

PI7C9X7958 / PI7C9X7954 / PI7C9X7952 PCIe[®] UART I/O Bridge EEPROM Programming Guide

1. Introduction

The PI7C9X7958/7954/7952 PCIe[®] UART products provide the hardware reconfiguration through Microwire compatible EEPROM. The EEPROM interface is provided for system implementation convenience. Some registers can be pre-programmed via hardware pin settings to facilitate system initialization. For programming flexibility, all of the default configuration registers can be overwritten by EEPROM data, such as sub-vendor and sub-system ID.

2. EEPROM Interface

The EEPROM interface consists of five pins: SR_DI (EEPROM data input), SR_DO (EEPROM data output), SR_CS (EEPROM chip select), SR_CLK_O (EEPROM clock output), and SR_ORG (EEPROM organization). The device may control a 93C56 or compatible parts using 2K bits. The EEPROM is used to initialize a number of registers before enumeration. This is accomplished at start-up when RTS[0] is de-asserted, at which time the data from the EEPROM is loaded. The EEPROM interface is organized into a 16-bit base, and the device supplies a 7-bit EEPROM word address.

During a reset, the device will automatically load the information/data from the EEPROM if the automatic load condition is met. The first offset in the EEPROM contains a signature. If the signature is recognized, and if RTS[0] is de-asserted, the autoload initiates right after the reset.

3. Guideline for Programming EEPROM

PI7C9X7958/54/52 supports 8-bit or 16-bit EEPROM structure. The user needs to set the hardware pin GPIO[7] correctly based on the EEPROM specification. When GPIO[7] is asserted at start-up, the EEPROM configuration data is organized in 16-bit structure. Otherwise, 8-bit structure is used.

3.1. Write Operation

The correct sequence needs to be followed to write to EEPROM in PI7C9X7958/7954/7952. Data must be written to EEPROM in this sequence: Address 0H to 1CH, and then Address 40H. There must be a delay of at least 10 ms between each Write command.

Unpredictable errors may occur if the correct write sequence is not followed or sufficient delay is not used.

3.2. Read Operation

Before using the content from EEPROM, the user needs to ensure that the value of the EEPROM Write Data Buffer is valid. The EEPROM Write Data Buffer is located at Address DCH, and it must be 12D8H. If the value is not 12D8H, the content of the EEPROM is not valid, and further read to EEPROM is prohibited. If the value of the buffer is 12D8H, the content of EEPROM is valid, and it can be read normally. There must also be a delay of at least 10 ms between each Read command.

3.3. Sample code to Read/Write EEPROM

```
#define PCIeUartEPStart      0x01
#define PCIeUartEPRdCmd     0x10
#define PCIeUartEPWrCmd     0x08
#define PCIeUartEPAddrOff   5
#define PCIeUartEPDataOff   16

// default value for PI7C9X7958
unsigned int EPDefData[15]={0xA868,
0x12D8, 0x7958, 0x12D8, 0x7958, 0x0A3E,
0x0, 0x0,
                                0x000F, 0x0,
0x0, 0x0001, 0x0200, 0x8001, 0x0};

void delay(long ns)
/* function: used to delay
*/
/* parameter: ns... the delay time, unit:
ns */
{
    long i;

    i=(ns+119)/120;
    while (i--)
        inp(0x61);
}

unsigned int eeprom_read(unsigned int
bus_dev_func_no, unsigned int eaddr)
/* function: read a word from EEPROM
*/
/* parameter: bus_dev_func_no...
bus/dev/func number for 7958 */
/*          eaddr... the addr of
eeprom which wants to read */
{
    unsigned char oldctrl;
    unsigned int data;
```

```

    unsigned int i=0;
    unsigned long epctrl;

    epctrl=PCIEUartEPRdCmd;
    // read cmd
    epctrl|=((unsigned long)epaddr<<5);
    // ep addr
    epctrl|=PCIEUartEPStart;
        // eeprom start
    writepcid(bus_dev_func_no,
PCIEEPControl, epctrl);
    // add 10 ms delay
    for (i=0;i<5000;++i)
        delay(200);
    while (((readpcib(bus_dev_func_no,
PCIEEPControl)&PCIEUartEPStart))&&(i<10
00))
    { // wait for read cycle complete
        ++i;
        delay(300);
    }
    return (readpciw(bus_dev_func_no,
PCIEEPData));
}

void eeprom_write(unsigned int
bus_dev_func_no, unsigned int epaddr,
unsigned int epdata)
/* function: write a word from EEPROM
*/
/* parameter: bus_dev_func_no...
bus/dev/func number for 7958 */
/* epaddr... the addr of
eeprom which wants to write */
/* epdata... the write data
*/
{
    unsigned char oldctrl;
    unsigned int i=0;
    unsigned long epctrl;

    epctrl=PCIEUartEPWrCmd;
        // write
cmd
    epctrl|=((unsigned long)epaddr<<
PCIEUartEPAddrOff); // ep addr
    epctrl|=((unsigned long)data)<<
PCIEUartEPDataOff); // ep data
    epctrl|=PCIEUartEPStart;
        //
eeprom start
    writepcid(bus_dev_func_no,
PCIEEPControl, epctrl);
    // add 10 ms delay
    for (i=0;i<5000;++i)
        delay(200);
    while (((readpcib(bus_dev_func_no,
PCIEEPControl)&PCIEUartEPStart))&&(i<10
00))
    { // wait for write cycle complete
        ++i;
        delay(300);
    }
}

void main()
{
    // example for read operation
    // check the value of EEPROM Write
Data Buffer
    if (readpciw(bus_dev_func_no,
PCIEEPControl)==0x12D8)
    {
        // read the eeprom from addr 0
to addr 1CH
        // 16-bit EEPROM
        for (i=0;i<=0x1C;i+=2)
        {
            eeprom_read(bus_dev_func_no,
i/2);
        }
        // 8-bit EEPROM
        for (i=0;i<=0x1C;i+=2)
        {
            eeprom_read(bus_dev_func_no,
i);
        }
    }

    // example for write operation
    // 16-bit EEPROM
    for (i=0;i<=0x1C;i+=2)
    {
        eeprom_write(bus_dev_func_no,
i/2, EPDefData[i/2]);
    }
    // write 12D8H to Addr 40H of
EEPROM
    eeprom_write(bus_dev_func_no, 0x40,
0x12d8);
    // 8-bit EEPROM
    for (i=0;i<=0x1C;i+=2)
    {
        eeprom_write(bus_dev_func_no, i,
EPDefData[i/2]);
    }
    // write 12D8H to Addr 40H of
EEPROM
    eeprom_write(bus_dev_func_no, 0x40,
0x12d8);
}

```