

μRTE

A modeling framework for safe code on (Multi-Core) Controllers

INTRODUCTION



M.Sc. Thomas Barth



Prof. Dr. -Ing. Peter Fromm



Motivation



Project timeline



Initial idea

Prof. Fromm starts the implementation of a lightweight runtime environment to be used in the academic environment



First industrial contact

Prof. Fromm consults the company Linde who considers to introduce AUTOSAR™ on an AURIX™ multi-core microcontroller for forklift control. After an evaluation phase, an improved version of the lightweight runtime environment is introduced instead.

Thomas Barth joins the research group for his master thesis and starts a PhD program afterwards.



Public funding

Along with the FZI at the Karlsruhe Institute of Technology and the company HighTec, the research group at Hochschule Darmstadt attracts funding for 2 years to research the applicability of multi-core microcontrollers for safety critical applications.

A feasibility study is conducted in which a possible framework for product development is identified. The development of μRTE starts during this phase.



Industrial evaluation

An prototype of μRTE is evaluated in an industrial project in which the software for an safety relevant HMI for an agricultural machine is implemented.

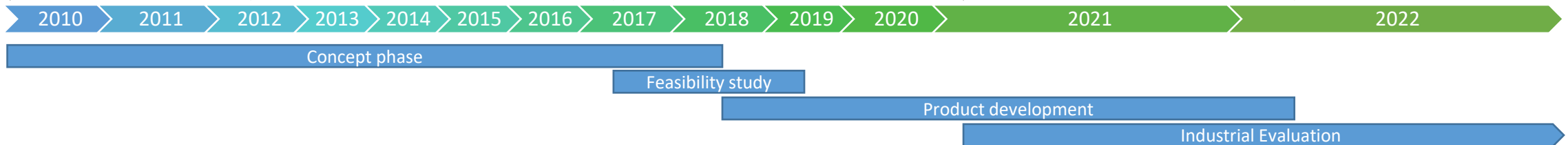
In this project, μRTE is used for software architecture definition, code generation, requirement management and documentation generation.



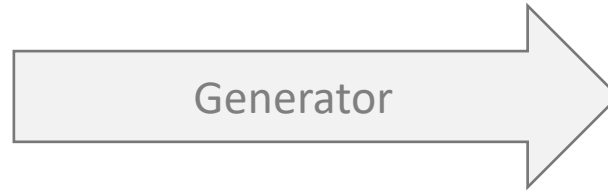
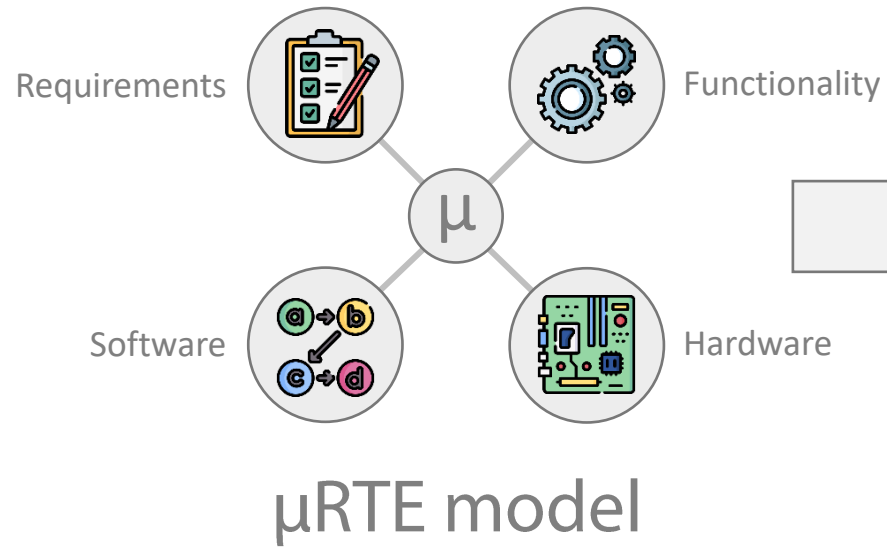
Pre-Release

A first version of μRTE including all core-features is released:

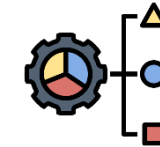
- Software Modeling
- Hardware Modeling
- Functional Modeling
- Requirements Modeling
- Automated model assessment
- Code generation
- Report generation



Scope



Code Frame
Activation Framework
Data encapsulation



Memory allocation
Memory protection



Report



Test Frame (WIP)

