# SPMV ON FPGA

FPGA101 PROJECT

ANDREA OGGIONI

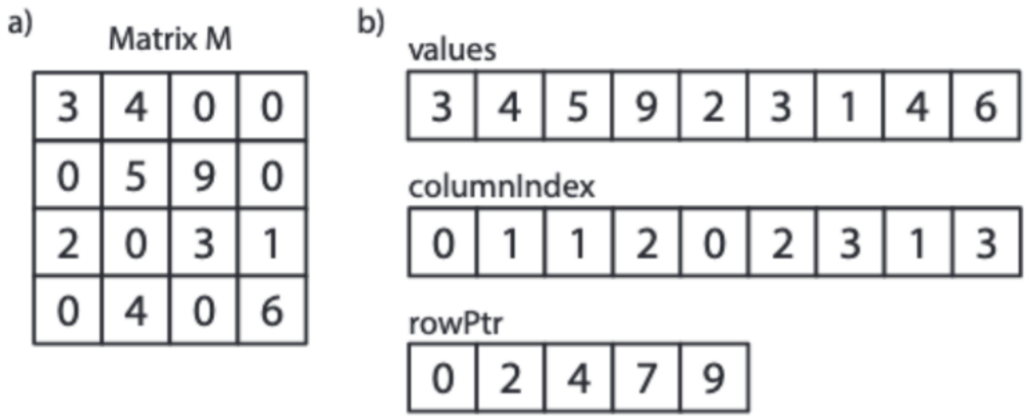
## WHAT IS ~~AN FPGA~~ SPMV

- Multiplication kernel between a sparse matrix and a vector.
- Only store and move the non-zero elements of the matrix in memory (CRS).
- Faster than traditional matrix-vector multiplication.

## HOW DOES IT WORK

- Target: $y = Ax$ where $A$ is $n \times m$ stored in CRS:
  - $a$: the vector of non-zero values in the matrix;
  - $c$: the vector of the column indexes;
  - $r$: the vector of the first column index of each row.

$$y_i = \sum_{k=r_i}^{r_{i+1}} a_k \cdot x_{c_k} \qquad \forall i \in [0, \dots, n-1]$$



a) Matrix M

| | | | |
|---|---|---|---|
| 3 | 4 | 0 | 0 |
| 0 | 5 | 9 | 0 |
| 2 | 0 | 3 | 1 |
| 0 | 4 | 0 | 6 |

b)

