4. Bus-Based Computer Systems

- CPU bus, I/O devices, and interfacing
- CPU system as a framework
- System level performance
- Development and debugging

Bus-Based Computer Systems

- Microprocessors
- Busses.
- Memory devices.
- I/O devices:
  - serial links
  - timers and counters
  - Keyboards, displays
  - analog I/O

Summary

- How to interconnect the components with the system bus

4.2 memory

4.3 I/O devices

4.4. Interfaces for memory and I/O devices

4.5 platform

4.6 debugging

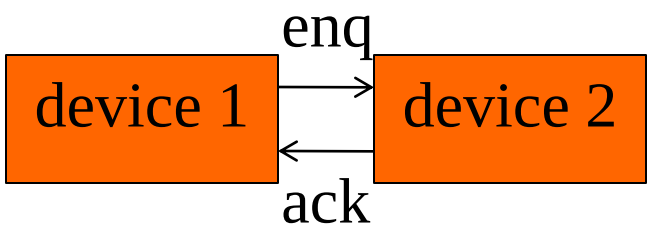
4.1 The CPU bus

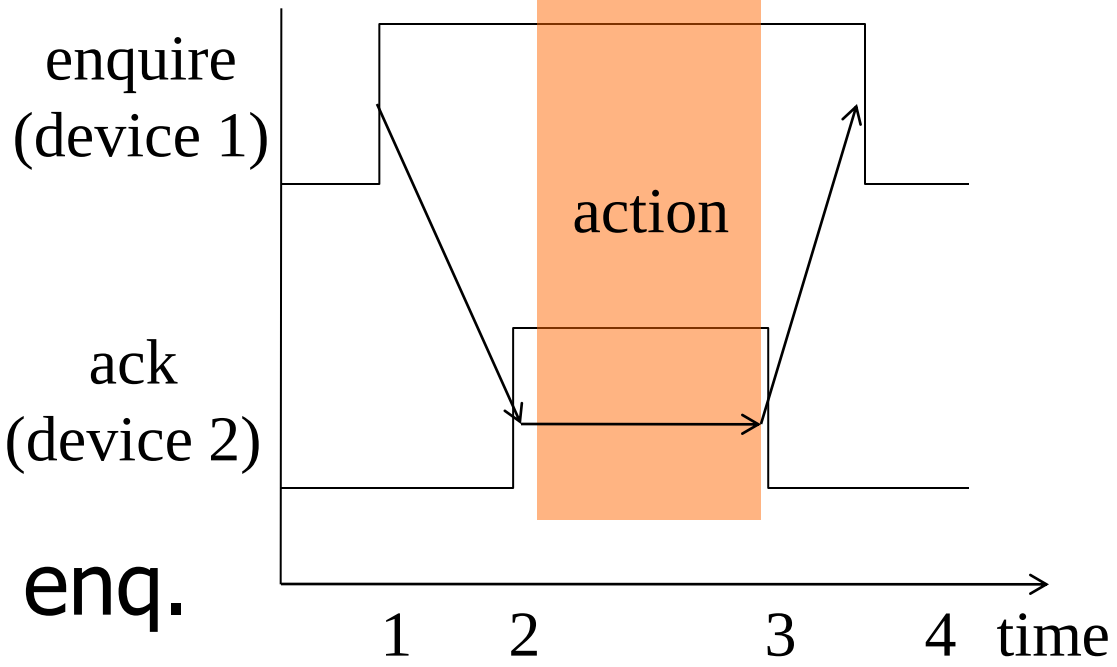
- Bus allows CPU, memory, devices to communicate.
  - Shared communication medium.
- A bus is:
  - A set of wires.
  - A communications protocol.

4.1.1 Bus protocols

- Bus protocol determines how devices communicate.
- Devices on the bus go through sequences of states.
  - Protocols are specified by state machines
  - One state machine per actor in the protocol
- May contain asynchronous logic behavior.

Four-cycle handshake



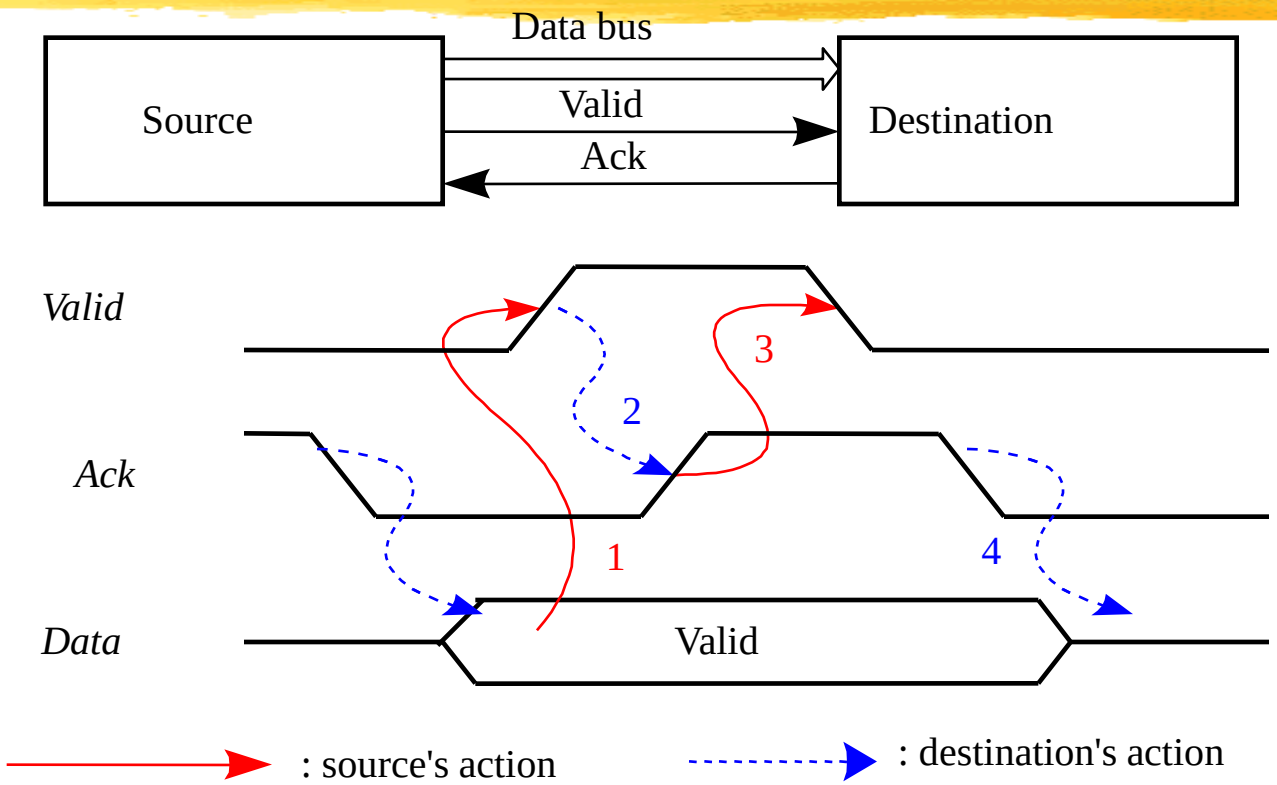


1. Device 1 raises enq.
2. Device 2 responds with ack.
3. Device 2 lowers ack once it has finished.
4. Device 1 lowers enq.

Handshaking

- In the handshaking transfer, four events are proceeded in a cycle order:
  1. ready (request):
  2. data valid:
  3. data acceptance:
  4. acknowledge:

Source-initiated transfer (Valid-Ack)



(ready)

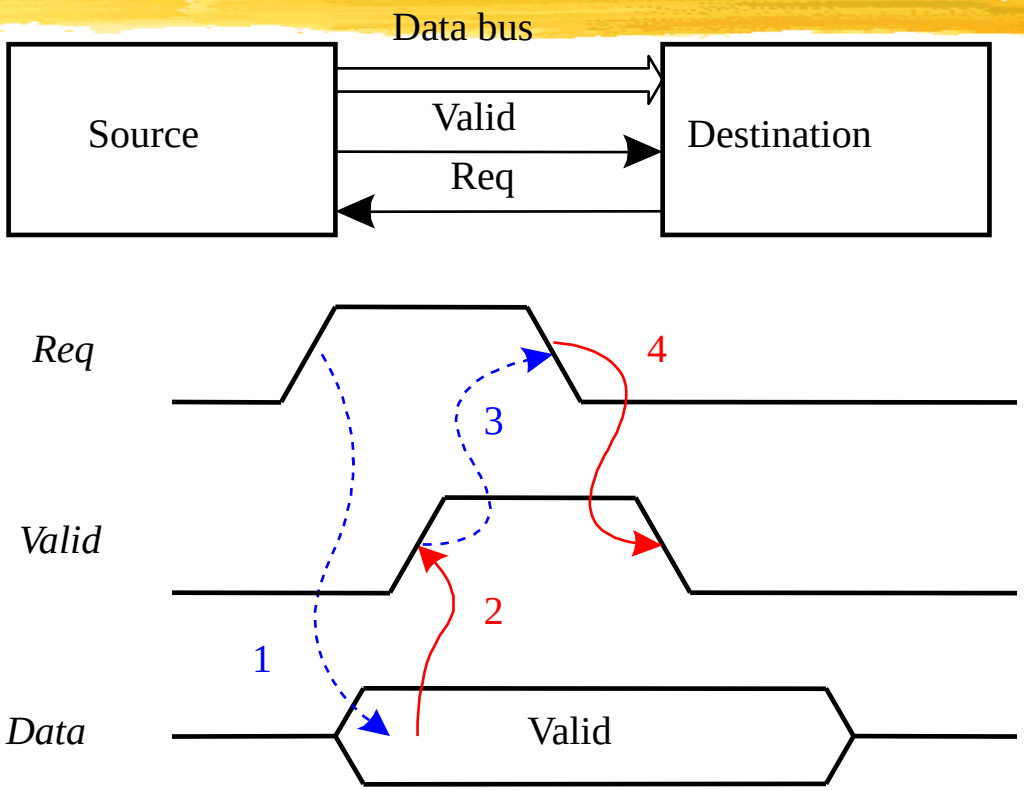
(1. Source sends data: Valid ↑) (2. Destination receives data: Ack ↑)

(3. Source acknowledge: Valid ↓) (4. Destination is ready : Ack ↓)

Source-initiated transfer

- In the handshaking transfer, four events are proceeded in a cycle order:
  1. ready: The destination device deasserts the acknowledge signal and is ready to accept the next data.
  2. data valid: The source device places the data onto the data bus and asserts the valid signal to notify the destination device that the data on the data bus is valid.
  3. data acceptance: The destination device accepts (latches) the data from the data bus and asserts the acknowledge signal.
  4. acknowledge: The source device invalidates data on the data bus and deasserts the valid signal

Destination-initiated transfer (Req–Valid)



(ready) ——→ : source's action ----------→ : destination's action

(1. Destination requests data: Req↑) (2. Source sends data: Valid ↑)

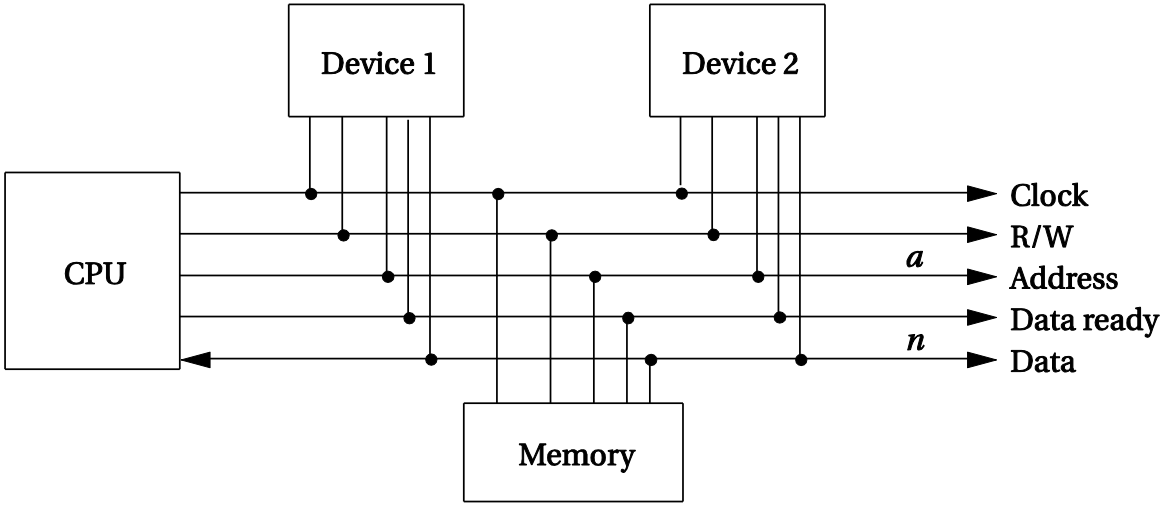
(3. Destination receives data: Req↓) (4. Source acknowledge: Valid ↓)

Destination-initiated transfer

- In the handshaking transfer, four events are proceeded in a cycle order:
  1. request: The destination device asserts the request signal to request data from the source device.
  2. data valid: The source device places the data on the data bus and asserts the valid signal to notify the destination device that the data is valid now.
  3. data acceptance: The destination device accepts (latches) the data from the data bus and asserts the request signal.
  4. acknowledge: The source device invalidates data on the data bus and deasserts the valid signal to notify the destination device that it has removed the data from the data bus

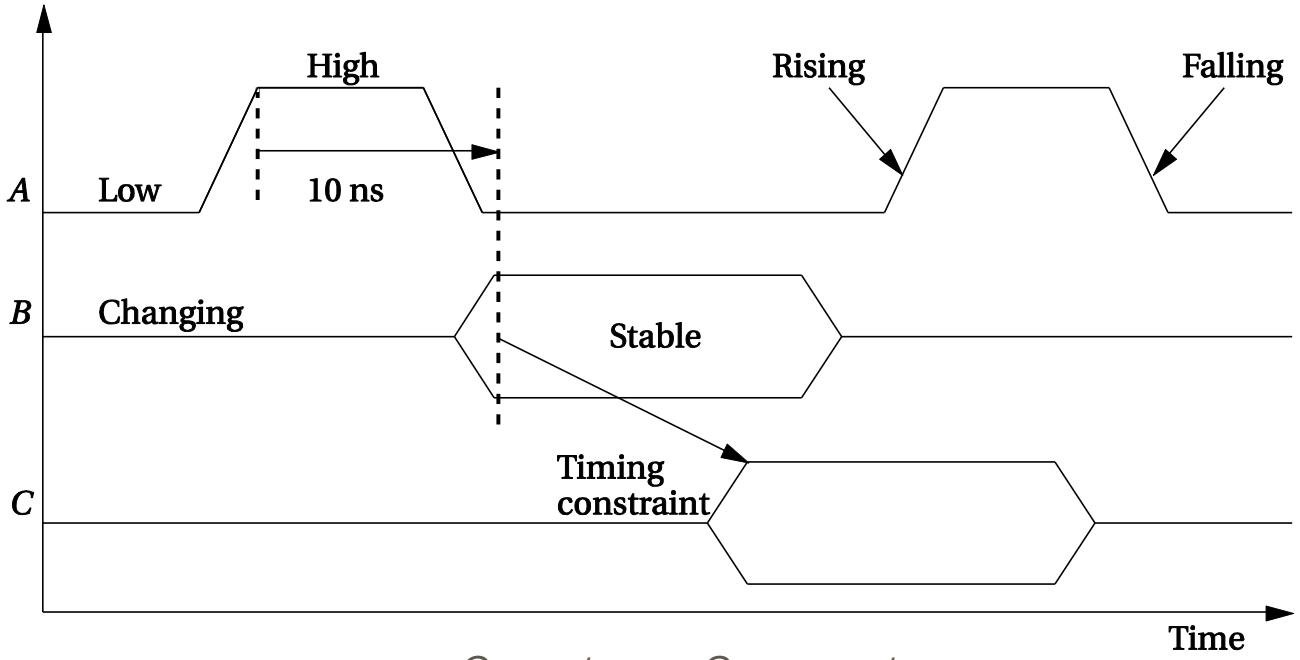
Microprocessor busses

- Clock provides synchronization.
- R/W is true when reading (R/W' is false when reading).
- Address is a-bit bundle of address lines.
- Data is n-bit bundle of data lines.
- Data ready signals when n-bit data is ready.

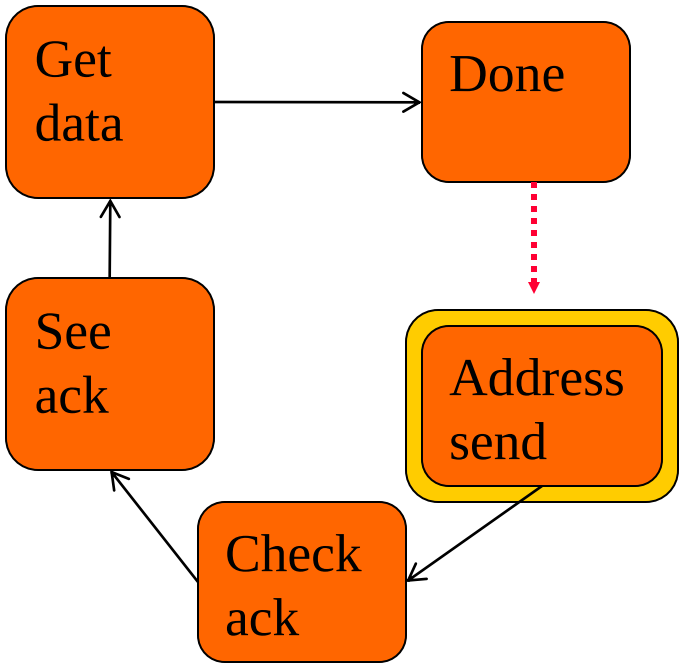


Timing diagrams

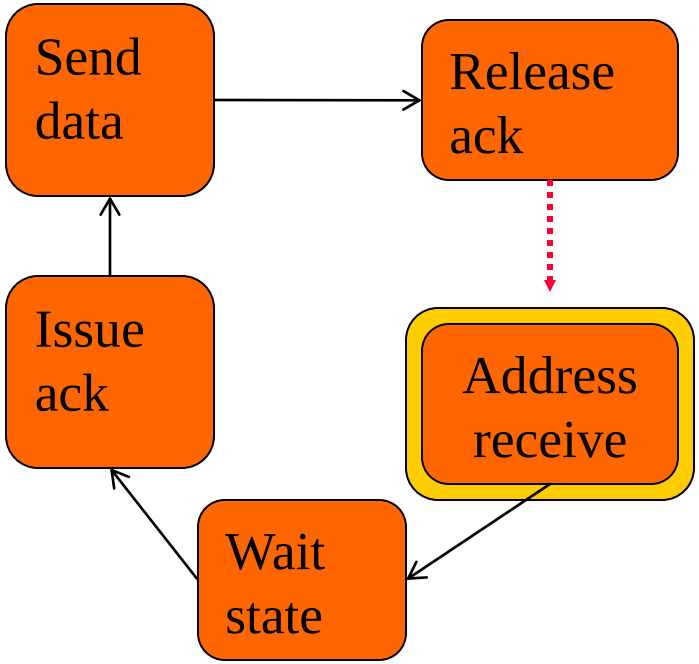
- Bus behavior is often specified as a timing diagram.
- Changing and stable
- Timing constraints



State diagrams for bus read

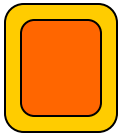


CPU's state diagam



Device's state diagram

Start state



Multiple Bus Reads

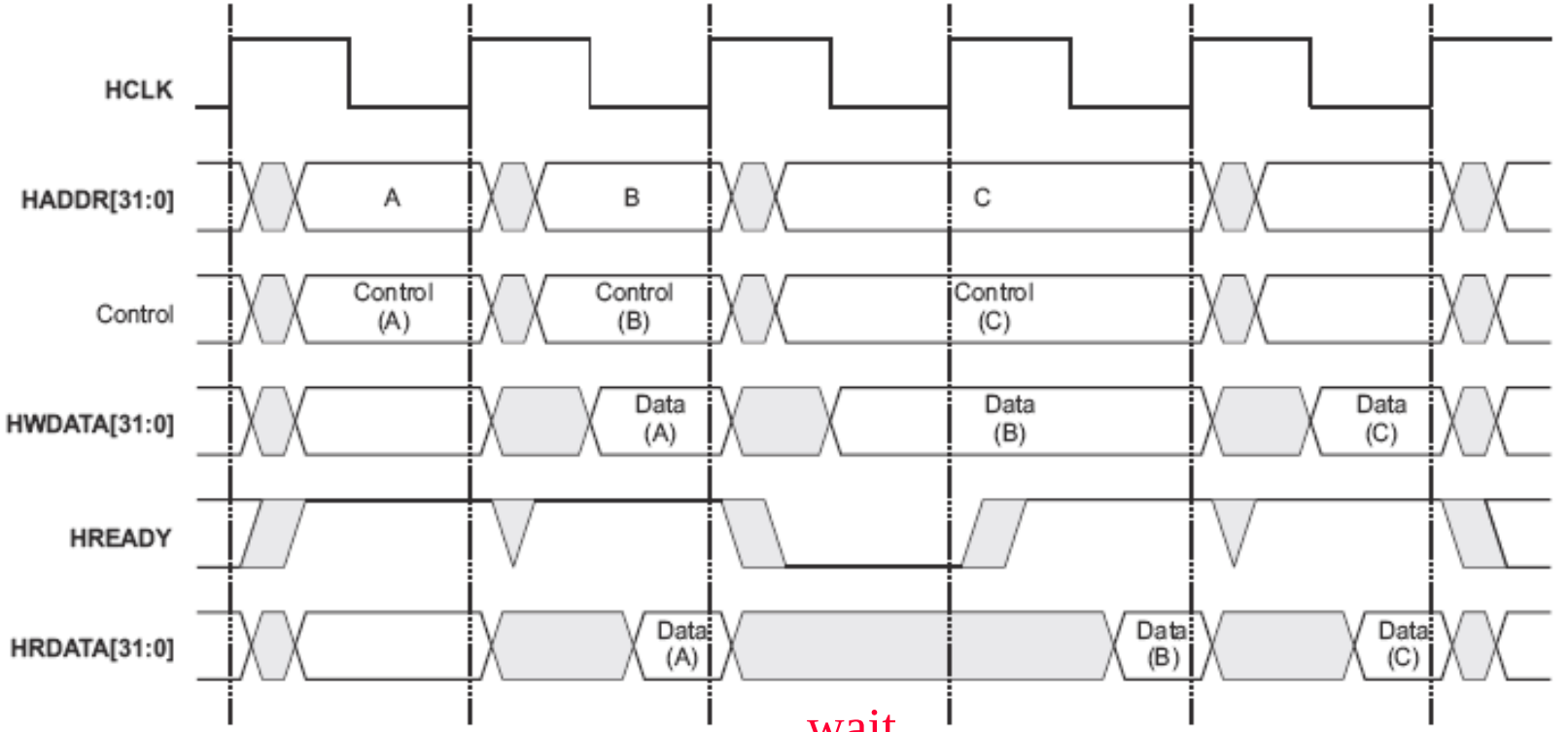
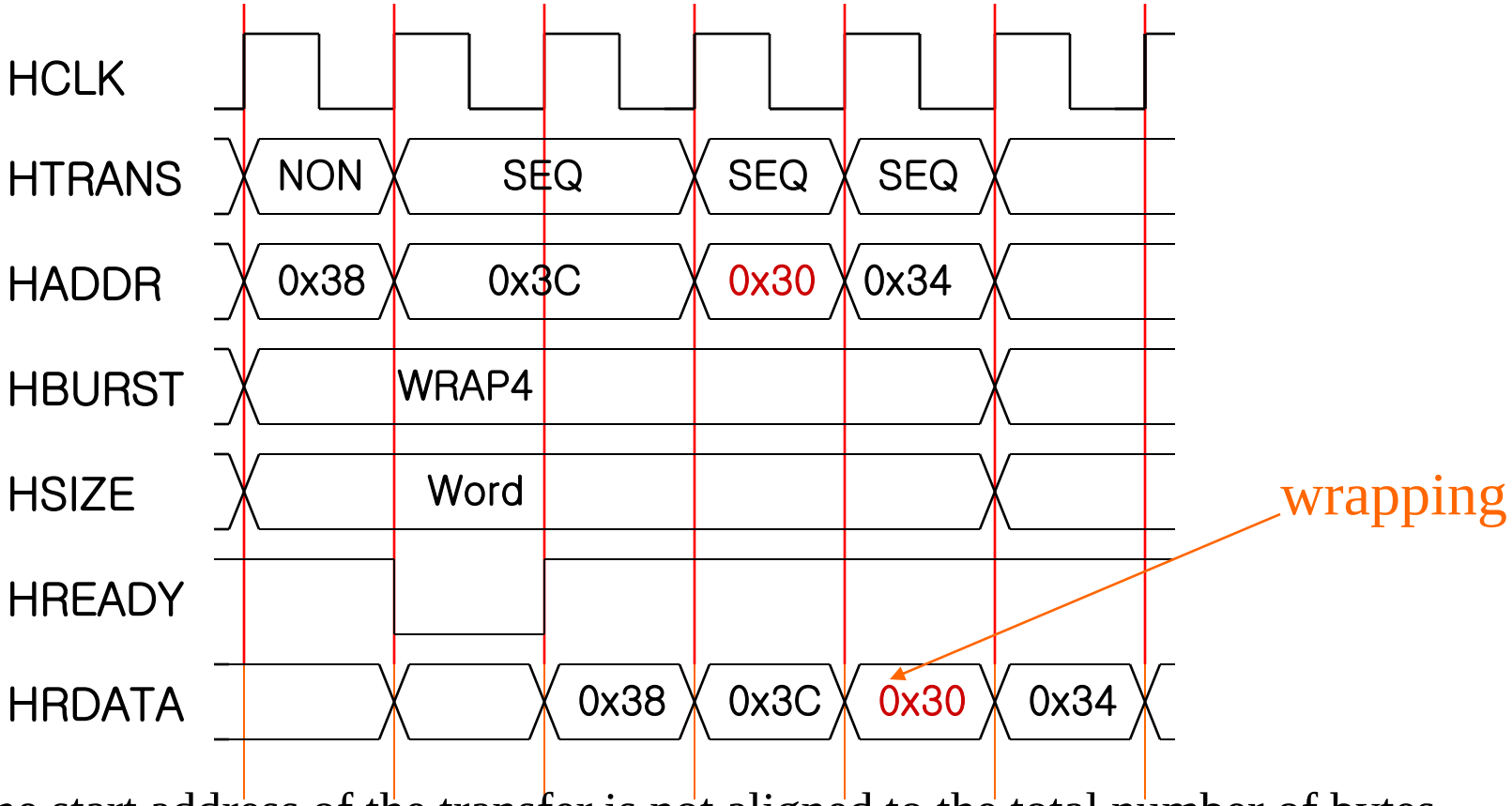


