БУДУЩЕЕ
В НАШИХ
РУКАХ

Итеративная сборка проектов ПЛИС

Константин Павлов

Константин Павлов

Старший инженер по разработке СнК
«ЯДРО Микропроцессоры»

Зачем нужна итеративная сборка?

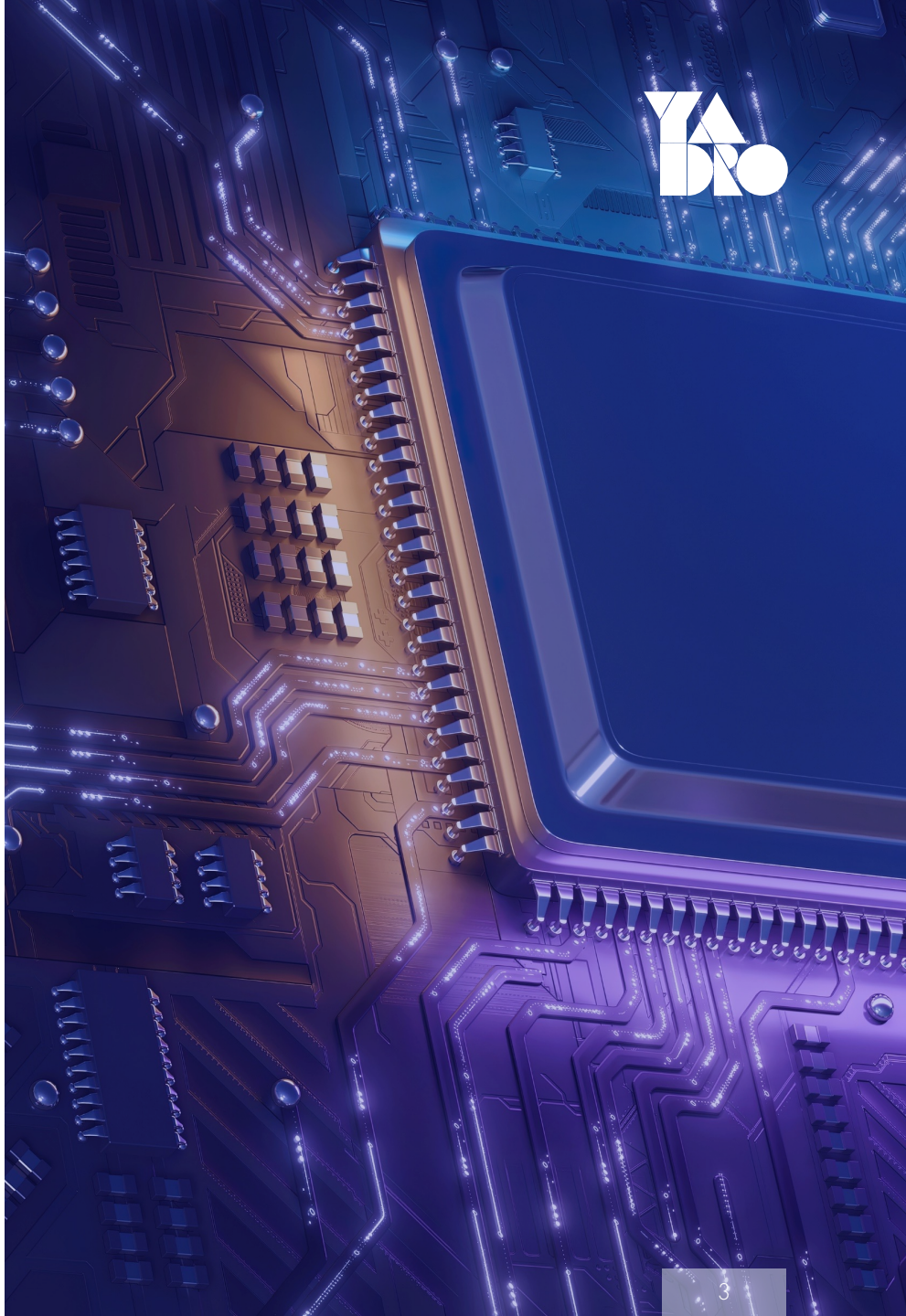
Основная проблема

HDL код – это ещё не всё!

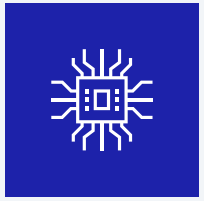
Даже абсолютно корректный RTL код
не всегда успешно заработает в целевом проекте!

Есть множество ограничений, зависящих

- от архитектуры семейства ПЛИС
- среды разработки
- от заполненности ПЛИС
- от особенностей тактирования
- от требований к latency обработки данных
- и так далее...



