Software Verification of an Emergency Recovery System for Micro Air Vehicles

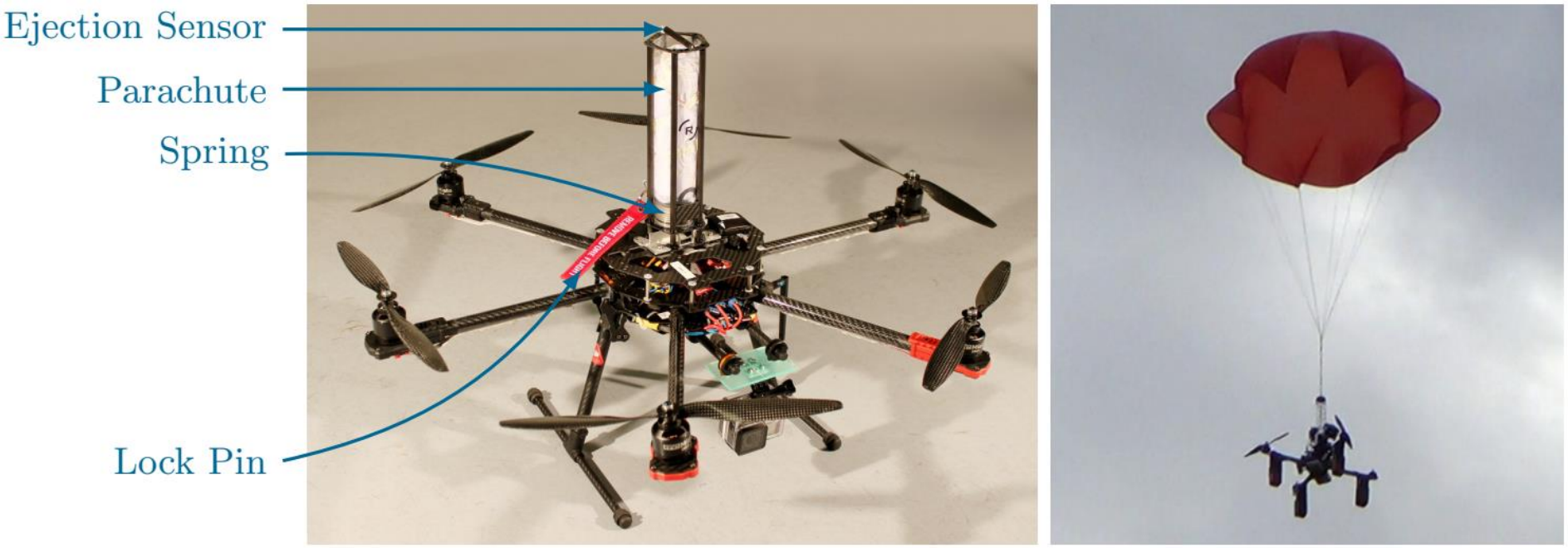
DOI: 10.1007/978-3-319-24249-1_32
Martin Becker, Markus Neumair et all.
Computer Safety, Reliability and Security, 34th International Conference SAFECOMP 2015

Software Verification and Validation

PhD student: Sergey Pogorelskiy
Professor: Dr. István Majzik
BME, 5 December 2019

Micro Air Vehicles (MAVs)

Prototypes of Emergency Recovery System



MAV safety systems

- MAVs that ship with a parachute system:

-MCFLY-Helios
-DropSafe for the *DJI Phantom*

- Operated solutions:

- Opale
- SKYCAT
- MARS

Challenges

- Low weight for system
- Interface to the MAV must be minimalist
- Software (deciding whether there is an emergency, and triggering the recovery independently of the pilot)

