AN189

Application Note

Interfacing SRAM to EP7xxx Series Microcontrollers

Note: Cirrus Logic assumes no responsibility for the attached information which is provided “AS IS” without warranty of any kind (expressed or implied).

Preliminary Product Information This document contains information for a new product.
Cirrus Logic reserves the right to modify this product without notice.

P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.cirrus.com

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This application note explains how to interface SRAM memory to the bus of Cirrus Logic ARM-based microcontrollers. The bus controller on these devices can interface to x8-, x16-, or x32-bit wide data memory. However, when using a x16 or x32 bus configurations, it requires some external decoding logic.

2. GENERAL DISCUSSION

The EP7xxx microcontrollers are capable of interfacing to a wide variety of SRAM memory devices. Extra SRAM may be required if the internal SRAM provided in the EP7xxx is insufficient for a particular application. SRAM can also be used to hold program memory. Since some SRAM devices can have fast access speeds, that can help to improve overall system performance. However, the ARM architecture does not support unaligned accesses, which means that accessing bytes on non-aligned addresses require an external decoder to access individual "byte-lanes". In most situations, external logic to decode the byte lanes may be required.

Three types of SRAM configurations are shown in Figure 2. The first configuration is for x8 memory. In this case, no extra decoding logic is needed. For reading and storing 8-bit data, this scheme works well as each byte is accessed for each data access cycle, assuming a STRB or LDRB instruction is used (store byte or load byte). Furthermore, the byte data can accessed on any byte-bounded memory location. When using x8 memory for program storage, the processor has to fetch 4 bytes to build up an instruction word (or two bytes in the case of a Thumb instruction) which can slow down performance by a factor of four.

The second SRAM memory bank example in Figure 3 is a x16 SRAM with no byte decoding since both nUB and nLB (upper-byte and lower-byte) signals are tied low. This limits the processor from using byte data since it can only access data on a half-word or on a 16-bit boundary. This means that the programmer must take care to cast all char data as half-word which pads the upper byte with don't cares thereby wasting half of available memory space. But it can also be dangerous, especially when using vendor provided libraries that expect data to be on byte-boundaries. Byte decoding of x16 SRAM may not be necessary when using SRAM to store instructions only. The reason is to take advantage of the faster access times of SRAM verses the slower memory access time of x16 Flash or EPROM. But it will still take two memory fetches for each ARM instruction (or one for Thumb).

It is possible to use x32 SRAM without byte decoding, but it is *not recommended*. In theory, one could use non-decoded x32 SRAM for storing instructions (for speed improvement). But in this case, Thumb instructions must be aligned on word boundaries which is impractical. If using this type of memory for data, then only 32-bit words can be reliably accessed.

In conclusion, it is generally recommended that when using x16 and x32 external SRAM, each byte lane be fully decoded.

3. DESIGN OF THE BYTE DECODERS

The program in Listing 1 (See "Listing 1 Sample Program") is designed to write 4 bytes, 4 half-ints, and 4 ints to a hypothetical memory device. Chip select #5 is used which gives a default address range starting at location 0x5000.0000 (this can be changed by the MMU). The results were sampled using a HP 16500B logic analyzer and is displayed in Figure 1. The signals sampled are: A0 and A1 (address bits), nCS5 (chip select 5), nMOE (output enable), HALFWORD and WORD. From the waveforms, truth tables can be derived and logic synthesized.

The first four access are byte oriented, the next four are word (32-bit) oriented, and the last four are

