F*

FemtoStar Helium

Fault-Tolerant Computing for When 2 + 2 = 262148

Ethan Boicey, Core Technologies Developer – *hardware@femtostar.com*

More Contact Info (join our Matrix room!) – *femtostar.com/about-contact*

5th Annual PocketQube Conference

About FemtoStar, Briefly

F*

- “Satellite communications, done differently”
- The FemtoStar Project is a global community, organized online, developing a satellite communications network
- “Open infrastructure” architecture – no strict “user terminal” versus “gateway” distinction, can connect between any two terminals
- Focused on user privacy and security
- Two satellite designs, targeting PocketQube (codenamed *Azimuth*) and CubeSat (codenamed *Horizon*)

FemtoStar's Mission Presents Unusual Challenges

- Real-time communications service as part of a constellation – a brief outage may be okay for store-and-forward, not acceptable here
- Medium-to-long-duration mission – designed for 6+ years in service
- Designed to support orbits up to 1000 km – on the fringes of the inner Van Allen belt.
- Rideshare launch provides infrequent opportunities for satellite replacement if a "hole" opens
- "Holes" in constellation reduce coverage angle, may make coverage intermittent in some areas

Image Credit: FemtoStar

Space – A Harsh Environment for Electronics

- Single-Event Upset (SEU) – radiation-induced errors
  - Mitigate with redundancy – the focus of this talk
- Total Ionizing Dose (TID) - Long-term radiation-induced damage
  - Mitigate with shielding or rad-hard parts
- Rapid temperature and battery charge cycling
  - Especially in low orbits – ~4800 orbits per year!
- Diagnostics and fixes must be done remotely
- Virtually no opportunity to repair hardware… unless you're these guys

Image Credit: NASA, Public domain, via Wikimedia Commons

Option 1: “Failure Is Not An Option”

- Traditional satellites rely on ultra-specialized, space-grade hardware and tightly-controlled manufacturing processes to mitigate hardware failure
  - Radiation-hardened components, clean-room manufacturing, etc
- Complex, expensive validation processes
- Relatively “old” hardware often gets used because it has already gone through validation or has flight heritage
  - Performance and efficiency may be sacrificed to use “proven” parts
- Usually prohibitively expensive for FemtoStar, and for many other small satellite projects

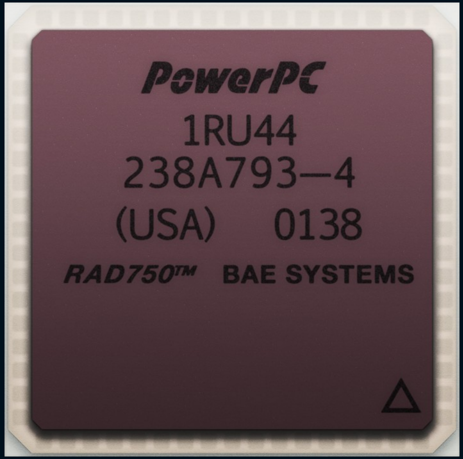
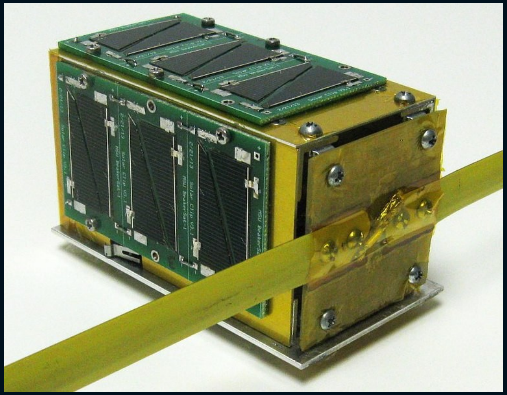


Image Credit: Henriok, CC0, via Wikimedia Commons

Option 2: "Failure Is Not An Issue"

F*

- Many small satellite projects take a dramatically different approach
- Build satellites like regular, "non-space-grade" electronic devices
- Some amount of extended component validation should still be done
- Short operational lifetime is acceptable for many scientific, amateur radio, or technology demonstrator missions
- Some proposals involve constellations of many such satellites, with a network tolerant of a few dead ones
- This can be a good idea, and is highly cost-effective
- For FemtoStar's mission, this is risky



