Overview

The PAC-Designer Software Development Kit (PDSDK) allows software developers to utilize the design and JTAG programming capabilities of PAC-Designer. These functions can be called from programs written in Microsoft® Visual Basic®, C++, or other languages that provide support for Automation. This allows programmers to easily incorporate support for ispPAC® products in custom-designed automatic test systems. PDSDK also allows a programmer to develop user-defined macros that can be called by PAC-Designer to simplify ispPAC design and analysis tasks.

This application note is organized by topics according to the following outline:

1. **PDSDK System Requirements**
2. **What is Automation and How Does it Work with PAC-Designer**
3. **Getting Started with PDSDK**
   3.1. Examples Included with the PDSDK
   3.2. Using the Type Library from Visual Basic
4. **Working with PAC-Designer Objects**
   4.1. Creating the PAC-Designer Application Object
   4.2. Creating a PAC-Designer Schematic Document Object
   4.3. Saving Documents
   4.4. Closing a Document
   4.5. Terminating PAC-Designer
   4.6. PAC-Designer with Stand-alone Programs vs. User-defined Macros
5. **Manipulating the Document – Common Methods**
   5.1. Identifying the Device
   5.2. Setting and Reading IA Gains
   5.3. Setting Capacitor Values
   5.4. Updating the Schematic Display
   5.5. Downloading and Uploading
6. **Choosing from Multiple Configurations**
7. **Complete Control**
8. **Programmable Features of the ispPAC10**
   8.1. OA Feedback Links
   8.2. Common-Mode Output Voltage Source
   8.3. IA Interconnections
9. **Programmable Features of the ispPAC20**
   9.1. OA Feedback Links
   9.2. Common-Mode Output Voltage Source
   9.3. IA Interconnections
   9.4. Comparator Interconnections and Options
   9.5. Setting the DAC Value and Mode
   9.6. IA4 Logic Modes
10. **Programmable Features of the ispPAC30**
   10.1. OA Mode Configuration
   10.2. IA and MDAC Interconnections
   10.3. Summing Bus Interconnections
   10.4. Setting the MDAC Values
   10.5. Setting Voltage Reference Values
11. Programmable Features of the ispPAC80/81

11.1. Source Code

Appendix A: Example Program – ispPAC30 Voltage Reference Builder
Appendix B: Example Program – ispPAC10 Gain Calculation Utility
Appendix C: Summary of Methods and Procedures
Appendix D: PDSDK Symbolic

1. PDSDK System Requirements

PDSDK is included at no charge with PAC-Designer versions 1.3 and higher. To use PDSDK, either for application development, or as a runtime object (as part of an end-user application), a valid PAC-Designer license must be obtained for each computer on which the package is installed. Each license is keyed to the target computer’s hard disk drive ID. PAC-Designer licenses can be obtained free of charge from Lattice Semiconductor.

Pac-Designer runs under Microsoft Windows (95/98/NT/2000/ME/XP). It exposes a series of objects through Microsoft’s Automation system, and can therefore be accessed through programming languages such as Visual Basic and Visual C++. An application programming language system (e.g. C++) is not included in PDSDK and must be obtained separately. Sample programs are written using Microsoft Visual Basic 6.0.

2. PAC-Designer and Automation

Automation is a technology which allows software packages to expose their features to other applications and become a ‘server’. Automation allows other applications (client programs) to access and utilize an Automation server’s unique features in a simple and consistent manner.

PAC-Designer 1.3’s automation hooks allow other programs written in languages (such as Visual Basic) to use PAC-Designer’s device programming capabilities. This means that a user-developed program can use PAC-Designer’s capabilities to develop ispPAC hardware configurations, and then download these configurations into the actual ispPAC device, all without manual user intervention. Figure 1 shows the relationship between Visual Basic, PAC-Designer, and any other automation servers that may be used in a system. A similar diagram can be drawn for a PAC-Designer-enabled system developed under other programming systems, such as Visual C/C++.

Figure 1. Relationship between PAC-Designer and Visual Basic

While there potentially are many application areas in which automated control of ispPAC devices may be employed, perhaps the most useful of these is in manufacturing automation.
One of the major advantages offered by ispPAC devices is their in-system programmability. This feature allows a user to solder a device down to a circuit board, and then customize the behavior as the last stage of the manufacturing process. This customization can range from simple parametric adjustment (gain, corner frequency, or voltage offset) to selecting from one or more completely different circuit configurations. By providing a means of using PAC-Designer’s device programming capabilities, PDSDK makes it straightforward to build automated test systems which can exploit the unique capabilities of ispPAC integrated circuits.

3. Getting Started with PDSDK

This application note uses Microsoft Visual Basic (V6.0) for all discussions and examples, and assumes that the reader is familiar with this programming system. The concepts presented here can also be implemented in other programming languages, such as Visual C++. Additionally, familiarity with the architectures of the various members of the ispPAC product family (ispPAC10/20/30/80/81) is assumed; readers unfamiliar with the features of these products can find the necessary background information in their respective data sheets.

The PDSDK contains the following components to support Automation programming:

1. PDSDK help documentation in PAC-Designer
2. Symbolic Constant Files (in PDSDK\PacConstants\)
   - PAC10.BAS
   - PAC20.BAS
   - PAC30.BAS
   - PAC80.BAS
3. Example Projects in Visual Basic (in PDSDK\Samples_VisualBasic\)
   - PAC10_SetGainto5
   - PAC20_DacOffset
   - PAC20_DacOffsetOpenFile
   - PAC80_FilterSelect
4. PAC-Designer Type library

All of these components are automatically installed with all PAC-Designer installations.

3.1. Examples Included with the PDSDK

The PAC-Designer SDK includes source code for several short examples implemented in Visual Basic. The examples included in the PAC-Designer SDK (v1.3) are:

- PAC10_SetGainto5 – Very simple example of user-defined macro. Sets IA gain to ‘5’
- PAC20_DacOffset – User-defined macro which allows user to interactively increment and decrement the value of the ispPAC20’s DAC.
- PAC20_DACOffsetOpenFile – Same features as PAC20_DacOffset, but also loads a *.PAC file to completely configure the device.
- PAC80_FilterSelect – Opens the ispPAC80 filter table, selects a filter configuration, and downloads it into the device.

Each of these examples is intended to be a fully-functional user-defined macro that illustrates one or more operating principles of the PDSDK. Each example includes the forms, code modules, and project definition files to properly set up the project within the Visual Basic development environment.

3.2. Using the Type Library from Visual Basic

Once PAC-Designer is installed and licensed on a system, you can immediately begin developing applications using PDSDK in Visual Basic. One especially useful feature of PDSDK is the type library, which provides templates for calling PAC-Designer methods and also lists the many symbolic constants which are used as parameters. The PAC-Designer type library can be loaded into Visual Basic by selecting Project References from the main menu. In the references dialog box, check ‘PAC-Designer 1.3 Type Library’, and click ok ‘OK’, as shown in Figure 2.
Once the PAC-Designer type library is loaded, the object browser (Figure 3) can be used to see the available methods and symbolic constants. The object browser is invoked either from the main menu by View ⇒ Object Browser or by pressing <F2>. Selecting the PAC-Designer object class (list-box in upper left corner of browser) will limit the displayed methods, properties, and constants to those associated with PAC-Designer.

**Figure 2. Loading PAC-Designer Type Library**

**Figure 3. Viewing PAC-Designer Methods with Object Browser**
Although the type library can be a convenient programming aid, it is also possible to write programs without using it, by explicitly including the symbolic constant (*.BAS) files provided with PDSDK. Even when using the type library, the symbolic constant files are useful as reference materials, providing descriptive detail not available in the type library.

4. Working with PAC-Designer Objects

The first thing that must be done by an application to access PAC-Designer is to create objects that reference PAC-Designer and its associated documents (schematic diagrams). This requires a minimum of two variables, which should be declared to be accessible to all routines which need to directly interact with PAC-Designer. Inside a VB form, they may be declared with Dim statements outside of all routines.

```vbnet
' These Variables need to be accessible to all code
' communicating with PAC-Designer

Dim PACapp As Object
Dim PACdoc as object
```

You may also declare these variables in a module as global to the entire application with the Global keyword.

```vbnet
' This declaration makes these variables global and available to all
' forms and modules within a Visual Basic application

Global PACapp As Object
Global PACdoc as object
```

4.1. Creating the PAC-Designer Application Object

After one has declared variables to hold PAC-Designer objects, the next step is to create a PAC-Designer object. This is done by invoking the CreateObject function, and assigning the result to the PACapp object. The following subroutine in VB illustrates how this is done.

```vbnet
Sub Create_PAC_Object()
    ' Creates PAC-Designer object and checks to see if successful

    ' Set up an error trap
    on error goto Error_Handler1

    ' Create PAC-Designer object
    Set PACapp = CreateObject("PACDesigner.Application")

    ' Test to see if object was created, take corrective action if not
    If PACapp Is Nothing Then
        MsgBox "Object was not Created"
    ' Error Recovery Code here
    End If

End Sub
```
' Error handler – take corrective action if CreateObject function
' fails in a way that returns an error condition
'
Error_Handler1:

MsgBox "Can't Create Object"

' Error Recovery Code here

End Sub

When trying to create an object it is important to check for possible error conditions. This is accomplished in two
ways in the above example. The first is by the use of an On Error statement. If the CreateObject function
causes an error, execution will resume following the Error_Handler1: label. This routine also checks for the pos-
sibility that the CreateObject function does not return an object, by checking if the newly created PACapp object
is equal to Nothing.

4.2. Creating a PAC-Designer Schematic Document Object

Once one has successfully created a PAC-Designer object, the next step is to create a schematic document. This
can be done either by creating a new blank schematic document, or by loading one from a *.PAC file stored on disk.
To create a new schematic, use the NewDocument and ActiveDocument methods. Example code for creating a
blank ispPAC20 schematic document is shown below.

    PACapp.NewDocument "ispPAC20"
    Set PACdoc = PACapp.ActiveDocument

The NewDocument method creates a schematic document for the specified ispPAC IC. The IC types supported by
PAC-Designer V1.3 are “ispPAC10”, “ispPAC20”, “ispPAC30”, “ispPAC80”, and “ispPAC81”. If the NewDocument
method is successful, the ActiveDocument method will return a valid document object, otherwise it will return a
value of Nothing. As in the case of creating the PAC-Designer application object, one should include appropriate
error checking to ensure a robust application.

