Python to Verilog

p2v

elite

Memory wrapper

P2V-mem: memory wrapper generator

COPYRIGHT

Email: eyalhoc@gmail.com

This project uses our open-source Python to Verilog compiler, available at:

https://github.com/eyalhoc/p2v

VERSION HISTORY TABLE

Version	Description	Date
0.1.0	Basic tier	7/10/2025
0.2.0	ECC tier	7/11/2025
0.3.0	AXI tier	7/13/2025

CONTENTS

Copyright 1

version history table 2

Introduction 5

What is a P2V memory wrapper? 5

IP tiers 5

Basic tier features 5

ECC tier features 5

AXI tier features 5

Basic Tier 6

Architecture – Basic Tier 6

Pin description – Basic Tier 7

Clocks 7

Ports 8

Generation – Basic Tier 9

Build parameters 9

Building from the command line 10

Building with a Python file 10

Building with a CSV file 11

ECC Tier 12

Architecture – ECC Tier 12

Sram padding 12

ECC Encoding and Decoding 13

Additional Ports for ECC Tier 14

Generation – ECC Tier 15

Build parameters 15

AXI Tier 16

Architecture – AXI Tier 16

Parallel access 16
AXI interface 16
ECC byte access 16
Testbench tasks 19
Write 19
Read 19
Write file 19

INTRODUCTION

What is a P2V memory wrapper?

A memory library supplies basic SRAM modules. To create a larger memory, tiling the SRAM modules together is required. This can be done by combining different SRAMs together, in different dimensions, port types and physical attributes. In addition, adding advanced features can improve timing, power consumption and robustness.

IP tiers

The p2v-mem IP is licensed under 3 tiers: Basic, ECC and AXI.

BASIC TIER FEATURES

1. Create large memories using two-dimensional SRAM tiling.
2. Support single or dual port of all read / write and clock combinations.
3. Support async reset, sync reset or both.
4. Support Bit and Byte select.
5. Flip-Flop implementation option.
6. Optional sampling of read data for better timing.
7. Testbench read, write and file loading tasks.
8. Generate memories from a Python file or a csv file.

ECC TIER FEATURES

- Includes Basic Tier features

1. Support mixing multiple SRAMs for the edges to match the exact requested dimensions.
2. ECC two-bit detection, one bit correction.
3. ECC input and output sampling options for better timing.
4. ECC error injection

