GPIO_CPP_SEV

Generated by Doxygen 1.9.7

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

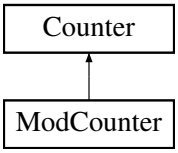
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Counter Class Reference

Inheritance diagram for Counter:



Public Member Functions

- **Counter** (int v=0)
- void **Inc** ()
- int **GetValue** ()

Protected Attributes

- int **count**

The documentation for this class was generated from the following file:

- Counter.h

4.2 GPIO Class Reference

```
#include <GPIO.h>
```

Public Member Functions

- GPIO (uint32_t GPIOx_BASE)
- void configurePin (uint8_t pin, uint8_t mode, uint8_t outputType, uint8_t speed, uint8_t pull)
- void configurePort (uint16_t mask)
- void setPin (uint16_t pin)
- void resetPin (uint16_t pin)
- void togglePin (uint16_t pin)
- void setPort (uint16_t mask)
- uint16_t getPort ()

4.2.1 Detailed Description

Class to directly control the GPIO Ports and Pins

4.2.2 Constructor & Destructor Documentation

4.2.2.1 GPIO()

```
GPIO::GPIO (
            uint32_t GPIOx_BASE )  [inline]
```

Parameters

GPIOx_BASE	The Base Adress of the GPIO Port

4.2.3 Member Function Documentation

4.2.3.1 configurePin()

```
void GPIO::configurePin (
            uint8_t pin,
            uint8_t mode,
            uint8_t outputType,
            uint8_t speed,
            uint8_t pull )  [inline]
```

Parameters

pin	The pin to be configured
mode	0 for Input, 1 for Output
outputType	0 for Push-Pull, 1 for Open-Drain
speed	0 for low speed
0	no Pull-Up/Pull-down, 1 for Pull-Up, 2 for Pull-Down

4.2.3.2 configurePort()

```
void GPIO::configurePort (
            uint16_t mask )  [inline]
```

Sets each pin of the port to either input or output Other Settings are: Push-Pull, 2MHz low speed, no Pull-Up/Pull-Down

Parameters

mask	The mask to set the individual pins, 0 for input, 1 for Output

4.2.3.3 getPort()

```
uint16_t GPIO::getPort ( )  [inline]
```

Get the states of all inputs of a specific port

Returns

The currently active inputs

4.2.3.4 resetPin()

```
void GPIO::resetPin (
            uint16_t pin )  [inline]
```

Sets the desired output Pin to low

Parameters

pin	The Pin to be reset

4.2.3.5 setPin()

```
void GPIO::setPin (
            uint16_t pin )  [inline]
```

Sets the desired Output Pin to High

Parameters

pin	The Pin to be set

4.2.3.6 setPort()

```
void GPIO::setPort (
            uint16_t mask )  [inline]
```

Sets multiple Output Pins at once, depending on the mask

Parameters

mask	Mask to select which Pins to set

4.2.3.7 togglePin()

```
void GPIO::togglePin (
            uint16_t pin )  [inline]
```

Toggles the desired Output Pin

Parameters

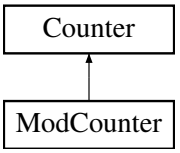
pin	The Pin to be toggled

The documentation for this class was generated from the following file:

- GPIO.h

4.3 ModCounter Class Reference

Inheritance diagram for ModCounter:



Public Member Functions

- **ModCounter** (int _modVal=2)
- void **Inc** ()

Public Member Functions inherited from Counter

- **Counter** (int v=0)
- void **Inc** ()
- int **GetValue** ()

Additional Inherited Members

Protected Attributes inherited from Counter

- int **count**

The documentation for this class was generated from the following file:

- ModCounter.h

4.4 Motor Class Reference

Public Member Functions

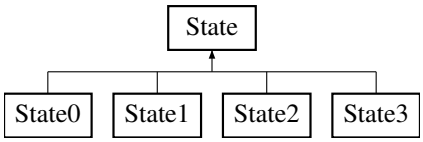
- void **setMotorState** (uint8_t motorState)

