SH7786 Group

SH7786 PCI Express Controller (PCIEC) Initialization Sample Program

R01AN0557EJ0100
Rev.1.00
Jul 15, 2011

Introduction

This application note presents a sample program for making the initial settings required by the PCI Express controller of the SH7786.

Target Device

SH7786

Contents

1. Introduction

1.1 Specifications

The PCI Express controller (PCIEC) initialization sample program presented in this application note makes initial settings to the local bus state controller (LBSC), DDR3-SDRAM interface (DBSC3), and PCI Express controller (PCIEC) after the power-on reset is cleared. After initialization, the PCI Express controller (PCIEC) operates either as a PCI Express root port or endpoint. When operating as a PCI Express root port, it displays on the serial console information such as the vendor ID and device ID of PCI Express endpoint devices and executes simple DMA transfers. When operating as a PCI Express endpoint, it specifies setting items such as vendor ID and device ID in the PCI Express controller (PCIEC).

1.2 Functions Used

- Local bus state controller (LBSC)
- DDR3-SDRAM interface (DBSC3)
- PCI Express controller (PCIEC)
- Serial communication interface (SCIF0)

Initial settings for the local bus state controller (LBSC) and DDR3-SDRAM interface (DBSC3) are described in *SH7786 Group Application Note: SH7786 Initial Settings Sample Program* (R01AN0519EJ0101). Refer to that document in conjunction with this application note.

Note that descriptions of the initial settings to the local bus state controller (LBSC) and DDR3-SDRAM interface (DBSC3) are omitted from this application note as the relevant operations are verified in *SH7786 Group Application Note: SH7786 Initial Settings Sample Program* (R01AN0519EJ0101).

1.3 Applicable Conditions

Table 1.1 Applicable Conditions

Evaluation board	AP-AH4AD-0A (Alpha Project)*1	
	CPU	SH7786
	Operating frequencies	Internal clock: 533 MHz SuperHyway clock: 267 MHz Peripheral clock: 44 MHz DDR3 clock: 533 MHz External bus clock: 89 MHz
	Clock operating mode	Clock mode 3 (MD0 = high, MD1 = high, MD2 = low, MD3 = low)
	Endian mode	Little endian (MD8 = high)
	Addressing mode	29-bit addressing mode (MD10 = low)
	Area 0 bus width	16 bits (MD4 = low, MD5 = High, MD6 = low)
	Memory	NOR flash memory, 16 MB (area 0): Spansion S29GL128P90TFIRI DDR3-SDRAM, 256 MB (areas 2 to 5): Micron MT41J64M16LA-187E (2 chips)
	PCI Express	SH7786 on-chip PCI Express controller (PCIEC) Support for PCI Express Base Specification, revision 1.1 PCI Express Generation 1: Bus frequency: 2.5 GHz Root port: PCI Express ×4 card slot, 1 channel Endpoint: PCI Express ×1 card edge, 1 channel
	Serial interface	SH7786 on-chip SCIF channel 0 (115,200 bps)
	PC-USB-02A (Alpha Project)*2	
	Serial console	TTL serial ↔ USB converter
Toolchain	Super-H RISC engine Standard Toolchain Version 9.3.2.0	
	Compiler options*3	-cpu=sh4a -endian=little -include="$(PROJDIR)\inc","$(PROJDIR)\inc\drv" -define=CONFIG_PCIE_ROOT=0 -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo
	Assembler options	-cpu=sh4a -endian=little -round=zero -denormalize=off -include="$(PROJDIR)\inc" -debug -object="$(CONFIGDIR)\$(FILELEAF).obj" -literal=pool,branch,jump,return -nolist -nologo -chgincpath -errorpath
	Linker options	-noprelink -rom=D=R -nomessage -list= "$(CONFIGDIR)\$(PROJECTNAME).map" -optimize=safe -start=INTHandler,VECTTBL,INTTBL,IntPRG/0800, PResetPRG/01000,P,C,C$BSEC,C$DSEC,D/02000, RSTHandler,PnonCACHE/0A0000000,B,R/0ADF00000, S/0ADFF0000 -nologo

Notes: 1. For detailed information on using the AP-SH4AD-0A, refer to *AP-SH4AD-0A Hardware Manual*.
2. For detailed information on using the PC-USB-02A, refer to *AP-SH4AD-0A Hardware Manual*.
3. To operate the PCI Express controller (PCIEC) as a PCI Express root port, specify CONFIG_PCIE_ROOT=0 in the macro definitions. To operate the PCI Express controller (PCIEC) as a PCI Express endpoint, specify CONFIG_PCIE_END=1 in the macro definitions.

Table 1.2 lists the section allocations used in the sample program.

Table 1.2 Section Allocations

Section	Section Usage	Area	Allocation Address (Virtual Address)	
INTHandler	Exception/interrupt handler	ROM	0x00000800	P0 area (Can be cached, MMU address conversion not possible)
VECTTBL	Reset vector table Interrupt vector table	ROM		
INTTBL	Interrupt mask table	ROM		
IntPRG	Interrupt function	ROM		
PResetPRG	Reset program	ROM	0x00001000	
P	Program area	ROM	0x00002000	
C	Constant area	ROM		
C$BSEC	Uninitialized data area address structure	ROM		
C$DSEC	Initialized data area address structure	ROM		
D	Initialized data	ROM		
RSTHandler	Reset handler	ROM	0xA0000000	P2 area (Can not be cached, MMU address conversion not possible)
PnonCACHE	Program area (Cache invalid access)	ROM		
B	Uninitialized data area	RAM	0xADF00000	
R	Initialized data area	RAM		
S	Stack area	RAM	0xADFF0000	

1.4 Descriptions of Terms Used in the Application Note

- PCI Express
  PCI Express is a serial transfer interface standard, established by PCI-SIG, that is intended as a replacement for the PCI bus standard. It is not compatible with the physical layer of the PCI bus, which has a 32-bit parallel interface, but it allows continued use of existing software resources because it uses common communication protocols.
  Each of the lanes used by PCI Express to transfer data consists of a differential pair (transmit and receive) to enable bidirectional communication (dual simplex). Revision 1.1 of the PCI Express Base Specification (commonly referred to as Gen1) supports a maximum unidirectional transfer rate of 2.5 Gbps per lane and a maximum bidirectional rate of 5.0 Gbps per lane. The data bandwidth can be increased by combining multiple lanes, using two lanes to double the transfer rate, four lanes to quadruple it, and so on.

- PCI Express root port
  A PCI Express root port performs overall control of the PCI Express system. Each PCI Express system must have at least one root port.
  When a configuration cycle commences, the PCI Express root port controls the PCI Express system overall, initializing the system, receiving error messages, recovering from errors, etc. The root port also transmits request packets, returns completion packets, and transmits and receives messages.

- PCI Express endpoint
  PCI Express endpoints are ports that perform data communication under the control of a root port. A PCI Express system can have multiple endpoints.
  After an endpoint is initialized in the configuration cycle, it performs error detection, sends notifications to the root port, etc. Endpoints also transmit request packets, return completion packets, and transmit and receive messages.

- I/O address space
  The I/O address space is PCI bus-compatible.

- Memory address space
  The memory address space is PCI bus-compatible.

- Configuration register space
  The configuration register space is PCI bus-compatible. The total configuration register space comprises 4,096 bytes, of which the lower 256 bytes compose an area that is compatible with the earlier PCI bus standard. The upper 3,840 bytes compose an area used by PCI Express that is called the PCI Express extended configuration space.

1.5 Scope of the Sample Program

The sample program introduced in this application note does not support all the functions of the PCI Express controller (PCIEC). The application note describes basic usage scenarios in which, after initialization, the PCI Express controller (PCIEC) operates either as a PCI Express root port or endpoint. When operating as a PCI Express root port, it displays on the serial console information such as the vendor ID and device ID of PCI Express endpoint devices and executes simple DMA transfers. When operating as a PCI Express endpoint, it specifies setting items such as vendor ID and device ID in the PCI Express controller (PCIEC).

The descriptions in this application note do not cover the following functions of the PCI Express controller (PCIEC).

- Message transmission and reception
- INTx/MSI interrupts
- Link power control function (L0, L0s, L1, and L3 states)

2. PCI Express Controller (PCIEC)

The PCI Express controller (PCIEC) performs PCI Express control and transfers data between the internal bus (SuperHyway bus) of the SH7786 and PCI devices connected to the PCI Express interface.

This section describes the PCIEC functions supported by the sample program. For a detailed description of the PCIEC, see section 13, PCI Express Controller (PCIEC), in *SH7786 Group User's Manual: Hardware* (REJ09B0501).

2.1 Supported Functions

(1) Packet Transmission/Reception

Table 2.1 lists the supported PCI Express packets. The PCIEC supports packets that are not prohibited by the standard.

Table 2.1 Supported PCI Express Packets

	Root Port		Endpoint	
Packet Type	**Transmission**	**Reception**	**Transmission**	**Reception**
Memory read	○	○	○	○
Memory write	○	○	○	○
I/O read	○	○	—	○
I/O write	○	○	—	○
Lock	○*	—	—	—
Configuration read	○	○	—	○
Configuration write	○	○	—	○
Message	○*	○*	○*	○*

Legend:
○: Supported by the PCIEC.
—: Use by PCI Express prohibited by the standard.
*: Supported by the PCIEC but not supported by the application program.

(2) Message Transmission/Reception

Table 2.2 lists the supported PCI Express messages. The PCIEC does not support vendor-defined messages. Note that the sample program does not support message transmission or reception.

Table 2.2 Supported PCI Express Messages

	Root Port		Endpoint	
Packet Type	**Transmission**	**Reception**	**Transmission**	**Reception**
Assert_INTA	—	○	○	—
Assert_INTB	—	○	○	—
Assert_INTC	—	○	○	—
Assert_INTD	—	○	○	—
Deassert_INTA	—	○	○	—
Deassert_INTB	—	○	○	—
Deassert_INTC	—	○	○	—
Deassert_INTD	—	○	○	—
PME_Active_State_Nak	Δ	—	—	Δ
PM_PME	—	Δ	Δ	—
PME_Turn_Off	Δ	—	—	Δ
PME_To_Ack	—	Δ	Δ	—
ERR_COR	—	○	○	—
ERR_NONFATAL	—	○	○	—
ERR_FATAL	—	○	○	—
Unlock	○	—	—	○
Set_Slot_Power_Limit	○	—	—	○
Vender_Define Type0	×	×	×	×
Vender_Define Type1	×	×	×	×

Legend:
○: Supported by the PCIEC.
Δ: Transmission/reception possible, but control by software is required.
—: Use by PCI Express prohibited by the standard.
×: Not supported by the PCIEC.

(3) Configuration Registers

Table 2.3 lists the supported PCI Express configuration registers. The PCIEC does not support registers related to the built-in self-test (BIST) function, switches, and expansion ROM.

Table 2.3 Supported PCI Express Configuration Registers

Configuration Register	PCIEC Register	Root Port	Endpoint
Vendor ID register	PCICONF0[15:0]	○	○
Device ID register	PCICONF0[31:16]	○	○
Command register	PCICONF1[15:0]	○	○
Status register	PCICONF1[31:16]	○	○
Revision ID register	PCICONF2[7:0]	○	○
Class code register	PCICONF2[31:8]	○	○
Cache line size	PCICONF3[7:0]	—	—
Master latency timer	PCICONF3[15:8]	—	—
Header type register	PCICONF3[23:16]	○	○
BIST register	PCICONF3[31:24]	×	×
Base address register 0	PCICONF4[31:0]	○	○
Base address register 1	PCICONF5[31:0]	○	○
Base address register 2	PCICONF6[31:0]	—	○
Primary bus number	PCICONF6[7:0]	○	—
Secondary bus number	PCICONF6[15:8]	○	—
Subordinate bus number	PCICONF6[23:16]	○	—
Secondary latency timer	PCICONF6[31:24]	—	—
Base address register 3	PCICONF7[31:0]	—	○
I/O base register	PCICONF7[7:0]	×	—
I/O limit register	PCICONF7[15:8]	×	—
Secondary status register	PCICONF7[31:16]	○	—
Base address register 4	PCICONF8[31:0]	—	○
Memory base	PCICONF8[15:0]	×	—
Memory limit	PCICONF8[31:16]	×	—
Base address register 5	PCICONF9[31:0]	—	○
Prefetchable memory base	PCICONF9[15:0]	×	—
Prefetchable memory limit	PCICONF9[31:16]	×	—
Card bus CIS pointer	PCICONF10[31:0]	—	×
Prefetchable base (upper 32 bits)	PCICONF10[31:0]	×	—
Subsystem ID register	PCICONF11[31:0]	—	○
Prefetchable limit (upper 32 bits)	PCICONF11[31:0]	×	—
Subsystem vendor ID register	PCICONF12[31:0]	—	○
I/O base register (upper 16 bits)	PCICONF12[15:0]	×	—
I/O limit register (lower 16 bits)	PCICONF12[31:16]	×	—
Capability pointer	PCICONF13[31:0]	○	○
Expansion ROM base address register	PCICONF14[31:16]	—	×
Interrupt line	PCICONF15[7:0]	○	○
Interrupt pin	PCICONF15[15:8]	○	○
Minimum grant	PCICONF15[23:16]	—	—
Maximum latency	PCICONF15[31:24]	—	—
Bridge control register	PCICONF15[31:16]	○	—

Legend:
○: Supported by the PCIEC.
—: Use by PCI Express prohibited by the standard.
×: Not supported by the PCIEC.

(4) Capability Structures

Table 2.4 lists the supported PCI Express capability structures. The PCIEC supports the capability structures listed. Note that the sample program does not support PCI Express capability structures.

Table 2.4 Supported PCI Express Capability Structures

Capability Structure	Supported/ Not Supported	Start Address
PCI power management	○	H'040
MSI	○	H'050
PCI Express	○	H'070
Advanced error reporting	×	—
Virtual channel	○	H'100
Device serial number	○*	H'1B0
PCI Express link complex declaration	×	—
PCI Express root complex internal link control	×	—
Power budgeting	×	—
PCI Express root complex event collector endpoint association	×	—
Multi-function virtual channel	×	—
Vendor-specific	×	—
RCRB header	×	—

Legend:
○: Supported by hardware.
×: Not supported by the PCIEC.
*: The PCIEC implements the device serial number capability structure, but no serial number is specified by the hardware. To use the device serial number capability structure, set the serial number by software. The device serial number capability structure is not included in the capability list chain in the initial state. To use the structure, add it to the capability list chain.

2.2 Pin Assignments

The PCIEC operates as either a root port or an endpoint as defined by the PCI Express standard. The operating mode is specified by the mode pins. Mode pin settings for the sample program are specified by using the DIP switches on the evaluation board. For details of the DIP switches, see 4.1, AP-SH4AD-0A SH7786 Evaluation Board.

The PCIEC does not support the legacy endpoint, root complex integrated endpoint, switch, and root complex invent controller operation modes defined in the PCI Express standard.

(1) Root Port

A root port performs overall control of the PCI Express system. Each PCI Express system must have at least one root port. The PCIEC can operate as a root port for which the SH processor acts as a host processor.

When a configuration cycle commences, the root port controls the PCI Express system overall, initializing the system, receiving error messages, recovering from errors, etc. The root port also transmits request packets, returns completion packets, and transmits and receives messages.

(2) Endpoint

An endpoint is a port that performs data communication under the control of a root port. A PCI Express system can have multiple endpoints. The PCIEC can operate as an endpoint.

After an endpoint is initialized in the configuration cycle, it performs error detection, sends notifications to the root port, etc. Endpoints also transmit request packets, return completion packets, and transmit and receive messages.

2.3 PCIEC Module Initialization

To enable PCI Express packet communication by using the PCIEC, it is necessary to: (1) make settings for the bridge function to link the PCI Express bus and the internal bus of the SH7786 (SuperHyway bus) and (2) establish a connection between the PCI Express bus and the PCIEC.

(1) Bridge Function Settings to Link PCI Express Bus and SuperHyway Bus

Making settings for the bridge function to link the PCI Express bus and the SuperHyway bus involves setting transfer information in the registers listed below. For details on the transfer information to be set, see 2.6, Target Transfers.

- PCIELAR0 to PCIELAR5
- PCIELAMR0 to PCIELAMR5

(2) Establishing Connection between PCI Express Bus and PCIEC

After specifying the transfer information in the above transfer control registers, set the CFINIT bit (bit 0) in PCIETCTLR to 1 to indicate the start of connection establishment. (The values of the above transfer control registers cannot be changed after CFINIT is set to 1).

Setting the CFINIT bit (bit 0) in PCIETCTLR to 1 starts the initialization of the data link layer to prepare for communication with the connection-target PCI Express device.

When initialization of the data link layer completes, the DL_Active state is entered, making the system ready for communication by VC0. Initialization is completed when the DL_Active state is confirmed by any of the following methods.

Establishment of communication by VC0

- DLLACT (bit 0) in PCIETSTR is set to 1.
- VC NeGotiation PenDing (bit 17) in VCCAP6 is set to 1.
- A INTDL interrupt indicating DL_Active is generated.

The following settings must be performed in advance in order to generate an INTDL interrupt by DL_Active.

- Set INTDLE (bit 14) in PCIEINTER to 1.
- Set Data Link Layer ACTive Enable (bit 31) in DLINTENR to 1.

The PCIEC does not support more than one virtual channel (VC). Only one virtual channel (VC0) can be used for communication.

2.4 Configuration Cycle (PCI Express Initialization)

When the PCIEC is used as a root port, a configuration cycle must be initiated to configure the connection-target device. In the configuration cycle, configuration access is performed to confirm the status of the configuration registers of the connection-target endpoints and, based on the result, values are assigned to the configuration registers of the root port itself and of the endpoints. The root port typically accesses its own configuration registers via the SuperHyway bus.

(1) Initiating Configuration Access

When the PCIEC is used as a root port, the configuration is accessed to start the configuration cycle, in which a variety of initial settings are made.

The procedure described below is used to access the configuration registers of external devices by means of configuration access by PCIEC.

The procedure described below should not be used by the PCIEC, when operating as a root port, to access its own configuration registers. Instead, perform access via the SuperHyway bus to registers mapped to the SuperHyway bus address space.

(a) PCIEPAR Settings

Specify the access destination configuration register number, extension register number, and access destination device bus, device, and function numbers in PCIEPAR.

(b) PCIEPCTLR Settings

Specify the type of configuration access to be initiated and set the access enable bit in

PCIEPCTLR.

(c) PCIEPDR Settings

Generate a configuration read by read-accessing PCIEPDR, and generate a configuration write by write-accessing PCIEPDR. Reading PCIEPDR returns the result of the configuration read.

(d) Checking PCIEPCTLR

Check the CRS bit (bit 16) in PCIEPCTLR to verify whether the configuration request retry status (CRS) has been returned.

A CRS value of 1 indicates that a correct response to the configuration request has not been made because the connection-target device has not been activated. If CRS is set to 1, write 1 to CRS to clear it and resume processing from step (c), above.

It is not necessary to recheck the CRS bit again once configuration access to a device is successful.

(2) Receiving Configuration Access

When the PCIEC is used as an endpoint, it receives configuration accesses from root ports and accepts initialization processing.

Reception of a configuration access by the PCIEC is handled automatically in hardware, and no software control is necessary.

Software processing is required, however, when a configuration write access to the PowerState field (bits 1 and 0) in PMCAP1 is used to change the power state.

When a normal configuration write is received, the PCIEC reads the bus number and device number in the received packet, and writes them to the BusNumber (bits 31 to 24), DeviceNumber (bits 23 to 19), and FunctionNumber (bits 18 to 16) fields in TLCTLR. These values are used as requester IDs for the packets generated by the PCIEC.

(3) Setting Details

To use the PCIEC as a root port, issue a configuration request and specify settings listed below to initialize the PCI Express. These settings are made by the root port to both the root port and endpoint registers.

The description below applies to the use of a single PCI Express device as the connection target. Additional settings are necessary when the connection target is a switch or a bridge.

(a) Max Payload Size (MPS) Setting

Examine the Max Payload Size Supported (MPSS) values in the configuration registers of all the PCI Express devices in the PCI Express system, including both root ports and endpoints, and use the smallest value as the system MPS. Set the MPS value in the configuration registers of all devices, both root ports and endpoints.

(b) Max Read Request Size (MRRS) Setting

For the PCIEC, the MRRS value should be the same as the MPS value. Set a value equal to MPS in the configuration registers of all the PCI Express devices, both root ports and endpoints.

(c) PCI Address Space Setting (BAR Setting)

Allocate PCI address space for each device.

Allocate address space according to the PCI Express standard, then set the BAR for each device to match.

(d) Operating Mode Setting

Set values in the configuration registers listed below to define the PCI Express operating mode. No setting needs to be made if the initial value is used. Do not change the values of these registers after the configuration cycle completes. For a detailed description of each register, see 13.4.5, Configuration Registers, in *SH7786 Group User's Manual: Hardware*.

PCICONF1[10].Interrupt Disable

PCICONF1[8].SERR Enable

PCICONF1[6].Parity Error Response

PCICONF15[17].SERR Enable (root port only)

PCICONF15[15:8].Interrupt Pin (endpoint only)

PCICONF15[7:0].Interrupt Line (root port only)

EXPCAP2[11].Enable No Snoop

EXPCAP2[4].Enable Relaxed Ordering

EXPCAP2[3].Unsupported Request Reporting Enable

EXPCAP2[2].Fatal Error Reporting Enable

EXPCAP2[2].Non Fatal Error Reporting Enable

EXPCAP2[2].Correctable Error Reporting Enable

EXPCAP3[20].Data Link Layer Active Reporting Capable (root port only)

EXPCAP7[4].CRS Software Visibility Enable

EXPCAP7[3].PME Interrupt Enable

EXPCAP7[2].System Error on Fatal Error Enable

EXPCAP7[1].System Error on Non-Fatal Error Enable

EXPCAP7[0].System Error on Correctable Error Enable

(e) INTx/MSI Interrupt Setting

Determine the type of interrupt to be used by the system, INTx or MSI, and make the corresponding setting for each device.

(f) Master Enable Setting

Set the Bus Master Enable bit (bit 2), Memory Space Enable bit (bit 1), and I/O Space Enable bit (bit 0) in PCICONF1 to match the transfer to be performed following initialization.

When the root port receives a request from an endpoint, first set the Bus Master Enable bit of the root port to 1. At the same time, set Memory Space Enable to 1 if memory access is to be accepted, or set I/O Space Enable to 1 if I/O access is to be accepted. Without these settings, the root port will not accept requests. Next, set the Bus Master Enable bit of the endpoint to 1. Without this setting, the endpoint cannot issue requests.

To enable memory access or I/O access to the endpoint, set the Memory Space Enable or I/O Space Enable bit of the endpoint to 1. Without this setting, the endpoint will not receive requests.

2.5 PI/O Transfers (Data Transfer from PCIEC to External Device)

As used here, PI/O transfer refers to a data transfer performed by accessing the PCIEC memory space via the internal bus in order to generate PCI Express packets.

(1) Overview

In a PI/O transfer, a unit such as the CPU accesses the PCIEC memory space via the SuperHyway bus in order to generate PCI Express packets that are then transmitted. PI/O transfers can be used to perform memory read/write and I/O read/write access to external PCI Express devices.

PI/O transfers make it possible easily to generate PCI Express packets by accessing the PCI memory space. PCI Express read packets are generated by read access, and PCI Express write packets are generated by write access.

In a typical PI/O transfer, one PCI Express packet is generated per access to the PCI memory space. The data length of the generated PCI Express packet is equal to the access size to the PCI memory space. For this reason, only short PCI Express packets with a data length of 4 bytes can be generated by 4-byte accesses by the CPU, so the data transfer efficiency is poor when large amounts of data are transferred.

To transfer large quantities of data, the packet connection function or the DMAC incorporated into the PCIEC should be used.

Note that the sample program does not support the packet connection function.

(2) Address Map (SuperHyway Space)

Table 2.5 shows the address map of the SuperHyway space.

The PCIEC has three address areas (consisting of eight physical types): the PCI memory area (six types), the control register area, and the configuration register area. PCI Express packets are generated by accessing the PCI memory area. The mapping between the PCI memory area and the PCI Express address space is described below.

Table 2.5 SuperHyway Space Address Map

Memory Area	PCIEC0	PCIEC1	PCIEC2	Physical Address Size
PCI are 0	H'FD00_0000 to H'FD7F_FFFF	H'FD80_0000 to H'FDFF_FFFF	H'FC80_0000 to H'FCBF_FFFF	PCIEC0/1: 8 MB PCIEC2: 4 MB
PCI are 1	Not available	Not available	Not available	512 MB
PCI are 2	H'1000_0000 to H'13FF_FFFF (only for memory space setting 1, 2, 5, or 6)	Not available	Not available	64 MB
PCI are 3	H'FE10_0000 to H'FE1F_FFFF	H'FE30_0000 to H'FE3F_FFFF	H'FCD0_0000 to H'FCDF_FFFF	1 MB
Control register area (1)	H'FE00_0000 to H'FE03_FFFF	H'FE20_0000 to H'FE23_FFFF	H'FCC0_0000 to H'FCC3_FFFF	256 kB
Configuration register	H'FE04_0000 to H'FE04_0FFF	H'FE24_0000 to H'FE24_0FFF	H'FCC4_0000 to H'FCC4_0FFF	4 kB
Control register area (2)	H'FE04_1000 to H'FE07_FFFF	H'FE24_1000 to H'FE27_FFFF	H'FCC4_1000 to H'FCC7_FFFF	252 kB

Notes: 1. The above address map is for the 29-bit address mode.
2. The sample program uses areas PCIEC0 and PCIEC1.

(3) Accessing PCI Memory Space and PCI I/O Space

Figure 2.1 illustrates the mapping of the SuperHyway address space to the PCI address spaces. As shown in figure 2.1, accesses to the PCI area in the SuperHyway address space are mapped to either a PCI address space or a PCI I/O space. The PI/O transfer control registers (described below) specify the space to which the mapping is to be performed or an address in a specific space where the mapping is to be performed. Access to a PCI memory space or a PCI I/O space can be performed by accessing the space (PCI area) on the SuperHyway that is mapped to the PCI space.

A read access to a PCI area generates a read packet for a PCI memory space or PCI I/O space, and a write access to a PCI area generates a write packet for a PCI memory space or PCI I/O space.