In addition to creating a blank schematic document, it is also possible to load an existing design from a *.PAC file
stored on disk. This is accomplished by use of the Open method. The following example code shows how this is
done.

    PACapp.Open "C:\PAC-Designer\Design1.PAC"
    Set PACdoc = PACapp.ActiveDocument

When specifying the filename, it is important to include the .PAC extension, as PDSDK will not assume the exten-
sion by default.

4.3. Saving Documents

It is also possible to save a document to a file. There are two methods for doing so; Save and SaveAs. The differ-
ence between the two methods is that the Save method uses a filename already associated with the document
while the SaveAs method explicitly requires a filename to be supplied. As in the case of the Open method, PAC-
Designer will not assume a filename extension; this must be explicitly provided.

    ' Example of Save method
    ,
    PACdoc.Save

    ' Example of SaveAs method
    ,
    PACdoc.SaveAs "C:\PAC-Designer\MyDesign.PAC"
4.4. Closing a Document
To close a document without saving it, one uses the Close method, as shown in the example below.

```vba
' Example of closing a Schematic document
PACdoc.close
Set PACdoc = Nothing
```

If changes have been made to the document, and have not been saved, invoking the Close method will result in a screen prompt asking the user if he wants to save the document. To avoid this, explicitly save the document using the SaveAs method before closing it.

Note that closing a document does not automatically reset the value of the PACdoc object to Nothing. To avoid problems that could potentially result from attempting to use the invalid PACdoc object, it should be explicitly set to Nothing immediately after closing the document.

If a single document is open in PAC-Designer, the Close method will also terminate and close PAC-Designer.

4.5. Terminating PAC-Designer
On exiting an application-level program, it is desirable to terminate PAC-Designer. The Quit method is provided for this purpose.

```vba
' Example of code to terminate PAC-Designer
Set PACdoc = Nothing
PACapp.Quit
Set PACapp = Nothing
```

As in the case of closing a document, if there are unsaved changes in any document open in PAC-Designer, invoking the Quit method will cause PAC-Designer to prompt the user asking to save the document(s).

Because the Quit method does not automatically reset the PACapp object to a value of Nothing, one should explicitly set it to this value after quitting PAC-Designer to avoid potential problems associated with using an invalid reference. Note that to successfully invoke the quit method, the PACapp object needs to be valid, which is why PACapp is set to Nothing after invoking Quit, and not before.

4.6. PAC-Designer with Stand-alone Programs vs. User-defined Macros
PAC-Designer Automation scenarios can be split into two types: “standalone” external program and user-defined macro:

The first scenario is the “standalone” external program that (for example) controls an automated end-of-line test system. PAC-Designer will play a role, together with the data-gathering programs. In this case, the external program is responsible for orchestrating the entire process, and lives longer than any other program in the process. Such “standalone” external programs launch PAC-Designer by creating an instance of the PAC-Designer object, create one or more schematic documents, set various parameters in those schematics, and then download them into actual parts. They may do this repeatedly using several different schematics.

The second scenario is when the external program is launched by PAC-Designer from the “User-Defined Macros” dialog box (on the Tools menu). (To enable PAC-Designer to find a user-defined macro, its executable file should be placed in the \PAC-Designer\UserDefinedMacro directory.) Launching the program this way in no way restricts what
it can do, but it should behave like a dialog box built right into PAC-Designer; see the BiQuad macro for ispPAC10 an example. Such programs should act on the currently open document and exit when done, leaving the document open.

PAC-Designer’s automation interface supports both types of interaction. The primary difference between the two scenarios is in how the automation link is created.

When writing a program that launches PAC-Designer, the following series of operations will typically be required:

1. Create PAC-Designer object
2. Create (or load) one or more schematic object(s)
3. Perform operations on schematic object(s)
4. Close schematic object(s)
5. Close PAC-Designer object and terminate PAC-Designer

The details of implementing each of these operations have been described in the preceding sections, and an example of a simple application program (ispPAC30 Voltage Reference Builder) that illustrates this flow is given in Appendix A.

A program that is launched by PAC-Designer, such as a user-defined macro, needs to perform a slightly different series of operations:

1. Program confirms that it is being called by PAC-Designer
2. Reference to existing PAC-Designer object is obtained
3. Reference to existing active schematic object is obtained
4. Schematic object is verified to be for correct device type (e.g. ispPAC30)
5. Perform operations on a schematic object.
6. Program terminates, without closing schematic object or PAC-Designer

When PAC-Designer launches a user-defined macro, it places the switch “/RunAsMacro” in the command line of the launched program. In Visual Basic, the command line can be read and tested for the presence of this switch by the following code:

```vbnet
If Instr(Command$, ”/RunAsMacro”) = 0 then
    End ' Program not launched from PAC-Designer, Terminate!
End if
```

Because PAC-Designer is presumed to be already running, one does not to create a PAC-Designer application object, but just needs to get a reference to the existing instance of PAC-Designer. This can be accomplished by using the GetObject() function, as shown below:

```vbnet
Set PACapp = GetObject(“”,”PACDesigner.Application”)  
```

After getting the reference to the PAC-Designer application object, the next thing to do is to get the reference to the active document. Again, since this program was launched from a PAC-Designer, it presumably was started to perform some operation on an already-open document. The following statement gets the reference to the active document:

```vbnet
Set PACdoc = PACapp.ActiveDocument
```

Because the launched program was not responsible for creating the schematic document it will be operating on, it is essential that it determine the device type, and take appropriate action. This is accomplished by reading and testing the schematic’s DeviceType property. The following code shows an example of testing for an ispPAC10 schematic:
Using the PAC-Designer
Software Development Kit

Lattice Semiconductor

IF PACdoc.DeviceType <> “ispPAC10” then
  End ‘ Not an ispPAC10, Terminate Program!
End If

When implementing a program that performs the above operations, appropriate error-handling procedures should be included. While omitted here, routines including error-handling code are presented in the sample program listed in Appendix B (ispPAC10 Gain Calculation Utility).

Again, to enable PAC-Designer to find a user-defined macro, its executable file should be placed in the \PAC-Designer\UserDefinedMacro directory.

5. Manipulating the Document – Common Methods

Once one has opened a document in PAC-Designer, one can manipulate all aspects of this document to specify a design. Among the items that can be controlled through PDSDK are gain, capacitance, DAC values, and on-chip interconnections. PDSDK provides numerous methods to allow the software designer to easily perform these tasks. This section will describe common operations, and how they are implemented through PDSDK.

5.1. Identifying the Device

When loading a document from an external file, or when using PDSDK to implement a design macro, it is important for an application to be able to determine the type of device represented in the document. The DeviceType property provides this function, returning a string representing the device. The following example shows how the DeviceType property can be used to determine if a document describes an ispPAC20 design.

' Test to see if document device is ispPAC20

MyDevice$ = PACdoc.DeviceType
If MyDevice$<> “ispPAC20” Then End ‘ terminate if not ispPAC20

Note that DeviceType is a **read-only** property. It is not possible to change the device type of a document after it has been created.

5.2. Setting and Reading IA Gains

Because setting IA gain is a commonly invoked function when using ispPAC IC’s, a method has been provided specifically for performing this operation. The SetGainValue method sets an IA’s gain based on the IA’s identifying symbol and the gain. For example, to set the gain of IA3 in an ispPAC10 to –7, one would use the following code:

' Sets gain of IA3 to -7

PACdoc.SetGainValue PAC10_PACBlock_2_IA3_Gain, -7

Invalid gain values will not cause error conditions, but will be coerced to an acceptable value by PAC-Designer. As an example, if a gain of +15 is specified, the SetGainValue method will set an actual value of +10.

A corresponding method, GetGainValue exists to read IA gains. The following example reads the gain of IA2 into a variable ‘GainValue’.

' Read IA2 Gain

GainValue = PACdoc.GetGainValue(PAC10_PACBlock_2_IA3_Gain)

Note that these methods act on the schematic object in PAC-Designer, and not on the ispPAC IC. To modify (or read) parameter values in the actual IC requires executing a download (or upload) cycle.
5.3. Setting Capacitor Values
In ispPAC parts, the corner frequencies of filters are controlled by selecting appropriate values for the device’s on-chip capacitors. To accommodate this need, PAC-Designer provides the SetCapValue method, which sets capacitor values based on the capacitor's identifying symbol and the floating-point value of the capacitor in Farads. For example, to set the PACBlock #1 capacitor in an ispPAC10 to 25.2pF, one would use the following code:

```
  ' Sets PACBlock 1’s capacitor to 25.5pF
  ,
  PACdoc.SetCapValue PAC10_PACBlock_2_Cap, 25.2E-12
```

Invalid gain values will not cause error conditions, but will be coerced to an acceptable value by PAC-Designer. Because capacitor values vary in discrete steps, PAC-Designer will select the nearest capacitor value to the one specified. In the above example, the ispPAC10 can't implement a capacitor value of 25.2pF, so PAC-Designer will round the requested value to the nearest value supported (25.08pF).

5.4. Updating the Schematic Display
None of the methods for changing device configuration will automatically refresh the PAC-Designer graphic display. In the case of a user-defined macro, refreshing the PAC-Designer after the macro has executed is essential. In the case of a automatic test system, having an up-to-date schematic display can be useful as a debugging aid. For these reasons PAC-Designer provides the UpdateViews method, which updates a schematic document's display. The following example shows how the UpdateViews method is invoked.

```
  ' Update Graphic Display of Document
  ,
  PACdoc.UpdateViews
```

Note that this method merely redraws the schematic document, and does not restore the display if minimized or bring the document to the foreground if it is under other documents.

5.5. Downloading and Uploading
Once a suitable design has been configured in a PAC-Designer schematic, the next step is to get that design into hardware. To do this requires the use of the Download method.

For programming purposes, individual ispPAC devices are organized as a set of one or more $E^2$ registers. The Download method can be used to either access individual registers, or update all registers at once. The following example illustrates downloading the ispPAC20’s DAC $E^2$ register, and downloading the whole schematic (including the DAC register) to the ispPAC20.

```
  ' download to just the DAC
  ,
  PACdoc.Download PAC20_JTAGREG_DAC

  ' Download whole device
  ,
  PACdoc.Download PAC20_JTAGREG_ALL
```

The advantage of downloading to a single register over downloading to a whole device is that it takes less time. This time savings can become significant in iterative test-trim schemes, where the device must be repetitively programmed with different values. For details on the register organization of particular devices, please refer to the online PDSDK documentation and the appropriate ispPAC data sheet.

