PCIE-A04-DIO64-16RS485-CT 10
Reference Manual

PCIE-A04-DIO64-16RS485-CT10 Reference Manual

Quad Analog Outputs, 64 digital I/O, 10 counter/timers and 16 RS485

Apex Embedded Systems

116 Owen Road

Monona, WI 53716

Phone 608.256.0767

www.apexembeddedsystems.com

customer.service@apexembeddedsystems.com

Table of Contents

Register Set 27

PCIE-A04-DIO64-16RS485-CT10 Reference Manual

1 Welcome

Dear Valued Customer:

Thank you for your interest in our products and services.

Apex Embedded Systems "Continuous improvement" policy utilizes customer feedback to improve existing products and create new product offerings based on the needs of our customers.

Continued Success,

Apex Embedded Systems

2 Legal Notice

Apex Embedded Systems' sole obligation for products that prove to be defective within 1 year from date of purchase will be for replacement or refund. Apex Embedded Systems gives no warranty, either expressed or implied, and specifically disclaims all other warranties, including warranties for merchantability and fitness. In no event shall Apex Embedded Systems' liability exceed the buyer's purchase price, nor shall Apex Embedded Systems be liable for any indirect or consequential damages.

This warranty does not apply to products which have been subject to misuse (including static discharge), neglect, accident or modification, or which have been soldered or altered during assembly and are not capable of being tested.

DO NOT USE PRODUCTS SOLD BY APEX EMBEDDED SYSTEMS AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS!

Products sold by Apex Embedded Systems are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

3 ESD Caution

A discharge of static electricity from your hands can seriously damage certain electrical components on any circuit board. Before handling any board, discharge static electricity from yourself by touching a grounded conductor such as your computer chassis (your computer must be turned off). Whenever you handle a board, hold it by the edges and avoid touching any board components or cable connectors.

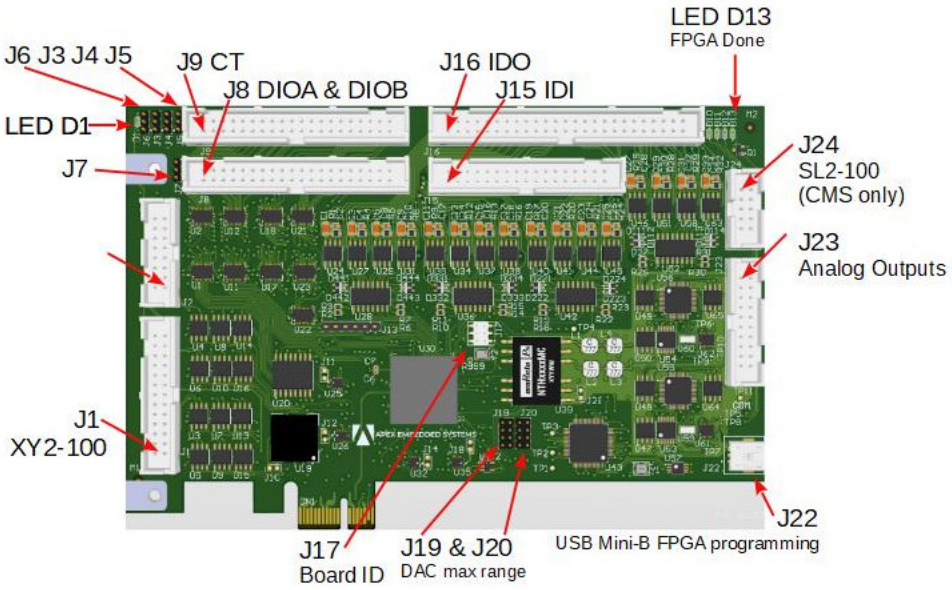
4 Definitions

Definitions

Component	Refers to an internal FPGA building block. Each building block will have its own unique ID

5 Board Connector and LED positions

Summary of Connector positions and LED status information



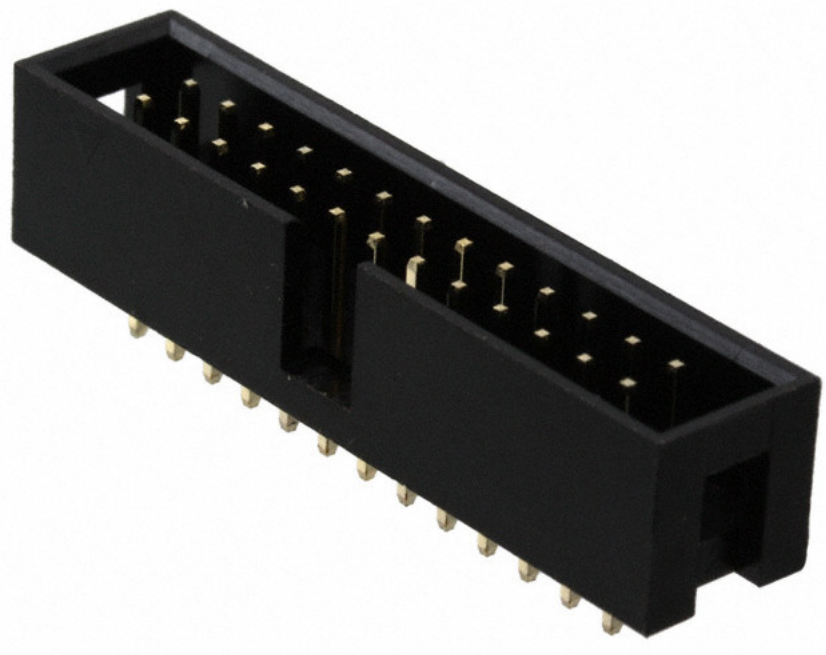
Notes

1. Connector designators are nearest pin one
2. Pin numbers for connectors on bottom side
3. LED D13 will be lit if FPGA is successfully programmed.
4. LED D1 will light upon any valid read/write cycle to a valid register location (same as STX104 board)
5. Board (see page 119) ID can be found in the SYS.BRD_ID register and is used by software to discern two or more boards in system. Default value is zero.
6. J19 and J20 set a maximum range for the analog outputs. These headers are identical to STX104 analog output jumper settings, but used to set a maximum output range.
7. Programming connector J22 is used to download bit image updates to the FPGA.

6 Connectors

6.1 XY2-100/200 (XY2, J1)

Reference Designator	J1
Manufacture	On Shore Technology Inc.
Manufacture Part Number	On Shore Technology Inc: 302-S261 TE: 5103308-6
Description	CONN 26-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Input	RS422 differential type, -7 to +12 volts. 200mV hysteresis on input. Terminated with 1Kohm resistor.
Outputs	RS422 differential type. Terminated with 1Kohm resistor.



```
SENDCK  =  Transmit clock or CLK

26-pin connector can be connected directly to DB25 and be compatible with other XY2-100
equipment with 25-pin D-SUB connectors
```

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
SENDCK-	1	2	SENDCK+
SYNC-	3	4	SYNC+
CHANNELX-	5	6	CHANNELX+
CHANNELY-	7	8	CHANNELY+
CHANNELZ-	9	10	CHANNELZ+
STATUS-	11	12	STATUS+
no connect	13	14	no connect
no connect	15	16	no connect
no connect	17	18	no connect
no connect	19	20	COM
COM	21	22	COM
no connect	23	24	no connect
no connect	25	26	no connect

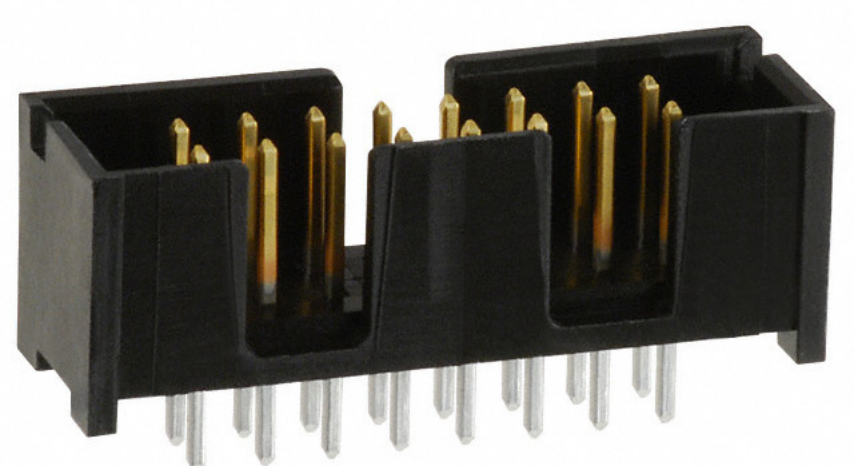
DB25 Connector pinout.

26-pin IDC <--> ribbon cable <--> 25-pin female D-SUB

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
SENDCK-	1	14	SENDCK+
SYNC-	2	15	SYNC+
CHANNELX-	3	16	CHANNELX+
CHANNELY-	4	17	CHANNELY+
CHANNELZ-	5	18	CHANNELZ+
STATUS-	6	19	STATUS+
no connect	7	20	COM
no connect	8	21	COM
no connect	9	22	no connect
no connect	10	23	no connect
COM	11	24	COM
no connect	12	25	no connect
no connect	13		

6.2 Quadrature Inputs (QEA & QEB, J2)

Reference Designator	J2
Manufacture	TE Connectivity AMP Connectors (see page 11)
Manufacture Part Number	5103308-3
Description	CONN 16-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Inputs	RS422 differential type, -7 to +12 volts. 200mV hysteresis on input. All inputs terminated with 1Kohm resistor.

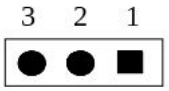


```
.A_P   = A+ or X+ input
.A_N   = A- or X- input
.B_P   = B+ or Y+ input
.B_N   = B- or Y- input
.Z_P   = Z+ or Index+ input
.Z_N   = Z- or Index- input
```

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
COM	1	2	+5V (current limited to 500mA)
QEA.Z_P	3	4	QEA.Z_N
QEA.A_P	5	6	QEA.A_N
QEA.B_P	7	8	QEA.B_N
COM	9	10	+5V (current limited to 500mA)
QEB.Z_P	11	12	QEB.Z_N
QEB.A_P	13	14	QEB.A_N
QEB.B_P	15	16	QEB.B_N

6.3 Digital Inputs and Outputs Pull Up/Down (DIOA, J3)

Reference Designator J3

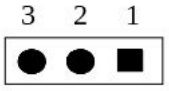


JUMPER POSITION	FUNCTION
1 - 2 *	All inputs/outputs are pulled-up
2 - 3	All inputs/outputs are pulled-down.

* Factory configured default.

6.4 Digital Inputs and Outputs Bias (DIOA, J4)

Reference Designator J4

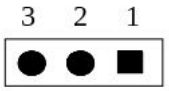


JUMPER POSITION	FUNCTION
1 - 2 *	Outputs drive 0 to 5V
2 - 3	Outputs drive 0 to 3.3V

* Factory configured default.

6.5 Counter Timer I/O Bias (CT, J5)

Reference Designator J4

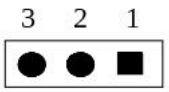


JUMPER POSITION	FUNCTION
1 - 2 *	Outputs drive 0 to 5V
2 - 3	Outputs drive 0 to 3.3V

* Factory configured default.

6.6 Digital Inputs and OUtputs Pull Up/Down (DIOB, J6)

Reference Designator J6

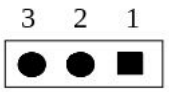


JUMPER POSITION	FUNCTION
1 - 2 *	All inputs/outputs are pulled-up
2 - 3	All inputs/outputs are pulled-down.

* Factory configured default.

6.7 Digital Inputs and Outputs Bias (DIOB, J7)

Reference Designator J7

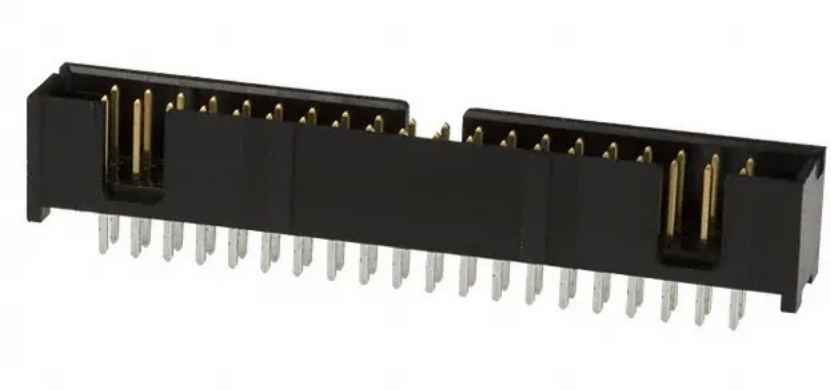


JUMPER POSITION	FUNCTION
1 - 2 *	Outputs drive 0 to 5V
2 - 3	Outputs drive 0 to 3.3V

* Factory configured default.

6.8 TTL Digital Inputs and Outputs (DIOA & DIOB, J8)

Reference Designator	J7
Manufacture	TE Connectivity AMP Connectors (see page 11)
Manufacture Part Number	5103308-8
Description	CONN 40-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Inputs / Outputs	LVTTL/TTL compatible. Output levels can be set for 5V or 3.3V range. 3.3V range, inputs are 5V tolerant.



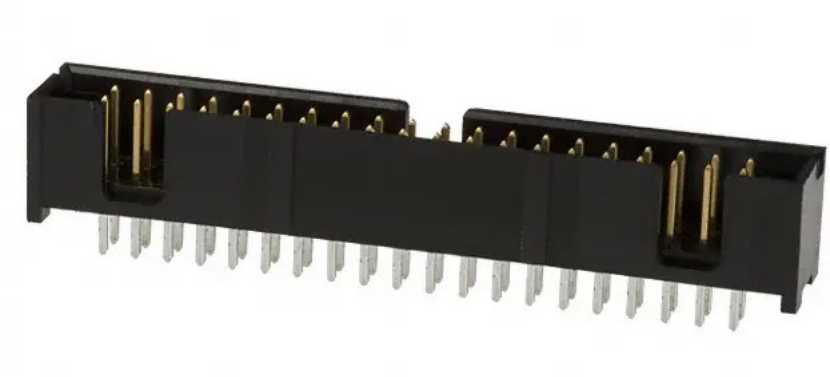
Note that the pinout for this connector is of a similar pattern of the Counter Timer Array Connector so that a mis-insertion does not result in board damage.

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
COM	1	2	COM
DIOA(0)	3	4	DIOA(1)
DIOA(2)	5	6	DIOA(3)
DIOA(4)	7	8	DIOA(5)
DIOA(6)	9	10	DIOA(7)
COM	11	12	COM
DIOA(8)	13	14	DIOA(9)
DIOA(10)	15	16	DIOA(11)
DIOA(12)	17	18	DIOA(13)
DIOA(14)	19	20	DIOA(15)
COM	21	22	COM

DIOB(0)	23	24	DIOB(1)
DIOB(2)	25	26	DIOB(3)
DIOB(4)	27	28	DIOB(5)
DIOB(6)	29	30	DIOB(7)
COM	31	32	COM
DIOB(8)	33	34	DIOB(9)
DIOB(10)	35	36	DIOB(11)
DIOB(12)	37	38	DIOB(13)
DIOB(14)	39	40	DIOB(151)

6.9 Counter Timer Array (CT, J9)

Reference Designator	J9
Manufacture	TE Connectivity AMP Connectors (see page 11)
Manufacture Part Number	5103308-8
Description	CONN 40-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Inputs / Outputs	LVTTL/TTL compatible. Output levels can be set for 5V or 3.3V range. 3.3V range, inputs are 5V tolerant.



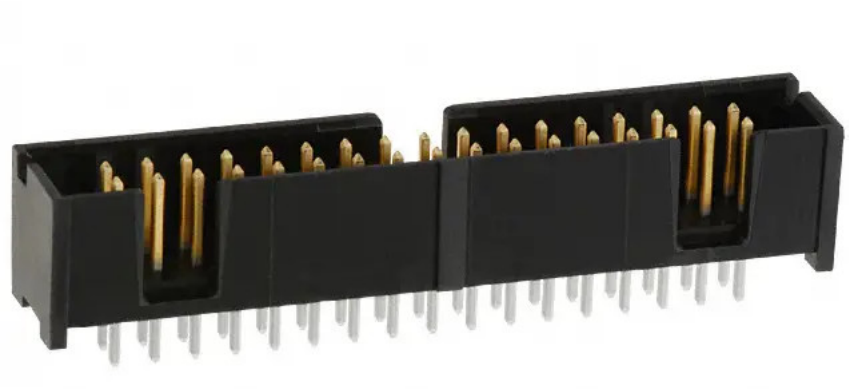
```
CT.SRC  = counter/timer source (A-input)
CT.GAT  = counter/timer gate   (B-input)
CT.OUT  = counter/timer out    (PWM, Pulses, etc)
COM     = system common or ground

Note that the pinout for this connector is of a similar pattern of the TTL digital inputs and
outputs in order that a mis-insertion does not result in board damage.
```

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
COM	1	2	COM
CT.SRC(0)	3	4	CT.SRC(1)
CT.SRC(2)	5	6	CT.SRC(3)
CT.SRC(4)	7	8	CT.SRC(5)
CT.SRC(6)	9	10	CT.SRC(7)
COM	11	12	COM
CT.SRC(8)	13	14	CT.SRC(9)
CT.GAT(0)	15	16	CT.GAT(1)
CT.GAT(2)	17	18	CT.GAT(3)
CT.GAT(4)	19	20	CT.GAT(5)
COM	21	22	COM
CT.GAT(6)	23	24	CT.GAT(7)
CT.GAT(8)	25	26	CT.GAT(9)
reserved	27	28	reserved
CT.OUT(0)	29	30	CT.OUT(1)
COM	31	32	COM
CT.OUT(2)	33	34	CT.OUT(3)
CT.OUT(4)	35	36	CT.OUT(5)
CT.OUT(6)	37	38	CT.OUT(7)
CT.OUT(8)	39	40	CT.OUT(9)

6.10 Isolated Digital Inputs (IDI, J15)

Reference Designator	J15
Manufacture	TE Connectivity AMP Connectors (see page 11)
Manufacture Part Number	5103308-7
Description	CONN 34-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Inputs	Differential pairs, isolated. Input voltage range +/-100V.