AXI TIER FEATURES

- Includes Basic and ECC tier features

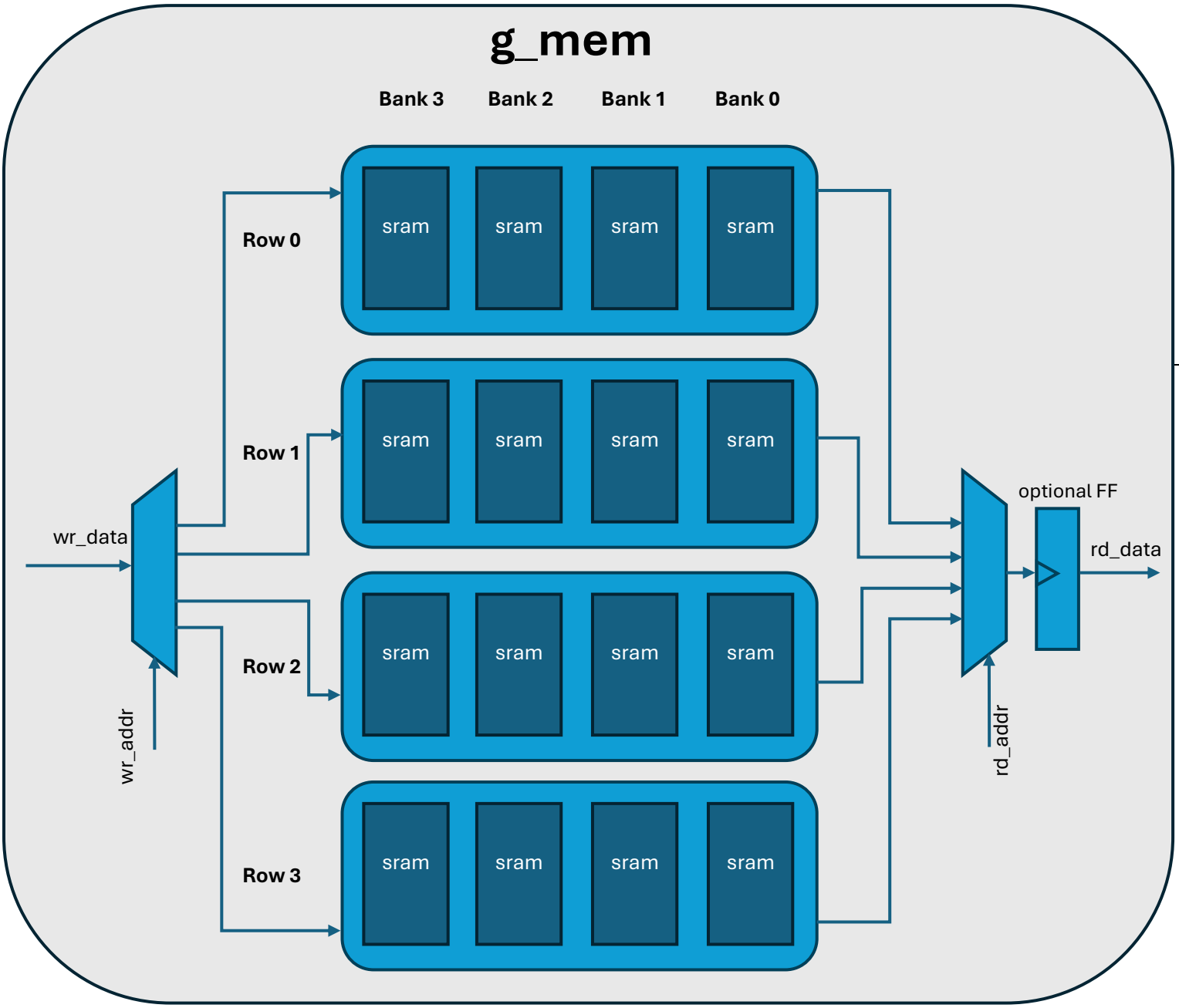
1. Parallel access from different AXI buses to different memory rows.
2. ECC bit select support (by performing read-modify-write).
3. Supports partial line read (partial bank read) for power reduction
4. Back pressure support.

BASIC TIER

Architecture – Basic Tier

The basic tier architecture uses a single identical sram for all tiles.

The memory generator extracts the attributes from the sram; data width, depth, single or dual clock, bit sel, etc. and build the memory wrapper accordingly. The required number of banks and rows is calculated, and the RTL is built creating the top ports according to the sram configuration and connecting all wires as needed.



Pin description – Basic Tier

CLOCKS

Single clock

Direction	Pin name	Bits	Exists if	Description
Input	clk	1		Clock
Input	rst_n	1	sync_reset is False	Async reset active low
Input	reset	1	sync_reset is True	Sync reset active high

One write port and one read port

Direction	Pin name	Bits	Exists if	Description
Input	wclk	1		Write clock
Input	wrst_n	1	sync_reset is False	Write async reset active low
Input	wreset	1	sync_reset is True	Write sync reset active high
Input	rclk	1		Read clock
Input	rrst_n	1	sync_reset is False	Read async reset active low
Input	rreset	1	sync_reset is True	Read sync reset active high

Multilpe read or write ports

Direction	Pin name	Bits	Exists if	Description
Input	clk0	1	Always	Port 0 clock
Input	clk0_rst_n	1	sync_reset is False	Port 0 async reset active low
Input	clk0_reset	1	sync_reset is True	Port 0 sync reset active high
Input	clk1	1		Port 1 clock
Input	clk1_rst_n	1	sync_reset is False	Port 1 async reset active low
Input	clk1_reset	1	sync_reset is True	Port 1 sync reset active high

PORTS

$n is the port index when port number is either 1 or 2.

