# **Embedded Systems Essentials with Arm: Get Practical with Hardware**

# Module 1

KV3: Asynchronous Serial Communication

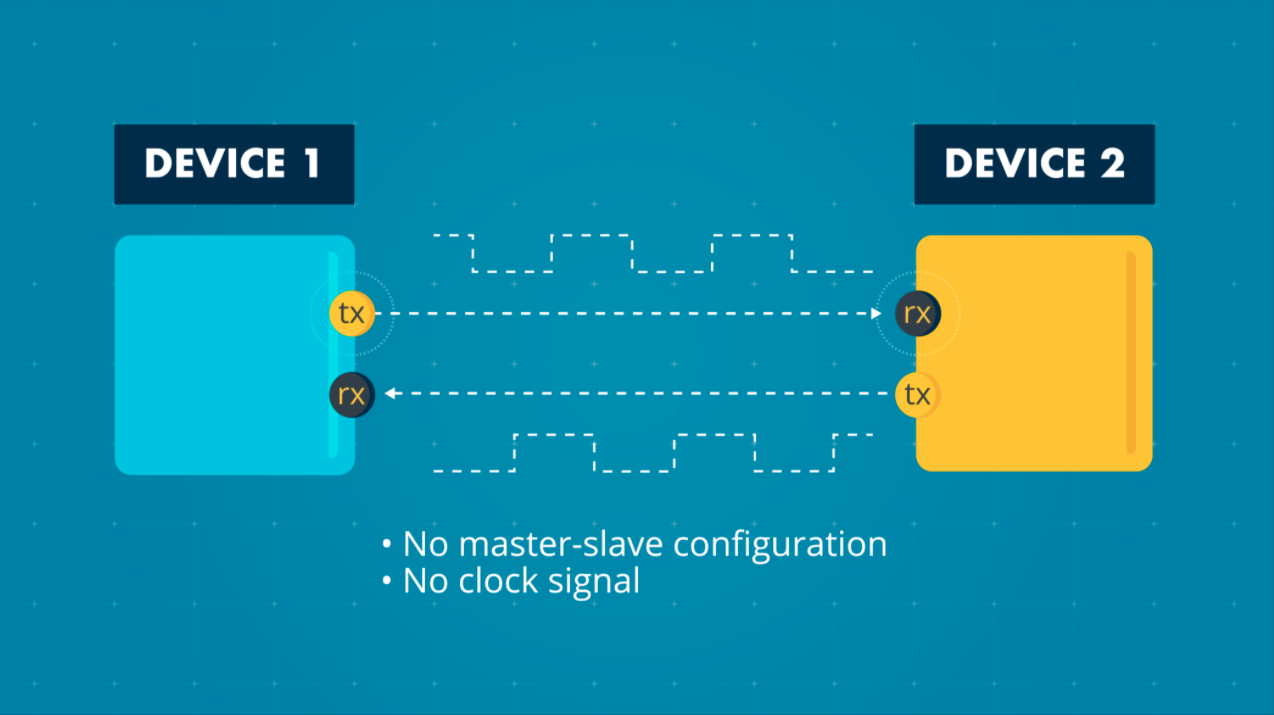
We’re now looking at asynchronous serial data communication. This is a serial data link where we’re not sending a clock signal from the transmitter to the receiver. The circuit block used to make an asynchronous connection is called a UART: Universal Asynchronous Receiver/Transmitter.

The UART has been around for more than 50 years. It’s been refined and improved over the years. These days we find UARTs embedded in microcontrollers and in other little sub-systems so that UART-based connections can be easily hooked up.

Unlike in synchronous communication, there’s no clock wire required in asynchronous communication.

The data transmission rate, known as the baud rate, is measured in bits per second and is pre-arranged between the transmitter and the receiver. The devices connected by a UART each have two wires: a transmitter, labeled tx, and a receiver, labeled rx. The tx of one device connects to the rx of the other.

Some UARTs have the ability to be also configured as synchronous, and are then called USART: Universal Synchronous/Asynchronous Receiver Transmitter. This diagram shows a 1-to-1 configuration, but it’s possible to make a longer connection as long as the tx of one device connects to the rx of the next device.



For example, it’s possible to create a ring configuration, called daisy chaining, where the data is transferred from the first device through the others.

Unlike in synchronous serial data communication, there is no master-slave configuration, which means there is no clock signal that coordinates data transmission. Because of this, framing bits are required to indicate to the receiver the beginning and end of a new data word. The data line will idle at Logic 1, when nothing is happening. The start of a new byte will then be signaled by a start bit which is at Logic 0. Eight bits will typically be transferred at a time. The receiver will recognize the end of that data word when it has clocked in 8 bits, and then sees a stop bit, i.e. Logic 1. That Logic 1 stop bit can merge with another idle stage, where it stays at Logic 1, or a new transmission can be started. If a new transmission is started immediately, the stop bit would give way to a new start bit after 1 bit’s duration.

Data can be transmitted in one of two patterns. The first is the standard 8-bit transmission that we have just looked at. The second is an eight bit plus parity pattern. The parity bit is an error-checking mechanism that helps to ensure data is correctly received.

Asynchronous serial communication has been widely used for transmitting alphanumeric data, so it’s worth reviewing how this is done. There are two in use:

Traditionally, the American Standard Code for Information Interchange (ASCII) has been used. ASCII encoded 128 characters: 95 printable characters (these are numbers and letters we’re familiar with in Western writing systems, such as “a”, “b”, “1”, “2”, etc) and 33 control characters such as next line, backspace, escape. These can be encoded in 7 bits, which are commonly stored as one byte for convenience.

In many applications, ASCII is now replaced by UTF-8 (Unicode Transformation Format—8-bit). This is compatible with ASCII and represents characters from global writing systems.

This table below shows 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​ |
| …​ | ​ | ​ | …​ | ​ | ​ | …​ | ​ |

The advantages of asynchronous serial communication are:

* Asynchronous serial links are good; we still use them today even though the technology was invented more than 50 years ago.
* Asynchronous serial communication is a comparatively simple and robust method of data communication over short to medium distances.
* We don’t have to transmit a clock around; there’s no need to connect it.

It’s downsides are:

* It’s a bit slower than SPI because we have to add framing bits (start and stop bits). They are an extra little overhead required for this type of communication.
* The hardware of the UART is more complex than the SPI because the receiver has to generate its own local clock and do the decoding of the incoming data, but the complexity is not overwhelming.

Overall, asynchronous serial communication isn’t suitable for complex or high-reliability data transfer. It’s best suited for simple, local data transfer.

**Visit arm.com for more information.**