

**Ostfalia Hochschule für Angewandte Wissenschaften**  
**Fakultät Informatik**

# **Modellbasierte Codegenerierung**

**Experimental Report**

**Name, Vorname : Ndeudje Cyprez Renault , Wen Lin**

**Matrikel-Nr: 70448500 , 70452531**

**Dozent: Günter Kircher**



**SS 2017**

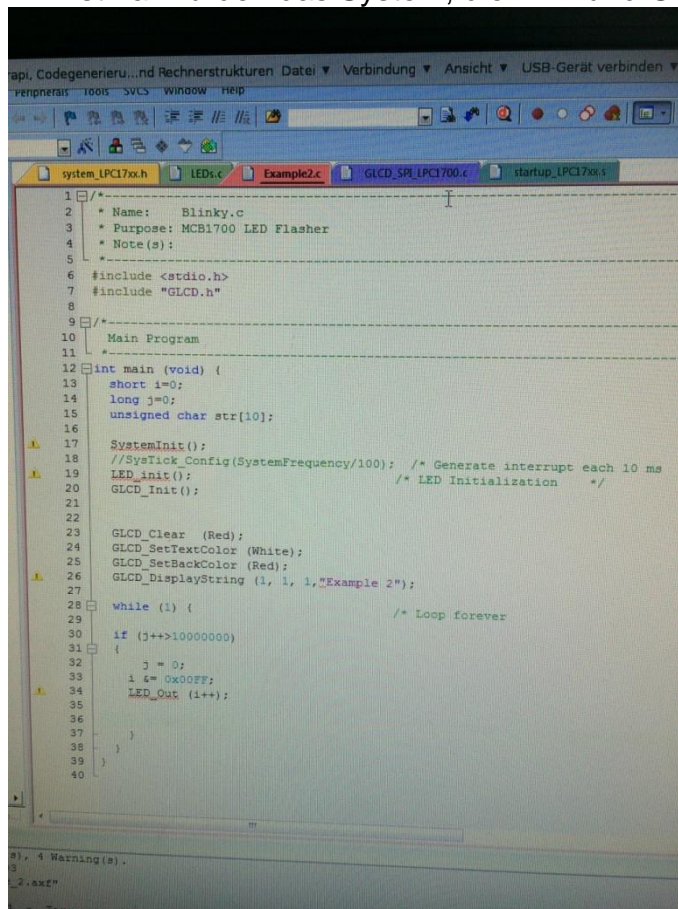
## Inhalt

1. Example 2 .....	2
2. Example 3 .....	2
3. Example 4 .....	6
4. MerapiUML.....	9
4.1. Klassendiagramm .....	9
4.2. Beschreibung Einzelne Methoden in Verschiedene Klassen.....	10
4.3. Object Diagram dient zur Struktur des Programms .....	12
4.4. State Diagram steht das Verfahren von dem gesamten Programm .....	13
5. Verhältnis von Klassen.....	17
6. XTEND Programmiersprache .....	21
6.1. TestOperation.....	21
6.2. TestParameter .....	24
7. Aktivitäts- Diagramm.....	25

# 1.Example 2

Ziel des Aufgabe war die Anzeige Verschiedene Daten auf dem Eva-Board Bildschirm und sich auch mit dem C-Programm zu gewöhnen

→ Erstmal wurden das System, die LED und GLCD initialisiert



```
1  /*
2  * Name: Blinky.c
3  * Purpose: MCB1700 LED Flasher
4  * Note(s):
5  */
6  #include <stdio.h>
7  #include "GLCD.h"
8
9  /*
10 * Main Program
11 */
12 int main (void) {
13     short i=0;
14     long j=0;
15     unsigned char str[10];
16
17     SystemInit();
18     //SysTick_Config(SystemFrequency/100); /* Generate interrupt each 10 ms
19     LED_init(); /* LED Initialization */
20     GLCD_Init();
21
22
23     GLCD_Clear (Red);
24     GLCD_SetTextColor (White);
25     GLCD_SetBackColor (Red);
26     GLCD_DisplayString (1, 1, 1, "Example 2");
27
28     while (1) { /* Loop forever
29
30         if (j++>100000000)
31         {
32             j = 0;
33             i ^= 0x00FF;
34             LED_Out (i++);
35
36         }
37     }
38 }
39
40
```

→ Dann wurden die Farben konfiguriert einmal für den Hintergrund des Eva-Board Bildschirms(siehe Zeile 23,Abbildung 1) , dann für die Daten , die angezeigt werden(Zeile 24,Abbildung 1) , zuletzt für „Example 2“ (Zeile 26,Abbildung 1)

Abbildung 1.Example2.c

## 2. Example 3

Ziel dieser Aufgabe war die Aktivation des Auto-Sensors durch erweiterung des C-Programms

```
22 uint8_t AD_done = 0; /* AD conversion done flag
23
24 /*
25  Function that initializes ADC
26  */
27 void ADC_Init (void) {
28     int pclkdiv, pclk,i;
29     int ADC_CLK = 1000000;
30
31     LPC_SC->PCONP |= (1 << 12); // Power einschalten
32     /* ADC Pins als Input konfigurieren */
33     LPC_PINCON->PINSEL1 |= (1U<<14); /* ADC0 */
34     LPC_PINCON->PINSEL1 |= (1U<<16); /* ADC1 */
35     LPC_PINCON->PINSEL1 |= (1U<<18); /* ADC2 */
36     LPC_PINCON->PINSEL1 |= (1U<<20); /* ADC3 */
37     LPC_PINCON->PINSEL3 |= (3U<<28); /* ADC4 */
38     LPC_PINCON->PINSEL3 |= (3U<<30); /* ADC5 */
39     LPC_PINCON->PINSEL0 |= (2U<<6); /* ADC6 */
40     LPC_PINCON->PINSEL0 |= (2U<<4); /* ADC7 */
41
42     for (i = 0; i < 5000000; i++);
43
44
45     /* By default, the PCLKSELxvalue is zero, thus, the PCLK(portclock) for all
46     pclkdiv= (LPC_SC->PCLKSELO >> 24) & 0x03;
47     switch(pclkdiv) {
48     case 0x00:
49     default:
50         pclk= SystemCoreClock/ 4;
51         break;
52     case 0x01:
53         pclk= SystemCoreClock;
54         break;
55     case 0x02:
56         pclk= SystemCoreClock/ 2;
57         break;
58     case 0x03:
59         pclk= SystemCoreClock/ 8;
60         break;
61     }
62     LPC_ADC->ADCR = (0x01 << 0) | /* SEL=1,select channel 0-7 on ADC0 */
63     ((pclk/ ADC_CLK -1) << 8) | (0 << 16) /* BURST = 0, softwarecontrolled*/
64     (0 << 17) | /* CLKS = 0, 11 clocks/10 bits */
65     (1 << 21) | /* PDN = 1, normal operation*/
66 }
```

