Dimm-Licht mit Fernbedienung

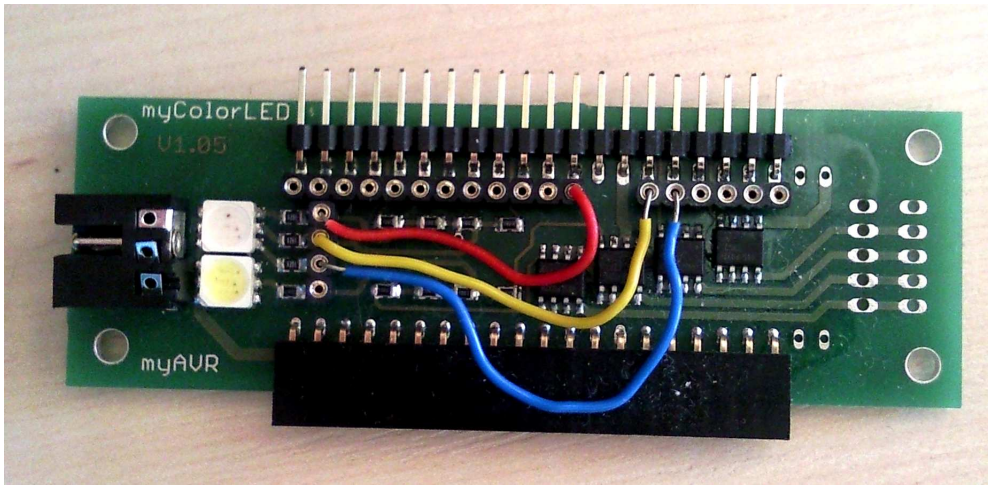
Projektbeschreibung

Dimmbare komfortable LED-Beleuchtung die mit Tasten oder Infrarot-Fernbedienung zu steuern ist.

Die Idee

Meine Idee war eine neue Beleuchtung für unser Wohnzimmer zu bauen. Die bisherige war ein Halogen-Deckenfluter mit 300 Watt. Der reichte zum Lesen nur gerade so aus und war auch schon klapprig und unschön.
Also wenn schon – denn schon: energiesparend und mit Fernbedienung.
Den Anstoß bekam ich durch das myColor-LED-Addon, als ich mir das myAVR-Aufsteigerset-PLUS bei CONRAD bestellte. Bisher hatte ich schon einige Sachen mit meinem MK2 gemacht.

myAVR-Aufsteigerset-MK3-PLUS bei CONRAD



myColorLED-Addon

Der Plan - Hardware

Es sollte wieder eine indirekte Beleuchtung werden, d.h. Licht an die Decke und von dort reflektiert in den ganzen Raum.

Wie anbringen? → Leiste ringsherum (naja nicht komplett) ca. 30cm. von oben und darauf LED-Stripes.

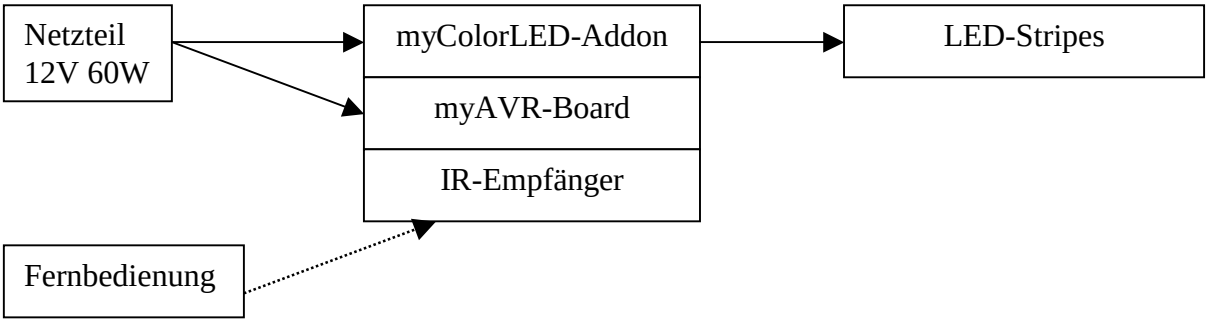
Welche LEDs? → Erst hatte ich warmweiß probiert, ist zwar kuscheliger aber zum lesen schlechter geeignet, also kaltweiß. Oder doch RGB? Wäre eine niedliche Spielerei, hab es aber dann abgewählt, da es in der Praxis später ja doch nicht benutzt würde. Praktisch ist auch, dass man die alle 10cm trennen kann.

Welche Stromversorgung? → Die Stripes brauchen keinen Stromregler ☺, dadurch nur ein 12V – Netzteil mit 60 Watt (10m Stripes bei 6 W/m). Das Netzteil soll auch gleich das myAVR-Board versorgen.

Welche Ansteuerung? → Aus dem Aufsteigerset das myAVR-Board-MK3 mit myColorLED-Addon. Hab dann später das myAVR-MK2-Board genommen, da es völlig ausreichte.

Welche Fernbedienung? → Die vorhandene programmierbare Infrarot-Fernbedienung, mit Tasten die ich sonst nicht nutze.

Welchen Infrarot-Empfänger? → Da hab ich mich erst mal belesen müssen. Nach einigen Tests klappte es mit dem TSOP34838 von Vishay.



Der Plan – Software

Da ich ja das tolle myAVR-Aufsteigerset hatte, hab ich mit dem enthaltenen SiSy begonnen. Assembler ist nix so richtig für mich und die Aufgabe nicht zeit- und platzkritisch.
Erst im "Kleinen Programm", aber nach etwas schnüffeln und Experimentieren hab ich das geniale Klassendiagramm gefunden. Hier die vorhandenen Beispiele mal ausgeführt und dran herumgeändert, die fertigen Pakete durchgestöbert und Fingerübungen gemacht.

Entschluß: Mit SiSy im Klassendiagramm in C++ programmieren.

Was soll die Software können?

- LED-Ansteuerung
- Taste für an – aber bitte langsam gedimmt
- Taste für aus – auch langsam gedimmt
- Infrarot-Empfänger/Decoder
- eine Taste (oder mehrere Tasten?) für Infrarot-Code-Lernen
- eine Status-LED (oder mehrere?)

Die Umsetzung - Software

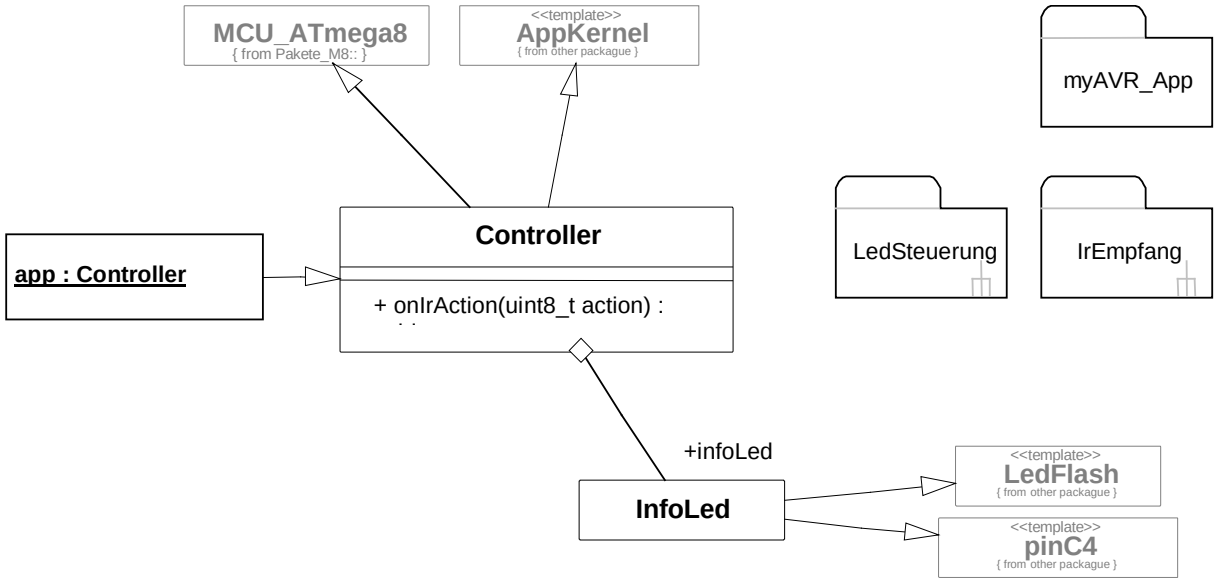
Um die Struktur festzulegen hab ich erst mal vorhanden Pakete und Klassen die für mich und das Projekt brauchbar schienen einzeln probiert – ging recht schnell. Dann ein neues Projekt gemacht und aus den Erfahrungen das Programm zusammengestellt.

