

ACME User Manual

This manual describes the usage of ACME as well as the modifications in the source files that have to be done to support additional FPGA parts.

PART 1: ACME usage

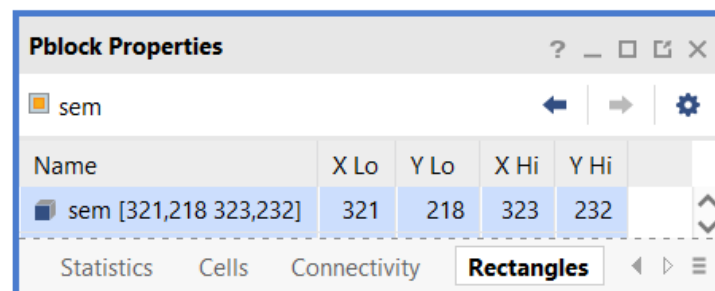
The precompiled executable file is located in /x64/Release/Acme.exe

Using the ACME tool is as simple as generating the EBD file with Vivado, placing it in the same Acme.exe folder, and executing the tool. However, to clarify some points, a step-by-step guide is detailed below:

1. Create your design in Xilinx Vivado hierarchically to identify the module or modules to be tested in an easier way.
2. Place the module to be tested (e.g. DUT) in a pBlock by using the Xilinx constraints file (XDC file). Example of constraint lines:

```
create_pblock MY_PBLOCK
resize_pblock -pblock MY_PBLOCK -add SLICE_X0Y96:SLICE_X7Y99
add_cells_to_pblock -pblock MY_PBLOCK -cells [get_cells DUT]
```

3. Run the implementation step in Vivado and then open the implemented design.
4. Left click in the pBlock and go to pBlock Properties > Rectangles. Write down the physical coordinates (X Lo, Y Lo), (X Hi, Y Hi) of the pBlock. These physical coordinates will be requested by ACME to generate the injection addresses.



NOTE: the physical (grid-based) coordinates **are not** the slice coordinates used in the constraints file.


5. Write the following line in the constraints file to generate the EBD file together with the bitstream:

```
set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_design]
```

6. Generate the bitstream and go to X:\project_name\project_name.runs\impl_1 to find the EBD file. Copy-paste it in the Acme.exe folder.
7. Execute Acme.exe, select the FPGA device by introducing a number.
8. Introduce the physical coordinates from step 4 and a text file with the injection addresses for the SEM IP will be generated in the .exe folder.
9. Use the text file in your research projects to perform fault injections in the configuration memory essential bits of the module inside the pBlock.

PART 2: Extending ACME to other FPGA devices

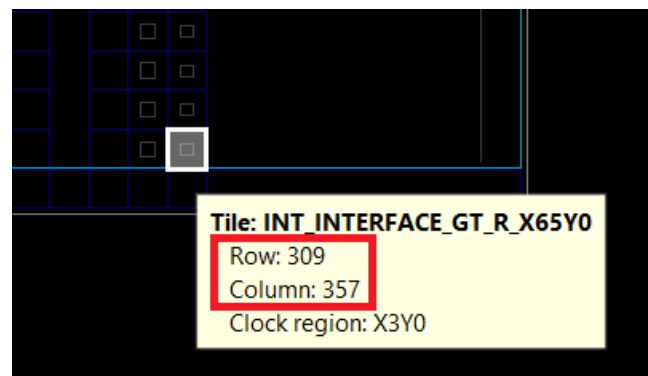
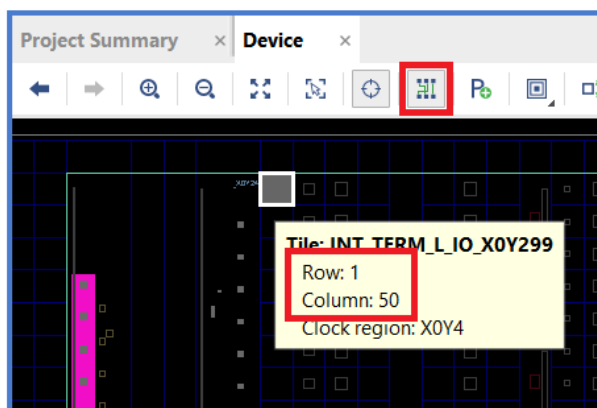
By default, ACME only supports the Kintex-7 KCU105 UltraScale device. However, the tool can be upgraded to support more devices. The steps to characterize the KCU105 and the modifications to be done in ACME are shown below for reference.

