

# High Level Synthesis techniques

Laurent Hili  
ESA-ESTEC  
19/09/2011

# What are the challenges ?



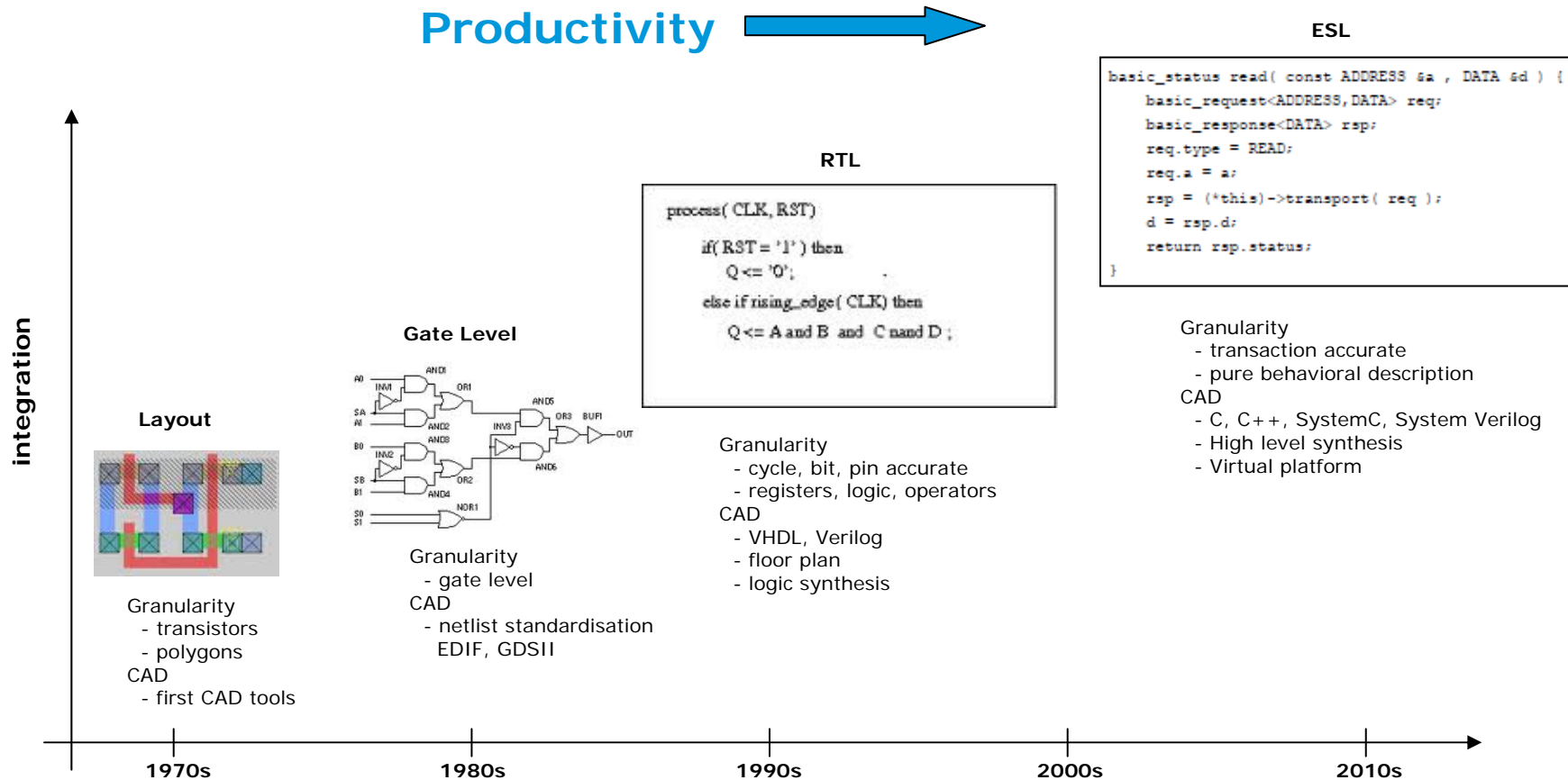
- ✓ Everything tends to become more complex (Moore's Law)
- ✓ Miniaturisation thanks to CMOS technologies (65, 45, 28 and 22nm) offers possibilities to design chips exhibiting a complexity beyond a billion of transistors
- ✓ Systems on board(s) tend to move to systems on chip (SoC)
- ✓ Possibility to integrate various technologies on the same chip (SW, HW digital, HW analog, MEMs, sensors)

# What are the challenges ?



- ✓ Necessity to have higher abstraction languages to face the new challenges raised by tighter HW/SW integration and trade offs
- ✓ Necessity to gain in productivity in order to handle the ever growing complexity while using the same number of designers (or even less)
- ✓ Necessity to put in place advanced CAD techniques to enable the productivity gains
  - ✓ Languages: C, C++, SystemC, System Verilog
  - ✓ Modeling: Transactions Level Modeling (TLM) & Transaction Based Verification (TBV)
  - ✓ CAD tools: High level synthesis / Virtual Platform

# Productivity through abstraction



# Benefits of High Level Synthesis (HLS)



- ✓ Higher productivity
- ✓ New IP development ~ 2 to 3 times faster than RTL (VHDL / Verilog)
- ✓ Possibility to describe an algorithm in a very concise way compared to HDL languages (~ 10 times fewer lines of code to maintain)
- ✓ The designer can better focus on the functionality rather than implementation details
- ✓ HLS can generate many RTL derivatives from same C code in a time effective manner (architecture exploration)
- ✓ HLS can easily be merged in a HW/SW co-design flow
  - ✓ Integration with virtual platform (SystemC TLM)
  - ✓ Integration with HDL flow (simulators, logic synthesis)

(ST Microelectronics source, 9<sup>th</sup> annual ESL Symposium, June 2011)