Das Ergebnis:

- Betriebsystem-Grundlage: AppKernel → autonome AppModule die sich selbst kümmern, mit Ereignisse wie onStart(), onWork(), onTimer10ms() ...
- AppModul: mit ein PWM-Channel für LED-Steuerung
- AppModul: Taste für Dimm-an (=klick) / Stopp (=klick) / Heller (=halten)
- AppModul: Taste für Dimm-aus (=klick) / Stopp (=klick) / Dunkler (=halten)
- AppModul: Infrarot-Decoder
- AppModul: Info-LED, Blinkt kurz wenn eine Aktion war

Klassendiagramm – Programm

Klasse: Controller	zentraler Kernel, mit Funktion die Ereignisse von IrEmpfang zu LedSteuerung bringt
globales Objekt: app	ist Programmeintrittspunkt
Klasse: InfoLed	LED die kurz blinkt wenn eine Aktion war, AppModul mit der Funktion flash() die das Aufblinken übernimmt und ggf. Dauerleuchten verhindert



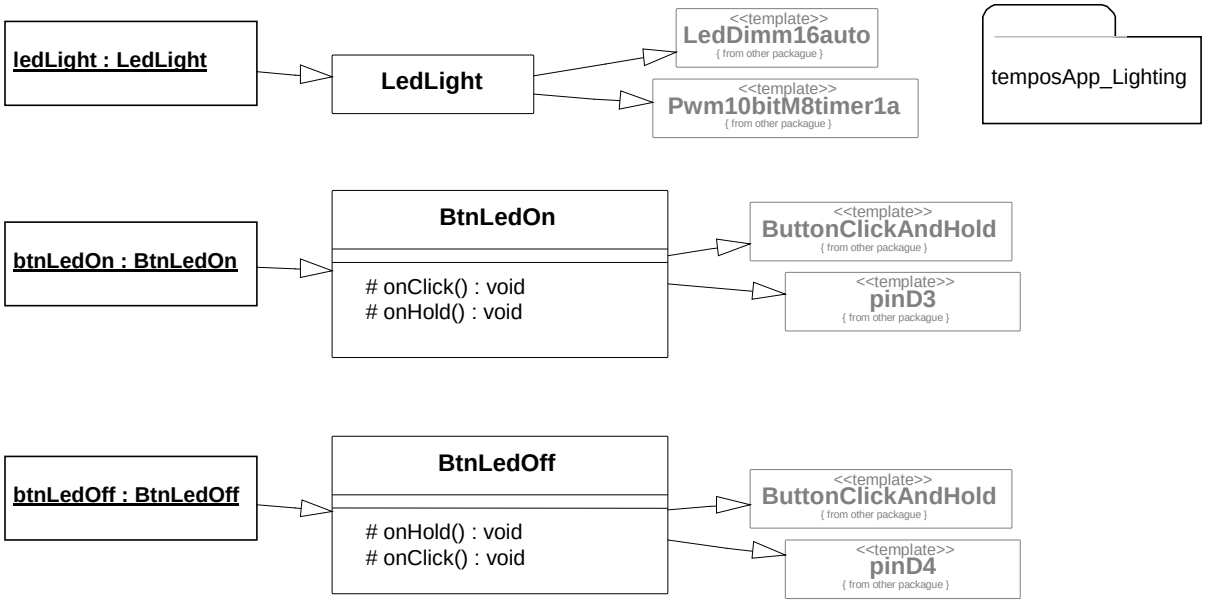
Klassendiagramm – Paket LedSteuerung

Klasse: LedLight — Steuert die LED-Ausgabe und Verhalten, ist komplexes AppModul mit komfortablen Funktionen wie stopDimming(), dimTo(0xffff), more(200), less(200) ...

Klasse: BtnLedOn — Taste für (langsam) an bzw. heller, AppModul mit Ereignissen wie onHold(), onClick() ... ist selbstverständlich entprellt

Klasse: BtnLedOff — Taste für (langsam) aus bzw. dunkler, AppModul mit Ereignissen wie onHold(), onClick() ... ist selbstverständlich entprellt

globale Objekte: ledLight, btnLedOn, btnLedOff

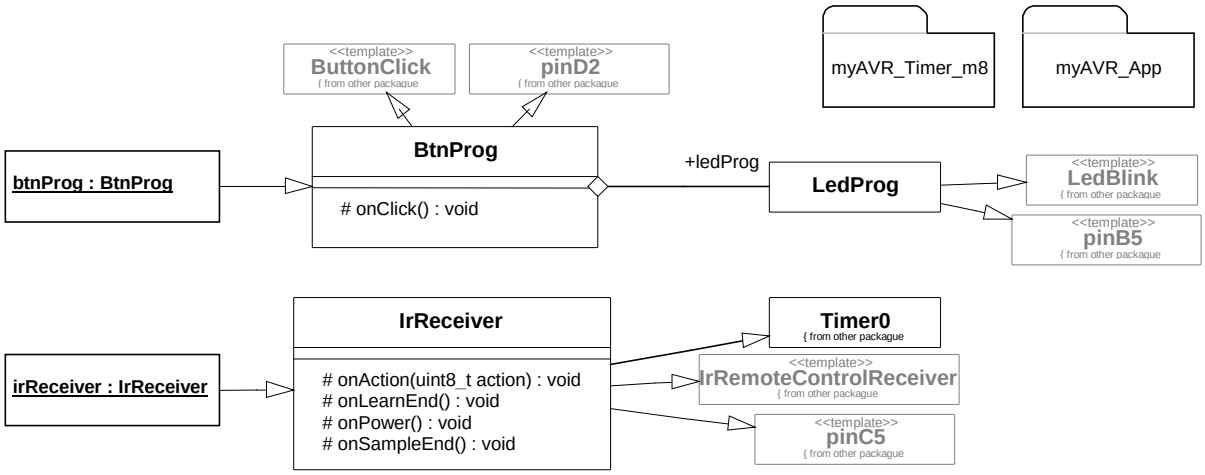


Klassendiagramm – Paket IrEmpfang

Klasse: BtnProg — AppModuls, entprellte Taste um Lernmodus für Infrarot-Empfang zu aktivieren und die Funktion anzuwählen (an/aus)