Option 3: “Failure Is Not A Failure”

- FemtoStar takes an alternate approach – massively reduce the number of potential failures of components that can cause the failure of a satellite
  - Single points of failure have been near-entirely eliminated
- As with most small satellites, off-the-shelf components are used
  - Redundancy allows for lessened reliability demands on individual parts
  - Component selection remains cautious and reliability-focused (highest available temperature grades, flexible-termination capacitors, etc)
  - As with most COTS parts in small satellites, extended testing is still critical
- Satellite must attempt automated recovery and allow for manual diagnostics and maintenance, even when operating with damaged hardware

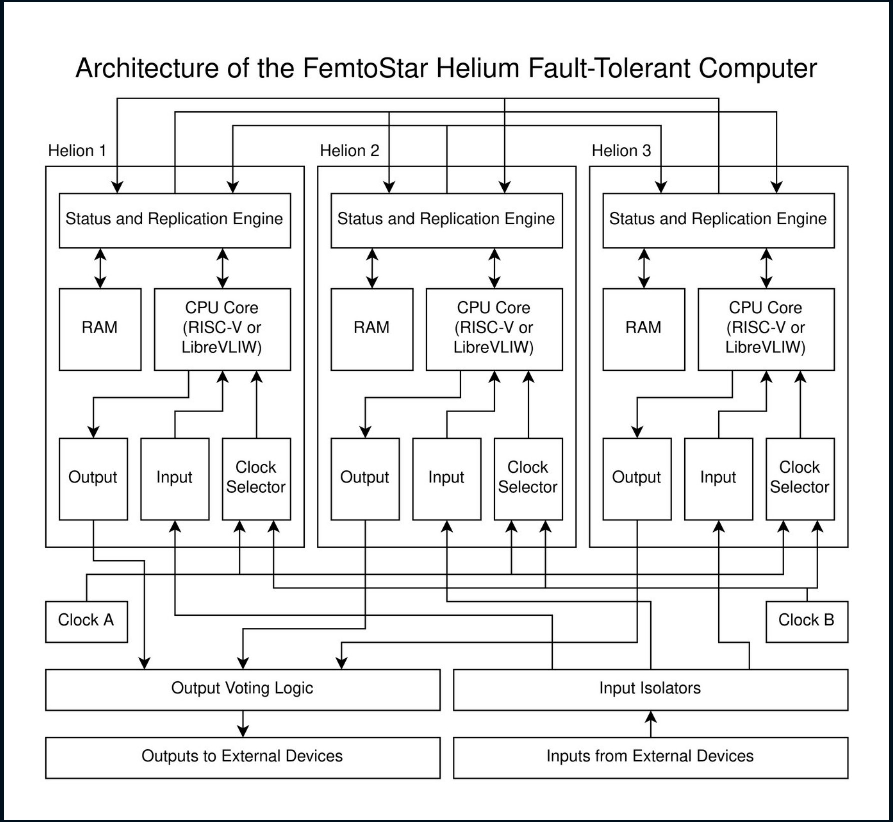
FemtoStar Satellite Architecture – What Is There To Break?

- Two independent electrical buses (one bus supported, but less redundant)
- Two, three, or four Service Transceiver Units (STU), which are independent from each other and have some internal redundancy
- ADCS magnetorquers and sensors, optional AIS thruster, etc.
- Optional backup telecommand-only transceiver, in addition to STUs
- Onboard compute (Helium – three units with multiple processors each)
  - Helium is responsible for monitoring, and, if failed, attempting to recover other hardware
  - “Core” hardware, serves as the central connection between other subsystems
  - Only complex subsystem without an onboard “twin” – demands high reliability

What Is Helium?

