SXSim

A Simulator for the SX Family of Microcontrollers for Microsoft Windows®

Version 2.08.06

by Günther Daubach

Im Eulenflug 25

D-51399 Burscheid, Germany

Tel./Fax: +49 2174 – 785 931

eMail: g.daubach@mda-burscheid.de

Table of Contents

What is SXSim?

SXSim is a software-emulation of the SX family of microcontrollers for PCs running the Windows operating system (Win 98 and higher).

SXSim has a “built-in virtual” SX controller that is able to execute the commands contained in a list file (LST) created by the current version of SASM which is integral part of the SX-Key development software released by Parallax Inc. Please note that SXSim may not work correctly with LST files that were created by other versions, or other types of SX assemblers because SXSim makes certain assumptions on the structure of the LST files, unique to the SX-Key SASM.

SXSim is not only intended to be a simulator for the SX devices but it may also be used as an analyzer for SX application programs, extending the debugging features of /n in-system SX debugger, where on the other hand, SXSim lacks features of the in-system debugger, especially the real-time execution speed.

SXSim is – by no means – yet finished, or fully tested, so don’t be surprised if certain SX features are not correctly supported, or are not supported at all. Nevertheless, this version of SXSim should give you an idea of what can be done. The current version of SXSim does support all SX devices, i.e. SX 18/20/28/48/52, but not yet the two additional timer/counters available in the SX48/52 devices.

Why SXSim?

On the SXTech List forum, I once found a message that read: “The best SX simulator is the SX itself together with SX-Key”. I fully agree to that statement because one of the unique features of the SX is that it can be debugged “on-line”, or “in-system” at real-time. But there are always some matters that ask for improvements, like:

- Whenever you make a change in the SX program, the SX program memory must be re-programmed for the next debugging session – with SXSim, you simply re-load the list file after re-assembling the source code file, and you are ready for a new simulation session.
- Due to the internal structure of the SX, you can only define one breakpoint at a time with SX-Key or any other SX development system. SXSim allows you to define as many breakpoints as you like, and it automatically saves the breakpoint positions (plus other important project-related information), so that they are there again when you re-load a LST file later.
- Due to the internal structure of the SX, you can not set a breakpoint on a NOP instruction – SXSim makes it possible.
- Due to the internal structure of the SX, the “breakpointed” instruction is always executed before the program execution stops. This means that breakpoints on JMP or CALL instructions actually halt the program execution on the first instruction at the JMP target, or at the first instruction of a subroutine. SXSim – on the other hand – stops program execution before the “breakpointed” instruction is executed.
- SXSim displays some additional, useful information, like the status of the port direction registers, the stack memory, and the number of executed cycles, and the elapsed execution time at a specified clock frequency. It also analyzes the clock cycles of the ISR code, and displays the contents of the SX “secret registers” which will be described later in this document.
- SXSim performs checks for subroutine stack underflow and overflow conditions. This helps you to find out if your code contains RET instructions without prior CALL instructions, or if your subroutines are nested too deeply. It also displays a “Stack

gauge", which gives you an idea of how deeply you have nested subroutine calls in a program.

- SXSim performs some other plausibility checks that can't be done by in-system debuggers, like jumps to un-programmed areas in program memory. These might help you, detecting abnormal operation conditions in a program that are hard to be found while debugging the "real silicon".
- SXSim comes with a "Walk" mode, i.e. a "slow motion" mode where you may select the speed. You can also select whether active breakpoints shall stop the "Walk" mode, or not. Both settings are saved in the SIM file, and will be restored when you re-load a project.
- SXSim has a "Step Over" function – this is a "Hyper Step" button, allowing you to skip over subroutines in high speed while stepping through a program.
- When testing an SX application, you sometimes might want to know "what happened before". SXSim records the last 1000 instructions executed, allowing you to "un-do" them, i.e. to "reverse-execute" the application under test.
- Many SX applications make use of delay timers, and while debugging a program, it is annoying to step or walk over such delay loops, or setting breakpoints after such loops, and "Run"ing them. Therefore, in SXSim you may assign the "Hyper Counter" property to a file register. When this property is active, the registers will immediately over-, or underflow as soon as an increment or decrement instruction is performed on them.
- In programs with interrupts on RTCC roll-overs, it is often annoying when the interrupt service routine is entered while single-stepping. Instead of setting a breakpoint on the RETI or RETIW instruction, and "Running" over the ISR code, in SXSim, you have the option to let it automatically execute the ISR code at full speed.
- SXSim also comes with a "smart" I/O panel configuring itself, depending on how the TRIS port registers are configured by the simulated program. When a port pin is configured as input, the panel activates a "virtual input button", when a pin is configured as output, the panel activates a "virtual LED". Input buttons can be locked if necessary in order to set more than one input pin to high level at the same time.

