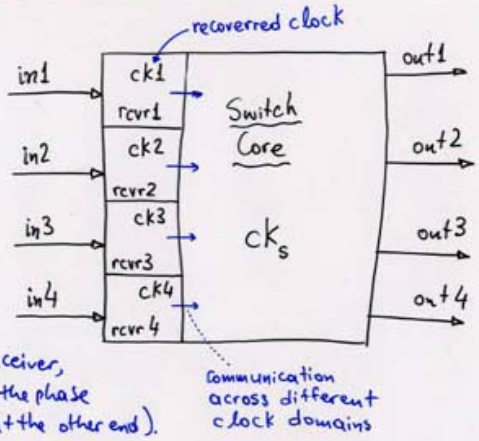
## Communicating Across Clock Domains

Why do we need it?

Even if we choose $ck_s$ to be e.g. $ck_s \equiv ck1$, we will still have $ck_s \neq ck2$, $ck_s \neq ck3$, $ck_s \neq ck4$.

(For long links, the phase of the recoverred clock, at the receiver, varies widely relative to the phase of the transmitter clock, at the other end).

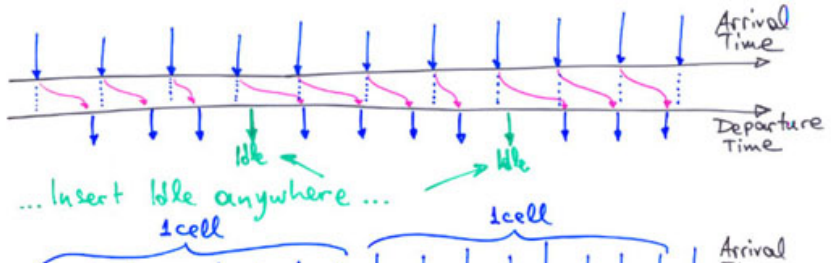



(Switch outputs can often be synchronized to the switch core clock $ck_s$; this reduces outgoing latency).

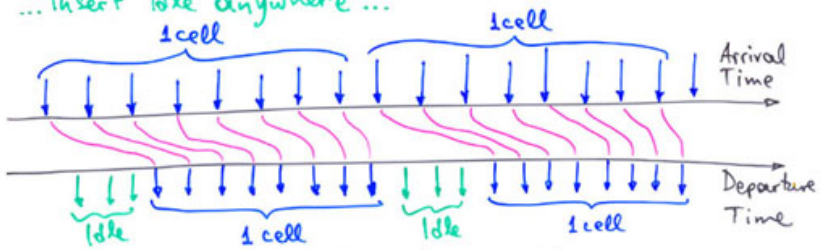


## Interfacing Slightly Different Clock Frequencies:

① Slow-to-Fast:
⇒ Insert Idle Symbols




... Insert Idle anywhere ...

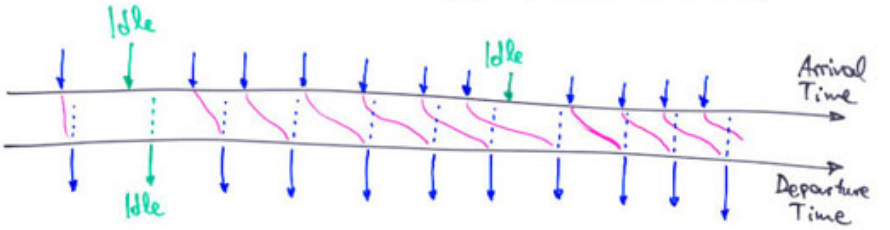



... or Insert Idle only between cells.

