

Abb. 8.15: Cluster zur Dekodierung eines Bits des erweiterten Systemstatus

Um nun jedes Bit zu dekodieren, wird ein Array aus den oben eingeführten Clustern erstellt. Die ersten neun Elemente dieses Arrays werden mit den in Tab. 8.4 gegebenen Werten besetzt. Diese Werte sind die Standardwerte der ersten neun Arrayelemente. Das Bitmuster wird über eine LV Funktion in ein Array von Wahrheitswerten umgewandelt. Dieses Array wird nun an eine For-Schleife angeschlossen und indiziert. Das bedeutet, dass die Schleife für jedes einzelne Bit des Bitmusters ein Mal durchlaufen wird; bei einem 32Bit-Integer also 32 Mal. Für jedes Bit aus den Nutzdaten, wird eine Überprüfung anhand des aus dem Array indizierten Clusters vorgenommen. Für jedes Bit der beiden Achsen wird eine lesbare Meldung in einen mehrzeiligen String ausgegeben. Zudem wird ein Wahrheitswert „kritischer Status“ ausgegeben, der wahr ist, sobald eines der als kritisch markierten Bits gesetzt ist [siehe Abb. 8.16].

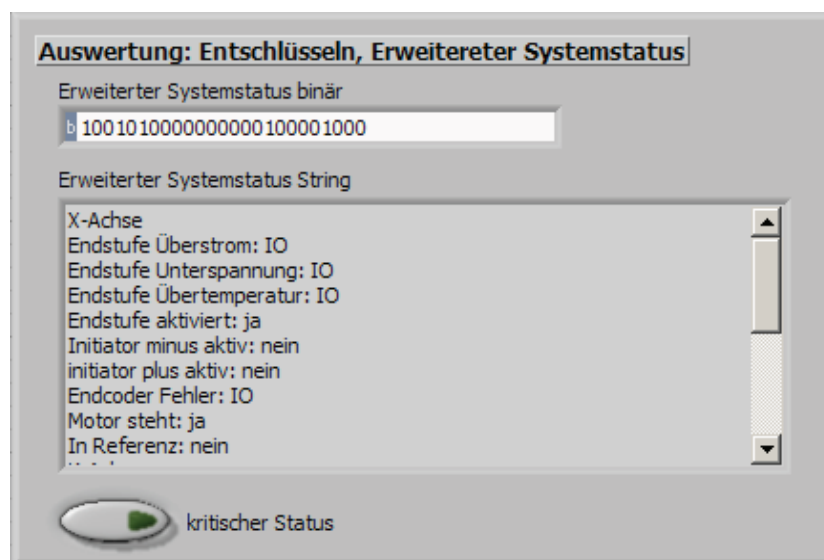


Abb. 8.16: Textdarstellung des Erweiterten Systemstatus

8.3.3 Steuerung der Messtechnik

Zur Steuerung der Messtechnik dient die LV-Bibliothek Pulse.lvlib. Die Bibliothek besteht aus insgesamt 9 VI's davon sind drei Sub-VI's

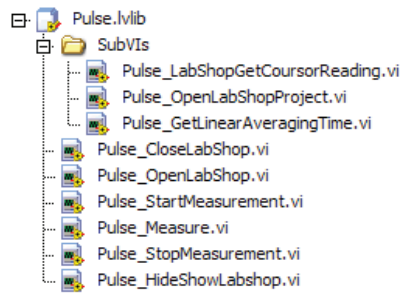


Abb. 8.17: Ordnerstruktur der Bibliothek Pulse.lvlib

Der Name der Bibliothek weist darauf hin, dass es hier um die Steuerung des im Labor vorhandenen Pulse-Systems geht. Da es vom Hersteller B&K nicht vorgesehen ist, die Hardware direkt zu steuern, wird ausschließlich der Pulse LabShop zur Steuerung der Messtechnik verwendet. Damit beschränkt sich die Steuerung der Messtechnik im Grunde auf das Steuern des Pulse LabShop's (Im Folgenden wird die Software Pulse LabShop als „der LabShop“ bezeichnet).

