOTIVATION

- There is today little doubt on the fact that a high-performance and cost-effective NoC can be designed in 45nm and beyond under a relaxed synchronization assumption
  - ✓ interconnect delay, process variation, etc.

- A possible solution: GALS NoC
  - ✓ Processing blocks are separated and clocked independently
  - ✓ No global clock distribution => simplified timing closure
  - ✓ No rigid timing constraints between local clock domains

GALS implementation

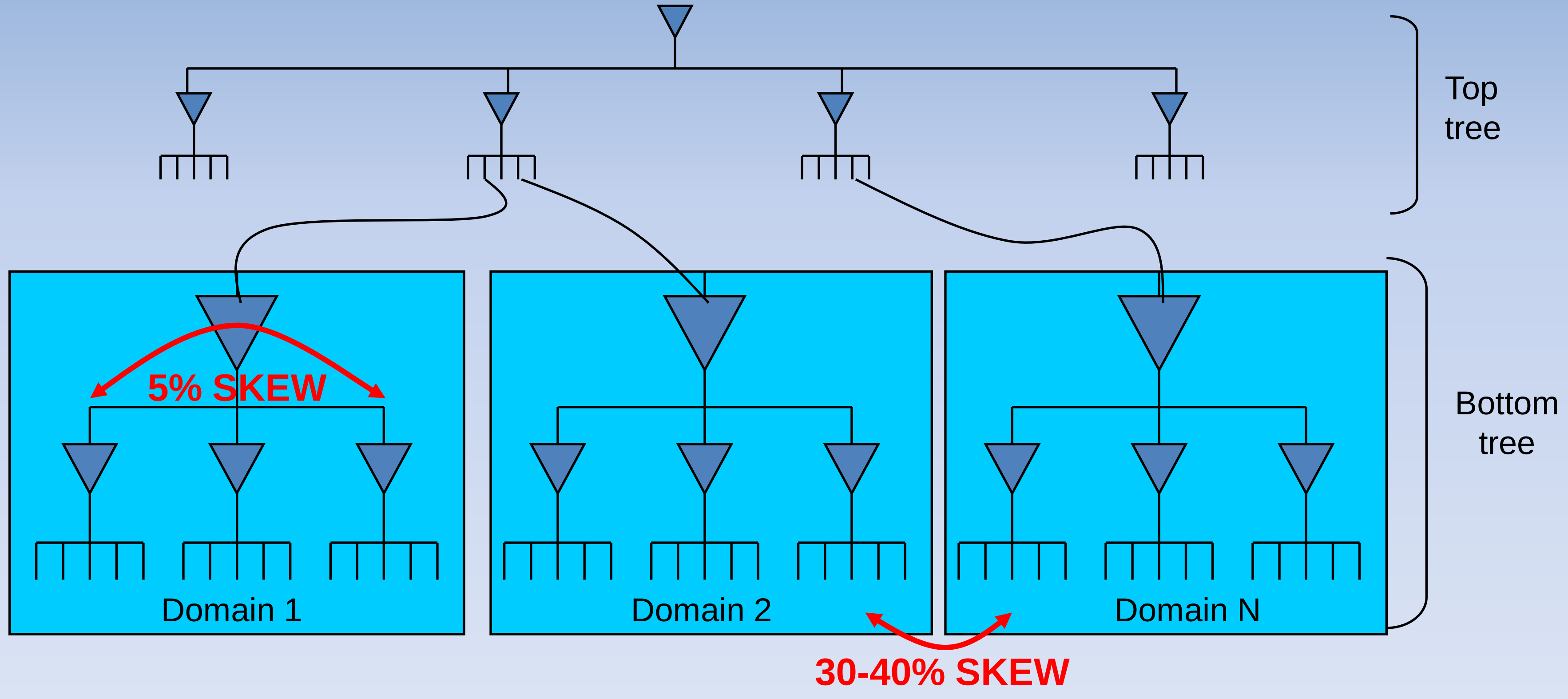
We chose one GALS implementation variant where the NoC is an independent clock domain



- Conscious use of area/power expensive ***dual-clock FIFOs*** *for throughput sensitive link to IP cores* ***(used only at the network boundary)***
- ***More compact mesochronous synchronizers*** are used in the network
- *Hierarchical Clock Tree Synthesis* ***to relieve clock phase offset constraints***

Mesochronous Synchronization

Hierarchical clock tree with relaxed skew constraints might significantly decrease clock tree power and make the chip-wide NoC domain feasible

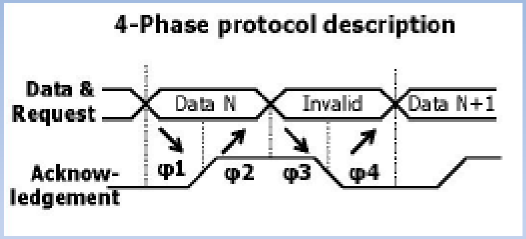


Challenge: implementing cost-effective mesochronous synchronization

[Source: MIRO-PANADES08]

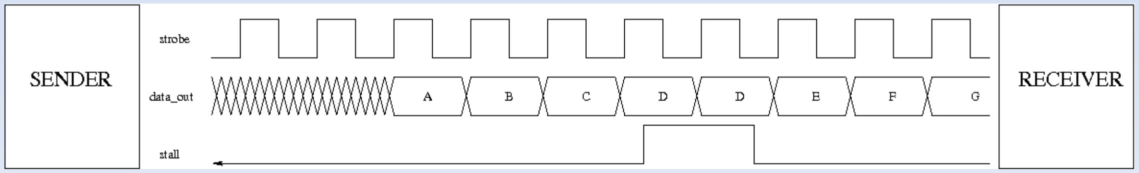
SYNCHRONIZATION MODELS

- ***Single transaction handshake design style***
  - ✓ Acknowledgment for each data word
  - ✓ Latency for each data transfer and lower throughput
  - ✓ Requires good asynch. knowledge
  - ✓ Low maturity for EDA tools



[Source: LATTARD07]

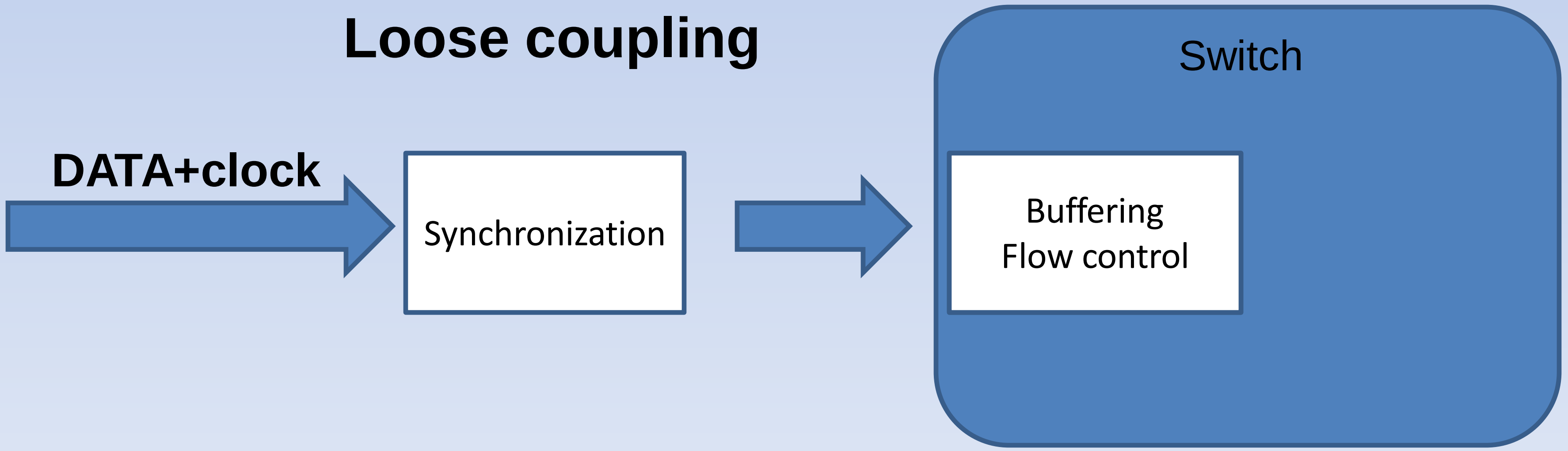
- ***Source synchronous design style (our choice!)***
  - ✓ The clock is routed along with the data it is going to strobe
  - ✓ Good for high-data rates
  - ✓ Requires only an incremental effort with current EDA tool flows
  - ✓ Potentially area/power-hungry, reliability concern



A STEP FORWARD

- With conventional design techniques, source synchronous interfaces are external blocks to the modules they synchronize => synch latency, area and power overhead fully exposed
- Mitigate synchronization overhead by co-designing the interface with the NoC submodules => to the limit: full merging