Features



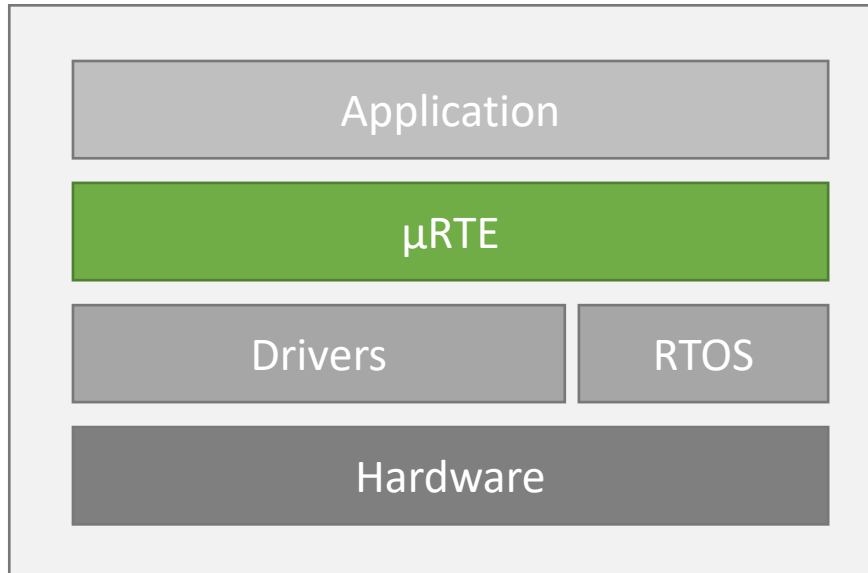
Portability

- Any RTOS
- Any Controller Single/Multi-Core
- Any Toolchain (C/C++ on GCC implemented)



Software features

- Cyclic and Signal driven activation of runnables
- Multiple system-states with own data-flows
- Extensive system and application error handling
- Human readable code



Safety

- Complete modeling of the safety case
- Inbuilt consistency and safety checks
- Complete traceability from requirements to implementation units
- Supports freedom from interference
- Generated code complies with MISRA standard



