MAC Engine Star

Revision History

Date	Revision	Revision
2010-07-22	Initial version	1.0
2010-07-28	Reduced the number of GPIO_LED external signals from 8 to 4 to be compliant with the SP601 hardware.	1.1
2015-04-02	Updated documentation to include instantiation options	1.2

4DSP LLC.

Email: support@4dsp.com

Table of Contents

1 Introduction

A constellation containing the MAC engine can be accessed through the Ethernet interface. At the time of writing this is supported on the ML605 and SP601 development boards from Xilinx. The MAC engine is a bridge between the Ethernet PHY and other stars.

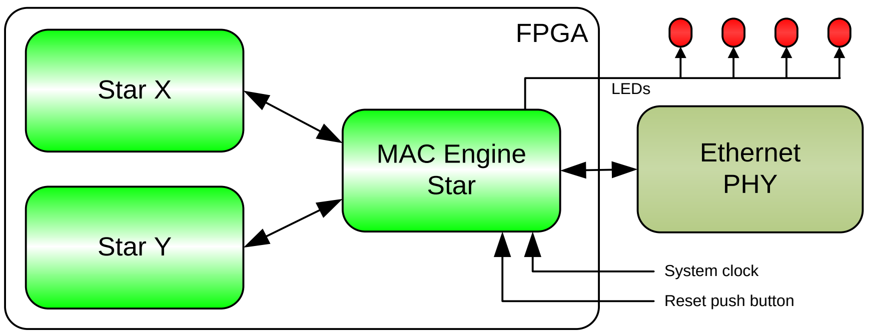

Figure 1: Hardware/firmware topology

2 Firmware Architecture

The MAC engine consists of three main blocks:

- Board clocks, containing a PLL to derive local clock signals from the system clock available on the hardware.
- GE MAC stream, the physical interface to the Ethernet PHY.
- Packet engine, the bridge between the incoming/outgoing MAC stream and the Stellar IP wormholes.

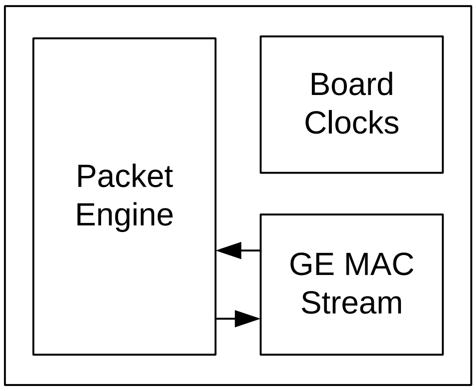

Figure 1: Firmware architecture

3 Star instantiation options

Depending on the target hardware configuration, it may be required to use different source and constraint files. This is supported by StellarIP through the .lst file. In most cases a different FPGA family requires a different FPGA type. Below is a list of the current lst files and their intended usage.

Table 1: Available lst files

File name	Target board	Description
sip_vc707_mac_engine_sgmii_v7.lst	VC707	This lst file is required when targeting boards with a V7 (485t) FPGA .
sip_vc707_mac_engine_sgmii_v7_vivado.lst	VC707	This lst file is required when targeting boards with a V7 (485t) FPGA when using Vivado Design Suite.

In StellarIP, the star index is used to target a specific UCF or XDC file for the star. StellarIP always searches for the UCF or XDC file that starts with the constellation name followed by target board name and then the star index. Not all possible combinations are available, but it does not necessarily mean a specific board/fpga type cannot be supported.

Table 2: Available UCF files

File name	Target board	Target FPGA type
sip_vc707_mac_engine_sgmii_vc707_0.ucf	VC707	X485T

For constellations that are targeting Vivado the star offers XDC files. The following table shows which XDC files are available and when they should be used.

