BSP Documentation

NXP MCIMX7-SABRE board

NXP Semiconductors

Contents

Chapter 1

BSP Overview

1.1 MCIMX7-SABRE board

Provided BSP relates to MCIMX7-SABRE board, rev. C

Chapter 2

How to start

2.1 How to import BSP into Momentics IDE

Before working with a BSP in the IDE, you must first import it. When you import the BSP source, the IDE creates a System Builder project.

To import the BSP source code:

1. Select **File** in QNX Momentics IDE menu and click on **Import**.
2. In opened **Import** window expand the QNX folder.
3. Select **QNX Source Package and BSP (archive)** from the list and click on **Next** button.
4. In the **Select the archive file** dialog click on **Browse...** button and then choose the BSP archive using the file browser.
5. Click on **Next** button.
6. In the **Package selected to import** dialog choose the BSP you want. You'll see a description of it.
7. Click on **Next** button.
8. Click Finish. All the projects will be created and the source brought from the archive.

2.2 How to create bootable SD card

You can use any or several different methods to prepare QNX bootable SD card.

The U-Boot utility offers many different ways to boot an image from the SD slot.

2.2.1 How to use Windows (7) to prepare and partition a bootable SD card

1. Build the BSP to produce the IPL and IFS images in the /**images** directory.
2. Convert IPL into BINARY format.
   Run DOS command line (press Windows key + R, type **cmd** and press ENTER):

```
1 Microsoft Windows [Version 6.1.7601]
2 Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
3
4 c:\>_
```

   Set QNX environment variables by **qnx660-env.bat** batch. This batch is stored in qnx660 installation path, by default in **c:/qnx660**/ place.

```
1 Microsoft Windows [Version 6.1.7601]
2 Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
3
4 c:\>c:\qnx660\qnx660-env.bat
5
6 c:\>REM  This script is sets environment variables requires to use this version
7  of QNX Software Development Platform 6.6
8
9 c:\>REM  from the command line.
10
11 ...
12
13 c:\>_
```

   Change directory path into **bsp-nxp-mx7d-sabre-sdp660** (for example: c:/QNX_Momentics/bsp-nxp-mx7d-sabre-sdp660/) and run the ntoarmv7-objcopy utility with correspond parameters:

```
1 c:\>cd c:\QNX_Momentics\bsp-nxp-mx7d-sabre-sdp660\
2
3 c:\QNX_Momentics\bsp-nxp-mx7d-sabre-sdp660\>ntoarmv7-objcopy
4 --input-format=elf32-littlearm --output-format=binary
5 install/armle-v7/boot/sys/ipl-imx-sabre images/ipl-imx-sabre.bin
6
7 c:\QNX_Momentics\bsp-nxp-mx7d-sabre-sdp660\>_
```

   The images directory should now have a file called **ipl-imx-sabre.bin**.
3. Insert the SD card into the reader. If inserted SD card contains one FAT16 or FAT32 partition, skip 4. - 9. steps and continue by step 10.
4. Start **diskpart** utility in command line:

```
1 c:\>diskpart
2
3 Microsoft DiskPart version 6.1.7601
4 Copyright (C) 1999-2008 Microsoft Corporation.
5 On computer: ...
6
7 DISKPART>_
```

5. Select disk (SD card) to partition:

```
c:\>list disk

DISKPART> list disk

Disk ###  Status         Size     Free     Dyn  Gpt
--------  -------------  -------  -------  ---  ---
Disk 0    Online          465 GB  1024 KB
Disk 1    Online         7580 MB  4407 MB

DISKPART>_
```

In list above, **Disk 1** is a SD card (according disk size).

```
DISKPART> select disk 1

Disk 1 is now the selected disk.

DISKPART>_
```

6. Delete disk (SD card) partition(s):

```
DISKPART> list part

Partition ###  Type              Size     Offset
-------------  ----------------  -------  -------
Partition 1    Primary            512 MB  1024 KB
Partition 0    Primary            612 MB   513 MB
Partition 0    Primary           2048 MB  1125 MB

DISKPART>_
```

Delete all available partitions...

```
DISKPART> select part 1

Partition 1 is now the selected partition.

DISKPART> delete part

DiskPart successfully deleted the selected partition.

DISKPART> select part 1

Partition 1 is now the selected partition.

DISKPART> delete part

DiskPart successfully deleted the selected partition.

DISKPART> select part 1

Partition 1 is now the selected partition.

DISKPART> delete part

DiskPart successfully deleted the selected partition.

DISKPART>_
```

7. Create primary partition:

```
1 DISKPART> create part pri
2
3 DiskPart succeeded in creating the specified partition.
4
5 DISKPART>_
```

8. Exit from diskpart utility:

```
1 DISKPART> exit
2
3 Leaving DiskPart...
4
5 c:\>_
```

9. Format SD cart to FAT32 file system:
   Type **format G: /FS:FAT32 /Q** to command line.
   Used parameters:

   - **G:** - SD card driver letter. **Make sure you select the correct disk!** Wrong selected driver letter cause data lost.
   - **/FS:FAT32** - Specifies the type of the file system.
   - **/Q** - Performs a quick format.

```
1 c:\>format G: /FS:FAT32 /Q
2
3 Insert new disk for drive G:
4 and press ENTER when ready...
```

   Press ENTER...

```
1 The type of the file system is NTFS.
2 The new file system is FAT32.
3 QuickFormatting 7579M
4 Initializing the File Allocation Table (FAT)...
```

   Enter volume label (e.g. QNX).

```
1 Volume label (11 characters, ENTER for none)? QNX
```

   Press ENTER...

```
1 Format complete.
2         7.4 GB total disk space.
3         7.4 GB are available.
4
5         4,096 bytes in each allocation unit.
6         1,936,127 allocation units available on disk.
7
8         32 bits in each FAT entry.
9
10         Volume Serial Number is D00F-4F25
11
12 c:\>_
```

10. Copy the IPL to the SD card:
    For copying IPL/u-boot to the booting SD card is possible to use **cfimager** utility. cfimager utility can be downloaded here.

```
1 c:\>cfimager -raw -offset 0x400 -skip 0x400 -f ipl-imx-sabre.bin -d G
2
3 c:\>_
```

Used parameters:

- **-raw** - Write Image to physical location.
- **-offset 0x400** - Physical location offset.
- **-skip 0x400** - Skip how many byte of firmware image.
- **-f u-boot.bin** - Input firmware file, IPL/u-boot binary file.
- **-d G** - Card reader drive letter without colon (e.g driver letter G).

11. Copy the QNX-IFS image to the SD card:

```
1 c:\QNX_Workspace\bsp-nxp-mx7d-sabre-sdp660\Images\>copy qnx-ifs G:\
2     1 file(s) copied
3
4 c:\>_
```

12. Remove the SD card from the card reader and insert SD card onto the selected boot slot on target board.

2.2.2 How to use Linux Ubuntu to prepare and partition a bootable SD card

1. Build the BSP to produce the IPL and IFS images in the **/images** directory.
2. Run the **mkflashimage** script from the **/images** directory. The images directory should now have a file called **ipl-imx-sabre.bin**.
3. Insert the SD card into the reader.
4. From the terminal, run the following command, once with the SD card out of the reader:

```
1 $ mount
2  . . .
```

Note the mounted devices listed. Now insert the card and run the same command a second time.

```
1 $ mount
2  . . .
3 /dev/sda1
```

When the command is run with the card inserted, an additional device should appear (in this case **sda1**). This additional device is the target device. For the remainder of these instructions, we will use **sda1** the device used.Please substitute your own device, which you got by running mountpoint

5. If inserted SD card contains one FAT16 or FAT32 partition, skip 6. - 13. steps and please continue by step 14.
Log on with administrator privileges, then run the following commands, substituting your device for sda1.
6. Build a new DOS disk label:

```
1 $ sudo fdisk /dev/sda
2
3 Command (m for help): u
4 Changing display/entry units to cylinders (DEPPRECATED!)
5
6 Command (m for help): o
7
8 Building a new DOS disklabel with disk identifier 0xcdd1b702.
9 Changes will remain in memory only, until you decide to write them.
```

```
10 After that, of course, the previous content won't be recoverable.
11
12 Warning: invalid flag 0x0000 of partition table 4 will be corrected by write.
13 WARNING: cylinders as display units are deprecated. Use command
14   'u' to change units to sectors.
15
16 Command (m for help): d
17     Selected partition 1
```

7. Remove the existing partitions. Keep entering the d command until no partitions are left:

```
1 Command (m for help): d
2 No partition is defined yet!
```

8. Create a new partition:

```
1 Command (m for help): n
2 Partition type:
3 p - primary (0 primary, 0 extended, 4 free)
4 e - extended
```

9. Set the partition as the primary partition:

```
1 Select (default p): p
2
3 Partition number (1-4, default 1): 1
4
5 First cylinder (1-240, default 1): 1
6
7 Last cylinder, (100-240, default 240): <ENTER>
8
9 Using default value 240
```

10. Set the active partition:

```
1 Command (m for help): a
2 Partition number (1-4): 1
```

11. Set the partition type:

```
1 Command (m for help): t
2 Selected partition 1
3
4 Hex code (type L to list codes): c
5 Changed system type of partition 1 to c (W95 FAT32 (LBA))
```

12. Write the changes:

```
1 Command (m for help): w
2
3 The partition table has been altered!
4
5 Calling ioctl() to re-read partition table.
6 Syncing disks.
```

13. Format the SD card:

```
1 $ sudo mkfs.vfat /dev/sda1
2 mkfs.msdos 3.0.12 (29 Oct 2011)
```

14. Copy the IPL to the SD card:

```
1 $ sudo dd if=ipl-imx-sabre.bin of=/dev/sda bs=512 seek=2
2 skip=2
3 37+1 records in
4 37+1 records out
5 19268 bytes (19 KB) copied, 0.0178466s, 1.1 MB/s
6
7 $ sync
```

    The SD card should appear in the list of mounted devices. If it does not appear, remove and re-insert it.

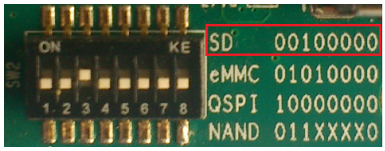
15. Copy the QNX IFS to the SD card by copying to the local mount point. Run mount again to find out your mount point:

```
1 $ mount
2  . . .
3 /dev/sda1 on /media/074B-DAC7
4 $ cp QNX-IFS /media/074B-DAC7
5
6 $ sync
```

16. Remove the SD card from the card reader and insert SD card onto the selected boot slot on target board.

2.3 How to configure the board switches

The NXP MCIMX7-SABRE board DIP switches should be set as shown below to select the desired functionality.



2.4 Start QNX on MCIMX7-SABRE board

1. Insert SD card with IPL and QNX-IFS image onto the selected boot slot on target board.
2. Connect a USB cable between the board's DEBUG UART (J11) and the USB port of your host machine (e.g. /dev/ttyS∗ on Linux, COM1 on Windows, etc).
   On your host machine, start your favourite terminal (e.g. TeraTerm, Putty, ...) program with these settings:
   - Baudrate: 115200
   - Data: 8 bits
   - Parity: None
   - Flow control: None
3. Connect external 5VDC power supply onto J1 connector on MCIMX7-SABRE board.
4. Power up the board by **SW1 PWR ON** switch.
   You should see output from IPL on your terminal console, similar to the following:

```
1 Welcome to QNX Neutrino Initial Program Loader for NXP i.MX7D Sabre (ARM Cortex-A7)
2 Command:
3 Press 'D' for serial download, using the 'sendnto' utility
4 Press 'M' for SDMMC download, IFS filename MUST be 'QNX-IFS'.
```

For QNX booting from SD1 BOOT card press 'M' letter.

```
1 SDMMC download...
2 load image done.
3 Found image               @ 0x88000008
4 Jumping to startup        @ 0x80805588
5
6 SCU_CONFIG = 00000002, 2 cpus
7 Enabling Dcache and MMU
8 CPU0: L1 Icache: 1024x32
9 CPU0: L1 Dcache: 512x64 WB
10 CPU0: L2 Dcache: 8192x64 WB
11 CPU0: VFP-d32 FPSID=41023075
12 CPU0: NEON MVFR0=10110222 MVFR1=11111111
13 CPU0: 410fc075: Cortex A7 rev 5 996MHz
14
15 Set recommended MCU voltages:
16     PMIC: SW1A = 1075 mV
17     PMIC: SW1B = 1000 mV
18
19 Detected i.MX7 Dual, revision TO1.1
20 Detected board revision: C
21
22 PLL_ARM    : 996MHz
23 PLL_SYS    : 480MHz
24 PLL_ENET   : 1000MHz
25 PLL_USB    : 480MHz
26 PLL_AUDIO  : 689MHz
27 PLL_VIDEO  : 24MHz
28 PLL_DRAM   : 792MHz
29 Cortex-A7     : 996000kHz
30 Cortex-M4     : 240000kHz
31 AHB clock     : 135000kHz
32 IPG clock     : 67500kHz
33 AXI clock     : 332308kHz
34 DDR clock     : 198000kHz
35 PLL_SYS PFD0 clock : 392728kHz
36 PLL_SYS PFD1 clock : 332308kHz
37 PLL_SYS PFD2 clock : 270000kHz
38 PLL_SYS PFD3 clock : 576000kHz
39 PLL_SYS PFD4 clock : 480000kHz
40 PLL_SYS PFD5 clock : 480000kHz
41 PLL_SYS PFD6 clock : 480000kHz
42 PLL_SYS PFD7 clock : 480000kHz
43 UART1 clock   : 80000kHz
44 GPT1 clock    : 24000kHz
45 ECSPI1 clock  : 60000kHz
46 EIM clock     : 120000kHz
47 NAND clock    : 480000kHz
48 QSPI clock    : 240000kHz
49 USDHC1 clock  : 196364kHz
50 USDHC2 clock  : 196364kHz
51 USDHC3 clock  : 392728kHz
52
53 decompressing...done
54 CPU1: L1 Icache: 1024x32
55 CPU1: L1 Dcache: 512x64 WB
56 CPU1: L2 Dcache: 8192x64 WB
57 CPU1: VFP-d32 FPSID=41023075
58 CPU1: NEON MVFR0=10110222 MVFR1=11111111
59 CPU1: 410fc075: Cortex A7 rev 5 996MHz
60 alloc_syspage_memory: syspage size:00001ab8 _syspage_ptr:80017000
61 callout_io_map: mapping paddr:31001000 returns:fc40e000
62 callout_io_map: mapping paddr:31002000 returns:fc40f000
```

```
63 callout_io_map: mapping paddr:30200000 returns:fc411000
64 callout_io_map: mapping paddr:30210000 returns:fc412000
65 callout_io_map: mapping paddr:30220000 returns:fc413000
66 callout_io_map: mapping paddr:30230000 returns:fc414000
67 callout_io_map: mapping paddr:30240000 returns:fc415000
68 callout_io_map: mapping paddr:30250000 returns:fc416000
69 callout_io_map: mapping paddr:30260000 returns:fc417000
70 callout_io_map: mapping paddr:31001000 returns:fc418000
71 callout_io_map: mapping paddr:30280000 returns:fc419000
72 callout_io_map: mapping paddr:302d0000 returns:fc429000
73 callout_io_map: mapping paddr:30860000 returns:fc42a000
74 callout_io_map: mapping paddr:30860000 returns:fc43a000
75 callout_io_map: mapping paddr:30860000 returns:fc44a000
76 cpu_startnext: cpu1 -> fc409844
77
78 System page at phys:80017000 user:fc408000 kern:fc408000
79 Starting next program at vfe05ee3c
80 cpu_startnext: cpu0 -> fe05ee3c
81 Welcome to QNX Neutrino 6.6.0 on the i.mx7 Sabre (ARM Cortex-A7)
82 Starting Watchdog driver...
83 Starting Serial driver (/dev/ser1)...
84 Starting I2C1,2,3,4 driver (/dev/i2c1,2,3,4)...
85 Starting SD1 memory card driver (/dev/sdx)...
86 Starting Ethernet driver (/dev/socket)...
87 Starting USB OTG1 USB OTG2 controllers in the host mode (/dev/io-usb/*)...
88 Starting CAN driver (/dev/can1/*)...
89 Starting RTC utility ...
90 setting env variables.
91 Launching devb-umass...
92
93 #
```

Chapter 3

Release Notes

This is a release note for QNX6.6.0 BSP for MCIMX7-SABRE board (rev. C)

BSP information	
Board	NXP MCIMX7-SABRE board, rev. C
BSP file	bsp-nxp-mx7d-sabre-qnx660-byymmdd.zip
Software Development Platform	QNX SDP 6.6.0
BSP Version	v1.0
Release Date	Jun, 2016
Producer	NXP Semiconductors

3.1 Installation

Please refer to the BSP Setup Manual for specific instructions on using this BSP with the specified target device.

3.2 Uninstallation

In order to uninstall the BSP, please just do a simple thing is delete bsp-nxp-mx7d-sabre-qnx660-byymmdd.zip file and extracted bsp-nxp-mx7d-sabre-qnx660-byymmdd folder from your system.

If using QNX IDE, delete the project by doing mouse right click on BSP project and selecting 'delete' option. When prompted select "Delete project contents on disk".

3.3 BSP Structure & Content

This BSP is based on BSP_freescale_imx6x-sabreARD_br-660 and BSP_freescale-imx6SoloX-sabre-sdb_br-660 BSPs and contains following drivers:

- IPL/Startup
- Audio driver

- SD/eMMC driver
- Debug serial driver
- DMA library
- I2C driver
- CAN driver
- Network driver
- USB driver
- Flash driver
- Nand driver
- Watchdog utility
- RTC driver

BSP Directory structure:

```
<bsp-nxp-mx7d-sabre-qnx660-byymmdd>
+-- <images>              : where the resultant boot images are places
+-- <install>             : gets populated at the beginning of the BSP
                            build process
+-- <prebuilt>            : contains the binaries, system binaries,
                            buidfiles, libraries, and header files that
                            are shipped with the BSP
+-- <src>                 : stored the whole source files of BSP
|   +-- <hardware>        : BSP source files
|   |   +-- <can>         : stores CAN driver source files
|   |   +-- <deva>        : stores Audio driver source files
|   |   +-- <devb>        : stores SD card and eMMC driver source files
|   |   +-- <devc>        : stores Serial driver source files
|   |   +-- <devnp>       : stores Network driver source files
|   |   +-- <devu>        : stores USB2.0 driver source files
|   |   |   +-- <dc>      : stores USB device driver source files
|   |   +-- <etfs>        : stores NAND driver source files
|   |   +-- <flash>       : stores QSPI flash driver source files
|   |   +-- <i2c>         : stores I2C driver source files
|   |   +-- <ipl>         : stores IPL source files
|   |   +-- <startup>     : stores Startup source files
|   |   +-- <support>
|   |   |   +-- <wdtkick> : stores watch dog timer utility source files
|   |   +-- Makefile      : compiling makefile
|   +-- <lib>             : drivers libraries and includes
|   |   +-- <dma>         : stores dma library source code
|   +-- <utils>           : test programs
|   |   +-- <r>           : stores RTC driver source files
|   +-- Makefile          : compiling makefile
+-- Makefile              : compiling make file
+-- source.xml            : some information about BSP, CPU name
```

3.4 BSP Change History

CREATED : 06.02.2016
MODIFIED : 06.02.2016

HISTORY: 06.02.2016 : v1.0

- Initial version established

3.5 Known Limitations

1. Flash driver (devf-qspi-imx) does not support Rx DMA transfer.
2. Nand driver (fs-etfs-imx-micron) supports only MT29F8G08xxxxx memory.
3. CAN driver (dev-can-mx7) doesn't support an external clock select.
4. SPI master driver is not supported yet.

Chapter 4

Ethernet driver for io-pkt (devnp-fec-imx.so)

This part describes an Ethernet driver.

Functions

- void bsd_mii_mediastatus (struct ifnet *ifp, struct ifmediareq *ifmr)
- int bsd_mii_mediachange (struct ifnet *ifp)
- void bsd_mii_initmedia (imx_fec_dev_t *imx_fec)
- void dump_mbuf (struct mbuf *m, uint32_t length)
- void * imx_rx_thread (void *arg)
- void imx_rx_thread_quiesce (void *arg, int die)
- void DumpMAC (imx_fec_dev_t *imx_fec)
- void DumpPhy (imx_fec_dev_t *imx_fec)
- void imx_speeduplex (imx_fec_dev_t *imx_fec)
- int imx_detect (void *dll_hdl, struct _iopkt_self *iopkt, char *options)
- int imx_ioctl (struct ifnet *ifp, unsigned long cmd, caddr_t data)
- int imx_process_queue (void *arg, struct nw_work_thread *wtp)
- int imx_enable_queue (void *arg)
- const struct sigevent * imx_isr (void *arg, int iid)
- int imx_enable_interrupt (void *arg)
- int imx_process_interrupt (void *arg, struct nw_work_thread *wtp)
- void imx_set_multicast (imx_fec_dev_t *)
- void imx_start (struct ifnet *)
- void imx_transmit_complete (imx_fec_dev_t *, uint8_t)
- int imx_set_tx_bw (imx_fec_dev_t *imx_fec, struct ifdrv *ifd)
- int imx_output (struct ifnet *, struct mbuf *, struct sockaddr *, struct rtentry *)
- int imx_receive (imx_fec_dev_t *, struct nw_work_thread *, uint8_t)
- void imx_MDI_MonitorPhy (void *)
- void imx_init_phy (imx_fec_dev_t *)
- int imx_get_phy_addr (imx_fec_dev_t *)
- int imx_setup_phy (imx_fec_dev_t *imx_fec)
- void imx_sabreauto_rework (imx_fec_dev_t *imx_fec)
- uint16_t imx_mii_read (void *handle, uint8_t phy_add, uint8_t reg_add)
- void imx_mii_write (void *handle, uint8_t phy_add, uint8_t reg_add, uint16_t data)
- void imx_mii_callback (void *handle, uchar_t phy, uchar_t newstate)
- void imx_bcm54220_phy_init (imx_fec_dev_t *imx_fec)
- void imx_update_stats (imx_fec_dev_t *)

- void imx_clear_stats (imx_fec_dev_t *)
- int imx_ptp_start (imx_fec_dev_t *)
- void imx_ptp_stop (imx_fec_dev_t *)
- int imx_ptp_is_eventmsg (struct mbuf *, ptpv2hdr_t **)
- void imx_ptp_add_rx_timestamp (imx_fec_dev_t *, ptpv2hdr_t *, mpc_bd_t *)
- void imx_ptp_add_tx_timestamp (imx_fec_dev_t *, ptpv2hdr_t *, mpc_bd_t *)
- int imx_ptp_get_rx_timestamp (imx_fec_dev_t *, ptp_extts_t *)
- int imx_ptp_get_tx_timestamp (imx_fec_dev_t *, ptp_extts_t *)
- int imx_ptp_ioctl (imx_fec_dev_t *, struct ifdrv *)
- void imx_ptp_get_cnt (imx_fec_dev_t *, ptp_time_t *)
- void imx_ptp_set_cnt (imx_fec_dev_t *, ptp_time_t)
- void imx_ptp_set_compensation (imx_fec_dev_t *, ptp_comp_t)
- void imx_ptp_cal (imx_fec_dev_t *imx_fec)
- int imx_tx (imx_fec_dev_t *imx_fec, struct mbuf *m, uint8_t queue)

4.1 Detailed Description

This part describes an Ethernet driver. Start of Ethernet driver is performed during QNX boot sequence via following commands in build file (name of the Ethernet driver is passed like a parameter into io-pkt networking stack):

```
io-pkt-v4 -d fec-imx
waitfor /dev/socket
```

After execution of both commands all available Ethernet controllers will be attached and hooked up to io-pkt networking stack and socket device will be created.

Next step is IP addresses assignment. This can be performed via `ifconfig` command.

```
ifconfig fec0 xxx.xxx.xxx.xxx
ifconfig fec1 xxx.xxx.xxx.xxx
```

Or IP addresses can be assigned automatically with use of DHCP. In that case you have to run DHCP client utility for every Ethernet interface you want to use. IP address assignment via DHCP can take a while.

```
dhcp.client -i fec0
dhcp.client -i fec1
```

After execution of this commands related Ethernet interfaces will be enabled and marks as UP in log of `ifconfig` utility. Output of ifconfig command should look like this(in this example interface fec0 in not connected to network):

```
# ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33192
        inet 127.0.0.1 netmask 0xff000000
fec0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:04:9f:04:01:cc
        media: Ethernet none
        inet 0.0.0.0 netmask 0xff000000 broadcast 255.255.255.255
fec1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:04:9f:04:01:bb
        media: Ethernet autoselect (1000baseT full-duplex)
        status: active
        inet 10.171.65.125 netmask 0xffffff00 broadcast 10.171.65.255
#
```

More detailed information about Ethernet interfaces can be displayed using `nicinfo` utility.

Functionality of the Ethernet interface is possible to briefly check using `ping` utility.

By default Ethernet driver is executed with no input parameters. For possible parameters please see src/hardware/devnp/imx/devnp-fec-imx.use file.

4.2 Function Documentation

4.2.0.0.1 void bsd_mii_initmedia (imx_fec_dev_t * *imx_fec*)

This function is called from imx_attach() in detect.c to hook up to the bsd media structure. Not entirely unlike kissing a porcupine, we must do so carefully, because we do not want to use the bsd mii management structure, because this driver uses the io-net2 lib/drvr mii code.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 281 of file bsd_media.c.

4.2.0.0.2 int bsd_mii_mediachange (struct ifnet * *ifp*)

This is a callback, made by the bsd media code. We passed a pointer to this function during the ifmedia_init() call in bsd_mii_initmedia(). This function is called when someone makes an ioctl into us, we call into the generic ifmedia source, and it make this callback to actually force the speed and duplex, just as if the user had set the cmd line options.

Parameters

ifp	Pointer to a structure containing data shared between Ethernet driver and io-pkt stack.

Returns

Function returns always 1.

Definition at line 164 of file bsd_media.c.

4.2.0.0.3 void bsd_mii_mediastatus (struct ifnet * *ifp,* struct ifmediareq * *ifmr*)

This is a callback, made by the bsd media code. We passed a pointer to this function during the ifmedia_init() call in bsd_mii_initmedia().

Parameters

ifp	Pointer to a structure containing data shared between Ethernet driver and io-pkt stack.
ifmr	Input parameter.

Definition at line 43 of file bsd_media.c.

4.2.0.0.4 void dump_mbuf (struct mbuf * *m,* uint32_t *length*)

Function prints out Tx/Rx packet buffers.

Parameters

m	Pointer to packet buffer.
length	Length of the buffer.

Definition at line 71 of file detect.c.

4.2.0.0.5 void DumpMAC (imx_fec_dev_t * *imx_fec*)

Function prints out actual configuration of ENET registers.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 1084 of file detect.c.

4.2.0.0.6 void DumpPhy (imx_fec_dev_t * *imx_fec*)

Function prints out actual configuration of PHY chip registers.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 1123 of file detect.c.

4.2.0.0.7 void imx_bcm54220_phy_init (imx_fec_dev_t * *imx_fec*)

Specific initialization of Broadcom BCM54220 PHY used on iMX7D board.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 455 of file mii.c.

4.2.0.0.8 void imx_clear_stats (imx_fec_dev_t * *imx_fec*)

Clear statistics. Called from imx_init().

Parameters

imx_fec	Pointer to structure containing data of the ENET device.

Definition at line 122 of file stats.c.

4.2.0.0.9 int imx_detect (void * *dll_hdl,* struct _iopkt_self * *iopkt,* char * *options*)

Function performs attaching of all available Ethernet controllers.

Parameters

dll_hdl	An opaque pointer that identifies the shared module within io-pkt.
iopkt	A structure used by the stack to reference its own internals.
options	The options string passed by the user.

Returns

Execution status

Return values

0	Success, OK.
-1	On error, Could not attach any Ethernet controller.

Definition at line 1384 of file detect.c.

4.2.0.0.10 int imx_enable_interrupt (void * *arg*)

Function enables all Tx interrupts.

Parameters

arg	Pointer to a structure containing data of the ENET device.

Returns

Execution status.

Return values

1	Success.

Definition at line 184 of file event.c.

4.2.0.0.11 int imx_enable_queue (void * *arg*)

Pseudo interrupt for Rx queue.

Parameters

arg	Pointer to a structure containing data of the ENET device.

Returns

Execution status.

Return values

1	Success.

Definition at line 68 of file event.c.

4.2.0.0.12 int imx_get_phy_addr (imx_fec_dev_t * *imx_fec*)

Function finds out ID of connected PHY device.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Returns

ID of connected PHY.

Return values

-1	If no PHY found.

Definition at line 315 of file mii.c.

4.2.0.0.13 void imx_init_phy (imx_fec_dev_t * *imx_fec*)

Final initialization common for Atheros, Kendin, Broadcom and Marvel PHYs.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 390 of file mii.c.

4.2.0.0.14 int imx_ioctl (struct ifnet * *ifp,* unsigned long *cmd,* caddr_t *data*)

Driver's IOCTL function.

Parameters

ifp	Pointer to a structure containing data shared between Ethernet driver and io-pkt stack.
cmd	IOCTL command.
data	Address containing IOCTL data.

Returns

Execution status.

Definition at line 63 of file devctl.c.

4.2.0.0.15 const struct sigevent * imx_isr (void * *arg*, int *iid*)

This is the interrupt handler which directly masks off the hardware interrupt at the ENET hw.

Parameters

arg	Pointer to a structure containing data of the ENET device.
iid	Unused parameter.

Returns

Flag of certain receive interrupt.

Definition at line 106 of file event.c.

4.2.0.0.16 void imx_MDI_MonitorPhy (void * *arg*)

Function is periodically called by io-pkt to probe PHY state and to clean out TX descriptor ring.

Parameters

arg	Pointer to a structure containing data of the ENET device.

Definition at line 264 of file mii.c.

4.2.0.0.17 void imx_mii_callback (void * *handle*, uchar_t *phy*, uchar_t *newstate*)

Callback function executed when PHY link state changes.

Parameters

handle	Pointer to a structure containing data of the ENET device.
phy	Identifier of the PHY on MDIO management bus.
newstate	New link state.

Definition at line 116 of file mii.c.

4.2.0.0.18 uint16_t imx_mii_read (void * *handle*, uint8_t *phy_add*, uint8_t *reg_add*)

Callback function to read PHY registers.

Parameters

handle	Pointer to a structure containing data of the ENET device.
phy_add	Address of PHY on MDIO management bus.
reg_add	Register address.

Returns

Read value.

Definition at line 45 of file mii.c.

4.2.0.0.19 void imx_mii_write (void * *handle,* uint8_t *phy_add,* uint8_t *reg_add,* uint16_t *data*)

Callback function to write data into PHY registers

Parameters

handle	Pointer to a structure containing data of the ENET device.
phy_add	Address of PHY on MDIO management bus.
reg_add	Register address.
data	Data to be written into PHY.

Definition at line 84 of file mii.c.

4.2.0.0.20 int imx_output (struct ifnet * *ifp,* struct mbuf * *m,* struct sockaddr * *dst,* struct rtentry * *rt*)

Function to transmit AVB packets.

Parameters

ifp	Pointer to structure containing data shared between Ethernet driver and io-pkt stack.
m	Packet mbuffer.
dst	Pointer to structure containing destination address.
rt	Pointer to a structure of an entry in the kernel routing table.

Returns

Status of the packet transmission.

Return values

ENOBUFS	No buffer space available.

Definition at line 448 of file transmit.c.

4.2.0.0.21 int imx_process_interrupt (void * *arg,* struct nw_work_thread * *wtp*)

Tx interrupts processing.

Parameters

arg	Pointer to a structure containing data of the ENET device.
wtp	Pointer to new working thread.

Returns

Execution status.

Return values

1	Success.

Definition at line 211 of file event.c.

4.2.0.0.22 int imx_process_queue (void * *arg,* struct nw_work_thread * *wtp*)

Pseudo interrupt for Rx queue.

Parameters

arg	Pointer to a structure containing data of the ENET device.
wtp	Pointer to new working thread.

Returns

Execution status.

Return values

1	Success.

Definition at line 40 of file event.c.

4.2.0.0.23 void imx_ptp_add_rx_timestamp (imx_fec_dev_t * *imx_fec,* ptpv2hdr_t * *ph,* mpc_bd_t * *bd*)

This function inserts RX timestamp into the corresponding buffer (depends on the message type). If the corresponding buffer already full, the timestamp will be inserted into the start of the buffer.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
ph	Pointer to packet header.
bd	pointer to receive buffer descriptor.

