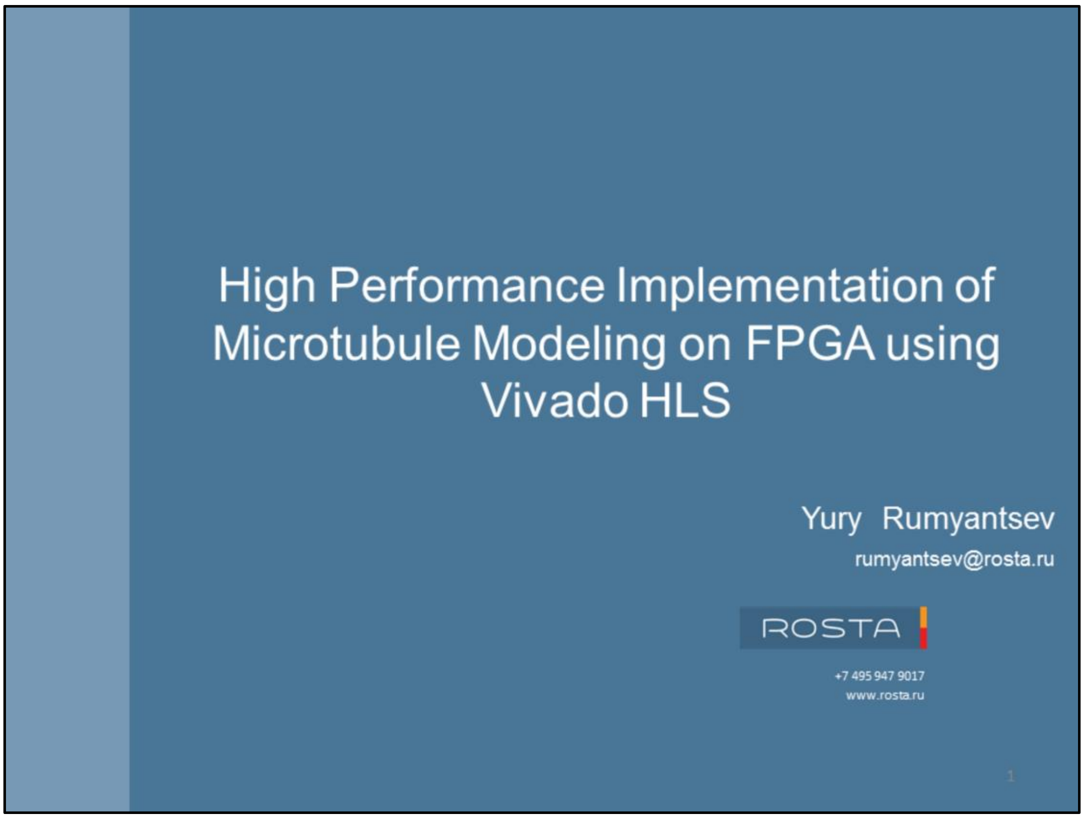


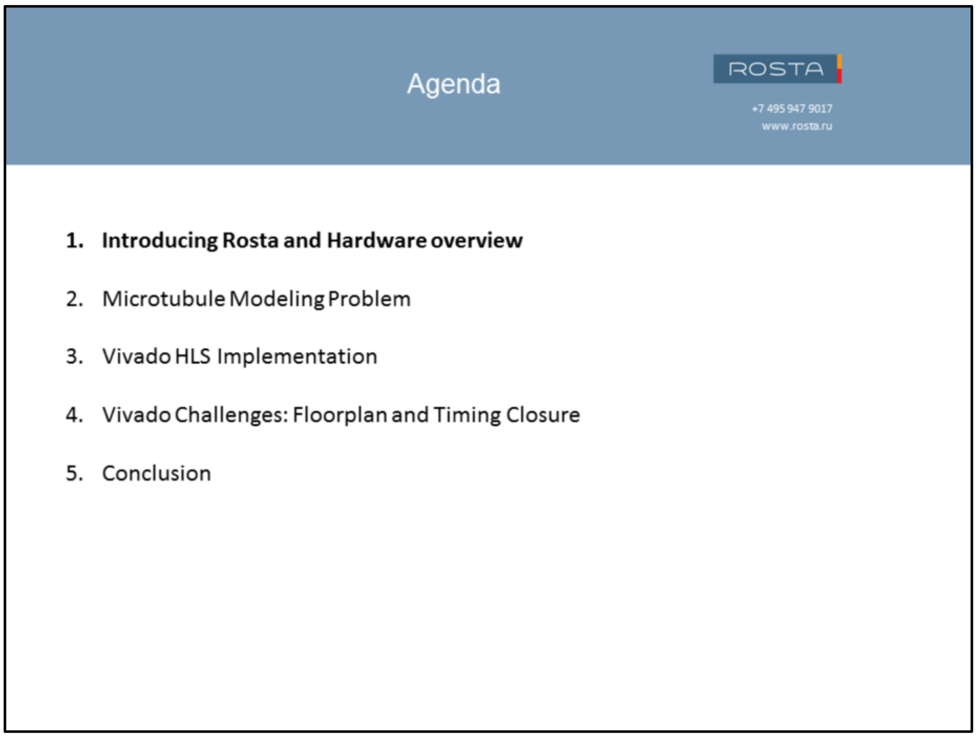
High Performance Implementation of Microtubule Modeling on FPGA using Vivado HLS

Yury Rumyantsev
rumyantsev@rosta.ru

ROSTA
+7 495 947 9017
www.rosta.ru

1

Hello my name is Yury Rumyanstev, I am FPGA and Sofware engineer at Rosta, Russian company based in Moscow. I will present details of HPC implementation of scientific computational algorithm named Microtubule modeling on FPGA using Vivado HLS tool. This algorithm belongs to molecule dynamics class and runs on multicore CPU for very long time. Using FPGA we achieved 15 times speedup.



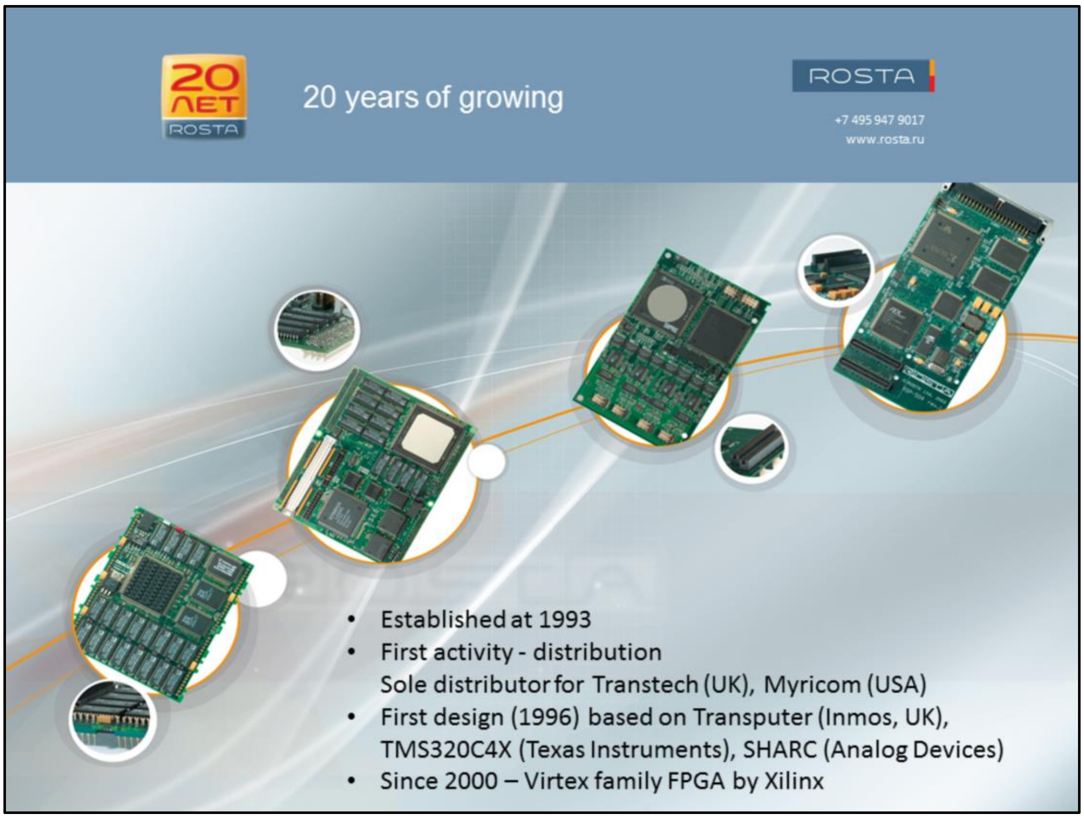
Agenda

ROSTA

+7 495 947 9017
www.rosta.ru

1. **Introducing Rosta and Hardware overview**
2. Microtubule Modeling Problem
3. Vivado HLS Implementation
4. Vivado Challenges: Floorplan and Timing Closure
5. Conclusion

My presentation will be as follows. First, I will briefly introduce Rosta, company I work for and describe hardware platform used for computation. Then I will generally talk about what is microtubule, why there is a need to model it and describe physical algorithm. After that, I will focus on Vivado HLS implementation of computational algorithm in C language. I will explain how we created computational pipelines and how we organized data to keep it running every cycle. Then I speak about challenges we faced during implementation in Vivado. And finally I will sum up my experience and give a hint on future work.



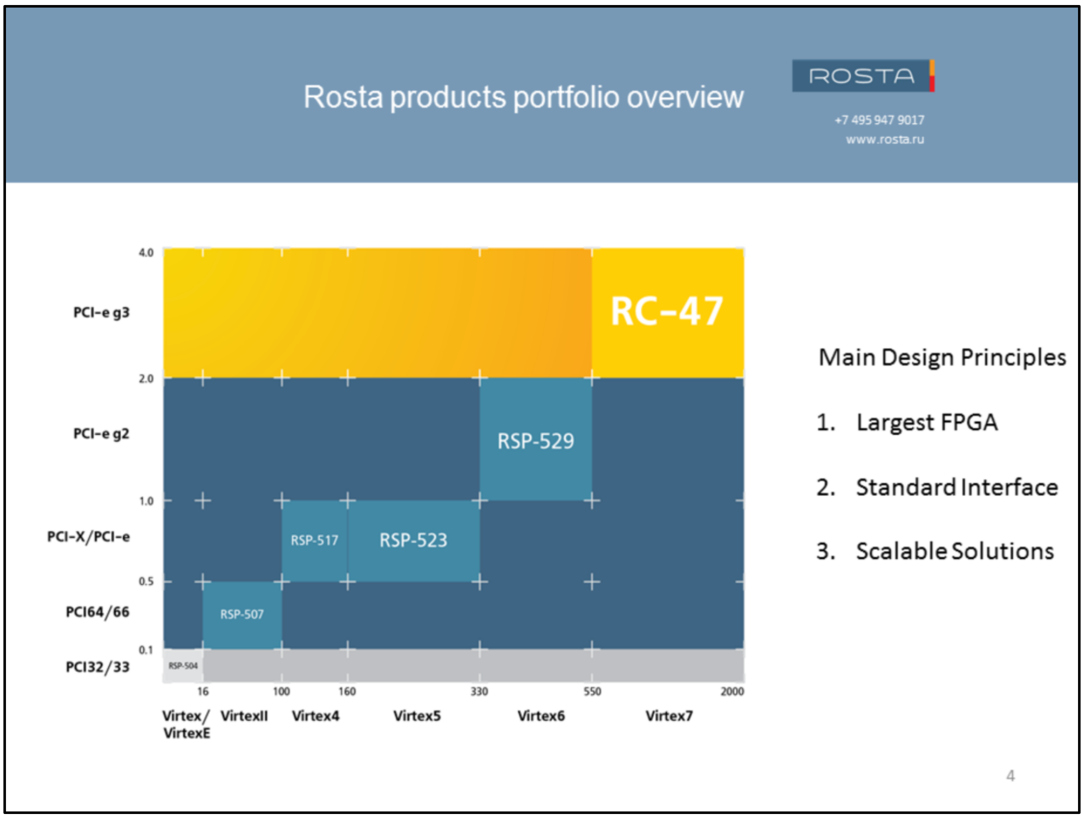
20 years of growing

ROSTA

+7 495 947 9017
www.rosta.ru

- Established at 1993
- First activity - distribution
  Sole distributor for Transtech (UK), Myricom (USA)
- First design (1996) based on Transputer (Inmos, UK), TMS320C4X (Texas Instruments), SHARC (Analog Devices)
- Since 2000 – Virtex family FPGA by Xilinx

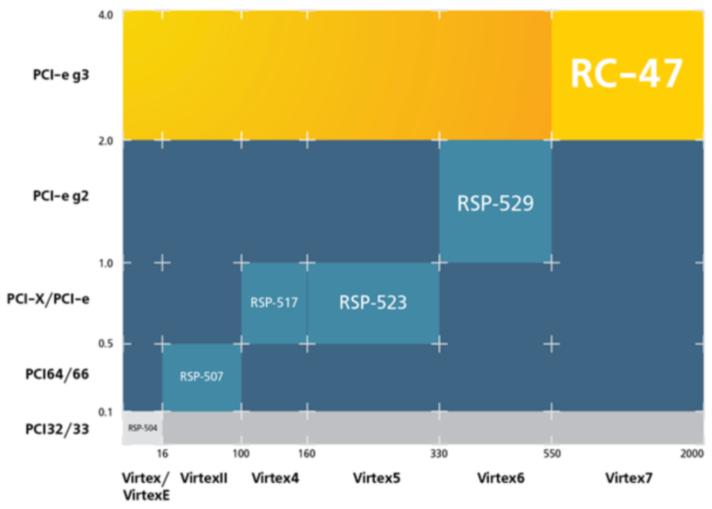
Rosta was established in 1993 and started from distribution of Transtech and Myricon technologies in Russia. Soon after that Rosta switched to design its own devices and started working with Xilinx since 2000.



