Ostfalia Hochschule für angewandte Wissenschaften

Fakultät Informatik

Studiengang: System Engeeniering

Seminar: Modellbasierte Codegenerierung

Sommer Semester 2016/17

Experimental Report

Name, Vorname Wen Lin,
Matrikel-Nr. 70452531,
Semester 4

1. Laboraufgabe 1

1.1 Beschreibung

由于 ADC 转换是以参考电压(VREFN、VREFP) 为基准，所以基准的变化必然也会导致 ADC 结果的偏差.
In diese Aufgabe, wir nutzen A/D Wandler als Sensoren der Helligkeit zu messen. Und die Daten Im Bildschilm zeigen.

1.2 Aufgabe

zuerste sollen wird die Hardware richtig kombinieren. Dann programieren.

1.2.1 ADC.c

Am Anfange erzeugen uns eine ADC.c File.
In diese File gibt es viele Funktionen ADC zu kontrollieren.
"ADC_Init" ist eine Funktion, die ADC initialisiert.
In diese Funktion, am Anfang schalten wir Power ein. Dann konfigurieren uns 7 ADP Pins. Dann machen eine Pause. Standardmäßig ist der PIXEL-Wert Null, deshalb ist die PCLK (Portclock) für alle Peripheriegeräte 1/4 des SystemCoreClock. Danach wahlen wir ein ritig Wert bei PCLK. Nachdem Channel auswahlen, setzen wir Startbit als null bevor der BURST-Modus eingestellt werden kann. Am Ende der " ADC_Init " einstellen wir Burst-Modus und konvertieren A / D Wandler.

```
void ADC_Init (void) {
	int pclkdiv, pclk,i;
	int ADC_CLK = 1000000;

	LPC_SC->PCONP |= (1 << 12); // Power einschalten
  /* ADC Pins als Input konfigurieren */
  LPC_PINCON->PINSEL1 |= (1U<<14); /* ADC0 */
  LPC_PINCON->PINSEL1 |= (1U<<16); /* ADC1 */
  LPC_PINCON->PINSEL1 |= (1U<<18); /* ADC2 */
  LPC_PINCON->PINSEL1 |= (1U<<20); /* ADC3 */
  LPC_PINCON->PINSEL3 |= (3U<<28); /* ADC4 */
  LPC_PINCON->PINSEL3 |= (3U<<30); /* ADC5 */
  LPC_PINCON->PINSEL0 |= (2U<<6); /* ADC6 */
  LPC_PINCON->PINSEL0 |= (2U<<4); /* ADC7 */

	for (i = 0; i < 5000000; i++);


	/* By default, the PCLKSELxvalue is zero, thus, the PCLK(portclock) for all the peripherals is 1/4 of the SystemCoreClock. Bit 24~25 is for ADC */
  pclkdiv= (LPC_SC->PCLKSEL0 >> 24) & 0x03;
  switch(pclkdiv) {
```

```
    case 0x00:
    default:
    pclk= SystemCoreClock/ 4;
    break;
    case 0x01:
    pclk= SystemCoreClock;
    break;
    case 0x02:
    pclk= SystemCoreClock/ 2;
    break;
    case 0x03:
    pclk= SystemCoreClock/ 8;
    break;
}
    LPC_ADC->ADCR = (0x01 << 0) | /* SEL=1,select channel 0~7 on ADC0 */
    ((pclk/ ADC_CLK -1) << 8) | (0 << 16) |/* BURST = 0, softwarecontrolled*/
    (0 << 17) | /* CLKS = 0, 11 clocks/10 bits */
    (1 << 21) | /* PDN = 1, normal operation*/
    (0 << 24) | /* START = 0 A/D conversion stops */
    (0 << 27); /* EDGE = 0 falling,triggerA/D */
    /* Start bits need to be zero before BURST mode can be set. */
    if (LPC_ADC->ADCR & (0x7 << 24)) {
    LPC_ADC->ADCR &= ~(0x7 << 24);
    }
    LPC_ADC->ADCR &= ~0xFF;
    /* Read all channels, 0 through 7. */
    LPC_ADC->ADCR |= 0xFF;
    LPC_ADC->ADCR |= (0x1 << 16); /* Set burst mode and start A/D convert */
}
```

" ADC_StartCnv" kann AD-Konversion starten.

