POLARIS BME DG.lvlib

Alexander Kessler
alexander.kessler@uni-jena.de
Helmholtz-Institut-Jena

Erstellt: 27.06.2012
Letzte Änderung: 05.07.2012

License Agreement for this software

Alexander Kessler
Helmholtz-Institut-Jena
Fröbelsteg 3
07743 Jena
Germany

Contact: alexander.kessler@uni-jena.de

Inhaltsverzeichnis

1 Einleitung

Die Bibliothek *POLARIS BME DG.lvlib* bietet LVOOP Klassen für die BME PCI Delaygeneratoren (i.w. DG). Zu ihrer Entwicklung wurde LabVIEW 2009 verwendet. Sie ist aus Sicht der Bedürfnisse des POLARIS Lasersystems entstanden und kann als Grundlage für die Anpassungen für weitere Anlagen wie ELBE oder PHELIX dienen. Für CS Framework Systeme existierten CS-Framework Klassen, die einen Client Server Betrieb übers Netzwerk ermöglichen.

Dieses Dokument beschreibt die Ideen der Implementierung und ist keine vollständige Dokumentation.

Insbesondere ist die Onlinehilfe zur Bergmann Delaygeneratoren erforderlich, weil dort die Parameter der Delaygeneratoren sehr detailliert beschrieben sind.

Die Kommentare zur Implementierung jeder einzelner Klasse und einzelnen VIs sind in der LV Hilfesystem enthalten, soweit es erforderlich ist.

Momentan werden die Produkte SG02P2, SG05P1, SG05P3 und SG08P2 unterstützt.

Die Klassen können auch mit anderen DG Versionen arbeiten, dafür ist aber ein Test mit der jeweiligen Hardware (i.w. HW) erforderlich.

Für die Verständnis der Implementierung sind folgende Kenntnisse vom Vorteil:

- Objektorientiertes Programmieren (Vererbung und Funktionsaufruf über Dynamic Dispatch Mechanismus)
- speziell LVOOP (http://www.ni.com/white-paper/3574/en)
- LabVIEW Reference Pattern (Beispiel in der LV Onlinehilfe: *ReferenceObject.lvproj*)
- Bergmann *DelayGenerator.dll* Funktionen, Übergabe von Parametern (Onlinehilfe zur Bergmann Applikation)

Versionsverwaltung und Zusammenarbeit

Für die Versionsverwaltung wird GIT verwendet. Ein bequemer GIT Client für Windows ist TortoiseGIT. Die Bibliothek wird über

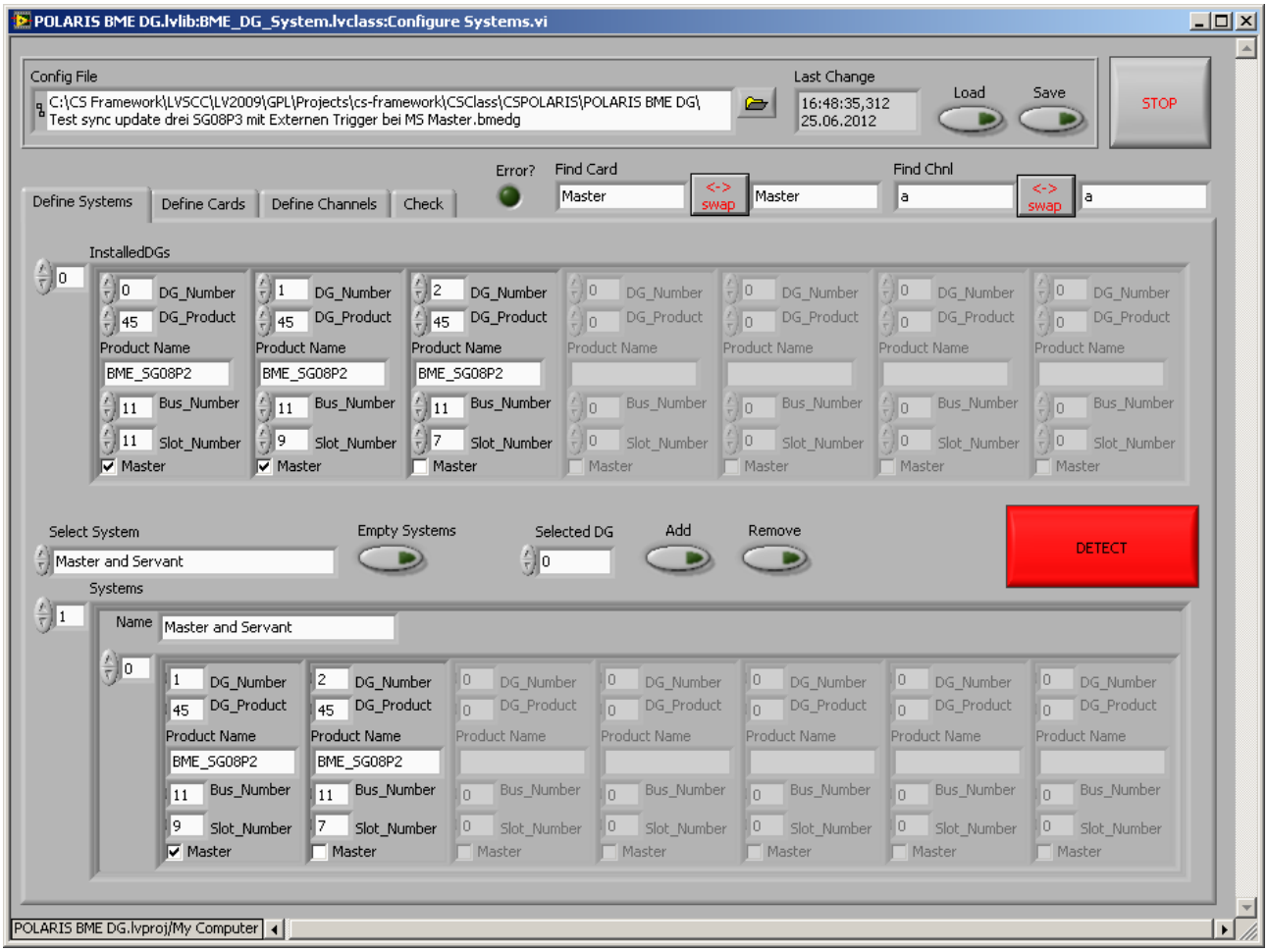
https://github.com/keale/POLARIS-BME-DG.git

veröffentlicht.

Für Rückmeldungen, Testberichte, konstruktive Kritik wird erbetten.

