

Tutorial 01: CanIf_AppDemo

The English version is [here](#).

Einleitung

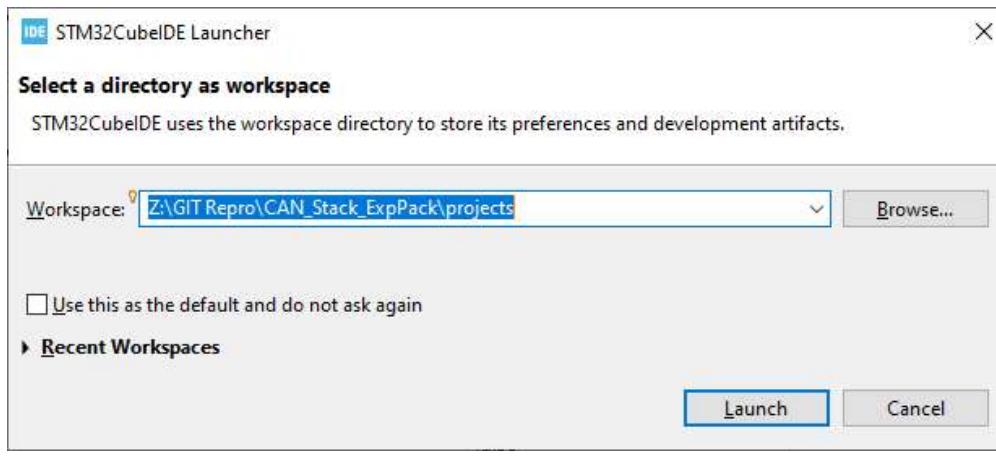
Dieses Tutorial beschreibt den grundsätzlichen Umgang mit dem ExpansionPack. Beginnend mit dem Aufsetzen eines neuen Projektes in der STM32CubeIDE über das Laden des Packs und dessen Konfiguration, sowie dem Senden (und später auch dem Empfangen) von einfachen CAN Messages. Die Tutorialbeschreibung ist als Einstieg für jeden gedacht und zur besser Verständlichkeit sehr Bilder lastig, man muss also relativ viel scrollen. Für jemanden der sich in der STM32CubeIDE oder auch in Eclipse schon zurecht findet, wird vieles obligatorisch sein.

Das STM32CubeIDE Expansion Package herunter laden

Es macht Sinn das ExpansionPack vor Beginn zu downloaden. Wenn man sich im Repository bewegt, hat man das ExpansionPack schon vorliegen. Nun auch auf [github](#)

Starten eines neuen Projektes

Nach dem Start der STM32CubeIDE wird man nach dem Workspace gefragt in welchem man arbeiten möchte. Ich nutze für das gesamte Projekt um den CAN Stack durchgängig den Workspace wie im folgenden Bild.



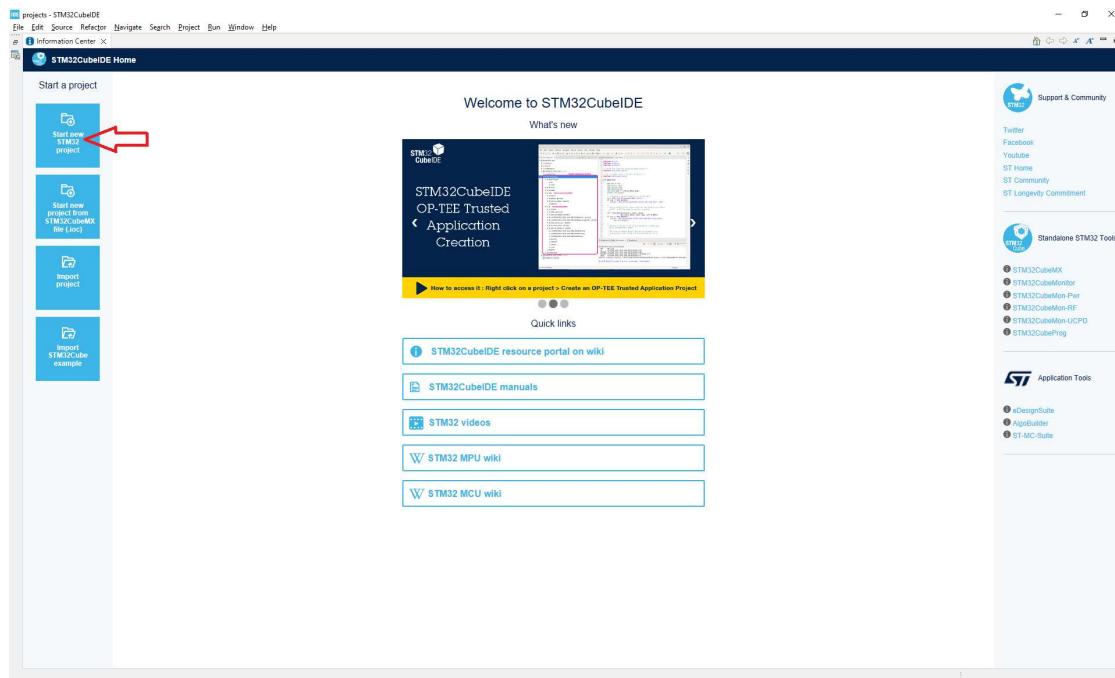
Select Workspace directory

Hierbei sei erwähnt, dass das "projects"-Verzeichnis eben auch jenem entspricht, welches als Spiegel zum Repository gepflegt wird. Möchte man das Tutorial unabhängig durch spielen, so macht es Sinn ein abweichendes Verzeichnis anzugeben.

Beim ersten Start der STM32CubeIDE wird man von dem "Information Center" begrüßt. Hier wählen wir natürlich für's Erste aus, dass wir ein neues Projekt starten möchten.

Table of Contents

- Einleitung
- Das STM32CubeIDE Expansion Package herunter laden
- Starten eines neuen Projektes
 - Kurzer Überblick über den Aufbau der main.c Datei
 - Wie kommt das Expansion Package in die STM32CubeIDE
 - Manage Software Packs
 - Auswahl der Module
 - Konfiguration der Module
 - CAN Driver
 - Bitrate Generator
 - bxCAN Master Setup / bxCAN Slave Setup
 - Software Message Buffer
 - CAN Interface
 - RX/TX PDU Configuration
 - Den Code für das CanIf Modul generieren
 - Das ExpansionPack übernimmt Aufgaben aus der HAL
 - Es wird Zeit für die Hardware
 - Was ist bei einem "normalen" Öffnen der IDE anders als beim ersten Start



Start new STM32 project

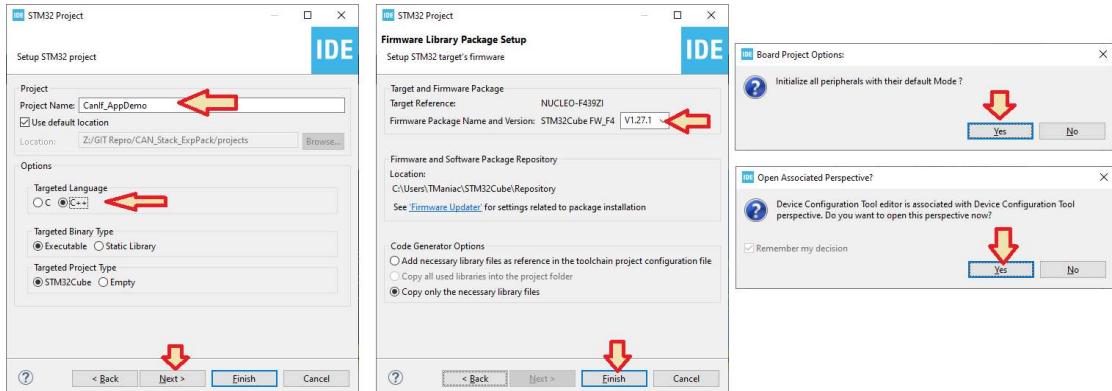
Später wird dieses "Information Center"-Fenster nicht unmittelbar beim Start angezeigt. Ich werde am **Ende des Tutorials** kurz auf den Neustart der STM32CubeIDE eingehen.

Ein STM32CubeIDE-Projekt (und auch ein STM32CubeMX-Projekt) baut natürlich auf einen bekannten STM32-Mikrocontroller auf. Ich habe ein Nucleo-F439ZI Board, auf welchen ich arbeite. Das NUCLEO-F439ZI hat, wie fast alle anderen NUCLEO-Boards auch, standardmäßig kein CAN. Man kommt aber über die "ARDUINO(R) UnoV3" Verbinder an die notwendigen Pins heran. So habe ich mir ein "Extension Board" mit zwei CAN Transceiver gebastelt. Wir werden später die verwendeten Port-Pins auswählen.

Commercial Part Number	Part Number	Product Name	Unit Price (US\$)	Mounted Device
	NUCLEO-F439ZI	STM32 Nucleo-144 development board with STM32F439ZI MCU, supports Arduino, ST Zio and morpho connectivity	23.0	STM32F439ZIT6
	NUCLEO-F413ZH	Nucleo-144		
	NUCLEO-F429ZI	Nucleo-144	Active	STM32F429ZIT6
	NUCLEO-F439ZI	Nucleo-144	Active	STM32F439ZIT6
	NUCLEO-F446ZE	Nucleo-144	Active	STM32F446ZET6

Select Board or MCU

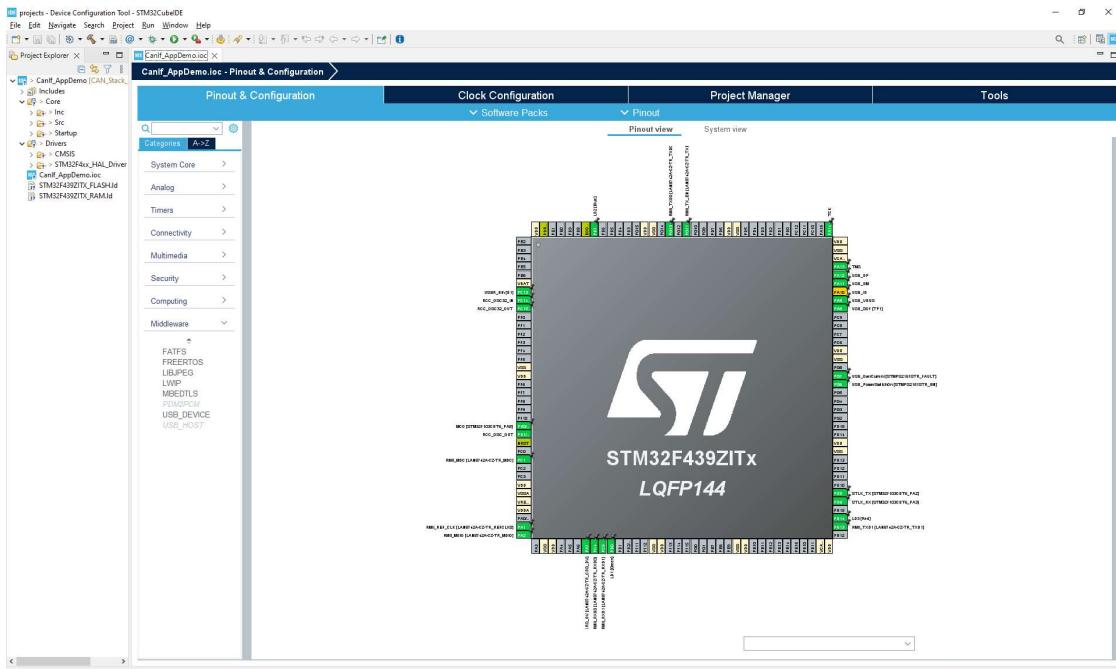
Die folgenden Dialoge sollten selbst erklärend sein. Wichtig ist, dass das ExpansionPack in C++ geschrieben ist. Die Voreinstellung ist hier leider immer C. Die Auswahl des verwendeten STM32Cube Firmware Package spielt aktuell noch keine Rolle, also sollte das aktuellste verwendet werden.