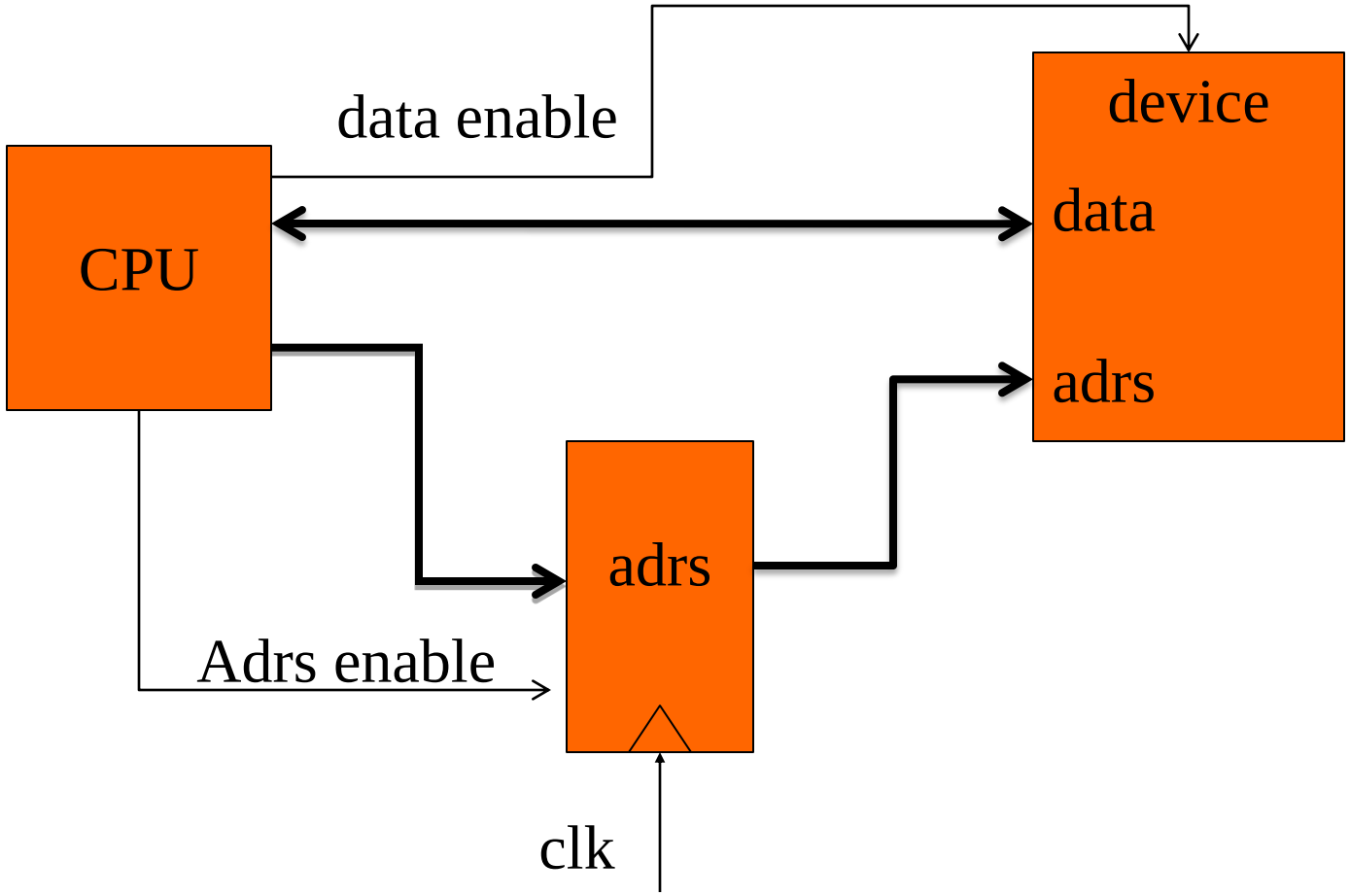
Figure 3-5 Multiple transfers

Four-beat Wrapping Burst



If the start address of the transfer is not aligned to the total number of bytes (size x beats) then the address of the transfers in the burst will wrap when the boundary is reached.

Address and data Buses



Address decoding

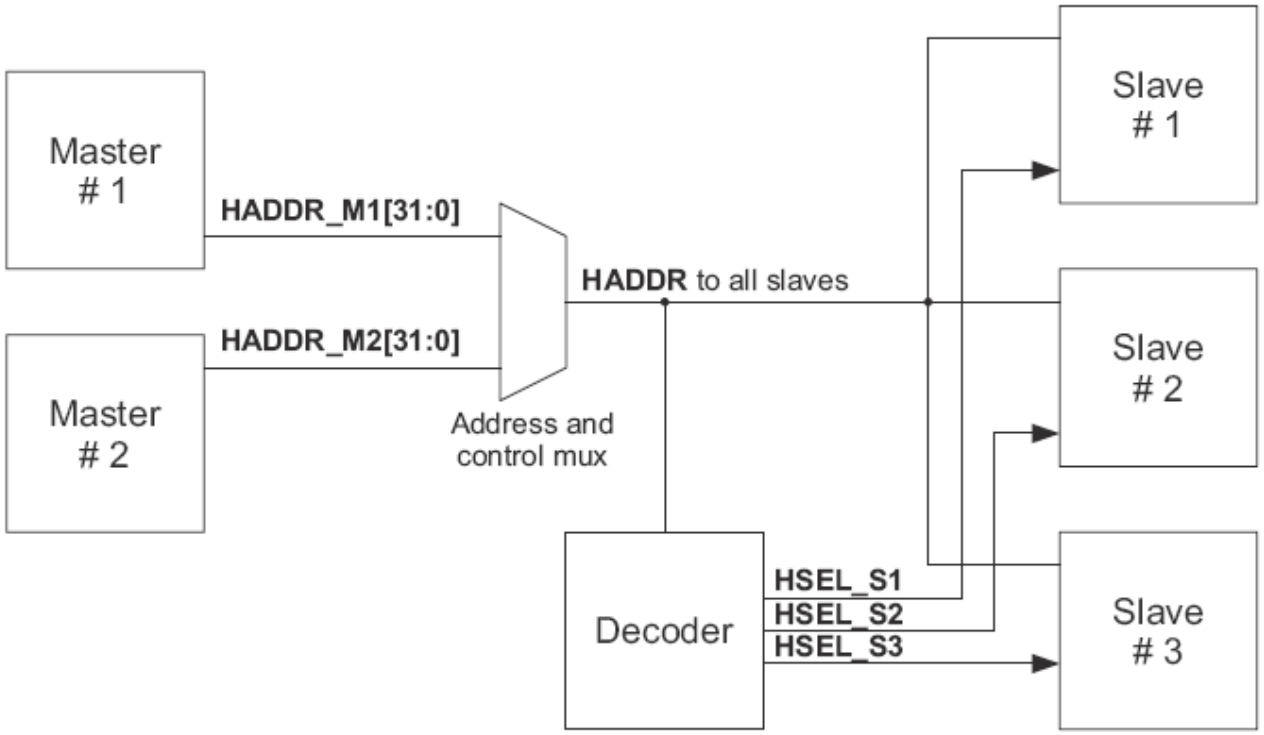
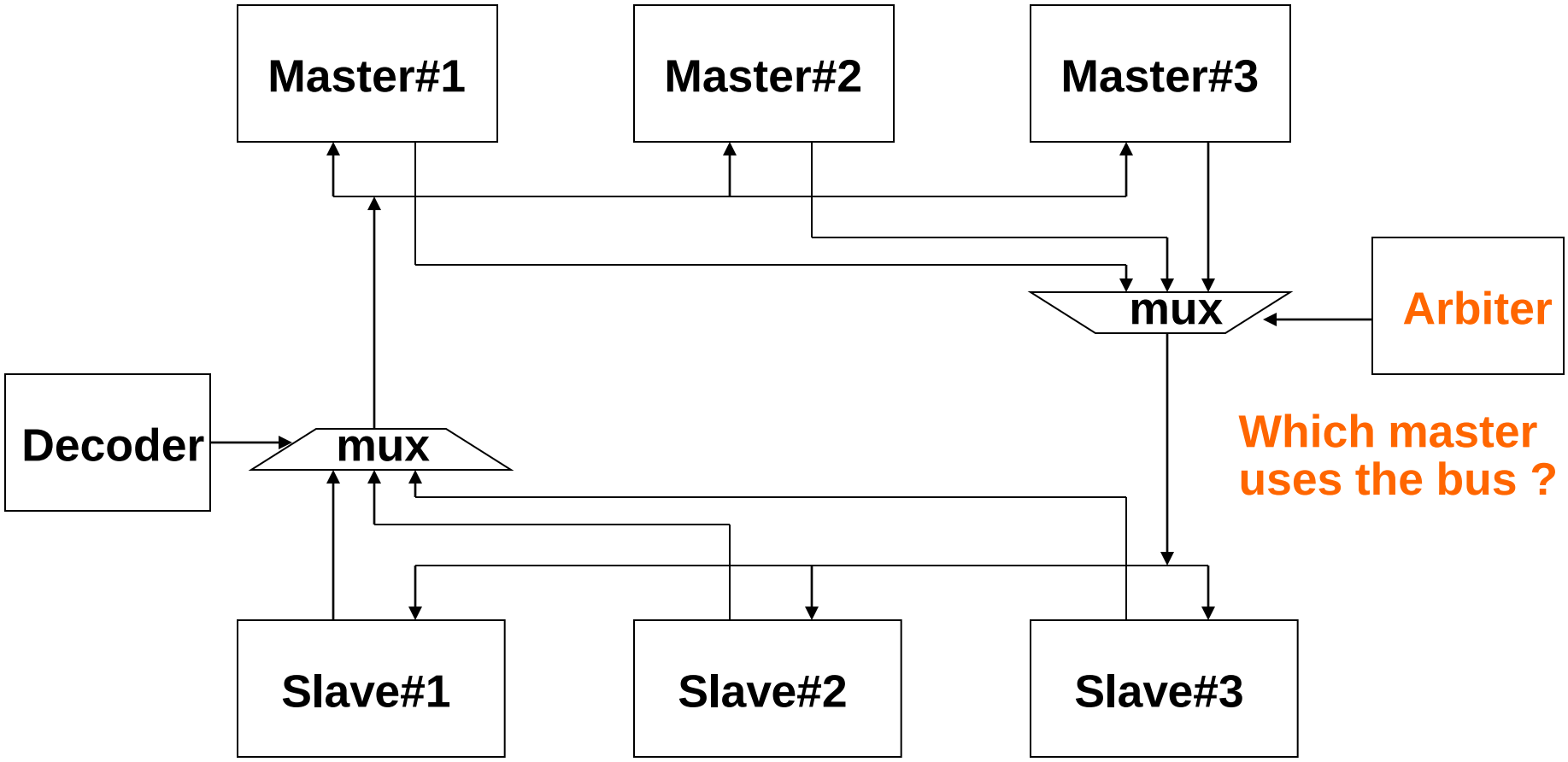


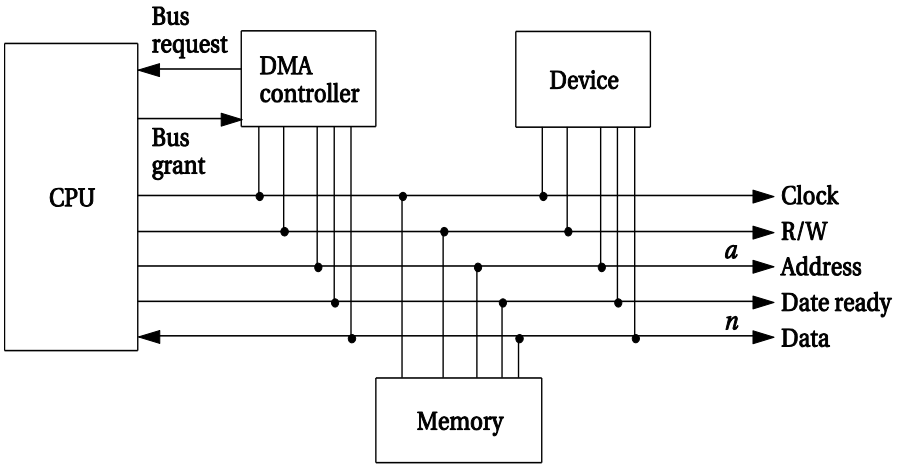
Figure 3-12 Slave select signals

Arbitration



4.1.2 DMA

- Direct memory access (DMA) performs data transfers without executing instructions.
  - CPU sets up transfer.
  - DMA engine fetches, writes.
- DMA controller is a separate unit.

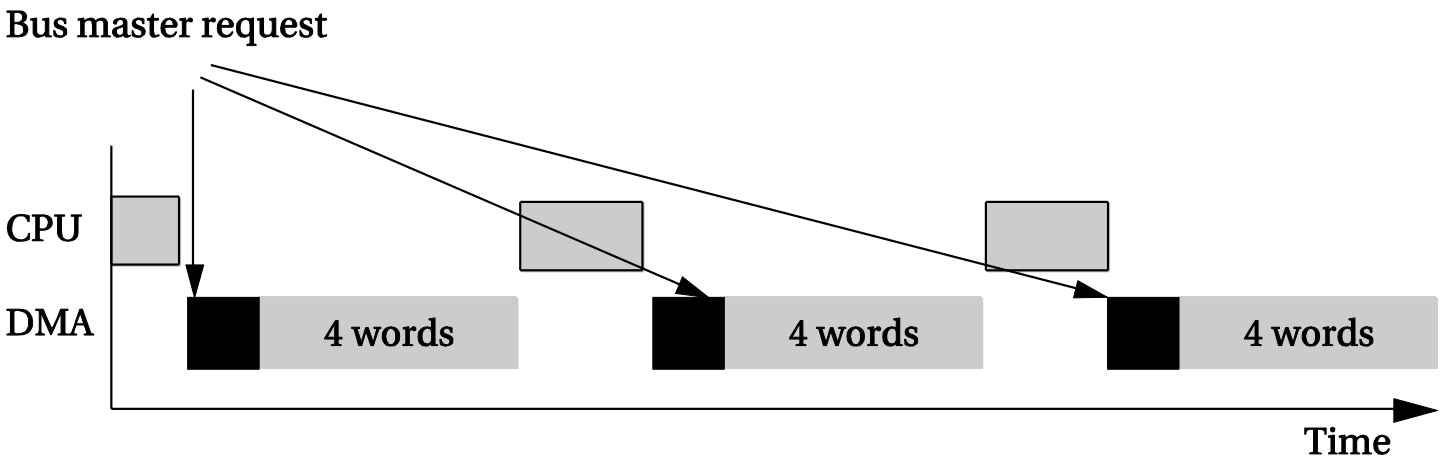


Bus mastership

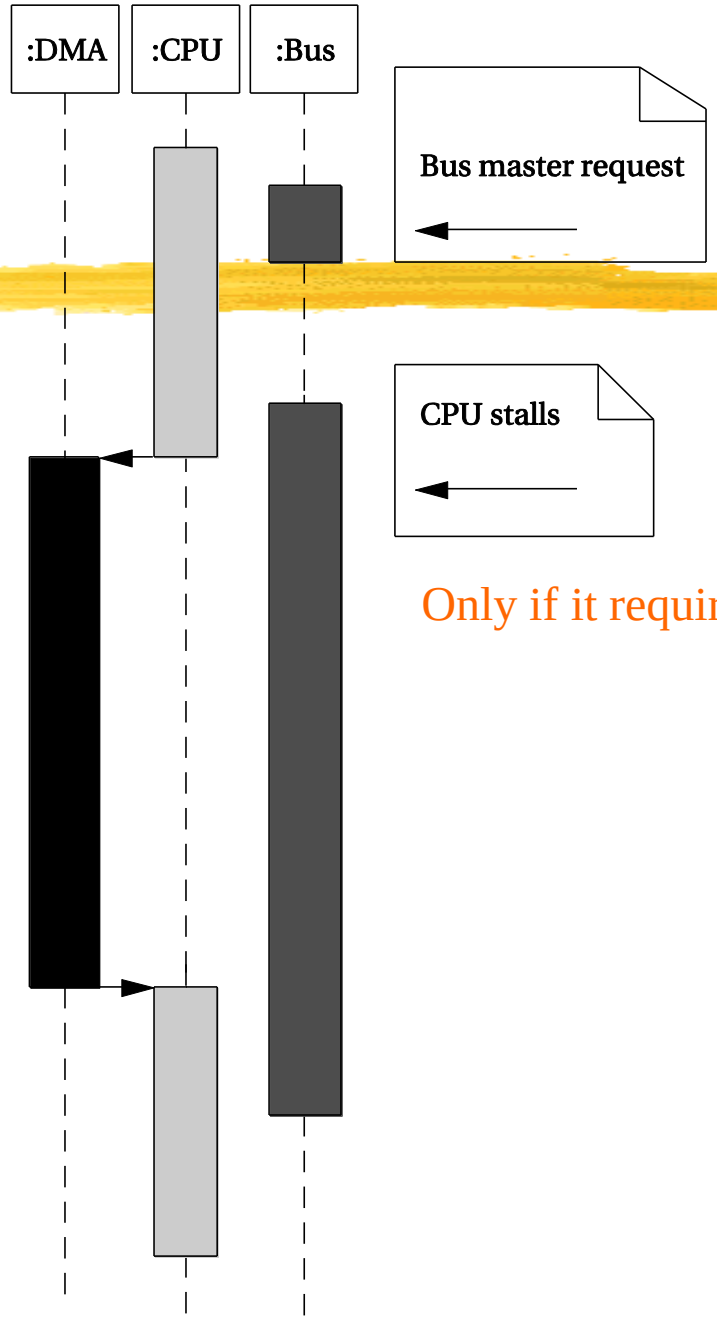
- By default, CPU is bus master and initiates transfers.
- DMA must become bus master to perform its work.
  - CPU can't use bus while DMA operates.
- Bus mastership protocol:
  - Bus request.
  - Bus grant.

DMA operation

- CPU sets DMA registers for start address, length.
- DMA status register controls the unit.
- Once DMA is bus master, it transfers automatically.
  - May run continuously until complete.
  - May use every n^{th} bus cycle.



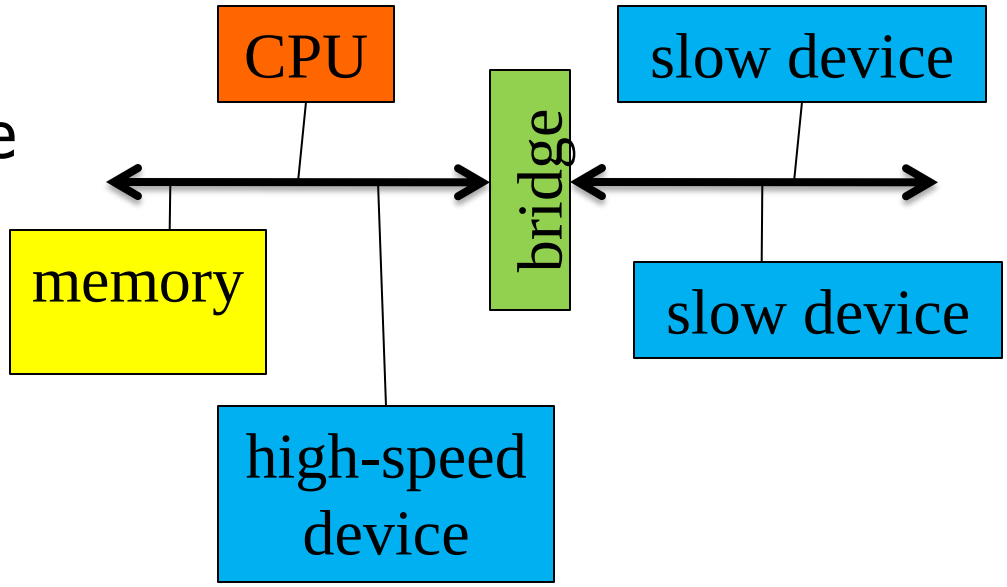
Bus transfer sequence diagram



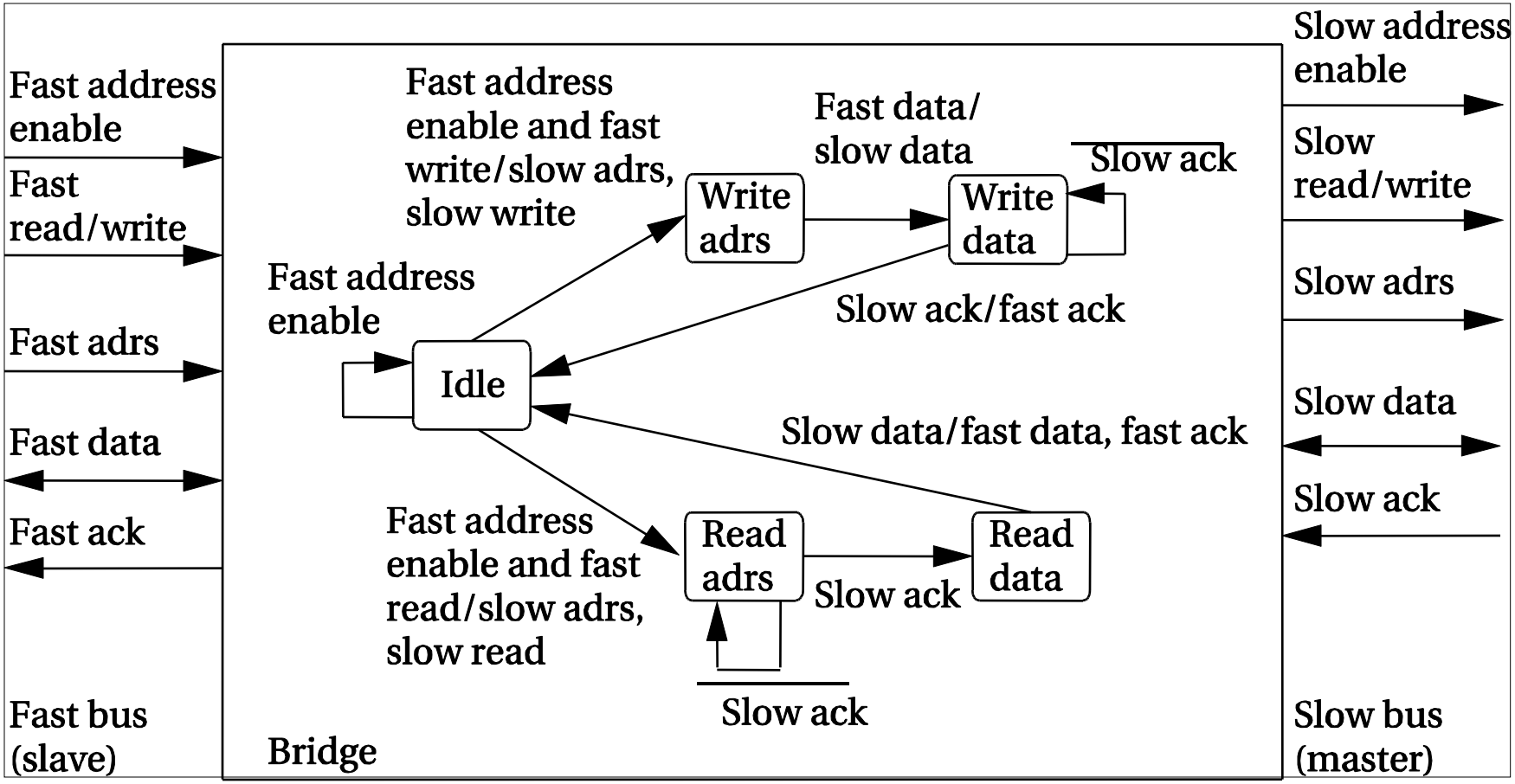
Only if it requires to use the bus

System bus configurations

- Multiple busses allow parallelism:
  - Slow devices on one bus.
  - Fast devices on separate bus.
- A bridge connects two busses.