When a PCI memory space is accessed, the packet length is determined according to the access size to the PCI area. In other words, if the PCI area is accessed by 4-byte access, read/write packets of 4 bytes (1 DW) are generated in the PCI memory space.

Only 4-byte (1 DW) access to a PCI I/O space is allowed. When a PCI area is mapped to a PCI I/O space, access to the PCI area should be made with an access size of 4 bytes.

The PIO transfer control registers specify the transfer destination space (selection of PCI memory or I/O space), the start address in each space, the size of transfer destination space, and the attributes of the transfer packets.

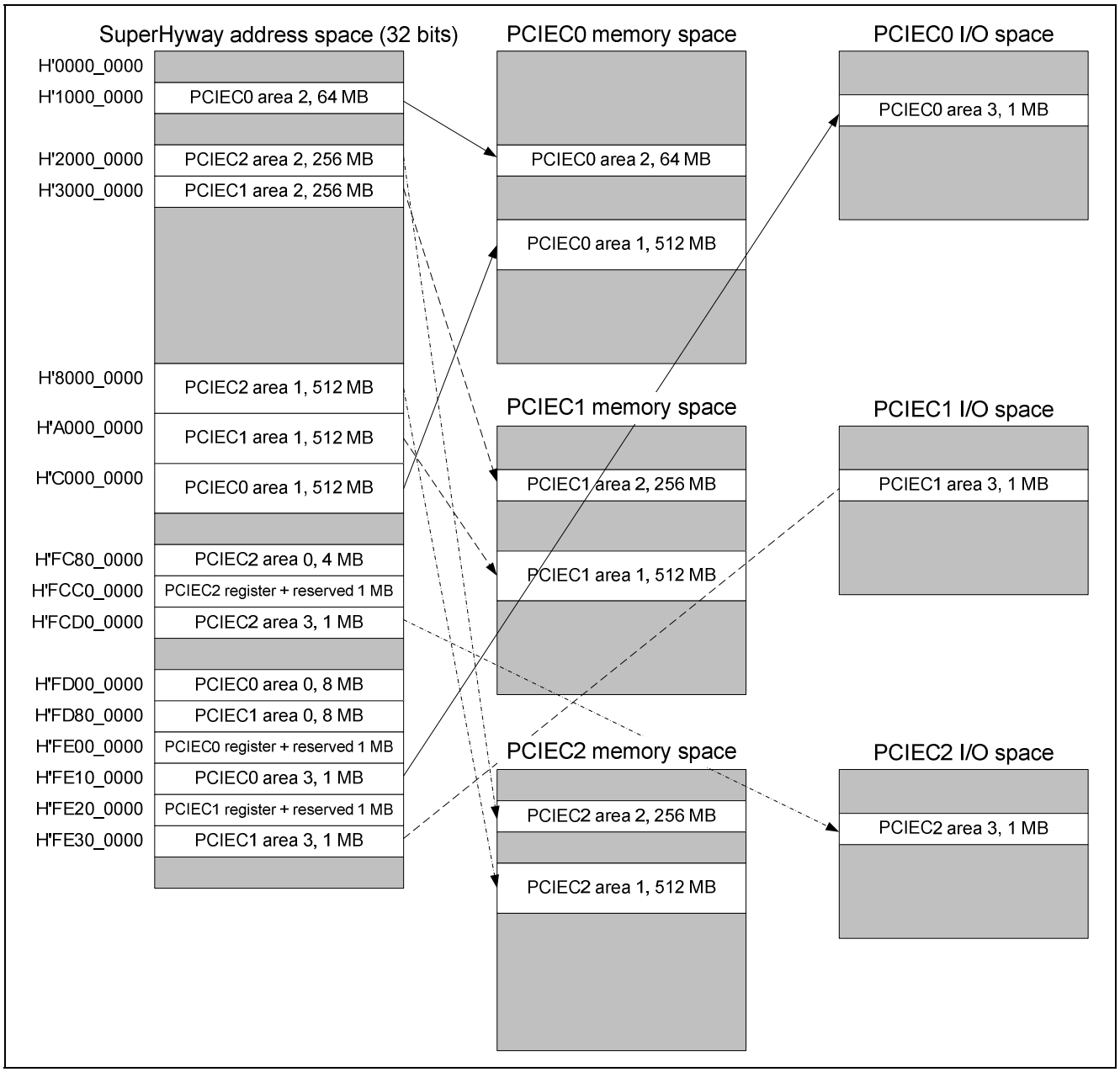

Figure 2.1 Mapping of SuperHyway Address Space to PCI Address Spaces

(4) Register Settings for PI/O Transfers

Table 2.6 lists the transfer control registers used for PI/O transfers. Accesses to PCI areas 0 to 3 are mapped to the PCI memory or I/O spaces according to these register settings. The functions of these registers are listed in table 2.6.

Table 2.6 Transfer Control Registers for PI/O Transfers

PCIEPALR0 to PCIEPALR3	Start addresses of the PCI address spaces to which PCI areas 0 to 3 are mapped (lower 32 bits)
PCIEPAHR0 to PCIEPAHR3	Start addresses of the PCI address spaces to which PCI areas 0 to 3 are mapped (upper 32 bits)
PCIEPAMR0 to PCIEPAMR3	Specifies the sizes of data in PCI areas 0 to 3 mapped to the PCI address spaces.
PCIEPTCTLR0 to PCIEPTCTLR3	Enables/disables PCI areas 0 to 3. Specifies the transfer destination space (PCI memory space or PCI I/O space). Specifies attributes (Lock, EP, No Snoop, Relax Ordering) for conversion.

PCIEPALRn and PCIEPAHRn (n = 0 to 3) specify an address in the PCI Express space to which PCI area n is mapped.

PCIEPAMRn specifies the size of the PCI area. It is not possible to specify a size larger than the size of the PCI area as listed in table 2.5, SuperHyway Space Address Map.

PCIEPTCTLRn specifies whether a given area is enabled or disabled, the transfer destination space, and the attributes of packets during the transfer process. Unless it is specified in PCIEPTCTLRn that PCI area n is enabled (default: disabled), any access to the corresponding PCI area is invalid. To perform a lock transfer or specify other attributes, settings must be made to PCIEPTCTLRn before accessing the PCI area.

(5) Address Conversion from SuperHyway Bus to PCI

The address used when accessing a PCI space by means of accessing a PCI area is determined by the address of the PCI area accessed and by the settings of the associated transfer control register. The address conversion details are described below and illustrated in figure 2.2, Address Conversion to PCI Space. (In the figure and description below, the number n represents a value of 0 to 3, which corresponds to a PCI areas from 0 to 3).

The lower 16 bits of the PCI address (bits 17 to 2) are generated from the lower bits of the SuperHyway address.

The middle 11 bits of the PCI address (bits 28 to 18) are selected from the corresponding bits of the SuperHyway address or PCIEPALRn, depending on the value of the transfer control register (PCIEPAMRn). (The SuperHyway address is used if the value of the corresponding bit in PCIEPAMRn is 1, and PCIEPALRn is used if it is 0.)

The contents of PCIEPAHRn and the upper 3 bits of PCIEPALRn are used as the upper 35 bits (bits 63 to 29) of the PCI address.

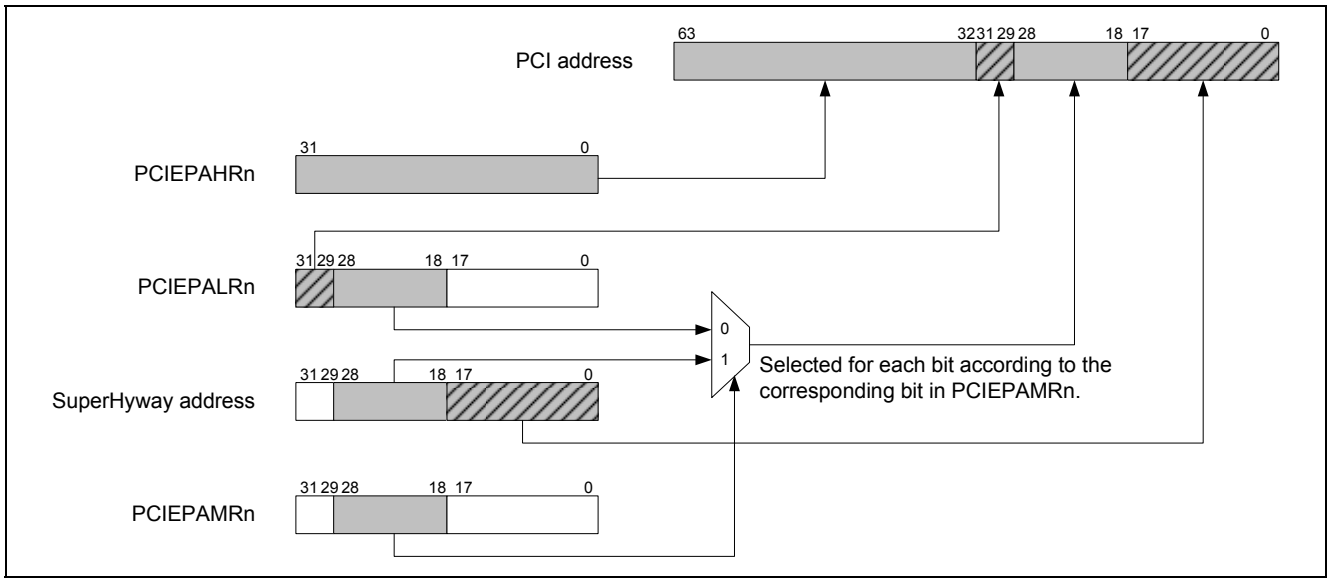

Figure 2.2 Address Conversion to PCI Space

2.6 Target Transfers (Data Transfer from External Device to PCIEC)

Target transfers are described below. As used here, target transfer refers to the reception of a PCI Express packet from an external device by the PCIEC and to the transfer of data to another module in the SH7786 via the SuperHyway bus.

(1) Overview

In a target transfer, an external device accesses the PCIEC by using a PCI Express packet, a request to the SuperHyway bus is generated, and data is transferred to another module. Target transfers enable an external device to perform read/write access to another module in the SH7786 or to external memory connected to it, such as DRAM, by transmitting memory read/write or I/O read/write packets.

In a target transfer, the PCIEC can receive packets of any data length less than or equal to the specified Max Payload Size (MPS). When a transfer using a size greater than that supported by the SuperHyway bus is specified, the PCIEC splits the packet and generates multiple internal bus requests.

(2) Address Map (PCI Express Space)

Figure 2.3 illustrates mapping of the PCI space to the SuperHyway space.

The assignment of addresses in the PCI Express space is dynamically determined by the root port during the configuration cycle, based on the register settings at initialization. The register settings at initialization specify the size and type (memory space or I/O space, etc.) of each area to be allocated. When initialization is completed by setting CFINIT to 1, the initialization details are reflected in the value of Base Address Register n (BARn) or the R/W attributes in the configuration registers. Here, n represents the BAR register number: n = 0 to 1 for a root port and n = 0 to 5 for an endpoint.

The root port references these settings during subsequent configuration cycles, determines the address mapping, and sets the result for each device in BARn in the corresponding configuration register. The address pointed to by BARn serves as the start address assigned to the individual device in the PCI Express space.

The PCIEC supports either a 64-bit or a 32-bit PCI address space (the first 4G area of the 64-bit space) as the area in which memory space is allocated. One BARn register is used to allocate an area in a 32-bit address space, and two contiguous BARn registers (BARn+1/BARn) are used to allocate an area in a 64-bit address space. For this reason, a maximum of one 64-bit address space area can be allocated for a root port and a maximum of three 64-bit address space areas for an endpoint.

In I/O spaces, areas are allocated using one BAR register.

Accesses to BARn by the PCI Express are received by the PCIEC, which converts them into accesses to the SuperHyway bus. The conversion destination address is specified by PCIELARn.

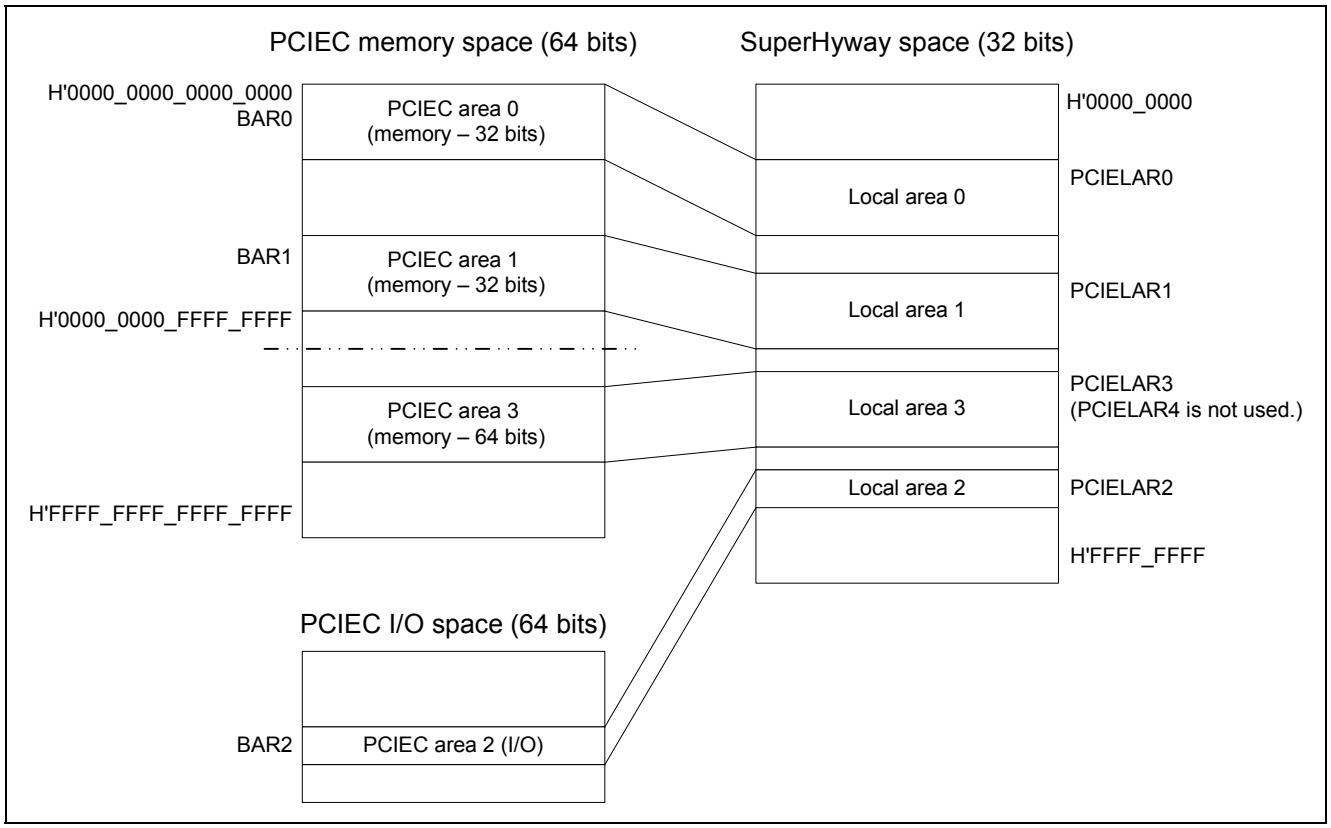

Figure 2.3 Mapping of PCI Spaces to SuperHyway Space

(3) Register Settings for Target Transfers

Table 2.7 lists the transfer control registers for target transfers. These registers control access to areas allocated in the PCI spaces and access to the internal bus from the allocated areas.

The PCIEC has six sets of target transfer registers. The PCIEC can allocate a maximum of two PCI areas in a PCI space when used as a root port and a maximum of six PCI areas in a PCI space when used as an endpoint. The PCIEC supports allocation of 64-bit or 32-bit spaces as memory space that can be allocated in a PCI space. One set of target transfer registers allocate one PCI space when a 32-bit space is used, and two sets of transfer registers allocate one PCI space when a 64-bit space is used.

Table 2.7 Transfer Control Registers for Target Transfers

PCIELARLn	Start address of the local bus (SuperHyway) space to which PCI area n will be mapped
PCIELAMRn	Specifies size of PCI area n.

Note: The value of n is 0 or 1 for a root port and 0 to 5 for an endpoint.

PCIELARLn specifies the address on the SuperHyway bus to which area BARn is mapped. The value of n can be 0 or 1 for a root port and 0 to 5 for an endpoint.

PCIELAMRn specifies the size and type (memory space, I/O space, etc.) of the PCI area allocated in the PCI space, and whether the area is enabled or disabled. The area cannot be allocated in the PCI space if it is not enabled in PCIELAMRn, and no transfers will be performed to the internal bus. (The initial setting after a reset is disabled for all areas.)

(4) Conversion from PCI Address to SuperHyway Bus Address

Figure 2.4 illustrates decoding of PCI space addresses, and figure 2.5 illustrates conversion of PCI addresses to SuperHyway addresses.

When a PCI Express packet is received, first its address is decoded. The address decoding differs depending on whether the address width of the received packet is 32 or 64 bits. If the address width is 32 bits, the address in the received packed is compared with BARn to determine the matching n value. Then the corresponding PCIELARn and PCIELAMRn are used to convert the address into a SuperHyway bus address. If the address width of the received packet is 64 bits, the 64-bit address in the received packet is compared the 64-bit address obtained by combining BARn+1 and BARn to determine the matching n value. Then the corresponding PCIELARn and PCIELAMRn are used to convert the address into a SuperHyway bus address.

The registers PCIELARn+1 and PCIELAMRn+1 are not used in this process.

The lower bits (bits 17 to 0) of the SuperHyway bus address after conversion are generated from the lower bits of the received PCI packet. For the middle bits (bits 28 to 18), the corresponding bits of the received packet address or PCIELARn are used according to the PCIELAMRn bit values. For the upper bits (bits 31 to 29), bits 31 to 29 in PCIELARn are used without modification.

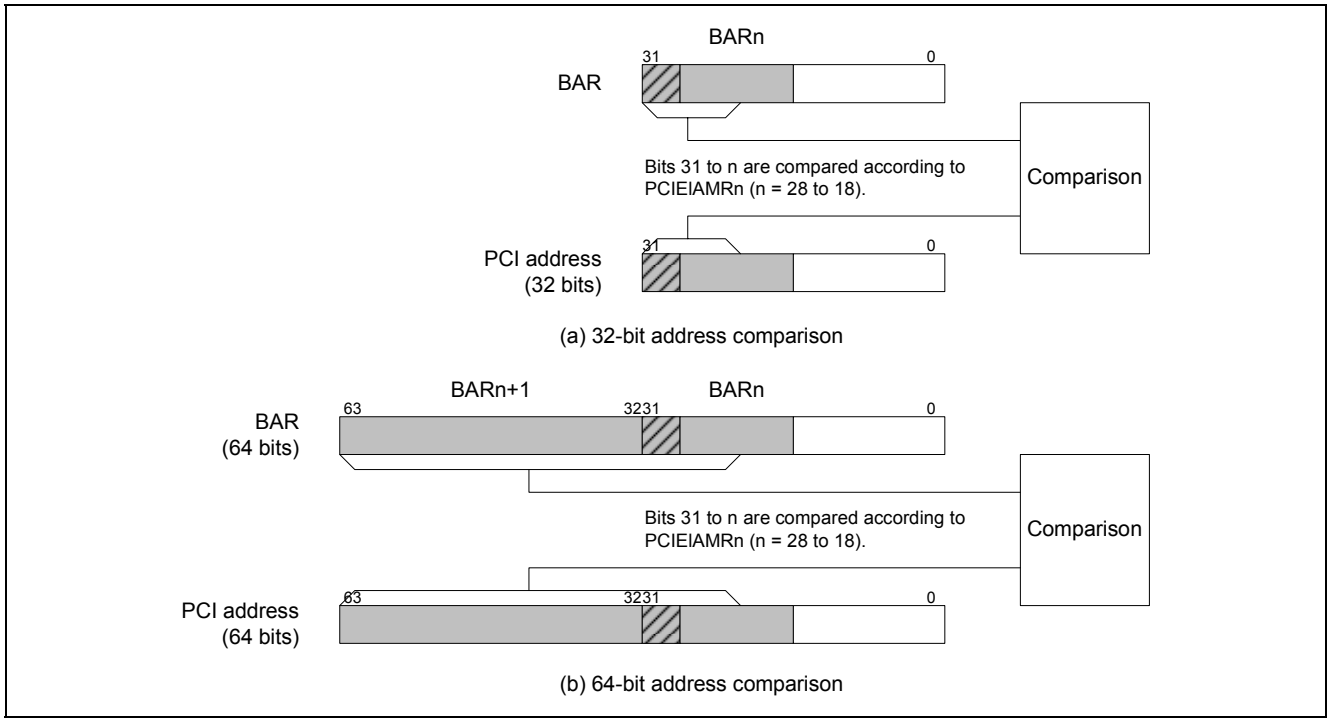

Figure 2.4 PCI Space Address Decoding

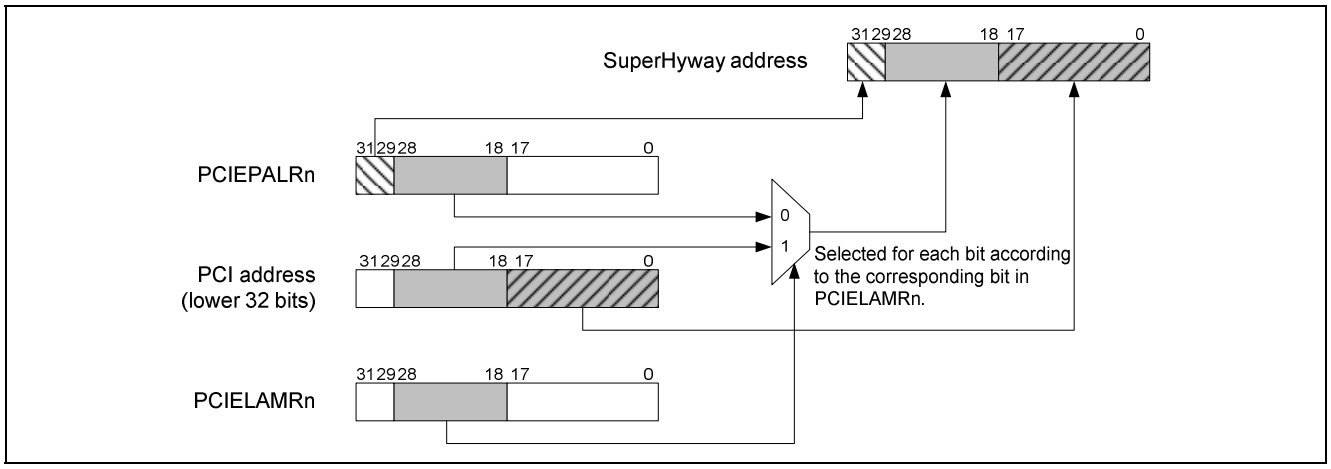

Figure 2.5 Conversion from PCI Address to SuperHyway Address

(5) Accessing the SuperHyway Bus from the PCI Express

The internal bus spaces that can be accessed by the PCI Express via the PCIEC are CS2#, CS3#, the DBSC space, and other PCIEC modules. Here, the other PCIEC modules that can be specified as transfer destinations are PCIEC1 and PCIEC2 when access is made from PCIEC0, PCIEC0 and PCIEC2 when access is made from PCIEC1, and PCIEC0 and PCIEC1 when access is made from PCIEC2.

2.7 DMA Transfer

DMA transfer using the DMAC incorporated in the PCIEC (PCIEC-DMAC) is described below.

(1) Overview

The PCIEC-DMAC enables efficient data transfer between the PCI Express and other modules or external memory devices that are connected via the SuperHyway bus. The PCIEC-DMAC is designed so that it can issue packets with a maximum data length of 1,024 bytes* to the PCI Express, making possible high-speed data transfers that fully exploit the high transfer capacity of the PCI Express.

Note: * Max Payload Size defines the maximum length of the packet data issued to the PCI Express.

The PCIEC-DMAC supports stride transfers for the transfer of data from non-contiguous areas and command chains as a function for executing multiple transfer commands. The stride transfer function enables transfers in which non-contiguous areas serve as the transfer source or destination by adding an offset to the transfer source or destination address after performing a fixed number of transfers. The command chain function treats a set of DMAC settings, such as transfer source and destination addresses and transfer sizes, as a command. By providing a function that sequentially reads and executes commands that are stored in memory, the PCIEC-DMAC supports the continuous execution of multiple transfers without CPU intervention.

(2) Features

- Number of channels: 4
- Address space: PCI Express = 64 bits, SuperHyway bus = 32 bits
- Transfer data length: PCI Express = 4 bytes to 1 KB, SuperHyway bus = 4 to 32 bytes
- Maximum transfer count: 536,870,912 (2^{29})
- Addressing mode: Dual mode
- Transfer requests: Auto-request (started by register control)
- Data transfer: Normal mode (continuous transfer), stride transfer, command chain
- Priority: Selectable between channel priority fixed mode and round-robin mode
- Interrupt requests: Interrupt requests can be issued to the INTC when a data transfer completes or when an error occurs.

(3) DMAC Transfer Requests

The PCIEC-DMAC supports auto-request mode. The PCIEC-DMAC is activated by writing to its registers by the CPU or other unit.

(4) Channel Priority

When receiving simultaneous transfer requests with respect to multiple channels, the PCIEC- DMAC performs transfers according to the specified priority. The priority of channels can be selected from two modes: fixed and round-robin. The mode is selected by the ABT bit in the PCIEDMAOR register.

To improve transfer efficiency, the PCIEC-DMAC uses the largest possible PCI Express packets for transfer. Once transmission or reception processing starts, it is not interrupted until transfer processing for the packet is completed. For this reason, even if a transfer request with a higher priority becomes executable, no channel switching occurs until the current packet transmission at that stage completes. Channel switching cannot occur until a transfer of up to 4 KB finishes.

Channel switching occurs when a data transfer set in the active channel completes. Here, transfer set completion means the point in time when the transfers for both the SuperHyway bus and PCI Express have completed.

(a) Fixed Mode

In fixed mode, the channel priority does not change. The priority is fixed as follows.

CH0 > CH1 > CH2 > CH3

(b) Round-Robin Mode

In round-robin mode, when a transfer set has completed in one channel, the channel priority changes so that the completed channel has the lowest priority.

