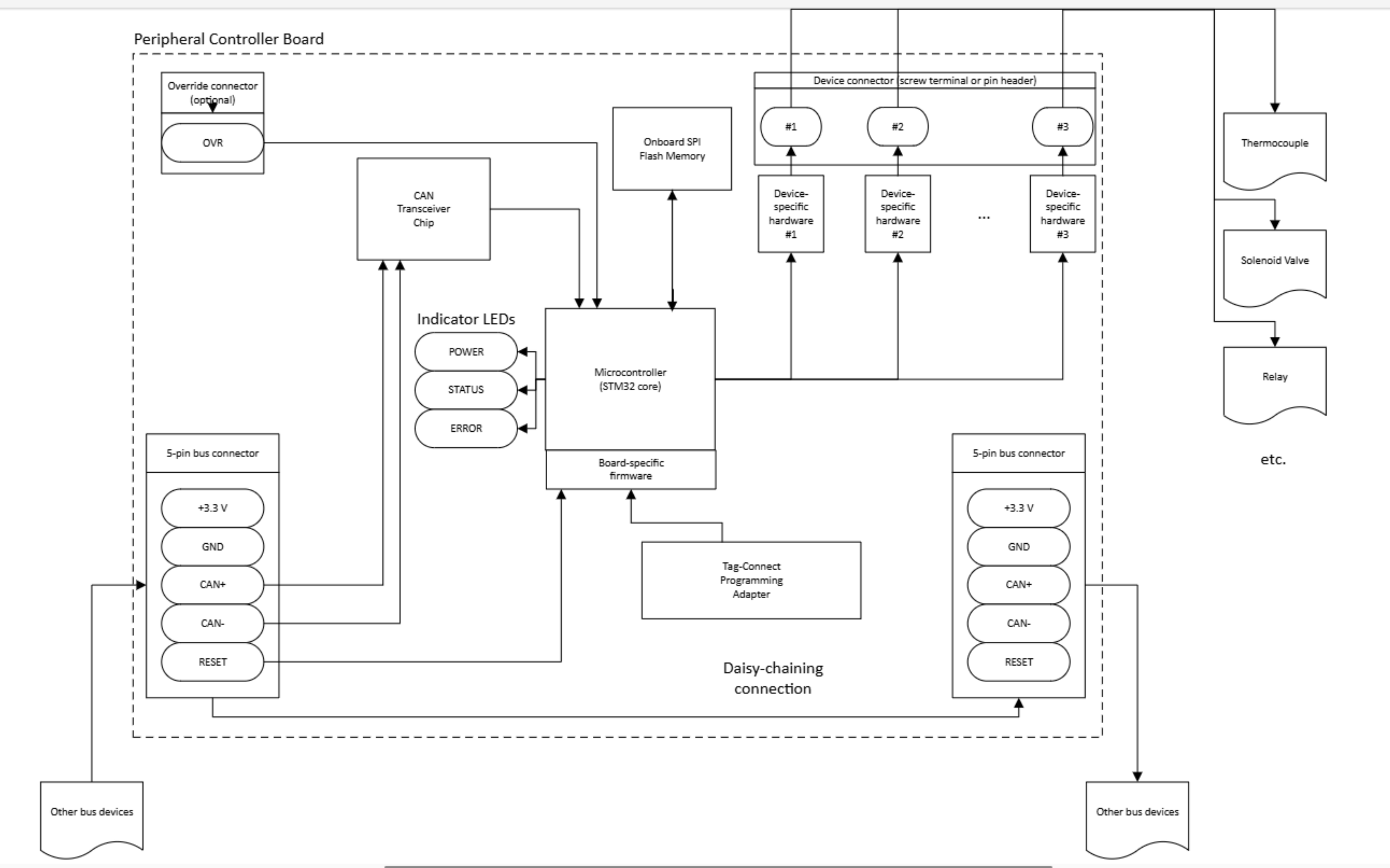
Project Goals

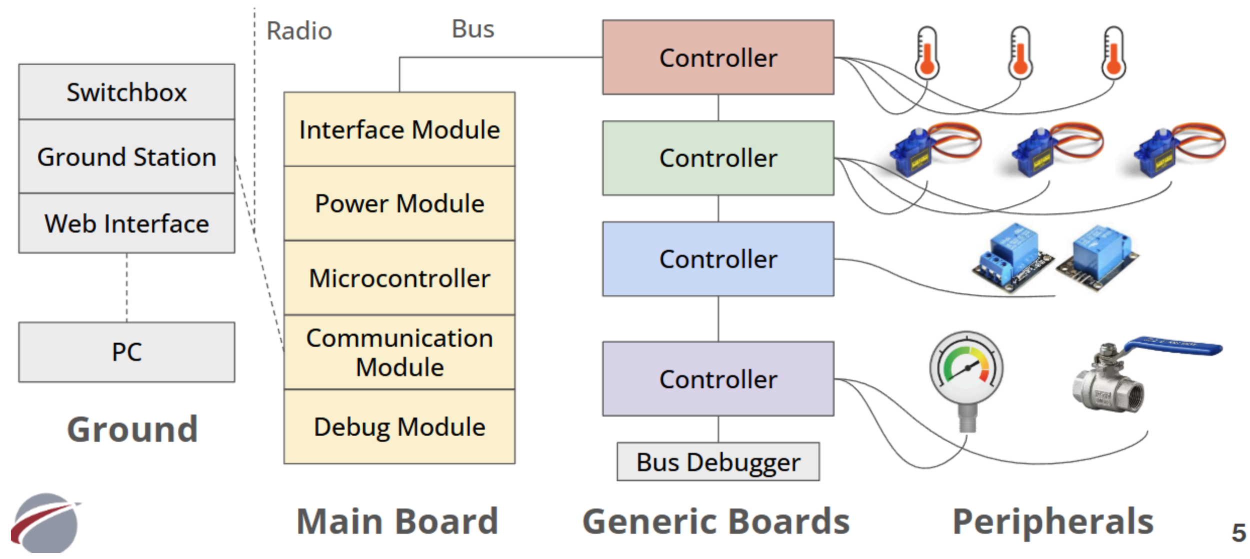
The goals of the **Papyrus System** are:

- To support all future Liquid Prop & Controls projects in avionics roles including DAQ, actuation, flight computer functionality, etc.
- To be adaptable to a new project without high costs/time input
- To be extensible as new functionality needed
- Generally, to free up design overhead for all future projects, on both the Avionics and Propulsion sides

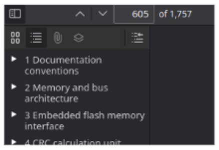
Papyrus Structure (as pictures):


Peripheral Controller Board
Override connector (optional)
OVR
CAN Transceiver Chip
Onboard SPI Flash Memory
Device connector (screw terminal or pin header)
#1
#2
#3
Device-specific hardware #1
Device-specific hardware #2
...
Device-specific hardware #3
Thermocouple
Solenoid Valve
Relay
etc.
Indicator LEDs
POWER
STATUS
ERROR
Microcontroller (STM32 core)
Board-specific firmware
5-pin bus connector
+3.3 V
GND
CAN+
CAN-
RESET
Tag-Connect Programming Adapter
Daisy-chaining connection
Other bus devices
Other bus devices

Papyrus Structure



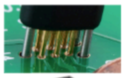
Hardware architecture


605 of 1,757

- 1 Documentation conventions
- 2 Memory and bus architecture
- 3 Embedded flash memory interface
- 4 CRC calculation unit