Direction	Pin name	Bits	Exists if	Description
Input	wr${n}	1	Port $n is write	Write strobe
Input	wr${n}_addr	Log2(line_num)	Port $n is write	Write address
Input	wr${n}_data	Bits	Port $n is write	Write data
Input	wr${n}_bsel	Bits	Port $n is write and has bit select	Write bit select
Input	wr${n}_strb	Bits / 8	Port $n is write and has byte select	Write byte select
Input	rd${n}	1	Port $n is read	Read strobe
Input	rd${n}_addr	Log2(line_num)	Port $n is read	Read address
Output	rd${n}_data	Bits	Port $n is read	Read data
Output	rd${n} _valid	1	Port $n is read	Read data qualifier

Generation – Basic Tier

BUILD PARAMETERS

Name	Type	Legal range	Default	Description
name	str, None		None	Explicitly set module name
sram_name	str, None		None	Sram Verilog module name, None uses FFs
bits	int	[1 .. 8K]	32	Total data width
line_num	Int	[1 .. 1M]	4K	Total depth
sample_rd_out	bool		False	Sample read outputs for better timing
sync_reset	bool		False	Use sync reset instead of async reset

All p2v modules can show their top-level parameters but running with the -help flag

```
python3 build/basic/g_mem_top.py -help

g_mem_top module parameters:
 * name = None (NoneType) # explicitly set module name
 * sram_name = None (NoneType) # name of sram verilog module, None uses flip-
flops
 * bits = 64 (int) # data width
 * line_num = 1024 (int) # line number
 * sample_rd_out = False (bool) # sample read port outputs for better timing
 * sync_reset = False (bool) # use sync reset instead of async reset
```


BUILDING FROM THE COMMAND LINE

A memory wrapper can be generated directly by calling g_mem.py with the configuration parameters

```
python3 build/basic/g_mem_top.py -I $TSMC_LIB -params
'{"name":"carmel_main","sram_name":"TS1N7UHDLVTA1024X64M2WBZHOCP","bits":1024,"li
ne_num":"16*1024","sample_rd_out":True}'

p2v-INFO: created: g_mux__num16_bits1024_encodeFalse_sampleTrue_validTrue.sv
p2v-INFO: created:
g_sram__sram_nameTS1N7UHDLVTA1024X64M2WBZHOCP_bits64_line_num1024_bit_sel1.sv
p2v-INFO: created: carmel_main_mem__row__single.sv
p2v-INFO: created: carmel_main_mem__row.sv
p2v-INFO: created: carmel_main_mem.sv
p2v-INFO: verilog generation completed successfully (4 sec)
p2v-INFO: verilog lint completed successfully
p2v-INFO: completed successfully
```


BUILDING WITH A PYTHON FILE

Multiple memories can be built using a P2V wrapper.

```
from p2v import p2v

import g_mem_top

class basic_mem(p2v):

    def module(self, project="carmel"):

        # main sram
        g_mem_top.g_mem_top(self).module(name=f"{project}_main", bits=1024,
line_num=256*1024, sram_name="TS1N7UHDLVTA512X256M1WBZHOCP", sample_rd_out=True)
```


The memories are built when the wrapper is compiled.

```
python3 build/basic/basic_mem.py -I $TSMC_LIB

p2v-INFO: created: g_mux__num16_bits1024_encodeFalse_sampleTrue_validTrue.sv
p2v-INFO: created:
g_sram__sram_nameTS1N7UHDLVTA1024X64M2WBZHOCP_bits64_line_num1024_bit_sel1.sv
p2v-INFO: created: carmel_main_mem__row__single.sv
p2v-INFO: created: carmel_main_mem__row.sv
p2v-INFO: created: carmel_main_mem.sv
p2v-INFO: verilog generation completed successfully (4 sec)
p2v-INFO: verilog lint completed successfully
p2v-INFO: completed successfully
```

Once the memories are created their configurations are listed in a csv file: cache/g_mem_top.gen.csv