values

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 9 | 2 | 3 | 1 | 4 | 6 |

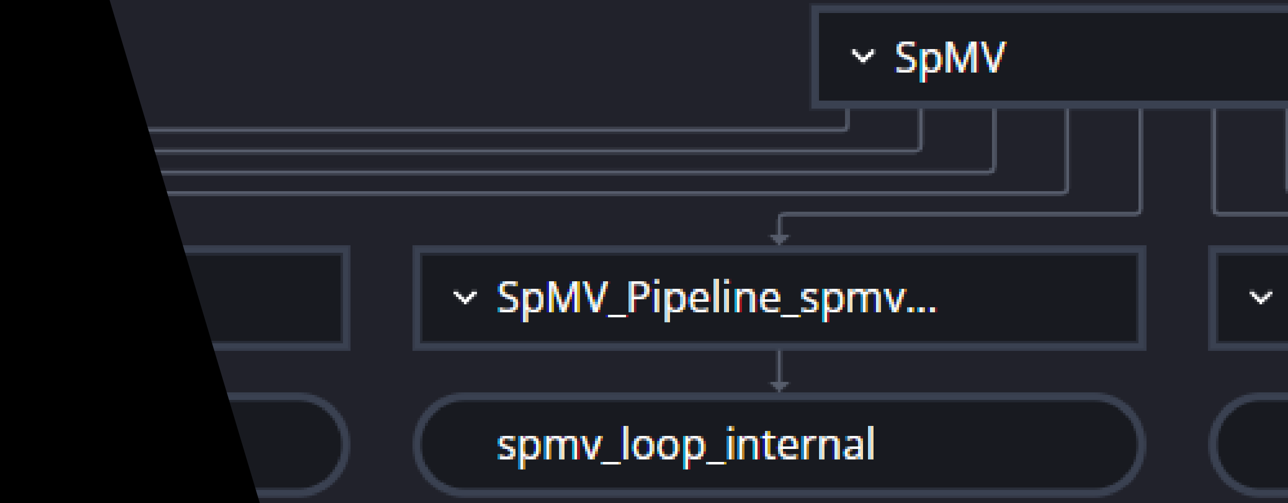
columnIndex

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 0 | 2 | 3 | 1 | 3 |

rowPtr

| | | | | |
|---|---|---|---|---|
| 0 | 2 | 4 | 7 | 9 |

# THE ALGORITHM



## PSEUDOCODE

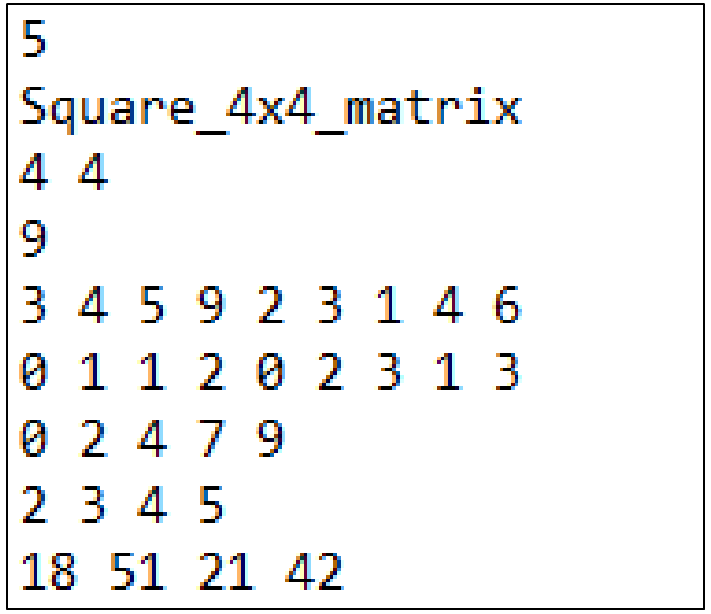
```
for i = 0 to n - 1 do
        y[i] = 0;
        for k = r[i] to r[i+1] do
                y[i] += a[k] * x[c[i]]
        end for
end for
```

## IMPLEMENTATION

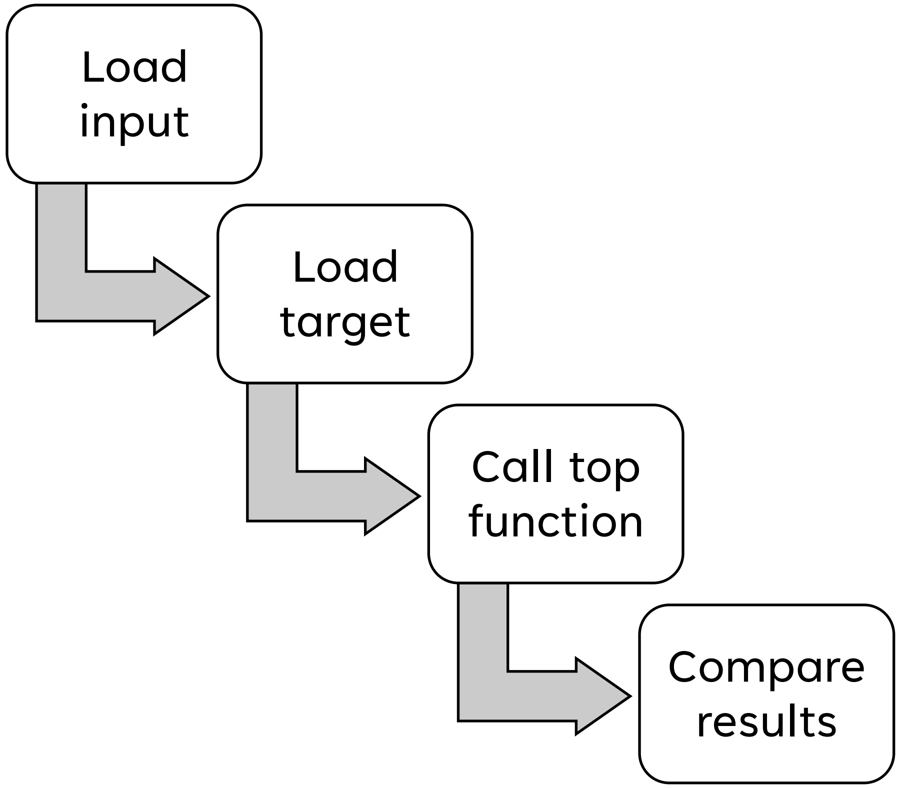
```
spmv_loop_external:for(VectorSize i = 0; i < MAX_MATRIX_SIDE_SIZE; i++) {
  #pragma HLS UNROLL
  int sum = 0;
  if(i < numOfRows) {
    spmv_loop_internal:for(ValuesSize j = rowPointers[i]; j < rowPointers[i + 1]; j++) {
      #pragma HLS PIPELINE II=1
      int matrix_value = values[j];
      ColumnIndex column_index = columnIndexes[j];
      int vector_value = vector[column_index];
      int temp = matrix_value * vector_value;
      sum += temp;
    }
  }
  output[i] = sum;
}
```

