# Polymorphic Register Files Simulator

Generated by Doxygen 1.8.11

# Contents

# Chapter 1

# Description

The Polymorphic Register File (PRF) are memory modules designed in order to allow fast parallel access to matrices in high performance applications. Through the use of mapping functions ( m_v(int,int,scheme,int,int), m_↩ h(int,int,scheme,int,int), A_standard(int,int,int,int)} ) an N-Dimensional matrix is stored in an N+1-Dimensional structure enabling, for certain type of matrix accesses, a faster retrival of the data. The PRF are implemented using an array of independent memory modules that can be read and written in parallel. There are multiple ways in which the data can be organized, those will be referred to as *Access Schemes*. Each Access Scheme allows to read the data stored in the PRF - in a conflict free manner - with different shapes, called *Access Types*. Assuming that the PRF is bidimensional and that is implemented using p∗q independent memory modules organized in a pxq matrix, all the possible parallel access type supported by a PRF are the following:

| Access Type | Description |
|---|---|
| Rectangular | pxq sub-matrix rectangle of the original matrix. |
| Row | p∗q element of a row of the original matrix. |
| Column | p∗q element of a column of the original matrix. |
| Secondary Diagonal | p∗q element of a main diagonal of the original matrix. |
| Transposed Rectangular | qxp sub-matrix rectangle of the original matrix. |

Once that the original input matrix is stored in a PRF using an access scheme, it will be possible to perform conflict free parallel accesses with one or more access types. The table below shows what access type are available for each access scheme.

| Access Type | Description |
|---|---|
| Rectangle only | Allows rectangular conflict free accesses. |
| Rect&Row | Allows rectangular, row, main diagonal and secondary diagonal conflict free access. |
| Rect&Col | Allows rectangular, column, main diagonal and secondary diagonal conflict free access. |
| Row&Col | Allows rectangular, row and column conflict free access. |
| Rect&Trect | Allows rectangular and transposed rectangular conflict free access. |

This simulator was written to allow a easier visualization of the effect of each access scheme and to provide a platform for the exploration of the access schemes.

The simulator is implemented following the description given in

## Installation

The sources for the program are available on git ( once obtained the permission from the owner of the repository ).

To download the code execute the following line from the terminal.

```
git clone https://github.com/giuliostramondo/prf-simulator.git
```

To compile the sources

```
cd prf_simulator
make
```

This produces an executable called prf_console in the ./bin folder.

## Usage

Usage: ./prf [Options]

-N <num> Change the horizontal size of the input matrix (default 9)

-M <num> Change the vertical size of the input matrix (default 9)

-p <num> Change the horizontal size of the PRF (default 3)

-q <num> Change the horizontal size of the PRF (default 3)

-s <num> Change the schema used by the PRF (default 0 -> RECTANGLE_ONLY) other schemes :

| Access Scheme | Description |
|---|---|
| 0 | Rectangle only |
| 1 | Rect&Row |
| 2 | Rect&Col |
| 3 | Row&Col |
| 4 | Rect&Trect |

Once that the prf_console is executed, an input matrix NxM is created and its cells are populated with different cells identifier. A PRF with p∗q different memory modules organized in a pxq matrix is instantiated, the memory modules are populated using the NxM input matrix and the given access scheme. The user can now type commands to the console in order to interact with the PRF simulator.

| Command | Description |
|---|---|
| show PRF; | Prints a graphical representation that shows how the input matrix ismapped in the 3D PRF. Each layer highlights in red the elements of theinput matrix stored in the pxq memory modules at the correspondent in-dex. After the highlighted matrix, a pxq matrix is printed to show whichPRF memory module is storing which data. |
| show s; | Prints the current schema value. |
| show matrix; | Prints the original NxM input matrix. |
| A[<num1>][<num2>]; | Performs a single access on the PRF; The memory module storing theaccessed data is identified using the functions m_v() and m_h(), the in-dex within the memory module is computed using A_standard(). Thosesteps are all implemented in readFromPR() which is called when this command is invoked. |

