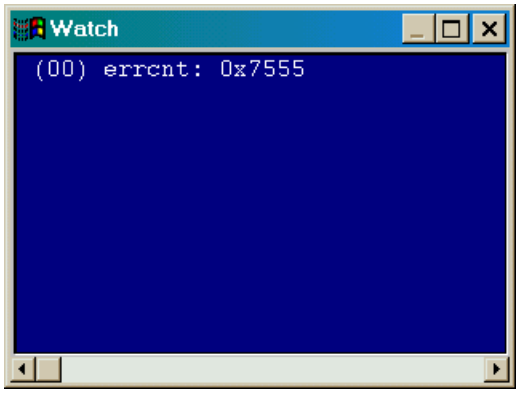
window will show the number of transmission errors that occurred during the one-second break in the loopback path.



Does this number make sense? The 8032 MCU is running at 25 MHz and the UART is in mode 2 so it transmits a single bit every 64 clock cycles = 2.56 μs. Each byte transmission consists of eleven bits (a start bit, eight data bits, one programmable data bit, and a stop bit) over an interval of 28.2 μs. So a one-second interruption in the loopback path will cause 1 s / 28.2 μs = 35,500 errors. The **Watch** window reports `errCnt` = 0x7555 = 30,037 which is pretty close given that you have to manually open and close the DIP switch.

Design 2.4 - 8032 MCU Memory Tester

Your last MCU-based design in this chapter will test the external SRAM chip on the CSoC Board. It will use the same hardware configuration as the first design (Figure 11). The MCU will determine the segment of SRAM to test by reading the DIP switch settings through port **P0**. Then the MCU will write a pseudo-random series of bytes to the SRAM segment and read it back to see if it was stored correctly. Finally, the result of the test is displayed on the LED digit.

Address Translation in the Triscend CSoC

In order to understand how the SRAM testing program works, you need to know how the CSoC assists the 8032 MCU with memory accesses. The CSoC has a 32-bit CSI address bus that is used to selectively access various components of the internal circuitry (registers in the CSL, DMA controllers, etc.) or external devices (e.g., memory devices connected to the MIU). But the 8032 MCU uses 16-bit addresses. In order for the MCU to be able to access all the circuitry inside and outside the CSoC, address mappers are provided that translate the 16-bit *logical addresses* into full 32-*bit physical addresses*.

An address mapper uses three parameters to control the translation process:

Block Size: This parameter specifies the size of the contiguous range of addresses on which the mapper performs translations. For the Triscend CSoC, the block size is always a power of two in the range from 2^8 = 256 up to 2^{16} = 65536 (which is the maximum range of the 8032 logical addresses). The block size exponent is stored in a five-bit field with permissible values in the range [8,16].

Zone Source Address: The address mapper is activated when the logical address from the 8032 MCU matches the zone source address. For the Triscend CSoC with a block size of 2^N, only the upper 16-N bits of the logical and zone source addresses are compared. For example, if the block size is 2^{14} = 16384 then only the upper two bits of the addresses are used in the comparison. If the upper two bits of the zone source address are 10, then the address mapper will trigger for any logical address in the range [32768, 49151]. Because the smallest block size is 2^8, the address comparison uses a maximum of 16-8 = 8 bits so the zone source address is stored in a single byte.

Target Address: When the address mapper triggers, this parameter fills in the upper bits of the physical address. For the Triscend CSoC with a block size of 2^N, the physical address is formed by concatenating the upper 32-N bits of the target address to the lower N bits of the logical address. For example, if the block size is 2^{14} then the target address fills in the upper 32-14 = 18 bits of the 32-bit physical address. Because the smallest block size is 2^8, the target address has to provide a maximum of 32-8 = 24 physical address bits so the target address is stored in three bytes.

Once the 32-bit physical address is generated, how does the CSoC know whether this address should be used to address a register in the CSL, a byte in the internal SRAM, or a byte in external SRAM (via the MIU)? The CSoC is hardwired such that different functional areas are selected by various ranges of physical addresses. The CSoC uses bits 16 through 23 (the second most-significant byte) of the physical address to select the functional areas as listed in Table 9.

Table 9: CSoC functions and their associated physical address ranges.

