N7 SPACE

CANopen library demonstration on SAMV71 MCU

Michał Mosdorf

Marcin Dzieżyc

CAN in Space

2019

N7 SPACE

- Joined venture between SPACEBEL and N7 Mobile located Poland
  - Software company founded in 2017
  - Started operation with projects previously executed by N7 Mobile that were transferred to N7 Space (e.g. PROBA3 SW)
  - 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/ACN, SDL, MSC, Capella, TASTE

Selected activities

- PROBA3 ASPIICS on-board software
  - CBK's subcontractor in PROBA3 mission responsible for on-board software
  - Responsibility for complete ECSS based software development lifecycle for GR712 boot SW, application SW based on RTEMS and Python based SVF
- ARM CoreSight usage in space applications
  - Prime contractor in PLIIS funded R&D activity focused on ARM CoreSight usage for multi-core software tracing and inter core interreference analysis on ARM Cortex-A53
- MBSE activities
  - ASN.1/ACN used for code generation in PUS TC/TM stacks used embedded projects
  - ASN.1 IDE based on Qt Creator with PUS-C library allowing tailoring for future missions
  - Plugins between Capella system level and TASTE environment to allow SDL modelling and code generation for Cortex-M4F (joined activity with Creotech Instruments)

©ESA–P. Carril

ASN.1 ACN

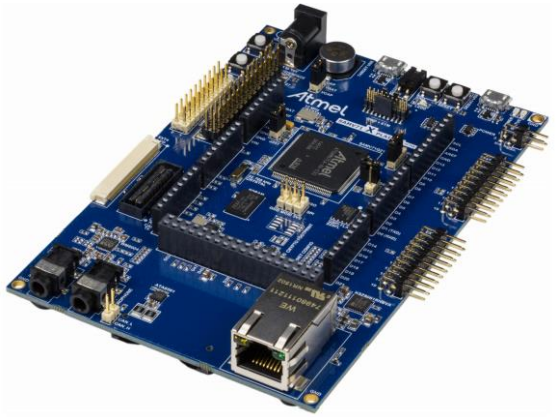
taste

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

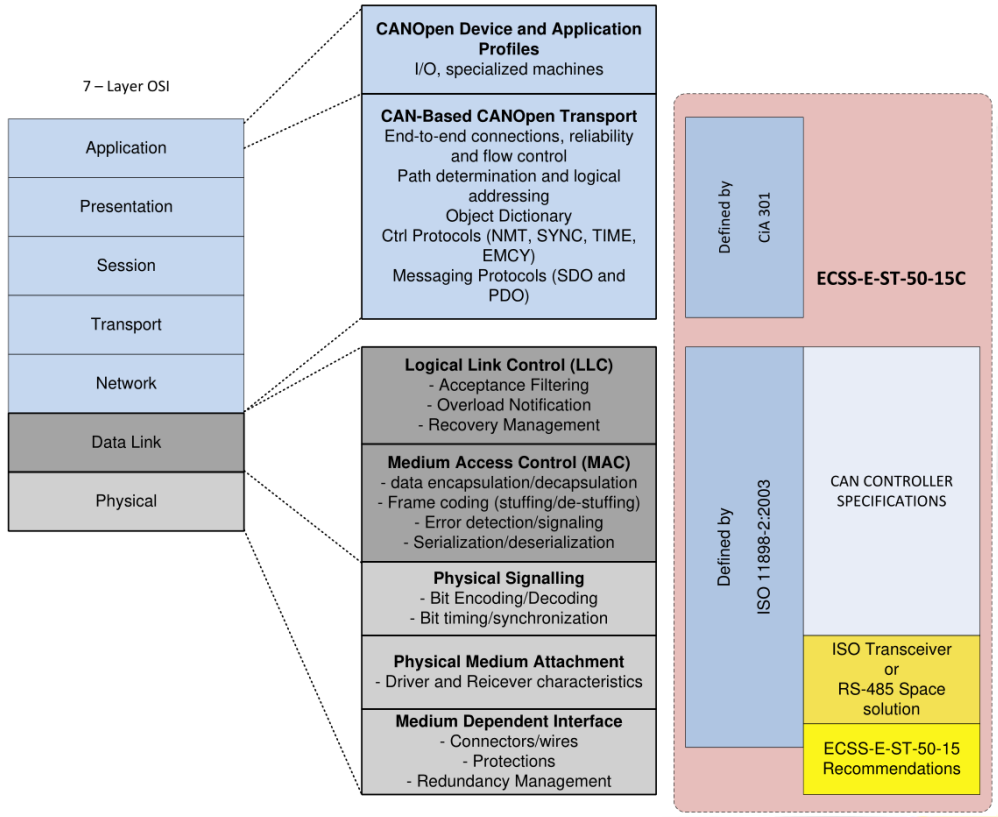
- Bootloader software for Cortex-M7 SAMV71
- Software requirements specification based on SAVOIR Flight Computer Initialisation Sequence Generic Specification provided by ESA
- Major characteristics
  - Model based PUS-C TC/TM stack developed using ASN.1/ACN modelling supported by ESA tool ASN1SCC
  - 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