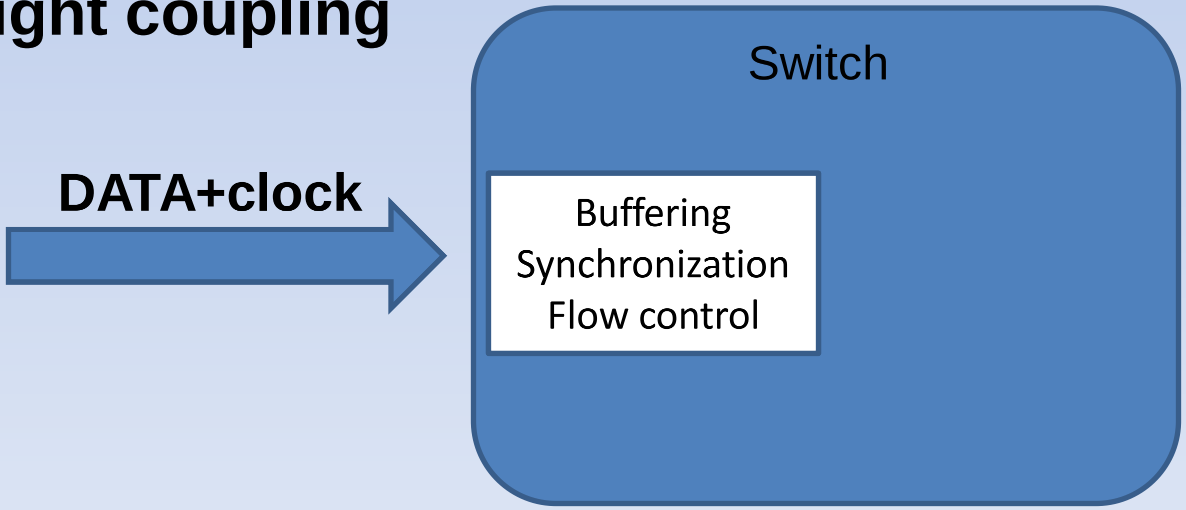
Loose coupling



A STEP FORWARD

- With conventional design techniques, source synchronous interfaces are external blocks to the modules they synchronize => synch latency, area and power overhead fully exposed
- Mitigate synchronization overhead by co-designing the interface with the NoC submodules => to the limit: full merging

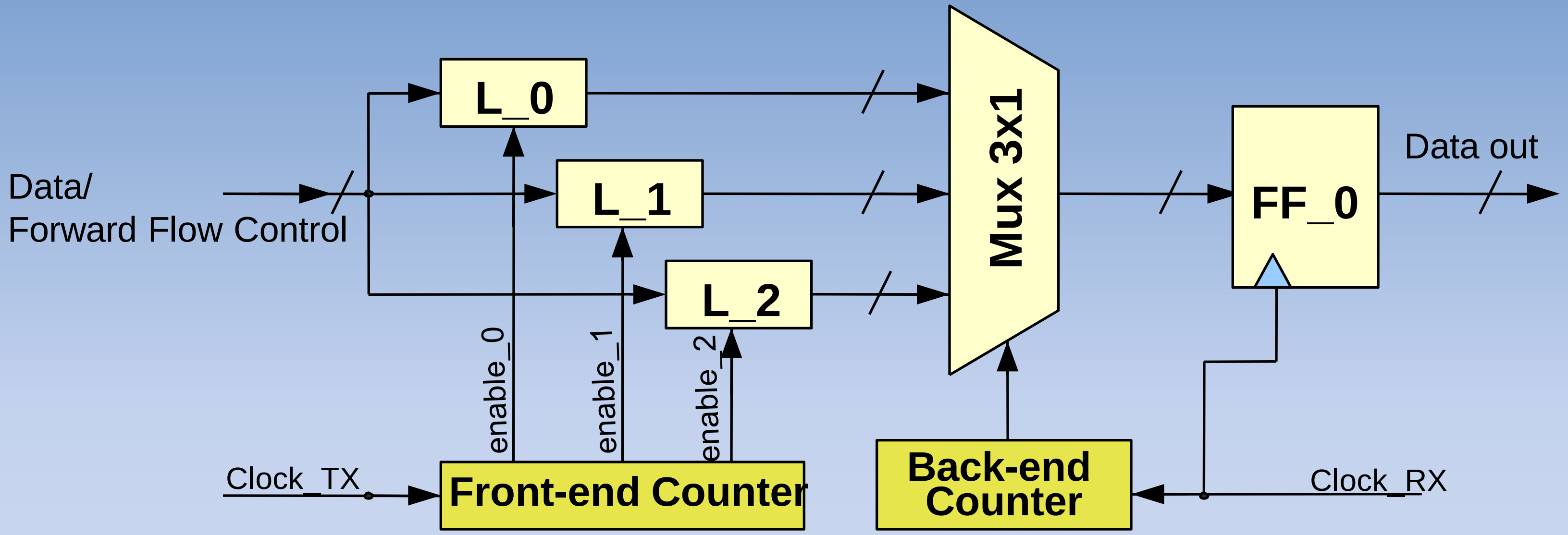
Tight coupling



achievement of major savings thanks to the sharing of expensive buffers

Tightly coupled mesochronous synchronizer with the switch architecture

Proposed synchronizer

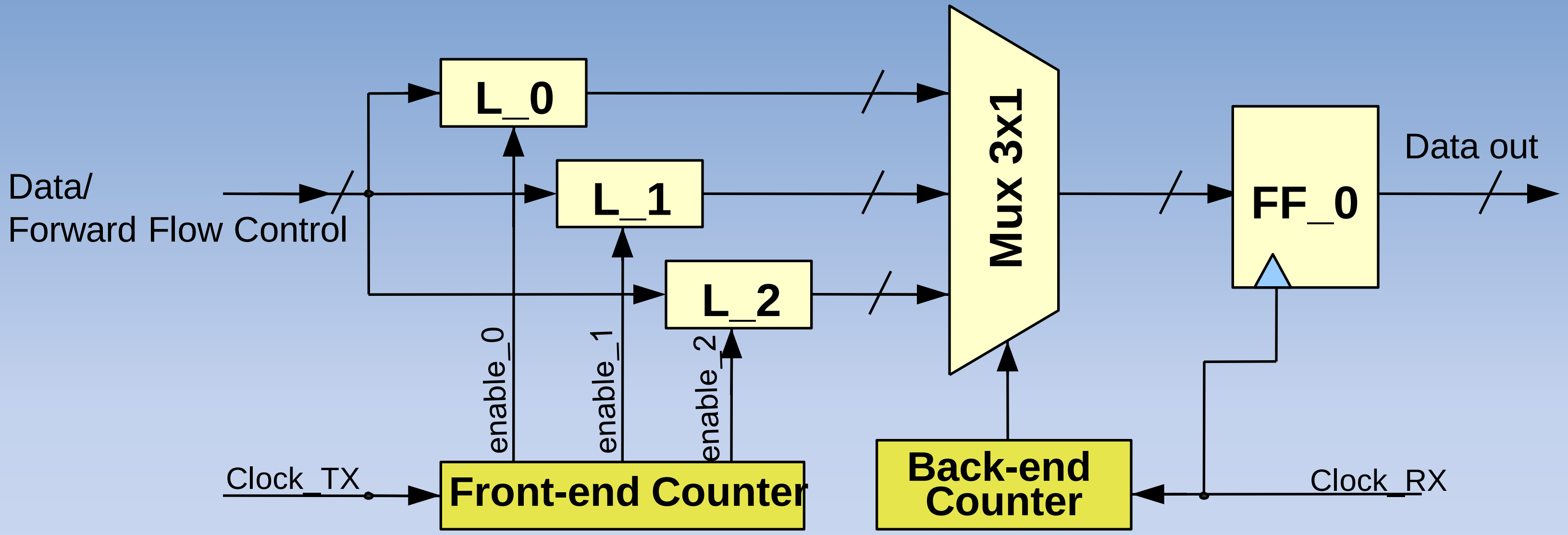


Underlying principle: Information can safely settle in the front-end latches before being sampled by the target domain clock

Front-end:

- Clock_TX used as a strobe signal for data and flow control wires, thus avoiding timing problems associated with phase offset of clock signals
- Sampling through a number of latches used in a rotating fashion based on a counter

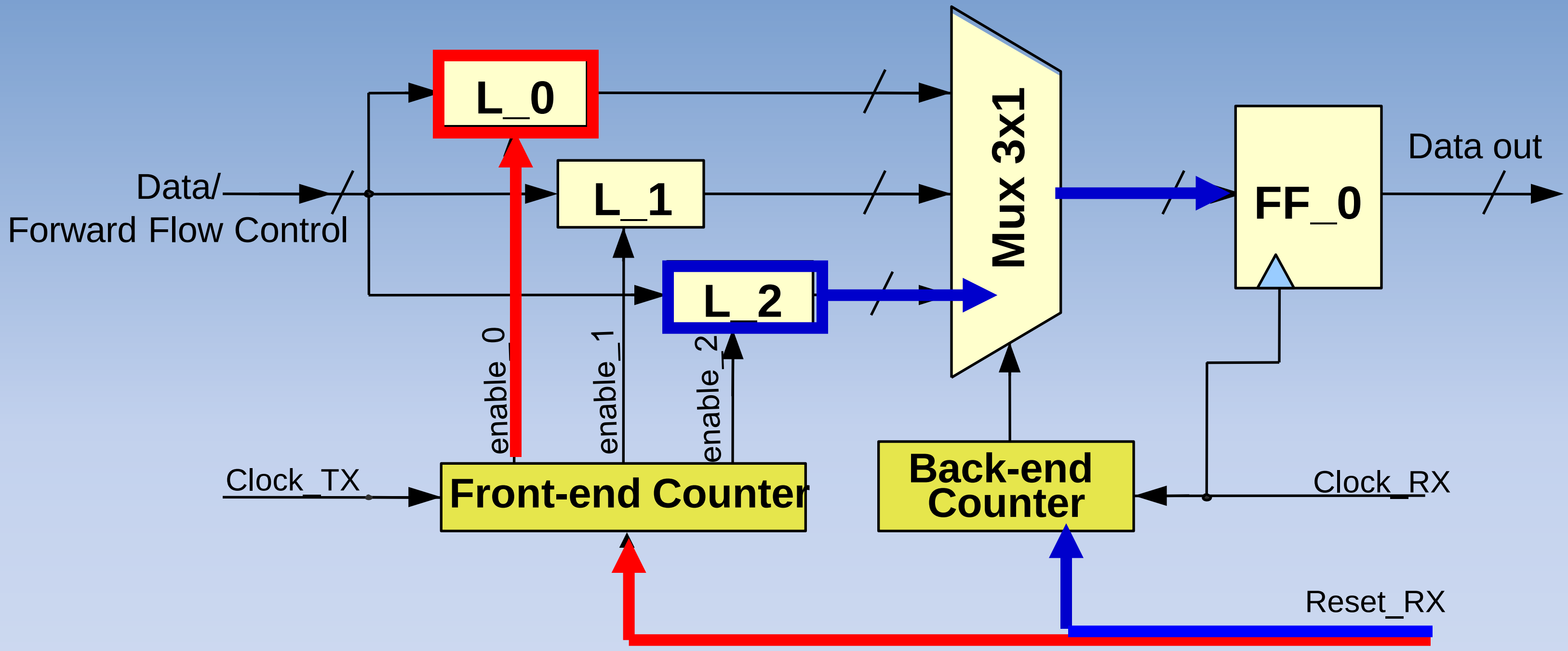
Proposed synchronizer



Underlying principle: Information can safely settle in the front-end latches before being sampled by the target domain clock

