Vitis Acceleration Flow on Trenz Electronic Modules

Lattice Filter as Systolic Array HW IP on Zynq UltraScale+ TE0808 Modules

Authors

Jiri Kadlec, UTIA AV CR v.v.i. kadlec@utia.cas.cz
Raissa Likhonina, UTIA AV CR v.v.i. likhonina@utia.cas.cz

Context

Lattice filter is used for adaptive noise cancellation. UTIA is using acceleration of the Lattice DSP algorithm for demonstration of support for Vitis acceleration flow on Trenz Electronic modules.

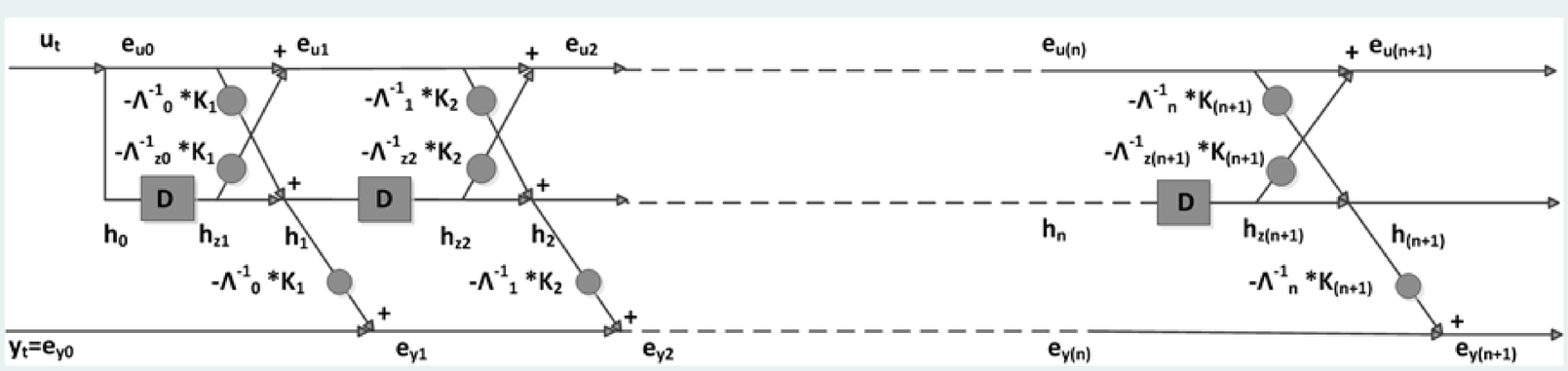
OBJECTIVES

1. Acceleration of relatively complex DSP algorithm in HW.
2. UTIA support for Vitis acceleration flow for HW accelerators on custom, industrial grade, Trenz Electronic Zynq UltraScale+ modules.
3. Demonstration of integration of custom, SIMD DPUs with HW data movers generated by HLS compiler from Vitis.
4. Demonstration of generated HW and acceleration results.

Lattice Filter to be accelerated as a Systolic Array in HW on Zynq UltraScale+

RLS QR LATTICE FILTER - TIME AND ORDER RECURSIVE DSP ALGORITHM

- System is modelled as finite impulse response (FIR) recursive least square (RLS) filter.
- Order recursion enables to solve the RLS system identification problem with linear complexity.
- Computation stages can be implemented as a systolic array suitable for on SIMD DPUs.



Start of the algorithm in time $t_0 = N + 1$

$\delta_0 > 0, \quad \delta_0 \to 0, \quad \alpha^2 \epsilon \langle 0, 1 \rangle$

$\Lambda(N+1|n) = \delta_0, \quad pro\ n = 0, 1, \ldots, N$

$\Lambda_z(N+1|n) = \delta_0, \quad pro\ n = 1, 2, \ldots, N$

$K(N+1|n) = 0, \quad pro\ n = 1, 2, \ldots, N$

Computation in time $t \geq t_0$

$\Lambda_0(N+1|n) = \alpha^2 \cdot (\Lambda_0(N+1|n) + y_t^2)$

$\nu = \alpha^2 \cdot (\nu + 1)$

$\xi_0(n) = 0$

$e_{u0}(N+1|n) = h_0(N+1|n) = u_t$

$e_{y0}(N+1|n) = y_t$

Cycle for $n = 1, 2, \ldots, N$

$h_z(N+1|n) = h \quad (N+1|n-1)$

$K_u(N+1|n) = \alpha^2 \cdot (K_u(N+1|n) + \frac{h_z(N+1|n) \cdot e_u(N+1|n-1)}{1 + \xi(n-1)})$

$\Lambda_z(N+1|n) = \alpha^2 \cdot (\Lambda_z(N+1|n) + \frac{h_z^2(N+1|n)}{1 + \xi(n-1)})$

$h(N+1|n) = h_z(N+1|n) - K(N+1|n) \cdot \Lambda^{-1}(N+1|n-1) \cdot e_u(N+1|n-1)$

$e_u(N+1|n) = e_u(N+1|n-1) - K(N+1|n) \cdot \Lambda_z^{-1}(N+1|n) \cdot h_z(N+1|n)$

$\xi(n) = \xi(n-1) + \Lambda_z^{-1}(N+1|n-1) \cdot h_z^2(N+1|n)$

$\Lambda(N+1|n) = \alpha^2 \cdot (\Lambda(N+1|n) + \frac{e_u^2(N+1|n)}{1 + \xi(n)})$