Rosta products portfolio overview

ROSTA

+7 495 947 9017
www.rosta.ru

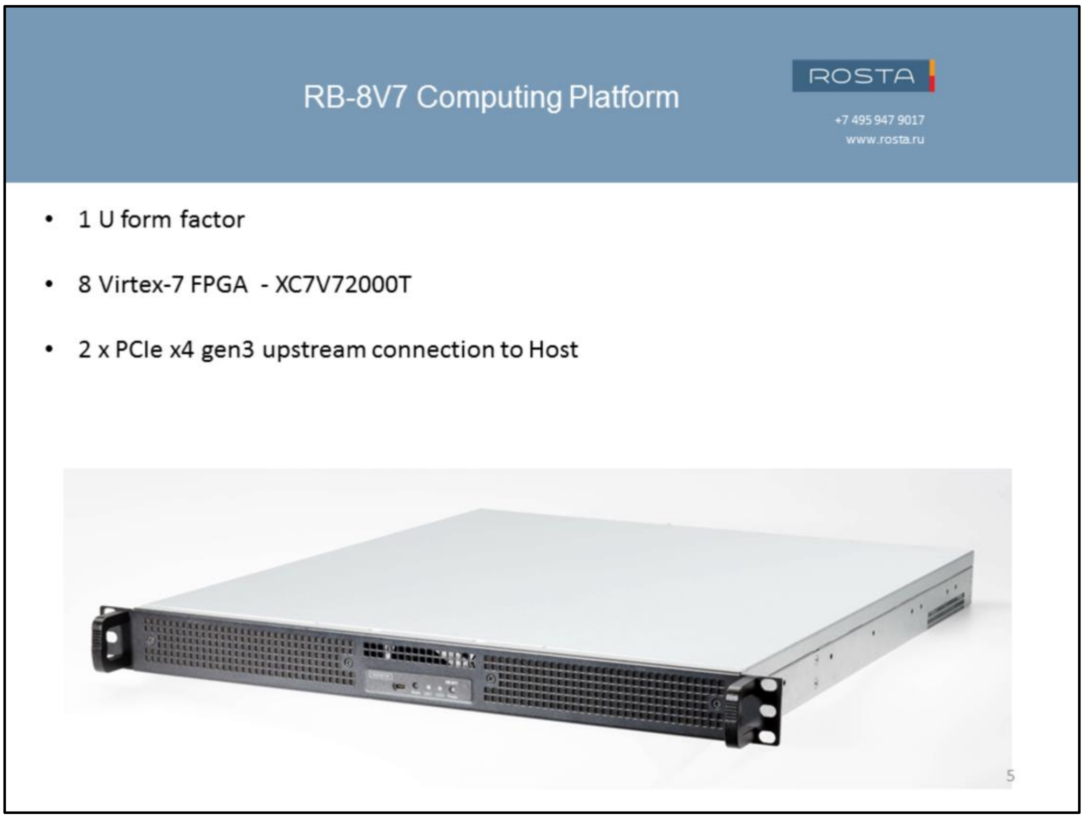


Main Design Principles

1. Largest FPGA
2. Standard Interface
3. Scalable Solutions

4

Main business was always about HPC accelerator cards. Rosta always followed Xilinx and created devices on all Virtex families also moving from PCI bus to PCI Express gen3. Our latest prototype device has Kintex UltraScale FPGA.



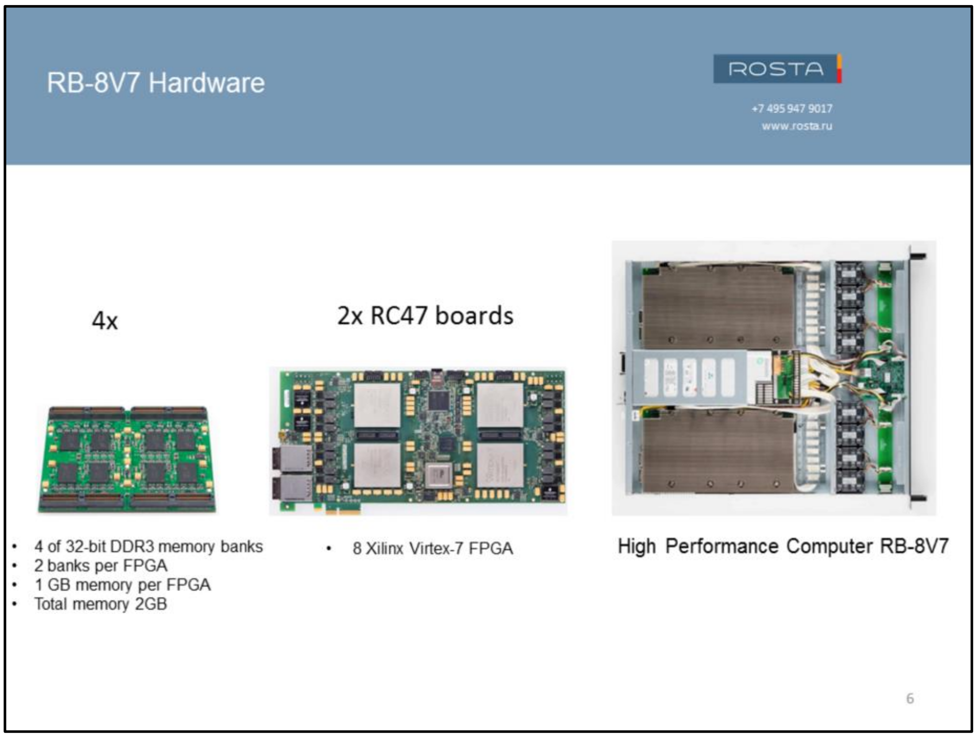
RB-8V7 Computing Platform

ROSTA
+7 495 947 9017
www.rosta.ru

- 1 U form factor
- 8 Virtex-7 FPGA - XC7V72000T
- 2 x PCIe x4 gen3 upstream connection to Host

5

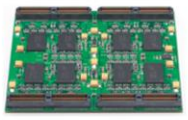
But today I'll talk about Virtex 7 computing platform. It is 1U form factor block to be installed in rack and consists of 8 Virtex-7 2000T FPGA and has PCI Express interface to host PC over optical cables.



RB-8V7 Hardware

ROSTA
+7 495 947 9017
www.rosta.ru

4x



- 4 of 32-bit DDR3 memory banks
- 2 banks per FPGA
- 1 GB memory per FPGA
- Total memory 2GB

2x RC47 boards

- 8 Xilinx Virtex-7 FPGA

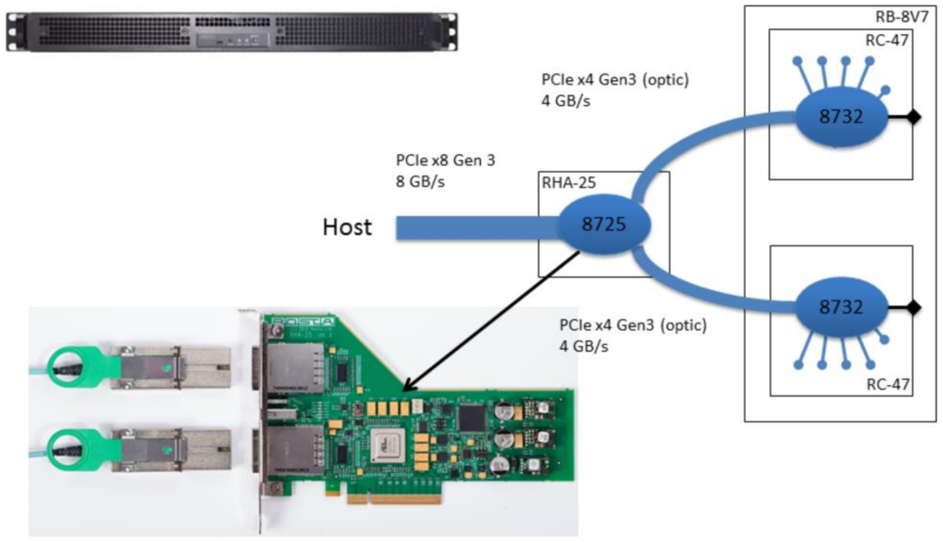
High Performance Computer RB-8V7

6

Inside there are two identical RC47 boards with 4 FPGA. Each FPGA has PCIe gen2 x4 interface to PCI Express switch. Also 1GB of DDR3 memory are connected to each FPGA.

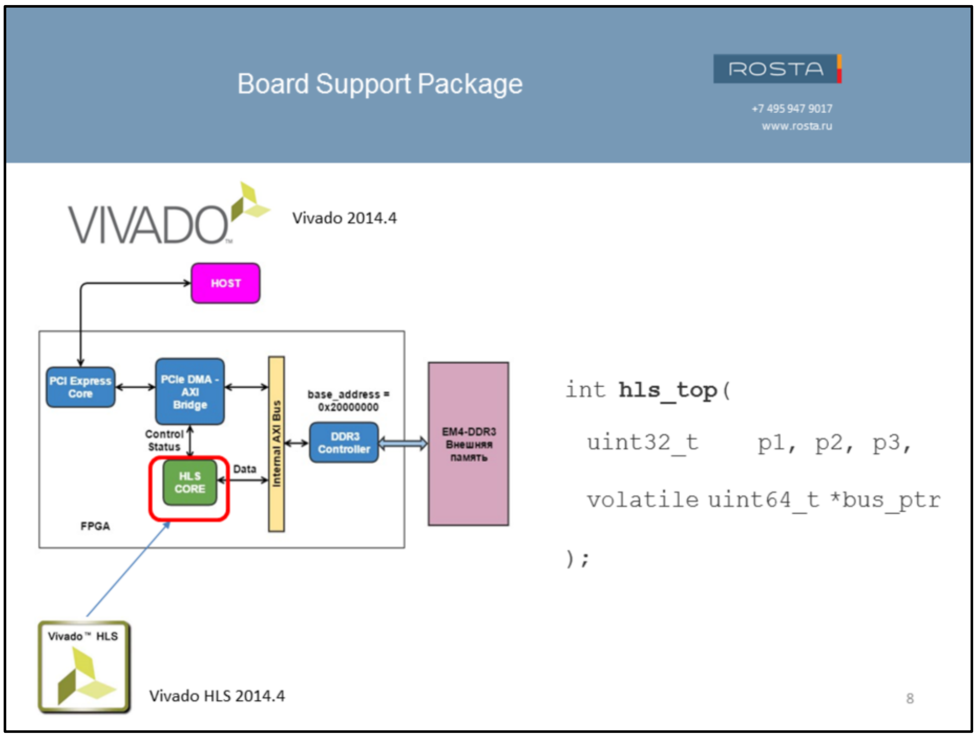
RB-8V7. Connection to Host

ROSTA
+7 495 947 9017
www.rosta.ru



7

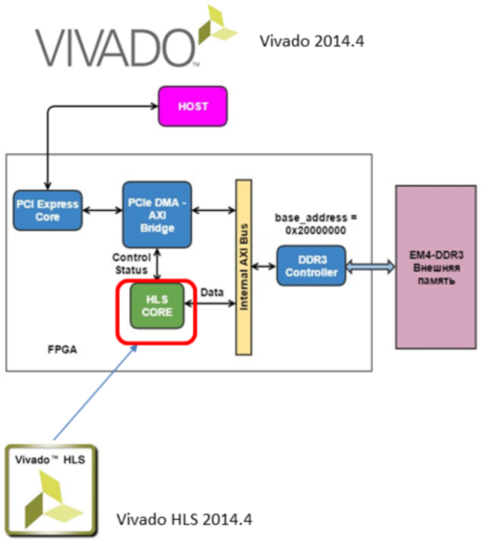
Connection to host PC is performed using two optical cables that connect two RC47 boards with PCIe adapter installed in host PC motherboard. Software running on PC sees our block as 8 independent PCIe FPGA devices.



