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

## Module 2

## KV4: Review, Making Serial Design Choices

As embedded systems designers, we need to know the various tools at our disposal. In this module, we’ve added the I2C protocol as a possibility for serial communication. Let’s now look at what it’s good at and some of its weaknesses.

Its strengths are:

* You can successfully set up fairly complex communication networks.
* You can add nodes to the system and change your software without changing the hardware.
* I2C is more flexible than SPI because some devices have multiple internal registers with different I2C addresses and you can access those.
* I2C is more reliable than SPI in that if an addressed device doesn’t send acknowledgement, the master can recognize that there’s a fault in the system.

The weaknesses of I2C are:

* Sending address data (“start,” “stop” and “acknowledge”) can slow down the data transfer process.
* I2C doesn’t completely remove the risk of interference. This is because there is no error checking mechanism.

Overall, I2C is more reliable than SPI but is still not suitable for complex data communication and use in high reliability systems.

Let’s evaluate the four protocols we’ve seen so far. These are:

* SPI (serial peripheral interface)
* Asynchronous serial communication based on UARTs
* I2C
* The USB

As embedded systems designers, we want to recognize the strengths and weaknesses of each.

SPI is a good, simple method of data transfer. I2C keeps the strengths of SPI and addresses some of its weaknesses. It adds a bit of complexity to the signal and to the underlying software. There are many peripheral devices that we can hook up to SPI or I2C.

With asynchronous serial communication, we don’t have the annoyance of connecting the clock. This is a useful protocol, but it’s not suitable for use in high-reliability systems. This is equally true for I2C or SPI.

The USB has a big place in the world of computing and has a smaller role in the world of embedded systems.

The following table summarizes some useful applications of each of the four protocols we’ve looked at in the context of embedded systems.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Transfer Requirement​** | **SPI​** | **UART​** | **I2C​** | **USB​** |
| Fixed one-to-one or one-to-few, within device ​ | √​ | √​ | √​ | X​ |
| Flexible one-to-few, within device ​ | ​ | ​ | √​ | X​ |
| One (or more)-to-many, within device ​ | ​ | ​ | √​ | X​ |
| One-to-one or one-to-few, between devices, perhaps character-oriented​ | ​ | √​ | ​ | ​ |
| Bulk transfer, occasional, one-to-one​ | ​ | ​ | ​ | √​ |
| “Plug and play” capability​ | ​ | ​ | ​ | √​ |
| High reliability application​ | X​ | X​ | X​ | X​ |

√ = good choice. Blank = possible but not preferred. X = inappropriate

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