



SPINasm & FV-1 Instruction Set

Assembler/Downloader for SPIN FV-1 Reverb Chip

22 April 2008
SPN1001-ASM-080422

SpinAsm Software Installation	1
Installed SpinAsm Folders.....	3
SPN1001 USB Development Board Installation	5
Plugging in the SPN1001	5
Running SpinAsm	8
The Main Toolbar	8
The Assembler Toolbar	9
SpinAsm Main Window status bar at the bottom of the SpinAsm window:	10
Assembly and testing programs	11
SpinAsm Assembler Errors & Warnings	13
SpinAsm Errors & Warnings.....	13
Warnings	14
SpinAsm Project Mode	17
Preserving Existing Programs in SPN1001 Program Memory	19
Building a Project	20
Chip Internals	21
Register Bank	21
Delay SRAM	21
LFOs.....	22
POTs	22
ADC/DAC	22
ALU	23
Instruction Line Format	24
Operand data types	24
Signed fixed point values	24
Unsigned and signed integers	25
Bit vectors.....	25
Assembler Statements	26
EQU Statement.....	26
MEM Statement	27
The FV-1 Instruction Set	28
Accumulator instructions	30
<i>SOF</i>	30
<i>AND</i>	31
<i>OR</i>	32
<i>XOR</i>	33
<i>LOG</i>	34
<i>EXP</i>	35
<i>SKP</i>	36
Register instructions.....	38
<i>RDAX</i>	38
<i>WRAX</i>	39
<i>MAXX</i>	40
<i>MULX</i>	41
<i>RDFX</i>	42

<i>WRLX</i>	43
<i>WRHX</i>	44
Delay Ram instructions	45
<i>RDA</i>	45
<i>RMPA</i>	46
<i>WRA</i>	47
<i>WRAP</i>	48
LFO instructions	49
<i>WLDS</i>	49
<i>WLDR</i>	50
<i>JAM</i>	51
<i>CHO RDA</i>	52
<i>CHO SOF</i>	54
<i>CHO RDAL</i>	55
Pseudo Opcodes	56
<i>CLR</i>	56
<i>NOT</i>	57
<i>ABSA</i>	58
<i>LDAX</i>	59
Predefined Symbols	60
Change Notes	62

SpinAsm Software Installation

Web or .msi file install:

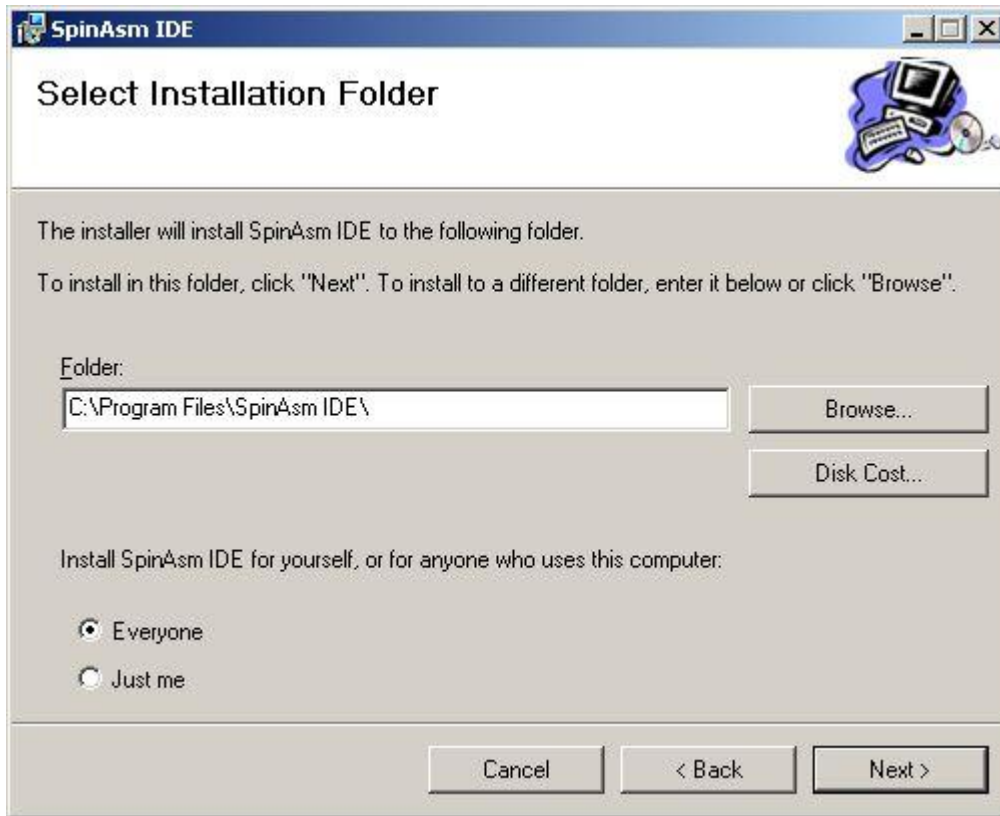
Launch SpinSetup.msi from anywhere.

You will see the “SpinAsm IDE Setup Wizard”.

Windows XP:



Initial SpinAsm software installation dialog.

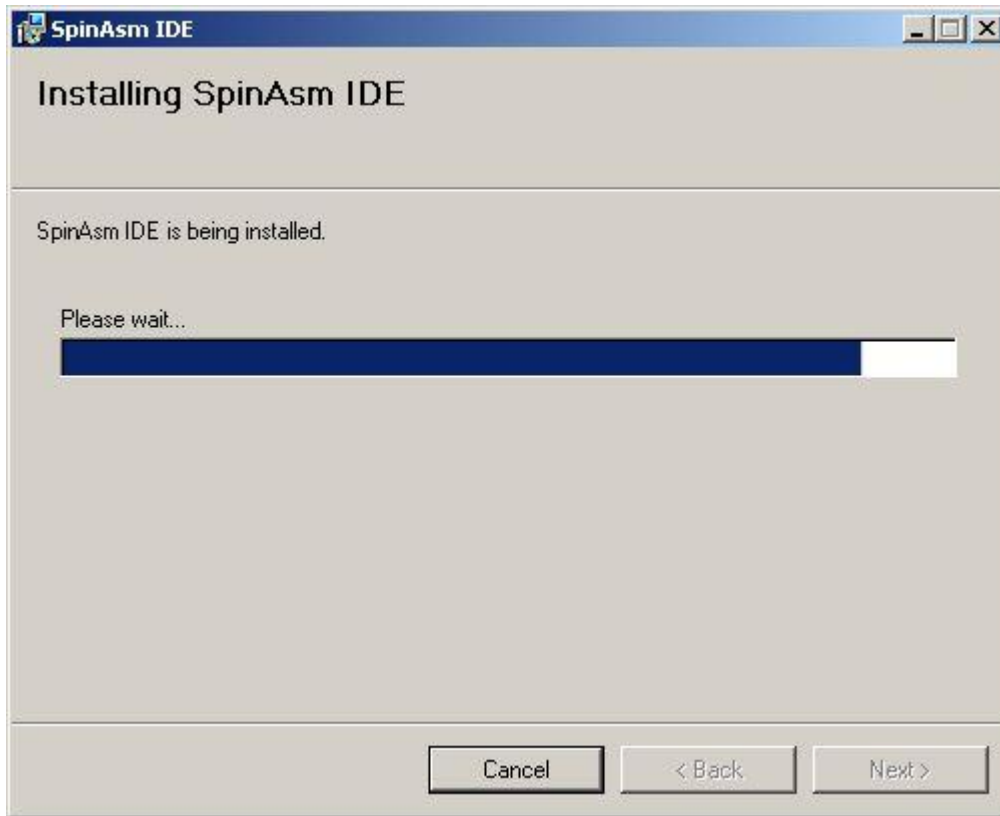


You may install SpinAsm wherever you like. The default “**c:\Program Files\SpinAsm IDE**” folder is used during setup unless you enter an install folder by using the “**Browse**” button or typing it in.

For simplicity, click “**Everyone**” to install SpinAsm for all users on your computer.

Click “**Next**” to proceed. The **Confirm Installation** dialog will be displayed.

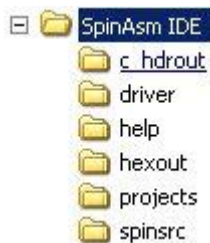
Also click “**Next**” on the “**Confirm Installation**” dialog to proceed.



This dialog is shown during the installation of SpinAsm’s files to your hard disk. Once it has completed SpinAsm software will be installed on your computer. Click the “**Close**” button in the “**Install Complete**” window to complete this phase of the installation.

Installed SpinAsm Folders

When the software installation is complete your SpinAsm folder contains the following folders:



- | | |
|----------|--|
| c_hdrout | Default folder for output of “C” source type header files generated from the <i>Project Build</i> feature. |
| driver | USB driver for the SPN1001-DEVB FV-1 development board. |
| help | SpinAsm help files and other documentation. |
| hexout | Default Folder for Intel© Hex file output from the <i>Project Build</i> feature. |
| projects | Default folder for SpinAsm projects. |

spnsrc

Default folder for SpinAsm source files.

These default directory settings can be changed in the Setup Dialog box.

SPN1001 USB Development Board Installation

If you have the SPN1001 Development board you will have to install the USB driver for it. The driver is located in the **driver** folder of the SpinAsm program folder. For most installations it will be **c:\Program Files\SpinAsm IDE\driver**.

The SPN1001 is actually a USB 2.0 device but will work on any USB 1 or 2 host.

Plugging in the SPN1001

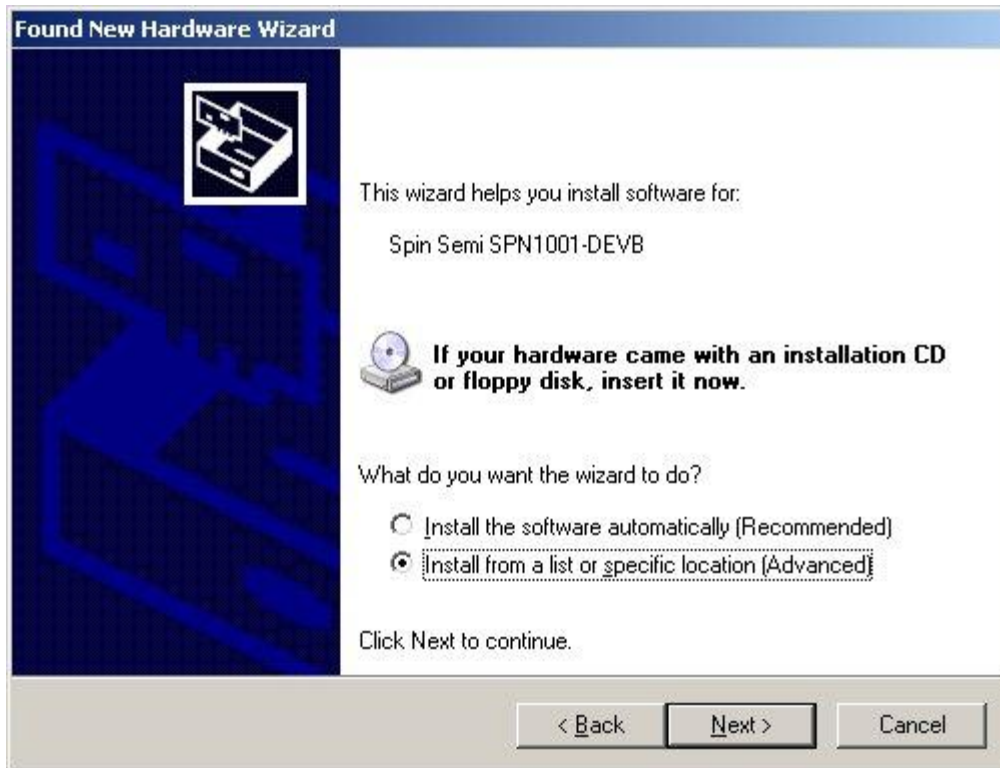
Plug the SPN1001 into your computer using a standard USB cable.

