UM0248
User manual

STR91xFA MAC/DMA controller (ENET) firmware library

Introduction

This user manual describes the driver functions developed for the STR91xFA MAC/DMA controller (ENET).

The document also provides an example of using the driver with the uIP free TCP/IP stack showing a webserver application.

Notes:

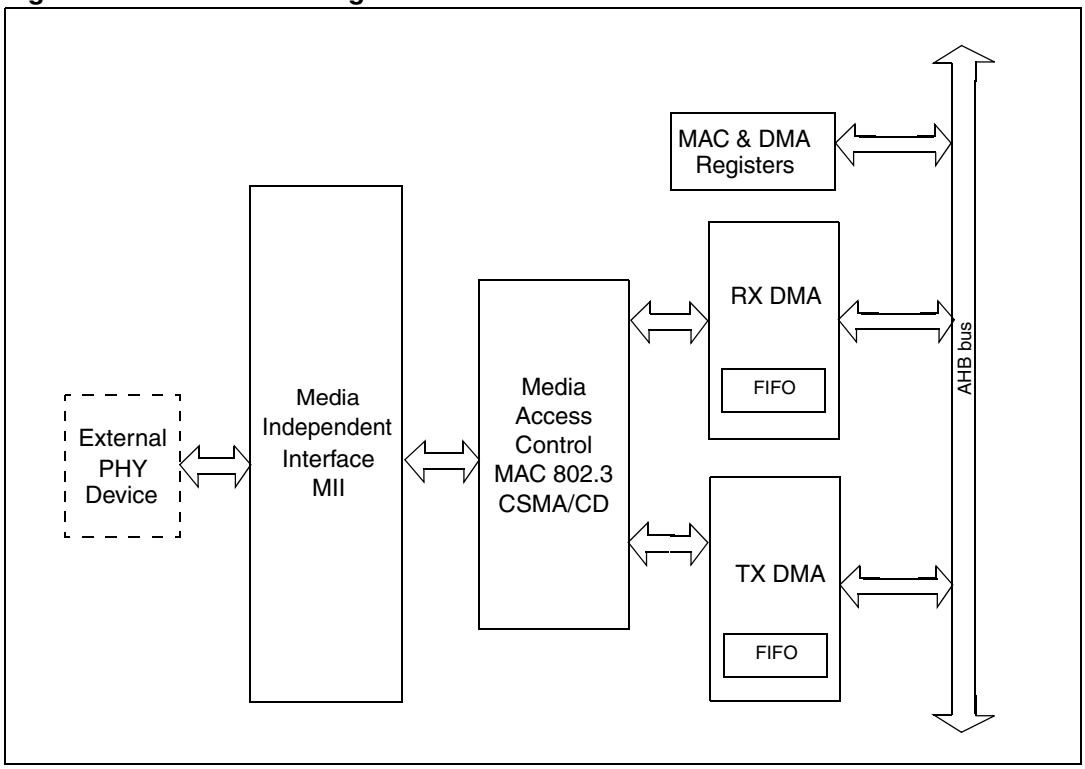
- For details on the STR91xFA MAC/DMA controller hardware, please refer to the STR91xFA reference manual (RM0006).
- Details about the uIP stack are available at: www.sics.se/~adam/uip/

Contents

1 ENET overview

1.1 ENET block diagram

Figure 1. ENET block diagram



The ENET is composed of three main parts which are:

- MAC 802.3: Media Access Control (IEEE 802.3 standard): assures the packets transmission and reception.
- DMA: DMA Rx/Tx: handles packet transfers between main memory and the FIFOs.
- MII: Media independent interface: interface between the MAC and the external PHY device.

1.2 ENET registers

The ENET registers are divided into two main parts: registers for MAC configuration and registers for DMA operation and configuration.

The following tables summarize these registers:

Table 1. ENET MAC registers

Register name	Description	Register main function
ENET_MCR	MAC Control register	MAC Control
ENET_MAH	MAC Address High register	MAC address
ENET_MAL	MAC Address Low register	
ENET_MCHA	Multicast Address High register	MAC Multicast address or Hash table value
ENET_MCLA	Multicast Address Low register	
ENET_MIIA	MII Address register	PHY control: Read/Write PHY registers
ENET_MIID	MII Data register	
ENET_MCF	MII Control Frame register	Control Frame (Pause Command)
ENET_VL1	VLAN1 register (TAG1 & ID1)	VLAN TAG & ID
ENET_VL2	VLAN2 register (TAG2 & ID2)	
ENET_MTS	MAC Transmission Status register	MAC Rx & Tx Status Flags
ENET_MRS	MAC Receive Status register	