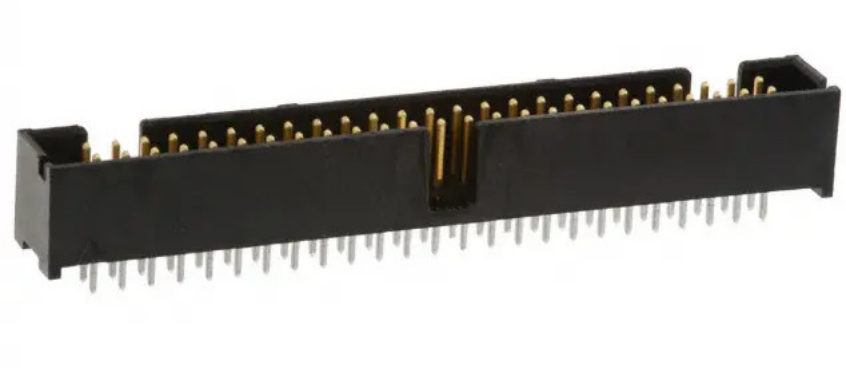
Note: connector pinout identical connector P2 on the AccessIO 104-IDIO-16

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
DI.ISO.A(0)	1	2	DI.ISO.B(0)
DI.ISO.A(1)	3	4	DI.ISO.B(1)
DI.ISO.A(2)	5	6	DI.ISO.B(2)
DI.ISO.A(3)	7	8	DI.ISO.B(3)
DI.ISO.A(4)	9	10	DI.ISO.B(4)
DI.ISO.A(5)	11	12	DI.ISO.B(5)
DI.ISO.A(6)	13	14	DI.ISO.B(6)
DI.ISO.A(7)	15	16	DI.ISO.B(7)
	17	18	
DI.ISO.A(8)	19	20	DI.ISO.B(8)
DI.ISO.A(9)	21	22	DI.ISO.B(9)
DI.ISO.A(10)	23	24	DI.ISO.B(10)
DI.ISO.A(11)	25	26	DI.ISO.B(11)
DI.ISO.A(12)	27	28	DI.ISO.B(12)
DI.ISO.A(13)	29	30	DI.ISO.B(13)
DI.ISO.A(14)	31	32	DI.ISO.B(14)
DI.ISO.A(15)	33	34	DI.ISO.B(15)

6.11 Isolated Digital Outputs (IDO, J16)

Reference Designator	J16
Manufacture	TE Connectivity AMP Connectors (see page 11)

Manufacture Part Number	1-5103308-0
Description	CONN 50-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Outputs	Each channel isolated. Outputs 5V to 30V to 1 Amp each.



```
IPDO.SVP = Isolated power digital output supply voltage positive
IPDO.SVN = Isolated power digital output supply voltage negative or floating common
IPDO.OUT = Isolated power digital output high-side switch output.

Note: connector pinout is the same as P1 on the AccessIO 104-IDIO-16 from pins 1
through 41, 42 through 50 are different due to issues meeting PCB route completion
and voltage isolation requirements.
```

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
IPDO.SVP(15)	1	2	IPDO.SVN(15)
IPDO.OUT(15)	3	4	IPDO.SVP(14)
IPDO.SVN(14)	5	6	IPDO.OUT(14)
IPDO.SVP(13)	7	8	IPDO.SVN(13)
IPDO.OUT(13)	9	10	IPDO.SVP(12)
IPDO.SVN(12)	11	12	IPDO.OUT(12)
IPDO.SVP(11)	13	14	IPDO.SVN(11)
IPDO.OUT(11)	15	16	IPDO.SVP(10)
IPDO.SVN(10)	17	18	IPDO.OUT(10)
IPDO.SVP(9)	19	20	IPDO.SVN(9)
IPDO.OUT(9)	21	22	IPDO.SVP(8)
IPDO.SVN(8)	23	24	IPDO.OUT(8)
	25	26	
IPDO.SVP(7)	27	28	IPDO.SVN(7)
IPDO.OUT(7)	29	30	IPDO.SVP(6)
IPDO.SVN(6)	31	32	IPDO.OUT(6)
IPDO.SVP(5)	33	34	IPDO.SVN(5)
IPDO.OUT(5)	35	36	IPDO.SVP(4)

IPDO.SVN(4)	37	38	**IPDO.OUT(4)**
IPDO.SVP(3)	39	40	**IPDO.SVN(3)**
IPDO.OUT(3)	41	42	**IPDO.SVN(2)**
IPDO.SVP(2)	43	44	**IPDO.OUT(2)**
IPDO.SVP(1)	45	46	**IPDO.SVN(1)**
IPDO.OUT(1)	47	48	**IPDO.SVN(0)**
IPDO.SVP(0)	49	50	**IPDO.OUT(0)**

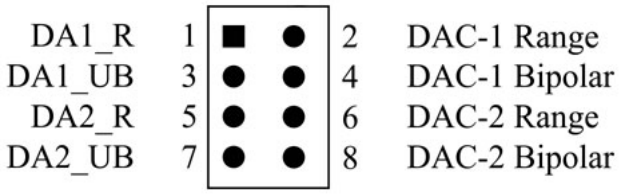
6.12 Analog Outputs Maximum Range Settings (AO CHC & CHD, J19)]

Analog Outputs Maximum Range Settings

The actual range of the analog output is set in software. The jumper configuration below limits that software range.

Reference Designator J19

TODO: update image



S/W	AO.CHC.UB	AO.CHC.R	AO CHANNEL-C, MAXIMUM ALLOWABLE RANGE
00	0	0	0 to +5 Volts *
01	0	1	0 to +10 Volts
10	1	0	-5 to +5 Volts
11	1	1	-10 to 10 Volts

* Factory Default

Note: 1 = Jumper installed, 0 = Jumper not installed.

S/W	AO.CHD.UB	AO.CHD.R	AO CHANNEL-D, MAXIMUM ALLOWABLE RANGE
00	0	0	0 to +5 Volts *
01	0	1	0 to +10 Volts
10	1	0	-5 to +5 Volts
11	1	1	-10 to 10 Volts

* Factory Default

Note: 1 = Jumper installed, 0 = Jumper not installed.

Example

Limit maximum allowable ranges to only unipolar outputs at analog output channel-C.

1. Install jumper at position AO.CHC.R.
2. Software can set overall range to either 00 or 01. If "10" or "11" used, it will be replaced with "01".

The values of .UB and .R jumpers can be read via a status register which allows for software implemented alternatives.

6

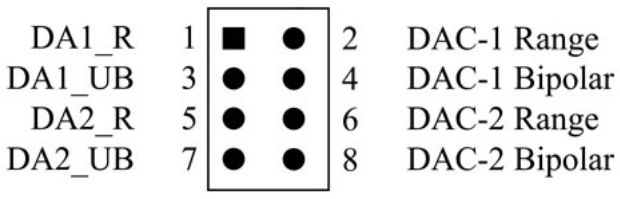
6.13 Analog Outputs Maximum Range Settings (AO CHA & CHB, J20)

Analog Outputs Maximum Range Settings

The actual range of the analog output is set in software. The jumper configuration below limits that software range.

Reference Designator J20

TODO: update image



S/W	AO.CHA.UB	AO.CHA.R	AO CHANNEL-A, MAXIMUM ALLOWABLE RANGE
00	0	0	0 to +5 Volts *
01	0	1	0 to +10 Volts
10	1	0	-5 to +5 Volts
11	1	1	-10 to 10 Volts

* Factory Default

Note: 1 = Jumper installed, 0 = Jumper not installed.

S/W	AO.CHB.UB	AO.CHB.R	AO CHANNEL-B, MAXIMUM ALLOWABLE RANGE
00	0	0	0 to +5 Volts *
01	0	1	0 to +10 Volts
10	1	0	-5 to +5 Volts
11	1	1	-10 to 10 Volts

* Factory Default

Note: 1 = Jumper installed, 0 = Jumper not installed.

Example

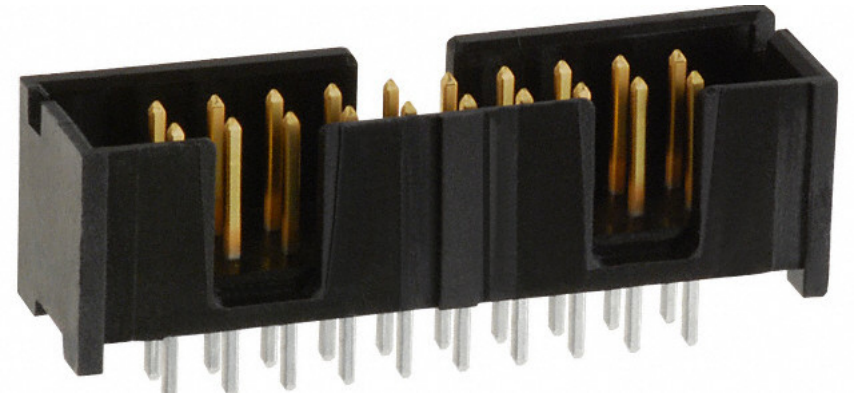
Limit maximum allowable ranges to only unipolar outputs at analog output channel-A.

1. Install jumper at position AO.CHA.R.
2. Software can set overall range to either 00 or 01. If "10" or "11" used, it will be replaced with "01".

The values of .UB and .R jumpers can be read via a status register which allows for software implemented alternatives.

6.14 Analog Outputs (AO, J23)

Reference Designator	J23
Manufacture	TE Connectivity AMP Connectors (see page 11)
Manufacture Part Number	5103308-5
Description	CONN 20-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Outputs	Single-Ended, 0-5V, 0-10V,



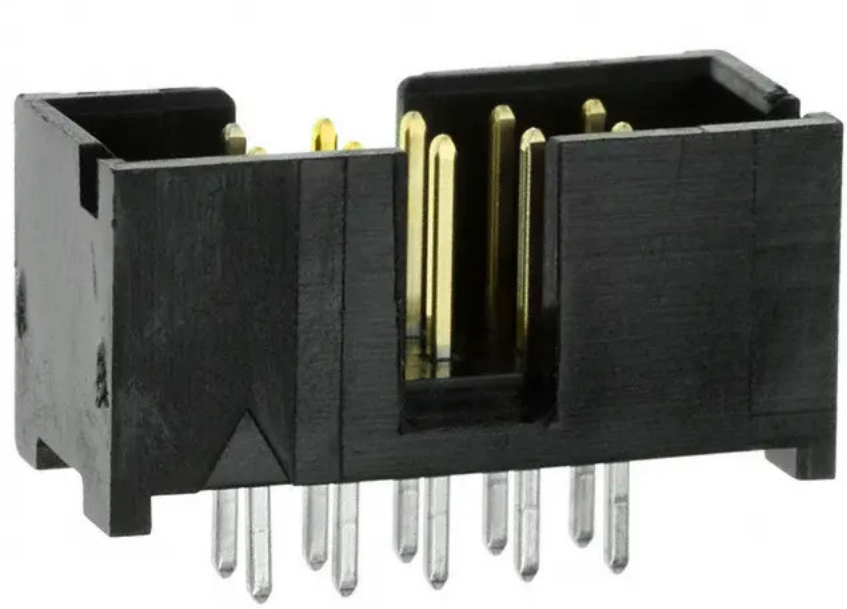
```
AO_P  = analog input positive lead input
AO_N  = analog output negative lead input (Same as ACOM) (*)
ACOM  = analog common (basically common of computer)

(*) Note:  We have setup the connector for future compatibility to support pseudo differential
outputs to handle equipment ground differences.
```

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
AO_P(0)	1	2	ACOM
AO_N(0)	3	4	ACOM
AO_P(1)	5	6	ACOM
AO_N(1)	7	8	ACOM
AO_P(2)	9	10	ACOM
AO_N(2)	11	12	ACOM
AO_P(3)	13	14	ACOM
AO_N(3)	15	16	ACOM
reserved	17	18	ACOM
reserved	19	20	ACOM

6.15 SL2-100 CMS only (SL2, J24)

Reference Designator	J24
Manufacture	TE Connectivity AMP Connectors (see page 11)
Manufacture Part Number	5103308-1
Description	CONN 10-position 4-wall polarized 0.100 inch (2.54mm) straight through-hole
Output	RS422 differential pair.



10-pin connector can be connected directly to DB9 and be compatible with other SL2-100 equipment with 9-pin D-SUB connectors

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
DATA_IN+	1	2	DATA_IN-
no connect	3	4	COM
no connect	5	6	COM
no connect	7	8	DATA_OUT-
DATA_OUT+	9	10	no connect

DB9 Connector pinout.

10-pin IDC <--> ribbon cable <--> 9-pin female D-SUB

PIN NAME	PIN NUMBER	PIN NUMBER	PIN NAME
DATA_IN+	1	6	DATA_IN-
no connect	2	7	COM
no connect	3	8	COM
no connect	4	9	DATA_OUT-
DATA_OUT+	5		

7 Register Set

- Work in progress -

7.1 Component ID summary

COMPONENT ACRONYM	ENUM SYMBOL	UNIQUE ID
SYS	CMP_ID_SYS	0x803B
IDI	CMP_ID_IDI	0x8033
IDO	CMP_ID_IDO	0x8034
DIOA	CMP_ID_DIOA	0x8035
DIOB	CMP_ID_DIOB	0x8835
QEA	CMP_ID_QEA	0x8036
QEB	CMP_ID_QEB	0x8836
XY2	CMP_ID_XY2	0x8037
AO	CMP_ID_AO	0x8038
SL2 (CMS Only)	CMP_ID_SL2	0x8039
CT	CMP_ID_CT	0x803A

```
/* channel is located at bits [14:11].  This allows a system to support up to 16 components of
the same type and revision. */
enum { CMP_ID_MASK_CHANNEL = 0x7800 };

enum component_id
{
  CMP_ID_SYS  = 0x803B,
  CMP_ID_IDI  = 0x8033,
  CMP_ID_IDO  = 0x8034,
  CMP_ID_DIOA = 0x8035,
  CMP_ID_DIOB = 0x8835,
  CMP_ID_QEA  = 0x8036,
  CMP_ID_QEB  = 0x8836,
  CMP_ID_XY2  = 0x8037,
  CMP_ID_AO   = 0x8038,
  CMP_ID_SL2  = 0x8039, /* CMS only */
  CMP_ID_CT   = 0x803A,
};
```

Notes

The intention of the ID values is to provide:

- Component revision information. For example, upgrading the CT we would change the ID to say 0x803C. Software then has option to support both 0x803A and 0x803C CT revisions, it just has to ID them. The new CT would simply be added to the above list.
- Component channel information. Allows up to 16 of the same components to exist within the same PCIe BAR space.
- Component location within memory space. Components can be moved around and software can still link up with them correctly.
- Used by driver/application to support multiple revisions of components in arbitrary memory locations

The bottom line is that the ID mechanism allows for future expansion will supporting anything legacy with minimal effort. This is unique to Apex Embedded Systems (well at the moment).

7.2 Component Offset summary

COMPONENT ACRONYM	SOFTWARE CONSTANT NAME	OFFSET
SYS	CMP_OFFSET_SYS	0x0000
IDI	CMP_OFFSET_IDI	0x0100
IDO	CMP_OFFSET_IDO	0x0200
DIOA	CMP_OFFSET_DIOA	0x0300
DIOB	CMP_OFFSET_DIOB	0x0400
QEA	CMP_OFFSET_QEA	0x0500
QEB	CMP_OFFSET_QEB	0x0600
XY2	CMP_OFFSET_XY2	0x0700
AO	CMP_OFFSET_AO	0x0800
SL2 (CMS Only)	CMP_OFFSET_SL2	0x0900
CT	CMP_OFFSET_CT	0x0A00

```
enum component_offset
{
  CMP_OFFSET_SYS  = 0x0000,
  CMP_OFFSET_IDI  = 0x0100,
  CMP_OFFSET_IDO  = 0x0200,
  CMP_OFFSET_DIOA = 0x0300,
  CMP_OFFSET_DIOB = 0x0400,
  CMP_OFFSET_QEA  = 0x0500,
  CMP_OFFSET_QEB  = 0x0600,
  CMP_OFFSET_XY2  = 0x0700,
  CMP_OFFSET_AO   = 0x0800,
  CMP_OFFSET_SL2  = 0x0900,
```

```
  CMP_OFFSET_CT    = 0x0A00
};
```

Notes

1. Software can be written to scan the register set to determine component offsets.
2. Apex intention is to allow for continued adding and updating of components

7.3 System (SYS, 0x803B)

The System component is a top level management component.

7.3.1 System ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_SYS + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_SYS + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
System Component ID	r	0x803B	0x803B

Description

This register indicates the ID of this component.

Example

7.3.2 System Board ID (0x02)

System board ID register

Register Layout

Register is a 16-bit register that is read only.

The purpose of this register is to allow driver or code to discern between multiple boards in one system. This is not used by the FPGA for decode purposes. It is strictly a high level mechanism for sorting/organizing systems with two or more boards within a system.

BAR + CMP_OFFSET_SYS + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	X	BRD_ID2	BRD_ID1	BRD_ID0

BAR + CMP_OFFSET_SYS + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	X	X

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	r	X	Don't care
BRD_ID[2:0]	r	000	The board ID as set by the jumpers stuffed at J17. If no jumpers stuffed the register defaults to "000".

Description

System board ID register

Example

7.3.3 System Build Version (0x04)

7.3.4 System Component Offset (0x06)

Address offset from component to component within the address space

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_SYS + 0x0006

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_SYS + 0x0007

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Component offsets	r	256	256

Description

Address offset from component to component within the address space.

Example

This is a demonstration of reading the range of registers to inventory all the components within the allowable address range.

```
int  offset;
u_int16_t id, tag_id, offset_component, offset_max;

printf( "SCANNING REGISTER SPACE:\n" );

offset_component = Read_U16( SYS_REG_COMP_OFFSET );
offset_max       = Read_U16( SYS_REG_OFFSET_MAX  );

tag_id = Read_U16( SYS_REG_TAG_A );
for (offset = 0; offset < offset_max; offset = offset + offset_component )
```

```
    {
        id = Read_U16( offset );
        tag_id = Read_U16( SYS_REG_TAG_A );
        if ( 0 != ( 0x7FFF & tag_id ) )
        {
            printf( " ID 0x%X found at offset 0x%X.  tag = 0x%X\n", (int) id, offset, (int)
tag_id );
        }
    }
```

7.3.5 System Offset Maximum (0x08)

Register set address range

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_SYS + 0x0008

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_SYS + 0x0009

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Maximum space used by register set	r	4095	4095

Description

Register set address range.

The register range within BAR0. 0 to offset maximum. In this case offset maximum is set to 4096.

Example

This is a demonstration of reading the range of registers to inventory all the components within the allowable address range.

```
    int  offset;
    u_int16_t id, tag_id, offset_component, offset_max;

    printf( "SCANNING REGISTER SPACE:\n" );
```

```
    offset_component = Read_U16( SYS_REG_COMP_OFFSET );
    offset_max       = Read_U16( SYS_REG_OFFSET_MAX  );

    tag_id = Read_U16( SYS_REG_TAG_A );
    for (offset = 0; offset < offset_max; offset = offset + offset_component )
    {
        id = Read_U16( offset );
        tag_id = Read_U16( SYS_REG_TAG_A );
        if ( 0 != ( 0x7FFF & tag_id ) )
        {
            printf( " ID 0x%X found at offset 0x%X.  tag = 0x%X\n", (int) id, offset, (int)
tag_id );
        }
    }
```

7.3.6 System Error (0x10)

Future feature

7.3.7 System Tag A (0x14)

Indicates if a previous read was from a component ID register.

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_SYS + 0x0014

D7	D6	D5	D4	D3	D2	D1	D0
QEB_ID	QEA_ID	DIOB_ID	DIOA_ID	XY2_ID	IDO_ID	IDI_ID	SYS_ID

BAR + CMP_OFFSET_SYS + 0x0015

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	CT_ID	SL2_ID	X	AO_ID

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	-	-	Don't Care
SYS_ID	r	0	Previous read had been from SYS ID register
IDI_ID	r	0	
IDO_ID	r	0	

XY2_ID	r	0	
DIOA_ID	r	0	
DIOB_ID	r	0	
QEA_ID	r	0	
QEB_ID	r	0	
AO_ID	r	0	
		0	
SL2_ID	r	0	
CT_ID	r	0	

Description

This register indicates if any previous read was from a component ID register. This is a way to verify that software has actually captured valid ID registers. Further, it allows FPGA components to be added, removed and changed and allow software a mechanism to build a table from component ID to proper set of functions in which to manipulate the component.

Any time this register is read, all ID indicator bits will be cleared.

IDs in general serve as a form of revision management. The values are arbitrary and sequential as we create components. Software upon startup will scan the ID registers and then all IDs identified can be used to create a table which points to the correct "mini-driver" to support that component. This is a way to future proof the design and allow ongoing development and improvements while supporting older revisions at the same time.

For example, the current counter/timer ID is set to 0x803A and say a new version is implement with an ID 0x8059. This means that when software scans the registers and finds 0x803A, it will use the "mini-driver" or set of functions that are pointed to by the ID. Same with 0x8059. Thus, it is possible to have two version of hardware and one software revision which contains support for both components.

A table of IDs will be made available for each firmware revision provided.

To properly scan FPGA components perform the following:

```
    int offset;
    u_int16_t offset_component = Read_U16( SYS_REG_COMP_OFFSET ); // obtain component base
offsets
    u_int16_t offset_max       = Read_U16( SYS_REG_OFFSET_MAX  ); // obtain maximum address
range to scan

    //the tag register will indicate if the previous read was an ID register
    tag_lo = Read_U16( SYS_REG_TAG_A ); // initial read of the id register to clear it
    for (offset = 0; offset < offset_max; offset = offset + offset_component )
    {
        id = Read_U16( offset );
        tag_lo = Read_U16( SYS_REG_TAG_A ); // was the last read a component ID register?
        if ( 0 != ( 0x7FFF & tag_lo ) )
```

```
        {
            printf( " ID 0x%X found at offset 0x%X.  tag = 0x%X\n", (int) id, index, (int)
tag_lo );
        }
    }
```

See Also

Register Summary

Example

7.3.8 System Scratch Register (0x20 - 0x27)

This register indicates the ID of this component

Register Layout

The purpose of the four scratch registers is for testing basic PCIe communications used during board initial testing. These registers can be used by the user application to store things like process ID or other key information for properly handling and negotiating the used of the I/Os to their intended master software components.

