Vivado: Picasso Mode

Автор: **KeisN13**

2019 г.

Оглавление

Аннотация

Безумию все возрасты покорны

При проектировании каких-либо модулей на ПЛИС невольно иногда приходит в голову мысль о не совсем стандартном использовании самой среды проектирования и инструментов, которые она предоставляет для проектирования. В этой небольшой заметке мы рассмотрим, как с помощью инструмента управления средой, реализованного на Tcl, мы можем буквально рисовать на ПЛИС фотографии, картины, портреты и мемасики.

Такой необычный «маршрут проектирования» был реализован еще полтора года тому назад, но вот только сейчас пришла мысль оформить его в виде заметки, в которой имеется небольшая практика применения Tcl скриптов для управления средой проектирования, в данном случае Vivado. Однако при небольших доработках все легко может быть адаптировано под другие среды разработки, например Quartus II.

Введение

Разумеется, идея не пришла в голову из неоткуда. Её появлению способствовала моя тогдашняя занятость проектами по обработке изображений и управлению видеопотоками на FPGA. У каждого бывает такое, что сидя над решением какой-то проблемы в голову приходит всякая ересь, почему оно не работает или работает именно так как и должно, но не так как мы ожидаем.

При решении проблемы пришлось прибегнуть к одному из инструментов среды Vivado, а именно окрашиванию соответствующих компонентов и модулей в проекте после размещения и трассировки и вглядыванию в бесконечные временные диаграммы.

В итоге, я окрасил несколько конфигурируемых логических блоков CLB в различные цвета, и меня «осенило» – это же пиксели изображения, так может попробовать нарисовать какую нить картинку, сопоставив каждому пикселю свой окрашенный CLB?… ну тут оно и понеслось

Чем нам поможет Tcl?

Предположим, что у нас есть небольшая картинка размера 100x100 пикселей. Теперь допустим, что для того чтобы окрасить CLB нам нужно совершить два действия: выбрать CLB и выбрать цвет. В картинке 100x100 у нас 10000 пикселей и делать такое окрашивание вручную достаточно утомительно, тем более что действия являются однотипными и повторяющимися. Таким образом, раскрашивать вручную каждый CLB это не есть выход и нужно воспользоваться Tcl и скриптам. Но с чего начать?

Первое, что пришло в голову – это найти нужную команду, отвечающую за назначение цвета выбранному элементу. К счастью, при ручном выполнении действий Vivado выводит соответствующие Tcl команды в консоль и вроде бы проблема с поиском должна быть решена максимально быстро. Однако не тут то было. Вывод команды на подсветку выбранных элементов Vivado просто игнорирует и единственным вариантом найти команду, а я был предельно уверен, что она должна быть, это окунуться с головой в гайд по Tcl командам, доступным в Vivado, а это почти 2000 страниц [1].

Не стоит отчаиваться, по ключевому слову «highlight» быстро нашлась соответствующая команда, которая называется `highlight_objects`. Эта команда подсвечивает указанные или выбранные объекты в определённый цвет, задаваемый с помощью опций. Опции у команды `highlight_objects` следующие:

- `color_index <arg>` – (не обязательная) допустимое значение аргумента опции должно быть число от 1 до 20. Цвет, в который будет окрашен выбранный объект, определяется его порядковым номером из палитры предустановленных цветов, которую можно найти в Colors → Highlight в разделе меню Tools → Settings.
- `rgb <arg>` – (не обязательная) задает цвет выбранного объекта в формате RGB
- `color <arg>` – (не обязательная) подсвечивает выбранный объект в один из следующих цветов: red, green, blue, magenta, yellow, cyan и orange

Остальные опции команды относятся к системным настройкам самой команды и нам не пригодятся. Однако при использовании команды `highlight_objects` следует учитывать, что две и более опций окрашивания не могут применяться одновременно.

Очевидно, что для нашей задачи подходит опция, задающая произвольный цвет в формате RGB – опция `rgb`

Теперь не плохо бы было получить значения пикселей изображения, но найти изображение, которое бы было представлено в формате bitmap, мне не удалось. Открывая каждый файл

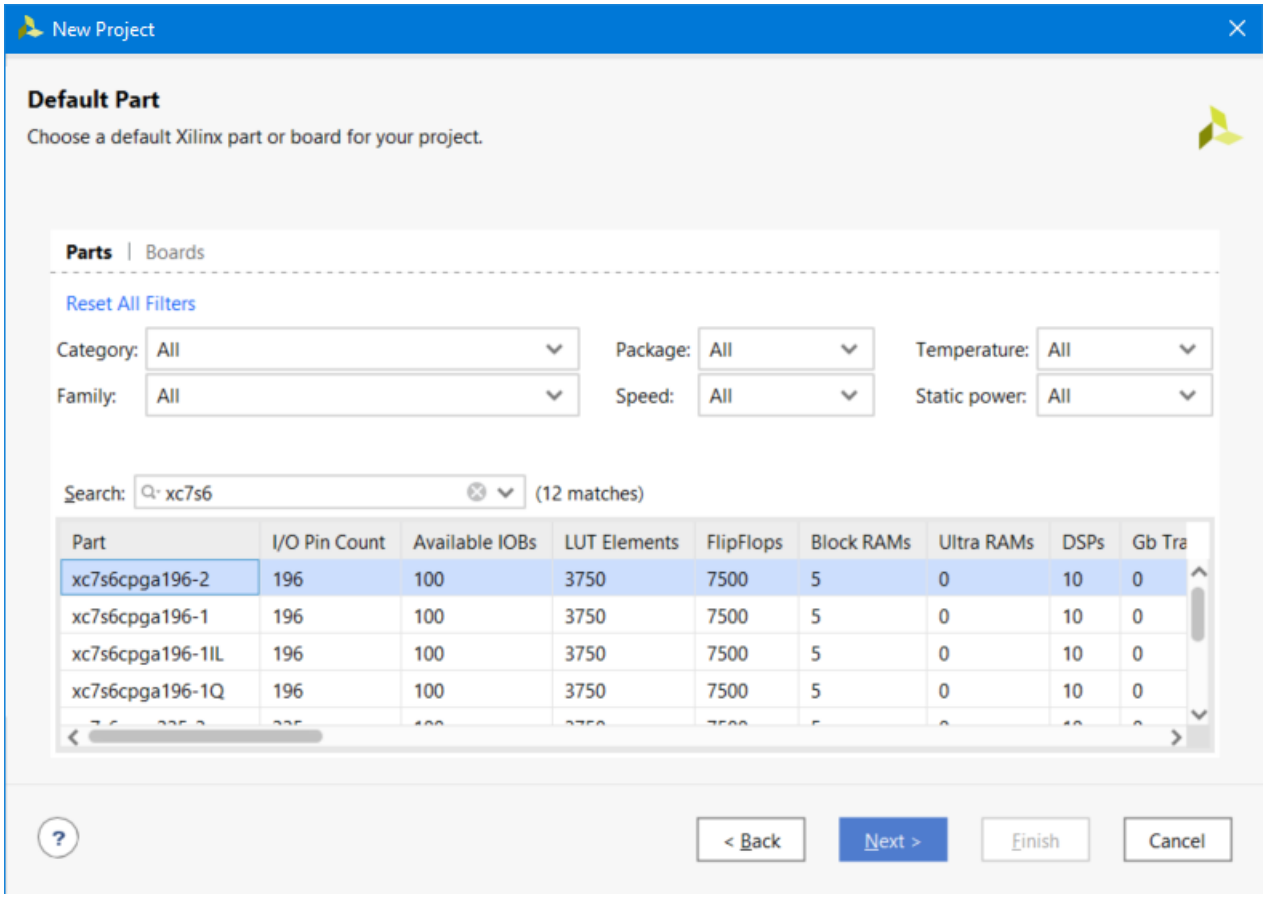
текстовым редактором, не удавалось найти строки со значением пикселей. Разумеется, писать программу преобразования изображений в формат bitmap я не стал, а просто полез в интернет искать готовое решение. Искать пришлось не слишком долго. Как оказалось, задача преобразования изображения в формат bitmap (то есть когда мы видим значения пикселей несжатого изображения) достаточно актуальна (наверное, такую задачу задают студентам-программистам в качестве домашнего задания к лабораторной работе). Не долгий поиск привел на github, откуда и была скачана программа Image2Bitmap [2].

