A New Design Workflow for PYNQ Enabled Xilinx Platforms Utilising the Simulink Environment for Vivado IPI Abstraction

Lewis D. McLaughlin, Louise H. Crockett, and Robert W. Stewart
Department of Electronic and Electrical Engineering,
University of Strathclyde, Glasgow, Scotland
Email: {lewis.mclaughlin, louise.crockett, r.stewart}@strath.ac.uk

Recently, FPGAs have been coupled with processors to form System on Chip (SoC) devices. SoC design is challenging as it combines both hardware and software elements. A variety of tools must therefore be utilised to design for these components, each requiring different knowledge.

The hardware accelerated portion of the system, targeting the programmable logic (PL), requires experience in writing HDL or tools which generate HDL such as MathWorks' HDL Coder, Xilinx System Generator, or Vivado High-Level Synthesis (HLS). Furthermore, this design must then be integrated with the platform, where interfacing with the processing system (PS) and any on-board peripherals must be considered in addition to clocks, resets and interrupts. This is usually achieved in Vivado IP Integrator (IPI), although methodologies exist for simplifying this platform integration such as MathWorks' Reference Designs [1] and the toolflow developed by The Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) [2]. Targeting the PS, software can be designed as a bare metal application, written in C, or written in Python when using a PYNQ enabled platform [3].

An engineer must consider all these aspects when designing for Xilinx SoCs. Therefore, reducing the amount of tools the engineer is exposed to will lead to increased productivity in addition to lessening the learning curve, making SoC design more accessible. This work achieves this by consolidating all hardware design into one environment and encouraging code reuse with the introduction of a new block type which generates Python to be run on the target.

Simulink was chosen as the frontend as it is a widely adopted tool which offers a rich, block-based design environment, providing information such as sample times, data types and port connectivity. Consideration has been made when developing this workflow to ensure additional environments can be integrated at a later date, opening the door to open-source frontends. The hardware design element of this workflow differs to that of established tools such as HDL Coder or System Generator, as it not only generates HDL for the algorithm performing the hardware acceleration, but also integrates this component, dynamically, with a given platform, using information provided by additional blocks in the frontend design. These blocks represent existing Vivado IPI blocks and have been parsed into Python objects and given an additional Python wrapper, allowing them to be abstracted from Vivado IPI and exploited in other tools. In this case, the MATLAB Engine API for Python is utilised to assimilate Vivado IPI blocks into Simulink.

The software design portion of the system is enhanced by the introduction of a new block type, the Pink Block. These Pink Blocks act as a hierarchy containing the Python objects representing IPI blocks and have associated Python classes which can be made parameterisable and use additional information provided by the frontend to generate application specific Python code for use on the target platform. When using Simulink, this additional information is provided using block masks. Data types found in the frontend are also identified by Pink Blocks allowing for all fixed-point conversions to be automatically implemented in the generated Python code when reading from and writing to the PS. This allows for libraries of hardware/software blocks to be built up for common tasks and reused easily across other applications.

By moving all hardware design, from the hardware accelerated algorithm through to platform integration, into one environment and coupling this with a process for attaching parameterisable Python code to single or groups of hardware blocks, this work enables rapid prototyping of SoC designs and makes the overall design process more accessible.

References

[1] MathWorks, *Board and Reference Design Registration System*, 2022. [Online]. [Accessed: 15-Jan-2022]. Available: https://www.mathworks.com/help/hdlcoder/ug/board-and-reference-design-system.html

[2] J. Hickish *et al.*, *A Decade of Developing Radio-Astronomy Instrumentation using CASPER Open-Source Technology,* Journal of Astronomical Instrumentation, vol. 5, no. 4, 2016. Available: https://arxiv.org/abs/1611.01826

[3] Xilinx, *PYNQ: Python Productivity*, 2022. [Online]. [Accessed: 15-Jan-2022]. Available: http://www.pynq.io/.

We acknowledge funding for Lewis McLaughlin under EPSRC grant no: EP/R513349/1.