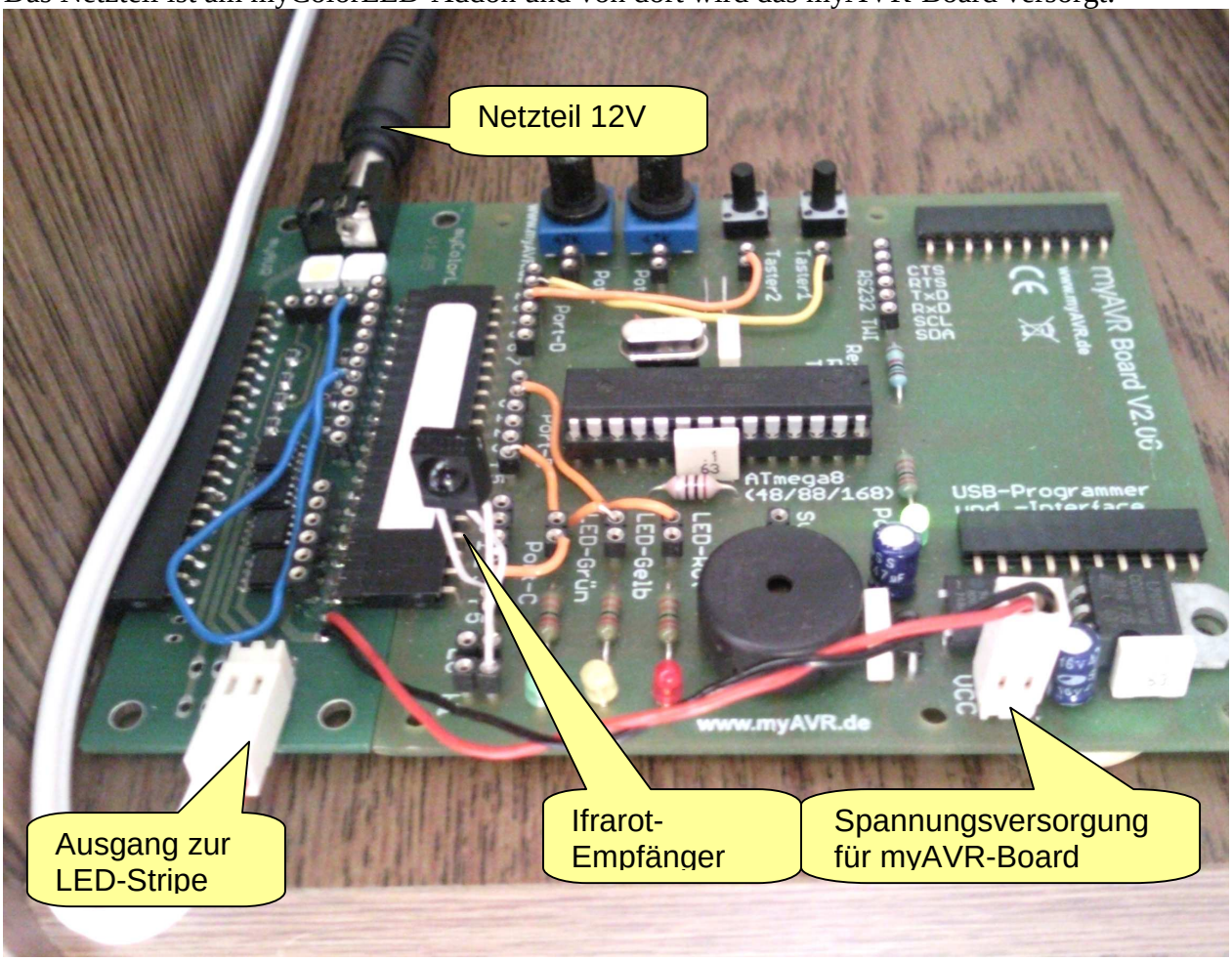
Klasse: LedProg — AppModul, Led die 1x blinkt für Lerne-Taste-An, 2x blinkt für Lerne-Taste-Aus und aus ist wenn der Lernmodus inaktiv ist

Klasse IrReceiver — komplexes AppModul, dass den vollständigen Infrarot-Empfang bearbeitet, einschließlich Lernfunktion



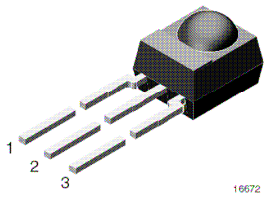
Die Umsetzung – Hardware

Das Netzteil ist am myColorLED-Addon und von dort wird das myAVR-Board versorgt.



Anschluss der LEDs: InfoLED an C4, LedProg an B5, LED-Stripe-Ausgang an B1 (zum Test auch an roter LED auf dem myAVR-Board)

Taster: BtnLedOn an D3, BtnbLedOff an D4, BtnProg an D2 (per Patchkabel, wird ja nur 1x benötigt)

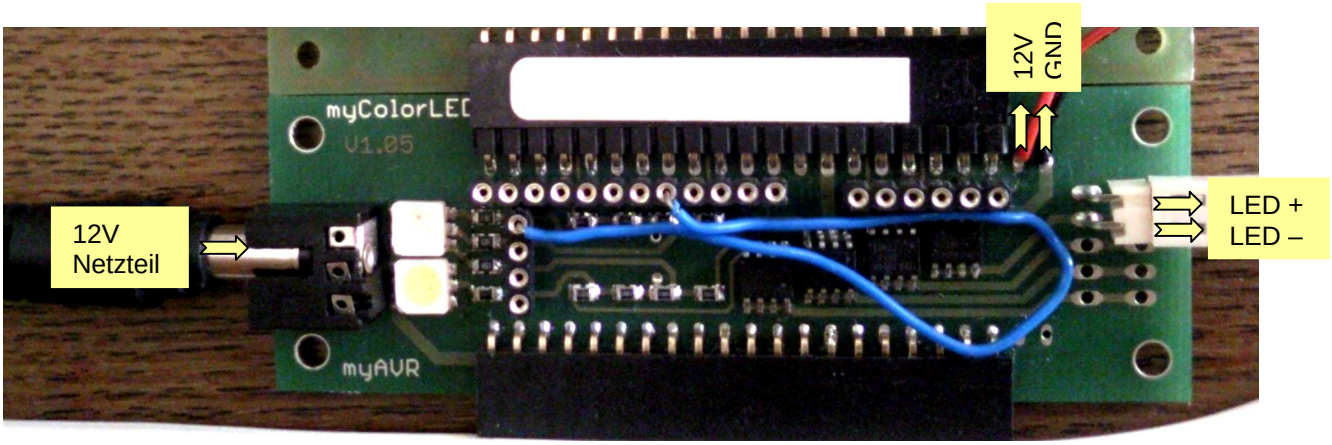


MECHANICAL DATA

Pinning:

1 = OUT, 2 = GND, 3 = V_S

Der Infrarot-Sensor ist einfach anzuschließen, Pin1 an C5, Pin2 an GND, Pin3 an 5V.

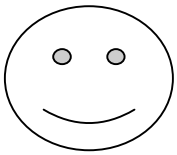


Am ColorLED-Addon B1 mit Rot verbinden (Patchkabel), Netzteil links ran, LED-Stripe rechts ran, und 12V für myAVR-Board oben rechts.

Inbetriebnahme

1. Hardware aufbauen
2. Software aufspielen
3. Test der Funktion mittels Tasten an/aus
4. Fernbedienung anlernen, Fernbedienung dabei ca. 20cm vor Infrarot-Empfänger halten und wenn möglich anderes störendes Licht vermeiden.
    - BtnProg betätigen (Patchkabel) bis LedProg 1x blinkt
    - auf Fernbedienung gewünschte An-Taste 1x drücken
    - BtnProg betätigen (Patchkabel) bis LedProg 2x blinkt
    - auf Fernbedienung gewünschte Aus-Taste 1x drücken
5. Test der Funktionen mit Fernbedieung

Fertig !



