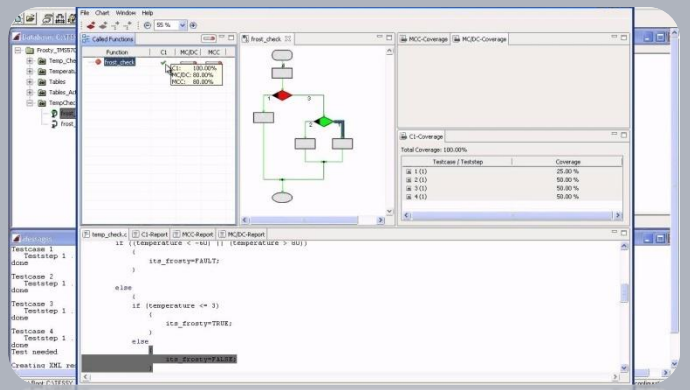
Simulationsbasiertes Testen von sicherheitskritischer Software auf Multicore Controllern

Prof. Dr.-Ing. Peter Fromm
Shrikant Sagar
Hochschule Darmstadt



Inhalt

- Functional Safety und Simulationsbasiertes Testen
- Das Werkzeug winAMS
- System under Test
- Evaluationsergebnisse
- Erweiterung des Ansatzes
- Zusammenfassung und Ausblick

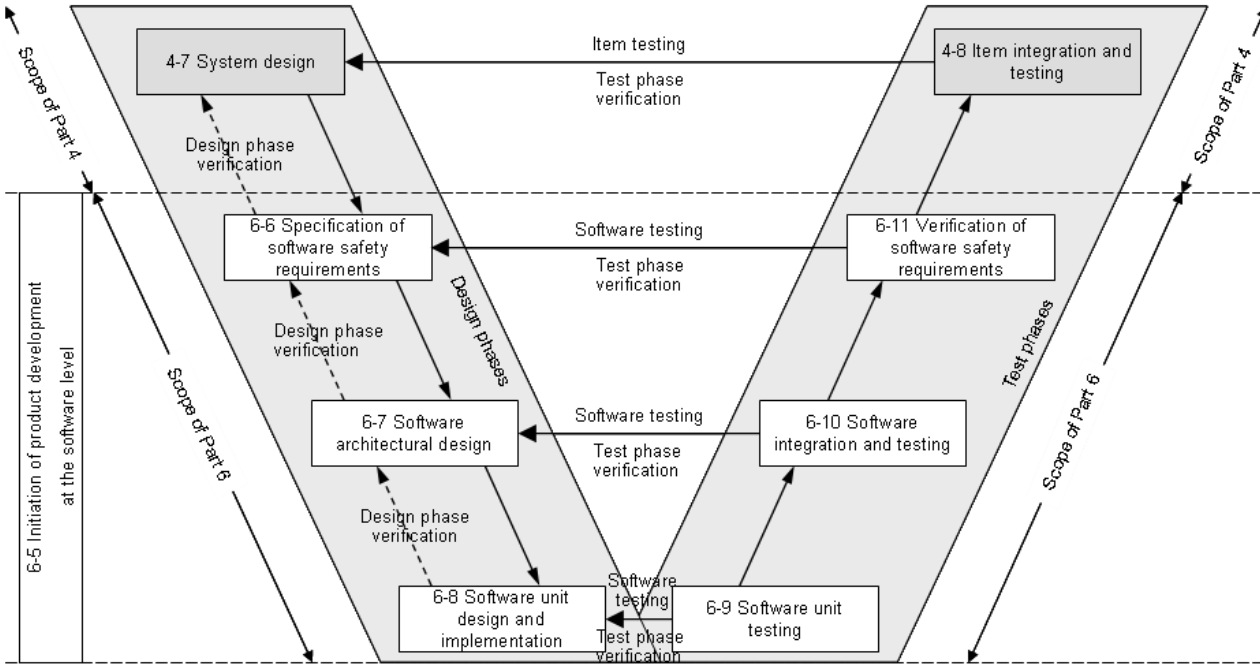
Funktionale Sicherheit

IEC 61508

ISO 26262
Automobil

ISO 13849
Maschinen

ISO xyz
...



Methodische Vorgaben am Beispiel ISO26262 - Auswahl

Table 11 — Methods for deriving test cases for software unit testing

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing[c]	+	+	+	+

[a] Equivalence classes can be identified based on the division of inputs and outputs, such that a representative test value can be selected for each class.

[b] This method applies to interfaces, values approaching and crossing the boundaries and out of range values.

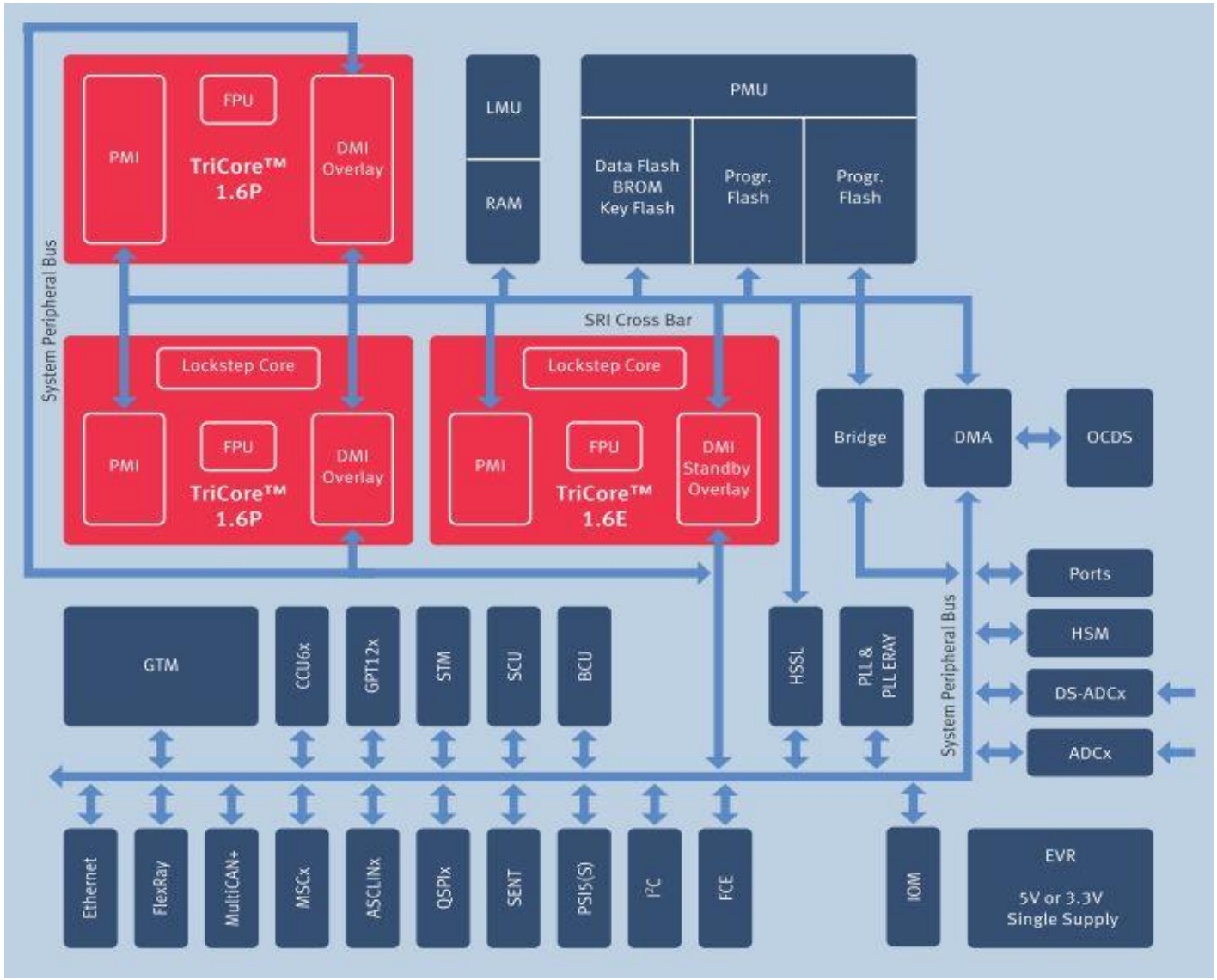
[c] "Error guessing tests" can be based on data collected through a "lessons learned" process and expert judgment.

