High-level Synthesis with **SLX FPGA**

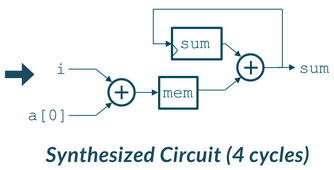
Matthias Gehre
gehre@silexica.com

SILEXICA
www. silexica.com

FPGAs can provide significant increases in performance when compared to general purpose processors, but they are hard to program for software developers. Even with the advent of High-level synthesis (HLS), which uses a subset of C/C++ plus HLS-specific pragmas to generate the FPGA bitstream, both the transformation of original C++ code to the supported subset and the insertion of appropriate pragmas requires large effort and deep FPGA expertise.

```
int top_level_hw(std::array<int, 4> a) {
  int sum = 0;
  for(int i = 0; i < a.size(); i++){
    sum += a[i];
  }
  return sum;
}
```

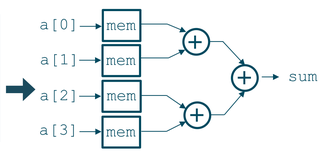
Example Source Code



Synthesized Circuit (4 cycles)

```
int top_level_hw(std::array<int, 4> a) {
  int sum = 0;
  for(int i = 0; i < a.size(); i++){
    #pragma HLS unroll factor=4
    sum += a[i];
  }
  return sum;
}
```

Example Source Code

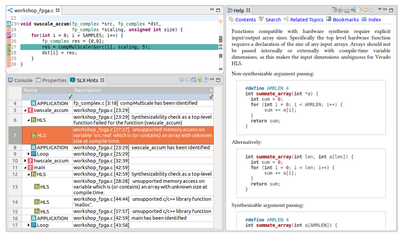


Synthesized Circuit (1 cycle)

HLS pragmas allow to exploit the parallel capabilities of an FPGA

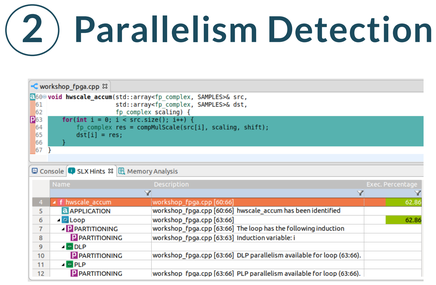
SLX FPGA provides a step by step flow to implement C/C++ code on an FPGA using High Level Synthesis, significantly reducing development time and ensuring an optimized hardware implementation of C/C++ code.

1 Refactor Non-Synthesizable Code for HLS



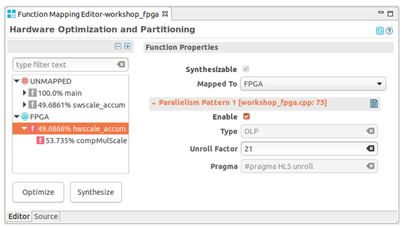
Refactoring C/C++ code for HLS **requires a significant amount of experience** with the methodology and tools while the available coding guidelines can seem overwhelming to non-experts. **SLX eases the rewriting of C/C++** software for HLS by automatically identifying incompatible constructs and **providing contextual guidelines** with examples to the users.

2 Parallelism Detection



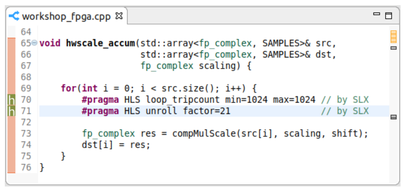
C/C++ code is typically executed sequentially on standard processors. SLX FPGA analyzes application C/C++ code using both **static analysis and runtime tracing** to identify functions and loops that can be **accelerated by executing in parallel** on an FPGA. The analysis exploits **LLVM[1] IR** to infer program flow and variable accesses that are visible **statically**. The remaining information (e.g. access through a pointer) is gathered by **instrumenting** the LLVM IR and **running** the application. From that information, functions and loops are analyzed for possible parallelism, taking into account the observed **control flow and data dependencies**.

3 HW Optimization and HW/SW Partitioning



The tool **iteratively optimizes** possible configurations of implementing profitable loops and functions to hardware and **estimates their effects on performance and area usage**. As the result of this process, **SLX FPGA** selects the configuration that provides the **best performance** while still satisfying user-supplied **area constraints**. The best configuration is then expressed in the form of HLS pragmas.

4 Pragma Insertion



Once the optimized hardware implementation is determined, SLX FPGA inserts **HLS pragmas** into the source code to direct the HLS compiler on how to implement the function in hardware.

SLX FPGA is fully integrated with **Xilinx® Vivado® HLS** and the **SDSoC™** Development Environment to create a complete path from C/C++ to FPGA synthesis.

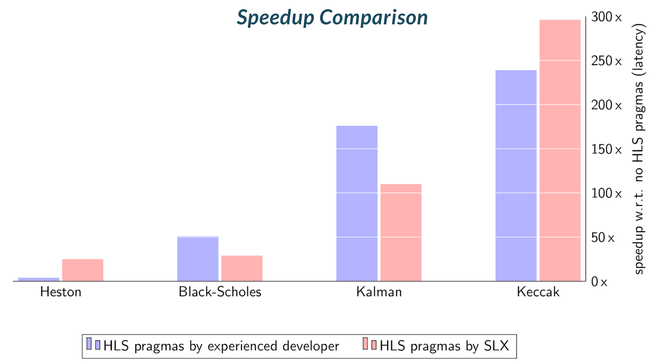
XILINX®
ALLIANCE PROGRAM MEMBER

5 Results

We have taken applications which had been annotated with HLS pragmas by experienced developers. We removed all pragmas and then ran **SLX FPGA** to automatically analyse and insert HLS pragmas. We compared the worst case latency estimates of both versions targeting the Xilinx® XCZU9EG FPGA[2]. The applications are

- ***Keccak:*** *SHA-3 cryptographic hashing algorithm[3]*
- ***Kalman:*** *MATLAB®-generated code for a Kalman filter using fixed- point arithmetic[4]*
- ***Black-Scholes :*** *Monte Carlo method for the Black-Scholes Model for option pricing[5]*
- ***Heston:*** *Monte Carlo method for the Heston model for option pricing[6]*

Speedup Comparison



1. http://www.llvm.org
2. Xilinx XCZU9EG FPGA contains 274K Lookup tables, 550K Flip-flops, 32 Mb memory, 2520 DSP slices.
3. http://cryptography.gmu.edu/athena/index.php?id=source_codes
4. https://www.silexica.com/blog/hls-from-MATLAB-generated-cc/
5. https://github.com/KitAway/BlackScholes_MonteCarlo
6. https://github.com/KitAway/FinancialModels_AmazonF1/tree/master/heston_model

For more info about **SLX FPGA**