(5) Transfer in Normal Mode

In a normal-mode transfer, data is transferred from a specified source address to a specified destination address. Either of the following transfer directions can be selected: PCI to SuperHyway bus or SuperHyway bus to PCI.

The procedure for performing normal-mode transfers with the PCIEC-DMAC is described below. For detailed specifications of the individual registers, see 13.4.4, PCIEC-DMAC Control Registers, in *SH7786 Group User's Manual: Hardware* (REJ09B0501).

(a) General PCIEC-DMAC Settings

Make settings in the PCIEDMAOR register to enable DMA and select the arbitration type.

(b) Transfer Settings

Specify the PCI and SuperHyway addresses and byte count, and the transfer termination interrupt.

Specify the source and destination addresses in the registers PCIEDMPALRn and PCIEDMPAHRn, PCIEDMSALRn, and PCIEDMBCNTRn, where n denotes the channel number (0 to 3). Regardless of the direction of transfer, specify the PCI address in PCIEDMPALRn and PCIEDMPAHRn and the SuperHyway bus address in PCIEDMSALRn.

To generate an interrupt when a transfer is completed, specify an interrupt setting in the PCIEDMCHSRn register.

If the stride transfer function will not be used, clear PCIEDMSBCNTRn and PCIEDMSTRRn to 0.

If the command chain function will not be used, clear PCIEDMCCARn to 0.

(c) Activating the DMAC

In PCIEDMCHCRn, specify the direction of transfer, and at the same time initiate the transfer process by enabling the channel.

If the stride transfer function will not be used, clear the SARE bit (bit 24) and PARE bit (bit 25) in PCIEDMCHCRn to 0.

If the command chain function will not be used, clear the CCRE bit (bit 29) in PCIEDMCHCRn to 0.

(d) Waiting for Transfer End

The end of the transfer can be determined by confirming that the TE bit (bit 0) in PCIEDMCHSRn is set to 1 or by detecting a transfer end interrupt.

(e) End Processing

Complete the transfer by clearing the CHE bit in (bit 31) in PCIEDMCHCRn to 0. Also, write 1 to the TE bit (bit 0) in PCIEDMCHSRn to clear it to 0.

The next DMA transfer cannot be started unless end processing is performed.

(6) Stride Transfer

In a stride transfer, a procedure called “striding” is used in which an offset is added to the source or destination address after the transfer of a specific number of bytes. By applying striding to the destination address, scatter transfer can be performed. Similarly, by applying striding to the source address, gather transfer can be performed. By applying striding to both the source and destination addresses, non-contiguous regions can be transferred.

To use stride transfer, set a stride interval (stride counter) in PCIEDMSBCNTRn and a stride width in PCIEDMSTRRn when specifying transfer settings. To use stride transfer on the PCI side or SuperHyway side only, set the stride width (SS or PS field in PCIEDMSTRRn) to 0 for the non-stride transfer side.

To start the DMAC, set the SARE bit (bit 24) or PARE bit (bit 25) in PCIEDMCHCRn to 1.

The other settings are the same as those for normal-mode transfer.

(7) Command Chain

The command chain function enables the consecutive execution of multiple DMAC commands. Here, DMAC command refers to a set of information that specifies a PCIEC-DMAC transfer; that is, the information specified in PCIEDMPALRn, PCIEDMSALRn, PCIEDMBCNTRn, PCIEDMSBCNTRn, PCIEDMSTRRn, PCIEDMCCARn, and PCIEDMCHCRn. This information can be set in the PCIE-DMAC control registers and also in the memory in the format shown in figure 2.6. (The upper 32 bits of the address on the PCI side cannot be specified by a DMAC command. The address specified in the PCIEC-DMAC control registers is always used.) The command chain function enables the PCIEC-DMAC to read the next DMAC command from memory after execution of a DMAC command, to write the DMAC command contents to the PCIEC-DMAC control registers, and to execute the DMAC command. By specifying the next DMAC command in each DMAC command that is read, a DMAC command chain can be built and consecutive transfers performed.

When a command chain is used, the PCIEC-DMAC first executes the DMAC command specified in the PCIEC-DMAC control registers for each channel. After execution of this DMAC command, the PCIEC-DMAC reads the next DMAC command from the address specified in PCIEDMCCARn, writes the command contents to the PCIEC-DMAC control registers for the corresponding channel, and executes it. If the value of the CCRE bit in the newly read DMAC command is 1, the PCIEC-DMAC reads the next command again from the memory and executes it after completion of that command. If the value of the CCRE bit in the read DMAC command is 0, execution of the series of commands in the chain ends upon completion of that command.

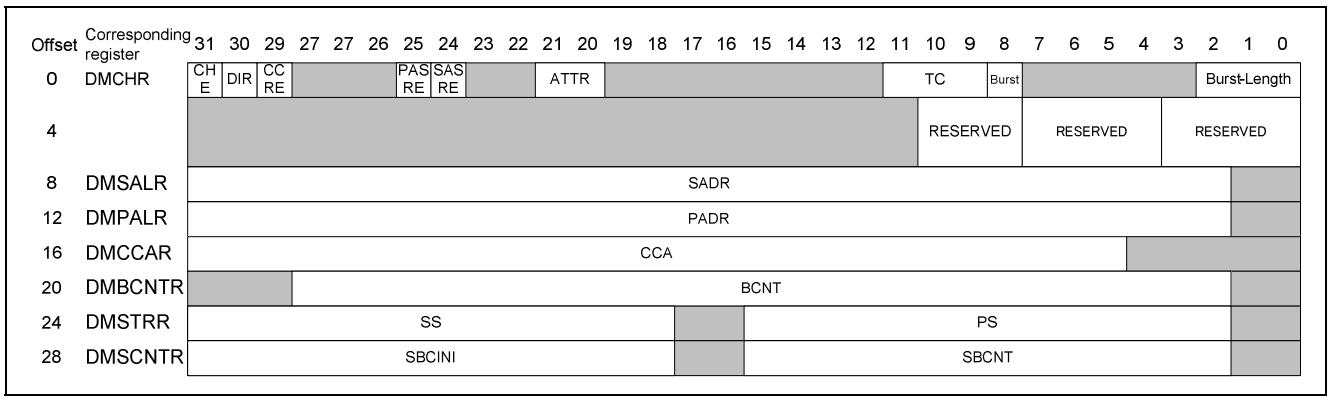

Figure 2.6 PCIEC-DMAC Command Format

Command chain execution starts when a channel is enabled while the CCRE bit (bit 29) in PCIEDMCHCRn is set to 1. Before starting a command chain, it is necessary first to store the chain of DMAC commands in a memory location accessible from the SuperHyway bus (DDR3-SDRAM, LBSC, IL memory, OL memory, or shared memory specified in L2CR), and to specify the address of the first command in PCIEDMCCARn.

Each DMAC command stored in memory must satisfy the conditions listed below. The DMAC commands should be located in a shared memory location such as DDR, LBSC, or LRMA.

- CHE field
  This must always be set to 1.
- ATTR field
  Set the ATTR field of the DMAC command in memory to the same value as that specified in the ATTR field in the PCIEC-DMAC control register. The value of the ATTR field cannot be modified by loading a command.
- TC field
  Set the traffic class (TC) field of the DMAC command in memory to specify the same virtual channel (VC0) as that specified by the TC field in the PCIEC-DMAC control register.
- RESERVED field
  This must always be set to 8.
- CCA field
  The value of the CCA field of the final command executed must be 0.

(8) PCIE-DMAC Interrupt Sources

The PCIEC-DMAC generates an interrupt for each channel at transfer end as well as an interrupt at error termination on a common basis for all channels.

3. Serial Communication Interface (SCIF0)

The serial communication interface (SCIF0) incorporates FIFO buffers and supports both asynchronous and clock synchronous serial communication.

Note that SCIF0 is used as an asynchronous serial console by the sample program.

For a detailed description of SCIF0, see section 24, Serial Communication Interface with FIFO (SCIF), in *SH7786 Group User's Manual: Hardware* (REJ09B0501).

4. Application Example

4.1 AP-SH4AD-0A SH7786 Evaluation Board

The sample program presented in this application note uses two AP-SH4AD-0A SH7786 evaluation boards, manufactured by Alpha Project, and operates the PCIEC on one of them as a PCI Express root port and the other as a PCI Express endpoint. For details of the AP-SH4AD-0A, see *AP-SH4AD-0A Hardware Manual*.

4.1.1 Memory Map

Table 4.1 shows a memory map of the AP-SH4AD-0A.

Table 4.1 AP-SH4AD-0A Memory Map

Area	Address	Connected Device	Bus Width
0	H'0000_0000 to H'00FF_FFFF	S29GL128P90TFIR20 (16 MB)	16 bits
	H'0100_0000 to H'03FF_FFFF	Shadow	
1	H'0400_0000 to H'0400_0FFF	LAN9221 (512 B)	16 bits
	H'0400_1000 to H'07FF_FFFF	Shadow	
2	H'0800_0000 to H'0BFF_FFFF	MT41J64M16LA-187E (256 MB)	32 bits
3	H'0C00_0000 to H'0FFF_FFFF		
4	H'1000_0000 to H'13FF_FFFF		
5	H'1400_0000 to H'17FF_FFFF		
6	H'1800_0000 to H'1BFF_FFFF	Left open by user	32 bits

4.1.2 Settings for PCI Express Root Port Mode

To set the AP-SH4AD-0A to operate in PCI Express root port mode, set the DIP switches as indicated below. For a detailed description of the DIP switches, see section 2, Functions, in *AP-SH4AD-0A Hardware Manual*.

- PCI Express mode setting

SW2-2	
MODE11	**PCI Express Mode**
ON	Root port mode

- PCI Express PHY mode setting

SW2-3	
MODE12	**PCI Express PHY Mode**
ON	4-lane + 1-lane

4.1.3 Settings for PCI Express Endpoint Mode

To set the AP-SH4AD-0A to operate in PCI Express endpoint mode, set the DIP switches as indicated below. For a detailed description of the DIP switches, see section 2, Functions, in *AP-SH4AD-0A Hardware Manual*.

- PCI Express mode setting

SW2-2	
MODE11	**PCI Express Mode**
OFF	Endpoint mode

- PCI Express PHY mode setting

SW2-3	
MODE12	**PCI Express PHY Mode**
ON	4-lane + 1-lane

Note: To operate the AP-SH4AD-0A in PCI Express endpoint mode while supplying power from the PCI Express card edge, use a soldering iron to open solder junction JP1 on the board. For a detailed description of power supply configuration, see section 3.7.1, Power Supply Examples, in *AP-SH4AD-0A Hardware Manual*.

4.1.4 Serial Console Settings

The sample program uses SCIF0 as the serial interface of the AP-SH4AD-0A, and the relevant settings are described below.

The PC-USB-02A is used as the serial console. The PC-USB-02A converts the TTL serial level of SCIF0 into a USB signal, allowing communication with a PC.

For details of the serial interface and console, see 3.7, Serial Interface, in *AP-SH4AD-0A Hardware Manual*.

Table 4.2 Serial Console Settings

Item	Specification
SCIF0	Asynchronous mode
Baud rate	115,200 bps
Data	8 bits
Parity bit	None
Stop bit	1 bit
Flow control	None

4.2 Description of Sample Program

The sample program presented in this application note uses two AP-SH4AD-0A boards, one operating as a PCI Express root port and one as a PCI Express endpoint. The description that follows covers the basic method whereby, after initial PCIEC settings, the PCI Express root port displays on the serial console information from the configuration registers of the PCI Express endpoint.

4.2.1 System Configuration for Sample Program

Two AP-SH4AD-0A boards, once set as a PCI Express root port and one as a PCI Express endpoint, are connected as shown below, and the PC-USB-02A serial console is used to display settings (Vendor ID, Device ID, etc.) from the configuration registers of the PCI Express endpoint on the console PC.

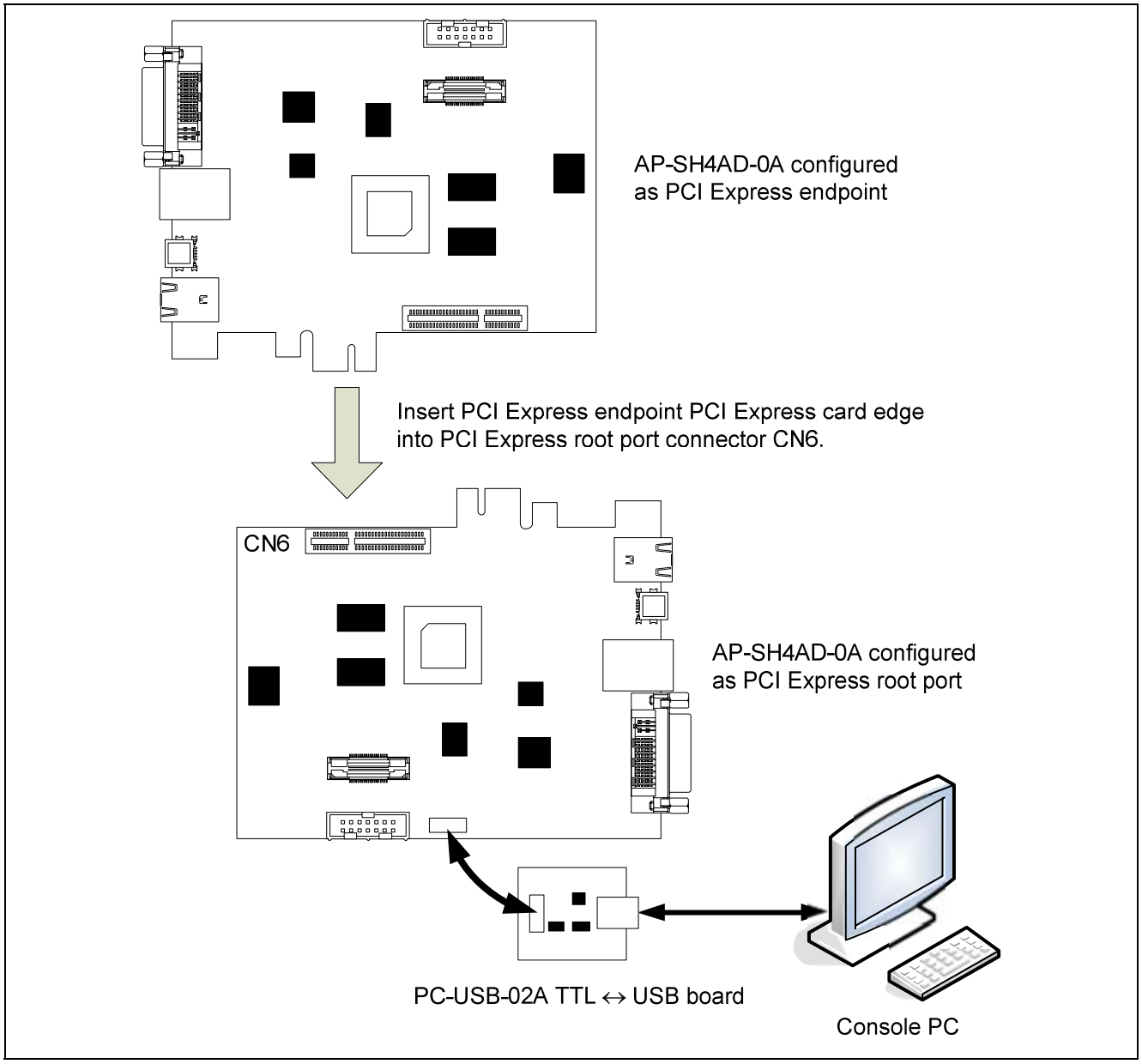

Figure 4.1 System Configuration

4.2.2 Specifications of Sample Program

- Serial console initial settings
- PCIEC initial settings (PCI Express root port mode or PCI Express endpoint mode settings)
- Display of PCI Express endpoint Vendor ID and Device ID
- Data transfer (transmit/receive) to PCI Express endpoint memory area or IO area
- Data transfer (transmit/receive) to PCI Express endpoint memory area by using the on-chip DMA
- The following operations are not supported by the sample program.
  - Stride transfer and command chain operation by using the on-chip DMA
  - Message transmit/receive
  - INTx or MSI interrupts
  - Link power control function (L0, L0s, L1, and L3 states)

4.2.3 Sample Program Flowcharts

(1) Flowchart of main() Function

This flowchart shows the processing sequence from the start of the main() function, which occurs after initial settings to LBSC and DBSC3 following a power-on reset.

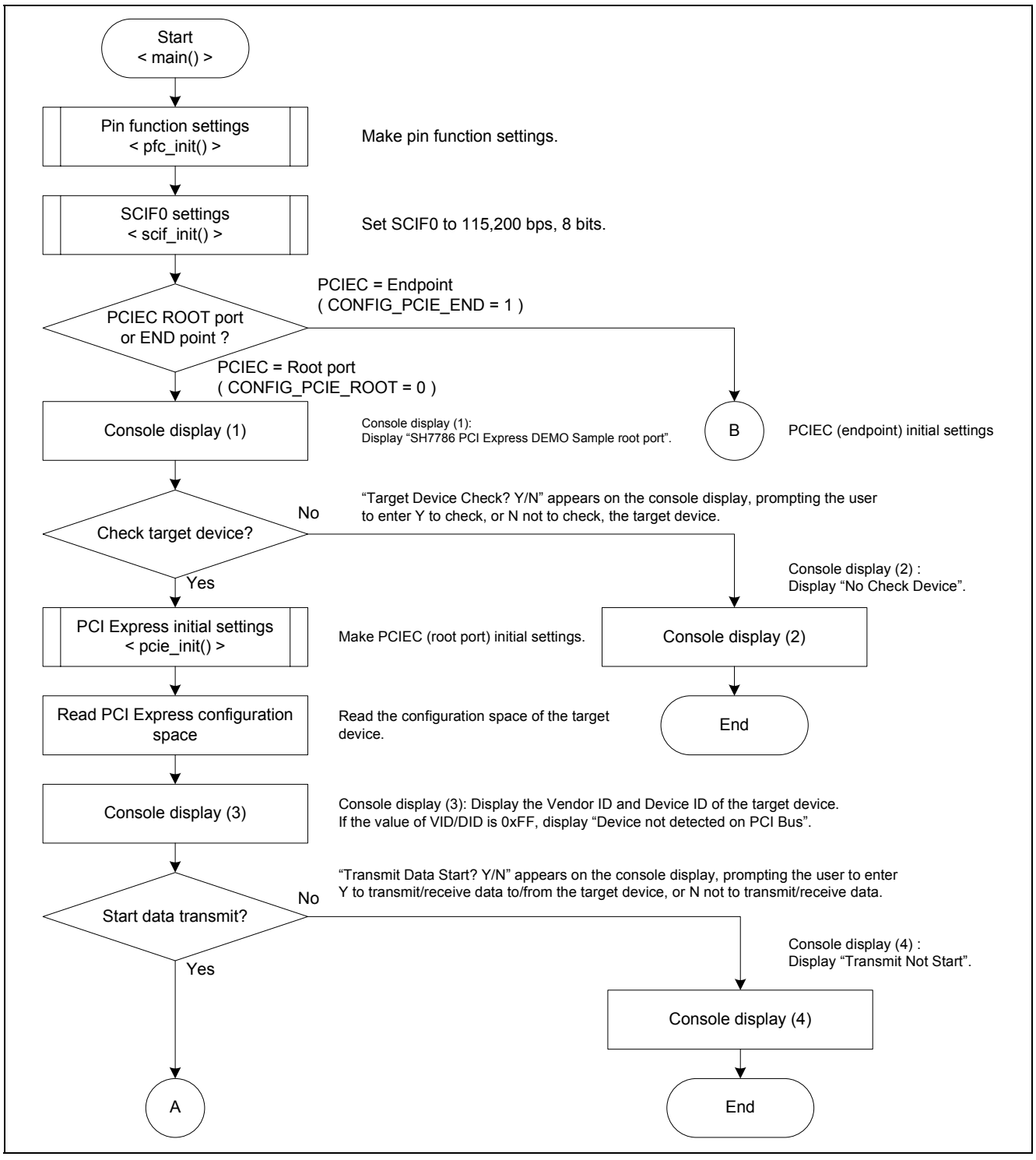

Figure 4.2 Flowchart of main (PCI Express Root Port) (1)

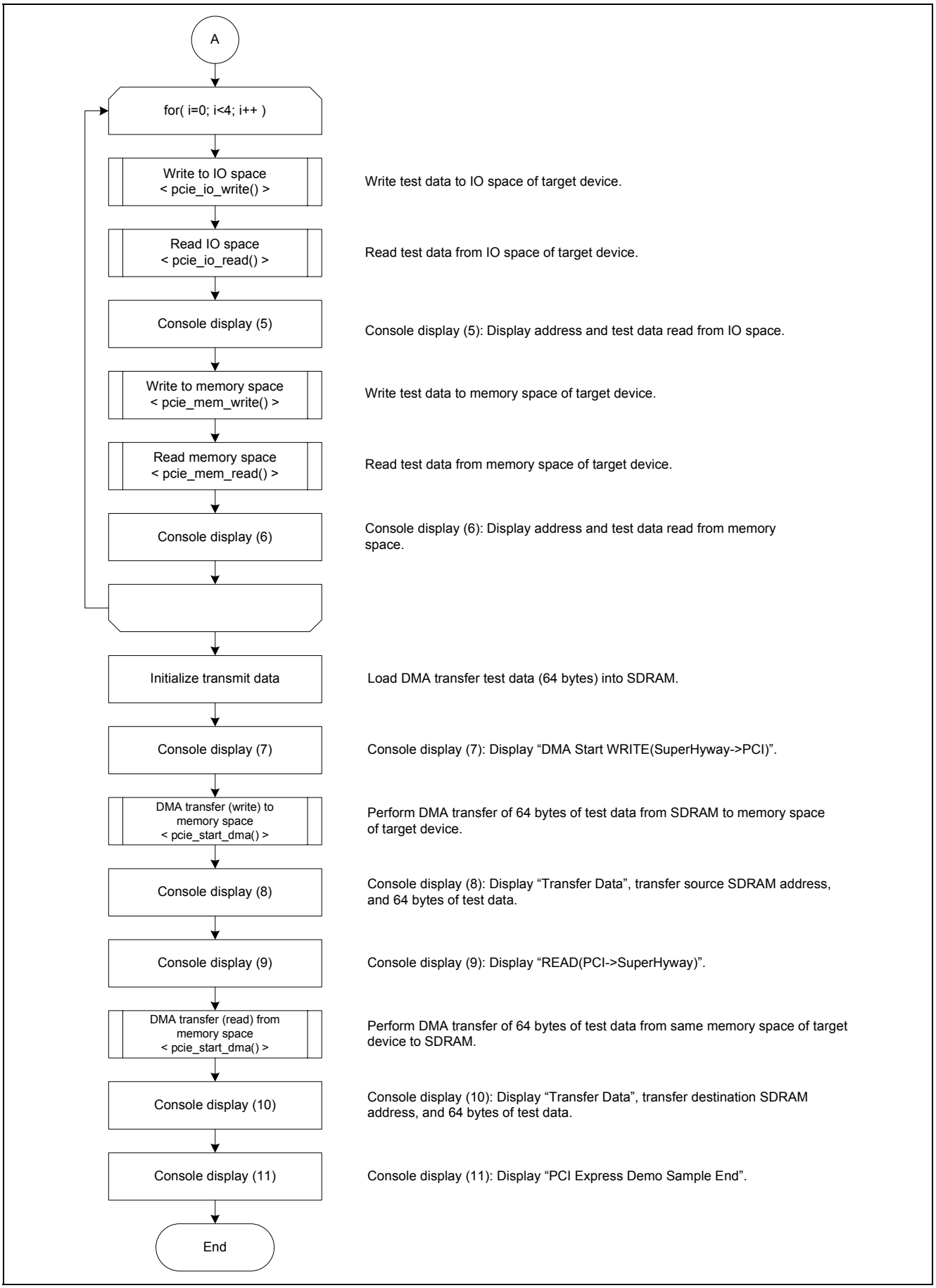

Figure 4.2 Flowchart of main (PCI Express Root Port) (2)

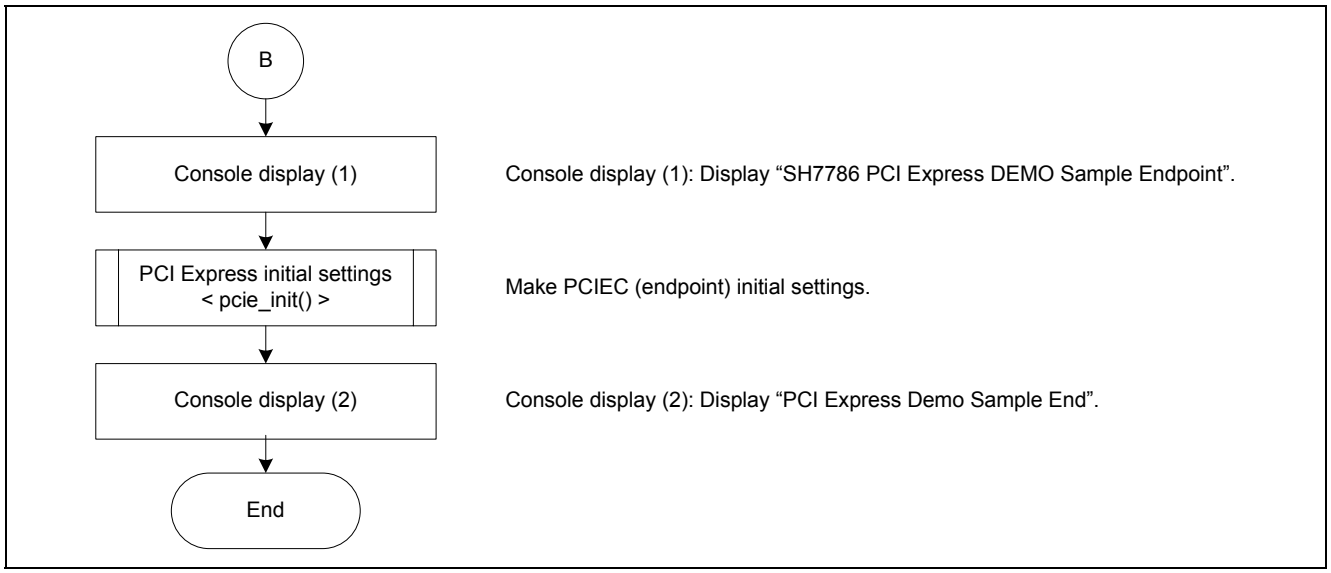

Figure 4.3 Flowchart of main (PCI Express Endpoint)

(2) Flowchart of Pin Function Settings

This flowchart shows the processing sequence for making pin function settings.