Definition at line 259 of file ptp.c.

4.2.0.0.24 void imx_ptp_add_tx_timestamp (imx_fec_dev_t * *imx_fec,* ptpv2hdr_t * *ph,* mpc_bd_t * *bd*)

This function inserts TX timestamp into the corresponding buffer (depends on the message type). If the corresponding buffer already full, the timestamp will be inserted into the start of the buffer.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
ph	Pointer to packet header.
bd	pointer to receive buffer descriptor.

Definition at line 295 of file ptp.c.

4.2.0.0.25 void imx_ptp_cal (imx_fec_dev_t * *imx_fec*)

Timer calibration.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 50 of file ptp.c.

4.2.0.0.26 void imx_ptp_get_cnt (imx_fec_dev_t * *imx_fec,* ptp_time_t * *cnt*)

This function returns the current timer value.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
cnt	Pointer to variable to store current timer value.

Definition at line 389 of file ptp.c.

4.2.0.0.27 int imx_ptp_get_rx_timestamp (imx_fec_dev_t * *imx_fec,* ptp_extts_t * *ts*)

This function searches a timestamp in the corresponding RX buffer, according to message type, sequence id and the source port id.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
ts	Pointer to Rx timestamp structure.

Returns

Flag if time stamp has been found.

Return values

0	If timestamp has not been found.
1	If timestamp has been found.

Definition at line 332 of file ptp.c.

4.2.0.0.28 int imx_ptp_get_tx_timestamp (imx_fec_dev_t * *imx_fec,* ptp_extts_t * *ts*)

This function searches a timestamp in the corresponding TX buffer, according to message type, sequence id and the source port id.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
ts	Pointer to TRx timestamp structure.

Returns

Flag if time stamp has been found.

Return values

0	If timestamp has not been found.
1	If timestamp has been found.

Definition at line 363 of file ptp.c.

4.2.0.0.29 int imx_ptp_ioctl (imx_fec_dev_t * *imx_fec,* struct ifdrv * *ifd*)

This is a PTP IO control function.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
ifd	Driver' IOCTL structure.

Returns

Non zero value if the error has been occured.

Return values

EOK	Success.
ENOENT	No such file or directory.

Return values

EINVAL	Invalid argument.
ENOTTY	Inappropriate I/O control operation.

Definition at line 539 of file ptp.c.

4.2.0.0.30 int imx_ptp_is_eventmsg (struct mbuf * *m,* ptpv2hdr_t ** *ph*)

This function checks the PTP message.

Parameters

m	Pointer to packet buffer.
ph	Pointer to packet header.

Returns

Returns flag If the frame contains the PTP event or not.

Return values

1	Frame contains the PTP event.
0	Frame does not contains the PTP event.

Definition at line 200 of file ptp.c.

4.2.0.0.31 void imx_ptp_set_cnt (imx_fec_dev_t * *imx_fec,* ptp_time_t *cnt*)

This function sets the current timer value.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
cnt	Current timer value.

Definition at line 443 of file ptp.c.

4.2.0.0.32 void imx_ptp_set_compensation (imx_fec_dev_t * *imx_fec,* ptp_comp_t *ptc*)

This function sets the clock compensation.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.
ptc	Compensation value.

Definition at line 464 of file ptp.c.

4.2.0.0.33 int imx_ptp_start (imx_fec_dev_t * *imx_fec*)

This function sets the default values for the counter and resets the timer.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Returns

Execution status.

Return values

0	Success.

Definition at line 128 of file ptp.c.

4.2.0.0.34 void imx_ptp_stop (imx_fec_dev_t * *imx_fec*)

This function resets the timer and turns it off.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 174 of file ptp.c.

4.2.0.0.35 int imx_receive (imx_fec_dev_t * *imx_fec,* struct nw_work_thread * *wtp,* uint8_t *queue*)

Function processes received packet.

Parameters

imx_fec	Pointer to structure containing data of the ENET device.
wtp	Working thread.
queue	Queue ID (0,1 or 2).

Returns

Execution status.

Return values

0	If queue overflowed, interrupts will be disabled until queue will be drained.
1	Packet received successfully.

Definition at line 84 of file receive.c.

4.2.0.0.36 void* imx_rx_thread (void * *arg*)

Receive thread. This thread is waiting for pulse from receive interrupt. Once pulse is received, received bytes will be filled into Rx buffer.

Parameters

arg	Pointer to a structure containing data of the ENET device.

Definition at line 211 of file detect.c.

4.2.0.0.37 void imx_rx_thread_quiesce (void * *arg,* int *die*)

Function sends QUIESCE pulse.

Parameters

arg	Pointer to a structure containing data of the ENET device.
die	Dying flag.

Definition at line 269 of file detect.c.

4.2.0.0.38 void imx_sabreauto_rework (imx_fec_dev_t * *imx_fec*)

Specific initialization of Atheros AR8031 PHY.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 476 of file mii.c.

4.2.0.0.39 void imx_set_multicast (imx_fec_dev_t * *imx_fec*)

Called from imx_init() and imx_ioctl() to calculate the multicast group address hash mask for the current set of multicast addresses.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 40 of file multicast.c.

4.2.0.0.40 int imx_set_tx_bw (imx_fec_dev_t * *imx_fec,* struct ifdrv * *ifd*)

Function sets bandwidth for AVB.

Parameters

imx_fec	Pointer to structure containing data of the ENET device.
ifd	Pointer to structure with NIC identification.

Returns

Execution status.

Return values

EOK	Success.
EINVAL	Invalid argument.

Definition at line 555 of file transmit.c.

4.2.0.0.41 int imx_setup_phy (imx_fec_dev_t * *imx_fec*)

Universal PHY initialization.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Returns

Execution status.

Return values

0	Success.
-1	Error.

Definition at line 349 of file mii.c.

4.2.0.0.42 void imx_speeduplex (imx_fec_dev_t * *imx_fec*)

Function configures ENET hardware according to link parameters detected by PHY and read via MII bus.

Parameters

imx_fec	Pointer to a structure containing data of the ENET device.

Definition at line 1255 of file detect.c.

4.2.0.0.43 void imx_start (struct ifnet * *ifp*)

Start transmitting.

Parameters

ifp	Pointer to structure containing data shared between Ethernet driver and io-pkt stack.

Definition at line 323 of file transmit.c.

4.2.0.0.44 void imx_transmit_complete (imx_fec_dev_t * *imx_fec,* uint8_t *queue*)

Process completed tx descriptors.

Parameters

imx_fec	Pointer to structure containing data of the ENET device.
queue	Index of the packet descriptors queue.

Definition at line 369 of file transmit.c.

4.2.0.0.45 int imx_tx (imx_fec_dev_t * *imx_fec,* struct mbuf * *m,* uint8_t *queue*)

Function transmits packet.

Parameters

imx_fec	Pointer to structure containing data of the ENET device.
m	Pointer to mbuf with packet to send.
queue	Index into queue of descriptors.

Returns

Execution status.

Return values

EOK	Success.
EINVAL	Invalid argument.
ENOMEM	Not enough space.

Definition at line 115 of file transmit.c.

4.2.0.0.46 void imx_update_stats (imx_fec_dev_t * *imx_fec*)

Function reads ENET MIB registers and stores data into device structure. Called from imx_ioctl() or imx_process↩ _interrupt().

Parameters

imx_fec	Pointer to structure containing data of the ENET device.

Definition at line 88 of file stats.c.

Chapter 5

Watchodog refresh utility (wdtkick)

This part describes the Watchodog refresh utility.

Functions

- int main (int argc, char ∗argv[ ])

5.1 Detailed Description

This part describes the Watchodog refresh utility. The Watchdog Timer (WDOG) protects against system failures by providing a method by which to escape from unexpected events or programming errors. Once the WDOG is activated, it must be serviced by the software on a periodic basis. If servicing does not take place, the timer times out. Upon timeout, the WDOG asserts the internal system reset signal.

To enable Watchdog timer module, run the BSP startup with **-W** option. This is done from build file by following command:

```
startup-imx7x-sabre -m -v -n0 -e -W
```

When -W option is used, clock gate for WDOG1 peripheral will be enabled and WDOG1 will be started with 30 second kick-off period by default. Use of WDOG1 and default period are hard coded in the startup code and cannot be affected by any other option. If you want to use another WDOG (2,3,4) peripheral, you have to modify the startup code accordingly.

Watchdog refresh utility should be started from the build file by following command:

```
wdtkick
```

Implementation of an utility is very simple. After parsing of possible command line options new thread will be created and program will enter endless loop. In this loop a service routine will be performed to refresh WDOG timer and after that the thread will be suspended for given length of time. If some process with higher priority exhausts all CPU resources, the thread will be never woken up, service routine will not be executed and the CPU reset will occur because of Watchdog timer times out.

By default Watchdog refresh utility is executed with no input parameters. List of possible options can be found in src/hardware/support/wdtkick/wdtkick.use file.

5.2 Function Documentation

5.2.0.0.1 int main (int *argc,* char * *argv[ ]*)

Main function of the Watchdog refresh utility.

Parameters

argc	Count of the command line arguments.
argv	Array of pointers to command line arguments.

Returns

Execution status.

Return values

EXIT_SUCCESS	If everything is OK.
EXIT_FAILURE	In case of any error.

Definition at line 57 of file main.c.

Chapter 6

SAI - Synchronous Audio Interface driver (deva-ctrl-mx-sai_wm8960.so)

This part describes the SAI audio driver library deva-ctrl-mx-sai_wm8960.so.

Modules

- WM8960 codec driver

  This part describes the WM8960 external codec driver.

Data Structures

- struct imx_stream_dma
- struct imx_stream_pcm
- struct imx_stream
- struct imx_sai_xfer_config
- struct imx_sai_data
- struct imx_card
- struct my_pulse_struct

Typedefs

- typedef enum imx_stream_status imx_stream_status_t
- typedef struct imx_stream_dma imx_stream_dma_t
- typedef struct imx_stream_pcm imx_stream_pcm_t
- typedef enum imx_sai_mode imx_sai_mode_t
- typedef enum imx_sai_protocol imx_sai_protocol_t
- typedef enum imx_sai_sync_mode imx_sai_sync_mode_t
- typedef struct imx_sai_xfer_config imx_sai_xfer_config_t
- typedef struct imx_sai_data imx_sai_data_t

Enumerations

Functions

- int imx_sai_set_clock_rate (struct imx_card *imx, int rate, imx_txrx_index_t idx)
- int32_t imx_capabilities (struct imx_card *imx, ado_pcm_t *pcm, snd_pcm_channel_info_t *info)
- int32_t imx_playback_aquire (struct imx_card *imx, struct imx_stream **pc, ado_pcm_config_t *config, ado_pcm_subchn_t *subchn, uint32_t *why_failed)
- int32_t imx_playback_release (struct imx_card *imx, struct imx_stream *pc, ado_pcm_config_t *config)
- int32_t imx_capture_aquire (struct imx_card *imx, struct imx_stream **pc, ado_pcm_config_t *config, ado_pcm_subchn_t *subchn, uint32_t *why_failed)
- int32_t imx_capture_release (struct imx_card *imx, struct imx_stream *pc, ado_pcm_config_t *config)
- int32_t imx_prepare (struct imx_card *imx, struct imx_stream *pc, ado_pcm_config_t *config)
- int32_t imx_playback_trigger (struct imx_card *imx, struct imx_stream *pc, uint32_t cmd)
- int32_t imx_capture_trigger (struct imx_card *imx, struct imx_stream *pc, uint32_t cmd)
- void imx_play_pulse_hdlr (struct imx_card *imx, struct sigevent *event)
- void imx_cap_pulse_hdlr (struct imx_card *imx, struct sigevent *event)
- int imx_sai_init (struct imx_card *imx)
- void ctrl_version (int *major, int *minor, char *date)
- void imx_sai_config_default_protocol_flags (struct imx_card *imx)
- int imx_parse_commandline (struct imx_card *imx, char *args)
- int ctrl_init (struct imx_card **hw_context, ado_card_t *card, char *args)
- int ctrl_destroy (struct imx_card *imx)
- int my_detach_pulse (void **x)
- int my_attach_pulse (void **x, struct sigevent *event, void(*handler)(struct imx_card *hw_context, struct sigevent *event), struct imx_card *hw_context)

6.1 Detailed Description

This part describes the SAI audio driver library deva-ctrl-mx-sai_wm8960.so. The SAI audio driver library is shared library for the io-audio manager. The io-audio manager provides support for dynamically loaded audio-driver modules. More information about io-audio manager can be found on QNX website help.

The i.MX7D Sabre board has on-board WM8960 codec connected to the SAI1 peripheral. Codec is configured as Slave and SAI peripheral is configured as Master with Master clock (MCLK) generated internally by Audio PLL. MCLK frequency is fixed to 12.288 MHz for sample rates 8, 16, 32, 48 KHz. Higher sample rates are not supported by WM8960 codec.

The SAI audio driver library does not configure clock sources and does not configure I2S pins directly. These are configured in BSP startup code.

The SAI audio driver library supports various command line options documented in src/hardware/deva/ctrl/mx/deva-ctrl-mx.use file. Content of this file is also accessible in running QNX system by executing use command:

```
# use deva-ctrl-mx-sai_wm8960.so
```

The SAI audio driver library is loaded by default when system starts. Manually can be loaded by executing command:

```
# io-audio -d mx-sai_wm8960 clk_mode=i2s_master,sys_clk=12288000
```

After loading the library a new device /dev/snd is created:

```
# ls /dev/snd
controlC0          pcmC0D0c           pcmC0D1p           pcmPreferredp
mixerC0D0          pcmC0D0p           pcmPreferredc
```

i.MX7 sabre board has one stereo audio output connected to audio JACK connector (Headphone) and Class-D amplifier stereo output connected to a speaker connector (Front). Both analog outputs are connected internally to stereo DAC in the WM8960 codec. The board has also one microphone input connected to audio JACK connector. Outputs and inputs have volume and mute control. These are accessible over `mix_ctl` application. This application is included in the BSP.

For list of supported Playback and Capture groups execute command:

```
# mix_ctl
"Master",0                              - Playback Group
"Headphone",0                           - Playback Group
"Front",0                               - Playback Group
"Mic In",0                              - Capture Group
"PCM Mixer",0                           - Playback Group
```

To control the Master group volume execute the command:

```
# mix_ctl group "Master" volume=240
"Master",0 - Playback Group
        Capabilities -  Volume Mute
        Channels - Front-Left Front-Right
        Volume Range - minimum=0, maximum=254
        Channel 0 Front-Left   - 240 ( 94%)
        Channel 1 Front-Right  - 240 ( 94%)
```

To enable Master group mute execute the command:

```
# mix_ctl group "Master" mute=1
"Master",0 - Playback Group
        Capabilities -  Volume Mute
        Channels - Front-Left Front-Right
        Volume Range - minimum=0, maximum=254
        Channel 0 Front-Left   - 240 ( 94%) Muted
        Channel 1 Front-Right  - 240 ( 94%) Muted
```

i.MX7 sabre BSP contains binaries for audio WAV files playback and capturing. WAV file can be played by `wave` command and a new WAV file can be recorded from MIC input by `waverec` command.

Example of recording audio data into /tmp/ directory in Mono mode with sample rate 48 kHz, capturing data for 20 seconds:

```
# waverec -m -t20 -r48000 /tmp/foo.wav
SampleRate = 48000, Channels = 1, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 8192
Total Frags 8
Rate 48000
Mixer Pcm Group [Mic In]
```

Example of audio WAV file playback:

```
# wave /tmp/foo.wav
SampleRate = 48000, Channels = 1, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 2048
Total Frags 64
Rate 48000
Voices 1
Mixer Pcm Group [Wave playback channel]
```

NOTE: The io-audio manager has ability to capture and playback with sample rates unsupported by hardware because of software resampler (resamples unsupported sample rates to sample rate supported by hardware).

6.2 Typedef Documentation

6.2.0.0.1 typedef struct imx_sai_data imx_sai_data_t

IMX SAI data structure.

6.2.0.0.2 typedef enum imx_sai_mode imx_sai_mode_t

IMX SAI mode.

6.2.0.0.3 typedef enum imx_sai_protocol imx_sai_protocol_t

IMX SAI protocol.

6.2.0.0.4 typedef enum imx_sai_sync_mode imx_sai_sync_mode_t

IMX SAI RX TX synchronization mode.

6.2.0.0.5 typedef struct imx_sai_xfer_config imx_sai_xfer_config_t

IMX SAI transfer configuration structure.

6.2.0.0.6 typedef struct imx_stream_dma imx_stream_dma_t

IMX stream DMA data.

6.2.0.0.7 typedef struct imx_stream_pcm imx_stream_pcm_t

PCM stream data structure.

6.2.0.0.8 typedef enum imx_stream_status imx_stream_status_t

IMX stream status.

6.3 Enumeration Type Documentation

6.3.0.0.1 enum imx_sai_mode

IMX SAI mode.

Enumerator

IMX_SAI_MASTER SAI device is master and ext. codec is slave.

IMX_SAI_SLAVE SAI device is slave and ext. codec is master.

Definition at line 86 of file imx_sai_dll.h.

6.3.0.0.2 enum imx_sai_protocol

IMX SAI protocol.

Enumerator

IMX_SAI_PROTOCOL_I2S I2S protocol.

IMX_SAI_PROTOCOL_PCM PCM protocol.

Definition at line 92 of file imx_sai_dll.h.

6.3.0.0.3 enum imx_sai_sync_mode

IMX SAI RX TX synchronization mode.

Enumerator

IMX_SAI_SYNC_RX_WITH_TX Rx uses Tx frame sync and bit clock.

IMX_SAI_ASYNC Rx and Tx are asynchronous.

Definition at line 98 of file imx_sai_dll.h.

6.3.0.0.4 enum imx_stream_status

IMX stream status.

Enumerator

IMX_STREAM_STOP Stream is stopped.

IMX_STREAM_ACQUIRE Stream is acquired.

IMX_STREAM_GO Stream is running.

Definition at line 52 of file imx_sai_dll.h.

6.4 Function Documentation

6.4.0.0.1 int ctrl_destroy (struct imx_card * *imx*)

Entry point called when card is unmounted.

Parameters

imx	IMX SAI hardware context structure.

Returns

Execution status.

Definition at line 1597 of file imx_sai_dll.c.

6.4.0.0.2 int ctrl_init (struct imx_card ** *hw_context,* ado_card_t * *card,* char * *args*)

Entry point of IMX SAI audio driver library.

Called when io-audio loads IMX SAI HW DLL.

Parameters

hw_context	Pointer which store IMX SAI hardware context structure.
card	Pointer to an internal card structure.
args	Any command-line arguments.

Returns

Execution status.

Definition at line 1389 of file imx_sai_dll.c.

6.4.0.0.3 void ctrl_version (int * *major,* int * *minor,* char * *date*)

Initializes SAI driver library version.

Parameters

major	Major version.
minor	Minor version.
date	Driver date.

Definition at line 874 of file imx_sai_dll.c.

6.4.0.0.4 void imx_cap_pulse_hdlr (struct imx_card * *imx,* struct sigevent * *event*)

Pulse handler (for capture).

Sends a signal that the current fragment of a subchannel has been completed by the DMA engine.

Parameters

imx	IMX SAI hardware context structure.
event	Pointer to event structure.

Definition at line 610 of file imx_sai_dll.c.

6.4.0.0.5 int32_t imx_capabilities (struct imx_card * *imx,* ado_pcm_t * *pcm,* snd_pcm_channel_info_t * *info*)

Gets IMX SAI capabilities.

This function is used to return to the client the capabilities of the device at this instant. When the device was created, its static capabilities were passed in as an argument; however, if a number of subchannels are already running, the device may no longer have the ability to support those capabilities.

Parameters

imx	IMX SAI hardware context structure.
pcm	PCM device structure.
info	Structure to fill. Contains info about IMX SAI capabilities.

Returns

Execution status.

Definition at line 86 of file imx_sai_dll.c.

6.4.0.0.6 int32_t imx_capture_aquire (struct imx_card * *imx,* struct imx_stream ** *pc,* ado_pcm_config_t * *config,* ado_pcm_subchn_t * *subchn,* uint32_t * *why_failed*)

Called when a client attempts to open a capture PCM stream.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
config	Contains all the parameters about how the channel is to be set up.
subchn	Structure for the subchannel.
why_failed	A pointer to the variable for returning detailed info why failed.

Returns

Execution status.

Definition at line 264 of file imx_sai_dll.c.

6.4.0.0.7 int32_t imx_capture_release (struct imx_card * *imx,* struct imx_stream * *pc,* ado_pcm_config_t * *config*)

Called by upper layer when client closes its connection to the device.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
config	Contains all the parameters about how the channel was set up.

Returns

Execution status.

Definition at line 345 of file imx_sai_dll.c.

6.4.0.0.8 int32_t imx_capture_trigger (struct imx_card * *imx,* struct imx_stream * *pc,* uint32_t *cmd*)

Called by upper layer to start or stop capture subchannel.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
cmd	Command which specifies GO or STOP trigger.

Returns

Execution status.

Definition at line 472 of file imx_sai_dll.c.

6.4.0.0.9 int imx_parse_commandline (struct imx_card * *imx,* char * *args*)

Parse command line parameters.

Parameters

imx	SAI hardware context structure.
args	Pointer to array of command line arguments.

Returns

Execution status.

Definition at line 948 of file imx_sai_dll.c.

6.4.0.0.10 void imx_play_pulse_hdlr (struct imx_card * *imx,* struct sigevent * *event*)

Pulse handler (for playback).

Sends a signal that the current fragment of a subchannel has been completed by the DMA engine.

Parameters

imx	IMX SAI hardware context structure.
event	Pointer to event structure.

Definition at line 593 of file imx_sai_dll.c.

6.4.0.0.11 int32_t imx_playback_aquire (struct imx_card * *imx,* struct imx_stream ** *pc,* ado_pcm_config_t * *config,* ado_pcm_subchn_t * *subchn,* uint32_t * *why_failed*)

Called when a client attempts to open a playback PCM stream.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
config	Contains all the parameters about how the channel is to be set up.
subchn	Structure for the subchannel.
why_failed	A pointer to the variable for returning detailed info why failed.

Returns

Execution status.

Definition at line 153 of file imx_sai_dll.c.

6.4.0.0.12 int32_t imx_playback_release (struct imx_card * *imx,* struct imx_stream * *pc,* ado_pcm_config_t * *config*)

Called by upper layer when client closes its connection to the device.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
config	Contains all the parameters about how the channel was set up.

Returns

Execution status.

Definition at line 240 of file imx_sai_dll.c.

6.4.0.0.13 int32_t imx_playback_trigger (struct imx_card * *imx,* struct imx_stream * *pc,* uint32_t *cmd*)

Called by upper layer to start or stop playback subchannel.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
cmd	Command which specifies GO or STOP trigger.

Returns

Execution status.

Definition at line 381 of file imx_sai_dll.c.

6.4.0.0.14 int32_t imx_prepare (struct imx_card * *imx,* struct imx_stream * *pc,* ado_pcm_config_t * *config*)

Called by upper layer to prepare hardware before it's started up.

Parameters

imx	IMX SAI hardware context structure.
pc	IMX SAI PCM subchannel context structure.
config	Contains all the parameters about how the channel is to be set up.

Returns

Execution status.

Definition at line 366 of file imx_sai_dll.c.

6.4.0.0.15 void imx_sai_config_default_protocol_flags (struct imx_card * *imx*)

This function configures the various protocol specific flags.

Parameters

imx	IMX SAI hardware context structure.

Definition at line 911 of file imx_sai_dll.c.

6.4.0.0.16 int imx_sai_init (struct imx_card * *imx*)

Initializes SAI hardware.

Parameters

imx	IMX SAI hardware context structure.

Returns

Execution status.

Definition at line 771 of file imx_sai_dll.c.

6.4.0.0.17 int imx_sai_set_clock_rate (struct imx_card * *imx,* int *rate,* imx_txrx_index_t *idx*)

Sets SAI clock rate according to required sample rate etc.

Parameters

imx	IMX SAI hardware context structure.
rate	Sample rate to be set.
idx	Specifies which part of SAI, TX or RX.

Returns

Execution status.

Definition at line 626 of file imx_sai_dll.c.

6.4.0.0.18 int my_attach_pulse (void ** *x,* struct sigevent * *event,* void(*)(struct imx_card *hw_context, struct sigevent *event) *handler,* struct imx_card * *hw_context*)

Attach a new pulse handler.

Parameters

x	Handle to pulse structure.
event	Pointer to event structure.
handler	Pointer to handler to be attached.
hw_context	Audio hardware context structure.

Returns

Execution status.

Definition at line 96 of file pulse.c.

6.4.0.0.19 int my_detach_pulse (void ** *x*)

Detach a pulse handler.

Parameters

x	Handle to pulse structure.

Returns

Execution status.

Definition at line 142 of file pulse.c.

6.5 WM8960 codec driver

This chapter describes the WM8960 external codec driver.

Data Structures

- struct wm8960_context

Macros

- #define WM8960_AVDD 3300
- #define WM8960_ANALOG_BIAS_THRESH 3000
- #define WM8960_SLAVE_ADDR (0x34 >> 1)

Typedefs

- typedef struct wm8960_context wm8960_context_t

Functions

- int codec_set_rate (HW_CONTEXT_T *mx)
- int codec_mixer (ado_card_t *card, HW_CONTEXT_T *mx)

6.5.1 Detailed Description

This chapter describes the WM8960 external codec driver.

The WM8960 codec driver is a part of deva audio driver library. In the i.MX7D Sabre BSP is a part of SAI driver library deva-ctrl-mx-mx7_wm8960.so. The WM8960 codec driver uses I2C resource manager driver to configure external WM8960 codec.

The WM8960 codec driver registers and maps mixer elements. Each registered mixer element is mapped to some WM8960 hardware functionality. For example Mute elements are mapped to WM8960 mute functionality etc. The driver creates 3 playback groups and one Capture group. See `mix_ctl` application output:

```
# mix_ctl
"Master",0                              - Playback Group
"Headphone",0                           - Playback Group
"Front",0                               - Playback Group
"Mic In",0                              - Capture Group
"PCM Mixer",0                           - Playback Group
```

A list of groups:

- Master - controls DAC output volume and mute
- Headphone - controls Headphone out volume and mute

- Front - controls Speaker out volume and mute
- Mic In - controls Microphone input volume and mute
- PCM Mixer - software mixer for playback

For each playback and capture group is possible to set volume and disable or enable mute function.

For example to control the Master group volume execute the command:

```
# mix_ctl group "Master" volume=240
"Master",0 - Playback Group
        Capabilities -  Volume Mute
        Channels - Front-Left Front-Right
        Volume Range - minimum=0, maximum=254
        Channel 0 Front-Left   - 240 ( 94%)
        Channel 1 Front-Right  - 240 ( 94%)
```

To enable Master group mute execute the command:

```
# mix_ctl group "Master" mute=1
"Master",0 - Playback Group
        Capabilities -  Volume Mute
        Channels - Front-Left Front-Right
        Volume Range - minimum=0, maximum=254
        Channel 0 Front-Left   - 240 ( 94%) Muted
        Channel 1 Front-Right  - 240 ( 94%) Muted
```

6.5.2 Macro Definition Documentation

6.5.2.0.1 #define WM8960_ANALOG_BIAS_THRESH 3000

Defines threshold in mV for analog bias increasing. Bias is increased when Analog Power Supply voltage level is below or equal to this threshold.

Definition at line 50 of file wm8960.h.

6.5.2.0.2 #define WM8960_AVDD 3300

WM8960 Analog Power Supply voltage level in mV

Definition at line 43 of file wm8960.h.

6.5.2.0.3 #define WM8960_SLAVE_ADDR (0x34 >> 1)

WM8960 I2C 7-bit slave address

Definition at line 54 of file wm8960.h.

6.5.3 Typedef Documentation

6.5.3.0.1 typedef struct wm8960_context wm8960_context_t

WM8960 mixer context structure

6.5.4 Function Documentation

6.5.4.0.1 int codec_mixer (ado_card_t * *card,* HW_CONTEXT_T * *mx*)

Called by SAI controller to initialize WM8960 codec.

Parameters

card	Card context structure.
mx	SAI context structure.

Returns

Execution status.

Definition at line 1447 of file wm8960.c.

6.5.4.0.2 int codec_set_rate (HW_CONTEXT_T * *mx*)

Called by SAI controller to configure sample rate on fly.

Parameters

mx	SAI context structure.

Returns

Execution status.

Definition at line 1426 of file wm8960.c.

Chapter 7

ETFS low level driver (fs-etfs-imx-micron)

This driver implements low level layer for `ETFS (Embedded transaction file system)` and is intended for NAND memories with 4096 bytes page size and 224 bytes spare area. There are supported 4 sub-pages within "standard" page due to design of BCH peripheral, for details please see picture below.
This driver is using HW (BCH) 24-bit ECC per 1032 bytes of data.
The ETFS driver does not configure clock sources and does not configure GPMI pins directly. These are configured in BSP startup code.

Modules

- Chip interface

  This part describes the NAND controller.

Functions

- int devio_options (struct etfs_devio ∗dev, char ∗optstr)
- int devio_init (struct etfs_devio ∗dev)
- int devio_readcluster (struct etfs_devio ∗dev, unsigned cluster, uint8_t ∗buf, struct etfs_trans ∗trp)
- int devio_readtrans (struct etfs_devio ∗dev, unsigned cluster, struct etfs_trans ∗trp)
- int devio_postcluster (struct etfs_devio ∗dev, unsigned cluster, uint8_t ∗buf, struct etfs_trans ∗trp)
- int devio_eraseblk (struct etfs_devio ∗dev, unsigned blk)
- int devio_sync (struct etfs_devio ∗dev)

7.1 Detailed Description

This driver implements low level layer for `ETFS (Embedded transaction file system)` and is intended for NAND memories with 4096 bytes page size and 224 bytes spare area. There are supported 4 sub-pages within "standard" page due to design of BCH peripheral, for details please see picture below.
This driver is using HW (BCH) 24-bit ECC per 1032 bytes of data.
The ETFS driver does not configure clock sources and does not configure GPMI pins directly. These are configured in BSP startup code.

NOTE: This driver currently supports only Micron MT29F8G08xxxxx devices.

The ETFS driver supports various command line options documented in src/hardware/etfs/nand4096/etfs.use file. Content of this file is also accessible in running QNX system by executing `use` command.

```
use fs-etfs-imx-micron
```

ETFS file system can be manually loaded by executing command below (for additional parameters please see use file options). After driver start dev/etfs1 and dev/etfs1 devices are created. First time use "fs-etfs-imx-micron -e". This erases the NAND flash. Later on use start the driver with auto-mount "fs-etfs-imx-micron -m /fs/etfs". New partition will appear as /fs/etfs.

```
fs-etfs-imx-micron -e -f
```

To see mounted partition size use `df` command:

```
df -h
```

```
 Sub-page ecc area in 4096 page + 224 spare on DEVICE
 24-bit ecc with 14 GF (24+14/8)=42
 ____________________________________________________________________
|Data      |Ecc||Data      |Ecc||Data      |Ecc||Data    * |Ecc|Unused |
|1024+8    |42 ||1024+8    |42 ||1024+8    |42 ||1024+8 * |42 |24     |
|__________|___||__________|___||__________|___||________*_|___|_______|
        0             1             2             3    BBI
 BBI is in data area and is swapped to fs meta area. Data byte vice versa.
```

7.2 Function Documentation

7.2.0.0.1 int devio_eraseblk (struct etfs_devio * *dev,* unsigned *blk*)

Erase a block.

Parameters

dev	ETFS handle.
blk	Block to erase (blk is 0,1,2,...).

Returns

Execution status (ETFS_TRANS_XXX).

Definition at line 587 of file devio.c.

7.2.0.0.2 int devio_init (struct etfs_devio * *dev*)

Initialize the part and stuff physical parameters for the part.

Parameters

dev	ETFS handle.

Returns

EOK always.

Definition at line 147 of file devio.c.

7.2.0.0.3 int devio_options (struct etfs_devio * *dev,* char * *optstr*)

Process device specific options (if any). This is always called before any access to the part. It is called by the -D option to the file system. If no -D option is given, this function will still be called with "" for optstr.

Parameters

dev	ETFS handle.
optstr	Driver string options.