The documentation for this class was generated from the following files:

- Motor.h
- Motor.cpp

4.5 State Class Reference

Inheritance diagram for State:



Public Member Functions

- **State** (uint8_t currCnt, uint8_t strdCnt)
- virtual void **performStateLogic** ()=0
- virtual State * **transitionToNextState** ()=0

Protected Member Functions

- void **Init** ()

Protected Attributes

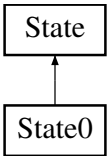
- volatile const uint8_t * **currCntVal**
- volatile const uint8_t * **strdCntVal**
- Motor * **MotorLED**

The documentation for this class was generated from the following files:

- StateMachine.h
- StateMachine.cpp

4.6 State0 Class Reference

Inheritance diagram for State0:



Public Member Functions

- **State0** (uint8_t currCnt, uint8_t strdCnt)
- void performStateLogic () override
- State * transitionToNextState () override

Public Member Functions inherited from State

- **State** (uint8_t currCnt, uint8_t strdCnt)
- virtual void **performStateLogic** ()=0
- virtual State * **transitionToNextState** ()=0

Additional Inherited Members

Protected Member Functions inherited from State

- void **Init** ()

Protected Attributes inherited from State

- volatile const uint8_t * **currCntVal**
- volatile const uint8_t * **strdCntVal**
- Motor * **MotorLED**

4.6.1 Member Function Documentation

4.6.1.1 performStateLogic()

```
void State0::performStateLogic ( )  [override], [virtual]
```

Implements State.

4.6.1.2 transitionToNextState()

```
State * State0::transitionToNextState ( )  [override], [virtual]
```

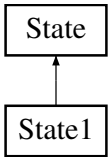
Implements State.

The documentation for this class was generated from the following files:

- StateMachine.h
- StateMachine.cpp

4.7 State1 Class Reference

Inheritance diagram for State1:



Public Member Functions

- void performStateLogic () override
- State * transitionToNextState () override

Public Member Functions inherited from State

- **State** (uint8_t currCnt, uint8_t strdCnt)
- virtual void **performStateLogic** ()=0
- virtual State * **transitionToNextState** ()=0

Additional Inherited Members

Protected Member Functions inherited from State

- void **Init** ()

Protected Attributes inherited from State

- volatile const uint8_t * **currCntVal**
- volatile const uint8_t * **strdCntVal**
- Motor * **MotorLED**

4.7.1 Member Function Documentation

4.7.1.1 performStateLogic()

```
void State1::performStateLogic ( )  [override], [virtual]
```

Implements State.

4.7.1.2 transitionToNextState()

```
State * State1::transitionToNextState ( )  [override], [virtual]
```

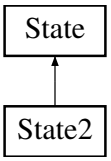
Implements State.

The documentation for this class was generated from the following files:

- StateMachine.h
- StateMachine.cpp

4.8 State2 Class Reference

Inheritance diagram for State2:



Public Member Functions

- void performStateLogic () override
- State * transitionToNextState () override

Public Member Functions inherited from State

- **State** (uint8_t currCnt, uint8_t strdCnt)
- virtual void **performStateLogic** ()=0
- virtual State * **transitionToNextState** ()=0

Additional Inherited Members

Protected Member Functions inherited from State

- void **Init** ()

Protected Attributes inherited from State

- volatile const uint8_t * **currCntVal**
- volatile const uint8_t * **strdCntVal**
- Motor * **MotorLED**

4.8.1 Member Function Documentation

4.8.1.1 performStateLogic()

```
void State2::performStateLogic ( )  [override], [virtual]
```

Implements State.

4.8.1.2 transitionToNextState()

```
State * State2::transitionToNextState ( )  [override], [virtual]
```

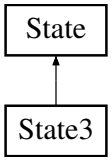
Implements State.