## Interfacing Slightly Different Clock Frequencies:

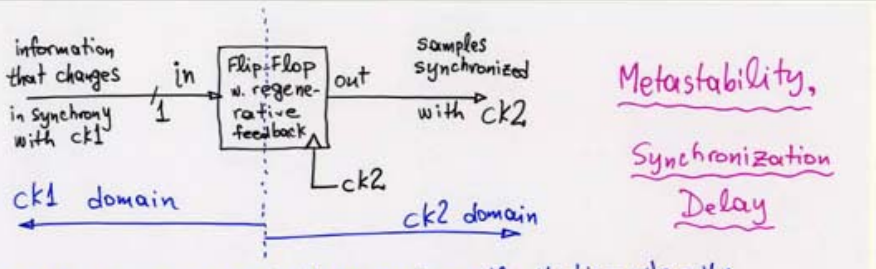
② Fast-to-Slow:

(α) ⇒ Remove Idle Symbols




and/or (β) ⇒ notify the transmitter, by feedback ("backpressure") to slow-down or wait, i.e. to insert more idle's (so that they can be removed) ...






## Metastability, Synchronization Delay

Q: What happens if "in" changes at exactly the time when the active edge of ck2 comes?

A: "out" may get into "metastable" state (intermediate voltage) and stay in it for an undetermined duration $t_{meta}$.

$$\text{Probability}\left(t_{meta} \geq \delta\right) \sim e^{-\frac{\delta}{\text{flip-flop feedback loop delay}}}$$

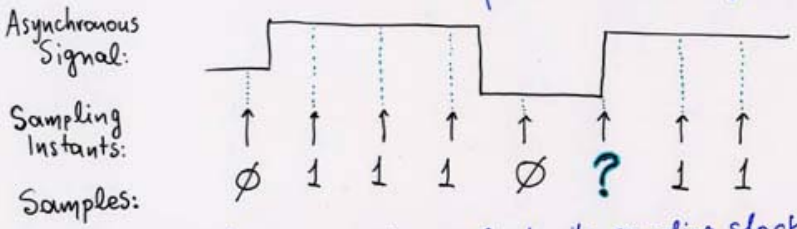
P(failure) ... also ~ frequency of edges in "in" × frequency of ck2

Solution: do NOT use "out" for a time δ on the order of tens or hundreds of FF feedback loop delays (synchronization failure frequency < 1 in e.g. 100 years...)

## Was the Signal Sampled before or after its Change?

### (A) Serial Signal Sampling:

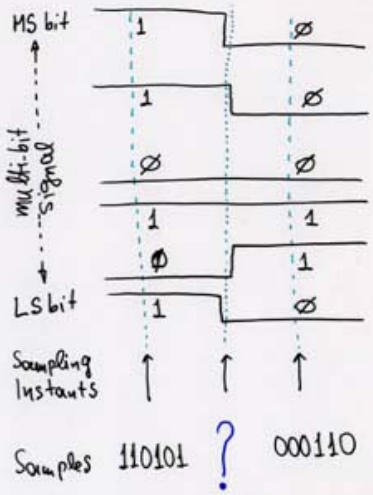
(Need $f_{sampling} > f_{signal\ change}$ in order to "see" all signal changes)




Since the signal changes asynchronously to the sampling clock, the sampling point "?" could have been a little before or a little after the signal changes, yielding either Ø or 1. Hence, it does not matter if the FF metastability eventually yields Ø or 1 - all that matters is that it becomes a valid binary value, that all of its receivers interpret in the same way.



