AXI UART and AXI4-Stream Peripherals

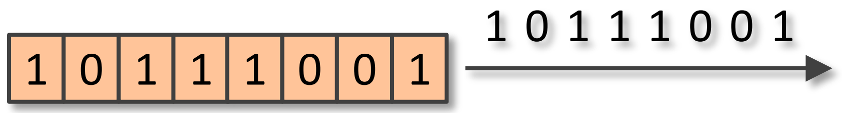
Learning Outcomes

At the end of this module, you will be able to:

- Compare and contrast the properties and operation of serial vs parallel, and synchronous vs asynchronous communication.
- Describe the purpose and operation of a UART protocol.
- Explain how AXI UART is implemented, including timing, control, and receiver and transmitter FIFOs.
- Outline the usage and purpose of the AXI4-Stream protocol and data streams.
- Explain the mechanism for AXI4-Stream, including the timing and transfer handshake mechanism.

Serial Communication

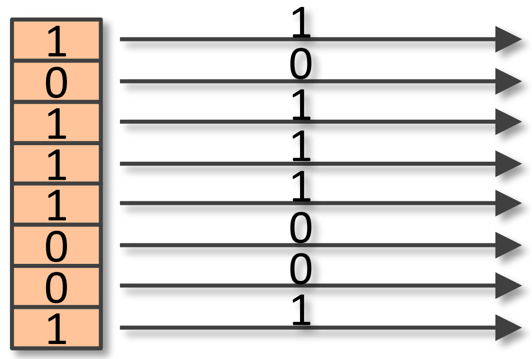
- Serial communication
  - Transmits data one bit at a time, in a sequential fashion.
  - In contrast to parallel communication, in which multiple bits are sent as a whole.
  - Commonly used for long-haul communication, modems, and non-networked communication between devices.
  - Example include UART, SPI, I2C, USB, Ethernet, PCI Express.



Serial communication

Parallel Communication

- Parallel communication
  - Transmits data bits simultaneously, in a parallel fashion.
  - In contrast to parallel communication, in which multiple bits are sent one bit at a time.
  - Fast, but more complex. Synchronization of signals essential.
  - Example include on-chip communication, connections between a processor core and the memory.



Parallel communication

Serial v Parallel Communication

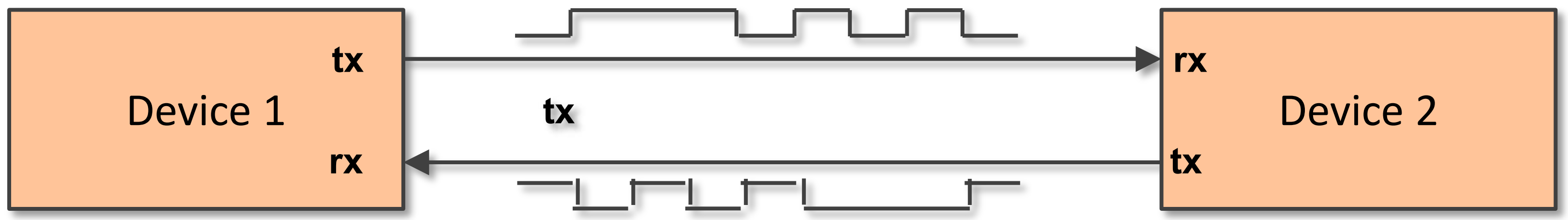
- Cost and weight
  - Less cost and weight for serial, as fewer wires and smaller connectors are needed compared to parallel communication
- Better reliability
  - Parallel communication may introduce more clock skews as well as crosstalk between different wires.
- Higher clock rate
  - Due to the higher reliability, serial communication can be clocked in a much higher frequency, hence increasing transmission speed.
- On the other hand, the conversion between serial and parallel data may produce some extra overhead.

Types of Serial Communication

- Synchronous serial transmission
  - A common clock is shared by both the sender and the receiver.
  - More efficient transmission, since one wire is dedicated to data transfer.
  - More costly, since an extra clock wire is required.
- Asynchronous serial transmission
  - The sender does not have to send a clock signal.
  - Both the sender and receiver agree on timing parameters in advance.
  - Special bits are added to synchronize transmission.