Steps to create project

Das Initialisieren der gesamten MCU-Peripherie ist nicht unbedingt erforderlich. Da es aber auch die Takterzeugung beinhaltet, ist es in den meisten Fällen sinnvoll.

Mit "Open Associated Perspective" wird in der STM32CubeIDE (entsprechend der Eclipse Umgebung) immer die Ansicht geöffnet, welche eben für den Arbeitsschritt am besten geeignet oder evtl sogar erforderlich ist. Hier wird im folgenden die Perspective geöffnet, welche die Konfiguration mit grafischer Unterstützung ermöglicht. Die Perspective sieht wie folgt aus:



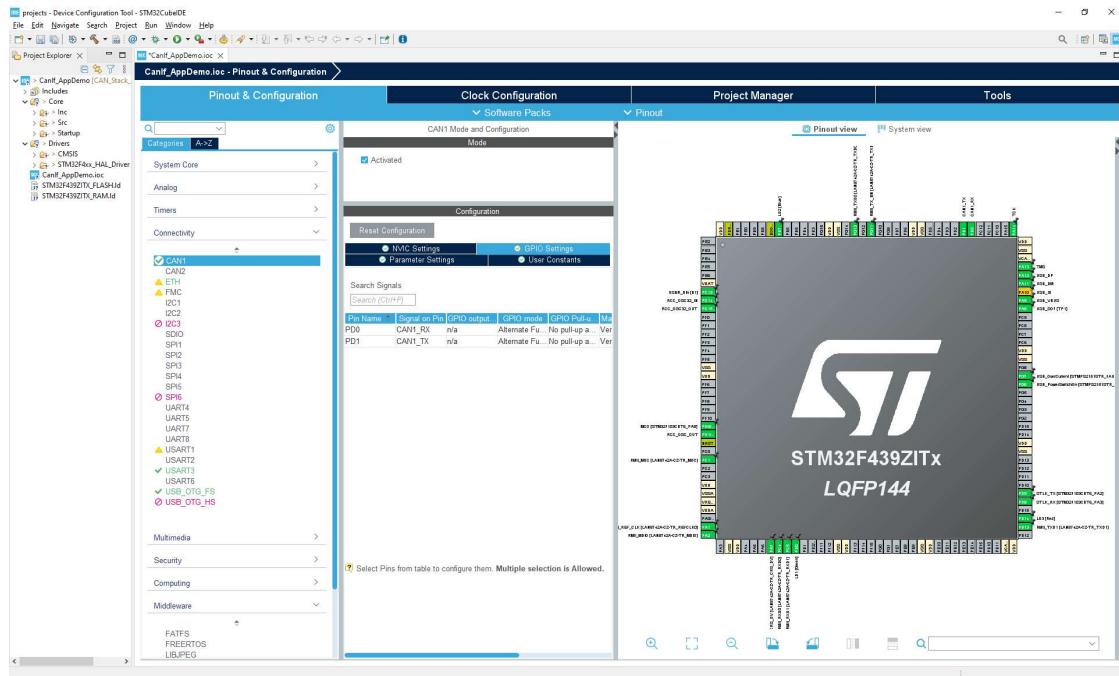
New project perspective

Ich möchte hier nicht explizit auf den Aufbau der STM32CubeIDE eingehen. Wichtig an dieser Stelle ist, dass wir auf der linken Seite den "normalen" Eclipse Project Browser haben. Der Großteil der Arbeitsfläche wird von der Darstellung der "*.ioc"-Datei verwendet. Die Ansicht des Controllers ist auf den jeweiligen Footprint angepasst. Wie man auf dem Bild erkennen kann, hat der STM32F439ZIT auf meinem NUCLEO-F439ZI Board einen LQFP144 Footprint.

Die STM32CubeIDE benennt die neu erzeugte "*.ioc"-Datei genauso wie das Projekt. Demnach haben wir jetzt eine "CanIf_AppDemo.ioc".

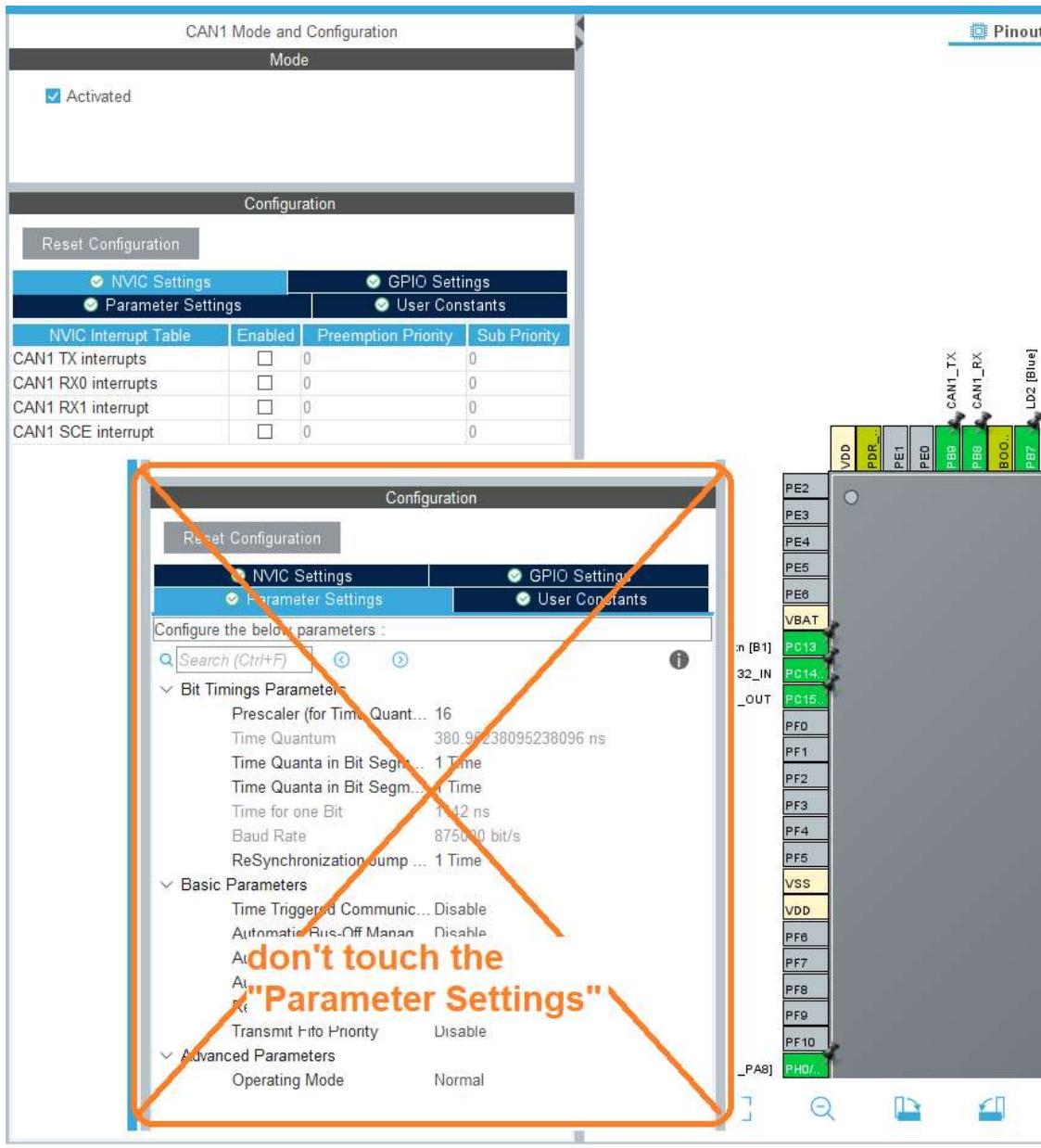
Da wir den CAN-Controller verwenden wollen, müssen wir diesen auch konfigurieren. CAN stellt eine "Connectivity" Peripherie dar. Also finden wir unter diesem Punkt auch die Module "CAN1" und "CAN2". Bei meinem NUCLEO-F439ZI gibt es standardmäßig keine Vorbereitung für CAN, also sind diese Module deaktiviert. Wählt man "CAN1" (oder auch "CAN2") aus, so erscheint eine zusätzliche Spalte mit der Bezeichnung "CAN1 Mode and Configuration". Die Darstellung mit diesen drei Spalten ist jener Aufbau welcher für die Konfiguration der STM32Cube-Elemente (welche intern immer als Pack gehandhabt werden) verwendet wird.

Im Bereich "Mode" aktiviert man die jeweiligen Module. Wir aktivieren nun das CAN1 Module durch setzen des Hakens "Activated".



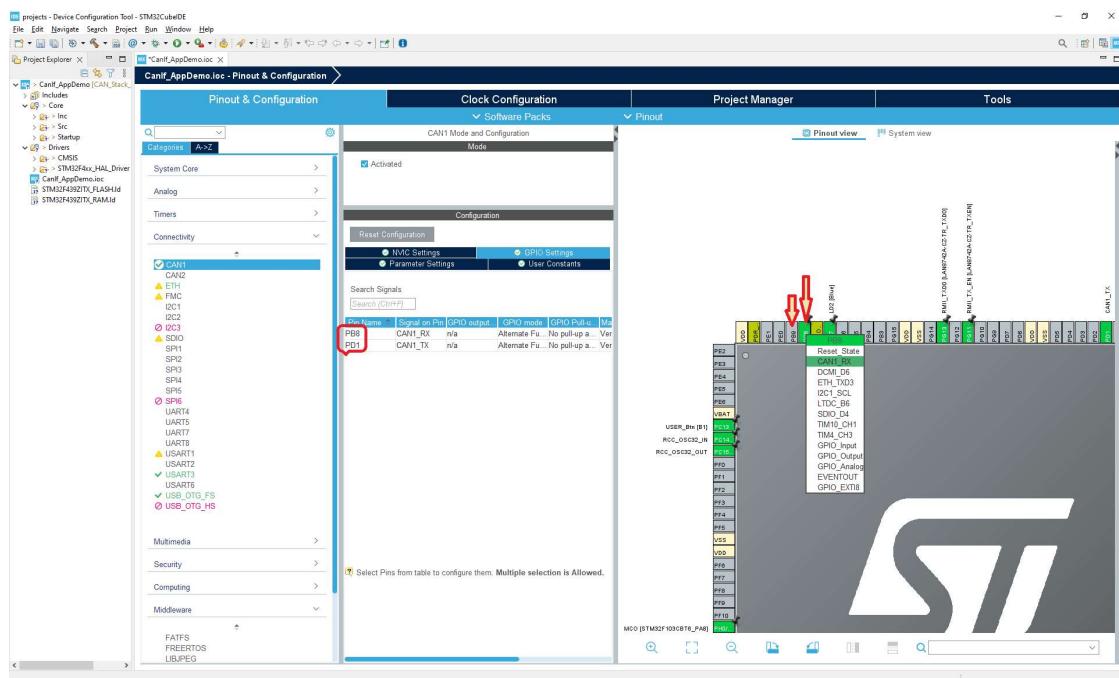
CAN1 activated

Jetzt werden im Bereich "Configuration" vier Tabs angezeigt. Der Tab "User Constants" spielt für uns keine Rolle. Dieser Tab wird in jeden Modul angezeigt. "Parameter Settings" beinhaltet die Startup Parameter des CAN-Controllers und NVIC Settings die Einstellung der verwendeten Interrupt-Callbacks. Das *ExpansionPack CAN Stack* übernimmt auch die Konfiguration der Startup Parameter, also lassen wir die "Parameter Settings" hier wie sie sind. Die Konfiguration der Interrupt-Callbacks werden wir später aufgreifen.