## TESTBENCH

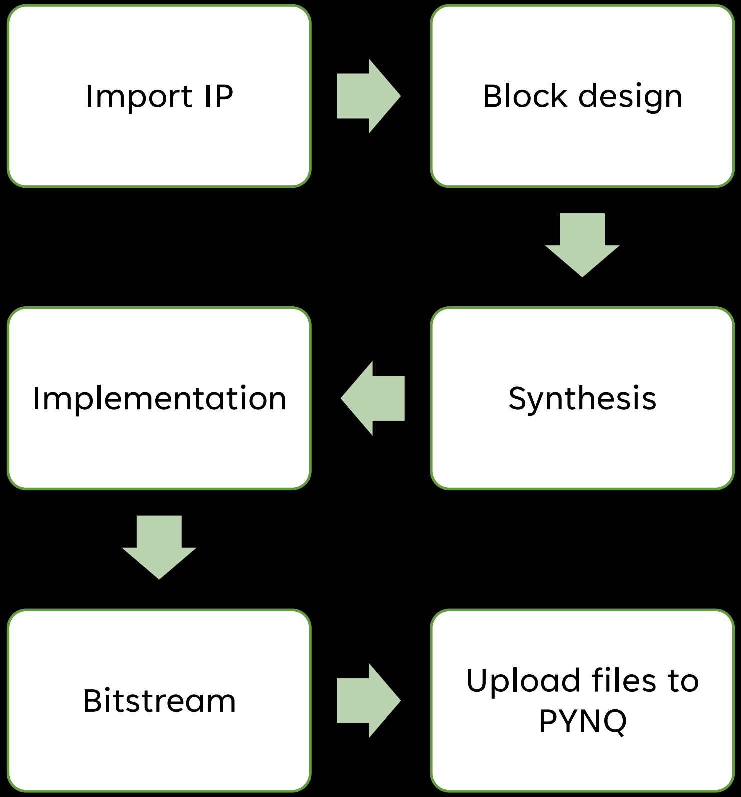
- **Testbench on text file.**
- **Python testcase generator.**

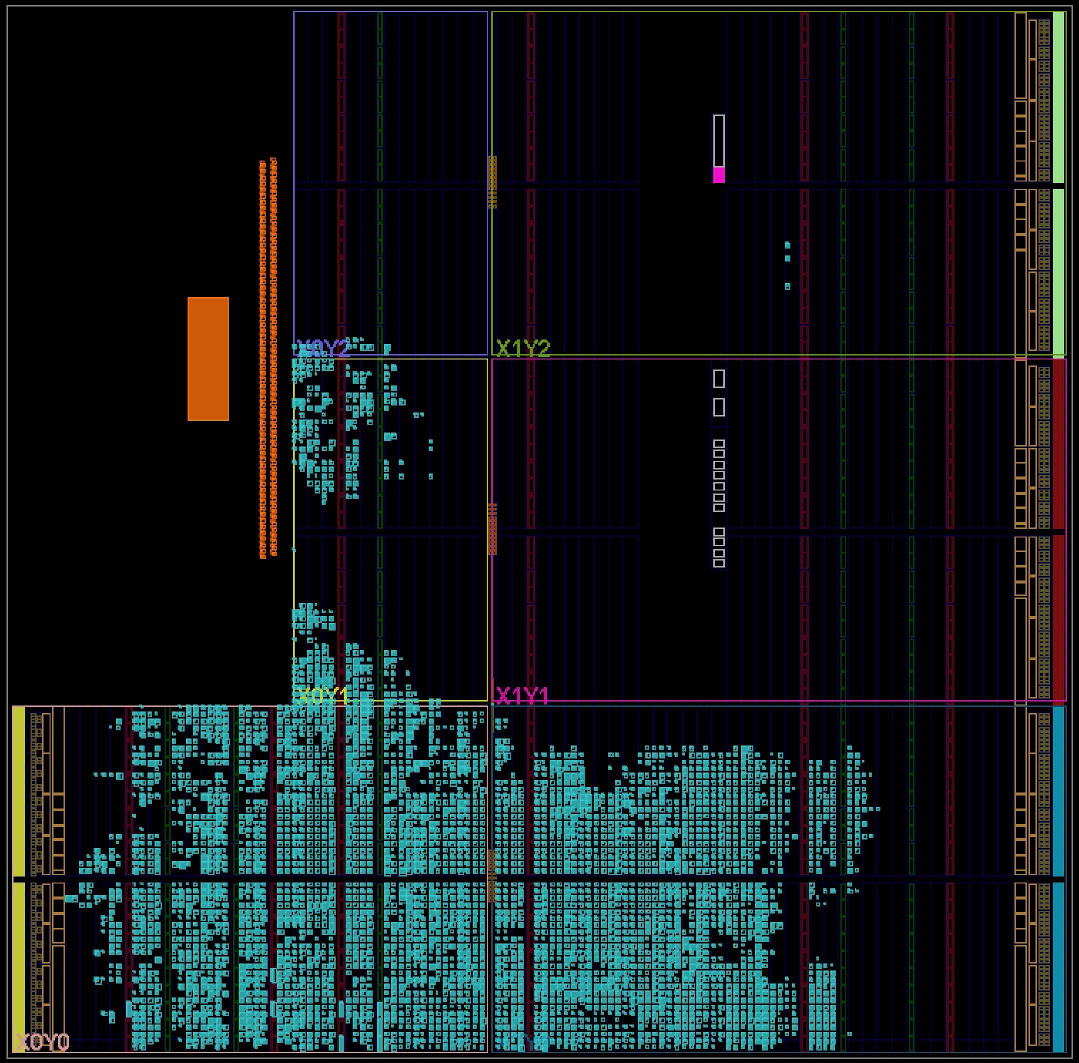


```
5
Square_4x4_matrix
4 4
9
3 4 5 9 2 3 1 4 6
0 1 1 2 0 2 3 1 3
0 2 4 7 9
2 3 4 5
18 51 21 42
```

