Notes

By Harpinder Singh

Revision History

November 15, 2000: Initial publication.

May 14, 2002: Revised document to reflect changes in the RC32334/RC32332 YC silicon.

October 18, 2002: Changed device designation to RC3233x which includes 3 devices: RC32334, RC32333, and RC32332.

PCI Satellite Mode

Instead of a Host mode of operation, the PCI bus interface for the RC3233x can have a Satellite mode of operation. The Satellite mode can be initiated in two ways:

- *The satellite can boot from the Memory Controller. In this case, the bootstrapping code for the satellite resides in the local memory space from which the satellite board boots up*
- *Or the satellite can boot from the PCI. In this case, the satellite loads its configuration registers from a serial EEPROM and then attempts to boot over the PCI bus.*

Either way, the host PCI bridge in the system is required to program the PCI Configuration Registers prior to the satellite generating or receiving any PCI cycles on the PCI bus.

To ensure the correct Satellite mode of operation, the System Controller needs to configure mem_addr[22:20] bits on reset. When mem_addr[22:20] is configured to [001], the satellite is set to boot from the Memory Controller. When mem_addr[22:20] is configured to [011], the satellite is set to boot from the PCI.

Booting from the Memory Controller

While booting from the Memory Controller, the system needs to perform the following initialization steps:

1. Configure the local boot ROM on the satellite system to:
   - *Initialize the PCI Memory/IO Space Base Registers and the PCI CPU Space Base Registers appropriately.*
   - *Set up the PCI Configuration Register Master Latency Timer, Cacheline Size, Retry Timeout, TRDY Timeout, etc.*
   - *Reset the PCI Target Not Ready bit in the PCI Arbitration Register.*
2. Configure the satellite PCI Configuration Registers using the host PCI bridge:
   - *Memory/IO Base Address Register 1 through 4 starting at offset 0x10.*
   - *Enable the Bus Master, Memory, and I/O Access in the PCI Configuration Command Register.*

Booting from the PCI Using Serial EEPROM

In PCI satellite boot from PCI mode, the satellite loads the PCI Configuration Registers from the serial EEPROM. The initialization steps are as follows:

1. Pre-program the serial EEPROM with the desired configuration register values.
2. Configure the satellite PCI Configuration Registers using the host PCI bridge.
   - *Memory/IO Base Address Register 1 through 4 starting at offset 0x10.*
   - *Enable the Bus Master, Memory, and I/O Access in the PCI Configuration Command Register.*

Once the satellite PCI interface is enabled by the host PCI bridge by writing to the Command Configuration Register, the satellite generates an Instruction Fetch cycle with the local bus physical address 0x1FC0 0000. This address is translated to the PCI bus address 0x0FC0 0000 before being placed on the PCI bus by the satellite's local PCI Memory Space 3 Base Register, its contents being all 0's on reset. Note that a satellite in boot from PCI mode uses PCI memory space 3, 0x1FC0 0000 - 0x1FFF FFFF

Notes

The satellite can only boot from a 32-bit port-width external device sitting across the PCI bus. The target device selected for the PCI address 0x0FC0 0000 must have a 32-bit boot memory in this address space (typically a 32-bit EPROM space or an SDRAM space where the bootstrap code for the satellite is placed prior to enabling the satellite). The Target Not Ready bit in the PCI Arbitration register is reset by default. Also, the BusError is disabled at power up in this mode. The BusError must be enabled by the startup code as soon as the satellite is initialized in order to catch any non-decodable address cycles on the PCI bus.

In the PCI-boot mode, the System Controller Internal BIU Buserror Register has the CPU Buserror, IP Buserror, and Watch Dog timeout bits disabled, which allows the RC32334 to wait indefinitely for the PCI host to initialize the system.

Serial EEPROM Interface

When booting from PCI, the serial EEPROM is used to load the PCI configuration header in the Satellite PCI bridge.

The boot serial EEPROM must be compatible with and at least as large as the NM93CS46 (1024-bit or greater), which uses the MICROWIRE™ serial protocol from National Semiconductor. The RC32334 will sequentially read each of the register addresses listed in the table below, starting from EEPROM address 0x00, skipping unused addresses, and continuing up to EEPROM address 0x3E. Each EEPROM address corresponds to a 16-bit datum, such that each EEPROM address holds a 16-bit PCI field. All odd EEPROM addresses are unused by the RC32334 PCI EEPROM interface and can be used for other storage purposes. The 16-bit PCI fields correspond to the definitions of the corresponding PCI Configuration Registers.

