SYNOPSYS(R)

DesignWare Cores
Ethernet MAC Universal

Device Driver Reference Manual 1.3

DWC Ether MAC 10/100/1000 Universal
DWC Ether MAC 10/100 Universal

Generated by Doxygen 1.5.3

Wed September 23 10:12:53 2009

Date	Software Release	Description
September 2009	1.3	Driver Compatible to GMAC Univ Release 3.60a. Driver modified to support AV testing
February 2009	1.2	Driver Compatible to GMAC Univ Release 3.50a. Driver modified to support Enh Desc with 8 words and IEEE 1588 Kernel API support
May 2008	1.1	Driver compatible to GMAC Univ Release 3.41a. Driver modified to support Enhanced Descriptor structures.
November 2007	1.0	Driver compatible to GMAC Univ Release 3.30a.

Contents

Chapter 1

The Synopsys GMAC IP Device Driver

1.1 Intended Audience

Device Driver developers for GMAC-UNIV core.

1.2 Purpose of this Chapter

This document describes the Synopsys provided GMAC-UNIV device driver software. This device driver software is developed and tested on Fedora distribution 2.6.11-1.1369_FC4 on Intel(R) Pentium(R) 4 CPU 2.80GHz system. Driver API's are developed for GMAC-AHB configuration where GMAC with AHB-interfaced to DMA(GMAC-AHB). Though the driver developed and tested on Intel Pentium platform running Linux 2.6.xx, the OS and platform dependent functionality is abstracted to a greater extent so the software can be ported on to different platform running different OS with minimal modifications. The architecture of the device driver is explained in Driver architecture section of this document.

1.3 Device Driver Architecture

As mentioned earlier Device Driver software is architected to make the software easily portable on to different operating sytems running on different platforms.

Device Driver porting on to different architecture and Different operating systems are handled in "Device Driver Porting Guide" section.

All the platform dependent functionalities are abstracted in to the following files. Functions for platform dependent memory allocations, platform dependent delay calls are wrapped in to Synopsys defined functions in synopGMAC_plat.c. All standard data types are typedefined to more generic data types in the corresponding header file. Basic hardware access functions are written as static inline functions in the same header file. Whenever the driver software is ported on to different platform these two files should be properly modified as demanded by the platform/OS.

```
synopGMAC_plat.h
synopGMAC_plat.c
```

Synopsys developed and tested driver uses 32 bit 33Mhz pci as the peripheral bus interface for control and data path. All the bus dependent functionalities are abstracted in to the files. Major function of this bus interface unit is to get memory allocation for GMAC CSR space, descriptor memory and the network

buffer memory. The GMAC device structure synopGMAC_device is populated with the base addresses for MAC, DMA and PHY which are obtained from the pci bus subsystem. PCI sybsystem provides the interrupt number for the device. This interface should be handled in the bus interface dependent code when the driver is ported on different platform.

```
synopGMAC_pci_bus_interface.h
synopGMAC_pci_bus_interface.c
```

The device specific register map and the data structures are defined in the header file

```
synopGMAC_Dev.h
```

platform and OS independent driver api's are abstracted in the file

```
synopGMAC_Dev.c
```

The api's provided is more generic and the user need not worry about the underlined memory map for the register space and the descriptor. Even the bit masks of the registers and descriptors are transparent to the user, when these api's are used. To facilitate greater flexibility, generic register access interface is also provided. These functions are defined in synopGMAC_plat.h. Since the descriptor memory allocation may vary (in terms of memory chunk available, alignment requirement) depending on the platform, the code for descriptor memory allocation and the network buffer allocation is kept outside the device independent part of the software. These functionalities are available in network dependent layer. Next section explains this in detail.

1.4 Linux Dependent Layer

The sample driver is developed using the platform and OS independent layer of the driver code mentioned in earlier sections. This sample code is developed and tested on Intel Pentium 4 platform running Linux 2.6.11. The code for this is abstracted in the following files.

```
synopGMAC_network_interface.h
synopGMAC_network_interface.c
```

These two files should be treated as example code while porting the driver code on to a different platform. Since the DMA-able memory is available through the pci device structure, the driver's memory allocation functions should be carefully modified/altered while porting.

This section briefly explains the Linux networking interface and the device driver's registration to these interface.

Networking stack in Linux sees the GMAC device as a network interface/device. This is represented by net_device structure. The important members of this device interface are given below.

```
struct net_device
      {
              .
              .
      int (* open) (struct net_device *dev);
      int (* stop) (struct net_device *dev);
      int (* hard_start_xmit) (struct sk_buff *skb, struct net_device *dev);
      struct net_device_stats* (* get_stats) (struct net_dvice *dev);
              .
              .

      void *priv;
      }
```

Device Driver registers it's functions to the Linux as given below.

```
netdev->open            = &synopGMAC_linux_open;
netdev->stop            = &synopGMAC_linux_close;
netdev->hard_start_xmit = &synopGMAC_linux_xmit_frames;
netdev->get_stats       = &synopGMAC_linux_get_stats;
```

1.5 Initialization sequence of GMAC

This section gives a brief flow chart for the GMAC driver initialization. The steps provided here serve as guide lines for the GMAC initialization.

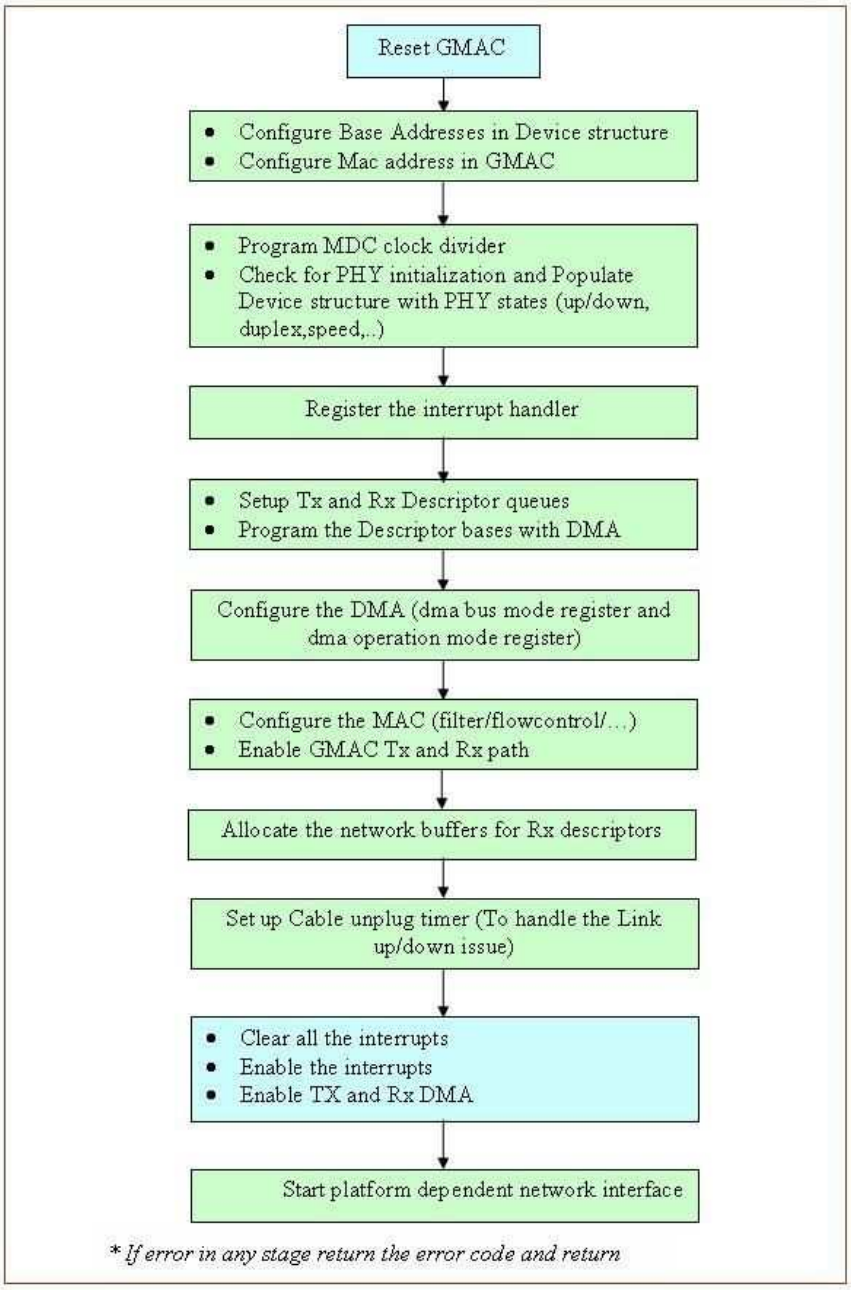


Figure 1.1: Initialization Sequence

1.6 Programming Sequence For Tx Path

The Transmit path starts when the OS has got a buffer ready to be transmitted. Kernel invokes/calls the function registered at netdev->hard_start_xmit. Action carried out in driver registerd function (synopGMAC_linux_xmit_frames), is provided in the form of a flow chart. Transmission is handled both in the normal execution context, which is nothing but inside a process context as well as partially (handing over the transmitted buffer memory to OS) in the interrupt context. please refer ISR sequence for the detailed description.

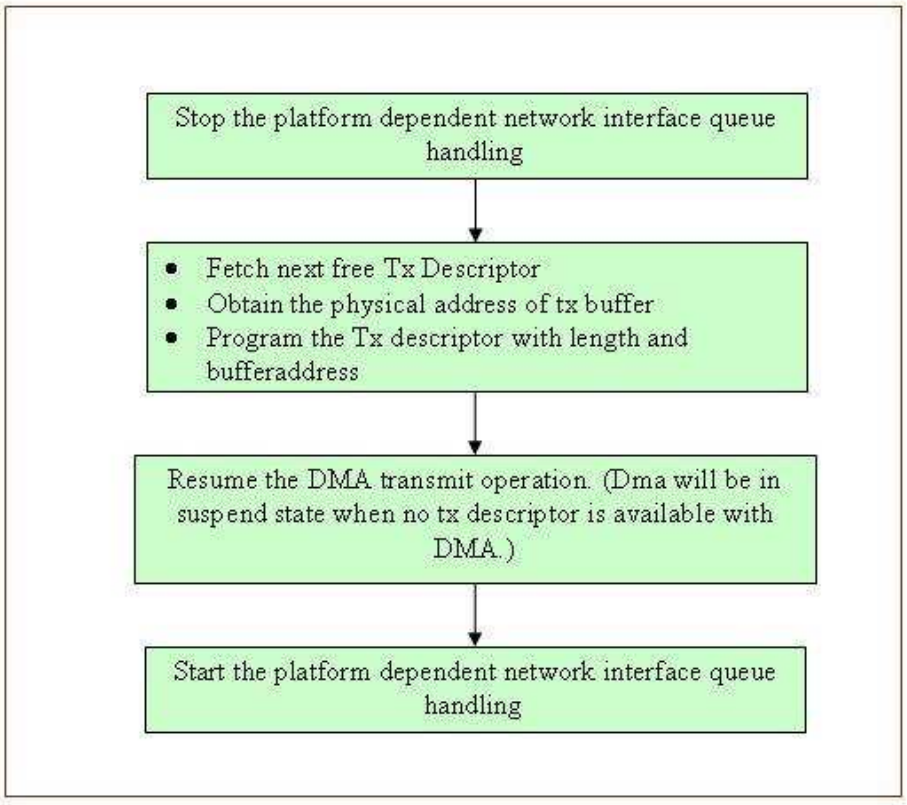


Figure 1.2: Tx Path Programming Sequence

1.7 Interrupt_service_routine Programming Sequence

In Linux the reception is through interrupts rather polling, no function registed with the kernel for handling received packets. The reception is completely handled in the interrupt service routine registered with the Kernel. Device generates transmission complete interrupt (generated after a packet is transmitted, refer transmit path for details), indicating completion of a packet transmission.

Note:

The flow chart only explains the sequence of operation assuming no errors occured in the process of transmission and reception. In case of errors, they should be handled properly to avoid system malfunctioning.

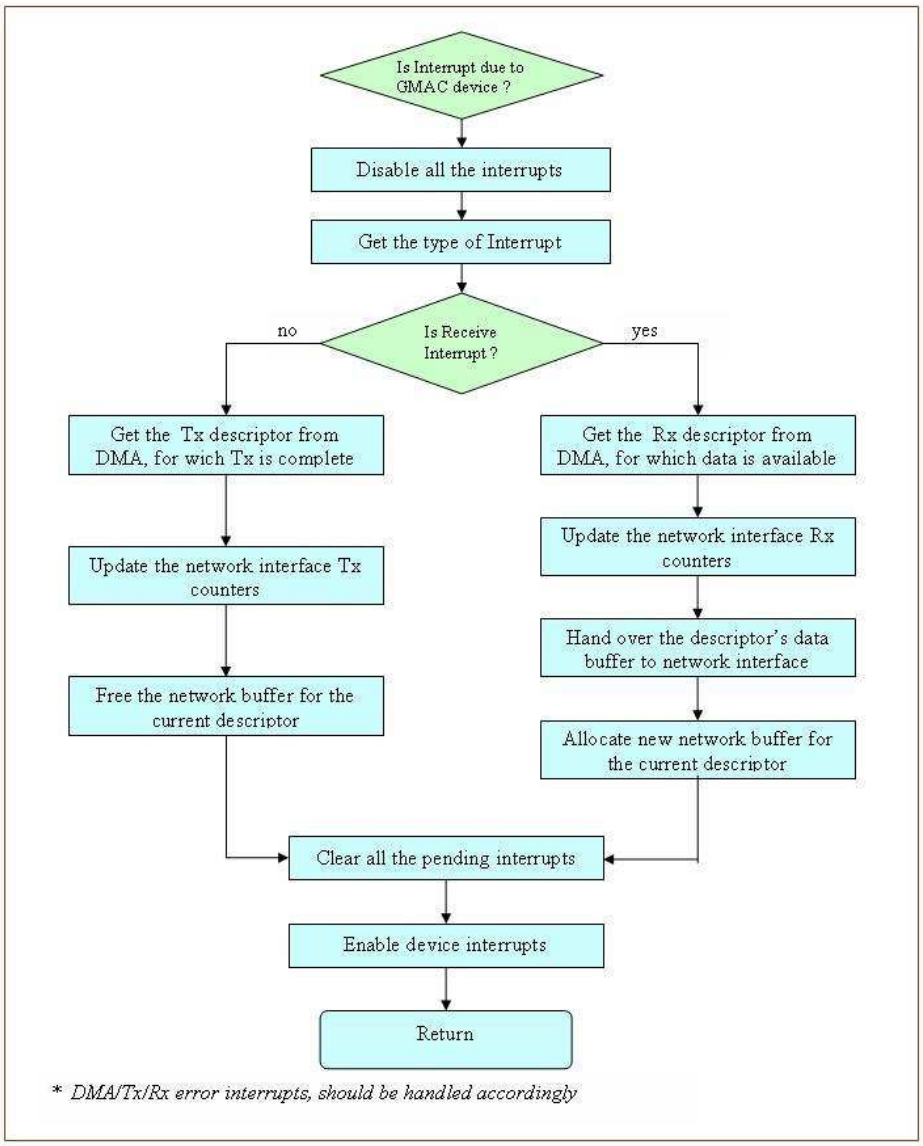


Figure 1.3: Programming Sequence For ISR

1.8 Debug Utility

A Debug utility synopGMAC_Debug.c is also provided along with the driver. This utility software should be compiled at user land. This software provides access to Device CSR space to facilitate access to Descriptors, status, and PMT and MMC (Refer to the corresponding sections for detailed descriptions) counters. The functionaly is through IOCTL calls. There are suitable entry points provided in the driver to carry out the IOCTL calls. When porting to a differnet Operating system, ioctl usage should be addressed with due care.

Note:

synopGMAC_Debug.c should be compiled with the define ENH_DESC_8W when the core and driver software are operating with 8 word descriptor format. If this procedure is compromised, it risks in system hanging as the utility tries to access the memory locations outside the allocated range.

1.9 Descriptors

GMAC supports Ring and Chain Structure for the descriptors. Driver supports either all descriptors are queued in Ring mode or in Chain mode. Only descriptor setup functions and descriptor freeing functions take one argument to indicate whether the descriptor structure to be created is either in RING or CHAIN. All the remaining Api's are intelligent enough to understand the descriptor structure (either ring or chain) and work accordingly. In the sample code provided the Both transmitter and Received descriptor queues contain 64 descriptors each. To change the number of descriptors change synopGMAC_Dev.h file for the follwoing accordingly.

```
#define TRANSMIT_DESC_SIZE  64 //256
#define RECEIVE_DESC_SIZE   64 //256
```

Note:

The interrupt service routine provided is not capable of handling the buffer2 pointing to data buffers in ring mode of operation. Make sure to modify the ISR to handle buffer2, when driver Api's are passed with non zero argument for buffer2.

1.10 Alternate (Enhanced) Descriptor structure

The default descritpor structure allows data buffers of up to 2,048 bytes. An Alternative descriptor structure has been implemented to support buffers of up to 8KB (Useful for Jumbo frames). This descriptor structure also consists of four 32 bit words, as in the default Receive and Transmit descripotrs. The main difference from the default is the re-assignment of the Control and Status Bits in TDES0, TDES1 and RDES1. Device driver can be compiled to work with Enhanced descriptor strucure by specifying the compilation switch "ENH_DESC".

The Alternate descriptor structure has been modified to support 8 words descriptor structure from GMAC release 3.50 onwards. By default GMAC core supports 4 word structure. If user wants to configure the core to operate with eight word descriptor structur, the Bus Mode Register's bit number seven should be set. This Descriptor is required for the core to support Full IPC Offloading and the IEEE 1588 time stamping. The device driver software should be compiled with switch "ENH_DESC" and "ENH_DESC_8W" to make use of Enhanced Descriptor with 8 words. If only "ENH_DESC" switch is used, the driver assumes that the Descriptor structure is of 4 words.

