



Communications Protocols

黃元豪

Yuan-Hao Huang

國立清華大學電機工程學系

Department of Electrical Engineering

National Tsing-Hua University

Steps for System Design



- Specification
- Partitioning
- Interface Specification
- Timing Design
 - Timing and sequencing of operations
- Module Design



Specification

- A good specification should include
 - Overall description
 - What the system is, what it does, how it is used.
 - Inputs and outputs
 - Formats, range of values, timing, and protocols
 - States
 - User visible states, including registers, mode bits, and internal memories
 - Modes
 - Options
 - All notable features of the system
 - All interesting edge cases

Interface Timing

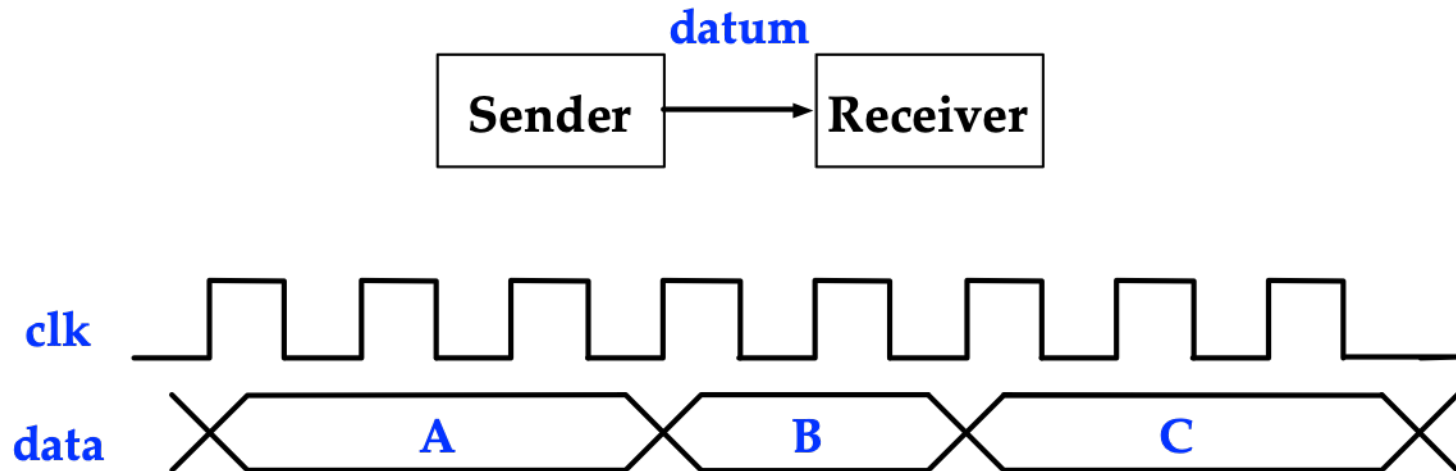


- Data transfer sequencing
 - When the data is valid from the source, and when the destination is ready to receive the data.
- Data transfer Strategies
 - Always valid timing
 - Periodically valid signals
 - Flow control

Data transfer Strategies – Always Valid Timing



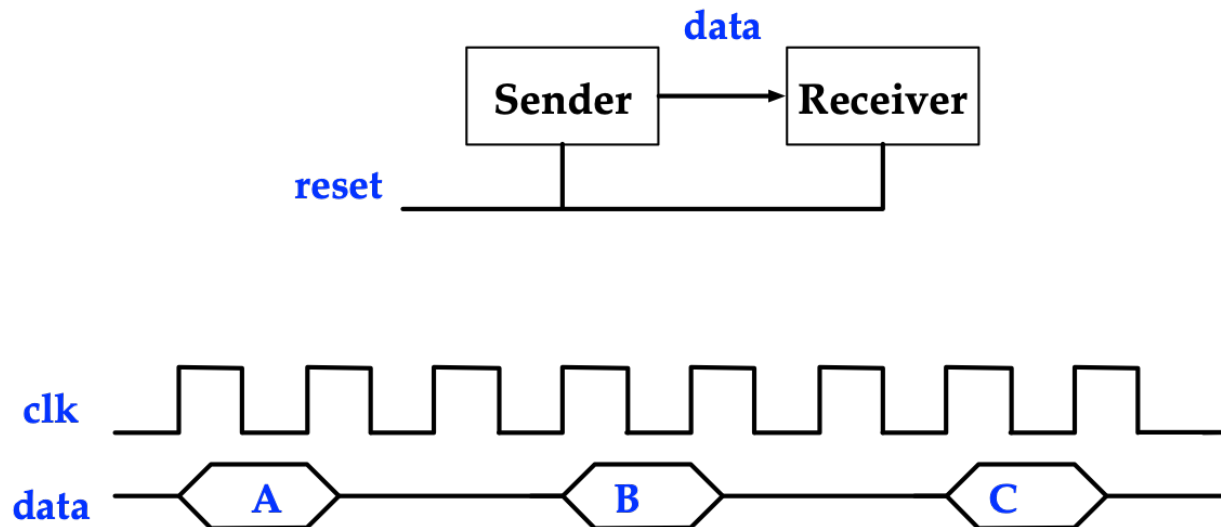
- An always valid signal represents a value that can be dropped or duplicated.
- A static or constant signal is a special case.



