SEMICON Solutions

Bus Structure

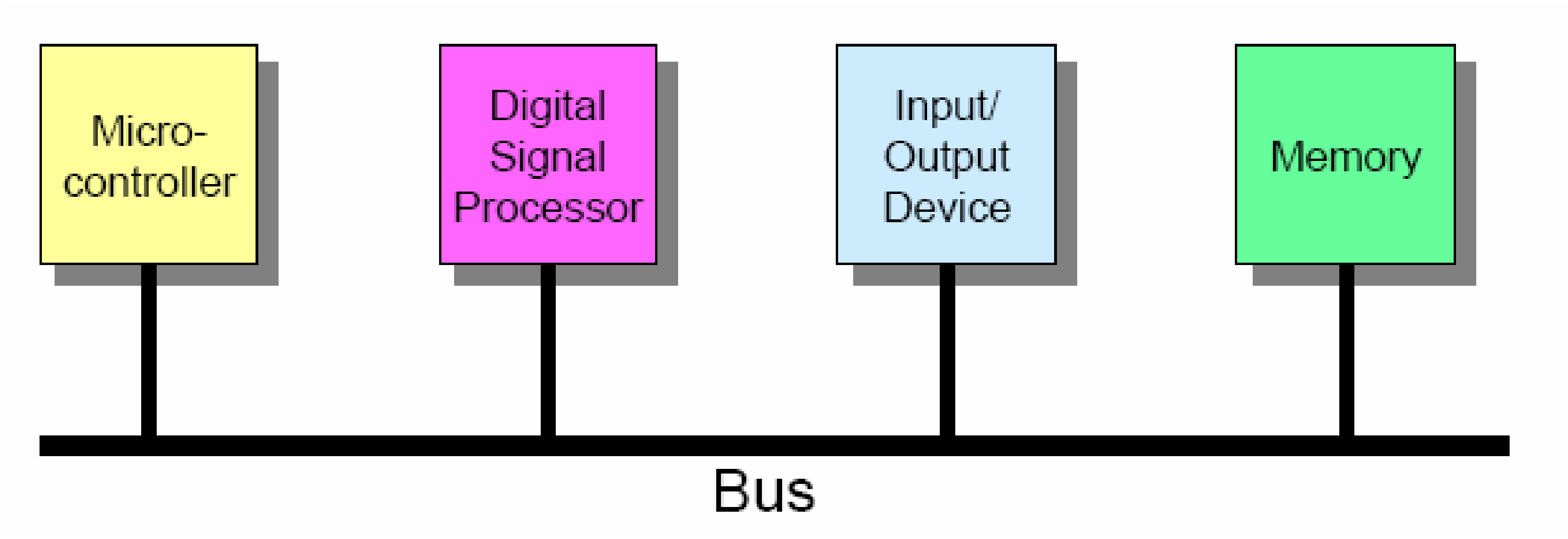
Created by:

Duong Dang

Date: 20th Oct,2010

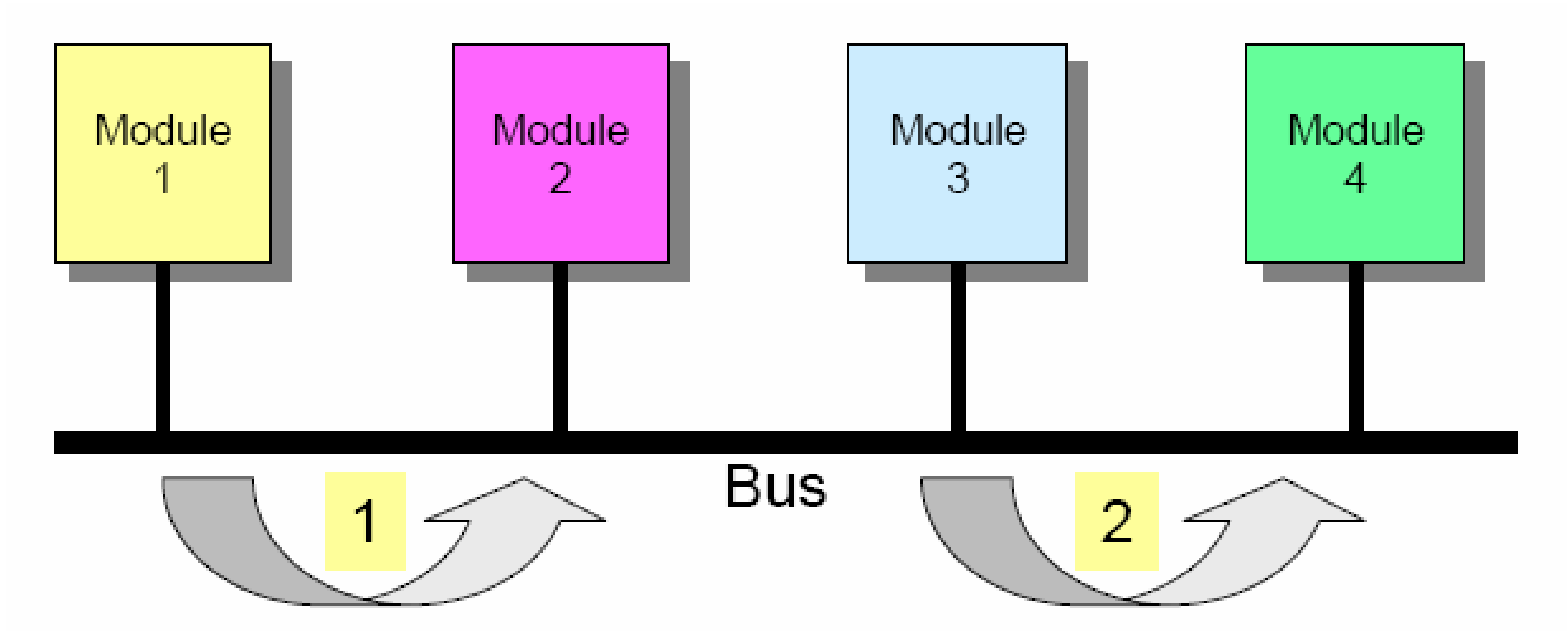
Introduction

- Buses are the simplest and most widely used interconnection networks
- A number of modules is connected via a single shared channel



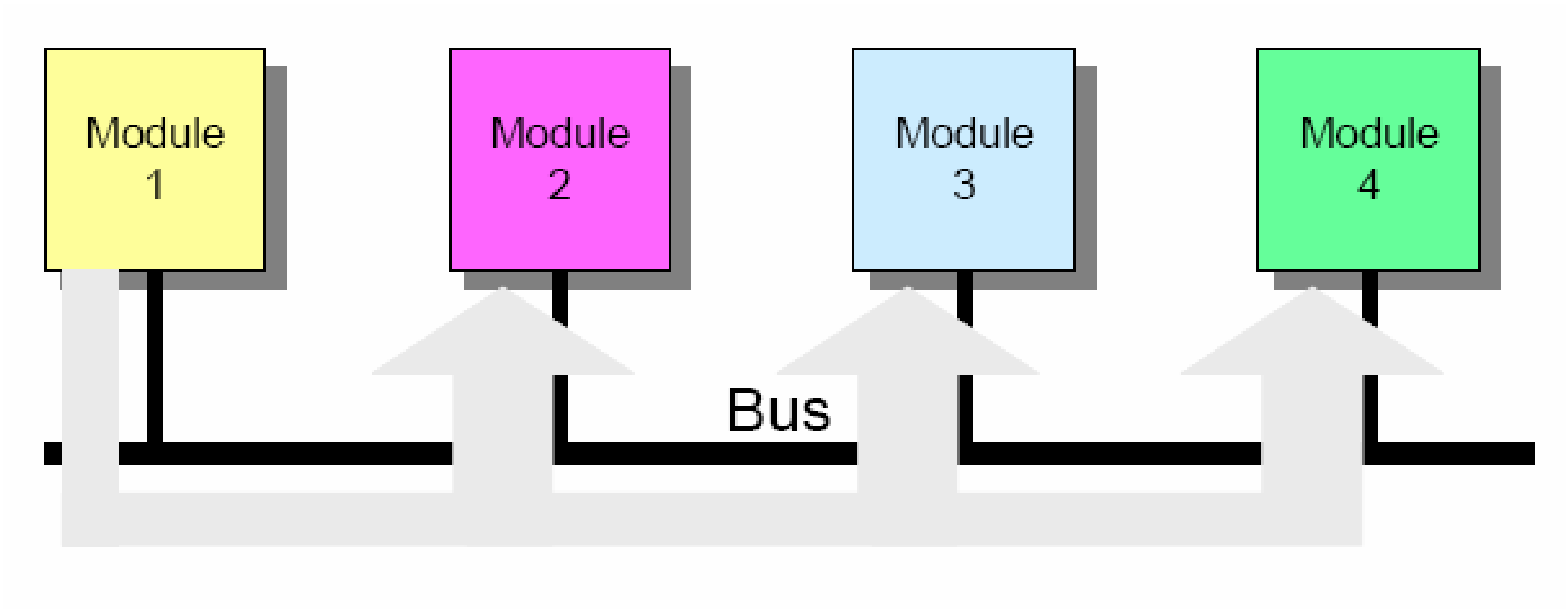
Bus Properties

- Serialization
  - Only one component can send a message at any given time
  - There is a total order of messages



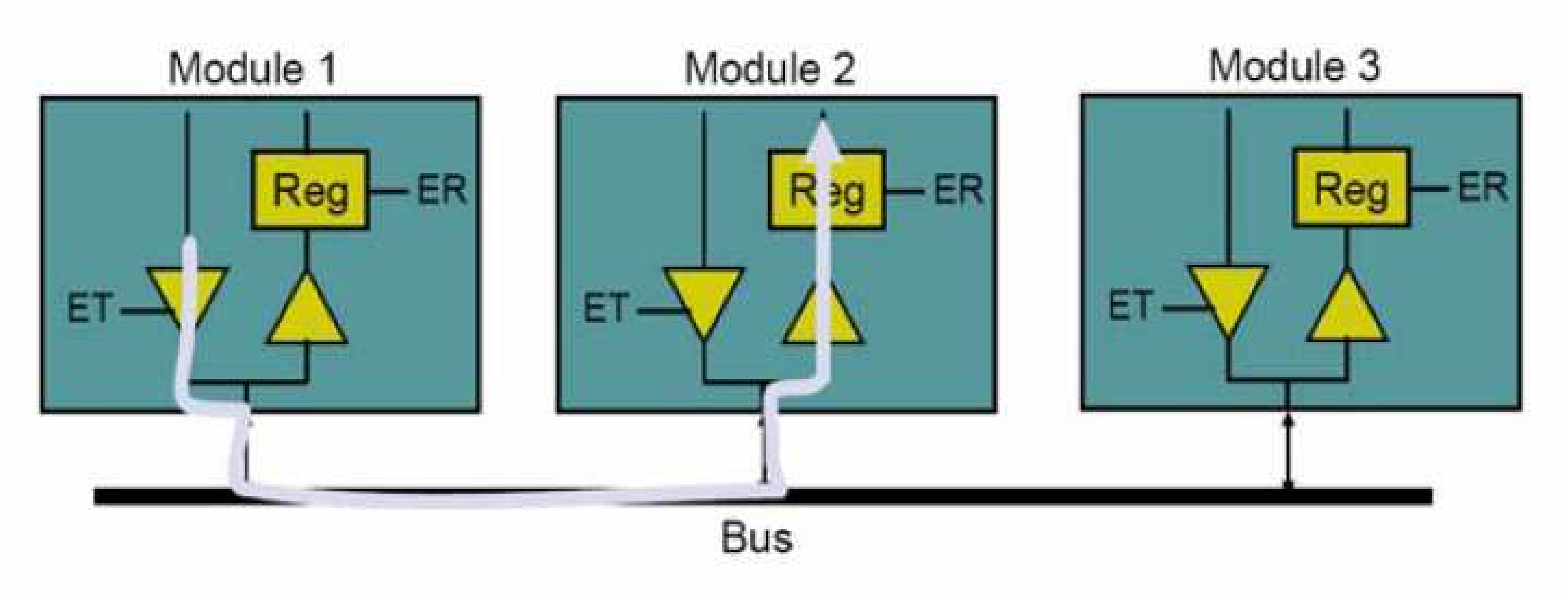
Bus Properties

- Broadcast
  - A module can send a message to several other components without an extra cost



Bus Hardware

- Principle for hardware to access the bus
  - Bus Transmit: ET active
  - Bus Receive: ER active



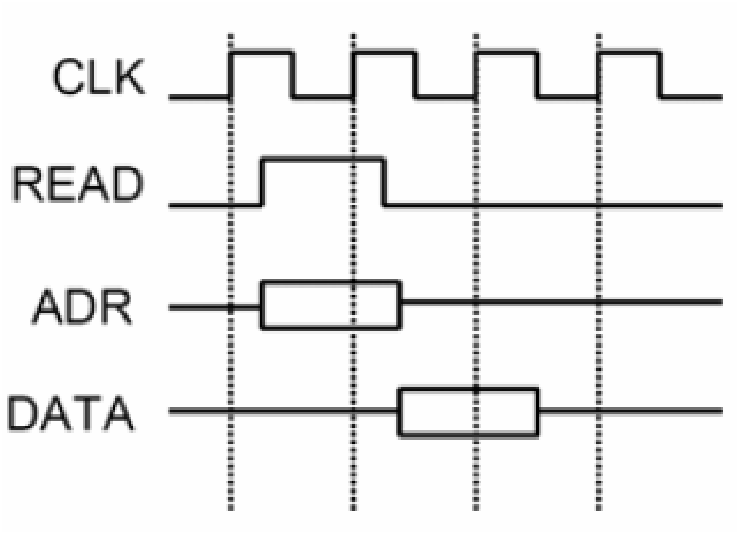
SEMICON

Cycles, Messages and Transactions

- Buses operate in units of cycles, messages and transactions
  - *Cycles*: A message requires a number of cycles to be sent from sender to receiver over the bus
  - *Message*: Logical unit of information (a read message contains an address and control signals for read)
  - *Transaction*: A transaction consists of a sequence of messages which together form a transaction (a memory read requires a memory read message and a reply with the requested data)

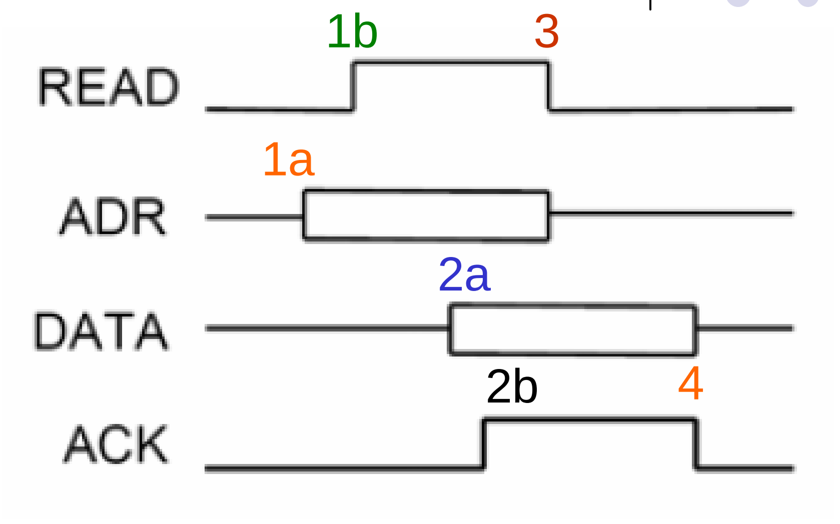
Synchronous Bus

- Includes a clock in the control lines
- A fixed protocol for communication that is relative to the clock
- Advantage: involves very little logic and can run very fast
- Disadvantages:
  - Every device on the bus must run at the same clock rate
  - To avoid clock skew, they cannot be long if they are fast



Asynchronous Bus

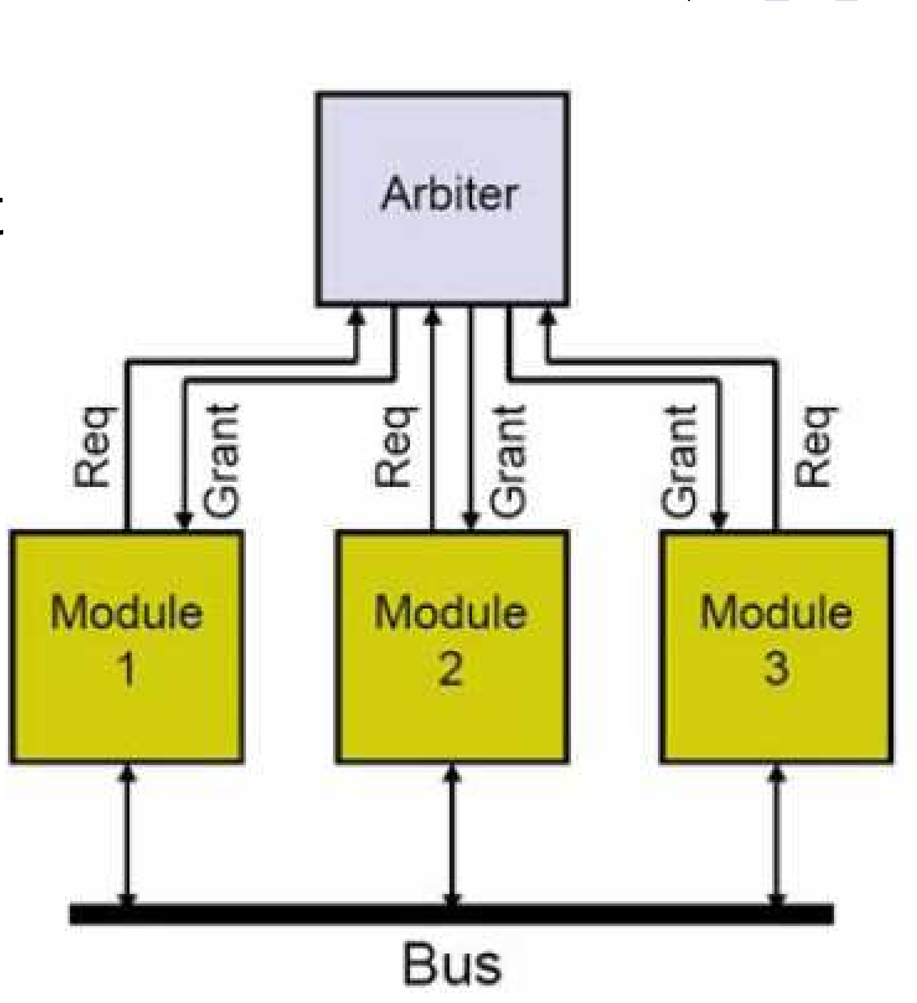
- It is not clocked
- It can accommodate a wide range of devices
- It can be lengthened without worrying about clock skew
- It requires a handshaking protocol



1. Master puts address on bus and asserts READ when address is stable
2. Memory puts data on bus and asserts ACK when data is stable
3. Master deasserts READ when data is read
4. Memory deasserts ACK

Bus Arbitration

- Since only one bus master can use the bus at a given time bus arbitration is used
- An arbiter collects the requests of all bus masters and gives only one module the right to access the bus (bus grant)



