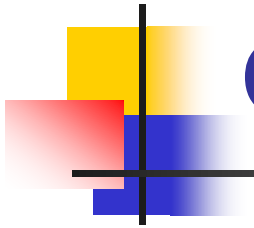
Аппаратная реализация видеофильтров на устройствах программируемой логики

Сергей Грошев

CS MSU Graphics & Media Lab
(Video Group)



Outline

- **Устройство FPGA**
- Устройство Development Boards
- Разработка

Устройство FPGA

ПЛИС – программируемая логическая интегральная схема

FPGA(Field Programmable Gate Array) разновидность ПЛИС

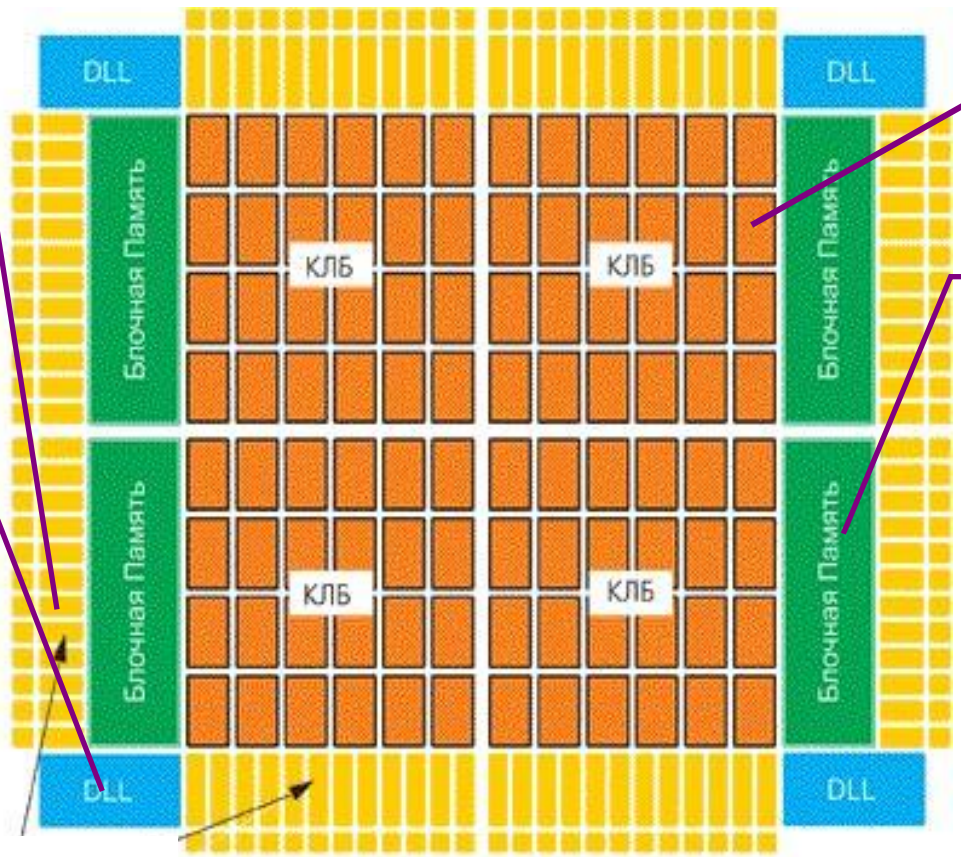
В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования (проектирования). Для программирования используются языки Verilog, VHDL.

Устройство FPGA

Архитектура кристалла FPGA состоит из пяти основных элементов:

- блоков ввода/вывода (БВВ)
- конфигурируемых логических блоков (КЛБ)
- блочной памяти
- модулей управления синхронизацией (DLL)
- трассировочных ресурсов

Структурная схема FPGA



Блоки ввода/вывода

DLL

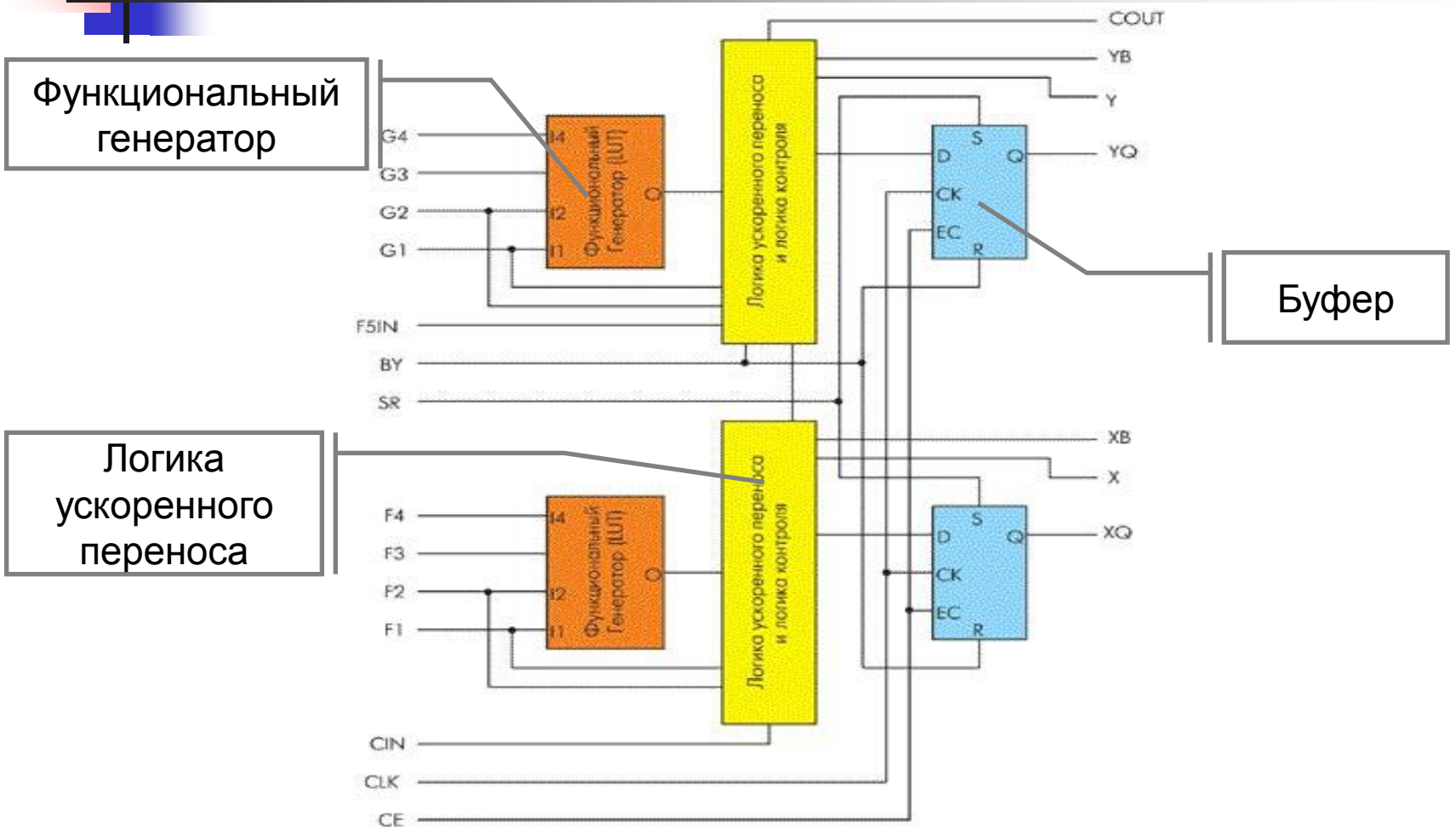
КЛБ

Блочная память

Конфигурируемый логический блок

- КЛБ реализуют логические и регистровые функции
- КЛБ состоит из двух секций, каждая из которых состоит из:
  - 4-входовых функциональных генераторов
  - логики ускоренного переноса
  - двух регистровых элементов
  - двух буферов с тремя состояниями

Структурная схема КЛБ



Функциональный генератор

Функциональные генераторы реализованы в виде 4-входовых таблиц преобразования (Look-Up Table - LUT)

Использование LUT-элементов:

- собственно функциональный генератор
- синхронная память типа RAM размерностью 16x1 бит
- из двух LUT-элементов реализовать
  - синхронную RAM-память размерностью 16x2 или 32x1 бит
  - двухпортовая синхронная RAM-память размерностью 16x1 бит

Блоки ввода/вывода

- Блоки ввода/вывода осуществляют соединение внутренней логики кристалла с выводами корпуса микросхемы
- БВВ поддерживают 19 сигнальных стандартов ввода/вывода, включая дифференциальные стандарты LVDS, BLVDS и LVPECL. Достоинства:
  - Дифференциальные стандарты LVDS и BLVDS позволяют коммутировать ПЛИС с другими микросхемами на плате без использования дополнительных конверторов или трансляторов.
  - Дифференциальные стандарты более скоростные, чем однопроводные,
  - Снижение энергопотребление кристалла и шумов
  - Устраняются электромагнитные наводки.