" ADC_StopCnv" stopen der AD-Koncersion.

" ADC_GetCnv" bekommen Konvertiert AD-Wert.

Nachdem Konvertierungsende bekommen wir die Wert der ADC. Durch verschiedene Wert bei unterschidlich ADC beurteilen es die verschiedenen Richtungen zu lenken.

```
    while (!(LPC_ADC->ADGDR & (1UL<<31))); /* Wait for Conversion end */
       ADCValue[0] = (LPC_ADC->ADDR0 >> 4) & 0xFFF;
    ADCValue[1] = (LPC_ADC->ADDR1 >> 4) & 0xFFF;
    ADCValue[2] = (LPC_ADC->ADDR2 >> 4) & 0xFFF;
    ADCValue[3] = (LPC_ADC->ADDR3 >> 4) & 0xFFF;
    ADCValue[4] = (LPC_ADC->ADDR5 >> 4) & 0xFFF;
//fahren mit hammerkopf
    if(ADCValue[2] < 0x0250) {
    adwert= 0; // mitte
    } else if (ADCValue[1] < 0x0250) {
```

```
   adwert= 1; // leicht rechts
   } else if (ADCValue[3] < 0x0250) {
   adwert= 2; //leicht links
   } else if (ADCValue[0] < 0x0250) {
   adwert= 3; //stark rechts
   } else if (ADCValue[4] < 0x0250) {
   adwert= 4; //stark links
   }
      return (adwert);
}
" ADC_IRQHandler" Wird ausgeführt, wenn die A / D-Konvertierung erfolgt ist
```

Wenn Wir fertig mit ADC machen. Sollen wir die Anzeige der Werte des ADC in Bildschilm programieren. Das sollen wir in main Funktion machen. Die main funktion hier heißt Example3.c.

1.2.2 Example3.c

In example3.c sollen wir "ADC_Init()" aufrufen, damit das Programm A/D-Converter Initialisieren wird. Dann drucken notwendig Satz.

```
ADC_Init();                              /* A/D-Converter Initialisation */


   GLCD_Clear    (Red);
   GLCD_SetTextColor (White);
   GLCD_SetBackColor (Red);
   GLCD_DisplayString (1, 1, 1, "Example 2");
```

Im eine pausenlos While-Schleife druken wir die Count. Aufrufen ADC_GetCnv(ADCValue). Bekommen die Wert judes ADC und legen sie in passenden Platz. Danach drucken die Werte juder ADC.

```
while (1) {                              /* Loop forever                    */

   if (j++>10000000)
      {
      j = 0;
         i &= 0x00FF;
         LED_Out (i++);
         sprintf (str,"Count = %i",i);
         GLCD_DisplayString (2, 1, 1, str);

         advalue = ADC_GetCnv(ADCValue);
         sprintf (str,"A/D = %4.4i",ADCValue[0]);
```

```
            GLCD_DisplayString (3, 1, 1, str);
            sprintf (str,"A/D = %4.4i",ADCValue[1]);
            GLCD_DisplayString (4, 1, 1, str);
            sprintf (str,"A/D = %4.4i",ADCValue[2]);
            GLCD_DisplayString (5, 1, 1, str);
            sprintf (str,"A/D = %4.4i",ADCValue[3]);
            GLCD_DisplayString (6, 1, 1, str);
            sprintf (str,"A/D = %4.4i",ADCValue[4]);
      GLCD_DisplayString (7, 1, 1, str);
      }
  }
```

