
# N7 SPACE

## ATMEL ARM BSP with CANopen library

Michał Mosdorf, Michał Kocon & The Team

Final Presentation Days 3-4 Dec 2019

# Agenda

- N7 Space introduction
- Project scope and objectives
- Design and development aspects
- Testing approach
- Conclusion and project continuation

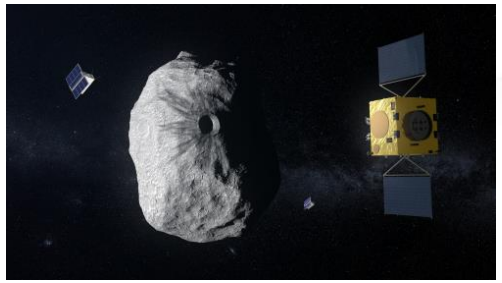
# N7 Space

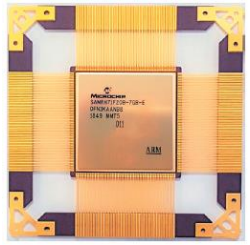
- Joined venture between SPACEBEL and N7 Mobile located Poland
  - Company founded in 2017
  - Started operation with projects previously executed by N7 Mobile that were transferred to N7 Space
  - Software engineering team located in Warsaw office with space experience since 2014
  - Focus on software development for upstream segment
    - On-board software
      - Leon3, Cortex-M7, Zynq
    - MBSE
      - ASN.1, SDL, AADL, MSC, Capella, TASTE

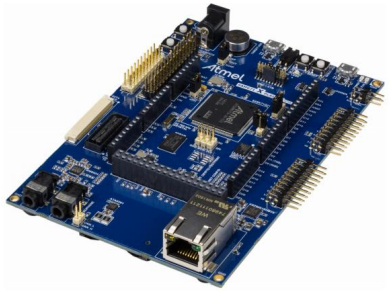

# Selected activities

- ESA missions
  - CBK's subcontractor in PROBA3 mission responsible for on-board software
  - SPACEBEL's subcontractor in HERA mission
- Software development for ARM ecosystem
  - Board Support Package and Boot Loader development for Microchip Cortex-M7 processor line
  - CANopen library and RTEMS 5 demo applications
  - CoreSight usage for multi-core software tracing on ARM Cortex-A53 Zynq

©ESA–P. Carril



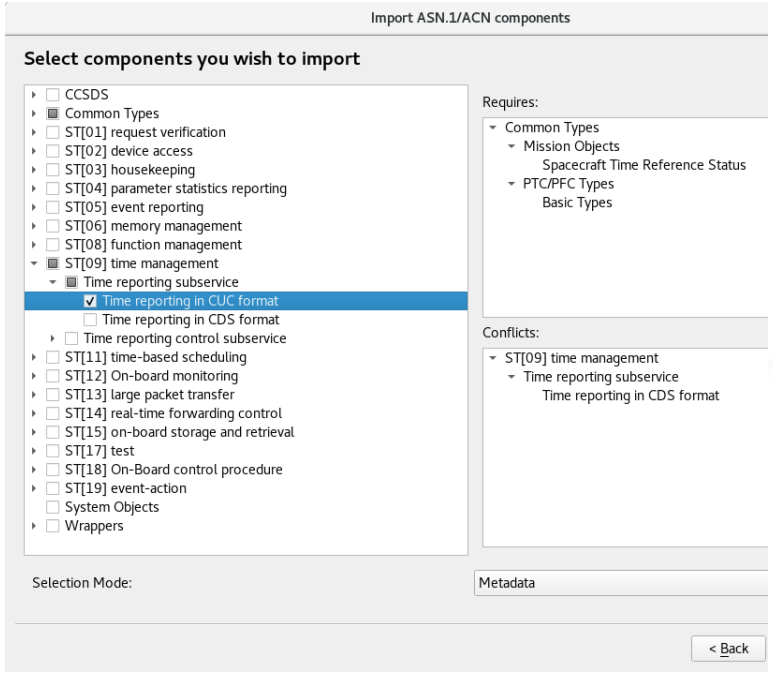




MICROCHIP

# Selected activities

- MBSE tools development
  - Qt based IDE for ASN.1 modelling with PUS-C template library
  - PUS-C deployment with automatic generation toolset
    - Database software, document generation ASN.1 modelling and generation
  - Capella plugins development
    - ASN.1 and AADL generator from Capella models
  - Test scripting languages
    - EBNF, Language Server Protocol Based IDE for ECSS languages
  - Providing TASTE support to target platform by integrating new compiler toolchain
  - Member of TASTE Steering Committee

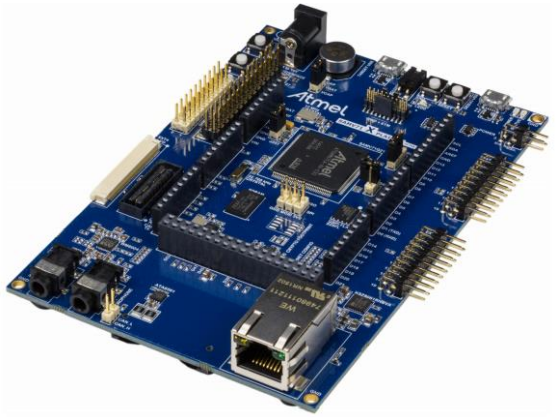



ASN.1 ACN taste
The Assert Set of Tools for Engineering

Capella

# ARM BSP with CANopen library

- Project executed under ESA Polish Incentive Scheme with Microchip
- Software development activities for SAMV71 Cortex-M7 MCU
  - Bootloader compliant with the ESA SAVOIR requirements
    - Utilization of PUS-C stack supported by ASN.1/ACN formal modelling
  - Board Support Package
    - Driver library for MCU
  - CANopen library implementing tailored ECSS-E-ST-50-15C
  - Demonstration applications based on RTEMS 5
- Lifecycle and target TRL
  - Project lifecycle and quality requirements based on tailored ECSS-E-ST-40C and ECSS-Q-ST-80C
  - Target criticality C and TRL6

