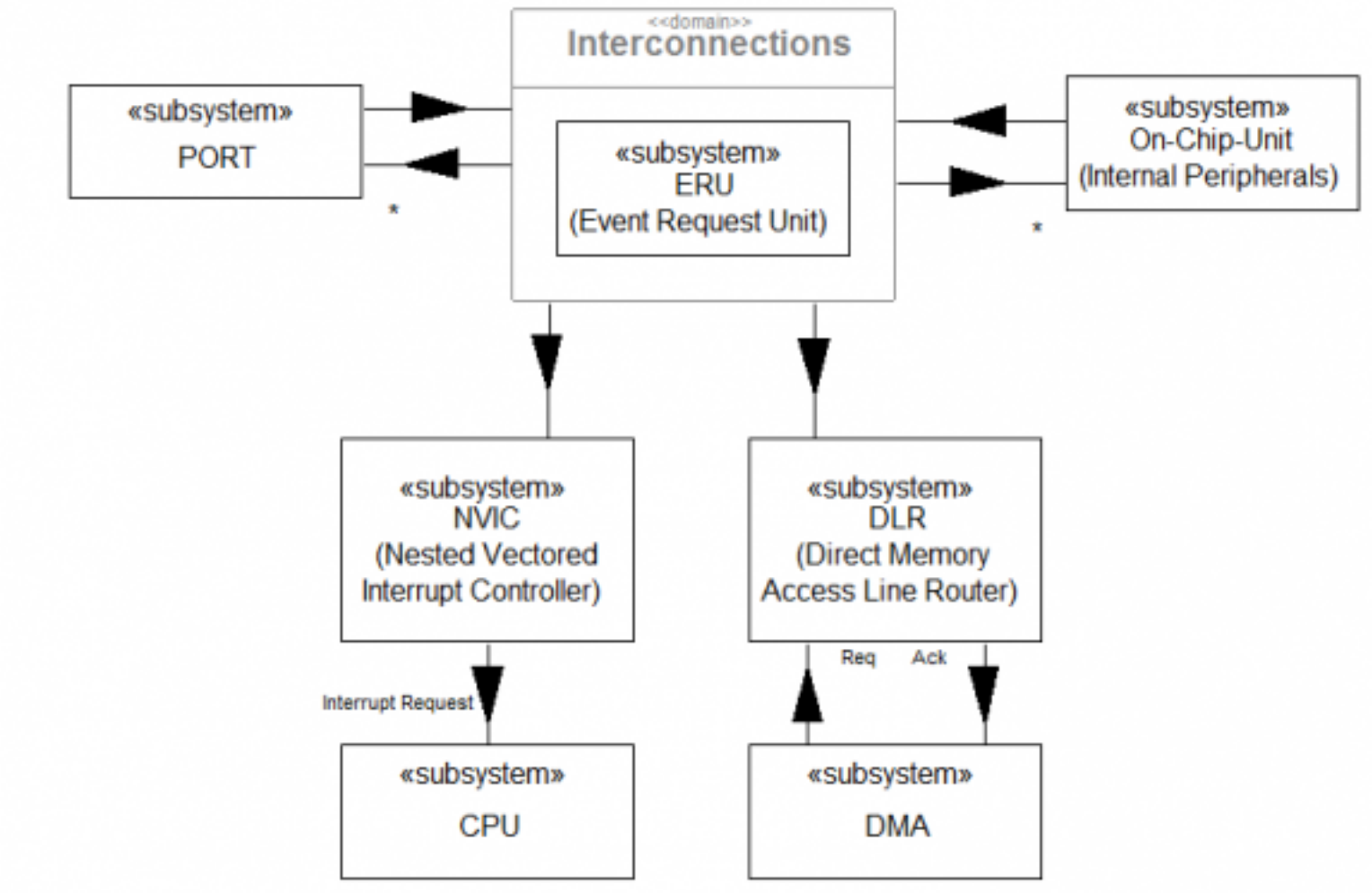
XMC Tutorial für C/C++ und UML

XMC Interrupts in C

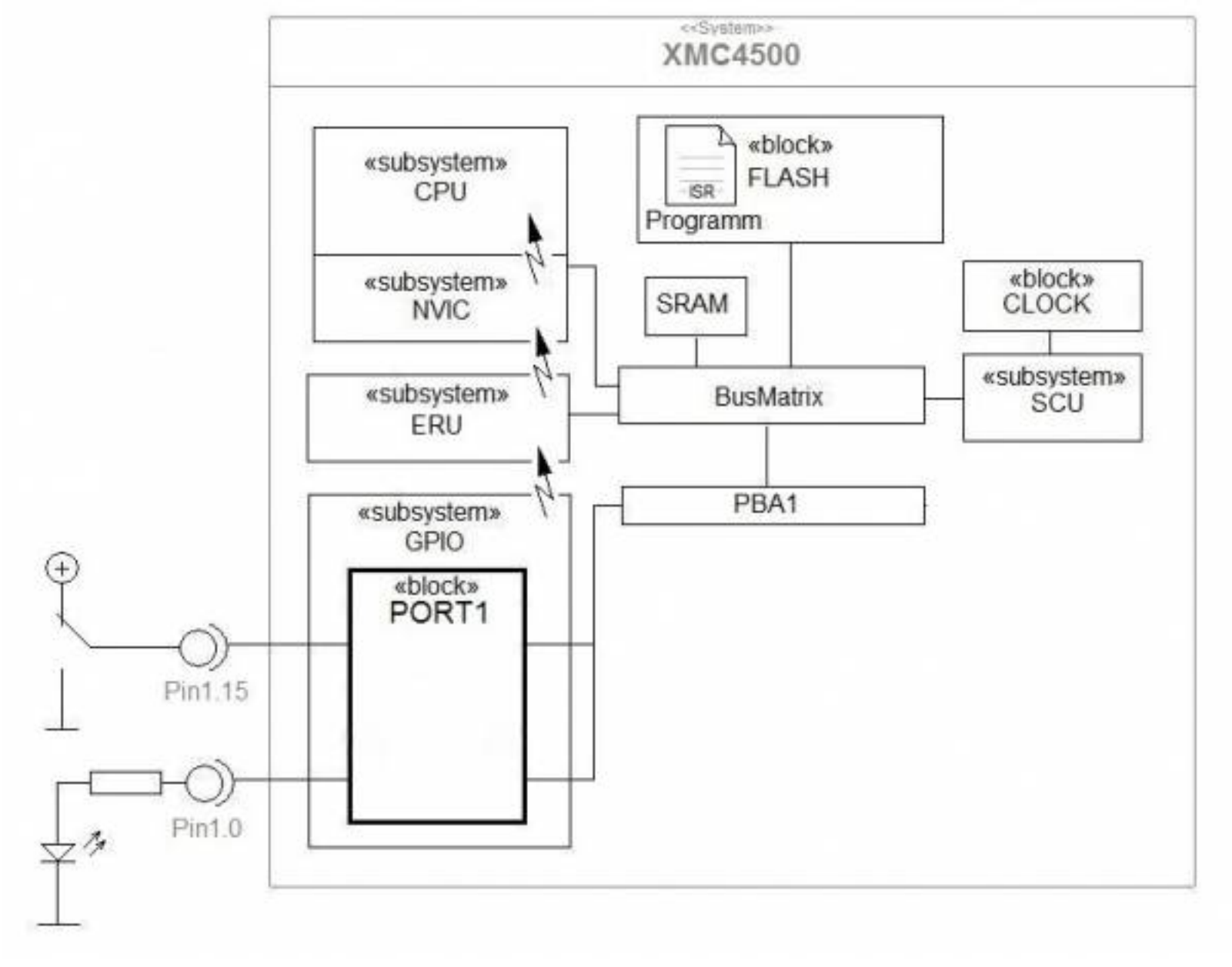
Die Controller der XMC-Familie verfügen über sehr leistungsfähige Bausteine für das Management von mehr als 100 verschiedenen Systemereignissen. Theoretisch kann ein ARM über seinen Standard Interruptcontroller NVIC bis 240 Ereignisse verwalten. Mit dieser Menge an möglichen Interrupts und seinem praktisch im Kern eingebetteten schnellen Interruptcontroller, kann ein Cortex M vor lauter Unterbrechungskraft sowieso schon kaum laufen. Die Auswahl und Konfiguration konkreter Interrupts kann für den XMC-Entwickler aber recht kniffelig werden. Neben dem NVIC (Nested Vectored Interrupt Controller) verfügt der XMC noch über eine komplexe Event Request Unit (ERU). Diese verknüpft Systembausteine mit dem NVIC und dem DMA-Line-Router, aber auch die Systembausteine untereinander.



Wir beginnen mit der Programmierung eines einfachen externen Interrupts vom Taster.

Die Aufgabe

Um die Interrupt-Programmierung des XMC kennen zu lernen, nehmen wir uns eine verhältnismäßig einfache Aufgabe vor. Lassen wir eine LED über einen Digital-Port ein-, aus- und umschalten.