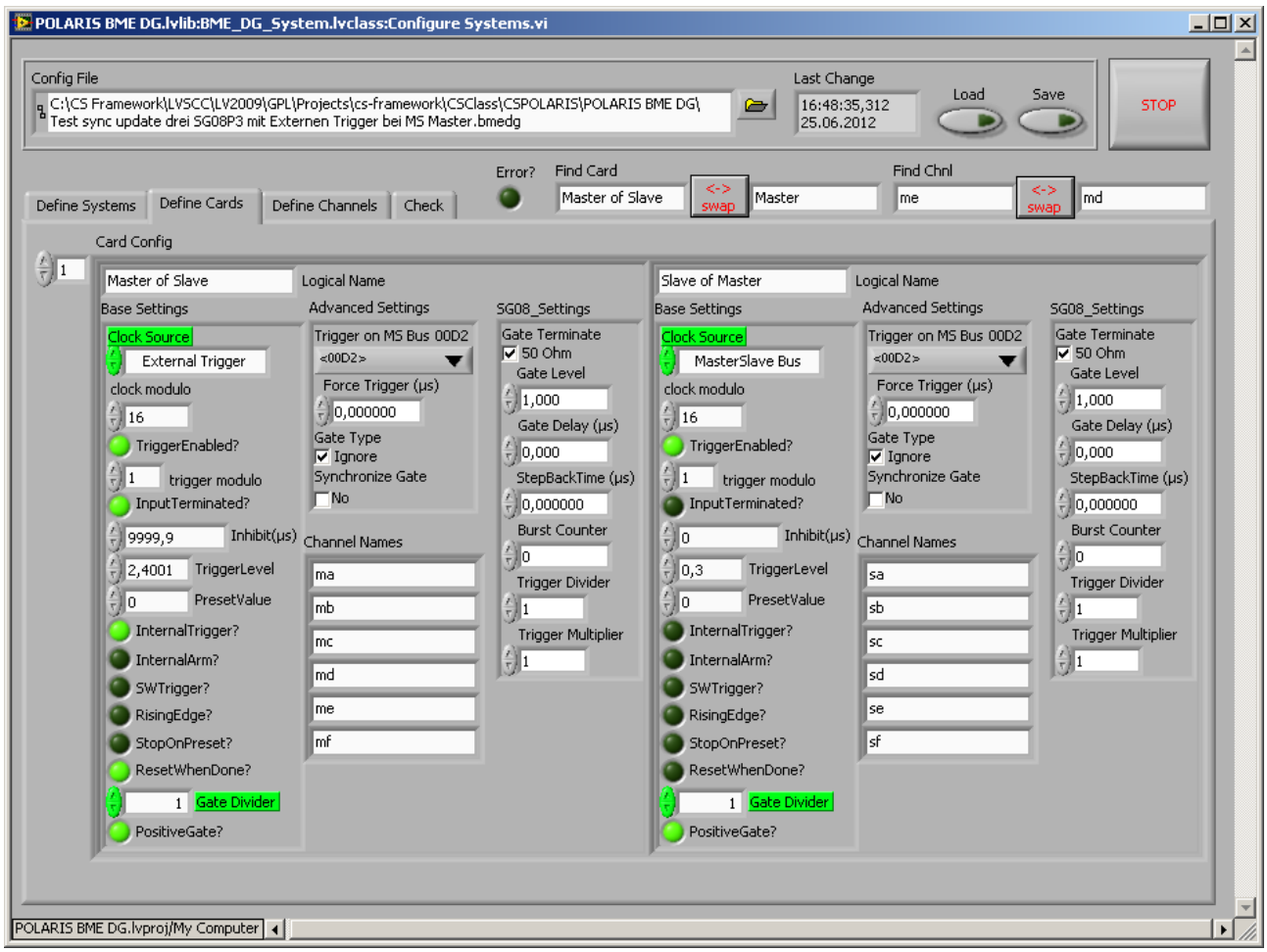
2 Erste Schritte

2.1 Erstellen einer Konfigurationsdatei



Zuerst müssen Sie eine Konfigurationsdatei für ihr System erstellen. Dafür führen Sie die *BME_DG_System:ConfigureSystems.vi* aus. Klicken Sie in der Lasche "Define Systems" auf "DETECT".

ACHTUNG, dabei werden evtl. laufende DGs gestoppt. Stellen Sie sicher, dass ihre Anlage dadurch keinen Schaden nimmt. Anschließend können bei "Systems" ein neues System definieren, indem Sie in Feld "Name" ihr System benennen. Dann können Sie einen Master DG sowie seine Slaves entsprechend Ihrer tatsächlicher HW Konfiguration mittels "Selected DG" und "Add" dem System hinzufügen. Haben Sie mehrere Master-Slave Konfigurationen in Ihrem Computer, so wiederholen Sie diese Schritte.