| Command | Description |
|---|---|
| A[<num1>][<num2>],<ACC_TYPE>; | Performs a block access. (num1,num2) are the coordinate of the top-leftelement in the accessed block, the shape of the block is specified by theaccess type <ACC_TYPE>. The original matrix is going to be printed,highlighting the element accessed in parallel. After the output producedby the PRF is shown in the form of a pxq matrix, where each item repre-sent the element read at the respective index. Lastly the conflict matrixis printed, this shows how many accesses have been performed on eachPRF memory module. Ideally if the block access is conflict free, the ma-trix will contain only 1. The highest number in this matrix also represents-the minimum number of memory accesses necessary, given the currentaccess scheme, to perform the block access. |
| A∗; | Performs one block access for each access type, therefore resulting in 5different block accesses. |
| set s <num>; | Changes the access scheme used by the PRF. After this command thedata in the PRF are remapped using the given scheme. |
| AGU(<num1>,<num2>,<ACC_TYPE>); | Prints the output of the AGU module generating the list addresses of the elements that are required to be accessed in order to perform a block access of the shape defined by <ACC_TYPE>. |
| m(<num1>,<num2>); | Given the current PRF scheme, shows the output of the functions m_v and m_h, defining which memory module holds the element having logical index (<num1>,<num2>). |

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

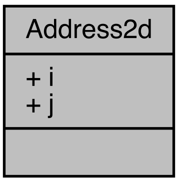
# Data Structure Documentation

## 4.1 Address2d Struct Reference

Data structure used for representing a 2D address.

```
#include <prf.h>
```

Collaboration diagram for Address2d:



### Data Fields

- int **i**
- int **j**

### 4.1.1 Detailed Description

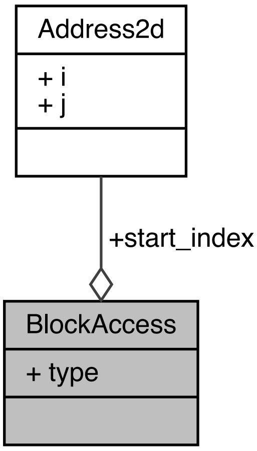
Data structure used for representing a 2D address.

The documentation for this struct was generated from the following file:

- src/prf.h

## 4.2 BlockAccess Struct Reference

Collaboration diagram for BlockAccess:



### Data Fields

- Address2d **start_index**
- acc_type **type**

The documentation for this struct was generated from the following file:

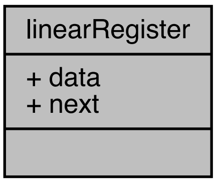
- src/prf.h

## 4.3 linearRegister Struct Reference

Data structure used for representing a linearly accessible register.

```
#include <prf.h>
```

Collaboration diagram for linearRegister:



## Data Fields

- int data

  *The integer stored by this node.*
- struct list ∗ next

  *Pointer to the next node.*

### 4.3.1 Detailed Description

Data structure used for representing a linearly accessible register.

To represent the fact that the register is accessible linearly a list structure has been used.

### 4.3.2 Field Documentation

#### 4.3.2.1 int linearRegister::data

The integer stored by this node.

#### 4.3.2.2 struct list∗ linearRegister::next

Pointer to the next node.

The documentation for this struct was generated from the following file:

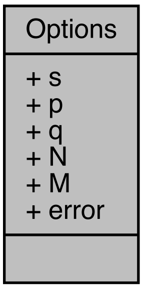
- src/prf.h

## 4.4 Options Struct Reference

Data structure used to store the user's given arguments.

```
#include <utility.h>
```

Collaboration diagram for Options:



### Data Fields

- scheme s

  *The mapping scheme used by the PRF.*
- int p

  *Horizontal size of the PRF.*
- int q

  *Vertical size of the PRF.*
- int N

  *Horizontal size of the input matrix.*
- int M

  *Vertical size of the input matrix.*
- int error

  *Variable used when the user arguments generate errors.*

### 4.4.1 Detailed Description

Data structure used to store the user's given arguments.