Return values

EOK	D options processed successfully.
-1	error in peripheral memory space allocation.
EINVAL	Invalid -D sub-option.

Definition at line 67 of file devio.c.

7.2.0.0.4 int devio_postcluster (struct etfs_devio * *dev,* unsigned *cluster,* uint8_t * *buf,* struct etfs_trans * *trp*)

Post a cluster of data. Set crc for both the spare area and the entire page (data + spare). The passed buffer "buf" is larger than the cluster size. It can hold PAGESIZE bytes. This is for convenience writing data to the device and calculating the crc. The work area after clustersize bytes is ignored by the caller.

Parameters

dev	Device structure.
cluster	Current cluster (page).
buf	Data buffer.
trp	Transaction pointer.

Returns

Execution status (ETFS_TRANS_XXX).

Definition at line 499 of file devio.c.

7.2.0.0.5 int devio_readcluster (struct etfs_devio * *dev,* unsigned *cluster,* uint8_t * *buf,* struct etfs_trans * *trp*)

Read a cluster of data. Verify crc for both the spare area and the entire page (data + spare). The passed buffer "buf" is larger than the cluster size. It can hold PAGESIZE bytes. This is for convenience when reading data from the device and calculating the crc. The work area after clustersize bytes is ignored by the caller.

Parameters

dev	ETFS handle.
cluster	Cluster (page) to read.
buf	Buffer to fill.
trp	Transaction pointer.

Returns

Execution status (ETFS_TRANS_XXX).

Definition at line 343 of file devio.c.

7.2.0.0.6 int devio_readtrans (struct etfs_devio * *dev,* unsigned *cluster,* struct etfs_trans * *trp*)

Read the spare area of a page (not the data) to return transaction information. This called is used heavily on startup to process the transactions. It is a cheaper call than devio_readcluster() since it reads less data and has a smaller checksum to calculate.

Parameters

dev	ETFS handle.
cluster	Cluster (page) to read.
trp	Transaction pointer.

Returns

Exectution status (ETFS_TRANS_XXX).

Definition at line 412 of file devio.c.

7.2.0.0.7 int devio_sync (struct etfs_devio * *dev*)

Called to allow the driver to flush any cached data that has not be written to the device. The NAND class driver does not need it.

Parameters

dev	ETFS handle.

Returns

-1 always

Definition at line 629 of file devio.c.

7.3 Chip interface

This chapter describes the NAND controller.

Modules

- DMA

  This chapter describes the DMA API of the NAND controller.
- ECC

  This chapter describes the ECC API of the NAND controller.
- PIO

  This chapter describes the PIO API of the NAND controller.

Data Structures

- struct _apbh_dma_t
- struct _apbh_dma_gpmi1_t
- struct _apbh_dma_gpmi3_t
- struct _apbh_dma_gpmi5_t
- struct _dma_blk_erase_t
- struct _dma_programEcc_t
- struct _dma_programRaw_t
- struct _dma_readEcc_device_t
- struct _dma_readRaw_device_t
- struct _NAND_dma_read_status_device_t
- struct _dma_reset_device_t
- struct _dma_read_id_device_t
- struct _chipio_t

Macros

- #define NAND_DMA_WAIT4RDY_CMD

  APBH DMA Macro for Wait4Ready command.
- #define NAND_DMA_WAIT4RDY_PIO(u32ChipSelect)

  GPMI PIO DMA Macro for Wait4Ready command.
- #define NAND_DMA_TXDATA_CMD(TransferSize, Semaphore, CommandWords, Wait4End, Cmd)

  APBH DMA Macro for Transmit Data command. Transfer TransferSize bytes with DMA. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Lock the NAND while waiting for this DMA chain to complete. Decrement semaphore if this is the last part of the chain. Another descriptor follows this one in the chain. This DMA is a read from System Memory - write to device.
- #define NAND_DMA_TXDATA_PIO(u32ChipSelect, TransferSize)

  GPMI PIO DMA Macro for Transmit Data command. Setup transfer as a write. Transfer NumBitsInWord bits per DMA cycle. Lock CS during this transaction. Select the appropriate chip. Address lines need to specify Data transfer (0b00) Transfer TransferSize - NumBitsInWord values.
- #define NAND_DMA_SENSE_CMD(SenseSemaphore)

  APBH DMA Macro for Sense command. Transfer no Bytes with DMA. Transfer no Words to PIO. Don't lock the NAND while waiting for Ready to go high. Decrement semaphore if this is the last part of the chain. Another descriptor follows this one in the chain.
- #define NAND_DMA_RX_CMD_ECC(TransferSize, Semaphore)

APBH DMA Macro for Read Data command with ECC. Receive TransferSize bytes with DMA. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Decrement semaphore if this is the last part of the chain. Unlock the NAND after this DMA chain completes. Another descriptor follows this one in the chain. No DMA transfer here; the ECC8 block becomes the bus master and performs the memory writes itself instead of the DMA.

- #define NAND_DMA_RX_NO_ECC_CMD(TransferSize, Semaphore)

  APBH DMA Macro for Receive Data with no ECC command. Receive TransferSize bytes with DMA but no ECC. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Decrement semaphore if this is the last part of the chain. Unlock the NAND after this DMA chain completes. Another descriptor follows this one in the chain. This DMA is a write to System Memory - read from device.
- #define NAND_DMA_RX_PIO(u32ChipSelect, TransferSize)

  GPMI PIO DMA Macro for Receive command. Setup transfer as a READ. Transfer NumBitsInWord bits per DMA cycle. Select the appropriate chip. Address lines need to specify Data transfer (0b00) Transfer TransferSize - NumBitsInWord values.
- #define NAND_DMA_COMMAND_CMD(TransferSize, Semaphore, NandLock, CmdWords)

  APBH DMA Macro for sending NAND Command sequence. Transmit TransferSize bytes to DMA. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Decrement semaphore if this is the last part of the chain. Lock the NAND until the next chain. Another descriptor follows this one in the chain. This DMA is a read from System Memory - write to device.
- #define NAND_DMA_COMMAND_PIO(u32ChipSelect, TransferSize, AssertCS)

  GPMI PIO DMA Macro when sending a command. Setup transfer as a WRITE. Transfer NumBitsInWord bits per DMA cycle. Lock CS during and after this transaction. Select the appropriate chip. Address lines need to specify Command transfer (0b01) Increment the Address lines if AddrIncr is set. Transfer TransferSize - NumBitsInWord values.
- #define NAND_DMA_ECC_PIO(EnableDisable) (IMX_GPMI_ECCCTRL_ENABLE_ECC(EnableDisable))

  GPMI PIO DMA Macro for disabling ECC during this write.
- #define NAND_DMA_ECC_CTRL_PIO(EccBufferMask, decode_encode_size)

  GPMI PIO DMA Macro sequence for ECC decode. Setup READ transfer ECC Control register. Setup for ECC Decode, 4 Bit. Enable the ECC block The ECC Buffer Mask determines which fields are corrected.

Typedefs

- typedef struct _apbh_dma_t apbh_dma_t
- typedef struct _apbh_dma_gpmi1_t apbh_dma_gpmi1_t
- typedef struct _apbh_dma_gpmi3_t apbh_dma_gpmi3_t
- typedef struct _apbh_dma_gpmi5_t apbh_dma_gpmi5_t
- typedef struct _dma_blk_erase_t dma_blk_erase_t
- typedef struct _dma_programEcc_t dma_programEcc_t
- typedef struct _dma_programRaw_t dma_programRaw_t
- typedef struct _dma_readEcc_device_t dma_readEcc_device_t
- typedef struct _dma_readRaw_device_t dma_readRaw_device_t
- typedef struct _NAND_dma_read_status_device_t dma_read_status_t
- typedef struct _dma_reset_device_t dma_reset_device_t
- typedef struct _dma_read_id_device_t dma_read_id_device_t
- typedef struct _chipio_t chipio

Functions

- void create_blockErase_descriptor (dma_blk_erase_t *superStruct, unsigned page)
- void create_readEcc_descriptor (dma_readEcc_device_t *superStruct, unsigned page, void *ret_data)
- void create_readRaw_descriptor (dma_readRaw_device_t *superStruct, uint32_t data_size, unsigned page, void *ret_data)
- void create_writeEcc_descriptor (dma_programEcc_t *superStruct, unsigned page, void *write_data, uint32_t data_size)

- void create_writeRaw_descriptor (dma_programRaw_t *superStruct, unsigned page, void *write_data, uint32_t data_size)
- void create_readStatus2_descriptor (dma_read_status_t *superStruct, uint8_t *memory_status, unsigned page)
- void create_readStatus_descriptor (dma_read_status_t *superStruct, uint8_t *memory_status)
- void create_readId_descriptor (dma_read_id_device_t *superStruct, uint8_t *ret_raw_id)
- void create_reset_descriptor (dma_reset_device_t *superStruct)
- int nand_init (chipio *cio)
- int run_dma (apbh_dma_t *dma, chipio *chipio, uint8_t wait_flag, uint32_t phy_addr)
- bool is_dma_active (chipio *chipio)
- int nand_wait_busy (chipio *cio, uint32_t time_out, uint8_t chip_select)
- void device_to_nfc (uint8_t *parsed_data, uint8_t *raw_data)
- void nfc_to_device (uint8_t *device_data, uint8_t *nfc_data, uint8_t ecc_size)

Internal NFC error codes

- #define **NAND_EOK** 0x55AA
- #define **NAND_EIO** 0x3
- #define **NAND_DMA_EIO** 0x33

Internal DMA flags - intended for run_dma method

- #define **NAND_DMA_GPMI_TRANS** 0x1
- #define **NAND_DMA_BCH_TRANS** 0x2

Address size description

- #define **NAND_COLUMN_ADDRESS_CYCLES** 2
- #define **NAND_ROW_ADDRESS_CYCLES** 3
- #define **NAND_ADDRESS_CYCLES** (NAND_COLUMN_ADDRESS_CYCLES + NAND_ROW_ADDRES↩ S_CYCLES)

Row and Column manipulation macros

- #define **NAND_ADDR_COL1**(addr) ((addr) & 0xff)
- #define **NAND_ADDR_COL2**(addr) (((addr) & 0x1f00) >> 8)
- #define **NAND_ADDR_ROW1**(page) ((page) & 0xff)
- #define **NAND_ADDR_ROW2**(page) (((page) & 0xff00) >> 8)
- #define **NAND_ADDR_ROW3**(page) (((page) & 0x70000) >> 16)

Reset command description

- #define **NANDCMD_RESET** 0xFF
- #define **NANDCMD_RESET_SIZE** 1

Status command description

- #define **NANDCMD_READ_STATUS_SIZE** 1
- #define **NANDCMD_READ_STATUS** 0x70
- #define **NANDCMD_READ_STATUS_ENHANCED** 0x78

Read Id command description - read device followed by one address cycle to specify read ID type

- #define **NANDCMD_READ_ID** 0x90
- #define **NANDCMD_READ_ID_TYPE** 0x00
- #define NANDCMD_READ_ID_SIZE 2

  Number of commands sent for a NAND Device Read ID.
- #define NANDCMD_READ_ID_RESULT_SIZE 5

  Size in bytes of a Read ID command result.

Block erase command description

- #define **NANDCMD_BLOCK_ERASE** 0x60
- #define **NANDCMD_BLOCK_ERASE_CONFIRM** 0xD0

Read command description

- #define **NANDCMD_READ** 0x00
- #define **NANDCMD_READ_CONFIRM** 0x30

Program command description

- #define **NANDCMD_PAGE_PROGRAM** 0x80
- #define **NANDCMD_PAGE_PROGRAM_CONFIRM** 0x10
- #define **NANDCMD_PAGE_CACHE_PROG_CFRM** 0x15

7.3.1 Detailed Description

This chapter describes the NAND controller.

7.3.2 Macro Definition Documentation

7.3.2.0.1 #define NAND_DMA_WAIT4RDY_CMD

Value:

```
(IMX_APBH_CHn_CMD_CMDWORDS(1) | \
     IMX_APBH_CHn_CMD_HALTONTERMINATE(1) | \
     IMX_APBH_CHn_CMD_WAIT4ENDCMD(1) | \
     IMX_APBH_CHn_CMD_NANDWAIT4READY(1) | \
     IMX_APBH_CHn_CMD_NANDLOCK(0) | \
     IMX_APBH_CHn_CMD_CHAIN(1) | \
     BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER))
```

APBH DMA Macro for Wait4Ready command.

Macro/Defines used to create a DMA command word in the chain.Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Wait for Ready before starting next DMA descriptor in chain. Don't lock the nand while waiting for Ready to go high. Another descriptor follows this one in the chain. This DMA has no transfer.

Definition at line 55 of file dma_descriptor.h.

7.3.2.0.2 #define NAND_DMA_WAIT4RDY_PIO(*u32ChipSelect*)

Value:

```
(BV_FLD(GPMI_CTRL0, COMMAND_MODE, WAIT_FOR_READY) | \
     IMX_GPMI_CTRL0_WORD_LENGTH(IMX_GPMI_CTRL0_WORD_LENGTH_BV_8_BIT) | \
     BV_FLD(GPMI_CTRL0, ADDRESS, NAND_DATA) | \
     IMX_GPMI_CTRL0_CS(u32ChipSelect))
```

GPMI PIO DMA Macro for Wait4Ready command.

Wait for Ready before sending IRQ interrupt. Use 8 bit word length (doesn't really matter since no transfer). Watch u32ChipSelect.

Definition at line 71 of file dma_descriptor.h.

7.3.3 Typedef Documentation

7.3.3.0.1 typedef struct _apbh_dma_gpmi1_t apbh_dma_gpmi1_t

Define the APBH DMA structure with 1 GPMI Parameter word writes.

7.3.3.0.2 typedef struct _apbh_dma_gpmi3_t apbh_dma_gpmi3_t

Define the APBH DMA structure with 3 GPMI Parameter word writes.

7.3.3.0.3 typedef struct _apbh_dma_gpmi5_t apbh_dma_gpmi5_t

Define the APBH DMA structure with 5 GPMI Parameter word writes.

7.3.3.0.4 typedef struct _apbh_dma_t apbh_dma_t

Define the APBH DMA structure without GPMI transfers.

7.3.3.0.5 typedef struct _chipio_t chipio

Low level driver structure

7.3.3.0.6 typedef struct _dma_blk_erase_t dma_blk_erase_t

DMA chain structure for device erase block.

7.3.3.0.7 typedef struct _dma_programEcc_t dma_programEcc_t

DMA chain structure for NAND Program.

7.3.3.0.8 typedef struct _dma_programRaw_t dma_programRaw_t

DMA chain structure for NAND Program.

7.3.3.0.9 typedef struct _dma_read_id_device_t dma_read_id_device_t

DMA chain structure for Read ID

7.3.3.0.10 typedef struct _NAND_dma_read_status_device_t dma_read_status_t

DMA chain structure for Read Status.

7.3.3.0.11 typedef struct _dma_readEcc_device_t dma_readEcc_device_t

DMA chain structure for Raw Read Page

7.3.3.0.12 typedef struct _dma_readRaw_device_t dma_readRaw_device_t

DMA chain structure for Raw Read Page

7.3.3.0.13 typedef struct _dma_reset_device_t dma_reset_device_t

DMA chain structure for Reset Device

7.3.4 Function Documentation

7.3.4.0.1 void create_blockErase_descriptor (dma_blk_erase_t * *superStruct,* unsigned *page*)

Creates block erase descriptor.

Parameters

superStruct	Block erase descriptor.
page	Page number.

Definition at line 55 of file chipio.c.

7.3.4.0.2 void create_readEcc_descriptor (dma_readEcc_device_t * *superStruct,* unsigned *page,* void * *ret_data*)

Creates read ECC descriptor (read data using BCH peripheral).

Parameters

superStruct	Read ECC descriptor.
page	Page number.
ret_data	Pointer to read data buffer.

Definition at line 113 of file chipio.c.

7.3.4.0.3 void create_readId_descriptor (dma_read_id_device_t * *superStruct,* uint8_t * *ret_raw_id*)

Creates read identification descriptor.

Parameters

superStruct	Read identification descriptor.
ret_raw_id	Raw identification value.

Definition at line 567 of file chipio.c.

7.3.4.0.4 void create_readRaw_descriptor (dma_readRaw_device_t * *superStruct,* uint32_t *data_size,* unsigned *page,* void * *ret_data*)

Creates read raw data descriptor (reads whole page (if there is ECC, reads it also))

Parameters

superStruct	Read raw data descriptor.
data_size	Size of data to read.
page	Page number.
ret_data	Pointer to read data buffer.

Definition at line 222 of file chipio.c.

7.3.4.0.5 void create_readStatus2_descriptor (dma_read_status_t * *superStruct,* uint8_t * *memory_status,* unsigned *page*)

Creates read status (enhanced).

Parameters

superStruct	Read status descriptor (enhanced).
memory_status	Memory status.
page	Page number.

Definition at line 460 of file chipio.c.

7.3.4.0.6 void create_readStatus_descriptor (dma_read_status_t * *superStruct,* uint8_t * *memory_status*)

Creates read status descriptor.

Parameters

superStruct	Read status descriptor.
memory_status	Memory status value.

Definition at line 515 of file chipio.c.

7.3.4.0.7 void create_reset_descriptor (dma_reset_device_t * *superStruct*)

Creates reset descriptor.

Parameters

superStruct	Reset descriptor.

Definition at line 619 of file chipio.c.

7.3.4.0.8 void create_writeEcc_descriptor (dma_programEcc_t * *superStruct,* unsigned *page,* void * *write_data,* uint32_t *data_size*)

Creates write ECC descriptor (write data using BCH peripheral)

Parameters

superStruct	Write ECC descriptor.
page	Page number.
write_data	Pointer to write data buffer.
data_size	Size of data to write.

Definition at line 300 of file chipio.c.

7.3.4.0.9 void create_writeRaw_descriptor (dma_programRaw_t * *superStruct,* unsigned *page,* void * *write_data,* uint32_t *data_size*)

Creates write data descriptor (raw -> without BCH).

Parameters

superStruct	Write raw data descriptor.
page	Page number.
write_data	Pointer to data buffer.
data_size	Data size.

Definition at line 391 of file chipio.c.

7.3.4.0.10 void device_to_nfc (uint8_t * *parsed_data,* uint8_t * *raw_data*)

Transforms page representation from device (Nand memory layout) to processor.

Parameters

parsed_data	Nfc data representation.
raw_data	Data from NAND memory.

Definition at line 829 of file chipio.c.

7.3.4.0.11 bool is_dma_active (chipio * *chipio*)

Checks if previous DMA transaction was OK.

Parameters

chipio	Low level driver handle.

Returns

Status of the DMA.

Definition at line 782 of file chipio.c.

7.3.4.0.12 int nand_init (chipio * *cio*)

NAND controller peripherals (GPMI, BCH, APBH-DMA) initialization.

Parameters

cio	Low level driver handle.

Returns

EOK always.

Definition at line 675 of file chipio.c.

7.3.4.0.13 int nand_wait_busy (chipio * *cio,* uint32_t *time_out,* uint8_t *chip_select*)

Waits to receive NAND ready busy signal.

Parameters

cio	Low level driver handle.
time_out	Requested time out value.
chip_select	Chip-select number.

Return values

0	If device is not busy.
-1	If error.

Definition at line 809 of file chipio.c.

7.3.4.0.14 void nfc_to_device (uint8_t * *device_data,* uint8_t * *nfc_data,* uint8_t *ecc_size*)

Transforms page representation in processor to device (NAND memory layout).

Parameters

device_data	Data to be written to NAND memory.
nfc_data	Data to parse.
ecc_size	Size of ECC in bytes t*GF/8, intended only for RAW writes to device. For BCH writes use 0 always.

Definition at line 861 of file chipio.c.

7.3.4.0.15 int run_dma (apbh_dma_t * *dma,* chipio * *chipio,* uint8_t *wait_flag,* uint32_t *phy_addr*)

Starts requested DMA transfer to GPMI.

Parameters

dma	DMA transfer configuration.
chipio	Low level driver handle.
wait_flag	Wait flag value.
phy_addr	Physical address of the dma chain.

Returns

Execution status (NAND_XXX error code).

Definition at line 730 of file chipio.c.

7.3.5 DMA

This section describes the DMA API of the NAND controller.

Functions

- int apbh_intr_wait (chipio *chipio)
- void * apbhint_thread (void *arg)
- void apbh_init (chipio *chipio)
- void apbh_init_dma_channel (chipio *chipio)

7.3.5.1 Detailed Description

This section describes the DMA API of the NAND controller.

7.3.5.2 Function Documentation

7.3.5.2.1 void apbh_init (chipio * *chipio*)

DMA global initialization.

Parameters

chipio	Low level driver handle.

Definition at line 97 of file apbh_dma.c.

7.3.5.2.2 void apbh_init_dma_channel (chipio * *chipio*)

DMA channel initialization.

Parameters

chipio	Low level driver handle.

Definition at line 118 of file apbh_dma.c.

7.3.5.2.3 int apbh_intr_wait (chipio * *chipio*)

Conditional wait for interrupt occurrence.

Parameters

chipio	Low level driver handle.

Returns

NULL always.

Definition at line 47 of file apbh_dma.c.

7.3.5.2.4 void * apbhint_thread (void * *arg*)

APBH interrupt thread.

Parameters

arg	Low level driver handle.

Definition at line 64 of file apbh_dma.c.

7.3.6 ECC

This section describes the ECC API of the NAND controller.

Macros

- #define BCH_SUBBLOCK_SIZE 1024

  BCH sub block size in bytes.
- #define BCH_ECC_SIZE 42

  BCH ECC size in bytes.

Functions

- int bch_intr_wait (chipio *chipio)
- void * bchint_thread (void *arg)
- void bch_init (chipio *chipio)
- void bch_set_layout (chipio *chipio)
- void bch_set_erase_threshold (chipio *chipio, uint8_t threshold)
- uint32_t bch_get_ecc_status (chipio *chipio)

7.3.6.1 Detailed Description

This section describes the ECC API of the NAND controller.

7.3.6.2 Function Documentation

7.3.6.2.1 uint32_t bch_get_ecc_status (chipio * *chipio*)

Parameters

chipio	Low level driver handle.

Returns

BCH status register value.

Definition at line 176 of file bch_ecc.c.

7.3.6.2.2 void bch_init (chipio * *chipio*)

BCH peripheral initialization routine.

Parameters

chipio	Low level driver handle.

Definition at line 106 of file bch_ecc.c.

7.3.6.2.3 int bch_intr_wait (chipio * *chipio*)

Conditional wait for interrupt occurrence.

Parameters

chipio	Low level driver handle.

Returns

NULL always.

Definition at line 56 of file bch_ecc.c.

7.3.6.2.4 void bch_set_erase_threshold (chipio * *chipio,* uint8_t *threshold*)

Sets the BCH erase threshold value.

Parameters

chipio	Low level driver handle.
threshold	Threshold value.

Definition at line 164 of file bch_ecc.c.

7.3.6.2.5 void bch_set_layout (chipio * *chipio*)

Sets device parameters for BCH engine. (memory layout, sub-block size, ecc size, etc...).

Parameters

chipio	Low level driver handle.

Definition at line 135 of file bch_ecc.c.

7.3.6.2.6 void * bchint_thread (void * *arg*)

BCH interrupt thread.

Parameters

arg	Low level driver handle.

Definition at line 73 of file bch_ecc.c.

7.3.7 PIO

This section describes the PIO API of the NAND controller.

Functions

- void gpmi_soft_reset (chipio *chipio)
- void gpmi_set_busy_timeout (chipio *chipio, uint16_t busy_timeout)

7.3.7.1 Detailed Description

This section describes the PIO API of the NAND controller.

7.3.7.2 Function Documentation

7.3.7.2.1 void gpmi_set_busy_timeout (chipio * *chipio,* uint16_t *busy_timeout*)

Sets busy time out for GPMI transfers.

Parameters

chipio	Low level driver handle.
busy_timeout	Time out value.

Definition at line 84 of file gpmi_pio.c.

7.3.7.2.2 void gpmi_soft_reset (chipio * *chipio*)

GPMI software reset.

Parameters

chipio	Low level driver handle.

Definition at line 42 of file gpmi_pio.c.

Chapter 8

FFS3 low level driver (devf-qspi-imx)

This driver implements low level layer for `FFS3`. The FFS3 file system drivers implement a POSIX-like file system on NOR flash memory devices. The FFS3 driver does not configure clock sources and does not configure QSPI pins directly. These are configured in BSP startup code. The flash file system allows creating/erasing/formatting partitions using `flashctl` utility.

Modules

- Flash Controller

  This part describes the flash controller API support.
- Commands

  This part describes the commands API support.

Macros

- #define CLOCK_RATE 240000000

  Peripheral input clock - see startup for details.
- #define SIO3_MUX_PHYS_ADDR 0x30330040

  SW_MUX_CTL_PAD_EPDC_DATA03.
- #define SIO3_PAD_PHYS_ADDR 0x303302B0

  SW_PAD_CTL_PAD_EPDC_DATA03.
- #define PAGE_SIZE 256

  Page size in Bytes.
- #define SECTOR_SIZE 256

  Number of pages in one sector.
- #define TOTAL_SIZE 1024

  Number of sectors.
- #define DIE_NUMBER 1

  Number of die in connected memory/memories.
- #define ADDR_MODE_4BYTE 32

  32 bit address
- #define SCLK_FREQ (CLOCK_RATE / 4)

Peripheral input clock / QSPI in-build divider.

- #define DEVICE_ID 0x20

  Memory type.
- #define MAN_ID 0xC2

  Manufacturer identification.
- #define PWR2_SIZE_UNIT 16

  64kB erase sector size (fs unit size)

Functions

- int32_t f3s_qspi_open (f3s_socket_t *socket, uint32_t flags)
- uint8_t * f3s_qspi_page (f3s_socket_t *socket, uint32_t page, uint32_t offset, int32_t *size)
- int32_t f3s_qspi_read (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t text_offset, int32_t buffer_size, uint8_t *buffer)
- int32_t f3s_qspi_status (f3s_socket_t *socket, uint32_t flags)
- void f3s_qspi_close (f3s_socket_t *socket, uint32_t flags)
- void f3s_qspi_reset (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)
- int32_t f3s_qspi_ident (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)
- int32_t f3s_qspi_write (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset, int32_t size, uint8_t *buffer)
- int f3s_qspi_erase (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)
- int32_t f3s_qspi_sync (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t text_offset)
- int main (int argc, char **argv)

8.1 Detailed Description

This driver implements low level layer for `FFS3`. The FFS3 file system drivers implement a POSIX-like file system on NOR flash memory devices. The FFS3 driver does not configure clock sources and does not configure QSPI pins directly. These are configured in BSP startup code. The flash file system allows creating/erasing/formatting partitions using `flashctl` utility.

The FFS3 driver supports various command line options documented in src/hardware/flash/mtd-flash/usagev3.use file. Content of this file is also accessible in running QNX system by executing `use` command.

```
use devf-qspi-imx
```

FFS3 file system can be manually loaded by executing command below (for additional parameters please see use file options). After driver start dev/fs0 and dev/fs0p0 devices are created.

```
devf-qspi-imx
```

Example of erase, format and mount using flashctl utility in verbose mode (-v to see erase progress). Please note that erasing the memory will take several minutes. After erasing, formatting and mounting a partition will appear as /fs0p0:

```
flashctl -p /dev/fs0p0 -efmv
```

To see mounted partition size use `df` command:

```
df -h
```

8.2 Function Documentation

8.2.0.0.1 void f3s_qspi_close (f3s_socket_t * *socket,* uint32_t *flags*)

This is the close callout for QSPI serial NOR flash driver.

Parameters

socket	Socket Services Info.
flags	Flags.

Definition at line 38 of file f3s_qspi_close.c.

8.2.0.0.2 int f3s_qspi_erase (f3s_dbase_t * *dbase,* f3s_access_t * *access,* uint32_t *flags,* uint32_t *offset*)

This is the erase callout for QSPI serial NOR flash driver.

Parameters

dbase	Flash Services Database.
access	Access Super Structure.
flags	Flags.
offset	Memory offset.

Return values

EOK	Everything is OK.
ERARANGE	The offset is out of bounds.
EIO	Previous erase fail.

Definition at line 46 of file f3s_qspi_erase.c.

8.2.0.0.3 int f3s_qspi_ident (f3s_dbase_t * *dbase,* f3s_access_t * *access,* uint32_t *flags,* uint32_t *offset*)

This is the ident callout for QSPI serial NOR flash driver.

Parameters

dbase	Flash Services Database.
access	Access Super Structure.
flags	Flags.
offset	Memory offset.

Return values

EOK	Device detection was successful.

Return values

ENOENT	Device detection fail.

Definition at line 72 of file f3s_qspi_ident.c.

8.2.0.0.4 int32_t f3s_qspi_open (f3s_socket_t * *socket,* uint32_t *flags*)

This is the open callout for the QSPI serial NOR flash driver.

Parameters

socket	Socket Services Info.
flags	Flags.

Return values

EOK	Everything is fine.
EAGAIN	Otherwise.

Definition at line 42 of file f3s_qspi_open.c.

8.2.0.0.5 uint8_t * f3s_qspi_page (f3s_socket_t * *socket,* uint32_t *flags,* uint32_t *offset,* int32_t * *size*)

This is the page callout for QSPI serial NOR flash driver.

Parameters

socket	Socket Services Info.
flags	Flags.
offset	Memory offset.
size	Size of data.

Return values

~*NULL*	every time the offset is within bounds.
NULL	otherwise.

Definition at line 42 of file f3s_qspi_page.c.

8.2.0.0.6 int32_t f3s_qspi_read (f3s_dbase_t * *dbase,* f3s_access_t * *access,* uint32_t *flags,* uint32_t *text_offset,* int32_t *buffer_size,* uint8_t * *buffer*)

This is the read callout for QSPI serial NOR flash driver.

Parameters

dbase	Flash Services Database.
access	Access Super Structure
flags	Flags.
text_offset	Offset of memory to read data.
buffer_size	Buffer size.
buffer	Pointer to buffer.

Returns

Returns number of bytes read or -1 in case of an error.

Definition at line 44 of file f3s_qspi_read.c.

8.2.0.0.7 void f3s_qspi_reset (f3s_dbase_t * *dbase,* f3s_access_t * *access,* uint32_t *flags,* uint32_t *offset*)

This is the reset callout for QSPI serial NOR flash driver.

Parameters

dbase	Flash Services Database.
access	Access Super Structure.
flags	Flags.
offset	memory offset

Definition at line 39 of file f3s_qspi_reset.c.

8.2.0.0.8 int32_t f3s_qspi_status (f3s_socket_t * *socket,* uint32_t *flags*)

This is the status callout for QSPI serial NOR flash driver.

Parameters

socket	Socket Services Info.
flags	Flags.

Returns

EOK always.

Definition at line 39 of file f3s_qspi_status.c.

8.2.0.0.9 int32_t f3s_qspi_sync (f3s_dbase_t * *dbase,* f3s_access_t * *access,* uint32_t *flags,* uint32_t *text_offset*)

This is the sync callout for QSPI serial NOR flash driver. Called together with erase function to check erase progress.

Parameters

dbase	Flash Services Database.
access	Access Super Structure.
flags	Flags.
text_offset	Memory offset.

Return values

EOK	Everything is fine. Memory is ready.
ERANGE	Offset out of bounds.
EIO	This return code currently not supported.
EAGAIN	WIP flag is set. Erase still in progress.

Definition at line 47 of file f3s_qspi_sync.c.

8.2.0.0.10 int32_t f3s_qspi_write (f3s_dbase_t * *dbase,* f3s_access_t * *access,* uint32_t *flags,* uint32_t *offset,* int32_t *size,* uint8_t * *buffer*)

This is the write callout for QSPI serial NOR flash driver.

Parameters

dbase	Flash Services Database.
access	Access Super Structure.
flags	Flags.
offset	Offset where to write data.
size	Size of data.
buffer	Buffer to write to memory.

Returns

Size of written data if everything is fine. EIO otherwise.

Definition at line 44 of file f3s_qspi_write.c.

8.2.0.0.11 int main (int *argc,* char ** *argv*)

This is the main function for the QSPI f3s flash file system.

Parameters

argc	Argument count.
argv	Argument vector.

Returns

Execution status of main function.

Definition at line 40 of file f3s_qspi_main.c.

8.3 Flash Controller

This chapter describes the flash controller API support.