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

The ECSS-E-ST-50-15C standard

- CANbus extension protocol
- Proposes a structure and implementation of specific network layers in a CAN-based network
- Includes CANopen as a protocol on top of the physical layer
- Describes key elements of the protocol, including time distribution, redundancy management and minimal implementation of the object dictionary
- Based on the CAN in Automation 301 document

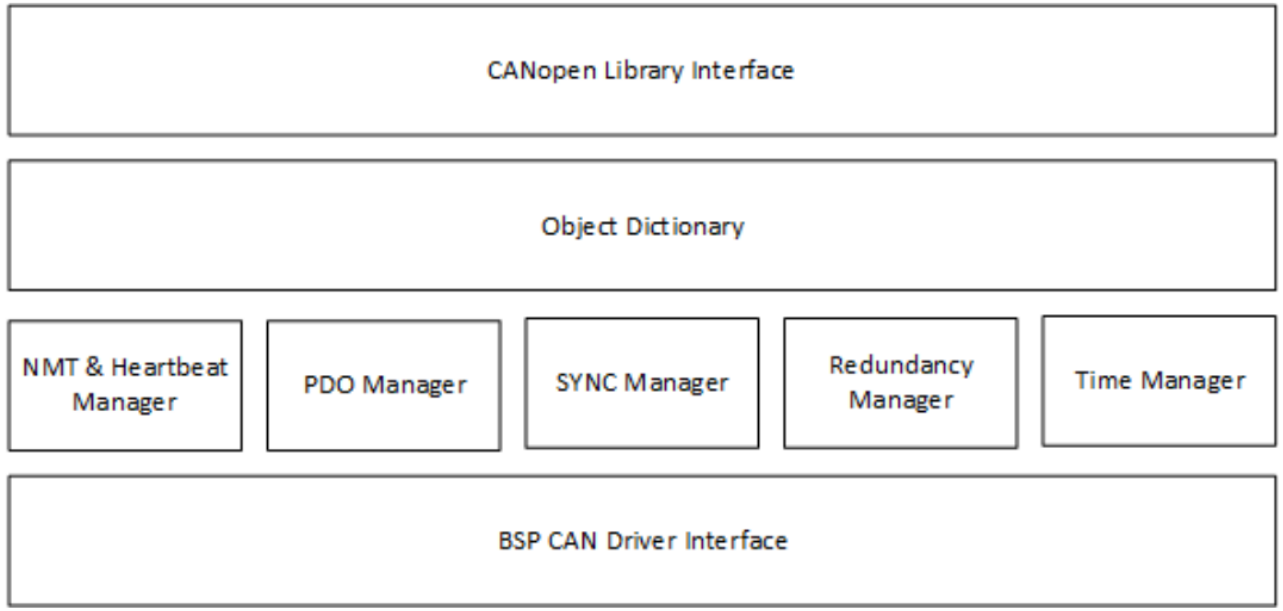


Tailoring of the CANopen library

- 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

Architecture and implementation

- Architecture and implementation
  - A number of components implementing specific services described in the standard
  - Sets of components then matched together to produce role-specific libraries



Object Dictionary

- Structure of the object dictionary
  - A dictionary of objects indexed by 16-bit integers
  - Each entry can be a record, further storing values under 8-bit subindices
  - Many different data types for objects: from boolean, through 8-bit to 64-bit integers, to Unicode strings
  - Certain address ranges are standardized
    - 0x1000 - 0x1FFF – Communication object area
    - 0x2000 - 0x5FFF – Manufacturer specific area
    - 0x6000 - 0x9FFF – Device profile specific area
    - 0xA000 - 0xBFFF – Interface profile specific area
  - The structure of the dictionary can't be changed at runtime

Index	SubIndex	Example Value	Description
0x1801	0x00	100	Highest subindex
	0x01	0x0000161	PDO COB ID
	0x02	0	
	0x03	1000	Inhibit time
	0x04	Unused	Unused
	0x05	0	Event timer
	[...]	[...]	[...]
0x1A01	0x00	3	Number of entries
	0x01	Index 0x2001, subindex 0x00	Mapped OD item 1
	0x02	Index 0x2001, subindex 0x03	Mapped OD item 2
	0x03	Index 0x6101, subindex 0x00	Mapped OD item 3

