Theory of Operation and General Documentation of the Ethernet Development Board v1.0 and its Software Components

Stefan Gloor

February 2, 2020

Abstract

I created this Ethernet Development Board as a learning platform for myself to develop network-enabled hardware and software. It offers a stand-alone solution with an 100Base-TX Ethernet Controller which can be used in conjuction with the on-board microcontroller. Alternatively, the board can also be used as a daughterboard for an Altera DE0-nano FPGA board to add 100Base-TX Ethernet functionalities to it. This document is a collection of thoughts that went into the development of this project and is meant as an addition to the doxygen documentation, which describes the microcontroller software in more detail. It can be used as a reference guide for future uses of its components in different projects.

Contents

1 Hardware **1**
- 1.1 Mechanical . . . 2
- 1.2 Capabilities . . . 2
- 1.3 Versions . . . 3
  - 1.3.1 FPGA . . . 3
  - 1.3.2 MPU . . . 3
- 1.4 Known Bugs . . . 4

2 Software **4**
- 2.1 Capabilities . . . 4
- 2.2 Theory of Operation . . . 5
  - 2.2.1 Memory Controller . . . 5

1 Hardware

When looking at the block diagram (Figure 1) it becomes clear that the two Ethernet channels can operate completely independent from one another. This is why different board versions can easily be achieved

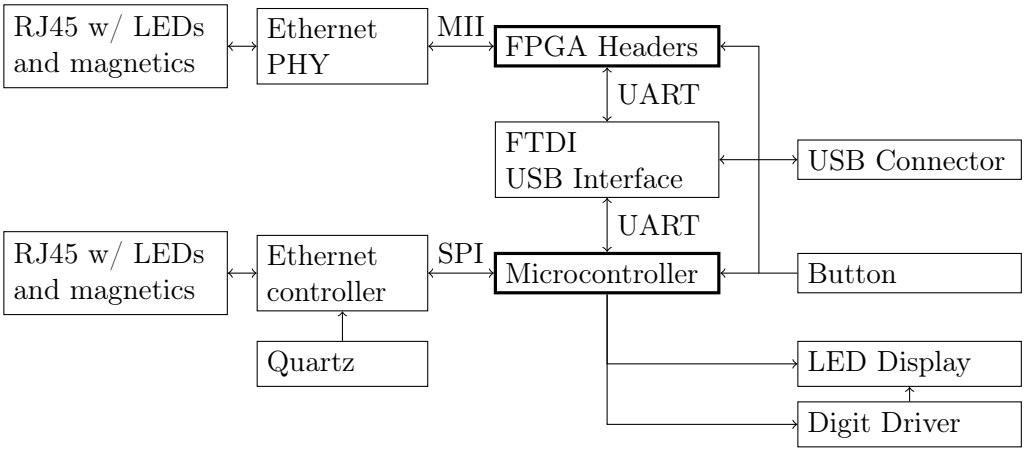

Figure 1: Block diagram illustrating the hardware of the development board.

by only populating the necessary components. Both systems (FPGA and MPU control) can be used in conjunction with the USB debug interface and the user button. However, the two serial interfaces can not be used simultaneously, since they both use the same physical connections. Therefore the transmit line output of one system must remain high-impedance at all times to prevent communication collisions. Since the UART pins of the Altera DE0-nano board can be chosen arbitrarily, a communication between the FPGA and the microcontroller could be achieved by using the shared UART debug interface, when the USB debug output is not needed. The seven-segment display can only be driven by the microcontroller. The microcontroller can be programmed in-system by the dedicated programming header. This header is designed to be used in conjunction with a Microchip PICKit3. Of course it would be possible to make it a field-programmable device by adding an Ethernet or USB (through the serial FTDI interface) bootloader.

1.1 Mechanical

The mechanical dimensions of the device as well as the location of the mounting holes were designed to match the ones on Altera's DE0 nano board. That way the footprint of the stack (FPGA board + EthernetDevBoard) is not bigger than the one of the DE0 nano alone.

1.2 Capabilities

Both Ethernet channels support IEEE 802.3 compliant 10Base-T/100Base-TX with auto-negotiation. The FPGA channel supports MDI/MDI-X (straight-through/crossover cable) detection and correction, Wake-on-LAN, as well as faulty cable diagnostics. Power-over-Ethernet is not supported by either version.