Data Structures

- union imx_qspi_lutx_t

Macros

- #define IMX_QSPI_AMBA_BASE_ADDRESS 0x60000000U

Functions

- int imx_qspi_setcfg (int fd)
- void * int_thread (void *arg)
- int imx_qspi_open (void)
- int imx_qspi_close (int fd)
- int imx_qspi_lock_lut (int fd)
- int imx_qspi_unlock_lut (int fd)
- imx_qspi_lutx_t imx_qspi_create_lut_record (uint8_t instr0, uint8_t pad0, uint8_t opr0, uint8_t instr1, uint8_t pad1, uint8_t opr1)
- int imx_qspi_write_lut (int fd, uint8_t index, imx_qspi_lutx_t *lutcmd0, imx_qspi_lutx_t *lutcmd1, imx_qspi_lutx_t *lutcmd2, imx_qspi_lutx_t *lutcmd3)
- int qspi_intr_wait (imx_qspi_t *dev)
- int imx_qspi_send_ip_nowait_cmd (int fd, uint8_t lut_index, uint32_t data_size_override)
- int imx_qspi_send_ip_cmd (int fd, uint8_t lut_index, uint32_t data_size_override)
- int imx_qspi_clear_fifo (const int qspi_fd, uint32_t mask)
- int imx_qspi_write_data (int fd, uint8_t *addr, uint32_t data_size)
- int imx_qspi_read_data (int fd, uint8_t *buffer, uint32_t size)

FIFO parameters

- #define IMX_QSPI_FIFO_WIDTH 4

  FIFO width: 4 bytes.
- #define IMX_QSPI_FIFO_DEPTH 32

  FIFO depth: 32 entries.

LUT commands

- #define IMX_QSPI_PAD_1 0x00

  Communication on 1 pad only.
- #define IMX_QSPI_PAD_4 0x02

  Communication on 4 pads.
- #define IMX_QSPI_INSTR_CMD 1

  FC command.
- #define IMX_QSPI_INSTR_ADDR 2

  FC address.
- #define IMX_QSPI_INSTR_DUMMY 3

  FC dummy operation.
- #define IMX_QSPI_INSTR_READ 7

  FC read.
- #define IMX_QSPI_INSTR_WRITE 8

  FC write.
- #define IMX_QSPI_INSTR_JMP_ON_CS 9

  FC jump on cs.
- #define IMX_QSPI_INSTR_ADDR_DDR 10

  FC ddr address.
- #define IMX_QSPI_INSTR_READ_DDR 14

  FC ddr read.
- #define IMX_QSPI_INSTR_STOP 0

  FC stop.
- #define IMX_QSPI_LUT_KEY 0x5AF05AF0U

  LUT key.
- #define IMX_QSPI_LUT_LOCK 0x1

  LUT contect lock.
- #define IMX_QSPI_LUT_UNLOCK 0x2

  LUT contect unlock.

LUT device command indexes

- #define IMX_QSPI_LUT_CMD_IDX_READ 0

  Normal read 24-bit address read.
- #define IMX_QSPI_LUT_CMD_IDX_RDIR 1

  Read identification.
- #define IMX_QSPI_LUT_CMD_IDX_WREN 2

  Write enable.
- #define IMX_QSPI_LUT_CMD_IDX_SE 3

  Sector erase.
- #define IMX_QSPI_LUT_CMD_IDX_RDSR 4

  Read status register.
- #define IMX_QSPI_LUT_CMD_IDX_RDSCUR 5

  Read security register.
- #define IMX_QSPI_LUT_CMD_IDX_4READ4B 6

  4 read 4B
- #define IMX_QSPI_LUT_CMD_IDX_4PP4B 7

  4 write 4B
- #define IMX_QSPI_LUT_CMD_IDX_WRSR 8

Write status register (Normal mode)

- #define IMX_QSPI_LUT_CMD_IDX_EN4B 9

  Enter 4-byte addressing mode.

- #define IMX_QSPI_LUT_CMD_IDX_RDCR 10

  Read configuration register.

8.3.1 Detailed Description

This chapter describes the flash controller API support.

8.3.2 Macro Definition Documentation

8.3.2.0.1 #define IMX_QSPI_AMBA_BASE_ADDRESS 0x60000000U

QSPI external memory mapping

Definition at line 48 of file imx_fc_qspi.h.

8.3.3 Function Documentation

8.3.3.0.1 int imx_qspi_clear_fifo (const int *qspi_fd,* uint32_t *mask*)

Clear either Rx or Tx FIFO or both. Depends on mask parameter.

Parameters

qspi_fd	File descriptor.
mask	Mask of the FIFO(s).

Returns

EOK always.

Definition at line 444 of file imx_fc_qspi.c.

8.3.3.0.2 int imx_qspi_close (int *fd*)

De-init method of the controller.

Parameters

fd	File descriptor.

Returns

EOK always.

Definition at line 254 of file imx_fc_qspi.c.

8.3.3.0.3 imx_qspi_lutx_t imx_qspi_create_lut_record (uint8_t *instr0,* uint8_t *pad0,* uint8_t *opr0,* uint8_t *instr1,* uint8_t *pad1,* uint8_t *opr1*)

Creates look-up record. One look-up entry contains up to 4 records. See imx_qspi_write_lut for details.

Parameters

instr0	Controller command.
pad0	Number of pins used for communication.
opr0	Device command, data size, etc.
instr1	Controller command.
pad1	Number of pins used for communication.
opr1	Device command, data size, etc.

Returns

created record

Definition at line 312 of file imx_fc_qspi.c.

8.3.3.0.4 int imx_qspi_lock_lut (int *fd*)

Locks look-up table content.

Parameters

fd	File descriptor.

Returns

EOK always.

Definition at line 273 of file imx_fc_qspi.c.

8.3.3.0.5 int imx_qspi_open (void)

First part of controller initialization

Returns

file descriptor

Definition at line 192 of file imx_fc_qspi.c.

8.3.3.0.6 int imx_qspi_read_data (int *fd,* uint8_t * *buffer,* uint32_t *size*)

Reads data from QSPI Rx FIFO.

Parameters

fd	File descriptor.
buffer	Pointer where to copy Rx data.
size	Expected amount of data.

Returns

Read data size.

Definition at line 510 of file imx_fc_qspi.c.

8.3.3.0.7 int imx_qspi_send_ip_cmd (int *fd,* uint8_t *lut_index,* uint32_t *data_size_override*)

Sends command to connected device. Intended for Tx operations. This method waits for command completion.

Parameters

fd	File descriptor.
lut_index	Look-up table index.
data_size_override	Optional parameter that will override default value of data size in LUT.

Returns

EOK always

Definition at line 419 of file imx_fc_qspi.c.

8.3.3.0.8 int imx_qspi_send_ip_nowait_cmd (int *fd,* uint8_t *lut_index,* uint32_t *data_size_override*)

Sends command to connected device. Intended for Rx operations. This method does not wait for command completion. Expected that additional code wait for Rx FIFO flag.

Parameters

fd	File descriptor.
lut_index	Look-up table index.
data_size_override	Optional parameter that will override default value of data size in LUT.

Returns

EOK always.

Definition at line 397 of file imx_fc_qspi.c.

8.3.3.0.9 int imx_qspi_setcfg (int *fd*)

Second part of controller initialization

Parameters

fd	file descriptor

Returns

EOK always

Definition at line 51 of file imx_fc_qspi.c.

8.3.3.0.10 int imx_qspi_unlock_lut (int *fd*)

Unlocks look-up table content.

Parameters

fd	File descriptor.

Returns

EOK always.

Definition at line 290 of file imx_fc_qspi.c.

8.3.3.0.11 int imx_qspi_write_data (int *fd,* uint8_t * *addr,* uint32_t *data_size*)

Writes data to QSPI Tx FIFO.

Parameters

fd	File descriptor.
addr	Address of the write data buffer.
data_size	Size of data to write.

Returns

EIO if data size exceed Tx FIFO size, EOK otherwise.

Definition at line 464 of file imx_fc_qspi.c.

8.3.3.0.12 int imx_qspi_write_lut (int *fd,* uint8_t *index,* imx_qspi_lutx_t * *lutcmd0,* imx_qspi_lutx_t * *lutcmd1,* imx_qspi_lutx_t * *lutcmd2,* imx_qspi_lutx_t * *lutcmd3*)

Creates look-up table entry.

Parameters

fd	File descriptor.
index	Index in look-up table.
lutcmd0	Record 0.
lutcmd1	Record 1.
lutcmd2	Record 2.
lutcmd3	Record 3.

Returns

EOK always.

Definition at line 339 of file imx_fc_qspi.c.

8.3.3.0.13 void* int_thread (void * *arg*)

This thread is dedicated to handling and managing interrupts

Parameters

arg	device handle

Definition at line 162 of file imx_fc_qspi.c.

8.3.3.0.14 int qspi_intr_wait (imx_qspi_t * *dev*)

Waits for interrupt occurrence.

Parameters

dev	Device handle.

Returns

NULL always.

Definition at line 375 of file imx_fc_qspi.c.

8.4 Commands

This chapter describes the commands API support.

Functions

- int iswriting (const int qspi_fd)
- int read_cfg (const int qspi_fd, uint8_t *cfg_reg)
- int read_erase_status (const int qspi_fd)
- int read_status (const int qspi_fd, uint8_t *stat_reg)
- int write_status (const int qspi_fd, uint8_t *stat_reg)
- int read_ident (const int qspi_fd, int *manufact_id, int *device_id, uint32_t *size)
- int pd_release (const int qspi_fd)
- int sector_erase (const int qspi_fd, const int offset)
- int page_program (const int qspi_fd, int offset, int len, uint8_t *data)
- int read_from (const int qspi_fd, int offset, int len, uint8_t *buffer)

Device status register definition - MX25L51245G

- #define DEVICE_SR_WIP (1 << 0)

  Write in Progress (WIP) bit.
- #define DEVICE_SR_WEL (1 << 1)

  Write Enable Latch (WEL) bit.
- #define DEVICE_SR_BP0 (1 << 2)

  Block Protect bit BP0.
- #define DEVICE_SR_BP1 (1 << 3)

  Block Protect bit BP1.
- #define DEVICE_SR_BP2 (1 << 4)

  Block Protect bit BP2.
- #define DEVICE_SR_BP3 (1 << 5)

  Block Protect bit BP3.
- #define DEVICE_SR_QE (1 << 6)

  4 mode enable bit
- #define DEVICE_SR_SRWD (1 << 7)

  Status Register Write Disable (SRWD) bit.

Flash commands

- #define FLASH_OPCODE_SE4B 0xDC

  (SPI) Erase 64 KB block (Sector erase) 4B
- #define FLASH_OPCODE_READ4B 0x13

  (SPI) Read data bytes 4B
- #define FLASH_OPCODE_4READ4B 0xEC

  (QSPI) Read data bytes 4B
- #define FLASH_OPCODE_PP4B 0x12

  (SPI) Page program 4B
- #define FLASH_OPCODE_4PP4B 0x3E

  (QSPI) Page program 4B

- #define FLASH_OPCODE_RDID 0x9F

  (SPI) Read JEDEC ID in normal mode
- #define FLASH_OPCODE_WREN 0x06

  (SPI) Write enable
- #define FLASH_OPCODE_WRSR 0x01

  (SPI) Write status register
- #define FLASH_OPCODE_RDSR 0x05

  (SPI) Read status register
- #define FLASH_OPCODE_EN4B 0xB7

  (SPI) Enable 4B
- #define FLASH_OPCODE_RDSCUR 0x2B

  (SPI) Read security register

8.4.1 Detailed Description

This chapter describes the commands API support.

8.4.2 Function Documentation

8.4.2.0.1 int iswriting (const int *qspi_fd*)

Check WIP bit of the status register of memory and returns 0/1.

Parameters

in	*qspi_fd*	File descriptor.

Return values

1	Writing in progress.
0	Otherwise.

Definition at line 48 of file qspi_cmds.c.

8.4.2.0.2 int page_program (const int *qspi_fd,* int *offset,* int *len,* uint8_t * *data*)

Program page at given offset.

Parameters

qspi_fd	File descriptor.
offset	Memory offset.
len	Size of data to write.
data	Pointer to data write buffer.

Returns

Size of written data.

Definition at line 210 of file qspi_cmds.c.

8.4.2.0.3 int pd_release (const int *qspi_fd*)

Do nothing (Originally intended for power down release).

Parameters

qspi_fd	File descriptor.

Returns

EOK always.

Definition at line 167 of file qspi_cmds.c.

8.4.2.0.4 int read_cfg (const int *qspi_fd,* uint8_t * *cfg_reg*)

Read configuration register (memory vendor specific).

Parameters

in	*qspi_fd*	File descriptor.
out	*cfg_reg*	Configuration register.

Returns

EOK always.

Definition at line 67 of file qspi_cmds.c.

8.4.2.0.5 int read_erase_status (const int *qspi_fd*)

Read erase status of the memory.

Parameters

in	*qspi_fd*	File descriptor.

Return values

EOK	If erase succeed.
EIO	If erase failed.

Definition at line 85 of file qspi_cmds.c.

8.4.2.0.6 int read_from (const int *qspi_fd,* int *offset,* int *len,* uint8_t * *buffer*)

Read data from given offset.

Parameters

qspi_fd	File descriptor.
offset	Memory offset.
len	Size of data to read.
buffer	Pointer to data read buffer.

Returns

Size of read data.

Definition at line 247 of file qspi_cmds.c.

8.4.2.0.7 int read_ident (const int *qspi_fd,* int * *manufact_id,* int * *device_id,* uint32_t * *size*)

Reads identification string of the memory device.

Parameters

in	*qspi_fd*	File descriptor.
out	*manufact_id*	Manufacturer identification.
out	*device_id*	Device memory identification.
out	*size*	Memory size.

Returns

EOK always.

Definition at line 144 of file qspi_cmds.c.

8.4.2.0.8 int read_status (const int *qspi_fd,* uint8_t * *stat_reg*)

Read status register of the memory device.

Parameters

in	*qspi_fd*	File descriptor.
out	*stat_reg*	Memory status register.

Returns

EOK always.

Definition at line 105 of file qspi_cmds.c.

8.4.2.0.9 int sector_erase (const int *qspi_fd,* const int *offset*)

Erase sector at given offset.

Parameters

qspi_fd	File descriptor.
offset	Memory offset.

Return values

EOK	If OK.
-1	If previous erase fail

See also

read_erase_status(const int qspi_fd).

Definition at line 181 of file qspi_cmds.c.

8.4.2.0.10 int write_status (const int *qspi_fd,* uint8_t * *stat_reg*)

Writes status register of the memory.

Parameters

in	*qspi_fd*	File descriptor.
in	*stat_reg*	Status register value.

Returns

EOK always.

Definition at line 123 of file qspi_cmds.c.

Chapter 9

I2C - Inter-Integrated Circuit driver (i2c-imx)

This part describes the I2C driver i2c-imx.

Data Structures

- struct _imx_dev

Typedefs

- typedef struct _imx_dev imx_dev_t

Functions

- unsigned int find_best_ic (unsigned int i2c_div)
- int imx_set_bus_speed (void *hdl, unsigned int speed, unsigned int *ospeed)
- void imx_i2c_reset (imx_dev_t *dev)
- i2c_status_t imx_recvbyte (imx_dev_t *dev, uint8_t *byte, int nack, int stop)
- i2c_status_t imx_sendbyte (imx_dev_t *dev, uint8_t byte)
- i2c_status_t imx_sendaddr7 (imx_dev_t *dev, unsigned addr, int read, int restart)
- i2c_status_t imx_sendaddr10 (imx_dev_t *dev, unsigned addr, int read, int restart)
- void imx_fini (void *hdl)
- int imx_driver_info (void *hdl, i2c_driver_info_t *info)
- void * imx_init (int argc, char *argv[ ])
- int i2c_master_getfuncs (i2c_master_funcs_t *funcs, int tabsize)
- int query_hwi_device (imx_dev_t *dev, unsigned unit)
- int imx_options (imx_dev_t *dev, int argc, char *argv[ ])
- int imx_wait_bus_not_busy (imx_dev_t *dev)
- int imx_set_slave_addr (void *hdl, unsigned int addr, i2c_addrfmt_t fmt)
- int imx_version_info (i2c_libversion_t *version)
- i2c_status_t imx_recv (void *hdl, void *buf, unsigned int len, unsigned int stop)
- i2c_status_t imx_send (void *hdl, void *buf, unsigned int len, unsigned int stop)
- uint32_t imx_wait_status (imx_dev_t *dev)

9.1 Detailed Description

This part describes the I2C driver i2c-imx. I2C (Inter-Integrated Circuit) is a simple serial protocol that connects multiple devices in a master-slave relationship. More information about I2C support in QNX OS can be found on QNX website - I2C (Inter-Integrated Circuit) Framework.

The I2C driver does not configure clock sources and does not configure I2C pins directly. These are configured in BSP startup code.

The I2C driver supports various command line options documented in src/hardware/i2c/imx/i2c-imx.use file. Content of this file is also accessible in running QNX system by executing use command:

```
# use i2c-imx
```

The I2C driver is loaded by default when system starts. Each I2C peripheral (I2C1,I2C2,..) has own driver instance.

Manually can be loaded by executing commands:

```
# i2c-imx -p 0x30A20000 -i67 -c24000000 --u 1
# i2c-imx -p 0x30A30000 -i68 -c24000000 --u 2
# i2c-imx -p 0x30A40000 -i69 -c24000000 --u 3
# i2c-imx -p 0x30A50000 -i70 -c24000000 --u 4
```

Peripheral physical address, IRQ number and clock source frequency should be passed as parameters to every I2C driver instance.

After loading drivers new devices /dev/i2cX are created:

```
# ls /dev/i2c
i2c1  i2c2  i2c3  i2c4
```

9.2 Typedef Documentation

9.2.0.0.1 typedef struct _imx_dev imx_dev_t

I2C driver device structure.

9.3 Function Documentation

9.3.0.0.1 unsigned int find_best_ic (unsigned int *i2c_div*)

Private function. Finds best IC register setting according to required divider value.

Parameters

i2c_div	I2C clock divider value.

Returns

Best value for register field IC or IMX_DEFAULT_IC value.

Definition at line 47 of file bus_speed.c.

9.3.0.0.2 int i2c_master_getfuncs (i2c_master_funcs_t * *funcs,* int *tabsize*)

Used by higher-level code to access the hardware-specific functions.

Parameters

funcs	The function table to fill in.
tabsize	The size of the structure that funcs points to, in bytes.

Returns

Execution status.

Return values

0	Success.

Definition at line 42 of file lib.c.

9.3.0.0.3 int imx_driver_info (void * *hdl,* i2c_driver_info_t * *info*)

Gets I2C driver info.

Parameters

hdl	Pointer to I2C driver device structure.
info	Pointer to i2c_driver_info_t structure.

Returns

Execution status.

Return values

0	Success.

Definition at line 42 of file info.c.

9.3.0.0.4 void imx_fini (void * *hdl*)

Cleans up driver.

Parameters

hdl	Pointer to I2C device structure.

Definition at line 38 of file fini.c.

9.3.0.0.5 void imx_i2c_reset (imx_dev_t * *dev*)

Private function. Resets I2C peripheral.

Parameters

dev	Pointer to I2C driver device structure.

Definition at line 38 of file common.c.

9.3.0.0.6 void * imx_init (int *argc,* char * *argv[]*)

Initializes I2C driver.

Parameters

argc	Command-line arguments count.
argv	Array of command-line arguments.

Returns

Pointer to I2C driver device structure or NULL when fails.

Definition at line 41 of file init.c.

9.3.0.0.7 int imx_options (imx_dev_t * *dev,* int *argc,* char * *argv[]*)

Parses I2C driver specific options.

Parameters

dev	Pointer to I2C driver device structure.
argc	Number of arguments.
argv	Pointer to arguments.

Returns

Execution status.

Return values

0	Success.
-1	Fail.

Definition at line 71 of file options.c.

9.3.0.0.8 i2c_status_t imx_recv (void * *hdl,* void * *buf,* unsigned int *len,* unsigned int *stop*)

Receives data from I2C bus.

Parameters

hdl	Pointer to I2C driver device structure.
buf	Pointer to data buffer.
len	Data buffer length.
stop	Determines if STOP condition is sent.

Returns

Execution status.

Return values

I2C_STATUS_DONE	Success.
I2C_STATUS_ERROR	I2C error.
I2C_STATUS_ARBL	I2C arbitration lost error.
I2C_STATUS_NACK	Slave no-acknowledge.

Definition at line 47 of file recv.c.

9.3.0.0.9 i2c_status_t imx_recvbyte (imx_dev_t * *dev,* uint8_t * *byte,* int *nack,* int *stop*)

Private function. Receives one byte over 12C bus.

Parameters

dev	Pointer to I2C driver device structure.
byte	Pointer to variable where store received byte.
nack	Whether send NACK or not.
stop	Whether send STOP or not.

Returns

Execution status.

Return values

I2C_STATUS_ERROR	I2C error.
0	Success.

Definition at line 87 of file common.c.

9.3.0.0.10 i2c_status_t imx_send (void * *hdl,* void * *buf,* unsigned int *len,* unsigned int *stop*)

Sends data over I2C bus.

Parameters

hdl	Pointer to I2C driver device structure.
buf	Pointer to data buffer.
len	Data buffer length in bytes.
stop	Determines whether STOP condition is sent after data transfer.

Returns

Execution status.

Return values

I2C_STATUS_DONE	Success.
I2C_STATUS_ERROR	I2C error.
I2C_STATUS_ARBL	I2C arbitration lost error.
I2C_STATUS_NACK	Slave no-acknowledge.

Definition at line 47 of file send.c.

9.3.0.0.11 i2c_status_t imx_sendaddr10 (imx_dev_t * *dev,* unsigned *addr,* int *read,* int *restart*)

Private function. Sends 10-bit slave address on I2C bus.

Parameters

dev	Pointer to I2C driver device structure.
addr	Slave device address.
read	Determines read (IMX_I2C_ADDR_RD) or write (IMX_I2C_ADDR_WR) on I2C bus.
restart	Determines if repeated Start condition is generated.

Returns

Execution status.

Return values

I2C_STATUS_ERROR	I2C error.
I2C_STATUS_ARBL	I2C arbitration lost error.
I2C_STATUS_NACK	Slave no-acknowledge.
0	Success.

Definition at line 195 of file common.c.

9.3.0.0.12 i2c_status_t imx_sendaddr7 (imx_dev_t * *dev,* unsigned *addr,* int *read,* int *restart*)

Private function. Sends 7-bit slave address on I2C bus.

Parameters

dev	Pointer to I2C driver device structure.
addr	Slave device address.
read	Determines read (IMX_I2C_ADDR_RD) or write (IMX_I2C_ADDR_WR) on I2C bus.
restart	Determines if repeated Start condition is generated.

Returns

Execution status.

Return values

I2C_STATUS_ERROR	I2C error.
I2C_STATUS_ARBL	I2C arbitration lost error.
I2C_STATUS_NACK	Slave no-acknowledge.
0	Success.

Definition at line 170 of file common.c.

9.3.0.0.13 i2c_status_t imx_sendbyte (imx_dev_t * *dev,* uint8_t *byte*)

Private function. Sends one byte on I2C bus.

Parameters

dev	Pointer to I2C driver device structure.
byte	Byte to send.

Returns

Execution status.

Return values

I2C_STATUS_ERROR	I2C error.
I2C_STATUS_ARBL	I2C arbitration lost error.
I2C_STATUS_NACK	Slave no-acknowledge.
0	Success.

Definition at line 121 of file common.c.

9.3.0.0.14 int imx_set_bus_speed (void * *hdl,* unsigned int *speed,* unsigned int * *ospeed*)

Sets i2c bus speed.

Parameters

hdl	Pointer to I2C device structure.
speed	Required speed.
ospeed	Pointer to variable where real calculated speed is returned.

Returns

Execution status.

Return values

-1	Fail.
0	Success.

Definition at line 70 of file bus_speed.c.

9.3.0.0.15 int imx_set_slave_addr (void * *hdl,* unsigned int *addr,* i2c_addrfmt_t *fmt*)

Sets I2C slave address for send() and receive() commands.

Parameters

hdl	Pointer to I2C driver device structure.
addr	I2C device slave address.
fmt	I2C device slave address length I2C_ADDRFMT_10BIT or I2C_ADDRFMT_7BIT.

Returns

Execution status.

Return values

-1	Fail with errno set.
0	Success.

Definition at line 44 of file slave_addr.c.

9.3.0.0.16 int imx_version_info (i2c_libversion_t * *version*)

Gets I2C library version.

Parameters

version	Pointer to i2c_libversion_t structure to fill.

Returns

Execution status.

Return values

0	Success.

Definition at line 45 of file version.c.

9.3.0.0.17 int imx_wait_bus_not_busy (imx_dev_t * *dev*)

Private function. Waits while I2C bus is in busy state.

Parameters

dev	Pointer to I2C driver device structure.

Returns

Execution status.

Return values

0	Success.
-1	Fails on timeout.

Definition at line 42 of file wait.c.

9.3.0.0.18 uint32_t imx_wait_status (imx_dev_t * *dev*)

Private function. Waits on I2C interrupt and then returns I2C Status register content.

Parameters

dev	Pointer to I2C driver device structure.

Returns

I2C Status register value or zero when timeouts.

Definition at line 75 of file wait.c.

9.3.0.0.19 int query_hwi_device (imx_dev_t * *dev,* unsigned *unit*)

Finds I2C device in hwi table.

Parameters

dev	Pointer to I2C driver device structure.
unit	I2C unit number.

Returns

Execution status.

Return values

1	Success.
0	Not found.

Definition at line 43 of file options.c.

Chapter 10

SD/eMMC driver (devb-sdmmc-imx)

This part describes the SD/eMMC driver.

Modules

- Board specific interface

  i.MX SD/eMMC board specific interface.
- Host controller interface

  i.MX SD/eMMC host controller interface.

10.1 Detailed Description

This part describes the SD/eMMC driver. *NOTE:* SD, SDIO and eMMC interfaces, including core modules are not documented using Doxygen.
The SD/eMMC driver does not configure clock sources and does not configure SD/eMMC pins directly. These are configured in BSP startup code.

The SD/eMMC driver supports various command line options documented in src/hardware/devb/sdmmc/devb-sdmmc.use file. Content of this file is also accessible in running QNX system by executing use command:

```
# use devb-sdmmc-imx
```

After driver start dev/hd*x* and dev/hd*xty* devices are commonly created.

- dev/hd*x* is the hard disk *x*. Hard disk - represents each raw disk and may be accessed using POSIX standard api.
- dev/hd*xty* is hdt*y* partition on it. QNX supports following partitions .

SD/eMMMC driver integration into QNX sub-system:

io-blk:

The `io-blk.so` library provides block I/O support for the devb-∗ driver. Most of the file system shared libraries ride on top of the Block I/O module. The `io-blk.so` module also acts as a resource manager and exports a block-special file for each physical device.

cam-disk:

The `cam-disk.so` provides common access methods (CAMs) for hard disk devices.

devb-∗ options to improve R/W performance:

```
blk noatime,commit=none,delwri=5:5,maxio=256,rapolicy=aggressive,cache=20m
```

For more details please refer `QNX knowledge base on web`.

10.2 Board specific interface

i.MX SD/eMMC board specific interface.

Data Structures

- struct _imx_ext

Typedefs

- typedef struct _imx_ext imx_ext_t

Functions

- int my_getsubopt (char **optionp, char *const *tokens, char **valuep)
- int bs_event (sdio_hc_t *hc, sdio_event_t *ev)

10.2.1 Detailed Description

i.MX SD/eMMC board specific interface.

10.2.2 Typedef Documentation

10.2.2.0.1 typedef struct _imx_ext imx_ext_t

Structure describing board specific configuration

10.2.3 Function Documentation

10.2.3.0.1 int bs_event (sdio_hc_t * *hc,* sdio_event_t * *ev*)

Board specific event

Parameters

hc	Host controller handle
ev	Event

Returns

Execution status

Definition at line 519 of file bs.c.

10.2.3.0.2 int my_getsubopt (char ** *optionp,* char *const * *tokens,* char ** *valuep*)

Use ":" as separator, other than the regular ","

Parameters

optionp	Option pointer
tokens	Tokens
valuep	Value pointer

Return values

-1	if index out of bounds.
Index	value.

Definition at line 178 of file bs.c.

10.3 Host controller interface

i.MX SD/eMMC host controller interface.

Data Structures

- struct _imx_sdhcx_adma32_t
- struct _imx_usdhcx_hc

Typedefs

- typedef struct _imx_sdhcx_adma32_t imx_sdhcx_adma32_t
- typedef struct _imx_usdhcx_hc imx_sdhcx_hc_t

Functions

- int imx_sdhcx_dinit (sdio_hc_t *hc)
- int imx_sdhcx_init (sdio_hc_t *hc)

10.3.1 Detailed Description

i.MX SD/eMMC host controller interface.

10.3.2 Typedef Documentation

10.3.2.0.1 typedef struct _imx_sdhcx_adma32_t imx_sdhcx_adma32_t

32 bit ADMA descriptor definition

10.3.2.0.2 typedef struct _imx_usdhcx_hc imx_sdhcx_hc_t

Processor specific structure

10.3.3 Function Documentation

10.3.3.0.1 int imx_sdhcx_dinit (sdio_hc_t * *hc*)

Host controller de-initialization

Parameters

hc	Host controller handle

Returns

EOK always

Definition at line 1234 of file imx_hc.c.

10.3.3.0.2 int imx_sdhcx_init (sdio_hc_t * *hc*)

Host controller initialization

Parameters

hc	Host controller handle

Returns

Execution status

Definition at line 1285 of file imx_hc.c.

Chapter 11

QNX startup program (startup-imx-sabre)

This part describes an QNX startup program code. Start code is performed during QNX boot sequence. The startup code initializes the hardware, fills the system page with information about the hardware, loads callout routines that the kernel uses for interacting with the hardware, and then loads and starts the microkernel and process manager, procnto. More information about QNX start can be found on QNX website .

Generic options for startup program are stored in build file:

```
startup-imx-sabre -m -v -n0 -e -W
```

Generic options:

- m
Enable Data-cache/MMU for startup code (it improves QNX boot time).

- n0
Enable NOR QSPI Flash memory. Initializes QSPI device clock speed and routes pins to NOR Flash memory (QSPI pin signals needs to be interconnected on board).

-n1
Enable NAND Flash memory (will re-route SAI1 and SD3 signals)

-e
Enable eMMC memory. Initializes USDHC3 device clock speed and routes pins to the eMMC Flash memory (Please check if eMMC (U10) chip is available/soldered on board).

-W
Enable watchdog device. If watchdog device is enabled, wdtkick driver must be started in build file (utility for watchdog timer refreshing).

Chapter 12

IPL - Initial Program Loader (ipl-imx-sabre)

This part describes an QNX Initial program loader (IPL) code. The initial task of the IPL is to minimally configure the hardware to create an environment that allows the startup program (e.g. startup-bios, startup-ixdp425, etc.), and consequently the Neutrino microkernel, to run. This includes at least the following:

1. Start execution from the reset vector.
2. Configure the memory controller. This may include configuring the chip selects and/or PCI controller.
3. Configure clocks.
4. Set up a stack to allow the IPL library to perform OS verification and setup (download, scan, set up, and jump to the OS image).

More information about QNX Initial Program Loader (IPL) can be can be found on QNX website.

Chapter 13

RTC - Real Time Clock

This part describes the RTC application.

Functions

- int init_mx7srtc (struct chip_loc *chip, char *argv[])
- int get_mx7srtc (struct tm *tm, int cent_reg)
- int set_mx7srtc (struct tm *tm, int cent_reg)
- int init_net (struct chip_loc *chip, char *argv[])
- int get_net (struct tm *tm, int cent_reg)
- int set_net (struct tm *tm, int cent_reg)
- char * query_clock_hw (struct chip_loc *chip)
- int load_external_clock (const char *given_name, struct rtc_desc *clk)
- int close_external_clock (void)
- unsigned chip_read (unsigned off, unsigned size)
- void chip_write (unsigned off, unsigned val, unsigned size)
- int main (int argc, char *argv[])

13.1 Detailed Description

This part describes the RTC application. RTC application sets or gets date from realtime clock. More information about RTC driver in QNX OS can be found on QNX website - `rtc`.

This driver version implements i.MX RTC driver as a part of rtc application.