2. PWM –Pulsweitenmodulation

2.1 Beschreibung

2.2 Aufgabe

Am Anfangen erzeugen wir PWM.c, um PWM zu kontrollieren. In Diese Methode gibt es zwei Funktionen "PWM_Init" und "PWM_set".
"PWM_Init()"initialisiert PWM. Einstellen Anfangswerte aller benötigten Variablen.

```
void PWM_Init(){
   LPC_PINCON->PINSEL4 = 0x0000155A; /* setGPIOs forall PWM pinson PWM */
   LPC_PWM1->TCR = 0x00000002; /* TCR_RESET-Counter Reset*/
   LPC_PWM1->PR = 0x00;/* countfrequency:Fpclk*/
   LPC_PWM1->MCR = (1 << 0); /* PWMMR0I-interrupton PWMMR0, reseton PWMMR0,
resetTC ifPWM matches*/
   LPC_PWM1->LER = (1 << 0) | (1 << 3) | (1 << 4); /* all PWM latchenabled*/
}
```

"PWM_set()" definieren die Verhalten des PWM. Zuerst setzen PWM-Zyklus, Wenn die Wert des Kanel ist null, funktionieren der Motor, Wenn die Wert des Kanel ist eins funktionieren die Lenkung. Und zählen aktiviert PWM.

```
void PWM_set( int channel, int offset) {
   LPC_PWM1->MR0 = 500000; /* setPWM cycle*/
   if(0 == channel)
            LPC_PWM1->MR3 = offset; // Motor
   if(1 == channel)
   LPC_PWM1->MR4 = offset; // Lenkung
```

```
   LPC_PWM1->LER = (1 << 0) | (1 << 3) | (1 << 4); // LatchEnableRegister
   LPC_PWM1->PCR = (1 << 11) | (1 << 12);// PWM Control Register
   LPC_PWM1->TCR = 0x00000001 | 0x00000008; // TimerControl Register: Counter
enablePWM enable
}
```

3. Laboraufgabe 3

2.1 Beschreibung

2.2 Aufgabe

2.2.1 Merapi Class Diagramm

2.2.1.1 Die Implementierung in PWM-initialize()

```
LPC_PINCON->PINSEL4 = 0x0000155A; /* setGPIOs forall PWM pinson PWM */
LPC_PWM1->TCR = 0x00000002; /* TCR_RESET-Counter Reset*/
LPC_PWM1->PR = 0x00;/* countfrequency:Fpclk*/
LPC_PWM1->MCR = (1 << 0); /* PWMMR0I-interrupton PWMMR0, reseton PWMMR0, resetTC
ifPWM matches*/
LPC_PWM1->LER = (1 << 0) | (1 << 3) | (1 << 4); /* all PWM latchenabled*/
```

Beschreibung: Im diese Funktion wird die PWM initialisiert. Alle Variable über PWM werden zugeordnent.

2.2.1.2 Die Implementierung in PWM-set(in offset: Integer)

```
LPC_PWM1->MR0 = 500000; /* setPWM cycle*/
if(0 == channel)
LPC_PWM1->MR3 = offset; // Motor
if(1 == channel)
LPC_PWM1->MR4 = offset; // Lenkung
LPC_PWM1->LER = (1 << 0) | (1 << 3) | (1 << 4); // LatchEnableRegister
LPC_PWM1->PCR = (1 << 11) | (1 << 12);// PWM Control Register
LPC_PWM1->TCR = 0x00000001 | 0x00000008; // TimerControl Register: Counter enablePWM
enable
```

Beschreibung: Hier definieren die Bewegung bei PWM. PWM ist eine Modulationsart, bei der

eine technische Größe zwischen zwei Werten wechselt. Dabei wird bei konstanter Frequenz der Tastgrad eines Rechteckpulses moduliert, also die Breite der ihn bildenden Impulse.Wenn hier eine Analogsignal bekommen, wird das auf Impulssignal transportieren. Hie PWM kontrollieren die Lekung des Auto wenn das Auto sraten.