Die in Kapitel 4 aufgeführten Module der Messtechnik werden vom LabShop zur Messung der Schallintensität verwendet. Dazu wird im LabShop ein Projekt erstellt. In diesem Projekt ist die für die Messung der Schallintensität verwendete Hardware konfiguriert. Das Erstellen eines solchen LabShop-Projekts war Thema mehrerer voriger Diplomarbeiten. Das hier eingesetzte Projekt (default.pls) dient als Vorlage für jegliche Art von Schallintensitätsmessungen. Zur Messung an einem bestimmten Messobjekt müssen gegebenenfalls Änderungen am LabShop-Projekt vorgenommen werden. Die eigentliche Aufgabe der Steuerung der Messtechnik ist jedoch unabhängig vom geladenen LabShop-Projekt.

Die Bibliothek bietet die folgenden Funktionen zur Steuerung des LabShop's:

- Öffnen und Schließen des Programms Pulse LabShop
- Laden eines LabShop-Projekts
- Anzeigen und Verbergen des LabShop Fensters
- Durchführen eines Messvorganges (Starten und Stoppen einer Messung und Einlesen des gewünschten Messwerts in LV)

Das Steuern des LabShop's geschieht über das „OLE2.0 Interface“ des Pulse LabShop's. Dieses Interface basiert auf der proprietären COM-Technologie (Component Object Model) von Microsoft [Lit. 9]. Durch COM ist es unter Windows möglich, verschiedene Software-Programme miteinander kommunizieren zu lassen. COM wird zum einen zur Entwicklung wieder verwendbarer Software-

komponenten eingesetzt, aber auch um Softwarekomponenten mit einander zu verbinden und um Dienste des Windows Betriebssystems zu nutzen. In Anwendungsprogrammen wie dem Pulse LabShop wird „COM Automation“ eingesetzt. Damit ist es möglich, oft wiederholte Aufgaben zu automatisieren, oder das Programm aus einer anderen Anwendung heraus zu steuern. Mit dem „OLE2.0 Interface“ des LabShop's ist es möglich, eine oder auch eine Vielzahl von Aufgaben im LabShop zu automatisieren. Diese Aufgaben können das Starten einer Messung sein oder auch die Durchführung einer gesamten Messreihe. Es lassen sich anwendungsorientierte Lösungen für das Darstellen der Messdaten und der Reporterstellung entwickeln, ohne dabei von Grund auf neu beginnen zu müssen [Lit. 10].

Der Pulse LabShop ist sehr vielseitig einsetzbar und bietet eine enorm große Anzahl an Konfigurationsmöglichkeiten. Dementsprechend verschachtelt ist die Dokumentation des OLE2.0 Interface [Lit. 10]. Zu jeder der weiter oben in diesem Kapitel aufgeführten Aufgaben sind die benötigten Eigenschaften und Methoden aus der Dokumentation zu entnehmen. Die Dokumentation enthält Informationen darüber, wie die einzelnen Eigenschaften und Methoden zu verwenden sind. Wie die Implementierung in LV aussieht zeigt Abb. 8.18:

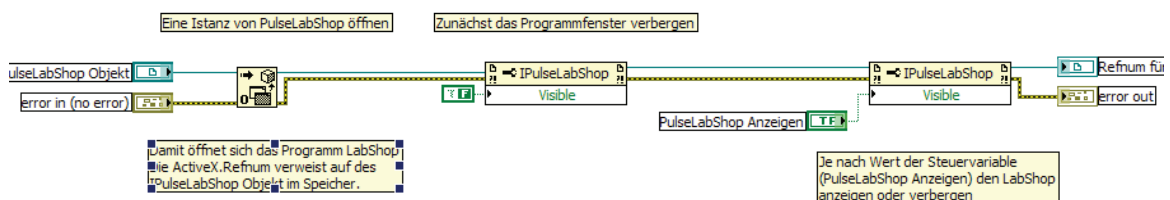
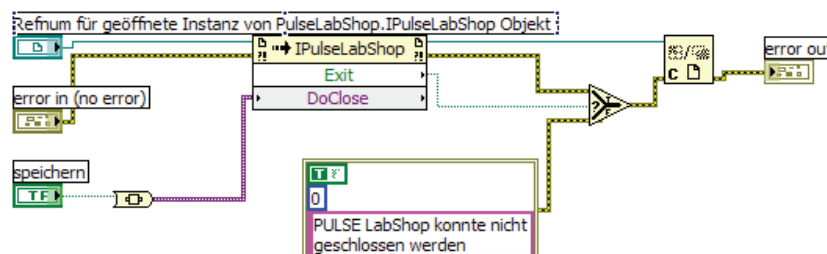