Поддерживаемые ПЛИС стандарты ввода/вывода

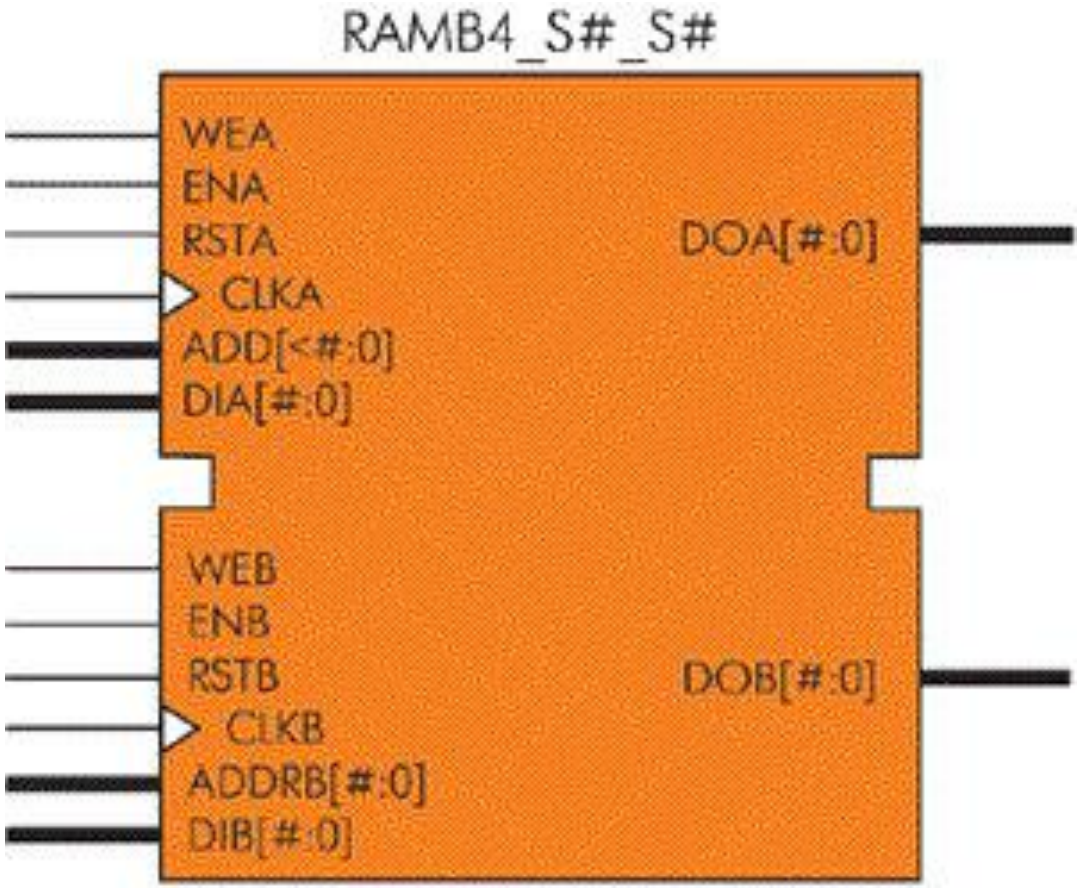
- Интерфейсы межкристального обмена
  - LVTTL
  - LVCMOS2
  - LVCMOS18
  - LVPECL
- Интерфейсы между платами
  - PCI 33/66 MHz, 3,3 V
  - HSTL-I
  - GTL
  - GTL+
  - HSTL-III & IV
  - SSTL3-I & II
  - LVDS
  - AGP-2X
  - SSTL2-I & II
  - BLVDS
  - CTT

Блочная память (Block RAM)

- Каждый блок памяти (4096 бит) - это синхронная двухпортовая RAM с независимым управлением для каждого порта
- Независимая конфигурация позволяет создавать преобразователи размерности шины

Разрядность	Глубина	Шина адреса	Шина данных
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	216	ADDR<7:0>	DATA<15:0>

Блочная память (схема)



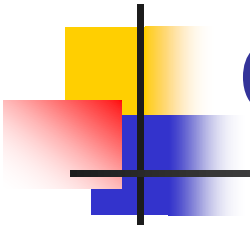
Модули автоподстройки задержки (DLL)

Полностью цифровая автоподстройка задержки (DLL) может устранять перекос задержек между синхросигналом на входном контакте микросхемы и сигналами на тактовых входах внутренних схем устройства путем введения дополнительной задержки (ДЗ).

Модули автоподстройки задержки (DLL)

- ДЗ вводится таким образом, чтобы фронты сигналов синхронизации достигали внутренних триггеров в точности на один период синхронизации позже их прихода на входной контакт

- модуль DLL может создавать четыре квадратурные фазы из исходного источника синхросигнала; удваивать частоту синхросигнала или делить эту частоту на 1.5, 2, 2.5, 3, 4, 5, 8 или 16.



Outline

- Устройство ПЛИС
- **Устройство Development Boards**
- Разработка

Устройство Development Boards

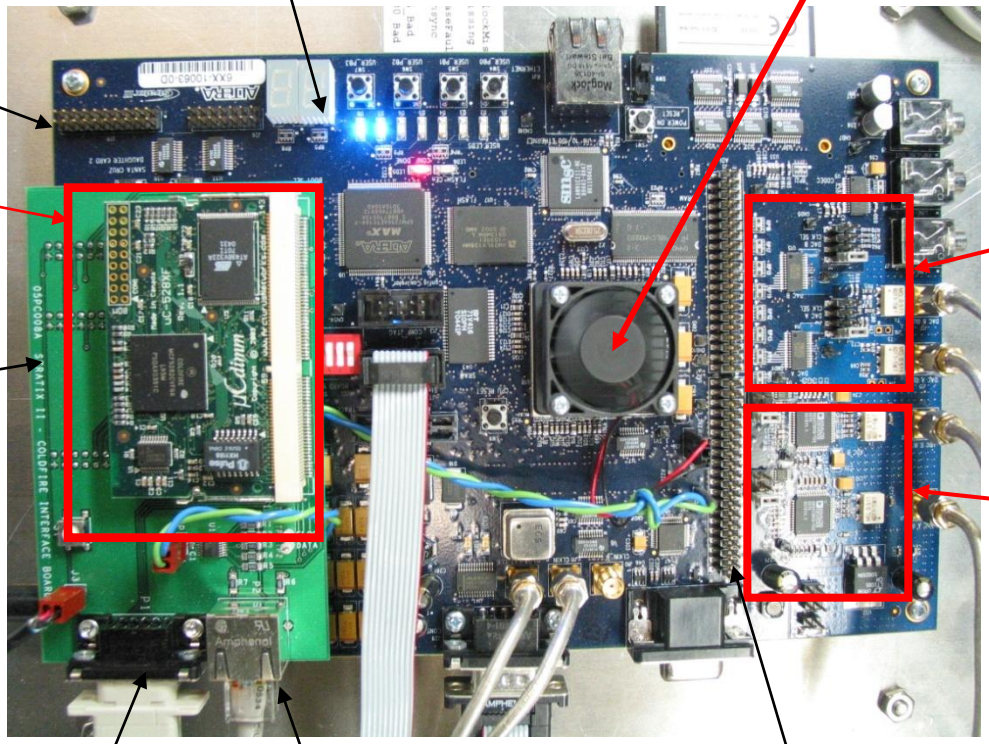
Development Board – специальная плата (внешняя или внутренняя на PCI/PCI-Ex) для разработки программ под ПЛИС.

