Vivado High Level Synthesis Tutorial

Aleksei Rostov, PhD,
Senior R&D Engineer,
FPGA/Embedded Linux Developer,
aleksei.rostov@protonmail.com

Agenda

- Introduction to Vivado HLS
  - Role for FPGA design
  - C/C++ language usage
  - Optimization strategies, RTL interfaces
- Design flow
- Case: FIR filter development

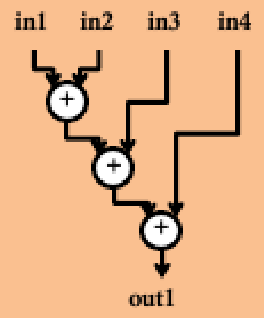
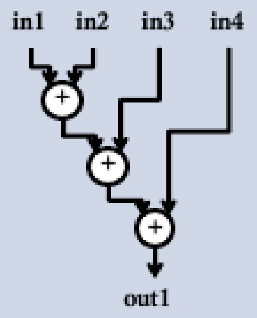
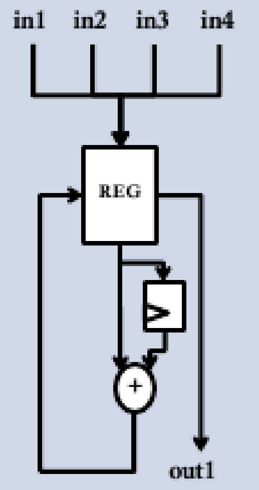
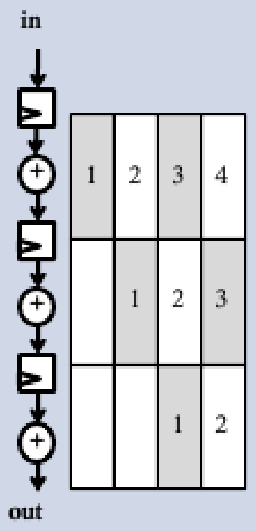
aleksei.rostov@protonmail.com

High-Level Synthesis (HLS)

- HLS is an **automated design process** that transforms a high-level functional specification to a [optimized] register-transfer level (RTL) description suitable for hardware implementation
- HLS tools are widely used for complex ASIC and FPGA design
- Main benefits
  - **Productivity**: lower design complexity and faster simulation speed
  - **Portability**: single source → multiple implementations
  - **Permutability**: rapid design space exploration → higher quality of result (QoR)

