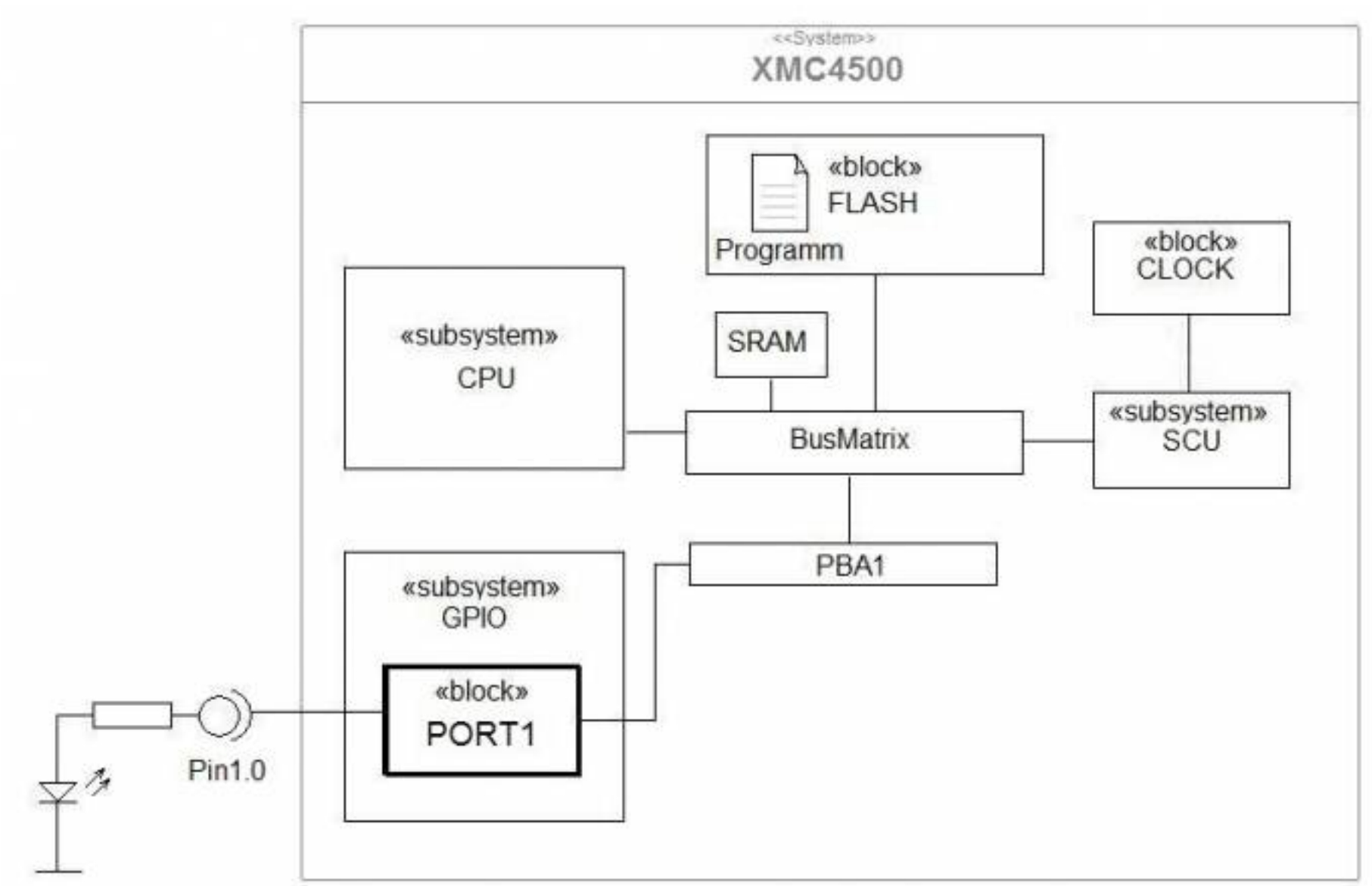
XMC Tutorial für C/C++ und UML

Hallo XMC mit einfachem C

Die erste Übung in jedem Programmierkurs ist das berühmte „Hallo Welt“. Damit wird versucht, dem Lernenden ein motivierendes **„AHA-Erlebnis“** zu vermitteln. OK … mal sehen, ob wir das auch hin bekommen. Bei der Programmierung von eingebetteten Systemen besteht oft das Problem, dass kein Bildschirm oder Display zur Textausgabe angeschlossen ist. Dann stehen für das „sich bemerkbar machen“ dem System nur LEDs zur Verfügung. Also leuchten und blinken eingebettete Systeme somit ihre Botschaft in die Welt. Ganz nebenbei lernen wir in diesem Abschnitt sehr viel über die digitale Ausgabe des XMC. Wir betrachten zuerst die Programmierung des XMC auf der Ebene der internen Register, also die hard core Variante. Dann schauen wir uns an wie die Lösung aussieht wenn wir die *Infineon XMC Low Level Treiber* benutzen.

Die Aufgabe

Die erste Übung soll das typische LED Einschalten sein. Dazu nutzen wir eine der LEDs auf dem XMC4500 Relax Kit. Die LED ist bereits fest mit dem *Pin1.0* verbunden.

