External Memory Interface (EMIF)

Introduction

Provides an introduction to the EMIF, the memory types it supports, and programming its configuration registers.

Learning Objectives

Outline

- Memory Maps
- Memory Types
- Programming the EMIF
- Additional Memory Topics
Chapter Topics

External Memory Interface (EMIF).................................................................................................................13-1

Memory Maps ...................................................................................................................................................13-3
Sidebar: Memory Addressing on C6x ..............................................................................................................13-4

Memory Types ..................................................................................................................................................13-5
Overview .......................................................................................................................................................13-5
Using SDRAM ................................................................................................................................................13-6
Using Asynchronous Memory .........................................................................................................................13-10

Sidebar: Optional Async Timing ....................................................................................................................13-14

Programming the EMIF .................................................................................................................................13-16
Using the EMIF with CSL ...............................................................................................................................13-16
Programming the EMIF with Assembly .........................................................................................................13-17
Programming the EMIF with GEL .................................................................................................................13-18

Additional Memory Topics .............................................................................................................................13-19
EMIF – CPU’s Access Performance ................................................................................................................13-19
Fanout ...............................................................................................................................................................13-21
Shared Memory ..............................................................................................................................................13-22
SBSRAM .........................................................................................................................................................13-24
SDRAM Optimization ....................................................................................................................................13-25
EMIF ‘C6x Family Comparison .......................................................................................................................13-25
Sidebar: C6x01 Memory Map ............................................................................................................................13-26
Memory Maps

A Memory Map is a table representation of memory...

C6713 DSK Memory Map

TMS320C6713

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000_0000</td>
<td>256KB Internal Program / Data</td>
</tr>
<tr>
<td>0180_0000</td>
<td>Peripheral Regs</td>
</tr>
<tr>
<td>8000_0000</td>
<td>128MB External</td>
</tr>
<tr>
<td>9000_0000</td>
<td>128MB External</td>
</tr>
<tr>
<td>A000_0000</td>
<td>128MB External</td>
</tr>
<tr>
<td>B000_0000</td>
<td>128MB External</td>
</tr>
<tr>
<td>FFFF_FFFF</td>
<td></td>
</tr>
</tbody>
</table>

'C6713 DSK

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000_0000</td>
<td>16MB SDRAM</td>
</tr>
<tr>
<td>9008_0000</td>
<td>256K byte FLASH</td>
</tr>
<tr>
<td></td>
<td>CPLD</td>
</tr>
<tr>
<td></td>
<td>Available via Daughter Card Connector</td>
</tr>
</tbody>
</table>

CPLD:
- LED’s
- DIP Switches
- DSK status
- DSK rev#
- Daughter Card

Memory Map Review

- C6000 CPU
- EMIF
- L2 SRAM
- 8000_0000 (CE0)
- 128 MB
- 9000_0000 (CE1)
- 128 MB
- A000_0000 (CE2)
- 128 MB
- B000_0000 (CE3)
- 128 MB

Memory Map is a table representation of memory...
Sidebar: Memory Addressing on C6x

**CE Pins Select Memory Space**

- **FFFF_FFFF**
- **0000_0000**

<table>
<thead>
<tr>
<th>Memory Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>64KB Internal</td>
</tr>
<tr>
<td>(Program or Data)</td>
</tr>
<tr>
<td>On-chip Periph</td>
</tr>
<tr>
<td>128MB External</td>
</tr>
<tr>
<td>128MB External</td>
</tr>
<tr>
<td>128MB External</td>
</tr>
<tr>
<td>128MB External</td>
</tr>
</tbody>
</table>

**C6x Addressing**

- With only 20 address pins, only SDRAM can access full 128M Bytes per CE space.
- Not all CPU/DMA address lines are used in C6x01 example above.
Memory Types

Overview

SDRAM - Synchronous (clocked) DRAM
- SDRAM provides lowest cost / bit cheap
- Operates up to 100 MHz fast
- Built-in SDRAM controller makes interfacing simple easy
- Only SDRAM can reach full address space big

ASYNC - Traditional (unclocked) memories
- Wide array of memories (Flash, SRAM, Regs, FPGA/ASIC) flexible
- Can use buffer/drivers, address decoding, etc.
- Allows multiprocessor access share