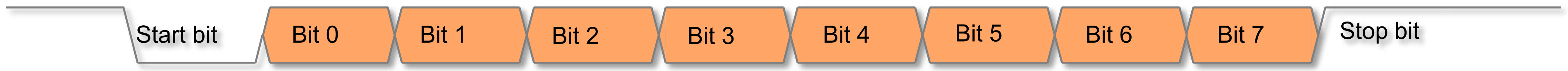
UART Overview

- Universal Asynchronous Receiver/Transmitter (UART)
  - Asynchronous communication, no clock wire required, pre-agreed baud rate
  - Separate transmission and receiving wires
- UART communication
  - Converts data from parallel to serial
  - Sequential data is transferred through serial cable.
  - Receives the sequential data and reassembles it back to parallel.

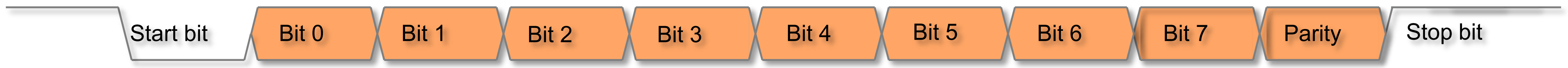


UART Protocol

- Data transfer starts with a starting bit, by driving logic to low for one clock cycle.
- In the next 8 clock cycles, 8 bits are sent sequentially from the transmitter.
- Optionally, one parity bit can be added to improve transfer reliability.
- In the end, the data wire is pull up to high to indicate finishing of the transfer.



Transfer one byte without parity bit



Transfer one byte with parity bit

Character-Encoding Scheme

- What data is sent to the UART in order to display a text?
- ASCII (Characters are coded in American Standard Code for Information Interchange)
  - Encodes 128 characters
  - Ninety-five printable characters, such as “a,” “b,” “1,” “2”
  - Thirty-three non-printing control characters, e.g., next line, back space, escape
  - Can be represented by seven bits, commonly stored as one byte for storage convenience
- UTF-8 (UCS Transformation Format—8-bit)
  - Derived from ASCII in 2007
  - Variable-width encoding scheme that avoids the complication of endianness and byte order marks
  - Widely used for the web pages
  - Compatible with the original ASCII