Abbildung 2:ADC.c

- ➔ Im ADC.c wird eine Funktion zur Initialisierung der ADC geschrieben, wobei Strom eingeschaltet wurde (Zeile 31 in Abbildung 2) und dabei die ADC als Input konfiguriert(Zeile 33 bis Zeile 40 in Abbildung 2)
- ➔ Dann wird in die Methode die Werte der Portclock(pclk) einzeln bestimmt(von Zeile 47 bis Zeile 60) mit SystemCoreClock ,die in system\_LPC17xx.c(Zeile 390 in Abbildung 3) definiert wurde

```

373
374
375  #define
376
377
378  /*
379  Define clocks
380  */
381  #define XTAL      (12000000UL)      /* Oscillator frequency */
382  #define OSC_CLK   ( XTAL)          /* Main oscillator frequency */
383  #define RTC_CLK   ( 32000UL)       /* RTC oscillator frequency */
384  #define IRC_osc   ( 4000000UL)     /* Internal RC oscillator frequency */
385
386
387  /*
388  Clock Variable definitions
389  */
390  uint32_t SystemCoreClock = IRC_osc; /*!< System Clock Frequency (Core Clock) */
391
392
393  /**
394   * Initialize the system
395   *
396   * @param none
397   * @return none
398   *
399   * @brief Setup the microcontroller system.
400   * Initialize the System and update the SystemFrequency variable.
401   */
402  void SystemInit (void)
403  {
404      if (CLOCK_SETUP)
405      {
406          LPC_SC->SCS = SCS_Val; /* Clock Setup */
407          if (SCS_Val & (1 << 5)) {
408              while ((LPC_SC->SCS & (1 << 6)) == 0); /* If Main Oscillator is enabled */
409              /* Wait for Oscillator to be ready */
410          }
411          LPC_SC->CLKCFG = CLKCFG_Val; /* Setup Clock Divider */
412          LPC_SC->CLKSEL0 = CLKSEL0_Val; /* Peripheral Clock Selection */
413          LPC_SC->CLKSEL1 = CLKSEL1_Val;
414          LPC_SC->CLKSRCSEL = CLKSRCSEL_Val; /* Select Clock Source for PLL0 */
415      }
416  }

```

Abbildung 3: System\_LPC17xx.h

- ➔ ADCValue wurde dann als Arrays definiert, konvertiert und die verschiedene Position des Arrays wurden dann für die Entsprechenden Richtungen(zum Beispiel Mitte, stark rechts usw.) eingerichtet und zurückgegeben. Dabei wird eine Schleife gesetzt auf die Ende der Konversion wartet(Zeile 104 in Abbildung 4)



```

98 |
99 | /*-----
100 | get converted AD value
101 | *-----
102 | uint16_t ADC_GetCnv (int ADCValue[5]) {
103 |     int adwert;
104 |     while (!(LPC_ADC->ADGDR & (1UL<<31))); /* Wait fo
105 |     ADCValue[0] = (LPC_ADC->ADDR0 >> 4) & 0xFFF;
106 |     ADCValue[1] = (LPC_ADC->ADDR1 >> 4) & 0xFFF;
107 |     ADCValue[2] = (LPC_ADC->ADDR2 >> 4) & 0xFFF;
108 |     ADCValue[3] = (LPC_ADC->ADDR3 >> 4) & 0xFFF;
109 |     ADCValue[4] = (LPC_ADC->ADDR5 >> 4) & 0xFFF;
110 |     //fahren mit hammerkopf
111 |     if(ADCValue[2] < 0x0250) {
112 |         adwert= 0; // mitte
113 |     } else if (ADCValue[1] < 0x0250) {
114 |         adwert= 1; // leicht rechts
115 |     } else if (ADCValue[3] < 0x0250) {
116 |         adwert= 2; //leicht links
117 |     } else if (ADCValue[0] < 0x0250) {
118 |         adwert= 3; //stark rechts
119 |     } else if (ADCValue[4] < 0x0250) {
120 |         adwert= 4; //stark links
121 |     }
122 |     return (adwert);
123 | }
124 |
125 |
126 | /*-----
127 | A/D IRQ: Executed when A/D Conversion is done
128 | *-----
129 | void IRQ_A_D (void) {

```

Abbildung 4: ADC.c

- ➔ Im Example3.c passiert es ziemlich gleich wie im Example2.c nur, dass die verschiedene Position der ADCValue für die Entsprechenden Richtungen(stark Links, leicht Rechts, Mitte usw.) zusätzlich hinzukommen(von Zeile 35 bis Zeile 56 in Abildung 5).Außerdem wurde hierbei ein Counter gesetzt, die die werten beeinflusst, wenn man vor dem Auto mit dem Straßenstückchen zu den Richtungen sich bewegt.

```

17 long j=0, advalue;
18 unsigned char str[20];
19 int ADCValue[5];
20
21 SystemInit();
22 // SysTick_Config(SystemFrequency/100); /* Generate inter
23 LED_Init(); /* LED Initializati
24 GLCD_Init(); /* LCD - Display Initialisation
25 ADC_Init(); /* A/D-Converter Initialisation */
26
27
28 GLCD_Clear (Red);
29 GLCD_SetTextColor (White);
30 GLCD_SetBackColor (Red);
31 GLCD_DisplayString (1, 1, 1, "Example 2");
32
33 //ADC_StartCnv();
34
35 while (1) { /* Loop forever
36
37 if (j++>100000000)
38 {
39 j = 0;
40 i = 0x00FF;
41 LED_Out (i++);
42 sprintf (str,"Count = %i",i);
43 GLCD_DisplayString (2, 1, 1, str);
44
45 advalue = ADC_GetCnv(ADCValue);
46 sprintf (str,"A/D = %4.4i",ADCValue[0]);
47
48 GLCD_DisplayString (3, 1, 1, str);
49 sprintf (str,"A/D = %4.4i",ADCValue[1]);
50 GLCD_DisplayString (4, 1, 1, str);
51 sprintf (str,"A/D = %4.4i",ADCValue[2]);
52 GLCD_DisplayString (5, 1, 1, str);
53 sprintf (str,"A/D = %4.4i",ADCValue[3]);
54 GLCD_DisplayString (6, 1, 1, str);
55 sprintf (str,"A/D = %4.4i",ADCValue[4]);
56 GLCD_DisplayString (7, 1, 1, str);

```

Abbildung 5: Example3.c

### 3.Example 4

Ziel hier war die Pulsweitenmodulation(PWM) zu implementieren im Programm, die zur Lenkung und gas geben von Auto dient