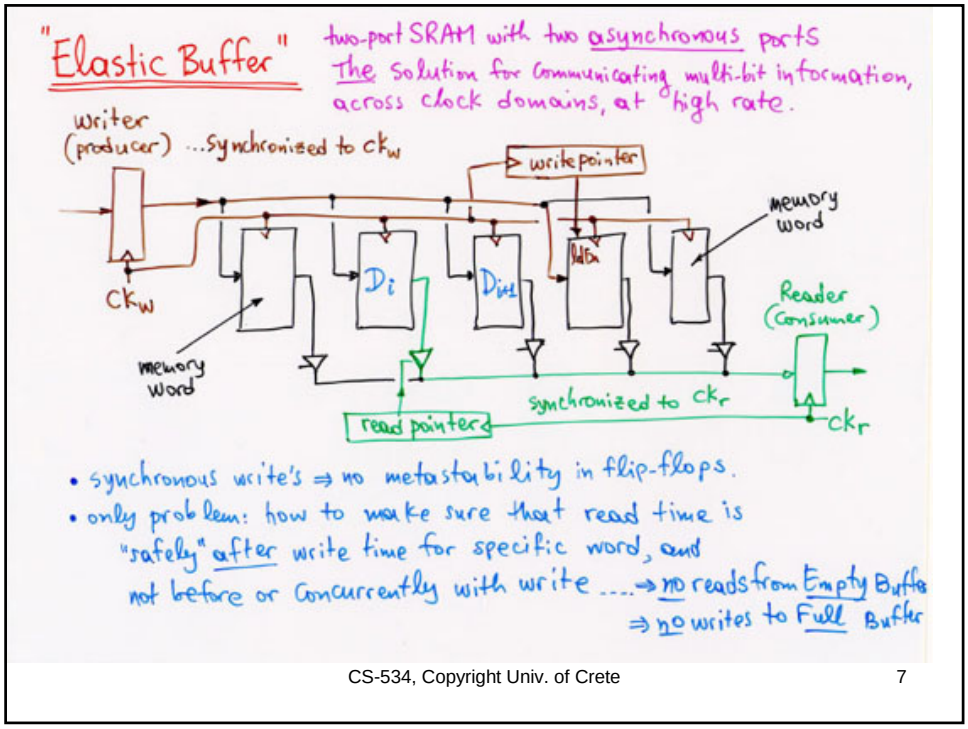
### (B) Parallel Signal Sampling (Impossible w. asynchronous clock)