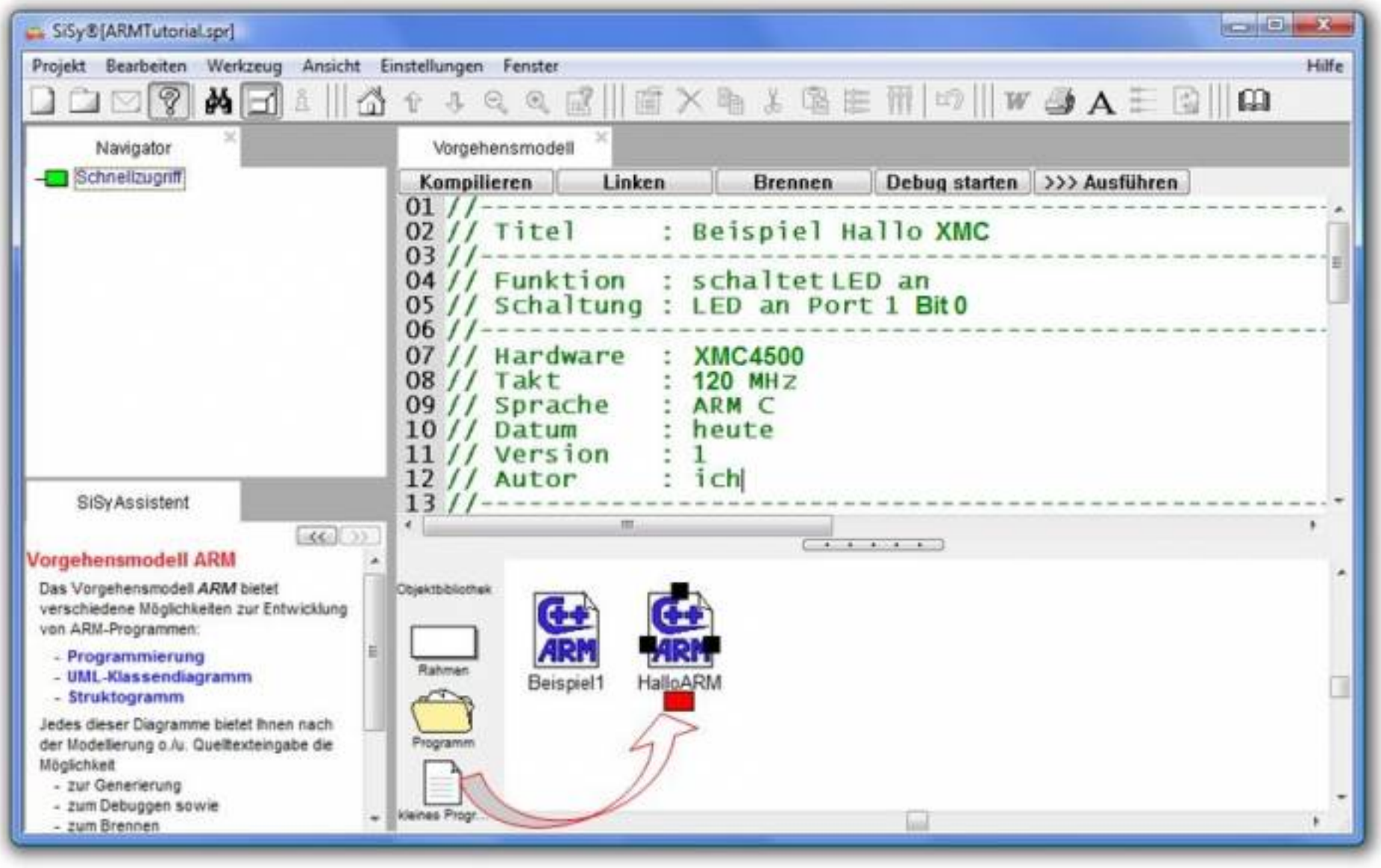


Die Aufgabe besteht darin:

1. Port 1 Bit 0 als Ausgang zu konfigurieren
2. und das Pin auf High zu schalten

Vorbereitung

Falls das Tutorial-Projekt nicht mehr offen ist, öffnen Sie dies. Legen Sie bitte ein neues *kleines Programm* an und laden das *Grundgerüst ARM C++ Anwendung*. Beachten Sie die Einstellungen für die Zielplattform *XMC4500 Relax Kit*.

