

I/O Expansion Bus and I/O Module Specifications for the RLC-XScale

*Schematics and mechanical drawings for all XScale I/O modules
are included on the RLC-CD provided in every development kit.*

Release 3.0

Copyright 2006, R.L.C. Enterprises, Inc., All Rights Reserved
R.L.C. reserves the right to change these specifications without notice.

INTRODUCTION

RLC offers a wide variety of off-the-shelf I/O expansion modules to extend the built-in functionality of the standard RLC-XScale units. This document outlines technical specifications to assist engineers in the creation of their own I/O modules. In addition to this document, we have provided complete schematics for all of our off-the-shelf I/O modules so that engineers who need to develop their own modules have working electrical models to use as examples. These documents can be found in PDF form on the RLC-CD. RLC also offers free custom I/O board engineering for customers who place quantity orders. For more information on having RLC customize I/O boards for you, contact our sales department at rlcsales@rlc.com or visit our website at <http://www.rlc.com>, and click on the 'custom' tab.

All of RLC's I/O modules are documented with electrical and mechanical schematics, which contain additional information on its device's memory offsets, default jumper settings, and other information useful in implementing your own board. Each RLC I/O module comes with its own software demo, both in Visual Basic and Visual C++. RLC includes the source code to these demos on the RLC-CD so that you can easily modify them for your own use.

THE RLC-XSCALE I/O EXPANSION BUS

RLC's I/O expansion bus on the RLC-XScale units is designed to enable engineers to easily integrate their specific I/O products with our main XScale board. The following information is intended for engineers planning to implement their own I/O device and interfacing it to the RLC-XScale units.

With the exception of the RLC-XScale-Lite, all RLC-XScale units have identical expansion buses. The physical interface is a dual row, .1" pitch, 40 pin port. It contains a subset of data, address, and control lines, which provide an easily accessible interface to external I/O devices. Digital voltage levels are LVTTTL(0-3.3VDC), but are standard TTL(0-5VDC) tolerant.

PINOUTS FOR THE RLC-XSCALE I/O EXPANSION BUS

PINS ON PORT J8	NAME	DESCRIPTION	ACTIVE STATE
1	+5V	+5V	-
2	+5V	+5V	-
3	GND	GROUND	-
4	GND	GROUND	-
5	+3.3V	+3.3V	-
6	+3.3V	+3.3V	-
7-22	BDATA00 -BDATA15	DATA BUS LINES 0-15	HIGH
23-29	BA01-BA07	ADDRESS BUS LINES 1-7	HIGH
30	/BCS_1	CHIP SELECT 1	LOW
31	/BCS_2	CHIP SELECT 2	LOW
32	RDY	DATA READY	HIGH
33	/BOE	BIT OUTPUT ENABLE	LOW
34	/BWE	BIT WRITE ENABLE	LOW
35	USER_INT1	USER INTERRUPT 1	RISING EDGE
36	USER_INT2	USER INTERRUPT 2	RISING EDGE
37	USER_INT3	USER INTERRUPT 3	RISING EDGE
38	USER_INT4	USER INTERRUPT 4	RISING EDGE
39	CPU_RES_OUT	SYSTEM RESET FROM ARM	HIGH
40	NC		-

Table 1.2 Pinouts of the RLC-XScale-Plus/Minus I/O Expansion Bus

SIGNAL DESCRIPTIONS FOR THE I/O EXPANSION BUS

DATA BUS

The data bus uses 16 lines to read and write data to the I/O devices. An 8-bit I/O device should be connected to data lines D0-D7.

ADDRESS BUS

The address bus uses lines A2-A7 and /CS1 or /CS2 in order to select a specific I/O device and its respective registers. All I/O addressing is done on 32-bit boundaries; hence address lines A0-A1 are not used. In order to access the bus the user must call specific RLC I/O functions with an appropriate offset. Specific examples of offset addressing are discussed later in this document. Below are the offset ranges which enable the bus's two chip select lines.

Chip Select	Offset Range
/CS1	0x000 - 0x0FC
/CS2	0x100 - 0x1FC

Table 1.3 Chip Select Memory Map

For 8-bit I/O addressing, all lines should be shifted by two bits, so that A0 on the I/O chip would be connected to the A2 line coming from the bus. (The A1 on the chip connects to A3 on the bus, A2 to A4, and so on . . .)

/BCS_1

An active low chip select used in conjunction with the memory address to access a specific I/O device. This line is set active low for 100ns when your call to user functions RLC_WriteUSER_IO or RLC_ReadUSER_IO contains a hex offset between 0x000 and 0x0FC. Check below for examples of I/O Read and Write user functions.