These are 16-bit registers, but can be read/written as 32-bit or 64-bit register as well.

SCRATCH A: BAR + CMP_OFFSET_SYS + 0x0020. Read/Write.

D7	D6	D5	D4	D3	D2	D1	D0

SCRATCH A: BAR + CMP_OFFSET_SYS + 0x0021. Read/Write.

D15	D14	D13	D12	D11	D10	D9	D8

SCRATCH B: BAR + CMP_OFFSET_SYS + 0x0022. Read/Write.

D7	D6	D5	D4	D3	D2	D1	D0

SCRATCH B: BAR + CMP_OFFSET_SYS + 0x0023. Read/Write.

D15	D14	D13	D12	D11	D10	D9	D8

SCRATCH C: BAR + CMP_OFFSET_SYS + 0x0024. Read/Write.

D7	D6	D5	D4	D3	D2	D1	D0

SCRATCH C: BAR + CMP_OFFSET_SYS + 0x0025. Read/Write.

D15	D14	D13	D12	D11	D10	D9	D8

SCRATCH D: BAR + CMP_OFFSET_SYS + 0x0026. Read/Write.

D7	D6	D5	D4	D3	D2	D1	D0

SCRATCH D: BAR + CMP_OFFSET_SYS + 0x0027. Read/Write.

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
SCRA[15:0]	rw	0x0000	Scratch register A
SCRB[15:0]	rw	0x0000	Scratch register B
SCRC[15:0]	rw	0x0000	Scratch register C
SCRD[15:0]	rw	0x0000	Scratch register D

Description

This register indicates the ID of this component.

Example

7.3.9 System Configuration (0x28)

System configuration register.

Register Layout

Register is a 16-bit register that is read/write.

BAR + CMP_OFFSET_SYS + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	LED3	LED2	LED1	LED0

BAR + CMP_OFFSET_SYS + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	X	X

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
LED[3:0]	rw	00	LED D1 signal selection 0000 = valid read/write cycles -nothing else has been assigned-

Description

System configuration register.

Example

7.3.10 System Component Reset (0x2A)

future feature

7.4 Analog Outputs (AO, 0x8038)

Analog output component has the following features:

Channel count	4
Resolutions	16-bits

Range	0-5V, 0-10V, ±5V, ±10V Set by channel configuration register Limited by maximum range jumpers.

Channels A & B share the same value output FIFO. The FIFO is 16 values deep. The status of the FIFO can be monitored by the AO status register. Data is delivered to the DACs at the fastest rate possible.

Channels C & D share the same value output FIFO. The FIFO is 16 values deep. The status of the FIFO can be monitored by the AO status register. Data is delivered to the DACs at the fastest rate possible.

7.4.1 Analog Outputs ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_AO + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Analog Output Component ID	r	0x8038	0x8038

Description

This register indicates the ID of this component.

Example

7.4.2 Analog Outputs Config CH-A (0x02)

Analog output configuration register.

Register Layout

Register is a 16-bit register that is read/write. When this register is written, the DAC will be updated with the latest range values or the maximum range values set by the hardware configured jumpers at J20.

BAR + CMP_OFFSET_AO + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
X	X	LDA(1)	LDA(0)	X	X	UBA_A	RA_A

BAR + CMP_OFFSET_AO + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	X	X

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
LDA[1:0]	rw	00	DAC Load behavior 00 = update immediately 01 = update via writing to update register 10 = update upon writing to the last DAC register (channel-D) 11 = update upon writing to the last DAC register (channel-D)
UBA_A	rw	0	Unipolar / Bipolar Analog Output channel-A 0 = Unipolar outupt range 1 = Bipolar output range
RA_A	rw	0	Range of Analog output channel-A 0 = 5 volt range 1 = 10 volt range

Description

Analog output configuration register.

Example

7.4.3 Analog Outputs Config CH-B (0x04)

Analog output configuration register.

Register Layout

Register is a 16-bit register that is read/write. When this register is written, the DAC will be updated with the latest range values or the maximum range values set by the hardware configured jumpers at J20.

BAR + CMP_OFFSET_AO + 0x0004

D7	D6	D5	D4	D3	D2	D1	D0
X	X	LDB(1)	LDB(0)	X	X	UBA_B	RA_B

BAR + CMP_OFFSET_AO + 0x0005

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	X	X

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
LDB[1:0]	rw	00	DAC Load behavior 00 = update immediately 01 = update via writing to update register 10 = update upon writing to the last DAC register (channel-D) 11 = update upon writing to the last DAC register (channel-D)
UBA_B	rw	0	Unipolar / Bipolar Analog Output channel-B 0 = Unipolar outupt range 1 = Bipolar output range
RA_B	rw	0	Range of Analog output channel-B 0 = 5 volt range 1 = 10 volt range

Description

Analog output configuration register.

Example

7.4.4 Analog Outupts Config CH-C (0x06)

Analog output configuration register.

Register Layout

Register is a 16-bit register that is read/write. When this register is written, the DAC will be updated with the latest range values or the maximum range values set by the hardware configured jumpers at J19.

BAR + CMP_OFFSET_AO + 0x0006

D7	D6	D5	D4	D3	D2	D1	D0
X	X	LDC(1)	LDC(0)	X	X	UBA_C	RA_C

BAR + CMP_OFFSET_AO + 0x0007

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	X	X

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
LDC[1:0]	rw	00	DAC Load behavior 00 = update immediately 01 = update via writing to update register 10 = update upon writing to the last DAC register (channel-D) 11 = update upon writing to the last DAC register (channel-D)
UBA_C	rw	0	Unipolar / Bipolar Analog Output channel-C 0 = Unipolar outupt range 1 = Bipolar output range
RA_C	rw	0	Range of Analog output channel-C 0 = 5 volt range 1 = 10 volt range

Description

Analog output configuration register.

Example

7.4.5 Analog Outputs Config CH-D (0x08)

Analog output configuration register.

Register Layout

Register is a 16-bit register that is read/write. When this register is written, the DAC will be updated with the latest range values or the maximum range values set by the hardware configured jumpers at J19.

BAR + CMP_OFFSET_AO + 0x0008

D7	D6	D5	D4	D3	D2	D1	D0
X	X	LDD(1)	LDD(0)	X	X	UBA_D	RA_D

BAR + CMP_OFFSET_AO + 0x0009

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	X	X

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
LDD[1:0]	rw	00	DAC Load behavior 00 = update immediately 01 = update via writing to update register 10 = update upon writing to the last DAC register (channel-D) 11 = update upon writing to the last DAC register (channel-D)
UBA_D	rw	0	Unipolar / Bipolar Analog Output channel-D 0 = Unipolar outupt range 1 = Bipolar output range
RA_D	rw	0	Range of Analog output channel-D 0 = 5 volt range 1 = 10 volt range

Description

Analog output configuration register.

Example

7.4.6 Analog Outputs Status (0x0E)

Analog output configuration register.

Register Layout

Register is a 16-bit register that is read only. The status is captured upon the read of the LSB portion of the register. If 16-bit read, then all captured simultaneously.

BAR + CMP_OFFSET_AO + 0x000E

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_FULL)_AB	RANGE_CAP_B	UBA_B	RA_B	FIFO_EMPTY_AB	RANGE_CAP_A	UBA_A	RA_A

BAR + CMP_OFFSET_AO + 0x000F

D15	D14	D13	D12	D11	D10	D9	D8
FIFO_FULL)_CD	RANGE_CAP_D	UBA_D	RA_D	FIFO_EMPTY_CD	RANGE_CAP_B	UBA_B	RA_B

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
FIFO_FULL_CD	r	0	Channel-C and Channel-D FIFO full status 0 = not full 1 = full
RANGE_CAP_D	r	0	Hardware range maximum limit reached and capping the range for channel-D. 0 = no maximum range capping 1 = range capping as set be J19 jumpers for channel-D
UBA_D	r	0	Unipolar / Bipolar Analog Output channel-D 0 = Unipolar outupt range 1 = Bipolar output range
RA_D	r	0	Range of Analog output channel-D 0 = 5 volt range 1 = 10 volt range
FIFO_EMPTY_CD	r	1	Channel-C and Channel-D FIFO empty status 0 = not empty 1 = empty
RANGE_CAP_C	r	0	Hardware range maximum limit reached and capping the range for channel-C. 0 = no maximum range capping 1 = range capping as set be J19 jumpers for channel-C

UBA_C	r	0	Unipolar / Bipolar Analog Output channel-C 0 = Unipolar outupt range 1 = Bipolar output range
RA_C	r	0	Range of Analog output channel-C 0 = 5 volt range 1 = 10 volt range
FIFO_FULL_AB	r	0	Channel-A and Channel-B FIFO full status 0 = not full 1 = full
RANGE_CAP_B	r	0	Hardware range maximum limit reached and capping the range for channel-B. 0 = no maximum range capping 1 = range capping as set be J20 jumpers for channel-B
UBA_B	r	0	Unipolar / Bipolar Analog Output channel-B 0 = Unipolar outupt range 1 = Bipolar output range
RA_B	r	0	Range of Analog output channel-B 0 = 5 volt range 1 = 10 volt range
FIFO_EMPTY_AB	r	1	Channel-A and Channel-B FIFO empty status 0 = not empty 1 = empty
RANGE_CAP_A	r	0	Hardware range maximum limit reached and capping the range for channel-A. 0 = no maximum range capping 1 = range capping as set be J20 jumpers for channel-A
UBA_A	r	0	Unipolar / Bipolar Analog Output channel-A 0 = Unipolar outupt range 1 = Bipolar output range
RA_A	r	0	Range of Analog output channel-A 0 = 5 volt range 1 = 10 volt range

Description

Analog output configuration register.

Example

7.4.7 Analog Outputs Value CH-A (0x10)

Analog output value register

Register Layout

Register is a 16-bit register that is write only. The value is committed on writing the MSB register. Depending on analog output DAC update settings, the value may be held until a simultaneous update command is issued.

BAR + CMP_OFFSET_AO + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0011

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
AO.VALUE.CHA[15:0]	w	-	16-bit value written to the analog output channel A located at connector J23.1.

Description

Analog output value register.

Example

7.4.8 Analog Outputs Value CH-B (0x12)

Analog output value register

Register Layout

Register is a 16-bit register that is write only. The value is committed on writing the MSB register. Depending on analog output DAC update settings, the value may be held until a simultaneous update command is issued.

BAR + CMP_OFFSET_AO + 0x0012

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0013

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
AO.VALUE.CHB[15:0]	w	-	16-bit value written to the analog output channel A located at connector J23.5.

Description

Analog output value register.

Example

7.4.9 Analog Outputs Value CH-C (0x14)

Analog output value register

Register Layout

Register is a 16-bit register that is write only. The value is committed on writing the MSB register. Depending on analog output DAC update settings, the value may be held until a simultaneous update command is issued.

BAR + CMP_OFFSET_AO + 0x0014

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0015

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
AO.VALUE.CHC[15:0]	w	-	16-bit value written to the analog output channel A located at connector J23.9.

Description

Analog output value register.

Example

7.4.10 Analog Outputs Value CH-D (0x16)

Analog output value register

Register Layout

Register is a 16-bit register that is write only. The value is committed on writing the MSB register. Depending on analog output DAC update settings, the value may be held until a simultaneous update command is issued.

BAR + CMP_OFFSET_AO + 0x0016

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0017

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
AO.VALUE.CHD[15:0]	w	-	16-bit value written to the analog output channel A located at connector J23.13.

Description

Analog output value register.

Example

7.4.11 Analog Outputs Update (0x18)

Analog output software simultaneous update register

Register Layout

Register is a 16-bit register that is write only. Writing to this register will update simultaneously any of the DACs that have LD<ch>[1:0]="01" for software update. The value written is a don't care. We suggest writing value 0x0000 as it allows for future possibilities.

BAR + CMP_OFFSET_AO + 0x0018

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0019

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	

Description

Analog output software simultaneous update register

Example

7.4.12 Analog Outputs Clear (0x1A)

Analog output software simultaneous clear register

Register Layout

Register is a 16-bit register that is write only. Writing to this register will update simultaneously reset all DAC outputs to a cold reset state. We suggest writing value 0x0000 as it allows for future possibilities.

BAR + CMP_OFFSET_AO + 0x0018

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_AO + 0x0019

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	

Description

Analog output software simultaneous clear register

Example

7.5 Isolated Digital Inputs (IDI, 0x8033)

7.5.1 Isolated Digital Inputs ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_IDI + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_IDI + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
IDI Component ID	r	0x803B	0x8033

Example

Description

This register indicates the ID of this component.

7.5.2 Isolated Digital Inputs Value (0x02)

The Isolated Digital Input value register

Register Layout

Register is a 16-bit register that is read only. Reading the LSB will capture the values simultaneously. Reading all 16-bits will result in the same simultaneous capturing.

BAR + CMP_OFFSET_IDI + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
IDI7	IDI6	IDI5	IDI4	IDI3	IDI2	IDI1	IDI0

BAR + CMP_OFFSET_IDI + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
IDI15	IDI14	IDI13	IDI12	IDI1	IDI10	IDI9	IDI8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
IDI[15:0]	r	0x0000	Isolated digital inputs 0 = no signal 1 = signal, polarity agnostic (>± 3 volts)

Description

The Isolated Digital Input value register

Example

7.5.3 Isolated Digital Inputs Pending (0x0C)

future feature

7.5.4 Isolated Digital Inputs Clear (0x0E)

future feature

7.5.5 Isolated Digital Inputs Configuration CH[15:0] (0x010-0x02F)

Isolated Digital Input configuration register.

Register Layout

Register is a 16-bit register that is read/write. This register is available for each individual isolated digital input, 16-channels in all.

BAR + CMP_OFFSET_IDI + 0x0010 + 2 * CHANNEL_INDEX

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0

BAR + CMP_OFFSET_IDI + 0x0011 + 2 * CHANNEL_INDEX

D15	D14	D13	D12	D11	D10	D9	D8
POLARITY	0	0	0	0	0	0	0

Where: 0 <= CHANNEL_INDEX <= 15

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
POLARITY	rw	0	Isolated digital input polarity 0 = non-inverted (i.e. signal => '1', otherwise => '0') 1 = inverted (i.e. signal => '0', otherwise => '1');

Description

Isolated Digital Input configuration register.

Example

7.6 Isolated Digital Outputs (IDO, 0x8034)

7.6.1 Isolated Digital Outputs ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_IDO + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_IDO + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
IDO Component ID	r	0x803B	0x8034

Description

This register indicates the ID of this component.

Example

7.6.2 Isolated Digital Outputs Value (0x02)

The Isolated Digital Output value register

Register Layout

Register is a 16-bit register that is read/write. Writing the MSB will simultaneously commit the values to the output hardware.

BAR + CMP_OFFSET_IDI + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
IDO7	IDO6	IDO5	IDO4	IDO3	IDO2	IDO1	IDO0

BAR + CMP_OFFSET_IDI + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
IDO15	IDO14	IDO13	IDO12	IDO11	IDO10	IDO9	IDO8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
IDO[15:0]	rw	0x0000	Isolated digital output value 0 = output off 1 = output on

Description

The Isolated Digital Output value register

Example

7.7 SL2-100 Interface (SL2, 0x8039)

7.7.1 SL2-100 Interface Licensing Restriction

May 1, 2020

Robert Faron

Control Micro Systems, Inc.

4420-A Metric Drive

Winter Park, FL 32792

7

Robert,

The SL2-100 VHDL proprietary code provided from Control Micro Systems to Apex Embedded Systems will only be used for products delivered from Apex Embedded Systems to Control Micro Systems. This restriction is due to SL2-100 licensing between Control Micro Systems and Scanlab.

Regards,

Mike Ihm

Owner

Apex Embedded Systems

7.7.2 SL2-100 Interface ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_SL2 + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_SL2 + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
SL2 Component ID	r	0x803B	0x8039

Description

This register indicates the ID of this component.

Example

7.7.3 SL2-100 Interface Config (0x02)

Unused, place older for any future configuration needs.

7.7.4 SL2-100 Interface Value XL (0x04)

SL2 X-Position LSB value register

Register Layout

Register is a 16-bit register that is read/write. Writing to the MSB of YH will commit both the X[24:0] and Y[24:0] values to the hardware.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the SL2 interface.

BAR + CMP_OFFSET_SL2 + 0x0004

D7	D6	D5	D4	D3	D2	D1	D0
X7	X6	X5	X4	X3	X2	X1	X0

BAR + CMP_OFFSET_SL2 + 0x0005

D15	D14	D13	D12	D11	D10	D9	D8
X15	X14	X13	X12	X11	X10	X9	X8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X[24:0]	rw	0x0000_0000	This is the low word portion of the X-position value.

Description

SL2 X-Position LSB value register

Example

7

7.7.5 SL2-100 Interface Value XH (0x06)

SL2 X-Position MSB value register

Register Layout

Register is a 16-bit register that is read/write. Writing to the MSB of YH will commit both the X[24:0] and Y[24:0] values to the hardware.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the SL2 interface.

BAR + CMP_OFFSET_SL2 + 0x0006

D7	D6	D5	D4	D3	D2	D1	D0
X23	X22	X21	X20	X19	X18	X17	X16

BAR + CMP_OFFSET_SL2 + 0x0007

D15	D14	D13	D12	D11	D10	D9	D8
							X24

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X[24:0]	rw	0x0000_0000	This is the high word portion of the X-position value.

Description

SL2 X-Position MSB value register

Example

7.7.6 SL2-100 Inteface Value YL (0x08)

SL2 Y-Position LSB value register

Register Layout

Register is a 16-bit register that is read/write. Writing to the MSB of YH will commit both the X[24:0] and Y[24:0] values to the hardware.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the SL2 interface.

BAR + CMP_OFFSET_SL2 + 0x0004

D7	D6	D5	D4	D3	D2	D1	D0
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

BAR + CMP_OFFSET_SL2 + 0x0005

D15	D14	D13	D12	D11	D10	D9	D8
Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[24:0]	rw	0x0000_0000	This is the the low word portion of the Y-position value.

Description

SL2 Y-Position LSB value register

Example

7.7.7 SL2-100 Interface Value YH (0x0A)

SL2 Y-Position MSB value register

Register Layout

Register is a 16-bit register that is read/write. Writing to the MSB of YH will commit both the X[24:0] and Y[24:0] values to the hardware.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the SL2 interface.

BAR + CMP_OFFSET_SL2 + 0x0004

D7	D6	D5	D4	D3	D2	D1	D0
Y23	Y22	Y21	Y20	Y19	Y18	Y17	Y16

BAR + CMP_OFFSET_SL2 + 0x0005

D15	D14	D13	D12	D11	D10	D9	D8
							Y24

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[24:0]	rw	0x0000_0000	This is the the high word portion of the Y-position value.

Description

SL2 Y-Position MSB value register

Example

7.7.8 SL2-100 Interface Status (0x0C)

Unused, place older for any future configuration needs.

7.8 XY2 Interface (XY2, 0x8037)

7.8.1 XY2 Interface ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_XY2 + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_XY2 + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
XY2 Component ID	r	0x803B	0x8037

Description

This register indicates the ID of this component.

Example

7.8.2 XY2 Interface Config (0x02)

XY2-100/200 configuration register.

Register Layout

Register is a 16-bit register that is read/write.

The default value upon power up will provide XY2-100 behavior and XYZ data is committed upon writing to the ZH register.

The CPHA and CPOL for either TX or RX is simply the SPI interface and the clock configurations and the behavior.

It is possible to use this component as a multiple SPI interface albeit a bit limited at this time.