1.3 Versions

As mentioned before, two versions of the development board can be achieved by populating the same PCB differently. I decided to create a fully-featured "FPGA" version and a lightweight "MPU" version which does not support communication with the Altera board. Since the FPGA version is fully populated, any software that runs on the MPU version is also fully compatible with the FPGA version. A third possibility, a FPGA-only version would theoretically be possible but has not been further tested, because it is not considered needed at this stage of the project. A more detailed description of the version differences can be found in the mechanical drawing document of the same name and in the BOM documents.

1.3.1 FPGA

Since the FPGA channel is only using an Ethernet PHY chip (KSZ8091) instead of a controller chip, MAC capabilities have to be implemented in the FPGA design. The PHY chip basically does the low-end interface to the physical medium (hence its name). The chip communicates with the FPGA through MII (Media Independend Interface). It passes the parsed bits recovered from the line to the FPGA. It does automatic MDI/MDI-X detection and correction, auto-negotiation, cable diagnostics etc.

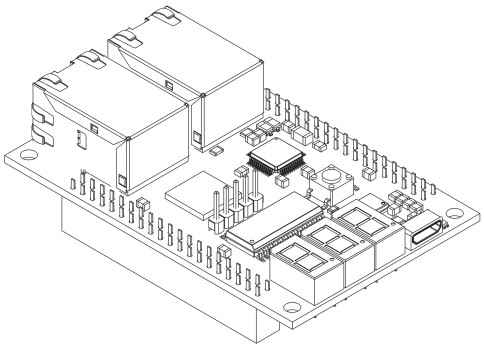

Figure 2: Version "FPGA" of the development board.

1.3.2 MPU

The MPU Ethernet channel of the board uses a highly integrated ethernet controller chip (ENC424J600). This controller chip offers PHY and MAC capabilities. It has integrated DMA-enabled buffer memory for sending and receiving packets. Which part of the memory is used for transmission or reception respectively, can be chosen freely. This offers great customizability for the target application. It does automatic CRC insertion and checking, automatic padding, MAC address insertion and filtering, auto-negotiation etc. on its own. It is connected to the microcontroller using a serial SPI interface.

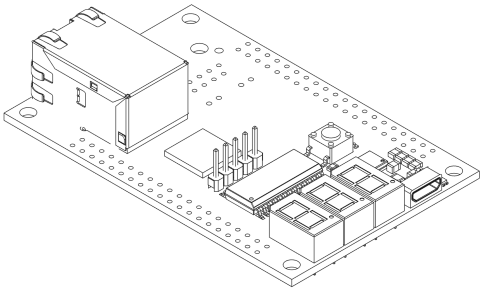

Figure 3: Version "MPU" of the development board.

1.4 Known Bugs

- Component SW1 (Pushbutton): Has the wrong footprint/schematic symbol. It is shorted out.
  Workarounds:
  - Remove upper right pin of the device (RJ45 jacks on the left).

  Solutions:
  - Change footprint and schematic symbol.
  - Choose different component with matching footprint.
- Component X5 (RJ45, Ethernet channel): Both LEDs are inverted.
  Workarounds:
  - Poll Link Status in software and manually set the LED state accordingly in software. There is no workaround for the activity indicator LED.

  Solutions:
  - Connect pin 11 and 13 of X5 both to GND instead of +3V3.
- Missing thermal relief pads on the entire board.
  Workarounds:
  - Preheat PCB prior to soldering.

  Solutions:
  - Add thermal relief pads.

2 Software

2.1 Capabilities

The following list summarizes the features supported in the current version. The layer names refer to the OSI model.

1. Link Layer
   - Ethernet driver

  - Transmission and reception of Ethernet packets with variable lengths ranging from 64...1518 bytes.
  - Provision of packet information; e.g. Broadcast or Unicast flag, CRC error status etc.
- Address resolution protocol (ARP)
  - Resolving the MAC address to a given IPv4 address using ARP requests.
  - Replying to ARP requests using ARP replies.
  - Storing the received ARP messages in a table with an expiry time.
  - Probing for IP address after it has changed or a Link Status change occured.