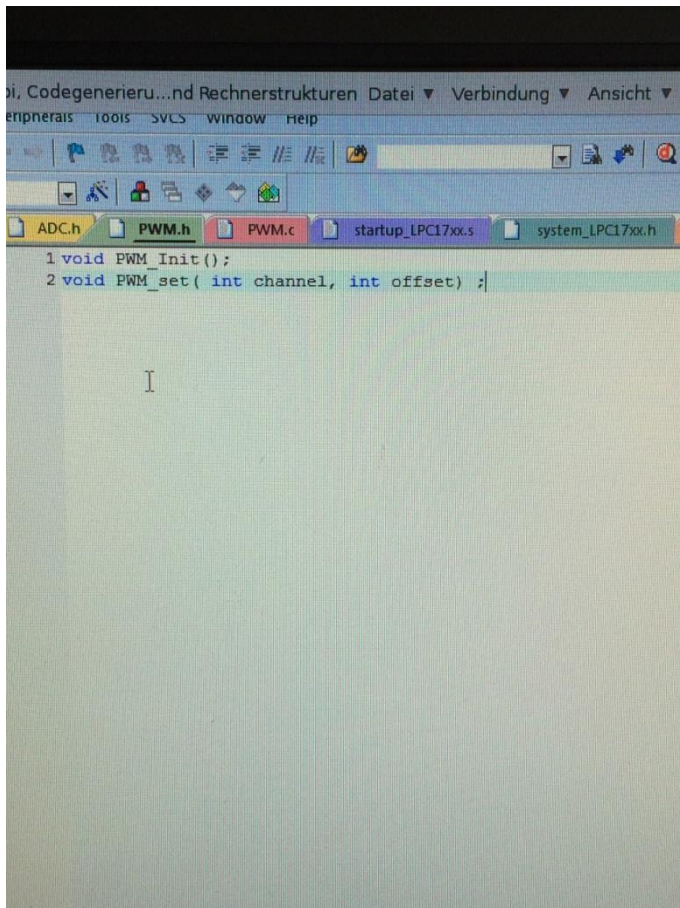


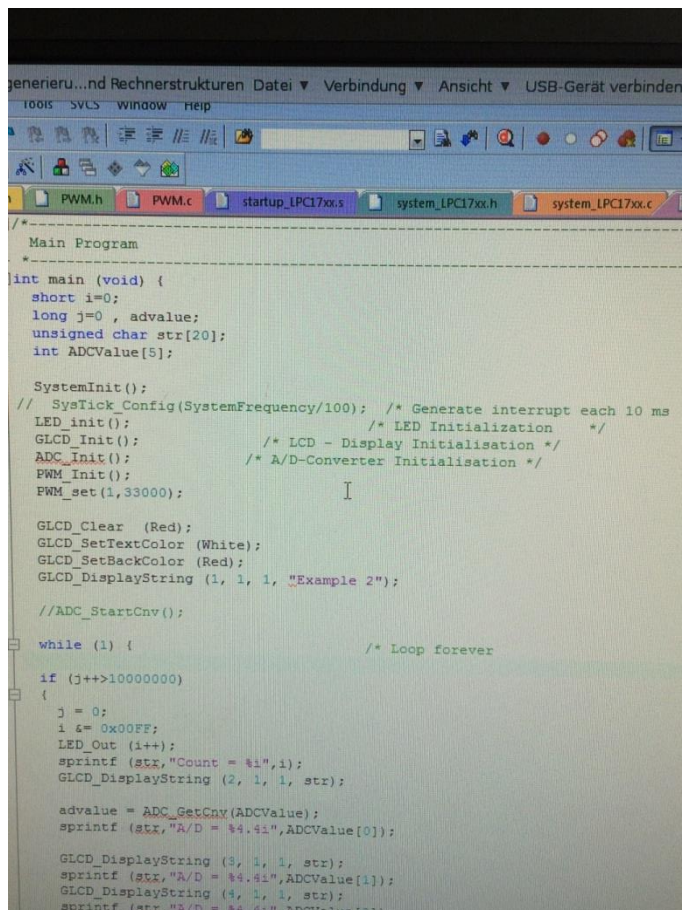
Abbildung 6: PWM.h

- ➔ In den PWM.h Datei wird die Methoden PWM\_Init und PWM\_set(mit zwei Parameter) deklariert(siehe Abbildung 6)
- ➔ Diese Methoden werden dann in die PWM.c Datei implementiert, wobei PWM\_Init für die Initialisierung der PWM dient (von Zeile 5 bis Zeile 10) und die PWM\_set methode für den Motor und die Lenkung am Auto dient mit den Jeweiligen Parametern channel (Motor) und Offset(Lenkung).Das ganze lässt sich dann deutlich in der Abbildung 7 darstellen:



Abbildung 7: PWM.c

- ➔ Im example3.c wird dann die PWM\_Init und die PWM\_set Methode aufgerufen mit gewissen Parametern(1 und 33000). Dies dient, dass die Räder des Auto sich nach Links, Rechts oder in der Mitte bewegen beim Testen vorne am Sensor mit dem Straßenstückchen. In der Abbildung 8 ist zeigt sich das ganze an:



```
/*
Main Program
*/
int main (void) {
    short i=0;
    long j=0, advalue;
    unsigned char str[20];
    int ADCValue[5];

    SystemInit();
    // SysTick Config(SystemFrequency/100); /* Generate interrupt each 10 ms
    LED_init(); /* LED Initialization */
    GLCD_Init(); /* LCD - Display Initialisation */
    ADC_Init(); /* A/D-Converter Initialisation */
    PWM_Init();
    PWM_set(1,33000);

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

    //ADC_StartCnv();

    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 = %4.1i", ADCValue[0]);

        GLCD_DisplayString (3, 1, 1, str);
        sprintf (str, "A/D = %4.1i", ADCValue[1]);
        GLCD_DisplayString (4, 1, 1, str);
        sprintf (str, "A/D = %4.1i", ADCValue[2]);
    }
```

Abbildung 8: Example3.c

## 4. MerapiUML

Ziel der Aufgabe hier war sich mit dem Class-, Object- und State-Diagramm und dessen Nutzung in Merapi vertraut machen und dabei codegenerieren, damit dem Auto auf eine Straße mit gegebene Geschwindigkeit fährt.

