Bus and Protocol

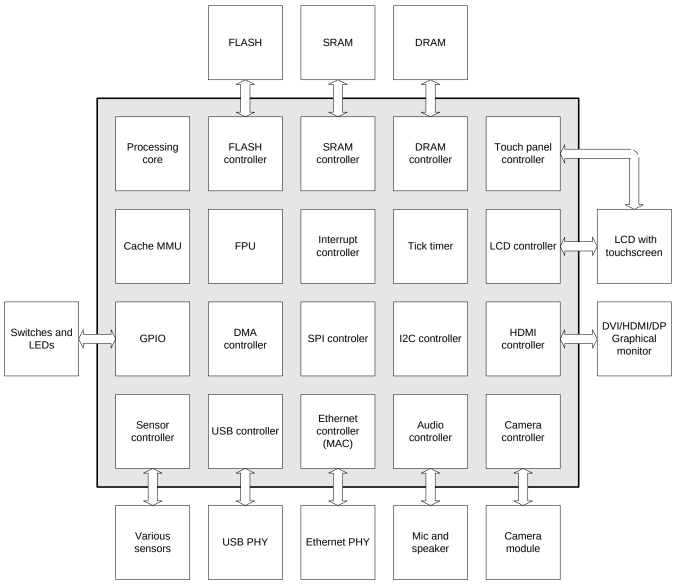
2013 – 2017 - 2018

Ando Ki, Ph.D.
(adki@future-ds.com)

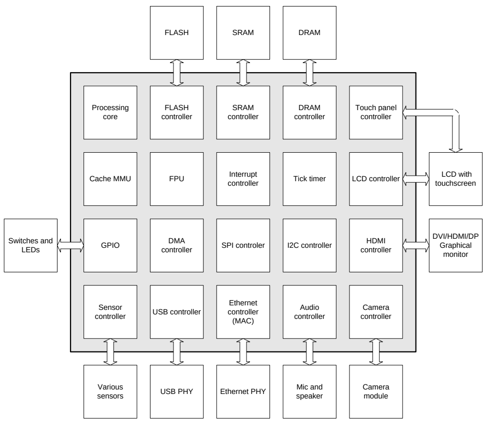
Agenda

- What is BUS
- How to send and receive data
- How to select slave
- How to select master
- Transfer types
- Burst transfers
- Pipelined and split transfers
- Data ordering
- Justified or non-justified
- Partial/narrow access
- Alignment
- Atomic
- Clock frequency and phase
- Synchronous or asynchronous
- Timing diagram conventions

How to integrate various components



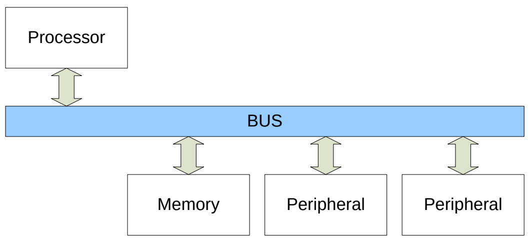
How to integrate various components



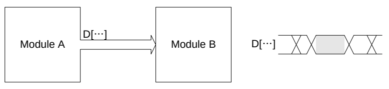
- What to be considered
  - How to support different access patterns
    - e.g., processing core accesses instructions and data
    - e.g., DMA moves a bunch of data as fast as possible
    - e.g., video-out controller reads image frames 30~60 times a second
    - e.g., video-in controller writes image frames 15~60 times a second
    - e.g., audio controller reads/writes PWM data stream
  - How to guarantee error-free data movement
  - How to provide fast response time and sufficient bandwidth
  - Most of all, all components should be accessed via processor
- Need something special, which is called bus, OCB (on-chip-bus), network, OCN (on-chip-network), switch, ...

What is bus

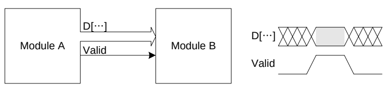
- **<u>Bus</u>** means a set of common lines that electrically connects various blocks in order to transfer data among them.
  - (버스란 여러 블록들이 상호 데이터를 전송하기 위해 이들 블록들을 전기적으로 연결한 공유 신호선)
  - <u>Protocol</u> (communication protocol) is a set of rules to accomplish data transfer among blocks along the bus.
    - (프로토콜(통신규약)은 버스를 통해 데이터를 전송하기 위한 규칙)



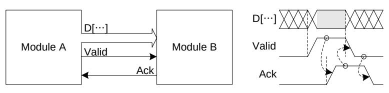
How to send and receive data



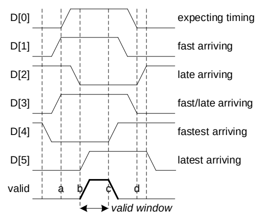
How does Module-B know the data is valid?

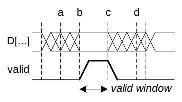


How does Module-A know Module-B got the data?



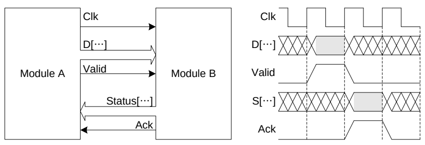
How does Module-B know the data is stable enough to get? Note that data ‘D[...]’ consists of multiple lines and there is skew among signals.



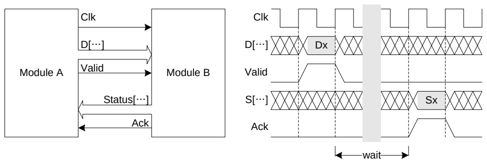


Skew is the time difference among propagation delays of any outputs of the same device.
Valid window will be very narrow.

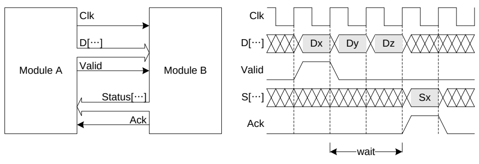
How to send and receive data



What if Module B cannot respond at the given time?

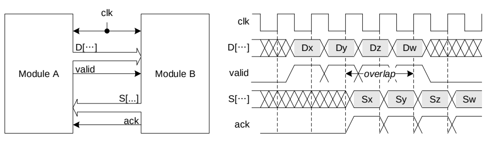


How to use the bus more effectively while waiting the ack?

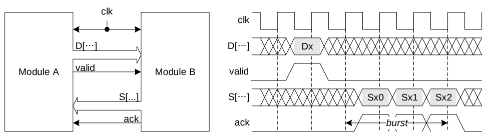


A series of transfers consisting of related or unrelated can help utilization efficiency of the bus.
How to manage those transfers?

