High Level Synthesis - from dream to reality

Bram Riemens
System Architect
040coders, Nov. 15th, 2018

Part of the StreamTV- SeeCubic Team

SeeCubic

High Level Synthesis

Back in the 70s…

- High level procedural languages: Algol, Pascal, C, …
- IC design … layout drawing tooling emerged

The dream

- What if we could “compile” for an ASIC?

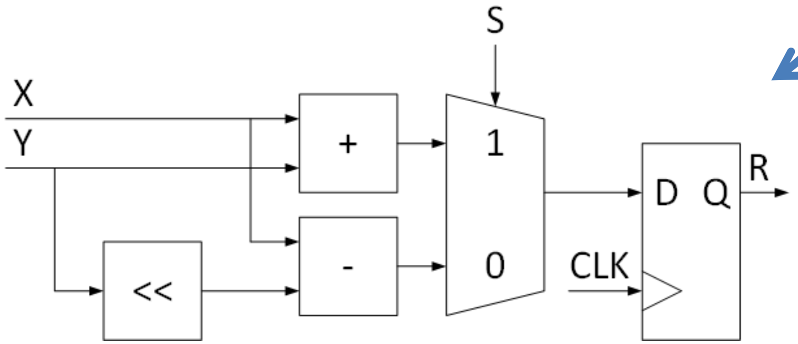
So, … let us consider compiling C++ code to hardware

Software vs. Hardware - fundamentals

- Software mental model
  - Sequential
- Hardware
  - Inherently parallel

```c
if (s) {
    r = x + y;
} else {
    r = x - 2*y;
}
```

```asm
    cmpl   $0, -20(%rbp)
    je     .L2
    movl   -8(%rbp), %eax
    movl   -4(%rbp), %edx
    addl   %edx, %eax
    movl   %eax, -12(%rbp)
    jmp    .L1
.L2:
    movl   -8(%rbp), %edx
    movl   $0, %eax
    subl   %edx, %eax
    addl   %eax, %eax
    movl   %eax, %edx
    movl   -4(%rbp), %eax
    addl   %edx, %eax
    movl   %eax, -12(%rbp)
.L1:
```



Properties of hardware

- No stack → No recursion
- No dynamic memory allocation
- No standard C library, STL, etc.
- …

- … just computation, buffering, channels
- in any wordwidth (uint10 + uint10 → uint11)

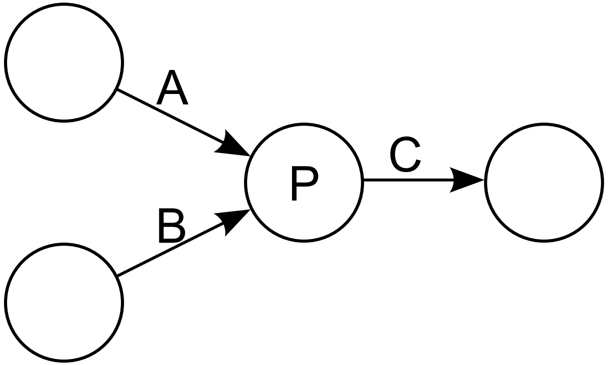
Kahn Process Network

https://en.wikipedia.org/wiki/Kahn_process_networks

Model hardware as set of communicating processes, connected via channels

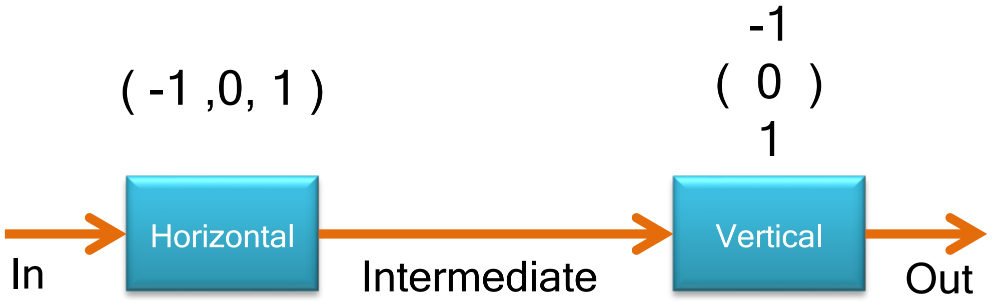
A process can *always write* into a channel, channels are unbound

Each process is repeatedly activated, and consumes input data if available



Kahn, G. (1974). Rosenfeld, Jack L., ed. *The semantics of a simple language for parallel programming* Proc. IFIP Congress on Information Processing. North-Holland.