Table 2. ENET DMA registers

Register name	Description	Register main function
ENET_SCR	DMA Control & status register	DMA general Control & Status
ENET_IER	DMA Interrupt Enable register	
ENET_ISR	DMA Interrupt Status register	
ENET_RXSTR	DMA Rx Start register	DMA Rx Control & Status
ENET_RXCR	DMA Rx Control register	
ENET_RXSAR	DMA Rx Start Address register	
ENET_RXNDAR	DMA Rx Next Address register	
ENET_RXCAR	DMA Rx Current Address register	
ENET_RXCTCR	DMA Rx Current Transfer Count register	
ENET_RXTOR	DMA Rx Timeout register	
ENET_RXSR	DMA Rx Status register	

Table 2. ENET DMA registers (continued)

Register name	Description	Register main function
ENET_TXSTR	DMA Tx Start register	DMA Tx Control & Status
ENET_TXCR	DMA Tx Control register	
ENET_TXSAR	DMA Tx Start Address register	
ENET_TXNDAR	DMA Tx Next Address register	
ENET_TXCAR	DMA Tx Current Address register	
ENET_TXCTCR	DMA Tx Current Transfer Count register	
ENET_TXTOR	DMA Tx Timeout register	
ENET_TXSR	DMA Tx Status register	
RX_FIFO[0:31]	Rx FIFO registers	DMA FIFOs
Tx_FIFO[0:31]	Tx FIFO registers	

2 ENET firmware library

2.1 Package description

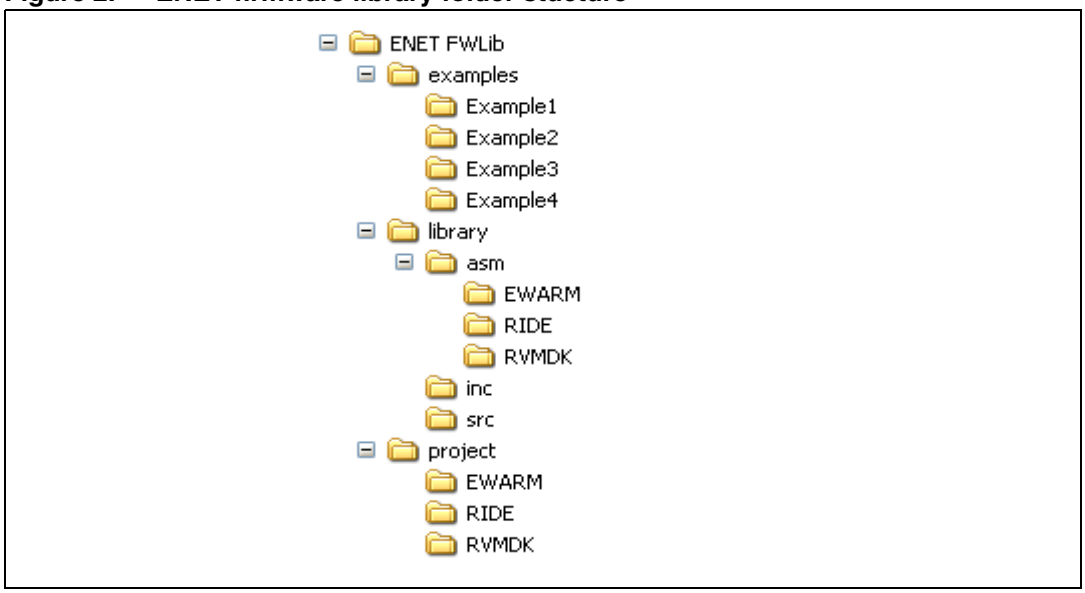
The firmware library is supplied in one compressed file. The extraction of the compressed file generates three folders:

- ENET FWlib: STR91x Ethernet firmware library
- FWlib: STR91x firmware library
- DEMOS: Webserver demo

2.1.1 ENET FWlib folder

The ENET Firmware library contains the following sub-folders:

Figure 2. ENET firmware library folder stucture



Library folder

The ENET library is composed of three folders:

- The src folder wich contains the 91x_enet.c file: ENET driver functions
- The inc folder wich contains the 91x_enet.h file: ENET driver defines and functions prototypes
- The asm folder wich contains sub-folders providing an additional file “memcopy.s” for each toolchain (EWARM, RIDE and RVMDK) wich provides an optimized memory copy function that can be used by the driver to copy received data in user memory buffer.

