

# Application Note



## 5800 Series- PUC Triggers and Timers

Reference: AN507-02



The Aeroflex Integrated Development Environment (AIDE) provides a comprehensive programming platform capable of controlling a vast array of devices in addition to the 5800 Automatic Test Equipment (ATE) hardware.

These notes describe how general purpose Trigger and Timer hardware facilities on the Power and Utility Card (PUC) may be utilised to monitor, test and control devices on a Unit Under Test (UUT) from the AIDE in a .NET environment.

## Functionality

The AIDE provides facilities for carrying out Analogue In-Circuit, Functional and Digital Tests within the .NET Framework. Within the AIDE both the high level functionality such as a RESISTOR test and the low level functionality such as a basic Analogue to Digital Conversion (ADC) measurement are made available through a series of .NET Assemblies with the general reference name structure of Aeroflex.ATE.AIDE.assembly AIDE test programs also have access to particular hardware and software facilities via a number of system Built In Variables (see system documentation for further details).

With the \$System and \$Fixture built in variables, the user can access the full range of timers, trigger inputs and trigger outputs available on the PUC.

## PUC Hardware

The PUC has a set of 4 fully configurable hardware timers (Timer 0..3) with comprehensive input and output triggering facilities plus a general purpose system timer (Timer 4). PUC trigger facilities include PXI Trigger Bus Lines (8), Fixture Trigger Lines (4) and External Trigger Lines (4). The architecture is shown in the following diagram created from information in design documents for the Power and Utility Card PCB Specification (NBT-2141.rtf) and the Interface Rack Boards Memory Map Definition (NBT-2501.rtf).

## Trigger Input Modules

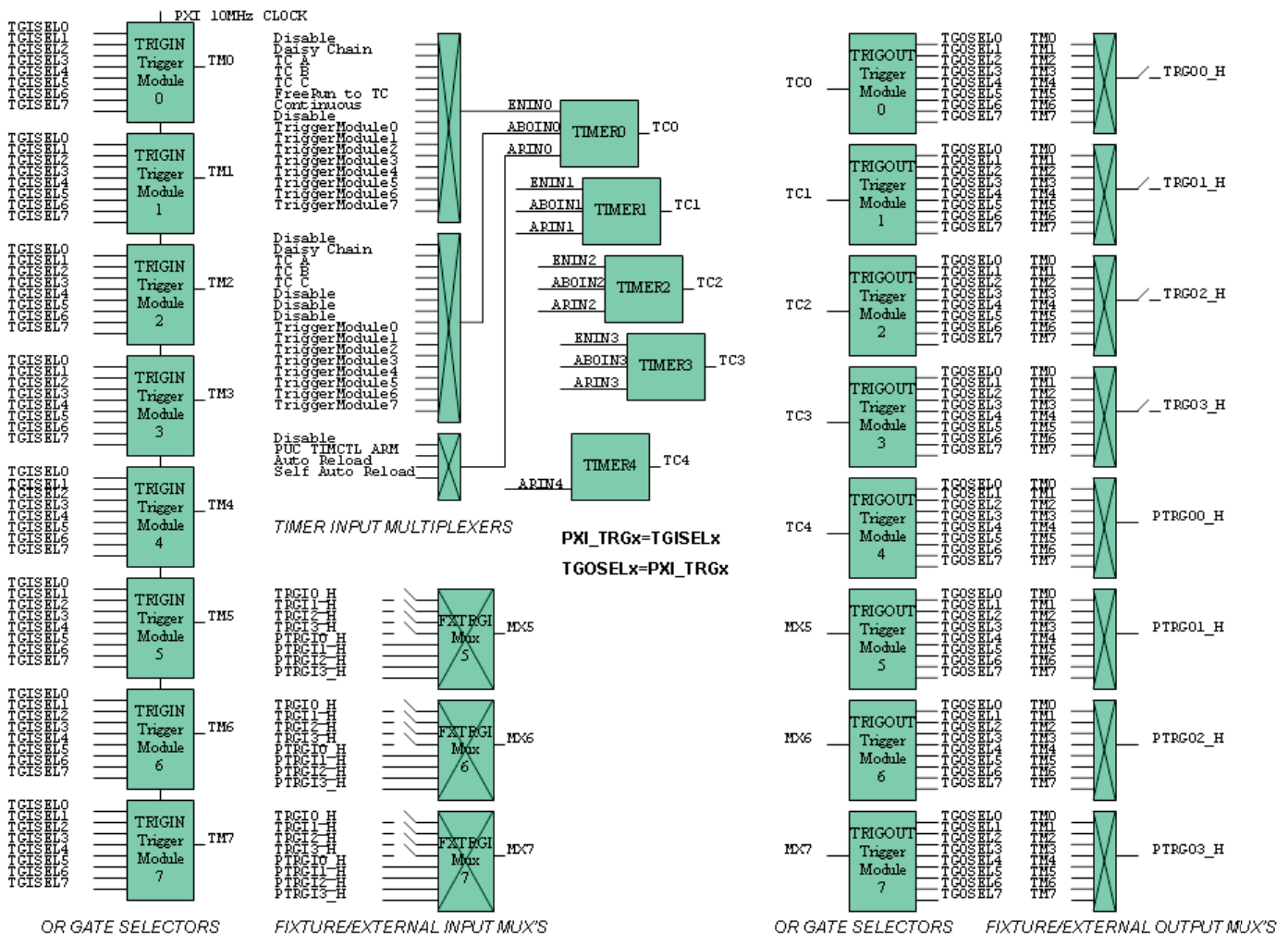
There are 8 Trigger Input Modules (0..7) with 8 separately ENABLED inputs FROM the PXI backplane trigger lines designated as TGISEL0..7 (corresponding to PXI\_TRG0..7). This allows any of the 8 PXI backplane trigger lines to generate an input trigger on any of the 8 Trigger Input Modules. Each module also has an input from the PXI 10 MHz CLOCK. When SYNCHRONOUS operation is selected the ENABLED inputs are GATED with the 10 MHz CLOCK.