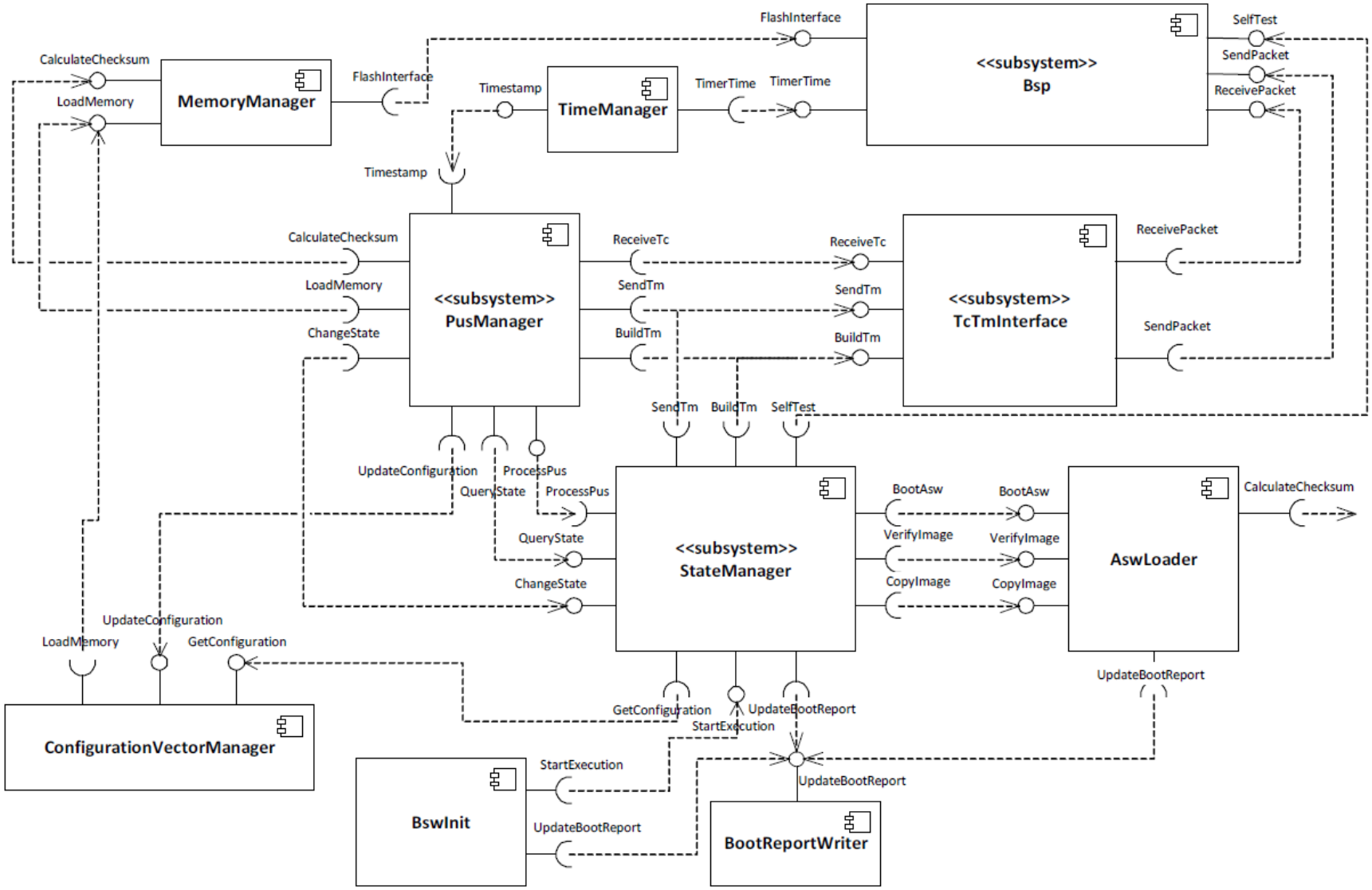


MICROCHIP

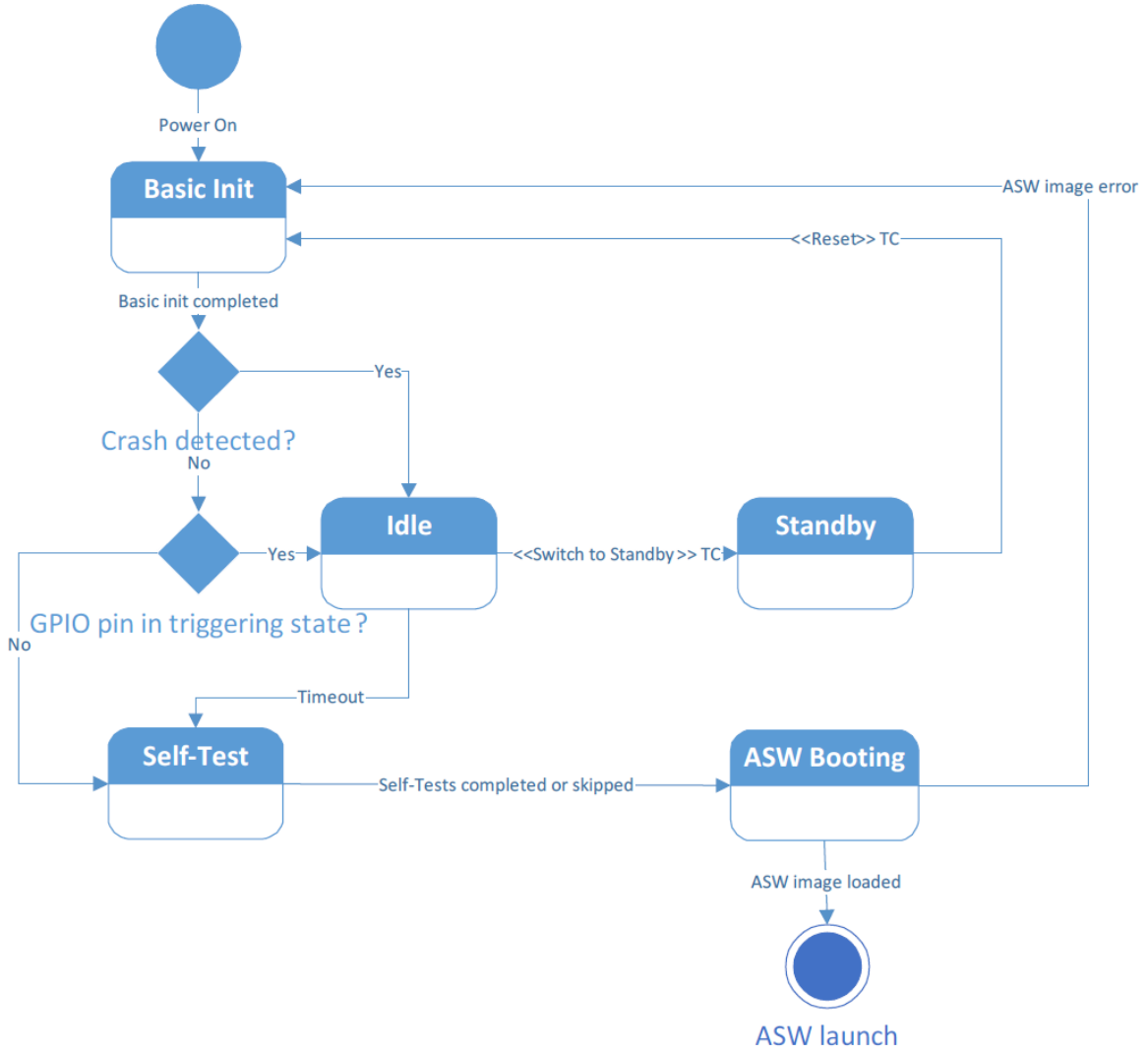
# Bootloader Software

- Bootloader software for Cortex-M7 SAMV71
- Software requirements specification based on SAVOIR Flight Computer Initialisation Sequence Generic Specification provided by ESA
- Major characteristics
  - Model based PUS-C TC/TM stack developed using ASN.1/ACN modelling supported by ESA tool ASN1SCC
  - Execution from internal Flash memory
  - Self-test of the internal SRAM and external SDRAM memories
  - Failure reporting through boot and death reports
  - Bare metal design (no RTOS used)
  - Utilizes a minimal set of BSP drivers developed in the project scope
  - Supported PUS (1, 5, 6, 8, 17)
    - Additional custom PUS 6 subservice for flash memory operations

# BSW architecture overview

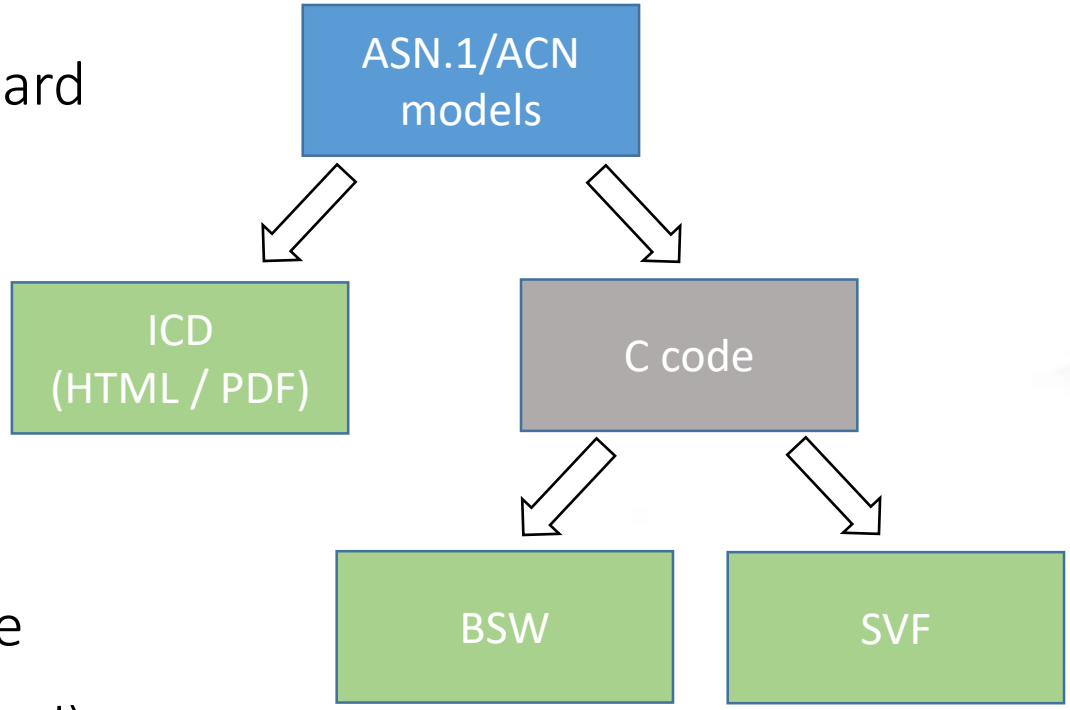


# BSW states overview



# ASN.1 data modelling

- ASN.1 – well established data modelling standard
- ACN – language for describing encoding rules, created by ESA
- ESA TASTE toolchain used to generate code and documentation
- *Single source of truth* – models ensure consistency between documentation and code
- Models distributed as part of ICD (can be reused)
- PUS services data models reused from components library




ASN.1 ACN

taste
The Assert Set of Tools for Engineering

# ASN.1 data modelling

```
PUS-6-2 DEFINITIONS AUTOMATIC TAGS ::= BEGIN
EXPORTS ALL;
IMPORTS
    BytePointer FROM RawMemory
    MemoryData FROM Data;