Abb. 8.18: Beispiel der Verwendung des OLE2.0 Interface in LV

Der LV-Funktion „ActiveX-Objekt öffnen“ wird eine leere ActiveX-Refnum übergeben. Diese Funktion öffnet dann eine Instanz des IPulseLabShop Objektes und gibt eine ActiveX-Refnum aus, die auf dieses Objekt verweist. ActiveX gehört zur Familie der COM-Technologien [Lit. 9]. Die LV Funktionen für ActiveX ermöglichen so die Verwendung des OLE2.0 Interface. Eine ActiveX-Refnum ist eine 32-Bit-Adresse, die auf ein bestimmtes ActiveX- bzw. COM-Objekt im Speicher verweist. Eine Refnum in LV ist vergleichbar mit einem Pointer wie er in der Programmiersprache C verwendet wird. Eine ActiveX-Refnum hat einen bestimmten Typ. Im obigen Beispiel ist die Refnum vom Typ IPulseLabShop. Das Lesen bzw. Schreiben von Eigenschaften geschieht über die LV-Funktion „Eigenschaftsknoten“. Einem Eigenschaftsknoten wird eine typisierte Refnum

übergeben. Über den Typ der Refnum lässt dieser Eigenschaftsknoten dann die Auswahl bestimmter Eigenschaften des Objektes zu. Die Zugriffsart (schreibend, lesend) lässt sich für jede ausgewählte Eigenschaft einstellen. Im Beispiel aus Abb. 8.18, wird die Eigenschaft Visible des Objektes PulseLabShop (Im Folgenden wird die Notation mit Punkt Operator verwendet: PulseLabShop.Visible = False) zunächst auf False gesetzt um den LabShop im ersten Moment zu verbergen. Gleich darauf wird PulseLabShop.Visible auf den Wert des Bedienelements „Pulse LabShop Anzeigen“ gesetzt. Wäre dieser True, würde das Programmfenster vom LabShop angezeigt.

Wichtig bei der Verwendung der LV-Bibliothek Pulse.lvlib ist, dass die Fehlercluster an jedes Sub-VI übergeben werden. Grund dafür ist, dass die Verwendung einer Softwareschnittstelle (OLE2.0 Interface), ähnlich wie eine Hardware-Schnittstelle, Fehlerquellen birgt. Beispielsweise ist es unbedingt erforderlich, dass der LabShop auf dem verwendeten System installiert ist und mit ihm das OLE2.0 Interface. Ist das nicht der Fall und wird trotzdem versucht auf die Softwareschnittstelle zuzugreifen, kann dies zu Fehlern führen. Zudem kann es bei jeder Software, die über COM gesteuert wird, zu Fehlern oder Abstürzen kommen, die eine erfolgreiche Kommunikation über die Softwareschnittstelle zum Scheitern bringen. Die in LV zur Verwendung des OLE2.0 Interfaces verwendeten ActiveX-Funktionen unterstützen die LV-Fehlerüberprüfung. Im Fehlerfall helfen die LV-Fehlermeldungen sowohl dem Benutzer der Anwendung als auch dem Entwickler, der die LV-Bibliothek verwendet, beim Beheben des Fehlers. Zudem werden eigene Fehler an das Fehlercluster ausgegeben. Das ist z.B. notwendig, wenn der Rückgabewert einer aufgerufenen Methode, aufzeigt, dass diese Methode nicht erwartungsgemäß ausgeführt wurde. Die nachfolgende Abbildung zeigt das BD des VI's Pulse_CloseLabShop.vi.