For more details about the memcopy functions, please refer to application note AN2367: “Optimized memory copy routine for TCP/IP on the STR91x”

Note: The ENET library uses the following register structures defined in the 91x_map.h standard library file:

- ENET_MAC_TypeDef: structure for MAC register definitions.

- ENET_DMA_TypeDef: structure for DMA register definitions.

Examples folder

The Examples folder contains four subdirectories. Each one of them contains the files needed to run an example with a readme.txt file explaining how to use it.

Project folder

The project folder is the location where the files must be copied to. It contains three folders, EWARM, RVDMK and RIDE used by the toolchain.

2.1.2 FWlib folder

The FWlib folder contains the STR91xFA firmware library.

2.1.3 DEMOS folder

The DEMOS folder contains the subdirectories and files needed to run the webserver demo.

2.2 ENET driver functions

The following table lists the ENET driver functions present in the 91x_enet.c file:

Table 3. ENET functions

Function name	Description
ENET_Init	MAC, DMA and PHY initializations
ENET_SetOperatingMode	Sets the operating mode (full/half duplex, 10/100Mbs)
ENET_Start	Starts ENET reception
ENET_UpdateRxDscr	Give the Rx buffer back to ENET DMA
ENET_HandleRxPkt	Checks if packet has been received and copies it to user buffer memory
ENET_UpdateTxDescr	Sends packet from the ENET DMA buffer memory
ENET_HandleTxPkt	Transmits packet from user buffer memory
ENET_GetRxStatus	Gets the Rx status flags
ENET_GetTxStatus	Gets the Tx status flags
ENET_MIIWriteReg	Writes in a PHY device register
ENET_MIIReadReg	Reads a PHY device register
ENET_ITConfig	Enables or disables the specified ENET DMA interrupt
ENET_GetITSrc	Specifies the source of the ENET DMA interrupt
ENET_ClearIT	Clears the ENET DMA's interrupt pending bit
ENET_PHYITConfig	Enables or disables the specified PHY interrupt
ENET_PHYGetITSrc	Specifies the source of the PHY interrupt
ENET_PHYIsolate	Isolates the PHY device from MII
ENET_PHYPowerdown	Puts the PHY device in the power-down mode
ENET_PHYLoopBack	Enables or disables the PHY loopback mode
ENET_PHYReset	PHY software reset

The following sections provide detailed descriptions of each function:

2.2.1 ENET_Init

Function name	ENET_Init
Function prototype	`void ENET_Init(ENET_MACConfig *MAC_Config)`
Behavior description	Does the following initializations: - ENET DMA initialization - ENET MAC initialization (with MAC_Config structure values) - MAC address initialization (the MAC address is defined in the 91x_enet.h file: “#define MAC_ADDRx YY”) - Multicast address initialization (the multicast address is defined in the 91x_enet.h file: “#define MCAST_ADDRx YY”) - intializes the Tx and the Rx descriptors - initializes the PHY device
Input parameter	Pointer to structure of type ENET_MACCOnfig (see below)
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIWriteReg ENET_RxDscrInit ENET_Tx_DscrInit

ENET_MACConfig structure

The ENET_MACConfig structure is defined in the *91x_enet.h* file:

```
typedef struct
{
  FunctionalState ReceiveALL;
  u32 MIIPrescaler;
  FunctionalState LoopbackMode;
  u32 AddressFilteringMode;
  u32 VLANFilterMode;
  FunctionalState PassWrongFrame;
  FunctionalState LateCollision;
  FunctionalState BroadcastFrameReception;
  FunctionalState PacketRetry;
  FunctionalState RxFrameFiltering;
  FunctionalState AutomaticPadRemoval;
  FunctionalState DeferralCheck;
} ENET_MACConfig;
```

ReceiveALL

Specifies the ENET MAC reception mode of incoming frames. This member can be one of the following values:

ReceiveALL	Meaning
ENABLE	ENET MAC receives all the valid incoming frames regardless of the destination address
DISABLE	The incoming frames are received only if the address matches with the programmed filtering rules

MIIPrescaler

Specifies the HCLK divider used to generate the MII_MDC clock. This member can be one of the following values:

MIIPrescaler	Meaning
MIIPrescaler_1	MIIPrescaler = 0, when HCLK < 50MHz
MIIPrescaler_2	MIIPrescaler = 1, when HCLK >= 50MHz

LoopbackMode

Selects the MAC normal or loopback operating mode. This member can be set to ENABLE or DISABLE.