## Trigger Output Modules

There are 8 Trigger Output Modules, each one associated with a particular trigger source, with 8 separately ENABLED outputs TO the PXI backplane trigger lines designated as TGOSEL0..7 (corresponding to PXI\_TRG0..7).

Trigger Output Modules 0 to 4 are directly connected to the TERMINAL COUNT (TC0..4) output of the corresponding Timer. This allows any of the Timers to generate an output trigger on any of the 8 PXI backplane trigger lines.

Trigger Output Modules 5 to 7 are directly connected to the corresponding Fixture and External Trigger Input Multiplexers designated as MX5..7. This allows any of the Fixture or External Trigger Inputs to generate an output trigger on any of the 8 PXI backplane trigger lines.



5800 PUC TIMER AND TRIGGER ARCHITECTURE

## Fixture and External Input Multiplexers

There are 3 Fixture and External Trigger Input Multiplexers, each one capable of selecting ONE of the 4 Fixture Input lines designated as TRGIO\_H to TRGI3\_H or ONE of the 4 External Input lines PTRGIO\_H to PTRGI3\_H. Note that the 4 Fixture Input lines have separate ENABLES which control their individual ISOLATION RELAYS.

## Fixture and External Output Multiplexers

There are 4 Fixture Output Multiplexers and 4 External Output Multiplexers, each one capable of selecting ONE of the 8 Trigger Input Modules designated as TM0..TM7 as the source of a trigger directed to the corresponding Fixture Output line TRGO0\_H to TRGO3\_H or External Output line PTRGO0\_H to PTRGO3\_H. Note that the 4 Fixture Output lines have separate ENABLES which control their individual ISOLATION RELAYS.

## Timer Input Multiplexers

The 4 fully configurable hardware timers each have 3 Input Multiplexers.

The Enable Input Multiplexer selects the appropriate Timer ENABLE INPUT source. This includes selection of any ONE of the Trigger Input Modules (TM0..7) or any ONE of the other 3 Timers TERMINAL COUNT outputs. The remaining selections determine whether the timer ENABLE input is Disabled (triggered by software only), Daisy chained, Free Running or Continuous.

The Abort Input Multiplexer selects the appropriate Timer ABORT INPUT source. This includes selection of any ONE of the Trigger Input Modules (TM0..7) or any ONE of the other 3 Timers TERMINAL COUNT outputs. The remaining selections determine whether the timer ABORT is Disabled (controlled by software only).