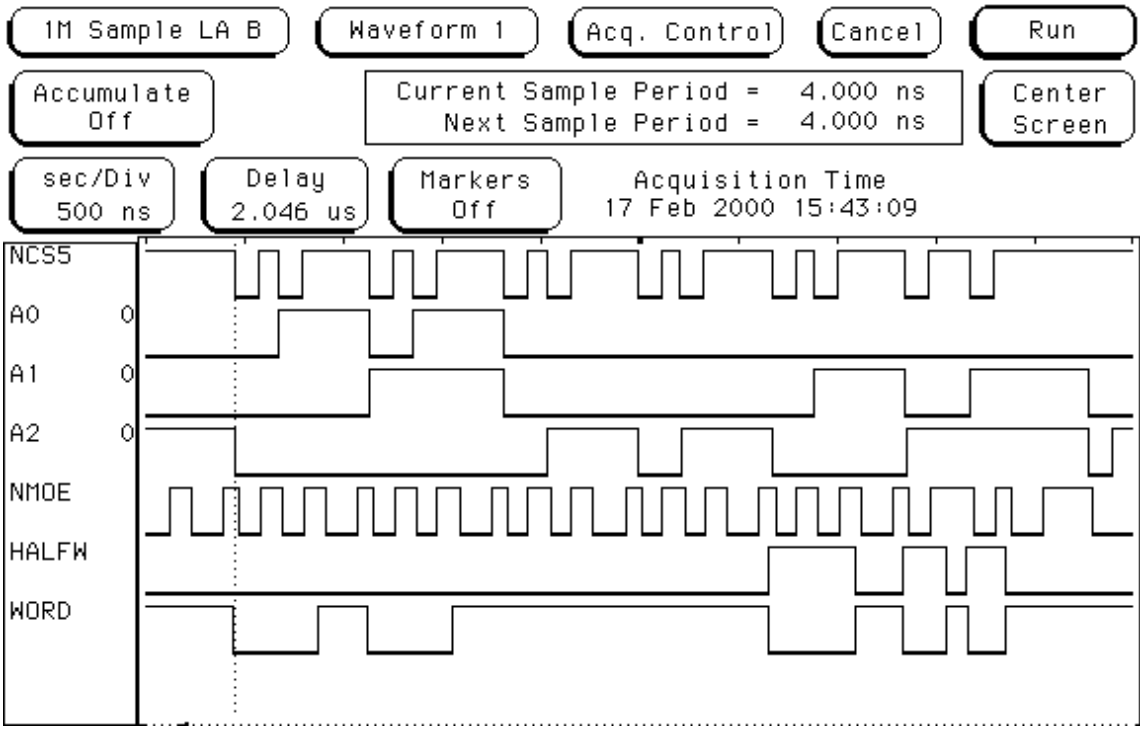

Figure 1. Listing 1 Timing Diagram

half-word accesses. Each access is denoted by a negative signal on NCS5.

3.1 Decoding x16 SRAM

From Figure 1, it can be seen that a truth table as shown in Table 1 can be constructed to provide decoding of byte accesses. In this case, nLB and nUB are signals applied to a x16 SRAM to decode the lower and upper bytes, respectively.

By inspection and by applying DeMorgan's Theorem, the equations for a byte decoder are listed below:

!nLB = !(W + HW + !A0)

!nUP = !(W + HW + A0)

The schematic in Figure 2 has logic that implements the equations above.

ACCESS	WORD	HALF WORD	A0	nLB	nUP
Word	1	X	X	0	0
Half	0	1	X	0	0
Byte	0	0	0	0	1
Byte	0	0	1	1	0