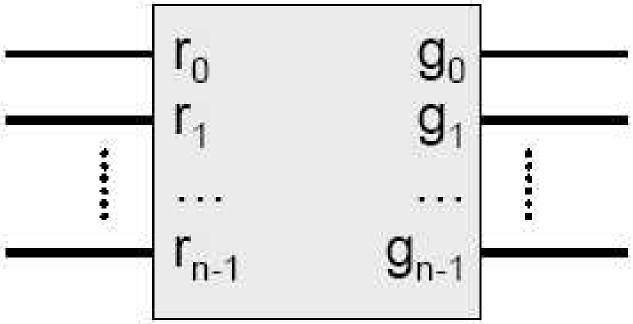
SEMICON

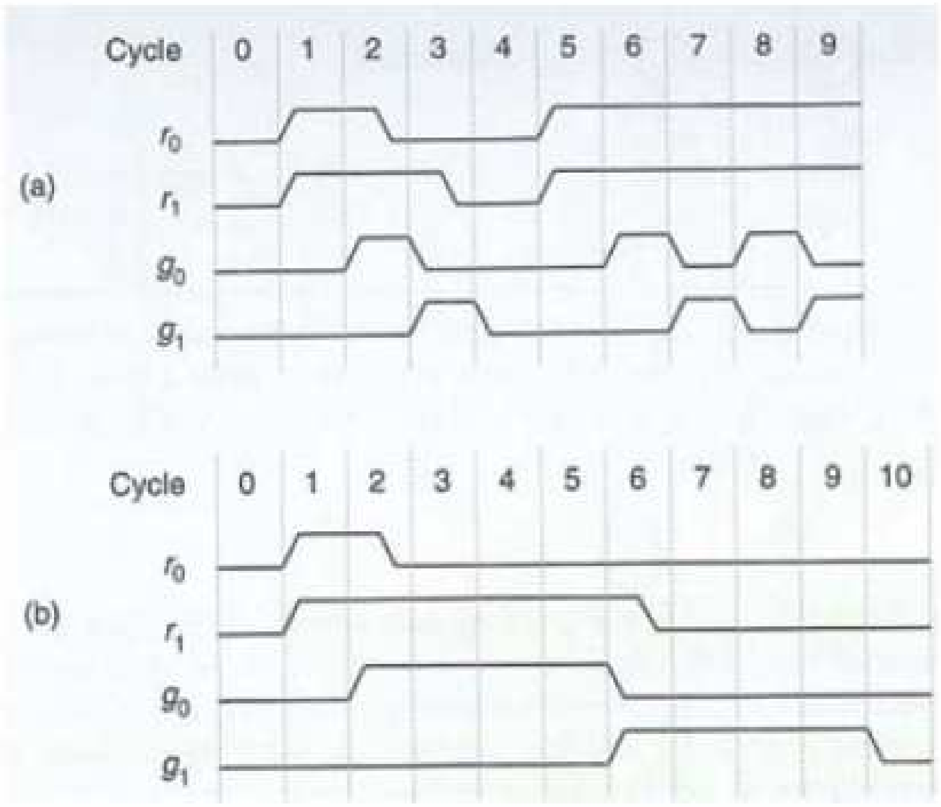
Importance of Arbiters

- Arbiters are not only used in bus-system, but everywhere where several devices request shared resources
- In network-on-chips arbitration is for instance needed, if two or more packets want to enter the same channel

Arbiter Interfaces

- This arbiter interface can be used to give a bus grant for a fixed number of cycles
  - (a): 1 cycle
  - (b): 4 cycles

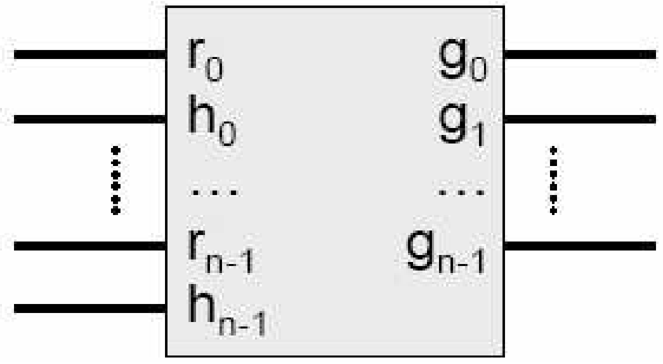


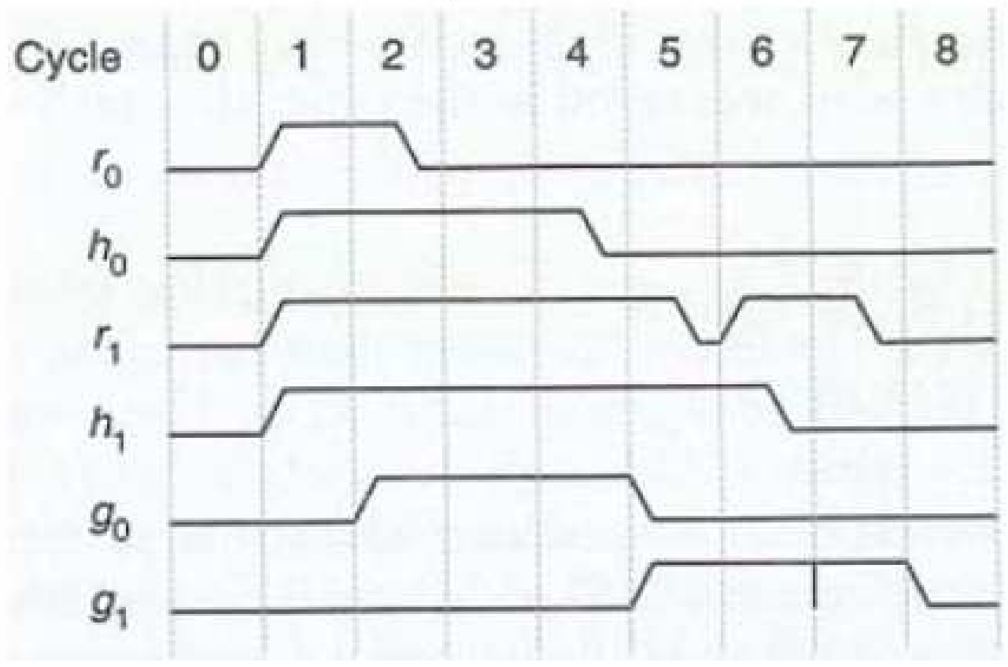


SEMICON

Arbiter Interfaces

- This arbiter allows for variable length grants
- The grant is hold as long as the “hold”-line is asserted
- In cycle 2 requester 0 gets the bus for 3 cycles
- In cycle 5 requester 1 gets the bus for 2 cycles
- In cycle 7 requester 1 gets the bus for one cycle





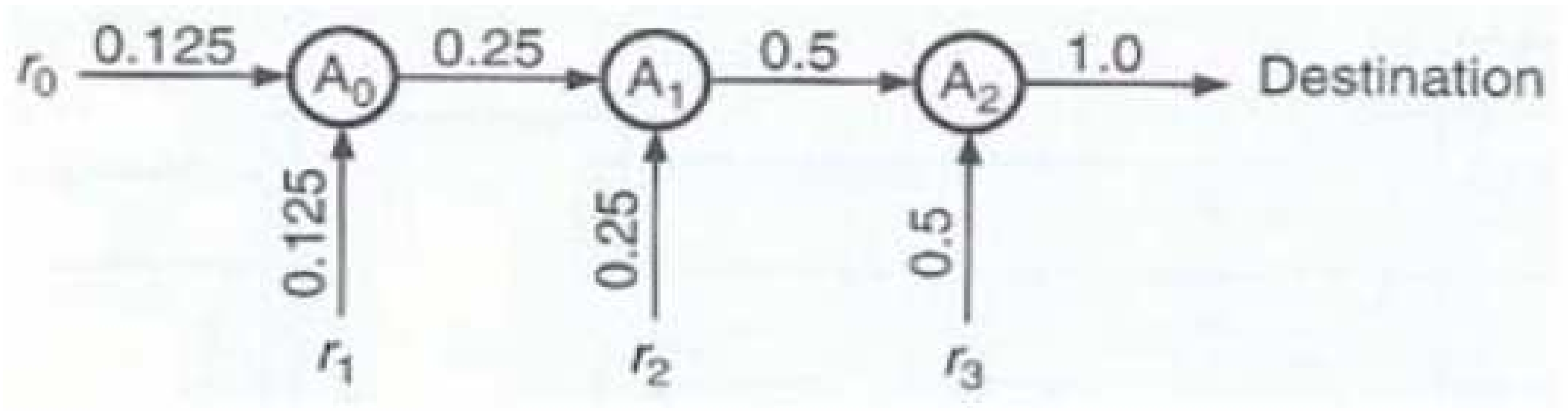
Fairness

- Fairness is a key property of an arbiter
- Some definitions
  - *Weak fairness*: Every request is eventually served
  - *Strong fairness*: Requests will be served equally often
  - *Weighted "strong" fairness*: The number of times requester *i* is served is equal to its weight *w*i
  - *FIFO fairness*: Requests are served in the order the requests have been made

SEMICON
THINKING OUT OF THE BOX

Local Fairness vs. Global Fairness

- Even if an arbiter is locally fair, a system with several arbiters employing that arbiter may not be fair

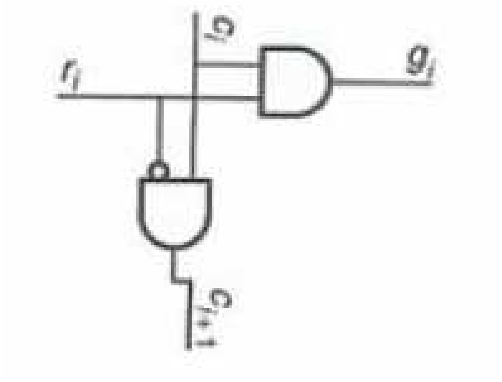


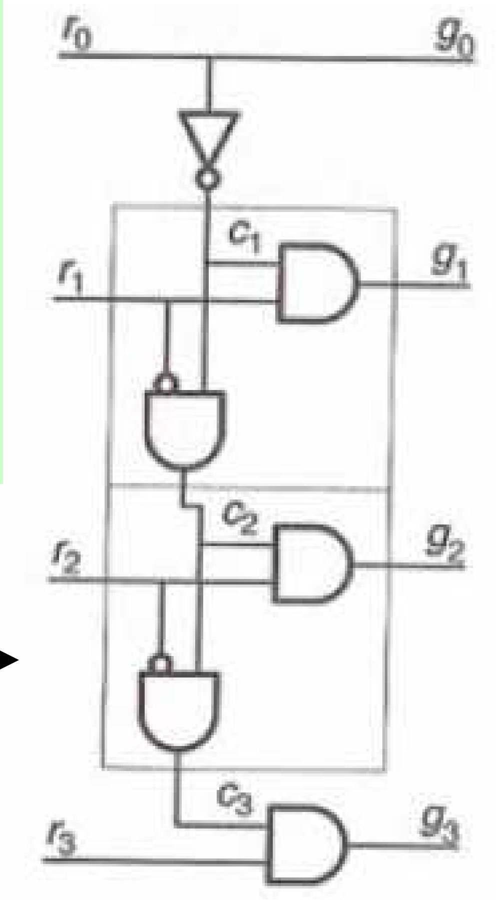
- Though each arbiter *A*i allocate 50% of their bandwidth to its two inputs, *r*0 only gets 12.5% of the total bandwidth, while *r*3 gets 50%

SEMICON
THINKING OUT OF THE BOX

Fixed-Priority Arbiter

- A fixed-priority arbiter can be constructed as an iterative circuit
- Each cell receives a request input *r*i and a carry input *c*i and generates a grant output *g*i and a carry output *c*i+1
- The resulting arbiter is not fair, since a continuously asserted request *r*0 means that none of the other requests will ever be served!





Variable-Priority Arbiters

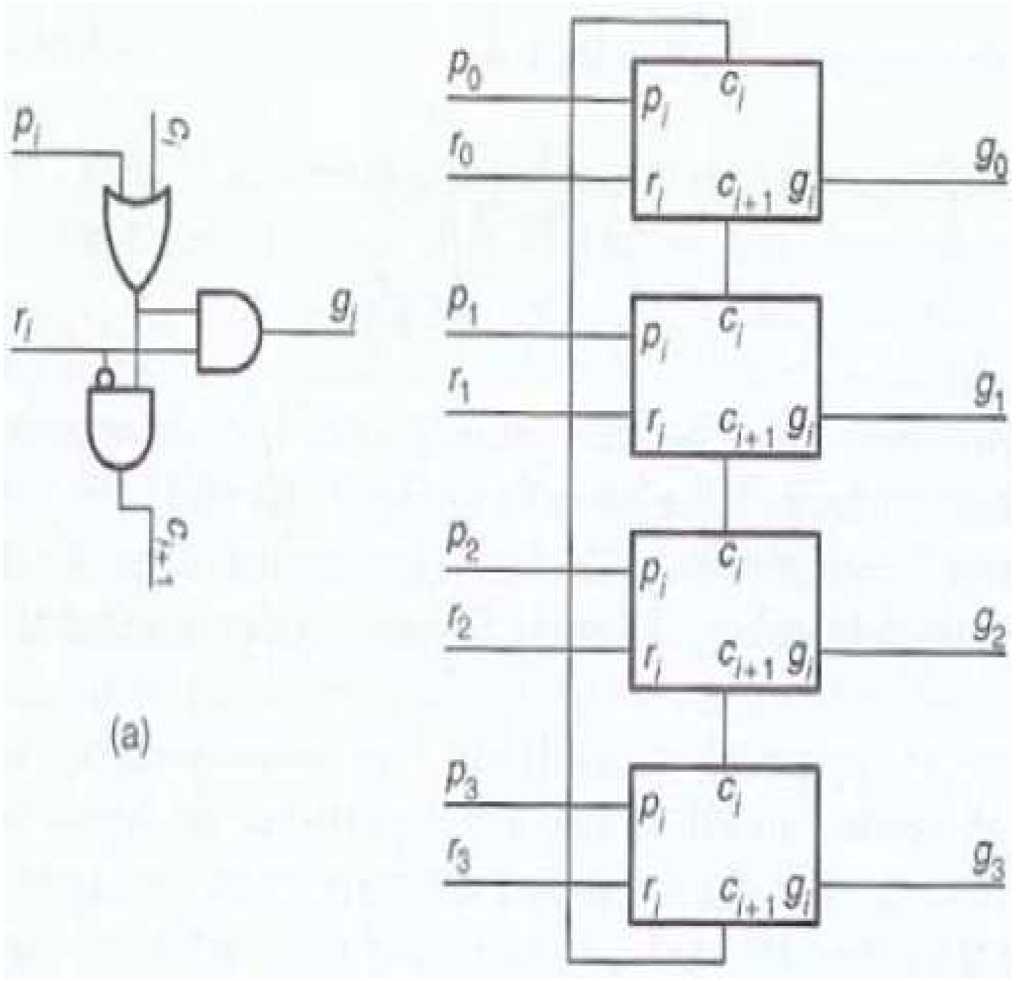
- Oblivious Arbiter
- Round-Robin Arbiter
- Grant-Hold Circuit
- Weighted Round-Robin Arbiter

SEMICON
THINKING OUT OF THE BOX

Fair Arbiters

- A fair arbiter can be generated by changing the priority from cycle to cycle

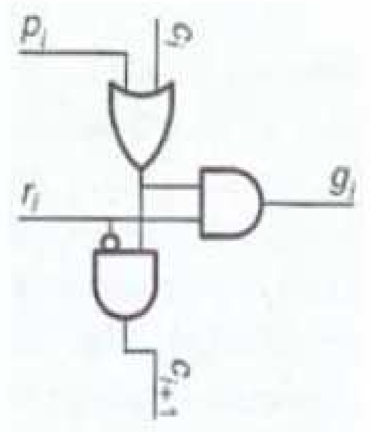
- Depending on the priority generation, different arbitration schemes and degrees of fairness can be achieved

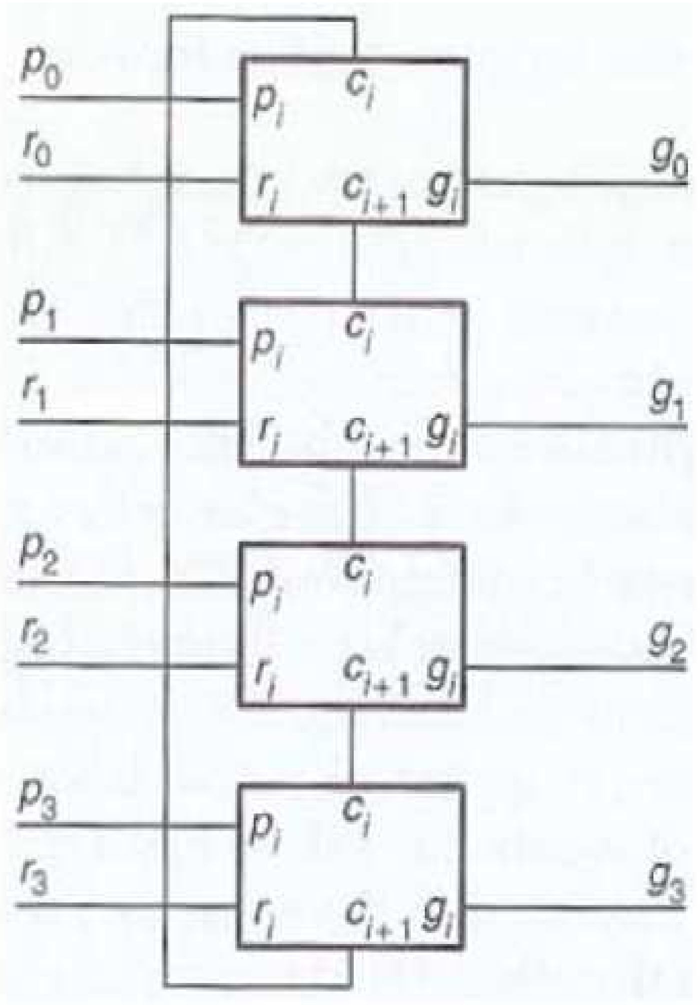


SEMICON

Fair Arbiters

- Oblivious Arbiters
  - If *pi* is generated without knowledge of *ri* and *gi*, the result is an oblivious (unconscious) arbiter
  - Examples are
    - Randomly generated *pi*
    - Rotating priorities (by shift register)

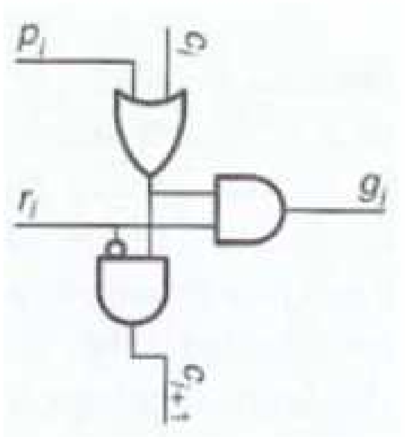


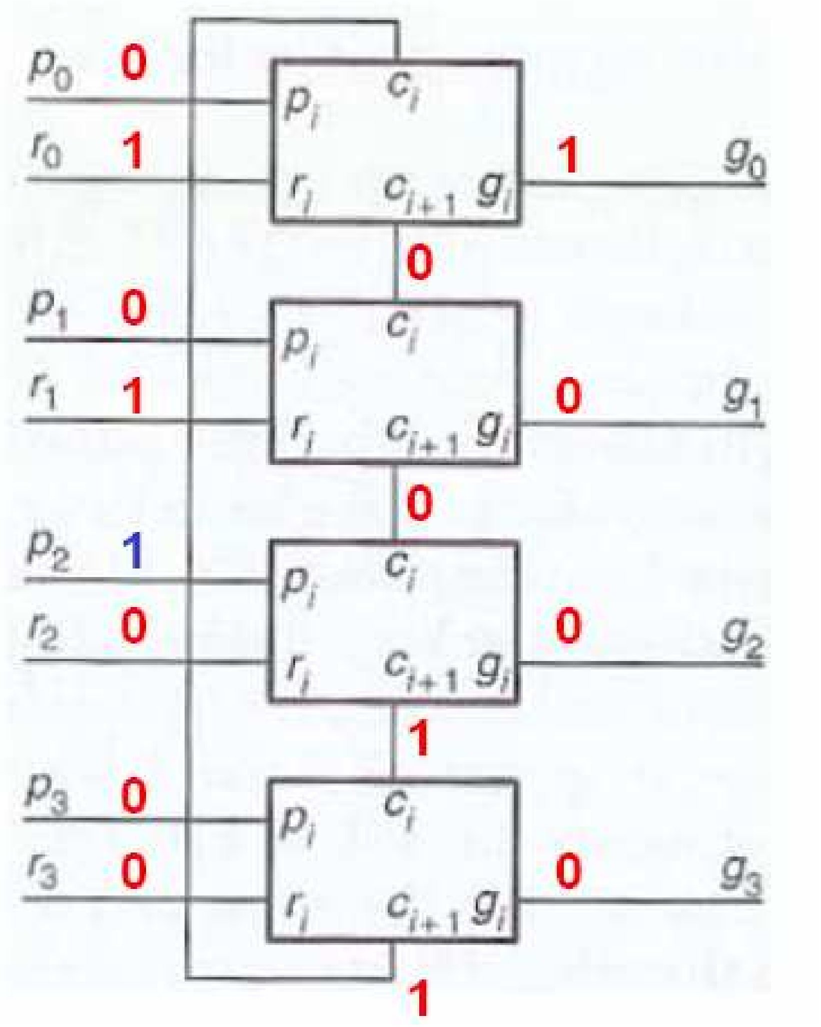


SEMICON
THINKING OUT OF THE BOX

Oblivious Arbiters

- Oblivious arbiters provide weak fairness but not strong fairness (i.e. if *r*0 and *r*1 are constantly asserted)