Object Dictionary

- Dictionary is defined in the form of a header file included in the application
  - Contains the definitions of the underlying objects and the dictionary structure
- User application can register custom object write handlers for each index

```
static uint32_t syncCobId = 0x00000000;
static uint32_t syncPeriod = 0x00000000;

static ObjectDictionary_Entry syncCobIdEntry[] = { { .subindex = 0x00,
  .dataType = ObjectDictionary_DataType_Unsigned32,
  .access = ObjectDictionary_Access_Const,
  .isMappable = false,
  .valueSize = bitsizeof(syncCobId),
  .value = &syncCobId } };

static ObjectDictionary_Entry syncPeriodEntry[] = { { .subindex = 0x00,
  .dataType = ObjectDictionary_DataType_Unsigned32,
  .access = ObjectDictionary_Access_RW,
  .isMappable = false,
  .valueSize = bitsizeof(syncPeriod),
  .value = &syncPeriod } };
```

Network Management & Redundancy

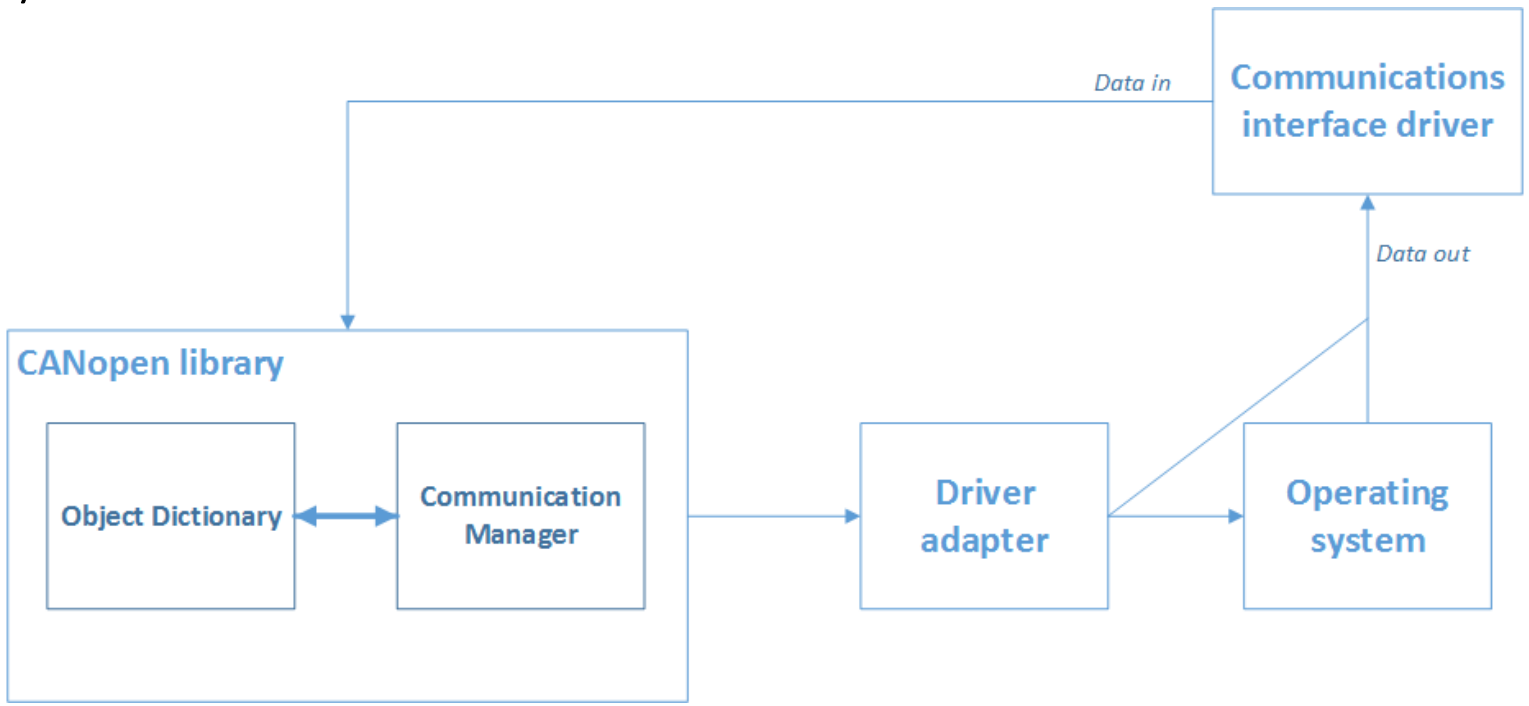
- Heartbeat messages sent periodically to verify status of node/bus
- Messages contain current status of node
  - Heartbeat handlers in the application can be used to guide the initialization process
- Library accommodates two driver adapters, for nominal and redundant bus connections
  - Both can refer to a single interface in case of no redundancy
- Bus switching behaviour configuration based on special values under index 0x2000:
  - Bdefault – default bus selector
  - Ttoggle – time before switching bus
  - Ntoggle – number of switches before giving up

