1. **Introduction**

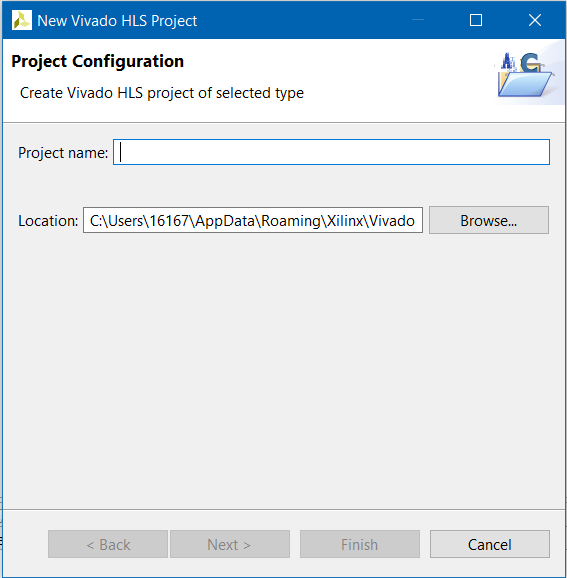
In number theory and computer science, the partition problem, or number partitioning, is the task of deciding whether a given multiset ‘S’ of positive integers can be partitioned into two subsets S1 and S2 such that the sum of the numbers in S1 equals the sum of the numbers in S2. This is one of the very time-consuming problem in the realm of computer science. As the set gets larger the execution time gets extremely longer. I tried to write my code to solve this problem in Python and it took me 15 minutes to solve the array set of size 25. So, how can we make it faster? Well, there are couple of ways to make it faster. We can certainly optimize the algorithm, use multicore CPUs, use GPUs or implement this in FPGA. Thus, in this tutorial our main focus is to implement this solution in FPGA.

***Note: To make this tutorial short, the detail algorithm of the partition problem is not described in this tutorial. Video tutorial for this project is found*** [***here***](https://youtu.be/0_A0uHv1igU)***.***

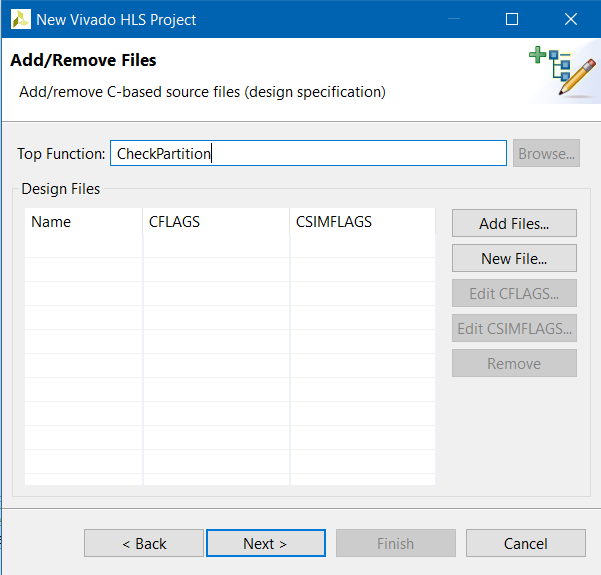
1. **Creating an overlay**
2. **Creating IP using Vivado HLS:**

Creating an IP using High Level Synthesis (HLS) is the very first step required to create a custom overlay. For the HLS of this project, Xilinx’s Vivado HLS is used. Different pragmas were inserted in a C program to boost the efficiency. To create an IP, follow the steps below:

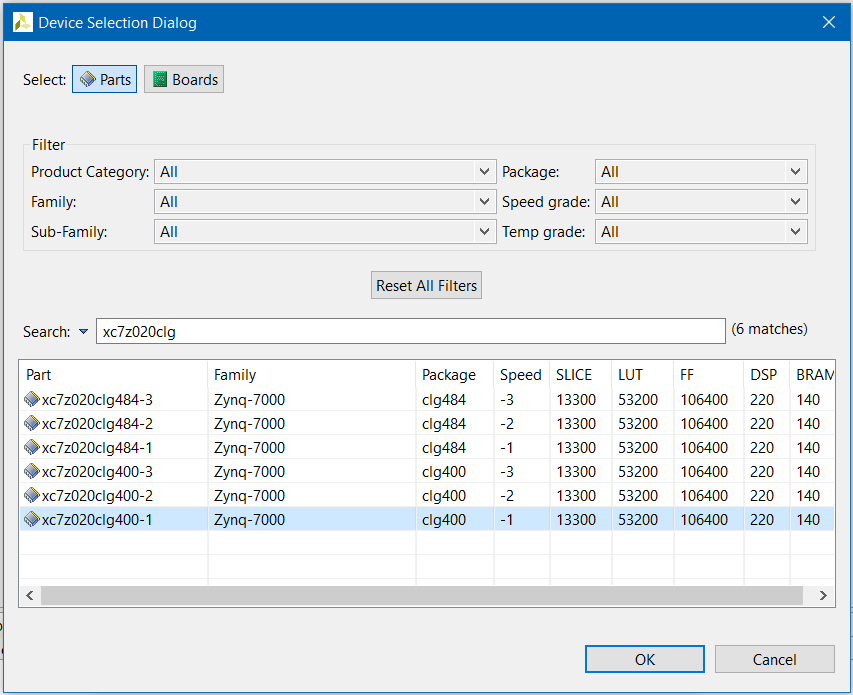
1. Install the Vivado package.
2. Open the Vivado HLS.
3. Create new project > Enter the Project Name