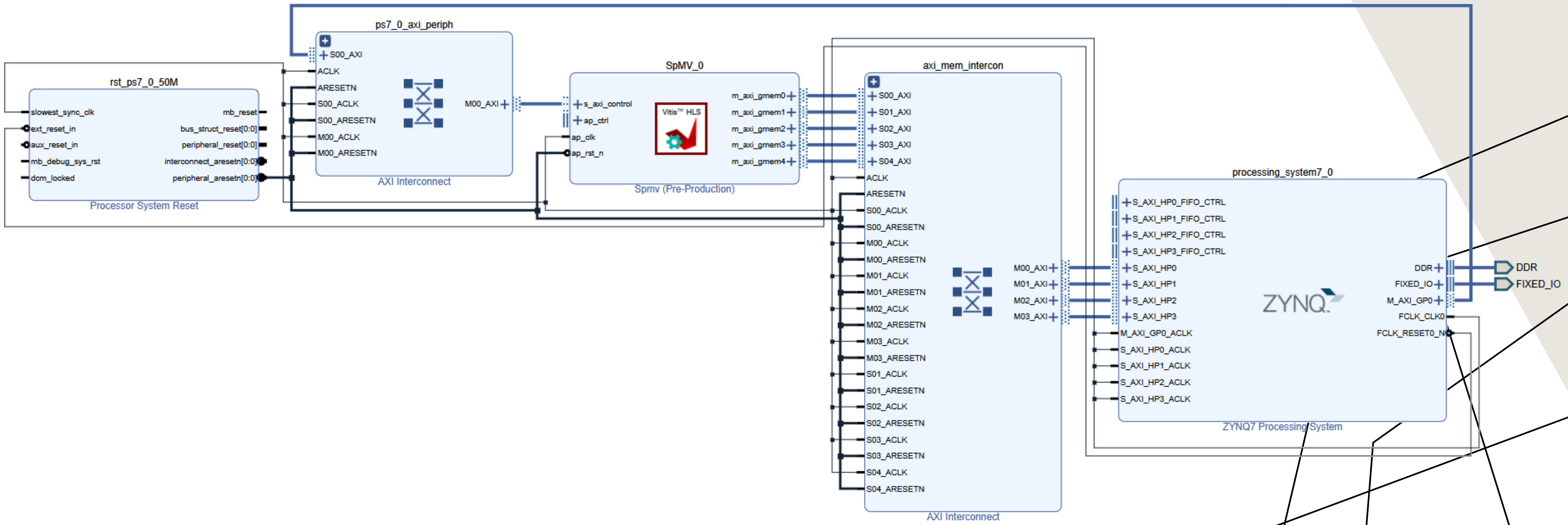


## IT’S BITSTREAM TIME





# BLOCK DESIGN



## USING THE KERNEL

Load overlay → Allocate buffers → Fill buffers → Write parameters → Start kernel → Wait for termination → Read result

jupyter

PYNQ™

## THANKS

Code, slides and report on [GitHub]()