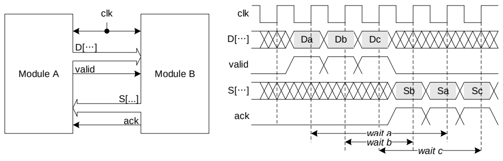
How to send and receive data



Increase utilization by overlapping



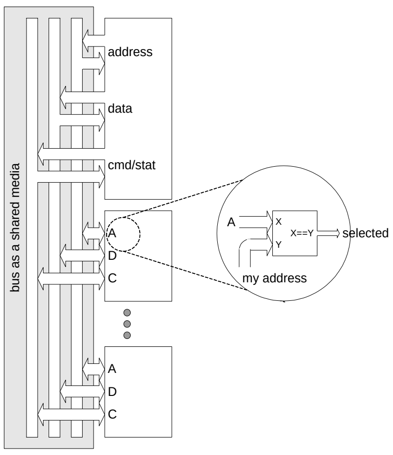
Increase utilization by burst operation

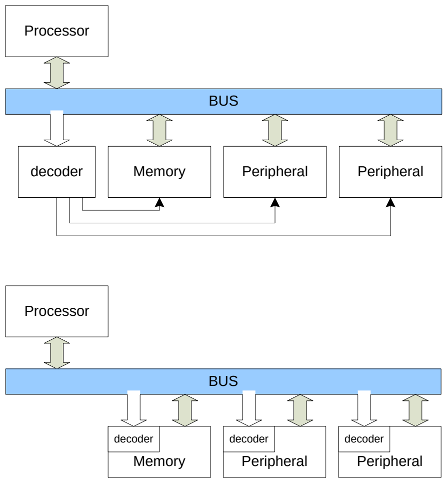


Increase utilization using multiple outstanding and/ or out-of-order completion

How to select one of many slaves

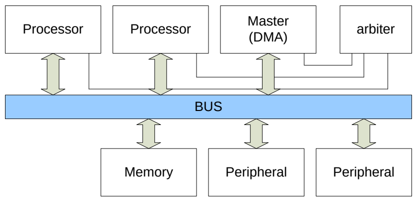
- Decoder (or selector)
  - Select one by decoding address
- Location of decoder
  - Centralized
  - Distributed

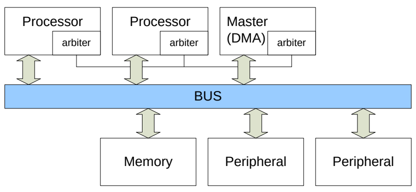




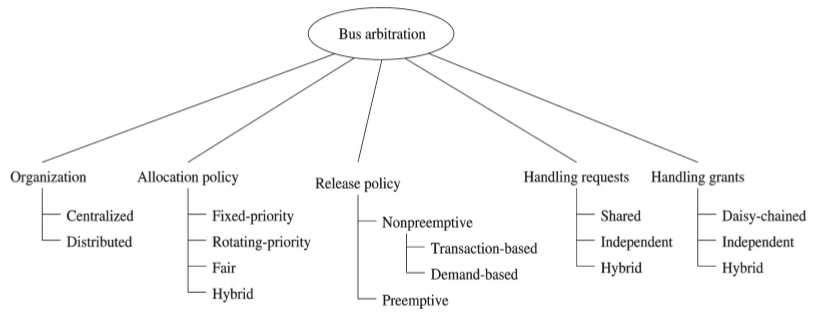
How to select one of many masters

- Need selection/election before using bus, since bus is shared resource
- Arbitration
  - Defines which module gains access to the bus
- Location of arbiter
  - Centralized
  - Distributed





Bus arbitration

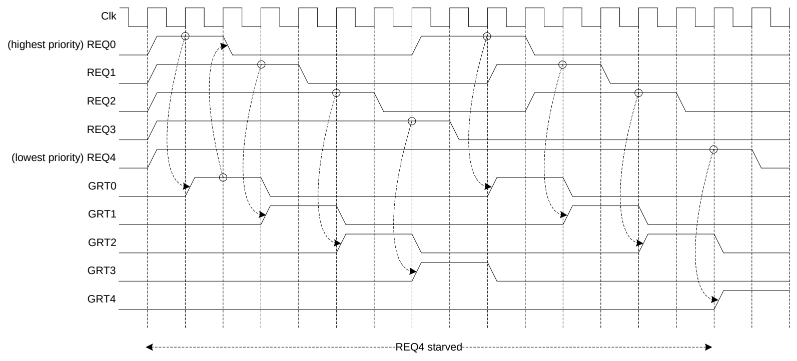


S. Dandamudi, "Fundamentals of Computer Organization and Design," Springer, 2003.

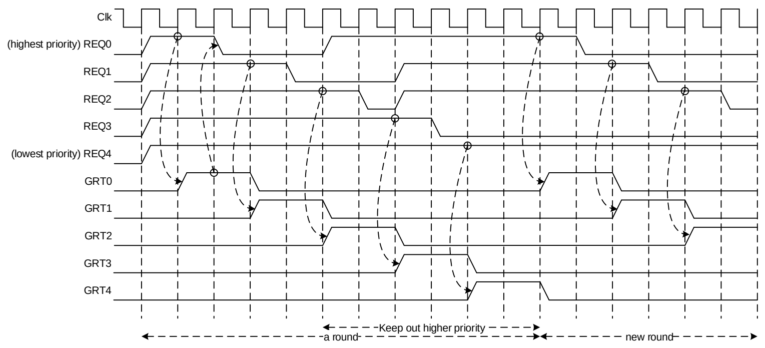
Algorithms and issues of arbitration

- Algorithm of arbitration
  - Fixed priority based
  - Round-robin
- Issues of arbitration
  - Fairness
  - Starvation
    - a situation where a block is unable to gain bus access without any progress (오랜 기간 아무런 진전 없이 기다리는 경우)
  - Live-lock
    - a situation where a block is unable to make progress although doing something busily (뭔가 열심으로 일을 하는데, 지전은 없는 경우 – Cache에서 계속 invalidation 되거나, …), it will eventually be resolved.
  - Deadlock
    - a situation where a block wait for some condition that will not resolved forever.

