The ESI System Construction Compiler in 2024

John Demme (john.demme@microsoft.com), Microsoft

ABSTRACT

Writing IP blocks is only half the battle in accelerator design. Although under-researched and under-discussed, system construction – gluing together blocks, connecting them to the host, and creating a host runtime – is a significant challenge with huge implications for productivity and deployment. ESI (Elastic Silicon Interconnect) is a system construction compiler which assists in all of the above tasks which we presented at LATTE'21.

Mostly in response to the needs of real, production systems, ESI and its plans have evolved considerably since we initially presented it at LATTE'21. Herein we detail the expanded current and planned capabilities of ESI and – perhaps more importantly – why they are important in building useful, complex, performant, supportable accelerators.

1 INTRODUCTION

While there has been a huge amount of work in languages for hardware accelerator IP block design, compilers for all the bits around them – connecting them and host software together – primarily focus on ASIC-style SoC, which tend to focus on one or more CPU cores driving/coordinating accelerators all tied together with a fully-connected NoC. communication synthesis[1, 2, 6, 7]. High-performance FPGA designs rarely fit in that SoC category and thus require different (though overlapping) features from a compiler, especially for accelerators with wide production deployment.

Now somewhat of a misnomer, ESI (Elastic Silicon Interconnect) is a system construction compiler targeting FPGAs. Originally, it merely provided typed, elastic[3, 4] point-to-point connections in between IP blocks and to the host. By the time we presented it at LATTE'21[5], it had grown to include the notion of *s*ervices – a mechanism to standardize both access to global resources (e.g. DRAM) and for the host to access IP resources (e.g. function calls, telemetry).

When designing an FPGA accelerator intended for deployment, however, there are a number of practical concerns that are not addressed (or are only partially addressed) by the current state of the art. We categorize those concerns and describe the problems, why it is important, and how ESI (will) ameliorate it.

2 ON-CHIP SIGNALING CONCERNS

Although not well-known or documented (and beyond the scope of this paper to show), valid-ready skid buffers limit FMax, especially in congested designs.[1] Instead, FPGA designs often (should) use feed-forward datapaths/interconnects (and possibly a runout FIFO at the end). Multiple levels of complexity exist to deal with back-pressure (flow control). The simplest is FIFO signaling wherein one simply pipelines the 'full' signal from the properly-sized runout FIFO. A more complex scheme involves the receiver issuing credits to the sender.

ESI signaling. At module boundaries (ports), ESI supports two different control styles: valid-ready and credits. To pipeline the connections, ESI does support skid buffers; however, it also supports FIFOs and credits and can/will automatically convert the module interface control signaling to a different pipeline control signaling scheme to optimize the interconnect. For instance, if IP block uses valid-ready signaling the designer can choose to avoid using skid buffers by instructing ESI to synthesize a FIFO-based pipelining control scheme with a valid-ready module signaling interface.

3 RESETS AND CLOCKS

ESI channels need to be aware of *clock domains* since they sometimes must include CDCs, which are notoriously error-prone when written (or merely instantiated) by hand. Additionally, software may want to control clock domains to save power when a particular IP block is not in use.

IP block resets are often just a 'rst' port which is held high for a number of cycles. This is insufficient since blocks can take arbitrarily long to come up (e.g. PHY training times are unbounded).

Bus resets occur whenever one has a set of channels between two different reset domains. Since either side can be reset independentally, the other side must know to reset its contol signaling scheme (credits). Additionlly, it may also be useful to know at the application level that the other side has been reset. For instance, if one side is a DRAM controller it needs to know that the IP which sent a request has been reset and isn't expecting the response any more thus the DRAM controller needs to drop the outstanding responses.

ESI resets and clocks. At compile time, IP blocks request a clock and a number of resets associated with that clock. ESI will synthesize a clock/reset controller (which can be poked from software) and drive the requests. ESI will automatically synthesize CDCs appropriately when it detects channels which need them. For IP block resets, ESI defines a handshaking reset protocol allowing the IP block to dynamically inform the reset controller that it is ready to go. Similarly, ESI defines a handshaking bus reset protocol and can automatically insert them and drive their signals based on the block-level reset.

4 HOST CONNECTIVITY CONCERNS

Often a significant bottleneck in end-to-end performance, data transfer between host and accelerator is riddled with tradeoffs due to CPU memory systems' complexity[2]. Often, it is easier to copy data into a large slab of physically contiguous shared memory for DMA transactions at the expense of *very significant* load on the CPU merely doing **memcpy**s, cutting into or destroying the benefits of

LATTE '24, April 28th, 2024, San Diego, CA, USA
2024.

[1] Intuitively, *a*ny sort of logic added to the datapath should be avoided. Since full-throughput skid buffers add a mux in the data path and control logic to drive the select (which is potentially high-fanout), they not only limit the length of wires (creating more congestion in the logic areas they are routed through), but they also preclude using "hyper registers" in the routing fabric (which avoid using ALM/CLB registers).
[2] Specifically, virtual memory and paging