2.2.1.3 Die Implementierung in Button-init()

```
LPC_GPIO2->FIODIR &= ~(1<< 10);/*PORT2.10 defined as input*/
```

Hier definition eine Button, wenn man diese Button druken, wird das Auto funktionieren

2.2.1.4 Die Implementierung in Button-wait()

```
whilr(LPC_GPIO2->FIOPIN & (1<<10));
```

Beschreibung: Nachdem Button gedruckt werden, soll das Auto ein bisschen Zeit warten, wenn das Auto klingelt, kann man weiter machen.

2.2.1.5 Die Implementierung in init()

```
intADC_CLK = 1000000; // 1MHz
LPC_SC->PCONP |= (1 << 12); // Power einschalten
/* ADC Pins als Input konfigurieren */
LPC_PINCON->PINSEL1 |= (1U<<14); /* ADC0 */
LPC_PINCON->PINSEL1 |= (1U<<16); /* ADC1 */
LPC_PINCON->PINSEL1 |= (1U<<18); /* ADC2 */
LPC_PINCON->PINSEL1 |= (1U<<20); /* ADC3 */
LPC_PINCON->PINSEL3 |= (3U<<28); /* ADC4 */
LPC_PINCON->PINSEL3 |= (3U<<30); /* ADC5 */
LPC_PINCON->PINSEL0 |= (2U<<6); /* ADC6 */
LPC_PINCON->PINSEL0 |= (2U<<4); /* ADC7 */
for (i = 0; i < 5000000; i++); // Delay..

/* By default, the PCLKSELxvalue is zero, thus, the PCLK for all the peripherals is 1/4 of the SystemCoreClock. Bit 24~25 is for ADC */
pclkdiv= (LPC_SC->PCLKSEL0 >> 24) & 0x03;
switch(pclkdiv) {
case0x00:
default:
pclk= SystemCoreClock/ 4;
break;
case0x01:
pclk= SystemCoreClock;
break;
case0x02:
```

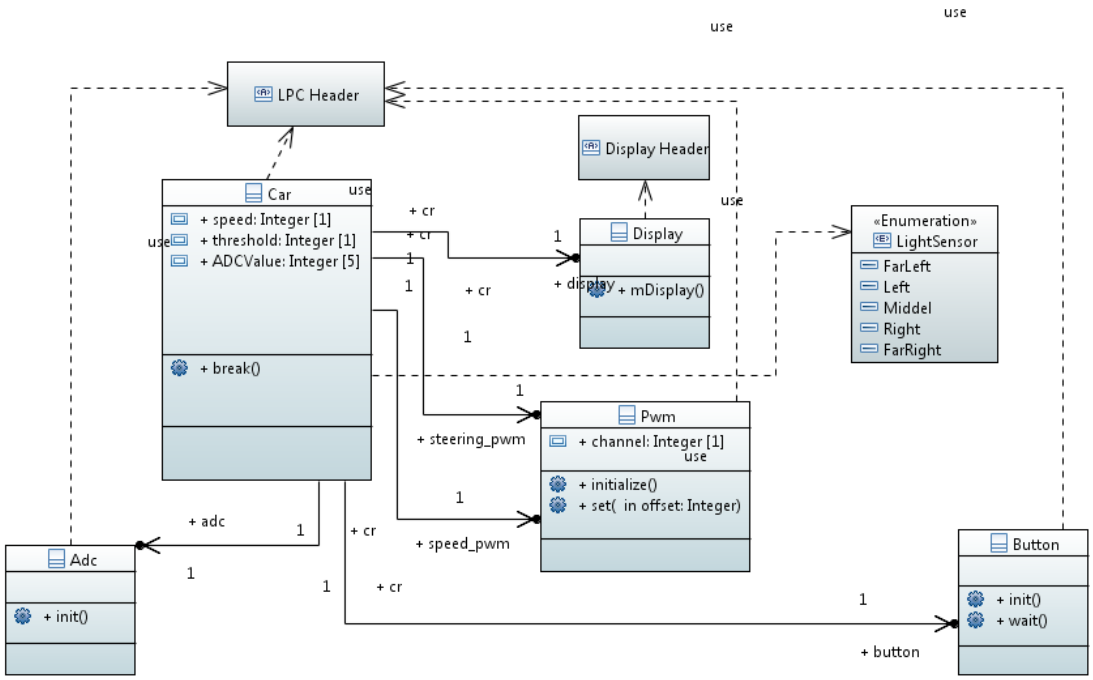
```
pclk= SystemCoreClock/ 2;
break;
case0x03:
pclk= SystemCoreClock/ 8;
break;
}

LPC_ADC->ADCR = (0x01 << 0) | /* SEL=1,select channel 0~7 on ADC0 */
((pclk/ ADC_CLK -1) << 8) | (0 << 16) |/* BURST = 0, softwarecontrolled*/
(0 << 17) | /* CLKS = 0, 11 clocks/10 bits */
(1 << 21) | /* PDN = 1, normal operation*/
(0 << 24) | /* START = 0 A/D conversion stops */
(0 << 27); /* EDGE = 0 falling,triggerA/D */
/* Start bits need to be zero before BURST mode can be set. */
if (LPC_ADC->ADCR & (0x7 << 24)) {
LPC_ADC->ADCR &= ~(0x7 << 24);
}
LPC_ADC->ADCR &= ~0xFF;
/* Read all channels, 0 through 7. */
LPC_ADC->ADCR |= 0xFF;
LPC_ADC->ADCR |= (0x1 << 16); /* Set burst mode and start A/D convert */
```