Note: SBSRAM is covered later in the chapter - it's not implemented on the DSK

Selecting Memory Type

| 180_0000 | Global Control |
| 180_0008 | CE0 Control    |
| 180_0004 | CE1 Control    |
| 180_0010 | CE2 Control    |
| 180_0014 | CE3 Control    |
| 180_0018 | SDRAM Control  |
| 180_001C | SDRAM Refresh Prd|
| 180_0020 | SDRAM Extension|

<table>
<thead>
<tr>
<th>MTYPE</th>
<th>CEEx Control Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW, +0010</td>
<td>7 4</td>
</tr>
</tbody>
</table>

0000b = 8-bit-wide Async
0001b = 16-bit-wide Async
0010b = 32-bit-wide Async
0011b = 32-bit-wide SDRAM
0100b = 32-bit-wide SBSRAM
1000b = 8-bit-wide SDRAM
1001b = 16-bit-wide SDRAM
1010b = 8-bit-wide SBSRAM
1011b = 16-bit-wide SBSRAM
Using SDRAM

1. Select SDRAM and verify it meets system performance timing

**DM642 SDRAM Recommendations**

- Due to datasheet requirements, the following is recommended:
  - 1 bank (max of 2 chips) of SDRAM connected to EMIF
  - Up to 1 bank of buffers connected to EMIF for async memories
  - Trace lengths between 1 and 3 inches
  - 183MHz SDRAM for 133MHz EMIF operation
  - 143MHz SDRAM for 100MHz EMIF operation

- Therefore:
  - To run the EMIF at 133MHz and meet the above requirements, the largest memory size available today is 16M Bytes using two 2Mx32 SDRAMs.

- Alternatively:
  - The largest memory size achievable using x32 devices is 32MBytes using 4Mx32 SDRAMs. However, these devices are only available at 166Mhz.
  - Another option is to use x16 devices, but you have to use four of these since the EMIF is 64 bits wide. Also, the fastest speed grade is 167MHz.

* These guidelines are for DM642 in June 2003. Other C6000 devices require similar consideration.

**SDRAM Design Considerations**

- Use Daisy chaining or minimum stub length routing on EMIF signals
- Keep trace lengths as close as possible to the same length
- ‘Swizzle’ signals such that they are flow through to avoid signal criss-crossing as much as possible. For example, on resistor packs or SDRAM data pins on a ‘byte’ boundary
- Serial termination resistors should be inserted into all EMIF output signal lines to maintain signal integrity
- Use controlled impedance of 50-60 ohms on layout/pwb fabrication
- Ground layer is a must, and can be duplicated to help with controlled impedance any time there is an odd number of layers

- Perform timing analysis to verify A/C timings are met using I/O Buffer Information Specification (IBIS)
  - In fact, using IBIS modeling you may find you can improve upon the suggestions provided on the previous slide
  - Refer to application note: Using IBIS Models for Timing Analysis
What is IBIS?

IBIS is a standard for describing the analog behavior of the buffers of digital devices using plain ASCII text formatted data.

- IBIS files are not models, they just contain the data that will be used by the simulation tool's behavioral models and algorithms.
- Started in the early 90's to promote tool independent I/O models for system level Signal Integrity work.
- It is now the ANSI/IEEE-656 and IEC 62814-1 standard.

What Are These Models Based On?

**SPICE (transistor) model**

- Voltage/current/capacitance relationships of device nodes are calculated with detailed equations using device geometry, and properties of materials.
- Measured data is curve fitted for the equations.

**IBIS (or behavioral) model**

- Current/voltage/time relationships of entire buffer (or building block) are based on lookup tables (IV and VI curves).
- Data tables are generated from full SPICE model simulations or external measurements.
- Data may be curve fit to equations for more efficiency and flexibility.

Model Characteristics

**SPICE (transistor) model**

- Simulates very slowly, because voltage/current relationships are calculated from lower level data.
- Voltages/currents are calculated for each circuit element in the buffer.
- Best for circuit designers.
- Too slow for system level interconnect design.
- Reveals process and circuit Intellectual Property.