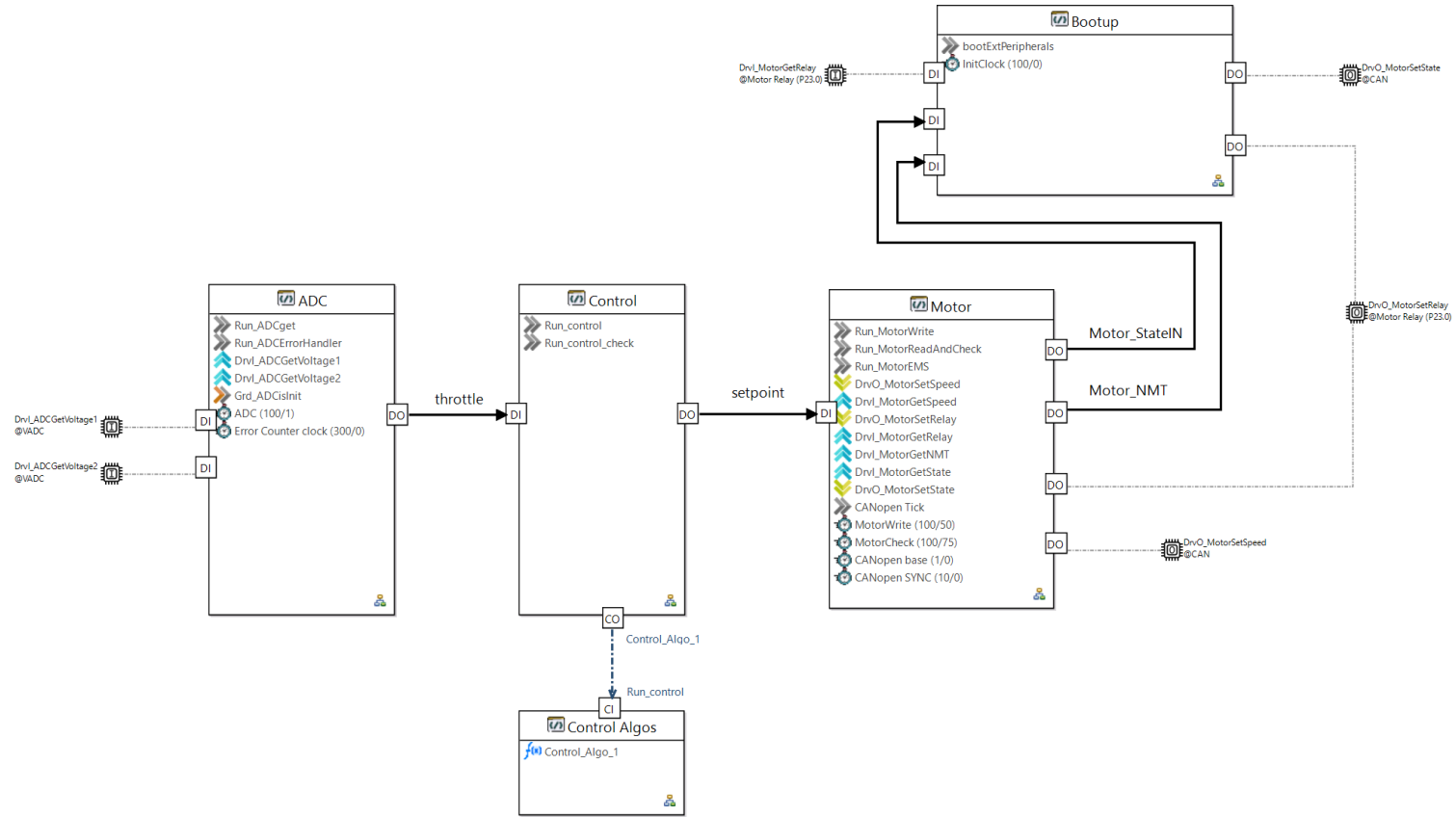
Process

- Encourages clean and modular solutions
- Eclipse plugin implementation
- Intuitive and rich graphical modeling
- Complete support of V-model
- Extensive auto-generated documentation

Software modeling

Each block represents a code file in which software functions are contained
 Arrows represent data exchange
 Chip-Icons represent hardware interfaces

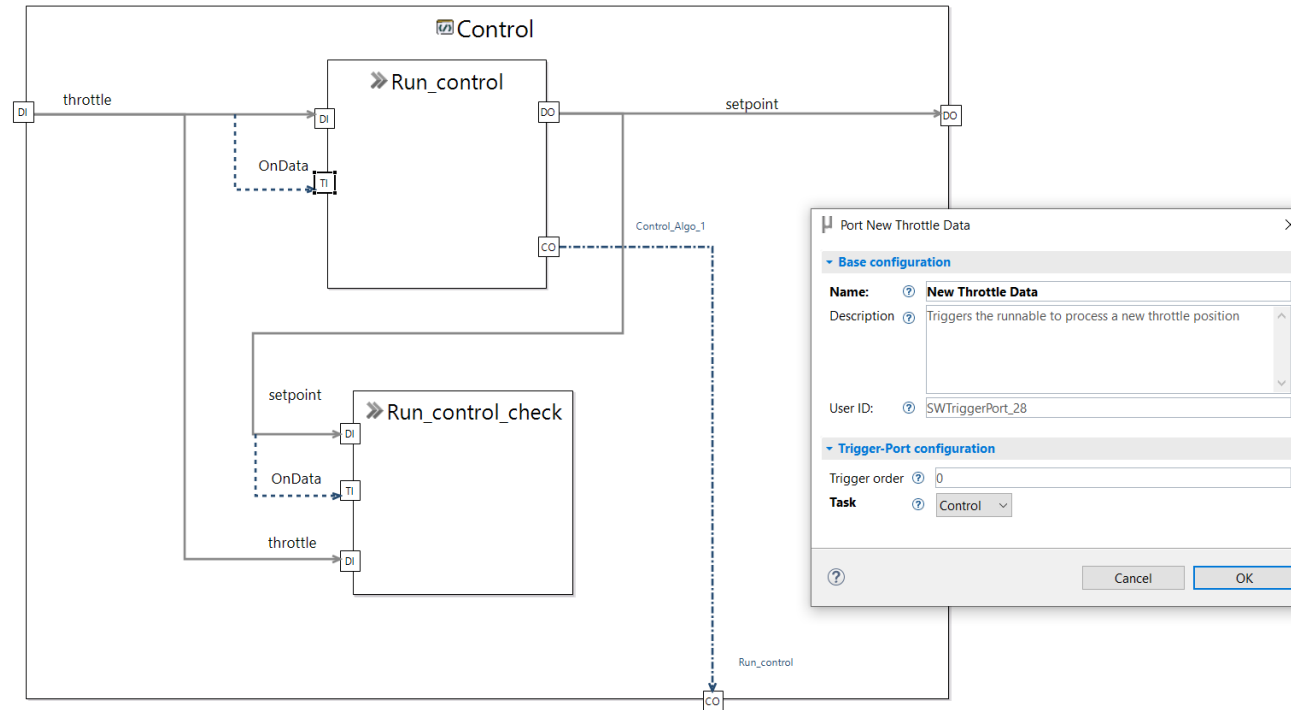
- μRTE System Model
 - 01 Software Model
 - Application
 - LED
 - Throttle Control
 - SWC ADC
 - SWC Control
 - SWC Motor
 - SWC Bootup
 - SWC Control Algos
 - SWC LED
 - Source-Folder Complex Device Drivers
 - Source-Folder Algorithms
 - RTE
 - RTOS
 - Linkage
 - DataTypes
 - Hardware Model
 - Logical Function Model
 - Requirement Model