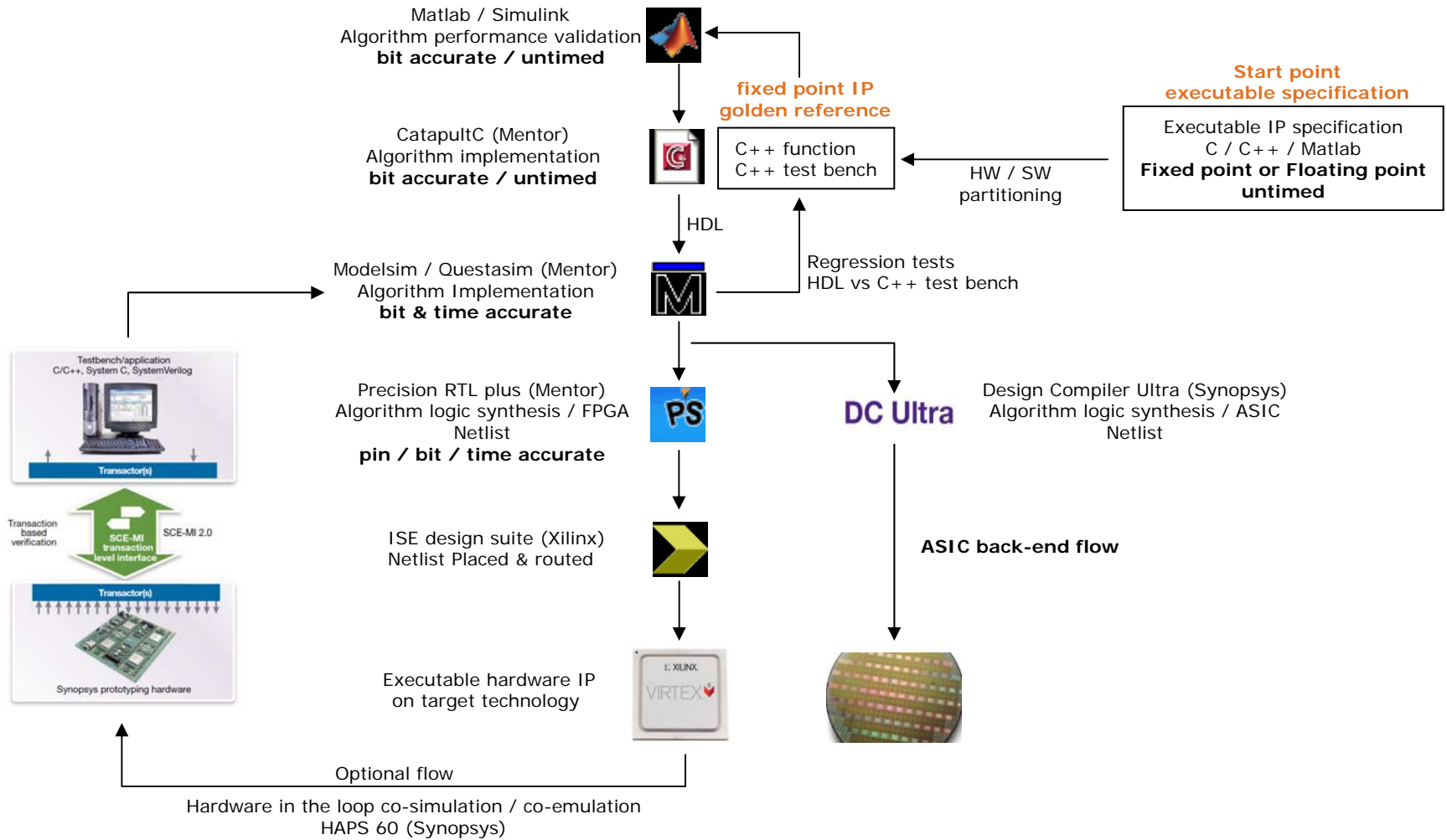
- ✓ Architecture exploration → ability to generate RTL variants
  - ✓ Loops optimisation: parallelisation and/or pipelining
  - ✓ Arrays optimisation: registers, RAMs, ROMs
  - ✓ Interfaces optimisation: wire, enable, handshake, bus, NoC (Network on Chip interface)
  
- ✓ Gantt chart analysis → ability to display concurrency and/or dependencies between resources / tasks
  
- ✓ Verification flow analysis → ability to run regression tests, Transaction Based Verifications (TBV) in order to check the RTL code automatically generated against original C++ code

- ✓ Cross probing analysis → ability to identify resources in the Gantt chart or RTL and map them to the original C++
- ✓ Interfacing with conventional ASIC / FPGA back end flows (RTL simulators and logic synthesis)
- ✓ Architecture exploration → target technology aware
  - ✓ Resources aware: operators, memories, interfaces
  - ✓ Timing aware
  - ✓ Area aware
  - ✓ Power aware (feature available as an option sometimes)

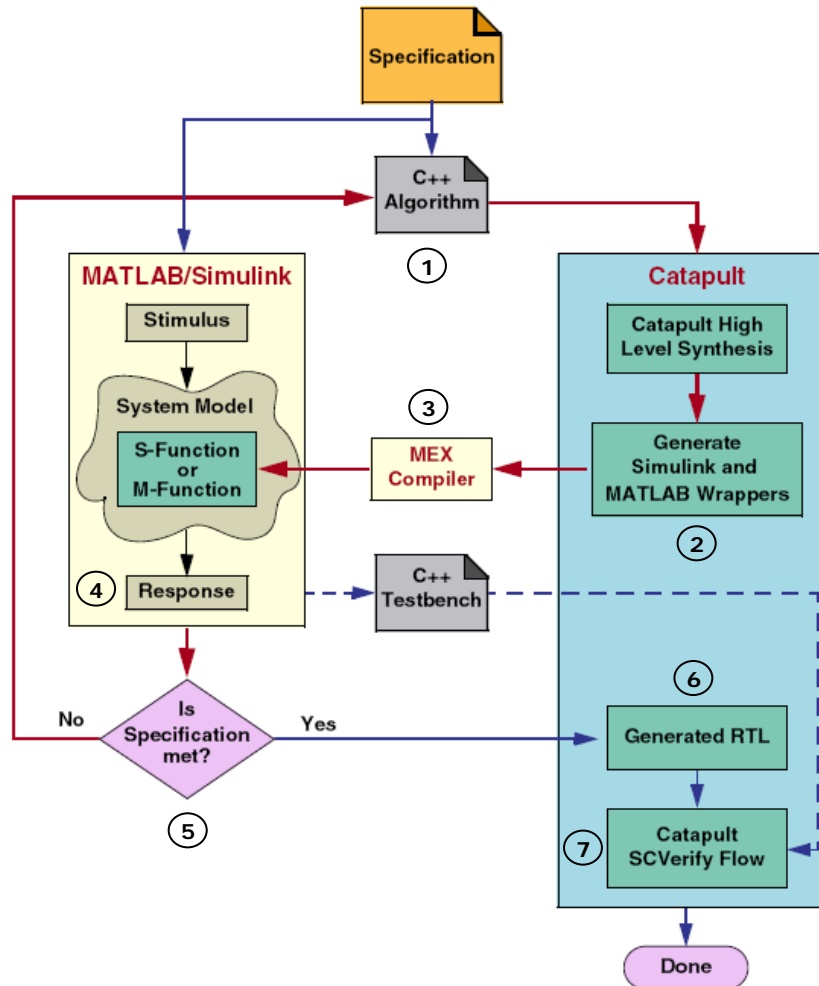
- ✓ CatapultC originally developed by Mentor Graphics has been spun off to Calypto the 26/8/2011. This tool has been one of the first used for ASICs tape out. CatapultC is in full production in Thales where 3 ASICs out of 4 are now produced using this methodology
- ✓ Symphony C compiler → Synopsys
- ✓ C to Silicon compiler → Cadence
- ✓ Cynthesizer → Forte Design Systems



# ESA High Level Synthesis flow



# CatapultC and Simulink integration



Step1: C++ algorithm coding

Step2: Simulink wrappers generation

Step3: Wrapper compilation with MEX

Step4: Simulink simulation (algo performances)