TC-6-2-LoadRawMemoryDataAreas ::= SEQUENCE
{
    dataAreas TC-6-2-DataArea
}

TC-6-2-DataArea ::= SEQUENCE
{
    startAddress BytePointer,
    dataToLoad MemoryData
}

END
```

SVF

```
def loadMemoryWithData(self, address, data):
    tcLoad = aut.asn.TC_PUS_6_2_LoadMemoryAbsolute()
    tcLoad.targetAddress.Set(address)
    tcLoad.data.contents.SetLength(len(data))
    for idx, datum in enumerate(data):
        tcLoad.data.contents[idx].Set(datum)

    self.sendTelecommand(tcLoad, ackExec=True)
    self.expectTelemetry(expectedTmTypes=[aut.asn.TM_PUS_1_7_ExecSuccess], timeout=60)
```

C code

```
const TC_6_2_LoadRawMemoryDataAreas* const data = &(tc->packetDataField.data.u.tc_6_2);
uint8_t* const targetAddress = data->dataAreas.startAddress;
const uint8_t* const memoryContent = data->dataAreas.dataToLoad.data.arr;
const size_t dataLength = (size_t)data->dataAreas.dataToLoad.data.nCount;
```

ICD

**TC-6-2-LoadRawMemoryDataAreas** (SEQUENCE) ASN.1 ACN — Min: 7 bytes Max: 1031 bytes

| No | Field | Comment | Present | Type | Constraint | Min Bits | Max Bits |
|---|---|---|---|---|---|---|---|
| 1 | n | | always | NULL | N.A. | 8 | 8 |
| 2 | dataAreas | | always | TC-6-2-DataArea | N.A. | 48 | 8240 |

**TC-6-2-DataArea** (SEQUENCE) ASN.1 ACN — Min: 6 bytes Max: 1030 bytes

| No | Field | Comment | Present | Type | Constraint | Min Bits | Max Bits |
|---|---|---|---|---|---|---|---|
| 1 | startAddress | | always | BytePointer | N.A. | 32 | 32 |
| 2 | dataToLoad | | always | MemoryData | N.A. | 16 | 8208 |