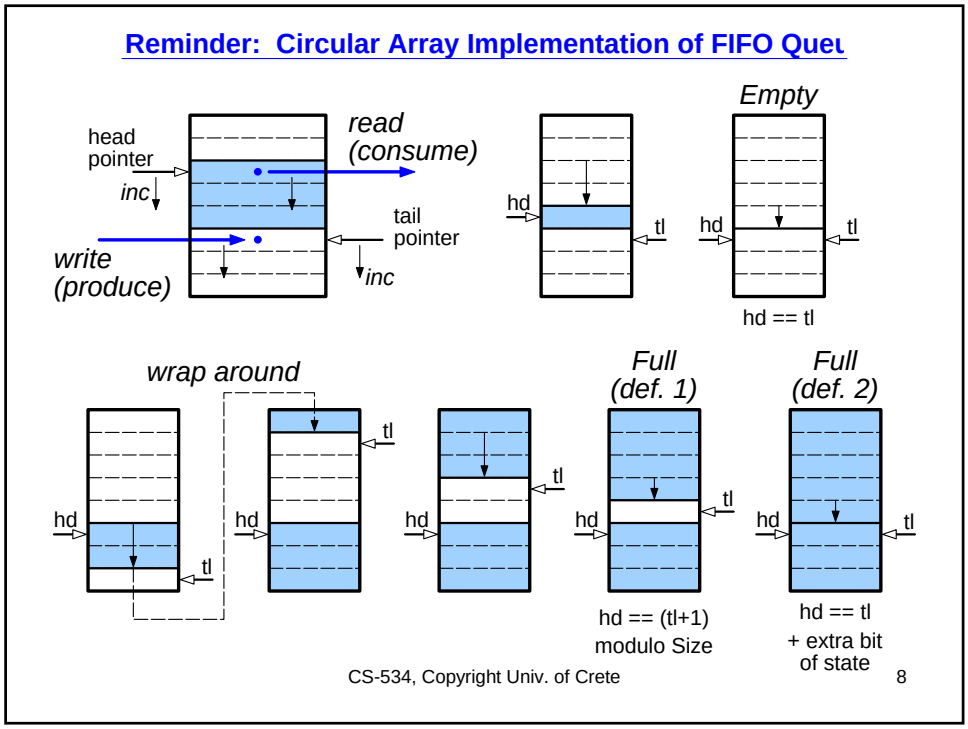


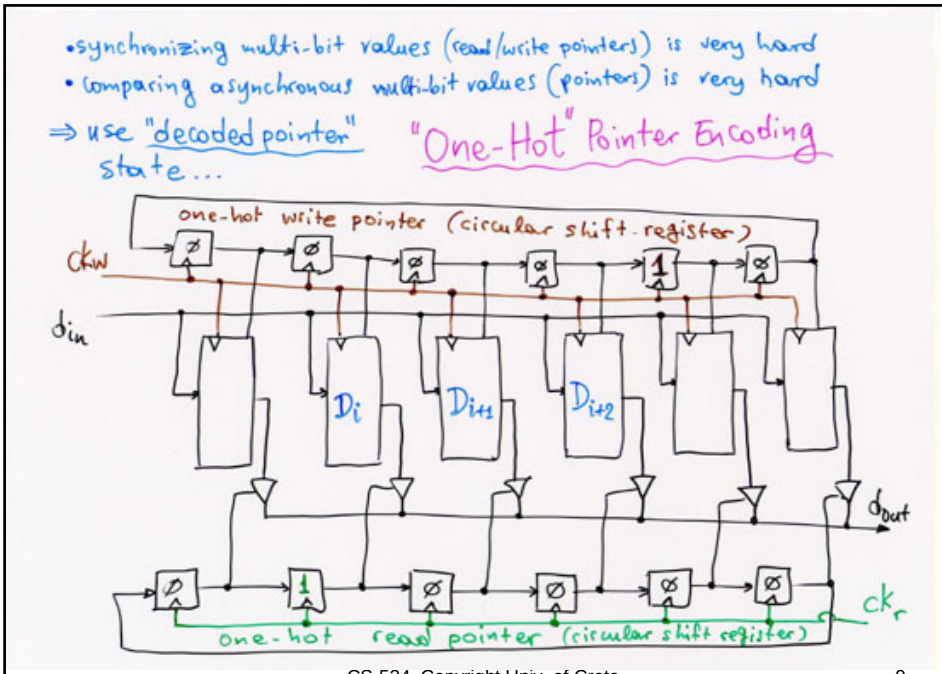

For multibit signals, asynchronous sampling is basically useless: there is no way to tell if the sampled word is a valid one (old or new) or an invalid word made by a random selection of bits from these two words.

(Only possible solution: $f_{sampling} > 2 \times f_{signal\ change}$, then expect to see each word twice or more...)