Physical Address Bits $A_{23}…A_{16}$	CSoC Function
0x00	8032 MCU internal ROM
0x01	Internal 16 KByte SRAM
0x02	System configuration registers (CRU)
0x03 - 0x07	Debugger resources
0x10 - 0x7F	CSL soft modules
0x80 - 0xFF	External memory (MIU)

What does the CSoC do with the most-significant byte of the physical address? Almost nothing. You can use it for debugging purposes by loading it with bit patterns that appear on the CSoC address lines when the associated address mapper is triggered. Given that you are reading through a beginner's tutorial, you probably won't use this feature for a while.

How many address mappers does the CSoC have in it? There are three *code mappers* (C0, C1, and C2) that handle the 8032 MCU instruction addresses. The code mappers are prioritized so only a single mapper is triggered by any logical address. Then there are six *data mappers* (D0 – D5) that translate data addresses. The data mappers are also prioritized. Finally, there is a single mapper for SFR addresses. These address mappers and their priorities are listed in Table 10.

Table 10: Code, data, and SFR address mappers and their priorities.

Address Mapper		Priority
code mappers	C0	3 (lowest)
	C1	2
	C2	1 (highest)
data mappers	D0	6 (lowest)
	D1	5
	D2	4
	D4	3
	D5	2
	D3	1 (highest)
SFR mapper	SFR	N/A

Testing the CSoC Board Memory

Now it's time to take your abstract knowledge of the CSoC address mappers and put it to use in a concrete application: testing the memory on your CSoC Board. Start by opening the ***Chap21*** FastChip project and saving it as a project named ***Chap24***. Then click on the Generate icon in the toolbar to create the Chap24.h header file.

Next create a keil folder within the Chap24 project folder. Use the Keil IDE to create a ***chap24*** project in the keil folder that includes the source code of Listing 5.

The first subroutine in Listing 5 (lines 5–13) is called whenever the watchdog timer in the 8032 MCU times out. Normally the watchdog timer is used to reset the MCU into a known state if the application code fails to periodically refresh the timer, but here we use the watchdog as a simple source of periodic interrupts. The watchdog interrupt identifier is 12 and the interrupt subroutine uses register bank 3. The first action of the `ledFlash` subroutine is to write a bit pattern to the LED digit (line 8) and then rotate the pattern one bit position to the left (line 10). Then the watchdog timer interrupt flag is cleared (line 11). The net result is that the `ledFlash` interrupt subroutine will sequentially light the segments of the LED digit as the watchdog timer generates a series of interrupts. You will use this as a visual signal to indicate the memory test is in progress.

The `genRand` subroutine on lines 15–26 implements a *simple linear-feedback shift register* (LFSR). The LFSR starts from a seed value and generates a series of 255 distinct values before it repeats. Several bits from the current seed value are exclusive-ORed together and the result is shifted into the most-significant bit of the seed to form the next value in the series (lines 20–24). This value is returned to the calling routine on line 25 and it also becomes the seed for the next value in the series.

Lines 30–61 define the memory testing subroutine `testMemRange`. It accepts two parameters: the lower and upper bounds on the section of memory it will test. The subroutine writes the values from the `genRand` subroutine into the memory range on lines 41–45. Then the contents of the memory range are read back and compared with the values from `genRand` on lines 49–56. (On line 49 the `genRand` subroutine is re-initialized with the same seed used for the memory writing loop so it generates the exact same series of values.) The `memOK` flag is cleared if the value read from memory doesn't match the output from the `genRand` subroutine. This indicates an error in the operation of the memory. Otherwise the `memOK` flag remains set which indicates the memory appears to be operating correctly. The value of the `memOK` flag is returned to the calling routine on line 60.

The `testMemRange` subroutine also enables the watchdog timer before the memory test runs (lines 36–38) and disables the watchdog timer after the test is completed (line 59). Thus, the LED will flash only during the memory test.

The `main` routine reads the DIP switch settings and uses them to set-up the data mappers on lines 77–89. Then the `testMemRange` subroutine is used to test the selected memory region on lines 92 and 93 and display an `0` or an `E` on the LED digit if the memory passes or fails, respectively.