The documentation for this class was generated from the following files:

- StateMachine.h
- StateMachine.cpp

4.9 State3 Class Reference

Inheritance diagram for State3:



Public Member Functions

- void performStateLogic () override
- State * transitionToNextState () override

Public Member Functions inherited from State

- **State** (uint8_t currCnt, uint8_t strdCnt)
- virtual void **performStateLogic** ()=0
- virtual State * **transitionToNextState** ()=0

Additional Inherited Members

Protected Member Functions inherited from State

- void **Init** ()

Protected Attributes inherited from State

- volatile const uint8_t * **currCntVal**
- volatile const uint8_t * **strdCntVal**
- Motor * **MotorLED**

4.9.1 Member Function Documentation

4.9.1.1 performStateLogic()

```
void State3::performStateLogic ( )  [override], [virtual]
```

Implements State.

4.9.1.2 transitionToNextState()

```
State * State3::transitionToNextState ( )  [override], [virtual]
```

Implements State.

The documentation for this class was generated from the following files:

- StateMachine.h
- StateMachine.cpp

4.10 UserButton Class Reference

Public Member Functions

- void **Init** ()
- bool **readButtonState** (uint16_t pin)
- char **Pressed** ()

The documentation for this class was generated from the following file:

- UserButton.h

4.11 UserLED Class Reference

Public Member Functions

- **UserLED** (uint32_t passPort)
- void **SetValue** (uint8_t bit8Stream)

The documentation for this class was generated from the following files:

- UserLED.h
- UserLED.cpp

Chapter 5

File Documentation

5.1 Counter.h

```
00001 //Counter.h
00002
00003 #ifndef _COUNTER_
00004 #define _COUNTER_
00005 class Counter
00006 {
00007 protected:
00008   int count;
00009
00010 public:
00011   Counter(int v=0) : count(v) {}
00012
00013   void Inc() {count++;};
00014   int GetValue() {return count;};
00015 };
00016 #endif
```

5.2 Def.h

```
00001 // Def.h
00002 /*
00003 Namespace for Standard-Parameters. Usage:
00004
00005 - Preprocessor -
00006 #include "Def.h"
00007 using namespace Def;(optional)
00008
00009 - Code -
00010 Def::<enumClass>::<Enum>
00011 */
00012
00013 #ifndef _DEF_
00014 #define _DEF_
00015 namespace Def
00016 {
00017     const enum enumPort
00018     {
00019         PortA = 0x40020000,
00020         PortB = 0x40020000,
00021     };
00022
00023     const enum enumLED
00024     {
00025         Bin0 = 0b00000000,
00026         Bin1,
00027         Bin2,
00028         Bin3,
00029         Bin4,
00030         Bin5,
00031         Bin6,
00032         Bin7,
00033     };
00034
00035     const enum enumButton
```

```
00036     {
00037
00038     };
00039
00040     const enum enumMotorZustand
00041     {
00042         Still = 0,
00043         Rauf = 1,
00044         Runter = 2
00045     };
00046 };
00047 #endif
```