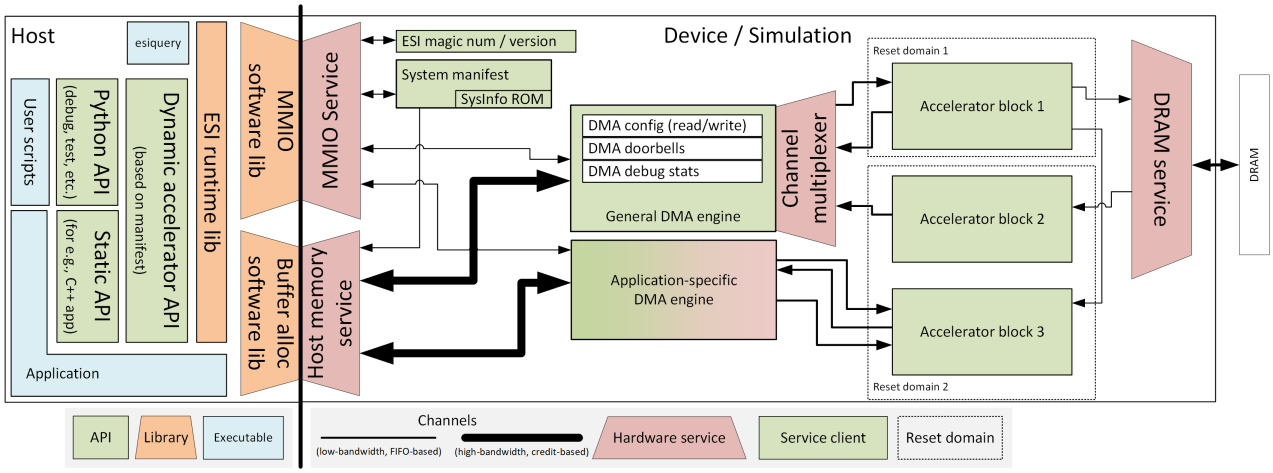

Figure 1: An example ESI system demonstrating some of the solutions discussed.

acceleration. As a result, the importance of avoiding host **memcpy**s ends up meaning that applications often need custom DMA engines which know something about host memory layout and implement application-driven access patterns/control schemes.

ESI host memory access service allows multiple DMA engines through a layered services approach. The base service is a "host memory" service which provides access to host memory (simple reads and writes to/from host memory addresses). On top of that, different DMA engines can be built with generic or custom control protocols. One may (and probably will) have a low-performance, generic DMA engine to enable misc host communications combined with one or more application-optimized engines.

Supporting this on the host side, the ESI runtime is heavily layered/pluggable (allowing customization at a number of points in the API layering). It will be *very* flexible in terms of data locations, allowing the API user to highly optimize **memcpy**s.

5 DEPLOYMENT CONCERNS

Partial compatibility. In production, atomic updates of distributed systems *never* happen. Rather, one part of the system must support a previous version until the entire system gets updated. Though a slightly narrower scope, in hardware acceleration one cannot break compatibility with host-side software from one version of an accelerator to the next unless it is actually a relevant breaking change *in the part of the accelerator software is using*. In software, this is typically handled by an RPC protocol like Protobuf/gRPC. Accelerators cannot afford the overhead of full RPC messaging, so often times this aspect is overlooked leading to headaches when trying to update large deployments.

Versioning, metadata, and telemetry. In support scenarios, it is often useful to quickly be able to tell, "what the $*&# is this FPGA running???" This may include the image name, version, git hash, repo url, short descriptions of the various IP blocks (and block parameters), and other metadata. Similarly, it is often useful to be able to quickly examine and interpret runtime metrics to debug either during development or – more importantly – during a high severity incident. Looking up offsets into the PCIe BAR, descriptions, and the unit of particular telemetry in documentation (which may or may not be the correct documentation for the image and image version running) and/or git repositories is both error-prone and slow.

ESI system manifests contain all relevant information about what's on the accelerator and how to communicate with it, including metadata about the IP blocks and a list of telemetry metrics. Further, it gets embedded into the image.[3] The ESI software runtime has the capability to get this manifest from *any* ESI accelerator, so one can – for instance – determine what an FPGA is running and gather readable, interpretable telemetry about its operation without any sideband information or application-specific executables.

Said manifest also contains type signatures for all of the channels on the accelerator, so the ESI runtime also provides a Python API which constructs a dynamic, accelerator-specific API to interact with the accelerator, enabling testing/debugging without needing a software API generated at design construction time.

Manifests also have the advantage of supporting partial compatibility. Statically generated APIs (i.e. C++ APIs) will check – at runtime – if the generated API for a particlar IP block is indeed type-compatible with the statically-generated types it uses. It also exposes the IP block version to the application for it to determine semantic compatibility.

6 LOOKING FORWARD

These concerns and corresponding ESI solutions represent only the issues which we have identified and solved (or planned a solution) to date (though we didn't cover a few due to space limitations). We have yet to start exploring multi-FPGA systems, security issues, FPGA DRAM sharing, or network-attached FPGAs just to name a few complexity-rich areas. Doubtless there will be others so creative (or obvious), well planned solutions will be necessary. We look forward to presenting them in future publications!

REFERENCES

[1] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste

[3] In zlib-compressed JSON format, stored in an ESI-standardized location.

Asanović, and Borivoje Nikolić. 2020. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21. https://doi.org/10.1109/MM.2020.2996616

[2] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[3] Luca P Carloni, Kenneth L McMillan, and Alberto L Sangiovanni-Vincentelli. 2001. Theory of latency-insensitive design. *IEEE Transactions on computer-aided design of integrated circuits and systems* 20, 9 (2001), 1059–1076.

[4] Josep Carmona, Jordi Cortadella, Mike Kishinevsky, and Alexander Taubin. 2009. Elastic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 10 (2009), 1437–1455.

[5] John Demme. 2021. Elastic Silicon Interconnects: Abstracting Communication in Accelerator Design. In *LATTE workshop.* https://capra.cs.cornell.edu/latte21/

[6] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. 2020. Agile SoC Development with Open ESP. *CoRR* abs/2009.01178 (2020). arXiv:2009.01178 https://arxiv.org/abs/2009.01178

[7] Tianrui Wei, Nazerke Turtayeva, Marcelo Orenes-Vera, Omkar Lonkar, and Jonathan Balkind. 2023. Cohort: Software-oriented acceleration for heterogeneous socs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3.* 105–117.