1. Then enter the Top Function name “**CheckPartition**” > Click on **New File**.



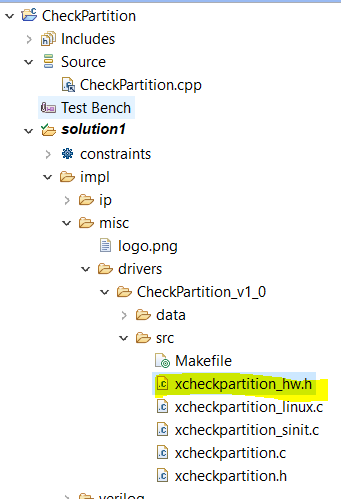
1. Enter the name of the file as “**CheckPartition.cpp**” and press Save button.
2. Click on Next > Next >Change the solution name to “**CheckPartition**”.
3. Select the part number for PYNQ board i.e. (**xc7z020clg400-1**).



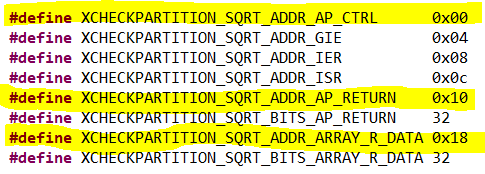
1. Click on Finish.

Now there should be a solution explorer on the left side pane. Inside the “**Source**” folder, there should be “CheckPartition.cpp” file which is our main HLS file where we write our main program.

1. Copy the code from **Appendix 1** and paste it inside the “**CheckPartition.cpp**” file. The code is also found [here](https://github.com/pratik-stha/PartitionProblemUsingFPGA/blob/master/PartitionHLS.cpp).
2. Press the button  to synthesize the program (small green play button).
3. After the synthesis has been successfully done, press the button  to export the RTL and press OK.
4. If everything has been done correctly then the process should generate correct result without any error message.
5. At last, we need to make a note of the memory addresses. It can be found in the folder location shown in the figure below. Double-Click on the “**xcheckpartition\_hw.h**” file.



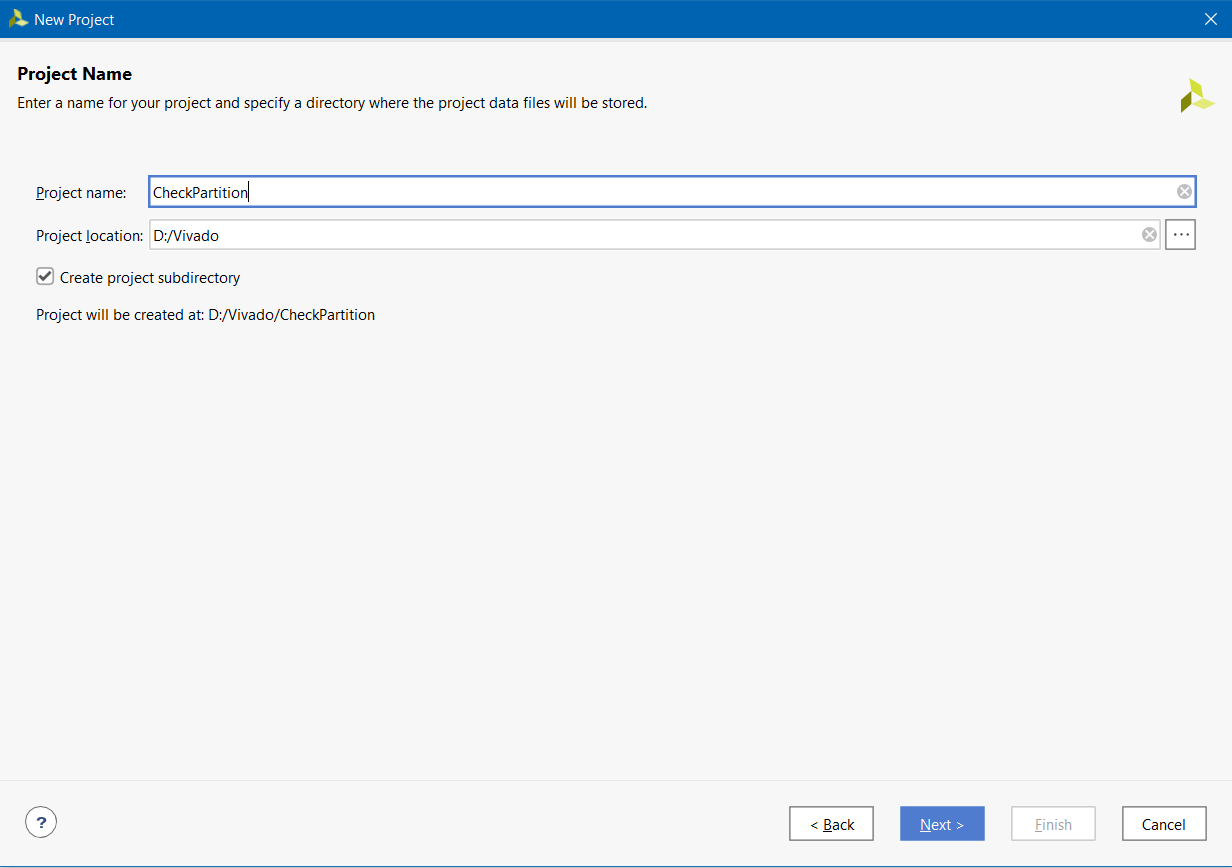
1. Address similar to following image should be seen. Note that these addresses may vary. According to the addresses here, 0x00 represents control address,0x18 represents port address for array input and 0x10 represents port address for the output.



