

Technical Notes on using Analog Devices' DSP components and development tools

Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

Copyright 1999, Analog Devices, Inc. All rights reserved. Analog Devices assumes no responsibility for customer product design or the use or application of customers' products or for any infringements of patents or rights of others which may result from Analog Devices assistance. All trademarks and logos are property of their respective holders. Information furnished by Analog Devices Applications and Development Tools Engineers is believed to be accurate and reliable, however no responsibility is assumed by Analog Devices regarding the technical accuracy of the content provided in all Analog Devices' Engineer-to-Engineer Notes.

ADSP-218x External Overlay Memory

By G. Y.

What do I do when I need more memory than what is given on the DSP chip?

The ADSP 2181 contains on-chip 16k x 24 program memory and 16k x 16 data memory. What happens when 16k of program memory is not enough for an application? The simple solution is to add more memory. This is where the flexibility of the ADSP 2181 comes in. The ADSP 2181 has the ability to access up to 2 pages of external overlay memory of both 8k for both program memory and data memory. Now, the DSP still contains 16k of program memory and data memory but the user can access up to another 16k of program memory and another 16k of data memory. Refer to **PM/DM Overlay & I/O Memory Expansion Board for ADSP 2181 EZ-KIT Lite** written by G. Yi.

How do I use external overlay memory?

In order to use external overlay memory, one must remember how the memory is organized. Memory organization is controlled by the MMAP pin and the overlay bit. When MMAP=0, the 16k of program memory is divided into two 8k segments, producing an overlay page 0 for internal memory and pages 1 and 2 for external access. The lower segment of program memory, from address 0x0000 to 0x1FFF, is always internal memory. The upper segment of program memory, from address 0x2000 to 0x3FFF, is where the overlay pages reside. The overlay bit controls which overlay page is active. For program memory, the PMOVLAY register controls which program memory overlay the user is accessing. If PMOVLAY=0, the user is accessing internal

memory. If PMOVLAY=1 or 2, the user is accessing external memory. When PMOVLAY=1, the external program memory address range from 0x0000 to 0x1FFF. When PMOVLAY=2, external address range from 0x2000 to 0x3FFF. The same applies toward data memory overlays. The same technique applies to external data overlay memory. Using the DMOVLAY bit, a user can control which overlay page is being accessed. Just like internal memory, external data memory is 16 bits wide and external program memory is 24 bits wide. The difference between data memory overlays and program memory overlays is the memory organization. Instead of accessing the upper portion memory, address 0x2000 to 0x3FFF as the overlay pages, the lower 8k segment of memory is used for data memory overlays.

If All The Data Lines and Address Lines are Multiplexed, How Come Only Addresses 0x2000 to 0x3FFF are used?

When using overlays, only 8k segments can be accessed at one time. The DSP only uses the 13 least significant bits, meaning only address lines A:12 to A:0. The 14th address line determines which overlay page is being used. If overlay page 1 is being used then address line 13 is set to 0. If overlay page 2 is being used then address line 13 is set to 1. This occurs for both program and data memory overlays.

Can I use external memory to run code?

Yes. Just like internal program memory, external program memory is also 24 bits wide, having the ability to fit code in it. External data memory are 16 bits wide having only enough room to fit data. Since external memory share address and data lines, it is the programmer's responsibility to keep track of which overlay page the DSP is accessing. It is hazardous to change the overlay bit while running code in an overlay page. If for any reason, the overlay bit changes while code is running in an

overlay page, the program might continue executing code, in a different overlay page, which might be in the middle of a loop. This will cause errors. For safety reasons, it is advised that only the routines lying in internal memory should be the only ones controlling the overlay bit. After the overlay page is changed, the program can jump to whatever routine is located on that overlay page. Then, after the routine is finished, it should jump back into internal memory.

How can symbols share the same address?

Unlike, data or code, symbols do not actually sit inside the actual memory cell. Symbols are used so that code can reference addresses with names instead of numbers. But now, external overlay memory is involved. As mentioned, external overlays share the same addresses and also, code can be run from external overlay. Because of this, different routines will need to share the same addresses. Since, the linker does not recognize physical overlay memory, it will output errors if different code modules are being linked to the same address. For example, if an application involves three different routines, *routine1*, *routine2*, and *routine3*, all starting at address 0x2000, errors will occur because the linker cannot place three different symbols at the same address. In order to resolve this problem, the newest linker, version 2.3 allows different modules to be linked to the same address. The readme file that comes with the latest linker explains the how the linker is used to have different modules reside at the same address.

How can I load code into external memory?

The new 6.x linker contains additional enhancements that will BDMA transfer routines from the EPROM to internal memory and then transfer the data from internal to external memory. Using the new linker, routines can be linked to the same address and also an overlay table is created in data memory. The table consists of where the routines will lie in the EPROM. The splitter initializes the table with the appropriate data so that the kernel is able to know where to start the BDMA transfer.