Windows will detect the new USB device:



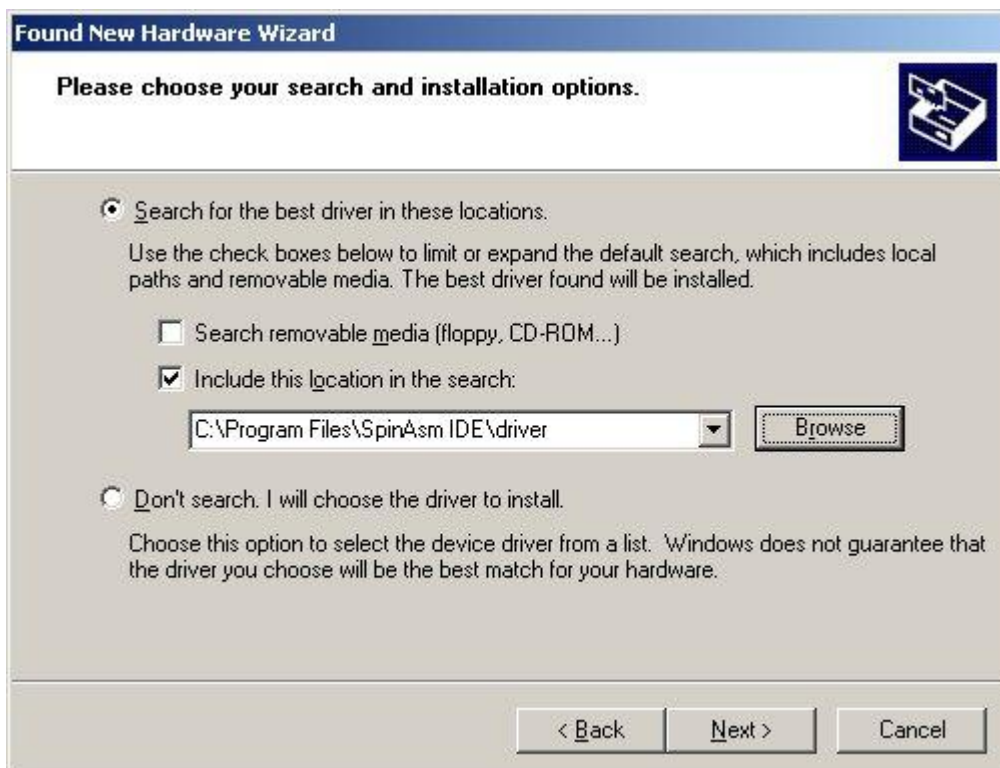
Click **“No, not this time”** to allow you to point the installer to the SpinAsm **“driver”** folder that was created during the SpinAsm software install.

Click **“Next”** to proceed.



Windows knows it is supposed to find the driver for the SPN1001 device. Click on **“Install from list or a specific location”**.

Click **“Next”** to proceed.



Set the options up as shown here. If you have installed the program in a different directory then you may have to use the **"Browse"** button to select the proper location of the **"driver"** folder.

Click **"Next"** to proceed.

The installer's transfer dialog will pop up and you will see this warning:



Click **"Continue Anyway"** to proceed. The installer will begin transferring files and a dialog will display a progress bar during copying. When file transfer and installation are complete you will see the following dialog:



At this point your SPN1001 is ready to use with the SpinAsm IDE to develop and test new programs.

Running SpinAsm



SpinAsm Shortcut

You can launch SpinAsm from the SpinAsm shortcut located on your desktop or you can go to **Start | Programs | SpinAsm IDE** and launch SpinAsm from there.

When SpinAsm starts you will see a blank work surface and two toolbars,

The Main Toolbar



This toolbar contains the standard file new, file open, file save icons.



Go to spinsemi.com

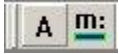


Open the Project Mode Dialog



Open the Setup Dialog

The Assembler Toolbar



(disabled when no source files are loaded)

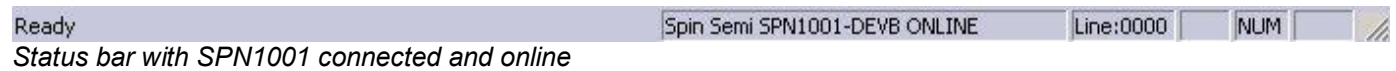


Assemble the current source file



Show the machine code from the last assembly

SpinAsm Main Window status bar at the bottom of the SpinAsm window:



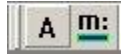
The SpinAsm *Status bar* at the bottom of the main window displays the following information:

1. Status of SPN1001 Development Board. Online or Offline.
2. Progress bars for downloading USB Software to the SPN1001 USB controller
3. Progress bars for sending SpinAsm assembled programs to SPN1001 Program Memory
4. Error Messages from Source Code Assembly during development
5. Editor Line Number in source file being edited.
6. Keyboard Status
 - a. **CAP** Caps Lock
 - b. **NUM** Num Lock
 - c. **SCRL** Scroll Lock
7. General program status and error messages from SpinAsm

With the SPN1001 disconnected you are still able to create and assemble programs for the FV-1. SpinAsm will not attempt to write to the SPN1001 and will simply assemble your program and allow you to debug it.

Assembly and testing programs

You can edit any number of files in SpinAsm at one time simply by loading them in or starting new files. SpinAsm contains a standard bare bones text editor with a single level of *undo* and simple find and replace tools.



Assembly Toolbar



Assemble Button

(disabled when no source files are loaded)

The file which is in the active window is the file which will be assembled when the **Assemble** button is pressed.

Program 0

If you have the SPN1001 development board connected via USB while you assemble SpinAsm will write the output from a successful assembly to the first program slot (prog 0) of the SPN1001 program memory..

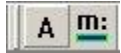
When the SPN1001 **INT – EXT** switch is set to **EXT** the FV-1 will read its programs from the program memory in the socket on the SPN1001. As you write and test programs you will use program 0 to test and modify them. Make sure the program selector switch on the SPN1001 is set to program zero.

After a successful write to the SPN1001 program memory the SPN1001 will toggle the FV-1's INT-EXT line and the FV-1 will load its program memory from the EEPROM.

NOTES:

When the SPN1001 is plugged in and on-line SpinAsm will write the output of successful assemblies into the SPN1001 program memory automatically.

In order to use hear the results this feature you must have the Program Selector switch on the SPN1001 set to Program 0 and the INT-EXT switch set to EXT.



Assembly Toolbar

SpinAsm Output Window

Once SpinAsm begins assembly of a source file an **Output** window will open which will display the results of the assembly.

```
Program Stats for : C:\Program Files\SpinAsm IDE\spinsrc\COPY of FLA_REV_2.spn
NO ERRORS

LABELS:
LOC: 2          Label: LOOP

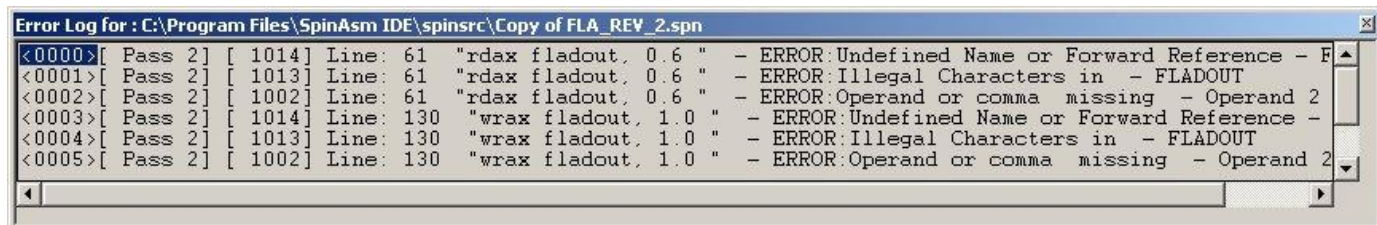
EQUATES:
FLADEL_138     138
FLADEL_139     139
RAMP           32
TRI            34
FLAOUT        35
FLADOUT       36
MIX           37
FBK           38
FRACT        39
K1            40
K2            41
TEMP         42
REVIN        43
RFIL         44

MEMORY MAP:
FLADEL       :0x0000 - 0x01FF size:0x0200 (512)
AP1          :0x0201 - 0x0296 size:0x0096 (150)
AP2          :0x0298 - 0x0374 size:0x00DD (221)
AP3          :0x0376 - 0x04CE size:0x0159 (345)
AP4          :0x04D0 - 0x067E size:0x01AF (431)
RAP1         :0x0680 - 0x0B04 size:0x0485 (1157)
RAP1B        :0x0B06 - 0x13D6 size:0x08D1 (2257)
RAP2         :0x13D8 - 0x1B91 size:0x07BA (1978)
RAP2B        :0x1B93 - 0x2220 size:0x068E (1678)
RAP3         :0x2222 - 0x295B size:0x073A (1850)
RAP3B        :0x295D - 0x32F4 size:0x0998 (2456)
RAP4         :0x32F6 - 0x37C7 size:0x04D2 (1234)
RAP4B        :0x37C9 - 0x3DE7 size:0x061F (1567)
D1           :0x3DE9 - 0x466C size:0x0884 (2180)
D2           :0x466E - 0x55E1 size:0x0F74 (3956)
D3           :0x55E3 - 0x6627 size:0x1045 (4165)
D4           :0x6629 - 0x73A8 size:0x0D80 (3456)
DRAM Memory Unallocated: 3158 bytes
```

A successful assembly

As you can see here a list of LABELS, EQUATES, MEMORY allocations and the available (unallocated) sample memory is displayed.

SpinAsm Assembler Errors & Warnings



Err # Asm Pass Err ID Line Num Source Code Error Description

SpinAsm Error Display

Clicking on **any** part of an error line will bring you to the source code where the error occurred.