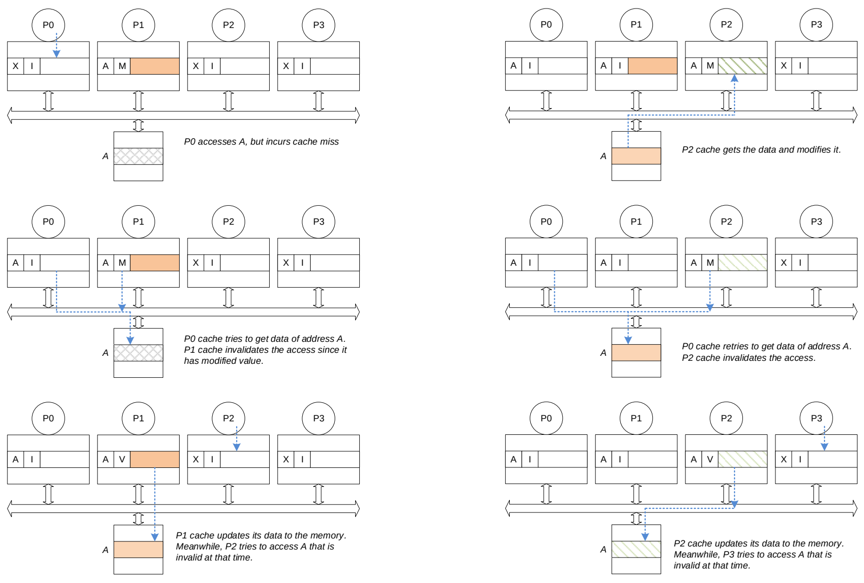
Starvation example



Fairness example



Live-lock example



P0 accesses A, but incurs cache miss

P2 cache gets the data and modifies it.

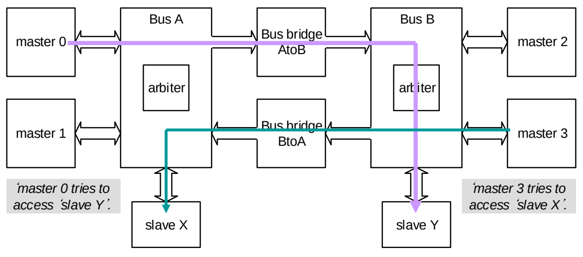
P0 cache tries to get data of address A. P1 cache invalidates the access since it has modified value.

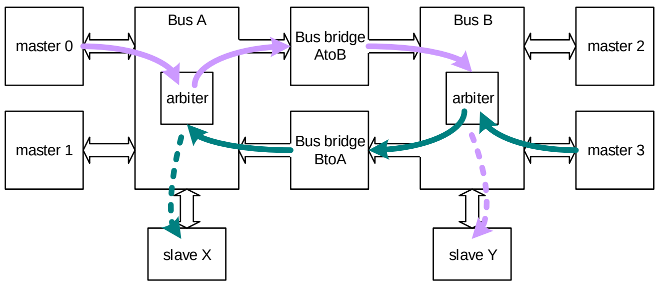
P0 cache retries to get data of address A. P2 cache invalidates the access.

P1 cache updates its data to the memory. Meanwhile, P2 tries to access A that is invalid at that time.

P2 cache updates its data to the memory. Meanwhile, P3 tries to access A that is invalid at that time.

Dead-lock example





Transfer types

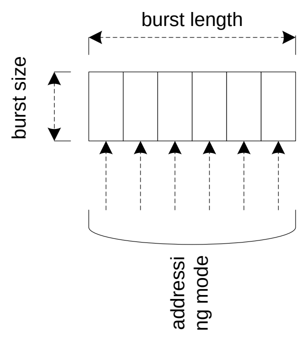
- How to use bus more efficiently
  - Latency minimization
  - Throughput maximization
  - Speed v.s. bandwidth (= capability)

[HiPi-Bus case]	[AMBA AHB case]
NRD: Normal-Read	NONSEQ – SEQ
NWR: Normal-Write	
RFR: Read-For-Read	SINGLE
RFW: Read-For-Write	INCR/INCR4/INCR8/INCR16
LCR: Lock-Read	WRAP4/WRAP8/WRAP16
LCW: Lock-Write	
WRB: Writeback	
INV: Invalidation	

- How to capture access-intentions
  - It can be used to make bus efficient.
  - DMA uses accesses that move a block of data
    - Need to support burst
  - CPU generates two types of accesses and the tip about types will be used by cache
    - instruction access: it never be modified by the CPU
    - data access: it may be altered by the CPU near future
  - CPU with cache needs more types of transfers

Burst transfers (1/3)

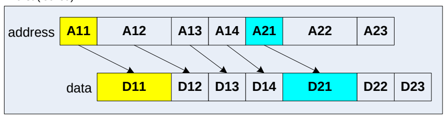
- Burst transfer in the bus is a means of data movement consisting of more than one transfer in order to get a higher throughput.
  - Burst length
    - num of beats in a burst
  - Burst size
    - num of bytes moved in a single beat
  - Addressing mode
    - incremental
    - wrapping
    - fixed
    - stride
  - Other issues
    - partial burst size case, in which burst size is smaller than data bus width



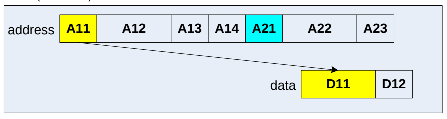
Burst transfers (2/3)

- burst (locked)
  - Address and data are locked together
  - Single pipeline stage
  - If one slave is very slow, all data is held up.
- one address for burst
  - One Address for entire burst

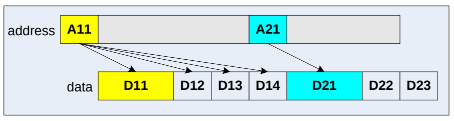
Burst (locked)



Burst (slow slave)



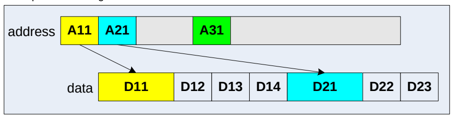
One address for a burst

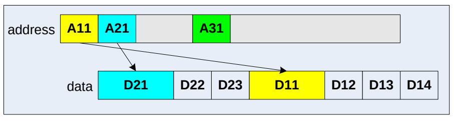


Burst transfers (3/3)

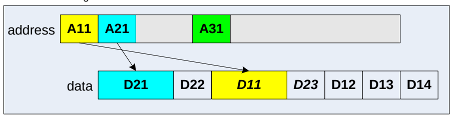
- multiple outstanding bursts
  - One Address for entire burst
  - Allows multiple outstanding addresses
- out-of-order completion
  - Masters can issue multiple ordered addresses
  - Fast slaves may return data ahead of slow slaves
- data interleaving
  - Returned data can be interleaved

Multiple outstanding bursts





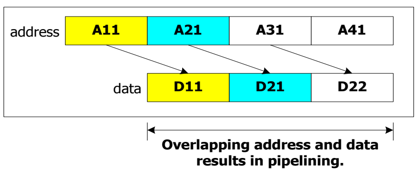
Data interleaving



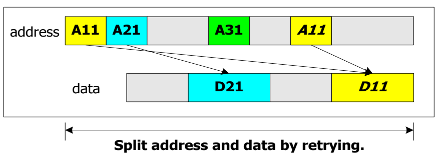
Pipelined and split transfers

- Pipeline bus protocol
  - arbitration, address, data phases can be overlapped

- Split bus protocol
  - Split transfers improve the overall utilization of the bus by separating the operation of the master providing the address to a slave from the operation of the salve responding with the appropriate data.