Программа требует на вход изображения и на выходе выдает значения пикселей в виде си массива с шестнадцатеричными значениями пикселей в формате RGB565. Этот формат говорит, что на кодирование цвета для красной компоненты используется 5 бит, зеленой 6 бит и синей 5 бит. Этого оказалось вполне достаточно для работы. Теперь лишь требуется отобразить полученные значения непосредственно на окрашиваемые секции (slice).

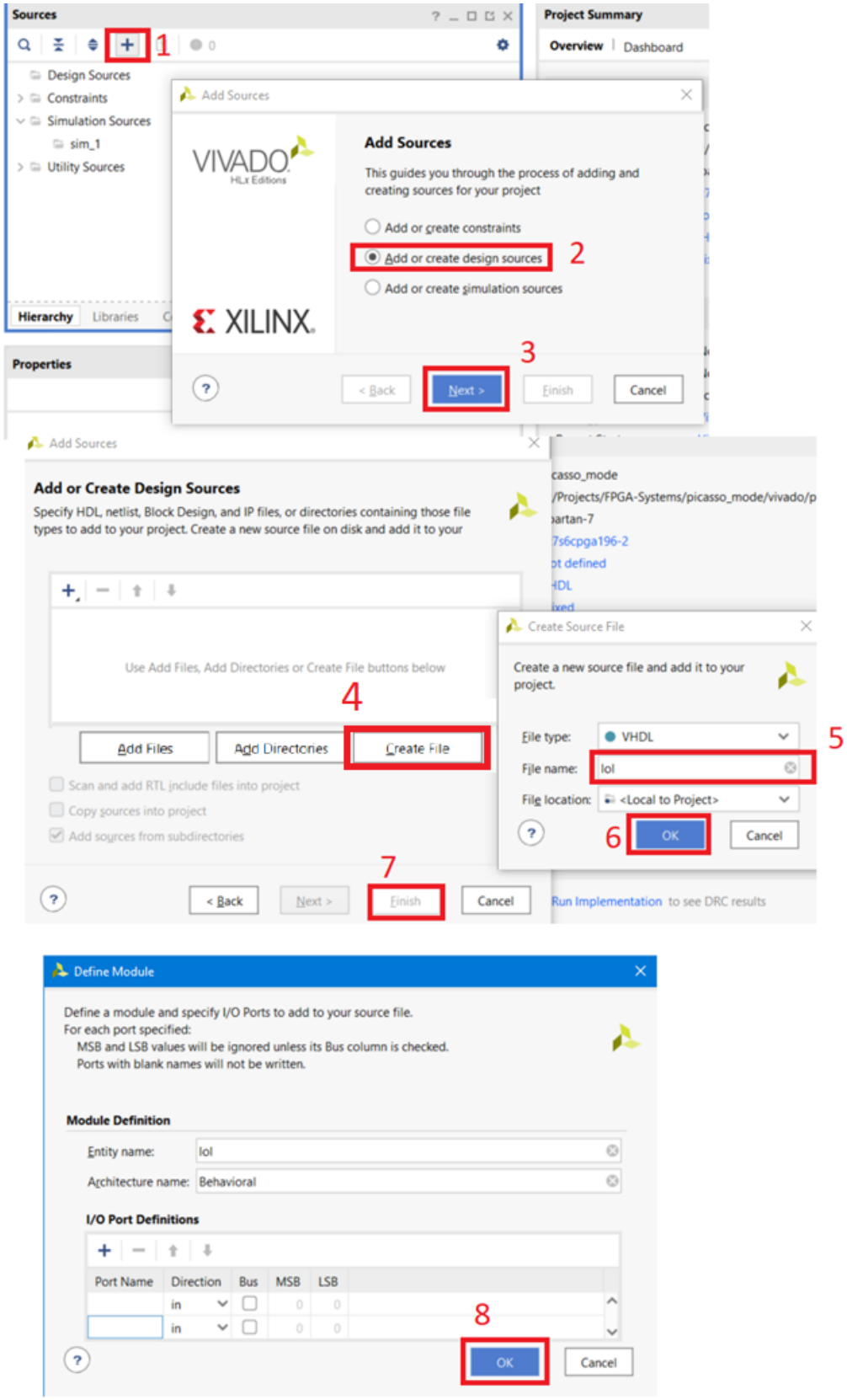
Выбор ПЛИС

Чем больше ПЛИС, тем больше в ней логических ресурсов, а значит и само «поле для творчества» больше и картинка будет чётче. Следует сразу отметить, что «разукрашивание» достаточно долгий процесс и может занять прилично времени, зависящее от размеров изображения. Для проведения тестирования стоит выбрать ПЛИС с небольшим количеством ресурсов. Например, семейства Spartan-7. После окончания тестирования, можно изменить ПЛИС на более «жирную», например, из семейства Ultrascale+.