Ausblick

Damit es hübsch aussieht und kein Staubfänger wird, will ich die Taster noch auf einen Unterputz-Wipp-Schalter verlängern und den Sensor mit den LEDs am Rand der Wippe im Rahmen unterbringen.

controller.h

```
/// Code generated by SiSy
///<ObjektNummer>2065</ObjektNummer>

#if !defined(h_Controller)
#define h_Controller

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include <avr\interrupt.h>
#include "InfoLed.h"
#include "AppModul.h"

#include "myAVR_App.h"
#include "IrEmpfang.h"
#include "LedSteuerung.h"

extern AppModul* pFirstAppModul;

#define IRRCR_AnzActions 4
//#define IsDEBUG                 // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class Controller
{

public:

    //  onIrAction()
    void onIrAction(uint8_t action);

    //  main()
    void main();

    //  onInt_timer10ms()
    void onInt_timer10ms();

    //  Zeit seit Systemstart in Sekunden
    uint32_t volatile systimeSec;
    //  Bruchteile einer System-Sekunde in 10ms
    uint8_t volatile systimeMsec10;
    InfoLed infoLed;
    //  Konstruktor
    Controller();

    //  Destruktor
    ~Controller();
```

```
private:

protected:

    //  onStart()
    void onStart();

    //  onTimer10ms()
    void onTimer10ms();

    //  onPower()
    void onPower();

    //  onWork()
    virtual void onWork();


};

#endif
```

controller.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2065</ObjektNummer>

#define GeneratedBySisy
#include "Controller.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#ifndef TIMER10MS
#define TIMER10MS 1
#endif
#if TIMER10MS==1
#define TIMER10MS_VECT TIMER1_COMPA_vect
#define TIMER10MS_CLASS Timer1
#elif TIMER10MS==2 && defined(TIMER2_COMPA_vect)
#define TIMER10MS_VECT TIMER2_COMPA_vect
#define TIMER10MS_CLASS Timer2
#elif TIMER10MS==2 && defined(TIMER2_COMP_vect)
#define TIMER10MS_VECT TIMER2_COMP_vect
#define TIMER10MS_CLASS Timer2
#elif TIMER10MS==3
#define TIMER10MS_VECT TIMER3_COMPA_vect
#define TIMER10MS_CLASS Timer3
#elif TIMER10MS==4
#define TIMER10MS_VECT TIMER4_COMPA_vect
#define TIMER10MS_CLASS Timer4
#elif TIMER10MS==5
#define TIMER10MS_VECT TIMER5_COMPA_vect
#define TIMER10MS_CLASS Timer5
#else
#error TIMER10MS ungültig
#endif
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


ISR(TIMER10MS_VECT)
{
    app.onInt_timer10ms();
}
////////////////////////////
```

```
//
//  Konstruktor
//
////////////////////////////
Controller::Controller()
{
systimeSec = 0;
systimeMsec10 = 0;


}
////////////////////////////
//
//  Destruktor
//
////////////////////////////
Controller::~Controller()
{

}
/*////////////////////////////
//
//  onStart()
//
////////////////////////////
*/
void Controller::onStart()
{


}
/*////////////////////////////
//
//  onTimer10ms()
//
////////////////////////////
*/
void Controller::onTimer10ms()
{


}
/*////////////////////////////
//
//  onPower()
//
////////////////////////////
*/
void Controller::onPower()
{

    //#################### DEBUG ##############
    UCSRB=0b00011000;   // RX+TX enable
    UCSRA|=0b00000010;  // U2X=1
```

```
    UBRRL=3;            // 115200baud
    //UBRRL=1;          // 230400baud
    DDRD|=BIT1;         // out

}
/*////////////////////////////
//
//  onIrAction()
//
////////////////////////////
*/
void Controller::onIrAction(uint8_t action)
{
    infoLed.flash();
    // Aktion von IR-Empfang
    if(action==0)
    {
        if(ledLight.isDimming())
            ledLight.stopDimming();
        else
            ledLight.dimTo(0xFFFF);
    }
    if(action==1)
    {
        if(ledLight.isDimming())
            ledLight.stopDimming();
        else
            ledLight.dimTo(0);
    }

}
/*////////////////////////////
//
//  main()
//
////////////////////////////
*/
void Controller::main()
{
    AppModul* pm;
    ////////// powerOn ////////////////////////
    //DebugPrintFlash("========== Power ON\n");
    // selbst
    onPower();
    // module
    pm=pFirstAppModul;
    while(pm)
    {
        pm->onPower();
        pm=pm->pNextObject;
    }
    ////////// Timer 10ms ////////////////////////
    // beachte: default-
```

```
Einstellung in Implementationsbereich
    TIMER10MS_CLASS timer10ms;
    #if TIMER10MS==2        // Timer0 ist nicht geeignet
        // wenn 8bit - Beachte: Rundungsfehler -
> ungenaue 10ms
        timer10ms.config_compareMatch( TIMER10MS_CLASS::s
ourcePrescale1024, F_CPU/1024/100 );
        timer10ms.configInt_compare(true);  // incl. sei(
)
        #warning Timer10ms wahrscheinlich ungenau
        // 1MHz=9,22ms; 3.6864MHz=10ms; 10MHz=9,93ms; 20M
Hz=9,98ms

    #else
        // wenn 16bit
        timer10ms.config_compareMatch( TIMER10MS_CLASS::s
ourcePrescale8, F_CPU/8/100);  // =25000 bei 20MHz; =2000
0 bei 16MHz; =10000 bei 8MHz; =4608 bei 3.6864MHz; ==>> 1
0ms
        timer10ms.configInt_compare(true);  // incl. sei(
)
    #endif

    ////////// powerOn2 ////////////////////////
    //DebugPrintFlash("---------- Start\n");
    // selbst
    onStart();
    // module
    pm=pFirstAppModul;
    while(pm)
    {
        pm->onStart();
        pm=pm->pNextObject;
    }
    ////////// Mainloop ////////////
    //DebugPrintFlash("---------- Mainloop\n");
    while(1)
    {
        // selbst
        onWork();
        // module
        pm=pFirstAppModul;
        while(pm)
        {
            pm->onWork();
            pm=pm->pNextObject;
        }
    }


}
/*////////////////////////////
//
//  onWork()
```

```
//
////////////////////////////
*/
void Controller::onWork()
{


}

////////////////////////////
//
//  onInt_timer10ms()
//
////////////////////////////
void Controller::onInt_timer10ms()
{
    onTimer10ms();
    ////// alle Module: onTimer10ms() ////////
    AppModul* pm=pFirstAppModul;
    while(pm)
    {
        pm->onTimer10ms();
        pm=pm->pNextObject;
    }
    ////// 1 Sekunde //////
    systimeMsec10++;
    // wenn Sekunde um ist
    if(systimeMsec10>=100)  //100 mal 10ms = 1s
    {
        systimeMsec10=0;
        systimeSec++;   // Sekunde hochzählen
        ////// alle Module: onTimer1s() ////////
        pm=pFirstAppModul;
        while(pm)
        {
            pm->onTimer1s();
            pm=pm->pNextObject;
        }
    }
}
```

InfoLed.h

```
/// Code generated by SiSy
///<ObjektNummer>2542</ObjektNummer>

#if !defined(h_InfoLed)
#define h_InfoLed

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include "AppModul.h"