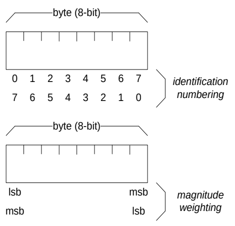
Overlapping address and data results in pipelining.

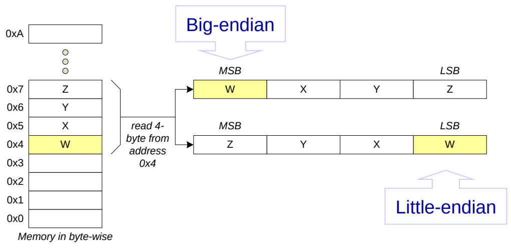


Split address and data by retrying.

Data ordering (1/2)

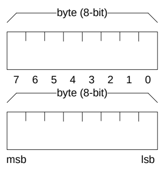
- Bit ordering within a byte
- Byte ordering of a multiple-of-byte

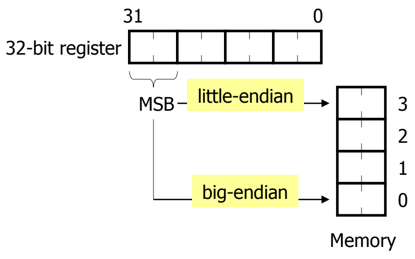




Data ordering (2/2)

- Bit ordering within a byte
  - Usually most significant bit is called 7th bit, but not always true.
- Byte ordering
  - How should bytes within multi-byte word be ordered in memory?
  - Big-endian: Least significant byte has highest address
    - Most significant byte has lowest address
    - Sun SPARC, HP, Macs, PowerPC, MIPS, DLX, IBM370, OpenRISC
    - best to handle left-to-right text.
  - Little-endian: Least significant byte has lowest address
    - Intel Pentium, DEC Alpha, ARM, VAX, PDP-11, Core-A
    - best to handle position-dependent data such as number.

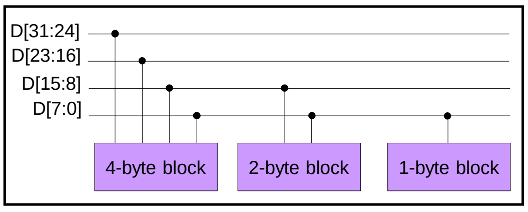




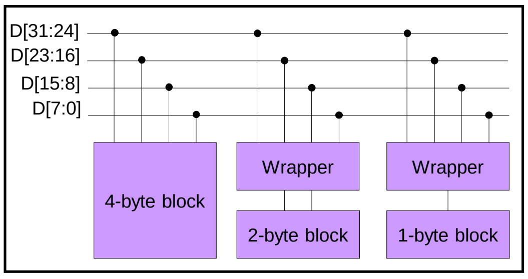
Refer to: 'On holy wars and a plea for peace' by D. Cohen, IEEE Computer Vol.14, No.10, Oct. 1981, p.48-54.

Justified or non-justified

- Justified bus
  - Byte always travels on rightmost or leftmost quarter of bus
    - size determines the lanes that data actually use.
  - Wishbone bus

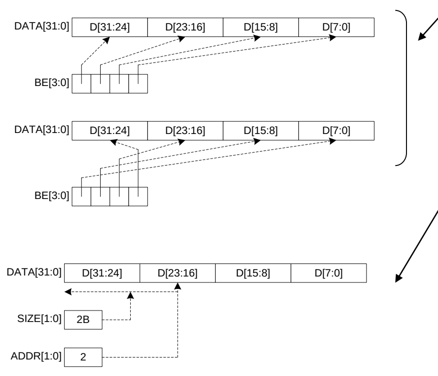


- Non-justified bus / unjustified bus
  - Bus lanes are extension of memory bank lane.
    - address determines the lanes that data actually use.
      - More complex cases with endianness.
  - AMBA bus
  - E.g., how to interface 32-bit memory with 64 or 128-bit AXI bus?



Partial/narrow access

- For example
  - How to read/write two-byte through 32-bit data bus?
    - For read, read full width and ignore some.
    - For write, need something to indicate which bytes are important

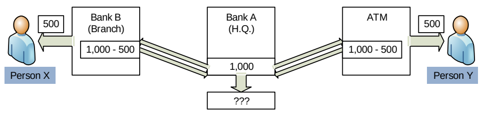


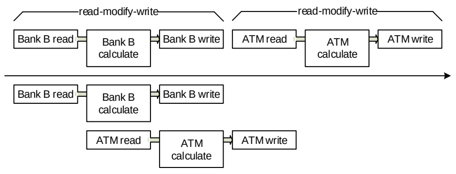
- Partial access is used to read/write a fewer number of bytes than the data bus width.
  - Byte enable
    - Enable signals are given to indicate which byte are active.
    - DATA[31:0] with BE[3:0]
  - Size and address
    - Size and a lower bits of address determines active bytes.
    - DATA[31:0] with SIZE[1:0] and ADDR[1:0]

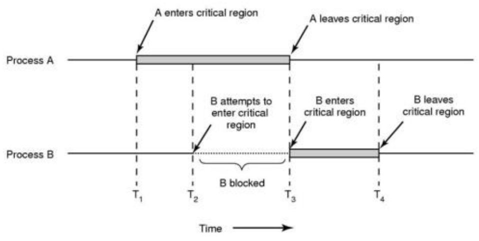
Alignment of access

- Are there any rule between address and size of access
  - two-byte access should be with address with a multiple of 2.
    - E.g, 0, 2, 4, 6, …
  - four-byte access should be with address with a multiple of 4.
    - E.g., 0, 4, 8, 16, …
  - How about three-byte case
    - Usually most processor does not generate this kind of access
    - So, most bus systems does not support this, but there are exceptions.
- How about burst accesses with bus wider than 4-bytes data lane?
  - E.g., 64-bit wide (8-byte data lane) or 128-bit wide
  - Is it possible to make all access be data-width aligned?
    - No, then what happens. Or how to deal with it.
    - Let see this for AXI case.