Note:

User of this driver should be aware of the GMAC core configuration and should use the driver cautiously.
The Enhanced Descriptor structure is not compatible with the default descriptor structure. If core is configured with Enhanced Descriptors, then driver should be compiled with "ENH_DESC" flag. If core is configured with default Descriptors, then driver should be compiled without "ENH_DESC" flag.

1.11 1588 Time Stamping

The driver provides Kernel API's for IEEE 1588 Time Stamping. They are here only for reference. User should make use of these API's with thorrough knowledge of the IEEE 1588 protocol. The driver doesnot differentiate between Version-1 and Version-2 time stamping. The time stamping software provided with this release assumes that hardware is configured to work with "Alternate (Enhanced) Descriptor" structure. All API's provided are not applicable to all the configuration. User should use API's as appropriate.

1.12 Power Down capability

GMAC supports the powerdown feature only if it is configured for PMT in corekit configurationn. GMAC can be put in powedown mode through an ioctl call IOCTL_POWER_DOWN. If req->unit is 1 the GMAC will be put in powerdown and if req->unit is 2 GMAC will come out of powerdown to normal mode of operation.

Testing of powerdown feature of GMAC:

The procedure given here is applicable only when the system (computer) with synopsys GMAC interfaced with other linux/unix system. This assumption is due to the fact that our setup makes use of wake up frames support provided in the Linux.

Make sure the System with Synopsys GMAC is up and running. Once the interface is up and running, the GMAC can be put in powerdown mode by using the following command

```
./synopGMAC_Debug ethx powerdown on
```

Note:

ethx is the network interface with Synopsys GMAC hardware. Change this to eth0/eth1/eth2/.... accordingly. GMAC will go in to powerdown mode. Packet transmission and reception ceases when GMAC enters power down mode.

GMAC can be brought back from power down mode to normal mode in two ways.

1 Using magic packets

Execute the following command from any other system (Make sure this "other system" is in the LAN along with the "system with Synopsys GMAC hardware" or connected directly through a cross-over cable)

```
ether-wake 00:55:7B:B5:7D:F7
```

Note:

00:55:7B:B5:7D:F7 is the default MAC address assigned to Synopsys GMAC by the driver. Refer compilation section to change default mac address.

When this command is executed a wake up packet is sent to the interface with the mac address mentioned. Now the Synopsys GMAC comes out of power down mode to normal operation.

2 Using Remote Wake-up frames

Execute the following command from any other system (Make sure this "other system" is in the LAN along with the "system with Synopsys GMAC hardware" or connected directly through a cross-over cable)

Note:

The remote wake-up frames given here work only with the default wake-up filter configuration provided by Synopsys driver. This makes use of synopGMAC_wakeup_filter_config3 configuration. Ac-

cording to this configuration, GMAC treats a frame with data containing eight consecutive 0x55 from offset 50 as the wake up frame.

```
ping 10.144.134.73 -p 55
```

With this command Synopsys GMAC comes out of power down mode to normal operation mode.

Note:

10.144.134.73 is the ip address of the Synopsys GMAC interface. Make sure to change this IP address accordingly.

1.13 IP/TCP checksum offloading

GMAC supports both IP and TCP/UDP/ICMP checksum offloading. This is avaialbe only if it is selected from the corekit configuration. Checksum offloading in Synopsys provided driver is controlled through compilation switch "IPC_OFFLOAD". To enable checksum offloading, the driver should be compiled with the IPC_OFFLOAD flag. Refer to the Makefile provided with the device driver code.

Note:

To carry out the checksum offloading, support in the networking is needed. In the sample driver, we make use of Linux networking capablility to offload the checksum computation in the hardware.

1.14 Mac Management Counters

Remote management counters support in hardware is availabe only if MMC module is selected in the corekit configuration. Synopsys provided driver provides Kernel APIs to control the counter management. The counters are provided to the use throgh an ioctl call IOCTL_READ_REGISTER. From user space one should issue the following to get the counters.

```
./synopGMAC_Debug ethx dump mmc
```

1.15 AV Support

The GMAC software release 3.60a onwards is delivered with helper functions to validated and test the AV feature supported by DW GMAC Ethernet Universal core.

The driver should be compiled with ENH_DESC, ENH_DESC_8W and AVB_SUPPORT compilation switches to enable the AV testing. If driver is compiled with only AVB_SUPPORT and without the other switches mentioned, functionality is undefined and the driver/system might hang. The accompanied software synopGMAC_Debug.c should also be compiled with AVB_SUPPORT. This can be done by enabling the macro in the source file itself.

The driver supports only RING_MODE of Descriptor structure for AV testing.

Note:

It is highly recommended to go through the Application note (Provided on request) to use the driver for AV testing.

1.16 Compilation

This section is applicable only for the device driver provided by Synopsys in its entiretly. The driver package is provided along with a makefile which allows the driver to be compiled as modules. Run make clean to clean the intermediate files.

```
make clean
```

Run make to generate the loadable kernel module synop_GMAC_ether.ko

```
make
```

Note:

Make sure no error/warning seen while compiling the driver. If any, fix the source code and then proceed.
To insert the driver module

```
insmod synop_GMAC_ether.ko
```

configure the ip as

```
ifconfig ethxx 10.144.134.73
```

To check the networking statistics

```
ifconfig ethxx
```

How to change the Mac address of the interface?

By default Synopsys GMAC interface is programmed with 00:55:7B:B5:7D:F7 as MAC address. This default address can be changed to user specific address by

```
ifconfig ethxx  hw ether 00:55:7B:B5:7D:F3
```

This changes the mac address from 00:55:7B:B5:7D:F5 to 00:55:7B:B5:7D:f3.

Note:

xx denotes the interface name as given by Linux. If the system contains Synpsys GMAC as the only network interface, then the interface is eth0. Ip address shown here is just an example.

The device should be up and running by this time. Ping from another system to make sure the proper functioning of hardware and driver software.

To remove the driver module

```
rmmod synop_GMAC_ether
```

The debug utility is compiled by runnig gcc as given below

```
gcc synopGMAC_Debug.c -o synopGMAC_Debug
```

1.17 Driver Limitations

Some of the limitations of the device driver software are listed here

- No support implemeted for jumbo frame handling. Driver modification needed to support jumbo frames. Modifications required even when Alternate descriptors are used.
- Driver works with the assumptions that the data received is at most 2048 bytes in case of Regular descriptors and 8192 bytes for Alternate descriptors. If larger frame received, driver behavior is unkown.

Chapter 2

Device Driver File Index

2.1 Device Driver File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Device Driver Page Index

3.1 Device Driver Related Pages

Here is a list of all related documentation pages:

Chapter 4

Device Driver File Documentation

4.1 synopGMAC_Dev.c File Reference

This file defines the synopsys GMAC device dependent functions.

```
#include "synopGMAC_Dev.h"
```

Functions

- s32 synopGMAC_set_mdc_clk_div (synopGMACdevice *gmacdev, u32 clk_div_val)
  Function to set the MDC clock for mdio transactiona.
- u32 synopGMAC_get_mdc_clk_div (synopGMACdevice *gmacdev)
  Returns the current MDC divider value programmed in the ip.
- s32 synopGMAC_read_phy_reg (u32 *RegBase, u32 PhyBase, u32 RegOffset, u16 *data)
  Function to read the Phy register.
- s32 synopGMAC_write_phy_reg (u32 *RegBase, u32 PhyBase, u32 RegOffset, u16 data)
  Function to write to the Phy register.
- s32 synopGMAC_phy_loopback (synopGMACdevice *gmacdev, bool loopback)
  Function to configure the phy in loopback mode.
- s32 synopGMAC_read_version (synopGMACdevice *gmacdev)
  Function to read the GMAC IP Version and populates the same in device data structure.
- s32 synopGMAC_reset (synopGMACdevice *gmacdev)
  Function to reset the GMAC core.
- s32 synopGMAC_dma_bus_mode_init (synopGMACdevice *gmacdev, u32 init_value)
  Function to program DMA bus mode register.
- s32 synopGMAC_dma_control_init (synopGMACdevice *gmacdev, u32 init_value)
  Function to program DMA Control register.

- void synopGMAC_wd_enable (synopGMACdevice *gmacdev)
  Enable the watchdog timer on the receiver.
- void synopGMAC_wd_disable (synopGMACdevice *gmacdev)
  Disable the watchdog timer on the receiver.
- void synopGMAC_jab_enable (synopGMACdevice *gmacdev)
  Enables the Jabber frame support.
- void synopGMAC_jab_disable (synopGMACdevice *gmacdev)
  Disables the Jabber frame support.
- void synopGMAC_frame_burst_enable (synopGMACdevice *gmacdev)
  Enables Frame bursting (Only in Half Duplex Mode).
- void synopGMAC_frame_burst_disable (synopGMACdevice *gmacdev)
  Disables Frame bursting.
- void synopGMAC_jumbo_frame_enable (synopGMACdevice *gmacdev)
  Enable Jumbo frame support.
- void synopGMAC_jumbo_frame_disable (synopGMACdevice *gmacdev)
  Disable Jumbo frame support.
- void synopGMAC_disable_crs (synopGMACdevice *gmacdev)
  Disable Carrier sense.
- void synopGMAC_select_gmii (synopGMACdevice *gmacdev)
  Selects the GMII port.
- void synopGMAC_select_mii (synopGMACdevice *gmacdev)
  Selects the MII port.
- void synopGMAC_rx_own_enable (synopGMACdevice *gmacdev)
  Enables Receive Own bit (Only in Half Duplex Mode).
- void synopGMAC_rx_own_disable (synopGMACdevice *gmacdev)
  Disables Receive Own bit (Only in Half Duplex Mode).
- void synopGMAC_loopback_on (synopGMACdevice *gmacdev)
  Sets the GMAC in loopback mode.
- void synopGMAC_loopback_off (synopGMACdevice *gmacdev)
  Sets the GMAC in Normal mode.
- void synopGMAC_set_full_duplex (synopGMACdevice *gmacdev)
  Sets the GMAC core in Full-Duplex mode.
- void synopGMAC_set_half_duplex (synopGMACdevice *gmacdev)

Sets the GMAC core in Half-Duplex mode.

- void synopGMAC_retry_enable (synopGMACdevice *gmacdev)

  GMAC tries retransmission (Only in Half Duplex mode).

- void synopGMAC_retry_disable (synopGMACdevice *gmacdev)

  GMAC tries only one transmission (Only in Half Duplex mode).

- void synopGMAC_pad_crc_strip_enable (synopGMACdevice *gmacdev)

  GMAC strips the Pad/FCS field of incoming frames.

- void synopGMAC_pad_crc_strip_disable (synopGMACdevice *gmacdev)

  GMAC doesnot strips the Pad/FCS field of incoming frames.

- void synopGMAC_back_off_limit (synopGMACdevice *gmacdev, u32 value)

  GMAC programmed with the back off limit value.

- void synopGMAC_deferral_check_enable (synopGMACdevice *gmacdev)

  Enables the Deferral check in GMAC (Only in Half Duplex mode) GMAC issues a Frame Abort Status, along with the excessive deferral error bit set in the transmit frame status when transmit state machine is deferred for more than
  - *24,288 bit times in 10/100Mbps mode*
  - *155,680 bit times in 1000Mbps mode or Jumbo frame mode in 10/100Mbps operation.*

- void synopGMAC_deferral_check_disable (synopGMACdevice *gmacdev)

  Disables the Deferral check in GMAC (Only in Half Duplex mode).

- void synopGMAC_rx_enable (synopGMACdevice *gmacdev)

  Enable the reception of frames on GMII/MII.

- void synopGMAC_rx_disable (synopGMACdevice *gmacdev)

  Disable the reception of frames on GMII/MII.

- void synopGMAC_tx_enable (synopGMACdevice *gmacdev)

  Enable the transmission of frames on GMII/MII.

- void synopGMAC_tx_disable (synopGMACdevice *gmacdev)

  Disable the transmission of frames on GMII/MII.

- void synopGMAC_frame_filter_enable (synopGMACdevice *gmacdev)

  Enables reception of all the frames to application.

- void synopGMAC_frame_filter_disable (synopGMACdevice *gmacdev)

  Disables reception of all the frames to application.

- void synopGMAC_write_hash_table_high (synopGMACdevice *gmacdev, u32 data)

  Populates the Hash High register with the data supplied.

- void synopGMAC_write_hash_table_low (synopGMACdevice *gmacdev, u32 data)

  Populates the Hash Low register with the data supplied.

- void synopGMAC_hash_perfect_filter_enable (synopGMACdevice *gmacdev)
  Enables Hash or Perfect filter (only if Hash filter is enabled in H/W).
- void synopGMAC_Hash_filter_only_enable (synopGMACdevice *gmacdev)
  Enables only Hash(only if Hash filter is enabled in H/W).
- void synopGMAC_src_addr_filter_enable (synopGMACdevice *gmacdev)
  Enables Source address filtering.
- void synopGMAC_src_addr_filter_disable (synopGMACdevice *gmacdev)
  Disables Source address filtering.
- void synopGMAC_dst_addr_filter_inverse (synopGMACdevice *gmacdev)
  Enables Inverse Destination address filtering.
- void synopGMAC_dst_addr_filter_normal (synopGMACdevice *gmacdev)
  Enables the normal Destination address filtering.
- void synopGMAC_set_pass_control (synopGMACdevice *gmacdev, u32 passcontrol)
  Enables forwarding of control frames.
- void synopGMAC_broadcast_enable (synopGMACdevice *gmacdev)
  Enables Broadcast frames.
- void synopGMAC_broadcast_disable (synopGMACdevice *gmacdev)
  Disable Broadcast frames.
- void synopGMAC_multicast_enable (synopGMACdevice *gmacdev)
  Enables Multicast frames.
- void synopGMAC_multicast_disable (synopGMACdevice *gmacdev)
  Disable Multicast frames.
- void synopGMAC_multicast_hash_filter_enable (synopGMACdevice *gmacdev)
  Enables multicast hash filtering.
- void synopGMAC_multicast_hash_filter_disable (synopGMACdevice *gmacdev)
  Disables multicast hash filtering.
- void synopGMAC_promisc_enable (synopGMACdevice *gmacdev)
  Enables promiscous mode.
- void synopGMAC_promisc_disable (synopGMACdevice *gmacdev)
  Clears promiscous mode.
- void synopGMAC_unicast_hash_filter_enable (synopGMACdevice *gmacdev)
  Enables unicast hash filtering.
- void synopGMAC_unicast_hash_filter_disable (synopGMACdevice *gmacdev)

Disables multicast hash filtering.

- void synopGMAC_unicast_pause_frame_detect_enable (synopGMACdevice *gmacdev)
  Enables detection of pause frames with stations unicast address.
- void synopGMAC_unicast_pause_frame_detect_disable (synopGMACdevice *gmacdev)
  Disables detection of pause frames with stations unicast address.
- void synopGMAC_rx_flow_control_enable (synopGMACdevice *gmacdev)
  Rx flow control enable.
- void synopGMAC_rx_flow_control_disable (synopGMACdevice *gmacdev)
  Rx flow control disable.
- void synopGMAC_tx_flow_control_enable (synopGMACdevice *gmacdev)
  Tx flow control enable.
- void synopGMAC_tx_flow_control_disable (synopGMACdevice *gmacdev)
  Tx flow control disable.
- void synopGMAC_tx_activate_flow_control (synopGMACdevice *gmacdev)
  Initiate Flowcontrol operation.
- void synopGMAC_tx_deactivate_flow_control (synopGMACdevice *gmacdev)
  stops Flowcontrol operation.
- void synopGMAC_pause_control (synopGMACdevice *gmacdev)
  This enables the pause frame generation after programming the appropriate registers.
- s32 synopGMAC_mac_init (synopGMACdevice *gmacdev)
  Example mac initialization sequence.
- s32 synopGMAC_check_phy_init (synopGMACdevice *gmacdev)
  Checks and initialze phy.
- s32 synopGMAC_set_mac_addr (synopGMACdevice *gmacdev, u32 MacHigh, u32 MacLow, u8 *MacAddr)
  Sets the Mac address in to GMAC register.
- s32 synopGMAC_get_mac_addr (synopGMACdevice *gmacdev, u32 MacHigh, u32 MacLow, u8 *MacAddr)
  Get the Mac address in to the address specified.
- s32 synopGMAC_attach (synopGMACdevice *gmacdev, u32 macBase, u32 dmaBase, u32 phyBase)
  Attaches the synopGMAC device structure to the hardware.
- void synopGMAC_rx_desc_init_ring (DmaDesc *desc, bool last_ring_desc)
  Initialize the rx descriptors for ring or chain mode operation.