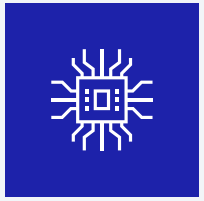
Задачи итеративной сборки



Проверить синтезируемость кода
во всем диапазоне параметров

качество исходного кода

Задачи итеративной сборки



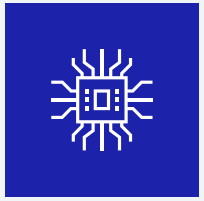
Проверить синтезируемость кода
во всем диапазоне параметров

качество исходного кода

Убедиться в отсутствии временных
нарушений

setup/hold slack, Fmax

Задачи итеративной сборки

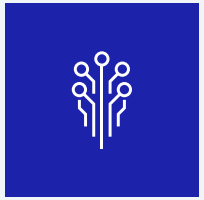


Проверить синтезируемость кода
во всем диапазоне параметров

качество исходного кода

Убедиться в отсутствии временных
нарушений

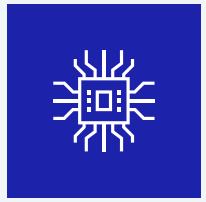
setup/hold slack, Fmax



Выбрать наилучшую
стратегию синтеза
и имплементации

performance/area/power

Задачи итеративной сборки

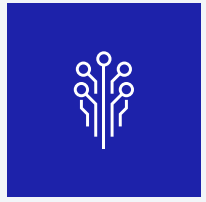


Проверить синтезируемость кода
во всем диапазоне параметров

качество исходного кода

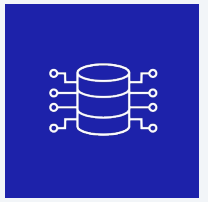
Убедиться в отсутствии временных
нарушений

setup/hold slack, Fmax



Выбрать наилучшую
стратегию синтеза
и имплементации

performance/area/power



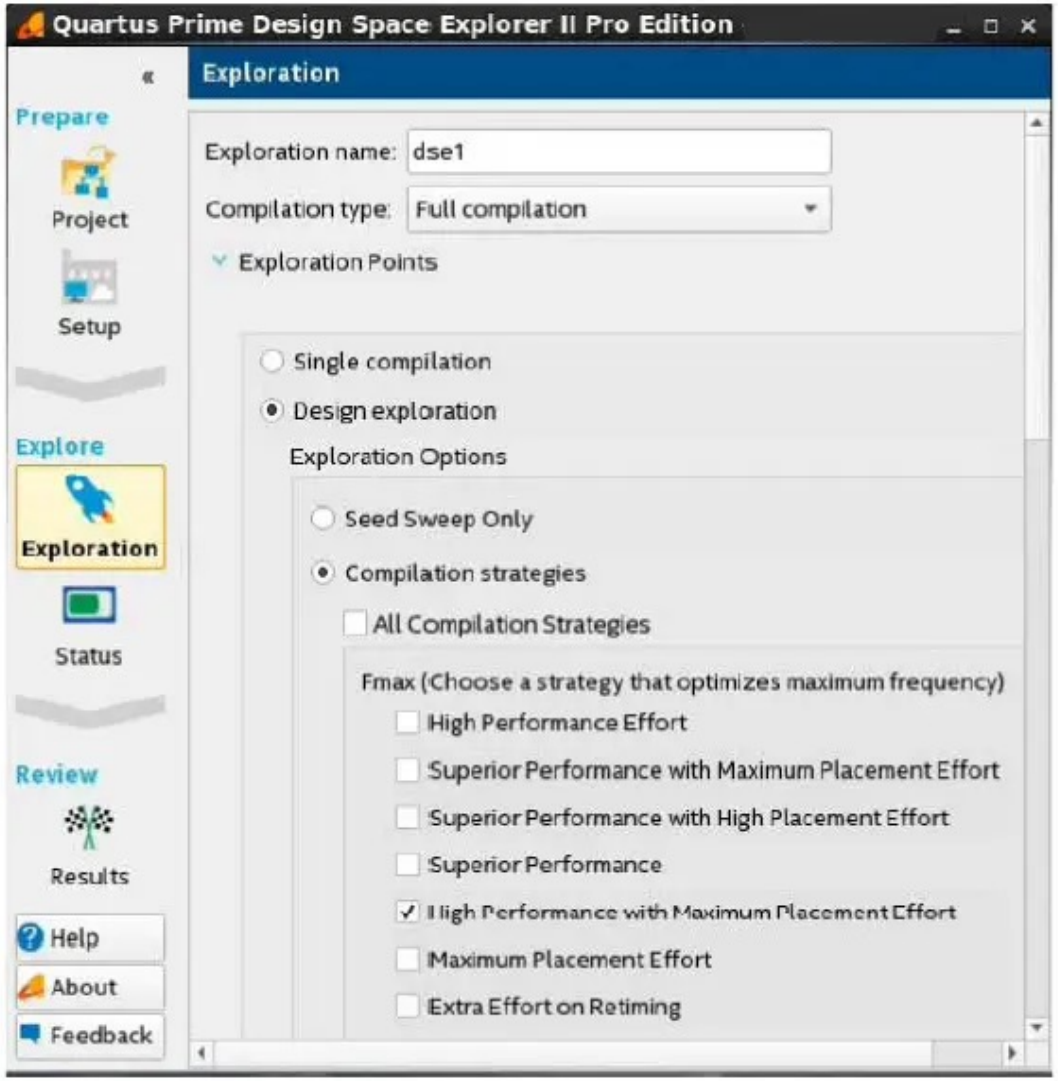
Сравнить производительность
различных серий ПЛИС
и возможности среды проектирования