5.3 GPIO.h

```
00001 #include <stdint.h>
00002 #include <stdio.h>
00003 #include "Def.h"
00004
00005 #ifndef _GPIO_
00006 #define _GPIO_
00010 class GPIO
00011 {
00012   private:
00013     volatile uint32_t* GPIOx_MODER;
00014     volatile uint32_t* GPIOx_OTYPER;
00015     volatile uint32_t* GPIOx_OSPEEDR;
00016     volatile uint32_t* GPIOx_PUPDR;
00017     volatile uint32_t* GPIOx_IDR;
00018     volatile uint32_t* GPIOx_ODR;
00019     volatile uint32_t* GPIOx_BSRRL;
00020     volatile uint32_t* GPIOx_BSRRH;
00021     volatile uint32_t* RCC_AHB1ENR;
00022 public:
00026   GPIO(uint32_t GPIOx_BASE)
00027     : GPIOx_MODER((volatile uint32_t*) GPIOx_BASE),
00028       GPIOx_OTYPER(GPIOx_MODER + 1),
00029       GPIOx_OSPEEDR(GPIOx_MODER + 2),
00030       GPIOx_PUPDR(GPIOx_MODER + 3),
00031       GPIOx_IDR(GPIOx_MODER + 4),
00032       GPIOx_ODR(GPIOx_MODER + 5),
00033       GPIOx_BSRRL((volatile uint32_t*) (GPIOx_BASE + 0x18)),
00034       GPIOx_BSRRH((volatile uint32_t*) (GPIOx_BASE + 0x1A)),
00035       RCC_AHB1ENR((volatile uint32_t*) 0x40023830)
00036   {
00037     // Calculate which CLock should be activated
00038     uint32_t registerByte = (GPIOx_BASE & 0xFF00) » 8;
00039     int registerNumber = (registerByte) / 0x04;
00040     // Activate the corresponding Bit
00041     *RCC_AHB1ENR |= (0x01 « registerNumber);
00042   }
00051    void configurePin(uint8_t pin, uint8_t mode, uint8_t outputType, uint8_t speed, uint8_t pull) {
00052         // Calculate register offset for the specified pin
00053         uint32_t offset = pin * 2;
00054
00055         // Configure pin mode
00056         *GPIOx_MODER &= ~(0x03 « offset);    // Clear mode bits
00057         *GPIOx_MODER |= (mode « offset);     // Set mode bits
00058
00059         // Configure output type
00060         *GPIOx_OTYPER &= ~(0x01 « pin);      // Clear output type bit
00061         *GPIOx_OTYPER |= (outputType « pin); // Set output type bit
00062
00063         // Configure output speed
00064         *GPIOx_OSPEEDR &= ~(0x03 « offset);  // Clear speed bits
00065         *GPIOx_OSPEEDR |= (speed « offset);  // Set speed bits
00066
00067         // Configure pull-up/pull-down
00068         *GPIOx_PUPDR &= ~(0x03 « offset);    // Clear pull bits
00069         *GPIOx_PUPDR |= (pull « offset);     // Set pull bits
00070     }
00071
00077     void configurePort(uint16_t mask){
00078       if (GPIOx_MODER == (volatile uint32_t*)Def::enumPort::PortA){
00079         for (int i = 0; i < 16; ++i) {
00080           int bit = (mask » i) & 1;
00081           if(i == 13 or i == 14){
00082             configurePin(i, 2, 0, 0, 0);
00083           }
00084           else{
00085             configurePin(i, bit, 0, 0, 0);
00086           }
00087         }
```

```
00088       }
00089       else{
00090         for (int i = 0; i < 16; ++i) {
00091           int bit = (mask >> i) & 1;
00092           configurePin(i, bit, 0, 0, 0);
00093         }
00094       }
00095     }
00096
00101    void setPin(uint16_t pin) {
00102         *GPIOx_BSRRL |= (0x01 << pin);  // Set pin
00103     }
00104
00109     void resetPin(uint16_t pin) {
00110         *GPIOx_BSRRH |= (0x01 << pin);  // Reset pin
00111     }
00112
00117     void togglePin(uint16_t pin) {
00118       *GPIOx_ODR ^= (0x01 << pin);
00119     }
00120
00125     void setPort(uint16_t mask){
00126       *GPIOx_ODR = mask;
00127     }
00128
00133     uint16_t getPort(){
00134
00135       return *GPIOx_IDR & 0xFFFF;
00136     }
00137 };
00138 #endif
00139
00140
```

5.4 ModCounter.h

```
00001 //ModCounter.h
00002 #include "counter.h"
00003
00004 #ifndef _MODCOUNTER_
00005 #define _MODCOUNTER_
00006 class ModCounter : public Counter
00007 {
00008 private:
00009   int modVal;
00010
00011 public:
00012   ModCounter(int _modVal = 2) : modVal(_modVal) {}
00013
00014   void Inc() {count = (count + 1)%modVal;}
00015
00016 };
00017 #endif
```