BAR + CMP_OFFSET_XY2 + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
0	0	RX_CPHA	RX_CPOL	0	TX_PARITY	TX_CPHA	TX_CPOL

BAR + CMP_OFFSET_XY2 + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
ENABLE	CLK2	CLK1	CLK0	0	0	COMMIT1	COMMIT0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
ENABLE	rw	0	Enables both transmit and receive sections of the XY2 component 0 = inactive 1 = active
CLK[2:0]	rw	000	The XY2 output clock frequency 000 = 2 MHz (default, XY2-100 compatible) 001 = 4 MHz (XY2-200 compatible) 010 = 1 MHz 011 = 500 KHz 1XX = none, off.

COMMIT[1:0]	rw	00	Determines when XYZ data is committed to the hardware for output. 00 = ZH. Writing to the Z MSB will cause XYZ data to be committed 01 = YH. Writing to the Y MSB will cause the XYZ data to be committed 10 = XH. Writing to the X MSB will cause the XYZ data to be committed 11 = invalid configuration
RX_CPHA	rw	0	Receiver (status) clock phase 0 = data change on rising edge (default) 1 = data change on falling edge
RX_CPOL	rw	0	Receiver (status) clock polarity 0 = non-inverted (default) 1 = inverted
TX_PARITY	rw	0	Transmit (XYZ) parity 0 = even parity 1 = odd parity
TX_CPHA	rw	0	Transmit (XYZ) clock phase 0 = data change on falling edge (default) 1 = data change on rising edge
TX_CPOL	rw	0	Transmit (XYZ) clock polarity 0 = non-inverted (default) 1 = inverted

Description

XY2-100/200 configuration register.

Example

7.8.3 XY2 Interface Value XL (0x10)

XY2-100/200 X-Position LSB value register

Register Layout

Register is a 16-bit register that is read/write.

The values are committed to the hardware based on configuration settings.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the XY2 interface.

BAR + CMP_OFFSET_XY2 + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
X7	X6	X5	X4	X3	X2	X1	X0

BAR + CMP_OFFSET_XY2 + 0x0011

D15	D14	D13	D12	D11	D10	D9	D8
X15	X14	X13	X12	X11	X10	X9	X8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X[19:0]	rw	0x0000_0000	This is the low word portion of the X-position value.

Description

XY2-100/200 X-Position LSB value register

Example

7.8.4 XY2 Interface Value XH (0x12)

XY2-100/200 X-Position MSB value register

Register Layout

Register is a 16-bit register that is read/write.

The values are committed to the hardware based on configuration settings.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the XY2 interface.

BAR + CMP_OFFSET_XY2 + 0x0012

D7	D6	D5	D4	D3	D2	D1	D0
				X19	X18	X17	X16

BAR + CMP_OFFSET_XY2 + 0x0013

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X[19:0]	rw	0x0000_0000	This is the high word portion of the X-position value.

Description

XY2-100/200 X-Position MSB value register

Example

7.8.5 XY2 Interface Value YL (0x14)

XY2-100/200 Y-Position LSB value register

Register Layout

Register is a 16-bit register that is read/write.

The values are committed to the hardware based on configuration settings.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the XY2 interface.

BAR + CMP_OFFSET_XY2 + 0x0014

D7	D6	D5	D4	D3	D2	D1	D0
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

BAR + CMP_OFFSET_XY2 + 0x0015

D15	D14	D13	D12	D11	D10	D9	D8
Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[19:0]	rw	0x0000_0000	This is the low word portion of the Y-position value.

Description

XY2-100/200 Y-Position LSB value register

Example

7.8.6 XY2 Interface Value YH (0x16)

XY2-100/200 Y-Position MSB value register

Register Layout

Register is a 16-bit register that is read/write.

The values are committed to the hardware based on configuration settings.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the XY2 interface.

BAR + CMP_OFFSET_XY2 + 0x0016

D7	D6	D5	D4	D3	D2	D1	D0
				Y19	Y18	Y17	Y16

BAR + CMP_OFFSET_XY2 + 0x0017

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[19:0]	rw	0x0000_0000	This is the high word portion of the Y-position value.

Description

XY2-100/200 Y-Position MSB value register

Example

7.8.7 XY2 Interface Value ZL (0x18)

XY2-100/200 Z-Position MSB value register

Register Layout

Register is a 16-bit register that is read/write.

The values are committed to the hardware based on configuration settings.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the XY2 interface.

BAR + CMP_OFFSET_XY2 + 0x0018

D7	D6	D5	D4	D3	D2	D1	D0
Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0

BAR + CMP_OFFSET_XY2 + 0x0019

D15	D14	D13	D12	D11	D10	D9	D8
Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Z[19:0]	rw	0x0000_0000	This is the low word portion of the Z-position value.

Description

XY2-100/200 Z-Position MSB value register

Example

7

7.8.8 XY2 Interface Value ZH (0x1A)

XY2-100/200 Z-Position MSB value register

Register Layout

Register is a 16-bit register that is read/write.

The values are committed to the hardware based on configuration settings.

The readability of this register is purely local within the FPGA. This can be useful for capturing the state of the XY2 interface.

BAR + CMP_OFFSET_XY2 + 0x001A

D7	D6	D5	D4	D3	D2	D1	D0
				Z19	Z18	Z17	Z16

BAR + CMP_OFFSET_XY2 + 0x001B

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Z[19:0]	rw	0x0000_0000	This is the the high word portion of the Z-position value.

Description

XY2-100/200 Z-Position MSB value register

Example

7.8.9 XY2 Interface VAlue SL (0x1C)

XY2-100/200 Status LSB value register

Register Layout

Register is a 16-bit register that is read only.

The value is captured upon an LSB read.

BAR + CMP_OFFSET_XY2 + 0x001C

D7	D6	D5	D4	D3	D2	D1	D0
S6	S5	S4	S3	S2	S1	S0	PARITY

BAR + CMP_OFFSET_XY2 + 0x001D

D15	D14	D13	D12	D11	D10	D9	D8
S14	S13	S12	S11	S10	S9	S8	S7

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
S[19:0]	r	0x0000_0000	This is the low word portion of the status value.
PARITY	r	0	This is the parity bit as read from the status register. The intent is to have software evaluate the parity bit and deal with any subsequent error handling sequences.

Description

XY2-100/200 Status LSB value register

Example

7.8.10 XY2 Interface Value SH (0x1E)

XY2-100/200 Status MSB value register

Register Layout

Register is a 16-bit register that is read only.

The value is captured upon an LSB read.

BAR + CMP_OFFSET_XY2 + 0x001E

D7	D6	D5	D4	D3	D2	D1	D0
			S19	S18	S17	S16	S15

BAR + CMP_OFFSET_XY2 + 0x001F

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
S[19:0]	r	0x0000_0000	This is the the high word portion of the status value.

Description

XY2-100/200 Status MSB value register

Example

7.9 Quadrature Encoder Input (QEA, 0x8036)

Note that the counter is currently emitted via the Y output register.

7.9.1 Quadrature Encoder A ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_QEA + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_QEA + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
QEA Component ID	r	0x8036	0x8036

Description

This register indicates the ID of this component.

Example

7.9.2 Quadrature Encoder A Config (0x02)

Quadrature Encoder Channel-A (QEA) configuration register.

Register Layout

Register is a 16-bit register that is read/write.

BAR + CMP_OFFSET_QEA + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
0	0	CNT1	CNT0	0	0	MODE1	MODE0

BAR + CMP_OFFSET_QEA + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
EN_5V	0	0	0	0	0	0	0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
EN_5V	rw	0	Enables current limited 5 volts to J2.2 and J2.10. 0 = 5V disabled (default) 1 = 5V active. If current exceeds 500mA, the circuit will automatically switch off. Check status register.
CNT[1:0]	rw	00	Counter Behavior 00 = normal 01 = -2^31 to 2^31 - 1 10 = -steps to +steps - 1 11 = single cycle
MODE[1:0]	rw	00	Quadrature stepping 00 = disabled 01 = x1 10 = x2 11 = x4

Description

Quadrature Encoder Channel-A (QEA) configuration register.

Example

7.9.3 Quadrature Encoder A Config Input-A (0x04)

Quadrature Encoder Input-A configuration register.

Register Layout

Register is a 16-bit register that is read/write.

BAR + CMP_OFFSET_QEA + 0x0004

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0

BAR + CMP_OFFSET_QEA + 0x0005

D15	D14	D13	D12	D11	D10	D9	D8
POLARITY	0	0	0	0	0	0	0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
POLARITY	rw	0	Isolated digital input polarity 0 = non-inverted (i.e. signal => '1', otherwise => '0') 1 = inverted (i.e. signal => '0', otherwise => '1');

Description

Quadrature Encoder Input-A configuration register.

Example

7.9.4 Quadrature Encoder A Config Input-B (0x06)

Quadrature Encoder Input-B configuration register.

Register Layout

Register is a 16-bit register that is read/write.

BAR + CMP_OFFSET_QEA + 0x0006

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0

BAR + CMP_OFFSET_QEA + 0x0007

D15	D14	D13	D12	D11	D10	D9	D8
POLARITY	0	0	0	0	0	0	0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
POLARITY	rw	0	Isolated digital input polarity 0 = non-inverted (i.e. signal => '1', otherwise => '0') 1 = inverted (i.e. signal => '0', otherwise => '1');

Description

Quadrature Encoder Input-B configuration register.

Example

7.9.5 Quadrature Encoder A Config Input-Z (0x08)

Quadrature Encoder Input-Z configuration register.

Register Layout

Register is a 16-bit register that is read/write.

BAR + CMP_OFFSET_QEA + 0x0008

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0

BAR + CMP_OFFSET_QEA + 0x0009

D15	D14	D13	D12	D11	D10	D9	D8
POLARITY	0	0	0	0	0	0	0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
POLARITY	rw	0	Isolated digital input polarity 0 = non-inverted (i.e. signal => '1', otherwise => '0') 1 = inverted (i.e. signal => '0', otherwise => '1');

Description

Quadrature Encoder Input-Z configuration register.

Example

7.9.6 Quadrature Encoder A Compare LSB (0x10)

future feature

7.9.7 Quadrature Encoder A Compare MSB (0x12)

7.9.8 Quadrature Encoder A Load LSB (0x14)

QEA Counter output LSB value register

Register Layout

Register is a 16-bit register that is read only register.

The 32-bit counter is captured in its entirety upon reading the LSB at 0x1C.

The counter output is captured by reading this register at this time.

BAR + CMP_OFFSET_QEA + 0x001C

D7	D6	D5	D4	D3	D2	D1	D0
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

BAR + CMP_OFFSET_QEA + 0x001D

D15	D14	D13	D12	D11	D10	D9	D8
Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[31:0]	r	0x0000_0000	This is the low word portion of the 32-bit counter. This is a signed value.

Description

QEA Counter output LSB value register

Example

7.9.9 Quadrature Encoder A Load MSB (0x16)

QEA Counter Load MSB value register

Register Layout

Register is a 16-bit register that is read/write register.

The 32-bit counter is captured in its entirety upon reading the LSB at 0x1C.

The counter output is captured by reading this register at this time.

BAR + CMP_OFFSET_XY2 + 0x0016

D7	D6	D5	D4	D3	D2	D1	D0
Y23	Y22	Y21	Y20	Y19	Y18	Y17	Y16

BAR + CMP_OFFSET_XY2 + 0x0017

D15	D14	D13	D12	D11	D10	D9	D8
Y31	Y30	Y29	Y28	Y27	Y26	Y25	Y24

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[31:0]	rw	0x0000_0000	This is the high word portion of the 32-bit counter. This is a signed value.

Description

QEA Counter Load MSB value register

Example

7.9.10 Quadrature Encoder A Out-X LSB (0x18)

future feature

7.9.11 Quadrature Encoder A Out-X MSB (0x1A)

future feature

7.9.12 Quadrature Encoder A Out-Y LSB (0x1C)

QEA Counter output LSB value register

Register Layout

Register is a 16-bit register that is read only register.

The 32-bit counter is captured in its entirety upon reading the LSB at 0x1C.

The counter output is captured by reading this register at this time.

BAR + CMP_OFFSET_QEA + 0x001C

D7	D6	D5	D4	D3	D2	D1	D0
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

BAR + CMP_OFFSET_QEA + 0x001D

D15	D14	D13	D12	D11	D10	D9	D8
Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[31:0]	r	0x0000_0000	This is the low word portion of the 32-bit counter. This is a signed value.

Description

QEA Counter output LSB value register

Example

7.9.13 Quadrature Encoder A Out-Y MSB (0x1E)

QEA Counter output MSB value register

Register Layout

Register is a 16-bit register that is read only register.

The 32-bit counter is captured in its entirety upon reading the LSB at 0x1C.

The counter output is captured by reading this register at this time.

BAR + CMP_OFFSET_XY2 + 0x0016

D7	D6	D5	D4	D3	D2	D1	D0
Y23	Y22	Y21	Y20	Y19	Y18	Y17	Y16

BAR + CMP_OFFSET_XY2 + 0x0017

D15	D14	D13	D12	D11	D10	D9	D8
Y31	Y30	Y29	Y28	Y27	Y26	Y25	Y24

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
Y[31:0]	rw	0x0000_0000	This is the high word portion of the 32-bit counter. This is a signed value.

Description

QEA Counter output MSB value register

Example

7.9.14 Quadrature Encoder A Time LSB (0x20)

future feature

7.9.15 Quadrature Encoder A Time MSB (0x22)

future feature

7.9.16 Quadrature Encoder A Status (0x24)

Quadrature Encoder Channel-A (QEA) status register.

Register Layout

Register is a 16-bit register that is read only. The status is captured upon the read of the LSB portion of the register. If 16-bit read, then all captured simultaneously.

BAR + CMP_OFFSET_QEA + 0x0024

D7	D6	D5	D4	D3	D2	D1	D0
					Z	B	A

BAR + CMP_OFFSET_QEA + 0x0025

D15	D14	D13	D12	D11	D10	D9	D8
							CL_FLG

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
CL_FLG	r	0	Status of current limit at the 5V outputs at J2.2 and J2.10. If current limit is active you must set CL_EN to zero and then set back to one. 0 = normal operation 1 = current limit active
Z	r	0	State of the Z-input
B	r	0	State of the B-input
A	r	0	State of the A-input

Description

Quadrature Encoder Channel-A (QEA) status register.

Example

7.10 Quadrature Encoder Input (QEB, 0x8836)

Refer to Quadrature Encoder Input QEA component. Replace CMP_OFFSET_QEA with CMP_OFFSET_QEB. Refer to Component Offset summary (see page 28).

7.11 Digital Inputs and Outputs (DIOA, 0x8035)

7.11.1 Digital Inputs and Outputs ID (0x00)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_DIOA + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_DIOA + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
DIOA Component ID	r	0x803B	0x8035

Description

This register indicates the ID of this component.

Example

7.11.2 Digital Inputs and Outputs Input Value (0x02)

Digital I/O Input value register

Register Layout

Register is a 16-bit register that is read only.

Reading the LSB will capture the values simultaneously.

Reading all 16-bits will result in the same simultaneous capturing.

The values read here are the values found at the pins at J7.

BAR + CMP_OFFSET_DIOA + 0x0002

D7	D6	D5	D4	D3	D2	D1	D0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

BAR + CMP_OFFSET_DIOA + 0x0003

D15	D14	D13	D12	D11	D10	D9	D8
DI15	DI14	DI13	DI12	DI1	DI10	DI9	DI8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
DI[15:0]	r	0x0000	TTL digital input state 0 = low input 1 = high input

Description

Digital I/O Input value register

Example

7.11.3 Digital Inputs and Outputs Output Value (0x04)

Digital I/O Output value register

Register Layout

Register is a 16-bit register that is read/write.

Writing the MSB will simultaneously commit the values to the output hardware.

The values propagate to the pins at J7 only if the direction register sets the byte/word as an output.

BAR + CMP_OFFSET_DIOA + 0x0004

D7	D6	D5	D4	D3	D2	D1	D0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

BAR + CMP_OFFSET_DIOA + 0x0005

D15	D14	D13	D12	D11	D10	D9	D8
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
DO[15:0]	rw	0x0000	Isolated digital output value 0 = output off 1 = output on

Description

Digital I/O Output value register

Example

7.11.4 Digital Inputs and Outputs Direction (0x0A)

Digital I/O Direction register

Register Layout

Register is a 16-bit register that is read/write.

BAR + CMP_OFFSET_DIOA + 0x000A

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	DIR1	DIR0

BAR + CMP_OFFSET_DIOA + 0x000B

D15	D14	D13	D12	D11	D10	D9	D8
0	0	0	0	0	0	0	0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
DIR1	rw	0	Sets the direction of the upper byte of DIO register 0 = DIO[15:8] set as inputs (default) 1 = DIO[15:8] set as outputs
DIR0	rw	0	Sets the direction of the lower byte of DIO register 0 = DIO[7:0] set as inputs (default) 1 = DIO[7:0] set as outputs

Description

Digital I/O Direction register

Example

7.11.5 Digital Inputs and Outputs Pending (0x0C)

future feature

7.11.6 Digital Inputs and Outputs Clear (0x0E)

Future feature

7.11.7 Digital Inputs and Outputs Input Config CH[15:0] (0x10-0x2F)

Digital I/O Group-A Input configuration register.

Register Layout

Register is a 16-bit register that is read/write. This register is available for each individual isolated digital input, 16-channels in all.

BAR + CMP_OFFSET_DIOA + 0x0010 + 2 * CHANNEL_INDEX

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0

BAR + CMP_OFFSET_DIOA + 0x0011 + 2 * CHANNEL_INDEX

D15	D14	D13	D12	D11	D10	D9	D8
POLARITY	0	0	0	0	0	0	0

Where: 0 <= CHANNEL_INDEX <= 15

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
POLARITY	rw	0	Isolated digital input polarity 0 = non-inverted (i.e. signal => '1', otherwise => '0') 1 = inverted (i.e. signal => '0', otherwise => '1');

Description

Digital I/O Group-A Input configuration register.

Example

7.12 Digital Inputs and Outupts (DIOB, 0x8835)

Refer to Digital Inputs and Outputs DIOA component. Replace CMP_OFFSET_DIOA with CMP_OFFSET_DIOB. Refer to Component Offset summary (see page 28).

7.13 Counter Timer Array 9513 Similar (CT, 0x803A)

The counter timer array component lives to the spirit of the classic CTS9513 / AM9513 counter/timer. This implementation is not an exact replica but an enhanced and expanded version.

Features different from classic CTS9513-2 / AM9513:

- Registers are not indexed
- There is no FOUT
- All counters are implemented nearly the same
- An alarm register for every counter

Todo

- Create timing diagrams of all modes
- Describe in register logic

Notes

The CTS9513-2 datasheet indicates that mode V does not work, well we have it functioning here!