**IBIS (or behavioral) model**

- Simulates fastest, because voltage/current/time relationships are given only for the external nodes of the entire buffer (or building block).
- No circuit detail involved.
- Useless for circuit designers.
- Ideal for system level interconnect design.
- Hides both process and circuit intellectual property.
2. Specify SDRAM Parameters

**SDRAM Control Register**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>20</th>
<th>19</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRC</td>
<td>reserved</td>
<td></td>
</tr>
</tbody>
</table>

- Calculate the number of cycles for each of the three timing parameters using the SDRAM datasheet. The following formula may help:

\[
TR_\text{__} = \left( \frac{t_{R\text{CD}}}{t_{E\text{CLKOUT}}} \right) - 1
\]

- There’s only one SDRAM Control Register, therefore all SDRAM spaces must have the same configuration

**TRCD** = \( \frac{30\text{ns}}{10\text{ns}} - 1 = 2 \)

**TRP** = \( \frac{30\text{ns}}{10\text{ns}} - 1 = 2 \)

**TRC** = \( \frac{90\text{ns}}{10\text{ns}} - 1 = 8 \)
3. Calculate Refresh Timing

SDRAM Refresh Timing Register

From the SDRAM data sheet:
Refresh Rate = “4K Auto Refresh each 64ms”
= 64 ms / 4096

Period = \( t_{\text{Refresh Rate}} / t_{\text{ECLK}} \)
= \((64\text{ms}/4096) / 10\text{ns})
= 1562 (0x61A)
Using Asynchronous Memory

- Generic Read Timing
- Async Example - Flash
- Flash Read Timing
- Flash Write Procedure

Asynchronous Memory - What is it?

- Traditional Memory Interface
  - Doesn’t require clock
  - Non Pipelined Accesses
  - Ex: SRAM, EPROM, Regs, Ext. Periph

- External buffers can be used for:
  - Shared memory
  - Increased fanout
  - Isolation

![Diagram showing asynchronous memory access]

Access 1
A
D

Access 2
A
D

Access 3
A
D
Async Read Timing

EDA

CEx Register

<table>
<thead>
<tr>
<th>19</th>
<th>16</th>
<th>13</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Setup</td>
<td>Read Strobe</td>
<td>MTYPE</td>
<td>Read Hold</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Setup = 1
- Strobe = 2
- Hold = 1

Ed

Ed

Ed

Ed
Memory Types

Async Flash Memory

Flash Read Timing

- DSK has 128K Flash
- Provides re-programmable, non-volatile memory
- Pre-program with code, init values and boot-strap program
- Stores non-volatile, run-time data

Looking more closely at the timing ...

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>AT29LV010A-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>tACC</td>
<td>Address to Output Delay</td>
<td></td>
</tr>
<tr>
<td>tCE[1]</td>
<td>CE to Output Delay</td>
<td></td>
</tr>
<tr>
<td>tCE[2]</td>
<td>CE to Output Delay</td>
<td>0 100</td>
</tr>
<tr>
<td>tOE[3]</td>
<td>OE or CE to Output Float</td>
<td>0 50</td>
</tr>
<tr>
<td>tOH</td>
<td>Output Hold from CE, OE or Address, whichever occurred first</td>
<td>0</td>
</tr>
</tbody>
</table>
Let's figure out the timing for the DSK's async Flash memory …

**Writing to Flash**

**Writing to DSK's Flash**

- Flash is a non-volatile memory, i.e. it can't normally be written to

- To change it's content, you must "unlock" it with a special procedure:

  1. Write 0xAA to 0x5555
  2. Write 0x55 to 0x2AAA
  3. Write 0xA0 to 0x5555
  4. Write new data to 128 byte sector  
     (data must be written in 128 byte chunks)

    - Flash requires 20ms to complete internal write cycle.  
      Data I/O7 can be polled to determine when write cycle is complete.

- PC based tools available for Flash programming

- BSL functions allow runtime writing to Flash
Sidebar: Optional Async Timing

Async Read - Maximum Speed

- ECLKOUT
- EA, CE, BE
- AOE
- ARE
- ED

Setup = 1  Strobe = 1  Hold = 0

Async Write Timing

- ECLKOUT
- EA, CE, BE
- AWE
- ED

Setup  Strobe  Hold

<table>
<thead>
<tr>
<th>31</th>
<th>28</th>
<th>27</th>
<th>22</th>
<th>21</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Setup</td>
<td>Write Strobe</td>
<td>Write Hold</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 - 15</td>
<td>1 - 63</td>
<td>0 - 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Minimum Turn-Around Time

- First R/W in a series requires an extra "setup" cycle
- CE on last access is held active for a minimum of 7 cycles
- Bus turn-around time (R→W or W→R) is approx 9 cycles
  (please refer to data sheet for specifics for each individual processor)