#include "myAVR_App.h"
#include "IrEmpfang.h"
#include "LedSteuerung.h"

#define IRRCR_AnzActions 4
//#define IsDEBUG                 // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class InfoLed : public AppModul
{

public:

    //  flash()
    void flash();
    //  0 = inaktiv
    //  20 = aus
    //  >20 an
    uint8_t timeout;
    //  Konstruktor
    InfoLed();

    //  Destruktor
    ~InfoLed();

private:

protected:

    //  onTimer10ms()
    virtual void onTimer10ms();
};
#endif
```

InfoLed.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2542</ObjektNummer>

////////////////////////////
//
//  Klasse: InfoLed
//
//  Diese Led blinkt wenn ein Signal empfangen wurde.
//
////////////////////////////
#define GeneratedBySisy
#include "InfoLed.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#ifndef LedFlash_Time
#define LedFlash_Time 10
#endif
#define RegPort PORTC
#define RegPin  PINC
#define RegDdr DDRC
#define PortBit BIT4
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


////////////////////////////
//
//  Konstruktor
//
////////////////////////////
InfoLed::InfoLed()
{
timeout = 0;
    //  aus Temlate: LedFlash
    RegDdr |= PortBit;  // Output

}
////////////////////////////
//
//  Destruktor
```

```
//
//////////////////////////////
InfoLed::~InfoLed()
{

}
/*//////////////////////////////
//
//  onTimer10ms()
//
//////////////////////////////
*/
void InfoLed::onTimer10ms()
{
    if(timeout)
    {
        timeout--;
        if(timeout==LedFlash_Time)
            RegPort &= ~PortBit;    // AUS
    }
}
/*//////////////////////////////
//
//  flash()
//
//////////////////////////////
*/
void InfoLed::flash()
{
    // nur wenn nicht an und keine Pause
    if(timeout==0)
    {
        timeout=LedFlash_Time+LedFlash_Time;    // An-
Zeit + Aus-Zeit
        RegPort |= PortBit;     // AN
    }

}
```

LedLight.h

```
/// Code generated by SiSy
///<ObjektNummer>2379</ObjektNummer>

#if !defined(h_LedLight)
#define h_LedLight

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include "AppModul.h"

#include "Lighting.h"

#define IRRCR_AnzActions 4
//#define IsDEBUG                 // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class LedLight : public AppModul
{

public:
    //  stopDimming()
    //  stopt einen eventuellen Dimm-Vorgang
    void stopDimming();
    //  dimTo()
    //  Dimmt die LED bis zum Zielwert.
    //
    //  PE:
    //      wert = Zielwert
    void dimTo(uint16_t wert);
    //  isDimming()
    //  Prüft ob ein selbständiges Dimmen aktiv ist
    //
    //
    //  PA:
    //      true, wenn Dimmen aktiv ist
    bool isDimming();

    //  set()
    void set(uint16_t newValue);

    //  less()
    void less(uint16_t wert);

    //  more()
    void more(uint16_t wert);
```

```
    //  get()
    uint16_t get();
    //  Schrittweite um die beim Dimmen je 10ms geändert
wird.
    uint16_t dimStep;
    //  Bestimmt ein exponentielles Verhalten beim Dimmen
.
    //  0 = lineares Verhalten
    //  1...15 = exponentielles Verhalten, wobei 1 sehr s
tark ist
    uint8_t expValue;
    //  Konstruktor
    LedLight();

    //  Destruktor
    ~LedLight();

private:

protected:

    //  init()
    void init();

    //  setPwm()
    void setPwm(uint16_t value);

    //  onTimer10ms()
    void onTimer10ms();
    bool isDimmingActiv;
    //  Zielwert während eines
    uint16_t dimDestination;
    uint16_t value;



};

#endif
```

LedLight.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2379</ObjektNummer>

#define GeneratedBySisy
#include "LedLight.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#define PwmBits 10
#ifndef LedDimm16_minimum
#define LedDimm16_minimum (0)
#endif
#ifndef LedDimm16_maximum
#define LedDimm16_maximum (0xFFFF)
#endif
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


//////////////////////////////
//
//  Konstruktor
//
//////////////////////////////
LedLight::LedLight()
{
dimStep = 200;
isDimmingActiv = false;
expValue = 8;
value = 0;
    //  aus Temlate: Pwm10bitM8timer1a
    init();

}
//////////////////////////////
//
//  Destruktor
//
//////////////////////////////
LedLight::~LedLight()
{
```

```
}
/*////////////////////////////
//
//  init()
//
////////////////////////////
*/
void LedLight::init()
{
    DDRB|=BIT1; // Ausgang
    Timer1 timer1;
    timer1.config_fastPwm10(Timer1::sourcePrescale1,true)
;


}
/*////////////////////////////
//
//  setPwm()
//
////////////////////////////
*/
void LedLight::setPwm(uint16_t value)
{
    OCR1A=value;
    ///// AUS
    if(value==0)
    {
        Timer1::stop();
        cbi(PORTB,1);   // Low
    }
    ///// AN
    else if(value>=1023)
    {
        Timer1::stop();
        sbi(PORTB,1);   // High
    }
    ///// DIMM
    else
    {
        init();     // Ausgang + PWM
    }


}
/*////////////////////////////
//
//  stopDimming()
//
//  stopt einen eventuellen Dimm-Vorgang
//
////////////////////////////
*/
```

```
void LedLight::stopDimming()
{
    isDimmingActiv=false;


}
/*////////////////////////////
//
//  dimTo()
//
//  Dimmt die LED bis zum Zielwert.
//
//  PE:
//      wert = Zielwert
//
////////////////////////////
*/
void LedLight::dimTo(uint16_t wert)
{
    isDimmingActiv=true;
    dimDestination=wert;


}
/*////////////////////////////
//
//  onTimer10ms()
//
////////////////////////////
*/
void LedLight::onTimer10ms()
{
    if(isDimmingActiv)
    {
        if(value==dimDestination)
            isDimmingActiv=false;
        else
        {
            // schrittweite bestimmen
            uint16_t step=dimStep;
            if( expValue!=0 )
                step+=step>>expValue;
            // Ausführen
            uint16_t diff;
            if(value<dimDestination)
            {
                diff=dimDestination-value;  // max. Step
                if(step>diff)
                    step=diff;
                more(step);
            }
            else // if(value>dimDestination)
            {
                diff=value-dimDestination;  // max. Step
```

```
                if(step>diff)
                    step=diff;
                less(step);
            }
        }
    }


}
/*////////////////////////////
//
//  isDimming()
//
//  Prüft ob ein selbständiges Dimmen gerade aktiv ist.
//
//
//  PA:
//      true, wenn Dimmen aktiv ist
//
////////////////////////////
*/
bool LedLight::isDimming()
{

return isDimmingActiv;
}
/*////////////////////////////
//
//  set()
//
////////////////////////////
*/
void LedLight::set(uint16_t newValue)
{
    if(value!=newValue)
    {
        // merken
        value=newValue;
        // "runden" + setzten
        setPwm( newValue >> (16-PwmBits) );
    }



}
/*////////////////////////////
//
//  less()
//
////////////////////////////
*/
void LedLight::less(uint16_t wert)
{
    if(value>wert && (value-wert)>LedDimm16_minimum )
```

```
        set(value-wert);
    else
        set(LedDimm16_minimum);


}
/*////////////////////////////
//
//  more()
//
////////////////////////////
*/
void LedLight::more(uint16_t wert)
{
    uint16_t neu;
    neu=value+wert;
    if( neu>value && neu<LedDimm16_maximum )
        set(neu);
    else
        set(LedDimm16_maximum);