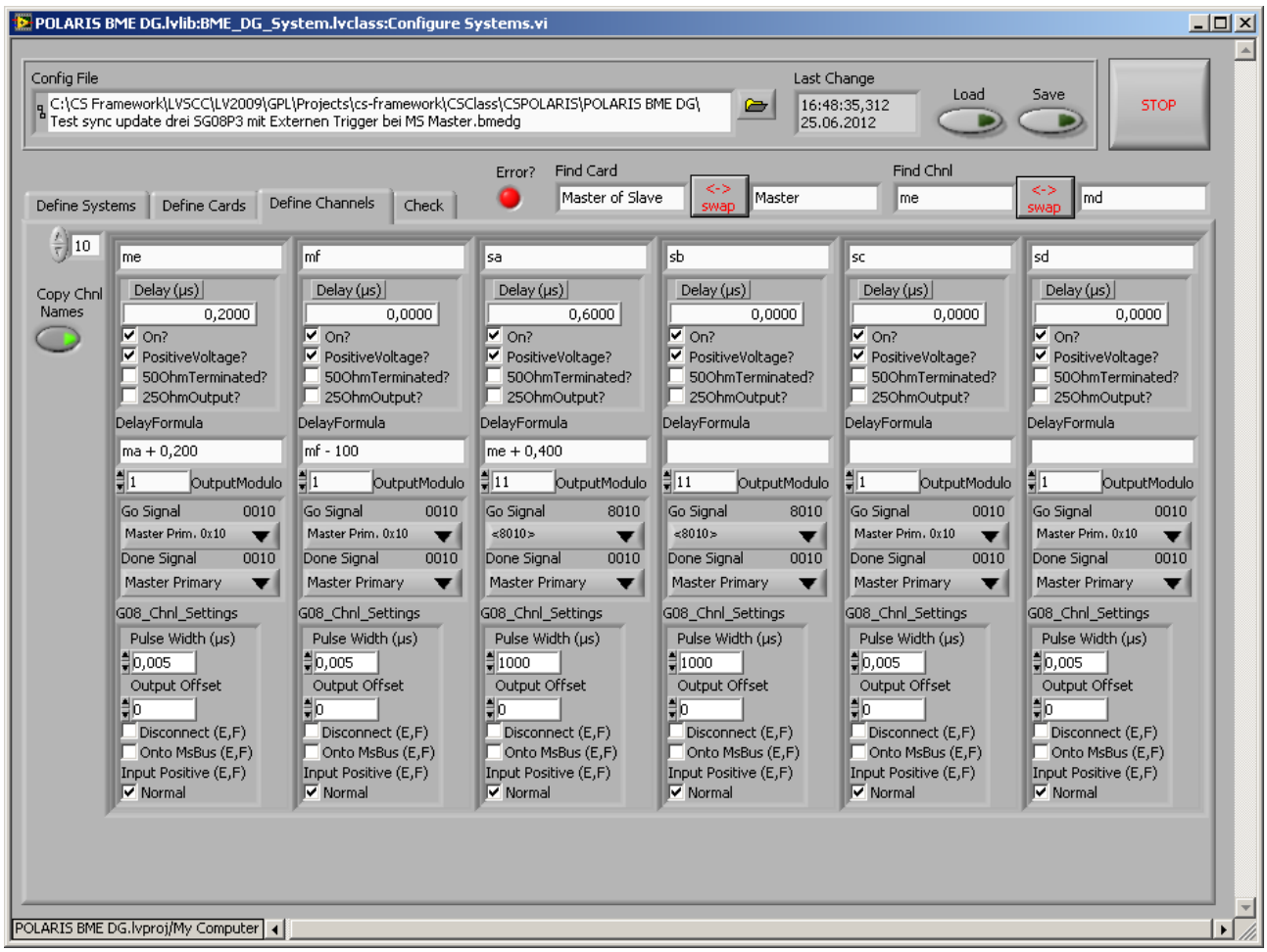
In der Lasche "Define Cards" können Sie den Trigger, Clock, Gate und Master-Slave Bus der DG Karten konfigurieren. Wie bei Systems können Sie eine Karte definieren indem Sie die Karte benennen. Bei Advaced Settings -> Trigger on MS Bus können mehrere Bits gesetzt sein. In der Lasche "Check" können Sie den für ihre Konfiguration richtigen Wert der Maske bequem ermitteln.

Zum Schluss definieren Sie die logischen Namen für die Kartenkanäle im Feld "Channel Names". Die Position in diesem Array entspricht dem physikalischen Kanal, also T, A, B bei DGs bis BME_DG_SG08 bzw. A,B,C,D,E,F bei BME_DG_SG08.

WICHTIG: Der Arrayindex (0...N) entspricht der DG-Nr. der Karte. Diese Nummer ist die physikalische Adresse der Karte und dient als Karten ID für die DLL Funktionen.

ACHTUNG: Die Bergmann-Applikation liefert in dem Dialog Define->Configure->Delay Generator List die Karten NICHT nach ihrer DG-Nr. sortiert.

Sollten Sie sich bei zwei Karten mit der Position im Array vertan haben, so können Sie mit "<->Swap" die Karten miteinander vertauschen. Sie können die Elemente im Array mittels Kontextmenüs bearbeiten.



In der dritten Lasche "Define Channels" konfigurieren Sie die einzelnen Kanäle. Sie können die Namen der Kanäle mittels "Copy Chnl Names" aus der Kartenkonfiguration übernehmen.

Hier ist die Position der Kanäle im Array willkürlich, weil die Zuordnung zum physikalischen Kanal über den logischen Namen erfolgt. Der logische Name muss aber bei der richtigen Karte (Arrayindex = DG Nummer) und auf der richtigen Position in "Channel Names" stehen.

Die Bitmaske für "Go Signal" können Sie in der Lasche "Check" ermitteln. Über „Go Signal“ stellen Sie die Kanalparameter ein, u.a. dass der Kanal bei der **synchronen Parameternübergabe berücksichtigt werden soll. Setzen Sie dafür das Bit „SyncReload=0x8000“.**

Die absoluten Delaywerte werden im Feld "Delay(µs)" angezeigt. Sie können nicht direkt verändert werden, sondern werden aus der Delayformel errechnet. Eine Delayformel hat eine der Formen:

- *logischer Name des Bezugskanals ± relativer Delaywert*
- *logischer Name des Bezugskanals*
- *absoluter Delaywert*

