CORE | Vision
NEXT LEVEL | EMBEDDED DEVELOPMENT

High-Level Synthesis for FPGA Designs

Frank de Bont
Trainer consultant

Cereslaan 10b
5384 VT Heesch
✆ +31 (0)412 660088
✉ info@core-vision.nl
www.core-vision.nl

Agenda

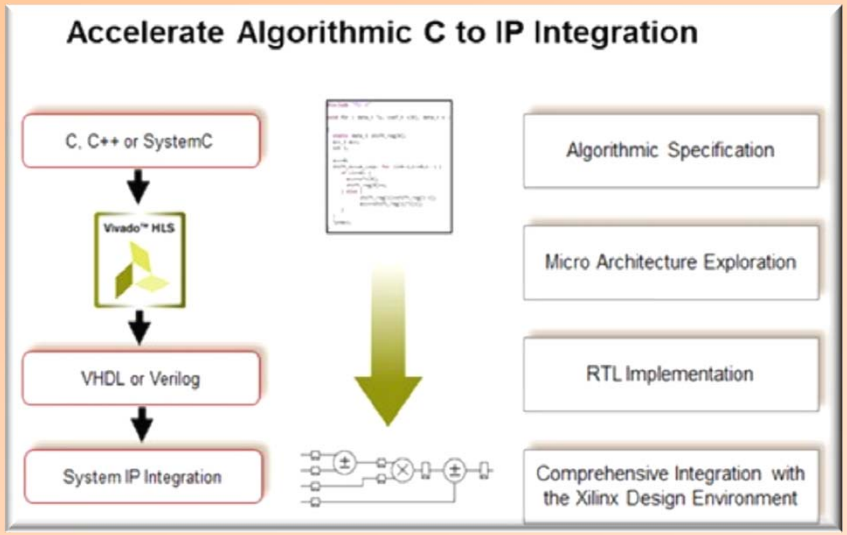
- Need for High-Level Synthesis
- High-Level Synthesis
- System Integration
- Design Exploration
- High-Level Synthesis Flow
- Control & Datapath Extraction
- Scheduling and Binding
- Example FIR C-code
- Who is Core|Vision

Need for High-Level Synthesis

- **Algorithmic-based approaches are popular due to accelerated design time and time-to-market pressures**
  - Larger designs pose challenges in design and verification of hardware
- **Industry trend is moving towards hardware acceleration to enhance performance and productivity**
  - CPU-intensive tasks are now offloaded to hardware accelerator
    Hardware accelerators require a lot of time to understand and design
- **Vivado HLS tool converts algorithmic description written in C-based design flow into hardware description (RTL)**
  - Elevates the abstraction level from RTL to algorithms
- **High-level synthesis is essential for maintaining design productivity for large designs**