Line 77 disables data mappers 2, 4, and 5. (The definitions of all the address mapper control registers are stored in the `chap24.h` header file.) Data mappers D3 and D0 are always enabled. The remaining data mapper, D1, is controlled by the DIP switches. The functions of the DIP switches in this application are listed in Table 11. DIP switches #1, #2, and #3 set-up the values of target address bits A_{14}, A_{15}, and A_{16}, respectively. DIP switches #4, #5, and #6 are used to set the size of the block of memory that will be tested. And DIP switch #7 is used to enable or disable the D3 data mapper. These values are read from the DIP switch through the **P0** port on lines 80–82. Note that the block size exponent value is limited to a value from 0 to 7 (it's only three bits), so the value is increased by nine on line 81. This lets the program test memory ranges from 2^9 = 512 bytes up to 2^{16} = 65536 bytes.

Table 11: Functions of the DIP switches in the memory test application.

DIP Switch	Function
#1, #2, #3	Target address bits A_{14}, A_{15}, A_{16}
#4, #5, #6	Block size exponent
#7	Data mapper enable

The DIP switch values are written into the data mapper control registers on lines 85–89. The enable bit is written to bit 6 of the `DMAP1_CTL` register and the block size exponent is written to the lower five bits of this register on line 85.

The target address for the data mapper is stored in registers `DMAP1_TAR_0` (address bits A_{15}–A_8), `DMAP1_TAR_1` (address bits A_{23}–A_{16}), and `DMAP1_TAR_2` (address bits A_{31}–A_{24}). DIP switches #1 and #2 set the values of A_{14} and A_{15} in the upper two bits of `DMAP1_TAR_0`, while DIP switch #3 sets the value of A_{16} in the least significant bit of `DMAP1_TAR_1`. The most significant bit of `DMAP1_TAR_1` (A_{23}) is set so that the data mapper will access the external SRAM on the CSoC Board through the MIU (see the last entry of Table 9). Finally, the upper byte of the target address isn't used for anything so it is just set to zero.

The zone source address is cleared on line 89 so the data mapper will respond to logical addresses from the MCU whose upper bits are all zero.

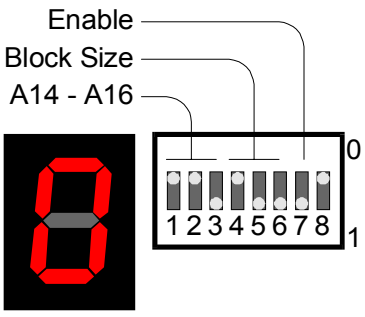
Listing 5: C application code for the CSoC memory testing program.

```
 1  #include "..\Chap24.h"
 2
 3  // interrupt routine called whenever watchdog times out
 4  // that flashes the LED digit
 5  unsigned char ledPattern = 1; // LED segment flash
 6  static void ledFlash(void) interrupt 12 using 3
 7  {
 8       ledPort = ledPattern;    // update LED
 9       // now rotate the pattern written to the LED
10       ledPattern = (ledPattern<<1) | (ledPattern>>7);
11       TA = 0xAA; TA = 0x55; WDIF = 0;    // reset watchdog int
12  }
13
14  // routine to generate a random number in [1..255]
15  unsigned char seed = 1;  // random number seed
16  unsigned char genRand()
17  {
18       unsigned char randbit;
19       randbit = 0;
20       if(seed&1)  randbit ^= 0x80;
```

```
21        if(seed&4)  randbit ^= 0x80;
22        if(seed&8)  randbit ^= 0x80;
23        if(seed&16) randbit ^= 0x80;
24        seed = (seed>>1) | randbit;
25        return seed;   // return the random number
26   }
27
28   // routine to test memory by writing random numbers
29   // to memory and then reading them back and comparing.
30   typedef volatile unsigned char xdata xdataChar;
31   unsigned char testMemRange(xdataChar* lo, xdataChar* hi)
32   {
33        xdataChar* p;  // pointer to memory space being tested
34        unsigned char memOK;     // memory test result flag
35
36        ledPattern = 0x01;  // initialize LED flash pattern
37        CKCON = 0x00;       // watchdog timeout every 2^17 clocks
38        EWDI = 1;           // enable watchdog timer
39
40        // write random number sequence to memory
41        seed = 1;           // seed the random number generator
42        p=lo;               // start writing at lower address
43        do
44             *p = genRand();     // store random number in memory
45        while(p++ != hi);   // write until upper address is reached
46
47        // now read the memory range and compare against the
48        // output of the re-initialized random number generator
49        seed = 1; // init the random generator with the same seed
50        memOK = 1;     // start off assuming the memory is OK
51        p=lo;          // start reading at lower address
52        do
53             // memOK stays true as long as the contents read from
54             // memory match the output from the random generator
55             memOK = memOK && (*p == genRand());
56        while((p++ != hi) && memOK);  // check entire memory
57                                      // range or until an error is seen
58
59        EWDI = 0;           // disable watchdog
60        return memOK;       // return result of memory test
61   }
62
63
64   #define MEM_OK 0x3F // bit pattern to display 'O' on LED
65   #define MEM_ERR 0x79   // bit pattern to display 'E' on LED
66
67   main()
```