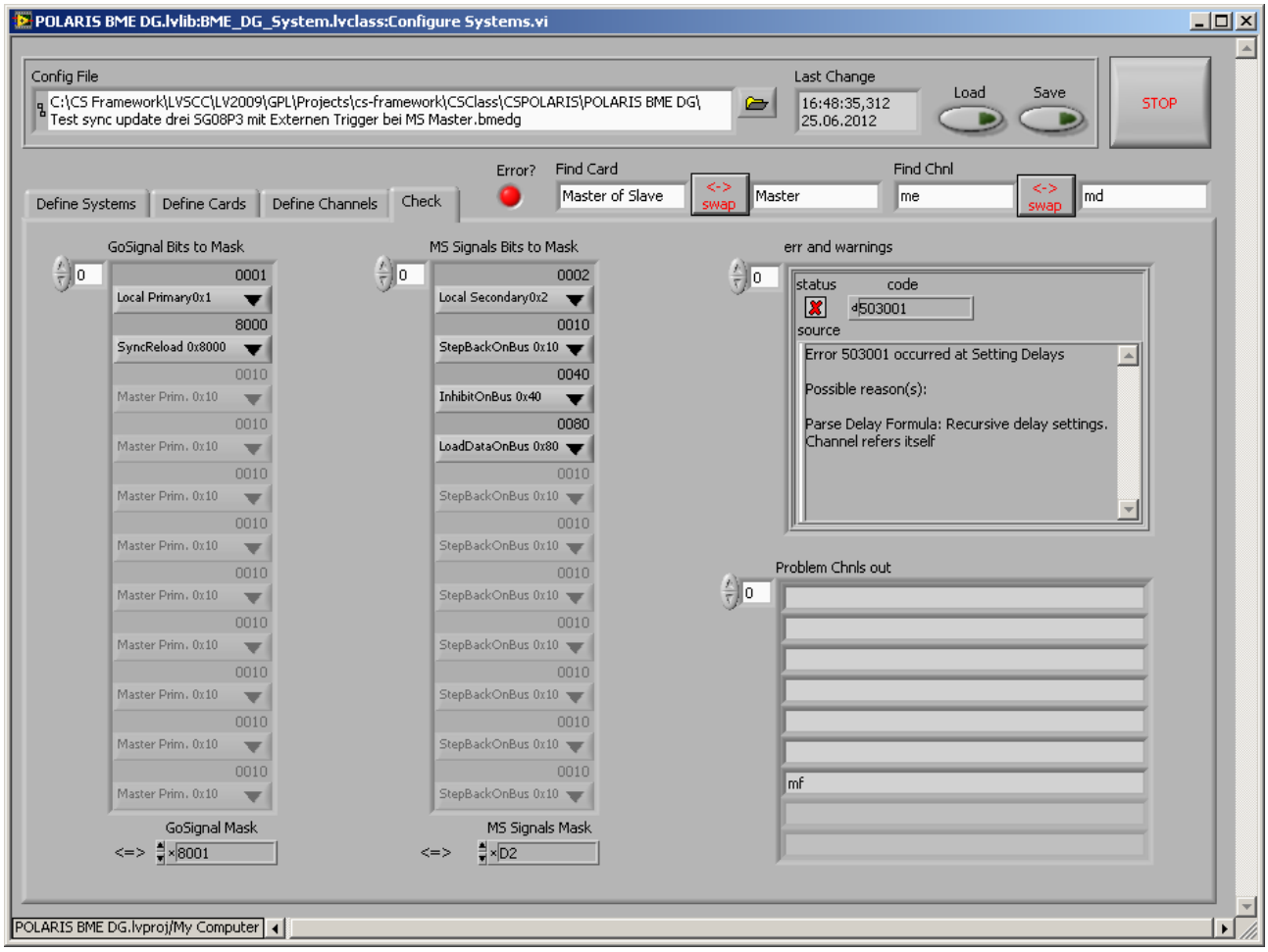
Bei jeder Änderung eines Werts im Kanal oder Kartenarray erfolgt eine Konsistenzüberprüfung. Schlägt sie fehl, so leuchtet der Indikator "Error?". Es kann passieren, dass die Delayformulas trotz richtiger Einträge nicht aufgelöst werden können, weil die Reihenfolge der Kanäle im Array ungünstig ist, z.B. *Kanal A = Kanal C + 100,* der Kanal C kommt im Array nach dem Kanal A. In diesem Fall kann die Position dieser Kanäle über "<->Swap" vertauscht werden.

Die Lasche "Check" bietet außer bequemer Berechnung der Bitmasken für GoSignal und MSBus

Signale auch Fehlerinformationen, falls ihre Konfiguration nicht konsistent ist. Das ist z.Z. der Fall, wenn

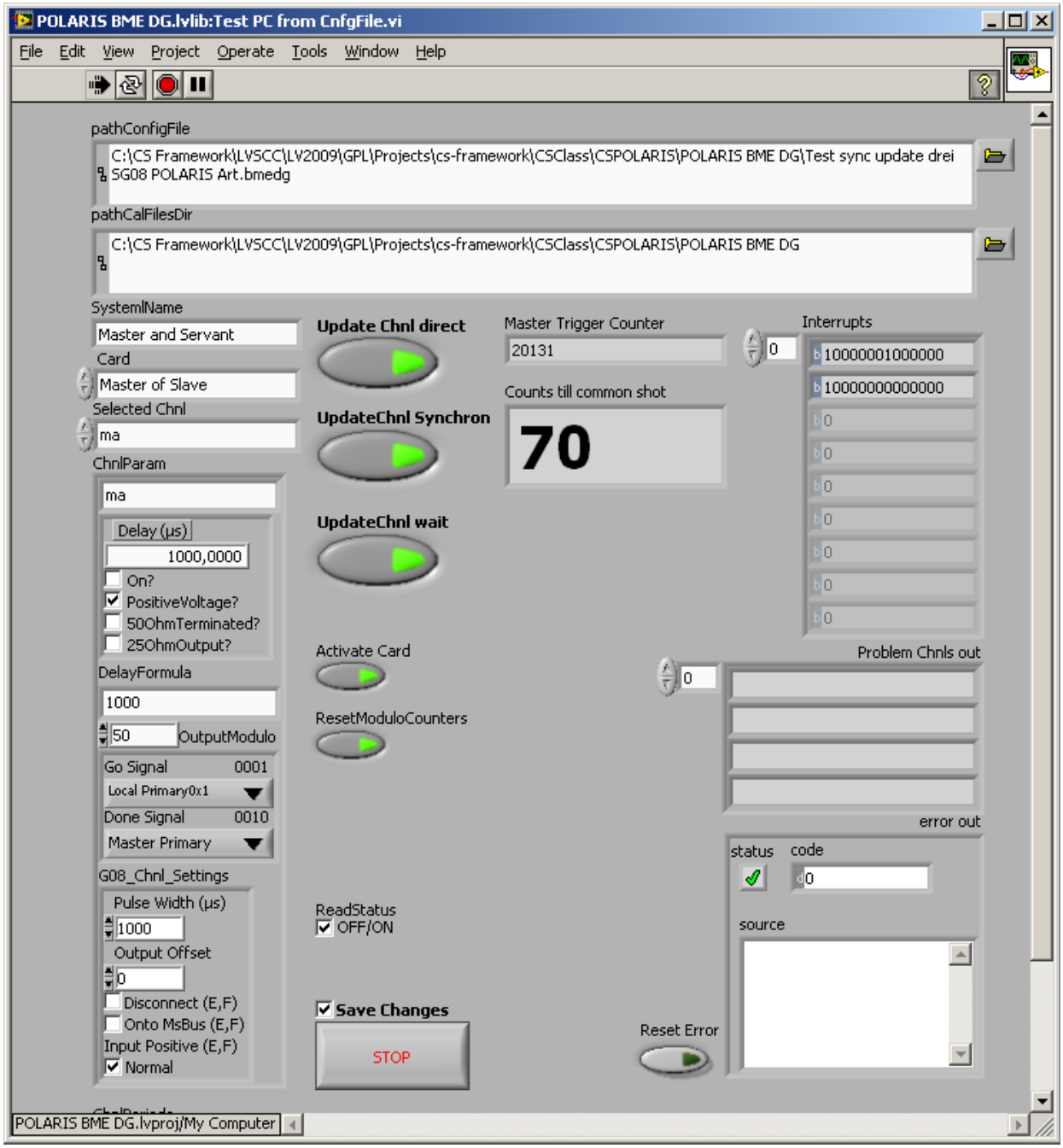
- die Delayformulas rekursive Abhängigkeiten aufweisen.
- Delayformulas negative absolute Delays bei mindestens einem Kanal verursachen
- Ein logischer Kanalname in der Kartenkonfiguration nicht vorhanden ist.

