Whitney LPDDR5 SystemC Model - User Guide

# Whitney LPDDR5 Memory Controller SystemC Model

## User Guide v1.0

## Table of Contents

1. 1. Introduction
2. 2. Getting Started
3. 3. Installation Guide
4. 4. Model Overview
5. 5. Interface Specifications
6. 6. Configuration Guide
7. 7. Running Simulations
8. 8. Customization and Integration
9. 9. Performance Analysis
10. 10. Troubleshooting
11. 11. API Reference
12. 12. Examples and Use Cases
13. 13. Support and Contact

## Introduction

### What is the Whitney LPDDR5 SystemC Model?

The Whitney LPDDR5 SystemC Model is a high-fidelity behavioral model of the Whitney Memory Controller designed for system-level verification and performance analysis. This model enables customers to:

* • Verify System Integration: Test the memory controller within larger SoC designs
* • Analyze Performance: Evaluate memory subsystem performance under various workloads
* • Develop Software: Create and test software drivers and applications
* • Validate Configurations: Test different memory configurations and timing parameters

### Key Benefits

* • Fast Simulation: Significantly faster than RTL simulation for system-level testing
* • Accurate Behavior: Cycle-accurate modeling of key memory controller functions
* • Easy Integration: Standard SystemC interfaces for seamless SoC integration
* • Comprehensive Logging: Detailed transaction and performance monitoring
* • Flexible Configuration: Configurable timing parameters and memory settings

### Target Audience

This user guide is intended for:  
- System architects and verification engineers  
- Software developers working with memory subsystems  
- Performance analysis engineers  
- Integration engineers working on SoC designs

## Getting Started

### Quick Start Checklist

Before you begin, ensure you have:

* • [ ] SystemC 2.3.3 or later installed
* • [ ] GCC 4.8+ or equivalent C++11 compiler
* • [ ] Basic understanding of SystemC and memory controller concepts
* • [ ] Whitney LPDDR5 SystemC model files

### 5-Minute Quick Start

1. 1. Extract the model files to your working directory
2. 2. Set environment variables:  
   bash  
   export SYSTEMC\_HOME=/usr/local/systemc-2.3.3
3. 3. Build the model:  
   bash  
   make
4. 4. Run the basic test:  
   bash  
   make run
5. 5. View results in the console output and whitney\_trace.vcd file

## Installation Guide

### System Requirements

#### Minimum Requirements

* • OS: Linux (Ubuntu 18.04+, CentOS 7+, RHEL 7+)
* • Memory: 4GB RAM
* • Disk: 1GB free space
* • Compiler: GCC 4.8+ with C++11 support

#### Recommended Requirements

* • OS: Linux (Ubuntu 20.04+)
* • Memory: 8GB RAM
* • Disk: 2GB free space
* • Compiler: GCC 7.0+ or Clang 6.0+

### SystemC Installation

#### Option 1: Install from Source (Recommended)

1. 1. Download SystemC:  
   bash  
   wget https://www.accellera.org/images/downloads/standards/systemc/systemc-2.3.3.tar.gz  
   tar -xzf systemc-2.3.3.tar.gz  
   cd systemc-2.3.3
2. 2. Configure and Build:  
   bash  
   mkdir objdir && cd objdir  
   ../configure --prefix=/usr/local/systemc-2.3.3 --enable-debug  
   make -j$(nproc)  
   sudo make install
3. 3. Set Environment Variables:  
   bash  
   export SYSTEMC\_HOME=/usr/local/systemc-2.3.3  
   export LD\_LIBRARY\_PATH=$SYSTEMC\_HOME/lib-linux64:$LD\_LIBRARY\_PATH

#### Option 2: Package Manager Installation

Ubuntu/Debian:  
bash  
sudo apt-get update  
sudo apt-get install systemc-dev  
export SYSTEMC\_HOME=/usr

CentOS/RHEL:  
bash  
sudo yum install systemc-devel  
export SYSTEMC\_HOME=/usr

### Model Installation

1. 1. Extract Model Files:  
   bash  
   tar -xzf whitney\_lpddr5\_systemc\_model.tar.gz  
   cd whitney\_lpddr5\_systemc\_model
2. 2. Verify Installation:  
   bash  
   make help
3. 3. Test Build:  
   bash  
   make clean && make

### Optional Tools

#### GTKWave (for waveform viewing)

```bash

# Ubuntu/Debian

sudo apt-get install gtkwave

# CentOS/RHEL

sudo yum install gtkwave  
```

## Model Overview

### Architecture Overview

The Whitney LPDDR5 SystemC Model consists of several key components:

┌─────────────────────────────────────────────────────────────┐  
│ Whitney SystemC Model │  
├─────────────────┬─────────────────┬─────────────────────────┤  
│ AXI Interface │ APB Interface │ DFI Interface │  
│ │ │ │  
│ ┌─────────────┐ │ ┌─────────────┐ │ ┌─────────────────────┐ │  
│ │Write Channel│ │ │ Registers │ │ │Command Interface │ │  
│ │Read Channel │ │ │ Status │ │ │Write Data Interface │ │  
│ │Response Ch. │ │ │ Control │ │ │Read Data Interface │ │  
│ └─────────────┘ │ └─────────────┘ │ └─────────────────────┘ │  
└─────────────────┴─────────────────┴─────────────────────────┘  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────────────────────────────────────────────────┐  
│ Internal Architecture │  
├─────────────────┬─────────────────┬─────────────────────────┤  
│ Scheduler │ Sequencer │ Buffer Manager │  
│ │ │ │  
│ • Page Policy │ • Command Gen │ • Write Buffers │  
│ • Bank Arb │ • Timing FSM │ • Read Buffers │  
│ • QoS │ • Refresh Ctrl │ • Address Translation │  
└─────────────────┴─────────────────┴─────────────────────────┘

### Key Features

#### Memory Controller Features

* • LPDDR5 Support: Full LPDDR5 protocol compliance
* • Multi-Bank Operation: Up to 8 banks with independent timing
* • Open Page Policy: Optimized for sequential access patterns
* • Refresh Management: Automatic refresh scheduling and execution
* • QoS Support: Configurable quality of service parameters

#### Interface Features

* • AXI4 Slave: 64-bit data width, 40-bit address space
* • APB Configuration: Full register access for configuration and monitoring
* • DFI 5.0: Complete DDR PHY Interface for LPDDR5
* • WCK Control: Write Clock management for high-speed operation