Async Memory - Summary

<table>
<thead>
<tr>
<th></th>
<th>Write Setup</th>
<th>Write Strobe</th>
<th>Write Hold</th>
<th>Read Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RW, +1111</td>
<td>RW, +111111</td>
<td>RW, +11</td>
<td>RW, +1111</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>Read Hold</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>Read Hold</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>RW, +111111</td>
<td>RW, +0010</td>
<td>RW, +011</td>
<td></td>
</tr>
</tbody>
</table>

0000b = 8-bit-wide Async
0001b = 16-bit-wide Async
0010b = 32-bit-wide Async
0011b = 32-bit-wide SDRAM
0100b = 32-bit-wide SBSRAM
1000b = 8-bit-wide SDRAM
1001b = 16-bit-wide SDRAM
1010b = 8-bit-wide SBSRAM
1011b = 16-bit-wide SBSRAM

Cycles

- Setup = 1* - 15
- Strobe = 1* - 63
- Read Hold = 0 - 7
- Write Hold = 0 - 3

* 0 → 1 and 1 → 1
Programming the EMIF

Using the EMIF with CSL

Program EMIF with CSL

```c
far const EMIFA_Config C6416DskEmifConfigA = {
    EMIF_GBLCTL_RMK{    // 0x00012070
        EMIF_GBLCTL_EK2RATE_FULLCLK, // bits 18-19 = 00
        EMIF_GBLCTL_EK2HZ_CLK,        // bit 17 = 0
        EMIF_GBLCTL_EK2EN_ENABLE,     // bit 16 = 1
        EMIF_GBLCTL_BRMODE_MRSTATUS,   // bit 13 = 1
        EMIF_GBLCTL_BUSREQ_LOW,       // bit 11 = 0
        ...                             // bit 3 = 0
    },
    0x00000000, /* cectl0 */
    ...                      /* cese3 */
};

void emifInit(){
    EMIFA_config(&C6416DskEmifConfigA);
}
```

- Program EMIF similar to other peripherals.
- Since EMIF is not a multi-channel peripheral, no _open function is required.
Programming the EMIF with Assembly

Program EMIF with Assembly (1)

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMIF</td>
<td>0x01800000</td>
</tr>
<tr>
<td>GBLCTL</td>
<td>0x________</td>
</tr>
<tr>
<td>CE0CTL</td>
<td>0x________</td>
</tr>
<tr>
<td>CE1CTL</td>
<td>0x________</td>
</tr>
<tr>
<td>CE2CTL</td>
<td>0x________</td>
</tr>
<tr>
<td>CE3CTL</td>
<td>0x________</td>
</tr>
<tr>
<td>SDCTL</td>
<td>0x________</td>
</tr>
<tr>
<td>SDTIM</td>
<td>0x________</td>
</tr>
<tr>
<td>SDOPT</td>
<td>0x________</td>
</tr>
</tbody>
</table>

cEMIF: mvkl EMIF, A0
mvkh GBLCTL, A1
mvkl CE0CTL, A1
mvkh CE1CTL, A1
stw A1, *+A0[0]
...  
mvkh SDINST, A1
stw A1, *+A0[2]

EMIF .equ 0x01800000

Global Control
CE0 Control
CE1 Control
CE2 Control
CE3 Control
SDRAM Control
SDRAM Ref Prd
SDRAM Extension

◆ Add the desired register values to the blank spaces and code will program EMIF
◆ Assembly code will work for all devices, if you ...

Better yet, ...

Program EMIF with Assembly (2)

/* Include Header File */
#include "csl_emif.h"

/* Config Structures */
far const EMIF_Config myEMIF;

0x00003078, /* Global Control Reg. (GBLCTL) */
0x00000020, /* CE0 Space Control Reg. (CE0CTL) */
0xFFFF3F23, /* CE1 Space Control Reg. (CE1CTL) */
0x00000030, /* CE2 Space Control Reg. (CE2CTL) */
0xFFFF3F23, /* CE3 Space Control Reg. (CE3CTL) */
0x0388F000, /* SDRAM Control Reg.(SDCTL) */
0x00000040 /* SDRAM Timing Reg.(SDTIM) */
0x00F02AE0 /* SDR... */

