# Brief Description of MATRIX\_MUL\_IP\_CORE

MATRIX\_MUL\_IP\_CORE is an IP core developed in VHDL and targeted for Xilinx FPGA’s Virtex family 5, 6 and 7. The IP core is parameterizable and can perform multiplication of two square Matrices of any size.

Please note that the IP core cannot be used in any other FPGA vendor product apart from Xilinx because at the time of development Xilinx Virtex family was targeted. Perhaps in the feature it may be ported to support other FPGA platforms.

The names of the Matrices to be multiplied by the IP Core are denoted as P Matrix and G Matrix. P matrix is on the right hand side of the multiplication sign (P \* G) while G matrix is on the left hand side of the multiplication sign. P matrix is loaded into the IP Core while G matrix is supplied externally during multiplication.

Before multiplication is performed, P matrix must be loaded into the IP Core one element per clock cycle. After P matrix is loaded multiplication can commence. During multiplication the IP core expects to receive element from G matrix. The IP Core automatically generates the address for the element it requires from G matrix. The IP core can perform the following matrix multiplication below.

1. P Matrix \* G Matrix
2. P Matrix \* G Matrix transposed
3. P Matrix transposed \* G Matrix, and
4. P Matrix transposed \* G Matrix transposed.

After multiplication has completed, the result can be offloaded from the IP core, one element per clock circle in a row wise fashion, and the ordering is from last element to first element of row, this is explained in [here](#_UN_LOAD_(Input)).