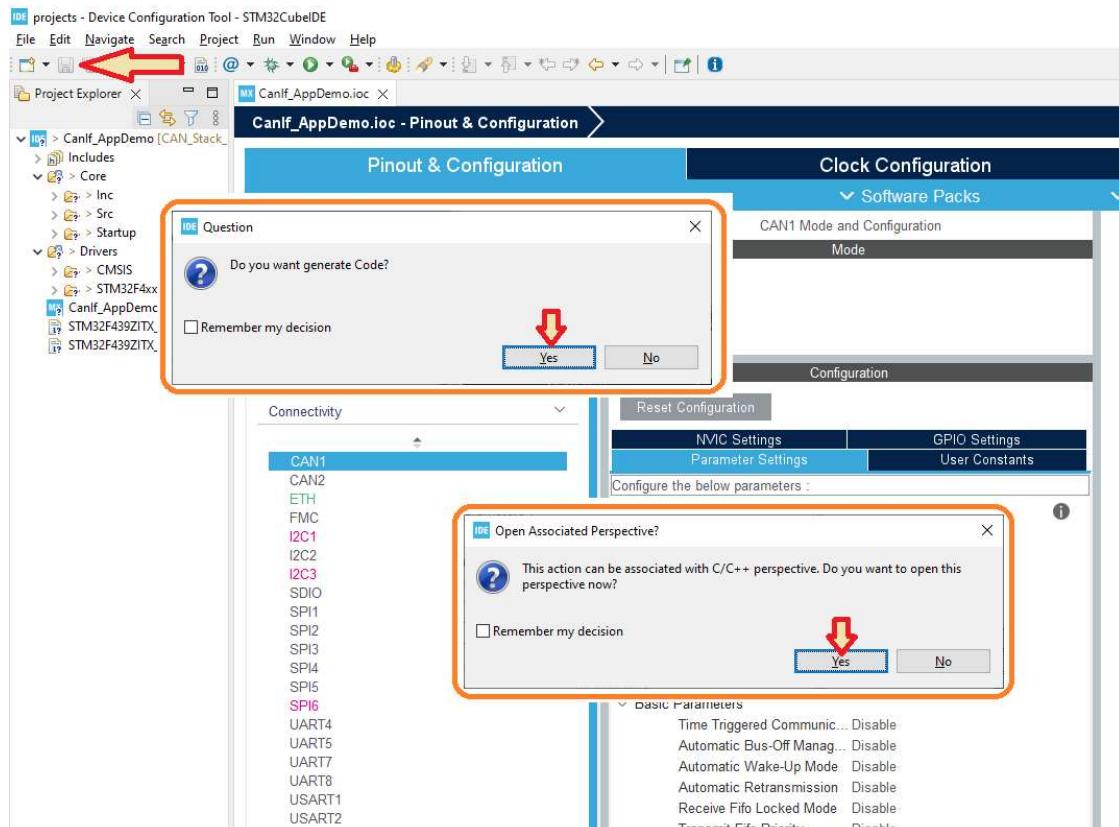
Parameter and NVIC Settings

Im Tab "GPIO Settings" werden die Port-Pins eingestellt, bzw angezeigt. Die Funktion der Pins lassen sich durch anklicken des Pins in der µC-Darstellung direkt auswählen. Bei meinem CAN-Transceiver-Board sind CAN1_RX an PB8 und CAN1_TX an PB9 angebunden. So nehme ich diese Einstellung hier auch vor.



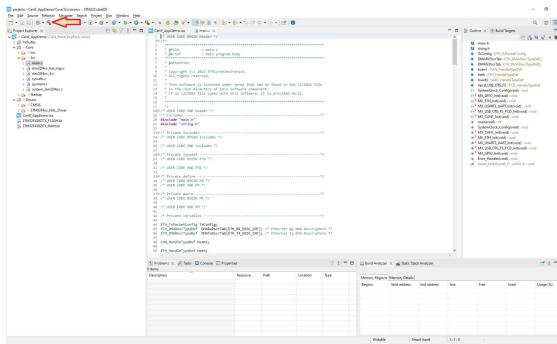
GPIO Settings

Nun wird es Zeit die Konfiguration das erste Mal zu speichern. Beim Klick auf die "Diskette" speichert STM32CubeIDE (entsprechend der Eclipse-Umgebung) die jeweils aktive Datei. Da wir gerade die "CanIf_AppDemo.ioc" bearbeitet haben, wird diese gespeichert. Beim speichern einer "*.ioc-Datei" fragt die STM32CubeIDE standardmäßig ob der Source Code generiert werden soll. (Das sollte man später im Hinterkopf behalten, wenn man testweise Code ändert.) Danach wird man gefragt ob man zur C/C++ Perspective wechseln möchte.



Save CanIf_AppDemo.ioc and generate code

Der erzeugte Code sollte build-fähig sein. Das können wir durch den Klick auf den "Hammer" probieren



Build generated code

Im Project Browser sind nun auch die neuen Dateien sichtbar. Die C/C++ Perspective bietet uns eine "Outline"-Übersicht über die aktive Datei. Im unteren Bereich der IDE findet man weitere hilfreiche Fenster. Details sollte man sich selber in diversen Eclipse Tutorials erarbeiten.

Kurzer Überblick über den Aufbau der main.c Datei

Ich möchte hier an Hand der main.c Datei ganz kurz auf ein paar Besonderheiten im Umgang mit der Code Generierung in der STM32CubeIDE eingehen. Vielleicht verschiebe ich das Kapitel mal und mache es etwas ausführlicher. Dann kann man auch Besonderheiten bei der Bearbeitung der *.ftl Templates eingehen.

```

main.c X MK_CanIf_AppDemo.ioc
1 /* USER CODE BEGIN Header */
2 /**
3  * @file           : main.c
4  * @brief          : Main program body
5  *
6  * @attention
7  *
8  * Copyright (c) 2022 STMicroelectronics.
9  * All rights reserved.
10 *
11 * This software is licensed under terms that can be found in the LICENSE file
12 * in the root directory of this software component.
13 * If no LICENSE file comes with this software, it is provided AS-IS.
14 *
15 */
16 /**
17 */
18 /* USER CODE END Header */
19 /* Includes -----*/
20 #include "main.h"
21 /**
22 * Private includes -----*/
23 /* USER CODE BEGIN Includes */
24 /**
25 * USER CODE END Includes */
26 /**
27 * Private typedef -----*/
28 /* USER CODE BEGIN PTD */
29 /**
30 * USER CODE END PTD */
31

```

how to work on main.c

STM nutzt eine Doxygen taugliche Formatierung, so dass man auch die Lizenzinformationen in der Source Code Doku einbinden kann. Der Dateikopf ist in einem

```

USER CODE BEGIN Header
[...]
USER CODE END Header

```

Block eingeordnet. Alles was in solch einem Block steht, wird bei einer erneuten Generierung nicht geändert. Diese Blöcke werden aber durch den Generator vorgegeben. Es macht also keinen Sinn selber solche Blöcke zu ergänzen. Man sieht hier zum Beispiel die "User Code" Blöcke für Includes und typedefs.

Es sei aber erwähnt, das der Code "verschwindet" wenn die Datei bei der Code Generation entfernt wird, zum Beispiel dann wenn man ein Modul deaktiviert.

```

40 /* Define handles */
41 /* @brief CAN handle structure definition */
42 CAN_HandleTypeDef hcan1;
43
44 UART_HandleTypeDef huart3;
45
46 /* USER CODE BEGIN PV */
47
48 /* USER CODE END PV */
49
50 /**
51 * @brief CAN handle Structure definition
52 */
53 #typedef struct _CAN_HandleTypeDef
54 {
55     /* CAN_TypeDef           *Instance;          /*!< Register base address */
56     /* CAN_InitTypeDef       Init;              /*!< CAN required parameters */
57     /* _IO HAL_CAN_StateTypeDef State;           /*!< CAN communication state */
58     /* _IO uint32_t          ErrorCode;         /*!< CAN Error code.
59     /* This parameter can be a value of @ref
60     /* @ref hal_error_t
61 }
62
63
64 /**
65 * @brief The application entry point.
66 */
67
68 /**
69 * @brief Application entry point.
70 */
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108

```

Handles to work with the hardware

STM nutzt "Handle"-Strukturen um mit der jeweiligen Peripherie (bis auf GPIO) zu arbeiten. Standardmäßig wird das Handle-Objekt im Kontext der `main.c` erzeugt. Dieses enthält neben dem Pointer zu den Registern auch eine Struktur zur Initialisierung.

```

50 /* Private function prototypes -----*/
51 void SystemClock_Config(void);
52 static void MX_GPIO_Init(void);
53 static void MX_USART3_UART_Init(void);
54 static void MX_CAN1_Init(void);
55 /* USER CODE BEGIN PFP */
56
57 /* USER CODE END PFP */
58
59 /* Private user code -----*/
60 /* USER CODE BEGIN 0 */
61
62 /* USER CODE END 0 */
63

```

Function declarations

Diese `MX_(HW-Module)_Init()` Funktionen werden für jedes Modul generiert. Logisch werden diese Funktionen weiter unten dann mit den Parametern aus dem `*.ioc` File gefüttert.

Darüber hinaus haben wir hier auch Platz um eigene (einfache) Funktionen zu deklarieren ("User Code * PFP") und auch zu definieren ("User Code * 0"). Wie gesagt, was man in solch einen Block hinein schreibt, bleibt auch bei einer Re-Generation erhalten.

```

63
64 /**
65 * @brief The application entry point.
66 */
67
68 int main(void)
69 {
70     /* USER CODE BEGIN 1 */
71
72     /* USER CODE END 1 */
73
74     /* MCU Configuration-----*/
75
76     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
77     HAL_Init();
78
79     /* USER CODE BEGIN Init */
80
81     /* USER CODE END Init */
82
83     /* Configure the system clock */
84     SystemClock_Config();
85
86     /* USER CODE BEGIN SysInit */
87
88     /* USER CODE END SysInit */
89
90     /* Initialize all configured peripherals */
91     MX_GPIO_Init();
92     MX_USART3_UART_Init();
93     MX_CAN1_Init();
94
95     /* USER CODE BEGIN 2 */
96
97     /* Infinite loop */
98
99     /* USER CODE BEGIN WHILE */
100    {
101        /* USER CODE END WHILE */
102
103        /* USER CODE BEGIN 3 */
104
105    }
106
107 }
108

```

main() Function

Die eigentliche `main()` Funktion ruft der Reihe nach die `Init()`-Funktionen auf und geht anschließend in die für Mikrocontroller typische `while(1)` Schleife.