Data transfer Strategies – Periodically Valid Signals



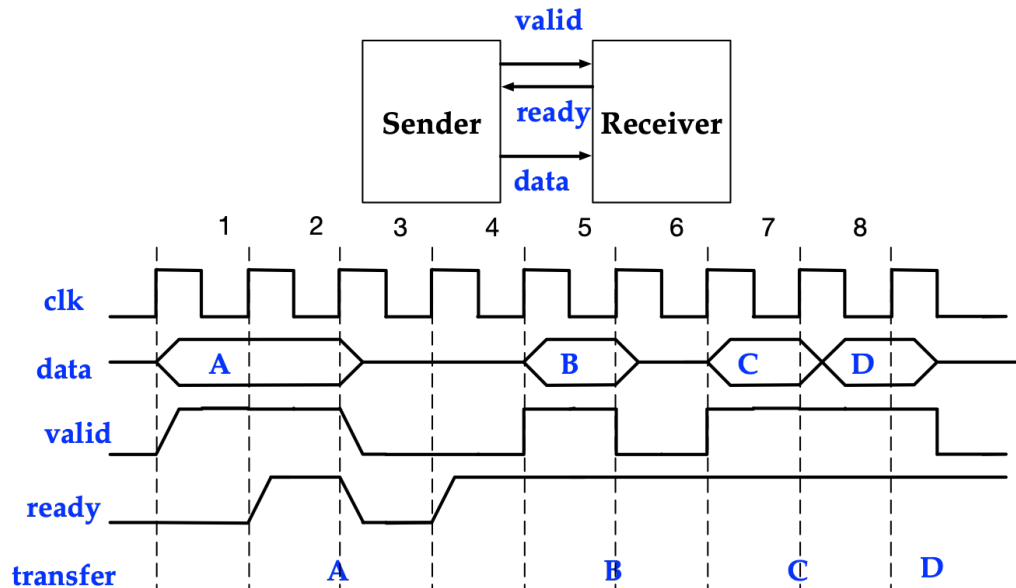
- A signal with *periodically valid timing* or *periodical timing* is valid once every N cycles.
 - Each value of a periodically valid signal represents a particular event, task, or token, and cannot be dropped or duplicated.
 - With periodically signaling, the sending and receiving modules must be synchronized.



Data transfer Strategies – Flow Control



- Use explicit sequencing signals: *valid* and *ready*.
 - Data is only transferred when both *valid* and *ready* are asserted.
 - pull timing: transfers are controlled by the receiver *ready*
 - push timing: transfers are controlled by the sender *valid*
 - *valid* signal can be encoded in the data signal by using an unused or invalid data code to imply not valid.
 - FIFO: output is *valid* unless it is empty, input is *ready* unless it is full



