Full verification of an ECSS-E-ST-50-15C implementation based on an open-source CANopen software stack

Karl-Emil Sandvik Bohne – Gianluca Furano

12/06/2019

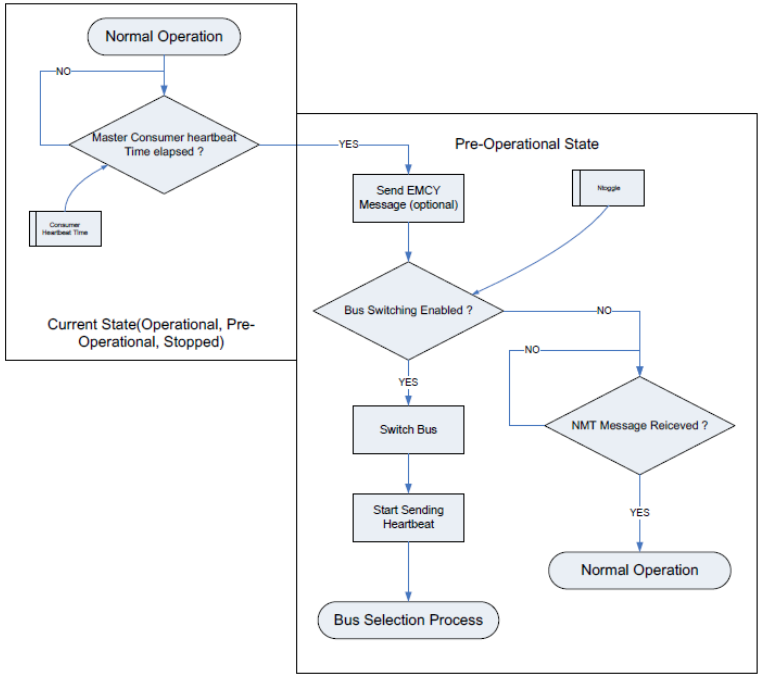
Background

- The use of CAN on uC likely to increase in coming years
  - Power consumption, design simplicity, PCB estate, decentralization of functions play a role in this
- Solid industry backing; automotive, critical embedded control industry already make heavy use of CAN
- An ESA activity created out of these observations is one with the goal of developing a software implementation of the ECSS-E-ST-50-15C standard
  - Configurability, reusability, small codebase, memory use and processing load emphasised
  - Focus on suitability for resource-constrained space-grade microcontrollers

CANopen

ECSS-E-ST-50-15C

- Places demands and restrictions on the physical aspects of CAN
- Specifies required CANopen functionality
- Extends CANopen with time distribution and redundancy management mechanisms
  - To an extent, these are implemented using existing CANopen functions and objects
- ECSS needs an update. Don't forget this !



ECSS-E-ST-50-15C bus monitoring logic

Open source alternatives

- Using open-source software has benefits:
  - Shortens development time, is cost-effective, community driven, and by definition shareable
  - Licensing can be an issue. Little to no support (usually)
- A few open-source, plain-C, CANopen implementations exist and were evaluated as part of the activity
  - CANFestival – oldest, mature solution
  - CANopenNode – newer, lightweight, aimed at microcontrollers
  - OpenCANopen – runs on Linux, but «...can be easily ported»
    - Does not implement many required features out-of-the-box
  - Lely CANopen – newer, extensive, feature-rich

Evaluating alternatives– features are not lacking

CANopen Feature	CANfestival	CANopenNode	Lely CANopen
NMT master / slave	✓ / ✓	✓ / ✓	✓ / ✓
NMT HB / guarding	✓ / ✓	✓ / ✗	✓ / ✓
SDO client / server	✓ / ✓	✓ / ✓	✓ / ✓
SDO exped. / segm.	✓ / ✓	✓ / ✓	✓ / ✓
PDO stat. / dyn. map	✓ / ✓	✓ / ✓	✓ / ✓
SYNC prod. / cons.	✓ / ✓	✓ / ✓	✓ / ✓
EMCY prod. / cons	✓ / ✓	✓ / ✓	✓ / ✓
Non-volatile storage support	✗	✓	✗
LSS	✓	✓*	✓

*Feature not on master branch

Evaluating alternatives

	CANFestival	CANopenNode	Lely CANopen
Features	✓	✓	✓
Stack Size	✗	✓	✗
Licensing	✓ (LGPL2)	✓ (GPL2 w/linking exception)	✓ (Apache v.2)
Portability	✓	✓	✓
Documentation	✓	✓	✓
Configurability	✓	✓	✓
Active dev. / -commu.	✓	✓	✓
Code quality	✓	✓	✓
EDS tool / -parser	✓	✓	✓