010100 ← not a valid word ≠ previous word ≠ next word

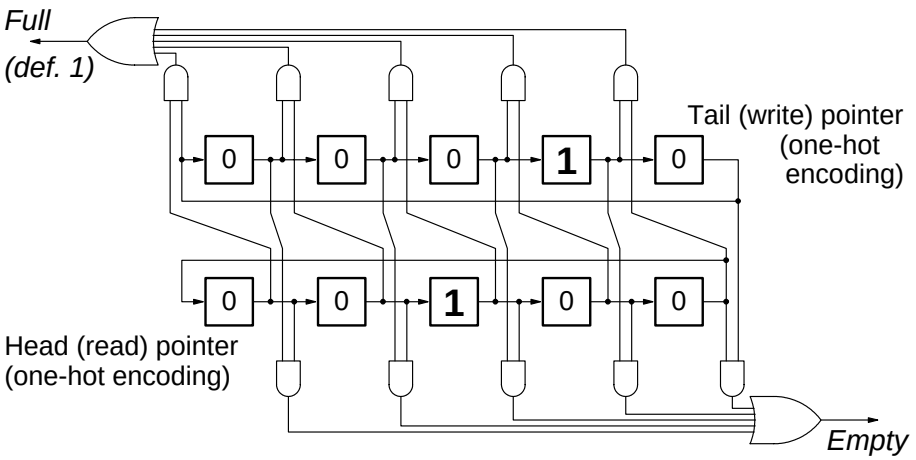

"Elastic Buffer"
two-port SRAM with two asynchronous ports
The solution for communicating multi-bit information, across clock domains, at high rate.
Writer (producer)
...synchronized to $ck_w$
write pointer
memory word
$ck_w$
$D_i$
$D_{i+1}$
Reader (consumer)
memory word
synchronized to $ck_r$
read pointer
$ck_r$
• synchronous write's ⇒ no metastability in flip-flops.
• only problem: how to make sure that read time is "safely" after write time for specific word, and not before or concurrently with write ....⇒ no reads from Empty Buffer
⇒ no writes to Full Buffer
CS-534, Copyright Univ. of Crete
7


Reminder: Circular Array Implementation of FIFO Queue
head pointer
inc
read (consume)
tail pointer
inc
write (produce)
Empty
hd
tl
hd == tl
wrap around
Full (def. 1)
Full (def. 2)
hd == (tl+1) modulo Size
hd == tl + extra bit of state
CS-534, Copyright Univ. of Crete
8