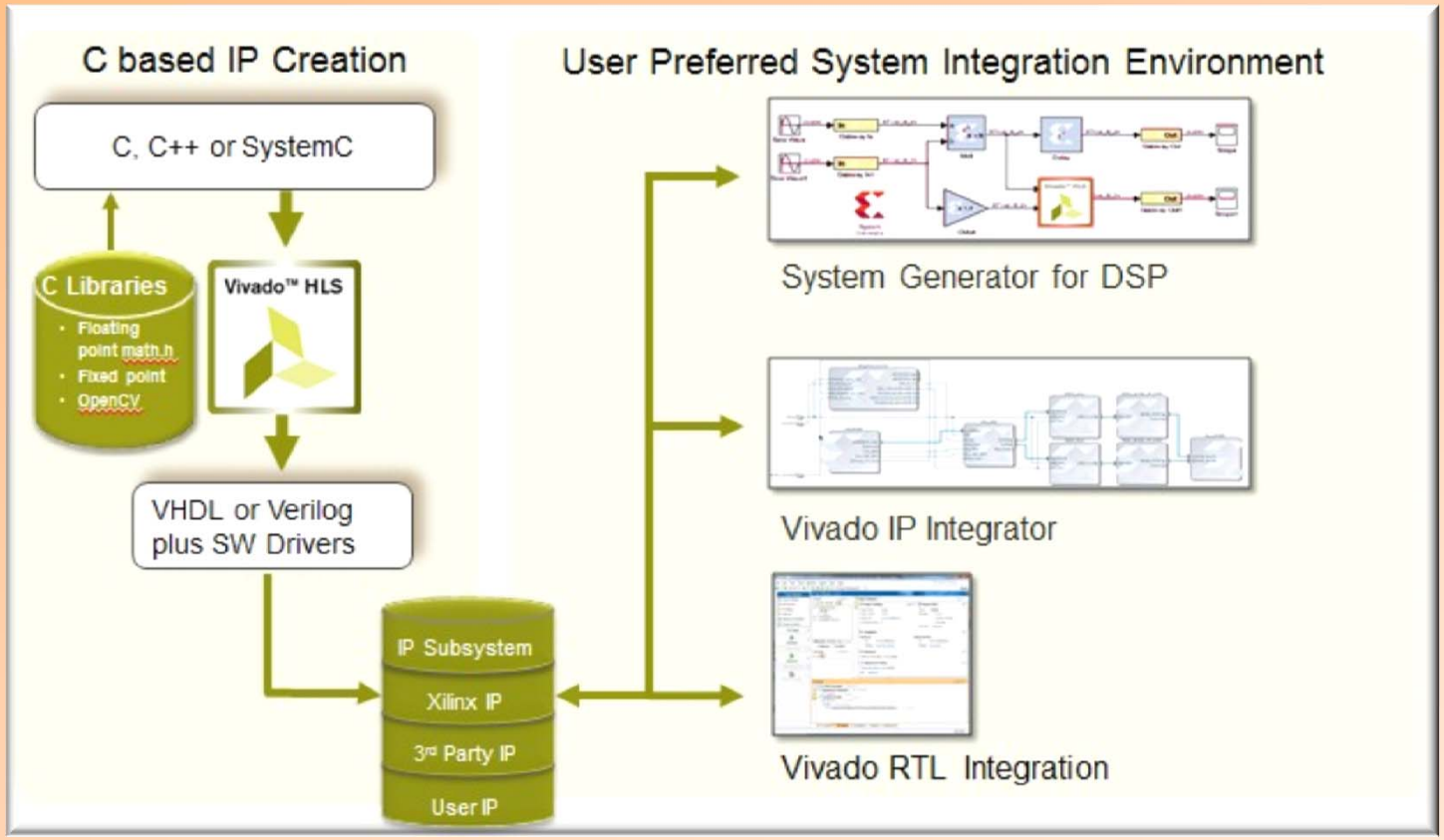
High-Level Synthesis: HLS

- **High-level synthesis**
  - Creates an RTL implementation from source code
  - C, C++, SystemC
  - Coding style impacts hardware realization
  - Limitations on certain constructs and access to libraries