Atomic & mutual exclusion

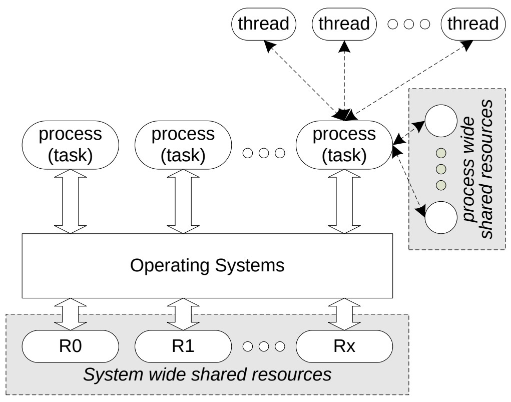






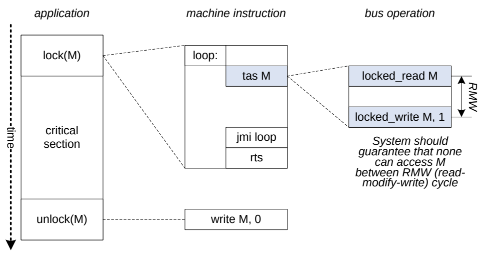
Atomic operations (1/2)

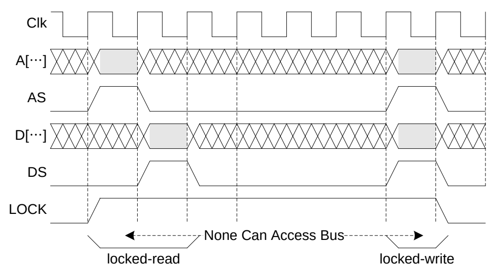
- Application requires mutual exclusion in order to protect critical section.
  - What kind of relations with system bus?



Atomic operations (2/2)

- Microprocessors have special instructions for atomic access in order to support mutual exclusion.
  - ARM: SWAP
    - LDREX (load exclusive), STREX (store exclusive)
  - MC680X0: CAS (Compare And Swap)
  - Intel: lock inc, lock dec, lock xchg, lock add, lock sub
  - Sparc: cas, ldstub
  - MIPS: ll (load linked), sc (store conditional)
  - DEC Alpha: ld_l, stl_c
  - PowerPC: lwarx, stwcx
  - Core-A: EXCHG





Atomic operations

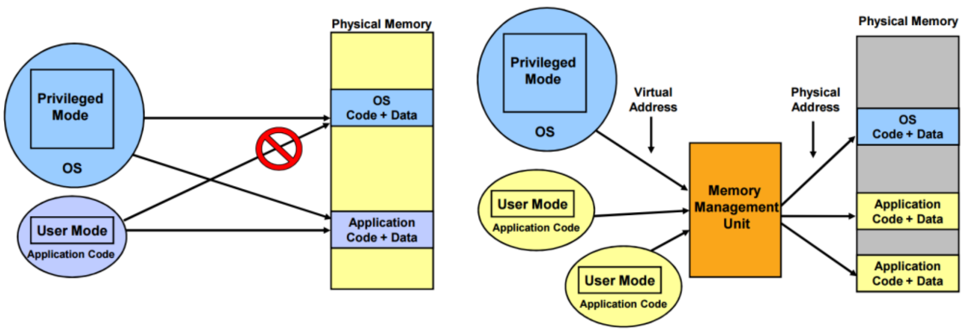
- ‘swap(X, 1);’ consists of
  - locked-read X
    - It gets current value of X
  - locked-write 1 to X
    - It updates X by 1 regardless its current value
  - locked-read and locked-write are protected by bus as read-modify-write
- Only one can enter the critical section at a given time and any other cannot enter the region until the first one unlock it.

Program A
```
…
…
while (swap(X, 1));
… // critical section
…
X = 0; // unlock
…
```

Program B
```
…
…
while (swap(X, 1));
… // critical section
…
X = 0; // unlock
…
```

Program C
```
…
…
while (swap(X, 1));
… // critical section
…
X = 0; // unlock
…
```

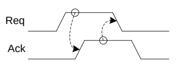
Memory protection



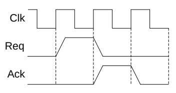
Memory Management Unit (MMU)
- Controls accesses to and from external memory
- Assigns access permissions to memory regions
- Performs virtual to physical address translation

Synchronous and asynchronous

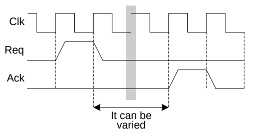
- asynchronous
  - No common clock



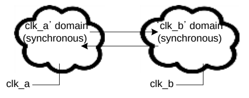
- synchronous
  - common clock
  - use pre-defined timing point in terms of clock-edge or # of clock-cycles to carry out bus operation



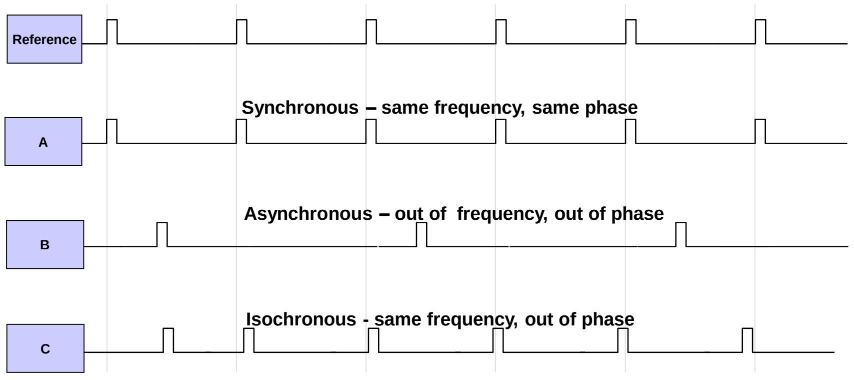
- How about this
  - Is it synchronous or asynchronous?
  - E.g., HADDR[…] to HREADY