1. Implement a simple design in Vivado (e.g. a logic gate), open the implemented design, and toggle to the *Routing Resources* view by clicking at 
2. In parallel, open the Acme project with MS Visual Studio. We have to edit the following files: mainHeader.h, Acme.cpp, welcome.cpp, adjustCoordinates.cpp, KCU105.cpp, ebdTranslate.cpp

mainHeader.h

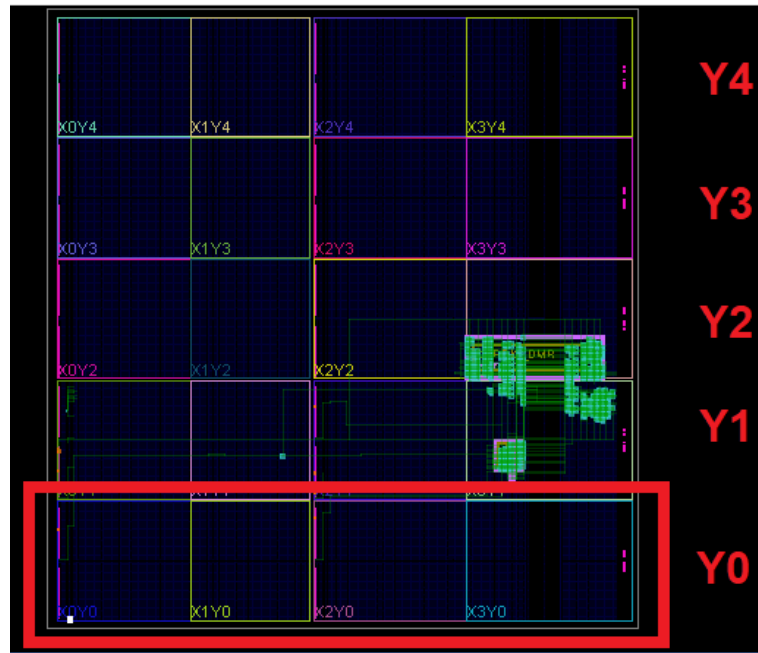
- a. Using Vivado, write down the minimum and maximum X-Y coordinate values of a CLB tile in the device. The minimum is located at top-left corner, the maximum is at bottom-right corner. These values are related to:

```
#define KCU105_MIN_X 50          // KCU105 Minimum X Coordinate
#define KCU105_MAX_X 357        // KCU105 Maximum X Coordinate
#define KCU105_MIN_Y 1          // KCU105 Minimum Y Coordinate
#define KCU105_MAX_Y 309        // KCU105 Maximum Y Coordinate
```



- b. Using Vivado, write down the maximum row value for the different clock regions of the device. These values are related to:

```
#define KCU105_Y4 61             // KCU105 Maximum Row of Clock Region Y4
#define KCU105_Y3 123           // KCU105 Maximum Row of Clock Region Y3
#define KCU105_Y2 185           // KCU105 Maximum Row of Clock Region Y2
#define KCU105_Y1 247           // KCU105 Maximum Row of Clock Region Y1
```



NOTE: Y0 value is not required since it is already defined by the KCU105_MAX_Y #define.

- c. Calculate the number of frames per row of clock regions (F_y) using the following equation, where dummy lines are 141 for the KCU105, the number of words per frame (W_f) is 123, and the number of clock regions in the vertical axis (C_y) is 5. For other devices, this information can be found in the Xilinx documentation [1-3].

$$F_y = \frac{F_t}{C_y} = \frac{\text{Total EBD lines} - \text{Dummy lines}}{W_f \cdot C_y}$$

- d. Calculate the number of frames per CLB tile coordinate (F_x) using the following equation, where C_x is the total number of CLB tile columns in the X axis. This value has to be obtained by manually counting these tiles. BRAM, DSP, IOB and other “on-chip” (not reconfigurable) tiles **must be omitted** from this count. This value is 238 for the KCU105.

$$F_x = \frac{F_y}{C_x} = \frac{\text{Total EBD lines} - \text{Dummy lines}}{W_f \cdot C_x \cdot C_y}$$