	A	B	C	D	E	F
1	name	sram_name	bits	line_num	sample_rd_out	sync_reset
2	carmel_main	"TS1N7UHDLVTA1024X64M2WBZHOCP"	1024	16384	True	False

BUILDING WITH A CSV FILE

P2V can also generate memories from a csv like the one generated above.

```
python3 build/basic/g_mem_top.py -params cache/g_mem_top.gen.csv -I $TSMC_LIB

p2v-INFO: starting gen iteration 0/0
p2v-INFO: created: g_mux__num16_bits1024_encodeFalse_sampleTrue_validTrue.sv
p2v-INFO: created:
g_sram__sram_nameTS1N7UHDLVTA1024X64M2WBZHOCP_bits64_line_num1024_bit_sel1.sv
p2v-INFO: created: carmel_main_mem__row__single.sv
p2v-INFO: created: carmel_main_mem__row.sv
p2v-INFO: created: carmel_main_mem.sv
p2v-INFO: verilog lint completed successfully
p2v-INFO: completed successfully
```

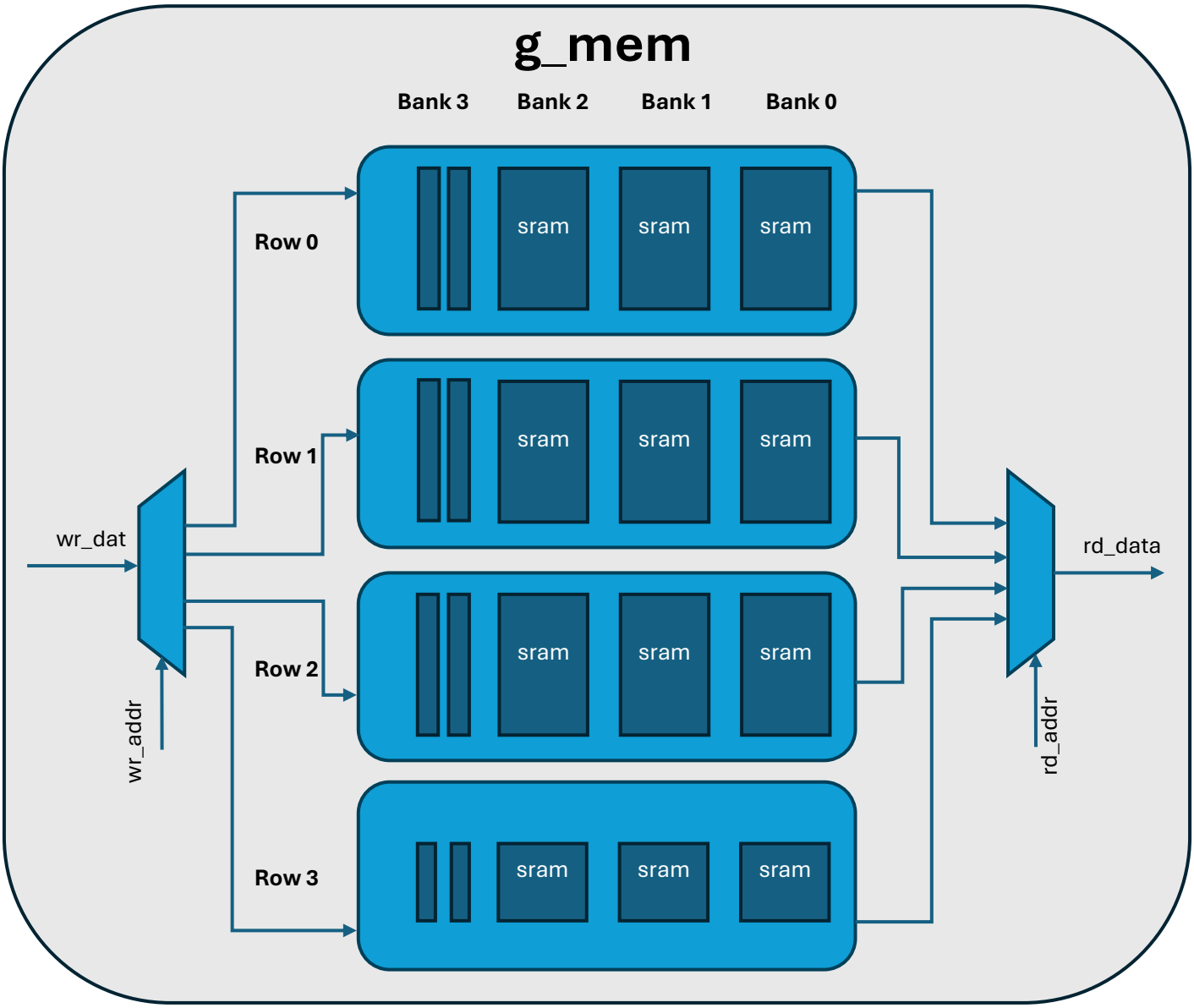

ECC TIER

Architecture – ECC Tier

SRAM PADDING

ECC is an error detection and correction mechanism. The ECC can detect two errors and can correct one error based on hamming code redundancy. When writing data to memory additional redundancy bits are added, this requires making the memory a little wider. Since ECC is typically added to large memories which use large internal SRAMs, using another bank of such SRAMs for the few redundancy bits is wasteful. To solve this, we present SRAM padding which allows us to use different dimension SRAMs for the left bank and for the bottom row.

This can be achieved by using the Python parameters: left_sram_name, bottom_sram_name and bottom_left_sram_name.