The Arm Input Multiplexer selects the appropriate Timer ARM INPUT source. This can be Disabled (armed by software only), Auto Reload, Self Auto Reload or Armed by the designated PUC Timer Master Control.

## Program Example 1 – Generating a Simple TIMEOUT using the PUC

Create a new Program called, for example, AIDetimers.

### Configuration Data

Expand the Configuration Data section of the program and if not already present, add the following .NET Assembly References from the .NET Global Assembly Cache (GAC) using the browse facilities provided :-

```
.NET Assembly Reference AssemblyFile:microsoft.dll
.NET Assembly Reference AssemblyFile:System
```

If not already present, add the following .NET Namespace References :-

```
.NET Namespace Reference System
```

## Main Method

Create a new Method called Main and reference it in the Program OnStart Property.

We can now create Evaluate statements that can be used to control the use of the PUC Timers and Triggers.

To Execute a SIMPLE TIMEOUT using the in-built \$System.Wait() Method, add the following to the Main Method :-

```
Evaluate $System.Wait(5s) Comment = "Wait for a 5s TIMEOUT period"
```

This command actually uses PUC Timer4. This hardware timer is also used by the system for some internal timeout functions and so it is not generally recommended that a programmer should utilise this timer other than with the \$System.Wait() Method.

To execute similar functionality using Timer0 add the following to the Main Method:-

```
Evaluate $System.PUC.Timers[0].Abort() Comment = "Abort any current timer activity"
Evaluate $System.PUC.Timers[0].InitialValue = 5000000 Comment = "set period to 5000000 us = 5s interval"
Evaluate $System.PUC.Timers[0].Resolution = Clock_1us Comment = "Select 1us input clock"
Evaluate $System.PUC.Timers[0].InputSelect = RunToTerminalCount Comment = "Run timer until count is ZERO"
Evaluate $System.PUC.Timers[0].AbortSelect = Disabled Comment = "Disable abort input"
Evaluate $System.PUC.Timers[0].ReArmSelect = Disabled Comment = "Disable re-arm input"
Evaluate $System.PUC.Timers[0].Arm() Comment = "Arm i.e. LOAD Count from InitialValue Register"
Evaluate $System.PUC.Timers[0].Enable = true Comment = "Start NOW - software trigger"
While ( !$System.PUC.Timers[0].Finished ) Comment = "Wait until timer count is ZERO"
```

This series of commands does not provide any more functionality than the previously used \$System.Wait() Method.

## Timeout Method

Create a new Method called Timeout that takes two parameters, the target timer and the period of the required delay :-

```
Method Timeout [] ( [ Int ] timer , [ Int ] period )
    Local Symbols
    Method Parameters
    Method Parameter [ Int ] timer Comment = "Timer (0..3)"
    Method Parameter [ Int ] period Comment = "Timeout period (1..4294967295)us"
Method Code
Evaluate $System.PUC.Timers[ timer ].Abort() Comment = "Abort any current timer activity"
Evaluate $System.PUC.Timers[ timer ].InitialValue = period Comment = "set period interval"
Evaluate $System.PUC.Timers[ timer ].Resolution = Clock_1us Comment = "Select 1us input clock"
Evaluate $System.PUC.Timers[ timer ].InputSelect = RunToTerminalCount Comment = "Run timer until
```

```

count is ZERO"
Evaluate $$System.PUC.Timers[ timer] .AbortSelect =
  Disabled Comment = "Disable abort input"
Evaluate $$System.PUC.Timers[ timer] .AbortSelect =
  Disabled Comment = "Disable abort input"
Evaluate $$System.PUC.Timers[ timer] .ReArmSelect =
  Disabled Comment = "Disable re-arm input"
Evaluate $$System.PUC.Timers[ timer] .Arm() Comment
  = "Arm i.e. LOAD Count from InitialValue
  Register"
Evaluate $$System.PUC.Timers[ timer] .Enable = true
  Comment = "Start NOW - software trigger"
While ( !$System.PUC.Timers[ timer] .Finished )
  Comment = "Wait until timer count is ZERO"

```

This Method can be used to set up and execute the required delay using the chosen timer.

```

Evaluate Timeout( 1 , 2000000 ) Comment = "Wait
  for 2s using timer1"

```

This timer routine is only marginally better than using \$\$System.Wait0, in that it merely allows the user to select a particular timer, however, if we remove the last statement of the Timeout0 Method to a second Method called Finished0 we can make better use of the facilities.