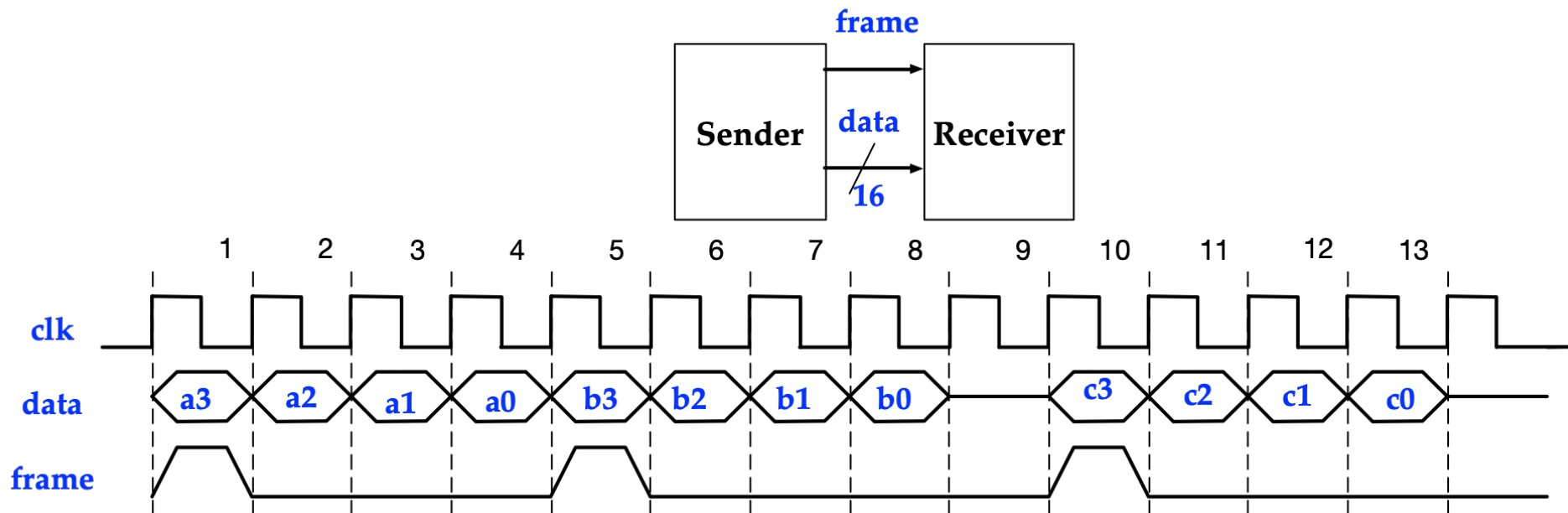


Interface Partitioning and Selection

- A common interface technique provides separate fields for *control*, *address*, and *data*.
- The *control* and *address* fields are selection fields.
 - The control field selects the operations to be performed
 - The address field selects which location the operation is performed on.
- Both the data and selection fields are sequenced using the above-mentioned timing conventions.

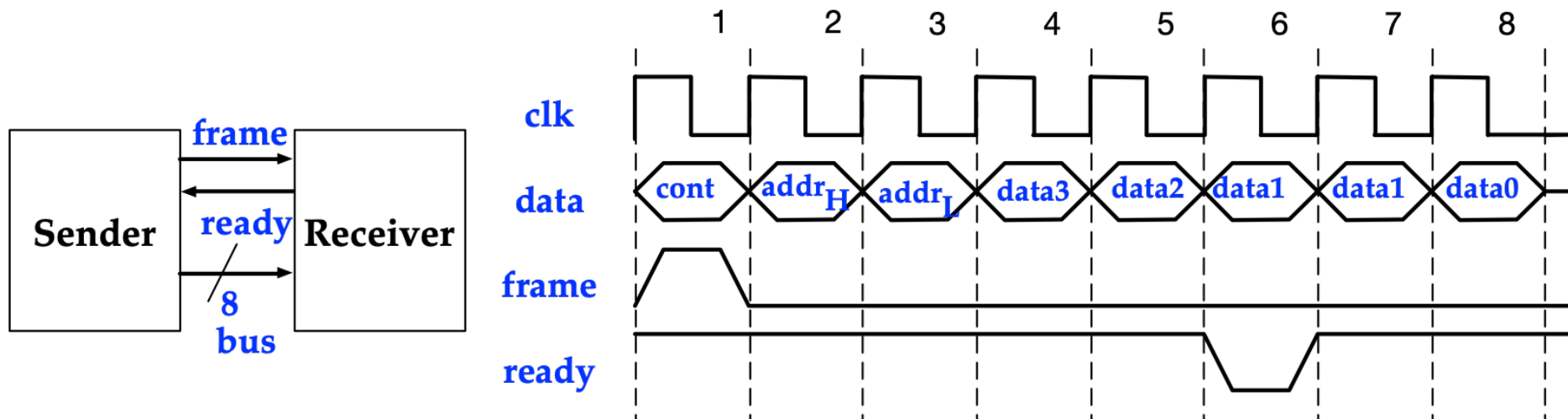
Serial and Packetized Interfaces

- The interface transfers 64-bit block of data once every four cycles.
 - Push timing: One-way flow control using *frame* (valid) signal
 - nested timing with push timing used at the frame level and periodically timing used at the cycle level.