8.1: Beispiel Fehlergenerierung mit ActiveX-Funktionen

Hier wird die Methode Exit des IPulseLabShop-Objektes ausgeführt. In der Dokumentation [Lit. 10] ist diese wie folgt beschrieben:

[Boolean =] Object.Exit ([DoClose])

Der Rückgabewert (*[Boolean =]*) gibt Auskunft über das erfolgreiche Schließen des Pulse LabShop. Wenn dieser Rückgabewert wahr (True) ist, wird kein Fehler eingereicht, ist er jedoch nicht wahr (False), dann war das Schließen des LabShop's nicht erfolgreich und ein entsprechender Fehler wird ausgegeben.

Die beiden folgenden Abschnitte dokumentieren die VI's Pulse_Measure.vi und das Sub-VI Pulse_GetLinearAveragingTime.vi. Diese zeigen wie das OLE2.0 Interface zur Steuerung des LabShop's eingesetzt wurde.

8.3.3.1 Pulse_Measure.vi

Dieses VI dient zur Aufnahme eines Messpunktes mit dem LabShop. Das beinhaltet die Funktionen Messung starten, Messdauer abwarten, Messung stoppen und Messwert einlesen. Für jede der Funktionen gibt es ein entsprechendes Sub-VI. Im Anhang A2 ist das Blockdiagramm von Pulse_Measure.vi enthalten. Man sieht, dass die Sub-VI's in einer flachen Sequenzstruktur nacheinander abgearbeitet werden. Pulse_Measure.vi erhält vom Aufrufer alle dazu notwendigen Werte und Referenzen. Einige dieser Werte sind spezifisch für das verwendete LabShop-Projekt. Diese Bedienelemente sind mit den entsprechenden Standardwerten besetzt, sodass mit dem voreingestellten LabShop-Projekt [siehe 8.3.3, Abs. 3, default.pls] gearbeitet werden kann. Diese Werte sind der Name der DisplayGroup und des darin enthaltenen Display-Objektes, die beide im LabShop-Projekt definiert sind. Die DisplayGroup enthält das Display-Objekt aus dem dann der Messwert gelesen werden kann. Die erwähnten Namen wurden nicht als Konstanten in Pulse_Measure.vi eingefügt, damit es bei einer Erweiterung des Softwaremoduls möglich ist auch andere LabShop-Projekte zu verwenden und die entsprechenden Namen an Pulse_Measure.vi zu übergeben.

Die Sequenz beginnt mit Pulse_StartMeasurement.vi. Dieses VI führt die Methode Start des IPulseLabShop-Objektes aus. Um zu veranschaulichen, wie der Zugriff auf diese Methode des „OLE2.0 Interface“ mit der Benutzeroberfläche des LabShop's zusammenhängt, ist in der folgenden Abbildung [Abb. 8.19] die Befehlsschaltfläche markiert, die die Aktion der Methode bei direkter Verwendung des LabShop's ausführt.