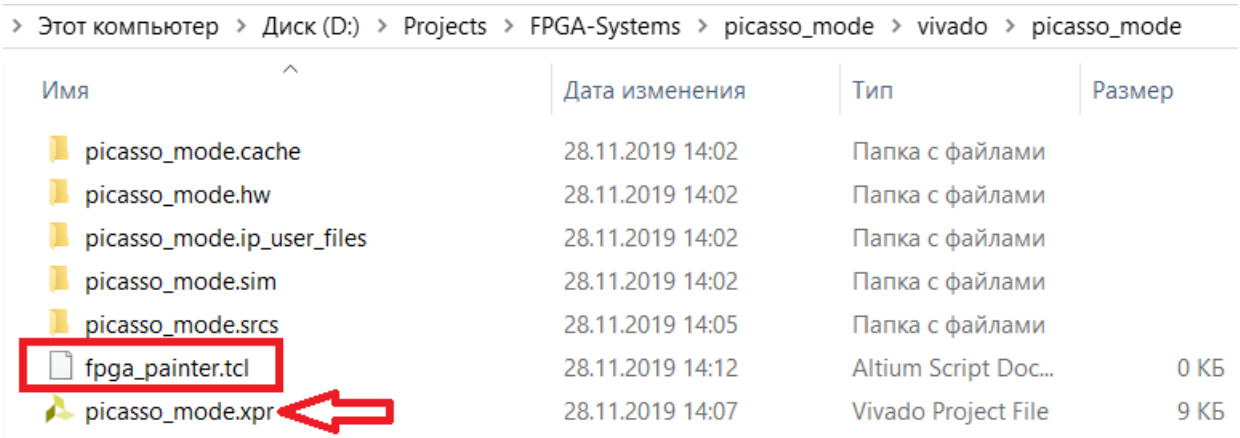
1. Запускаем Vivado и создаем проект
2. Выбираем кристалл **xc7s6cpga196-2**, на котором будем рисовать тестовое изображение



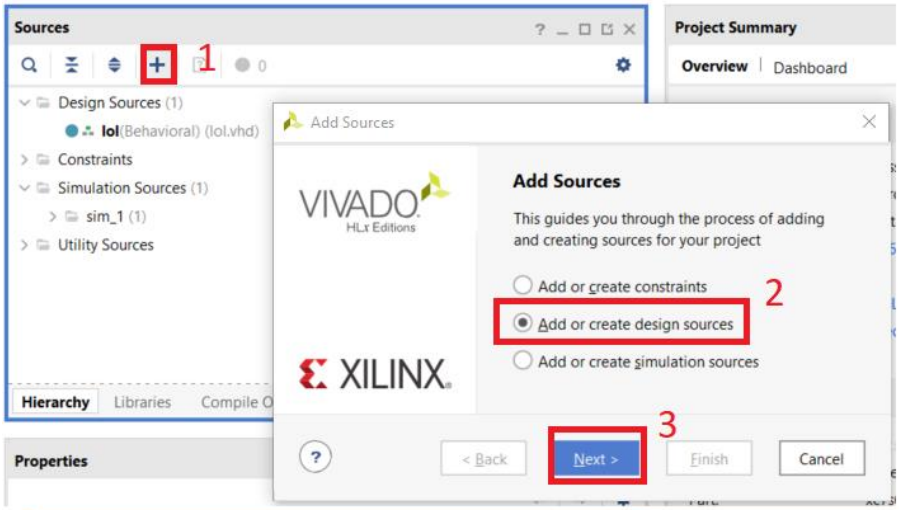
3. Для отображения нарисованного нам понадобится открыть само изображение кристалла, однако, это можно сделать после этапа синтеза либо elaborate. В Vivado нам для этого понадобится создать модуль-пустышку на любом языке.

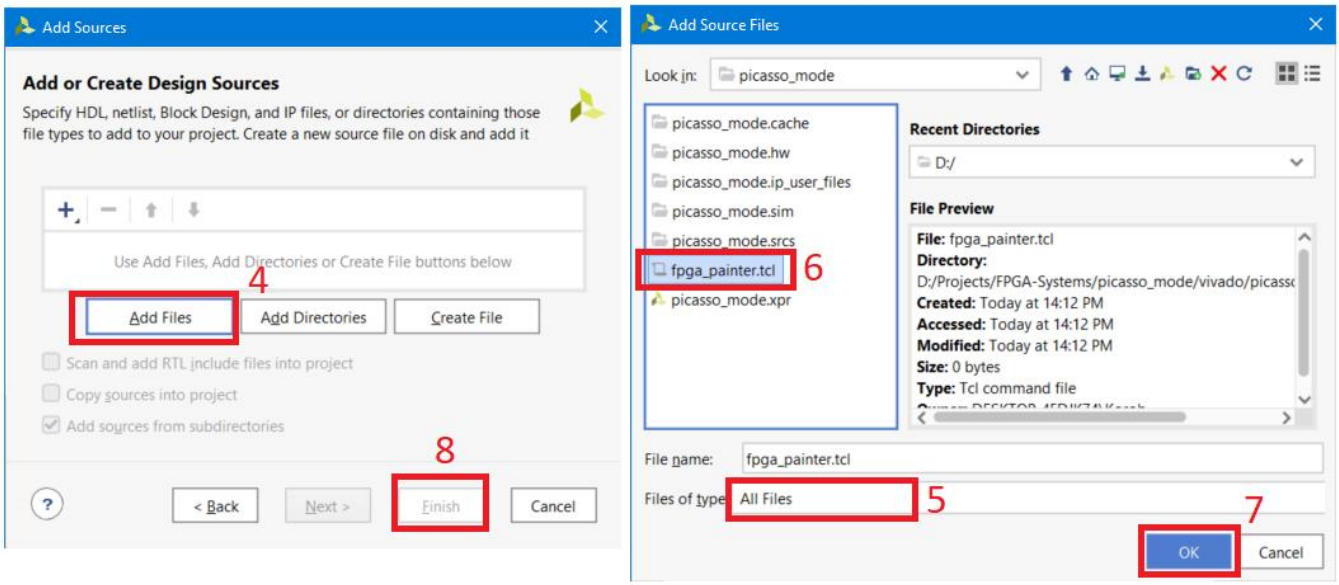


4. Добавим в проект Tcl скрипт.
    a. Для этого, создайте в папке с проектом файл с расширением «.tcl», например «fpga_painter.tcl»

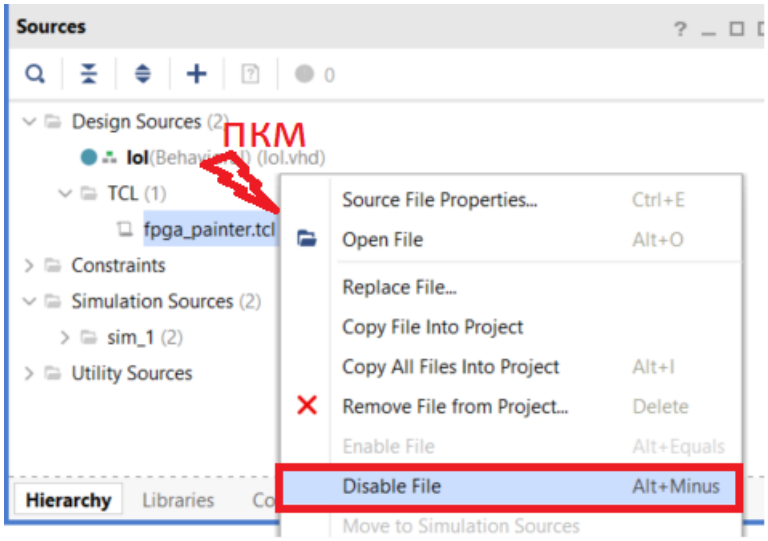


b. Перейдите в Vivado, и добавите этот файл в проект.