Back-end:
- Leverages local clock of the RX domain
- Samples data from one of the latches in the front-end thanks to multiplexing logic based on a counter

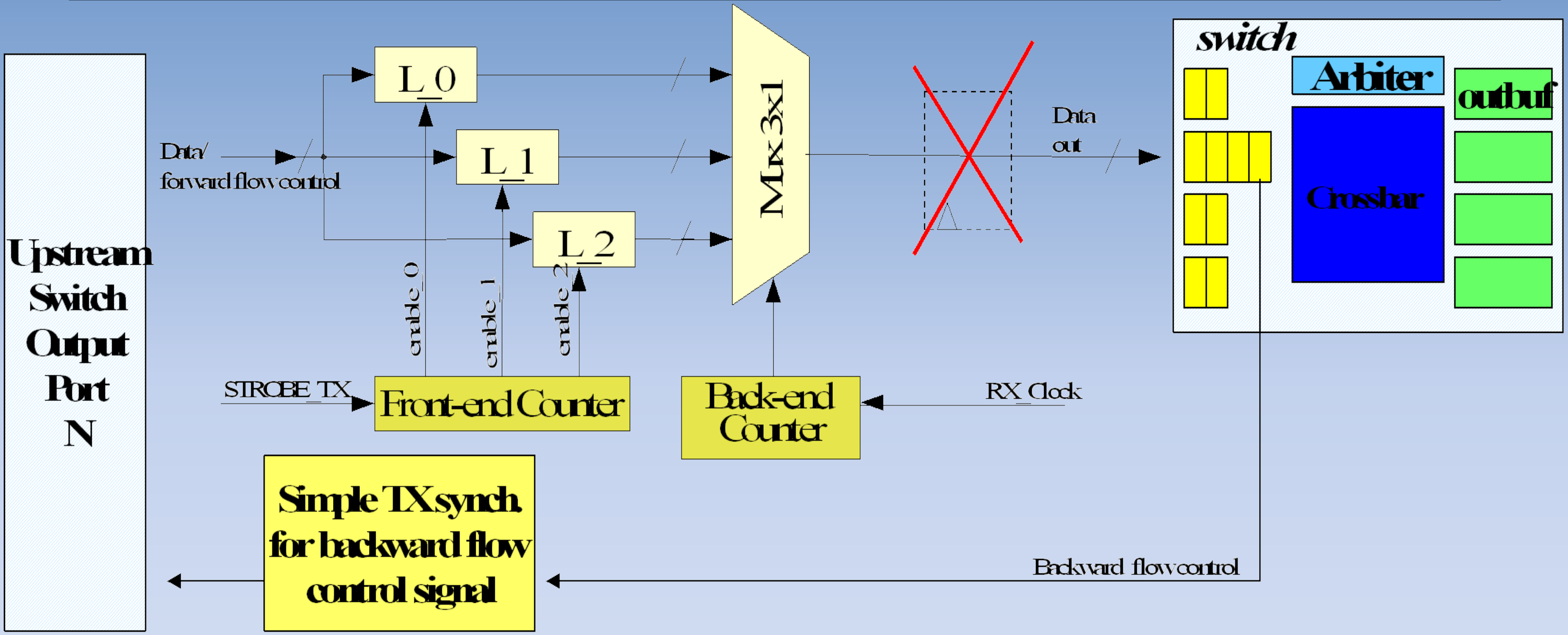
Proposed synchronizer



- 3 input latch banks ensure timing constraints are safely met
  - ✓ data stability window at latch outputs is enough to tolerate wide range of clock phase offset
  - ✓ phase detector can be avoided
    - ✓A unique bootstrap configuration can deal with all phase skew scenarios

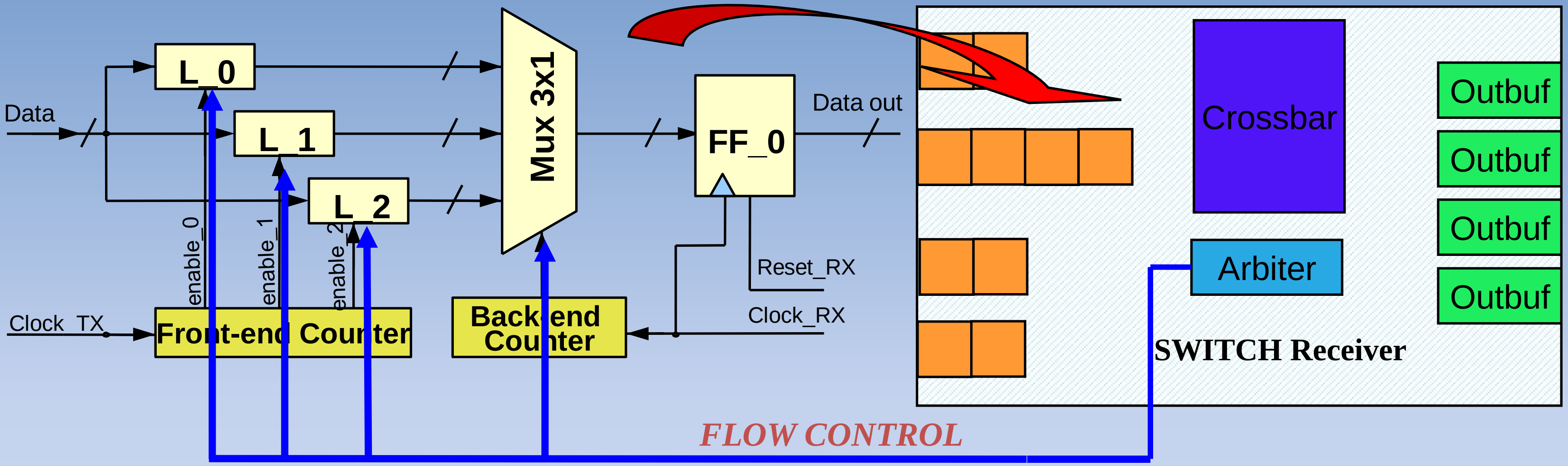
- Main challenge:
  - ✓enforce timing margins for the NoC domain
  - ✓study implications of synchronizer integration into a NoC (e.g., flow control)

Flow control



- Flow control implications considered
  - ✓ xpipes comes with stall/go flow control; 2-stage buffer at each switch input
  - ✓ Optimization: the back-end flip-flop IS the switch input buffer
  - ✓ At least a 4 slot buffer is needed to keep using stall/go
  - ✓ A small single-bit synchronizer needed to synchronize backward flow control signal

Optimization



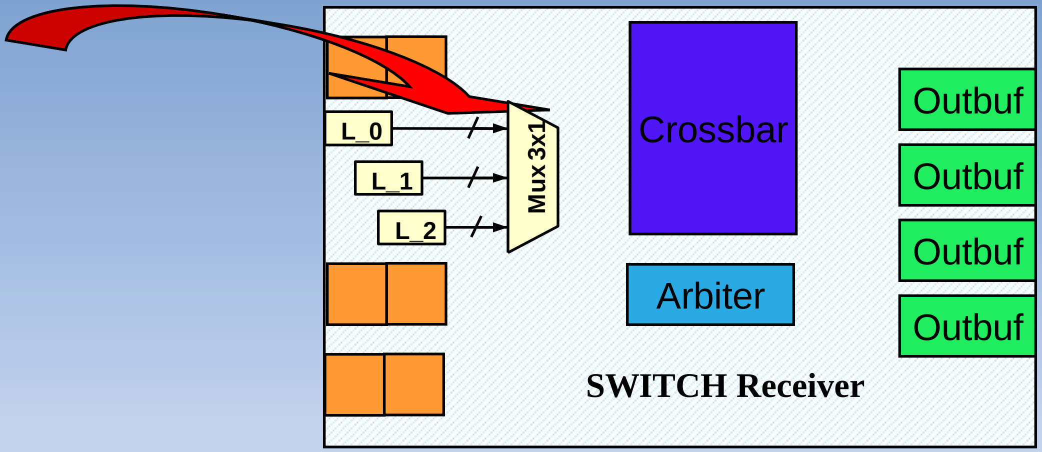
-Why not bringing flow control to the synchronizer latches as well?

-So that data can be stalled there, without need for extra buffer in the switch.

-Why not using the synchronizer IN PLACE OF the switch input buffer at all?

