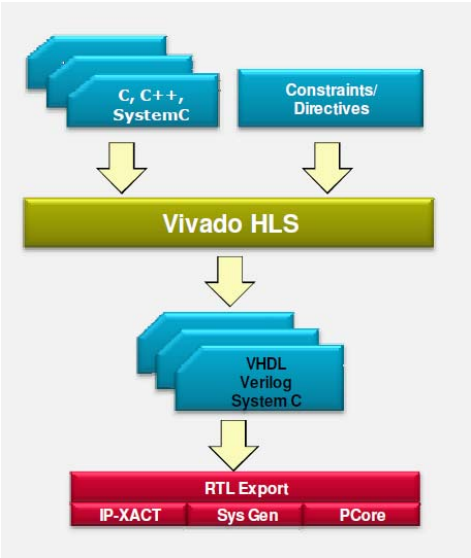
High Level Synthesis
Xilinx Vivado

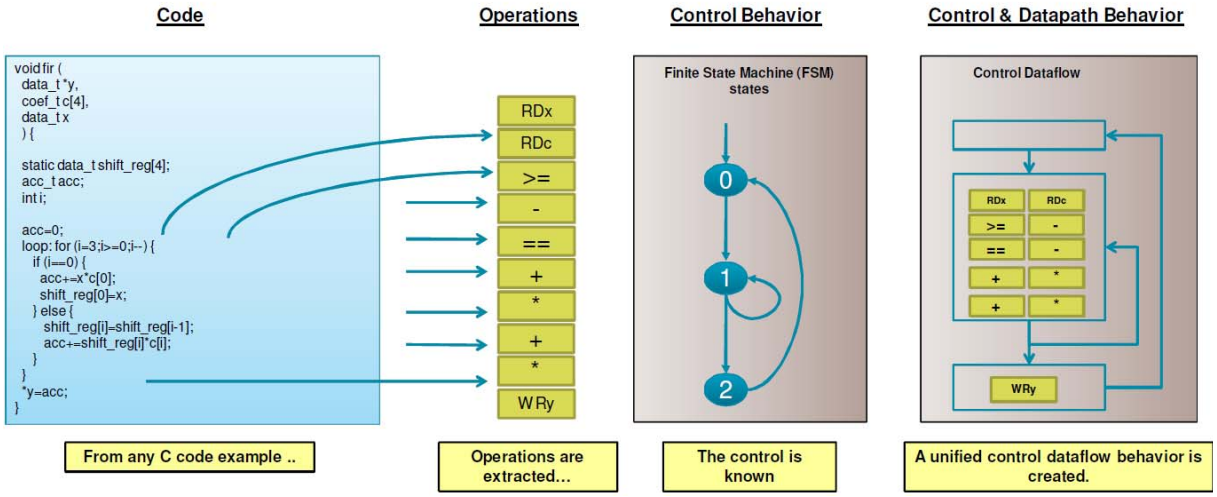
Elías Todorovich
etodorov@exa.unicen.edu.ar
July 2013

Vivado: Introduction

- High-Level Synthesis
  - Creates an RTL implementation from C/C++/SystemC source code
  - Extracts control and dataflow from the source code
  - Implements the design based on defaults and user applied directives
- Many implementation are possible from the same source description
  - Smaller designs, faster designs…
  - Enables design exploration.