It is also possible to read the contents of an ispPAC device into the schematic document using the Upload method. The following example illustrates uploading the contents of an ispPAC20 into a schematic document.
Using the PAC-Designer
Lattice Semiconductor Software Development Kit

' Upload whole device
PACdoc.Upload PAC20_JTAGREG_ALL

It is also possible to upload individual registers, doing so will only modify the relevant parts of the schematic document.

6. Choosing from Multiple Configurations

One scenario in a manufacturing programming environment is where an ispPAC device will be selectively programmed with one of small number of configurations, with the choice of configurations determined at final test. While one approach might be to load a given schematic document when a particular configuration when required, and then close it prior to loading a new configuration, this technique could cause application or OS stability problems, especially in the case that tens of thousands of open/close cycles are executed. A better solution is to load all of the potentially required configurations into PAC-Designer at startup, and then select which of these configurations is to be downloaded. In addition to providing a reduced likelihood of adverse system-related occurrences, this technique is also significantly faster, as it avoids the overhead associated with file access and document creation.

Multiple schematic documents can be accessed in PAC-Designer by declaring an array of document objects, and assigning the objects created with each document to a member of the array. The following code illustrates loading an array of five unique ispPAC20 schematics. In this example, error handling code is omitted for clarity.

' Declare PAC objects

Dim PACapp as object
Dim PACdoc(1 to 5) as object

Sub create_docs()

' assumes PACapp already defined!

PACapp.Open "C:\PAC-Designer\Design1.PAC"
Set PACdoc(1) = PACapp.ActiveDocument

PACapp.Open "C:\PAC-Designer\Design2.PAC"
Set PACdoc(2) = PACapp.ActiveDocument

PACapp.Open "C:\PAC-Designer\Design3.PAC"
Set PACdoc(3) = PACapp.ActiveDocument

PACapp.Open "C:\PAC-Designer\Design4.PAC"
Set PACdoc(4) = PACapp.ActiveDocument

PACapp.Open "C:\PAC-Designer\Design5.PAC"
Set PACdoc(5) = PACapp.ActiveDocument

End Sub

To download a particular document, use the Download method associated with the desired document. To continue from the above example, to download 'Design3.PAC', which is associated with the PACdoc(3) array element, one would execute the following code:

' Download 'Design3.PAC' (PACdoc(3))

PACdoc(3).Download
In this way, PAC-Designer provides a simple and efficient method for allowing the programmer to implement test systems which program one of a few configurations.

7. Complete Control

There are many programmable features in ispPAC devices beyond gain and capacitor value, such as interconnections, DAC and MDAC values, and operating mode options for many PACells. PAC-Designer provides complete control of these features through the Automation interface through the `SetSymValue` and `GetSymValue` methods. These two methods allow one to set or read the value for any PAC-Designer symbol. A PAC-Designer symbol is a bit or group of bits in a document database that controls a feature for an ispPAC device.

When the `SetSymValue` method is invoked, it must be supplied with a *symbol identifier* and a *symbol value*. As an example, consider using `SetSymValue` to set the value of the ispPAC20's DAC.

```
    ' Set ispPAC20 DAC to 45 (-1.954V)
    ',
    PACdoc.SetSymValue PAC20_PACell_DAC_Value, 45
```

In this example, the symbol identifier is `PAC20_PACell_DAC_Value`, which identifies the ispPAC20 DAC register symbol. The value to be written into this symbol is the constant 45. After the `SetSymValue` method is invoked, the ipPAC20's DAC will be set to a value of 45 (-1.954V).

The `GetSymValue` method allows one to read the value of any given symbol back from a PAC-Designer schematic document. This method takes a symbol identifier as input, and returns the value of that symbol from the schematic document. As an example, the following code reads the value of the ispPAC20 DAC:

```
    ' Read ispPAC20 DAC to MyDACValue
    ',
    MyDACValue = PACdoc.GetSymValue(PAC20_PACell_DAC_Value)
```

Because there are significant user-level architectural differences among the various members of the ispPAC product line, each member has its own set of programmable features. The following sections will outline the features unique to each ispPAC device.

8. Programmable Features of the ispPAC10

In addition to IA gains and feedback capacitor values, the following features can be programmed:

- OA Feedback Links
- OA Common-Mode Output Voltage
- IA interconnections

8.1. OA Feedback Links

The feedback links for the OAs control whether they will operate in DC closed loop or open loop mode. When the feedback link is closed, the OA functions as a unity-gain amplifier/lown-pass filter, with a corner frequency determined by the feedback capacitor value. When the feedback link is opened, the OA's DC gain is typically 80dB, and the OA behaves like an integrator, with an integration time constant dependent on the feedback capacitor value. The following code provides an example of opening and closing feedback links.

```
    ' Open Feedback link on OA1
    ',
    PACdoc.SetSymValue PAC10_PACBlock_1_Feedback, PAC10_CONN_FB_OPEN

    ' Close Feedback link on OA3
    ',
    PACdoc.SetSymValue PAC10_PACBlock_3_Feedback, PAC10_CONN_FB_CLOSED
```
8.2. Common-Mode Output Voltage Source
The ispPAC10 also allows the user to control the common-mode output voltage of the outputs. Each output's common-mode voltage can be set to either the internal 2.5V reference, or set to track the voltage at the CMVIN input pin. The common-mode output voltage source for each OA can be selected independently. The following code shows how an OA’s CMVout can be set to track either internal (2.5V) or external (CMVIN) references.

```plaintext
' Set OA2 CMVout to 2.5V (internal) - '0' is symbol value
PACdoc.SetSymValue PAC10_PACBlock_2_CMVinEnable, 0

' Set OA3 CMVout to CMVIN pin (external) - '1' is symbol value
PACdoc.SetSymValue PAC10_PACBlock_3_CMVinEnable, 1
```

8.3. IA Interconnections
Each IA in an ispPAC10 can be connected to one of several signal sources, or left in an unconnected state. To modify an IA’s input connection, the user must define both the IA, and the source to which it should be connected. Note that the signal sources available to IA1 through IA4 are different from those available to IA5 through IA8. The following code segment shows examples of connecting IN2 to IA1, and disconnecting IA5.

```plaintext
' Connect IA1 input to IN2
PACdoc.SetSymValue PAC10_PACBlock_1_IA1_Interconnect, PAC10_CONN_IA1234_IN2

' Disconnect IA5 input (no connection)
PACdoc.SetSymValue PAC10_PACBlock_3_IA5_Interconnect, PAC10_CONN_NONE
```

When writing an application which reconfigures the interconnections, it is often worthwhile to disconnect ALL relevant interconnections before establishing new ones. Doing so allows the application to begin with a ‘clean sheet’ and prevents the previous state of the schematic document from adversely affecting the results of the reconfiguration.

9. Programmable Features of the ispPAC20
In addition to IA gains and feedback capacitor values, the following features can be programmed in the ispPAC20:

- OA Feedback Links
- Common-Mode Output Voltage
- IA Interconnection
- Comparator Interconnections
- DAC value
- Comparator/Logic modes

In addition to the code shown below, complete examples of simple user-defined macros are presented in PAC20_DacOffset and PAC20_DacOffsetOpenFile, both found at \PAC-Designer\PDSDK\Samples_Visual Basic.

9.1. OA Feedback Links
As in the case of the ispPAC10, the feedback links for the ispPAC20’s OAs control whether they will operate in DC closed loop or open loop mode. When the feedback link is closed, the OA functions as a unity-gain amplifier/low-pass filter, with a corner frequency determined by the feedback capacitor value. When the feedback link is opened, the OA’s DC gain is typically 80dB, and the OA behaves like an integrator, with an integration time constant dependent on the feedback capacitor value. The following code provides an example of opening and closing feedback links.
' Open Feedback link on OA1
',
PACdoc.SetSymValue PAC20_PACBlock_1_Feedback, PAC20_CONN_FB_OPEN

' Close Feedback link on OA2
',
PACdoc.SetSymValue PAC20_PACBlock_2_Feedback, PAC20_CONN_FB_CLOSED

9.2. Common-Mode Output Voltage Source

The ispPAC20 also allows the user to control the common-mode output voltage of the outputs. Each output's common-mode voltage can be set to either the internal 2.5V reference, or set to track the voltage at the CMVIN input pin. The common-mode output voltage source for each OA can be selected independently. The following code shows how an OA's CMVout can be set to track either internal (2.5V) or external (CMVIN) references.

' Set OA1 CMVout to 2.5V (internal) - '0' indicates internal reference
',
PACdoc.SetSymValue PAC20_PACBlock_1_CMVinEnable, 0

' Set OA2 CMVout to CMVIN pin (external) - '1' indicates CMVIN pin
',
PACdoc.SetSymValue PAC20_PACBlock_2_CMVinEnable, 1

9.3. IA Interconnections

Each IA in an ispPAC20 can be connected to one of several signal sources, or left in an unconnected state. To modify an IA's input connection, the user must define both the IA, and the source to which it should be connected. For the ispPAC20, IA4 has a different set of potential input sources, and therefore a separate set of source identifiers than those used with IA1 through IA3. Additionally, each multiplier input to IA1 one can be connected independently. The following example code connects the 'A' multiplexor input of IA1 to IN2, and connects IA4 to the +3V reference.

' Connect IA1 (Mux A) input to IN2
',
PACdoc.SetSymValue PAC20_PACBlock_1_IA1_A_Interconnect, PAC20_CONN_IA123_IN2

' Connect IA4 to 3V reference
',
PACdoc.SetSymValue PAC20_PACBlock_2_IA4_Interconnect, PAC20_CONN_IA4_3V

When writing an application which reconfigures the interconnections, it is often worthwhile to disconnect ALL relevant interconnections before establishing new ones. Doing so allows the application to begin with a 'clean sheet' and prevents the previous state of the schematic document from adversely affecting the results of the reconfiguration.