Software modeling

Detailed view describes runnable execution

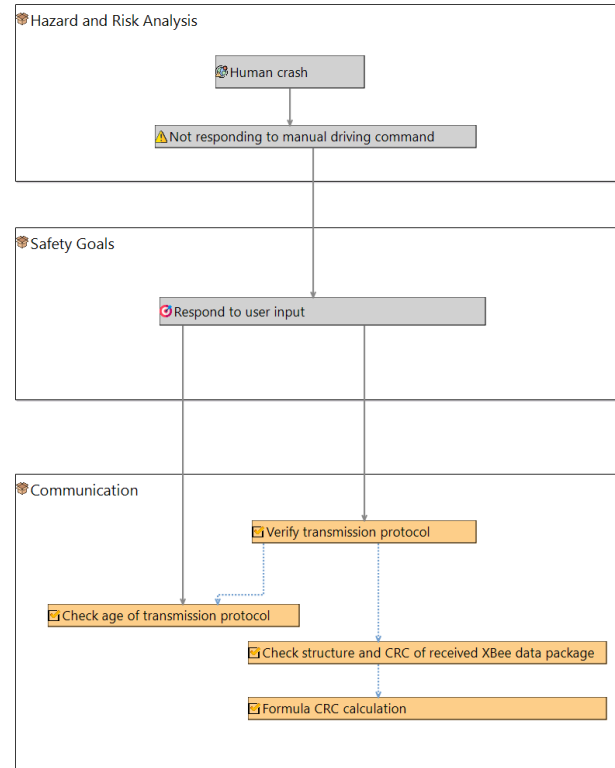
- μRTE System Model
 - 01 Software Model
 - Application
 - LED
 - Throttle Control
 - SWC ADC
 - SWC Control
 - Control
 - Runnable Run_control
 - Runnable Run_control_check
 - SWC Motor
 - SWC Bootup
 - SWC Control Algos
 - SWC LED
 - Source-Folder Complex Device Drivers
 - Source-Folder Algorithms
 - RTE
 - RTOS
 - Linkage
 - DataTypes
 - Hardware Model
 - Logical Function Model
 - Requirement Model



Requirements modeling

Derivation of requirements
Relationship (refinement/conflict) among requirements
Mapping to implementation units

- μRTE System Model
 - Software Model
 - Hardware Model
 - Logical Function Model
 - Requirement Model
 - Overview
 - Package Top Level Requirements
 - Package Hazard and Risk Analysis
 - Hazardous Event Not responding to manual driving command
 - Hazardous Event Not detecting obstacle
 - Hazardous Event Wrong actuator action
 - Hazardous Event Wrong targetspeed / controlspeed calculation
 - Hazard Scenario Human crash
 - Package Safety Goals
 - Safety Goal Respond to user input
 - Safety Goal Stop in front of obstacle
 - Safety Goal Supervise engine
 - Safety Goal Plausibility of targetspeed values
 - Package Communication
 - Safety Requirement Verify transmission protocol
 - refined by Check structure and CRC of received XBee data package
 - refined by Check age of transmission protocol
 - Safety Requirement Check age of transmission protocol
 - Requirement Dispatch received protocol



Report

Generated as HTML
 Each object in the model has a an own report page
 Properties and dependencies are shown textual and graphical

Car_5
 Runnable: ENGINE_setSpeed_run (Runnable_125)

Runnable
ENGINE_setSpeed_run
 Calculate the speed for the 4 engines

Warnings

Safety (1)
 Safety Warnings for this Runnable.
 Safety warnings are related to the Requirements Layer, especially the SIL

Design (3)
 Design Warnings for this Runnable
 Design warnings are related the configuration of elements within the model.

Diagrams

Logical Function Model Requirements Model Software Model Hardware Model

Safety

Required	
SIL derived	SIL_1
overwrite (SIL_manual)	derived
reason (SIL_manual_reason)	
SIL	SIL_1
Achieved	
SIL achieved	QM
justification	

Requirement Layer

(Safety)Requirements (4)
 (Safety)Requirements referencing to runnable ENGINE_setSpeed_run.

Software Layer

IN Signals (2)
 Ingoing signals of runnable ENGINE_setSpeed_run.

OUT Signals (1)
 Outgoing signals of runnable ENGINE_setSpeed_run.

Hidden columns:
 » Tasks » SystemStates » Requirements » minimum Age » maximum Age » Checksum » Force Sync » effective inline » SIL req » SIL ach » Initial value (D) » Pointer access (D) » Datatype (D) » Alt-In (D) » Alt-Out (D) » In-Driver (D) » Out-Driver (D) » OnData (D) » OnError (D) » OnTrigger (E)

Signal	Type	Storage	Runnables OUT	Runnables IN
FAULHABER_speed_out Interface towards the speed object on the motion controller	Data	local in BaseSystem	ENGINE_setSpeed_run	CANOPEN_bc_run

Tasks (1)
 Tasks runnable ENGINE_setSpeed_run is executed by.

Activation-Events (1)
 Activation Events triggering the execution of runnable ENGINE_setSpeed_run.

Runnables providing data (2)
 Runnables providing data for runnable ENGINE_setSpeed_run.