The documentation for this struct was generated from the following file:

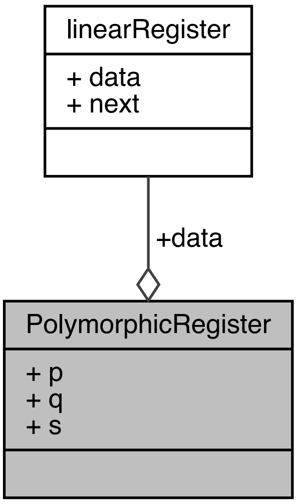
- src/utility.h

## 4.5 PolymorphicRegister Struct Reference

Data structure used for representing a Polymorphic Register.

```
#include <prf.h>
```

Collaboration diagram for PolymorphicRegister:



## Data Fields

- linearRegister ** data

  *2D array of linearRegister (lists).*
- int p

  *Size of the first dimension of the linearRegister array.*
- int q

  *Size of the second dimension of the linearRegister array.*
- scheme s

  *Eunum which identifies the mapping scheme used by this register.*

## 4.5.1 Detailed Description

Data structure used for representing a Polymorphic Register.

## 4.5.2 Field Documentation

### 4.5.2.1 linearRegister** PolymorphicRegister::data

2D array of linearRegister (lists).

### 4.5.2.2 int PolymorphicRegister::p

Size of the first dimension of the linearRegister array.

### 4.5.2.3 int PolymorphicRegister::q

Size of the second dimension of the linearRegister array.

### 4.5.2.4 scheme PolymorphicRegister::s

Eunum which identifies the mapping scheme used by this register.

**See also**

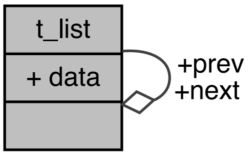
scheme

The documentation for this struct was generated from the following file:

- src/prf.h

## 4.6 t_list Struct Reference

Collaboration diagram for t_list:



### Data Fields

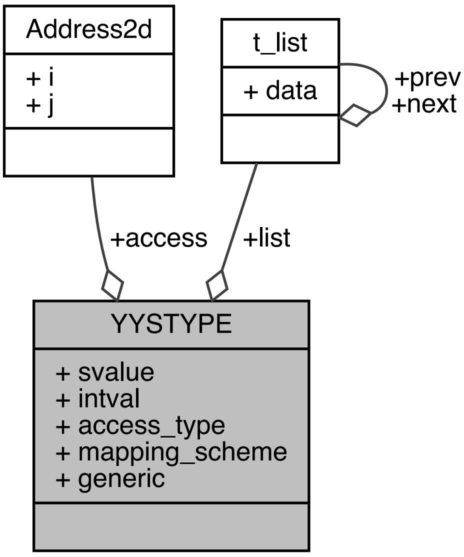
- void ∗ **data**
- struct t_list ∗ **next**
- struct t_list ∗ **prev**

The documentation for this struct was generated from the following file:

- src/collection.h

## 4.7 YYSTYPE Union Reference

Collaboration diagram for YYSTYPE:



### Data Fields

- char ∗ **svalue**
- int **intval**
- acc_type **access_type**
- scheme **mapping_scheme**
- t_list ∗ **list**
- void ∗ **generic**
- Address2d ∗ **access**

The documentation for this union was generated from the following file:

- src/parser.tab.h

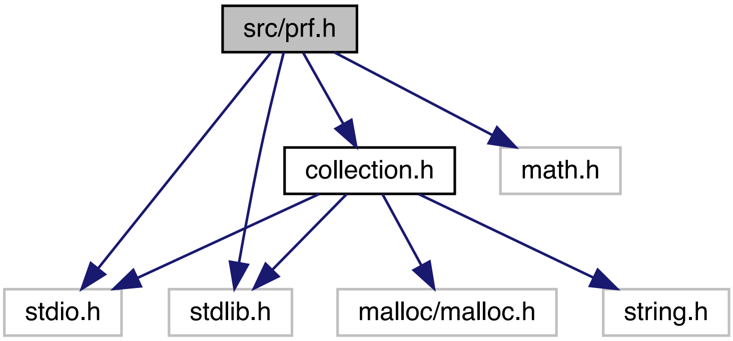
# Chapter 5

# File Documentation

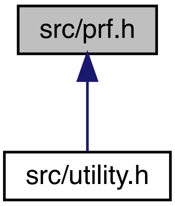
## 5.1 src/prf.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "collection.h"
```

Include dependency graph for prf.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct linearRegister

  *Data structure used for representing a linearly accessible register.*
- struct Address2d

  *Data structure used for representing a 2D address.*
- struct BlockAccess
- struct PolymorphicRegister

  *Data structure used for representing a Polymorphic Register.*

## Macros

- #define ERR -1

  *The default error value.*

## Enumerations

- enum scheme {
  RECTANGLE_ONLY, RECT_ROW, RECT_COL, ROW_COL,
  RECT_TRECT, **UNDEFINED** }

  *Enum containing all the available mapping scheme.*
- enum acc_type {
  RECTANGLE, TRANSP_RECTANGLE, ROW, COLUMN,
  MAIN_DIAG, SECONDARY_DIAG, **DEFAULT** }

  *Enum containing all the available access types.*

## Functions

- int m_v (int index_i, int index_j, scheme s, int p, int q)

  *Given two input indices and a PRF mapping scheme, outputs the row of the corrispondent linear register where the data is stored.*
- int m_h (int index_i, int index_j, scheme s, int p, int q)

  *Given two input indices and a PRF mapping scheme, outputs the column of the corrispondent linear register where the data is stored.*

- int A_standard (int index_i, int index_j, int p, int q)

  *Given two input indices and a PRF mapping scheme, outputs the linear register index where the data is stored.*
- int **A_custom** (PolymorphicRegister ∗pR, int index_i, int index_j, int alpha, int beta, acc_type type)
- PolymorphicRegister ∗ createPolymorphicRegister (int p, int q, int linRegSize)

  *Allocates memory required for a Polymorphic Register and returns a pointer to it.*
- void writeToPR (PolymorphicRegister ∗pR, int data, int index_i, int index_j)

  *Stores an integer at the given "logical" index in the Polymorphic Register given as argument.*
- int readFromPR (PolymorphicRegister ∗pR, int index_i, int index_j)

  *Reads an integer at the given "logical" index in the Polymorphic Register given as argument.*
- int ∗∗ parallelReadFromPR (PolymorphicRegister ∗pR, int z)

  *Reads an array of integer at the given depth in the Polymorphic Register given as argument.*
- int ∗∗ readBlock (PolymorphicRegister ∗pR, int index_i, int index_j, acc_type type)

  *Performs a block read on the PolymorphicRegister.*
- int ∗∗ **readBlockCustom** (PolymorphicRegister ∗pR, int index_i, int index_j, acc_type type)
- int ∗∗ computeConflicts (PolymorphicRegister ∗pR, int index_i, int index_j, acc_type type)

  *Computes the conflict matrix relative to a block read on the PolymorphicRegister.*
- Address2d ∗ AGU (int index_i, int index_j, int p, int q, acc_type type)

  *Generates all the 2D logical addresses of the elements read in a block read.*
- int **compareAddress** (void ∗a, void ∗b)
- t_list ∗ **parallelAccessCoverage** (PolymorphicRegister ∗pR, t_list ∗parallel_accesses)

## Variables

- int N

  *Size of the horizontal dimension of the original matrix.*
- int M

  *Size of the vertical dimension of the original matrix.*

## 5.1.1 Macro Definition Documentation

### 5.1.1.1 #define ERR -1

The default error value.