- void synopGMAC_tx_desc_init_ring (DmaDesc *desc, bool last_ring_desc)
  Initialize the tx descriptors for ring or chain mode operation.
- void synopGMAC_rx_desc_init_chain (DmaDesc *desc)
  Initialize the rx descriptors for chain mode of operation.
- void synopGMAC_tx_desc_init_chain (DmaDesc *desc)
  Initialize the rx descriptors for chain mode of operation.
- void synopGMAC_init_rx_desc_base (synopGMACdevice *gmacdev)
  Programs the DmaRxBaseAddress with the Rx descriptor base address.
- void synopGMAC_init_tx_desc_base (synopGMACdevice *gmacdev)
  Programs the DmaTxBaseAddress with the Tx descriptor base address.
- void synopGMAC_set_owner_dma (DmaDesc *desc)
  Makes the Dma as owner for this descriptor.
- void synopGMAC_set_desc_sof (DmaDesc *desc)
  set tx descriptor to indicate SOF.
- void synopGMAC_set_desc_eof (DmaDesc *desc)
  set tx descriptor to indicate EOF.
- bool synopGMAC_is_sof_in_rx_desc (DmaDesc *desc)
  checks whether this descriptor contains start of frame.
- bool synopGMAC_is_eof_in_rx_desc (DmaDesc *desc)
  checks whether this descriptor contains end of frame.
- bool synopGMAC_is_da_filter_failed (DmaDesc *desc)
  checks whether destination address filter failed in the rx frame.
- bool synopGMAC_is_sa_filter_failed (DmaDesc *desc)
  checks whether source address filter failed in the rx frame.
- bool synopGMAC_is_desc_owned_by_dma (DmaDesc *desc)
  Checks whether the descriptor is owned by DMA.
- u32 synopGMAC_get_rx_desc_frame_length (u32 status)
  returns the byte length of received frame including CRC.
- bool synopGMAC_is_desc_valid (u32 status)
  Checks whether the descriptor is valid if no errors such as CRC/Receive Error/Watchdog Timeout/Late collision/Giant Frame/Overflow/Descriptor error the descritpor is said to be a valid descriptor.
- bool synopGMAC_is_desc_empty (DmaDesc *desc)
  Checks whether the descriptor is empty.
- bool synopGMAC_is_rx_desc_valid (u32 status)

Checks whether the rx descriptor is valid.

- bool synopGMAC_is_tx_aborted (u32 status)

  Checks whether the tx is aborted due to collisions.

- bool synopGMAC_is_tx_carrier_error (u32 status)

  Checks whether the tx carrier error.

- u32 synopGMAC_get_tx_collision_count (u32 status)

  Gives the transmission collision count.

- bool synopGMAC_is_rx_frame_damaged (u32 status)

  Check for damaged frame due to overflow or collision.

- bool synopGMAC_is_rx_frame_collision (u32 status)

  Check for damaged frame due to collision.

- bool synopGMAC_is_rx_crc (u32 status)

  Check for receive CRC error.

- bool synopGMAC_is_frame_dribbling_errors (u32 status)

  Indicates rx frame has non integer multiple of bytes.

- bool synopGMAC_is_rx_frame_length_errors (u32 status)

  Indicates error in rx frame length.

- bool synopGMAC_is_last_rx_desc (synopGMACdevice *gmacdev, DmaDesc *desc)

  Checks whether this rx descriptor is last rx descriptor.

- bool synopGMAC_is_last_tx_desc (synopGMACdevice *gmacdev, DmaDesc *desc)

  Checks whether this tx descriptor is last tx descriptor.

- bool synopGMAC_is_rx_desc_chained (DmaDesc *desc)

  Checks whether this rx descriptor is in chain mode.

- bool synopGMAC_is_tx_desc_chained (DmaDesc *desc)

  Checks whether this tx descriptor is in chain mode.

- void synopGMAC_get_desc_data (DmaDesc *desc, u32 *Status, u32 *Buffer1, u32 *Length1, u32 *Data1, u32 *Buffer2, u32 *Length2, u32 *Data2)

  Driver Api to get the descriptor field information.

- s32 synopGMAC_get_tx_qptr (synopGMACdevice *gmacdev, u32 *Status, u32 *Buffer1, u32 *Length1, u32 *Data1, u32 *Buffer2, u32 *Length2, u32 *Data2)

  Get the index and address of Tx desc.

- s32 synopGMAC_set_tx_qptr (synopGMACdevice *gmacdev, u32 Buffer1, u32 Length1, u32 Data1, u32 Buffer2, u32 Length2, u32 Data2, u32 offload_needed)

  Populate the tx desc structure with the buffer address.

- s32 synopGMAC_set_rx_qptr (synopGMACdevice *gmacdev, u32 Buffer1, u32 Length1, u32 Data1, u32 Buffer2, u32 Length2, u32 Data2)

  Prepares the descriptor to receive packets.

- s32 synopGMAC_get_rx_qptr (synopGMACdevice *gmacdev, u32 *Status, u32 *Buffer1, u32 *Length1, u32 *Data1, u32 *Buffer2, u32 *Length2, u32 *Data2)

  Get back the descriptor from DMA after data has been received.

- void synopGMAC_clear_interrupt (synopGMACdevice *gmacdev)

  Clears all the pending interrupts.

- u32 synopGMAC_get_interrupt_type (synopGMACdevice *gmacdev)

  Returns the all unmasked interrupt status after reading the DmaStatus register.

- u32 synopGMAC_get_interrupt_mask (synopGMACdevice *gmacdev)

  Returns the interrupt mask.

- void synopGMAC_enable_interrupt (synopGMACdevice *gmacdev, u32 interrupts)

  Enable all the interrupts.

- void synopGMAC_disable_interrupt_all (synopGMACdevice *gmacdev)

  Disable all the interrupts.

- void synopGMAC_disable_interrupt (synopGMACdevice *gmacdev, u32 interrupts)

  Disable interrupt according to the bitfield supplied.

- void synopGMAC_enable_dma_rx (synopGMACdevice *gmacdev)

  Enable the DMA Reception.

- void synopGMAC_enable_dma_tx (synopGMACdevice *gmacdev)

  Enable the DMA Transmission.

- void synopGMAC_resume_dma_tx (synopGMACdevice *gmacdev)

  Resumes the DMA Transmission.

- void synopGMAC_resume_dma_rx (synopGMACdevice *gmacdev)

  Resumes the DMA Reception.

- void synopGMAC_take_desc_ownership (DmaDesc *desc)

  Take ownership of this Descriptor.

- void synopGMAC_take_desc_ownership_rx (synopGMACdevice *gmacdev)

  Take ownership of all the rx Descriptors.

- void synopGMAC_take_desc_ownership_tx (synopGMACdevice *gmacdev)

  Take ownership of all the rx Descriptors.

- void synopGMAC_disable_dma_tx (synopGMACdevice *gmacdev)

  Disable the DMA for Transmission.

- void synopGMAC_disable_dma_rx (synopGMACdevice *gmacdev)
  Disable the DMA for Reception.
- void synopGMAC_pmt_int_enable (synopGMACdevice *gmacdev)
  Enables the assertion of PMT interrupt.
- void synopGMAC_pmt_int_disable (synopGMACdevice *gmacdev)
  Disables the assertion of PMT interrupt.
- void synopGMAC_power_down_enable (synopGMACdevice *gmacdev)
  Enables the power down mode of GMAC.
- void synopGMAC_power_down_disable (synopGMACdevice *gmacdev)
  Disables the powerd down setting of GMAC.
- void synopGMAC_enable_pmt_interrupt (synopGMACdevice *gmacdev)
  Enables the pmt interrupt generation in powerdown mode.
- void synopGMAC_disable_pmt_interrupt (synopGMACdevice *gmacdev)
  Disables the pmt interrupt generation in powerdown mode.
- void synopGMAC_magic_packet_enable (synopGMACdevice *gmacdev)
  Enables GMAC to look for Magic packet.
- void synopGMAC_wakeup_frame_enable (synopGMACdevice *gmacdev)
  Enables GMAC to look for wake up frame.
- void synopGMAC_pmt_unicast_enable (synopGMACdevice *gmacdev)
  Enables wake-up frame filter to handle unicast packets.
- bool synopGMAC_is_magic_packet_received (synopGMACdevice *gmacdev)
  Checks whether the packet received is a magic packet?.
- bool synopGMAC_is_wakeup_frame_received (synopGMACdevice *gmacdev)
  Checks whether the packet received is a wakeup frame?.
- void synopGMAC_write_wakeup_frame_register (synopGMACdevice *gmacdev, u32 *filter_contents)
  Populates the remote wakeup frame registers.
- void synopGMAC_mmc_counters_stop (synopGMACdevice *gmacdev)
  Freezes the MMC counters.
- void synopGMAC_mmc_counters_resume (synopGMACdevice *gmacdev)
  Resumes the MMC counter updation.
- void synopGMAC_mmc_counters_set_selfclear (synopGMACdevice *gmacdev)
  Configures the MMC in Self clearing mode.
- void synopGMAC_mmc_counters_reset_selfclear (synopGMACdevice *gmacdev)

Configures the MMC in non-Self clearing mode.

- void synopGMAC_mmc_counters_disable_rollover (synopGMACdevice *gmacdev)
  Configures the MMC to stop rollover.
- void synopGMAC_mmc_counters_enable_rollover (synopGMACdevice *gmacdev)
  Configures the MMC to rollover.
- u32 synopGMAC_read_mmc_counter (synopGMACdevice *gmacdev, u32 counter)
  Read the MMC Counter.
- u32 synopGMAC_read_mmc_rx_int_status (synopGMACdevice *gmacdev)
  Read the MMC Rx interrupt status.
- u32 synopGMAC_read_mmc_tx_int_status (synopGMACdevice *gmacdev)
  Read the MMC Tx interrupt status.
- void synopGMAC_disable_mmc_tx_interrupt (synopGMACdevice *gmacdev, u32 mask)
  Disable the MMC Tx interrupt.
- void synopGMAC_enable_mmc_tx_interrupt (synopGMACdevice *gmacdev, u32 mask)
  Enable the MMC Tx interrupt.
- void synopGMAC_disable_mmc_rx_interrupt (synopGMACdevice *gmacdev, u32 mask)
  Disable the MMC Rx interrupt.
- void synopGMAC_enable_mmc_rx_interrupt (synopGMACdevice *gmacdev, u32 mask)
  Enable the MMC Rx interrupt.
- void synopGMAC_disable_mmc_ipc_rx_interrupt (synopGMACdevice *gmacdev, u32 mask)
  Disable the MMC ipc rx checksum offload interrupt.
- void synopGMAC_enable_mmc_ipc_rx_interrupt (synopGMACdevice *gmacdev, u32 mask)
  Enable the MMC ipc rx checksum offload interrupt.
- void synopGMAC_enable_rx_chksum_offload (synopGMACdevice *gmacdev)
  Enables the ip checksum offloading in receive path.
- void synopGMAC_disable_rx_Ipchecksum_offload (synopGMACdevice *gmacdev)
  Disable the ip checksum offloading in receive path.
- void synopGMAC_rx_tcpip_chksum_drop_enable (synopGMACdevice *gmacdev)
  Instruct the DMA to drop the packets fails tcp ip checksum.
- void synopGMAC_rx_tcpip_chksum_drop_disable (synopGMACdevice *gmacdev)
  Instruct the DMA not to drop the packets even if it fails tcp ip checksum.
- u32 synopGMAC_is_rx_checksum_error (synopGMACdevice *gmacdev, u32 status)
  When the Enhanced Descriptor is enabled then the bit 0 of RDES0 indicates whether the Extended Status is available (RDES4).

- bool synopGMAC_is_tx_ipv4header_checksum_error (synopGMACdevice *gmacdev, u32 status)

  Checks if any Ipv4 header checksum error in the frame just transmitted.

- bool synopGMAC_is_tx_payload_checksum_error (synopGMACdevice *gmacdev, u32 status)

  Checks if any payload checksum error in the frame just transmitted.

- void synopGMAC_tx_checksum_offload_bypass (synopGMACdevice *gmacdev, DmaDesc *desc)

  The check summ offload engine is bypassed in the tx path.

- void synopGMAC_tx_checksum_offload_ipv4hdr (synopGMACdevice *gmacdev, DmaDesc *desc)

  The check summ offload engine is enabled to do only IPV4 header checksum.

- void synopGMAC_tx_checksum_offload_tcponly (synopGMACdevice *gmacdev, DmaDesc *desc)

  The check summ offload engine is enabled to do TCPIP checsum assuming Pseudo header is available.

- void synopGMAC_tx_checksum_offload_tcp_pseudo (synopGMACdevice *gmacdev, DmaDesc *desc)

  The check summ offload engine is enabled to do complete checksum computation.

- void synopGMAC_TS_enable (synopGMACdevice *gmacdev)

  This function enables the timestamping.

- void synopGMAC_TS_disable (synopGMACdevice *gmacdev)

  This function disables the timestamping.

- void synopGMAC_TS_int_enable (synopGMACdevice *gmacdev)

  Enable the interrupt to get timestamping interrupt.

- void synopGMAC_TS_int_disable (synopGMACdevice *gmacdev)

  Disable the interrupt to get timestamping interrupt.

- void synopGMAC_TS_mac_addr_filt_enable (synopGMACdevice *gmacdev)

  Enable MAC address for PTP frame filtering.

- void synopGMAC_TS_mac_addr_filt_disable (synopGMACdevice *gmacdev)

  Disables MAC address for PTP frame filtering.

- void synopGMAC_TS_set_clk_type (synopGMACdevice *gmacdev, u32 clk_type)

  Selet the type of clock mode for PTP.

- void synopGMAC_TS_master_enable (synopGMACdevice *gmacdev)

  Enable Snapshot for messages relevant to Master.

- void synopGMAC_TS_master_disable (synopGMACdevice *gmacdev)

  Disable Snapshot for messages relevant to Master.

- void synopGMAC_TS_event_enable (synopGMACdevice *gmacdev)

  Enable Snapshot for Event messages.

- void synopGMAC_TS_event_disable (synopGMACdevice *gmacdev)

  Disable Snapshot for Event messages.

- void synopGMAC_TS_IPV4_enable (synopGMACdevice *gmacdev)

  Enable time stamp snapshot for IPV4 frames.

- void synopGMAC_TS_IPV4_disable (synopGMACdevice *gmacdev)

  Disable time stamp snapshot for IPV4 frames.

- void synopGMAC_TS_IPV6_enable (synopGMACdevice *gmacdev)

  Enable time stamp snapshot for IPV6 frames.

- void synopGMAC_TS_IPV6_disable (synopGMACdevice *gmacdev)

  Disable time stamp snapshot for IPV6 frames.

- void synopGMAC_TS_ptp_over_ethernet_enable (synopGMACdevice *gmacdev)

  Enable time stamp snapshot for PTP over Ethernet frames.

- void synopGMAC_TS_ptp_over_ethernet_disable (synopGMACdevice *gmacdev)

  Disable time stamp snapshot for PTP over Ethernet frames.

- void synopGMAC_TS_pkt_snoop_ver2 (synopGMACdevice *gmacdev)

  Snoop PTP packet for version 2 format When set the PTP packets are snooped using the version 2 format.

- void synopGMAC_TS_pkt_snoop_ver1 (synopGMACdevice *gmacdev)

  Snoop PTP packet for version 2 format When set the PTP packets are snooped using the version 2 format.

- void synopGMAC_TS_digital_rollover_enable (synopGMACdevice *gmacdev)

  Timestamp digital rollover When set the timestamp low register rolls over after 0x3B9A_C9FF value.

- void synopGMAC_TS_binary_rollover_enable (synopGMACdevice *gmacdev)

  Timestamp binary rollover When set the timestamp low register rolls over after 0x7FFF_FFFF value.

- void synopGMAC_TS_all_frames_enable (synopGMACdevice *gmacdev)

  Enable Time Stamp for All frames When set the timestamp snap shot is enabled for all frames received by the core.

- void synopGMAC_TS_all_frames_disable (synopGMACdevice *gmacdev)

  Disable Time Stamp for All frames When reset the timestamp snap shot is not enabled for all frames received by the core.

- s32 synopGMAC_TS_addend_update (synopGMACdevice *gmacdev, u32 addend_value)

  Addend Register Update This function loads the contents of Time stamp addend register with the supplied 32 value.

- s32 synopGMAC_TS_timestamp_update (synopGMACdevice *gmacdev, u32 high_value, u32 low_value)

time stamp Update This function updates (adds/subtracts) with the value specified in the Timestamp High Update and Timestamp Low Update register.

- s32 synopGMAC_TS_timestamp_init (synopGMACdevice *gmacdev, u32 high_value, u32 low_value)

  time stamp Initialize This function Loads/Initializes h the value specified in the Timestamp High Update and Timestamp Low Update register.

- void synopGMAC_TS_coarse_update (synopGMACdevice *gmacdev)

  Time Stamp Update Coarse When reset the timestamp update is done using coarse method.

- void synopGMAC_TS_fine_update (synopGMACdevice *gmacdev)

  Time Stamp Update Fine When reset the timestamp update is done using Fine method.

- void synopGMAC_TS_subsecond_init (synopGMACdevice *gmacdev, u32 sub_sec_inc_value)

  Load the Sub Second Increment value in to Sub Second increment register.

- void synopGMAC_TS_read_timestamp (synopGMACdevice *gmacdev, u16 *higher_sec_val, u32 *sec_val, u32 *sub_sec_val)

  Reads the time stamp contents in to the respective pointers These registers are readonly.

- void synopGMAC_TS_load_timestamp_higher_val (synopGMACdevice *gmacdev, u32 higher_sec_val)

  Loads the time stamp higher sec value from the value supplied.

- void synopGMAC_TS_read_timestamp_higher_val (synopGMACdevice *gmacdev, u16 *higher_sec_val)

  Reads the time stamp higher sec value to respective pointers.

- void synopGMAC_TS_load_target_timestamp (synopGMACdevice *gmacdev, u32 sec_val, u32 sub_sec_val)

  Load the Target time stamp registers This function Loads the target time stamp registers with the values proviced.

- void synopGMAC_TS_read_target_timestamp (synopGMACdevice *gmacdev, u32 *sec_val, u32 *sub_sec_val)

  Reads the Target time stamp registers This function Loads the target time stamp registers with the values proviced.

4.1.1 Detailed Description

This file defines the synopsys GMAC device dependent functions.

Most of the operations on the GMAC device are available in this file. Functions for initiliasing and accessing MAC/DMA/PHY registers and the DMA descriptors are encapsulated in this file. The functions are platform/host/OS independent. These functions in turn use the low level device dependent (HAL) functions to access the register space.

4.1.2 Function Documentation

4.1.2.1 s32 synopGMAC_attach (synopGMACdevice * *gmacdev*, u32 *macBase*, u32 *dmaBase*, u32 *phyBase*)

Attaches the synopGMAC device structure to the hardware.

Device structure is populated with MAC/DMA and PHY base addresses.