### 4.1. Klassendiagramm

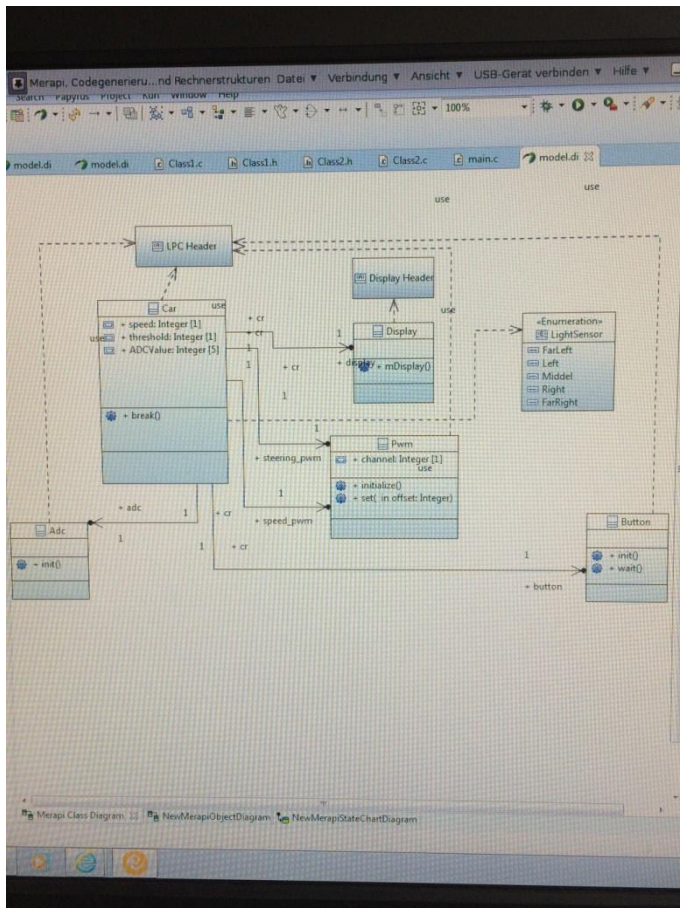


Abbildung 9: Class Diagram

- ➔ Car ist die Hauptklasse, die die Klassen Adc, Pwm, Display und Button (beim Drücken werden sich die Räder nach einem Kligelton mit entsprechenden Geschwindigkeit drehen, die enthält eine init- und eine wait-methode) mit jeweils Property (Attribute) und Operation (Methoden) nutzt

## 4.2. Beschreibung Einzelne Methoden in Verschiedene Klassen

- ➔ Die Pwm-Klasse nutzt speed und steering. In die initialize-Methode wird die Pulsweitenmodulation initialisiert (siehe Abbildung 10)



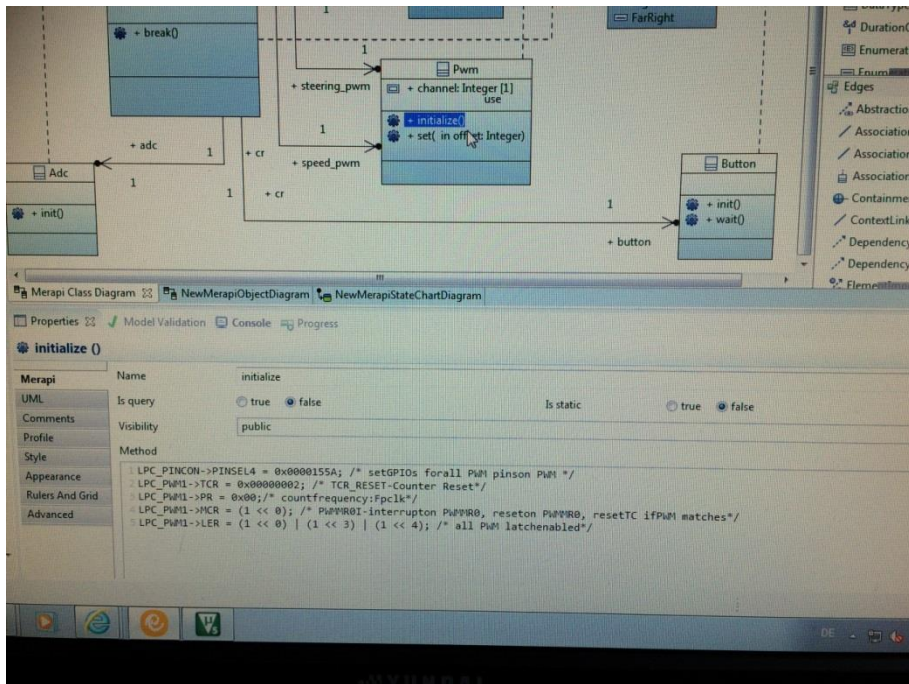


Abbildung 10: Pwm\_initialize

- ➔ Die Pwm-Klasse nutzt speed und steering. In die set -Methode wird für die Pulsweitenmodulation eine Funktion für die Lenkung(steering) und den Motor(speed) gesetzt(siehe Abbildung 11)

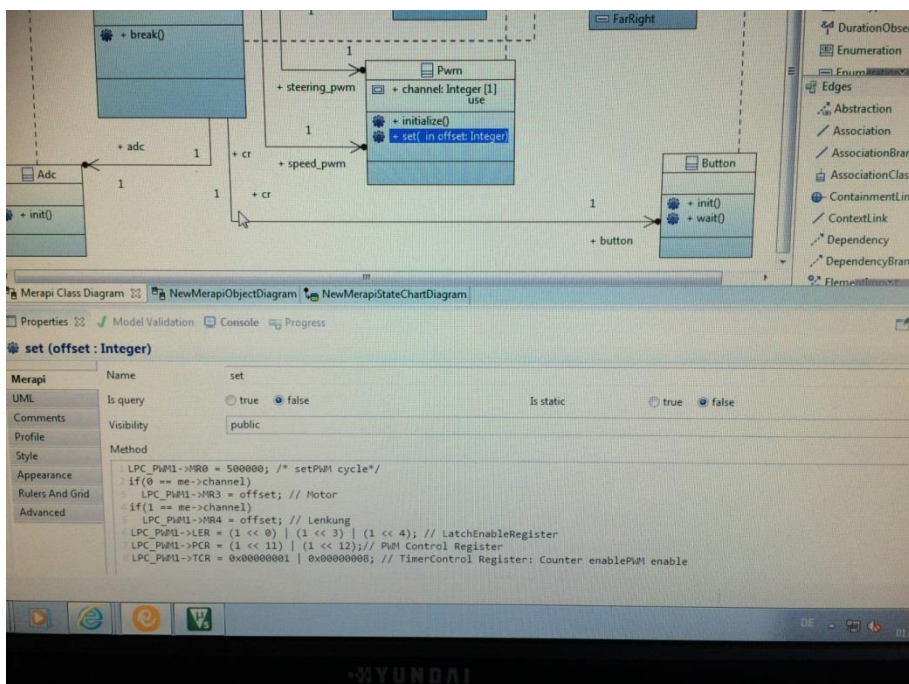


Abbildung 11: Pwm\_set

➔ In der Klasse Adc wird in der init-Methode den portclock(pclk) initialisiert, den Strom eingeschaltet und die ADC Pins als Input konfiguriert.

