hochschule hof
University of Applied Sciences

UNIVERSIDAD DE MÁLAGA

Overview of advanced storage technologies and storage virtualization

Pablo Pérez Trabado

Dept. of Computer Architecture

University of Malaga (Spain)

Disclaimers

- Some of the figures in the slides are taken from SNIA Education tutorials. To comply with the SNIA conditions of use for these documents, any SNIA tutorial used in the elaboration of these slides has been also provided as a handout

- Many (but not all) figures in the slides have been taken from Internet-available resources, including Wikipedia. Whenever possible by copyright restrictions, the original source has been also provided as a handout along with the slides. Also, whenever possible the original source is quoted. In any case, no ownership claims are made over images in the slides not drawn by the author himself.

hochschule hof
University of Applied Sciences

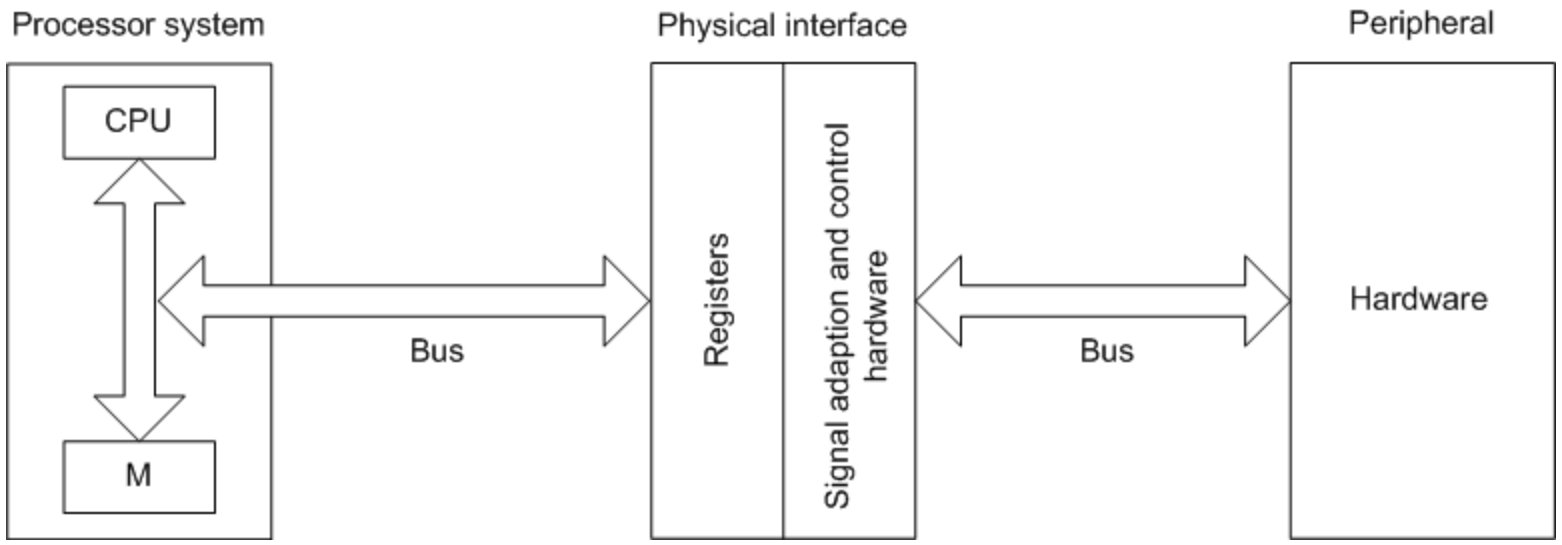
UNIVERSIDAD DE MÁLAGA

Basics of I/O

What will we learn?