STM32 Microcontrollers

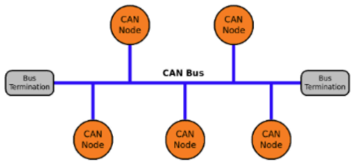
JST-XH



Tag-Connect

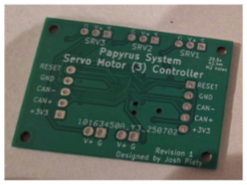
Screw Terminals

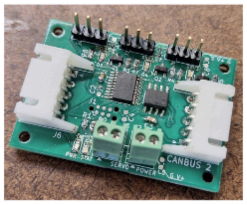
CAN Bus

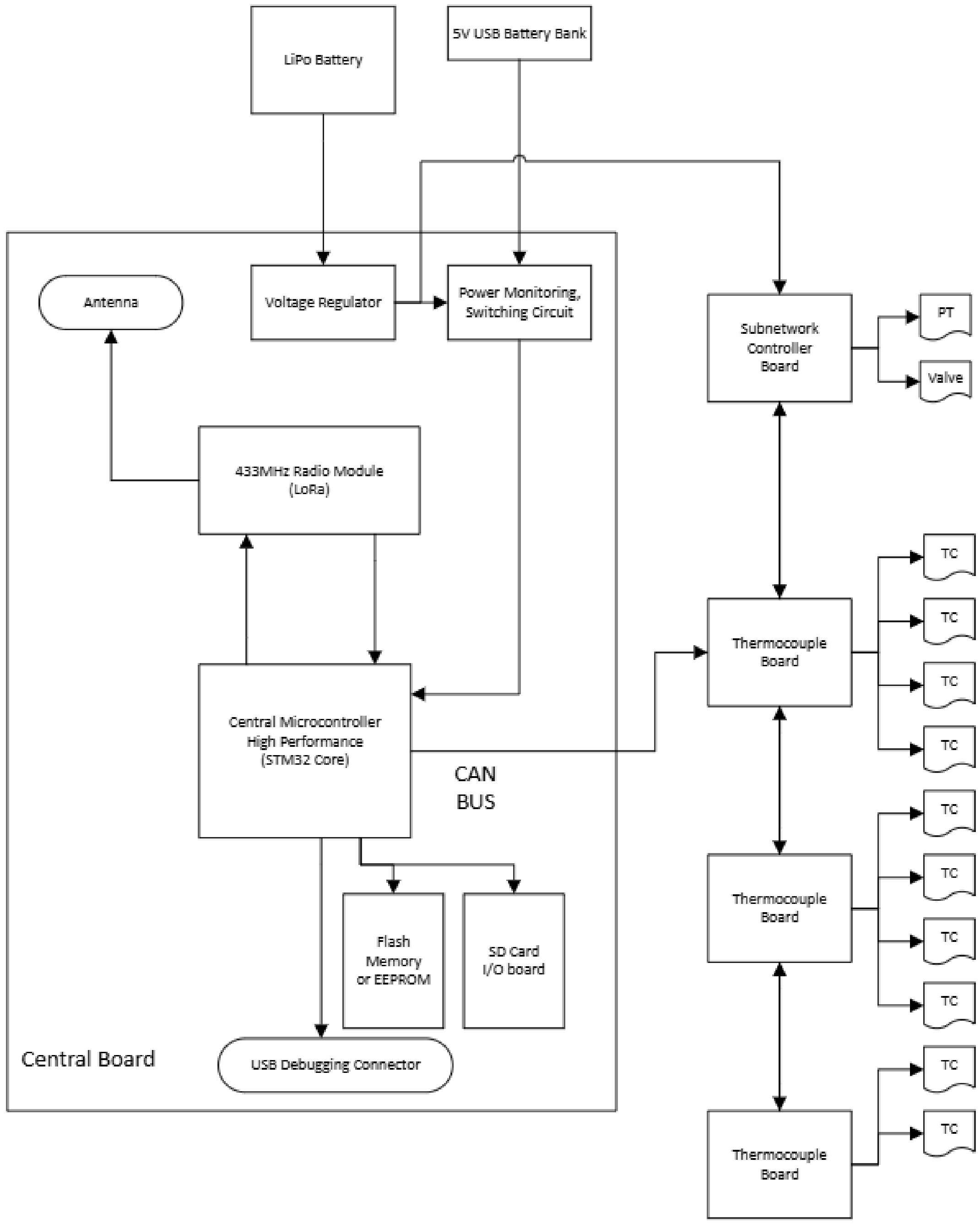


- Many nodes on one bus
- Noise resistant (differential)
- Built-in error handling
- High-speed and reliable
- Industry standard

SRAD boards







Papyrus Core Components

- Controller Boards - “control” a set of devices
  - Generic, interchangeable, and can be iterated on

    - Ex. 3 identical servo controller boards, each of which controls 3 servos
- Main Board - coordinates Controller Boards
    - Centralized functionality - radio, data storage, power supply (?)
    - High-power microcontroller to run control loops and data processing
- Ground Station - convenient DAQ interface
    - Skeletal functionality: radio receiver connected to web server and laptop
- Bus Debugger - make integration and testing easier
- External power hardware, processing hardware, etc.
- Main Board Components/Subsystems (Notes)
- Central microcontroller
- Should be a powerful STM32 with built-in CAN bus and a large number of pins - 64?
- Power management
- Power sources: buck converter from LiPo (12Vish), or USB power (5V) to an LDO
- Automatically switch between two power

sources as needed

- Monitor current passing through power supplies, and allow switching and monitoring of multiple power output ports
- Interfaces
- Multiple CAN JSTs
- STDC debug connector and standard Tag-Connect
- Pin headers breaking out pins
- USB-C port, screw terminals for power
- LiPo JST?
- SMA?? connector to antenna
- SD card slot
- Miscellaneous
- I2C and SPI buses for attaching peripherals
- Onboard temperature sensor
- SD card storage
- External flash memory chip
- Crystal oscillator
- Switches and buttons and stuff for

configuration

- Indicator LEDs
- Structure
- 4-layer board (We’re worth it)
- Submodules separated by board-edge connectors? To enable upgrades/replacement/fixing when i fuck something up
- Individual Modules:
- Microcontroller Module
- Contains STM32, supporting hardware, basic indicator devices, and flash, Tag-Connect
- Power Module
- Contains buck converters, switching hardware, monitoring hardware, and LiPo/USB power connector, and power output ports
- Data Module
- Contains radio, antenna connector, alternate interface for Ethernet?? or similar wired