Evaluating alternatives

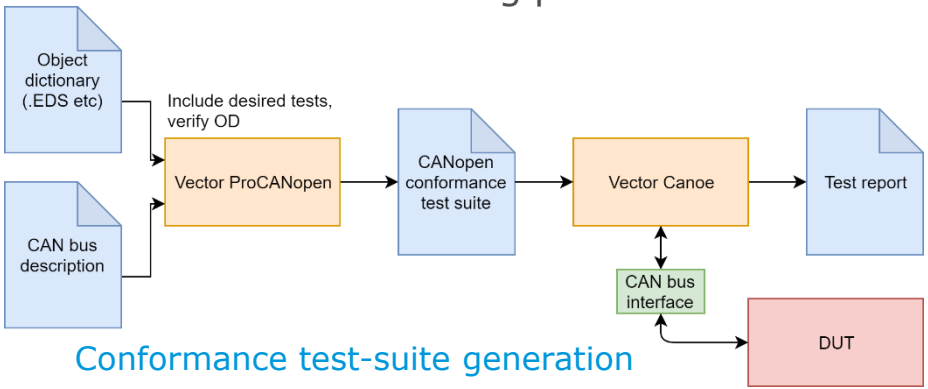
- Generally, evaluated stacks contain required functionality and where features are missing, these are not critical
- Lely CANopen:
  - High code-quality, configurable, well documented
  - Extensive
  - Compiled library quite large (>100kB)
  - Requires additional dependencies
- CANopenNode
  - Small stack size
  - Well documented, simple porting scheme, favorable license and configurability made this a suitable starting point

Porting to LEON2/3 targets

- **CANopenNode** was ported to representative OBCs and MCUs in both bare-metal and RTOS (RTEMS) configurations
  - Standard GRLIB/RTEMS device drivers used
- Codebase pruned and optimized, features deemed not necessary made optional via compile-time flags
- Deployed on **AT697 OBC** and **GR716 MCU**
  - Pruned stack fits in 15kB of RAM
  - Integrated in small CANopen network in ESTEC laboratory for testing

Conformance-testing and evaluation

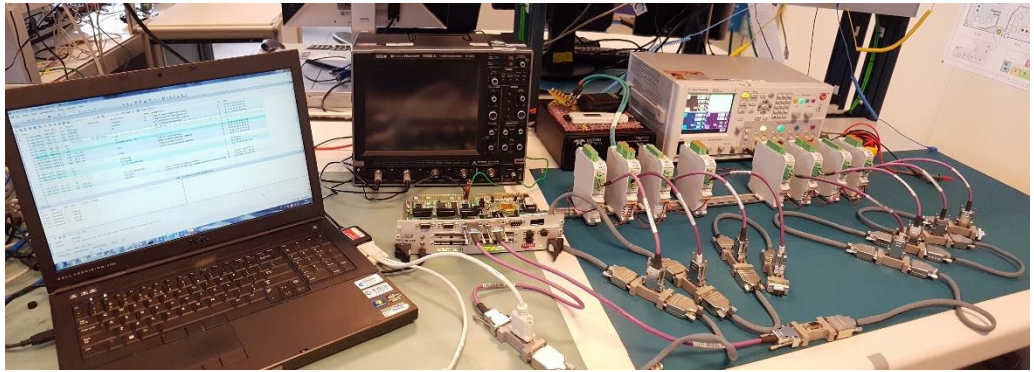
- Complete implementation of CANopen functions and their conformance not required:
  - For instance, ECSS-E-ST-50-15-C requires the heartbeat service, but not node-guarding or LSS
  - Desirable that the CANopen functionality specified in 50-15-C is compliant
- Industry-standard SW/HW tools used to verify standard compliance
  - Conformance-testing performed with the DUT placed in physical test-setup



Conformance test-suite generation

Conformance

- Output of conformance test-tools is an itemized list of test-cases and outcomes
- Out of the box, issues are found
  - Majority were rectified and will be reported to developers by activity end



From the left: Vector CANoe, GR716, CAN bus with CANopen devices

Statistics

Overall number of test cases	36	
Executed test cases	36	100% of all test cases
Not executed test cases	0	0% of all test cases
Test cases passed	36	100% of executed test cases
- Test cases passed without warning	36	100% of executed test cases
- Test cases passed with warning	0	0% of executed test cases
Test cases failed	0	0% of executed test cases

Test Case Results

Preparation of Test Module	
1 SDO-Upload for node 0x1	
1.1 Check upload from readable objects	
1.1.1 Execute: 'Check upload from readable objects'	Passed
1.1.2 [1000,0]	Passed
1.2 Check SDO object access in different NMT states	
1.2.1 Execute: 'Check SDO object access in different NMT states'	Passed
1.2.2 [1000,0]	Passed
1.3 Check if the SDO server aborts current transfer on timeout	
1.3.1 Execute: 'Check if the SDO server aborts current transfer on timeout'	Passed
1.3.2 [1008,0]	Passed
1.4 Check if the SDO Server stops transfer on receipt of an abort message	
1.4.1 Execute: 'Check if the SDO Server stops transfer on receipt of an abort message'	Passed
1.4.2 [1008,0]	Passed
1.5 Check reinitialization of SDO transfers during transfer init	
1.5.1 Execute: 'Check reinitialization of SDO transfers during transfer init'	Passed
1.5.2 [1008,0]	Passed
1.6 Check reinitialization of SDO transfer during segment transfer	
1.6.1 Execute: 'Check reinitialization of SDO transfer during segment transfer'	Passed
1.6.2 [1008,0]	Passed
1.7 Check reinitialization of different transfer with valid CCS	
1.7.1 Execute: 'Check reinitialization of different transfer with valid CCS'	Passed