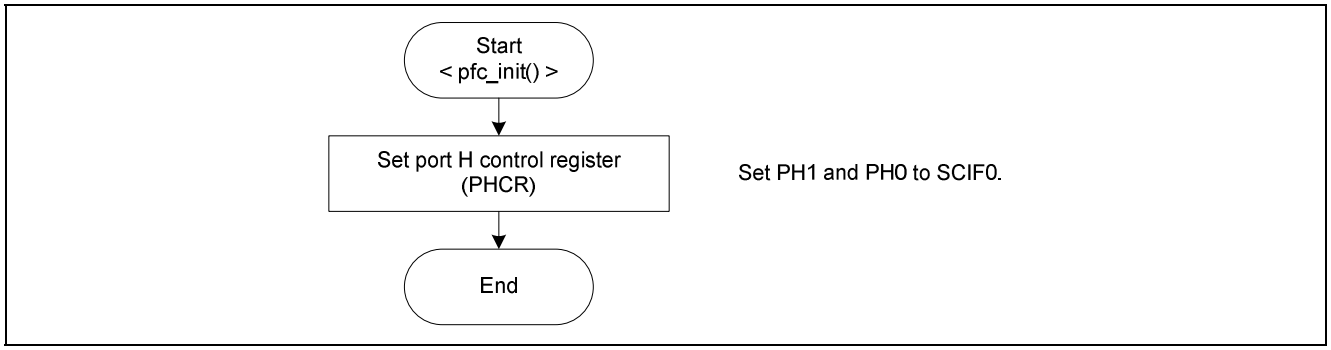

Figure 4.4 Flowchart of Pin Function Settings

(3) Flowchart of SCIF0 Initial Settings

This flowchart shows the processing sequence for making SCIF0 initial settings for use as a serial console.

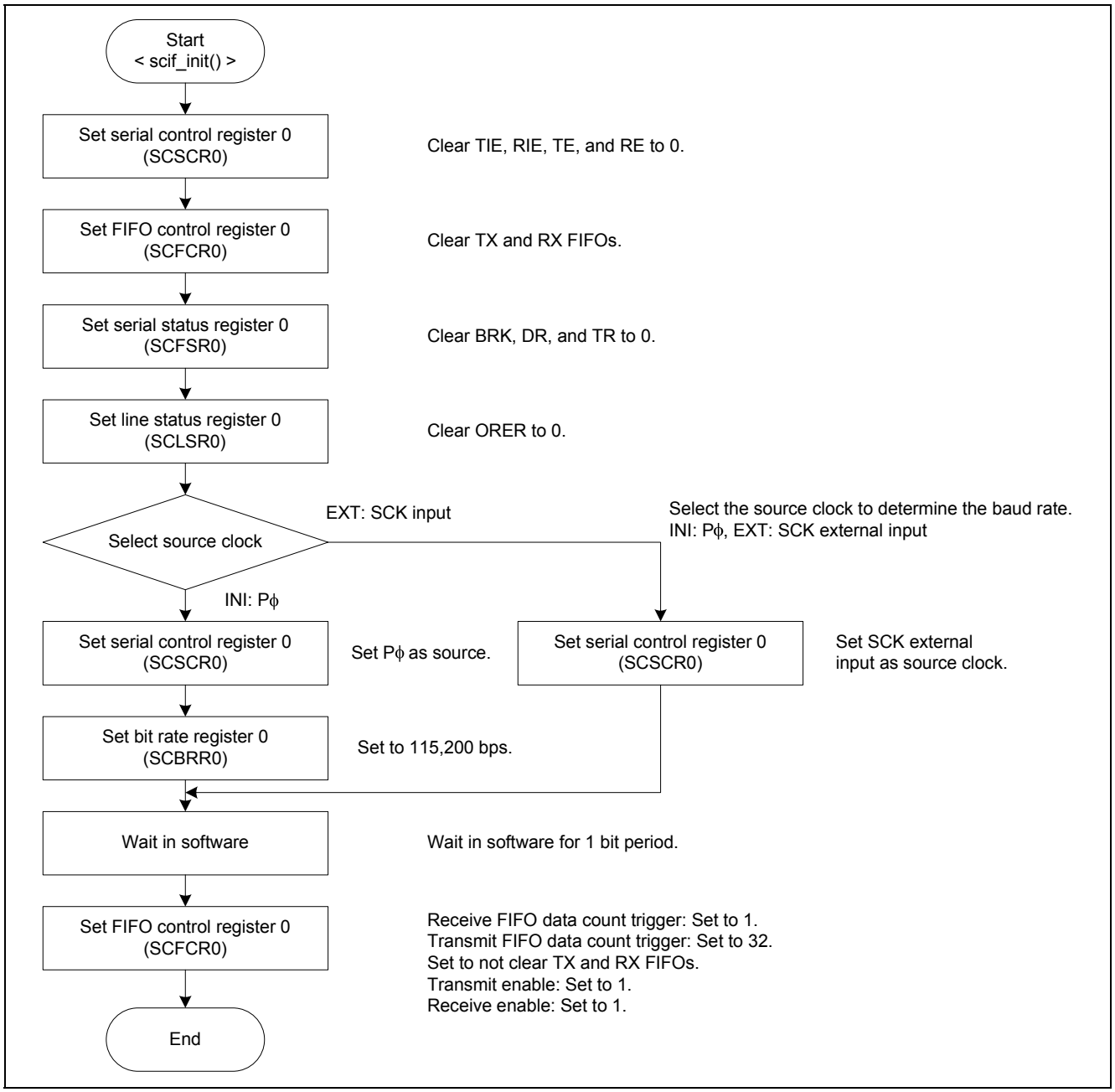

Figure 4.5 Flowchart of SCIF0 Initial Settings

(4) Flowchart of PCI Express Bus Initial Settings

This flowchart shows the processing sequence for making PCI Express bus initial settings.

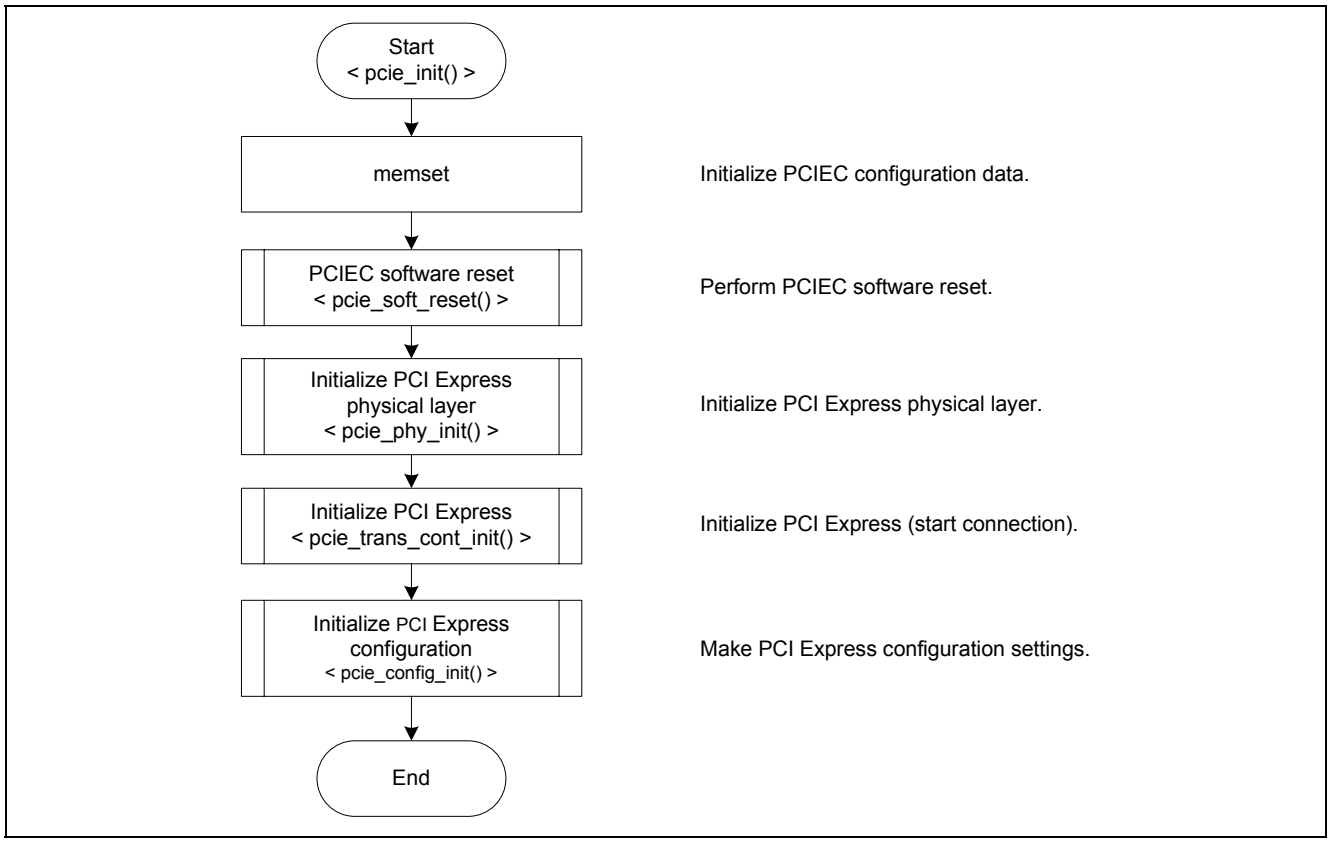

Figure 4.6 Flowchart of PCI Express Bus Initial Settings

(5) Flowchart of PCIEC Software Reset

This flowchart shows the processing sequence for performing a PCIEC software reset.

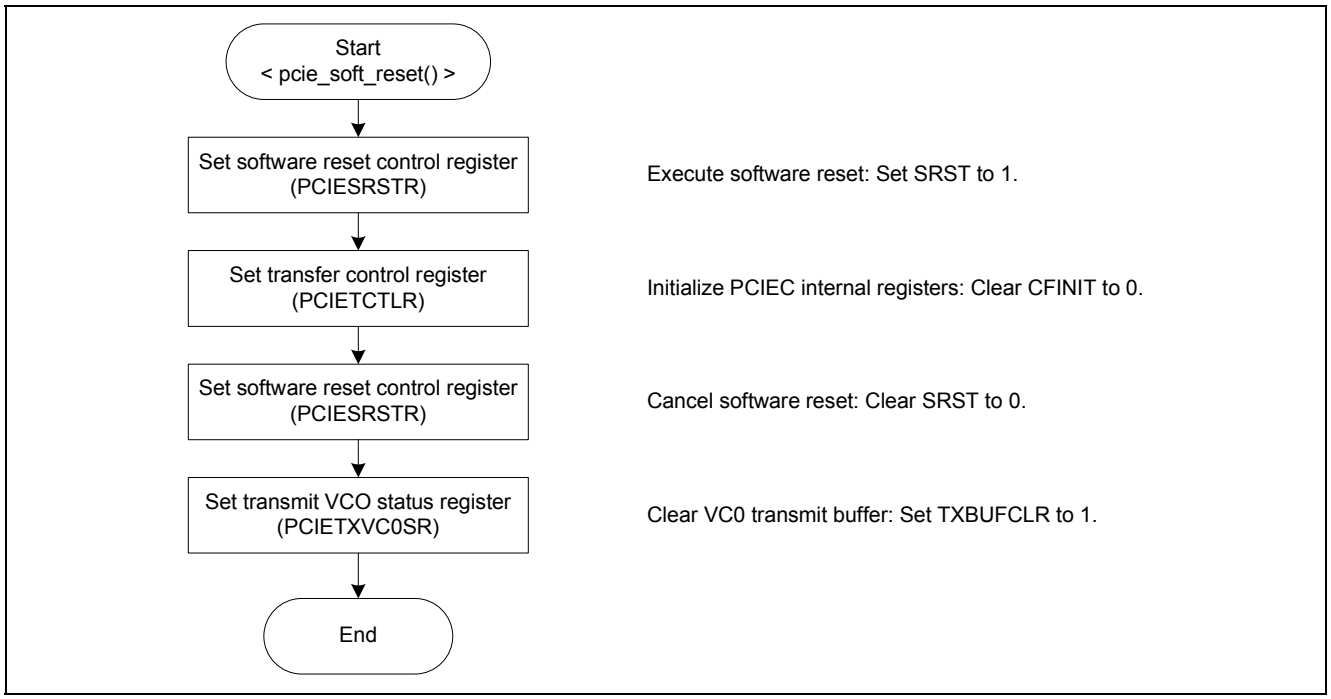

Figure 4.7 Flowchart of PCIEC Software Reset

(6) Flowchart of PCIEC Physical Layer Initialization

This flowchart shows the processing sequence for initializing the PCIEC physical layer.

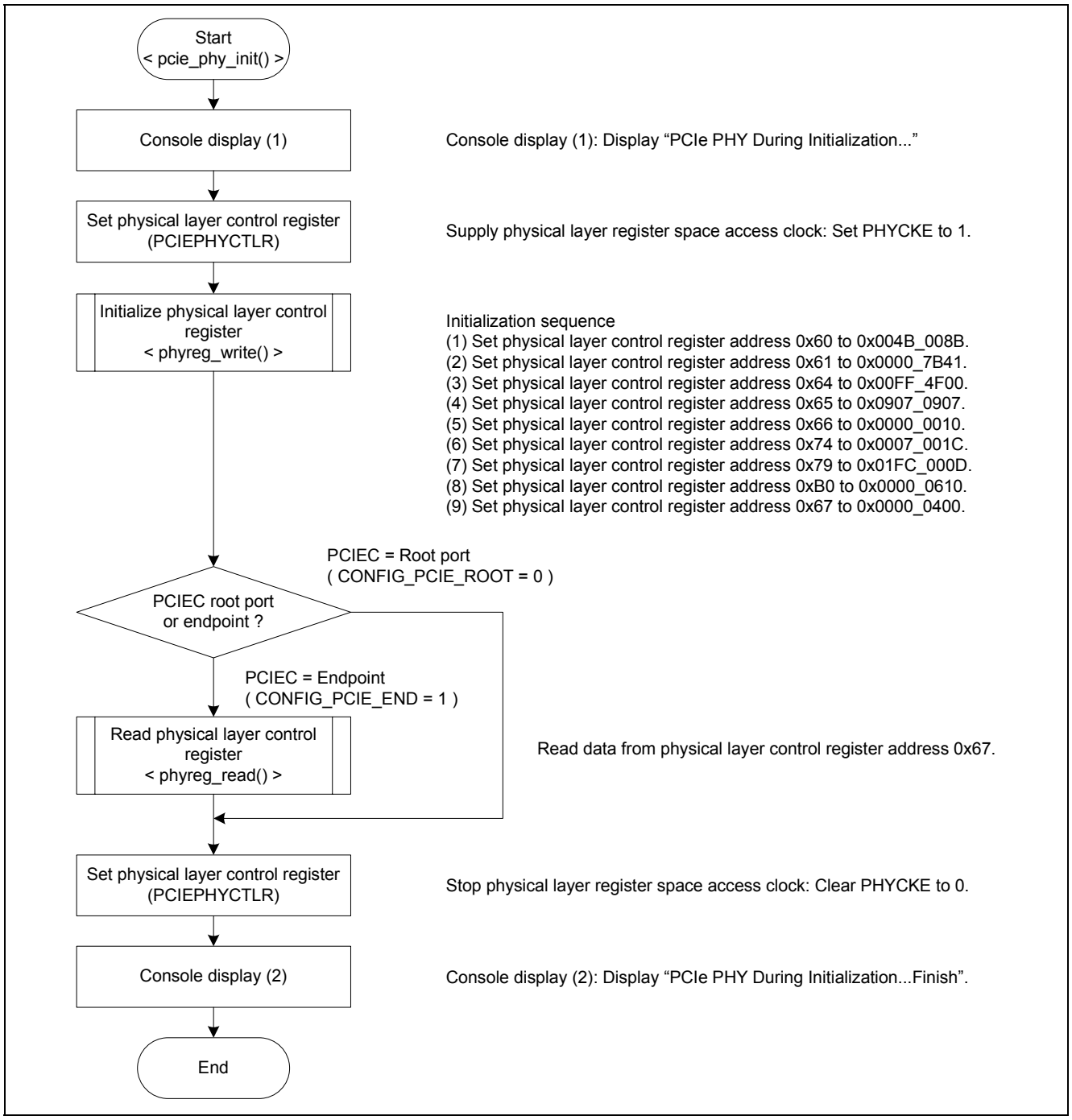

Figure 4.8 Flowchart of PCIEC Physical Layer Initialization

(7) Flowchart of PCIEC Initialization

This flowchart shows the processing sequence for PCIEC initialization.

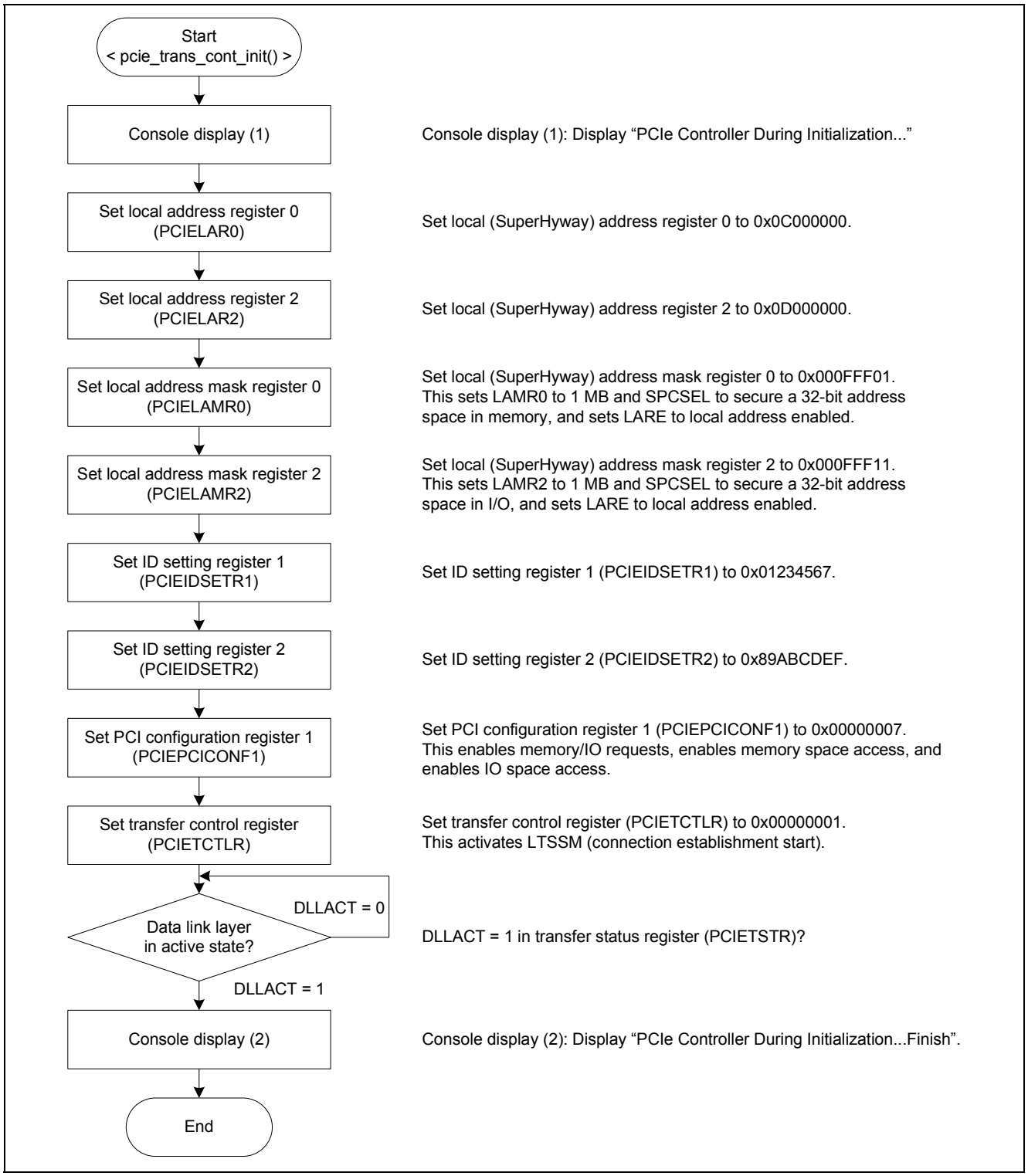

Figure 4.9 Flowchart of PCIEC Initialization

(8) Flowchart of PCI Express Configuration Register Initial Settings

This flowchart shows the processing sequence for making PCI Express configuration register initial settings.

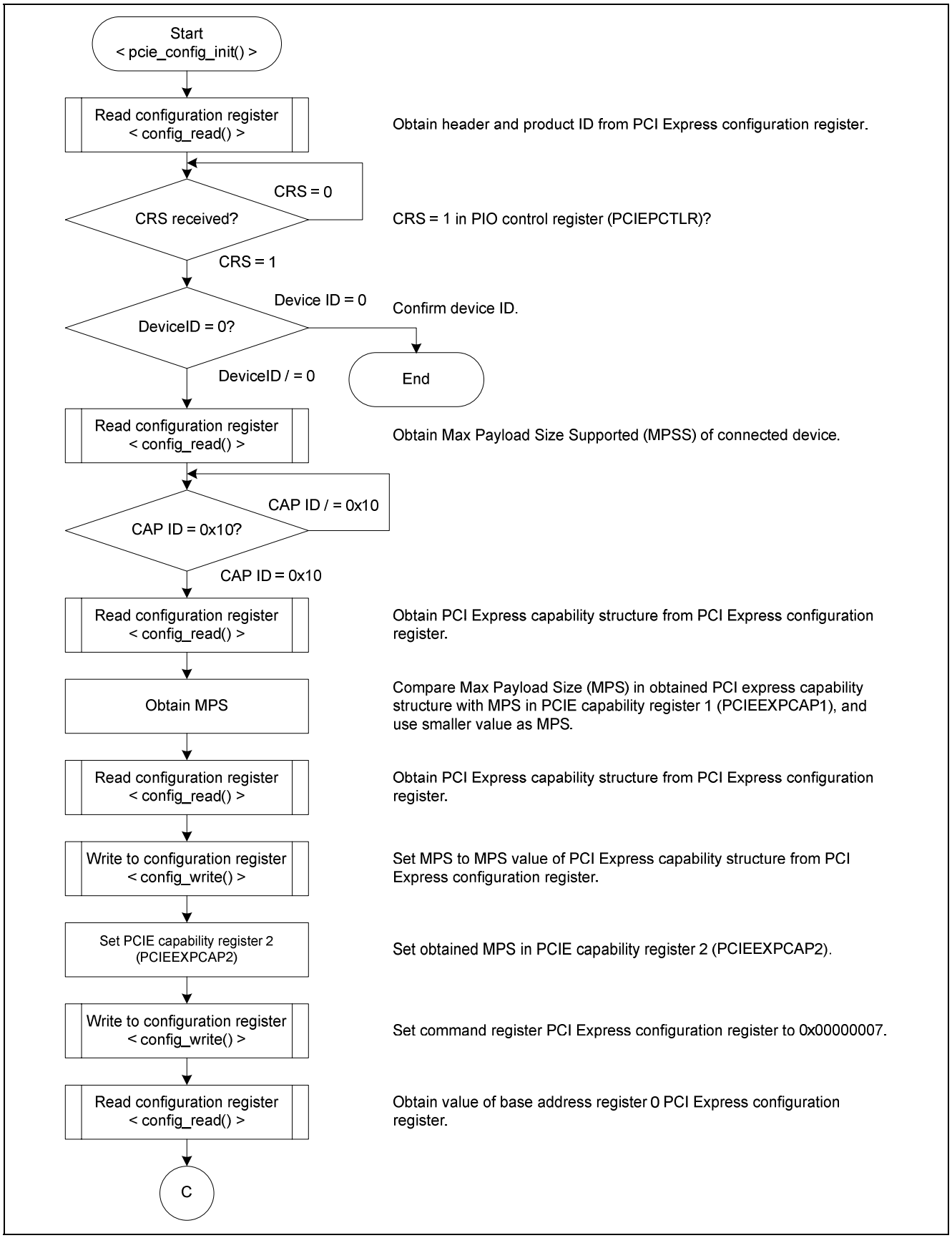

Figure 4.10 Flowchart of PCI Express Configuration Register Initial Settings (1)

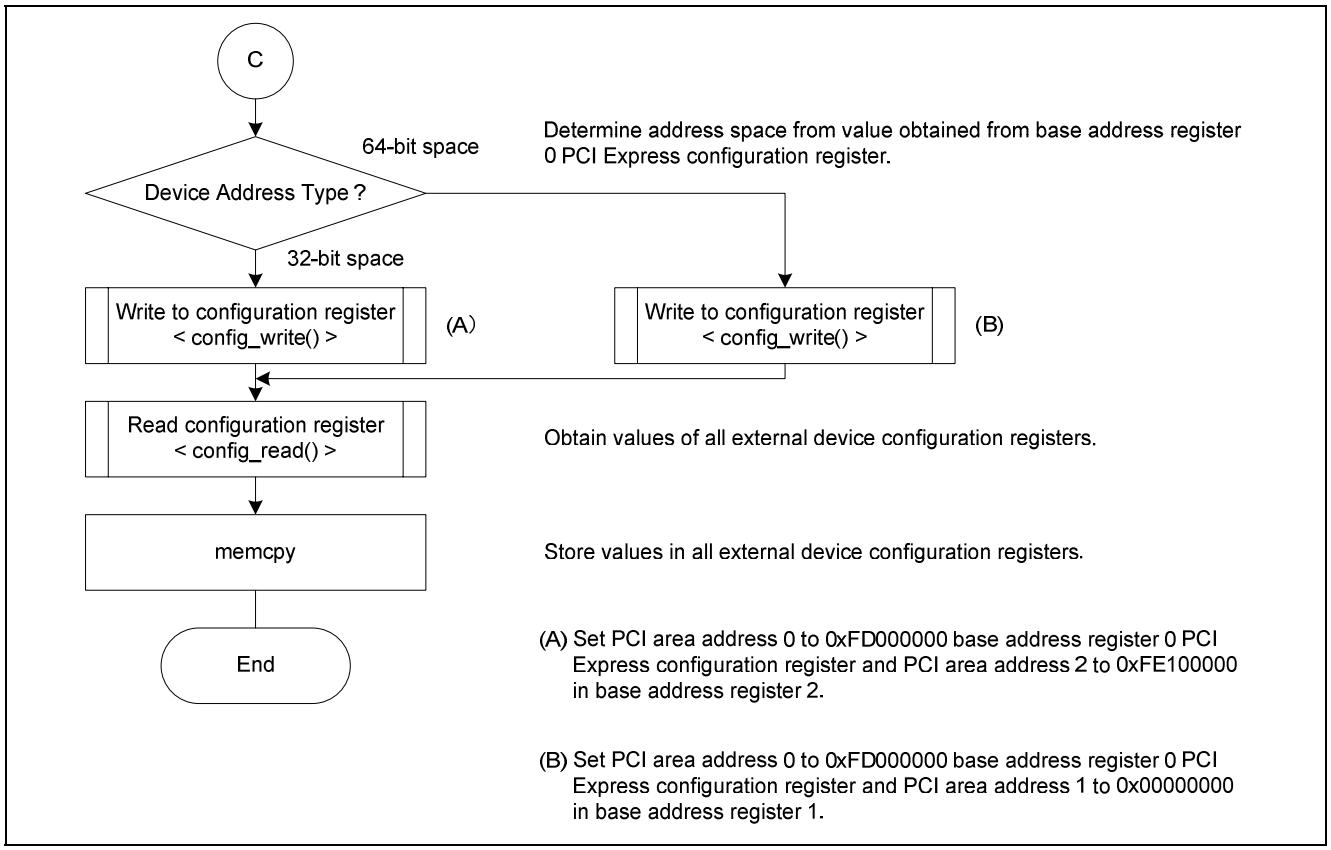

Figure 4.10 Flowchart of PCI Express Configuration Register Initial Settings (2)

(9) Flowchart of PCI Express Memory Transfer (Write)

This flowchart shows the processing sequence for performing a PCI Express memory transfer (write).