A multi-purpose switch input buffer (buffering, synchronization and flow control) might lead to large area/power savings, lower latency and would preserve modularity

Optimization



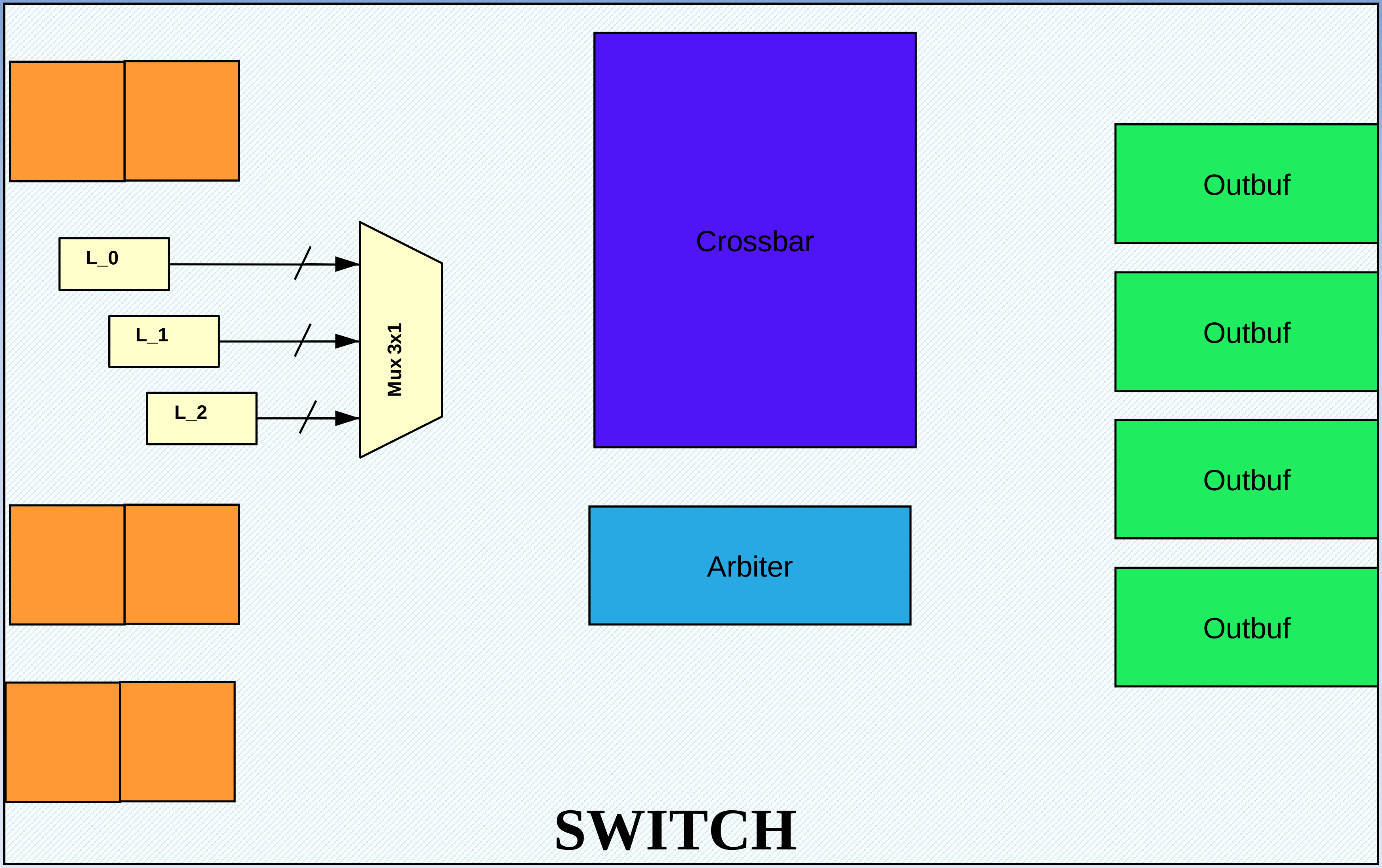
-Why not bringing flow control to the synchronizer latches as well?

-So that data can be stalled there, without need for extra buffer in the switch.

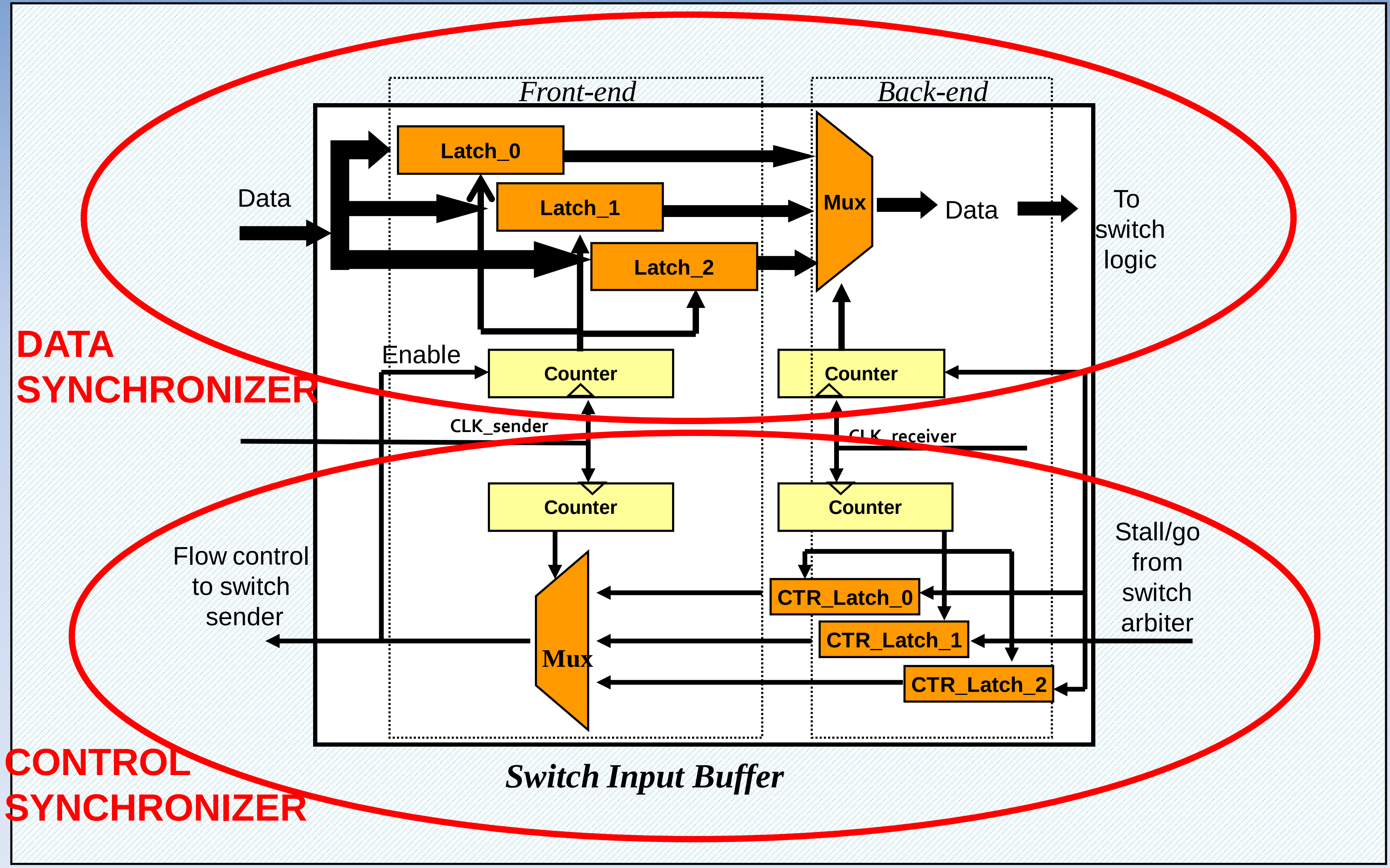
-Why not using the synchronizer IN PLACE OF the switch input buffer at all?

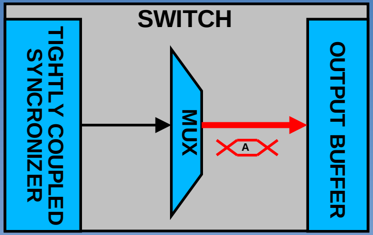
A multi-purpose switch input buffer (buffering, synchronization and flow control) might lead to large area/power savings, lower latency and would preserve modularity

Tightly-coupled synchronizer (in the switch architecture)