9.4. Comparator Interconnections and Options

The ispPAC20's comparators (CP1, CP2) have both programmable interconnections, and a wide variety of functional options. The comparators receive their inputs from a different set of signal than the IAs, and can also be left in an unconnected state. To modify a comparator input connection, the user must define both the comparator input to be connected, and the signal source to which it should be connected. The following example code connects the CP1's '+' input to CPIN, and CP2's '-' input to the output of the DAC.
' Connect + input of CP1 to CPIN

PACdoc.SetSymValue PAC20_PACell_Comp_1_in1, PAC20_CONN_CxIN1_CIN1

' Connect - input of CP2 to DAC

PACdoc.SetSymValue PAC20_PACell_Comp_2_in2, PAC20_CONN_CxIN2_DAC

Note that the '-' terminals of the comparators use a different set of constants (PAC20_CONN_CxIN1_xxxx) to define their signal sources than the '+' inputs (PAC20_CONN_CxIN2_xxxx).

Another feature of the ispPAC20's comparators is an optional 47mV (nominal) hysteresis. Enabling or disabling the hysteresis enables or disables it for both comparators. Example code showing how to do this is shown below:

' Enable Hysteresis on BOTH comparators

PACdoc.SetSymValue PAC20_PACell_Comp_hyst, 1

' Disable Hysteresis on BOTH comparators

'PACdoc.SetSymValue PAC20_PACell_Comp_hyst, 0

The outputs of the comparators can be set to either an active, or hi-Z state. The following code shows how to do this.

' Set Comparator Outputs to Hi-Z

PACdoc.SetSymValue PAC20_PACell_Comp_COENB, 1

' Enable Comparator Outputs

PACdoc.SetSymValue PAC20_PACell_Comp_COENB, 0

Additionally, the output of the CP1 comparator can either be set to asynchronous, or clocked mode (with PC providing the clock input). The following code shows how to set CP1 into both latched and asynchronous modes

' Enable CP1 flip-flop, clocked by PC pin

PACdoc.SetSymValue PAC20_PACell_C1_latch, 1

' Make CP1 Asynchronous (bypass flip-flop)

PACdoc.SetSymValue PAC20_PACell_C1_latch, 0

Finally, the ispPAC20’s comparator outputs can either be XOR’ed to provide a windowed comparator function, or used to control the SET and RESET inputs of an RS flip-flop. The following code shows how to put the WINDOW output into each of these modes

' Enable FLIP-FLIP mode

PACdoc.SetSymValue PAC20_PACell_Comp_WSELB, 1

' Enable WINDOW mode

PACdoc.SetSymValue PAC20_PACell_Comp_WSELB, 0
9.5. Setting the DAC Value and Mode

The ispPAC20’s DAC can be used to provide a programmable voltage ranging from –3 to +3 V. The DAC is controlled by an 8-bit control word (0-255), with 0 representing full-scale negative output (-3.00V), 255 representing full-scale positive output (2.977V), and 128 representing an output of zero volts. The following code loads the DAC with a value of 192 (VOUT = +1.5V).

    ' Load DAC with 192 ( 0xC0 )
    '
    PACdoc.SetSymValue PAC20_PACell_DAC_Value, 192

Additionally, the ispPAC20’s DAC has two operating modes which control its source of data, parallel mode and shift register mode. In parallel mode, the DAC is loaded from either the E2CMOS memory or from the parallel interface, while in parallel mode the DAC is loaded from either the JTAG or SPI serial interfaces. Example code for setting these two operating modes is shown below.

    ' Set DAC into Parallel Mode
    '
    PACdoc.SetSymValue PAC20_PACell_DAC_DACMODE, 0

    ' Set DAC into Shift-register Mode
    '
    PACdoc.SetSymValue PAC20_PACell_DAC_DACMODE, 1

9.6. IA4 Logic Modes

IA4 receives its input through a +/- polarity switch. The state of this switch can be controlled in one of several ways:

1. Fixed, non-inverting
2. Through a signal input to the PC pin
3. Flip-Flop mode (State of Window output flip-flop)
4. From the state of the CP1OUT pin

The following code shows how to set the polarity switch to each of these behaviors.

    ' Set IA4 Polarity switch to non-inverting mode
    '
    PACdoc.SetSymValue PAC20_PACBlock_2_IA4_PCmode, 0

    ' Set IA4 Polarity switch to follow PC pin
    '
    PACdoc.SetSymValue PAC20_PACBlock_2_IA4_PCmode, 1

    ' Set IA4 Polarity switch to Flip-Flop mode
    '
    PACdoc.SetSymValue PAC20_PACBlock_2_IA4_PCmode, 2

    ' Set IA4 Polarity switch to follow CP1OUT
    '
    PACdoc.SetSymValue PAC20_PACBlock_2_IA4_PCmode, 3

10. Programmable Features of the ispPAC30

In addition to IA gains and feedback capacitor values, the following features can be programmed in the ispPAC30:

- OA mode configurations
- IA Interconnections
- Summing Bus Interconnections
• MDAC value
• Voltage Reference Values

10.1. OA Mode Configuration
The ispPAC30’s output amplifiers can be configured to work either as an amplifier/low-pass filter, an integrator, or a comparator, depending on the configuration of the feedback links. The following code shows examples of setting output amplifier 1 into its various operating modes.

```
' Set OA1 mode to Amplifier/Filter (mode=0)
',
PACdoc.SetSymValue PAC30_PACBlock_OA1_Mode, 0
' Set OA1 mode to Amplifier/Filter (mode=2)
',
PACdoc.SetSymValue PAC30_PACBlock_OA1_Mode, 2
' Set OA1 mode to Amplifier/Filter (mode=3)
',
PACdoc.SetSymValue PAC30_PACBlock_OA1_Mode, 3
```

Note that changing the OA operating mode does not automatically change the value of the feedback capacitor.

Additionally, it is possible to select whether an individual OA will be powered up or in low-power sleep mode. Configuring OAs into low-power mode can useful in battery-powered applications because it ensures that the ispPAC30 will consume minimal power at system turn-on; a power-up command can then be sent by a supervisory microcontroller when the ispPAC30’s function is needed. The two OAs have independent power control, as is illustrated in the following code.

```
' Power-down OA1
',
PACdoc.SetSymValue PAC30_PACBlock_OA1PD, 1
' Power-up OA2
',
PACdoc.SetSymValue PAC30_PACBlock_OA2PD, 0
```

10.2. IA and MDAC Interconnections
Each IA in an ispPAC30 can be connected to one of several signal sources, or left in an unconnected state. To modify an IA’s input connection, the user must define both the IA, and the source to which it should be connected. Note that the signal sources available to IA1, IA2, and MDAC1 are different form those available to IA3,IA4, and MDAC2. The following code segment shows examples of connecting IN2 to the ‘A’ multiplexor input of IA1, and connecting VREF2 to MDAC2.

```
' Connect IA1A (MUX ‘A’) input to IN2
',
PACdoc.SetSymValue PAC30_PACBlock_IA1A_Interconnect, PAC30_CONN_IN2
' Connect MDAC2 to VREF2
',
PACdoc.SetSymValue PAC30_PACBlock_DAC2_Interconnect, PAC30_CONN_REF
```

When writing an application which reconfigures the interconnections, it is often worthwhile to disconnect ALL relevant interconnections before establishing new ones. Doing so allows the application to begin with a ‘clean sheet’ and prevents the previous state of the schematic document from adversely affecting the results of the reconfiguration.
10.3. Summing Bus Interconnections
The ispPAC30 provides a routing pool that allows variable interconnections from the IAs and MDACs to the output amplifiers. Each IA or MDAC can be independently added into either the input of OA1 or OA2. The following code connects the output of IA1 to the input of OA2.

```plaintext
' Connect output of IA1 to input of OA2
PACdoc.SetSymValue PAC30_PACBlock_IA1_IAtoOA, PAC30_CONN_IAtoOA_OA2
```

10.4. Setting the MDAC Values
The ispPAC30’s MDACs allow the user to easily implement precision gain and offset adjustment circuits. Each MDAC is controlled by an 8-bit control word (0-255), with 0 representing full-scale negative gain (-1.00), 255 representing full-scale positive gain (0.992), and 128 representing a gain of zero. The following code loads MDAC1 with a value of 192 (Gain = +0.5).

```plaintext
' Load MDAC1 with 192 ( 0xC0 )
PACdoc.SetSymValue PAC30_PACBlock_DAC1, 192
```

10.5. Setting Voltage Reference Values
The ispPAC30 provides two separate voltage references, each of which can be independently programmed to output one of 7 voltages. The example below shows code to set VREF1 to 1024mV.

```plaintext
' Set VREF1 to 1024mV (code=4)
PACdoc.SetSymValue PAC30_PACBlock_Ref1Level, 4
```

The following table provides the codes required to set the VREF sources

<table>
<thead>
<tr>
<th>VREF Voltage (mV)</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>512</td>
<td>3</td>
</tr>
<tr>
<td>1024</td>
<td>4</td>
</tr>
<tr>
<td>2048</td>
<td>5</td>
</tr>
<tr>
<td>2500</td>
<td>6</td>
</tr>
</tbody>
</table>

11. Programmable Features of the ispPAC80/81
The ispPAC80 and ispPAC81 are relatively straightforward parts to program, as most of the configuration is in the capacitors and the front-end IA gain. In the ispPAC80/81, the gain of this amplifier can only assume values of +1, +2, +5 and +10.

The only other significant programmable feature is that of the A/B start-up configuration. The following code shows examples of setting the startup configuration to both ‘A’ and ‘B’ states.

```plaintext
' Setting the symbol to ’1’ selects the ’B’ configuration...
PACdoc.SetSymValue PAC80_AB, 1

' while setting the symbol to ’0’ selects the ’A’ configuration
PACdoc.SetSymValue PAC80_AB, 0
```
Because the ispPAC80 and ispPAC81 designs are frequently taken from filter databases included with PAC-Designer, two additional methods have been provided to take advantage of this resource. The `LoadFilterTable` method opens the filter database, automatically selecting the database appropriate to the device type represented by the document (ispPAC80 or ispPAC81 only). Once a filter table has been loaded, the `CopyfilterToSchematic` method allows one to load a particular filter, selected by ID number, into the schematic document. The following code shows how to load filter #1423 (50.01 kHz Legendre) from the filter database into an ispPAC80 schematic document.

```vbscript
' Open the ispPAC80 Filter Table – Assumes Document is ispPAC80!
',
PACdoc.LoadFilterTable

' Copy filter #1423 (50.01kHz Legendre) into configuration ‘A’
' ..to copy to config ‘B’, change the ‘0’ to a ‘1’
',
PACdoc.CopyFilterToSchematic 1423, 0
```

### 11.1. Source Code

A complete example of a simple user-defined macro for the ispPAC80 is presented in `PAC80_FilterSelect`, which can be found at `\PAC-Designer\PDSDK\Samples_Visual Basic`. 