#### Verification Features

* • Transaction Logging: Detailed logging of all memory transactions
* • Performance Counters: Built-in statistics collection
* • Waveform Generation: VCD output for signal analysis
* • Configurable Scenarios: Flexible test configuration options

## Interface Specifications

### AXI4 Slave Interface

The Whitney model implements a full AXI4 slave interface with the following specifications:

#### Signal List

| Signal Name | Width | Direction | Description |  
|-------------|-------|-----------|-------------|  
| mc0aclk | 1 | Input | AXI clock |  
| mc0aresetn | 1 | Input | AXI reset (active low) |  
| mc0axiawid | 12 | Input | Write address ID |  
| mc0axiawaddr | 40 | Input | Write address |  
| mc0axiawlen | 8 | Input | Write burst length |  
| mc0axiawsize | 3 | Input | Write transfer size |  
| mc0axiawburst | 2 | Input | Write burst type |  
| mc0axiawvalid | 1 | Input | Write address valid |  
| mc0axiawready | 1 | Output | Write address ready |  
| mc0axiwdata | 64 | Input | Write data |  
| mc0axiwstrb | 8 | Input | Write strobes |  
| mc0axiwlast | 1 | Input | Write last |  
| mc0axiwvalid | 1 | Input | Write data valid |  
| mc0axiwready | 1 | Output | Write data ready |  
| mc0axibid | 12 | Output | Write response ID |  
| mc0axibresp | 2 | Output | Write response |  
| mc0axibvalid | 1 | Output | Write response valid |  
| mc0axibready | 1 | Input | Write response ready |  
| mc0axiarid | 12 | Input | Read address ID |  
| mc0axiaraddr | 40 | Input | Read address |  
| mc0axiarlen | 8 | Input | Read burst length |  
| mc0axiarsize | 3 | Input | Read transfer size |  
| mc0axiarburst | 2 | Input | Read burst type |  
| mc0axiarvalid | 1 | Input | Read address valid |  
| mc0axiarready | 1 | Output | Read address ready |  
| mc0axirid | 12 | Output | Read data ID |  
| mc0axirdata | 64 | Output | Read data |  
| mc0axirresp | 2 | Output | Read response |  
| mc0axirlast | 1 | Output | Read last |  
| mc0axirvalid | 1 | Output | Read data valid |  
| mc0axirready | 1 | Input | Read data ready |

#### Supported Features

* • Burst Types: INCR, WRAP (FIXED not supported)
* • Burst Lengths: 1-256 beats
* • Transfer Sizes: 1, 2, 4, 8 bytes
* • Outstanding Transactions: Up to 16 read, 16 write
* • Response Types: OKAY, SLVERR

### APB Configuration Interface

#### Signal List

| Signal Name | Width | Direction | Description |  
|-------------|-------|-----------|-------------|  
| mcpsel | 1 | Input | APB select |  
| mcpenable | 1 | Input | APB enable |  
| mcpwr | 1 | Input | APB write enable |  
| mcpaddr | 10 | Input | APB address |  
| mcpwdata | 32 | Input | APB write data |  
| mcprdata | 32 | Output | APB read data |  
| mcpready | 1 | Output | APB ready |  
| mcpslverr | 1 | Output | APB slave error |

### DFI Interface

The DFI (DDR PHY Interface) provides the connection to the LPDDR5 PHY:

#### Command Interface

| Signal Name | Width | Description |  
|-------------|-------|-------------|  
| dfics0p0/p1 | 2 | Chip select phase 0/1 |  
| dfics1p2/p3 | 2 | Chip select phase 2/3 |  
| dfiaddress0p0/p1 | 7 | Address phase 0/1 |  
| dfiaddress1p2/p3 | 7 | Address phase 2/3 |  
| dfiresetn | 1 | DDR reset |

#### Write Data Interface

| Signal Name | Width | Description |  
|-------------|-------|-------------|  
| dfiwrdata0-15 | 32 | Write data phases 0-15 |  
| dfiwrdatamask0-15 | 4 | Write data mask phases 0-15 |  
| dfiwrdataen0-15 | 4 | Write data enable phases 0-15 |

#### Read Data Interface

| Signal Name | Width | Description |  
|-------------|-------|-------------|  
| dfirddata0-15 | 32 | Read data phases 0-15 |  
| dfirddatavalid | 1 | Read data valid |  
| dfirddataen | 1 | Read data enable |

## Configuration Guide

### Register Map

The Whitney model provides a comprehensive register interface for configuration and monitoring:

#### Core Configuration Registers

| Address | Register Name | Access | Reset Value | Description |  
|---------|---------------|--------|-------------|-------------|  
| 0x000 | SEQCONTROL | R/W | 0x00000001 | Sequencer control and status |  
| 0x004 | BUFCONFIG | R/W | 0x00000000 | Buffer configuration |  
| 0x008 | DDRCONFIG | R/W | 0x00030000 | DDR type and configuration |  
| 0x00C | DDRADRCONFIG | R/W | 0x00000000 | Address mapping configuration |  
| 0x048 | REFRESHCNTRL | R/W | 0x00001F40 | Refresh control |

#### Timing Configuration Registers

| Address | Register Name | Description |  
|---------|---------------|-------------|  
| 0x020 | ACTIMINGREG1 | tCL, tWL, tRCD, tRP |  
| 0x024 | ACTIMINGREG2 | tRAS, tRC, tRRD, tFAW |  
| 0x028 | ACTIMINGREG3 | tWTR, tRTP, tCCD, tBL |  
| 0x02C | ACTIMINGREG4 | tREFI, tRFC, tXSR |  
| 0x030 | ACTIMINGREG5 | tMRD, tMOD, tZQCS |  
| 0x034 | ACTIMINGREG6 | tCKE, tCKESR, tXP |  
| 0x038 | ACTIMINGREG7 | tODTon, tODToff, tAON |  
| 0x03C | ACTIMINGREG8 | tDQSCK, tWCKPRE, tWCKDQO |  
| 0x040 | ACTIMINGREG9 | tRPRE, tWPRE, tMRR |  
| 0x044 | ACTIMINGREG10 | tVREF, tFCDLR, tOSCO |

#### Status and Monitoring Registers

| Address | Register Name | Access | Description |  
|---------|---------------|--------|-------------|  
| 0x01C | PMUSTATUS | R | PMU status and sequencer type |  
| 0x04C | TESTCONFIG | R/W | Test and debug configuration |