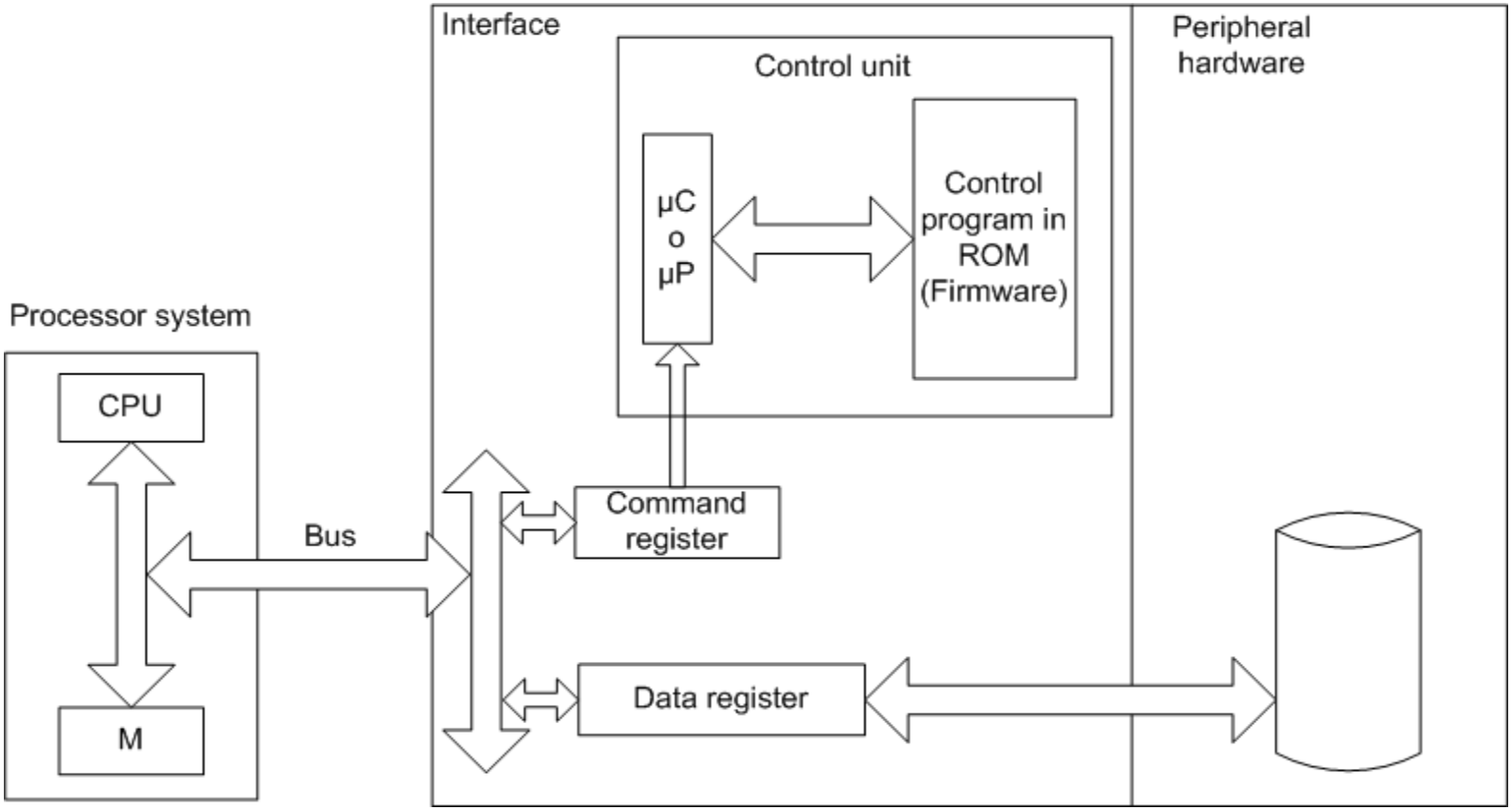
- Difference between bus and interface
- What is an I/O transaction
- Roles of initiator and target
- Synchronous and asynchronous transactions