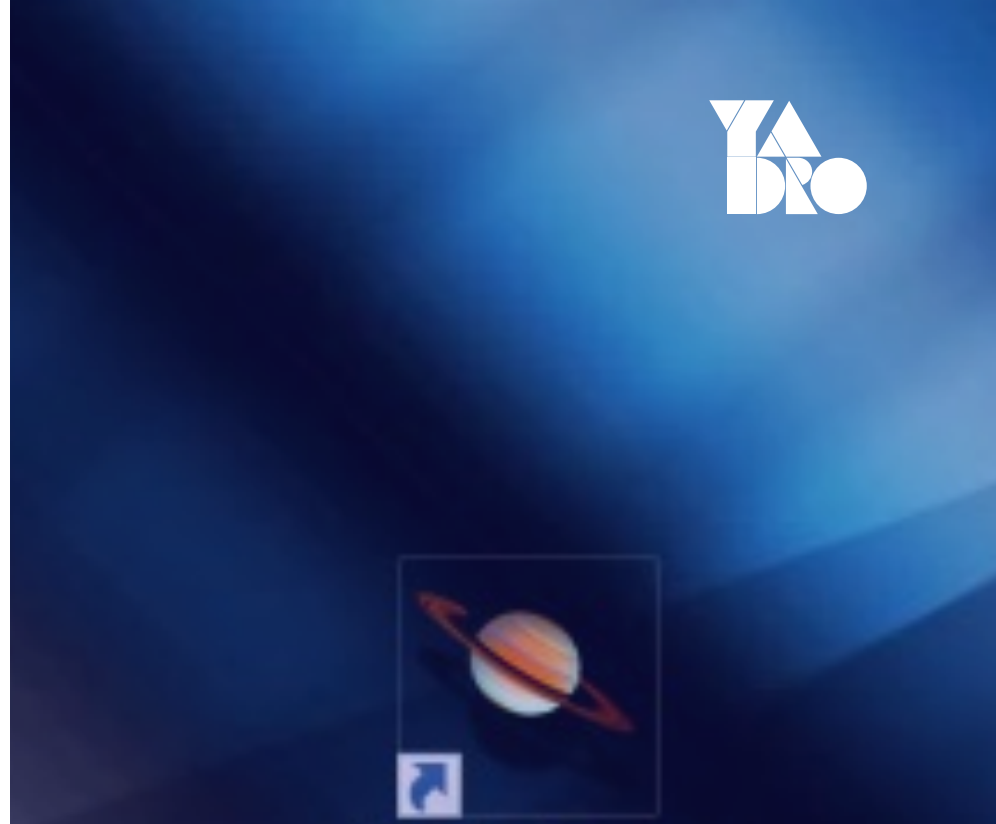
При выборе ЭКБ для проекта

Способы организации итеративной сборки

	Собирать несколько проектов вручную	Использовать design runs	Использовать инкрементальную компиляцию
✗	Трудоемко, результаты нельзя обобщить, выявить закономерности	Отсутствие понятия SEED, доступны только в AMD/Xilinx Vivado	По-разному реализована в САПР, не в каждой САПР поддержана
✓	Просто	Можно настроить из GUI	Можно настроить из GUI