Table 3: Available XDC files

File name	Target board	Target FPGA type
sip_vc707_mac_engine_sgmii_VC707_0.xdc	VC707	X485T

4 Wormhole descriptions

4.1 Rst_out wormhole of type rst_out

Signal Name	Direction	Width	Description
Rstout	Output	32	A vector that carries the same reset signal on all signals of the vector. Pushing the hardware reset button as well as an access to the reset register causes the reset to be asserted.

4.2 Clkout wormhole of type clkout

Signal Name	Direction	Width	Description
Clkout	Output	32	A vector that carries a 125MHz clock on all signals of the vector. Data wormholes wh_in and wh_out are synchronised to this clock.

4.3 Cmdclk_out wormhole of type cmdclk_out

Signal Name	Direction	Width	Description
Cmdclk	Output	1	A 125MHz clock output. Command wormholes are synchronous to this clock.

4.4 Cmd_out wormhole of type cmd_out

Signal Name	Direction	Width	Description
Cmdout	Output	64	This signal caries the outgoing command data packet [63..60] = command word [59..32] = address [31..0] = data
Cmdout_val	Output	1	When asserted high the cmdout carries a valid command packet.

4.5 Cmd_in wormhole of type cmd_in

Signal Name	Direction	Width	Description
Cmdin	Input	64	This signal caries the incoming response data packet [63..60] = command word [59..32] = address [31..0] = data
Cmdin_val	Input	1	When asserted high the cmdin carries a valid response packet.

4.6 In_data wormhole of type wh_in

Signal Name	Direction	Width	Description
In_stop	Output	1	In_stop is asserted by the MAC engine when its internal FIFO is almost full. The sending star should stop transferring data to the MAC engine when in_stop is asserted.
In_dval	Input	1	Asserted when in_data holds valid data that needs to be written to the MAC engine.
In_data	Input	64	Input data bus.

4.7 Out_data wormhole of type wh_out

Signal Name	Direction	Width	Description
Out_stop	Input	1	Out_stop is asserted by the receiving star when its internal FIFO is almost full. The MAC engine will stop transferring data to the receiving star when out_stop is asserted.
Out_dval	Output	1	Asserted when out_data holds valid data that needs to be written to the receiving star.
Out_data	Output	64	Output data bus.

4.8 Ext_mac_engine wormhole of type ext_mac_engine

Signal Name	Direction	Width	Description
Cpu_reset	In	1	Global reset input, physically connected to the reset push-button on the board.
Sysclk_p	In	1	System clock
Sysclk_n	In	1	System clock, inverted
phy_reset_l	Out	1	Connects to the external Ethernet PHY.
phy_mdc	Out	1	Connects to the external Ethernet PHY.
phy_mdio	Inout	1	Connects to the external Ethernet PHY.
phy_txctl_txen	Out	1	Connects to the external Ethernet PHY.
phy_txer	Out	1	Connects to the external Ethernet PHY.
phy_txc_gtxclk	Out	1	Connects to the external Ethernet PHY.
phy_txd	In	1	Connects to the external Ethernet PHY.
phy_txclk	Out	8	Connects to the external Ethernet PHY.
phy_crs	In	1	Connects to the external Ethernet PHY.
phy_col	In	1	Connects to the external Ethernet PHY.
phy_rxer	In	1	Connects to the external Ethernet PHY.

phy_rxctrl_rxdv	In	1	Connects to the external Ethernet PHY.
phy_rxclk	In	1	Connects to the external Ethernet PHY.
phy_rxd	In	8	Connects to the external Ethernet PHY.
gpio_led	Out	4	Physically connected to LEDs on the board.

5 Registers Map

The register map of the Mac engine star is defined in the table below.