# BSP and CANopen library

- Bare metal driver library with support for following peripherals
  - Serial interfaces:
    - Ethernet, I2C, SPI, CAN, UART, ISI, QSPI
  - Other modules:
    - SDRAMC, WDT, RSWDT, LPOW, NVIC, SYSTICK, XDMAC, TIC, PWM, RTC, RTT, PIO, AFEC, FPU, EEFC, PMC, RSTC, SUPC
- Integration of the selected drivers with the RTEMS 5
  - RTEMS clock Driver Shell (SysTick)
  - RTEMS TCP/IP Driver (Ethernet)
  - RTEMS RTC device (Timer)
  - RTEMS I/O Manager (MCAN)
- CANopen library
  - ECSS-E-ST-50-15C, clauses: 9 (Minimal implementation), 7 (Time distribution), 8 (Redundancy management)
  - Master and slave modes
  - PDO data transfer: unconfirmed command, telemetry request, SYNC

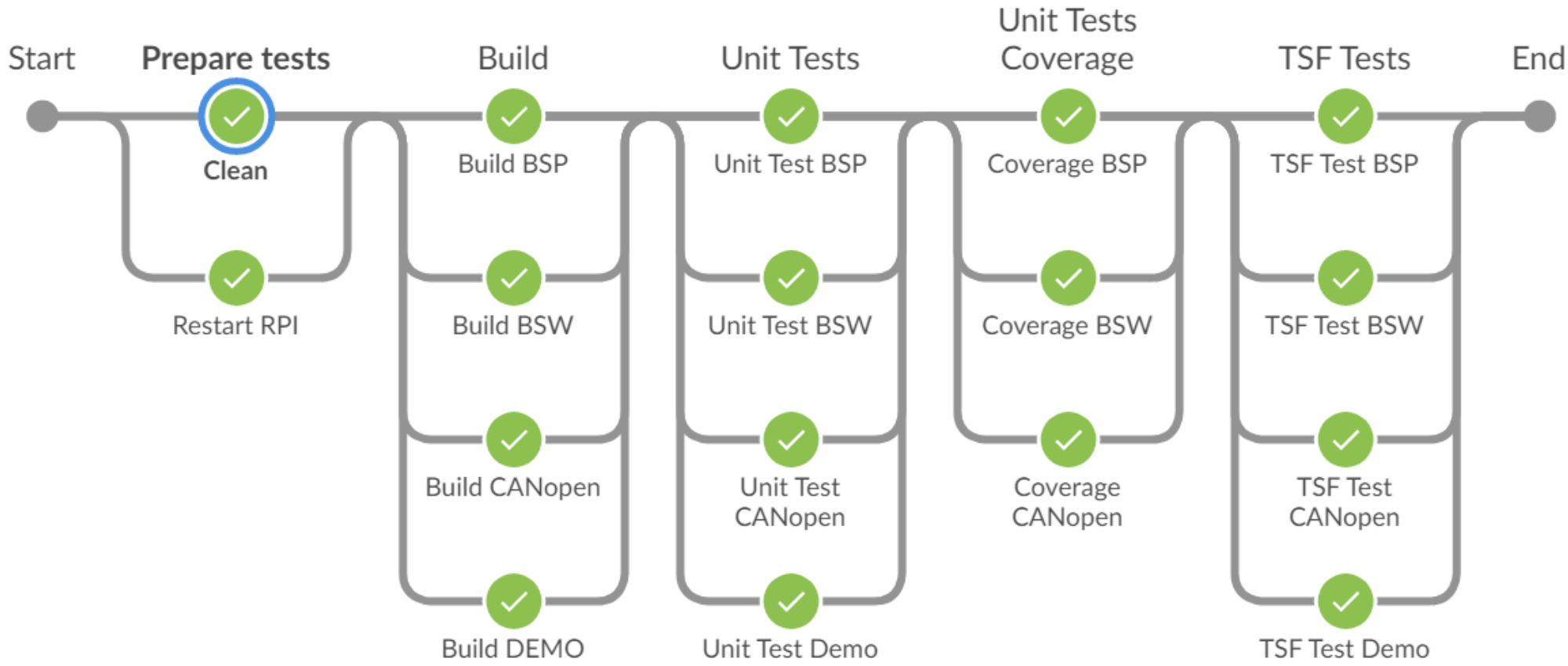
# Tailoring of the CANopen library

- Implementation followed the following clauses of the ECSS-E-ST-50-15C standard:
  - Clause 7 – Time distribution
  - Clause 8 – Redundancy management
  - Clause 9 – Minimal implementation of CANopen protocol for highly asymmetrical control applications
- Main elements left out from the implementation:
  - SDOs (*Service Data Objects*)
  - Support for remote transmission request (RTR)
  - Support for setting remote SCET time
  - Implementation of an EDS-to-OD (Electronic Data Sheet to Object Dictionary) converter

# Testing & qualification approach

- Test environment
  - Controlled by CI Jenkins server
  - Unit tests implemented in open-source Cmocka
    - Achieved >80% code and branch coverage
  - Code coverage analysed with ported gcov
  - Static analysis
    - MISRA compliance with Cppcheck
    - Code metrics with Lizard
    - clang-tidy, clang-format
  - Traceability matrixes generation based on Doxygen comments
  - Integration and validation supported by Python scripting environment responsible for C&C communication
  - PEAK dongle and CANfestival used for CANopen validation
  - BSP integrated into Microchip web server demo

# CI test stages



# Static analysis

- Checks performed as part of build steps
- Validated on CI
- Zero tolerance (all warnings are errors)
  - failing checks block next build steps
- Checks include:
  - cppcheck – static analysis and MISRA rules verification
  - clang-tidy – static analysis
  - clang-format – code style conformance
  - lizard – complexity analysis
- Project was early adopter of cppcheck MISRA addon
  - A few bug fixes were submitted to cppcheck

# Test Suite Framework

- In-house built testing framework
- With core being platform-independent, framework was adapted for use with ATSAMV71Q21
- Main features:
  - Based on Python test scripts
  - Easy integration with different hardware platforms and setups
  - Automatic collection of debugger and application outputs
  - Enables construction of platform-agnostic, portable test scenarios