Bridge state diagram

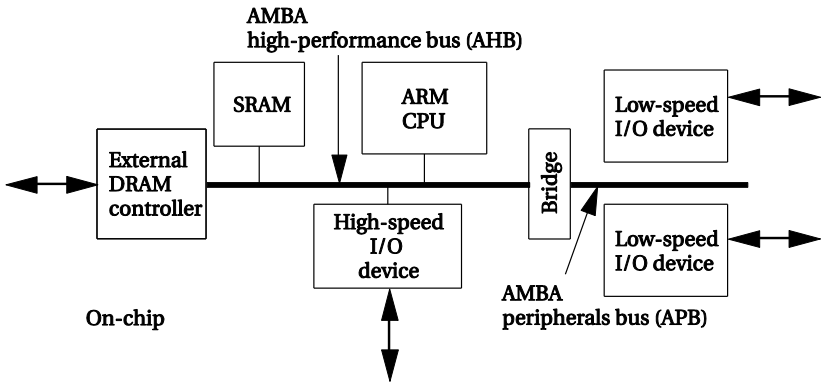


Bus Bridge

- A slave on a fast bus and the master of the slow bus
- It takes commands from the fast bus and issues those commands on the slow bus
- It also returns the results form the slow bus to the fast bus.

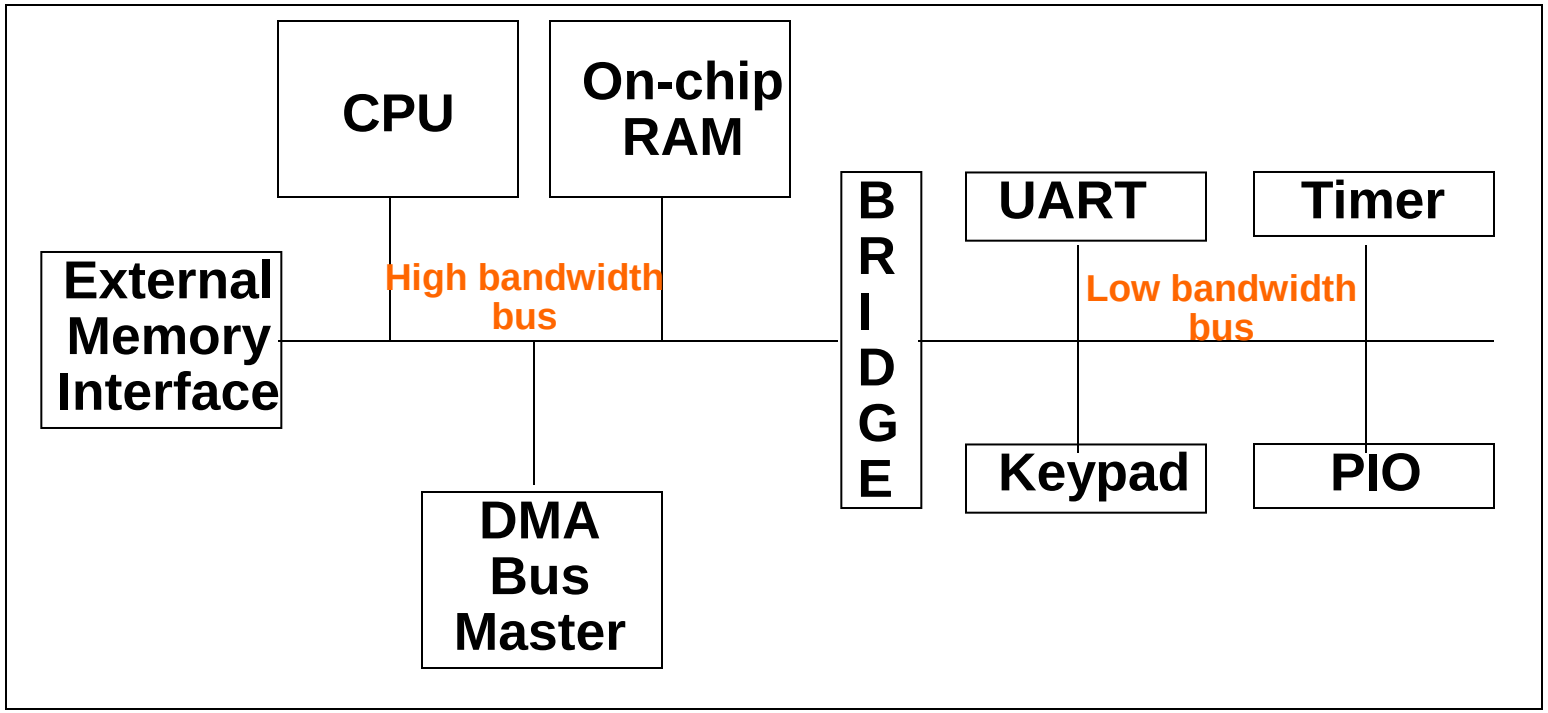
ARM AMBA bus

- Two varieties:
  - AHB is high-performance.
  - APB is lower-speed, lower cost.
- AHB supports pipelining, burst transfers, split transactions, multiple bus masters.
- All devices are slaves on APB.
- Read AMBA specification



On-Chip Bus (OCB)

- Interconnect components inside a single chip



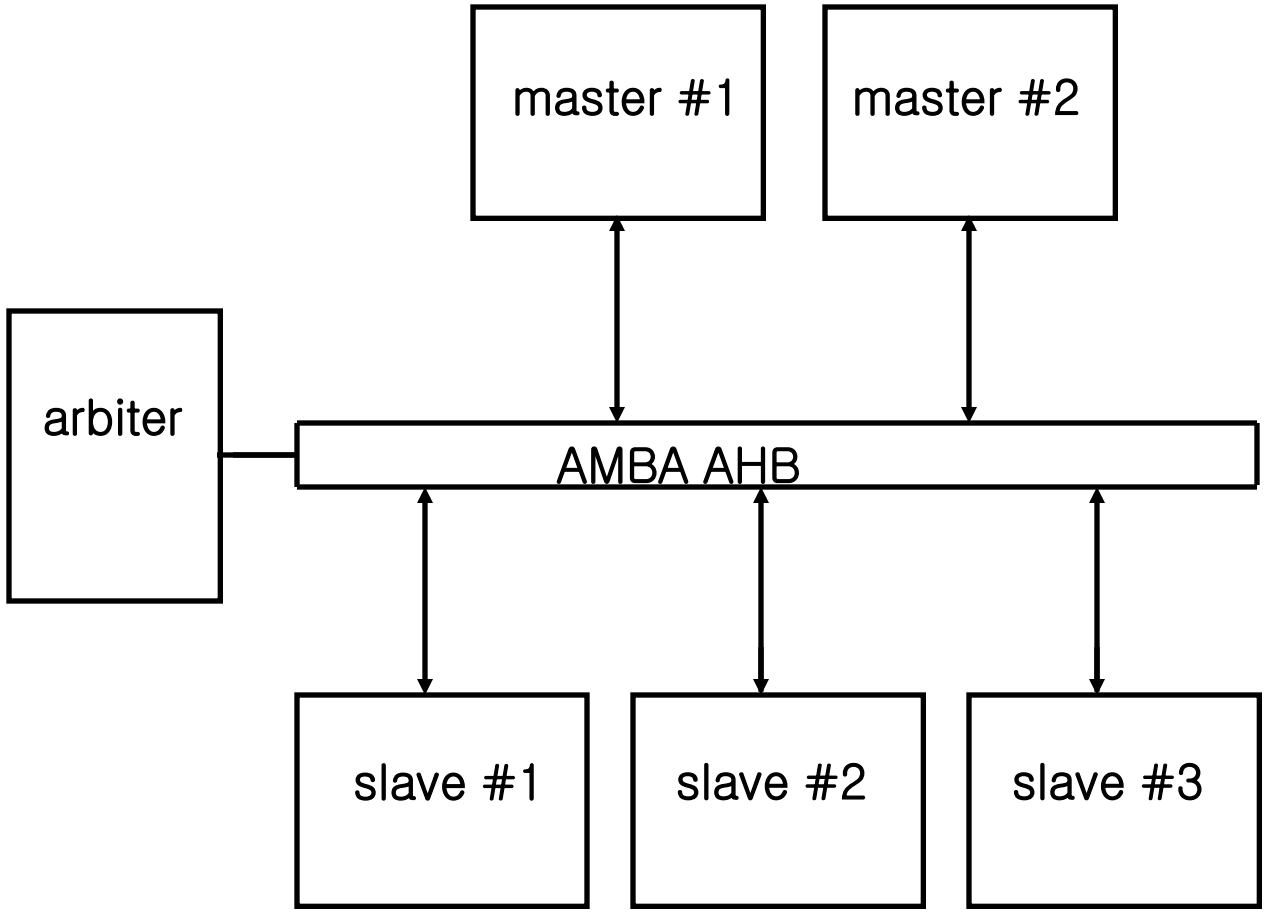
Importance of OCB for SOC

- Different components (IPs) may be developed by different vendors
- On-chip bus: interface between different vendors
- Functional test vector generation based on the bus protocol

AMBA 2.0

- Advanced Microprocessor Bus Architecture
- On-chip bus proposed by ARM
- Very simple protocol
- High bandwidth bus
  - AHB (Advanced High-performance Bus)
  - ASB (Advanced System Bus)
- Low bandwidth bus
  - APB (Advanced Peripheral Bus)

AMBA AHB Components



On-Chip Bus (OCB)

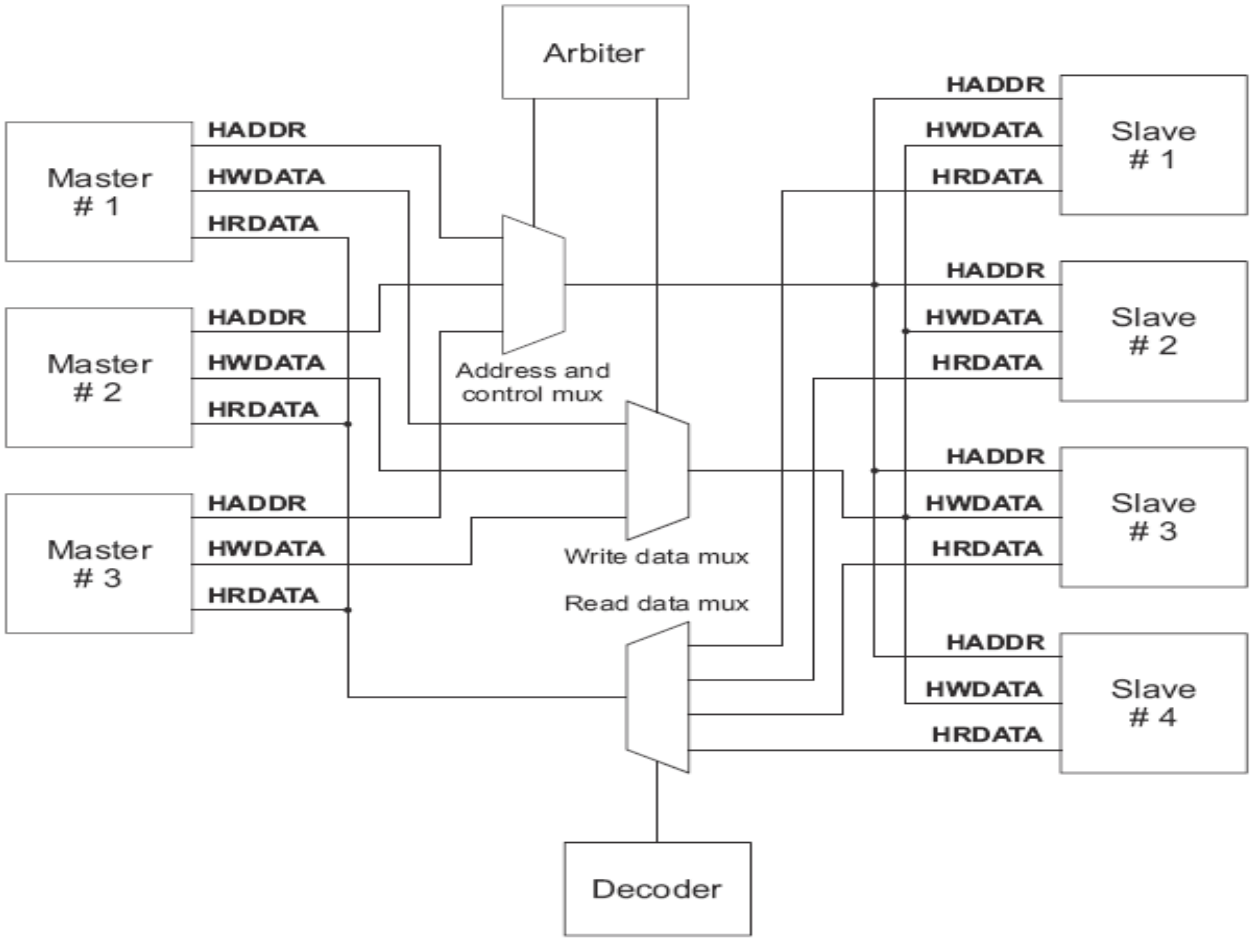


Figure 3-2 Multiplexor interconnection

AMBA AHB Features

- Pipelined transfer
- Burst transfers
- Split Txns (Transactions)
- Single cycle bus master handover
- Single clock edge operation
- Non-tristate implementation
- Wider data bus configurations (64/128 bits)

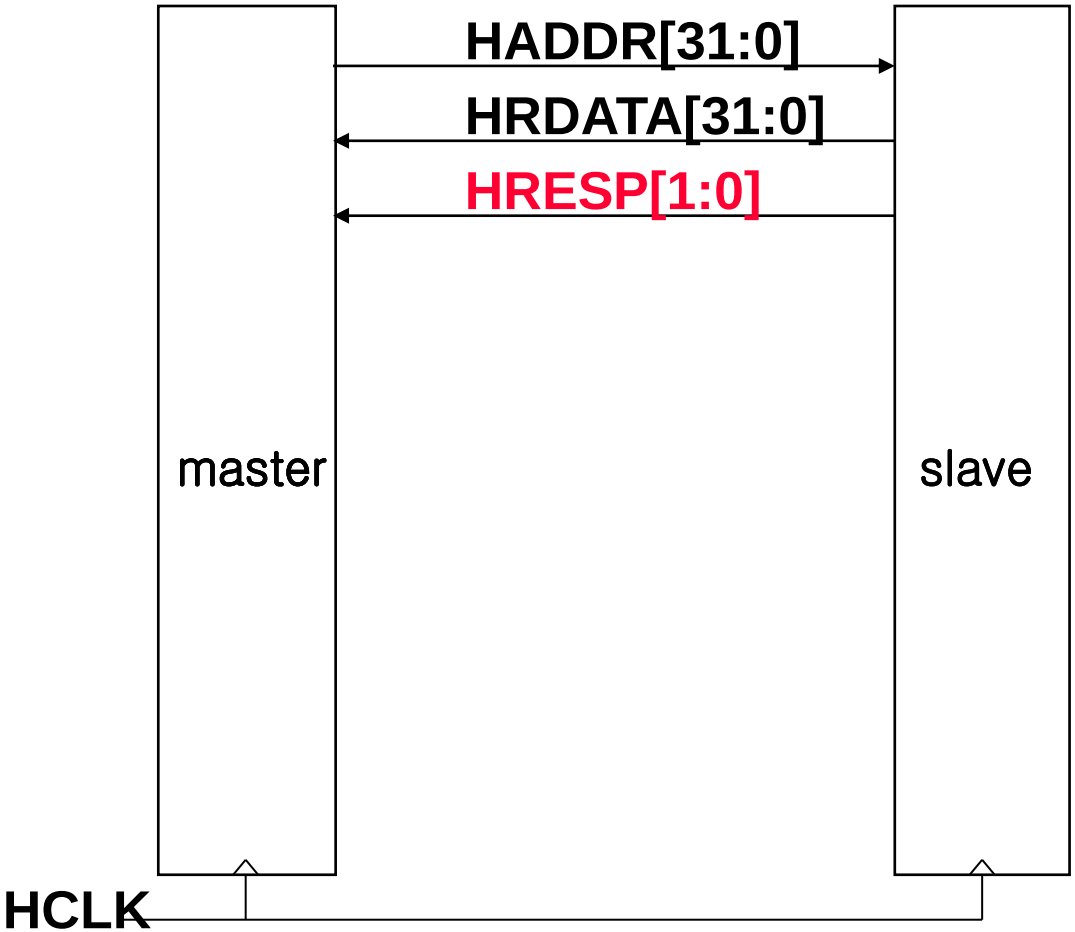
AHB - Operation

- Master sends a request signal to the Arbiter
- Arbiter grants the bus to the Master
- Master starts transfer by sending address and control signals and data
- Slave responds by sending the status signal
- Uses Write data bus for data transfer from Master to Slave
- Uses Read data bus for data transfer from Slave to Master

Each transfer

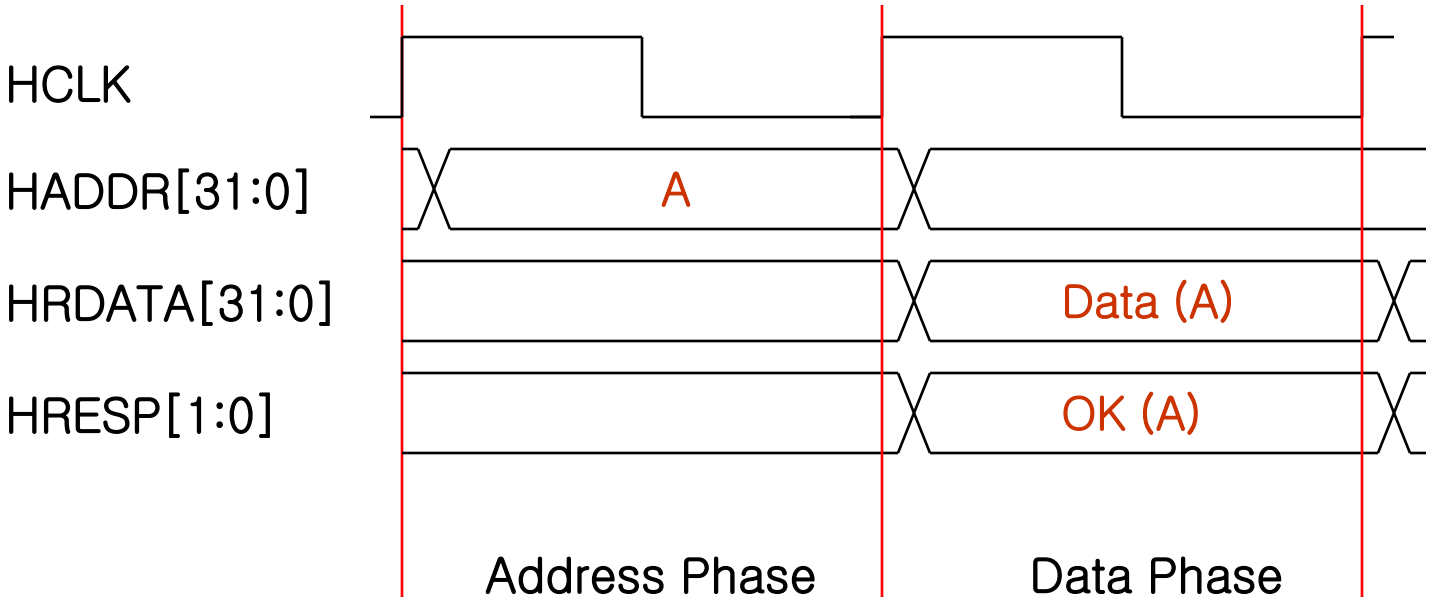
- Each transfer consists of
  - An address and control cycle
  - One or more cycles for the data
- Two forms of bursts
  - Incrementing bursts
  - Wrapping bursts
- The address cycle cannot be extended
- The data cycle can be extended
  - Using HREADY signal.