5.5 Motor.h

```
00001 #include <iostream>
00002 #include "UserLED.h"
00003
00004 using namespace std;
00005
00006 #ifndef _MOTOR_
00007 #define _MOTOR_
00008 class Motor
00009 {
00010 private:
00011     UserLED *DispLED;
00012
00013 public:
00014     Motor();
00015     void setMotorState(uint8_t motorState);
00016 };
00017 #endif
```

5.6 StateMachine.h

```
00001 #include "Motor.h"
```

```
00002
00003 class State {
00004 protected:
00005     volatile const uint8_t* currCntVal;
00006     volatile const uint8_t* strdCntVal;
00007     Motor *MotorLED;
00008     void Init();
00009 public:
00010     State(uint8_t currCnt, uint8_t strdCnt);
00011     virtual void performStateLogic() = 0;
00012     virtual State* transitionToNextState() = 0;
00013 };
00014
00015 /*Stillstand*/
00016 class State0 : public State {
00017 public:
00018     State0(uint8_t currCnt, uint8_t strdCnt);
00019     void performStateLogic() override;
00020     State* transitionToNextState() override;
00021 };
00022
00023 /*Nach oben*/
00024 class State1 : public State {
00025 public:
00026     void performStateLogic() override;
00027     State* transitionToNextState() override;
00028 };
00029
00030 /*Nach Unten*/
00031 class State2 : public State {
00032 public:
00033     void performStateLogic() override;
00034     State* transitionToNextState() override;
00035 };
00036
00037 /*Störung*/
00038 class State3 : public State {
00039 public:
00040     void performStateLogic() override;
00041     State* transitionToNextState() override;
00042 };
```

5.7 UserButton.h

```
00001 //UserButton.h
00002 #include "GPIO.h"
00003
00004 #ifndef _USERBUTTON_
00005 #define _USERBUTTON_
00006 class UserButton
00007 {
00008 private:
00009   unsigned long ButtonPortAddr;
00010   GPIO *ButtonPort;
00011
00012   const int debounceDelay = 50;
00013   unsigned long debounceCounter;
00014   uint16_t buttonState;
00015   uint16_t lastButtonState;
00016
00017 public:
00018   //TODO: change address to PortB
00019   UserButton():ButtonPortAddr(0x40020000)
00020   {
00021     ButtonPort = new GPIO(ButtonPortAddr);
00022     //Init();
00023   }
00024
00025   void Init()
00026   {
00027     ButtonPort->configurePort(0xFFFF);
00028   }
00029
00030   bool readButtonState(uint16_t pin)
00031   {
00032     uint16_t readMask = 0x01 « pin;
00033     return (bool)(ButtonPort->getPort() & readMask);
00034   }
00035
00036
00037   char Pressed()
00038   {
00039     //check current state of the user-input
```

```
00040     bool StartButton  = readButtonState(9); //TODO: change to needed pin number
00041     bool SelectButton = readButtonState(10); //TODO: change to needed pin number
00042     //bool SensorButton = readButtonState(10);
00043
00044     if      (StartButton && SelectButton)
00045       return 'X';
00046     else if (StartButton && !SelectButton)
00047       return 'B';
00048     else if (!StartButton && SelectButton)
00049       return 'F';
00050     else
00051       return 0;
00052   }
00053 };
00054 #endif
```

5.8 UserLED.h

```
00001 #include "GPIO.h"
00002
00003
00004 #ifndef _USERLED_
00005 #define _USERLED_
00006 class UserLED
00007 {
00008 private:
00009     const uint32_t LEDPortAddr;
00010     uint16_t setPortStream;
00011     GPIO *LEDPort;
00012     void Init();
00013
00014 public:
00015     UserLED(uint32_t passPort);
00016     //~UserLED();
00017     void SetValue(uint8_t bit8Stream);
00018 };
00019 #endif
```

Index