Bitte speichern Sie die Konfiguration in einer Datei mittels "Save". Die Konfigurationsdateien sollen die Endung .bmedg haben. Sie basieren auf auf LabVIEW XML Schema.



2.2 Test ihrer Konfiguration

Öffnen Sie *Test PC from CnfgFile.vi.* Wählen Sie ihre Konfigurationsdatei in "pathConfigFile" aus und geben Sie das Verzeichnis im dem sich die Kalibrierungsdateien *Module.cal* und *BME_G0x.cal* befinden in "pathCalFileDir". Alle in der Konfigurationsdatei angegebene Systeme werden aktiviert. Allerdings können nur die Kanalparameter eines einzelnen Systems aus diesem Test VI verändert werden. Geben Sie den Namen des gewünschten Systems in "SystemName" ein. Führen sie das VI aus und vergewissern sich, dass die einzelnen Kanäle sich nach Ihren Vorstellungen verhalten.



2.2.1 Asynchrone Änderung der Kanalparameter mit DeAktivieren der Karten

Das erfolgt mit dem Button "Update Chnl Direct". Dabei wird zuerst die Masterkarte und dann die Slaveskarten deaktiviert, falls sie aktiv sind. Dann werden die neuen Parameter in die Karte geschrieben, Modulo Counter zurückgesetzt und die Karten wieder aktiviert, falls sie vor der Deaktivierung aktiv waren, zuerst die Slavekarten, dann die Masterkarte.

2.2.2 Änderung der Parameter ohne DeAktivieren der Karten

Beim Klick auf "UpdateChnl Synchron" werden die Paramer in die Karten geschrieben, ohne dass

die Karten deaktiviert und wieder aktiviert werden. Dabei kann es ab BME_DG_SG08P3 zu zwei Szenarien kommen:

Synchrone Parameterübergabe:

Ab BME_DG_SG08P2 ist es möglich, die Parameter der Kanäle synchron zu verändern, wenn OutputModuloCounter aller entprechend parametrierter Kanäle an der **MASTERKARTE** den Zustand Null erreicht hat. Dabei muss das Bit für Signal "LoadDataOnBus=0x80" bei MS-Bus Maske in der Kartenkonfiguration gesetzt sein. Der MS-Buses muss bei allen Karten im System identisch konfiguriert sein. Bei den Kanälen, die an der synchronen Parameterübergabe beteiligt sein sollen, muss bei der Maske für "Go Signal" das Bit "SyncReload = 0x8000" gesetzt sein.

Asynchrone Parameterübergabe:

Ist der DG älter als SG08P2 oder ist die Konfiguration nicht so wie oben beschrieben, dann werden die Parameter asynchron von der Karte übernommen. In diesem Fall ist ein besonders intensiver Test aller möglicher Bedienfällen notwendig. Insbesondere kann es bei einer Änderung der Output Modulo Counters zur nicht konsistenten Zuständen an den Kanälen kommen.

2.2.3 Änderung der Parameter mit Deaktivierung und Reaktivierung der Karten und Abwarten eines gemeinsamen Ausgangssignals.

Bei einigen DG Konfiguration (BME_DG_SG05P3) wird Output Modulo Counter von der Bergmann Applikation zurückgesetzt, sobald man in Dialog Define->Delays auf "Übenehmen" klickt, falls an einem beliebigen Kanäle der Wert des Output Modulos verändert wird. Dadurch wird sofort ein Signal an allen Kanälen ausgelöst. Wenn eine Anlage auf Einhaltung der Wartezeiten angewiesen ist, kommt es in ungünstigen Fall zu Schäden.

Um dieses Verhalten zu umgehen, wurde folgende Lösung entwickelt:
Falls die Kanäle mit einer langen Periode (>1sec) feuern als die Pollingperiode der HW Statusabfrage (PC spezifisch, ≈50ms), ermittelt das Programm die Zeit bis zum nächsten gemeinsamen Signal an allen Kanälen. Kurz davor wird die Änderung übernommen. Dieses Szenario wird durch das Button "Update Chnl wait" ausgelöst.

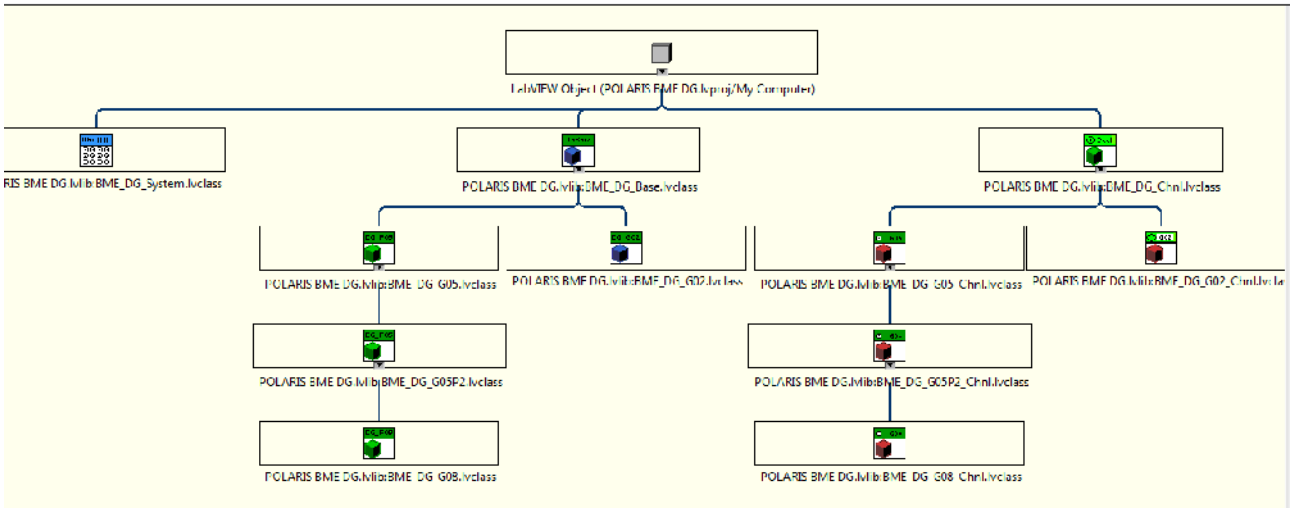
3 Implementierung

Die Delaygeneratoren werden durch LVOOP Objekte repräsentiert. Die Klassen VIs kann man in zwei Gruppen einteilen. Die erste Gruppe bilden VIs für den Zugriff auf die Objekt Membervariablen. Sie machen keine HW Modifikationen. Bevor man auf die HW zugreift, muss man die Objekte richtig initialisieren und mit Parameter versorgen.