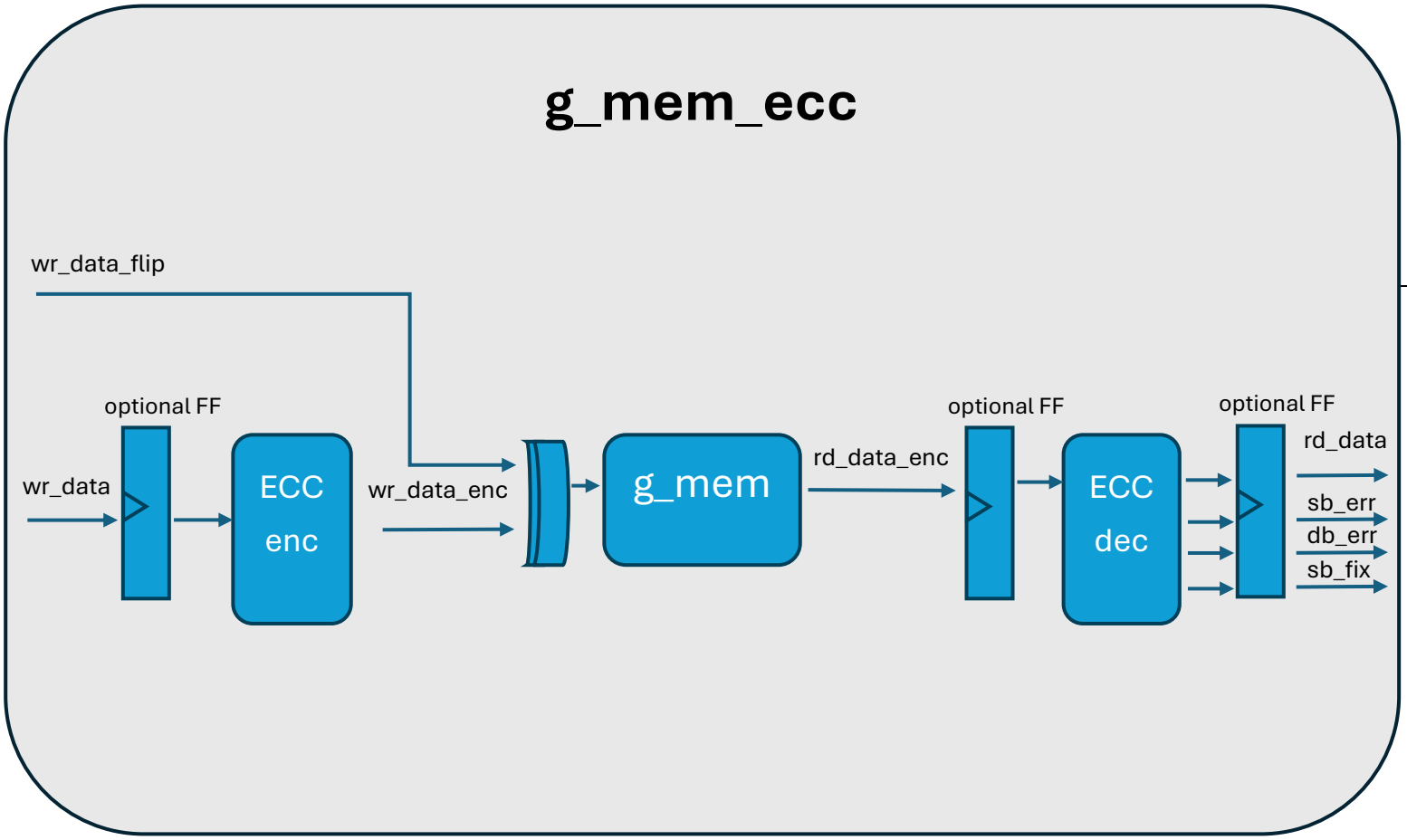


ECC ENCODING AND DECODING

When ECC is enabled a hamming code encoder and decoder are inserted. These can detect two errors in a memory word and to fix a single error. A single bit error cannot always be fixed, if the error occurs on the information bits it will be fixed otherwise it will only be detected.

It is possible to inject intentional errors for testing purposes by flipping selected bits of the encoded data.

When using ECC, the encoded data word is slightly larger than the original data word, this requires adding an additional bank to the memory. It is recommended to use an additional narrower SRAM to match better the exact required data width without using suboptimal SRAMs. This can be done by using the SRAM padding feature.



ECC tier uses the same pins as the Basic tier with additions.

ADDITIONAL PORTS FOR ECC TIER

$n is the port index when port number is either 1 or 2.

Direction	Pin name	Bits	Exists if	Description
Input	wr${n}_data_flip	Bits	Port $n is write and ECC is enabled	Flip data bits to induce ECC errors
Output	rd${n} _ecc_sb_err	1	Port $n is read and ECC is enabled	ECC single error detected
Output	rd${n} _ecc_db_err	1	Port $n is read and ECC is enabled	ECC double error detected
Output	rd${n} _ecc_sb_fix	1	Port $n is read and ECC is enabled	ECC single error fixed

Generation – ECC Tier

BUILD PARAMETERS

ECC tier uses the same build parameters as the Basic tier with additions.

Name	Type	Legal range	Default	Description
