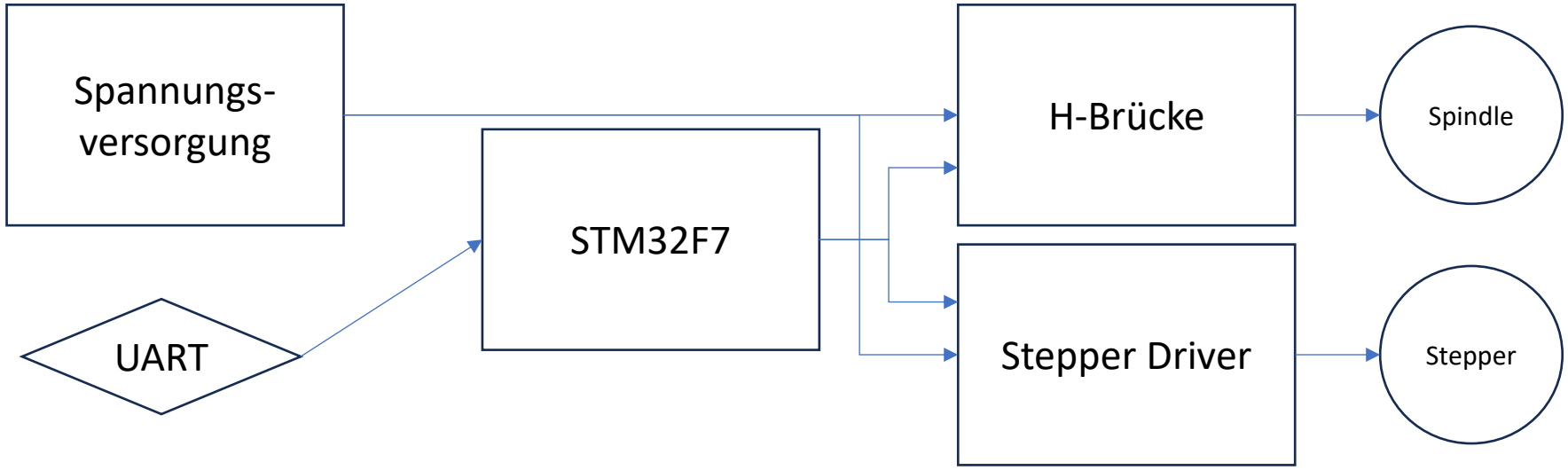
BALLUFF

Einordnung der Aufgabenstellung

Grobe Architektur

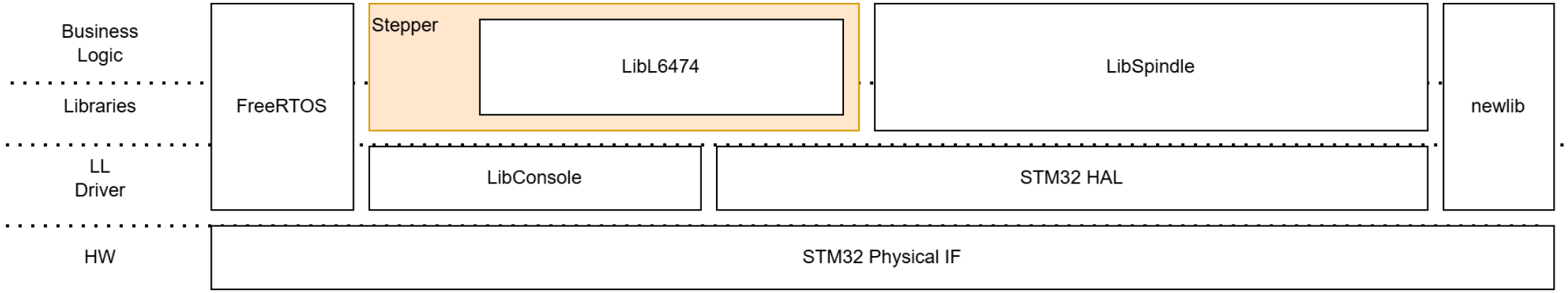
DHBW

Block Diagramm HW



Grobe Einordnung der SW-Architektur

- Der orangene Kasten ist die Primäraufgabe
- Die Libraries müssen eingebunden werden aber sind bereits implementiert



Was wird gestellt, was nicht?

- Alle Bibliotheken werden bereitgestellt
  - LibSpindle
  - LibL6474
  - LibRtosConsole
- Es muss ein Controller für den Stepper integriert werden, der aber LibL6474 einbetten soll, um die Kommunikation zu vereinfachen
- Ein Testaufbau mit Schrittmotor wird pro Gruppe zur Verfügung gestellt, Ein Gesamtaufbau pro Kurs wird ebenfalls für Tests zur Verfügung gestellt.