Basic Signals for Read Txn



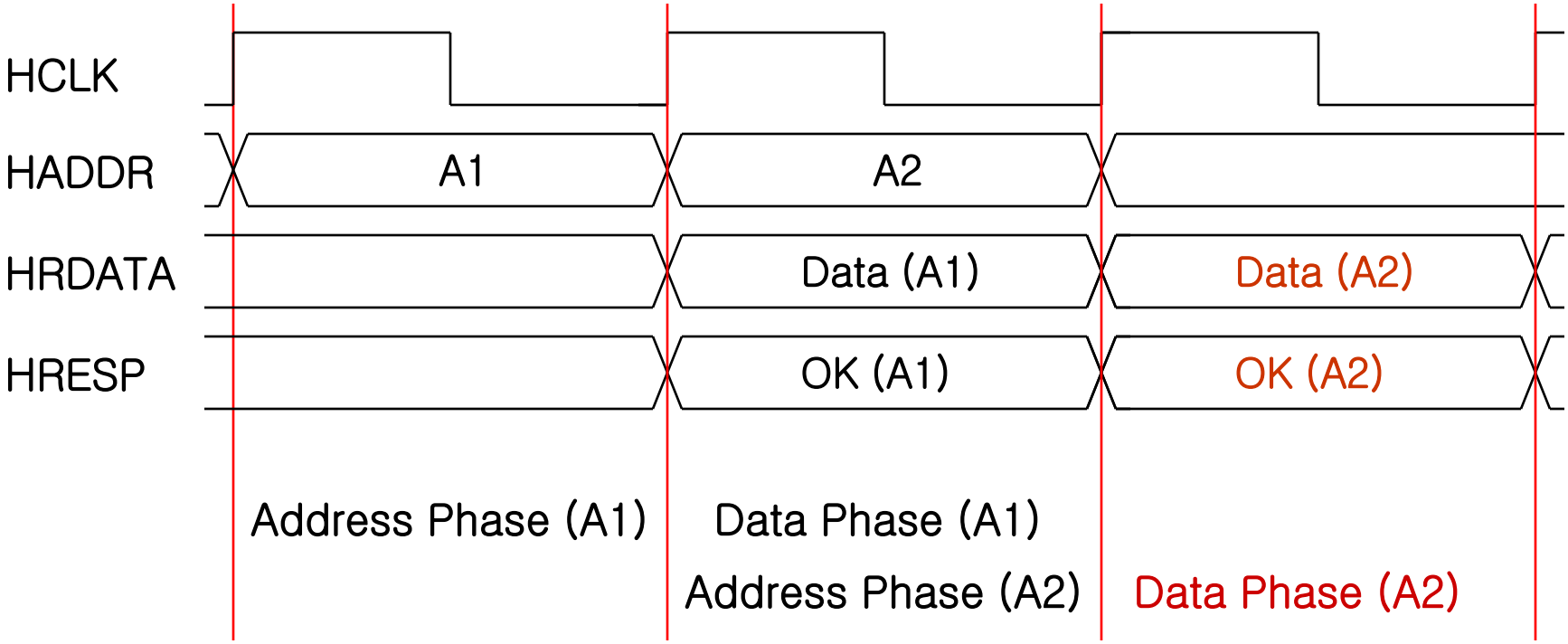
HRESP[1:0]: transfer response
- OKAY,
- ERROR,
- RETRY,
- SPLIT

Operation for a Read Txn

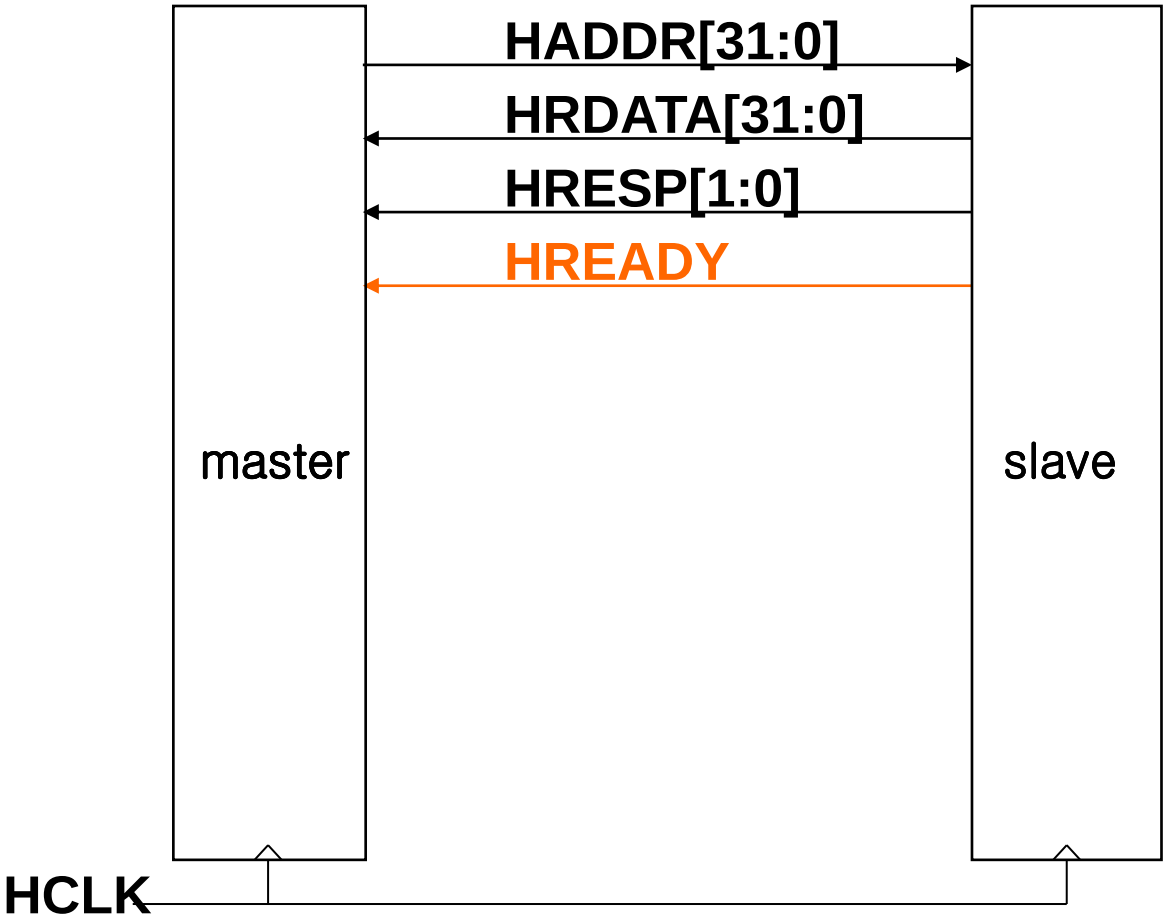


OKAY: The transfer is normal. When READY goes high, the transfer has completed successfully.