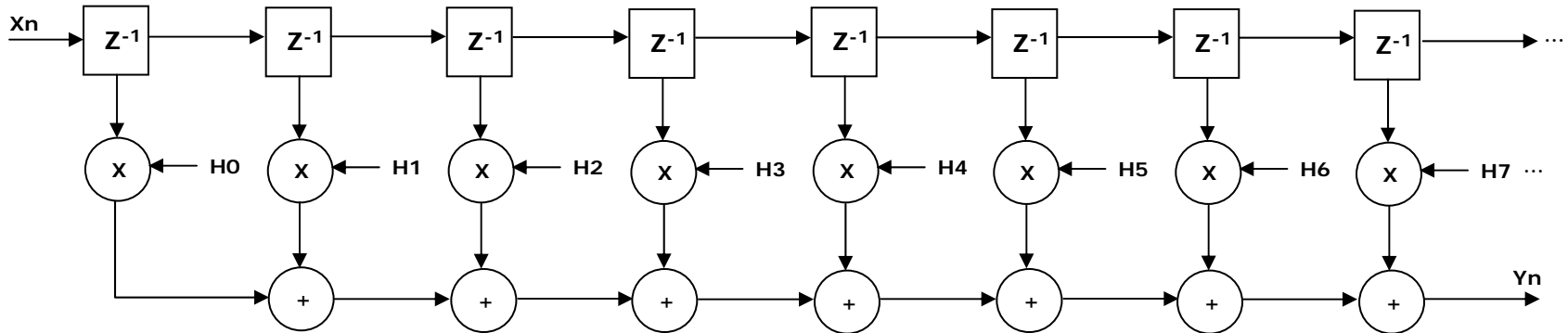
Step5: if step4 not OK then iterate in step1 otherwise goto step6

Step6: Generate RTL code

Step7: Run regression tests (SystemC verification flow / Transaction Based Verification TBV)

# FIR filter example (algorithm)

FIR filter direct implementation



$$Y(n) = \sum_{i=0}^N H(i) \times X(n-i)$$

Let's assume a 8 TAPS FIR filter  $N=7$

$$Y(n) = \sum_{i=0}^7 H(i) \times X(n-i)$$

$$Y(n) = H(0).X(n) + H(1).X(n-1) + H(2).X(n-2) + H(3).X(n-3) + \dots + H(7).X(n-7)$$

Let's assume  $n=7$

$$Y(7) = H(0).X(7) + H(1).X(6) + H(2).X(5) + H(3).X(4) + \dots + H(7).X(0)$$

# FIR filter example (source code)



## fir.cpp

```
#include "fir.h"

#pragma hls_design top
void fir_filter (data_t &input, data_t coeffs[NUM_TAPS], data_t &output)
{
    static data_t regs[NUM_TAPS];
    accu_t temp = 0;

    SHIFT:
    for (int i = NUM_TAPS-1; i >= 0; i--)
    {
        if (i == 0)
            regs[i] = input;
        else
            regs[i] = regs[i-1];

    }

    MAC:
    for (int i = NUM_TAPS-1; i >= 0; i--)
    {
        temp += coeffs[i] * regs[i];
    }

    output = temp;
}
```

Pragma identifying the top or code to be synthesised



Loops must be labelled



8 registers 18 bits



## fir.h

```
#ifndef _FIR_FILTER_H
#define _FIR_FILTER_H

#include <ac_fixed.h>
#define NUM_TAPS 8

typedef ac_fixed<18, 2, 1, AC_TRN, AC_WRAP> data_t;
typedef ac_fixed<24, 8, 1, AC_TRN, AC_WRAP> accu_t;

void fir_filter (data_t& input, data_t coeffs[NUM_TAPS], data_t& output);

#endif
```

8 coefficients



Fixed point types



# FIR filter example (test bench)



## fir-tb.cpp

```
#include <iostream>

#include "mc_scverify.h"
#include "fir.h"

// some functions for generating random test vectors
.....

// -----
// Start of MAIN

CCS_MAIN(int argc, char *argv)
{
    // Place local testbench variables here
    ac_fixed<18, 2, true, AC_TRN, AC_WRAP > input;
    data_t coeffs[8];
    ac_fixed<18, 2, true, AC_TRN, AC_WRAP > output;

    // Initialize local variables to zero
    init_input(input);
    init_coeffs(coeffs);
    init_output(output);

    // Main test iterations start here
    for (int iteration = 0; iteration < 100; ++iteration) {

        // Set test values for this iteration
        throw_dice_for_input(input);
        throw_dice_for_coeffs(coeffs);

        // Call original function and capture data
        CCS_DESIGN(fir_filter)(input, coeffs, output);
    }
    // Return success
    CCS_RETURN(0);
}
```

Unique test bench is used for regression tests RTL vs C++ (Transaction Based Verification)

Instructions in red are only ones differing from original C++ test bench

Those instructions are used by CatapultC synthesis

# FIR filter example (architecture constraints)



Task Bar: Synthesis Tasks

- Add Input Files
- Setup Design
- Architecture Constrai...
- Schedule
- Generate RTL

Project Files: fir\_filter.v3 (Passed Extract)

- Input Files
  - fir.cpp
  - fir\_tb.cpp (Excluded)
- Output Files
- Reports
- Schedule
- VHDL
- SCVerify
- Schematics
- Verification
- MS Visual C++ 9.0
- ModelSim
  - Cycle VHDL output 'cycle.vhdl' vs Untimed C++
  - RTL VHDL output 'rtl.vhdl' vs Untimed C++
  - Mapped VHDL output 'rtl.vhdl' vs Untimed C++
  - Gate VHDL output 'gate.vhdl' vs Untimed C++
- Simulation
  - MATLAB and Simulink
  - Synthesis
  - Precision

Details - Loop

Path: /fir\_filter/core/main  
File: C:\PROGRA~1\MENTOR~1\CATAPU~1.104\mgc\_home\pkgs\CCS\_TO~1\flows\matlab\src\fir.cpp(4)

Loop: main

Iteration Count: [ ]

Unroll

Partial: [ ]

Pipeline

Initiation Interval: [ ]

Generate distributed pipeline

Decoupling stages: [ ]

Loop can be Merged

Settings [Apply] [Cancel]

Transcript

0 Errors 0 Warnings 0 Infos 0 Comments 0 Commands Location





```
# Message
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\mgc_Xilinx-VIRTEX-6-1L_beh_psr.lib' [mgc_Xilinx-VIRTEX-6-1L_beh_psr]... LIB-49
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\ram_Xilinx-VIRTEX-6-1L_RAMDB.lib' [ram_Xilinx-VIRTEX-6-1L_RAMDB]... LIB-49
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\ram_Xilinx-VIRTEX-6-1L_PIPE.lib' [ram_Xilinx-VIRTEX-6-1L_PIPE]... LIB-49
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\ram_Xilinx-VIRTEX-6-1L_RAMSB.lib' [ram_Xilinx-VIRTEX-6-1L_RAMSB]... LIB-49
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\rom_Xilinx-VIRTEX-6-1L.lib' [rom_Xilinx-VIRTEX-6-1L]... LIB-49
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\rom_Xilinx-VIRTEX-6-1L_SYNC_regin.lib' [rom_Xilinx-VIRTEX-6-1L_SYNC_regin]... LIB-49
# Reading component library '$MGC_HOME\pkgs\siflibs\psr2009a_up2\rom_Xilinx-VIRTEX-6-1L_SYNC_regout.lib' [rom_Xilinx-VIRTEX-6-1L_SYNC_regout]... LIB-49
```

fir\_filter{2}>

Ready

Project Dir: Catapult Working Dir: ... Settings\Laurent Hill\Desktop\FIR Filter example