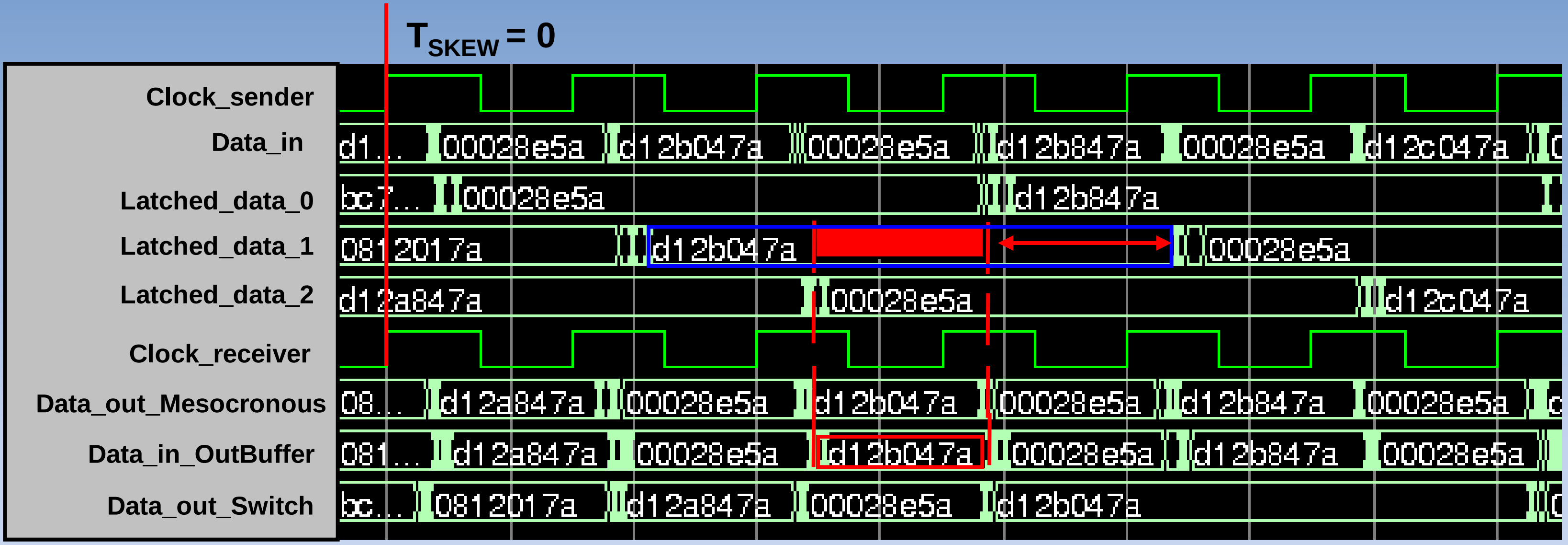


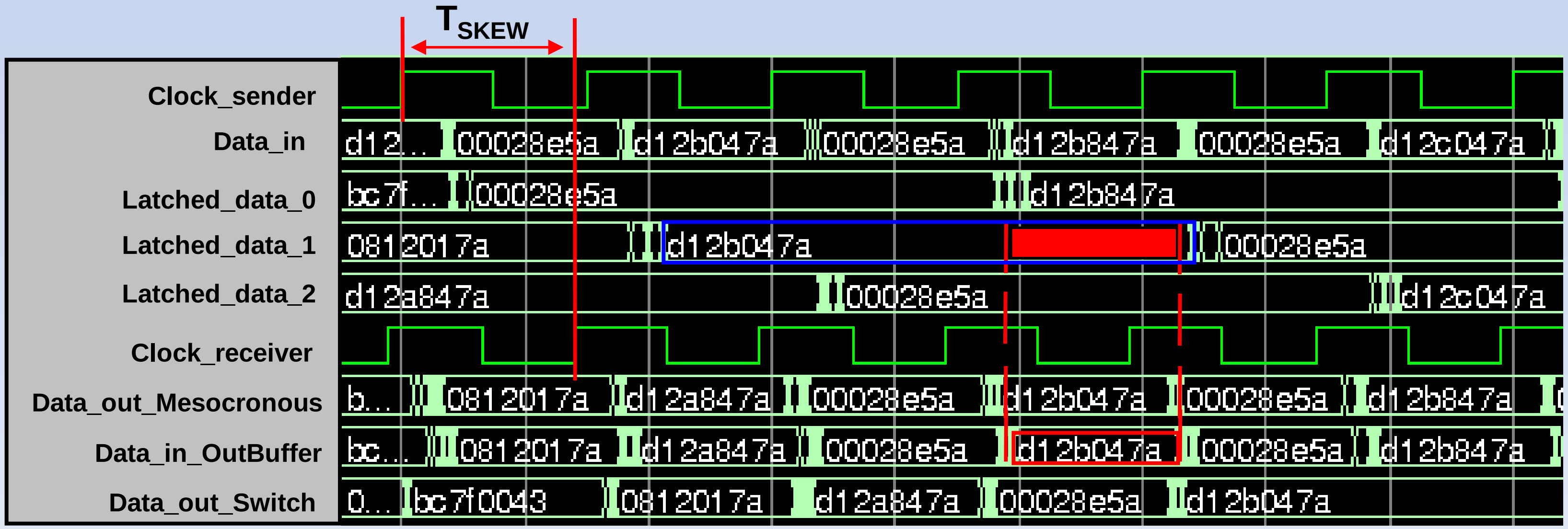
Tightly-coupled synchronizer (in the switch architecture)