Die Implementierung in querySensors

```
me->ADCValue[0] = (LPC_ADC->ADDR0) >> 4) & 0xFFF;
me->ADCValue[1] = (LPC_ADC->ADDR1) >> 4) & 0xFFF;
me->ADCValue[2] = (LPC_ADC->ADDR2) >> 4) & 0xFFF;
me->ADCValue[3] = (LPC_ADC->ADDR3) >> 4) & 0xFFF;
me->ADCValue[4] = (LPC_ADC->ADDR5) >> 4) & 0xFFF;
```

Beschreibung: ADC wird hier initialisiert wid.
PCLK?????



2.2.2 Merapi Objekt Diagramm

Das Objektdiagramm ist ein *Strukturdiagramm*, denn es zeigt eine bestimmte Sicht auf die Struktur des modellierten Systems. Die Darstellung umfasst dabei typischerweise Ausprägungsspezifikationen von Klassen und Assoziationen.
In diese Diagramm, "Car" ist das main Methode. Und andere Methode gehören zu Car-Methode.
"Display" kontrollieren die Auszeichen der Bildschilm.
"pwm-steering" kontrollieren die Lekung des Auto.
"Button"
"pwm-speed" kontrollieren die Geschwindigkeit des Auto. Wir müssen unbdint ein passende Geschwindigkeit auswahlen. Wenn die Geschwindigkeit zu schnell ist, wird die Rades des Auto rückwärtsdrehren.
"adc" Kotrollieren die Lekung wenn das Auto fahren. Sensoren kann die Licht messen. Durch A/D Wandle, kann man das Signal bekommen und bearbeiten.

2.2.3 Merapi State Diagramm

2.2.3.1 Die Implementierung in init()

```
int i;
SystemInit();
Button_init(me->);
ADC_init(me->adc）;
Pwm_initialize(me->speed_pwm);
Pwm_set(me->speed_pwm, 30000);
Button_wait(me->button);
Pwm_set(me->speed_pwm, 27500);
```

Beschreibung: Hier definieren alle Bewegung des Autos. Am anfangen wird Button aufgerufen wird, Dann funktionieren ADC. Setzen die Geschwindigkeit des Autos.