The RTC application supports various command line options documented in src/utils/r/rtc/rtc.use file. Content of this file is also accessible in running QNX system by executing `use` command:

```
use rtc
```

The RTC driver is executed by default when system starts thus system time is synchronized with RTC. RTC can be also executed manually:

```
rtc hw
```

When system starts first time (eg. after RTC backup battery replacement) the RTC peripheral is stopped and should be manually synchronized with system time by execution:

```
rtc -s hw
```

13.2 Function Documentation

13.2.0.0.1 unsigned chip_read (unsigned *off,* unsigned *size*)

Reads value stored in register specified by register offset and register size.

Parameters

off	Register offset.
size	Register size - 8, 16, 32 bits.

Returns

Value read from register.

Definition at line 66 of file qnxrtc.c.

13.2.0.0.2 void chip_write (unsigned *off,* unsigned *val,* unsigned *size*)

Writes value into a register specified by register offset and register size.

Parameters

off	Register offset.
val	Value to write.
size	Register size in bits - 8, 16, 32 bits.

Definition at line 123 of file qnxrtc.c.

13.2.0.0.3 int close_external_clock (void)

Closes rtc dynamic library.

Returns

Execution status.

Return values

-1	Execution failed.
0	Dynamic library closed.

Definition at line 264 of file support.c.

13.2.0.0.4 int get_mx7srtc (struct tm * *tm,* int *cent_reg*)

Gets i.mx rtc time.

Parameters

tm	Pointer to a time structure.
cent_reg	Unussed parameter.

Returns

Execution status.

Return values

0	Success.

Definition at line 70 of file clk_mx7srtc.c.

13.2.0.0.5 int get_net (struct tm * *tm,* int *cent_reg*)

Gets network rtc time.

Parameters

tm	Pointer to a time structure.
cent_reg	Unussed parameter.

Returns

Execution status.

Return values

-1	Failed.
0	Success.

Definition at line 88 of file clk_net.c.

13.2.0.0.6 int init_mx7srtc (struct chip_loc * *chip,* char * *argv[]*)

Initializes i.mx rtc.

Parameters

chip	Pointer to chip structure.
argv	Command line arguments.

Returns

SNVS registers size.

Definition at line 49 of file clk_mx7srtc.c.

13.2.0.0.7 int init_net (struct chip_loc * *chip,* char * *argv[]*)

Initializes network rtc.

Parameters

chip	Pointer to chip structure.
argv	Command line arguments.

Returns

Execution status.

Return values

0	Success.
-1	Failed.

Definition at line 52 of file clk_net.c.

13.2.0.0.8 int load_external_clock (const char * *given_name,* struct rtc_desc * *clk*)

Loads rtc functions from specified dynamic library.

Parameters

given_name	A name of dynamic library.
clk	Pointer to rtc_desc structure.

Returns

Execution status.

Return values

-1	Execution failed.
0	Dynamic library open failed.
1	Success.

Definition at line 211 of file support.c.

13.2.0.0.9 int main (int *argc,* char * *argv[]*)

Application main function.

Parameters

argc	Argument count.
argv	Pointer to argument array.

Returns

Execution status.

Definition at line 203 of file qnxrtc.c.

13.2.0.0.10 char* query_clock_hw (struct chip_loc * *chip*)

Gets HW rtc type from environment variable, syspage or hwinfo table.

Parameters

chip	Pointer to RTC chip structure.

Returns

Pointer to a string with RTC name or NULL.

Definition at line 141 of file support.c.

13.2.0.0.11 int set_mx7srtc (struct tm * *tm,* int *cent_reg*)

Sets i.mx rtc time.

Parameters

tm	Pointer to a time structure.
cent_reg	Unussed parameter.

Returns

Execution status.

Return values

0	Success.

Definition at line 115 of file clk_mx7srtc.c.

13.2.0.0.12 int set_net (struct tm * *tm,* int *cent_reg*)

Sets network rtc time.

Parameters

tm	Pointer to a time structure.
cent_reg	Unussed parameter.

Returns

Execution status.

Return values

-1	Failed.
0	Success.

Definition at line 121 of file clk_net.c.

Chapter 14

CAN - Controller Area Network driver (dev-can-imx)

This part describes the CAN driver.

Functions

- void can_tx (CANDEV_FLEXCAN *cdev, canmsg_t *txmsg)
- void can_debug (CANDEV_FLEXCAN *dev)
- void can_print_mailbox (CANDEV_FLEXCAN_INFO *devinfo)
- void can_print_reg (CANDEV_FLEXCAN_INFO *devinfo)
- void set_port32 (unsigned port, uint32_t mask, uint32_t data)
- const struct sigevent * can_intr (void *area, int id)
- void can_drvr_transmit (CANDEV *cdev)
- int can_drvr_devctl (CANDEV *cdev, int dcmd, DCMD_DATA *data)
- void can_init_intr (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit, uint32_t mdriver_intr)
- void can_init_hw (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)
- void device_init (int argc, char *argv[ ])
- void create_device (CANDEV_FLEXCAN_INIT *devinit)
- int main (int argc, char *argv[ ])
- void mdriver_print_data (CANDEV_FLEXCAN_INFO *devinfo)
- int mdriver_init (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)
- int mdriver_find_mbxid (CANDEV_FLEXCAN_INFO *devinfo, uint32_t canmid)
- void mdriver_init_data (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)

14.1 Detailed Description

This part describes the CAN driver. There are two FlexCAN peripherals on i.MX7 platform but only FlexCAN 2 is used on i.MX7D Sabre board. Start of the CAN driver (FlexCAN 2 device) is performed during the QNX boot sequence via following command in the build file. A default CAN bitrate is 250k.

```
#########################################################################
## CAN - FlexCAN 2 module used; Bitrate = 250k
#########################################################################
dev-can-imx -b250K can1
```

For possible input parameters see src/hardware/can/imx/dev-can-imx.use file.

For examples how to use CAN driver see src/hardware/can/imx/canimx.readme file.

14.2 Function Documentation

14.2.0.0.1 void can_debug (CANDEV_FLEXCAN * *dev*)

Print debug information.

Parameters

dev	Pointer to a device structure.

Definition at line 1052 of file canimx.c.

14.2.0.0.2 int can_drvr_devctl (CANDEV * *cdev,* int *dcmd,* DCMD_DATA * *data*)

LIBCAN driver devctl function.

Parameters

cdev	Pointer to a CAN device.
dcmd	CAN command.
data	Pointer to a CAN command data.

Return values

EOK	In case of success.
EINVAL	Invalid argument.
EXIT_FAILURE	FlexCAN Freeze Mode failed.
ENOTSUP	Command not supported.

Definition at line 415 of file canimx.c.

14.2.0.0.3 void can_drvr_transmit (CANDEV * *cdev*)

LIBCAN driver transmit function.

Parameters

cdev	Pointer to a CAN device.

Returns

none

Definition at line 387 of file canimx.c.

14.2.0.0.4 void can_init_hw (CANDEV_FLEXCAN_INFO * *devinfo,* CANDEV_FLEXCAN_INIT * *devinit*)

Initialize CAN device registers.

Parameters

devinfo	Pointer to a device info structure.
devinit	Pointer to a device init structure.

Definition at line 707 of file canimx.c.

14.2.0.0.5 void can_init_intr (CANDEV_FLEXCAN_INFO * *devinfo,* CANDEV_FLEXCAN_INIT * *devinit,* uint32_t *mdriver_intr*)

Initialize CAN device registers

Parameters

devinfo	Pointer to a device info structure.
devinit	Pointer to a device init structure.
mdriver_intr	Mini_driver active status flag.

Definition at line 656 of file canimx.c.

14.2.0.0.6 const struct sigevent* can_intr (void * *area,* int *id*)

CAN interrupt handler.

Parameters

area	Pointer to a CAN device info.
id	Mailbox ID.

Returns

Pointer to an event structure.

Definition at line 88 of file canimx.c.

14.2.0.0.7 void can_print_mailbox (CANDEV_FLEXCAN_INFO * *devinfo*)

Print CAN device mailbox memory.

Parameters

devinfo	Device info structure.

Definition at line 1088 of file canimx.c.

14.2.0.0.8 void can_print_reg (CANDEV_FLEXCAN_INFO * *devinfo*)

Print CAN device registers.

Parameters

devinfo	Pointer to a device info structure.

Definition at line 1065 of file canimx.c.

14.2.0.0.9 void can_tx (CANDEV_FLEXCAN * *dev,* canmsg_t * *txmsg*)

Transmit a CAN message from the specified mailbox.

Parameters

dev	Pointer to a CAN device structure.
txmsg	Pointer to a CAN message structure.

Definition at line 983 of file canimx.c.

14.2.0.0.10 void create_device (CANDEV_FLEXCAN_INIT * *devinit*)

Create CAN device.

Parameters

devinit	Pointer to a device init structure.

Definition at line 418 of file driver.c.

14.2.0.0.11 void device_init (int *argc,* char * *argv[ ]*)

Initialize CAN device.

Parameters

argc	Parameter passed from the main function.
argv	Parameter passed from the main function.

Definition at line 176 of file driver.c.

14.2.0.0.12 int main (int *argc,* char * *argv[ ]*)

CAN driver main function.

Parameters

argc	The count of total command line arguments passed to executable on execution.
argv	The array of character string of each command line argument passed to executable on execution.

Return values

0	In case of success.
-1	In case of error.

Definition at line 146 of file driver.c.

14.2.0.0.13 int mdriver_find_mbxid (CANDEV_FLEXCAN_INFO * *devinfo,* uint32_t *canmid*)

Function to search though all devices to find a matching message ID.

Parameters

devinfo	Pointer to a device info structure.
canmid	CAN message ID.

Returns

Matching mailbox ID or -1.

Definition at line 108 of file mdriver.c.

14.2.0.0.14 int mdriver_init (CANDEV_FLEXCAN_INFO * *devinfo,* CANDEV_FLEXCAN_INIT * *devinit*)

Function to search through the list of mini-drivers for one that matches our interrupt vector.

Parameters

devinfo	Pointer to a device info structure.
devinit	Pointer to a device init structure.

Returns

Matching interrupt vector or -1.

Definition at line 53 of file mdriver.c.

14.2.0.0.15 void mdriver_init_data (CANDEV_FLEXCAN_INFO * *devinfo,* CANDEV_FLEXCAN_INIT * *devinit*)

Function to add buffered mdriver CAN messages to the driver's message buffer. This function also sorts the messages into the appropriate device according to the message ID.

Parameters

devinfo	Pointer to a device info structure.
devinit	Pointer to a device init structure.

Definition at line 131 of file mdriver.c.

14.2.0.0.16 void mdriver_print_data (CANDEV_FLEXCAN_INFO * *devinfo*)

Function to print out the mdriver's status and buffered data.

Parameters

devinfo	Pointer to a device info structure.

Definition at line 204 of file mdriver.c.

14.2.0.0.17 void set_port32 (unsigned *port,* uint32_t *mask,* uint32_t *data*)

Function to modify a register.

Parameters

port	Register address.
mask	Clear mask.
data	Data to be written into a register.

Definition at line 75 of file canimx.c.

Chapter 15

Serial Asynchronous driver (devc-serial-imx)

This part describes the serial (devc) driver.

Data Structures

- struct my_pulse_struct

Functions

- int query_hwi_device (TTYINIT_MX1 *dip, unsigned unit)
- unsigned options (int argc, char *argv[ ])
- int my_attach_pulse (void **x, struct sigevent *event, void(*handler)(DEV_MX1 *dev, struct sigevent *event), DEV_MX1 *dev)
- int my_detach_pulse (void **x)
- int edit (TTYDEV *dev, unsigned c)
- int tto (TTYDEV *ttydev, int action, int arg1)
- void ser_stty (DEV_MX1 *dev)
- int drain_check (TTYDEV *ttydev, uintptr_t *count)

15.1 Detailed Description

This part describes the serial (devc) driver. There are five UART peripherals on i.MX7 but only UART1 is initialized by default (console functionality). By default start of the serial driver (UART1) is performed during the QNX boot sequence via following command in the build file. A default communicate speed is 115200 bauds.

```
######################################################################
## UART drivers
## UART 1 (started above) is connected to the USB->RS232 bridge
## UART 2,3,4,5 are unused
######################################################################
display_msg Starting serial driver (/dev/ser1)...
devc-serial-imx -e -F -S -c80000000 0x30860000,58
waitfor /dev/ser1 4
reopen /dev/ser1
```

For possible input parameters see src/hardware/devc/serial/imx/devc-serial-imx.use file.

15.2 Function Documentation

15.2.0.0.1 int drain_check (TTYDEV * *ttydev,* uintptr_t * *count*)

Check whether a device has drained.

Parameters

ttydev	Pointer to a tty device structure.
count	

Returns

Execution status

Return values

1	Device has drained.
0	Device has not drained.

Definition at line 590 of file tto.c.

15.2.0.0.2 int edit (TTYDEV * *dev,* unsigned *c*)

This function takes data from io-char's output buffer and gives it to the hardware. It also deals with stty commands, by calling ser_stty(), and provides line control and line status information.

Parameters

dev	Pointer to the driver's TTYDEV structure.
c	

Returns

Always 0.

Definition at line 41 of file tedit.c.

15.2.0.0.3 int my_attach_pulse (void ** *x,* struct sigevent * *event,* void(*)(DEV_MX1 *dev, struct sigevent *event) *handler,* DEV_MX1 * *dev*)

Attach a new pulse handler.

Parameters

x	Handler to the pulse structure.
event	Pointer to the event structure.
handler	Pointer to the handler to be attached.
dev	Serial device context structure.

Returns

Execution status.

Definition at line 88 of file pulse.c.

15.2.0.0.4 int my_detach_pulse (void ∗∗ *x*)

Detach a pulse handler.

Parameters

x	Handler to the pulse structure.

Returns

Execution status.

Definition at line 138 of file pulse.c.

15.2.0.0.5 unsigned options (int *argc,* char ∗ *argv[ ]*)

Parse input options and set device parameters.

Parameters

argc	The count of total command line arguments passed to executable on execution.
argv	The array of character string of each command line argument passed to executable on execution. •

Returns

Number of ports.

Definition at line 70 of file options.c.

15.2.0.0.6 int query_hwi_device (TTYINIT_MX1 ∗ *dip,* unsigned *unit*)

Specify parameters for default devices from hwi_info tags.

Parameters

dip	Pointer to a tty device structure.
unit	Index of the device.

Returns

Execution status.

Definition at line 41 of file options.c.

15.2.0.0.7 void ser_stty (DEV_MX1 * *dev*)

Configures registers that can be changed dynamically at runtime (baud, parity, stop bits, etc.).

Parameters

dev	Pointer to a device structure.

Definition at line 324 of file tto.c.

15.2.0.0.8 int tto (TTYDEV * *ttydev,* int *action,* int *arg1*)

This function takes data from io-char's output buffer and gives it to the hardware. It also deals with stty commands, by calling ser_stty(), and provides line control and line status information.

Parameters

ttydev	Pointer to the driver's TTYDEV structure.
action	One of: TTO_STTY - An stty command was received. It's called by io-char when the stty command is performed on the device. This action calls ser_stty(); the argument is ignored. TTO_CTRL - set the characteristics of the port i.e. control RS-232 modem lines. arg1 _SERCTL_BRK_CHG - called by io-char when the application requests a break such as tcsendbreak() be sent arg1 _SERCTL_DTR_CHG - changes the DTR line arg1 _SERCTL_RTS_CHG - changes the RTS line; io-char calls this to assert hardware flow control when the input buffer is filling up (based on the high-water level) TTO_LINESTATUS - a request for line status. Returns the status of the Modem Status and Modem Control registers when the user performs a devctl() with DCMD_CHR_LINESTATUS; the argument is ignored. TTO_DATA - used if tto() is called directly from the interrupt handler to transmit data or when io-char's write handler calls down to initiate a transfer. TTO_EVENT - used to call into the tto() at thread time to transmit data. The interrupt handler can return this event rather than calling tto() directly.
arg1	A data value which has different meanings for different actions. It's used to pass flags that modify the action.

Returns

Execution status

Definition at line 70 of file tto.c.

Chapter 16

SDMA library (libdma-sdma-imx7x)

This part describes SDMA library.

Data Structures

- struct sdma_bd_cmd_and_status_s
- struct sdma_bd_t
- struct sdma_ccb_t
- struct sdma_ch_ctx_t
- struct microcode_info_t
- struct sdma_scriptinfo_t
- struct sdma_shmem_t

Typedefs

- typedef struct sdma_bd_cmd_and_status_s sdma_bd_cmd_and_status_t

Functions

- sdma_chan_t * chan_create (unsigned ch_num)
- void chan_destroy (sdma_chan_t *chan_ptr)
- void register_init (void)
- int parse_init_options (const char *options)
- int parse_channel_options (sdma_chan_t *chan_ptr, const char *options)
- void callback_reenable_descr (unsigned ch_num)
- int sdma_init (const char *options)
- void sdma_fini (void)
- void sdma_query_channel (void *handle, dma_channel_query_t *chinfo)
- int sdma_driver_info (dma_driver_info_t *info)
- int sdma_channel_info (unsigned channel, dma_channel_info_t *info)
- void * sdma_channel_attach (const char *optstring, const struct sigevent *event, unsigned *channel, int prio, unsigned flags)
- void sdma_channel_release (void *handle)

- int sdma_setup_xfer (void *handle, const dma_transfer_t *tinfo)
- int sdma_xfer_start (void *handle)
- int sdma_xfer_abort (void *handle)
- unsigned sdma_bytes_left (void *handle)
- int sdma_xfer_complete (void *handle)
- int get_dmafuncs (dma_functions_t *functable, int tabsize)
- int sdmaram_script_load ()
- int sdmacmd_cmdch_create (void)
- void sdmacmd_cmdch_destroy (void)
- void sdmacmd_ctx_config (sdma_chan_t *chan_ptr)
- int sdmacmd_ctx_load (sdma_chan_t *chan_ptr)
- int sdmascript_lookup (sdma_scriptinfo_t *scriptinfo)
- void ctor (void)
- void dtor (void)
- const struct sigevent * irq_handler (void *area, int id)
- int sdmairq_init (uint32_t irq)
- void sdmairq_fini (void)
- void sdmairq_event_add (uint32_t channel, const struct sigevent *event)
- void sdmairq_event_remove (uint32_t channel)
- void sdmairq_callback_add (uint32_t channel, sdmairq_callback_t func_ptr)
- void sdmairq_callback_remove (uint32_t channel)
- int sdmasync_init (void)
- void sdmasync_fini (void)
- pthread_mutex_t * sdmasync_cmdmutex_get (void)
- pthread_mutex_t * sdmasync_libinit_mutex_get (void)
- pthread_mutex_t * sdmasync_regmutex_get (void)
- int sdmasync_is_first_process (void)
- int sdmasync_is_last_process (void)
- void sdmasync_process_cnt_incr (void)
- void sdmasync_process_cnt_decr (void)
- off64_t sdmasync_ccb_paddr_get (void)
- sdma_ccb_t * sdmasync_ccb_ptr_get (void)

16.1 Detailed Description

This part describes SDMA library. SDMA library is intended to be used by other drivers but it can be also used in appliaction e.g. for memory to memory transfers.

Library can be linked statically (BSP_root/src/lib/dma/sdma/imx7x/arm/a.le.v7/libdma-sdma-imx7x.a) or dynamically (BSP_root/src/lib/dma/sdma/imx7x/arm/so.le.v7/libdma-sdma-imx7x.so). For more information about libraries see chapter "Using libaries" in the "QNX Neutrino RTOS Programmer's Guide".

For examples how to use SDMA library see src/lib/dma/sdma/README.txt file.

16.2 Typedef Documentation

16.2.0.0.1 typedef struct sdma_bd_cmd_and_status_s sdma_bd_cmd_and_status_t

SDMA Buffer descriptor Command register bits definition *

16.3 Function Documentation

16.3.0.0.1 void callback_reenable_descr (unsigned *ch_num*)

Set DONE bit in all BDs for selected channel.

Parameters

ch_num	Channel number.

Definition at line 330 of file api.c.

16.3.0.0.2 sdma_chan_t* chan_create (unsigned *ch_num*)

Allocates all the data structures required by the channel. All structure member not explicitly initialized here are set to zero by default.

Parameters

ch_num	Channel number.

Returns

Pointer to the SDMA library Channel control structure.

Definition at line 96 of file api.c.

16.3.0.0.3 void chan_destroy (sdma_chan_t * *chan_ptr*)

Deallocate channel resources.

Parameters

chan_ptr	Pointer to the SDMA library Channel control structure.

Definition at line 176 of file api.c.

16.3.0.0.4 void ctor (void)

SDMA library constructor. It's run when a shared library is loaded.

Definition at line 122 of file init.c.

16.3.0.0.5 void dtor (void)

SDMA library destructor.

Definition at line 151 of file init.c.

16.3.0.0.6 int get_dmafuncs (dma_functions_t * *functable,* int *tabsize*)

Initialize function pointer table.

Parameters

functable	Function pointer table address.
tabsize	Function pointer table size.

Returns

Always returns 0.

Definition at line 845 of file api.c.

16.3.0.0.7 const struct sigevent* irq_handler (void * *area,* int *id*)

SDAM interrupt request handler.

Parameters

area	Not used.
id	Not used.

Returns

Address of event structure or NULL.

Definition at line 56 of file irq.c.

16.3.0.0.8 int parse_channel_options (sdma_chan_t * *chan_ptr,* const char * *options*)

Parse "options" string and sets "Channel control structure" parameters.

Parameters

chan_ptr	SDMA library Channel control structure pointer.
options	String containing additional channel options passed as the first argument of the sdma_channel_attach() method.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 277 of file api.c.

16.3.0.0.9 int parse_init_options (const char * *options*)

Parse command line driver parameters.

Parameters

options	Command line option string.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 219 of file api.c.

16.3.0.0.10 void register_init (void)

Initialize SDMA registers.

Definition at line 187 of file api.c.

16.3.0.0.11 unsigned sdma_bytes_left (void * *handle*)

This method should return number of bytes left for transfer. It works only for UART SDMA scripts only.

Parameters

handle	Channel handle.

Returns

Number of bytes not transferred yet.

Definition at line 784 of file api.c.

16.3.0.0.12 void* sdma_channel_attach (const char * *optstring,* const struct sigevent * *event,* unsigned * *channel,* int *prio,* unsigned *flags*)

Prepare chanel for data transfer.

Parameters

optstring	DMA channel attache optional parameters string.
event	Event signaled on interrupt.

Parameters

channel	Channel type variable address (Channel type value: e.g. SDMA script type, e.g. IMX_SDMA_CHTYPE_AP_2_AP).
prio	Channel priority in range [IMX_SDMA_CH_PRIO_LO(1)..IMX_SDMA_CH_PRIO_HI(7)].
flags	Channel attache flags (DMA_ATTACH_EVENT_ON_COMPLETE, DMA_ATTACH_EVENT_PER_SEGMENT and DMA_ATTACH_PRIORITY_HIGHEST are supported by this driver).

Definition at line 494 of file api.c.

16.3.0.0.13 int sdma_channel_info (unsigned *channel,* dma_channel_info_t * *info*)

Fill dma_channel_info_t structure.

Parameters

channel	Channel number.
info	Channel information structure address.

Returns

Always returns 0.

Definition at line 466 of file api.c.

16.3.0.0.14 void sdma_channel_release (void * *handle*)

Release resources allocated by sdma_channel_attach() methods.

Parameters

handle	Channel handle.

Definition at line 579 of file api.c.

16.3.0.0.15 int sdma_driver_info (dma_driver_info_t * *info*)

Fill dma_driver_info_t structure.

Parameters

info	Driver information structure address.

Returns

Always returns 0.

Definition at line 446 of file api.c.

16.3.0.0.16 void sdma_fini (void)

Driver cleanup function.

Definition at line 407 of file api.c.

16.3.0.0.17 int sdma_init (const char * *options*)

Initialize SDMA driver.

Parameters

options	Optional driver start option. Supported options are: regbase=0x30BD0000 and irq=34.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 352 of file api.c.

16.3.0.0.18 void sdma_query_channel (void * *handle,* dma_channel_query_t * *chinfo*)

Fill dma_channel_query_t structure.

Parameters

handle	Channel handle.
chinfo	Channel information structure address.

Definition at line 431 of file api.c.

16.3.0.0.19 int sdma_setup_xfer (void * *handle,* const dma_transfer_t * *tinfo*)

Prepare SDMA channel for transfer.

Parameters

handle	Channel handle.
tinfo	Transfer information structure address.

Returns

Always returns 0.

Definition at line 614 of file api.c.

16.3.0.0.20 int sdma_xfer_abort (void * *handle*)

Abort transfer.

Parameters

handle	Channel handle.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 750 of file api.c.

16.3.0.0.21 int sdma_xfer_complete (void * *handle*)

Transfer complete function. Set SDMA_HOSTOVRn = 0 and return state of error("R") bit.

Parameters

handle	Channel handle.

Returns

Error bit("R") state.

Definition at line 805 of file api.c.

16.3.0.0.22 int sdma_xfer_start (void * *handle*)

Start transfer.

Parameters

handle	Channel handle.

Returns

Always returns 0.

Definition at line 725 of file api.c.

16.3.0.0.23 int sdmacmd_cmdch_create (void)

Create the command channel descriptor buffer, and associate it with the command control block. The command channel lives at channel 0 and is used to transfer data to/from private SDMA memory. Currently, the command channel is only used to write contexts, but could be used for other purposes.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 170 of file cmd.c.

16.3.0.0.24 void sdmacmd_cmdch_destroy (void)

Release resources allocated by sdmacmd_cmdch_create() function.

Definition at line 257 of file cmd.c.

16.3.0.0.25 void sdmacmd_ctx_config (sdma_chan_t * *chan_ptr*)

Configure a 'Channel context' based on the channel type. The necessary Channel context configuration can be found in the SDMA Scripts Library Specification.

Parameters

chan_ptr	SDMA library Channel control structure pointer.

Definition at line 269 of file cmd.c.

16.3.0.0.26 int sdmacmd_ctx_load (sdma_chan_t * *chan_ptr*)

Load the 'context image' configured by the sdmacmd_ctx_config() function into SDMA private context memory.

Parameters

chan_ptr	SDMA library Channel control structure pointer.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 316 of file cmd.c.

16.3.0.0.27 void sdmairq_callback_add (uint32_t *channel,* sdmairq_callback_t *func_ptr*)

Attache callback function to the channel.

Parameters

channel	Channel number.
func_ptr	Callback function address.

Definition at line 137 of file irq.c.

16.3.0.0.28 void sdmairq_callback_remove (uint32_t *channel*)

Remove channel callback.

Parameters

channel	Channel number.

Definition at line 147 of file irq.c.

16.3.0.0.29 void sdmairq_event_add (uint32_t *channel,* const struct sigevent ∗ *event*)

Attached event to the channel.

Parameters

channel	Channel number.
event	

Definition at line 116 of file irq.c.

16.3.0.0.30 void sdmairq_event_remove (uint32_t *channel*)

Remove event from the channel.

Parameters

channel	Channel number.

Definition at line 126 of file irq.c.

16.3.0.0.31 void sdmairq_fini (void)

Deattache from the interrupt vector.

Definition at line 105 of file irq.c.

16.3.0.0.32 int sdmairq_init (uint32_t *irq*)

Attache to the interrupt vector and initialize event and callback arrays.

Parameters

irq	Interrupt vector number.

Returns

Always return 0.

Definition at line 88 of file irq.c.

16.3.0.0.33 int sdmaram_script_load ( )

Load a SDMA script into SDMA private context memory (experimental)

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 71 of file cmd.c.

16.3.0.0.34 int sdmascript_lookup (sdma_scriptinfo_t * *scriptinfo*)

Initializes sdma_scriptinfo_t structure.

Parameters

scriptinfo	- Address of sdma_scriptinfo_t structure to be initialized.

Returns

Always returns EOK.

Definition at line 60 of file script.c.

16.3.0.0.35 off64_t sdmasync_ccb_paddr_get (void)

Return physical address of CCB array.

Returns

Physical address of CCB array.

Definition at line 170 of file sync.c.

16.3.0.0.36 sdma_ccb_t * sdmasync_ccb_ptr_get (void)

Return CCB array address.

Returns

CCB array address.

Definition at line 180 of file sync.c.

16.3.0.0.37 pthread_mutex_t * sdmasync_cmdmutex_get (void)

Return command mutex address.

Returns

Command mutex address.

Definition at line 92 of file sync.c.

16.3.0.0.38 void sdmasync_fini (void)

Release driver resources.

Definition at line 79 of file sync.c.

16.3.0.0.39 int sdmasync_init (void)

Open and map shared memory.

Returns

Execution status.

Return values

0	Success.
-1	In case of any other error.

Definition at line 53 of file sync.c.

16.3.0.0.40 int sdmasync_is_first_process (void)

Check if this process is the first process.

Returns

Execution status.

Return values

0	First process.
1	More processes are running.

Definition at line 124 of file sync.c.

16.3.0.0.41 int sdmasync_is_last_process (void)

Check if this process is the last process.

Returns

Execution status.

Return values

0	Last process.
1	More processes are running.

Definition at line 140 of file sync.c.

16.3.0.0.42 pthread_mutex_t * sdmasync_libinit_mutex_get (void)

Return library mutex address.

Returns

Library mutex address.

Definition at line 102 of file sync.c.

16.3.0.0.43 void sdmasync_process_cnt_decr (void)

Decrement process counter.

Definition at line 160 of file sync.c.

16.3.0.0.44 void sdmasync_process_cnt_incr (void)

Increment process counter.

Definition at line 152 of file sync.c.

16.3.0.0.45 pthread_mutex_t * sdmasync_regmutex_get (void)

Return register mutex address.

Returns

Register mutex address.

Definition at line 112 of file sync.c.

Chapter 17

USB controller device mode mass storage mode DLL (devu-usbmass-imx-ci.so)

This part describes an USB controller device mode mass storage library usage.

Functions

- int imx_otg_init (chip_ideadc *dcctrl)
- int imx_otg_fini (chip_ideadc *dcctrl)
- void imx_extra_process_args_callout (chip_ideadc *dcctrl, char *options)
- void imx_sync_flush (void)

17.1 Detailed Description

This part describes an USB controller device mode mass storage library usage. USB Device controller driver(io-usb-dcd) example(s) can be started during QNX boot sequence by uncommenting following commands in the build file (name of the USB controler DLL is passed like a parameter into io-usb-dc stack). Note: All USB device mode exapmles are commented out in the build file and both USB controllers (USB OTG1 and USB OTG2) are started in host mode by default. Please don't start USB OTG1 controller in host mode if you uncomment a device mode example.

```
##### Example of Mass Storage device #####
# Step 1 - Create a ram disk
devb-ram ram capacity=16384,nodinit,cache=512k disk name=hd@10
waitfor /dev/hd10
fdisk /dev/hd10 add -t 6
mount -e /dev/hd10
waitfor /dev/hd10t6
mkdosfs /dev/hd10t6

# Step 2 - Start device stack
display_msg Starting USB OTG 1 controller in the device mode (/dev/io-usb-dcd/*)...
io-usb-dcd -dusbumass-imx-ci ioport=0x30B10000,irq=75
waitfor /dev/io-usb-dcd/io-usb 4
waitfor /dev/io-usb-dcd/devu-usbumass-imx-ci.so 4

# Step 3 - Start Mass Storage function driver and enable USB soft connect
devu-umass_client-block -l lun=0,devno=1,iface=0,fname=/dev/hd10
ulink_ctrl -l 1
```

17.2 Function Documentation

17.2.0.0.1 void imx_extra_process_args_callout (chip_ideadc * *dcctrl,* char * *options*)

Function used to analyze additional parameters passed to this dll.

Parameters

dcctrl	USB_OTG controller specific device data structure pointer.
options	optional parameters passed to this dll from command line.

Definition at line 101 of file imx.c.

17.2.0.0.2 int imx_otg_fini (chip_ideadc * *dcctrl*)

Function used to switch on VBUS (VBUS must be switched on in order to turn on Host mode).

Parameters

dcctrl	USB_OTG controller specific device data structure pointer.

Returns

Execution status.

Return values

EOK	Success.
errno	See mmap_device_memory() error codes.

Definition at line 71 of file imx.c.

17.2.0.0.3 int imx_otg_init (chip_ideadc * *dcctrl*)

Function used to switch off VBUS (VBUS is switch on in startup code).

Parameters

dcctrl	USB_OTG controller specific device data structure pointer.

Returns

Execution status.