AddressFilteringMode

Specifies the address filtering mode rules applied to the received frames. This member can be one of the following values:

AddressFilteringMode	Meaning
MAC_Perfect_Multicast	Perfect address filtering for MAC physical address and Multicast address
MAC_Perfect_Muticast_Hash	Perfect address filtering for MAC physical address and Hash filtering for Multicast address
MAC_Hash_Multicast_Hash	Hash filtering for physical and Muticast address
Inverse	The frame is accepted when MAC pysical address and mutlicast address failed
Promiscuous	No address filtering, accept all frames
MAC_Hash_Muticast_All	Hash filtering for MAC physical address and accept all muticast addresses

VLANFilteringMode

Specifies the VLAN filtering mode. This member can be one of the following values:

VLANFilteringMode	Meaning
VLANFilter_VLTAG_VLID	VLAN filtering for TAG and ID
VLANfilter_VLTAG	VLAN filtering for TAG only

PassWrongFrame

Selects when received if the wrong frames will be passed. This member can be set to ENABLE or DISABLE.

LateCollision

Selects if the frame will be retransmitted when a late collision occurs. This member can be set to ENABLE or DISABLE.

BroadcastFrameReception

Selects if the broadcast frames will be received. This member can be set to ENABLE or DISABLE.

PacketRetry

Selects if the frame will be retransmitted when a normal collision occurs. This member can be set to ENABLE or DISABLE.

RxFrameFiltering

Selects if filtering fail and early runt frames will be filtering by the VCI Rx. This member can be set to ENABLE or DISABLE.

AutomaticPadRemoval

Selects if the PAd and CRC fields will be automatically removed. This member can be set to ENABLE or DISABLE.

DeferralCheck

Selects the status of the deferral check. This member can be set to ENABLE or DISABLE.

2.2.2 ENET_SetOperatingMode

Function name	ENET_SetOperatingMode
Function prototype	`void ENET_SetOperatingMode(u32 ENET_OperatingMode)`
Behavior description	Sets the operating mode (Speed and Duplex)
Input parameter	ENET_OperationMode: Refer to *Table 5: ENET_IT parameter values* the allowed values of this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIReadReg ENET_MIIWriteReg

Table 4. ENET_OperatingMode paramter values

Value	Meaning
AUTO_NEGOTIATION	Auto-Negotiation
FULLDUPLEX_100M	Full duplex 100Mb/s
HALFDUPLEX_100M	Half duplex 100Mb/s
FULLDUPLEX_10M	Full duplex 10Mb/s
HALFDUPLEX_10M	Half duplex 10Mb/s

Example:

```
/* Configuration to operate at 100Mb/s in Full Duplex mode */
ENET_SetOperatingMode(FULLDUPLEX_100M);

/* Use the Auto-Negotiation process to set the operationg mode */
ENET_SetOperatingMode(AUTO_NEGOTIATION);
```

2.2.3 ENET_MIIWriteReg

Function name	ENET_MIIWriteReg
Function prototype	`void ENET_MIIWriteReg(u8 phyDev, u8 phyReg, u32 phyVal)`
Behavior description	Writes a value in PHY register
Input parameter 1	phyDev: PHY register address
Input parameter 2	phyReg: register address
Input parameter 3	phyVal: register value to write
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

2.2.4 ENET_MIIReadReg

Function name	ENET_MIIReadReg
Function prototype	`u32 ENET_MIIReadReg(u8 phyDev, u8 phyReg)`
Behavior description	Reads a value from PHY register
Input parameter 1	phyDev: PHY register address
Input parameter 2	phyReg: register address
Output parameter	None
Return parameter	phyReg value
Required preconditions	None
Called functions	None

2.2.5 ENET_RxDscrInit

Function name	ENET_RxDscrInit
Function prototype	`void ENET_RxDscrInit(void)`
Behavior description	Initializes ENET DMA Rx descriptors in memory
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The ENET driver provides the capability to work with single memory buffer or multiple memory buffers for frame reception. This allows selecting the appropriate configuration for each type of application. The number of buffers is defined in the 91x_enet.h file:

```
#define ENET_RXBUFNB
```

Example:

```
/* Define 8 Rx buffers */
#define ENET_RXBUFNB8
```

2.2.6 ENET_TxDscrInit

Function name	ENET_TxDscrInit
Function prototype	`void ENET_TxDscrInit(void)`
Behavior description	Initializes ENET DMA Tx descriptors in memory
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