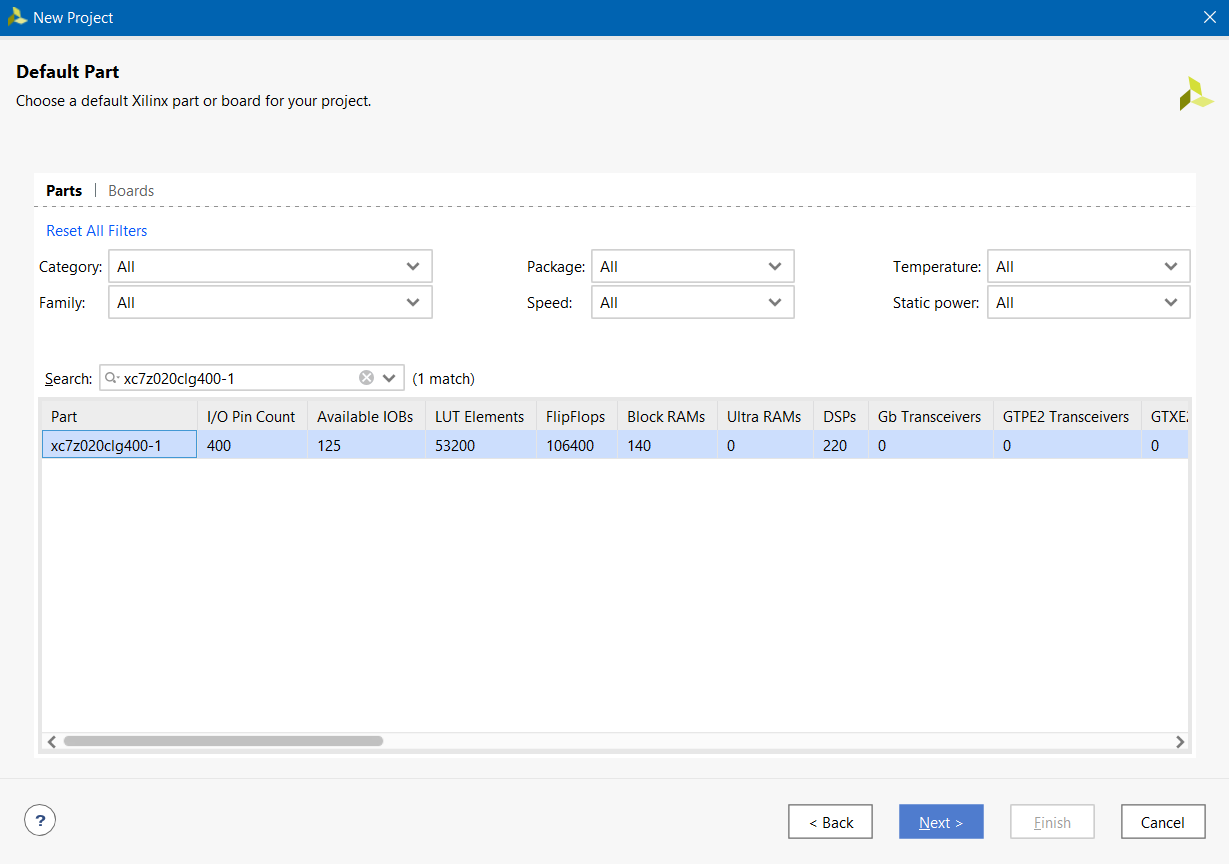
1. **Importing the IP and generating the overlay file:**

Now, the programming portion is done. Lets, create the overlay design using this IP we just created using following steps.