### Finished Method

Create a new Method called Finished that takes one parameter, the target timer :-

```

Method Finished [ ] ( [ Int ] timer)
  Local Symbols
  Method Parameters
  Method Parameter [ Int ] timer Comment = "Timer
  (0..3)"
Method Code
Return $$System.PUC.Timers[ timer] .Finished Comment
  = "Return timer FINISHED flag"

```

The timer routines can now be used together to allow other operations until the timer period expires.

```

Evaluate Timeout( 2 , 5000000 ) Comment = "Setup
  timeout for 5s using TIMER2"
While ( !Finished( 2 ) Comment = "Wait until
  TIMER2 count is ZERO"
  Evaluate Timeout( 1 , 500000 ) Comment = "Setup
  timeout for 500ms using TIMER1"
While ( !Finished( 1 ) Comment = "Wait until
  TIMER1 count is ZERO"

```

Here the program first sets up Timer 2, then, while waiting for Timer 2 to Finish, repeatedly sets up and monitors Timer 1 with a shorter interval until the Timer 2 period eventually expires.

## Program Example 2 – Using a Simple Interval Timer

The AIDE has an IntervalTimer Class that can be used to provide both simple time delays and the measurement of elapsed time.

### Local Symbols

Declare two Local Variables , one of type IntervalTimer , the other of type SITime as shown.

Local Symbols

Variables

```

Variable [ IntervalTimer ] intervalTimerA
Variable [ SITime ] intervalA

```

The interval timer can now be used to generate a simple timeout by execution of the following code.

```

Evaluate intervalTimerA.Trigger( 500ms ) Comment
  = "Setup and execute timeout for 500ms using
  intervalTimerA"
Evaluate ... Comment = "do something here!"
Evaluate intervalTimerA.WaitForExpiry() Comment =
  "Wait for intervalTimerA to reach 500ms (execu
  tion suspended until interval expires)"

```

The interval timer can also be used to mark and measure time intervals by execution of the following code.

```

Evaluate intervalTimerA.Mark() Comment = "Trigger
  intervalTimerA (mark start time)"
Evaluate Timeout( 0 , 2000000 ) Comment = "Setup
  timeout for 2s using TIMER0"
While ( !Finished( 0 ) Comment = "Wait until
  TIMER0 count is ZERO"
Evaluate intervalA = intervalTimerA.ElapsedTime
  Comment = "read current elapsed time of
  intervalTimerA"

```

The intervalA value returned should be approximately equal to 2s.

See the 5800 Documentation for 3rd party tools for more information.

Select the Search tab and enter IntervalTimer as the search string.

In the Select topic window, double click the IntervalTimer Class entry to view the description. Click the IntervalTimer Members reference to view the accessible Properties and Methods.

## Program Example 3 – Generating a Simple CLOCK using the PUC

Create a new Method called PXIClock.

A CLOCK signal can be generated on any of the 7 available PXI Trigger Lines. The allocation of pins might be done as follows.

Name	Function	PUC Signal	PUC/PSU/PXI Connection
CLK	Backplane Clock	TGOSELO	PXI_TRGO

To use TIMER 0 to generate a 100 KHz CLOCK on PXI Trigger Line 0 add the following lines of code.

```

Evaluate $$System.PUC.Timers[ 0] .Abort() Comment =
  "Abort any current timer 0 activity"
Evaluate $$System.PUC.Timers[ 0] .InitialValue = 10
  Comment = "set period to 10us"
Evaluate $$System.PUC.Timers[ 0] .Resolution =
  Clock_1us Comment = "Select 1us input clock"
Evaluate $$System.PUC.Timers[ 0] .InputSelect =
  Continuous Comment = "Run timer continuously"
Evaluate $$System.PUC.Timers[ 0] .AbortSelect =
  Disabled Comment = "Disable abort input"
Evaluate $$System.PUC.Timers[ 0] .ReArmSelect =
  SelfAutoReload Comment = "Enable re-arm input"
Evaluate $$System.PUC.Timers[ 0] .Arm() Comment =
  "Arm i.e. LOAD Count from InitialValue
  Register"