2.2.3.2 Die Implementierung der Verbindung zwischen querySensors und FarLeft

```
me->ADCValue[LightSensor_FarLeft] > me -> threshold
```

2.2.3.3 Die Implementierung der Verbindung zwischen querySensors und Left

```
me->ADCValue[LightSensor_Left] > me -> threshold
```

2.2.3.4 Die Implementierung der Verbindung zwischen querySensors und Middle

```
me->ADCValue[LightSensor_Middle] > me -> threshold
```

2.2.3.5 Die Implementierung der Verbindung zwischen querySensors und Right

```
me->ADCValue[LightSensor_Right] > me -> threshold
```

2.2.3.6 Die Implementierung der Verbindung zwischen querySensors und FarRight

```
me->ADCValue[LightSensor_FarRight] > me -> threshold
```

Beschreibung: Es gibt 5 Lichtsensoren. Jeder Sensoren arbeiten allein. Wenn die Helligkeit erfüllen die Anforderung, Wird passende Bewegung machen.

2.2.3.7 Die Implementierung in FatLeft

```
Pwm_set(me->steering_Pwm, 28000);
```

2.2.3.8 Die Implementierung in FatLeft

```
Pwm_set(me->steering_Pwm, 34000);
```

2.2.3.9 Die Implementierung in FatLeft

Pwm_set(me->steering_Pwm, 38000);

2.2.3.10 Die Implementierung in FatLeft

Pwm_set(me->steering_Pwm, 40000);

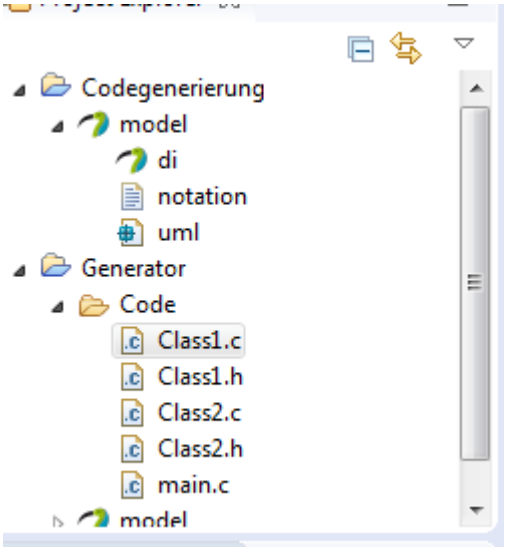
2.2.3.11 Die Implementierung in FatLeft

Pwm_set(me->steering_Pwm, 42000);

Beschreibung: Wenn die passende Sensoren aktivieren, kotrollieren das Auto ihre Lekung. Danach wird das Auto entlang die weiße Linie fahren.

4. ??????

Wenn man ein ganz einfach UML-Diagramm wie folgende Bilden zeigen. Es gibt zwei Classe: Class1 und Class2. Wenn wir das auf Code compilieren, wird es fuer jeder Class eine Class.c und Class.h erzeugen.



Wenn A->B, alle Methode in B-Class werden auch in A-Class decliered. Alle Variable in b.h werden auch in a.h zeigen.

In Jeder Class.c gibt es eine "#include "Class.h". Das erzeugt automatisch.
Class.h nur declarieren die Methode. Class.c hat spezifische Methode.

In eine Class.c Datei wird es automatisch erzeugen alle eng verwandte Typen. Wenn die Class2 ist die submethode für Class1 wird alle diese Variable auch in Class1 declariert.

```
/* ---------------------------------------------------------------------- *
 *                  shortened names for closely related Types             *
 * ---------------------------------------------------------------------- */

#define Class1              RootElement_Class1
#define Cleanup(ME)         RootElement_Class1_Cleanup(ME)
#define Destructor(ME)      RootElement_Class1_Destructor(ME)
#define Init(ME)            RootElement_Class1_Init(ME)
#define New()               RootElement_Class1_New()
#define Operation1(ME)      RootElement_Class1_Operation1(ME)

#define Class2                    RootElement_Class2
#define Class2_break(ME)          RootElement_Class2_break(ME)
#define Class2_Cleanup(ME)        RootElement_Class2_Cleanup(ME)
#define Class2_Destructor(ME)     RootElement_Class2_Destructor(ME)
#define Class2_Init(ME)           RootElement_Class2_Init(ME)
#define Class2_New()              RootElement_Class2_New()
```