Table 12 — Structural coverage metrics at the software unit level

Methods		ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

9.4.6 The test environment for software unit testing shall correspond as closely as possible to the target environment. If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases.

Aurix


PMI
FPU
TriCore™ 1.6P
DMI Overlay
LMU
RAM
PMU
Data Flash BROM Key Flash
Progr. Flash
Progr. Flash
System Peripheral Bus
SRI Cross Bar
Lockstep Core
PMI
FPU
TriCore™ 1.6P
DMI Overlay
Lockstep Core
PMI
FPU
TriCore™ 1.6E
DMI Standby Overlay
Bridge
DMA
OCDS
Ports
HSM
DS-ADCx
ADCx
GTM
CCU6x
GPT12x
STM
SCU
BCU
HSSL
PLL & PLLERAY
System Peripheral Bus
Ethernet
FlexRay
MultiCAN+
MSCx
ASCLINx
QSPIx
SENT
PSI5(S)
I²C
FCE
IOM
EVR
5V or 3.3V Single Supply

Mögliche Testumgebungen

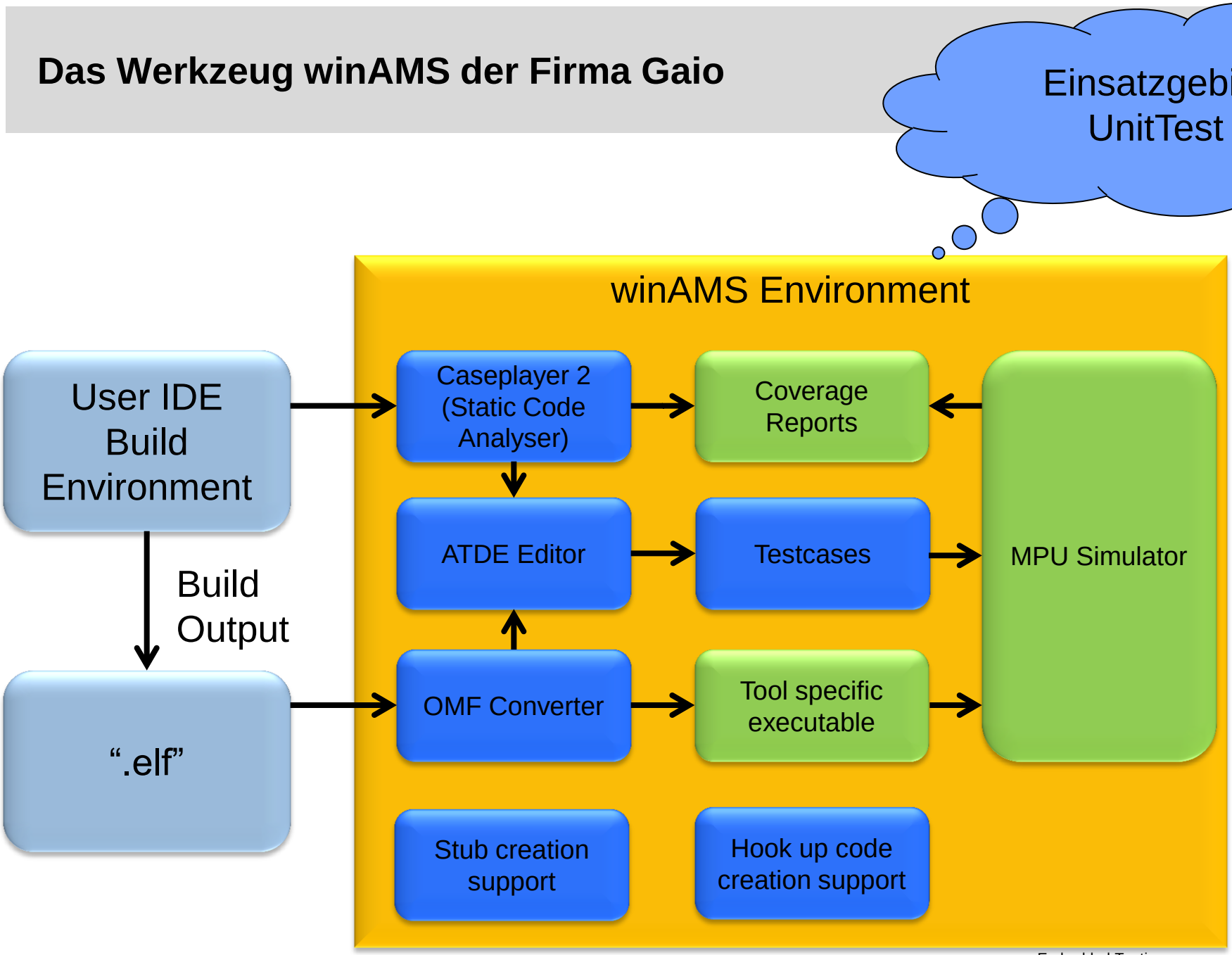
Vorteile
- Kostengünstig
- Zugriff auf alle Daten
- Einfache Coverage Messung
- Breite Toolauswahl

Nachteile
- Erfüllt nicht die ISO26262
- Es werden nur “logische Fehler” gefunden
- Beschränkung auf 100% applikativen Code

Vorteile
- Erfüllt die Vorgaben der ISO26262
- Echter “Objektcode” wird getestet

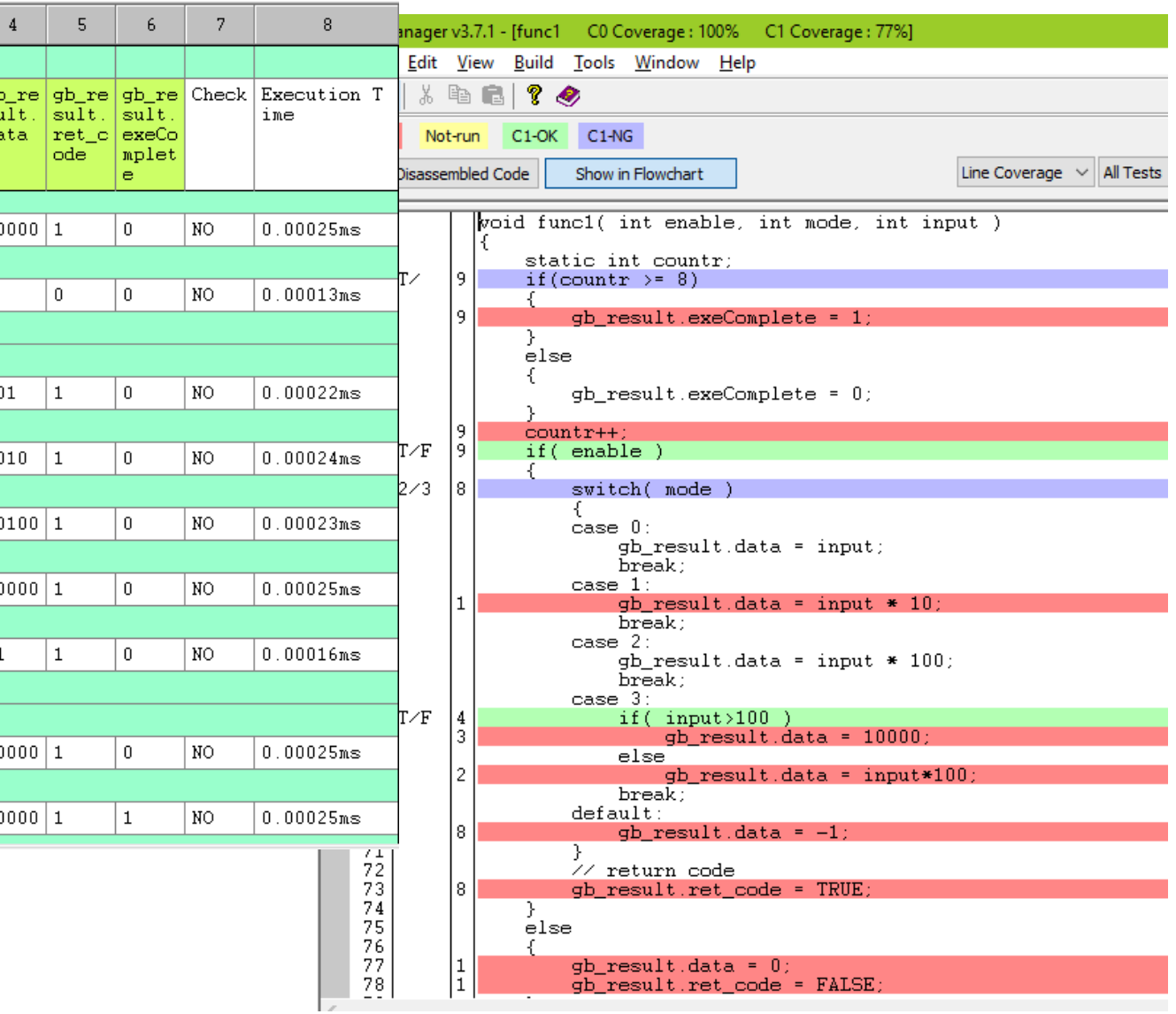
Nachteile
- Aufwändig
- Nicht alle “Zustände” erreichbar
- Coverage eingeschränkt
- Targetspezifisch
- Teure Hardware