```

```
Evaluate
$System.PUC.Timers[ 0 ].TriggerOutputModule.
Select[ $System.PUC.PXITriggerLines[ 0 ] ] = true
Comment = "Enable TIMER 0 TC on to PXI Trigger
Line 0"
```

```
Evaluate $System.PUC.TriggerLines[ 0 ].
BufferEnable = true Comment = "Enable PXI
Trigger Line 0 output on to the PXI Backplane"
```

```
Evaluate $System.PUC.TriggerLines[ 0 ].Drive =
Normal Comment = "Set PXI Trigger Line 0 output
to Normal i.e. drives signal level from the con
figured TriggerOutputModule"
```

```
Evaluate $System.PUC.Timers[ 0 ].Enable = true
Comment = "Start NOW - software trigger"
```

## Program Example 4 – Routing Fixture Lines

Create a new Method called FixtureLineRouting.

The 4 Fixture Input Lines can be multiplexed on to any of the 8 PXI Trigger Lines (Note the presence of fixture ISOLATION RELAYS - see diagram).

The 8 PXI Trigger Lines can be multiplexed on to any of the 4 Fixture Output Lines (Note the presence of fixture ISOLATION RELAYS - see diagram).

To switch FixtureLine0(TRGIO\_H) on to PXITriggerLine0(TGOSELO) we need to utilise one of the 3 Fixture/External Input Multiplexers (MX5, MX6 or MX7). To achieve this, add the following lines of code which use Input Multiplexer 5.

Evaluate

```
$Fixture.TriggerInputSignalEnable[ [ Int]
PUC.TriggerSignalSelect.FixtureLine0].Enable =
true Comment = "Enable INPUT of
FixtureLine0(TRGIO H) from PUC pinface (i.e.
CLOSE the INPUT ISOLATION RELAY)"
```

Evaluate

```
System.PUC.TriggerInputSignalSelect[ 5 ].Select =
PUC.TriggerSignalSelect.FixtureLine0 Comment =
"Select FixtureLine0(TRGIO_H) as INPUT on MX5"
```

Evaluate

```
$System.PUC.TriggerInputSignalSelect[ 5 ].Trigger
OutputModule.Select[ $System.PUC.PXITriggerLines
[ 0 ] ] = true Comment = "Enable OUTPUT of MX5 on
PXITriggerLine0"
```

```
Evaluate $System.PUC.TriggerLines[ 0 ].
BufferEnable = true Comment = "Enable
PXITriggerLine0(TGOSELO) output on to the PXI
Backplane"
```

```
Evaluate $System.PUC.TriggerLines[ 0 ].Drive =
Normal Comment = "Set PXITriggerLine 0 output
to Normal i.e. drive level from configured
TriggerOutputModule"
```

To switch PXITriggerLine7 on to FixtureLine1 we need to utilize one of the 8 Trigger Input Selectors (TM0..TM7). To achieve this, add the following lines of code which uses Trigger Input Module 1 (TM1).

```
Evaluate $Fixture.TriggerOutputSignalEnable[
[ Int] PUC.TriggerSignalSelect.FixtureLine1].
Enable = true Comment = "Enable OUTPUT of
FixtureLine1 to PUC pinface (i.e. CLOSE the
OUTPUT ISOLATION RELAY)"
```

```
Evaluate System.PUC.TriggerOutputSignalSelect[
FixtureLine1 ] =
$System.PUC.TriggerInputModules[ 1 ] Comment =
"Select TM1 as OUTPUT on FixtureLine1"
```

Evaluate

```
$System.PUC.TriggerInputModules[ 1 ].Select[ $
System.PUC.PXITriggerLines[ 7 ] ] = true Comment =
"Enable INPUT of PXITriggerLine7 on TM1"
```

```
Evaluate $System.PUC.TriggerInputModules[ 1 ].
InputMode = Direct Comment = "Enable direct
INPUT on TM1"
```

## Program Example 5 – Routing External Lines

Create a new Method called ExternalLineRouting.

