Embedded systems are designed to rely on various communication interfaces using external components & peripherals. We have General-Purpose Input/Output (GPIO, Serial peripheral Interface (SPI), & Universal Asynchronous Receiver/Transmitter (UART). Each one of these has its own purpose in embedded system design, highlighted by different capabilities such as speed & complexity.

GPIO is regarded as the most basic interface in an embedded system. The digital pins can be configured as either inputs or outputs, making the microcontroller able to read binary states or control simple devices. Some simple tasks include button presses controlling of LEDs. The response is instant since there is no additional communication needed, but it lacks ability to submit complex data or run error checking mechanisms. This is still a perfect application for items that require nothing more than basic digital signal control.

SPI is a bit more sophisticated solution in compared to GPIO. SPI uses clock signals to coordinate the data transmission between devices, enabling high speed communication. The interface features a clock (SLCK), master out slave in (MOSI), master in slave out (MISO), & chip select (CS). This allows for full-duplex communication, where data can be sent & received simultaneously. SPI can reach speeds up to 100 Mbps to assist with rapid data transfer used in memory chips or high-resolution displays. A drawback of it is the need for multiple physical connections.

Lastly, we have UART, which should be viewed as a balance between the two above. It eliminat4s the need for a clock sign by relying on predetermined timings for sync. UART relies on transmit (TX) & receive (RX), making it even easier to implement. With this in mind, it can be used to communicate between interfaces that don’t share a common clock source. Common applications include system debugging through serial consoles and communication with wireless modules. While UART is more versatile than GPIO and simpler than SPI, its asynchronous operation imposes speed limitations compared to synchronous interfaces.

If I had to pick one over there other, then it would be based on the need the system at the time to determine the best course of action. Simple systems could be quickly & easily programmed using GPIO. It’s the easiest to set up & perfect for basic tasks that don’t require super-fast data transfer. If I need complete a bit more advanced project that requires quick data exchange, SPI should be the preferred go-to. Though the setup for multiple wires may slow-down the initial process. If in need of separate systems or debugging, I believe UART is the optimal method. It can work through different systems while remain simple albeit, not the fastest.

Hopkins, J. (2021, December 1). *I2C vs SPI VS UART – introduction and comparison of their similarities and differences*. Total Phase Blog. https://www.totalphase.com/blog/2021/12/i2c-vs-spi-vs-uart-introduction-and-comparison-similarities-differences/

*All what you have ever wanted to know about GPIO, I2C, SPI, DAC, ADC, UART and more*. .NET nanoFramework Documentation. (n.d.). https://docs.nanoframework.net/content/getting-started-guides/beginner-explained.html