Das Werkzeug winAMS der Firma Gaio



Das Werkzeug winAMS der Firma Gaio

	COMMENT	1	2	3	4	5	6	7	8
COMMENT									
NAME	Comment	@enable	@mode	@input	gb_result.data	gb_result.ret_code	gb_result.exeCompletee	Check	Execution Time
3		TRUE							
4		1	3	101	10000	1	0	NO	0.00025ms
5		FALSE							
6		0	3	101	0	0	0	NO	0.00013ms
7		switch (mode)							
8		case 0:							
9		1	0	101	101	1	0	NO	0.00022ms
10		case 1:							
11		1	1	101	1010	1	0	NO	0.00024ms
12		case 2:							
13		1	2	101	10100	1	0	NO	0.00023ms
14		case 3:							
15		1	3	101	10000	1	0	NO	0.00025ms
16		default:							
17		1	4	101	-1	1	0	NO	0.00016ms
18		if (input>100)							
19		TRUE							
20		1	3	101	10000	1	0	NO	0.00025ms
21		FALSE							
22		1	3	100	10000	1	1	NO	0.00025ms



Code Execution im Simulator

```
23
24  80000238  +0  int main(){
25            +1
26            +2      // Storage for the result of PxInit
27  8000023C  +3      PxError_t   PxInit_ret = PXERR_NOERROR;
28            +4
29            +5      //storage for the current CPU-ID
30  80000242  +6      CpuId_t     CoreID = SYSTEM_GetCoreId();
31            +7
32            +8      // Core 0 performs driver initialization
33  8000024C  +9      if(CoreID == cpu0)
34           +10      {
35           +11
36           +12          /*********************
37           +13           * Check the interru
38           +14           *********************
39  80000252 +15          if (!InterruptRouter_
40           +16          {
41  8000025A +17              DET_stop(AUTOCOR
42           +18          }
```

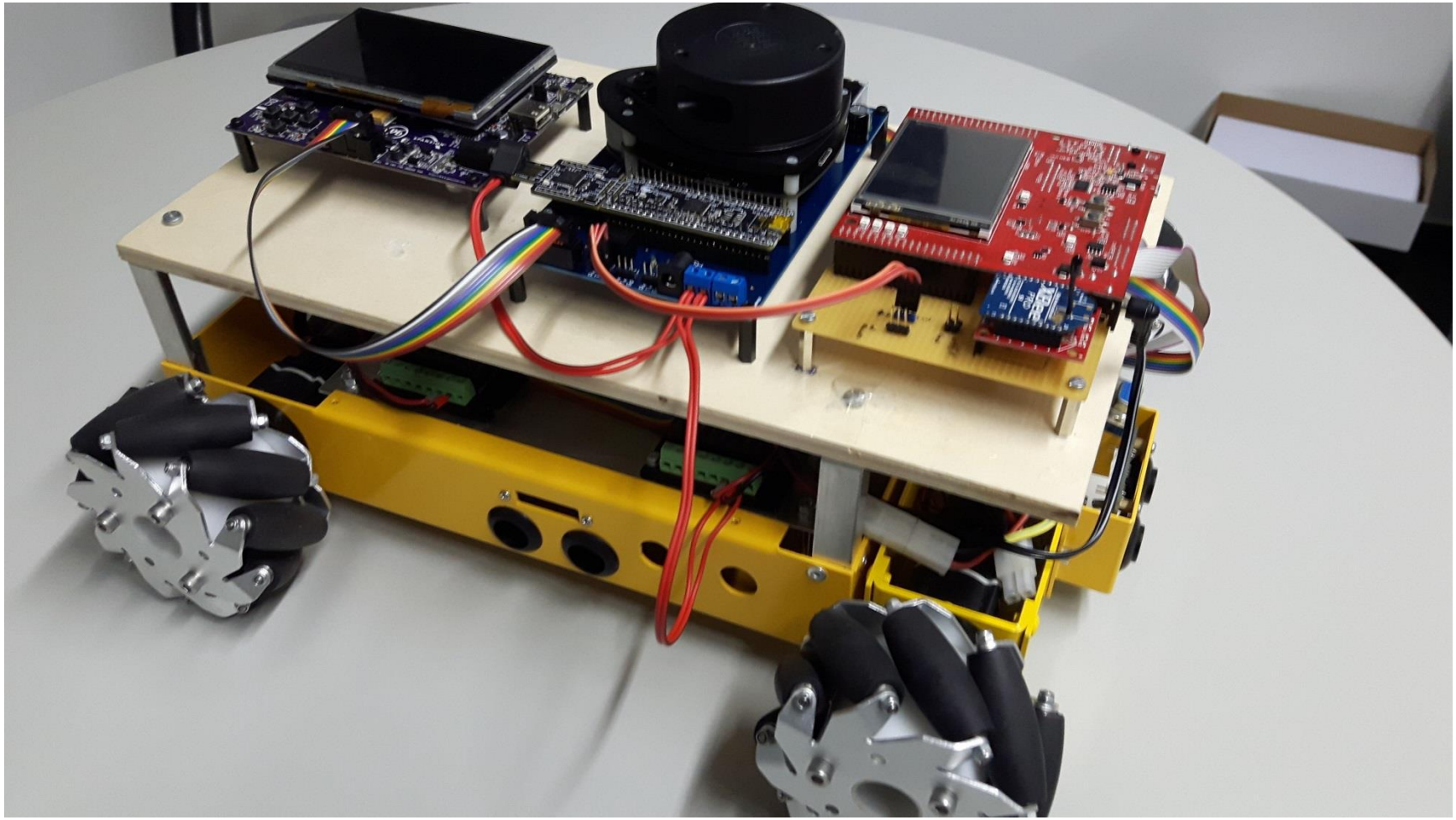
Memory1

Address 0xF0036000 | Row Size 8 | Memory Area MEM

Address	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
F0036000	00	00	00	00	00	00	00	00	....
F0036008	00	00	00	00	00	00	00	00	....
F0036010	1C	00	07	00	24	00	00	00	....$...
F0036018	20	76	01	01	02	02	02	00	v.....
F0036020	00	00	00	00	00	00	00	00	....
F0036028	00	00	00	00	00	00	00	00	....
F0036030	12	01	12	52	42	22	11	52	...RB".R
F0036038	00	00	00	00	00	00	00	00	....
F0036040	02	00	00	40	00	00	00	00	...@....
F0036048	00	00	00	00	00	00	00	00	....
F0036050	00	00	00	00	00	00	00	00	....
F0036058	00	00	00	00	00	00	00	00	....
F0036060	00	00	00	00	00	00	00	00	....
F0036068	00	00	00	00	00	00	00	00	....
F0036070	00	00	00	00	00	00	00	00	....