The 4 External Input Lines can be multiplexed on to any of the 8 PXI Trigger Lines (Note there are no ISOLATION RELAYS on these lines - see diagram).

The 8 PXI Trigger Lines can be multiplexed on to any of the 4 External Output Lines (Note there are no ISOLATION RELAYS on these lines - see diagram).

To switch ExternalLine3(PTRGI3\_H) on to PXITriggerLine5 we need to utilise one of the 3 Fixture/External Input Multiplexers (MX5, MX6 or MX7). To achieve this, add the following lines of code which use Input Multiplexer 6.

Evaluate

```
$System.PUC.TriggerInputSignalSelect [ 6 ].Select
= PUC.TriggerSignalSelect.ExternalLine 3
Comment = "Select ExternalLine3 (PTRGI3_H) as
INPUT on MX6"
```

Evaluate

```
$System.PUC.TriggerInputSignalSelect[ 6 ].
Trigger OutputModule.Select[ $System.PUC.PXI
TriggerLines [ 5 ] ] = true Comment = "Enable
OUTPUT of MX6 on PXITriggerLine5 (TGOSEL5)"
```

Evaluate

```
$System.PUC.TriggerLines[ 5 ].BufferEnable = true
Comment = "Enable PXITriggerLine5(TGOSEL5)
OUTPUT on to the PXI Backplane"
```

Evaluate

```
$System.PUC.TriggerLines[ 5 ].Drive = Normal
Comment = "Set PXI Trigger Line 5 output to
Normal i.e. drives signal level from the con
figured TriggerOutputModule"
```

To switch PXITriggerLine4 on to ExternalLine2 synchronised with the PXI 10 MHz CLOCK we need to utilise one of the 8 Trigger Input Selectors (TM0..TM7). To achieve this, add the following lines of code which use Trigger Input Module 3 (TM3).

Evaluate

```
$System.PUC.TriggerInputModules[ 3 ].Select[ $
System.PUC.PXITriggerLines[ 4 ] ] = true Comment
= "Enable INPUT of PXITriggerLine4(TGISEL4) on
TM3"
```

```
$System.PUC.TriggerInputModules[ 3 ].InputMode =
Synchronous Comment = "Enable SYNCHRONOUS
INPUT on TM3 i.e. gated with the 10MHz PXI
CLOCK"
```

Evaluate

```
$System.PUC.TriggerOutputSignalSelect[ External
Line2 ] = $System.PUC.TriggerInputModules[ 3 ]
Comment = "Select TM3 as OUTPUT on
ExternalLine2(PTRGO2_H)"
```

## Program Example 6 – Routing External Lines To Fixture Lines

Create a new Method called ExternalToFixtureLineRouting.

The 4 External Input Lines and the 4 Fixture Input Lines can be multiplexed on to any of the 8 PXI Trigger Lines.

The 8 PXI Trigger Lines can be multiplexed on to any of the 4 External Output Lines and any of the 4 Fixture Output Lines.

The External Input Lines cannot be directly multiplexed on to the Fixture Output Lines nor the Fixture Input Lines directly multiplexed on to the External Output Lines, however, by using the internal PXI Trigger Line routing that is available, the required cross-trigger connection path can be created.

To switch ExternalLine0 on to FixtureLine0 we need to utilise one of the 3 Fixture/External Input Multiplexers (MX5, MX6 or MX7), one of the Trigger Input Modules (TM0..TM7) and a PXI Trigger Line. To achieve this, add the following lines of code which use Input Multiplexer 5 (MX5), Trigger Input Module 5 (TM5) and PXI Trigger Line 5 (TGOSEL5 & TGISEL5).

```
Evaluate
[System.PUC.TriggerInputSignalSelect[ 5] .Select
= PUC.TriggerSignalSelect.ExternalLine0
Comment = "Select ExternalLine0 (PTRGIO_H) as
INPUT on MX5"

Evaluate
[System.PUC.TriggerInputSignalSelect[ 5] .Trigger
OutputModule.Select[ $System.PUC.PXITriggerLines
[ 5]] = true Comment = "Enable OUTPUT of MX5 on
PXITriggerLine5 (TGOSEL5)"

Evaluate
[System.PUC.TriggerLines[ 5] .BufferEnable = true
Comment = "Enable PXITriggerLine5(TGOSEL5)
READBACK on to the PXI Backplane"

Evaluate
[System.PUC.TriggerLines[ 5] .Drive = Normal
Comment = "Set PXI Trigger Line 5 output to
Normal i.e. drives signal level from the con
figured TriggerOutputModule"

Evaluate
[System.PUC.TriggerInputModules[ 5] .InputMode =
Direct Comment = "Enable DIRECT INPUT on TM5"

Evaluate
[System.PUC.TriggerOutputSignalSelect[ Fixture
Line0] = $System.PUC.TriggerInputModules[ 5]
Comment = "Select TM5 as OUTPUT on
FixtureLine2 (TRGO0_H)"

Evaluate
[Fixture.TriggerOutputSignalEnable[ [ Int]
PUC.TriggerSignalSelect.FixtureLine0] .Enable =
true Comment = "Enable OUTPUT of FixtureLine0
to PUC pinface i.e. close the isolating relay"
```