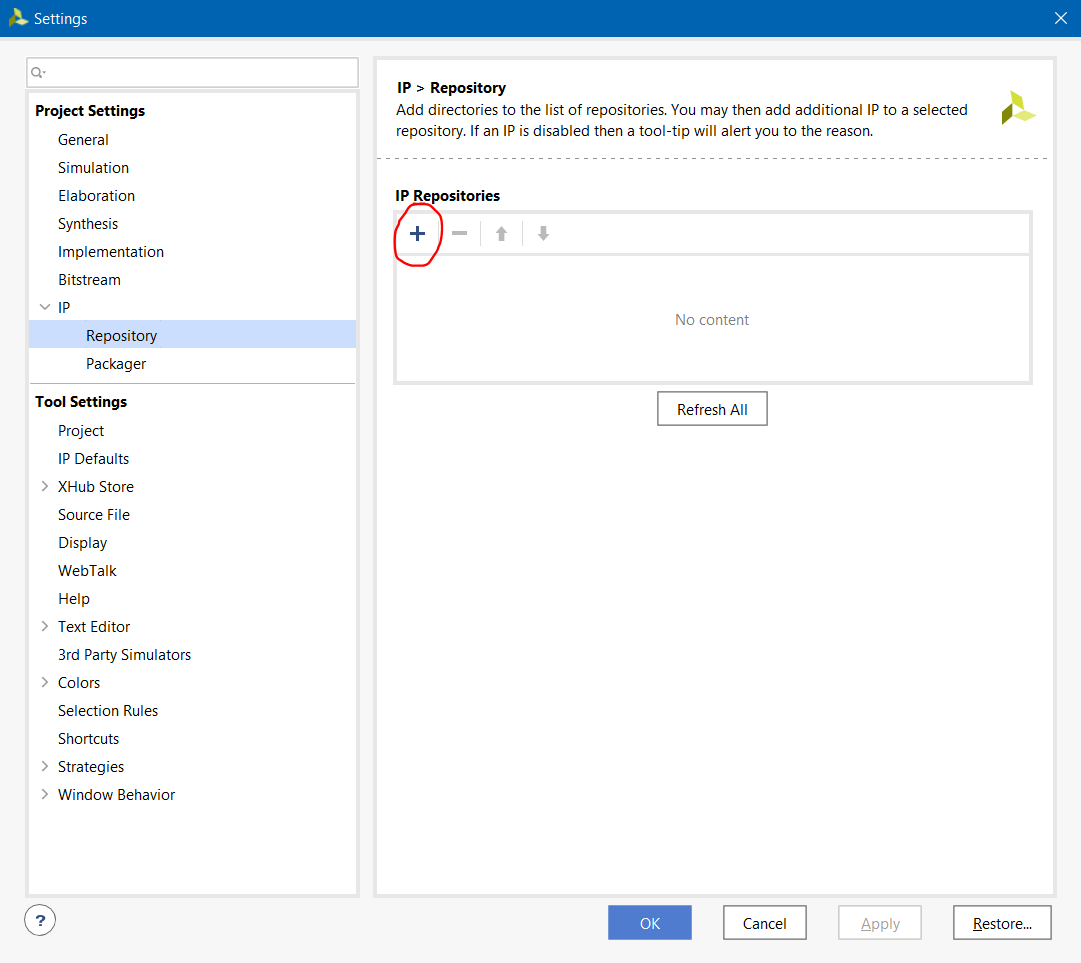
1. Open the **Vivado** **Design** **Suite**.
2. Enter the project name as “**CheckPartition**”. Make sure the “**Create Project Subdirectory**” is checked. The click Next.



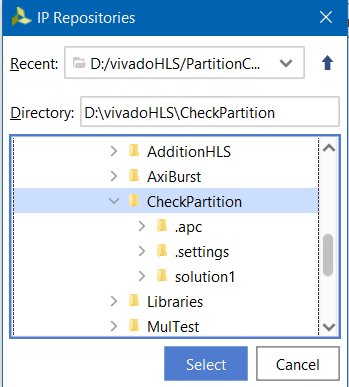
1. Click Next. Then select the part name (**xc7z020clg400-1**) as shown in figure below. Then click **Next**.



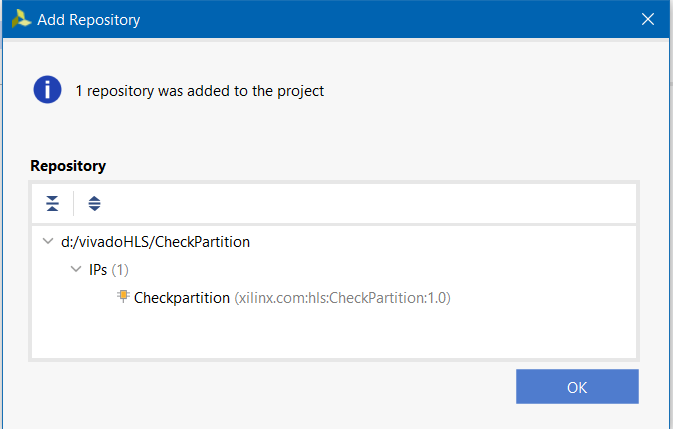
1. Then click Finish.
2. There should be Project Manager on the left pane. Inside the **Project Manager**, Click on **Settings** > Click on **IP** > Select **Repository**.