Sonderfall: HAL autogenerierter Code

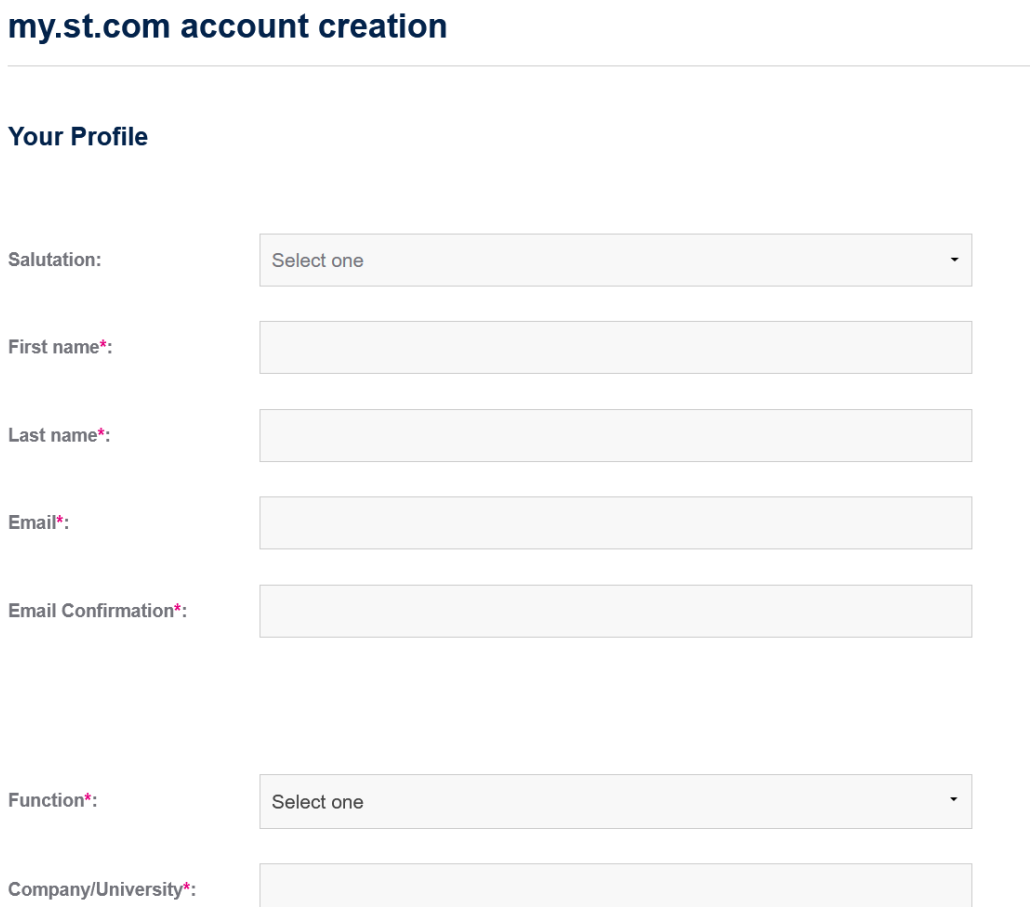
- Möglichst schnell in eigene C- und H-Files mit der Business Logik, da die generierten Files sehr stark im LL-Bereich angesiedelt sind
- Die AutoGen-Mimik war historisch nicht immer stabil und je nach Anwendung gab es Code-Verluste
- Das File wird sehr schnell sehr unübersichtlich
- Eine Austauschbarkeit wird erschwert, wenn alles im main.c der HAL implementiert wird!

Mögliche Lösungsansätze für die PWM-Generierung des Steppers

Grobe Skizzierung der Ansätze und ein paar Hinweise

my.st.com account creation

Link

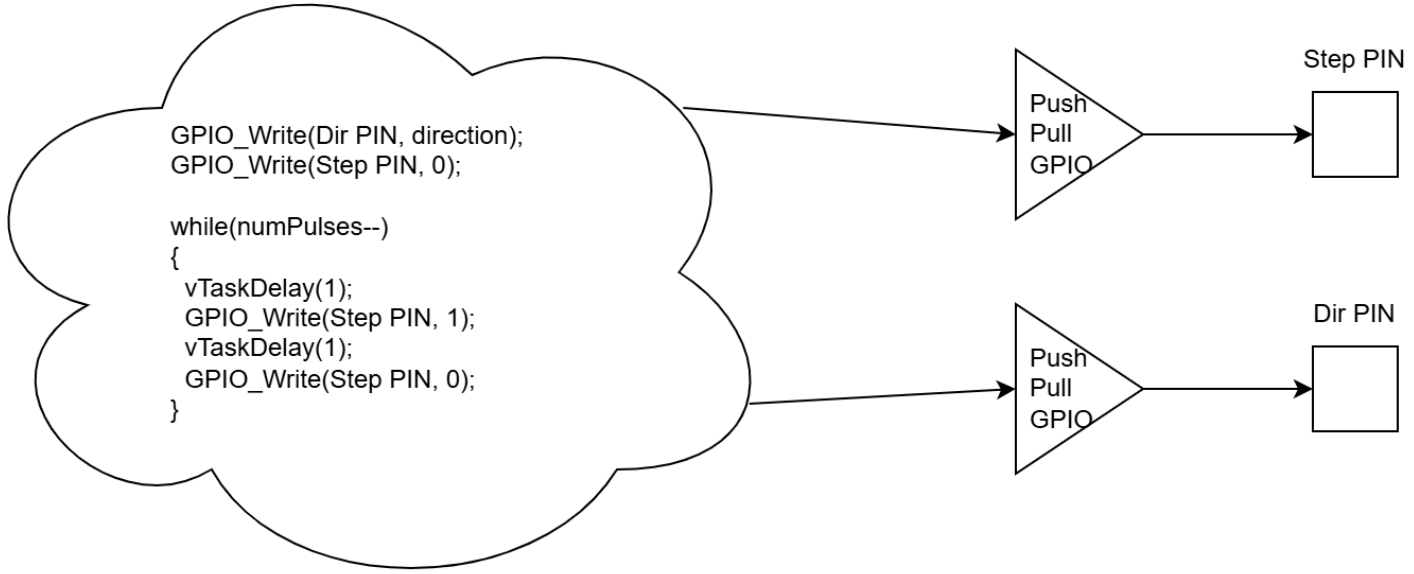


GPIO-Lösung

Easy Going

Direkteste und einfachste Lösung, die keine Asynchronen Vorgänge erlaubt und eine