Утилита Design Space Explorer





Пример использования Design Space Explorer

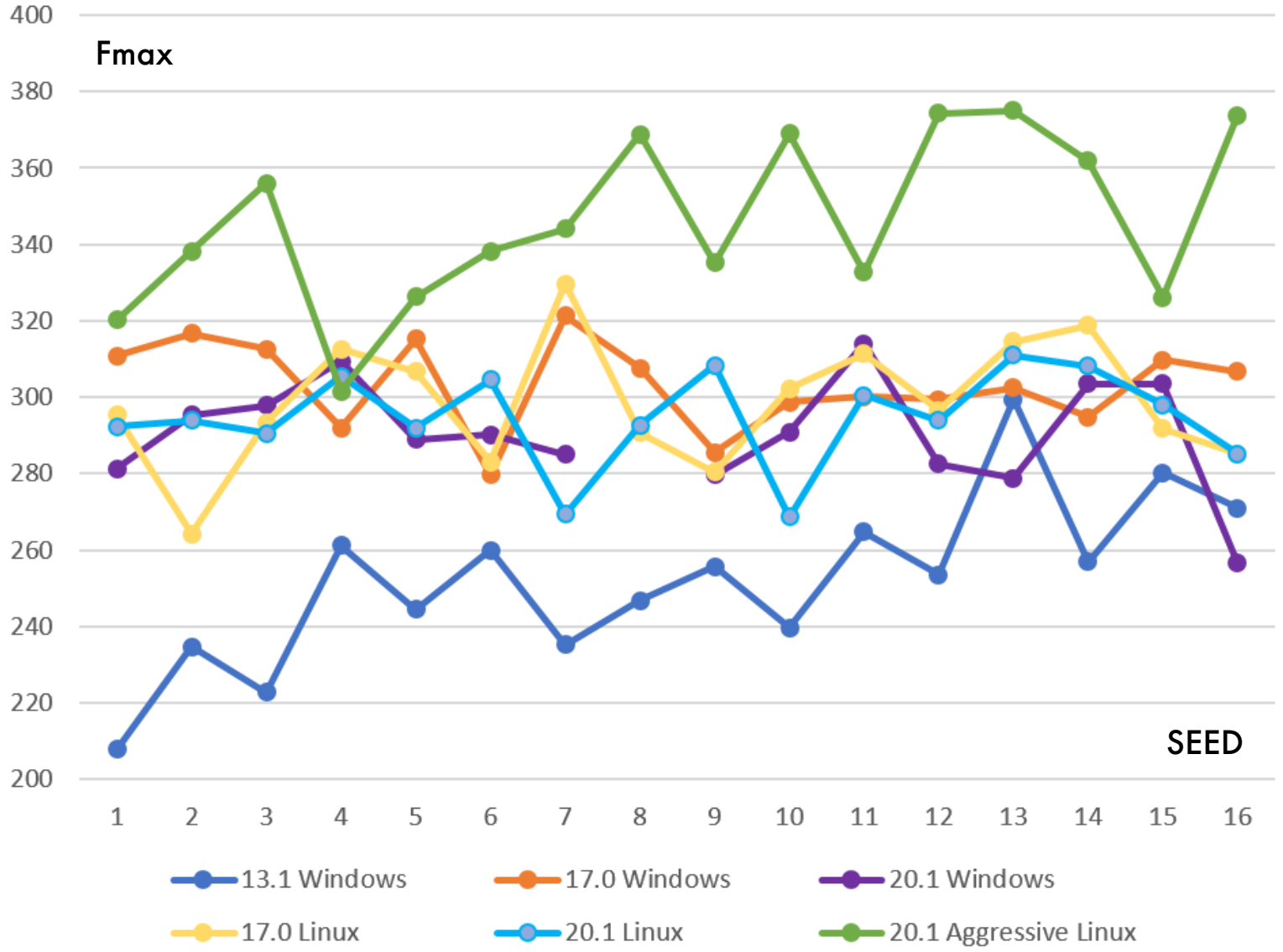
SEED	Windows 7				Manjaro Linux		
	13.1	13.1 Aggressive	17.0	20.1	17.0	20.1	20.1 Aggressive
1	207	207	310	281	295	292	320
2	234	234	316	295	264	293	338
3	222	222	312	297	293	290	356
4	261	261	291	309	312	305	301
5	244	244	315	288	306	291	326
6	260	260	279	290	283	304	338
7	235	235	321	285	329	269	344
8	246	246	307	-	290	292	368
9	255	255	285	279	280	308	335
10	239	239	298	290	302	268	369
11	264	264	300	313	311	300	332
12	253	253	299	282	296	293	374
13	299	299	302	278	314	310	374
14	256	256	294	303	318	308	361
15	280	280	309	303	291	298	326
16	271	271	306	256	285	285	373
AVG	**252**	**252**	**303**	**290**	**298**	**294**	**346**