➔ Die Abbildung 12 ist eine Illustration davon:

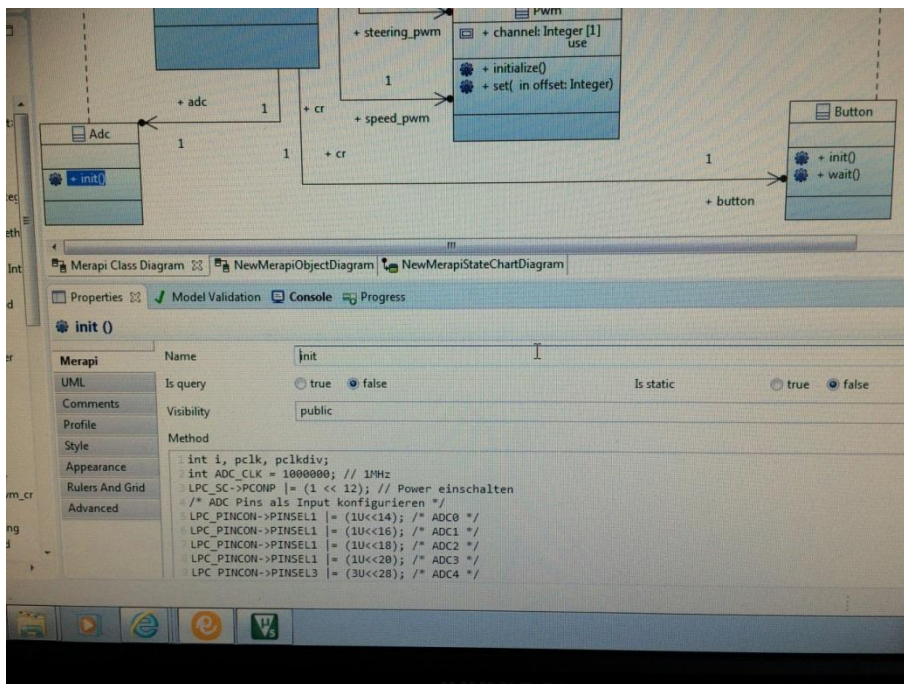


Abbildung 12: Adc\_init

### 4.3. Object Diagram dient zur Struktur des Programms

Das Objektdiagramm ist ein Strukturdiagramm, denn es zeigt eine bestimmte Sicht auf die Struktur des modellierten Systems, wobei jedes



Objekt für eine Klasse steht. Die Darstellung umfasst dabei typischerweise Ausprägungsspezifikationen von Klassen und Assoziationen.

In diesem Diagramm spielt "Car" die main-Methode Rolle. Andere Methoden gehören zu Car-Methode. "Display" kontrollieren die Auszeichen des Bildschirms.

"pwm-steering" kontrolliert die Lenkung des Autos.

"Button"

"pwm-speed" kontrolliert die Geschwindigkeit des Autos. Wir müssen unbedingt eine passende Geschwindigkeit auswählen. Wenn die Geschwindigkeit zu schnell ist, werden sich die Räder des Autos rückwärtsdrehen.

"adc" Kontrolliert die Lenkung wenn das Auto gefahren wird. Sensoren kann das Licht messen. Durch A/D Wandler, kann man das Signal bekommen und bearbeiten.