/BCS_2

An active low chip select used in conjunction with the memory address to access a specific I/O device. This line is set active low for 100ns when your call to user functions RLC_WriteUSER_IO or RLC_ReadUSER_IO contains a hex offset between 0x100 and 0x1FC. Check below for examples of I/O Read and Write user functions

RDY

Serves as an active high optional input to the CPU which extends an active chip select line until RDY is released. Chip selects are normally held low for 100ns, so in the event that an I/O device takes longer than 100ns for its acquisition, the RDY line can be used to keep a chip select active until the I/O is complete.

/BOE

This active low output line from the CPU notifies the I/O device that the CPU is ready to read its data, and that the I/O device can now latch its data onto the data bus. Attach this line to the read pin on your I/O device.

/BWE

An active low signal line used to write data to your I/O device. Attach this line to the write pin on your I/O device so that the device can latch the data into its memory.

USER_INT1-4

These four high or low edge-programmable user interrupts are available on the I/O bus so that each I/O board can easily handle up to four devices that require interrupts. Functions contained in the RLC_ARM.dll provide all calls to set-up these interrupts as events and handle them in any visual C++ application. Check the visual C++ Interrupt demo for examples of how to use these interrupts as events in your own application.

CPU_RESET_OUT

This reset line is asserted (active high) by the CPU to the expansion bus any time software or hardware controlled resets happen. The CPU_RESET_OUT stays asserted for 640ms during a hardware reset, and for 256 processor clock cycles(1.21us) for a software reset.

The RLC-XScale is able to read data into an array at approximately 2MHz. With one word data bus width, we can expect a maximum data transfer rate of $2\text{MHz} * 2\text{Bytes} = 4\text{Mbytes/second}$.

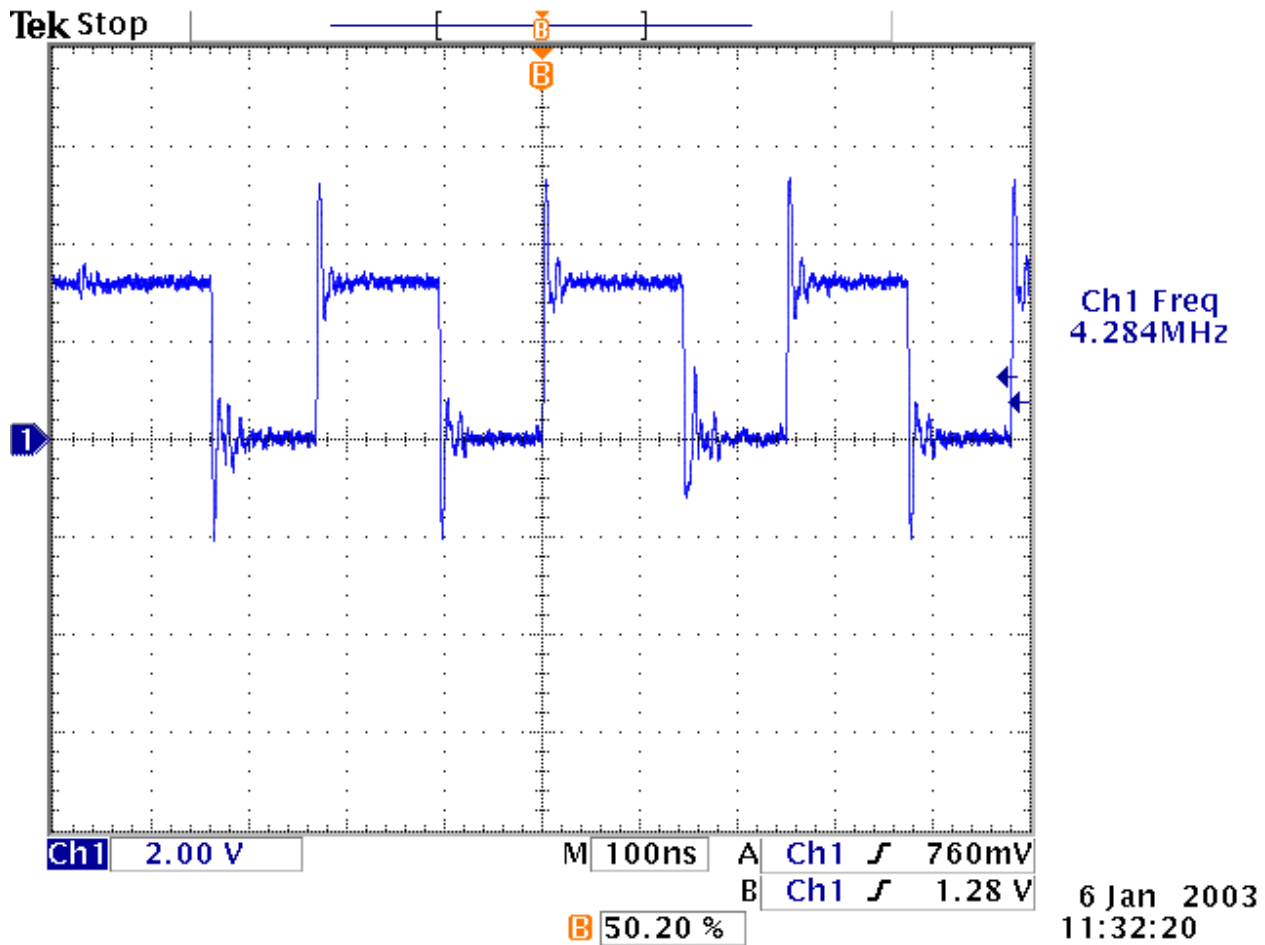
WRITE CYCLE

The maximum frequency for the write cycle was determined for the RLC-XScale with the following circumstances:

- The frequency was determined by measuring the chip select output from the GAL on the XScale-PIO-64.
- The only running processes are those booted during startup and the testing application.
- The write loop code segment is the following in VC++:

```
for (int i=0;i<1000;i++){  
    RLC_WriteUSER_IO(0x40, tempArray[i]);  
}
```

Results:



The RLC-XScale is able to write incremented data to the I/O expansion bus at approximately 4MHz. With one word data bus width, we can expect a maximum data transfer rate of $4\text{MHz} * 2\text{Bytes} = \mathbf{8\text{Mbytes/second}}$.

Disclaimer: This information is to be used as a general estimate for design guidelines. MANY other factors such as multiple processes, frequent interrupts, changes in scheduling and priority levels, etc.... will reduce available bandwidth.

FUNCTIONS FOR THE RLC-ARM EXPANSION I/O BUS:

Functions for the RLC-XScale boards are built into the Windows CE operating system, and can be called from either Visual Basic or Visual C++. These function calls are useful when designing your I/O board for use with the RLC-XScale, and will be used in your application to send and receive data and interrupts from your expansion board. Specific examples of these functions and their implementations are in the demos for the off-the-shelf modules, and are supplied on the RLC CD.

RLC_WriteUSER_IO

Description: writes a word of data to an offset address on the I/O expansion bus

Parameters: USHORT, an offset address to be written to
USHORT, a word to write to the offset

Returns: void

RLC_ReadUSER_IO

Description: reads a word of data from an offset address on the I/O expansion bus

Parameters: USHORT, an offset address for the port to be read

Returns: USHORT, a word containing the data from the read

RLC_InitUSER_INT

Description: associates a pointer handle to the user interrupt which allows the user to check this handle to see if an interrupt has occurred.

Parameters: USHORT, numbers 1-4 as the interrupt to initialize
HANDLE, the handle to be assigned to the interrupt event

Returns: void

RLC_USER_INT_Done

Description: tells the Operating System that interrupt processing has been completed for a particular USER_INT.

Parameters: USHORT, numbers 1-4 as the interrupt to disassociate with its handle

Returns: void

RLC_PollUSER_INT

Description: polls a specified interrupt line and returns its current state

Parameters: USHORT, numbers 1-4 as the interrupt line to poll

Returns: USHORT, 1 if the interrupt line is high, 0 if it is low

EXAMPLES USING I/O FUNCTIONS

Function Statement	Description
RLC_WriteUSER_IO(0x040, 0x00FF)	Activates pin /CS1 and writes 0xFF to I/O memory offset 40
RLC_WriteUSER_IO(0x144, 0x00FF)	Activates pin /CS2 and writes 0xFF to I/O memory offset 44
RLC_WriteUSER_IO(0x040, 0x00A5)	Activates pin /CS1 and writes 0xA5 to I/O memory offset 40
RLC_ReadUSER_IO(0x0F4)	Activates pin /CS1 and reads from I/O memory offset F4
RLC_ReadUSER_IO(0x1A0)	Activates pin /CS2 and reads from I/O memory offset A0
RLC_ReadUSER_IO(0x144)	Activates pin /CS2 and reads from I/O memory offset 44

Table 1.4 Read and write user examples

EXAMPLES USING INTERRUPT FUNCTIONS

Function Statement	Description
RLC_InitUSER_INT(1,hIntrEvent1,1);	Associates interrupt 1 with a handle named hIntrEvent1 to trigger on the rising edge of a signal
RLC_InitUSER_INT(1,hIntrEvent1,0);	Associates interrupt 1 with a handle named hIntrEvent1 to trigger on the falling edge of a signal
RLC_USER_INT_Done(4)	Cleans up 4th interrupt when finished
RLC_Poll_USER_INT(4);	Returns the state of the USER_INT4 pin.

Table 1.5 Examples of interrupt function calls

More information for implementing your own I/O module is available on the RLC CD. We've included the schematics of our off the shelf I/O modules which address many types of interfaces including serial, parallel, A/D, and D/A. The GPIO module is a combination of these devices, including interrupts, and serves as a great getting started point for both software and hardware design. Implementations of the above interfaces are in the XSCALE-GPIO demo, and we've included all the source code to assist you with a quick start for your own design.