Geschwindigkeitsregelung unmöglich macht. Reicht für eine Minimalvariante der Bewegung aus



GPIO-Lösung

- Für die Erstinbetriebnahme zu bevorzugen, da synchron und sehr einfach um schnell eine Bewegung zu erzeugen
  - Quasi ein schneller Machbarkeitstest
- Die L6474-Bibliothek muss dafür im Config-Header auf synchron umgestellt werden
  - #define LIBL6474_STEP_ASYNC 0

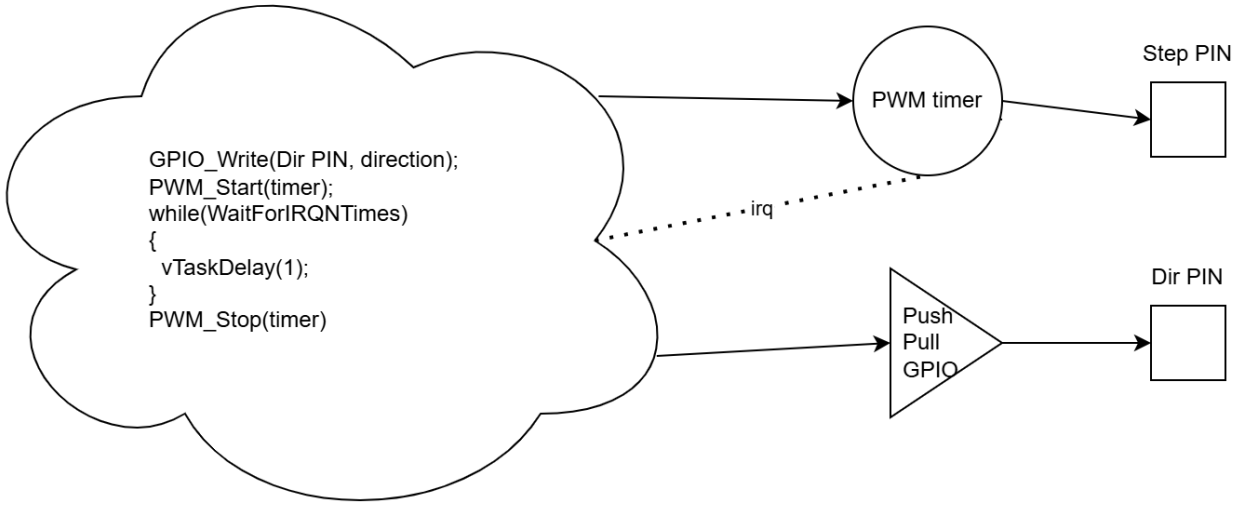
Umsetzung

- Bereits im Ausgangsprojekt bereitgestellt.
  - HAL_GPIO_Write(STEP_PULSE_GPIO_Port, STEP_PULSE_Pin, XYZ);

Single-timer-Lösung

Zwischending mit reiner PWM

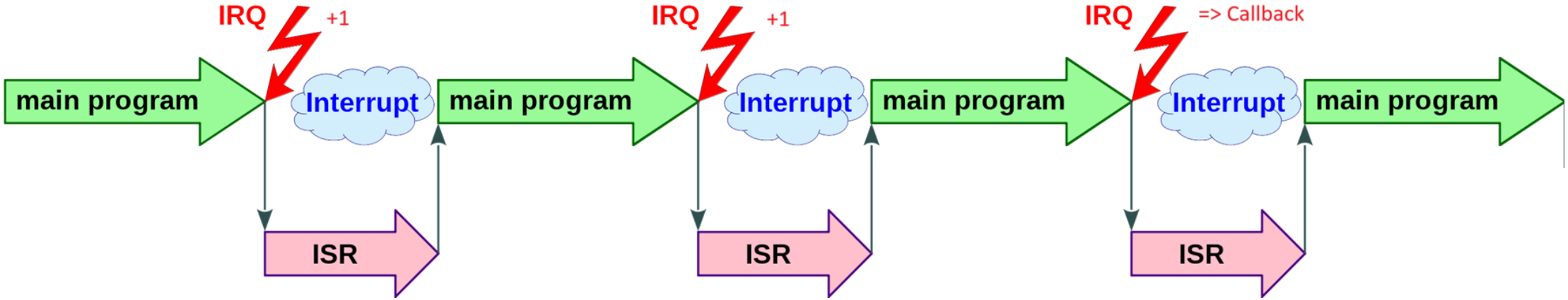
Vertretbare Lösung, die aber etwas aufwendig im Detail sein kann. Erlaubt asynchrone Vorgänge und Geschwindigkeitsregelung aber kann bei der genauen Positionsregelung problematisch werden und ggf. auch je nach Konfiguration einen großen Verwaltungsaufwand mitbringen, um die korrekte Position beizubehalten