Watch1 | Call Stack | Classes | Register1 | Memory1

System under Test

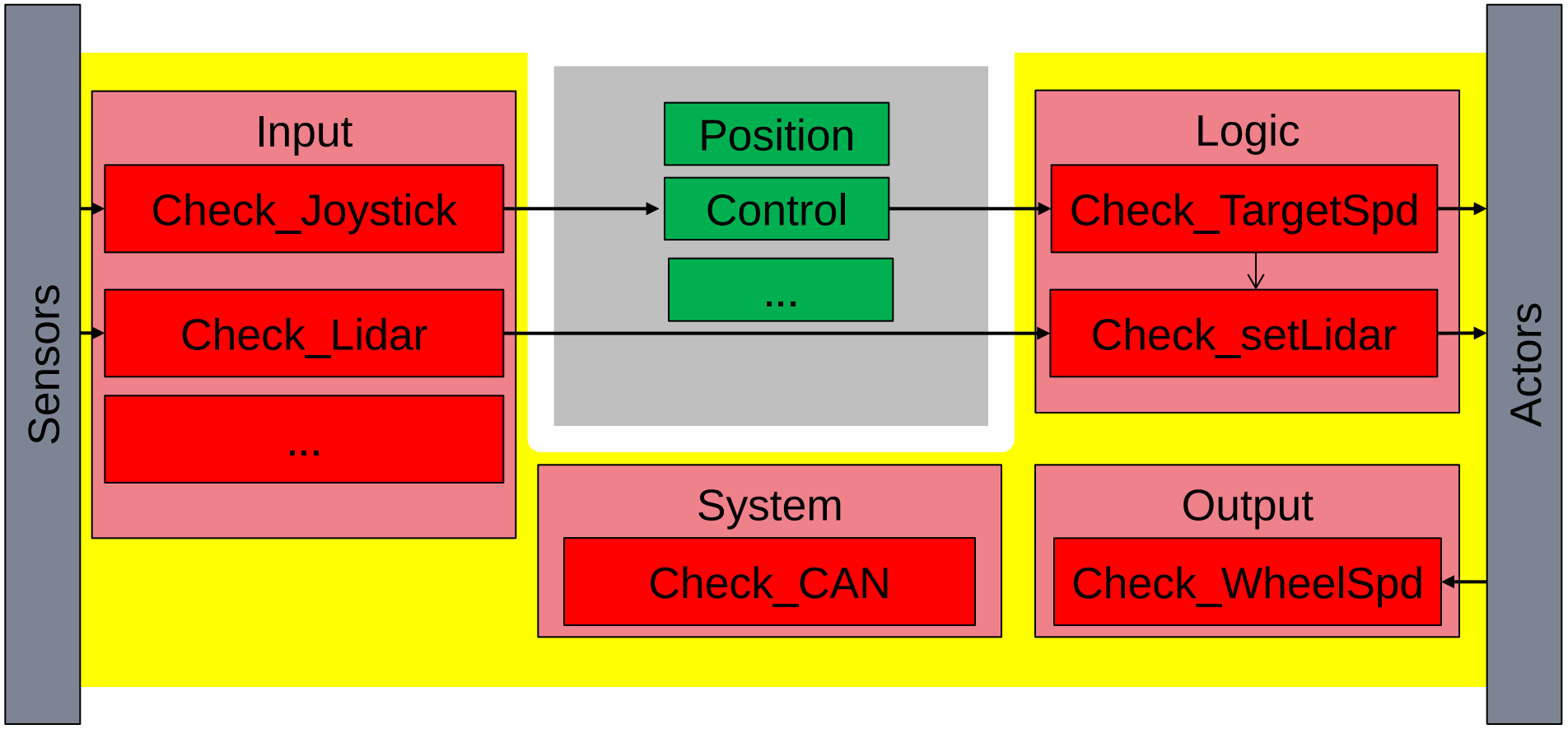


Sicherheitsfunktionen (Auszug)

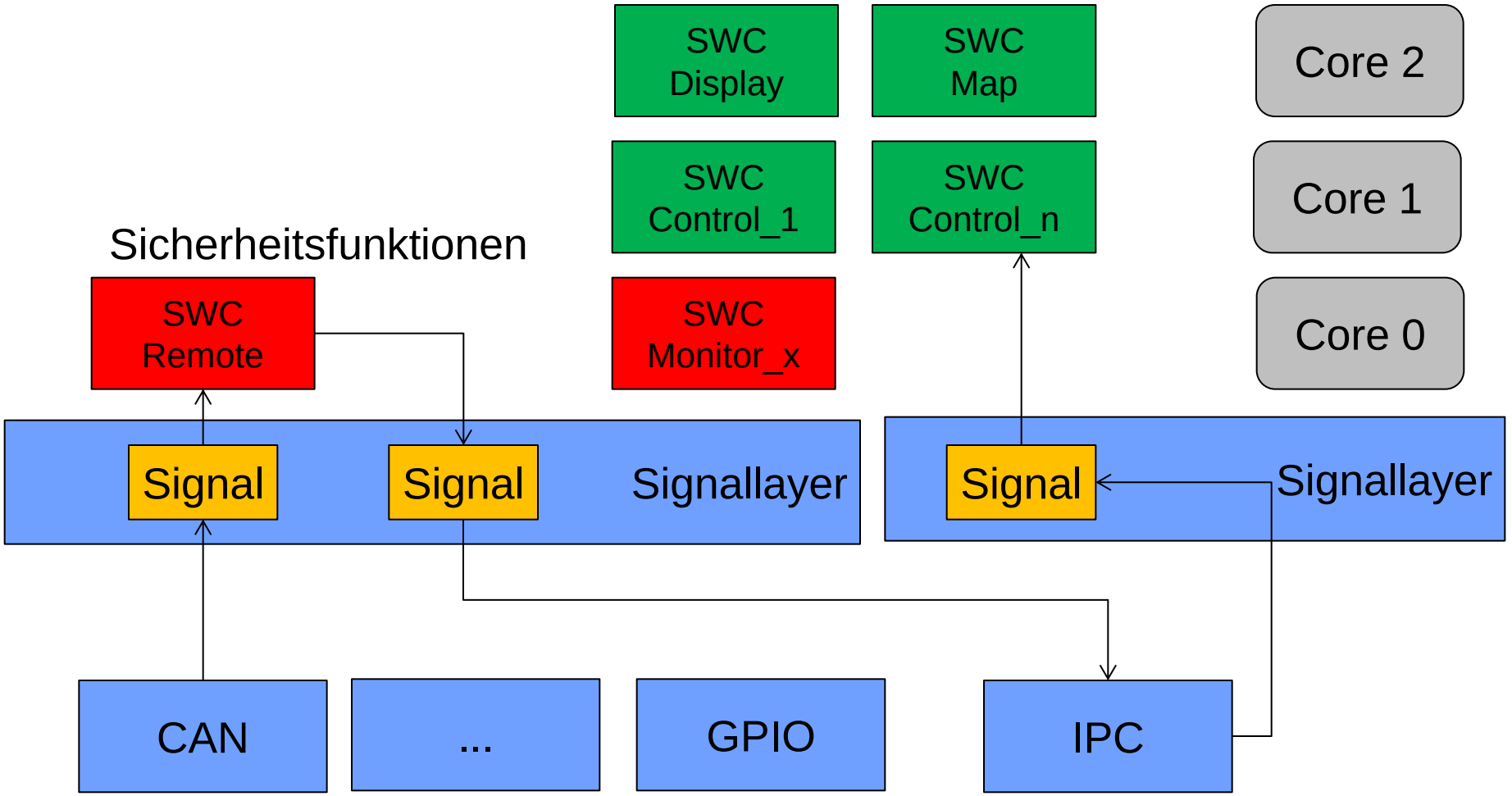
Group	Requirement	Implementation	Requested State
TargetSpeed	The targetspeed has to be updated by the control logic	• Check targetspeed age	• Safe 2
TargetSpeed	The targetspeed has to be in a valid range, considering the Error State constraints	• Check Targetspeed value before setting engine signal	• Correct targetspeed value
LIDAR	The car shall not drive against an obstacle	• Check collision signal of LIDAR • Use Watchdog / CRC to ensure updated and correct LIDAR signal	• Safe 1 (d < x) • Safe 2 (d < y) • Safe 2 (no signal)
EngineSpeed	The enginespeed may only be set to a non-zero value in the ISRUNNING state of the car	• Check car state in event driven Engine runnable (problem, if event is not fired) • And in addition in cyclic logic runnable	• Safe 2