### Configuration Examples

#### Basic LPDDR5 Configuration

```cpp  
// Configure for LPDDR5 operation  
apbwrite(0x008, 0x00030520); // DDRCONFIG: LPDDR5, BL32, 64-bit  
apbwrite(0x000, 0x00000001); // SEQCONTROL: Enable sequencer  
apbwrite(0x048, 0x00001F41); // REFRESHCNTRL: Enable refresh, 7.8us

// Configure timing parameters (example values)  
apbwrite(0x020, 0x12345678); // ACTIMINGREG1  
apbwrite(0x024, 0x9ABCDEF0); // ACTIMINGREG2  
apbwrite(0x028, 0x11223344); // ACTIMING\_REG3  
```

#### Performance Optimization Configuration

cpp  
// Optimize for high performance  
apb\_write(0x004, 0x00000080); // BUF\_CONFIG: Larger buffers  
apb\_write(0x00C, 0x00000123); // DDR\_ADR\_CONFIG: Optimized mapping  
apb\_write(0x04C, 0x00000001); // TEST\_CONFIG: Enable performance counters

#### Low Power Configuration

cpp  
// Configure for low power operation  
apb\_write(0x048, 0x00003E80); // REFRESH\_CNTRL: Longer refresh interval  
apb\_write(0x034, 0x12345678); // AC\_TIMING\_REG6: Extended power-down timings

## Running Simulations

### Basic Simulation Flow

#### Step 1: Environment Setup

```bash

# Set SystemC environment

export SYSTEMCHOME=/usr/local/systemc-2.3.3  
export LDLIBRARYPATH=$SYSTEMCHOME/lib-linux64:$LDLIBRARYPATH

# Navigate to model directory

cd whitneylpddr5systemc\_model  
```

#### Step 2: Build the Model

```bash

# Clean previous builds

make clean

# Build with debug information

make CXXFLAGS="-g -O0 -DDEBUG"

# Or build optimized version

make  
```

#### Step 3: Run Simulation

```bash

# Run with default testbench

./whitney\_simulation

# Run with custom parameters

./whitney\_simulation +trace +verbose

# Run with specific test duration

./whitneysimulation +maxtime=1000ns  
```

#### Step 4: Analyze Results

```bash

# View console output

less simulation.log

# View waveforms

gtkwave whitney\_trace.vcd

# Analyze performance statistics

grep "Statistics" simulation.log  
```

### Advanced Simulation Options

#### Command Line Arguments

The simulation executable supports several command line options:

| Option | Description | Example |  
|--------|-------------|---------|  
| +trace | Enable VCD tracing | ./whitney\_simulation +trace |  
| +verbose | Enable verbose logging | ./whitney\_simulation +verbose |  
| +max\_time= | Set simulation time limit | ./whitney\_simulation +max\_time=500ns |  
| +seed= | Set random seed | ./whitney\_simulation +seed=12345 |  
| +config= | Load configuration file | ./whitney\_simulation +config=test.cfg |

#### Environment Variables

| Variable | Description | Default |  
|----------|-------------|---------|  
| WHITNEYLOGLEVEL | Logging verbosity (0-3) | 1 |  
| WHITNEYTRACEFILE | VCD trace filename | whitneytrace.vcd |  
| WHITNEYMAX\_CYCLES | Maximum simulation cycles | 100000 |

### Custom Test Scenarios

#### Creating Custom Tests

1. 1. Modify the testbench:  
   cpp  
   // In whitney\_testbench.cpp, modify stimulus\_process()  
   void AXIMaster::stimulus\_process() {  
    // Your custom test sequence here  
    write\_transaction(0x10000000, 0x123456789ABCDEF0ULL, 1);  
    read\_transaction(0x10000000, 2);  
   }
2. 2. Create configuration files:  
   ```ini  
   test\_config.cfg  
   [memory]  
   size = 1GB  
   banks = 8  
   [timing]  
   tCL = 18  
   tWL = 14  
   ```
3. 3. Build and run:  
   bash  
   make clean && make  
   ./whitney\_simulation +config=test\_config.cfg

#### Predefined Test Scenarios

The model includes several predefined test scenarios:

1. 1. Basic Functionality Test: Tests basic read/write operations
2. 2. Burst Test: Tests various burst lengths and types
3. 3. Bank Interleaving Test: Tests multi-bank operations
4. 4. Page Hit/Miss Test: Tests page policy effectiveness
5. 5. Refresh Test: Tests refresh operation
6. 6. Performance Test: Measures peak performance

To run a specific test:  
bash  
./whitney\_simulation +test=burst\_test  
./whitney\_simulation +test=performance\_test

## Customization and Integration

### Integrating with Your SystemC Design

#### Basic Integration

1. 1. Include the model header:  
   ```cpp  
   include "whitneysystemcmodel.h"  
   ```
2. 2. Instantiate the model:  
   cpp  
   WhitneySystemCModel whitney\_mc("whitney\_mc");
3. 3. Connect interfaces:  
   ```cpp  
   // Connect clocks and resets  
   whitneymc.mck(systemclock);  
   whitneymc.mcrstb(systemreset\_n);  
   // Connect AXI interface  
   whitneymc.mc0axiawaddr(axiawaddr);  
   whitneymc.mc0axiawvalid(axiawvalid);  
   // ... connect all AXI signals  
   // Connect APB interface  
   whitneymc.mcpaddr(apbpaddr);  
   whitneymc.mcpsel(apbpsel);  
   // ... connect all APB signals  
   ```

#### Advanced Integration

For complex SoC integration, consider these patterns:

1. 1. Wrapper Module:  
   ```cpp  
   SC\_MODULE(WhitneyWrapper) {  
    // Additional logic for protocol conversion  
    // Clock domain crossing  
    // Power management  
   WhitneySystemCModel whitney\_core;  
     
   SC\_CTOR(WhitneyWrapper) : whitney\_core("core") {  
    // Connect and configure  
   }  
     
   };  
   ```
2. 2. Multi-Instance Integration:  
   ```cpp  
   // For multi-channel memory systems  
   WhitneySystemCModel whitneych0("whitneych0");  
   WhitneySystemCModel whitneych1("whitneych1");  
   // Implement channel arbitration logic  
   ```

