SRAM Controller

1.0 Highlights

This section of the manual contains the following topics:

- Introduction
- IO Ports
- SRAM Transaction Control Register
- SRAM Configuration Registers
- SRAM Address Register
- SRAM Modes
- Programming the SRAM interface

1.1 Introduction

The SRAM controller was designed as a software programmable hardware interface for Micron SRAM devices, with applications to general SRAM devices as well. This controller simplifies the transmission and reception of data to and from SRAMs connected to the SoC via the dedicated SRAM address and data bus lines. The Controller by handles all of the data transmission and reception protocol formalities, and SRAM configuration (done through software control). The controller gives the user the option of controlling up to 8 external SRAM devices each of which can be programmed in different modes and for different applications.

The following are some of the key features of this module:

- Can control up to 8 external SRAMs
- Supports independent device configuration
- Software programmable port configuration
- Supports Page Mode, Burst Mode and Asynchronous Mode
- Pipelined data transmissions and reception design
- Static synchronous design
- Fully synthesizable

1.2 Reference Material

1. Async/Page/Burst CellularRAMTM 1.5 reference manual.

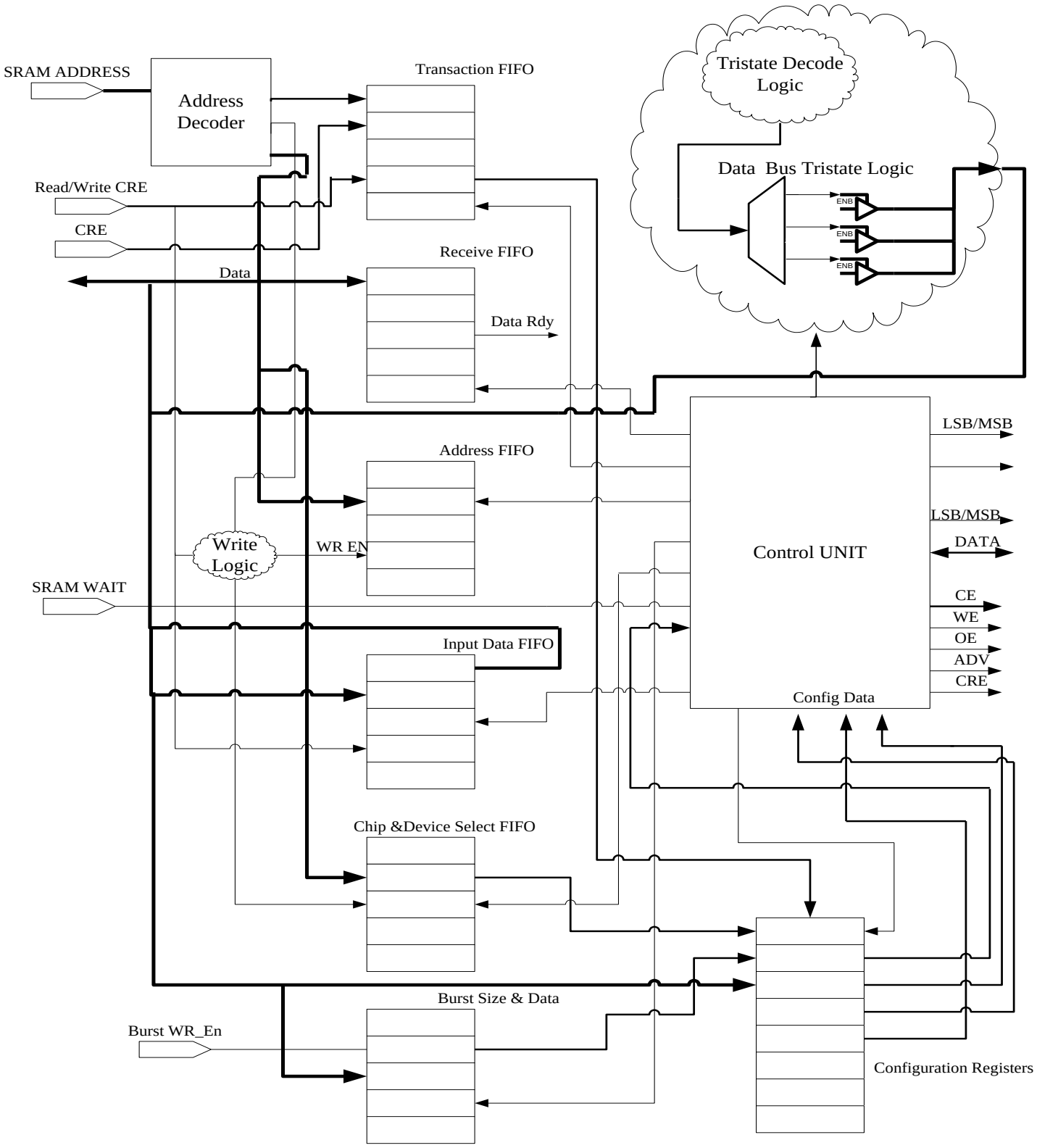
1.3 Terms and Definitions

Table 1-1

Term	Definition
SRAM	Static Random Access Memory
X	Don’t Care

Transaction Queue	SRAM's Transaction Control Queue. A FIFO that holds all of the available transactions for processing.