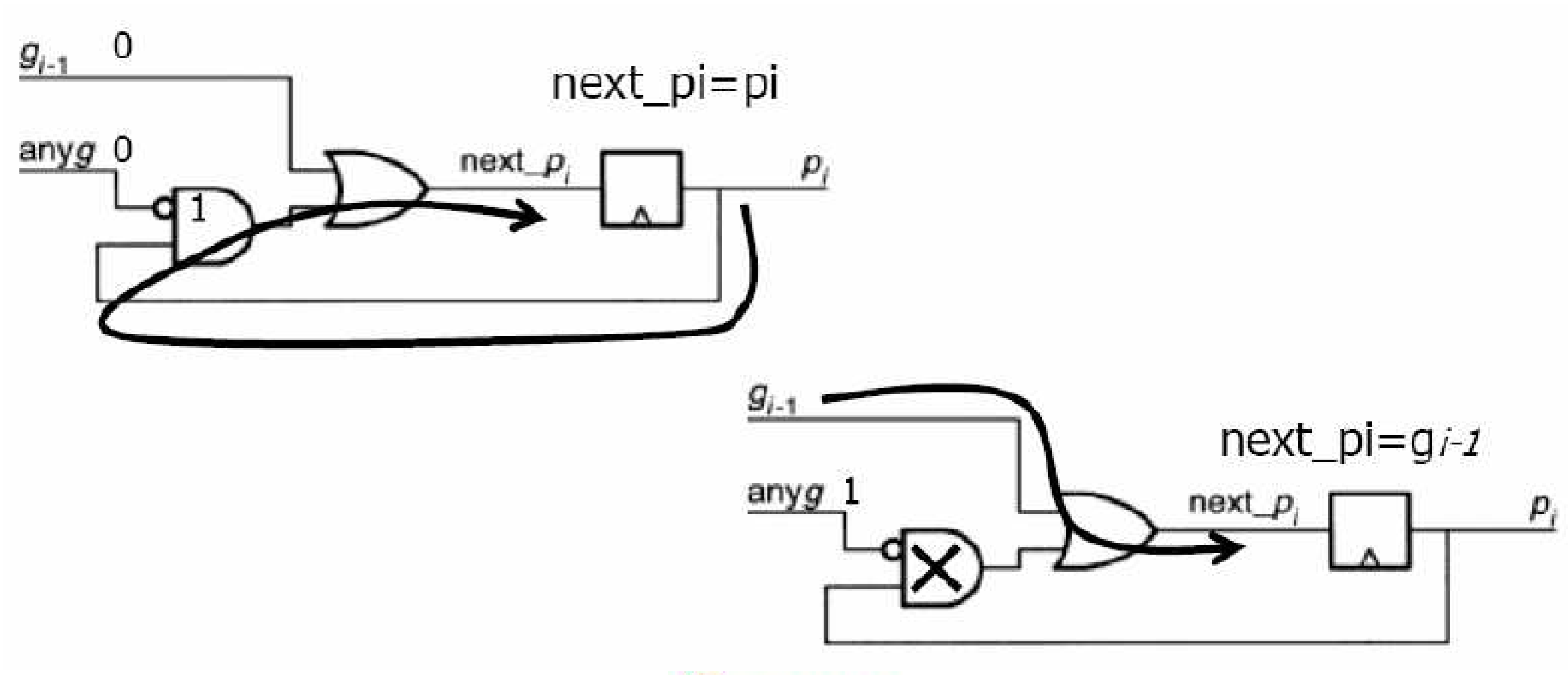




SEMICON
THINKING OUT OF THE BOX

Round-Robin Arbiter

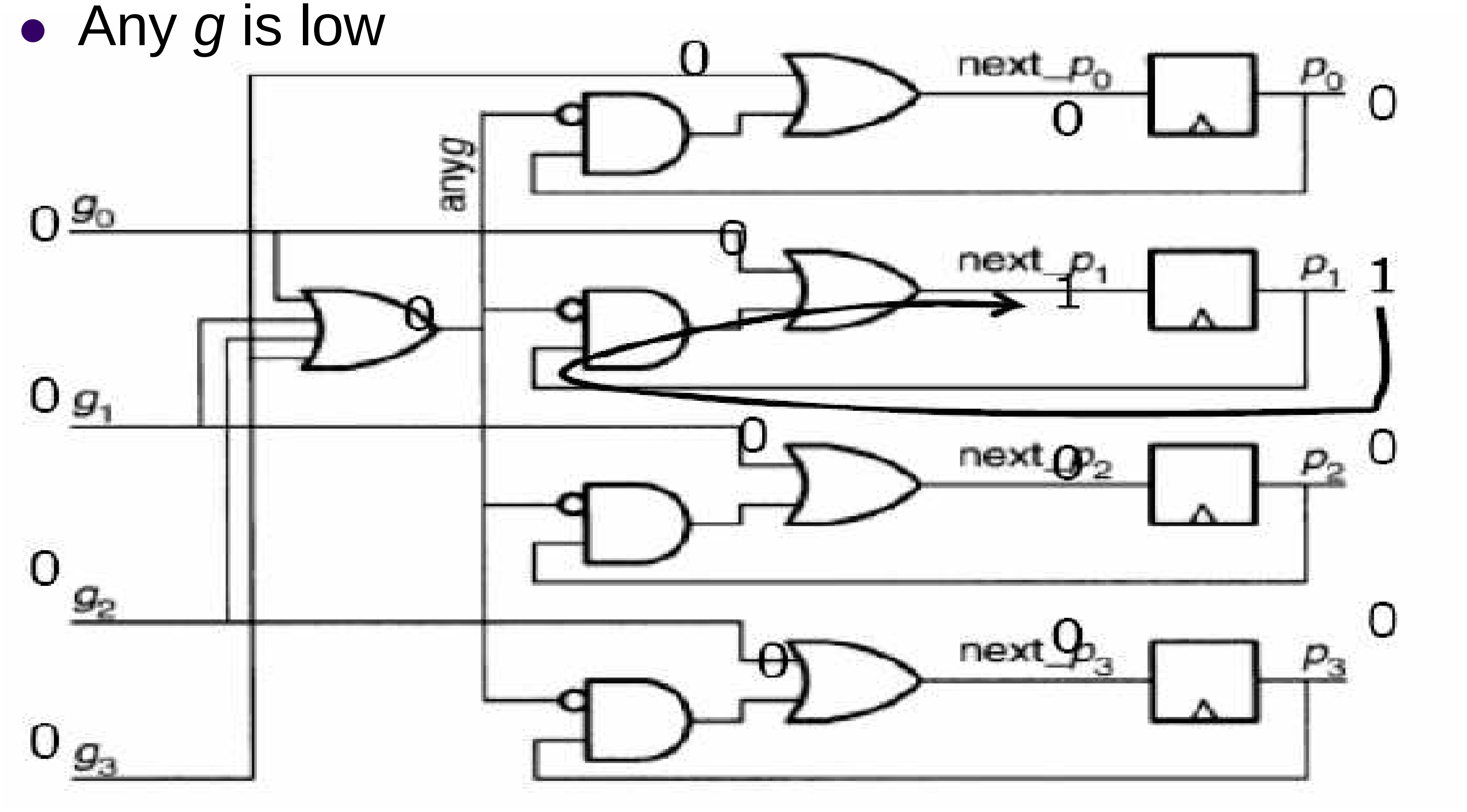
- A round-robin arbiter achieves strong fairness
  - A request that was just served gets the lowest priority
- One-bit round-robin arbiter



SEMICON
THINKING OUT OF THE BOX

Round-Robin Arbiter

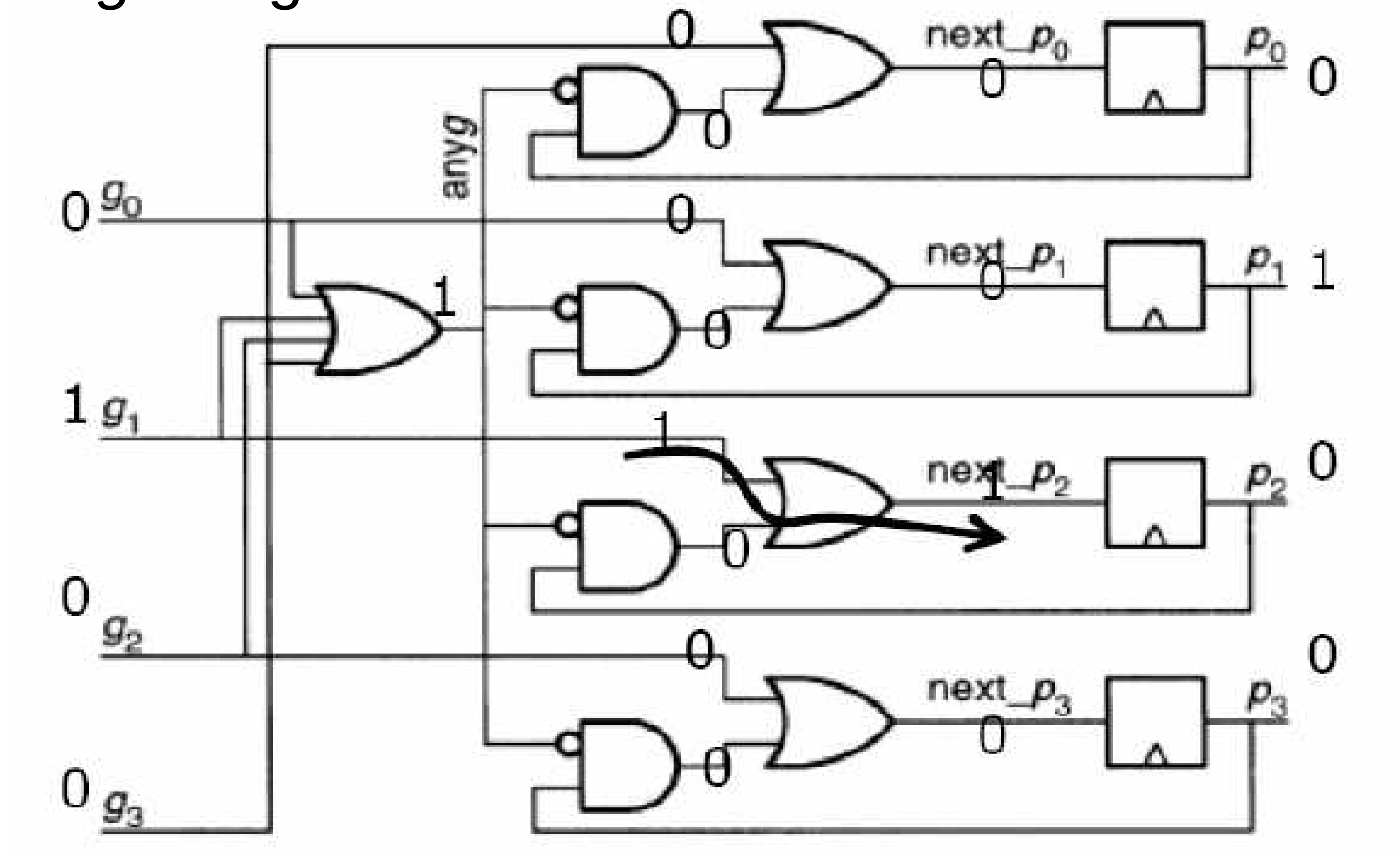
- Any *g* is low



SEMICON

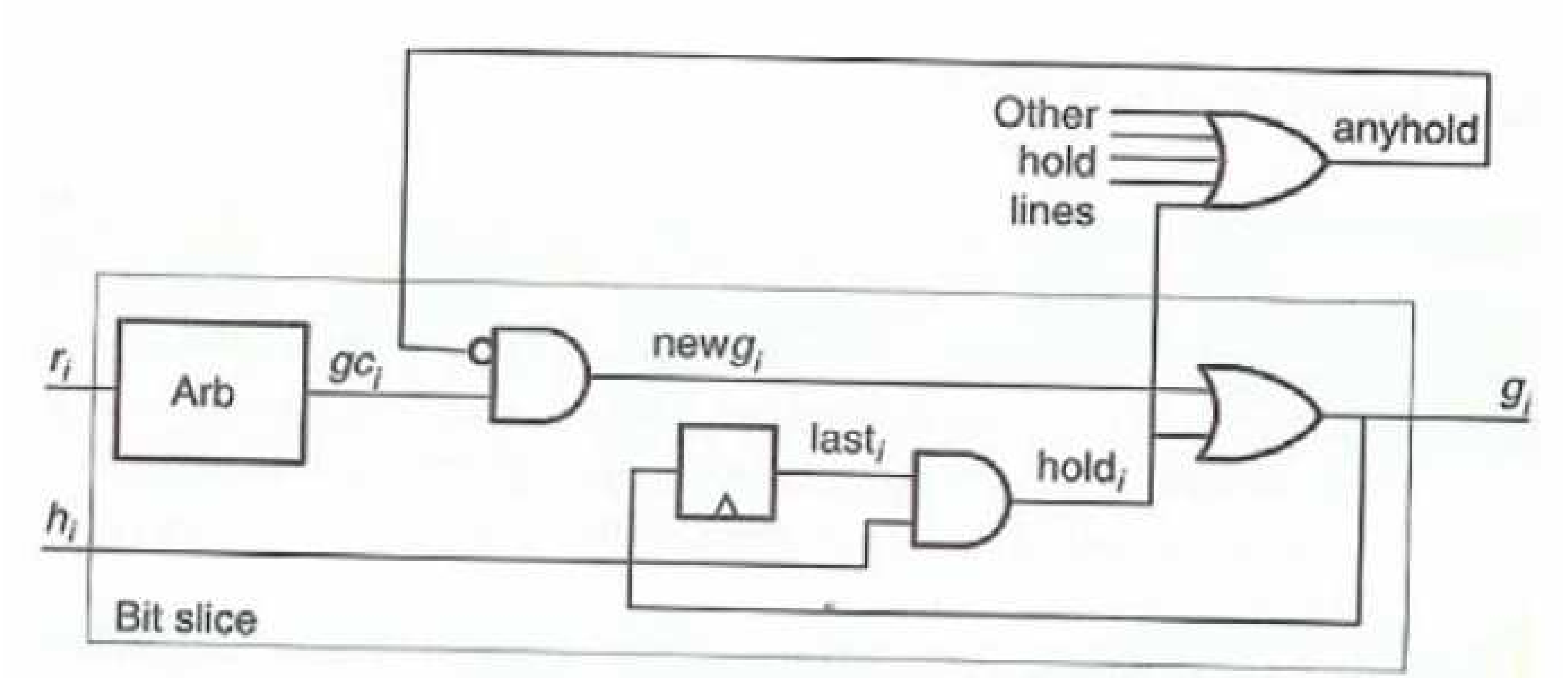
Round-Robin Arbiter

- One *g* is high



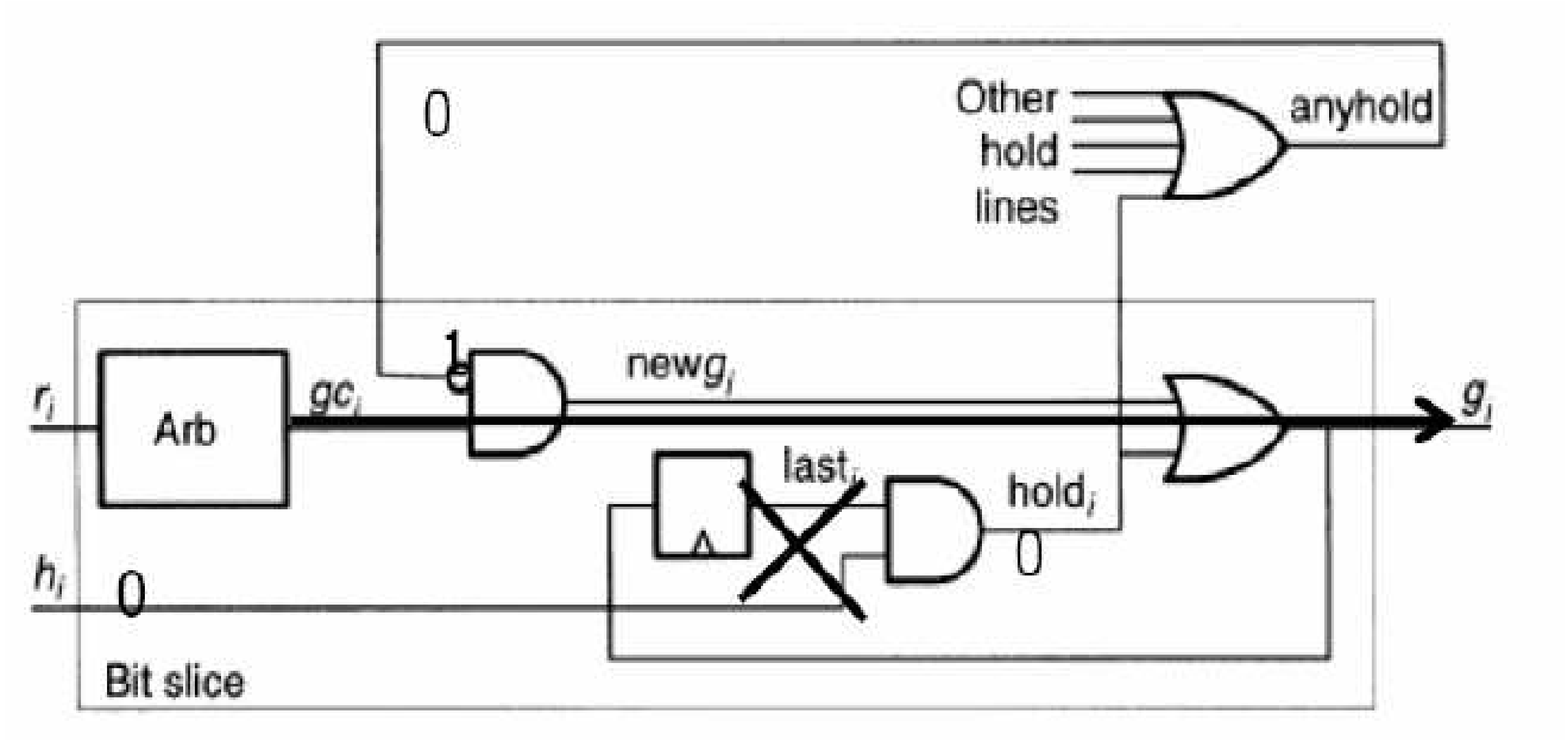
Grant-Hold Circuit

- Extends the duration of a grant
  - As long as hold is asserted further arbitration is disabled