SpinAsm Errors & Warnings

General Error	ERR_GENERAL
Program Failure	ERR_PROGRAM_FAIL
Operand or comma missing	ERR_NO_OPERAND
Calc Error in operand 1	ERR_CALCERR
Address out of range	ERR_DELAYADDR_RANGE
Coefficient out of range	ERR_COEFF_RANGE
Address register out of range	ERR_REGISTER_RANGE
Extra operand(s) on line	ERR_EXTRA_OPERAND
Mask bit width out of range	ERR_MASK_RANGE
Too many elements in operand	ERR_OPERAND_SIZE
Bad skip flag USE{ RUN,ZC,Z,GE,N}	ERR_BAD_SKPFLAG
Skip out of range	ERR_SKIP_RANGE
Too Many Math Operators	ERR_EXTRA_MATHOPS
Illegal Characters in	ERR_ILLEGAL_CHARS
Undefined Name or Forward Reference	ERR_FORWARD_REF
Program Length Exceeds Limit	ERR_PROGRAM_LENGTH
Invalid Equate	ERR_INVALID_NAME
Equate Value Error	ERR_EQUATE_VALUE
Non-Alpha Char can not begin Name	ERR_NONALPHA_START
Bad Lfo Value	ERR_BAD_LFOVAL
Invalid Expression	ERR_INVALID_EXPRESSION
Integer Value out of Range	ERR_INT_RANGE
Name Exists as a Label	ERR_NAME_EXISTS_AS_LABEL
Name Exists as Equate	ERR_NAME_EXISTS_AS_EQUATE
Name Exists as Mem Define	ERR_NAME_EXISTS_AS_MEM
Name Exists as Reserved Word	ERR_NAME_EXISTS_AS_RESERVED
Memory Define Error	ERR_MEMORY_ERROR
No Label Text Preceeds Colon	ERR_NO_LABEL
Whitespace in label	ERR_LABEL_WHITESP
SRAM area exceeded	ERR_SRAM_EXCEEDED
Unrecognized or obsolete Opcode	ERR_BAD_OPCODE
FAILED On Pass	ERR_FAILED_PASS
Unimplemented Opcode	ERR_UNIMPLEMENTED_OPCODE

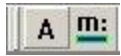
Warnings

Redefinition EQU or MEM

WARN_REDEFINE

Neg & Pos skip flags in SKP Condition

WARN_SKP_FLAGS



Assembly Toolbar

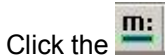



Machine Code Button

```

Machine Code Output for : C:\Program Files\SpinAsm IDE\spinsrc\COPY of FLA_REV_2.spn
0000      80200011      :skp RUN, LOOP
0001      00C01412      :wlds 0, 12, 160
0002 LOOP:
0002      7FEF0244      :rdax pot2, 1.999
0003      000004A6      :wrax mix, 0
0004      40000244      :rdax pot2, 1.0
0005      0000024A      :mulx pot2
0006      000004C6      :wrax fbk, 0
0007      26660484      :rdax fladout, 0.6
0008      000004CA      :mulx fbk
0009      20000284      :rdax adcl, 0.5
0010      200002A4      :rdax adcr, 0.5
0011      00000002      :wra fladel, 0
0012      40000224      :rdax pot1, 1.0
0013      0000022A      :mulx pot1
0014      033300AD      :sof 0.05, 0.005
0015      0010000D      :sof .001, 0
0016      40000404      :rdax ramp, 1.0
Tick      Opcode      Source Code

```



Click the  button If you want to view the actual machine code produced by SpinAsm after an assembly . SpinAsm will display the machine codes listing in the SpinAsm Output Window. You may copy the contents of this window into the clipboard by dragging your mouse to select and typing CTRL+C. Or RIGHT CLICK and **Select All** and then right click again and select **Copy**.

Note, in the SpinAsm project mode this feature will only show the results for the last program assembled in the build. Use it when editing and testing single files as a reference.

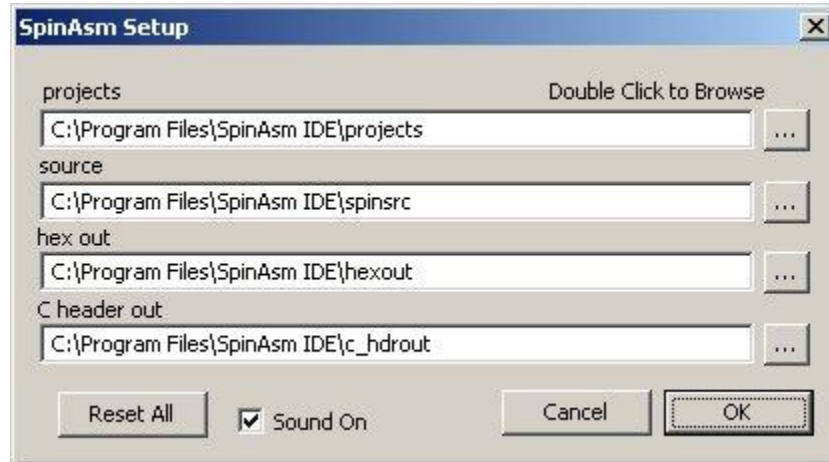
SpinAsm Setup Dialog



Main Toolbar



SpinAsm Setup Dialog



SpinAsm Setup Dialog

Use the browse buttons to set the folders to your desired locations. See (Installed Folders) for a description of each of the folders. You can also double-click on the white areas to browse for a folder.

SpinAsm saves these settings in the spinasm.ini file, rather than the registry, in your windows directory.

Sound On

SpinAsm will beep your computer's speaker when errors occur and at the end of a successful assembly. Use this checkbox to turn those sounds off. All other system sounds will work the same.

Reset All

Reset All will reset the default folders for SpinAsm just as they were when you installed it. It will also reset the locations of any SpinAsm windows to their default positions. Use this feature if you have inadvertently placed the output window or any other off the screen or if you have removed a monitor from your workstation and can no longer view your output or project windows.

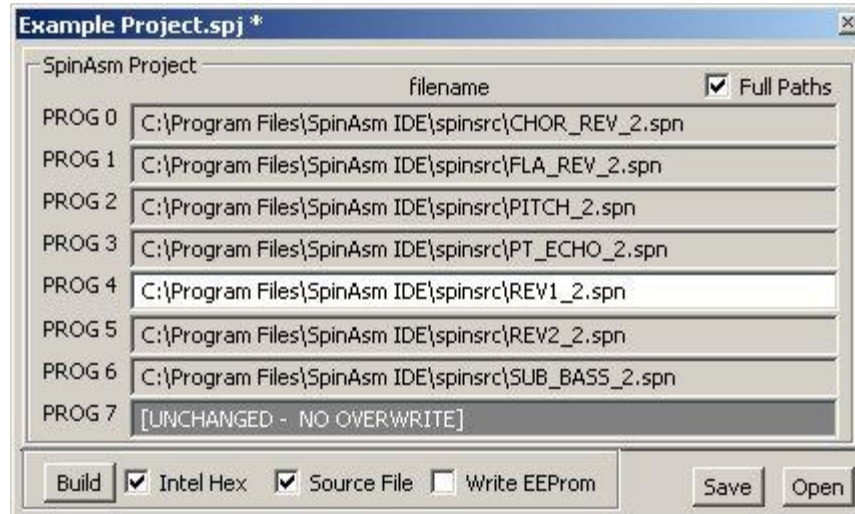
SpinAsm Project Mode



Main Toolbar



SpinAsm Project Mode Dialog



SpinAsm Project Mode Dialog Box.

The Project Mode allows you to organize up to 8 FV-1 programs for writing to the EEPROM program memory on the SPN1001 development board.

From the project mode dialog you can:

- a. Load specific FV-1 programs into program slots 1-8
- b. Clear program slots
- c. Direct the build to generate Intel Hex files and C header type source files.
- d. Enable writing the build to the SPN-1001 program memory.

Each entry in the Project Dialog is a filename which represents a SpinAsm source file. There are eight entries corresponding to the eight program slots in the SPN1001 program memory. When you build your project SpinAsm will load each of the files one at a time and assemble them automatically. If there are any errors the current build source file will stay open and the SpinAsm output window will remain open with the list of errors. As in the normal editing mode, clicking on an error will bring you to the place in the source file where the error is. You can then correct the error and click on the **Build** button to rebuild the project.

Use the **Full Paths** checkbox to show only the filenames **or** the full paths of your source files.



Full Paths unchecked

Project Mode Right-Click Menu

If you right-click on one of the program slots you will see the following menu:



right-click on a program slot

Load File Entry:

This menu selection will allow you to browse your source code files for a SpinAsm source file for that slot. The file you select will be assembled and used during a project build.

Clear File Entry:

This menu selection will only be enabled if there is a filename in the program slot. Selecting this will clear the slot back to **[UNCHANGED - NO OVERWRITE]** see "Preserving Existing Programs".

Edit This File:

The menu selection will only be enabled when there is a file name in the program slot you have right-clicked on. Selecting **Edit This File** will open the source file in the editor. If the file is already open it stays open and is selected for editing.

Preserving Existing Programs in SPN1001 Program Memory

When you open a new project all of the program slots will contain the text:



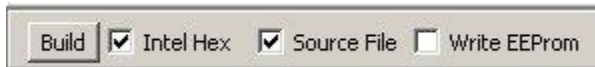
[UNCHANGED - NO OVERWRITE]

When you see the [UNCHANGED – NO OVERWRITE] entry in a program slot it means that when you build your project it will preserve any programs already in those memory locations in the SPN1001 program memory. This feature enables you to build programs without erasing existing programs in the SPN1001 you wish to preserve.

NOTE:

If there is no SPN1001 plugged in any program slots with no file name entry will be built with NOPS. This is because the SPN1001 EEPROM will not exist to be read from so SpinAsm defaults filling those slots with NOPS.

Building a Project



Once you have selected all of the files you want in a particular program group you may build them into an EEPROM image and, if your SPN1001 is connected, write that image to the SPN1001 program memory.

Here are the three choices for a program build:

Intel Hex

Write an Intel hex formatted text file of the build. This will always include all 8 program locations.

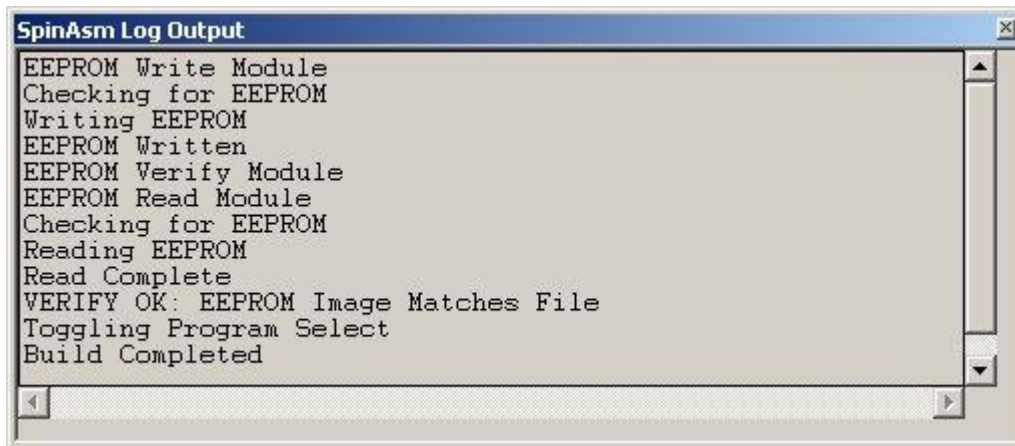
Source File

Write a C formatted header type file with array entries for each program. SpinAsm will separate each program with a new array name.