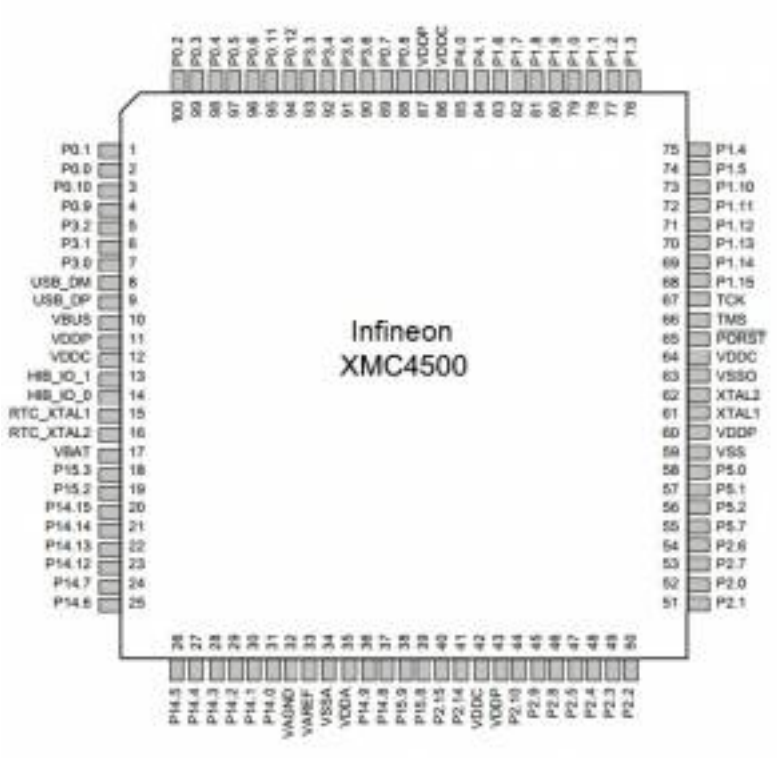


Erstellen Sie die Programmkopfdokumentation. Übersetzen und übertragen Sie das noch leere Programm auf den Controller, um die Verbindung zu testen.

```
//------------------------------------------------------------------
// Titel     : Beispiel Hallo Welt mit SiSy XMC
//------------------------------------------------------------------
// Funktion  : schaltet eine LED an
// Schaltung : LED an Port 1 Bit 0
//------------------------------------------------------------------
// Hardware  : XMC4500
// Takt      : 120 MHz
// Sprache   : ARM C++
// Datum     : heute
// Version   : 1
// Autor     : ich
//------------------------------------------------------------------
```

Grundlagen

Unter *GPIO (General Purpose Input/Output)* verstehen wir zunächst einmal einen allgemeinen Pin, der als Kontakt aus dem Controllergehäuse herausgeführt ist. Dieser hat nach dem Einschalten bzw. RESET, also dem Start des Controllers, keine konkrete Funktion. Die gewünschte Funktion des GPIO-Pins kann vom Programmierer eines ARM verhältnismäßig frei festgelegt werden. Die Freiheit bewegt sich natürlich nur innerhalb der Möglichkeiten der Bus- und Cross-Connect-Matrix des jeweiligen Controllers. Die meisten GPIO-Pins sind geeignet digitale Ein- oder Ausgaben auszuführen. Dafür verfügen Sie über entsprechende Treiberstufen. Die Leistung der Ausgangstreiberstufen muss im Datenblatt des jeweiligen Controllers nachgeschlagen werden. Beim XMC4500 kann ein Pin in der Regel mit bis zu 10 mA dauerhaft belastet werden. Die Eingangstreiber sind hochohmig. Andere Funktionen, als die einfache per Software gesteuerte digitale Ein- und Ausgabe, werden *Alternativfunktionen* genannt. das können Analogfunktionen (ADC, DAC), aber auch spezielle Peripheriefunktionen (USART, SPI, I2C, uvm.) sein. Beim XMC werden jeweils 16 Pins zu einem GPIO-Port zusammengefasst. Die Ports werden durchnummeriert. Der XMC4500 verfügt, je nach Gehäuseform, über die GPIO-Ports *PORT0* bis *PORT6* sowie über *PORT14* und *PORT15*.