Smart peripherals, bus and interface



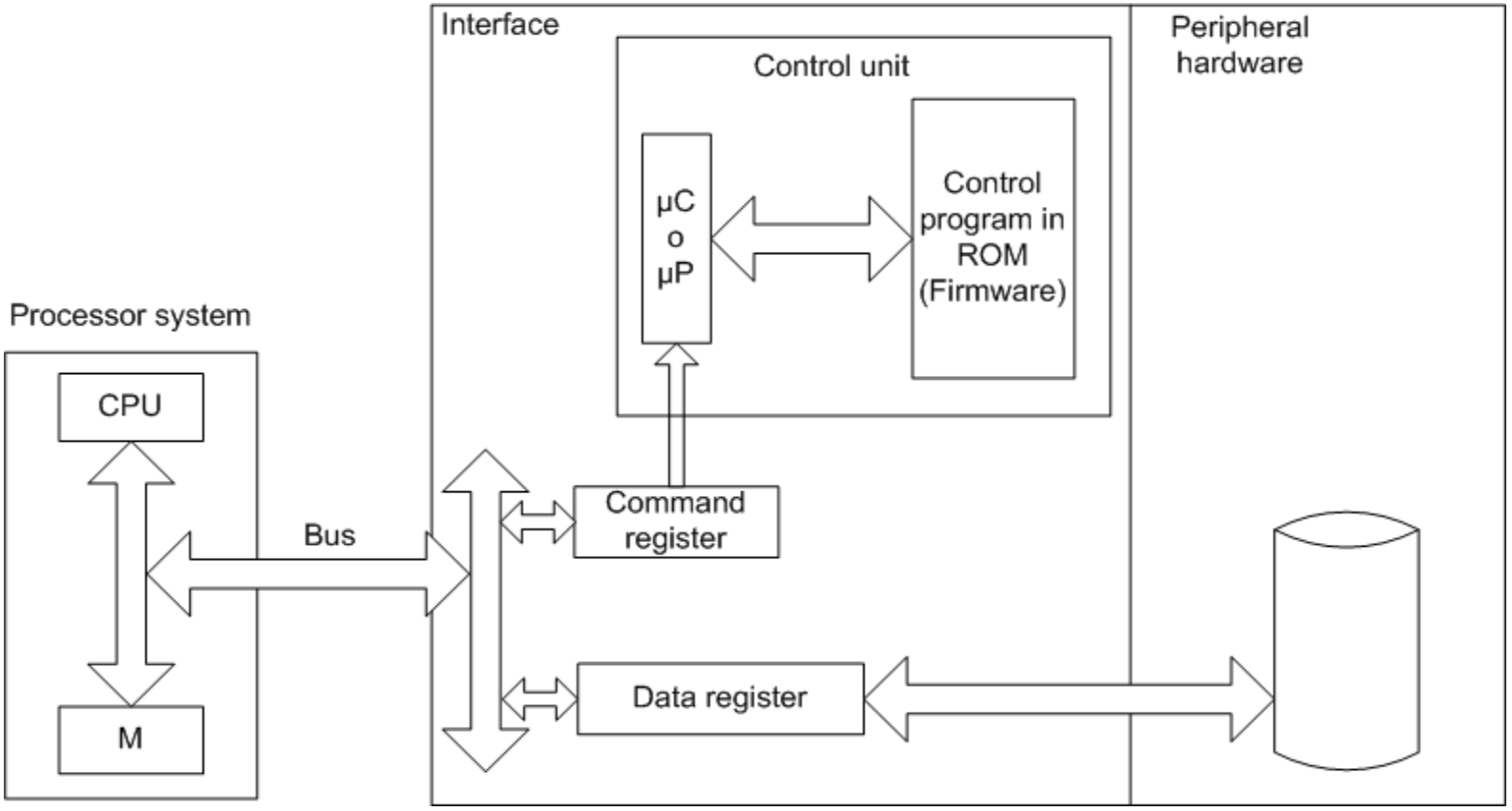
- Figure shows basic structure of any storage I/O
- Physical interface = circuitry intermediating between processor system and storage hardware
  - Usually integrated within peripheral
- Processor system can also be CPU/RAM of adaptor card (i.e, SCSI HBA)

Smart peripherals, bus and interface



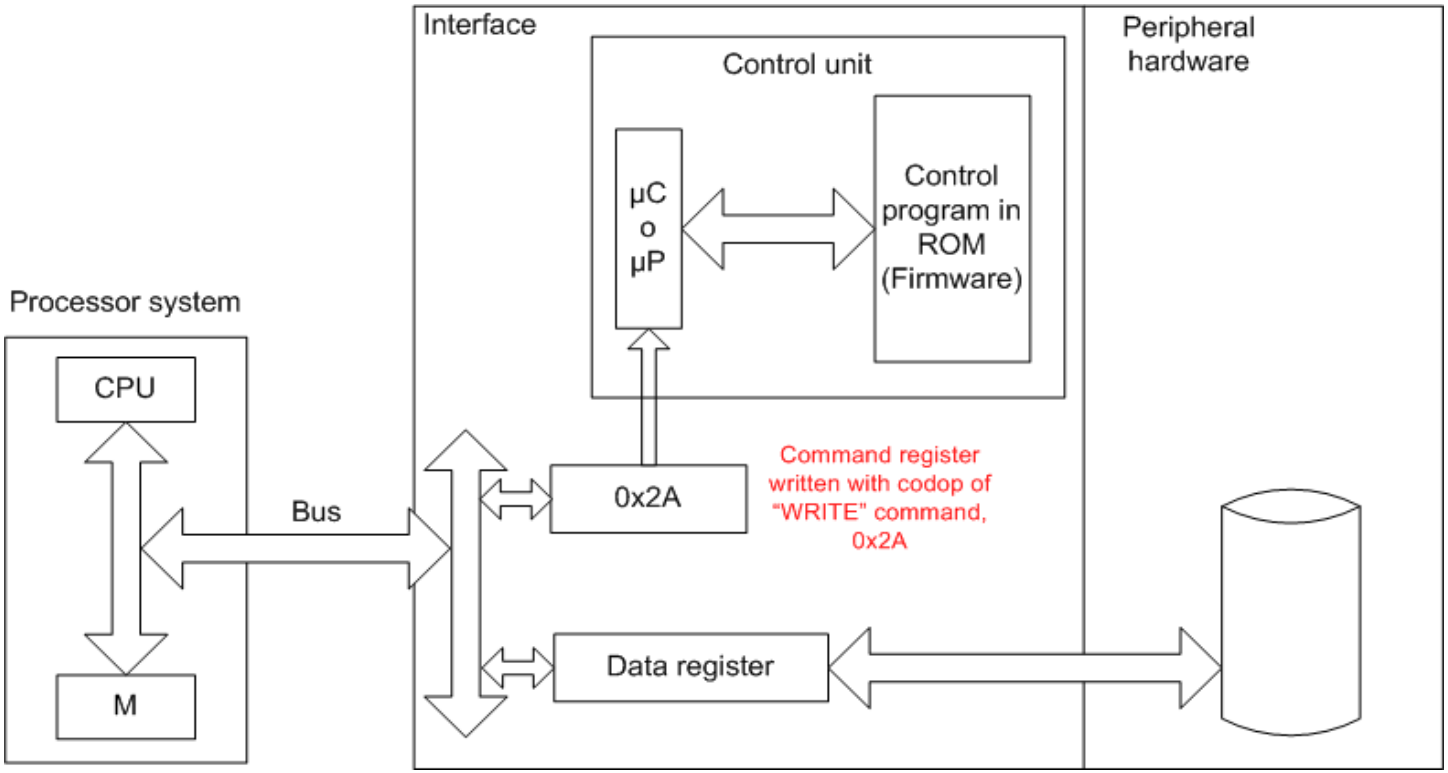
- Peripheral is "smart" when:
  - It is able to execute autonomously commands sent by processor system
  - As a result of this execution, returns to processor system data and/or status info

