# XMOS SDRAM Component

REV A

Publication Date: 2013/10/30

# Table of Contents

# 1 Overview

IN THIS CHAPTER



## 1.1 SDRAM Controller Component

The SDRAM module is designed for 16 bit read and write access of arbitrary length at up to 50MHz clock rates. It uses an optimal pinout with address and data lines overlaid to implement 16 bit read/write with up to 13 address lines in just 20 pins.

The module currently targets the ISSI 6400 SDRAM but may easily any single data rate SDRAM memory from other manufacturers.

### 1.1.1 SDRAM Component Features

The SDRAM component has the following features:

- 
  - SDRAM geometry,
  - clock rate,
  - refresh properties,
  - server commands supported,
  - port mapping of the SDRAM.
- 
  - buffer read,
  - buffer write,
  - full row(page) read,
  - full row(page) write,
  - refresh handled by the SDRAM component itself.
- 
  - The function `sdram_server` requires just one core, the client functions, located in `sdram.h` are very low overhead and are called from the application.

### 1.1.2 Memory Requirements

| Resource | Usage |
|---|---|
| Stack | 256 bytes |
| Program | 10272 bytes |

### 1.1.3 Resource requirements

| Resource | Usage |
|---|---|
| Channels | 1 |
| Timers | 1 |
| Clocks | 1 |
| Logical Cores | 1 |

### 1.1.4 Performance

The achievable effective bandwidth varies according to the available xCORE MIPS. This information has been obtained by testing on real hardware.

| xCORE MIPS | Cores | System Clock | Max Read (MB/s) | Max Write (MB/s) |
|---|---|---|---|---|
| 50 | 8 | 400MHz | 66.84 | 70.75 |
| 57 | 7 | 400MHz | 68.13 | 71.68 |
| 66 | 6 | 400MHz | 69.83 | 73.41 |
| 80 | 5 | 400MHz | 71.68 | 74.99 |
| 100 | 4 | 400MHz | 71.89 | 75.22 |
| 100 | 3 | 400MHz | 71.89 | 75.22 |
| 100 | 2 | 400MHz | 71.89 | 75.22 |
| 62.5 | 8 | 500MHz | 66.82 | 70.34 |
| 83 | 7 | 500MHz | 68.08 | 71.47 |
| 100 | 6 | 500MHz | 69.83 | 73.19 |
| 125 | 5 | 500MHz | 71.68 | 74.76 |
| 125 | 4 | 500MHz | 71.89 | 74.99 |
| 125 | 3 | 500MHz | 71.89 | 74.99 |
| 125 | 2 | 500MHz | 71.89 | 74.99 |

## 1.2 SDRAM Memory Mapper

A memory mapper module called `module_sdram_memory_mapper` may be used in order to abstract the physical geometry of the SDRAM from the application. Its only function is to map the physical geometry of the SDRAM to a virtual byte addresses that the application can use.

### 1.2.1 Memory Requirements

| Resource | Usage |
|---|---|
| Stack | 0 bytes |
| Program | 32 bytes |

### 1.2.2 Resource Requirements

| Resource | Usage |
|---|---|
| Channels | 0 |
| Timers | 0 |
| Clocks | 0 |
| Logical Cores | 0 |

# 2 Hardware Requirements

IN THIS CHAPTER



## 2.1 Recommended Hardware

### 2.1.1 sliceKIT

This module may be evaluated using the sliceKIT Modular Development Platform, available from digikey. Required board SKUs are:

- XP-SKC-L2 (sliceKIT L16 Core Board) plus XA-SK-SDRAM plus XA-SK-XTAG2 (sliceKIT XTAG adaptor)

## 2.2 Demonstration Applications

### 2.2.1 Testbench Application

This application serves as a software regression to aid implementing new SDRAM interfaces and verifying current ones. The testbench runs a series of regression tests of increasing difficulty, beginning from using a single core for the server and a single core for the sdram_server progressing to all cores being loaded to simulate an xCORE under full load.

- Package: sc_sdram_burst
- Application: app_sdram_testbench

### 2.2.2 Benchmark Application

This application benchmarks the performance of the module. It does no correctness testing but instead tests the throughput of the SDRAM server.

- Package: sc_sdram_burst
- Application: app_sdram_benchmark

### 2.2.3 Demo Application

This application demonstrates how the module is used to access memory on the SDRAM.