Board Support Package

ROSTA
+7 495 947 9017
www.rosta.ru

VIVADO Vivado 2014.4

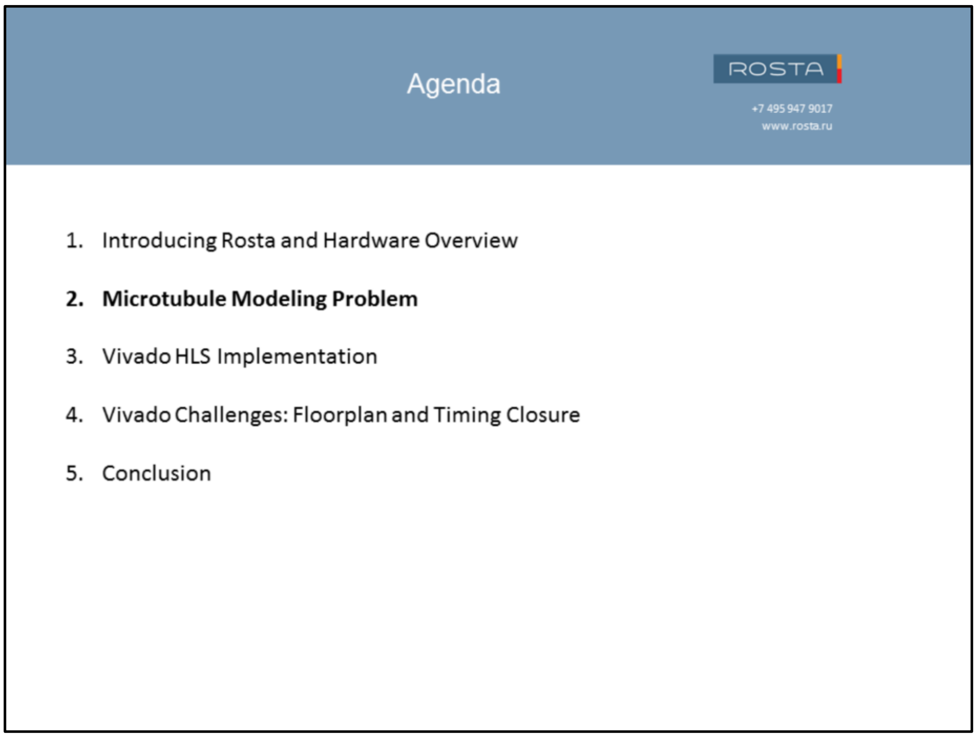


```
int hls_top(
  uint32_t    p1, p2, p3,
  volatile uint64_t *bus_ptr
);
```

Vivado HLS 2014.4

8

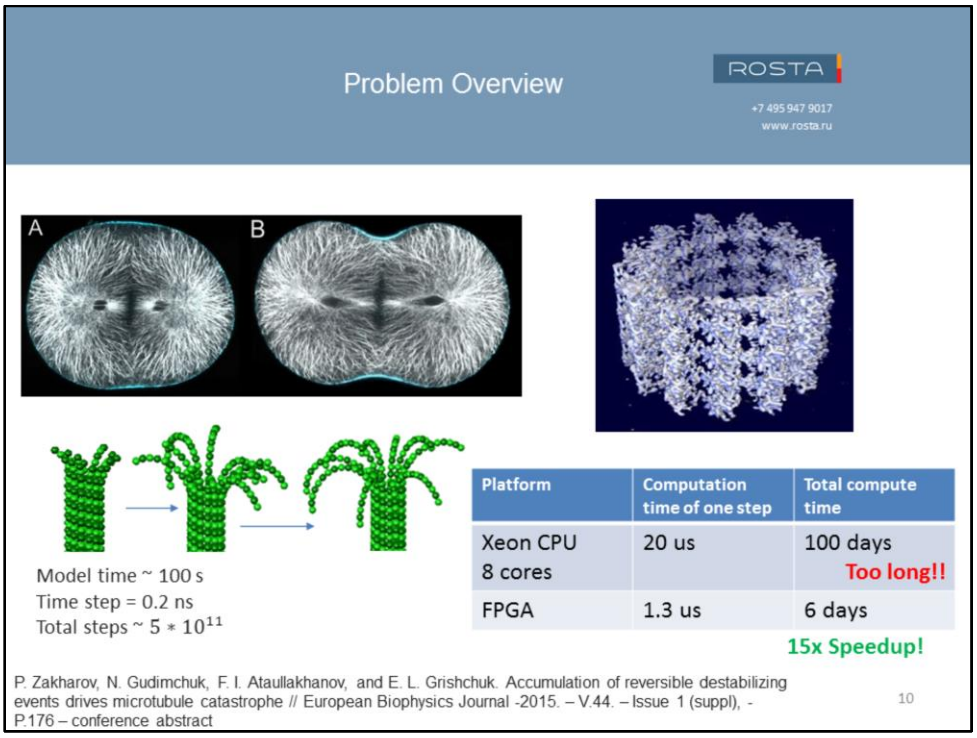
Our programming model is very close to OpenCL. We have host that is connected to FPGA Compute device. FPGA has external DDR3 memory. On FPGA side we have Board Support Package with PCIe DMA, DDR controller, Internal AXI bus and template for connecting Vivado HLS core. On top level HLS core has the following interface described in C language. Bus pointer interface is used to access DDR memory through AXI bus. Our problem size was not big so external DDR was used only for IO with host.



Agenda

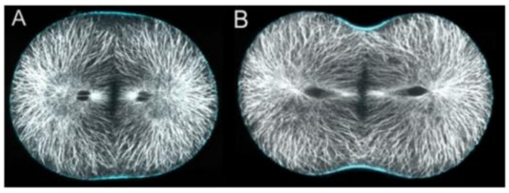
ROSTA
+7 495 947 9017
www.rosta.ru

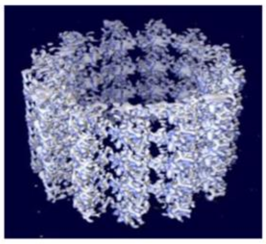
1. Introducing Rosta and Hardware Overview
2. **Microtubule Modeling Problem**
3. Vivado HLS Implementation
4. Vivado Challenges: Floorplan and Timing Closure
5. Conclusion

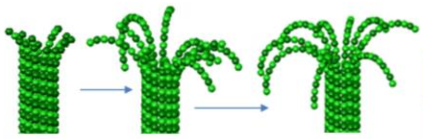


Problem Overview

ROSTA
+7 495 947 9017
www.rosta.ru







Model time ~ 100 s
Time step = 0.2 ns
Total steps ~ $5 * 10^{11}$

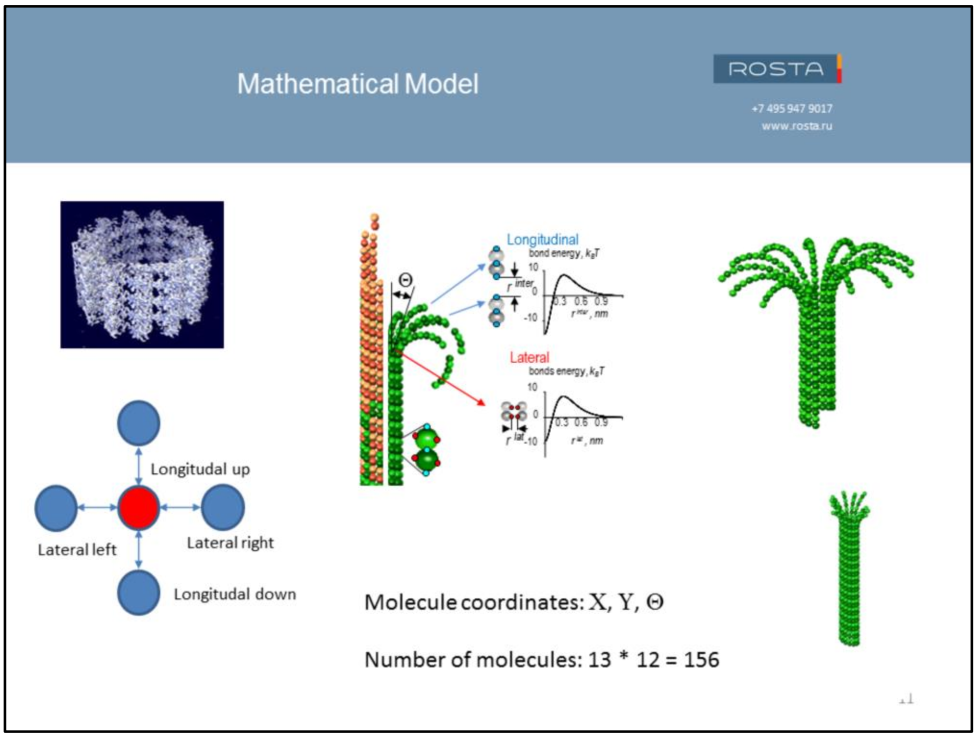
Platform	Computation time of one step	Total compute time
Xeon CPU 8 cores	20 us	100 days Too long!!
FPGA	1.3 us	6 days

15x Speedup!

P. Zakharov, N. Gudimchuk, F. I. Ataullakhanov, and E. L. Grishchuk. Accumulation of reversible destabilizing events drives microtubule catastrophe // European Biophysics Journal -2015. – V.44. – Issue 1 (suppl), - P.176 – conference abstract

10

Microtubules are molecular structures that can be found in any biological cell during its division and are represented by long cylinders, made of tubulin heterodimers. Microtubules are dynamically unstable, that is they exhibit sudden transitions between phases of slow growth and rapid shortening and vice versa.
In dividing cells dynamic microtubules are responsible for search and capture of chromosomes, their delivery to cell equator and subsequent segregation between daughter cells. Inhibition of this process can prevent cell division. This fact makes microtubules one of the most successful targets of modern anti-cancer therapy.
Our colleagues from Physics Department of Moscow State University have developed a molecular dynamic model of microtubule. Essential part of this model is that it takes into account heat Brownian motion and thus should be calculated with time step of 0.2 ns. They implemented this model using on multicore CPU. With this approach, it takes twenty microseconds of real time to model one time step of 0.2 ns. Problem is that to get new scientific data abound 100 sec of microtubule lifetime should be modeled which results in 100 days of computing! Using FPGA we managed to speedup computation 15 times, which makes FPGA more desirable candidate for such simulations.



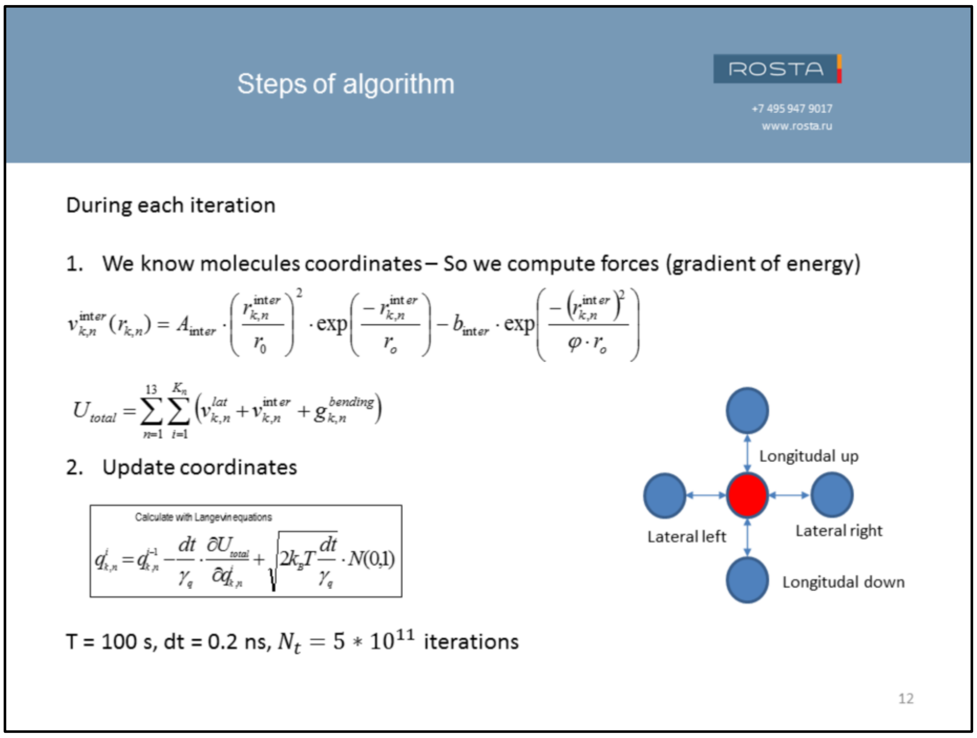
Mathematical Model

Molecule coordinates: X, Y, Θ

Number of molecules: 13 * 12 = 156

So let’s have a look at mathematical model. In the upper left corner, there is electronic photo of MT. You can see a cylinder consisting of 13 vertical lines of monomers organized into so called protofilaments.
Position and orientation of each monomer is defined by three coordinates: (x_i, y_i, τ_i), where x_i and y_i are positions of the center of the subunit and τ_i is the orientation angle. Monomers interact at four contact points with the neighboring subunits: there are two longitudal and two lateral bonds per monomer. We will address them as longitudal up and down bonds and lateral left and right bonds.