Die Reihenfolge der `MX_(HW-Module)_Init()` Funktionen wird leider vom Code Generator vorgegeben. So lange man ausschließlich die HAL nutzt, spielt das keine Rolle. Da wir in unserem Expansion Pack aber HAL-

Aufgaben übernehmen wollen, braucht es da einen gezielten Ansatz.

Nach der main() kommen dann die einzelnen MX_(HW-Module)_Init() Funktionen. Als Beispiel hier nur kurz MX_CAN1_Init() Funktionen

```
154 */  
155 * @brief CAN1 Initialization Function  
156 * @param None  
157 * @retval None  
158 */  
159 static void MX_CAN1_Init(void)  
160 {  
161     /* USER CODE BEGIN CAN1_Init 0 */  
162     hcan1.Instance = CAN1;  
163     hcan1.Init.Prescaler = 16;  
164     hcan1.Init.Mode = CAN_MODE_NORMAL;  
165     hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;  
166     hcan1.Init.TimeSeg1 = CAN_BS1_1TQ;  
167     hcan1.Init.TimeSeg2 = CAN_BS2_1TQ;  
168     hcan1.Init.TimeTriggeredMode = DISABLE;  
169     hcan1.Init.AutoBusOff = DISABLE;  
170     hcan1.Init.AutoWakeUp = DISABLE;  
171     hcan1.Init.AutoRetransmission = DISABLE;  
172     hcan1.Init.ReceiveFifoLocked = DISABLE;  
173     hcan1.Init.TransmitFifoPriority = DISABLE;  
174     if (HAL_CAN_Init(&hcan1) != HAL_OK)  
175     {  
176         Error_Handler();  
177     }  
178     /* USER CODE BEGIN CAN1_Init 2 */  
179     /* USER CODE END CAN1_Init 2 */  
180 }  
181 }
```

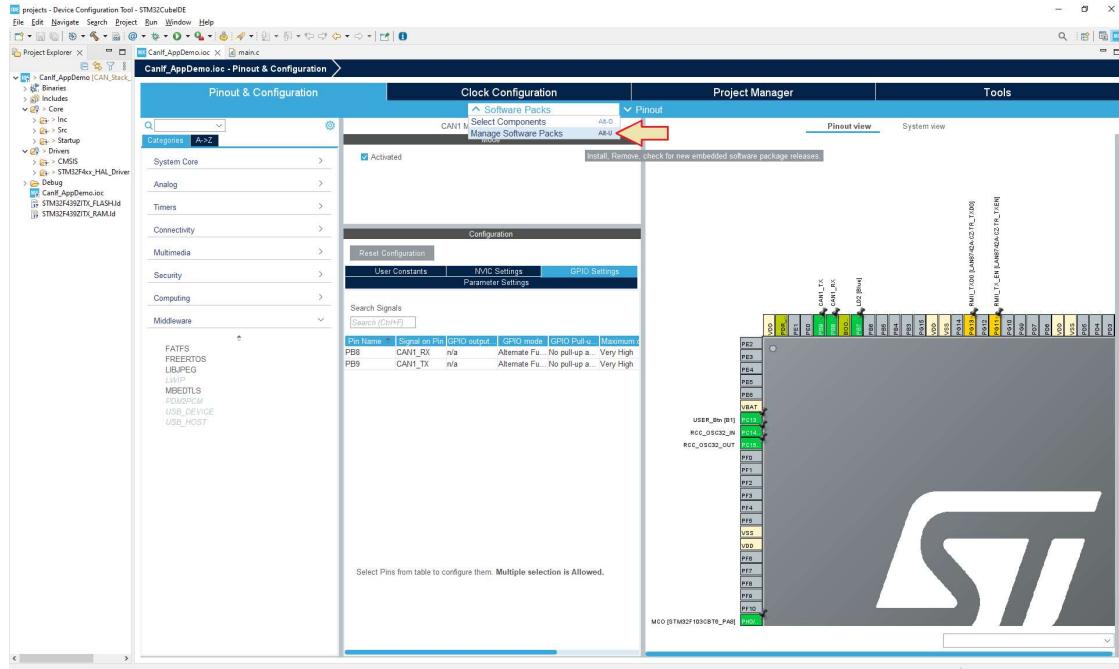
Init the CAN1 Controller

Grundsätzlich wird nur die Init-Struktur gefüllt und dann die eigentliche HAL_CAN_Init() aufgerufen. Nur wo werden die Pins zu geordnet. Dazu komme ich dann wenn es so weit ist.

Wie kommt das Expansion Package in die STM32CubeIDE

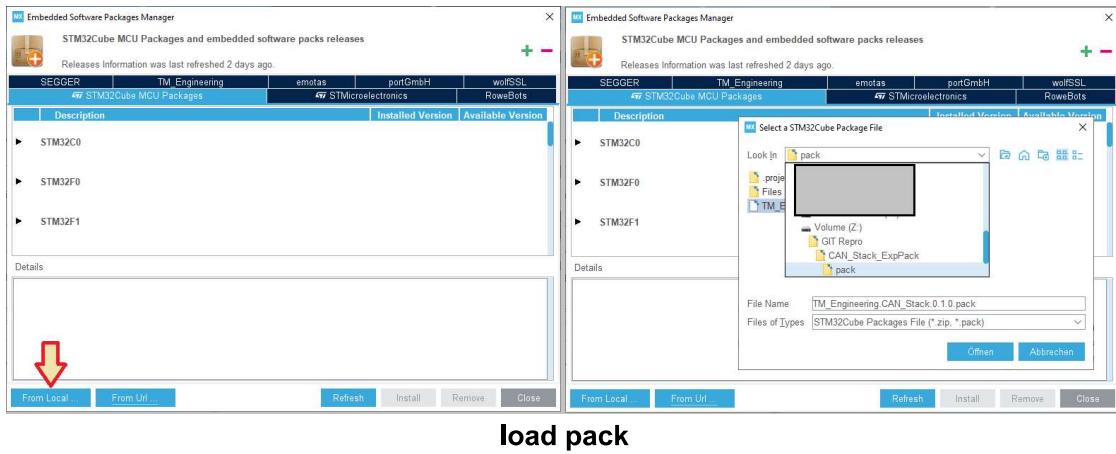
Manage Software Packs

Nun wollen wir aber endlich das STM32Cube ExpansionPack zu unserem Projekt hinzufügen. Als erstes müssen wir der IDE sagen wo sie das ExpansionPack findet.

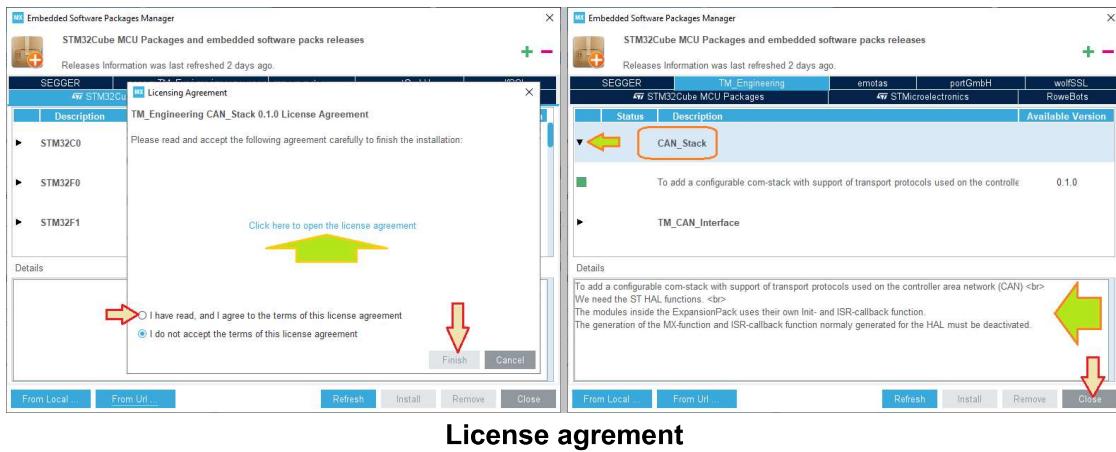


Open manage software packs dialog

Wir gehen also wieder in die Ansicht der CanIf_AppDemo.ioc Datei und wählen "Manage Software Packs".



Man kann das ExpansionPack entweder lokal von der Festplatte oder per URL aus der Ferne integrieren. Wir hatten **am Anfang** das ExpansionPack herunter geladen (oder wir haben es parallel in einem Development-Verzeichnis). Dieses ExpansionPack suchen wir hier und wählen es aus.

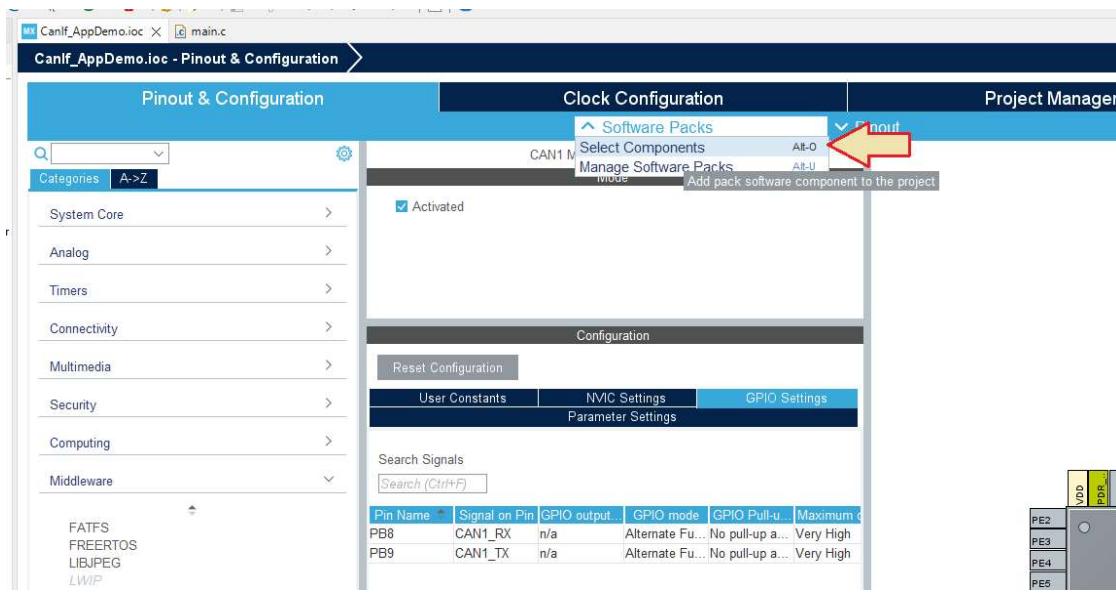


Wenn wir dann die Lizenzbedingungen gelesen und akzeptiert haben, ist das ExpansionPack integriert.