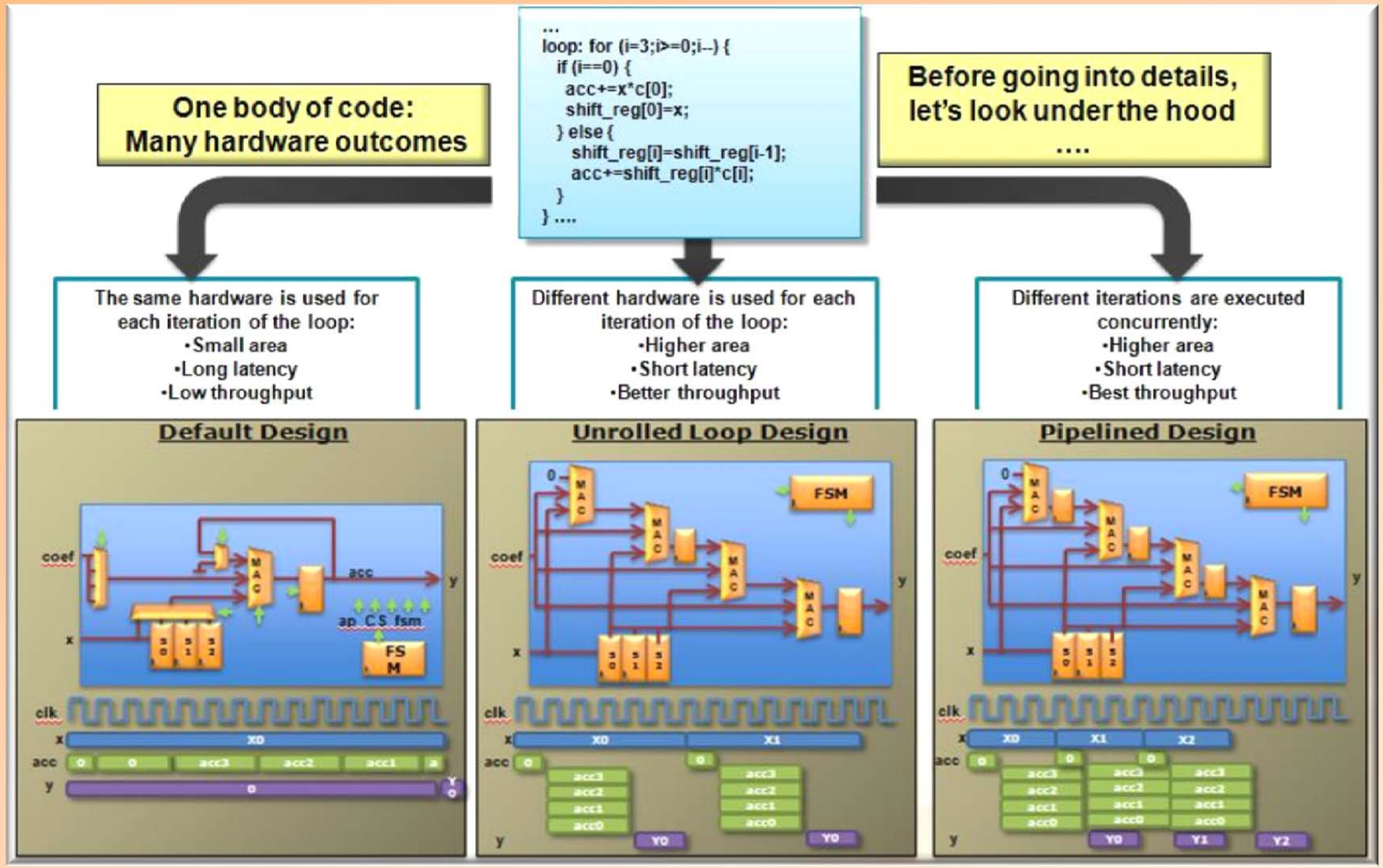


  - Extracts control and dataflow from the source code
  - Implements the design based on defaults and user-applied directives
- **Many implementations are possible from the same source description**
  - Smaller designs, faster designs, optimal designs
- **Enables design exploration**