The ENET driver provides the capability to work with single memory buffer or multiple memory buffers for frame transmission. This allows selecting the appropriate configuration for each type of application. The number of buffers is defined in the 91x_enet.h file:

```
#define ENET_TXBUFNB
```

Example:

```
/* Define 2 Tx buffers */
#define ENET_TXBUFNB2
```

2.2.7 ENET_Start

Function name	ENET_Start
Function prototype	`void ENET_Start(void)`
Behavior description	Start MAC receive and transmission and DMA Rx descriptor fetch
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

2.2.8 ENET_GetRxStatus

Function name	ENET_GetRxStatus
Function prototype	`void ENET_GetRxStatus(ENET_RxStatus *RxStatus)`
Behavior description	Gets Rx Status flags from ENET_MRS register
Input parameter	Pointer to structure of type ENET_RxStatus (see below)
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ENET_RxStatus structure

The ENET_RxStatus structure is defined in the *91x_enet.h* file:

```
typedef struct {
FlagStatus FrameAborted;
FlagStatus PacketFilter;
FlagStatus FilteringFail;
FlagStatus BroadCastFrame;
FlagStatus MulticastFrame;
FlagStatus UnsupportedControFrame;
FlagStatus ControlFrame;
FlagStatus LengthError;
FlagStatus Vlan2Tag;
FlagStatus Vlan1Tag;
FlagStatus CRCError;
FlagStatus ExtraBit;
FlagStatus MIIError;
FlagStatus FrameType;
FlagStatus LateCollision;
FlagStatus OverLength;
FlagStatus RuntFrame;
FlagStatus WatchDogTimout;
FlagStatus FalseCarrierIndication;
u16 FrameLength;
} ENET_RxStatus;
```

2.2.9 ENET_GetTxStatus

Function name	ENET_GetTxStatus
Function prototype	`void ENET_GetRxStatus(ENET_RxStatus *TxStatus)`
Behavior description	Gets Tx Status flags from ENET_MTS register
Input parameter	Pointer to structure of type ENET_TxStatus (see below)
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

ENET_TxStatus structure

The ENET_TxStatus structure is defined in the *91x_enet.h* file:

```
typedef struct {
FlagStatus PacketRetry;
u16 ByteCount;
u8 collisionCount;
FlagStatus LateCollisionObserved;
FlagStatus Deffered;
FlagStatus UnderRun;
FlagStatus ExcessiveCollision;
FlagStatus LateCollision;
FlagStatus ExcessiveDefferal;
FlagStatus LossOfCarrier;
FlagStatus NoCarrier;
FlagStatus FrameAborted;
} ENET_TxStatus;
```

2.2.10 ENET_UpdateRxDscr

Function name	ENET_UpdateRxPkt
Function prototype	`void ENET_UpdateRxDscr(void)`
Behavior description	Give the buffer back to ENET
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

2.2.11 ENET_HandleRxPkt

Function name	ENET_HandleRxPkt
Function prototype	`u32 ENET_HandleRxPkt(void *ppkt)`
Behavior description	Receives a packet and copies it to memory buffer pointed by ppkt.
Input parameter	ppkt: pointer on application receive buffer.
Output parameter	None
Return parameter	Packet size value or 0 (if there is no packet).
Required preconditions	None
Called functions	ENET_GetRxStatus ENET_UpdateRxDscr

2.2.12 ENET_UpdateTxDescr

Function name	ENET_UpdateTxDescr
Function prototype	`void ENET_SendPkt(void)`
Behavior description	Starts frame sending by setting the valid bit in the ENET DMA PACKET STATUS word.
Input parameter	None
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

2.2.13 ENET_HandleTxPkt

Function name	ENET_HandleTxPkt
Function prototype	`void ENET_HandleTxPkt(void *ppkt, u16 size)`
Behavior description	Performs the packet transmission: - copies the packet pointed by ppkt to the ENET DMA buffer memory, - starts packet sending by setting the valid bit in the PACKET STATUS word of the corresponding ENET DMA descriptor.
Input parameter 1	ppkt: pointer on application receive buffer.
Input parameter 2	size: the size of the packet to be copied.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_UpdateTxDscr

2.2.14 ENET_ITConfig

Function name	ENET_ITConfig
Function prototype	`void ENET_ITConfig(u32 ENET_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified ENET DMA interrupt.
Input parameter 1	ENET_IT: specifies the ENET DMA interrupt sources to be enabled or disabled. Refer to *Table 5: ENET_IT parameter values* the allowed values of this parameter.
Input parameter 2	NewState: new state of the specified ENET DMA interrupts. This parameter can be: ENABLE or DISABLE.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