Write EEPROM

If the SPN1001 board is plugged in the build will be written to the onboard EEPROM

Select your output choices with the checkboxes and click on the **Build** button to begin the build. The SpinAsm **Output** window will open to display the status of the build procedure.



Successful project build with write enabled

As you can see we've kept the output verbose. You will see these messages on the output screen during a build/write cycle. Should you have problems with your production system this output can be helpful in debugging it.

Chip Internals

The FV-1 contains a rich set of features that allows the developer to create exciting effects. These features are described below.

Register Bank

The FV-1 has an internal register bank that provides access to the various I/Os like ADC, DAC, POT inputs, etc. Additionally it has 32 24-bit registers for use as local registers separate from the delay memory. The instruction used determines whether the user is accessing the register bank or the delay memory, instructions that end with and 'X' (RDAX, WRAX, etc.) will access the register bank while instructions that do not end in 'X' (RDA, WRA, etc.) access the delay memory. Please see the instruction set information later in this manual.

Register bank memory map

Address	Name	R/W	Comments
0	SIN0_RATE	W	Write SIN 0 frequency coefficient
1	SIN0_RANGE	W	Write SIN 0 range
2	SIN1_RATE	W	Write SIN 1 frequency coefficient
3	SIN1_RANGE	W	Write SIN 1 range
4	RMP0_RATE	W	Write RMP 0 frequency coefficient
5	RMP0_RANGE	W	Write RMP 0 range
6	RMP1_RATE	W	Write RMP 1 frequency coefficient
7	RMP1_RANGE	W	Write RMP 1 range
8	Not used		
9	Not used		
10	Not used		
11	Not used		
12	Not used		
13	Not used		
14	Not used		
15	Not used		
16	POT0	R	Read POT 0 input
17	POT1	R	Read POT 1 input
18	POT2	R	Read POT 2 input
19	Not used		
20	ADCL	R	Read left ADC input
21	ADCR	R	Read right ADC input
22	DACL	W	Write left DAC output
23	DACR	W	Write right DAC output
24	ADDR_PTR	W	Write address pointer register
25 – 31	Not used		
32 – 63	REG0 – REG31	R/W	24-bit general purpose registers

Delay SRAM

The internal SRAM is configured as 32Kx14. Data is stored in a compressed floating point format, it is expanded to 24-bit fixed point S.23 format after being read and prior to being used in the ALU. The ACC in the ALU can be written to the SRAM, it is converted to the 14-bit floating point format prior to being written to SRAM. The SRAM address is generally calculated by adding the address in the instruction to a down counter that decrements once each sample period and if it is a chorus instruction that is being executed then also the offset from the LFO. As a result of using a down counter, delays are written to the lower address and read from the upper address. I.e. if a 20 sample delay is desired it can be implemented by writing to address 0 and reading from address 20.

LFOs

The FV-1 contains two SIN (LFO0 and LFO1) and two ramp (LFO2 and LFO3) LFOs. The SIN LFOs can be used for effects such as chorus, ring modulators, flange, etc. The ramps can be used for pitch shifting up or down. The SIN LFOs produce both an address offset that is added to the address to the SRAM and a coefficient for use by the ALU multiplier for interpolation between values. The ramp generators generate an address offset, an interpolation coefficient and a cross-fade coefficient to cross fade between the ramp exiting one end of the delay and entering the other end. The ramp can generate appropriate wave forms for pitching up or down based on the sign of the frequency coefficient, positive is pitch up, negative is pitch down.

Coefficients from the LFOs range from 0 to +1.0

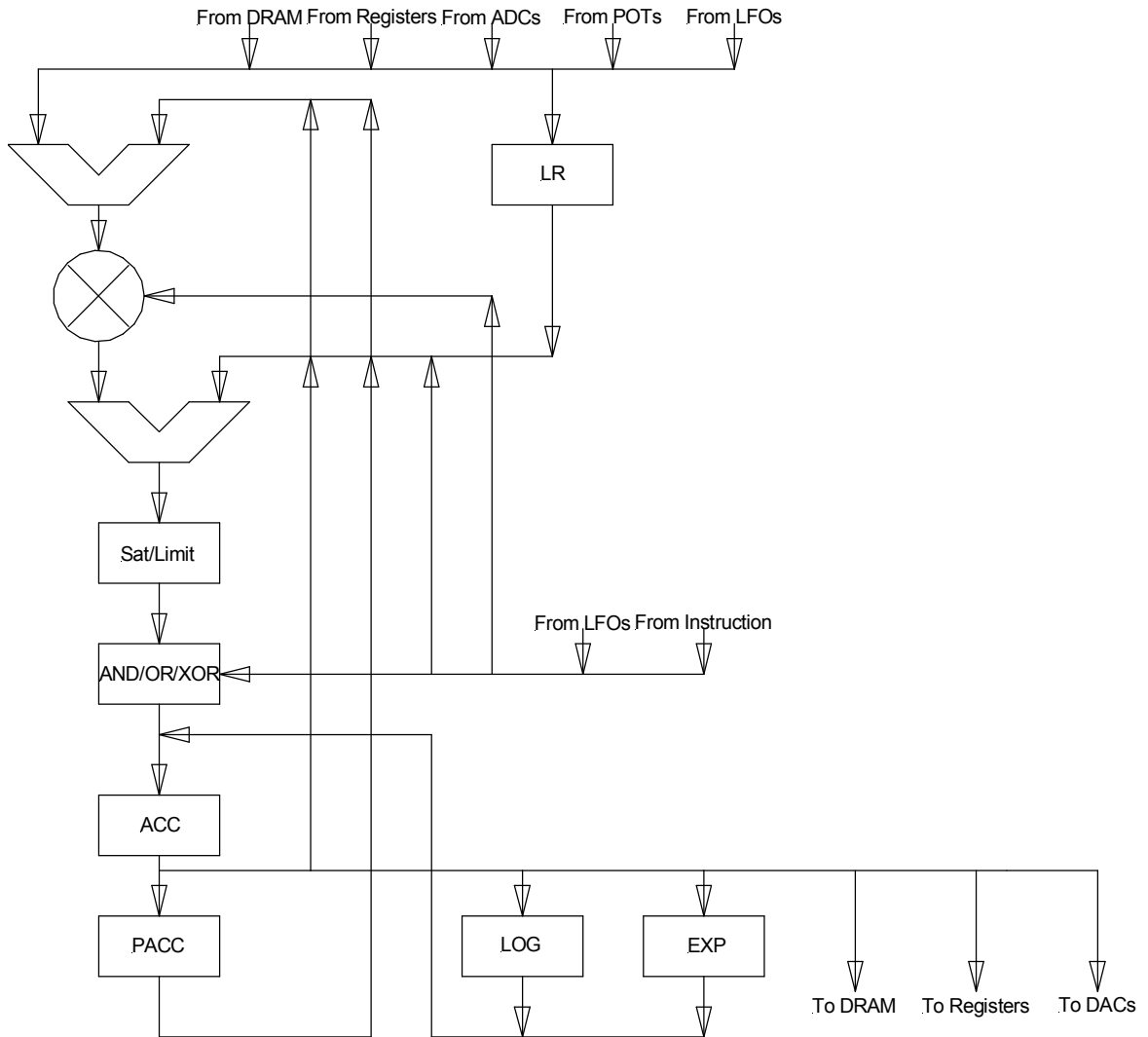
POTs

The chip can read the value of three external potentiometers connected to pins 20, 21 and 22. The pots can be read with approximately a 10-bit resolution and the values can be used as coefficients in programs. The values from the POTs ranges from 0 to +0.99...

ADC/DAC

The internal ADCs provides 24-bit values that ranges from -1.0 to +0.99... Values written to the DAC will also be in the range -1.0 to +0.99...

ALU



The top ALU adder is 25-bits, the 24-bit data from the ADC/SRAM/etc. is sign extended to 25-bits. The multiplier is 25-bits by 16-bits. The 16-bit coefficient actually depends on the instruction being executed. Some instructions only allow for an 11-bit coefficient field, in these cases the coefficient is 0 padded in its LSBs. The format of the coefficient is 2-comp S1.X where X is 14 for a 16-bit coefficient and 9 for an 11-bit coefficient. As a result coefficient range is -2.0 to +1.9...

The top 27-bits from the multiplier are fed into the second adder and the result of the second adder is fed into a saturation-limiter to limit the result to 24-bits in a S.23 format.

The PACC register is the ACC register delayed one state.

Instruction Line Format

The general instruction format within a source line is:

```
[Label:] Opcode(,SubOpcode),Operand1(,Operand2) [;Comment]
```

As indicated by the square brackets, the label and comment fields are optional. The presence of the `SubOpcode` as well as the number and type of operand fields are dependent on the `Opcode` field and will be explained in further detail within the description of the FV-1 instruction set.

[Label:]

Labels can be seen as symbolical representations of instruction lines and are intended to be used as an operand within the SKP instruction. They are allowed either within an instruction line preceding the `Opcode` Field or standalone in a separate line. The label length is limited to 32 characters (blanks are prohibited), the first character must be a letter and each Label must be terminated with a colon.

[;Comment]

Each instruction may be followed by a comment, which must be delimited from the instruction by a semicolon. Since the assembler will ignore all characters from the semicolon to the end of the line, all printable characters are allowed within a comment.

Operand data types

SPINAsm will process three basic operand data types:

- Signed fixed point values
- Unsigned integers
- Bit vectors

For all operand data types SPINAsm performs extensive range checking. Whenever SPINAsm encounters an operand that is out of range, an error message will be displayed indicating the line the error was detected on.

Signed fixed point values

Signed fixed point values are primarily used as coefficients (`Operand2`) for the multiply portion of an instruction. Depending on the actual opcode they may be in one of three different formats, "S1.14", "S1.9" and "S.10".

"S1.14" means that the 16 bit coefficient has one sign bit (MSB), one integer bit left to the binary point followed by 14 fractional bits right to the binary point. "S1.9" denotes an 11 bit coefficient which differs from "S1.14" in that it has fewer bits available to represent the fractional portion of the signed fixed point value (lower resolution). Last but not least the "S.10" format is also a 11 bit coefficient, however in comparison to the "S1.9" format its higher fractional resolution comes at the expense of lacking the integer bit (smaller range). Here's a quick overview regarding range and resolution of the three different coefficient formats.