left_sram_name	str, None		None	Sram Verilog module name for left bank if it is different than main sram
bottom_sram_name	str, None		None	Sram Verilog module name for bottom bank if it is different than main sram
bottom_eft_sram_name	str, None		None	Sram Verilog module name for bottom left bank if it is different than main sram
ecc	bool		False	Enable ECC encoding
sample_wr_in	bool		False	Sample write inputs for better timing
sample_rd_in	bool		False	Sample read inputs for better timing
sample_rd_out	bool		False	Sample read outputs for better timing

All p2v modules can show their top-level parameters but running with the -help flag


```
python3 build/ecc/g_mem_top.py -help
g_mem_top module parameters:
 * name = None (NoneType) # explicitly set module name
 * sram_name = None (NoneType) # name of sram verilog module, None uses flip-
flops
 * bits = 64 (int) # data width
 * line_num = 1024 (int) # line number
 * left_sram_name = None (NoneType) # name of narrow sram module for left column
 * bottom_sram_name = None (NoneType) # name of short sram module for bottom line
 * bottom_left_sram_name = None (NoneType) # name of small sram module for bottom
left corner
 * ecc = False (bool) # insert ecc detection and correction
 * sample_wr_in = False (bool) # sample write port inputs for better timing
 * sample_rd_in = False (bool) # sample read port inputs for better timing
 * sample_rd_out = False (bool) # sample read port outputs for better timing
 * sync_reset = False (bool) # use sync reset instead of async reset
```