Die zweite Gruppe sind die VIs für den HW Zugriff. Es wird die **Namenskonvention** verwendet, dass diese VIs den Suffix **HW** haben, z.B. *BME_DG_Base:ConfigureHW.vi*. Grundsätzlich ändern alle VIs entsprechende Parameter in den Objekten bevor eine HW Modifikation erfolgt, so dass normaler Weise SW Repräsentation dem tatsächlichen HW Zustand entspricht.

ACHTUNG: z.Z. haben gibt es keine Möglichkeit den realen HW Zustand auszulesen. Es wird davon ausgegangen, das die Parameter der Objekte dem HW Zustand entsprechen.

3.1 Klassenhierarchie



Eine HW Kombination aus mindestens einer Masterkarte und evtl. mehreren Slavekarten wird durch ein Objekt der Klasse *BME_DG_System* repräsentiert. Die Klasse besitzt ein Array aus *BME_DG_Base* Objekten. Beachte, dass somit auch jeder Objekt der von *BME_DG_Base* abgeleiteten Klassen enthalten werden kann. In der BME_DG_System VIs wird auf die VIs der Basisklasse zugegriffen. Zur Laufzeit werden aber über das Dynamic Dispatch Verfahren die VIs der jeweiligen Kindklasse aufgerufen. Diese berücksichtigen die spezifischen Eingenschaften der jeweiligen DG Typs.

Das gleiche passiert mit den Kanälen. Die Klasse *BME_DG_Base* hat ein Array mit Objekten der Klasse *BME_DG_Chnl*. Somit kann jede Karte auch Objekte der abgeleiteten Kanalklassen enthalten. Natürlich soll ein Objekt der Klasse *BME_DG_G08* entsprechend Kanalobjekte der Klasse *BME_DG_G08_Chnl* enthalten. Das wird in *Initialize.vi* sichergestellt. Die Kindklassen überschreiben das *BME_DG_Base:Initialize.vi.* Das *BME_DG_G08:Initialize.vi* erzeugt eben die Objekte der Klasse *BME_DG_G08_Chnl* und speichert sie in der Variable *objChnls,* die von der Basisklasse geerbt wurde

Im weiteren folgt die Beschreibung einzelner Klassen.

3.2 BME_DG_System

Die tatsächlich installierten DGs werden durch die Klasse BME_DG_System repräsentiert. Sie bietet VIs für die wichtigsten **Anwendungsfälle**:

- Detektieren der PCI DGs an mehreren Busen *(DetectMultiBusPCIDGsHW.vi).*
- Erzeugen einer Konfigurationsdatei (*ConfigureSystems.vi*)
- Konfigurieren und Kalibrieren der DG HW anhand der Konfiguraitonsdatei (*InitDGSystems from CnfgFile HW.vi*)
- Finden der Kanal- und Kartenobjekte anhand des logischen Namens (*Get DG by LogicalName.vi* und *Get Chnl by LogicalName.vi*)
- Das gleiche im Fall von mehreren Systemen *(Select System By Chnl.vi, Select System By Name.vi)*
- Aktivieren der Karten in der Reihenfolge Slaves und Masters *(Activate AllHW.vi)* oder nur Masters *(Activate MastersHW.vi)* bzw. Slaves (*Activate MastersHW.vi*)
- Enstprechend das Deakivieren

- Aktualisieren der Kanalparameter synchron ab BME_DG_SG08P2, *Update Chnls SyncHW.vi*)
- und asynchron mit Deaktivieren und wieder Aktivieren aller Karten *(Update Chnls with DeActivateHW.vi)*
- Berechnen Anzahl der Trigger bis zum nächsten gemeinsamen Trigger an allen Kanälen *(CountsTillNextCommonTrigger.vi)*
- Speichern der während des Betriebs veränderter Kanalparameter in der Konfiguraionsdatei *(Write All Chnl Param2CnfgFile.vi)*
- Einlesen der Delayformeln und Überprüfung der Konsistenz der Delaywerten *(Check Chnl Settings.vi,)*

Ein System besteht aus **genau einer** Masterkarte und **0 bis mehreren** Slavekarten. Es ist möglich, mehrere Systemobjekte parallel laufen zu lassen.

Bei der Initialisierung wird jeder Karte und jedem Kanal ein logischer Name vergeben. Der weitere Zugriff auf die einzelnen Karten und Kanäle erfolgt über diesen Namen.

3.2.1 Delay Managment

Wie in Bergmann Applikation kann man die Delays für Kanäle relativ zu einander in Form :

Delay Kanal B = log Name Kanal A +- rel.Delay(µs)

definieren. Wird das Delay vom Kanal A verändert, so wird auch das absolute Delay für Kanal B und aller davon abhängigen Kanäle neu berechnet in HW aktualisiert. Das BME_DG_System überprüft vor der Änderung eines Delays, ob die davon abhängigen Kanäle negative absolute Delaywerte bekommen würden. Ist es der Fall, so wird die Änderung verworfen.

Ferner werden rekursive Delayformeln z.B. Kanal A hängt vom Kanal B ab, C von B, A von C abgefangen.

Wenn eine neue Delayformel bei einem Kanal gesetzt werden soll, wird das VI *BME_DG_System:Check new DelayFormula and Update DelaysHW.vi* aufgerufen. Der gewünschte Kanal wird ermittelt und *BME_DG_Chnl:Parse DelayFormula.vi* wird aufgerufen.

BME_DG_Chnl:Parse DelayFormula.vi

1. liest das absolute Delaywert des referenzierten Kanals
2. berechnet das neue absolute Delaywert für das eigene Kanal
3. Wenn der neue Wert >= 0, wird zuerst eine Überprüfung der abhängigen Kanäle durchgeführt, ob eins davon negatives Delay bekommen würde, oder ob in den Delayformel rekursive Abhängigkeiten auftreten. Dafür wird *BME_DG_System:Check Delay Dependencies.vi* aufgerufen. Es arbeitet mit einer Kopie des Systemobjekts, weil die Delays in den Objekten verändert und im Fall eines Fehlers nicht zurückgesetzt werden.

