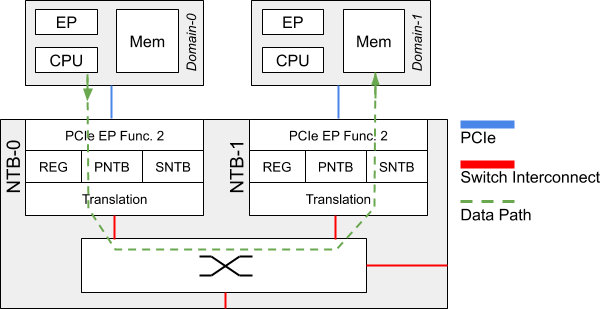
**. PCIe NTB to Connect Multiple CPUs, GPUs & FPGAs**

NTB stands for Non-Transparent Bridge. Unlike in a PCIe (transparent) Bridge where the RC “sees” all the PCIe busses all the way to all the Endpoints, an NTB forwards the PCIe traffic between the separate PCIe busses like a bridge. Each RC sees the NTB as an Endpoint device but does not see the other RC and devices on the other side. Means, everything behind the NTB is not directly visible to the particular RC, thus “Non-Transparent”.

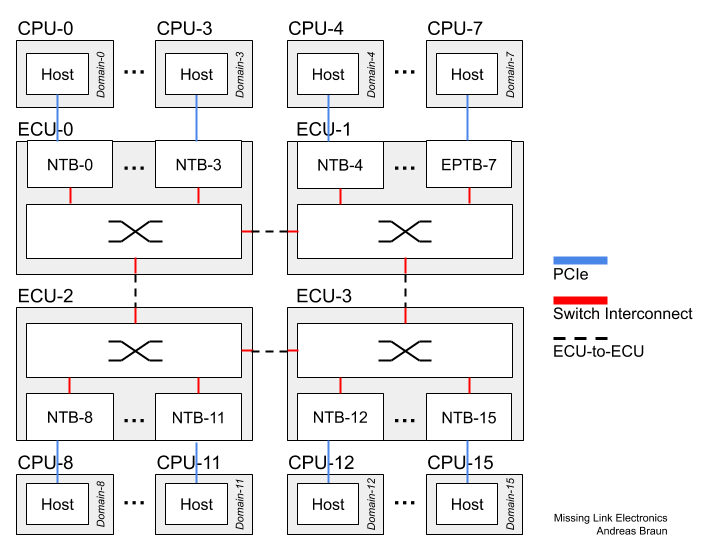
Figure 5: Connectivity via PCIe Non-Transparent Bridge

For each attached RC there is a memory aperture allocated to the NTB within the main memory of said RC. PCIe writes (and reads) are translated across. Given the nature of PCIe so-called Posted Writes, writing data into a remote RC’s main memory carries less overhead than reading data from a remote RC’s main memory. Therefore, at MLE for sending data from one RC to a second RC we fully use this concept of Posted Writes together with so-called Doorbell Registers to implement a very efficient and high-performance communication mechanism. Groundwork for this was laid more than a decade ago by work from Jack Regula  and others . Figure 6 shows this communication mechanism across an NTB based on a ring buffer structure for the example of two RCs, Host A and Host B.

Figure 6: Receiving data Using Write-Only Communication and Ring Buffers

As NTB has been enjoying support in the Open Source Linux kernel we do adhere to this de facto standard API described in Linux ntb.h  and made our programmable logic implementation compatible to it.

Unlike in Datacenter applications with their wide PCIe busses of 16 or 32 lanes (or even more) in an automotive ECU you have to be frugal with those high-speed IO lanes. This means that we cannot use the Daisy Chain connectivity scheme for connecting multiple RCs (means three or more) which is, for example, described in “Intel Xeon Processor C5500/C3500 Series Non-Transparent Bridge”:  When using four, or less, PCIe lanes to connect, traffic would quickly saturate the PCIe links of the RCs on the “inside” of the Daisy Chain. Therefore at MLE we have implemented a Peer-to-Peer connectivity scheme which, by the way, can be done quite efficiently in an FPGA in terms of resources and performance when using Xilinx AXI4 network-on-chip infrastructure. This Peer-to-Peer connectivity scheme avoids lookup tables for translating between sender and receiver PCIe Device IDs, something that has to be done using the Daisy Chain approach.

Figure 7: System-of-Systems with multiple ECUs

And, we can also extend connectivity to more than 32 RCs (the PCIe Device ID field is only 5 bits) without any extra measures which would add to the overall communication latency. This enables you to build a system-of-systems, for example by connecting multiple ECUs via automotive high-speed Ethernet (10G or soon 25G) as shown in Figure 7 where all 16 CPUs can directly exchange data with all other 15 CPUs via PCIe.

Furthermore, one of the key reasons of using multiple CPUs or GPUs or FPGAs in an automotive application is resilience to hardware failures to deliver a high level of functional safety according to DIN/ISO 61508 or DIN/ISO 26262. This is why we have added certain checks into MLE’s NTB technology.

The PNTB (Primary NTB) which is in charge of egress PCIe traffic, besides other checks, on-the-fly validates address ranges – easy to do in a Xilinx FPGA – to prevent that a software on the sending RC sends data from memory spaces other than the memory space pre-registered for NTB communication. Similarly, the SNTB (Secondary NTB) which is in charge of ingress PCIe traffic checks that communication writes only into valid address spaces. In combination, this also allows us to maintain a reliable communication even if a SEU (Single Event Upset) might have flipped an address bit. Further checks allow to detect malfunction of any RC and to respond to it by rerouting the PCIe traffic to a backup RC.

With all these measure it made implementing MLE’s NTB technology quite a challenge – and this is why we looked closely into good tools for FPGA system debugging.