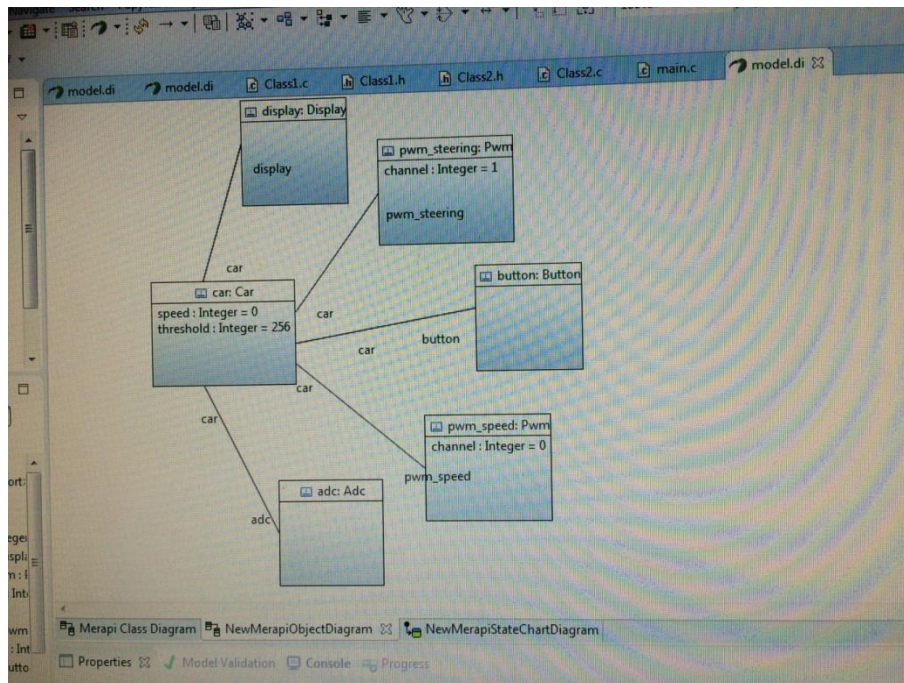


Abbildung 13: Object Diagram

#### 4.4. State Diagram steht das Verfahren von dem gesamten Programm

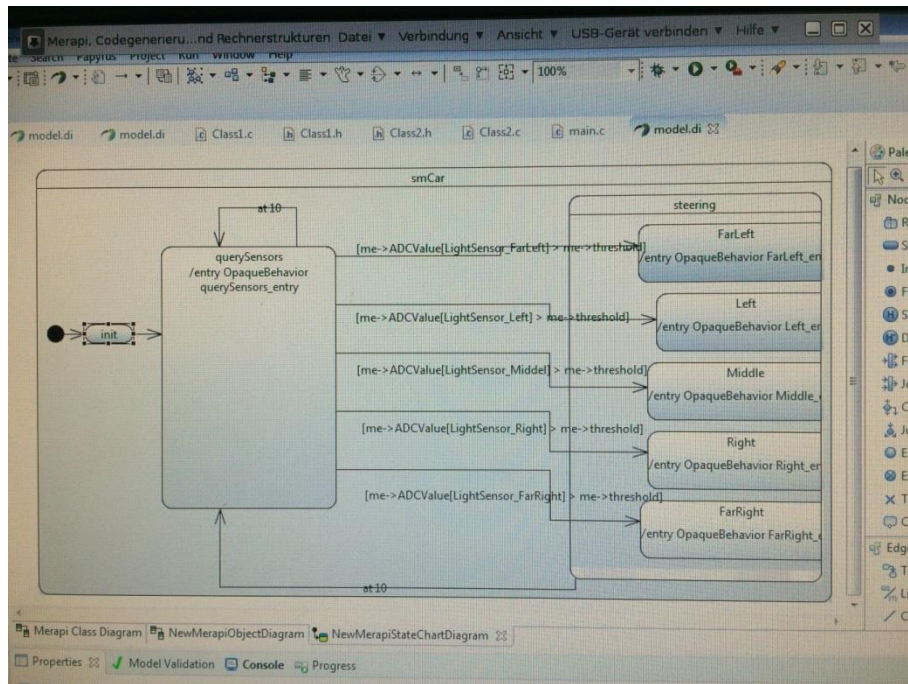


Abbildung 14: State Diagram

➔ Im „init“ wird die zu durchführenden Schritte als Methoden deklariert(siehe Abbildung 15)

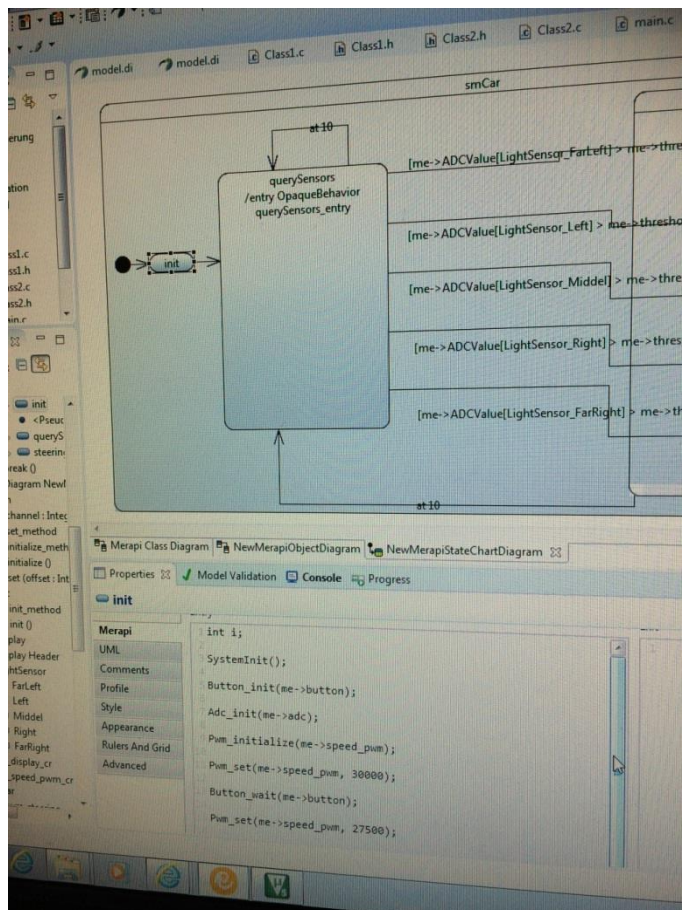


Abbildung 15: init

- ➔ Im „querySensors“ werden die Werte der ADCValue deklariert und initialisiert. Mehrere Vorstellung kann man sich mit dem Abbildung 16 vorstellen

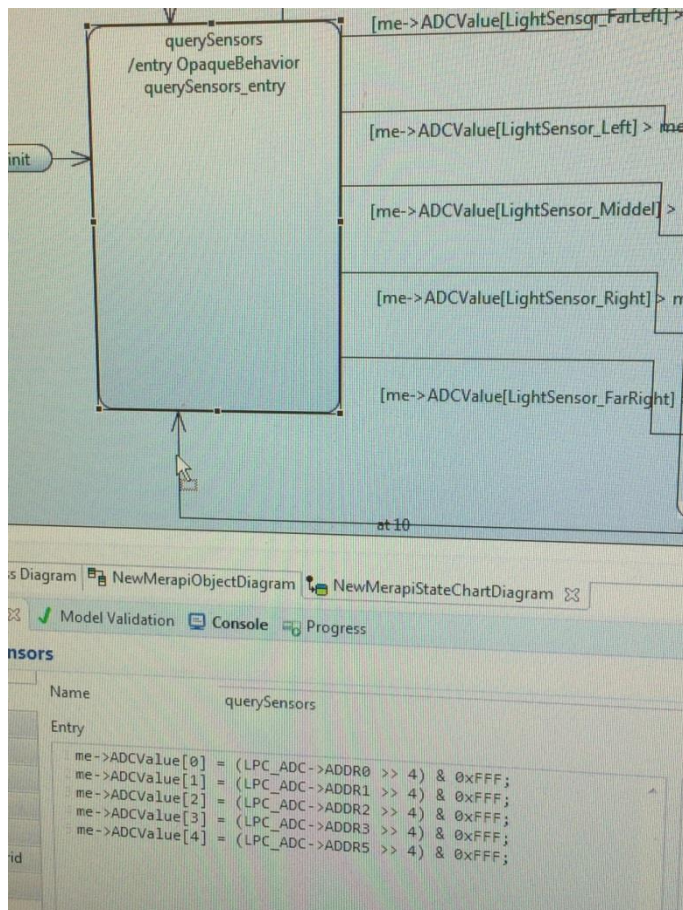


Abbildung 16: querySensors

- ➔ Letztes werden die entsprechenden `guard[me->ADCValue[LightSensor_FarLeft] > me->threshold]` zu den dazugehörigen Steuerungen eingerichtet (zum Beispiel FarLeft). Das ganze sieht man wieder auf dem Abbildung 17