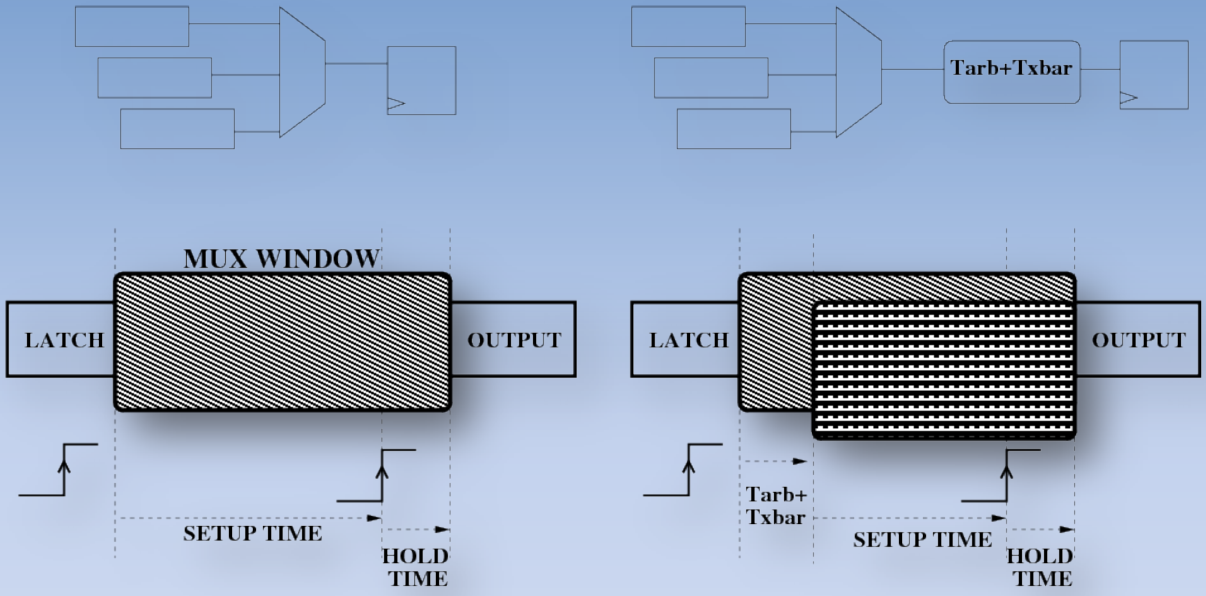


TIGHTLY COUPLED SYNCRONIZER



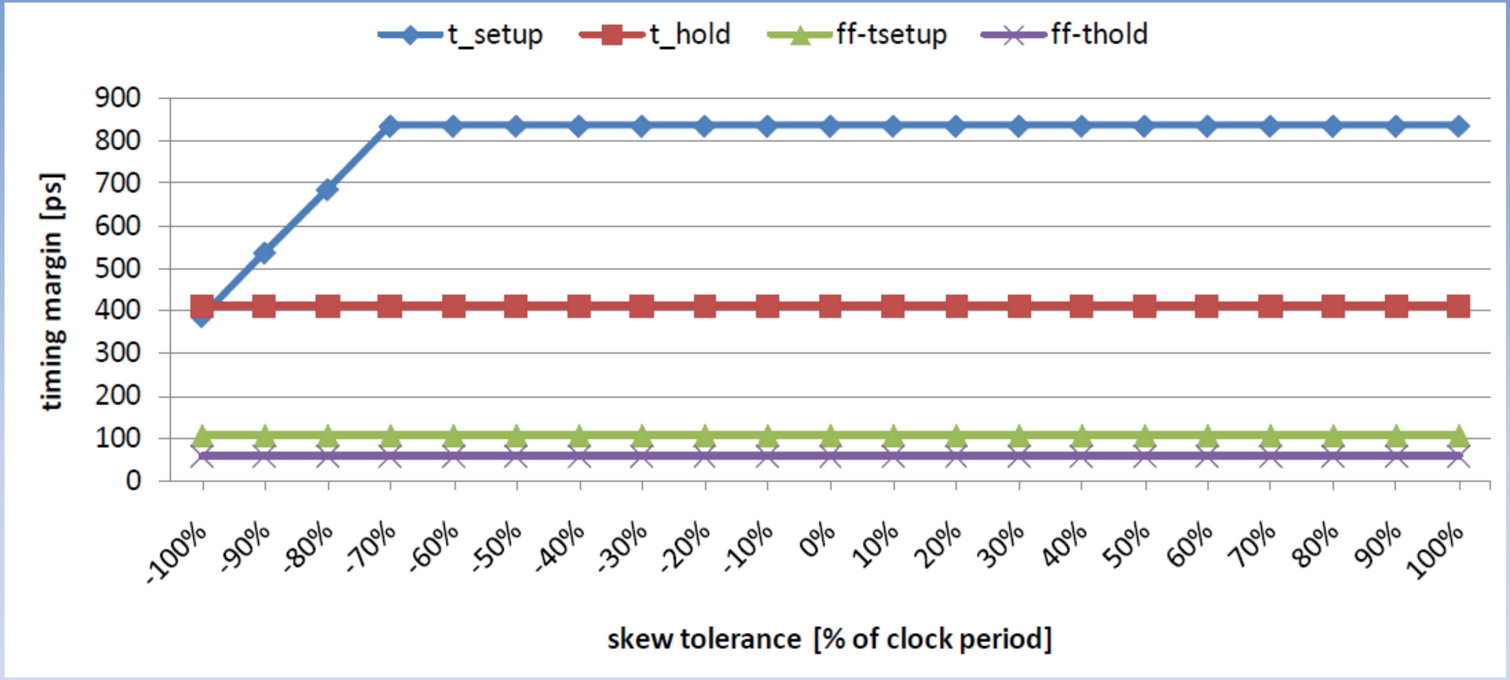


SKEW TOLERANCE



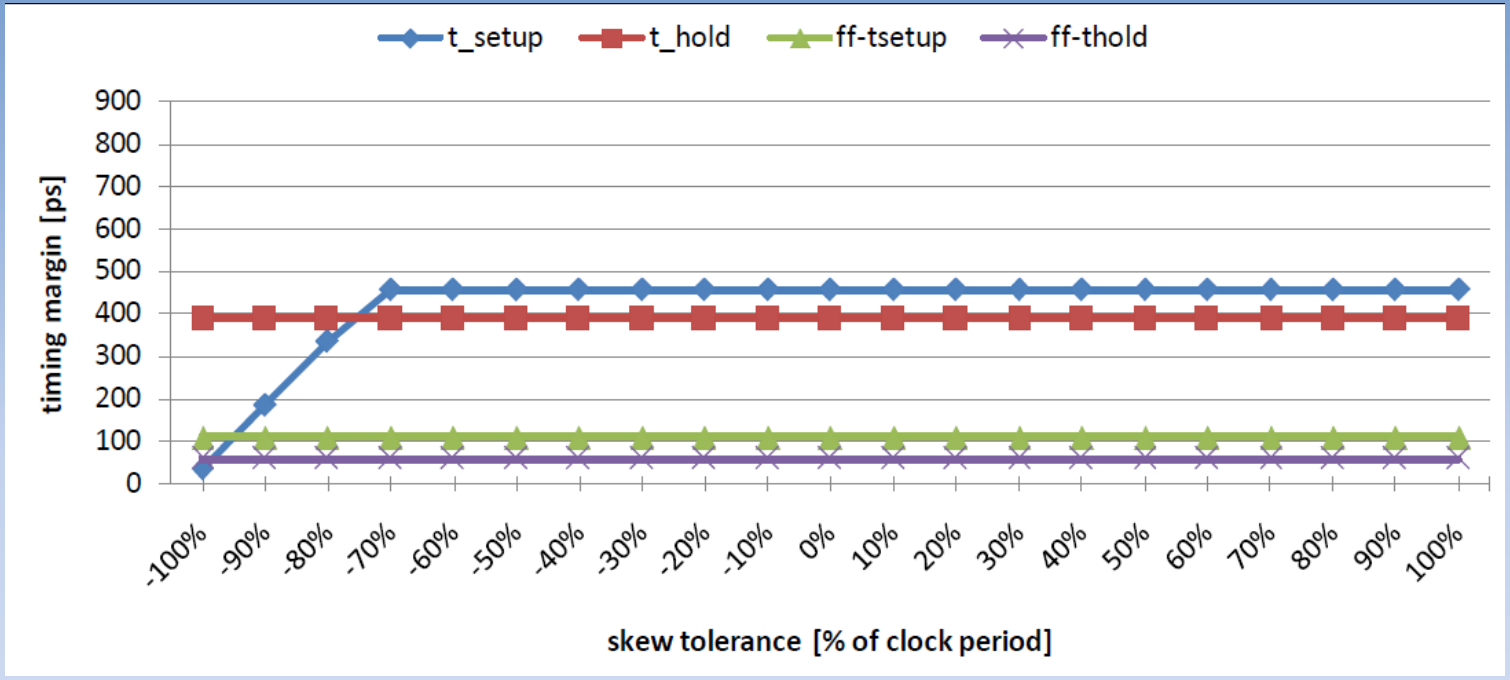
- Setup Time: from the beginning of mux window to the rising edge of the sampling element.
- Hold Time: from the rising edge of the sampling element to the end of the mux window.
- For the tightly coupled these metrics are taken at the output buffer. Tarb+Txbar reduces “setup time” for the tightly coupled synchronizer.

Loosely Coupled Skew Tolerance



- Pos. and Neg. skew are expressed as % of the clock period.
- Setup and Hold time compared with those of a FF in 65nm lib.
- Hold Time is stable and it has a solid margin.
- Setup Time decreases when latch outputs end switching inside the mux window BUT there is still a safe margin!

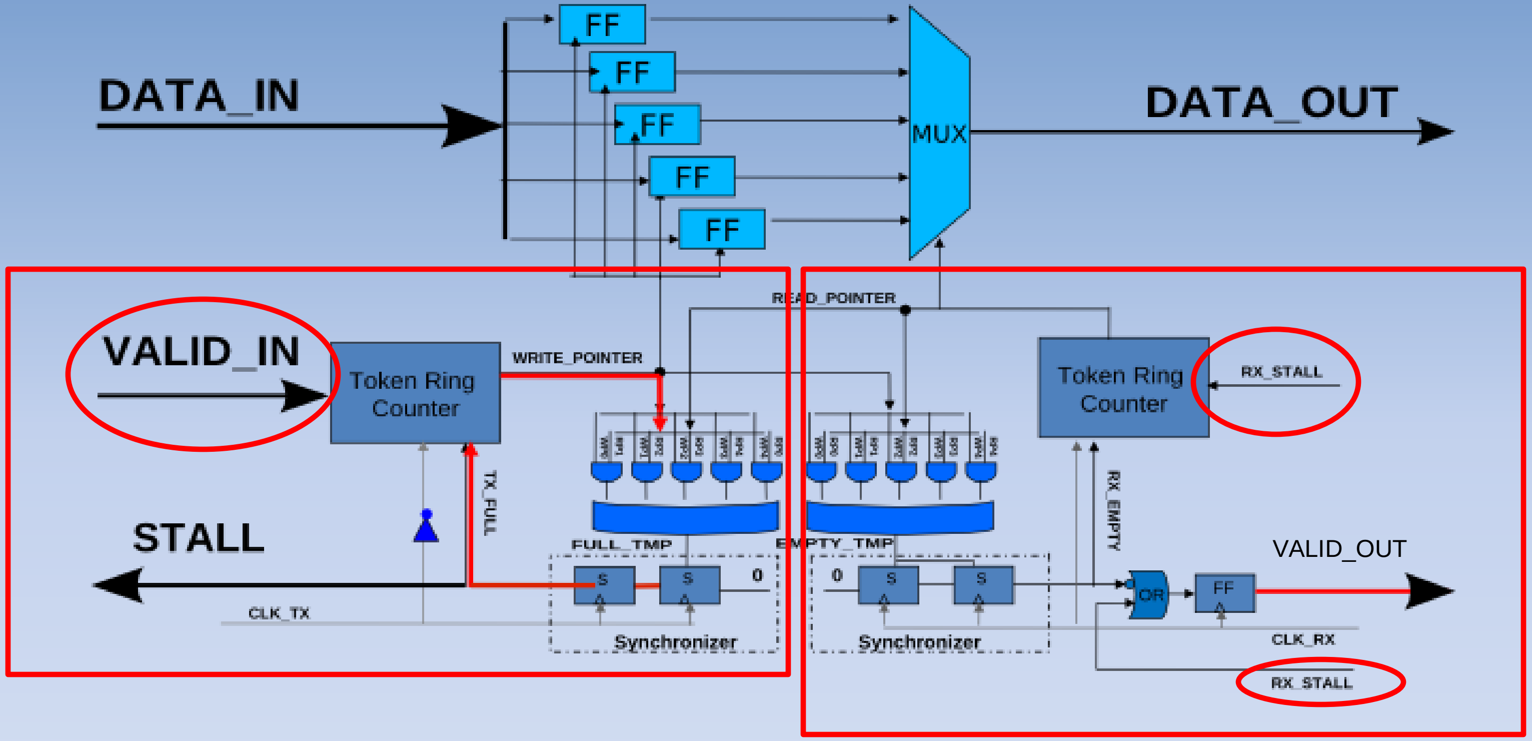
Tightly Coupled Skew Tolerance



- Hold Time is stable and it has a solid margin
- Tarb+Txbar lower the Setup Time curve starting point
- Setup Time becomes even more critical for high negative skew
- Tightly coupled synch cannot work beyond -95% skew!

Tightly coupled dual-clock FIFO synchronizer with the switch architecture

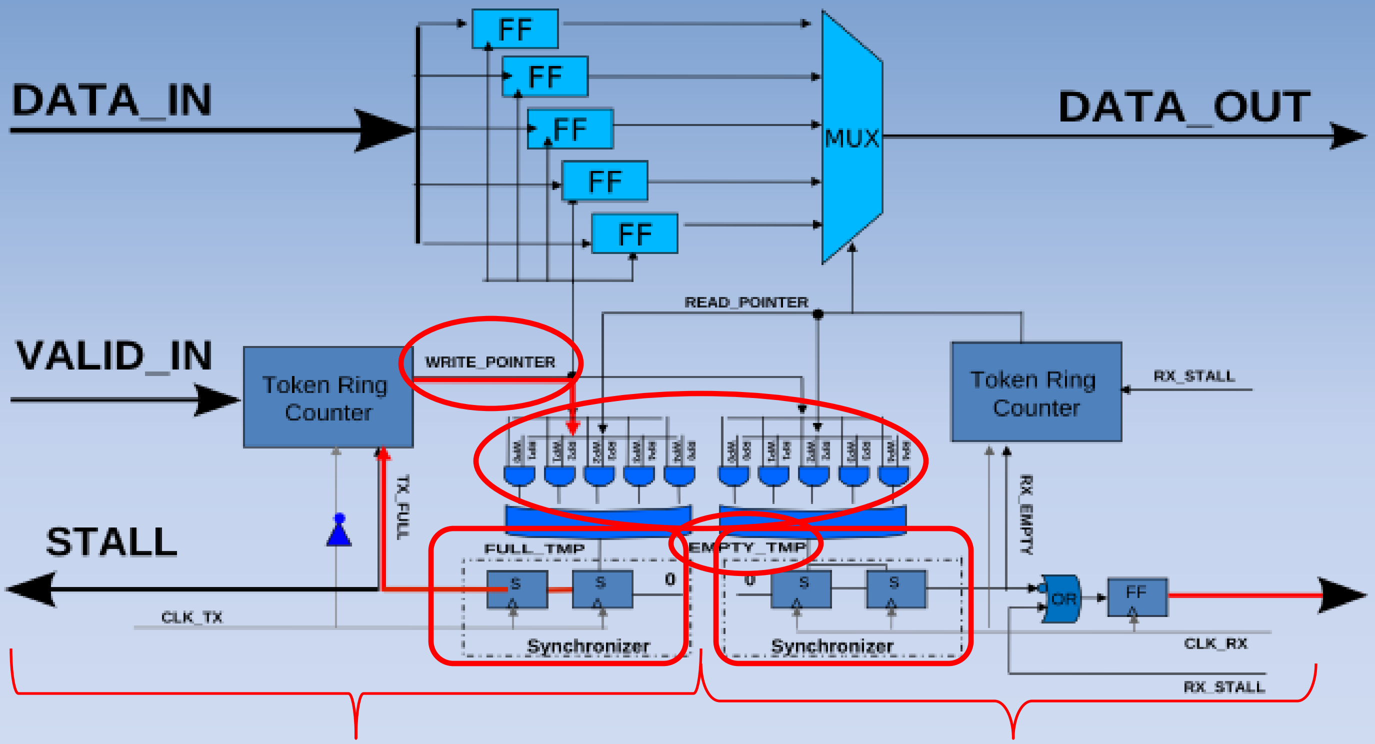
Dual-Clock FIFO Architecture



- data is enqueued when is valid and the buffer is not full and it is dequeued in presence of a go-signal (no stall) and the buffer is not empty