2. Internet Layer
   - Internet Protocol Version 4 (IPv4): Fixed Header length of 20 Bytes..
3. Transport Layer
4. Application Layer

2.2 Theory of Operation

Figure 4 shows a basic system overview of the microcontroller board.

2.2.1 Memory Controller

This software was written with the assumption that it is run on a microcontroller which is not able to provide sufficient RAM to buffer entire Ethernet frames itself. Instead, it is designed to be used in conjunction with either an Ethernet controller, like ENC424J600, which offers internal SRAM or with a separate, external memory chip. Currently it is logically connected to the internal memory of ENC424J600, but it should be easily portable to a design which uses an external storage device by altering a few functions. This is why this memory controller only refers to a software component and not a physical device. A memory controller is used so several packets can be prepared for transmission simultaneously. This allows the IP module to write data to the buffer right after the memory field request, without being blocked by the ARP module which needs to send a request for the address resolution first.

The memory controller takes in a request for a memory field from the Ethernet block. The length of the requested memory field is passed to the memory controller, which then tries to fit the given number of bytes it into free space in the buffer. If it fits, it returns a start memory pointer back. Each protocol module is responsible for writing its data (with the correct length!) to the given location in buffer space. The start pointer is handed through the software stack, with each protocol incrementing the pointer according to the upper layers data field. If, for example, the Ethernet module is given a start address of 0x0000, it will add its header length (2 MAC addresses + EtherType field = 18

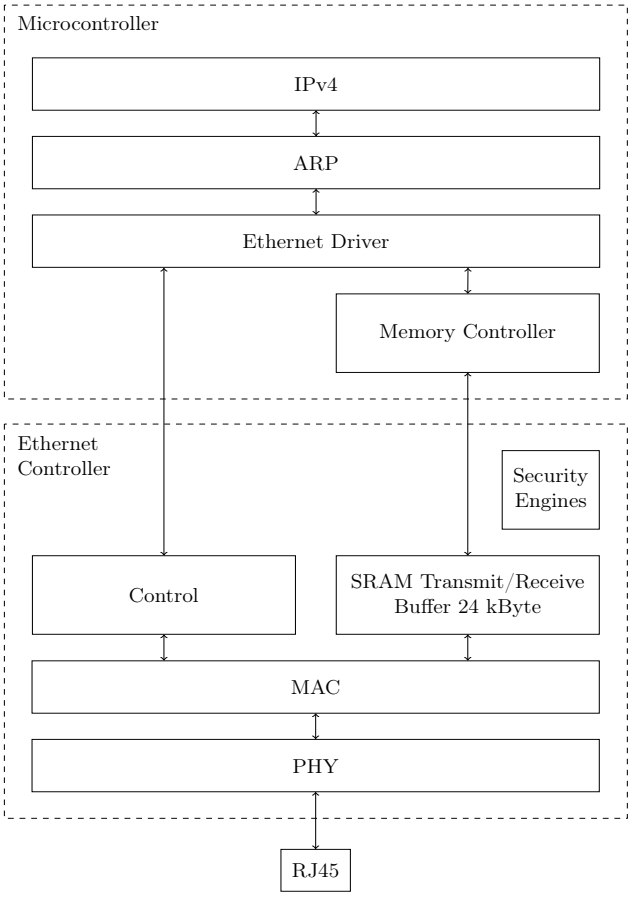

Figure 4: General system overview

Bytes) to the pointer value and hand over the start address of 0x0012 to the upper lP layer, which then adds its header length to the pointer value and gives the resulting address to the upper laying TCP module.