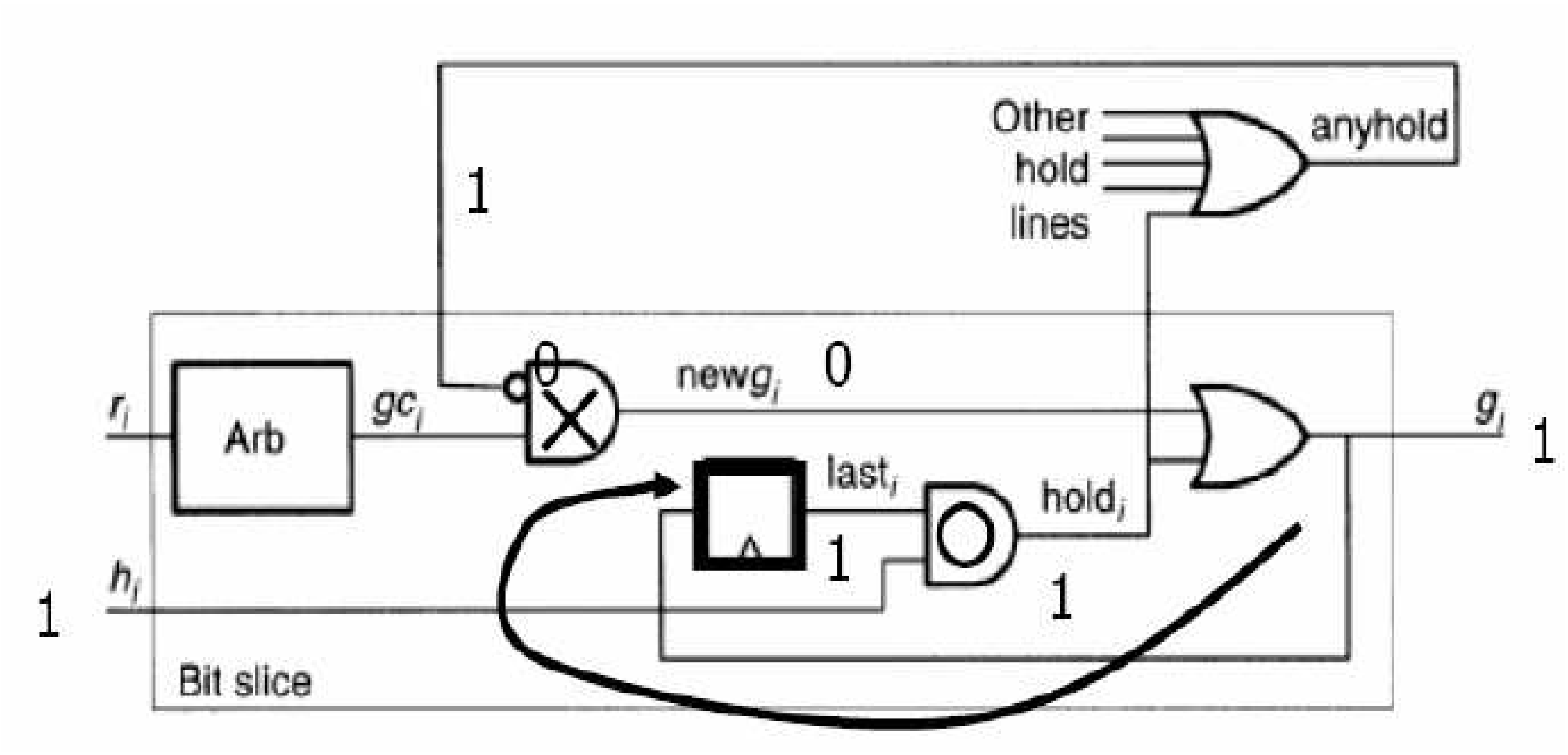
Grant-Hold Circuit

- An example for all *h*i=0



Grant-Hold Circuit

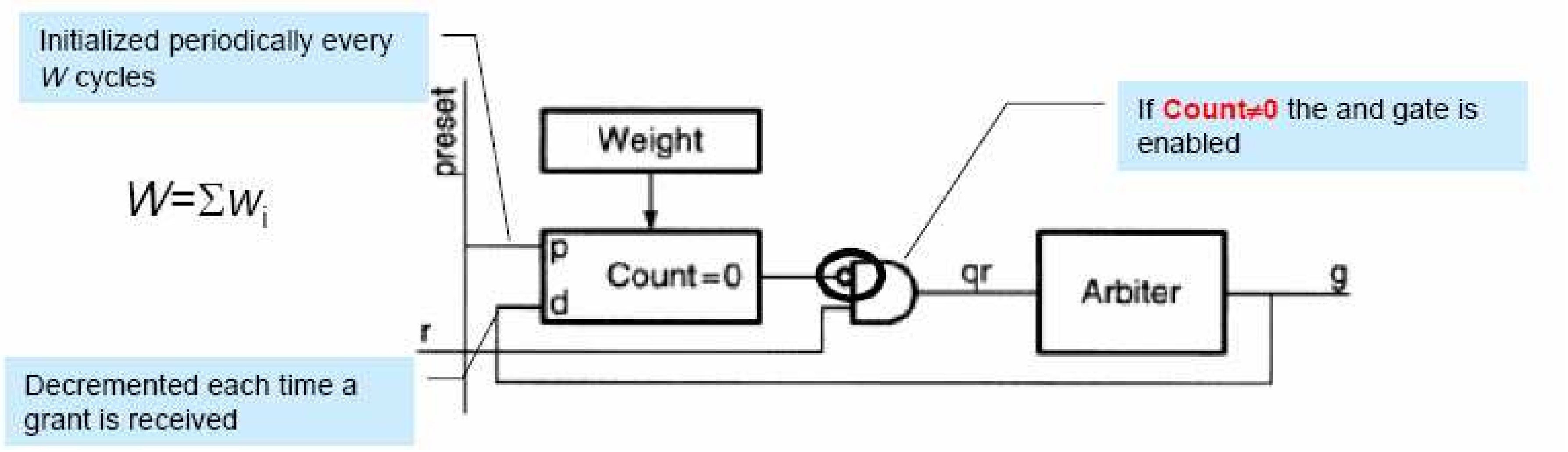
- An example for holding a grant



SEMICON
THINKING OUT OF THE BOX

Weighted Round-Robin Arbiter

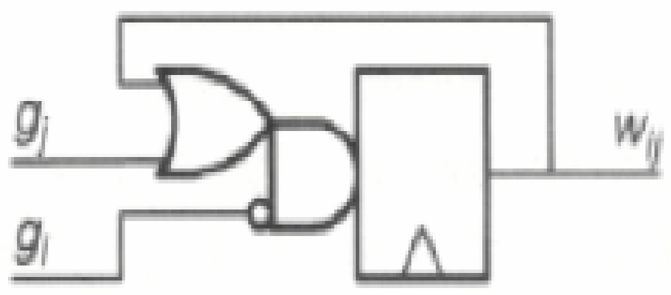
- A weighted round-robin arbiter allows to give requesters a larger number of grants than other requesters in a controlled fashion
- If three devices have the weight 1,2,3 they get 1/6, 1/3 and 1/2 of the grants
- The *preset* line is activated periodically after *N* (here 6 cycles) to load the counter with its weight
- If some arbiters do not issue any requests during that interval, the shared resource will remain idle until the next preset cycle

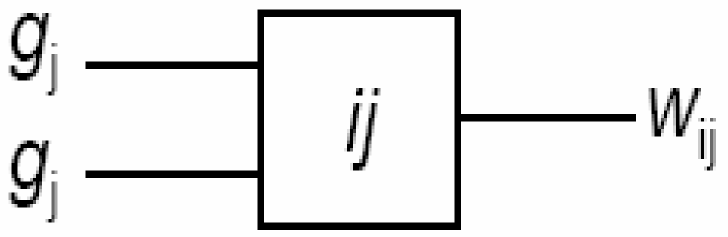

Initialized periodically every *W* cycles
$W=\Sigma w_i$
preset
Weight
p
Count=0
d
qr
Arbiter
g
r
If Count≠0 the and gate is enabled
Decremented each time a grant is received

SEMICON

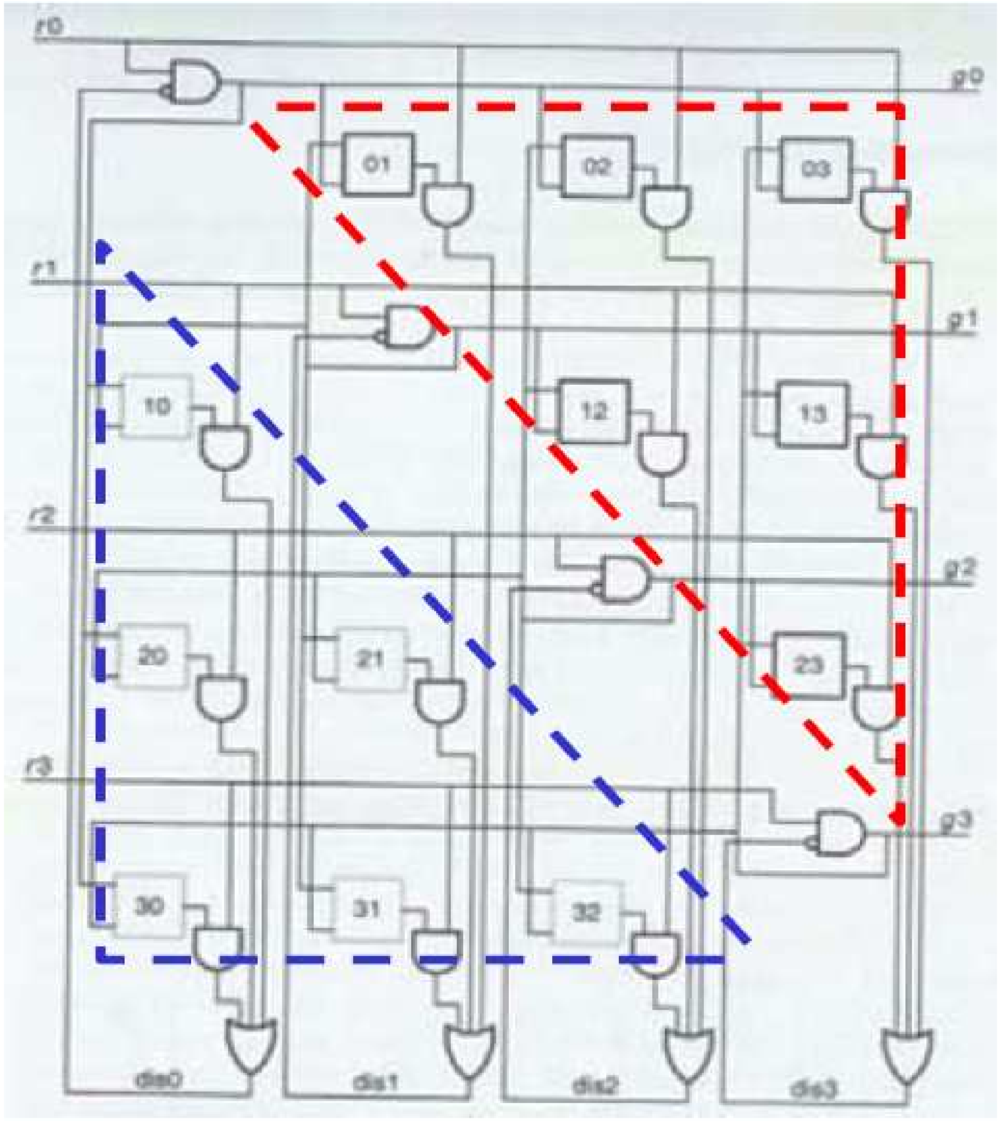
Matrix Arbiter

- Matrix *W*=[*w*ij] of weights
- *w*ij=1 if request *i* take priority over *j*





Matrix Arbiter



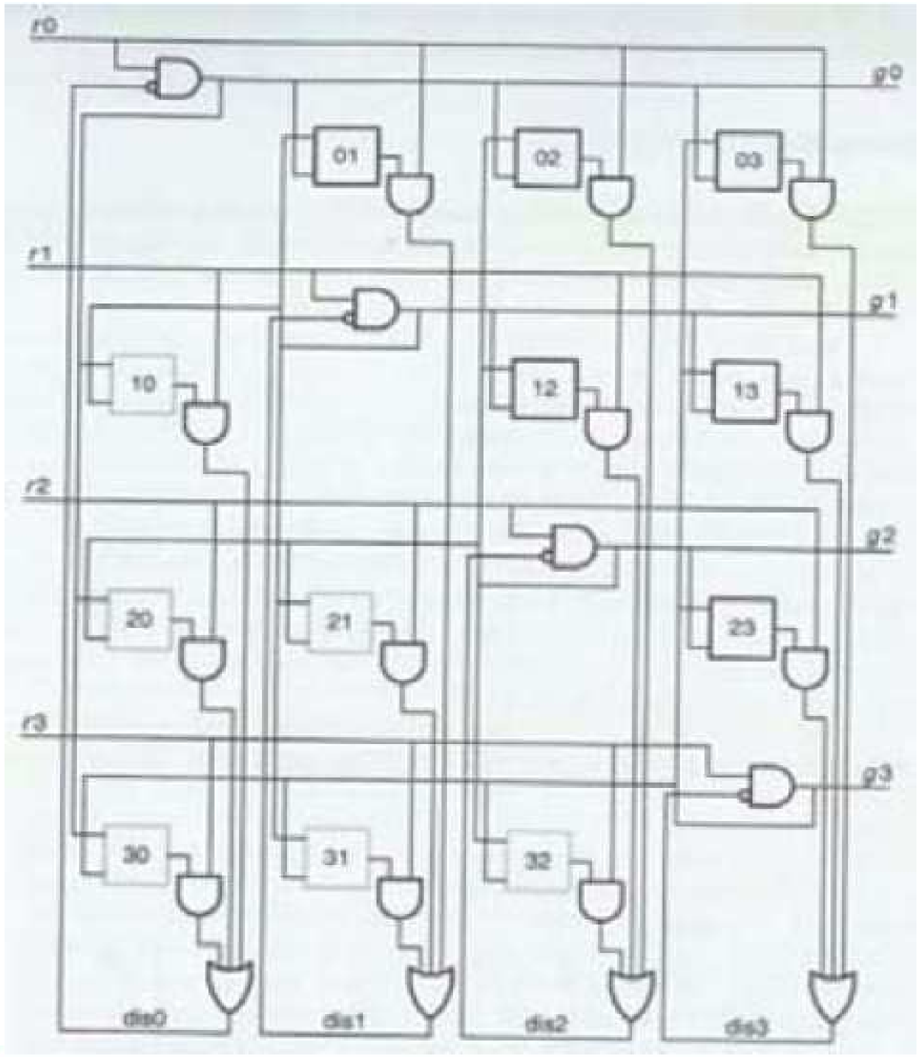
- *w*ij = ¬*w*ij ∀ *i*≠*j*
- A requester will be granted the resource if no other higher priority requester is bidding for the same resource
- Once a requester succeeds in being granted a resource, its priority is updated and set to be the lowest among all requesters
- Request *i* granted
  - [i,*] ← 0
  - [*,i] ← 1

SEMICON
THINKING OUT OF THE BOX

Matrix Arbiter

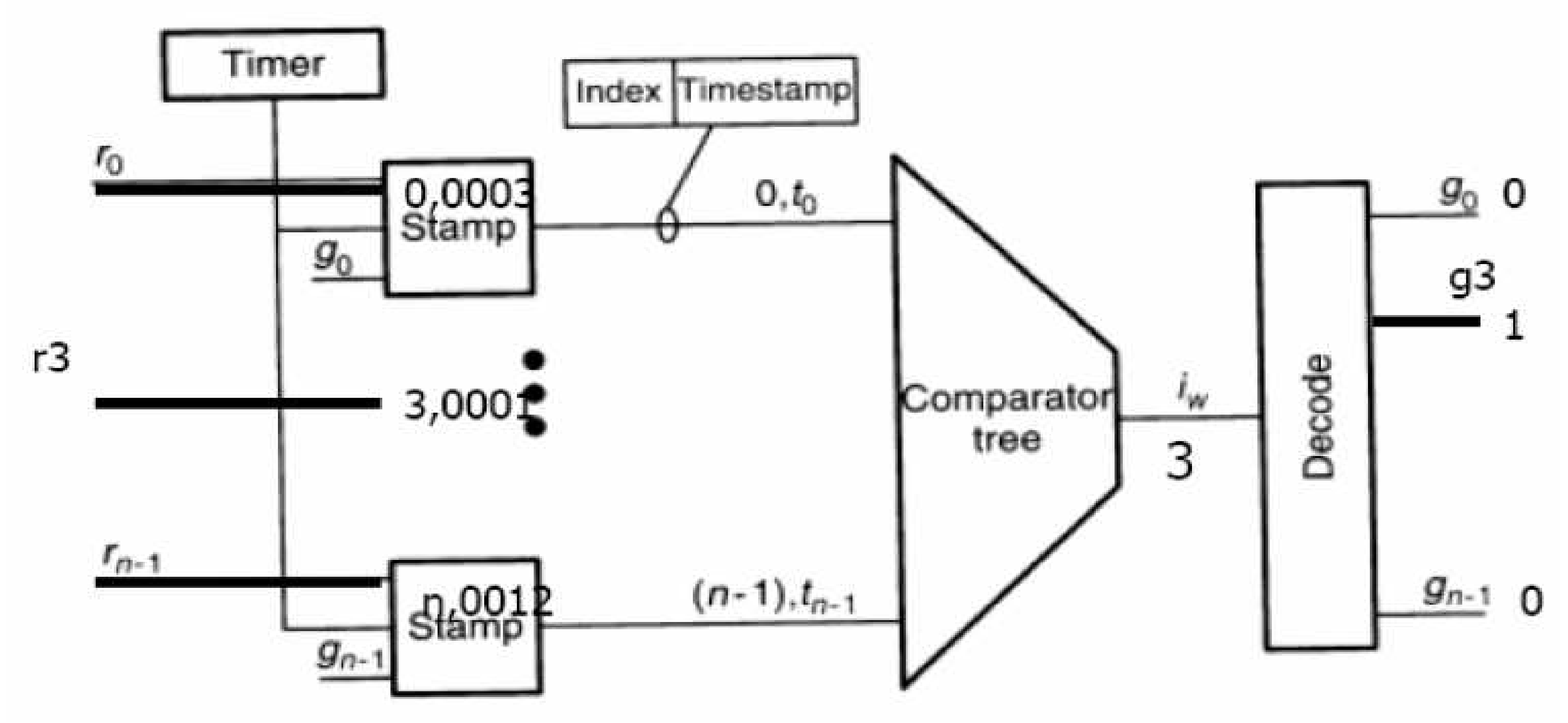
- A matrix arbiter implements a least recently served priority scheme by maintaining a triangular array of state bits *wij* for all *I* < *j*

- The Matrix arbiter is very good suited for a small number of inputs, since it is fast, easy to implement and provides strong fairness!