Sicherheitskonzept

- Sicherheitsmonitorfunktionen überwachen die Applikation
- Monitorfunktionen sind kritisch und müssen daher nach SIL qualifiziert werden



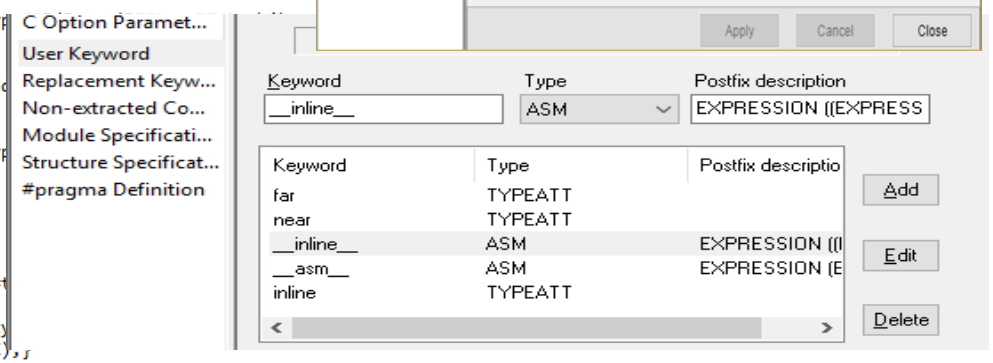
Architekturkonzept

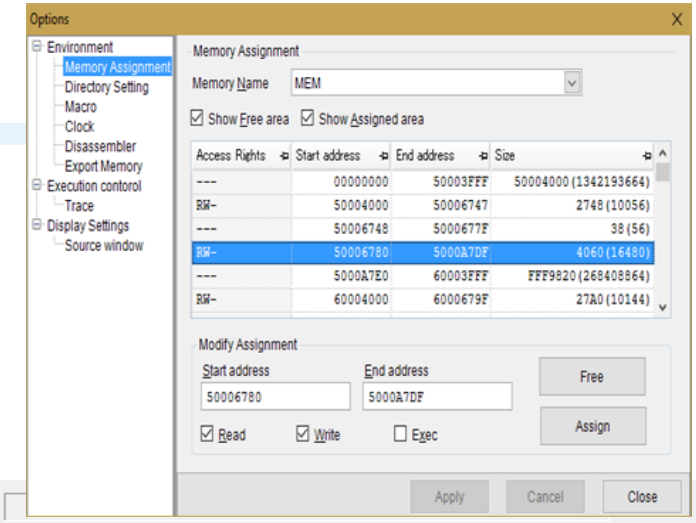


Unittest von “reinem” Applikationscode
Schritt 1 – Projekt aufsetzen

- C ist eine Standard Sprache, oder?
- Compilerspezifische Anpassung und Projektkonfiguration sehr aufwändig
- Kein C++ Support

```
351 \return the last error of the object handle !OBJ!
352 */
353 #define __PXOBJ_ACCESS_FUNCTION(type)                                   \
354 __attribute__ ((always_inline)) static inline int P ## type ## IdIsValid(P##type##_t obj)
355 {   return !((PxShort_t)(obj.id) <= 0);}        \
356                                                 \
357 __attribute__ ((always_inline)) static inline P##type##_t P##type##IdInvalidate(void)
358 {   return (P##type##_t){_PXIllegalObjId,PXERR_NOERROR}; }
359                                                 \
360 __attribute__ ((always_inline)) static inline P##type##_t P##type##IdSet(PxObjId_t id)
361 {   return (P##type##_t){id,PXERR_NOERROR}; }                                   \
362                                                 \
363 __attribute__ ((always_inline)) static inline PxObjId_t P##type##IdGet(P##type##_t obj)
364 {   return obj.id;}                             \
365                                                 \
366                                                 \
367 __attribute__ ((always_inline)) static inline PxError_t P##typ
368 {   return obj.error;}                          \
369                                                 \
370 __attribute__ ((always_inline)) static inline void P##type##Id
371 {   obj->error = err; }                         \
372                                                 \
373 __attribute__ ((always_inline)) static inline PxError_t P##typ
374 {                                               \
375     PxError_t err = obj->error;                 \
376     obj->error = PXERR_NOERROR;                 \
377     return err;                                 \
378 } \
379 __attribute__ ((always_inline)) static inline P##type##_t P##t
380 {    return (P##type##_t){id,err}; } \
381 __attribute__ ((always_inline)) static inline PxCoreId_t P##ty
382 {   return (PxInt_t)((obj.id>>PXCOREID_SHIFT) & PXCOREID_MASK);
383
```





Unittest von “reinem” Applikationscode
Testfälle generieren - Qualität der Testfälle (MC/DC)

		Input Parameters							Output Parameters			
Function Under Test	Test Case No.	so_carstate.value.stateNormal	so_targetspeed.value.vx	so_targetspeed.value.vy	so_targetspeed.value.vphi	maxSpeedParameter	so_carstate.value.speedLimit	slowSpeedParameter	return Value	so_targetspeed.value.vx	so_targetspeed.value.vy	so_targetspeed.value.vphi
SAFETYLOGIC_run_checkEngineTargetSpeed *(Manual Test Cases)*	1	0	0	0	0	0	0	1	0	0	0	0
	2	1	1	1	1	1	1	1	0	1	1	1
	3	5	-1	-1	-1	1	255	1	0	-1	-1	-1
	4	5	127	127	127	127	80	0.629921	0	0	0	0
	5	5	126	126	126	126	100	0.793651	0	0	0	0
	6	5	-128	-128	-128	-128	50	1	0	-128	-128	-128
	7	5	10	20	30	30	50	1	0	10	20	30
	8	5	30	20	10	30	20	0.666667	0	0	0	0
SAFETYLOGIC_run_checkEngineTargetSpeed *(Generated by the tool)*	1	0	-128	-128	-128		100		0	-128	-128	-128
	2	1	127	127	127		50		0	0	0	0
	3	5	-127	-127	-127		0		0	0	0	0
	4	0	126	126	126		255		0	126	126	126
	5	1	0	0	0		1		0	0	0	0
	6	5	50	50	50		100		0	50	50	50
	7	0	80	80	80		50		0	0	0	0

Gute Unterstützung z.B. mittels MC/DC.
Grenzen werden bei zu komplexen Datenstrukturen erreicht.