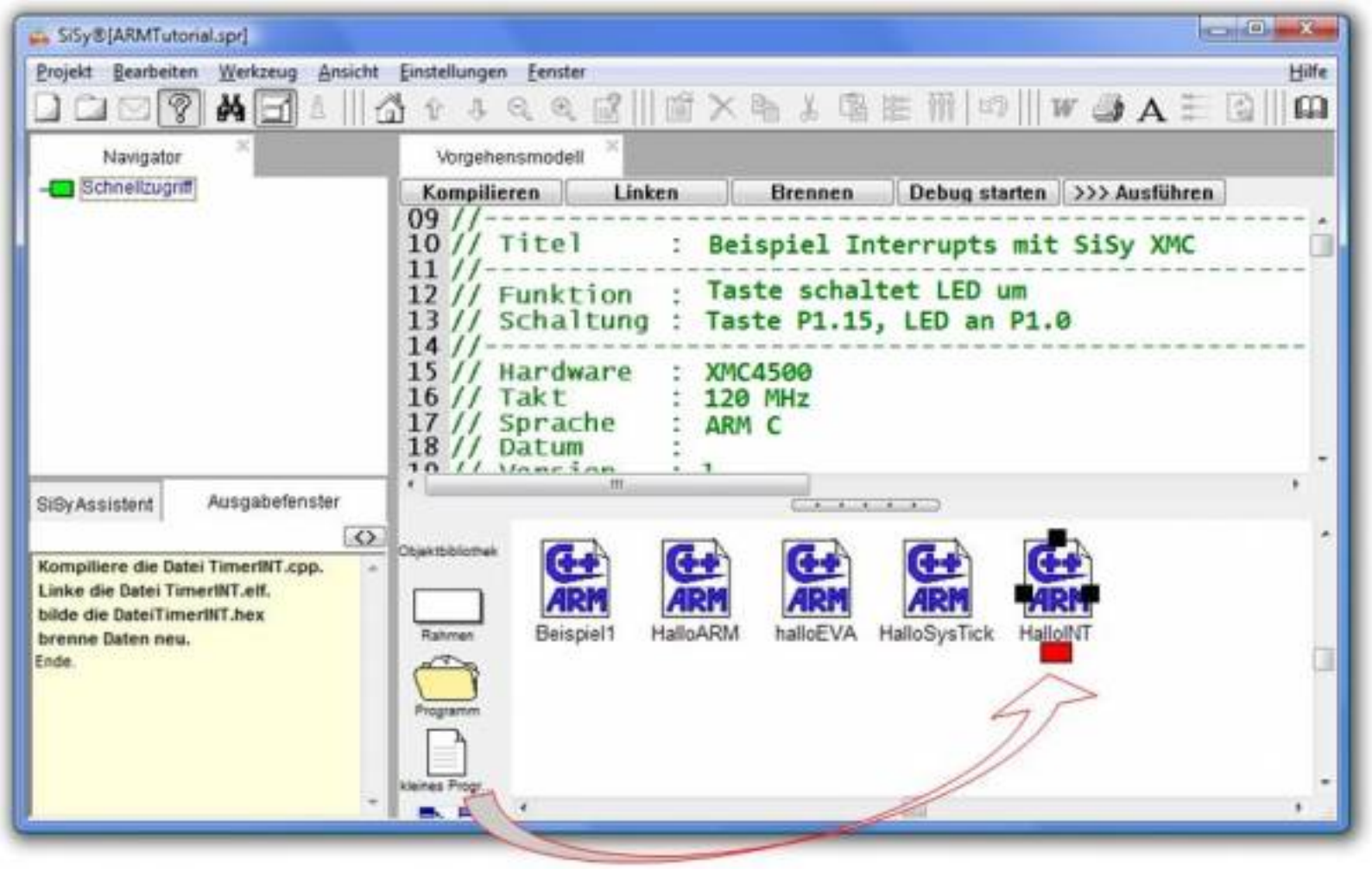
Der Blick auf das Blockbild verrät uns, dass zwei weitere für uns neue Bausteine programmiert werden müssen, die Event Request Unit ERU und der Interruptcontroller NVIC. Diesen müssen wir beibringen, dass wir das GPIO-Pin-Ereignis *„Taste wurde gedrückt“* als Unterbrechungsanforderung behandeln möchten. Fassen wir die Aufgabenstellung kurz zusammen:

1. Pin1.15 als Eingang konfigurieren
2. Pin1.0 als Ausgang konfigurieren
3. ERU konfigurieren
4. NVIC initialisieren
5. eine ISR schreiben

Detaillierte Informationen zur Programmierung der ERU und des NVIC finden sie im bereits erwähnten XMC4500-Referenzhandbuch.

Vorbereitung

Falls das Tutorial-Projekt nicht offen ist, öffnen Sie dies. Legen Sie bitte ein neues kleines Programm an und laden das Grundgerüst für eine XMC4500 Anwendung. Beachten Sie die Einstellungen für die Zielplattform XMC4500 Releax Kit.



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

```
//------------------------------------------------------------------
// Titel     : Beispiel externer Interrupt in SiSy XMC
//------------------------------------------------------------------
```