Smart peripherals, bus and interface



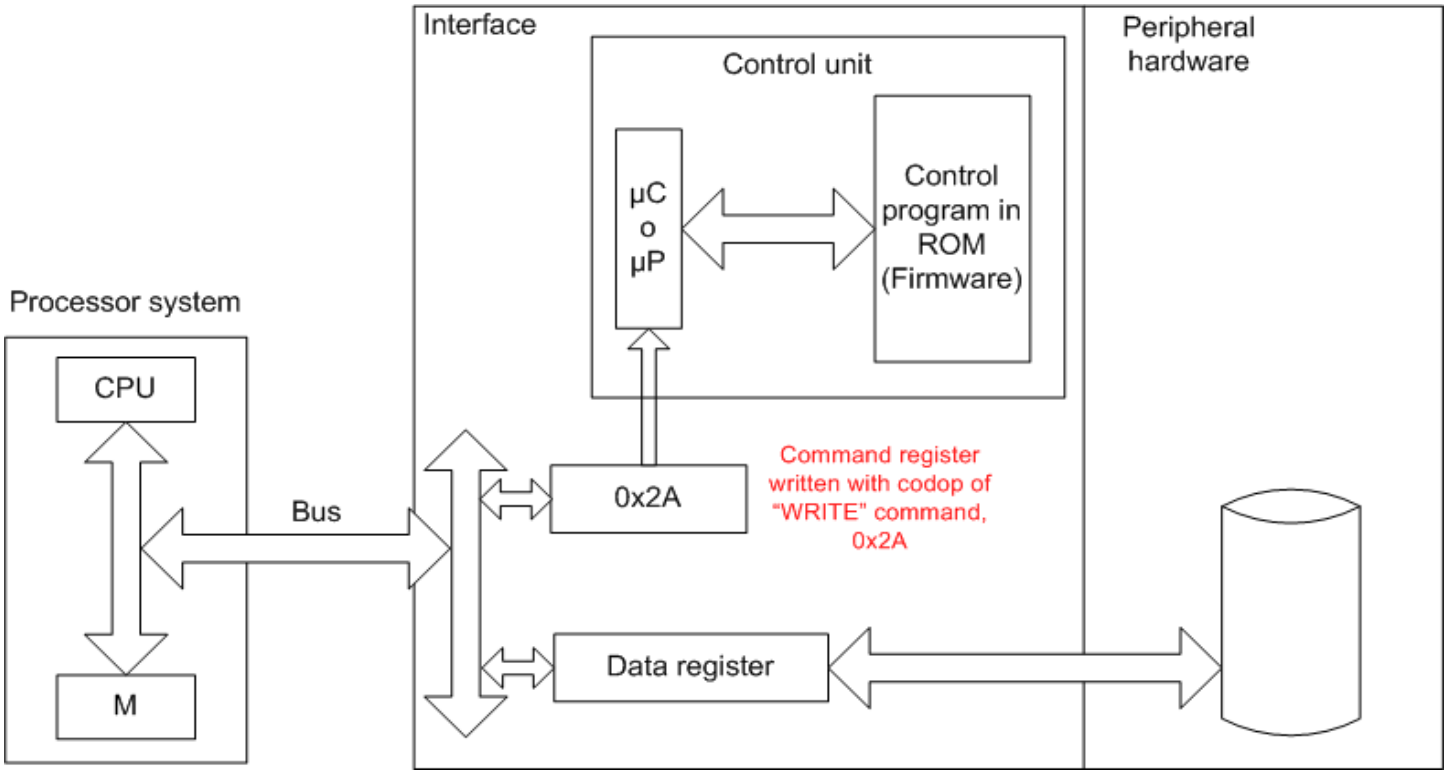
- Physical interface of smart peripheral holds complex control unit, co containing:
  - A microprocessor or advanced microcontroller
  - A ROM with the code ("firmware") that implements each command