Состав:

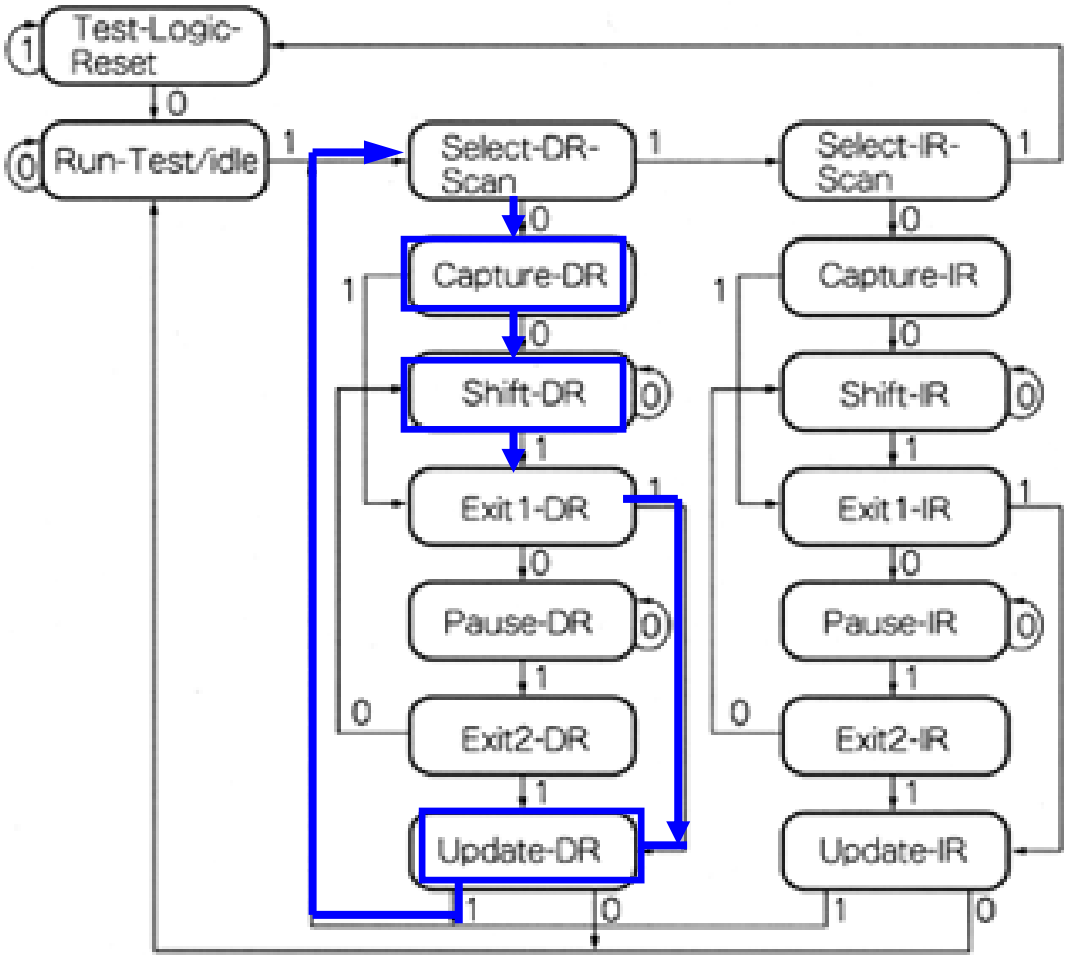
- Одна или несколько ПЛИС
- Микросхема процессора (опционально)
- Различные порты для подключения и их контроллеры

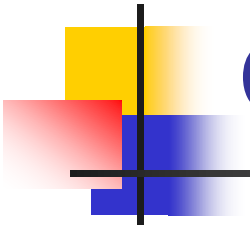
Устройство ПЛИС

Общий вид Development Board



Конфигурация через порт JTAG





Outline

- Устройство ПЛИС
- Устройство Development Boards
- **Разработка программ**

Разработка программ

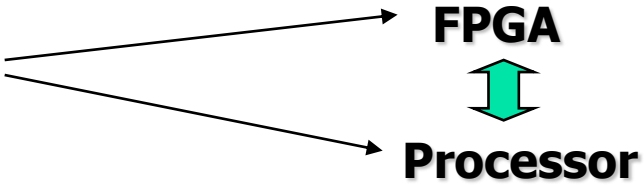
Принцип работы FPGA

≠

принцип работы процессора

Возникают проблемы:

- вопросы архитектуры (до начала разработки)
- выбор языка
- разделение вычислений



Начало разработки

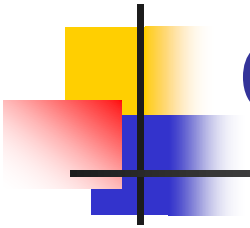
- Перед началом разработки под ПЛИС следует выяснить детали:
  - Логическое устройство
  - Ввод/вывод

Логические элементы

- Количество входов
- Количество рабочих функций
  - У всех ли функций n входов
  - Как входы микросхемы связаны с входами функций
- Специальная логика
  - Сумматор и т.д.
- Регистры

Блоки ввода вывода

- Сколько ножек микросхемы
- Индивидуальное или групповое программирование входов
- Все ли входы могут работать со всеми функциями



Outline

- Устройство ПЛИС
- Устройство Development Boards
- **Разработка программ**

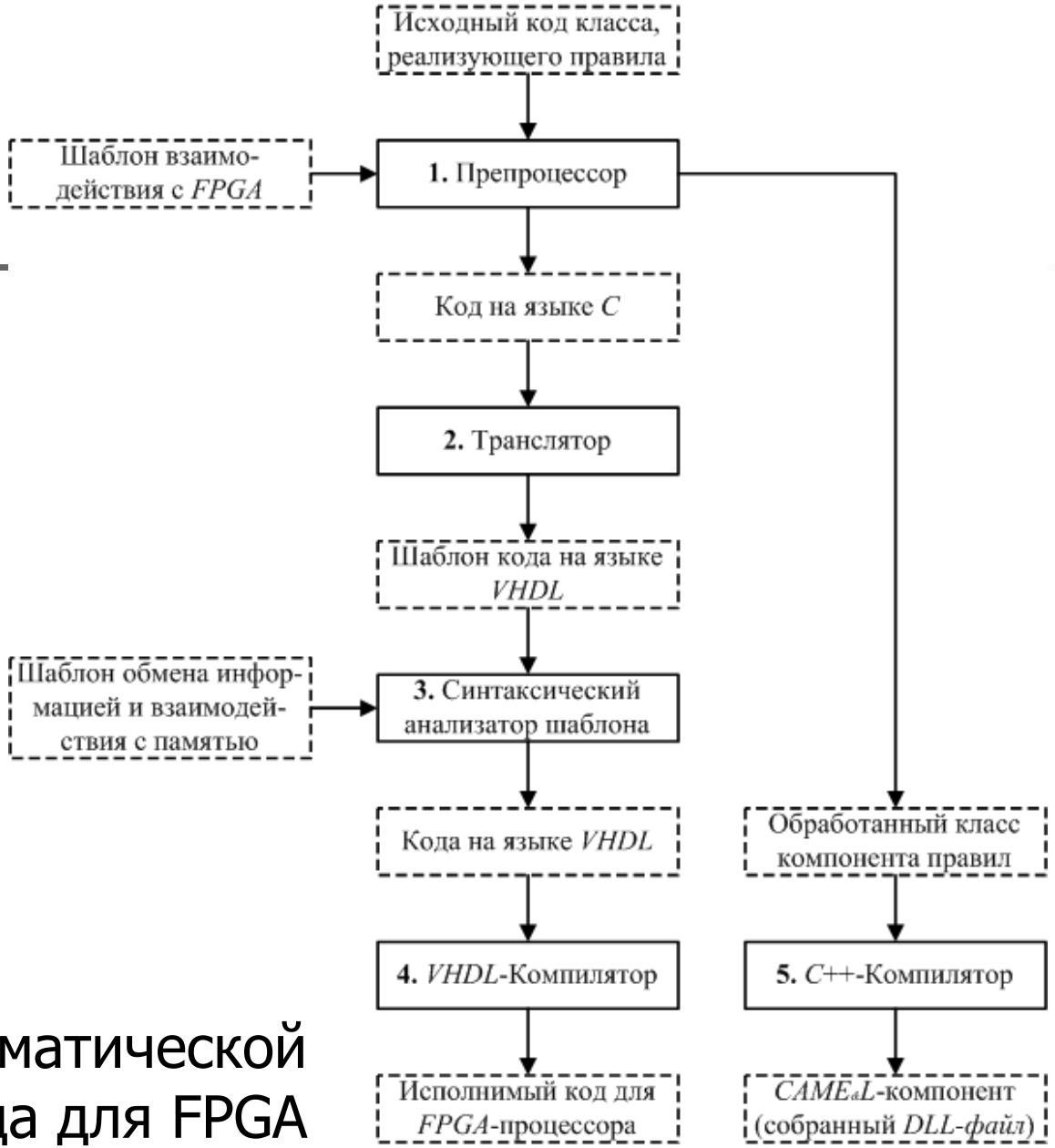
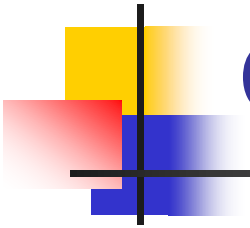


Схема автоматической генерации кода для FPGA

Инструменты разработки Altera

- Mentor Graphics Catapult C Synthesis
- Quartus II
  - Симулирование и компилирование Verilog в gates
- Nios-II Development Kit
  - Компилирование C-кода в NIOS-код
- ModelSim
  - Verilog симулятор



Outline

- Устройство ПЛИС
- Устройство Development Boards
- Разработка программ
  - **Catapult C Synthesis**
  - Обзор пакета QuartusII
  - Обзор NIOS Development Kit
  - Оптимизация работы