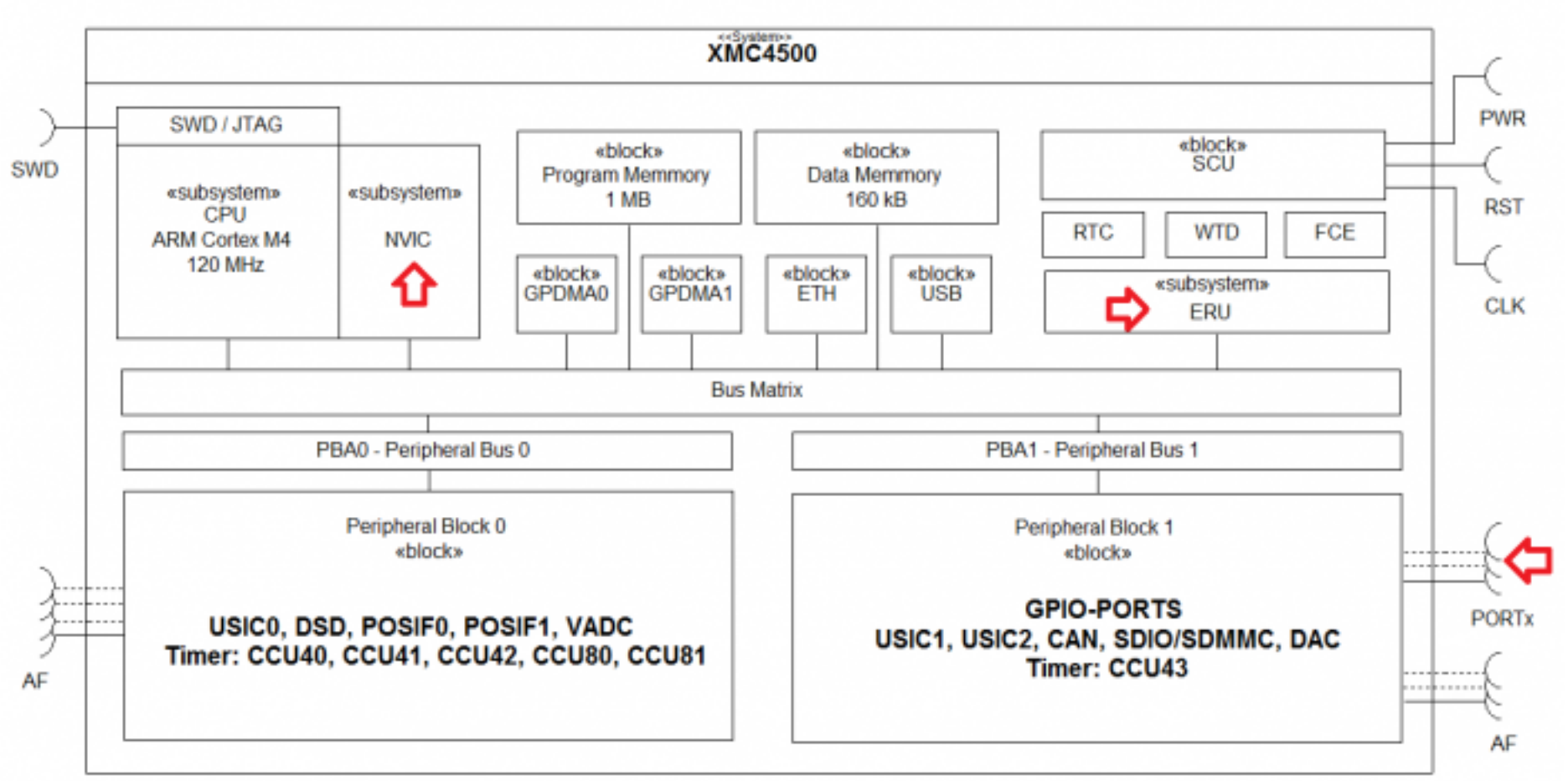
```
// Funktion  : Taster schaltet LED
// Schaltung : Taster an P1.14/P1.15 LED an P1.0/P1.1
//----------------------------------------------------------------------
// Hardware  : XMC4500
// Takt      : 120 MHz
// Sprache   : ARM C++
// Datum     : heute
// Version   : 1
// Autor     : ich
//----------------------------------------------------------------------
```

Grundlagen

Vergegenwärtigen wir uns in der Übersicht noch mal welche Bausteine jetzt eine entscheidende Rolle spielen.

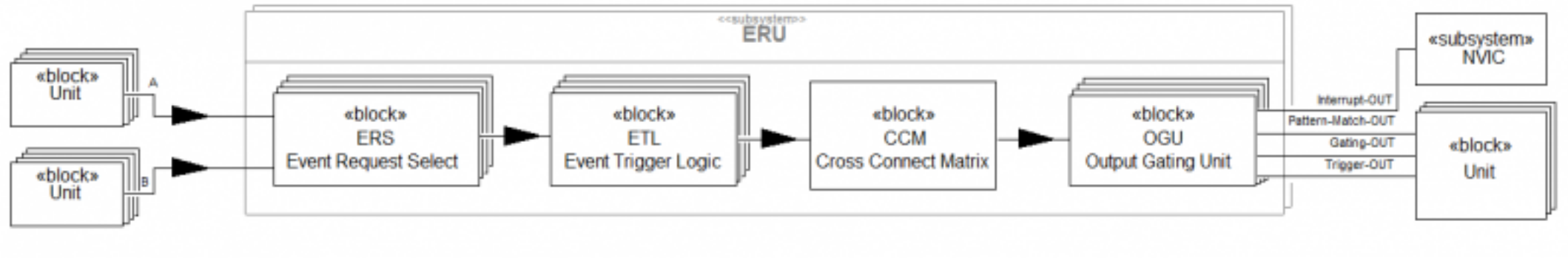
- NVIC, Nested Vectored Interrupt Controller
- ERU, Event Request Unit
- GPIO-Ports, General Purpose I/O Ports

Alles was wir in diesem Abschnitt programmieren verwendet sofort die XMC Lib. Auf Registerebene wird es jetzt langsam viel zu aufwändig 😕



Die Event Request Unit (ERU) ist verantwortlich ein bestimmtes Ereignismuster (Pattern) zu erkennen und an einen Empfänger zu melden. Die ERU gliedert sich intern in vier Blöcke.

- ERU-Eingangslogik
  - ERS, Event Request Select, Empfang der Ereignisse von den Ereignisquellen
  - ETL, Event Trigger Logic, logische Verknüpfung der Ereignisquellen
- ERU-Ausgangslogik
  - CCM, Cross Connect Matrix, Verknüpfung der Ereignisquellen mit dem Ausgabekanal OGU
  - OGU, Output Gating Unit, Verteilung/Senden der Ereignisse an Empfänger



Dabei stehen für vier ERS Eingangsseitig je zwei Kanäle mit wiederum vier möglichen Ereignisquellen zur Verfügung. Diese werden logisch verknüpft (Polarität,AND,OR) und an die ETL weitergeleitet. Die ETL generiert die gewünschten Triggerereignisse (steigende oder fallende Flanke, etc.) um diese über die CCM an die OGU weiter zu leiten. Die OGU erzeugt vier verschiedene, kaskadierte Ausgaben

1. PDOUT, Pattern Detect OUT, Ereignismuster erkannt
2. GOUT, Gating OUT, PDOUT + Select Gating Scheme, Ereignismuster + Übertragungsschema erkannt
3. TOUT, Trigger OUT, GOUT + Trigger Logik, Ereignismuster + Übertragungsschema + Triggerbedingung erkannt
4. IOUT, Interrupt OUT, TOUT + Interrupt Logik, Ereignismuster + Übertragungsschema + Triggerbedingung + Interruptbedingung erkannt

Für uns ist letztlich das Interrupt OUT Ereignis relevant. Dieses wird an den NVIC als Serviceanforderung weitergeleitet.