Timer-Lösung

- Für die weitere Verwendung zu bevorzugen, da schnelle Pulsfolge und dennoch einfach in der Nutzung
- Es müssen mit jedem Puls IRQs mitgezählt werden, da kein Repetition-Feature des PWM-Timers vorhanden. Vermutlich ist die maximale Geschwindigkeit (IRQ/s) limitiert und könnte nicht für die maximale Geschwindigkeit des Steppers ausreichen
- Die L6474-Bibliothek muss dafür im Config-Header auf asynchron umgestellt werden
  - #define LIBL6474_STEP_ASYNC 1
- Ein Callback der L6474-Bibliothek muss am Ende des Pulsvorgangs aufgerufen werden

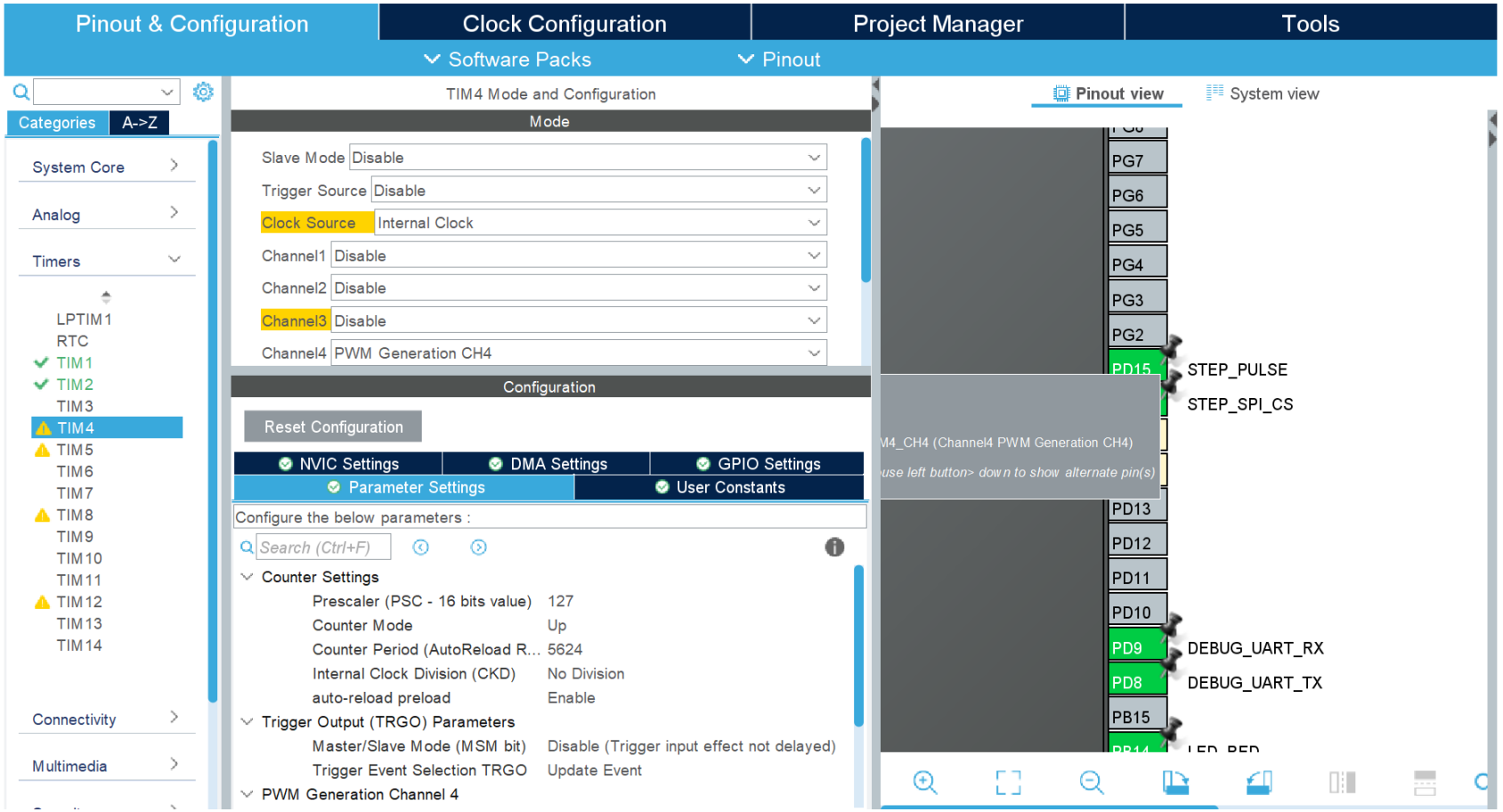
Timer/Callback-Schema



Umsetzung

- Muss mittels des Cube Projekts neu erzeugt werden.
- Der Timer TIM4 muss als PWM-Generator konfiguriert werden und der Interrupt eingeschaltet werden
- Es gibt keinen Repititon Count, somit jeder Puls ein Interrupt
  - HAL_TIM_PeriodElapsed_Callback() wird aufgerufen, wenn der Timer einen Puls erzeugt hat.
  - Wenn genügend Pulse gezählt wurden, kann asynchron oder synchron gegen die L6474 Bibliothek quittiert werden.