SRAM Controller



1.4 I/O ports

Table 1-2

Port	Width	Direction	Description
RST	1 bit	Input	Asynchronous reset active high
NV_RAM_CLK	1 bit	Input	This clock signal is used to synchronize the external RAM to the system operating frequency during synchronous operations. NV_RAM_CLK is held LOW during asynchronous read and write operations and also during Page Mode operations.
CLK	1 bit	Input	This clock input represents the master clock which the entire module runs off of. This clock must be at least four times the frequency of the NV_RAM_CLK for the controller to perform to spec.
SRAM_READ	1 bit	Input	The SRAM read input pin enables a read transaction. When asserted high in conjunction with the SRAM chip select, the module latches the read data targeted by the address decoder onto the SRAM_RDDATAport.
SRAM_WRITE	1 bit	Input	The SRAM write input pin enables a write transaction. When asserted high in conjunction with SRAM_CS the module latches the data on the SRAM _WRDATA port into the register or sub-module targeted by the address decoder.
SRAM_CS	1 bit	Input	The SRAM Chip Select input is active high and used in conjunction with the read and write strobes to enable register read and write operations.
SRAM_WRDATA	32 bits	Input	SRAM module data input port.
SRAM_ADDRESS	32 bits	Input	This input is used to decode the internal addresses for SRAM Controller and the external SRAM Memory Arrays. Therefore special consideration must be taken at the system level when addressing internal SRAM controller registers, and FIFOs. Bits 31:28 are reserved for use by the system architect. The SRAM controller does not examine, store or use these bits for any of its internal processes. Bits 27:24 are used by the controller to address its internal configuration registers, and FIFOs. Bits 22:0 of the address port are used by the controller to address the memory space in the external SRAM exclusively. These bits are stored in the internal “Address FIFO” and used during RAM transactions to address SRAM’s memory

			array.
SRAM_WAIT	1 bit	Input	This input provides throttling control for the slave device during burst read and write operations. The signal is gated by CE. WAIT is used to arbitrate collisions between refresh and read/write operations. WAIT is also asserted at the end of a row unless the device is configured to allow wrapping within the burst length. WAIT should be ignored during asynchronous and page mode operations. WAIT is High-Z when CE is deasserted.
RAM_ADDRESS	22 bits	Output	This port is the external address output port for all SRAM device connected to the SoC.
DATA_RDY	1 bit	Output	This bit indicates that there is data present in the Read Data FIFO.
DATA_VALID	1 bit	Output	The bit indicates that the data on the output data port is valid. The bit is pulsed high for one "Master Clock" clock cycle.
SRAM_BUSY	1 bit	Output	This bit indicates that the controller's transaction FIFO and or the Write Data FIFO are full and no more transactions can be issued until there is room in the FIFOs.
ADDRESS_ERROR	1 bit	Output	This bit indicates that a transaction has been issued to a device which is outside of the address range of the devices configured on the 8 available ports.
Read FIFO Empty	1 bit	Output	When asserted this bit indicates that the Read Data FIFO is empty.
SRAM_CLK	1 bit	Output	Output clock to SRAM devices.
CE	8 bits	Output	The Chip Enable activates the targeted device when LOW. When CE is deasserted, the device goes into standby or deep power-down mode.
WE	1 bit	Output	This output signal is used by the external RAM to determine if the current transaction is read or a write. If the WE signal is asserted (low) the targeted device prepares for a write operation.
OE	1 bit	Output	This output signal is used by the external SRAM devices to enable the output buffers. When OE is deasserted, the output buffers are disabled.
CRE	1 bit	Output	The Control Register Enable (CRE). When CRE is asserted, a write operation will load input data into the RCR or BCR registers, and read operations latch the RCR, BCR, or DIDR registers onto the output port.
ADV	1 bit	Output	Address valid indicates the address is present on the address input port is stable and can be latched. Addresses can be latched on the rising edge of
LSB_MSB	2 bits	Output	Lower byte enable. DQ[7:0] Upper byte enable. DQ[15:8]

SRAM_RDDATA	16 bits	Output	Read Data output port to the SoC.
DATA	16 bits	InOut	Data inputs/outputs from and to the external SRAM devices.

2.0 Functional Description & Modes

The SRAM controller was designed to handle communication between industry standard SRAM devices and a MCU. The SRAM Controller is capable of communicating with up to 8 external SRAM devices independently. The controller's 8 separate virtual ports can each be configured to an individual mode, corresponding to the mode of the SRAM device attached to that port. Subsequently this gives the system engineer the flexibility to configure each SRAM within the system to a different mode for different applications, while still communicating to all SRAMs through a single controller.

The controller supports two types of transactions; those are read and write transactions. All read and write transactions to internal configuration registers and the external SRAM devices are stored in the transaction queue, with the except of a read of the Status Register. Software must issue a transaction to the transaction queue for processing to read or write the controller's internal registers and FIFOs, and also to initiate external transactions with the SRAM devices attached to the controllers 8 ports. All "write data" transactions must be accompanied by a write to the Data FIFO to initiate the start of the write transactions.