Unittest von “reinem” Applikationscode
Startup File

- Sicherstellung von definiertem Systemkontext

```
1
2  ;Definition of the Macro to initialize Context
3  macro/system    init_context    $p0/unsigned,   $p1/unsigned
4  assign $p0#$p1      ;Assign Context Area
5
6  define/value    end_addr = $p0 + $p1
7  define/value    count = 0
8  define/value    seg_addr = ($p0 & 0xf0000000) >> 12
9  define/value    context_addr = $p0
10
11     store/loc=4     context_addr = 0
12     define/value    context_addr = context_addr+ 64
13 loop:
14     if ( context_addr >= end_addr ) then goto next
15     define/value    link_word = seg_addr + (($p0 & 0x003fffc0) >> 6)
16     store/loc=4 context_addr = link_word
17     define/value    count = count + 1
18     define/value    context_addr = context_addr+ 64
19     goto    loop
20 next:
21 set reg pcxi = 0x0000fe00+ link_word    ;Context Start Address
22 set reg fcx  = link_word + 1        ;Context Start Address
23 define/value low_context = seg_addr + (($p0 & 0x003fffc0) >> 6) + 1
24 set reg lcx  = low_context      ;Context End Address
25 mend
26
27 ; Call to the macro to Initialize Context, with the Start Address and
28 run/macro init_context  0xd0000000,     0x4000 ;
29
30 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
31
32
33 set reg pc =  0x80000238;[your entry point here]
34 set reg PSW = 0x00000000; [your 0x18000AFFProgram Status Word]
35
36 ; Common Stack
37 set reg SP  = 0x70004100;this is just an example
38 assign  0x70004100#4000h ;this is just an example
```

```
39
40
41 ; Interrupt Stack
42 ;assign 70004100#0x4000;this is just an example
43 ;set reg ISP = 70004100  ;this is just an example
44
45 ;
46 SET CYCLE/CLOCK 300
47 ;assign 0x70004100:0x700080FF;
48 ;set register %SP=0x700080F0;
49 assign /read/exec 0x80000000:0x803E7FFF;
50 assign /read/exec 0xA0000000:0xA03E7FFF;
51 assign /read/write/exec 0x70000000:0x7001DFFF;
52 assign /read/write/exec 0x60000000:0x6003BFFF;
53 assign /read/write/exec 0x50000000:0x5003BFFF;
54 assign /read/exec 0x70100000:0x70107FFF;
55 assign /read/exec 0x60100000:0x60107FFF;
56 assign /read/exec 0x50100000:0x50107FFF;
57 assign /read/write/exec 0xC0000000:0xC0007FFF;
58 assign /read/write/exec 0x90000000:0x90007FFF;
59 assign /read/write/exec 0xB0000000:0xB0007FFF;
60 assign /read/write/exec 0xF0000000:0xFFFFFFFF;
61 assign /read/write/exec 0xD0000000:0xD0003FFF;
```

Testen von Code mit Hardwarezugriffen

- winAMS bietet die Möglichkeit Hardwareaufrufe durch Stubs zu ersetzen
- Für Register - Schreiboperationen ist dieses eine akzeptable Lösung, für Leseoperationen eher nicht

```
RC_t CAN_Frame_Read(CAN_RX_t* rx_msg){

    //Message Object buffer
    uint8_t     __MObj_i;

    //list iterator
    uint8_t     __LMOBJ_i;

    //Check if CAN had been initialized
    if(CAN__state!=service_READY)
        return RC_ERROR_INVALID_STATE;

    // Iterate trough message pending lists in static allocation MPSEL
    for(__LMOBJ_i=0;__LMOBJ_i<(DERIVATE_NUM_CAN_OBJ/sizeof(uint32_t));__LMOBJ_i++){

        //check if we have pending bits
        if(MODULE_CAN.MSID[__LMOBJ_i].B.INDEX!=0b100000){

            //get index
            __MObj_i=(__LMOBJ_i<<NIBBLE_L)|MODULE_CAN.MSID[__LMOBJ_i].B.INDEX;

            do{
                //Clear NEWDAT flag
                MODULE_CAN.MO[__MObj_i].CTR.U=BIT_1<<IFX_CAN_MO_CTR_RESNEWDAT_OFF;

                //Read message meta data from the message object
                rx_msg->frame.DLC   =   MODULE_CAN.MO[__MObj_i].FCR.B.DLC;
                rx_msg->frame.MsgID =   MODULE_CAN.MO[__MObj_i].AR.B.ID>>CAN_ACR_ID_SHIFT;

                //Read message content from the message object
                rx_msg->frame.dataL.U   = MODULE_CAN.MO[__MObj_i].DATAL.U;
                rx_msg->frame.dataH.U   = MODULE_CAN.MO[__MObj_i].DATAH.U;

                //Set pointer to message object in user-argument
                rx_msg->p_MObj  = &MODULE_CAN.MO[__MObj_i];

            //check if a new frame has been received in the meantime and data might got corrupted
            }while(MODULE_CAN.MO[__MObj_i].STAT.B.NEWDAT==TRUE||MODULE_CAN.MO[__MObj_i].STAT.B.RXUPD==TRUE);
```

```
//Clear NEWDAT flag
MODULE_CAN.MO[__MObj_i].CTR.U=BIT_1<<IFX_CAN_MO_
CTR_RESNEWDAT_OFF;
```

Kombinatorik
True / False Bedingungen
Zeitverhalten....

```
//check if a new frame has been received in the meantime and data might got corrupted
while(MODULE_CAN.MO[__MObj_i].STAT.B.NEWDAT==TRUE||MODULE_CAN.MO[__MObj_i].STAT.B.RXUPD==TRUE)
```

```
    return RC_ERROR_READ_FAILS;

}
```

Testen von Code mit OS-Aufrufen

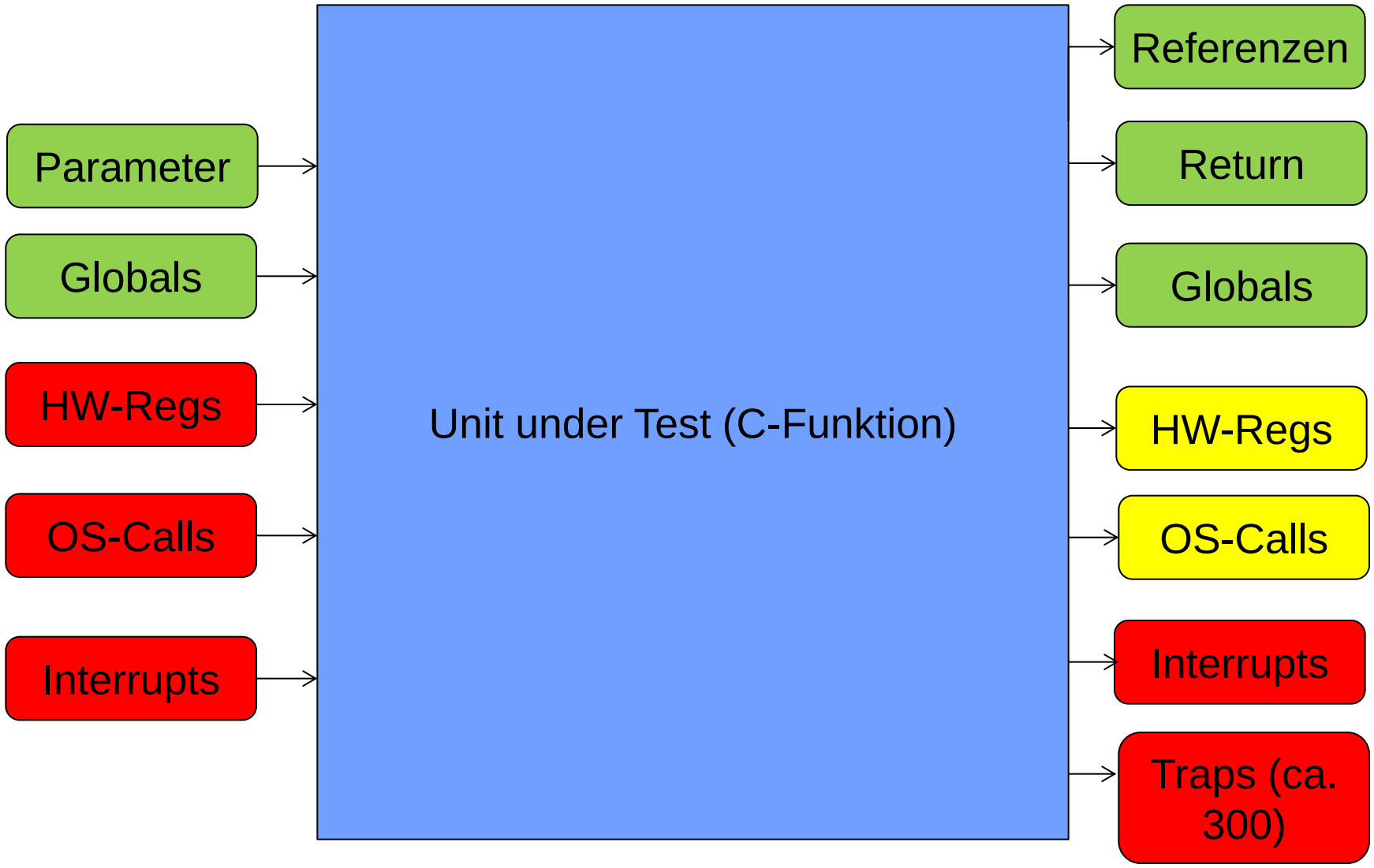
- Betriebssysteminitialisierung lässt sich im Simulator nicht ausführen
- Betriebssystemaufrufe können nicht getestet werden (Kontext kann nicht ohne weiteres hergestellt werden)
- Stubbing ist prinzipiell möglich aber mit viel Aufwand versehen
- Messaging über Coregrenzen hinweg kann nicht simuliert werden

```
static void STATE__normal_StopCar()
{
    //Fire the event for emergency brake to the safety logic task
    PxTask_t receiverTask = Task_C0_Safety_Logic_getTaskId();
    PxTaskSignalEvents(receiverTask, ev_Task_C0_Safety_Logic_3_requestStop);
}

static void STATE_normal_emergencyBrake()
{
    //Fire the event for emergency brake to the safety logic task
    PxTask_t receiverTask = Task_C0_Safety_Logic_getTaskId();
    PxTaskSignalEvents(receiverTask, ev_Task_C0_Safety_Logic_2_requestEmergencyStop);

}
```