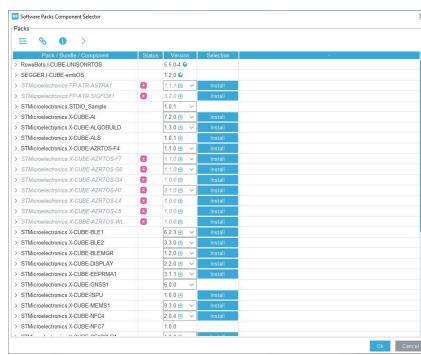
Man findet hier in dem Fenster auch ein paar nützliche Informationen zum Umgang mit dem ExpansionPack. Da steht zum Beispiel etwas zu den MX-Funktionen.

Auswahl der Module

Jetzt müssen wir noch auswählen, welche Komponenten aus dem ExpansionPack genutzt werden sollen.

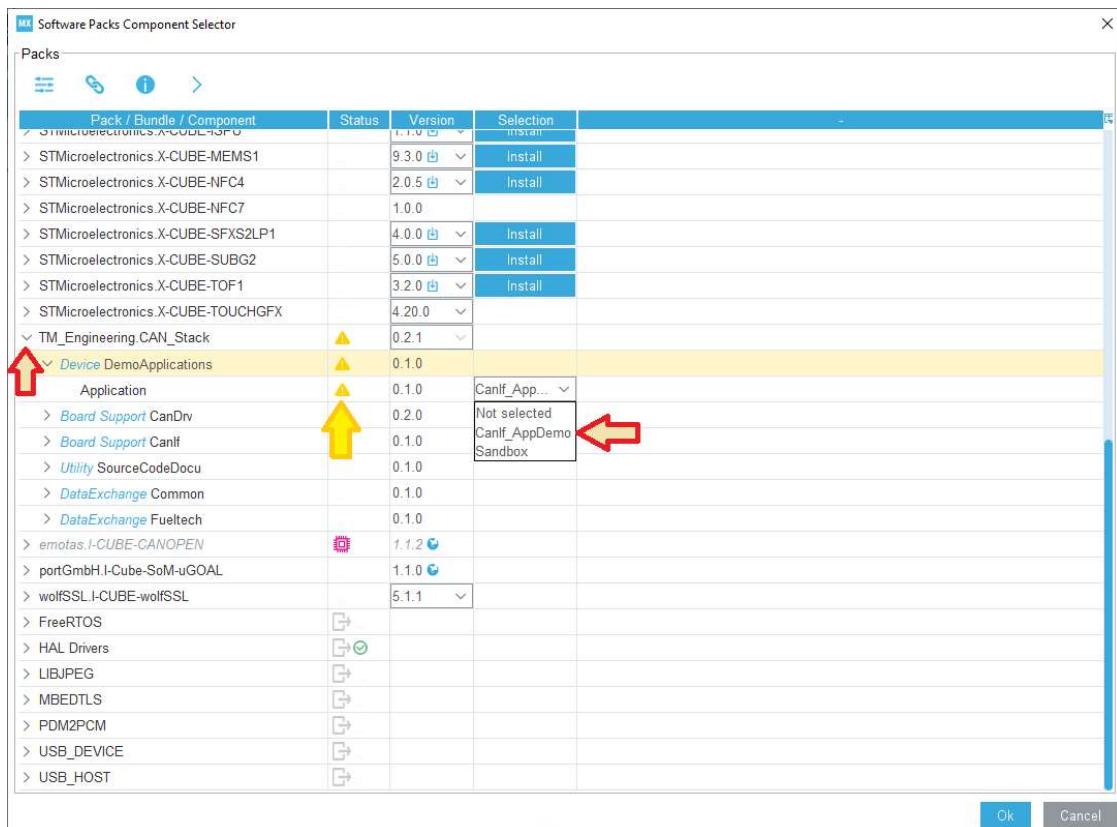


Es öffnet sich der "Software Pack Component Selector". Erst einmal ein kleines Bild. Ins Detail gehen wir gleich.



Software Pack Component Selector

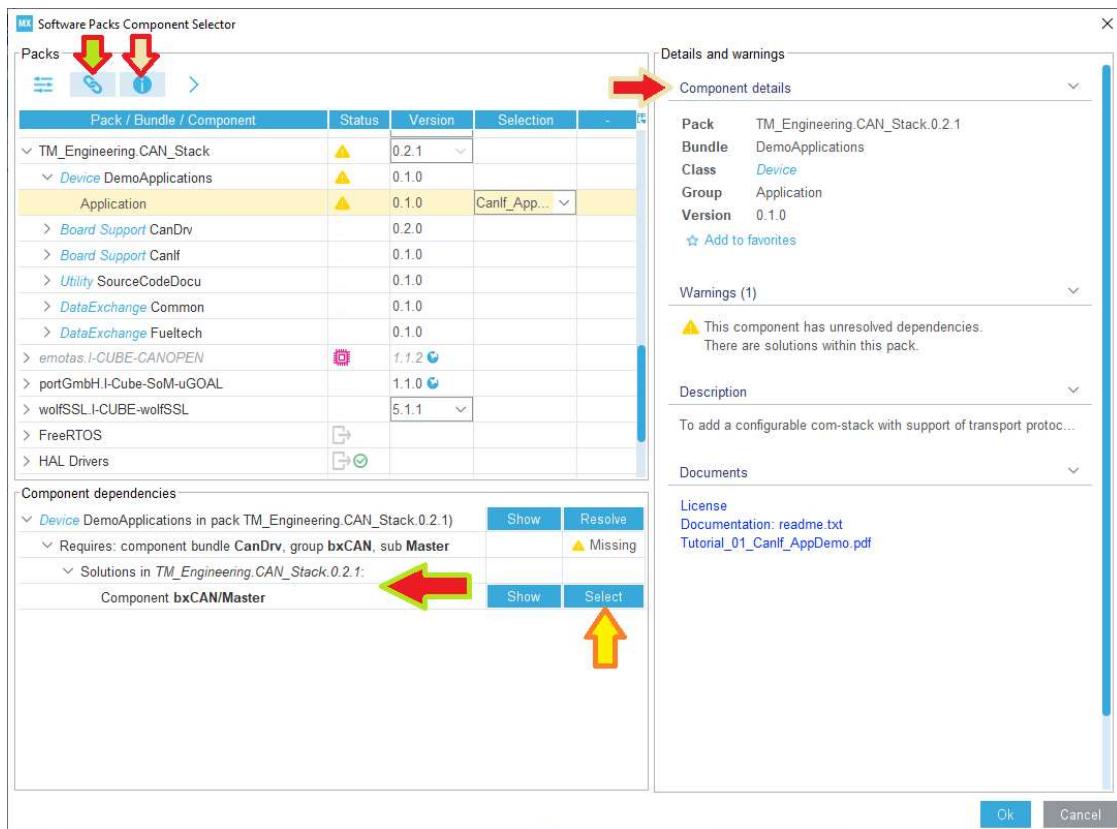
Hier sind unter anderem alle von STMicroelectronics verfügbaren ExpansionPacks auswählbar. Klickt man auf Install wird das jeweilige ExpansionPack bei STM herunter geladen und installiert. Wir scrollen aber ein wenig nach unten und suchen **TM_Engineering.CAN_Stack**.



Select the Demo Application

Durch Klick auf ">" kann man die einzelnen Module, oder genauer gesagt die einzelnen Bundles öffnen. Naiv wählen wir die Demo Application **CanIf_AppDemo** (deswegen haben wir unser Projekt am Anfang auch so benannt) aus.

Nun werden da aber ein paar gelbe Warnschilder mit Ausrufezeichen angezeigt.



Bundle warnings and Info

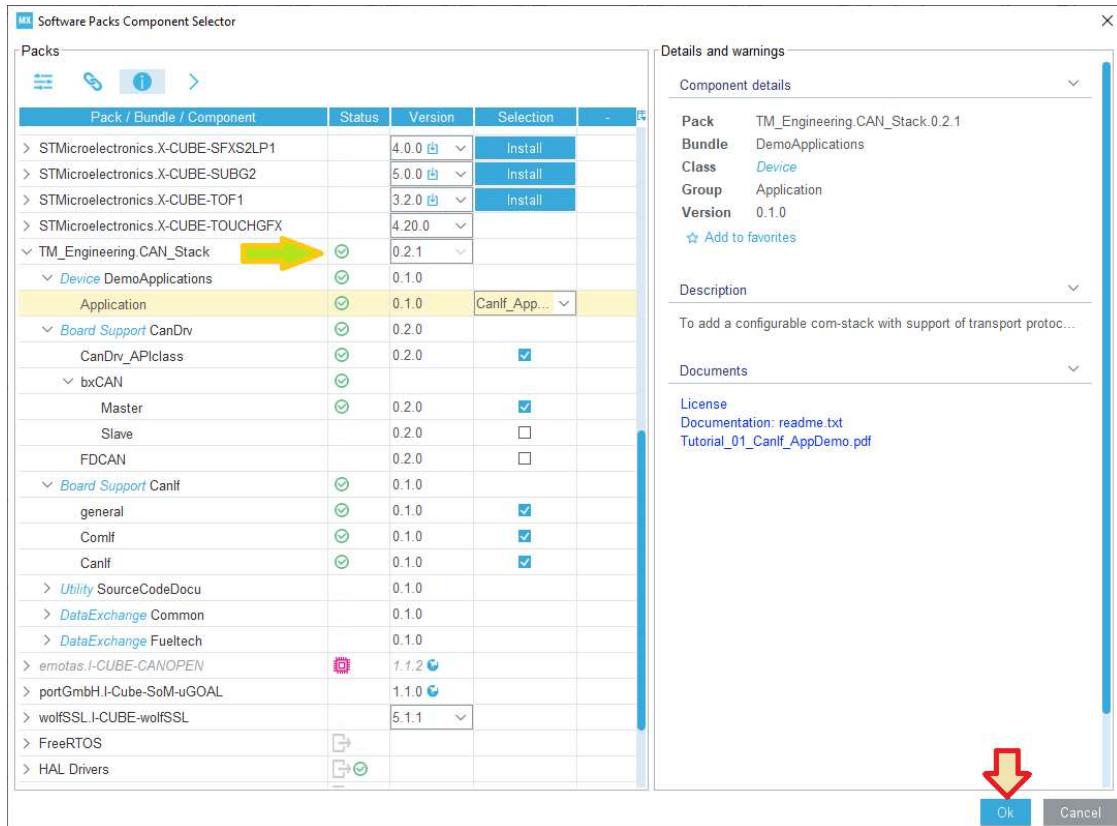
Klickt man auf den Button (siehe beige/roter Pfeil) kann man ein paar Informationen zum jeweils aktiven Element bekommen. So steht zum Beispiel bei unserer Application: "This component has unresolved dependencies". Es braucht also noch irgend ein anderes Modul. Weiter steht da "There are solutions within this pack". Die Lösung des Problems liegt also nahe.

Mit dem Button (grün/roter Pfeil) dann kann man die Abhängigkeiten unmittelbar anzeigen. In unserem Fall braucht die DemoApplication eine Komponente bxCAN/Master

Wir ergänzen also die noch notwendigen Module die sich alle im Bundle Board Support befinden.

- Man kann durch Auswahl der einzelnen Module versuchen das Problem selber zu lösen
- Man kann auch einzelne Probleme mit "Select" lösen. (So lernt man vielleicht etwas über den Aufbau des Packs)
- Mit "Show" kann man die Lösung vor selektieren.
- Und mit "Resolve" löst man alle verketteten Probleme mit einmal. Also in unserem Fall werden die fünf erforderlichen Module ausgewählt.