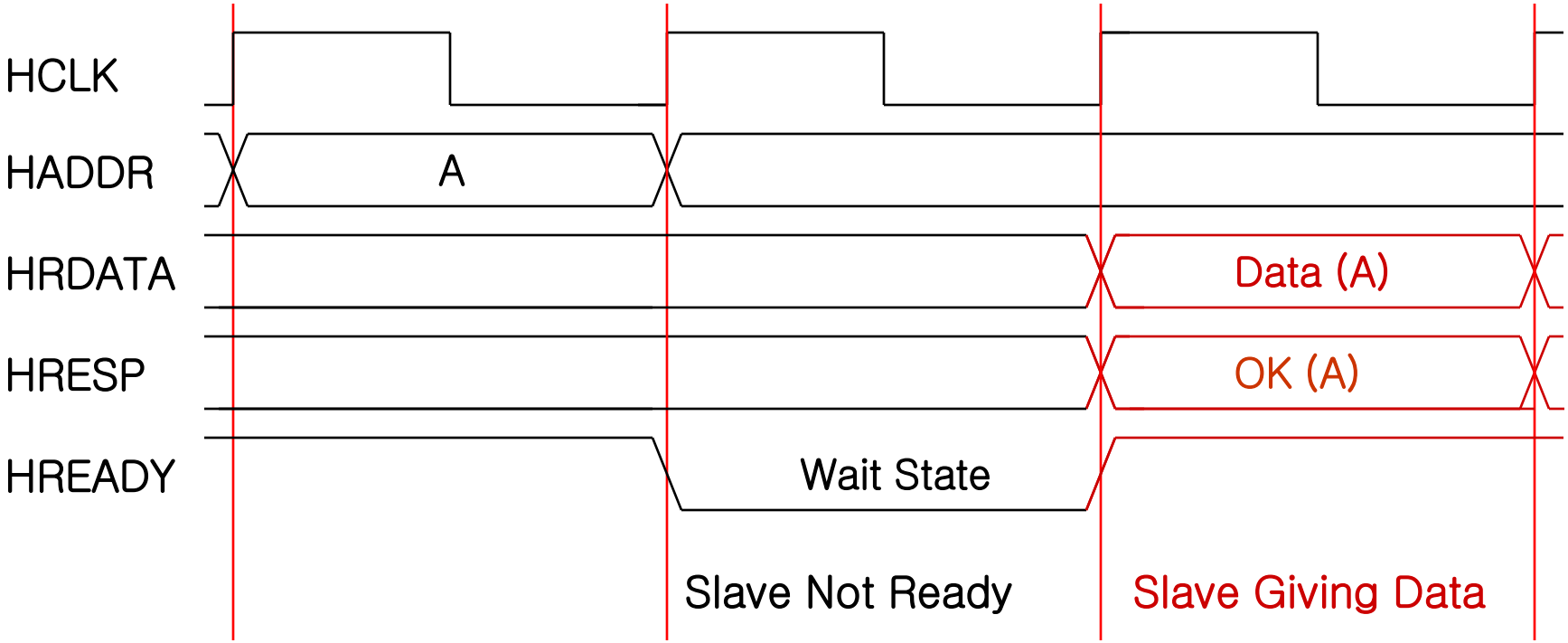
Pipelined Operation



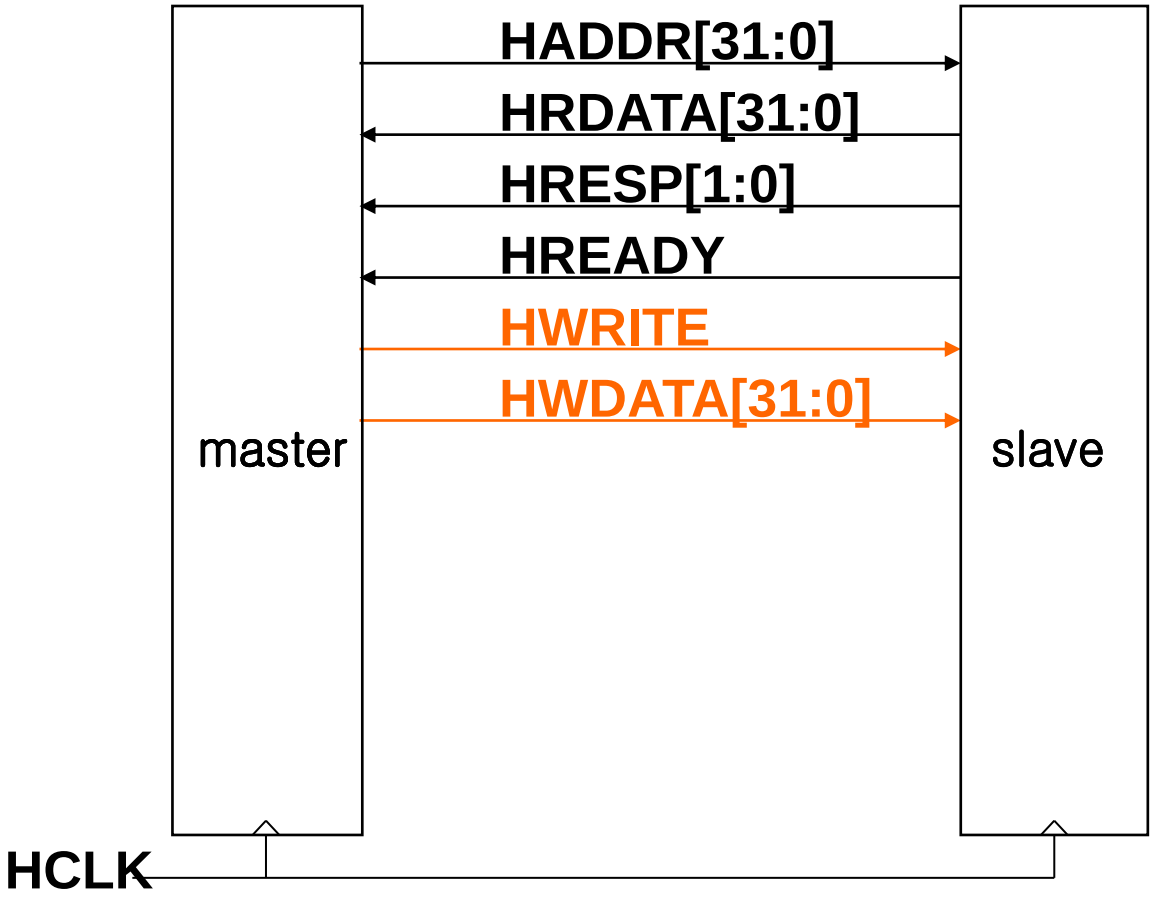
HREADY for a Slow Slave



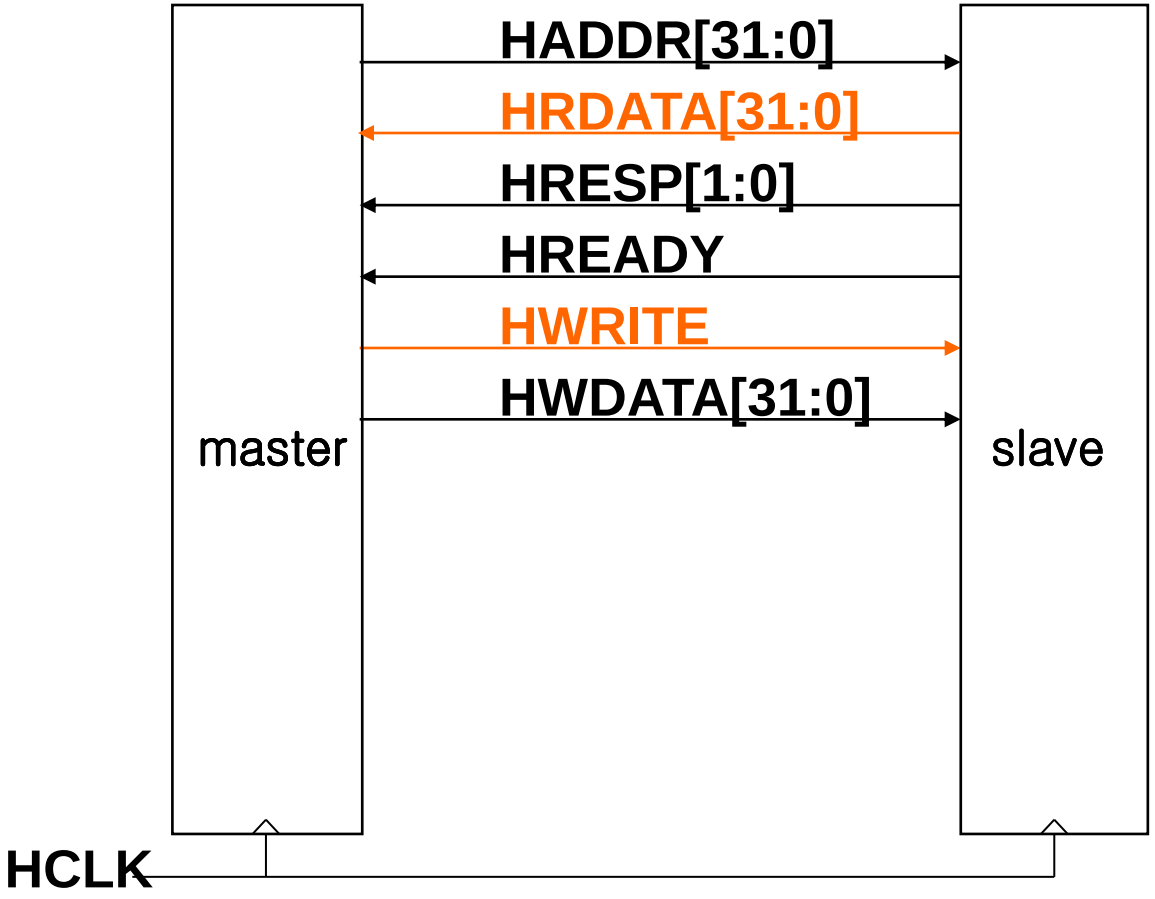
Wait State Insertion



HWRITE/HWDATA for a Write Txn

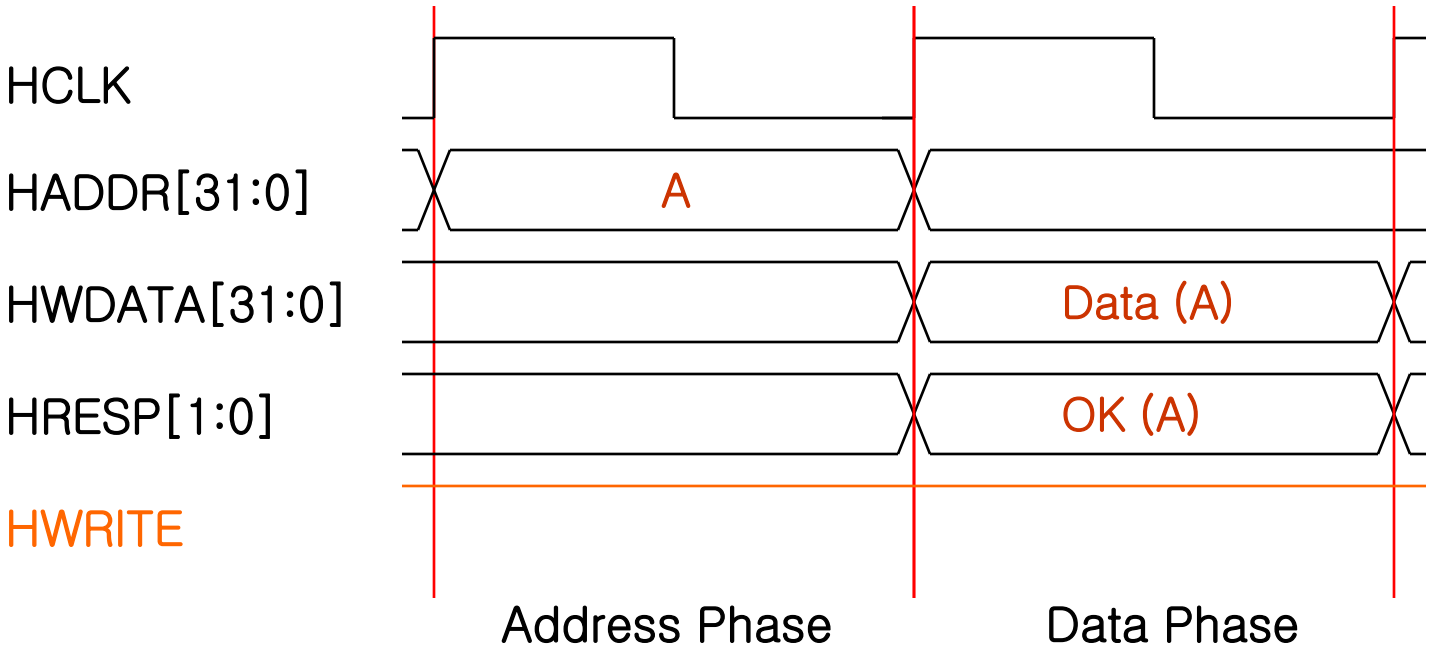


HWRITE/HRDATA for a Read Txn

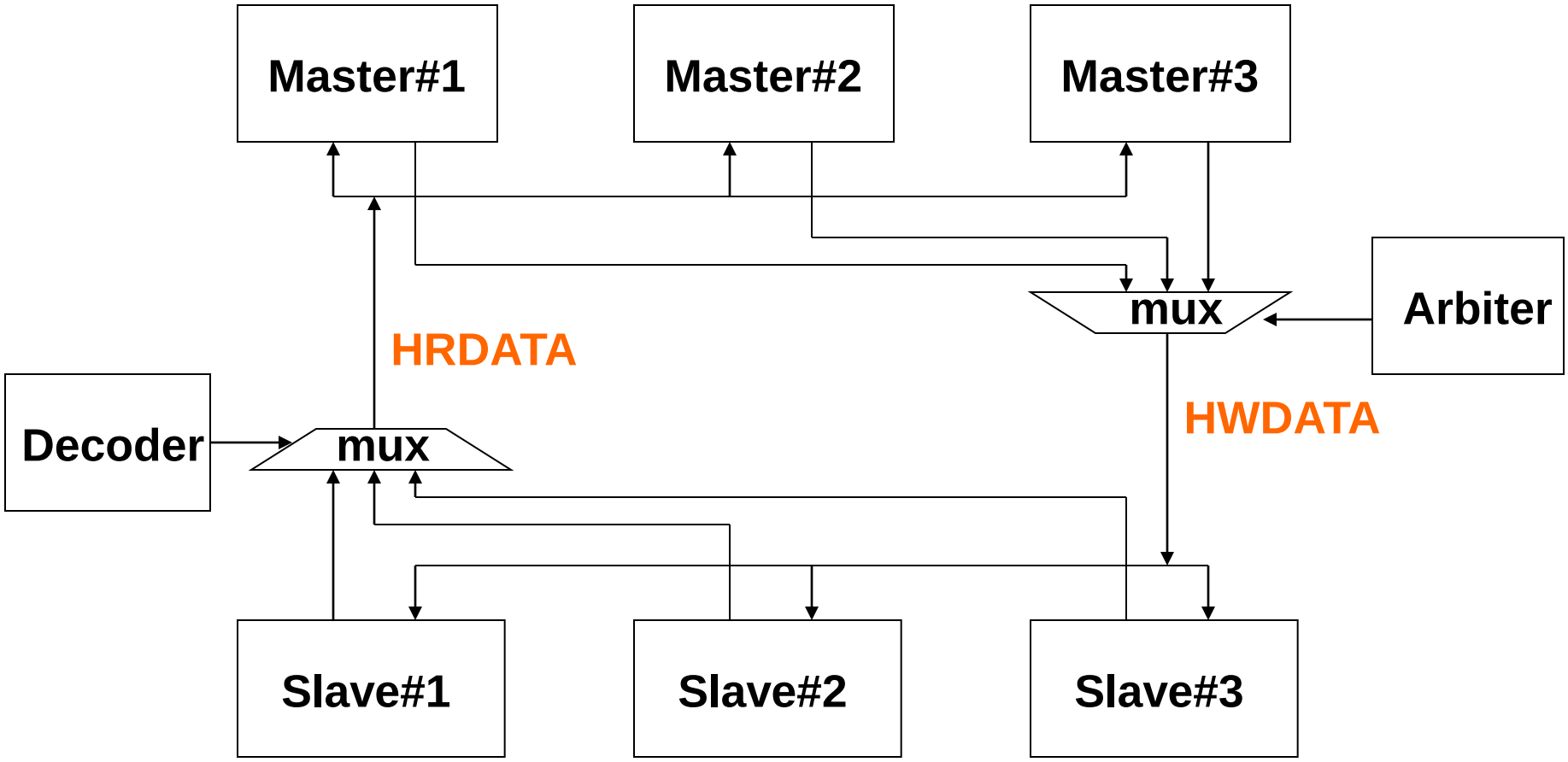


HWRITE: a read transfer when low

Operation for a Write Txn



Interconnection of Data Buses

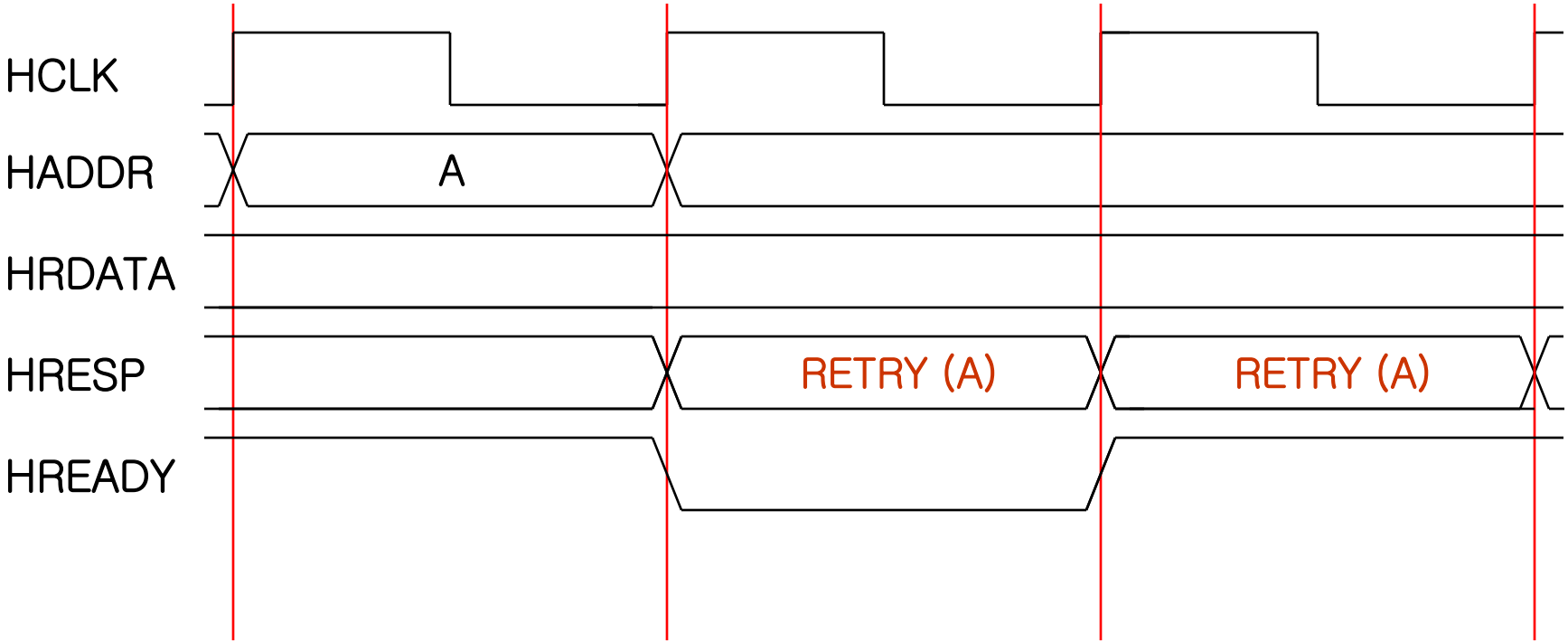


Response Type

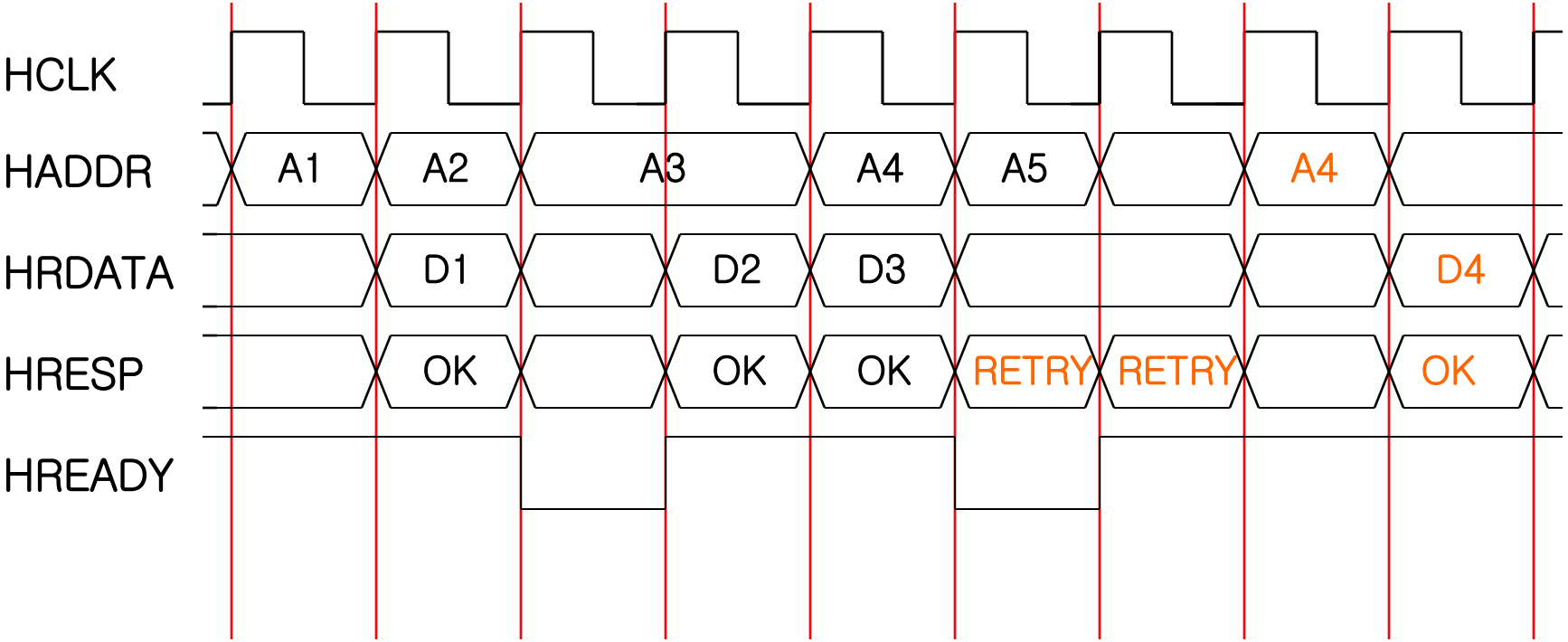
HRESP[1:0]	Response	Description
00	OKAY	Transaction Completed
01	ERROR	Error Occurs
10	RETRY	Transaction Not Completed Master Must Retry
11	SPLIT	Transaction Not Completed Master Must Retry Slave Informs Completion

ERROR/RETRY/SPLIT: requires at least two cycle response

Two Cycle Response

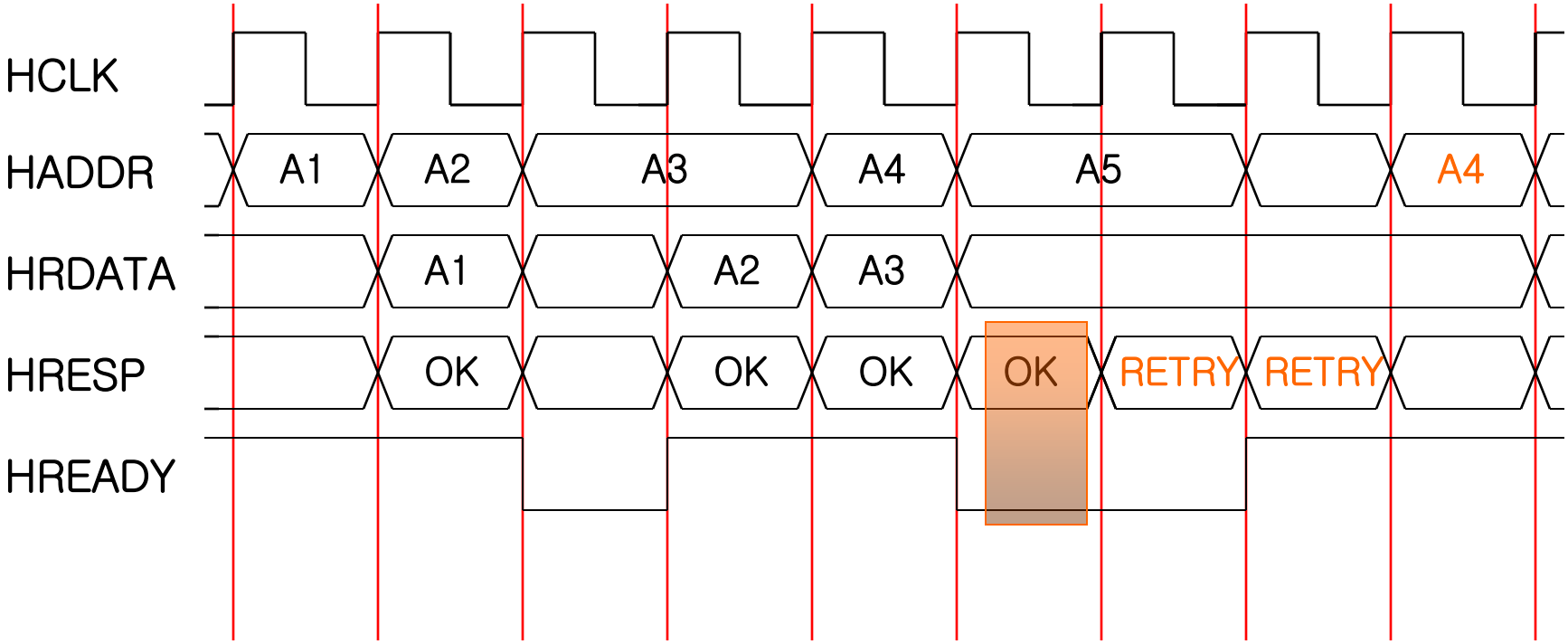


Timing Diagram Practice



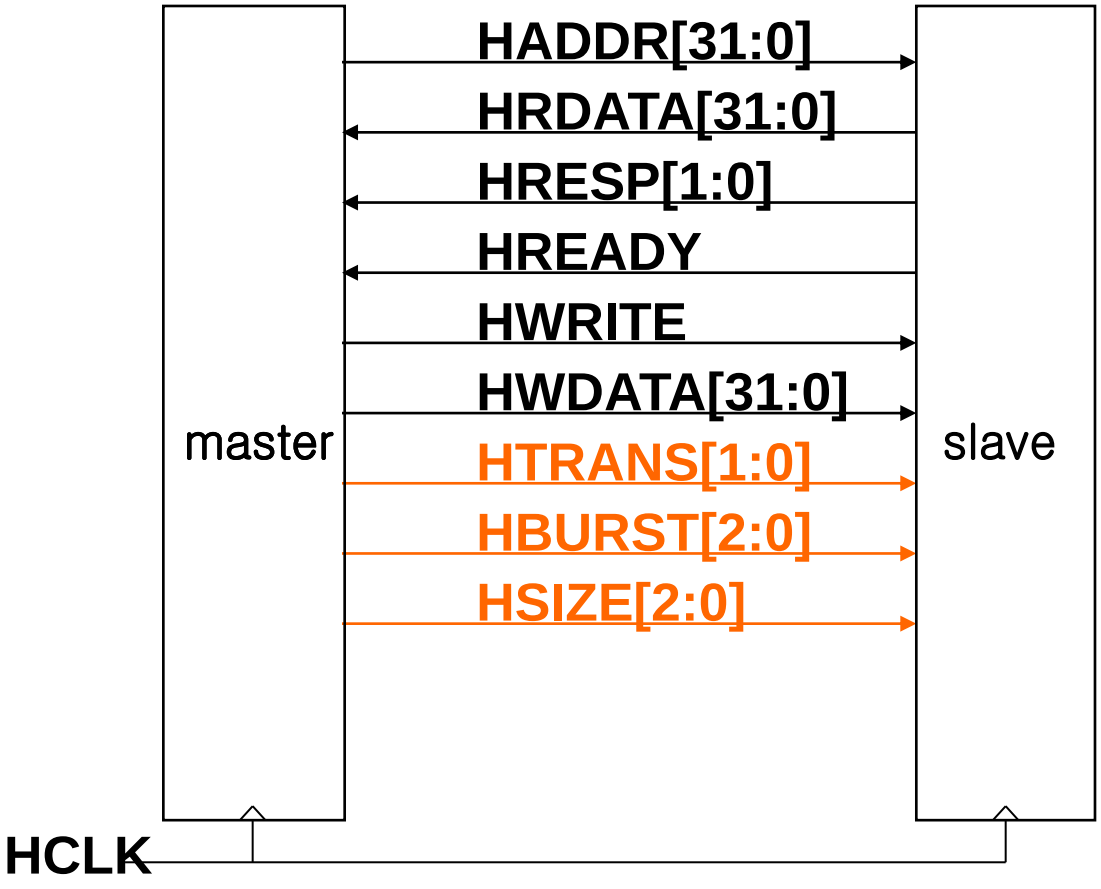
The tow cycle response allows sufficient time for the master
to cancel the address already broadcasted and drive HTRANS[1:0] to IDLE
Before the start of the next transfer

Timing Diagram Practice



If the slave need more than two cycles to provide the ERROR, SPLIT or RERTY response, then additional wait state may be inserted at the start of the transfer.

Burst Operation



HTRANS[1:0]: transfer types
IDLE,
BUSY,
NONSEQ,
SEQ

HBURST[2:0]: burst types
Incrementing bursts,
wrapping bursts

HSIZE[2:0]: transfer size
8×2^n
8, 16, 32, 64, 128,
256, 512, 1024

Transfer Types

HTRANS[1:0]	Type	Description
00	IDLE	No data transfer required Requires zero wait state OKAY response
01	BUSY	Same as IDLE in the middle of burst transfers
10	NONSEQ	Single transfer or the first of a burst Address/Control unrelated previous to the previous transfer
11	SEQ	Remaining transfers in a burst Address/control related to the previous transfer

Burst Modes

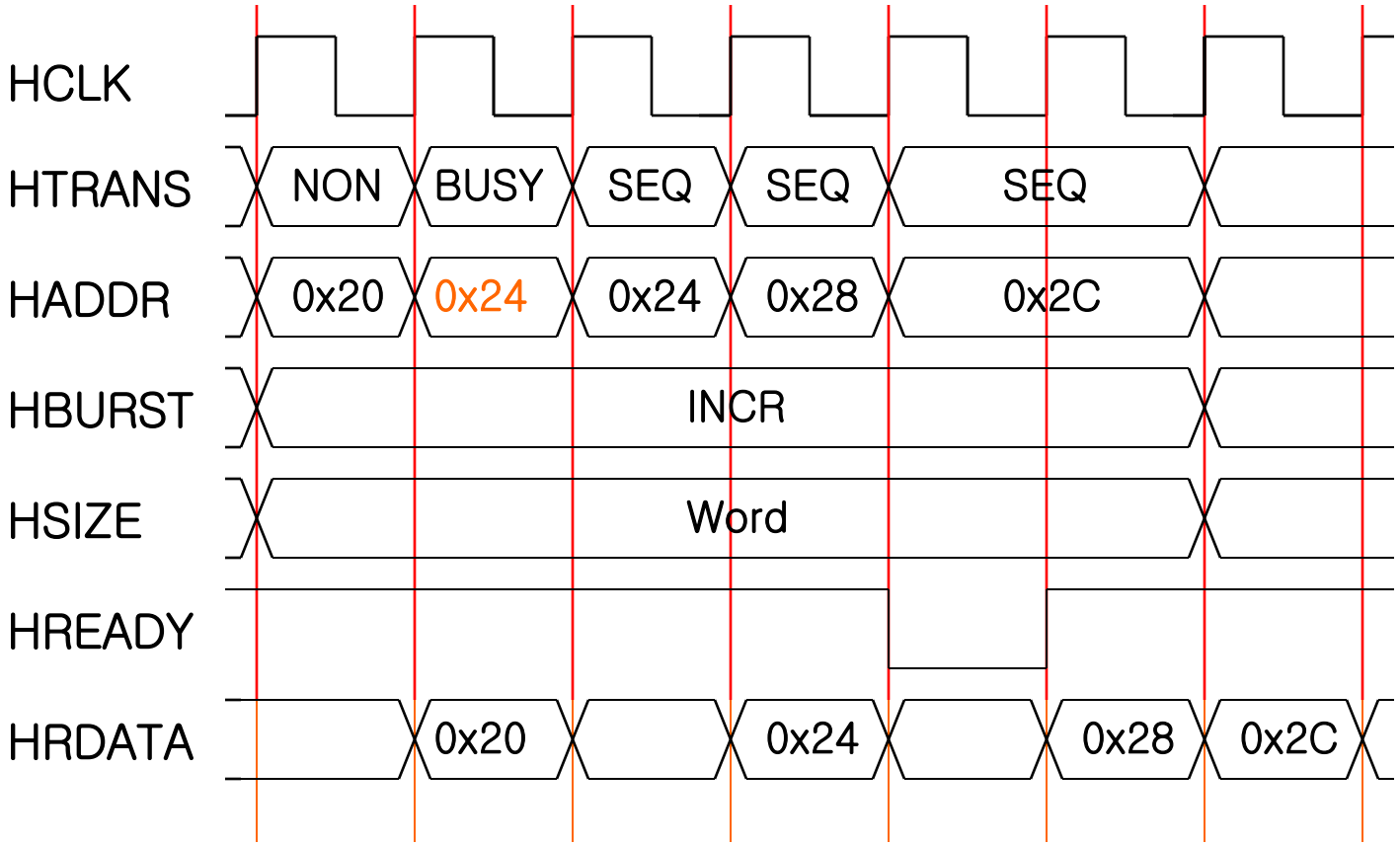
HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Burst cannot cross a 1KB address boundary.

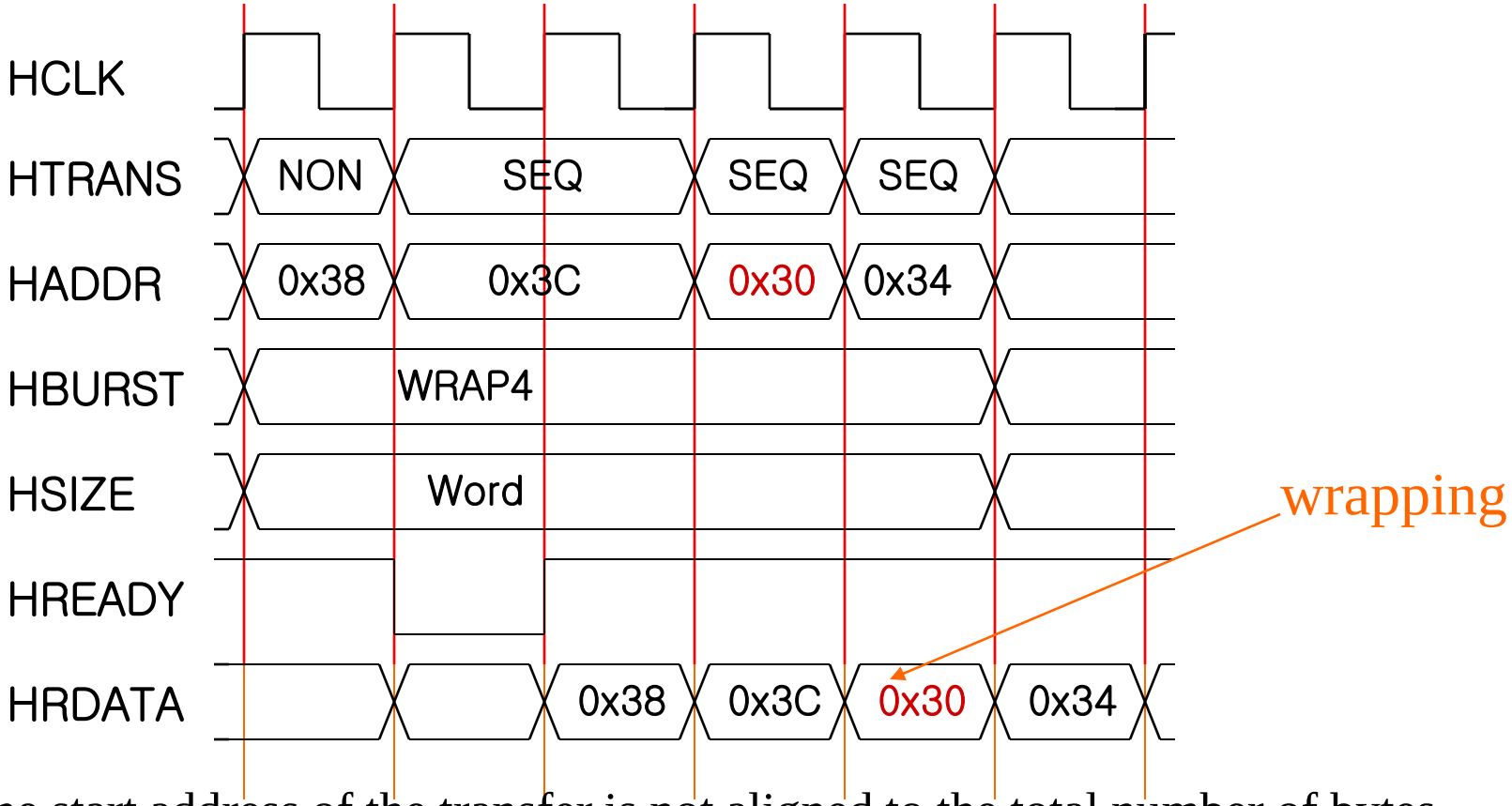
Transfer Sizes

HSIZE[2:0]	Size	Description
000	8 bits	Byte
001	16 bits	Halfword
010	32 bits	Word
011	64 bits	-
100	128 bits	4-word line
101	256 bits	8-word line
110	512 bits	-
111	1024 bits	-

Transfer Type Examples

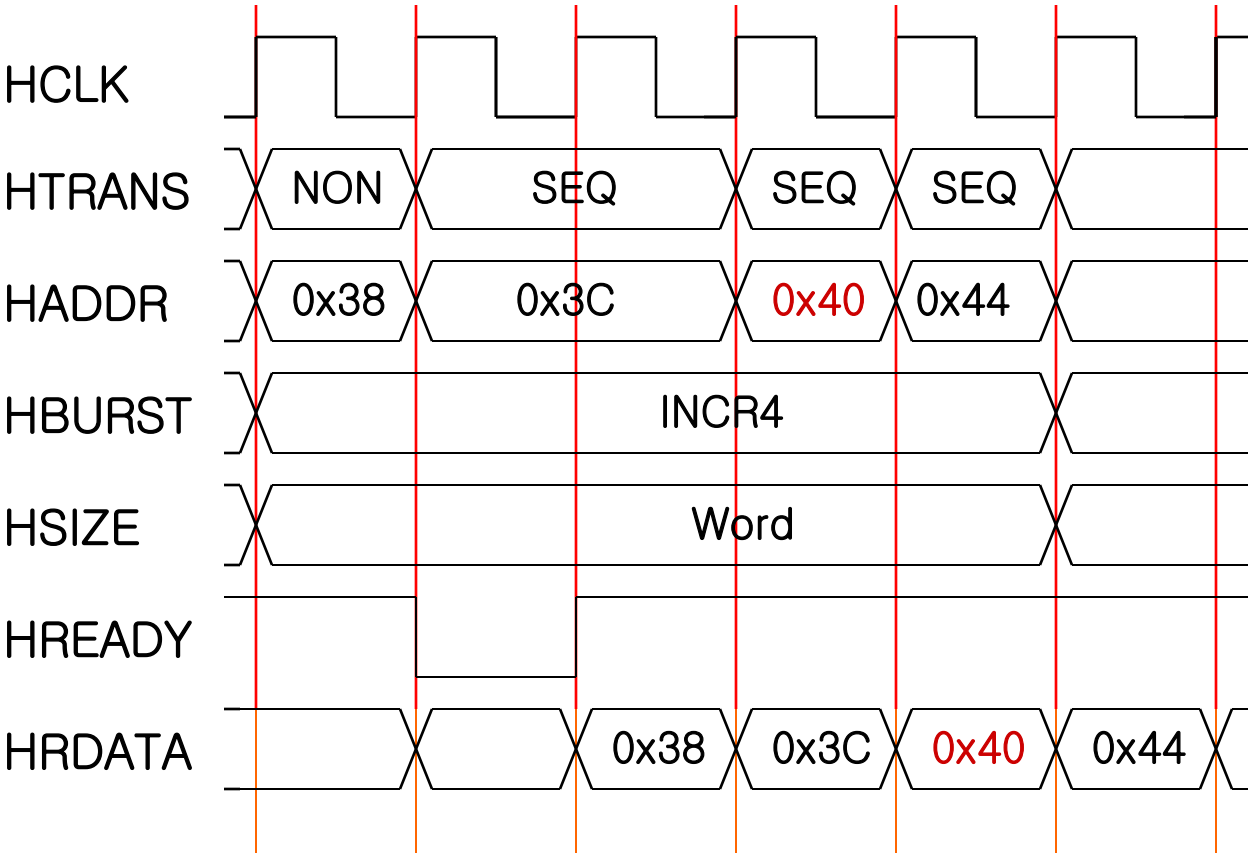


Four-beat Wrapping Burst

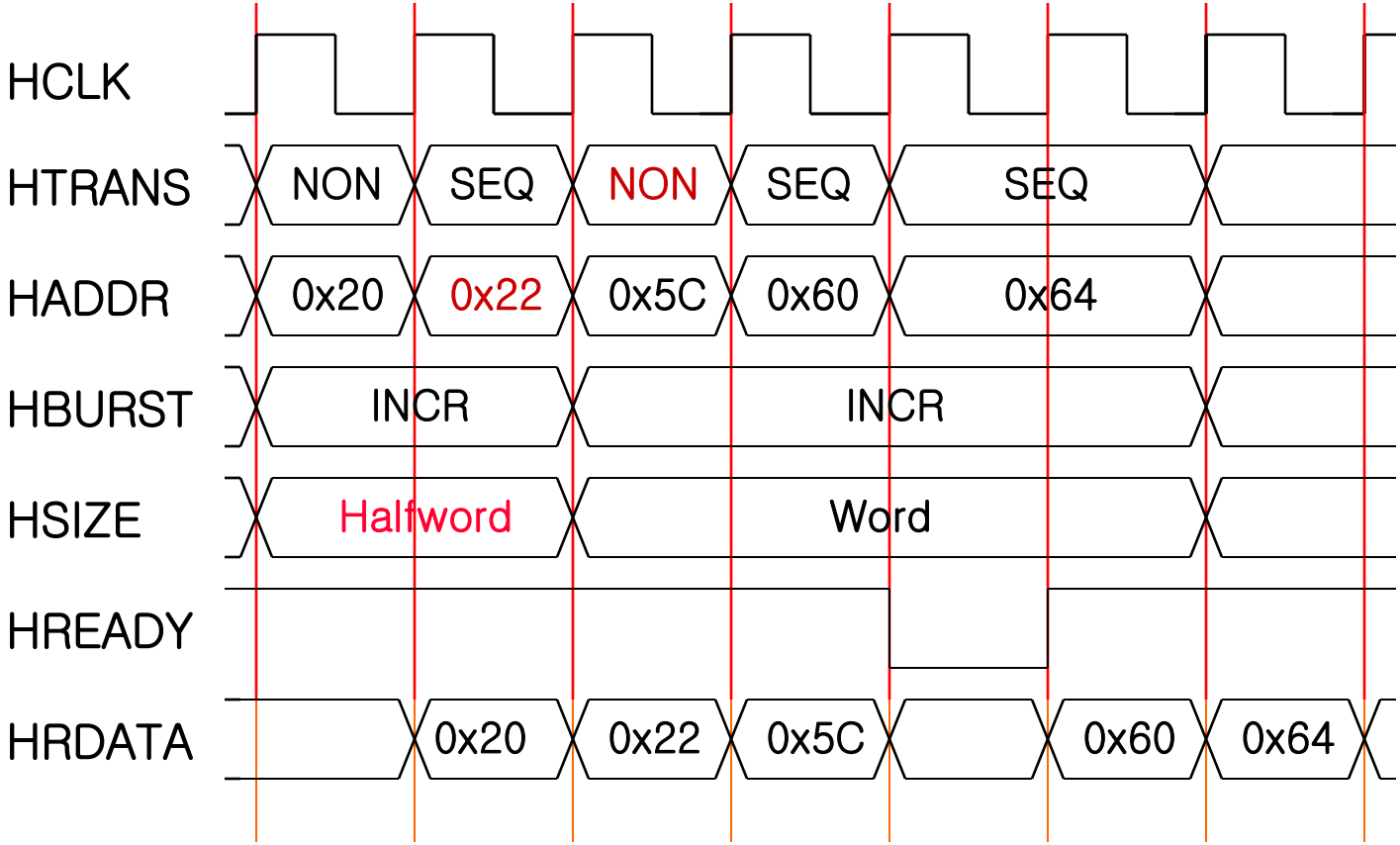


If the start address of the transfer is not aligned to the total number of bytes (size x beats) then the address of the transfers in the burst will wrap when The boundary is reached.