NOTE: Resulting F_x and F_y values must be integer

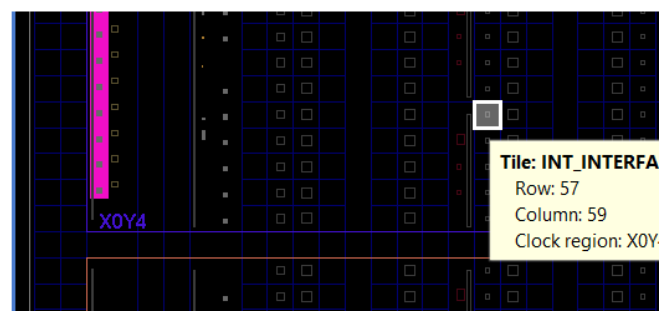
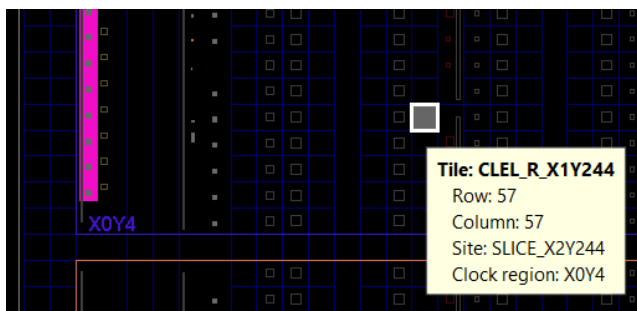
- e. Include a new function prototype for your device similar to the “KCU105” function.

welcome.cpp

- Include an option for the new device in the screen layout and assign it to a number.
- Include a condition in the if-else at line 76 with the min-max X-Y defines of your device.
- Include a condition in the if-else at line 164 with the device name for the summary.

adjustCoordinates.cpp

- Include a condition in the if-else at line 36 for the new device.
- Do a similar for loop with the X values of your device. You have to start from the first CLB column at the left and move to the right. Every time you encounter a DSP, BRAM, IOB or other non-reconfigurable column you have to subtract 1 (or the correct amount of non-reconfigurable columns) from the X coordinate value.



Example: There is a BRAM column at column 58, for this reason, X coordinate values between 59 and 65 must be adjusted by subtracting 1 to its value. If the coordinate is greater than 66 then we must subtract 2 (the previous BRAM column and a new DSP column), and so on. This is because BRAM, DPS, etc. information is not collected in the EBD files and should be omitted.

KCU105.cpp

- Create a new cpp function named as your device (and as the function prototype you wrote in mainHeader.h)
- Use the content of KCU105.cpp as a reference for the function of your device.
- This function determines which and how many clock regions in the vertical axis your design is using. It uses the Y coordinates of the rectangle introduced as input by the user to do that. Each if-else condition refers to a different possibility. The comment at the top of each if-else condition explains each particular case.
- You have to create all the if-else conditions for your device.

Acme.cpp

- a. Include a condition in the if-else at line 62 calling the cpp function you created before using KCU105.cpp as a reference code.

ebdTranslate.cpp

- a. Include or modify the condition in the if-else at line 49 to include your device if necessary.
3. After modifying the previous files, compile the project in MS Visual Studio to generate a new executable file.

Final notes: *If the device being included is not from the 7-series nor the UltraScale family, the following values in mainHeader.h have to be determined by searching in the Xilinx EBD file documentation:*

```
#define BITS_IN_LINE 32           // Number of Bits in an EBD Line
#define WF_ULTRASCALE 123        // Words per Frame (UltraScale Family)
#define DUMMY_ULTRASCALE 141    // UltraScale EBD Header + Dummy Lines
```

Additionally, ebdTranslate.cpp would have to be modified as well with new equations for the linear address, word, and bit values.

REFERENCES

- [1] <https://www.xilinx.com/support/answers/67337.html> 7 Series - SEM IP - How to use the SEM IP error report to look up bit error locations using essential bit data in an EBD file?
- [2] <https://www.xilinx.com/support/answers/67086.html> UltraScale - SEM IP - How to use the SEM IP error report to look up bit error locations using essential bit data in the EBD file
- [3] <https://www.xilinx.com/support/answers/70684.html> UltraScale+ - SEM IP - How to use the SEM IP error report to look up essential bit error locations using the EBD (Essential Bit Data) file?