Example - 2D edge detection filter

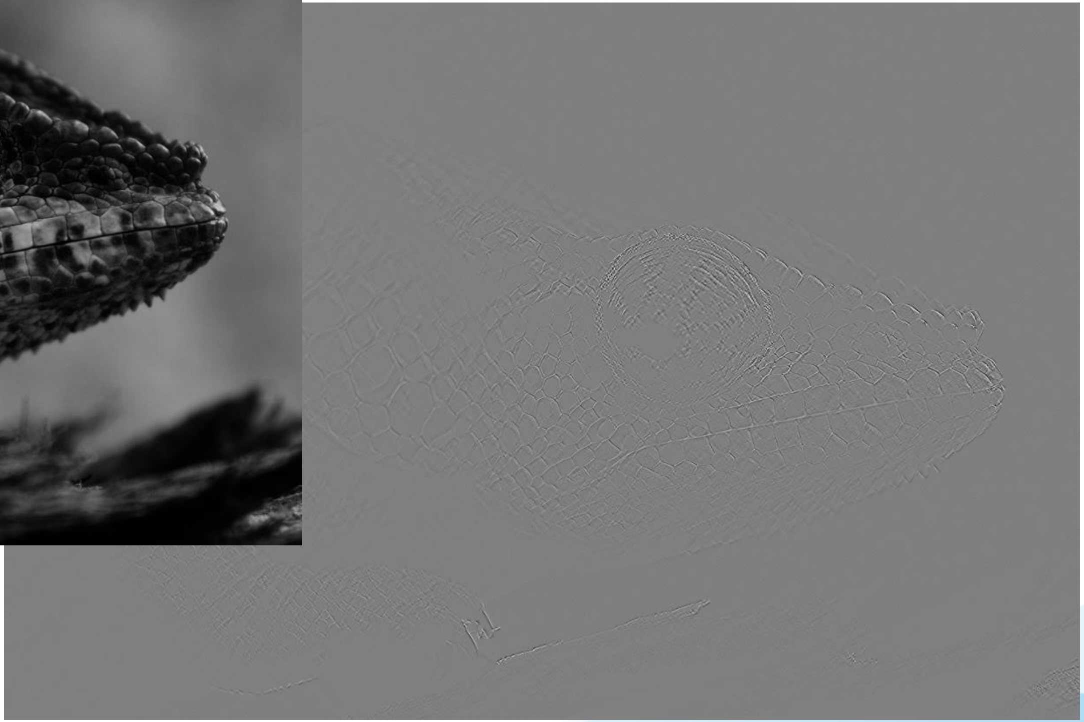


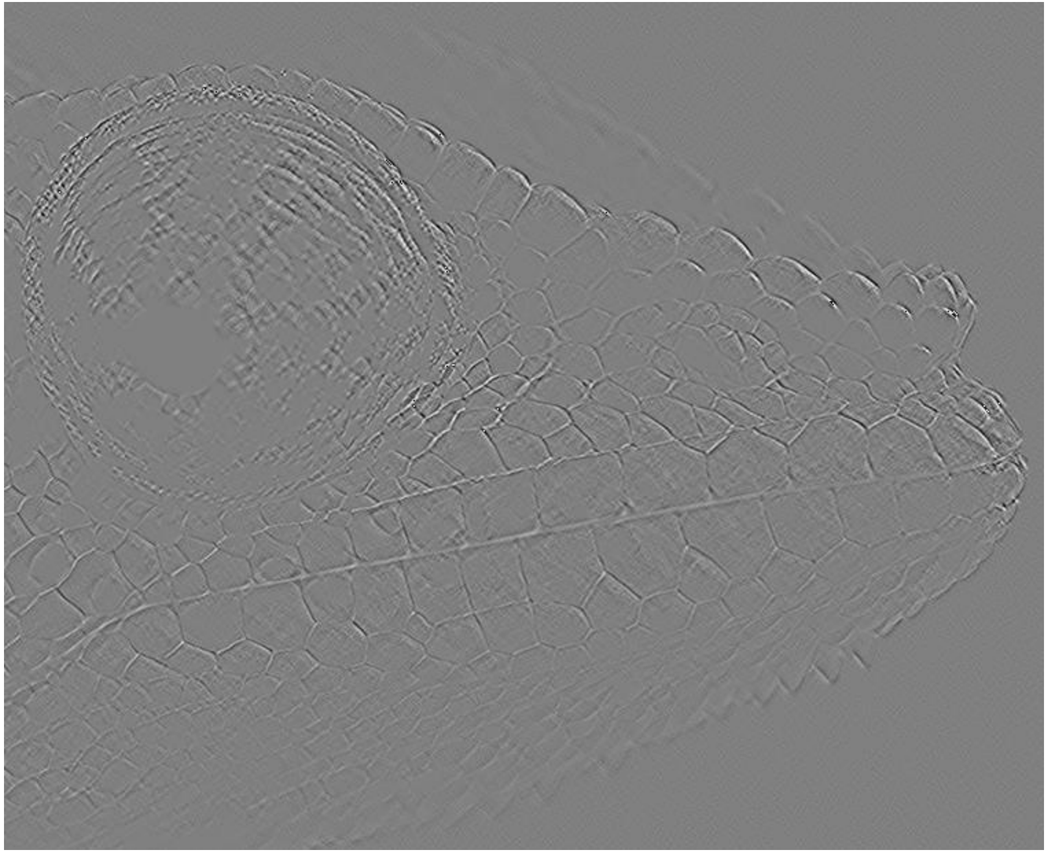
Functional code

```
/* (-1, 0, +1) filter */
static uint8_t filter(uint8_t val_neg, uint8_t val_pos)
{
    return 128 + val_pos/2 - val_neg/2;
} /* filter */

std::vector<uint8_t> edge_func(std::vector<uint8_t> const& in, uint32_t width, uint32_t height)
{
    std::vector<uint8_t> intermediate(width * height, 128);
    std::vector<uint8_t> out(width * height, 128);

    <horizontal filter from "in" to "intermediate">
    <vertical filter from "intermediate" to "out">

    return out;
} /* edge_func */
```

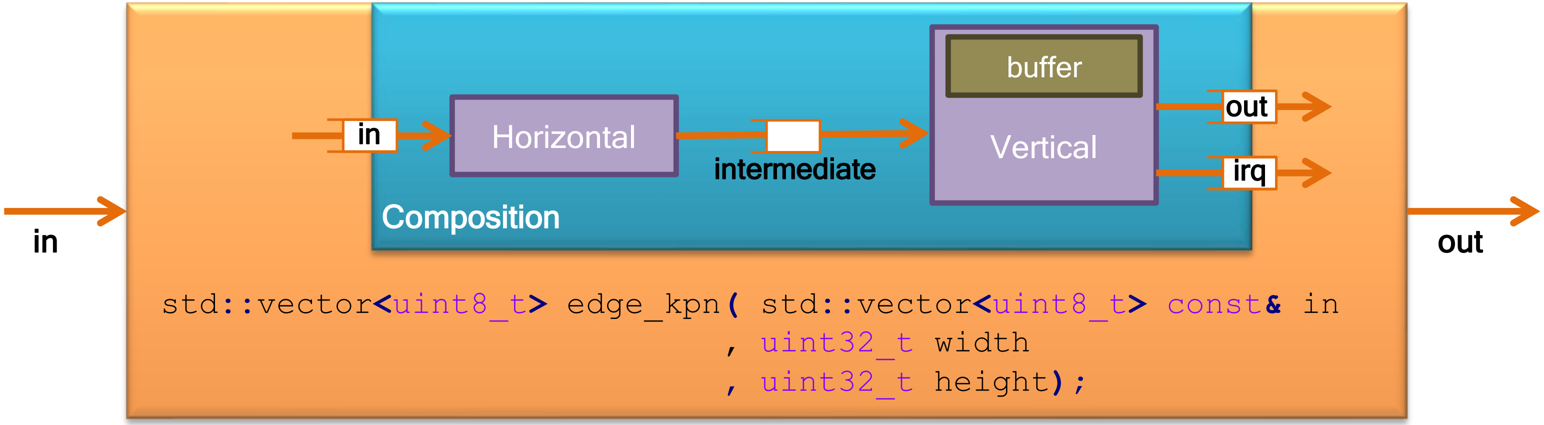
Functional code

```
for (uint32_t py = 0; py < height; py++) {            /* Horizontal filter */
    for (uint32_t px = 1; px < width-1; px++) {
        uint8_t val_left  = in[py*width + px-1];
        uint8_t val_right = in[py*width + px+1];
        intermediate[py*width + px] = filter(val_left, val_right);
    }
}

for (uint32_t py = 1; py < height-1; py++) {          /* Vertical filter */
    for (uint32_t px = 0; px < width; px++) {
        uint8_t val_up   = intermediate[(py-1)*width + px];
        uint8_t val_down = intermediate[(py+1)*width + px];
        out[py*width + px] = filter(val_up, val_down);
    }
}
```





KPN model - top-down design



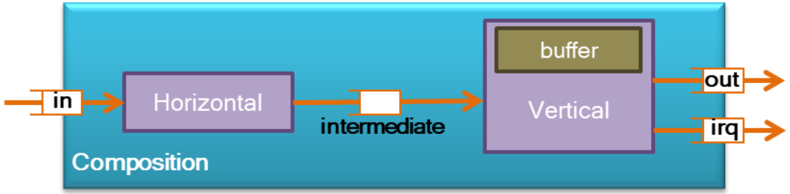
KPN model - types and channel tokens

```
typedef ac_int<11,false> ww_uint11;
typedef ac_int<10,false> ww_uint10;
typedef ac_int<8, false> ww_uint8;
typedef ac_int<7, false> ww_uint7;
typedef ac_int<2, false> ww_uint2;

struct sync_t {
    bool     start_of_frame;
    bool     end_of_frame;
    bool     start_of_line;
    bool     end_of_line;
};

struct pixel_t {
    ww_uint8 value;
    sync_t   sync;
};
```

KPN model - computation - filter kernel

```
/* (-1, 0, +1) filter */
static ww_uint8 filter(ww_uint8 val_neg, ww_uint8 val_pos)
{
    ww_uint7  pos_half = val_pos/2;
    ww_uint7  neg_half = val_neg/2;
    ww_uint10 result = 128 + pos_half - neg_half;
    return static_cast<ww_uint8>(result);
} /* filter */
```

KPN model - horizontal filter

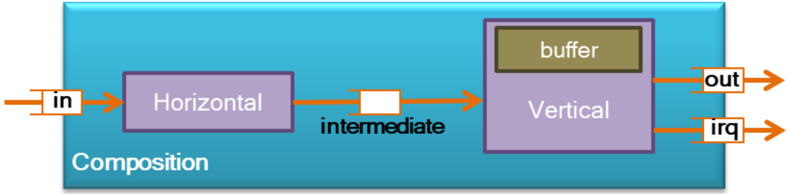



```
class Horizontal_edge
{
    ww_uint8 m_pre_prev_value;
    ww_uint8 m_prev_value;

public:
    Horizontal_edge() : m_pre_prev_value(0), m_prev_value(0) {}

    #pragma hls_design interface
    void CCS_BLOCK(block)( ac_channel<pixel_t>& input_ch
                         , ac_channel<pixel_t>& output_ch)
    {
            . . .
    }
};
```

```
ww_uint8 m_pre_prev_value;
ww_uint8 m_prev_value;
```

```
void CCS_BLOCK(block)( ac_channel<pixel_t>& input_ch
                     , ac_channel<pixel_t>& output_ch)
{
    if (input_ch.available(1)) {
        pixel_t input_value = input_ch.read();

        pixel_t output_value;
        output_value.value = filter(m_pre_prev_value, input_value.value);
        output_value.sync  = input_value.sync;
        output_ch.write(output_value);

        m_pre_prev_value = m_prev_value;
        m_prev_value     = input_value.value;
    }
}
```