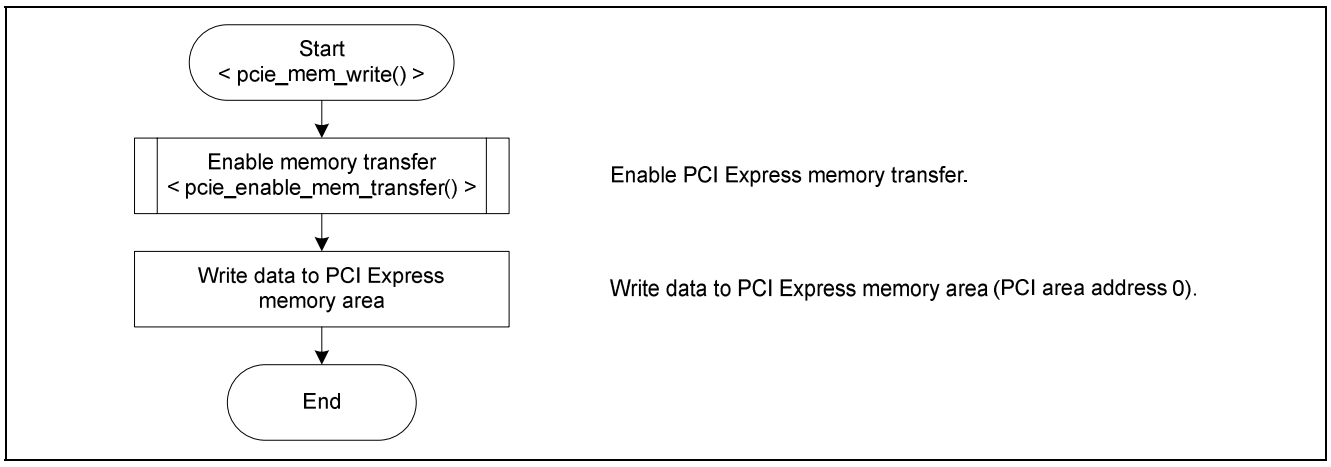

Figure 4.11 Flowchart of PCI Express Memory Transfer (Write)

(10) Flowchart of PCI Express Memory Transfer (Read)

This flowchart shows the processing sequence for performing a PCI Express memory transfer (read).

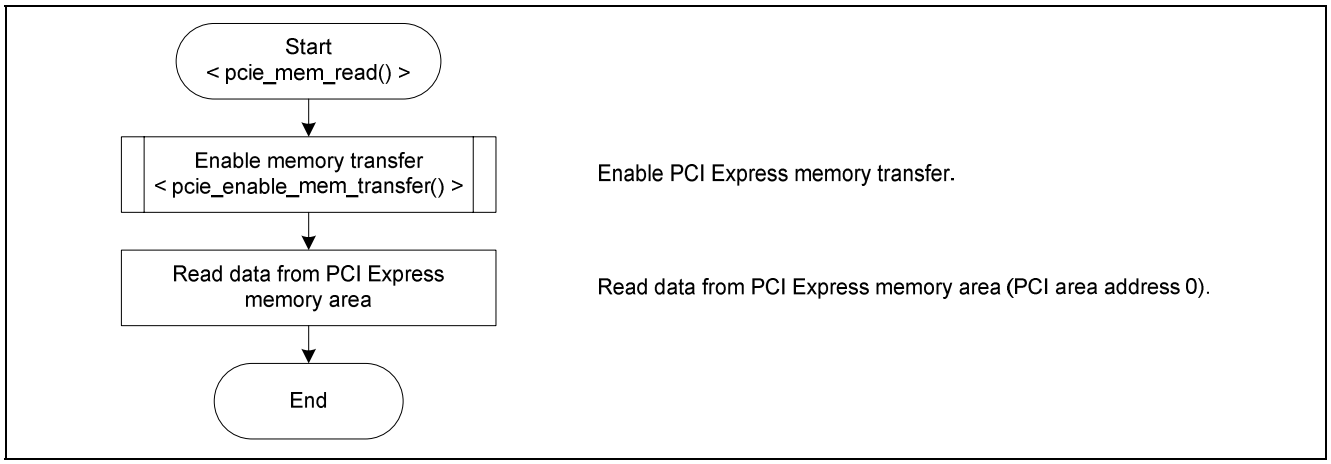

Figure 4.12 Flowchart of PCI Express Memory Transfer (Read)

(11) Flowchart of PCI Express Memory Transfer Enable Settings

This flowchart shows the processing sequence for making PCI Express memory transfer enable settings.

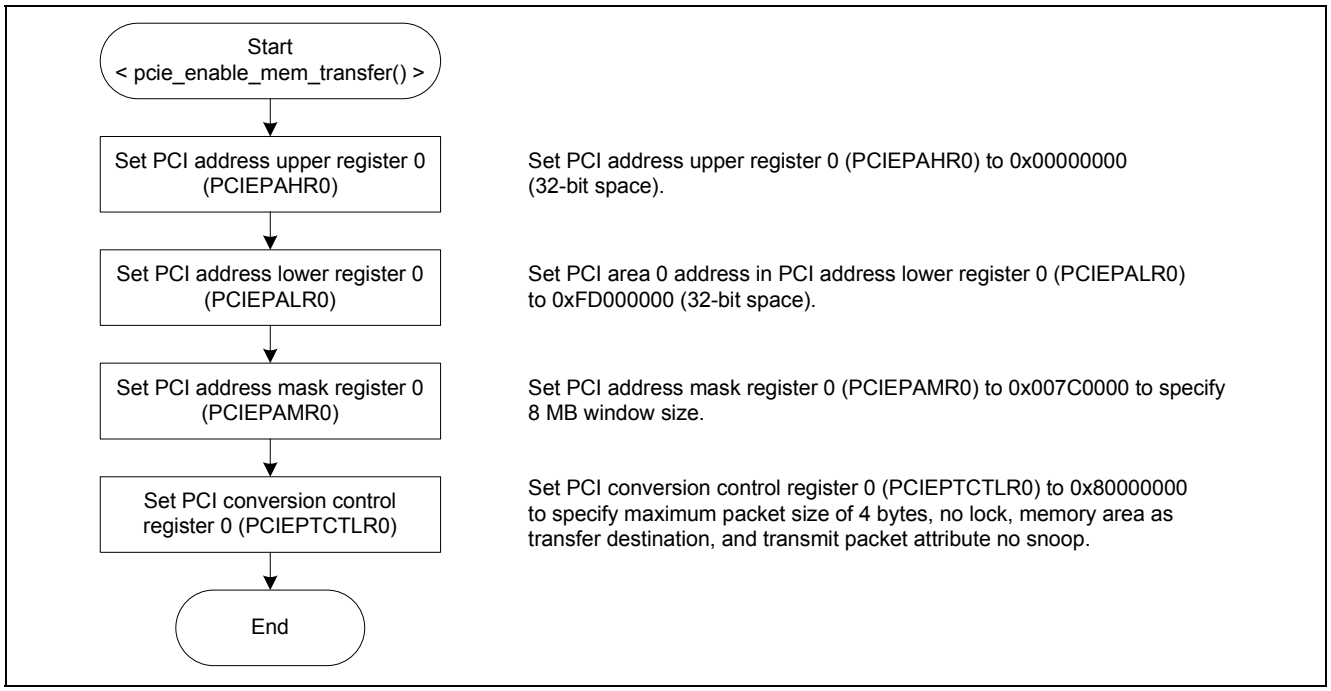

Figure 4.13 Flowchart of PCI Express Memory Transfer Enable Settings

(12) Flowchart of PCI Express IO Transfer (Write)

This flowchart shows the processing sequence for performing a PCI Express IO transfer (write).

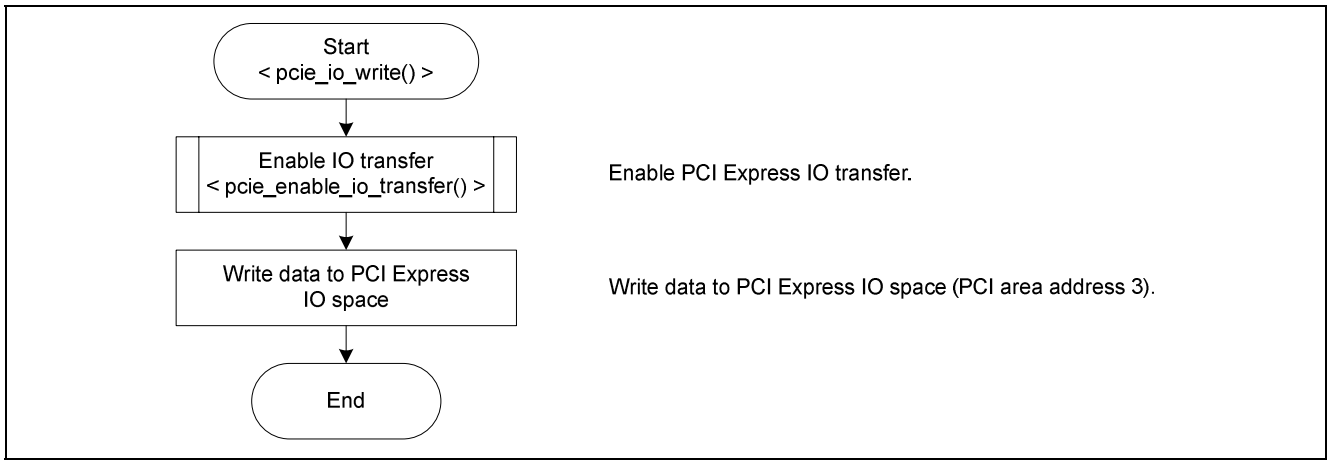

Figure 4.14 Flowchart of PCI Express IO Transfer (Write)

(13) Flowchart of PCI Express IO Transfer (Read)

This flowchart shows the processing sequence for performing a PCI Express IO transfer (read).

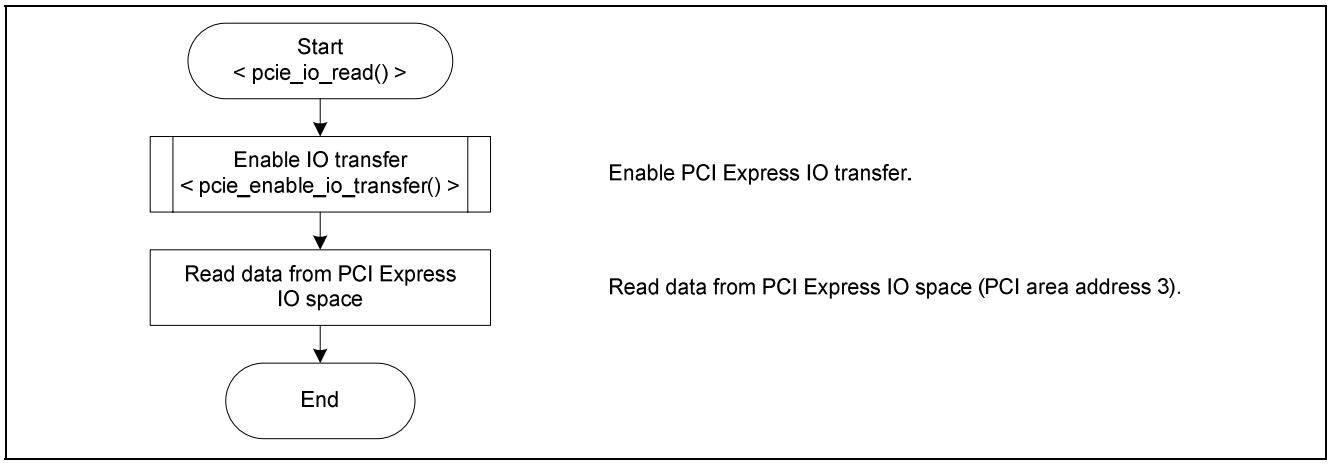

Figure 4.15 Flowchart of PCI Express IO Transfer (Read)

(14) Flowchart of PCI Express IO Transfer Enable Settings

This flowchart shows the processing sequence for making PCI Express IO transfer enable settings.

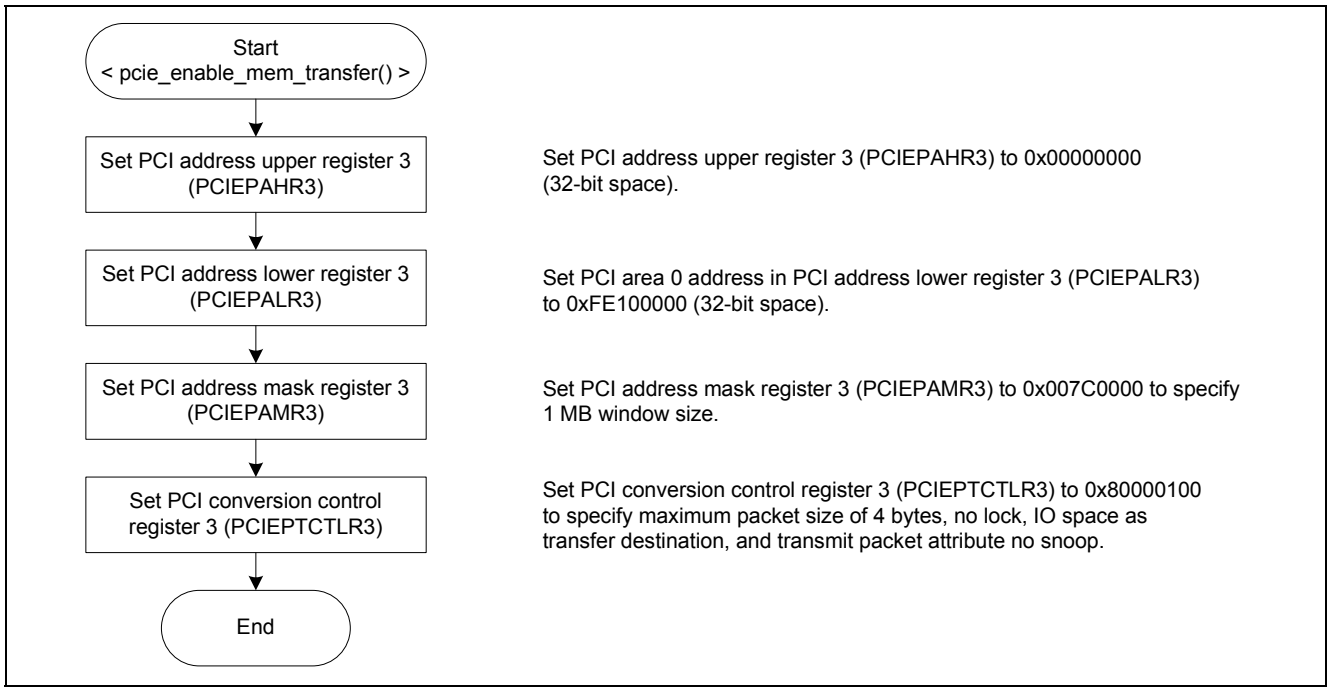

Figure 4.16 Flowchart of PCI Express IO Transfer Enable Settings

(15) Flowchart of PCI Express Physical Layer Control Register Write

This flowchart shows the processing sequence for writing to the PCI Express physical layer control register.

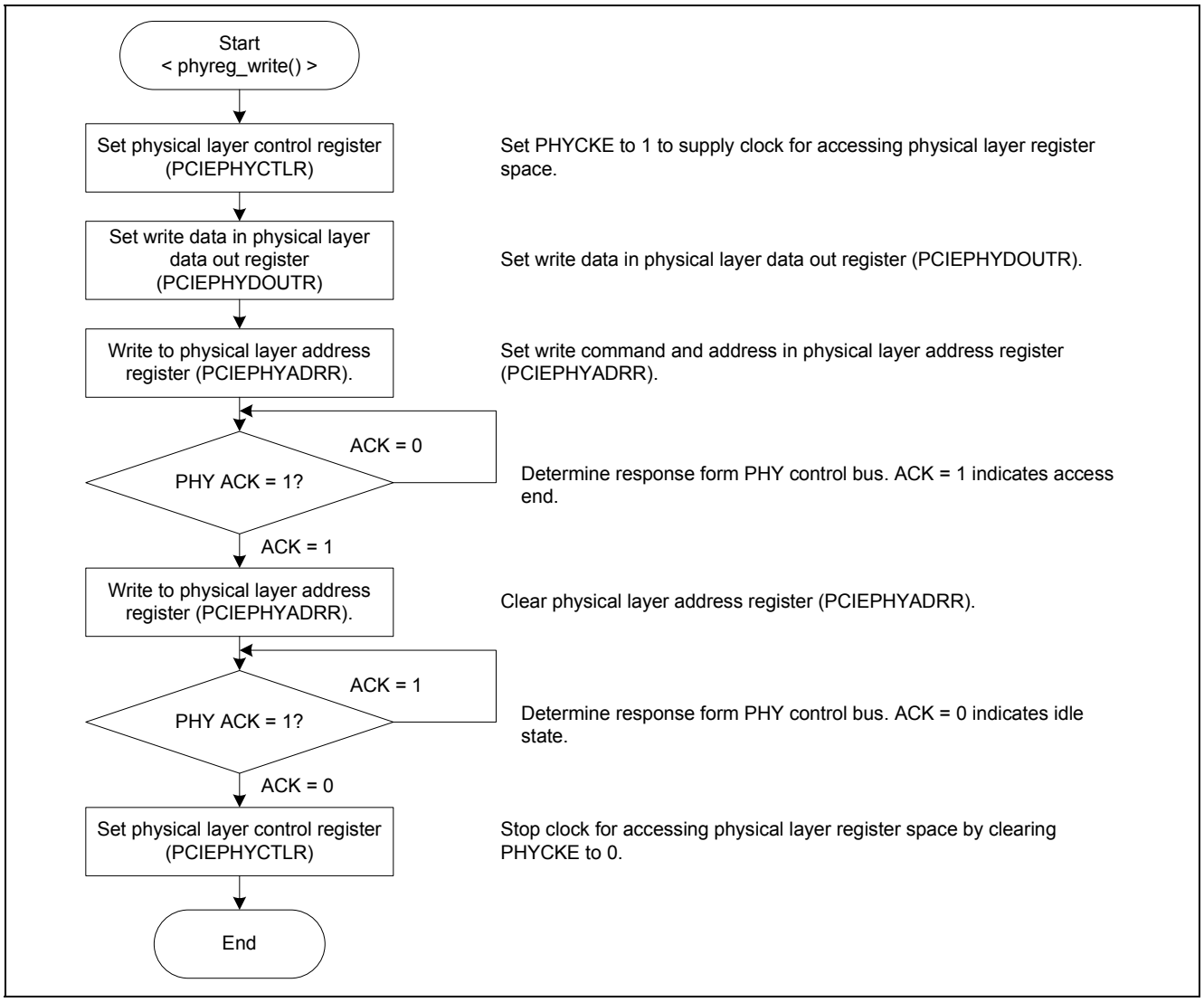

Figure 4.17 Flowchart of PCI Express Physical Layer Control Register Write

(16) Flowchart of PCI Express Configuration Register Read

This flowchart shows the processing sequence for reading the PCI Express configuration register.

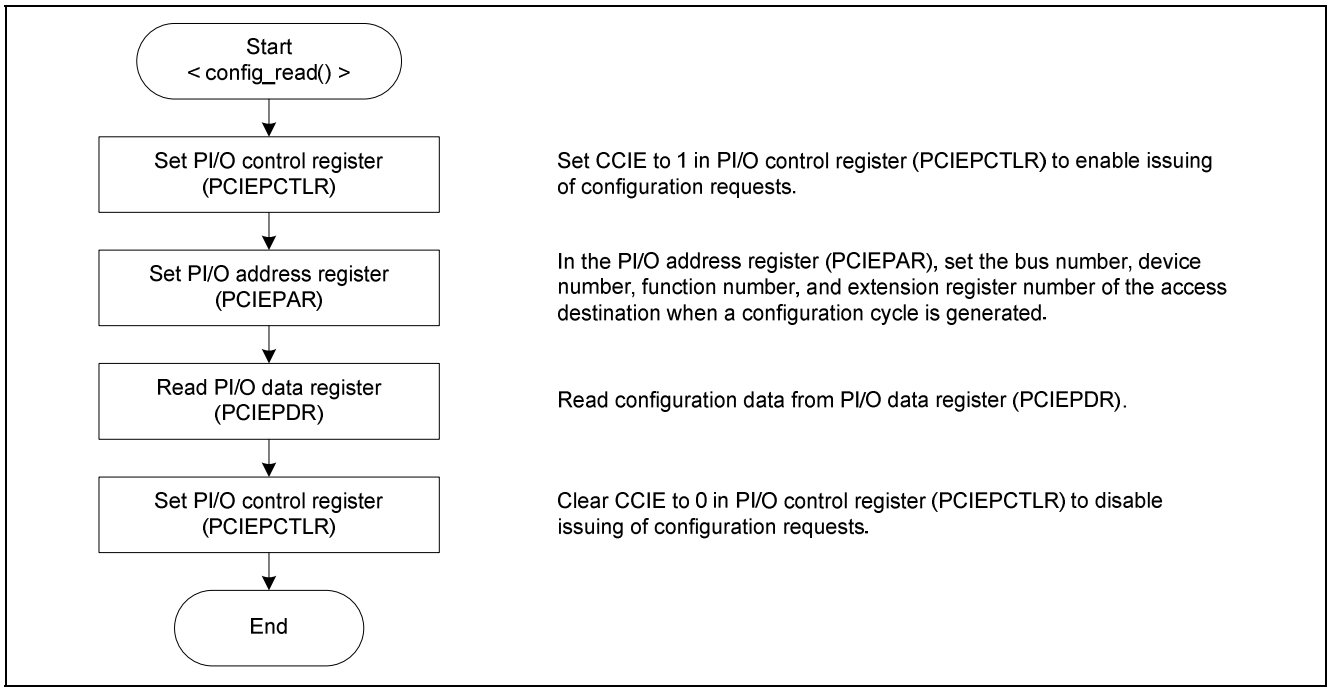

Figure 4.18 Flowchart of PCI Express Configuration Register Read

(17) Flowchart of PCI Express Configuration Register Write

This flowchart shows the processing sequence for writing to the PCI Express configuration register.

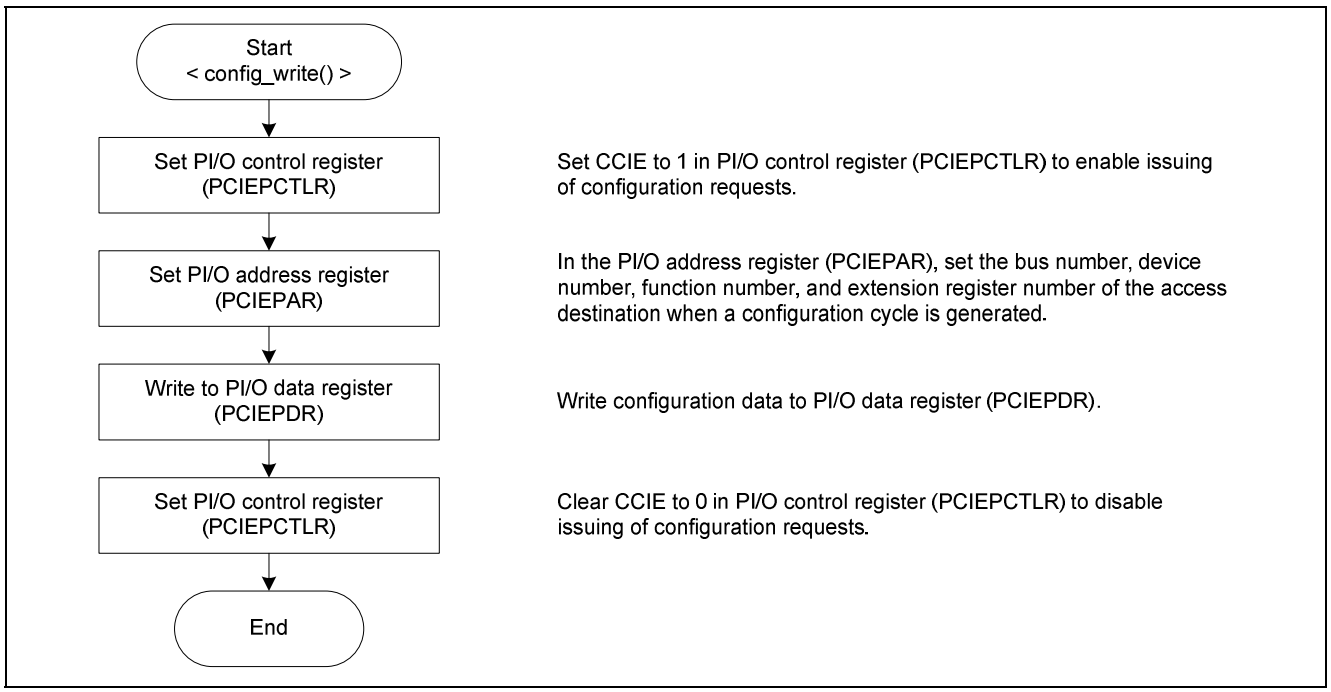

Figure 4.19 Flowchart of PCI Express Configuration Register Write

(18) Flowchart of PCIEC DMA Transfer

This flowchart shows the processing sequence for performing a PCIEC internal DMAC transfer.