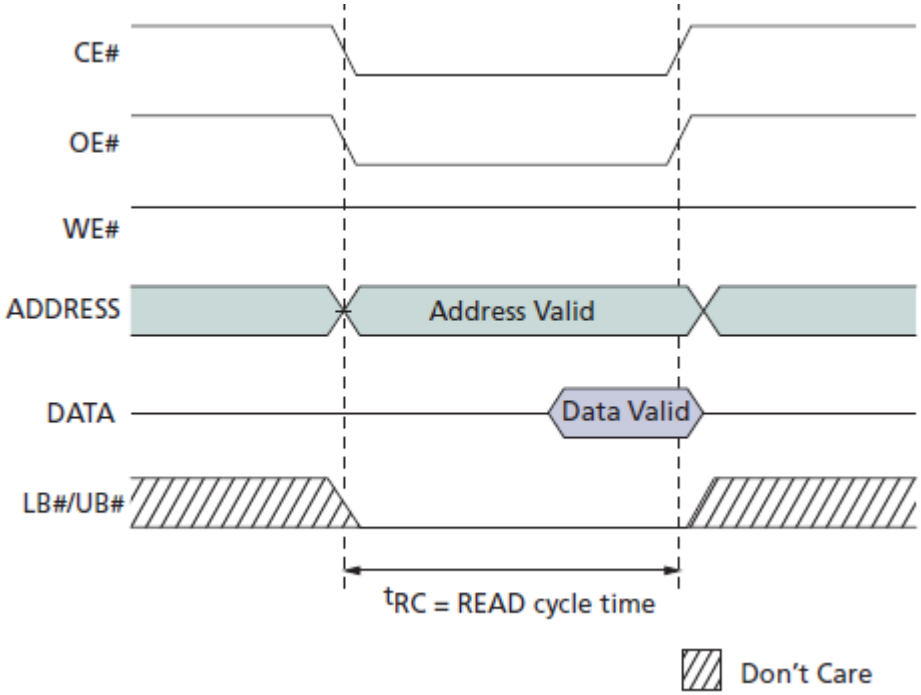
The amount of data written to the Write Data FIFO depends on how the targeted port has been configured. For example if the port is configured for burst mode the amount of data written to the Write Data FIFO should equal the value stored in the Burst Size Configuration Register for that port.

As mentioned before each port can be configured to a different mode, the available configuration modes are Asynchronous, Page and Burst mode (for Micron CellularRAM). These modes each offer advantages in performance depending on the application. A brief description and a timing diagram for each mode is given below.

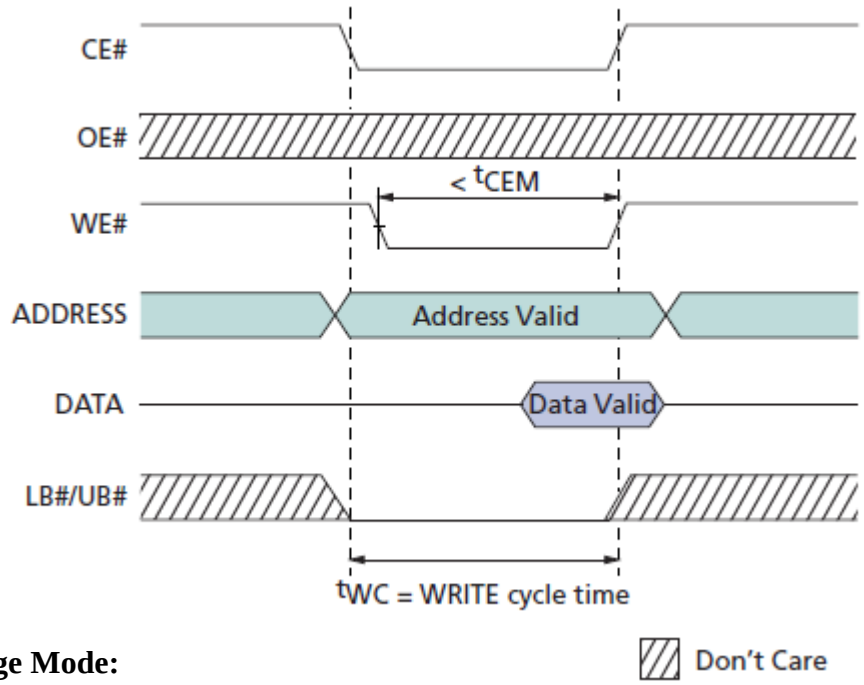
2.1 Asynchronous Mode:

In asynchronous mode the SRAM controller communicates with the external RAM asynchronously as the name suggests. As there is no clock signal to synchronize the two devices the controller uses the values stored in the port's read and write cycle time configuration registers to synchronize read and write operations. The SRAM's device specification should be examined to determine the appropriate read and write cycle time for asynchronous mode. All transaction done in asynchronous mode must be accompanied by a valid memory array address. The timing diagrams shown below illustrate a read and a write operation in asynchronous mode.

READ Operation (ADV# LOW)



WRITE Operation (ADV# LOW)