Serial and Packetized Interfaces

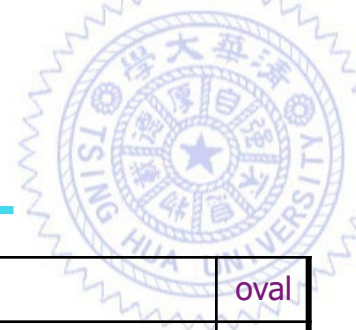
- Memory and I/O interfaces often serialize the command, address, and data fields to transmit over a shared, narrow bus.
 - Cycle-valid/frame-ready flow control
- Serialized interfaces can be thought of as being *packetized*.
 - Each item transmitted is a *packet* of information containing many fields and possibly of variable length



Serial and Packetized Interfaces



- Serialized vs. parallel interface
 - Considerations of cost and performance
 - Pros of serialized interface: reduction of # of pins or wires
 - Cons of serialized interface: increase in latency and complexity of the serialization/de-serialization, and framing
- On-chips
 - cost of additional wires is small => parallel
- Off-chips
 - chip pins and system-level signals are expensive => serialized



Timing Tables

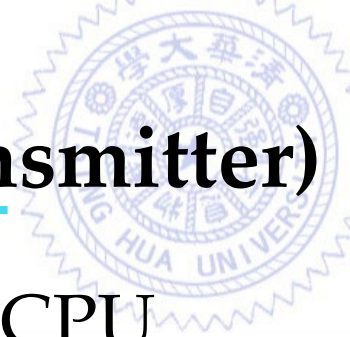
- Timing diagrams illustrate timing relationships for visualizing binary signals for a few cycles.
- For multi-bit signals with many more cycles => timing tables

cycle	rst	ival	in	count	out	oval
0	1	x	x	x	x	x
1	0	1	A ₀	0	00000000	0
2	0	1	A ₁	1	0000000A ₀	0
3	0	1	A ₂	2	000000A ₁ A ₀	0
...	0	1	A			0
8	0	1	A ₇	7	0A ₆ A ₅ A ₄ A ₃ A ₂ A ₁ A ₀	0
9	0	1	B ₀	0	A ₇ A ₆ A ₅ A ₄ A ₃ A ₂ A ₁ A ₀	1
10	0	0	x	1	A ₇ A ₆ A ₅ A ₄ A ₃ A ₂ A ₁ B ₀	0
11	0	1	B ₁	1	A ₇ A ₆ A ₅ A ₄ A ₃ A ₂ A ₁ B ₀	0
12	0	1	B ₂	2	A ₇ A ₆ A ₅ A ₄ A ₃ A ₂ B ₁ B ₀	0
...	0	1	B			0
18	0	0	x	0	B ₇ B ₆ B ₅ B ₄ B ₃ B ₂ B ₁ B ₀	1

De-serializer:

1. Convert a 1-bit wide data input to 8-bit wide data output
2. Both input and output use push flow control
3. The output is not valid until 8 valid inputs have been received.

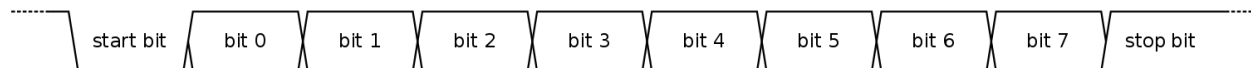
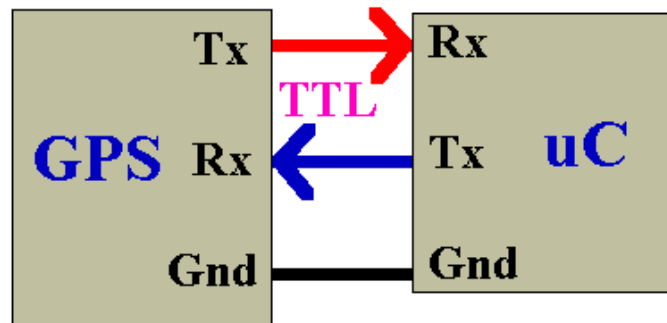
UART