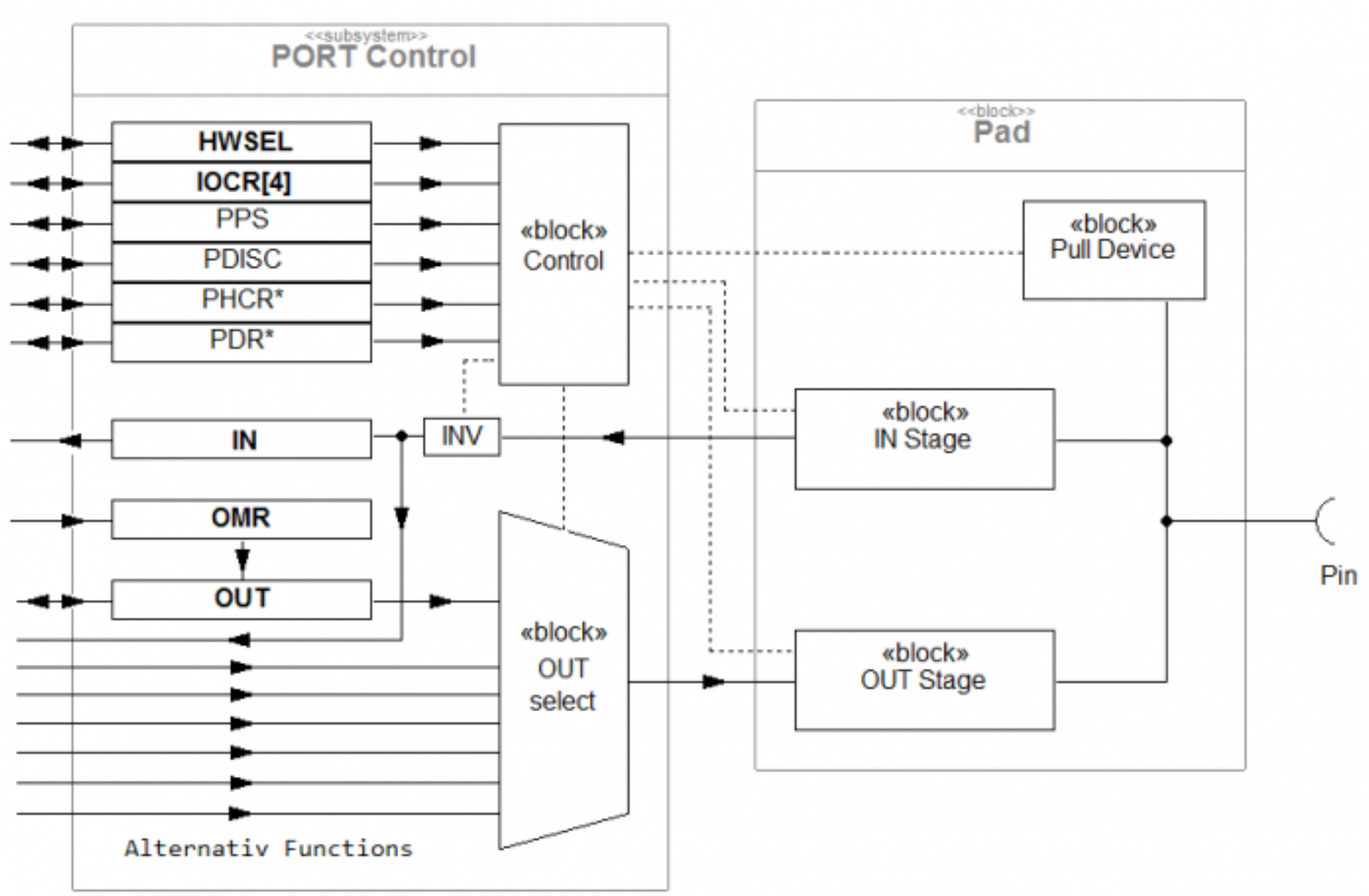


Der allgemeine Aufbau eines GPIO Ports wird beim XMC in zwei Bereiche unterteilt:

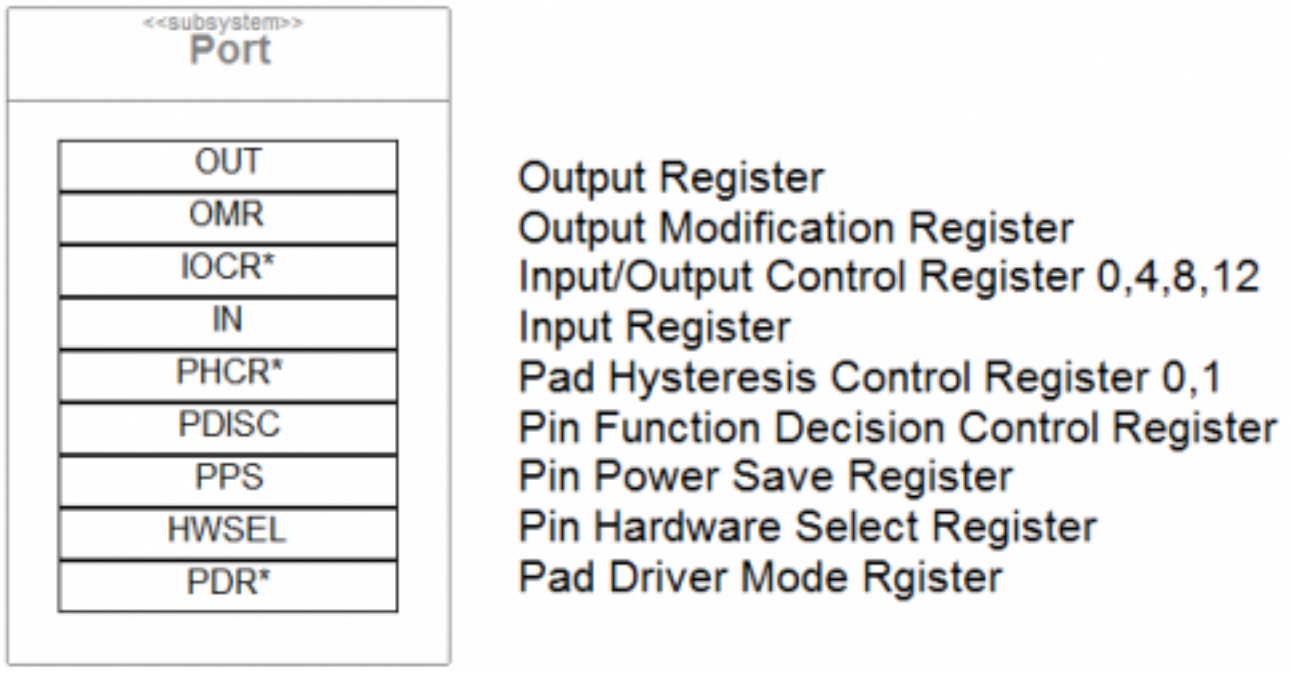
1. den *Port Slice* mit der Steuer-, Daten- und Auswahllogik
2. und dem *Pad* mit den Treiberstufen

Der Steuerblock (*Port Slice*) ließe sich nochmal unterteilen in:

- die eigentliche Steuerlogik mit den zugeordneten Steuer- und Kontroll-Registern
- den Eingabe- und Ausgabe-Bereich mit seinen IN/OUT-Registern
- und den Funktionsauswahlblock mit der möglichen Anbindung von Alternativfunktionen an das Port Pin



Auf die einzelnen Funktionen des Ports wird, wie gehabt, über IN-, OUT- sowie Steuer-Register zugegriffen.