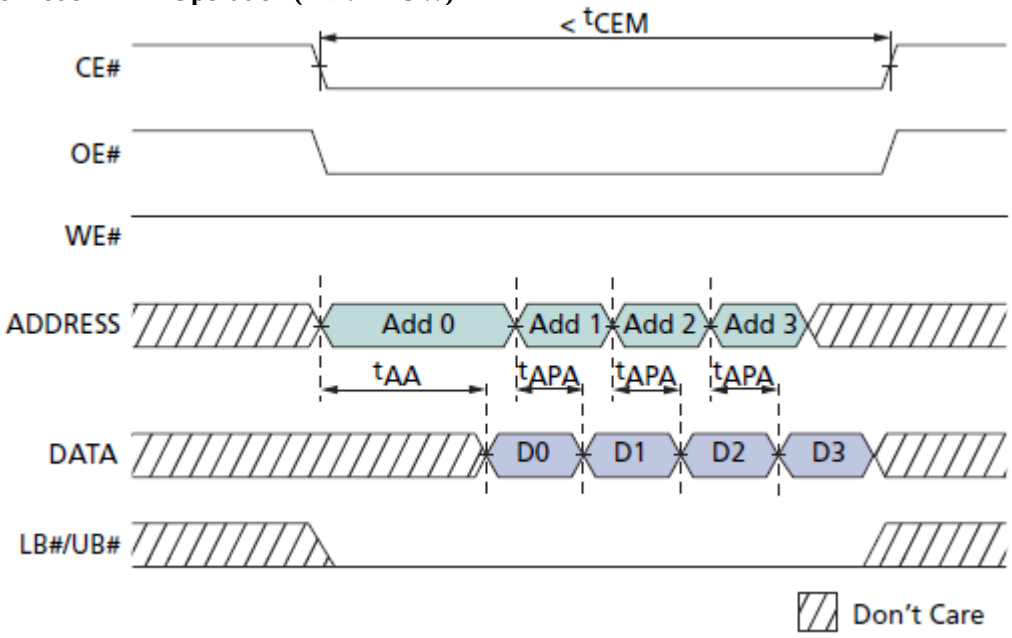


2.2 Page Mode:

Page mode is another form of asynchronous communication between the controller and the SRAM. Page mode differs from asynchronous mode in several respects, the first of which is that all page mode operations are read only operations and the second is that page mode operations are always a burst of 4. Another difference is that unlike asynchronous mode, page mode only requires one initial start address to retrieve the first data value, all subsequent data values are retrieved from increments of the initial address.

There are special considerations that must be taken when configuring a port for page mode, those are initial read cycle time and subsequent read cycle time. There is an initial latency period at the start of all operations for a device configured in page mode, during this period data is not valid. After this period expires the first data value is valid, and the subsequent values each become valid after a fixed amount of time. The controller uses the initial read cycle time and the subsequent read cycle time registers to synchronize the capture of data when configured in page mode. The timing diagram below shows an example of a page mode read operation.

Page Mode READ Operation (ADV# LOW)



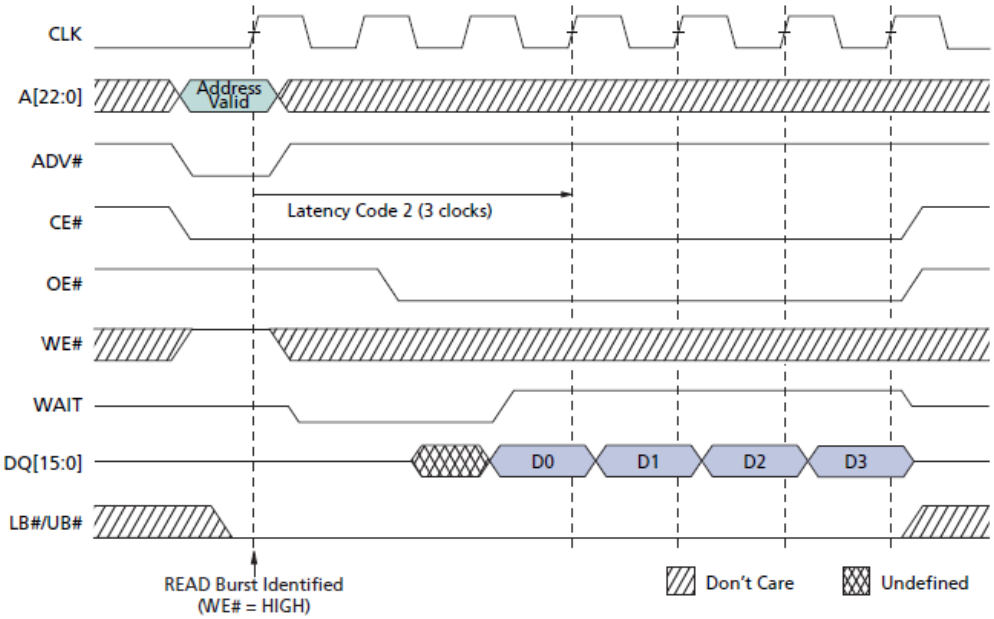
2.3 Burst Mode:

Burst mode is a synchronous form of communication where multiple values data are exchanged. Like page mode all burst transactions are delayed by an initial amount, called "Initial Burst Latency." However as burst mode operations are synchronous operations, the initial burst latency is defined in SRAM clock cycles and not units of time. All subsequent data values are valid on the next rising edge of the SRAM clock. Addressing of subsequent data is handled as it is in page mode for both read and write transactions.

Occasionally the SRAM device may require some additional time to produce or consume the next data value; this is an indeterminate amount of time and is signaled by the SRAM's WAIT pin. The SRAM Controller supports an active high WAIT signal and will wait to trap or change the data value if this signal goes high during a burst transaction. There are two timing diagrams shown below which illustrate burst read and write transactions.

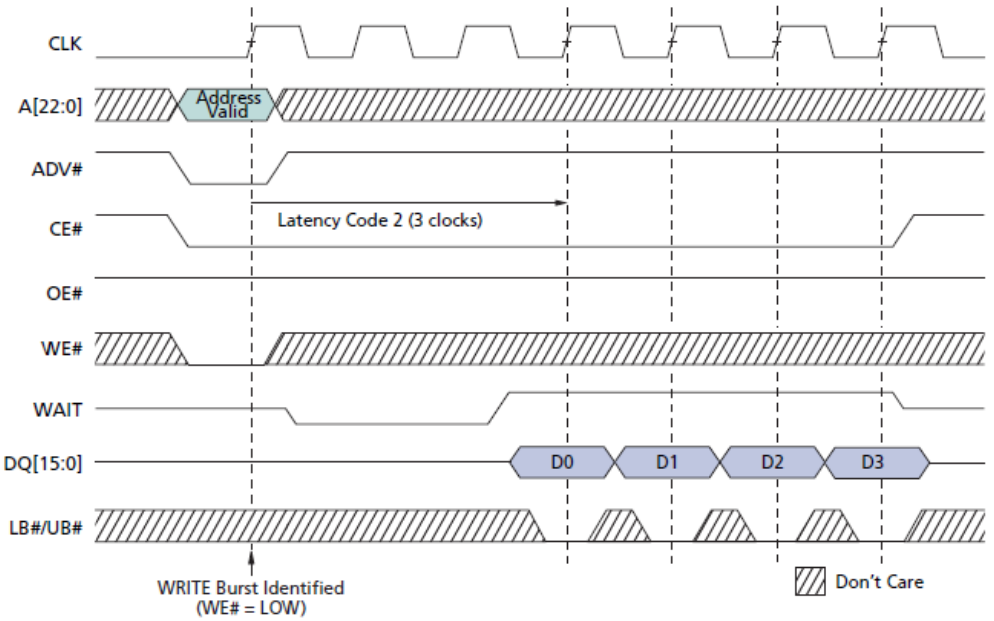
***Note**: Please consult the SRAM manufacture's device specification for the appropriate initial burst latency clock cycles for your device.*