Queuing Arbiter

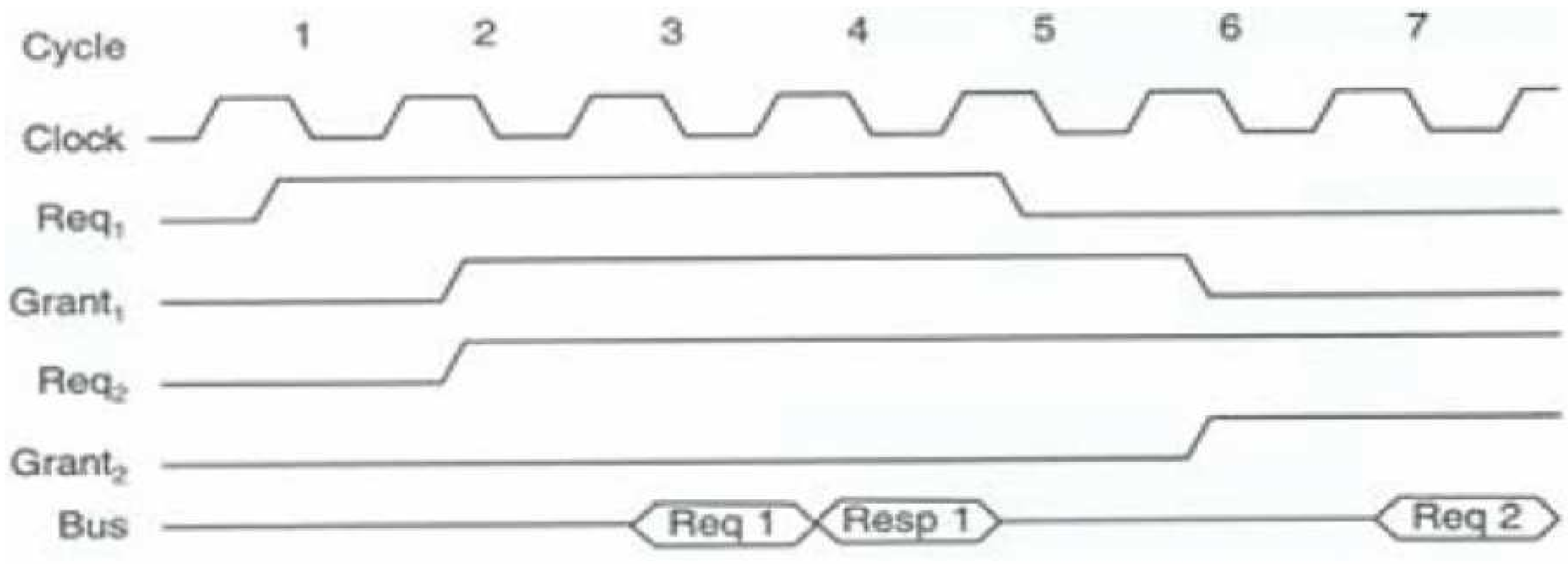
- A queuing arbiter provides FIFO fairness
- It assigns each request a time stamp when it is asserted
- The request with the earliest time stamp receives the grant



SEMICON

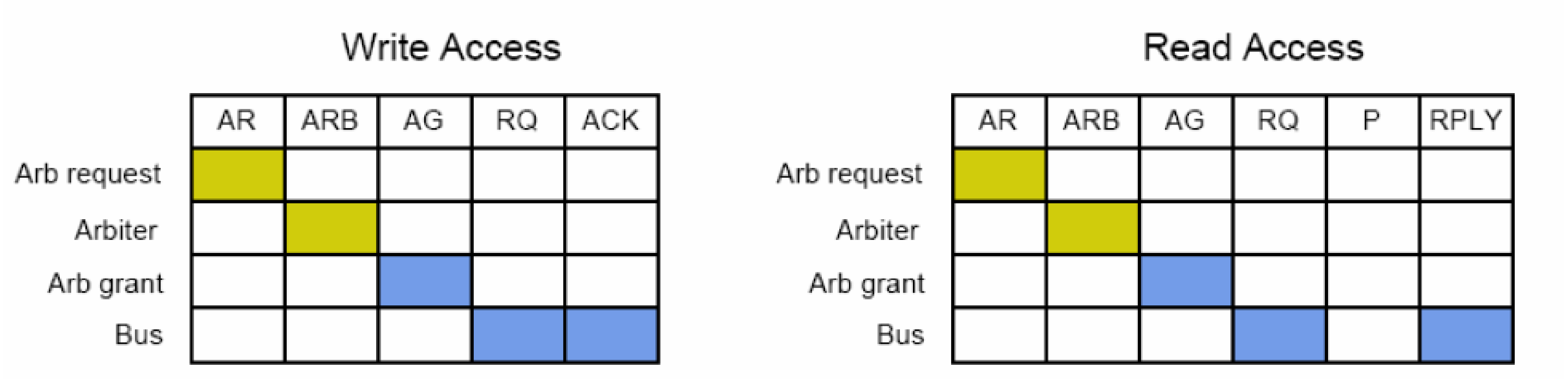
Low Performance Bus Protocol

- Without a special bus protocol the bus is not efficiently used
- In the example module 2 requests the bus in cycle 2, but must wait until cycle 6 to receive the grant



Bus Pipelining

- A memory access consists of several cycles (including arbitration)
- Since the bus is not used in all cycles, pipelining can be used to increase the performance



Write Access

	AR	ARB	AG	RQ	ACK
Arb request	🟨				
Arbiter		🟨			
Arb grant			🟦		
Bus				🟦	🟦

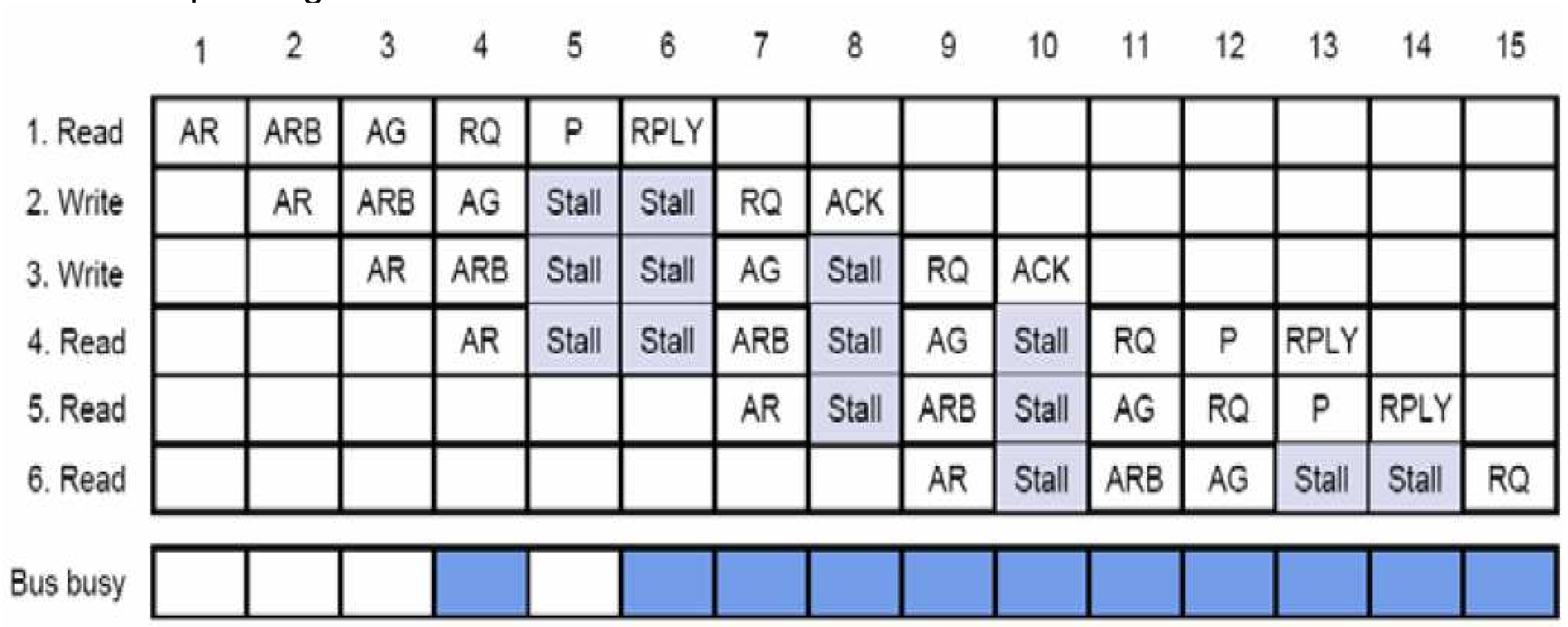
Read Access

	AR	ARB	AG	RQ	P	RPLY
Arb request	🟨					
Arbiter		🟨				
Arb grant			🟦			
Bus				🟦		🟦

- Only one transaction can
  - Receive the grant during a given cycle
  - Use the bus during a given cycle

Bus Pipelining

- Pipelining leads to an efficient use of the bus



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. Read	AR	ARB	AG	RQ	P	RPLY									
2. Write		AR	ARB	AG	Stall	Stall	RQ	ACK							
3. Write			AR	ARB	Stall	Stall	AG	Stall	RQ	ACK					
4. Read				AR	Stall	Stall	ARB	Stall	AG	Stall	RQ	P	RPLY		
5. Read							AR	Stall	ARB	Stall	AG	RQ	P	RPLY	
6. Read									AR	Stall	ARB	AG	Stall	Stall	RQ
Bus busy				🟦		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

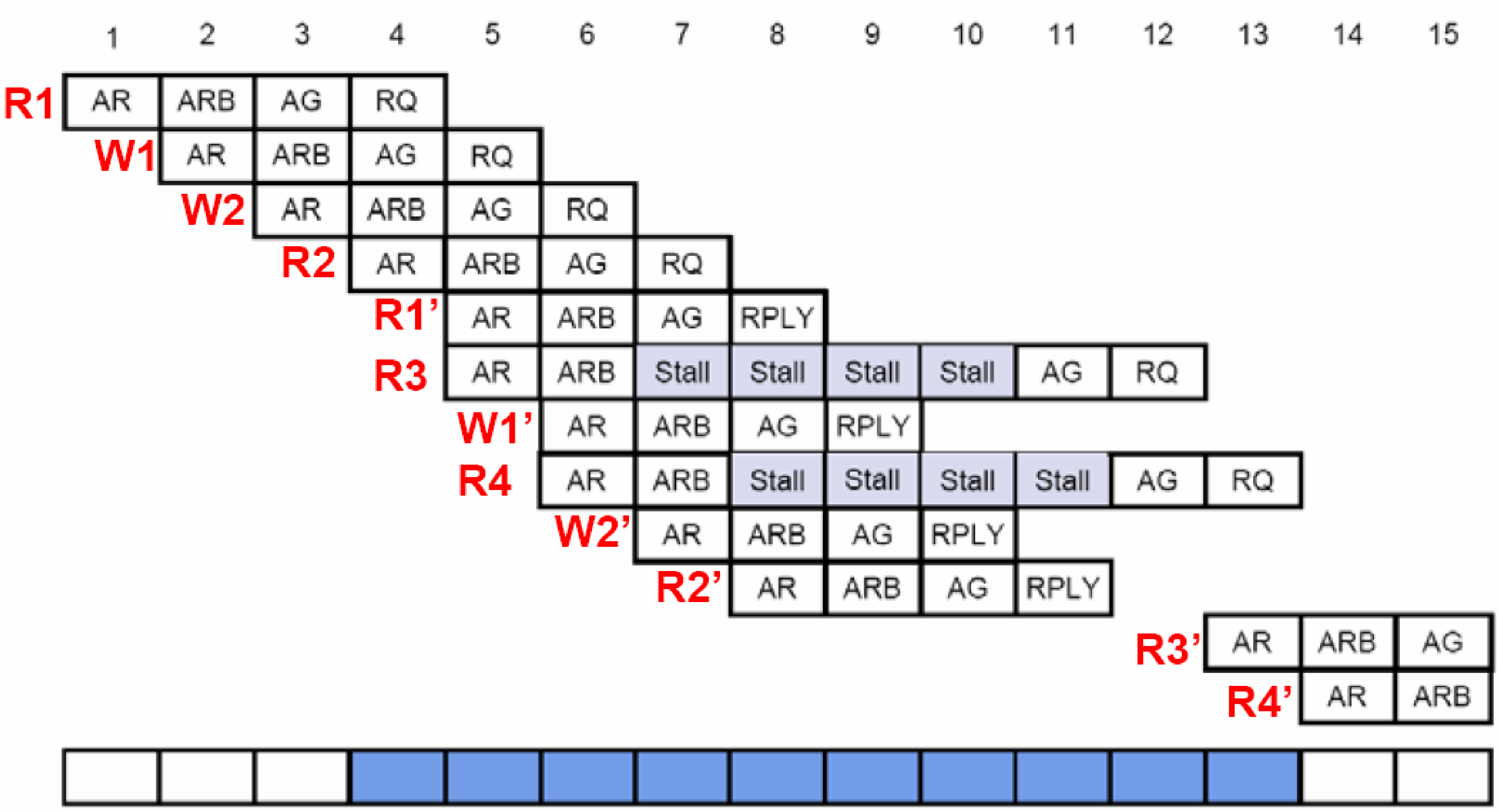
- Stalls are inserted since only one instance can use the bus
- Sometimes (cycle 12) two transactions can overlap
- However this cannot be done in cycle 5 (2. Write) since otherwise RPLY and ACK would overlap in cycle 6!

SEMICON
THINKING OUT OF THE BOX

Split-Transaction Bus

- In a split-transaction bus a transaction is splitted into a two transactions
  - ”request”-transaction
  - ”reply”-transaction
- Both transactions have to compete for the bus by arbitration

Split-Transaction Bus

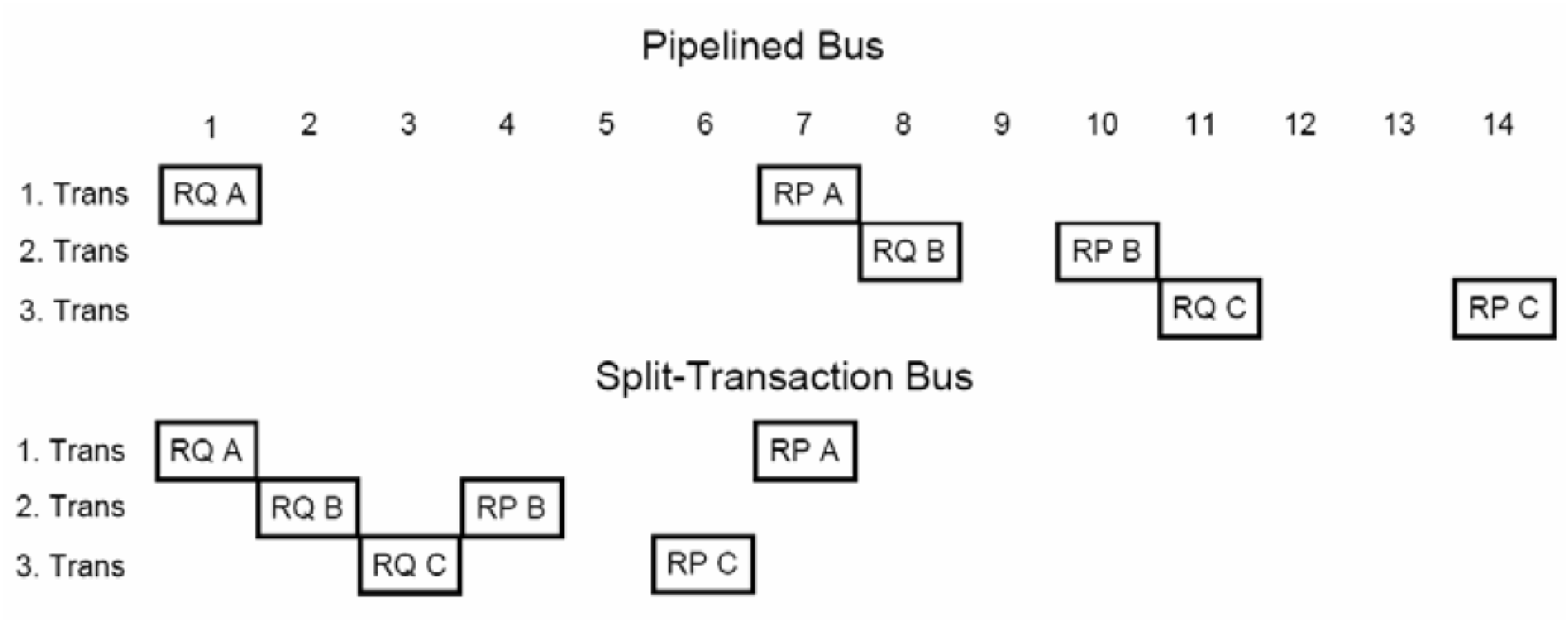


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R1	AR	ARB	AG	RQ											
W1		AR	ARB	AG	RQ										
W2			AR	ARB	AG	RQ									
R2				AR	ARB	AG	RQ								
R1’					AR	ARB	AG	RPLY							
R3					AR	ARB	Stall	Stall	Stall	Stall	AG	RQ			
W1’						AR	ARB	AG	RPLY						
R4						AR	ARB	Stall	Stall	Stall	Stall	AG	RQ		
W2’							AR	ARB	AG	RPLY					
R2’								AR	ARB	AG	RPLY				
R3’													AR	ARB	AG
R4’														AR	ARB
				🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		

SEMICON

Split-Transaction Bus

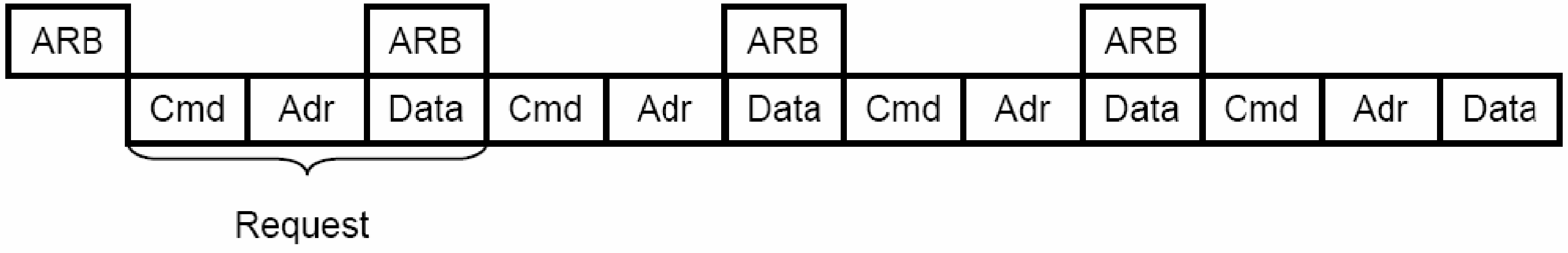
- The advantages of the split-transaction bus are evident, if there is a variable delay for requests, since then transactions cannot overlap



SEMICON
THINKING OUT OF THE BOX

Burst Messages

- There is a considerable amount of overhead in a bus transaction
  - Arbitration
  - Addressing
  - Acknowledgement