Table 1. Truth Table for x16 Memory Accesses

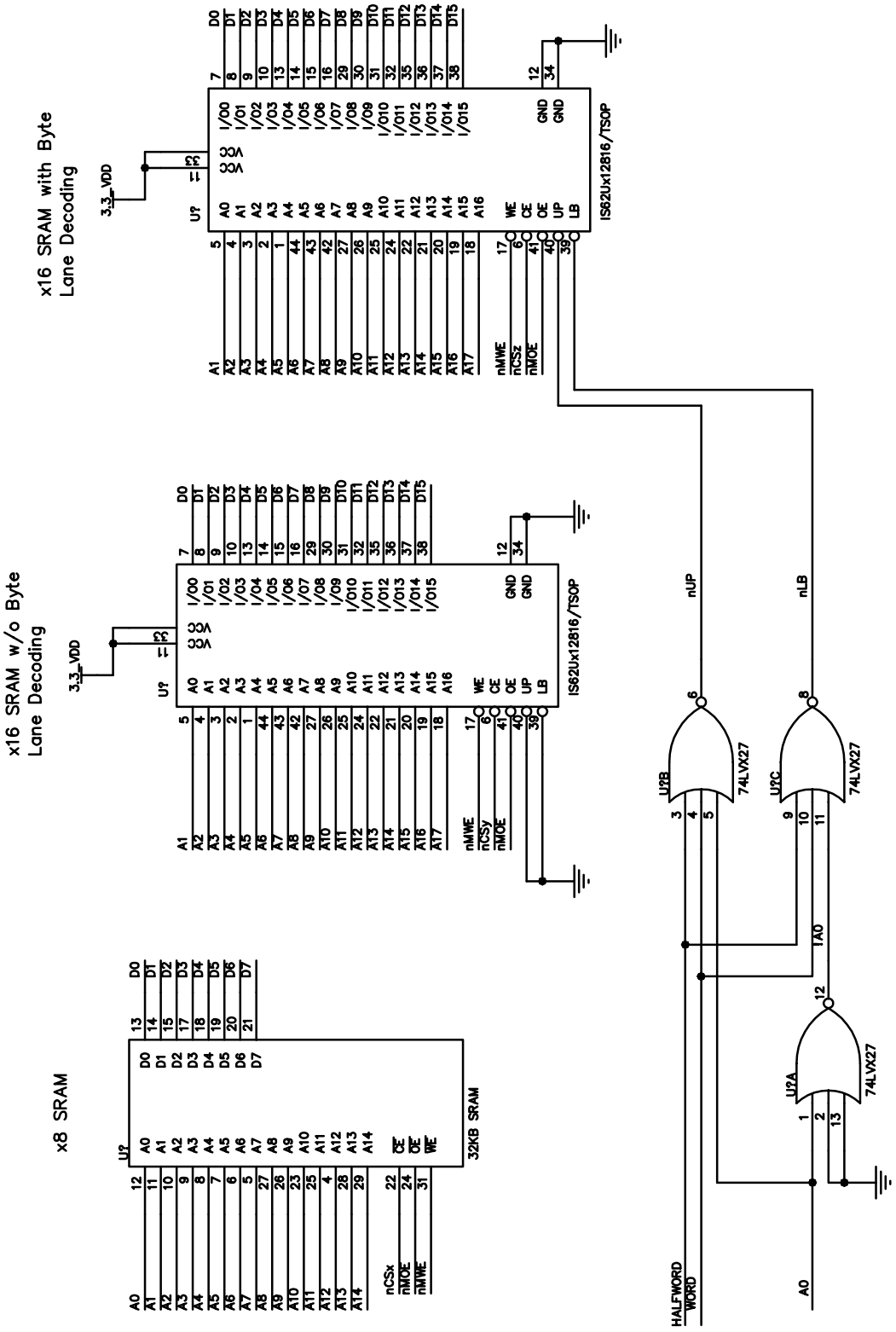

Figure 2. Schematic for Three Types of SRAM Interfaces (Landscape View)

3.2 Decoding x32 SRAM

Typically, two x16 devices would be used to create a x32 SRAM memory block. In this case, A0 and A1 are used to decode the byte lanes along with HALFWORD and WORD. The truth table is listed below in Table 2:

Access	WORD	HALFWORD	A0	A1	nB0	nB1	nB2	nB3
Word	1	X	X	X	0	0	0	0
Lower Half	0	1	X	0	0	0	1	1
Upper Half	0	1	X	1	1	1	0	0
Byte 0	0	0	0	0	0	1	1	1
Byte 1	0	0	1	0	1	0	1	1
Byte 2	0	0	0	1	1	1	0	1
Byte 3	0	0	1	1	1	1	1	0

Table 2. Truth Table for x32 Memory Accesses

The logic to implement the byte decoding is more complicated, but can be easily realized using a low cost PAL or generic logic, such as a GAL16V8 or PAL16L8. The equations are derived using Karnaugh graphs. The equations are not fully reduced to minimum terms. The terms W and HW stand for WORD and HALFWORD, respectively. The schematic in Figure 3 illustrates one method.