Burst Mode READ (4-word burst)



Notes: 1. Non-default BCR settings for burst mode READ (4-word burst): Fixed or variable latency; latency code two (three clocks); WAIT active LOW; WAIT asserted during delay. The diagram above is representative of variable latency with no refresh collision or fixed-latency access.

Burst Mode WRITE (4-word burst)



Notes: 1. Non-default BCR settings for burst mode WRITE (4-word burst): Fixed or variable latency; latency code two (three clocks); WAIT active LOW; WAIT asserted during delay.

3.0 Memory Mapped Registers

The SRAM module has memory mapped registers that are accessible by software through the I/O ports detailed in section 1.4. It is worth mentioning that the memory mapped registers within the SRAM controller are addressed using bits 27:24 of the address port, as explained in section 1.4. Below is a table listing all of the software accessible registers. The details of each of these registers are explained in this section.

Table 3-1

Relative Offset	Register Name	Size	Read Write
0xX0000000	Transaction Control Queue	16 bits	W
0xX0100000	Write Data FIFO	16 bits	W
0xX0200000	Read Data FIFO	16 bits	R
0xX0300000	Status Register	32 bits	R
0xX0400000	Reset Overflow Flag Register	1 bit	W

3.1 Transaction Control Register

The SRAM controller module has one Transaction Control Queue, and each entry into this queue represents a single transaction control register for a single transaction. This register tells the controller the detail about the transaction to be processed in the next operation.

The table below summarizes the SRAM controller Transaction Control register.

Table 3-2

Bits	Field Name	Description
31:13	-	Reserved.
12:9	Byte Enables	This field holds the byte enables for each transaction. The controller uses this field to enable and disable the upper and lower byte enables on the external SRAM, as dictated by this field.
10:8	Port Select	This bit field is used to decode the virtual port to which the current transaction targets. Each virtual port has its own set of configuration registers and an individual SRAM device attached to it. The SRAM controller can support up to 8 individual ports, all of which can be configured and maintained independently of each other.
7:3	Storage Unit Select	This bit field is used to decode the targeted storage unit. The available storage units are internal configuration registers, and the external SRAMs and their internal configuration registers and memory arrays. A decoding for the storage units is shown below. 0x00. Burst Size Configuration Register 0x01. Read Cycle Time Configuration Register 0x03. Write Cycle Time Configuration Register 0x03. Burst Initial Latency Time Configuration register 0x04. Initial Read Cycle Time Configuration Register 0x05. Reserved.

		0x06. Subsequent Read Cycle Time Configuration Register 0x07. Reserved 0x08. The SRAM Controller's Mode Register 0x09. The External SRAM's Configuration Register 0x0A. The External SRAM's Memory Array
2	Control Register Enable	The CRE bit indicates to the controller that this operation targets the external SRAM's control register. It has to be used in conjunction with the "Storage Unit Select" field set to 0x09. For read transactions the SRAM Read bit should be set to 1 and the SRAM Write bit should set to 0. Vice versa for write transactions. *Two user-accessible configuration registers define the SRAM device operation. The BCR defines how the CellularRAM interacts with the system memory bus and is nearly identical to its counterpart on burst mode Flash devices. The RCR is used to control how refresh is performed on the DRAM array. These registers are automatically loaded with default settings during power-up, and can be updated any time the devices are operating in a standby state.* *A DIDR provides information on the device manufacturer, CellularRAM generation, and the specific device configuration. The DIDR is read-only.* For more information on the CRE enable bit and the internal device configuration register see the micron CellularRAM documentation spec.
1	SRAM Read	This bit indicates to the controller that this transaction is to be a read. Multiple reads can be issued to the controller however the data is not usually present instantly so software should monitor the data ready bit of the SRAM controller's status register.
0	SRAM Write	This bit indicates to the controller that this transaction is to be a write.

3.2 Write Data FIFO

Table 3-3

Bits	Description	Read/Write
31:16	Reserved.	-
15:0	This register holds the write data value for an SRAM write transaction.	W

3.3 Read Data FIFO

Table 3-4

Bits	Description	Read/Write
31:16	Reserved.	-
15:0	This register holds the read data value for an SRAM read transaction.	R

3.4 Status Register

Table 3-5

Bits	Field Name	Description	Read/Write

31:4	-	Reserved.	-
3	RDFFIFOFL	This bit indicates that the Read Data FIFO is full	R
2	RDFFIFOVRFL	This bit indicates that the Read Data FIFO has overflowed.	R
1	RDFFIFOEMPTY	The bit indicates that the Read Data FIFO is empty	R
0	BUSY	This bit indicates that SRAM controller is busy and cannot accept any new transactions.	R

3.5 Configuration Registers

The configuration registers allow software to configure each port to meet the timing and burst size requirements of the specific SRAM connected to that port. These register are accessed via the "Storage Unit Select" field of the transaction control register.