KPN model - vertical filter



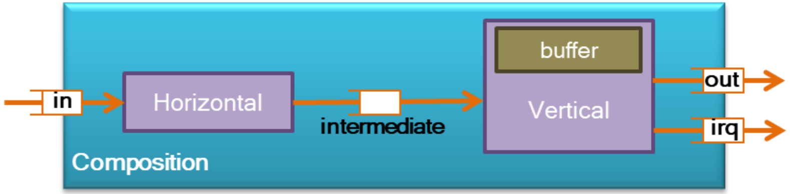

```
void incr_wrap(ww_uint2 &value)
{
    if (value == 2) {
        value = 0;
    } else {
        value++;
    }
}
```

```
class Vertical_edge
{
    ww_uint8  m_buffer[3][2048];
    ww_uint2  m_curr_line_idx;
    ww_uint2  m_preprev_line_idx;
    ww_uint11 m_px;

public:
    Vertical_edge() : m_curr_line_idx(2), m_preprev_line_idx(0), m_px(0) {}

    #pragma hls_design interface
    void CCS_BLOCK(block)( ac_channel<pixel_t>& input_ch
                         , ac_channel<pixel_t>& output_ch
                         , ac_channel<bool>&    irq_ch)
    {
            . . .
    }
};
```

```
void CCS_BLOCK(block)( ac_channel<pixel_t>& input_ch
                     , ac_channel<pixel_t>& output_ch
                     , ac_channel<bool>&    irq_ch)
{
    if (input_ch.available(1)) {
        pixel_t input_value = input_ch.read();
        m_buffer[m_curr_line_idx][m_px] = input_value.value;

        pixel_t output_value;
        output_value.value = filter(m_buffer[m_preprev_line_idx][m_px], input_value.value);
        output_value.sync  = input_value.sync;
        output_ch.write(output_value);
        if (input_value.sync.end_of_line) {
            m_px = 0;
            incr_wrap(m_curr_line_idx);
            incr_wrap(m_preprev_line_idx);
        } else {
            m_px++;
        }
        if (input_value.sync.end_of_frame) {
            irq_ch.write(true);
        }
    }
}
```

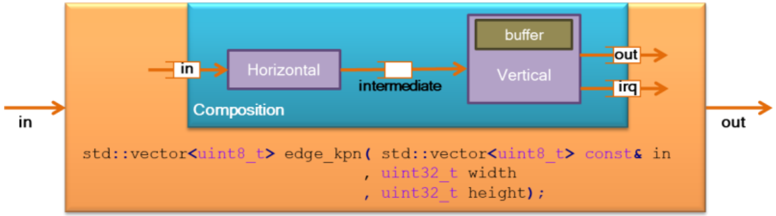
```
ww_uint8  m_buffer[3][2048];
ww_uint2  m_curr_line_idx;
ww_uint2  m_preprev_line_idx;
ww_uint11 m_px;
```

KPN model - Composition




```
class Composition
{
    ac_channel<pixel_t>  m_intermediate_ch;
    Horizontal_edge      m_horizontal_edge;
    Vertical_edge        m_vertical_edge;

public:
    Composition() {}

    #pragma hls_design interface
    void CCS_BLOCK(block)( ac_channel<pixel_t>& input_ch
                         , ac_channel<pixel_t>& output_ch
                         , ac_channel<bool>&    irq_ch)
    {
        m_horizontal_edge.block(input_ch, m_intermediate_ch);
        m_vertical_edge.block(m_intermediate_ch, output_ch, irq_ch);
    }
};
```

KPN model - KPN process wrapper




```
std::vector<uint8_t> edge_kpn(std::vector<uint8_t> const& in, uint32_t width, uint32_t height)
{
    pixel_t pix;
    ac_channel<pixel_t> input_ch, output_ch;
    ac_channel<bool>    irq_ch;
    Composition         composition;

    for (all input values) : construct pix (value and sync) from input vector
        input_ch.write(pix);

    while (!irq_ch.available(1)) {
        composition.block(input_ch, output_ch, irq_ch);
    }
    bool done = irq_ch.read();

    std::vector<uint8_t> out(width*height);
    for (all output values): read pix from channel and copy to out vector
        pix = output_ch.read();    out[ii] = pix.value;

    return out;
} /* edge_kpn */
```

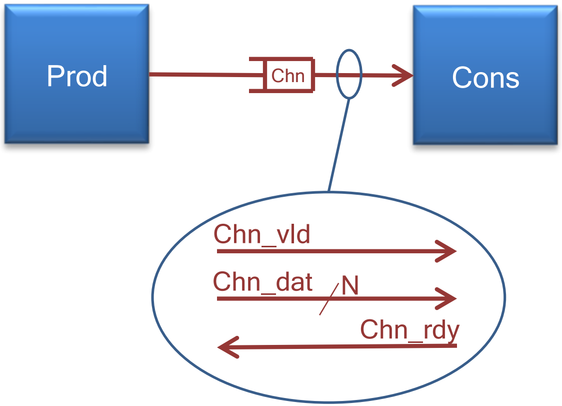
Test program - self-testing unit-test

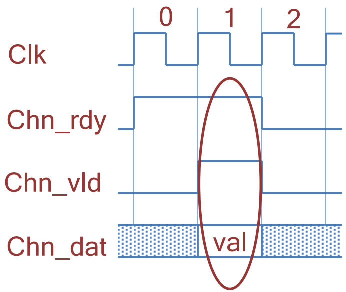
```
#define WIDTH_chameleon 1920
#define HEIGHT_chameleon 1280
static uint8_t data_chameleon[] = {
198,198,198,198,200, . . .
};
```

```
int run_test()
{
    uint32_t width = WIDTH_chameleon;  uint32_t height = HEIGHT_chameleon;  bool fail = false;
    std::vector<uint8_t> source_data(data_chameleon, data_chameleon + width*height);

    std::vector<uint8_t> out_func = edge_func(source_data, width, height);
    std::vector<uint8_t> out_kpn = edge_kpn(source_data, width, height);

    for (uint32_t py = 2; py < height-2; py++) {
        for (uint32_t px = 2; px < width-2; px++) {
            if (out_kpn[py*width + px] != out_func[(py-1)*width + (px-1)]) {
                fail = true;
            }
        }
    }

    return(fail ? EXIT_FAILURE : EXIT_SUCCESS);
} /* run_test */
```

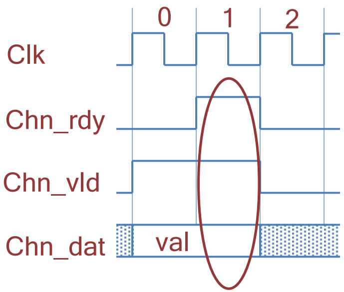
Now… lets move to the hardware view…

Channel in Software	Channel in Hardware
Sequential execution of processes Unbounded fifo channels	Defined fifo depth (no of elements) Handshake at write (may stall) Handshake at read (may stall)

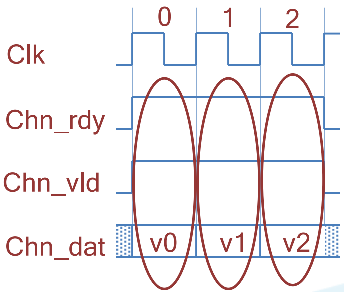




Data token transfer



Data token transfer



Data token transfers

Semantics of ac_channel operations

Operation	Software	Hardware
.write(token)	Always completes immediately; "infinite" fifo size	May stall the block until there is a room in the channel, or the consuming block is ready
.read()	Only allowed if token in the channel	May stall until token available in the channel or the producing block has valid data
.available(N)	Checks whether the input channel contains N tokens	Always evaluates to true
.nb_read(token)	Returns availability of input token; if true, it reads the token from the input channel	Idem, never stalls the block Availability of token is based on handshake signals
.size()	Returns number of tokens in the channel	Idem, never stalls the block Additional size signal is added in the channel interface - beware to specify max. channel if used on block interface

Horizontal_edge code hardware view

```
ww_uint8 m_pre_prev_value;
ww_uint8 m_prev_value;
```

```
void CCS_BLOCK(block)( ac_channel<pixel_t>& input_ch
                     , ac_channel<pixel_t>& output_ch)
{
    main:
    while (true) {
        if (input_ch.available(1)) {
            pixel_t input_value = input_ch.read();

            pixel_t output_value;
            output_value.value = filter(m_pre_prev_value, input_value.value);
            output_value.sync  = input_value.sync;
            output_ch.write(output_value);

            m_pre_prev_value = m_prev_value;
            m_prev_value     = input_value.value;
        }
    }
}
```

Development steps

- Functional code
  - Algorithm development (no theoretical or mathematical description)
  - Simplest possible C++ description
- KPN code – adds design detail 3 - 10 x SLOC
  - Partitioning and task parallelism
  - Local buffering and addressing
  - Test against functional code; test sequences (same interface)
- Catapult High Level Synthesis – adds time
  - Scripted directives how to introduce "time"
  - Generates schedule, data path and state machine (for e.g. stalling)
  - Per block verification using digital simulation
    - Check data throughput, channel sizes, directives,