Bundle	Module	Inhalt
CanDrv	CanDrv_APIClass	Das ist eine virtuelle Klasse Can . Diese stellt das API für alle kompatiblen CanDrv bereit
	bxCAN / Master	Dies ist die Umsetzung des CanDrv_bxCAN für den STM bxCAN Controller. Für das Tutorial reicht uns der Master Controller
CanIf	general	Hier sind die Header Files mit den Compiler Abhängigkeiten oder auch eine Klasse zur Handhabung von Versionsinformationen zur Laufzeit enthalten
	ComIf	Darin ist eine virtuelle Klasse enthalten, welche zur Abstraktion von Kommunikation Interface Klassen ähnlich dem AUTOSAR Ansatz vorgesehen ist



Select needed modules

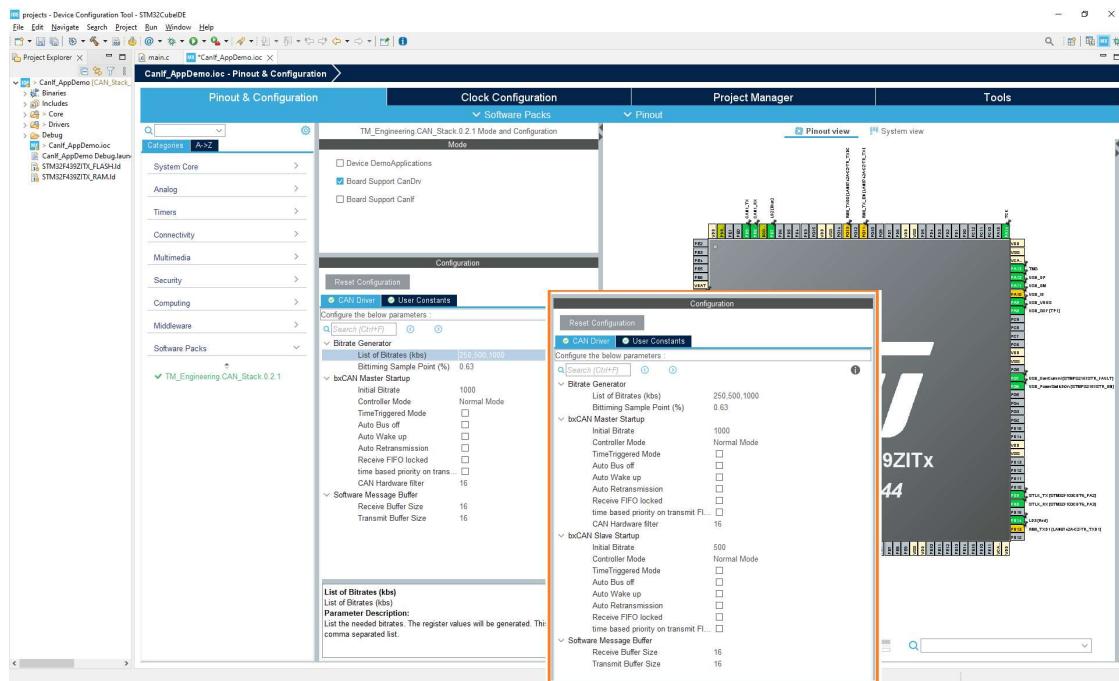
Haben wir diese fünf Module aktiv, werden wir mit grünen Häkchen belohnt und können den Selector wieder schließen.

Konfiguration der Module

Das Aktivieren der Module erfolgt auf dem gleichen Weg, wie wir zu Beginn das CAN-Modul aktiviert haben. In der linken Spalte des CanIf_AppDemo.ioc Viewers findet man jetzt einen weiteren Punkt "Software Packs". Unter diesen Punkt werden alle aktiven ExpansionPacks aufgeführt. Aktuell haben wir nur ein Pack, also ist hier auch nur unser TM_Engineering.CAN_Stack zu finden. Aktiviert man hier die Module erhält man im Configuration Abschnitt die Tabs "CAN Driver", "CAN Interface" und "User Constants". Der Tab "User Constants" ist der gleiche den wir schon aus den Einstellungen des CAN-Modules kennen.

CAN Driver

Der Tab CAN Driver beinhaltet die Konfigurationsparameter welche unmittelbar auf die Hardware wirken. In der AUTOSAR Spec werden diese Parameter durch das CAN-Modul bearbeitet.



Config the Hardware Settings of the CanDrv module

Auch in dem "Configuration" Abschnitt gibt es wieder einen (i) Button um mehr über die einzelnen Parameter zu erfahren.

Bitrate Generator

Mit diese Parameter werden durch den Source Code Generator die notwendigen Hardware Parameter für das Bit Timing berechnet um die jeweiligen Baudraten zu erzeugen. Neben diesen Parametern wird die Clock des CAN Modules aus den Systemparametern genutzt.

Remarks

Aktuell existiert kein Support für System Clock Switch während der Laufzeit!!!

Parameter	Beschreibung
List of Bitrates	Hier wird die Liste der gewünschten Bitraten angelegt. Die einzelnen Bitraten werden durch Kommas getrennt. Die hier gewählten Baudraten sind während der Laufzeit auswählbar.
Bittiming Sample Point	Sample Point als Prozent oder Kommazahl. Der Sample Point muss entweder größer 50% oder eben $0.5 < x < 1$ sein

bxCAN Master Setup / bxCAN Slave Setup

Hier werden die Startup Parameter des bxCAN Master Controllers festgelegt.

Wenn man den bxCAN Slave auch aktiviert hat wird ein identischer Abschnitt für diesen angezeigt.

Parameter	Wert	Beschreibung
Initial Bitrate	[Zahl, Integer]	Hier kann eine Bitrate ausgewählt werden welche zuvor im Bereich "Bitrate Generator" definiert wurde
Controller Mode	Normal	Standard. Der Controller nimmt ganz normal am Bus teil
	Loopback	CAN-Tx und CAN-Rx des Controllers werden innerhalb des µC verbunden. Damit lässt sich ohne weiteren Teilnehmer ein aktives Busverhalten

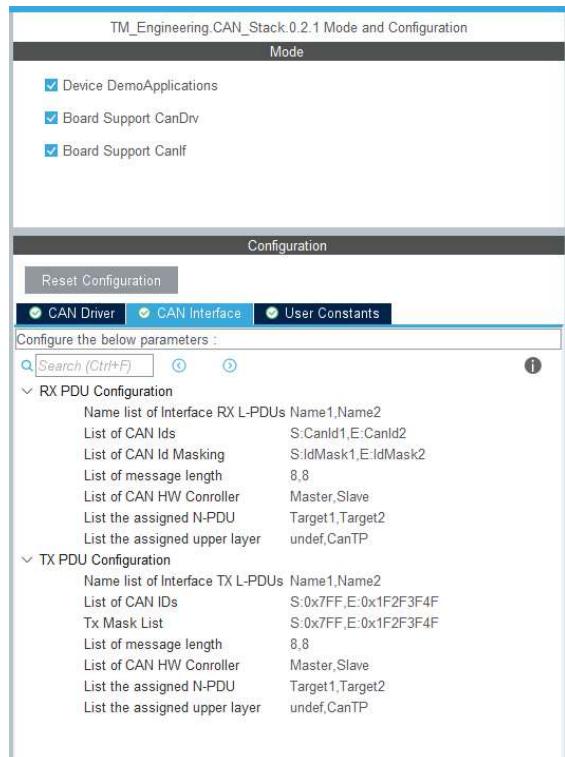
		simulieren.
	Silent	Der Controller nimmt nicht an der Arbitrierung Teil. Damit kann man auf einem Bus mithören ohne diesen zu beeinflussen
	Silent & Loopback	Kombination aus den beiden
Time Triggered Mode	true / false	Zeitstempel
Auto Bus off	true	Der CAN Controller geht nach einem Bus-Off von allein in den aktiven Zustand
	false	Der CAN Controller muss per Software vom Bus-Off in den aktiven Zustand gebracht werden.
	-	Der Bus-Off Zustand kann erst verlassen werden wenn 128 mal 11 rezessive Bits empfangen wurden (Siehe CAN-Error Management)
Auto Wake up	true	Der CAN Controller wacht bei eingehenden Nachrichten von alleine auf
	false	Der CAN Controller muss per Software geweckt werden
Auto Retransmission	true / false	Wählt aus ob die Nachrichten bei fehlgeschlagener Arbitrierung erneut gesendet werden sollen
	Attention	This value is inverted to the Bit 4 NART: No automatic retransmission of the bxCAN controller. The inversion is done inside the HAL Init function.
Receive FIFO Locked	true	Neue Nachrichten überschreiben den vollen Rx-FIFO
	false	Es werden keine weiteren Nachrichten mehr angenommen, wenn der Rx-FIFO voll ist
time based priority on transmission	true	Das Senden erfolgt in der zeitlichen Reihenfolge wie die Messages dem Tx-FIFO übergeben werden
	false	Das Senden unterliegt den normalen Arbitrierungsregeln, es wird also entsprechend der CAN-Id der Vorrang gegeben
CAN Hardware filter	[Zahl, Integer]	Gibt die maximale Anzahl der genutzten Hardware Filter für diesen Controller an. Der bxCAN Slave hat die restlichen Filter des bxCAN Masters zur Verfügung

Software Message Buffer

Der bxCAN Controller stellt nur einen drei Nachrichten großen Receive FIFO und sowie drei Transmit Mailboxen zur Verfügung. Ich habe dem Modul je einen Software Buffer für Receive und Transmit spendiert. Die Größe kann hier über die Parameter eingestellt werden.

CAN Interface

Ich muss hier dann noch das System mit den PDUs erklären.



Parameter on CAN Interface

RX/TX PDU Configuration

Alle Parameter sind als Komma-Listen zu benutzen. Das heißt auch, dass in jeder Liste die gleiche Anzahl an Elementen vorhanden sein muss. Stimmt die Anzahl der Elemente eines Parameters nicht, so wird der Source Code Generator eine Fehlermeldung generieren.

In diesem Fall sind die generierten Konfigurationsdateien nutzlos.

Todo:

Vielleicht kennt jemand ja einen Weg diese Konfiguration in Tabellenform ähnlich der FreeRTOS Konfiguration in der STM32CubeIDE umzusetzen.

Ziel ist es, dass die Konfiguration hier nur für Nachrichten ohne Übertragungsprotokoll notwendig ist. Das [CanFT2p0](#), sowie das [IsoTp](#)-Protokoll führen die Konfiguration schon selber durch.

Kurzbeschreibung

Die Tabelle gibt einen ersten Überblick. Die Parameter werden danach im Detail beschrieben.

Parameter	Wert (Aufbau)	Beschreibung
Name L-PDU	Name	Name als String
CAN Id	S:0x123	Standard 11bit Identifier in hexadezimal Format
	E:0x123456	Extended 29bit Identifier in hexadezimal Format
CAN Id Masking	S/E:0x0	Aufbau genauso wie CAN Id
HW Controller	Master oder M	wählt den bxCAN Master als Controller aus
	Slave oder	wählt den bxCAN Slave als Controller aus

	S	
assigned N-PDU	Name	gibt den Namen der Nachricht im UpperLayer (was in der Regel das Übertragungsprotokoll ist) an
assigned upper layer	undef	es wurde kein spezielles UpperLayer ausgewählt. Die Message muss im CAN Interface abgeholt werden
	CanTP	das UpperLayer dieser Message ist das IsoTP
	CanFT	das UpperLayer dieser Nachricht ist das CANFT2.0 Protokoll
	...	die möglichen Protokolle können erweitert werden.