Appendix A. Example Program: ispPAC30 Voltage Reference Builder

This example shows code that invokes PAC-Designer to program an ispPAC30 as a variable voltage source. This program allows a user to specify a desired voltage ranging from 0 to 2.48V, and configures VREF1, MDAC1, and OA1 in the ispPAC30 to provide the desired voltage at OUT1.

This program comprises three subroutines:

- **FormLoad()** - Initializes link to PAC-Designer, creates blank ispPAC30 schematic
- **btnProgram_Click()** - Configures ispPAC30 and downloads to the device
- **btnExit_Click()** - Terminates PAC-Designer and closes program

The design form for this program, with control names indicated, looks like this:

![Design Form Diagram](image)

The program source code is given here:

```vbnet
' VREFMAKE.EXE - Copyright (C) 2001 Lattice Semiconductor Corp.
'
' This program calls PAC-Designer to configure an ispPAC30 as a programmable
' voltage reference
' It uses VREF1, MDAC1, and OA1 to implement the reference. It allows the user
' to enter a desired voltage between 0 and 2.5V and will attempt to scale the
' reference and MDAC to provide the most accurate reference. It also sets any
' necessary interconnections, assuming an initially blank schematic.
'
' These Variables are globally accessable from all procedures in this program
'
Dim PACapp As Object
Dim PACdoc As Object
Dim Gain As Integer
```

'==============================================================================

```vbnet
' Uses VREF1, MDAC1, and OA1 to make a voltage reference at OUT1.
Desired Voltage (0-2.5V): 0
Actual Programmed: 0
```

```vbnet
Program
```

```vbnet
Exit
```

```
```
Private Sub btnProgram_Click()

    ' This routine executes when the PROGRAM button is Clicked
    ' Read Desired Voltage from txtGain Text entry box, trim blanks,
    ' convert to numeric value and limit range to [0..2.48]
    Dim voltage As Single, vref_scale As Single
    Dim vref_range As Long, MDAC_count As Long

    voltage = Val(Trim(txtVRequested.Text))
    If voltage < 0 Then voltage = 0
    If voltage > 2.48 Then voltage = 2.48

    ' Determine which range for VREF to use for maximum resolution
    Select Case voltage
        Case 0 To 0.0635
            vref_range = 0
            vref_scale = 0.064
        Case 0.0635 To 0.127
            vref_range = 1
            vref_scale = 0.128
        Case 0.127 To 0.254
            vref_range = 2
            vref_scale = 0.256
        Case 0.254 To 0.508
            vref_range = 3
            vref_scale = 0.512
        Case 0.508 To 1.016
            vref_range = 4
            vref_scale = 1.024
        Case 1.016 To 2.032
            vref_range = 5
            vref_scale = 2.048
        Case 2.032 To 2.48
            vref_range = 6
            vref_scale = 2.5
    End Select

    ' Calculate MDAC count based on voltage and vref_scale
    MDAC_count = 128 + Int((128 * voltage / vref_scale) + 0.5)

    ' Display actual voltage in lb1VActual
    lb1VActual.Caption = Str$(CSng(MDAC_count - 128) * vref_scale / 128)

    ' Set VREF1
    PACdoc.SetSymValue PAC30_PACBlock_Ref1Level, vref_range
' Set MDAC1
','PACdoc.SetSymValue PAC30_PACBlock_DAC1, MDAC_count

' Connect MDAC1 to VREF1
','PACdoc.SetSymValue PAC30_PACBlock_DAC1_Interconnect, PAC30_CONN_REF

' Connect MDAC1 to OA1
','PACdoc.SetSymValue PAC30_PACBlock_DAC1_IAtoOA, PAC30_CONN_IAtoOA_OA1

' Set OA1 to Amplifier/Filter mode
','PACdoc.SetSymValue PAC30_PACBlock_OA1_Mode, 0

' Set OA1 Capacitor to maximum value (64.01pF)
','PACdoc.SetCapValue PAC30_PACBlock_OA1_CAP, 0.00000000006401

' Remember to Update the Schematic Display!
','PACdoc.UpdateViews

' Download from PAC-Designer to an actual Part
','PACdoc.Download PAC30_JTAGREG_ALL

End Sub

'==============================================================================

Private Sub btnExit_Click()

' This routine executes when the QUIT button is clicked
'
' Because we changed the schematic, we want to save it to a junk file
' so PAC-Designer doesn’t ask if you want to save
','PACdoc.SaveAs “JUNK_XXX.PAC”

' We want to release object references and terminate PAC-Designer.
'
Private Sub Form_Load()
    ' This routine is the first thing called when this program is
    ' started. It takes care of the housekeeping needed to establish
    ' the link to PAC-Designer and check to see if everything is OK.
    '  
    ' The following operations must be performed:
    '  
    ' 1) Create PAC-Designer Application Object
    ' 2) Create a blank ispPAC30 Schematic
    '  
    ' Set Error trap
    '  
    On Error GoTo err_handler
    ' 1) Create PAC-Designer Application object, using CreateObject() method
    Set PACapp = CreateObject("PacDesigner.Application")
    If PACapp Is Nothing Then
        MsgBox "Unable to Create PAC-Designer Object"
    End If
    ' 2) Create New ispPAC30 schematic
    PACapp.NewDocument "ispPAC30"
    Set PACdoc = PACapp.ActiveDocument
    If PACdoc Is Nothing Then
        MsgBox "No Active Schematic Found"
    End If
    Exit Sub
    
err_handler:
    MsgBox "Unidentified Error!"
End Sub
Appendix B. Example Program: ispPAC10 Gain Calculation Macro

This example shows a simple program which is called from PAC-Designer as a user-defined macro. This program allows a user to specify a gain from -110 to 110, and it configures OA1 and OA2 in the ispPAC10 to implement the gain from IN1 to OUT1. Note that although this program is designed to be called from PAC-Designer (to illustrate construction of a user-defined macro), minor modifications can be made to allow it to run as a standalone program which calls PAC-Designer as a programming utility.

This program comprises three subroutines:

- FormLoad() - Initializes link to PAC-Designer
- btnConfig_Click() - Performs configuration based on gain
- btnQuit_Click() - Closes program

The design form for this program, with control names indicated, looks like this:

![Design Form](image)

