Space Wire on the Raspberry Pi Pico

Aidan Cooley, Justin Tripp, Zack Baker | CCS-3 Information Sciences, Mission and Integrative Computer Science

What is Space Wire?

The Spacecraft communication network

Space Wire is a full-duplex, bi-directional network communication protocol utilizing an eight wired data and strobe signal to communicate effectively across a spacecraft.

Who uses Space Wire?

Because of Space Wires efficient transfer rates, compatibility, and reliable transfer method it has become a great candidate for communication in LANL and across the world.
Possibly the best-known satellite using Space Wire today is NASA's James Webb Telescope, which uses the network extensively to handle various types of data.

Why use Space Wire over other communication?

Space Wire	Ethernet	I2C
Bi-Directional	Bi-Directional	Bi-Directional
8 wires	8 wires	2 wires
2-200 Mbps	10 Mbps - 40Gbps	100 kbps
Less Error Prone	More easily lose information	More easily lose information
Full-Duplex	Half/Full-Duplex	Half-Duplex
Built-in time protocol	No built-in time protocol	No built-in time protocol

The Pi Pico

This Pico is a microcontroller made by Raspberry Pi. It utilizes a 133MHz dual ARM Cortex-M0+ processor with 264 KB of SRAM.
And for this project we care most about the 8 programmable PIO state-machines that talk to the 30 GPIO pins.

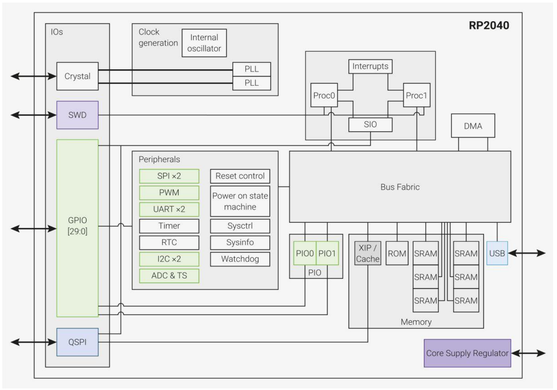

Image 1. The layout of a Pi Pico

Why Put Space Wire on a Pi Pico?

Most Space Wire protocols are implemented on FPGAs. Doing so increases cost and complexity. That is where the $4 Pi Pico comes in.
The goal of this project is to push the limits of the Pico to effectively implement Space Wire on a much simpler and inexpensive device.

Implementing Space Wire on a Pico

Due to the complexity, timing, and tricky bit math involved, we use the on-board PIOs(Programmable Input/Output) with their 4 separate state machines to send and receive the information.
These machines will take us even closer to the hardware. We can set how fast we clock them, how many clock cycles each command or bit lasts for and base our operations off varying signals across the board. The processor will pull bits from the main C code, send them across the wires, receive bits, and push back to C all with a measly seven assembly commands.

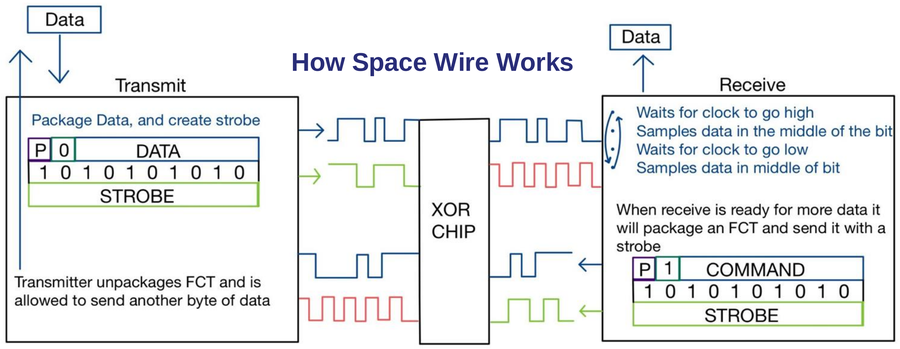

Image 2. Space Wire Network & Packet Layout

Each device utilizes six state machines and two cores:

Core 1: Transmit
1 – Send data
2 – Send data strobe
3 – Receive command

Core 2: Receive
4 – Send command
5 – Send command strobe
6 – Receive data

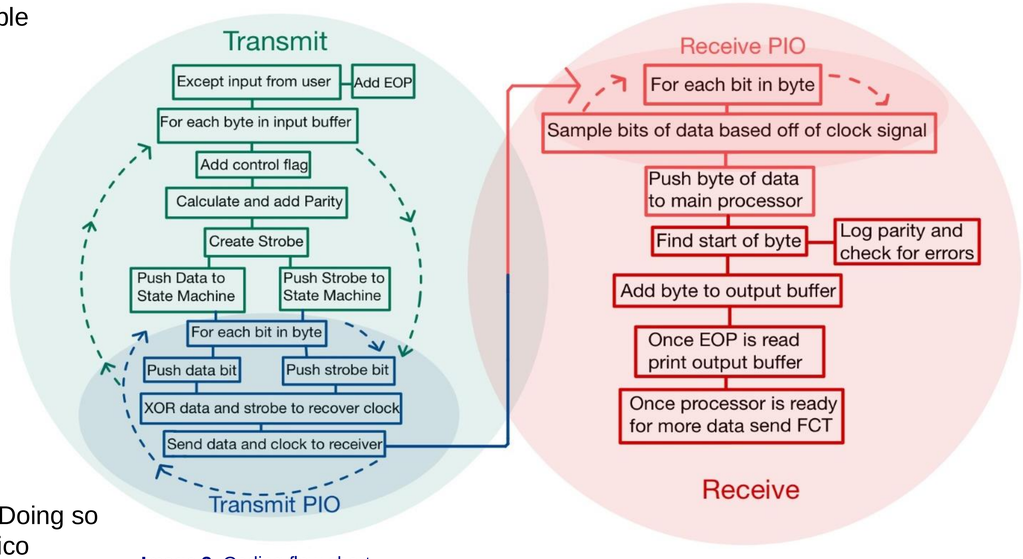

Image 3. Coding flow chart

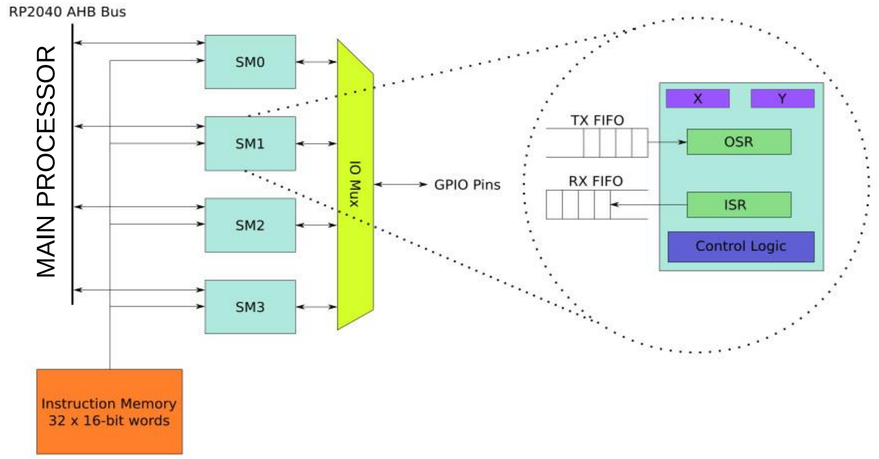

Image 4. Relationship between PIO and main Processor

Perfecting timing

The timing is a key component of this network protocol, while you can clock the state machines at the same speed as the main processor (133MHz) it is not necessary, nor a very good idea. However, it is essential that the receiver is running faster than the transmitter so it can more accurately sample the bits and won't miss any data.

In earlier iterations I attempted a few functions to perfect this timing:
- Oversampling the data and clock
- Using complex assembly commands to try to imitate more advanced logic such as an if/else block
- Folding the data and strobe signal into one, and separating again on receiving end

These proved to be too inconsistent, complicated, or useless and I settled on an external XOR chip to handle creation of the clock, and then waited to sample based off that signal.

The Result

Progress nearing the end of the internship

Space Wire on the Pi Pico (at the time of writing this) is bi-directional and can scan inputs from the user without disrupting the constant communication between devices. The transmit side can calculate parity, and package data as needed, sending an EOP once the excepted input is sent. While the receive side can build up the message and only returns it to the terminal after reading the EOP token.

Where to go Next?

- The next step is implementing all the small but key logic pieces of Space Wire; checking for error with parity, using other commands, etc.
- Next, speeding up the program from 5-10 MHz to be much faster while retaining functionality would be another useful feature to aid in testing against other boards.
- Then extensive testing to figure out where I can make the network more reliable with large data sets.
- Lastly, writing some pre-made user inputs and storing them in the on-board SRAM would enable other devices to consistently test against the Pico.

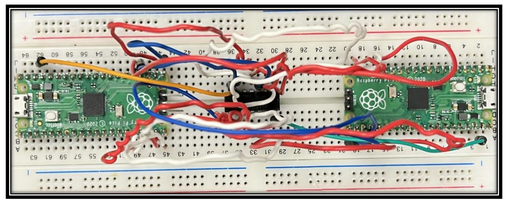

Image 5. Wired set up for two Picos

Los Alamos NATIONAL LABORATORY
NNSA National Nuclear Security Administration
Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.
LA-UR# LA-UR-24-26829