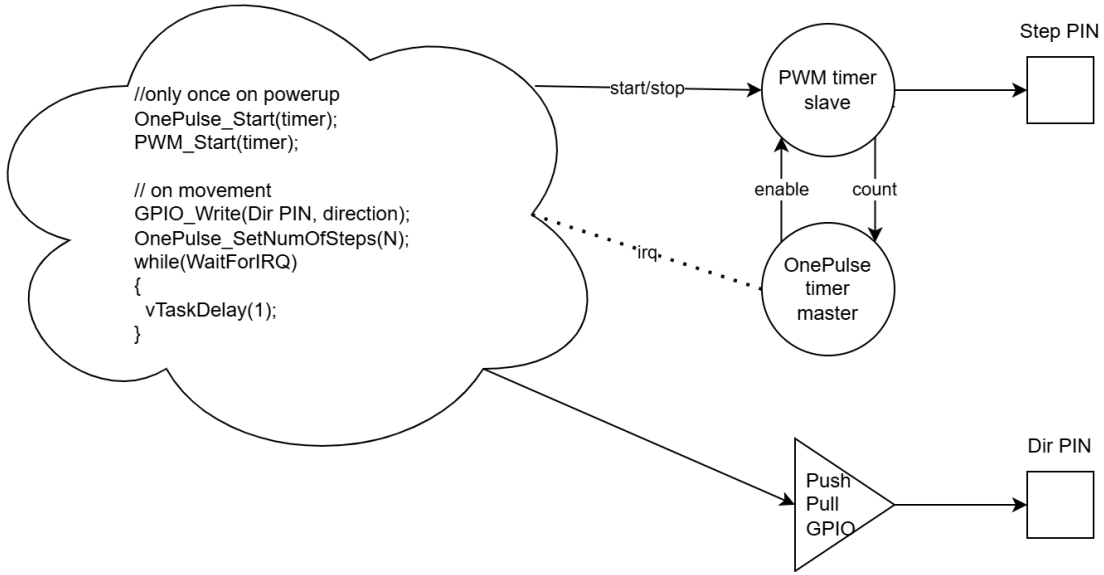
Cube Konfiguration



Master-Slave-Timer-Lösung

Exakte PWM mit automatischer Zählung durch HW

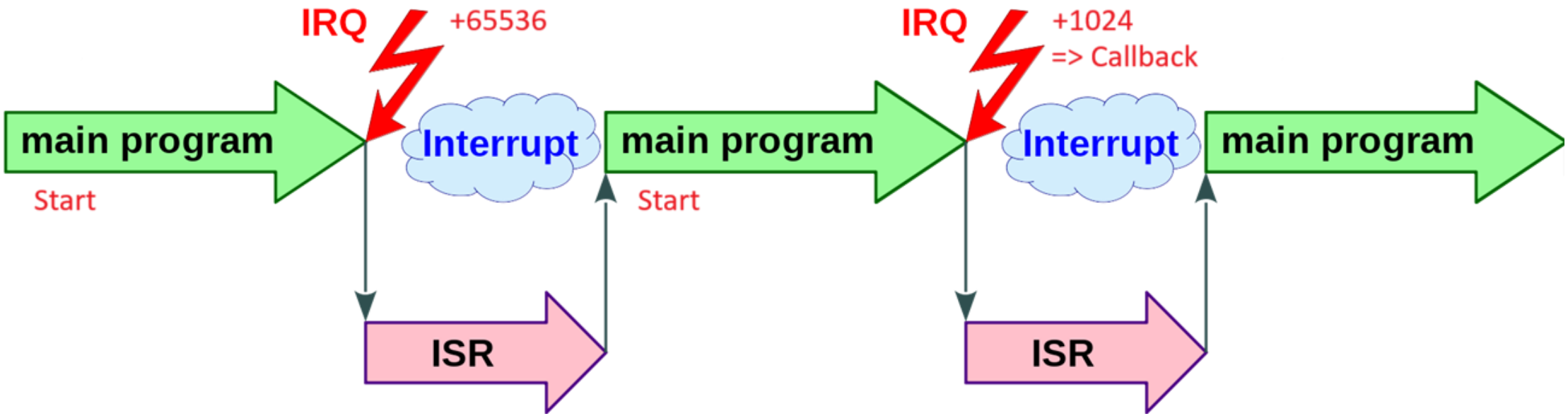
Sicherste Lösung für die exakte Zahl an Pulsen. Zusätzlich eignet sich diese Lösung perfekt für die Geschwindigkeitsregelung, da lediglich der PWM timer geändert wird. Ein Asynchroner Betrieb ist einfach Möglich, da man lediglich auf 2 Interrupt (eigentlich nur einer im besten Fall) reagieren muss. Der OnePulse Timer schaltet den PWM timer exakt nach der gegebenen Zahl an Pulsen ab!