- Package: sc_sdram_burst
- Application: app_sdram_demo

### 2.2.4 Display Controller Application

This combination demo employs this module along with the `module_lcd` LCD driver and the "module_display_controller" framebuffer framework component to implement a 480x272 display controller.

Required board SKUs for this demo are:

- XP-SKC-L2 (sliceKIT L16 Core Board) plus XA-SK-SDRAM plus XA-SK-LCD480 plus XA-SK-XTAG2 (sliceKIT XTAG adaptor)
- Package: sw_display_controller
- Application: app_graphics_demo

# 3 API

IN THIS CHAPTER



## 3.1 Configuration Defines

The file `sdram_conf.h` must be provided in the application source code, and it must define:

SDRAM_DEFAULT_IMPLEMENTATION

It can also be used to override the default values specified in

- IMPL/sdram_config_IMPL.h
- IMPL/sdram_geometry_IMPL.h
- sdram_commands_IMPL.h

where IMPL is the SDRAM implementation to be overridden. These files can set the following defines:

### 3.1.1 Implementation Specific Defines

When overriding one of these defines a suffix of `_IMPL` needs to be added. For example, to override `SDRAM_CLOCK_DIVIDER` to 2 for the `PINOUT_V1_IS42S16100F` target add the line:

```
#define SDRAM_CLOCK_DIVIDER_PINOUT_V1_IS42S16100F 2
```

to `sdram_conf.h`.

**SDRAM_REFRESH_MS**
This specifies that during a period of SDRAM_REFRESH_MS milliseconds a total of SDRAM_REFRESH_CYCLES refresh instructions must be issued to maintain the contents of the SDRAM.

**SDRAM_REFRESH_CYCLES**
As above.

**SDRAM_ACCEPTABLE_REFRESH_GAP**
This define specifies how long the sdram_server can go between issuing bursts of refreshes. The SDRAM server issues refreshes in bursts when it is not servicing a read/write command. The number of refresh commands for a burst is automatically calculated, hence, if a read or write command is being serviced when a refresh burst should start then it will wait until the service is over then increase its burst size appropriately. If set above SDRAM_REFRESH_CYCLES then the SDRAM will fail. The default is (SDRAM_REFRESH_CYCLES/8). The unit is given in refresh periods. For example, the value would mean that the SDRAM is allowed to go SDRAM_REFRESH_MS/SDRAM_REFRESH_CYCLES*N milliseconds before refreshing. The larger the number (up to SDRAM_REFRESH_CYCLES) the smaller the constant time impact but the larger the overall impact.

**SDRAM_CMDS_PER_REFRESH**
This defines the minimum time between refreshes in SDRAM Clk cycles. Must be in the range from 2 to 4 inclusive.

**SDRAM_EXTERNAL_MEMORY_ACCESSOR**
This defines if the memory is accessed by another device(other than the xCORE). If not defined then faster code will be produced.

**SDRAM_CLOCK_DIVIDER**
Set SDRAM_CLOCK_DIVIDER to divide down the reference clock to get the desired SDRAM Clock. The reference clock is divided by 2*SDRAM_CLOCK_DIVIDER.

**SDRAM_MODE_REGISTER**
This defines the configuration of the SDRAM. This is the value to be loaded into the mode register.

### 3.1.2 SDRAM Geometry Defines

These are implementation specific.

**SDRAM_ROW_ADDRESS_BITS**
This defines the number of row address bits.

**SDRAM_COL_ADDRESS_BITS**
This defines the number of column address bits.

**SDRAM_BANK_ADDRESS_BITS**
This defines the number of bank address bits.

**SDRAM_COL_BITS**
This defines the number of bits per column, i.e. the data width. This should only be changed if an SDRAM of bus width other than 16 is used.

### 3.1.3 SDRAM Commands Defines

These are non-implementation specific.

**SDRAM_ENABLE_CMD_WAIT_UNTIL_IDLE**
Enable/Disable the wait until idle command.

**SDRAM_ENABLE_CMD_BUFFER_READ**
Enable/Disable the buffer read command.

**SDRAM_ENABLE_CMD_BUFFER_WRITE**
Enable/Disable the buffer write command.

**SDRAM_ENABLE_CMD_FULL_ROW_READ**
Enable/Disable the full row read command.

**SDRAM_ENABLE_CMD_FULL_ROW_WRITE**
Enable/Disable the full row write command.

**SDRAM_ENABLE_CMD_COL_WRITE**
Enable/Disable the col write command.

These defines switch commands on and off in the server and client. Set to 0 for disable, set to 1 for enable. Disabling unused commands will cause a code size decrease.