Smart peripherals, bus and interface



- Processor system sends to peripheral request to execute command
  - Request is numerical encoding for command (*operation code* = *codop)* and parameters
    - Each command has an unique opcode
  - Opcode (and parameters) written to command register through bus

Smart peripherals, bus and interface



- Control unit interprets opcode and parameters as bitfields, and executes requested actions
- Data and results produced by command execution are read from data/status registers through bus

Bus and interface

- To properly understand operation of a smart peripheral, we must distinguish between *bus* and *interface*

- Bus = subsystem which interconnects two or more devices through a set of common lines, defined by:
  - A specification of physical lines and signaling
  - A functional and timing specification of data exchanges through these lines, and their control mechanisms

- Important: a bus does **not** interpret (decode as codop or parameters) data moved through its lines

Bus and interface

- (Logical) interface = The protocol through which a smart peripheral can be requested the autonomous execution of a command, and the return of data and/or status info
  - An interface requires a bus over which processor system and peripheral exchange data
  - But the interface adds interpretation (as commands, parameters or results) to the data being transmitted over the bus
- Definition of interface includes also its own rules on ordering, control and timing of command a data transfers
- In brief, interface = bus + "intelligence" = bus + autonomous processing of commands

Bus and interface

- By far, SCSI is the most widely used (logical) interface for storage devices
  - It is also the core of this course

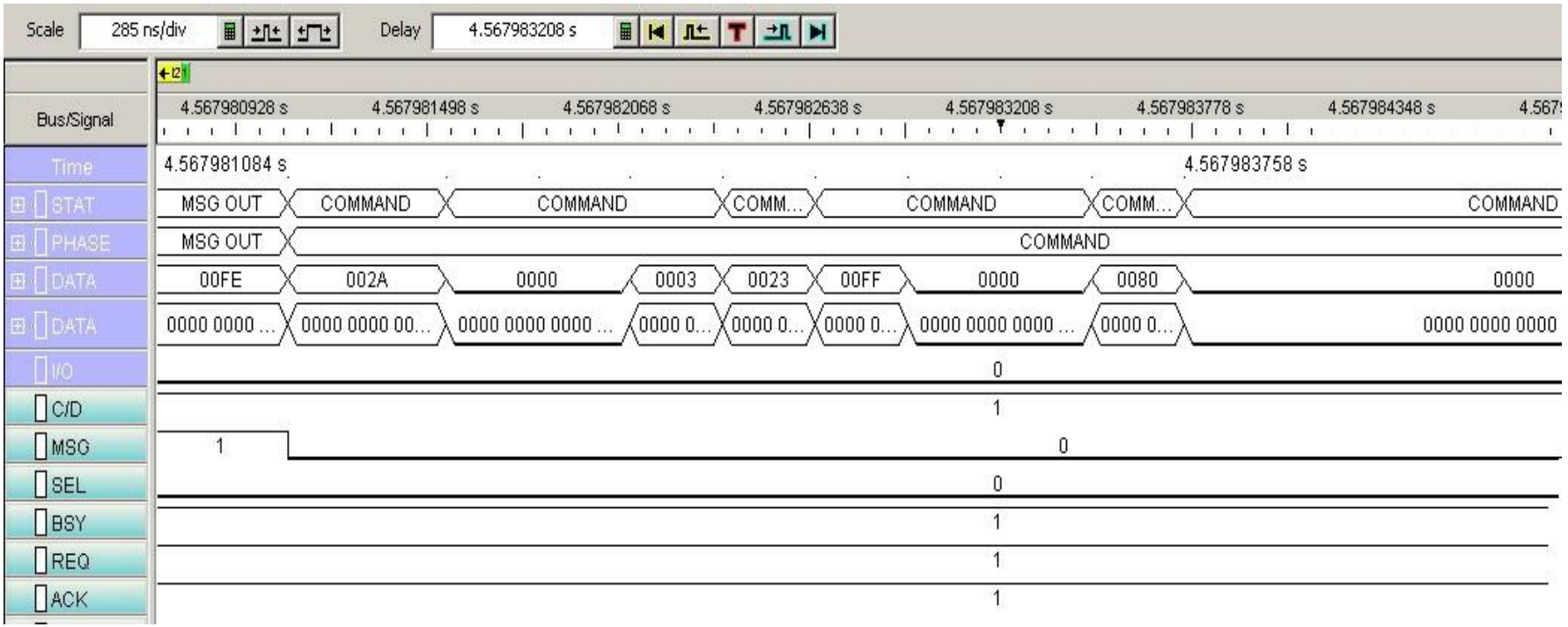
- Unfortunately, there is a lot of ambiguity in the daily use of the words "interface" and "scsi"
  - **"Interface"** is routinely used to name both the *physical* interface (hardware circuitry on the peripheral side) and the *logical* interface (the protocol for command and response exchanges)
    - In this course, **interface** will be synonym of "*logical interface*"
  - **"SCSI"** is frequently used to name both the logical interface "SCSI" and the parallel bus (SPI) over which it can be transported
    - In this course, the word **"SCSI"** alone will always mean the logical interface SCSI