Parameters:

← ***pointer*** to synopGMACdevice to populate mac dma and phy addresses.

← ***GMAC*** IP mac base address.

← ***GMAC*** IP dma base address.

← ***GMAC*** IP phy base address.

Returns:

0 upon success. Error code upon failure.

Note:

This is important function. No kernel api provided by Synopsys

4.1.2.2 void synopGMAC_back_off_limit (synopGMACdevice * *gmacdev*, u32 *value*)

GMAC programmed with the back off limit value.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

This function is tightly coupled with synopGMAC_retry_enable(synopGMACdevice * gmacdev)

4.1.2.3 void synopGMAC_broadcast_disable (synopGMACdevice * *gmacdev*)

Disable Broadcast frames.

When disabled Address filtering module filters all incoming broadcast frames.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.4 void synopGMAC_broadcast_enable (synopGMACdevice * *gmacdev*)

Enables Broadcast frames.

When enabled Address filtering module passes all incoming broadcast frames.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.5 s32 synopGMAC_check_phy_init (synopGMACdevice * *gmacdev*)

Checks and initialze phy.

This function checks whether the phy initialization is complete.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

0 if success else returns the error number.

4.1.2.6 void synopGMAC_clear_interrupt (synopGMACdevice * *gmacdev*)

Clears all the pending interrupts.

If the Dma status register is read then all the interrupts gets cleared

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.7 void synopGMAC_deferral_check_disable (synopGMACdevice * *gmacdev*)

Disables the Deferral check in GMAC (Only in Half Duplex mode).

GMAC defers until the CRS signal goes inactive.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.8 void synopGMAC_deferral_check_enable (synopGMACdevice * *gmacdev*)

Enables the Deferral check in GMAC (Only in Half Duplex mode) GMAC issues a Frame Abort Status, along with the excessive deferral error bit set in the transmit frame status when transmit state machine is deferred for more than

- 24,288 bit times in 10/100Mbps mode
- 155,680 bit times in 1000Mbps mode or Jumbo frame mode in 10/100Mbps operation.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

Deferral begins when transmitter is ready to transmit, but is prevented because of an active CRS (carrier sense)

4.1.2.9 void synopGMAC_disable_crs (synopGMACdevice * *gmacdev*)

Disable Carrier sense.

When Disabled GMAC ignores CRS signal during frame transmission in half duplex mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.10 void synopGMAC_disable_dma_rx (synopGMACdevice * *gmacdev*)

Disable the DMA for Reception.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.11 void synopGMAC_disable_dma_tx (synopGMACdevice * *gmacdev*)

Disable the DMA for Transmission.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.12 void synopGMAC_disable_interrupt (synopGMACdevice * *gmacdev*, u32 *interrupts*)

Disable interrupt according to the bitfield supplied.

Disables only those interrupts specified in the bit mask in second argument.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***bit*** mask for interrupts to be disabled.

Returns:

returns void.

4.1.2.13 void synopGMAC_disable_interrupt_all (synopGMACdevice * *gmacdev*)

Disable all the interrupts.

Disables all DMA interrupts.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

This function disabled all the interrupts, if you want to disable a particular interrupt then use synopGMAC_disable_interrupt().

4.1.2.14 void synopGMAC_disable_mmc_ipc_rx_interrupt (synopGMACdevice * *gmacdev*, u32 *mask*)

Disable the MMC ipc rx checksum offload interrupt.

The MMC ipc rx checksum offload interrupts are masked out as per the mask specified.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***rx*** interrupt bit mask for which interrupts needs to be disabled.

Returns:

returns void.

4.1.2.15 void synopGMAC_disable_mmc_rx_interrupt (synopGMACdevice * *gmacdev*, u32 *mask*)

Disable the MMC Rx interrupt.

The MMC rx interrupts are masked out as per the mask specified.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***rx*** interrupt bit mask for which interrupts needs to be disabled.

Returns:

returns void.

4.1.2.16 void synopGMAC_disable_mmc_tx_interrupt (synopGMACdevice * *gmacdev*, u32 *mask*)

Disable the MMC Tx interrupt.

The MMC tx interrupts are masked out as per the mask specified.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***tx*** interrupt bit mask for which interrupts needs to be disabled.

Returns:

returns void.

4.1.2.17 void synopGMAC_disable_pmt_interrupt (synopGMACdevice * *gmacdev*)

Disables the pmt interrupt generation in powerdown mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.18 void synopGMAC_disable_rx_Ipchecksum_offload (synopGMACdevice * *gmacdev*)

Disable the ip checksum offloading in receive path.

Ip checksum offloading is disabled in the receive path.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.19 s32 synopGMAC_dma_bus_mode_init (synopGMACdevice * *gmacdev*, u32 *init_value*)

Function to program DMA bus mode register.

The Bus Mode register is programmed with the value given. The bits to be set are bit wise or'ed and sent as the second argument to this function.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***the*** data to be programmed.

Returns:

0 on success else return the error status.

4.1.2.20 s32 synopGMAC_dma_control_init (synopGMACdevice * *gmacdev*, u32 *init_value*)

Function to program DMA Control register.

The Dma Control register is programmed with the value given. The bits to be set are bit wise or'ed and sent as the second argument to this function.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***the*** data to be programmed.

Returns:

0 on success else return the error status.

4.1.2.21 void synopGMAC_dst_addr_filter_inverse (synopGMACdevice * *gmacdev*)

Enables Inverse Destination address filtering.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.22 void synopGMAC_dst_addr_filter_normal (synopGMACdevice * *gmacdev*)

Enables the normal Destination address filtering.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.23 void synopGMAC_enable_dma_rx (synopGMACdevice * *gmacdev*)

Enable the DMA Reception.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.24 void synopGMAC_enable_dma_tx (synopGMACdevice * *gmacdev*)

Enable the DMA Transmission.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.25 void synopGMAC_enable_interrupt (synopGMACdevice * *gmacdev*, u32 *interrupts*)

Enable all the interrupts.

Enables the DMA interrupt as specified by the bit mask.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***bit*** mask of interrupts to be enabled.

Returns:

returns void.

4.1.2.26 void synopGMAC_enable_mmc_ipc_rx_interrupt (synopGMACdevice * *gmacdev*, u32 *mask*)

Enable the MMC ipc rx checksum offload interrupt.

The MMC ipc rx checksum offload interrupts are enabled as per the mask specified.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***rx*** interrupt bit mask for which interrupts needs to be enabled.

Returns:

returns void.

4.1.2.27 void synopGMAC_enable_mmc_rx_interrupt (synopGMACdevice * *gmacdev*, u32 *mask*)

Enable the MMC Rx interrupt.

The MMC rx interrupts are enabled as per the mask specified.

Parameters:

← ***pointer*** to synopGMACdevice.
← ***rx*** interrupt bit mask for which interrupts needs to be enabled.

Returns:

returns void.

4.1.2.28 void synopGMAC_enable_mmc_tx_interrupt (synopGMACdevice * *gmacdev*, u32 *mask*)

Enable the MMC Tx interrupt.

The MMC tx interrupts are enabled as per the mask specified.

Parameters:

← ***pointer*** to synopGMACdevice.
← ***tx*** interrupt bit mask for which interrupts needs to be enabled.

Returns:

returns void.

4.1.2.29 void synopGMAC_enable_pmt_interrupt (synopGMACdevice * *gmacdev*)

Enables the pmt interrupt generation in powerdown mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.30 void synopGMAC_enable_rx_chksum_offload (synopGMACdevice * *gmacdev*)

Enables the ip checksum offloading in receive path.

When set GMAC calculates 16 bit 1's complement of all received ethernet frame payload. It also checks IPv4 Header checksum is correct. GMAC core appends the 16 bit checksum calculated for payload of IP datagram and appends it to Ethernet frame transferred to the application.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.31 void synopGMAC_frame_burst_disable (synopGMACdevice * *gmacdev*)

Disables Frame bursting.

When Disabled, frame bursting is not supported.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.32 void synopGMAC_frame_burst_enable (synopGMACdevice * *gmacdev*)

Enables Frame bursting (Only in Half Duplex Mode).

When enabled, GMAC allows frame bursting in GMII Half Duplex mode. Reserved in 10/100 and Full-Duplex configurations.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.33 void synopGMAC_frame_filter_disable (synopGMACdevice * *gmacdev*)

Disables reception of all the frames to application.

GMAC passes only those received frames to application which pass SA/DA address filtering.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.34 void synopGMAC_frame_filter_enable (synopGMACdevice * *gmacdev*)

Enables reception of all the frames to application.

GMAC passes all the frames received to application irrespective of whether they pass SA/DA address filtering or not.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.35 void synopGMAC_get_desc_data (DmaDesc * *desc*, u32 * *Status*, u32 * *Buffer1*, u32 * *Length1*, u32 * *Data1*, u32 * *Buffer2*, u32 * *Length2*, u32 * *Data2*)

Driver Api to get the descriptor field information.

This returns the status, dma-able address of buffer1, the length of buffer1, virtual address of buffer1 dma-able address of buffer2, length of buffer2, virtural adddress of buffer2.

Parameters:

← ***pointer*** to DmaDesc structure.

→ ***pointer*** to status field fo descriptor.

→ ***dma-able*** address of buffer1.

→ ***length*** of buffer1.

→ ***virtual*** address of buffer1.

→ ***dma-able*** address of buffer2.

→ ***length*** of buffer2.

→ ***virtual*** address of buffer2.

Returns:

returns void.

4.1.2.36 u32 synopGMAC_get_interrupt_mask (synopGMACdevice * *gmacdev*)

Returns the interrupt mask.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

0 upon success. Error code upon failure.

4.1.2.37 u32 synopGMAC_get_interrupt_type (synopGMACdevice * *gmacdev*)

Returns the all unmasked interrupt status after reading the DmaStatus register.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

0 upon success. Error code upon failure.

4.1.2.38 s32 synopGMAC_get_mac_addr (synopGMACdevice * *gmacdev*, u32 *MacHigh*, u32 *MacLow*, u8 * *MacAddr*)

Get the Mac address in to the address specified.

The mac register contents are read and written to buffer passed.

Parameters:

← ***pointer*** to synopGMACdevice to populate mac dma and phy addresses.

← ***Register*** offset for Mac address high

← ***Register*** offset for Mac address low

→ ***buffer*** containing the device mac address.

Returns:

0 upon success. Error code upon failure.

4.1.2.39 u32 synopGMAC_get_mdc_clk_div (synopGMACdevice * *gmacdev*)

Returns the current MDC divider value programmed in the ip.

Parameters:

← ***pointer*** to device structure.

← ***clk*** divider value.

Returns:

Returns the MDC divider value read.

4.1.2.40 u32 synopGMAC_get_rx_desc_frame_length (u32 *status*)

returns the byte length of received frame including CRC.

This returns the no of bytes received in the received ethernet frame including CRC(FCS).

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns the length of received frame lengths in bytes.

4.1.2.41 s32 synopGMAC_get_rx_qptr (synopGMACdevice * *gmacdev*, u32 * *Status*, u32 * *Buffer1*, u32 * *Length1*, u32 * *Data1*, u32 * *Buffer2*, u32 * *Length2*, u32 * *Data2*)

Get back the descriptor from DMA after data has been received.

When the DMA indicates that the data is received (interrupt is generated), this function should be called to get the descriptor and hence the data buffers received. With successful return from this function caller gets the descriptor fields for processing. check the parameters to understand the fields returned.‘

Parameters:

← ***pointer*** to synopGMACdevice.
→ ***pointer*** to hold the status of DMA.
→ ***Dma-able*** buffer1 pointer.
→ ***pointer*** to hold length of buffer1 (Max is 2048).
→ ***virtual*** pointer for buffer1.
→ ***Dma-able*** buffer2 pointer.
→ ***pointer*** to hold length of buffer2 (Max is 2048).
→ ***virtual*** pointer for buffer2.

Returns:

returns present rx descriptor index on success. Negative value if error.

4.1.2.42 u32 synopGMAC_get_tx_collision_count (u32 *status*)

Gives the transmission collision count.

returns the transmission collision count indicating number of collisions occured before the frame was transmitted. Make sure to check excessive collision didnot happen to ensure the count is valid.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns the count value of collision.

4.1.2.43 s32 synopGMAC_get_tx_qptr (synopGMACdevice * *gmacdev*, u32 * *Status*, u32 * *Buffer1*, u32 * *Length1*, u32 * *Data1*, u32 * *Buffer2*, u32 * *Length2*, u32 * *Data2*)

Get the index and address of Tx desc.

This api is same for both ring mode and chain mode. This function tracks the tx descriptor the DMA just closed after the transmission of data from this descriptor is over. This returns the descriptor fields to the caller.

Parameters:

← ***pointer*** to synopGMACdevice.
→ ***status*** field of the descriptor.
→ ***Dma-able*** buffer1 pointer.
→ ***length*** of buffer1 (Max is 2048).
→ ***virtual*** pointer for buffer1.
→ ***Dma-able*** buffer2 pointer.
→ ***length*** of buffer2 (Max is 2048).
→ ***virtual*** pointer for buffer2.
→ ***u32*** data indicating whether the descriptor is in ring mode or chain mode.

Returns:

returns present tx descriptor index on success. Negative value if error.

4.1.2.44 void synopGMAC_Hash_filter_only_enable (synopGMACdevice * *gmacdev*)

Enables only Hash(only if Hash filter is enabled in H/W).

Only frames matching Hash Filtering as per HMC and HUC configuration are sent to application.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.45 void synopGMAC_hash_perfect_filter_enable (synopGMACdevice * *gmacdev*)

Enables Hash or Perfect filter (only if Hash filter is enabled in H/W).

Only frames matching either perfect filtering or Hash Filtering as per HMC and HUC configuration are sent to application.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.46 void synopGMAC_init_rx_desc_base (synopGMACdevice * *gmacdev*)

Programs the DmaRxBaseAddress with the Rx descriptor base address.

Rx Descriptor's base address is available in the gmacdev structure. This function progrms the Dma Rx Base address with the starting address of the descriptor ring or chain.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.47 void synopGMAC_init_tx_desc_base (synopGMACdevice * *gmacdev*)

Programs the DmaTxBaseAddress with the Tx descriptor base address.

Tx Descriptor's base address is available in the gmacdev structure. This function progrms the Dma Tx Base address with the starting address of the descriptor ring or chain.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.48 bool synopGMAC_is_da_filter_failed (DmaDesc * *desc*)

checks whether destination address filter failed in the rx frame.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if Failed, false if not.

4.1.2.49 bool synopGMAC_is_desc_empty (DmaDesc * *desc*)

Checks whether the descriptor is empty.

If the buffer1 and buffer2 lengths are zero in ring mode descriptor is empty. In chain mode buffer2 length is 0 but buffer2 itself contains the next descriptor address.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if descriptor is empty, false if not empty.

4.1.2.50 bool synopGMAC_is_desc_owned_by_dma (DmaDesc * *desc*)

Checks whether the descriptor is owned by DMA.

If descriptor is owned by DMA then the OWN bit is set to 1. This API is same for both ring and chain mode.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if Dma owns descriptor and false if not.

4.1.2.51 bool synopGMAC_is_desc_valid (u32 *status*)

Checks whether the descriptor is valid if no errors such as CRC/Receive Error/Watchdog Timeout/Late collision/Giant Frame/Overflow/Descriptor error the descritpor is said to be a valid descriptor.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

True if desc valid. false if error.

4.1.2.52 bool synopGMAC_is_eof_in_rx_desc (DmaDesc * *desc*)

checks whether this descriptor contains end of frame.

This function is to check whether the descriptor's data buffer contains end of ethernet frame?

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if SOF in current descriptor, else returns fail.

4.1.2.53 bool synopGMAC_is_frame_dribbling_errors (u32 *status*)

Indicates rx frame has non integer multiple of bytes.

(odd nibbles). Retruns true if dribbling error in rx frame.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if error else returns false.

4.1.2.54 bool synopGMAC_is_last_rx_desc (synopGMACdevice * *gmacdev*, DmaDesc * *desc*)

Checks whether this rx descriptor is last rx descriptor.

This returns true if it is last descriptor either in ring mode or in chain mode.

Parameters:

← ***pointer*** to devic structure.

← ***pointer*** to DmaDesc structure.

Returns:

returns true if it is last descriptor, false if not.

Note:

This function should not be called before initializing the descriptor using synopGMAC_desc_init().

4.1.2.55 bool synopGMAC_is_last_tx_desc (synopGMACdevice * *gmacdev*, DmaDesc * *desc*)

Checks whether this tx descriptor is last tx descriptor.

This returns true if it is last descriptor either in ring mode or in chain mode.

Parameters:

← ***pointer*** to devic structure.

← ***pointer*** to DmaDesc structure.

Returns:

returns true if it is last descriptor, false if not.

Note:

This function should not be called before initializing the descriptor using synopGMAC_desc_init().

4.1.2.56 bool synopGMAC_is_magic_packet_received (synopGMACdevice * *gmacdev*)

Checks whether the packet received is a magic packet?.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns True if magic packet received else returns false.

4.1.2.57 u32 synopGMAC_is_rx_checksum_error (synopGMACdevice * *gmacdev*, u32 *status*)

When the Enhanced Descriptor is enabled then the bit 0 of RDES0 indicates whether the Extended Status is available (RDES4).

Time Stamp feature and the Checksum Offload Engine2 makes use of this extended status to provide the status of the received packet.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns TRUE or FALSE Decodes the Rx Descriptor status to various checksum error conditions.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***u32*** status field of the corresponding descriptor.

Returns:

returns decoded enum (u32) indicating the status.

4.1.2.58 bool synopGMAC_is_rx_crc (u32 *status*)

Check for receive CRC error.

Retruns true if rx frame CRC error occured.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if error else returns false.

4.1.2.59 bool synopGMAC_is_rx_desc_chained (DmaDesc * *desc*)

Checks whether this rx descriptor is in chain mode.