Table 5. ENET_IT parameter values

EDMA_IT	Meaning
ENET_IT_TX_CURR_DONE	Current Transmission Done interrupt mask.
ENET_IT_TX_MERR_INT	Master Error during Transmission interrupt mask.
ENET_IT_TX_DONE	Transmission operations Done interrupt mask.
ENET_IT_TX_NEXT	Fetch Next Tx descriptor Error interrupt mask.
ENET_IT_TX_TO	Tx Timeout interrupt mask.
ENET_IT_TX_ENTRY	Tx DMA Entry Error interrupt mask.
ENET_IT_TX_FULL	Tx FIFO Full interrupt mask.
ENET_IT8TX_EMPTY	Tx FIFO Empty interrupt mask.
ENET_IT_RX_CURR_DONE	Current receive done interrupt mask.
ENET_IT_RX_MERR_INT	Master Error during receive interrupt mask.
ENET_IT_RX_DONE	Receive operations done interrupt mask.
ENET_IT_RX_NEXT	Fetch Next Rx descriptor Error interrupt mask.
ENET_IT_PACKET_LOST	Packet Lost interrupt mask.
ENET_IT_RX_TO	Rx Timeout interrupt mask.
ENET_IT_RX_ENTRY	Rx DMA Entry Error interrupt mask.
ENET_IT_RX_FULL	Rx FIFO Full interrupt mask.
ENET_IT_RX_EMPTY	Rx FIFO Empty interrupt mask.

Note: In case of single descriptor, the ENET_IT_TX_DONE interrupt is the same as ENET_TX_CURRENT_DONE. But the difference is noticed when a linked list of descriptors is programmed:

- Enabling ENET_IT_TX_CURR_DONE will generate interrupt after the end of each transmission.

- Enabling ENET_IT_TX_DONE will generate interrupt after the whole list chain transmission.

2.2.15 ENET_GetItSrc

Function name	ENET_GetItSrc
Function prototype	`u32 ENET_GetItSrc(void)`
Behavior description	Specifies the source of the ENET DMA interrupt.
Input parameter	None
Output parameter	None
Return parameter	The ENET DMA interrupt source.
Required preconditions	None
Called functions	None

2.2.16 ENET_ClearIT

Function name	ENET_ClearIT
Function prototype	`void ENET_ClearIT(u32 EDMA_IT)`
Behavior description	Clears the ENET DMA's interrupt pending bit.
Input parameter	ENET_IT: specifies the interrupt pending bit to clear. Refer to *Table 5: ENET_IT parameter values* the allowed values of this parameter.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	None

2.2.17 ENET_PHYITConfig

Function name	ENET_PHYITConfig
Function prototype	`void ENET_PHYITConfig(u8 phyDev, u32 PHY_IT, FunctionalState NewState)`
Behavior description	Enables or disables the specified PHY interrupt.
Input parameter 1	phyDev: specifies the PHY device address.
Input parameter 2	PHY_IT: specifies the PHY interrupt source to be configured.
Input parameter 3	NewState: new state of the specified PHY interrupt.
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIWriteReg ENET_MIIReadReg

2.2.18 ENET_PHYGetITSrc

Function name	ENET_PHYGetITSrc
Function prototype	`u32 ENET_PHYGetITSrc(u8 phyDev)`
Behavior description	Specifies the source of the PHY interrupt
Input parameter	phyDev: specifies the PHY device address
Output parameter	None
Return parameter	The PHY source interrupt
Required preconditions	None
Called functions	ENET_MIIReadReg

2.2.19 ENET_PHYIsolate

Function name	ENET_PHYIsolate
Function prototype	`void ENET_PHYIsoalte(u8 phyDev, FunctionalState NewState)`
Behavior description	Isolates the PHY device from the MII
Input parameter 1	phyDev: specifies the PHY device address
Input parameter 2	NewState: new state of link PHY/MII
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIReadReg ENET_MIIWriteReg

2.2.20 ENET_PHYPowerdown

Function name	ENET_PHYPowerdown
Function prototype	`u32 ENET_PHYPowerdown(u8 phyDev, FunctionalState NewState)`
Behavior description	Enable or disable the power down mode of the PHY device
Input parameter 1	phyDev: specifies the PHY device address
Input parameter 2	NewState: new state of the PHY power mode
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIReadReg ENET_MIIWriteReg