# FIR filter example (architecture exploration)

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
 fir_filter.v3 (extract)	8	400.00	10	500.00	43.03	858.44
 fir_filter.v5 (extract)	9	450.00	10	500.00	44.89	678.12
 fir_filter.v11 (extract)	8	400.00	8	400.00	44.51	593.86
 fir_filter.v12 (extract)	<b>1</b>	<b>50.00</b>	<b>1</b>	<b>50.00</b>	<b>42.85</b>	<b>2966.85</b>

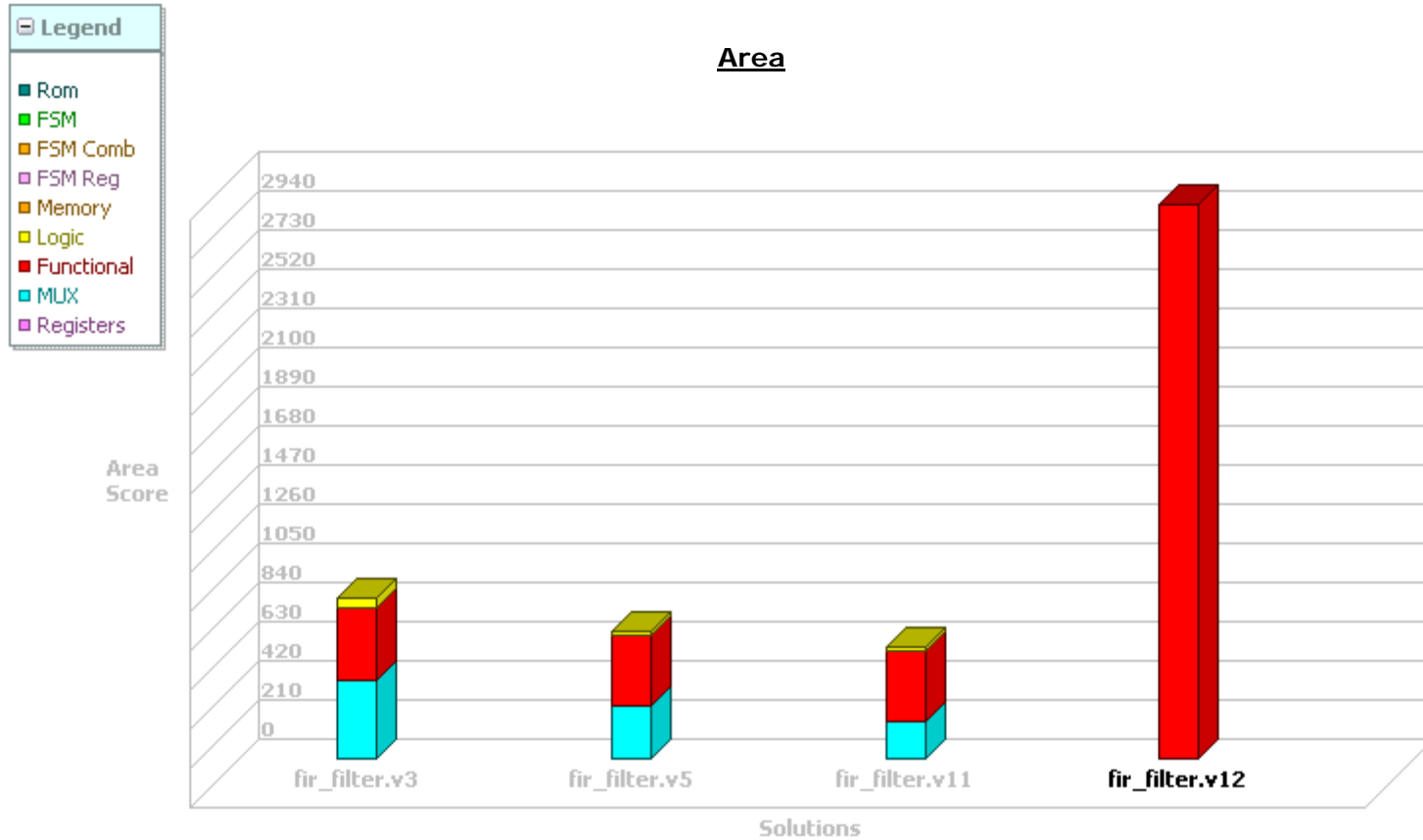
Clock period constrained to 50ns (20 MHz)

solution	resources		
	MAIN loop	SHIFT loop	MAC loop
Fir_filter.v3	Rolled	Rolled	Rolled
Fir_filter.v5	Rolled	Unrolled	Rolled
Fir_filter.v11	Pipelined II = 1	Unrolled	Pipelined II = 1
Fir_filter.v12	Pipelined II = 1	Unrolled	Unrolled

Default implementation loops are rolled and if timings constraints allow loops are merged