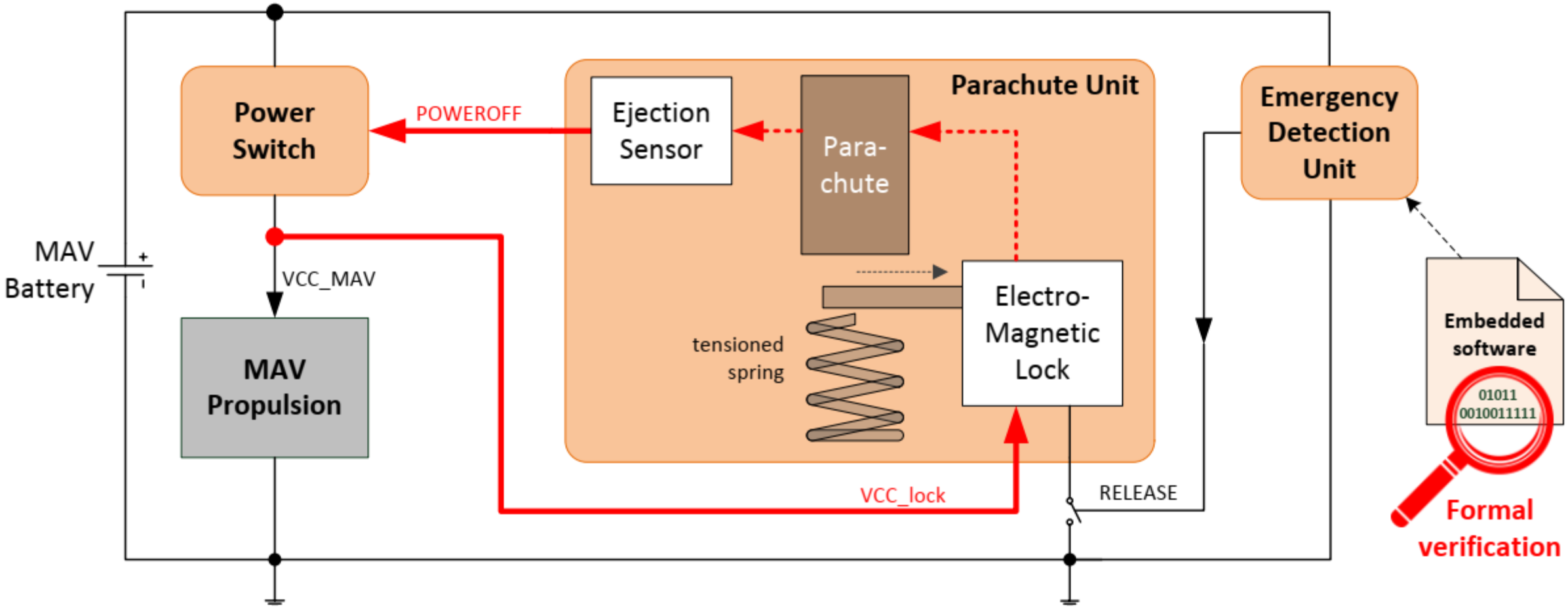
Proposed Emergency Recovery System for MAVs

It is a parachute system, designed to increase the overall safety of the MAV. In case of an, the ERS automatically turns off the propulsion and deploys a parachute.