### 3.1.4 Port Config

The port config is given in `\IMPL\sdram_ports_IMPL.h` and is implementation specific.

## 3.2 SDRAM API

These are the functions that are called from the application and are included in `sdram.h`.

### 3.2.1 Server Functions

## 3.3 C and xC Interface

```
void sdram_server(chanend c_client,
                  struct sdram_ports_PINOUT_V1_IS42S16400F &ports)
```

The SDRAM server thread.

This function has the following parameters:

| | |
|---|---|
| `c_client` | The channel end connecting the application to the server |
| `ports` | The structure carrying the SDRAM port details. |

## 3.4 xC Interface

```
void sdram_wait_until_idle(chanend c_server, unsigned buffer[])
```

Function to wait until the SDRAM server is idle and ready to accept another command.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server |
| `buffer[]` | The buffer where the data was written or read from in the previous command. |

```
void sdram_buffer_write(chanend c_server,
                        unsigned bank,
                        unsigned start_row,
                        unsigned start_col,
                        unsigned width_words,
                        unsigned buffer[])
```

Used to write an arbitrary sized buffer of data to the SDRAM.

Note: no buffer overrun checking is performed.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server. |
| `bank` | The bank number in the SDRAM into which the buffer of data should be written. |
| `start_row` | The starting row number in the SDRAM into which the buffer of data should be written. |
| `start_col` | The starting column number in the SDRAM into which the buffer of data should be written. |
| `width_words` | The number of words to be written to the SDRAM. |
| `buffer[]` | The buffer where the data will be read from. |

```
void sdram_full_row_write(chanend c_server,
                          unsigned bank,
                          unsigned row,
                          unsigned buffer[])
```

Used to write a full row of data from a buffer to the SDRAM.

Note: no buffer overrun checking is performed. Full row accesses are always begin aligned to coloumn 0.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server |
| `bank` | The bank number in the SDRAM into which the buffer of data should be written |

| | |
|---|---|
| `row` | The row number in the SDRAM into which the buffer of data should be written. |
| `buffer[]` | The buffer where the data will be read from. |

```
void sdram_buffer_read(chanend c_server,
                       unsigned bank,
                       unsigned start_row,
                       unsigned start_col,
                       unsigned width_words,
                       unsigned buffer[])
```

Used to read to an arbitrary size buffer of data from the SDRAM.

Note: no buffer overrun checking is performed.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server |
| `bank` | The bank number in the SDRAM from which the SDRAM data should be read. |
| `start_row` | The starting row number in the SDRAM from which the SDRAM data should be read. |
| `start_col` | The starting column number in the SDRAM from which the SDRAM data should be read. |
| `width_words` | The number of words to be read from the SDRAM. |
| `buffer[]` | The buffer where the data will be written to. |

```
void sdram_full_row_read(chanend c_server,
                         unsigned bank,
                         unsigned row,
                         unsigned buffer[])
```

Used to read a full row of data from a buffer to the SDRAM.

Note: no buffer overrun checking is performed. Full row accesses are always begin aligned to coloumn 0.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server. |
| `bank` | The bank number in the SDRAM from which the SDRAM data should be read. |

| | |
|---|---|
| `row` | The row number in the SDRAM from which the SDRAM data should be read. |
| `buffer[]` | The buffer where the data will be written to. |

## 3.5 C Interface

```
void sdram_wait_until_idle_p(chanend c_server, intptr_t buffer)
```

Function to wait until the SDRAM server is idle and ready to accept another command.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server |
| `buffer` | A pointer to the buffer where the data was written or read from in the previous command. |

```
void sdram_buffer_write_p(chanend c_server,
                          unsigned bank,
                          unsigned start_row,
                          unsigned start_col,
                          unsigned width_words,
                          intptr_t buffer)
```

Used to write an arbitrary sized buffer of data to the SDRAM.

Note: no buffer overrun checking is performed.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting the application to the server. |
| `bank` | The bank number in the SDRAM into which the buffer of data should be written. |
| `start_row` | The starting row number in the SDRAM into which the buffer of data should be written. |
| `start_col` | The starting column number in the SDRAM into which the buffer of data should be written. |
| `width_words` | The number of words to be written to the SDRAM. |
| `buffer[]` | The buffer where the data will be read from. |

```
void sdram_full_row_write_p(chanend c_server,
```

```
unsigned bank,
unsigned row,
intptr_t buffer)
```

Used to write a full row of data from a buffer to the SDRAM.