Catapult initialization and technology selection

```
flow package forget  /Concat
flow package require /SCVerify

# set all clocks to 150 MHz (= 6.667ns clock period)
options set Interface/DefaultClockPeriod 6.667
options set Interface/DefaultClockOverhead 25.0

solution library remove *
solution library add mgc_Altera-Arria-V-3_beh -- \
    -rtlsyntool Quartus -manufacturer Altera \
    -family {Arria V} -speed 3 -part 5ASTFD5K3F40I3
solution library add Altera_DIST
solution library add Altera_FIFO

options set Input/SearchPath "."
solution file add edge_kpn.cpp
solution file add edge_func.cpp -exclude true
solution file add test.cpp -exclude true
```

Select flows

Clock settings

Technology library

Source code

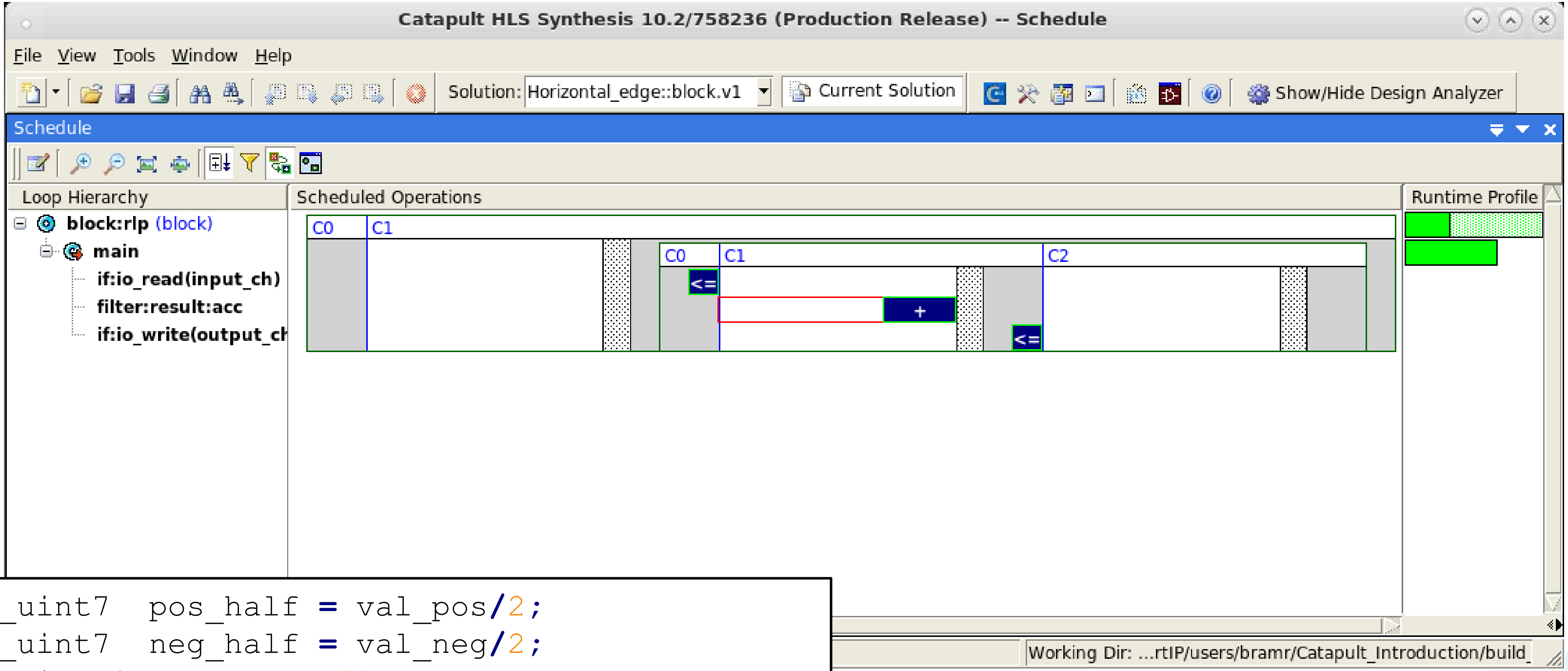
Catapult - build the Horizontal_edge block

```
# Synthesis task: Hierarchy
go analyze

directive set -DESIGN_HIERARCHY Horizontal_edge::block

# Synthesis tasks: Libraries - Mapping - Architecture
go compile
go libraries
go assembly

directive set /Horizontal_edge::block/block/main -PIPELINE_INIT_INTERVAL 1

# Synthesis tasks: Resources - Schedule - RTL
go architect
go allocate
go extract
```

```
Top-level block
All functions used are inlined
```

```
Initiation Interval
One execution is initiated
every clock cycle
```

Generated schedule of Horizontal_edge block



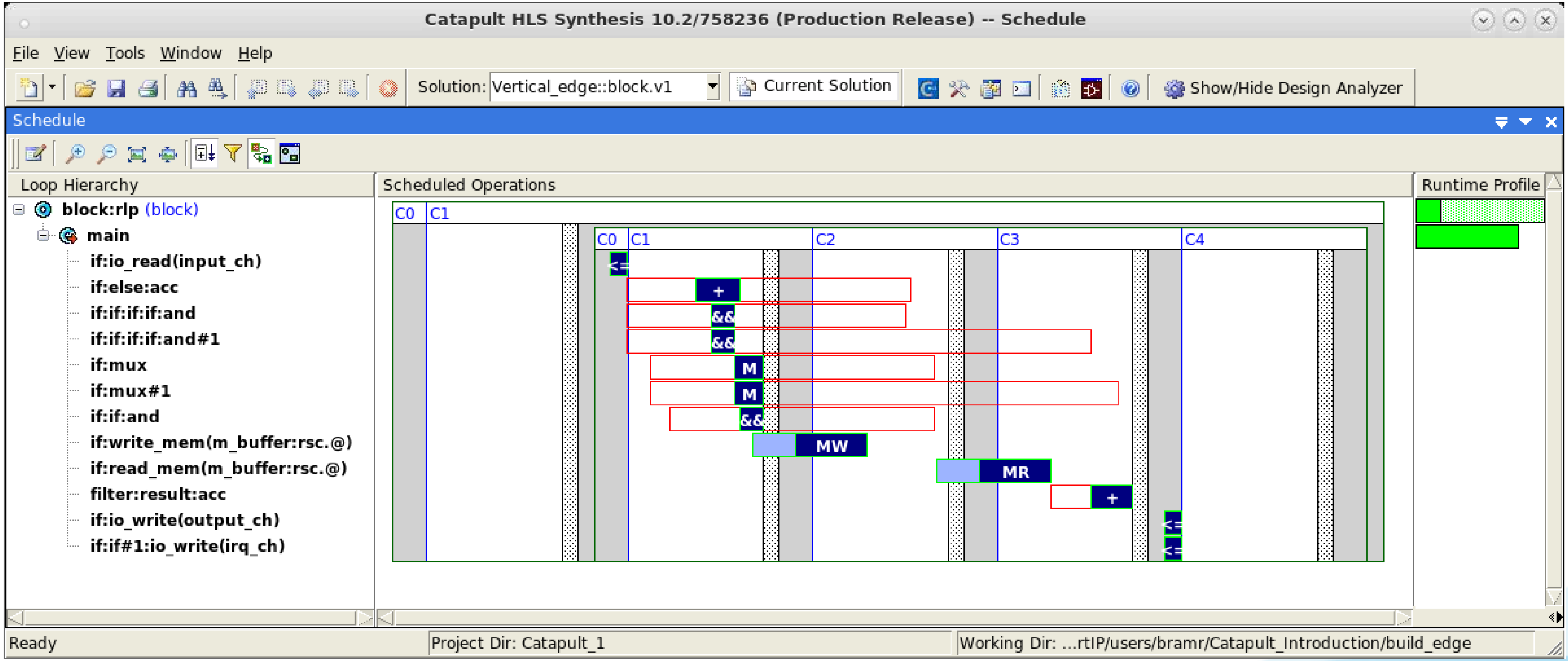

```
ww_uint7  pos_half = val_pos/2;
ww_uint7  neg_half = val_neg/2;
ww_uint10 result = 128 + pos_half - neg_half;
static_cast<ww_uint8>(result);
```