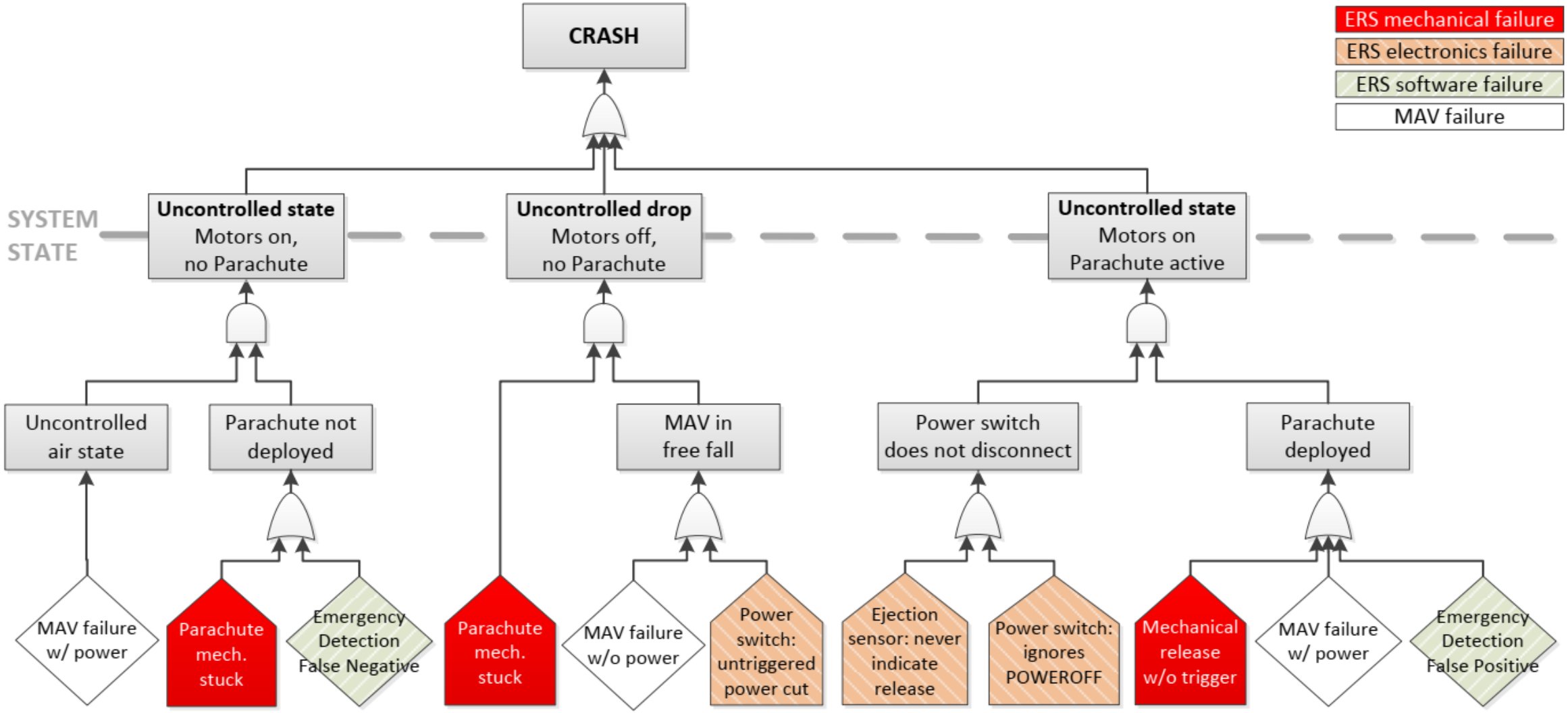
The technical specifications:

Property	Value
total weight	320 g
input voltage	6. . . 25.2 V (2. . . 6 LiPo cells)
power consumption	< 3 W depending on propulsion state
worst-case trigger time	≤140 ms
terminal speed & min. altitude	4.5 m/s within 10 m