### Customizing the Model

#### Adding Custom Features

1. 1. Extend the register map:  
   cpp  
   // In whitney\_systemc\_model.cpp  
   sc\_uint<32> WhitneySystemCModel::read\_register(sc\_uint<10> addr) {  
    switch (addr.to\_uint()) {  
    // Existing registers...  
    case 0x100: return my\_custom\_reg; // Add custom register  
    default: return 0xDEADBEEF;  
    }  
   }
2. 2. Add custom processes:  
   ```cpp  
   // In constructor  
   SCMETHOD(mycustomprocess);  
   sensitive << mck.pos();  
   dontinitialize();  
   // Implementation  
   void WhitneySystemCModel::mycustomprocess() {  
    // Custom functionality  
   }  
   ```
3. 3. Modify timing behavior:  
   cpp  
   // Customize timing parameters  
   void WhitneySystemCModel::configure\_timing() {  
    // Load timing from configuration file  
    // Implement temperature/voltage scaling  
    // Add process variation modeling  
   }

#### Configuration File Support

Create a configuration system for easy customization:

```cpp  
// configmanager.h  
class ConfigManager {  
public:  
 void loadconfig(const std::string& filename);  
 uint32t gettimingparam(const std::string& param);  
 bool getfeature\_enable(const std::string& feature);  
};

// Usage in model  
ConfigManager config;  
config.loadconfig("whitneyconfig.json");  
tCL = config.gettimingparam("tCL");  
```

### Performance Optimization

#### Simulation Speed Optimization

1. 1. Reduce logging verbosity:  
   ```cpp  
   ifdef FAST\_SIM  
   define WHITNEY\_LOG(x) // Disable logging  
   else  
   define WHITNEY\_LOG(x) std::cout << x << std::endl  
   endif  
   ```
2. 2. Optimize data structures:  
   cpp  
   // Use fixed-size arrays instead of vectors for critical paths  
   // Implement efficient queue management  
   // Cache frequently accessed data
3. 3. Conditional compilation:  
   ```cpp  
   ifdef DETAILED\_MODELING  
   // Detailed timing and protocol checking  
     
   else  
   // Simplified, faster modeling  
     
   endif  
   ```

## Performance Analysis

### Built-in Performance Metrics

The Whitney model automatically collects various performance metrics:

#### Transaction Statistics

* • Total Write Transactions: Number of AXI write transactions processed
* • Total Read Transactions: Number of AXI read transactions processed
* • Average Transaction Latency: Mean time from AXI request to response
* • Peak Transaction Rate: Maximum transactions per second achieved

#### Memory Efficiency Metrics

* • Page Hit Rate: Percentage of memory accesses that hit open pages
* • Bank Utilization: Utilization percentage for each memory bank
* • Buffer Occupancy: Average occupancy of read/write buffers
* • Refresh Overhead: Percentage of time spent on refresh operations

#### Bandwidth Analysis

* • Effective Bandwidth: Actual data transfer rate achieved
* • Theoretical Bandwidth: Maximum possible bandwidth
* • Bandwidth Efficiency: Ratio of effective to theoretical bandwidth

### Accessing Performance Data

#### Runtime Statistics

```cpp  
// Enable statistics collection  
whitney.enable\_statistics(true);

// Run simulation  
scstart(1000, SCNS);

// Print statistics  
whitney.print\_statistics();

// Get specific metrics  
uint32t pagehits = whitney.getpagehits();  
uint32t pagemisses = whitney.getpagemisses();  
double hitrate = (double)pagehits / (pagehits + pagemisses);  
```

#### Custom Performance Monitoring

```cpp  
// Add custom performance counters  
class PerformanceMonitor {  
private:  
 uint64t cyclecount;  
 uint64t activecycles;  
 uint64t idlecycles;

public:  
 void updatecounters() {  
 cyclecount++;  
 if (isactive()) activecycles++;  
 else idle\_cycles++;  
 }

double get\_utilization() {  
 return (double)active\_cycles / cycle\_count;  
}

};  
```

### Performance Optimization Guidelines

#### Memory Access Patterns

1. 1. Sequential Access: Optimize for page hits  
   cpp  
   // Good: Sequential addresses in same page  
   for (int i = 0; i < 16; i++) {  
    write\_transaction(base\_addr + i\*8, data[i], i);  
   }
2. 2. Bank Interleaving: Distribute across banks  
   cpp  
   // Good: Addresses map to different banks  
   write\_transaction(0x10000000, data0, 1); // Bank 0  
   write\_transaction(0x11000000, data1, 2); // Bank 1  
   write\_transaction(0x12000000, data2, 3); // Bank 2
3. 3. Burst Optimization: Use appropriate burst lengths  
   cpp  
   // Configure for optimal burst length  
   apb\_write(DDR\_CONFIG, 0x00030520); // BL32 for LPDDR5

#### Configuration Tuning

1. 1. Buffer Sizing:  
   cpp  
   // Increase buffer depth for high-throughput applications  
   apb\_write(BUF\_CONFIG, 0x000000FF);
2. 2. Refresh Optimization:  
   cpp  
   // Adjust refresh interval based on temperature  
   uint32\_t refresh\_val = calculate\_refresh\_interval(temperature);  
   apb\_write(REFRESH\_CNTRL, refresh\_val);
3. 3. Timing Optimization:  
   cpp  
   // Optimize timing for specific LPDDR5 device  
   configure\_device\_timing("LPDDR5-6400");

### Benchmarking and Validation

#### Standard Benchmarks

The model includes several standard memory benchmarks:

1. 1. STREAM Benchmark: Measures memory bandwidth
2. 2. Random Access: Tests random access performance
3. 3. Mixed Workload: Combines read/write operations
4. 4. Latency Test: Measures access latency

Run benchmarks:  
bash  
./whitney\_simulation +benchmark=stream  
./whitney\_simulation +benchmark=random\_access  
./whitney\_simulation +benchmark=mixed\_workload

#### Custom Benchmarks

Create application-specific benchmarks:

```cpp  
// Custom benchmark for video processing  
class VideoBenchmark {  
public:  
 void runbenchmark() {  
 // Simulate video frame processing  
 for (int frame = 0; frame < 100; frame++) {  
 processframe(frame);  
 }  
 }

private:  
 void processframe(int frameid) {  
 // Read frame data  
 for (int line = 0; line < 1080; line++) {  
 readtransaction(framebuffer + line19204, line);  
 }

// Write processed data  
 for (int line = 0; line < 1080; line++) {  
 write\_transaction(output\_buffer + line\*1920\*4, processed\_data[line], line+1000);  
 }  
}

};  
```

## Troubleshooting

### Common Issues and Solutions

#### Build Issues

Issue: SystemC headers not found  
  
Error: cannot open source file "systemc.h"  
  