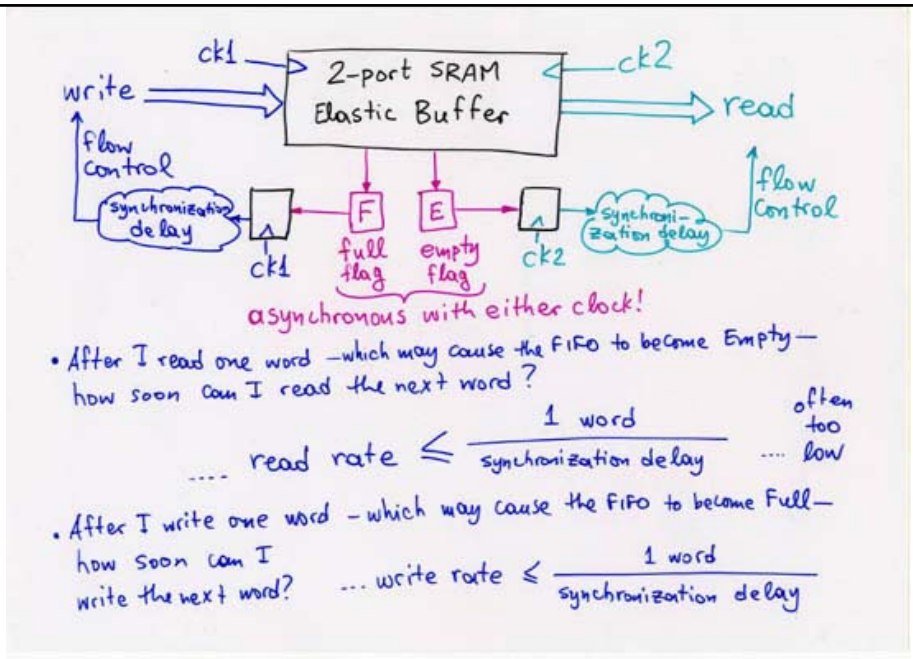



CS-534, Copyright Univ. of Crete 9

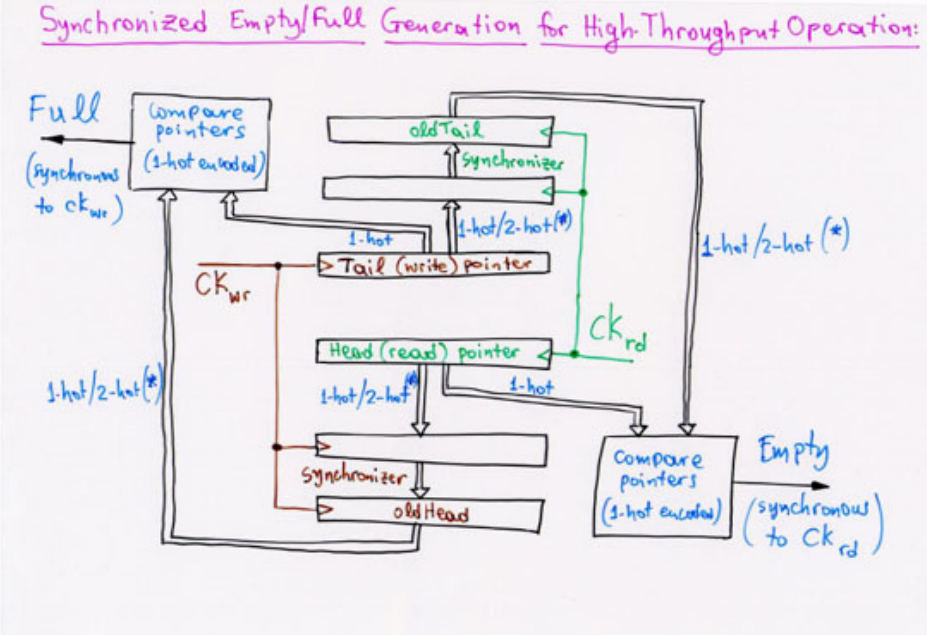
## Empty/Full FIFO Detection using One-Hot Pointer Encodin




CS-534, Copyright Univ. of Crete 10


ck1
2-port SRAM
Elastic Buffer
ck2
write
read
flow control
synchronization delay
F
E
full flag
empty flag
ck1
ck2
synchroni-zation delay
flow control
asynchronous with either clock!
• After I read one word –which may cause the FIFO to become Empty–
how soon can I read the next word?
read rate ≤ 1 word / synchronization delay
often too low
• After I write one word – which may cause the FIFO to become Full–
how soon can I write the next word?
... write rate ≤ 1 word / synchronization delay




Synchronized Empty/Full Generation for High-Throughput Operation:
Full
(synchronous to ck_wr)
Compare pointers (1-hot encoded)
oldTail
Synchronizer
1-hot
1-hot/2-hot(*)
1-hot/2-hot (*)
Tail (write) pointer
CK_wr
ck_rd
Head (read) pointer
1-hot/2-hot(*)
1-hot
1-hot/2-hot(*)
Synchronizer
oldHead
Compare pointers (1-hot encoded)
Empty
(synchronous to Ck_rd)

## Timing & Synchronicity of Full & Empty Flags

- **Full** flag – Synchronous to $ck_{wr}$
  - asserted as soon as a write operation fills the FIFO up (def.1 "full")
  - negated after a word is read from the FIFO and the synchronization delay elapses
- **Empty** flag – Synchronous to $ck_{rd}$
  - asserted as soon as a read operation empties the FIFO
  - negated after a word is written into the FIFO and the synchronization delay elapses

- *Reference* on Synchronization and Elastic Buffers: *W. Dally, J. Poulton: "Digital Systems Engineering"*, Cambridge University Press, 1998, ISBN 0-521-59292-5 (sections 10.2 and 10.3 –especially 10.3.4.2).



## Sampling 1-hot pointers for synchronization purposes: 1-hot/2-hot versions

- A 1-hot encoded pointer is a multi-bit value.
- When sampling any such value with an asynchronous clock for synchronization purposes, there is always the possibility that *some bits are sampled* "before" and some "after" they transition.
- This may result in the sampled pointer containing 2 bits ON, or 1 bit ON, or no bit ON (2-hot, or 1-hot, or 0-hot).
- 2-hot is "OK": conservative!
- 1-hot is normal.
- 0-hot is bad: empty/full is not asserted even when the FIFO is in one of these states ➔ we have to ensure that 0-hot never happens!
- ⇒ Use a 1-hot/2-hot version of the pointer for synchronization purposes: make sure that the new "hot" bit is turned ON safely before the old "hot" bit is turned OFF (e.g. use appropriate OR function of master & slave flip-flops).