Example: SCSI WRITE over SPI bus

Time	Time	STAT	DATA-8
4554214772 ns	151.61 us	ARBITRATION	80
4554215356 ns	0.584 us	SELECT	88
4554218924 ns	3.568 us	MSG OUT	C0
4554220922 ns	1.998 us	MSG OUT	20
4554223122 ns	2.2 us	MSG OUT	FE
4554228472 ns	5.35 us	COMMAND	2A
4554228896 ns	0.424 us	COMMAND	00
4554229172 ns	0.276 us	COMMAND	00
4554229448 ns	0.276 us	COMMAND	03
4554229722 ns	0.274 us	COMMAND	22
4554229998 ns	0.276 us	COMMAND	FF
4554230272 ns	0.274 us	COMMAND	00
4554230548 ns	0.276 us	COMMAND	00
4554230822 ns	0.274 us	COMMAND	80
4554231096 ns	0.274 us	COMMAND	00
4554236148 ns	5.052 us	DATA OUT	00
4554236248 ns	0.1 us	DATA OUT	00
4554236350 ns	0.102 us	DATA OUT	20
4560918946 ns	6682.596 us	STATUS	00
4560920940 ns	1.994 us	MSG IN	00

- Figure shows listing of contents sent over the SCSI parallel bus (SPI) to request and execute a WRITE command on hard disk
- At the SCSI interface level, controllers on both disk and HBA interpret values on bus as bitfields, from which opcode, parameters and results are extracted
  - Interface standarization allows precise, non-ambiguous interpretation of value and role for each byte

Example: SCSI WRITE over SPI bus



- Figure shows chronogram of same data going through the SCSI parallel bus
  - Bus just transfers data between the devices
  - But completely ignore any meaning associated to this data

Basic I/O concepts: I/O transaction

- **I/O transaction** = sequence of atomic operations required to initiate and fulfill a complete data transfer
- Require always:
  - An *address phase* (Dir), in which the device initiating the transaction identifies its peer device for this transaction, and the type of I/O transaction to perform
  - One or several data transfer phases (D), in which the actual data is sent
- Can be performed at the interface or bus level
  - When at bus level, we speak of ***Bus transaction***
  - So, we'll use *I/O transaction* as shorthand for "interface-level I/O transaction"

Bus master and bus slave

- A bus transaction involves always a single bus master and at least one bus slave

- Bus master: device with capability to start a bus transaction
  - Only a bus master can generate a bus address phase
  - Typical bus masters are CPU and DMA-capable controllers