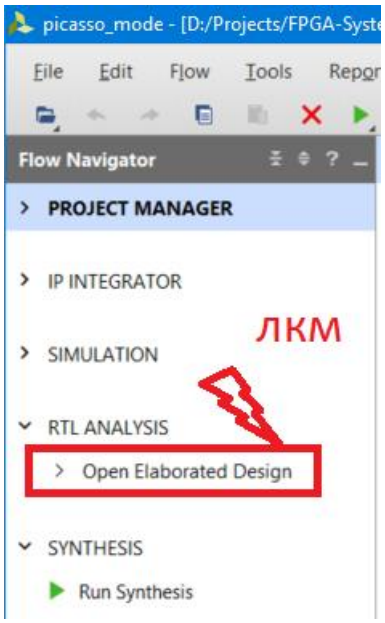




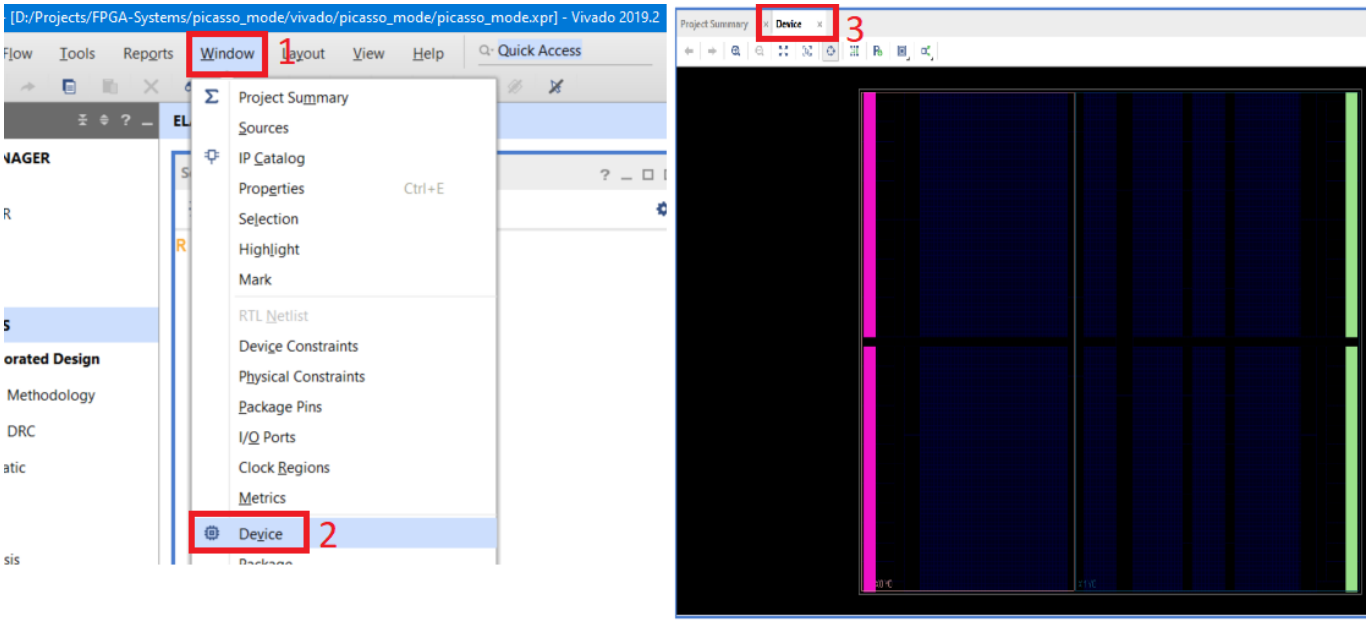
c. После окончания обновления иерархии проекта, сделайте файл неактивным.



5. После создания модуля, он появится в окне иерархии проекта и нам будет доступна кнопка Open Elaborate Design. Нажимаем её.



6. После открытия Elaborate Design переходим в Window→Device. Появится отображение поля нашей ПЛИС.



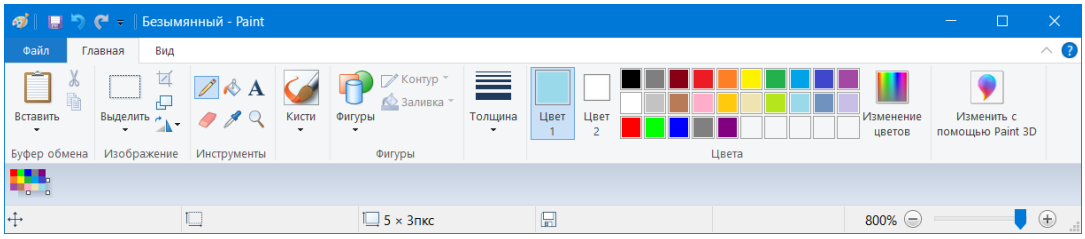
7. Подготовка закончена, приступаем к написанию скрипта.

Определение параметров и процедур

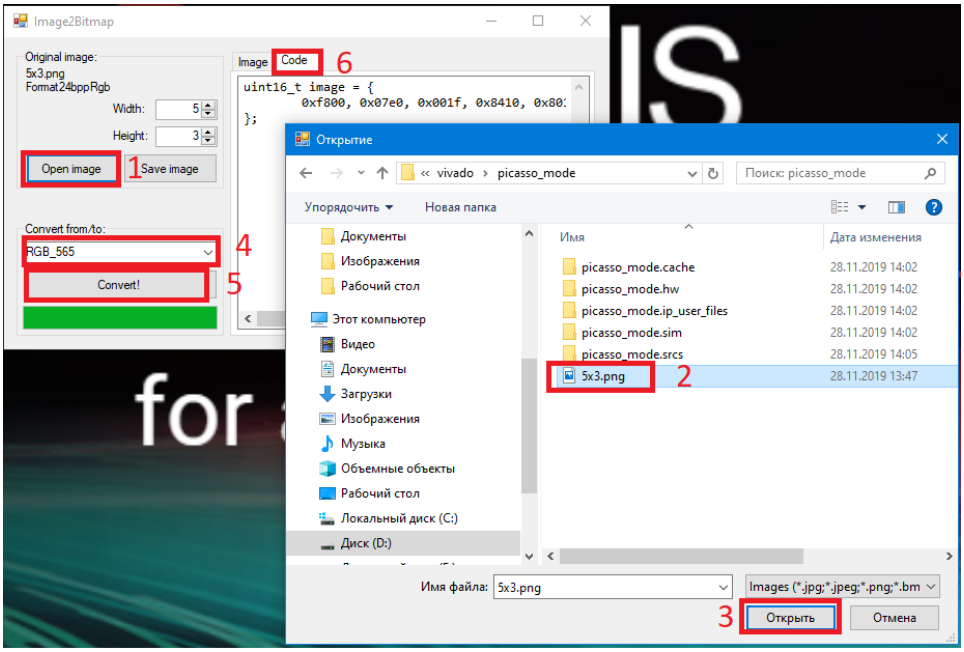
Тестовое изображение

Для начала давайте отладим алгоритм/код как таковой на небольшом изображении, скажем 5x3, а затем запустим его «на полную катушку».