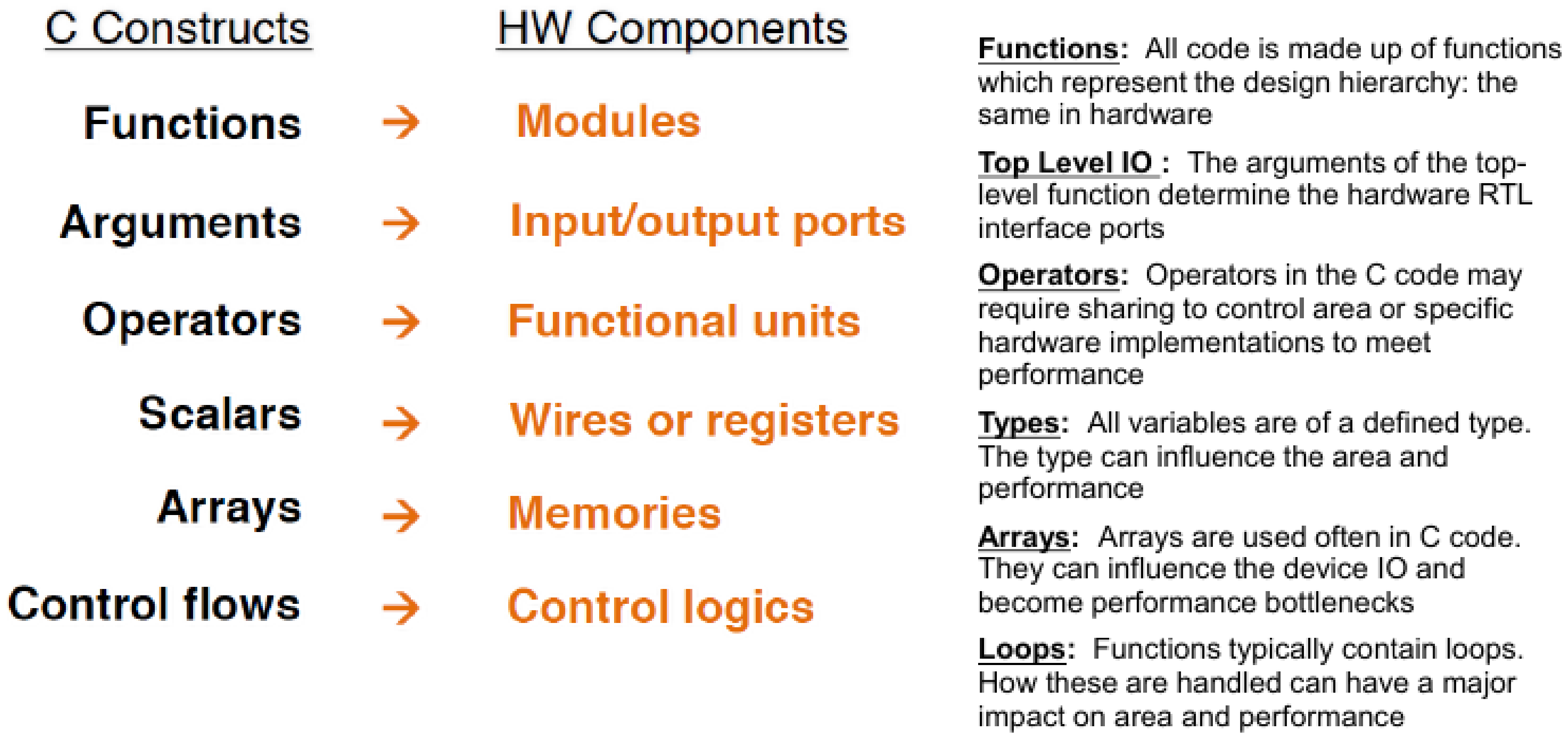
aleksei.rostov@protonmail.com

Permutability: Faster Design Space Exploration

	Latency	Area	Throughput
 **Control-Data Flow Graph**			
$out1 = f(in1, in2, in3, in4)$	$t_{clk} \quad 3 \quad d_{add}$ $T_1 = 1/t_{clk}$ $A_1 = 3 * A_{add}$	$t_{clk} \approx d_{add} + d_{setup}$ $T_2 = 1/(3 * t_{clk})$ $A_2 = A_{add} + 2 * A_{reg}$	$t_{clk} \quad d_{add} + d_{setup}$ $T_3 = 1/t_{clk}$ $A_3 = 3 * A_{add} + 6 * A_{reg}$
Untimed	**Combinational**	**Sequential**	**Pipelined**

aleksei.rostov@protonmail.com

Mapping of C/C++ constructs to RTL



C Constructs		HW Components
Functions	→	Modules
Arguments	→	Input/output ports
Operators	→	Functional units
Scalars	→	Wires or registers
Arrays	→	Memories
Control flows	→	Control logics

Functions: All code is made up of functions which represent the design hierarchy: the same in hardware

Top Level IO : The arguments of the top-level function determine the hardware RTL interface ports

Operators: Operators in the C code may require sharing to control area or specific hardware implementations to meet performance

Types: All variables are of a defined type. The type can influence the area and performance

Arrays: Arrays are used often in C code. They can influence the device IO and become performance bottlenecks