	Bits	Range	Resolution (LSB value)
S1.14	16	-2 to 1.99993896484	0.00006103516
S1.9	11	-2 to 1.998046875	0.001953125

S.10 11 -1 to 0.9990234375 0.0009765625

Entry formats

SPINAsm allows one to specify signed fixed point values either as real numbers or directly in hexadecimal. Real numbers may have a one digit integer portion, a decimal point, multiple fractional digits and can be prefixed with a "+" or "-" sign. Please note that SPINAsm will round the real decimal number to the nearest LSB value of the required coefficient format

If signed fixed point values are entered in hexadecimal format, they must be prefixed with a "\$" character. Hex values are always assumed to be right justified which means that leading zeros between the "\$" specifier and the first nonzero digit are optional.

S1.14

Real	-2	-0.00006103516	0	0.00006103516	1.99993896484
Hex	\$8000	\$FFFF	\$0000	\$0001	\$7FFF

S1.9

Real	-2	-0.001953125	0	0.001953125	1.998046875
Hex	\$400	\$7FF	\$000	\$001	\$3FF

S.10

Real	-1	-0.0009765625	0	0.0009765625	0.9990234375
-------------	----	---------------	---	--------------	--------------

Please note "S.10" signed fixed point values cannot be entered in hexadecimal.

Unsigned and signed integers

Unsigned integers are primarily used to specify an address (*Operand1*) within an instruction. The (address) range of an unsigned integer is dependent on the actual opcode, specifically whether the instruction will access the delay ram or the internal register file.

The second application for unsigned integers is to specify the number of instructions to be skipped within the SKP instruction. In this case the unsigned integer must be entered in decimal.

Entry formats

Unsigned and signed integers may be entered either in decimal or hexadecimal, in the latter case they must be prefixed by a "\$" character.

Bit vectors

The current FV-1 instruction set supports bit vectors of three different sizes: 5-bit, 6-bit and 24-bit as defined by the individual opcode.

Entry formats

In general bit vectors can be entered in binary representation as a combination of "0" and "1" characters, prefixed with a "%" character. If entered in binary (MSB first), all elements (bit positions) within the bit vector must explicitly be declared, that is a %01001 literal for a 6 bit vector is illegal. To enhance readability especially of 24-bit vectors, underscore characters are allowed after the "%" prefix. Example: %10110001_11111111_00000001.

The second way of entering bit vectors is in hexadecimal format whereas the hex value is treated as being right justified. As an example \$13 will result in a %010011 pattern if applied to a 6 bit vector.

A third method of entering bit vectors is by ORing values together to set particular bits. As an example, "4|1" would result in %000101

Assembler Statements

EQU Statement

The EQU statement allows one to define symbolic operands in order to increase the readability of the source code. Technically an EQU statement such as

```
Name EQU Value           [;Comment]
```

will cause SPINasm to replace any occurrence of the literal "Name" by the literal "Value" within each instruction line during the assembly process excluding the comment portion of an instruction line.

With the exception of blanks, any printable character is allowed within the literal "Name". However there are restrictions: "Name" must be a unique string, is limited to 32 characters and the first character must be a letter excluding the "+" and "-" signs and the "!" character.

The reason for not allowing these characters being the first character of "Name" is that any symbolic operand may be prefixed with a sign or the "!" negation operator within the instruction line. The assembler will then perform the required conversion of the operand while processing the individual instruction lines.

There is another, not syntax related, restriction when using symbolic operands defined by an EQU statement: Predefined symbols. As given in the end of the manual there is a set of predefined symbolic operands which should be omitted as "Name" literals within an EQU statement. It is not that these predefined symbols are prohibited, it is just that using them within an EQU statement will overwrite their predefined value.

With the literal "Value" things are slightly more complicated since its format has to comply with the syntactical rules defined for the operand type it is to represent.

Although it is suggested to place EQU statements at the beginning of the source code file, this is not mandatory. However, the EQU statement has to be defined before the literal "Name" can be used as a symbolical operand within an instruction line.

Remark:

SPINasm has no way of performing range checking while processing the EQU statement. This is because the operand type of value is not known to SPINasm at the time the EQU statement is processed. As a result, range checking is performed when assembling the instruction line in which "Name" is to be replaced by "Value".

Example:

```
Attn      EQU 0.5           ; 0.5 = -6dB attenuation
Tmp_Reg   EQU 63            ; Temporary register within register file
Tmp_Del   EQU $2000        ; Temporary memory location within delay ram
;
;-----
    sof    0,0              ; Clear ACC
    rda   Tmp_Del,Attn     ; Load sample from delay ram $2000,
                          ; multiply it by 0.5 and add ACC content
    wrax  Tmp_Reg,1.0      ; Save result to Tmp_Reg but keep it in ACC
    wrax  DACL,0           ; Move ACC to DAC left (predefined symbol)
                          ; and then clear ACC
```

If `Tmp_Del` was accidentally replaced by `Tmp_Reg` within the `rda` instruction line, SPINasm would not detect this semantic error – simply because using `Tmp_Reg` would be syntactically correct.

If `Tmp_Reg` was mixed up with `Tmp_Del` in the first `wrax` instruction line, a \$2000 value for referencing an internal register would clearly cause a range check error – an appropriate error message would be generated.

MEM Statement

The MEM Statement allows the user to partition the delay ram memory into individual blocks. A memory block declared by the statement

```
Name MEM Value          [;Comment]
```

can be referenced by "Name" from within an instruction line. "Name" has to comply with the same syntactical rules previously defined with the EQU statement, "Size" is an unsigned integer in the range of 1 to 32768 which might be entered either in decimal or in hexadecimal.

Besides the explicit identifier "Name" the assembler defines two additional implicit identifiers, "Name#" and "Name^". "Name" refers to the first memory location within the memory block, whereas "Name#" refers to the last memory location. The identifier "Name^" references the middle of the memory block, or in other words it's center. If a memory block of size 1 is defined, all three identifiers will address the same memory location. In case the memory block is of size 2, "Name" and "Name^" will address the same memory location, if the size is an even number the memory block cannot exactly be halved – the midpoint "Name^" will be calculated as: $\text{size MOD } 2$

Optionally all three identifiers can be offset by a positive or negative integer which is entered in decimal. Although range checking is performed when using offsets, there is no error generated if the result of the address calculation exceeds the address range of the memory block. This is also true for those cases in which the result will "wrap around" the physical 32k boundary of the delay memory. However, a warning will be issued in order to alert the user regarding the out of range condition.

Mapping the memory blocks to their physical delay ram addresses is solely handled by SPINasm. The user has no possibility to explicitly force SPINasm to place a certain memory block to a specific physical address range. This of course does not mean that the user has no control over the layout of the delay ram at all: Knowing that SPINasm will map memory blocks in the order they become defined within the source file, the user can implicitly control the memory map of the delay ram.

Example: (might sound awful)

```
DelR MEM 1024          ; Right channel delay line
DelL MEM 1024          ; Left channel delay line
;
;-----
    sof 0,0            ; Clear ACC
    rdax ADCL,1.0      ; Read in left ADC
    wra DelL,0.25      ; Save it to the start of the left delay
                        ; line and keep a -12dB replica in ACC
    rdax DelL^+20,0.25 ; Add sample from "center of the left delay
                        ; line + 20 samples" times 0.25 to ACC
    rdax DelL#,0.25    ; Add sample from "end of the left delay line
                        ; line" times 0.25 to ACC
    rdax DelL-512,0.25 ; Add sample from "start of the left delay
                        ; line - 512 samples" times 0.25 to ACC
```

Remark:

At this point the result of the address calculation will reference a sample from outside the "DelL" memory block. While being syntactically correct, the instruction might not result in what the user intended. In order to make the user aware of that potential semantic error, a warning will be issued.

```
wrax  DACL,0          ; Result to DACL, clear ACC
      ;
rdax  ADCR,1.0        ; Read in right ADC
wra   DelR,0.25       ; Save it to the start of the right delay
      ; line and keep a -12dB replica in ACC
rdax  DelR^-20,0.25   ; Add sample from center of the right delay
      ; line - 20 samples times 0.25 to ACC
rdax  DelR#,0.25      ; Add sample from end of the right delay line
      ; line times 0.25 to ACC
rdax  DelR-512,0.25   ; Add sample from start of the right delay
      ; line - 512 samples times 0.25 to ACC
```

Remark:

At this point the result of the address calculation will reference a sample from outside the "DelR" memory block. And even worse than the previous case: This time the sample be fetched from delay ram address 32256 which will contain a sample that is apx. 1 second old !
Again, syntactically correct but most likely a semantic error – warnings will be issued.

```
wrax  DACR,0          ; Result to DACR, clear ACC
```

The FV-1 Instruction Set

The instruction set of the FV-1 processor is divided into five basic groups of instructions:

- Accumulator instructions
- Register instructions
- Delay Ram instructions
- LFO instructions
- Pseudo opcodes

FV-1 instructions are 32 bits wide. Except for the more specialized LFO instructions as well as the boolean accumulator instructions, each 32 bit instruction word has to encode its 5 bit opcode, a coefficient and an address specifier.

Within the register instructions only 6 bits are required for addressing the internal register file, the coefficient is 16 bits wide and the remaining 5 bits are reserved and should be set to 0.

Within the delay ram instructions the address portion occupies 16 bits, (although in the current version of the chip only the 15 LSBs are used) accordingly the coefficient is limited to 11 bits.

That means that algorithms requiring higher coefficient resolution (such as high Q IIR filters) should preferably be implemented using the internal general purpose registers as temporary storage locations.

Pseudo opcodes do not add new functionality to the instruction set, all pseudo opcodes could be replaced by the generic instruction(s) they are based upon. All they do is to combine a generic instruction with a special parameter to emulate a more specialized function. For example the FV-1 instruction set features a generic AND MASK function. This one simply performs the "and" function of the current ACC and the

specified 24 bit mask. Clearly, if MASK is \$000000 then ACC becomes cleared and this is exactly what the pseudo opcode "CLR" will do.

Accumulator instructions

SOF

Mnemonic	Operation	Instruction coding
SOF	C * ACC + D	CCCCCCCCCCCCCCCCDDDDDDDDDD01101

Description

SOF will multiply the current value in ACC with C and will then add the constant D to the result.

Please note the absence of an integer entry format for D. This is not by mistake but it should emphasize that D is not intended to become used for integer arithmetic. The reason for this instruction is that the 11 bit constant D would be placed into ACC left justified or in other words 13 bits shifted to the left. D is intended to offset ACC by a constant in the range from -1 to +0.9990234375.