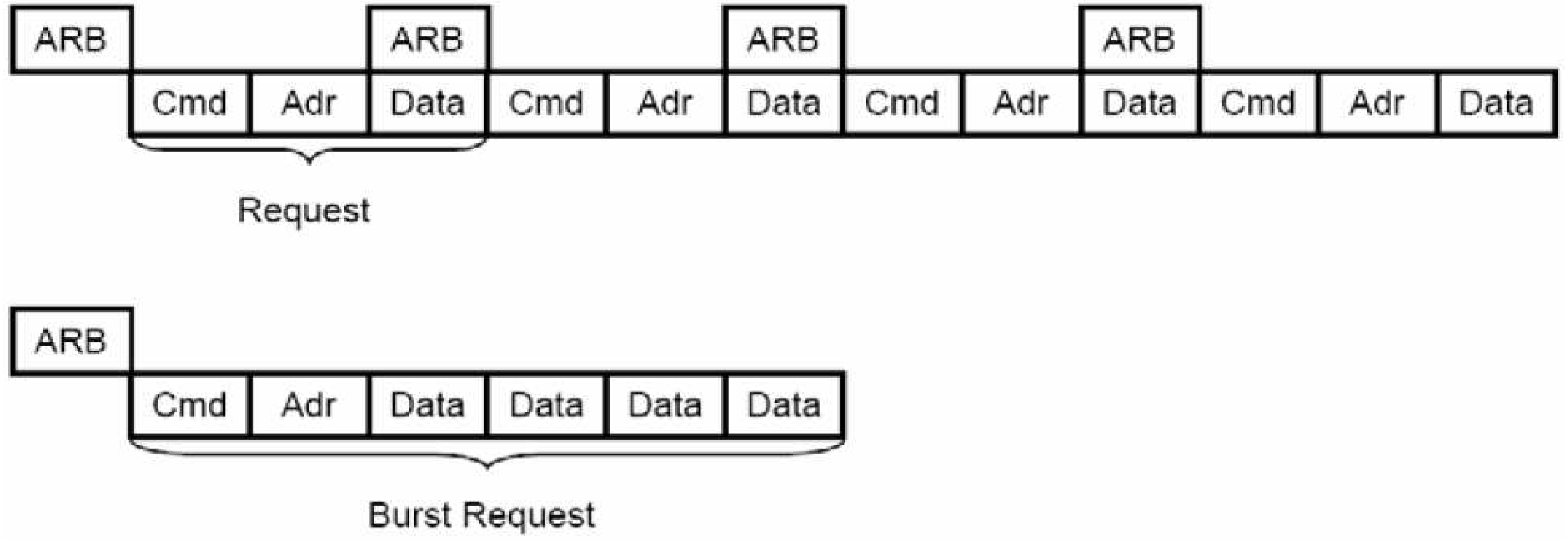


Efficiency = Transmitted Words / Message Size = 1/3

SEMICON

Burst Messages

- The overhead can be reduced, if messages are sent as blocks (bursts)

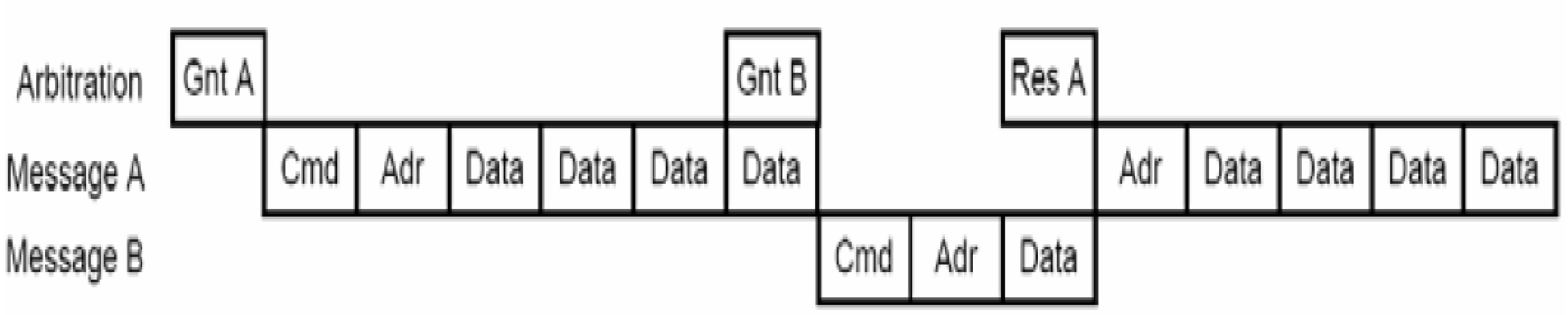


Efficiency = Transmitted Words / Message Size = 2/3

SEMICON

Burst Messages

- The longer the burst, the better the efficiency
- BUT
  - Other bus masters have to wait, which may be unacceptable in many systems (Real-Time)
- Possible solution
  - Maximum length for a burst
  - Interrupt of long messages
    - Restart or Resume



Embedded Busses

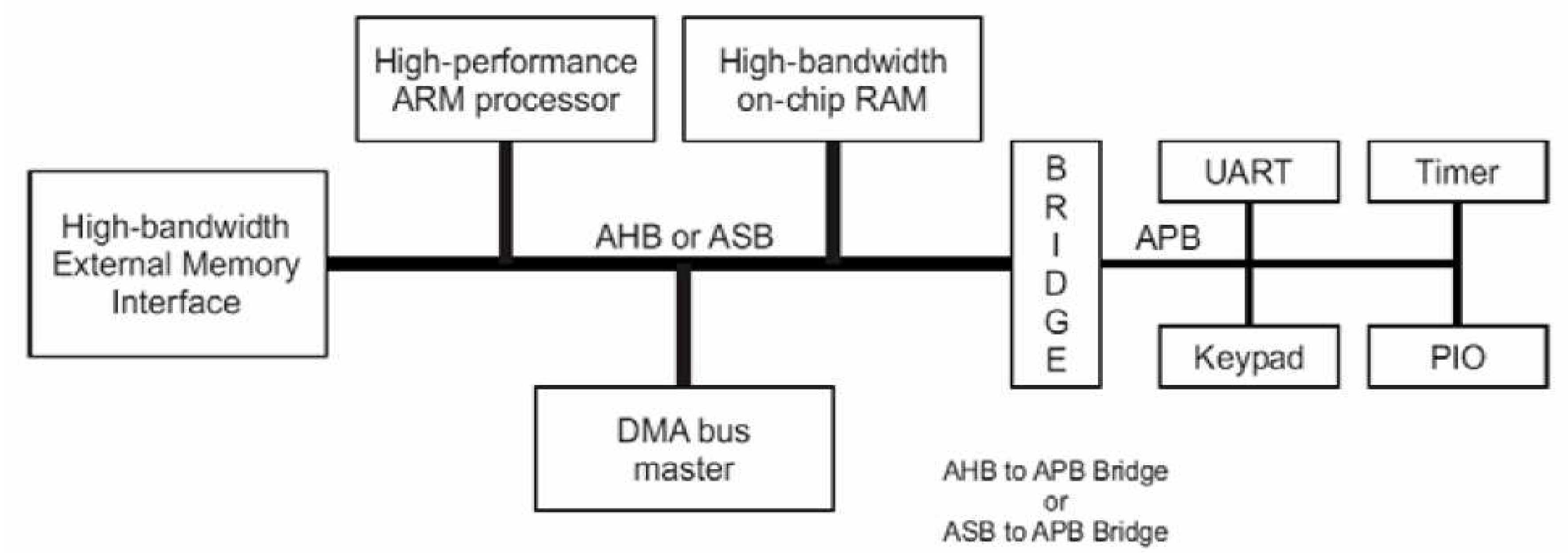
- Current system-on-chips are advanced enough to need a hierarchy of busses
- A new set of bus standards have been defined to be used in SoCs, e.g.
  - ARM Amba
  - STBus
  - Altera Avalon
  - ...
- These busses allow for higher performance than traditional Tri-State busses

AMBA Specifications

- The AMBA specification defines an on-chip communications standard for designing high performance embedded micro-controllers
- Three buses are defined
  - Advanced High-Performance Bus (AHB)
  - Advanced System Bus (ASB)
  - Advanced Peripheral Bus (APB)

System based on an AMBA Bus

- An AMBA system typically contains a high speed bus (ASB or AHB) for CPU, fast memory and DMA and a bus for peripherals (APB), which is connected via a bridge to the high-speed bus


High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth External Memory Interface
AHB or ASB
DMA bus master
BRIDGE
UART
Timer
APB
Keypad
PIO
AHB to APB Bridge or ASB to APB Bridge

SEMICON

AMBA Buses

- AMBA AHB (new standard)
  - High Performance
  - Pipelined Operation
  - Multiple Bus Masters
  - Burst Transfers
  - Split Transactions
- AMBA ASB (older standard)
  - High Performance
  - Pipelined Operation
  - Multiple Bus Masters
- AMBA APB
  - Low Power
  - Latched Address and Control
  - Simple Interface
  - Suitable for many peripherals

AMBA AHB System

- AHB Master
  - A bus master is able to initiate read and write information by providing address and control information. Only one bus master can use the bus at the same time
- AHB Slave
  - A bus slave responds to a read and write operation within a given address-space range. The bus slave signals back to the active bus master the success, failure or waiting of the data transfer

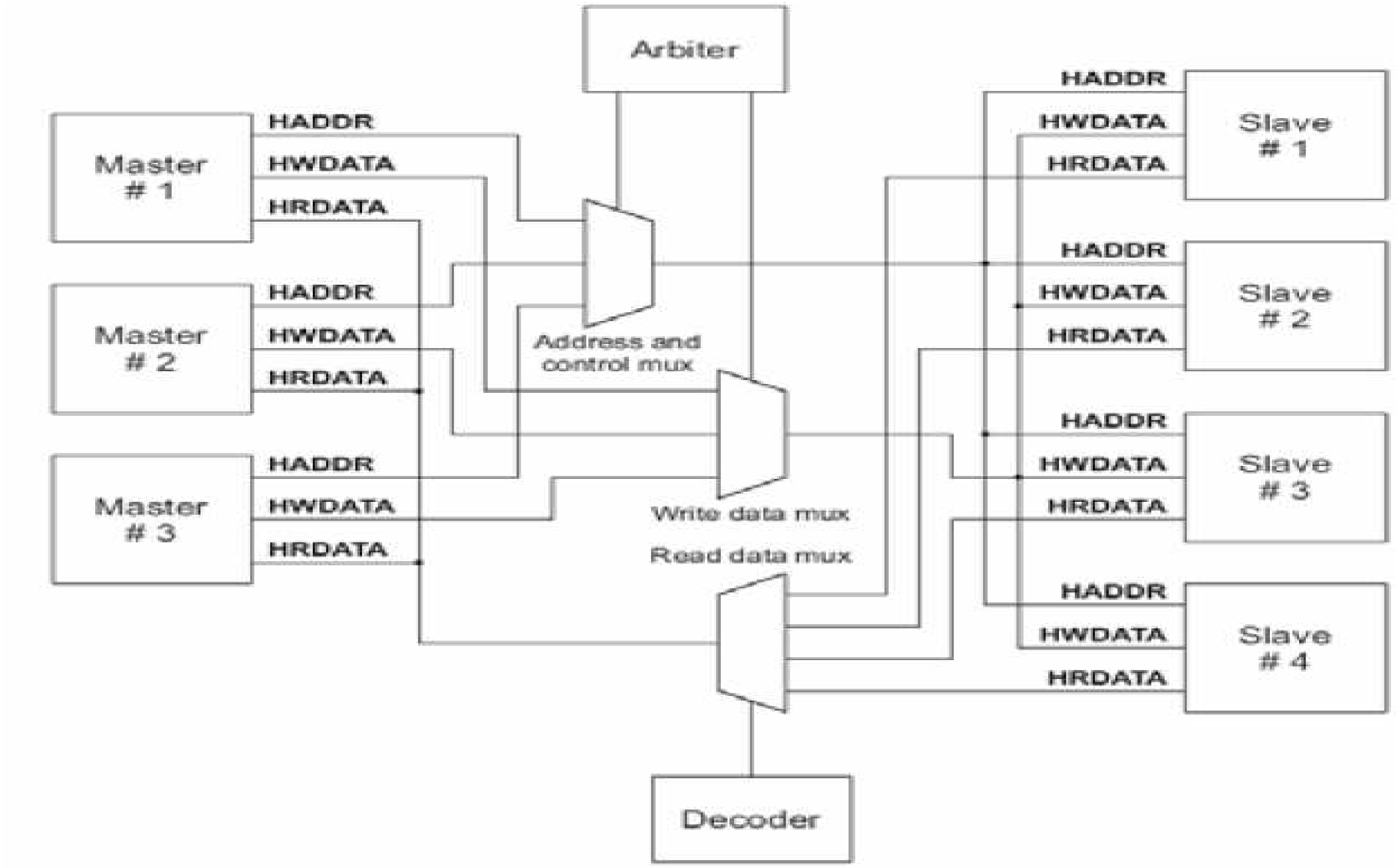
AMBA AHB System

- AHB Arbiter
  - The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as highest priority or fair access can be implemented depending on the application requirements
  - An AHB includes only one arbiter
- AHB Decoder
  - The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer
  - A single centralized decoder is required in all AHB implementations

AMBA AHB Bus Interconnection

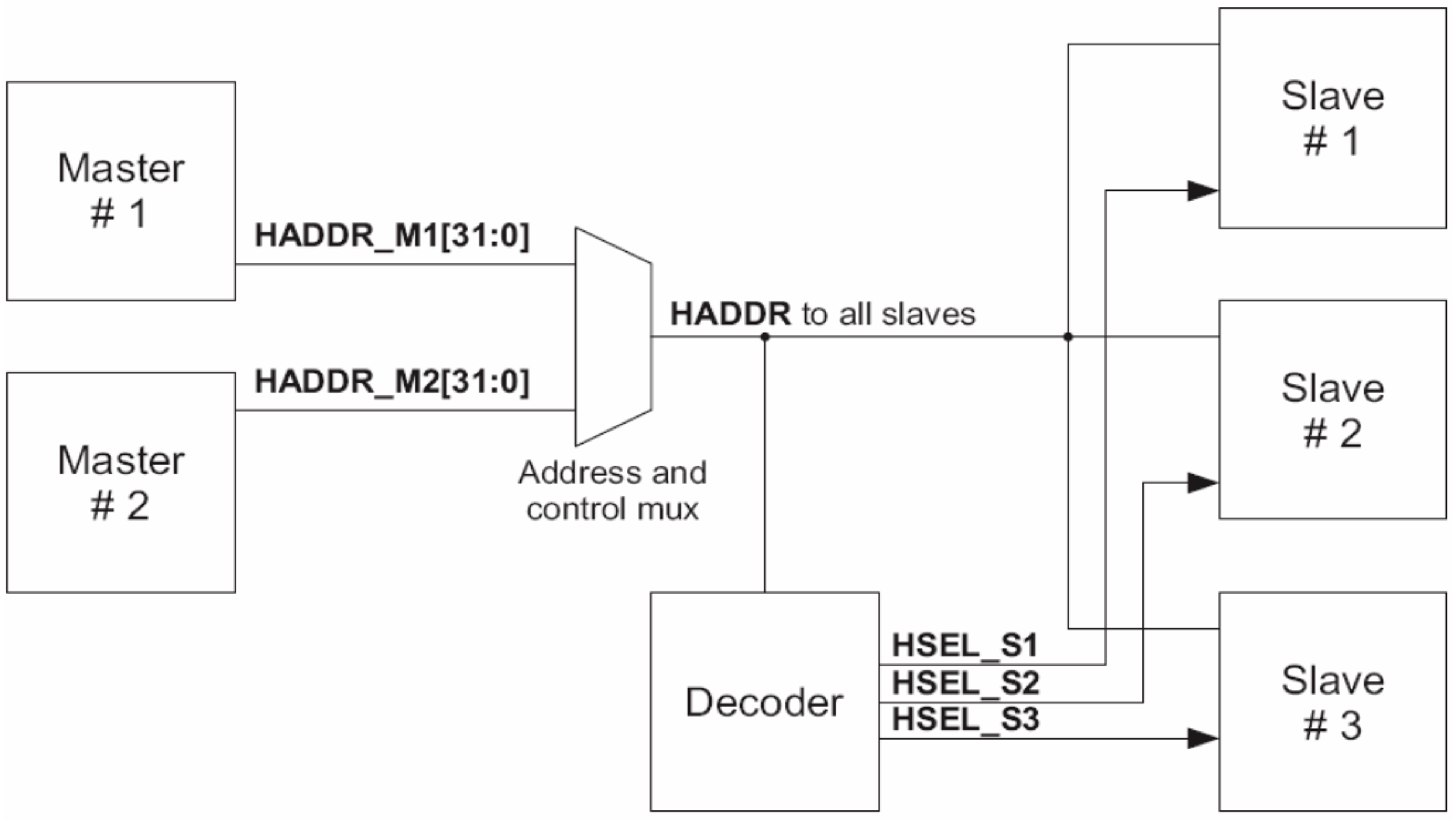
- AHB Protocol is based on a central multiplexer interconnection scheme
- All bus masters send their request in form of address and control signals
- The arbiter chooses one master. The address and control signals are routed to all slaves
- The decoder selects the signals from the slave that is involved in the transfer with the bus master