The table below shows the configuration registers with the select encoding, bit width and a short description.

Table 3-6

Select Encoding	Register	Size	Description
0x0C	Chip Enable Setup Time.	7 bits	This register holds the setup time value used by the controller to determine exactly when to assert the chip enable so to not violate the setup CE time of the SRAM.
0x0B	Address Valid Setup Time	7 bits	This register holds the address setup time value that the controller uses to wait before asserting the ADV line. The value is measured in SRAM clock cycles.
0x0A	External SRAM		The External SRAM's Memory Array.
0x09	External SRAM Configuration Register		The External SRAM's Configuration Register.
0x08	SRAM Mode Register	2 bits	This register allows software to configure the controller's port in one of four modes. The port's mode tells the controller how to communicate with the external SRAM device. The configuration decode is shown below. "00" Asynchronous Mode "01" Page Mode "10" Burst Mode "11" NOT valid
0x07	-	-	Reserved.
0x06	Subsequent Read Cycle Time	7 bits	This register holds the subsequent read cycle time in SRAM clock cycles. The subsequent read cycle time is the minimum amount of time it will take a device configured in page mode to output the data after the first initial data word

			has been latched onto the data bus.
0x05	-	-	Reserved.
0x04	Initial Read Cycle	7 bits	This register holds the initial read cycle time in SRAM clock cycles. The initial read cycle time is the minimum amount of time it will take a device configured in page mode to output data once the address has been latched internally.
0x03	Burst Initial Latency	7 bits	This register holds the initial latency count value for a burst transaction. *The latency count stored in the BCR defines the number of clock cycles that elapse before the initial data value is transferred between the processor and CellularRAM device. The initial latency for READ operations can be configured as fixed or variable (WRITE operations always use fixed latency). Variable latency enables the CellularRAM to be configured for minimum latency at high clock frequencies, but the controller must monitor WAIT to detect any conflict with refresh cycles.*
0x02	Write Cycle Time	7 bits	This register holds the write cycle time in SRAM clock cycles. The write cycle time is the minimum amount of time it will take the device to latch the data once the address has been latched internally.
0x01	Read Cycle Time	7 bits	This register holds the read cycle time in SRAM clock cycles. The read cycle time is the minimum amount of time it will take the device to return data once the address has been latched internally.
0x00	Burst Size	7 bits	This register holds the burst size value used to determine how many read and write operations will take place during a read or write transaction. The data width is dependent upon the SRAM device, therefore size refers to how many reads or writes will be performed, not how many bytes will be read or written during the transaction.

3.6 Read Data FIFO

Table 3-4

Bits	Description	Read/Write
31:1	Reserved.	-
0	This bit resets the read FIFO overflow error flag.	R

4.0 Programming the SRAM Controller

The SRAM controller has a total of eight available ports which the system architect can attach eight separate SRAM devices. Each port must be configured independently to match the timing specification and configuration of the SRAM attached to that particular port. There are a total of ten configuration register used to configure each port to a particular mode and according to the specific timing constraints of the SRAM attached to that port.

4.1 Write Transactions

All write transactions require at least two steps.

1. Write the transaction to the Transaction Queue.
    a. Target the correct port
    b. Target the correct storage unit
    c. Set write bit to 1 and read to 0

2. Write the data value(s) to the Write Data FIFO.

4.2 Read Transactions

All read transactions (with the exception of a read from the Status Register) require at least two steps, shown as step 1 and 3, the second step listed is optional but recommended.

1. Write the transaction to the Transaction Queue.
    a. Target the correct port
    b. Target the correct storage unit
    c. Select a read bit to 1 and write to 0

2. Read the Status Register to determine if the read data is available.
3. Read the read data from the Read Data FIFO.

4.3.1 Reading the Status Register

1. Read the data at address 0x03000000

4.3 Configuring the External SRAM

The SRAM Controller has all of its ports set to Asynchronous Mode at power on reset. Use the following procedure to change the external SRAM device's mode.

1. Ensure that the address value reflects the chosen mode for the SRAM device.

2. Issue a CRE Register write for the targeted device to the transaction queue, using the address defined above.
3. Issue a dummy write value of 1 to the Write Data FIFO to start the transaction.
4. Issue a configuration register write transaction to the "Mode Configuration Register" for the targeted port.
5. Write the mode encoded value to the Write Data FIFO.
6. Configure the appropriate latency, read, write cycle times and or burst size configuration registers for that port.