8. Открываем Paint, ограничиваем поле изображения 5x3 пиксела (можно взять любые цвета). Сохраните файл как «5x3.png»



9. Откроем программу Image2Bitmap и преобразуем нашу картинку в массив RGB565.

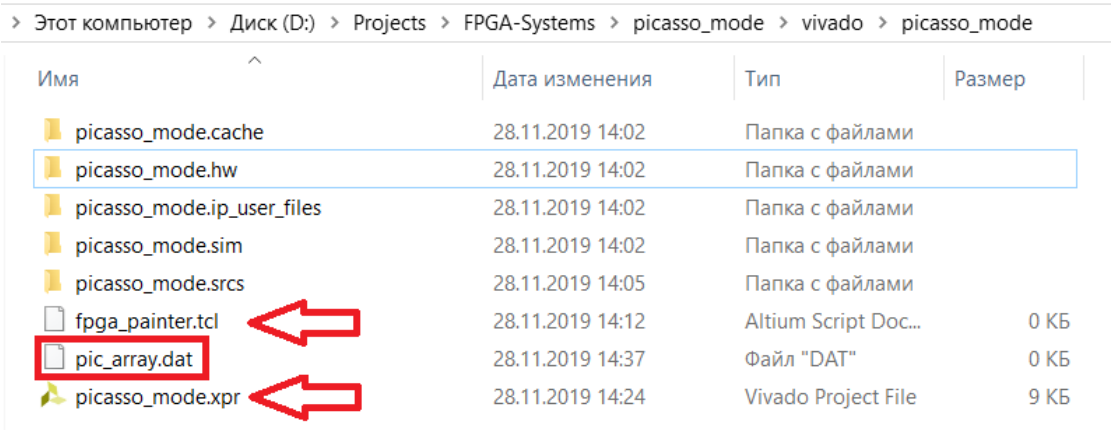


10. После преобразования программа выдаст нам массив из 15 пикселей

```
uint16_t image = {
    0xf800, 0x07e0, 0x001f, 0x8410, 0x8010, 0xfbe4, 0xff80, 0x2589, 0x051d,
    0x3a59, 0xa254, 0xbbca, 0xfd79, 0xef36, 0x9edd,
};
```

Подготовка данных

Перед тем как приступить к обработке пикселов преобразуем данные, выдаваемые программой Image2Bitmap в простой список, в котором будут записаны шестнадцатеричные значения пикселов. Сами данные программы мы скопируем и сохраним в файл «pic_array.dat», который следует расположить в папке с проектом.



При запуске создаваемого скрипта нам предстоит обработать файл «pic_array.dat». Стоит отметить, что количество элементов в строке, возвращаемой программой Image2Bitmap, не соответствует количеству пикселей в строке преобразуемого изображения, по этой причине мы и сформируем отдельный список «pixels».

```
#Создаём список, в котором сохраним значения пикселов изображения
set pixels [list ]
```

При чтении файла нужно игнорировать первую строку «uint16_t image = {» и последнюю «};». Для пропуска первой строки при чтении файла просто разместим чтение строки перед циклом чтения всего файла.

```
#Открываем файл с шестнадцетиричными значениями пикселов изображения
set fId [open {pic_array.dat} r]

#Читаем первйю строку из файла. Строка не нужна
gets $fId line
```

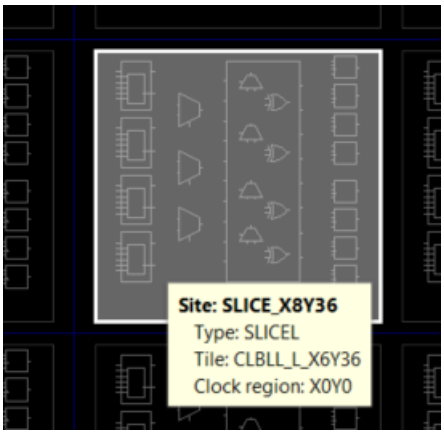
После чтения всего файла, мы увидим, что последняя строка файла «};» стала элементом списка, который просто удаляется.

```
#Читаем остальные строки. В них хранятся значения пикселов.
while {[gets $fId line] > 0} {
    set pixels [concat $pixels $line]
}

#Удаляем из списка pixels последний элемент "};", который формируется программой Image2Bitmap
set pixels [lrange $pixels 0 end-1]
```

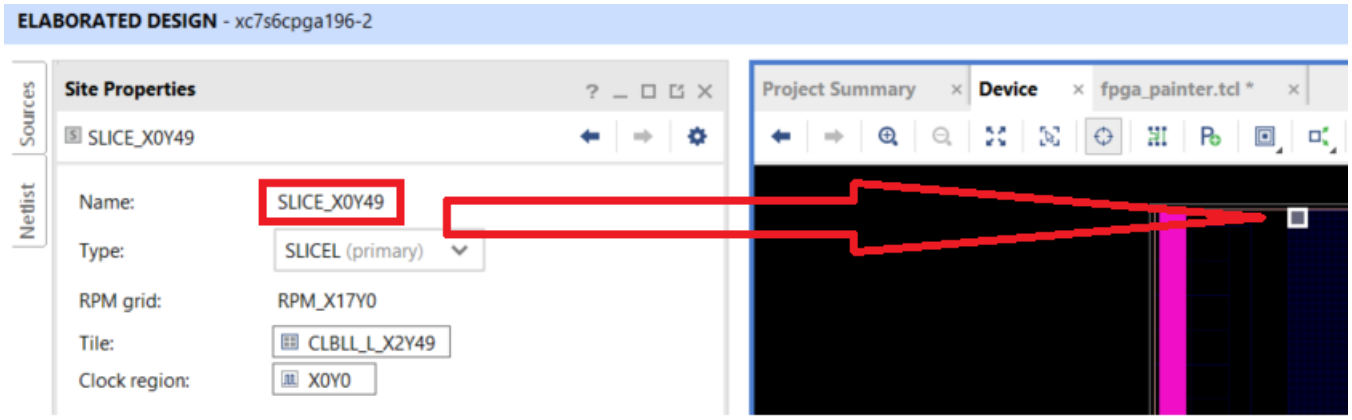
На этом формирование списка с шестнадцатеричными значениями пикселов закончено. Теперь приступим к их обработке.

Определение размера изображения

Еще раз взглянем на поле ПЛИС и изображение. Поле ПЛИС разбито на секции (SLICE), которые имеют соответствующие координаты по горизонтали «X» и вертикали «Y». Например, SLICE_X6Y36.



Изображение в свою же очередь имеет пиксели, так же с координатами по горизонтали и вертикали. При наложении изображения на ПЛИС нам следует совместить верхний левый пиксель с верхней левой секцией ПЛИС. В данном случае, выбранный кристалл имеет верхнюю секцию с координатой X0Y49.



Размер изображения будет определяться количеством секций в ПЛИС по горизонтали и вертикали. У выбранного кристалла горизонтальная координата секций изменяется от X0 до X131, а по вертикальная от Y49 до Y0. Отсюда следует, что теоретически мы можем нарисовать на выбранном кристалле изображение размером 132x50.