F*

- Triple-modular-redundant computer
- Three units, each known as a Helion
- Helions run in lockstep (cycle-for-cycle)
- All units monitor each other
- Designed to replicate state in real time when recovering a failed Helion
- I/O is achieved through voting – a faulty Helion will be outvoted by the other two.
- Runs from multiple redundant clocks

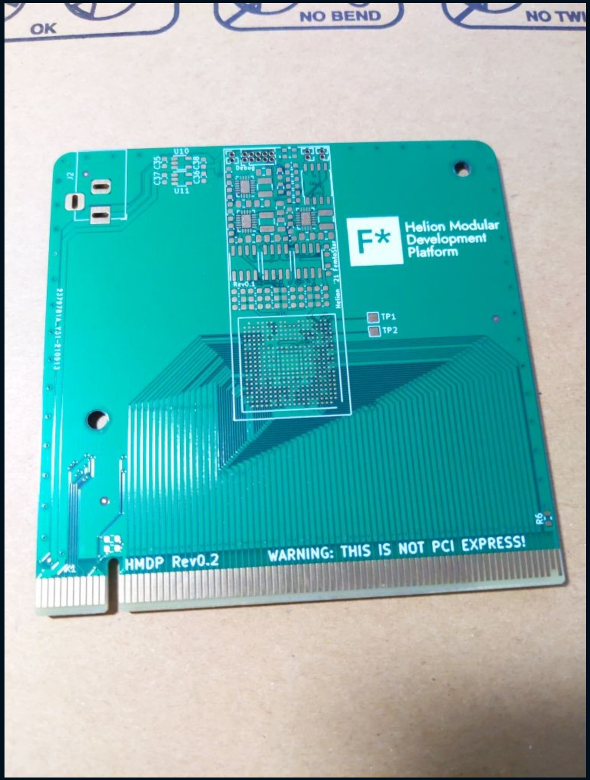


Architecture of the FemtoStar Helium Fault-Tolerant Computer
Helion 1
Helion 2
Helion 3
Status and Replication Engine
RAM
CPU Core (RISC-V or LibreVLIW)
Output
Input
Clock Selector
Clock A
Clock B
Output Voting Logic
Input Isolators
Outputs to External Devices
Inputs from External Devices

Image Credit: FemtoStar

Inside the Helion

- Based around a Lattice ECP5 FPGA
  - Quite well-reverse-engineered
  - Supported by a FOSS toolchain!
  - Not too expensive, quite power-efficient
- Relatively self-sufficient within Helium
  - Includes RAM (24MB external, ~58KB on FPGA) and CPU core (RISC-V or in-house LibreVLIW)
  - Generates its own power rails from the bus
  - Configuration flash is NOT shared
  - Selects its own clock from multiple available


NO BEND
Helion Modular Development Platform
HMDP Rev0.2
WARNING: THIS IS NOT PCI EXPRESS!

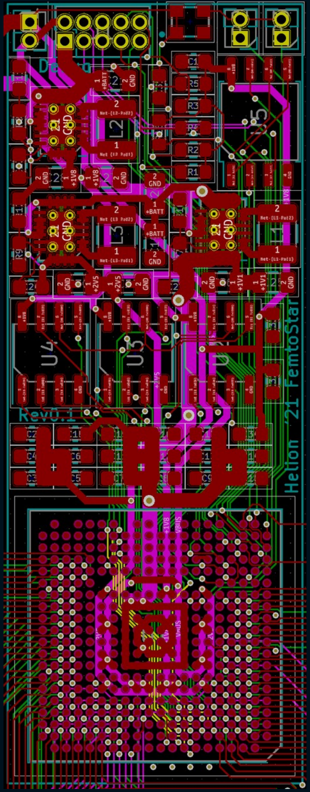


Image Credit: FemtoStar

Redundancy In Theory

- Most people have a relatively intuitive understanding of redundancy
  - “If one breaks, you have another” is a simple, intuitive concept
- Having two of a device allows one to fail, but not to disagree
  - If results differ, it is not clear which is correct – “never go to sea with two chronometers”
- Having three of a device allows one to disagree on outputs
  - Triple-Modular Redundancy: two correct results outnumber one incorrect result
- Many explanations of redundant hardware don’t go much further than this
- Actual implementation of this principle is more complex

Redundancy In Practice

F*

- Computers are theoretically deterministic – same inputs, same outputs
  - Multiple CPUs given the same inputs at the same times will have the same internal state
  - This only holds true if their initial states are the same and synchronization is maintained
  - Clock signal is an input too – must match, else lockstep synchronization is lost
- A failure in one unit is likely to cause it to fall out of sync with the others, or at the very least output incorrect data
  - Not imminently dangerous (faulty unit outvoted), but a TMR system already operating with one failed unit has no further redundancy against failure of another unit
  - Recovery from this state without resetting the entire system is not trivial
- Even with three units, something must still decide which output is correct