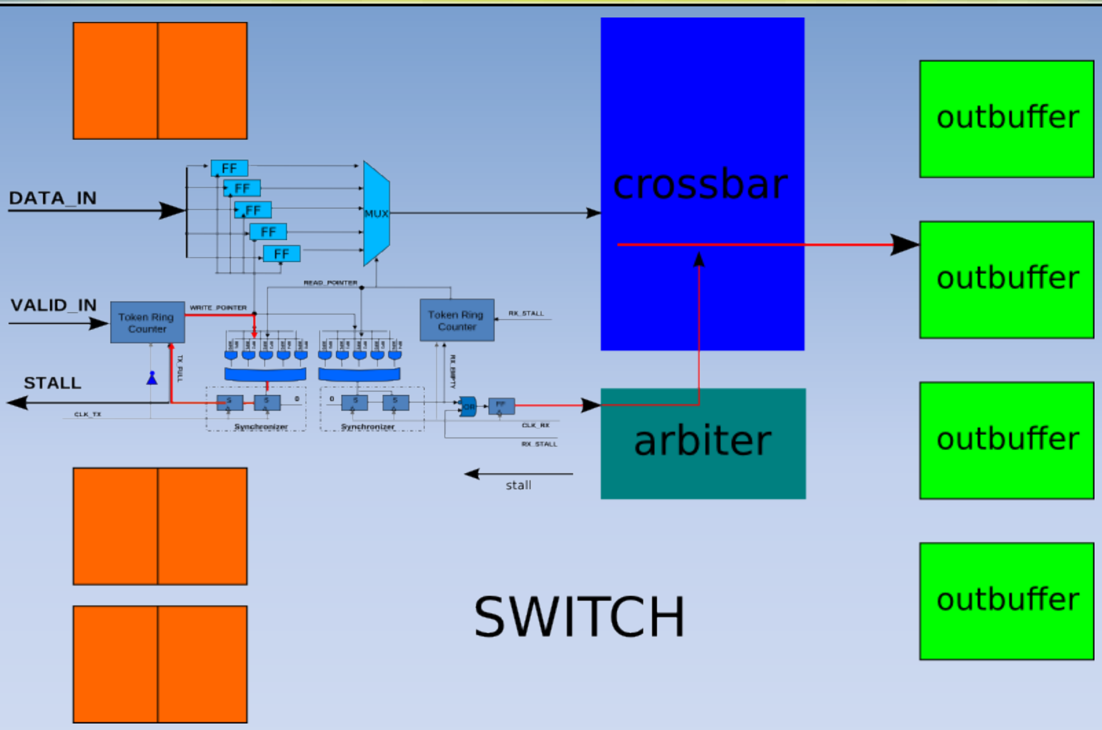
- clear separation between sender and receiver interfaces: token ring counters generate write and read pointer indicating where the operation occurs in the buffer

Dual-Clock FIFO Architecture



- full and empty detectors catch the status of the FIFO buffer by performing an asynchronous comparison between write and read signals
- Assertion of *empty_tmp (full_tmp)* signal is synch with the RX-domain (TX-domain)
- Deassertion of empty_tmp (full_tmp) happens when the write (read) pointer increased
- The ultimate consequence is that *empty_tmp* and *full_tmp* need to be synchronized by means of bruce force synchronizers

Tight Integration in the Switch



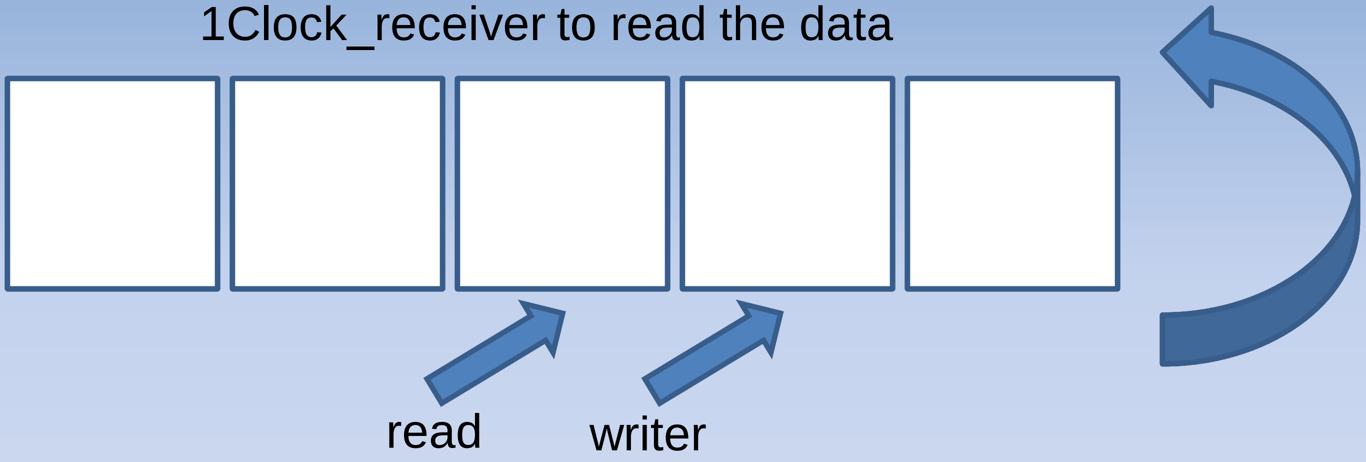
- Seamless integration as for the mesochronous synchronizer
- xpipesLite is natively output buffered (2in – 6out) but nothing prevents to resize the output buffer to 2 and have an integrated FIFO of 6 slots => no buffering overhead
- Performance evaluation at system-level is our ongoing work

LATENCY ANALYSIS

Minimum latency: ΔTrx + 1Clock_receiver

ΔTrx to open the mux window

1Clock_receiver to read the data



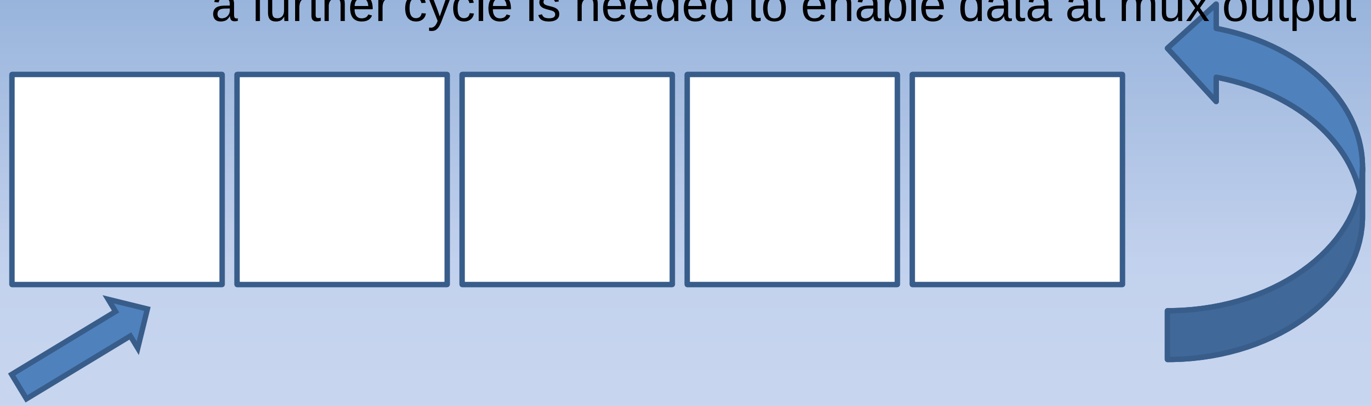
- Latency of the Dual-clock FIFO depends on the relation between sender and receiver clocks: $\Delta T_{rx} + \Omega$
- $0 < \Delta T_{rx} < 1$ is the skew between clk_sender and clk_receiver
- Ω is is the number of clock cycles required by the read pointer to reach the location pointed by the writer