Field Name	EEPROM Address
Device ID	0x00
Vendor ID	0x02
Status	0x04
Class Code (MSB's)	0x08
Class Code (LSB), Revision ID	0x0A
Header Type	0x0C
Subsystem ID	0x2C
Subsystem Vendor ID	0x2E
Min_Lat, Min Gnt	0x3C
Interrupt Pin	0x3E

PCI Serial EEPROM Address Fields

Serial Eeprom (Microwire synchronous bus) Driver Interface

The Microwire interface is a 4-wire serial synchronous bus interface with Chipselect(CS), Clock(SK), Data Input (DI) and Data Output (DO) signals. A typical Microwire cycle begins first with selecting the device with Chipselect as High. Once the device is selected, a valid Start Bit ("1") is issued to begin a new cycle. A 2 Bit Opcode of the instruction follows the Start Bit. Then, depending on the instruction under execution, the 6 Bit address and 16 bit data phases begin.

```
#define  PIO_REG0_BASE    0xb8000600
#define  PIO_REG1_BASE    0xb8000610
```

Notes

```
#define  FUNCTION_REG1    (PIO_REG1_BASE + 0x8 )
#define  DIRECTION_REG1   (PIO_REG1_BASE + 0x4 )
#define  DATA_REG1        (PIO_REG1_BASE + 0x0 )

#define  FUNCTION_REG0    (PIO_REG0_BASE + 0x8 )
#define  DIRECTION_REG0   (PIO_REG0_BASE + 0x4 )
#define  DATA_REG0        (PIO_REG0_BASE + 0x0 )

#define  SK_BIT            0x0400  /* Serial Eeprom Clock Bit */
#define  DO_BIT            0x0800   /* Serial Eeprom Data Out  */
#define  DI_BIT            0x0100  /*  Serial Eeprom Data In    */
#define  CS_BIT            0x0001  /*  Serial Eeprom Chip Sel   */

#define  WEN_CMD             4
#define  RD_CMD              6
#define  WR_CMD              5

#define  ERROR              -1
#define  OK                  0

typedef unsigned int        UINT32  ;
typedef unsigned short      UINT16  ;
typedef unsigned char       UINT8   ;

/*
* Function Name :  initSerialDevice
 * Parameters     :  None
 * This function programs the PIO lines of RC32334 to implement the chip select, Serial
 * clock, Data In and Data out for the Serial Eeprom.
 */
void initSerialDevice( )
{
  UINT32*   ptrReg ;
  UINT32    tempRegVal ;

  ptrReg = (UINT32*)FUNCTION_REG1 ;

  tempRegVal = *ptrReg ;
```

Notes

```
    /* Select Bit0 : pci_eeprom_cs as general purpose */
    *ptrReg = (tempRegVal & ~((UINT32)CS_BIT)) ;

    ptrReg = (UINT32*)DIRECTION_REG1 ;

   /* Set the direction of the pci_eeprom_cs as  Output */
    *ptrReg= (*ptrReg | (UINT32)CS_BIT) ;

    ptrReg = (UINT32*)DATA_REG1 ;

    /* Disable the Chip Select for Serial Eeprom to begin with */
    *ptrReg= (*ptrReg & ~((UINT32)CS_BIT ) ) ;

    ptrReg = (UINT32*)FUNCTION_REG0 ;

    tempRegVal = *ptrReg ;

    /* Select the PIO 0 lines for SK, DO, DI as general purpose I/O */
    *ptrReg = ( (tempRegVal & ~(UINT32)(SK_BIT|DO_BIT|DI_BIT)) | 0x0200) ;

    ptrReg = (UINT32*)DIRECTION_REG0 ;

    tempRegVal = *ptrReg ;

    /* Select SK and DO as Output and DI as Input */
    *ptrReg = ( ( tempRegVal | (UINT32)(SK_BIT | DO_BIT) ) & ~(UINT32)(DI_BIT) ) ;

    ptrReg = (UINT32*)DATA_REG0 ;

    tempRegVal = *ptrReg ;
    /* Force 0's on the SK and DO lines  to begin with */
    *ptrReg = ( tempRegVal & ~(UINT32)(SK_BIT | DO_BIT) ) ;
}

/*
* Function Name : memsetSerial
 * Parameters       :
 *                      Pointer to Buffer to copy to Serial Eeprom - serialBufSrc
 *                      Number of  Unsigned Short entries to Copy
```