.global _myEMIF

EMIF .equ 0x01800000

cEMIF: mvkl EMIF, A0
mvkh EMIF, A0
mvkl _myEMIF, A1
mvkh _myEMIF, A1
ldw *A1, A2
stw A2, *A0
ldw ***A1[1], A2
stw A2, *+A0[2]
...
ldw ***A1[1], A2
stw A2, *+A0[8]
Programming the EMIF with GEL

Program EMIF with GEL

```c
init_emif()
{
    // First we define the EMIF addresses
    #define EMIF_GCTL         0x01800000
    #define EMIF_CE1          0x01800004
    #define EMIF_CE0          0x01800008
    #define EMIF_CE2          0x01800010
    #define EMIF_CE3          0x01800014
    #define EMIF_SDRAMCTL     0x01800018
    #define EMIF_SDRAMTIMING  0x0180001C
    #define EMIF_SDRAMEXT     0x01800020

    // Now we set the values
    *(int *)EMIF_GCTL = 0x00003300;        // EMIF global
    *(int *)EMIF_CE0 = 0x00000030;         // CE0-SDRAM
    *(int *)EMIF_CE2 = 0xFFFFFF23;         // CE2-32bit async on daughtercard
    *(int *)EMIF_CE3 = 0xFFFFFF23;         // CE3-32bit async on daughtercard
    *(int *)EMIF_SDRAMCTL = 0x07227000;    // SDRAM control register(100 MHz)
    *(int *)EMIF_SDRAMTIMING = 0x0000061A; // SDRAM Timing register
    *(int *)EMIF_SDRAMEXT = 0x00054529;    // SDRAM Extension register
```

When does this GEL script get executed?

GEL Startup

```
StartUp() {
    setup_memory_map();
    GEL_Reset();
    init_emif();
}
```

* The StartUp() function is called every time you start Code Composer.
* You can customize this function to perform desired initialization.
* This function may be commented out if no initialization is needed.
*/
EMIF – CPU’s Access Performance

Even though an internal memory access requires a four cycle access time, as with most modern RISC processors, the C6000’s pipelined architecture provides means to overcome this delay.
Even providing a zero wait-state off-chip memory, the CPU’s access time for external memory will be upwards of 18 cycles.

- Total affect is a 14 cycle delay. (18 cycles less four afforded by C6000’s hardware pipelining.)
- C6201 details are shown here. Similar issues affect all C6000 devices (in fact, all high perf μP), but they are manifested differently. For example, the cache in more recent devices mitigate the affect of these delays by keeping often used code and data in faster on-chip memory.

Besides cache, what is a better way to increase EMIF throughput?

Unlike the CPU, the EDMA (and DMA) can pipeline-up access through the EMIF delays to achieve single-cycle throughput from zero wait-state external memories.

- While the first access may take 14 cycles, subsequent accesses can get down to a single cycle.
Fanout

‘C6201 Bus Fanout

- Bus pin drivers rated for 30pf loading
  - Devices are designed for 45pf loads, but testing equipment cannot guarantee it
- Most memory devices present 5pf loads
- Total fanout is six memory devices
- While this slide is slightly old, the issue remains. Again, IBIS modeling is an excellent way to deal with this issue.

<table>
<thead>
<tr>
<th>Type</th>
<th>Top Speed</th>
<th>H/W Wait</th>
<th>Max Size/Fan</th>
<th>Glueless</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYNC</td>
<td>100 MHz</td>
<td>Yes</td>
<td>16 M/∞</td>
<td>Yes/No</td>
</tr>
<tr>
<td>SBSRAM</td>
<td>200 MHz</td>
<td>No</td>
<td>3 MB</td>
<td>Yes</td>
</tr>
<tr>
<td>SDRAM</td>
<td>100 MHz</td>
<td>No</td>
<td>48 MB</td>
<td>Yes</td>
</tr>
</tbody>
</table>

System with All Memory Types

‘C6201

CBTs and Widebus Transceivers work great
Shared Memory

Costs you extra:
Speed, Power, Reliability, Money, etc.

Using 3-state buffers.
One of the μP or another device arbitrates.