7.13.1 Counter Timer Array Mode Summary

MODE (per CTS9513-2 datasheet)	CM[15:13] Gate Control	CM[7:5] GATE RELOAD REPETITION	Description
A	000	000	Mode A - Software triggered strobe with no gating
B	Level	000	Mode B - Software triggered strobe with level gating
C	Edge	000	Mode C - Hardware triggered strobe
D	000	001	Mode D - Rate generator with no hardware gating
E	Level	001	Mode E - Rate Generator with level gating
F	Edge	001	Mode F - Non-retriggerable one shot
G	000	010	Mode G - Software triggered delayed pulse one-shot
H	Level	010	Mode H - Software triggered delayed pulse one-shot with hardware gating
I	Edge	010	Mode I - Hardware triggered delayed pulse strobe
J	000	011	Mode J - Variable duty cycle rate generator with no hardware gating
K	Level	011	Mode K - Variable duty cycle rate generator with level gating
L	Edge	011	Mode L - Hardware triggered delayed pulse one-shot
N	Level	100	Mode N - Software triggered strobe with level gating and hardware retriggering
O	Edge	100	Mode O - Software triggered strobe with edge gating and hardware retriggering
Q	Level	101	Mode Q - Rate generator with synchronization
R	Edge	101	Mode R - Retriggerable one-shot
S	000	110	Mode S - Gate controlled strobe
V	000	111	Mode V - Frequency shift keying

7.13.2 Counter Timer Array I/O Summary

GATE(n) inputs from J9 connector

GATE - As referenced within Register set definition	PIN at J9
GATE(0)	15
GATE(1)	16
GATE(2)	17

GATE(3)	18
GATE(4)	19
GATE(5)	20
GATE(6)	23
GATE(7)	24
GATE(8)	25
GATE(9)	26

SOURCE(n) inputs from J9 connector

SOURCE - As referenced within Register set definition	PIN at J9
SOURCE(0)	3
SOURCE(1)	4
SOURCE(2)	5
SOURCE(3)	6
SOURCE(4)	7
SOURCE(5)	8
SOURCE(6)	9
SOURCE(7)	10
SOURCE(8)	13
SOURCE(9)	14

OUT(n) from each counter to J9 connector outputs

OUT - As referenced within Register set definition	PIN at J9
OUT(0)	29
OUT(1)	30
OUT(2)	33
OUT(3)	34
OUT(4)	35
OUT(5)	36
OUT(6)	37
OUT(7)	38
OUT(8)	39
OUT(9)	40

Terminal Count from each counter

TC	Counter Index
TC(0)	0
TC(1)	1
TC(2)	2
TC(3)	3
TC(4)	4
TC(5)	5
TC(6)	6
TC(7)	7
TC(8)	8
TC(9)	9

FREQUENCY(n) from internal time base

FREQUENCE(n) - As referenced within Register set definition	FREQUENCY
FREQUENCY(0)	10 MHz
FREQUENCY(1)	5 MHz
FREQUENCY(2)	2 MHz
FREQUENCY(3)	1 MHz
FREQUENCY(4)	500 KHz

7.13.3 Counter Timer Array ID (0x0)

This register indicates the ID of this component

Register Layout

Register is a 16-bit register that is read only.

BAR + CMP_OFFSET_CT + 0x0000

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_CT + 0x0001

D15	D14	D13	D12	D11	D10	D9	D8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
CT Component ID	r	0x803A	0x803A

Description

This register indicates the ID of this component.

Example

7.13.4 Counter Timer Array Status Output (0x0C)

Counter Timer Array Output Status register

Register Layout

Register is a 16-bit register that is read only. Reports the states of all counter OUT values.

Reading the MSB captures all 10 values simultaneously.

BAR + CMP_OFFSET_CT + 0x000C

D7	D6	D5	D4	D3	D2	D1	D0
OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0

BAR + CMP_OFFSET_CT + 0x000D

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	OUT9	OUT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	r	-	Don't Care
OUT[9:0]	r	0	Reports of the state of the OUTputs (i.e. J9 OUT[9:0] signals).

Description

Counter Timer Array Output Status register

Example

7.13.5 Counter Timer Array Status Compare (0x0E)

Counter Timer Array Output Status register

Register Layout

Register is a 16-bit register that is read only. Reports the states of the compare between the counter value and alarm register. When the count equals alarm, the value will be '1' otherwise '0'.

Reading the MSB captures all 10 values simultaneously.

BAR + CMP_OFFSET_CT + 0x000C

D7	D6	D5	D4	D3	D2	D1	D0
CMP7	CMP6	CMP5	CMP4	CMP3	CMP2	CMP1	CMP0

BAR + CMP_OFFSET_CT + 0x000D

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CMP9	CMP8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	r	-	Don't Care
CMP[9:0]	r	0	Reports of the state of the compare (i.e. Counter value == Alarm Register).

Description

Counter Timer Array Output Status register

Example

7.13.6 Counter Timer Array Counter Mode Register 0 (0x10)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(1) 011 = Active high GATE(n-1) = GATE(9) 100 = Active high GATE(0) 101 = Active low GATE(0) 110 = GATE(0) rising edge 111 = GATE(0) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(9) 0001 = SOURCE(0) 0010 = SOURCE(1) 0011 = SOURCE(2) 0100 = SOURCE(3) 0101 = SOURCE(4) 0110 = GATE(0) 0111 = GATE(1) 1000 = GATE(2) 1001 = GATE(3) 1010 = GATE(4) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.7 Counter Timer Array Counter Mode Register 1 (0x12)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0012

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0013

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(0) 010 = Active High Level, GATE(n+1) = GATE(2) 011 = Active high GATE(n-1) = GATE(0) 100 = Active high GATE(1) 101 = Active low GATE(1) 110 = GATE(1) rising edge 111 = GATE(1) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(0) 0001 = SOURCE(0) 0010 = SOURCE(1) 0011 = SOURCE(2) 0100 = SOURCE(3) 0101 = SOURCE(4) 0110 = GATE(0) 0111 = GATE(1) 1000 = GATE(2) 1001 = GATE(3) 1010 = GATE(4) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.8 Counter Timer Array Counter Mode Register 2 (0x14)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(3) 011 = Active high GATE(n-1) = GATE(1) 100 = Active high GATE(2) 101 = Active low GATE(2) 110 = GATE(2) rising edge 111 = GATE(2) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(1) 0001 = SOURCE(0) 0010 = SOURCE(1) 0011 = SOURCE(2) 0100 = SOURCE(3) 0101 = SOURCE(4) 0110 = GATE(0) 0111 = GATE(1) 1000 = GATE(2) 1001 = GATE(3) 1010 = GATE(4) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.9 Counter Timer Array Counter Mode Register 3 (0x16)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(4) 011 = Active high GATE(n-1) = GATE(2) 100 = Active high GATE(3) 101 = Active low GATE(3) 110 = GATE(3) rising edge 111 = GATE(3) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(2) 0001 = SOURCE(0) 0010 = SOURCE(1) 0011 = SOURCE(2) 0100 = SOURCE(3) 0101 = SOURCE(4) 0110 = GATE(0) 0111 = GATE(1) 1000 = GATE(2) 1001 = GATE(3) 1010 = GATE(4) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.10 Counter Timer Array Counter Mode Register 4 (0x18)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(5) 011 = Active high GATE(n-1) = GATE(3) 100 = Active high GATE(4) 101 = Active low GATE(4) 110 = GATE(4) rising edge 111 = GATE(4) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(3) 0001 = SOURCE(0) 0010 = SOURCE(1) 0011 = SOURCE(2) 0100 = SOURCE(3) 0101 = SOURCE(4) 0110 = GATE(0) 0111 = GATE(1) 1000 = GATE(2) 1001 = GATE(3) 1010 = GATE(4) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.11 Counter Timer Array Counter Mode Register 5 (0x1A)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(6) 011 = Active high GATE(n-1) = GATE(4) 100 = Active high GATE(5) 101 = Active low GATE(5) 110 = GATE(5) rising edge 111 = GATE(5) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(4) 0001 = SOURCE(5) 0010 = SOURCE(6) 0011 = SOURCE(7) 0100 = SOURCE(8) 0101 = SOURCE(9) 0110 = GATE(5) 0111 = GATE(6) 1000 = GATE(7) 1001 = GATE(8) 1010 = GATE(9) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.12 Counter Timer Array Counter Mode Register 6 (0x1C)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(7) 011 = Active high GATE(n-1) = GATE(5) 100 = Active high GATE(6) 101 = Active low GATE(6) 110 = GATE(6) rising edge 111 = GATE(6) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(5) 0001 = SOURCE(5) 0010 = SOURCE(6) 0011 = SOURCE(7) 0100 = SOURCE(8) 0101 = SOURCE(9) 0110 = GATE(5) 0111 = GATE(6) 1000 = GATE(7) 1001 = GATE(8) 1010 = GATE(9) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.13 Counter Timer Array Counter Mode Register 7 (0x1E)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(8) 011 = Active high GATE(n-1) = GATE(6) 100 = Active high GATE(7) 101 = Active low GATE(7) 110 = GATE(7) rising edge 111 = GATE(7) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(6) 0001 = SOURCE(5) 0010 = SOURCE(6) 0011 = SOURCE(7) 0100 = SOURCE(8) 0101 = SOURCE(9) 0110 = GATE(5) 0111 = GATE(6) 1000 = GATE(7) 1001 = GATE(8) 1010 = GATE(9) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.14 Counter Timer Array Counter Mode Register 8 (0x20)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(9) 011 = Active high GATE(n-1) = GATE(7) 100 = Active high GATE(8) 101 = Active low GATE(8) 110 = GATE(8) rising edge 111 = GATE(8) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(7) 0001 = SOURCE(5) 0010 = SOURCE(6) 0011 = SOURCE(7) 0100 = SOURCE(8) 0101 = SOURCE(9) 0110 = GATE(5) 0111 = GATE(6) 1000 = GATE(7) 1001 = GATE(8) 1010 = GATE(9) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.15 Counter Timer Array Counter Mode Register 9 (0x22)

Counter Timer Counter Mode Register (CM) register.

Register Layout

Register is a 16-bit register that is read/write.

This register is similar to the classic 9513.

BAR + CMP_OFFSET_CT + 0x0010

D7	D6	D5	D4	D3	D2	D1	D0
GATE_MODE	RELOAD	REPEAT	0	DIRECTION	OUT_SEL2	OUT_SEL1	OUT_SEL0

BAR + CMP_OFFSET_CT + 0x0010

D15	D14	D13	D12	D11	D10	D9	D8
GATE_SEL2	GATE_SEL1	GATE_SEL0	SRC_EDGE	SRC_SEL3	SRC_SEL2	SRC_SEL1	SRC_SEL0

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	rw	X	Don't care
GATE_SEL[2:0]	rw	000	000 = no gating 001 = Active high, TC(n-1) = TC(9) 010 = Active High Level, GATE(n+1) = GATE(0) 011 = Active high GATE(n-1) = GATE(8) 100 = Active high GATE(9) 101 = Active low GATE(9) 110 = GATE(9) rising edge 111 = GATE(9) falling edge
SRC_EDGE	rw	0	0 = Rising 1 = Falling

SRC_SEL[3:0]	rw	0000	0000 = TC(8) 0001 = SOURCE(5) 0010 = SOURCE(6) 0011 = SOURCE(7) 0100 = SOURCE(8) 0101 = SOURCE(9) 0110 = GATE(5) 0111 = GATE(6) 1000 = GATE(7) 1001 = GATE(8) 1010 = GATE(9) 1011 = FREQUENCY(0) 1100 = FREQUENCY(1) 1101 = FREQUENCY(2) 1110 = FREQUENCY(3) 1111 = FREQUENCY(4)
GATE_MODE	rw	0	0 = Off 1 = On
RELOAD	rw	0	0 = Load 1 = Both
REPEAT	rw	0	0 = Once 1 = Repeat
DIRECTION	rw	0	0 = Down 1 = Up
OUT_SEL[2:0]	rw	000	000 = Inactive, output low 001 = Active high on TC 010 = TC Toggled Non-inverted 011 = TC Toggled Inverted 100 = Active high on TC 101 = Active low on TC 110 = Compare (ALARM == COUNTER) non-inverted 111 = Compare (ALARM == COUNTER) Inverted

Description

Counter Timer Counter Mode Register (CM) register.

Example

7.13.16 Counter Timer Array Load Register CH[9:0] (0x24-0x37)

Counter Timer Array Load register

Register Layout

Register is a 16-bit register that is read/write.

Writing the MSB will simultaneously commit the values to the counter load register.

BAR + CMP_OFFSET_CT + 0x0024 + 2 * CHANNEL_INDEX

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_CT + 0x0025 + 2 * CHANNEL_INDEX

D15	D14	D13	D12	D11	D10	D9	D8

Where: 0 <= CHANNEL_INDEX <= 9

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
LOAD[15:0]	rw	0x0000	

Description

Counter Timer Array Load register

Example

7

7.13.17 Counter Timer Array Hold Register CH[9:0] (0x38-0x4B)

Counter Timer Array Hold register

Register Layout

Register is a 16-bit register that is read/write.

Writing the MSB will simultaneously commit the values to the counter load register.

BAR + CMP_OFFSET_CT + 0x0038 + 2 * CHANNEL_INDEX

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_CT + 0x0039 + 2 * CHANNEL_INDEX

D15	D14	D13	D12	D11	D10	D9	D8

Where: 0 <= CHANNEL_INDEX <= 9

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
HOLD[15:0]	rw	0x0000	

Description

Counter Timer Array Hold register

Example

7.13.18 Counter Timer Array Alarm Register CH[9:0] (0x4C-0x5F)

Counter Timer Array Alarm register

Register Layout

Register is a 16-bit register that is read/write.

Writing the MSB will simultaneously commit the values to the counter load register.

BAR + CMP_OFFSET_CT + 0x004C + 2 * CHANNEL_INDEX

D7	D6	D5	D4	D3	D2	D1	D0

BAR + CMP_OFFSET_CT + 0x004D + 2 * CHANNEL_INDEX

D15	D14	D13	D12	D11	D10	D9	D8

Where: 0 <= CHANNEL_INDEX <= 9

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
ALARM[15:0]	rw	0x0000	

Description

Counter Timer Array Alarm register

Example

7.13.19 Counter Timer Array Command ARM (0x60)

Counter Timer Array Command ARM register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0060

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0061

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the ARM command to be issued to those counter timers.

Description

Counter Timer Array Command ARM register

Example

7.13.20 Counter Timer Array Command LOAD (0x62)

Counter Timer Array Command LOAD register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0062

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0063

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the LOAD command to be issued to those counter timers.

Description

Counter Timer Array Command LOAD register

Example

7.13.21 Counter Timer Array Command LOAD ARM (0x64)

Counter Timer Array Command LOAD and ARM register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0064

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0065

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the LOAD and ARM command to be issued to those counter timers all at the same time.

Description

Counter Timer Array Command LOAD and ARM register

Example

7

7.13.22 Counter Timer Array Command DISARM SAVE (0x66)

Counter Timer Array Command DISARM and SAVE register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0066

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0067

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the DISARM and SAVE command to be issued to those counter timers all at the same time. The SAVE command will wirte current counter data to the hold register for subsequent readout.

Description

Counter Timer Array Command DISARM and SAVE register

Example

7.13.23 Counter Timer Array Command SAVE (0x68)

Counter Timer Array Command SAVE register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0068

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0069

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the SAVE command to be issued to those counter timers. The SAVE command will wirte current counter data to the hold register for subsequent readout.

Description

Counter Timer Array Command SAVE register

Example

7.13.24 Counter Timer Array Command DISARM (0x6A)

Counter Timer Array Command DISARM register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x006A

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x006B

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the DISARM command to be issued to those counter timers.

Description

Counter Timer Array Command DISARM register

Example

7.13.25 Counter Timer Array Command OUT CLEAR (0x6C)

Counter Timer Array Command OUT CLEAR register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x006C

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x006D

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the counter output to be cleared (i.e. OUT[9:0] at J9)

Description

Counter Timer Array Command OUT CLEAR register

Example

7.13.26 Counter Timer Array Command OUT SET (0x6E)

Counter Timer Array Command OUT SET register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x006E

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x006F

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the counter output to be set (i.e. OUT[9:0] at J9)

Description

Counter Timer Array Command OUT SET register

Example

7.13.27 Counter Timer Array Command STEP (0x70)

Counter Timer Array Command STEP register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0070

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0071

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the counter to advance (increment/decrement); same as issuing a SOURCE input.

Description

Counter Timer Array Command STEP register

Example

7.13.28 Counter Timer Array Command RESET (0x72)

Counter Timer Array Command RESET register

Register Layout

Register is a 16-bit register that is write only

BAR + CMP_OFFSET_CT + 0x0070

D7	D6	D5	D4	D3	D2	D1	D0
CT7	CT6	CT5	CT4	CT3	CT2	CT1	CT0

BAR + CMP_OFFSET_CT + 0x0071

D15	D14	D13	D12	D11	D10	D9	D8
X	X	X	X	X	X	CT9	CT8

Bit Definitions

NAME	DIRECTION	DEFAULT	DESCRIPTION
X	w	-	Don't Care
CT[9:0]	w	-	Writing a '1' to each corresponding CT bit will cause the counter to advance (increment/decrement); same as issuing a SOURCE input.

Description

Counter Timer Array Command RESET register

Example

8 Specifications

8.1 Board

8.2 Digital I/O (DIOA & DIOB & CT, J8 & J9)

Applies to Connectors (see page 11)	J8 Digital I/O group A and group B J9 Counter Timer Array
Voltage range:	0 - 5 volts or 0 - 3.3 volts based on bias jumper (J4, J5, J7)
Voltage output high minimum (VOH) - Bias set to 5V	3.8 volts @ -24 mA
Voltage output high minimum (VOH) - Bias set to 3.3V	2.4 volts @ -24 mA
Voltage output low maximum (VOL) - Bias set to 5V	0.55 volts
Voltage output low maximum (VOL) - Bias set to 3.3V	0.55 volts
Voltage input high minimum (VIH) - Bias set to 5V	3.5 volts
Voltage input high minimum (VIH) - Bias set to 3.3V	2 votls
Voltage input low maximum (VIL) - Bias set to 5V	1.5 volts
Voltage input low maximum (VIL) - Bias set to 3.3V	0.8 volts

8.3 Isolated Digital Inputs (J15)

Number of channels	sixteen
Type	Isolated digital input
Voltage range:	±3 to ±100 VDC
Isolation	500 volts
Input resistance	2K ohms or greater, current limited to approximately 1 mA
Response time	2 ms

8.4 Isolated Digital Outputs (J16)

Number of inputs	Sixteen
Type	Isolated digital output, high-side switch
Voltage range:	5 to 30 VDC
Isolation	500 volts
Current max	1 amps
Turn-on time	2 ms or faster
Turn-off time	2 ms or faster

8.5 Analog Outputs (J23)

Number of channels	Four
Resolution	16-bits (1/65536 of full scale) Monotonicity guaranteed

Voltage ranges	0-5V (reset default) 0-10V ±5V ±10V Each channel independently configurable
Output current per channel	+/- 5 milliamps
Offset error	Less than 16 LSB
Gain error	Adjustable to 0 LSB
Differential non-linearity	± 1 LSB max
Settle time	10 microseconds
integral non-linearity	± 1 LSB max

8.6 SL2-100 Interface CMS only (J24)

Number of channels	one
Differential pairs	RS422 compatible

8.7 XY2-100 Interface

Number of channels	one
Differential pair	RS422 compatible

8.8 Quadrature Encoder Inputs

Number of channels	two
Type	
Voltage range:	RS422 compatible

9 FPGA Firmware

1. Optional step. It is possible to install the Lattice programming software on a computer different from the target. Once the PCIe board has been programmed, the target will need to be power cycled for the update to take effect.

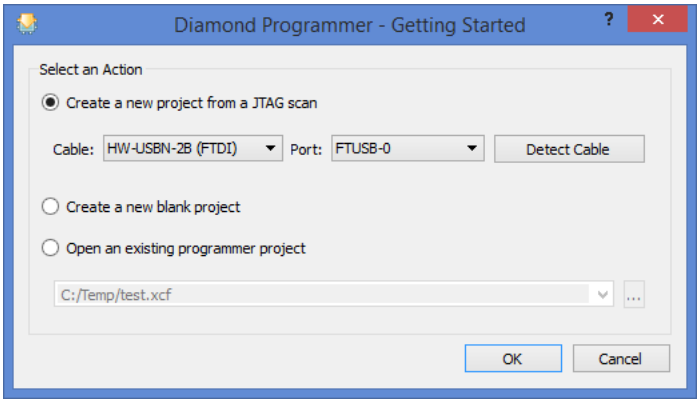
Power down computer and install the AES_PCIE-A04-D1064-16RS485-CT10 board. Power up the computer.