# Test cases definition

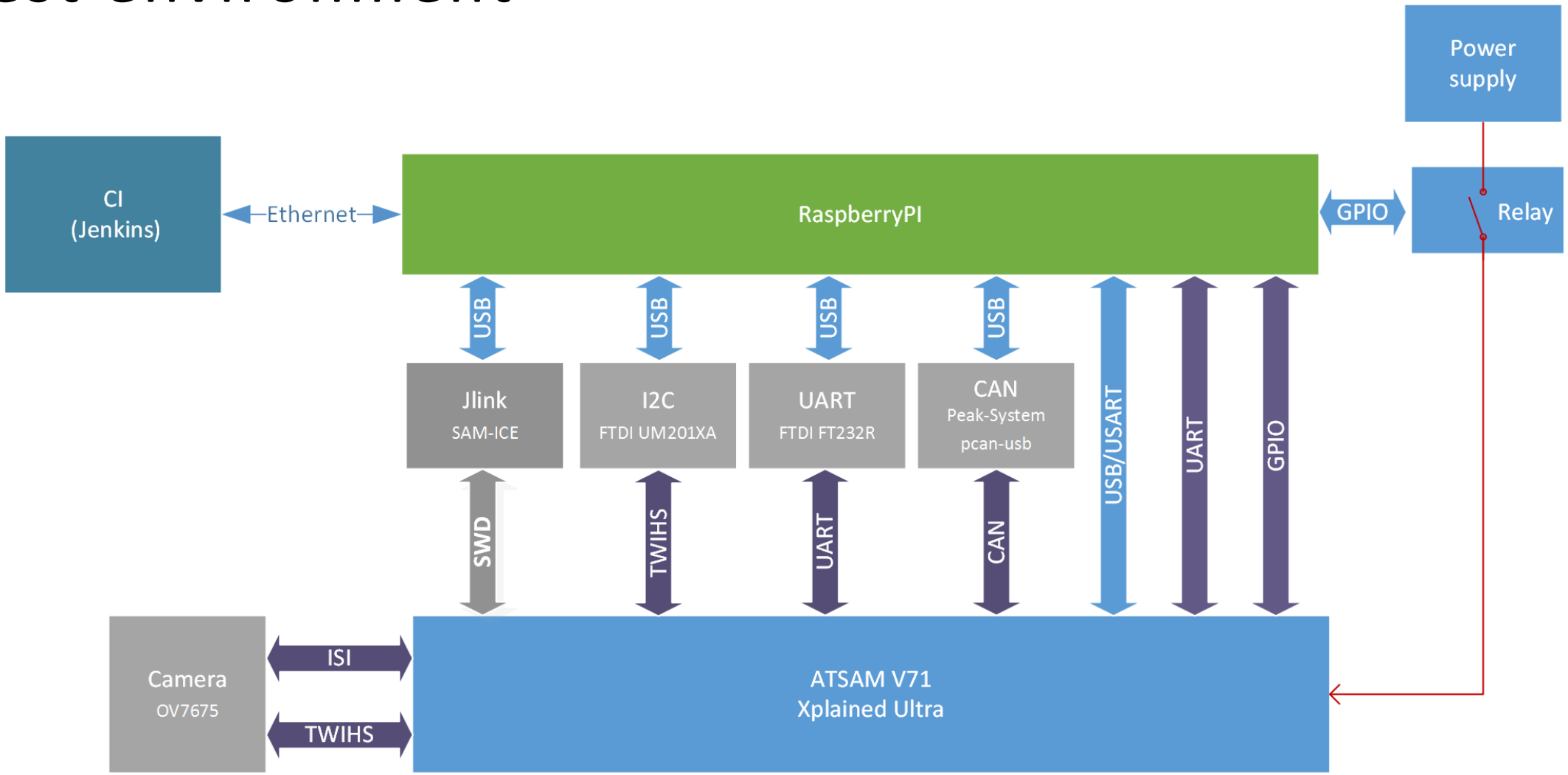
- Allows instrumentalization and inspection of communication over the C&C link

```
12  ## @Context \li BSW image compiled.
13  #           \li C&C UART link connected.
14  class Pus17Test(aut.TestSuite):
15
16      ## \Given BSW in Idle state
17      #  \When TC[17,1] is sent
18      #  \Then it is answered with TM[17,2].
19      #  @SRS SRS-BSW-IF-330
20      def test_applicationRespondsToPus17Request(self):
21          self.awaitIdleModeEntry()
22
23          self.pusInterface.sendTelecommand(
24              aut.asn.TC_17_1_PerformAnAreYouAliveConnectionTest(), timeout=15)
25
26          msg = self.pusInterface.expectTelemetry(
27              expectedTmTypes=[aut.asn.TM_17_2_AreYouAliveConnectionTestReport], timeout=15)
28
29          self.assertTmDataEqual(aut.asn.TM_17_2_AreYouAliveConnectionTestReport(), msg)
```

- For the purpose of the ARMBSP project, enables instrumentation of various microprocessor peripherals