Four-beat Incrementing Burst



Undefined-length Bursts



Address decoding

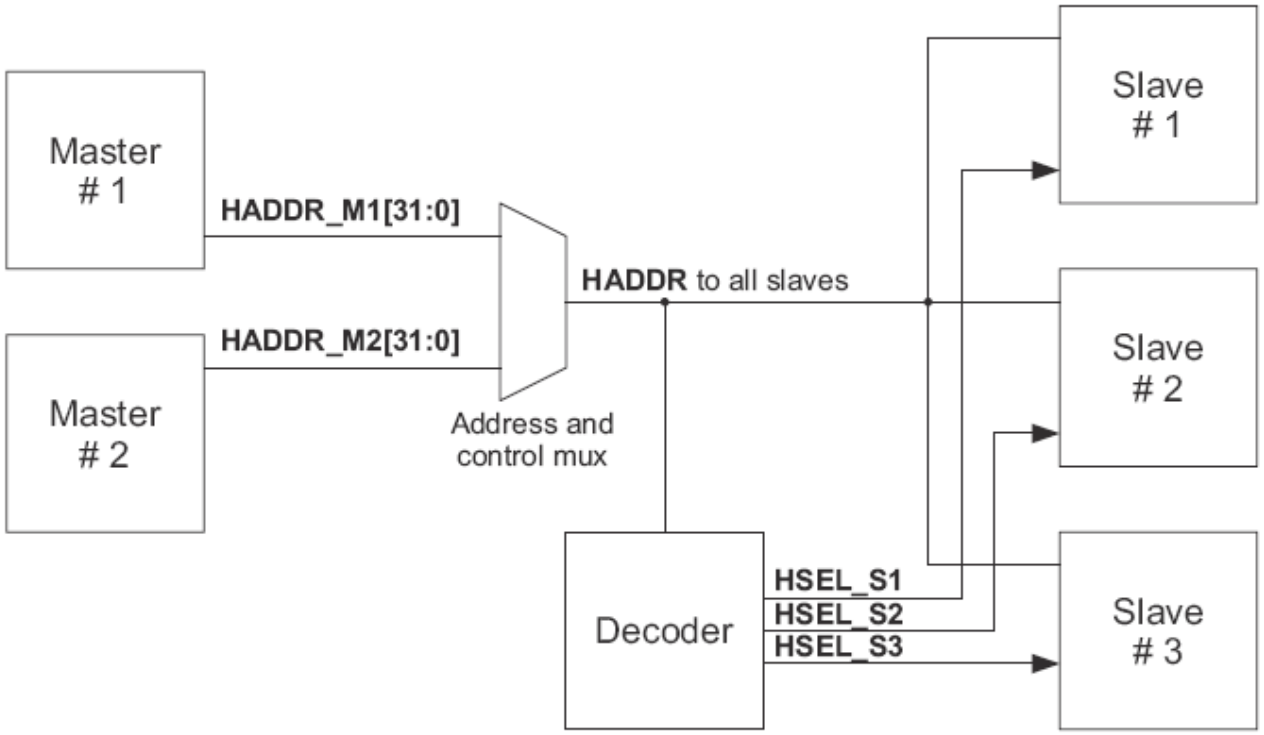
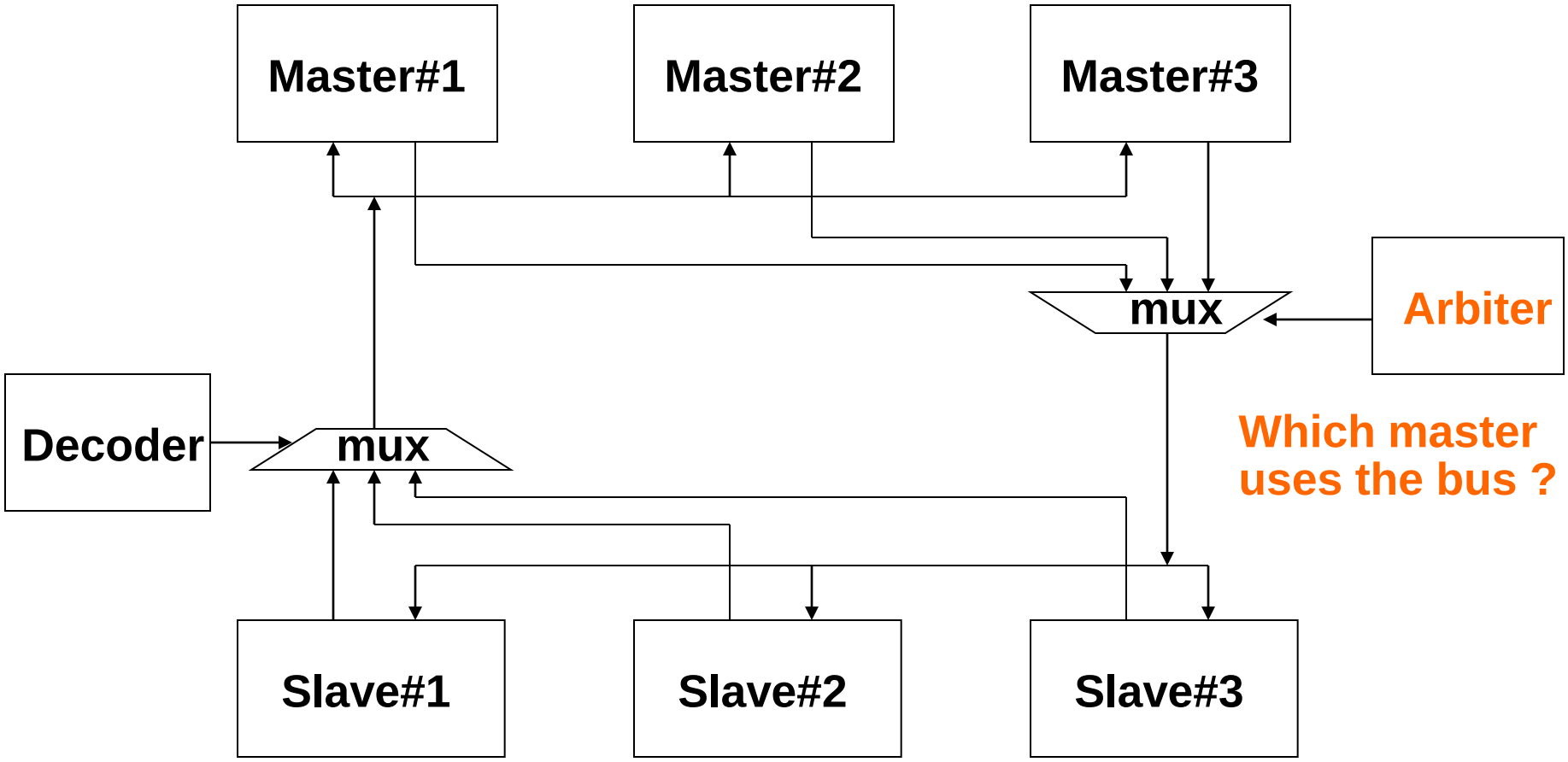
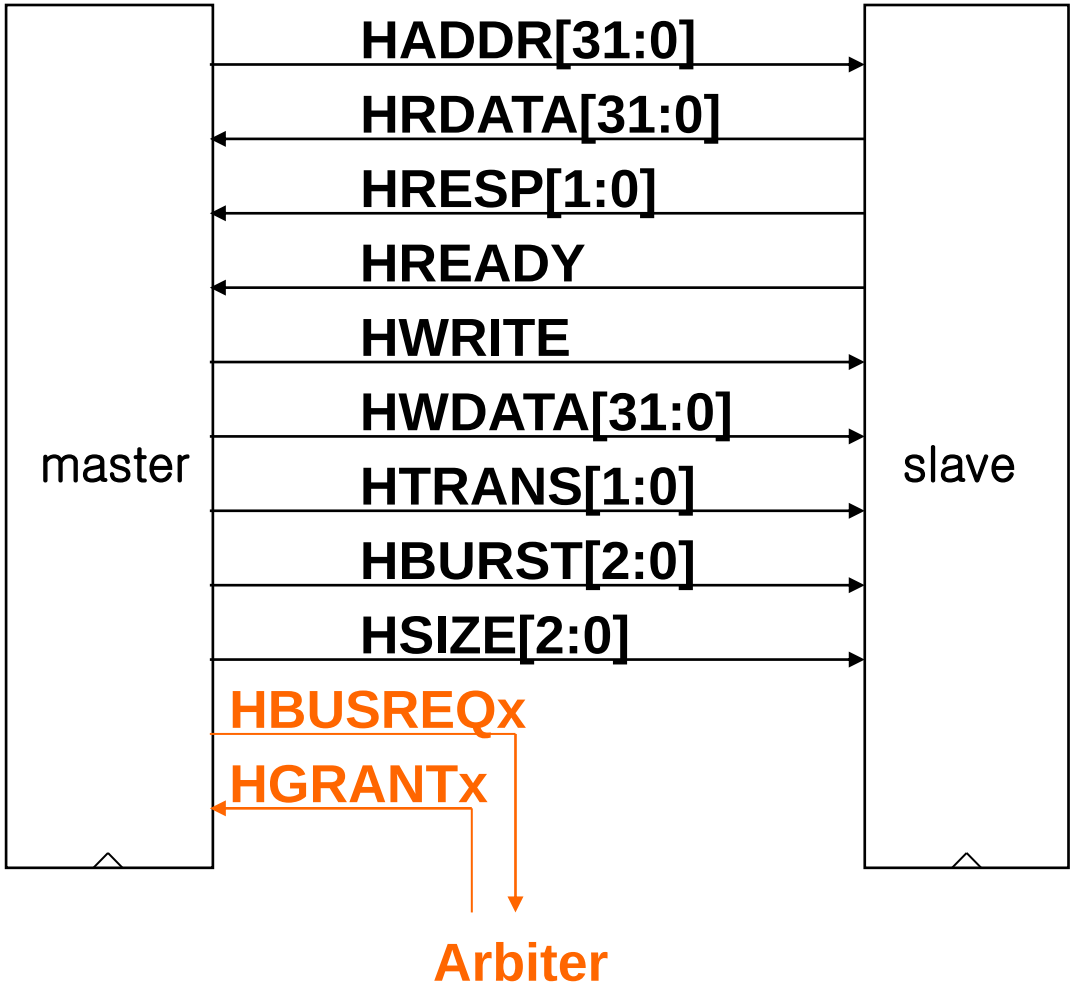


Figure 3-12 Slave select signals

Arbitration



Arbitration Signals



Arbitration signals

- Masters
  - HBUSREQx: up to 16 separate bus masters
  - HLOCKx: request locking during burst transactions
- Arbiter
  - HGRANTx:
  - HMASTER[3:0]: granted master ID
    - Used by a MUX that selects a granted master
    - Also used by split-capable slaves
  - HMASTERLOCK: indicate that the current transfer is part of the locked sequence
- Slaves
  - HSPLIT[15:0]: indicate which master can complete a SPLIT transaction

Arbitration Phase

- A bus master uses the HBUSREQx signal to request access to the bus.
- The arbiter will sample the request on the rising edge of the clock and then … decide which master will be the next to gain access to the bus
- A master gains ownership of the address bus when HGRANTx is HIGH and HREADY is HIGH at the rising edge of HCLK.

Bus Master grant signals

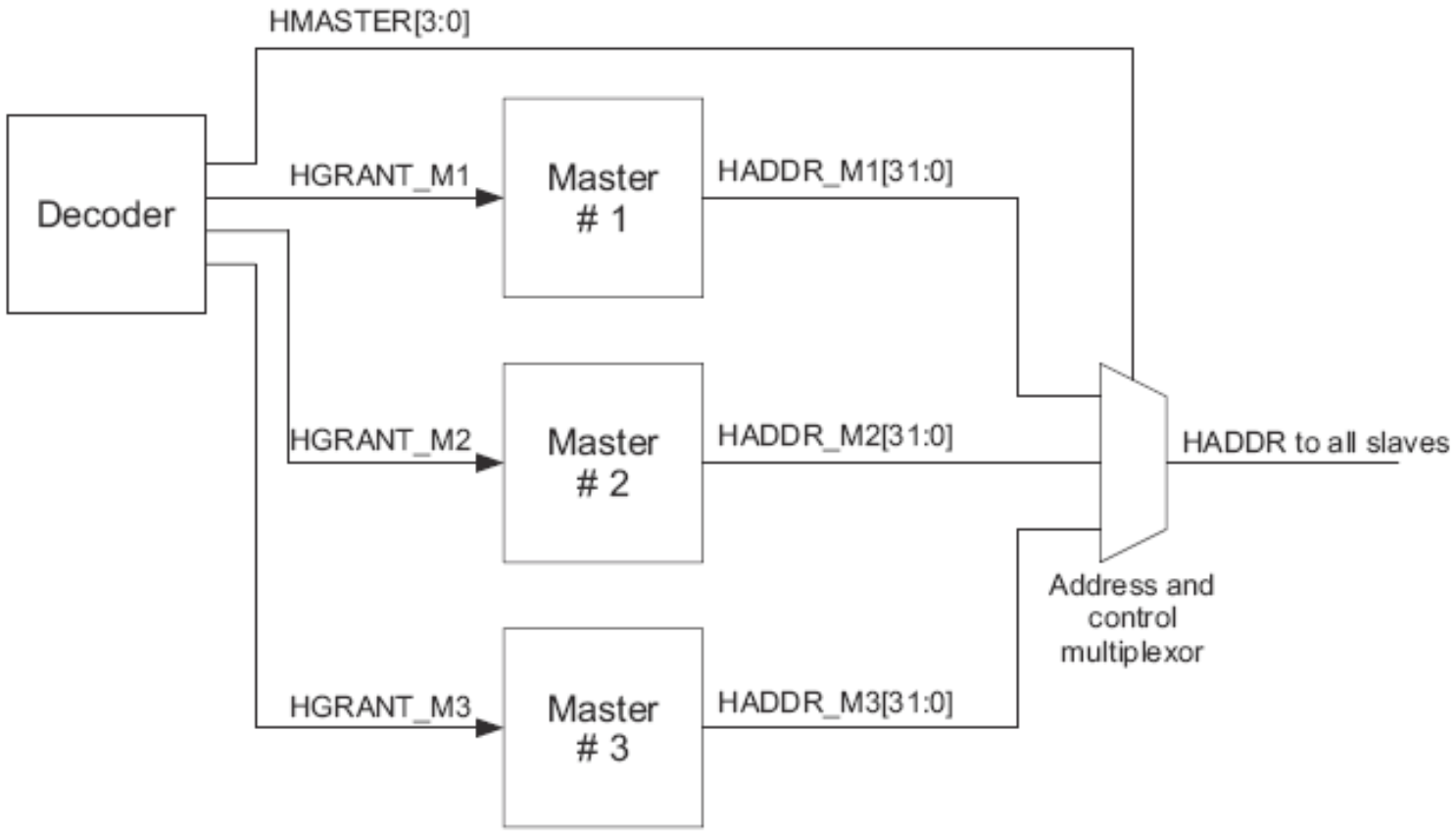
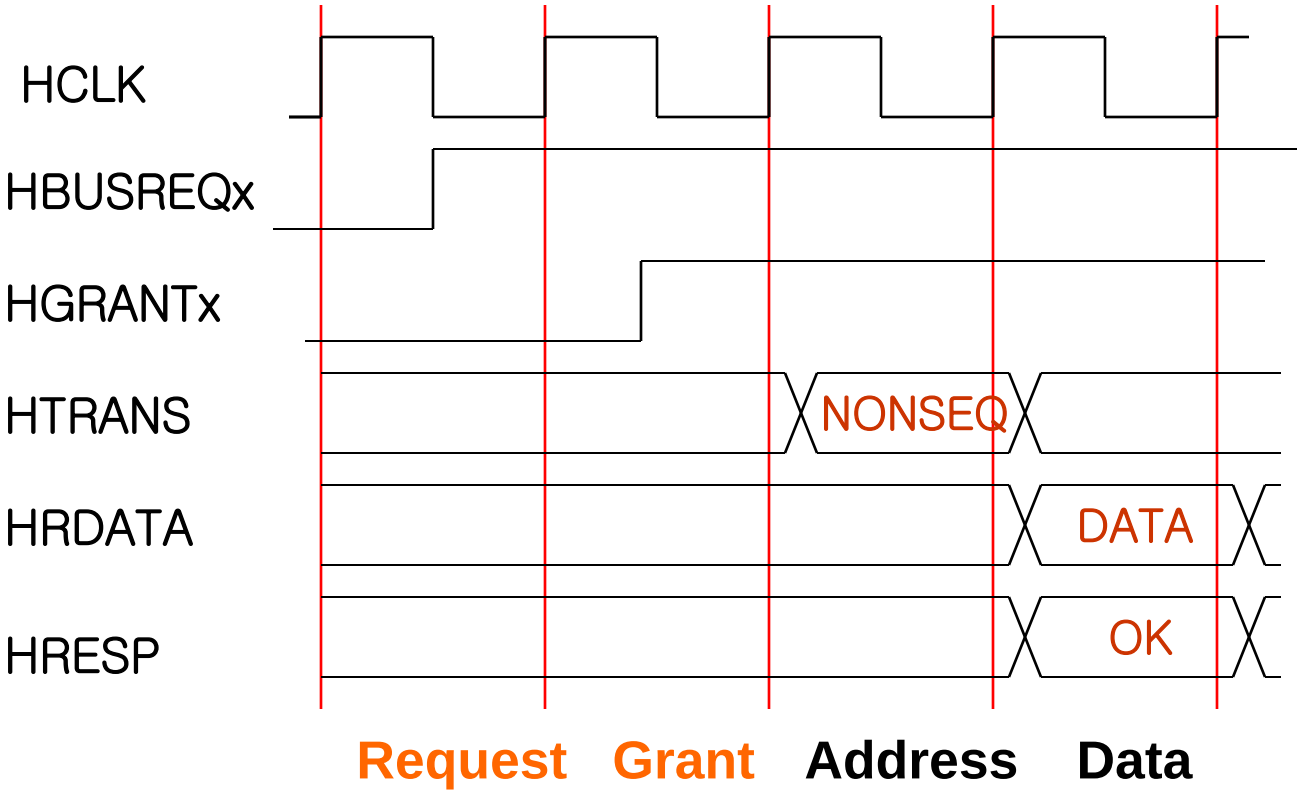


Figure 3-19 Bus master grant signals

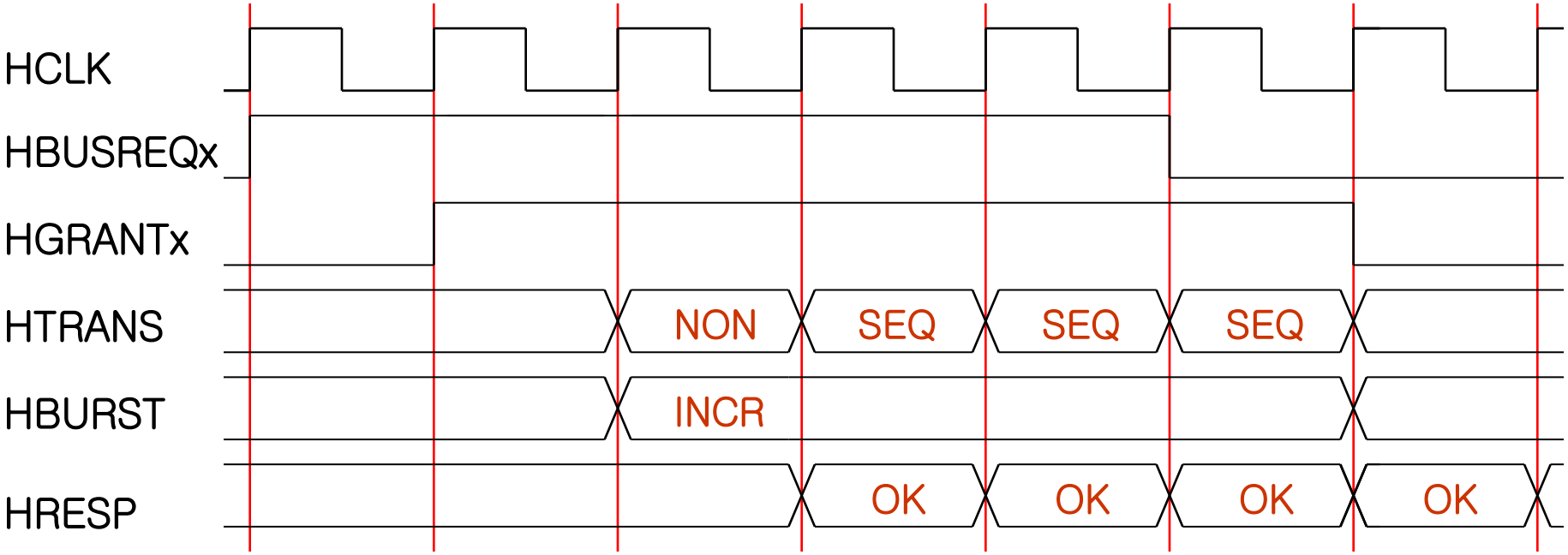
Arbitration Phases



Undefined Length Burst

- For undefined length bursts the master should continue to assert the request until it has started the last transfer.
- The arbiter cannot predict when to change the arbitration at the end of an undefined length burst.