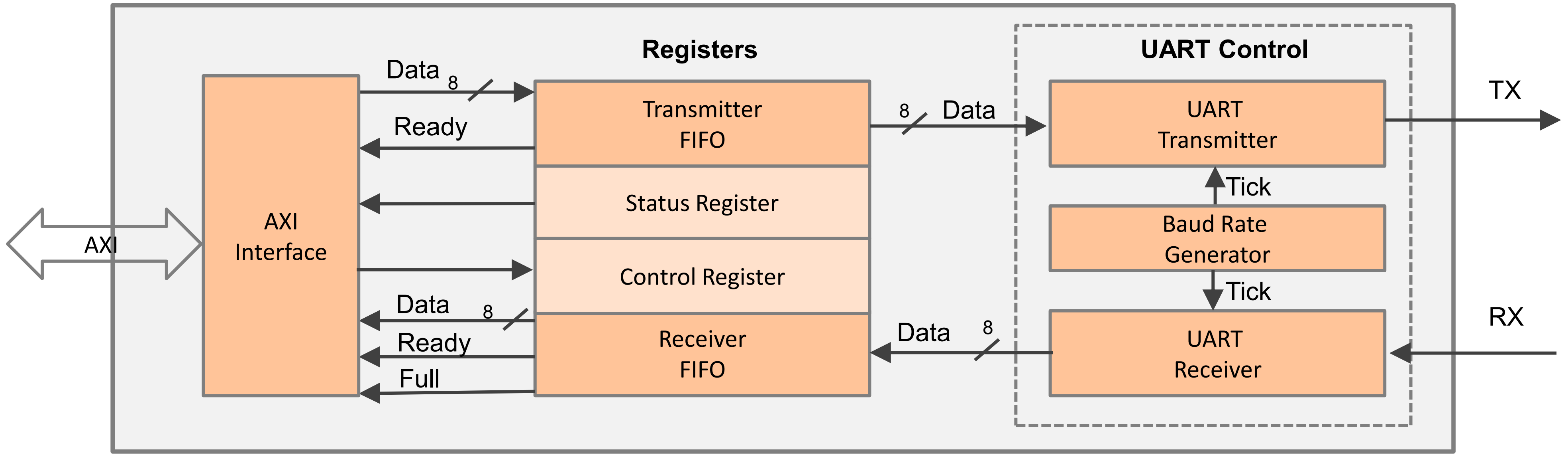
ASCII Encoded Characters

- Example: Some frequently used characters coded in ASCII

Hex	Character		Hex	Character		Hex	Character
0x30	0		0x41	A		0x61	a
0x31	1		0x42	B		0x62	b
0x32	2		0x43	C		0x63	c
0x33	3		0x44	D		0x64	d
0x34	4		0x45	E		0x65	e
0x35	5		0x46	F		0x66	f
0x36	6		0x47	G		0x67	g
0x37	7		0x48	H		0x68	h
0x38	8		0x49	I		0x69	i
0x39	9		0x4A	J		0x6A	J
…			…			…	

AXI UART Implementation

- General block diagram
  - AXI interface: Subordinate AXI interface for register access and data transfer
  - UART registers: Memory mapped registers, including a pair of transmit/receive FIFOs, a control register, and a status register
  - UART control: Includes the receiver, the transmitter, and the baud rate generator



UART Control

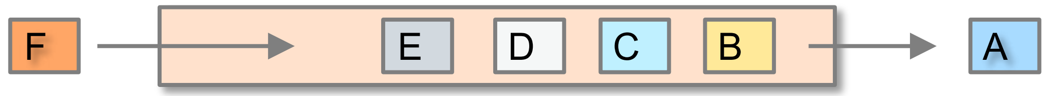
- Contains transmission/receiving controllers and a baud rate generator
- UART transmitter
  - Reads data (in byte) from the transmitter FIFO
  - Converts a single byte data to sequential bits
  - Sends bits to the Tx pin, clocked in a fixed rate provided from the baud generator
- UART receiver
  - Receives the sequential bits from the Rx pin using the clock generated from the baud generator
  - Reassembles the bits into a single byte
  - Writes the received byte to the receiver FIFO
- Baud rate generator (BRG)
  - Generates system ticks for a fixed transmission baud rate, e.g., 19200 bps byte to the receiver FIFO
    - $Baud\ \text{rate} = \frac{on\ chip\ clock\ freq}{sampling\ rate \times divisor}$

UART Register Block

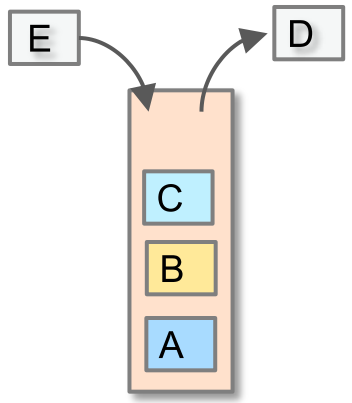
- Contains a control register, a status register, and transmit/receive FIFOs.
- Control register
  - Reset bits for receive/transmit FIFOs
  - Enables bits
  - Mode bits: baud rate, bit length, parity
- Status register
  - Status of the transmit/receive FIFOs
  - Error bits: parity error, frame error, overrun error
- Transmitter/receiver FIFOs
  - Temporarily stores data written from the AXI interface and the UART receiver

First In First Out (FIFO)

- First In First Out (FIFO) refers to a data buffer that outputs its earliest input data, like a data queue.
- In contrast, Last In First Out (LIFO) is a buffer that outputs its latest input data, e.g., program stack.
- Synchronous FIFO
  - Same clock is used for both reading and writing
- Asynchronous FIFO
  - Different clocks are used for reading and writing