model.di | model.di | Class1.c | main.c | Class1.h | Class2.h | Class2.c

```
  Code Generator    : ModelXChanger UML->C Generator

#include "Class2.h"    // this class' header
#include <stdlib.h>    // default include

/* ---------------------------------------------------------------------- *
 *                  shortened names for closely related Types             *
 * ---------------------------------------------------------------------- */

#define Class2             RootElement_Class2
#define break(ME)          RootElement_Class2_break(ME)
#define Cleanup(ME)        RootElement_Class2_Cleanup(ME)
#define Destructor(ME)     RootElement_Class2_Destructor(ME)
#define Init(ME)           RootElement_Class2_Init(ME)
#define New()              RootElement_Class2_New()

/* ---------------------------------------------------------------------- *
 *                          Class management functions                    *
 * ---------------------------------------------------------------------- */
```

Die software wird die Funktionen "void RootElement_Class_Inti()" , "RootElement_Class* Rootelement_Class_New()" , "void RootElement_Class_Cleanup()" und "void RootElement_Class_Destructor()" automatische erzeugen. Alle Class müssen diese vier Funktionen haben.

Wenn man die Operation in eine Class hinzugüt, wird die ganzen Methode in Class.c zeigen.

```
void RootElement_Class2_break(RootElement_Class2* const me) {
    /*## Behavior RootElement::Class2::break_method */
    break;
}
```

Wenn Class1 -> Class2 dann wird die Funktionen in Class2 auch in Class1.h declariert werden.

```
void RootElement_Class1_Operation1(RootElement_Class1* const me);

void RootElement_Class2_break(RootElement_Class2* const me);
```

5. ??????

5.1Beschreibung

diese übnung sollen wir die vorliegende Code kolliegieren. Code läuft in JUnit. Es gibt noch einige Fehle. JUnit ist eine Einheit des Test-Framework für Java-Sprache. In JUnit suchen wir, wo und wie die Fehler ist. Danach kolligieren wir die Fehler.
Wir sollen zwei Teile Machen. Eine ist über Parameter, andere ist über Operration.

5.2 Lösungen

5.2.1 kolliergieren über Parameter

Es gibt schon Parameter in umlElement d.h alle Methode über parameter wird aufgeruft.

```
switch umlElement {
            Activity:
                return generateAktivity(umlElement)
            Class:
                return generateClass(umlElement)
            Parameter:
                return generateParameter(umlElement)
            default:
                return ""
        }



        if(ParameterDirectionKind.INOUT_LITERAL == umlParameter.direction &&
            umlParameter.type instanceof PrimitiveType
        ){
            return '''«generateType(umlParameter.type)»* «umlParameter.name»'''
        }
@Test def testPrimitiveInOutParameter() {
        val parameter = createParameter => [
            name = "primitiveInOutParameter"
            direction = ParameterDirectionKind.INOUT_LITERAL
            type = createPrimitiveType => [name = "uint32"]
        ]
        val code = (new Uml2C).generateCode(parameter)
        Assert.assertEquals("uint32* primitiveInOutParameter", code)
    }

else if( ParameterDirectionKind.RETURN_LITERAL == umlParameter.direction){
            return '''«generateType(umlParameter.type)»'''
```

```
        }
val parameter = createParameter => [
            name = "primitiveReturnParameter"
            direction = ParameterDirectionKind.RETURN_LITERAL
            type = createPrimitiveType => [name = "uint32"]
        ]
        val code = (new Uml2C).generateCode(parameter)
        Assert.assertEquals("uint32", code)
    }

else if(ParameterDirectionKind.OUT_LITERAL == umlParameter.direction){
            return '''«generateType(umlParameter.type)»* «umlParameter.name»'''
        }
@Test def testComplexOutParameter() {
        val parameter = createParameter => [
            name = "complexOutParameter"
            direction = ParameterDirectionKind.OUT_LITERAL
            type = createClass => [name = "ComplexType"]
        ]

        val code = (new Uml2C).generateCode(parameter)

        Assert.assertEquals("ComplexType** complexOutParameter", code)
    }


else if(ParameterEffectKind.READ_LITERAL == umlParameter.effect ){
            return '''«generateType(umlParameter.type)» const «umlParameter.name»'''
        }

@Test def testReadParameter() {
        val parameter = createParameter => [
            name = "readParameter"
            effect = ParameterEffectKind.READ_LITERAL
            type = createClass => [name = "ComplexType"]
        ]

        val code = (new Uml2C).generateCode(parameter)

        Assert.assertEquals("ComplexType* const readParameter", code)
    }
```