## 5.1.2 Enumeration Type Documentation

### 5.1.2.1 enum acc_type

Enum containing all the available access types.

**Enumerator**

***RECTANGLE*** Access p x q rectangle.

***TRANSP_RECTANGLE*** Access qxp rectangle.

***ROW*** Access 1 x p∗q rows.

***COLUMN*** Access p∗q x 1 columns.

***MAIN_DIAG*** Access elements in the main diagonal.

***SECONDARY_DIAG*** Access elements in the secondary diagonal.

#### 5.1.2.2 enum scheme

Enum containing all the available mapping scheme.

suitable for mapping input matrix logical addresses into physical Polymorphic Register addresses;

**Enumerator**

- ***RECTANGLE_ONLY*** This access scheme allows only conflict free rectangular block accesses.
- ***RECT_ROW*** This access scheme allows only conflict free rectangular, row, main diagonal and secondary diagonal block accesses.
- ***RECT_COL*** This access scheme allows only conflict free rectangular, column, main diagonal and secondary diagonal block accesses.
- ***ROW_COL*** This access scheme allows only conflict free rectangular, column, and row block accesses.
- ***RECT_TRECT*** This access scheme allows only conflict free rectangular, and transposed rectangular block accesses.

### 5.1.3 Function Documentation

#### 5.1.3.1 int A_standard ( int *index_i*, int *index_j*, int *p*, int *q* )

Given two input indices and a PRF mapping scheme, outputs the linear register index where the data is stored.

**Parameters**

| | |
|---|---|
| *index←↩_i* | index of the access. |
| *index←↩_j* | index of the access. |
| *p* | size of the PRF on its first dimension. |
| *q* | size of the PRF on its second dimension. |

**Returns**

Correspondent index in the linear register.

#### 5.1.3.2 Address2d∗ AGU ( int *index_i*, int *index_j*, int *p*, int *q*, acc_type *type* )

Generates all the 2D logical addresses of the elements read in a block read.

**Parameters**

| | |
|---|---|
| *index←↩_i* | index of the top-left element in the block read. |
| *index←↩_j* | index of the top-left element in the block read. |
| *p* | size of the PRF on its first dimension. |
| *q* | size of the PRF on its second dimension. |
| *type* | access type specifying the shape of the block access. |

**Returns**

list of the 2D addresses of the block read.

#### 5.1.3.3 int∗∗ computeConflicts ( PolymorphicRegister ∗ *pR*, int *index_i*, int *index_j*, acc_type *type* )

Computes the conflict matrix relative to a block read on the PolymorphicRegister.

**Parameters**

| | |
|---|---|
| *pR* | Pointer to the Polymorphic Register. |
| *index←↩ _i* | Logical index on the horizontal dimension of the top-left element in the accessed block. |
| *index←↩ _j* | Logical index on the vertical dimension of the top-left element in the accessed block. |
| *type* | Access Type defining the shape of the block access. |

**Returns**

2D array containing the conflict matrix.

#### 5.1.3.4 PolymorphicRegister∗ createPolymorphicRegister ( int *p*, int *q*, int *linRegSize* )

Allocates memory required for a Polymorphic Register and returns a pointer to it.

**Parameters**

| | |
|---|---|
| *p* | size of the PRF on its first dimension. |
| *q* | size of the PRF on its second dimension. |
| *linRegSize* | size of each linear register in the PRF. |

