1 µTosNet

Compared to the TosNet framework, the idea of µTosNet is basically to cut out the TosNet network itself, creating a single-node system, where the node contains low-level hardware interfaces on one side, and a PC interface on the other. To create a working system, students and researchers will only need to create the low-level hardware interfaces in the FPGA, and an application on the PC. Everything else is provided by the framework.
By employing some of the work already done on the TosNet gateways for the µTosNet PC interface, the hope is to create a consistency, which will make it easy to transition both knowledge and existing modules between the two frameworks.
A number of example designs exist in the OpenCores repository, which should serve as a decent starting point for learning to use µTosNet.

1.1 Architectural and conceptual differences compared to TosNet

The main idea of µTosNet is to provide a simplified version of TosNet that is easier to use and understand, and that is usable on smaller FPGA chips. To achieve this, the following simplifications have been done to TosNet:

- The TosNet core is exchanged with µTosNet, consisting of a small dual-port BlockRAM
- The network is removed, thus only a single node is supported
- No double buffering or other enforced access control
- No asynchronous channel

As only a single node and no TosNet network are present in a µTosNet system, there are of course also no network cycles. The timing in the system is thus completely determined by the PC application and whatever device interface is implemented in the FPGA.
Block diagrams showing the difference between TosNet and µTosNet can be seen in figure 1.

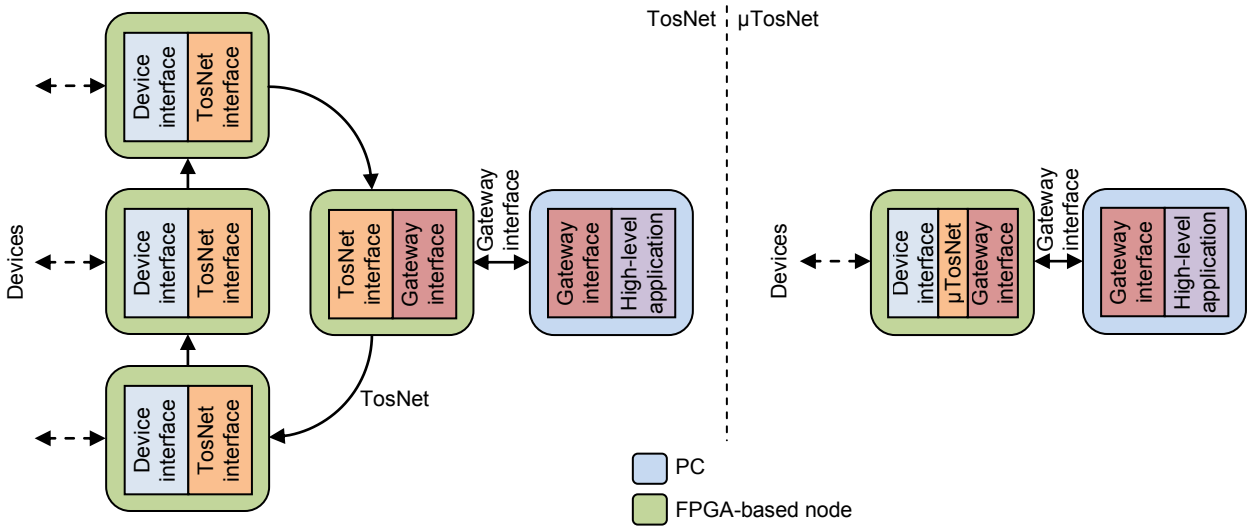

Figure 1: Examples of TosNet (left) and µTosNet (right) systems, showcasing the architectural differences between the two.

As can be seen from the figure, µTosNet cuts out the TosNet network, and basically just attaches devices directly to a gateway node. By reusing the same concepts as employed in TosNet, a high level of reuse will be possible between the two frameworks.

1.2 FPGA-side interface to µTosNet

The FPGA interface to the TosNet core mainly consists of an interface to the BlockRAM holding the registers, with some control signals added for handling commits, checking the current status of

the network, and for using the asynchronous channel. For µTosNet though, all these extra control signals are not necessary, and the interface thus only consists of an interface to the BlockRAM block.
To make it even easier for beginners to use µTosNet, a wrapper has been made by Anders Sørensen, which continuously runs through all registers and indexes, outputting the contents of the node's input registers, and accepting new data for its output registers.

1.3 Gateways

Of the interfaces and gateways that exist for TosNet, only the Ethernet and RS232 gateways are of interest for µTosNet.

1.3.1 Ethernet protocol

In the case of the Ethernet gateway a slightly simplified version of the version for the full TosNet is used. The protocol remains almost the same: the address fields are reduced to be 6 bits in size, and all flags relating to skip counters and commit signals are ignored. This can be seen in table 2.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Not used				Do read	Not used					Read Address						Not used				Do write	Not used					Write Address					

Table 2: The layout of the 32 bits in the control DWORD used for the generic gateway protocol for µTosNet. The unused fields are left available for future additions.

Due to requiring a Digi embedded module, the Ethernet based gateway is not assumed to be the one that will be used most with µTosNet. This position is instead held by the RS232 gateway, due to its greater simplicity and ease-of-use.