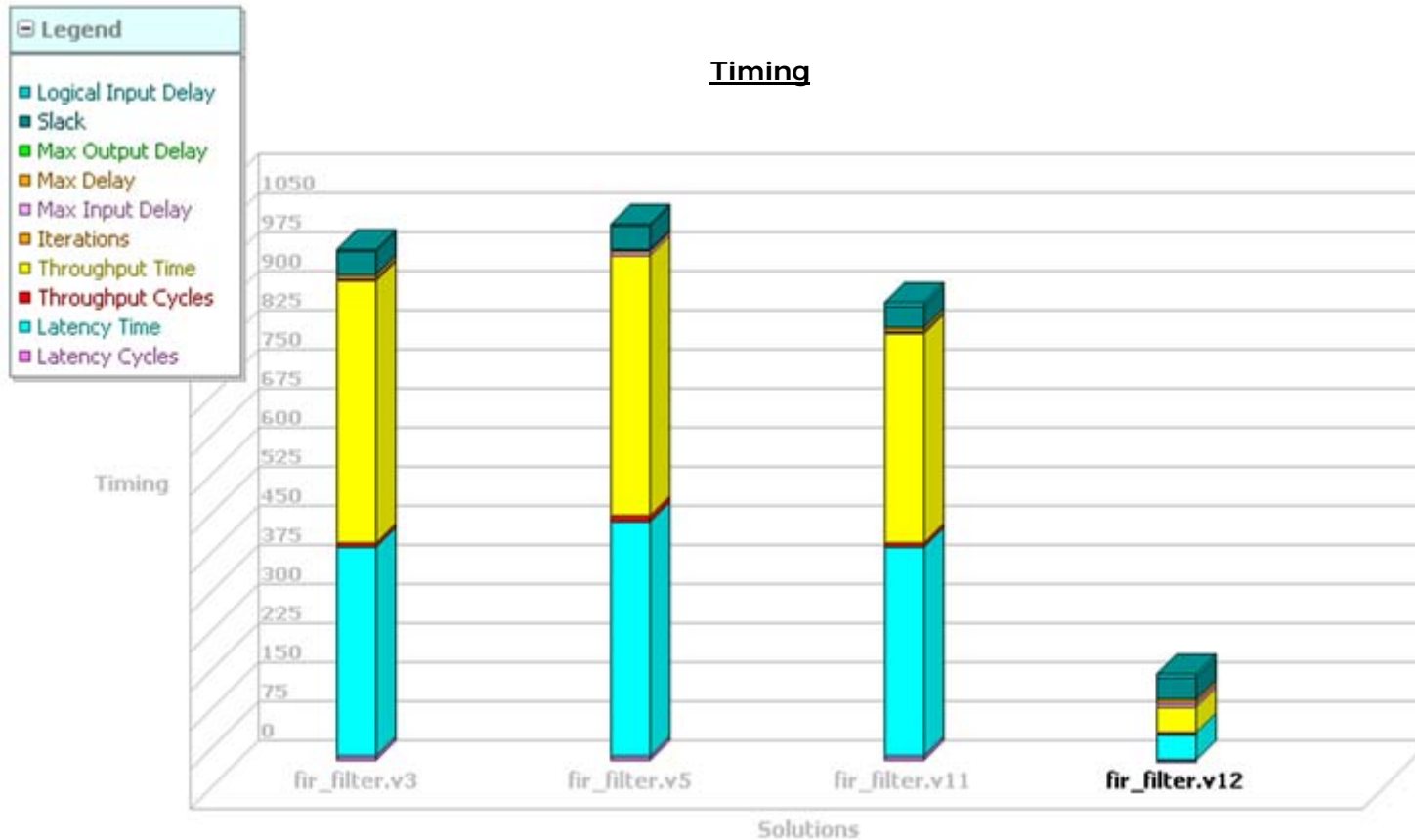
II = 1  
Initial Interval = 1  
1 data fed in the pipeline  
Every clock cycle