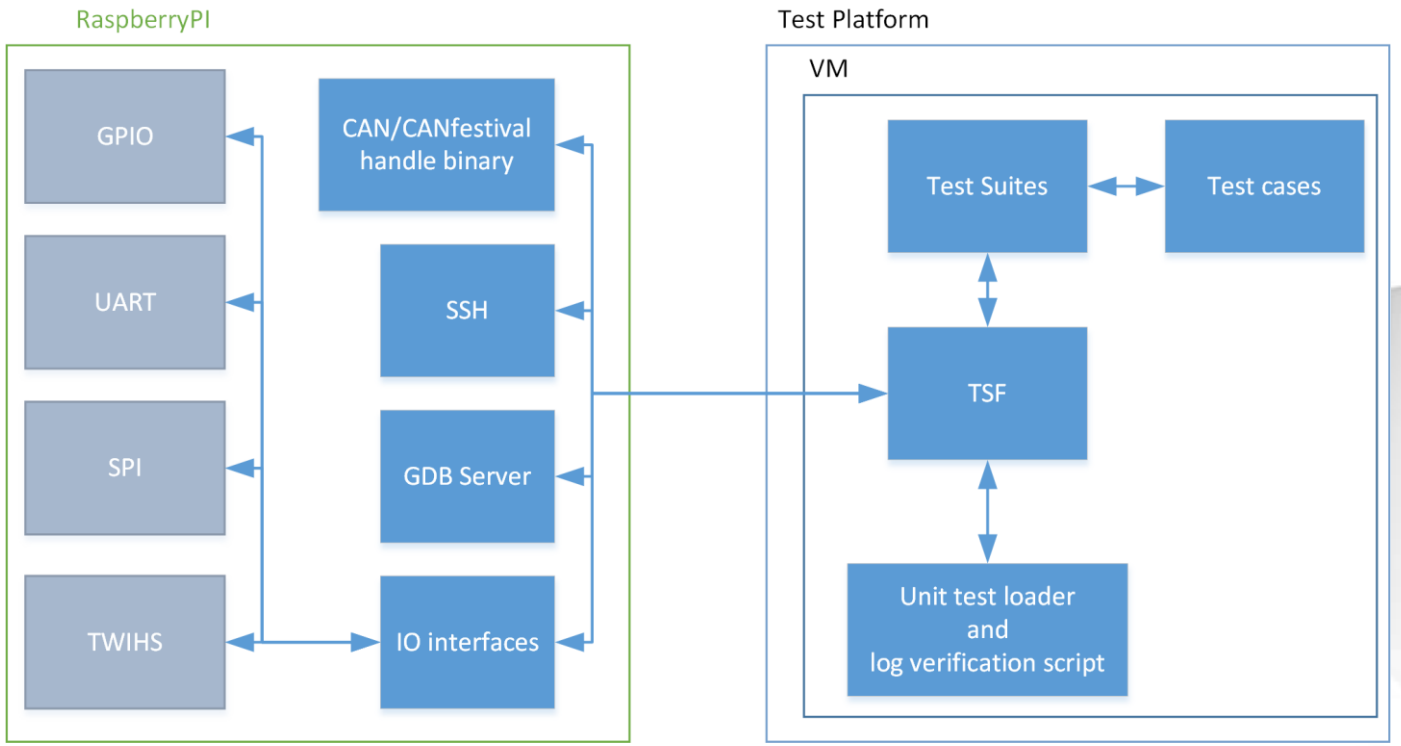
```
82      def test_PioControlsInterrupts(self):
83          self.egse.setEgsePin(self.PIO_OUT_PIN, high=False)
84          time.sleep(self.REMOTE_PIN_SETTING_DELAY)
85          self.executeAnArmockaCommand("disableInterrupt")
86          self.flushUartBuffers()
87
88          # Trigger an interrupt.
89          self.egse.setEgsePin(self.PIO_OUT_PIN, high=True)
90          time.sleep(self.REMOTE_PIN_SETTING_DELAY)
91          self.egse.setEgsePin(self.PIO_OUT_PIN, high=False)
92          time.sleep(self.REMOTE_PIN_SETTING_DELAY)
93
94          response = self.executeAnArmockaCommand("readIrqStatus")
95          self.ensureArmockaProgramRespondsWithString(self.STATUS_LOW_STRING, response)
96
97          self.executeAnArmockaCommand("enableInterrupt")
98          self.flushUartBuffers()
99
100         # Trigger an interrupt.
101         self.egse.setEgsePin(self.PIO_OUT_PIN, high=True)
102         time.sleep(self.REMOTE_PIN_SETTING_DELAY)
103         self.egse.setEgsePin(self.PIO_OUT_PIN, high=False)
104         time.sleep(self.REMOTE_PIN_SETTING_DELAY)
105
106         response = self.executeAnArmockaCommand("readIrqStatus")
107         self.ensureArmockaProgramRespondsWithString(self.STATUS_HIGH_STRING, response)
```

# Test environment



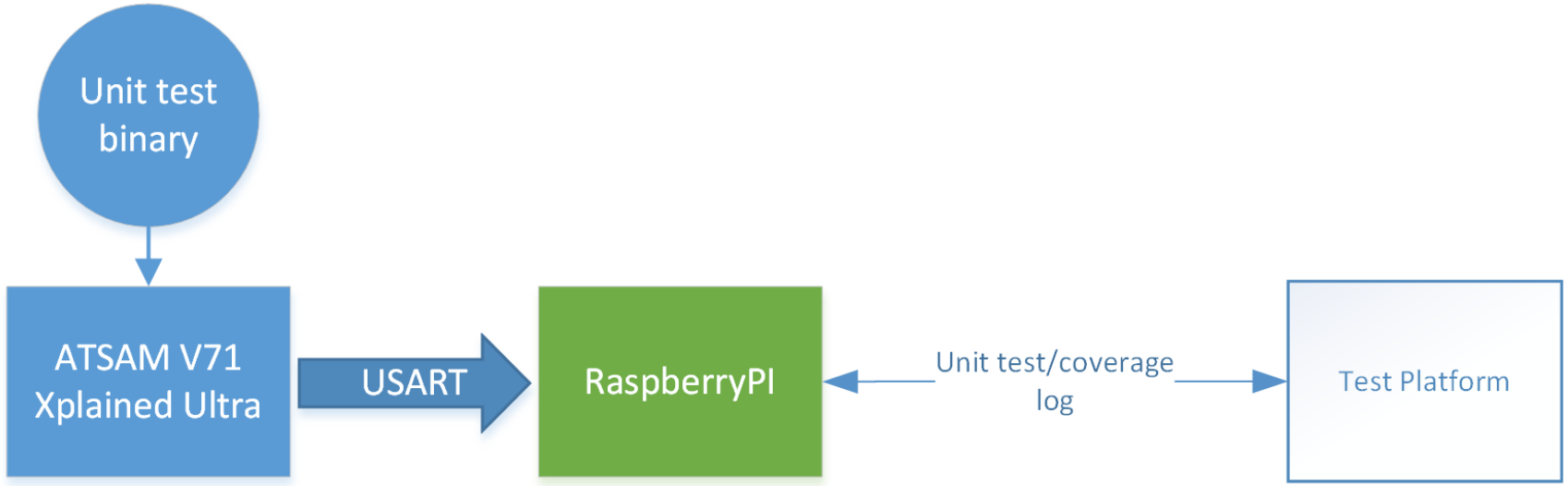
# Test environment

- RaspberryPI used as an interface to the ATSAM V71
  - Remote IO control
  - RaspberryPI configuration with ssh
  - GDB Server host
  - Environment for running binaries to handle CAN bus and CANfestival
- Test Platform
  - Interface to the RaspberryPI with TSF (Test Suite Framework)
  - Unit test log verification
  - Several Test Suites to handle integration testing