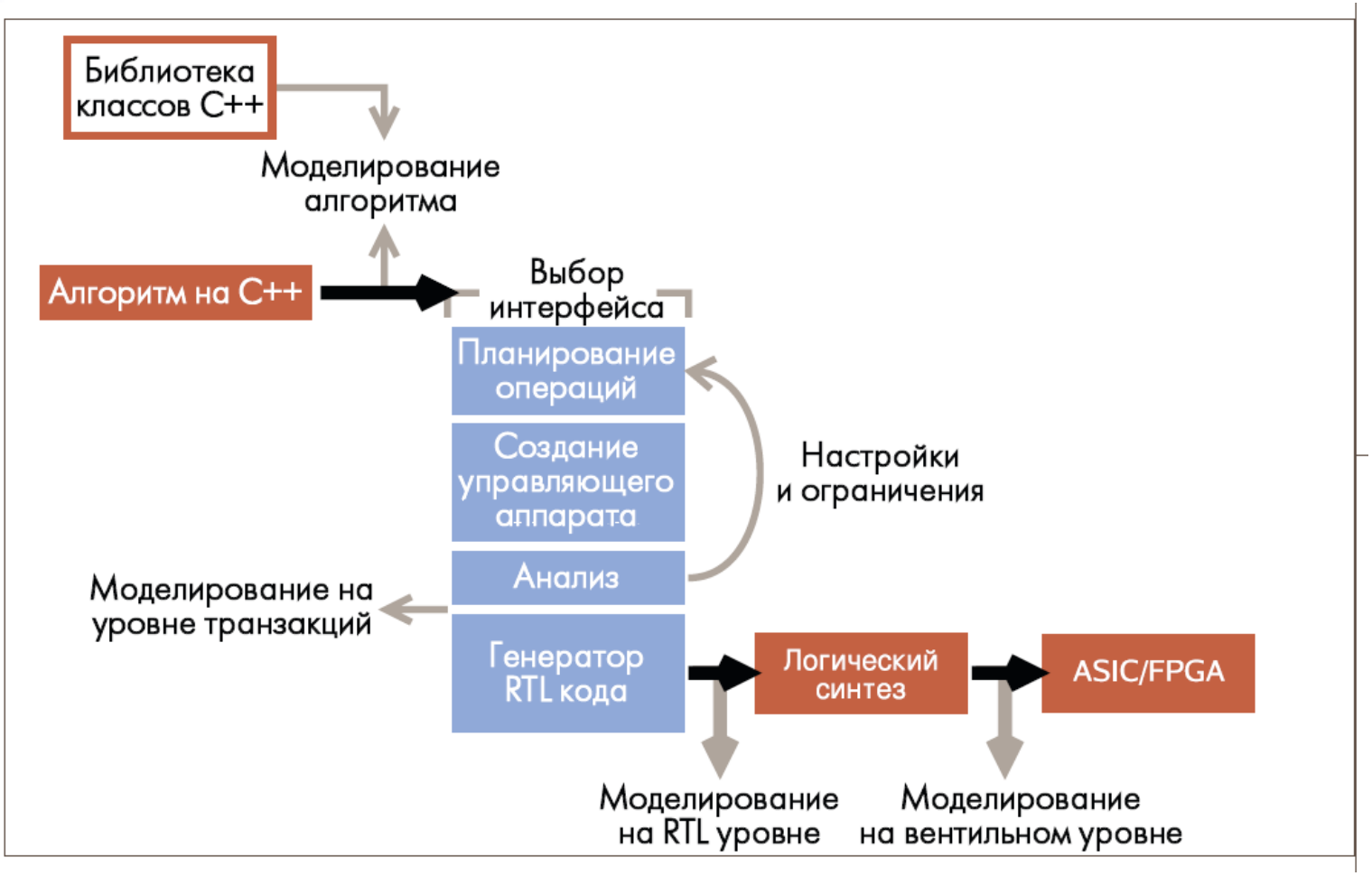
Catapult C Synthesis

- использование системы Catapult C Synthesis позволяет автоматически синтезировать RTLописание из исходного описания алгоритмов на C/C++.

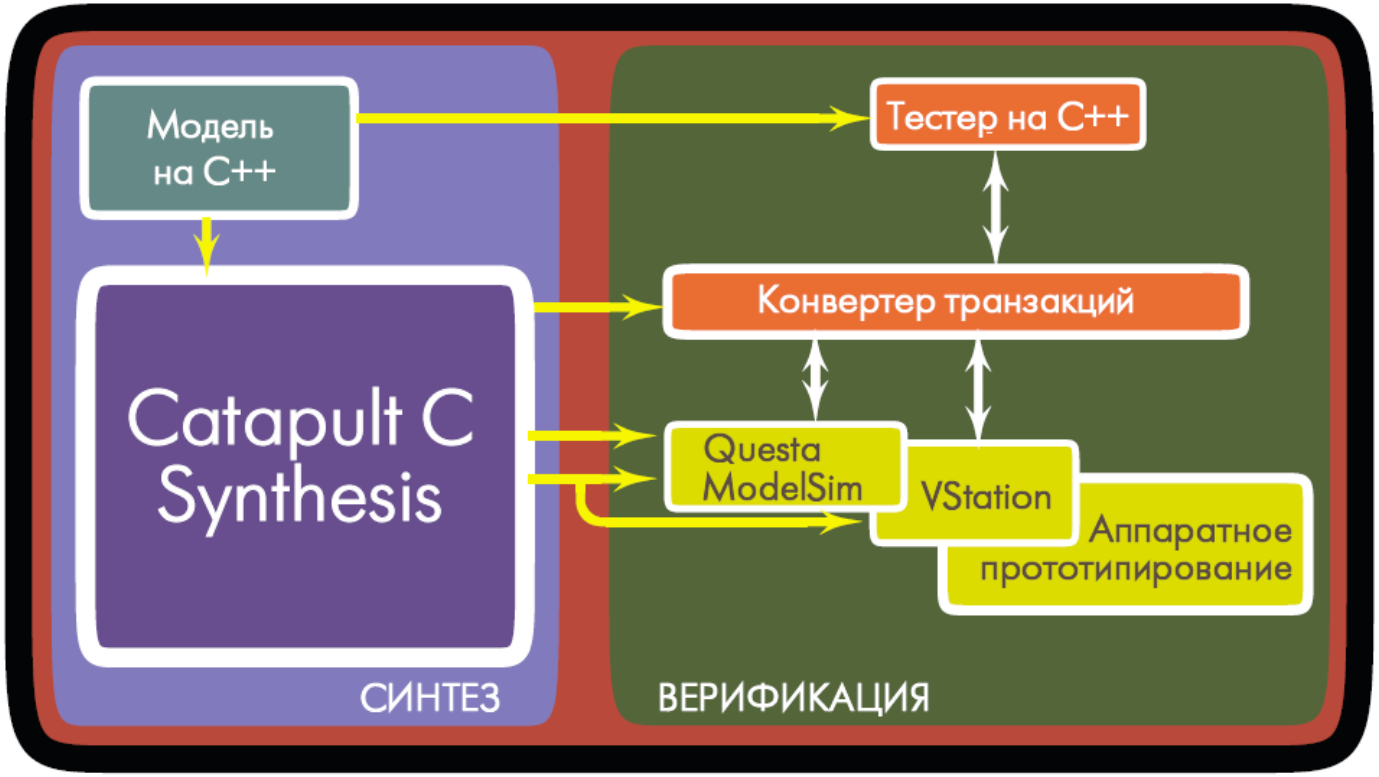
Catapult C Synthesis

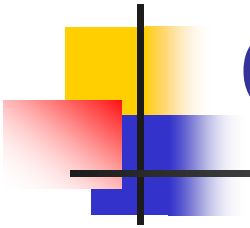
- основная нагрузка по отладке и верификации проекта переносится на более высокий уровень абстракции
- анализ альтернативных вариантов реализации алгоритмов на C/C++
- описание проекта на C/C++ не привязано к выбору микроархитектуры и конкретной технологии реализации.
- В зависимости от потребностей текущей разработки одно и то же описание может быть использовано как для реализации проекта на базе FPGA различных типов, так и в виде ASIC.