Note: no buffer overrun checking is performed. Full row accesses are always begin aligned to coloumn 0.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting the application to the server |
| bank | The bank number in the SDRAM into which the buffer of data should be written |
| row | The row number in the SDRAM into which the buffer of data should be written. |
| buffer | A pointer to the buffer where the data will be read from. |

```
void sdram_buffer_read_p(chanend c_server,
                         unsigned bank,
                         unsigned start_row,
                         unsigned start_col,
                         unsigned width_words,
                         intptr_t buffer)
```

Used to read to an arbitrary size buffer of data from the SDRAM.

Note: no buffer overrun checking is performed.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting the application to the server |
| bank | The bank number in the SDRAM from which the SDRAM data should be read. |
| start_row | The starting row number in the SDRAM from which the SDRAM data should be read. |
| start_col | The starting column number in the SDRAM from which the SDRAM data should be read. |
| width_words | The number of words to be read from the SDRAM. |
| buffer | A pointer to the buffer where the data will be written to. |

```
void sdram_full_row_read_p(chanend c_server,
                           unsigned bank,
                           unsigned row,
                           intptr_t buffer)
```

Used to read a full row of data from a buffer to the SDRAM.

Note: no buffer overrun checking is performed. Full row accesses are always begin aligned to coloumn 0.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting the application to the server. |
| bank | The bank number in the SDRAM from which the SDRAM data should be read. |
| row | The row number in the SDRAM from which the SDRAM data should be read. |
| buffer | A pointer to the buffer where the data will be written to. |

## 3.6 C and xC Interface

```
void sdram_col_write(chanend c_server,
                     unsigned bank,
                     unsigned row,
                     unsigned col,
                     short data)
```

Used to write a single column of data to the SDRAM.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting the application to the server |
| bank | The bank number in the SDRAM into which the data should be written |
| row | The row number in the SDRAM into which the data should be written. |
| col | The col number in the SDRAM into which the data should be written. |
| data | A short of data to write to the SDRAM. |

```
void sdram_shutdown(chanend c_server)
```

Shutdown the SDRAM server.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting the application to the server |

## 3.7 SDRAM Memory Mapper API

These are the functions that are called from the application and are included in `sdram_memory_mapper.h`.

### 3.7.1 Server Functions

## 3.8 xC Interface

```
void mm_read_words(chanend c_server,
                   unsigned address,
                   unsigned words,
                   unsigned buffer[])
```

Reads words from the SDRAM server on the end of the channel provided.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting to the SDRAM server. |
| `address` | The virtual byte address of where the read will begin from. |
| `words` | The count of words to be read |
| `buffer[]` | The buffer where the data will be written to. |

```
void mm_write_words(chanend c_server,
                    unsigned address,
                    unsigned words,
                    unsigned buffer[])
```

Writes words to the SDRAM server on the end of the channel provided.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting to the SDRAM server. |
| `address` | The virtual byte address of where the write will begin from. |
| `words` | The count of words to be written. |
| `buffer[]` | The buffer where the data will be written to. |

```
void mm_wait_until_idle(chanend c_server, unsigned buffer[])
```

Returns when the SDRAM server is in the idle state.

This function has the following parameters:

| | |
|---|---|
| `c_server` | The channel end connecting to the SDRAM server. |

buffer[] The buffer which the last command was performed on.

## 3.9 C Interface

```
void mm_read_words_p(chanend c_server,
                     unsigned address,
                     unsigned words,
                     intptr_t buffer)
```

Reads words from the SDRAM server on the end of the channel provided.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting to the SDRAM server. |
| address | The virtual byte address of where the read will begin from. |
| words | The count of words to be read |
| buffer | A pointer to the buffer where the data will be written to. |

```
void mm_write_words_p(chanend c_server,
                      unsigned address,
                      unsigned words,
                      intptr_t buffer)
```

Writes words to the SDRAM server on the end of the channel provided.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting to the SDRAM server. |
| address | The virtual byte address of where the write will begin from. |
| words | The count of words to be written. |
| buffer | A pointer to the buffer where the data will be written to. |

```
void mm_wait_until_idle_p(chanend c_server, intptr_t buffer)
```

Returns when the SDRAM server is in the idle state.