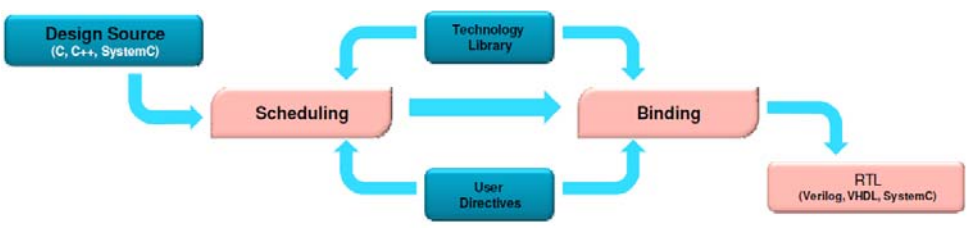


Control and datapath extraction



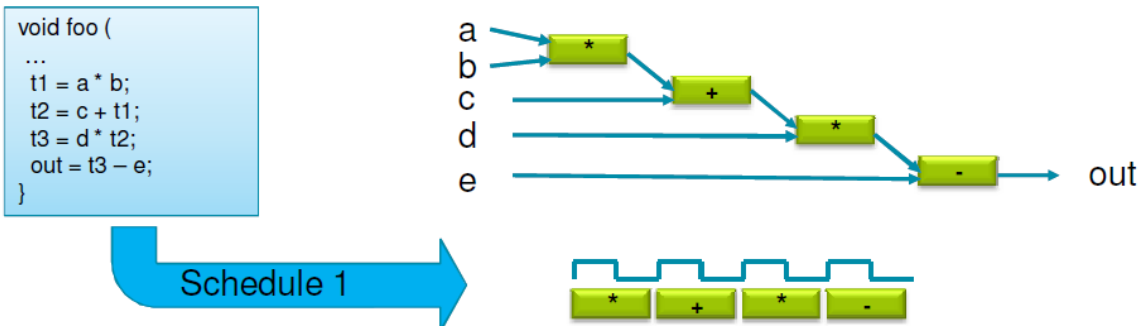
HLS: Scheduling & Binding

- Scheduling determines in which clock cycle an operation will occur
  - Takes into account the control, dataflow and user directives
  - The allocation of resources can be constrained
- Binding determines which library cell is used for each operation
  - Takes into account component delays, user directives



Scheduling

- The operations in the control flow graph are mapped into clock cycles



Binding

- Binding is where operations are mapped to cores from the hardware library.
- To share or not to share
  - Binding may decide to share (muxing) the multipliers (each is used in a different cycle)

Key attributes of C code

```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop:for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      shift_reg[i]=shift_reg[i-1];
      acc+=shift_reg[i] * c[i];
}
  }
  *y=acc;
}
```

- **Functions:** All code is made up of functions which represent the design hierarchy
- **Top Level IO :** The arguments of the top-level function determine the hardware RTL interface ports
- **Types:** The type can influence the area and performance
- **Loops:** Functions typically contain loops. How these are handled can have a major impact on area and performance
- **Arrays:** They can influence the device IO and become performance bottlenecks
- **Operators:** Operators in the C code may require sharing to control area or specific hardware implementations to meet performance

Functions modularity vs. RTL hierarchy

- Each function is translated into an RTL block
  - But functions may be inlined to dissolve their hierarchy

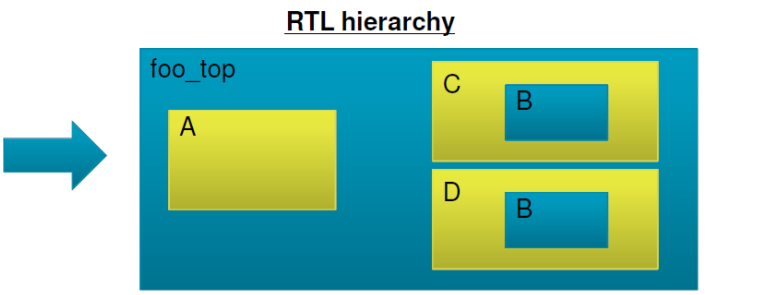
Source Code

```
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



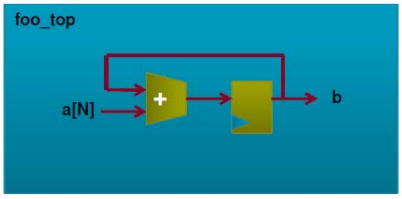
Each function/block can be shared like any other component (add, sub, etc) provided it's not in use at the same time

Loops in HLS

- By default, loops are rolled. Each C loop iteration is implemented
  - in the same state
  - with same resources