Сборка тестового проекта проводилась:

- В различных версиях среды Quartus Prime (столбцы)
- С различными значениями SEED (строки)

Показаны значения Fmax для каждого запуска сборки

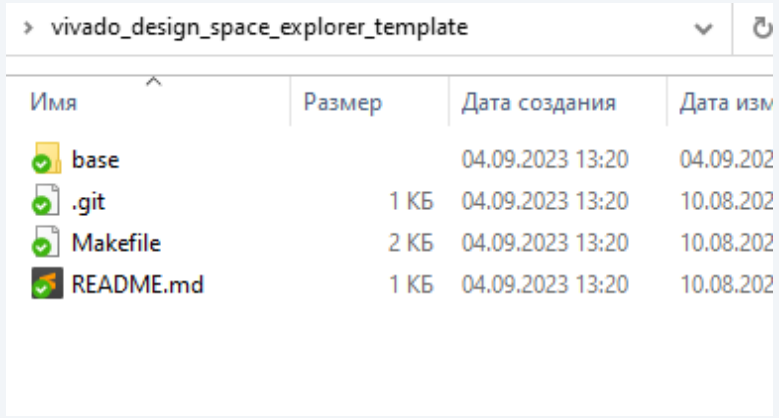
Пример использования Design Space Explorer



Сделанные выводы:

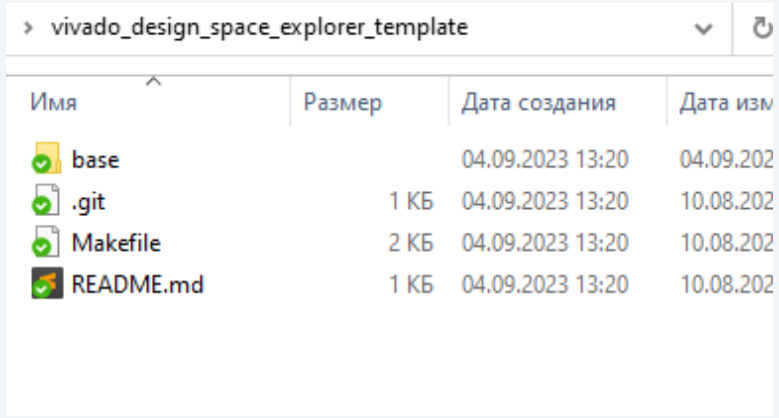
- В САПР Quartus 13.1 показано наихудшее качество PnR
- Наилучший результат достигнут при сборке в САПР Quartus 20.1 + настройки Aggressive performance