1.3.2 RS232 protocol

To adapt the RS232 gateway for use with µTosNet, the protocol is modified slightly. Only the read and write commands are now available, and with a simpler syntax:

- *Read*
  Reads the value of a register.

  ```
  Command:  rRI
  Response: XXXXXXXX
  ```

  The register index, *R*, and Dword index, *I*, should be numbers in the range 0-7. Dword indexes 0-3 indicate the output blocks of a specific register, while Dword indexes 4-7 indicate the input blocks. The response, *XXXXXXXX*, will consist of an 8-digit lower-case hexadecimal number.

- *Write*
  Writes a value to a register.

  ```
  Command:  wRI XXXXXXXX
  Response: N/A
  ```

  Again, the register index, *R*, and Dword index, *I*, should be numbers in the range 0-7. Dword indexes 0-3 indicate the output blocks of a specific register, while Dword indexes 4-7 indicate the input blocks. The value to write, *XXXXXXXX*, should consist of an 8-digit lower-case hexadecimal number, and must be preceded by a space character. No response is made.

Just like for the full TosNet, the protocol is implemented in VHDL as an acceptor-type finite state machine, and connected directly to the µTosNet BlockRAM.
Table 3 shows the default settings that should be used on the PC side, to connect to the FPGA.

Speed	115200
Data bits	8
Stop bits	1
Parity	None
Flow control	None

Table 3: The settings for using the RS232 / USB gateway.

2 Prerequisites

To use the µTosNet protocol, the following is needed:

- Xilinx ISE 9 or newer (other versions might work too, but have not been tested)
- Working Spartan3/6 board with RS232/USB or Digi Connect ME9210 interface
- VHDL files:
  - uTosNet_uart.vhd OR uTosNet_spi.vhd
  - uTosNet_ctrl.vhd (optional, only for UART version)
  - uart_rx.vhd (from PicoBlaze UART, not included)
  - uart_tx.vhd (from PicoBlaze UART, not included)
  - kcuart_rx.vhd (from PicoBlaze UART, not included)
  - kcuart_tx.vhd (from PicoBlaze UART, not included)
  - bbfifo_16x8.vhd (from PicoBlaze UART, not included)
- CoreGen XCO files: (use the appropriate files for your device, or generate new ones)
  - dataRegister.xco

As can be seen, the UART version of µTosNet depends on the PicoBlaze UART implementation by Ken Chapman. It can be downloaded for free from Xilinx' webpage:
http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm

3 How to instantiate the µTosNet VHDL module in an existing design

To use the network implementation for a design, the following steps should be followed:

1. Add all the *.vhd* and *.xco* files to the project.
2. Add the UART tx/rx or SPI miso/mosi/enable/clk signals to the top level.
3. Connect the new top level signals to the correct pins on the FPGA device.
4. Instantiate the appropriate µTosNet module.
5. Connect the existing design to the needed BlockRAM or uTosNet_ctrl signals.

4 Interface signals

This section describes the interface signals used for communicating with the TosNet component. The *data_reg_* signals are directly connected to the BlockRAM, and thus work just like any other memory.

4.1 dataReg_addr<5:0> (input)

This is the address bus of the memory block, and is organized as can be seen in table 4.

5	4	3	2	1	0
Register			I/O	Dword	

Table 4: How to address into the µTosNet memory block.

The four different parts have the following meaning:

- *Register*
  The register to target.
- *I/O*
  '0' targets output space, '1' targets input space.
- *Dword*
  Specifies which 32 bit part of the 128 bit large register to target.

Unlike the full TosNet network, it is possible to ignore the *I/O* bit in the addressing, and use any register for input, output or both. This is not at all encouraged though, as it will decrease portability between µTosNet and TosNet.
With only 6 bits of addressing space, and no double buffering used, a µTosNet core will only use 256 bytes of BlockRAM, compared to the 8 kB used by the full TosNet (not including the network register).
So if for example the bitstring "100100" is put on the address bus, it will target the first 32 bits of the in-part of register "100".

4.2 dataReg_dataIn<31:0> (input) / dataReg_dataOut<31:0> (output) / dataReg_writeEnable<0:0> (input) / dataReg_clk (input)

These are the rest of the BlockRAM interface signals for the memory block. To write data to the block, do the following:

1. The data to write is put on *dataReg_dataIn*, the write enable signal, *dataReg_writeEnable*, is pulled high, and the address to write to is put on *dataReg_addr*.
2. *dataReg_clk* is pulled high, which performs the write.

To read data from the block, do the following:

1. The write enable signal, *dataReg_writeEnable*, is pulled low, and the address to read from is put on *dataReg_addr*.
2. *dataReg_clk* is pulled high, which performs the read.
3. The read data is now available on *dataReg_dataOut*.

4.3 serial_in (input) / serial_out (output) OR spi_miso (output) / spi_mosi (input) / spi_clk (input) / spi_en (input)

These are the signals from and to the transmitter components.

4.4 clk_50M (input)

This is the 50 MHz clock signal required by the TosNet core.

5 Conclusion

Realizing that the full TosNet system is overkill in many situations that only require a single low-level hardware device to be interfaced to for instance a PC, and that the FPGA boards required to hold the TosNet network core are quite expensive, µTosNet was created. µTosNet implements a scaled-down, single-node version of the interfaces found in a standard TosNet system with an RS232/USB gateway, and thus provides the same interfacing benefits at a much lower complexity and cost.