2.2.21 ENET_PHYLoopBack

Function name	ENET_PHYLoopBack
Function prototype	`u32 ENET_PHYLoopBack(u8 phyDev, FunctionalState NewState)`
Behavior description	Enable or disable the loop-back mode of the PHY device
Input parameter 1	phyDev: specifies the PHY device address
Input parameter 2	NewState: new state of the PHY power mode
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIReadReg ENET_MIIWriteReg

2.2.22 ENET_PHYReset

Function name	ENET_PHYReset
Function prototype	`u32 ENET_PHYReset(u8 phyDev)`
Behavior description	PHY device software reset
Input parameter	phyDev: specifies the PHY device address
Output parameter	None
Return parameter	None
Required preconditions	None
Called functions	ENET_MIIWriteReg

3 Webserver demo with uIP TCP/IP stack

3.1 uIP overview

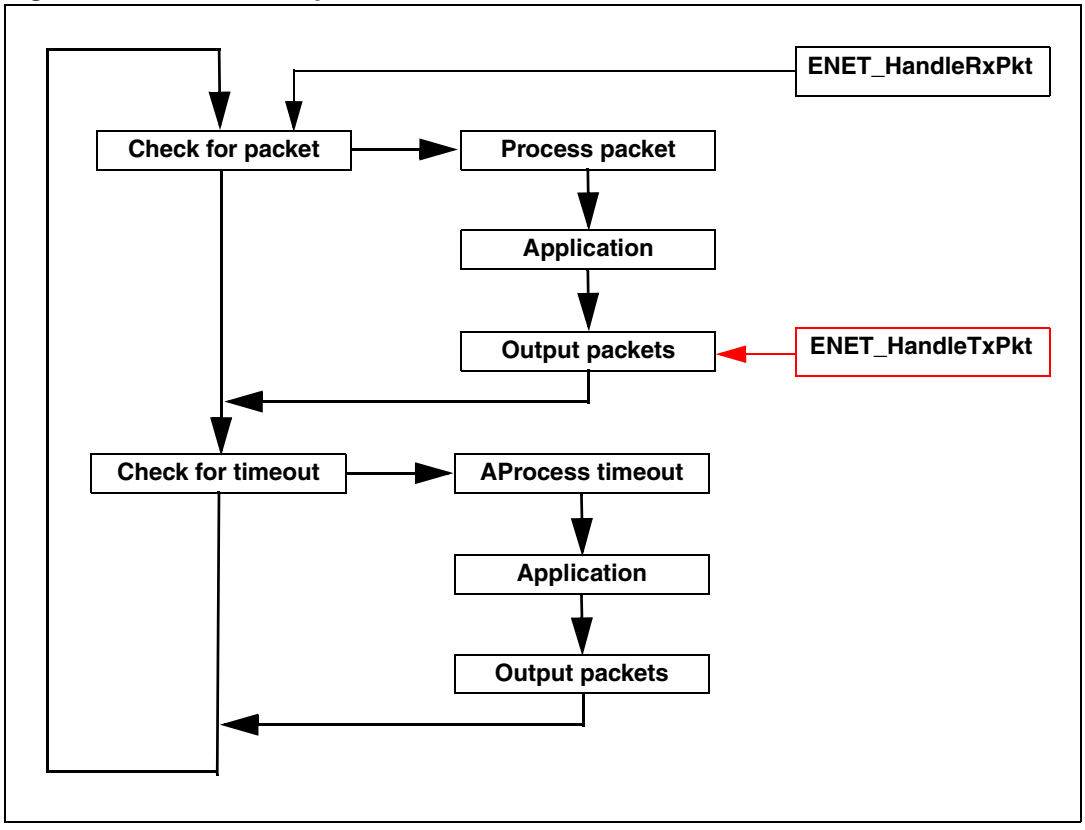
uIP is a free TCP-IP stack designed originally for 8-bit/16-bit microcontrollers, it needs only a few kilobytes of RAM and less than 100 Kbytes for code.
The uIP stack provides support for several applications like web server and client applications, ICMP , telnet,...

For more details about the uIP stack please refer to: www.sics.se/~adam/uip/

3.2 ENET driver interfacing with the uIP stack

The webserver demo in the uIP stack is based on a main loop as shown in *Figure 3*.

Figure 3. uIP main loop



In order to interface with the uIP stack, you just need to provide driver functions to receive and send IP packets.

The driver functions used here are:

- ENET_HandleTxPkt : sends an IP packet.
- ENET_HandleRxPkt: checks for a received packet and copies it to the stack buffer.