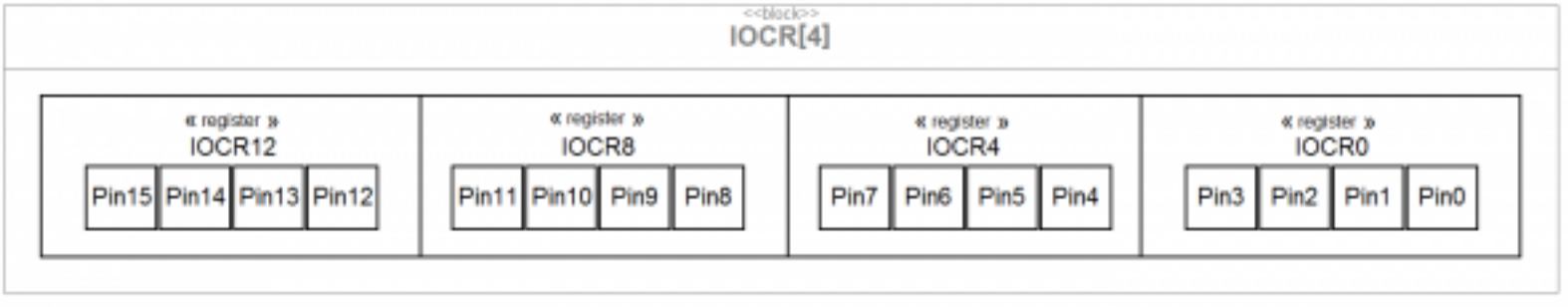


Nicht alle Register sind für die erste einfache Ausgabe interessant. Wichtig ist die Konfiguration der Datenrichtung des gewünschten Port-Pins und die eigentliche Ausgabe. Dazu werden vorerst folgende Register benötigt:

- *IOCRx*, Input/Output Control Register für die Datenrichtung
- *OUT*, Output Register für die direkte Ausagbe
- evtl. das *OMR*, Output Modifikation Register für gezielte Änderungen des Output Registers

Für digitale Ausgaben kann später in konkreten Projekten noch das Pad Driver Mode Register (PDR) interessant werden. Es bestimmt die Schaltgeschwindigkeit der Treiberstufen in Abhängigkeit der Pad-Klassifikation (A1=Normal, A1+=Schnell, A2= sehr Schnell). Für unsere LED ist das jedoch absolut unerheblich, da wir uns bei dieser Anwendung mit Sicherheit nicht über Nanosekunden Gedanken machen müssen.

Kommen wir zuerst zur Steuerung der Datenrichtung des Pins über das Input Output Control Register IOCR. Korrekterweise muss man hier von einem Registerblock sprechen, da es sich nicht um ein, sondern um vier Register handelt. Da die einzelnen Möglichkeiten der Konfiguration eines Pins nicht mit einem oder zwei Bit codiert werden können, sondern dafür beim XMC fünf Bit nötig sind, wird jedem Pin ein Byte für die Konfiguration zugeordnet. Die überzähligen 3 Bit werden für zukünftige Entwicklungen reserviert. Daraus ergibt sich folgende Struktur des IOCR-Registerblocks:



Die Namen der jeweiligen IOCR-Register orientieren sich an dem Basis-Pin, welches als Erstes abgebildet wird. Die folgenden Schlüsselworte repräsentieren die Bit-Definitionen aus den Header-Dateien des ausgewählten XMC-Controllers, für die möglichen Pin-Konfigurationen:

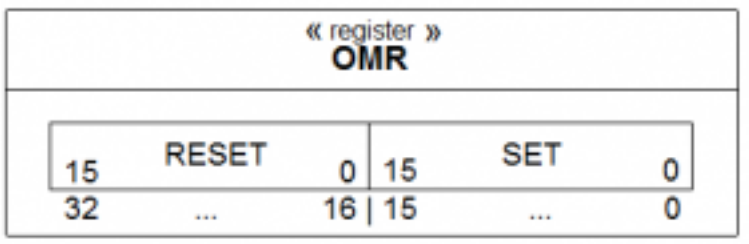
- XMC_GPIO_MODE_OUTPUT_PUSH_PULL, Push-pull general-purpose output
- XMC_GPIO_MODE_OUTPUT_OPEN_DRAIN, Open-drain general-purpose output
- XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT1 - ALT4, Push-pull alternate output function 1-4
- XMC_GPIO_MODE_OUTPUT_OPEN_DRAIN_ALT1 - ALT4, Open drain alternate output function 1-4
- XMC_GPIO_MODE_INPUT_TRISTATE, No internal pull device active
- XMC_GPIO_MODE_INPUT_PULL_DOW, Internal pull-down device active
- XMC_GPIO_MODE_INPUT_PULL_UP, Internal pull-up device active
- XMC_GPIO_MODE_INPUT_SAMPLING, No internal pull device active;Pn_OUTx continuously samples the input value
- XMC_GPIO_MODE_INPUT_INVERTED_TRISTATE, Inverted no internal pull device active
- XMC_GPIO_MODE_INPUT_INVERTED_PULL_DOWN, Inverted internal pull-down device active
- XMC_GPIO_MODE_INPUT_INVERTED_PULL_UP, Inverted internal pull-up device active
- XMC_GPIO_MODE_INPUT_INVERTED_SAMPLING, Inverted no internal pull device active; Pn_OUTx continuously samples the input value

Für das Einschalten der LED benötigen wir:

- **XMC_GPIO_MODE_OUTPUT_PUSH_PULL**

Das Ausgaberegister *OUT* ist für die unmittelbare Ausgabe verantwortlich. Jedes Bit repräsentiert den geforderten Zustand am Pin (0 = low, 1 = high). Zusätzlich kann die Ausgabe auch indirekt über das Register OMR (Output Modification Register) erfolgen. Dieses Register kann wie folgt auf das OUT Register wirken:

- ein oder mehrere Pins setzen
- ein oder mehrere Pins zurücksetzen
- ein oder mehrere Pins umschalten



Dafür bildet das 32 Bit Output Modification Register OMR jedes Port-Pin zwei mal ab. Eine logische 1 in den unteren 16 Bit signalisiert, dass dieses im OUT-Register auf 1 gesetzt werden soll. Eine logische 1 in den oberen 16 Bit signalisiert, dass dieses im OUT-Register auf 0 zurück gesetzt werden soll. Sind im unteren und im oberen Teil des OMR beide korrespondierenden Bits gesetzt, signalisiert diese Codierung, dass das betreffende Bit im OUT-Register umgeschaltet werden soll.