What is the drawback of using a buffer here?
Costs you extra: Speed, Power, Reliability, Money, etc.
**Shared Memory**

When ‘C6x drives HOLDA active:
- EMIF signals tri-stated
- CPU continues to execute as long as no off-chip access is needed

---

**HOLD Status Bits (GBLCTL)**

- HOLD and HOLDA status
- Disable HOLD feature (NOHOLD = 1)

**C6711 EMIF GBLCTL**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>rsv</td>
</tr>
<tr>
<td>15</td>
<td>rsv</td>
</tr>
<tr>
<td>14</td>
<td>rsv</td>
</tr>
<tr>
<td>13</td>
<td>rsv</td>
</tr>
<tr>
<td>12</td>
<td>BUSREQ</td>
</tr>
<tr>
<td>11</td>
<td>ARDY</td>
</tr>
<tr>
<td>10</td>
<td>R, +0</td>
</tr>
<tr>
<td>9</td>
<td>RW, +0</td>
</tr>
<tr>
<td>8</td>
<td>RW, +1</td>
</tr>
<tr>
<td>7</td>
<td>RW, +1</td>
</tr>
<tr>
<td>6</td>
<td>R, +0</td>
</tr>
<tr>
<td>5</td>
<td>R, +x</td>
</tr>
<tr>
<td>4</td>
<td>R, +x</td>
</tr>
<tr>
<td>3</td>
<td>RW, +x</td>
</tr>
<tr>
<td>2</td>
<td>RW, +1</td>
</tr>
<tr>
<td>1</td>
<td>RW, +1</td>
</tr>
<tr>
<td>0</td>
<td>R, +000</td>
</tr>
</tbody>
</table>
SBSRAM

Synchronous Burst SRAM (SBSRAM)

- SBSRAM's pipelines memory accesses
- With Burst mode a processor only needs to generate an address every four sequential accesses
  - Not required by C6000 DSP's as they're fast enough
  - '0x devices don't use (have) this feature
  - '1x devices include the burst feature for power savings (only one address pin needs toggling for four sequential accesses)

SBSRAM Timing

- Data is available 2 cycles after address appears
- Data can be accessed at the rate of 1 per cycle
SDRAM Optimization

<table>
<thead>
<tr>
<th>SDRAM Extension Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td>RESERVED</td>
</tr>
<tr>
<td>15 14</td>
</tr>
<tr>
<td>DQM</td>
</tr>
</tbody>
</table>

- Most SDRAMs will work without programming this register. This is the case for the C6711 DSK.
- Program the SDRAM Extension (SDOPT) register to optimize SDRAM performance.
- Please refer to the SDRAM applications note (at the TI website) for further details on programming this register.

EMIF ‘C6x Family Comparison

<table>
<thead>
<tr>
<th>EMIF Variations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices</td>
</tr>
<tr>
<td>Scheme</td>
</tr>
<tr>
<td>Bus Width</td>
</tr>
<tr>
<td>Size (MB)</td>
</tr>
<tr>
<td>Sync Clocking</td>
</tr>
<tr>
<td>CE1 Types</td>
</tr>
<tr>
<td>Sync Mem Allowed in System</td>
</tr>
<tr>
<td>Pipelined SBSRAM</td>
</tr>
<tr>
<td>Flow thru SBSRAM</td>
</tr>
<tr>
<td>ZBT SRAM</td>
</tr>
<tr>
<td>Std Sync FIFO</td>
</tr>
<tr>
<td>F W F T FIFO</td>
</tr>
</tbody>
</table>
Sidebar: C6x01 Memory Map

**C6201/C6701 Memory Ranges**

- 0000_0000
  - CE0
  - 4M x 8 Async or SBSRAM
  - CE0
  - 16M x 8 SDRAM (access as 32-bit only)

- 0100_0000
  - CE1
  - 4M x 8 Async or SBSRAM
  - CE1
  - (read access as 8/16/32-bit, write access as 32-bit only)

- 0140_0000
  - CE0
  - 2K x 256 Int'l Prog
  - CE0
  - On-chip Peripherals

- 0200_0000
  - CE2
  - 4M x 8 Async or SBSRAM
  - CE2
  - 16M x 8 SDRAM (access as 32-bit only)

- 0300_0000
  - CE3
  - 4M x 8 Async or SBSRAM
  - CE3
  - 16M x 8 SDRAM (access as 32-bit only)

- 0x000_0000
  - CE0
  - 64K x 8 Int'l Data

---

**‘C6x01 - MAP 0 vs. MAP 1**

Memory

- 000_0000
  - CE0 (16M)
  - CE1 (4M)
  - P
  - CE2
  - CE3
  - D

Memory

- 000_0000
  - CE0 (16M)
  - CE1 (4M)
  - P
  - CE2
  - CE3
  - D