System Integration



Design Exploration



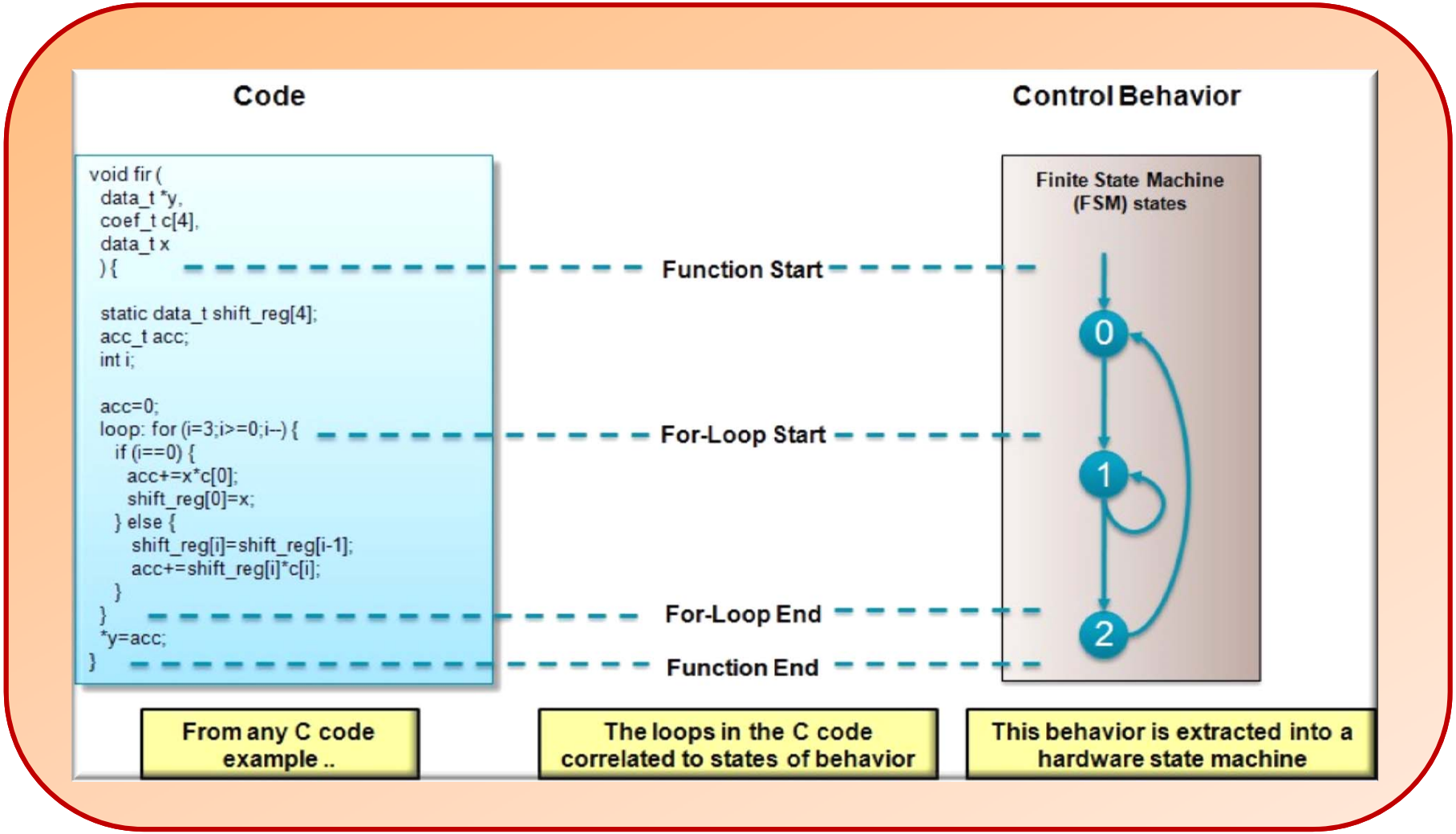
High-Level Synthesis flow

- **Hardware extraction from C code**
- **Control and datapath can be extracted from C code at the top level**
- **Same principles used in the example can be applied to sub-functions**
  - At some point in the top-level control flow, control is passed to a sub-function
  - Sub-function can be implemented to execute concurrently with the top level and or other sub-functions

