I^{2}CIP: Inter-Integrated Circuit Intra-networking Protocols

Design Report

Jayden Lefebvre - Founder & CEO, Lead Engineer

Northumberland County, ON, Canada

Primary Contact Email: contact@peapodtech.com

Contents

1 Design Abstract

The Intra-Integrated Circuit Intra-networking Protocols (I^{2}CIP) is a hardware design specification and communications protocols suite that enables hot-swappable, plug-and-play modular components for embedded systems. By leveraging existing I^{2}C components and enhancing them with a bus-switching architecture and dynamic communication protocols, I^{2}CIP allows for rapid prototyping and deployment of embedded systems with minimal configuration. Through this approach, I^{2}CIP aims to revolutionize the way embedded systems are designed and implemented, providing a flexible and scalable solution for developers and engineers.

2 Design Description

The I^{2}CIP technology is designed to facilitate seamless communication and interoperability between various I^{2}C devices in a modular embedded system environment. It achieves this by defining a set of hardware and software standards that enable hot-swapping of components, dynamic routing of data, and efficient management of device resources.

2.1   Hardware Specification

- **Electrical Isolator**: Prevents ground loops and protects sensitive components (**ISO1540**).

- **Level Shifters**: Allows communication between devices operating at different voltage levels.

- **Modules**: A physical collection of one or more I²C devices connected to a common switch, including one SPRT EEPROM and the switch itself.

    - **Switching Multiplexer** (MUX): A bus-switching multiplexer (**TCA9548A**). Eight possible MUX addresses (0x70-0x77) enables the instantiation of up to 8 modules per I²C bus.

    - **SPRT EEPROM**: A non-volatile memory device (**24LC32**) located on MUX bus 0 at default address 0x50. Stores a Static Partial Routing Table (SPRT) encoding intra-network location information for all devices on one switch's subnetworks.

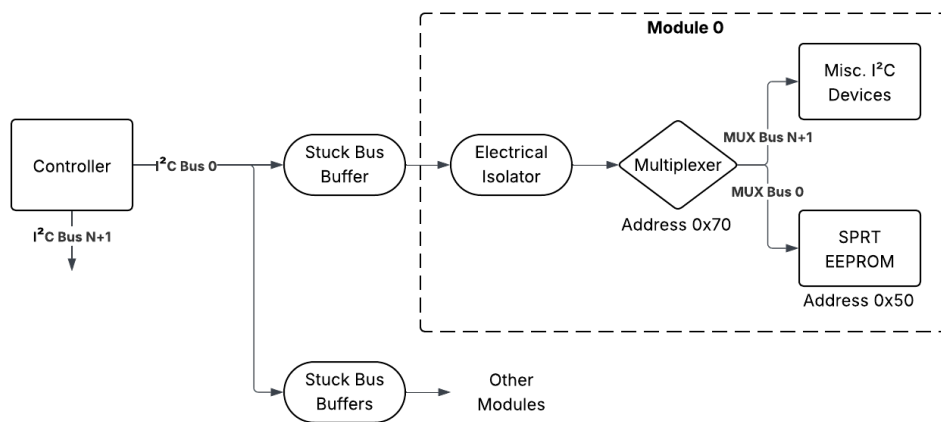   - **Stuck-Bus Buffer**: A buffer that holds the I²C bus state during module hot-swapping (**TCA4307**).

Figure 1: I²CIP Hardware Architecture Diagram

2.2   Software Protocols

- **Fully-Qualified Addressing** (FQA): Enables unique identification of each device in the network. Encodes I²C bus (3 bits; MSB 0-2), module number (3 bits; MSB 3-5), multiplexer bus (3 bits; MSB 6-8), and device address (7 bits; MSB 9-15) into a single 16-bit address.

- **Quality-of-Service Protocol** (QSP): Pings devices prior to and after any transmission, including multiplexers. Ensures transmission integrity and device availability using acknowledgments and retries with timeouts. Provides error level feedback to the Dynamic Routing Protocol (DRP).

- **Bus Switching Protocol** (BSP): Multiplexer control protocol that manages switching between multiple I²C buses on a single module.