Runnables triggered (1)
 Runnables triggered by runnable ENGINE_setSpeed_run.

Runnables receiving data (1)
 Runnables receiving data from runnable ENGINE_setSpeed_run.

Hardware Layer

Hardware Components (1)
 Hardware runnable ENGINE_setSpeed_run is associated with. Including executing CPUs, hardware referenced by its protectionSets and driver hardware of connecting signals.

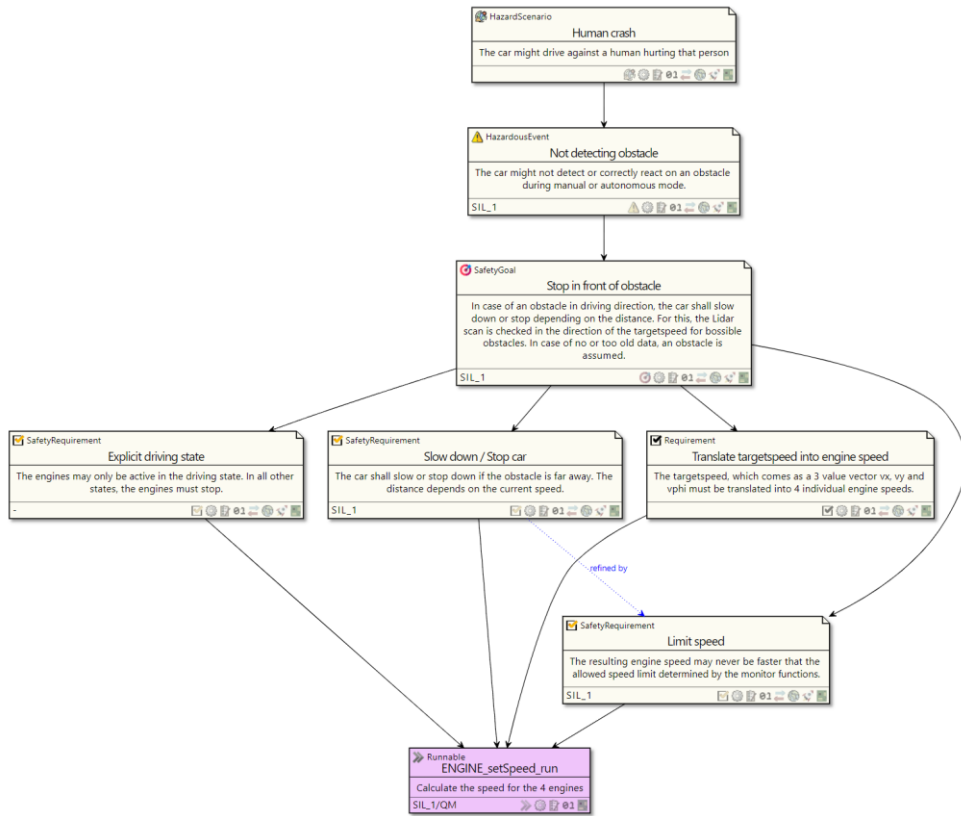
Report

Generated as HTML

Each object in the tree has a an own report

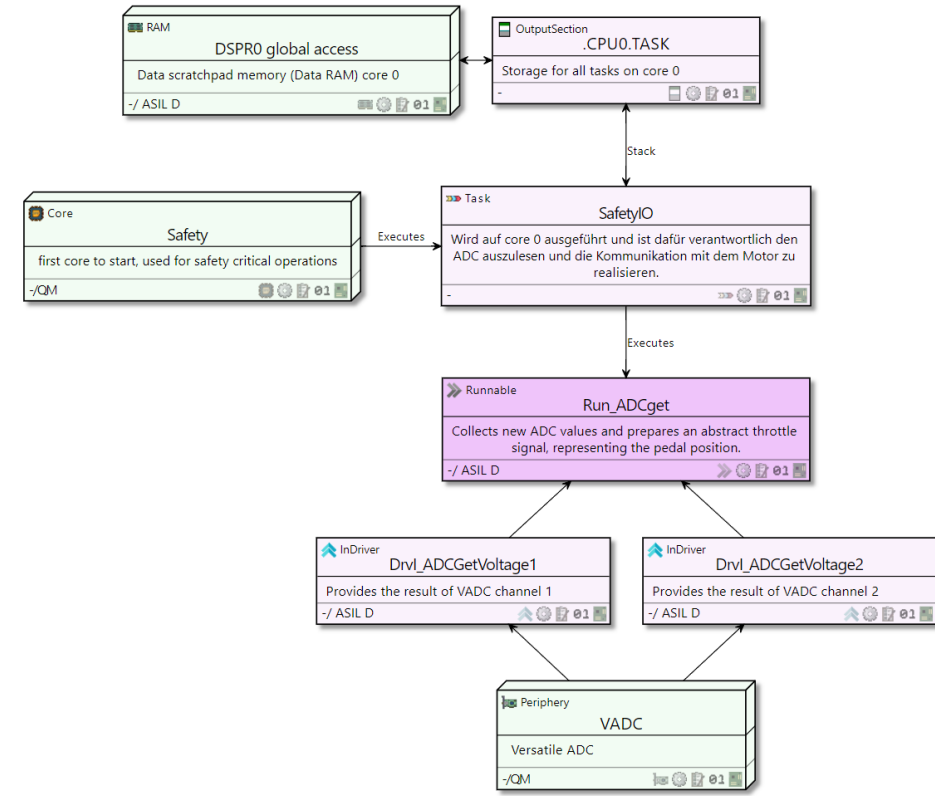
Properties and dependencies are shown textual and graphical