# FIR filter example (architecture exploration)



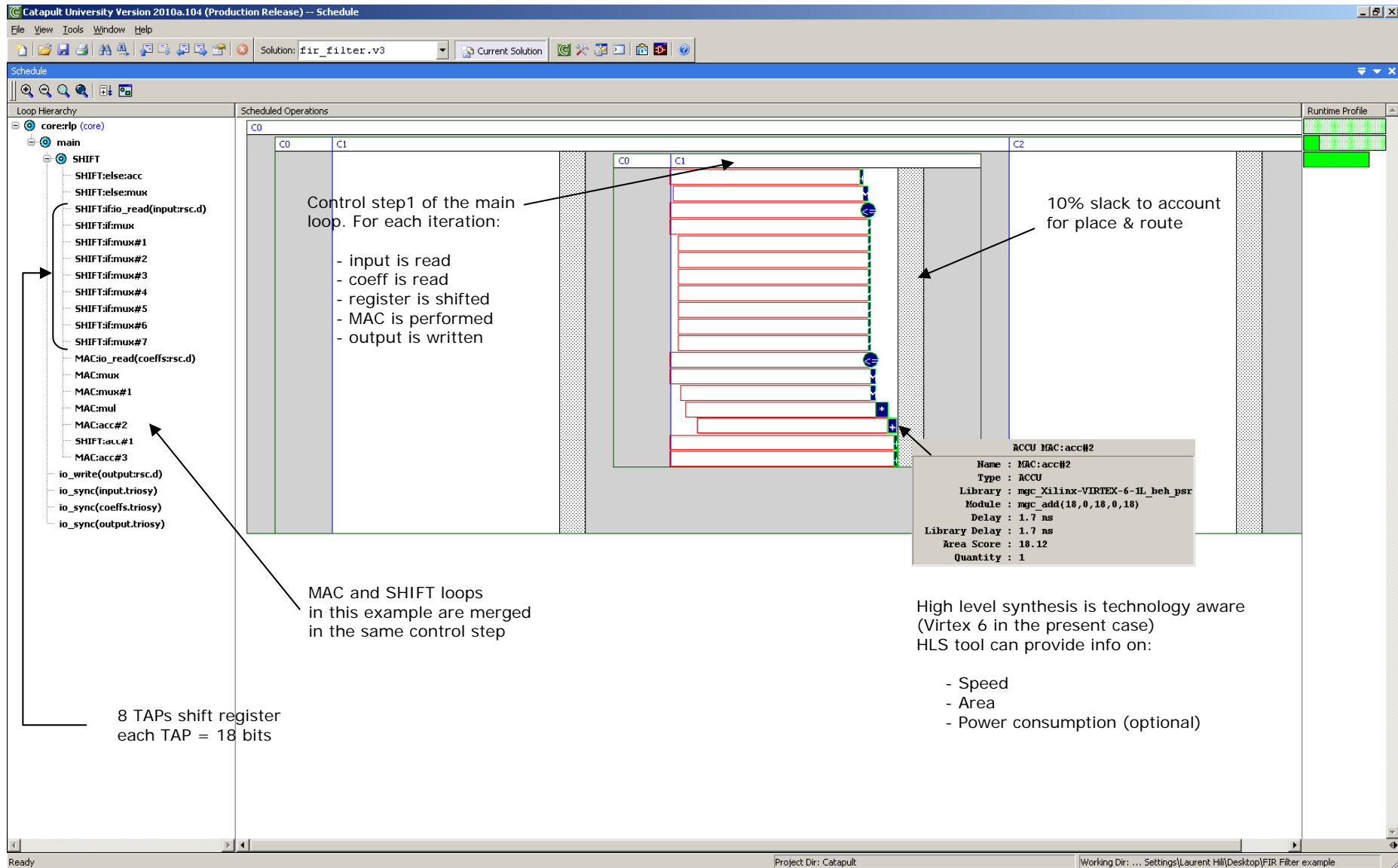


# FIR filter example (architecture exploration)

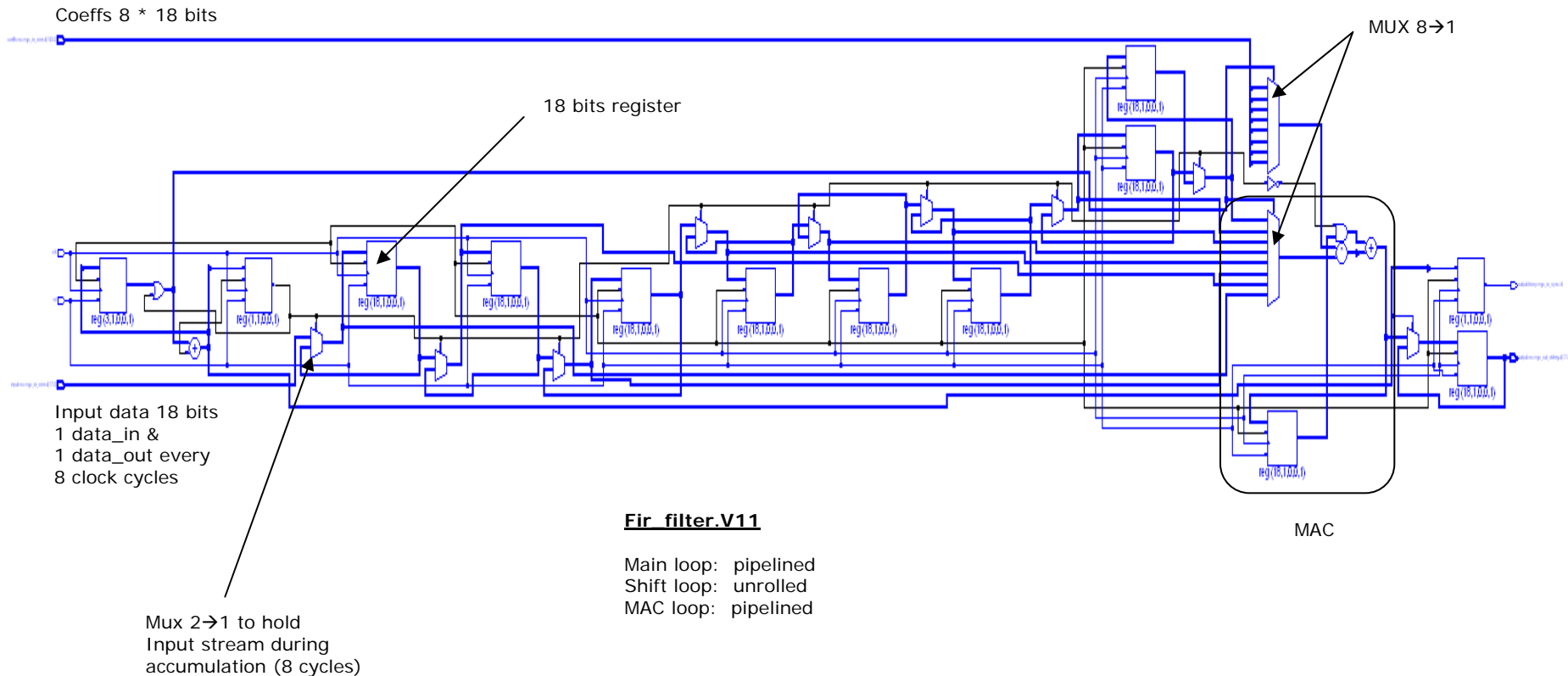


Fir\_filter.V12 is the fastest solution but also consumes more resources (direct implementation)