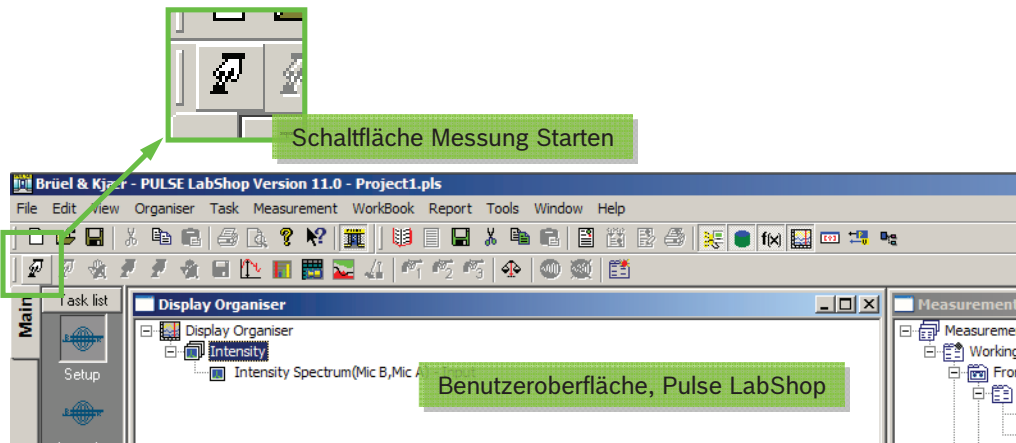


Abb. 8.19: Zusammenhang OLE2.0 und LabShop-GUI

Mit dieser Methode wird also zunächst der Messvorgang gestartet. Der nächste Case enthält eine Warte-Funktion. Diese dient dazu, die folgenden Aufrufe nicht unmittelbar nach dem Start der Messung an LabShop zu senden, da dies zu Fehlern führen kann. Danach wird das VI `Pulse_GetLinearAveragingTime.vi` ausgeführt. Es erhält vom Aufrufer eine ActiveX-Refnum und das Fehlercluster. Dieses VI wird im folgenden Kapitel [0] genauer erläutert. Es ermittelt die im LabShop-Projekt eingestellte Messdauer. Nachdem diese Messdauer abgewartet wurde, wird der Messwert mit dem VI, `Pulse_LabShopGetCursorReading.vi` ausgelesen. Dieses VI erhält die Refnum auf das Projekt und die weiter oben erwähnten Namen der DisplayGroup und des Displays. Mit diesem VI ist es möglich, verschiedene Werte der durchgeführten Messung auszulesen. Nach jedem Messvorgang wird das Messergebnis im dafür vorgesehenen Display angezeigt. Es gibt verschiedene so genannte „CursorReadings“, die in einem Display angezeigt werden können. Unabhängig von den angezeigten CursorReadings ermöglicht es dieses VI, eine Auswahl der zur Verfügung stehenden CursorReadings auszulesen. Diese beinhalten den Summenpegel, den Minimalpegel und den Maximalpegel. Von eigentlichem Interesse ist der Summenpegel: dieser wird vom Pulse System über die Messdauer hinweg berechnet.

8.3.3.2 `Pulse_GetLinearAveragingTime.vi`

`Pulse_GetLinearAveragingTime.vi` ist ein gutes Beispiel für die Verwendung des OLE2.0 Interface. Die Eigenschaft `LinearAveragingTime` findet sich in einer der unteren Ebenen der Struktur des Interface. Die Aufgabe dieses VI's sollte sein, die Dauer einer Messung anhand des LabShop-Projekts zu ermitteln. Dabei wurde wie folgt vorgegangen: zunächst wurde die Einstellung unter Verwen-

dung des LabShop's gesucht. Danach wurde das entsprechende Objekt in der Dokumentation des OLE2.0 Interface identifiziert, und die benötigte Eigenschaft ausgewählt.