Figure 5 shows the basic operating principle of the memory controller. It uses memory fields, which are represented by an Assigned-flag, a starting and an ending address, and a length. The algorithm places the different fields with variable length at some point in free buffer space. It is also able to wrap fields around the end of the transmit buffer area. The fields can be freed up and reassigned again in an arbitrary order. The problem with this implementation is, that, if fields 0 to 2 are assigned, then field 1 gets cleared and reassigned to a new frame with a shorter length it creates a memory gap between the end of field 1 and the start address of field 2. This memory gap will not be used until field 2 gets reassigned. This leads to scenarios where the algorithm returns an Out-of-Memory error although there is still space left inbetween assigned fields. This is also partly due to the fact

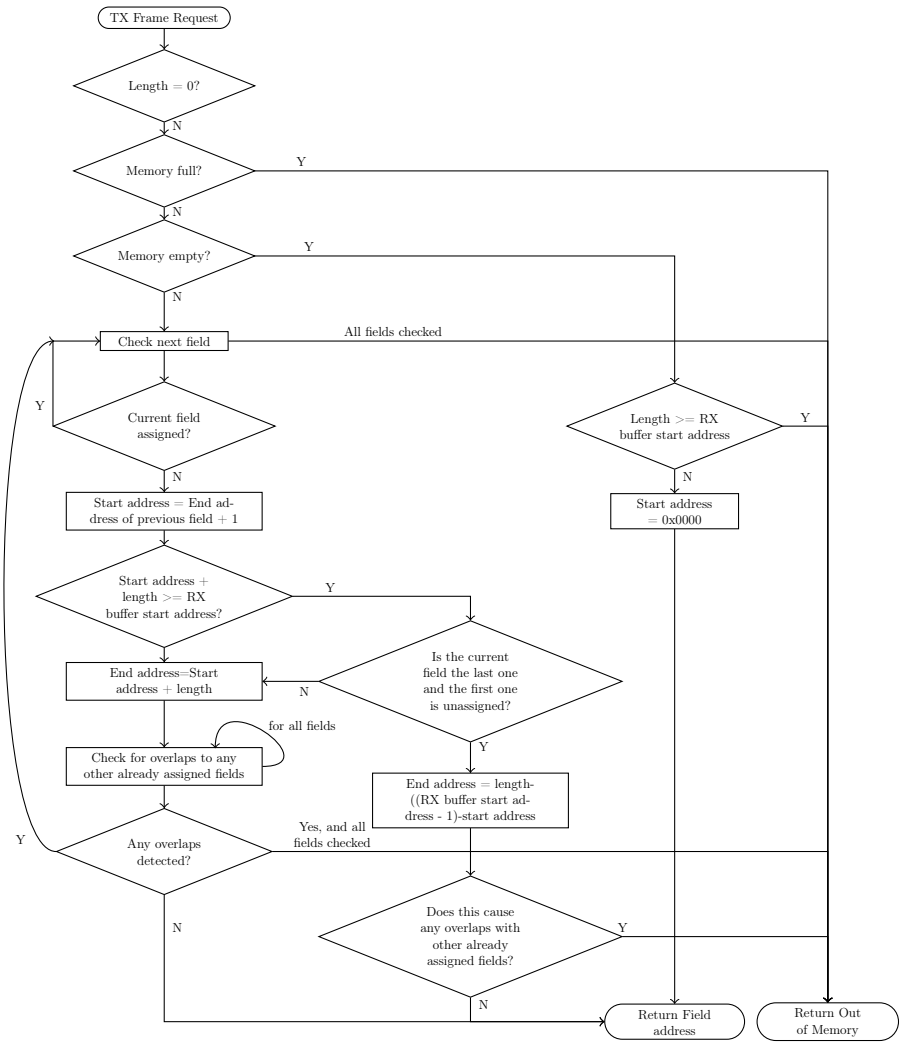

Figure 5: Flow chart of the buffer memory allocation algorithm

that this implementation uses a fixed number of memory fields.

After reset, there exists a finite number of memory fields which are all unassigned, with an undefined start and ending address. If the Ethernet module requests memory for storing an outgoing frame some invalid conditions (if the passed length equals zero) and some special edge cases (if the memory is already entirely full or completely empty) are catched. After this, all remaining unassigned fields are checked in a loop. After the start address is set, the algorithm checks for any resulting overlaps in this temporary configuration. If there are no overlaps, the Assigned-flag is set and the field information is returned back to the Ethernet module. The memory controller can also return an Out-of-Memory error when the algorithm couldn't fit the required frame length in the buffer. If the data transmission is completed, the

field has to be freed up manually by the module which wrote data into the buffer.