This function has the following parameters:

| | |
|---|---|
| c_server | The channel end connecting to the SDRAM server. |
| buffer | A pointer to the buffer which the last command was performed on. |

# 4 Programming Guide

IN THIS CHAPTER



This section provides information on how to program applications using the SDRAM module.

## 4.1 SDRAM Default implementation

- PINOUT_V2_IS42S16400F - This corresponds to the ISSI part IS42S16400F in a 21 pin configuration.
- PINOUT_V1_IS42S16400F - This corresponds to the ISSI part IS42S16400F in a 20 pin configuration.
- PINOUT_V1_IS42S16160D - This corresponds to the ISSI part IS42S16160D in a 20 pin configuration.
- PINOUT_V0 - This is for a legacy 22 pin configuration.

See the individual `port.h` files to find the port configurations.

## 4.2 Single SDRAM Support

For a application with a single SDRAM the default implementation should be set. If it is not set then the explicit `sdram_server` and `sdram_ports` must be used. The same applied for all the implementation specific defines.

## 4.3 Multiple Homogeneous SDRAM Support

For a application with a single SDRAM the default implementation should be set. For example, to drive two IS42S16400F parts, set the `SDRAM_DEFAULT_IMPLEMENTATION` to `PINOUT_V1_IS42S16400F` then the following will create the servers:

```
chan c,d;
par {
        sdram_server(c, ports_0);
        sdram_server(d, ports_1);
        app_0(c);
        app_1(d);
}
```

and the ports for the above would have been created by:

```
struct sdram_ports ports_0 = {
        XS1_PORT_16A,
        XS1_PORT_1B,
        XS1_PORT_1G,
        XS1_PORT_1C,
        XS1_PORT_1F,
        XS1_CLKBLK_1
};
struct sdram_ports ports_1 = {
        XS1_PORT_16B,
        XS1_PORT_1J,
        XS1_PORT_1I,
        XS1_PORT_1K,
        XS1_PORT_1L,
        XS1_CLKBLK_1
};
```

## 4.4 Multiple Heterogeneous SDRAM Support

It is possible for the application to drive multiple heterogeneous SDRAM devices simultaneously. In this case each `sdram_server` and `sdram_ports` usage must be explicit to the implementation. For example, to drive an IS42S16400F part and an IS42S16160D part, then the following will create the servers:

```
chan c,d;
par {
        sdram_server_PINOUT_V1_IS42S16400F(c, ports_0);
        sdram_server_PINOUT_V1_IS42S16160D(d, ports_1);
        app_0(c);
        app_1(d);
}
```

and the ports for the above would have been created by:

```
struct sdram_ports_PINOUT_V1_IS42S16400F ports_0 = {
        XS1_PORT_16A,
        XS1_PORT_1B,
        XS1_PORT_1G,
        XS1_PORT_1C,
        XS1_PORT_1F,
        XS1_CLKBLK_1
};
struct sdram_ports_PINOUT_V1_IS42S16160D ports_1 = {
        XS1_PORT_16B,
        XS1_PORT_1J,
        XS1_PORT_1I,
        XS1_PORT_1K,
        XS1_PORT_1L,
        XS1_CLKBLK_1
};
```

## 4.5 Notes

The sdram_server and application must be on the same tile.

## 4.6 Source Code Structure

### 4.6.1 Directory Structure

A typical SDRAM application will have at least three top level directories. The application will be contained in a directory starting with app_, the sdram module source is in the module_sdram directory and the directory module_xcommon contains files required to build the application.

```
app_[my_app_name]/
module_sdram/
module_xcommon/
```

Of course the application may use other modules which can also be directories at this level. Which modules are compiled into the application is controlled by the USED_MODULES define in the application Makefile.

### 4.6.2 Key Files

The following header file contains prototypes of all functions required to use use the SDRAM module. The API is described in §3.