For implementation details please refer to uIP stack file: uIPMain.c file

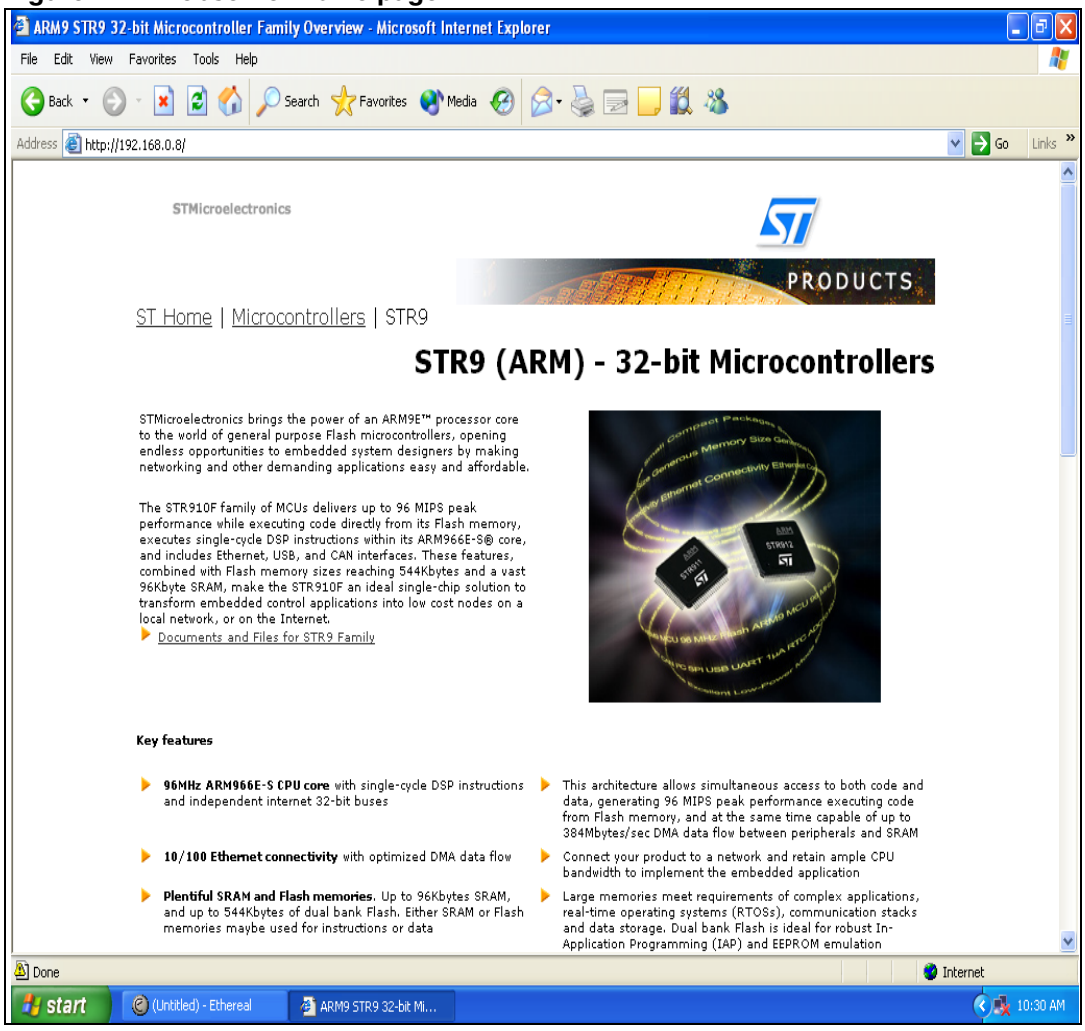
3.3 Running the webserver demo

In order to test the webserver demo application you need to:

1. Connect the STR9-EVAL board to a PC using a crossover Ethernet cable or through an Ethernet switch.
2. Change your PC IP address to: 192.168.0.x (with x value different from 8)
3. If you are using a firewall application, you will need to disable it.
4. Run the webserver demo
5. To test the connection, ping the board using the following address: 192.168.0.8.
6. Connect to the uIP webpage by typing http://192.168.0.8 on your browser

The following page should appear on the browser:

Figure 4. Webserver home page



4 Revision history

Table 6. Document revison history

Date	Revision	Changes
29-May-2006	1	Initial release.
4-Feb-2008	2	Revised for rev 2 firmware release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com