Our task was to calculate evolution of system. Have a look at animation made in MATLAB. It is built on results obtained with our final implementation. Please don’t take into account flying parts. Sometimes longitudal bonds can break and upper part of PF detaches from MT. We just had no time to update software to remove them from animation.



Steps of algorithm

ROSTA

+7 495 947 9017

www.rosta.ru

During each iteration

1. We know molecules coordinates – So we compute forces (gradient of energy)

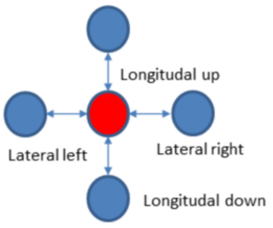
$$v_{k,n}^{\text{inter}}(r_{k,n}) = A_{\text{inter}} \cdot \left(\frac{r_{k,n}^{\text{inter}}}{r_0}\right)^2 \cdot \exp\left(\frac{-r_{k,n}^{\text{inter}}}{r_o}\right) - b_{\text{inter}} \cdot \exp\left(\frac{-\left(r_{k,n}^{\text{inter}}\right)^2}{\varphi \cdot r_o}\right)$$

$$U_{total} = \sum_{n=1}^{13} \sum_{i=1}^{K_n} \left(v_{k,n}^{lat} + v_{k,n}^{\text{inter}} + g_{k,n}^{bending}\right)$$

2. Update coordinates

Calculate with Langevin equations

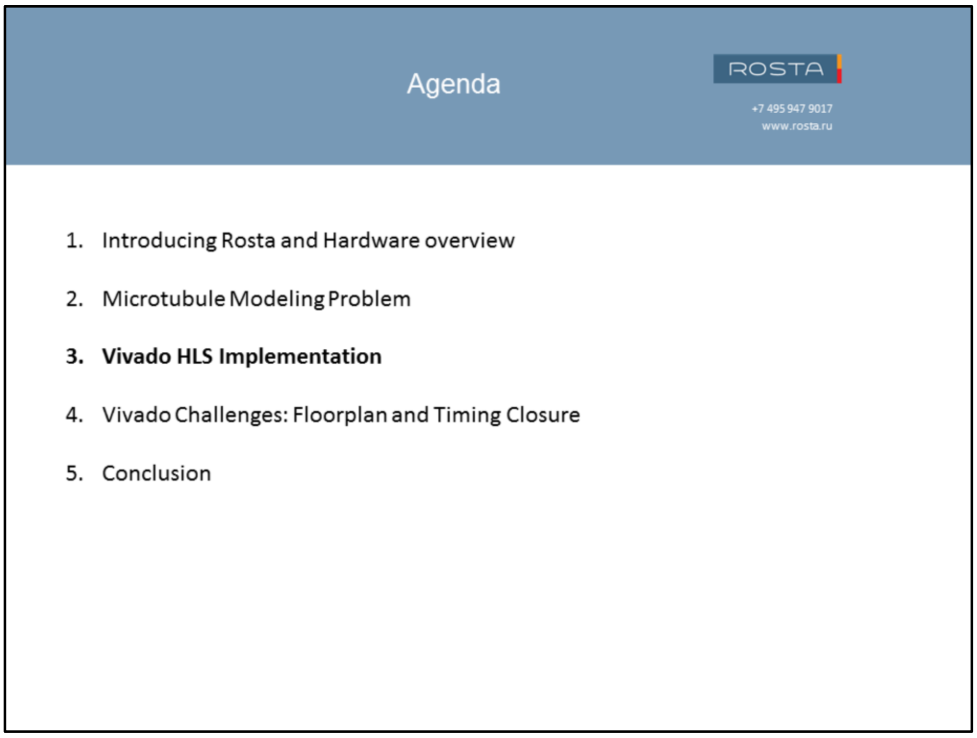
$$q_{k,n}^{i} = q_{k,n}^{i-1} - \frac{dt}{\gamma_q} \cdot \frac{\partial U_{total}}{\partial q_{k,n}^{i}} + \sqrt{2k_B T \frac{dt}{\gamma_q}} \cdot N(0,1)$$



T = 100 s, dt = 0.2 ns, $N_t = 5 * 10^{11}$ iterations

12

Let’s move on to algorithm. We have period of time divided into time steps. On each time step first we have to compute all forces between molecules. We compute them as gradients of interaction energy which depends only on coordinates. After we know forces we can update coordinates. Physical effect can be found after modeling during hundreds of seconds with step equal to 0.2 ns. It means we have to go over 5 multiply by 10 to 11 power of iterations.
Let’s take a look at coordinates update equation. This part describes deterministic part of algorithm. But our model also takes into account Brownian motion by adding normally distributed random number. We need quite a lot of random numbers. We can not afford loading them from host – in this case there will be no acceleration. Instead, we implemented pseudo random number generators inside FPGA.

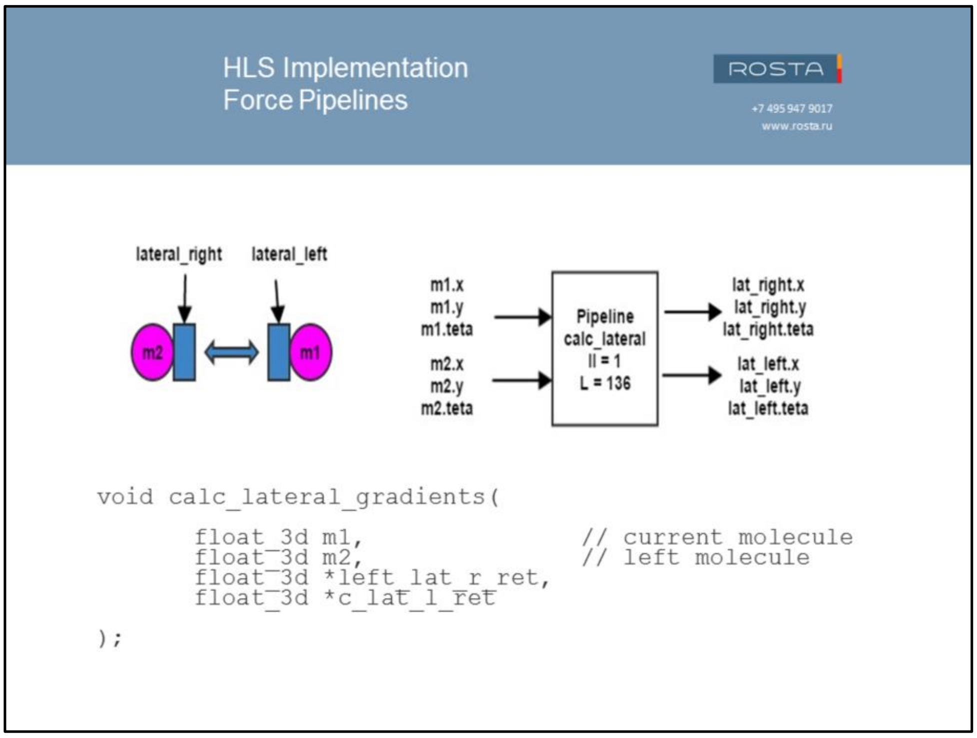


Agenda

ROSTA
+7 495 947 9017
www.rosta.ru

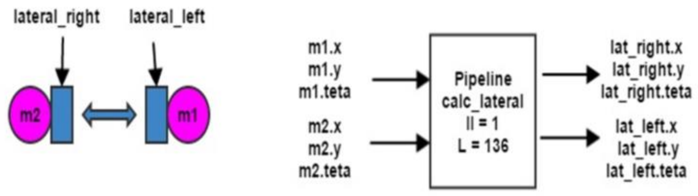
1. Introducing Rosta and Hardware overview
2. Microtubule Modeling Problem
3. **Vivado HLS Implementation**
4. Vivado Challenges: Floorplan and Timing Closure
5. Conclusion

But let’s talk first about deterministic part of algorithm and force computation.



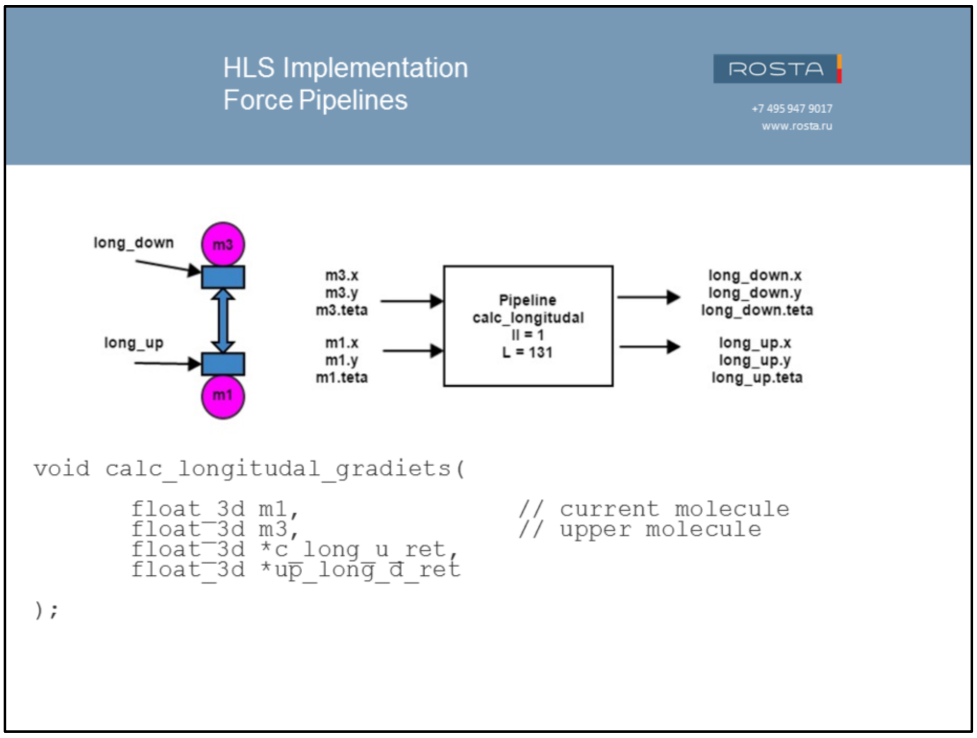
HLS Implementation
Force Pipelines

ROSTA
+7 495 947 9017
www.rosta.ru



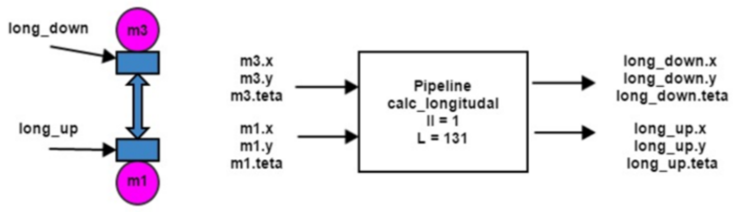
```
void calc_lateral_gradients(

       float_3d m1,                  // current molecule
       float_3d m2,                  // left molecule
       float_3d *left_lat_r_ret,
       float_3d *c_lat_l_ret

);
```

On the picture circles represent molecules and blue rectangles represent interaction forces. Force computation can be represented by C function which takes two molecules coordinates as inputs and produces two forces represented by pointer arguments as outputs. This function is synthesized as good pipeline with one cycle initiation interval and 136 cycles latency.