2. Download and install Lattice Diamond Programmer Standalone.

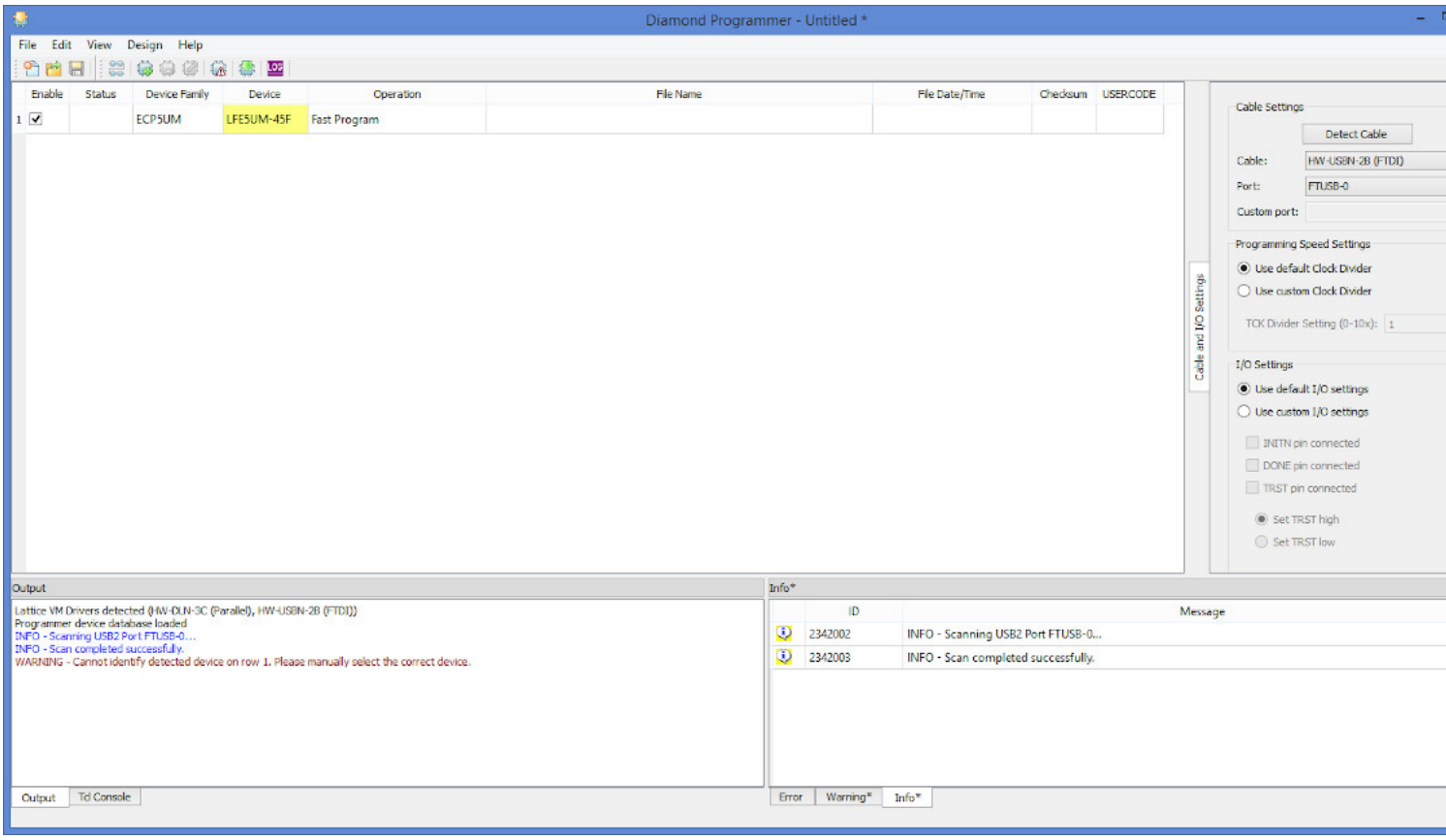
3. Connect USB cable between the PCIe board and to PCIe board and computer. Note that the computer that performs the FPGA firmware update can be different than the PCIe target.

4. Run Lattice Diamond Programmer

4.a. The first screen that will be presented is the "Getting Started" screen. Select "Create a new project from a JTAG scan".



4.b. Once a scan is complete you will be presented with the following screen.



4.c. Moving the cursor into the "file name" area and clicking you will want to click on the elipse "..." to select the bit file for download.

4.d. Next, click on the "Device" and you will be presented with a pull-down. Select "LFE5UM-45F".

4.e. Double click on "Fast Program" under Operation. You will be presented with a screen "ECP5UM - LFE5UM-45F - Device Properties".

4.f. In the "General" tab under "Access mode:" click on the pull-down and select "SPI Flash Background Programming". The window will change.

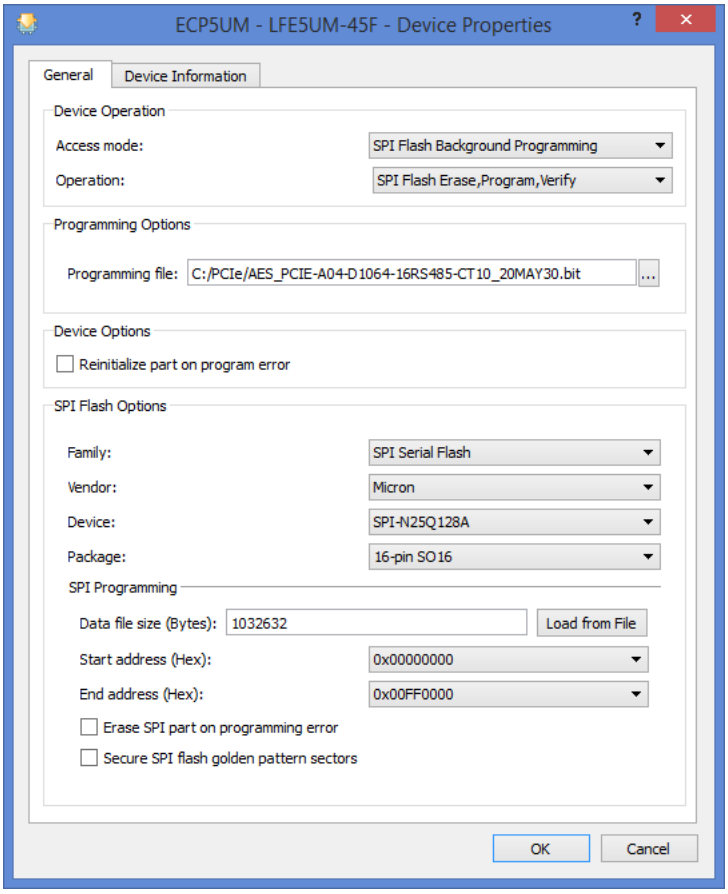
4.g. Set up the window as shown below. Note that the "Programming file:" will have already been populated with the bit file you selected earlier. Once setup properly click OK.

4.g.1. Change SPI flash options Family to "SPI Serial Flash"

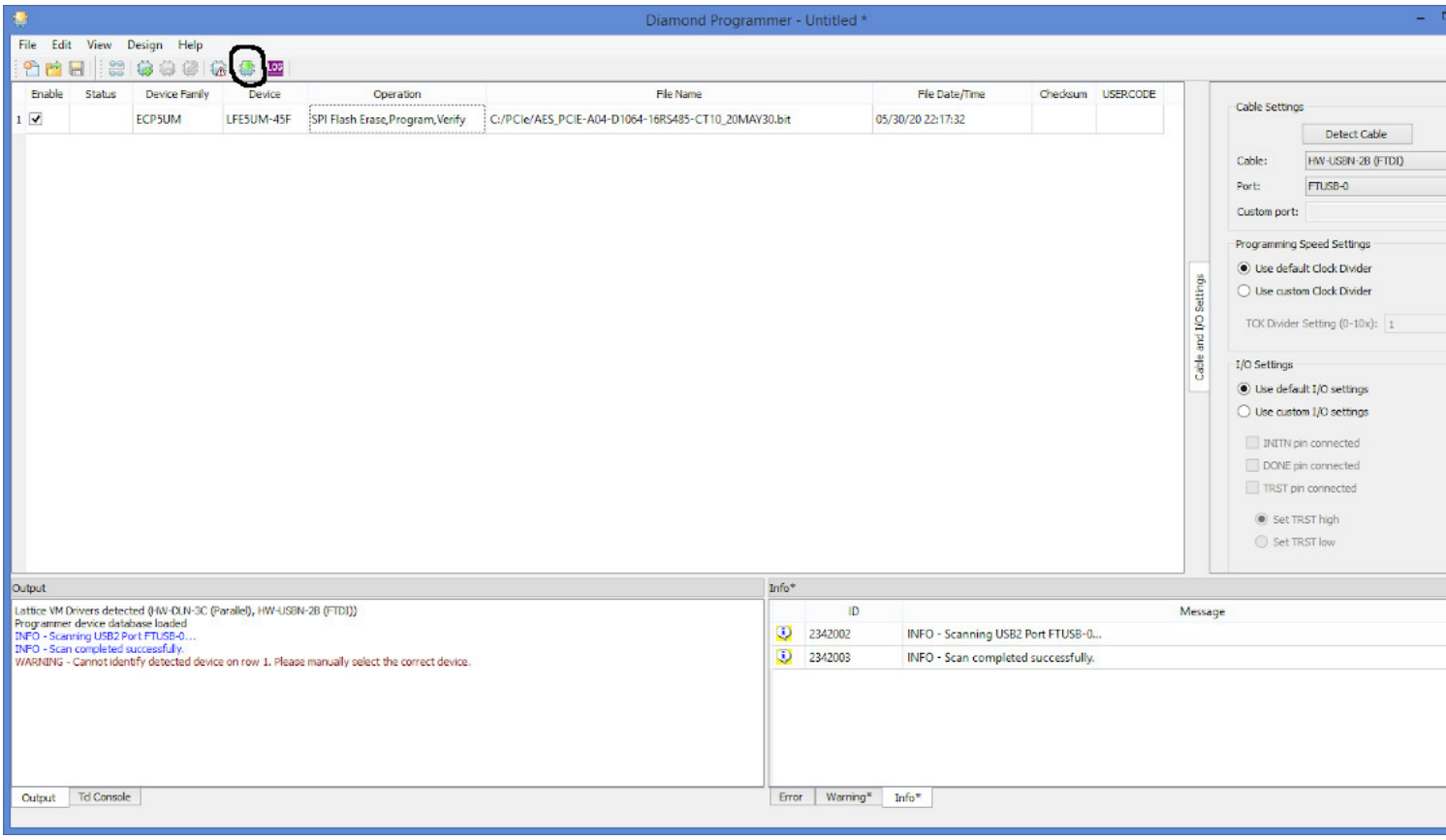
4.g.2. Change SPI flash options Vendor to "Micron"

4.g.3. Change SPI flash options Device to "SPI-N25Q128A"

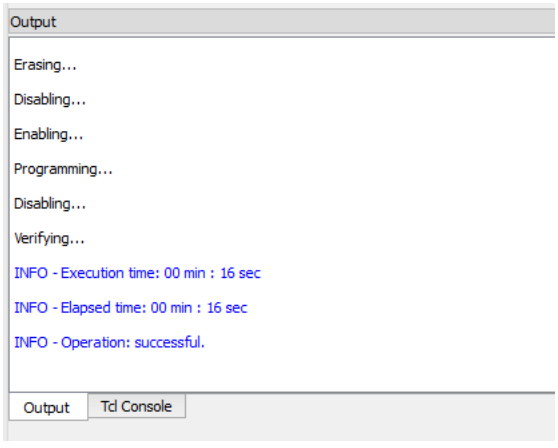
4.g.4. Change SPI flash options Package to "16-pin SO16"



4.h. The software is ready to program the FPGA. Click on the programming icon as shown circled below.



4.i Once programming is complete the Output box in the window (lower left) will display "INFO - Operation Successful".



4.j. Programming is now complete. In order for the new update to take affect you will need to power cycle the PCIe board.

10 Software Functions

Software was written in Eclipses and compiled using GCC.

Ubuntu 20.04 was the operating system used to test the code.

The software is written in application space using sysfs and mmap as shown here: https://github.com/billfarrow/pcimem

Notes

We are in the process of cleaning up the code but are providing it here so that you have something to get a feel for complexity.

More details to follow.

Symbol Reference

Name	Description
Functions (see page 127)	The following table lists functions in this documentation.
Files (see page 147)	The following table lists files in this documentation.

10.1 Symbol Reference

10.1.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Write_U16 (see page 128)	
	Read_U16 (see page 129)	
	ReadWrite_Init (see page 129)	
	ReadWrite_Term (see page 130)	
	PCIe_AO_All_Set (see page 130)	
	PCIe_AO_Status_Get (see page 131)	
	PCIe_AO_Value_Cycle (see page 131)	
	PCIe_AO_Value_Set (see page 132)	
	PCIe_Component_IDs (see page 132)	
	PCIe_Component_Offsets (see page 134)	
	PCIe_CT_Config (see page 134)	

	PCIe_DIO_Loopback (see page 135)	
	PCIe_IDI_All_Get (see page 137)	
	PCIe_IDO_All_Set (see page 138)	
	PCIe_IDO_Value_Set (see page 138)	
	PCIe_QEA_Config_Set (see page 139)	
	PCIe_QEA_Count_Get (see page 139)	
	PCIe_QEB_Config_Set (see page 140)	
	PCIe_QEB_Count_Get (see page 141)	
	PCIe_Reg_Read (see page 141)	
	PCIe_Reg_Write (see page 142)	
	PCIe_SL2_Config_Set (see page 142)	
	PCIe_SL2_Status_Get (see page 143)	
	PCIe_SL2_X_Set (see page 143)	
	PCIe_SL2_Y_Set (see page 143)	
	PCIe_XY2_Config_Set (see page 144)	
	PCIe_XY2_Status_Get (see page 144)	
	PCIe_XY2_X_Set (see page 145)	
	PCIe_XY2_Y_Set (see page 145)	
	PCIe_XY2_Z_Set (see page 146)	

Legend

	Method

10.1.1.1 Write_U16 Function

C++

```
void Write_U16(int offset, uint16_t value);
```

File

test.c

Body Source

```
void Write_U16( int offset, uint16_t value )
{
 off_t target, target_base;
 void * virt_addr;

 target   = offset + BRD_OFFSET;
    target_base = target & ~(sysconf(_SC_PAGE_SIZE)-1);

    virt_addr = g_map_base + target - target_base;
 *((uint16_t *) virt_addr) = value;
}
```

10.1.1.2 Read_U16 Function

C++

```
uint16_t Read_U16(int offset);
```

File

test.c

Body Source

```
uint16_t Read_U16( int offset )
{
 off_t target, target_base;
 void * virt_addr;
 uint16_t value;

 target   = offset + BRD_OFFSET;
    target_base = target & ~(sysconf(_SC_PAGE_SIZE)-1);

    virt_addr = g_map_base + target - target_base;
 value = *((uint16_t *) virt_addr);
 return value;
}
```

10.1.1.3 ReadWrite_Init Function

C++

```
int ReadWrite_Init(int * fd);
```

File

test.c

Body Source

```
int ReadWrite_Init( int * fd )
{
 int f;
 int map_size = 4096UL;
 int items_count = 1;  /* just one read/write transfer (for now) */
 int type_width  = 2;  /* half word or uint16_t */
 off_t target, target_base;

 target = BRD_OFFSET;

 f = open(device_name, O_RDWR | O_SYNC);

    if( f == -1)
    {
     PRINT_ERROR;
    }
    *fd = f;
    printf("%s opened.\n", device_name);
    printf("Target offset is 0x%x, page size is %ld\n", (int) target, sysconf(_SC_PAGE_SIZE));
```

```
      fflush(stdout);

   target_base = target & ~(sysconf(_SC_PAGE_SIZE)-1);
      if (target + items_count*type_width - target_base > map_size)
      {
       map_size = target + items_count*type_width - target_base;
      }

      g_map_base = mmap(0, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, f, target_base);
      if(g_map_base == (void *) -1)
      {
       PRINT_ERROR;
      }
      printf("PCI Memory mapped to address 0x%08lx.\n", (unsigned long) g_map_base);
      fflush(stdout);

      return 0;
   }
```

10.1.1.4 ReadWrite_Term Function

C++

```
int ReadWrite_Term(int * fd);
```

File

test.c

Body Source

```
int ReadWrite_Term( int * fd )
{
 close( *fd );
 return 0;
}
```

10.1.1.5 PCIe_AO_All_Set Function

C++

```
int PCIe_AO_All_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_AO_All_Set( struct menu_command_params * p )
{
 u_int16_t value;
 u_int16_t range;
 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "Range (0,1,2,3) = " );
 range = Menu_Ask_Uint16( stdin, stdout, question_str );
```

```
 if ( range > 3 ) range = 3;
 Write_U16( AO_REG_CFGA, range );
 Write_U16( AO_REG_CFGB, range );
 Write_U16( AO_REG_CFGC, range );
 Write_U16( AO_REG_CFGD, range );

 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "AO = " );
 value = Menu_Ask_Uint16( stdin, stdout, question_str );
 Write_U16( AO_REG_QA, value );
 Write_U16( AO_REG_QB, value );
 Write_U16( AO_REG_QC, value );
 Write_U16( AO_REG_QD, value );
 return SUCCESS;
}
```

10.1.1.6 PCIe_AO_Status_Get Function

C++

```
int PCIe_AO_Status_Get(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_AO_Status_Get( struct menu_command_params * p )
{
 u_int16_t reg_val;

 reg_val = Read_U16( AO_REG_STATUS );
 printf( " AO.STATUS = 0x%4X\n  ", (int)reg_val );
 return SUCCESS;
}
```

10.1.1.7 PCIe_AO_Value_Cycle Function

C++

```
int PCIe_AO_Value_Cycle(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_AO_Value_Cycle( struct menu_command_params * p )
{
 Write_U16( AO_REG_QA, 0x0000 );
 Write_U16( AO_REG_QA, 0x0000 );
 Write_U16( AO_REG_QA, 0xFFFF );
 Write_U16( AO_REG_QA, 0xFFFF );
 Write_U16( AO_REG_QA, 0x0000 );
 Write_U16( AO_REG_QA, 0x0000 );
 printf( "PCIe_AO_Value_Cycle - done\n" );
```

```
 return SUCCESS;
}
```

10.1.1.8 PCIe_AO_Value_Set Function

C++

```
int PCIe_AO_Value_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_AO_Value_Set( struct menu_command_params * p )
{
 u_int16_t value;
 char   question_str[LOCAL_STRING_BUFFER_SIZE];
 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "AO[%d] = ", p->channel );
 value = Menu_Ask_Uint16( stdin, stdout, question_str );

 switch( p->channel )
 {
 case 0:
  Write_U16( AO_REG_QA, value );
  break;
 case 1:
  Write_U16( AO_REG_QB, value );
  break;
 case 2:
  Write_U16( AO_REG_QC, value );
  break;
 case 3:
  Write_U16( AO_REG_QD, value );
  break;
 }
 return SUCCESS;
}
```