Loops: Functions typically contain loops. How these are handled can have a major impact on area and performance

aleksei.rostov@protonmail.com

Functions & RTL hierarchy

- Each function is translated into an RTL block
  - Verilog module, VHDL entity

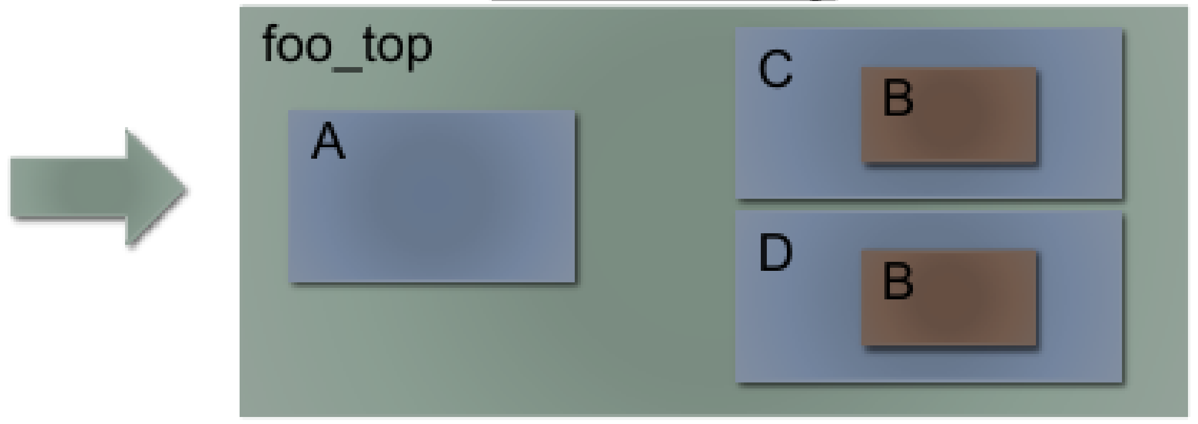
Source Code

```
void A() { ..body A..}
void B() { ..body B..}
void C() {
        B();
}
void D() {
        B();
}


void foo_top() {
        A(...);
        C(...);
        D(...)
}
```

my_code.c

RTL hierarchy



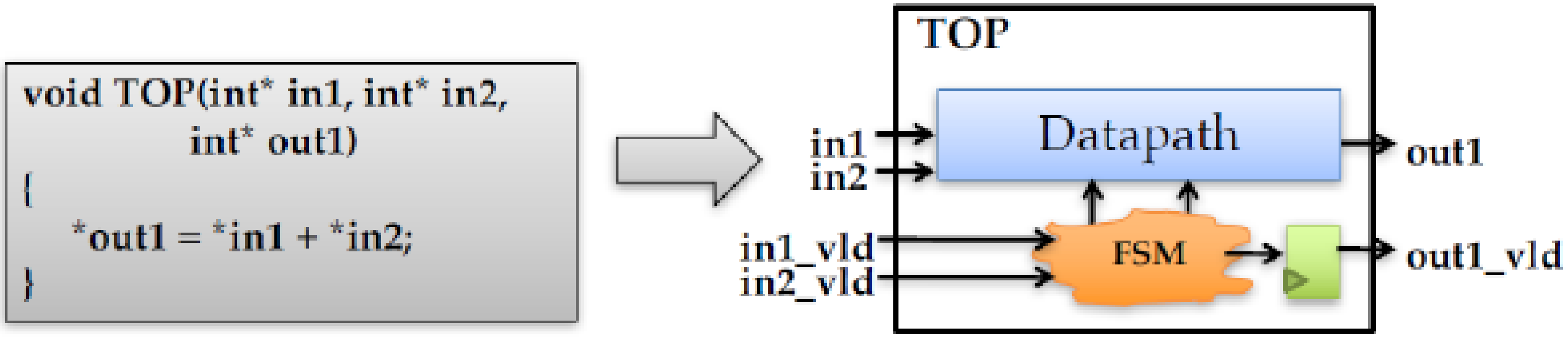
Each function/block can be shared like any other component (add, sub, etc) provided it's not in use at the same time

- By default, each function is implemented using a common instance
- Functions may be inlined to dissolve their hierarchy
  - Small functions may be automatically inlined

aleksei.rostov@protonmail.com

Function arguments

- Function arguments become ports on the RTL blocks
  - Additional control ports are added to the design



```
void TOP(int* in1, int* in2,
              int* out1)
{
    *out1 = *in1 + *in2;
}
```