HLS Implementation
Force Pipelines

ROSTA
+7 495 947 9017
www.rosta.ru

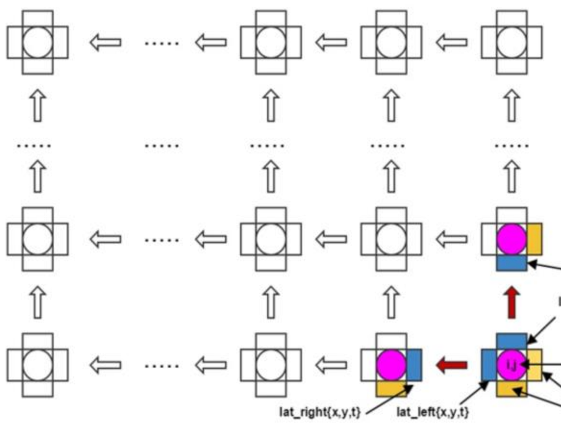


```
void calc_longitudal_gradiets(

      float_3d m1,                 // current molecule
      float_3d m3,                 // upper molecule
      float_3d *c_long_u_ret,
      float_3d *up_long_d_ret
);
```

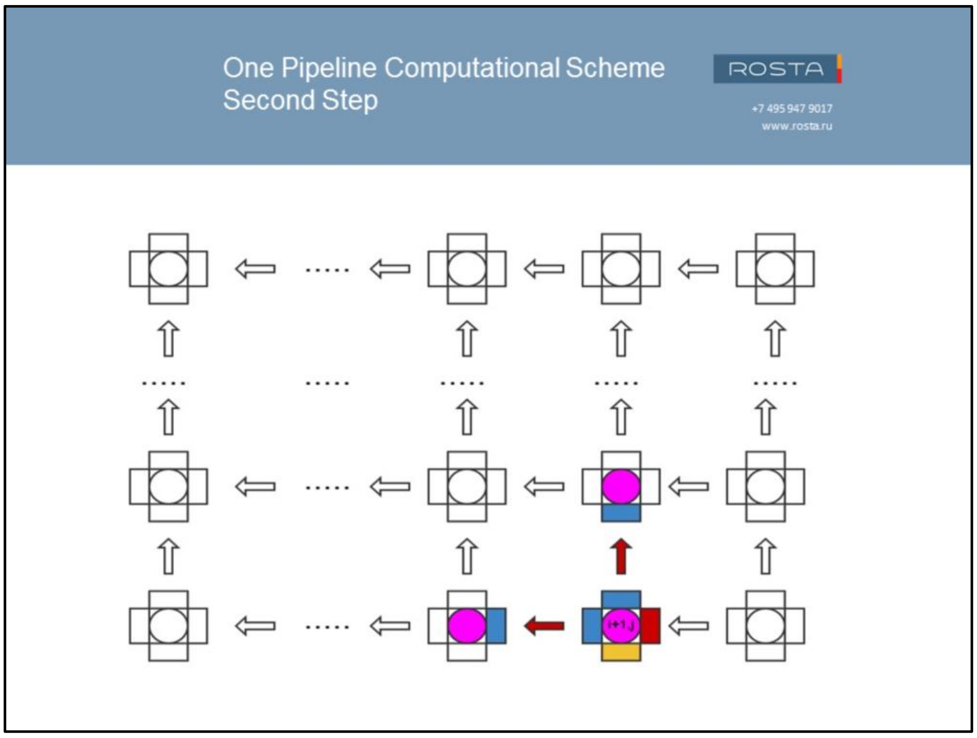
The same can be done with longitudal forces but with slightly different latency.

One Pipeline Computational Scheme
First Step

+7 495 947 9017
www.rosta.ru



Ok now we have force computation pipelines and we start pushing molecule coordinates into them. I start from the bottom right corner of microtubule. On first step I load coordinates of 3 molecules from local memory and push them to pipeline.

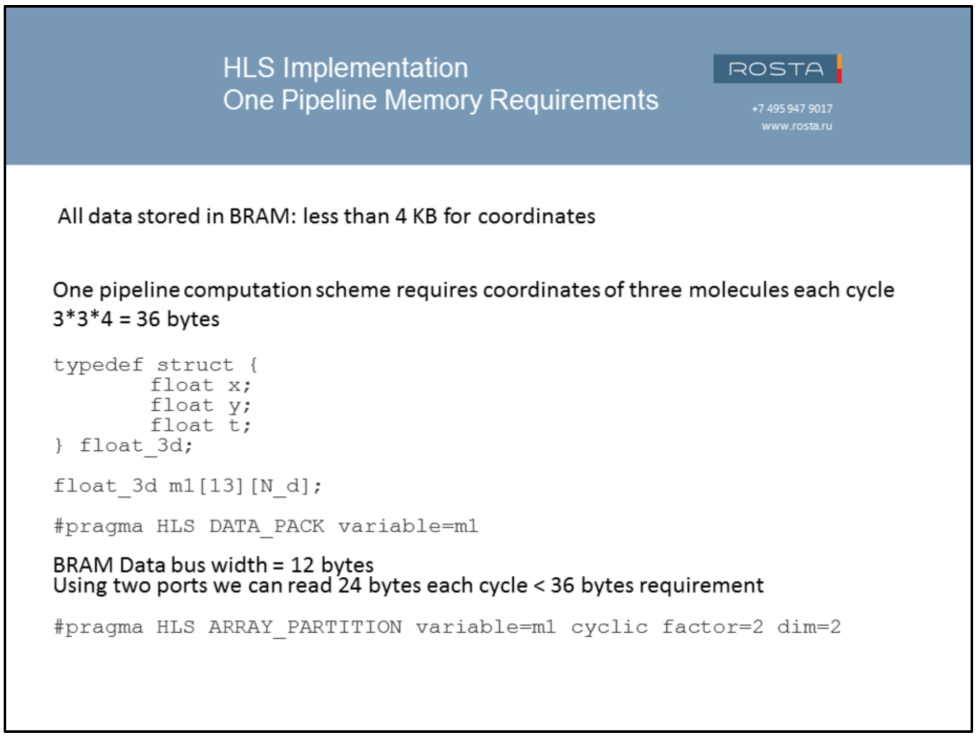


One Pipeline Computational Scheme Second Step

ROSTA

+7 495 947 9017
www.rosta.ru

On next clock cycle I load coordinates of the next 3 molecules to the left from memory and push them to pipeline. And then move on position by position from right to left. When I reach the end of the first row I switch to second row and continue. From C language point of view this is nothing but nested for loop. When after initial latency pipeline starts produce force data on its output it is used to update molecule coordinates. On bottom row I have boundary condition (represented by yellow rectangles), down force components of these molecules are zero. To update coordinates of current molecule we need four components of forces around it. Computed forces for neighboring molecules are saved and passed to next computational steps. For example this lateral right force value will be used to update coordinates of molecule on the left on next cycle. On this slide this force is red.



HLS Implementation
One Pipeline Memory Requirements

ROSTA
+7 495 947 9017
www.rosta.ru

All data stored in BRAM: less than 4 KB for coordinates

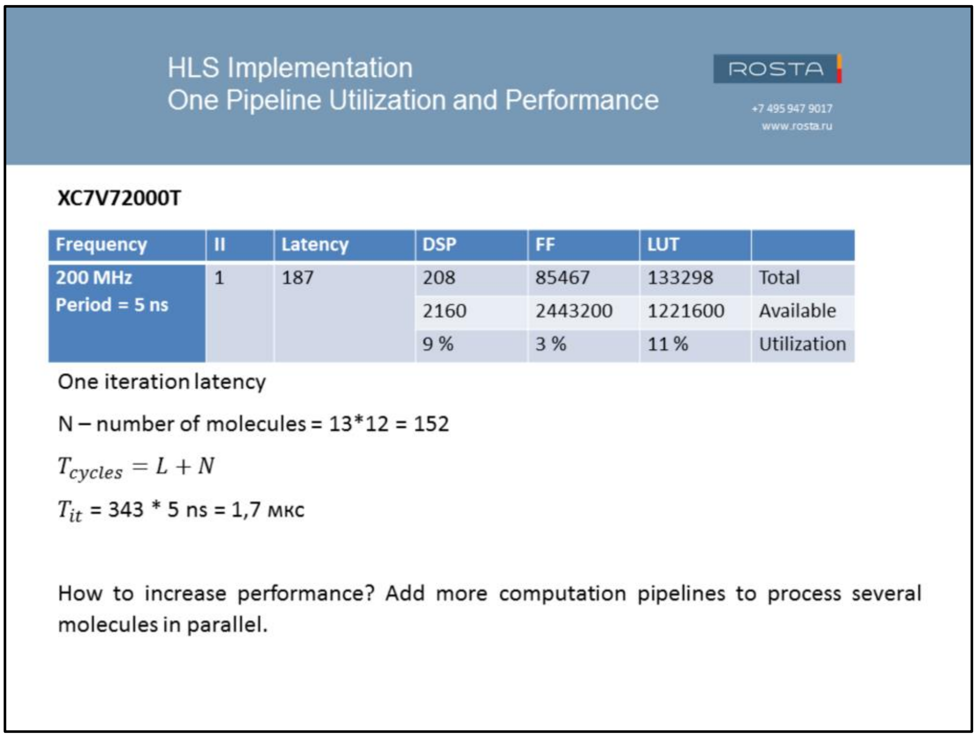
One pipeline computation scheme requires coordinates of three molecules each cycle
3*3*4 = 36 bytes

```
typedef struct {
        float x;
        float y;
        float t;
} float_3d;

float_3d m1[13][N_d];

#pragma HLS DATA_PACK variable=m1
```

BRAM Data bus width = 12 bytes
Using two ports we can read 24 bytes each cycle < 36 bytes requirement

```
#pragma HLS ARRAY_PARTITION variable=m1 cyclic factor=2 dim=2
```

Now we have pipeline, let’s see how to keep it busy. Every time iteration pipeline updates all molecule coordinates. We have relatively small number of molecules so all coordinates are stored in local BRAM. We need to load from BRAM coordinates of three molecules every cycle. Coordinates are stored in the following data type structure in two dimensional array. Using Vivado HLS DATA PACK directive I have concatenated all structure fields into one word 12 bytes wide. VHSL array is implemented as dual port BRAM so I can read 24 bytes each cycle, but my pipeline need 36 bytes. So I have used ARRAY PARTITION directive to split data into two different BRAM cores to access it in parallel.


HLS Implementation
One Pipeline Utilization and Performance

ROSTA
+7 495 947 9017
www.rosta.ru

XC7V72000T

Frequency	II	Latency	DSP	FF	LUT	
200 MHz Period = 5 ns	1	187	208	85467	133298	Total
			2160	2443200	1221600	Available
			9 %	3 %	11 %	Utilization

One iteration latency

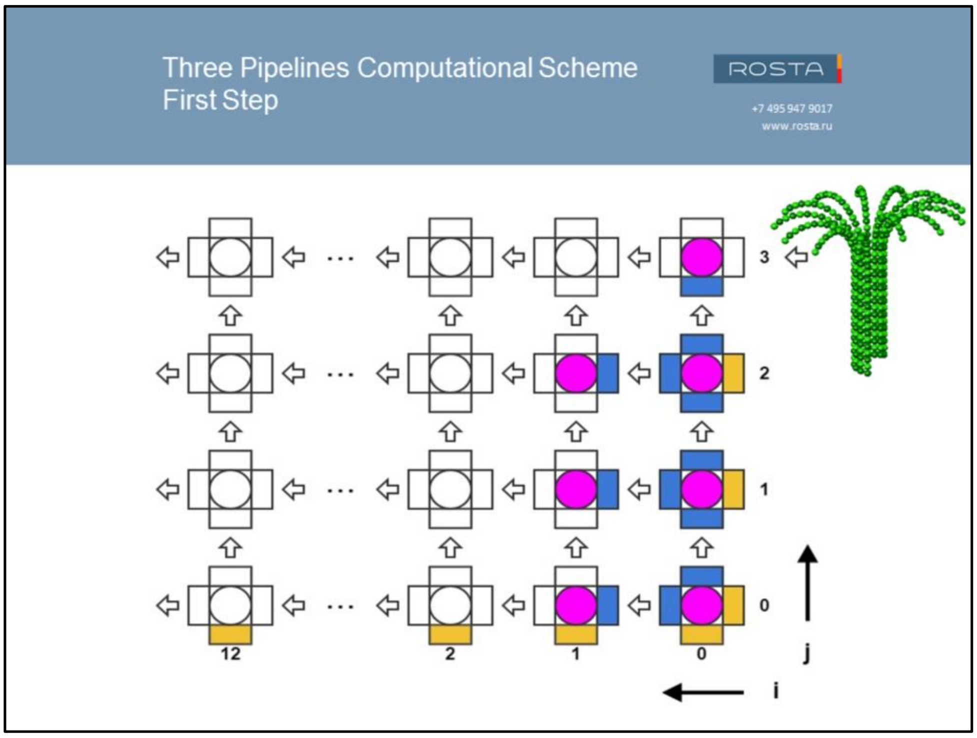
N – number of molecules = 13*12 = 152

$T_{cycles} = L + N$

T_{it} = 343 * 5 ns = 1,7 мкс

How to increase performance? Add more computation pipelines to process several molecules in parallel.

After synthesis I got the following utilization. HLS core used 9% of DSP, 3% of FF and 11 %of LUT of Virtex7 2000t silicon.
As to performance, key value here is time iteration latency. It equals to sum of pipeline latency and number of molecules. Latency was equal to 187 cycles, number of molecules was 152. The pipleline was running at 200 MHz so it took 1.7 micosecond to compute one time iteration.
We can see that there is a lot of logic left untouched in silicon so I could increase performance by adding more computational pipelines to process several molecules in parallel.