LATENCY ANALYSIS

Empty deassertion: ΔTrx + 2Clock_receiver

ΔTrx + 1Clock_receiver to clear the emptiness

a further cycle is needed to enable data at mux output

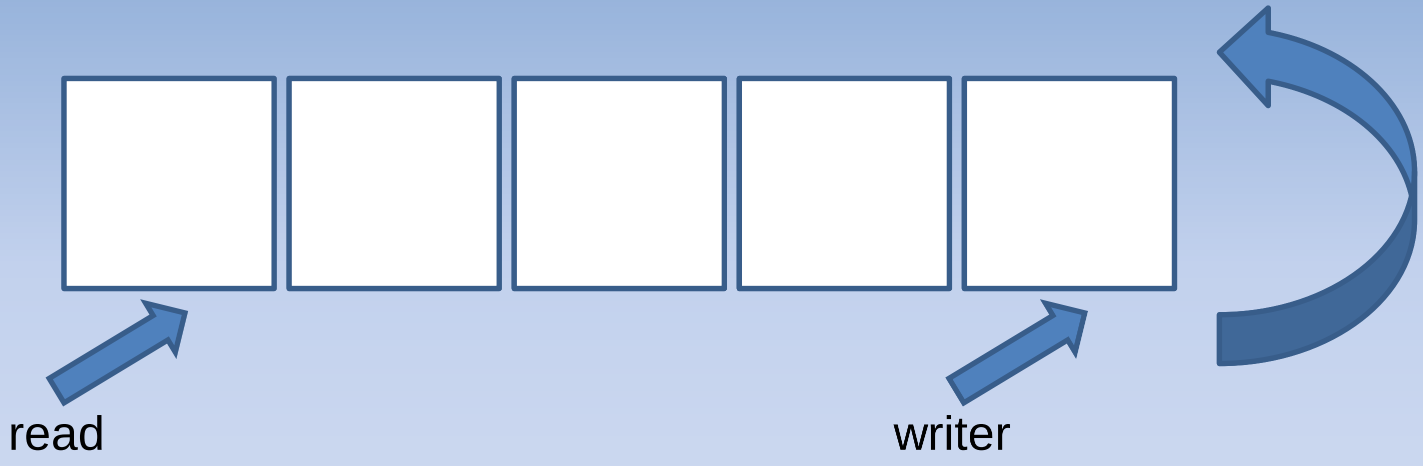


writer

- Latency of the Dual-clock FIFO depends on the relation between sender and receiver clocks: $\Delta T_{rx} + \Omega$
- $0 < \Delta T_{rx} < 1$ is the skew between clk_sender and clk_receiver
- Ω is is the number of clock cycles required by the read pointer to reach the location pointed by the writer

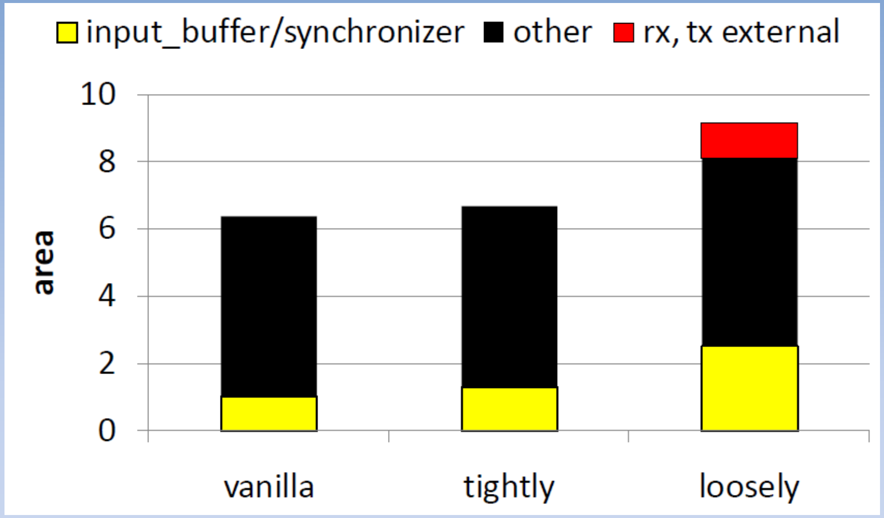
LATENCY ANALYSIS

Maximum latency: ΔTrx + Clock_receiver * (BufferDepth – 2)

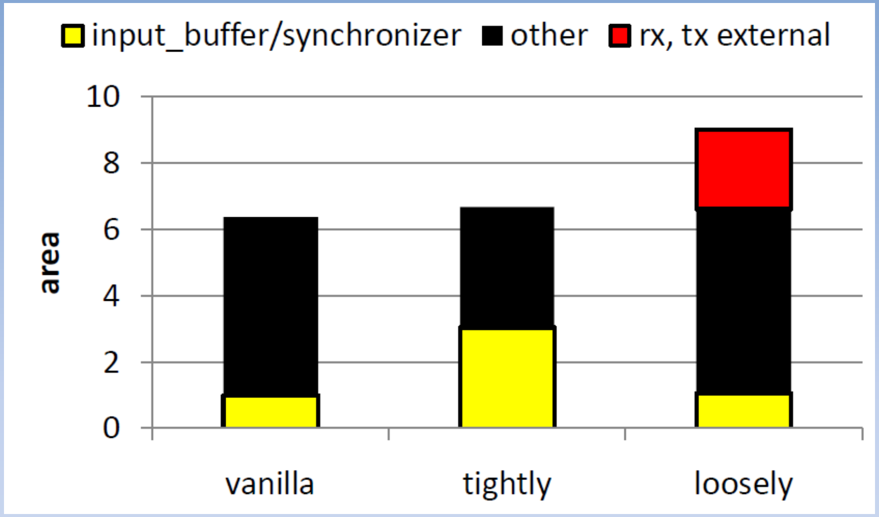


- Latency of the Dual-clock FIFO depends on the relation between sender and receiver clocks: $\Delta T_{rx} + \Omega$
- $0 < \Delta T_{rx} < 1$ is the skew between clk_sender and clk_receiver
- Ω is is the number of clock cycles required by the read pointer to reach the location pointed by the writer

AREA OVERHEAD



Mesochronous synchronizer

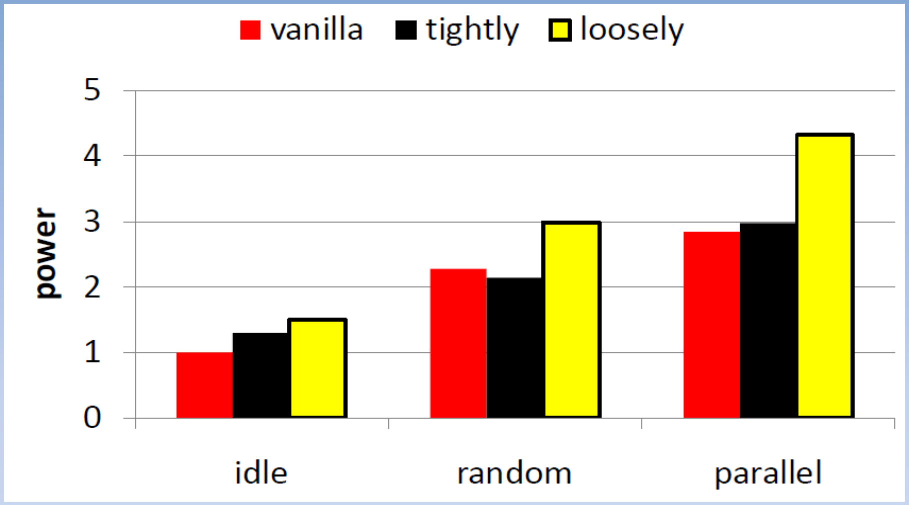


Dual-clock FIFO interface

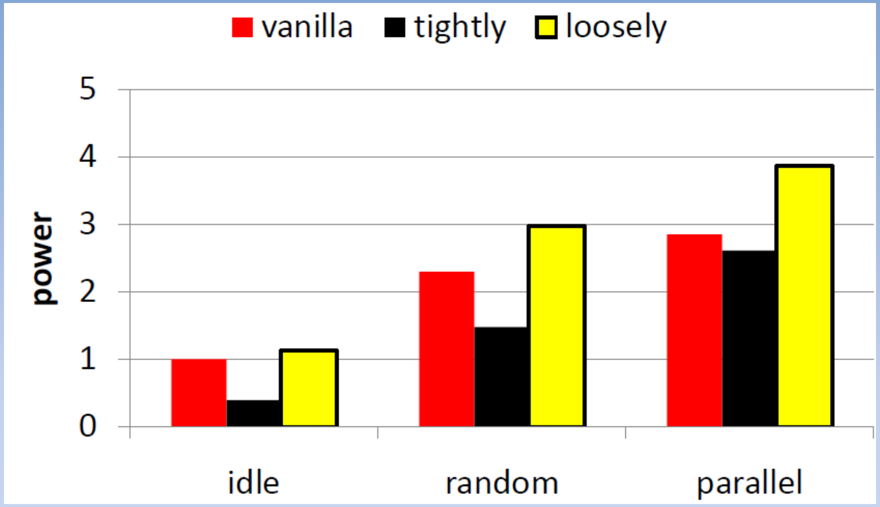
- Breakdown of total switch area
- 65nm UMC technology library, target frequency 1GHz
- Both tightly coupled architectures have a comparable area footprint with their respective vanilla switches

The loosely coupled solution requires up to 43% more area with respect to the vanilla switch!

POWER ANALYSIS



Mesochronous synchronizer



Dual-clock FIFO interface

- Post-layout simulations carried out at 800MHz
- Area overhead comes with a power penalty!
- Tightly coupled mesochronous power figures reflect those of vanilla switch (as for the area)
- Tightly coupled dual-clock FIFO inherently clock gates the input buffer when data is not valid (not available in vanilla switch)

Summing up

A loosely coupled synchronizer in front of the switch fabric

- implies large buffering in the switch input
- fully exposes its area and power overhead

We advocate for a tightly coupled synchronizer with the switch architecture.

- multi-purpose input buffer in charge of synchronization, buffering and flow control.
- major savings thanks to sharing of expensive buffering
- marginal area/power/timing overhead with respect to a fully synchronous switch

Thank you!