Master-Slave-Timer-Lösung

- Vollausbau, der „komplette“ Pulszählung und Geschwindigkeit der Pulse (Pulsrate) über je einen Timer einstellbar macht.
- Es können maximal 65536 Pulse pro Timer-Start gezählt werden, dann folgt spätestens ein Interrupt und man muss ggf. noch weitere Runden anstoßen.
- Die L6474-Bibliothek muss dafür im Config-Header auf asynchron umgestellt werden
  - #define LIBL6474_STEP_ASYNC 1
- Ein Callback der L6474-Bibliothek muss am Ende des Pulsvorgangs aufgerufen werden

Timer/Callback-Schema



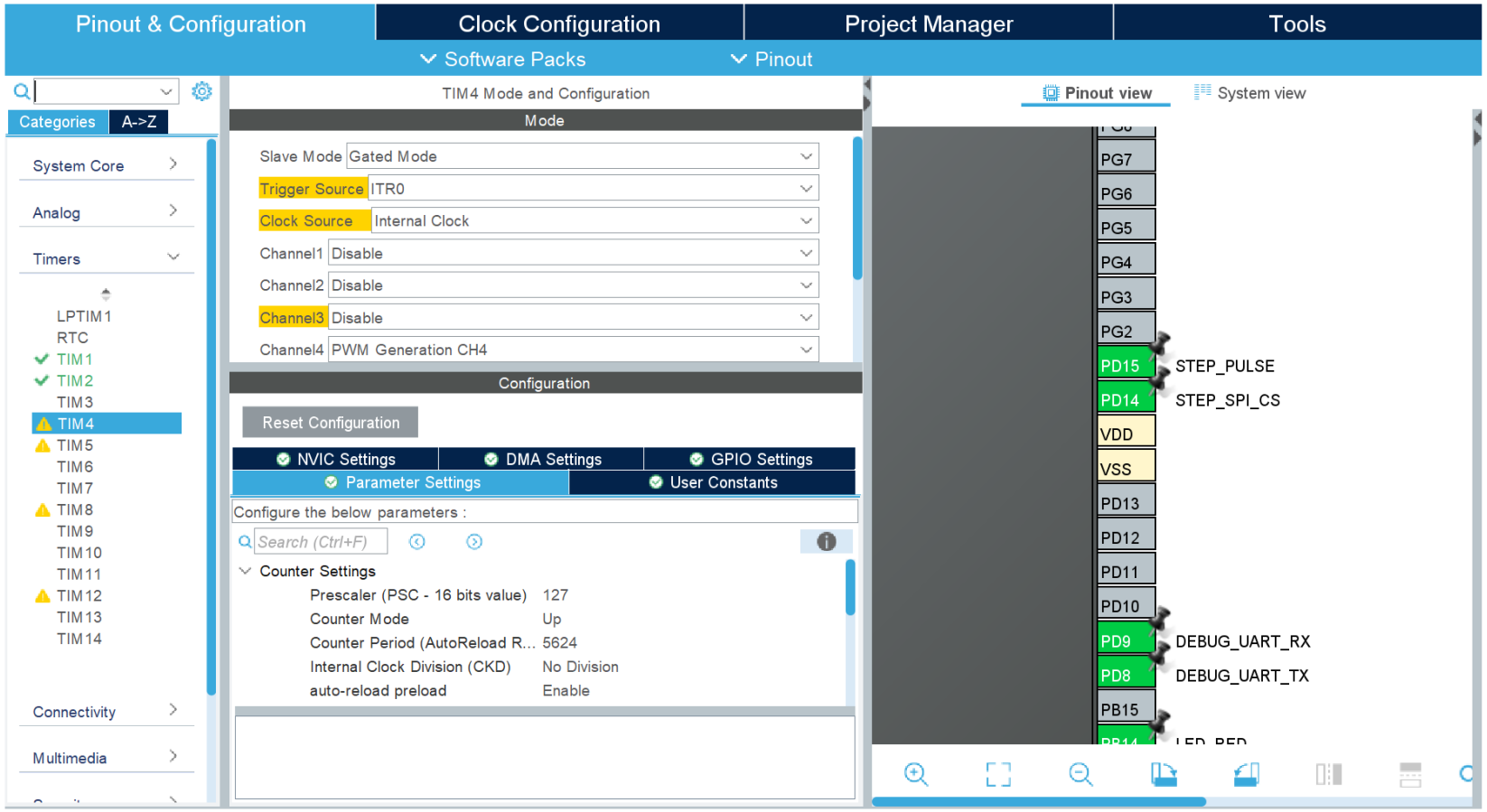
Umsetzung

- Muss mittels des Cube Projekts neu erzeugt werden.
- Der Timer TIM4 muss als PWM-Generator konfiguriert werden. Der TIM1 wird als OnePulse-Mode Timer sowohl als Master als auch als Slave aktiviert und der Interrupt eingeschaltet (capture compare)
- Das ARR-Register des TIM1 gibt die Zahl der Pulse an. Es gibt immer zwei Interrupts
  - HAL_TIM_PWM_PulseFinishedCallback() wird aufgerufen, wenn der Timer den Compare Wert erreicht hat und wenn die Zahl der Pulse erreicht wurden.
  - Wenn genügend Pulse gezählt wurden, kann asynchron oder synchron gegen die L6474 Bibliothek quittiert werden.