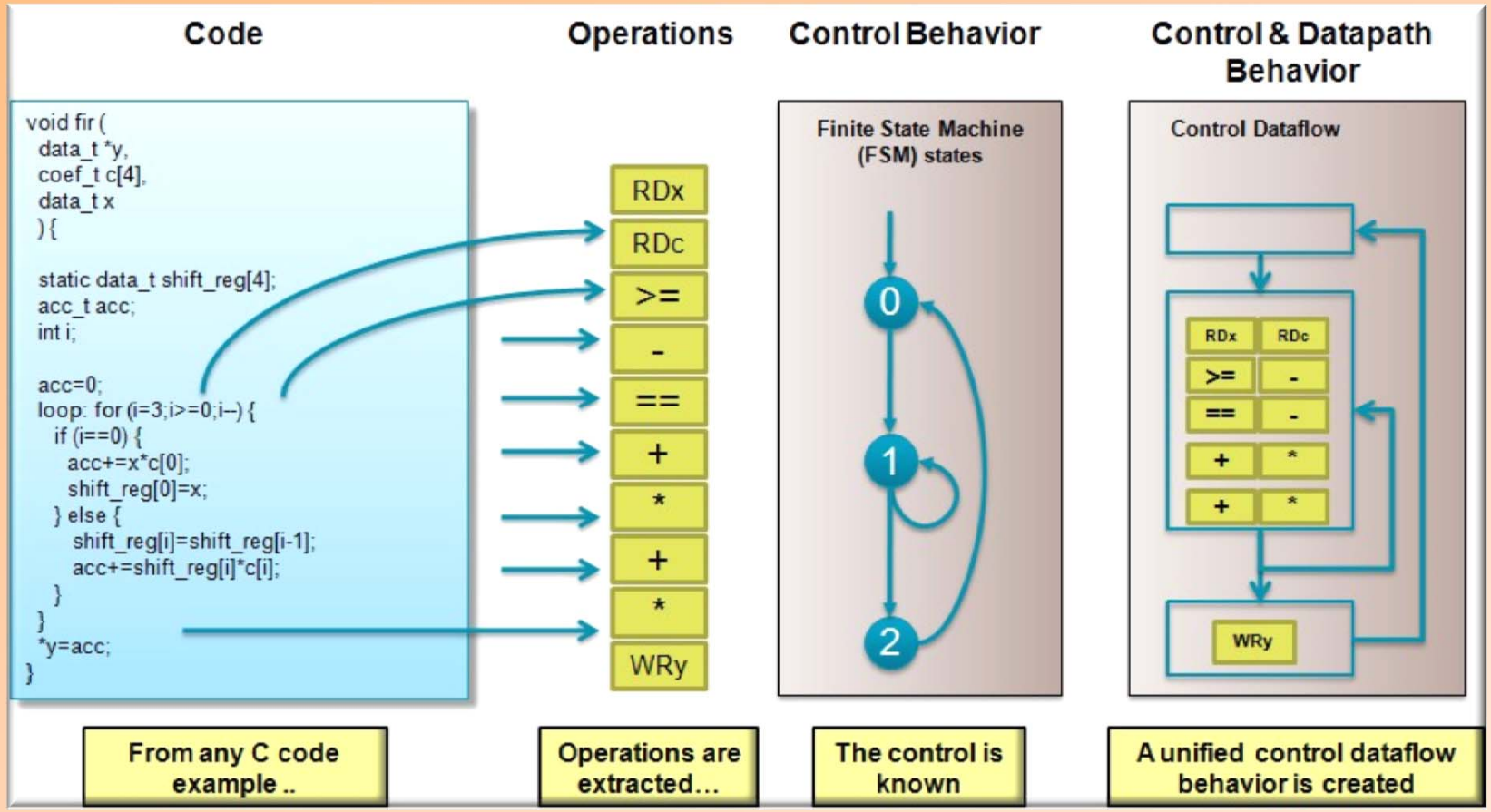
High-Level Synthesis Flow cont

- **Scheduling and binding processes create hardware design from control flow graph considering the constraints and directives**
  - Scheduling process maps the operations into cycles
  - Binding process determines which hardware resource, or core, is used for each operation
  - Binding decisions are considered during scheduling because the decisions in the binding process can influence the scheduling of operations
  - For example, using a pipelined multiplier instead of a standard combinational multiplier