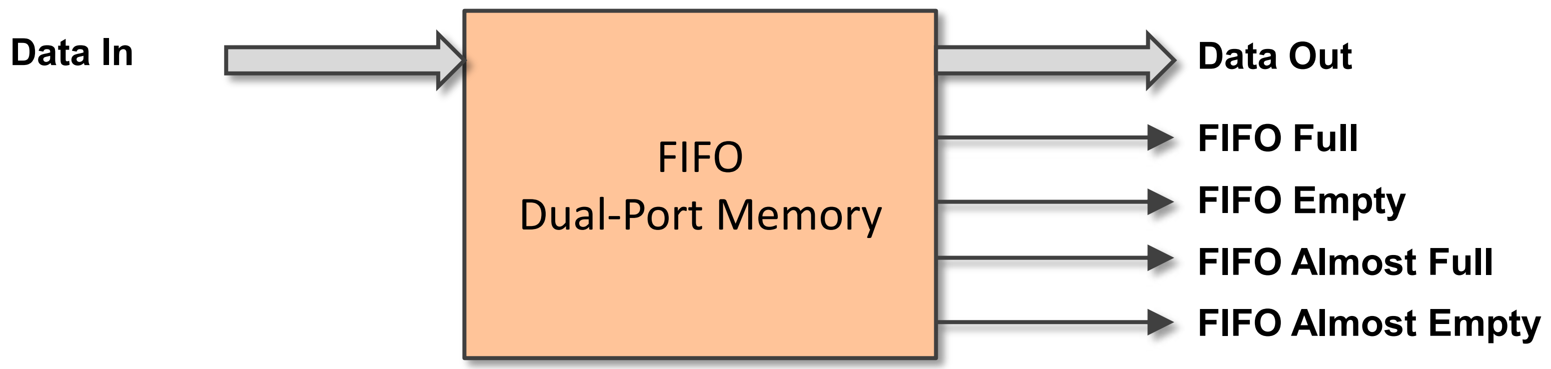
Queue: First In First Out



Stack : Last In First Out

First In First Out (FIFO)

- FIFO implementation:
  - Shift registers: as many as storable word-bits
  - Dual-port memory: one port for reading and the other one for writing
- Additional flags are used to indicate the status of the FIFO
  - FIFO full: all memory space is used; hence, no more data can be written
  - FIFO empty: no data in the memory; thus, no more data can be read
  - Some FIFOs also provide half full/empty signal



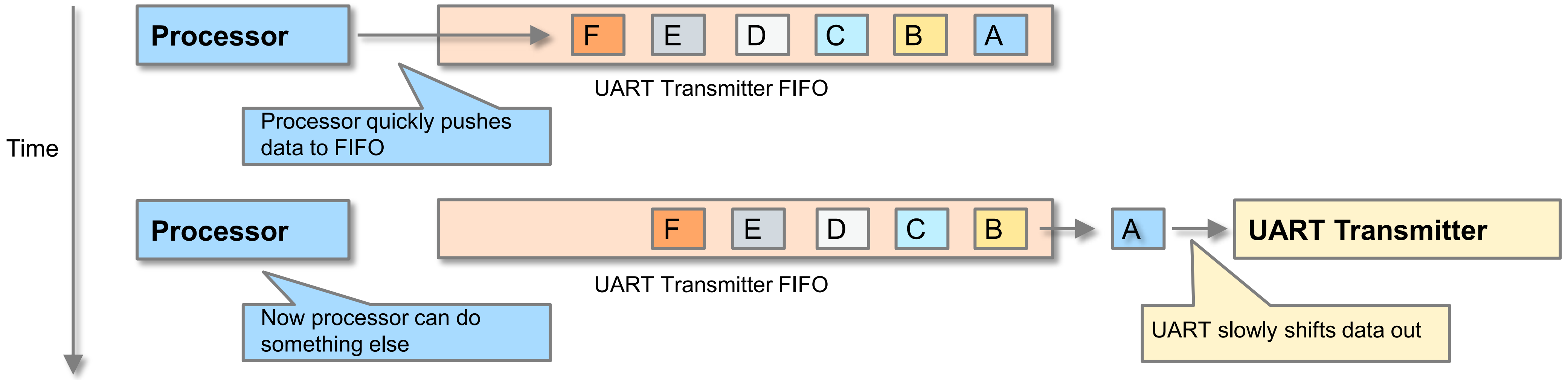
Dual Port RAM FIFO

- The address of the two ports is automatically managed as data move to/from the memory.
- The address can be coded in either natural binary code or Gray code.
- Binary code
  - Natural way of incrementing a number
  - One or multiple bits are changed when incrementing the number
  - Larger power consumption, longer switching time
- Gray code
  - Only one bit is changed in each increment
  - Less power consumption, shorter switching time

Dec	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

UART FIFOs

- Improve system operation efficiency
  - Processor operates at a higher clock frequency, e.g., 50,000,000Hz
  - UART is transmitted at a much lower frequency, e.g., 19,200 Hz
  - If the processor waits for the UART, a large amount of time is wasted.
  - This is why FIFOs are used to improve system efficiency.