Name list of Interface L-PDUs

Liste der L-PDU Namen.

Mit Hilfe der PDU-Namen wird der jeweilige "Kommunikationspfad" ausgewählt. So wird zum Beispiel der Funktion `CanIf::Transmit()` die TxPduld und ein Pointer zu den zu sendenden Datenbytes übergeben. Die Namen werden hier in der Oberfläche immer ohne irgendwelche Prefixe oder Ergänzungen genannt. Im Code werden die Namen dann mit dem Prefix des jeweiligen Modules sowie mit Rx-/Tx-Kennung versehen. Die erzeugten Namen aller Module sind dann in der EcuNames_Cfg.h zu finden.

Wir können im Tutorial die Namen erst einmal so belassen. Ich werde bei den folgenden Punkten nur dann explizit erwähnen, das wir etwas ändern, wo es notwendig ist.

List of CAN Ids / List of CAN Id Masking

Diese zwei Parameter werden zur Konfiguration der Hardware Filter genutzt.

Das erste Zeichen gibt an ob das Extended Id Bit (IDE) gesetzt werden soll. Aktuell wird nur geprüft ob das erste Zeichen ein 'E' oder ein 'e' ist. Ist dies nicht der Fall, dann wird von Standard CAN ausgegangen. Das Trennzeichen an zweiter Stelle ist egal. Alle Zeichen ab der dritten Stelle bis zum nächsten Komma werden unverändert in den Source Code übernommen. Es stehen also alle gängigen Formationen (hex, dec, oct, bin) zur Verfügung.

CanId1, CanId2 oder auch IdMask1 und IdMask2 sind keine gültigen Zahlen. Demnach müssen wir hier im Tutorial gültige CAN-Ids eingeben

List of message length

Anzahl der Datenbytes

List of CAN HW Controller

Der verwendete CAN Controller per Name. Aktuell werden die Namen des bxCan Modules unterstützt. Also können Master oder Slave benannt werden. Alternativ kann mit dem ersten Buchstaben, also 'M' oder 'S' abgekürzt werden.

Die FDCAN Controller der neueren STM32 werden durchnummeriert. Ich werde aber auch Master als CAN1 und Slave als CAN2 akzeptieren.

Wir nutzen im Tutorial nur den Master des bxCAN. Also müssen wir aus dem Slave auch einen Master (oder Kurzform "M") machen.

List of assigned N-PDU

Hier wird der Name des übergelagerten PDU benannt. Diese Namen sollten durch die Konfiguration des jeweiligen Protokollmodules entstehen

Im Tutorial nutzen wir keine höheren Layer. Da diese Namen trotzdem vom Compiler gesucht werden, müssen diese Dummynamen bleiben.

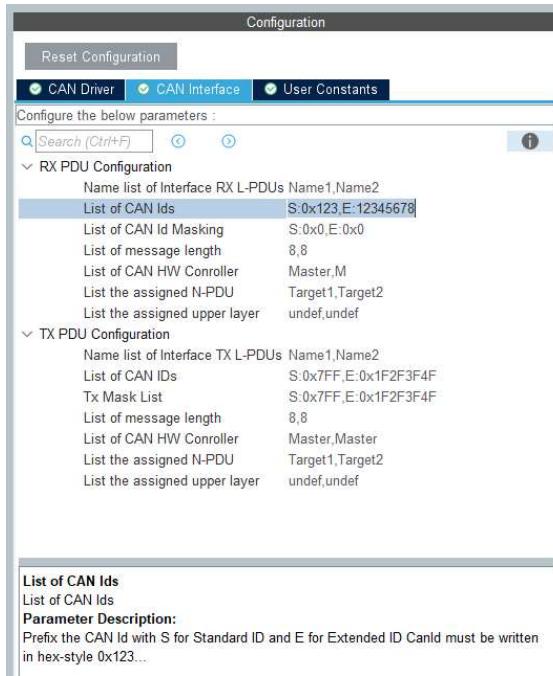
List the assigned upper layer

Verwendetes Transport Layer (siehe auch [CanIf_UpperLayerType](#)). Das **CanIf** muss wissen an welches Protokollmodul eine empfangene Nachricht weiter gereicht werden muss, bzw wohin ein erfolgreiches Versenden zurück gemeldet werden muss.

AUTOSAR nutzt hierzu die Kombination aus "übergeordneten Layer" und "PDU-Id". Mit C++ lässt sich das auch einfach in Funktionspointeraufrufen realisieren. Es wird also die Zukunft zeigen, wie weit wir das hier nutzen.

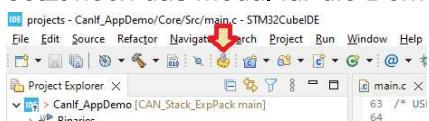
Damit das **CanIf** nicht versucht das IsoTP-Protokoll anzusprechen, müssen wir das CanTP zweimal zu undef ändern.

Nach der Änderung der Parameter sollte die Konfiguration so aussehen

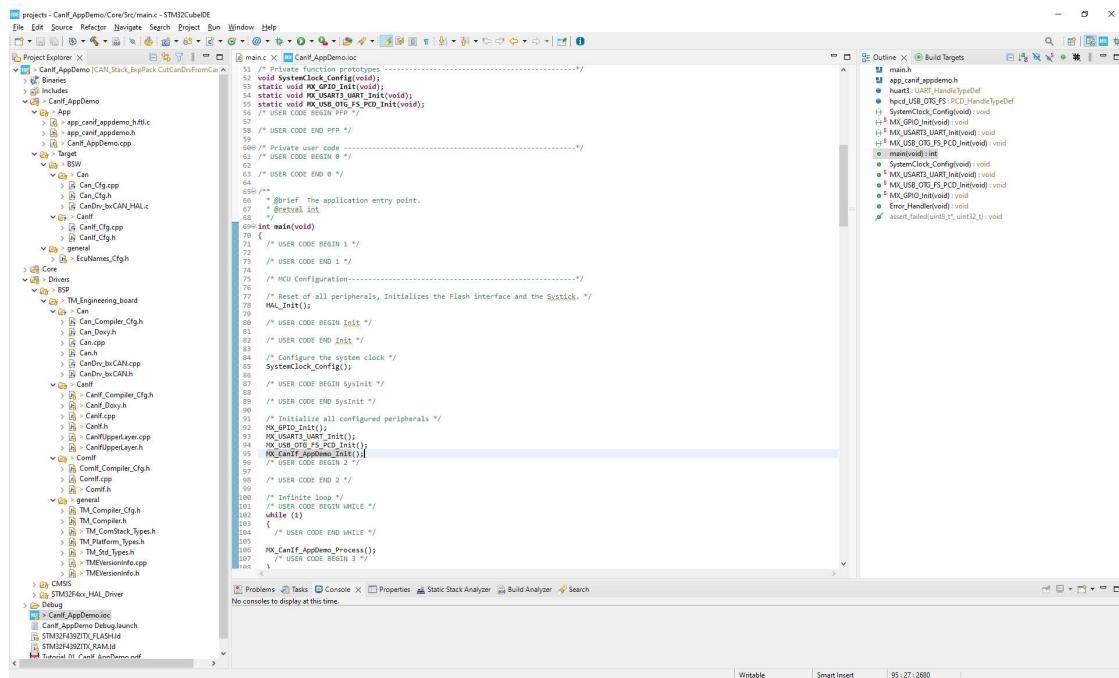


Den Code für das CanIf Modul generieren

Jetzt noch das Modul für die DemoApplication aktivieren und wir können den Code Generator starten.



Wenn man jetzt im Projekt Browser mal in die vielen neuen Verzeichnisse hinein schaut, wird man neue Dateien entdecken.



All generated files

Alles was sich im Verzeichnis mit dem Application Namen befindet, sind generierte Dateien. Im Unterverzeichnis "Target" werden die eigentlichen Konfigurationen erzeugt. Das Verzeichnis "App" beinhaltet die eigentliche Application.

Die generierten Dateien verhalten sich sich genauso wie wir es in der main.c kennen gelernt haben. Es werden also nur Änderungen innerhalb der "User Code Blöcke" über eine Code Generation hinweg beibehalten.

Schaut man zum Beispiel einmal in die EcuNames_Cfg.h so findet man ziemlich am Anfang (die Kommentare sind für die Doku geändert)

```

/* USER CODE BEGIN EcuNames_Cfg_h 0 * /

/* ** @brief this enum shows the naming conventions of used PDU names * /
typedef enum
{
    N_PDU_Dummy_for_Test,
    L_PDU_Dummy_for_Test,
    CanUndef1_Rx_Target1,
    CanUndef1_Rx_Target2,
    CanUndef1_Tx_Target1,
    CanUndef1_Tx_Target2
}CanUL_PDU_for_Test;

/* USER CODE END EcuNames_Cfg_h 0 * /

```

Man kann dieses enum also frei bearbeiten. Hier sind auch unsere Rx und Tx PDU Namen Target1 und Target2 als Dummy angelegt.

Im Verzeichnis "Drivers/BSP" gibt es jetzt ein Verzeichnis "TM_Engineering_board". Darin sind die kopierten Dateien aus dem Board Support Package. Diese Dateien werden bei jeder Code Generation neu hier her kopiert. Änderungen da drin können also nur zu Versuchszwecken durch geführt werden und sind nicht angedacht.

In der oben geöffneten main() kann man auch erkennen das eine MX_CanIf_AppDemo_Init() und eine MX_CanIf_AppDemo_Process() Funktion aufgerufen wird. Beide Funktionen sind in der CanIf_AppDemo.cpp

implementiert. Wir können also abseits der eigentlichen `main()` mit C++ objektorientiert weiter arbeiten.