Catapult - build the Vertical_edge block

```
# Synthesis task: Hierarchy
go analyze

directive set -DESIGN_HIERARCHY Vertical_edge::block

# Synthesis tasks: Libraries - Mapping - Architecture
go compile
go libraries
go assembly

directive set /Vertical_edge::block/block/main -PIPELINE_INIT_INTERVAL 1
directive set /Vertical_edge::block/block/m_buffer:rsc -MAP_TO_MODULE Altera_DIST.DIST_1R1W_RBW

# Synthesis tasks: Resources - Schedule - RTL
go architect
go allocate
go extract
```

Buffer with one write port and one read port
Every cycle one read and one write access

Generated schedule of Vertical_edge block



Catapult - build the Composition block 1/2

```
# Synthesis task: Hierarchy
go analyze

directive set -DESIGN_HIERARCHY {Composition::block
    Horizontal_edge::block
    Vertical_edge::block
}

# Synthesis task: Libraries
go compile

solution library remove *
solution library add {[Block] Horizontal_edge::block.v1}
solution library add {[Block] Vertical_edge::block.v1}
```

Load sub-blocks into library

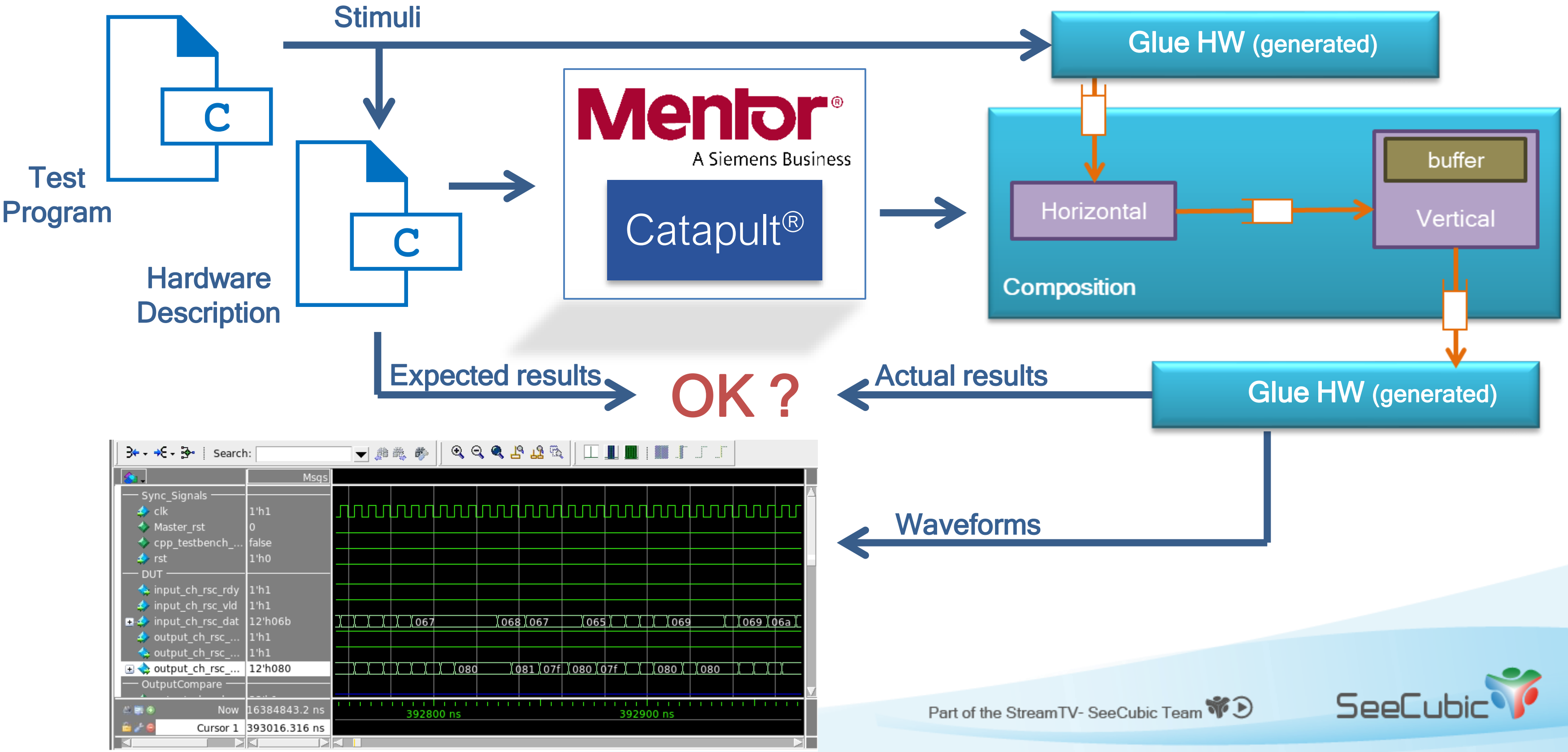
Catapult - build the Composition block 2/2

```
# Synthesis tasks: Mapping
go libraries

directive set /Composition::block/Horizontal_edge::block \
                              -MAP_TO_MODULE {[Block] Horizontal_edge::block.v1}
directive set /Composition::block/Vertical_edge::block \
                              -MAP_TO_MODULE {[Block] Vertical_edge::block.v1}

# Synthesis tasks: Architecture
go assembly

directive set /Composition::block/m_intermediate_ch:cns -MAP_TO_MODULE mgc_ioport.mgc_pipe_regfile
directive set /Composition::block/m_intermediate_ch:cns -FIFO_DEPTH 0

# Synthesis tasks: Resources - Schedule - RTL
go architect
go allocate
go extract
```