Solution:  
```bash

# Check SystemC installation

ls $SYSTEMC\_HOME/include/systemc.h

# Set correct path

export SYSTEMC\_HOME=/usr/local/systemc-2.3.3

# Update Makefile if needed

make SYSTEMC\_HOME=/path/to/systemc  
```

Issue: Linking errors  
  
Error: undefined reference to `sc\_main'  
  
Solution:  
```bash

# Ensure all source files are included

make clean && make

# Check library path

export LDLIBRARYPATH=$SYSTEMCHOME/lib-linux64:$LDLIBRARY\_PATH  
```

#### Runtime Issues

Issue: Simulation hangs  
Symptoms: No output, process doesn't terminate  
Solution:  
1. Check clock connections  
2. Verify reset sequence  
3. Enable debug output:  
 cpp  
 #define DEBUG\_ENABLED

Issue: Incorrect behavior  
Symptoms: Unexpected transaction responses  
Solution:  
1. Enable VCD tracing:  
 bash  
 ./whitney\_simulation +trace  
 gtkwave whitney\_trace.vcd  
  
2. Check configuration registers  
3. Verify timing parameters

Issue: Performance issues  
Symptoms: Slow simulation speed  
Solution:  
1. Disable detailed logging  
2. Reduce trace output  
3. Use optimized build:  
 bash  
 make CXXFLAGS="-O3 -DNDEBUG"

#### Memory Issues

Issue: Segmentation fault  
Solution:  
1. Check array bounds  
2. Verify pointer initialization  
3. Use debugging tools:  
 bash  
 gdb ./whitney\_simulation  
 valgrind ./whitney\_simulation

Issue: Memory leaks  
Solution:  
1. Check queue management  
2. Verify object cleanup  
3. Use memory profiling:  
 bash  
 valgrind --leak-check=full ./whitney\_simulation

### Debug Features

#### Logging Levels

The model supports multiple logging levels:

```cpp  
// Set logging level  
export WHITNEYLOGLEVEL=3

// Levels:  
// 0 = Errors only  
// 1 = Warnings and errors  
// 2 = Info, warnings, and errors  
// 3 = Debug, info, warnings, and errors  
```

#### Trace Points

Enable specific trace points:

```cpp  
// In model code

# ifdef TRACE\_AXI

std::cout << "AXI Transaction: " << transaction\_info << std::endl;

# endif

# ifdef TRACE\_DDR

std::cout << "DDR Command: " << command\_info << std::endl;

# endif

```

Build with specific traces:  
bash  
make CXXFLAGS="-DTRACE\_AXI -DTRACE\_DDR"

#### Assertion Checking

Enable runtime assertions:

```cpp  
// Enable assertions

# define SCENABLEASSERTIONS

// Use in code  
sc\_assert(condition && "Error message");  
```

### Getting Help

#### Self-Diagnosis Checklist

Before contacting support, please check:

* • [ ] SystemC version compatibility (2.3.3 or later)
* • [ ] Compiler version (GCC 4.8+ or Clang 3.4+)
* • [ ] Environment variables set correctly
* • [ ] All required files present
* • [ ] Build completed without errors
* • [ ] Basic test runs successfully

#### Collecting Debug Information

When reporting issues, please provide:

1. 1. System Information:  
   bash  
   uname -a  
   gcc --version  
   echo $SYSTEMC\_HOME
2. 2. Build Log:  
   bash  
   make clean && make 2>&1 | tee build.log
3. 3. Runtime Log:  
   bash  
   ./whitney\_simulation +verbose 2>&1 | tee runtime.log
4. 4. Configuration:  
     
   Model version  
   Custom modifications  
   Test scenario used
5. 5. Model version
6. 6. Custom modifications
7. 7. Test scenario used

## API Reference

### WhitneySystemCModel Class

#### Constructor

cpp  
WhitneySystemCModel(sc\_module\_name name);

#### Public Methods

cpp  
void configure\_timing(const TimingConfig& config);  
void set\_memory\_size(uint64\_t size\_bytes);  
void enable\_feature(const std::string& feature, bool enable);

cpp  
void enable\_statistics(bool enable);  
void reset\_statistics();  
void print\_statistics();  
uint32\_t get\_transaction\_count(TransactionType type);  
double get\_bandwidth\_utilization();  
double get\_page\_hit\_rate();

cpp  
void set\_log\_level(int level);  
void enable\_trace(const std::string& trace\_type);  
void dump\_state();

#### Configuration Structures

```cpp  
struct TimingConfig {  
 uint32t tCL; // CAS Latency  
 uint32t tWL; // Write Latency  
 uint32t tRCD; // RAS to CAS Delay  
 uint32t tRP; // Row Precharge Time  
 uint32t tRAS; // Row Active Time  
 uint32t tRC; // Row Cycle Time  
 uint32t tREFI; // Refresh Interval  
 uint32t tRFC; // Refresh Cycle Time  
};

struct MemoryConfig {  
 uint32t numbanks;  
 uint32t numranks;  
 uint32t pagesize;  
 uint32t burstlength;  
 MemoryType type; // LPDDR5, LPDDR4X, etc.  
};  
```

#### Enumerations

```cpp  
enum TransactionType {  
 WRITETRANSACTION,  
 READTRANSACTION,  
 ALL\_TRANSACTIONS  
};

enum MemoryType {  
 LPDDR4X,  
 LPDDR5,  
 LPDDR5X  
};

enum LogLevel {  
 LOGERROR = 0,  
 LOGWARNING = 1,  
 LOGINFO = 2,  
 LOGDEBUG = 3  
};  
```

### Testbench Classes

#### AXIMaster Class

```cpp  
class AXIMaster : public scmodule {  
public:  
 // Constructor  
 SCCTOR(AXIMaster);

// Transaction methods  
void write\_transaction(sc\_uint<40> addr, sc\_uint<64> data, sc\_uint<12> id = 0);  
void read\_transaction(sc\_uint<40> addr, sc\_uint<12> id = 0);  
void burst\_write(sc\_uint<40> start\_addr, const std::vector<sc\_uint<64>>& data);  
void burst\_read(sc\_uint<40> start\_addr, uint32\_t length);  
  
// Configuration methods  
void set\_transaction\_rate(double rate);  
void set\_address\_pattern(AddressPattern pattern);

};  
```