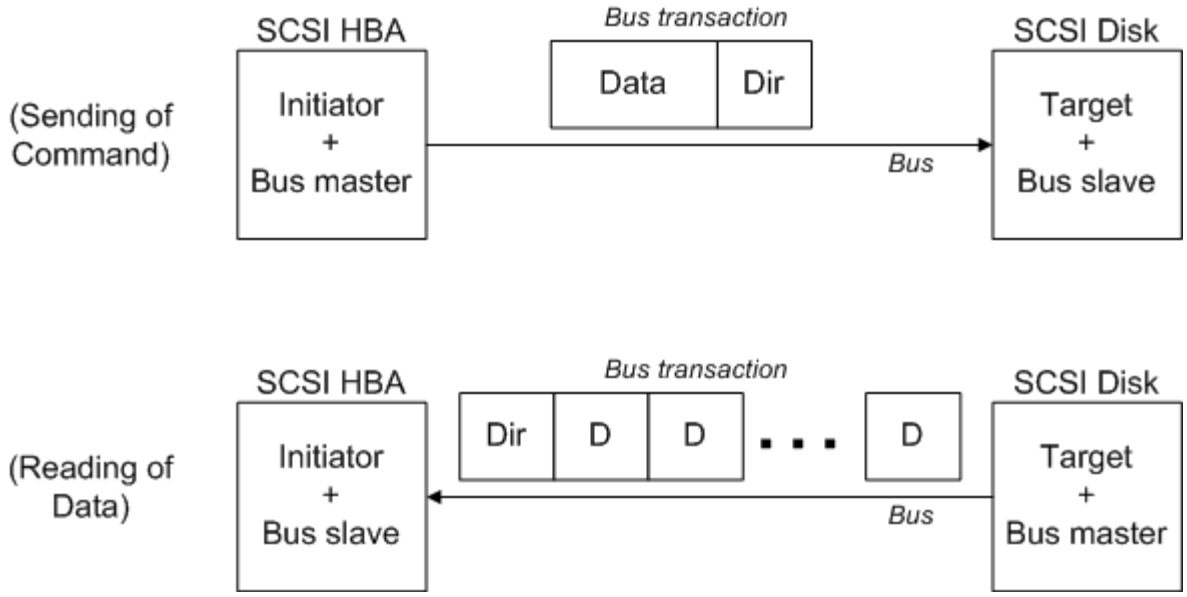
- Bus slave: Device which can participate in a bus transaction, but lacks the capability of initiating it
  - Must wait passively to be addressed by a bus master in the address phase of a bus transaction

Initiator and target

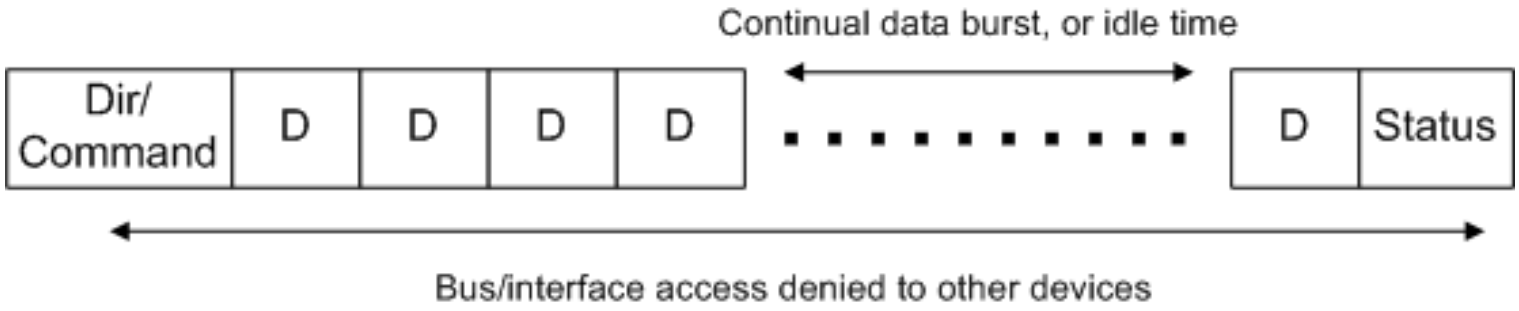
- Roles equivalent to bus master and bus slave, but at interface level
- I/O transaction involves always a single initiator and at least one target
- Initiator: device with capability to initiate an I/O transaction
  - Only an initiator can issue an interface command
  - Commands are sent as payload in data phases of a bus transaction
  - Example of initiator: SCSI controller card (*HBA = Host Bus Adapter*)
- Target: Device which can participate in an I/O transaction, but lacks the capability of initiating it
  - Must wait passively to be addressed and sent a command by an initiator
  - Once received, target will perform and complete command autonomously

Initiator and target

- A single I/O transaction may require one or several bus transactions to complete
  - Roles of bus master and bus slave can swap throughout transaction
- Example: SCSI READ from disk:



Blocking ("synchronous") transactions

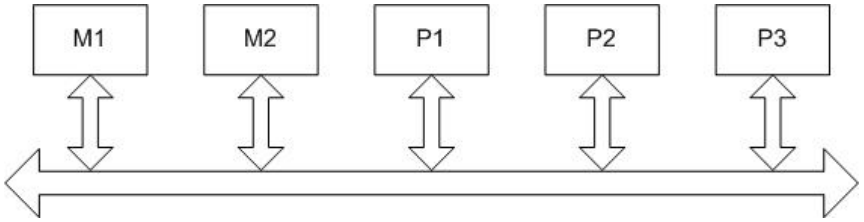


- I/O or bus transaction is blocking (*synchronous*) if, once started, it must be finished before a new transaction can be issued
  - Simpler to implement
  - Efficiency problems when very long transfers (bus hogging) or when long idle times (bus stall)