Catapult C Synthesis



Catapult C Synthesis





Outline

- Устройство ПЛИС
- Устройство Development Boards
- Разработка программ
  - Catapult C Synthesis
  - **Обзор пакета QuartusII**
  - Обзор NIOS Development Kit
  - Оптимизация работы

Altera Quartus II

Последовательность разработки

1. Create a “new project”
2. Add Verilog code to the project
3. Edit your Verilog code
4. Compile it
5. Simulate it
6. Debug it
7. Go to step 3

Quartus: новый проект

File → New Project Wizard

- Next
- Выбрать каталог
  - Назвать project (eg, my_xor)
  - Назвать top-level design file (eg, my_xor)
- Добавить файлы: eg, my_xor.v
- Нажать Next несколько раз
  - Выбор ‘Cyclone’ device family
  - Выбор ‘7’ as the speed grade, then select ‘1C20F400C7’
- Finish

Verilog Example: XOR Gate

```
module my_xor( C, A, B );
    output C;
    input A, B;

    assign C = (A ^ B);
endmodule
```

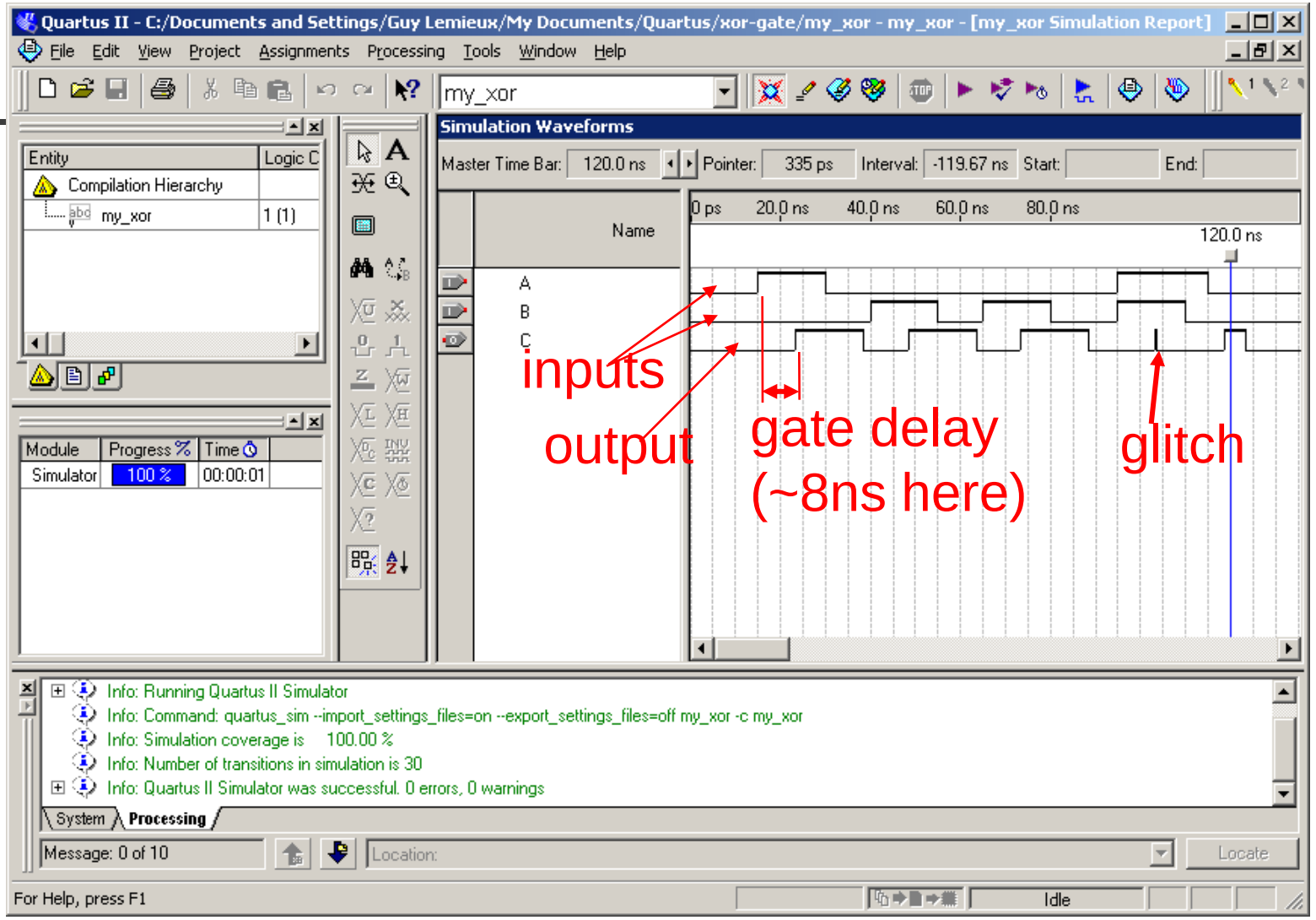
Operation	Operator
~	Bitwise NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

Convention: Outputs come first in the “parameter list”

Quartus: Компилирование и симулирование

- Processing → Start Compilation
  - (wait a while)
- Open my_xor.vwf
  - Right-click in the ‘Name’ column (white area)
  - Insert Node or Bus…
    - Node Finder…
      - Press List button
      - Under ‘Nodes Found’, choose inputs ‘A’ and ‘B’
      - Press ‘>’ button to move them to ‘Selected nodes’
      - Press OK
    - Press OK
  - You now have two input waveforms
    - Try to change their value (select region, right-click or press buttons)
  - Save
- Processing → Start Simulation
  - (wait a bit)

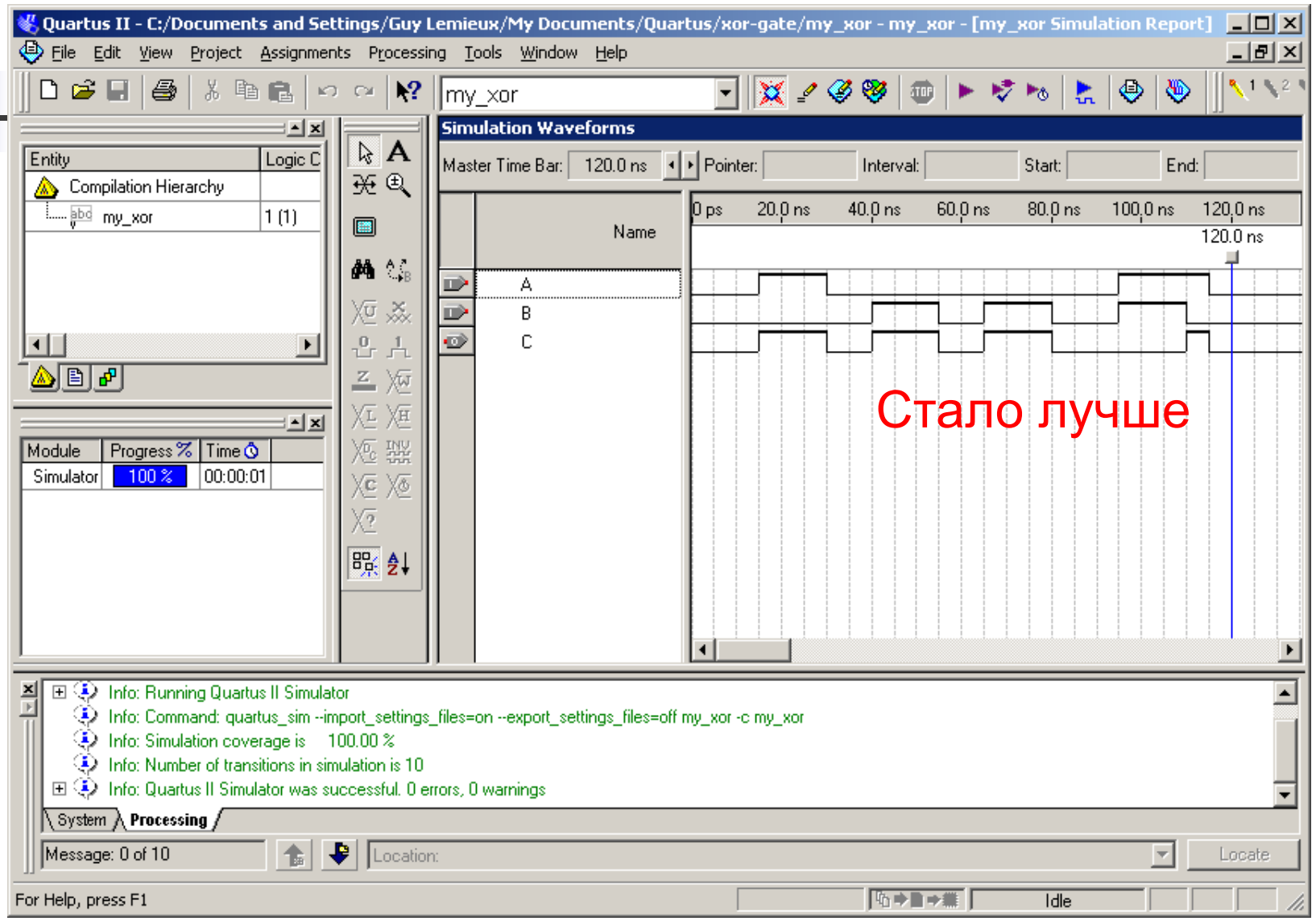
Quartus:Timing Waveforms

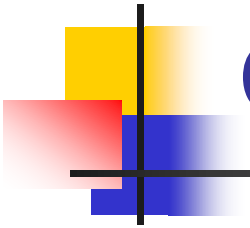


Quartus: Быстрое компилирование и симулирование – Using Functional Mode

- Processing → Generate Functional Simulation Netlist
- Assignments → Settings…
  - Select Simulator
  - Simulation mode: Functional