Port konfigurieren

Die Konfiguration der Ports haben wir in den vorangegangenen Abschnitten bereits besprochen. Die Ports selbst „merken“ nichts davon, dass sie als Interruptquellen dienen. Zu beachten ist jedoch, dass nicht jeder Pin einen „Draht“ zur ERU hat (siehe Referenzunterlagen zum Controller).

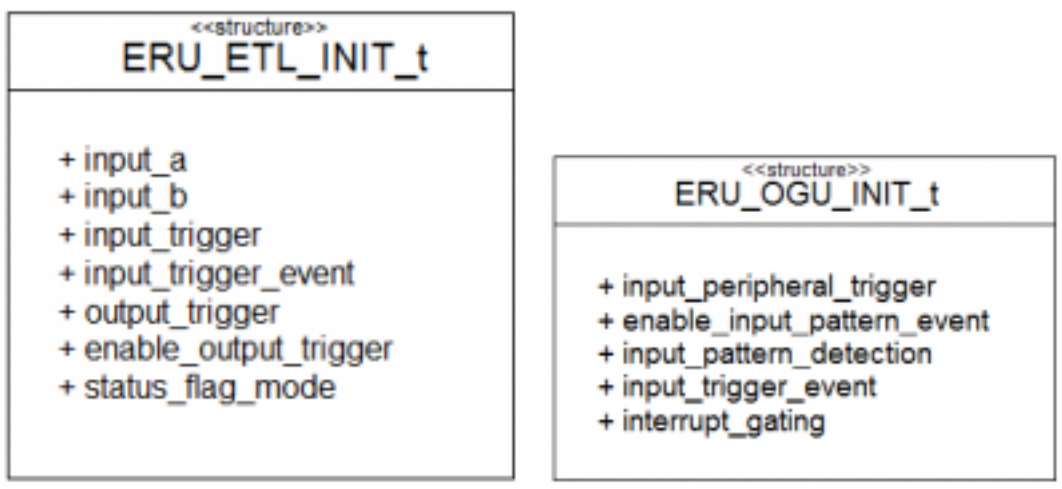
```
// Konfiguriere Pin1.0 als Ausgang
XMC_GPIO_SetMode(XMC_GPIO_PORT1, 0, XMC_GPIO_MODE_OUTPUT_PUSH_PULL );

// Konfiguriere Pin1.15 als Eingang ohne PullUp
XMC_GPIO_SetMode(XMC_GPIO_PORT1, 13, XMC_GPIO_MODE_INPUT_TRISTATE);
```

Bevor wir uns an die Konfiguration des Interrupt machen sollten wir einen möglichen RESET-Zustand dieser Ereignisquelle löschen.

```
// ggf. Ereignissequelle aus dem Zustand RESET holen (nur für XMC4xxx nötig)
XMC_SCU_INTERRUPT_ClearEventStatus(XMC_SCU_PERIPHERAL_RESET_ERU1);
```

ERU konfigurieren

Für die Initialisierung der ERU gehen wir nicht direkt auf die Register sondern nutzen Initialisierungsstrukturen und dazugehörige Initialisierungsfunktionen. Die Initialisierungsstruktur für die Eingangsseite (ETL) und Ausgangsseite (OGU) der ERU sind wie folgt aufgebaut:

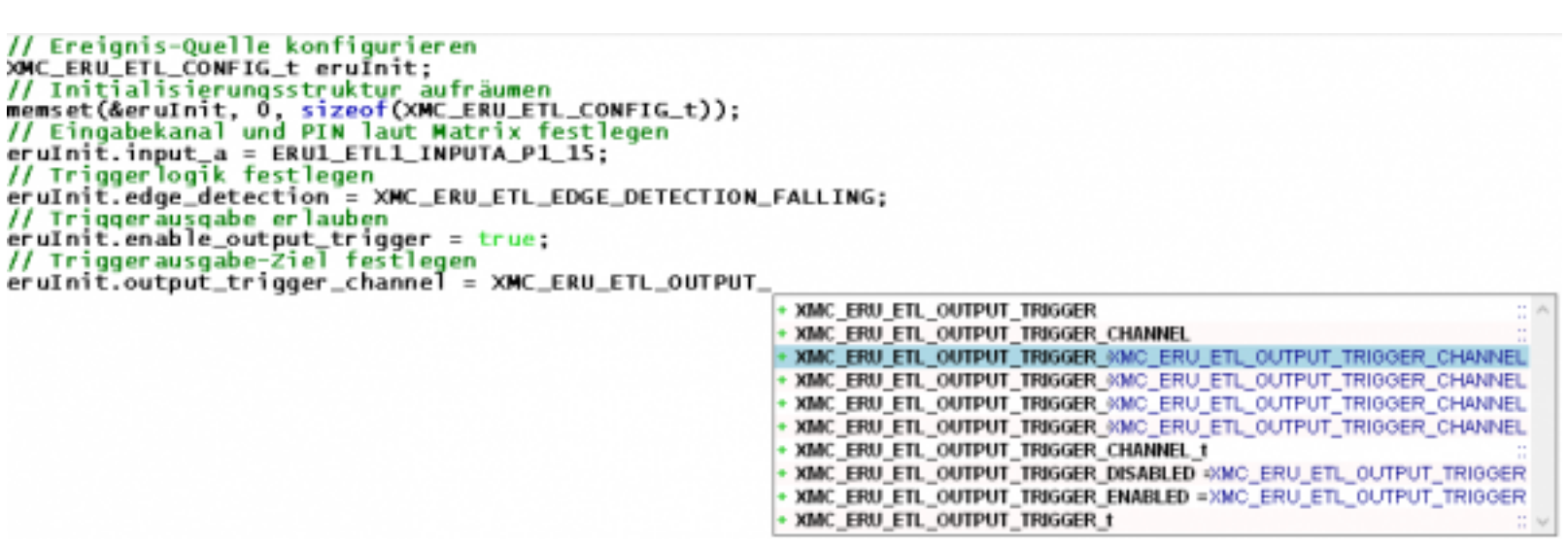


Von dieser Struktur müssen Instanzen angelegt werden. Die erforderlichen Strukturelemente müssen mit korrespondierenden Werten gefüllt werden. Dabei hilft uns die Codevervollständigung in SiSy, die myXMC Referenzkarte und im Zweifelsfall das Referenzhandbuch. Auf der myXMC Referenzkarte findet sich eine Übersicht zur ETL Konfiguration.

GPIO Ports - Externe Interrupts

		ERU0				ERU1			
		ETL0	ETL1	ETL2	ETL3	ETL0	ETL1	ETL2	ETL3
A	0	P0.1	P0.10	P1.5	P1.1	P1.5	P1.15	P1.3	P0.5
	1	P3.2		P0.8	P3.6				
	2	P2.5	P2.3	P0.13	P0.11				
	3								
B	0	P0.0	P0.9	P1.4	P1.0	P2.1	P2.7	P1.2	P0.3
	1	P3.1	RTC_XTAL1	P0.7	P3.5				
	2	P2.4	P2.2	P0.12	P.06				
	3	P2.0	P2.6	P0.4	P0.2				

Das gewünschte Pin1.15 finden wir in der Referenz an ERU1, Input Chanel A0 an ETL1. Diese Information benötigen wir um jetzt die korrekten Werte in die Strukturen einzutragen. Wir werden uns recht bald an die Namenskonventionen der Definitionen gewöhnen. Die Codevervollständigung in SiSy hilft uns dann ungemein den richtigen Parameter herauszufinden.



Die frisch angelegten Instanzen der Strukturen füllen wir sicherheitshalber mit Nullen auf, da in C/C++ angeforderter Speicher nicht von Hause aus aufgeräumt wird und Datenmüll enthalten kann.

```
// Ereignis-Quelle konfigurieren
XMC_ERU_ETL_CONFIG_t eruInit;
// Initialisierungsstruktur aufräumen
memset(&eruInit, 0, sizeof(XMC_ERU_ETL_CONFIG_t));
// Eingabekanal und PIN laut Matrix festlegen
```

```
eruInit.input_a = ERU1_ETL1_INPUTA_P1_15;
// Triggerlogik festlegen
eruInit.edge_detection = XMC_ERU_ETL_EDGE_DETECTION_FALLING;
// Triggerausgabe erlauben
eruInit.enable_output_trigger = true;
// Triggerausgabe-Ziel festlegen
eruInit.output_trigger_channel = XMC_ERU_ETL_OUTPUT_TRIGGER_CHANNEL0;
```

Für die OGU müssen wir festlegen, dass keine Interrupts gesperrt werden sollen. Der Initialzustand in der Struktur wäre sonst *ERU_OGU_INTERRUPT_GATING_ALWAYS*, womit alle Interruptgenerierung (service request) unterbunden wäre.

```
// Ereignis Ziel konfigurieren
XMC_ERU_OGU_CONFIG_t oguInit;
// Initialisierungsstruktur aufräumen
memset(&oguInit, 0, sizeof(XMC_ERU_OGU_CONFIG_t));
// Interrupt erlauben
oguInit.service_request = true;
```

Die so gefüllten Strukturen können wir jetzt an die Initialisierungsfunktionen übergeben. Dabei ist zu beachten, dass die richtige ERU-ETL-OGU-Konfiguration eingehalten wird.

```
// ERU Eingangsseite Konfigurieren
XMC_ERU_ETL_Init(ERU1_ETL1, &eruInit);
// ERU Ausgangsseite Konfigurieren
XMC_ERU_OGU_Init(ERU1_OGU0, &oguInit);
```