5.2.1 kolliergieren über Operation

```
switch umlElement {
```

```
            Activity:
                return generateAktivity(umlElement)
            Class:
                return generateClass(umlElement)
            Parameter:
                return generateParameter(umlElement)
            Operation:
                return generateOperation(umlElement)
            default:
                return ""
        }
```

```
        if(operation.name == "oneParameterOperation"){
        ){
            return '''void (TestClass* const me, uint32 param1);'''
        }

    @Test def testOneParameterOperation() {
        val operation = createOperation => [
            name = "oneParameterOperation"
        ......
        val code = (new Uml2C).generateCode(operation)
        Assert.assertEquals("void oneParameterOperation(TestClass* const me, uint32
param1);", code)
    }
```

```
else if(operation.name == "twoParameterOperation" ){
            return '''void twoParameterOperation(TestClass* const me, uint32 param1,
uint8 param2);'''
        }

@Test def testTwoParameterOperation() {
        val operation = createOperation => [
            name = "twoParameterOperation"
        ......
        val code = (new Uml2C).generateCode(operation)
```

```
        Assert.assertEquals("void twoParameterOperation(TestClass* const me, uint32
param1, uint8 param2);", code)
    }



else if(operation.name == "returningOperation" ){
            return '''uint32 returningOperation(TestClass* const me);'''
        }

@Test def testReturningOperation() {
        val operation = createOperation => [
            name = "returningOperation"
        ......
        val code = (new Uml2C).generateCode(operation)
        Assert.assertEquals("uint32 returningOperation(TestClass* const me);", code)
    }



else if(operation.name == "implementedOperation" ){
            return '''
            void implementedOperation(TestClass* const me) {
                /* hier kaennte Ihre Werbung stehen */
            }
        '''
        }

@Test def testImplementedOperation() {
        val operation = createOperation => [
            name = "implementedOperation"
        ......
        Assert.assertEquals(
            '''
            void implementedOperation(TestClass* const me) {
                /* hier kaennte Ihre Werbung stehen */
            }
        '''.toString, code)
    }


else if(operation.name == "privateOperation" ){
            return '''static void privateOperation(TestClass* const me);'''
```

```
}

@Test def testPrivateOperation() {
        val operation = createOperation => [
            name = "privateOperation"
        ......
        Assert.assertEquals("static void privateOperation(TestClass* const me);", code)
    }



else if(operation.name == "staticOperation" ){
            return '''void staticOperation(uint32 param1);'''
        }

@Test def testStaticOperation() {
        val operation = createOperation => [
            name = "staticOperation"
        ......
        Assert.assertEquals("void staticOperation(uint32 param1);", code)
    }


else if
        (operation.name == "queryOperation" ){
            return '''void queryOperation(const TestClass* const me);'''
        }
@Test def testQueryOperation() {
        val operation = createOperation => [
            name = "queryOperation"
        ......
        Assert.assertEquals("void queryOperation(const TestClass* const me);", code)
    }
```