  The I/O Panel also has a special "virtual input button" that becomes active when the SX is configured to have the RTCC incremented by external pulses.

As you can see, SXSim comes with some features that are not available with "real-silicon" debuggers. Nevertheless, SXSim's target is not to replace such debuggers, like the SX-Key. It has been designed as a utility for those users who like to "test before buy" a "real" SX development system, and for SX developers who want to "test before flash" an application.

You should also keep in mind that due to the fact that the SX is simulated on a PC, the program execution is a lot slower compared to a "real" SX, i.e. SXSim can't by no means run an application in real-time as it is possible with the SX-Key.

Installing SXSim

SXSim is a Windows application written in Visual Basic 6.0. It has been tested with Windows 98, ME, XP, and 2000.

After you have downloaded the SXSim.ZIP archive, extract SXSim.EXE which is contained in this archive into any folder you like. There is no special Setup program required, as SXSim only makes use of DLL and OCX files that should be available on every Windows machine. If you like, define a shortcut to SXSim on the Desktop, or in the task bar.

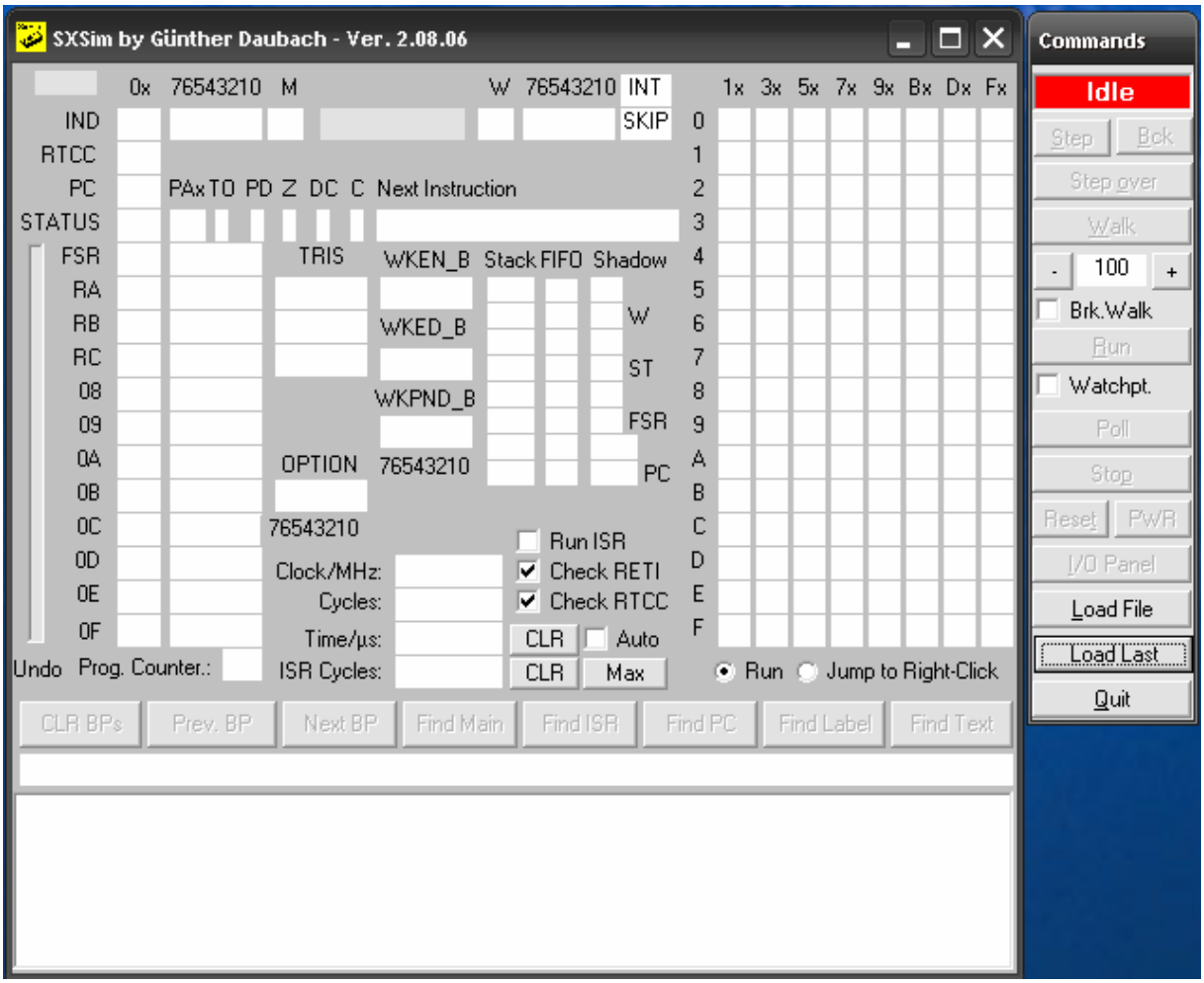
Note: When you are using the new SX-Key IDE with integration of SXSim, it is important to install new versions of SXSim in the folder “Tools\SXSim” which can be found in the main folder of SX-Key.