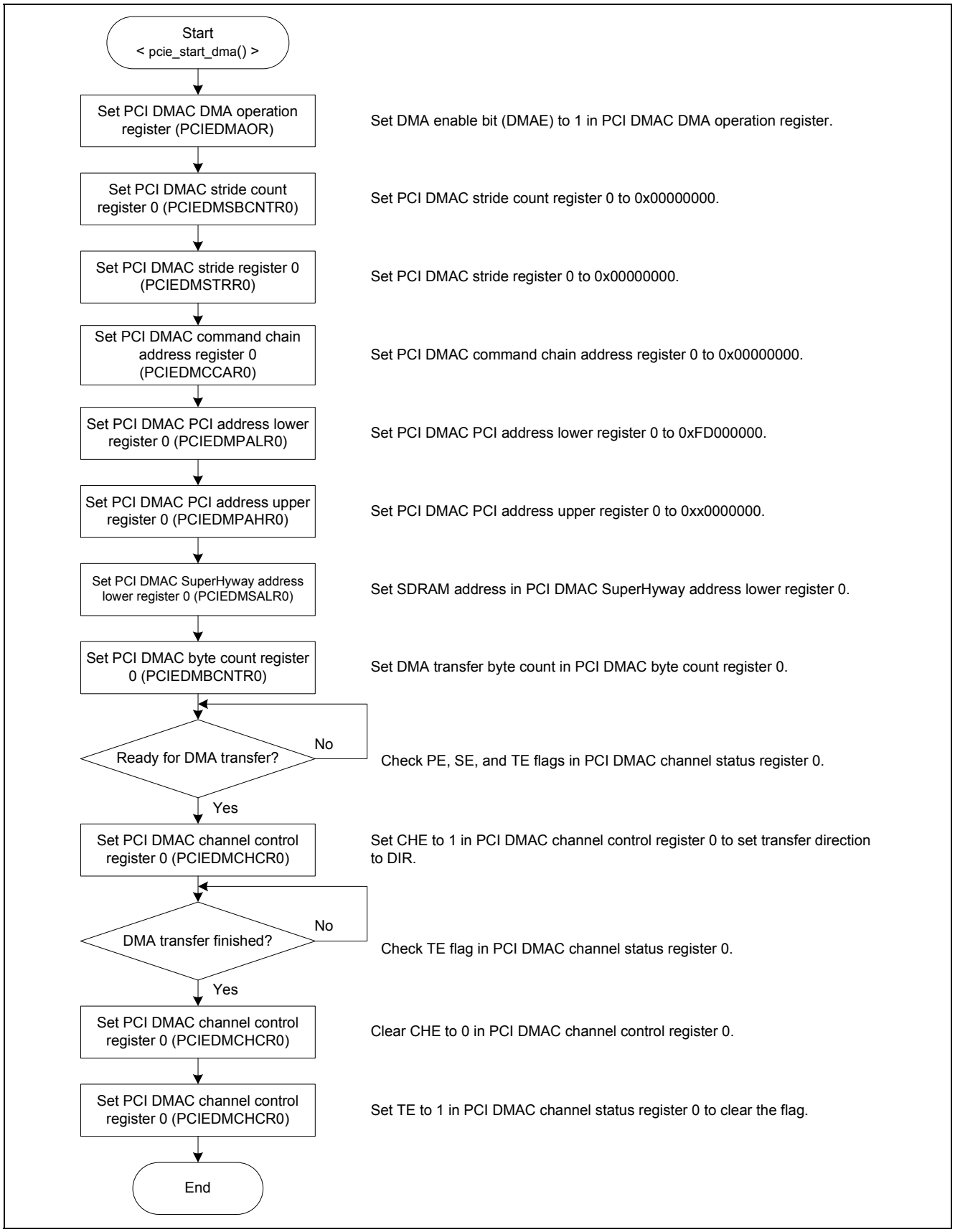

Figure 4.20 Flowchart of PCIEC Internal DMAC Transfer

4.2.4 Sample Program

(1) PCIe_DemoSample.c

This is a program listing of the main function.

```
001 /***********************************************************************/
002 /*                                                                     */
003 /*  FILE        :PCIe_DemoSample.c                                     */
004 /*  DATE        :Wed, Nov 17, 2010                                     */
005 /*  DESCRIPTION :Main Program                                          */
006 /*  CPU TYPE    :Other                                                 */
007 /*                                                                     */
008 /*  This file is generated by Renesas Project Generator (Ver.4.16).    */
009 /*                                                                     */
010 /***********************************************************************/
011
012
013
014 //#include "typedefine.h"
015 #include "config.h"
016 #include "pcie.h"
017
018 #ifdef __cplusplus
019 //#include <ios>                        // Remove the comment when you use ios
020 //_SINT ios_base::Init::init_cnt;       // Remove the comment when you use ios
021 #endif
022
023 void main(void);
024 #ifdef __cplusplus
025 extern "C" {
026 void abort(void);
027 }
028 #endif
029
030 /* ==== Variable declaration ==== */
031 //extern static PCIE_CONF_DATA conf_data;
032 #define  BUFF_MAX  7
033 #define TransByte  64
034
035 /* ==== Function declaration ==== */
036 volatile void BuffClear(char *pBuff, int size);
037 volatile void SdramDataInit(int cnt);
038
039
040 void pfc_init(void);
041
042 /*""FUNC COMMENT""*****************************************************
043 * ID                :
044 * Outline           : Sample program main
045 *                   :
046 * Include           :
047 * Declaration       : void main(void)
048 * Description       : Main program
049 *                   :
050 *                   :
051 *                   :
052 *                   :
053 *                   :
```

```
054 * Limitation        :
055 *                   :
056 * Argument          : none
057 * Return Value      : none
058 * Calling Functions :
059 *""FUNC COMMENT END""***************************************************/
060 void main(void)
061 {
062    volatile static int ret = 0;
063    int i, j;
064    char KeyBuff[BUFF_MAX];
065    unsigned long data;
066    int startadd;
067
068    pfc_init();
069    ret = scif_init();
070
071 #ifdef CONFIG_PCIE_ROOT
072    if( ret == 0 )
073       printf("\n\r_/_/_/ SH7786 PCI Express DEMO Sample ROOT Port _/_/_/\n\r");
074    BuffClear( KeyBuff, BUFF_MAX );             // Buffer clear
075    printf("Target Device Check? Y/N\n\r");
076    while( scif_recive_data( KeyBuff ) != 0);
077    switch(KeyBuff[0]) {
078       case 'Y' :
079                pcie_init(CONFIG_PCIE_ROOT);
080                pcie_check(CONFIG_PCIE_ROOT);
081                break;
082       case 'N' :
083                printf("Not Check Device\n\r");
084                break;
085       default :
086                break;
087    }
088    delay(1000);
089    BuffClear( KeyBuff, BUFF_MAX );             // Buffer clear
090    printf("Transmit Data Start? Y/N\n\r");
091    while( scif_recive_data( KeyBuff ) != 0);
092    switch(KeyBuff[0]) {
093       case 'Y' :
094          printf("Transmit Start\n\r");
095          for(i=0;i<4;i++) {
096             pcie_io_write(CONFIG_PCIE_ROOT, i*4, i+1, Long);
097             data = pcie_io_read(CONFIG_PCIE_ROOT, i*4, Long);
098             printf("Addr = %08x, Data = %08x\n\r", 0xFE100000 + (i*4), data);
099             pcie_mem_write(CONFIG_PCIE_ROOT, i*4, i+2, Long);
100             data = pcie_mem_read(CONFIG_PCIE_ROOT, i*4, Long);
101             printf("Addr = %08x, Data = %08x\n\r", 0xFD000000 + (i*4), data);
102          }
103          /* DMA Transfer Test */
104          SdramDataInit(TransByte);     /* 64Byte data set */
105          printf("\r\nDMA Start ");
106
107          printf("\n\rWRTITE(SuperHyway->PCI)\n");
108          ret = pcie_start_dma(CONFIG_PCIE_ROOT, PCIE_AREA_ADDR, sdram_data_area, PCIE_WRITE, TransByte);
109          if(!ret) {
110             printf("\n\rPCIE DMA Error\n");
```

```
111             break;
112          }
113
114          startadd = 0xA0000000 | sdram_data_area;
115          printf("\r\nTransfer Data\n\r");
116          for(i=0;i<(TransByte/16);i++) {
117             for(j=0;j<4;j++) {
118                printf("%08x ", (*(unsigned long *)startadd));
119                startadd += 4;
120                }
121             printf("\n\r");
122          }
123
124
125          printf("\r\nREAD(PCI->SuperHyway)\n");
126          ret = pcie_start_dma(CONFIG_PCIE_ROOT, PCIE_AREA_ADDR, sdram_data_area + 0x1000, PCIE_READ, TransByte);
127          if(!ret) {
128             printf("\n\rPCIE DMA Error\n");
129             break;
130          }
131          startadd = 0xA0001000 | sdram_data_area;
132          printf("\r\nTransfer Data\n\r");
133          for(i=0;i<(TransByte/16);i++) {
134             for(j=0;j<4;j++) {
135                printf("%08x ", (*(unsigned long *)startadd));
136                startadd += 4;
137             }
138          printf("\n\r");
139          }
140
141          break;
142       case 'N' :
143          printf("Transmit Not Start\n\r");
144          break;
145       default :
146          break;
147    }
148
149 #else
150    if( ret == 0 )
151       printf("\n\r_/_/_/ SH7786 PCI Express DEMO Sample END Point _/_/_/\n\r");
152    pcie_init(CONFIG_PCIE_END);
153
154 #endif
155    printf("PCI Express Demo Sample End\n\r");
156
157 }
158
159 /*""FUNC COMMENT""*****************************************************
160 * ID                : 
161 * Outline           : Sample program main
162 *                   : (PCI Express)
163 * Include           : 
164 * Declaration       : void pfc_init( void )
165 * Description       : A set of a pin function
166 *                   : 
167 *                   :
```

```
168 *                   :
169 *                   :
170 *                   :
171 * Limitation        :
172 *                   :
173 * Argument          : none
174 * Return Value      : none
175 * Calling Functions :
176 *""FUNC COMMENT END""**************************************************/
177 void pfc_init(void)
178 {
179    /* SCIF0 */
180    GPIOR.PHCR.WORD = 0xFC30;
181 }
182
183 /*""FUNC COMMENT""**************************************************
184 * ID                :
185 * Outline           : Sample program main
186 *                   : (PCI Express)
187 * Include           :
188 * Declaration       : void BuffClear(char *pBuff, int size)
189 * Description       : A initialization of the buffer for serial receive datas
190 *                   :
191 *                   :
192 *                   :
193 *                   :
194 *                   :
195 * Limitation        :
196 *                   :
197 * Argument          : *pBuff:Buffer, size:Buffer size
198 * Return Value      : none
199 * Calling Functions :
200 *""FUNC COMMENT END""**************************************************/
201 volatile void BuffClear(char *pBuff, int size)
202 {
203    int i;
204    for( i = 0; i < size; i++ )   /* A clear of a serial data receiving workpiece*/
205    {
206       *( pBuff + i ) = 0;
207    }
208 }
209
210 /*""FUNC COMMENT""**************************************************
211 * ID                :
212 * Outline           : Sample program main
213 *                   : (PCI Express)
214 * Include           :
215 * Declaration       : volatile void SdramDataInit(int cnt)
216 * Description       : A initialization of the data for sdram
217 *                   :
218 *                   :
219 *                   :
220 *                   :
221 *                   :
222 * Limitation        :
223 *                   :
224 * Argument          : none
```

```
225 * Return Value      : none
226 * Calling Functions :
227 *""FUNC COMMENT END""***************************************************/
228 volatile void SdramDataInit(int cnt)
229 {
230    int i;
231    int sdram_add = 0xA0000000 | sdram_data_area;
232    for( i = 0; i < cnt/4; i++ )  /* SDRAM Data Initialize cnt/4 byte  */
233    {
234       (*(unsigned long *)sdram_add) = i;
235       sdram_add += 4;
236    }
237 }
238
239
240 #ifdef __cplusplus
241 void abort(void)
242 {
243
244 }
245 #endif
```

(2) config.h

This is a header file used by the main function.

```
01 #ifndef _CONFIG_H_
02 #define _CONFIG_H_
03
04 #include <stdarg.h>
05 #include <stdio.h>
06 #include <stdlib.h>
07 #include "iodefine.h"
08
09 #defineLong   4
10 #defineWord   2
11 #defineByte   1
12
13 /* SCIF */
14 #define    CONFIG_PCLK   44400000
15 #define    CONFIG_SCIF0
16 //#define  CONFIG_SCIF1
17 //#define  CONFIG_SCIF2
18 //#define  CONFIG_SCIF3
19 //#define  CONFIG_SCIF4
20 //#define  CONFIG_SCIF5
21 //#define  CONFIG_SCIF_CLK_EXTERNAL
22 #define    CONFIG_SCIF_CLK_PCLKCONFIG_PCLK
23 #define    CONFIG_BPS115200
24
25
26 //#define CONFIG_PCIE_ROOT 0
27 //#define  CONFIG_PCIE_END  1
28
29 #undef printf
30 #define printf   scif_printf
31
32 /*** SCIF ***/
33 extern void delay( int cnt );
34 extern int scif_init(void);
35 extern char   scif_recive_data( char  *Data );
36 extern char   scif_recive_data_byte( char   *Data );
37 extern void scif_transmit_data( char *Data );
38 extern void scif_transmit_data_byte( char *Data );
39 extern void scif_printf(char* str, ...);
40
41 /*** PCIe ***/
42 extern void pcie_init(int sel);
43 extern void pcie_check(int sel);
44 extern void pcie_enable_mem_transfer(int sel);
45 extern long pcie_mem_write(int sel, unsigned long addr, unsigned long data, unsigned long size);
46 extern unsigned long pcie_mem_read(int sel, unsigned long addr, unsigned long size);
47 extern void pcie_enable_io_transfer(int sel);
48 extern long pcie_io_write(int sel, unsigned long addr, unsigned long data, unsigned long size);
49 extern unsigned long pcie_io_read(int sel, unsigned long addr, unsigned long size);
50
51 #endif /* _CONFIG_H_ */
```