The program source code is given here:

```vbnet
' GAINCALC.EXE - Copyright (C) 2001 Lattice Semiconductor Corp.
'
' This program runs as a PAC-Designer user-defined macro, and configures OA1 and OA2 to provide a variable gain amplifier from input1 to output 1
' It allows the user to enter a gain between -110 and +110, and configures both the IA’s, OA’s, and interconnections to perform this function.
'
' These Variables are globally accessible from all procedures in this program
'
Dim PACapp As Object
Dim PACdoc As Object
Dim Gain As Integer

'==============================================================================
Private Sub btnConfig_Click()

' This routine executes when the CONFIGURE button is Clicked
',
' Read Gain from txtGain Text entry box, trim blanks, convert to number,
' convert result to integer
'
Gain = Int(Val(Trim(txtGain.Text)))

' SELECT CASE Statement allows you to different configurations depending on
' the entered gain. Gains from 1-10 only use one PACBlock, while gains from
' 11-110 use both
'
Select Case Gain
Case 0
   ' Configure only OA1, but disconnect form inputs
   ' Close OA1 Feedback Link, Select Minimum Cap Value
   PACdoc.SetSymValue PAC10_PACBlock_1_Feedback, PAC10_CONN_FB_CLOSED
   PACdoc.SetCapValue PAC10_PACBlock_1_CAP, 0.000000000001
   ' Disconnect IA1,IA2 inputs, set gains to 1
   PACdoc.SetSymValue PAC10_PACBlock_1_IA1_Interconnect, PAC10_CONN_NONE
   PACdoc.SetSymValue PAC10_PACBlock_1_IA2_Interconnect, PAC10_CONN_NONE
   PACdoc.SetGainValue PAC10_PACBlock_1_IA1_Gain, 1
   PACdoc.SetGainValue PAC10_PACBlock_1_IA2_Gain, 1
Case -10 To -1, 1 To 10
   ' Configure OA1, using only IA1
   ' Close OA1 Feedback Link, Select Minimum Cap Value
   PACdoc.SetSymValue PAC10_PACBlock_1_Feedback, PAC10_CONN_FB_CLOSED
   PACdoc.SetCapValue PAC10_PACBlock_1_CAP, 0.000000000001
   ' connect IA1 to IN1, disconnect IA2, set gains to IA1 gain=GAIN, IA2 gain=0
   PACdoc.SetSymValue PAC10_PACBlock_1_IA1_Interconnect, PAC10_CONN_IA1234_IN1
   PACdoc.SetSymValue PAC10_PACBlock_1_IA2_Interconnect, PAC10_CONN_NONE
   PACdoc.SetGainValue PAC10_PACBlock_1_IA1_Gain, Gain
   PACdoc.SetGainValue PAC10_PACBlock_1_IA2_Gain, 1
Case -110 To -11, 11 To 110
   ' Need to split up gain into 10X (g10) and 1X (g1) pieces. Also need to
   ' determine sign.
   ',
   g = Abs(Gain)    ' need to start with |Gain| to make this work
   g10 = g \ 10      ' separate 10X section
   g1 = g - (10 * g10)  ' separate 1X section
   g10 = g10 * Sgn(Gain)  ' set to right sign
   g1 = g1 * Sgn(Gain)    ' also set this to the right sign
Lattice Semiconductor

Using the PAC-Designer
Software Development Kit

' Configure OA1 and OA2
',
' Close OA1, OA2 Feedback Links, Select Minimum Cap Values
',
PACdoc.SetSymValue PAC10_PACBlock_1_Feedback, PAC10_CONN_FB_CLOSED
PACdoc.SetCapValue PAC10_PACBlock_1_CAP, 0.000000000001
PACdoc.SetSymValue PAC10_PACBlock_2_Feedback, PAC10_CONN_FB_CLOSED
PACdoc.SetCapValue PAC10_PACBlock_2_CAP, 0.000000000001
,
' Connect IA1 to OA2, IA2 to IN1, IA3 to IN1, IA4 to nothing
',
PACdoc.SetSymValue PAC10_PACBlock_1_IA1_Interconnect, PAC10_CONN_IA1234.OUT2
PACdoc.SetSymValue PAC10_PACBlock_1_IA2_Interconnect, PAC10_CONN_IA1234.IN1
PACdoc.SetSymValue PAC10_PACBlock_2_IA3_Interconnect, PAC10_CONN_IA1234.IN1
PACdoc.SetSymValue PAC10_PACBlock_2_IA4_Interconnect, PAC10_CONN_NONE
,
' Set IA1 gain=10, IA2 gain=g1, IA3 gain=g10, IA4 gain=1
',
PACdoc.SetGainValue PAC10_PACBlock_1_IA1_Gain, 10
PACdoc.SetGainValue PAC10_PACBlock_1_IA2_Gain, g1
PACdoc.SetGainValue PAC10_PACBlock_2_IA3_Gain, g10
PACdoc.SetGainValue PAC10_PACBlock_2_IA4_Gain, 1

Case Else
',
' Invalid Gain!
',
MsgBox "Invalid Gain: " + Str$(Gain)

End Select

' Remember to Update the Schematic Display!
',
PACdoc.UpdateViews

End Sub

'==============================================================================

Private Sub btnQuit_Click()

' This routine executes when the QUIT button is clicked
',
' Note that we DON’T want to close either the open document or PAC-Designer,
' merely exit THIS program
',
Set PACdoc = Nothing
Set PACapp = Nothing
End

End
End Sub

'=============================================================

Private Sub Form_Load()

' This routine is the first thing called when the calculator is
' invoked from PAC designer. It takes care of the housekeeping
' needed to establish the link to PAC-Designer and check to see if
' everything is OK.

' The following operations must be performed:
'
' 1) Check command line to see if launched form PAC-Designer
' 2) Get a reference to the PAC-Designer Application Object
' 3) Get the reference to the active schematic
' 4) Check to see if Device type is really an ispPAC10
'
' 1) Check to see if Program launched from PAC-Designer
' PAC-Designer includes '/RunAsMacro' switch in command line
' when it launches a user-defined macro
'
If InStr(Command$, "'/RunAsMacro") = 0 Then
    MsgBox "This Program Must Be run From PAC-Designer"
End
End If

' Set Error trap
'
On Error GoTo err_handler

' 2) Get PAC-Designer Application object, using GetObject() method
'
Set PACapp = GetObject("", "PacDesigner.Application")
If PACapp Is Nothing Then
    MsgBox "Unable to Find PAC-Designer Object"
End
End If

' 3) Get active schematic
'
Set PACdoc = PACapp.ActiveDocument
If PACdoc Is Nothing Then
    MsgBox "No Active Schematic Found"
End
End If
' 4) Check if Active Document is for an ispPAC10
'
If PACdoc.DeviceType <> "ispPAC10" Then
    MsgBox "Active Schematic is not for an ispPAC10"
    End
End If

Exit Sub

err_handler:

    MsgBox "Unidentified Error!"
    End

End Sub
## Appendix C. Summary of PDSDK (Version 1.3) Methods and Properties

### Application Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveDocument()</td>
<td>Returns active document Object</td>
</tr>
<tr>
<td>NewDocument()</td>
<td>Creates a new document object</td>
</tr>
<tr>
<td>Open(Filename as string)</td>
<td>Creates a new document object from a *.PAC file</td>
</tr>
<tr>
<td>Quit()</td>
<td>Terminates PAC-Designer</td>
</tr>
</tbody>
</table>

### Document (Schematic) Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close()</td>
<td>Close document</td>
</tr>
<tr>
<td>CopyFilterToSchematic(...)</td>
<td>Loads specified filter from table into specified configuration register (A or B).</td>
</tr>
<tr>
<td>DeviceType</td>
<td>Returns device type of document (string)</td>
</tr>
<tr>
<td>Download(...)</td>
<td>Writes specified register from document to physical IC</td>
</tr>
<tr>
<td>GetGainValue(...)</td>
<td>Returns gain value for specified amplifier</td>
</tr>
<tr>
<td>GetSymValue(...)</td>
<td>Returns value for specified symbol</td>
</tr>
<tr>
<td>LoadFilterTable()</td>
<td>Loads Filter Table</td>
</tr>
<tr>
<td>Save()</td>
<td>Saves document under current filename</td>
</tr>
<tr>
<td>SaveAs(Filename as String)</td>
<td>Saves document under specified filename</td>
</tr>
<tr>
<td>SetCapValue(...)</td>
<td>Set capacitor value for specified capacitor. Capacitor value in pF</td>
</tr>
<tr>
<td>SetGainValue(...)</td>
<td>Set gain value for specified amplifier</td>
</tr>
<tr>
<td>SetSymValue(...)</td>
<td>Set symbol value for specified symbol</td>
</tr>
<tr>
<td>UpdateViews()</td>
<td>Refresh graphic display of document</td>
</tr>
<tr>
<td>Upload(Register as integer)</td>
<td>Read specified register from IC to document</td>
</tr>
</tbody>
</table>
Appendix D. PDSDK Symbolic Constants

These constants are defined in the *.BAS files in the PDSDK constants directory. Additionally, these constants are also included in the PAC-Designer type library where they can be accessed automatically when Visual Basic’s references are set to include PAC-Designer.

**ispPAC10 Constants**

### PACblock 1 Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC10_PACBlock_1_Feedback</td>
<td>3</td>
<td>References Feedback Link, use PAC10_CONN_FB_OPEN PAC10_CONN_FB_CLOSED to set</td>
</tr>
<tr>
<td>PAC10_PACBlock_1_CAP</td>
<td>6</td>
<td>References Capacitor for SetCapValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_1_CMVinEnable</td>
<td>45</td>
<td>References CMV source 0=2.5,1=CMVext pin</td>
</tr>
<tr>
<td>PACBlock_1_IA1_Gain</td>
<td>4</td>
<td>References IA1 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_1_IA1_Invert</td>
<td>7</td>
<td>References IA1 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_1_IA1_Interconnect</td>
<td>9</td>
<td>References Source for IA1</td>
</tr>
<tr>
<td>PACBlock_1_IA2_Gain</td>
<td>5</td>
<td>References IA2 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_1_IA2_Invert</td>
<td>8</td>
<td>References IA2 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_1_IA2_Interconnect</td>
<td>10</td>
<td>References Source for IA2</td>
</tr>
</tbody>
</table>

### PACblock 2 Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC10_PACBlock_2_Feedback</td>
<td>14</td>
<td>References Feedback Link, use PAC10_CONN_FB_OPEN PAC10_CONN_FB_CLOSED to set</td>
</tr>
<tr>
<td>PAC10_PACBlock_2_CAP</td>
<td>17</td>
<td>References Capacitor for SetCapValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_2_CMVinEnable</td>
<td>46</td>
<td>References CMV source 0=2.5,1=CMVext pin</td>
</tr>
<tr>
<td>PACBlock_2_IA3_Gain</td>
<td>15</td>
<td>References IA3 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_2_IA3_Invert</td>
<td>18</td>
<td>References IA3 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_2_IA3_Interconnect</td>
<td>20</td>
<td>References Source for IA3</td>
</tr>
<tr>
<td>PACBlock_2_IA4_Gain</td>
<td>16</td>
<td>References IA4 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_2_IA4_Invert</td>
<td>19</td>
<td>References IA4 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_2_IA4_Interconnect</td>
<td>21</td>
<td>References Source for IA4</td>
</tr>
</tbody>
</table>

### PACblock 3 Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC10_PACBlock_3_Feedback</td>
<td>25</td>
<td>References Feedback Link, use PAC10_CONN_FB_OPEN PAC10_CONN_FB_CLOSED to set</td>
</tr>
<tr>