Начальные параметры

Подведём итог: начальными параметрами нашего скрипта будут:

1. Стартовая позиция секции по оси X: имя переменной start_x

```
#смещение левого верхнего пиксела изображения на поле ПЛИС
set start_x  0; #Горизонтальное
```

2. Стартовая позиция секции по оси Y: имя переменной start_y

```
set start_y  49; #Вертикальное
```

3. Ширина изображения (для тестового изображения равна 5): переменная w

```
set w 5; #Ширина изображения (количество пикселей в строке)
```

4. Высота изображения (для тестового изображения равна 3): переменная h

```
set h 3; #Высота изображения (количество строк)
```

Корректировка цвета пиксела

Программа Image2Bitmap выдает массив пикселов в формате RGB565 в виде 16 битного числа, записанного в шестнадцатеричном формате. Нам следует:

1. Преобразовать значение пиксела в двоичный формат. Это можно сделать с помощью процедуры hex2bin, которую можно найти в [3]

```
proc hex2bin hex {
    set t [list 0 0000 1 0001 2 0010 3 0011 4 0100 \
        5 0101 6 0110 7 0111 8 1000 9 1001 \
        a 1010 b 1011 c 1100 d 1101 e 1110 f 1111 \
        A 1010 B 1011 C 1100 D 1101 E 1110 F 1111]
    regsub {^0[xX]} $hex {} hex
    return [string map -nocase $t $hex]
}
```

2. Сопоставить биты с соответствующими цветовыми компонентами:
   - Красная компонента R[15:11]

```
#Берем двоичное значение красного цвета
set R_bin [string range $pix_bin 0 4]
```

- Зелёная компонента G[10:5]

```
set G_bin [string range $pix_bin 5 10]
```

- Синяя компонента B[4:0]

```
set B_bin [string range $pix_bin 11 15]
```

Пояснение: порядок изменён в виду того, что процедура hex2bin возвращает строку, в которой нумерация элементов начинается с 0, то есть 15-му биту соответствует 0-ой элемент строки, а 0-му биту 15-ый элемент строки

3. Преобразовать значение цветовой компоненты из binary в decimal. Это можно сделать с помощью процедуры bin2dec, которую можно найти [3]:

```
proc bin2dec bin {
    #returns integer equivalent of $bin
    set res 0
    if {$bin == 0} {
        return 0
    } elseif {[string match -* $bin]} {
        set sign -
        set bin [string range $bin[set bin {}] 1 end]
    } else {
        set sign {}
    }
    foreach i [split $bin {}] {
        set res [expr {$res*2+$i}]
    }
    return $sign$res
}
```

```
#Преобразуем его в десятичное
set R_dec [ bin2dec $R_bin ]
```

4. Преобразовать значения пикселов из RGB565 в формат RGB888, для более плавного отображения картинки. Это делается с помощью двух списков, которые можно найти в [4]. Как это работает:
    - Разрядность цветовых компонент R и B 5 бит. Взяв десятичное значение компоненты, мы сопоставим его с позицией числа, записанного в списке t5, а значение компоненты изменится на значение в таблице

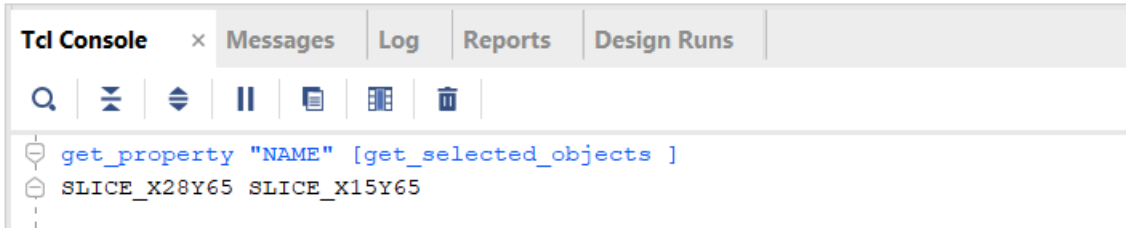
```
#RGB565 -> RGB888 using tables
set t5 [list 0   8   16   25   33   41   49   58 66 74 82 90 99 107 115 123 132\
 140 148 156 165 173 181 189 197 206 214 222 230 239 247 255]

#Корреектируем оттенок красного в соответствии с таблицей t5
set R [lindex $t5 $R_dec]

set B [lindex $t5 [ bin2dec $B_bin ] ]
```

    - Аналогично для компоненты G и таблицы t6

```
set t6 [list 0 4 8 12 16 20 24 28 32 36 40 45 49 53 57 61 65 69\
 73 77 81 85 89 93 97 101 105 109 113 117 121 125 130 134 138\
 142 146 150 154 158 162 166 170 174 178 182 186 190 194 198\
 202 206 210 215 219 223 227 231 235 239 243 247 251 255]

set G [lindex $t6 [ bin2dec $G_bin ]]
```

Определение наличия секции

Внутри некоторых ПЛИС имеются специальные ресурсы или пустоты, которые могут нарушить последовательную нумерацию координат секций. Например, на рисунке ниже видно, что нумерация секций прерывается (кристалл xc7s50)



По этой причине перед окрашиванием мы сначала проверим существование секции. Если она существует, то окрашиваем, если не существует, то переходим к следующему пикселу

```
#Убеждаемся, что секция для окрашивания существует
set cond [get_sites "SLICE_X${X}Y${Y}"]
if {$cond ne ""} {
```

Окрашивание секции

Цвет секции определён, наличие секции проверено. Теперь цвет нужно назначить секции с помощью команды `highlight_objects`:

```
    #Окрашиваем секцию
    highlight_objects [get_sites "SLICE_X${X}Y${Y}"]  -rgb "$R $G $B"
}
```

Вектор →Двухмерный массив

В начале, мы преобразовали данные изображения в список `pixels`, в котором хранится построчная развертка изображения. Для организации картинки введем две переменные, `x` и `y`, которые будут соответствовать положению пикселей в изображении. Последовательно считывая

элементы списка `pixels`, мы сформируем изображение, используя два цикла for: один по количеству строк, второй по положению пиксела в строке

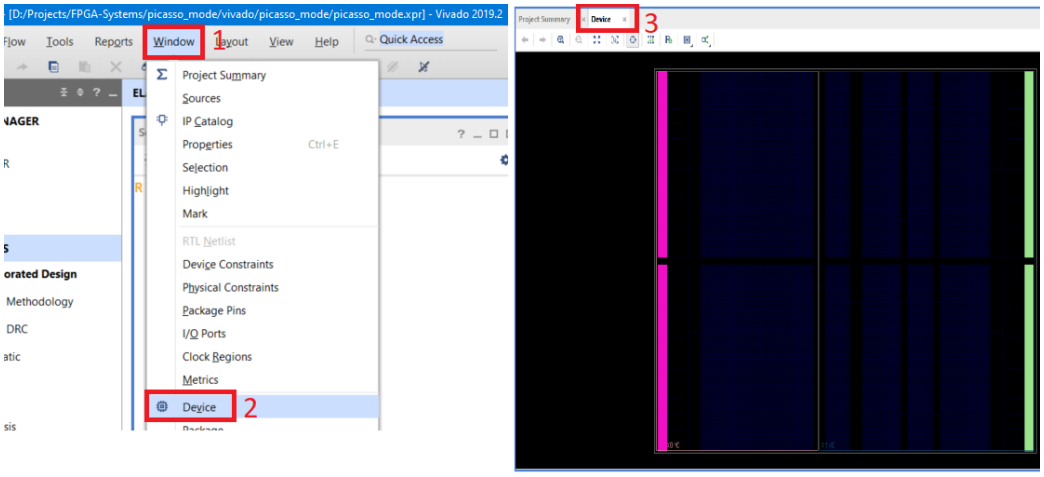
```
#Преобразуем последовательный список пикселов в изображение
for {set y 0} { $y < $h } {incr y} {

 # Обработка пикселов строки
     for {set x 0} { $x < $w } {incr x } {
```