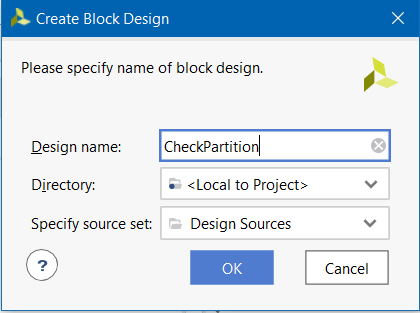
1. Click on the **‘+**’ sign shown in the figure above. Then locate to the directory where the HLS project was created. Then select the same project “**CheckPartition**”. This is our IP.



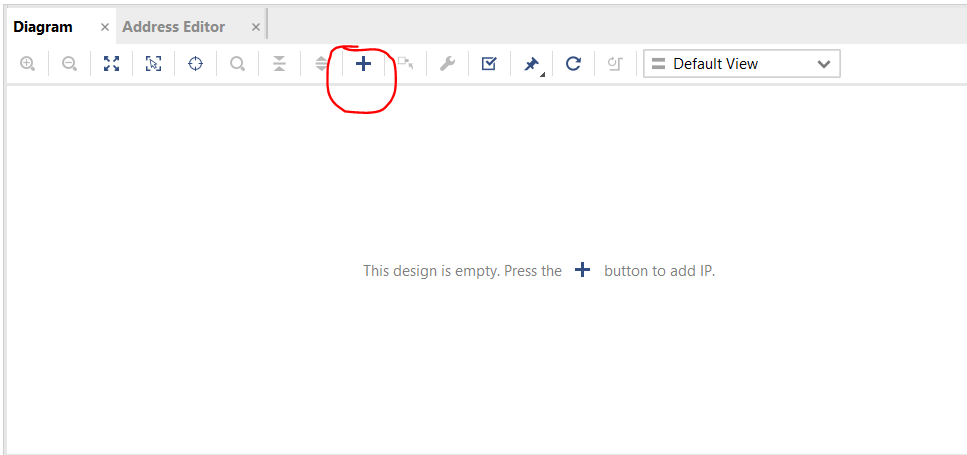
1. After selecting, click on “CheckPartition” inside the IP then press **OK**.



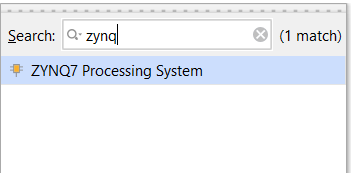
1. Back to the **Flow Navigator** on left pane of Vivado Design Suite. Click on “**Create Block Design**”. Name the design as shown in figure below. Click OK.



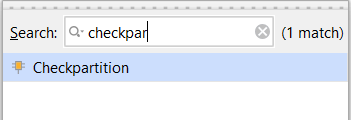
1. Now click on the ‘**+**’ sign in the Diagram tab as shown below.



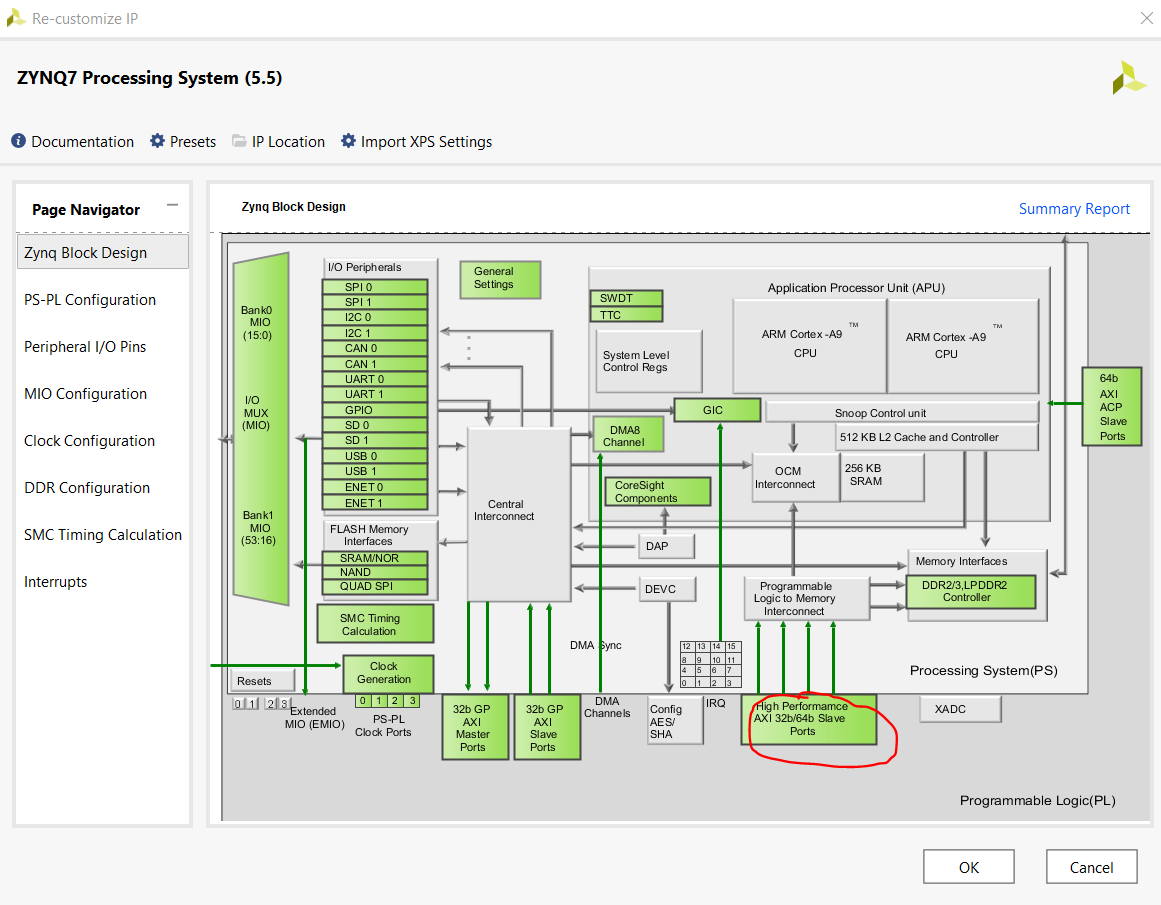
1. Select the “**Zynq Processing system**” and press Enter.