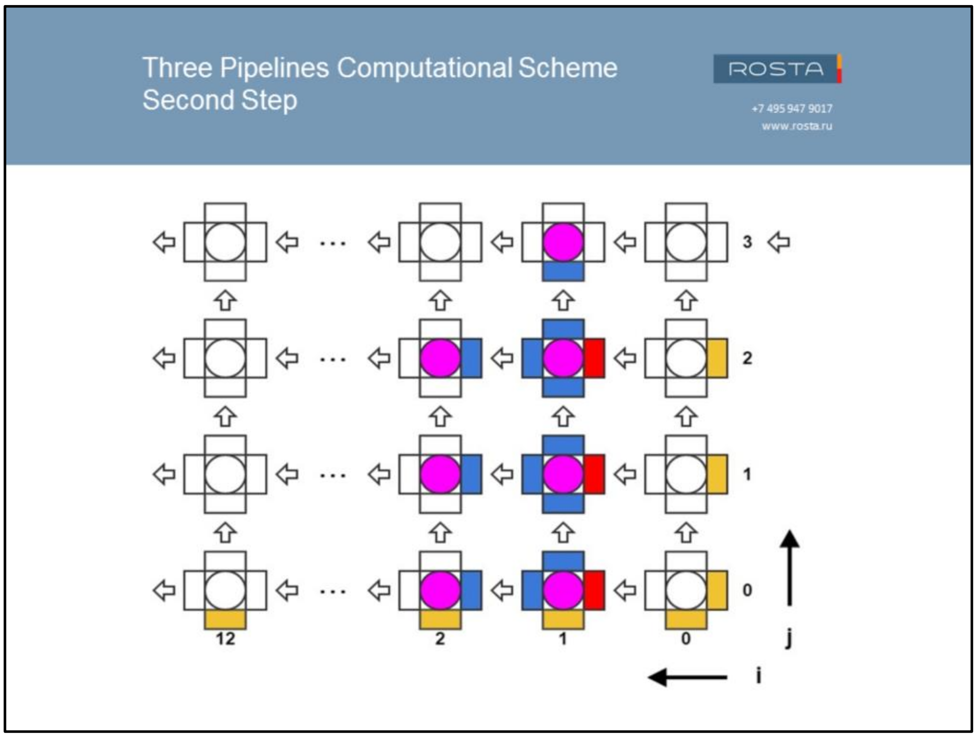
Three Pipelines Computational Scheme First Step

ROSTA

+7 495 947 9017

www.rosta.ru

From point of view of the algorithm it is possible to compute ALL molecule coordinates in parallel. But It will require too much hardware. I chose to limit my scheme to 3 pipelines to compute 3 molecules in parallel.
Again, I start from the bottom right corner of mictotubule. I load 7 molecule coordinates from BRAM and push them to three pipelines, these 3 molecules to first pipeline, these three to second and these 3 to third.

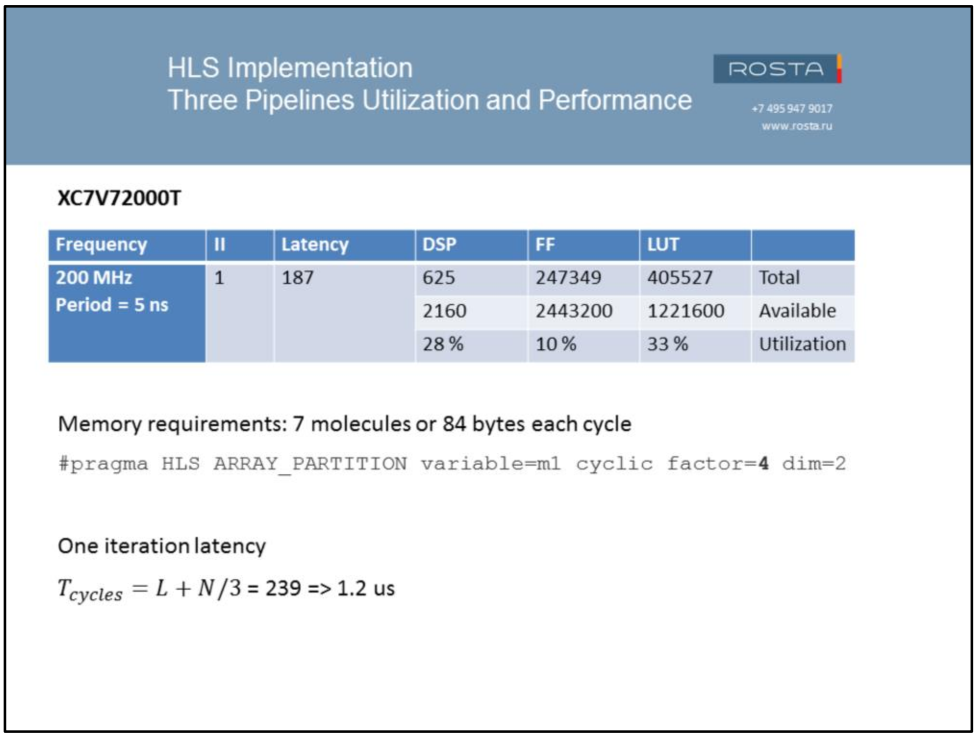


Three Pipelines Computational Scheme Second Step

ROSTA

+7 495 947 9017
www.rosta.ru

On next clock cycle I advance one molecule to the left just as with one pipeline scheme.



HLS Implementation
Three Pipelines Utilization and Performance

ROSTA
+7 495 947 9017
www.rosta.ru

XC7V72000T

Frequency	II	Latency	DSP	FF	LUT	
200 MHz Period = 5 ns	1	187	625	247349	405527	Total
			2160	2443200	1221600	Available
			28 %	10 %	33 %	Utilization

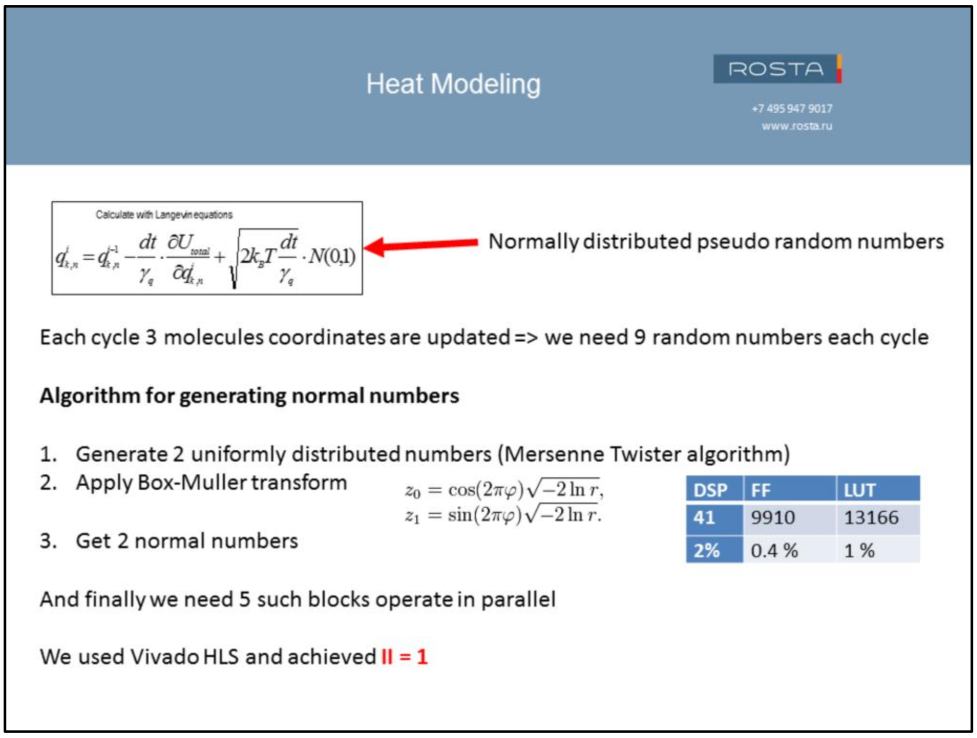
Memory requirements: 7 molecules or 84 bytes each cycle

```
#pragma HLS ARRAY_PARTITION variable=m1 cyclic factor=4 dim=2
```

One iteration latency

$T_{cycles} = L + N/3$ = 239 => 1.2 us

As expected, this computational scheme requires 3 times more hardware. Pipeline latency remains the same. Major difference is that now I need more bandwidth to local BRAM as compute scheme requires coordinates of 7 molecules every cycle. So I changed the parameter in ARRAY PARTITION directive and split my array into 4 different BRAM cores in cyclic fashion.
Now iteration latency is represented by this formula. There is still latency of pipeline, but now each of three pipeline should process 3 times less molecules. This leads to 1.2 microseconds per time iteration.
Theoretically I could go further and try to implement 6 or even more pipelines so long I have spare hardware. But it won’t add much performance because of initial pipeline latency that will still be there. It’ll be impossible to get iteration latency less then 1 microsecond, but instead there will be more timing issues during implementation in Vivado. So I decided that I was happy with 1.2 microseconds with is already 15 times faster than original implementation on muticore CPU.



Heat Modeling

ROSTA
+7 495 947 9017
www.rosta.ru

Calculate with Langevin equations

$$q^{j}_{k,n} = q^{j-1}_{k,n} - \frac{dt}{\gamma_q} \cdot \frac{\partial U_{total}}{\partial q^{j}_{k,n}} + \sqrt{2k_B T \frac{dt}{\gamma_q}} \cdot N(0,1)$$

Normally distributed pseudo random numbers

Each cycle 3 molecules coordinates are updated => we need 9 random numbers each cycle

Algorithm for generating normal numbers

1. Generate 2 uniformly distributed numbers (Mersenne Twister algorithm)
2. Apply Box-Muller transform
3. Get 2 normal numbers

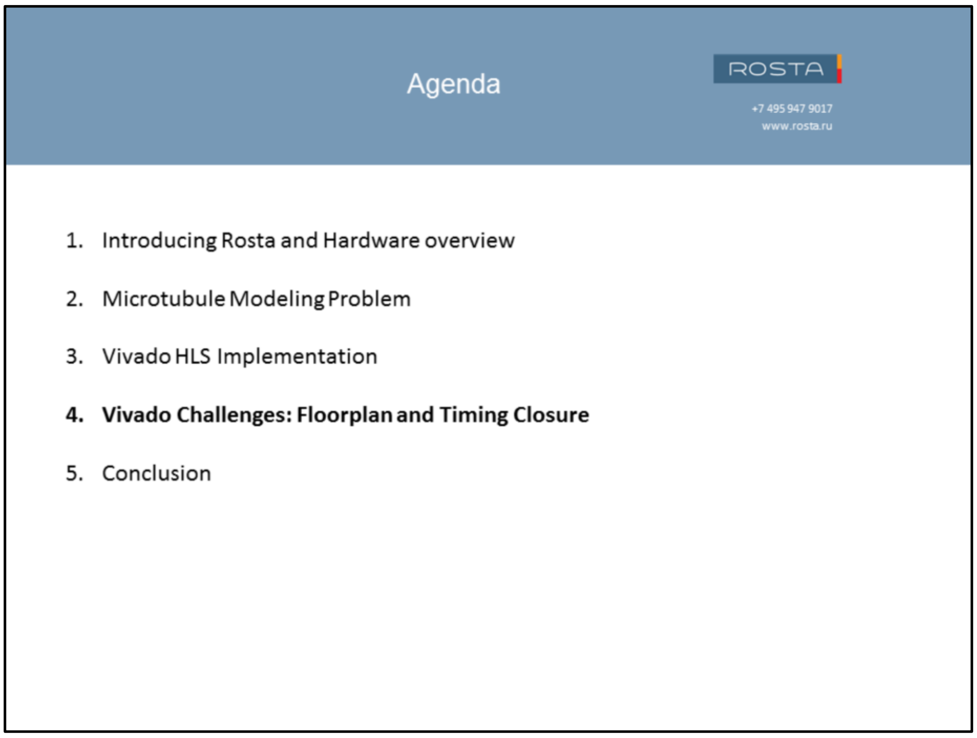
$$z_0 = \cos(2\pi\varphi)\sqrt{-2\ln r},$$
$$z_1 = \sin(2\pi\varphi)\sqrt{-2\ln r}.$$

DSP	FF	LUT
41	9910	13166
2%	0.4 %	1 %

And finally we need 5 such blocks operate in parallel

We used Vivado HLS and achieved **II = 1**

Final challenge on Vivado HLS level was to account for Brownian motion. It is modeled as small random addition to deterministic part of coordinates update. This random addition is normally distributed random number multiplied by constant. Each cycle 3 molecules coordinates are updated so we need 9 random numbers per clock cycle. We generated pseudo random numbers inside FPGA using the following algorithm.
First generate two uniformly distributed numbers, second apply Box-Muller transform and get two normal numbers. We used Mersenne Twister algorithm for generating uniformly distributed numbers. It is complex and proven algorithm with huge period. It uses internal memory buffer. To get new number algorithm reads 3 words from it and writes one word back. We had hard times to make Vivado HLS fit these operations in one clock cycle but finally managed to do that. After that it was straightforward conversion using these formulas.
One such block after synthesis occupied 2% of DSP resources 0.4 % of FF, and 1 % of LUT. We needed 5 such blocks.