Parameters

Name	Width	Entry formats, range
C	16 Bit	Real (S1.14) Hex (\$0000 - \$FFFF) Symbolic
D	11 Bit	Real(S.10) Symbolic

Syntax

SOF C,D

Coding Example:

```

Off EQU 1.0 ;
;
; Halve way rectifier -----
    sof 0,0 ; Clear ACC
    rdax ADCL,1.0 ; Read from left ADC channel
    sof 1.0,-Off ; Subtract offset
    sof 1.0,Off ; Add offset

```

AND

Mnemonic	Operation	Instruction coding
AND	ACC & MASK	MM000001110

Description

AND will perform a bit wise "and" of the current ACC and the 24-bit MASK specified within the instruction word.

The instruction might be used to load a constant into ACC provided ACC contains \$FFFFFF or to clear ACC if MASK equals \$000000. (see also the pseudo opcode section)

Parameters

Name	Width	Entry formats, range
M	24 Bit	Binary Hex (\$000000 - \$FFFFFF) Symbolic

Syntax

AND M

Coding Example:

```
AMASK EQU    $F0FFFF          ;
;
;-----
    or      $FFFFFF          ; Set all bits within ACC
    and     $FFFFFF          ; Clear LSB
    and     %01111111_11111111_11111111 ; Clear MSB
    and     AMASK            ; Clear ACC[19..16]
    and     $0               ; Clear ACC
```

OR

Mnemonic	Operation	Instruction coding
OR	ACC MASK	MM000001111

Description

OR will perform a bit wise "or" of the current ACC and the 24-bit MASK specified within the instruction word.

The instruction might be used to load a constant into ACC provided ACC contains \$000000.

Parameters

Name	Width	Entry formats, range
M	24 Bit	Binary Hex (\$000000 - \$FFFFFF) Symbolic

Syntax

OR M

Coding Example:

```
OMASK EQU    $0F0000                                ;
;                                                     ;
;-----;
    sof      0,0                                     ; Clear all bits within ACC
    or       $1                                     ; Set LSB
    or       %10000000_00000000_00000000          ; Set MSB
    or       OMASK                                  ; Set ACC[19..16]
    and     %S=[15..8]                              ; Set ACC[15..8]
```

XOR

Mnemonic	Operation	Instruction coding
XOR	ACC ^ MASK	MM000010000

Description

XOR will perform a bit wise "xor" of the current ACC and the 24-bit MASK specified within the instruction word.

The instruction will invert ACC provided MASK equals \$FFFFFF. (see also the pseudo opcode section)

Parameters

Name	Width	Entry formats, range
M	24 Bit	Binary Hex (\$000000 - \$FFFFFF) Symbolic

Syntax

XOR M

Coding Example:

```
XMASK EQU    $AAAAAA                ;
;
;-----
    sof      0,0                      ; Clear all bits within ACC
    xor      $0                        ; Set all ACC bits
    xor      %01010101_01010101_01010101 ; Invert all even numbered bits
    xor      XMASK                     ; Invert all odd numbered bits
```

LOG

Mnemonic	Operation	Instruction coding
LOG	$C * \text{LOG}(\text{ACC}) + D$	CCCCCCCCCCCCCCCCDDDDDDDDDDDD01011

Description

LOG will multiply the Base2 LOG of the current absolute value in ACC with C and add the constant D to the result.

It is important to note that the LOG function returns a fixed point number in S4.19 format instead of the standard S.23 format, which in turn means that the most negative Base2 LOG value is -16.

The LOG instruction can handle absolute linear accumulator values from 0.99999988 to 0.00001526 which translates to a dynamic range of apx. 96dB.

D an offset to be added to the logarithmic value in the range of -16 to + 15.999998.

Parameters

Name	Width	Entry formats, range
C	16 Bit	Real (S1.14) Hex (\$0000 - \$FFFF) Symbolic
D	11 Bit	Real(S4.6) Symbolic

Syntax

LOG C,D

Coding Example:

log 1.0,0

EXP

Mnemonic	Operation	Instruction coding
EXP	$C * EXP(ACC) + D$	CCCCCCCCCCCCCCCCDDDDDDDDDD01100

Description

EXP will multiply 2^{ACC} with C and add the constant D to the result.

Since ACC (in it's role as the destination for the EXP instruction) is limited to linear values from 0 to +0.9999988, the EXP instruction is limited to logarithmic ACC values (in it's role as the source operand for the EXP instruction) from -16 to 0. Like the LOG instruction, EXP will treat the ACC content as a S4.19 number. Positive logarithmic ACC values will be clipped to +0.9999988 which is the most positive linear value that can be represented within the accumulator.

D is intended to allow the linear ACC to be offset by a constant in the range from -1 to +0.9990234375

Parameters

Name	Width	Entry formats, range
C	16 Bit	Real (S1.14) Hex (\$0000 - \$FFFF) Symbolic
D	11 Bit	Real(S.10) Symbolic

Syntax

EXP C,D

Coding Example:

```
exp 0.8,0
```

SKP

Mnemonic	Operation	Instruction coding
SKP	CMASK N	CCCCN>NNNNN000000000000000010001

Description

The SKP instruction allows conditional program execution. The FV-1 features five condition flags that can be used to conditionally skip the next N instructions. The selection of which condition flag(s) must be asserted in order to skip the next N instructions is made by the five bit condition mask "CMASK". Only if all condition flags that correspond to a logic "1" within CMASK are asserted are the following N instructions skipped. The individual bits within CMASK correspond to the FV-1 condition flags as follows:

CMASK	Flag	Description
b4	RUN	The RUN flag is cleared after the program has executed for the first time after it was loaded into the internal program memory. The purpose of the RUN flag is to allow the program to initialize registers and LFOs during the first sample iteration then to skip those initializations from then on.
b3	ZRC	The ZRC flag is asserted if the sign of ACC and PACC is different, a condition that indicates a Zero Crossing.
b2	ZRO	Z is asserted if ACC = 0
b1	GEZ	GEZ is asserted if ACC >= 0
b0	NEG	N is asserted if ACC is negative

Parameters

Name	Width	Entry formats, range
CMASK	5 Bit	Binary Hex (\$00 - \$1F) Symbolic
N	6 Bit	Decimal (1 – 63) Label

Maybe the most efficient way to define the condition mask is using it's symbolic representation. In order to simplify the SKP syntax, SPINasm has a predefined set of symbols which correspond to the name of the individual condition flags. (RUN,ZRC,ZRO,GEZ,NEG). Although most of the condition flags are mutually exclusive, SPINasm allows you to specify more than one condition flag to become evaluated simply by separating multiple predefined symbols by the "|" character. Accordingly "skp ZRC|N, 6" would skip the following six instructions in case of a zero crossing to a negative value.

Syntax

SKP CMASK,N

Coding Example:

```
; A bridge rectifier          ;
                               ;
    sof  0,0                    ; Clear ACC
    rdax ADCL,1.0              ; Read from left ADC channel
    skp  GEZ,pos               ; Skip next instruction if ACC >= 0
    sof  -1.0,0                ; Make ACC positive
pos: wrax DACL,0               ; Result to DACL, clear ACC
    rdax ADCL,1.0              ; Read from left ADC channel
    skp  N,neg                 ; Skip next instruction if ACC < 0
```

```
      sof  -1.0,0      ; Make ACC negative
pos:  wrax 0,DACR     ; Result to DACR, clear ACC
```


Register instructions

RDAX

Mnemonic	Operation	Instruction coding
RDAX	C * REG[ADDR] + ACC	CCCCCCCCCCCCCCCC00000AAAAA00100

Description

RDAX will fetch the value contained in [ADDR] from the register file, multiply it with C and add the result to the previous content of ACC. This multiply accumulate is probably the most popular operation found in DSP algorithms.

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic
C	16 Bit	Real (S1.14) Hex (\$8000 - \$0000 - \$7FFF) Symbolic

In order to simplify the RDAX syntax, see the list of predefined symbols for all registers within the FV-1 register file.

Syntax

RDAX ADDR,C

Coding Example:

```

; Crude mono
;
;
; Clear ACC
rdax ADCL,0.5 ; Get ADCL value and divide it by two
rdax ADCR,0.5 ; Get ADCR value, divide it by two
; and add to the half of ADCL
wrx DACL,1.0 ; Result to DACL
wrx DACR,0 ; Result to DACR and clear ACC

```

WRAX

Mnemonic	Operation	Instruction coding
WRAX	ACC->REG[ADDR], C * ACC	CCCCCCCCCCCCCCCC00000AAAAA00110

Description

WRAX will save the current value in ACC to [ADDR] and then multiply ACC by C. This instruction can be used to write ACC to one DAC channel while clearing ACC for processing the next audio channel.

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic
C	16 Bit	Real (S1.14) Hex (\$8000 - \$0000 - \$7FFF) Symbolic

In order to simplify the WRAX syntax, see the list of predefined symbols for all registers within the FV-1.

Syntax

WRAX ADDR,C

Coding Example:

```

; Stereo processing
;
;
rdax ADCL,1.0 ; Read left ADC into previously cleared ACC
;-----
.... ; ...left channel
.... ; processing...
;-----
wrax DACL,0 ; Result to DACL and clear ACC for right
; channel processing
rdax ADCR,1.0 ; Read right ADC into previously cleared ACC
;-----
.... ; ...right channel
.... ; processing...
;-----
wrax DACR,0 ; Result to DACR and clear ACC for left
; channel processing

```

MAXX

Mnemonic	Operation	Instruction coding
MAXX	MAX(REG[ADDR] * C , ACC)	CCCCCCCCCCCCCCCC00000AAAAAA01001

Description

MAXX will compare the absolute value of ACC versus C times the absolute value of the register pointed to by ADDR. If the absolute value of ACC is larger ACC will be loaded with |ACC|, otherwise the accumulator becomes overwritten by |REG[ADDR] * C|.

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic
C	16 Bit	Real (S1.14) Hex (\$8000 - \$0000 - \$7FFF) Symbolic

In order to simplify the MAXX syntax, see the list of predefined symbols for all registers within the FV-1 register file.

Syntax

MAXX ADDR,C

Coding Example:

```

; Peak follower
Peak      EQU    32          ; Peak hold register
;
;-----
;
      sof    0,0            ; Clear ACC
      rdax  ADCL,1.0       ; Read left ADC
      maxx Peak,1.0       ; Keep larger absolute value in ACC

; For a peak meter insert decay code here...

      wrax  Peak,0         ; Save (new) peak and clear ACC

```


RDFX

Mnemonic	Operation	Instruction coding
RDFX	$(ACC-REG[ADDR])*C + REG[ADDR]$	CCCCCCCCCCCCCCCC00000AAAAA00101

Description

RDFX will subtract the value of the register pointed to by ADDR from ACC, multiply the result by C and then add the value of the register pointed to by ADDR. RDFX is an extremely powerful instruction in that it represents the major portion of a single order low pass filter.

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic
C	16 Bit	Real (S1.14) Hex (\$8000 - \$0000 - \$7FFF) Symbolic

In order to simplify the RDFX syntax, see the list of predefined symbols for all registers within the FV-1 register file.