Notes

```
* Return Value    : Number of elements copied to SerialEprom
 *  The memsetSerial function copies the contents of the Buffer to the Serial Eeprom
*  starting with address ZERO.
*/
int    memsetSerial( UINT16* serialBufSrc, int  len )
{
  UINT16  dataOut ;
   int         length, index ;

  UINT16* pciConfigSpace ;

  pciConfigSpace = (UINT16*) serialBufSrc;
  length              =   len  ;

 /* Initialize the Serial Eeprom Device */
  initSerialDevice() ;

 /* Enable Write to Serial Eprom: WEN
  *  Data         :  0
  *  Address     :  11xx xxxx binary
  * Command  : WEN_CMD
  *  Number of Bits to Write : 0
  */
  writeData( 0, 0x30, (int) WEN_CMD, 0x0);

  for(index=0; index <=length ;index++){
     dataOut  =( (UINT16)(*pciConfigSpace) & 0xffff );

     writeData(dataOut,index, (int) WR_CMD ,0xf);

     /* Wait for Serial Eeprom to finish Write Operation */
     checkBusy();

     pciConfigSpace++;
  }
 return ( (int) length );
}
```

1.4 Low Level Assembly Routines :

Notes

```
#include "idtcpu.h"
#include "iregdef.h"

#define  PIO_REG0_BASE    0xb8000600
#define  PIO_REG1_BASE    0xb8000610

#define  FUNCTION_REG1   (PIO_REG1_BASE + 0x8 )
#define  DIRECTION_REG1  (PIO_REG1_BASE + 0x4 )
#define  DATA_REG1           (PIO_REG1_BASE + 0x0 )

#define  FUNCTION_REG0   (PIO_REG0_BASE + 0x8 )
#define  DIRECTION_REG0  (PIO_REG0_BASE + 0x4 )
#define  DATA_REG0           (PIO_REG0_BASE + 0x0 )

#define  SK_BIT          0x0400
#define  DO_BIT          0x0800
#define  DI_BIT          0x0100

#define  CS_BIT           0x0001

    .globl    skClkGen
    .globl    ckPulseHigh
    .globl    ckPulseLow
    .globl    writeData
    .globl    readData
    .globl   checkBusy

/*
 * Function Name : skClkGen
 *             Serial Clock Generate
 * Parameters    :
 *             a0 = data on the DO_BIT
 * Return Value  : None
 */
    .ent  skClkGen
skClkGen:
    .set  noreorder
     li   t0, DATA_REG0
```

Notes

```
        li   t1, ~SK_BIT

        lw   t2, 0x0(t0)
        and  t2, t1
        or    t2, a0     /* Maintain the DO_BIT */

        /* Force SK low to begin with */
        sw   t2, 0x0(t0)

        /* some dummy cycles */
        nop
        nop
        nop

        /* Force SK High */
        ori  t2, SK_BIT
        sw   t2,0x0(t0)

        /* some dummy cycles */
        nop
        nop
        nop

        /* Force SK Low again */
        and  t2, t1
        sw   t2, 0x0(t0)

        j    ra
        nop
        .set reorder
        .end skClkGen

/*
 *    Function Name :  ckPulseHigh
*    Generates a level High on Chip Select Line
*/
        .ent ckPulseHigh
ckPulseHigh:
        .set noreorder
```

Notes

```
        li   t0,DATA_REG1
        li   t1,~CS_BIT

        lw   t2, 0x0(t0)
        and  t2, t1

        /* Force CS High */
        ori  t2,CS_BIT
        sw   t2,0x0(t0)

        j    ra
        nop
        .set reorder
        .end ckPulseHigh

/*
* Function Name : ckPulseLow
* Generate a level Low on Chip Select  Line
  */
     .ent ckPulseLow
ckPulseLow:
        .set noreorder
        li   t0,DATA_REG1
        li   t1,~CS_BIT

        lw   t2, 0x0(t0)
        and  t2, t1

        /* Force CS Low */
        sw   t2,0x0(t0)

        j    ra
        nop
        .set reorder
        .end ckPulseLow


/* Function Name   :write data
   Parameters:
```

Notes

```
   a0 = Data
   a1 = Index
   a2 = Cmd
   a3 = Number of bits to Write
   Return Value: None
*/
    .ent writeData
writeData:
    .set noreorder
    li  t3,DATA_REG0
    move t9,a0      /* save a0 as its being used in
                       call to skClkGen */

    move v1, ra

    jal  ckPulseHigh
    nop

    move ra, v1

    /* Write the command */
    li   t5, 0x2
    li   t6, 0xb
1:
    move t4, a2
    srl  t4,t5
    andi t4,0x1
    sll  t4,t6
    andi t4,DO_BIT
    lw   t7,0x0(t3)
    and  t7,~DO_BIT
    or   t7,t4
    sw   t7,0x0(t3)
    move a0,t4
    move v1,ra

    jal  skClkGen
    nop
```