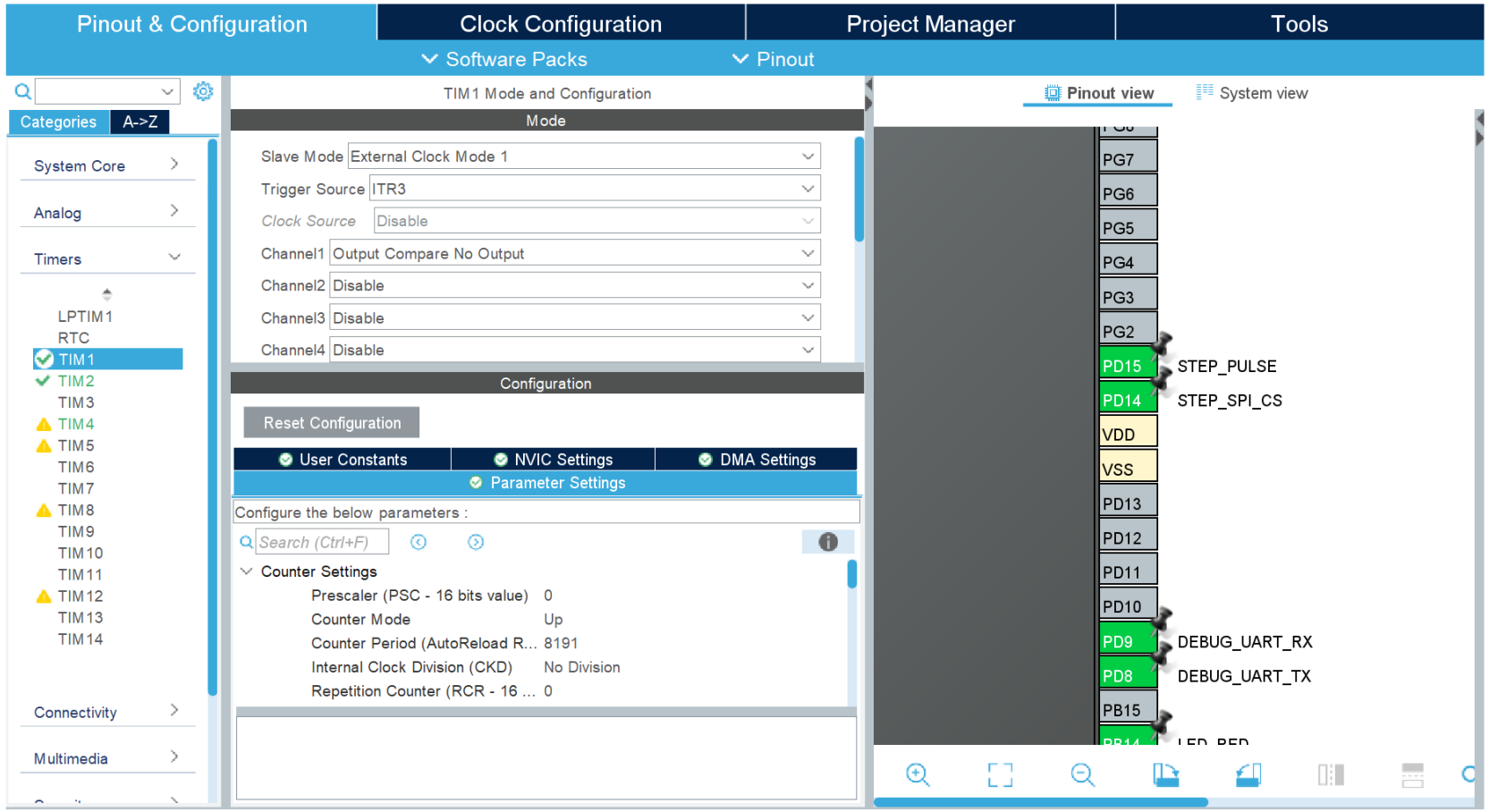
Anmerkung

- Nicht klar ob da ein Bug in der HAL ist, somit den PWM-Timer nie wieder stoppen. Ansonsten gilt folgende Reihenfolge:
  - **HAL_TIM_OnePulse_Stop_IT**(&htim1, TIM_CHANNEL_1);
  - **__HAL_TIM_SET_AUTORELOAD**(&htim1, numPulses);
  - **HAL_TIM_GenerateEvent**(&htim1, TIM_EVENTSOURCE_UPDATE);
  - **HAL_TIM_OnePulse_Start_IT**(&htim1, TIM_CHANNEL_1);
  - **__HAL_TIM_ENABLE**(&htim1);
- Der TIM4 muss vorher gestartet werden und darf nicht mehr gestoppt werden! Er wird im Lauf jeweils vor TIM1 konfiguriert