Internal Structure



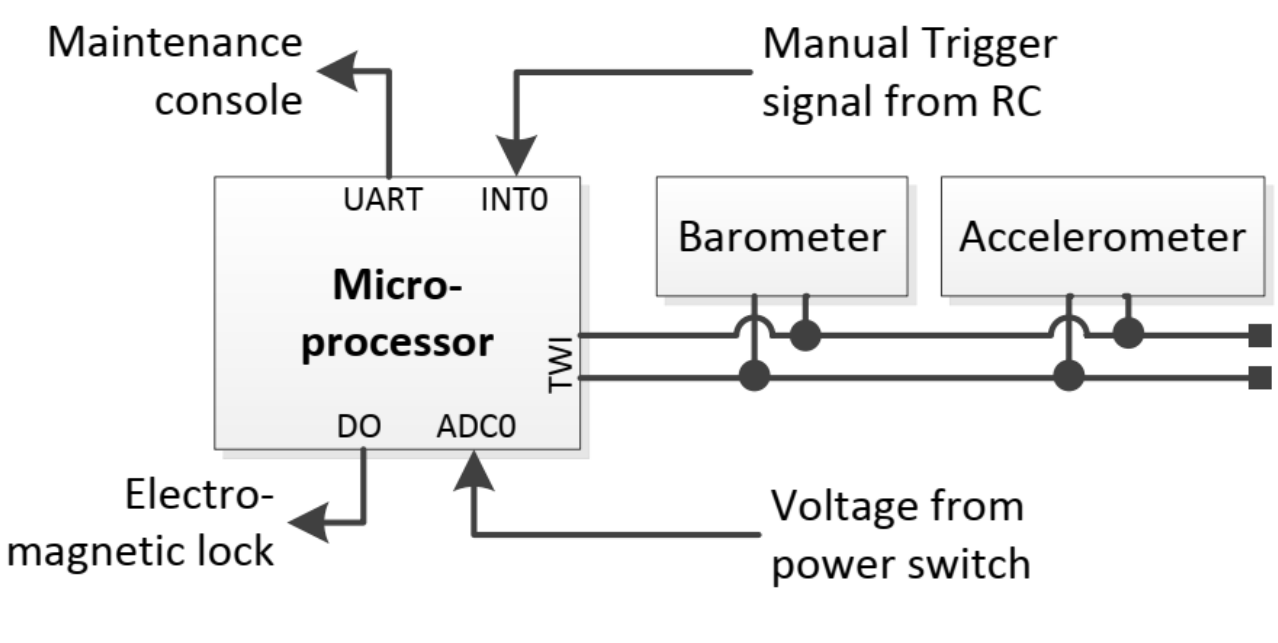
Fault Tree

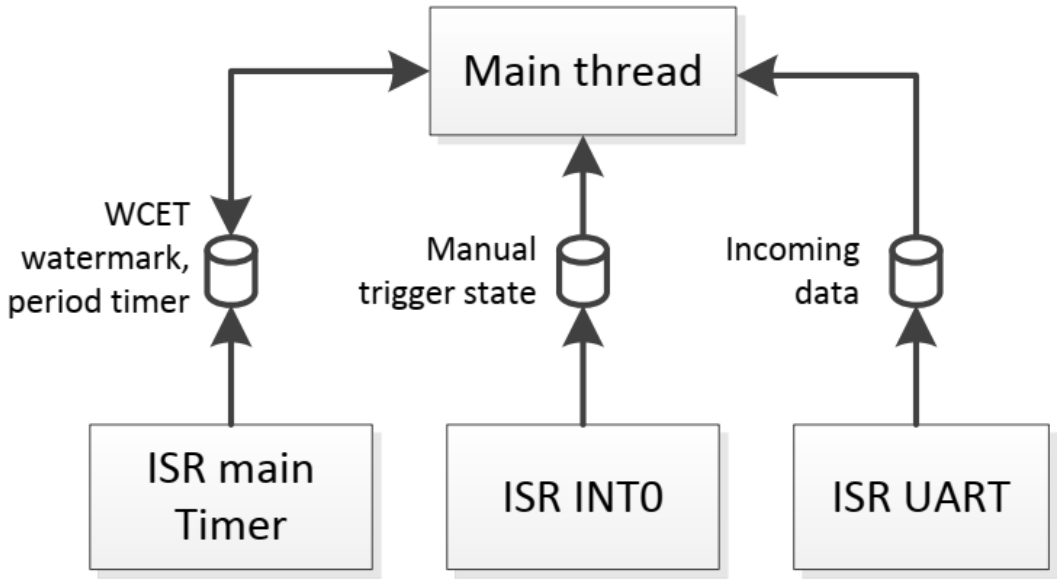

ERS mechanical failure
ERS electronics failure
ERS software failure
MAV failure
CRASH
SYSTEM STATE
Uncontrolled state
Motors on,
no Parachute
Uncontrolled drop
Motors off,
no Parachute
Uncontrolled state
Motors on
Parachute active
Uncontrolled air state
Parachute not deployed
MAV in free fall
Power switch does not disconnect
Parachute deployed
MAV failure w/ power
Parachute mech. stuck
Emergency Detection False Negative
Parachute mech. stuck
MAV failure w/o power
Power switch: untriggered power cut
Ejection sensor: never indicate release
Power switch: ignores POWEROFF
Mechanical release w/o trigger
MAV failure w/ power
Emergency Detection False Positive