Return values

EOK	Success.
errno	See mmap_device_memory() error codes.

Definition at line 41 of file imx.c.

17.2.0.0.4 void imx_sync_flush (void) [inline]

Data synchronization function.

Definition at line 116 of file imx.c.

Chapter 18

USB controller host mode DLL (devu-ehci-mx28.so)

This part describes an USB controller host mode library usage.

This part describes an USB controller host mode library usage. Start of USB Host controller driver(io-usb) is performed during QNX boot sequence via following commands in build file (name of the USB controller DLL is passed like a parameter into io-usb stack). Note: Both USB controllers are started in the host mode. Don't forget to remove OTG1 controller (ioport=0x30B10000,irq=75) from io-usb parameters if you plan to start some of the device mode examples.

```
io-usb -dehci-mx28 ioport=0x30B20100,irq=74,ioport=0x30B10100,irq=75
waitfor /dev/io-usb/io-usb 4
waitfor /dev/io-usb/devu-ehci-mx28.so 4
```

Starting Mass-storage devices

The devb-umass driver supports devices that follow the Mass Storage Class Specification. You can determine that the device is suitable by looking for the following information in the output from usb -vv:

```
# usb -vv
USB 0 (EHCI) v1.10, v1.01 DDK, v1.01 HCD
    Control, Interrupt, Bulk(SG), Isoch(Stream), High speed

Device Address             : 1
Upstream Host Controller   : 0
Upstream Device Address    : 0
Upstream Port              : 1
Upstream Port Speed        : High
Vendor                     : 0x1005
Product                    : 0xb113 (USB Flash Drive)
Device Release             : r1.00
USB Spec Release           : v2.00
Serial Number              : 070200010914A70
Class                      : 0x00 (Independent per interface)
Max PacketSize0            : 64
Languages                  : 0x0409 (English)
Configurations             : 1
  Configuration            : 1
    Attributes             : 0x80 (Bus-powered)
    Max Power              : 500 mA
    Interfaces             : 1
      Interface            : 0 / 0
        Class              : 0x08 (Mass Storage)
        Subclass           : 0x06 (SCSI)
        Protocol           : 0x50
```

To use a USB mass-storage device on a Neutrino system, start io-usb as described above, then the devb-umass driver. By default, this driver creates an entry for disk-based devices in /dev in the form /dev/hdn, where n is the drive number. Once you've started the driver, you can treat the device like a disk.

For example, for a mass-storage device, type:

```
devb-umass cam pnp
```

To acces files on attached device you have to mount it. You can do it by the following command:

```
mount -t dos /dev/hd0 /mnt/disk0
```

Chapter 19

Data Structure Documentation

19.1 _apbh_dma_gpmi1_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.1.1 Detailed Description

Define the APBH DMA structure with 1 GPMI Parameter word writes.

Definition at line 106 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.2 _apbh_dma_gpmi3_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.2.1 Detailed Description

Define the APBH DMA structure with 3 GPMI Parameter word writes.

Definition at line 114 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.3 _apbh_dma_gpmi5_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.3.1 Detailed Description

Define the APBH DMA structure with 5 GPMI Parameter word writes.

Definition at line 124 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.4 _apbh_dma_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.4.1 Detailed Description

Define the APBH DMA structure without GPMI transfers.

Definition at line 99 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.5 _chipio_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

Data Fields

- uint8_t * v_data_fs_buf
- uint8_t * v_data_page_buf

19.5.1 Detailed Description

Low level driver structure

Definition at line 294 of file chipio.h.

19.5.2 Field Documentation

19.5.2.1 uint8_t* _chipio_t::v_data_fs_buf

This is intended for ECC engine. Buffer contains only data and fs meta

Definition at line 304 of file chipio.h.

19.5.2.2 uint8_t* _chipio_t::v_data_page_buf

This is intended for RAW writes. Buffer contains data, ecc and fs meta

Definition at line 306 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.6 _dma_blk_erase_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.6.1 Detailed Description

DMA chain structure for device erase block.

Definition at line 136 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.7 _dma_programEcc_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.7.1 Detailed Description

DMA chain structure for NAND Program.

Definition at line 154 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.8 _dma_programRaw_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.8.1 Detailed Description

DMA chain structure for NAND Program.

Definition at line 177 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.9 _dma_read_id_device_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.9.1 Detailed Description

DMA chain structure for Read ID

Definition at line 278 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.10 _dma_readEcc_device_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.10.1 Detailed Description

DMA chain structure for Raw Read Page

Definition at line 198 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.11 _dma_readRaw_device_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.11.1 Detailed Description

DMA chain structure for Raw Read Page

Definition at line 222 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.12 _dma_reset_device_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.12.1 Detailed Description

DMA chain structure for Reset Device

Definition at line 261 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.13 _imx_dev Struct Reference

`#include <src/hardware/i2c/imx/proto.h>`

Data Fields

- unsigned reglen
- uintptr_t regbase
- unsigned physbase
- int intr
- int iid
- struct sigevent intrevent
- unsigned slave_addr
- unsigned own_addr
- i2c_addrfmt_t slave_addr_fmt
- unsigned restart
- unsigned input_clk
- unsigned speed
- unsigned i2c_freq_val

19.13.1 Detailed Description

I2C driver device structure.

Definition at line 52 of file proto.h.

19.13.2 Field Documentation

19.13.2.1 unsigned _imx_dev::i2c_freq_val

I2C IC register value

Definition at line 66 of file proto.h.

19.13.2.2 int _imx_dev::iid

Interrupt event ID returned by InterruptAttachEvent()

Definition at line 58 of file proto.h.

19.13.2.3 unsigned _imx_dev::input_clk

I2C peripheral input clock frequency in Hz

Definition at line 64 of file proto.h.

19.13.2.4 int _imx_dev::intr

I2C interrupt event number from reference manual

Definition at line 57 of file proto.h.

19.13.2.5 struct sigevent _imx_dev::intrevent

sigevent structure which is delivered when I2C interrupt occurs

Definition at line 59 of file proto.h.

19.13.2.6 unsigned _imx_dev::own_addr

Own I2C address in slave mode

Definition at line 61 of file proto.h.

19.13.2.7 unsigned _imx_dev::physbase

Base address of I2C peripheral registers

Definition at line 56 of file proto.h.

19.13.2.8 uintptr_t _imx_dev::regbase

Virtual address of I2C peripheral registers mapped by mmap_device_io()

Definition at line 55 of file proto.h.

19.13.2.9 unsigned _imx_dev::reglen

Length of mapped memory area which contains I2C registers

Definition at line 54 of file proto.h.

19.13.2.10 unsigned _imx_dev::restart

Determines if repeated Start condition is generated

Definition at line 63 of file proto.h.

19.13.2.11 unsigned _imx_dev::slave_addr

Slave device address

Definition at line 60 of file proto.h.

19.13.2.12 i2c_addrfmt_t _imx_dev::slave_addr_fmt

Determines slave address format 7-bit or 10-bit

Definition at line 62 of file proto.h.

19.13.2.13 unsigned _imx_dev::speed

I2C bus speed in Hz

Definition at line 65 of file proto.h.

The documentation for this struct was generated from the following file:

- src/hardware/i2c/imx/proto.h

19.14 _imx_ext Struct Reference

`#include <src/hardware/devb/sdmmc/arm/imx.le.v7/bs.h>`

Data Fields

- int emmc
- int nocd
- int cd_irq
- int cd_iid
- unsigned cd_pbase
- uintptr_t cd_base
- int cd_pin
- unsigned wp_pbase
- uintptr_t wp_base
- int wp_pin
- int bw
- int vdd1_8
- int rs_pbase
- int rs_pin

19.14.1 Detailed Description

Structure describing board specific configuration

Definition at line 40 of file bs.h.

19.14.2 Field Documentation

19.14.2.1 int _imx_ext::bw

Data bus width

Definition at line 51 of file bs.h.

19.14.2.2 uintptr_t _imx_ext::cd_base

CD I/O base address

Definition at line 46 of file bs.h.

19.14.2.3 int _imx_ext::cd_iid

CD GPIO interrupt id

Definition at line 44 of file bs.h.

19.14.2.4 int _imx_ext::cd_irq

CD GPIO IRQ

Definition at line 43 of file bs.h.

19.14.2.5 unsigned _imx_ext::cd_pbase

CD I/O base physical address

Definition at line 45 of file bs.h.

19.14.2.6 int _imx_ext::cd_pin

CD I/O bit number

Definition at line 47 of file bs.h.

19.14.2.7 int _imx_ext::emmc

If non-0, implies "nocd" option as well

Definition at line 41 of file bs.h.

19.14.2.8 int _imx_ext::nocd

If non-0, indicates CD is not supported

Definition at line 42 of file bs.h.

19.14.2.9 int _imx_ext::rs_pbase

Reset GPIO pin base address

Definition at line 53 of file bs.h.

19.14.2.10 int _imx_ext::rs_pin

Reset GPIO pin number

Definition at line 54 of file bs.h.

19.14.2.11 int _imx_ext::vdd1_8

1.8V support

Definition at line 52 of file bs.h.

19.14.2.12 uintptr_t _imx_ext::wp_base

Mapped WP I/O base address

Definition at line 49 of file bs.h.

19.14.2.13 unsigned _imx_ext::wp_pbase

WP I/O base physical address

Definition at line 48 of file bs.h.

19.14.2.14 int _imx_ext::wp_pin

WP I/O bit number

Definition at line 50 of file bs.h.

The documentation for this struct was generated from the following file:

- src/hardware/devb/sdmmc/arm/imx.le.v7/bs.h

19.15 _imx_qspi_t Struct Reference

`#include <src/hardware/flash/boards/qspi-imx/imx_fc_qspi.h>`

19.15.1 Detailed Description

Low level driver handle

Definition at line 91 of file imx_fc_qspi.h.

The documentation for this struct was generated from the following file:

- src/hardware/flash/boards/qspi-imx/imx_fc_qspi.h

19.16 _imx_sdhcx_adma32_t Struct Reference

`#include <src/hardware/devb/sdmmc/sdiodi/hc/imx_hc.h>`

19.16.1 Detailed Description

32 bit ADMA descriptor definition

Definition at line 46 of file imx_hc.h.

The documentation for this struct was generated from the following file:

- src/hardware/devb/sdmmc/sdiodi/hc/imx_hc.h

19.17 _imx_usdhcx_hc Struct Reference

`#include <src/hardware/devb/sdmmc/sdiodi/hc/imx_hc.h>`

Data Fields

- int sdma_iid

19.17.1 Detailed Description

Processor specific structure

Definition at line 61 of file imx_hc.h.

19.17.2 Field Documentation

19.17.2.1 int _imx_usdhcx_hc::sdma_iid

SDMA interrupt id

Definition at line 73 of file imx_hc.h.

The documentation for this struct was generated from the following file:

- src/hardware/devb/sdmmc/sdiodi/hc/imx_hc.h

19.18 _NAND_dma_read_status_device_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/chipio.h>`

19.18.1 Detailed Description

DMA chain structure for Read Status.

Definition at line 244 of file chipio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/chipio.h

19.19 _spare_t Struct Reference

`#include <src/hardware/etfs/nand4096/imx-micron/devio.h>`

Data Fields

- uint8_t status
- uint8_t status2
- uint16_t nclusters
- uint8_t align0 [4]
- uint32_t sequence
- uint16_t fid
- uint8_t align1 [2]
- uint32_t cluster
- uint8_t align2 [4]
- uint32_t erasesig [2]

19.19.1 Detailed Description

NAND spare area for BCH engine

Definition at line 57 of file devio.h.

19.19.2 Field Documentation

19.19.2.1 uint8_t _spare_t::align0[4]

4 bytes align since we have 8 byte of meta-data in each sub-sector

Definition at line 63 of file devio.h.

19.19.2.2 uint8_t _spare_t::align1[2]

2 bytes align since we have 8 byte of meta-data in each sub-sector

Definition at line 69 of file devio.h.

19.19.2.3 uint8_t _spare_t::align2[4]

4 bytes align since we have 8 byte of meta-data in each sub-sector

Definition at line 74 of file devio.h.

19.19.2.4 uint32_t _spare_t::cluster

Cluster number

Definition at line 73 of file devio.h.

19.19.2.5 uint32_t _spare_t::erasesig[2]

The erase signature created by devio_eraseblk

Definition at line 78 of file devio.h.

19.19.2.6 uint16_t _spare_t::fid

File id

Definition at line 68 of file devio.h.

19.19.2.7 uint16_t _spare_t::nclusters

Number of clusters

Definition at line 62 of file devio.h.

19.19.2.8 uint32_t _spare_t::sequence

Sequence number

Definition at line 67 of file devio.h.

19.19.2.9 uint8_t _spare_t::status

Factory marking for bad block (0xff == GOOD)

Definition at line 60 of file devio.h.

19.19.2.10 uint8_t _spare_t::status2

For 16 bit wide parts

Definition at line 61 of file devio.h.

The documentation for this struct was generated from the following file:

- src/hardware/etfs/nand4096/imx-micron/devio.h

19.20 imx_card Struct Reference

`#include <src/hardware/deva/ctrl/mx/imx_sai_dll.h>`

Data Fields

- ado_mutex_t lock
- ado_pcm_t * pcm
- ado_mixer_t * mixer
- imx_stream_t strm [2]
- dma_functions_t sdmafuncs
- imx_sai_data_t sai
- char mixeropts [100+1]
- uint8_t i2c_dev
- uint32_t sys_clk

19.20.1 Detailed Description

IMX Card data structure.

Definition at line 134 of file imx_sai_dll.h.

19.20.2 Field Documentation

19.20.2.1 uint8_t imx_card::i2c_dev

i2c device instance number.

Definition at line 145 of file imx_sai_dll.h.

19.20.2.2 ado_mutex_t imx_card::lock

Mutex for hardware and common data lock.

Definition at line 135 of file imx_sai_dll.h.

19.20.2.3 ado_mixer_t* imx_card::mixer

Mixer data.

Definition at line 137 of file imx_sai_dll.h.

19.20.2.4 char imx_card::mixeropts[100+1]

Mixer Specific Options.

Definition at line 143 of file imx_sai_dll.h.

19.20.2.5 ado_pcm_t* imx_card::pcm

PCM data.

Definition at line 136 of file imx_sai_dll.h.

19.20.2.6 imx_sai_data_t imx_card::sai

SAI peripheral data.

Definition at line 140 of file imx_sai_dll.h.

19.20.2.7 dma_functions_t imx_card::sdmafuncs

DMA functions.

Definition at line 139 of file imx_sai_dll.h.

19.20.2.8 imx_stream_t imx_card::strm[2]

Tx and Rx stream structure.

Definition at line 138 of file imx_sai_dll.h.

19.20.2.9 uint32_t imx_card::sys_clk

System clock frequency in Hz.

Definition at line 146 of file imx_sai_dll.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/imx_sai_dll.h

19.21 imx_qspi_lutx_t Union Reference

`#include <src/hardware/flash/boards/qspi-imx/imx_fc_qspi.h>`

19.21.1 Detailed Description

LUT entry

Definition at line 35 of file imx_fc_qspi.h.

The documentation for this union was generated from the following file:

- src/hardware/flash/boards/qspi-imx/imx_fc_qspi.h

19.22 imx_sai_data Struct Reference

`#include <src/hardware/deva/ctrl/mx/imx_sai_dll.h>`

Data Fields

- uint32_t base
- imx_sai_t * reg
- imx_sai_sync_mode_t xfer_sync_mode
- imx_sai_xfer_config_t cfg [2]

19.22.1 Detailed Description

IMX SAI data structure.

Definition at line 121 of file imx_sai_dll.h.

19.22.2 Field Documentation

19.22.2.1 uint32_t imx_sai_data::base

SAI Peripheral base address.

Definition at line 122 of file imx_sai_dll.h.

19.22.2.2 imx_sai_xfer_config_t imx_sai_data::cfg[2]

SAI transfer config for Tx and Rx.

Definition at line 129 of file imx_sai_dll.h.

19.22.2.3 imx_sai_t* imx_sai_data::reg

SAI Peripheral registers structure Tx:[0] Rx:[1].

Definition at line 123 of file imx_sai_dll.h.

19.22.2.4 imx_sai_sync_mode_t imx_sai_data::xfer_sync_mode

SAI TX RX xfer synchronization.

Definition at line 128 of file imx_sai_dll.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/imx_sai_dll.h

19.23 imx_sai_xfer_config Struct Reference

`#include <src/hardware/deva/ctrl/mx/imx_sai_dll.h>`

Data Fields

- imx_sai_mode_t mode
- imx_sai_protocol_t protocol
- int sample_rate
- int sample_rate_min
- int sample_rate_max
- int sample_size
- int voices
- int nslots
- uint8_t clk_pol
- uint8_t sync_pol
- uint8_t bit_delay
- uint8_t sync_len
- uint8_t msel

19.23.1 Detailed Description

IMX SAI transfer configuration structure.

Definition at line 104 of file imx_sai_dll.h.

19.23.2 Field Documentation

19.23.2.1 uint8_t imx_sai_xfer_config::bit_delay

Bit clock delay 0,1. Allows early frame sync.

Definition at line 115 of file imx_sai_dll.h.

19.23.2.2 uint8_t imx_sai_xfer_config::clk_pol

Bit clock polarity.

Definition at line 113 of file imx_sai_dll.h.

19.23.2.3 imx_sai_mode_t imx_sai_xfer_config::mode

SAI Master/Slave mode.

Definition at line 105 of file imx_sai_dll.h.

19.23.2.4 uint8_t imx_sai_xfer_config::msel

MCLK select. See chip specific information in RM.

Definition at line 117 of file imx_sai_dll.h.

19.23.2.5 int imx_sai_xfer_config::nslots

Number of slots.

Definition at line 112 of file imx_sai_dll.h.

19.23.2.6 imx_sai_protocol_t imx_sai_xfer_config::protocol

SAI protocol.

Definition at line 106 of file imx_sai_dll.h.

19.23.2.7 int imx_sai_xfer_config::sample_rate

SAI sample rate in Hz.

Definition at line 107 of file imx_sai_dll.h.

19.23.2.8 int imx_sai_xfer_config::sample_rate_max

Highest supported sample rate in Hz.

Definition at line 109 of file imx_sai_dll.h.

19.23.2.9 int imx_sai_xfer_config::sample_rate_min

Lowest supported sample rate in Hz.

Definition at line 108 of file imx_sai_dll.h.

19.23.2.10 int imx_sai_xfer_config::sample_size

Sample size.

Definition at line 110 of file imx_sai_dll.h.

19.23.2.11 uint8_t imx_sai_xfer_config::sync_len

Frame sync len.

Definition at line 116 of file imx_sai_dll.h.

19.23.2.12 uint8_t imx_sai_xfer_config::sync_pol

Bit clock synchronization polarity.

Definition at line 114 of file imx_sai_dll.h.

19.23.2.13 int imx_sai_xfer_config::voices

Number of voices.

Definition at line 111 of file imx_sai_dll.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/imx_sai_dll.h

19.24 imx_stream Struct Reference

`#include <src/hardware/deva/ctrl/mx/imx_sai_dll.h>`

Data Fields

- imx_stream_pcm_t pcm
- volatile imx_stream_status_t status
- imx_stream_dma_t dma

19.24.1 Detailed Description

IMX stream data structure.

Definition at line 77 of file imx_sai_dll.h.

19.24.2 Field Documentation

19.24.2.1 imx_stream_dma_t imx_stream::dma

DMA related data.

Definition at line 80 of file imx_sai_dll.h.

19.24.2.2 imx_stream_pcm_t imx_stream::pcm

ADO PCM stream data.

Definition at line 78 of file imx_sai_dll.h.

19.24.2.3 volatile imx_stream_status_t imx_stream::status

Stream status.

Definition at line 79 of file imx_sai_dll.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/imx_sai_dll.h

19.25 imx_stream_dma Struct Reference

`#include <src/hardware/deva/ctrl/mx/imx_sai_dll.h>`

Data Fields

- uint32_t event_num
- uint32_t chnl_type
- void * chn
- dma_addr_t * buf
- struct sigevent event
- void * pulse

19.25.1 Detailed Description

IMX stream DMA data.

Definition at line 59 of file imx_sai_dll.h.

19.25.2 Field Documentation

19.25.2.1 dma_addr_t* imx_stream_dma::buf

DMA buffer allocated by the driver.

Definition at line 63 of file imx_sai_dll.h.

19.25.2.2 void* imx_stream_dma::chn

DMA channel for transfer.

Definition at line 62 of file imx_sai_dll.h.

19.25.2.3 uint32_t imx_stream_dma::chnl_type

DMA channel type.

Definition at line 61 of file imx_sai_dll.h.

19.25.2.4 struct sigevent imx_stream_dma::event

DMA event associated with SAI.

Definition at line 64 of file imx_sai_dll.h.

19.25.2.5 uint32_t imx_stream_dma::event_num

DMA peripheral request number.

Definition at line 60 of file imx_sai_dll.h.

19.25.2.6 void* imx_stream_dma::pulse

DMA pulse handler for pulse receive.

Definition at line 65 of file imx_sai_dll.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/imx_sai_dll.h

19.26 imx_stream_pcm Struct Reference

`#include <src/hardware/deva/ctrl/mx/imx_sai_dll.h>`

Data Fields

- ado_pcm_subchn_t * subchn
- ado_pcm_cap_t caps
- ado_pcm_hw_t funcs
- ado_pcm_config_t * config

19.26.1 Detailed Description

PCM stream data structure.

Definition at line 69 of file imx_sai_dll.h.

19.26.2 Field Documentation

19.26.2.1 ado_pcm_cap_t imx_stream_pcm::caps

Data structure of capabilities of a PCM device.

Definition at line 71 of file imx_sai_dll.h.

19.26.2.2 ado_pcm_config_t* imx_stream_pcm::config

Data structure that describes the configuration of a PCM subchannel.

Definition at line 73 of file imx_sai_dll.h.

19.26.2.3 ado_pcm_hw_t imx_stream_pcm::funcs

Data structure of callbacks for PCM devices.

Definition at line 72 of file imx_sai_dll.h.

19.26.2.4 ado_pcm_subchn_t* imx_stream_pcm::subchn

Pointer to subchannel structure.

Definition at line 70 of file imx_sai_dll.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/imx_sai_dll.h

19.27 microcode_info_t Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

Data Fields

- const uint16_t * p
- uint32_t addr
- uint32_t size

19.27.1 Detailed Description

Microcode info structure (SDMA image info).

Definition at line 225 of file sdma.h.

19.27.2 Field Documentation

19.27.2.1 uint32_t microcode_info_t::addr

Address of the SDMA RAM image in the SDMA engine address space.

Definition at line 227 of file sdma.h.

19.27.2.2 const uint16_t* microcode_info_t::p

Address of the SDMA RAM image in the process address space.

Definition at line 226 of file sdma.h.

19.27.2.3 uint32_t microcode_info_t::size

Size of SDMA RAM image.

Definition at line 228 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.28 my_pulse_struct Struct Reference

19.28.1 Detailed Description

Pulse structure.

Definition at line 37 of file pulse.c.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/pulse.c

19.29 sdma_bd_cmd_and_status_s Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

19.29.1 Detailed Description

SDMA Buffer descriptor Command register bits definition ∗

Definition at line 168 of file sdma.h.

19.29.2 Field Documentation

19.29.2.1 uint32_t sdma_bd_cmd_and_status_s::C

Wrap. Indicates if this buffer descriptor is the last one for the channel control block.

Definition at line 175 of file sdma.h.

19.29.2.2 uint32_t sdma_bd_cmd_and_status_s::COMMAND

Extended. When this bit is set, the extended buffer address of the buffer descriptor is used.

Definition at line 181 of file sdma.h.

19.29.2.3 uint32_t sdma_bd_cmd_and_status_s::D

Count. Indicates the size of the data to be transmitted, the size of the data buffer pointed to by the buffer descriptor.

Definition at line 173 of file sdma.h.

19.29.2.4 uint32_t sdma_bd_cmd_and_status_s::E

Reserved

Definition at line 180 of file sdma.h.

19.29.2.5 uint32_t sdma_bd_cmd_and_status_s::I

Continuous. This buffer is allowed to receive multiple transmit buffers or is allowed to transmit to multiple receive buffers.

Definition at line 176 of file sdma.h.

19.29.2.6 uint32_t sdma_bd_cmd_and_status_s::L

Error. Indicates an error occurred on the channel's buffer descriptor requested command.

Definition at line 178 of file sdma.h.

19.29.2.7 uint32_t sdma_bd_cmd_and_status_s::R

Interrupt. When SDMA has finished to process data transfer attached to this buffer descriptor, send an interrupt to the ARM platform.

Definition at line 177 of file sdma.h.

19.29.2.8 uint32_t sdma_bd_cmd_and_status_s::Reserved_22

Last Buffer Descriptor.

Definition at line 179 of file sdma.h.

19.29.2.9 uint32_t sdma_bd_cmd_and_status_s::W

Done. Indicates the "ownership" of the buffer descriptor. When D=0 the ARM owns the buffer descriptor; when D=1 SDMA owns the buffer descriptor.

Definition at line 174 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.30 sdma_bd_t Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

Data Fields

- uint32_t buf_paddr
- uint32_t ext_buf_paddr

19.30.1 Detailed Description

Buffer Descriptor Structure

Definition at line 189 of file sdma.h.

19.30.2 Field Documentation

19.30.2.1 uint32_t sdma_bd_t::buf_paddr

Command and status register.

Definition at line 191 of file sdma.h.

19.30.2.2 uint32_t sdma_bd_t::ext_buf_paddr

Buffer physical address.

Definition at line 192 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.31 sdma_ccb_t Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

19.31.1 Detailed Description

Channel Control Block Structure.

Definition at line 203 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.32 sdma_ch_ctx_t Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

Data Fields

- uint32_t pc
- uint32_t spc
- uint32_t g_reg [8]
- uint32_t dma_xfer_regs [14]
- uint32_t scratch [8]

19.32.1 Detailed Description

SDMA Channel context structure. It contains SDMA internal registers (General registers, pc, epc, rpc, DF, SF, T, MSA, MDA, ...).

Definition at line 214 of file sdma.h.

19.32.2 Field Documentation

19.32.2.1 uint32_t sdma_ch_ctx_t::dma_xfer_regs[14]

Functional units state registers

Definition at line 218 of file sdma.h.

19.32.2.2 uint32_t sdma_ch_ctx_t::g_reg[8]

General purpose registers

Definition at line 217 of file sdma.h.

19.32.2.3 uint32_t sdma_ch_ctx_t::pc

Bits description: 31:SF, 29-16:RPC, 15:T, 13-0:PC

Definition at line 215 of file sdma.h.

19.32.2.4 uint32_t sdma_ch_ctx_t::scratch[8]

Scratch RAM

Definition at line 219 of file sdma.h.

19.32.2.5 uint32_t sdma_ch_ctx_t::spc

Bits description: 31-30:LM, 29-16:EPC, 15:DF, 13-0:SPC

Definition at line 216 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.33 sdma_scriptinfo_t Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

Data Fields

- uint32_t script_addr_arr [255]
- microcode_info_t ram_microcode_info

19.33.1 Detailed Description

SDMA scripts info structure

Definition at line 234 of file sdma.h.

19.33.2 Field Documentation

19.33.2.1 microcode_info_t sdma_scriptinfo_t::ram_microcode_info

SDMA RAM image info

Definition at line 236 of file sdma.h.

19.33.2.2 uint32_t sdma_scriptinfo_t::script_addr_arr[255]

SDMA scripts address array

Definition at line 235 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.34 sdma_shmem_t Struct Reference

`#include <src/lib/dma/sdma/sdma.h>`

Data Fields

- off64_t paddr64
- sdma_ccb_t ccb_arr [32]

19.34.1 Detailed Description

SDAM library shared data structure.

Definition at line 242 of file sdma.h.

19.34.2 Field Documentation

19.34.2.1 sdma_ccb_t sdma_shmem_t::ccb_arr[32]

Channel control block(CCB) array.

Definition at line 248 of file sdma.h.

19.34.2.2 off64_t sdma_shmem_t::paddr64

Physical address of CCB array (ccb_arr).

Definition at line 247 of file sdma.h.

The documentation for this struct was generated from the following file:

- src/lib/dma/sdma/sdma.h

19.35 wm8960_context Struct Reference

`#include <src/hardware/deva/ctrl/mx/nto/arm/dll.le.v7.sai_wm8960/wm8960.h>`

Data Fields

- char i2c_num
- int i2c_fd
- uint32_t mclk
- uint32_t sample_rate
- uint8_t sample_size
- uint8_t use_dac_lrck
- uint16_t dac_mute
- uint16_t adc_mute
- uint16_t hp_mute
- uint16_t spk_mute
- uint8_t spk_volume [2]
- uint8_t hp_volume [2]
- uint8_t dac_volume [2]
- uint8_t adc_volume [2]
- uint16_t regs [56]

19.35.1 Detailed Description

WM8960 mixer context structure

Definition at line 291 of file wm8960.h.

19.35.2 Field Documentation

19.35.2.1 uint16_t wm8960_context::adc_mute

ADC mute state

Definition at line 301 of file wm8960.h.

19.35.2.2 uint8_t wm8960_context::adc_volume[2]

ADC volume

Definition at line 308 of file wm8960.h.

19.35.2.3 uint16_t wm8960_context::dac_mute

DAC mute state

Definition at line 300 of file wm8960.h.

19.35.2.4 uint8_t wm8960_context::dac_volume[2]

DAC volume

Definition at line 307 of file wm8960.h.

19.35.2.5 uint16_t wm8960_context::hp_mute

Headphone mute state

Definition at line 302 of file wm8960.h.

19.35.2.6 uint8_t wm8960_context::hp_volume[2]

Headphone volume

Definition at line 306 of file wm8960.h.

19.35.2.7 int wm8960_context::i2c_fd

I2C device handle

Definition at line 294 of file wm8960.h.

19.35.2.8 char wm8960_context::i2c_num

I2C bus number

Definition at line 293 of file wm8960.h.

19.35.2.9 uint32_t wm8960_context::mclk

External SYS_MCLK frequency

Definition at line 295 of file wm8960.h.

19.35.2.10 uint16_t wm8960_context::regs[56]

Stores WM8960 register states

Definition at line 310 of file wm8960.h.

19.35.2.11 uint32_t wm8960_context::sample_rate

Sample rate

Definition at line 296 of file wm8960.h.

19.35.2.12 uint8_t wm8960_context::sample_size

Sample size

Definition at line 297 of file wm8960.h.

19.35.2.13 uint16_t wm8960_context::spk_mute

Speaker mute state a

Definition at line 303 of file wm8960.h.

19.35.2.14 uint8_t wm8960_context::spk_volume[2]

Speaker volume

Definition at line 305 of file wm8960.h.

19.35.2.15 uint8_t wm8960_context::use_dac_lrck

Set to 1 when ADC uses DAC LRCK pin

Definition at line 298 of file wm8960.h.