   BME_DG_System:Check Delay Dependencies.vi:

   3.1. Erhöht den Rekursionsindikator *BME_DG_System:DeadloopBlocker* um eins.
   3.2. Ruft für jeden Kanal im System *BME_DG_Chnl:Check Delay Dependencies.vi* auf. Es
      a) überprüft ob das eigene Kanal von dem gefragten Kanal abhängt. Falls nicht, ist die Überprüfung für das eigene Kanal zu ende
      b) andernfalls ruft für das eigene Kanal *BME_DG_System:Check Delay Dependencies.vi* auf. Es kommt also zu einem rekursiven Aufruf ab Punkt 3.1

4. Zurück zu *BME_DG_Chnl:Parse DelayFormula.vi:*
   Ist die Überprüfung ohne Fehler verlaufen, werden die Delayformel und der absolute Delaywert im Kanal gespeichert.
5. das VI *BME_DG_System:Update Delays.vi* wird mit dem eigenen Kanal als Parameter aufgerufen, um die absoluten Delaywerte der vom eigenen Kanal abhängigen Kanäle zu aktualisieren.
   BME_DG_System:Update Delays.vi
   5.1. ruft für jedes Kanal *BME_DG_Chnl:Update Delays.vi* auf.
   BME_DG_Chnl:Update Delays.vi
   a) überprüft ob das eigene Kanal vom gefragten Kanal abhängt. Ist es der Fall, so wird der absolute Delaywert neuberechnet und
   b) *BME_DG_System:Update Delays.vi* wird für das eigene Kanal von *BME_DG_Chnl:Update Delays.vi* aufgerufen.

3.3 BME_DG_Base

Das ist die Basisklasse für alle Delaygeneratoren Klassen. Diese Klassen verwalten Clock, Trigger und Gate Einstellungen der DGs. Die Verwaltung der Kanalparameter wird an die Kanalklassen delegiert.

BME_DG_Base definiert eine gemeinsame Schnittstelle und bietet für alle Delaygeneratoren gemeinsamen Funktionen und Parameter an. Das sind:

- ProductID:long	Bergmann laufende Produktnummer
- ProductName:string	Bergmann DG Typ Bezeichnung
- LogicalName:string	Ihr Name für das DG
- Master?:bool	True, wenn DG als Master konfiguriert ist
- Active?:bool	True, wenn DG ist aktiviert
- objChnls[]: BME_DG_Chnl	Enthält Objekte der von BME_DG_Chnl abgeleiteten Klassen. Das Array wird von der *Initialize.vi* initialisiert
- BaseTriggerSettings: BME_DG_TriggerSettings.ctl	Struktur mit den grundlegendsten Clock, Trigger und Gate Parametern.
- TriggerCounter:64bitLong	Zähler des DG Triggers
- Interrupts:ulong	Enthält Interruptbits
- ClockEnable:bool	DG Systemuhr ist aktiviert.
- PulseOutputLevel:enum	TTL, NIM oder ECL – Signaltyp
- PulseWidth:	Pulsbreite in Systemuhrperioden

BME_DG_TriggerSettings.ctl

- ClockSource:enum	Interner Oszillator, Externer Trigger, Externer Trigger und im Fall seines Ausfall interner Oszillator, Trigger vom MS-Bus
- clock modulo:long	Jeder N-te Puls vom Internen Oszillator wird genommen. Die Frequenz wird auf diese Weise heruntergeteilt.

- TriggerEnabled	DG Triggerschaltkreis ist eingeschaltet
- TriggerModulo	Jeder N-te Puls am Triggereingang wird verarbeitet
- Inhibit(µs)	Sperrzeit nach einem Triggersignal, während der keine weiteren Trigger verarbeitet werden. Entspricht der **internen Periode**
- Trigger Level:double	von -2.4 bis +2.5Volt
- PresetValue:long	nach N-Triggern wird DG erfolgen keine weitere Trigger mehr
- Internal Trigger?:bool	
- Internal Arm?:bool	Siehe Bergmann Online Hilfe
- SWTrigger?:bool	
- RisingEdge?:bool	
- ResetWhenDone?:bool	
- StopOnPreset?:bool	Falls PresetValue aktiv sein soll
- ResetWhenDone?:bool	Muss True sein, damit mehr als ein Triggersignal verarbeitet wird.

Weiter folgt die Beschreibung einiger VIs. Für die meisten oben aufgelisteten Attribute existiert ein öffentliches Schreib und Lese VI. Diese VIs modifizieren nur die Objektparameter, sie werden unten nicht aufgelistet. Man muss dafür sorgen, dass die Änderung am Objekt mit entsprechenden HW Zugrifffunktionen in die Karten übertragen wird.

Ein Anwenderprogramm kann das DG-Objekt vom BMD_DG_System erhalten und verändern. Weil einige VIs andere DGs in System beeinflussen könnten, ist der Zugriff auf sie auf "Community“ eingeschränkt. Access Scope Community entspricht „for freands only“ in C++. Zur "Community" gehört BME_DG_System Klassse, weil in dieser Klasse das Zusammenspiel aller Kanäle koordiniert wird.

(- privite, # protected, + public, ~ community)

# SetTriggerHW	Konfiguriert Trigger, ruft DLL *Set_TriggerParameters()* auf. In den Kindklassen werden zusätzlich DLL *Set_V2_TriggerParameters()* und *Set_G08_TriggerParameters()* falls die DGs entsprechende Parameter benötigen.
# SetClockHW	Konfiguriert die Uhr.
# SetChnlHW	Ruft für alle Kanäle *ModifyChnlHW* der Kanalklasse auf. Je nach Kanalklasse werden DLL Funktionen *Set_G02_Delay(), Set_G05_Delay(), Set_G05P2_Delay()* oder *Set_G08_Delay()* aufgerufen. Das VI ist geschützt, damit vor der Parameterübergabe evt. die Karte deaktiviert und danach aktiviert wird.
~ ConfigureHW	Realisiert mit oberen drei VIs ein sogenanntes "Channeling Pattern" (s. https://decibel.ni.com/content/docs/DOC-13725), d.h. zur Laufzeit des Programms werden über Dynamic Dispatch SetTriggerHW, SetClockHW und SetChnlHW der Kindklassen aufgerufen.
~ UpdateChnlDelaysHW	Ruft zur Zeit nur SetChnlHW auf, war ebenfalls als "Channeling Pattern" gedacht. Falls man z.B. bei bestimmten Operationen ein Deaktivieren und wiederaktivieren der Karte braucht, kann es dieser VI hinzugefügt werden.

+ Init Obj From Cnfg File.vi	Initialisiert ein Objekt mit den Werten aus der Konfigurationsdatei. Bekommt Cluster vom Typ *Card Config.ctl* als Eingangsparameter. Das Cluster ist an BME_DG_SG08P2 angelehnt. Jede DG Klasse holt ihre Teilmenge der Parameter und ignoriert den Rest.
+ Read Trigger Counter HW	Liefert Triggerzähler des DGs
~ Common Trigger	Anzahl der Trigger bis alle Kanäle einen gemeinsamen Signal erzeugen.