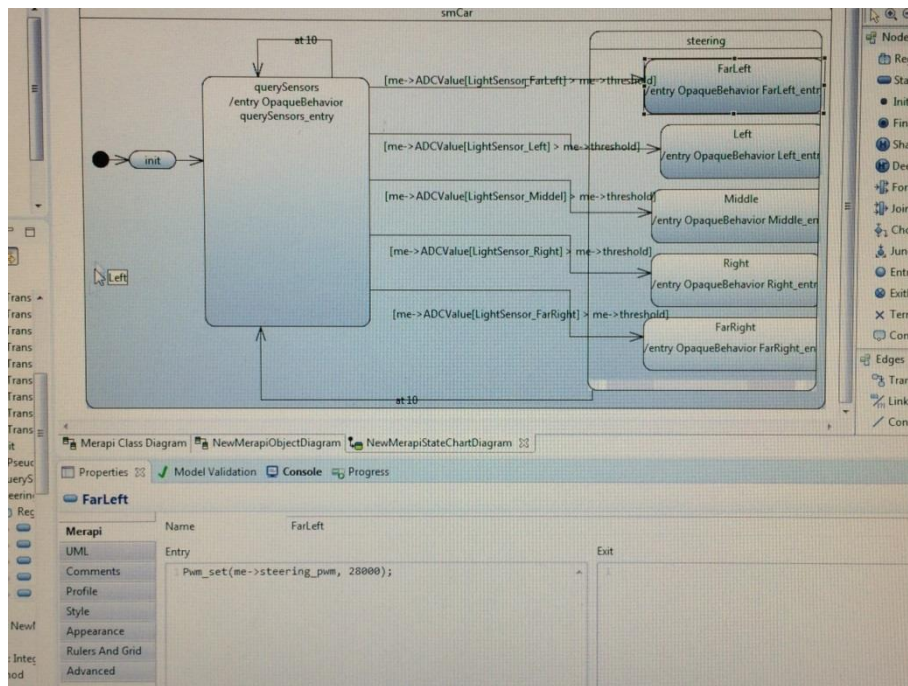


Abbildung 17: Steering

- Das geschriebene Programm in Merapi lässt sich zusammen in Keil darstellen und mit-Kompilieren. Hier Nochmal die komprimierteZip-Datei davon:



Codegenerierung.zip

## 5. Verhältnis von Klassen

Ziel der Aufgabe war sich den Verhältnis von Class1 und Class2 anschauen

- ➔ Wenn wir im UML Klassendiagramm eine Klasse anlegen, erzeugt sich diese automatisch unter den Papyrus Projekt



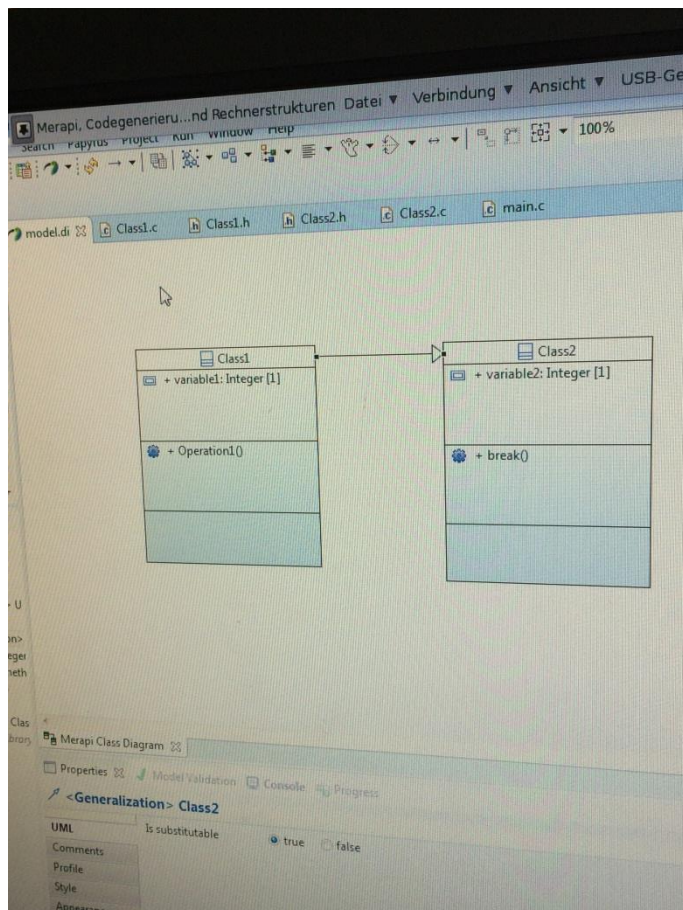


Abbildung 18: Generator Class Diagram

- ➔ Auf diesem Klassendiagramm(Abbildung 18) stehen Class1 und Class2 die mit eine Aggregation zueinander verkoppelt sind
- ➔ Durch dem Symbol wird angedeutet, dass die Methoden von Class2 in Class1 sind
- ➔ Es wird die Funktionen "void RootElement\_Class\_Inti()" , "RootElement\_Class\* Rootelement\_Class\_New()" , "void RootElement\_Class\_Cleanup()" und "void RootElement\_Class\_Destructor()" automatisch erzeugt. Alle Klassen müssen diese vier Funktionen haben.(siehe Abbildung 19)

```

----- Class management functions -----
/** -- auto-generated initialization -- */
void RootElement_Class2_Init(RootElement_Class2* const me) {
    /* No code was specified in the model. */
}

/** -- auto-generated constructor -- */
RootElement_Class2* RootElement_Class2_New(void) {
    RootElement_Class2* me = (RootElement_Class2 *) malloc(sizeof(RootElement_Class2));
    if(me != NULL) {
        RootElement_Class2_Init(me);
    }
    return me;
}

/** -- auto-generated cleanup -- */
void RootElement_Class2_Cleanup(RootElement_Class2* const me) {
    /* No code was specified in the model. */
}

/** -- auto-generated destructor -- */
void RootElement_Class2_Destructor(RootElement_Class2* const me) {
    if (me != NULL) {
        RootElement_Class2_Cleanup(me);
    }
    free(me);
}

```

Abbildung 19: Class2.c

- ➔ In Class1.c wird die in Class1.h definierte Methode(Operation1) implementiert(siehe Abbildung 20).Sonst wird in Class1.h und Class2.h die variable1 und 2 jeweils definiert, zusätzlich wird die variable2 und Methode break von Class2 in Class1.h definiert.
- ➔ Die in Class1.h und Class2.h definierten Methoden werden beziehungsweise in Class1.c und Class2.c implementiert. Da meistens in h-Files Definition stehen und in c-Files Implementierung.