Similar code can be created to route a FixtureLine to an ExternalLine.

### Libraries

It becomes much simpler as projects and solutions develop to place all the Methods that might be shared into appropriate Libraries. For example the above methods associated with using the PUC Timers and Trigger facilities can easily be placed in a PUCLibrary, in Modules called TIMER, TRIGGER etc. Accessing these Methods simply requires the Library to be imported into the user's Program and the Methods referenced by statements of the form :-

```
Evaluate PUCLibrary.TIMER.Timeout(timer, period)
```

```
Evaluate PUCLibrary.TIMER.Clock(timer, period,
output)
```

```
Evaluate PUCLibrary.TRIGGER.Route(input, output)
```

A preliminary version of the PUC Library is currently under development and may be obtained from Aeroflex Customer Support.

### Summary

It is relatively complex to utilise the Timer and Trigger facilities on the Power and Utility Card (PUC). The above methods provide some simplified examples of how the hardware may be utilised. Where more complex configurations are required, the above routines may be treated as templates from which alternative set-ups may be derived.

For the very latest specifications visit [www.aeroflex.com](http://www.aeroflex.com)

**CHINA Beijing**

Tel: [+86] (10) 6539 1166  
Fax: [+86] (10) 6539 1778

**CHINA Shanghai**

Tel: [+86] (21) 5109 5128  
Fax: [+86] (21) 5150 6112

**CHINA Shenzhen**

Tel: [+86] (755) 3301 9358  
Tel: [+86] (755) 3301 9356

**FINLAND**

Tel: [+358] (9) 2709 5541  
Fax: [+358] (9) 804 2441

**FRANCE**

Tel: [+33] 1 60 79 96 00  
Fax: [+33] 1 60 77 69 22

**GERMANY**

Tel: [+49] 89 99641 0  
Fax: [+49] 89 99641 160

**HONG KONG**

Tel: [+852] 2832 7988  
Fax: [+852] 2834 5364

**INDIA**

Tel: [+91] 80 [4] 115 4501  
Fax: [+91] 80 [4] 115 4502

**JAPAN**

Tel: [+81] (3) 3500 5591  
Fax: [+81] (3) 3500 5592

**KOREA**

Tel: [+82] (2) 3424 2719  
Fax: [+82] (2) 3424 8620

**SCANDINAVIA**

Tel: [+45] 9614 0045  
Fax: [+45] 9614 0047

**SINGAPORE**

Tel: [+65] 6873 0991  
Fax: [+65] 6873 0992

**UK Stevenage**

Tel: [+44] (0) 1438 742200  
Fax: [+44] (0) 1438 727601  
Freephone: 0800 282388

**USA**

Tel: [+1] (316) 522 4981  
Fax: [+1] (316) 522 1360  
Toll Free: 800 835 2352

As we are always seeking to improve our products, the information in this document gives only a general indication of the product capacity, performance and suitability, none of which shall form part of any contract. We reserve the right to make design changes without notice. All trademarks are acknowledged. Parent company Aeroflex, Inc. ©Aeroflex 2011.

[www.aeroflex.com](http://www.aeroflex.com)  
[info-test@eroflex.com](mailto:info-test@eroflex.com)



Our passion for performance is defined by three attributes represented by these three icons: solution-minded, performance-driven and customer-focused.