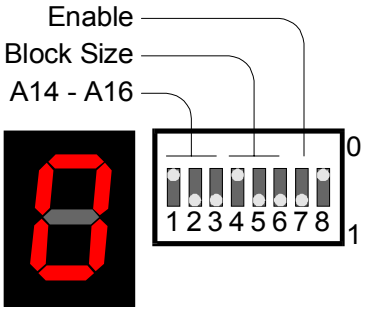
```
68  {
69       unsigned char enable, blockSize, targetAddr;
70       unsigned int hi;
71
72       Chap24_INIT();
73
74       EA = 1;         // enable interrupts
75
76       // disable data mappers 2, 4, and 5
77       DMAP2_CTL = DMAP4_CTL = DMAP5_CTL = 0;
78
79       // get the memory test settings from DIP switches 1-7
80       enable     =  (P0>>6) & 0x1;
81       blockSize  = ((P0>>3) & 0x7) + 9;
82       targetAddr =   P0     & 0x7;
83
84       // set-up data mapper D1
85       DMAP1_CTL   = (enable<<5) | blockSize;
86       DMAP1_TAR_0 = targetAddr<<6;
87       DMAP1_TAR_1 = 0x80 | ((targetAddr>>2) & 0x01);
88       DMAP1_TAR_2 = 0x00;
89       DMAP1_SRC   = 0;
90
91       // test the memory range [0,hi] and report result on LED
92       hi = (1<<blockSize) - 1;
93       ledPort = testMemRange(0,hi) ? MEM_OK:MEM_ERR;
94       while(1) ;      // loop forever
95  }
```

Once the code in Listing 5 is compiled and linked, you can use FastChip to bind and download the hardware and application code for the CSoC. Then use dScope to set-up a debugging link to your CSoC Board and reset the MCU. Set the DIP switches, click on Go!, and you should see the LED digit flash and then display either a 0 or E. Various DIP switch settings and the memory test results are shown in the following paragraphs.

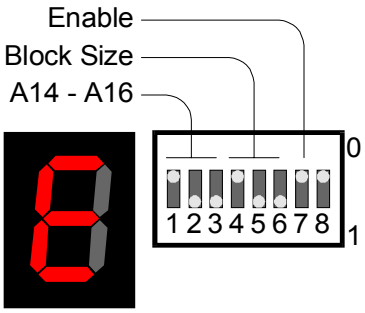


DIP switch #7 is set to one so data mapper D1 is enabled. The block size is set to 2^{6+9} = 32768 and the program tests logical addresses 0x0000 through 0x7FFF. The resulting physical address range put out by the data mapper is [0x810000, 0x817FFF]. The external 128 KByte SRAM on the CSoC Board is only connected to the lower seventeen bits of the physical address, so SRAM addresses during the test are in the range [0x10000, 0x17FFF]. Provided the SRAM chip is functioning properly, the memory test should pass and an 0 will be displayed.