<td>PAC10_PACBlock_3_CAP</td>
<td>28</td>
<td>References Capacitor for SetCapValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_3_CMVinEnable</td>
<td>45</td>
<td>References CMV source 0=2.5,1=CMVext pin</td>
</tr>
<tr>
<td>PACBlock_3_IA5_Gain</td>
<td>26</td>
<td>References IA5 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_3_IA5_Invert</td>
<td>29</td>
<td>References IA5 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_3_IA5_Interconnect</td>
<td>31</td>
<td>References Source for IA5</td>
</tr>
<tr>
<td>PACBlock_3_IA6_Gain</td>
<td>27</td>
<td>References IA6 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_3_IA6_Invert</td>
<td>30</td>
<td>References IA6 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_3_IA6_Interconnect</td>
<td>32</td>
<td>References Source for IA6</td>
</tr>
</tbody>
</table>
### PacBlock 4 Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC10_PACBlock_4_Feedback</td>
<td>36</td>
<td>References Feedback Link, use PAC10_CONN_FB_OPEN or PAC10_CONN_FB_CLOSED to set.</td>
</tr>
<tr>
<td>PAC10_PACBlock_4_CAP</td>
<td>39</td>
<td>References Capacitor for SetCapValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_4_CMVinEnable</td>
<td>45</td>
<td>References CMV source 0=2.5,1=CMVext pin</td>
</tr>
<tr>
<td>PACBlock_4_IA7_Gain</td>
<td>37</td>
<td>References IA7 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_4_IA7_Invert</td>
<td>40</td>
<td>References IA7 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_4_IA7_Interconnect</td>
<td>42</td>
<td>References Source for IA7</td>
</tr>
<tr>
<td>PACBlock_4_IA8_Gain</td>
<td>38</td>
<td>References IA8 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC10_PACBlock_4_IA8_Invert</td>
<td>41</td>
<td>References IA8 Gain polarity</td>
</tr>
<tr>
<td>PAC10_PACBlock_4_IA8_Interconnect</td>
<td>43</td>
<td>References Source for IA8</td>
</tr>
</tbody>
</table>

### IA Input Sources

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC_10_CONN_NONE</td>
<td>0</td>
<td>Unconnected state for all IAs</td>
</tr>
<tr>
<td>PAC10_CONN_IA1234_IN1</td>
<td>1</td>
<td>Connect IA1,2,3,4 to IN1</td>
</tr>
<tr>
<td>PAC10_CONN_IA1234_IN2</td>
<td>2</td>
<td>Connect IA1,2,3,4 to IN2</td>
</tr>
<tr>
<td>PAC10_CONN_IA1234_OUT1</td>
<td>3</td>
<td>Connect IA1,2,3,4 to OUT1</td>
</tr>
<tr>
<td>PAC10_CONN_IA1234_OUT2</td>
<td>4</td>
<td>Connect IA1,2,3,4 to OUT2</td>
</tr>
<tr>
<td>PAC10_CONN_IA1234_OUT4</td>
<td>5</td>
<td>Connect IA1,2,3,4 to OUT4</td>
</tr>
<tr>
<td>PAC10_CONN_IA5678_IN3</td>
<td>1</td>
<td>Connect IA5,6,7,8 to IN3</td>
</tr>
<tr>
<td>PAC10_CONN_IA5678_IN4</td>
<td>2</td>
<td>Connect IA5,6,7,8 to IN4</td>
</tr>
<tr>
<td>PAC10_CONN_IA5678_OUT3</td>
<td>3</td>
<td>Connect IA5,6,7,8 to OUT3</td>
</tr>
<tr>
<td>PAC10_CONN_IA5678_OUT4</td>
<td>4</td>
<td>Connect IA5,6,7,8 to OUT4</td>
</tr>
<tr>
<td>PAC10_CONN_IA5678_OUT2</td>
<td>5</td>
<td>Connect IA5,6,7,8 to OUT2</td>
</tr>
</tbody>
</table>

### Miscellaneous ispPAC10 Control Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC10_CONN_FB_CLOSED</td>
<td>0</td>
<td>Use to define Feedback link closed</td>
</tr>
<tr>
<td>PAC10_CONN_FB_OPEN</td>
<td>1</td>
<td>Use to define Feedback link open</td>
</tr>
<tr>
<td>PAC10_ESF</td>
<td>44</td>
<td>References Electronic Security fuse 0=unsecured 1=secured</td>
</tr>
<tr>
<td>PAC10_JTAGREG_USR</td>
<td>0</td>
<td>References USR register (download method)</td>
</tr>
<tr>
<td>PAC10_JTAGREG_CALIBRATE</td>
<td>1</td>
<td>References Calibrate Register (factory use only)</td>
</tr>
<tr>
<td>PAC10_JTAGREG_ALL</td>
<td>-4</td>
<td>References ALL registers for download</td>
</tr>
</tbody>
</table>
ispPAC20 Constants

### PACBlock 1 Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_PACBLOCK_1_CAP</td>
<td>6</td>
<td>Reference OA1 cap</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_FEEDBACK</td>
<td>3</td>
<td>Reference OA1 feedback link</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA1_GAIN</td>
<td>5</td>
<td>Reference IA1 gain</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA1_INVERT</td>
<td>8</td>
<td>Reference IA1 gain inversion</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA1_A_INTERCONNECT</td>
<td>10</td>
<td>Reference IA1 mux A input connect</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA1_B_INTERCONNECT</td>
<td>11</td>
<td>Reference IA1 mux B input connect</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA2_GAIN</td>
<td>4</td>
<td>Reference IA2 gain</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA2_INVERT</td>
<td>7</td>
<td>Reference IA2 gain inversion</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_IA2_INTERCONNECT</td>
<td>9</td>
<td>Reference IA2 input connect</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_1_CMVINENABLE</td>
<td>25</td>
<td>Reference OA1 CMV source; 0 = 2.5V internal, 1 = external CMVin pin</td>
</tr>
</tbody>
</table>

### PACBlock 2 Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_PACBLOCK_2_CAP</td>
<td>18</td>
<td>Reference OA2 cap</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_FEEDBACK</td>
<td>15</td>
<td>Reference OA2 feedback link</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_IA3_GAIN</td>
<td>16</td>
<td>Reference IA3 gain</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_IA3_INVERT</td>
<td>19</td>
<td>Reference IA3 gain inversion</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_IA3_INTERCONNECT</td>
<td>21</td>
<td>Reference IA3 input connect</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_IA4_GAIN</td>
<td>17</td>
<td>Reference IA4 gain</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_IA4_INTERCONNECT</td>
<td>22</td>
<td>Reference IA4 input connect</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_IA4_PCMODE</td>
<td>20</td>
<td>Reference IA4 Polarity control mode. Control options are: 0 = Fixed, non-inverting, 1 = Controlled by PC pin, 2 = Flip-Flop mode, 3 = Controlled by CP1out</td>
</tr>
<tr>
<td>PAC20_PACBLOCK_2_CMVINENABLE</td>
<td>26</td>
<td>Reference OA1 CMV source; 0 = 2.5V internal, 1 = external CMVin pin</td>
</tr>
</tbody>
</table>
### IA Source Connection Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_CONN_IA123_NONE</td>
<td>0</td>
<td>No connection</td>
</tr>
<tr>
<td>PAC20_CONN_IA123_IN1</td>
<td>1</td>
<td>Connect IA1,2,3 to IN1</td>
</tr>
<tr>
<td>PAC20_CONN_IA123_IN2</td>
<td>2</td>
<td>Connect IA1,2,3 to IN2</td>
</tr>
<tr>
<td>PAC20_CONN_IA123_OUT1</td>
<td>3</td>
<td>Connect IA1,2,3 to OUT1</td>
</tr>
<tr>
<td>PAC20_CONN_IA123_OUT2</td>
<td>4</td>
<td>Connect IA1,2,3 to OUT2</td>
</tr>
<tr>
<td>PAC20_CONN_IA123_IN3</td>
<td>5</td>
<td>Connect IA1,2,3 to IN3</td>
</tr>
<tr>
<td>PAC20_CONN_IA123_DAC</td>
<td>6</td>
<td>Connect IA1,2,3 to DAC</td>
</tr>
<tr>
<td>PAC20_CONN_IA4_IN5V</td>
<td>1</td>
<td>Connect IA4 to 1.5V Reference</td>
</tr>
<tr>
<td>PAC20_CONN_IA4_IN2</td>
<td>2</td>
<td>Connect IA4 to IN2</td>
</tr>
<tr>
<td>PAC20_CONN_IA4_OUT1</td>
<td>3</td>
<td>Connect IA4 to OUT1</td>
</tr>
<tr>
<td>PAC20_CONN_IA4_OUT2</td>
<td>4</td>
<td>Connect IA4 to OUT2</td>
</tr>
<tr>
<td>PAC20_CONN_IA4_3V</td>
<td>5</td>
<td>Connect IA4 to 3V Reference</td>
</tr>
<tr>
<td>PAC20_CONN_IA4_DAC</td>
<td>6</td>
<td>Connect IA4 to DAC</td>
</tr>
</tbody>
</table>

### DAC Configuration Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_PACELL_DAC_VALUE</td>
<td>39</td>
<td>Reference DAC value (0-255)</td>
</tr>
<tr>
<td>PAC20_PACELL_DAC_DACMODE</td>
<td>35</td>
<td>Reference DAC mode control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Parallel mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Shift-register mode</td>
</tr>
</tbody>
</table>

### Comparator Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_PACELL_COMP_1_IN1</td>
<td>27</td>
<td>References comparator 1 ‘+’ input source</td>
</tr>
<tr>
<td>PAC20_PACELL_COMP_1_IN2</td>
<td>28</td>
<td>References comparator 1 ‘-’ input source</td>
</tr>
<tr>
<td>PAC20_PACELL_COMP_2_IN1</td>
<td>29</td>
<td>References comparator 2 ‘+’ input source</td>
</tr>
<tr>
<td>PAC20_PACELL_COMP_2_IN2</td>
<td>30</td>
<td>References comparator 2 ‘-’ input source</td>
</tr>
<tr>
<td>PAC20_PACELL_COMP_HYST</td>
<td>31</td>
<td>Reference hysteresis control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = disabled</td>
</tr>
<tr>
<td>PAC20_PACELL_C1_LATCH</td>
<td>32</td>
<td>Reference CP1out Flipflop enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = direct</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = clocked</td>
</tr>
<tr>
<td>PAC20_PACELL_COMP_WSELB</td>
<td>33</td>
<td>Reference Window/Flip-flop mode control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = window mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = flip-flop mode</td>
</tr>
<tr>
<td>PAC20_PACELL_COMP_COENB</td>
<td>34</td>
<td>Reference Comparator output enables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = hi-Z</td>
</tr>
</tbody>
</table>
### Comparator Input Source Constants

Note that ‘CXIN1’ constants are for use with positive comparator inputs and ‘CXIN2’ constants are for use with negative comparator inputs.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_CONN_CXIN1_3V</td>
<td>0</td>
<td>Connect 3V reference to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN1_CIN1</td>
<td>1</td>
<td>Connect CIN input to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN1_DAC</td>
<td>2</td>
<td>Connect DAC to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN1_IN3</td>
<td>3</td>
<td>Connect IN3 input to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN1_OUT2</td>
<td>4</td>
<td>Connect OUT2 to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN1_NONE</td>
<td>5</td>
<td>No connection to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN1_1P5V</td>
<td>6</td>
<td>Connect 1.5V reference to ‘+’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_NONE</td>
<td>0</td>
<td>No connection to ‘-’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_CIN1</td>
<td>1</td>