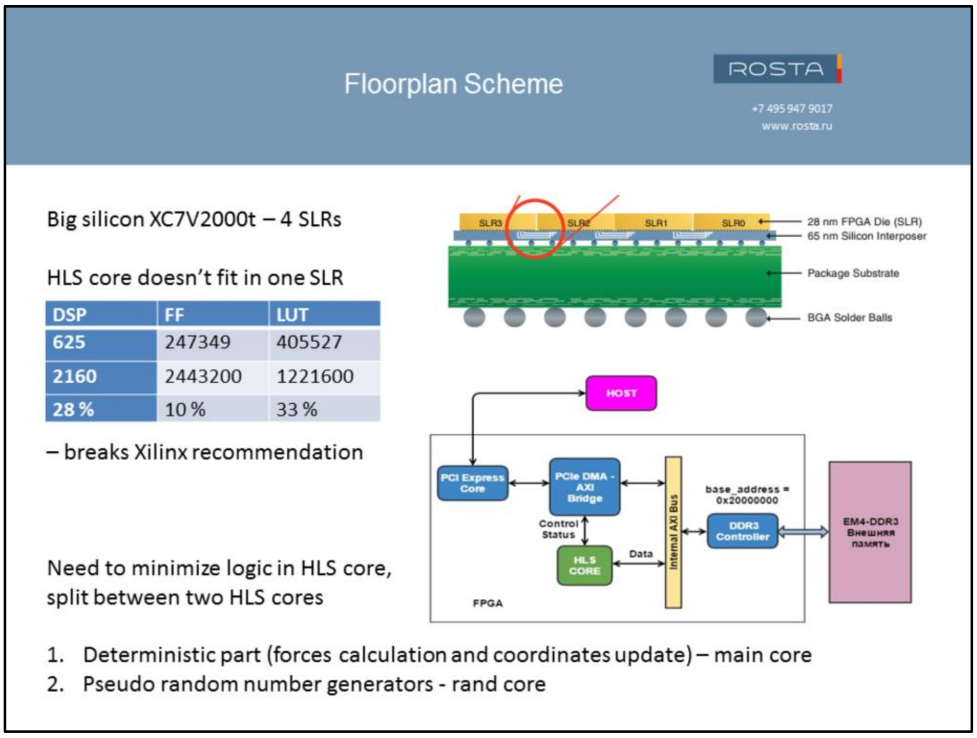


Agenda

ROSTA
+7 495 947 9017
www.rosta.ru

1. Introducing Rosta and Hardware overview
2. Microtubule Modeling Problem
3. Vivado HLS Implementation
4. **Vivado Challenges: Floorplan and Timing Closure**
5. Conclusion

That is all I have to say about mathematical aspect of problem and implementation in Vivado HLS. At this point we had synthesized working FPGA model which was verified against golden results obtained with CPU program. Our next step was to implement this model on FPGA using our Board Support Package.



Floorplan Scheme

ROSTA

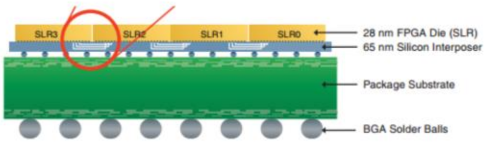
+7 495 947 9017
www.rosta.ru

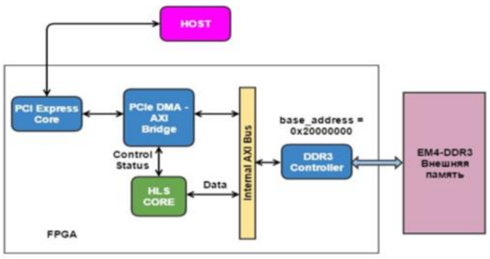
Big silicon XC7V2000t – 4 SLRs

HLS core doesn't fit in one SLR

DSP	FF	LUT
625	247349	405527
2160	2443200	1221600
28 %	10 %	33 %

– breaks Xilinx recommendation





Need to minimize logic in HLS core, split between two HLS cores

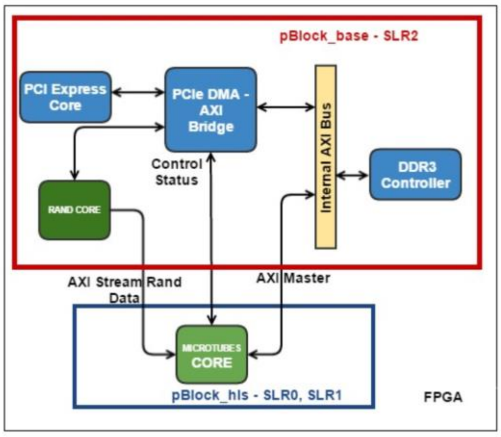
1. Deterministic part (forces calculation and coordinates update) – main core
2. Pseudo random number generators - rand core

Let me remind that we have big silicon Virtex7 2000T FPGA that consists of 4 Super Logic Regions – separate silicon dies connected over interposer. Inside we have PCIe DMA, external memory controller, AXI bus and template for connecting Vivado HLS core. Pure BSP with small HLS core consumes little logic. The main problem was to implement HLS core.

Only deterministic part of algorithm without random number core utilizes lots of recourses – 28% of DSP blocks and 33% of LUTs. This core cannot fit into one SLR, which is Xilinx recommendation in fact. My first attempt was to implement only deterministic part in hardware. And it actually worked. But after I added random number generation timing score was very bad and hardware failed to operate properly. I understood that I need to separate HLS cores and floorplan my design. I come up with the idea to split deterministic and random number generation parts into separate HLS cores and put them into different logic regions.

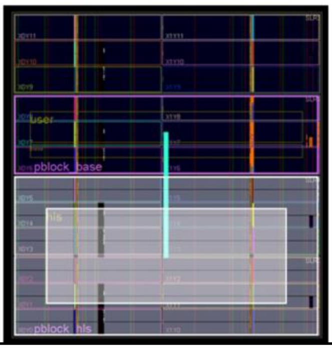
Floorplan Scheme

ROSTA
+7 495 947 9017
www.rosta.ru



pblock_base – PCIe DMA, DDR3 controller, Rand HLS core – **SLR2**

pblock_hls – Main HLS core **SLR0 + SLR1**

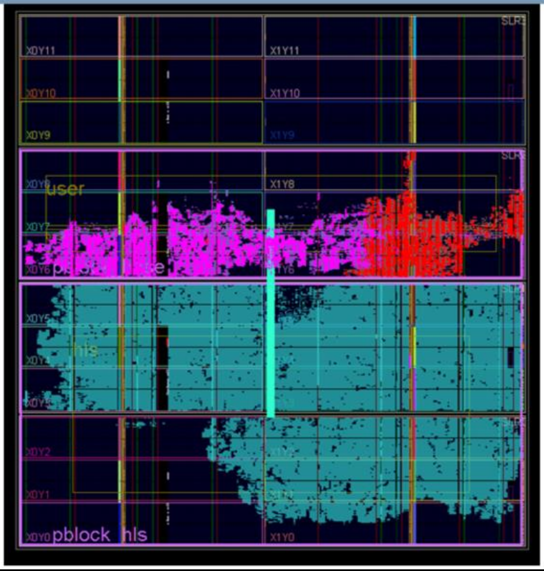


So I chose the following scheme. I created two pBlocks: pBlock_base and pblock_hls. I put PCIe, DDR and random numbers logic into pBlock_base and constrained it to logic region two. pBlock_hls that contained main hls core was constrained to logic regions zero and one.

Floorplan Scheme
Implementation Results

ROSTA

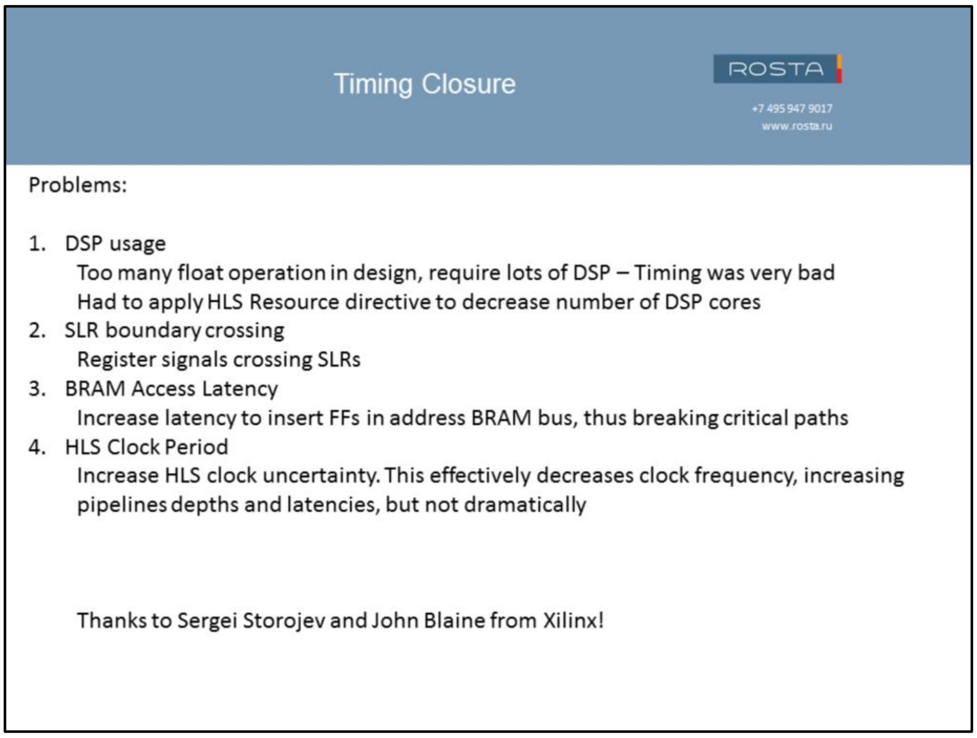
+7 495 947 9017
www.rosta.ru



RED
PCIe DMA, DDR3 controller
PURPLE
Rand HLS core
CYAN
Main HLS core

DSP	BRAM	Slices	Power
28 %	7%	38 %	12 W

On this slide you can see implementation results. RED is PCIe DMA and DDR logic, purple is random hls core and cyan is main hls core.
This approach proved to work in hardware but there was more to it than just floorplanning to get it working.



Timing Closure

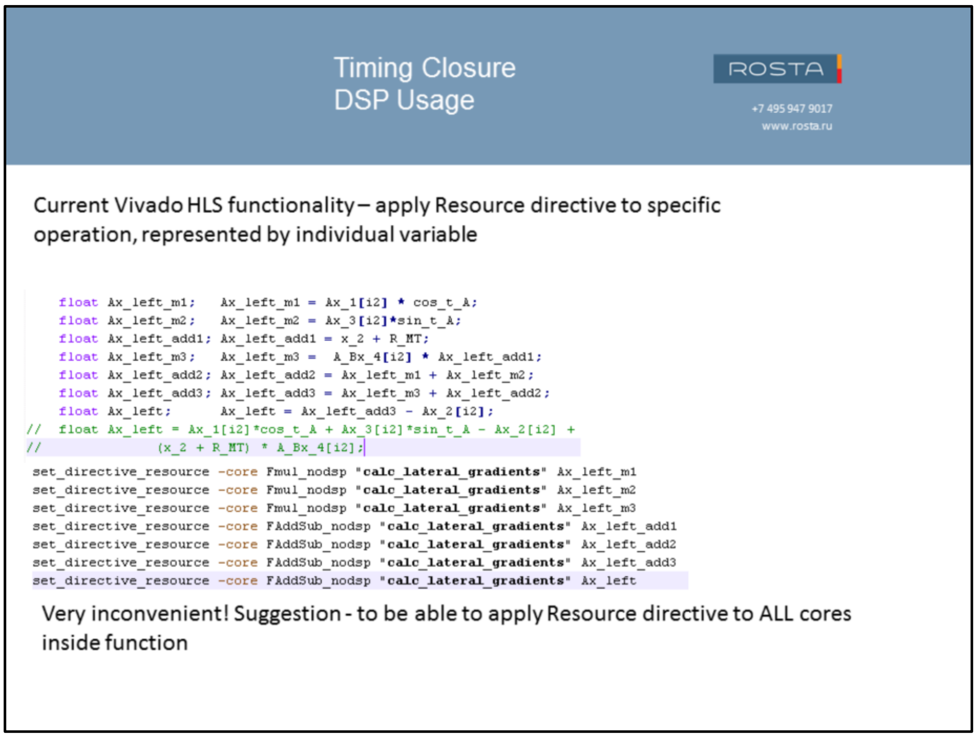
ROSTA
+7 495 947 9017
www.rosta.ru

Problems:

1. DSP usage
   Too many float operation in design, require lots of DSP – Timing was very bad
   Had to apply HLS Resource directive to decrease number of DSP cores
2. SLR boundary crossing
   Register signals crossing SLRs
3. BRAM Access Latency
   Increase latency to insert FFs in address BRAM bus, thus breaking critical paths
4. HLS Clock Period
   Increase HLS clock uncertainty. This effectively decreases clock frequency, increasing pipelines depths and latencies, but not dramatically

Thanks to Sergei Storojev and John Blaine from Xilinx!