AXI TIER

Architecture – AXI Tier

AXI INTERFACE

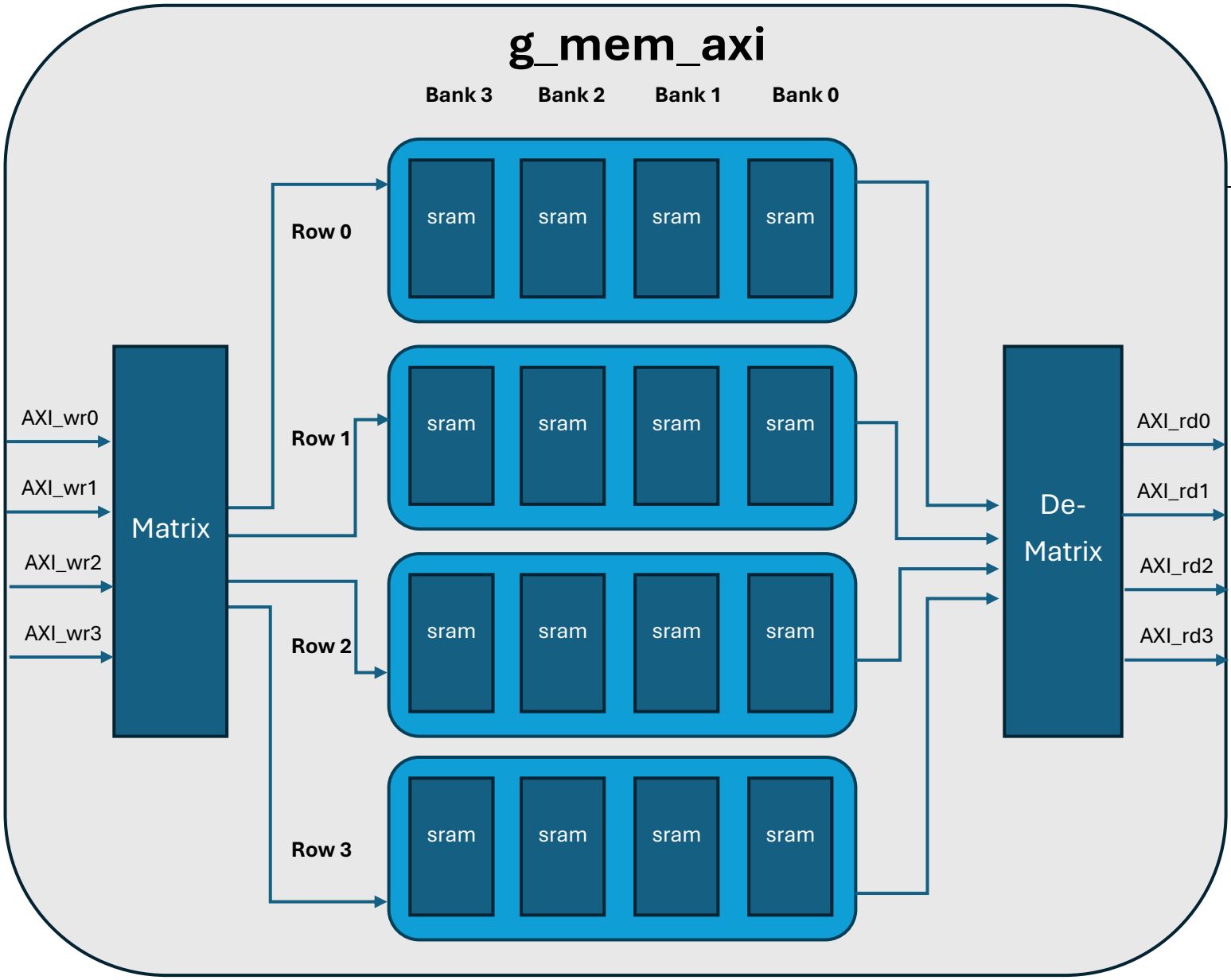
The AXI Tier supports an AXI interface per memory port. I the memory has two ports (two read or two write ports) two AXI buses will be used. When an AXI bus is used the memory supports internally back pressure coming back from the AXI bus.