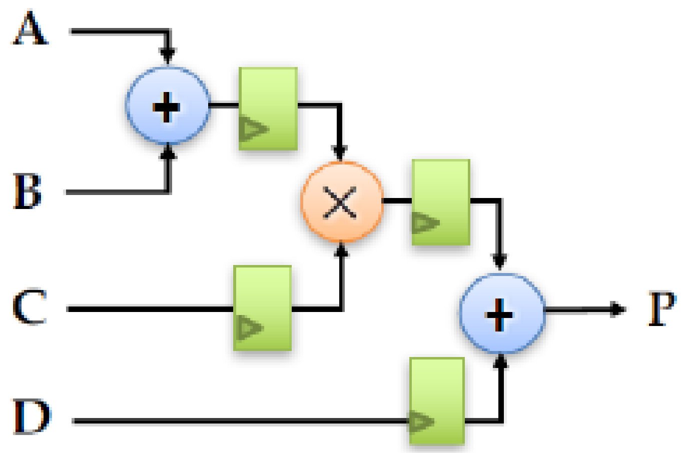
- Input/output (I/O) protocols
  - They allow RTL blocks to synchronize data exchange

Expressions

- HLS generates datapath circuits mostly from expressions
  - Timing constraints influence the use of registers

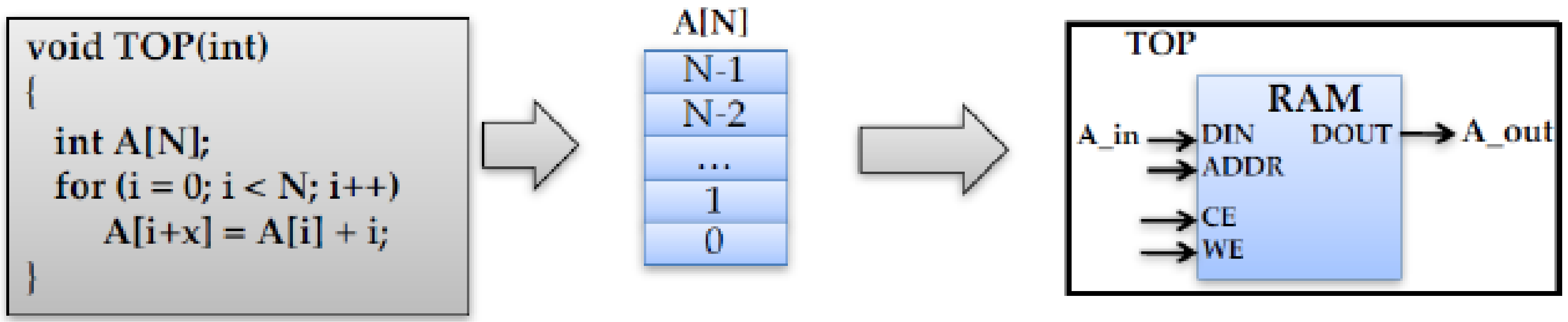
```
char A, B, C, D,
int P;

P = (A+B)*C+D
```



Arrays

- By default, an array in C code is typically implemented by a memory block in the RTL
  - Read & write array → RAM; Constant array → ROM

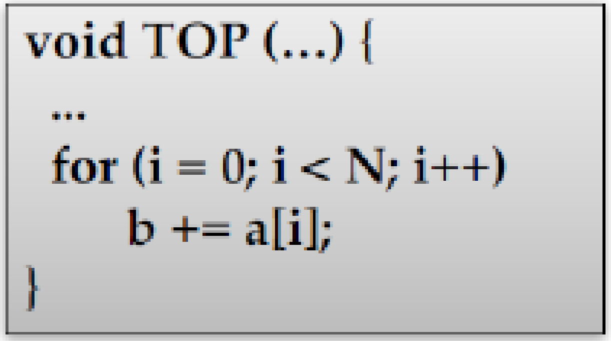


```
void TOP(int)
{
  int A[N];
  for (i = 0; i < N; i++)
      A[i+x] = A[i] + i;
}
```