Проект vivado_design_space_explorer_template

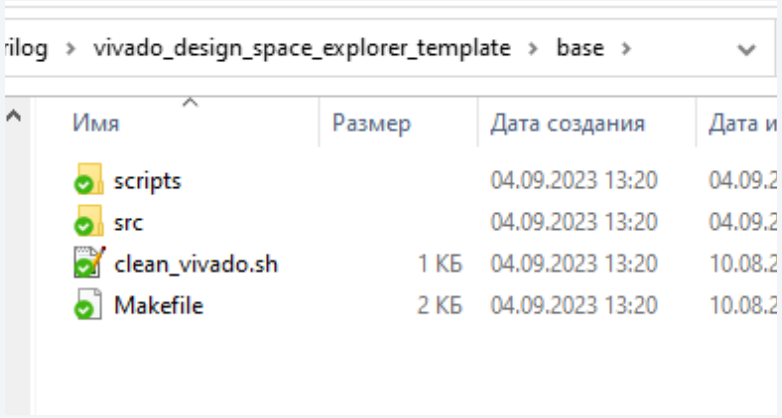


Корневая директория, головной make-файл

Проект vivado_design_space_explorer_template

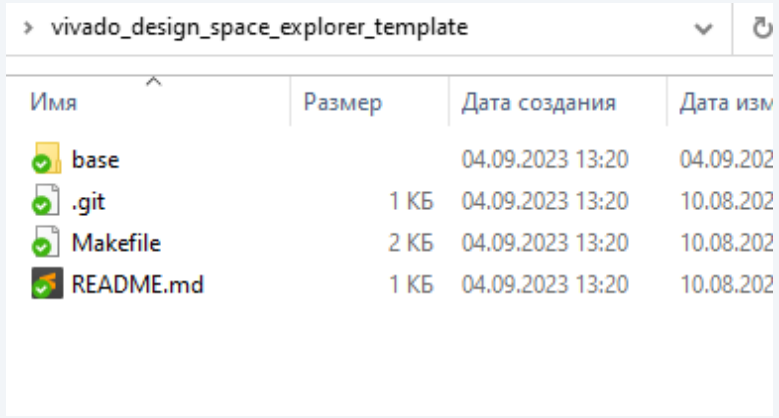


Корневая директория, головной make-файл

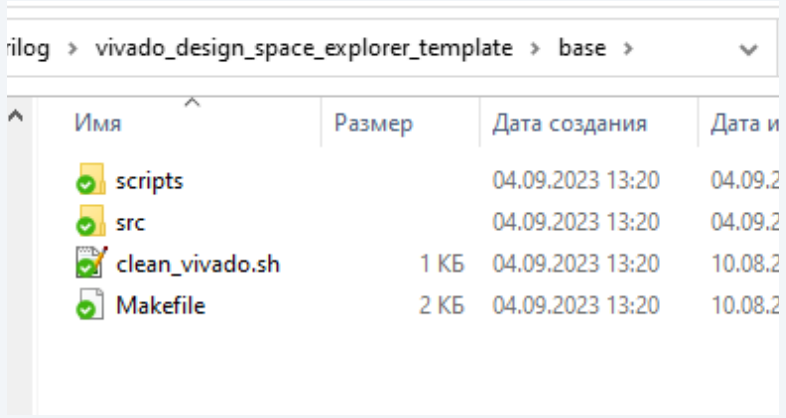


Директория референсного проекта, на основе которого будут генерироваться копии

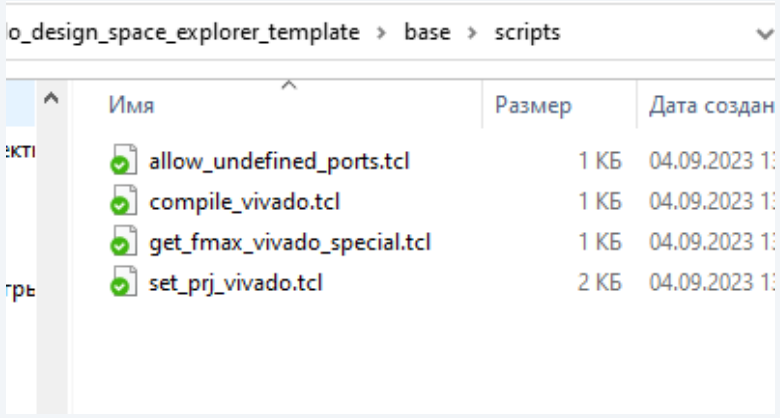
Проект vivado_design_space_explorer_template



Корневая директория, головной make-файл



Директория референсного проекта, на основе которого будут генерироваться копии



Скрипты компиляции проекта

Makefile (внешний)

```
VAR_START = 5
VAR_STOP = 64
VAR = $(shell seq $(VAR_START) ${VAR_STOP})

JOBS = $(addprefix job,${VAR})

all: fmax.csv
echo '$@ success'

${JOBS}: job%:
	mkdir -p ./$*; \
	cp -r ./base/* ./$*; \
	echo "// Do not edit. This file is auto-generated" > ./$*/src/define.vh; \
	echo "\`define WIDTH $*" >> ./$*/src/define.vh; \
	$(MAKE) -C ./$* all

fmax.csv: ${JOBS}
	v=$(VAR_START); while [ "$$v" -le $(VAR_STOP) ]; do \
	    cd $$v; \
	    vivado -mode batch -source scripts/get_fmax_vivado_special.tcl \
	              | tail -n2 | head -n1 >> ../fmax.csv; \
	    cd ..; \
	    v=$$((v+1)); \
	done
```

Задаем диапазон перестройки «переменной сборки»

Makefile (внешний)

```
VAR_START = 5
VAR_STOP = 64
VAR = $(shell seq $(VAR_START) ${VAR_STOP})

JOBS = $(addprefix job,${VAR})

all: fmax.csv
echo '$@ success'

${JOBS}: job%:
	mkdir -p ./$*; \
	cp -r ./base/* ./$*; \
	echo "// Do not edit. This file is auto-generated" > ./$*/src/define.vh; \
	echo "\`define WIDTH $*" >> ./$*/src/define.vh; \
	$(MAKE) -C ./$* all

fmax.csv: ${JOBS}
	v=$(VAR_START); while [ "$$v" -le $(VAR_STOP) ]; do \
	    cd $$v; \
	    vivado -mode batch -source scripts/get_fmax_vivado_special.tcl \
	              | tail -n2 | head -n1 >> ../fmax.csv; \
	    cd ..; \
	    v=$$((v+1)); \
	done
```

Задаем диапазон перестройки «переменной сборки»

