2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

# How to achieve verification closure of configurable code by combining static analysis and dynamic testing

Antonello Celano, STMicroelectronics,

Alexandre Langenieux, MathWorks

life.augmented

MathWorks®
Accelerating the pace of engineering and science

accellera
SYSTEMS INITIATIVE

# Context

- Develop generic MCAL  Module drivers (e.g. Mcu,CAN) for PowerPC and ARM for Automotive customers
- Those drivers must be configurable for each customer
- Respect of Automotive safety and security standards

# Challenge

- Extreme configurable code

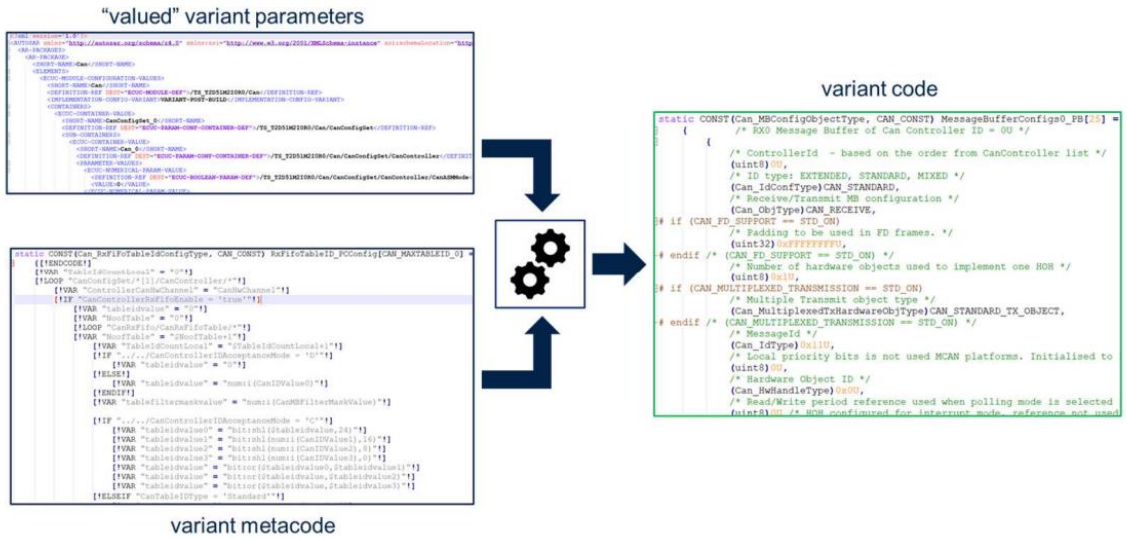
| Project | Number of variant parameters | Number of boolean preprocessor macro (#define) | Number of software variants considering only boolean parameters |
|---|---|---|---|
| MCU Driver | 357 (208 booleans, 78 enumerations, 103 integers, 9 strings) | 58 | $2^{58} = 288*10^{15}$ |
| CAN Module | 97 ( 32 booleans, 25 enumerations, 39 integers, 9 strings) | 50 | $2^{50} = 10^{15}$ |

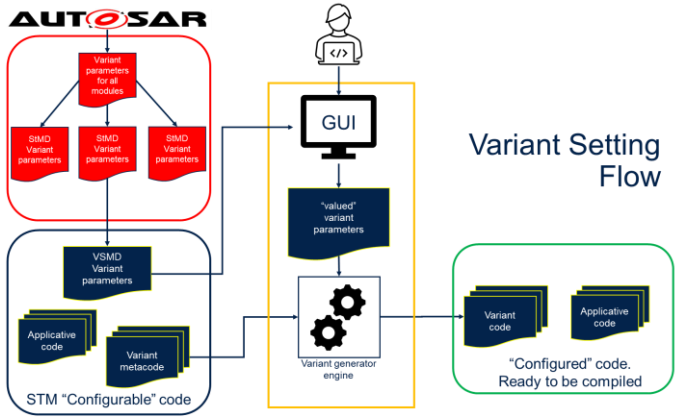
- Fulfill standard code verification requirements
  - MISRA-C:2012
  - CERT-C
  - Code Coverage (Decision, Condition, MC/DC)