**Returns**

Pointer to the Polymorphic Register

#### 5.1.3.5 int m_h ( int *index_i*, int *index_j*, scheme *s*, int *p*, int *q* )

Given two input indices and a PRF mapping scheme, outputs the column of the corrispondent linear register where the data is stored.

**Parameters**

| | |
|---|---|
| *index←↩ _i* | index of the access. |
| *index←↩ _j* | index of the access. |
| *s* | selected mapping scheme for the access. |
| *p* | size of the PRF on its first dimension. |
| *q* | size of the PRF on its second dimension. |

**Returns**

Correspondent linear register column.

#### 5.1.3.6 int m_v ( int *index_i,* int *index_j,* scheme *s,* int *p,* int *q* )

Given two input indices and a PRF mapping scheme, outputs the row of the corrispondent linear register where the data is stored.

**Parameters**

| | |
|---|---|
| *index_i* | index of the access. |
| *index_j* | index of the access. |
| *s* | selected mapping scheme for the access. |
| *p* | size of the PRF on its first dimension. |
| *q* | size of the PRF on its second dimension. |

**Returns**

Correspondent linear register row.

#### 5.1.3.7 int** parallelReadFromPR ( PolymorphicRegister * *pR,* int *z* )

Reads an array of integer at the given depth in the Polymorphic Register given as argument.

**Parameters**

| | |
|---|---|
| *pR* | Pointer to the Polymorphic Register. |
| *z* | Depth of the 2D array in the Polymorphic Register. |

**Returns**

2D array resulting from the parallel read.

#### 5.1.3.8 int** readBlock ( PolymorphicRegister * *pR,* int *index_i,* int *index_j,* acc_type *type* )

Performs a block read on the PolymorphicRegister.

**Parameters**

| | |
|---|---|
| *pR* | Pointer to the Polymorphic Register. |
| *index_i* | Logical index on the horizontal dimension of the top-left element in the accessed block. |
| *index_j* | Logical index on the vertical dimension of the top-left element in the accessed block. |
| *type* | Access Type defining the shape of the block access. |

**Returns**

2D array resulting from the block read.

#### 5.1.3.9 int readFromPR ( PolymorphicRegister ∗ *pR*, int *index_i*, int *index_j* )

Reads an integer at the given "logical" index in the Polymorphic Register given as argument.

**Parameters**

| | |
|---|---|
| *pR* | Pointer to the Polymorphic Register. |
| *index_i* | Logical index on the first dimension. |
| *index_j* | Logical index on the second dimension. |

**Returns**

integer at the given logical position.

#### 5.1.3.10 void writeToPR ( PolymorphicRegister ∗ *pR*, int *data*, int *index_i*, int *index_j* )

Stores an integer at the given "logical" index in the Polymorphic Register given as argument.

**Parameters**

| | |
|---|---|
| *pR* | Pointer to the Polymorphic Register. |
| *index_i* | Logical index on the first dimension. |
| *index_j* | Logical index on the second dimension. |
| *data* | integer to store |

**Returns**

Pointer to the Polymorphic Register

### 5.1.4 Variable Documentation

#### 5.1.4.1 int M

Size of the vertical dimension of the original matrix.

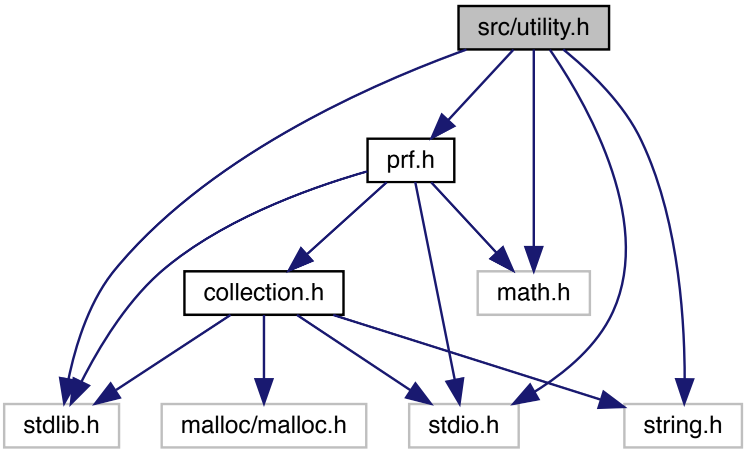
#### 5.1.4.2 int N

Size of the horizontal dimension of the original matrix.

## 5.2 src/utility.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "prf.h"
```