Example Program: Configuring SRAM & SRAM Controller Port 0 for Page Mode.

```
//Configure the SRAM for Page Mode
#define   sram_page_confg    (*((volatile signed long *) 0x40000090))  // Address settings for Page Mode
#define   sram_trans_fifo       (*((volatile signed long *) 0x40000000)) // TRANSACTION FIFO Pointer
#define   WRITE_MODE_CFG_REG0 0x00000081  //Mode Configuration Register for Port 0
#define   PAGE_MODE 0x00000001  //Page Mode Configuration Encoding.
 //Issue a write to the SRAM device's CRE register, to change the device mode to page mode.
* sram_page_confg  =  0x00000095; //Issue a write the CRE register for port 0
*sram_wrdata_fifo      = 0x01;//Write a dummy value to start the transaction.

//Set SRAM Controller's port 0 to page mode.
sram_trans_fifo         = WRITE_MODE_CFG_REG0//
sram_wrdata_fifo      = PAGE_MODE;

//Set the port 0's initial and subsequent read cycle time
// configurations register to appropriate values for page mode.
//Consult the device specification for the appropriate values for these registers.

#define    WRITE_SUBSE_PAGE_RD_TIME          0x00000061
#define    WRITE_INIT_RD_TIME                      0x00000041

//Issue a write to the initial Read Cycle Time Register for port zero
sram_trans_fifo         = WRITE_SUBSE_PAGE_RD_TIME;
sram_wrdata_fifo      = 1; //Write the read cycle time value base on the spec for page mode.

//Issue a write to the Page Mode Subsequent Read Cycle Time Register
sram_trans_fifo         = WRITE_SUBSE_PAGE_RD_TIME;
sram_wrdata_fifo      = 0x98; //Write the subsequent read cycle time base on the spec for page mode.
```

4.4 Asynchronous Mode Read & Write:

1. Follow the steps outlined above to put the SRAM device into Asynchronous Mode.
2. Set the Read and Write cycle times form asynchronous operations to their proper time limits. These limits should be based on the system clock frequency, consult the device specification to ensure the proper times are set.

3. Issue write and read transactions to the controller by writing to the transaction queue, using the steps outlined in section 4.1 and 4.3.

Example Program: Asynchronous mode

```
sram_trans_fifo        = WRITE_RD_TIME;
sram_wrdata_fifo       = 1; //Write read cycle time base on spec for default mode
sram_trans_fifo        = WRITE_WR_TIME;
sram_wrdata_fifo       = 1; //Write write cycle time base on spec for default mode

//Issues write to the SRAM
sram_trans_fifo        = SRAM0_WRITE;
sram_wrdata_fifo       = 0x8900;//write the data word 8900 to SRAM connect to port zero

//Issue a read to the SRAM
sram_trans_fifo        = SRAM0_Read;

//Check the Status Register
While ( (*sram_status & 0x00000010) == 0 )

//read the data from the Read Data FIFO
dummy_var  = sram_rddata_fifo;
```

4.5 Page Mode Configuration & Read Procedure

1. First configure the SRAM model for page mode transactions using the initial configuration procedure outlined in section 4.3.
2. Configure the Initial and Subsequent Read Cycle time configuration registers using the steps outline in section 4.1
3. Issue a read transaction to the transaction queue.
4. Follow the procedures outlined in section 4.2 to retrieve the data.

Example Page Mode Program:

```
#define   SRAM0_READ    0x000000A2 // SRAM READ PORT 0

sram_trans_fifo        = WRITE_RD_TIME;
sram_wrdata_fifo       = 1; //Write read cycle time base on spec for default mode
sram_trans_fifo        = WRITE_WR_TIME;
sram_wrdata_fifo       = 1; //Write write cycle time base on spec for default mode

//Configuration the SRAM for Page Mode
* sram_page_confg    =  WRITE_SRAM0_CRE; //Configuration Address for Async

//Set SRAM Controller's port 1 to page mode.
sram_trans_fifo        = WRITE_MODE_CFG_REG1//
sram_wrdata_fifo       = PAGE_MODE;

sram_trans_fifo        = WRITE_INIT_RD_TIME;
sram_trans_fifo        = WRITE_SUBSE_PAGE_RD_TIME;

sram_wrdata_fifo       = 3; //Write initial read cycle time for page mode
sram_wrdata_fifo       = 2; //Write subsequent read cycle time for page mode
```

//The Controller and SRAM is now Page Mode reads, now issue read commands at will

```
sram_trans_fifo    = SRAM0_READ;
index = 0;
page_mode_read_cnt = 4-1;
```

```
while (index != page_mode_read_cnt)
        if ((*sram_status & 0x00000010)  != ) page_data [index++] = sram_rddata_fifo;
```

4.4 SRAM Burst Mode Read &Write Procedures

1. Configure the external SRAM for burst mode transactions using the configuration procedure in section 4.3.

2. Set the Burst Initial Latency configuration register to the appropriate value.

3. Set the burst size value in the Burst Size configuration register.

4. Issue a transaction
    a. For a write transaction follow the steps described in section 4.1.
        i. The number of write data operations should be equal to the value in the Burst Size register.
    b. For a read transaction follow the steps as described in section 4.3.
        i. The number of read data operations should be equal to the value in the Burst Size register.