- Increase confidence in software verification

# MCAL and CAN Drivers Variant Definition

- Configurable code
  - Metacode
  - Source Code
  - Configuration parameters



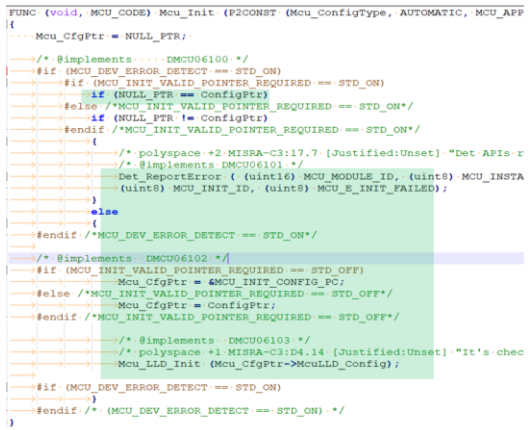




- Variant
  - Source code
  - Generated code
    - Metacode
    - "Valued" cfg parameters

# Variant Subset Selection for Verification Closure

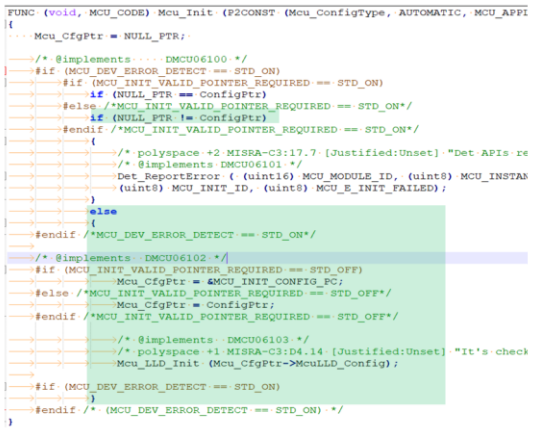
Variant
MCU_DEV_ERROR_DETECT=STD_ON
MCU_INIT_VALID_POINTER_REQUIRED= STD_ON