For nB0:

	A0, A1			
	00	**01**	**11**	**10**
W, HW **00**	0	1	1	1
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

Table 3. Karnaugh Graph #!

nB0 = (!W * A1) + (!W * !HW * A0)

For nB1:

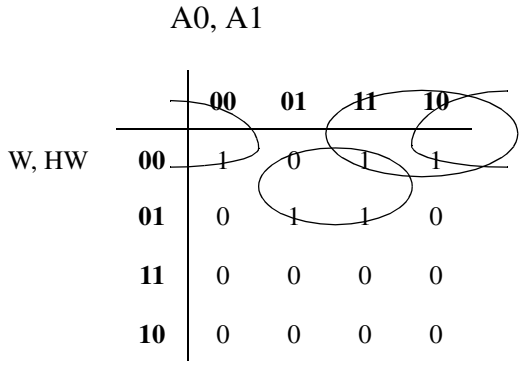

Table 4. Karnaugh Graph #2

nB1 = (!W * HW * !A0) + (!W * !HW * A0) + (!W * !HW * !A1)

For nB2:

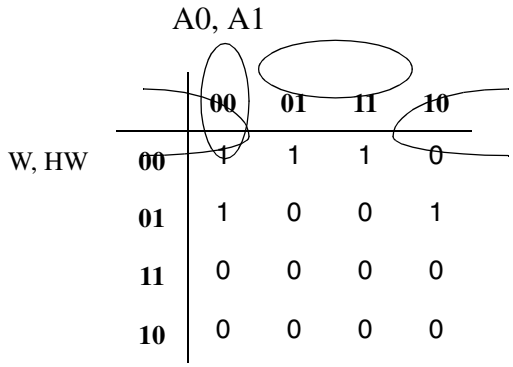

Table 5. Karnaugh Graph #3

nB2 = (!W * HW *!A1) + (!W * !HW * A0) + (!W * !HW * A1)

For nB3:

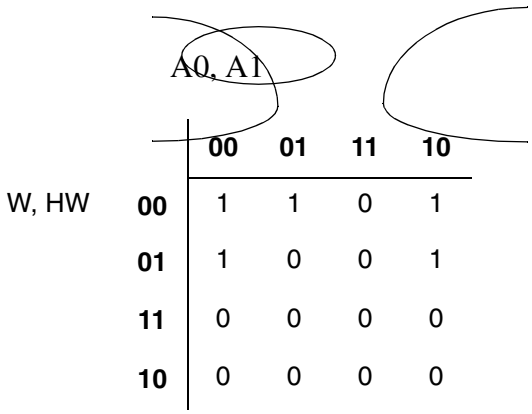

Table 6. Karnaugh Graph #4

nB3 = (!W * !HW * !A0) + (!W * !A1)