HLS: Control Extraction

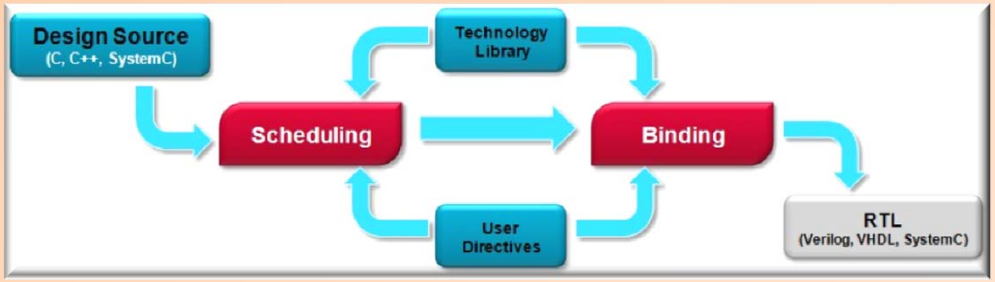


Control and Datapath Extraction



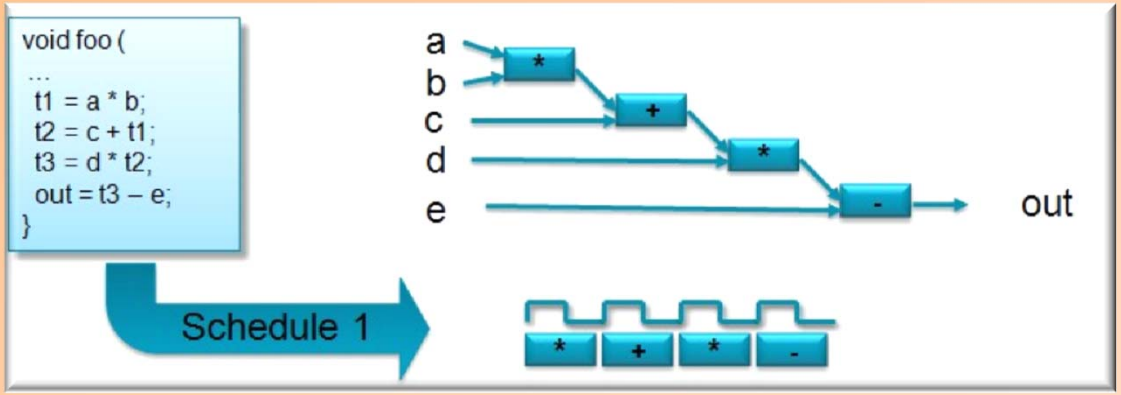
Scheduling and Binding

- **Scheduling and binding**
  - Heart of HLS
- **Scheduling determines in which clock cycle an operation will occur**
  - Takes into account the control, dataflow, and user directives
    Allocation of resources can be constrained (discussed in detail later)
- **Binding determines which library cell is used for each operation**
  - Takes into account component delays and user directives



Scheduling

- **Operations in the control flow graph are mapped into clock cycles**



- **Technology and user constraints impact the schedule**
  - Faster technology (or slower clock) can allow more operations to occur in the same clock cycle

- **Code also impacts the schedule**
  - Code implications and data dependencies must be obeyed

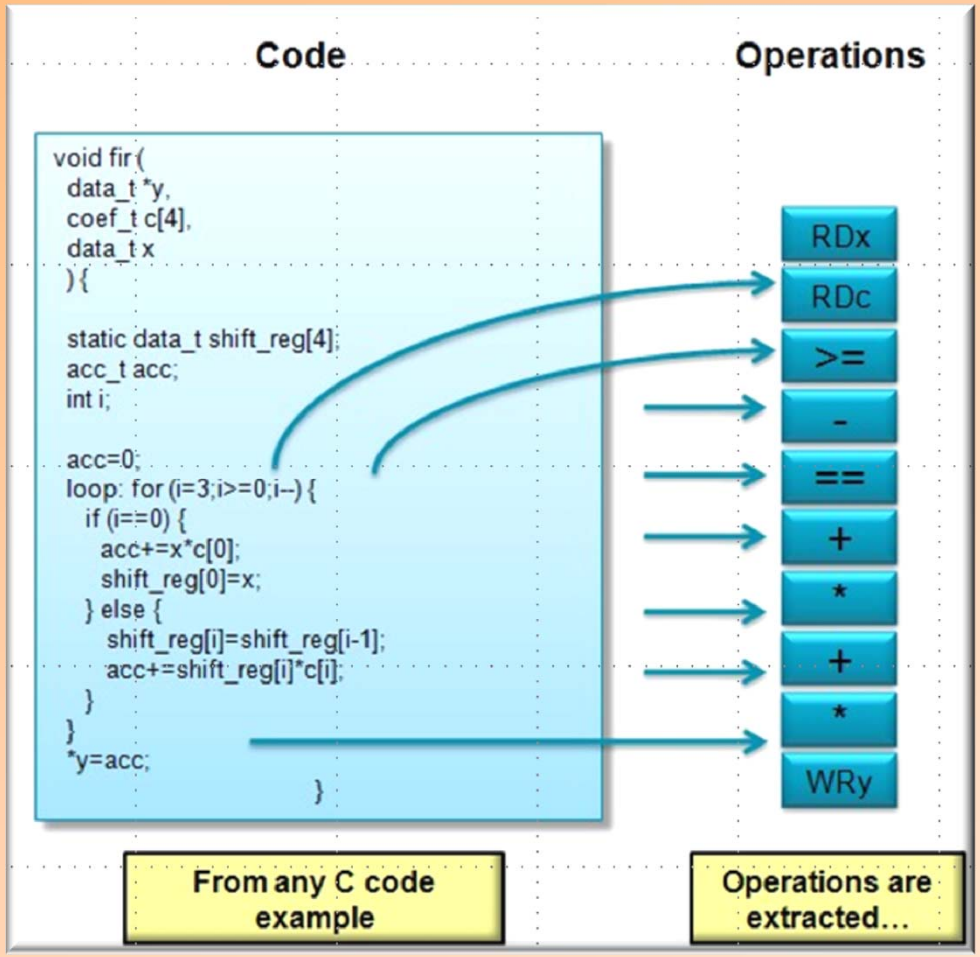
Binding

- **Binding is where operations are mapped to hardware**
  - Operators extracted from the C code are mapped to RTL cores
- **Binding decision: to share**
  - Given the following schedule