Runnable ENGINE_setSpeed_run - Requirements Model Dependencies



µRTE **PROTOTYPE** Runnable ENGINE_setSpeed_run in Model Car_5 generated on 25.03.2022 10:49:51
Model hash: IRkmbQ8E9d+bAqIP5eYzHBqRPFHFGmOH6eK3x9r7nd6PhPJDPH8I3hbl37LkBOU5gT37AGA267sw01KA==

Runnable Run_ADCget - Hardware Model Dependencies



µRTE **PROTOTYPE** Runnable Run_ADCget in Model SafetyBoard_AURIX generated on 01.04.2022 17:09:05
Model hash: Zdnr0wRMsh0jIn/U1sar5d2xNXaSNpRQ+9HjuXlOLqIMVJnR9thFCISH8lq/svFSWooVj3/HFofAmMGcbEA==

Report

Checkers for inconsistency, safety and design
Warnings are reported as globally and associated with affected elements

Runnable
Monitor_cyclicCheck_run
This runnable will perform cyclic sanity checks for the system

Warnings

Safety (2)
Safety Warnings for this Runnable.
Safety warnings are related to the Requirements Layer, especially the SIL

- Monitor_cyclicCheck_run has a SIL effective of SIL_1 but a SIL achieved of QM
- Multiple Technical functions for Monitor_cyclicCheck_run: Data transmission over XBee, Detect obstacle

Problems Console
0 errors, 76 warnings, 0 others

Description	Location
Warnings (76 items)	
Core Arm Cortex®-M7 (QM) was selected for execution for ActivationEngine which does not satisfy the SIL required of SIL_1.	ActivationEngine
CyclicEvent CAN_dispatch_clock is not associated with any trigger port.	CyclicEvent_CAN_dispatch_clock
DataType bool is unused.	DataType_bool
DataType CAN_RX_MSG_t is unused.	DataType_CAN_RX_MSG_t
DataType int16_t is unused.	DataType_int16_t
DataType sint32_t is unused.	DataType_sint32_t
DataType uint32_t is unused.	DataType_uint32_t
ECU PSOC SmartPower has a SIL effective of SIL_1 but a SIL achieved of QM	ECU_PSOC SmartPower
Function DRIVE_0_calibration_fun is not directly associated with a Technical Function.	Function_DRIVE_0_calibration_fun

Safety Warnings (2)
All Safety Warnings of this model.
Safety warnings are related to the Requirements Layer, especially the SIL.

- .CPU0.TASK is set as stack for task SafetyIO but has more references: ActivationEngine, SafetyIO.
- .CPU0.TASK is set as stack for the central activation engine but has more references: ActivationEngine, SafetyIO.

Design Warnings (3)
All Design Warnings of this model.
Design warnings are related the configuration of elements within the model.

- The Runnable Data-Port-IN name "setpoint" was given multiple times at runnables: Run_MotorWrite, Run_control_check
- The Runnable Data-Port-IN name "throttle" was given multiple times at runnables: Run_control, Run_control_check
- WatchPool does not contain any signals.

RTE Warnings (6)
All RTE Warnings of this model.
RTE warnings are related to the configured behaviour of the RTE.

- LEDs has no maximum age and therefore never expires.
- Motor_SpeedOUT has no maximum age and therefore never expires.
- Motor_StateOUT has no maximum age and therefore never expires.
- Local signal buffers for runnables will be created without initial value and the invalid signal behavior is set to "no update". It is the responsibility of the user to make sure that appropriate error handling is implemented when using the signal buffers as they might contain corrupted data in case of failed signal read operations.
- .RTE_globalLMU is accessible from multiple tasks: Control, QM, SafetyIO.
- MotorRelay is written from multiple runnables but in different system states: Run_MotorEMS with states: error_detected, operational, bootExtPeripherals with states: init