Entwurf

Gewöhnen wir uns gleich daran einigermaßen systematisch vorzugehen. Bevor wir die Befehle in unseren Code wild hineinhacken, schreiben wir erst die Kommentare, was wir an dieser oder jener Stelle im Code tun wollen.

```
//-------------------------------------------------------------------
// Titel     : Beispiel Hallo Welt mit SiSy XMC
//-------------------------------------------------------------------
// Funktion  : schaltet eine LED an
// Schaltung : LED an Port 1 Bit 0
//-------------------------------------------------------------------
// Hardware  : XMC4500
// Takt      : 120 MHz
// Sprache   : ARM C++
// Datum     : heute
// Version   : 1
// Autor     : ich
//-------------------------------------------------------------------
#include <stddef.h>
#include <stdlib.h>
#include "hardware.h"
#include "xmc_common.h"
#include "xmc_gpio.h"

void initApplication()
{
        SysTick_Config(SystemCoreClock/100);

        // Pin1.0 auf Ausgang konfigurieren
        // Pin1.0 anschalten

}

int main(void)
{
        SystemInit();
        initApplication();
        do{

                // bleibt noch leer

        } while (true);
        return 0;
}

extern "C" void SysTickFunction(void)
{
        // Application SysTick
```

```
            // bleibt auch erst mal leer
}
```

Jetzt nehmen wir die Finger von der Tastatur, atmen tief durch und schauen noch mal in Ruhe über unseren Entwurf. Dann kann es los gehen.

Realisierung

Die Konfiguration eines Pins erfolgt dadurch, dass die korrekte Bitkombination (*laut Referenzhandbuch 0b10000000 = XMC_GPIO_MODE_OUTPUT_PUSH_PULL*) auf die entsprechende Position des IOCR-Blocks geschrieben wird. Für das Pin 1.0 ergibt sich die Position IOCR0 Offset 0. Eine einfache Möglichkeit der Konfiguration ist die direkte Zuweisung der erforderlichen Werte an das Register.

```
//IOCR[3,2,1,0] |Pin 1.3|Pin 1.2|Pin 1.1|Pin 1.0|
PORT1->IOCR0 = 0b00000000000000000000000010000000;
```

Doch diese Schreibweise ist nicht wirklich selbsterklärend und wartungsfreundlich. Besser ist es, für diese Bitkodierung die dafür vorgesehenen Definitionen zu verwenden. Des weiteren werden die Programme recht schnell größer und die Anweisungen für die Konfiguration konkreter Pins lassen sich nicht in einer Zeile realisieren. Damit ist eine plumpe Wertzuweisung nicht wirklich sexy. Statt das Bit mit dem Vorschlaghammer in das Register zu prügeln und damit alle anderen Bits auch gleich platt zu machen, benutzen wir die elegantere OR-Verknüpfung, um nur die wirklich betroffenen Bits zu verändern. Wenn wir sehr diszipliniert sind, erfolgt die Konfiguration von Pin1.0 als Ausgang mit der Anweisung:

```
PORT1->IOCR0 |= (XMC_GPIO_MODE_OUTPUT_PUSH_PULL<<0*8);
```

Die korrekte Position im Register hat den Offset 0. Jede andere Position innerhalb des IOCR0 müsste mit einem Verschieben um jeweils 8 Bittstellen angesprochen werden. Das diese Schiebeoperation um *0*8* Bitpositionen hier angegeben wird ist zwar redundant, da gar 0-mal geschoben wird, aber für Dokumentations- und Lernzwecke durchaus sinnvoll.

Das Einschalten der LED erfolgt mit der Ausgabe über das OUT-Register. Die disziplinierte Version der Anweisung lautet wie folgt:

```
PORT1->OUT |= (XMC_GPIO_OUTPUT_LEVEL_HIGH<<0);
```

Es kann aber auch die Deklaration der Bitmaske für *BIT0* benutzt werden. Die folgende Anweisung sollte hinreichend gut lesbar sein:

```
PORT1->OUT |= BIT0;
```

Ergänzen Sie den Quellcode des Beispiel *HalloXMC* wie folgt:

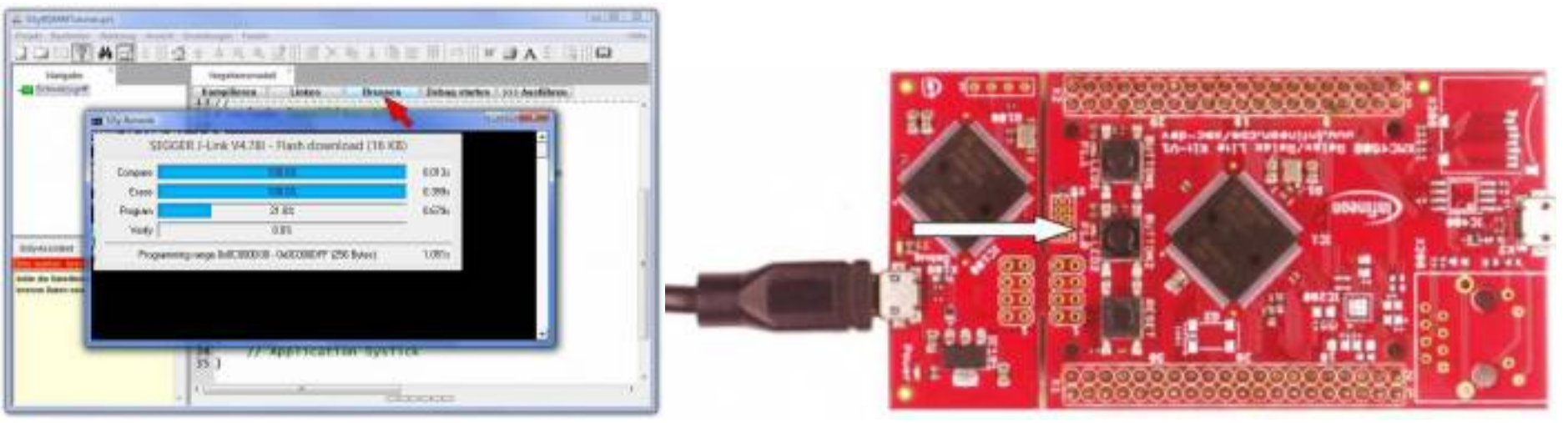
```
//-----------------------------------------------------------------
// Titel     : Hallo XMC
//-----------------------------------------------------------------
// Funktion  : LED leuchtet
// Schaltung : LED an Port 1, Bit 0
//-----------------------------------------------------------------
// Hardware  : XMC4500 Releax kit
// Takt      : 120 MHz
// Sprache   : ARM C++
// Autor     : Alexander Huwaldt
//-----------------------------------------------------------------
#include <stddef.h>
#include <stdlib.h>
#include "hardware.h"
#include "xmc_common.h"
#include "xmc_gpio.h"

void initApplication()
{
        // u.a. nötig für waitMs(..) und waitUs(..)
        SysTick_Config(SystemCoreClock/100);
        // weitere Initialisierungen durchführen

        // Port 1 Input Output Control Register, Bit 0 = OUT
        PORT1->IOCR0 |= (XMC_GPIO_MODE_OUTPUT_PUSH_PULL<<0*8);
        // Port 1 Output Register, Bit 0 = on
        PORT1->OUT   |= (XMC_GPIO_OUTPUT_LEVEL_HIGH<<0);
}

int main(void)
```

```
{
        SystemInit();
        initApplication();
        do{

                // bleibt erst mal leer

        } while (true);
        return 0;
}

extern "C" void SysTick_Handler(void)
{
        // Application SysTick
        // bleibt vorerst auch leer
}
```

Test

Übersetzen Sie das Programm. Korrigieren Sie ggf. Schreibfehler. Übertragen Sie das lauffähige Programm in den Programmspeicher des Controllers.

1. Kompilieren
2. Linken
3. Brennen



Gratulation! Sie haben Ihre erste Ausgabe realisiert. Die eine LED auf dem XMC4500 Relax Kit leuchtet jetzt.

Videozusammenfassung

Und hier diesen Abschnitt wiederum als Videozusammenfassung.

(Mit installiertem Flash kann man an dieser Stelle ein Video in dieser Web-Seite ansehen.)

besser auf youTube

Erweiterung zum Klassiker: Blinky

In den meisten Fällen gehen Beispiele für die erste einfache Ausgabe schon weiter und lassen gern mal eine oder mehrere LEDs blinken. Manchmal konfiguriert man dafür den SysTick so langsam, dass man dort das blinken codieren kann. Diesen Weg werden wir nicht gehen. Der SysTick bleibt als System Ereignis mit 10 Millisekunden unverändert. Der zweite Weg, der in Beispielen oft beschritten wird, ist der, eine kleine Wartefunktion zu bauen, die es ermöglicht, das Blinken in der *Mainloop* zu realisieren. In SiSy gibt es bereits vorgefertigte Warteroutinen. Wir verwenden die Funktion *WaitMs*.

Das Einschalten der LED erfolgte durch das Setzen des entsprechenden Bits im OUT-Register. Dafür benutzten wir oben die bitweise OR-Verknüpfung.

```
PORT1->OUT |= BIT0;
```

Jetzt müssen wir die LED aber auch ausschalten. Dafür maskieren wir das gewünschte Bit aus. Dazu wollen wir die Bitmaske invertieren (bitweise Negation) und die negierte Maske mit dem OUT-Register bitweise AND-verknüpfen.

```
PORT1->OUT &= ~BIT0;
```

Durch diese Anweisungsfolge wird das gewünschte Bit explizit auf 0 gesetzt. Bitte legen Sie ein neues kleines Programm an und ergänzen sie die Anwendung wie folgt:

```
//-----------------------------------------------------------------------
// Titel     : Blinky mit dem XMC4500
//-----------------------------------------------------------------------
// Funktion  : LED blinkt
// Schaltung : LED an Port 1, Bit 0
//-----------------------------------------------------------------------
// Hardware  : XMC4500 Releax kit
// Takt      : 120 MHz
// Sprache   : ARM C++
// Autor     : Alexander Huwladt
//-----------------------------------------------------------------------
#include <stddef.h>
#include <stdlib.h>
#include "hardware.h"
#include "xmc_common.h"
#include "xmc_gpio.h"

void initApplication()
{
        // u.a. nötig für waitMs(..) und waitUs(..)
        SysTick_Config(SystemCoreClock/100);
        // weitere Initialisierungen durchführen

        // Port 1 Input Output Control Register, Bit 0 = OUT
        PORT1->IOCR0 |= (XMC_GPIO_MODE_OUTPUT_PUSH_PULL<<0*8);
}

int main(void)
{
        SystemInit();
        initApplication();
        do{
                // Port 1 Output Register, Bit 0 = 1
                PORT1->OUT   |= BIT0;
                // 1. Halbzyklus
                waitMs(200);
                // Port 1 Output Register, Bit 0 = 0
                PORT1->OUT   &= ~BIT0;
                // 2. Halbzyklus
                waitMs(200);

        } while (true);
        return 0;
}

extern "C" void SysTick_Handler(void)
{
        // Application SysTick
}
```

