Within the past decade, there has been a rise in technology which incorporates software processors with specialised hardware to boost productivity and performance. However, this comes with many challenges stemming from the skills required to design the hardware elements which requires hardware description languages (HDL) such as Verilog and VHDL. High level synthesis (HLS), which provides a higher level of abstraction, was developed to tackle this problem. Over the years, HLS has evolved to increase practicality and ease of adoption, yet several challenges still exist which stagnate greater integration into the world of software-hardware development.

Historically, the HLS workflow has been broken down into several stages, the most important being parsing, scheduling, and binding. The parsing stage involves transforming the high-level program (nowadays C, C++) into some intermediate form, typically a control & data flow graph (CDFG), for later stages to work on. Optimisation is an important part of this stage, and among the available techniques there are a few that are especially relevant to HLS. Next, the scheduling stage divides the CDFG into a series of time slots, to enable the creation of a finite state machine. Many algorithms have been developed over the years for this, from simple greedy ones to more sophisticated genetic ones. A designer may want to optimise for time or resources, and some algorithms perform better than others for each. Finally, binding matches operations to hardware by choosing an appropriate library component for each operation, and adding registers where required. In modern HLS tools, e.g. Vivado HLS, these three stages are performed together, which allows extra optimisations to be performed.

Despite these advances, HLS still doesn’t quite provide the software developer a simple tool to port to hardware. The complexity of hardware systems is currently growing faster than the productivity of system designers and programmers. Some people argue that the foundation of common HLS tools in working from procedural languages has irreconcilable shortcomings when it comes to hardware description, and there have been recent flavour of the month alternative HLS tools and specific languages developed from a functional paradigm that claim to address these issues more intuitively than the C-extended languages such as ROCCC and eXcite. We will be discussing the merits of these different approaches in reference to the goals of HLS in reducing the design productivity gap.