Spear PCIe Gadget Driver

Author

Pratyush Anand (pratyush.anand@gmail.com)

Location

driver/misc/spear13xx_pcie_gadget.c

Supported Chip:

SPEAr1300 SPEAr1310

Menuconfig option:

Device Drivers

Misc devices

PCIe gadget support for SPEAr13XX platform

purpose

This driver has several nodes which can be read/written by configfs interface. Its main purpose is to configure selected dual mode PCIe controller as device and then program its various registers to configure it as a particular device type. This driver can be used to show spear's PCIe device capability.

Description of different nodes:

read behavior of nodes:

link	gives ltssm status.
int_type	type of supported interrupt
no_of_msi	zero if MSI is not enabled by host. A positive value is the number of MSI vector granted.
vendor_id	returns programmed vendor id (hex)
device_id	returns programmed device id(hex)
bar0_size:	returns size of bar0 in hex.
bar0_address	returns address of bar0 mapped area in hex.
bar0_rw_offset	returns offset of bar0 for which bar0_data will return value.
bar0_data	returns data at bar0_rw_offset.

write behavior of nodes:

link	write UP to enable ltsmm DOWN to disable
int_type	write interrupt type to be configured and (int_type could be INTA, MSI or NO_INT). Select MSI only when you have programmed no_of_msi node.
no_of_msi	number of MSI vector needed.
inta	write 1 to assert INTA and 0 to de-assert.
send_msi	write MSI vector to be sent.
vendor_id	write vendor id(hex) to be programmed.
device_id	write device id(hex) to be programmed.
bar0_size	write size of bar0 in hex. default bar0 size is 1000 (hex) bytes.
bar0_address	write address of bar0 mapped area in hex. (default mapping of bar0 is SYSRAM1(E0800000). Always program bar size before bar address. Kernel might modify bar size and address for alignment, so read back bar size and address after writing to cross check.
bar0_rw_offset	write offset of bar0 for which bar0_data will write value.
bar0_data	write data to be written at bar0_rw_offset.

Node programming example

Program all PCIe registers in such a way that when this device is connected to the PCIe host, then host sees this device as 1MB

RAM.

```
#mount -t configfs none /Config
```

For nth PCIe Device Controller:

```
# cd /config/pcie_gadget.n/
```

Now you have all the nodes in this directory. program vendor id as 0x104a:

```
# echo 104A >> vendor_id
```

program device id as 0xCD80:

```
# echo CD80 >> device_id
```

program BAR0 size as 1MB:

```
# echo 100000 >> bar0_size
```

check for programmed bar0 size:

```
# cat bar0_size
```

Program BAR0 Address as DDR (0x2100000). This is the physical address of memory, which is to be made visible to PCIe host. Similarly any other peripheral can also be made visible to PCIe host. E.g., if you program base address of UART as BAR0 address then when this device will be connected to a host, it will be visible as UART.

```
# echo 2100000 >> bar0_address
```

program interrupt type : INTA:

```
# echo INTA >> int_type
```

go for link up now:

```
# echo UP >> link
```

It will have to be insured that, once link up is done on gadget, then only host is initialized and start to search PCIe devices on its port.

```
/*wait till link is up*/
# cat link
```

Wait till it returns UP.

To assert INTA:

```
# echo 1 >> inta
```

To de-assert INTA:

```
# echo 0 >> inta
```

if MSI is to be used as interrupt, program no of msi vector needed (say4):

```
# echo 4 >> no_of_msi
```

select MSI as interrupt type:

```
# echo MSI >> int_type
```

go for link up now:

```
# echo UP >> link
```

wait till link is up:

```
# cat link
```

An application can repetitively read this node till link is found UP. It can sleep between two read.

wait till msi is enabled:

```
# cat no_of_msi
```

Should return 4 (number of requested MSI vector)

to send msi vector 2:

```
# echo 2 >> send_msi
# cd -
```