This returns true if it is this descriptor is in chain mode.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if chain mode is set, false if not.

4.1.2.60 bool synopGMAC_is_rx_desc_valid (u32 *status*)

Checks whether the rx descriptor is valid.

if rx descripor is not in error and complete frame is available in the same descriptor

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if no error and first and last desc bits are set, otherwise it returns false.

4.1.2.61 bool synopGMAC_is_rx_frame_collision (u32 *status*)

Check for damaged frame due to collision.

Retruns true if rx frame was damaged due to late collision in half duplex mode.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if error else returns false.

4.1.2.62 bool synopGMAC_is_rx_frame_damaged (u32 *status*)

Check for damaged frame due to overflow or collision.

Retruns true if rx frame was damaged due to buffer overflow in MTL or late collision in half duplex mode.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if error else returns false.

4.1.2.63 bool synopGMAC_is_rx_frame_length_errors (u32 *status*)

Indicates error in rx frame length.

Retruns true if received frame length doesnot match with the length field

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if error else returns false.

4.1.2.64 bool synopGMAC_is_sa_filter_failed (DmaDesc * *desc*)

checks whether source address filter failed in the rx frame.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if Failed, false if not.

4.1.2.65 bool synopGMAC_is_sof_in_rx_desc (DmaDesc * *desc*)

checks whether this descriptor contains start of frame.

This function is to check whether the descriptor's data buffer contains a fresh ethernet frame?

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if SOF in current descriptor, else returns fail.

4.1.2.66 bool synopGMAC_is_tx_aborted (u32 *status*)

Checks whether the tx is aborted due to collisions.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if collisions, else returns false.

4.1.2.67 bool synopGMAC_is_tx_carrier_error (u32 *status*)

Checks whether the tx carrier error.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if carrier error occured, else returns falser.

4.1.2.68 bool synopGMAC_is_tx_desc_chained (DmaDesc * *desc*)

Checks whether this tx descriptor is in chain mode.

This returns true if it is this descriptor is in chain mode.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns true if chain mode is set, false if not.

4.1.2.69 bool synopGMAC_is_tx_ipv4header_checksum_error (synopGMACdevice * *gmacdev*, u32 *status*)

Checks if any Ipv4 header checksum error in the frame just transmitted.

This serves as indication that error occureed in the IPv4 header checksum insertion. The sent out frame doesnot carry any ipv4 header checksum inserted by the hardware.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***u32*** status field of the corresponding descriptor.

Returns:

returns true if error in ipv4 header checksum, else returns false.

4.1.2.70 bool synopGMAC_is_tx_payload_checksum_error (synopGMACdevice * *gmacdev*, u32 *status*)

Checks if any payload checksum error in the frame just transmitted.

This serves as indication that error occureed in the payload checksum insertion. The sent out frame doesnot carry any payload checksum inserted by the hardware.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***u32*** status field of the corresponding descriptor.

Returns:

returns true if error in ipv4 header checksum, else returns false.

4.1.2.71 bool synopGMAC_is_wakeup_frame_received (synopGMACdevice * *gmacdev*)

Checks whether the packet received is a wakeup frame?.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns true if wakeup frame received else returns false.

4.1.2.72 void synopGMAC_jab_disable (synopGMACdevice * *gmacdev*)

Disables the Jabber frame support.

When disabled, GMAC enables jabber timer. It cuts of transmitter if application sends more than 2048 bytes of data (10240 if Jumbo frame enabled).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.73 void synopGMAC_jab_enable (synopGMACdevice * *gmacdev*)

Enables the Jabber frame support.

When enabled, GMAC disabled the jabber timer, and can transfer 16,384 byte frames.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.74 void synopGMAC_jumbo_frame_disable (synopGMACdevice * *gmacdev*)

Disable Jumbo frame support.

When Disabled GMAC does not supports jumbo frames. Giant frame error is reported in receive frame status.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.75 void synopGMAC_jumbo_frame_enable (synopGMACdevice * *gmacdev*)

Enable Jumbo frame support.

When Enabled GMAC supports jumbo frames of 9018/9022(VLAN tagged). Giant frame error is not reported in receive frame status.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.76 void synopGMAC_loopback_off (synopGMACdevice * *gmacdev*)

Sets the GMAC in Normal mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.77 void synopGMAC_loopback_on (synopGMACdevice * *gmacdev*)

Sets the GMAC in loopback mode.

When on GMAC operates in loop-back mode at GMII/MII.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

(G)MII Receive clock is required for loopback to work properly, as transmit clock is not looped back internally.

4.1.2.78 s32 synopGMAC_mac_init (synopGMACdevice * *gmacdev*)

Example mac initialization sequence.

This function calls the initialization routines to initialize the GMAC register. One can change the functions invoked here to have different configuration as per the requirement

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

Returns 0 on success.

4.1.2.79 void synopGMAC_magic_packet_enable (synopGMACdevice * *gmacdev*)

Enables GMAC to look for Magic packet.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.80 void synopGMAC_mmc_counters_disable_rollover (synopGMACdevice * *gmacdev*)

Configures the MMC to stop rollover.

Programs MMC interface so that counters will not rollover after reaching maximum value.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.81 void synopGMAC_mmc_counters_enable_rollover (synopGMACdevice * *gmacdev*)

Configures the MMC to rollover.

Programs MMC interface so that counters will rollover after reaching maximum value.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.82 void synopGMAC_mmc_counters_reset_selfclear (synopGMACdevice * *gmacdev*)

Configures the MMC in non-Self clearing mode.

Programs MMC interface so that counters are cleared when the counters are read.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.83 void synopGMAC_mmc_counters_resume (synopGMACdevice * *gmacdev*)

Resumes the MMC counter updation.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.84 void synopGMAC_mmc_counters_set_selfclear (synopGMACdevice * *gmacdev*)

Configures the MMC in Self clearing mode.

Programs MMC interface so that counters are cleared when the counters are read.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.85 void synopGMAC_mmc_counters_stop (synopGMACdevice * *gmacdev*)

Freezes the MMC counters.

This function call freezes the MMC counters. None of the MMC counters are updated due to any tx or rx frames until synopGMAC_mmc_counters_resume is called.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.86 void synopGMAC_multicast_disable (synopGMACdevice * *gmacdev*)

Disable Multicast frames.

When disabled multicast frame filtering depends on HMC bit.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.87 void synopGMAC_multicast_enable (synopGMACdevice * *gmacdev*)

Enables Multicast frames.

When enabled all multicast frames are passed.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.88 void synopGMAC_multicast_hash_filter_disable (synopGMACdevice * *gmacdev*)

Disables multicast hash filtering.

When disabled GMAC performs perfect destination address filtering for multicast frames, it compares DA field with the value programmed in DA register.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.89 void synopGMAC_multicast_hash_filter_enable (synopGMACdevice * *gmacdev*)

Enables multicast hash filtering.

When enabled GMAC performs teh destination address filtering according to the hash table.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.90 void synopGMAC_pad_crc_strip_disable (synopGMACdevice * *gmacdev*)

GMAC doesnot strips the Pad/FCS field of incoming frames.

GMAC will pass all the incoming frames to Host unmodified.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.91 void synopGMAC_pad_crc_strip_enable (synopGMACdevice * *gmacdev*)

GMAC strips the Pad/FCS field of incoming frames.

This is true only if the length field value is less than or equal to 1500 bytes. All received frames with length field greater than or equal to 1501 bytes are passed to the application without stripping the Pad/FCS field.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.92 void synopGMAC_pause_control (synopGMACdevice * *gmacdev*)

This enables the pause frame generation after programming the appropriate registers.

presently activation is set at 3k and deactivation set at 4k. These may have to tweaked if found any issues

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.93 s32 synopGMAC_phy_loopback (synopGMACdevice * *gmacdev*, bool *loopback*)

Function to configure the phy in loopback mode.

Parameters:

← ***pointer*** to synopGMACdevice.
← ***enable*** or disable the loopback.

Returns:

0 on success else return the error status.

Note:

Don't get confused with mac loop-back synopGMAC_loopback_on(synopGMACdevice *) and synopGMAC_loopback_off(synopGMACdevice *) functions.

4.1.2.94 void synopGMAC_pmt_int_disable (synopGMACdevice * *gmacdev*)

Disables the assertion of PMT interrupt.

This disables the assertion of PMT interrupt due to Magic Pkt or Wakeup frame reception.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.95 void synopGMAC_pmt_int_enable (synopGMACdevice * *gmacdev*)

Enables the assertion of PMT interrupt.

This enables the assertion of PMT interrupt due to Magic Pkt or Wakeup frame reception.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.96 void synopGMAC_pmt_unicast_enable (synopGMACdevice * *gmacdev*)

Enables wake-up frame filter to handle unicast packets.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.97 void synopGMAC_power_down_disable (synopGMACdevice * *gmacdev*)

Disables the powerd down setting of GMAC.

If the driver wants to bring up the GMAC from powerdown mode, even though the magic packet or the wake up frames received from the network, this function should be called.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.98 void synopGMAC_power_down_enable (synopGMACdevice * *gmacdev*)

Enables the power down mode of GMAC.

This function puts the Gmac in power down mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.99 void synopGMAC_promisc_disable (synopGMACdevice * *gmacdev*)

Clears promiscous mode.

When called the GMAC falls back to normal operation from promiscous mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.100 void synopGMAC_promisc_enable (synopGMACdevice * *gmacdev*)

Enables promiscous mode.

When enabled Address filter modules pass all incoming frames regardless of their Destination and source addresses.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.101 u32 synopGMAC_read_mmc_counter (synopGMACdevice * *gmacdev*, u32 *counter*)

Read the MMC Counter.

Parameters:

← ***pointer*** to synopGMACdevice.
← ***the*** counter to be read.

Returns:

returns the read count value.

4.1.2.102 u32 synopGMAC_read_mmc_rx_int_status (synopGMACdevice * *gmacdev*)

Read the MMC Rx interrupt status.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns the Rx interrupt status.

4.1.2.103 u32 synopGMAC_read_mmc_tx_int_status (synopGMACdevice * *gmacdev*)

Read the MMC Tx interrupt status.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns the Tx interrupt status.

4.1.2.104 s32 synopGMAC_read_phy_reg (u32 * *RegBase*, u32 *PhyBase*, u32 *RegOffset*, u16 * *data*)

Function to read the Phy register.

The access to phy register is a slow process as the data is moved accross MDI/MDO interface

Parameters:

← ***pointer*** to Register Base (It is the mac base in our case) .
← ***PhyBase*** register is the index of one of supported 32 PHY devices.
← ***Register*** offset is the index of one of the 32 phy register.
→ ***u16*** data read from the respective phy register (only valid iff return value is 0).

Returns:

Returns 0 on success else return the error status.

4.1.2.105 s32 synopGMAC_read_version (synopGMACdevice * *gmacdev*)

Function to read the GMAC IP Version and populates the same in device data structure.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

Always return 0.

4.1.2.106 s32 synopGMAC_reset (synopGMACdevice * *gmacdev*)

Function to reset the GMAC core.

This reests the DMA and GMAC core. After reset all the registers holds their respective reset value

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

0 on success else return the error status.

4.1.2.107 void synopGMAC_resume_dma_rx (synopGMACdevice * *gmacdev*)

Resumes the DMA Reception.

the DmaRxPollDemand is written. (the data writeen could be anything). This forces the DMA to resume reception.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.108 void synopGMAC_resume_dma_tx (synopGMACdevice * *gmacdev*)

Resumes the DMA Transmission.

the DmaTxPollDemand is written. (the data writeen could be anything). This forces the DMA to resume transmission.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.109 void synopGMAC_retry_disable (synopGMACdevice * *gmacdev*)

GMAC tries only one transmission (Only in Half Duplex mode).

If collision occurs on the GMII/MII, GMAC will ignore the current frami transmission and report a frame abort with excessive collision in tranmit frame status.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.110 void synopGMAC_retry_enable (synopGMACdevice * *gmacdev*)

GMAC tries retransmission (Only in Half Duplex mode).

If collision occurs on the GMII/MII, GMAC attempt retries based on the back off limit configured.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

This function is tightly coupled with synopGMAC_back_off_limit(synopGMACdev *, u32).

4.1.2.111 void synopGMAC_rx_desc_init_chain (DmaDesc * *desc*)

Initialize the rx descriptors for chain mode of operation.

- Status field is initialized to 0.
- EndOfRing set for the last descriptor.
- buffer1 and buffer2 set to 0.
- data1 and data2 set to 0.

Parameters:

← ***pointer*** to DmaDesc structure.
← ***whether*** end of ring

Returns:

void.

4.1.2.112 void synopGMAC_rx_desc_init_ring (DmaDesc * *desc*, bool *last_ring_desc*)

Initialize the rx descriptors for ring or chain mode operation.

- Status field is initialized to 0.
- EndOfRing set for the last descriptor.
- buffer1 and buffer2 set to 0 for ring mode of operation. (note)
- data1 and data2 set to 0. (note)

Parameters:

← ***pointer*** to DmaDesc structure.
← ***whether*** end of ring

Returns:

void.

Note:

Initialization of the buffer1, buffer2, data1,data2 and status are not done here. This only initializes whether one wants to use this descriptor in chain mode or ring mode. For chain mode of operation the buffer2 and data2 are programmed before calling this function.

4.1.2.113 void synopGMAC_rx_disable (synopGMACdevice * *gmacdev*)

Disable the reception of frames on GMII/MII.

GMAC receive state machine is disabled after completion of reception of current frame.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.114 void synopGMAC_rx_enable (synopGMACdevice * *gmacdev*)

Enable the reception of frames on GMII/MII.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.115 void synopGMAC_rx_flow_control_disable (synopGMACdevice * *gmacdev*)

Rx flow control disable.

When disabled GMAC will not decode pause frame.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.116 void synopGMAC_rx_flow_control_enable (synopGMACdevice * *gmacdev*)

Rx flow control enable.

When Enabled GMAC will decode the rx pause frame and disable the tx for a specified time.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.117 void synopGMAC_rx_own_disable (synopGMACdevice * *gmacdev*)

Disables Receive Own bit (Only in Half Duplex Mode).

When enaled GMAC disables the reception of frames when gmii_txen_o is asserted.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.118 void synopGMAC_rx_own_enable (synopGMACdevice * *gmacdev*)

Enables Receive Own bit (Only in Half Duplex Mode).

When enaled GMAC receives all the packets given by phy while transmitting.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.119 void synopGMAC_rx_tcpip_chksum_drop_disable (synopGMACdevice * *gmacdev*)

Instruct the DMA not to drop the packets even if it fails tcp ip checksum.

This is to instruct the receive DMA engine to allow the packets even if recevied packet fails the tcp/ip checksum in hardware. Valid only when full checksum offloading is enabled(type-2).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.120 void synopGMAC_rx_tcpip_chksum_drop_enable (synopGMACdevice * *gmacdev*)

Instruct the DMA to drop the packets fails tcp ip checksum.

This is to instruct the receive DMA engine to drop the recevied packet if they fails the tcp/ip checksum in hardware. Valid only when full checksum offloading is enabled(type-2).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.121 void synopGMAC_select_gmii (synopGMACdevice * *gmacdev*)

Selects the GMII port.

When called GMII (1000Mbps) port is selected (programmable only in 10/100/1000 Mbps configuration).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.122 void synopGMAC_select_mii (synopGMACdevice * *gmacdev*)

Selects the MII port.

When called MII (10/100Mbps) port is selected (programmable only in 10/100/1000 Mbps configuration).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.123 void synopGMAC_set_desc_eof (DmaDesc * *desc*)

set tx descriptor to indicate EOF.

This descriptor contains the End of ethernet frame.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns void.

4.1.2.124 void synopGMAC_set_desc_sof (DmaDesc * *desc*)

set tx descriptor to indicate SOF.

This Descriptor contains the start of ethernet frame.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns void.

4.1.2.125 void synopGMAC_set_full_duplex (synopGMACdevice * *gmacdev*)

Sets the GMAC core in Full-Duplex mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.126 void synopGMAC_set_half_duplex (synopGMACdevice * *gmacdev*)

Sets the GMAC core in Half-Duplex mode.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.127 s32 synopGMAC_set_mac_addr (synopGMACdevice * *gmacdev*, u32 *MacHigh*, u32 *MacLow*, u8 * *MacAddr*)

Sets the Mac address in to GMAC register.

This function sets the MAC address to the MAC register in question.

Parameters:

← ***pointer*** to synopGMACdevice to populate mac dma and phy addresses.

← ***Register*** offset for Mac address high

← ***Register*** offset for Mac address low

← ***buffer*** containing mac address to be programmed.

Returns:

0 upon success. Error code upon failure.

4.1.2.128 s32 synopGMAC_set_mdc_clk_div (synopGMACdevice * *gmacdev*, u32 *clk_div_val*)

Function to set the MDC clock for mdio transactiona.

Parameters:

← ***pointer*** to device structure.

← ***clk*** divider value.

Returns:

Reuturns 0 on success else return the error value.

4.1.2.129 void synopGMAC_set_owner_dma (DmaDesc * *desc*)

Makes the Dma as owner for this descriptor.

This function sets the own bit of status field of the DMA descriptor, indicating the DMA is the owner for this descriptor.

Parameters:

← ***pointer*** to DmaDesc structure.

Returns:

returns void.

4.1.2.130 void synopGMAC_set_pass_control (synopGMACdevice * *gmacdev*, u32 *passcontrol*)

Enables forwarding of control frames.

When set forwards all the control frames (incl. unicast and multicast PAUSE frames).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

Note:

Depends on RFE of FlowControlRegister[2]

4.1.2.131 s32 synopGMAC_set_rx_qptr (synopGMACdevice * *gmacdev*, u32 *Buffer1*, u32 *Length1*, u32 *Data1*, u32 *Buffer2*, u32 *Length2*, u32 *Data2*)

Prepares the descriptor to receive packets.