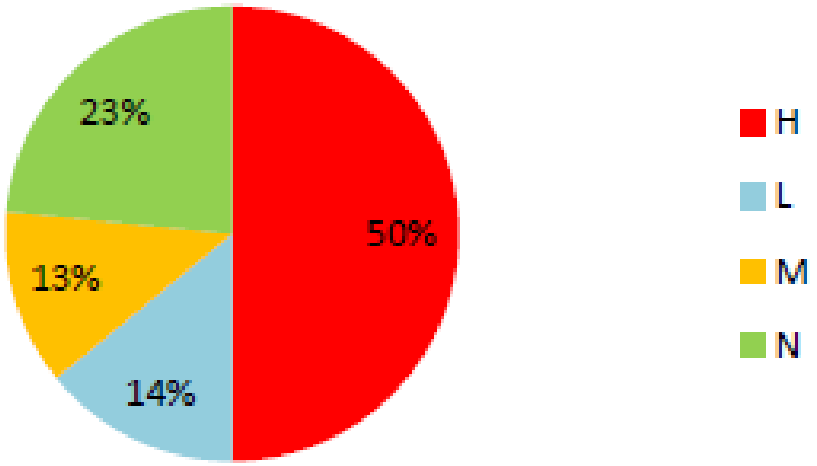
Input und Output Parameter
Eine erweiterte Definition



Spannende Fragen...

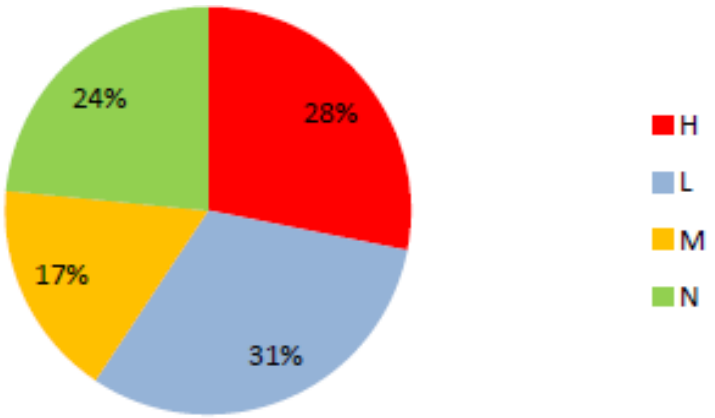
Reicht der Unittest aus?
Wie wirtschaftlich ist das Ganze?
Wie sieht eine optimale Teststrategie aus?

Wieviele Funktionen können getestet werden? Eine Aufwandsbetrachtung....

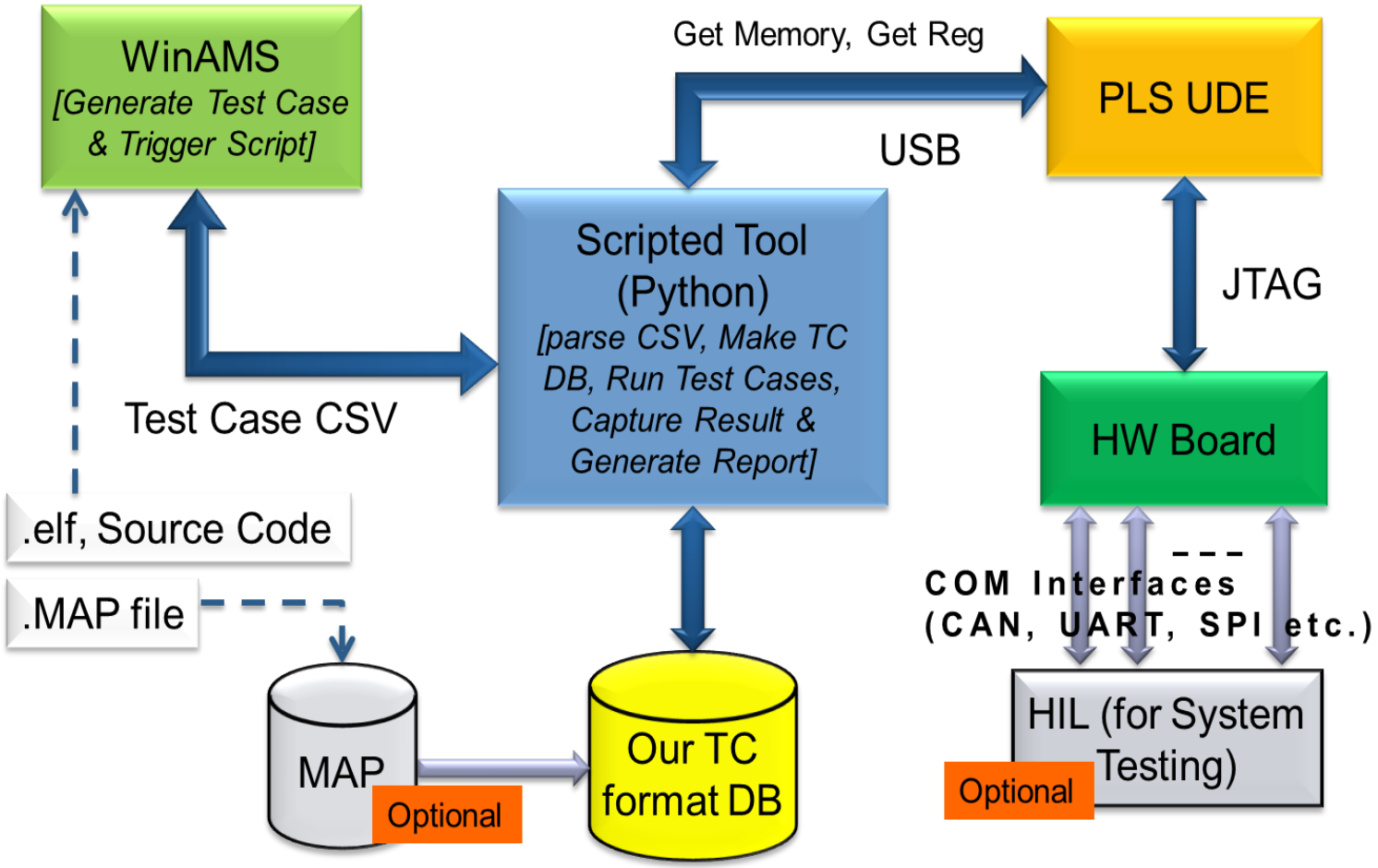


Initiales Design

Verstärkte Abstraktion über Signallayer



Erweiterter Ansatz....