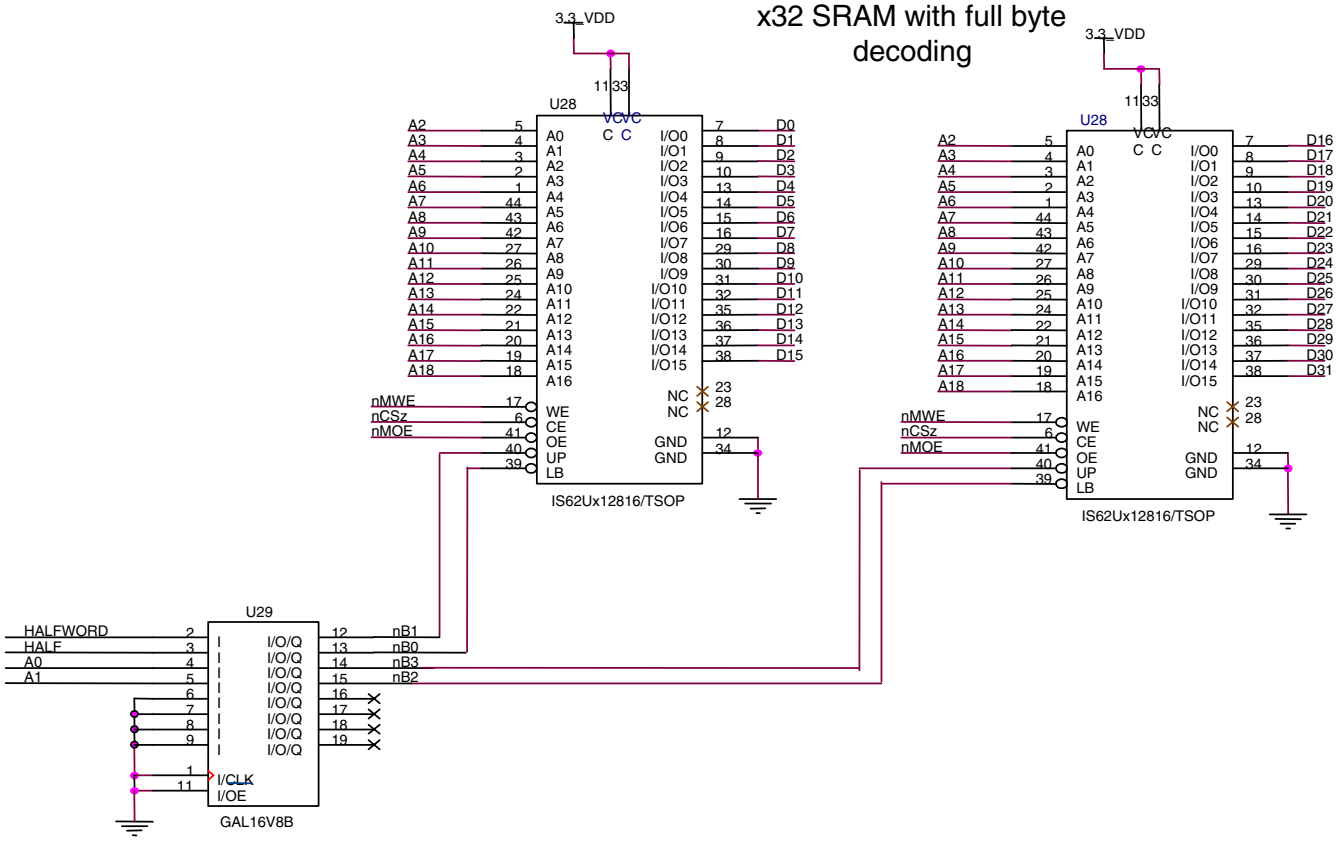

Figure 3. x32 SRAM Interface with Decoding of Four Byte Lanes

4. LISTING 1 SAMPLE PROGRAM

The program set out below in this section was used to read 8, 32, and 16-bit data from the EP72xx. The timing diagram on shown in Figure 1 is a print out from a logic analyzer showing the results.

```
//byte, int, and half word access program
#Define byte unsigned char
int
main(void)
{
byte * mem;
int * mem32;
byte a,b,c,d;
int e,f,g,h;
short * mem16;
short i,j,k,l;
while(1)
{
//Set start address to 0x5000.0000 and read four bytes
mem = (byte*)0x50000000;
a=*mem;
mem++;
b=*mem;
mem++;
c=*mem;
mem++;
d=*mem;
//Next, reset 32 bit pointer and read in four words
mem32 = (int*)0x50000000;
e=*mem32;
mem32++;
f=*mem32;
mem32++;
g=*mem32;
mem32++;
h=*mem32;
//Finally, define a 16-bit pointer and read in four half-words
(short)
mem16 = (short *)0x50000000;
i=*mem16;
mem16++;
j=*mem16;
mem16++;
k=*mem16;
mem16++;
l=*mem16;
}
```


Maverick™
from
CIRRUS LOGIC