NVIC initialisieren

Der NVIC (Nested Vectored Interrupt Controller) zeichnet sich durch seine enge Kopplung an den Kern des Prozessors aus. Ergebnis sind extrem kurze Latenzzeiten der Interrupts. Ein weiterer Schritt zu einem erwachsenen Interruptcontroller ist die echte Vektorisierung im Vergleich z.B. zu älterer ARM oder dem AVR. Das bedeutet, dass in der Vektortabelle keine Sprungbefehle mehr stehen, sondern Adressen, welche vom Kern quasi direkt in den Programmcounter übernommen werden können. Die Arbeit mit Adressen erleichtert auch die Priorisierung und Neuordnung der Vektoren zur Laufzeit. Beim NVIC ist die Priorität einer Interruptanforderung nicht über die Position in der Tabelle definiert. Interrupts können in Kanälen gruppiert werden. Diese Gruppen erhalten eine Gruppen-Priorität. Innerhalb einer solchen Gruppe können ebenfalls Sub-Prioritäten vergeben werden. Die Priorität wird beim Cortex mit 4 Bit codiert und ist somit von 0-15 abstufbar. Der höchste Wert besitzt dabei die kleinste Priorität. Die Ausnutzung weiterer Fähigkeiten des NVIC sind dann wohl eher Echtbetriebssystemen vorbehalten. Die Möglichkeit zur dynamischen Neupriorisierung und Verkettung von Interrupts sowie die automatische Speicherung und Wiederherstellung von Prozessorzuständen schaffen Voraussetzungen für ein effektives Multitaskingbetriebssystem.

```
// NVIC konfigurieren
// Interruptpriorität niedrig
// Interruptquelle ERU1, OGU0
NVIC_SetPriority(ERU1_0_IRQn, 15);
// Interrupt erlauben für ERU1, OGU0
NVIC_EnableIRQ(ERU1_0_IRQn);
```

Die gewünschte OGU generiert nach den Vorgaben der ERU-Initialisierung eine Interruptanfrage (Interrupt Request IRQ) an den Interruptcontroller NVIC. Dieser schaut ob der gewünschte Innterupt erlaubt ist und falls gleichzeitig andere IRQ vorliegen checkt er die Prioritäten. Letztlich sorgt der NVIC dafür, dass die zum IRQ gehörende Interrupt Service Routine ISR aufgerufen wird. In unserem Falle schaltet dann die LED endlich um 😉

ISR schreiben

Ganz so schnell schießen die Preußen nun aber auch wieder nicht. Es ist natürlich nötig erst noch eine Funktion zur Ereignisbehandlung zu schreiben. Derartige Funktionen sind als *extern „C“ void* zu deklarieren und besitzen fest vorgegebene Namen, die es dem Compiler ermöglichen, die ISR dem richtigen Interrupt zuzuordnen. Die Namen der Eventhandler folgen immer dem Muster *Gerät**_IRQHandler***. Wenn wir in SiSy die Zeichenfolge *ERU1_0…* eingeben können wir in der angebotenen Liste den Namen der ISR auswählen.

```
// ISR für ERU1 OGU0 IRQ
extern "C" void ERU1_0_IRQHandler(void)
{
    // LED toggle
    XMC_GPIO_ToggleOutput(XMC_GPIO_PORT1,0);
}
```

Entwurf

Jetzt sollte alles beieinander sein und wir tasten uns an die Lösung der Aufgabe heran. Schreiben Sie zuerst die Kommentare *WAS* in welcher Reihenfolge, also *WANN* zu tun ist. Beachten Sie in welchen Programmbereichen, also *WO* die einzelnen Aktionen zukünftig ausgeführt werden sollen.

```
//-------------------------------------------------------------------
// Titel     : Beispiel externer Interrupt in SiSy XMC
//-------------------------------------------------------------------
// Funktion  : Taster schaltet LED
// Schaltung : Taster an P1.14/P1.15 LED an P1.0/P1.1
```

```
//-----------------------------------------------------------------------
// Hardware  : XMC4500
// Takt      : 120 MHz
// Sprache   : ARM C++
// Datum     : heute
// Version   : 1
// Autor     : ich
//-----------------------------------------------------------------------

#include <stddef.h>
#include <stdlib.h>
#include "hardware.h"

void initApplication()
{

        SysTick_Config(SystemCoreClock/100);
        // weitere Initialisierungen durchführen

        // Konfiguriere LED
        // Konfiguriere Taster als INT-Quelle
        // Interruptcontroller konfigurieren
}

int main(void)
{
        SystemInit();
        initApplication();
        while(true)
        {
                //leer
        }
}

extern "C" void SysTickFunction(void)
{
        //leer
}

// hier ISR schreiben
```

Realisierung

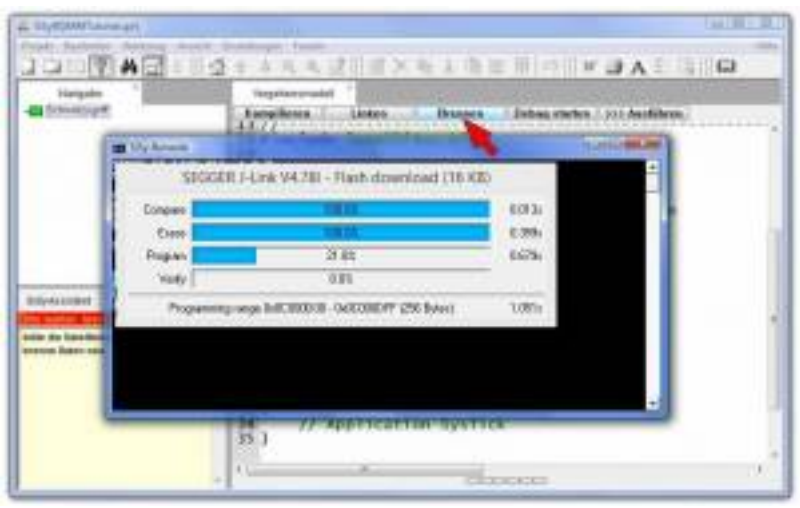
Sie wissen ja, tief durchatmen, noch mal drüber schauen und dann selbst und bewusst die Befehlszeilen eingeben.