- **Device Lookup Protocol** (DLP): Enables groups of device FQAs to be looked up by ID.

- **Reverse Device Lookup Protocol** (RDLP): Enables devices' IDs to be looked up by their FQA.

- **Intra-network Discovery Protocol** (IDP): Reads SPRT EEPROMs to build two parallel working maps of the intra-network topology: a hash table for DLP and a binary tree for RDLP.

- **Dynamic Routing Protocol** (DRP): Utilizes the working maps generated by IDP and the error level feedback from QSP to inform routing decisions for data transmissions; i.e. if a module is unreachable, DRP will remove the module's devices from the working maps.

3   Design Operations

For each type of device connected to an I²CIP network, a corresponding device driver must be implemented within the host system's software stack. These drivers are responsible for managing communication with their respective devices, interpreting data, and providing a standardized interface for higher-level applications. These classes inherit from a common `Device` base class (which contains the FQA data and implements the QSP and BSP protocols) as well as inheriting from either a `InputInterface` or `OutputInterface` (or `IOInterface`) abstract class, depending on the device's functionality. The `Interface` classes provide a templated set of methods for caching data read from input devices and , while standardizing argument types and return types across all device drivers.

For example, the `EEPROM` class (the device driver for the 24LC32 IC) inherits from the `Device` base class and the `IOInterface` abstract class, with template parameters `<char*, uint16_t, char*, uint16_t>`. In this order, these template parameters represent the input buffer type, input argument (read length), output buffer type, and output argument (write length), respectively.

Assumptions for operation of the I²CIP technology include:

- All devices are contained within modules that adhere to the I²CIP hardware specification.

- Each module contains a properly formatted SPRT EEPROM with accurate device information.

- All SPRT entries contain device IDs that correspond to actual device types for which a corresponding driver exists.

3.1   Static Partial Routing Table

The Static Partial Routing Table (SPRT) is a critical component of the I²CIP technology, serving as a non-volatile memory storage for intra-network location information of all devices connected to a module's switch. Each module contains one SPRT EEPROM that holds the FQAs and IDs of its connected devices, allowing for efficient device discovery and routing within the network. The SPRT contents are UTF-8 encoded JSON, structured as an array of objects representing each multiplexer bus:

```
// Example SPRT JSON Contents
[
    // Array of MUX buses
    {
        // MUX bus 0
        "24LC32" : [ 80 ],     // Decimal encoding of device address 0x50
        "SHT31" : [ 68, 69 ], // Two SHT31 devices on MUX bus 0, 0x44 and 0x45
        "MCP23017" : [ 32 ],  // Another example device, 0x20
        // etc.
    },
    // Up to MUX bus 7
    { },
    {
        "24LC32" : [ 80, 81 ], // E.g. two more different EEPROM devices on MUX bus 2
    },
    { }, { }, { }, { }, { }
]
```
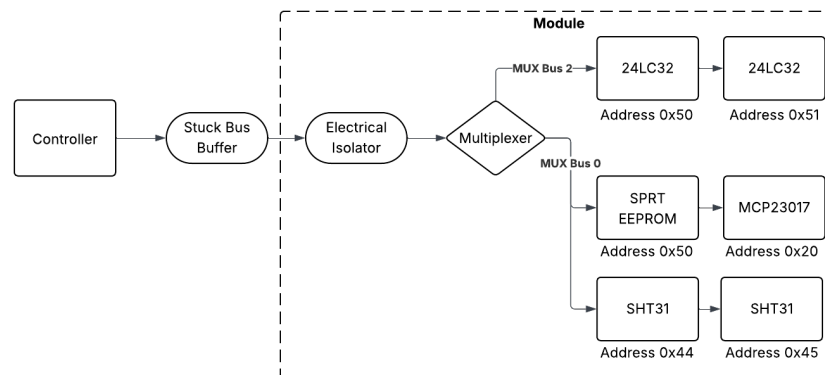

Figure 2: Graphical Representation of Example SPRT Module

4   Design Requirements

5   Design Performance

6   Design Potential