10.1.1.9 PCIe_Component_IDs Function

C++

```
int PCIe_Component_IDs(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_Component_IDs( struct menu_command_params * p )
{
 int  offset;
 u_int16_t id, tag_id, offset_component, offset_max;
```

```
  printf( "SCANNING REGISTER SPACE:\n" );

  offset_component = Read_U16( SYS_REG_COMP_OFFSET );
  offset_max       = Read_U16( SYS_REG_OFFSET_MAX  );

  tag_id = Read_U16( SYS_REG_TAG_A );
  for (offset = 0; offset < offset_max; offset = offset + offset_component )
  {
   id = Read_U16( offset );
   tag_id = Read_U16( SYS_REG_TAG_A );
   if ( 0 != ( 0x7FFF & tag_id ) )
   {
    printf( " ID 0x%X found at offset 0x%X.  tag = 0x%X\n", (int) id, offset, (int) tag_id );
   }
  }

  printf( "Board ID = %d.  Build version = %d\n", (int) Read_U16( SYS_REG_BRD_ID), (int)
Read_U16(SYS_REG_BUILD_VERSION) );



  printf( "COMPONENT ID LISTING:\n" );
  id = Read_U16( SYS_REG_ID ); printf( " SYS:  0x%X\n", id );
  id = Read_U16( IDI_REG_ID ); printf( " IDI:  0x%X\n", id );
  id = Read_U16( IDO_REG_ID ); printf( " IDO:  0x%X\n", id );
  id = Read_U16( DIOA_REG_ID ); printf( " DIOA: 0x%X\n", id );
  id = Read_U16( DIOB_REG_ID ); printf( " DIOB: 0x%X\n", id );
  id = Read_U16( QEA_REG_ID ); printf( " QEA:  0x%X\n", id );
  id = Read_U16( QEB_REG_ID ); printf( " QEB:  0x%X\n", id );
  id = Read_U16( XY2_REG_ID ); printf( " XY2:  0x%X\n", id );
  id = Read_U16( AO_REG_ID );  printf( " AO:   0x%X\n", id );
  id = Read_U16( SL2_REG_ID ); printf( " SL2:  0x%X\n", id );
  id = Read_U16( CT_REG_ID );  printf( " CT:   0x%X\n", id );

  printf( "COMPONENT OFFSETS:\n" );
  printf( " SYS_OFFSET   0x%X\n", SYS_OFFSET   );
  printf( " IDI_OFFSET   0x%X\n", IDI_OFFSET   );
  printf( " IDO_OFFSET   0x%X\n", IDO_OFFSET   );
  printf( " XY2_OFFSET   0x%X\n", XY2_OFFSET   );
  printf( " DIOA_OFFSET  0x%X\n", DIOA_OFFSET  );
  printf( " DIOB_OFFSET  0x%X\n", DIOB_OFFSET  );
  printf( " QEA_OFFSET   0x%X\n", QEA_OFFSET   );
  printf( " QEB_OFFSET   0x%X\n", QEB_OFFSET   );
  printf( " AO_OFFSET    0x%X\n", AO_OFFSET    );
  printf( " AOCAL_OFFSET 0x%X\n", AOCAL_OFFSET );
  printf( " SL2_OFFSET   0x%X\n", SL2_OFFSET   );
  printf( " CT_OFFSET    0x%X\n", CT_OFFSET    );

  tag_id = Read_U16( SYS_REG_TAG_A );
  printf( "SYS TAG LIST = 0x%X\n", (int) tag_id );
  tag_id = Read_U16( SYS_REG_TAG_A );
  printf( "SYS TAG LIST = 0x%X\n", (int) tag_id );
  printf( "\n" );

 /*
 const int SYS_CODE  = 0x803B;
 const int IDI_CODE     = 0x8033;
 const int IDO_CODE     = 0x8034;
 const int DIOA_CODE   = 0x8035;
 const int DIOB_CODE   = 0x8835;
 const int QEA_CODE  = 0x8036;
 const int QEB_CODE  = 0x8836;
 const int XY2_CODE  = 0x8037;
 const int AO_CODE        = 0x8038;
```

```
const int SL2_CODE       = 0x8039;
const int CT_CODE        = 0x803A;
*/
 return SUCCESS;
}
```

10.1.1.10 PCIe_Component_Offsets Function

C++

```
int PCIe_Component_Offsets(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_Component_Offsets( struct menu_command_params * p )
{
 printf( "SYS_OFFSET   0x%X\n", SYS_OFFSET   );
 printf( "IDI_OFFSET   0x%X\n", IDI_OFFSET   );
 printf( "IDO_OFFSET   0x%X\n", IDO_OFFSET   );
 printf( "XY2_OFFSET   0x%X\n", XY2_OFFSET   );
 printf( "DIOA_OFFSET  0x%X\n", DIOA_OFFSET  );
 printf( "DIOB_OFFSET  0x%X\n", DIOB_OFFSET  );
 printf( "QEA_OFFSET   0x%X\n", QEA_OFFSET   );
 printf( "QEB_OFFSET   0x%X\n", QEB_OFFSET   );
 printf( "AO_OFFSET    0x%X\n", AO_OFFSET    );
 printf( "AOCAL_OFFSET 0x%X\n", AOCAL_OFFSET );
 printf( "SL2_OFFSET   0x%X\n", SL2_OFFSET   );
 printf( "CT_OFFSET    0x%X\n", CT_OFFSET    );
 return SUCCESS;
}
```

10.1.1.11 PCIe_CT_Config Function

C++

```
int PCIe_CT_Config(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_CT_Config( struct menu_command_params * p )
{
 Write_U16( CT_REG_LOAD0,  18 );
 Write_U16( CT_REG_LOAD1,  17 );
 Write_U16( CT_REG_LOAD2,  16 );
 Write_U16( CT_REG_LOAD3,  15 );
 Write_U16( CT_REG_LOAD4,  14 );
 Write_U16( CT_REG_LOAD5,  13 );
 Write_U16( CT_REG_LOAD6,  12 );
 Write_U16( CT_REG_LOAD7,  11 );
 Write_U16( CT_REG_LOAD8,  10 );
```

```
 Write_U16( CT_REG_LOAD9,   9 );

 Write_U16( CT_REG_HOLD0,   2 );
 Write_U16( CT_REG_HOLD1,   3 );
 Write_U16( CT_REG_HOLD2,   4 );
 Write_U16( CT_REG_HOLD3,   5 );
 Write_U16( CT_REG_HOLD4,   6 );
 Write_U16( CT_REG_HOLD5,   7 );
 Write_U16( CT_REG_HOLD6,   8 );
 Write_U16( CT_REG_HOLD7,   9 );
 Write_U16( CT_REG_HOLD8,  10 );
 Write_U16( CT_REG_HOLD9,  11 );

 /* MODE J, SRC=F1 10MHz, OUT=Toggle */
 Write_U16( CT_REG_CMR0,  0x0B62 );
 Write_U16( CT_REG_CMR1,  0x0B62 );
 Write_U16( CT_REG_CMR2,  0x0B62 );
 Write_U16( CT_REG_CMR3,  0x0B62 );
 Write_U16( CT_REG_CMR4,  0x0B62 );
 Write_U16( CT_REG_CMR5,  0x0B62 );
 Write_U16( CT_REG_CMR6,  0x0B62 );
 Write_U16( CT_REG_CMR7,  0x0B62 );
 Write_U16( CT_REG_CMR8,  0x0B62 );
 Write_U16( CT_REG_CMR9,  0x0B62 );

 Write_U16( CT_REG_IO_CFG,    0x0005 );
 Write_U16( CT_REG_IO_GATE,   0      );
 Write_U16( CT_REG_IO_SOURCE, 0      );

 Write_U16( CT_REG_CMD_LOAD_ARM, 0x003FF );

 return SUCCESS;
}
```

10.1.1.12 PCIe_DIO_Loopback Function

C++

```
int PCIe_DIO_Loopback(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_DIO_Loopback( struct menu_command_params * p )
{
 int index;
 u_int16_t dia, dib, mask;

 Write_U16( DIOA_REG_DIR, 0 );
 Write_U16( DIOA_REG_DO,  0 );
 dia = Read_U16( DIOA_REG_DI );

 Write_U16( DIOB_REG_DIR, 0 );
 Write_U16( DIOB_REG_DO,  0 );
 dib = Read_U16( DIOB_REG_DI );

 printf( "DIOA INIT: dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib );
```

```
 Write_U16( DIOA_REG_DIR, 1 );
 dia = Read_U16( DIOA_REG_DI );
 dib = Read_U16( DIOB_REG_DI );
 printf( "DIOA.DIR[1:0]= \"01\": dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib
);

 Write_U16( DIOA_REG_DIR, 3 );
 dia = Read_U16( DIOA_REG_DI );
 dib = Read_U16( DIOB_REG_DI );
 printf( "DIOA.DIR[1:0]= \"11\": dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib
);

 mask = 0x8000;
 for ( index = 0; index < 16; index++ )
 {
  Write_U16( DIOA_REG_DO,  mask );
  dia = Read_U16( DIOA_REG_DI );
  dib = Read_U16( DIOB_REG_DI );
  printf( "DIOA.DO = 0x%04X: dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) mask, (int) dia,
(int) dib );
  mask = mask >> 1;
 }
 Write_U16( DIOA_REG_DO,  0xFFFF );
 Write_U16( DIOB_REG_DO,  0xFFFF );
 dia = Read_U16( DIOA_REG_DI );
 dib = Read_U16( DIOB_REG_DI );
 printf( "DIOA TERM: dioa.di = 0x%04X.  dib = 0x%04X\n", (int) dia, (int) dib );

 Write_U16( DIOA_REG_DIR, 0 );
 Write_U16( DIOA_REG_DO,  0 );
 dia = Read_U16( DIOA_REG_DI );

 Write_U16( DIOB_REG_DIR, 0 );
 Write_U16( DIOB_REG_DO,  0 );
 dib = Read_U16( DIOB_REG_DI );

 printf( "DIOA TERM: dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib );


 Write_U16( DIOA_REG_DIR, 0 );
 Write_U16( DIOA_REG_DO,  0 );
 dia = Read_U16( DIOB_REG_DI );

 Write_U16( DIOB_REG_DIR, 0 );
 Write_U16( DIOB_REG_DO,  0 );
 dib = Read_U16( DIOB_REG_DI );

 printf( "DIOB INIT: dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib );

 Write_U16( DIOB_REG_DIR, 1 );
 dia = Read_U16( DIOA_REG_DI );
 dib = Read_U16( DIOB_REG_DI );
 printf( "DIOB.DIR[1:0]= \"01\": dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib
);

 Write_U16( DIOB_REG_DIR, 3 );
 dia = Read_U16( DIOA_REG_DI );
 dib = Read_U16( DIOB_REG_DI );
 printf( "DIOB.DIR[1:0]= \"11\": dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib
);

 mask = 0x8000;
 for ( index = 0; index < 16; index++ )
 {
```

```
   Write_U16( DIOB_REG_DO,  mask );
   dia = Read_U16( DIOA_REG_DI );
   dib = Read_U16( DIOB_REG_DI );
   printf( "DIOB.DO = 0x%04X: dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) mask, (int) dia,
(int) dib );
   mask = mask >> 1;
  }
  Write_U16( DIOA_REG_DO,  0xFFFF );
  Write_U16( DIOB_REG_DO,  0xFFFF );
  dia = Read_U16( DIOA_REG_DI );
  dib = Read_U16( DIOB_REG_DI );
  printf( "DIOA TERM: dioa.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib );

  Write_U16( DIOA_REG_DIR, 0 );
  Write_U16( DIOA_REG_DO,  0 );
  dia = Read_U16( DIOA_REG_DI );

  Write_U16( DIOB_REG_DIR, 0 );
  Write_U16( DIOB_REG_DO,  0 );
  dib = Read_U16( DIOB_REG_DI );

  printf( "DIOB TERM: diao.di = 0x%04X.  diob.di = 0x%04X\n", (int) dia, (int) dib );

  return SUCCESS;
}
```

10.1.1.13 PCIe_IDI_All_Get Function

C++

```
int PCIe_IDI_All_Get(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_IDI_All_Get( struct menu_command_params * p )
{
 int index;
 u_int16_t value;
 u_int16_t scratch;
 u_int16_t mask = 0x8000;

 value = Read_U16( IDI_REG_DI );

 printf( " IDI[15:0] = 0x%04X    ", value );

 for ( index = 0; index < 16; index++ )
 {
  if ( 0 == ( index % 4 ) ) printf( " " );

  scratch = value & mask;
  if ( 0 == scratch )
  {
   printf( "0" );
  }
  else
  {
   printf( "1" );
```

```
    }
    mask = mask >> 1;
   }

  #if 0
   switch( g_end_line_method )
   {
    case 0:  printf( "\n" );  break;
    case 1:  printf( "\r" );  break;
    case 2:  break;
   }
  #endif

   printf( "\n" );
   return SUCCESS;
  }
```

10.1.1.14 PCIe_IDO_All_Set Function

C++

```
int PCIe_IDO_All_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_IDO_All_Set( struct menu_command_params * p )
{
 u_int16_t value;
 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "IDO[all] = " );
 value = Menu_Ask_Uint16( stdin, stdout, question_str );

 Write_U16( IDO_REG_DOUT, value );
 return SUCCESS;
}
```

10.1.1.15 PCIe_IDO_Value_Set Function

C++

```
int PCIe_IDO_Value_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_IDO_Value_Set( struct menu_command_params * p )
{
 u_int16_t value;
 u_int16_t channel;
```

```
 u_int16_t reg_val;
 u_int16_t mask;
 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "IDO channel (0-15 = " );
 channel = Menu_Ask_Uint16( stdin, stdout, question_str );

 if ( channel > 15 ) channel = 15;

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "IDO[%d] = ", channel  );
 value = Menu_Ask_Uint16( stdin, stdout, question_str );

 reg_val = Read_U16( IDO_REG_DOUT );
 mask = 0x0001 << channel;

 if ( 0 == value )
 {
  reg_val = reg_val & ~mask;
 }
 else
 {
  reg_val = reg_val | mask;
 }

 Write_U16( IDO_REG_DOUT, reg_val );
 return SUCCESS;
}
```

10.1.1.16 PCIe_QEA_Config_Set Function

C++

```
int PCIe_QEA_Config_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_QEA_Config_Set( struct menu_command_params * p )
{
 Write_U16( QEA_REG_CFG_A, 0x0000 );
 Write_U16( QEA_REG_CFG_B, 0x0000 );
 Write_U16( QEA_REG_CFG_Z, 0x0000 );
 Write_U16( QEA_REG_CFG,   0x0003 );
 Write_U16( QEA_REG_COMPARE_L, 0x1000 );
 Write_U16( QEA_REG_COMPARE_H, 0x0000 );
 printf( "PCIe_QEA_Config_Set - done\n" );
 return SUCCESS;
}
```

10.1.1.17 PCIe_QEA_Count_Get Function

C++

```
int PCIe_QEA_Count_Get(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_QEA_Count_Get( struct menu_command_params * p )
{
 u_int16_t lo, hi;
 u_int32_t uval;

 lo = Read_U16( QEA_REG_OUTX_L );
 hi = Read_U16( QEA_REG_OUTX_H );
 uval = (u_int32_t)( lo );
 uval = uval | ( (u_int32_t)(hi << 16) );
 printf( "X = 0x%08X", (int) uval );

 lo = Read_U16( QEA_REG_OUTY_L );
 hi = Read_U16( QEA_REG_OUTY_H );
 uval = (u_int32_t)( lo );
 uval = uval | ( (u_int32_t)(hi << 16) );
 printf( "  Y = 0x%08X", (int) uval );

 lo = Read_U16( QEA_REG_STATUS );
 printf( "  Status = 0x%4X", (int) lo );
 printf( "\n" );
 return SUCCESS;
}
```

10.1.1.18 PCIe_QEB_Config_Set Function

C++

```
int PCIe_QEB_Config_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_QEB_Config_Set( struct menu_command_params * p )
{
 Write_U16( QEB_REG_CFG_A, 0x0000 );
 Write_U16( QEB_REG_CFG_B, 0x0000 );
 Write_U16( QEB_REG_CFG_Z, 0x0000 );
 Write_U16( QEB_REG_CFG,   0x8003 );
 Write_U16( QEB_REG_COMPARE_L, 0x1000 );
 Write_U16( QEB_REG_COMPARE_H, 0x0000 );
 printf( "PCIe_QEB_Config_Set - done\n" );
 return SUCCESS;
}
```

10.1.1.19 PCIe_QEB_Count_Get Function

C++

```
int PCIe_QEB_Count_Get(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_QEB_Count_Get( struct menu_command_params * p )
{
 u_int16_t lo, hi;
 u_int32_t uval;

 lo = Read_U16( QEB_REG_OUTX_L );
 hi = Read_U16( QEB_REG_OUTX_H );
 uval = (u_int32_t)( lo );
 uval = uval | ( (u_int32_t)(hi << 16) );
 printf( "X = 0x%08X", (int) uval );

 lo = Read_U16( QEB_REG_OUTY_L );
 hi = Read_U16( QEB_REG_OUTY_H );
 uval = (u_int32_t)( lo );
 uval = uval | ( (u_int32_t)(hi << 16) );
 printf( "  Y = 0x%08X", (int) uval );

 lo = Read_U16( QEB_REG_STATUS );
 printf( "  Status = 0x%4X", (int) lo );
 printf( "\n" );
 return SUCCESS;
}
```

10.1.1.20 PCIe_Reg_Read Function

C++

```
int PCIe_Reg_Read(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_Reg_Read( struct menu_command_params * p )
{
 u_int32_t offset;
 u_int16_t value;
 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "register offset = " );
 offset = Menu_Ask_Uint32( stdin, stdout, question_str );
```

```
 value = Read_U16( offset );
 printf( "Read: [0x%X] => 0x%X\n", (int) offset, (int) value );
 return SUCCESS;
}
```

10.1.1.21 PCIe_Reg_Write Function

C++

```
int PCIe_Reg_Write(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_Reg_Write( struct menu_command_params * p )
{
 u_int32_t offset;
 u_int16_t value;
 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "register offset = " );
 offset = Menu_Ask_Uint32( stdin, stdout, question_str );

 Menu_Display_Clear_Line( stdout );
 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "value = " );
 value = Menu_Ask_Uint16( stdin, stdout, question_str );

 Write_U16( offset, value );
 printf( "Wrote: [0x%X] <= 0x%X\n", (int) offset, (int) value );
 return SUCCESS;
}
```

10.1.1.22 PCIe_SL2_Config_Set Function

C++

```
int PCIe_SL2_Config_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_SL2_Config_Set( struct menu_command_params * p )
{
 return SUCCESS;
}
```

10.1.1.23 PCIe_SL2_Status_Get Function

C++

```
int PCIe_SL2_Status_Get(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_SL2_Status_Get( struct menu_command_params * p )
{
 return SUCCESS;
}
```

10.1.1.24 PCIe_SL2_X_Set Function

C++

```
int PCIe_SL2_X_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_SL2_X_Set( struct menu_command_params * p )
{
 u_int32_t value;
 u_int16_t lo, hi;

 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "X (32-bit) = " );
 value = Menu_Ask_Uint32( stdin, stdout, question_str );

 lo = (u_int16_t)( value & 0xFFFF );
 hi = (u_int16_t)( ( value >> 16 ) & 0xFFFF );

 Write_U16( SL2_REG_XL, lo );
 Write_U16( SL2_REG_XH, hi );
 return SUCCESS;
}
```

10.1.1.25 PCIe_SL2_Y_Set Function

C++

```
int PCIe_SL2_Y_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_SL2_Y_Set( struct menu_command_params * p )
{
 u_int32_t value;
 u_int16_t lo, hi;

 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "Y (32-bit) = " );
 value = Menu_Ask_Uint32( stdin, stdout, question_str );

 lo = (u_int16_t)( value & 0xFFFF );
 hi = (u_int16_t)( ( value >> 16 ) & 0xFFFF );

 Write_U16( SL2_REG_YL, lo );
 Write_U16( SL2_REG_YH, hi );
 return SUCCESS;
}
```

10.1.1.26 PCIe_XY2_Config_Set Function

C++

```
int PCIe_XY2_Config_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_XY2_Config_Set( struct menu_command_params * p )
{
 //CFG[15]=active
 //CFG[9;8]= update upon write to 00=z, 01=y, 10=z, 11=none
 //CFG[1]= CPHA
 //CFG[0]=CPOL
 //CFG[2]=TX_PARITY

 //CFG[14:12]=clock.  000=2MHz, 001=4MHz, 010=1MHz, 011=500KHz, 1XX=none/off.
 Write_U16( XY2_REG_CFG, 0x8000 );
 printf( "PCIe_XY2_Config_Set - done\n" );
 return SUCCESS;
}
```

10.1.1.27 PCIe_XY2_Status_Get Function

C++

```
int PCIe_XY2_Status_Get(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_XY2_Status_Get( struct menu_command_params * p )
{
 u_int32_t value;
 u_int16_t lo, hi;

 lo = Read_U16( XY2_REG_SL );
 hi = Read_U16( XY2_REG_SH );

 value = ( ( (u_int32_t) hi ) << 16 ) | ( (u_int32_t) lo );

 printf( "XY2 Status = 0x%X\n", value );
 return SUCCESS;
}
```

10.1.1.28 PCIe_XY2_X_Set Function

C++

```
int PCIe_XY2_X_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_XY2_X_Set( struct menu_command_params * p )
{
 u_int32_t value;
 u_int16_t lo, hi;

 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "X (32-bit) = " );
 value = Menu_Ask_Uint32( stdin, stdout, question_str );

 lo = (u_int16_t)( value & 0xFFFF );
 hi = (u_int16_t)( ( value >> 16 ) & 0xFFFF );

 Write_U16( XY2_REG_XL, lo );
 Write_U16( XY2_REG_XH, hi );
 return SUCCESS;
}
```

10.1.1.29 PCIe_XY2_Y_Set Function

C++

```
int PCIe_XY2_Y_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_XY2_Y_Set( struct menu_command_params * p )
{
 u_int32_t value;
 u_int16_t lo, hi;

 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "Y (32-bit) = " );
 value = Menu_Ask_Uint32( stdin, stdout, question_str );

 lo = (u_int16_t)( value & 0xFFFF );
 hi = (u_int16_t)( ( value >> 16 ) & 0xFFFF );

 Write_U16( XY2_REG_YL, lo );
 Write_U16( XY2_REG_YH, hi );
 return SUCCESS;
}
```

10.1.1.30 PCIe_XY2_Z_Set Function

C++

```
int PCIe_XY2_Z_Set(struct menu_command_params * p);
```

File

test.c

Body Source

```
int PCIe_XY2_Z_Set( struct menu_command_params * p )
{
 u_int32_t value;
 u_int16_t lo, hi;

 char   question_str[LOCAL_STRING_BUFFER_SIZE];

 Menu_Display_Clear_Line( stdout );

 snprintf( question_str, LOCAL_STRING_BUFFER_SIZE, "Z (32-bit) = " );
 value = Menu_Ask_Uint32( stdin, stdout, question_str );

 lo = (u_int16_t)( value & 0xFFFF );
 hi = (u_int16_t)( ( value >> 16 ) & 0xFFFF );

 Write_U16( XY2_REG_ZL, lo );
 Write_U16( XY2_REG_ZH, hi );
 return SUCCESS;
}
```

10.1.2 Files

The following table lists files in this documentation.