}
/*////////////////////////////
//
//  get()
//
////////////////////////////
*/
uint16_t LedLight::get()
{

return value;
}
```

BtnLedOn.h

```
/// Code generated by SiSy
///<ObjektNummer>2372</ObjektNummer>

#if !defined(h_BtnLedOn)
#define h_BtnLedOn

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include "AppModul.h"

#include "Lighting.h"

#define IRRCR_AnzActions 4
//#define IsDEBUG                 // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class BtnLedOn : public AppModul
{

public:

    //  getState()
    uint8_t getState();

    //  Enum state
    enum{state_state_Nothing=1,state_state_down,state_sta
te_click,state_state_hold};

    //  changeState_state()
    void changeState_state(state_t newState);

    //  Konstruktor
    BtnLedOn();

    //  Destruktor
    ~BtnLedOn();

private:

protected:

    //  onClick()
    void onClick();

    //  onHold()
    void onHold();
```

```
    //  onTimer10ms()
    virtual void onTimer10ms();

    //  onHoldStart()
    virtual void onHoldStart();

    //  onHoldEnd()
    virtual void onHoldEnd();
    //  Zeit seit erstem Drücken
    uint8_t volatile holdCounter;
    //  Zeit seit erstem Drücken
    uint8_t volatile releaseCounter;

    state_t state;


};

#endif
```

BtnLedOn.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2372</ObjektNummer>

#define GeneratedBySisy
#include "BtnLedOn.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#ifndef ButtonClickDebounce
#define ButtonClickDebounce 5
#endif
#ifndef ButtonHoldTime
#define ButtonHoldTime 100
#endif
#define RegPort PORTD
#define RegPin  PIND
#define RegDdr DDRD
#define PortBit BIT3
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


////////////////////////////
//
//  Konstruktor
//
////////////////////////////
BtnLedOn::BtnLedOn()
{
holdCounter = 0;
releaseCounter = 0;
    //  aus Temlate: ButtonClickAndHold
    RegDdr &= ~PortBit;      // Input
    RegPort |= PortBit;      // Pullup


    state=0;

    changeState_state(state_state_Nothing);
```

```
}
////////////////////////////
//
//  Destruktor
//
////////////////////////////
BtnLedOn::~BtnLedOn()
{


}
/*////////////////////////////
//
//  onClick()
//
////////////////////////////
*/
void BtnLedOn::onClick()
{
    if(ledLight.isDimming())
        ledLight.stopDimming();
    else
        ledLight.dimTo(0xffff);


}
/*////////////////////////////
//
//  onHold()
//
////////////////////////////
*/
void BtnLedOn::onHold()
{
    ledLight.more(200);


}
/*////////////////////////////
//
//  onTimer10ms()
//
////////////////////////////
*/
void BtnLedOn::onTimer10ms()
{
    switch(state)
    {
        case state_state_Nothing:
        {
            if(getState()==0)
            {
                changeState_state(state_state_down);
                break;
```

```
            }
            break;
        }
        case state_state_down:
        {
            if (getState()==0)
            {
                holdCounter++;
                releaseCounter=0;
            }
            else
            {
                holdCounter=0;
                releaseCounter++;
            }

            if(releaseCounter>=ButtonClickDebounce)
            {
                changeState_state(state_state_click);
                break;
            }
            if(holdCounter>=ButtonHoldTime )
            {
                changeState_state(state_state_hold);
                break;
            }
            break;
        }
        case state_state_click:
        {
            changeState_state(state_state_Nothing);
            break;
        }
        case state_state_hold:
        {
            onHold();
            if(getState()!=0)
            {
                changeState_state(state_state_Nothing);
                break;
            }
            break;
        }
    }


}
/*////////////////////////////
//
//  onHoldStart()
//
////////////////////////////
*/
void BtnLedOn::onHoldStart()
```

```
{


}
/*////////////////////////////
//
//  onHoldEnd()
//
////////////////////////////
*/
void BtnLedOn::onHoldEnd()
{


}
/*////////////////////////////
//
//  getState()
//
////////////////////////////
*/
uint8_t BtnLedOn::getState()
{

return RegPin&PortBit;
}

////////////////////////////
//
//  changeState_state()
//
////////////////////////////
void BtnLedOn::changeState_state(state_t newState)
{
    switch(state)
    {
        case state_state_hold:
        {
            onHoldEnd();

            break;
        }
    }
    state = newState;
    switch(state)
    {
        case state_state_down:
        {
            releaseCounter=0;
            holdCounter=0;

            break;
        }
        case state_state_click:
```

```
            {
                onClick();

                break;
            }
            case state_state_hold:
            {
                onHoldStart();

                break;
            }
        }
    }
```

BtnLedOff.h

```
/// Code generated by SiSy
///<ObjektNummer>2413</ObjektNummer>

#if !defined(h_BtnLedOff)
#define h_BtnLedOff

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include "AppModul.h"

#include "Lighting.h"

#define IRRCR_AnzActions 4
//#define IsDEBUG                 // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class BtnLedOff : public AppModul
{

public:

    //  getState()
    uint8_t getState();

    //  Enum state
    enum{state_state_Nothing=1,state_state_down,state_sta
te_click,state_state_hold};

    //  changeState_state()
    void changeState_state(state_t newState);

    //  Konstruktor
    BtnLedOff();

    //  Destruktor
    ~BtnLedOff();

private:

protected:

    //  onHold()
    void onHold();

    //  onClick()
    void onClick();
```

```
    //  onTimer10ms()
    virtual void onTimer10ms();

    //  onHoldStart()
    virtual void onHoldStart();

    //  onHoldEnd()
    virtual void onHoldEnd();
    //  Zeit seit erstem Drücken
    uint8_t volatile holdCounter;
    //  Zeit seit erstem Drücken
    uint8_t volatile releaseCounter;

    state_t state;


};

#endif
```

BtnLedOff.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2413</ObjektNummer>

#define GeneratedBySisy
#include "BtnLedOff.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#ifndef ButtonClickDebounce
#define ButtonClickDebounce 5
#endif
#ifndef ButtonHoldTime
#define ButtonHoldTime 100
#endif
#define RegPort PORTD
#define RegPin  PIND
#define RegDdr DDRD
#define PortBit BIT4
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


////////////////////////////
//
//  Konstruktor
//
////////////////////////////
BtnLedOff::BtnLedOff()
{
holdCounter = 0;
releaseCounter = 0;
    //  aus Temlate: ButtonClickAndHold
    RegDdr &= ~PortBit;      // Input
    RegPort |= PortBit;      // Pullup

    state=0;

    changeState_state(state_state_Nothing);


}
```

```
////////////////////////////
//
//  Destruktor
//
////////////////////////////
BtnLedOff::~BtnLedOff()
{


}
/*////////////////////////////
//
//  onHold()
//
////////////////////////////
*/
void BtnLedOff::onHold()
{
    ledLight.less(200);


}
/*////////////////////////////
//
//  onClick()
//
////////////////////////////
*/
void BtnLedOff::onClick()
{

    if(ledLight.isDimming())
        ledLight.stopDimming();
    else
        ledLight.dimTo(0);


}
/*////////////////////////////
//
//  onTimer10ms()
//
////////////////////////////
*/
void BtnLedOff::onTimer10ms()
{
    switch(state)
    {
        case state_state_Nothing:
        {
            if(getState()==0)
            {
                changeState_state(state_state_down);
                break;
```

