SensactHAL

# Intro

* Geeignet für STM32F4, F1, F0 und ESP32
* Interface sensacthal mit Implementierungen für die og. Chips
* Funktionalität einer SensactBinary ergibt sich aus
  + dem Basischip 🡪auf jeden Fall eine eigene cpp-Datei
  + dem Board und seiner Version 🡪Parameter des Konstruktors oder eigene cpp-Datei bei starken Veränderungen
  + dem technischen Ausbauzustand des Boards 🡪auf jeden Fall Parameter des Konstruktors
  + den SensactBus mit den GPIOs
  + den am Board angeschlossenen Busteilnehmern
    - für Sensact-Bus-Geräte: automatische Enumeration beim Hochfahren
    - für 1Wire-Geräte: Globales Zuordnungsstruktur [64bit\_1Wire\_ID; ID des verwaltenden Sensact-Gerätes; 8bit-ID des 1Wire-Busses des Sensact-Gerätes; 8bit-Geräteindex am 1Wire-Bus
  + Dem Betriebsmodus, also welche Log-Ausgaben freigeschaltet sind und ob das Gerät „normal“ läuft oder sich in einem Hardware-Selbsttest (zum Test nach dem Aufbau/Löten) befindet.

# Globals

Es gibt

HAL: Hardware-Abstraktion

NODE: Konfiguration der Station mit allen Bussen

MODEL: Applikationslogik

Und sie alle gehören zum SensactContext-Objekt

# Handling mit den Bussen

* Es gibt mehrere lokale Busse

Fraglich ist: Machen wir eine automatische Enumeration oder geben wir vor, welche Devices an den Bussen angeschlossen sein müssen 🡪 Entscheidung: Es wird vorgegeben!

* cBoardHAL: Die Komponenten, die auf dem Board fest montiert sind, müssen über den Konstruktor definiert werden können.
* cNode: Beschreibung des per I2C bzw. OneWire angeschlossenen Umfeldes. Ein OneWire-Gerät ist an einem DS2480 angeschlossen, der im I2C-Bus einen Subbus aufmacht.  
  cStation enthält für JEDES Sensact-Gerät (aktuell 5+ggf. Testgerät) eine per #ifdef abgetrennte Konfiguration

Das cModel.cpp enthält alles das, was

1. Außerhalb des Boards selbst liege