Wem der C-Code für das Setzen und Löschen eines Bits zu kryptisch ist, kann sich auch der folgenden Makros bedienen:

```
// setze Bit (Register, Position)
SET_BIT(PORT1->OUT,0);
// lösche Bit (Register, Position)
CLR_BIT(PORT1->OUT,0);
```

Für das Blinky-Beispiel wird aber noch ein Register interessant. Blinken bedeutet zyklisches Umschalten. Das Umschalten eines Pins können wir wie oben beschrieben mit der entsprechenden Bitkombination im Output Modification Register bewirken. Dazu muss das Bit 0 im oberen und unteren Teil des OMR gleichzeitig gesetzt werden. Modifizieren Sie die Anwendung wie folgt:

```
//-----------------------------------------------------------------------
// Titel     : Blinky 2 mit dem XMC4500
//-----------------------------------------------------------------------
// Funktion  : LED blinkt
// Schaltung : LED an Port 1, Bit 0
//-----------------------------------------------------------------------
// Hardware  : XMC4500 Releax kit
// Takt      : 120 MHz
```

```
// Sprache   : ARM C++
// Autor     : Alexander Huwaldt
//-----------------------------------------------------------------------
#include <stddef.h>
#include <stdlib.h>
#include "hardware.h"
#include "xmc_common.h"
#include "xmc_gpio.h"

void initApplication()
{
        // u.a. nötig für waitMs(..) und waitUs(..)
        SysTick_Config(SystemCoreClock/100);
        // weitere Initialisierungen durchführen

        // Port 1 Input Output Control Register, Bit 0 = OUT
        PORT1->IOCR0 |= (XMC_GPIO_MODE_OUTPUT_PUSH_PULL<<0*8);
}

int main(void)
{
        SystemInit();
        initApplication();
        do{
                // Port 1 Output Modification Register,
                // Bit 0  High-Word + Low-Word = toggle
                PORT1->OMR |= BIT0<<16|BIT0;
                // Halbzyklus
                waitMs(200);
        } while (true);
        return 0;
}

extern "C" void SysTick_Handler(void)
{
        // Application SysTick
}
```

Die Infineons Low Level Treiber (XMC Lib) anwenden

Dieselbe Aufgabe lässt sich natürlich auch mit den von Infineon zur Verfügung gestellten Treiberbibliotheken recht elegant erledigen. Wobei es hier noch nicht wirklich viel Erleichterung bringt, außer dass man sich nicht mehr mit der internen Registerstruktur auseinandersetzen muss. Das wird bei komplexeren Aufgaben jedoch anders, dann entlastet es den Entwickler sehr wenn er gegen die Treiber API und nicht gegen die Register programmiert. Für die digitale Ausgabe sind zunächst folgende Treiberfunktionen interessant:

- XMC_GPIO_SetMode(PORT, PinNummer, MODE);
- XMC_GPIO_SetOutputHigh(PORT, PinNummer);
- XMC_GPIO_SetOutputLow(PORT, PinNummer);
- XMC_GPIO_SetOutputLevel(PORT, PinNummer, LEVEL);
- XMC_GPIO_ToggleOutput(PORT, PinNummer);

Charakteristisch für die neue XMC Lib ist der Präfix XMC für alle Bezeichner (Definitionen, Aufzählungen, Strukturen, Funktionen, …) gefolgt von der angesprochenen Hardware und der gewünschten Funktion: XMC_*Unit_Funktion*(Parameter);

```
//-----------------------------------------------------------------------
// Titel     : Blinky 2 mit dem XMC4500 (XMC Lib)
//-----------------------------------------------------------------------
// Funktion  : LED blinkt
// Schaltung : LED an Port 1, Bit 0
//-----------------------------------------------------------------------
// Hardware  : XMC4500 Releax kit
// Takt      : 120 MHz
// Sprache   : ARM C++
// Autor     : Alexander Huwaldt
//-----------------------------------------------------------------------
#include <stddef.h>
#include <stdlib.h>
#include "hardware.h"
#include "xmc_common.h"
#include "xmc_gpio.h"
```

```
void initApplication()
{
    // u.a. nötig für waitMs(..) und waitUs(..)
    SysTick_Config(SystemCoreClock/100);
    // weitere Initialisierungen durchführen

    // Port 1 Input Output Control Register, Bit 0 = OUT
    // PORT1->IOCR0 |= (XMC_GPIO_MODE_OUTPUT_PUSH_PULL<<0*8);
    XMC_GPIO_SetMode(XMC_GPIO_PORT1, 0, XMC_GPIO_MODE_OUTPUT_PUSH_PULL );

    // Port 1 Output Register, Bit 0 = on
    // PORT1->OUT   |= (XMC_GPIO_OUTPUT_LEVEL_HIGH<<0);
    XMC_GPIO_SetOutputHigh(XMC_GPIO_PORT1, 0);
}

int main(void)
{
    SystemInit();
    initApplication();
    do{
        // PORT1->OMR |= BIT0<<16|BIT0;
        XMC_GPIO_ToggleOutput(XMC_GPIO_PORT1,0);
        waitMs(300);

    } while (true);
    return 0;
}

extern "C" void SysTick_Handler(void)
{
    // Application SysTick
    // bleibt vorerst auch leer
}
```

Nächstes Thema

- Einfache Ein- und Ausgaben mit dem XMC