(3) pcie.c

This is a program listing of the PCIEC initialization function, PCI Express control function, and DMAC control function.

```
001 /***********************************************************************
002 * DISCLAIMER
003
004 * This software is supplied by Renesas Electronics  Corporation. and is only
005 * intended for use with Renesas products. No other uses are authorized.
006
007 * This software is owned by Renesas Electronics Corporation. and is protected under
008 * all applicable laws, including copyright laws.
009
010 * THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
011 * REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
012 * INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
013 * PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
014 * DISCLAIMED.
015
016 * TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
017 * ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
018 * FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
019 * FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
020 * AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
021
022 * Renesas reserves the right, without notice, to make changes to this
023 * software and to discontinue the availability of this software.
024 * By using this software, you agree to the additional terms and
025 * conditions found by accessing the following link:
026 * http://www.renesas.com/disclaimer
027 ***********************************************************************/
028 /* Copyright (C) 2010. Renesas Electronics Corporation., All Rights Reserved.*/
029 /*""FILE COMMENT""*********** Technical reference data ****************
030 * System Name  : SH7786 Sample Program
031 * File Name    : scif.c
032 * Abstract     : The example of a set of PCI Express Sample Program
033 * Version      : Ver 1.00
034 * Device       : SH7786
035 * Tool-Chain   : High-performance Embedded Workshop (Version 4.07.00.007)
036 *              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
037 * OS           : None
038 * H/W Platform : SH-4A Board P/N:AP-SH4AD-3A (Manufacturer:ALPHA PROJECT)
039 * Description  : It is an example program of the example of a SH7786 PCI Express set.
040 *              :
041 * Operation    :
042 * Limitation   :
043 *              :
044 ********************************************************************
045 * History      : 01.Sep.2010 Ver. 1.00 First Release
046 *""FILE COMMENT END""*************************************************/
047
048
049 #include "pcie.h"
050
051 static PCIE_CONF_DATA conf_data;
052
053 /*""FUNC COMMENT""*****************************************************
```

```
054 * ID                :
055 * Outline           : Sample Program Main
056 *                   : (PCI Express)
057 * Include           :
058 * Declaration       : void pcie_enable_mem_transfer(int sel)
059 * Description       : Memory transfer significance
060 *                   :
061 *                   :
062 *                   :
063 *                   :
064 *                   :
065 * Limitation        :
066 *                   :
067 * Argument          : none
068 * Return Value      : none
069 * Calling Functions :
070 *""FUNC COMMENT END""************************************************/
071 void pcie_enable_mem_transfer(int sel)
072 {
073    /* A set of a register (A set of a window) */
074    PCIE_REG(sel, PAHR0) = 0x00000000;      /* 32bit of a upper address */
075    PCIE_REG(sel, PALR0) = PCIE_AREA_ADDR;  /* 32 bits of a lower address */
076    PCIE_REG(sel, PAMR0) = 0x007C0000;      /* A window size is specified(8M) */
077    PCIE_REG(sel, PTCTLR0) = 0x80000000;     /* A transmitting packet property is specified */
078 }
079
080 /*""FUNC COMMENT""*****************************************************
081 * ID                :
082 * Outline           : Sample Program Main
083 *                   : (PCI Express)
084 * Include           :
085 * Declaration  : long pcie_mem_write(int sel, unsigned long addr, unsigned long data, unsigned long size)
086 * Description       : It writes in a memory space
087 *                   :
088 *                   :
089 *                   :
090 *                   :
091 *                   :
092 * Limitation        :
093 *                   :
094 * Argument          : none
095 * Return Value      : -1:size error, 0:Normal
096 * Calling Functions :
097 *""FUNC COMMENT END""************************************************/
098 long pcie_mem_write(int sel, unsigned long addr, unsigned long data, unsigned long size)
099 {
100    unsigned long pcie_addr;
101
102    /* Memory transfer significance */
103    pcie_enable_mem_transfer(sel);
104
105    /* The write to a memory space */
106    pcie_addr = PCIE_AREA_ADDR + addr;
107    switch(size){
108       case 1:
109          PCIE_WRITEB(pcie_addr, data);
110          break;
```

```
111       case 2:
112          PCIE_WRITEW(pcie_addr, data);
113          break;
114       case 4:
115          PCIE_WRITEL(pcie_addr, data);
116          break;
117       default:
118          return -1;
119    }
120
121    return 0;
122 }
123
124 /*""FUNC COMMENT""*****************************************************
125 * ID                : 
126 * Outline           : Sample Program Main
127 *                   : (PCI Express)
128 * Include           : 
129 * Declaration : long pcie_endpoint_mem_burst_write(int sel, unsigned long addr, unsigned long *data, unsigned long size)
130 * Description       : It is a burst write to a memory space
131 *                   : 
132 *                   : 
133 *                   : 
134 *                   : 
135 *                   : 
136 * Limitation        : 
137 *                   : 
138 * Argument          : none
139 * Return Value      : -1:size error, 0:Normal
140 * Calling Functions : 
141 *""FUNC COMMENT END""*************************************************/
142 long pcie_mem_burst_write(int sel, unsigned long addr, unsigned long *data, unsigned long size)
143 {
144    unsigned long pcie_addr;
145    int i;
146
147    /* Memory transfer is validated */
148    pcie_enable_mem_transfer(sel);
149
150    /* Validation of a packet joining (MAX 4096 byte) */
151    PCIE_REG(sel, PTCTLR0) = 0x9B000000;
152
153    /* The 16byte write to a memory space */
154    pcie_addr = PCIE_AREA_ADDR + addr;
155
156    for(i = 0; i < (size/4); i++) {
157       PCIE_WRITEL(pcie_addr + i * 4, data[i]);
158    }
159
160    /* A run of a packet joining */
161    PCIE_REG(sel, PCCTLR) = 0x00000001;
162
163    return 0;
164 }
165 /*""FUNC COMMENT""*****************************************************
166 * ID                : 
167 * Outline           : Sample Program Main
```

```
168 *                   : (PCI Express)
169 * Include           :
170 * Declaration     : long pcie_mem_read(unsigned long addr, unsigned long *data, unsigned long size)
171 * Description       : Read of a memory space
172 *                   :
173 *                   :
174 *                   :
175 *                   :
176 *                   :
177 * Limitation        :
178 *                   :
179 * Argument          : none
180 * Return Value      : -1:size error, 0:Normal
181 * Calling Functions :
182 *""FUNC COMMENT END""***************************************************/
183 unsigned long pcie_mem_read(int sel, unsigned long addr, unsigned long size)
184 {
185    unsigned long pcie_addr;
186    unsigned short wdata;
187    unsigned char bdata;
188    unsigned long data;
189
190    /* Memory transfer is validated */
191    pcie_enable_mem_transfer(sel);
192
193    /* From a memory space to read */
194    pcie_addr = PCIE_AREA_ADDR + addr;
195    switch(size){
196       case 1:
197          bdata = PCIE_READB(pcie_addr);
198          data = (unsigned long) bdata;
199          break;
200       case 2:
201          wdata = PCIE_READW(pcie_addr);
202          data = (unsigned long) wdata;
203          break;
204       case 4:
205          data = PCIE_READL(pcie_addr);
206          break;
207       default:
208          data = 0;
209          break;;
210    }
211    return data;
212 }
213
214
215 /*""FUNC COMMENT""*****************************************************
216 * ID                :
217 * Outline           : Sample Program Main
218 *                   : (PCI Express)
219 * Include           :
220 * Declaration       : void pcie_enable_io_transfer(int sel)
221 * Description       : An I/O transmission is validated
222 *                   :
223 *                   :
224 *                   :
```

```
225 *                   :
226 *                   :
227 * Limitation        :
228 *                   :
229 * Argument          : none
230 * Return Value      : none
231 * Calling Functions :
232 *""FUNC COMMENT END""*************************************************/
233 void pcie_enable_io_transfer(int sel)
234 {
235    /* A set of a register (A set of a window) */
236    PCIE_REG(sel, PAHR3) = 0x00000000;          /* 32 bits of a upper address */
237    PCIE_REG(sel, PALR3) = PCIE_AREA_IO_ADDR;   /* 32 bits of a lower address */
238    PCIE_REG(sel, PAMR3) = 0x000C0000;          /* A window size is specified(1M) */
239    PCIE_REG(sel, PTCTLR3) = 0x80000100;     /* A transmitting packet property is specified */
240 }
241
242 /*""FUNC COMMENT""*****************************************************
243 * ID                :
244 * Outline           : Sample Program Main
245 *                   : (PCI Express)
246 * Include           :
247 * Declaration     : long pcie_io_write(unsigned long addr, unsigned long data, unsigned long size)
248 * Description       : It writes in an I/O field
249 *                   :
250 *                   :
251 *                   :
252 *                   :
253 *                   :
254 * Limitation        :
255 *                   :
256 * Argument          : none
257 * Return Value      : -1:size error, 0:Normal
258 * Calling Functions :
259 *""FUNC COMMENT END""*************************************************/
260 long pcie_io_write(int sel, unsigned long addr, unsigned long data, unsigned long size)
261 {
262    unsigned long pcie_addr;
263
264    /* Validation of an I/O transmission */
265    pcie_enable_io_transfer(sel);
266
267    /* The write to an I/O field */
268    pcie_addr = PCIE_AREA_IO_ADDR + addr;
269    switch(size){
270       case 1:
271          PCIE_WRITEB(pcie_addr, data);
272          break;
273       case 2:
274          PCIE_WRITEW(pcie_addr, data);
275          break;
276       case 4:
277          PCIE_WRITEL(pcie_addr, data);
278          break;
279       default:
280          return -1;
281    }
```

```
282
283    return 0;
284 }
285
286 /*""FUNC COMMENT""*****************************************************
287 * ID                : 
288 * Outline           : Sample Program Main
289 *                   : (PCI Express)
290 * Include           : 
291 * Declaration    : long pcie_io_read(int sel, unsigned long addr, unsigned long *data, unsigned long size)
292 * Description       : Read of an input-output field
293 *                   : 
294 *                   : 
295 *                   : 
296 *                   : 
297 *                   : 
298 * Limitation        : 
299 *                   : 
300 * Argument          : none
301 * Return Value      : -1:size error, 0:Normal
302 * Calling Functions : 
303 *""FUNC COMMENT END""************************************************/
304 unsigned long pcie_io_read(int sel, unsigned long addr, unsigned long size)
305 {
306    unsigned long pcie_addr;
307    unsigned short wdata;
308    unsigned char bdata;
309    unsigned long data;
310
311    /* Validation of an I/O transmission */
312    pcie_enable_io_transfer(sel);
313
314    /* From an I/O field to read */
315    pcie_addr = PCIE_AREA_IO_ADDR + addr;
316    switch(size){
317       case 1:
318          bdata = PCIE_READB(pcie_addr);
319          data = (unsigned long) bdata;
320          break;
321       case 2:
322          wdata = PCIE_READW(pcie_addr);
323          data = (unsigned long) wdata;
324          break;
325       case 4:
326          data = PCIE_READL(pcie_addr);
327          break;
328       default:
329          data = 0;
330          break;
331    }
332
333    return data;
334
335 }
336
337
338 /*""FUNC COMMENT""*****************************************************
```

```
339 * ID                :
340 * Outline           : Sample Program Main
341 *                    : (PCI Express)
342 * Include           :
343 * Declaration     : static int phyreg_write(int sel, int addr, int lane, unsigned long data)
344 * Description       : The write to a physical-layer control register
345 *                    :
346 *                    :
347 *                    :
348 *                    :
349 *                    :
350 * Limitation        :
351 *                    :
352 * Argument          : none
353 * Return Value      : 0
354 * Calling Functions :
355 *""FUNC COMMENT END""**************************************************/
356 static int phyreg_write(int sel, int addr, int lane, unsigned long data)
357 {
358    unsigned long wdata;
359
360    /*The write to a physical-layer control register */
361    wdata = 0x00010000 + ((lane & 0xf) << 8) + (addr & 0xff);
362    PCIE_REG(sel, PHYCTLR) = 0x00000001; /* clock enable */
363    PCIE_REG(sel, PHYDOUTR) = data;      /* A set of a writed data */
364    PCIE_REG(sel, PHYADRR) = wdata;      /* A set of a command/address */
365    while( (PCIE_REG(sel, PHYADRR) & 0x01000000) == 0 );
366
367    /* Waiting for ACK */
368    PCIE_REG(sel, PHYADRR) = 0x00000000; /* Command clear */
369    while( (PCIE_REG(sel, PHYADRR) & 0x01000000) != 0 );
370
371    PCIE_REG(sel, PHYCTLR) = 0x00000000; /* Clock disabling */
372
373    return 0;
374 }
375
376 /*""FUNC COMMENT""*****************************************************
377 * ID                :
378 * Outline           : Sample Program Main
379 *                    : (PCI Express)
380 * Include           :
381 * Declaration     : static int phyreg_read(int sel, int addr, int lane, unsigned long *data)
382 * Description       : Read of a physical-layer control register
383 *                    :
384 *                    :
385 *                    :
386 *                    :
387 *                    :
388 * Limitation        :
389 *                    :
390 * Argument          : none
391 * Return Value      : 0
392 * Calling Functions :
393 *""FUNC COMMENT END""**************************************************/
394 static int phyreg_read(int sel, int addr, int lane, unsigned long *data)
395 {
```

```
396    unsigned long wdata;
397
398    /*Read of a physical-layer control register */
399    wdata = 0x00020000 + ((lane & 0xf) << 8) + (addr & 0xff);
400    PCIE_REG(sel, PHYCTLR) = 0x00000001; /* clock enable */
401    PCIE_REG(sel, PHYADRR) = wdata;      /* A set of a command/address */
402    while( (PCIE_REG(sel, PHYADRR) & 0x01000000) == 0 );
403
404    /* Waiting for ACK */
405    *data = PCIE_REG(sel, PHYDINR);      /* Read data */
406    PCIE_REG(sel, PHYADRR) = 0x00000000; /* Command clear */
407    while( (PCIE_REG(sel, PHYADRR) & 0x01000000) != 0 );
408
409    PCIE_REG(sel, PHYCTLR) = 0x00000000; /* Clock disabling */
410
411    return 0;
412 }
413
414 /*""FUNC COMMENT""*****************************************************
415 * ID                 :
416 * Outline            : Sample Program Main
417 *                    : (PCI Express)
418 * Include            :
419 * Declaration: static int config_read(int sel, int bus, int dev, int func, int regno, unsigned long *data)
420 * Description        : Read of a configuration register
421 *                    :
422 *                    :
423 *                    :
424 *                    :
425 *                    :
426 * Limitation         :
427 *                    :
428 * Argument           : none
429 * Return Value       : 0
430 * Calling Functions :
431 *""FUNC COMMENT END""*************************************************/
432 static int config_read(int sel, int bus, int dev, int func, int regno, unsigned long *data)
433 {
434    unsigned long wdata;
435
436    PCIE_REG(sel, PCTLR) = 0x80000000;/* An issue Clearance of a configuration request */
437    wdata = (bus << 24) + (dev << 19) + (func << 16) + regno;
438    PCIE_REG(sel, PAR) = wdata;
439
440    /* Set of addr, bus/dev/func */
441    *data = PCIE_REG(sel, PDR);          /* Issue of configuration read */
442    PCIE_REG(sel, PCTLR) = 0x00000000;/* Issue of a configuration request is forbidden */
443
444    return 0;
445 }
446
447 /*""FUNC COMMENT""*****************************************************
448 * ID                 :
449 * Outline            : Sample Program Main
450 *                    : (PCI Express)
451 * Include            :
452 * Declaration: static int config_write(int sel, int bus, int dev, int func, int regno, unsigned long data)
```

```
453 * Description       : Write of a configuration register
454 *                   :
455 *                   :
456 *                   :
457 *                   :
458 *                   :
459 * Limitation        :
460 *                   :
461 * Argument          : none
462 * Return Value      : 0
463 * Calling Functions :
464 *""FUNC COMMENT END""*************************************************/
465 static int config_write(int sel, int bus, int dev, int func, int regno, unsigned long data)
466 {
467    unsigned long wdata;
468
469    PCIE_REG(sel, PCTLR) = 0x80000000;    /* An issue Clearance of a configuration request */
470    wdata = (bus << 24) + (dev << 19) + (func << 16) + regno;
471    PCIE_REG(sel, PAR) = wdata;
472
473    /* Set of addr, bus/dev/func */
474    PCIE_REG(sel, PDR) = data;        /* Issue of configuration read */
475    PCIE_REG(sel, PCTLR) = 0x00000000;    /* Issue of a configuration request is forbidden */
476
477    return 0;
478 }
479
480 /*""FUNC COMMENT""*****************************************************
481 * ID                :
482 * Outline           : Sample Program Main
483 *                   : (PCI Express)
484 * Include           :
485 * Declaration       : static void pcie_soft_reset(int sel)
486 * Description       : Software reset of a PCIE controller
487 *                   :
488 *                   :
489 *                   :
490 *                   :
491 *                   :
492 * Limitation        :
493 *                   :
494 * Argument          : none
495 * Return Value      : none
496 * Calling Functions :
497 *""FUNC COMMENT END""*************************************************/
498 static void pcie_soft_reset(int sel)
499 {
500    /* A run of software reset */
501    PCIE_REG(sel, SRSTR) = 0x00000001;
502
503    /* A reset of a PCIE internal register */
504    PCIE_REG(sel, TCTLR) = 0x00000000;
505
506    /* A reset of software reset */
507    PCIE_REG(sel, SRSTR) = 0x00000000;
508
509    /* A clear of a VCO transmitter buffer */
```

```
510    PCIE_REG(sel, TXVC0SR) = 0x80000000;
511 }
512
513 /*""FUNC COMMENT""*****************************************************
514 * ID                :
515 * Outline           : Sample Program Main
516 *                   : (PCI Express)
517 * Include           :
518 * Declaration       : static int pcie_phy_init(int sel)
519 * Description       : The initialization of a PCIE controller transfer control register
520 *                   :
521 *                   :
522 *                   :
523 *                   :
524 *                   :
525 * Limitation        :
526 *                   :
527 * Argument          : none
528 * Return Value      : -1:Time out, 0:Normal
529 * Calling Functions :
530 *""FUNC COMMENT END""*************************************************/
531 static int pcie_phy_init(int sel)
532 {
533    unsigned long stime;
534    static unsigned long data;
535
536    printf("PCI Express PHY During Initialization...");
537    /* A clock supply of a physical-layer register space accessing */
538    PCIE_REG(sel, PHYCTLR) = 0x00000001;
539
540    /* A physical layer's initialization */
541    phyreg_write(sel, 0x60, 0xf, 0x004B008B);
542    phyreg_write(sel, 0x61, 0xf, 0x00007B41);
543    phyreg_write(sel, 0x64, 0xf, 0x00FF4F00);
544    phyreg_write(sel, 0x65, 0xf, 0x09070907);
545    phyreg_write(sel, 0x66, 0xf, 0x00000010);
546    phyreg_write(sel, 0x74, 0xf, 0x0007001C);
547    phyreg_write(sel, 0x79, 0xf, 0x01FC000D);
548    phyreg_write(sel, 0xB0, 0xf, 0x00000610);
549
550    /* A boot of a physical layer */
551    phyreg_write(sel, 0x67, 0x1, 0x00000400);
552
553    if(sel)
554       phyreg_read(sel, 0x67, 0x1, &data);
555
556    /*The clock of a physical-layer register space accessing is suspended */
557    PCIE_REG(sel, PHYCTLR) = 0x00000000;
558
559    /* Waiting for set a physical module */
560    stime = 1000;
561    while(stime--) {
562       /* It waits until a physical module will be in a ready state */
563       if( (PCIE_REG(sel, PHYSR) & 0x00000001) != 0 ) {
564          break;
565       }
566       delay(1000);
```

```
567    }
568    if(!stime)
569       return -1;
570
571    printf("      Finish\n\r");
572
573    return 0;
574 }
575
576 /*""FUNC COMMENT""*****************************************************
577 * ID                :
578 * Outline           : Sample Program Main
579 *                   : (PCI Express)
580 * Include           :
581 * Declaration       : static int pcie_trans_cont_init(int sel)
582 * Description       : The initialization of a PCIE controller transfer control register
583 *                   :
584 *                   :
585 *                   :
586 *                   :
587 *                   :
588 * Limitation        :
589 *                   :
590 * Argument          : none
591 * Return Value      : -1:Time out, 0:Normal
592 * Calling Functions :
593 *""FUNC COMMENT END""*************************************************/
594 static int pcie_trans_cont_init(int sel)
595 {
596    unsigned long stime;
597
598    printf("PCI Express Controller During Initialization...");
599
600    /* A set of a bridge facility */
601    PCIE_REG(sel, LAR0) = 0x0C000000; /* A local address 0 is specified */
602    PCIE_REG(sel, LAR2) = 0x0D000000;    /* A local address 1 is specified */
603
604    PCIE_REG(sel, LAMR0) = 0x000FFF01;   /* Local(SHwy)space register 0 */
605                      /* 1MB */
606                      /* A memory is secured in 32 bit address space */
607                      /* Local address enabling is specified */
608    PCIE_REG(sel, LAMR2) = 0x000FFF11;   /* Local(SHwy)space register 1 */
609                      /* 1MB */
610                      /* A memory is secured in 32 bit address space */
611                      /* Local address enabling is specified */
612
613    /* A set of a configuration register */
614    if(sel) {
615       PCIE_REG(sel, IDSETR1) = 0x01234567;
616       PCIE_REG(sel, IDSETR2) = 0x89ABCDEF;
617    }
618
619    PCIE_REG(sel, PCICONF1) = 0x00000007;
620
621    /* A wake-up of LTSSM(A settlement of a connection is started) */
622    PCIE_REG(sel, TCTLR) = 0x00000001;
623
```

```
624    /* A connection's settlement waiting */
625    stime = 10000;
626    while(stime--) {
627        /* Data Link Layer Active check */
628        if(PCIE_REG(sel, TSTR) != 0) {
629            break;
630        }
631        delay(1000);
632    }
633    if(!stime)
634        return -1;
635    printf("     Finish\n\r");
636    return 0;
637 }
638
639 /*""FUNC COMMENT""*****************************************************
640 * ID                : 
641 * Outline           : Sample Program Main
642 *                   : (PCI Express)
643 * Include           : 
644 * Declaration       : static int pcie_config_init(void)
645 * Description       : A set of the configuration of a PCIE controller
646 *                   : 
647 *                   : 
648 *                   : 
649 *                   : 
650 *                   : 
651 * Limitation        : 
652 *                   : 
653 * Argument          : none
654 * Return Value      : -1:Inaccurate ID, 0:Normal
655 * Calling Functions : 
656 *""FUNC COMMENT END""*************************************************/
657 static int pcie_config_init(int sel)
658 {
659    unsigned long config_data[(PCIE_MAX_CONFREG_SIZE/4)];
660    unsigned long regno;
661    unsigned long data;
662    unsigned long dev_id;
663    unsigned long dev_cap;
664    unsigned long dev_ctrl;
665    unsigned long dev_bar;
666    unsigned long dev_type;
667    unsigned char cap_ptr, next_ptr, cap_id, mpss1, mpss2, mps;
668    int stime = 1000;
669
670    /* A confirm of a vender and product ID */
671    while(stime--) {
672        config_read(sel, 0, 1, 0, PCIE_CONF_DEVICE_ID, &data);
673        if(!(PCIE_REG(sel, PCTLR) & 0x00010000))
674            break;
675        delay(1000);
676    }
677    if(!stime)
678        return -1;
679
680    dev_id = data;
```

```
681    if(dev_id == 0x00000000) {
682       return -1;
683    }
684
685    /* Max Payload Size Supported of a splicing place device is acquired. */
686    config_read(sel, 0, 1, 0, PCIE_CONF_CAP_PTR, &data);
687    next_ptr = data & 0xff;
688    cap_id = 0xff;
689    while(1) {
690       if (cap_id == 0x10) {
691          /* PCI Express Capability Structure */
692          config_read(sel, 0, 1, 0, (unsigned long)(cap_ptr+0x04), &dev_cap);
693          mpss1 = dev_cap & 0x07;
694          break;
695       } else {
696          /* other capability list */
697          cap_ptr = next_ptr;
698          config_read(sel, 0, 1, 0, (unsigned long)cap_ptr, &data);
699          cap_id = data & 0xff;
700          next_ptr = (data >> 8) & 0xff;
701       }
702    }
703
704    /* MPSS of a self-device */
705    mpss2 = PCIE_REG(sel, EXPCAP1) & 0x07;
706    mps = (mpss1 < mpss2) ? mpss1 : mpss2; /* Both smallest value is taken. */
707
708    /* Set of MPS : Splicing place device */
709    config_read(sel, 0,1,0,(unsigned long)(cap_ptr + 0x08), &dev_ctrl);
710    dev_ctrl = (dev_ctrl & 0xFFFFFF1F) | (mps << 5); /* bit7-5 is substituted for MPS */
711    config_write(sel, 0,1,0,(unsigned long)(cap_ptr + 0x08), dev_ctrl);
712
713    /* Set of MPS: self-device */
714    dev_ctrl = PCIE_REG(sel, EXPCAP2);
715    dev_ctrl = (dev_ctrl & 0xFFFFFF1F) | (mps << 5); /* bit7-5 is substituted for MPS */
716    PCIE_REG(sel, EXPCAP1) = dev_ctrl;
717
718    /* A set of a command and a register */
719    config_write(sel, 0, 1, 0, PCIE_CONF_COMMAND, 0x00000007);    /* External device */
720
721    /* Set of BAR: Splicing place device */
722    config_read(sel, 0, 1, 0, PCIE_CONF_BASE_ADDRESS_0, &dev_bar);
723    dev_type = dev_bar & 0x06;
724    if(dev_type == 0x00) {
725       /* 32 bit space */
726       config_write(sel, 0, 1, 0, PCIE_CONF_BASE_ADDRESS_0, PCIE_AREA_ADDR);
727       config_write(sel, 0, 1, 0, PCIE_CONF_BASE_ADDRESS_2, PCIE_AREA_IO_ADDR);
728    } else if(dev_type == 0x04) {
729       /* 64 bit space */
730       config_write(sel, 0, 1, 0, PCIE_CONF_BASE_ADDRESS_0, PCIE_AREA_ADDR);
731       config_write(sel, 0, 1, 0, PCIE_CONF_BASE_ADDRESS_1, 0x00000000);
732    }
733
734    /* An obtaining of all the configuration registers of an external device */
735    for (regno = 0; regno < (PCIE_MAX_CONFREG_SIZE/4); regno++) {
736       config_read(sel, 0, 1, 0, regno * 4, &data);
737       config_data[regno] = data;
```

```
738    }
739    memcpy( &conf_data, config_data, sizeof( PCIE_CONF_DATA ) );
740
741    return 0;
742 }
743
744 /*""FUNC COMMENT""*****************************************************
745 * ID                :
746 * Outline           : Sample Program Main
747 *                   : (PCI Express)
748 * Include           :
749 * Declaration       : static unsigned long pcie_link_lane(void)
750 * Description       : An obtaining of an effective lane
751 *                   :
752 *                   :
753 *                   :
754 *                   :
755 *                   :
756 * Limitation        :
757 *                   :
758 * Argument          : none
759 * Return Value      : An obtaining of a link data
760 * Calling Functions :
761 *""FUNC COMMENT END""************************************************/
762 static unsigned long pcie_link_lane(int sel)
763 {
764    unsigned long data;
765
766    /* The read of a link status */
767    data = PCIE_REG(sel, EXPCAP4);
768    data >>= 20;
769    data &= 0x3f;
770
771    return data;
772 }
773
774 /*""FUNC COMMENT""*****************************************************
775 * ID                :
776 * Outline           : Sample Program Main
777 *                   : (PCI Express)
778 * Include           :
779 * Declaration       : void pcie_init(int sel)
780 * Description       : The initialization of a PCIE controller
781 *                   :
782 *                   :
783 *                   :
784 *                   :
785 *                   :
786 * Limitation        :
787 *                   :
788 * Argument          : none
789 * Return Value      : none
790 * Calling Functions :
791 *""FUNC COMMENT END""************************************************/
792 void pcie_init(int sel)
793 {
794    /* The initialization of a PCIE configuration data */
```

```
795    memset( &conf_data, 0xFF, sizeof( PCIE_CONF_DATA ) );
796
797    /* Software reset */
798    pcie_soft_reset(sel);
799
800    /* A physical layer's initialization */
801    if( pcie_phy_init(sel) < 0 ) {
802       return;
803    }
804
805    /* The initialization of PCIE (A connection start) */
806    if( pcie_trans_cont_init(sel) < 0 ) {
807       return;
808    }
809
810    /* A set of a configuration */
811    if( pcie_config_init(sel) < 0 ) {
812       return;
813    }
814
815 }
816
817 /*""FUNC COMMENT""*****************************************************
818 * ID                :
819 * Outline           : Sample Program Main
820 *                   : (PCI Express)
821 * Include           :
822 * Declaration       : void pcie_check(int sel)
823 * Description       : The check of a PCIE controller device
824 *                   :
825 *                   :
826 *                   :
827 *                   :
828 *                   :
829 * Limitation        :
830 *                   :
831 * Argument          : none
832 * Return Value      : none
833 * Calling Functions :
834 *""FUNC COMMENT END""*************************************************/
835 void pcie_check(int sel)
836 {
837    unsigned long lane;
838
839    /* A view of a configuration register */
840    if( (conf_data.VenderID != 0xFFFF) && (conf_data.DeviceID != 0xFFFF) ) {
841       scif_printf(" Enable lane : %d LANE\n\r", pcie_link_lane(sel));
842       scif_printf("\n\r");
843
844       scif_printf(" Vender ID   : %04x\n\r", conf_data.VenderID);
845       scif_printf(" Device ID   : %04x\n\r", conf_data.DeviceID);
846       scif_printf(" Command     : %04x\n\r", conf_data.Command );
847       scif_printf(" Status      : %04x\n\r", conf_data.Status  );
848       scif_printf(" Revision ID : %02x\n\r", conf_data.RevisionID);
849       scif_printf(" ProgrammingInterface : %02x\n\r", conf_data.ProgrammingInterface);
850       scif_printf(" SubClass    : %02x\n\r", conf_data.SubClass);
851       scif_printf(" BaseClass   : %02x\n\r", conf_data.BaseClass);
```

```
852       scif_printf(" CachLineSize: %02x\n\r", conf_data.CachLineSize);
853       scif_printf(" LatencyTimer: %02x\n\r", conf_data.LatencyTimer);
854       scif_printf(" HeaderType  : %02x\n\r", conf_data.HeaderType);
855       scif_printf(" BIST        : %02x\n\r", conf_data.BIST);
856       scif_printf(" BaseAdrsREG : %08x\n\r", conf_data.BaseAddressRegisters[0]);
857       scif_printf(" BaseAdrsREG : %08x\n\r", conf_data.BaseAddressRegisters[1]);
858       scif_printf(" BaseAdrsREG : %08x\n\r", conf_data.BaseAddressRegisters[2]);
859       scif_printf(" BaseAdrsREG : %08x\n\r", conf_data.BaseAddressRegisters[3]);
860       scif_printf(" BaseAdrsREG : %08x\n\r", conf_data.BaseAddressRegisters[4]);
861       scif_printf(" BaseAdrsREG : %08x\n\r", conf_data.BaseAddressRegisters[5]);
862       scif_printf(" CardbusCISpointer : %08x\n\r", conf_data.CardbusCISpointer);
863       scif_printf(" SubsystemVenderID : %04x\n\r", conf_data.SubsystemVenderID);
864       scif_printf(" SubsystemID : %04x\n\r", conf_data.SubsystemID);
865       scif_printf(" ExpantionROMbaseAddress : %08x\n\r", conf_data.ExpantionROMbaseAddress);
866       scif_printf(" CAP_PTR     : %02x\n\r", conf_data.CAP_PTR);
867       scif_printf(" Int Line    : %02x\n\r", conf_data.InttreuptLine);
868       scif_printf(" Int Pin     : %02x\n\r", conf_data.InttreuptPin);
869       scif_printf(" Min_Gnt     : %02x\n\r", conf_data.Min_Gnt);
870       scif_printf(" Max_Lat     : %02x\n\r", conf_data.Max_Lat);
871    } else {
872       /* no device */
873       scif_printf("Device not detected on PCI Bus!\r\n");
874    }
875 }
876
877 /*""FUNC COMMENT""*****************************************************
878 * ID                :
879 * Outline           : Sample Program Main
880 *                   : (PCI Express)
881 * Include           :
882 * Declaration       : int pcie_start_dma(int sel, int pciadd, int shadd, int dir, int cnt)
883 * Description       : PCIEC-DMAC Setting and start
884 *                   :
885 *                   :
886 *                   :
887 *                   :
888 *                   :
889 * Limitation        :
890 *                   :
891 * Argument          : none
892 * Return Value      : none
893 * Calling Functions :
894 *""FUNC COMMENT END""*************************************************/
895 int pcie_start_dma(int sel, int pciadd, int shadd, int dir, int cnt)
896 {
897    int stime, status;
898    /* DMA Nomal Transfer */
899    PCIE_REG(sel, DMAOR) = DMAE;            /* DMAC Enable */
900
901    PCIE_REG(sel, DMSBCNTR0) = 0x00000000;  /* No Stride Transfer */
902    PCIE_REG(sel, DMSTRR0) = 0x00000000;
903    PCIE_REG(sel, DMCCAR0) = 0x00000000;    /* No DMAC Command chain Transfer */
904
905    PCIE_REG(sel, DMPALR0) = pciadd;        /* 32bit of a lower address for PCI */
906    PCIE_REG(sel, DMPAHR0) = 0x00000000;    /* 32bit of a upper address for PCI */
907
908    PCIE_REG(sel, DMSALR0) = shadd;         /* 32bit of a address for SuperHyway */
```

```
909
910    PCIE_REG(sel, DMBCNTR0) = cnt;          /* Transfer Count */
911
912    /* DMA Transfer Raedy settlement waiting */
913    stime = 10000;
914    while(stime--) {
915       /* Channel Status PE & SE & TE = 0 */
916       status = PCIE_REG(sel, DMCHSR0);
917       if(((status & PE) != PE) |
918          ((status & SE) != SE) |
919          ((status & TE) != TE)) {
920          break;
921       }
922       delay(1000);
923    }
924    if(!stime)
925       return -1;
926
927    PCIE_REG(sel, DMCHCR0) = CHE | DIR(dir);    /* Transfer Start / Direction */
928
929    while((PCIE_REG(sel, DMCHSR0) & TE) != TE); /* Transfer End Wait */
930    PCIE_REG(sel, DMCHCR0) &= !CHE;             /* Transfer Disable */
931    PCIE_REG(sel, DMCHSR0) = TE;
932
933 }
```

(4) ”pcie.h”