Undefined Length Burst

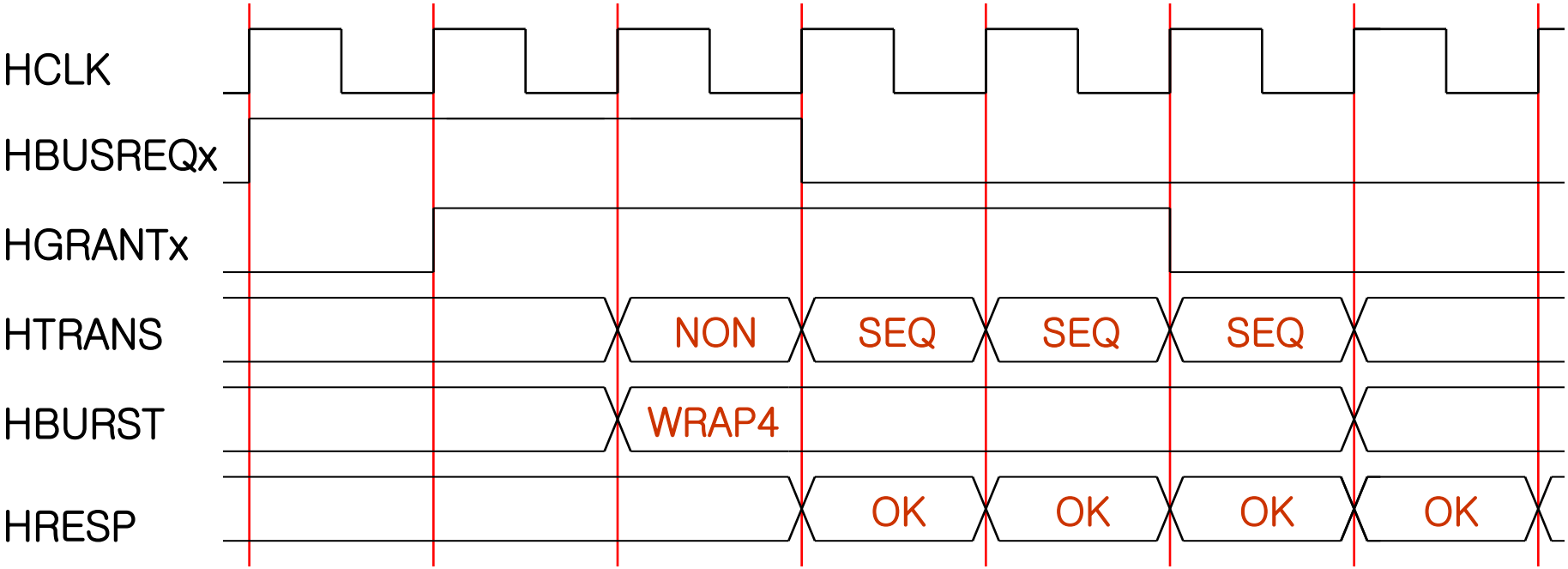


For undefined length burst the master should continue to assert the request until it has started the last transfer.

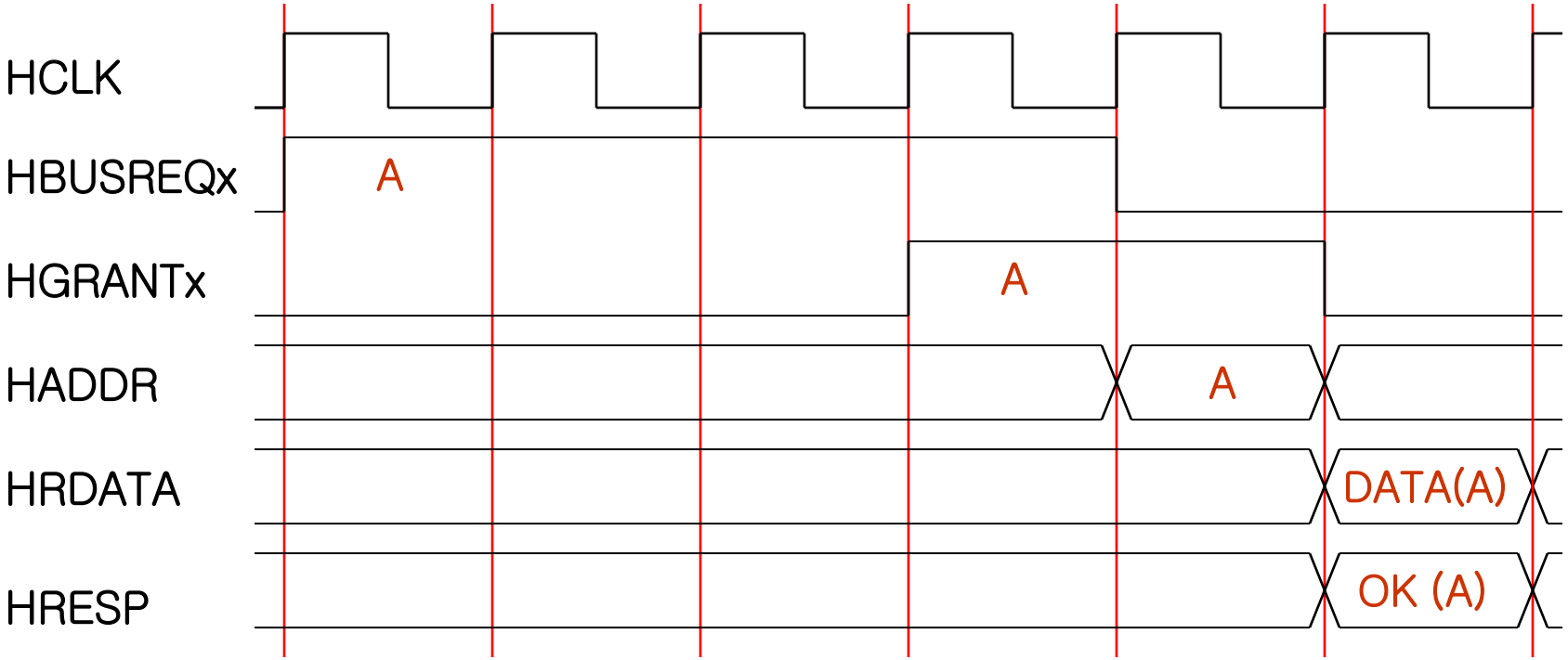
Fixed Length Burst

- When a master is granted the bus and is performing a fixed length burst it is not necessary to continue to request the bus in order to complete the burst.
- The arbiter observes the progress of the burst and uses the HBURST[2:0] signals to determine how many transfers are required by the master.
- Normally the arbiter will only grant a different bus master when a burst is completing. However, if required, the arbiter can terminate a burst early to allow a higher priority master access to the bus.

Fixed Length Burst



Arbitration Example: Slow Grant



Arbitration Example: Slow Grant

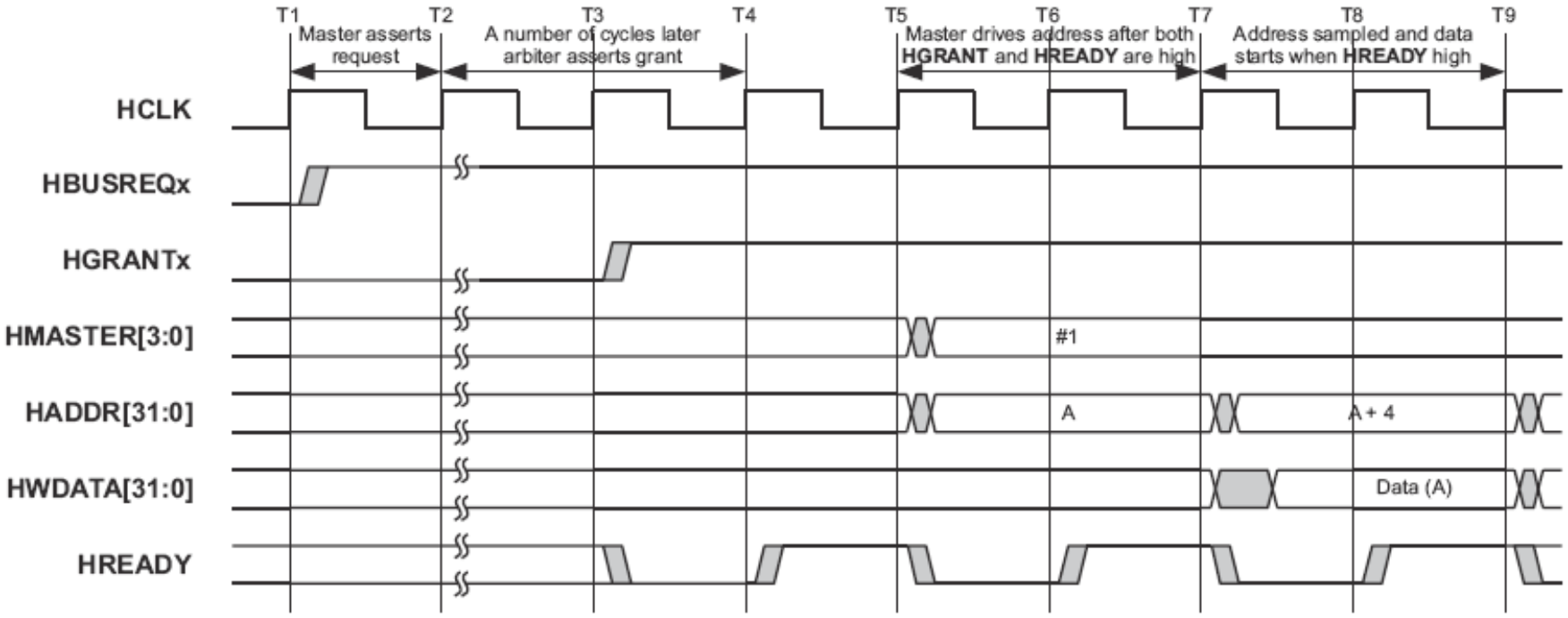
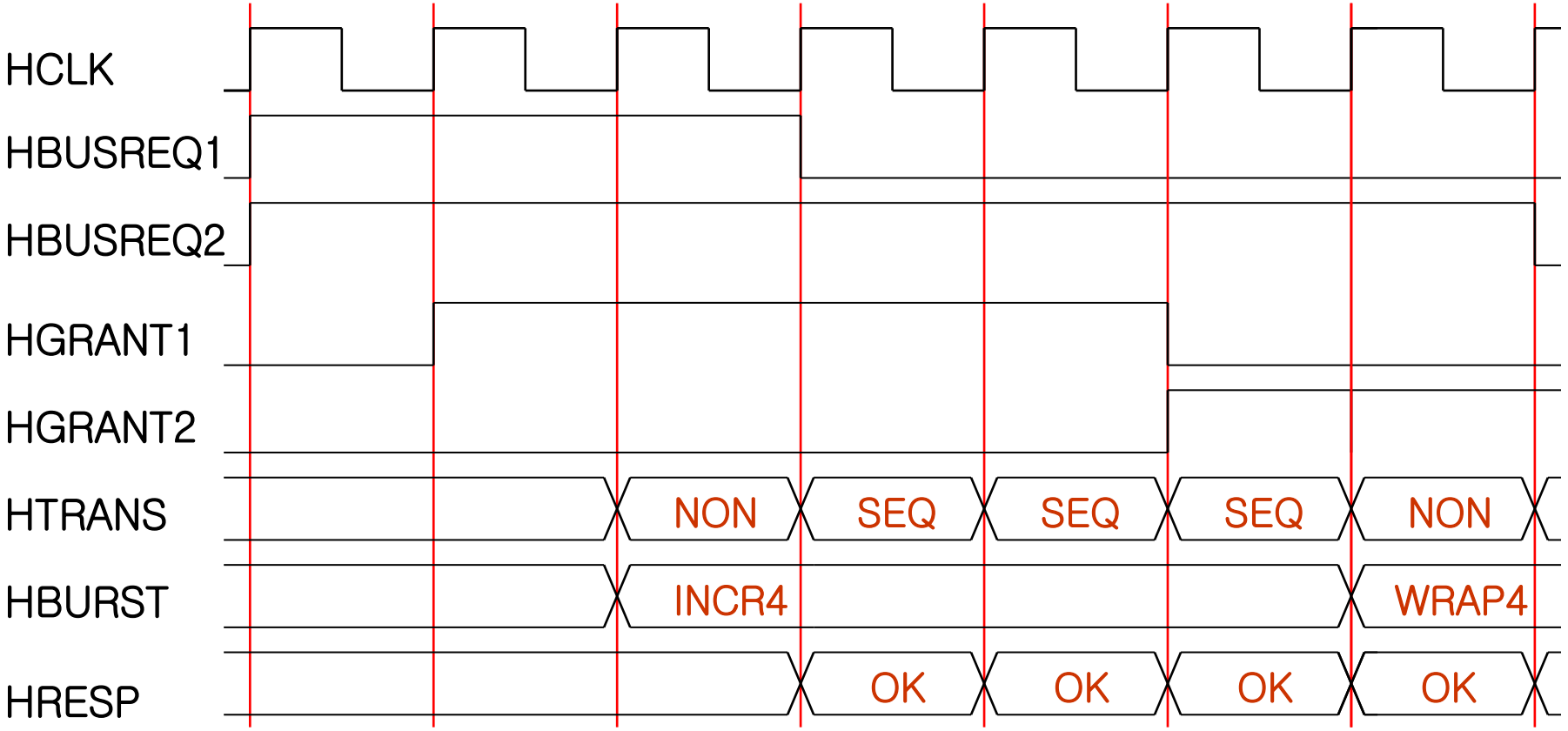


Figure 3-16 Granting access with wait states

Arbitration Example: Two Masters



Arbitration Example: Two Masters

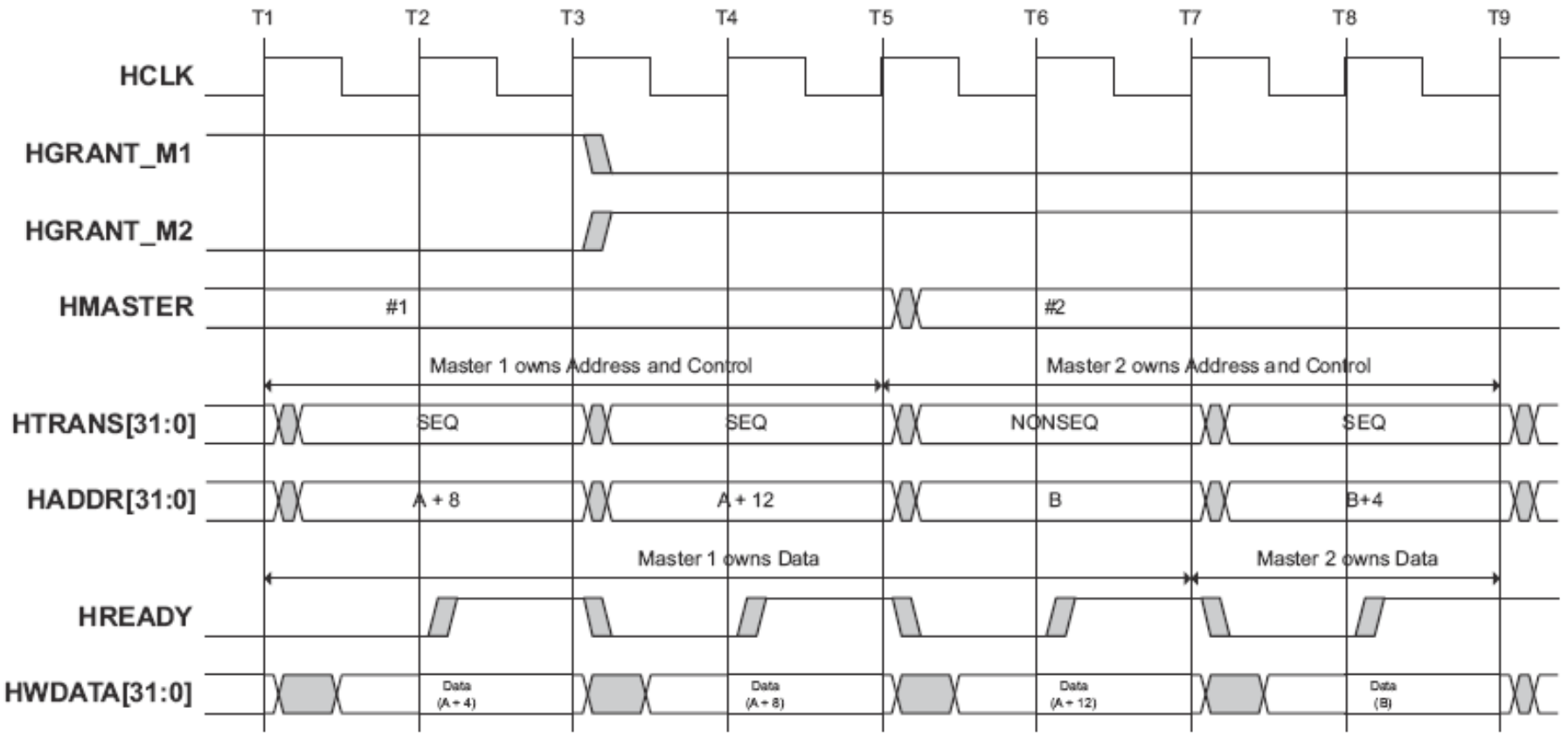


Figure 3-17 Data bus ownership

Handover after burst

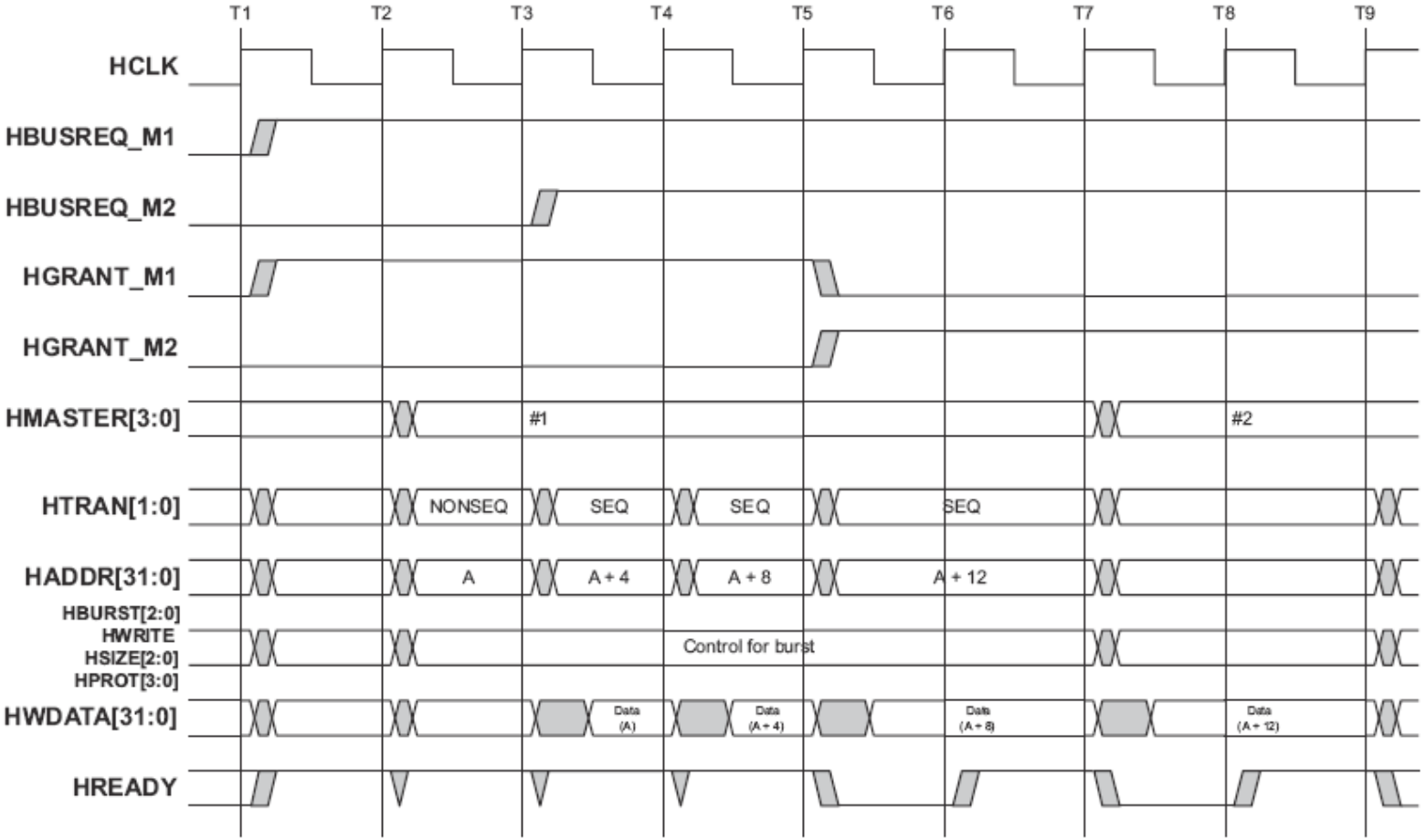
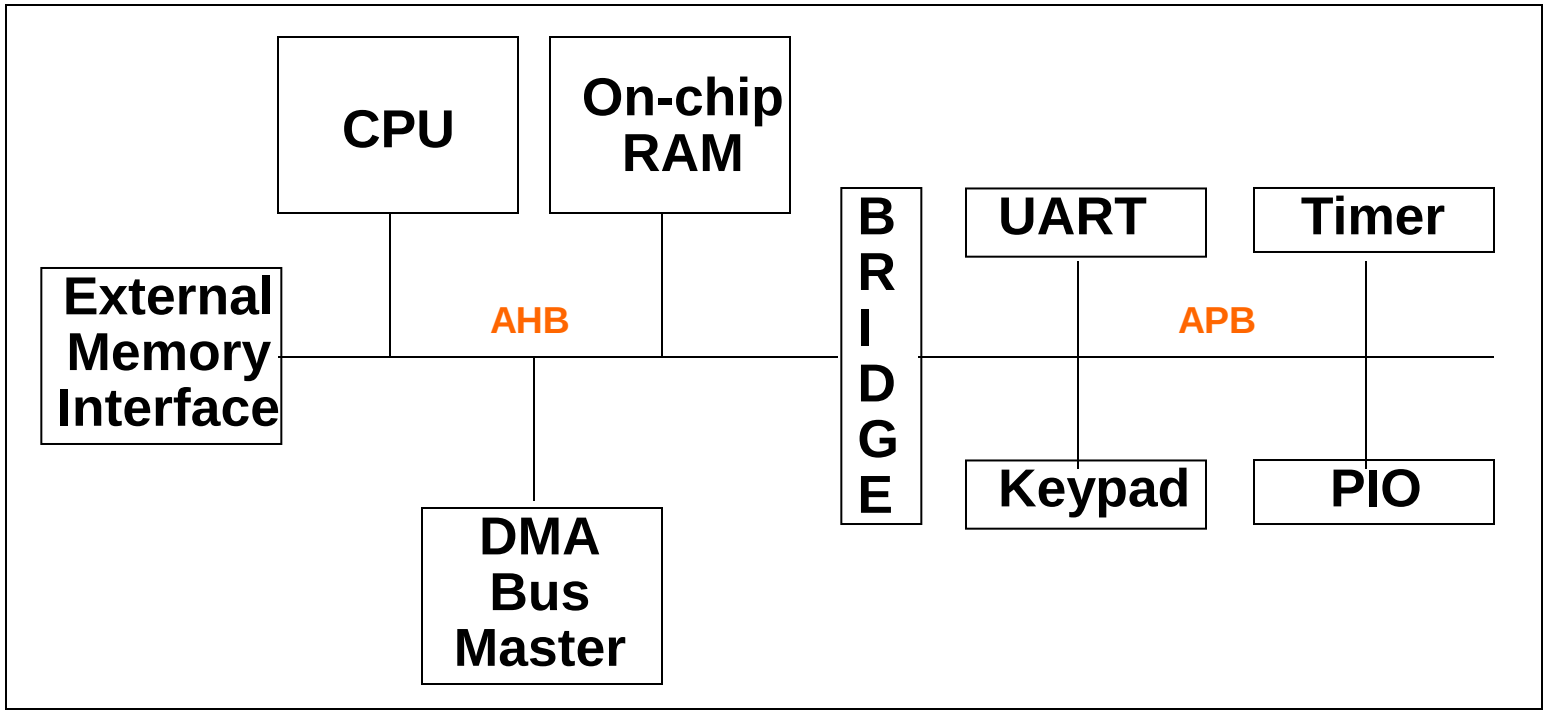