The descriptor is allocated with the valid buffer addresses (sk_buff address) and the length fields and handed over to DMA by setting the ownership. After successful return from this function the descriptor is added to the receive descriptor pool/queue. This api is same for both ring mode and chain mode.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***Dma-able*** buffer1 pointer.

← ***length*** of buffer1 (Max is 2048).

← ***Dma-able*** buffer2 pointer.

← ***length*** of buffer2 (Max is 2048).

← ***u32*** data indicating whether the descriptor is in ring mode or chain mode.

Returns:

returns present rx descriptor index on success. Negative value if error.

4.1.2.132 s32 synopGMAC_set_tx_qptr (synopGMACdevice * *gmacdev*, u32 *Buffer1*, u32 *Length1*, u32 *Data1*, u32 *Buffer2*, u32 *Length2*, u32 *Data2*, u32 *offload_needed*)

Populate the tx desc structure with the buffer address.

Once the driver has a packet ready to be transmitted, this function is called with the valid dma-able buffer addresses and their lengths. This function populates the descriptor and make the DMA the owner for the descriptor. This function also controls whetther Checksum offloading to be done in hardware or not. This api is same for both ring mode and chain mode.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***Dma-able*** buffer1 pointer.

← ***length*** of buffer1 (Max is 2048).

← ***virtual*** pointer for buffer1.

← ***Dma-able*** buffer2 pointer.

← ***length*** of buffer2 (Max is 2048).

← ***virtual*** pointer for buffer2.

← ***u32*** data indicating whether the descriptor is in ring mode or chain mode.

← ***u32*** indicating whether the checksum offloading in HW/SW.

Returns:

returns present tx descriptor index on success. Negative value if error.

4.1.2.133 void synopGMAC_src_addr_filter_disable (synopGMACdevice * *gmacdev*)

Disables Source address filtering.

When disabled GMAC forwards the received frames with updated SAMatch bit in RxStatus.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.134 void synopGMAC_src_addr_filter_enable (synopGMACdevice * *gmacdev*)

Enables Source address filtering.

When enabled source address filtering is performed. Only frames matching SA filtering are passed to application with SAMatch bit of RxStatus is set. GMAC drops failed frames.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

Note:

This function is overriden by synopGMAC_frame_filter_disable(synopGMACdevice *)

4.1.2.135 void synopGMAC_take_desc_ownership (DmaDesc * *desc*)

Take ownership of this Descriptor.

The function is same for both the ring mode and the chain mode DMA structures.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.136 void synopGMAC_take_desc_ownership_rx (synopGMACdevice * *gmacdev*)

Take ownership of all the rx Descriptors.

This function is called when there is fatal error in DMA transmission. When called it takes the ownership of all the rx descriptor in rx descriptor pool/queue from DMA. The function is same for both the ring mode and the chain mode DMA structures.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

Make sure to disable the transmission before calling this function, otherwise may result in racing situation.

4.1.2.137 void synopGMAC_take_desc_ownership_tx (synopGMACdevice * *gmacdev*)

Take ownership of all the rx Descriptors.

This function is called when there is fatal error in DMA transmission. When called it takes the ownership of all the tx descriptor in tx descriptor pool/queue from DMA. The function is same for both the ring mode and the chain mode DMA structures.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

Note:

Make sure to disable the transmission before calling this function, otherwise may result in racing situation.

4.1.2.138 s32 synopGMAC_TS_addend_update (synopGMACdevice * *gmacdev*, u32 *addend_value*)

Addend Register Update This function loads the contents of Time stamp addend register with the supplied 32 value.

This is reserved function when only coarse correction option is selected

Parameters:

← ***pointer*** to synopGMACdevice

← ***32*** bit addend value

Returns:

returns 0 for Success or else Failure

4.1.2.139 void synopGMAC_TS_all_frames_disable (synopGMACdevice * *gmacdev*)

Disable Time Stamp for All frames When reset the timestamp snap shot is not enabled for all frames received by the core.

Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.140 void synopGMAC_TS_all_frames_enable (synopGMACdevice * *gmacdev*)

Enable Time Stamp for All frames When set the timestamp snap shot is enabled for all frames received by the core.

Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.141 void synopGMAC_TS_binary_rollover_enable (synopGMACdevice * *gmacdev*)

Timestamp binary rollover When set the timestamp low register rolls over after 0x7FFF_FFFF value.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.142 void synopGMAC_TS_coarse_update (synopGMACdevice * *gmacdev*)

Time Stamp Update Coarse When reset the timestamp update is done using coarse method.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.143 void synopGMAC_TS_digital_rollover_enable (synopGMACdevice * *gmacdev*)

Timestamp digital rollover When set the timestamp low register rolls over after 0x3B9A_C9FF value.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.144 void synopGMAC_TS_disable (synopGMACdevice * *gmacdev*)

This function disables the timestamping.

When disabled timestamp is not added to tx and receive frames and timestamp generator is suspended.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.145 void synopGMAC_TS_enable (synopGMACdevice * *gmacdev*)

This function enables the timestamping.

This enables the timestamping for transmit and receive frames. When disabled timestamp is not added to tx and receive frames and timestamp generator is suspended.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.146 void synopGMAC_TS_event_disable (synopGMACdevice * *gmacdev*)

Disable Snapshot for Event messages.

When disabled, snapshot is taken for all messages except Announce, Management and Signaling. Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.147 void synopGMAC_TS_event_enable (synopGMACdevice * *gmacdev*)

Enable Snapshot for Event messages.

When enabled, snapshot is taken for event messages only (SYNC, Delay_Req, Pdelay_Req or Pdelay_-Resp) When disabled, snapshot is taken for all messages except Announce, Management and Signaling. Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.148 void synopGMAC_TS_fine_update (synopGMACdevice * *gmacdev*)

Time Stamp Update Fine When reset the timestamp update is done using Fine method.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.149 void synopGMAC_TS_int_disable (synopGMACdevice * *gmacdev*)

Disable the interrupt to get timestamping interrupt.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.150 void synopGMAC_TS_int_enable (synopGMACdevice * *gmacdev*)

Enable the interrupt to get timestamping interrupt.

This enables the host to get the interrupt when (1) system time is greater or equal to the target time high and low register or (2) there is a overflow in th esecond register.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.151 void synopGMAC_TS_IPV4_disable (synopGMACdevice * *gmacdev*)

Disable time stamp snapshot for IPV4 frames.

When disabled, time stamp snapshot is not taken for IPV4 frames Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.152 void synopGMAC_TS_IPV4_enable (synopGMACdevice * *gmacdev*)

Enable time stamp snapshot for IPV4 frames.

When enabled, time stamp snapshot is taken for IPV4 frames Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.153 void synopGMAC_TS_IPV6_disable (synopGMACdevice * *gmacdev*)

Disable time stamp snapshot for IPV6 frames.

When disabled, time stamp snapshot is not taken for IPV6 frames Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.154 void synopGMAC_TS_IPV6_enable (synopGMACdevice * *gmacdev*)

Enable time stamp snapshot for IPV6 frames.

When enabled, time stamp snapshot is taken for IPV6 frames Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.155 void synopGMAC_TS_load_target_timestamp (synopGMACdevice * *gmacdev*, u32 *sec_val*, u32 *sub_sec_val*)

Load the Target time stamp registers This function Loads the target time stamp registers with the values proviced.

Parameters:

← ***pointer*** to synopGMACdevice
← ***target*** Timestamp High value
← ***target*** Timestamp Low value

Returns:

returns 0 for Success or else Failure

4.1.2.156 void synopGMAC_TS_load_timestamp_higher_val (synopGMACdevice * *gmacdev*, u32 *higher_sec_val*)

Loads the time stamp higher sec value from the value supplied.

Parameters:

← ***pointer*** to synopGMACdevice
← ***16*** higher bit second register contents passed as 32 bit value

Returns:

returns void

4.1.2.157 void synopGMAC_TS_mac_addr_filt_disable (synopGMACdevice * *gmacdev*)

Disables MAC address for PTP frame filtering.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.158 void synopGMAC_TS_mac_addr_filt_enable (synopGMACdevice * *gmacdev*)

Enable MAC address for PTP frame filtering.

When enabled, uses MAC address (apart from MAC address 0) to filter the PTP frames when PTP is sent directly over Ethernet.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.159 void synopGMAC_TS_master_disable (synopGMACdevice * *gmacdev*)

Disable Snapshot for messages relevant to Master.

When disabled, snapshot is taken for messages relevant to slave node. Valid only for Ordinary clock and Boundary clock Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.160 void synopGMAC_TS_master_enable (synopGMACdevice * *gmacdev*)

Enable Snapshot for messages relevant to Master.

When enabled, snapshot is taken for messages relevant to master mode only, else snapshot is taken for messages relevant to slave node. Valid only for Ordinary clock and Boundary clock Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.161 void synopGMAC_TS_pkt_snoop_ver1 (synopGMACdevice * *gmacdev*)

Snoop PTP packet for version 2 format When set the PTP packets are snooped using the version 2 format.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.162 void synopGMAC_TS_pkt_snoop_ver2 (synopGMACdevice * *gmacdev*)

Snoop PTP packet for version 2 format When set the PTP packets are snooped using the version 2 format.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.163 void synopGMAC_TS_ptp_over_ethernet_disable (synopGMACdevice * *gmacdev*)

Disable time stamp snapshot for PTP over Ethernet frames.

When disabled, time stamp snapshot is not taken for PTP over Ethernet frames Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.164 void synopGMAC_TS_ptp_over_ethernet_enable (synopGMACdevice * *gmacdev*)

Enable time stamp snapshot for PTP over Ethernet frames.

When enabled, time stamp snapshot is taken for PTP over Ethernet frames Reserved when "Advanced Time Stamp" is not selected

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.165 void synopGMAC_TS_read_target_timestamp (synopGMACdevice * *gmacdev*, u32 * *sec_val*, u32 * *sub_sec_val*)

Reads the Target time stamp registers This function Loads the target time stamp registers with the values proviced.

Parameters:

← ***pointer*** to synopGMACdevice

← ***pointer*** to hold target Timestamp High value

← ***pointer*** to hold target Timestamp Low value

Returns:

returns 0 for Success or else Failure

4.1.2.166 void synopGMAC_TS_read_timestamp (synopGMACdevice * *gmacdev*, u16 * *higher_sec_val*, u32 * *sec_val*, u32 * *sub_sec_val*)

Reads the time stamp contents in to the respective pointers These registers are readonly.

This function returns the 48 bit time stamp assuming Version 2 timestamp with higher word is selected.

Parameters:

← ***pointer*** to synopGMACdevice

← ***pointer*** to hold 16 higher bit second register contents

← ***pointer*** to hold 32 bit second register contents

← ***pointer*** to hold 32 bit subnanosecond register contents

Returns:

returns void

Note:

Please note that since the atomic access to the timestamp registers is not possible, the contents read may be different from the actual time stamp.

4.1.2.167 void synopGMAC_TS_read_timestamp_higher_val (synopGMACdevice * *gmacdev*, u16 * *higher_sec_val*)

Reads the time stamp higher sec value to respective pointers.

Parameters:

← ***pointer*** to synopGMACdevice

← ***pointer*** to hold 16 higher bit second register contents

Returns:

returns void

4.1.2.168 void synopGMAC_TS_set_clk_type (synopGMACdevice * *gmacdev*, u32 *clk_type*)

Selet the type of clock mode for PTP.

Please note to use one of the follwoing as the clk_type argument. GmacTSOrdClk = 0x00000000, 00=> Ordinary clock GmacTSBouClk = 0x00010000, 01=> Boundary clock GmacTSEtoEClk = 0x00020000, 10=> End-to-End transparent clock GmacTSPtoPClk = 0x00030000, 11=> P-to-P transparent clock

Parameters:

← ***pointer*** to synopGMACdevice

← ***u32*** value representing one of the above clk value

Returns:

returns void

4.1.2.169 void synopGMAC_TS_subsecond_init (synopGMACdevice * *gmacdev*, u32 *sub_sec_inc_value*)

Load the Sub Second Increment value in to Sub Second increment register.

Parameters:

← ***pointer*** to synopGMACdevice

Returns:

returns void

4.1.2.170 s32 synopGMAC_TS_timestamp_init (synopGMACdevice * *gmacdev*, u32 *high_value*, u32 *low_value*)

time stamp Initialize This function Loads/Initializes h the value specified in the Timestamp High Update and Timestamp Low Update register.

Parameters:

← ***pointer*** to synopGMACdevice
← ***Timestamp*** High Load value
← ***Timestamp*** Low Load value

Returns:

returns 0 for Success or else Failure

4.1.2.171 s32 synopGMAC_TS_timestamp_update (synopGMACdevice * *gmacdev*, u32 *high_value*, u32 *low_value*)

time stamp Update This function updates (adds/subtracts) with the value specified in the Timestamp High Update and Timestamp Low Update register.

Parameters:

← ***pointer*** to synopGMACdevice
← ***Timestamp*** High Update value
← ***Timestamp*** Low Update value

Returns:

returns 0 for Success or else Failure

4.1.2.172 void synopGMAC_tx_activate_flow_control (synopGMACdevice * *gmacdev*)

Initiate Flowcontrol operation.

When Set

- In full duplex GMAC initiates pause control frame.
- In Half duplex GMAC initiates back pressure function.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.173 void synopGMAC_tx_checksum_offload_bypass (synopGMACdevice * *gmacdev*, DmaDesc * *desc*)

The check summ offload engine is bypassed in the tx path.

Checksum is not computed in the Hardware.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***Pointer*** to tx descriptor for which ointer to synopGMACdevice.

Returns:

returns void.

4.1.2.174 void synopGMAC_tx_checksum_offload_ipv4hdr (synopGMACdevice * *gmacdev*, DmaDesc * *desc*)

The check summ offload engine is enabled to do only IPV4 header checksum.

IPV4 header Checksum is computed in the Hardware.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***Pointer*** to tx descriptor for which ointer to synopGMACdevice.

Returns:

returns void.

4.1.2.175 void synopGMAC_tx_checksum_offload_tcp_pseudo (synopGMACdevice * *gmacdev*, DmaDesc * *desc*)

The check summ offload engine is enabled to do complete checksum computation.

Hardware computes the tcp ip checksum including the pseudo header checksum. Here the tcp payload checksum field should be set to 0000. Ipv4 header checksum is also inserted.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***Pointer*** to tx descriptor for which ointer to synopGMACdevice.

Returns:

returns void.

4.1.2.176 void synopGMAC_tx_checksum_offload_tcponly (synopGMACdevice * *gmacdev*, DmaDesc * *desc*)

The check summ offload engine is enabled to do TCPIP checsum assuming Pseudo header is available.

Hardware computes the tcp ip checksum assuming pseudo header checksum is computed in software. Ipv4 header checksum is also inserted.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***Pointer*** to tx descriptor for which ointer to synopGMACdevice.

Returns:

returns void.

4.1.2.177 void synopGMAC_tx_deactivate_flow_control (synopGMACdevice * *gmacdev*)

stops Flowcontrol operation.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.178 void synopGMAC_tx_desc_init_chain (DmaDesc * *desc*)

Initialize the rx descriptors for chain mode of operation.

- Status field is initialized to 0.
- EndOfRing set for the last descriptor.
- buffer1 and buffer2 set to 0.
- data1 and data2 set to 0.

Parameters:

← ***pointer*** to DmaDesc structure.
← ***whether*** end of ring

Returns:

void.

4.1.2.179 void synopGMAC_tx_desc_init_ring (DmaDesc * *desc*, bool *last_ring_desc*)

Initialize the tx descriptors for ring or chain mode operation.

- Status field is initialized to 0.
- EndOfRing set for the last descriptor.
- buffer1 and buffer2 set to 0 for ring mode of operation. (note)
- data1 and data2 set to 0. (note)

Parameters:

← ***pointer*** to DmaDesc structure.
← ***whether*** end of ring

Returns:

void.

Note:

Initialization of the buffer1, buffer2, data1,data2 and status are not done here. This only initializes whether one wants to use this descriptor in chain mode or ring mode. For chain mode of operation the buffer2 and data2 are programmed before calling this function.

4.1.2.180 void synopGMAC_tx_disable (synopGMACdevice * *gmacdev*)

Disable the transmission of frames on GMII/MII.

GMAC transmit state machine is disabled after completion of transmission of current frame.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.181 void synopGMAC_tx_enable (synopGMACdevice * *gmacdev*)

Enable the transmission of frames on GMII/MII.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.182 void synopGMAC_tx_flow_control_disable (synopGMACdevice * *gmacdev*)

Tx flow control disable.

When Disabled

- In full duplex GMAC will not transmit any pause frames.
- In Half duplex GMAC disables the back pressure feature.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.183 void synopGMAC_tx_flow_control_enable (synopGMACdevice * *gmacdev*)

Tx flow control enable.

When Enabled

- In full duplex GMAC enables flow control operation to transmit pause frames.
- In Half duplex GMAC enables the back pressure operation

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.184 void synopGMAC_unicast_hash_filter_disable (synopGMACdevice * *gmacdev*)

Disables multicast hash filtering.

When disabled GMAC performs perfect destination address filtering for unicast frames, it compares DA field with the value programmed in DA register.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.185 void synopGMAC_unicast_hash_filter_enable (synopGMACdevice * *gmacdev*)

Enables unicast hash filtering.

When enabled GMAC performs the destination address filtering of unicast frames according to the hash table.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.186 void synopGMAC_unicast_pause_frame_detect_disable (synopGMACdevice * *gmacdev*)

Disables detection of pause frames with stations unicast address.

When disabled GMAC only detects with the unique multicast address (802.3x).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.187 void synopGMAC_unicast_pause_frame_detect_enable (synopGMACdevice * *gmacdev*)

Enables detection of pause frames with stations unicast address.

When enabled GMAC detects the pause frames with stations unicast address in addition to the detection of pause frames with unique multicast address.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

void.