# Pinout

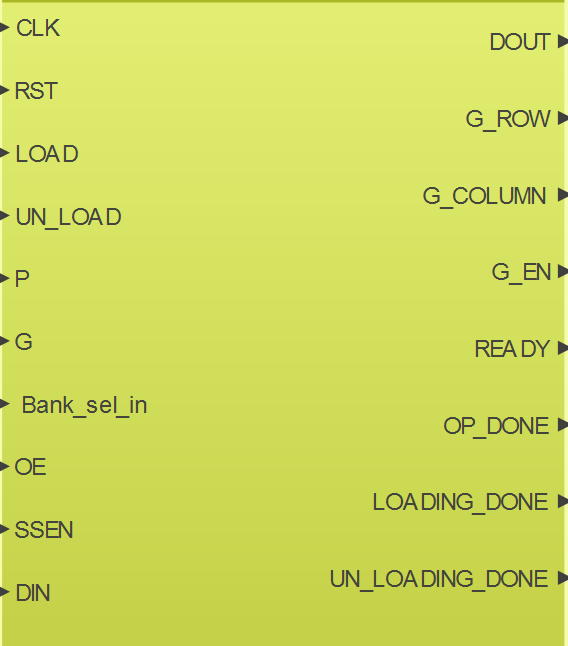


Figure 1: Core Schematic Symbol

|  |  |  |  |
| --- | --- | --- | --- |
| **GENERICS** | | | |
| **Signal** | **type** | **Edit** | **Description** |
| COLUMN\_TOTAL | Integer | Can Edit | This generic Specifies the Dimension of the Matrix. Change the value to match the number of columns and rows your matrix will have. |
| OPCODE\_WIDTH | Integer | Do not Edit | This generic specifies the size of the OPCODE |
| ADDR\_WIDTH | Integer | Do not Edit | This generic specifies the size of the address |
| DATA\_WIDTH | Integer | Do not Edit | This generic specifies the size of the Data bus |
| DATA\_WIDE\_WIDTH | Integer | Do not Edit | This is used internally by the IP core. |
| **PORTS** | | | |
| **Signal** | **Direction** | **Optional** | **Description** |
| CLK | Input |  | Clock signal input - active when rising edge |
| RST | Input |  | Reset (synchronous) - active when high. Asserting RST synchronously with CLK resets only FSM in IP core, all other elements in IP Core remain intact. |
| LOAD | Input |  | LOAD - active when high. Asserting LOAD puts the IP core in the *loading state* and then data (for P Matrix) can be fed into the IP Core for storage in internal Block RAM. |
| UN\_LOAD | Input |  | UN\_LOAD - active when high. Asserting UN\_LOAD puts the IP core in the *unloading state* and the entire matrix data in the specified bank of internal Block RAM of the IP core is offloaded through DOUT |
| P | Input |  | P Matrix - Setting P to high will multiply P matrix transposed otherwise P matrix is multiplied in normal form. |
| G | Input |  | G Matrix - Setting G to high will multiply G matrix transposed otherwise G matrix is multiplied in normal form. |
| Bank\_sel\_in | Input |  | Bank select - When set to high the lower bank is selected, when set to low the upper bank is selected[[1]](#footnote-1) |
| DIN(DATA\_WIDTH-1 : 0) | Input |  | Data input Port - Serves as data input to IP Core. Max and optimal data width is 18 bits. This port is used to load P matrix data into the IP core before multiplication, and it is also used to supply G matrix data during Multiplication. |
| DOUT(DATA\_WIDTH-1 : 0) | Output |  | Data output Port - Serves as data output from IP Core. Max and optimal data width is 18 bits. Data is offloaded from IP Core, one element per clock cycle. |
| G\_ROW(ADDR\_WIDTH-1 : 0) | Output |  | G Matrix memory row address. This is generated automatically during multiplication. This should be used with G matrix memory row input. |
| G\_COLUMN(COLUMN\_TOTAL-1 : 0) | Output |  | G Matrix column address. This is generated automatically during multiplication. This should be used with G matrix memory column input. |
| G\_O\_EN | Output | Yes | G memory output enable. This signal should be used to enable the output of the G Matrix RAM Memory. If G matrix is not in separate RAM this signal can be ignored. |
| READY | Output |  | READY - This signal is set high when the IP core is ready to receive data for loading into the internal Block RAM. DIN should hold the first valid data at the time this signal is set high. It is also set high immediately the first data offloaded from the IP core is present in the output port DOUT. |
| OP\_DONE | Output |  | Operation done (Complete) - This signal is set high when the IP core has successfully multiplied P matrix by G matrix |
| LOADING\_DONE | Output |  | LOADING DONE (Complete) - This signal is set high when the IP core has finished loading the Block RAMs with P matrix data. |
| UN\_LOADING\_DONE | Output |  | UN\_LOADING DONE (Complete) - This signal is set high when the IP core has finished off-loading the Block RAMs. |

Table 1: Brief descriptions of IP Generics, Input and Output ports.

# IP Core Overview

The IP core is composed of 3 key sub components.

1. DSP48 block

2. Block RAM (BRAM)

3. FSM unit.

## DSP48 block:

The Number of DSP48 block available in the IP core is a variable and it is equivalent to the number of Columns of the Matrix (The value specified on the *COLUMN\_TOTAL* generic). In this IP Core the DSP48 blocks are configured mainly for Multiplier and Accumulate operations (MACC). The data width is fixed at 18 bits and this is because for multiplication operation the input data width are as follows: (port A = 25 bits, and Port B = 18 bits). In other to maintain consistency DSP is fixed to operate with 18 bits data width for both inputs.

## Block RAM:

The Number of Block RAMs used in the IP core is also a variable and it is equivalent to the number of Columns in the Matrix (The value specified on the *COLUMN\_TOTAL* generic). Actually the block RAM has a total width of 36 bits so one (1) block RAM can accommodate two (2) Columns.

The block RAM is a dual port Block RAM with separate *Read* and *Write* addresses. The address range of the block RAM is 10 bits (0 to 1023). In the block RAM each memory location is configured as 18 bit word but can accommodate 36 bits maximum. For this IP core 18 bits is the recommended size because of limitation from the DSP48 block.

The block RAM is divided into two (2) banks (upper and lower banks) and the user can specify which bank data should be loaded to, or offloaded from. This is very useful when performing matrix multiplication with result of previous operation.

## FSM Unit:

There is just one FSM unit that controls the entire IP core. The FSM unit is responsible for loading data into the BRAM and offloading data from the BRAM. It also generates, control signals for the DSP48 block and IP core as well as the flags for the IP core. The FSM can be controlled by the user based on the following control input pins.

1. **RST**
2. **LOAD**
3. **UN\_LOAD**
4. **P** and **G**

All input control signals to the IP core are active high. When the **RST** input is asserted only the FSM is placed in the reset state all other components remain intact. Typically before any operation is carried out with the IP core it is recommended to reset the FSM in the IP core using the RST input then before releasing the FSM, the desired state of the control input signals should be configured, after that the FSM can then be released from *reset\_state*. The FSM can be released from *reset\_state* by setting the RST input to low state.