Software Verification

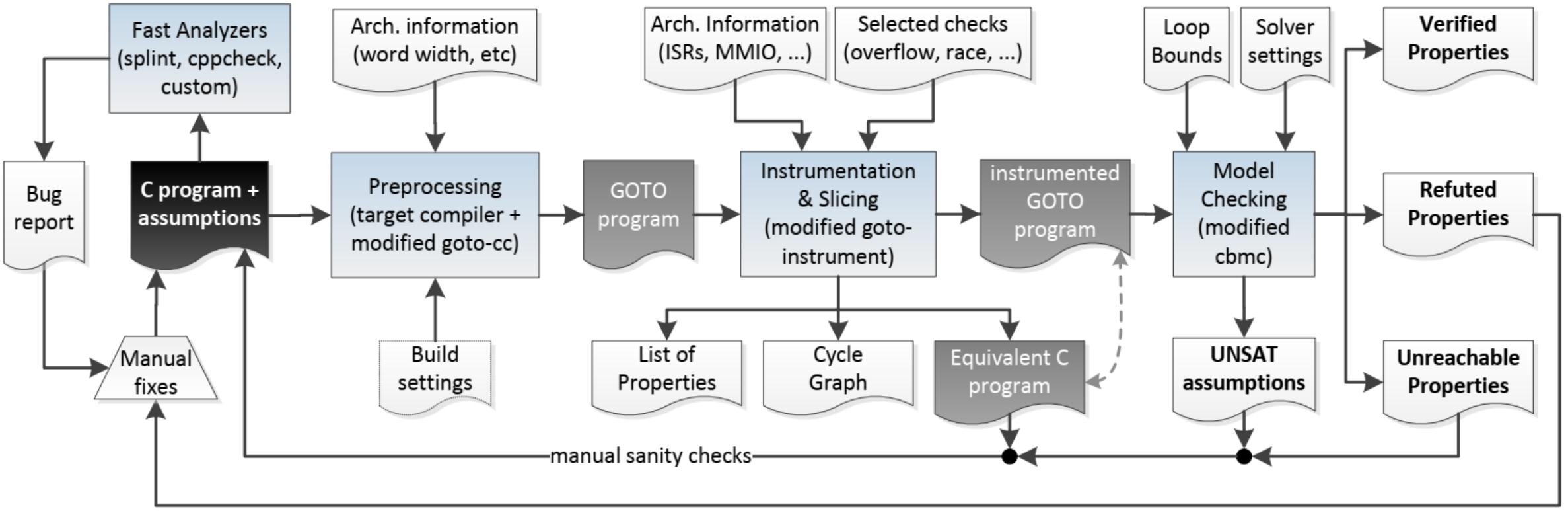
Software Structure:
1. **Initialization:** Initializes all sensors, and captures environmental conditions (e.g., pressure at ground level). When completed, the ERS switches to *self-check mode*.
2. **Self-Check:** To ensure that there is not already a failure in the ERS during startup, we added built-in self tests covering the major subsystems of the ERS. When completed, the ERS switches to *detection mode*.
3. **Detection:** The software periodically reads all sensors and estimates the MAV's air state. If the emergency conditions apply, the EM lock is released and the software switches to *emergency handling mode*.
4. **Emergency Handling:** Current sensor data and decision conditions are written to EEPROM, to enable a post-flight analysis.

Microprocessor with interfaces to its environment (left) and the resulting concurrency in the software (right).





Verification Workflow



Missing Architectural Information

A problem in static verification is implicit semantics that depends on the target, for example that certain functions are set up as interrupt service routines (ISRs) and thus their effect needs to be considered, although they never seem to be invoked. Another example is memory-mapped I/O, which may seem like ordinary reads from memory, but in fact could inject nondeterministic inputs from the environment.

Memory-Mapped I/O

All I/O variables (the sensor inputs) must be annotated to be nondeterministic. One option for that would be using the flag *--nondet-volatile* for *goto-instrument* to regard all volatiles as nondeterministic, however, this results in overapproximation for *all* shared variables (which are volatile as well), allowing for valuations which are actually infeasible due to the nature of the algorithms operating on the shared variable.

Verification Results

Mode →	Initialization	Self-Check	Detection	All
lines of code	1,097	976	1,044	2,513
#functions	36	29	43	94
#persistent variables	36	38	59	72
#live variables at exit	31	31	n.a.	n.a.
#properties	249	221	175	458
#VCCs	11,895	35,001	15,166	330,394
#SAT variables	5,025,141	8,616,178	6,114,116	n.a.
SAT solver run-time[a]	16 min	14 min	28 min	infeasible[b]

[a] On an Intel Core-i7 vPro at 2.8 Ghz and 4 GB RAM.

[b] Out of memory after 3 hours; #VCCs and SAT variables were still growing.

Conclusion

In presentation were described approaches in developing a safety-critical emergency recovery system for MAVs, in particular efforts in applying methods and tools for formal verification of embedded software. This study has shown that formal verification of the entire, original software running on a microcontroller is possible, if appropriate preprocessing techniques are applied.

Thank you for your attention!