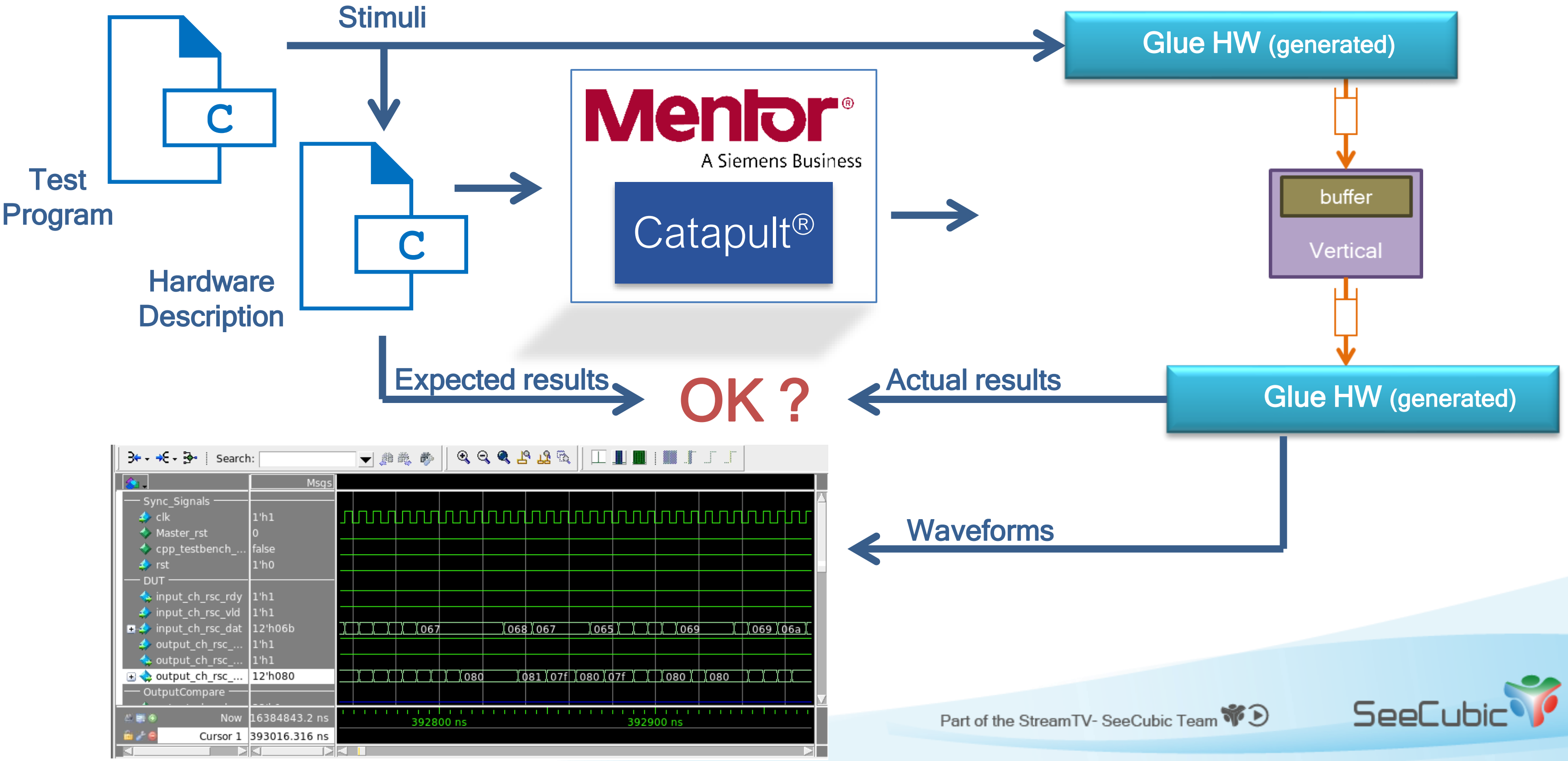
Use sub-blocks from library

Define what channel to use
and amount of tokens

Verification by co-simulation "SCverify" - Composition



Verification by co-simulation "SCverify" - Any [sub]block

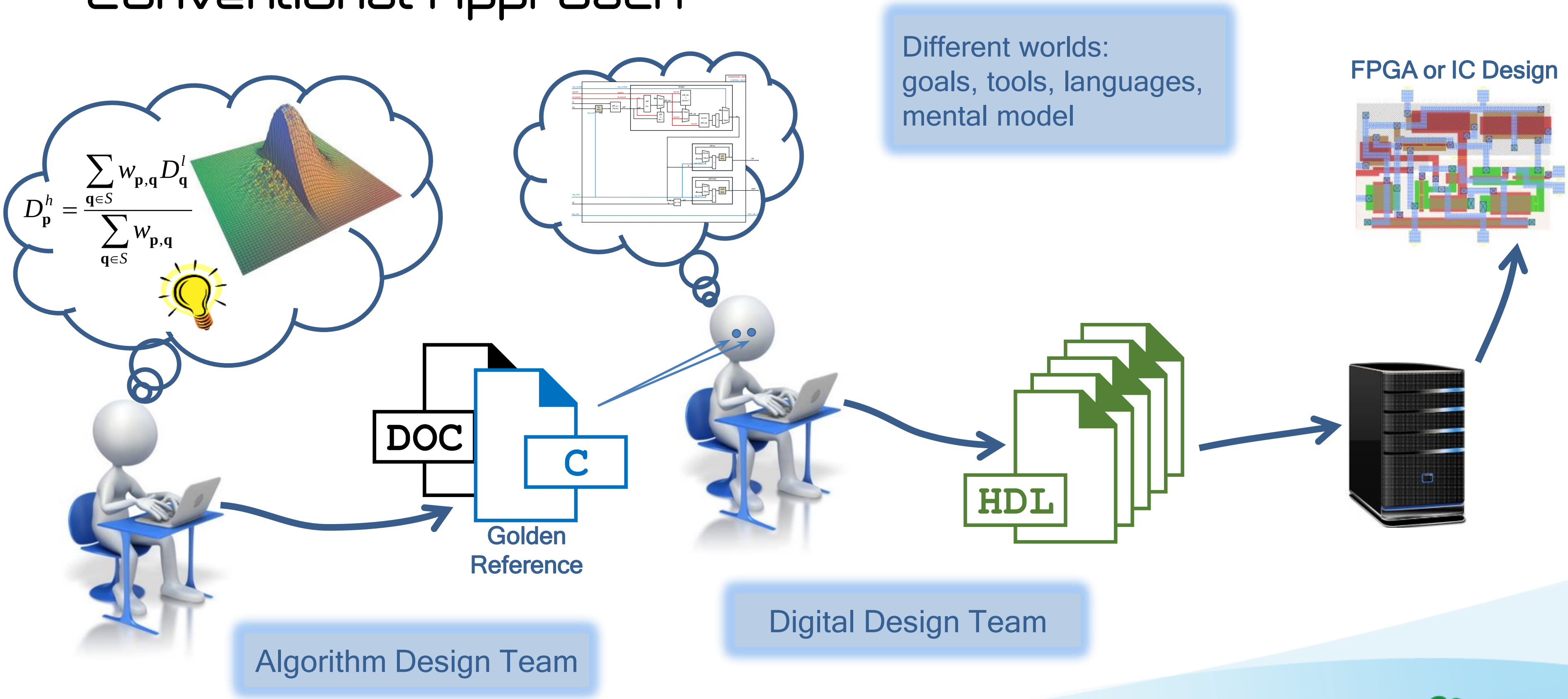


Observations

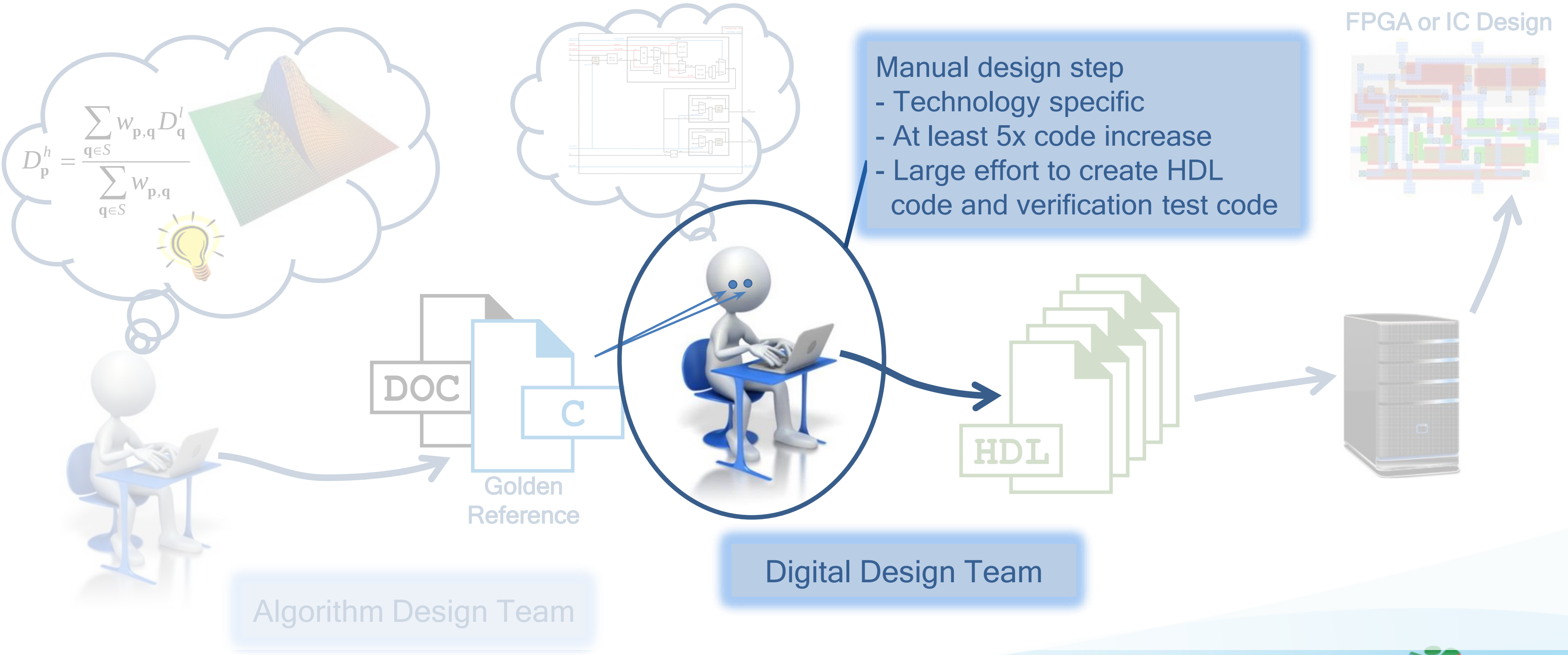
- All design in done in the C++ domain
- Algorithm, tests cases, hardware description

- Lets contemplate …
  - What does this mean?
  - What are the consequences?