The documentation for this struct was generated from the following file:

- src/hardware/deva/ctrl/mx/nto/arm/dll.le.v7.sai_wm8960/wm8960.h

Chapter 20

File Documentation

20.1 src/hardware/can/imx/canimx.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <errno.h>
#include <malloc.h>
#include <string.h>
#include <devctl.h>
#include <atomic.h>
#include <unistd.h>
#include <gulliver.h>
#include <sys/mman.h>
#include <hw/inout.h>
#include "canimx.h"
#include "proto.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

Functions

- void can_tx (CANDEV_FLEXCAN *cdev, canmsg_t *txmsg)
- void can_debug (CANDEV_FLEXCAN *dev)
- void can_print_mailbox (CANDEV_FLEXCAN_INFO *devinfo)
- void can_print_reg (CANDEV_FLEXCAN_INFO *devinfo)
- void set_port32 (unsigned port, uint32_t mask, uint32_t data)
- const struct sigevent * can_intr (void *area, int id)
- void can_drvr_transmit (CANDEV *cdev)
- int can_drvr_devctl (CANDEV *cdev, int dcmd, DCMD_DATA *data)
- void can_init_intr (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit, uint32_t mdriver_intr)
- void can_init_hw (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)

20.2 src/hardware/can/imx/canimx.h File Reference

```
#include <hw/imx_can.h>
```

Functions

- void can_drvr_transmit (CANDEV *cdev)
- int can_drvr_devctl (CANDEV *cdev, int dcmd, DCMD_DATA *data)
- void can_init_hw (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)
- void can_init_intr (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit, uint32_t mdriver_intr)
- void can_print_reg (CANDEV_FLEXCAN_INFO *devinfo)
- void can_print_mailbox (CANDEV_FLEXCAN_INFO *devinfo)
- void set_port32 (unsigned port, uint32_t mask, uint32_t data)

20.3 src/hardware/can/imx/driver.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <malloc.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <hw/inout.h>
#include <sys/syspage.h>
#include <inttypes.h>
#include <hw/sysinfo.h>
#include <drvr/hwinfo.h>
#include "canimx.h"
#include "proto.h"
```

Functions

- void device_init (int argc, char *argv[])
- void create_device (CANDEV_FLEXCAN_INIT *devinit)
- int main (int argc, char *argv[])

20.4 src/hardware/can/imx/mdriver.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <malloc.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <hw/inout.h>
#include <sys/syspage.h>
#include "canimx.h"
```

Functions

- void mdriver_print_data (CANDEV_FLEXCAN_INFO *devinfo)
- int mdriver_init (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)
- int mdriver_find_mbxid (CANDEV_FLEXCAN_INFO *devinfo, uint32_t canmid)
- void mdriver_init_data (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)

20.5 src/hardware/can/public/hw/imx_can.h File Reference

```
#include <hw/libcan.h>
#include <hw/mini_driver.h>
#include <arm/imx/imx_flexcan.h>
```

20.6 src/hardware/deva/ctrl/mx/imx_sai_dll.c File Reference

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <time.h>
#include <sys/mman.h>
#include <hw/inout.h>
#include <sys/asoundlib.h>
#include <devctl.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "imx_sai_dll.h"
```

Functions

- int imx_sai_set_clock_rate (struct imx_card *imx, int rate, imx_txrx_index_t idx)
- int32_t imx_capabilities (struct imx_card *imx, ado_pcm_t *pcm, snd_pcm_channel_info_t *info)
- int32_t imx_playback_aquire (struct imx_card *imx, struct imx_stream **pc, ado_pcm_config_t *config, ado_pcm_subchn_t *subchn, uint32_t *why_failed)
- int32_t imx_playback_release (struct imx_card *imx, struct imx_stream *pc, ado_pcm_config_t *config)
- int32_t imx_capture_aquire (struct imx_card *imx, struct imx_stream **pc, ado_pcm_config_t *config, ado_pcm_subchn_t *subchn, uint32_t *why_failed)
- int32_t imx_capture_release (struct imx_card *imx, struct imx_stream *pc, ado_pcm_config_t *config)
- int32_t imx_prepare (struct imx_card *imx, struct imx_stream *pc, ado_pcm_config_t *config)
- int32_t imx_playback_trigger (struct imx_card *imx, struct imx_stream *pc, uint32_t cmd)
- int32_t imx_capture_trigger (struct imx_card *imx, struct imx_stream *pc, uint32_t cmd)
- void imx_play_pulse_hdlr (struct imx_card *imx, struct sigevent *event)
- void imx_cap_pulse_hdlr (struct imx_card *imx, struct sigevent *event)
- int imx_sai_init (struct imx_card *imx)
- void ctrl_version (int *major, int *minor, char *date)
- void imx_sai_config_default_protocol_flags (struct imx_card *imx)
- int imx_parse_commandline (struct imx_card *imx, char *args)
- int ctrl_init (struct imx_card **hw_context, ado_card_t *card, char *args)
- int ctrl_destroy (struct imx_card *imx)

20.6.1 Detailed Description

i.MX SAI audio driver source file.

20.7 src/hardware/deva/ctrl/mx/imx_sai_dll.h File Reference

```
#include <audio_driver.h>
#include <hw/dma.h>
#include <string.h>
#include <proto.h>
#include <sys/hwinfo.h>
#include <drvr/hwinfo.h>
#include "variant.h"
#include <arm/imx/imx_sai.h>
```

Data Structures

- struct imx_stream_dma
- struct imx_stream_pcm
- struct imx_stream
- struct imx_sai_xfer_config
- struct imx_sai_data
- struct imx_card

Typedefs

- typedef enum imx_stream_status imx_stream_status_t
- typedef struct imx_stream_dma imx_stream_dma_t
- typedef struct imx_stream_pcm imx_stream_pcm_t
- typedef enum imx_sai_mode imx_sai_mode_t
- typedef enum imx_sai_protocol imx_sai_protocol_t
- typedef enum imx_sai_sync_mode imx_sai_sync_mode_t
- typedef struct imx_sai_xfer_config imx_sai_xfer_config_t
- typedef struct imx_sai_data imx_sai_data_t

Enumerations

20.7.1 Detailed Description

i.MX SAI audio driver header file.

20.8 src/hardware/deva/ctrl/mx/nto/arm/dll.le.v7.sai_wm8960/variant.h File Reference

Functions

- int codec_set_rate (HW_CONTEXT_T *mx)
- int codec_mixer (ado_card_t *card, HW_CONTEXT_T *mx)

20.8.1 Detailed Description

Driver configuration header file.

20.9 src/hardware/flash/boards/qspi-imx/arm/le.v7/variant.h File Reference

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <errno.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/neutrino.h>
#include <hw/inout.h>
#include <hw/spi-master.h>
#include <arm/imx/imx_qspi.h>
```

Macros

- #define CLOCK_RATE 240000000

  Peripheral input clock - see startup for details.
- #define SIO3_MUX_PHYS_ADDR 0x30330040

  SW_MUX_CTL_PAD_EPDC_DATA03.
- #define SIO3_PAD_PHYS_ADDR 0x303302B0

  SW_PAD_CTL_PAD_EPDC_DATA03.
- #define PAGE_SIZE 256

  Page size in Bytes.
- #define SECTOR_SIZE 256

  Number of pages in one sector.
- #define TOTAL_SIZE 1024

  Number of sectors.
- #define DIE_NUMBER 1

  Number of die in connected memory/memories.

- #define ADDR_MODE_4BYTE 32

  32 bit address
- #define SCLK_FREQ (CLOCK_RATE / 4)

  Peripheral input clock / QSPI in-build divider.
- #define DEVICE_ID 0x20

  Memory type.
- #define MAN_ID 0xC2

  Manufacturer identification.
- #define PWR2_SIZE_UNIT 16

  64kB erase sector size (fs unit size)

20.10 src/hardware/deva/ctrl/mx/nto/arm/dll.le.v7.sai_wm8960/wm8960 File Reference

```
#include <audio_driver.h>
#include <string.h>
#include <hw/i2c.h>
#include <stdint.h>
#include "imx_sai_dll.h"
#include "wm8960.h"
```

Functions

- int codec_set_rate (HW_CONTEXT_T *mx)
- int codec_mixer (ado_card_t *card, HW_CONTEXT_T *mx)

20.10.1 Detailed Description

WM8960 driver source file.

20.11 src/hardware/deva/ctrl/mx/nto/arm/dll.le.v7.sai_wm8960/wm8960 File Reference

```
#include "variant.h"
```

Data Structures

- struct wm8960_context

Macros

- #define WM8960_AVDD 3300
- #define WM8960_ANALOG_BIAS_THRESH 3000
- #define WM8960_SLAVE_ADDR (0x34 >> 1)

Typedefs

- typedef struct wm8960_context wm8960_context_t

Functions

- int codec_set_rate (HW_CONTEXT_T *mx)
- int codec_mixer (ado_card_t *card, HW_CONTEXT_T *mx)

20.11.1 Detailed Description

WM8960 driver header file.

20.12 src/hardware/deva/ctrl/mx/proto.h File Reference

Functions

- int my_detach_pulse (void **x)

20.12.1 Detailed Description

i.MX SAI audio driver header file.

20.13 src/hardware/i2c/imx/proto.h File Reference

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/neutrino.h>
#include <sys/mman.h>
#include <hw/inout.h>
#include <hw/i2c.h>
#include <sys/hwinfo.h>
#include <drvr/hwinfo.h>
#include <sys/slog.h>
#include <sys/slogcodes.h>
#include <arm/imx/imx_i2c.h>
```

Data Structures

- struct _imx_dev

Typedefs

- typedef struct _imx_dev imx_dev_t

Functions

- void * imx_init (int argc, char *argv[ ])
- void imx_fini (void *hdl)
- int imx_options (imx_dev_t *dev, int argc, char *argv[ ])
- int imx_wait_bus_not_busy (imx_dev_t *dev)
- int imx_set_slave_addr (void *hdl, unsigned int addr, i2c_addrfmt_t fmt)
- int imx_set_bus_speed (void *hdl, unsigned int speed, unsigned int *ospeed)
- int imx_version_info (i2c_libversion_t *version)
- int imx_driver_info (void *hdl, i2c_driver_info_t *info)
- i2c_status_t imx_recv (void *hdl, void *buf, unsigned int len, unsigned int stop)
- i2c_status_t imx_send (void *hdl, void *buf, unsigned int len, unsigned int stop)
- uint32_t imx_wait_status (imx_dev_t *dev)
- i2c_status_t imx_sendaddr7 (imx_dev_t *dev, unsigned addr, int read, int restart)
- i2c_status_t imx_sendaddr10 (imx_dev_t *dev, unsigned addr, int read, int restart)
- i2c_status_t imx_sendbyte (imx_dev_t *dev, uint8_t byte)
- i2c_status_t imx_recvbyte (imx_dev_t *dev, uint8_t *byte, int nack, int stop)
- void imx_i2c_reset (imx_dev_t *dev)

20.13.1 Detailed Description

i.MX I2C driver header file.

20.14 src/hardware/can/imx/proto.h File Reference

```
#include "canimx.h"
```

Functions

- int mdriver_init (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)
- void mdriver_init_data (CANDEV_FLEXCAN_INFO *devinfo, CANDEV_FLEXCAN_INIT *devinit)

20.15 src/hardware/devc/serial/imx/proto.h File Reference

Functions

- void ser_stty (DEV_MX1 *dev)
- unsigned options (int argc, char *argv[ ])

20.16 src/hardware/deva/ctrl/mx/pulse.c File Reference

```
#include "imx_sai_dll.h"
```

Data Structures

- struct my_pulse_struct

Functions

- int my_attach_pulse (void **x, struct sigevent *event, void(*handler)(struct imx_card *hw_context, struct sigevent *event), struct imx_card *hw_context)
- int my_detach_pulse (void **x)

20.16.1 Detailed Description

i.MX SAI audio driver source file.

20.17 src/hardware/devc/serial/imx/pulse.c File Reference

```
#include "externs.h"
```

Data Structures

- struct my_pulse_struct

Functions

- int my_attach_pulse (void **x, struct sigevent *event, void(*handler)(DEV_MX1 *dev, struct sigevent *event), DEV_MX1 *dev)
- int my_detach_pulse (void **x)

20.18 src/hardware/devb/sdmmc/arm/imx.le.v7/bs.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/mman.h>
#include <hw/inout.h>
#include <unistd.h>
#include <arm/imx/imx_gpio.h>
#include <arm/imx/imx_usdhc.h>
#include <imx_hc.h>
#include <bs.h>
```

Functions

- int my_getsubopt (char **optionp, char *const *tokens, char **valuep)
- int bs_event (sdio_hc_t *hc, sdio_event_t *ev)

20.18.1 Detailed Description

Board specific interface

20.19 src/hardware/devb/sdmmc/arm/imx.le.v7/bs.h File Reference

```
#include <sys/utsname.h>
```

Data Structures

- struct _imx_ext

Typedefs

- typedef struct _imx_ext imx_ext_t

20.19.1 Detailed Description

Board specific interface

20.20 src/hardware/devb/sdmmc/sdiodi/hc/imx_hc.c File Reference

```
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <hw/inout.h>
#include <sys/mman.h>
#include <internal.h>
#include <sys/syspage.h>
#include <inttypes.h>
#include <arm/imx/imx_usdhc.h>
#include <imx_hc.h>
```

Functions

- int imx_sdhcx_dinit (sdio_hc_t *hc)
- int imx_sdhcx_init (sdio_hc_t *hc)

20.20.1 Detailed Description

Host controller interface

20.21 src/hardware/devb/sdmmc/sdiodi/hc/imx_hc.h File Reference

```
#include <internal.h>
```

Data Structures

- struct _imx_sdhcx_adma32_t
- struct _imx_usdhcx_hc

Typedefs

- typedef struct _imx_sdhcx_adma32_t imx_sdhcx_adma32_t
- typedef struct _imx_usdhcx_hc imx_sdhcx_hc_t

Functions

- int imx_sdhcx_init (sdio_hc_t *hc)
- int imx_sdhcx_dinit (sdio_hc_t *hc)

20.21.1 Detailed Description

Host controller interface

20.22 src/hardware/devc/serial/imx/externs.c File Reference

```
#include "externs.h"
```

20.23 src/hardware/devc/serial/imx/externs.h File Reference

```
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <malloc.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/neutrino.h>
#include <termios.h>
#include <devctl.h>
#include <sys/dcmd_chr.h>
#include <sys/iomsg.h>
#include <atomic.h>
#include <hw/inout.h>
#include <arm/imx/imx_uart.h>
#include <sys/io-char.h>
#include <sys/hwinfo.h>
#include <drvr/hwinfo.h>
#include <pthread.h>
#include <sys/rsrcdbmgr.h>
#include <sys/dispatch.h>
#include <sys/slog.h>
#include <sys/slogcodes.h>
#include "proto.h"
```

20.24 src/hardware/devc/serial/imx/intr.c File Reference

```
#include "externs.h"
```

20.25 src/hardware/devc/serial/imx/tedit.c File Reference

```
#include "externs.h"
```

Functions

- int edit (TTYDEV *dev, unsigned c)

20.26 src/hardware/devc/serial/imx/tto.c File Reference

```
#include "externs.h"
```

Functions

- int tto (TTYDEV *ttydev, int action, int arg1)
- void ser_stty (DEV_MX1 *dev)
- int drain_check (TTYDEV *ttydev, uintptr_t *count)

20.27 src/hardware/devnp/imx/bsd_media.c File Reference

```
#include <fec_imx.h>
#include <device_qnx.h>
```

Functions

- void bsd_mii_mediastatus (struct ifnet *ifp, struct ifmediareq *ifmr)
- int bsd_mii_mediachange (struct ifnet *ifp)
- void bsd_mii_initmedia (imx_fec_dev_t *imx_fec)

20.28 src/hardware/devnp/imx/detect.c File Reference

```
#include "fec_imx.h"
#include <device_qnx.h>
```

Functions

- void dump_mbuf (struct mbuf *m, uint32_t length)
- void * imx_rx_thread (void *arg)
- void imx_rx_thread_quiesce (void *arg, int die)
- void DumpMAC (imx_fec_dev_t *imx_fec)
- void DumpPhy (imx_fec_dev_t *imx_fec)
- void imx_speeduplex (imx_fec_dev_t *imx_fec)
- int imx_detect (void *dll_hdl, struct _iopkt_self *iopkt, char *options)

20.29 src/hardware/devnp/imx/devctl.c File Reference

```
#include "fec_imx.h"
#include <net/ifdrvcom.h>
#include <sys/sockio.h>
#include <avb.h>
```

Functions

- int imx_ioctl (struct ifnet *ifp, unsigned long cmd, caddr_t data)

20.30 src/hardware/devnp/imx/event.c File Reference

```
#include "fec_imx.h"
```

Functions

- int imx_process_queue (void *arg, struct nw_work_thread *wtp)
- int imx_enable_queue (void *arg)
- const struct sigevent * imx_isr (void *arg, int iid)
- int imx_enable_interrupt (void *arg)
- int imx_process_interrupt (void *arg, struct nw_work_thread *wtp)

20.31 src/hardware/devnp/imx/fec_imx.c File Reference

```
#include "fec_imx.h"
```

20.32 src/hardware/devnp/imx/fec_imx.h File Reference

```
#include <io-pkt/iopkt_driver.h>
```

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/syslog.h>
#include <sys/slogcodes.h>
#include <sys/cache.h>
#include <sys/param.h>
#include <sys/syspage.h>
#include <sys/malloc.h>
#include <sys/mman.h>
#include <sys/io-pkt.h>
#include <sys/callout.h>
#include <sys/mbuf.h>
#include <sys/ioctl.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>
#include <netdrvr/mdi.h>
#include <netdrvr/eth.h>
#include <netdrvr/nicsupport.h>
#include <netdrvr/common.h>
#include <netdrvr/ptp.h>
#include <hw/nicinfo.h>
#include <net/if.h>
#include <net/if_dl.h>
#include <net/if_ether.h>
#include <net/if_types.h>
#include <quiesce.h>
#include <siglock.h>
#include <nw_thread.h>
#include <sys/device.h>
#include <net/if_media.h>
#include <dev/mii/miivar.h>
#include <sys/hwinfo.h>
#include <drvr/hwinfo.h>
```

Functions

- const struct sigevent * imx_isr (void *arg, int iid)
- int imx_enable_interrupt (void *arg)
- int imx_process_interrupt (void *arg, struct nw_work_thread *wtp)
- int imx_enable_queue (void *arg)
- int imx_process_queue (void *arg, struct nw_work_thread *wtp)
- void imx_set_multicast (imx_fec_dev_t *)
- void dump_mbuf (struct mbuf *m, uint32_t length)
- int imx_detect (void *dll_hdl, struct _iopkt_self *iopkt, char *options)
- void imx_speeduplex (imx_fec_dev_t *imx_fec)
- int imx_ioctl (struct ifnet *ifp, unsigned long cmd, caddr_t data)
- void imx_start (struct ifnet *)
- void imx_transmit_complete (imx_fec_dev_t *, uint8_t)
- int imx_set_tx_bw (imx_fec_dev_t *imx_fec, struct ifdrv *ifd)
- int imx_output (struct ifnet *, struct mbuf *, struct sockaddr *, struct rtentry *)
- int imx_receive (imx_fec_dev_t *, struct nw_work_thread *, uint8_t)
- void imx_MDI_MonitorPhy (void *)

- void imx_init_phy (imx_fec_dev_t *)
- int imx_get_phy_addr (imx_fec_dev_t *)
- int imx_setup_phy (imx_fec_dev_t *imx_fec)
- void imx_sabreauto_rework (imx_fec_dev_t *imx_fec)
- uint16_t imx_mii_read (void *handle, uint8_t phy_add, uint8_t reg_add)
- void imx_mii_write (void *handle, uint8_t phy_add, uint8_t reg_add, uint16_t data)
- void imx_mii_callback (void *handle, uchar_t phy, uchar_t newstate)
- void imx_bcm54220_phy_init (imx_fec_dev_t *imx_fec)
- void imx_update_stats (imx_fec_dev_t *)
- void imx_clear_stats (imx_fec_dev_t *)
- void bsd_mii_initmedia (imx_fec_dev_t *imx_fec)
- int imx_ptp_start (imx_fec_dev_t *)
- void imx_ptp_stop (imx_fec_dev_t *)
- int imx_ptp_is_eventmsg (struct mbuf *, ptpv2hdr_t **)
- void imx_ptp_add_rx_timestamp (imx_fec_dev_t *, ptpv2hdr_t *, mpc_bd_t *)
- void imx_ptp_add_tx_timestamp (imx_fec_dev_t *, ptpv2hdr_t *, mpc_bd_t *)
- int imx_ptp_get_rx_timestamp (imx_fec_dev_t *, ptp_extts_t *)
- int imx_ptp_get_tx_timestamp (imx_fec_dev_t *, ptp_extts_t *)
- int imx_ptp_ioctl (imx_fec_dev_t *, struct ifdrv *)
- void imx_ptp_get_cnt (imx_fec_dev_t *, ptp_time_t *)
- void imx_ptp_set_cnt (imx_fec_dev_t *, ptp_time_t)
- void imx_ptp_set_compensation (imx_fec_dev_t *, ptp_comp_t)

20.33 src/hardware/devnp/imx/mii.c File Reference

```
#include "fec_imx.h"
```

Functions

- void DumpMAC (imx_fec_dev_t *imx_fec)
- void DumpPhy (imx_fec_dev_t *imx_fec)
- uint16_t imx_mii_read (void *handle, uint8_t phy_add, uint8_t reg_add)
- void imx_mii_write (void *handle, uint8_t phy_add, uint8_t reg_add, uint16_t data)
- void imx_mii_callback (void *handle, uchar_t phy, uchar_t newstate)
- void imx_MDI_MonitorPhy (void *)
- int imx_get_phy_addr (imx_fec_dev_t *)
- int imx_setup_phy (imx_fec_dev_t *imx_fec)
- void imx_init_phy (imx_fec_dev_t *)
- void imx_bcm54220_phy_init (imx_fec_dev_t *imx_fec)
- void imx_sabreauto_rework (imx_fec_dev_t *imx_fec)

20.34 src/hardware/devnp/imx/multicast.c File Reference

```
#include "fec_imx.h"
```

Functions

- void imx_set_multicast (imx_fec_dev_t *)

20.35 src/hardware/devnp/imx/ptp.c File Reference

```
#include "fec_imx.h"
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/in.h>
#include <sys/proc.h>
```

Functions

- void imx_ptp_cal (imx_fec_dev_t *imx_fec)
- int imx_ptp_start (imx_fec_dev_t *)
- void imx_ptp_stop (imx_fec_dev_t *)
- int imx_ptp_is_eventmsg (struct mbuf *, ptpv2hdr_t **)
- void imx_ptp_add_rx_timestamp (imx_fec_dev_t *, ptpv2hdr_t *, mpc_bd_t *)
- void imx_ptp_add_tx_timestamp (imx_fec_dev_t *, ptpv2hdr_t *, mpc_bd_t *)
- int imx_ptp_get_rx_timestamp (imx_fec_dev_t *, ptp_extts_t *)
- int imx_ptp_get_tx_timestamp (imx_fec_dev_t *, ptp_extts_t *)
- void imx_ptp_get_cnt (imx_fec_dev_t *, ptp_time_t *)
- void imx_ptp_set_cnt (imx_fec_dev_t *, ptp_time_t)
- void imx_ptp_set_compensation (imx_fec_dev_t *, ptp_comp_t)
- int imx_ptp_ioctl (imx_fec_dev_t *, struct ifdrv *)

20.36 src/hardware/devnp/imx/receive.c File Reference

```
#include "bpfilter.h"
#include "fec_imx.h"
#include <net/if_vlanvar.h>
#include <netinet/in.h>
#include <avb.h>
```

Functions

- int imx_receive (imx_fec_dev_t *, struct nw_work_thread *, uint8_t)

20.37 src/hardware/devnp/imx/stats.c File Reference

```
#include "fec_imx.h"
```

Functions

- void imx_update_stats (imx_fec_dev_t *)
- void imx_clear_stats (imx_fec_dev_t *)

20.38 src/hardware/devnp/imx/transmit.c File Reference

```
#include "bpfilter.h"
#include "fec_imx.h"
#include <device_qnx.h>
#include <avb.h>
```

Functions

- int imx_tx (imx_fec_dev_t *imx_fec, struct mbuf *m, uint8_t queue)
- void imx_start (struct ifnet *)
- void imx_transmit_complete (imx_fec_dev_t *, uint8_t)
- int imx_output (struct ifnet *, struct mbuf *, struct sockaddr *, struct rtentry *)
- int imx_set_tx_bw (imx_fec_dev_t *imx_fec, struct ifdrv *ifd)

20.39 src/hardware/etfs/nand4096/imx-micron/apbh_dma.c File Reference

```
#include <arm/inout.h>
#include <sys/mman.h>
#include <errno.h>
#include <sys/slogcodes.h>
#include <unistd.h>
#include <arm/imx/imx_bch.h>
#include "chipio.h"
#include "apbh_dma.h"
```

Functions

- int apbh_intr_wait (chipio *chipio)
- void * apbhint_thread (void *arg)
- void apbh_init (chipio *chipio)
- void apbh_init_dma_channel (chipio *chipio)

20.40 src/hardware/etfs/nand4096/imx-micron/apbh_dma.h File Reference

```
#include <stdbool.h>
#include <sys/neutrino.h>
#include <pthread.h>
#include <arm/imx/imx_apbh.h>
#include <arm/imx/imx_gpmi.h>
```

Functions

- int apbh_intr_wait (chipio *chipio)
- void * apbhint_thread (void *arg)
- void apbh_init (chipio *chipio)
- void apbh_init_dma_channel (chipio *chipio)

20.41 src/hardware/etfs/nand4096/imx-micron/bch_ecc.c File Reference

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/slog.h>
#include <sys/mman.h>
#include <sys/neutrino.h>
#include <hw/inout.h>
#include <fs/etfs.h>
#include <string.h>
#include <assert.h>
#include <pthread.h>
#include <arm/imx/imx_bch.h>
#include <arm/imx/imx_gpmi.h>
#include "bch_ecc.h"
#include "chipio.h"
#include "devio.h"
```

Functions

- int bch_intr_wait (chipio *chipio)
- void * bchint_thread (void *arg)
- void bch_init (chipio *chipio)
- void bch_set_layout (chipio *chipio)
- void bch_set_erase_threshold (chipio *chipio, uint8_t threshold)
- uint32_t bch_get_ecc_status (chipio *chipio)

20.42 src/hardware/etfs/nand4096/imx-micron/bch_ecc.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
```

Macros

- #define BCH_SUBBLOCK_SIZE 1024

  BCH sub block size in bytes.
- #define BCH_ECC_SIZE 42

  BCH ECC size in bytes.

Functions

- void * bchint_thread (void *arg)
- int bch_intr_wait (chipio *chipio)
- void bch_init (chipio *chipio)
- void bch_set_layout (chipio *chipio)
- uint32_t bch_get_ecc_status (chipio *chipio)
- void bch_set_erase_threshold (chipio *chipio, uint8_t threshold)

20.43 src/hardware/etfs/nand4096/imx-micron/chipio.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <fs/etfs.h>
#include <arm/inout.h>
#include <string.h>
#include <sys/neutrino.h>
#include <unistd.h>
#include <errno.h>
#include <arm/imx/imx_apbh.h>
#include <arm/imx/imx_gpmi.h>
#include <arm/imx/imx_bch.h>
#include "chipio.h"
#include "devio.h"
#include "dma_descriptor.h"
#include "bch_ecc.h"
#include "apbh_dma.h"
#include "gpmi_pio.h"
```

Functions

- void create_blockErase_descriptor (dma_blk_erase_t *superStruct, unsigned page)
- void create_readEcc_descriptor (dma_readEcc_device_t *superStruct, unsigned page, void *ret_data)
- void create_readRaw_descriptor (dma_readRaw_device_t *superStruct, uint32_t data_size, unsigned page, void *ret_data)
- void create_writeEcc_descriptor (dma_programEcc_t *superStruct, unsigned page, void *write_data, uint32_t data_size)
- void create_writeRaw_descriptor (dma_programRaw_t *superStruct, unsigned page, void *write_data, uint32_t data_size)
- void create_readStatus2_descriptor (dma_read_status_t *superStruct, uint8_t *memory_status, unsigned page)
- void create_readStatus_descriptor (dma_read_status_t *superStruct, uint8_t *memory_status)
- void create_readId_descriptor (dma_read_id_device_t *superStruct, uint8_t *ret_raw_id)
- void create_reset_descriptor (dma_reset_device_t *superStruct)
- int nand_init (chipio *cio)
- int run_dma (apbh_dma_t *dma, chipio *chipio, uint8_t wait_flag, uint32_t phy_addr)
- bool is_dma_active (chipio *chipio)
- int nand_wait_busy (chipio *cio, uint32_t time_out, uint8_t chip_select)
- void device_to_nfc (uint8_t *parsed_data, uint8_t *raw_data)
- void nfc_to_device (uint8_t *device_data, uint8_t *nfc_data, uint8_t ecc_size)

20.44 src/hardware/etfs/nand4096/imx-micron/chipio.h File Reference

```
#include <stdbool.h>
#include <stdio.h>
#include <sys/mman.h>
#include <arm/imx/imx_apbh.h>
#include <arm/imx/imx_gpmi.h>
```

Data Structures

- struct _apbh_dma_t
- struct _apbh_dma_gpmi1_t
- struct _apbh_dma_gpmi3_t
- struct _apbh_dma_gpmi5_t
- struct _dma_blk_erase_t
- struct _dma_programEcc_t
- struct _dma_programRaw_t
- struct _dma_readEcc_device_t
- struct _dma_readRaw_device_t
- struct _NAND_dma_read_status_device_t
- struct _dma_reset_device_t
- struct _dma_read_id_device_t
- struct _chipio_t

Macros

Internal NFC error codes

- #define **NAND_EOK** 0x55AA
- #define **NAND_EIO** 0x3
- #define **NAND_DMA_EIO** 0x33

Internal DMA flags - intended for run_dma method

- #define **NAND_DMA_GPMI_TRANS** 0x1
- #define **NAND_DMA_BCH_TRANS** 0x2

Address size description

- #define **NAND_COLUMN_ADDRESS_CYCLES** 2
- #define **NAND_ROW_ADDRESS_CYCLES** 3
- #define **NAND_ADDRESS_CYCLES** (NAND_COLUMN_ADDRESS_CYCLES + NAND_ROW_ADDR↩ ESS_CYCLES)

Row and Column manipulation macros

- #define **NAND_ADDR_COL1**(addr) ((addr) & 0xff)
- #define **NAND_ADDR_COL2**(addr) (((addr) & 0x1f00) >> 8)
- #define **NAND_ADDR_ROW1**(page) ((page) & 0xff)
- #define **NAND_ADDR_ROW2**(page) (((page) & 0xff00) >> 8)
- #define **NAND_ADDR_ROW3**(page) (((page) & 0x70000) >> 16)

Reset command description

- #define **NANDCMD_RESET** 0xFF
- #define **NANDCMD_RESET_SIZE** 1

Status command description

- #define **NANDCMD_READ_STATUS_SIZE** 1
- #define **NANDCMD_READ_STATUS** 0x70
- #define **NANDCMD_READ_STATUS_ENHANCED** 0x78

Read Id command description - read device followed by one address cycle to specify read ID type

- #define **NANDCMD_READ_ID** 0x90
- #define **NANDCMD_READ_ID_TYPE** 0x00
- #define NANDCMD_READ_ID_SIZE 2

  Number of commands sent for a NAND Device Read ID.
- #define NANDCMD_READ_ID_RESULT_SIZE 5

  Size in bytes of a Read ID command result.