  - Binding must use two multipliers because both are in the same cycle
  - It can decide to use an adder *and* **subtractor** *or* one **addsub**
- **Binding decision: or not to share**
  - Given the following schedule

  - Binding may decide to share the multipliers (each is used in a different cycle)
  - Or it may decide the cost of sharing (MUXing) would impact timing and it may decide not to share them
  - It may make this same decision in the first example above as well

Example FIR C-Code

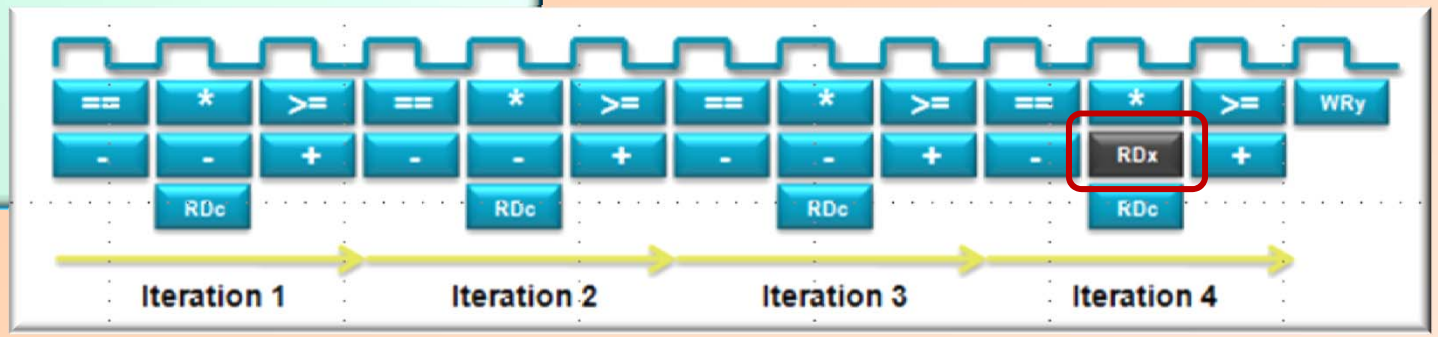


- **By default, loops are rolled**
  - Each C loop iteration implemented in the same state
  - Each C loop iteration implemented with same resources
- **Loops can be unrolled if their indices are statically determinable at elaboration time**

Example FIR C-Code cont

```
void fir (
...
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;
}
```

- **Read on port X can occur anywhere from the start to iteration 4**
  - Only constraint on **RDx** is that it occur before the final multiplication
- **There are no advantages to reading any earlier (unless you want it registered)**
- **However, the final multiplication is very constrained…**



Example FIR C-Code cont

- **Schedule after loop optimization**
  - With the loop unrolled (partial / full)
    - Dependeny on loop iterations is gone
    - Operations can occur in parallel
    - Design finished faster but more operators
    - Two multipliers and two adders

- **Schedule after array optimization**
  - With the existing code and defaults
    - Port C is by default dual port RAM
    - Allows two reads per clock cycle
    - Max number of simultaneous reads and writes


```
void fir (
...
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;
}
```

Example FIR C-Code cont

- **With the C port partitioned into (4) separate ports**
  - All reads and multiply can occur in one cycle
- **If the timing allows**
  - The additions can also occur in the same cycle
  - The write can be performed in the same cycles
  - Optionally the port reads and writes could be registered
- **This solution uses much more hardware resources in only one clock cycle**