#### LPDDR5PHY Class

```cpp  
class LPDDR5PHY : public scmodule {  
public:  
 // Constructor  
 SCCTOR(LPDDR5PHY);

// Configuration methods  
void configure\_device(const DeviceConfig& config);  
void set\_timing\_parameters(const TimingConfig& timing);  
  
// Memory simulation  
void load\_memory\_image(const std::string& filename);  
void save\_memory\_image(const std::string& filename);

};  
```

## Examples and Use Cases

### Example 1: Basic Memory Test

```cpp

# include "whitneysystemcmodel.h"

int scmain(int argc, char\* argv[]) {  
 // Create clock and reset  
 scclock clk("clk", 10, SCNS);  
 scsignal rst\_n;

// Create model instance  
WhitneySystemCModel whitney("whitney");  
  
// Connect basic signals  
whitney.mck(clk);  
whitney.mc\_rst\_b(rst\_n);  
  
// Configure model  
whitney.configure\_timing(lpddr5\_timing);  
whitney.enable\_statistics(true);  
  
// Reset sequence  
rst\_n = false;  
sc\_start(20, SC\_NS);  
rst\_n = true;  
  
// Run test  
sc\_start(1000, SC\_NS);  
  
// Print results  
whitney.print\_statistics();  
  
return 0;

}  
```

### Example 2: Performance Benchmark

```cpp  
class PerformanceBenchmark {  
private:  
 WhitneySystemCModel\* whitney;  
 AXIMaster\* axi\_master;

public:  
 void runbandwidthtest() {  
 const uint32t numtransactions = 1000;  
 const uint64t startaddr = 0x10000000;

// Record start time  
 sc\_time start\_time = sc\_time\_stamp();  
  
 // Generate sequential write transactions  
 for (uint32\_t i = 0; i < num\_transactions; i++) {  
 uint64\_t addr = start\_addr + i \* 64;  
 uint64\_t data = 0xDEADBEEF00000000ULL + i;  
 axi\_master->write\_transaction(addr, data, i);  
 }  
  
 // Wait for completion  
 wait\_for\_completion();  
  
 // Calculate bandwidth  
 sc\_time end\_time = sc\_time\_stamp();  
 sc\_time duration = end\_time - start\_time;  
  
 double bandwidth = calculate\_bandwidth(num\_transactions \* 8, duration);  
 std::cout << "Achieved Bandwidth: " << bandwidth << " GB/s" << std::endl;  
}  
  
void run\_latency\_test() {  
 const uint32\_t num\_samples = 100;  
 std::vector<sc\_time> latencies;  
  
 for (uint32\_t i = 0; i < num\_samples; i++) {  
 sc\_time start = sc\_time\_stamp();  
  
 // Single transaction  
 axi\_master->read\_transaction(0x10000000 + i\*8, i);  
  
 // Wait for response  
 wait\_for\_response(i);  
  
 sc\_time end = sc\_time\_stamp();  
 latencies.push\_back(end - start);  
 }  
  
 // Calculate statistics  
 double avg\_latency = calculate\_average(latencies);  
 double max\_latency = calculate\_max(latencies);  
  
 std::cout << "Average Latency: " << avg\_latency << " ns" << std::endl;  
 std::cout << "Maximum Latency: " << max\_latency << " ns" << std::endl;  
}

};  
```

### Example 3: Custom Configuration

```cpp  
void configureformobileapplication() {  
 // Low power configuration for mobile device  
 TimingConfig mobiletiming = {  
 .tCL = 18, // Relaxed timing for power savings  
 .tWL = 14,  
 .tRCD = 18,  
 .tRP = 21,  
 .tRAS = 42,  
 .tRC = 63,  
 .tREFI = 7800, // Standard refresh interval  
 .tRFC = 280  
 };

MemoryConfig mobile\_memory = {  
 .num\_banks = 8,  
 .num\_ranks = 1,  
 .page\_size = 4096,  
 .burst\_length = 32,  
 .type = LPDDR5  
};  
  
// Apply configuration  
whitney.configure\_timing(mobile\_timing);  
whitney.configure\_memory(mobile\_memory);  
  
// Enable power-saving features  
whitney.enable\_feature("auto\_precharge", true);  
whitney.enable\_feature("power\_down", true);  
whitney.enable\_feature("self\_refresh", true);

}

void configureforhighperformance() {  
 // High performance configuration for server/desktop  
 TimingConfig perftiming = {  
 .tCL = 14, // Aggressive timing for performance  
 .tWL = 10,  
 .tRCD = 14,  
 .tRP = 14,  
 .tRAS = 32,  
 .tRC = 46,  
 .tREFI = 3900, // More frequent refresh  
 .tRFC = 210  
 };

// Apply configuration  
whitney.configure\_timing(perf\_timing);  
  
// Enable performance features  
whitney.enable\_feature("command\_queue", true);  
whitney.enable\_feature("bank\_interleaving", true);  
whitney.enable\_feature("prefetch", true);

}  
```

### Example 4: Multi-Channel System

```cpp  
class MultiChannelSystem {  
private:  
 static const int NUMCHANNELS = 4;  
 WhitneySystemCModel\* channels[NUMCHANNELS];  
 ChannelArbiter\* arbiter;

public:  
 void setupsystem() {  
 // Create multiple memory channels  
 for (int i = 0; i < NUMCHANNELS; i++) {  
 std::string name = "whitneych" + std::tostring(i);  
 channels[i] = new WhitneySystemCModel(name.c\_str());

// Configure each channel  
 configure\_channel(channels[i], i);  
 }  
  
 // Create arbiter for channel selection  
 arbiter = new ChannelArbiter("arbiter");  
 connect\_arbiter();  
}  
  
void run\_multi\_channel\_test() {  
 // Distribute transactions across channels  
 for (int txn = 0; txn < 1000; txn++) {  
 int channel = txn % NUM\_CHANNELS;  
 uint64\_t addr = 0x10000000 + (txn / NUM\_CHANNELS) \* 64;  
  
 // Send transaction to selected channel  
 send\_to\_channel(channel, addr, txn);  
 }  
  
 // Collect statistics from all channels  
 for (int i = 0; i < NUM\_CHANNELS; i++) {  
 std::cout << "Channel " << i << " Statistics:" << std::endl;  
 channels[i]->print\_statistics();  
 }  
}

};  
```