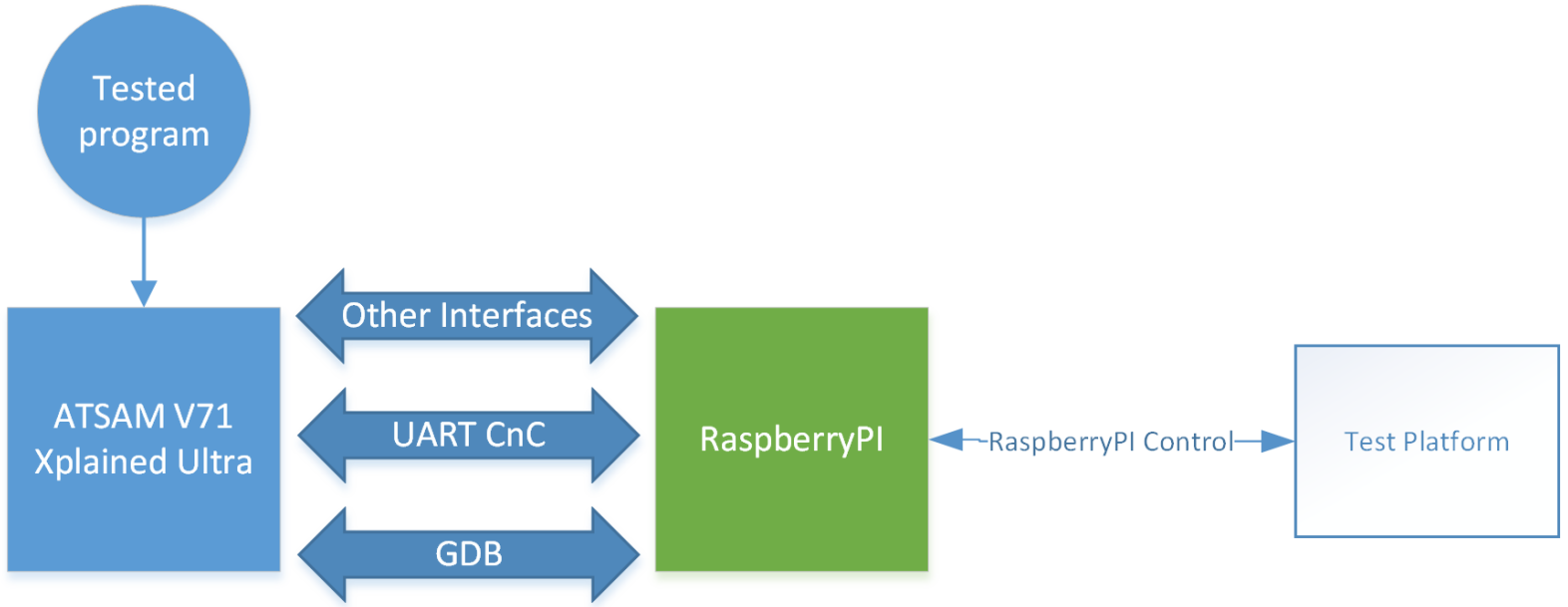
# Unit testing

- Cmocka unit test framework
- Each module has it’s own binary with test cases
- Checking done on the MCU with log sent using USART
- Verification of the results done by the Test Platform



# Testing performed by Test Suite Framework

- Direct program execution control with python scripts.
  - GDB interface
  - C&C interface communication with UART
  - CAN bus communication using CanFestival and basic CAN framework



# CANopen performance measurement scenarios

- We performed test measurements of performance of the library in three scenarios:
  - Active waiting – transmit a single message or a 16-message burst and wait until the hardware queue is empty before queuing more messages;
  - Event-based transmission – transmit the messages upon reception of a system event generated in the transmission interrupt handler;
  - Active queue filling – variation of active waiting; poll the hardware queue status and transmit the messages whenever there's space in the queue.
- Measurements were performed with an RTEMS-based demo application, with the processor clock running at 150MHz and bus baudrate of 1MBit/s
- Data reception performed by PEAK dongle and CANfestival controlled by RPi

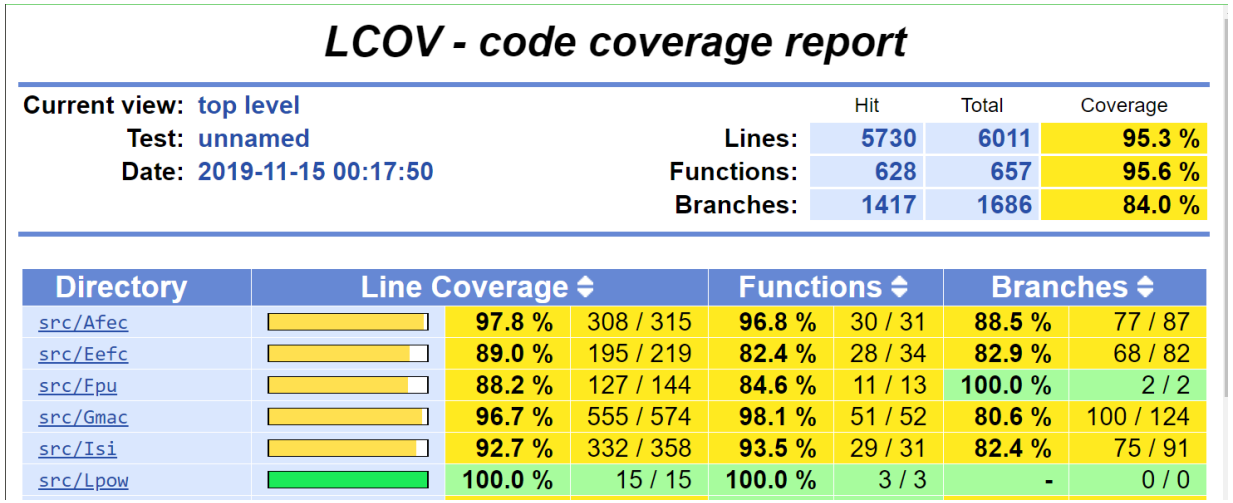
# CANopen performance measurement results

- Performed by triggering 10000 queueing operations (giving 10000 messages for single message transmissions and 160000 messages for bursts).
- With baudrate of 1Mbit/s, average user data rate is ~530kbit/s.

| | Active waiting | | Event-based transmission | | Active queue filling |
|---|---|---|---|---|---|
| | Single message | 16-message burst | Single message | 16-message burst | |
| Data bandwidth usage | 69.4% | 93.6% | 62.9% | 92.9% | 95.79% |
| CPU load from CANopen library | 29.8% | 34.7% | 30.5% | 33.2% | 34.3% |

# GCOV/LCOV

- Adapted with coverage stubs
  - Linked custom _read, _write, _open, _close etc. functions
  - _write forwards the coverage data to the USART
- Standard *gcc --coverage* binary compilation.
- Output captured by the EGSE and saved to the files for further analysis with GCOV/LCOV