Notes

```
        move ra,v1

        subu t5,0x1

        bgez t5, 1b
        nop

       /* Write the Address == Index */
        li   t5, 0x5
1:
        move t4, a1
        srl  t4,t5
        andi t4,0x1
        sll  t4,t6
        andi t4,DO_BIT
        lw   t7,0x0(t3)
        and  t7,~DO_BIT
        or   t7,t4
        sw   t7,0x0(t3)

        move a0,t4
        move v1,ra

        jal  skClkGen
        nop

        move ra,v1

        subu t5,0x1

        bgez t5, 1b
        nop

       /* Write the data if any */
        move t4, a3
        beq  t4, r0, 2f
        nop

        li   t5,0xf
```

Notes

```
1:
    move t4,t9
    srl     t4,t5
    andi   t4,0x1
    sll     t4,t6
    andi   t4,DO_BIT
    lw      t7,0x0(t3)
    and    t7,~DO_BIT
    or      t7,t4
    sw     t7,0x0(t3)

    move a0,t4
    move v1,ra

    jal  skClkGen
    nop

    move ra,v1

    subu t5,0x1

    bgez t5, 1b
    nop

2:
   /* Generate CS Pulse */

    move v1, ra

    jal  ckPulseLow
    nop

    move ra, v1

    j    ra
    nop
    .set reorder
    .end writeData
```

Notes

```
/*
 *  Function Name :  readData
 *  Parameters    :
 *                  a0 = index
 *  Return Value  :
 *                  v0 = data read (unsigned short)
 */


    .ent readData
readData:
    .set reorder
      li  t3,DATA_REG0
      move t9,a0       /* save index as a0 is being
                          used in call to skClkGen */

      move v1, ra

      jal  ckPulseHigh
      nop

      move ra, v1

      /* Write the command */
      li   t5, 0x2
      li   t6, 0xb
      li   t8, 0x6   /* Read Command */
1:
      move t4,t8
      srl  t4,t5
      andi t4,0x1
      sll  t4,t6
      andi t4,DO_BIT
      lw   t7,0x0(t3)
      and  t7,~DO_BIT
      or   t7,t4
      sw   t7,0x0(t3)
      move a0,t4
      move v1,ra
```

Notes

```
        jal  skClkGen
        nop

        move ra,v1

        subu t5,0x1

        bgez t5, 1b
        nop

       /* Write the Address == Index */
        li   t5, 0x5
1:
        move t4, t9
        srl  t4,t5
        andi t4,0x1
        sll  t4,t6
        andi t4,DO_BIT
        lw   t7,0x0(t3)
        and  t7,~DO_BIT
        or   t7,t4
        sw   t7,0x0(t3)

        move a0,t4
        move v1,ra

        jal  skClkGen
        nop

        move ra,v1

        subu t5,0x1

        bgez t5, 1b
        nop

       /* Read the dummy data*/

1:
```

Notes

```
        lw   t7,0x0(t3)
        andi t7,DI_BIT

        bne  t7,zero,1b
        nop

        /* Read the data */
        li   t5,0xf
        li   t6,0x0
        li   t4,0x8


1:
        move v1, ra

        jal  skClkGen
        nop

        move ra, v1

        lw   t7,0x0(t3)

        andi t7,DI_BIT

        srl  t7,t4

        andi t7,0x1

        sll  t7,t5

        or   t6,t7

        subu t5, 1

        bgez t5, 1b
        nop

        /* Copy data read to return register*/
        move v0, t6
```

Notes

```
    /* Generate CS Pulse */
    move v1, ra

    jal  ckPulseLow
    nop

    move ra, v1


    j    ra
    nop

    .set reorder
    .end readData

    .ent checkBusy
checkBusy:
    .set noreorder
    li  t3, DATA_REG0

    move v1, ra
    jal  ckPulseHigh
    nop

    move ra, v1

1:
    lw  t4,0x0(t3)

    andi t4,0x100

    bne  t4,zero,1b
    nop

1:
    lw  t4,0x0(t3)

    andi t4,0x100
```

Notes

```
beq  t4,zero,1b
nop

move v1, ra

jal  ckPulseLow
nop

move ra, v1

j    ra
nop
.set reorder
.end checkBusy
```

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.