This is a header file used by the main function, PCIEC initialization function, PCI Express control function, and DMAC control function.

```
001 #ifndef  _PCIE_H_
002 #define  _PCIE_H_
003
004 #include "config.h"
005
006 #define  PCIE_BASE 0xFE000000
007 #define  PCIE_REG(p, x)   (*(volatile unsigned long  *)(PCIE_BASE | (p << 21) | x))
008
009 #define ENBLR       0x00008
010 #define ECR         0x0000C
011 #define PAR         0x00010
012 #define PCTLR       0x00018
013 #define PDR         0x00020
014 #define MSGALR      0x00030
015 #define MSGAHR      0x00034
016 #define MSGCTLR     0x00038
017 #define UNLOCKCR    0x00048
018 #define IDR         0x00060
019 #define DBGCTLR     0x00100
020 #define INTXR       0x04000
021 #define RMSGR       0x04010
022 #define RMSGIER     0x04040
023 #define RSTR0       0x08000
024 #define RSTR1       0x08004
025 #define RSTR2       0x08008
026 #define RSTR3       0x0800C
027 #define SRSTR       0x08040
028 #define PHYCTLR     0x10000
029 #define PHYADRR     0x10004
030 #define PHYDINR     0x10008
031 #define PHYDOUTR    0x1000C
032 #define PHYSR       0x10010
033 #define TCTLR       0x20000
034 #define TSTR        0x20004
035 #define INTR        0x20008
036 #define INTER       0x2000C
037 #define EH0R        0x20010
038 #define EH1R        0x20014
039 #define EH2R        0x20018
040 #define EH3R        0x2001C
041 #define ERRFR       0x20020
042 #define ERRFER      0x20024
043 #define ERRFR2      0x20028
044 #define MSIR        0x20040
045 #define MSIFR       0x20044
046 #define PWRCTLR     0x20100
047 #define PCCTLR      0x20180
048 #define LAR0        0x20200
049 #define LAMR0       0x20208
050 #define LAR1        0x20220
051 #define LAMR1       0x20228
052 #define LAR2        0x20240
053 #define LAMR2       0x20248
```

```
054 #define LAR3         0x20260
055 #define LAMR3        0x20268
056 #define LAR4         0x20280
057 #define LAMR4        0x20288
058 #define LAR5         0x202A0
059 #define LAMR5        0x202A8
060 #define PALR0        0x20400
061 #define PAHR0        0x20404
062 #define PAMR0        0x20408
063 #define PTCTLR0      0x2040C
064 #define PALR1        0x20420
065 #define PAHR1        0x20424
066 #define PAMR1        0x20428
067 #define PTCTLR1      0x2042C
068 #define PALR2        0x20440
069 #define PAHR2        0x20444
070 #define PAMR2        0x20448
071 #define PTCTLR2      0x2044C
072 #define PALR3        0x20460
073 #define PAHR3        0x20464
074 #define PAMR3        0x20468
075 #define PTCTLR3      0x2046C
076 #define DMAOR        0x21000
077 #define DMPALR0      0x21100
078 #define DMPAHR0      0x21104
079 #define DMSALR0      0x21108
080 #define DMBCNTR0     0x21110
081 #define DMSBCNTR0    0x21114
082 #define DMSTRR0      0x21118
083 #define DMCCAR0      0x21120
084 #define DMCHCR0      0x21128
085 #define DMCHSR0      0x2112C
086 #define DMPALR1      0x21140
087 #define DMPAHR1      0x21144
088 #define DMSALR1      0x21148
089 #define DMBCNTR1     0x21150
090 #define DMSBCNTR1    0x21154
091 #define DMSTRR1      0x21158
092 #define DMCCAR1      0x21160
093 #define DMCHCR1      0x21168
094 #define DMCHSR1      0x2116C
095 #define DMPALR2      0x21180
096 #define DMPAHR2      0x21184
097 #define DMSALR2      0x21188
098 #define DMBCNTR2     0x21190
099 #define DMSBCNTR2    0x21194
100 #define DMSTRR2      0x21198
101 #define DMCCAR2      0x211A0
102 #define DMCHCR2      0x211A8
103 #define DMCHSR2      0x211AC
104 #define DMPALR3      0x211C0
105 #define DMPAHR3      0x211C4
106 #define DMSALR3      0x211C8
107 #define DMBCNTR3     0x211D0
108 #define DMSBCNTR3    0x211D4
109 #define DMSTRR3      0x211D8
110 #define DMCCAR3      0x211E0
```

```
111 #define DMCHCR3     0x211E8
112 #define DMCHSR3     0x211EC
113 #define PCICONF0    0x40000
114 #define PCICONF1    0x40004
115 #define PCICONF2    0x40008
116 #define PCICONF3    0x4000C
117 #define PCICONF4    0x40010
118 #define PCICONF5    0x40014
119 #define PCICONF6    0x40018
120 #define PCICONF7    0x4001C
121 #define PCICONF8    0x40020
122 #define PCICONF9    0x40024
123 #define PCICONF10   0x40028
124 #define PCICONF11   0x4002C
125 #define PCICONF12   0x40030
126 #define PCICONF13   0x40034
127 #define PCICONF14   0x40038
128 #define PCICONF15   0x4003C
129 #define PMCAP0      0x40040
130 #define PMCAP1      0x40044
131 #define MSICAP0     0x40050
132 #define MSICAP1     0x40054
133 #define MSICAP2     0x40058
134 #define MSICAP3     0x4005C
135 #define MSICAP4     0x40060
136 #define MSICAP5     0x40064
137 #define EXPCAP0     0x40070
138 #define EXPCAP1     0x40074
139 #define EXPCAP2     0x40078
140 #define EXPCAP3     0x4007C
141 #define EXPCAP4     0x40080
142 #define EXPCAP5     0x40084
143 #define EXPCAP6     0x40088
144 #define EXPCAP7     0x4008C
145 #define EXPCAP8     0x40090
146 #define VCCAP0      0x40100
147 #define VCCAP1      0x40104
148 #define VCCAP2      0x40108
149 #define VCCAP3      0x4010C
150 #define VCCAP4      0x40110
151 #define VCCAP5      0x40114
152 #define VCCAP6      0x40118
153 #define VCCAP7      0x4011C
154 #define VCCAP8      0x40120
155 #define VCCAP9      0x40124
156 #define NUMCAP0     0x401B0
157 #define NUMCAP1     0x401B4
158 #define NUMCAP2     0x401B8
159 #define IDSETR1     0x41004
160 #define IDSETR2     0x41024
161 #define DSERSETR0   0x4102C
162 #define DSERSETR1   0x41030
163 #define TLSR        0x41044
164 #define TLCTLR      0x41048
165 #define DLSR        0x4104C
166 #define DLCTLR      0x41050
167 #define MACSR       0x41054
```

```
168 #define MACCTLR     0x41058
169 #define PMSR        0x4105C
170 #define PMCTLR      0x41060
171 #define TLINTENR    0x41064
172 #define DLINTENR    0x41068
173 #define MACINTENR   0x4106C
174 #define PMINTENR    0x41070
175 #define TXSR        0x44028
176 #define TXVC0SR     0x44108
177
178
179 /*
180  * PCIE DMAC
181  */
182 /* DMAOR */
183 #define   DMAE      (1 << 31)
184
185 /* CHCR */
186 #define   CHE    (1 << 31)
187 #define   DIR(x)    (x << 30)
188
189 #define PCIE_WRITE  1
190 #define PCIE_READ      0
191
192 /* CHSR */
193 #define   PE     (1 << 11)
194 #define   SE     (1 << 9)
195 #define   TE     (1 << 0)
196
197
198 /*
199  * PCI area
200  */
201 #ifdef CONFIG_PCIE_ROOT
202 #define PCIE_AREA_ADDR    0xFD000000       /* PCI area 0 address */
203 #define PCIE_AREA_IO_ADDR 0xFE100000       /* PCI area 3 address */
204 #else
205 #define PCIE_AREA_ADDR    0xFD800000       /* PCI area 0 address */
206 #define PCIE_AREA_IO_ADDR 0xFE300000       /* PCI area 3 address */
207 #endif
208
209 /*
210  * SDRAM DATA area
211  */
212 #define sdram_data_area       0x0C000000    /* SDRAM DATA AREA */
213
214 /*
215  * PCIE data macroinstruction
216  */
217 #define   PCIE_MAX_CONFREG_SIZE   0x1000    /* Configuration register size 256 byte */
218
219 /*
220  * PCIE Configuration register
221  */
222 #define PCIE_CONF_DEVICE_ID         0x00
223 #define PCIE_CONF_VENDER_ID         0x00
224 #define PCIE_CONF_STATUS            0x04
```

```
225 #define PCIE_CONF_COMMAND           0x04
226 #define PCIE_CONF_CLASS_CODE        0x08
227 #define PCIE_CONF_REVISION_ID       0x08
228 #define PCIE_CONF_BIST              0x0C
229 #define PCIE_CONF_HEADER_TYPE       0x0C
230 #define PCIE_CONF_LATENCY_TIMER     0x0C
231 #define PCIE_CONF_CACHE_LINE_SIZE   0x0C
232 #define PCIE_CONF_BASE_ADDRESS_0    0x10
233 #define PCIE_CONF_BASE_ADDRESS_1    0x14
234 #define PCIE_CONF_BASE_ADDRESS_2    0x18
235 #define PCIE_CONF_BASE_ADDRESS_3    0x1C
236 #define PCIE_CONF_BASE_ADDRESS_4    0x20
237 #define PCIE_CONF_BASE_ADDRESS_5    0x24
238 #define PCIE_CONF_SUB_SYSTEM_ID     0x2C
239 #define PCIE_CONF_SUB_VENDER_ID     0x2C
240 #define PCIE_CONF_EXP_ROM_BASE      0x30
241 #define PCIE_CONF_CAP_PTR           0x34
242 #define PCIE_CONF_MAX_LAT           0x3C
243 #define PCIE_CONF_MIN_GNT           0x3C
244 #define PCIE_CONF_INTERRUPT_PIN     0x3C
245 #define PCIE_CONF_INTERRUPT_LINE    0x3C
246
247 /* Register accessing  */
248 #define PCIE_WRITEL(addr,data)     (*((volatile unsigned long  *)(addr)) = (unsigned long)(data))
249 #define  PCIE_READL(addr)    (*((volatile unsigned long  *)(addr)))
250 #define PCIE_WRITEW(addr,data)     (*((volatile unsigned short *)(addr)) = (unsigned short)(data))
251 #define  PCIE_READW(addr)    (*((volatile unsigned short *)(addr)))
252 #define PCIE_WRITEB(addr,data)     (*((volatile unsigned char  *)(addr)) = (unsigned char)(data))
253 #define  PCIE_READB(addr)    (*((volatile unsigned char  *)(addr)))
254
255
256 /*
257  * A configuration space register's structure
258  */
259
260 /* Header Type 0 */
261 typedef struct _PCIE_CONF_DATA {
262 #if defined(_BIG)
263    unsigned short   DeviceID;              /* Device ID */
264    unsigned short   VenderID;              /* Vender ID */
265
266    unsigned short   Status;                /* Device Status */
267    unsigned short   Command;               /* Device Control */
268
269    unsigned char BaseClass;                /* Base Class */
270    unsigned char SubClass;                 /* Sub Class */
271    unsigned char ProgrammingInterface;     /* Programming Interface */
272    unsigned char RevisionID;               /* Revision ID */
273
274    unsigned char BIST;                     /* BIST */
275    unsigned char HeaderType;               /* Header Type */
276    unsigned char LatencyTimer;             /* Master latency timer */
277    unsigned char CachLineSize;             /* Cach Line Size */
278
279    unsigned long BaseAddressRegisters[6];  /* Base Address Registers space */
280
281    unsigned long CardbusCISpointer;        /* Card Bus CIS pointer */
```

```
282
283    unsigned short   SubsystemID;           /* Subsystem ID */
284    unsigned short   SubsystemVenderID;     /* Subsystem Vender ID */
285
286    unsigned long ExpantionROMbaseAddress;  /* Expantion ROM base Address */
287
288    unsigned char reserve0[3];              /* Reserve */
289    unsigned char CAP_PTR;                  /* New facility pointer */
290
291    unsigned long reserve1;                 /* Reserve */
292
293    unsigned char Max_Lat;                  /* MAX latency */
294    unsigned char Min_Gnt;                  /* MIN Grant */
295    unsigned char InttreuptPin;             /* PCI Inttreupt pin */
296    unsigned char InttreuptLine;            /* Interrupt line */
297 #else
298    unsigned short   VenderID;              /* Vender ID */
299    unsigned short   DeviceID;              /* Device ID */
300
301    unsigned short   Command;               /* Device Control */
302    unsigned short   Status;                /* Device Status */
303
304    unsigned char RevisionID;               /* Revision ID */
305    unsigned char ProgrammingInterface;     /* Programming Interface */
306    unsigned char SubClass;                 /* Sub Class */
307    unsigned char BaseClass;                /* Base Class */
308
309    unsigned char CachLineSize;             /* Cach Line Size */
310    unsigned char LatencyTimer;             /* Master latency timer */
311    unsigned char HeaderType;               /* Header Type */
312    unsigned char BIST;                     /* BIST */
313
314    unsigned long BaseAddressRegisters[6];  /* Base Address Registers space */
315
316    unsigned long CardbusCISpointer;        /* Card Bus CIS pointer */
317
318    unsigned short   SubsystemVenderID;     /* Subsystem Vender ID */
319    unsigned short   SubsystemID;           /* Subsystem ID */
320
321    unsigned long ExpantionROMbaseAddress;  /* Expantion ROM base Address */
322
323    unsigned char CAP_PTR;                  /* New facility pointer */
324    unsigned char reserve0[3];              /* Reserve */
325
326    unsigned long reserve1;                 /* Reserve */
327
328    unsigned char InttreuptLine;            /* Interrupt line */
329    unsigned char InttreuptPin;             /* PCI Inttreupt pin */
330    unsigned char Min_Gnt;                  /* MIN Grant */
331    unsigned char Max_Lat;                  /* MAX latency */
332
333 #endif
334 } PCIE_CONF_DATA;
335
336
337 #endif /* _PCIE_H_ */
```

(5) ”scif.c”

This is a program listing of the SCIF0 initial settings and serial driver function.

```
001 /***********************************************************************
002 * DISCLAIMER
003
004 * This software is supplied by Renesas Electronics  Corporation. and is only
005 * intended for use with Renesas products. No other uses are authorized.
006
007 * This software is owned by Renesas Electronics  Corporation. and is protected under
008 * all applicable laws, including copyright laws.
009
010 * THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
011 * REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
012 * INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
013 * PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
014 * DISCLAIMED.
015
016 * TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
017 * ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
018 * FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
019 * FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
020 * AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
021
022 * Renesas reserves the right, without notice, to make changes to this
023 * software and to discontinue the availability of this software.
024 * By using this software, you agree to the additional terms and
025 * conditions found by accessing the following link:
026 * http://www.renesas.com/disclaimer
027 ***********************************************************************/
028 /* Copyright (C) 2010. Renesas Electronics Corporation., All Rights Reserved.*/
029 /*""FILE COMMENT""************ Technical reference data ****************
030 * System Name  : SH7786 Sample Program
031 * File Name    : scif.c
032 * Abstract     : The example of a set of SCIF Sample Program
033 * Version      : Ver 1.00
034 * Device       : SH7786
035 * Tool-Chain   : High-performance Embedded Workshop (Version 4.07.00.007)
036 *              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
037 * OS           : None
038 * H/W Platform : SH-4A Board P/N:AP-SH4AD-3A (Manufacturer:ALPHA PROJECT)
039 * Description  : It is an example program of the example of a SH7786 SCIF set.
040 *              :
041 * Operation    :
042 * Limitation   :
043 *              :
044 ********************************************************************
045 * History      : 01.Sep.2010 Ver. 1.00 First Release
046 *""FILE COMMENT END""*************************************************/
047
048
049 #include "scif.h"
050
051 /*""FUNC COMMENT""*****************************************************
052 * ID               :
053 * Outline          : Sample Program Main
054 *                  :
```

```
055 * Include           :
056 * Declaration       : int delay( int cnt )
057 * Description       : Software weight
058 *                   : A part for the count of "cnt" and a "for" are repeated.
059 *                   :
060 *                   :
061 *                   :
062 *                   :
063 * Limitation        :
064 *                   :
065 * Argument          : cnt
066 * Return Value      : none
067 * Calling Functions :
068 *""FUNC COMMENT END""**************************************************/
069 void delay( int cnt )
070 {
071    int i;
072    for(i=0;i<cnt;i++);
073 }
074
075 /*""FUNC COMMENT""*****************************************************
076 * ID                :
077 * Outline           : Sample Program Main
078 *                   :
079 * Include           :
080 * Declaration       : int scif_init(void)
081 * Description       : The initialization of SCIF
082 *                   :
083 *                   :
084 *                   :
085 *                   :
086 *                   :
087 * Limitation        :
088 *                   :
089 * Argument          : none
090 * Return Value      : -1: Baud rate clock count error
091 * Calling Functions :
092 *""FUNC COMMENT END""*************************************************/
093 int scif_init(void)
094 {
095    unsigned short data;
096    int t = -1, cnt = 0;
097
098    SCIF.SCSCR.WORD = 0x0000;  /* TIE, RIE, TE, RE Clear */
099
100    SCIF.SCFCR.BIT.TFCL = 1;   /* Tx FIFO Clear */
101    SCIF.SCFCR.BIT.RFCL = 1;   /* Rx FIFO Clear */
102
103    SCIF.SCFSR.WORD = 0x0000;  /* BRK, DR, TR Clear */
104    SCIF.SCLSR.BIT.ORER = 0;   /* ORER Clear */
105
106 #if defined(CONFIG_SCIF_CLK_EXTERNAL)
107    SCIF.SCSCR.BIT.CKE = 2; /* Clock source: SCK */
108 #elif  defined(CONFIG_SCIF_CLK_PCLK)
109    SCIF.SCSCR.BIT.CKE = 0; /*  Clock source: PCLK */
110    t = SCBRR_VALUE(CONFIG_BPS, CONFIG_SCIF_CLK_PCLK);
111 #endif /* CONFIG_SCIF_CLK */
```

```
112
113    if(t > 0) {
114       while(t >= 256) {
115          cnt++;
116          t >> 2;
117       }
118       if(cnt > 3)
119          return -1;
120
121       SCIF.SCSMR.BIT.CKS = cnt;
122       SCIF.SCBRR = t;
123    }
124    delay(1000);
125
126    SCIF.SCFCR.BIT.RTRG = 0;
127    SCIF.SCFCR.BIT.TTRG = 0;
128    SCIF.SCFCR.BIT.TFCL = 1;   /* Tx FIFO Clear */
129    SCIF.SCFCR.BIT.RFCL = 1;   /* Rx FIFO Clear */
130
131    SCIF.SCFCR.BIT.TFCL = 0;   /* Tx FIFO Not Clear */
132    SCIF.SCFCR.BIT.RFCL = 0;   /* Rx FIFO Not Clear */
133    SCIF.SCSCR.BIT.TE  = 1;
134    SCIF.SCSCR.BIT.RE  = 1;
135    return 0;
136 }
137
138 /*""FUNC COMMENT""*****************************************************
139 * ID                :
140 * Outline           : Sample Program Main
141 *                   :
142 * Include           :
143 * Declaration       : void scif_transmit_data( char *Data )
144 * Description       : A transmission of two or more byte data of SCIF.
145 *                   :
146 *                   :
147 *                   :
148 *                   :
149 *                   :
150 * Limitation        :
151 *                   :
152 * Argument          : *Data: A send data is stored.
153 * Return Value      : none
154 * Calling Functions :
155 *""FUNC COMMENT END""*************************************************/
156 void  scif_transmit_data( char   *Data )
157 {
158    while( *Data )
159    {
160       while(!(SCIF.SCFSR.BIT.TDFE));   /* Weight is carried out until the write of a send data will be in an authorized state.*/
161       SCIF.SCFTDR = *Data;              /* A set of a send data */
162       Data++;
163       while(!(SCIF.SCFSR.BIT.TEND));    /* Waiting for the quit of transmitting */
164       SCIF.SCFSR.BIT.TDFE = 0;
165       SCIF.SCFSR.BIT.TEND = 0;
166    }
167 }
168
```

```
/*""FUNC COMMENT""*****************************************************
* ID                : 
* Outline           : Sample Program Main
*                   : (PCIe)
* Include           : 
* Declaration       : void scif_transmit_byte_data( char *Data )
* Description       : A transmission of the single byte data of SCIF
*                   : 
*                   : 
*                   : 
*                   : 
*                   : 
* Limitation        : 
*                   : 
* Argument          : *Data: A send data is stored.
* Return Value      : none
* Calling Functions : 
*""FUNC COMMENT END""*************************************************/
void  scif_transmit_data_byte( char  *Data )
{
      while(!(SCIF.SCFSR.BIT.TDFE));   /* Weight is carried out until the write of a send data will be in an authorized state. */
      SCIF.SCFTDR = *Data;              /* A set of a send data */
      while(!(SCIF.SCFSR.BIT.TEND));    /* Waiting for the quit of transmitting  */
      SCIF.SCFSR.BIT.TDFE = 0;
      SCIF.SCFSR.BIT.TEND = 0;
}


/*""FUNC COMMENT""*****************************************************
* ID                : 
* Outline           : Sample Program Main
*                   : (PCIe)
* Include           : 
* Declaration       : void sci_printf(char* str, ...)
* Description       : A text with a format is outputted.
*                   : 
*                   : 
*                   : 
*                   : 
*                   : 
* Limitation        : 
*                   : 
* Argument          : *Data: A send data is stored.
* Return Value      : none
* Calling Functions : 
*""FUNC COMMENT END""*************************************************/
/***********************************/
/* A text with a format is outputted    */
/***********************************/
#define PRINTF_SIZE 1024
static char printf_str[PRINTF_SIZE];

void scif_printf(char* str, ...)
{
   va_list args;
   size_t size;

```

```
226    size = strlen(str);
227
228    if( size > PRINTF_SIZE ) {
229       return;
230    }
231
232    va_start(args, str);
233    vsprintf(printf_str, str, args);
234    va_end(args);
235
236    scif_transmit_data(printf_str);
237 }
238
239 /*""FUNC COMMENT""*****************************************************
240 * ID                :
241 * Outline           : Sample Program Main
242 *                   :
243 * Include           :
244 * Declaration       : char scif_recive_data( char *Data )
245 * Description       : The data of SCIF is received.
246 *                   :
247 *                   :
248 *                   :
249 *                   :
250 *                   :
251 * Limitation        :
252 *                   :
253 * Argument          : *Data: A receive data is stored.
254 * Return Value      : -1: A receive data error
255 * Calling Functions :
256 *""FUNC COMMENT END""************************************************/
257 char  scif_recive_data( char  *Data )
258 {
259    unsigned char ReadData, i = 0;
260    char   ret_cd = 0;
261
262    for(;;)
263    {
264       if(( SCIF.SCFSR.BIT.ER  ) ||
265          ( SCIF.SCFSR.BIT.BRK ) ||
266          ( SCIF.SCFSR.BIT.DR  ))       /* An error occurs? */
267       {
268          ReadData = SCIF.SCFRDR;       /* Read of a data dummy */
269          ret_cd = -1;  /* A set of a reception error */
270          SCIF.SCFSR.WORD &= 0x0000;    /* A clear of an error  */
271          SCIF.SCLSR.WORD &= 0x0000;
272       }
273       else if( SCIF.SCFSR.BIT.RDF )    /* A data was received? */
274       {
275          *Data = SCIF.SCFRDR;          /* A data is acquired   */
276          SCIF.SCFSR.BIT.RDF = 0;       /* A clear of a receive data sign */
277          SCIF.SCFSR.BIT.DR  = 0;       /* A clear of a reception sign */
278          scif_transmit_data_byte( Data );
279          if( *Data == '\n' )           /* An obtaining data is CR? */
280          {
281             break; /* A processing is completed. */
282          }
```

```
283          if( *Data == 0x0d )           /* An obtaining data is CR? */
284          {
285             break; /* A processing is completed. */
286          }
287          Data++; /* The following set of the one-plus-one address which the data acquired */
288          if( ++i == 4 )
289          {
290             ret_cd = -1;
291          }
292       }
293       if( ret_cd == -1 )
294       {
295          break;
296       }
297    }
298    return( ret_cd );
299 }
300
301 /*""FUNC COMMENT""*****************************************************
302 * ID                :
303 * Outline           : Sample Program Main
304 *                   :
305 * Include           :
306 * Declaration       : char scif_recive_data_byte( char   *Data )
307 * Description       : A data reception of SCIF
308 *                   :
309 *                   :
310 *                   :
311 *                   :
312 *                   :
313 * Limitation        :
314 *                   :
315 * Argument          : *Data: A receive data is stored.
316 * Return Value      : -1: A receive data error
317 * Calling Functions :
318 *""FUNC COMMENT END""*************************************************/
319 char  scif_recive_data_byte( char   *Data )
320 {
321    unsigned char ReadData, i = 0;
322    char   ret_cd = 0;
323
324    for(;;)
325    {
326       if(( SCIF.SCFSR.BIT.ER  ) ||
327          ( SCIF.SCFSR.BIT.BRK ) ||
328          ( SCIF.SCFSR.BIT.DR  ))       /* An error occurs? */
329       {
330          ReadData = SCIF.SCFRDR;       /* Read of a data dummy */
331          ret_cd = -1;  /* A set of a reception error  */
332          SCIF.SCFSR.WORD &= 0x0000;    /* A clear of an error  */
333          SCIF.SCLSR.WORD &= 0x0000;
334       }
335       else if( SCIF.SCFSR.BIT.RDF )    /* A data was received? */
336       {
337          *Data = SCIF.SCFRDR;          /* A data is acquired   */
338          SCIF.SCFSR.BIT.RDF = 0;       /* A clear of a receive data sign */
339          SCIF.SCFSR.BIT.DR  = 0;       /* A clear of a receive data sign */
```

```
340 //        scif_transmit_data_byte( Data );
341           break; /* A processing is completed. */
342        }
343    }
344    return( ret_cd );
345 }
346
```

(6) ”scif.h”

This is a header file used by the SCIF0 initial settings and serial driver function.

```
01
02 #ifndef _SCIF_H
03 #define _SCIF_H
04
05 #include "config.h"
06
07 #if defined(CONFIG_SCIF0)
08 #define SCIF   (*(volatile struct st_scif    *)0xFFEA0000) /* SCIF0   Address */
09 #elif defined(CONFIG_SCIF1)
10 #define SCIF   (*(volatile struct st_scif    *)0xFFEB0000) /* SCIF1   Address */
11 #elif defined(CONFIG_SCIF2)
12 #define SCIF   (*(volatile struct st_scif    *)0xFFEC0000) /* SCIF2   Address */
13 #elif defined(CONFIG_SCIF3)
14 #define SCIF   (*(volatile struct st_scif    *)0xFFED0000) /* SCIF3   Address */
15 #elif defined(CONFIG_SCIF4)
16 #define SCIF   (*(volatile struct st_scif    *)0xFFEE0000) /* SCIF4   Address */
17 #elif defined(CONFIG_SCIF5)
18 #define SCIF   (*(volatile struct st_scif    *)0xFFEF0000) /* SCIF5   Address */
19 #endif /* CONFIG_SCIFn */
20
21 //#define SCBRR_VALUE(bps, clk) ((clk+16*bps)/(16*bps)-1)
22 #define SCBRR_VALUE(bps, clk) ((clk)/(32*bps)-1)
23
24 /* SCFCR */
25 #defineRTRG1  0
26 #defineRTRG16 1
27 #defineRTRG32 2
28 #defineRTRG48 3
29 #defineTTRG32 0
30 #defineTTRG16 1
31 #defineTTRG2  2
32 #defineTTRG0  3
33
34
35
36 #endif /* _SCIF_H */
```

5. Reference Documents

- Software Manual
  SH4-A Software Manual (REJ09B0003)
  (The latest version can be downloaded from the Renesas Electronics Web site.)
- Hardware Manual
  SH7786 Group User's Manual: Hardware (REJ09B0501)
  (The latest version can be downloaded from the Renesas Electronics Web site.)
- Evaluation Board Manual
  AP-SH4AD-0A SH-4A Multi SH7786 CPU Board: Hardware Manual
  (The latest version can be downloaded from the Alpha Project Web site.)

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

PCIe® is a registered trademark of PCI-SIG®.

All trademarks and registered trademarks are the property of their respective owners.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jul.15.11	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2011 Renesas Electronics Corporation. All rights reserved.

Colophon 1.1