The Majority Gate

- “Voter” circuit - given three bits, decide whether the majority are 1 or 0
- Can create a single point of failure if you’re not careful
  - Voter failure is no worse than failure of whatever that voter outputs to
  - Nearly everything outside Helium on FemtoStar is redundant
- Can be extremely simple
  - Easily implemented with one or two discrete logic ICs
  - Even simpler discrete MOSFET implementations possible
  - Discrete FETs have large features, lack the parasitic thyristor structures that allow for radiation-induced single-event latch-up
  - Optionally, rad-hard parts can be used for non-redundant outputs (e.g. thruster)

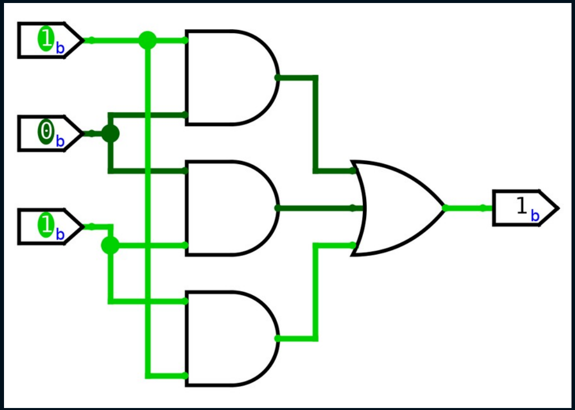


Image Credit: FemtoStar, created with logisim-evolution

Problem 1: The Clock

- In order to maintain lockstep, all units must work from the same clock
  - Specifics of this clock are not as important as the fact that there is “one true clock”
- The need for a single, shared clock signal runs counter to redundancy
- Three clocks and a majority gate is not practical – minor real-world phase/frequency differences will produce an unusable output
- Central “clock supervisor” devices result in single points of failure
- The solution: feed all clocks to all Helions, let them choose deterministically
  - If each Helion chooses a clock signal the exact same way, all will choose same clock
  - Failure in clock monitoring becomes no worse than any other failure of one Helion

Problem 2: System Monitoring

F*

- The system as a whole should know the status of all units
- However, a unit must not know which unit it is, lest that affect its outputs
  - Remember: same outputs depend on same inputs, and this is a differing input
- Central “system supervisors” are single points of failure
  - Helions must determine each others status, but this means knowing which unit is which
- The solution: expose the unit number only to a “Status Unit” on the Helion
  - Status Unit combines its known unit number with information from other Helions
  - Determines the status of the entire system, including all Helions, based on this
  - Exposes only system status to its host processor – all CPUs see same status

Problem 3: Detecting Failure

F*

- Majority gates will prevent faulty outputs, but don’t alert you to failures
- Logic failures, especially on SRAM-based FPGAs, can be subtle
    - Logic gates are LUTs in SRAM – one bit flip in a LUT can make 2+2 = 262148 in an adder
    - An internal failure that has not yet affected output should still be detectable
- Helions need a way to “prove” to eachother that they are working
- The solution: expose a “state check” bit to the other Helions
    - Check bit generated based on various pieces of the Helion’s internal state, each cycle
    - Provided to the Status Engine of other Helions to check against other state check bits
    - Must be correct for millions-to-billions of cycles before a Helion is deemed operational

Problem 4: Replication

- When a unit fails, you need to try to bring it back before another fails too
  - Remember: TMR only protects against one failure at a time
  - After a Helion is recovered (e.g. by watchdog reset, automated power cycle, workaround FPGA bitstream from ground crew, etc), state must be transferred back onto it
- This is trivial if all Helions are reset, and this is available as a backup
- However, to do so without interrupting service is more complex
- The solution: the Status and Replication Engine
  - SRE "shadows" memory writes from working Helions to target Helion in real-time
  - Unused memory cycles used to scrub memory and replicate the scrubbed data
  - Once all memory is replicated, registers are dumped and replicated, target is started

F*

Thank You!

Questions are welcome, if time permits

Ethan Boicey, Core Technologies Developer – *hardware@femtostar.com*

More Contact Info (join our Matrix room!) – *femtostar.com/about-contact*

5th Annual PocketQube Conference