Создаем копии базового проекта и запускаем make для каждого из них

Makefile (внешний)

```
VAR_START = 5
VAR_STOP = 64
VAR = $(shell seq $(VAR_START) ${VAR_STOP})

JOBS = $(addprefix job,${VAR})

all: fmax.csv
echo '$@ success'

${JOBS}: job%:
	mkdir -p ./$*; \
	cp -r ./base/* ./$*; \
	echo "// Do not edit. This file is auto-generated" > ./$*/src/define.vh; \
	echo "\`define WIDTH $*" >> ./$*/src/define.vh; \
	$(MAKE) -C ./$* all

fmax.csv: ${JOBS}
	v=$(VAR_START); while [ "$$v" -le $(VAR_STOP) ]; do \
	    cd $$v; \
	    vivado -mode batch -source scripts/get_fmax_vivado_special.tcl \
	              | tail -n2 | head -n1 >> ../fmax.csv; \
	    cd ..; \
	    v=$$((v+1)); \
	done
```

Задаем диапазон перестройки «переменной сборки»

Создаем копии базового проекта и запускаем make для каждого из них

Индивидуальный define.vh для каждого запуска сборки

Makefile (внешний)

```
VAR_START = 5
VAR_STOP = 64
VAR = $(shell seq $(VAR_START) ${VAR_STOP})

JOBS = $(addprefix job,${VAR})

all: fmax.csv
echo '$@ success'

${JOBS}: job%:
	mkdir -p ./$*; \
	cp -r ./base/* ./$*; \
	echo "// Do not edit. This file is auto-generated" > ./$*/src/define.vh; \
	echo "\`define WIDTH $*" >> ./$*/src/define.vh; \
	$(MAKE) -C ./$* all

fmax.csv: ${JOBS}
	v=$(VAR_START); while [ "$$v" -le $(VAR_STOP) ]; do \
	    cd $$v; \
	    vivado -mode batch -source scripts/get_fmax_vivado_special.tcl \
	              | tail -n2 | head -n1 >> ../fmax.csv; \
	    cd ..; \
	    v=$$((v+1)); \
	done
```

Задаем диапазон перестройки «переменной сборки»

Создаем копии базового проекта и запускаем make для каждого из них

Индивидуальный define.vh для каждого запуска сборки

Анализируем репорты Vivado по всем запускам и сводим данные в единый текстовый отчет

Makefile (внутренний)

```
PROJ = test
PART = xc7z020clg400-1

SRCS = src/main.sv src/define.vh src/fast_counter.sv
XDCS = src/timing.xdc
SCRIPTS = scripts/allow_undefined_ports.tcl

all: setup compile

setup: .setup.done

.setup.done: $(SRCS) $(SCRIPTS)
	echo $(PROJ) > .proj
	echo $(PART) > .part
	echo $(SRCS) | tr -s " " "\012" > .srcs
	echo $(XDCS) | tr -s " " "\012" > .xdcs
	echo $(SCRIPTS) | tr -s " " "\012" > .scripts
	vivado -mode batch -source scripts/set_prj_vivado.tcl
	rm -f .proj .part .srcs .xdcs .scripts

compile: .compile.done

.compile.done: .setup.done
vivado -mode batch -source scripts/compile_vivado.tcl
```

→ Перечисляем исходники проекта

Makefile (внутренний)

```
PROJ = test
PART = xc7z020clg400-1

SRCS = src/main.sv src/define.vh src/fast_counter.sv
XDCS = src/timing.xdc
SCRIPTS = scripts/allow_undefined_ports.tcl
```

Перечисляем исходники проекта

```
all: setup compile

setup: .setup.done
```

```
.setup.done: $(SRCS) $(SCRIPTS)
	echo $(PROJ) > .proj
	echo $(PART) > .part
	echo $(SRCS) | tr -s " " "\012" > .srcs
	echo $(XDCS) | tr -s " " "\012" > .xdcs
	echo $(SCRIPTS) | tr -s " " "\012" > .scripts
	vivado -mode batch -source scripts/set_prj_vivado.tcl
	rm -f .proj .part .srcs .xdcs .scripts
```

Передаем параметры в TCL-скрипты через текстовые файлы

```
compile: .compile.done

.compile.done: .setup.done
vivado -mode batch -source scripts/compile_vivado.tcl
```

Makefile (внутренний)

```
PROJ = test
PART = xc7z020clg400-1

SRCS = src/main.sv src/define.vh src/fast_counter.sv
XDCS = src/timing.xdc
SCRIPTS = scripts/allow_undefined_ports.tcl
```