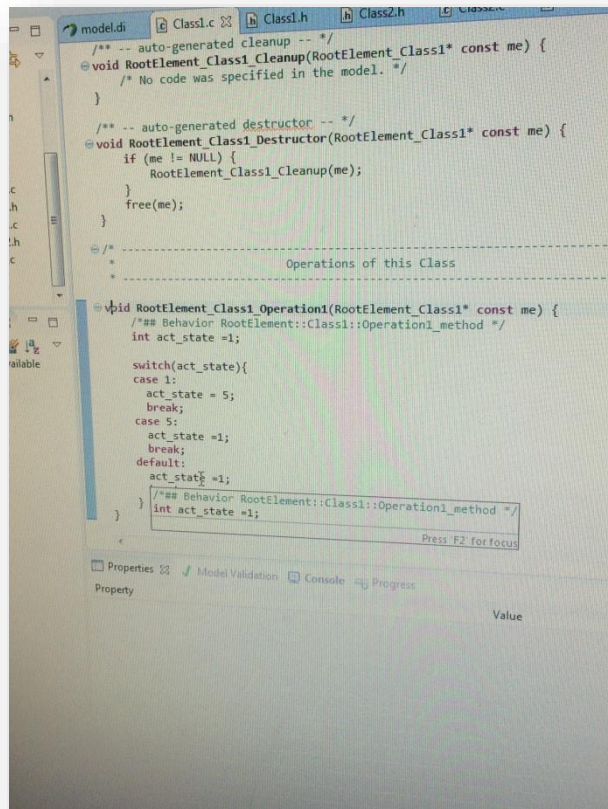


Abbildung 20: Class1.c

- ➔ Code schreibt man unter Implementation beim Class diagram und taucht automatisch in der entsprechenden c-Datei. Zum Beispiel die Methode break von Class2. Illustration davon sieht man auf den Abbildung 21 und Abbildung 22:

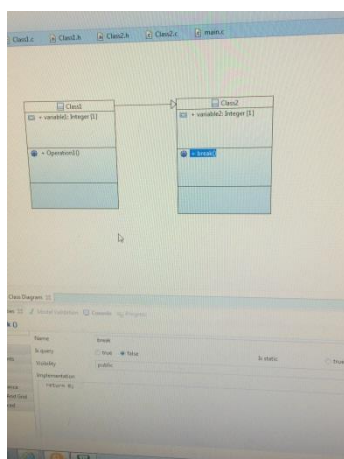


Abbildung 21: Class2\_break

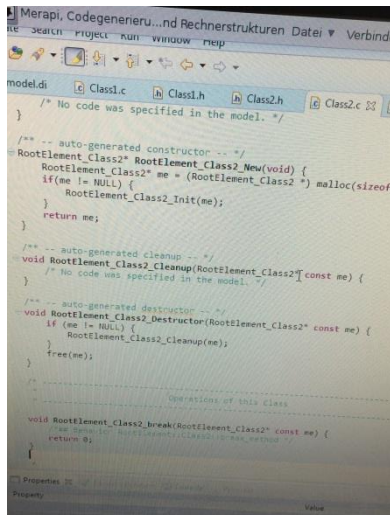


Abbildung 22: Class2.c

## 6.XTEND Programmiersprache

In diese Übung sollten wir den vorliegenden Code korrigieren. Code testen läuft in JUnit, die eine Einheit des Test-Frameworks für Java-Programme. In JUnit suchen wir, wo die Fehler sind. Danach korrigieren wir diese. Wir sollten zwei Teile machen. Eine ist über Parameter und das andere über Operation.

➔ Folgend sind einige JUnit-Test, die wir gelöst haben und die dazugehörigen Lösungen, die wir hergestellt haben.

. blau sind Test

. rot sind Lösungen

### 6.1.TestOperation

```

@Test def testOneParameterOperation() {
    val operation = createOperation => [

```

```

        name = "oneParameterOperation"
        .....
        val code = (new Uml2C).generateCode(operation)
        Assert.assertEquals("void  oneParameterOperation(TestClass*  const  me,
uint32 param1);", code)
    }
    if(operation.name == "oneParameterOperation"){
        ){
            return '''void (TestClass* const me, uint32 param1);'''
        }
    }
    @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 == "twoParameterOperation" ){
        return '''void twoParameterOperation(TestClass* const me, uint32
param1, uint8 param2);'''
    }

    @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 == "returningOperation" ){
        return '''uint32 returningOperation(TestClass* const me);'''
    }

    @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 == "implementedOperation" ){
    return ""
    void implementedOperation(TestClass* const me) {
        /* hier kaennte Ihre Werbung stehen */
    }
    ""
}

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

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

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

```

## 6.2.TestParameter

```
@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)
}

if(ParameterDirectionKind.INOUT_LITERAL == umlParameter.direction &&
    umlParameter.type instanceof PrimitiveType
){
    return '''«generateType(umlParameter.type)»*
«umlParameter.name>'''
}

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.RETURN_LITERAL == umlParameter.direction) {
    return '''«generateType(umlParameter.type)'''
}

@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(ParameterDirectionKind.OUT_LITERAL == umlParameter.direction){
    return '''«generateType(umlParameter.type)»*
«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)
    }

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

```

## 7. Aktivitäts- Diagramm

Diagramm anlegen geht über: Papyrus projekt-neu-Papyrus modell-name eingeben-  
create new-activity Diagram.Ergebnis wird auf dem Abbildung 23 angezeigt:

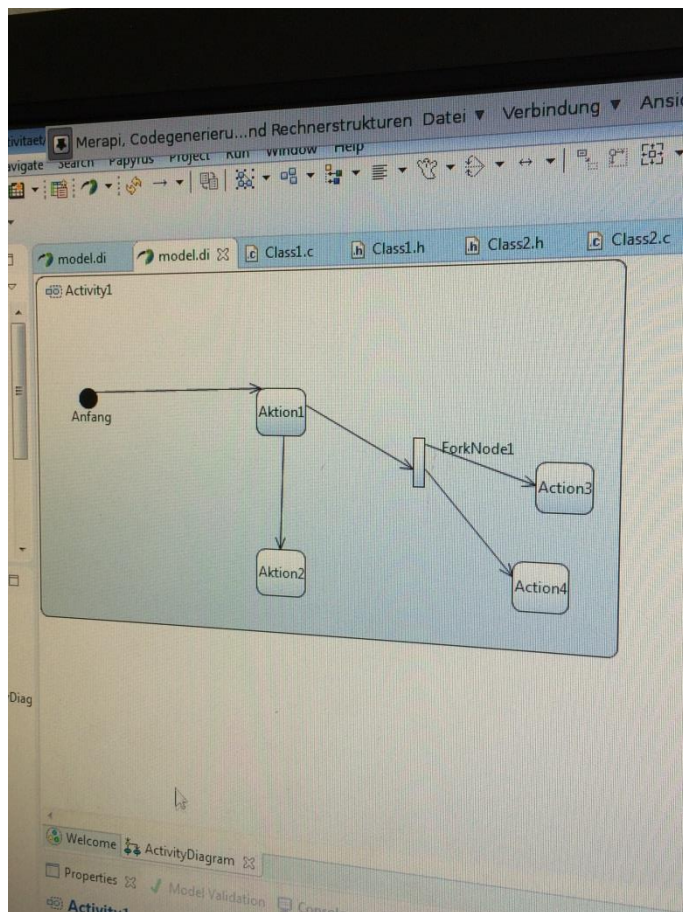


Abbildung 23: Activity Diagram