Syntax

RDFX ADDR C

Coding Example:

```

; Single order LP filter
Tmp_LP EQU 32 ; Temporary register for first order LP
;
;-----
;
ldax ADCL ; Read left ADC
rdfx Tmp_LP,x.x ; First order...
wrax Tmp_LP,1.0 ; ...LP filter
wrax DACL,0 ; Result to DACL and clear ACC

```

WRLX

Mnemonic	Operation	Instruction coding
WRLX	ACC->REG[ADDR], (PACC-ACC) * C + PACC	CCCCCCCCCCCCCCCC00000AAAAA01000

Description

First the current ACC value is stored into the register pointed to by ADDR, then ACC is subtracted from the previous content of ACC (PACC). The difference is then multiplied by C and finally PACC is added to the result. WRLX is an extremely powerful instruction in that when combined with RDFX, it forms a single order low pass shelving filter

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic
C	16 Bit	Real (S1.14) Hex (\$8000 - \$0000 - \$7FFF) Symbolic

In order to simplify the WRLX syntax, see the list of predefined symbols for all registers within the FV-1 register file.

Syntax

WRLX ADDR,C

Coding Example:

```
; Single order LP shelving filter
Tmp_LP      EQU    32          ; Temporary register for first order LP
;-----
;
    sof      0,0              ; Clear ACC
    rdax     ADCL,1.0         ; Read left ADC
    rdfx     Tmp_LP,x.x       ; First order LP...
    wrlx     Tmp_LP,y.y       ; ...shelving filter
    wrax     DACL,1.0         ; Result to DACL and clear ACC
```

WRHX

Mnemonic	Operation	Instruction coding
WRHX	ACC->REG[ADDR], (ACC*C) + PACC	CCCCCCCCCCCCCCCC00000AAAAA00111

Description

The current ACC value is stored in the register pointed to by ADDR, then ACC is multiplied by C. Finally the previous content of ACC (PACC) is added to the product. WRHX is an extremely powerful instruction in that when combined with RDFX, it forms a single order high pass shelving filter.

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic
C	16 Bit	Real (S1.14) Hex (\$8000 - \$0000 - \$7FFF) Symbolic

In order to simplify the WRHX syntax, see the list of predefined symbols for all registers within the FV-1 register file.

Syntax

WRHX ADDR,C

Coding Example:

```
; Single order HP shelving filter
Tmp_HP      EQU    32          ; Temporary register for first order HP
;
;-----
;
    sof      0,0              ; Clear ACC
    rdax     ADCL,1.0         ; Read left ADC
    rdfx     Tmp_HP,x.x      ; First order HP...
    wrhx     Tmp_HP,y.y      ; ...shelving filter
    wrax     DACL,0          ; Result to DACL and clear ACC
```

Delay Ram instructions

RDA

Mnemonic	Operation	Instruction coding
RDA	SRAM[ADDR] * C + ACC	CCCCCCCCCCCCAAAAAAAAAAAAAAAA0000

Description

RDA will fetch the sample [ADDR] from the delay ram, multiply it by C and add the result to the previous content of ACC. This multiply accumulate is probably the most popular operation found in DSP algorithms.

Parameters

Name	Width	Entry formats, range
ADDR	(1)+15 Bit	Decimal(0 – 32767) Hex(\$0 - \$7FFF) Symbolic
C	11 Bit	Real (S1.9) Hex (\$400 - \$000 - \$3FF) Symbolic

Syntax

RDA ADDR,C

Coding Example:

```

Delay MEM    1024           ;
Coeff EQU   1.55           ;
Tmp EQU     $2000          ;
               ;
    rda     1000,1.9        ;
    rda     Delay+20,Coeff  ;
    rda     Tmp,-2         ;
    rda     $7FFF,$7FF     ;

```


RMPA

Mnemonic	Operation	Instruction coding
RMPA	SRAM[PNTR[N]] * C + ACC	CCCCCCCC000000000110000001

Description

RMPA provides indirect delay line addressing in that the delay line address of the sample to be multiplied by C is not explicitly given in the instruction itself but contained within the pointer register ADDR_PTR (absolute address 24 within the internal register file.)

RMPA will fetch the indirectly addressed sample from the delay ram, multiply it by C and add the result to the previous content of ACC.

Parameters

Name	Width	Entry formats, range
C	11 Bit	Real (S1.9) Hex (\$400 - \$000 - \$3FF) Symbolic

Syntax

RMPA C

Coding Example:

```
; Crude variable delay line addressing

    sof    0,0           ; Clear ACC
    rdax  POT1,1.0      ; Read POT1 value
    wrax  ADDR_PTR,0    ; Write value to pointer register, clear ACC
    rmpa  1.0           ; Read sample from delay line
    wrax  DACL,0        ; Result to DACL and clear ACC
```

WRA

Mnemonic	Operation	Instruction coding
WRA	ACC->SRAM[ADDR], ACC * C	CCCCCCCCAA00010

Description

WRA will store ACC to the delay ram location addressed by ADDR and then multiply ACC by C.

Parameters

Name	Width	Entry formats, range
ADDR	(1)+15 Bit	Decimal(0 – 32767) Hex(\$0 - \$7FFF) Symbolic
C	11 Bit	Real (S1.9) Hex (\$400 - \$000 - \$3FF) Symbolic

Syntax

WRA ADDR,C

Coding Example:

```
Delay MEM    1024      ;
Coeff EQU    0.5      ;

    sof    0,0          ; Clear ACC
    rdax   ADCL,1.0     ; Read left ADC
    wra    Delay,Coeff  ; Write to start of delay line, halve ACC
    rda    Delay#,Coeff ; Add half of the sample from
                        ; the end of the delay line
    wrax   DACL,0       ; Result to DACL and clear ACC
```

WRAP

Mnemonic	Operation	Instruction coding
WRAP	ACC->SRAM[ADDR], (ACC*C) + LR	CCCCCCCCAA00011

Description

WRAP will store ACC to the delay ram location addressed by ADDR then multiply ACC by C and finally add the content of the LR register to the product. Please note that the LR register contains the last sample value read from the delay ram memory. This instruction is typically used for all-pass filters in a reverb program.

Parameters

Name	Width	Entry formats, range
ADDR	(1)+15 Bit	Decimal(0 – 32767) Hex(\$0 - \$7FFF) Symbolic
C	11 Bit	Real (S1.9) Hex (\$400 - \$000 - \$3FF) Symbolic

Syntax

WRAP ADDR,C

Coding Example:

```
rda  ap1#,kap ; Read output of all-pass 1 and multiply it by kap
wrap ap1,-kap ; Write ACC to input of all-pass 1 and do
                ; ACC*(-kap)+ap1# (ap1# is in LR register)
```

LFO instructions

WLDS

Mnemonic	Operation	Instruction coding
WLDS	See Description	00NFFFFFFFFFAAAAAAAAAAAAAA10010

Description

WLDS will load frequency and amplitude control values into the selected SIN LFO (0 or 1). This instruction is intended to setup the selected SIN LFO which is typically done within the first sample iteration after a new program is loaded. As a result WLDS will in most cases be used in combination with a SKP RUN instruction. For a more detailed description regarding the frequency and amplitude control values see application note AN-0001.

Parameters

Name	Width	Entry formats, range
N	1 Bit	SIN LFO select: (0, 1)
F	9 Bit	Decimal(0 – 511) Hex (\$000 - \$1FF) Symbolic
A	15 Bit	Decimal(0 – 32767) Hex (\$0000 - \$7FFF) Symbolic

Syntax

WLDS N,F,A

Coding Example:

```

Amp EQU 8194 ; Amplitude for a 4097 sample delay line
Freq EQU 51 ; Apx. 2Hz at 32kHz sampling rate
;-----
; Setup SIN LFO 0 ;
    skp run,start ; Skip next instruction if not first iteration
    wlds 0,Freq,Amp ; Setup SIN LFO 0
;
start: sof 0,0 ;
    ....
    ....

```

WLDR

Mnemonic	Operation	Instruction coding
WLDR	See Description	01NFFFFFFFFFFFFFFFFFFFF000000AA10010

Description

WLDR will load frequency and amplitude control values into the selected RAMP LFO. (0 or 1) This instruction is intended to setup the selected RAMP LFO which is typically done within the first sample iteration after a new program became loaded. As a result WLDR will in most cases be used in combination with a SKP RUN instruction. For a more detailed description regarding the frequency and amplitude control values see application note AN-0001.

Parameters

Name	Width	Entry formats, range
N	1 Bit	RAMP LFO select: 0, 1
F	16 Bit	Decimal(-16384 – 32767) Hex (\$4000 - \$000 - \$7FFF) Symbolic
A	2 Bit	Decimal (512, 1024, 2048, 4096) Symbolic

Syntax

WLDR N,F,A

Coding Example:

```
Amp EQU 4096 ; LFO will modulate a 4096 samples delay line
Freq EQU $100 ;
;-----
; Setup RAMP LFO 0 ;
    skp run, start ; Skip next instruction if not first iteration
    wldr 0,Freq,Amp ; Setup RAMP LFO 0
;
start: and 0 ;
    ....
    ....
```

JAM

Mnemonic	Operation	Instruction coding
JAM	0 -> RAMP LFO N	000000000000000000000000000000001N010011

Description

JAM will reset the selected RAMP LFO to its starting point.

Parameters

Name	Width	Entry formats, range
N	1 Bit	RAMP LFO select: 0, 1

Syntax

JAM N

Coding Example:

```
jam 0 ; Force ramp 0 LFO to it's starting osition
```

CHO RDA

Mnemonic	Operation	Instruction coding
CHO RDA	See Description	00CCCCC0NNAAAAAAAAAAAAAAAAA10100

Description

Like the RDA instruction, CHO RDA will read a sample from the delay ram, multiply it by a coefficient and add the product to the previous content of ACC. However, in contrast to RDA the coefficient is not explicitly embedded within the instruction and the effective delay ram address is not solely determined by the address parameter. Instead, both values are modulated by the selected LFO at run time, for an in depth explanation please consult the FV-1 datasheet alongside with application note AN-0001. CHO RDA is a very flexible and powerful instruction, especially useful for delay line modulation effects such as chorus or pitch shifting.