4.1.2.188 void synopGMAC_wakeup_frame_enable (synopGMACdevice * *gmacdev*)

Enables GMAC to look for wake up frame.

Wake up frame is defined by the user.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.189 void synopGMAC_wd_disable (synopGMACdevice * *gmacdev*)

Disable the watchdog timer on the receiver.

When disabled, Gmac disabled watchdog timer, and can receive frames up to 16,384 bytes.

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.190 void synopGMAC_wd_enable (synopGMACdevice * *gmacdev*)

Enable the watchdog timer on the receiver.

When enabled, Gmac enables Watchdog timer, and GMAC allows no more than 2048 bytes of data (10,240 if Jumbo frame enabled).

Parameters:

← ***pointer*** to synopGMACdevice.

Returns:

returns void.

4.1.2.191 void synopGMAC_write_hash_table_high (synopGMACdevice * *gmacdev*, u32 *data*)

Populates the Hash High register with the data supplied.

This function is called when the Hash filtering is to be enabled.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***data*** to be written to hash table high register.

Returns:

void.

4.1.2.192 void synopGMAC_write_hash_table_low (synopGMACdevice * *gmacdev*, u32 *data*)

Populates the Hash Low register with the data supplied.

This function is called when the Hash filtering is to be enabled.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***data*** to be written to hash table low register.

Returns:

void.

4.1.2.193 s32 synopGMAC_write_phy_reg (u32 * *RegBase*, u32 *PhyBase*, u32 *RegOffset*, u16 *data*)

Function to write to the Phy register.

The access to phy register is a slow process as the data is moved accross MDI/MDO interface

Parameters:

← ***pointer*** to Register Base (It is the mac base in our case) .

← ***PhyBase*** register is the index of one of supported 32 PHY devices.

← ***Register*** offset is the index of one of the 32 phy register.

← ***data*** to be written to the respective phy register.

Returns:

Returns 0 on success else return the error status.

4.1.2.194 void synopGMAC_write_wakeup_frame_register (synopGMACdevice * *gmacdev*, u32 * *filter_contents*)

Populates the remote wakeup frame registers.

Consecutive 8 writes to GmacWakeupAddr writes the wakeup frame filter registers. Before commensing a new write, frame filter pointer is reset to 0x0000. A small delay is introduced to allow frame filter pointer reset operation.

Parameters:

← ***pointer*** to synopGMACdevice.

← ***pointer*** to frame filter contents array.

Returns:

returns void.

4.2 synopGMAC_Host.c File Reference

The top most file which makes use of synopsys GMAC driver code.

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/delay.h>
#include <linux/interrupt.h>
#include <linux/device.h>
#include <linux/pci.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>
#include "synopGMAC_Host.h"
#include "synopGMAC_banner.h"
#include "synopGMAC_plat.h"
#include "synopGMAC_pci_bus_interface.h"
#include "synopGMAC_network_interface.h"
#include "synopGMAC_Dev.h"
```

4.2.1 Detailed Description

The top most file which makes use of synopsys GMAC driver code.

This file can be treated as the example code for writing a application driver for synopsys GMAC device using the driver provided by Synopsys. This exmple is for Linux 2.6.xx kernel

- Uses 32 bit 33MHz PCI Interface as the host bus interface
- Uses Linux network driver and the TCP/IP stack framework
- Uses the Device Specific Synopsys GMAC Kernel APIs

4.3 synopGMAC_network_interface.c File Reference

This is the network dependent layer to handle network related functionality.

```
#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/pci.h>
#include <linux/init.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include "synopGMAC_Host.h"
#include "synopGMAC_plat.h"
#include "synopGMAC_network_interface.h"
#include "synopGMAC_Dev.h"
```

Functions

- static void synopGMAC_linux_cable_unplug_function (u32 notused)

  Function used to detect the cable plugging and unplugging.

- s32 synopGMAC_setup_tx_desc_queue (synopGMACdevice *gmacdev, struct pci_dev *pcidev, u32 no_of_desc, u32 desc_mode)

  This sets up the transmit Descriptor queue in ring or chain mode.

- s32 synopGMAC_setup_rx_desc_queue (synopGMACdevice *gmacdev, struct pci_dev *pcidev, u32 no_of_desc, u32 desc_mode)

  This sets up the receive Descriptor queue in ring or chain mode.

- void synopGMAC_giveup_rx_desc_queue (synopGMACdevice *gmacdev, struct pci_dev *pcidev, u32 desc_mode)

  This gives up the receive Descriptor queue in ring or chain mode.

- void synopGMAC_giveup_tx_desc_queue (synopGMACdevice *gmacdev, struct pci_dev *pcidev, u32 desc_mode)

  This gives up the transmit Descriptor queue in ring or chain mode.

- void synop_handle_transmit_over (struct net_device *netdev)

  Function to handle housekeeping after a packet is transmitted over the wire.

- void synop_handle_received_data (struct net_device *netdev)

  Function to Receive a packet from the interface.

- irqreturn_t synopGMAC_intr_handler (s32 intr_num, void *dev_id, struct pt_regs *regs)

  Interrupt service routing.

- s32 synopGMAC_linux_open (struct net_device *netdev)

  Function used when the interface is opened for use.

- s32 synopGMAC_linux_close (struct net_device *netdev)

  Function used when the interface is closed.

- s32 synopGMAC_linux_xmit_frames (struct sk_buff *skb, struct net_device *netdev)

  Function to transmit a given packet on the wire.

- struct net_device_stats * synopGMAC_linux_get_stats (struct net_device *netdev)

  Function provides the network interface statistics.

- void synopGMAC_linux_set_multicast_list (struct net_device *netdev)

  Function to set multicast and promiscous mode.

- s32 synopGMAC_linux_set_mac_address (struct net_device *netdev, void *macaddr)

  Function to set ethernet address of the NIC.

- s32 synopGMAC_linux_change_mtu (struct net_device *netdev, s32 newmtu)

  Function to change the Maximum Transfer Unit.

- s32 synopGMAC_linux_do_ioctl (struct net_device *netdev, struct ifreq *ifr, s32 cmd)

  IOCTL interface.

- void synopGMAC_linux_tx_timeout (struct net_device *netdev)

  Function to handle a Tx Hang.

- s32 __init synopGMAC_init_network_interface (void)

  Function to initialize the Linux network interface.

- void __exit synopGMAC_exit_network_interface (void)

  Function to initialize the Linux network interface.

4.3.1 Detailed Description

This is the network dependent layer to handle network related functionality.

This file is tightly coupled to neworking frame work of linux 2.6.xx kernel. The functionality carried out in this file should be treated as an example only if the underlying operating system is not Linux.

Note:

Many of the functions other than the device specific functions changes for operating system other than Linux 2.6.xx

4.3.2 Function Documentation

4.3.2.1 void synop_handle_received_data (struct net_device * *netdev*)

Function to Receive a packet from the interface.

After Receiving a packet, DMA transfers the received packet to the system memory and generates corresponding interrupt (if it is enabled). This function prepares the sk_buff for received packet after removing the ethernet CRC, and hands it over to linux networking stack.

- Updataes the networking interface statistics
- Keeps track of the rx descriptors

Parameters:

← ***pointer*** to net_device structure.

Returns:

void.

Note:

This function runs in interrupt context.

4.3.2.2 void synop_handle_transmit_over (struct net_device * *netdev*)

Function to handle housekeeping after a packet is transmitted over the wire.

After the transmission of a packet DMA generates corresponding interrupt (if it is enabled). It takes care of returning the sk_buff to the linux kernel, updating the networking statistics and tracking the descriptors.

Parameters:

← ***pointer*** to net_device structure.

Returns:

void.

Note:

This function runs in interrupt context

4.3.2.3 void __exit synopGMAC_exit_network_interface (void)

Function to initialize the Linux network interface.

Linux dependent Network interface is setup here. This provides an example to handle the network dependent functionality.

Returns:

Returns 0 on success and Error code on failure.

4.3.2.4 void synopGMAC_giveup_rx_desc_queue (synopGMACdevice * *gmacdev*, struct pci_dev * *pcidev*, u32 *desc_mode*)

This gives up the receive Descriptor queue in ring or chain mode.

This function is tightly coupled to the platform and operating system Once device's Dma is stopped the memory descriptor memory and the buffer memory deallocation, is completely handled by the operating system, this call is kept outside the device driver Api. This function should be treated as an example code to de-allocate the descriptor structures in ring mode or chain mode and network buffer deallocation. This function depends on the pcidev structure for dma-able memory deallocation for both descriptor memory and the network buffer memory under linux. The responsibility of this function is to

- Free the network buffer memory if any.
  - Fee the memory allocated for the descriptors.

  Parameters:

  ← ***pointer*** to synopGMACdevice.
  ← ***pointer*** to pci_device structure.
  ← ***number*** of descriptor expected in rx descriptor queue.
  ← ***whether*** descriptors to be created in RING mode or CHAIN mode.

  Returns:

  0 upon success. Error code upon failure.

  Note:

  No referece should be made to descriptors once this function is called. This function is invoked when the device is closed.

4.3.2.5 void synopGMAC_giveup_tx_desc_queue (synopGMACdevice * *gmacdev*, struct pci_dev * *pcidev*, u32 *desc_mode*)

This gives up the transmit Descriptor queue in ring or chain mode.

This function is tightly coupled to the platform and operating system Once device's Dma is stopped the memory descriptor memory and the buffer memory deallocation, is completely handled by the operating system, this call is kept outside the device driver Api. This function should be treated as an example code to de-allocate the descriptor structures in ring mode or chain mode and network buffer deallocation. This function depends on the pcidev structure for dma-able memory deallocation for both descriptor memory and the network buffer memory under linux. The responsibility of this function is to

- Free the network buffer memory if any.
  - Fee the memory allocated for the descriptors.

  Parameters:

  ← ***pointer*** to synopGMACdevice.
  ← ***pointer*** to pci_device structure.
  ← ***number*** of descriptor expected in tx descriptor queue.
  ← ***whether*** descriptors to be created in RING mode or CHAIN mode.

  Returns:

  0 upon success. Error code upon failure.

  Note:

  No reference should be made to descriptors once this function is called. This function is invoked when the device is closed.

4.3.2.6 s32 __init synopGMAC_init_network_interface (void)

Function to initialize the Linux network interface.

Linux dependent Network interface is setup here. This provides an example to handle the network dependent functionality.

Returns:

Returns 0 on success and Error code on failure.

4.3.2.7 irqreturn_t synopGMAC_intr_handler (s32 *intr_num*, void * *dev_id*, struct pt_regs * *regs*)

Interrupt service routing.

This is the function registered as ISR for device interrupts.

Parameters:

← ***interrupt*** number.

← ***void*** pointer to device unique structure (Required for shared interrupts in Linux).

← ***pointer*** to pt_regs (not used).

Returns:

Returns IRQ_NONE if not device interrupts IRQ_HANDLED for device interrupts.

Note:

This function runs in interrupt context

4.3.2.8 static void synopGMAC_linux_cable_unplug_function (u32 *notused*) [static]

Function used to detect the cable plugging and unplugging.

This function gets scheduled once in every second and polls the PHY register for network cable plug/unplug. Once the connection is back the GMAC device is configured as per new Duplex mode and Speed of the connection.

Parameters:

← ***u32*** type but is not used currently.

Returns:

returns void.

Note:

This function is tightly coupled with Linux 2.6.xx.

4.3.2.9 s32 synopGMAC_linux_change_mtu (struct net_device * *netdev*, s32 *newmtu*)

Function to change the Maximum Transfer Unit.

Parameters:

← ***pointer*** to net_device structure.

← ***New*** value for maximum frame size.

Returns:

Returns 0 on success Errorcode on failure.

4.3.2.10 s32 synopGMAC_linux_close (struct net_device * *netdev*)

Function used when the interface is closed.

This function is registered to linux stop() function. This function is called whenever ifconfig (in Linux) closes the device (for example "ifconfig eth0 down"). This releases all the system resources allocated during open call. system resources int needs

- Disable the device interrupts
- Stop the receiver and get back all the rx descriptors from the DMA
- Stop the transmitter and get back all the tx descriptors from the DMA
- Stop the Linux network queue interface
- Free the irq (ISR registered is removed from the kernel)
- Release the TX and RX descripor memory
- De-initialize one second timer rgistered for cable plug/unplug tracking

Parameters:

← ***pointer*** to net_device structure.

Returns:

Returns 0 on success and error status upon failure.

4.3.2.11 s32 synopGMAC_linux_do_ioctl (struct net_device * *netdev*, struct ifreq * *ifr*, s32 *cmd*)

IOCTL interface.

This function is mainly for debugging purpose. This provides hooks for Register read write, Retrieve descriptor status and Retreiving Device structure information.

Parameters:

← ***pointer*** to net_device structure.

← ***pointer*** to ifreq structure.

← ***ioctl*** command.

Returns:

Returns 0 on success Error code on failure.

4.3.2.12 struct net_device_stats* synopGMAC_linux_get_stats (struct net_device * *netdev*) [read]

Function provides the network interface statistics.

Function is registered to linux get_stats() function. This function is called whenever ifconfig (in Linux) asks for networkig statistics (for example "ifconfig eth0").

Parameters:

← ***pointer*** to net_device structure.

Returns:

Returns pointer to net_device_stats structure.

4.3.2.13 s32 synopGMAC_linux_open (struct net_device * *netdev*)

Function used when the interface is opened for use.

We register synopGMAC_linux_open function to linux open(). Basically this function prepares the the device for operation . This function is called whenever ifconfig (in Linux) activates the device (for example "ifconfig eth0 up"). This function registers system resources needed

- Attaches device to device specific structure
- Programs the MDC clock for PHY configuration
- Check and initialize the PHY interface
- ISR registration
- Setup and initialize Tx and Rx descriptors
- Initialize MAC and DMA
- Allocate Memory for RX descriptors (The should be DMAable)
- Initialize one second timer to detect cable plug/unplug
- Configure and Enable Interrupts
- Enable Tx and Rx
- start the Linux network queue interface

Parameters:

← ***pointer*** to net_device structure.

Returns:

Returns 0 on success and error status upon failure.

4.3.2.14 s32 synopGMAC_linux_set_mac_address (struct net_device * *netdev*, void * *macaddr*)

Function to set ethernet address of the NIC.

Parameters:

← ***pointer*** to net_device structure.

← ***pointer*** to an address structure.

Returns:

Returns 0 on success Errorcode on failure.

4.3.2.15 void synopGMAC_linux_set_multicast_list (struct net_device * *netdev*)

Function to set multicast and promiscous mode.

Parameters:

← ***pointer*** to net_device structure.

Returns:

returns void.

4.3.2.16 void synopGMAC_linux_tx_timeout (struct net_device * *netdev*)

Function to handle a Tx Hang.

This is a software hook (Linux) to handle transmitter hang if any. We get transmitter hang in the device interrupt status, and is handled in ISR. This function is here as a place holder.

Parameters:

← ***pointer*** to net_device structure

Returns:

void.

4.3.2.17 s32 synopGMAC_linux_xmit_frames (struct sk_buff * *skb*, struct net_device * *netdev*)

Function to transmit a given packet on the wire.

Whenever Linux Kernel has a packet ready to be transmitted, this function is called. The function prepares a packet and prepares the descriptor and enables/resumes the transmission.

Parameters:

← ***pointer*** to sk_buff structure.

← ***pointer*** to net_device structure.

Returns:

Returns 0 on success and Error code on failure.

Note:

structure sk_buff is used to hold packet in Linux networking stacks.

4.3.2.18 s32 synopGMAC_setup_rx_desc_queue (synopGMACdevice * *gmacdev*, struct pci_dev * *pcidev*, u32 *no_of_desc*, u32 *desc_mode*)

This sets up the receive Descriptor queue in ring or chain mode.

This function is tightly coupled to the platform and operating system Device is interested only after the descriptors are setup. Therefore this function is not included in the device driver API. This function should be treated as an example code to design the descriptor structures in ring mode or chain mode. This function depends on the pcidev structure for allocation of consistent dma-able memory in case of linux. This limitation is due to the fact that linux uses pci structure to allocate a dmable memory

- Allocates the memory for the descriptors.
- Initialize the Busy and Next descriptors indices to 0(Indicating first descriptor).
- Initialize the Busy and Next descriptors to first descriptor address.
- Initialize the last descriptor with the endof ring in case of ring mode.
- Initialize the descriptors in chain mode.

Parameters:

← ***pointer*** to synopGMACdevice.
← ***pointer*** to pci_device structure.
← ***number*** of descriptor expected in rx descriptor queue.
← ***whether*** descriptors to be created in RING mode or CHAIN mode.

Returns:

0 upon success. Error code upon failure.

Note:

This function fails if allocation fails for required number of descriptors in Ring mode, but in chain mode function returns -ESYNOPGMACNOMEM in the process of descriptor chain creation. once returned from this function user should for gmacdev->RxDescCount to see how many descriptors are there in the chain. Should continue further only if the number of descriptors in the chain meets the requirements

4.3.2.19 s32 synopGMAC_setup_tx_desc_queue (synopGMACdevice * *gmacdev*, struct pci_dev * *pcidev*, u32 *no_of_desc*, u32 *desc_mode*)

This sets up the transmit Descriptor queue in ring or chain mode.

This function is tightly coupled to the platform and operating system Device is interested only after the descriptors are setup. Therefore this function is not included in the device driver API. This function should be treated as an example code to design the descriptor structures for ring mode or chain mode. This function depends on the pcidev structure for allocation consistent dma-able memory in case of linux. This limitation is due to the fact that linux uses pci structure to allocate a dmable memory

- Allocates the memory for the descriptors.
- Initialize the Busy and Next descriptors indices to 0(Indicating first descriptor).
- Initialize the Busy and Next descriptors to first descriptor address.
- Initialize the last descriptor with the endof ring in case of ring mode.
- Initialize the descriptors in chain mode.

Parameters:

← ***pointer*** to synopGMACdevice.
← ***pointer*** to pci_device structure.
← ***number*** of descriptor expected in tx descriptor queue.
← ***whether*** descriptors to be created in RING mode or CHAIN mode.

Returns:

0 upon success. Error code upon failure.

Note:

This function fails if allocation fails for required number of descriptors in Ring mode, but in chain mode function returns -ESYNOPGMACNOMEM in the process of descriptor chain creation. once returned from this function user should for gmacdev->TxDescCount to see how many descriptors are there in the chain. Should continue further only if the number of descriptors in the chain meets the requirements

4.4 synopGMAC_pci_bus_interface.c File Reference

This file encapsulates all the PCI dependent initialization and resource allocation on Linux.