When the **LOAD** input is asserted the FSM goes into the *Loading\_state*. ***Please note\*:*** *Assuming the IP core is in reset\_state already, The LOAD input should be set high before the RST input is set low, otherwise the FSM may not enter the loading\_state*. When the IP core is ready to start receiving data it will set the **READY** output signal to high state, at this point the user can start sending data to the IP core through the DIN input port. When the READY output signal is set, the first valid data should be present in DIN port.

When the UN\_LOAD input is asserted the FSM goes into the *unload\_state.* ***Please note\*****: The LOAD input must be set to low otherwise the FSM will not respond to the UN\_LOAD input. Assuming the IP core is in reset\_state already, make sure LOAD signal is set low then set UN\_LOAD high, after that release IP from reset\_state by setting RST to low.* When valid data from BRAM reaches the DOUT port, the IP core will set the READY output signal to high state, at this point the user can start receiving valid data from the **DOUT** output port. The READY output signal is set in the same clock cycle that valid data reaches **DOUT.**

**P** and **G** input signalsare used to specify what type of matrix multiplication the IP core should perform. The table below summarizes the configuration of PG and the resulting multiplication.

|  |  |  |
| --- | --- | --- |
| **P** | **G** | **Action** |
| 0 | 0 | P Matrix multiplied by G Matrix |
| 0 | 1 | P Matrix multiplied by transposed G Matrix |
| 1 | 0 | Transposed P Matrix Multiplied by G Matrix |
| 1 | 1 | Transposed P Matrix multiplied by Transposed G Matrix |

Table 2: P G configuration

# Detailed Step by step setup and usage of IP Core

## LOAD (Input)

This Pin is active high and when asserted the FSM in the IP Core goes into the loading state to Load P Matrix data into the IP core. As soon as the IP Core is ready to start receiving data the READY flag is set high and at that same clock circle the first data of P Matrix should be present in the DIN input port of the IP core. The order of data loading is row wise, first element of each row to last element of the row, then next row.

Before putting the IP core in the loading state the bank should be specified.

Signals that Influence LOAD input are: RST and Bank\_sel\_in

The following flags serve as feedback: READY, LOADING\_DONE

### Typical sequence of micro operations for LOADING P Matrix data.

**Step 1:**

1. RST => set high (at least 3 clock cycles).
2. LOAD => set high.
3. Bank\_sel\_in: set to proper bank ('0' => Upper bank, '1' => Lower bank).

**Step 2:**

1. RST => Set low.
2. READY => wait until it is high, start sending data immediately.

**Step 3:**

1. LOADING\_DONE: Becomes high when data completely loaded.

## UN\_LOAD (Input)

This pin is active high and when asserted the FSM in the IP Core goes into the unloading state to offload P Matrix data from the IP core. As soon as the first valid data arrives at the output port DOUT the READY flag is set high. The order of data offloading is row wise from last element of each row to first element of the row, and then next row follows with the same pattern. It is important to note that the Matrix is stored internally as a circulate matrix. Please refer to the following figures below for an example of normal matrix and circulate matrix.

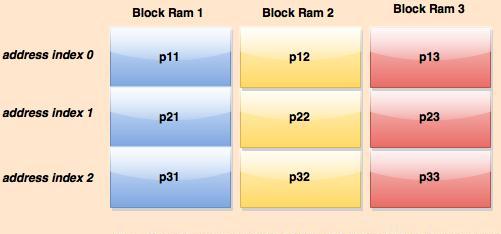


Figure 3: Normal Matrix

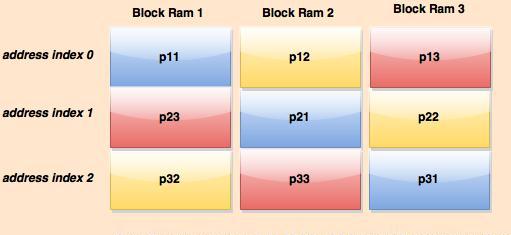


Figure 4: Circulant Matrix

The IP core stores the Matrix internally as shown in Figure 4. Please keep this in mind when offloading data. Using figure 4, assuming we are to offload data from IP core after asserting the UN\_LOAD input we wait until the READY flag becomes high then DOUT will contain first data p13, and subsequently in the next clock circles p12, p11, … then next row p22, p21, p23, and finally last row p31, p33, and p32. After the last data comes out of the output port DOUT the UN\_LOADING\_DONE flag will be set high. Please also note that it is the duty of the user to organize the matrix data in the way it is desired.