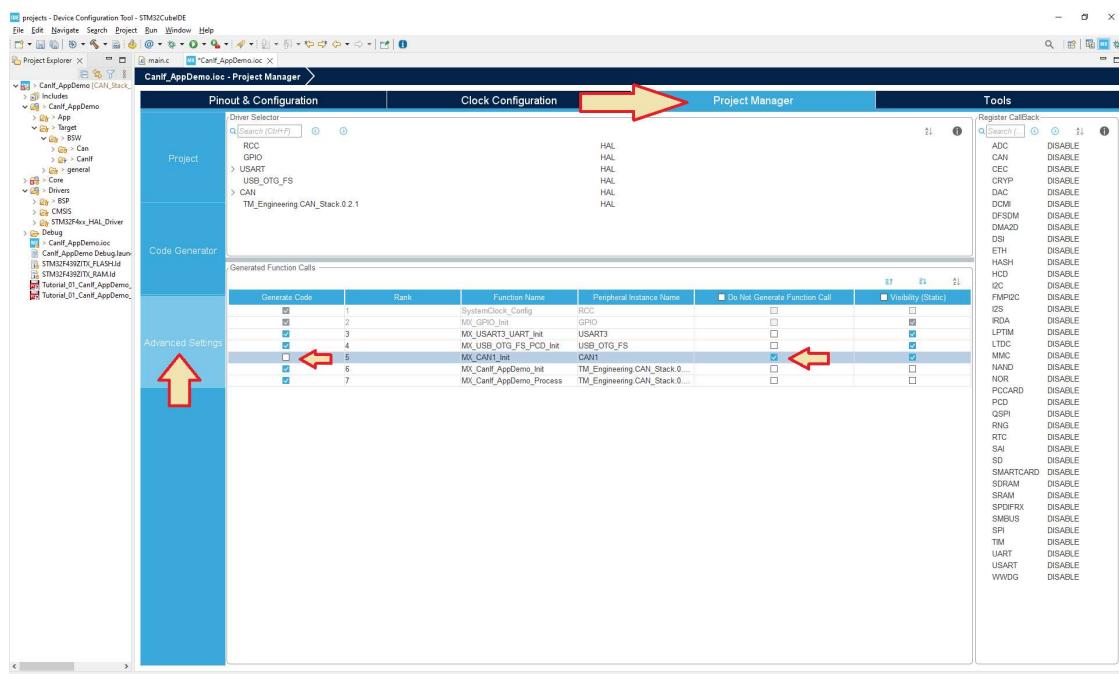
Das ExpansionPack übernimmt Aufgaben aus der HAL

Also wollen wir mal ausprobieren ob sich die "CanIf_AppDemo" bauen lässt.

Linker error multiple definition of 'HAL_CAN_MspInit'

Wir erinnern uns daran, dass in der Information des ExpansionPack etwas stand, dass wir das Generieren der HAL-Funktionen deaktivieren müssen. Und genau das ist der Grund für diesen Linker Fehler.

Wir müssen also noch ein mal in die Konfiguration zurück.



Advanced Project Settings

Der CanIf_AppDemo.ioc Viewer hat ein Tab "Project Manager". Hier findet man verschiedene Einstellungsmöglichkeiten für den Umgang mit dem STM32Cube Code Generator. Im Bereich "Advanced Settings"

findet man die Möglichkeit die Generierung für die aktiven Module zu beeinflussen. So kann man unter anderem auch wählen ob man die HAL oder gar eine Low Level Ansatz für die Module nutzen möchte.

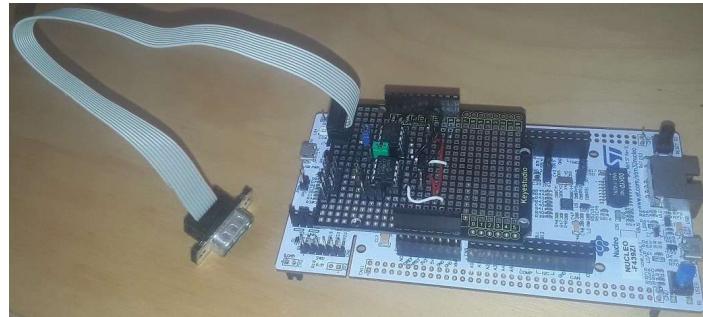
Im Bereich "Generated Function Calls" werden alle durch den Code Generator erzeugbaren Funktionen aufgelistet. Das TM_Engineering.CAN_Stack Modul soll die Aufgaben des CAN1 (oder evtl auch des CAN2) Modules übernehmen. Wir müssen somit die Code Generierung sowie den Funktionsaufruf der MX_CAN1_Init() unterbinden.

Haben wir das getan, können wir den Code Generator noch einmal starten und anschließend das Projekt erfolgreich compilieren.

Es wird Zeit für die Hardware

Wir wollen Software für einen STM32 Mikrocontroller programmieren. Also brauchen wir auch einen Controller welcher unsere Software ausführen soll.

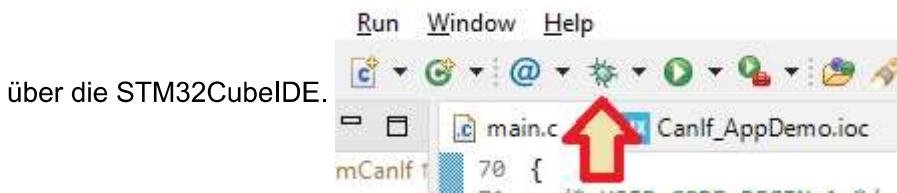
Wie ich zum Start des Tutorials erwähnt habe, möchte ich als Beispiel ein Nulceo-F439ZI Board, welches per Aufsteckboard mit CAN Transciever erweitert wurde, nutzen.



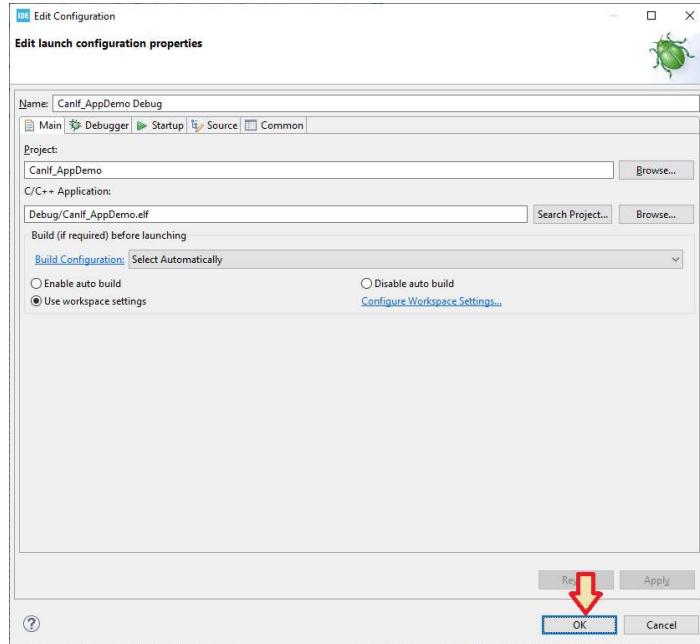
The NUCLEO-F439ZI with CAN

Die Beschreibung des Aufsteckboards soll hier nicht explizit erklärt werden. Bei Interesse kann ich das irgendwann mal ergänzen.

Wir verbinden den auf dem NUCLEO-Board vorhandenen ST-Link mit dem PC und starten den Debug Modus



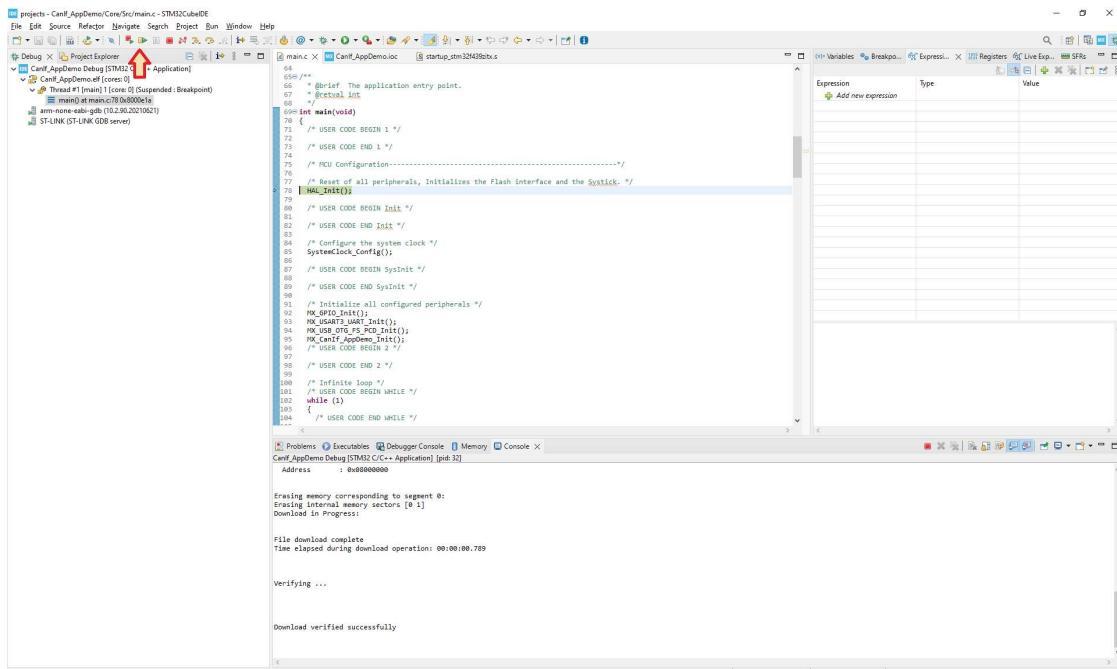
Beim ersten Start des Debuggers müssen wir diesen auch einrichten.



Setup Debugger

Bei unserem aktuellen Tutorialprojekt braucht es keine speziellen Einstellungen. Also können wir mit den vor eingestellten Optionen arbeiten und alles mit OK bestätigen. Nun wird noch die schon bekannte Frage nach zugeordneten "Ansicht, bzw auf Englisch "perspective" gestellt, welche wir auch mit "Switch" bestätigen.

Die Debugger Ansicht wird nach einem kurzen Moment mit einem Standard-Breakpoint in der `main()`-Function zur Ruhe kommen.



Start debugging/testing

Mit einem Klick auf den "Play"-Button wird das bereits in den Controller geladene Programm gestartet.

Wenn alles gut gegangen ist, sollte man jetzt mit Hilfe eines beliebigen CAN-Tools Nachrichten empfangen können.

Time [s.ms]	Frame	ID	DLC	DATA [HEX]	DATA [ASCII]
0.000	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
0.499	STD	7FF	8	01 02 04 08 10 20 04 80
0.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
1.499	STD	7FF	8	01 02 04 08 10 20 04 80
1.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
2.499	STD	7FF	8	01 02 04 08 10 20 04 80
2.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
3.499	STD	7FF	8	01 02 04 08 10 20 04 80
3.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
4.499	STD	7FF	8	01 02 04 08 10 20 04 80
4.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
5.499	STD	7FF	8	01 02 04 08 10 20 04 80
5.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
6.500	STD	7FF	8	01 02 04 08 10 20 04 80
6.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.
7.499	STD	7FF	8	01 02 04 08 10 20 04 80
7.999	STD	1FF	8	S4 4D 43 49 30 32 31 00	TMCI021.

CAN data transmission

Das Tutorialprogramm sendet mit der CAN Id 0x1FF die Versionsinfo des [CanIf](#) Modules.

Note

Today the tutorial has no data receive. I will add this in a upcoming version.

Was ist bei einem "normalen" Öffnen der IDE anders als beim ersten Start

Öffnet man die STM32CubeIDE "normal" (also nicht zum ersten Mal) so werden die zuletzt geöffneten Dateien auch wieder geöffnet angezeigt. Eine Ausnahme stellt die "*.ioc" Datei dar. Diese ist nicht geöffnet. Hierbei muss man im Hinterkopf behalten, dass der Code Generator  nur dann anwählbar ist, wenn die "*.ioc" Datei geöffnet ist. Man kann also auch umgekehrt vermeiden, aus Versehene heraus den Code Generator zu starten in dem man die "*.ioc" Datei schließt.

Das "Information Center" welches uns beim ersten Öffnen der STM32CubeIDE begrüßt hat, lässt sich mit dem



- Button aufrufen.