PARALLEL ACCESS

A matrix is added to allow different ports to access different SRAM rows simultaneously. If two or more ports attempt to access the same row at the same time a round robin arbiter fairly manages access.

ECC BYTE ACCESS

To support byte access to ECC enabled memories the AXI tier wrapper will perform read-modify-write on the requested byte since normal byte select cannot be used on the ECC encrypted data in memory.



Pin description – AXI Tier

$i is the AXI interface index

Direction	Pin name	Bits	Exists if	Description
Input	ACLK	1		AXI clock
Input	ARESETN	1		Async reset active low
Input	AWID$i	Id bits		Write command ID
Input	AWADDR$i	Log2(line_num)		Write command address
Input	AWLEN$i	4		Write command burst length
Input	AWSIZE$i	3		Write command data size
Input	AWVALID$i	1		Write command valid
output	AWREADY$i	1		Write command ready
Input	WDATA$i	Bits		Write data
Input	WSTRB$i	Bits / 8		Write byte select
Input	WLAST$i	1		Write last in burst
Input	WVALID$i	1		Write valid
Output	WREADY$i	1		Write ready
Output	BID$i	Id bits		Write response ID
Output	BRESP$i	2		Write response status
Output	BVALID$i	1		Write response valid
Input	BREADY$i	1		Write response ready
Input	ARID$i	Id bits		Read command ID
Input	ARADDR$i	Log2(line_num)		Read command address
Input	ARLEN$i	4		Read command burst length
Input	ARSIZE$i	3		Read command data size
Input	ARVALID$i	1		Read command valid
output	ARREADY$i	1		Read command ready
Output	RID$i	Id bits		Read ID
Output	RRESP$i	2		Read status
Output	RDATA$i	Bits		Read data
Input	RLAST$i	1		Read last in burst
Output	RVALID$i	1		Read valid
Input	RREADY$i	1		Read ready

Generation – AXI Tier

BUILD PARAMETERS

AXI tier uses the same build parameters as the ECC tier with additions.

AXI interface does not support a sync reset.

Name	Type	Legal range	Default	Description
axi	bool		bool	Use AXI interface
Interface_num	int		1	Number of parallel access interfaces
Id_bits	Int		4	AXI ID bits

All p2v modules can show their top-level parameters but running with the -help flag

```
python3 build/axi/g_mem_top.py -help
g_mem_top module parameters:
 * name = None (NoneType) #  explicitly set module name
 * sram_name = None (NoneType) #  name of sram verilog module, None uses flip-
flops
 * bits = 64 (int) #  data width
 * line_num = 1024 (int) #  line number
 * left_sram_name = None (NoneType) #  name of narrow sram module for left column
 * bottom_sram_name = None (NoneType) #  name of short sram module for bottom
line
 * bottom_left_sram_name = None (NoneType) #  name of small sram module for
bottom left corner
 * ecc = False (bool) #  insert ecc detection and correction
 * sample_wr_in = False (bool) #  sample write port inputs for better timing
 * sample_rd_in = False (bool) #  sample read port inputs for better timing
 * axi = False (bool) #  use AXI interface
 * interface_num = 1 (int) #  number of parallel interfaces
 * id_bits = 4 (int) #  sample read port outputs for better timing
 * sample_rd_out = False (bool) #  sample read port outputs for better timing
 * sync_reset = False (bool) #  use sync reset instead of async reset
```


TESTBENCH TASKS

All tasks take care of internal build of memory making all banks and rows seamless. The tasks do not change regardless of the internal build or configuration.

Tasks are similar for all tiers.

WRITE

Write a line of data to memory.

task write(input[31:0] addr, input [bits-1:0] data)

READ

Reads a line of data from memory.

task read(input[31:0] addr, output [bits-1:0] data)

WRITE FILE

Write a hexadecimal text file to memory

task write_file(input[1023:0] filename)