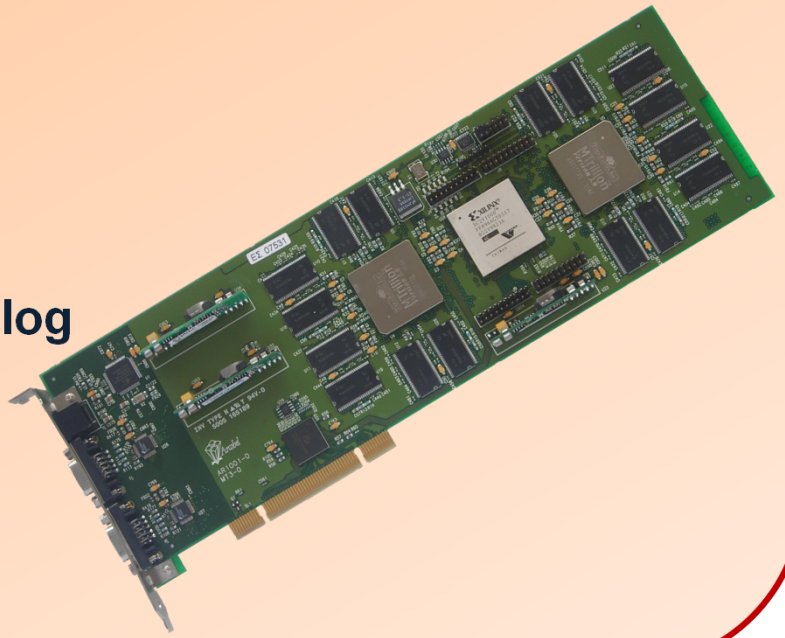
```
void fir (
...
acc = 0;
  loop: for ( i=3; i>= 0; i--){
    if (i==0){
      acc +=x*c[0];
      shift_reg[0] = x;
    } else {
      shift_reg[i] = shift_reg[i-1];
      acc += shift_reg[i]*c[i];
    }
  }
*y = acc;
}
```

Core|Vision

Our competences

Core|Vision has more than 100 man years of design experience in hard- and software development. Our competence areas are:

- System Design
- FPGA Design
- Consultancy / Training
- Digital Signal Processing
- Embedded Real-time Software
- App development, IOS Android
- Data Acquisition, digital and analog
- Modeling & Simulation
- ASIC Conversion & Prototyping
- PCB design & Layout
- Doulos & Xilinx Training Partner



CORE | Vision
NEXT LEVEL | EMBEDDED DEVELOPMENT

Cereslaan 10b
5384 VT Heesch
✆ +31 (0)412 660088

www.*core-vision*.nl

Email : info@*core-vision*.nl

Training Program

Essentials of FPGA Design	*1 day*
Designing for Performance	*2 days*
Advanced FPGA Implementation	*2 days*
Design Techniques for Lower Cost	*1 day*
Designing with Spartan-6 and Virtex-6 Family	*3 days*
Essential Design with the PlanAhead Analysis Tool	*1 day*
Advanced Design with the PlanAhead Analysis Tool	*2 days*
Xilinx Partial Reconfiguration Tools and Techniques	*2 days*
Designing with the 7 Series Families	*2 days*

Training Program

Vivado Essentials of FPGA Design	*2 days*
Vivado Design Suite Tool Flow	*1 day*
Vivado Design Suite for ISE Users	*1 day*
Vivado Avanced XDC and STA for ISE Users	*2 days*
Vivado Advanced Tools & Techniques	*2 days*
Vivado Static Timing Analysis and XDC	*2 days*
Debugging Techniques Using Vivado Logic Analyzer	*1 day*
Essential Tcl Scripting for Vivado Design Suite	*1 day*
Vivado FPGA Design Methodology	*1 day*

Training Program

➢ Designing with Multi Gigabit Serial IO	*3 days*
➢ High Level Synthesis with Vivado	*2 days*
➢ C-Based HLS Coding for Hardware Designers	*1 day*
➢ C-Based HLS Coding for Software Designers	*1 day*
➢ DSP Design Using System Generator	*2 days*
➢ Essential DSP Implementation Techniques for Xilinx FPGAs	*2 days*

Training Program

- Embedded Systems Development *2 days*
- Embedded Systems Software Development *2 days*
- Advanced Features and Techniques of SDK *2 days*
- Advanced Features and Techniques of EDK *2 days*
- Zynq All Programmable SoC Systems Archicture *2 days*
- C Language Programming with SDK *2 days*

Training Program

- VHDL Design for FPGA — *3 days*
- Advanced VDHL — *2 days*
- Comprehensive VHDL — *5 days*
- Exprt VHDL Verification — *3 days*
- Expert VDHL Design — *2 days*
- Expert VHDL — *5 days*
- Essential Digital Design Techniques — *2 days*