# FIR filter example (Gantt chart analysis)

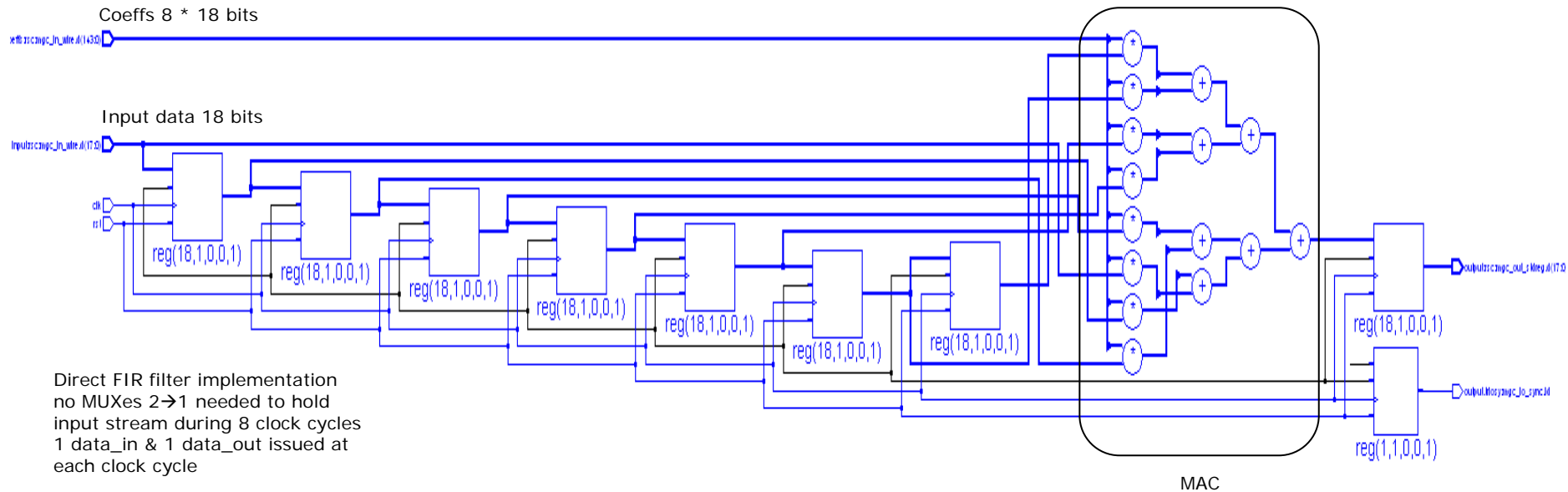


# FIR filter example (RTL code & target technology netlist generation)



***Best area solution***

# FIR filter example (RTL code & target technology netlist generation)



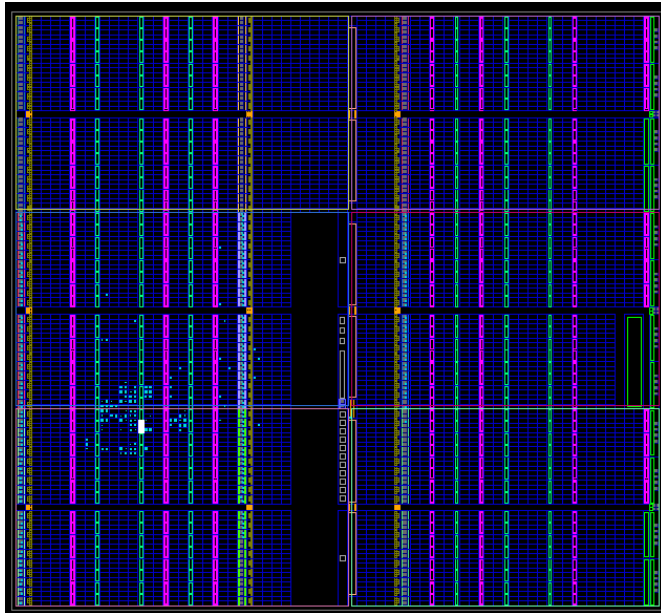
## Fir filter.V12

Main loop: pipelined  
Shift loop: unrolled  
MAC loop: unrolled

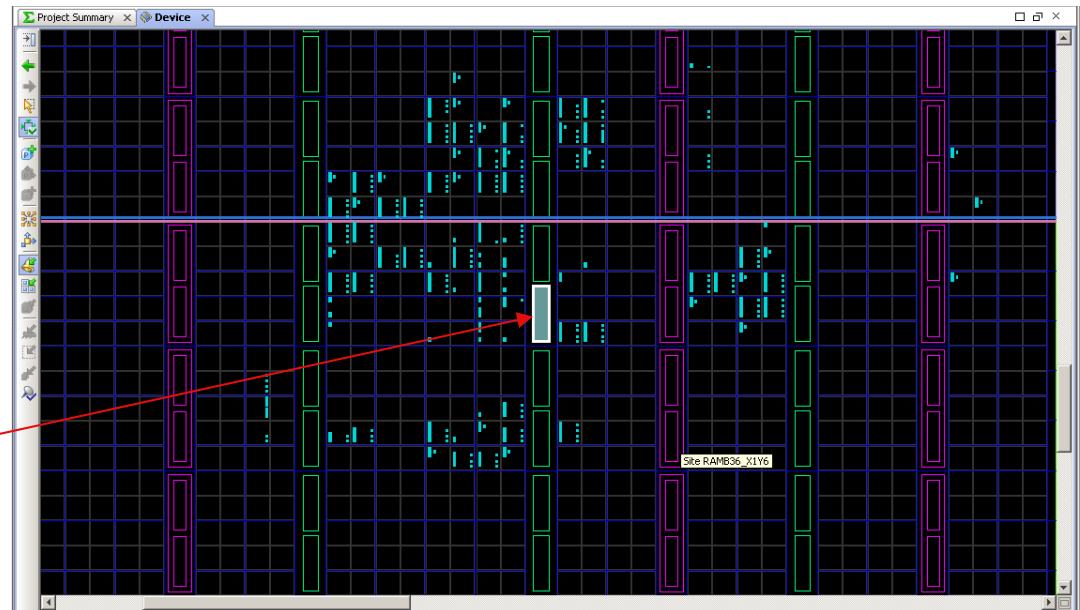
**Best timing solution**

# FIR filter example (implementation after synthesis & place-route)

Complete layout