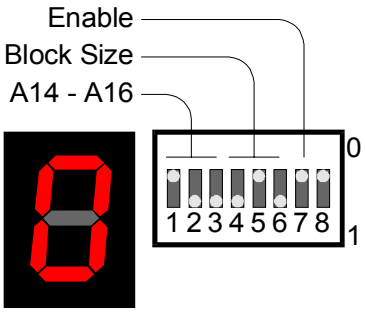
Everything is identical to the previous test except the resulting physical address range put out by the data mapper is moved up to the range [0x818000, 0x81FFFF]. This moves the SRAM addresses during the test to the range [0x18000, 0x1FFFF]. Once again this is in the functioning address range for the SRAM chip, so the memory test should pass and an 0 will be displayed.



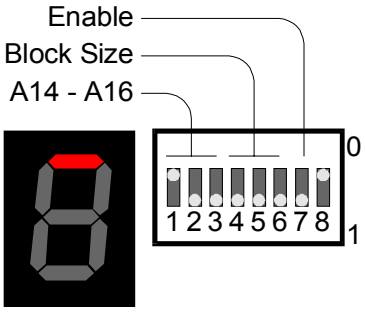
Everything is identical to the previous test except data mapper D1 is disabled. This means the next active data mapper in the priority list, D0, will trigger on the logical addresses in the range [0x0000, 0x7FFF]. D0 maps logical addresses into the physical address range of the internal SRAM in the TE505 CSoC. The 16 KByte internal SRAM cannot pass a 32 KByte memory test, so at the conclusion an E is displayed on the LED digit.



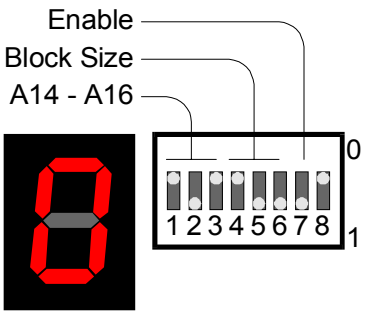
Everything is identical to the previous test except the block size is set to 2^{5+9} = 16384 so the program tests logical addresses 0x0000 through 0x3FFF. Now the address range checked by the program fits within the 16 KByte internal SRAM, so at the conclusion an 0 is displayed on the LED digit.



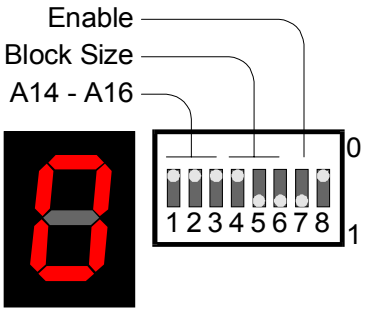
Now the D1 data mapper is enabled again and the block size is set to 2^{7+9} = 65536. The program tests logical addresses 0x0000 through 0xFFFF. D1 translates logical addresses 0x0000 through 0xFEFF to physical addresses for the external SRAM. But logical addresses in the range [0xFF00, 0xFFFF] trigger data mapper D3. D3 is the data mapper with the highest priority and it takes precedence over D1. D3 generates physical addresses that access the *configuration registers* (CRU), so the memory test program starts writing test data into these registers. This is not good. The configuration registers include all the control registers for the code mappers so rewriting these registers will prevent the MCU from accessing the memory where the program instructions are stored. As a result, the program stops running in mid-test and the LED



digit stops with a single segment illuminated. You will have to use FastChip to download the hardware configuration and application program into the CSoC before it operates correctly again.



The block size is reduced to 2^{6+9} = 32768 so the program tests logical addresses 0x0000 through 0x7FFF. The target address bits are changed such that the resulting physical address range put out by data mapper D3 is [0x808000, 0x80FFFF]. The external 128 KByte SRAM on the CSoC Board is only connected to the lower seventeen bits of the physical address, so SRAM addresses during the test are in the range [0x08000, 0x0FFFF]. This is in the valid address range for the external SRAM chip, so the memory test should pass and an 0 will be displayed.



Everything is identical to the previous test except the resulting physical address range put out by the data mapper is moved down to the range [0x800000, 0x807FFF]. The address to the external SRAM is in the range [0x00000, 0x07FFF]. But the instructions for the program are stored in this section of the SRAM so the memory tester writes over its own program. As a result, the program stops running before the interrupt subroutine can even flash the LED digit. So the LED digit will show whatever it was showing when the program was started. You will have to use FastChip to download the hardware configuration and application program into the CSoC before it operates correctly again.