```
            }
            break;
        }
        case state_state_down:
        {
            if (getState()==0)
            {
                holdCounter++;
                releaseCounter=0;
            }
            else
            {
                holdCounter=0;
                releaseCounter++;
            }

            if(releaseCounter>=ButtonClickDebounce)
            {
                changeState_state(state_state_click);
                break;
            }
            if(holdCounter>=ButtonHoldTime )
            {
                changeState_state(state_state_hold);
                break;
            }
            break;
        }
        case state_state_click:
        {
            changeState_state(state_state_Nothing);
            break;
        }
        case state_state_hold:
        {
            onHold();
            if(getState()!=0)
            {
                changeState_state(state_state_Nothing);
                break;
            }
            break;
        }
    }


}
/*////////////////////////////
//
//  onHoldStart()
//
////////////////////////////
*/
void BtnLedOff::onHoldStart()
```

```
{


}
/*////////////////////////////
//
//  onHoldEnd()
//
////////////////////////////
*/
void BtnLedOff::onHoldEnd()
{


}
/*////////////////////////////
//
//  getState()
//
////////////////////////////
*/
uint8_t BtnLedOff::getState()
{

return RegPin&PortBit;
}

////////////////////////////
//
//  changeState_state()
//
////////////////////////////
void BtnLedOff::changeState_state(state_t newState)
{
    switch(state)
    {
        case state_state_hold:
        {
            onHoldEnd();

            break;
        }
    }
    state = newState;
    switch(state)
    {
        case state_state_down:
        {
            releaseCounter=0;
            holdCounter=0;

            break;
        }
        case state_state_click:
```

```
            {
                onClick();

                break;
            }
            case state_state_hold:
            {
                onHoldStart();

                break;
            }
        }
    }
```

IrReceiver.h

```
/// Code generated by SiSy
///<ObjektNummer>2487</ObjektNummer>

#if !defined(h_IrReceiver)
#define h_IrReceiver

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include "Timer0.h"
#include "AppModul.h"

#include "myAVR_Timer_m8.h"
#include "myAVR_App.h"

#ifndef IRRCR_AnzActions
#define IRRCR_AnzActions 1      // Anzahl der möglichen g
espeicherten Actions, Standard=1
#endif
#ifndef IRRCR_AnzSamples
#define IRRCR_AnzSamples 100    // Anzahl der Samples Hig
h+Low-Zeiten, Standard=100
#endif
#ifndef IRRCR_Pause
#define IRRCR_Pause 250         // Länge der Pause/Ende,
Standard=250
#endif
#ifndef IRRCR_Noise
#define IRRCR_Noise 1           // länge eines Rausch-
Impulses, Standard=1
#endif
#ifndef IRRCR_MinSize
#define IRRCR_MinSize 10        // Mindest-
Signal in Samples, Standard=10
#endif
#ifndef IRRCR_SampleTolleranz
#define IRRCR_SampleTolleranz 2 // Zeit die je Sample tol
leriert wird, Standard=2
#endif
#include <avr\eeprom.h>

#define IRRCR_AnzActions 4
//#define IsDEBUG               // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class IrReceiver : public Timer0,public AppModul
{
```

```
public:
    //  ID der Aktion die gelernt werden soll, 0xFF für k
ein Lernen
    uint8_t learnAction;
    //  Konstruktor
    IrReceiver();

    //  Destruktor
    ~IrReceiver();

private:

protected:

    //  onPower()
    void onPower();

    //  onAction()
    void onAction(uint8_t action);

    //  onLearnEnd()
    void onLearnEnd();

    //  onSampleEnd()
    void onSampleEnd();

    //  onWork()
    void onWork();
    //  compareSignal()
    //  vergleicht sample[..] mit den gespeicherten Aktio
nen
    //
    //
    //  PA:
    //      0xFF = keine Übereinstimmung
    //      0... = Übereinstimmung mit action[0...]
    uint8_t compareSignal();
    //  onPinChange()
    //  muss aufgerufen werden, wenn ein Signalwechsel am
 IR-Decoder erfolgte
    void onPinChange();
    //  zeitg an, das auf das erste Bit gewartet wird
    bool firstBit;
    //  alter Pin-Status
    uint8_t oldPinState;
    //  Puffer für der laufenden Empfang,
    //  im ersten Byte steht die Anzahl der Samples.
    uint8_t samples[IRRCR_AnzSamples+1];