Customizability

- Certain service managers (NMT, SYNC, Time) and the Object Dictionary allow registering custom reception event handlers
  - Synchronization service can be used to invoke periodical behaviour in nodes and schedule periodical updates of contents of specific objects
  - This enables development of fully event-driven applications
- In combination with small library size and independence from operating systems, our implementation can be fit into small MCU-driven devices, e.g. measurement probes

Portability

- Implementation designed to remain independent from processor architecture, operating system and communication interfaces
- API allowing usage of any character-based communication medium



Portability

- Custom driver adapters can be used to further integrate the library with an underlying operating system (e.g. RTEMS)
- The library offers a data entry procedure, which can be directly incorporated in interrupt handlers

```
static bool write(const uint16_t canId,
  void* const data,
  const size_t dataSize,
  void* arg,
  int* const errCode)
{
  (void)arg;

  McanRtems_DataBuffer txBuffer;
  txBuffer.id = canId;
  txBuffer.length = (uint8_t)dataSize;
  memcpy(txBuffer.data, data, dataSize);
  *errCode = rtems_io_write(mcan, ACTIVE_MCAN_MINOR, &txBuffer);
  return *errCode == RTEMS_SUCCESSFUL;
}
```

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
  - 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

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

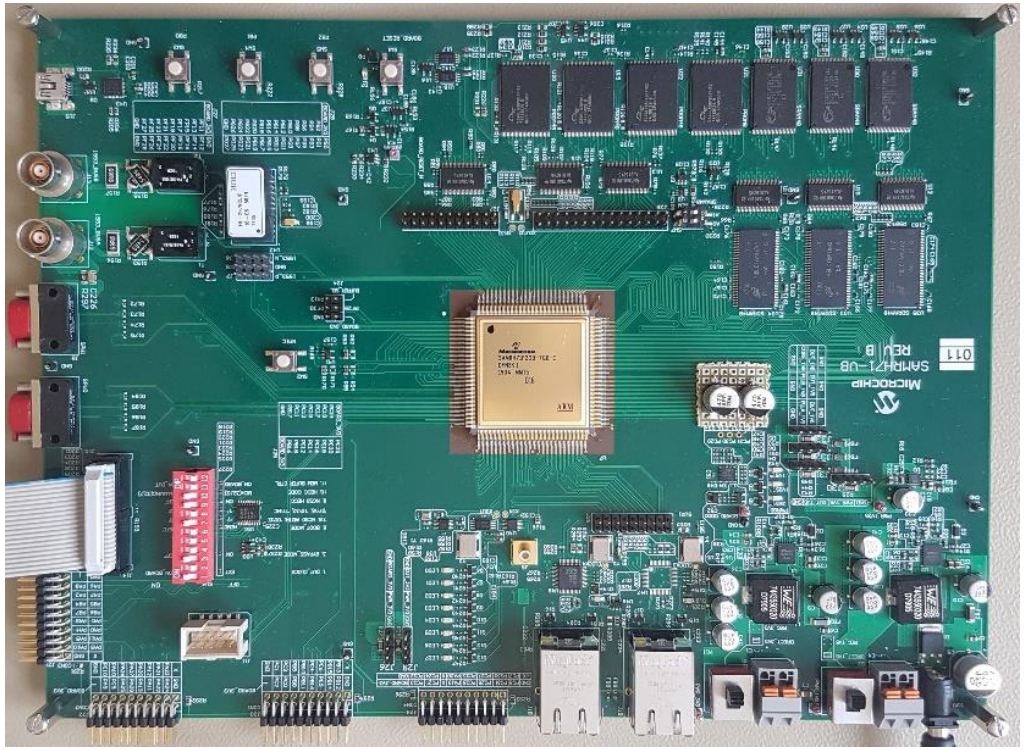
Performance measurement results

- 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%

Conclusion and future

- Reusable software suite for Cortex-M7 processor line from Microchip
  - Boot software
  - Board Support Package
- Portable CANopen library
- Future steps
  - CAN FD support
  - Eagle Eye integration
  - BSW & BSP adaptation to the RH71
    - Support for SpaceWire and IO Switch Matrix
    - Remote application booting through SPI and RMAP
    - FreeRTOS integration
  - Criticality B qualification



Thank you for your attention

Michał Mosdorf
mmosdorf@n7space.com

Marcin Dzieżyc
mdziezyc@n7space.com

+48 22 299 20 50
www.n7space.com