### Example 5: Software Driver Development

```cpp  
class MemoryDriver {  
private:  
 WhitneySystemCModel\* whitney;  
 uint32t baseaddress;

public:  
 // Initialize memory controller  
 bool initialize() {  
 // Reset sequence  
 writeregister(SEQCONTROL, 0x00000000); // Reset  
 wait(100, SC\_NS);

// Configure for LPDDR5  
 write\_register(DDR\_CONFIG, 0x00030520);  
 write\_register(REFRESH\_CNTRL, 0x00001F41);  
  
 // Load timing parameters  
 load\_timing\_parameters();  
  
 // Enable controller  
 write\_register(SEQ\_CONTROL, 0x00000001);  
  
 // Verify initialization  
 return verify\_initialization();  
}  
  
// Memory allocation  
void\* allocate\_memory(size\_t size) {  
 // Find free memory region  
 uint64\_t addr = find\_free\_region(size);  
  
 // Mark as allocated  
 mark\_allocated(addr, size);  
  
 return reinterpret\_cast<void\*>(addr);  
}  
  
// Memory read/write operations  
void write\_memory(void\* addr, const void\* data, size\_t size) {  
 uint64\_t mem\_addr = reinterpret\_cast<uint64\_t>(addr);  
 const uint8\_t\* src = static\_cast<const uint8\_t\*>(data);  
  
 // Perform write operations  
 for (size\_t i = 0; i < size; i += 8) {  
 uint64\_t write\_data = \*reinterpret\_cast<const uint64\_t\*>(src + i);  
 axi\_write(mem\_addr + i, write\_data);  
 }  
}  
  
void read\_memory(const void\* addr, void\* data, size\_t size) {  
 uint64\_t mem\_addr = reinterpret\_cast<uint64\_t>(addr);  
 uint8\_t\* dst = static\_cast<uint8\_t\*>(data);  
  
 // Perform read operations  
 for (size\_t i = 0; i < size; i += 8) {  
 uint64\_t read\_data = axi\_read(mem\_addr + i);  
 \*reinterpret\_cast<uint64\_t\*>(dst + i) = read\_data;  
 }  
}

private:  
 void writeregister(uint32t addr, uint32t data) {  
 // APB write transaction  
 apbwrite(addr, data);  
 }

uint32\_t read\_register(uint32\_t addr) {  
 // APB read transaction  
 return apb\_read(addr);  
}  
  
void load\_timing\_parameters() {  
 // Load LPDDR5 timing parameters  
 write\_register(AC\_TIMING\_REG1, 0x120E1215); // tCL=18, tWL=14, tRCD=18, tRP=21  
 write\_register(AC\_TIMING\_REG2, 0x2A3F1008); // tRAS=42, tRC=63, tRRD=16, tFAW=8  
 write\_register(AC\_TIMING\_REG3, 0x08040820); // tWTR=8, tRTP=4, tCCD=8, tBL=32  
 write\_register(AC\_TIMING\_REG4, 0x1E780118); // tREFI=7800, tRFC=280, tXSR=280  
}  
  
bool verify\_initialization() {  
 uint32\_t status = read\_register(PMU\_STATUS);  
 return (status & 0x30) == 0x30; // Check sequencer type = LPDDR5x  
}

};  
```

## Support and Contact

### Technical Support

For technical support with the Whitney LPDDR5 SystemC Model, please use the following resources:

#### Documentation and Resources

* • User Guide: This document (WhitneyLPDDR5SystemCUserGuide.md)
* • API Reference: Complete API documentation in Section 11
* • Example Code: Reference implementations in Section 12
* • README: Basic setup and usage instructions

#### Self-Service Support

Before contacting technical support, please:

1. 1. Check the Troubleshooting Section: Section 10 covers common issues and solutions
2. 2. Review the FAQ: Frequently asked questions and answers
3. 3. Run the Self-Diagnosis: Use the checklist in Section 10.3.1
4. 4. Check System Requirements: Verify your environment meets the requirements

#### Contacting Support

When contacting support, please provide:

1. 1. Model Version: Whitney LPDDR5 SystemC Model v1.0
2. 2. System Information: OS, compiler version, SystemC version
3. 3. Issue Description: Detailed description of the problem
4. 4. Reproduction Steps: Step-by-step instructions to reproduce the issue
5. 5. Log Files: Build logs, runtime logs, and error messages
6. 6. Configuration: Any custom configurations or modifications

#### Support Channels

* • Email Support: systemc-support@company.com
* • Online Portal: https://support.company.com/systemc
* • Knowledge Base: https://kb.company.com/whitney-systemc

### Frequently Asked Questions (FAQ)

#### General Questions

Q: What is the difference between this SystemC model and the RTL?  
A: The SystemC model is a behavioral model optimized for system-level simulation and software development. It provides the same functional behavior as the RTL but with faster simulation speed and higher abstraction level.

Q: Can I use this model for silicon validation?  
A: This model is intended for system-level verification and software development. For silicon validation, please use the RTL implementation.

Q: What LPDDR5 features are supported?  
A: The model supports core LPDDR5 features including BL32, multi-bank operation, refresh management, and DFI interface. See Section 4 for detailed feature list.

#### Installation and Setup

Q: Which SystemC version is required?  
A: SystemC 2.3.3 or later is required. SystemC 2.3.4 is recommended for best compatibility.

Q: Can I use this model on Windows?  
A: The model is primarily tested on Linux. Windows support may be available through WSL (Windows Subsystem for Linux) or Cygwin.

Q: Do I need a SystemC license?  
A: SystemC is open source and freely available from Accellera. No license is required for SystemC itself.

#### Usage and Integration

Q: How do I integrate this model with my existing SystemC testbench?  
A: See Section 8.1 for integration guidelines and examples.

Q: Can I modify the model for my specific requirements?  
A: Yes, the model is designed to be customizable. See Section 8.2 for customization guidelines.

Q: How accurate is the timing model?  
A: The model provides cycle-accurate behavior for key functions. Some detailed timing aspects are simplified for simulation speed.

#### Performance and Optimization

Q: How can I improve simulation speed?  
A: See Section 8.3 for performance optimization techniques including logging reduction and conditional compilation.

Q: What is the typical simulation speed compared to RTL?  
A: The SystemC model typically runs 10-100x faster than RTL simulation, depending on the test scenario and optimization level.

Q: Can I run multiple instances for multi-channel systems?  
A: Yes, the model supports multi-instance usage. See Example 4 in Section 12 for multi-channel implementation.