Split ("asynchronous") transactions

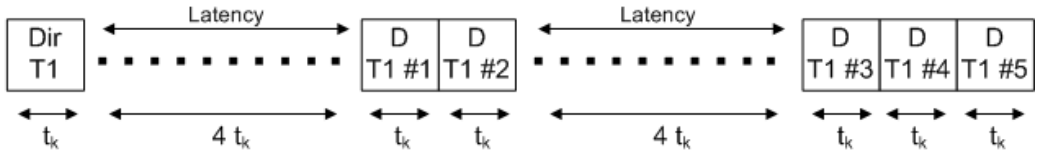
- Asynchronous I/O transactions:
  - Can be interrupted ("*split*") before their completion. Split possible:
    - Between address phase and first data phase
    - Between two arbitrary data phases
  - Once interrupted, can be resumed from interruption point
    - Requires additional address phase to indentify current transaction
  - Allow interleaved execution of several I/O transactions
- Initiator and target must be able to store status info on interrupted transaction
- Coupled with a interface-level command queue, improves performance of I/O
  - Avoids wastage of I/O capacity by not idling bus while waiting for peripheral
  - May allow reordering of commands in queue

Example of split transactions

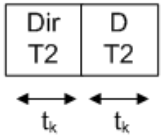


- Let's suppose a multi-drop bus with two initiators (M1, M2) and three targets (P1, P2, P3), in which these three I/O transactions must be performed:

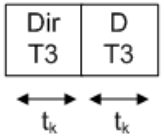
T1: M1 to P1, 5 data phases



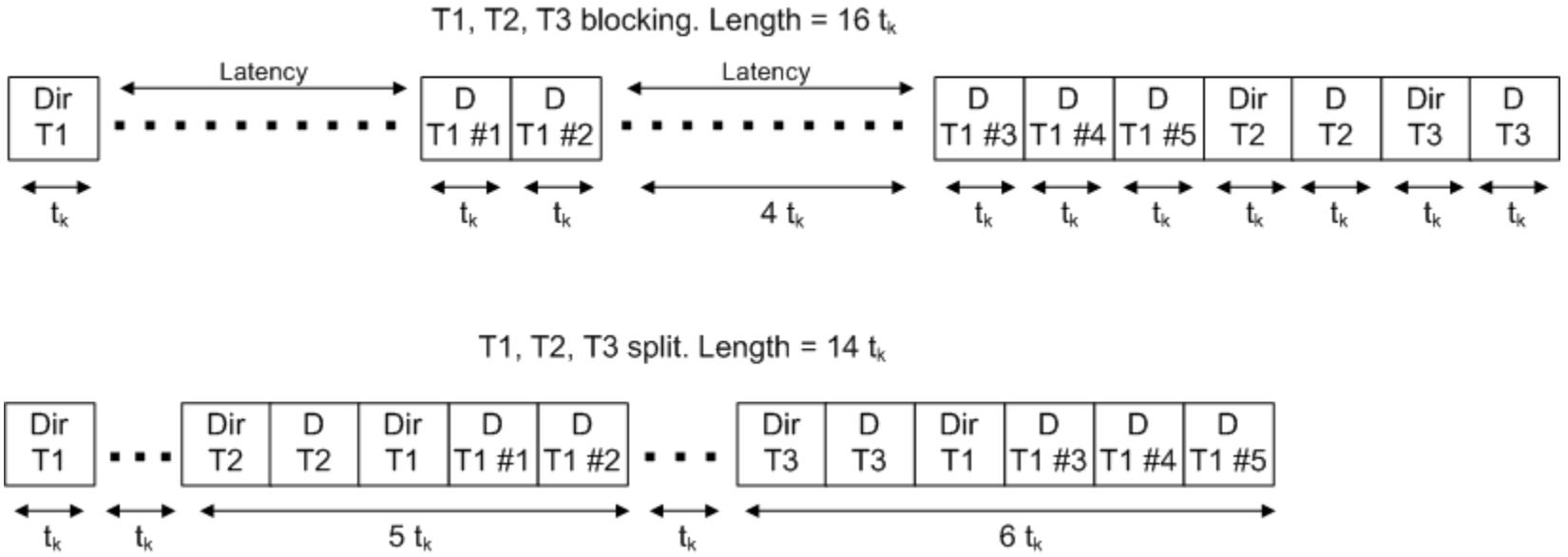
T2: M2 a P2, 1 data phase



T3: M2 a P3, 1 data phase



Example of split transactions



- Figure shows outcome of execution of those commands:
  - using blocking (synchronous) transactions (above)
  - With split (asynchronous) transactions (below)
- Asynchronous execution has reduced total time, plus individual latencies for commands coming from initiator M2
  - Improved responsiveness of system M2

What's next?

- Study of the most important of the current day I/O interfaces: SCSI