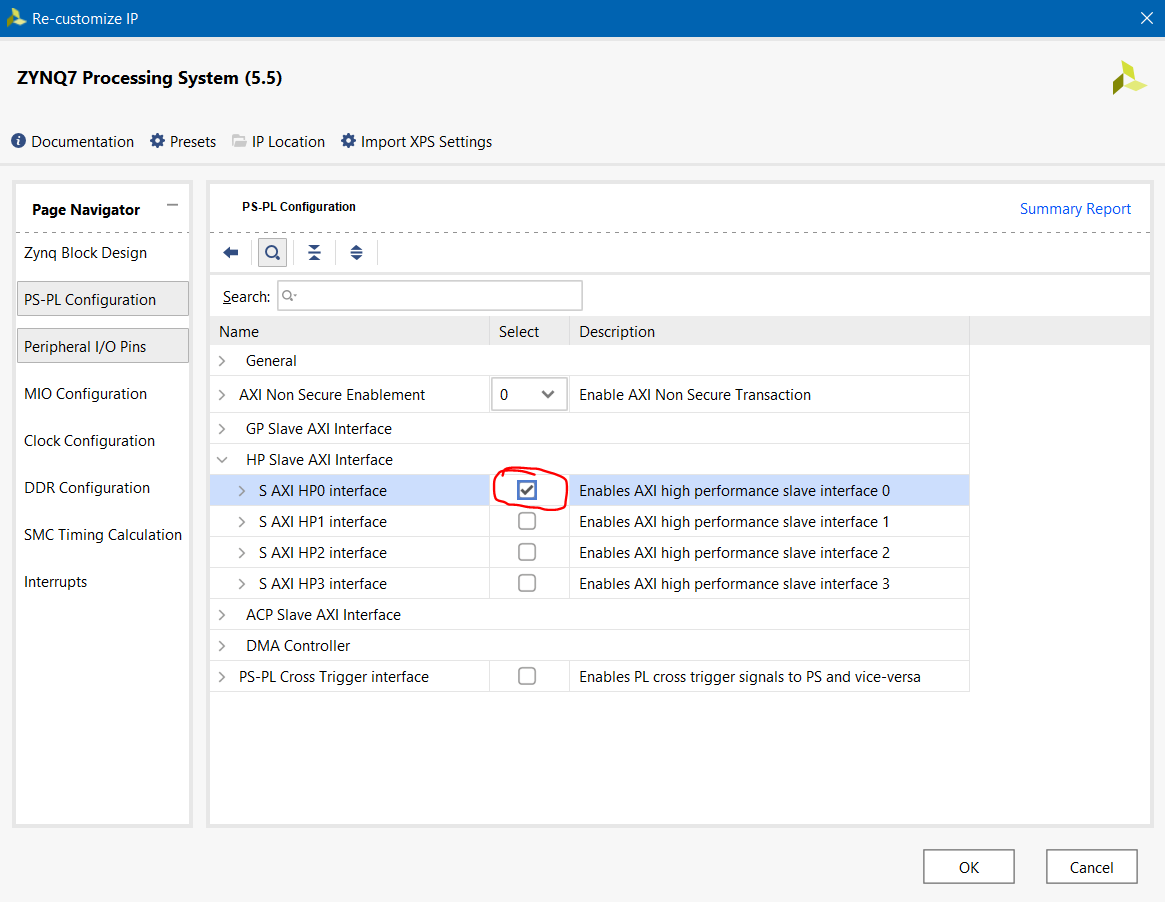
1. Again, repeat the same process and import the IP just created in previous section.