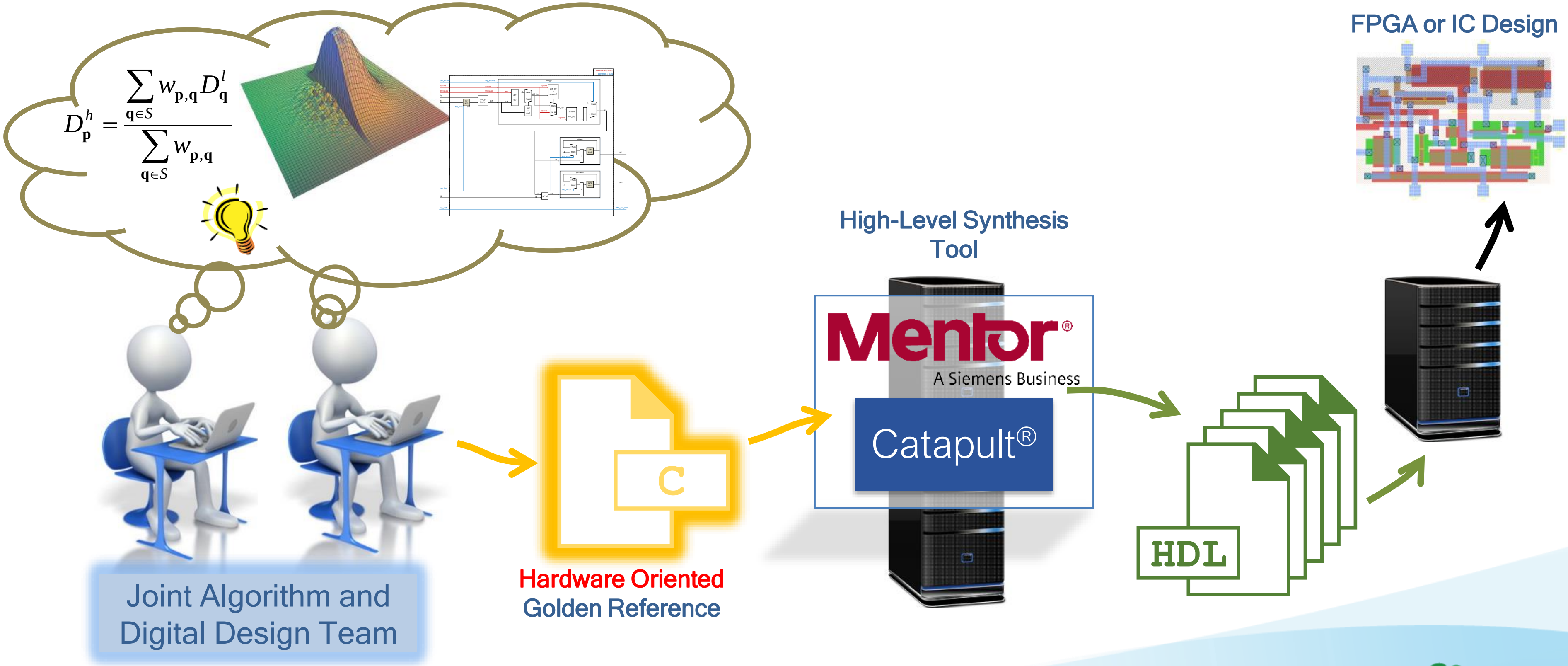
Conventional Approach



Conventional approach - Key Issues

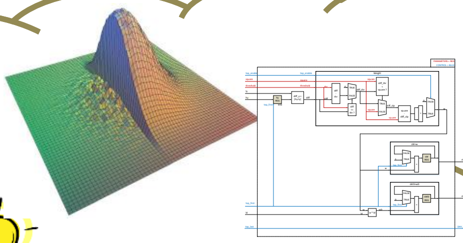


Design flow with Catapult HLS



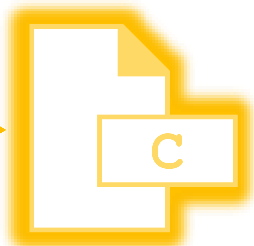
Designing with Catapult HLS

$$D_{\mathbf{p}}^{h}=\frac{\sum_{\mathbf{q}\in S} w_{\mathbf{p},\mathbf{q}} D_{\mathbf{q}}^{l}}{\sum_{\mathbf{q}\in S} w_{\mathbf{p},\mathbf{q}}}$$



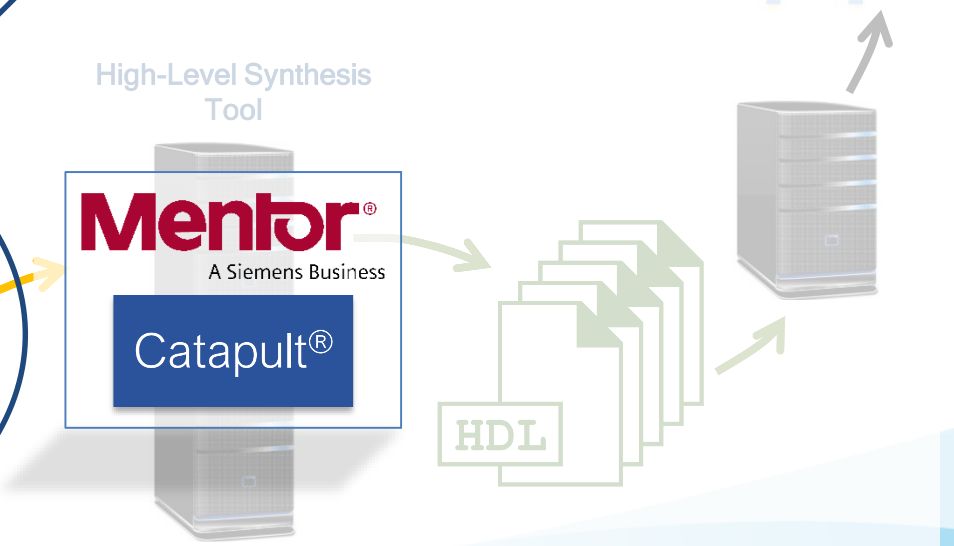
- One single team, different expertise
- Single codebase
- C/C++ as design language
- Design and test/verification in same enviornment