};
```

```
#endif
```

IrReceiver.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2487</ObjektNummer>

#define GeneratedBySisy
#include "IrReceiver.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#define RegPort PORTC
#define RegPin  PINC
#define RegDdr DDRC
#define PortBit BIT5
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


//  Puffer für gespeicherte Aktionen,
//  im ersten Byte steht die Anzahl der Samples.
uint8_t EEMEM   IrReceiverActions[IRRCR_AnzActions][IRRCR
_AnzSamples+1];
////////////////////////////
//
//  Konstruktor
//
////////////////////////////
IrReceiver::IrReceiver()
{
learnAction = 0xff;
firstBit = true;
oldPinState = 0;
    //  aus Temlate: IrRemoteControlReceiver
    RegDdr &= ~PortBit;     // Input
    RegPort |= PortBit;     // Pullup
    // je Aktion, Signal=0 lang
    //for( uint8_t a=0; a<IRRCR_AnzActions; a++)
    //  actions[a][0]=0;
```

```
}
////////////////////////////
//
//  Destruktor
//
////////////////////////////
IrReceiver::~IrReceiver()
{

}
/*////////////////////////////
//
//  onPower()
//
////////////////////////////
*/
void IrReceiver::onPower()
{
    config_overflow(Timer0::sourcePrescale256);     // =
14400Hz
    //config_overflow(Timer0::sourcePrescale64);        /
/ = 57600Hz


}
/*////////////////////////////
//
//  onAction()
//
////////////////////////////
*/
void IrReceiver::onAction(uint8_t action)
{
    app.onIrAction(action);


}
/*////////////////////////////
//
//  onLearnEnd()
//
////////////////////////////
*/
void IrReceiver::onLearnEnd()
{
    btnProg.ledProg.setBlinkCode(0);
    app.infoLed.flash();


}
/*////////////////////////////
//
//  onSampleEnd()
//
```

```
////////////////////////////
*/
void IrReceiver::onSampleEnd()
{
    //  aus Temlate: IrRemoteControlReceiver
    // kein lernen
    if(learnAction==0xFF)
    {
        uint8_t action=compareSignal();
        // wenn erkannte action
        if(action!=0xFF)
            onAction(action);
    }
    // lernen -> Speichern in Eeprom
    else if(learnAction<IRRCR_AnzActions)
    {
        // speichern des Empfangenen
        //memcpy( actions[learnAction], samples, IRRCR_An
zSamples+1 );
        uint8_t* pa=&IrReceiverActions[learnAction][0];
        uint8_t* ps=&samples[0];
        for(uint8_t i=0;i<IRRCR_AnzSamples+1;i++)
        {
            eeprom_write_byte( pa, *ps);
            pa++;
            ps++;
        }
        // fertig
        onLearnEnd();
        learnAction=0xff;   // Learn aus
    }


    //app.infoLed.flash();


}
/*////////////////////////////
//
//  onWork()
//
////////////////////////////
*/
void IrReceiver::onWork()
{
    /// wenn Flankenwechsel
    if( (RegPin&PortBit) != oldPinState)
    {
        oldPinState = RegPin&PortBit;
        onPinChange();
    }
    /// wenn Timeout
    if(getCounter()>IRRCR_Pause)
    {
```

```
            // Fertig
            if(samples[0]>=IRRCR_MinSize)
            {
                onSampleEnd();
            }
            // Restart
            samples[0]=0;
            firstBit=true;
            setCounter(0);
            oldPinState=0;
        }
        /// wenn Overflow
        if(samples[0]>=IRRCR_AnzSamples)
        {
            // Restart
            samples[0]=0;
            firstBit=true;
            setCounter(0);
            oldPinState=0;
        }


}
/*////////////////////////////
//
//  compareSignal()
//
//  vergleicht sample[..] mit den gespeicherten Aktionen
//
//
//  PA:
//      0xFF = keine Übereinstimmung
//      0... = Übereinstimmung mit action[0...]
//
////////////////////////////
*/
uint8_t IrReceiver::compareSignal()
{
    uint8_t action=0xFF;    // PA, = nichts passend

    uint8_t anz,ist,soll,unten,oben;
    uint8_t* pAction;
    uint8_t* pSample;

    // while (!(UCSRA&32)){};UDR=□*□;   /////############
########
    // while (!(UCSRA&32)){};UDR=□*□;   /////############
########
    // while (!(UCSRA&32)){};UDR=□*□;   /////############
########
    // while (!(UCSRA&32)){};UDR=□*□;   /////############
########
    // while (!(UCSRA&32)){};UDR=□*□;   /////############
```

```
########
    // while (!(UCSRA&32)){};UDR=□*□;   /////############
########


    // je Aktion
    for( uint8_t a=0; a<IRRCR_AnzActions && action==0xFF;
 a++)
    {
        pSample=&samples[0];
    // if(a==0)while (!(UCSRA&32)){};UDR=((uint16_t)pSamp
le)>>8;    /////####################
    // if(a==0)while (!(UCSRA&32)){};UDR=((uint16_t)pSamp
le)&0xFF;  /////####################
        pAction=&IrReceiverActions[a][0];
    // if(a==0)while (!(UCSRA&32)){};UDR=((uint16_t)pActi
on)>>8;    /////####################
    // if(a==0)while (!(UCSRA&32)){};UDR=((uint16_t)pActi
on)&0xFF;  /////####################
        // Anzahl muss stimmen
        anz=eeprom_read_byte(pAction);
    // if(a==0)while (!(UCSRA&32)){};UDR=*pSample;  /////
####################
    // if(a==0)while (!(UCSRA&32)){};UDR=anz;   /////####
################
        if(anz!=*pSample)
            continue;
        // je Sample vergleichen
        pAction++;
        pSample++;
        action=a;   // = ggf. passend
    //  for(uint8_t x=0; x<anz; x++)
        for(uint8_t x=0; x<anz-
1; x++)  // das letzte ist evt. fehlerhaft
        {
            soll=eeprom_read_byte(pAction);
            ist =*pSample;
    // if(a==0)while (!(UCSRA&32)){};UDR=soll;  /////####
################
    // if(a==0)while (!(UCSRA&32)){};UDR=ist;   /////####
################
            pAction++;
            pSample++;
            // gültigkeitsbereich
            unten= soll-IRRCR_SampleTolleranz;
            if(unten>soll)  // wenn überlauf
                unten=0;
            oben = soll+IRRCR_SampleTolleranz;
            if(oben<soll)   // wenn überlauf
                oben=255;
            // wenn außerhalb der Tolleranz
            if( ist<unten || ist>oben)
            {
                action=0xFF; // nicht passend
```

```
                break;
            }
        }
    }

return action;
}
/*////////////////////////////
//
//  onPinChange()
//
//  muss aufgerufen werden, wenn ein Signalwechsel am IR-
Decoder erfolgte
//
////////////////////////////
*/
void IrReceiver::onPinChange()
{
    // Zeit holen
    uint8_t zeit=getCounter();
    if(zeit>IRRCR_Noise)    // mini-flackern unterdrücken
    {
        // neue Zeitmessung starten, ###ggf timer anhalte
n
        setCounter(0);

        // das erste ist die Wartezeit bis Beginn -
> uninteressant
        ///// starten
        if(firstBit)
        {
            samples[0]=0;
            firstBit=false;
        }
        ///// speichern
        else
        {
            // aktuellen offset holen
            uint8_t sampleOffset=samples[0];
            // speichern
            samples[sampleOffset]=zeit;
            // hochzählen
            sampleOffset++;
            samples[0]=sampleOffset;
        }
    }



}
```

BtnProg.h

```
/// Code generated by SiSy
///<ObjektNummer>2462</ObjektNummer>

#if !defined(h_BtnProg)
#define h_BtnProg

// Defines
#define F_CPU 7372800
#define MCU ATMEGA8

#include <avr\io.h>
#include "LedProg.h"
#include "AppModul.h"

#include "myAVR_Timer_m8.h"
#include "myAVR_App.h"

#define IRRCR_AnzActions 4
//#define IsDEBUG                 // erlaubt DebugAusgaben
//#define DEBUG_UART_NR 0
//#define DEBUG_BAUD 115200
//#include "debugUart.h"


class BtnProg : public AppModul
{

public:

    //  onTimer10ms()
    void onTimer10ms();
    LedProg ledProg;
    //  Konstruktor
    BtnProg();

    //  Destruktor
    ~BtnProg();

private:

protected:

    //  onClick()
    void onClick();
    //  Null = freilauf, sonst Zeit seit letztem loslasse
n
    uint8_t debounce;
};

#endif
```

BtnProg.cpp

```
/// Code generated by SiSy
///<ObjektNummer>2462</ObjektNummer>

#define GeneratedBySisy
#include "BtnProg.h"

#include "Controller.h"
#include "BtnProg.h"
#include "IrReceiver.h"
#include "LedLight.h"
#include "BtnLedOn.h"
#include "BtnLedOff.h"

#define TIMER10MS 2

#define RegPort PORTD
#define RegPin  PIND
#define RegDdr DDRD
#define PortBit BIT2
extern  Controller app;
extern  BtnProg btnProg;
extern  IrReceiver irReceiver;
extern  LedLight ledLight;
extern  BtnLedOn btnLedOn;
extern  BtnLedOff btnLedOff;


//////////////////////////////
//
//  Konstruktor
//
//////////////////////////////
BtnProg::BtnProg()
{
debounce = 0;
    //  aus Temlate: ButtonClick
    RegDdr &= ~PortBit;     // Input
    RegPort |= PortBit;     // Pullup


}
//////////////////////////////
//
//  Destruktor
//
//////////////////////////////
BtnProg::~BtnProg()
{

}
/*//////////////////////////////
//
```

```
//  onClick()
//
//////////////////////////////
*/
void BtnProg::onClick()
{
    // Blinkcodes durchschalten Aus,1,2,3,4
    uint8_t state;
    state=ledProg.getBlinkCode();
    state++;
    if(state>4)
        state=0;
    ledProg.setBlinkCode(state);

    // an Empfänger
    irReceiver.learnAction=state-1;


}
/*//////////////////////////////
//
//  onTimer10ms()
//
//////////////////////////////
*/
void BtnProg::onTimer10ms()
{
    #define ButtonClickDebounce 6
    // wenn frisch gedrückt
    if( debounce==0 && (RegPin&PortBit)==0 )
    {
        debounce=1;
        onClick();
    }
    // wenn nicht gedrückt -
> warten auf Zeitspanne ohne Tastendruck
    else if( debounce!=0 )
    {
        // wenn gedrückt
        if( (RegPin&PortBit)==0)
            debounce=1;
        // wenn losgelassen und timeout
        else
        {
            debounce++;
            // wenn Zeit um -> Neuer Zyklus
            if( debounce==ButtonClickDebounce )
                debounce=0;
        }
    }
}
```