The SXSim User Interface

The user interface of SXSim has been designed to come very close to the popular SX-Key IDE, distributed by Parallax Inc.

Most controls of the SXSim user-interface come with integrated tool tip help messages, so just move the mouse cursor on top of an item for more information.

Here is how the SXSim user-interface looks like after you have launched SXSim:



If you are familiar with the SX-Key IDE, you will find many similarities between SXSim and SX-Key, but there are also some differences, and a couple of new items in the SXSim IDE.

First, you might notice that SXSim does not have a menu bar, like SX-Key. Instead, all SXSim operations can be executed by clicking the buttons, or other items that are shown in the “Main” and the “Commands” windows.

The “Commands” window, initially positioned to the right of the main window, acts as SXSim’s “Control Center”, similar to the one in SX-Key, but it has some additional command buttons. For now, there are only two or three buttons active in this area: “Load File”, “Load Last” (in case you have opened a file with SXSim before) and “Quit”.

Before performing any meaningful operation with SXSim, it is necessary to load a list file created from SASM while assembling the program, you want to simulate. In other words, any

program to be simulated must first pass SASM, i.e. it must be assembled without any errors (and warnings – if possible).

Click the “Load” button to open a file select dialog, where you can navigate through your computer’s folders, and select the LST (list file) of the program that you like to simulate.

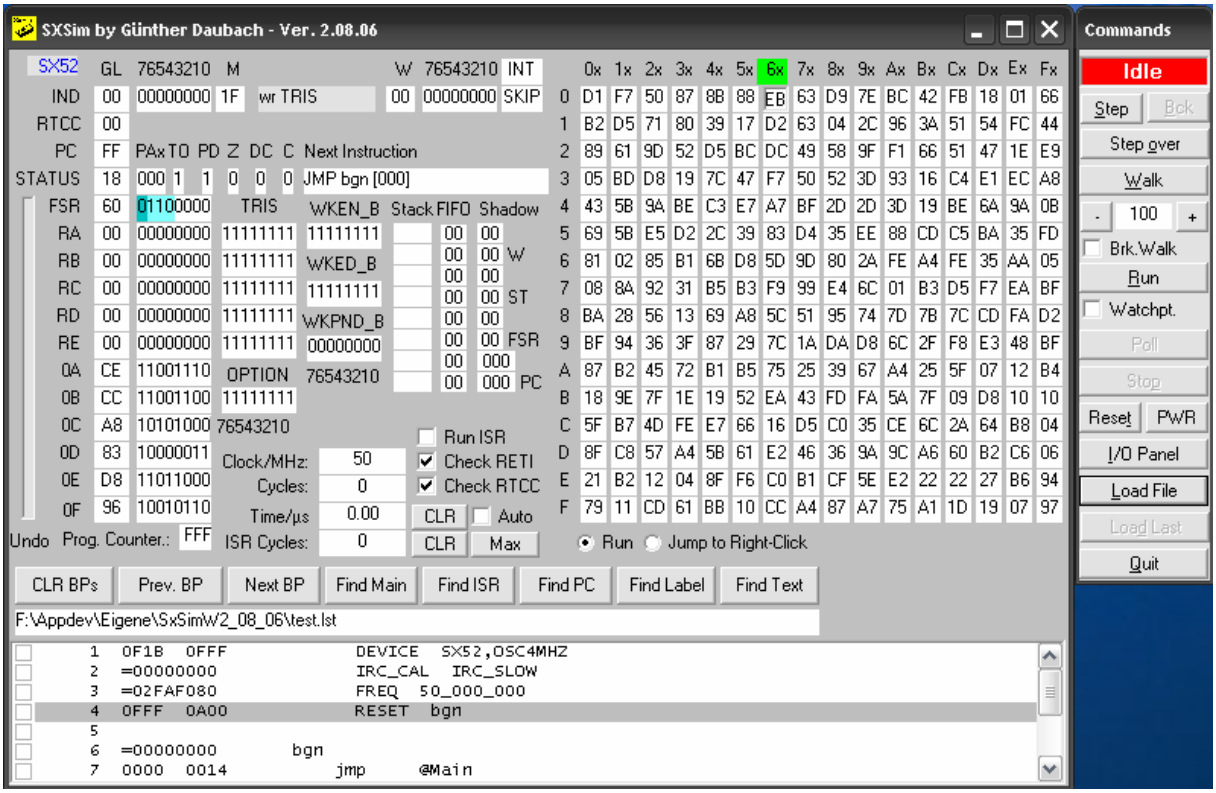
You may also drag and drop an icon assigned to a list file on the SXSim program icon in order to start SXSim and have it automatically load this list file.