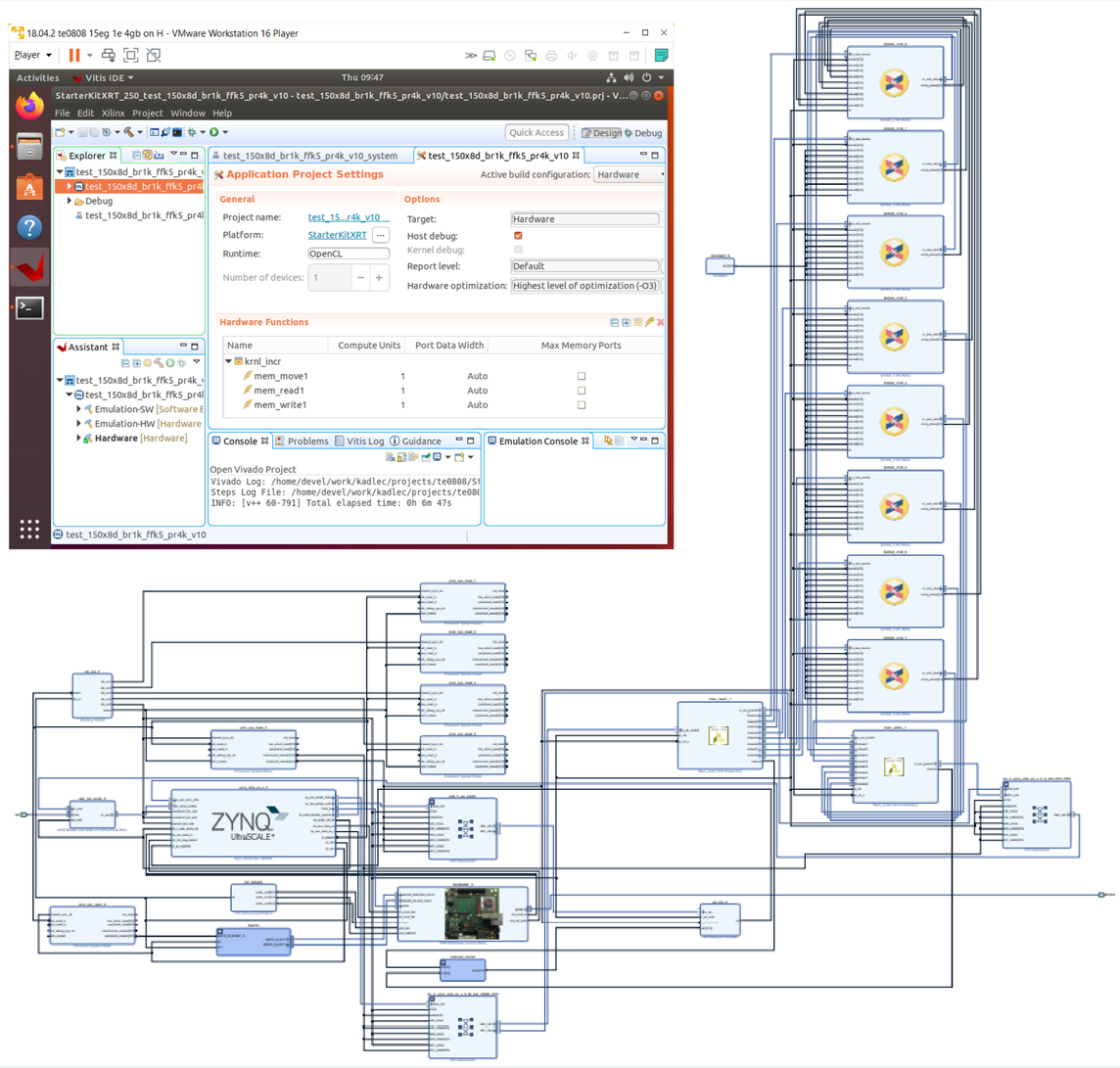
$K_y(N+1|n) = \alpha^2 \cdot (K_y(N+1|n) + \frac{h_z(N+1|n) \cdot e_y(N+1|n-1)}{1 + \xi(n-1)})$

$e_y(N+1|n) = e_y(N+1|n-1) - K_y(N+1|n) \cdot \Lambda^{-1}(N+1|n) \cdot h \quad (N+1|n)$

Vitis OpenCL implementation on TEBF0808 carrier and TE0808-09/15EG 1E modules

IMPLEMENTATION OF LATTICE, ORDER 1024

- Lattice stages are mapped to 64 FP32 computing layers located in eight 8xSIMD DPUs developed by UTIA.
- Each 8xSIMD DPU supports SIMD FP32 ADD, MUL and DIV vector operations. DPUs are programmed by 128 bit wide VLIW instructions (programs with up to 4096 instructions). DPUs are connected by 256bit wide AXI-S with I/O FIFOs.
- Data are transferred to and from the DPUs by HW data mover functions generated by the HLS compiler in Vitis.
- DPUs are connected via device buffers allocated in DDR4 by a dedicated data mover HW generated also in Vitis.
- Lattice filter internal stage variables are updated in place and stay inside of DPUs.
- The SW API is the Vitis 2019.2 OpenCL and XRT.
- **ARM 4 cores** 256 threads: 60s of data computed in **200 sec**
- **HW** (240MHz) 256 threads: 60s of data computed in **17 sec**



ECSEL Joint Undertaking

storAIge project receives funding from European H2020 programme, ECSEL Joint Undertaking, and involved National Funding Authorities under grant agreement no. 101007321. In case of UTIA from MSMT 8A21009.