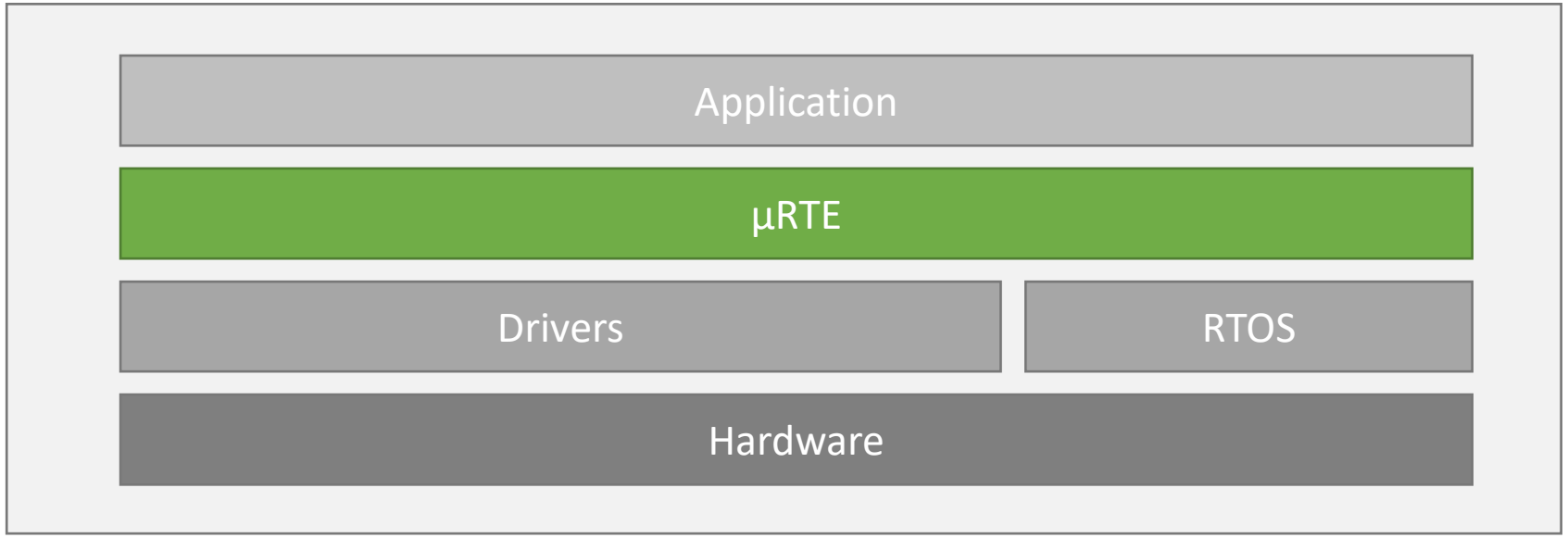
µRTE

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

Code Overview

µRTE is middleware



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_BSW_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 that 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
```

µRTE **Code overview**

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;
}
```