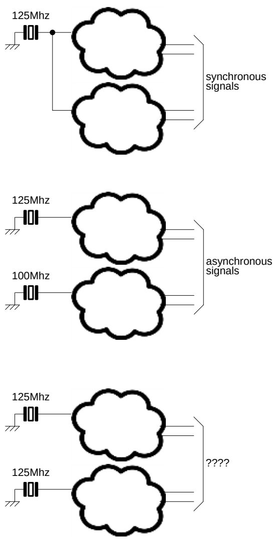
- Clock-domain crossing: asynchronous

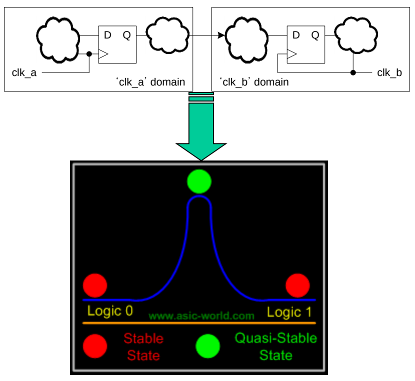


Clock frequency and phase


Reference
A
Synchronous – same frequency, same phase
B
Asynchronous – out of frequency, out of phase
C
Isochronous - same frequency, out of phase

Synchronous or not

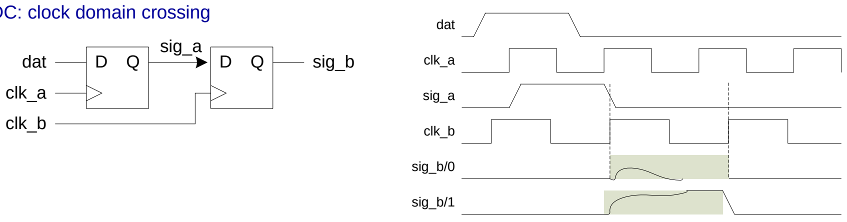




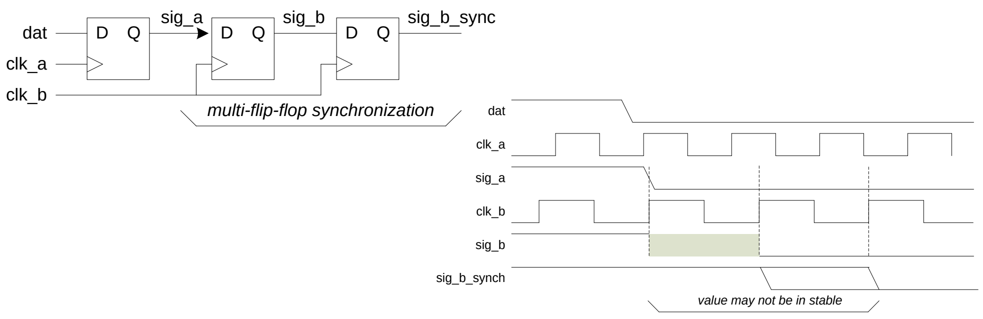
Whenever the input signal D does not meet the Tsu (Setup) and Th (Hold) of the given D flip-flop, metastability occurs.

CDC and Multi-flip-flop synchronization

CDC: clock domain crossing

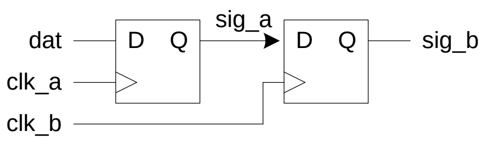


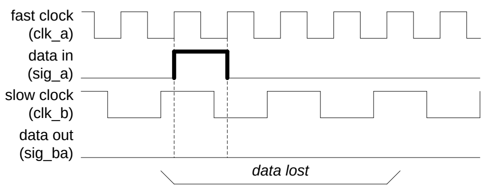
'sig_b' can be settle down to '0' or '1' depending on situation (fan-out, temperature, ...).



CDC problems

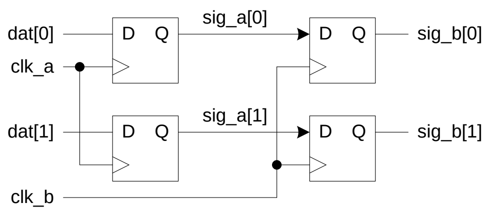
- Data can be lost with fast-to-slow clock domain
- Solutions
  - Edge detection
  - Feedback or handshake

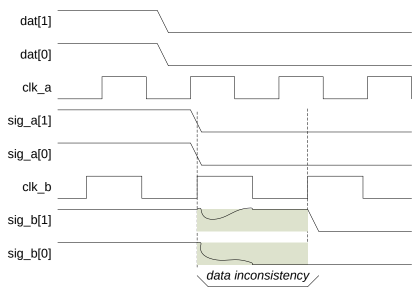




CDC problems

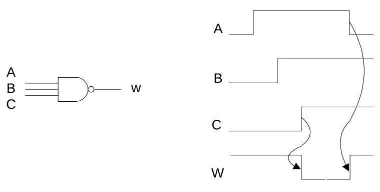
- Parallel data can be inconsistency due to meta-stability
- Solutions
  - Synchronization and ignore some intermediate cycles
  - MUX synchronization
  - Use single-bit changing code, such an grey code
    - But not applicable for all applications

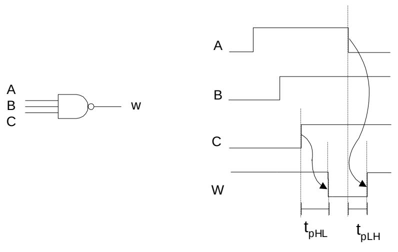




Timing diagram

- Functional Timing Diagram
  - assumes zero delays
  - simply demonstrates logic relations
- Timing Diagram with delay
  - tpHL: High-to-Low propagation delay
  - tpLH: Low-to-High propagation delay





Timing diagram convention

Ideal signals have 0 rise and fall times

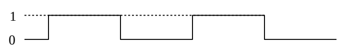


Figure 5.5 Ideal logic level

A real signal has nonzero rise and fall times

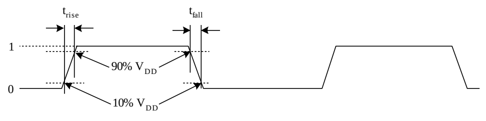


Figure 5.6 Real signal

Normal signal representation: single and multi-bit

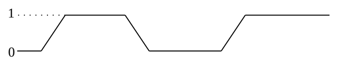


Figure 5.7 Single-signal waveform

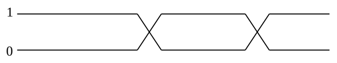
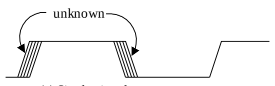
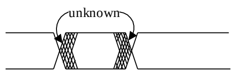