To get things done and obtain repetitive results we faced the following issues.
On this slide I want to thank Xilinx and personally Sergei Storojev and John Blaine for helping with design timing closure.
Lets discuss first three issues in more details.



Timing Closure
DSP Usage

ROSTA

+7 495 947 9017
www.rosta.ru

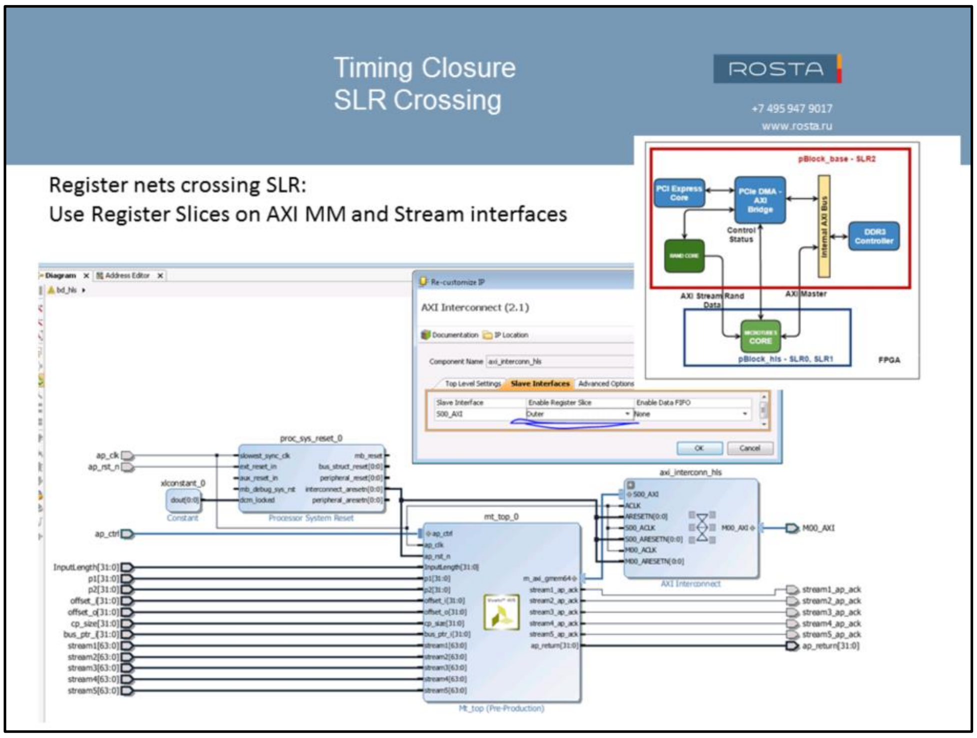
Current Vivado HLS functionality – apply Resource directive to specific operation, represented by individual variable

```
    float Ax_left_m1;   Ax_left_m1 = Ax_1[i2] * cos_t_A;
    float Ax_left_m2;   Ax_left_m2 = Ax_3[i2]*sin_t_A;
    float Ax_left_add1; Ax_left_add1 = x_2 + R_MT;
    float Ax_left_m3;   Ax_left_m3 =  A_Bx_4[i2] * Ax_left_add1;
    float Ax_left_add2; Ax_left_add2 = Ax_left_m1 + Ax_left_m2;
    float Ax_left_add3; Ax_left_add3 = Ax_left_m3 + Ax_left_add2;
    float Ax_left;      Ax_left = Ax_left_add3 - Ax_2[i2];
//  float Ax_left = Ax_1[i2]*cos_t_A + Ax_3[i2]*sin_t_A - Ax_2[i2] +
//                (x_2 + R_MT) * A_Bx_4[i2];
```

```
set_directive_resource -core Fmul_nodsp "calc_lateral_gradients" Ax_left_m1
set_directive_resource -core Fmul_nodsp "calc_lateral_gradients" Ax_left_m2
set_directive_resource -core Fmul_nodsp "calc_lateral_gradients" Ax_left_m3
set_directive_resource -core FAddSub_nodsp "calc_lateral_gradients" Ax_left_add1
set_directive_resource -core FAddSub_nodsp "calc_lateral_gradients" Ax_left_add2
set_directive_resource -core FAddSub_nodsp "calc_lateral_gradients" Ax_left_add3
set_directive_resource -core FAddSub_nodsp "calc_lateral_gradients" Ax_left
```

Very inconvenient! Suggestion - to be able to apply Resource directive to ALL cores inside function

DSP usage. In my code there were constructions like that. You see 4 add sub and 3 multiply operations on one line of C code. Each operation requires separate hardware block which will be implemented using Xilinx floating point IP core. This IP has parameter that controls DSP block usage for it. And Vivado HLS has resource directive that can be used to control this parameter. In my case I didn't want to use DSP recourses at all so I had to apply resource directive with nodsp parameter. Also current Vivado HLS functionality is to apply resource directive to specific operation represented by individual variable, so I had to rewrite my code by hand and get 7 lines instead of 1. Also I had to apply 7 different directives for different variables. Actually It is very inconvenient. My code was long and I spent a lot of time to rewrite it in one operation – one variable fashion. And here I have suggestion to be able to apply resource directive to all cores inside a function.



Timing Closure
SLR Crossing

ROSTA

+7 495 947 9017

www.rosta.ru

Register nets crossing SLR:

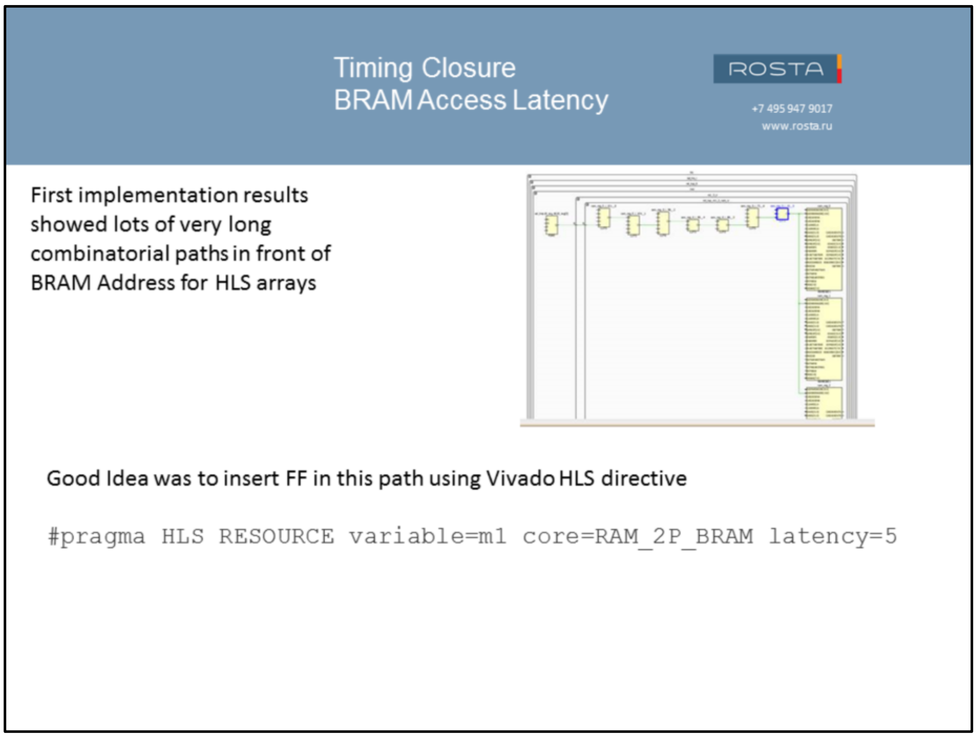
Use Register Slices on AXI MM and Stream interfaces

AXI Interconnect (2.1)

Component Name axi_interconn_hls

Slave Interface	Enable Register Slice	Enable Data FIFO
S00_AXI	Outer	None

Another problem was to deal with SLR crossing nets. . Here you see Block Design that was used to wrap main hls core. Random generator hls core and ddr3 controller were located in different logic regions. There were nets that crossed SLR boundary. These are AXI Master bus nets that were used by hls core to access external memory and AXI Stream nets that were used to pass random numbers from one hls core to another. All nets should be registered on both sides of SLR boundary For AXI Master nets it was easy to do using Enable Register Slice option of AXI Interconnect IP. And for AXI Stream nets I had to insert different IP called AXI Stream Register Slice on RTL level – not shown here.



Timing Closure
BRAM Access Latency

ROSTA

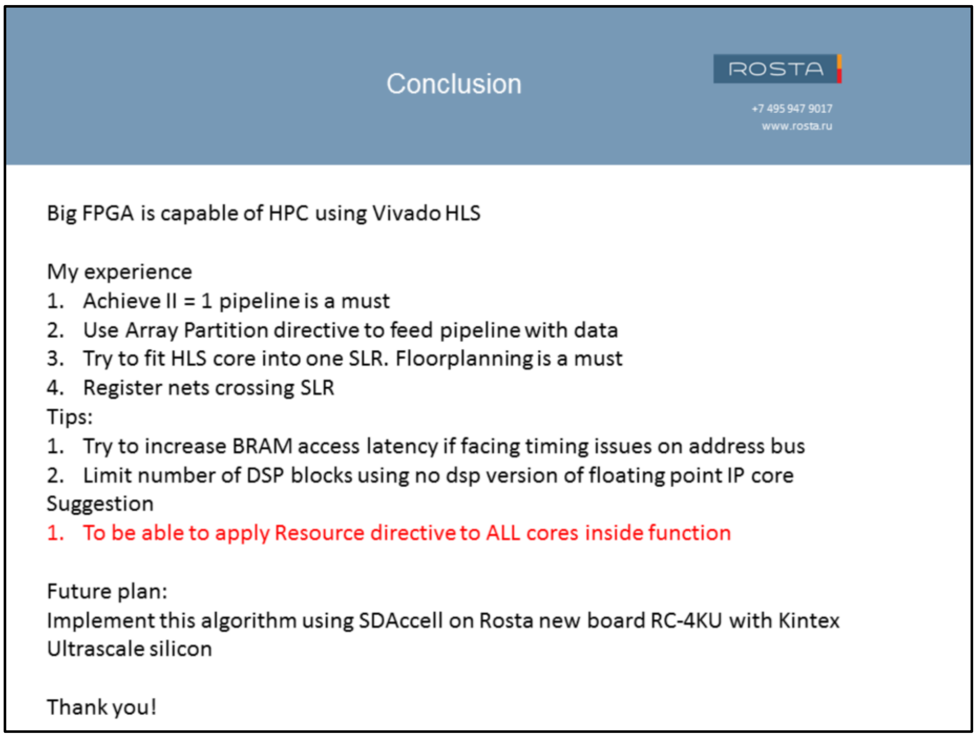
+7 495 947 9017
www.rosta.ru

First implementation results showed lots of very long combinatorial paths in front of BRAM Address for HLS arrays

Good Idea was to insert FF in this path using Vivado HLS directive

```
#pragma HLS RESOURCE variable=m1 core=RAM_2P_BRAM latency=5
```

This is the last problem I want to mention. First implementation results showed lots of very long combinatorial nets driving BRAM address input of HLS memory arrays. HLS Resource directive can be used to increase read latency of these arrays. It is nothing that just this line. Realy nothing more. And suddenly problem disappeared. And this is magic that does Vivado HLS. It automatically adjusts control state machine and reschedules all operations to match new read latency.



Conclusion

ROSTA

+7 495 947 9017
www.rosta.ru

Big FPGA is capable of HPC using Vivado HLS

My experience
1. Achieve II = 1 pipeline is a must
2. Use Array Partition directive to feed pipeline with data
3. Try to fit HLS core into one SLR. Floorplanning is a must
4. Register nets crossing SLR

Tips:
1. Try to increase BRAM access latency if facing timing issues on address bus
2. Limit number of DSP blocks using no dsp version of floating point IP core

Suggestion
1. To be able to apply Resource directive to ALL cores inside function

Future plan:
Implement this algorithm using SDAccell on Rosta new board RC-4KU with Kintex Ultrascale silicon

Thank you!

I want to finish my presentation. Main conclusion that I can make it that Vivado HLS is capable of HPC. We took real complex scientific problem and not only made it work on FPGA but achieved 15 times speedup compared to multicore CPU implementation. We even beat GPU, but it is different story.

Another point is big FPGA. On the one hand, it gives us freedom, we have lots of hardware and can implement big algorithms. On the other hand, we need to pay attention on SLR issues.

We are on step of obtaining new scientific results using our accelerated implementation.

Our Future technical plan is to go for SDAccel. Rosta has new board with Kintex Ultrascale silicon. Right now we are working on creating SDAccel BSP for this board. We will try to implement this algorithm both on OpenCL and using regular Vivado HLS way on SDAccel BSP.