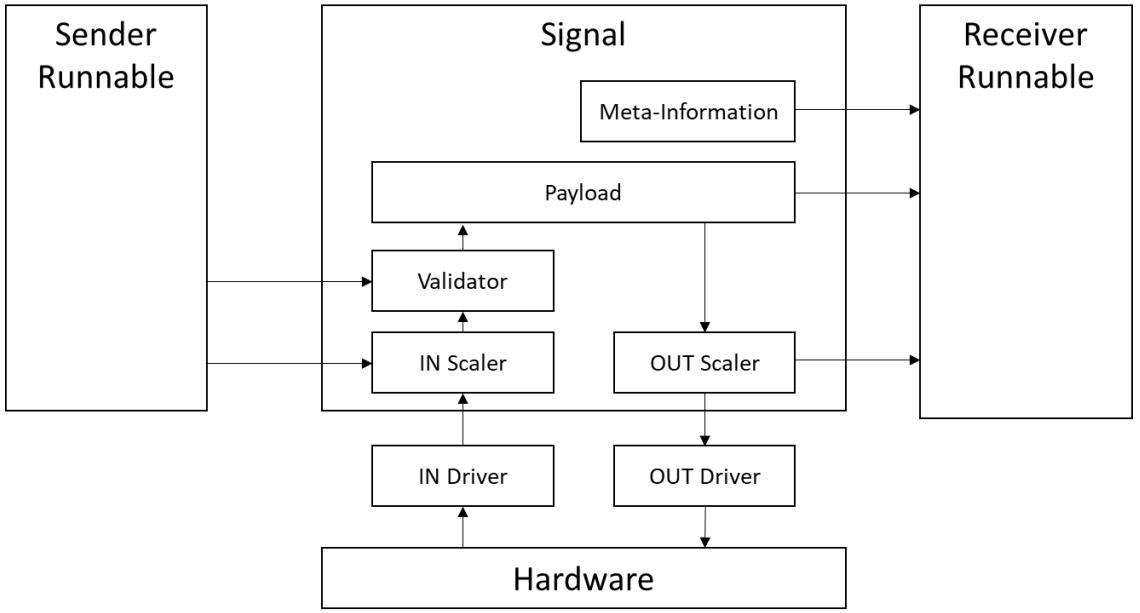
Signal internal code

```
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

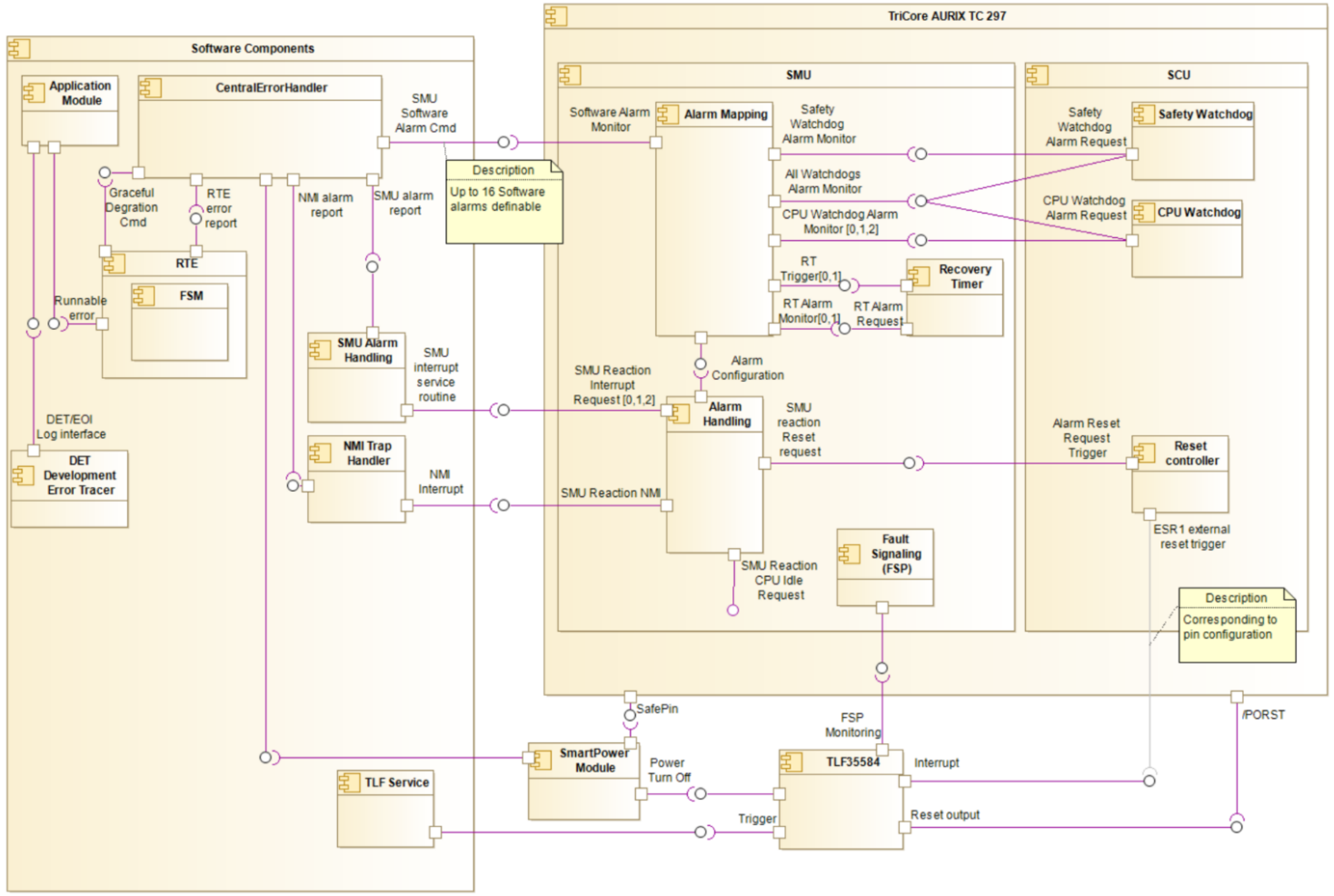
Signals

Signals abstract data exchange for inter-runnable and hardware communication
New data written to a signal needs to pass a validator before the payload is updated
If in- or outgoing datatypes differ from the signals payload, scalers are generated
Meta information such as the signals age, checksum or validity are saved along with the payload
Signal events like new data or errors can trigger the execution of runnables



Integration

Error handling example on Infineon AURIX TC297



µRTE **Code overview**