Перечисляем исходники проекта

```
all: setup compile

setup: .setup.done
```

```
.setup.done: $(SRCS) $(SCRIPTS)
	echo $(PROJ) > .proj
	echo $(PART) > .part
	echo $(SRCS) | tr -s " " "\012" > .srcs
	echo $(XDCS) | tr -s " " "\012" > .xdcs
	echo $(SCRIPTS) | tr -s " " "\012" > .scripts
	vivado -mode batch -source scripts/set_prj_vivado.tcl
	rm -f .proj .part .srcs .xdcs .scripts
```

Передаем параметры в TCL-скрипты через текстовые файлы

```
compile: .compile.done
```

```
.compile.done: .setup.done
vivado -mode batch -source scripts/compile_vivado.tcl
```

Запускаем сборку с помощью TCL

Скрипт set_prj_vivado.tcl

```
set proj [split [read [open ".proj" r]] "\n"]
set part [split [read [open ".part" r]] "\n"]
set srcs [split [read [open ".srcs" r]] "\n"]
set xdcs [split [read [open ".xdcs" r]] "\n"]
set scripts [split [read [open ".scripts" r]] "\n"]

create_project -force ${proj} . -part ${part}

add_files -fileset sources_1 ${srcs}
update_compile_order -fileset sources_1

add_files -fileset constrs_1 ${xdcs}

add_files -fileset utils_1 ${scripts}

exec touch .setup.done
```

Считываем аргументы скрипта в переменные TCL

Скрипт set_prj_vivado.tcl

```
set proj [split [read [open ".proj" r]] "\n"]
set part [split [read [open ".part" r]] "\n"]
set srcs [split [read [open ".srcs" r]] "\n"]
set xdcs [split [read [open ".xdcs" r]] "\n"]
set scripts [split [read [open ".scripts" r]] "\n"]
```

Считываем аргументы скрипта в переменные TCL

```
create_project -force ${proj} . -part ${part}

add_files -fileset sources_1 ${srcs}
update_compile_order -fileset sources_1

add_files -fileset constrs_1 ${xdcs}

add_files -fileset utils_1 ${scripts}
```

Пользуемся TCL-командами Vivado для создания проекта

```
exec touch .setup.done
```

Скрипт compile_vivado.tcl

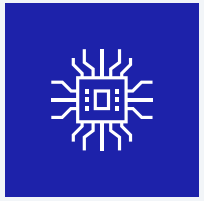
```
open_project test.xpr

reset_runs impl_1
launch_runs impl_1
wait_on_run impl_1

exec touch .compile.done
```

→ Собираем проект!

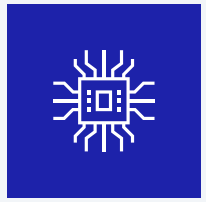
Преимущества разработанного решения



Компактный код, который легко адаптировать под задачу

Например, сделать итерации по двум переменным

Преимущества разработанного решения



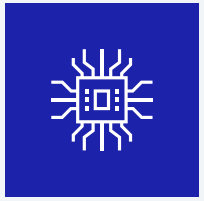
Компактный код, который легко адаптировать под задачу

Например, сделать итерации по двум переменным

Минимальное использование особенностей конкретной САПР

Подходит также для Quartus Prime, Gowin EDA, IceCube ...

Преимущества разработанного решения



Компактный код, который легко адаптировать под задачу

Например, сделать итерации по двум переменным

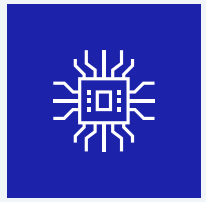
Минимальное использование особенностей конкретной САПР

Подходит также для Quartus Prime, Gowin EDA, IceCube ...

Механизм генерации проектов не зависит от среды проектирования

make работает почти везде

Преимущества разработанного решения

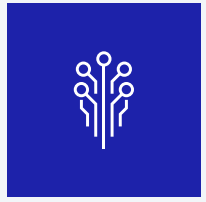


Компактный код, который легко адаптировать под задачу

Например, сделать итерации по двум переменным

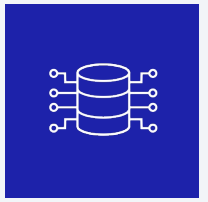
Минимальное использование особенностей конкретной САПР

Подходит также для Quartus Prime, Gowin EDA, IceCube ...



Механизм генерации проектов не зависит от среды проектирования

make работает почти везде



«Встроенные» возможности по анализу отчетов САПР

grep, sed, awk ...

Шаблон проекта размещен на гитхабе

https://github.com/pConst/vivado_design_space_explorer_template


БУДУЩЕЕ
В НАШИХ
РУКАХ