Before putting the IP core in the *unloading state* the bank should be specified, keep in mind that for unloading the Bank\_sel\_in input is inverted (0 => Lower bank, 1 => upper bank)

The following flags serve as feedback: READY, UNLOADING\_DONE.

There are two (2) ways to offload data from the IP core:

1. First case: When data has just been loaded into the IP Core and the LOADING\_DONE flag is set high, or when the IP core has just been reset.
2. Second Case: When the IP core has just finished performing a P and G Matrix multiplication and OP\_DONE flag is set high.

### First case: Sequence of micro operations for offloading Matrix data from IP Core (after data has just been loaded).

Signals that Influence UN\_LOAD input are: RST, LOAD, and Bank\_sel\_in.

Flags: LOADING\_DONE, READY, UN\_LOADING\_DONE

**Step 1:**

1. LOADING\_DONE => Wait until it is set high.
2. LOAD => set low.
3. Bank\_sel\_in: set to proper bank ('1' => Upper bank, '0' => Lower bank)
4. UN\_LOAD => set high

**Step 2:**

1. RST => Set low
2. READY => wait until it is high, start receiving data immediately

**Step 3:**

1. UN\_LOADING\_DONE => Becomes high when data completely loaded.

### First case: Sequence of micro operations for offloading Matrix data from IP Core (when IP Core is reset).

Signals that Influence UN\_LOAD input are: RST, LOAD, and Bank\_sel\_in.

Flags: READY, UN\_LOADING\_DONE

**Step 1:**

1. RST => set high (at least 3 clock cycles)
2. Bank\_sel\_in: set to proper bank ('1' => Upper bank, '0' => Lower bank)
3. LOAD => set low
4. UN\_LOAD => set high

**Step 2:**

1. RST => Set low
2. READY => wait until it is high, start receiving data immediately

**Step 3:**

1. UN\_LOADING\_DONE: Becomes high when data completely loaded.

### Second case: Sequence of micro operations for offloading Matrix data from IP Core (After IP has completed multiplication operation).

Signals that Influence UN\_LOAD input are: Bank\_sel\_in.

Flags: OP\_DONE, READY, UN\_LOADING\_DONE.

**Step 1:**

1. OP\_DONE => Wait until flag is set high.
2. UN\_LOAD => set high.

**Step 2:**

1. READY => wait until it is high, start receiving data immediately

**Step 3:**

1. UN\_LOADING\_DONE => Becomes high when data completely loaded.

## P and G (Input)

These input pins are used to set P and G matrix. The configuration has been described in table 2 previously please refer to it.

Basically these input determine what kind of matrix multiplication will take place between P and G matrix.

There are two (2) ways to multiply P and G matrix with the IP core.

1. **First case:** The IP core is empty, P matrix data has not yet been loaded into the IP core. We specify what bank P Matrix data should be loaded then we proceed by loading P matrix data into IP core (keep in mind that for loading we set Bank\_sel\_in as follows, ‘0’ => upper bank, ‘1’ => lower bank). After loading the IP core with P matrix data, then the IP core can be taken to multiplication state to perform the multiplication specified by P and G setting.
2. **Second Case:** The IP core already contains Matrix data from a previous multiplication. We specify what bank data should be saved then we take the IP to a desired matrix multiplication state based on P and G setting.

### First case: Sequence of micro operations when using new P matrix data

**Step 1:**

1. *Follow the sequence for LOADING, wait until LOADING\_DONE is set high then proceed as follows*
2. Bank\_sel\_in => set to proper bank ('1' => P matrix in up bank, '0' => P matrix in low bank)
3. Configure P and G as required (refer to Table 1)
4. LOAD => Set low
5. UNLOAD => Set low

**Step 2:**

1. Wait until OP\_DONE becomes high

### Second case: Sequence of micro operations when using old matrix data

**Step 1:**

1. RST => set high (at least 3 clock cycles)
2. Bank\_sel\_in: set to proper bank ('1' => P matrix in up bank, '0' => P matrix in low bank)
3. LOAD => set low
4. UN\_LOAD => set high
5. P and G (Set according to operation, refer to table 2)

**Step 2:**

1. Wait until OP\_DONE becomes high

1. Bank\_sel\_in is inverted for Read Address but it is not for Write Address. When Bank\_sel\_in is set high the Write Address points to lower bank, but the Read address points to upper bank because of the inversion. When Bank\_sel\_in is set low, Write Address points to upper bank and Read address points to lower bank. [↑](#footnote-ref-1)