Address	Bit 31..16	Bit 15..1	Bit 0
0x00	Reserved		RST
0x01	Reserved		
0x02	Reserved		
0x03	DST_MAC_L		
0x04	Reserved	DST_MAC_H	

Table 4: Register Map

- The addresses are absolute addresses. No address offset is assigned to the MAC engine star.
- The **RST** field is used to assert a global reset. This bit is self clearing. **DST_MAC_L** and **DST_MAC_H** can be used to overwrite the default destination MAC address. DST_MAC_L holds the lower 4 bytes, DST_MAC_H holds the upper 2 bytes

6 Access through the Ethernet Interface

The MAC engine supports register writing, register reading, auto offload, block writing, and block reading. An Ethernet frame consists of:

- 6 bytes destination MAC address
- 6 bytes source MAC address
- 2 bytes protocol (0xF000)
- 46-1500 bytes data
- 4 byte checksum

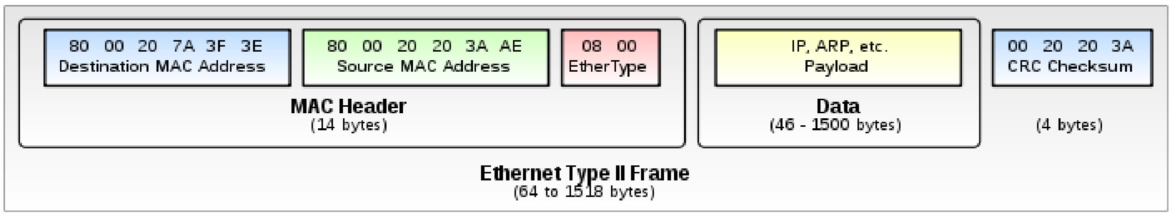

Figure 2 Ethernet frame

The MAC address of the board is 34_44_53_50_30_31 (4DSP01 in ASCII). The default MAC address used as destination for sending frames is 34_44_53_50_30_30 (4DSP00 in ASCII). The default destination MAC address can be overwritten by registers DST_MAC_L and DST_MAC_H.

6.1 Register writing

For register writing an Ethernet frame is send to the board. The frame data consist of:

- 1 byte operation code; 1 for register access
- 4 bytes address field consisting of:
  - 4-bit Stellar-IP command; 1 for register writing
  - 28-bit register address offset
- 4 bytes register data
- 37 bytes zero padding

```
# Ethernet HEADER
  34 44 53 50 30 31   # Destination MAC
  34 44 53 50 30 30   # Source MAC
  F0 00               # Protocol
# Data
  01                  # OPCODE
  20 01 00 10         # CMD(0x1) + Address(0x0000120)
  35 00 10 00         # Data (0x00100035)
# Zero padding
  00 .. .. .. .. 00   # 37 bytes
# CRC
  00 00 00 00
```

6.2 Register reading

For register reading an Ethernet frame is send to the board and a response frame will be received. The data in the request frame consist of:

- 1 byte operation code; 1 for register access
- 4 bytes address field consisting of:
  - 4-bit Stellar-IP command; 2 for register reading
  - 28-bit register address offset
- 4 bytes register data (don’t care)
- 37 bytes zero padding

```
# Ethernet HEADER
  34 44 53 50 30 31   # Destination MAC
  34 44 53 50 30 30   # Source MAC
  F0 00               # Protocol
# Data
  01                  # OPCODE
  30 04 00 20         # CMD(0x2) + Address(0x0000430)
  00 00 00 00         # Data
# Zero padding
  00 .. .. .. .. 00   # 37 bytes
# CRC
```

```
  00 00 00 00
```

The data in the response frame contains:

- 1 byte operation code; 0x81 for register access acknowledge
- 4 bytes address field consist of:
    - 4-bit Stellar-IP command; 4 for register read response
    - 28-bit register address offset
- 4 bytes register data
- 37 bytes zero padding

6.3 Auto offload

In auto offload mode data from the board is send in frames to Ethernet port. The maximum amount of data in a single frame is 1500 bytes; a data block may therefore be split into multiple frames. A frame consists of:

- 1 byte operation code; 0x82 for auto offload
- 1 byte channel number
- 2 bytes sub block size; range from 8 to 1488 in steps of 8 bytes.
- 4 bytes start address (0 for the first frame of a block)
- 4 bytes block size (block size = sum of all sub block sizes)
- 8 to 1488 block data bytes (zero padded to 34 bytes when sub block size is smaller)

```
# Ethernet HEADER
  34 44 53 50 30 31   # Destination MAC
  34 44 53 50 30 30   # Source MAC
  F0 00               # Protocol
# Data
  82 01 20 00         # OPCODE(0x82) / CHANNEL(0x01) / SIZE(0x0020)
  00 00 00 00         # Start address(0x00000000)
  20 00 00 00         # Block size(0x00000020)
  00 01 02 03 04 05 06 07  # Block data bytes
  08 09 0A 0B 0C 0D 0E 0F
  10 11 12 13 14 15 16 17
  18 19 1A 1B 1C 1D 1E 1F
  00 00               # Zero padding
# CRC
  00 00 00 00
```

6.4 Block writing

For writing a block of data an Ethernet frame is send to the board. The maximum amount of data in a single frame is 1500 bytes; a data block may therefore be split into multiple frames. The data of a frame consists of:

- 1 byte operation code; 3 for block writing
- 1 byte channel number

- 2 bytes sub block size; range from 8 to 1488 in steps of 8 bytes.
- 4 bytes start address (0 for the first frame of a block)
- 4 byte block size (block size = sum of all sub block sizes)
- 8 to 1488 block data bytes (zero padded to 34 bytes when sub block size is smaller)

```
# Ethernet HEADER
  34 44 53 50 30 31   # Destination MAC
  34 44 53 50 30 30   # Source MAC
  F0 00               # Protocol
# Data
  03 01 20 00         # OPCODE(0x03) / CHANNEL(1) / SIZE(0x0020)
  00 00 00 00         # Start address(0x00000000)
  20 00 00 00         # Block size(0x00000020)
  00 01 02 03 04 05 06 07   # Block data bytes
  08 09 0A 0B 0C 0D 0E 0F
  10 11 12 13 14 15 16 17
  18 19 1A 1B 1C 1D 1E 1F
  00 00               # Zero padding
# CRC
  00 00 00 00
```

After the complete block of data has been received by the board an acknowledge frame is send, with in the data field:

- 1 byte operation code; 0x83 for block write acknowledge
- 1 byte channel number
- 2 bytes sub block size (don’t care)
- 4 bytes start address (don’t care)
- 4 byte block size
- 34 bytes zero padding

6.5 Block reading

For reading a block of data an Ethernet frame is send to the board and response frame(s) will be received. The data in the request frame consists of:

- 1 byte operation code; 4 for block reading
- 1 byte channel number
- 2 bytes sub block size (don’t care)
- 4 bytes start address (0 for the first frame of a block)
- 4 bytes block size
- 34 bytes zero padding

```
# Ethernet HEADER
  34 44 53 50 30 31   # Destination MAC
  34 44 53 50 30 30   # Source MAC
  08 00                 # Protocol
# Data
  04 01 00 00           # OPCODE(0x04) / CHANNEL(1) / SIZE(0x0000)
  00 00 00 00           # Start address(0x00000000)
  00 00 10 00           # Block size(0x00100000)
# Zero padding
  00 .. .. .. .. 00     # 34 bytes
# CRC
  00 00 00 00
```

The response frame(s) will contain the block data. The maximum amount of data in a frame is 1500 bytes; a data block may therefore be split into multiple frames, with in the data field:

- 1 byte operation code; 0x84 for block reading response
- 1 byte channel number
- 2 byte sub block size; range from 8 to 1488 in steps of 8 bytes.
- 4 bytes start address (0 for the first frame of a block)
- 4 byte block size (block size = sum of all sub block sizes)
- 8 to 1488 block data bytes (zero padded to 34 bytes when sub block size is smaller)