Quartus


Quartus II - C:/Documents and Settings/Guy Lemieux/My Documents/Quartus/xor-gate/my_xor - my_xor - [my_xor Simulation Report]
File Edit View Project Assignments Processing Tools Window Help
my_xor
Entity
Logic C
Compilation Hierarchy
my_xor
1 (1)
Module
Progress %
Time
Simulator
100 %
00:00:01
Simulation Waveforms
Master Time Bar: 120.0 ns
Pointer:
Interval:
Start:
End:
Name
A
B
C
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
120.0 ns
Стало лучше
Info: Running Quartus II Simulator
Info: Command: quartus_sim --import_settings_files=on --export_settings_files=off my_xor -c my_xor
Info: Simulation coverage is 100.00 %
Info: Number of transitions in simulation is 10
Info: Quartus II Simulator was successful. 0 errors, 0 warnings
System
Processing
Message: 0 of 10
Location:
Locate
For Help, press F1
Idle



Outline

- Устройство ПЛИС
- Устройство Development Boards
- Разработка программ
  - Обзор пакета QuartusII
  - **Обзор NIOS Development Kit**
  - Оптимизация работы
  - Использование MAX+PLUSII

Altera Nios II
NIOS Development Kit (NDK)

- В составе четыре ключевых компонента:
  - NIOS II IDE
    - Integrated Development Environment for Windows, GUI-based programming environment
    - Based on Eclipse http://www.eclipse.org
  - Cygwin http://www.cygwin.com
    - “Linux” layer on top of Windows
  - GNU C Compiler Tools ‘gcc’ http://gcc.gnu.org
    - A popular, free C compiler that targets many different CPUs
    - Integrated into the IDE
  - Documentation
    - Click ‘full documentation’ under item 4.
    - Click ‘The Nios II Processor Reference Handbook’

NIOS IDE: New Project

Этапы создания нового проекта

1. Выбор типа проекта
   - Altera NIOS II, C/C++ Application, Next
2. Выбрать шаблон: Dhrystone
3. Выбрать Железо: Browse…
   - (Cyclone, Aria GX, Spartain etc.)
4. Next/Finish

NIOS Cygwin

- Функциональность:
  - Возможность использования Unix функций в коде
  - Unix утилиты: ls, less, grep, sed, awk, etc…
- Применение
  - Использование gcc для компиляции C в NIOS код
  - Возможность делать assemble/disassemble NIOS кода

GCC and Objdump

- Компилирование и дизассемблинг:

% nios2-elf-gcc –O –c dhrystone.c
compiles program, optimizes code, writes object file dhrystone.o

% nios2-elf-objdump –S dhrystone.o
disassembles object file (or executable file) to screen (stdout)
shows instructions, labels, memory addresses, and binary machine code

- Компилирование для сборки и сборка:

% nios2-elf-gcc –O –S dhrystone.c
compiles program, optimizes code, writes assembly file dhrystone.s
shows labels and instructions only (no addresses or binary code)

% nios2-elf-gcc –c dhrystone.s
assembles program, writes object file dhrystone.o

Пример NIOS Assembly

C Code

```
int main()
{
    int a = 1;
    int b = 1;
    int c;
    int i;

    for( i=2; i <= 100; i++ ) {
        c = a + b;
        a = b;
        b = c;
    }

    return c;
}


Run these commands:
% nios2-elf-gcc -O2 -S fib.c
% cat fib.s

Output is shown in next column.

% nios2-elf-gcc -c fib.s
```

Assembly Code

```
 .file   "fib.c"
       .section        .text
       .align  3
       .global main
       .type   main, @function
main:
       addi    sp, sp, -8
       movi    r4, 1
       stw     fp, 4(sp)
       mov     r5, r4
       mov     fp, sp
       movi    r3, 98
.L6:
       add     r2, r5, r4
       addi    r3, r3, -1
       mov     r5, r4
       mov     r4, r2
       bge     r3, zero, .L6

       ldw     fp, 4(sp)
       addi    sp, sp, 8
       ret
       .size   main, .-main
       .ident  "GCC: (GNU) 3.3.3 (Altera Nios II 1.0 b316)"
```

Пример NIOS Disassembly

C Code

```
int main()
{
    int a = 1;
    int b = 1;
    int c;
    int i;

    for( i=2; i <= 100; i++ ) {
        c = a + b;
        a = b;
        b = c;
    }

    return c;
}


Run these commands:
% nios2-elf-gcc -O2 -c fib.c
% nios2-elf-objdump -S fib.o

Output is shown in next column.
```

Disassembled Code

```
fib.o:     file format elf32-littlenios2

Disassembly of section .text:

00000000 <main>:
   0:   defffe04        addi    sp,sp,-8
   4:   01000044        movi    r4,1
   8:   df000115        stw     fp,4(sp)
   c:   200b883a        mov     r5,r4
  10:   d839883a        mov     fp,sp

  14:   00c01884        movi    r3,98

  18:   2905883a        add     r2,r5,r4
  1c:   18ffffc4        addi    r3,r3,-1
  20:   200b883a        mov     r5,r4
  24:   1009883a        mov     r4,r2
  28:   183ffb0e        bge     r3,zero,18 <main+0x18>

  2c:   df000117        ldw     fp,4(sp)
  30:   dec00204        addi    sp,sp,8
  34:   f800283a        ret
```

Процесс написания NIOS программы

Последовательность действий

1. Написание на C

```
% vi foo.c
```

2. Компилирование в assembly code

```
% nios2-elf-gcc -O -S foo.c
```

3. Редактирование assembly code (напр. удаление неподдерживаемых инструкций)

```
% vi foo.s
```

4. Сборка (ассемблирование)