**LCOV - code coverage report**

Current view: top level
Test: unnamed
Date: 2019-11-15 00:17:50

| | Hit | Total | Coverage |
|---|---|---|---|
| Lines: | 5730 | 6011 | 95.3 % |
| Functions: | 628 | 657 | 95.6 % |
| Branches: | 1417 | 1686 | 84.0 % |

| Directory | Line Coverage | | Functions | | Branches | |
|---|---|---|---|---|---|---|
| src/Afec | 97.8 % | 308 / 315 | 96.8 % | 30 / 31 | 88.5 % | 77 / 87 |
| src/Eefc | 89.0 % | 195 / 219 | 82.4 % | 28 / 34 | 82.9 % | 68 / 82 |
| src/Fpu | 88.2 % | 127 / 144 | 84.6 % | 11 / 13 | 100.0 % | 2 / 2 |
| src/Gmac | 96.7 % | 555 / 574 | 98.1 % | 51 / 52 | 80.6 % | 100 / 124 |
| src/Isi | 92.7 % | 332 / 358 | 93.5 % | 29 / 31 | 82.4 % | 75 / 91 |
| src/Lpow | 100.0 % | 15 / 15 | 100.0 % | 3 / 3 | - | 0 / 0 |

```
<><> performing unit tests.
>> UNIT TEST RESULT - BEGIN <<
  <testsuite name="WdtTests" time="0.000" tests="8" failures="0"
    <testcase name="Fpu_hasCorrectFeatures" time="0.000" >
    </testcase>
    <testcase name="Fpu_correctlySetsAndGetsConfig"     <testcas
    </testcase>
    <testcase name="Fpu_correctlyHandlesFlushToZero" time="0.000
    </testcase>
    <testcase name="Fpu_correctlyDetectsErrors" time="0.000" >
    </testcase>
  </testsuite>
</testsuites>
>> UNIT TEST RESULT
>>>/home/arm/dev/armmcu/BSP/build/coverage/src/Fpu/Fpu.gcda
0600000002000000000000000C000000000000000100000003000000000000000
>>>/home/arm/dev/armmc6164636772323741A67ED2B6000000A37A000000CE
0600000002000000000000000C000000000000000100000003000000000000000
>>>/home/arm/dev/armmcu/BSP/build/coverage/src/Pmc/Pmc.gcda
0600000002000000000000000C000000000000000100000003000000000000000
>>>/home/arm/dev/armmcu/BSP/build/coverage/src/Nvic/Nvic.gcda
0600000002000000000000000C000000000000000100000003000000000000000
>>>/home/arm/dev/armmcu/BSP/build/coverage/src/Startup/startup_s
0600000002000000000000000C000000000000000100000003000000000000000
0600000002000000000000000C000000000000000100000003000000000000000
>>>/home/arm/dev/armmcu/BSP/build/coverage/src/Fpu/tests/FpuTest
0600000002000000000000000C000000000000000100000003000000000000000
>> COVERAGE RESULT - END <<
```

# MC/DC coverage

- C language introduces “branching” in all complex conditions
  (“short circuit” in **&&** and **||**)
- LCOV measures branch coverage using branches from assembly, not from C code
- In result branch coverage calculated by LCOV is *almost* equivalent to modified condition/decision coverage
- Difference lays in “Boolean vectors” checks (including bit fields)

```
        [ +  + ]:          5 :   if (MemoryManager_isBusy(memoryManager))
                 :          1 :     return returnError(errCode, MemoryManager_ErrorCode_Busy);
[ +  + ][ +  + ]:          4 :   if (!isAligned(destination) || !isAligned(source))
                 :          2 :     return returnError(errCode, MemoryManager_ErrorCode_MemoryNotAligned);
        [ +  + ]:          2 :   if (size > MemoryManager_BlockMaxSize)
                 :          1 :     return returnError(errCode, MemoryManager_ErrorCode_BlockTooBig);
```

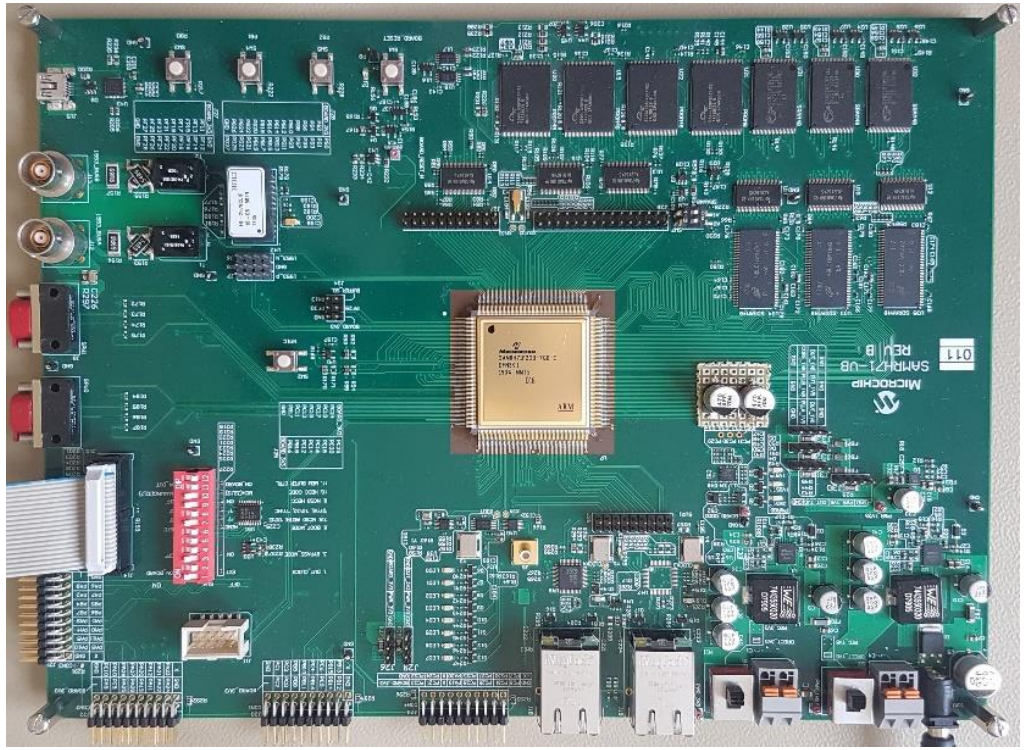
# Test summary

- Code & branch coverage computed from unit tests execution

| | BSP | BSW | CANopen |
|---|---|---|---|
| Line coverage | 5730/6011 (95.3%) | 6738/7052 (95.5%) | 3999/4207 (95.1%) |
| Branch coverage | 1417/1686 (84%) | 1439/1700 (84.3%) | 1095/1290 (84.9%) |
| Unit test cases | 476 | 682 | 385 |
| Integration test cases | 39 | 62 | 12 |

# Conclusion and future

- Reusable software suite for Cortex-M7 processor line from Microchip
  - BSW, BSP
  - CANopen library
- Automatic test environment based on dev kit and RaspberryPi ensuring external access to interfaces
- BSW was successfully integrated with Microchip web server demo
- Future steps
  - Ongoing adaptation of BSW and BSP to SAMRH71 and future ARM MCU
    - Support for SpaceWire and IO Switch Matrix
    - Remote application booting through SPI and RMAP
  - Usage foreseen in future ICECube project for ISS
  - Need for criticality B qualification



# Thank you for your attention

Michał Mosdorf
mmosdorf@n7space.com

Michał Kocon
mkocon@n7space.com

+48 22 299 20 50
www.n7space.com