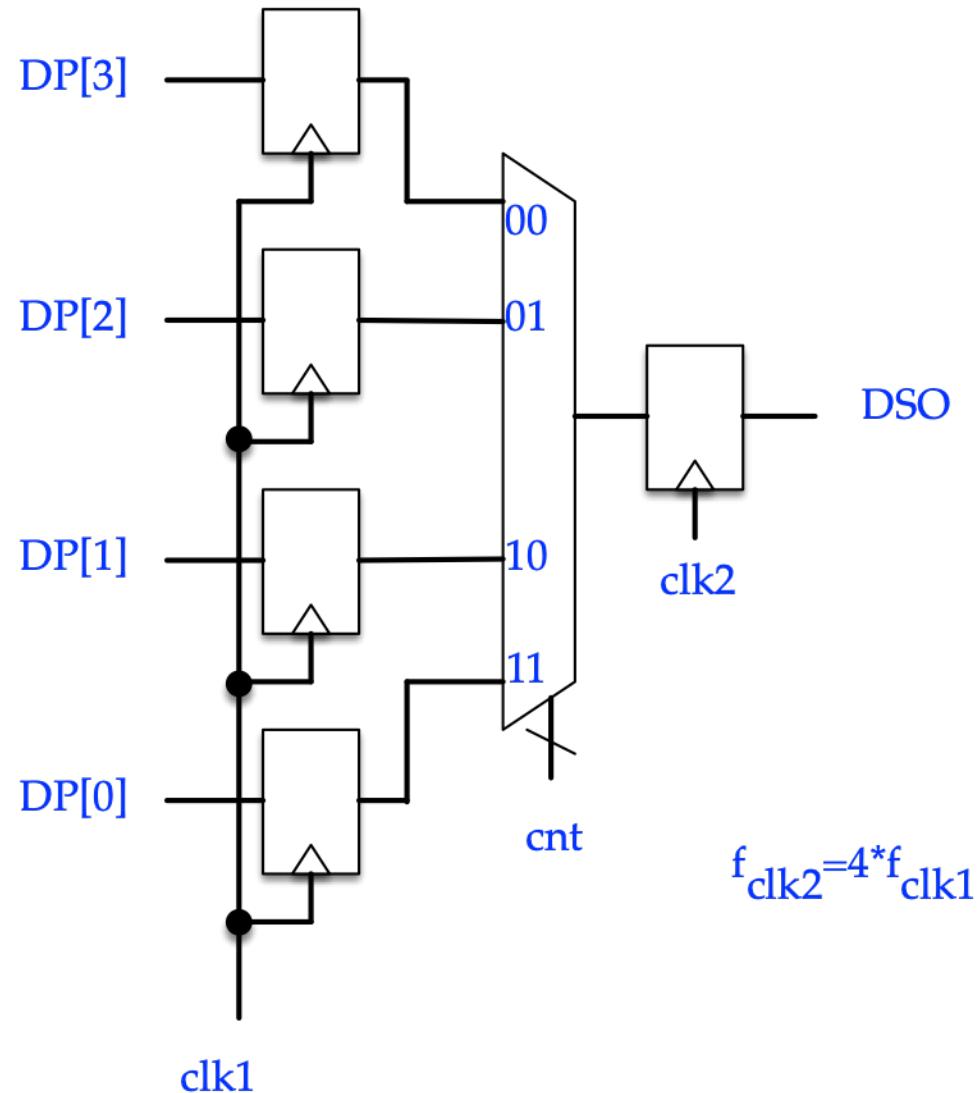
(Universal Asynchronous Receiver/Transmitter)

- serial-to-parallel conversion: peripheral to CPU
- parallel-to-serial conversion: CPU to peripheral
- RS232, RS485
- Baud rate

UART Communication



Parallel-to-Serial Conversion



Serial-to-Parallel Conversion



$$f_{\text{clk2}} = 4 * f_{\text{clk1}}$$