Time
Processor
F
E
D
C
B
A
UART Transmitter FIFO
Processor quickly pushes data to FIFO
Processor
F
E
D
C
B
A
UART Transmitter
UART Transmitter FIFO
Now processor can do something else
UART slowly shifts data out

Memory Space

- UART peripheral registers are mapped to memory.
- Memory mapping depends on the implementation. For example:
  - Control register
  - Status register
  - Rx FIFO
  - Tx FIFO

Register	Address Offset	Size
Receive FIFO	0h	4 Byte
Transmit FIFO	04h	4 Byte
Status Register	08h	4 Byte
Control Register	0Ch	4 Byte

Example of AXI-UART memory map

Stream Data Transmission

- What is stream data?
  - A sequence of data elements that contain information
  - Sent from a source to a sink (usually continuously)
- Examples:
  - Sound
  - Video
  - Computer network traffic
  - Sensor data

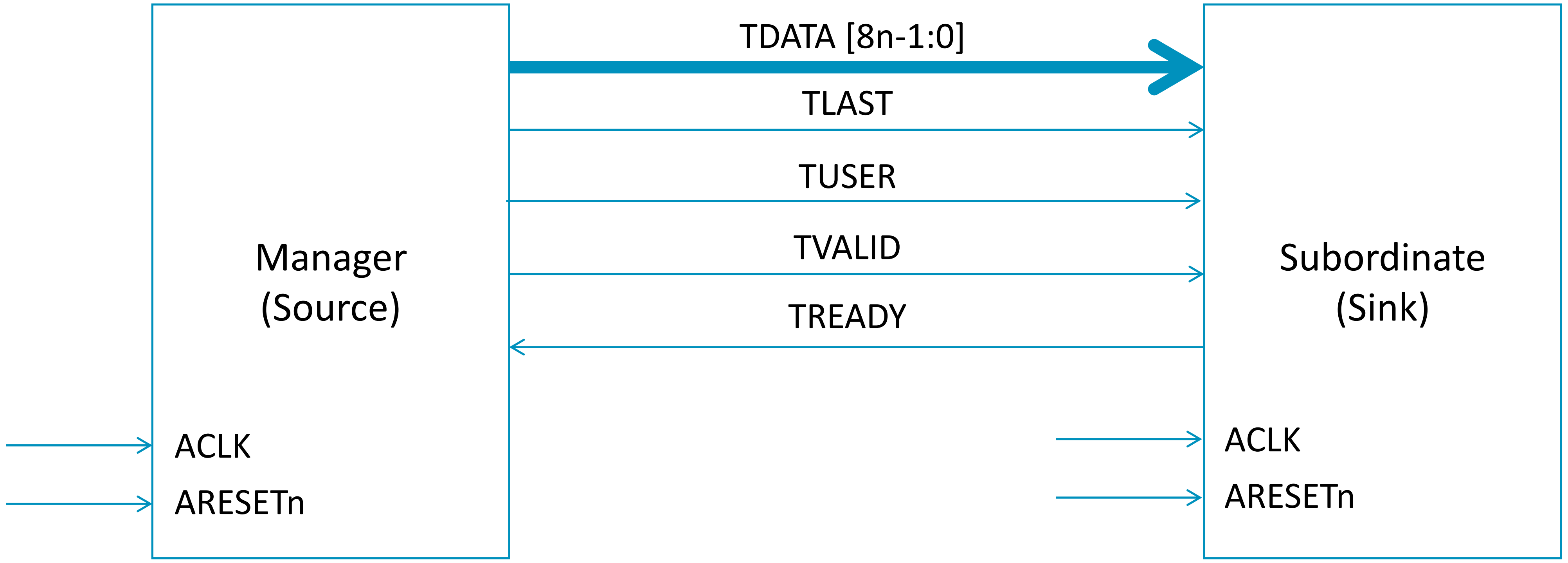
AXI4-Stream Protocol

- AXI4-Stream is an interconnect protocol used for stream data transmission.
- Supports a wide range of different stream types
- Single Manager, single Subordinate or multiple Managers, multiple Subordinate
- Easy to implement in SoC design
- Stream terminology:
  - **Transfer**: A single transfer of data across an AXI4-Stream interface. A single transfer is defined by a single TVALID, TREADY handshake.
  - **Packet**: A group of bytes that are transported together across an AXI4-Stream interface
  - **Frame**: A frame contains an integer number of packets.
  - **Data Stream**: The transport of data from one source to one destination