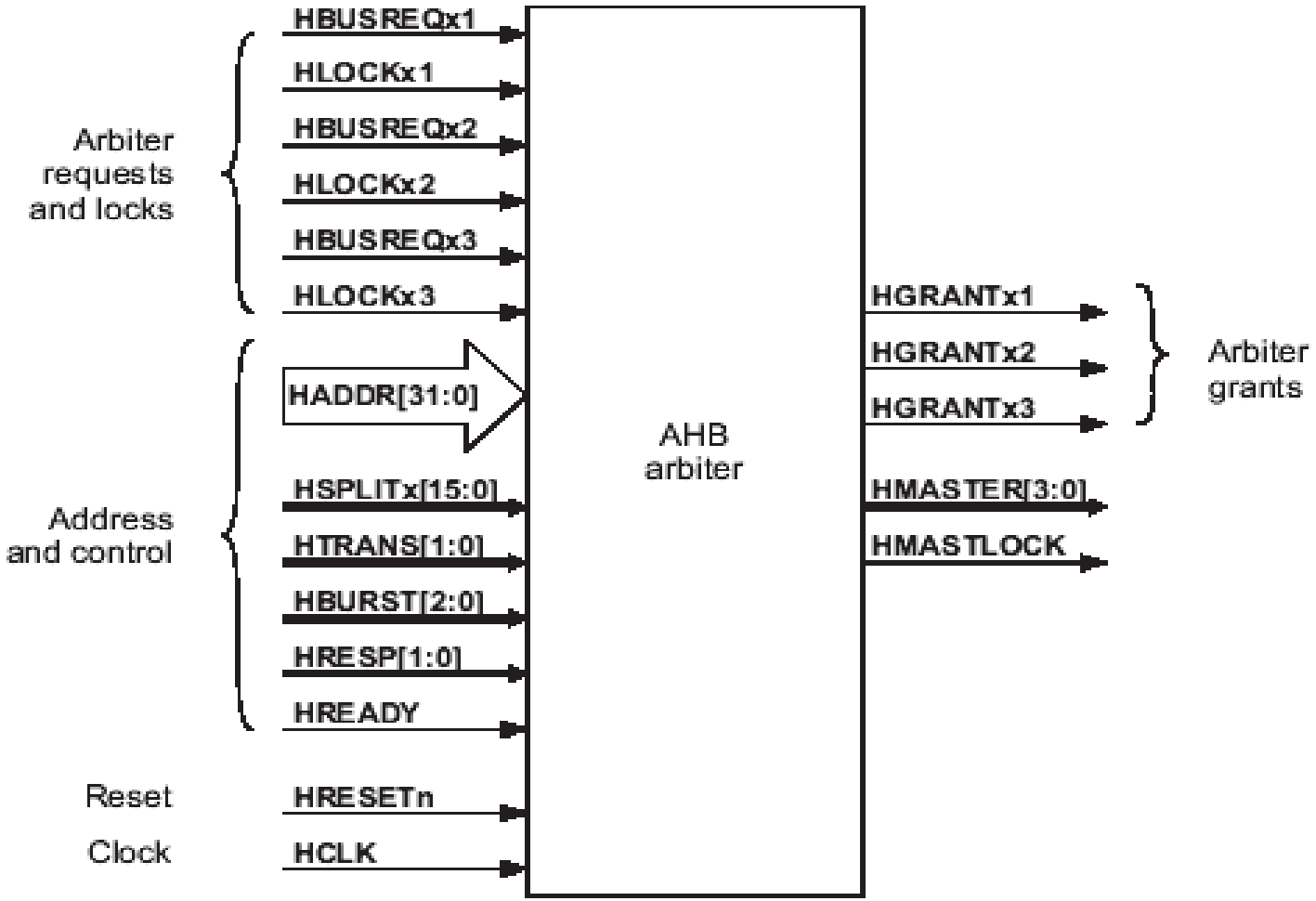
Figure 3-18 Handover after burst

On-Chip Bus (OCB)

- Interconnect components inside a single chip



AHB Arbiter Interface Diagram



AMBA APB

- APB: Advanced Peripheral Bus
- Low power
- Latched address and control
- Simple interface
- Suitable for many peripherals
- No wait state allowed
- No burst transfers
- No arbitration (bridge the only master)
- No pipelined transfer
- No response signal

APB State diagram

- IDLE: default state
- SETUP: only for one clock cycle
- ENABLE: address, data and select signals all must remain stable during the transition for SETUP to ENABLE state. Only for one clock cycle.
- Latched address and control

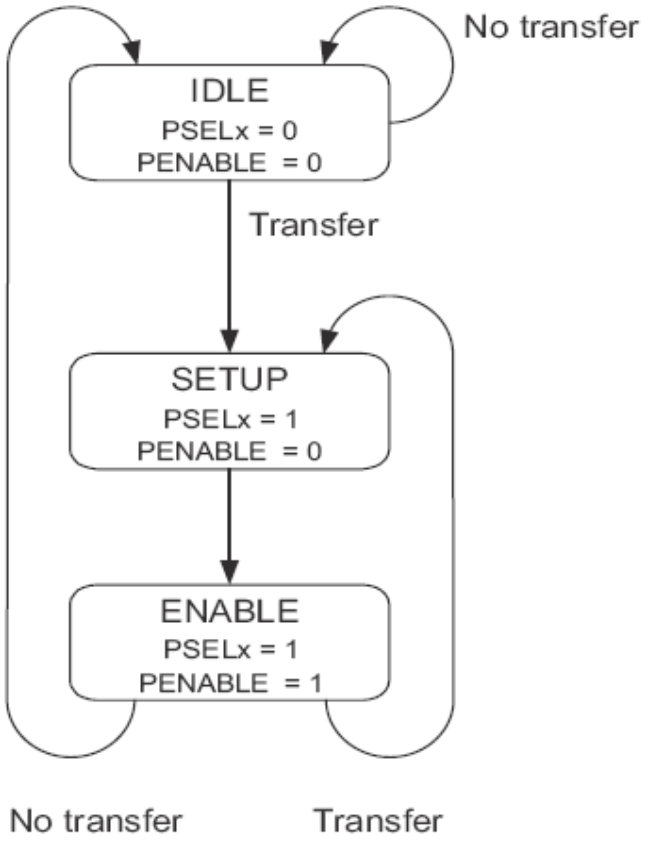
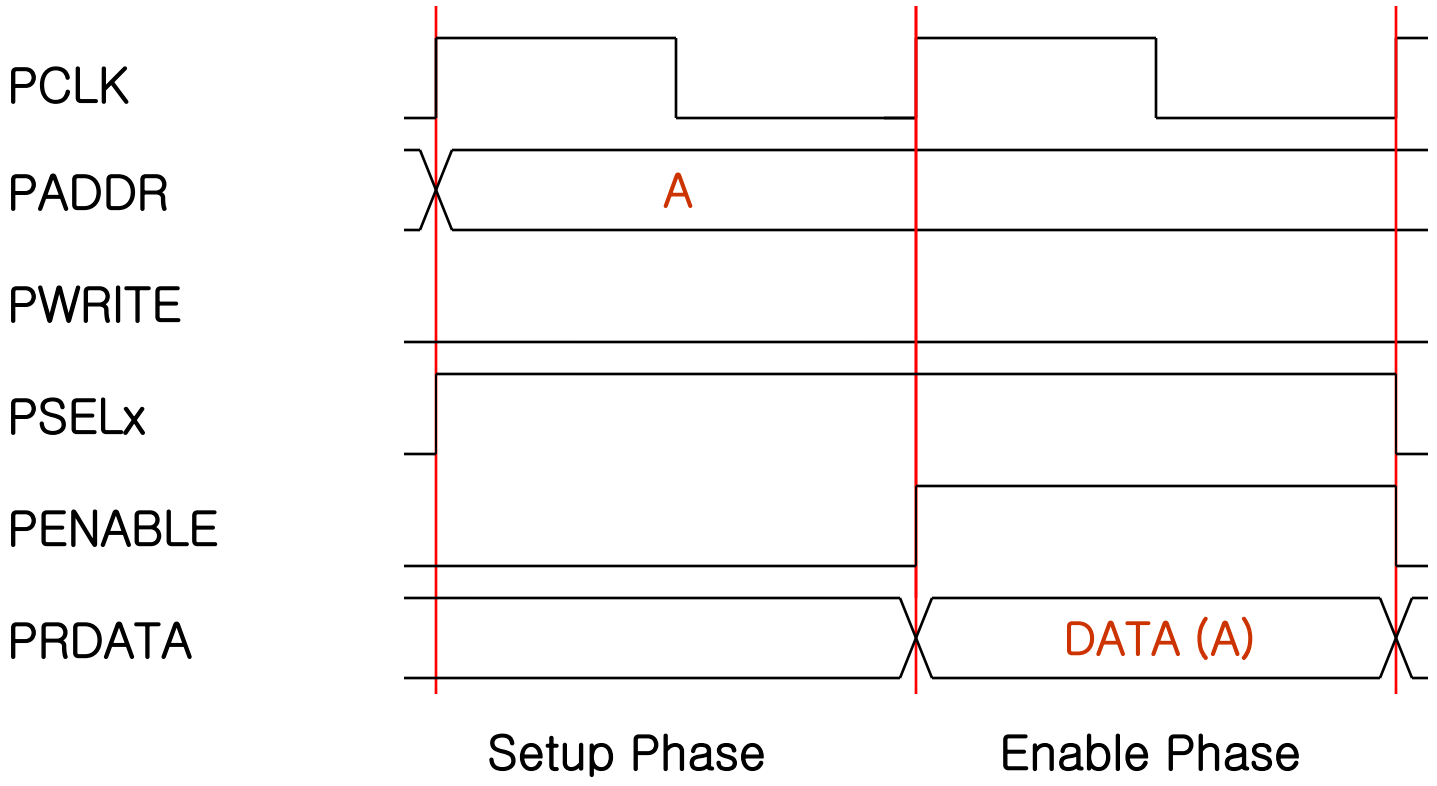
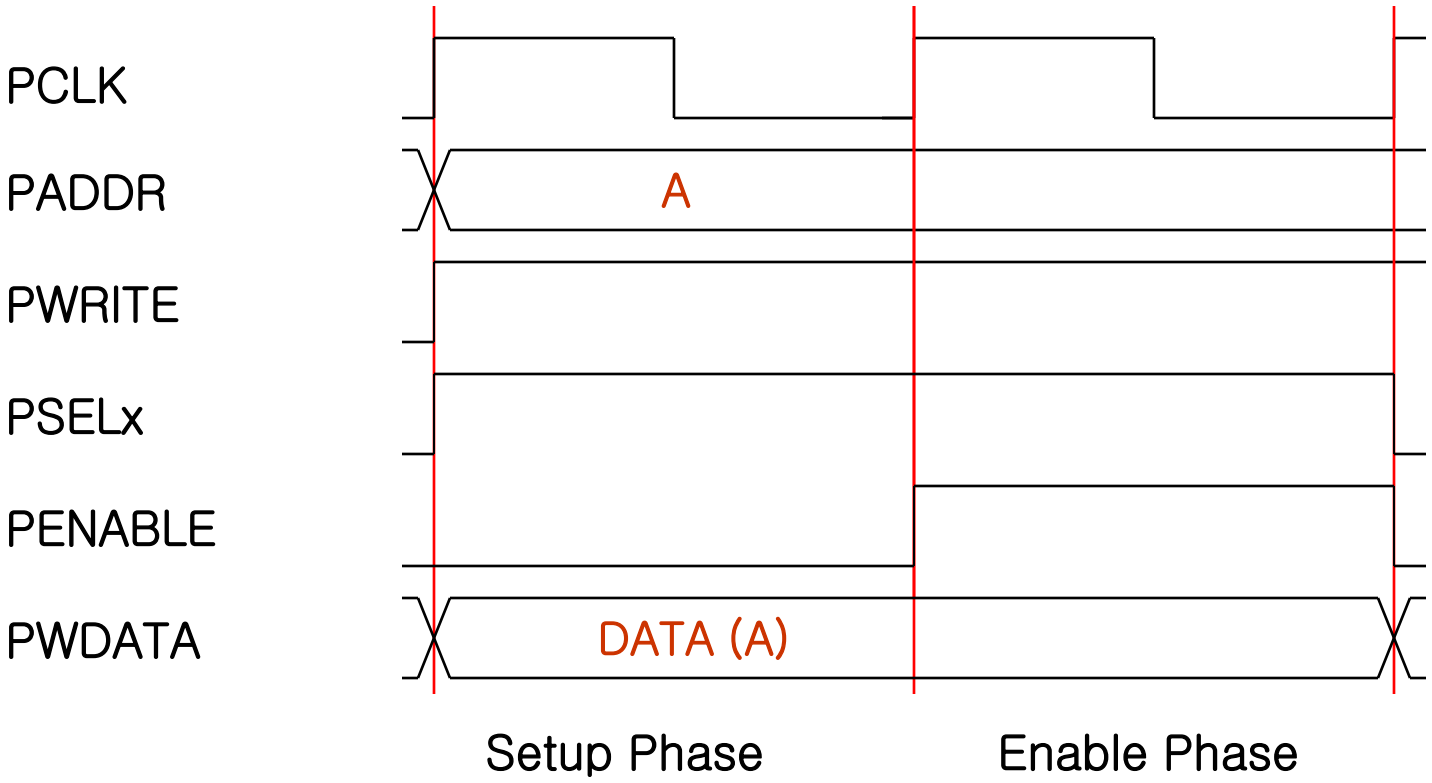


Figure 5-2 State diagram

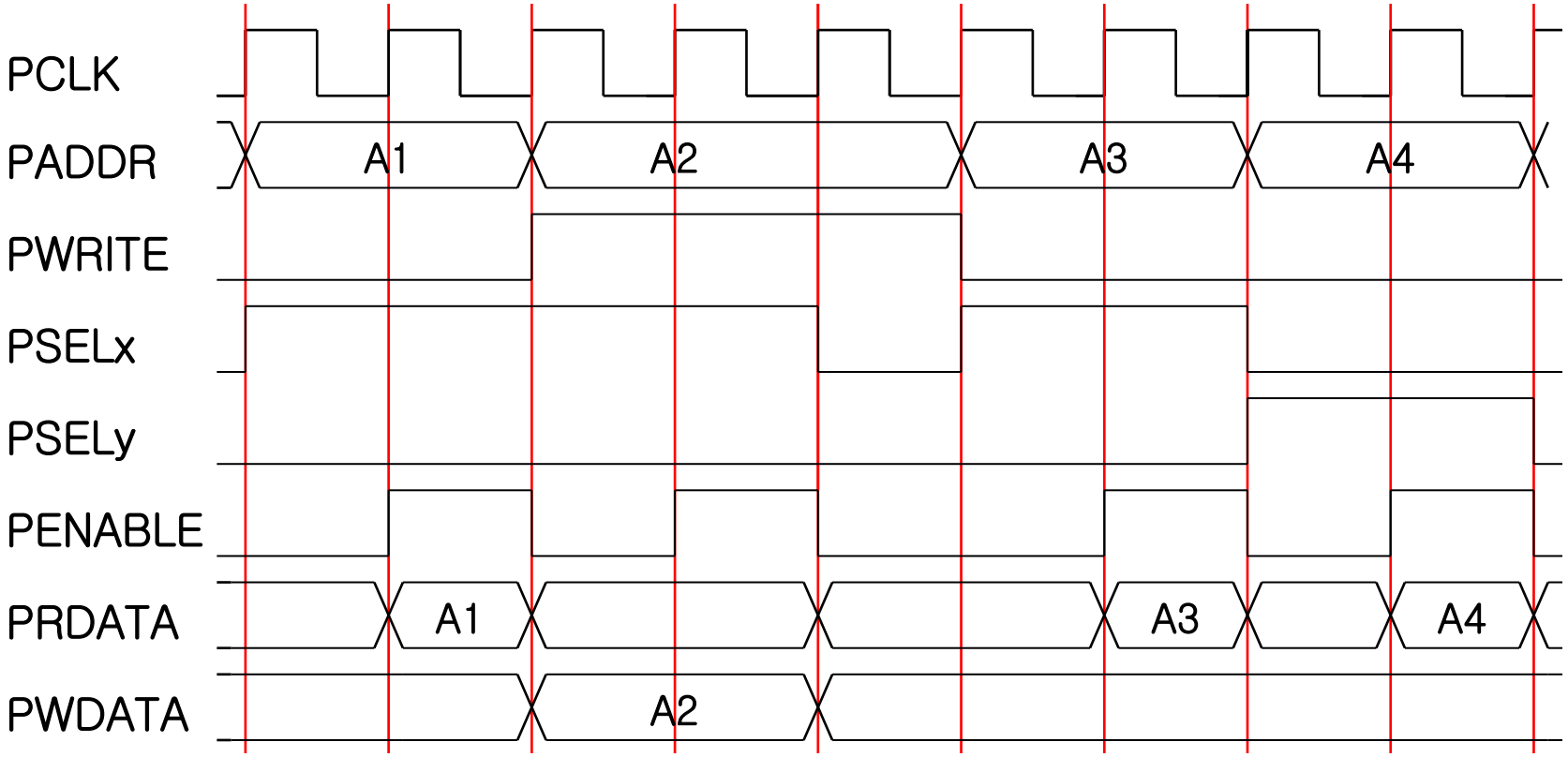
Operation for a Read Txn



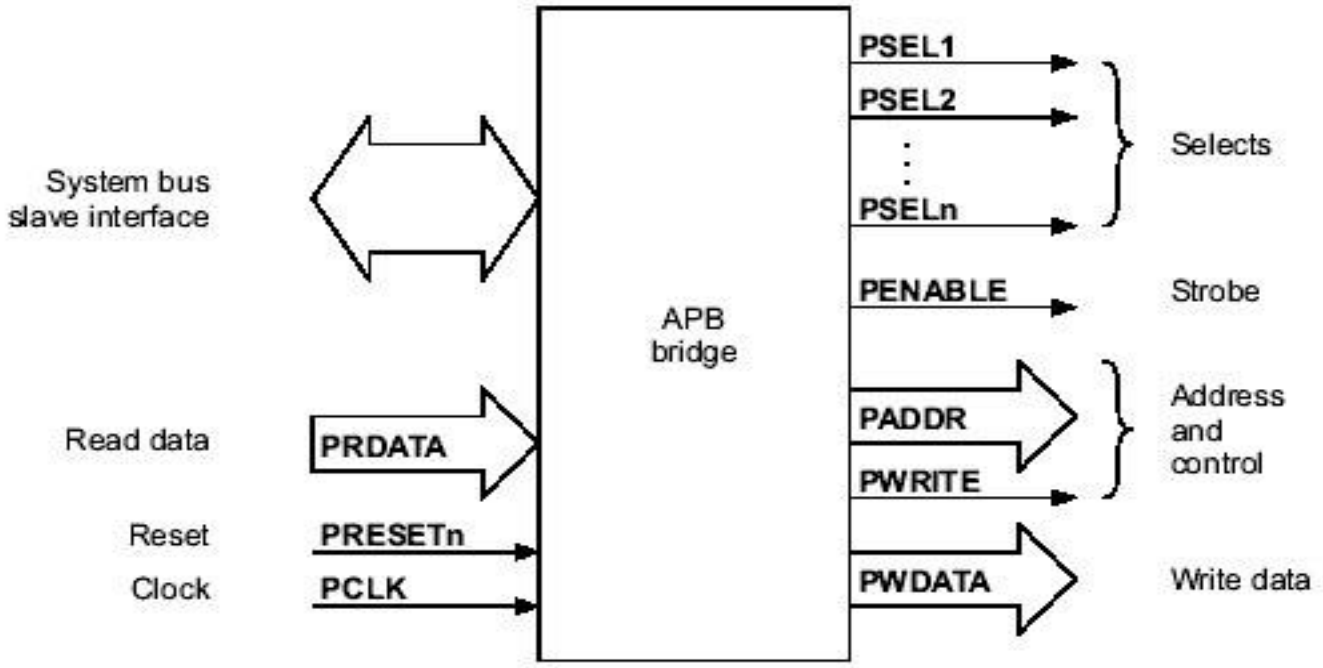
Operation for a Write Txn



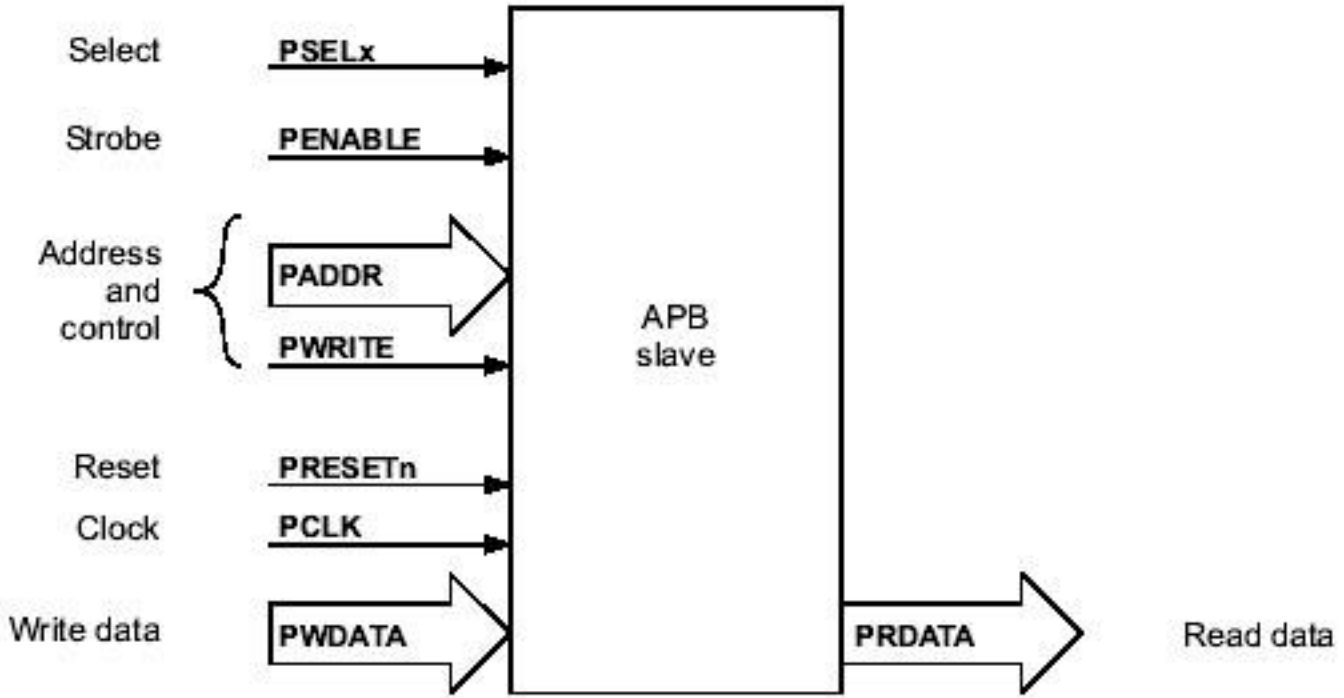
APB Operation Example



APB Bridge


System bus
slave interface
Read data
Reset
Clock
PRDATA
PRESETn
PCLK
APB
bridge
PSEL1
PSEL2
PSELn
PENABLE
PADDR
PWRITE
PWDATA
Selects
Strobe
Address
and
control
Write data

APB Slave


Select
PSELx
Strobe
PENABLE
Address and control
PADDR
PWRITE
Reset
PRESETn
Clock
PCLK
Write data
PWDATA
APB slave
PRDATA
Read data

AMBA AXI

- Targeted at high-performance, high-frequency system designs
- Backward compatible with AHB and APB interfaces
- Separate address/control and data phases
- Support for unaligned data transfers using byte strobes
- Burst-based transaction with only start address issued
- Separate read and write data channels to enable low-cost DMA
- Ability to issue multiple outstanding addresses
- Out-of-order transaction completion
- Easy to add register stages for providing timing closure