Figure 5.8 Multiple-signal waveform

Unknown signals (when they are changing) representation



(a) Single signal

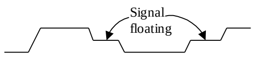


(b) multiple signals

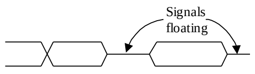
Figure 5.9 Unknown signals

Timing diagram convention

Floating signal



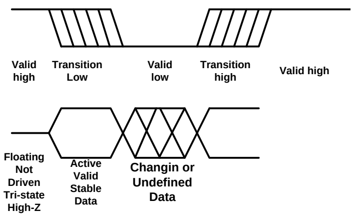
(a) Single signal



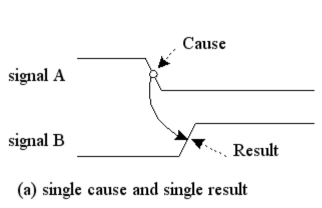
(b) multiple signals

Figure 5.10 Floating signals

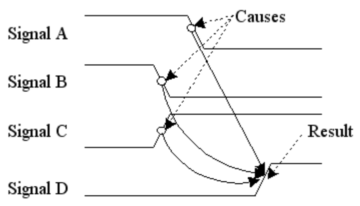
Unknown or changing signals



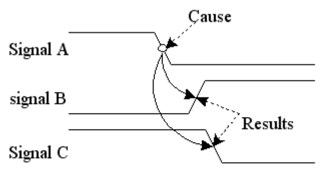
Signal causal relationships



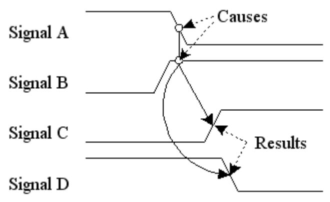
(a) single cause and single result



(b) multiple causes and single result



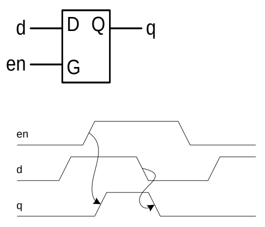
(c) single cause and multiple results

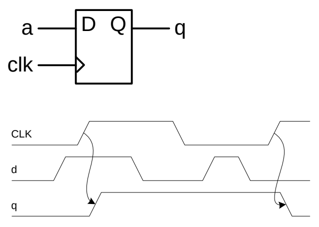


(d) multiple causes and multiple results

Timing diagram example

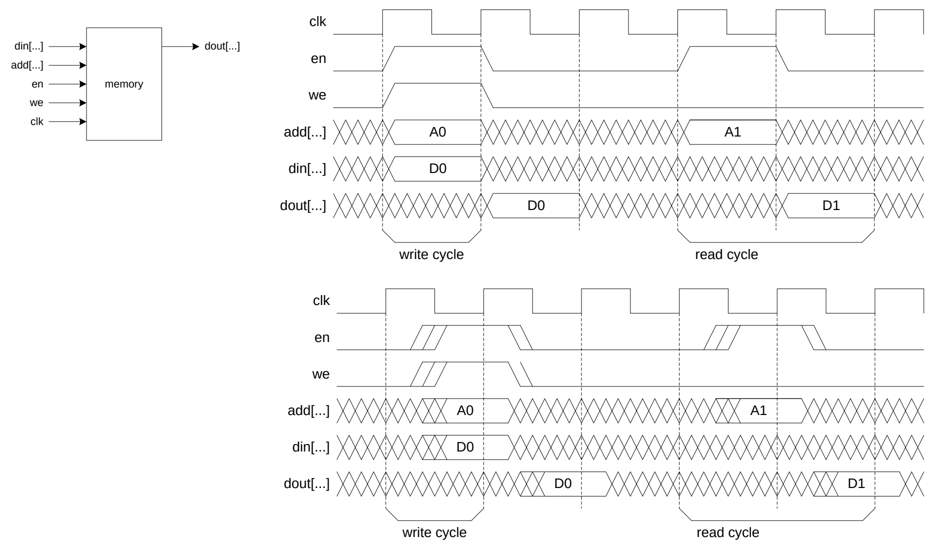
- Latch
  - When en = 1, latch is *transparent*
    - D flows through to Q like a buffer
  - When en = 0, the latch is *opaque*
    - Q holds its old value independent of D
  - a.k.a. *transparent latch* or *level-sensitive latch*
- Flip-flop
  - When CLK rises, D is copied to Q
  - At all other times, Q holds its value
  - a.k.a. *positive edge-triggered flip-flop*, *master-slave flip-flop*





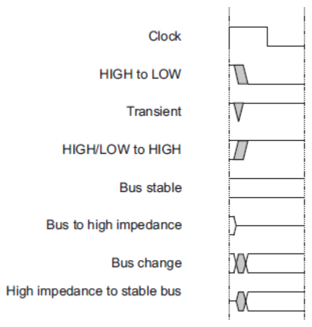
Timing diagram example

- An example of simple memory

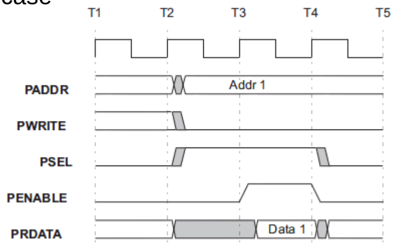


Timing diagram example

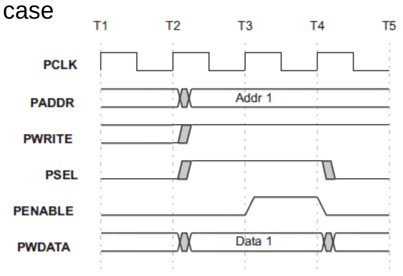
- An example of AMBA APB read/write timing diagram with bus timing notation.