```
void foo_top (…) {
  ...
  Add: for (i=3;i>=0;i--) {
      b = a[i] + b;
  ...
  }
```

Loops require labels if they are to be referenced by Tcl directives
(GUI will auto-add labels)



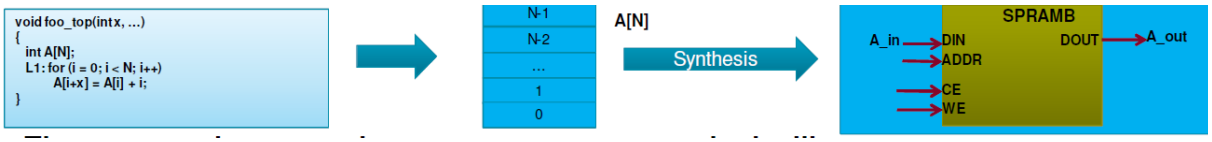
- Loops can be unrolled if their indices are statically determinable at elaboration time

Loop unrolling

- With the loop unrolled (completely)
  - The dependency on loop iterations is gone
  - Operations can now occur in parallel
    - If data dependencies allow
    - If operator timing allows
  - Design finished faster but uses more operators

Arrays in HLS

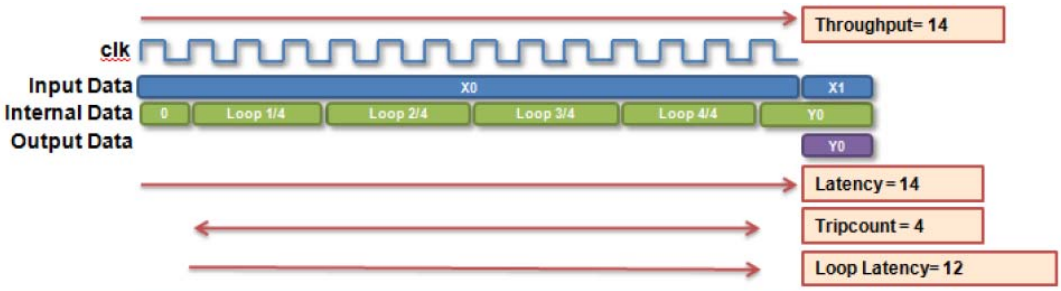
- An array in C code is implemented by a memory in the RTL
  - By default, arrays are implemented as RAMs
- Arrays can be
  - merged with other arrays and reconfigured or
  - partitioned into individual elements



Function arguments in HLS

- Top-level function arguments
  - All top-level function arguments have a default hardware port type
- When an array is an argument of the top-level function
  - The array/RAM is “off-chip”
  - The type of memory resource determines the top-level IO ports
  - Arrays on the interface can be mapped & partitioned
    - E.g. partitioned into separate ports for each element in the array
  - Array optimization impacts on schedule

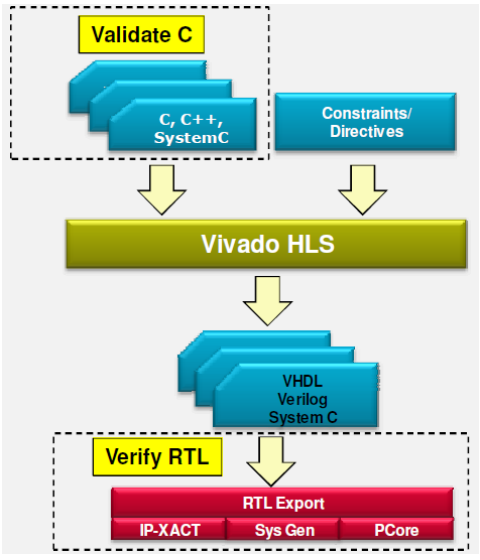
Vivado HLS Terminology



Latency	The number of cycles from input to output (final output of an array write)	14 cycles
Throughput	The number of cycle between new input samples (in this example it must wait for all operations to complete before it can read a new input)	14 cycles
Initiation Interval (II)	The number of cycles between new inputs to a pipeline (the same as throughput, but this term is used with pipelines).	Not shown in this example.
Data Rate	The 1/throughout * clock frequency	10ns clock => 7.14 Mhz, ((1/10e9)*14)
Trip count	The number of iterations in a loop	4
Loop Latency	The latency of the entire loop (divide by tripcount to get the latency for each loop iteration)	12 cycles

Functional Verification

- C validation
  - A HUGE reason users want to use HLS
  - Fast, free verification
  - Validate the algorithm is correct before synthesis
  - Follow the test bench tips given over
- RTL Verification
  - Vivado HLS can co-simulate the RTL with the original test bench



Functional Verification

- The test bench is the level above the function
  - Usually the main() function is above the function to be synthesized
- Good Practices
  - The test bench should compare the results with golden data
  - Automatically confirms any changes to the C are validated and verifies the RTL is correct
  - The test bench should return a 0 if the self-checking is correct

```
int main () {
  int ret=0;
  ...
  ret = system("diff --brief  -w output.dat output.golden.dat");
  if (ret != 0) {
      printf("Test failed  !!!\n");
      ret=1;
  } else {
      printf("Test passed !\n");
  }
  ...
  return ret;
}
```


High Level Synthesis
Xilinx Vivado

Elías Todorovich
etodorov@exa.unicen.edu.ar
July 2013