Prototypische Implementierung

- Implementierung für Aurix und dem PLS Debugger ist erfolgt
- Testfälle werden von winAMS generiert

- Auf dem Target läuft ein spezieller TestTask, der im Testmodus den Aufruf der Runnables orchestriert

- Unterstützung von
  - Hardware Aufrufen
  - OS Aufrufen
  - ISR Betrieb
  - Überwachung von Timing Anforderungen
  - Traps
  - Unit, Integrations und Systemtests

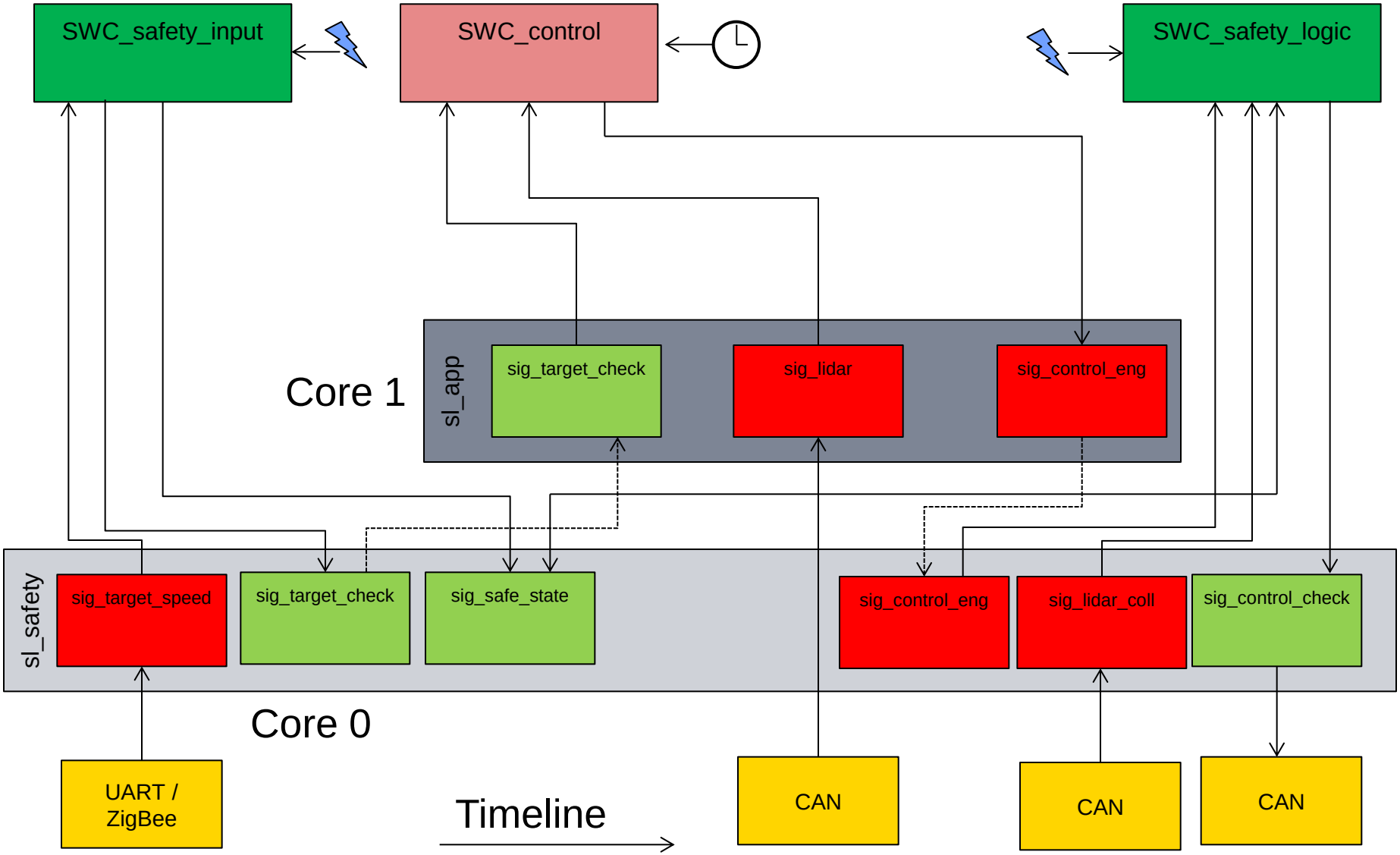
Input/Output über einfache CSV Dateien

	A	B	C	D	E
1	modi	input/write param begin	REMOTE_run_readAndCheckProtocol		21
2	#InitWheneverCall				
3	0				
4	#COMMENT	SIGNALTEST_testID	UARTPROT_remoteRxBuffer.m_pBuffer[0]	UARTPROT_remoteRxBuffer.m_pBuffer[1]	UARTPROT_remoteRxBuffer.m_pBuffer[2]
5		0	67	16	42
6		0	67	16	32
7					
8					

W	X	Y	Z
output/read param begin			
UARTPROT_remoteTxBuffer.m_pBuffer[0]	UARTPROT_remoteTxBuffer.m_pBuffer[1]	UARTPROT_remoteTxBuffer.m_pBuffer[2]	UARTPROT_remoteTxBuffer
67(67)OK	1(32)NG	16(255)NG	2(2)OK
67(67)OK	1(32)NG	16(255)NG	2(2)OK
67(67)OK	1(32)NG	16(255)NG	2(2)OK

```
/**
 * Signal Test Table (executed sequentially)
 * delayToHoldTask_in_ms is set to 200ms -> allows other core application to run before ending signalTest ta
 * PRE_RUN_FUNCTION is used to execute preRun function which triggers events to certain tasks and triggers a
 * POST_RUN_FUNCTION is mostly not used as after every signal test the target is RESET so that,
 * next signal flow is tested by bring the target to defined state.
 */
const SignalTestTable_t SignalTestTable[] = {
      //{TEST_ID,    PRE_RUN_FUNCTION,              delayToHoldTask_in_ms,  POST_RUN_FUNCTION}
        {0,          (runnablePtr)preRunSignal_0,   SIGNALTEST_DELAY,       (runnablePtr)0   },
        {1,          (runnablePtr)0,                SIGNALTEST_DELAY,       (runnablePtr)0   },
        {2,          (runnablePtr)0,                SIGNALTEST_DELAY,       (runnablePtr)0   }
};
const uint16_t SignalTestTable_size = sizeof(SignalTestTable)/sizeof(SignalTestTable_t);
```

Multicore Integrationstest
Testen von Signalflüssen



Zusammenfassung und Ausblick
“There is no silver bullet”

Zusammenfassung und Ausblick

- Simulationsbasiertes Testen ist für reinen Applikationscode eine interessante Testtechnologie
- Im Gegensatz zu reinem PC-basierten Testen erfüllt es die Anforderungen der ISO26262
- Konfiguration des Systems sollte vereinfacht werden, neue Code Standards und C++ sollten unterstützt werden

- Test von hardwarenahen Modulen und OS-Aufrufen ist wirtschaftlich nicht gut möglich
- In der Architektur muss eine strenge Trennung zwischen Applikationscode und Basissoftware realisiert sein

- Eine Erweiterung mit Anbindung an Targethardware ist relativ einfach möglich und bringt deutlichen Mehrwert

Questions / Contact

Prof. Dr.-Ing. Peter Fromm
Microcontroller und Informationstechnik

Hochschule Darmstadt - University of Applied Sciences
FB Elektrotechnik und Informationstechnik (EIT)
Birkenweg 8
64295 Darmstadt

Tel.: +49 (6151) 16-38237

Email: peter.fromm@h-da.de
Web: www.eit.h-da.de