Code

Runnables and other functions (Drivers etc.):
 Developer works in designated code blocks
 Documentation and dependencies are generated as comment

```

/* ----- Runnable "ENGINE_setSpeed_run" -----
 * Calculate the speed for the 4 engines
 *
 * User-ID:      Runnable_125
 * SystemStates: "normal"
 * WCET:        0
 * Stack:       0
 * ROM:         0
 *
 * --- Triggers ---
 * - TriggerPort "TI_ENGINE_setSpeed_run_50_0ms"
 *   Task: "Safety"
 *   System Events:
 *     - Cyclic Trigger "setSpeedClock" Cycle Time: 50, Offset 0
 *
 * --- Data Signals IN ---
 * - DataPort "DI_CONTROL_limits" User-ID: SWDataPort_129
 *   Signals:
 *     - "CONTROL_limits" User-ID: SignalDataObject_14:
 *       Description: Structure containing driving limiting parameters like maxSpeed
 *       Age:          Max: 300
 *       Storage:      Signalpool "Safety"
 *       Payload Datatype: generated, see #CONTROL_limits_cfg.h
 *       Runnables writing to this signal:
 *         - SWC "SWC_Monitor"
 *         - "Monitor_cyclicCheck_run"
 *
 * - DataPort "DI_CONTROL_targetspeed" User-ID: SWDataPort_126
 *   Signals:
 *     - "CONTROL_targetspeed" User-ID: SignalDataObject_16:
 *       Description: The car targetspeed
 *       Age:          Max: 300
 *       Storage:      local signal
 *       Payload Datatype: generated, see #CONTROL_targetspeed_cfg.h
 *       Runnables writing to this signal:
 *         - SWC "SWC_Control"
 *         - "CONTROL_drive_run"
 *
 * --- Data Signals OUT ---
 * - DataPort "DO_FAULHABER_speed_out" User-ID: SWDataPort_128
 *   Signals:
 *     - "FAULHABER_speed_out" User-ID: SignalDataObject_20:
 *       Description: Interface towards the speed object on the motion controller
 *       Storage:      local signal
 *       Payload Datatype: generated, see #FAULHABER_speed_out_cfg.h
 *       Runnables reading from this signal:
 *         - SWC "SWC_BSH_CanOpen"
 *         - "CANOPEN_tx_run"
 *
 *   Trigger:
 *     OnData:
 *       - "TI_CANOPEN_tx_run_FAULHABER_speed_out_onData" triggering runnable "CANOPEN_tx_run" in Task "Safety"
    
```

```

 * --- SAFETY ---
 *   SIL manual:    derived
 *   SIL effective: SIL_1
 *   SIL achieved:  QM
 *
 * --- Requirements ---
 * - "Translate targetspeed into engine speed"
 *   Description:
 *     The targetspeed, which comes as a 3 value vector vx, vy and vphi must be translated into 4 individual engine speeds.
 *
 * --- SafetyRequirements ---
 * - "Explicit driving state" SIL effective: derived
 *   Description:
 *     The engines may only be active in the driving state. In all other states, the engines must stop.
 *   Other References of the SafetyRequirement:
 *     - Runnable Monitor_cyclicCheck_run
 * - "Limit speed" SIL effective: SIL_1
 *   Description:
 *     The resulting engine speed may never be faster than the allowed speed limit determined by the monitor functions.
 * - "Slow down / Stop car" SIL effective: SIL_1
 *   Description:
 *     The car shall slow or stop down if the obstacle is far away. The distance depends on the current speed.
 *   Other References of the SafetyRequirement:
 *     - Runnable Monitor_cyclicCheck_run
 *
 * Start of user code ENGINE_setSpeed_run implementation notes
 *
 * End of user code
 */
void ENGINE_setSpeed_run(
    /* Trigger */
    const urTE_Triggers_t triggerPort,

    /* Incoming Data-Signals */
    Signal_CONTROL_limits& sig_IN_CONTROL_limits, /* Structure containing driving limiting parameters like maxSpeed */
    Signal_CONTROL_targetspeed& sig_IN_CONTROL_targetspeed, /* The car targetspeed */

    /* Outgoing Data-Signals */
    Signal_FAULHABER_speed_out* const p_sig_OUT_FAULHABER_speed_out /* Interface towards the speed object on the motion controller */
){
    //local IN signal value buffers
    Signal_CONTROL_limits::data_t CONTROL_limits_data;
    Signal_CONTROL_targetspeed::data_t CONTROL_targetspeed_data;

    //Start of user code implementation ENGINE_setSpeed_run
    
```

Code

Internal code:

Focus on readability and simplicity

Generic and minimalistic low level interfaces for easy integration

Avoidance of inheritance with dedicated code generated for each element

```
uRTE_boolean_t Signal_ADC1::isValid(void) const {
    //return value buffer
    uRTE_boolean_t ret = false;

    //check if signal is in the valid state
    ret = uRTE_SignalStatus_valid==this->m_content.m_status;

    //check age only if previous steps succeeded
    if(uRTE_true==ret){
        //signal age buffer
        const uRTE_SignalTime_t signal_age = uRTE_getTimeStamp() - this->m_content.m_timestamp;

        //check if the signal is older than the maximal age
        if(SIGNAL_ADC1_AGE_MAX<signal_age){
            //signal is too old
            ret=uRTE_false;
        }
    }

    //check checksum only if previous steps succeeded
    if(uRTE_true==ret){
        ret = uRTE_checkChecksum(&(this->m_content), sizeof(Signal_ADC1::content_t), this->m_checksum);
    }

    return ret;
}
```