Полный листинг скрипта

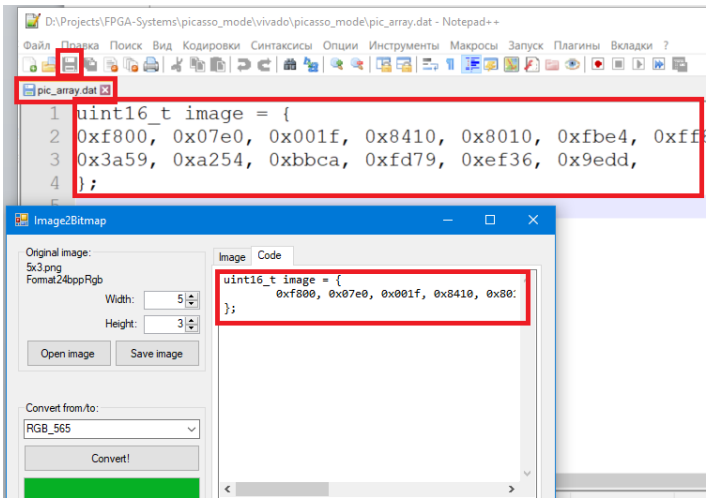
Листинг приведён в приложении

Тестирование

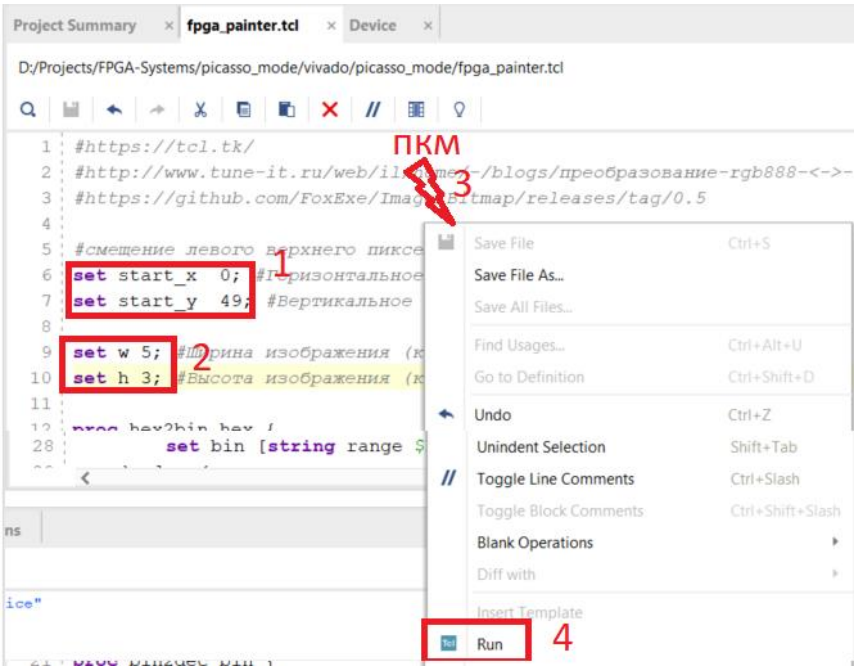
1. Для начала тестирования, убедитесь, что Вам доступно поле ПЛИС, т.е. открыт один из этапов проектирования: elaborated, synthesis или implemented. Для отображения поля ПЛИС выберите Window → Device



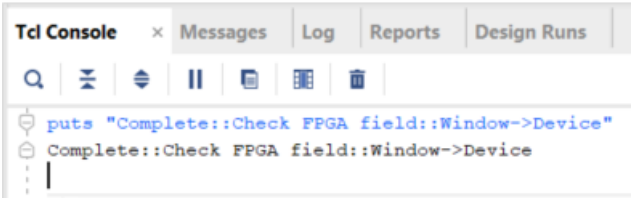
2. Откроем файл «pic_array.dat» и скопируем в файл данные из программы Image2Bitmap. Сохраняем файл



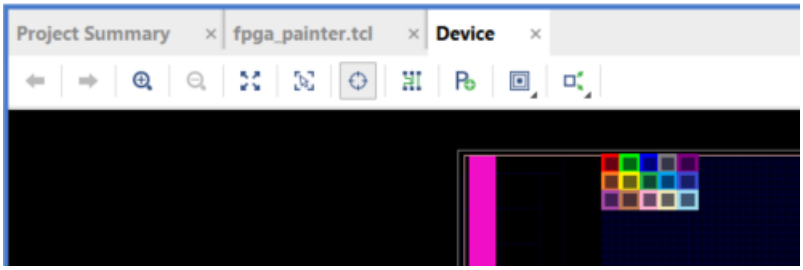
3. Откроем скрипт. Установим координату левого верхнего пиксела 0 и 49, размер тестового изображения 5 на 3 и запустим скрипт. Для этого в поле скрипта нажмите правой кнопкой и выберите Run.



4. Перейдите в Tcl console и убедитесь, что скрипт был выполнен.