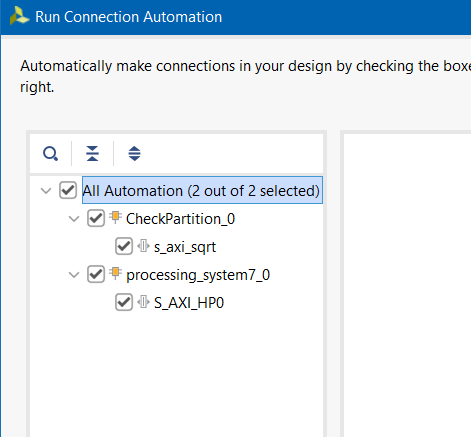
1. As two modules have been imported in the design block now, double click on the **Zynq Processing System** block. Then Double click on “**High performance AXI 32b/64b slave ports**” as shown in figure below.



1. Under **PS-PL configuration** section, check the “**S\_AXI HP0 Interface**” as shown in figure.

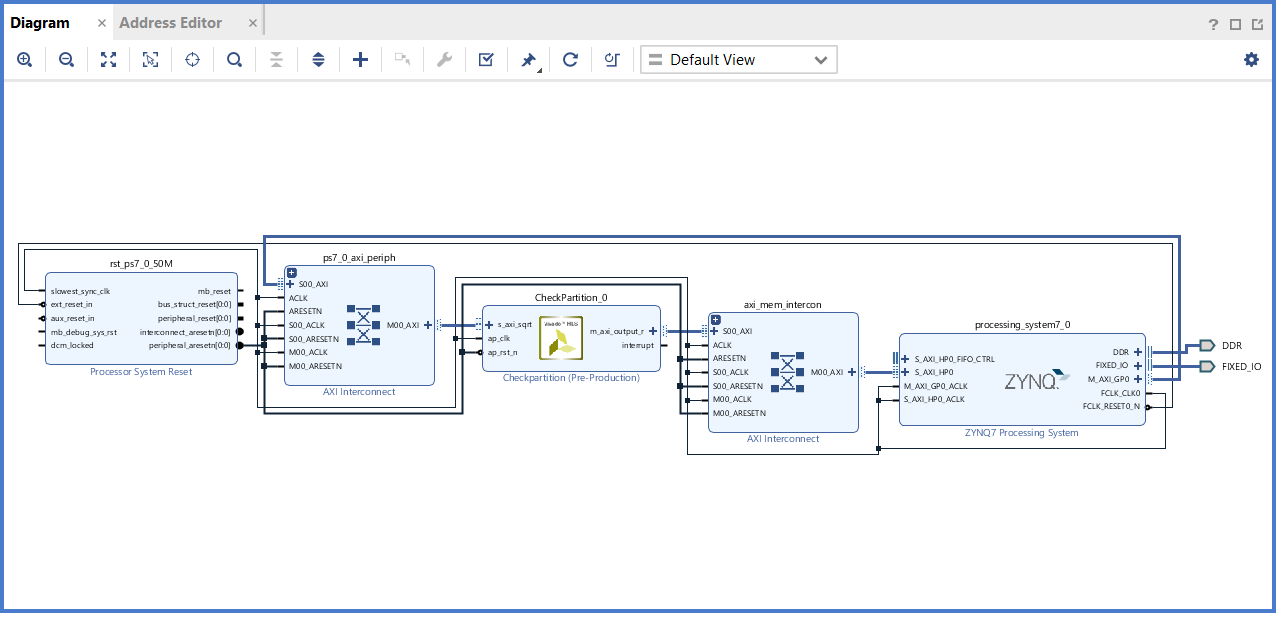


1. Click on **Run Connection Automation** and make sure to select everything on left pane.

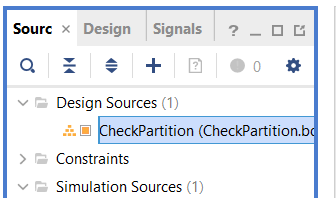


xv. Then click on **Run Block Automation.**

1. The block design should look something like shown below.



xvii. Now go to the source tab and Right click on the “**CheckPartition**” and click on “**Create HDL wrapper**”. Then click on OK. It may take little time.



xviii. Now its time to generate the bit file. Click on “**Generate Bitstream**” located on the bottom left pane of Flow Navigator. This may take little more time depending on the capacity of the machine.

xix. If everything has been done correctly then there should not be any error message. Click on OK when the message for successful completion shows up.

Congratulations !! we have successfully created our own overlay.

1. **Importing the overlay into Python**

If we have made it this far, it means we have successfully created our custom overlay that takes an array as input and produces the output 1 or 0. Figure below shows the black box diagram of the overlay.

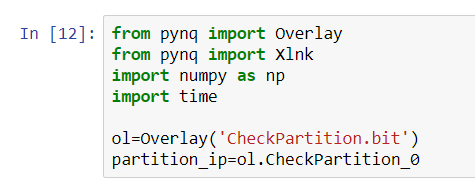


Now, we are done with Vivado. For rest of the process, we will be working on Python and PYNQ board. Power up the PYNQ board with USB and connect it with the machine through ethernet. Boot up the PYNQ board. When the LEDS flash blue, it indicates that the board has been successfully booted and is ready to work. Refer to [the official tutorial](https://pynq.readthedocs.io/en/v2.0/getting_started.html) if there is any problem connecting the board.