**Figure 1:** Key Files

| File | Description |
|---|---|
| sdram.h | SDRAM API header file |

## 4.7 Module Usage

To use the SDRAM module first set up the directory structure as shown above. Create a file in the app folder called sdram_conf.h and into it insert a define for SDRAM_DEFAULT_IMPLEMENTATION. It should be defined as the implementation you want to use, for example for the sliceKIT the following would be correct:

```
#define SDRAM_DEFAULT_IMPLEMENTATION PINOUT_V1_IS42S16160D
```

Declare the sdram_ports structure used by the sdram_server. This will look like:

```
struct sdram_ports sdram_ports = {
        XS1_PORT_16A,
        XS1_PORT_1B,
        XS1_PORT_1G,
        XS1_PORT_1C,
        XS1_PORT_1F,
        XS1_CLKBLK_1
};
```

Next create a main function with a par of both the sdram_server function and an application function, these will require a channel to connect them. For example:

```
int main() {
  chan sdram_c;
  par {
    sdram_server(sdram_c, sdram_ports);
    application(sdram_c);
  }
  return 0;
}
```

Now the application function is able to use the SDRAM server.

# 5 SDRAM Memory Mapper Programming Guide

IN THIS CHAPTER



The SDRAM memory mapper has a simple interface where to the `mm_read_words` and `mm_write_words` functions a virtual address is passed, this virtual address is mapped to a physical address and the I/O is performed there. The `mm_wait_until_idle` exists so that the application can run the I/O commands in a non-blocking manner then confirm that the command has when the `mm_wait_until_idle` returns.

## 5.1 Software Requirements

The component is built on xTIMEcomposer Tools version 12.0. The component can be used in version 12.0 or any higher version of xTIMEcomposer Tools.

# 6 Example Applications

IN THIS CHAPTER



This tutorial describes the demo applications included in the XMOS SDRAM software component. §2.1 describes the required hardware setups to run the demos.

## 6.1 app_sdram_demo

This application demonstrates how the module is used to accesses memory on the SDRAM. The purpose of this application is to show how data is written to and read from the SDRAM in a safe manner.

### 6.1.1 Getting Started

1. Plug the XA-SK-SDRAM Slice Card into the STAR slot of the sliceKIT Core Board.
2. Plug the XA-SK-XTAG2 Card into the sliceKIT Core Board.
3. Ensure the XMOS LINK switch on the XA-SK-XTAG2 is set to “off”.
4. Open `app_sdram_demo.xc` and build it.
5. run the program on the hardware.

The output produced should look like:

```
00000000      00000000
00000001      00000001
00000002      00000002
00000003      00000003
00000004      00000004
00000005      00000005
SDRAM demo complete.
```

### 6.1.2 Notes

- There are 4 SDRAM I/O commands: `sdram_buffer_write`, `sdram_buffer_read`, `sdram_full_page_write`, `sdram_full_page_read`. They must all be followed by a `sdram_wait_until_idle` before another I/O command may be issued. When the `sdram_wait_until_idle` returns then the data is now at it destination. This functionality allows the application to be getting on with something else whilst the SDRAM server is busy with the I/O.

- There is no need to explictly refresh the SDRAM as this is managed by the `sdram_server`.

## 6.2 app_sdram_testbench

This application serves as a software regression to aid implementing new SDRAM interfaces and verifying current ones. The testbench runs a series of regression tests of increasing difficulty, beginning from using a single core for the sdram_server with one core loaded progressing to all cores being loaded to simulate an xCORE under full load.

### 6.2.1 Getting Started

1. Plug the XA-SK-SDRAM Slice Card into the `STAR` slot of the sliceKIT Core Board.
2. Plug the XA-SK-XTAG2 Card into the sliceKIT Core Board.
3. Ensure the XMOS LINK switch on the XA-SK-XTAG2 is set to "off".
4. Open `app_sdram_testbench.xc` and build it.
5. run the program on the hardware.

With verbose output turned on (controlled by VERBOSE_MSG and VERBOSE_ERR), the output produced should look like:

```
Test suite begin
8 threaded test suite start
Begin sanity_check
...
```

## 6.3 app_sdram_benchmark

This application benchmarks the performance of the module. It does no correctness testing but instead tests the throughput of the SDRAM server.

### 6.3.1 Getting Started

1. Plug the XA-SK-SDRAM Slice Card into the `STAR` slot of the sliceKIT Core Board.
2. Plug the XA-SK-XTAG2 Card into the sliceKIT Core Board.
3. Ensure the XMOS LINK switch on the XA-SK-XTAG2 is set to "off".
4. Open `app_sdram_benchmark.xc` and build it.
5. run the program on the hardware.

The output produced should look like:

```
Cores active: 8
Max write: 70.34 MB/s
Max read : 66.82 MB/s
Cores active: 7
Max write: 71.47 MB/s
Max read : 68.08 MB/s
...
```