Reachable lines = 10 - Tested Line = 7
Statement coverage score = 70%



Variant
MCU_DEV_ERROR_DETECT=STD_ON
MCU_INIT_VALID_POINTER_REQUIRED= STD_OFF

Reachable lines = 10 - Tested Line = 7
Statement coverage score = 70%



Aggregate

Reachable lines = 11 - Tested Line = 8
Statement coverage score = 72%

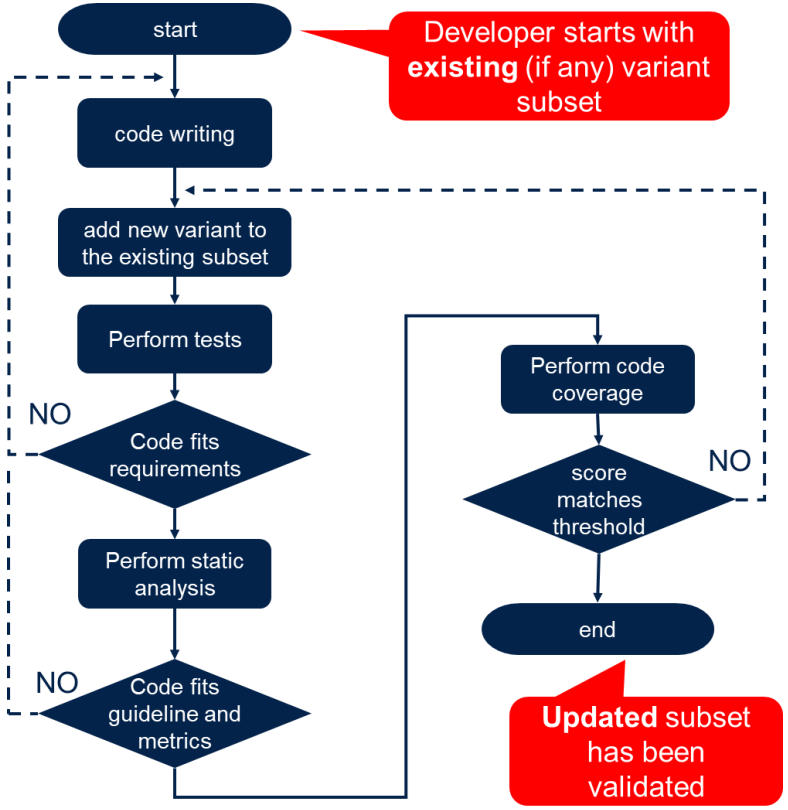
```
FUNC (void, MCU_CODE) Mcu_Init (P2CONST (Mcu_ConfigType, AUTOMATIC, MCU_APPL_CONST) Cc
{
    Mcu_CfgPtr = NULL_PTR;

    /* @implements     DMCU06100 */
#if (MCU_DEV_ERROR_DETECT == STD_ON)
    #if (MCU_INIT_VALID_POINTER_REQUIRED == STD_ON)
        if (NULL_PTR == ConfigPtr)
    #else /*MCU_INIT_VALID_POINTER_REQUIRED == STD_ON*/
        if (NULL_PTR != ConfigPtr)
    #endif /*MCU_INIT_VALID_POINTER_REQUIRED == STD_ON*/
        {
            /* polyspace +2 MISRA-C3:17.7 [Justified:Unset] "Det APIs return alway
            /* @implements DMCU06101 */
            Det_ReportError ( (uint16) MCU_MODULE_ID, (uint8) MCU_INSTANCE_ID, \
            (uint8) MCU_INIT_ID, (uint8) MCU_E_INIT_FAILED);
        }
        else
        {
#endif /*MCU_DEV_ERROR_DETECT == STD_ON*/

    /* @implements DMCU06102 */
#if (MCU_INIT_VALID_POINTER_REQUIRED == STD_OFF)
        Mcu_CfgPtr = &MCU_INIT_CONFIG_PC;
#else /*MCU_INIT_VALID_POINTER_REQUIRED == STD_OFF*/
        Mcu_CfgPtr = ConfigPtr;
#endif /*MCU_INIT_VALID_POINTER_REQUIRED == STD_OFF*/

        /* @implements DMCU06103 */
        /* polyspace +1 MISRA-C3:D4.14 [Justified:Unset] "It's checked when DE
        Mcu_LLD_Init (Mcu_CfgPtr->McuLLD_Config);

#if (MCU_DEV_ERROR_DETECT == STD_ON)
        }
#endif /* (MCU_DEV_ERROR_DETECT == STD_ON) */
}
```

Variant Selection:

- iterative process
- developer expertise
- aggregated code coverage

# Verification Closure Development Process

- On each selected variant
  - Testing
    - Functional testing
    - Robustness testing
  - Static analysis
    - Coding rules checking
    - Code metrics
- Consolidate results for each category



# Results

- Errors found earlier in software, before reaching the customer
- Quality of all possible variants is controlled

| Project | Number of software variants considering only boolean parameters | Subset of variants used with the described methodology | Coverage (Statement, branch, MCDC) thresholds | Achieved Coverage Score |
|---|---|---|---|---|
| MCU Driver | $2^{58} = 288*10^{15}$ | 177 | 100% | 100% |
| CAN Module | $2^{50} = 10^{15}$ | 179 | 100% | 100% |

# Future Enhancements

- Automatic extraction of the smallest software variants
- Improve code metrics consolidation across variants
- Extend code verification to formal code verification

# Take Away

- Significant improvement of
  - productivity of ST development team
  - the quality of the configurable software
- Reusable framework beyond firmware development
- Possible to extend this method to other software verification activities

# Questions