Seehau51 Bank switching
Application Notes

(File Bankapps.doc)

As of 12:00 PM 09 May 2001

History:

In the past the MCS-51 architecture was limited to 64k addressable space for CODE and XDATA. Users started running into the boundaries of this micro family. In this case the C-compiler vendors started to struggle to optimize their compilers so that users could stay with-in the constraints of the family's architecture and could not. Henceforth the birth of bank switching (paging a section of the code memory) to allow much larger applications to run on the very popular MCS-51 family of parts.

Nohau was one of the first emulator companies to work with some of these customers to develop away of handling this need. Since then there has been a quasi-standard way of working with banked applications that has been industry wide. The most common is two have the upper 32k page of memory swap between different code pages thus leaving the lower 32k open for the critical functions that need to common to overall operation of a user's application.

Hardware Identification:

Do to this migration the emulator equipment has evolved from the earliest version of the boards, which had many hand done modifications to the E128 emulator board. (Full length 8-bit ISA style of PCB). To the PCB sold today that uses surface mount technology and the PCB is a 2/3^{rds} length 8-bit ISA board.

Older hand-modified Bank switch emulator(s): There we a few types of boards available from Nohau and these were the following:
(*lots of added wires, PALs oversized memory, etc.*)
E128-BSW
E128-BSW-DMA
E256-BSW
E256-BSW-DMA
E320-BSW (**sold only internationally*)

Today's Generation emulator(s):
EA256-BSW
EA256-C320-BSW
EA256-C530-BSW
EA256-MX
EA768-BSW
EA768-C320-BSW
EA768-C530-BSW
EA768-MX

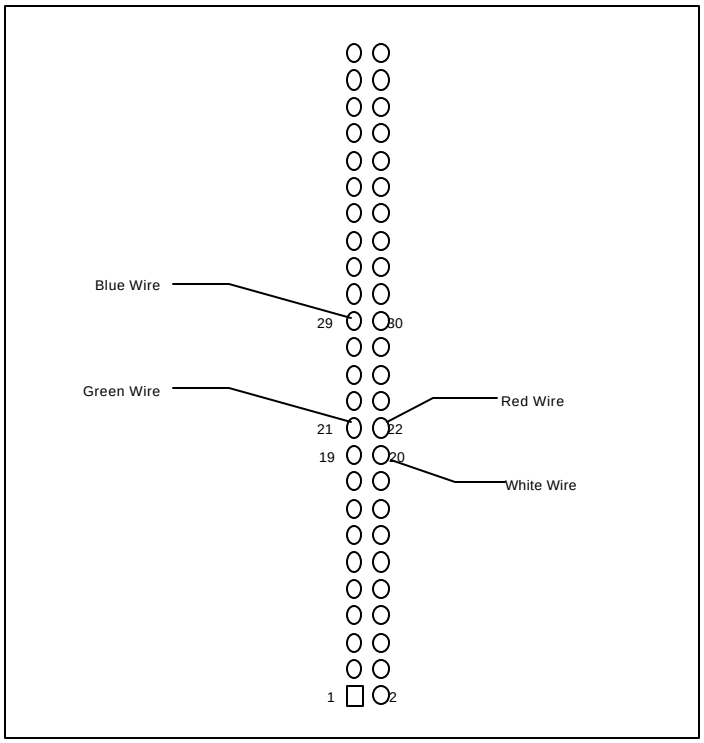
Please refer to your documentation regarding the various memory configurations that are available for the emulator board that you have.

In the case of the option for memory size in the emulator startup most of the older generation bank switch boards are started with the size option set to 32k, except for the E320-BSW which is set for 128k. When using Seehau you will be selecting either the E32 (32k) or E128 (128k) option depending on your board type. With the newer generation of emulators the memory configuration uses the 128k-memory option or the board type option from the Seehau configuration.

Setting the memory operation is critical for proper operation of the emulator. An incorrect operation will cause various problems with code loading and execution.

POD Boards: Most all of the 8051 family pods can be modified to work with bank switching with an exception of a few where the micro itself is not designed for external memory access anyway. The mod mostly includes the addition of wires that are soldered to the bottom side of the POD. These wires are used to connect to the source that drives the extra address lines: **A16, A17, A18, and A19**.