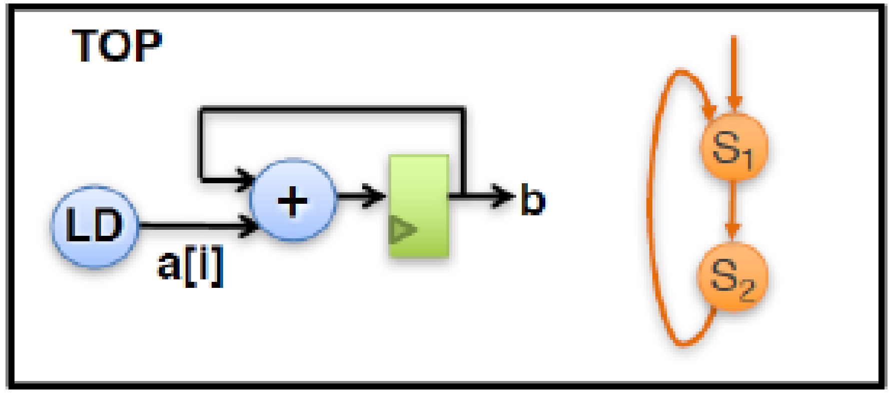
- An array can be partitioned and map to multiple RAMs
- Multiples arrays can be merged and map to one RAM
- An array can be partitioned into individual elements and mapped to registers

Loops

- By default, loop iterations are executed in order
  - Each loop iteration corresponds to a “sequence” of states (possibly a DAG)
  - This state sequence will be repeated multiple times based on the **loop trip count**



```
void TOP (...) {
  ...
  for (i = 0; i < N; i++)
        b += a[i];
}
```



aleksei.rostov@protonmail.com

Loop unrolling

- Loop unrolling to expose **higher parallelism** and achieve shorter latency

- **Pros**
  - Decrease loop overhead
  - Increase **parallelism** for scheduling
  - Facilitate constant propagation and array-to-scalar promotion
- **Cons**
  - Increase operation count, which may negatively impact *area*, *power*, and *timing*

```
for (int i = 0; i < N; i++)
  A[i] = C[i] + D[i];
```

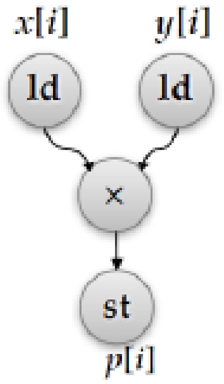
↓

```
A[0] = C[0] + D[0];
A[1] = C[1] + D[1];
A[2] = C[2] + D[2];
   .....
```

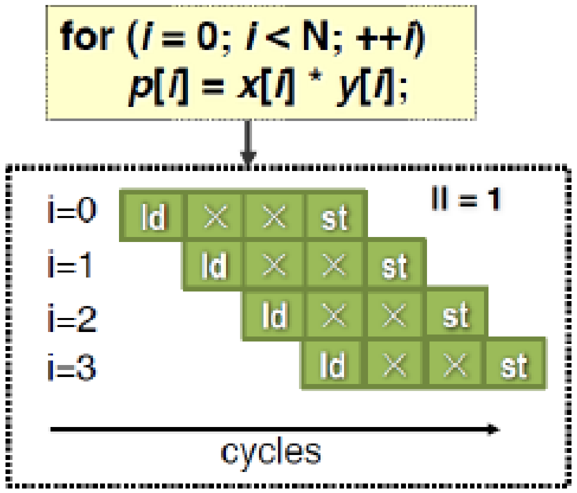
aleksei.rostov@protonmail.com

Loop pipelining

- Loop pipelining is one of the most important optimizations for high-level synthesis
  - Allows a new iteration to begin processing before the previous iteration is complete
  - Key metric: **Initiation Interval (II)** expressed in number of cycles



ld – Load
st – Store



```
for (i = 0; i < N; ++i)
    p[i] = x[i] * y[i];
```

aleksei.rostov@protonmail.com

Typical C/C++ Synthesizable Subset

- Data types:
  - Primitive types: (u)char, (u)short , (u)int, (u)long, float, double
  - Arbitrary precision integer or fixed-point types
  - Composite types: array, struct, class
  - Templated types: template<>
  - Statically determinable pointers
- No support for dynamic memory allocations
- No support for recursive function calls