2. Link and Memory
Architectures and Technologies

2.1 Links, Thruput/Buffering, Multi-Access Ovrhd
2.2 Memories: On-chip / Off-chip SRAM, DRAM

Manolis Katevenis
CS-534 – Univ. of Crete and FORTH, Greece
http://archvlsi.ics.forth.gr/~kateveni/534
2.A Elastic Buffers for Cross-Clock Communication

Table of Contents:

• Oscillator frequency deviation
  – have *idle* symbols in the flow, for removal/insertion at interfaces

• Single-bit communication
  – either 0 or 1 is OK, but *not* intermediate – *not metastable*

• Multi-bit word communication, at almost 1 word/cycle rate
  – *cannot* sample asynchronously to the source clock
  – need 2-port FIFO
  – read-pointer to write-pointer comparison problem
  – synchronizing the empty/full flags leads to low access rate
  – 1-hot (or 2-hot) encoding; compare to an old, conservative, synchronized version of the other-domain’s pointer
Communicating Across Clock Domains

Why do we need it?

Even if we choose $c_k$ to be e.g. $c_k = c_k^1$, we will still have $c_k 
eq c_k^2$, $c_k 
eq c_k^3$, $c_k 
eq c_k^4$.

(For long links, the phase of the recovered 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 $c_k^*$; this reduces outgoing latency.)
Interfacing Slightly Different Clock Frequencies:

1. Slow-to-Fast:
   - Insert Idle Symbols

   ... Insert Idle anywhere ...

   1 cell

   ... 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) ...
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 $\sim$ frequency of edges in "in" x frequency of ck2

Solution: do NOT use "out" for a time $\delta$ 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: \( f_{sampling} > f_{signal\ change} \)

in order to "see" all signal changes.

Asynchronous Signal:

Sampling Instants:

Samples:

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 \( \phi \) or 1. Hence, it does not matter if the FF metastability eventually yields \( \phi \) 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: \( f_{\text{sampling}} > 2 \times f_{\text{signal change}} \), then expect to see each word twice or more...)

Samples 110101? 000110

010100 \( \not\equiv \) not a valid word \( \neq \) previous word
010100 \( \not\equiv \) next word
"Elastic Buffer" two-port SRAM with two asynchronous ports

The solution for communicating multi-bit information, across clock domains, at high rate.

- 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
Reminder: Circular Array Implementation of FIFO Queue

- **Head Pointer**: \( \text{head} \)
- **Tail Pointer**: \( \text{tail} \)
- **Write (Produce)**: \( \text{write} \)
- **Read (Consume)**: \( \text{read} \)

**Empty**: \( \text{hd} = \text{tl} \)

**Full (def. 1)**: \( \text{hd} = (\text{tl} + 1) \) modulo Size

**Full (def. 2)**: \( \text{hd} = \text{tl} \) + extra bit of state

- **Wrap Around**
- Synchronizing multi-bit values (read/write pointers) is very hard
- Comparing asynchronous multi-bit values (pointers) is very hard

\[ \Rightarrow \text{use "decoded pointer" state...} \]

"One-Hot" Pointer Encoding

```
+-----------------+
| one-hot write pointer (circular shift register) |
+-----------------+
```

```
+-----------------+
| one-hot read pointer (circular shift register) |
+-----------------+
```
Empty/Full FIFO Detection using One-Hot Pointer Encoding

Head (read) pointer (one-hot encoding)

Tail (write) pointer (one-hot encoding)

Full (def. 1)
2-port SRAM Elastic Buffer

- 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?
  \[
  \text{read rate} \leq \frac{1 \text{ word}}{\text{synchronization delay}} \quad \text{often too low}
  \]
- After I write one word — which may cause the FIFO to become Full —
  how soon can I write the next word?
  \[
  \text{write rate} \leq \frac{1 \text{ word}}{\text{synchronization delay}}
  \]
Synchronized Empty/Full Generation for High-Throughput Operation:

Full
(synchronous to \( CK_{wr} \))

\( CK_{wr} \)

\( 1\text{-hot} / 2\text{-hot} (\ast) \)

\( 1\text{-hot} / 2\text{-hot} (\ast) \)

\( \text{Head (read) pointer} \)

\( \text{Tail (write) pointer} \)

\( \text{Compare pointers (1-hot encoded)} \)

\( \text{Old Head} \)

\( \text{Old Tail} \)

\( \text{Synchronizer} \)

\( \text{Compare pointers (1-hot encoded)} \)

\( \text{Empty} \)

(synchronous to \( CK_{rd} \))
Timing & Synchronicity of Full & Empty Flags

- **Full** flag – Synchronous to $c_{\text{wr}}$
  - asserted as soon as a write op. 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 $c_{\text{rd}}$
  - asserted as soon as a read operation empties the FIFO
  - negated after a word is written into the FIFO and the synchronization delay elapses

**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).