Block erase command description

- #define **NANDCMD_BLOCK_ERASE** 0x60
- #define **NANDCMD_BLOCK_ERASE_CONFIRM** 0xD0

Read command description

- #define **NANDCMD_READ** 0x00
- #define **NANDCMD_READ_CONFIRM** 0x30

Program command description

- #define **NANDCMD_PAGE_PROGRAM** 0x80
- #define **NANDCMD_PAGE_PROGRAM_CONFIRM** 0x10
- #define **NANDCMD_PAGE_CACHE_PROG_CFRM** 0x15

Typedefs

- typedef struct _apbh_dma_t apbh_dma_t
- typedef struct _apbh_dma_gpmi1_t apbh_dma_gpmi1_t
- typedef struct _apbh_dma_gpmi3_t apbh_dma_gpmi3_t
- typedef struct _apbh_dma_gpmi5_t apbh_dma_gpmi5_t
- typedef struct _dma_blk_erase_t dma_blk_erase_t
- typedef struct _dma_programEcc_t dma_programEcc_t
- typedef struct _dma_programRaw_t dma_programRaw_t
- typedef struct _dma_readEcc_device_t dma_readEcc_device_t
- typedef struct _dma_readRaw_device_t dma_readRaw_device_t
- typedef struct _NAND_dma_read_status_device_t dma_read_status_t
- typedef struct _dma_reset_device_t dma_reset_device_t
- typedef struct _dma_read_id_device_t dma_read_id_device_t
- typedef struct _chipio_t chipio

Functions

- void create_blockErase_descriptor (dma_blk_erase_t *superStruct, unsigned page)
- void create_writeRaw_descriptor (dma_programRaw_t *superStruct, unsigned page, void *write_data, uint32_t data_size)
- void create_writeEcc_descriptor (dma_programEcc_t *superStruct, unsigned page, void *write_data, uint32_t data_size)
- void create_readEcc_descriptor (dma_readEcc_device_t *superStruct, unsigned page, void *ret_data)
- void create_readRaw_descriptor (dma_readRaw_device_t *superStruct, uint32_t data_size, unsigned page, void *ret_data)
- void create_readStatus_descriptor (dma_read_status_t *superStruct, uint8_t *memory_status)
- void create_readStatus2_descriptor (dma_read_status_t *superStruct, uint8_t *memory_status, unsigned page)
- void create_reset_descriptor (dma_reset_device_t *superStruct)
- void create_readId_descriptor (dma_read_id_device_t *superStruct, uint8_t *ret_raw_id)
- void device_to_nfc (uint8_t *parsed_data, uint8_t *raw_data)
- void nfc_to_device (uint8_t *device_data, uint8_t *nfc_data, uint8_t ecc_size)
- int run_dma (apbh_dma_t *dma, chipio *chipio, uint8_t wait_flag, uint32_t phy_addr)
- bool is_dma_active (chipio *chipio)
- int nand_init (chipio *cio)
- int nand_wait_busy (chipio *cio, uint32_t time_out, uint8_t chip_select)

20.45 src/hardware/etfs/nand4096/imx-micron/devio.c File Reference

```
#include <arm/inout.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <gulliver.h>
#include <sys/slog.h>
#include <sys/neutrino.h>
#include <fs/etfs.h>
#include <sys/mman.h>
#include <arm/imx/imx_bch.h>
#include "devio.h"
#include "apbh_dma.h"
#include "bch_ecc.h"
```

Functions

- int devio_options (struct etfs_devio ∗dev, char ∗optstr)
- int devio_init (struct etfs_devio ∗dev)
- int devio_readcluster (struct etfs_devio ∗dev, unsigned cluster, uint8_t ∗buf, struct etfs_trans ∗trp)
- int devio_readtrans (struct etfs_devio ∗dev, unsigned cluster, struct etfs_trans ∗trp)
- int devio_postcluster (struct etfs_devio ∗dev, unsigned cluster, uint8_t ∗buf, struct etfs_trans ∗trp)
- int devio_eraseblk (struct etfs_devio ∗dev, unsigned blk)
- int devio_sync (struct etfs_devio ∗dev)

20.46 src/hardware/etfs/nand4096/imx-micron/devio.h File Reference

```
#include <stdint.h>
#include "chipio.h"
```

Data Structures

- struct _spare_t

Macros

- #define DATASIZE 4096

  Page data size.
- #define SPARESIZE 224

  Page spare size.
- #define PAGESIZE (DATASIZE + SPARESIZE)

  Page size.
- #define ETFS_META_SIZE_PER_SUBBLOCK 8

Typedefs

- typedef struct _spare_t spare_t

20.46.1 Macro Definition Documentation

20.46.1.1 #define ETFS_META_SIZE_PER_SUBBLOCK 8

Meta information size per sub-block in bytes

Definition at line 54 of file devio.h.

20.46.2 Typedef Documentation

20.46.2.1 typedef struct _spare_t spare_t

NAND spare area for BCH engine

20.47 src/hardware/etfs/nand4096/imx-micron/dma_descriptor.h File Reference

```
#include <arm/imx/imx_gpmi.h>
#include <arm/imx/imx_apbh.h>
```

Macros

- #define NAND_DMA_WAIT4RDY_CMD

  APBH DMA Macro for Wait4Ready command.
- #define NAND_DMA_WAIT4RDY_PIO(u32ChipSelect)

  GPMI PIO DMA Macro for Wait4Ready command.
- #define NAND_DMA_TXDATA_CMD(TransferSize, Semaphore, CommandWords, Wait4End, Cmd)

  APBH DMA Macro for Transmit Data command. Transfer TransferSize bytes with DMA. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Lock the NAND while waiting for this DMA chain to complete. Decrement semaphore if this is the last part of the chain. Another descriptor follows this one in the chain. This DMA is a read from System Memory - write to device.
- #define NAND_DMA_TXDATA_PIO(u32ChipSelect, TransferSize)

  GPMI PIO DMA Macro for Transmit Data command. Setup transfer as a write. Transfer NumBitsInWord bits per DMA cycle. Lock CS during this transaction. Select the appropriate chip. Address lines need to specify Data transfer (0b00) Transfer TransferSize - NumBitsInWord values.
- #define NAND_DMA_SENSE_CMD(SenseSemaphore)

  APBH DMA Macro for Sense command. Transfer no Bytes with DMA. Transfer no Words to PIO. Don't lock the NAND while waiting for Ready to go high. Decrement semaphore if this is the last part of the chain. Another descriptor follows this one in the chain.
- #define NAND_DMA_RX_CMD_ECC(TransferSize, Semaphore)

  APBH DMA Macro for Read Data command with ECC. Receive TransferSize bytes with DMA. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Decrement semaphore if this is the last part of the chain. Unlock the NAND after this DMA chain completes. Another descriptor follows this one in the chain. No DMA transfer here; the ECC8 block becomes the bus master and performs the memory writes itself instead of the DMA.
- #define NAND_DMA_RX_NO_ECC_CMD(TransferSize, Semaphore)

  APBH DMA Macro for Receive Data with no ECC command. Receive TransferSize bytes with DMA but no E↩ CC. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Decrement semaphore if this is the last part of the chain. Unlock the NAND after this DMA chain completes. Another descriptor follows this one in the chain. This DMA is a write to System Memory - read from device.
- #define NAND_DMA_RX_PIO(u32ChipSelect, TransferSize)

  GPMI PIO DMA Macro for Receive command. Setup transfer as a READ. Transfer NumBitsInWord bits per D↩ MA cycle. Select the appropriate chip. Address lines need to specify Data transfer (0b00) Transfer TransferSize - NumBitsInWord values.
- #define NAND_DMA_COMMAND_CMD(TransferSize, Semaphore, NandLock, CmdWords)

APBH DMA Macro for sending NAND Command sequence. Transmit TransferSize bytes to DMA. Transfer one Word to PIO. Wait for DMA to complete before starting next DMA descriptor in chain. Decrement semaphore if this is the last part of the chain. Lock the NAND until the next chain. Another descriptor follows this one in the chain. This DMA is a read from System Memory - write to device.

- #define NAND_DMA_COMMAND_PIO(u32ChipSelect, TransferSize, AssertCS)

  GPMI PIO DMA Macro when sending a command. Setup transfer as a WRITE. Transfer NumBitsInWord bits per DMA cycle. Lock CS during and after this transaction. Select the appropriate chip. Address lines need to specify Command transfer (0b01) Increment the Address lines if AddrIncr is set. Transfer TransferSize - NumBitsInWord values.
- #define NAND_DMA_ECC_PIO(EnableDisable) (IMX_GPMI_ECCCTRL_ENABLE_ECC(EnableDisable))

  GPMI PIO DMA Macro for disabling ECC during this write.
- #define NAND_DMA_ECC_CTRL_PIO(EccBufferMask, decode_encode_size)

  GPMI PIO DMA Macro sequence for ECC decode. Setup READ transfer ECC Control register. Setup for ECC Decode, 4 Bit. Enable the ECC block The ECC Buffer Mask determines which fields are corrected.

20.48 src/hardware/etfs/nand4096/imx-micron/gpmi_pio.c File Reference

```
#include <stdint.h>
#include <arm/inout.h>
#include <unistd.h>
#include <arm/imx/imx_gpmi.h>
#include "chipio.h"
```

Functions

- void gpmi_soft_reset (chipio *chipio)
- void gpmi_set_busy_timeout (chipio *chipio, uint16_t busy_timeout)

20.49 src/hardware/etfs/nand4096/imx-micron/gpmi_pio.h File Reference

Functions

- void gpmi_soft_reset (chipio *chipio)
- void gpmi_set_busy_timeout (chipio *chipio, uint16_t busy_timeout)

20.50 src/hardware/flash/boards/qspi-imx/f3s_qspi.h File Reference

```
#include <sys/f3s_mtd.h>
#include "qspi_cmds.h"
```

Functions

- int32_t f3s_qspi_open (f3s_socket_t *socket, uint32_t flags)
- uint8_t * f3s_qspi_page (f3s_socket_t *socket, uint32_t page, uint32_t offset, int32_t *size)
- int32_t f3s_qspi_read (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t text_offset, int32_t buffer_size, uint8_t *buffer)
- int32_t f3s_qspi_status (f3s_socket_t *socket, uint32_t flags)
- void f3s_qspi_close (f3s_socket_t *socket, uint32_t flags)
- void f3s_qspi_reset (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)
- int32_t f3s_qspi_ident (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)
- int32_t f3s_qspi_write (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset, int32_t size, uint8_t *buffer)
- int f3s_qspi_erase (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)
- int32_t f3s_qspi_sync (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t text_offset)

20.51 src/hardware/flash/boards/qspi-imx/f3s_qspi_close.c File Reference

```
#include "f3s_qspi.h"
#include "imx_fc_qspi.h"
```

Functions

- void f3s_qspi_close (f3s_socket_t *socket, uint32_t flags)

20.52 src/hardware/flash/boards/qspi-imx/f3s_qspi_erase.c File Reference

```
#include "f3s_qspi.h"
```

Functions

- int f3s_qspi_erase (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)

20.53 src/hardware/flash/boards/qspi-imx/f3s_qspi_ident.c File Reference

```
#include <sys/slogcodes.h>
#include "f3s_qspi.h"
#include "imx_fc_qspi.h"
#include "qspi_cmds.h"
```

Functions

- int32_t f3s_qspi_ident (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)

20.54 src/hardware/flash/boards/qspi-imx/f3s_qspi_main.c File Reference

```
#include "f3s_qspi.h"
```

Functions

- int main (int argc, char **argv)

20.55 src/hardware/flash/boards/qspi-imx/f3s_qspi_open.c File Reference

```
#include "f3s_qspi.h"
#include "imx_fc_qspi.h"
```

Functions

- int32_t f3s_qspi_open (f3s_socket_t *socket, uint32_t flags)

20.56 src/hardware/flash/boards/qspi-imx/f3s_qspi_page.c File Reference

```
#include "f3s_qspi.h"
```

Functions

- uint8_t * f3s_qspi_page (f3s_socket_t *socket, uint32_t page, uint32_t offset, int32_t *size)

20.57 src/hardware/flash/boards/qspi-imx/f3s_qspi_read.c File Reference

```
#include "f3s_qspi.h"
#include <sys/slog.h>
```

Functions

- int32_t f3s_qspi_read (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t text_offset, int32_t buffer_size, uint8_t *buffer)

20.58 src/hardware/flash/boards/qspi-imx/f3s_qspi_reset.c File Reference

```
#include "f3s_qspi.h"
```

Functions

- void f3s_qspi_reset (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset)

20.59 src/hardware/flash/boards/qspi-imx/f3s_qspi_status.c File Reference

```
#include "f3s_qspi.h"
```

Functions

- int32_t f3s_qspi_status (f3s_socket_t *socket, uint32_t flags)

20.60 src/hardware/flash/boards/qspi-imx/f3s_qspi_sync.c File Reference

```
#include "f3s_qspi.h"
#include "imx_fc_qspi.h"
```

Functions

- int32_t f3s_qspi_sync (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t text_offset)

20.61 src/hardware/flash/boards/qspi-imx/f3s_qspi_write.c File Reference

```
#include "f3s_qspi.h"
#include <sys/slog.h>
```

Functions

- int32_t f3s_qspi_write (f3s_dbase_t *dbase, f3s_access_t *access, uint32_t flags, uint32_t offset, int32_t size, uint8_t *buffer)

20.62 src/hardware/flash/boards/qspi-imx/imx_fc_qspi.c File Reference

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/slog.h>
#include <sys/slogcodes.h>
#include <sys/mman.h>
#include <pthread.h>
#include <string.h>
#include <arm/imx/imx_qspi.h>
#include "variant.h"
#include "imx_fc_qspi.h"
#include "qspi_cmds.h"
```

Functions

- int imx_qspi_setcfg (int fd)
- void * int_thread (void *arg)
- int imx_qspi_open (void)
- int imx_qspi_close (int fd)
- int imx_qspi_lock_lut (int fd)
- int imx_qspi_unlock_lut (int fd)
- imx_qspi_lutx_t imx_qspi_create_lut_record (uint8_t instr0, uint8_t pad0, uint8_t opr0, uint8_t instr1, uint8_t pad1, uint8_t opr1)
- int imx_qspi_write_lut (int fd, uint8_t index, imx_qspi_lutx_t *lutcmd0, imx_qspi_lutx_t *lutcmd1, imx_qspi_lutx_t *lutcmd2, imx_qspi_lutx_t *lutcmd3)
- int qspi_intr_wait (imx_qspi_t *dev)
- int imx_qspi_send_ip_nowait_cmd (int fd, uint8_t lut_index, uint32_t data_size_override)
- int imx_qspi_send_ip_cmd (int fd, uint8_t lut_index, uint32_t data_size_override)
- int imx_qspi_clear_fifo (const int qspi_fd, uint32_t mask)
- int imx_qspi_write_data (int fd, uint8_t *addr, uint32_t data_size)
- int imx_qspi_read_data (int fd, uint8_t *buffer, uint32_t size)

20.63 src/hardware/flash/boards/qspi-imx/imx_fc_qspi.h File Reference

```
#include <stdint.h>
```

Data Structures

- union imx_qspi_lutx_t
- struct _imx_qspi_t

Macros

- #define IMX_QSPI_AMBA_BASE_ADDRESS 0x60000000U

FIFO parameters

- #define IMX_QSPI_FIFO_WIDTH 4
  FIFO width: 4 bytes.
- #define IMX_QSPI_FIFO_DEPTH 32
  FIFO depth: 32 entries.

LUT commands

- #define IMX_QSPI_PAD_1 0x00
  Communication on 1 pad only.
- #define IMX_QSPI_PAD_4 0x02
  Communication on 4 pads.
- #define IMX_QSPI_INSTR_CMD 1
  FC command.
- #define IMX_QSPI_INSTR_ADDR 2
  FC address.
- #define IMX_QSPI_INSTR_DUMMY 3
  FC dummy operation.
- #define IMX_QSPI_INSTR_READ 7
  FC read.
- #define IMX_QSPI_INSTR_WRITE 8
  FC write.
- #define IMX_QSPI_INSTR_JMP_ON_CS 9
  FC jump on cs.
- #define IMX_QSPI_INSTR_ADDR_DDR 10
  FC ddr address.
- #define IMX_QSPI_INSTR_READ_DDR 14
  FC ddr read.
- #define IMX_QSPI_INSTR_STOP 0
  FC stop.
- #define IMX_QSPI_LUT_KEY 0x5AF05AF0U
  LUT key.
- #define IMX_QSPI_LUT_LOCK 0x1
  LUT contect lock.
- #define IMX_QSPI_LUT_UNLOCK 0x2
  LUT contect unlock.

LUT device command indexes

- #define IMX_QSPI_LUT_CMD_IDX_READ 0
  Normal read 24-bit address read.
- #define IMX_QSPI_LUT_CMD_IDX_RDIR 1
  Read identification.
- #define IMX_QSPI_LUT_CMD_IDX_WREN 2
  Write enable.
- #define IMX_QSPI_LUT_CMD_IDX_SE 3
  Sector erase.
- #define IMX_QSPI_LUT_CMD_IDX_RDSR 4
  Read status register.
- #define IMX_QSPI_LUT_CMD_IDX_RDSCUR 5
  Read security register.
- #define IMX_QSPI_LUT_CMD_IDX_4READ4B 6
  4 read 4B
- #define IMX_QSPI_LUT_CMD_IDX_4PP4B 7
  4 write 4B
- #define IMX_QSPI_LUT_CMD_IDX_WRSR 8
  Write status register (Normal mode)
- #define IMX_QSPI_LUT_CMD_IDX_EN4B 9
  Enter 4-byte addressing mode.
- #define IMX_QSPI_LUT_CMD_IDX_RDCR 10
  Read configuration register.

Typedefs

- typedef struct _imx_qspi_t imx_qspi_t

Functions

- int imx_qspi_setcfg (int fd)
- int imx_qspi_open (void)
- int imx_qspi_close (int fd)
- int imx_qspi_write_data (int fd, uint8_t *addr, uint32_t data_size)
- int imx_qspi_read_data (int fd, uint8_t *buffer, uint32_t size)
- int imx_qspi_clear_fifo (const int qspi_fd, uint32_t mask)
- int imx_qspi_send_ip_nowait_cmd (int fd, uint8_t lut_index, uint32_t data_size_override)
- int imx_qspi_send_ip_cmd (int fd, uint8_t lut_index, uint32_t data_size_override)
- int imx_qspi_unlock_lut (int fd)
- int imx_qspi_lock_lut (int fd)
- imx_qspi_lutx_t imx_qspi_create_lut_record (uint8_t instr0, uint8_t pad0, uint8_t opr0, uint8_t instr1, uint8_t pad1, uint8_t opr1)
- int imx_qspi_write_lut (int fd, uint8_t index, imx_qspi_lutx_t *lutcmd0, imx_qspi_lutx_t *lutcmd1, imx_qspi_lutx_t *lutcmd2, imx_qspi_lutx_t *lutcmd3)

20.63.1 Typedef Documentation

20.63.1.1 typedef struct _imx_qspi_t imx_qspi_t

Low level driver handle

20.64 src/hardware/flash/boards/qspi-imx/qspi_cmds.c File Reference

```
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <time.h>
#include "qspi_cmds.h"
#include "imx_fc_qspi.h"
#include <arm/imx/imx_qspi.h>
```

Functions

- int iswriting (const int qspi_fd)
- int read_cfg (const int qspi_fd, uint8_t *cfg_reg)
- int read_erase_status (const int qspi_fd)
- int read_status (const int qspi_fd, uint8_t *stat_reg)
- int write_status (const int qspi_fd, uint8_t *stat_reg)
- int read_ident (const int qspi_fd, int *manufact_id, int *device_id, uint32_t *size)
- int pd_release (const int qspi_fd)
- int sector_erase (const int qspi_fd, const int offset)
- int page_program (const int qspi_fd, int offset, int len, uint8_t *data)
- int read_from (const int qspi_fd, int offset, int len, uint8_t *buffer)

20.65 src/hardware/flash/boards/qspi-imx/qspi_cmds.h File Reference

```
#include <stdint.h>
#include "variant.h"
```

Macros

Device status register definition - MX25L51245G

- #define DEVICE_SR_WIP (1 << 0)
  Write in Progress (WIP) bit.
- #define DEVICE_SR_WEL (1 << 1)
  Write Enable Latch (WEL) bit.
- #define DEVICE_SR_BP0 (1 << 2)
  Block Protect bit BP0.
- #define DEVICE_SR_BP1 (1 << 3)
  Block Protect bit BP1.
- #define DEVICE_SR_BP2 (1 << 4)
  Block Protect bit BP2.
- #define DEVICE_SR_BP3 (1 << 5)

Block Protect bit BP3.

- #define DEVICE_SR_QE (1 << 6)
  4 mode enable bit
- #define DEVICE_SR_SRWD (1 << 7)
  Status Register Write Disable (SRWD) bit.

Flash commands

- #define FLASH_OPCODE_SE4B 0xDC
  (SPI) Erase 64 KB block (Sector erase) 4B
- #define FLASH_OPCODE_READ4B 0x13
  (SPI) Read data bytes 4B
- #define FLASH_OPCODE_4READ4B 0xEC
  (QSPI) Read data bytes 4B
- #define FLASH_OPCODE_PP4B 0x12
  (SPI) Page program 4B
- #define FLASH_OPCODE_4PP4B 0x3E
  (QSPI) Page program 4B
- #define FLASH_OPCODE_RDID 0x9F
  (SPI) Read JEDEC ID in normal mode
- #define FLASH_OPCODE_WREN 0x06
  (SPI) Write enable
- #define FLASH_OPCODE_WRSR 0x01
  (SPI) Write status register
- #define FLASH_OPCODE_RDSR 0x05
  (SPI) Read status register
- #define FLASH_OPCODE_EN4B 0xB7
  (SPI) Enable 4B
- #define FLASH_OPCODE_RDSCUR 0x2B
  (SPI) Read security register

Functions

- int iswriting (const int qspi_fd)
- int write_status (const int qspi_fd, uint8_t *stat_reg)
- int read_cfg (const int qspi_fd, uint8_t *cfg_reg)
- int read_status (const int qspi_fd, uint8_t *stat_reg)
- int read_ident (const int qspi_fd, int *manufact_id, int *device_id, uint32_t *size)
- int pd_release (const int qspi_fd)
- int sector_erase (const int qspi_fd, const int offset)
- int page_program (const int qspi_fd, int offset, int len, uint8_t *data)
- int read_from (const int qspi_fd, int offset, int len, uint8_t *buffer)

20.66 src/hardware/i2c/imx/bus_speed.c File Reference

```
#include "proto.h"
```

Functions

- unsigned int find_best_ic (unsigned int i2c_div)
- int imx_set_bus_speed (void *hdl, unsigned int speed, unsigned int *ospeed)

20.66.1 Detailed Description

i.MX I2C driver source file.

20.67 src/hardware/i2c/imx/common.c File Reference

```
#include "proto.h"
```

Functions

- void imx_i2c_reset (imx_dev_t *dev)
- i2c_status_t imx_recvbyte (imx_dev_t *dev, uint8_t *byte, int nack, int stop)
- i2c_status_t imx_sendbyte (imx_dev_t *dev, uint8_t byte)
- i2c_status_t imx_sendaddr7 (imx_dev_t *dev, unsigned addr, int read, int restart)
- i2c_status_t imx_sendaddr10 (imx_dev_t *dev, unsigned addr, int read, int restart)

20.67.1 Detailed Description

i.MX I2C driver source file.

20.68 src/hardware/i2c/imx/fini.c File Reference

```
#include "proto.h"
```

Functions

- void imx_fini (void *hdl)

20.68.1 Detailed Description

i.MX I2C driver source file.

20.69 src/hardware/i2c/imx/info.c File Reference

```
#include "proto.h"
```

Functions

- int imx_driver_info (void *hdl, i2c_driver_info_t *info)

20.69.1 Detailed Description

i.MX I2C driver source file.

20.70 src/hardware/i2c/imx/init.c File Reference

```
#include "proto.h"
```

Functions

- void * imx_init (int argc, char *argv[ ])

20.70.1 Detailed Description

i.MX I2C driver source file.

20.71 src/hardware/devc/serial/imx/init.c File Reference

```
#include "externs.h"
#include <sys/mman.h>
#include <string.h>
```

20.72 src/lib/dma/sdma/init.c File Reference

```
#include "sdma.h"
```

Functions

- void ctor (void)
- void dtor (void)

20.73 src/hardware/i2c/imx/lib.c File Reference

```
#include "proto.h"
```

Functions

- int i2c_master_getfuncs (i2c_master_funcs_t *funcs, int tabsize)

20.73.1 Detailed Description

i.MX I2C driver source file.

20.74 src/hardware/i2c/imx/options.c File Reference

```
#include "proto.h"
```

Functions

- int query_hwi_device (imx_dev_t *dev, unsigned unit)
- int imx_options (imx_dev_t *dev, int argc, char *argv[ ])

20.74.1 Detailed Description

i.MX I2C driver source file.

20.75 src/hardware/devc/serial/imx/options.c File Reference

```
#include "externs.h"
```

Functions

- int query_hwi_device (TTYINIT_MX1 *dip, unsigned unit)
- unsigned options (int argc, char *argv[ ])

20.76 src/hardware/i2c/imx/recv.c File Reference

```
#include "proto.h"
```

Functions

- i2c_status_t imx_recv (void *hdl, void *buf, unsigned int len, unsigned int stop)

20.76.1 Detailed Description

i.MX I2C driver source file.

20.77 src/hardware/i2c/imx/send.c File Reference

```
#include "proto.h"
```

Functions

- i2c_status_t imx_send (void *hdl, void *buf, unsigned int len, unsigned int stop)

20.77.1 Detailed Description

i.MX I2C driver source file.

20.78 src/hardware/i2c/imx/slave_addr.c File Reference

```
#include "proto.h"
```

Functions

- int imx_set_slave_addr (void *hdl, unsigned int addr, i2c_addrfmt_t fmt)

20.78.1 Detailed Description

i.MX I2C driver source file.

20.79 src/hardware/i2c/imx/version.c File Reference

```
#include "proto.h"
```

Functions

- int imx_version_info (i2c_libversion_t *version)

20.79.1 Detailed Description

i.MX I2C driver source file.

20.80 src/hardware/i2c/imx/wait.c File Reference

```
#include "proto.h"
```

Functions

- int imx_wait_bus_not_busy (imx_dev_t *dev)
- uint32_t imx_wait_status (imx_dev_t *dev)

20.80.1 Detailed Description

i.MX I2C driver source file.

20.81 src/hardware/support/wdtkick/main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/resmgr.h>
#include <sys/neutrino.h>
#include <hw/inout.h>
#include <sys/slog.h>
#include <sys/slogcodes.h>
#include <errno.h>
#include <arm/imx/imx_wdog.h>
#include <sys/procmgr.h>
#include <drvr/hwinfo.h>
```

Functions

- int main (int argc, char *argv[ ])

20.82 src/hardware/devc/serial/imx/main.c File Reference

```
#include "externs.h"
```

20.83 src/lib/dma/sdma/api.c File Reference

```
#include "sdma.h"
```

Functions

- sdma_chan_t * chan_create (unsigned ch_num)
- void chan_destroy (sdma_chan_t *chan_ptr)
- void register_init (void)
- int parse_init_options (const char *options)
- int parse_channel_options (sdma_chan_t *chan_ptr, const char *options)
- void callback_reenable_descr (unsigned ch_num)
- int sdma_init (const char *options)
- void sdma_fini (void)
- void sdma_query_channel (void *handle, dma_channel_query_t *chinfo)
- int sdma_driver_info (dma_driver_info_t *info)
- int sdma_channel_info (unsigned channel, dma_channel_info_t *info)
- void * sdma_channel_attach (const char *optstring, const struct sigevent *event, unsigned *channel, int prio, unsigned flags)
- void sdma_channel_release (void *handle)
- int sdma_setup_xfer (void *handle, const dma_transfer_t *tinfo)
- int sdma_xfer_start (void *handle)
- int sdma_xfer_abort (void *handle)
- unsigned sdma_bytes_left (void *handle)
- int sdma_xfer_complete (void *handle)
- int get_dmafuncs (dma_functions_t *functable, int tabsize)

20.84 src/lib/dma/sdma/cmd.c File Reference

```
#include "sdma.h"
```

Functions

- int sdmaram_script_load ()
- int sdmacmd_cmdch_create (void)
- void sdmacmd_cmdch_destroy (void)
- void sdmacmd_ctx_config (sdma_chan_t *chan_ptr)
- int sdmacmd_ctx_load (sdma_chan_t *chan_ptr)

20.85 src/lib/dma/sdma/imx/microcode.h File Reference

```
#include <stdint.h>
```

20.86 src/lib/dma/sdma/imx/script.c File Reference

```
#include "sdma.h"
#include "microcode.h"
```

Functions

- int sdmascript_lookup (sdma_scriptinfo_t *scriptinfo)

20.87 src/lib/dma/sdma/irq.c File Reference

```
#include "sdma.h"
```

Functions

- const struct sigevent * irq_handler (void *area, int id)
- int sdmairq_init (uint32_t irq)
- void sdmairq_fini (void)
- void sdmairq_event_add (uint32_t channel, const struct sigevent *event)
- void sdmairq_event_remove (uint32_t channel)
- void sdmairq_callback_add (uint32_t channel, sdmairq_callback_t func_ptr)
- void sdmairq_callback_remove (uint32_t channel)

20.88 src/lib/dma/sdma/sdma.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <inttypes.h>
#include <pthread.h>
#include <sys/siginfo.h>
#include <sys/mman.h>
#include <hw/inout.h>
#include <sys/neutrino.h>
#include <errno.h>
#include <atomic.h>
#include <fcntl.h>
#include <sys/rsrcdbmgr.h>
#include <sys/rsrcdbmsg.h>
#include <hw/dma.h>
#include <sys/hwinfo.h>
#include <drvr/hwinfo.h>
#include <sys/cache.h>
```

Data Structures

- struct sdma_bd_cmd_and_status_s
- struct sdma_bd_t
- struct sdma_ccb_t
- struct sdma_ch_ctx_t
- struct microcode_info_t
- struct sdma_scriptinfo_t
- struct sdma_shmem_t

Typedefs

- typedef struct sdma_bd_cmd_and_status_s sdma_bd_cmd_and_status_t

Functions

- int sdmasync_init (void)
- void sdmasync_fini (void)
- pthread_mutex_t * sdmasync_cmdmutex_get (void)
- pthread_mutex_t * sdmasync_libinit_mutex_get (void)
- pthread_mutex_t * sdmasync_regmutex_get (void)
- int sdmasync_is_first_process (void)
- int sdmasync_is_last_process (void)
- void sdmasync_process_cnt_incr (void)
- void sdmasync_process_cnt_decr (void)
- off64_t sdmasync_ccb_paddr_get (void)
- sdma_ccb_t * sdmasync_ccb_ptr_get (void)
- int sdmacmd_cmdch_create (void)

- void sdmacmd_cmdch_destroy (void)
- void sdmacmd_ctx_config (sdma_chan_t *chan_ptr)
- int sdmacmd_ctx_load (sdma_chan_t *chan_ptr)
- int sdmairq_init (uint32_t irq)
- void sdmairq_fini (void)
- void sdmairq_event_add (uint32_t channel, const struct sigevent *event)
- void sdmairq_event_remove (uint32_t channel)
- void sdmairq_callback_add (uint32_t channel, sdmairq_callback_t func_ptr)
- void sdmairq_callback_remove (uint32_t channel)
- int sdmascript_lookup (sdma_scriptinfo_t *scriptinfo)

20.89 src/lib/dma/sdma/sync.c File Reference

```
#include "sdma.h"
```

Functions

- int sdmasync_init (void)
- void sdmasync_fini (void)
- pthread_mutex_t * sdmasync_cmdmutex_get (void)
- pthread_mutex_t * sdmasync_libinit_mutex_get (void)
- pthread_mutex_t * sdmasync_regmutex_get (void)
- int sdmasync_is_first_process (void)
- int sdmasync_is_last_process (void)
- void sdmasync_process_cnt_incr (void)
- void sdmasync_process_cnt_decr (void)
- off64_t sdmasync_ccb_paddr_get (void)
- sdma_ccb_t * sdmasync_ccb_ptr_get (void)

20.90 src/utils/r/rtc/nto/arm/clk_mx7srtc.c File Reference

```
#include "rtc.h"
#include <time.h>
#include <arm/mx7x.h>
```

Functions

- int init_mx7srtc (struct chip_loc *chip, char *argv[ ])
- int get_mx7srtc (struct tm *tm, int cent_reg)
- int set_mx7srtc (struct tm *tm, int cent_reg)

20.90.1 Detailed Description

RTC driver source file.

20.91 src/utils/r/rtc/nto/clk_net.c File Reference

```
#include "rtc.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/procfs.h>
#include <sys/netmgr.h>
#include <fcntl.h>
```

Functions

- int init_net (struct chip_loc *chip, char *argv[ ])
- int get_net (struct tm *tm, int cent_reg)
- int set_net (struct tm *tm, int cent_reg)

20.91.1 Detailed Description

RTC driver source file.

20.92 src/utils/r/rtc/nto/support.c File Reference

```
#include <stddef.h>
#include <inttypes.h>
#include <dlfcn.h>
#include <sys/syspage.h>
#include <hw/sysinfo.h>
#include <sys/hwinfo.h>
#include "rtc.h"
```

Functions

- char * query_clock_hw (struct chip_loc *chip)
- int load_external_clock (const char *given_name, struct rtc_desc *clk)
- int close_external_clock (void)

20.92.1 Detailed Description

RTC driver source file.

20.93 src/utils/r/rtc/qnxrtc.c File Reference

```
#include "rtc.h"
```

Functions

- unsigned chip_read (unsigned off, unsigned size)
- void chip_write (unsigned off, unsigned val, unsigned size)
- int main (int argc, char *argv[ ])

20.93.1 Detailed Description

RTC driver source file.

20.94 src/utils/r/rtc/rtc.h File Reference

```
#include <inttypes.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/timers.h>
#include <sys/osinfo.h>
#include <i86.h>
#include <conio.h>
```

Functions

- int load_external_clock (const char *given_name, struct rtc_desc *clk)
- int close_external_clock (void)
- char * query_clock_hw (struct chip_loc *)
- unsigned chip_read (unsigned off, unsigned size)
- void chip_write (unsigned off, unsigned val, unsigned size)
- int init_net (struct chip_loc *chip, char *argv[ ])
- int get_net (struct tm *tm, int cent_reg)
- int set_net (struct tm *tm, int cent_reg)

20.94.1 Detailed Description

RTC driver header file.

Index