The coefficient field of the "CHO" instructions are used as control bits to select various aspects of the LFO. These bits can be set using predefined flags that are ORed together to create the required bit field. For a sine wave LFO (SIN0 or SIN1), valid flags are:

SIN COS REG COMPC COMPA

While for a ramp LFO (RMP0 and RMP1), valid flags are:

REG COMPC COMPA RPTR2 NA

These flags are defined as:

Flag	HEX value	Description
SIN	\$0	Select SIN output (default) (Sine LFO only)
COS	\$1	Select COS output (Sine LFO only)
REG	\$2	Save the output of the LFO into an internal LFO register.
COMPC	\$4	Complement the coefficient (1-coeff)
COMPA	\$8	Complement the address offset from the LFO
RPTR2	\$10	Select the ramp+1/2 pointer (Ramp LFO only)
NA	\$20	Select x-fade coefficient and do not add address offset

Parameters

Name	Width	Entry formats, range
N	2 Bit	LFO select: SIN0, SIN1, RMP0, RMP1
C	6 Bit	Binary Bit flags
ADDR	(1)+15 Bit	Decimal(0 – 32767) Hex(\$0 - \$7FFF) Symbolic

Syntax

CHO RDA,N,C,ADDR

Coding Example:

```
; A chorus
```

```
Delay MEM 4097 ; Chorus delay line
Amp EQU 8195 ; Amplitude for a 4097 sample delay line
Freq EQU 51 ; Apx. 2Hz at 32kHz sampling rate
;-----;
```

```

; Setup SIN LFO 0
    skp    run,start          ; Skip if not first iteration
    wlds   0,Freq,Amp        ; Setup SIN LFO 0
start:
    sof    0,0               ; Clear ACC
    rdax   ADCL,0.5          ; Read left ADC * 0.5
    wra    Delay,1.0         ; Write to chorus delay line
    cho rda,SIN0,REG|COMPC,Delay^ ; See application note AN-0001
    cho rda,SIN0,,Delay^+1   ; for detailed examples & explanation
    wrax   DACL,0           ; Result to DACL and clear ACC

```


CHO SOF

Mnemonic	Operation	Instruction coding
CHO SOF	See Description	10CCCCC0NNDDDDDDDDDDDDDDDD10100

Description

Like the SOF instruction, CHO SOF will multiply ACC by a coefficient and add the constant D to the result. However, in contrast to SOF the coefficient is not explicitly embedded within the instruction. Instead, based on the selected LFO and the 6 bit vector C, the coefficient is picked from a list of possible coefficients available within the LFO block of the FV-1. For an in depth explanation please consult the FV-1 datasheet alongside with application note AN-0001. CHO SOF is a very flexible and powerful instruction, especially useful for the cross fading portion of pitch shift algorithms.

Please see "CHO RDA" for a description of field flags.

Parameters

Name	Width	Entry formats, range
N	2 Bit	LFO select: SIN0, SIN1, RMP0, RMP1
C	6 Bit	Binary Bit flags
D	16 Bit	Real(S.15) Symbolic

Syntax

CHO SOF,N,C,D

Coding Example:

```

; Pitch shift

Delay MEM 4096 ; Pitch shift delay line
Temp MEM 1 ; Temporary storage
Amp EQU 4096 ; RAMP LFO amplitude (4096 samples)
Freq EQU -8192 ; RAMP LFO frequency
;-----;
;
; Setup RAMP LFO 0
;
; skip run,cont ; Skip if not first iteration
; wldr 0,Freq,Amp ; Setup SIN LFO 0
;
cont:
;
; sof 0,0 ; Clear ACC
; rdax ADCL,1.0 ; Read left ADC * 1.0
; wra Delay,0 ; Write to delay line, clear ACC
; cho rda,RMP0,COMPC|REG,Delay ; See application note AN-0001
; cho rda,RMP0,,Delay+1 ; for detailed examples & explanation
; wra Temp,0 ;
; cho rda,RMP0,COMPC|RPTR2,Delay ;
; cho rda,RMP0,RPTR2,Delay+1 ;
; cho sof,RMP0,NA|COMPC,0 ;
; cho rda,RMP0,NA,Temp ;
; wrax DACL,0 ; Result to DACL and clear ACC

```

CHO RDAL

Mnemonic	Operation	Instruction coding
CHO RDAL	LFO * 1 -> ACC	110000100NN00000000000000010100

Description

CHO RDAL will read the current value of the selected LFO into ACC.

Parameters

Name	Width	Entry formats, range
N	2 Bit	LFO select: SIN0,COS0,SIN1,COS1,RMP0,RMP1

Syntax

CHO RDAL,N

Coding Example:

```
cho rdal,SIN0           ; Read LFO S0 into ACC
wrax  DACL,0           ; Result to DACL and clear ACC
```

Pseudo Opcodes

CLR

Mnemonic	Operation	Instruction coding
CLR	0 -> ACC	00000000000000000000000000000000000001110

Description

CLR will clear the accumulator.

Parameters **None**

Syntax

CLR

Coding Example:

```
clr                ; Clear ACC
rdax  ADCL,1.0    ; Read left ADC
                  ;-----
....             ; ...left channel
....             ; processing...
                  ;-----
wrax  DACL,0      ; Result to DACL and clear ACC
```

NOT

Mnemonic	Operation	Instruction coding
NOT	/ACC -> ACC	1111111111111111111111111111111100010000

Description

NOT will negate all bit positions within accumulator thus performing a 1's complement.

Parameters None

Syntax

NOT

Coding Example:

```
not                                   ; 1's comp ACC
```

ABSA

Mnemonic	Operation	Instruction coding
ABSA	ACC -> ACC	00000000000000000000000001001

Description

Loads the accumulator with the absolute value of the accumulator.

Parameters None

Syntax

ABSA

Coding Example:

```
absa                                   ; Absolute value of ACC -> ACC
```

LDAX

Mnemonic	Operation	Instruction coding
LDAX	REG[ADDR]-> ACC	000000000000000000000000000000101

Description

Loads the accumulator with the contents of the addressed register.

Parameters

Name	Width	Entry formats, range
ADDR	6 Bit	Decimal(0 – 63) Hex(\$0 - \$3F) Symbolic

Syntax

LDAX REG

Coding Example:

```
ldax adcl           ; ADC left input -> ACC
```

Predefined Symbols

Following is the list of predefined symbols in the SPINAsm assembler:

Symbol	Value: hex (dec)	Notes
SIN0_RATE	0x00 (0)	SIN 0 rate
SIN0_RANGE	0x01 (1)	SIN 0 range
SIN1_RATE	0x02 (2)	SIN 1 rate
SIN1_RANGE	0x03 (3)	SIN 1 range
RMP0_RATE	0x04 (4)	RMP 0 rate
RMP0_RANGE	0x05 (5)	RMP 0 range
RMP1_RATE	0x06 (6)	RMP 1 rate
RMP1_RANGE	0x07 (7)	RMP 1 range
POT0	0x10 (16)	Pot 0 input register
POT1	0x11 (17)	Pot 1 input register
POT2	0x12 (18)	Pot 2 input register
ADCL	0x14 (20)	ADC input register left channel
ADCR	0x15 (21)	ADC input register right channel
DACL	0x16 (22)	DAC output register left channel
DACR	0x17 (23)	DAC output register right channel
ADDR_PTR	0x18 (24)	Used with 'RMPA' instruction for indirect read
REG0	0x20 (32)	Register 00
REG1	0x21 (33)	Register 01
REG2	0x22 (34)	Register 02
REG3	0x23 (35)	Register 03
REG4	0x24 (36)	Register 04
REG5	0x25 (37)	Register 05
REG6	0x26 (38)	Register 06
REG7	0x27 (39)	Register 07
REG8	0x28 (40)	Register 08
REG9	0x29 (41)	Register 09
REG10	0x2A (42)	Register 10
REG11	0x2B (43)	Register 11
REG12	0x2C (44)	Register 12
REG13	0x2D (45)	Register 13
REG14	0x2E (46)	Register 14
REG15	0x2F (47)	Register 15
REG16	0x30 (48)	Register 16
REG17	0x31 (49)	Register 17
REG18	0x32 (50)	Register 18
REG19	0x33 (51)	Register 19
REG20	0x34 (52)	Register 20
REG21	0x35 (53)	Register 21
REG22	0x36 (54)	Register 22
REG23	0x37 (55)	Register 23
REG24	0x38 (56)	Register 24
REG25	0x39 (57)	Register 25
REG26	0x3A (58)	Register 26
REG27	0x3B (59)	Register 27
REG28	0x3C (60)	Register 28
REG29	0x3D (61)	Register 29
REG30	0x3E (62)	Register 30
REG31	0x3F (63)	Register 31
SIN0	0x00 (0)	USED with 'CHO' instruction: SINE LFO 0

SIN1	0x01 (1)	USED with 'CHO' instruction: SINE LFO 1
RMP0	0x02 (2)	USED with 'CHO' instruction: RAMP LFO 0
RMP1	0x03 (3)	USED with 'CHO' instruction: RAMP LFO 1
RDA	0x00 (0)	USED with 'CHO' instruction: ACC += (SRAM * COEFF)
SOF	0x02 (2)	USED with 'CHO' instruction: ACC = (ACC * LFO COEFF) + Constant
RDAL	0x03 (3)	USED with 'CHO' instruction: Reads value of selected LFO into the ACC
SIN	0x00 (0)	USED with 'CHO' instruction: SIN/COS from SINE LFO
COS	0x01 (1)	USED with 'CHO' instruction: SIN/COS from SINE LFO
REG	0x02 (2)	USED with 'CHO' instruction: Save LFO temp reg in LFO block
COMPC	0x04 (4)	USED with 'CHO' instruction: 2's comp : Generate 1-x for interpolate
COMPA	0x08 (8)	USED with 'CHO' instruction: 1's comp address offset (Generate SIN or COS)
RPTR2	0x10 (16)	USED with 'CHO' instruction: Add 1/2 to ramp to generate 2nd ramp for pitch shift
NA	0x20 (32)	USED with 'CHO' instruction: Do NOT add LFO to address and select cross-fade coefficient
RUN	0x80000000	USED with 'SKP' instruction: Skip if NOT FIRST time through program
ZRC	0x40000000	USED with 'SKP' instruction: Skip On Zero Crossing
ZRO	0x20000000	USED with 'SKP' instruction: Skip if ACC = 0
GEZ	0x10000000	USED with 'SKP' instruction: Skip if ACC is' >= 0'
NEG	0x80000000	USED with 'SKP' instruction: Skip if ACC is Negative

Change Notes

- 11 July 2006: First release
- 28 August 2006: Fixed Typo in "cho rdal" example, change "S0" to "SIN0"
- 15 November 2006: Added COSX to "cho rdal" to allow COS outputs to be read as well as SIN, SpinAsm updated to support syntax.
Fixed other instances of SX/RX to SINX/RMPX
DRAM references updated to SRAM
- 22 April 2008: Fixed JAM instruction, typo in SpinAsm doc and assembly error in SpinAsm.

Notice

Spin Semiconductor reserves the right to make changes to, or to discontinue availability of, any product or service without notice.

Spin Semiconductor assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using any Spin Semiconductor product or service. To minimize the risks associated with customer products or applications, customers should provide adequate design and operating safeguards.

Spin Semiconductor make no warranty, expressed or implied, of the fitness of any product or service for any particular application.

Contact Information

Spin Semiconductor
Phone: (310) 417-4956
Web: www.spinsemi.com

Mailing:
Spin Semiconductor
c/o OCT Distribution
6504 1/2 Arizona Ave.
Los Angeles, CA 90045