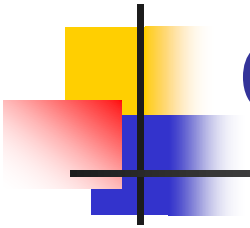
```
% nios2-elf-gcc -c foo.s
```

5. Дизассемлирование

```
% nios2-elf-objdump -S foo.o > foo.txt
```

6. Извлечение бинарного машинного кода

```
% vi foo.txt
```

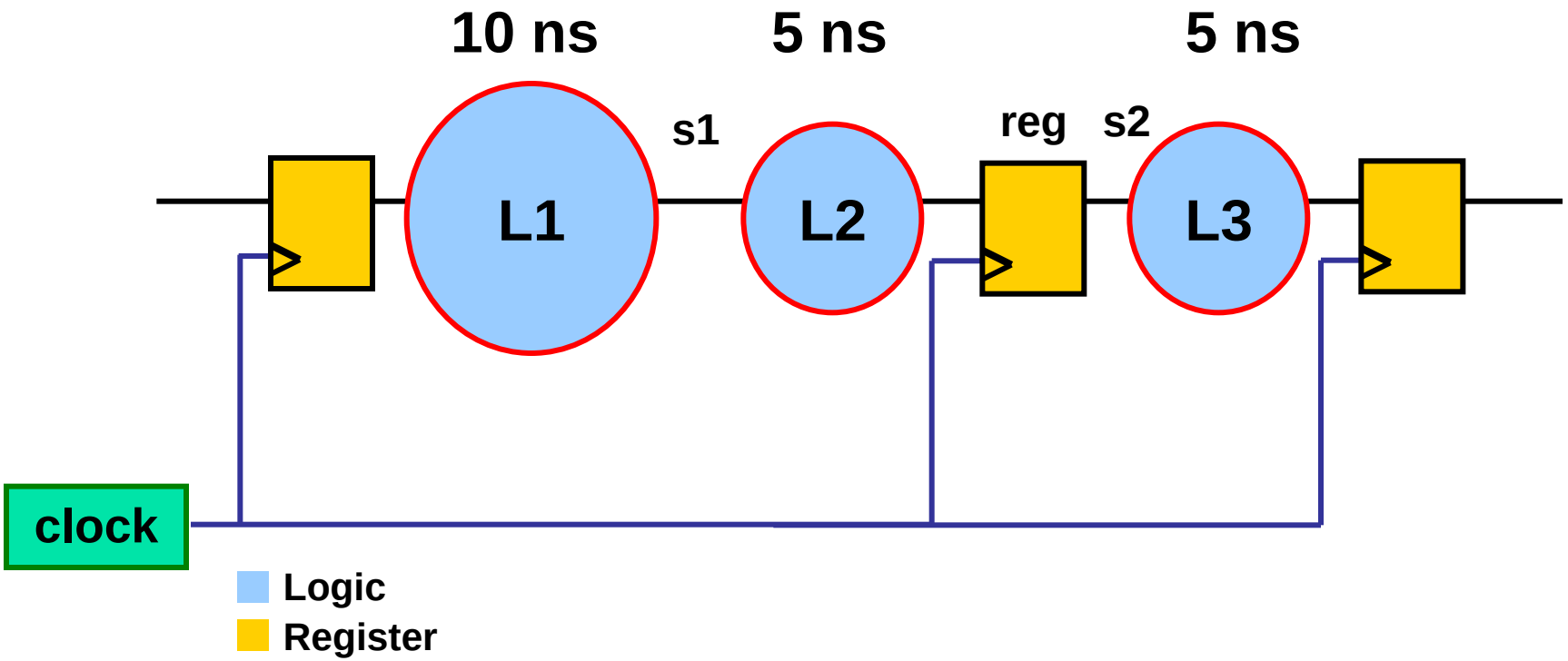


Outline

- Устройство ПЛИС
- Устройство Development Boards
- Разработка программ
  - Обзор пакета QuartusII
  - Обзор NIOS Development Kit
  - **Оптимизация работы**

Последовательность работы программы

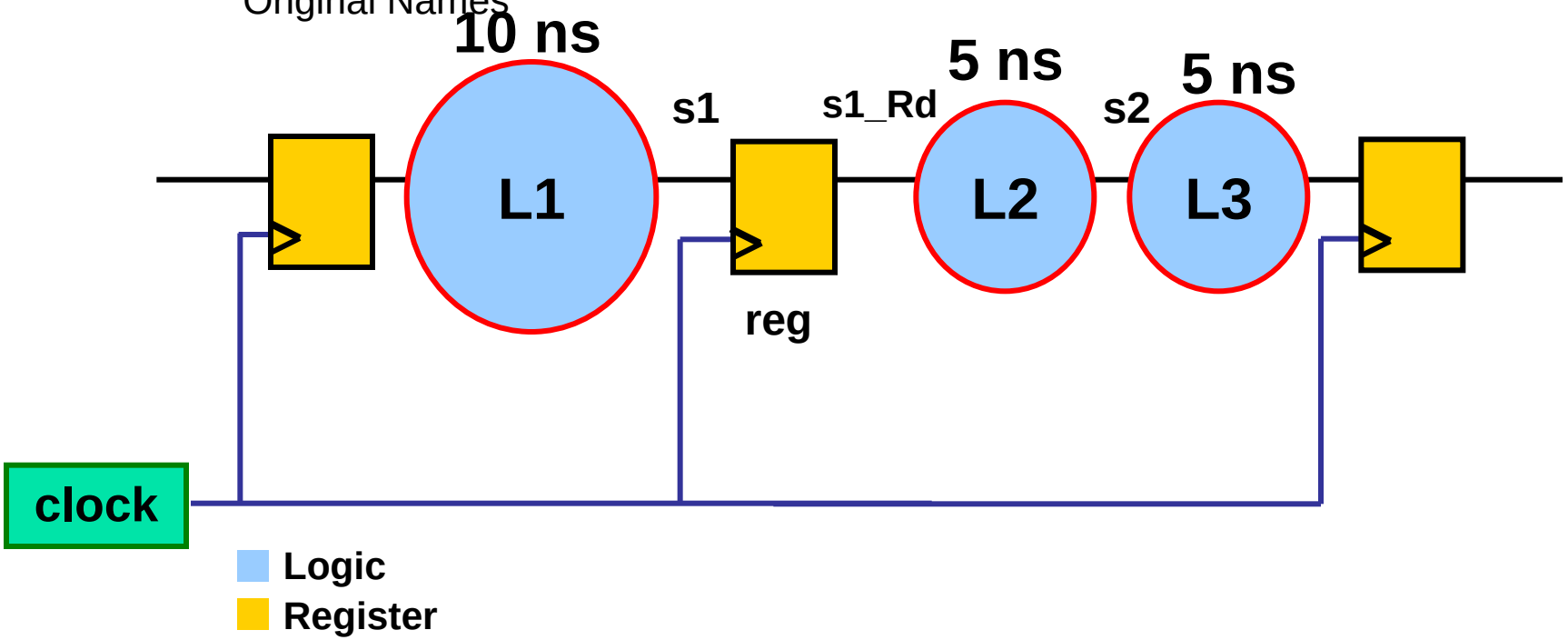
- Longest Path = 15 ns
- fMAX = 73 MHz



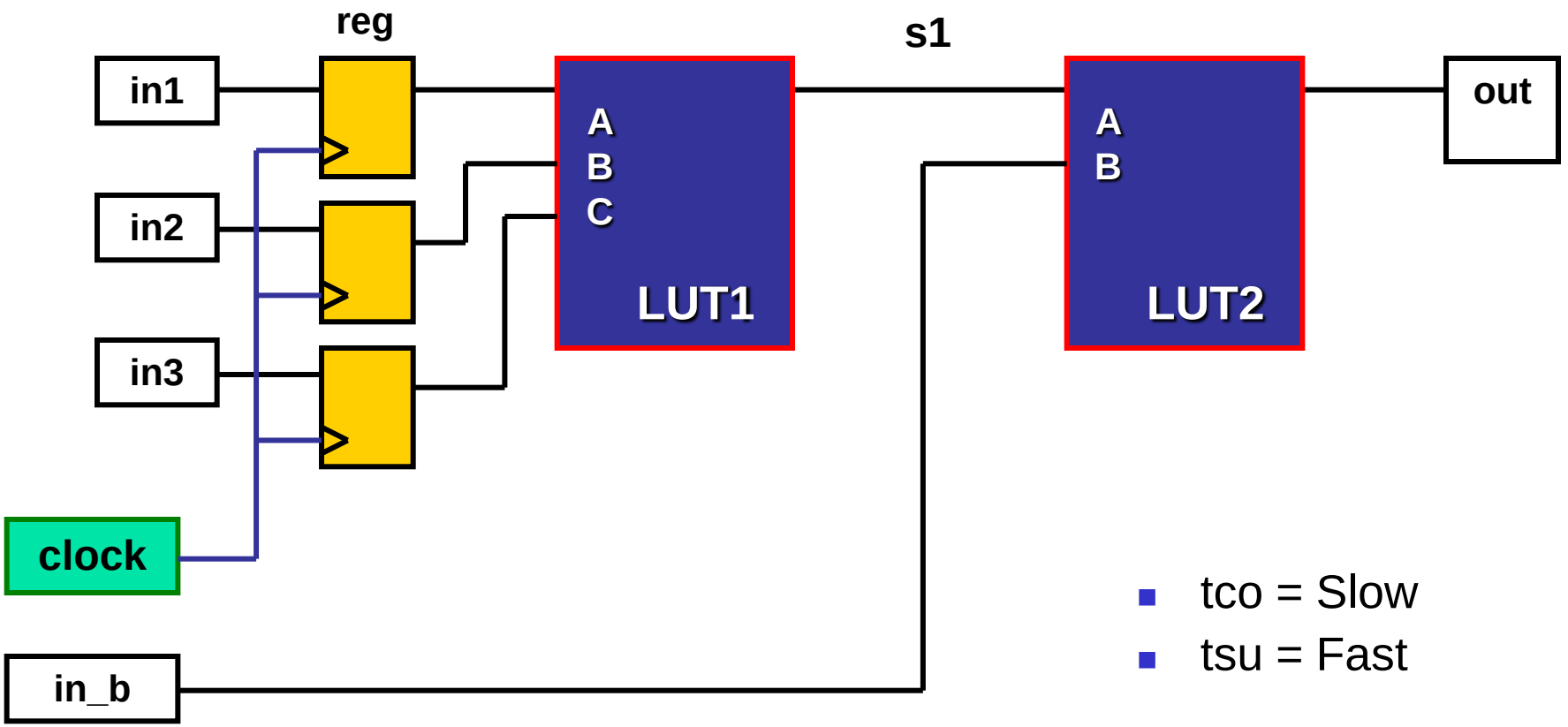
Последовательность работы программы

- Longest Path = 10 ns
- fMAX = 100 MHz
- Logic in L2 Pushed Forward Across Register
- All Signals in L2 Are Delayed by One Cycle & "_Rd" is Appended to Their Original Names