- Read case

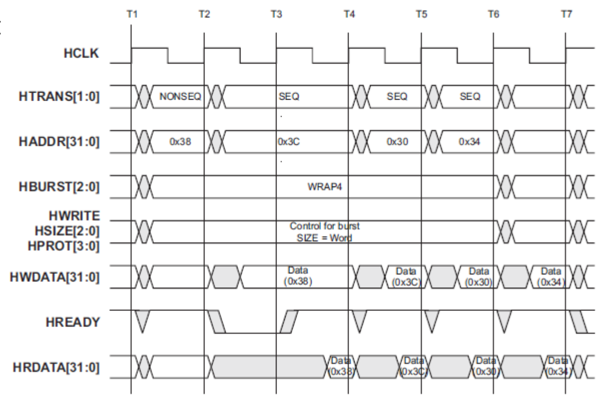


- Write case



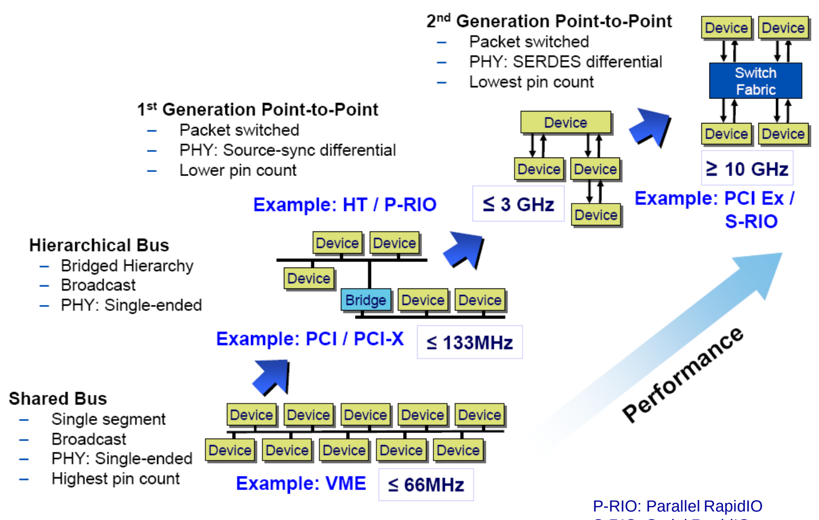
Timing diagram example

- AMBA AHB four-beat wrapping case



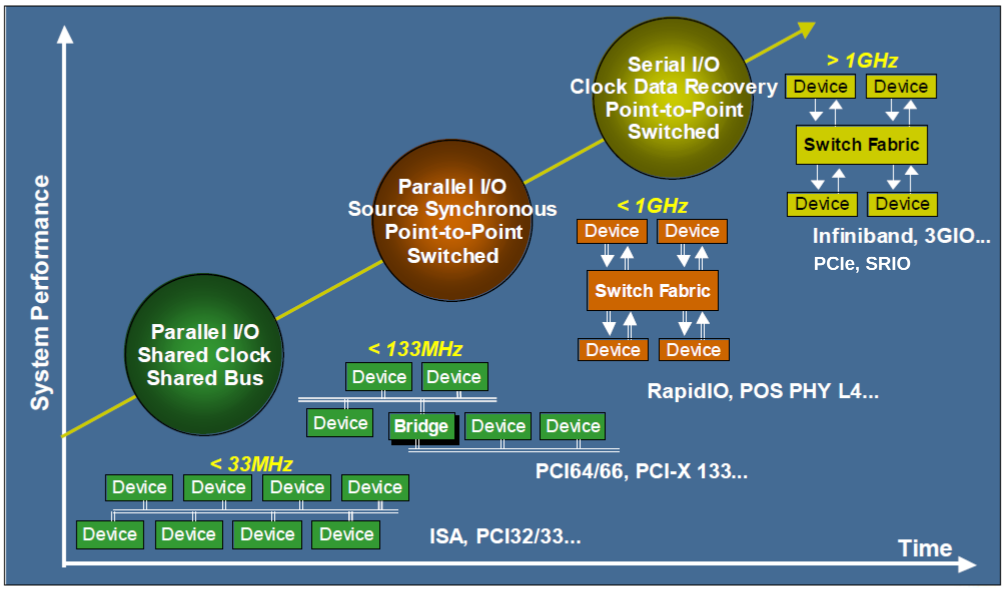
Trends of interconnect

High performance system tends to adopt point-to-point switched interconnects.



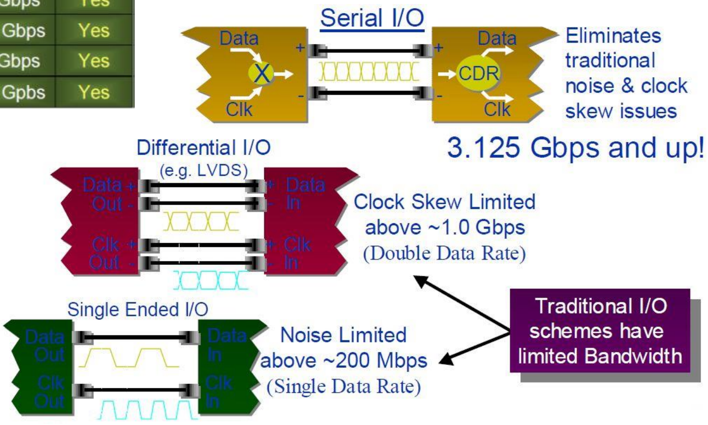
P-RIO: Parallel RapidIO
S-RIO: Serial RapidIO

Trends of interconnect



Serial standards

Standards	Line Speed	8B/10B ENDEC
3GIO™	2.5 Gbps	Yes
Serial ATA™	1.5 Gbps	Yes
InfiniBand™	2.5 Gbps	Yes
Gb Ethernet	1.25 Gbps	Yes
10Gb Ethernet (XAUI)	3.125 Gbps	Yes
Serial RapidIO™	1.25 Gbps	Yes
Xilinx 40G Backplane	3.125 Gpbs	Yes



Serial I/O
Data
Clk
X
CDR
Eliminates traditional noise & clock skew issues
3.125 Gbps and up!
Differential I/O
(e.g. LVDS)
Data Out
Clk Out
Data In
Clk In
Clock Skew Limited above ~1.0 Gbps
(Double Data Rate)
Single Ended I/O
Noise Limited above ~200 Mbps
(Single Data Rate)
Traditional I/O schemes have limited Bandwidth

References

- D. Del Corso et.al., Microcomputer buses and links, Academic press, 1986.
- D. Cohen, On holy wars and a plea for peace, IEEE Computer Vol.14, No.10, Oct. 1981, p.48-54.
- S. Pasricha and N. Dutt, On-Chip Communication Architectures System on Chip Interconnect, Morgan Kaufmann Pub. 2011.

- 기안도 외, 고중첩 버스 : HiPi-Bus (Highly Pipelined Bus : Hipi-Bus), 대한전자공학회 학술발표회 논문집 (반도체/재료부품/CAD/VLSI) 제10권 1호, 1992.1, 31-37.
- Ando Ki et.al., Higly Pipelined Bus: HiPi-Bus, JTC-CSCC : Joint Technical Conference on Circuits Systems, Computers and Communications, 1991.
- Seongwoon Kim and Ando Ki et.al., RACE on a physically distributed and logically shared memory system.