Data Streams

- Data streams take many forms.
  - Byte stream
    - The transmission of a number of data and null bytes
    - On each transfer, any number of data bytes can be transferred
  - Continuous aligned stream
    - The transmission of a number of data bytes, where every packet has no position or null bytes
  - Continuous unaligned stream
    - The transmission of a number of data bytes, where there are no position bytes between the first data byte and the last data byte
  - Sparse stream
    - The transmission of a number of data bytes and position bytes.

A Simple AXI4-Stream Mechanism



Signal	Source	Description
ACLK	Clock source	The global clock signal; all signals are sampled on the rising edge of ACLK
ARESETn	Reset source	The global reset signal; ARESETn is active-LOW

Manager Signals and Subordinate signal

Manager signals

Signal	Source	Description
TVALID	Manager	TVALID indicates that the Manager is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
TDATA [(8n-1):0]	Manager	TDATA is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
TLAST	Manager	TLAST indicates the boundary of a packet.
TUSER [(u-1):0]	Manager	TUSER is user defined sideband information that can be transmitted alongside the data Stream.

Subordinate signal

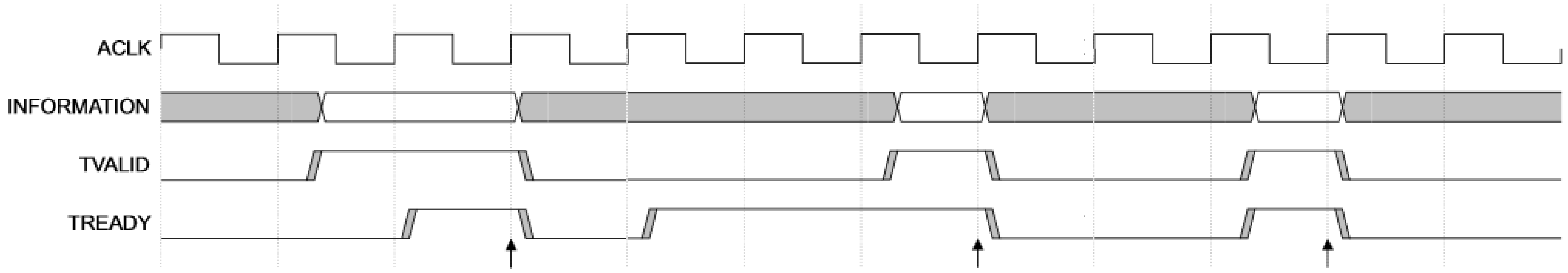
Signal	Source	Description
TREADY	Subordinate	TREADY indicates that the Subordinate can accept a transfer in the current cycle.

Clock and Reset

- Clock
  - Every component uses a single clock signal, ACLK.
  - All input signals are sampled on the rising edge of ACLK.
  - All output signal changes must occur after the rising edge of ACLK.
- Reset
  - A single active-LOW reset signal, ARESETn
  - Can be asserted asynchronously, but de-assertion must be synchronous after the rising edge of ACLK
  - During reset, TVALID must be driven LOW.
  - All other signals can be driven to any value

Handshake

- Transfer takes place in the cycle where TVALID and TREADY are both asserted.



Packet Boundaries

- TLAST can be used by a destination to indicate a packet boundary.
- The uses of TLAST are:
  - When de-asserted, TLAST indicates that another transfer can follow and therefore it is acceptable to delay the current transfer for the purpose of upsizing, downsizing, or merging.
  - When asserted, TLAST can be used by a destination to indicate a packet boundary.
  - When asserted, TLAST indicates an efficient point to make an arbitration change on a shared link.