Joint Algorithm and Digital Design Team

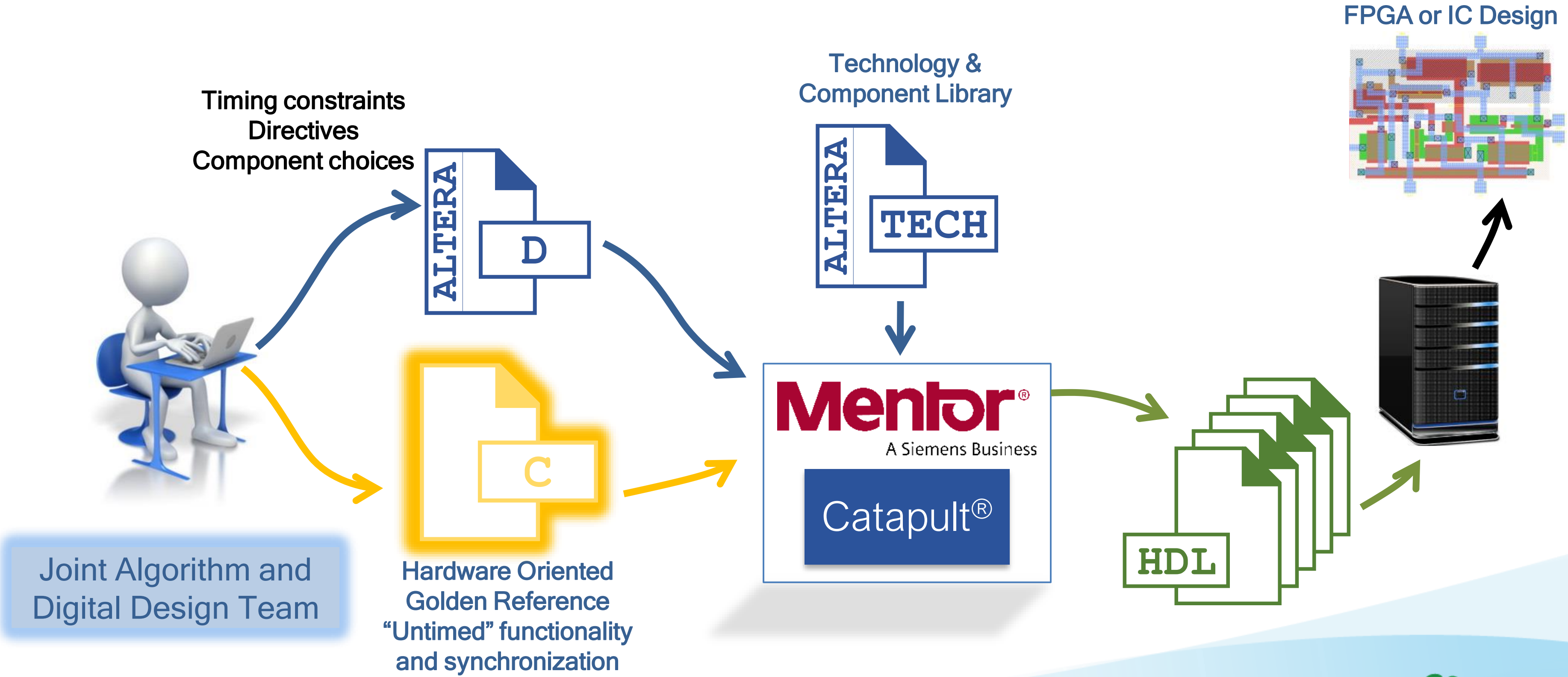


Hardware Oriented Golden Reference

High-Level Synthesis Tool



Catapult technology library



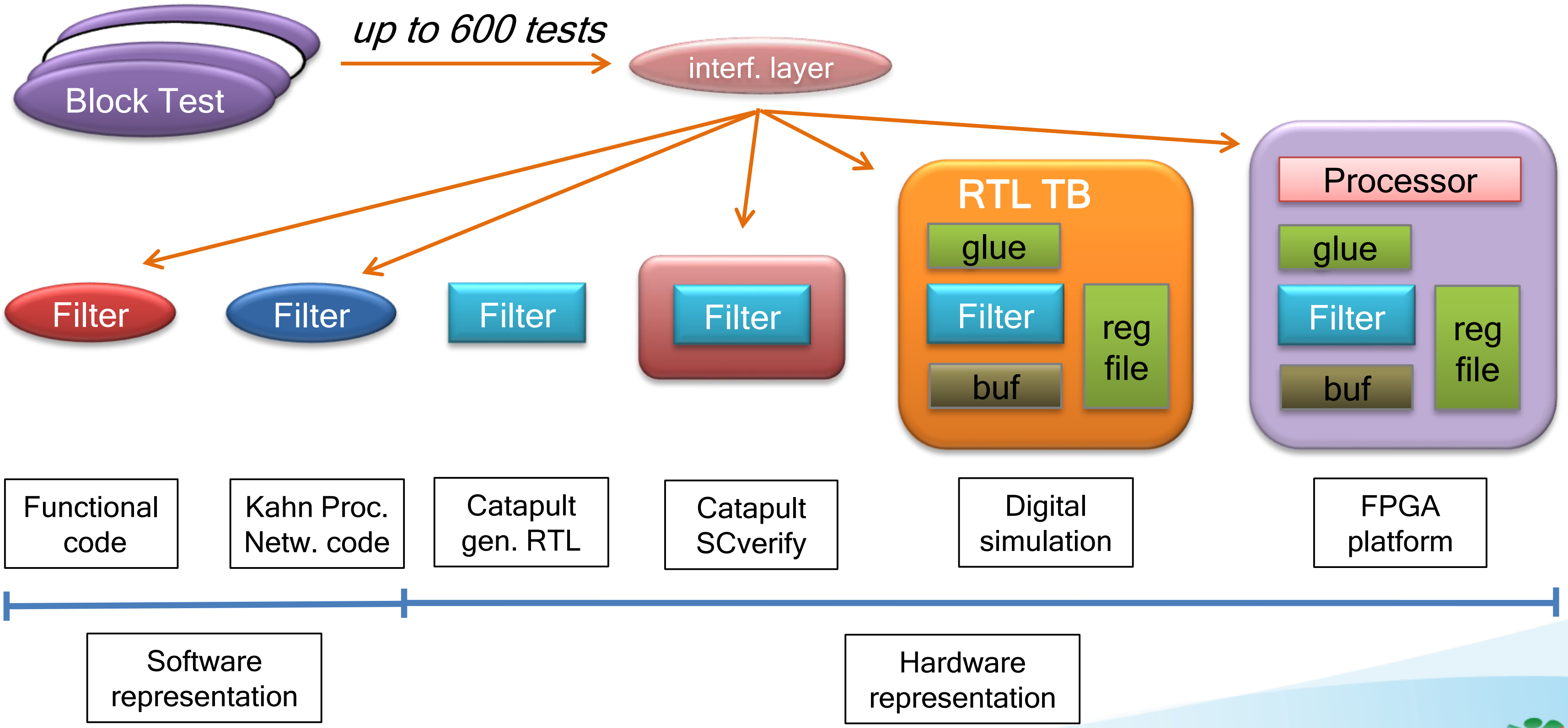
Observations

- All design in done in the C++ domain
- Algorithm, tests cases, hardware description

- Distinction between hardware and software has faded

- Lets further contemplate …
  - What does this mean?
  - What are the consequences?

Verification and validation overview



Experiences

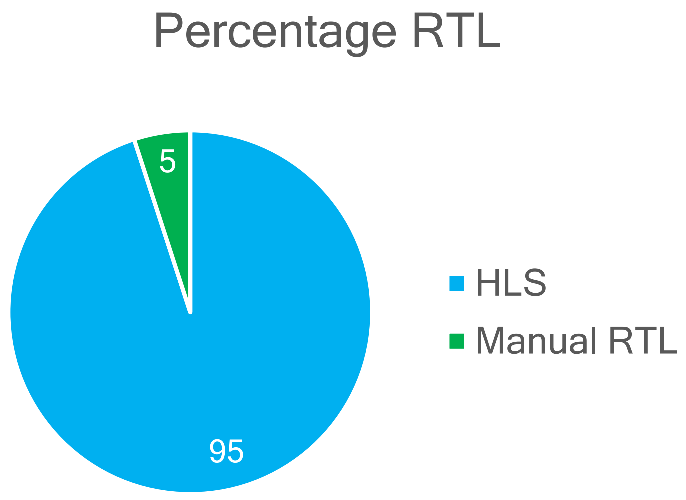
- Steep learning curve – support from Mentor Graphics is essential
- The tool provides many benefits and flexibility on design aspects
  - Algorithmic changes can still be applied late in the development process
  - Experimenting with different concepts is rather easily facilitated
- Catapult HLS optimal for
  - Signal processing HLS
  - New IP blocks
- Control blocks are more difficult (but we have used it)
- Some functions still in manual RTL (VHDL/Verilog)

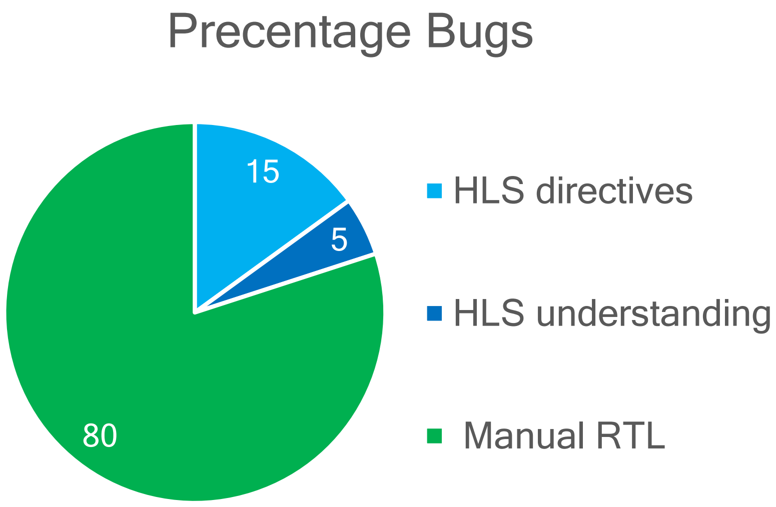
Lessons learned

- With HLS, digital designers start designing in C/C++
- Software designers really need to understand digital design
  - Catapult is *NOT* just another C-compiler
  - Need to understand what the backend tools do

- HLS designers must understand how and why the C/C++ code results in the generated hardware
- Adapt
  - Project organization
  - Design and verification methodology

Review of the project





Review of the project [2/2]

- For us, as a startup, Catapult required a serious financial investment
- We reduced more than 50% of our development time
  - Benefits hugely outperform the cost
  - Even with the learning curve penalty

- Boundary condition: new IP, starting from advanced algorithm

- We will use HLS for every new project — this is the future!

Answered?

What if we could “compile” for an ASIC?

- 40 years later: Yes, we can…
- …but not just “any” code…

Demo

- Depth estimation in FPGA shown on Ultra-D display