### Training and Education

#### Available Training Materials

1. 1. SystemC Fundamentals: Introduction to SystemC modeling concepts
2. 2. Memory Controller Architecture: Understanding LPDDR5 and memory controller design
3. 3. Model Usage Workshop: Hands-on training with the Whitney SystemC model
4. 4. Advanced Customization: Deep dive into model customization and extension

#### Online Resources

* • Video Tutorials: Step-by-step video guides for common tasks
* • Webinar Series: Regular webinars on SystemC modeling best practices
* • Community Forum: User community for questions and discussions
* • Sample Projects: Complete example projects demonstrating various use cases

### Feedback and Suggestions

We value your feedback to improve the Whitney LPDDR5 SystemC Model:

#### How to Provide Feedback

* • Feature Requests: Submit requests for new features or enhancements
* • Bug Reports: Report issues or unexpected behavior
* • Documentation: Suggest improvements to documentation and examples
* • Usability: Share your experience and suggestions for better usability

#### Feedback Channels

* • Email: feedback@company.com
* • Online Survey: https://survey.company.com/whitney-systemc
* • User Forum: https://forum.company.com/systemc-models
* • Direct Contact: Through your account manager or sales representative

### Version History and Updates

#### Current Version: v1.0

* • Initial release of Whitney LPDDR5 SystemC Model
* • Full AXI4, APB, and DFI interface support
* • Comprehensive testbench and examples
* • Complete documentation and user guide

#### Planned Updates

* • v1.1: Enhanced performance monitoring and additional timing parameters
* • v1.2: Support for additional LPDDR5 features and configurations
* • v2.0: Multi-channel support and advanced power modeling

#### Update Notifications

* • Subscribe to our mailing list for update notifications
* • Check the support portal for latest releases
* • Follow release notes for new features and bug fixes

### Legal and Licensing

#### Model License

This SystemC model is provided under a commercial license agreement. Please refer to your license agreement for terms and conditions of use.

#### SystemC License

SystemC is provided under the Apache 2.0 license by Accellera Systems Initiative.

#### Third-Party Components

This model may include third-party components with their own licensing terms. See the LICENSES file for details.

#### Export Control

This software may be subject to export control regulations. Please ensure compliance with applicable laws and regulations.

## Appendices

### Appendix A: Register Bit Definitions

#### SEQ\_CONTROL Register (0x000)

| Bits | Field Name | Access | Description |  
|------|------------|--------|-------------|  
| [0] | DDRINITDONE | R/W | DDR initialization complete |  
| [1] | SEQENABLE | R/W | Sequencer enable |  
| [2] | AUTOREFRESHEN | R/W | Auto refresh enable |  
| [7:3] | RESERVED | R | Reserved |  
| [15:8] | SEQSTATE | R | Current sequencer state |  
| [31:16] | RESERVED | R | Reserved |

#### DDR\_CONFIG Register (0x008)

| Bits | Field Name | Access | Description |  
|------|------------|--------|-------------|  
| [1:0] | DDRTYPE | R/W | 00=LPDDR4X, 01=LPDDR5, 10=LPDDR5X |  
| [3:2] | DATAWIDTH | R/W | 00=16-bit, 01=32-bit, 10=64-bit |  
| [7:4] | BURSTLENGTH | R/W | 0000=BL16, 0001=BL32 |  
| [11:8] | NUMBANKS | R/W | Number of banks (0-15) |  
| [15:12] | NUM\_RANKS | R/W | Number of ranks (0-3) |  
| [31:16] | RESERVED | R/W | Reserved |

### Appendix B: Timing Parameter Definitions

#### LPDDR5 Standard Timing Parameters

| Parameter | Description | Min (ns) | Typical (ns) | Max (ns) |  
|-----------|-------------|----------|--------------|----------|  
| tCL | CAS Latency | 14 | 18 | 24 |  
| tWL | Write Latency | 10 | 14 | 18 |  
| tRCD | RAS to CAS Delay | 14 | 18 | 24 |  
| tRP | Row Precharge Time | 14 | 21 | 27 |  
| tRAS | Row Active Time | 32 | 42 | 70 |  
| tRC | Row Cycle Time | 46 | 63 | 97 |  
| tREFI | Refresh Interval | 3900 | 7800 | 7800 |  
| tRFC | Refresh Cycle Time | 210 | 280 | 380 |

### Appendix C: Error Codes and Messages

#### Common Error Codes

| Code | Message | Description | Solution |  
|------|---------|-------------|----------|  
| E001 | SystemC header not found | SystemC installation issue | Check SYSTEMCHOME path |  
| E002 | Linking error | Library path issue | Check LDLIBRARY\_PATH |  
| E003 | Simulation timeout | Infinite loop or deadlock | Check clock/reset connections |  
| E004 | Invalid configuration | Register value out of range | Verify configuration values |  
| E005 | Memory allocation failed | Insufficient system memory | Increase available memory |

#### Warning Messages

| Code | Message | Description | Action |  
|------|---------|-------------|--------|  
| W001 | Timing violation | DDR timing constraint violated | Check timing parameters |  
| W002 | Buffer overflow | Internal buffer full | Reduce transaction rate |  
| W003 | Page miss | Memory access missed open page | Optimize access pattern |  
| W004 | Refresh pending | Refresh operation delayed | Check refresh configuration |

### Appendix D: Performance Benchmarks

#### Reference Performance Data

Based on typical LPDDR5-6400 configuration:

| Metric | Value | Unit |  
|--------|-------|------|  
| Peak Bandwidth | 51.2 | GB/s |  
| Random Access Latency | 120 | ns |  
| Sequential Access Latency | 80 | ns |  
| Page Hit Rate (Sequential) | 95 | % |  
| Page Hit Rate (Random) | 15 | % |  
| Refresh Overhead | 2.5 | % |

#### Benchmark Test Results

| Test Scenario | Bandwidth (GB/s) | Latency (ns) | Efficiency (%) |  
|---------------|-------------------|--------------|----------------|  
| Sequential Write | 48.5 | 85 | 94.7 |  
| Sequential Read | 49.2 | 82 | 96.1 |  
| Random Write | 12.8 | 145 | 25.0 |  
| Random Read | 15.2 | 135 | 29.7 |  
| Mixed Workload | 28.4 | 110 | 55.5 |

Document Information:  
- Title: Whitney LPDDR5 Memory Controller SystemC Model User Guide  
- Version: 1.0  
- Date: January 2024  
- Authors: SystemC Modeling Team  
- Document ID: WH-SC-UG-001

Copyright Notice:  
© 2024 Company Name. All rights reserved. This document contains confidential and proprietary information and is intended solely for the use of licensed customers.

End of User Guide