In some cases, for those users of the newer generation emulators that did not purchase the bank switch option to the POD at the time of purchase it may require an additional mod that will be need to be done. (Contact Nohau Technical Support for assistance.)



Definitions of Bank switch connections:

Bank switch Wire color	Bank switch Input bit	Targets chip select or extended Address bit	Connections for Trace inputs, used for breakpoints and triggering on bank specific locations
RED	0	CS0 / A16	E0 [1]
GREEN	1	CS1 / A17	E1 [1]
WHITE	2	CS2 / A18	SY0
BLUE	3	CS3 / A19	SY1

(1) Some of Nohau's pods don't have a labeled connection for E0 and E1, in most cases these two inputs can be found on the trace input jumpers for that have P3 labeled on it. In these cases the user must remove the jumpers for P3 bits 6 and 7 and connect the E0 to P3.6 and E1 to P3.7 .

C' Compilers:

Most of the compilers today have some form of support for the bank switching. Some support both banked code and XDATA memory. Nohau only supports the banked code portion of the application and the target system must support the banked XDATA memory.

When building a banked application it may require linking the application differently than what you would when you are programming an EPROM or FLASH memory to be placed on your target. Basically this means that for each bank you will increment the base address by 0x10000.

There in most cases is an area of code memory that is known as "COMMON" and this is memory that is normally in a non-banked area or is common and in the same location in every bank when dealing with 64k paging. The "COMMON" code would always include the vector table and any other functions that need to be quickly accessed from within any bank without bank switching.

Emulator's Memory for Banked Memory Regions

32k Banks	48k Banks	64k Banks	Bank number	Select bits / Ex/Syx bit pattern
0000-7fff non-banked	0000-3fff non-banked			S S E E Y Y 1 0 1 0 _ _
08000-0ffff	04000-0ffff	00000-0ffff	0	0 0 0 0
18000-1ffff	14000-1ffff	10000-1ffff	1	0 0 0 1
28000-2ffff	24000-2ffff	20000-2ffff	2	0 0 1 0
38000-3ffff		30000-3ffff	3	0 0 1 1
48000-4ffff		40000-4ffff	4	0 1 0 0
58000-5ffff		50000-5ffff	5	0 1 0 1
68000-6ffff		60000-6ffff	6	0 1 1 0
78000-7ffff		70000-7ffff	7	0 1 1 1
88000-8ffff			8	1 0 0 0
98000-9ffff			9	1 0 0 1
A8000-Affff			10	1 0 1 0
B8000-Bffff			11	1 0 1 1
C8000-Cffff			12	1 1 0 0
D8000-Dffff			13	1 1 0 1
E8000-Effff			14	1 1 1 0
F8000-Fffff			15	1 1 1 1

General Operation:

To successfully work with a banked application a user must do the following:

- Set the Emulator board jumpers correctly,
- Startup the emulator with all the associated parameters correctly
- Setup the banking configuration in the emulator software
- Set the memory mapping configuration correctly for the emulator
- Have all associated wire connections for the bank switching setup. (*If you are using a setup that bridges the numbers of available banks but you are not using all the banks that are available in an emulator configuration you should ground the extra bank switch wires so that you do not get spurious switching of banks.*)

Important Considerations:

A) Bank switch control
   1) via microcontroller's port * Does not require a **Shadow Byte**
   2) via an external data memory mapped I/O.
      The memory location that is written to for controlling the banks: can it be read-back with the same value on the associated bits to confirm the bank number?
      - Yes – No **Shadow Byte** required
      - No – A **Shadow Byte** will be required

Question: What is a **Shadow Byte**?
Answer: A **Shadow Byte** it a location in memory that is in either internal chip data memory or off-chip XDATA memory that your program would also have to write to at the time you write to the control register for the bank selection. This would give us the ability to read the current bank.

Question: Do I need to use the **Bank Translation Table**?
Answer: If you have consecutive bits in the byte location and then count up in a normal binary incrementing pattern, then **NO**. If you are not using consecutive bits or you have a special bit pattern for banks then **YES** you will need to enable the table and place the hex value for each bank into the table.