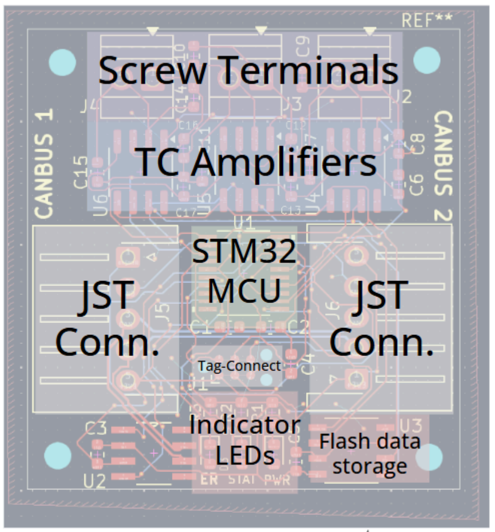
connection, microSD or SD slot

- Interface Module
- Contains CAN bus connectors, I2C/SPI bus, pin breakouts, STDC-14 connector, indicators
-
- Bus Debugger
- Contains CAN JST and LiPo screw terminal, battery pack, LCD display
-
- Components Selection & Guidelines
- All boards should be 2 layers and satisfy fabrication guidelines for JLCPCB
- MCU for controller boards: STM32C092FCP6
- CAN Transceiver: SN65HVD232QDRG4Q1
- SPI NOR Flash: SST25PF020B-80-4C-SAE
- Programming Connector: Tag-Connect 6-pins no-legs
- Connector type for CAN bus: JST-XH
- All components must run on 3.3V and consume as little power as possible

- Use 0603 passives if possible, 0805 if necessary, try to avoid going smaller
- Try to avoid no-leads packages (harder to solder) - SOP and QFP are fair game
- Include M2 mounting holes on corners/edges of boards if possible
-

Controller Boards

- “Nodes” of CAN network
- Each has a microcontroller, connectors, optional data storage and indicator LEDs
- Potential for complex tasks
- Incorporate as many functions into controllers as possible
- Low cost and low power usage



Ex. TC3Controller

10

Main Board

- Modular structure to ease development - PCBs joined with JSTs
- High-performance STM32 microcontroller
  - Ideally capable of running control loops, etc. in real

time, large data storage

- Radio communication using 433MHz or 915MHz radio modules
    - Many options, some with higher ranges (LoRa?) - ideally, allow upgrading
- Power control board - monitoring, buck converter and safety
    - More advanced switching functionality will be in a controller board
    - Designed to run off of a 12V 3S LiPo battery, with optional USB power, at 3.3V
- Interface module - breaks out necessary functionality
    - USB ports, SD card slots, CAN JST connectors, and miscellaneous GPIO

Ground Station & User Interface

- Ground Station: receives radio signals and hosts web interface
    - Designed for simplicity - basic shell between web server and Papyrus
    - Allows for direct control via a serial-attached laptop, with optional GUI
    - Optionally, physical interface for crucial functionality

    (e.g. igniter or tank valve)

- Web interface & software stack
    - Allows graphical representation of live state - data logging/plotting, P&ID status, fault conditions, and electrical current/voltage monitoring
    - Enables actuation or configuration of each type of controller
    - Multiple users can access at once, but only one can operate
    - Possible extension features: scripting/automation, custom data views, mobile app, data archiving/record

Solution 1: Design Decisions for Reliability

- Controller boards can be hot-swapped, so extras can be prepared and tested in case of hardware failure
- Documenting firmware and hardware legibly is essential
    - Non-Avionics people should be able to read and use documentation
- Error reporting in as specific a place as possible

    - Each controller board has FAULT indicator LEDs
    - Boards will report errors to the bus and retain error state
- Self-documenting hardware - silkscreen and labeling crucial
- Independent integration steps, so we find errors early

Solution 2: Standalone Bus Debugger

- Allows for independent troubleshooting of controller-specific problems, as well as hardware issues
- Self-contained user interface, connection to bus and power supply
- Speeds up integration (parallelization)
- Also a useful development tool!

Solution 3: Swappable Simplifications for Main Board

- Modularize main board PCB and develop simpler copies of each

- Best of both worlds - high complexity if things work well, but high reliability if they don't - on a component level
- Power board: independent power supplies with no switching
- Communication board: wired option, simplified hardware
- Interface board: extra debugging connectors for fixes
- Software stack: directly forward through commands and log data
- Could be expensive and time-consuming to develop

Solution 4: Monolithic "backup DAQ"

- At the extreme, CAN bus failure would paralyze the entire DAQ
- CAN bus is very reliable so this is unlikely, but would be critical
- Backup DAQ could control a subset of

devices
  - Simplified protocol, no GUI
  - Similar to Polaris DAQ
- This would be a lot of additional cost and development time and would be very inflexible, so this would only be done if it’s deemed essential for reliability