Conformance test snippet (SDO upload component)

Further testing

- Standard *GCC GCOV* used to evaluate test-suite code coverage
  - Used in conjunction with conformance tests
  - Unit tests verify individual stack components using mock-inputs without inclusion of hardware

```
    :
    : /**
    :  * Function for accessing _Synchronous counter overflow value_ (index 0x1019) from SDO server
    :  *
    :  * For more information see file CO_SDO.h.
    :  */
  1 : static CO_SDO_abortCode_t CO_ODF_1019(CO_ODF_arg_t *ODF_arg){
    :     CO_SYNC_t *SYNC;
    :     uint8_t value;
  1 :     CO_SDO_abortCode_t ret = CO_SDO_AB_NONE;
    :
  1 :     SYNC = (CO_SYNC_t*) ODF_arg->object;
  1 :     value = ODF_arg->data[0];
    :
  1 :     if(!ODF_arg->reading){
  0 :         uint8_t len = 0U;
    :
  0 :         if(SYNC->periodTime){
  0 :             ret = CO_SDO_AB_DATA_DEV_STATE;
    :         }
  0 :         else if((value == 1) || (value > 240 && value <= 255)){
  0 :             ret = CO_SDO_AB_INVALID_VALUE;
    :         }
    :         else{
  0 :             SYNC->counterOverflowValue = value;
  0 :             if(value != 0){
  0 :                 len = 1U;
    :             }
    :
  0 :             SYNC->CANtxBuff = CO_CANtxBufferInit(
    :                     SYNC->CANdevTx,         /* CAN device */
  0 :                     SYNC->CANdevTxIdx,      /* index of specific buffer inside CAN module */
  0 :                     SYNC->COB_ID,           /* CAN identifier */
    :                     0,                      /* rtr */
    :                     len,                    /* number of data bytes */
    :                     0);                     /* synchronous message flag bit */
    :         }
    :     }
    :
  1 :     return ret;
    : }
```

```
    :
    : /**
    :  * Function for accessing _Synchronous counter overflow value_ (index 0x1019) from SDO server.
    :  *
    :  * For more information see file CO_SDO.h.
    :  */
  4 : static CO_SDO_abortCode_t CO_ODF_1019(CO_ODF_arg_t *ODF_arg){
    :     CO_SYNC_t *SYNC;
    :     uint8_t value;
  4 :     CO_SDO_abortCode_t ret = CO_SDO_AB_NONE;
    :
  4 :     SYNC = (CO_SYNC_t*) ODF_arg->object;
  4 :     value = ODF_arg->data[0];
    :
  4 :     if(!ODF_arg->reading){
  4 :         uint8_t len = 0U;
    :
  4 :         if(SYNC->periodTime){
  2 :             ret = CO_SDO_AB_DATA_DEV_STATE;
    :         }
  3 :         else if((value == 1) || (value > 240 && value <= 255)){
  1 :             ret = CO_SDO_AB_INVALID_VALUE;
    :         }
    :         else{
  1 :             SYNC->counterOverflowValue = value;
  1 :             if(value != 0){
  1 :                 len = 1U;
    :             }
    :
  3 :             SYNC->CANtxBuff = CO_CANtxBufferInit(
    :                     SYNC->CANdevTx,         /* CAN device */
  1 :                     SYNC->CANdevTxIdx,      /* index of specific buffer inside CAN module */
  1 :                     SYNC->COB_ID,           /* CAN identifier */
    :                     0,                      /* rtr */
    :                     len,                    /* number of data bytes */
    :                     0);                     /* synchronous message flag bit */
    :         }
    :     }
    :
  4 :     return ret;
    : }
```

Coverage testing

Activity outcomes

- Evaluated multiple open-source CANopen alternatives
  - Features abundant, quality of implementations generally high
- Selected candiate among alternatives
  - Pruned codebase
  - Extended stack to include ECSS-ST-50-15-C functionality
  - Verified compliance to required CANopen components to a high degree
  - Developed test-suite providing code-coverage to a high degree
- Full qualification to software criticality level B still withstanding

I will not use the word FIRMWARE again. I will not use the word FIRMWARE again.
I will not use the word FIRMWARE again. I will not use the word FIRMWARE again.
I will not use the word FIRMWARE again. I will not use the word FIRMWARE again.
I will not use the word FIRMWARE again. I will not use the word FIRMWARE again.
I will not use the word FIRMWARE again. I will not use the word FIRMWARE again.
I will not use the word FIRMWARE again. I will not use the word FIRMWA in.
I will not use the word FIRMWARE again. I will not use the word FIRMW ain.
I will not use the word FIRMWARE again. I will not use the word FIRM gain.
I will not use the word FIRMWARE again. I will not use the word FIRMWA gain.
I will not use the word FIRMWARE again. I will not use the

15k stack to use full CANOpen. Compare it with FPGA.