<td>Connect CIN input to ‘-’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_DAC</td>
<td>2</td>
<td>Connect DAC to ‘-’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_IN3</td>
<td>3</td>
<td>Connect IN3 input to ‘-’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_OUT2</td>
<td>4</td>
<td>Connect OUT2 to ‘-’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_3V</td>
<td>5</td>
<td>Connect 3V ref to ‘-’ comparator input</td>
</tr>
<tr>
<td>PAC20_CONN_CXIN2_1P5V</td>
<td>6</td>
<td>Connect 1.5V ref to ‘-’ comparator input</td>
</tr>
</tbody>
</table>

### Miscellaneous Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC20_JTAGREF_USR</td>
<td>0</td>
<td>Specify USR register for download</td>
</tr>
<tr>
<td>PAC20_JTAGREF_CALIBRATE</td>
<td>1</td>
<td>Specify CAL register for download (for factory use only)</td>
</tr>
<tr>
<td>PAC20_JTAGREF_DAC</td>
<td>2</td>
<td>Specify DAC register for download</td>
</tr>
<tr>
<td>PAC20_JTAGREF_ALL</td>
<td>-4</td>
<td>Specify ALL registers for download</td>
</tr>
<tr>
<td>PAC20_ESF</td>
<td>38</td>
<td>Reference Security fuse 0=unsecured/1=secured</td>
</tr>
<tr>
<td>PAC20_CONN_FB_CLOSED</td>
<td>0</td>
<td>Close OA feedback link</td>
</tr>
<tr>
<td>PAC20_CONN_FB_OPEN</td>
<td>1</td>
<td>Open OA feedback link</td>
</tr>
</tbody>
</table>
# ispPAC30 Constants

## IA Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_PACBLOCK_IA1_GAIN</td>
<td>4</td>
<td>References IA1 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA1_POL</td>
<td>29</td>
<td>References IA1 Gain polarity</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA1_SELMUX1</td>
<td>12</td>
<td>References mux select polarity: PAC30_CONN_MUX_A implies MSEL = 0, selects A; PAC30_CONN_MUX_B implies MSEL = 0, selects B</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA1A_INTERCONNECT</td>
<td>13</td>
<td>References IA1 mux ‘A’ input source</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA1b_INTERCONNECT</td>
<td>14</td>
<td>References IA1 mux ‘B’ input source</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA2_GAIN</td>
<td>5</td>
<td>References IA2 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA2_POL</td>
<td>30</td>
<td>References IA2 Gain polarity</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA2_INTERCONNECT</td>
<td>11</td>
<td>References IA2 Input Source</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA3_GAIN</td>
<td>6</td>
<td>References IA3 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA3_POL</td>
<td>31</td>
<td>References IA3 Gain polarity</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA3_INTERCONNECT</td>
<td>10</td>
<td>References IA3 Input Source</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4_GAIN</td>
<td>7</td>
<td>References IA4 (Gain) for SetGainValue() method</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4_POL</td>
<td>32</td>
<td>References IA4 Gain polarity</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4_SELMUX1</td>
<td>15</td>
<td>References mux select polarity: PAC30_CONN_MUX_A implies MSEL = 0, selects A; PAC30_CONN_MUX_B implies MSEL = 0, selects B</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4A_INTERCONNECT</td>
<td>16</td>
<td>References IA4 mux ‘A’ input source</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4b_INTERCONNECT</td>
<td>17</td>
<td>References IA4 mux ‘B’ input source</td>
</tr>
</tbody>
</table>

## MDAC Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_PACBLOCK_DAC1</td>
<td>8</td>
<td>References MDAC1 code (0-255 range; x80 (128) = 0%</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC1_INTERCONNECT</td>
<td>18</td>
<td>References MDAC1 Input Source</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC2</td>
<td>9</td>
<td>References MDAC2 code</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC2_INTERCONNECT</td>
<td>19</td>
<td>References MDAC2 Input Source</td>
</tr>
</tbody>
</table>

## VREF Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_PACBLOCK_REF1LEVEL</td>
<td>26</td>
<td>References VREF1 level. Valid codes are: 0=64mV 4=1024mV 1=128mV 5=2048mV 2=256mV 6=2500mV 3=512mV 7=undefined</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_REF2LEVEL</td>
<td>27</td>
<td>References VREF2 level. Same codes as above</td>
</tr>
</tbody>
</table>
### IA/MDAC Input Source Interconnection Codes

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_CONN_NONE</td>
<td>0</td>
<td>Leave input unconnected (0V)</td>
</tr>
<tr>
<td>PAC30_CONN_IN1</td>
<td>1</td>
<td>Connect to IN1</td>
</tr>
<tr>
<td>PAC30_CONN_IN2</td>
<td>2</td>
<td>Connect to IN2</td>
</tr>
<tr>
<td>PAC30_CONN_IN3</td>
<td>3</td>
<td>Connect to IN3</td>
</tr>
<tr>
<td>PAC30_CONN_IN4</td>
<td>4</td>
<td>Connect to IN4</td>
</tr>
<tr>
<td>PAC30_CONN_OUT1</td>
<td>5</td>
<td>Connect to OUT1</td>
</tr>
<tr>
<td>PAC30_CONN_OUT2</td>
<td>6</td>
<td>Connect to OUT2</td>
</tr>
<tr>
<td>PAC30_CONN_REF</td>
<td>7</td>
<td>Connect to VREF</td>
</tr>
</tbody>
</table>

### OA Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_PACBLOCK_OA1_MODE</td>
<td>0</td>
<td>References OA1 operating mode. Valid settings are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-Amplifier/Filter, 2-Integrator, 3-Comparator</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_OA2_MODE</td>
<td>1</td>
<td>References OA2 operating mode. Uses setting values above</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_OA1_CAP</td>
<td>2</td>
<td>References OA1 Cap</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_OA2_CAP</td>
<td>3</td>
<td>References OA2 Cap</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_OA1PD</td>
<td>35</td>
<td>References OA1 Power-down mode control. 1=Powerdown/0=operate</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_OA2PD</td>
<td>36</td>
<td>References OA2 Power-down mode</td>
</tr>
</tbody>
</table>

### IA/MDAC to OA Summing Bus Interconnections

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_PACBLOCK_IA1_IATOOA</td>
<td>20</td>
<td>Reference IA1 output connection</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA2_IATOOA</td>
<td>21</td>
<td>Reference IA2 output connection</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA3_IATOOA</td>
<td>22</td>
<td>Reference IA3 output connection</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4_IATOOA</td>
<td>23</td>
<td>Reference IA4 output connection</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC1_IATOOA</td>
<td>24</td>
<td>Reference DAC1 output connection</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC2_IATOOA</td>
<td>25</td>
<td>Reference DAC2 output connection</td>
</tr>
<tr>
<td>PAC30_CONN_IATOOA_OA1</td>
<td>0</td>
<td>Set output connection to OA1 summing bus</td>
</tr>
<tr>
<td>PAC30_CONN_IATOOA_OA2</td>
<td>1</td>
<td>Set output connection to OA2 summing bus</td>
</tr>
</tbody>
</table>

### Miscellaneous Configuration Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC30_JTAGREG_ALL</td>
<td>-4</td>
<td>Download all configurations</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC1_CAL</td>
<td>37</td>
<td>Set Cal level for MDAC10=0V, 1=2.5V</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_DAC2_CAL</td>
<td>42</td>
<td>Set Cal level for MDAC1</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA1_CAL</td>
<td>38</td>
<td>Set Cal level for IA1</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA2_CAL</td>
<td>39</td>
<td>Set Cal level for IA2</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA3_CAL</td>
<td>40</td>
<td>Set Cal level for IA3</td>
</tr>
<tr>
<td>PAC30_PACBLOCK_IA4_CAL</td>
<td>41</td>
<td>Set Cal level for IA4</td>
</tr>
</tbody>
</table>
### ispPAC80/81 Constants

#### Bank A Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC80_C1A</td>
<td>0</td>
<td>References C1 Capacitor</td>
</tr>
<tr>
<td>PAC80_C2A</td>
<td>1</td>
<td>References C2 Capacitor</td>
</tr>
<tr>
<td>PAC80_L2A</td>
<td>2</td>
<td>References L2 Capacitor</td>
</tr>
<tr>
<td>PAC80_C3A</td>
<td>3</td>
<td>References C3 Capacitor</td>
</tr>
<tr>
<td>PAC80_C4A</td>
<td>4</td>
<td>References C4 Capacitor</td>
</tr>
<tr>
<td>PAC80_L4A</td>
<td>5</td>
<td>References L4 Capacitor</td>
</tr>
<tr>
<td>PAC80_C5A</td>
<td>6</td>
<td>References C5 Capacitor</td>
</tr>
<tr>
<td>PAC80_PM1A</td>
<td>7</td>
<td>References phase margin bit. Provides compensation for high frequencies (&gt;200kHz in ispPAC80, &gt;40kHz in ispPAC80) 1=ON, 0=OFF</td>
</tr>
</tbody>
</table>

#### Bank B Configuration

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC80_C1B</td>
<td>12</td>
<td>References C1 Capacitor</td>
</tr>
<tr>
<td>PAC80_C2B</td>
<td>13</td>
<td>References C2 Capacitor</td>
</tr>
<tr>
<td>PAC80_L2B</td>
<td>14</td>
<td>References L2 Capacitor</td>
</tr>
<tr>
<td>PAC80_C3B</td>
<td>15</td>
<td>References C3 Capacitor</td>
</tr>
<tr>
<td>PAC80_C4B</td>
<td>16</td>
<td>References C4 Capacitor</td>
</tr>
<tr>
<td>PAC80_L4B</td>
<td>17</td>
<td>References L4 Capacitor</td>
</tr>
<tr>
<td>PAC80_C5B</td>
<td>18</td>
<td>References C5 Capacitor</td>
</tr>
<tr>
<td>PAC80_PM1B</td>
<td>19</td>
<td>References phase margin bit. Provides compensation for high frequencies (&gt;200kHz in ispPAC80, &gt;40kHz in ispPAC80) 1=ON, 0=OFF</td>
</tr>
</tbody>
</table>

#### Miscellaneous ispPAC80 Control Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC80_ESF</td>
<td>9</td>
<td>References Electronic Security fuse 0=unsecured 1=secured</td>
</tr>
<tr>
<td>PAC80_JTAGREG_A</td>
<td>0</td>
<td>Reference 'A' Configuration register</td>
</tr>
<tr>
<td>PAC80_JTAGREG_CALIBRATE</td>
<td>1</td>
<td>Reference Calibrate Register (Factory Use Only)</td>
</tr>
<tr>
<td>PAC80_JTAGREG_B</td>
<td>2</td>
<td>Reference 'B' Configuration register</td>
</tr>
<tr>
<td>PAC80_JTAGREG_ALL</td>
<td>-4</td>
<td>Reference ALL registers</td>
</tr>
<tr>
<td>PAC80_PG</td>
<td>11</td>
<td>References Programmable Gain Amp. Setting Gain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 1 2 2 5 3 10</td>
</tr>
</tbody>
</table>