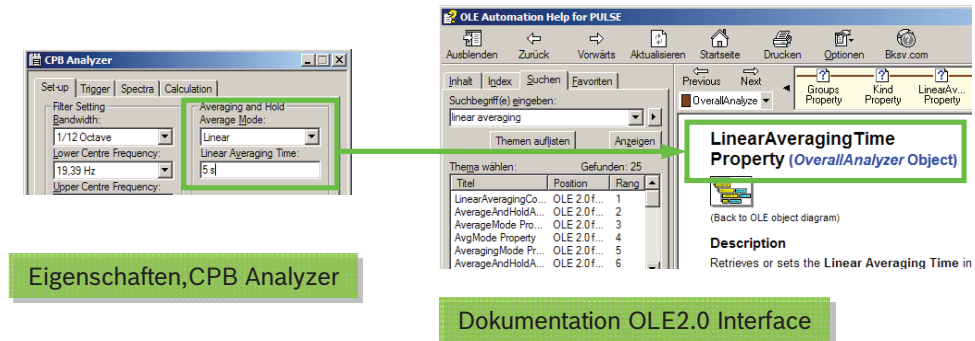
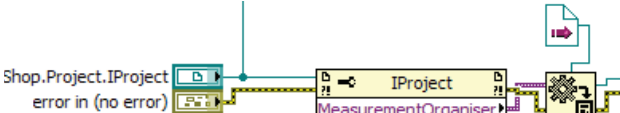
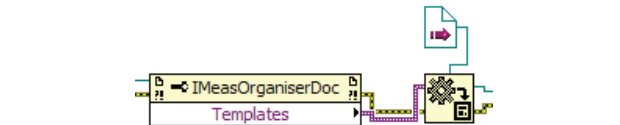
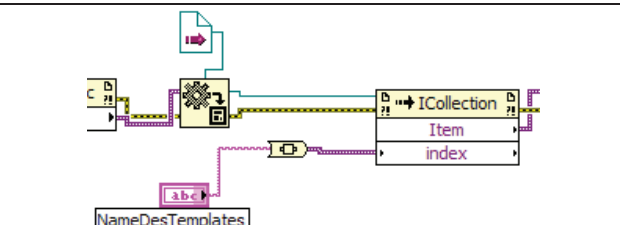
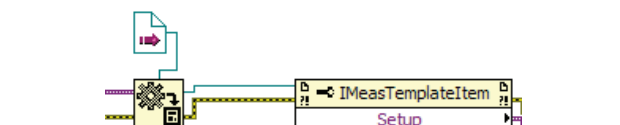
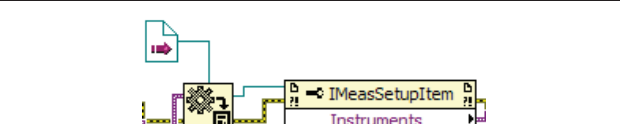
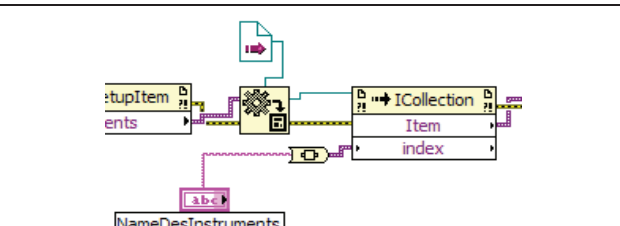
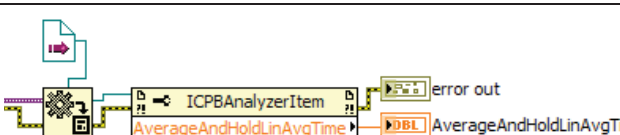


Abb. 8.20: Linear Averaging Time

Die Dokumentation liefert folgenden Pseudo-Code, der den Pfad zur gesuchten Eigenschaft darstellt.



```
Project.MeasurementOrganiser.Templates("Template").Setup.  
Instruments("OverallAnalyzer").LinearAveragingTime
```

Mit der Information aus der Dokumentation konnte nun das VI Implementiert werden. Um zu der Eigenschaft `LinearAveragingTime` zu gelangen, müssen die einzelnen Objekte in LV nacheinander aufgerufen werden. Tabelle [Tab. 8.5] zeigt wie dabei schrittweise vorzugehen ist.

LV Code	Erläuterung
 <p>Shop.Project.IProject error in (no error)</p> <p>MeasurementOrganiser</p>	Lesen der Eigenschaft: MeasurementOrganiser
 <p>IMeasOrganiserDoc Templates</p>	Lesen der Eigenschaft: Templates
 <p>ICollection Item index</p> <p>NameDesTemplates</p>	Templates ist eine Collection, über den Namen wird das MeasTemplateItem gelesen
 <p>IMeasTemplateItem Setup</p>	Lesen der Eigenschaft: Setup
 <p>IMeasSetupItem Instruments</p>	Lesen der Eigenschaft: Instruments
 <p>ICollection Item index</p> <p>NameDesInstruments</p>	Instruments ist ebenfalls eine Col- lection, das CPBAnalyzerItem wird über dessen Namen gelesen
 <p>ICPBAnalyzerItem AverageAndHoldLinAvgTi</p> <p>error out</p> <p>AverageAndHoldLinAvgTi</p>	

Tab. 8.5: Vorgehen beim Auslesen der Messzeit

Die LV Funktionen,

-  Daten nach Variant
-  und Variant nach Daten

ermöglichen den eigentlichen Austausch der Daten. Variant ist ein universeller Datentyp der Windows Betriebssystemfamilie. Daten des Typs Variant müssen in LV erst in einen bekannten Datentyp umgewandelt werden; genauso wie LV-Datentypen in Variant umgewandelt werden müssen, um sie einer ActiveX-Funktion zu übergeben.