Files

Name	Description
aes_pcie.h (see page 147)	This is file aes_pcie.h.

10.1.2.1 aes_pcie.h

This is file aes_pcie.h.

Body Source

```
#ifndef AES_PCIE_H_
#define AES_PCIE_H_

/* offset into board */
const int BRD_OFFSET = 0x0000;

/* component ID codes */
const int SYS_CODE = 0x803B;
const int IDI_CODE  = 0x8033;
const int IDO_CODE  = 0x8034;
const int DIOA_CODE  = 0x8035;
const int DIOB_CODE  = 0x8835;
const int QEA_CODE = 0x8036;
const int QEB_CODE = 0x8836;
const int XY2_CODE = 0x8037;
const int AO_CODE       = 0x8038;
const int SL2_CODE      = 0x8039;
const int CT_CODE       = 0x803A;

const int COMP_OFFSET    = 0x0100;

/* component offsets */
const int SYS_OFFSET = 0x0000;
const int IDI_OFFSET = SYS_OFFSET   + COMP_OFFSET;
const int IDO_OFFSET = IDI_OFFSET   + COMP_OFFSET;
const int XY2_OFFSET = IDO_OFFSET   + COMP_OFFSET;
const int DIOA_OFFSET = XY2_OFFSET   + COMP_OFFSET;
const int DIOB_OFFSET = DIOA_OFFSET  + COMP_OFFSET;
const int QEA_OFFSET = DIOB_OFFSET  + COMP_OFFSET;
const int QEB_OFFSET = QEA_OFFSET   + COMP_OFFSET;
const int AO_OFFSET = QEB_OFFSET   + COMP_OFFSET;
const int AOCAL_OFFSET = AO_OFFSET    + COMP_OFFSET;
const int SL2_OFFSET = AOCAL_OFFSET + COMP_OFFSET;
const int CT_OFFSET     = SL2_OFFSET   + COMP_OFFSET;


/* component register sets */

  /* System component */
const int   SYS_REG_ID  = 0x0000 + SYS_OFFSET;
const int   SYS_REG_BRD_ID  = 0x0002 + SYS_OFFSET;
```

```
const int   SYS_REG_BUILD_VERSION  = 0x0004 + SYS_OFFSET;
const int   SYS_REG_COMP_OFFSET  = 0x0006 + SYS_OFFSET;
const int   SYS_REG_OFFSET_MAX   = 0x0008 + SYS_OFFSET;
const int   SYS_REG_ERROR   = 0x0010 + SYS_OFFSET;
const int   SYS_REG_TAG_A   = 0x0014 + SYS_OFFSET;
const int   SYS_REG_TAG_B   = 0x0016 + SYS_OFFSET;
const int   SYS_REG_SCRATCH_A = 0x0020 + SYS_OFFSET;
const int   SYS_REG_SCRATCH_B = 0x0022 + SYS_OFFSET;
const int   SYS_REG_SCRATCH_C = 0x0024 + SYS_OFFSET;
const int   SYS_REG_SCRATCH_D    = 0x0026 + SYS_OFFSET;
const int   SYS_REG_CFG  = 0x0028 + SYS_OFFSET;
const int   SYS_REG_RESET      = 0x002A + SYS_OFFSET;

  /* Isolated Digital Input component */
const int   IDI_REG_ID  = 0x0000 + IDI_OFFSET;
const int   IDI_REG_DI  = 0x0002 + IDI_OFFSET;
const int   IDI_REG_PENDING  = 0x000C + IDI_OFFSET;
const int   IDI_REG_CLR      = 0x000E + IDI_OFFSET;
const int   IDI_REG_CFG0  = 0x0010 + IDI_OFFSET;
const int   IDI_REG_CFG1  = 0x0012 + IDI_OFFSET;
const int   IDI_REG_CFG2  = 0x0014 + IDI_OFFSET;
const int   IDI_REG_CFG3  = 0x0016 + IDI_OFFSET;
const int   IDI_REG_CFG4  = 0x0018 + IDI_OFFSET;
const int   IDI_REG_CFG5  = 0x001A + IDI_OFFSET;
const int   IDI_REG_CFG6  = 0x001C + IDI_OFFSET;
const int   IDI_REG_CFG7  = 0x001E + IDI_OFFSET;
const int   IDI_REG_CFG8  = 0x0020 + IDI_OFFSET;
const int   IDI_REG_CFG9  = 0x0022 + IDI_OFFSET;
const int   IDI_REG_CFG10  = 0x0024 + IDI_OFFSET;
const int   IDI_REG_CFG11  = 0x0026 + IDI_OFFSET;
const int   IDI_REG_CFG12  = 0x0028 + IDI_OFFSET;
const int   IDI_REG_CFG13  = 0x002A + IDI_OFFSET;
const int   IDI_REG_CFG14  = 0x002C + IDI_OFFSET;
const int   IDI_REG_CFG15  = 0x002E + IDI_OFFSET;
  /* Isolated Digital Output component */
const int   IDO_REG_ID  = 0x0000 + IDO_OFFSET;
const int   IDO_REG_DOUT  = 0x0002 + IDO_OFFSET;
  /* XY2 Component */
const int   XY2_REG_ID  = 0x0000 + XY2_OFFSET;
const int   XY2_REG_CFG   = 0x0002 + XY2_OFFSET;
//  --XY2_REG_CFG_X  = 0x0004 + XY2_OFFSET;
//  --XY2_REG_CFG_Y  = 0x0006 + XY2_OFFSET;
//  --XY2_REG_CFG_Z  = 0x0008 + XY2_OFFSET;
//  --XY2_REG_CFG_S  = 0x000A + XY2_OFFSET;
const int   XY2_REG_XL   = 0x0010 + XY2_OFFSET;
const int   XY2_REG_XH   = 0x0012 + XY2_OFFSET;
const int   XY2_REG_YL   = 0x0014 + XY2_OFFSET;
const int   XY2_REG_YH   = 0x0016 + XY2_OFFSET;
const int   XY2_REG_ZL   = 0x0018 + XY2_OFFSET;
const int   XY2_REG_ZH   = 0x001A + XY2_OFFSET;
const int   XY2_REG_SL  = 0x001C + XY2_OFFSET;
const int   XY2_REG_SH  = 0x001E + XY2_OFFSET;
/************** Digital I/O Channel A Component */
const int   DIOA_REG_ID  = 0x0000 + DIOA_OFFSET;
const int   DIOA_REG_DI  = 0x0002 + DIOA_OFFSET;
const int   DIOA_REG_DO       = 0x0004 + DIOA_OFFSET;
const int   DIOA_REG_DIR  = 0x000A + DIOA_OFFSET;
const int   DIOA_REG_PENDING  = 0x000C + DIOA_OFFSET;
const int   DIOA_REG_CLR      = 0x000E + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG0 = 0x0010 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG1 = 0x0012 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG2 = 0x0014 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG3 = 0x0016 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG4 = 0x0018 + DIOA_OFFSET;
```

```
const int   DIOA_REG_DI_CFG5 = 0x001A + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG6 = 0x001C + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG7 = 0x001E + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG8 = 0x0020 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG9 = 0x0022 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG10 = 0x0024 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG11 = 0x0026 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG12 = 0x0028 + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG13 = 0x002A + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG14 = 0x002C + DIOA_OFFSET;
const int   DIOA_REG_DI_CFG15 = 0x002E + DIOA_OFFSET;
  /************** Digital I/O Channel B Component */
const int   DIOB_REG_ID  = 0x0000 + DIOB_OFFSET;
const int   DIOB_REG_DI  = 0x0002 + DIOB_OFFSET;
const int   DIOB_REG_DO       = 0x0004 + DIOB_OFFSET;
const int   DIOB_REG_DIR  = 0x000A + DIOB_OFFSET;
const int   DIOB_REG_PENDING  = 0x000C + DIOB_OFFSET;
const int   DIOB_REG_CLR      = 0x000E + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG0 = 0x0010 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG1 = 0x0012 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG2 = 0x0014 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG3 = 0x0016 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG4 = 0x0018 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG5 = 0x001A + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG6 = 0x001C + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG7 = 0x001E + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG8 = 0x0020 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG9 = 0x0022 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG10 = 0x0024 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG11 = 0x0026 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG12 = 0x0028 + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG13 = 0x002A + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG14 = 0x002C + DIOB_OFFSET;
const int   DIOB_REG_DI_CFG15 = 0x002E + DIOB_OFFSET;
  /************** Quadrature encoder Channel A Component */
const int   QEA_REG_ID  = 0x0000 + QEA_OFFSET;
const int   QEA_REG_CFG  = 0x0002 + QEA_OFFSET;  //-- CFG[3:0]=MODE[3:0].
CFG[7:4]=COUNTER_MODE[3:0].  CFG[11:8]=Z_MODE[3:0] (index).  CFG[15:12]=event to interrupts?
const int   QEA_REG_CFG_A   = 0x0004 + QEA_OFFSET;
const int   QEA_REG_CFG_B   = 0x0006 + QEA_OFFSET;
const int   QEA_REG_CFG_Z   = 0x0008 + QEA_OFFSET;
const int   QEA_REG_COMPARE_L = 0x0010 + QEA_OFFSET;
const int   QEA_REG_COMPARE_H = 0x0012 + QEA_OFFSET;
const int   QEA_REG_LOAD_L   = 0x0014 + QEA_OFFSET;
const int   QEA_REG_LOAD_H   = 0x0016 + QEA_OFFSET;
const int   QEA_REG_OUTX_L   = 0x0018 + QEA_OFFSET;
const int   QEA_REG_OUTX_H   = 0x001A + QEA_OFFSET;
const int   QEA_REG_OUTY_L   = 0x001C + QEA_OFFSET;
const int   QEA_REG_OUTY_H   = 0x001E + QEA_OFFSET;
const int   QEA_REG_TIME_L   = 0x0020 + QEA_OFFSET;
const int   QEA_REG_TIME_H   = 0x0022 + QEA_OFFSET;
const int   QEA_REG_STATUS   = 0x0024 + QEA_OFFSET; //-- writing '1' to any bit location will
acknowledge or clear any pending posted events.
  /************** Quadrature encoder Channel B Component */
const int   QEB_REG_ID   = 0x0000 + QEB_OFFSET;
const int   QEB_REG_CFG   = 0x0002 + QEB_OFFSET;  //-- CFG[3:0]=MODE[3:0].
CFG[7:4]=COUNTER_MODE[3:0].  CFG[11:8]=Z_MODE[3:0] (index).  CFG[15:12]=event to interrupts?
const int   QEB_REG_CFG_A   = 0x0004 + QEB_OFFSET;
const int   QEB_REG_CFG_B   = 0x0006 + QEB_OFFSET;
const int   QEB_REG_CFG_Z   = 0x0008 + QEB_OFFSET;
const int   QEB_REG_COMPARE_L = 0x0010 + QEB_OFFSET;
const int   QEB_REG_COMPARE_H = 0x0012 + QEB_OFFSET;
const int   QEB_REG_LOAD_L   = 0x0014 + QEB_OFFSET;
const int   QEB_REG_LOAD_H   = 0x0016 + QEB_OFFSET;
```

```
const int  QEB_REG_OUTX_L   = 0x0018 + QEB_OFFSET;
const int  QEB_REG_OUTX_H   = 0x001A + QEB_OFFSET;
const int  QEB_REG_OUTY_L   = 0x001C + QEB_OFFSET;
const int  QEB_REG_OUTY_H   = 0x001E + QEB_OFFSET;
const int  QEB_REG_TIME_L   = 0x0020 + QEB_OFFSET;
const int  QEB_REG_TIME_H   = 0x0022 + QEB_OFFSET;
const int  QEB_REG_STATUS   = 0x0024 + QEB_OFFSET; //-- writing '1' to any bit location will
acknowledge or clear any pending posted events.
  /************* Analog Output quad channels Component */
const int  AO_REG_ID  = 0x0000 + AO_OFFSET;
const int  AO_REG_CFGA  = 0x0002 + AO_OFFSET;
const int  AO_REG_CFGB   = 0x0004 + AO_OFFSET;
const int  AO_REG_CFGC   = 0x0006 + AO_OFFSET;
const int  AO_REG_CFGD   = 0x0008 + AO_OFFSET;
const int  AO_REG_STATUS  = 0x000E + AO_OFFSET;
const int  AO_REG_QA   = 0x0010 + AO_OFFSET;
const int  AO_REG_QB   = 0x0012 + AO_OFFSET;
const int  AO_REG_QC    = 0x0014 + AO_OFFSET;
const int  AO_REG_QD    = 0x0016 + AO_OFFSET;
const int  AO_REG_UPDATE  = 0x0018 + AO_OFFSET;
const int  AO_REG_CLEAR  = 0x001A + AO_OFFSET; //-- writing to this register clears the DACs
  /************* SL2-100 Component */
const int  SL2_REG_ID   = 0x0000 + SL2_OFFSET;
const int  SL2_REG_CFG   = 0x0002 + SL2_OFFSET;  //-- sync out select.  laser on/off?
const int  SL2_REG_XL    = 0x0004 + SL2_OFFSET;
const int  SL2_REG_XH    = 0x0006 + SL2_OFFSET;
const int  SL2_REG_YL    = 0x0008 + SL2_OFFSET;
const int  SL2_REG_YH    = 0x000A + SL2_OFFSET;
const int  SL2_REG_STATUS   = 0x000C + SL2_OFFSET;
  /************* COUTNER/TIMER Component */
const int  CT_REG_ID    = 0x0000 + CT_OFFSET;
const int  CT_REG_STATUS_OUT  = 0x000C + CT_OFFSET;
const int  CT_REG_STATUS_COMPARE = 0x000E + CT_OFFSET;
const int  CT_REG_CMR0   = 0x0010 + CT_OFFSET;
const int  CT_REG_CMR1   = 0x0012 + CT_OFFSET;
const int  CT_REG_CMR2   = 0x0014 + CT_OFFSET;
const int  CT_REG_CMR3   = 0x0016 + CT_OFFSET;
const int  CT_REG_CMR4   = 0x0018 + CT_OFFSET;
const int  CT_REG_CMR5   = 0x001A + CT_OFFSET;
const int  CT_REG_CMR6   = 0x001C + CT_OFFSET;
const int  CT_REG_CMR7   = 0x001E + CT_OFFSET;
const int  CT_REG_CMR8   = 0x0020 + CT_OFFSET;
const int  CT_REG_CMR9   = 0x0022 + CT_OFFSET;
const int  CT_REG_LOAD0   = 0x0024 + CT_OFFSET;
const int  CT_REG_LOAD1   = 0x0026 + CT_OFFSET;
const int  CT_REG_LOAD2   = 0x0028 + CT_OFFSET;
const int  CT_REG_LOAD3   = 0x002A + CT_OFFSET;
const int  CT_REG_LOAD4   = 0x002C + CT_OFFSET;
const int  CT_REG_LOAD5   = 0x002E + CT_OFFSET;
const int  CT_REG_LOAD6   = 0x0030 + CT_OFFSET;
const int  CT_REG_LOAD7   = 0x0032 + CT_OFFSET;
const int  CT_REG_LOAD8   = 0x0034 + CT_OFFSET;
const int  CT_REG_LOAD9   = 0x0036 + CT_OFFSET;
const int  CT_REG_HOLD0   = 0x0038 + CT_OFFSET;
const int  CT_REG_HOLD1   = 0x003A + CT_OFFSET;
const int  CT_REG_HOLD2   = 0x003C + CT_OFFSET;
const int  CT_REG_HOLD3   = 0x003E + CT_OFFSET;
const int  CT_REG_HOLD4   = 0x0040 + CT_OFFSET;
const int  CT_REG_HOLD5   = 0x0042 + CT_OFFSET;
const int  CT_REG_HOLD6   = 0x0044 + CT_OFFSET;
const int  CT_REG_HOLD7   = 0x0046 + CT_OFFSET;
const int  CT_REG_HOLD8   = 0x0048 + CT_OFFSET;
const int  CT_REG_HOLD9   = 0x004A + CT_OFFSET;
const int  CT_REG_ALARM0   = 0x004C + CT_OFFSET;
```

```
const int  CT_REG_ALARM1   = 0x004E + CT_OFFSET;
const int  CT_REG_ALARM2   = 0x0050 + CT_OFFSET;
const int  CT_REG_ALARM3   = 0x0052 + CT_OFFSET;
const int  CT_REG_ALARM4   = 0x0054 + CT_OFFSET;
const int  CT_REG_ALARM5   = 0x0056 + CT_OFFSET;
const int  CT_REG_ALARM6   = 0x0058 + CT_OFFSET;
const int  CT_REG_ALARM7   = 0x005A + CT_OFFSET;
const int  CT_REG_ALARM8   = 0x005C + CT_OFFSET;
const int  CT_REG_ALARM9   = 0x005E + CT_OFFSET;
const int  CT_REG_CMD_ARM   = 0x0060 + CT_OFFSET;
const int  CT_REG_CMD_LOAD   = 0x0062 + CT_OFFSET;
const int  CT_REG_CMD_LOAD_ARM  = 0x0064 + CT_OFFSET;
const int  CT_REG_CMD_DISARM_SAVE = 0x0066 + CT_OFFSET;
const int  CT_REG_CMD_SAVE   = 0x0068 + CT_OFFSET;
const int  CT_REG_CMD_DISARM  = 0x006A + CT_OFFSET;
const int  CT_REG_CMD_OUT_CLEAR = 0x006C + CT_OFFSET;
const int  CT_REG_CMD_OUT_SET  = 0x006E + CT_OFFSET;
const int  CT_REG_CMD_STEP   = 0x0070 + CT_OFFSET;
const int  CT_REG_CMD_RESET  = 0x0072 + CT_OFFSET;
const int  CT_REG_IO_CFG   = 0x0074 + CT_OFFSET;
const int  CT_REG_IO_GATE   = 0x0076 + CT_OFFSET;
const int  CT_REG_IO_SOURCE  = 0x0078 + CT_OFFSET;
const int  CT_REG_IO_OUT   = 0x007A + CT_OFFSET;
const int  CT_REG_IO_READ_GATE  = 0x007C + CT_OFFSET;
const int  CT_REG_IO_READ_SOURCE = 0x007E + CT_OFFSET;
const int  CT_REG_IO_READ_OUT  = 0x0080 + CT_OFFSET;


#endif /* AES_PCIE_H_ */
```

Index

A

B

C

D

E

F

I

L

P

Q

R

S

T

W

X