3.3.1 BME_DG_G02

Die Klasse erbt von *BME_DG_Base*. Die Klasse überschreibt *Initliaze.vi*. Das VI erzeugt drei Objekte der Klasse *BME_DG_G02_Chnl* und speichert sie in *objChnls.*

3.3.2 BME_DG_G05

Die Klasse erbt von BME_DG_Base. Zusätzlich kommen folgende Klassenvariablen mit entsprechenden Zugriff VIs hinzu: *V2 Trigger Setting* vom Typ *BME_DG_V2TriggerSettings.ctl*.

Die Klasse überschreibt *Init Obj From Cnfg File.vi* und *Initliaze.vi*. Das *Initliaze.vi* erzeugt drei Objekte der Klasse *BME_DG_G05_Chnl* und speichert sie in *objChnls.*

3.3.3 BME_DG_G05P2

Die Klasse erbt von BME_DG_G05 und überschreibt *Initliaze.vi*. Das *Initliaze.vi* erzeugt drei Objekte der Klasse *BME_DG_G05P2_Chnl* und speichert sie in *objChnls.*

3.3.4 BME_DG_G08

Die Klasse erbt von BME_DG_G05P2. Zusätzlich kommen folgende Klassenvariablen mit entsprechenden Zugriff VIs hinzu: *SG08_Settings* vom gleichnamigen Typ.

Die Klasse überschreibt *Init Obj From Cnfg File.vi* und *Initliaze.vi*. Das *Initliaze.vi* erzeugt sechs Objekte der Klasse *BME_DG_G08_Chnl* und speichert sie in *objChnls.*

3.4 BME_DG_Chnl

Das ist die Basisklasse für alle Delaygenerator Kanälklassen.

BME_DG_Chnl definiert eine gemeinsame Schnittstelle und bietet für alle Kanäle gemeinsame Funktionen und Parameter an. Das sind:

-ChnlParameter:ChnlSettings.ctl	Grundeinstellungen für Kanal
-ChnlNr:long	Kanalnummer, die für DLL Funktionen Set_Gxx_Delay erforderlich ist
-Name:string	logischer Name
-DGNr:long	DGNr der Karte
-TriggerTillShot:long	Anzahl der Triggerereignisse, bis dieses Kanal ein Signal ausgibt
-LastTrigCounterUpdate:	Wann TriggerTillShot zuletzt aktualisiert wurde

Timestamp	
-DelayFormula : string	Delayformel in einer der Formen: • logischer Name des Referenzkanals ± rel. Delaywert • logischer Name des Referenzkanals • absoluter Delaywert
- DIMServiceID:long	Für DIM Protokoll Service ID in CS-Framework Kontext.
- InterruptBit:bool	Bildet das Bit des Interruptstatusregister fürs Kanal.
-InerruptUpdatePeriode:double	Wie schnell das InterruptBit aktualisiert wird
-LastInterruptSet:Timestamp	Wann das InterruptBit gesetzt wurde
-LastInterruptReset:Timestamp	Wann das InterruptBit zurückgesetzt wurde
-ShotPeriod:double	Schuss Periode des Kanals

VIs: *(- privite, # protected, + public, ~ community)*

~Delay Managment VIs	Siehe Erläuterungen zu BME_DG_System
~ Update and Acknowledge InterruptBit HW.vi	Aktualisiert InterrurptBit des Kanals und bestätigt das Interrupt im Delaygenerator
~ActivateChnlHW.vi	Ruft DLL Set_Gxx_Delay mit Bool Active = TRUE Achtung: Asynchrone Parameteränderung
~DeactivateChnlHW.vi	Ruft DLL Set_Gxx_Delay mit Bool Active = FALSE Achtung: Asynchrone Parameteränderung
+ModifyChnlHW.vi	Wird in den Kindklassen überschrieben, ruft die zum Kanaltyp passende DLL Set_Gxx_Delay auf.

3.4.1 BME_DG_G02_Chnl

Die Klasse erbt von BME_DG_Chnl. Die Klasse überschreibt *ModifyChnlHW.vi*

3.4.2 BME_DG_G05_Chnl

Die Klasse erbt von BME_DG_Chnl. Neue Klassenvariable:

- OutputModulo.

Die Klasse überschreibt *ModifyChnlHW.vi*

3.4.3 BME_DG_G05P2_Chnl

Die Klasse erbt von BME_DG_G05_Chnl. Zusätzliche Klassenvariablen:

- Go Signal
- Done Signal.

Die Klasse überschreibt:

- *ModifyChnlHW.vi*
- *Read All ChnParameter.vi.*

- *Write Chnl from CnfgFile.vi*

3.4.4 BME_DG_G08_Chnl

Die Klasse erbt von BME_DG_G05P2_Chnl. Neue Klassenvariablen:

- G08_Chnl_Settings

Die Klasse überschreibt

- *ModifyChnlHW.vi*
- *Read All ChnParameter.vi.*
- *Write Chnl from CnfgFile.vi*

4 Hinzufügen weiterer Delaygenerator Typen

- Untersuche anhand der Bergmann Hilfe, ob die Karte spezielle Funktionen bietet, die nicht durch bereits vorhandene Klassen abgedeckt werden
- Falls es der Falls ist, implementiere eine neue Klasse für Kanäle, falls bei den Kanälen neue Funktionen dazugekommen sind oder DG Klasse falls Clock, Trigger oder Gate neue Funktionen abietet.
- in jedem Fall muss man in der *BME_DG_System:ObjectFabric ProduktID2Object.vi* entsprechende Zuordnung zwischen der Produkt ID und einer passenden DG Klasse herstellen.

5 Probleme

- Das synchrone Update erfolgt dann, wenn die Modulo Counter der "SyncLoad"-Kanäle der **Masterkarte** Null erreicht haben. Die ModuloCounter der Slavekarten spielen keine Rolle.
- Ein "SyncLoad"-Kanal der Masterkarte muss aktiv sein, sonst wird er nicht berücksichtigt.
- Auch Parameter der **nicht "SyncLoad"-Kanäle** werden synchron übernommen. Das hat zur Konsequenz, das auch schnell laufende Kanäle (1Hz) mit der langsamstem Kanal z.B. (1/80 Hz). Das macht im Fall von POLARIS Probleme beim Anpassen der Umlaufzeiten der ersten beiden regenerativen Verstärkern.
- Triggercounter der Slavekarten bleibt bei Null, wenn es auf "Extern" gesetzt ist.
- DLL *ReadInteruptState(...)* liefert die Kanalbits nicht zuverlässig. Sporadisch wird das Interruptbit nicht ausgelesen, obwohl Kanal gefeuert hat.
- DLL *ResetModuloCounter(...)* hat bei SG08P2 keine Wirkung.