Question: Do I need a **Trace** board to work with a banked application?
Answer: If you don't have a trace then you will not be able to set a breakpoint that will break in the proper bank that you want. The emulator will break in every bank at the same logical address. At that point you would have to go again until you reach the bank you want to really stop at. (Can be automated with a "*Seehau Macro*".)

Question: Do the bank switch signals have to remain at a constant state while inside a bank?
Answer: Yes, if the signals change states while in a bank then the emulators memory will also change banks and you will not be able to execute your code correctly. The banking control lines therefore must be latched to a solid state.

Question: When placing an address in the trigger condition address field what address do I use?
Answer: The Seehau interface will use full symbolic operation, this means that you can place the symbolic function name, or you can use the physical address (this includes the upper nibble of the extended address). Say you have a function at logical address 87F2h that is in bank 3 then the physical address would be 387F2h. When you exit the trace configuration we will automatically setup the Ex and SYx bits correctly base on you bank mask byte.

Question: How do I set the **Control and Status Byte** information in the configuration setup?
Answer: *Address* = the address that your program writes to control the banking lines.
Mask = the byte mask that filters all un-used bits for banking at the byte address. Setting the bit to a **1** says that the bit is used for the banking control line. E.g. you are using bits 3 through 7 for controlling your bank selects the mask in this case would be 38 hex.
Memory Selection Pull-down = Allows you to select either SFR (special function register) or XDATA based on you method of switching banks.

<u>Troubleshooting</u>:

Problem: I load my banked application and it does not look like it loaded correctly.

1) Check to make sure you have connected all the bank control wires from the pod to your target systems device that is controlling the banks.
2) Check to make sure you have enabled the bank switch option in the emulators configuration.
3) If a special bit pattern or shadow byte is required make sure that you have configured these correctly.

Problem: Some banks have the wrong data, they appear to have the same information that is in other banks.

1) Connecting the bank switch wires to the device that is driving the wires.
2) Under the configuration option in the emulator software, disable the bank switching option.
3) Now write a value via the DATA window or the Register window to the address that controls the bank switch wires.

4) Check the wires to see if they have the proper values (high or low) for each bit.
5) Continue checking all bits with different patterns for the output driver.

If some bits will not go to a full high or low.

1) Disconnect the wires from the device that is controlling these lines and retest. If these signals work correctly then the following may be true.
   A) You have an EA 'series emulator and the BS0 or BS1 jumpers are installed on the board. You should remove them.
   B) You have an EA 'series emulator and it has not been modified correctly so that the termination resister pack has been removed from the circuit.
   C) Some pods require additional modification due to some signal conflicts and you need to get this correctly through Nohau. (*If you had the pod purchased from Nohau with the bank switch modification option, then it should have been don already*.)
   D) Hardware failure, you should contact Nohau Technical Support Department for assistance.

If some / all bits don't move at all.

1) Disconnect the wires from the device that is controlling these lines and retest.
2) If these signals work correctly then the following may be true.
   A) Hardware failure, you should contact Nohau Technical Support Department for assistance.
3) If these signal still don't move at all. Either stuck high or low, then the following may be true.
   A) You are using one of the ports on the micro that is being traced and the trace buffer is damaged causing the signal to not be outputted correctly from the micro. Try removing the trace jumper for that port bit(s). At this point if the signal works or still does not work then contact Nohau Technical Support Department for assistance.
   B) You are using a memory mapped I/O device and the programming of this device is either not correct, or the output from the device is damaged.

Problem: I set a break point by clicking in the source / program window and my code does not stop at this address.

1) Check to make sure that you have also connected the E0, E1, SY0 and SY1 trace input bits to each bit that is used for banking. Refer to table that was earlier in this document. If these bits are not connected then the trace can not trigger correctly and thus stop the emulator.
2) You must have a trace to break correctly in a specific bank, otherwise your program will break at the normal 64k logical address for every bank.
3) The code may have never gotten to this point.

Problem: I set a break point and the emulation did stop but it was a little bit further down the code that I expected.

1) Do to the slight delay for the trace to compare the trigger condition and issue the break signal back to the emulator board the execution of the program will slide an instruction or two.