```
#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/pci.h>
#include <linux/init.h>
#include <linux/dma-mapping.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>
#include "synopGMAC_plat.h"
#include "synopGMAC_pci_bus_interface.h"
```

Functions

- static int probe (struct pci_dev *pdev, const struct pci_device_id *my_pci_id)
  probe function of Linux pci driver.
- static void remove (struct pci_dev *dev)
  remove function of Linux pci driver.
- s32 __init synopGMAC_init_pci_bus_interface (void)
  Function to initialize the Linux Pci Bus Interface.
- void __exit synopGMAC_exit_pci_bus_interface (void)
  Function to De-initialize the Linux Pci Bus Interface.

4.4.1 Detailed Description

This file encapsulates all the PCI dependent initialization and resource allocation on Linux.

4.4.2 Function Documentation

4.4.2.1 static int probe (struct pci_dev * *pdev*, const struct pci_device_id * *my_pci_id*) [static]

probe function of Linux pci driver.

- Ioremap the BARx memory (It is BAR0 here)
- lock the memory for the device

Returns:

Returns 0 on success and Error code on failure.

4.4.2.2 static void remove (struct pci_dev * *dev*) [static]

remove function of Linux pci driver.

- Releases the memory allocated by probe function
- Unmaps the memory region

Returns:

Returns 0 on success and Error code on failure.

4.4.2.3 void __exit synopGMAC_exit_pci_bus_interface (void)

Function to De-initialize the Linux Pci Bus Interface.

Unregisters the pci_driver

Returns:

Returns 0 on success and Error code on failure.

4.4.2.4 s32 __init synopGMAC_init_pci_bus_interface (void)

Function to initialize the Linux Pci Bus Interface.

Registers the pci_driver

Returns:

Returns 0 on success and Error code on failure.

4.5 synopGMAC_plat.c File Reference

This file defines the wrapper for the platform/OS related functions The function definitions needs to be modified according to the platform and the Operating system used.

```
#include "synopGMAC_plat.h"
```

Functions

- void * plat_alloc_memory (u32 bytes)

  This is a wrapper function for Memory allocation routine.

- void * plat_alloc_consistent_dmaable_memory (struct pci_dev *pcidev, u32 size, u32 *addr)

  This is a wrapper function for consistent dma-able Memory allocation routine.

- void plat_free_consistent_dmaable_memory (struct pci_dev *pcidev, u32 size, void *addr, u32 dma_addr)

  This is a wrapper function for freeing consistent dma-able Memory.

- void plat_free_memory (void *buffer)

  This is a wrapper function for Memory free routine.

- void plat_delay (u32 delay)

  This is a wrapper function for platform dependent delay Take care while passing the argument to this function.

4.5.1 Detailed Description

This file defines the wrapper for the platform/OS related functions The function definitions needs to be modified according to the platform and the Operating system used.

This file should be handled with greatest care while porting the driver to a different platform running different operating system other than Linux 2.6.xx.

4.5.2 Function Documentation

4.5.2.1 void* plat_alloc_consistent_dmaable_memory (struct pci_dev * *pcidev*, u32 *size*, u32 * *addr*)

This is a wrapper function for consistent dma-able Memory allocation routine.

In linux Kernel, it depends on pci dev structure

Parameters:

← ***bytes*** in bytes to allocate

4.5.2.2 void* plat_alloc_memory (u32 *bytes*)

This is a wrapper function for Memory allocation routine.

These are the wrapper function prototypes for OS/platform related routines.

In linux Kernel it it kmalloc function

Parameters:

← ***bytes*** in bytes to allocate

4.5.2.3 void plat_delay (u32 *delay*)

This is a wrapper function for platform dependent delay Take care while passing the argument to this function.

Parameters:

← ***buffer*** pointer to be freed

4.5.2.4 void plat_free_consistent_dmaable_memory (struct pci_dev * *pcidev*, u32 *size*, void * *addr*, u32 *dma_addr*)

This is a wrapper function for freeing consistent dma-able Memory.

In linux Kernel, it depends on pci dev structure

Parameters:

← ***bytes*** in bytes to allocate

4.5.2.5 void plat_free_memory (void * *buffer*)

This is a wrapper function for Memory free routine.

In linux Kernel it it kfree function

Parameters:

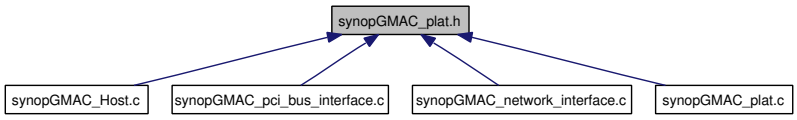
← ***buffer*** pointer to be freed

4.6 synopGMAC_plat.h File Reference

This file serves as the wrapper for the platform/OS dependent functions It is needed to modify these functions accordingly based on the platform and the OS.

```
#include <linux/kernel.h>
#include <asm/io.h>
#include <linux/gfp.h>
#include <linux/slab.h>
#include <linux/pci.h>
```

This graph shows which files directly or indirectly include this file:



Functions

- void * plat_alloc_memory (u32)

  These are the wrapper function prototypes for OS/platform related routines.

- void plat_free_memory (void *)

  This is a wrapper function for Memory free routine.

- void * plat_alloc_consistent_dmaable_memory (struct pci_dev *, u32, u32 *)

  This is a wrapper function for consistent dma-able Memory allocation routine.

- void plat_free_consistent_dmaable_memory (struct pci_dev *, u32, void *, u32)

  This is a wrapper function for freeing consistent dma-able Memory.

- void plat_delay (u32)

  This is a wrapper function for platform dependent delay Take care while passing the argument to this function.

- static u32 __inline__ synopGMACReadReg (u32 *RegBase, u32 RegOffset)

  The Low level function to read register contents from Hardware.

- static void __inline__ synopGMACWriteReg (u32 *RegBase, u32 RegOffset, u32 RegData)

  The Low level function to write to a register in Hardware.

- static void __inline__ synopGMACSetBits (u32 *RegBase, u32 RegOffset, u32 BitPos)

  The Low level function to set bits of a register in Hardware.

- static void __inline__ synopGMACClearBits (u32 *RegBase, u32 RegOffset, u32 BitPos)

  The Low level function to clear bits of a register in Hardware.

- static bool __inline__ synopGMACCheckBits (u32 *RegBase, u32 RegOffset, u32 BitPos)

The Low level function to Check the setting of the bits.

4.6.1 Detailed Description

This file serves as the wrapper for the platform/OS dependent functions It is needed to modify these functions accordingly based on the platform and the OS.

Whenever the synopsys GMAC driver ported on to different platform, this file should be handled at most care. The corresponding function definitions for non-inline functions are available in synopGMAC_plat.c file.

4.6.2 Function Documentation

4.6.2.1 void* plat_alloc_consistent_dmaable_memory (struct pci_dev * *pcidev*, u32 *size*, u32 * *addr*)

This is a wrapper function for consistent dma-able Memory allocation routine.

In linux Kernel, it depends on pci dev structure

Parameters:

← ***bytes*** in bytes to allocate

4.6.2.2 void* plat_alloc_memory (u32 *bytes*)

These are the wrapper function prototypes for OS/platform related routines.

These are the wrapper function prototypes for OS/platform related routines.

In linux Kernel it it kmalloc function

Parameters:

← ***bytes*** in bytes to allocate

4.6.2.3 void plat_delay (u32 *delay*)

This is a wrapper function for platform dependent delay Take care while passing the argument to this function.

Parameters:

← ***buffer*** pointer to be freed

4.6.2.4 void plat_free_consistent_dmaable_memory (struct pci_dev * *pcidev*, u32 *size*, void * *addr*, u32 *dma_addr*)

This is a wrapper function for freeing consistent dma-able Memory.

In linux Kernel, it depends on pci dev structure

Parameters:

← ***bytes*** in bytes to allocate

4.6.2.5 void plat_free_memory (void * *buffer*)

This is a wrapper function for Memory free routine.

In linux Kernel it it kfree function

Parameters:

← ***buffer*** pointer to be freed

4.6.2.6 static bool __inline__ synopGMACCheckBits (u32 * *RegBase*, u32 *RegOffset*, u32 *BitPos*) [static]

The Low level function to Check the setting of the bits.

Parameters:

← ***pointer*** to the base of register map

← ***Offset*** from the base

← ***Bit*** mask to set bits to logical 1

Returns:

returns TRUE if set to '1' returns FALSE if set to '0'. Result undefined there are no bit set in the BitPos argument.

4.6.2.7 static void __inline__ synopGMACClearBits (u32 * *RegBase*, u32 *RegOffset*, u32 *BitPos*) [static]

The Low level function to clear bits of a register in Hardware.

Parameters:

← ***pointer*** to the base of register map

← ***Offset*** from the base

← ***Bit*** mask to clear bits to logical 0

Returns:

void

4.6.2.8 static u32 __inline__ synopGMACReadReg (u32 * *RegBase*, u32 *RegOffset*) [static]

The Low level function to read register contents from Hardware.

Parameters:

← ***pointer*** to the base of register map

← ***Offset*** from the base

Returns:

Returns the register contents

4.6.2.9 static void __inline__ synopGMACSetBits (u32 * *RegBase*, u32 *RegOffset*, u32 *BitPos*) [static]

The Low level function to set bits of a register in Hardware.

Parameters:

← ***pointer*** to the base of register map

← ***Offset*** from the base

← ***Bit*** mask to set bits to logical 1

Returns:

void

4.6.2.10 static void __inline__ synopGMACWriteReg (u32 * *RegBase*, u32 *RegOffset*, u32 *RegData*) [static]

The Low level function to write to a register in Hardware.

Parameters:

← ***pointer*** to the base of register map

← ***Offset*** from the base

← ***Data*** to be written

Returns:

void

Chapter 5

Device Driver Page Documentation

5.1 Device Driver Porting Guide

5.1.1 Purpose of this Chapter

Synopsys provided GMAC driver software supplied in the official release is for Linux2.6.xx kernel. Sample driver assumes bus interface used as PCI. This limitation is due to the fact that development platform was an Intel based desktop system and theses systems does not provide AHB/APB interface as a generic SOC based systems do.

Since the driver developed is highly modular, porting the same on to different platform is relatively simple. This document makes an effort to explain how the driver can be ported on to different architecture running Linux or a different operating system. Even though the document refers the to an ARM based host interfaced to GMAC using AHB bus, conceptually it is applicable to any platform of interest.

Below are the device driver source and header files.

```
synopGMAC_banner.h
synopGMAC_plat.c
synopGMAC_plat.h
synopGMAC_Dev.c
synopGMAC_Dev.h
synopGMAC_pci_bus_interface.c
synopGMAC_pci_bus_interface.h
synopGMAC_network_interface.c
synopGMAC_network_interface.h
synopGMAC_Host.c
synopGMAC_Host.h
```

The corresponding make file to compile the kernel source in to loadable module under Linux is

```
Makefile
```

The Debug utility to test some of the features of GMAC ip

```
synopGMAC_Debug.c
```

Subsequent sections explain the porting requirement of particular files.

5.1.2 Porting synopGMAC_banner.h

This file contains the device driver software version and the copyright information. While porting on to a different processor platform, this file need not be changed and can be ported as is.

5.1.3 Porting synopGMAC_plat.c

This file serves as a wrapper for the Processor/Operating system dependent layer. This C source file is accompanied by the corresponding header file synopGMAC_plat.h.

The header file contains the low level GMAC hardware access functions. Generally hardware access layer is coded as Macro or inline function to circumvent the compiler optimization issues and it is likely that these functions get tightly coupled with the platform and the OS they are running on. This explains why this functionality is wrapped in platform dependent header file. The file is likely to remain as is if under-lying operating system is Linux2.6.xx, otherwise the file may need modification based on the infrastructure provided by the chosen operating system. Even minor changes required in the device access function depending on how hardware is accessed(Note).

The source file synopGMAC_plat.c is abstracting the operating system dependent calls such as memory allocation and de-allocation, platform dependent delay routines etc. Again this file likely to remain as is if under-lying operating system is Linux2.6.xx.

Note:

Note that the CSR (Control and Status Register) space is assumed to be visible as a memory mapped device. If it is mapped to an IO space, modification required in the way the CSR registers is accessed.

5.1.4 Porting synopGMAC_Dev.c

This file is the Hardware Abstraction Layer (HAL) of the GMAC device driver. All the device dependent functions (Device driver Kernel APIs (Note)) are coded here. Corresponding driver APIs are prototyped in synopGMAC_Dev.h header file.

These file likely to remain as is in case either/both platform and OS changes. Only caution is to make sure that basic data types u8/u16/u32 are correct. Refer synopGMAC_plat.h for the definitions of these data types.

Note:

Notion of Kernel API is due to the fact that corresponding functions run in Kernel space in Linux. Generically these can be called driver APIs.

5.1.5 Porting synopGMAC_pci_bus_interface.c

As mentioned earlier the default sample driver code provided by Synopsys uses PCI as the bus interface. The bus independent layer seeks only following information from this PCI dependent layer. Basically this layer

1. Allocates memory for the device CSR space (this is what is referred to as the Base address)
2. Obtains the irq number the device is connected to.

There are two extern variables given below to facilitate the memory base address/size holders. Along with that the interrupt line is embedded in the pci device structure which is again declared as an external variable.

```
extern u8 *synopGMACMappedAddr;
extern u32 synopGMACMappedAddrSize;
extern struct pci_dev *synopGMACpcidev;
```

The Device dependent layer is only concerned with the first two variables and the irq number in pci_-dev structure. On a PCI based system these are populated by pci probe function synopGMAC_pci_bus_-interface.c. For non PCI system, these have to be populated accordingly.

For example: In case of ARM system with GMAC interfaced to AHB, the base address and the size should be populated based on the system specifications. Generally they are hard coded to a value defined in one of the header files. Please see the next section to understand how irq is handled to register the device driver ISR with the kernel.

5.1.6 Porting synopGMAC_network_interface.c

This file is accompanied by the corresponding header file synopGMAC_network_interface.h. These files heavily depend on the network interface/support provided by the Operating system in question. In the sample driver provided, the operating system used was Linux2.6.xx (Linux2.6.11 Fedora distribution to be precise). Therefore this network interface dependent layer makes use of Linux provided network interface to register the driver functions. While porting on to different architecture such as ARM, no changes are required in the files only if the operating system used remains Linux and bus interface being PCI.

Following changes are required if the bus interface is not PCI

- Check for the following externs in synopGMAC_network_interface.c

```
extern u8*                              synopGMACMappedAddr;
extern u32                              synopGMACMappedAddrSize;
extern struct pci_dev *         synopGMACpcidev;
```

These are used to communicate about the base address and its size along with the interrupt line number as indicated in the previous section.

- The variable synop_pci_using_dac was used for 64 bit setup PCI interface

```
extern u32 synop_pci_using_dac;
```

  This variable is set to 0 now;

- Interrupt Service Routine registration

```
if(request_irq (pcidev->irq, synopGMAC_intr_handler, SA_SHIRQ | SA_INTERRUPT, netdev->name,netdev))
```

Since PCI subsystem automatically identifies the irq line and populates in the pci_dev structure request_irq is called with the first argument as pcidev->irq. SA_INTERRUPT used to indicate the handler is a fast handler and SA_SHIRQ to tell the kernel that this is a shared handler.

For subsystems not using the PCI, this function should be properly passed with the valid arguments.

- Pci device structure passed to tx and rx descriptors.

```
synopGMAC_setup_tx_desc_queue(gmacdev,pcidev,TRANSMIT_DESC_SIZE, MODE);
synopGMAC_giveup_tx_desc_queue(gmacdev,pcidev, MODE);
synopGMAC_setup_rx_desc_queue(gmacdev,pcidev,RECEIVE_DESC_SIZE, MODE);
synopGMAC_giveup_rx_desc_queue(gmacdev,pcidev, MODE);
```

This is due to the fact that in Linux, when using PCI bus, the dma-able memory allocation requires the pci_dev structure as the argument. This should be properly taken care for non pci systems.

All remaining functions likely to remains same.

Note:

But if ported on to different OS, such as WindowsNT/Wince, these files demand detailed inspection and modification.

5.1.7 Porting synopGMAC_Host.c

This serves as the top most file of synopGMAC device driver software along with synopGMAC_Host.h header file. These files are responsible for initializing bus interface subsystem (PCI in Synopsys default driver) and the network interface subsystem. The call to routines for bus interface subsystem and the networking subsystem should be properly modified for system other than PCI and Linux networking subsystems.

5.1.8 Porting Makefile

The Makefile provided along with device driver source is written to build loadable kernel module in Linux. This file requires changes if any of the files are removed or new files introduced. If ported on to different operating system, make procedure may change as demanded by the operating system used.

5.1.9 synopGMAC_Debug.c

This file is not part of the device driver. This is a helper utility to provide direct interface to the hardware under Linux. The file communicates with the hardware using Linux kernel provided IOCTL interface. The compilation is done in user land rather in kernel land. The porting of this file to different operating system depends on the IOCTL and the socket interface provided by the chosen operating system.

5.1.10 References

[1] GMAC-UNIV data book gmac_ahb_databook.pdf

Index