AMBA AHB Bus Interconnection



SEMICON
THINKING OUT OF THE BOX

AMBA Address Decoding System

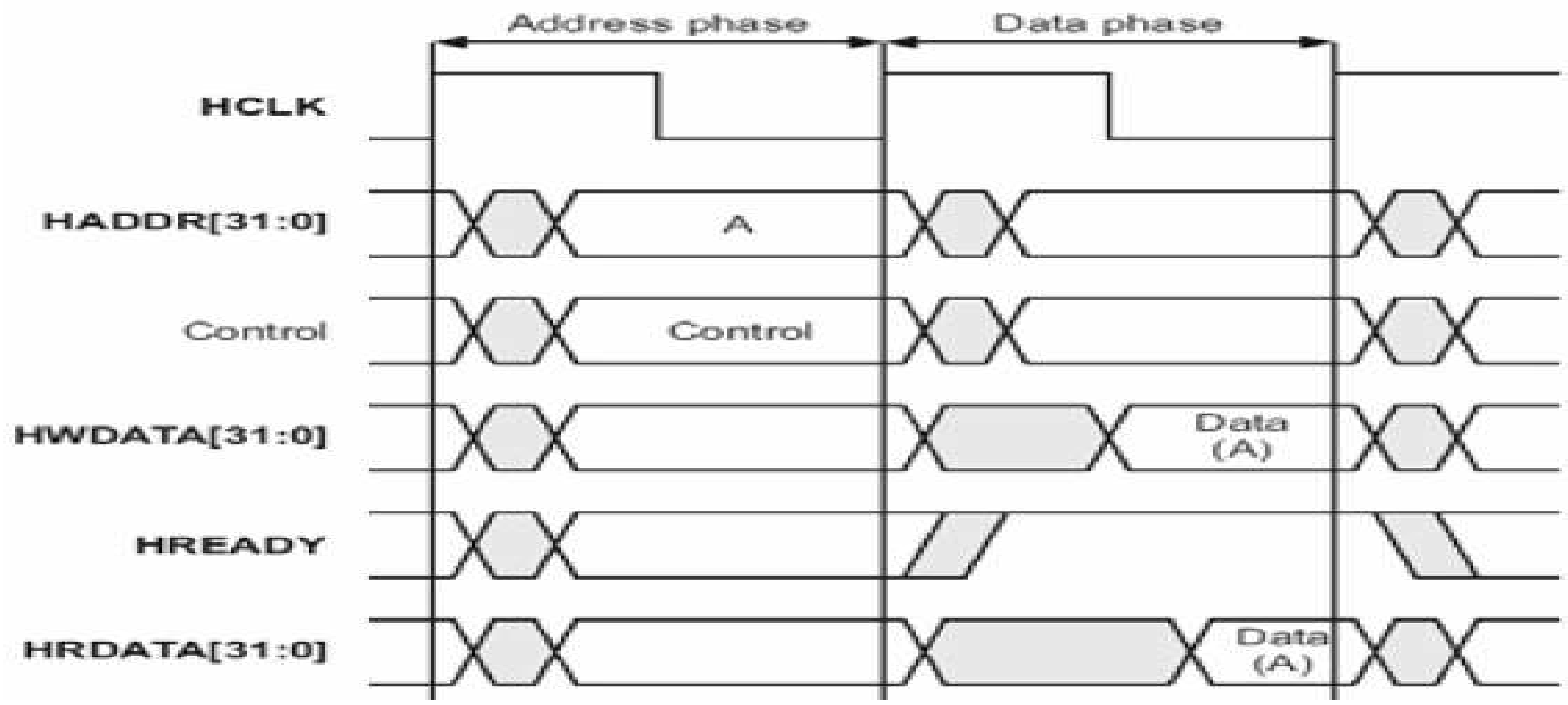


SEMICON

THINKING OUT OF THE BOX

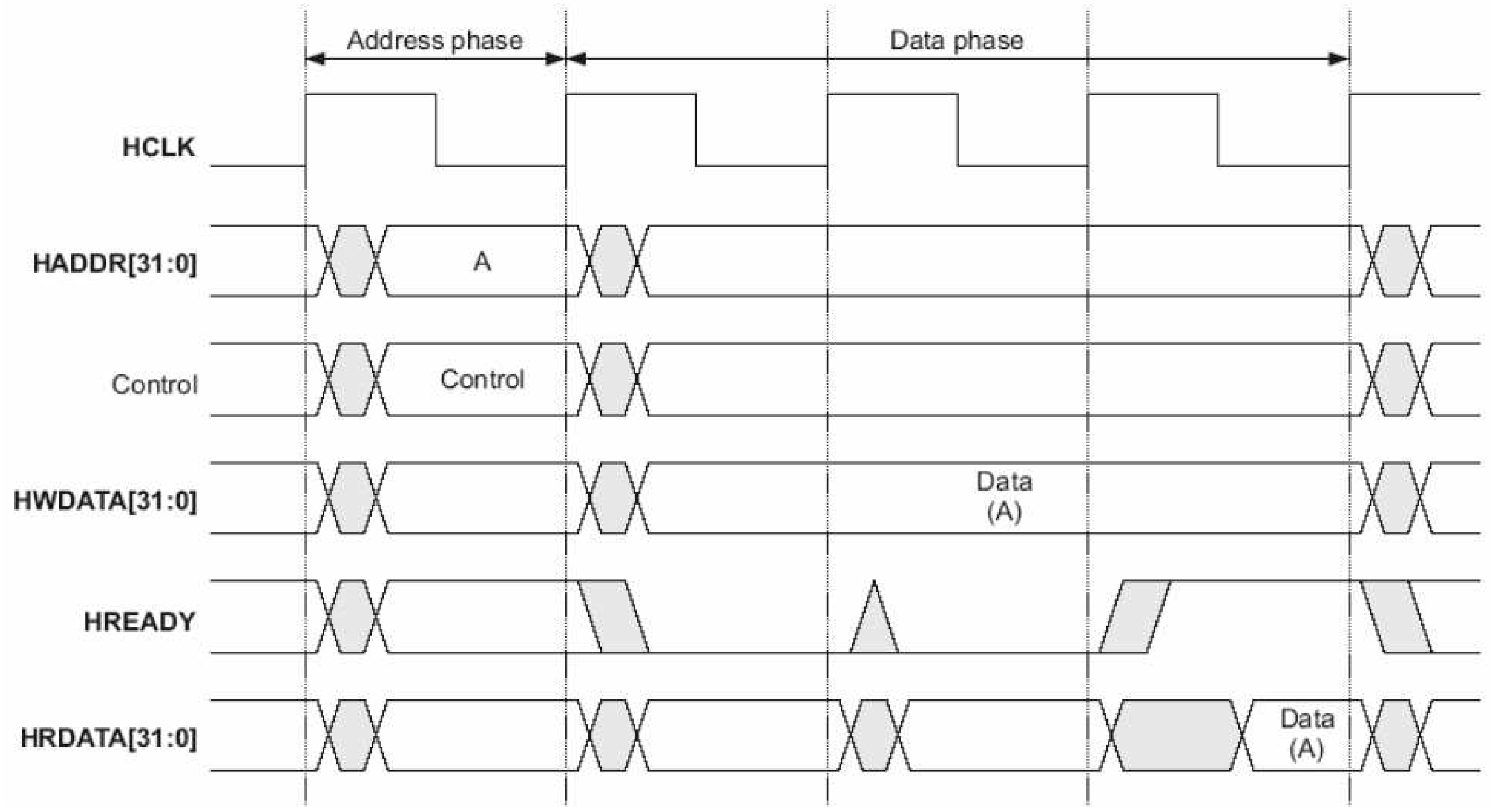
Basic Transfer

- An AHB transfer consists of two distinct sections
  - The address phase, which lasts only a single cycle
  - The data phase, which may require several cycles. This is achieved using the HREADY signal



SEMICON

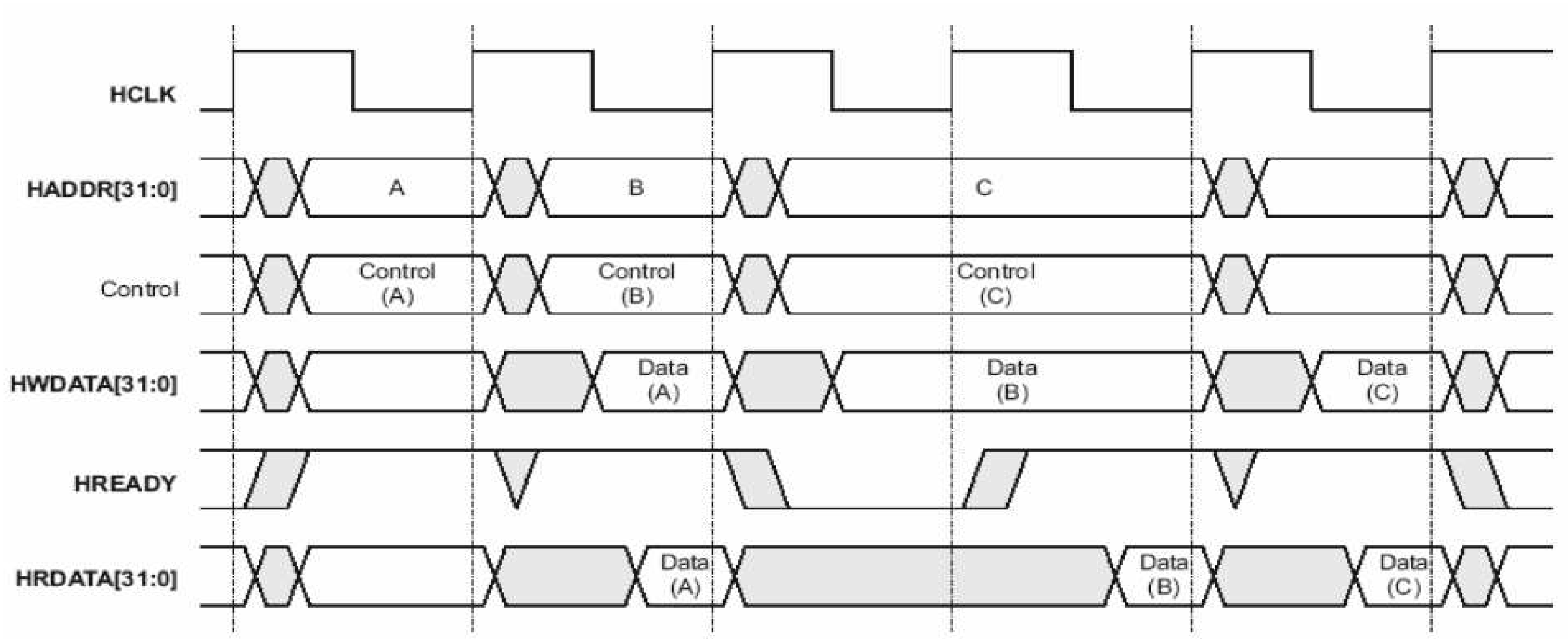
Transfer with wait states



SEMICON
THINKING OUT OF THE BOX

Multiple Transfers

- When a transfer is extended in this way it will have the side-effect of extending the address phase of the following transfer

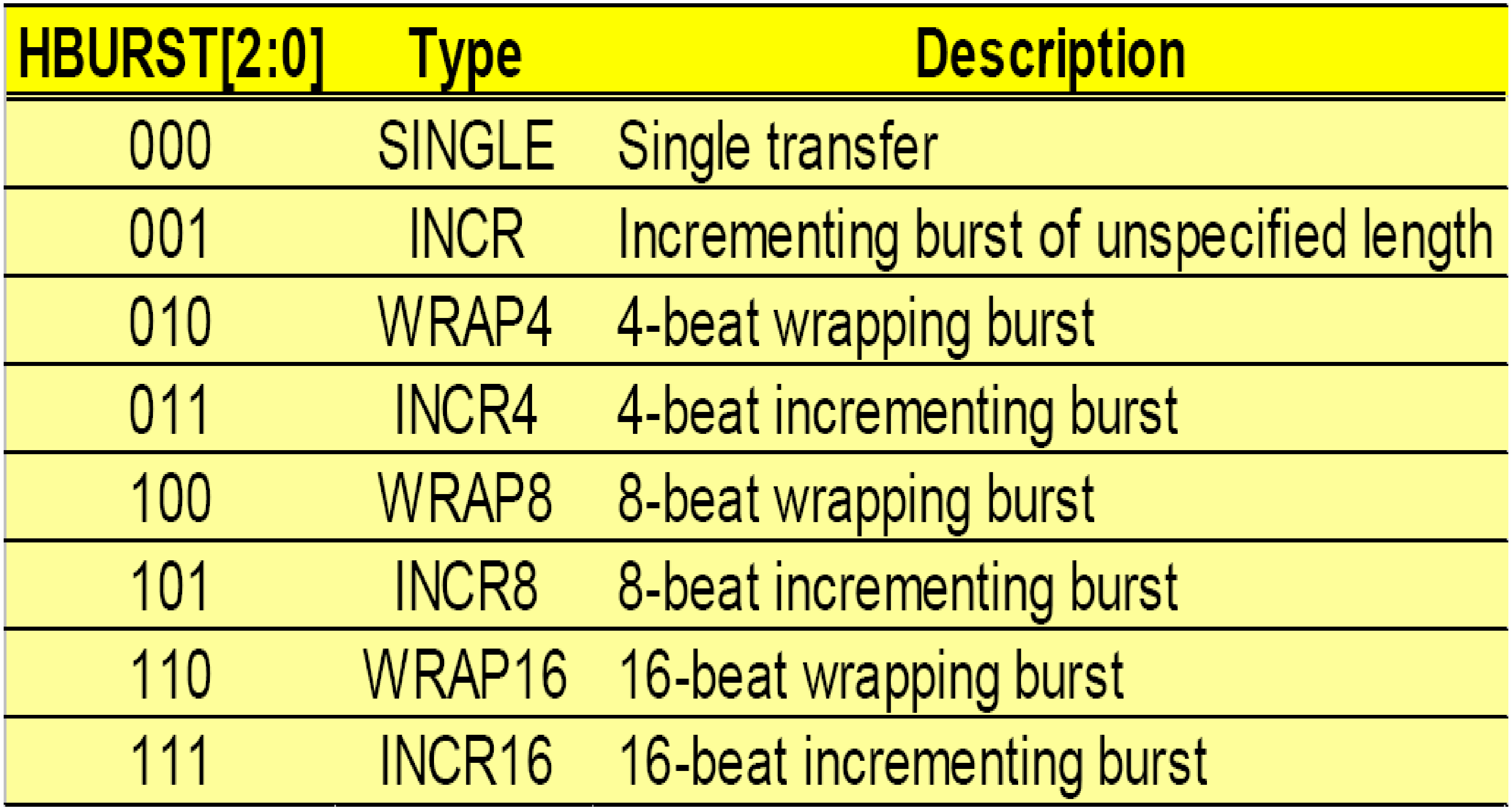


SEMICON

Burst Signal Encoding

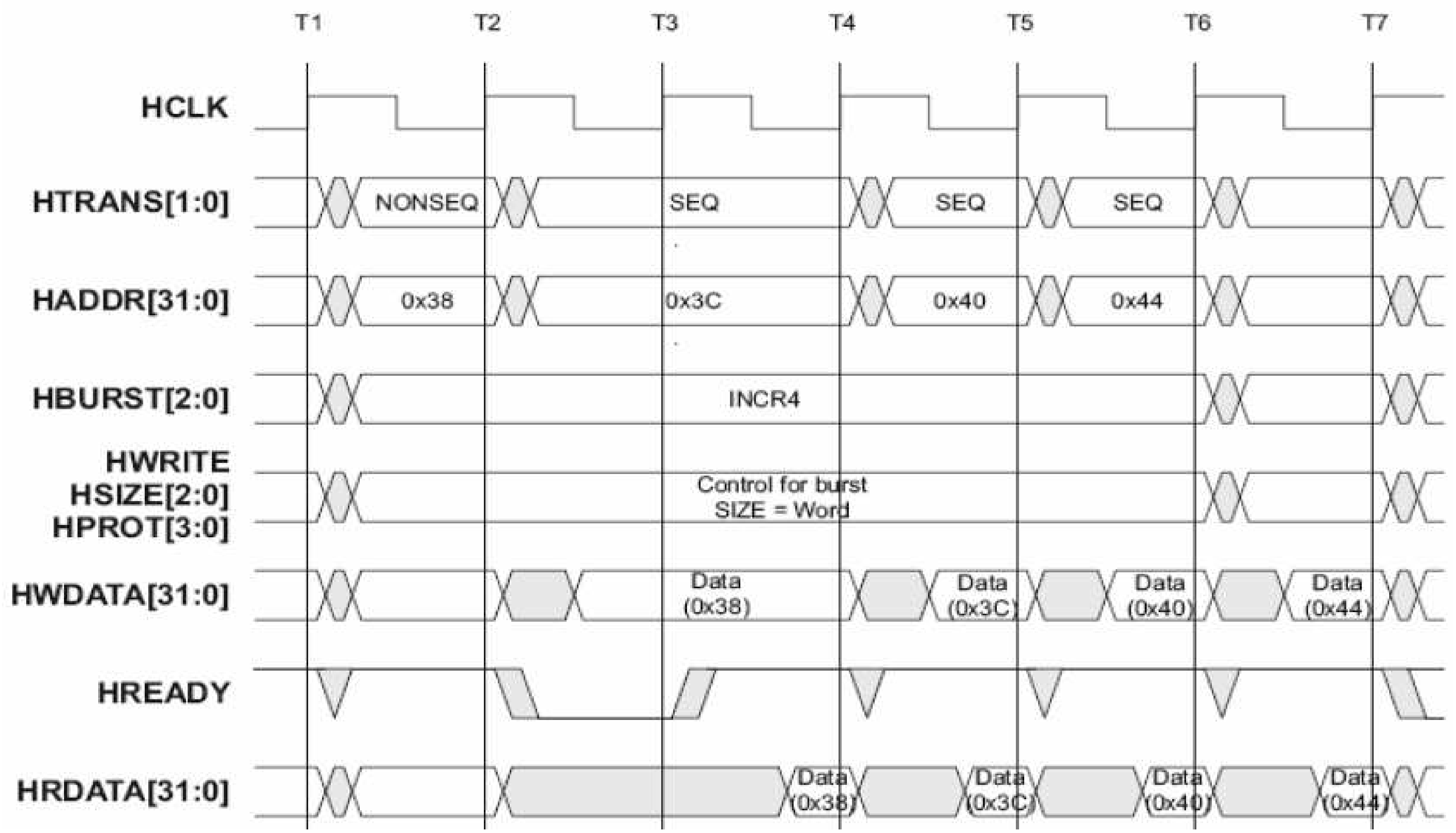
- Both incrementing and wrapping bursts are supported in the protocol
  - *Incrementing bursts* access sequential locations
  - For *wrapping bursts*, if the start address of the transfer is not aligned to the total number of bytes in the burst (size x beats) then the address of the transfers in the burst will wrap when the boundary is reached

Burst Signal Encoding

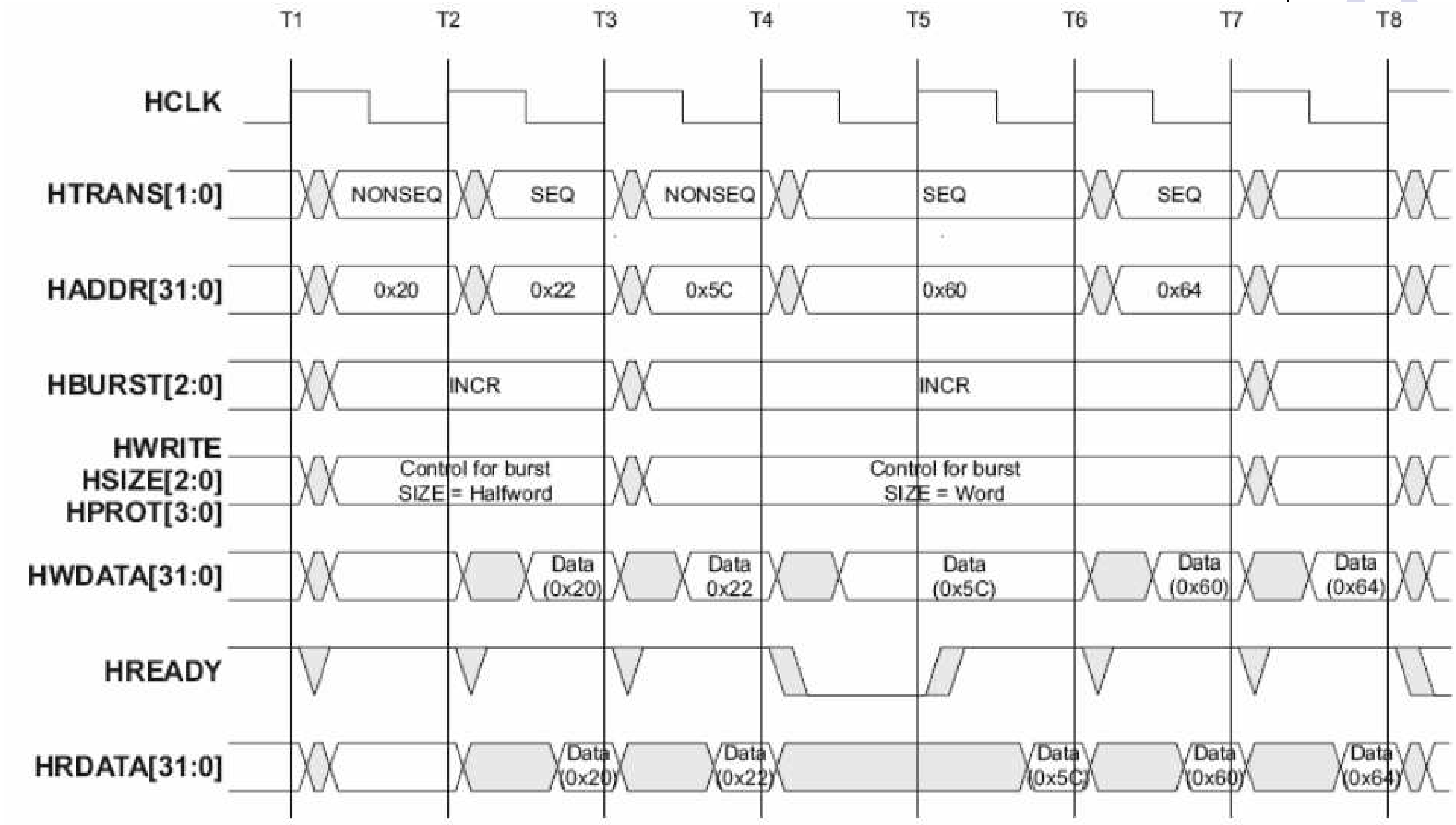


HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing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