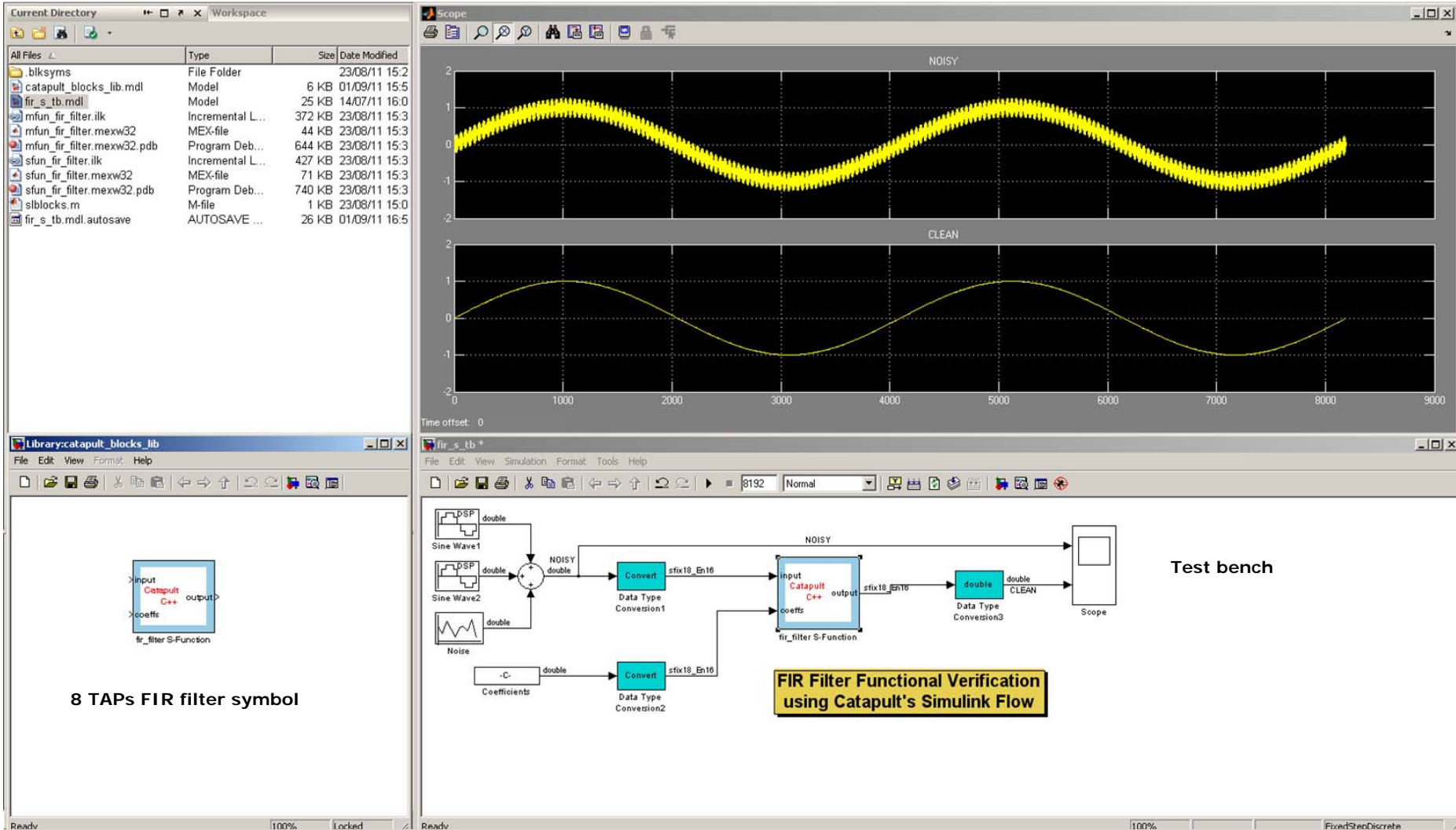


Layout zoom

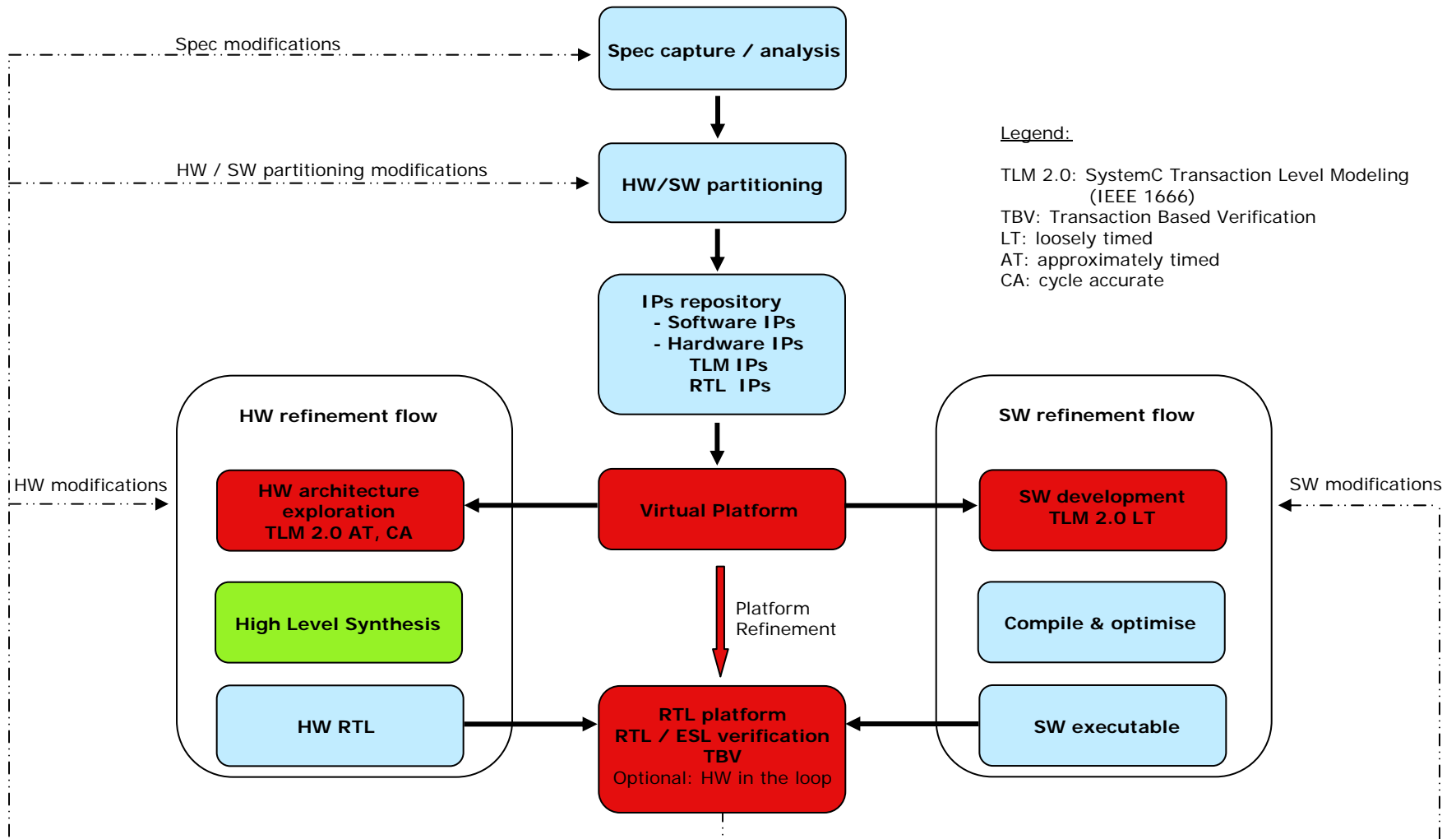


MAC mapped on DSP 48 block  
(example based on Virtex 6)

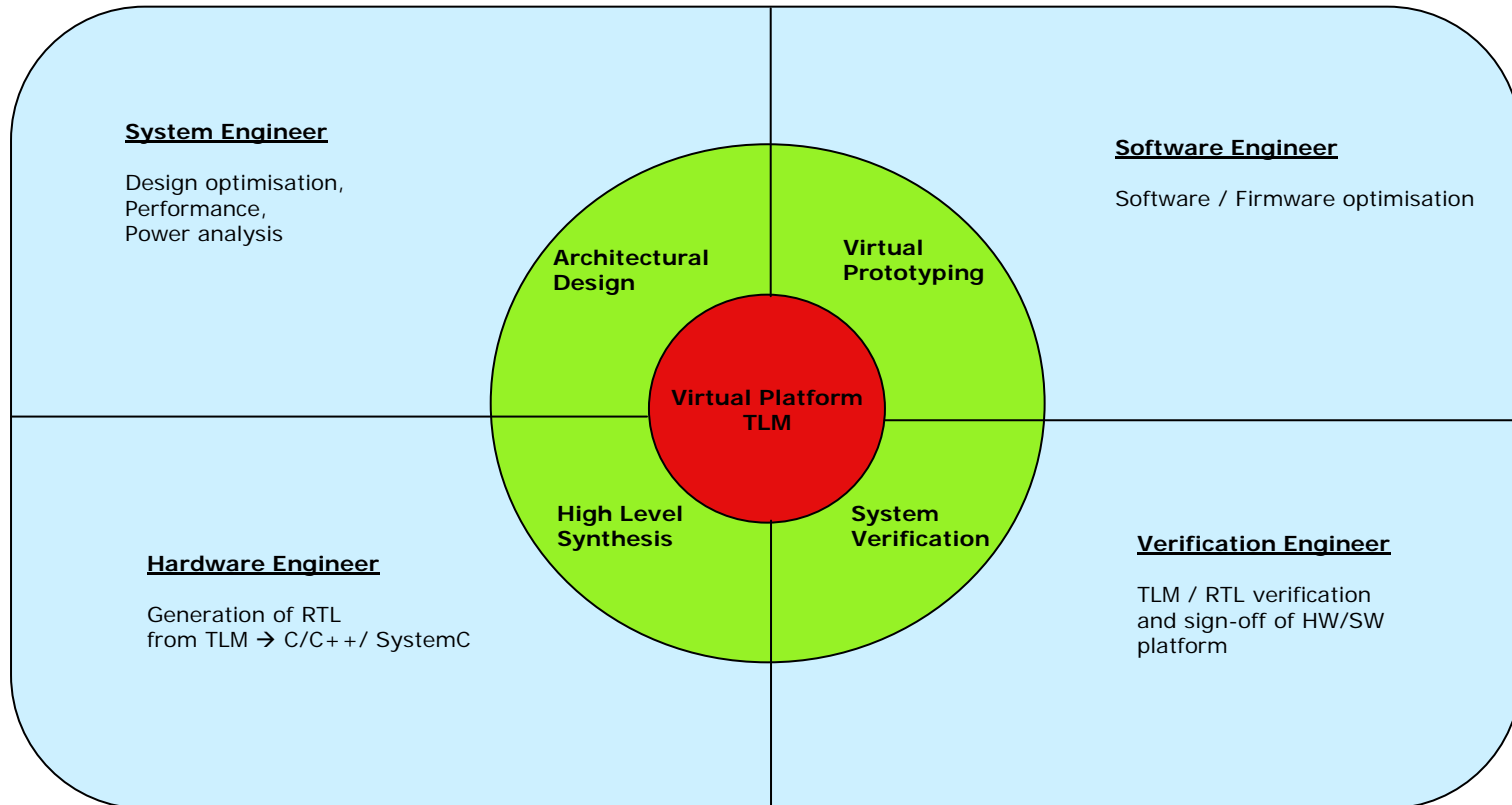
# FIR filter example (simulink validation)



# Integration of HLS flow with virtual platform



# Virtual Platform deployment view





# Thanks for your attention

Special thanks to my colleague Jelle Poupaert  
and Stephane Labert (Mentor Graphics France)  
for their support

Any question ?