Example Burst Mode Program:

```
//=============================================================
//                    SRAM CONTROLLER INTERFACE
//=============================================================

#define   sram_async_confg  (*((volatile signed long *) 0x4000010)) // Address for Async Mode
#define   sram_page_confg    (*((volatile signed long *) 0x40000090)) // Address for Page  Mode
#define   sram_burst_confg    (*((volatile signed long *) 0x40081C1F)) // Address for Burst Mode

#define   sram_trans_fifo       (*((volatile signed long *) 0x40000000)) // TRANSACTION FIFO
#define   sram_wrdata_fifo     (*((volatile signed long *) 0x41000000)) // WRITE DATA FIFO
#define   sram_rddata_fifo      (*((volatile signed long *) 0x42000000)) // READ DATA

volatile char *volatile_sram_ptr = (char(*)) 0x48000000;

// WRITE CONFIGURATION REGISTER VALUES
#define    WRITE_MODE_CFG_REG                0x00000081
#define    WRITE_BURST_SIZE                  0x00000001
#define    WRITE_RD_TIME                     0x00000011
#define    WRITE_WR_TIME                     0x00000021
#define    WRITE_INIT_RD_TIME                0x00000041
#define    WRITE_INIT_WR_TIME                0x00000051
#define    WRITE_SUBSE_PAGE_WR_TIME          0x00000071
#define    WRITE_SUBSE_PAGE_RD_TIME          0x00000061
#define    WRITE_BURST_INIT_LATENCY          0x00000031

// READ CONFIGURATION REGISTER VALUES
```

```
#define    READ_MODE_CFG_REG                0x00000082
#define    READ_BURST_SIZE                  0x00000002
#define    READ_RD_TIME                     0x00000012
#define    READ_WR_TIME                     0x00000022
#define    READ_INIT_RD_TIME                0x00000042
#define    READ_INIT_WR_TIME                0x00000052
#define    READ_SUBSE_PAGE_WR_TIME          0x00000072
#define    READ_SUBSE_PAGE_RD_TIME          0x00000062
#define    READ_BURST_INIT_LATENCY          0x00000032
//WRITE CRE REGISTER VALUE
#define    WRITE_SRAM0_CRE                  0x00000095

//READ CRE REGISTER VALUE
#define    READ_SRAM0_CRE                   0x00000096

//CONSTANTS READ WRITE SRAM VALUES EXAMPLE FOR AN SRAM ON PORT 0

#define   SRAM0_WRITE                       0x000000A1 // SRAM WRITE PORT 0
#define   SRAM0_READ                        0x000000A2 // SRAM READ PORT 0

sram_index = 0;

 sram_trans_fifo = WRITE_MODE_CFG_REG;      // write transaction to the transaction fifo
 sram_trans_fifo = WRITE_RD_TIME;           // write transaction to the transaction fifo
 sram_trans_fifo = WRITE_WR_TIME;           // write transaction to the transaction fifo

 sram_wrdata_fifo = 0;                      // write transaction to the data fifo
 sram_wrdata_fifo = 7;                      // write transaction to the data fifo
 sram_wrdata_fifo = 7;                      // transaction to the data fifo

 sram_burst_confg = WRITE_SRAM0_CRE;
 sram_wrdata_fifo = 0; //write dummy value to the FIFO to ensure the transaction will start

 sram_trans_fifo = WRITE_BURST_SIZE;         //write the burst size configuration register
 sram_trans_fifo = WRITE_INIT_RD_TIME;       // issuing a write burst write cycle time register
 sram_trans_fifo = WRITE_BURST_INIT_LATENCY; // issuing a write burst init latency register
 sram_trans_fifo = WRITE_MODE_CFG_REG;       // issuing a write to the mode register

 sram_wrdata_fifo = 24;          // writting the data associated with the writes above to the data fifo
 sram_wrdata_fifo = 7;           // writting the data associated with the writes above to the data fifo
 sram_wrdata_fifo = 5;           // writting the data associated with the writes above to the data fifo
 sram_wrdata_fifo = 2;           // writting the data associated with the writes above to the data fifo

 //Course_Speed   = 200;
 altitude_dummy = 20000;
 lat_dummy              = 37371;
 lon_dummy              = 122225;

 //sram_wrdata_fifo = (Course_Speed & 0x0000FFFF);
 //sram_wrdata_fifo = (Course_Speed & 0xFFFF0000) >> 16;

 sram_wrdata_fifo = (altitude_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (altitude_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lat_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lat_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lon_dummy & 0x0000FFFF);
```

```
 sram_wrdata_fifo = (lon_dummy & 0xFFFF0000) >> 16;

altitude_dummy = 10000;
 lat_dummy                  = 38196;
 lon_dummy                  = 121517;

 sram_wrdata_fifo = (altitude_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (altitude_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lat_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lat_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lon_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lon_dummy & 0xFFFF0000) >> 16;

altitude_dummy = 5000;
 lat_dummy                  = 38417;
 lon_dummy                  = 121354;

 sram_wrdata_fifo = (altitude_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (altitude_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lat_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lat_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lon_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lon_dummy & 0xFFFF0000) >> 16;

 //Course_Speed   = 500;
 altitude_dummy = 5000;
 lat_dummy = 38683;
 lon_dummy = 121580;

sram_wrdata_fifo = (altitude_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (altitude_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lat_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lat_dummy & 0xFFFF0000) >> 16;
 sram_wrdata_fifo = (lon_dummy & 0x0000FFFF);
 sram_wrdata_fifo = (lon_dummy & 0xFFFF0000) >> 16;

  *( (int *) volatile_sram_ptr ) = SRAM0_WRITE;// ISSUE THE FIRST BURST OF 24 WORDS

 sram_trans_fifo  = WRITE_BURST_SIZE;
 sram_wrdata_fifo = 6; //CHANGE BURST SIZE TO 6
```

To prevent the system from locking up on a read the user should check the SRAM receive FIFO flag to ensure that there is data in the FIFO before issuing a read because if it is empty the system will halt until there data is available.

```
dataArray = (int *) 0x10000000;

        *( (int *) (volatile_sram_ptr + sram_index) ) = SRAM0_READ; // ISSUE THE BURST READ
        sram_index+=6;

        while ( sram_index < 24 ) {

        // check to see if the sram data is ready
        if ( (status_reg & 0x00000010) != 0 ) {
```

```
                for ( int i = 0; i <= 2; i++ ) {

                        *dataArray = sram_rddata_fifo;//Write the value to SharedRam
                        *dataArray++ = ( ( sram_rddata_fifo << 16 ) | *dataArray );
                }

                *( (int *) (volatile_sram_ptr + sram_index) ) = SRAM0_READ; // Issue Burst Read
                sram_index+=6;
        }
}
```