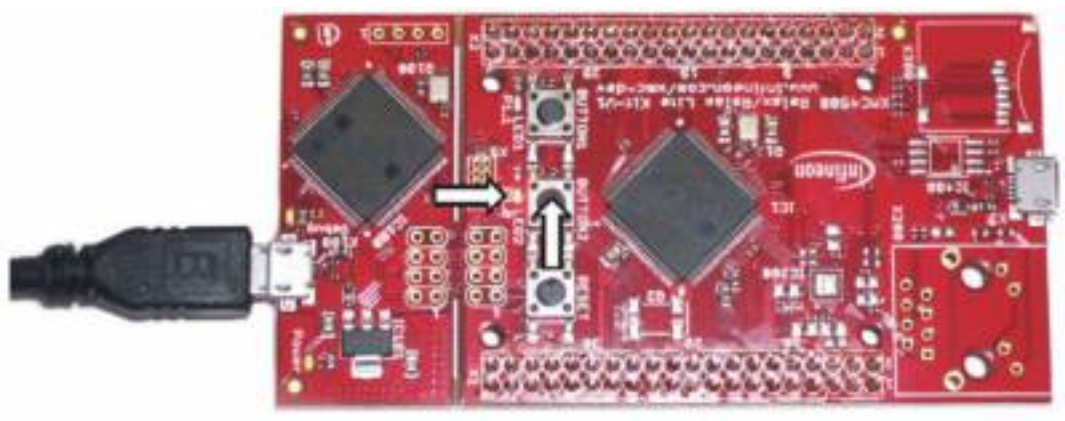
```
//-----------------------------------------------------------------------
// Titel     : Beispiel einfache XMC-Interruptprogrammierung
//-----------------------------------------------------------------------
// Funktion  : Taste schaltet LED um
// Schaltung : Taste an Pin1.15, LED an Pin1.0
//-----------------------------------------------------------------------
// Hardware  : XMC4500
// Takt      : 120 MHz
// Sprache   : ARM C++
// Datum     : heute
// Version   : 1
// Autor     : ich
//-----------------------------------------------------------------------
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "hardware.h"
#include "xmc_gpio.h"
#include "xmc_scu.h"
#include "xmc_eru.h"


void initApplication()
{
    // config auf 10ms SystemTimer
    SysTick_Config(SystemCoreClock/100);
    // Konfiguriere Pin1.0 als Ausgang
    XMC_GPIO_SetMode(XMC_GPIO_PORT1, 0, XMC_GPIO_MODE_OUTPUT_PUSH_PULL );

    // Konfiguriere Pin1.13 als Eingang ohne PullUp
    XMC_GPIO_SetMode(XMC_GPIO_PORT1, 13, XMC_GPIO_MODE_INPUT_INVERTED_TRISTATE);

    // ggf. Ereignissequelle aus dem Zustand RESET holen (nur für XMC4xxx nötig)
    XMC_SCU_INTERRUPT_ClearEventStatus(XMC_SCU_PERIPHERAL_RESET_ERU1);

    // Ereignis-Quelle konfigurieren
    XMC_ERU_ETL_CONFIG_t eruInit;
    memset(&eruInit, 0, sizeof(XMC_ERU_ETL_CONFIG_t));
    eruInit.input_a = ERU1_ETL1_INPUTA_P1_15;
```

```
    eruInit.edge_detection = XMC_ERU_ETL_EDGE_DETECTION_FALLING;
    eruInit.enable_output_trigger = true;
    eruInit.output_trigger_channel = XMC_ERU_ETL_OUTPUT_TRIGGER_CHANNEL0;

    // Ereignis Ziel konfigurieren
    XMC_ERU_OGU_CONFIG_t oguInit;
    memset(&oguInit, 0, sizeof(XMC_ERU_OGU_CONFIG_t));
    oguInit.service_request = true;

    // ERU initialisieren
    XMC_ERU_ETL_Init(ERU1_ETL1, &eruInit);
    XMC_ERU_OGU_Init(ERU1_OGU0, &oguInit);

    // NVIC konfigurieren
    NVIC_SetPriority(ERU1_0_IRQn, 15);
    NVIC_EnableIRQ(ERU1_0_IRQn);
}

int main(void)
{
    SystemInit();
    initApplication();
    do{
        // bleibt leer
    } while (true);
    return 0;
}

extern "C" void ERU1_0_IRQHandler(void)
{
    // LED toggle
    XMC_GPIO_ToggleOutput(XMC_GPIO_PORT1,0);
}


extern "C" void SysTick_Handler(void)
{
    // bleibt leer
}
```

Test

Übersetzen und übertragen Sie das Programm. Testen Sie die Anwendung.





Videozusammenfassung

Das war ein ziemlicher Aufwand den wir betrieben haben, um eine LED umzuschalten. Beachten sie jedoch, dass wir dies mit einem Interrupt gelöst haben. Dieses Programmiermodell ist zwar aufwändig, aber auch mächtig. Hier dieser Abschnitt wiederum als Videozusammenfassung.

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

besser auf youTube

Nächstes Thema

- spätestens jetzt in die UML reinschauen oder gleich zu
- Grundstruktur einer objektorientierten XMC Anwendung
- oder erst mal noch mit C weiter machen