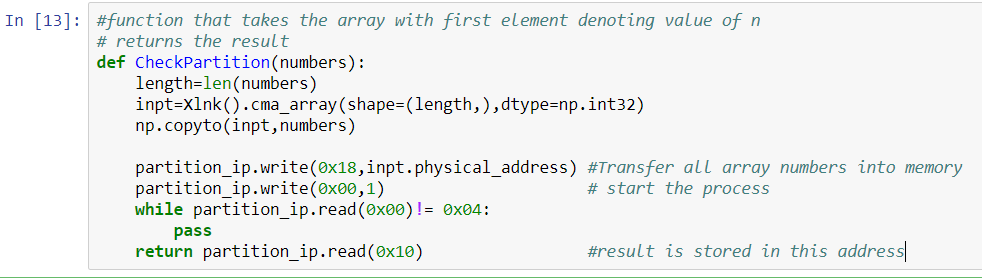
We have our overlay design ready. We are interested in two files. One is \*.BIT file and other is \*.HWH file. Navigate to the project directory>CheckPartition>CheckPartition.runs>impl\_1. Copy the **CheckPartition\_wrapper.bit** file and store it somewhere safe. Then rename it as “**CheckPartition.bit**”. Similarly, navigate to the project directory > CheckPartitions>Sources\_1>bd>CheckPartition>hw\_handoff. Copy the “**CheckPartition.hwh**” file and store it somewhere safe. Now navigate to [\\pynq\xilinx\jupyter\_notebooks](file:///\\pynq\xilinx\jupyter_notebooks) and copy both of these files and paste it here. Please make sure you can successfully access the PYNQ board first.

Follow the steps below to import the overlay and use it.

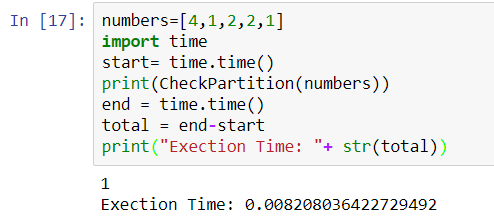
1. Open the Jupyter Notebook through PYNQ remote access using the IP address.
2. Create a python 3 file (click on **New** > **Python 3**) at same location where we have copied the two files.
3. Write down the following codes as shown in image below then Run it.



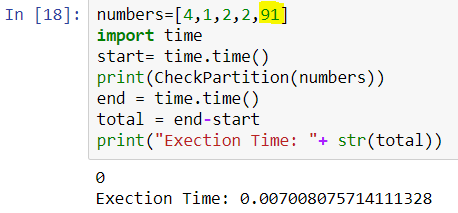
1. After successful execution, create one more cell underneath this cell and write down the following code. Then Run it.



1. Create one more cell underneath this cell and write down the following code which calls the previous function and sends the parameter. We are now testing if our overlay actually worked or not.



1. Here, ‘**numbers**’ is the array of which the first element ‘**4**’ denotes there are altogether 4 numbers that needs to be partitioned. All the numbers are the first element are the actual numbers that needs to be partitioned. After execution of this cell, the function should return the value of 1 indicating that the given set can be partitioned with the total execution time. If it returns 0 then it means the given set cannot be partitioned.
2. Go ahead and try to change the number as shown in figure and see what happens.



Congratulations !! we have successfully created our own overlay that takes the array as input and produces the single output indicating if the given set can be partitioned or not.

**Appendix 1:**

#include <string.h>

#include <math.h>

int PartitionNum(unsigned int,unsigned int, int\* );

int CheckPartition(int\* array) {

#pragma HLS INTERFACE s\_axilite port=return bundle=sqrt

#pragma HLS INTERFACE m\_axi depth=50 port=array offset=slave bundle=output

#pragma HLS INTERFACE s\_axilite port=array

int buff[100];

unsigned int nPartitions = 1;

unsigned int n= array[0];

memcpy(buff,(const float\*)array,(n+1)\*sizeof(int));

for(int i = 0;i < n;i++) {

#pragma HLS pipeline II=1

nPartitions = nPartitions \* 2;

}

nPartitions = nPartitions / 2;

int solutionFound = 0;

int solution = -1;

for(unsigned int i = 1;i < (nPartitions);i++) {

#pragma HLS pipeline II=1

if (PartitionNum(i,n,buff) == 1) {

solutionFound = 1;

solution = i;

break;

}

}

if (solutionFound) {

return 1;

}

else {

return 0;

}

return 0;

}

int PartitionNum( unsigned int value,unsigned int n, int \*array) {

int sum0s = 0;

int sum1s = 0;

unsigned int mask = 1;

for(int i = 1;i <= n;i++) {

#pragma HLS pipeline II=1

if ((mask & value) != 0) {

sum1s = sum1s + array[i];

}

else {

sum0s = sum0s + array[i];

}

mask = mask \* 2;

}

if (sum0s == sum1s)

return 1;

else

return 0;

}