Four-beat incrementing burst

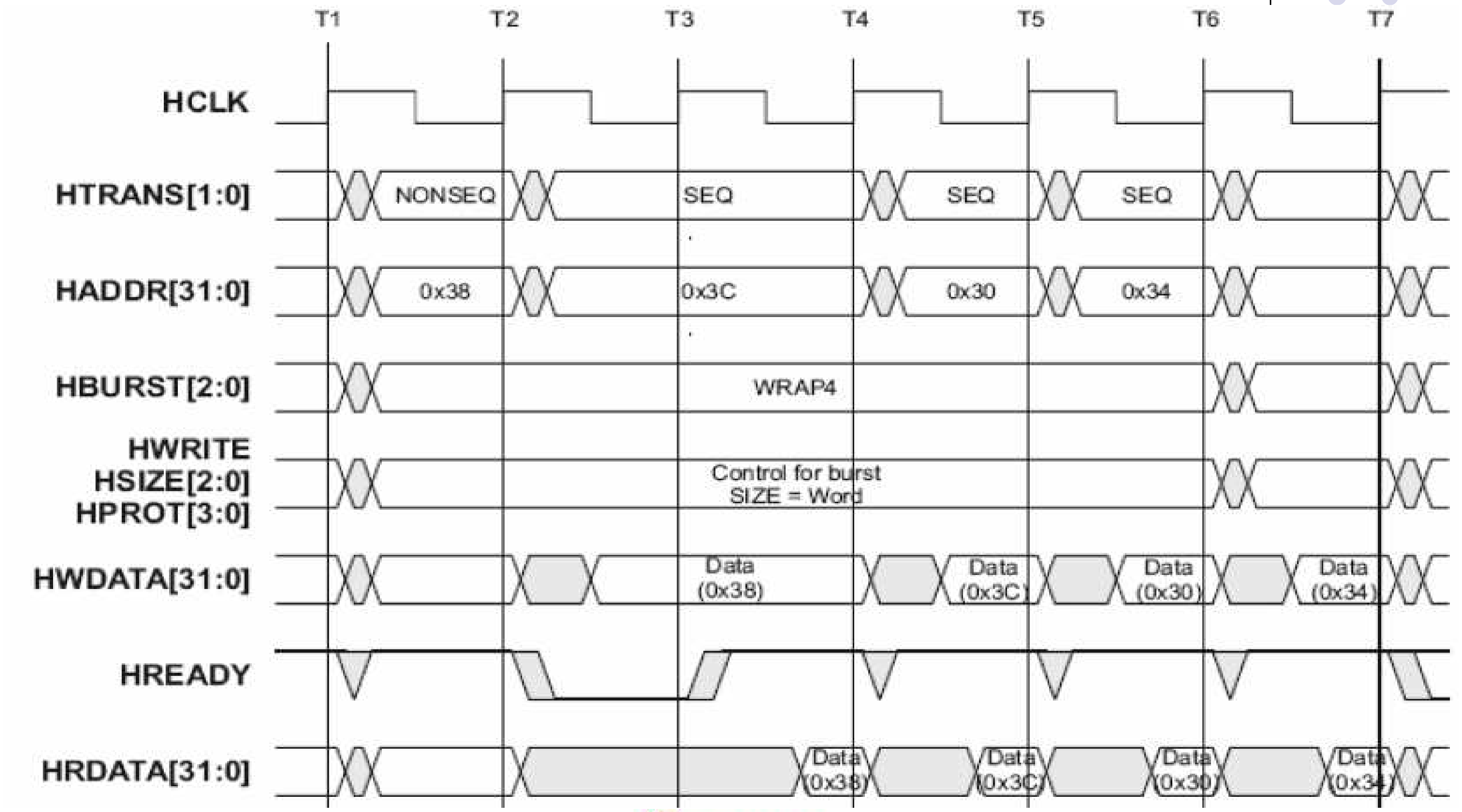


Undefined-length bursts



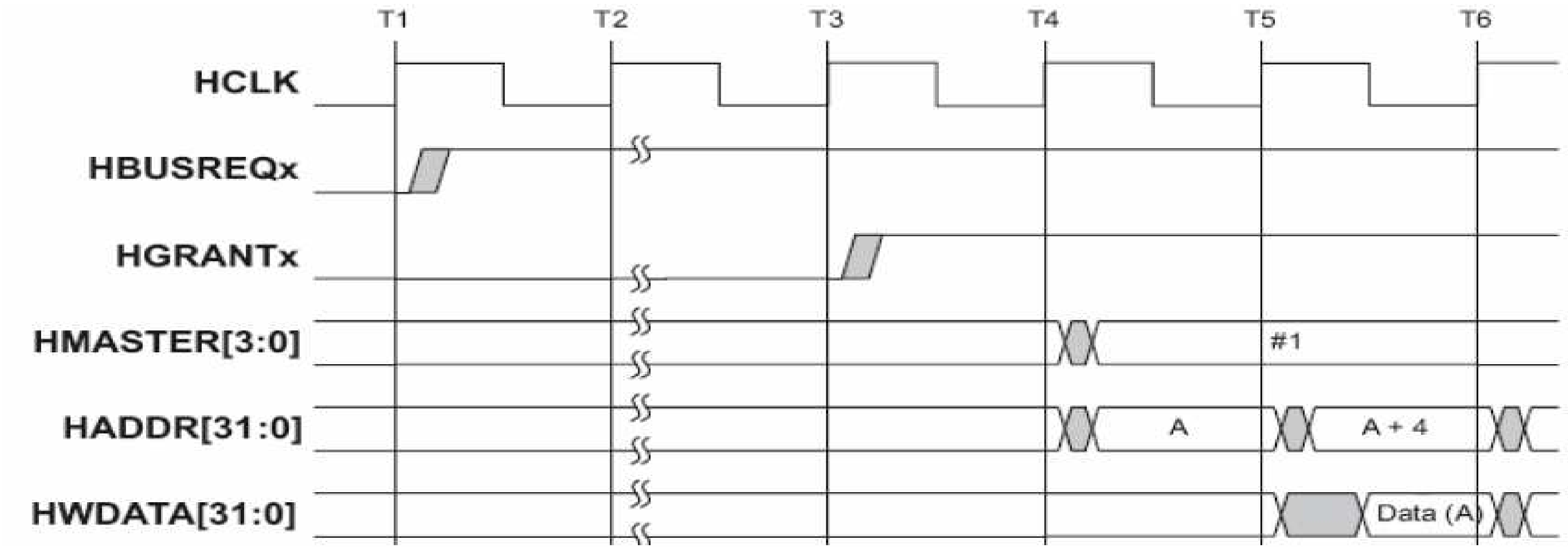
SEMICON

Four-beat wrapping burst



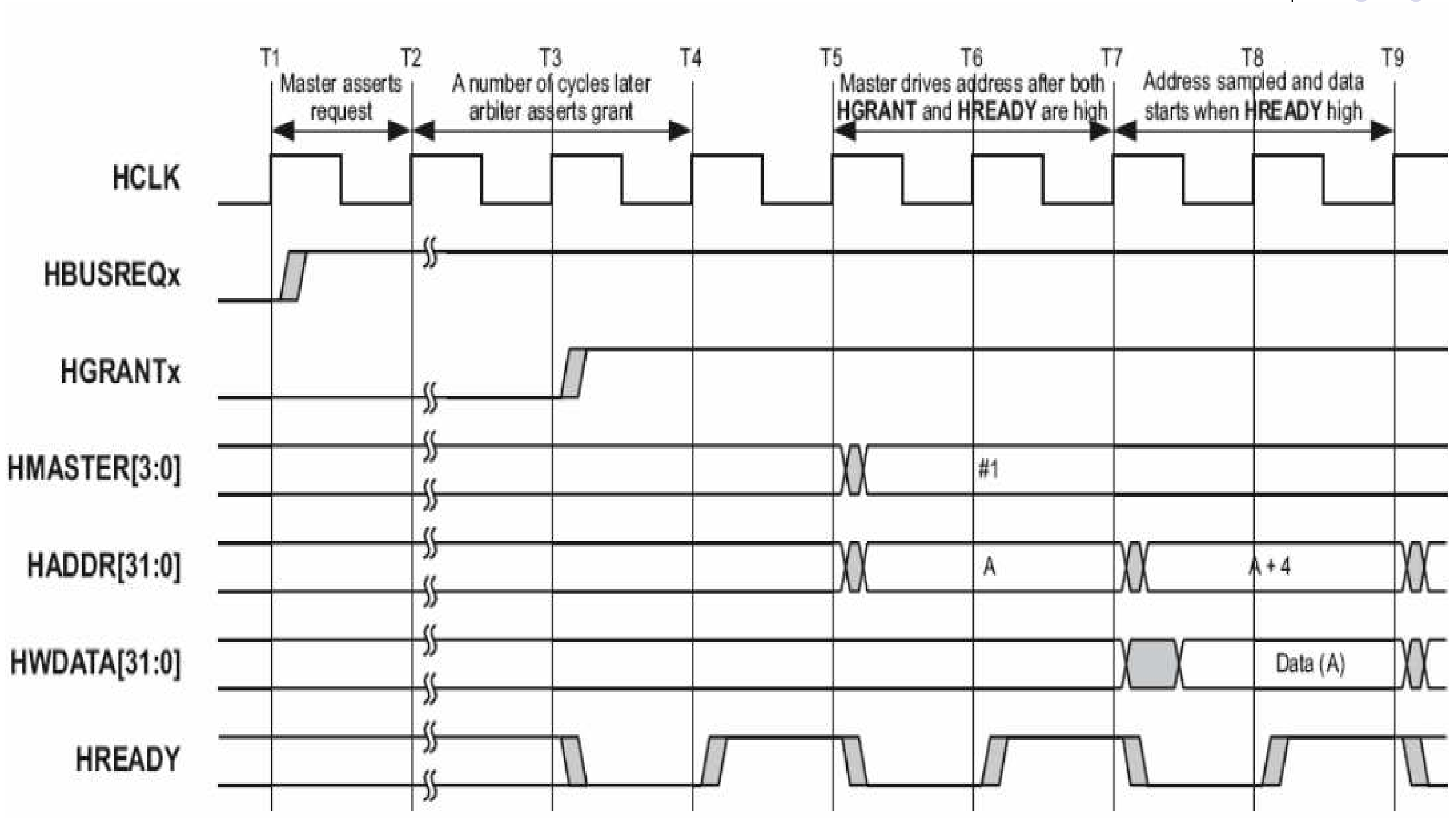
Arbitration

- The *arbitration* mechanism is used to ensure that only one master has access to the bus at any one time
- 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



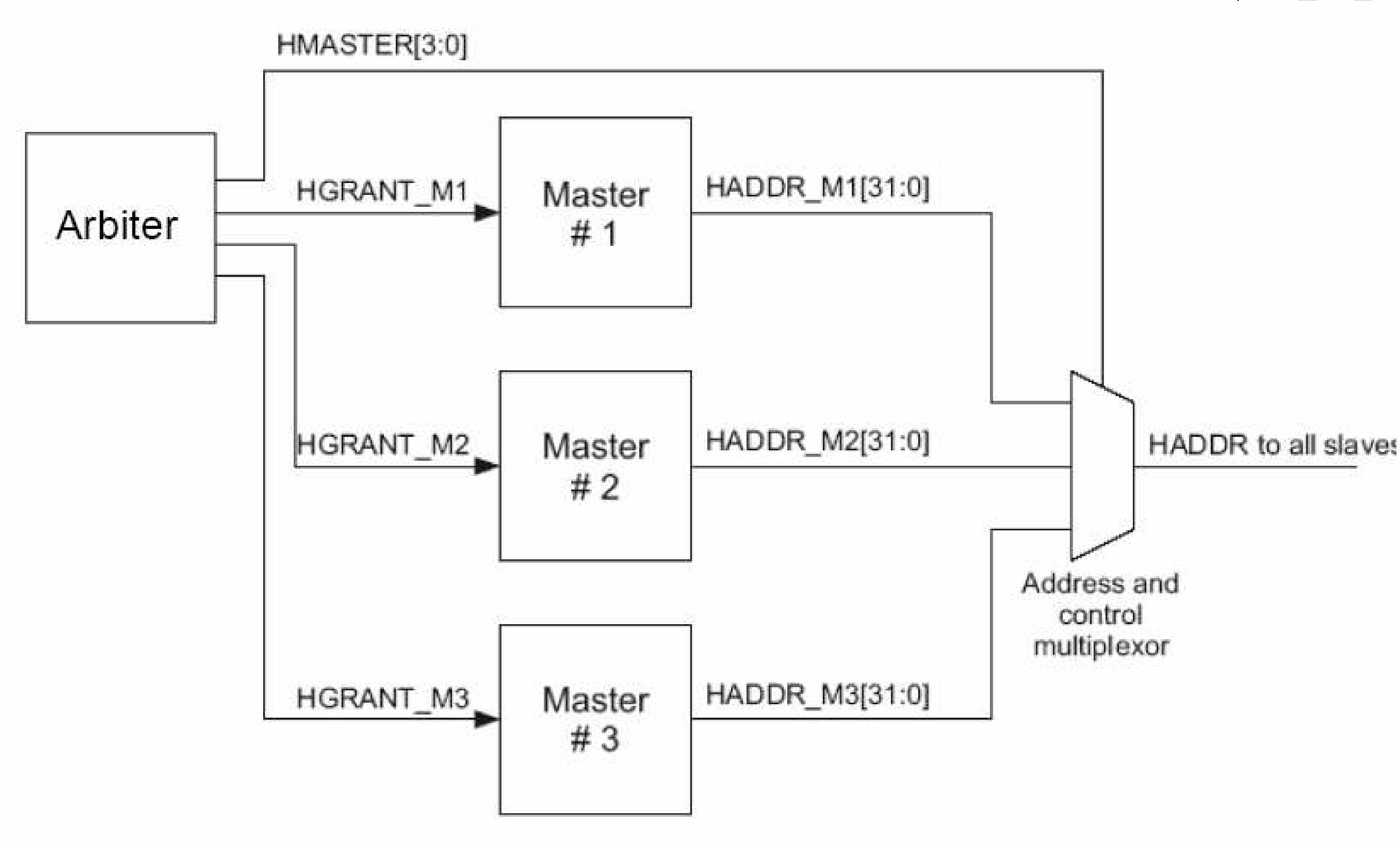
SEMICON
THINKING OUT OF THE BOX

Granting access with wait states


T1
T2
T3
T4
T5
T6
T7
T8
T9
Master asserts request
A number of cycles later arbiter asserts grant
Master drives address after both HGRANT and HREADY are high
Address sampled and data starts when HREADY high
HCLK
HBUSREQx
HGRANTx
HMASTER[3:0]
#1
HADDR[31:0]
A
A + 4
HWDATA[31:0]
Data (A)
HREADY

SEMICON

Bus master grant signals



SEMICON
THINKING OUT OF THE BOX

Split Transfers

- SPLIT transfers improve the overall utilization of the bus
  - separating (or splitting) the operation of the master providing the address to a slave from the operation of the slave responding with the appropriate data

Split Transfers

- When a transfer occurs the slave can decide to issue a SPLIT response if it believes the transfer will take long time
- This signals to the arbiter that the master which is attempting the transfer should not be granted access to the bus until the slave indicates it is ready to complete the transfer
- The arbiter is responsible for observing the response signals and internally masking any requests from masters which have been SPLIT

Split Transfer Sequence

1. The master starts the transfer in an identical way to any other transfer and issues address and control information

2. If the slave is able to provide data immediately it may do so. If the slave decide that it may take a number of cycles to obtain the data it gives a SPLIT transfer response

3. The arbiter grants other masters use of the bus

Split Transfer Sequence

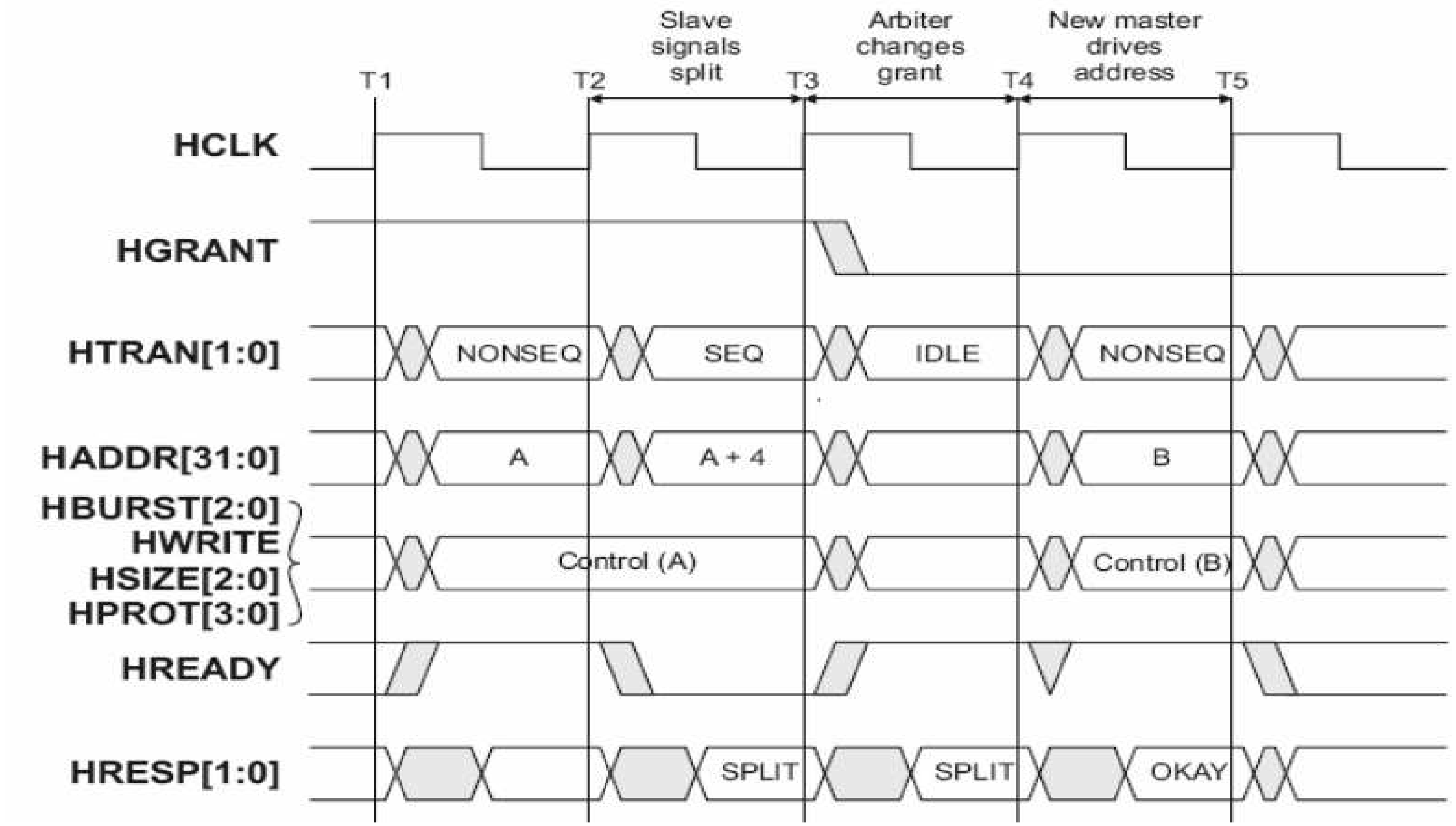
4. When the slave is ready to complete the transfer it asserts the appropriate bit of the HSPLITx bus to the arbiter to indicate which master should be regranted access to the bus

5. The arbiter observes the HSPLITx signals on every cycle, and when any bit of HSPLITx is asserted the arbiter restores the priority of the appropriate master

6. Eventually the arbiter will grant the master so it can re-attempt the transfer. This may not occur immediately if a higher priority master is using the bus

7. When the transfer eventually takes place the slave finishes with an OKAY transfer response

Handover after split transfer



SEMICON

Q & A

SEMICON
THINKING OUT OF THE BOX