Signal internal code

```
static const PxProtectRegion_T Task_Control_Regions[] = {
    //RTE
    uRTE_TAE_CONTROL_MEMPROTSET

    //do not remove this line!
    {0,0,(PxProtectType_t)0}
};

void CTask_Control::Task_Func(PxTask_t myID, PxMbx_t myMailbox, PxEvents_t myActivationEvents)
{
    //Run activation engine
    uRTE_TAE_Control::Run();
}
```

Task integration

μRTE workflow

Integration
μRTE needs to be integrated onto a specific target and RTOS once



Requirements
The behavior of the system including safety requirements is defined



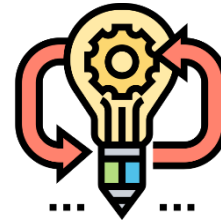
Test and Qualification
Unit, integration and system tests based on the report supported by the uniform structure of the generated code



Architecture
Definition of hardware, data-flows and overall software-structure



Iterative work and late changes possible even on architectural level



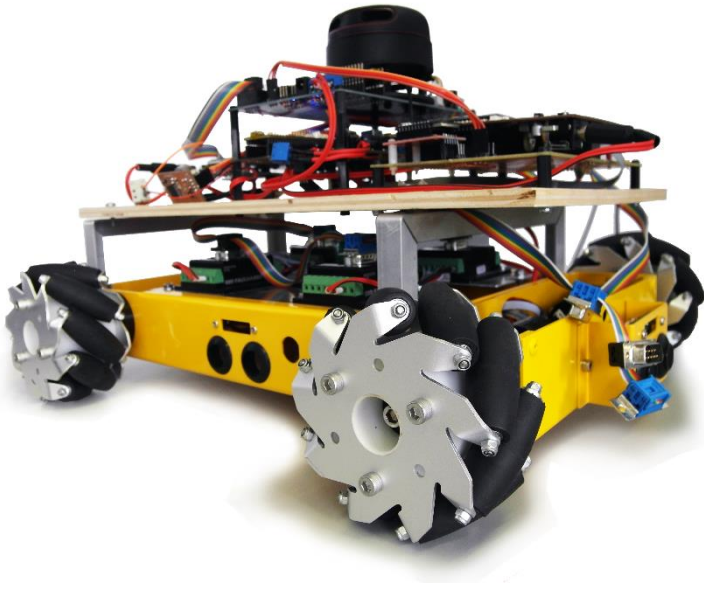
Development
Implementation of functionalities within the generated code-frame based on the report



Generation
Report and Code-Frame
Existing user-code is preserved



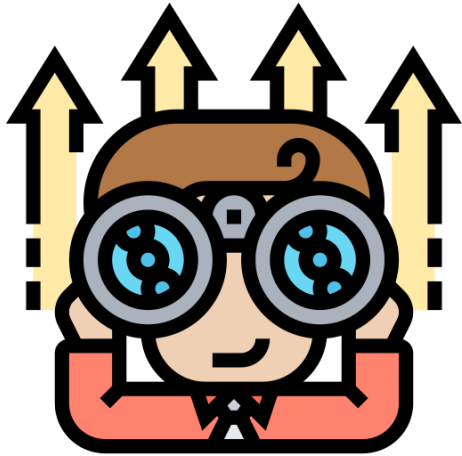
Experience



μRTE Demonstrator

- Fast familiarization
- Code focus moves to architectural focus
- Significantly improved system understanding
- Architectural refactoring support
- Cross platform reusability
- Good code analyzability / debugability

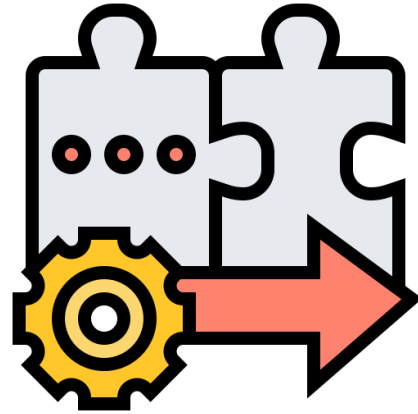
Possible extensions



- Test-Framework generation (WIP)
- OS-less implementation
- Extension towards interrupts, centralized error handling and more
- Certifiable implementation
- Readback of generated artifacts and comparison against model
- Linkage to other tools, CI/CD etc.

Integration

Interfaces



- One additional system task with central tick
- Tasks are not generated
- Global timestamp source
- Interfaces for inter-task messaging and notification