Оптимальнее

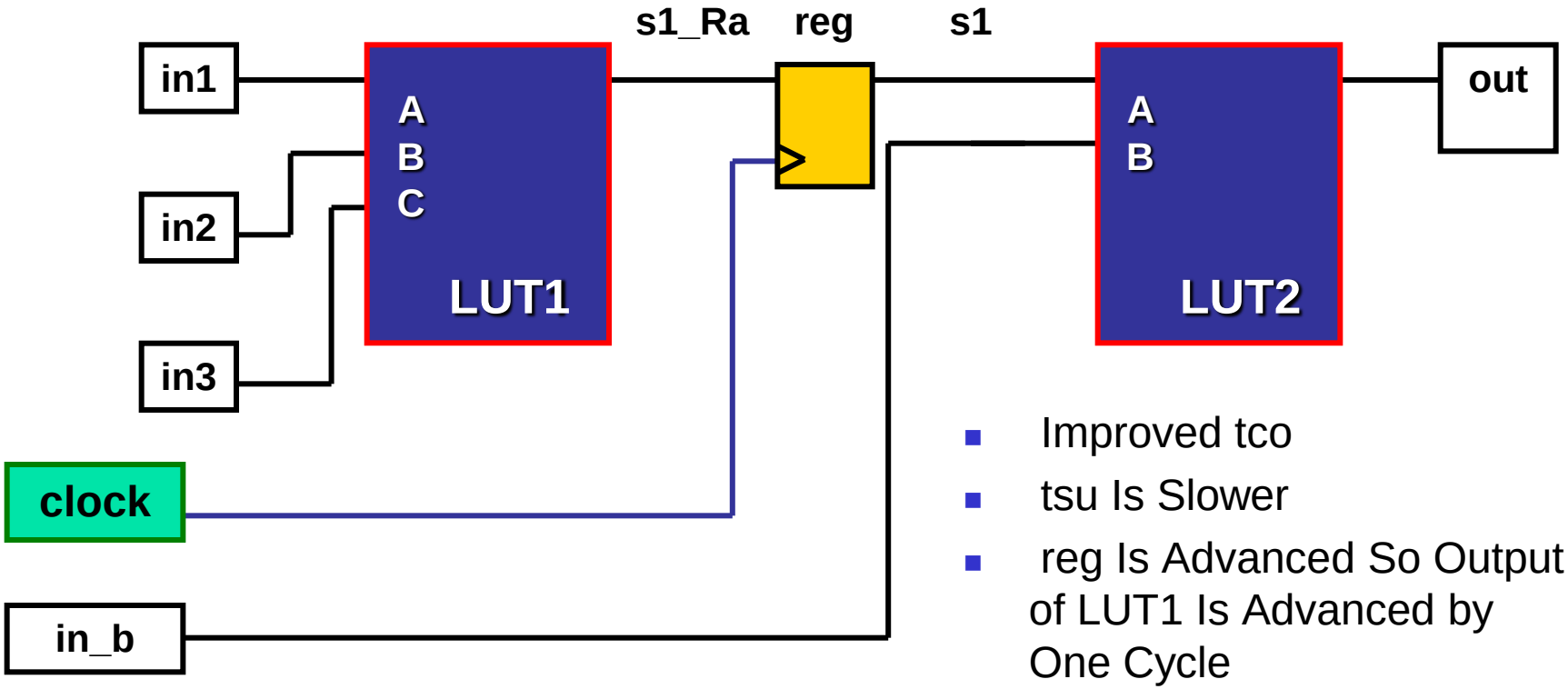


Экономия регистров

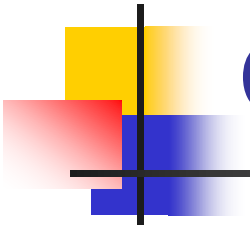


- tco = Slow
- tsu = Fast

Экономия регистров



- Improved tco
- tsu Is Slower
- reg Is Advanced So Output of LUT1 Is Advanced by One Cycle
- Eliminated 2 Registers



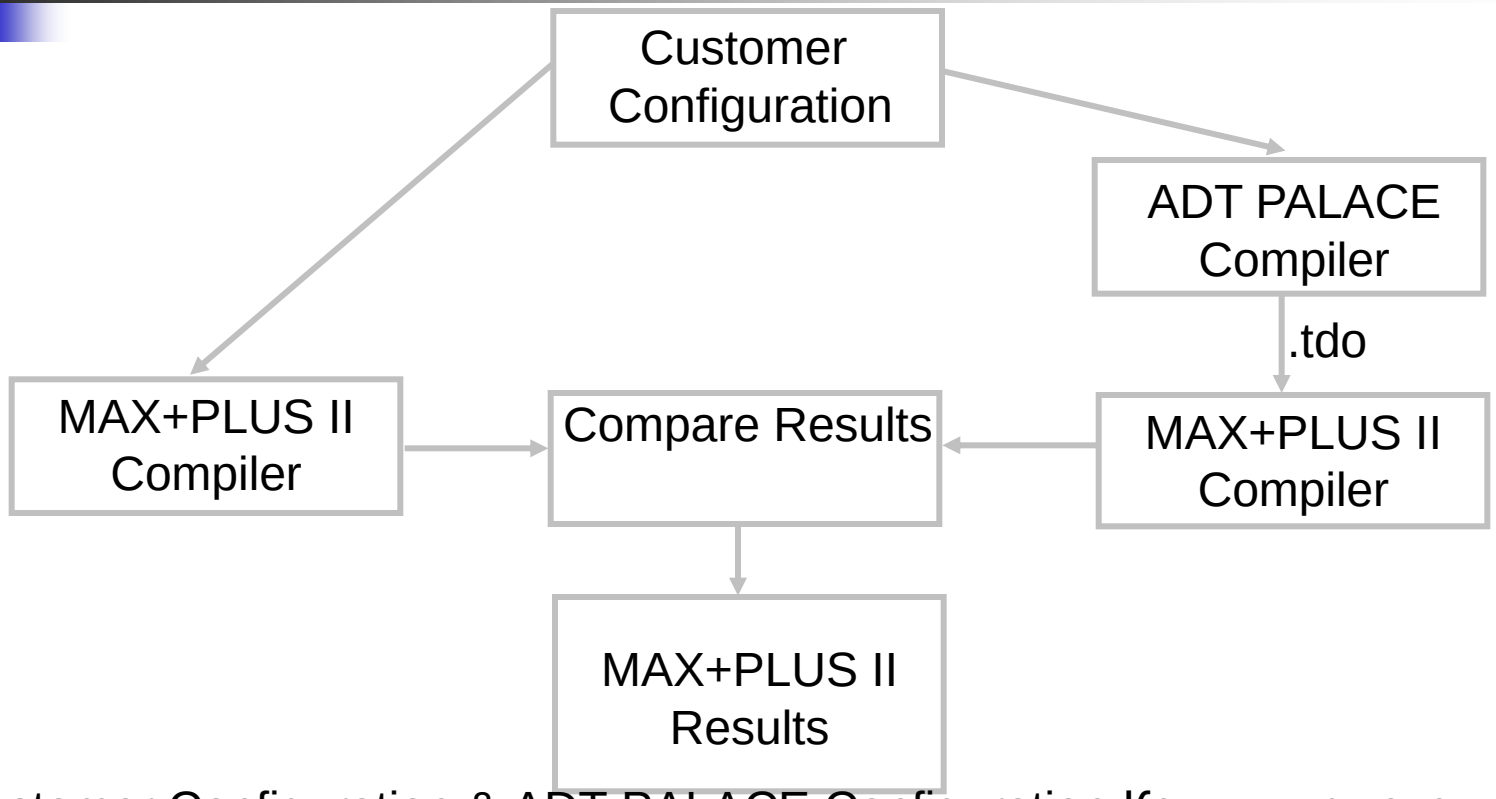
Outline

- Устройство ПЛИС
- Устройство Development Boards
- Разработка программ
  - Обзор пакета QuartusII
  - Обзор NIOS Development Kit
  - Оптимизация работы
  - **Использование MAX+PLUSII**

MAX+PLUS II Работа

GRAPHICS & MEDIA LAB
VIDEO GROUP

(ADT PALACE Compiler Enabled)



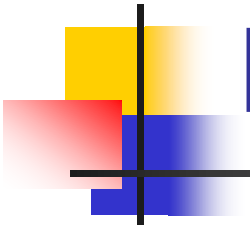
- Customer Configuration & ADT PALACE Configuration Компилируются автоматически
- MAX+PLUS II выдает результаты через **GREATER** fMAX on Slowest Clock

MAX+PLUS II Vs. Quartus II for MAX

- MAX+PLUS II v10.2 Provides Best Average f_{MAX} Performance at the Expense of Longer Compile Times
- Quartus II v2.1 Uses Fewest Macrocells With Default Settings

Рекомендации

- Quartus II использовать для новых 2.5V and 3.3V FPGA
- MAX+PLUS II использовать для несложных проектов и 5-V MAX FPGA



Вопросы

?