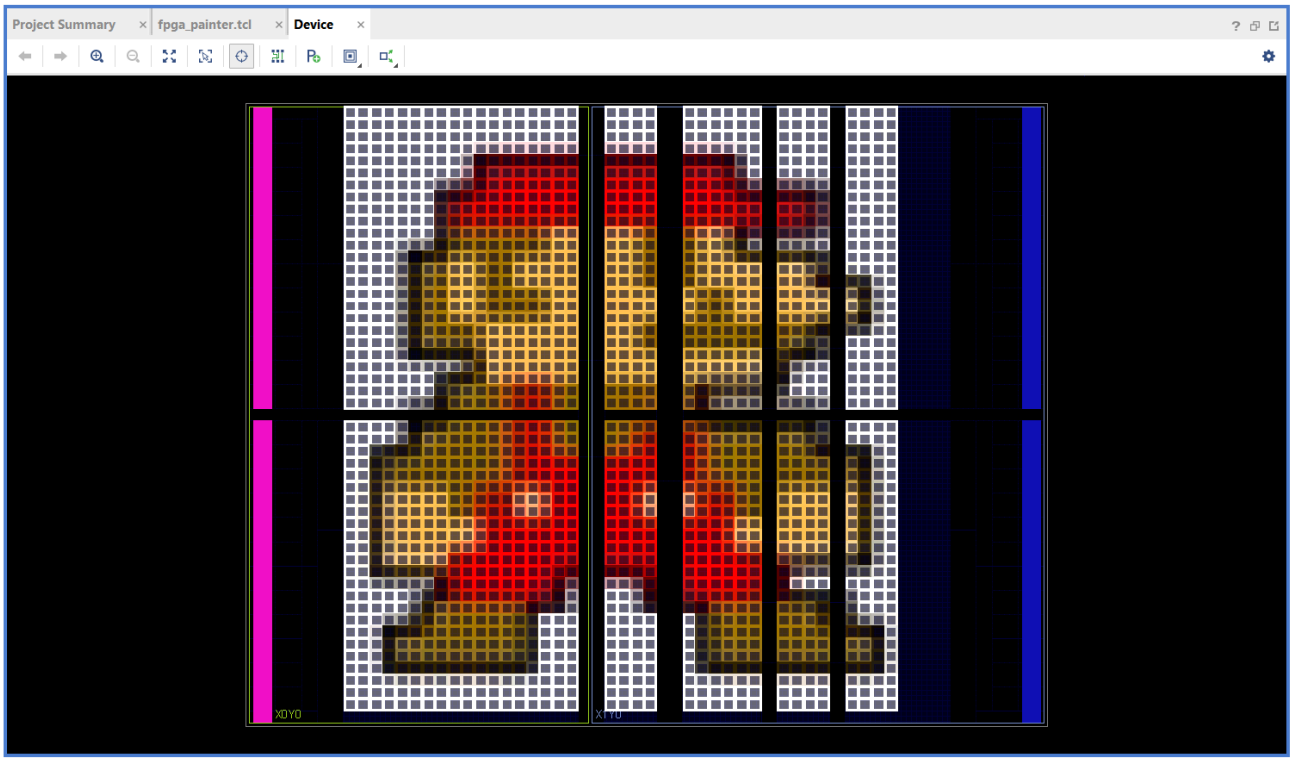


5. Перейдите во вкладку Device и убедитесь, что ячейки окрашены в соответствующий цвет.



6. Теперь можем взять любое изображение, преобразовать его до нужного размера и закрасить поле ПЛИС. Ниже приведены несколько примеров.

PS: произвольное назначение цветов заменяет стандартную настройку цветов Vivado



Список литературы

1. UG835. Vivado Design Suite TclCommand Reference Guide
2. https://github.com/FoxExe/Image2Bitmap/releases/tag/0.5
3. https://tcl.tk
4. http://www.tune-it.ru/web/il/home/-/blogs/преобразование-rgb888-<->-rgb565-и-защита-грибов-от-выцветания

Понравилась статья? Не забудьте поддержать автора

Приложение 1

Листинг скрипта

```
#https://tcl.tk/
#http://www.tune-it.ru/web/il/home/-/blogs/преобразование-rgb888-<->-rgb565-и-защита-грибов-от-выцветания
#https://github.com/FoxExe/Image2Bitmap/releases/tag/0.5

#смещение левого верхнего пиксела изображения на поле ПЛИС
set start_x  0; #Горизонтальное
set start_y  49; #Вертикальное

set w 5; #Ширина изображения (количество пикселей в строке)
set h 3; #Высота изображения (количество строк)

proc hex2bin hex {
   set t [list 0 0000 1 0001 2 0010 3 0011 4 0100 \
            5 0101 6 0110 7 0111 8 1000 9 1001 \
            a 1010 b 1011 c 1100 d 1101 e 1110 f 1111 \
            A 1010 B 1011 C 1100 D 1101 E 1110 F 1111]
   regsub {^0[xX]} $hex {} hex
   return [string map -nocase $t $hex]
}

proc bin2dec bin {
   #returns integer equivalent of $bin
   set res 0
   if {$bin == 0} {
      return 0
   } elseif {[string match -* $bin]} {
      set sign -
      set bin [string range $bin[set bin {}] 1 end]
   } else {
      set sign {}
   }
```

```
    foreach i [split $bin {}] {
        set res [expr {$res*2+$i}]
    }
    return $sign$res
}

###############
#RGB565 -> RGB888 using tables
set t5 [list 0  8  16  25  33  41  49  58 66 74 82 90 99 107 115 123 132\
 140 148 156 165 173 181 189 197 206 214 222 230 239 247 255]

set t6 [list 0 4 8 12 16 20 24 28 32 36 40 45 49 53 57 61 65 69\
 73 77 81 85 89 93 97 101 105 109 113 117 121 125 130 134 138\
 142 146 150 154 158 162 166 170 174 178 182 186 190 194 198\
 202 206 210 215 219 223 227 231 235 239 243 247 251 255]
###############

#Путь до скрипта. В нашем случае лежит в папке с проектом
set script_path [get_property DIRECTORY [get_projects *]]
cd $script_path

#Открываем файл с шестнадцетиричными значениями пикселов изображения
set fId [open {pic_array.dat} r]

#Создаём список, в котором сохраним значения пикселов изображения
set pixels [list ]

#Читаем первйю строку из файла. Строка не нужна
gets $fId line

#Читаем остальные строки. В них хранятся значения пикселов.
while {[gets $fId line] > 0} {
        set pixels [concat $pixels $line]
}

#Удаляем из списка pixels последний элемент "};", который формируется программой
```

```
Image2Bitmap
set pixels [lrange $pixels 0 end-1]

#номер пиксела в списке pixels
set pix_num 0;

#Преобразуем последовательный список пикселов в изображение
for {set y 0} { $y < $h } {incr y} {

	#Координата слайса по вертикали (номер строки)
	set Y [expr {$start_y - $y}]

	# Обработка пикселов строки
		for {set x 0} { $x < $w } {incr x } {

			set pix_val [lindex $pixels $pix_num];

			incr pix_num

			#значенеи пиксела в binary формате
			set pix_bin [hex2bin $pix_val] ;

			#Берем двоичное значение красного цвета
			set R_bin [string range $pix_bin 0 4]
			#Преобразуем его в десятичное
			set R_dec [ bin2dec $R_bin ]
			#Корреектируем оттенок красного в соответствии с таблицей t5
			set R [lindex $t5 $R_dec]

			#Повторяем процедуру для зеленой и синей компоненты
			set G_bin [string range $pix_bin 5 10]
			set G [lindex $t6 [ bin2dec $G_bin ]]

			set B_bin [string range $pix_bin 11 15]
			set B [lindex $t5 [ bin2dec $B_bin ] ]
```

```
                    #Координата слайса по горизонтали
                    set X [expr {$start_x + $x}]

                    #Убеждаемся, что секция для окрашивания существует
                    set cond [get_sites "SLICE_X${X}Y${Y}"]
                    if {$cond ne ""} {
                        #Окрашиваем секцию
                        highlight_objects [get_sites "SLICE_X${X}Y${Y}"]  -rgb "$R $G $B"
                    }
                    puts "X = $X   Y = $Y; $R $G $B"
            }
}
close $fId
puts "Complete::Check FPGA field::Window->Device"
```