Include dependency graph for utility.h:



### Data Structures

- struct Options

  *Data structure used to store the user's given arguments.*

### Functions

- Options parseArguments (int argc, char ∗∗argv)

  *Parses the user arguments.*
- void printMatrix (int ∗∗inputMatrix, int dim1, int dim2)

  *Prints in the console a formatted 2D matrix.*
- void printConflicts (int ∗∗inputMatrix, int dim1, int dim2)

  *Prints in the console a conflict matrix, highlighting the conflicts.*
- void printMatrixHighlight (int ∗∗inputMatrix, int dim1, int dim2, int ∗∗highlightMatrix, int dimH1, int dimH2)

  *Prints in the console a matrix, highlighing certain elements.*
- void printUsage (char ∗programName)

  *Prints in the console the usage informations.*
- void performBlockRead (int index_i, int index_j, acc_type type, int ∗∗data_elements1, PolymorphicRegister ∗pR, int mode)

  *Prints the report information of a block read.*
- char ∗ **accessStringFromAccessType** (acc_type type)
- int **compareAddress** (void ∗a, void ∗b)

## 5.2.1 Function Documentation

### 5.2.1.1 Options parseArguments ( int *argc*, char ∗∗ *argv* )

Parses the user arguments.

**Parameters**

| | |
|---|---|
| *argc* | number of user arguments. |
| *argv* | list of user's arguments. |

**Returns**

Options struct containing the settings defined by the user.

**See also**

Options

### 5.2.1.2 void performBlockRead ( int *index_i*, int *index_j*, acc_type *type*, int ∗∗ *data_elements1*, PolymorphicRegister ∗ *pR*, int *mode* )

Prints the report information of a block read.

**Parameters**

| | |
|---|---|
| *index_i* | the horizontal index of the top-left element of the block read. |
| *index_j* | the vertical index of the top-left element of the block read. |
| *type* | access type defining the block access shape. |
| *data_elements1* | the 2D array containing original input matrix. |
| *pR* | pointer to the PolymorphicRegister used for the block read. |
| *mode* | select the access mode ( STANDARD or CUSTOM ) |

### 5.2.1.3 void printConflicts ( int ∗∗ *inputMatrix*, int *dim1*, int *dim2* )

Prints in the console a conflict matrix, highlighting the conflicts.

**Parameters**

| | |
|---|---|
| *inputMatrix* | the 2D array containing the confict's data. |
| *dim1* | the horizontal dimension of the given matrix. |
| *dim2* | the vertical dimension of the given matrix. |

#### 5.2.1.4 void printMatrix ( int ∗∗ *inputMatrix,* int *dim1,* int *dim2* )

Prints in the console a formatted 2D matrix.

**Parameters**

| | |
|---|---|
| *inputMatrix* | the 2D array containing the data. |
| *dim1* | the horizontal dimension of the given matrix. |
| *dim2* | the vertical dimension of the given matrix. |

#### 5.2.1.5 void printMatrixHighlight ( int ∗∗ *inputMatrix,* int *dim1,* int *dim2,* int ∗∗ *highlightMatrix,* int *dimH1,* int *dimH2* )

Prints in the console a matrix, highlighing certain elements.

**Parameters**

| | |
|---|---|
| *inputMatrix* | the 2D array containing all the matrix data. |
| *dim1* | the horizontal dimension of the given matrix. |
| *dim2* | the vertical dimension of the given matrix. |
| *highlightMatrix* | the 2D array containing the elements of the inputMatrix to highlight. |
| *dimH1* | the horizontal dimension of the given highlightMatrix. |
| *dimH2* | the vertical dimension of the given highlightMatrix. |

#### 5.2.1.6 void printUsage ( char ∗ *programName* )

Prints in the console the usage informations.

**Parameters**

| | |
|---|---|
| *programName* | name of the executable |

# Index