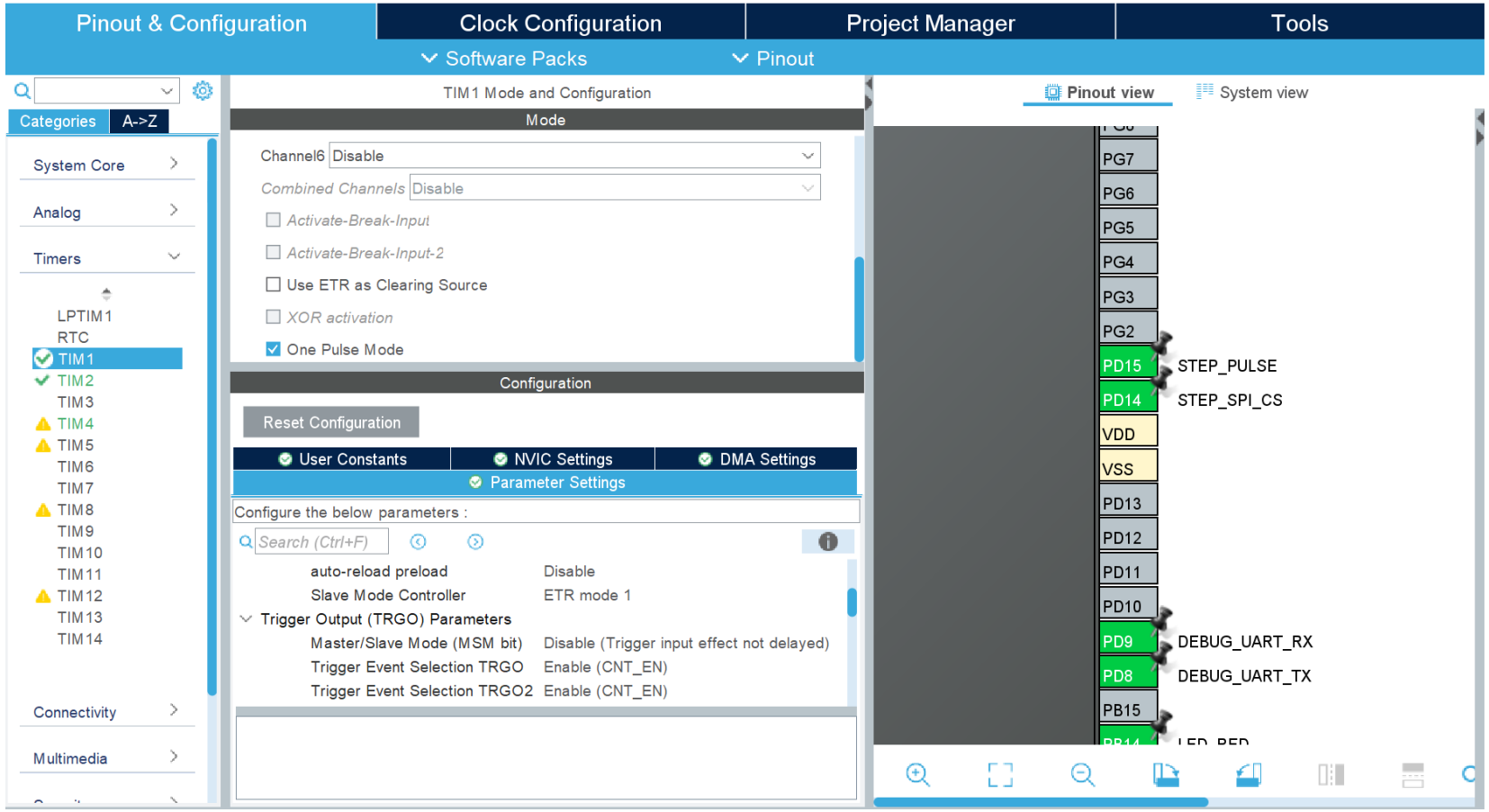
Puls-Timer [CubeIDE]



Counter-Timer [CubeIDE]



Counter-Timer [CubeIDE]



Anmerkung zu der GPIO-Lösung

- Zeitverhalten:
  - Dies ist die einfachste aller Lösungen, allerdings auch die Ungenauste. Sie limitiert die maximale Pulsfrequenz und kann nur synchron zur Ausführung implementiert werden**.
  - Mit dieser Lösung erzielt man lediglich ~500Hz sodass nur ein Bruchteil der maximalen Geschwindigkeit gefahren werden kann!

Anmerkung zu den PWM-Lösungen

- Drehzahlsteuerung:
  - Die Berechnung der Taktfrequenz erfolgt in der Regel mit einem Prescaler- und einem Period-Register. Es gibt somit ein Optimierungsproblem mit zwei Größen, um den geeignetsten Arbeitspunkt zu bestimmen.
- Master-Slave-Timer:
  - Bei der Master-Slave Lösung muss man beachten, dass der OnePulse Timer lediglich 16 Bit Zählen kann, somit ist die Zahl an Pulsen auf 65535 Pulse limitiert. In einem finalen Interrupt muss also geprüft werden, ob damit bereits alle Pulse generiert wurden oder ob man erneut den OnePulse Mode mit den verbleibenden Pulsen startet! Achtung, ein Puls von 1 funktioniert nicht korrekt (mindestens 2 Pulse)

Formeln

- Der Nutzer gibt die Geschwindigkeit v in [mm/min] an
  - $x = \frac{v*s*r}{60*q}$

v = speed [mm/min]
s = steps per turn [ ]
r = resolution [ ] (microstepping 16)
q = mm per turn [mm]

x = steps per second [1/s]

- Der Berechnete Weg wird wie folgt in Pulse umgerechnet
  - $x = \frac{(p_{soll} - p_{ist})*s*r}{q}$

p = position [mm]
s = steps per turn [ ]
r = resolution [ ] (microstepping 16)
q = mm per turn [mm]

x = steps [ ]

  - !! p_{ist} erfordert eine korrekte Umrechnung und kann mithilfe der L6474 Bibliothek zurückgelesen werden (in Schritte)

Weitere Details

- Gehen wir am Platz in Gruppen durch ☺