SXSim also accepts a list file name passed via the command line. The command line must contain the full path specification, the file name and the .LST extension. For example:

```
SXSim c:\appdev\sx\project1\test.lst
```

When you activate SXSim from the SX-Key IDE, the list file generated from the source code currently shown in the IDE Editor will be automatically loaded. If necessary, the IDE first runs the assembler in order to generate the list file.

After you have selected a list file, the SXSim display will look similar to the next picure:



As in SX-Key, the upper half of the main window shows the “SX internals”, where the lower area displays a view into the application’s list file that is currently open with its path/name in the box on top of the list display. In SXSim, this file is not shown in a separate window that can be moved, re-sized, or closed, but in the lower fixed area of the Main window instead. Nevertheless, you may re-size the Main window, and the list section’s size will be automatically adjusted, Please note that re-sizing the Main window is limited to a minimum, so that the file register array at the top right, and at least four lines in the list section always remain visible.

The Commands window has been defined to be always on top of other windows, including the Main window.

In order to avoid that the Commands window overlaps the main window, it is “pushed” to the right when the width of the main window is increased.

In the upper half of the Main window, the register contents of Bank 0 in hexadecimal, and binary representation are shown, like SX-Key does. Note that the upper three bits of FSR’s binary representation are highlighted because of their special meaning as bank-select bits.

When SX48 or SX52 devices are simulated, bits 6, 5, and 4 are highlighted instead because these are affected by the BANK instruction. Bit 7 is then highlighted differently to indicate that this bit must be set by a separate instruction if necessary.

Similar to SX-Key, the contents of the M (MODE) and W registers are shown at the top center of the window, together with the fields “INT” and “SKIP” that will become highlighted while an interrupt is handled, or a skip will be executed next.

Note the blue text in the upper left corner of the main window, indicating the SX device that is currently being simulated.

The center area, where SX-Key displays the current instruction codes does not exist in SXSim. Instead, the “Next Instruction” field has been introduced showing the mnemonic of the next instruction to be executed, together with any symbolic name of an operand or a label (when defined). The remaining space in this area is used to display other important information, like the setting of the port direction registers (TRIS) in binary representation, the setting of the special port B configuration registers (WKEN_B, WKED_B, and WKPND_B), the “Next Instruction”, and the CALL-Stack display. New since SXSim version 1.50 are the fields in this area showing the status of the “internal SX secret registers”, like the 8-level FIFO buffer, and the shadow stacks for W, STATUS, FSR, and PC. See the chapter “The SX Secrets” later in this document for more details.

The buttons and displays in the lower area of the center part of the window deal with the SX system clock.

We will address these sections in more details later in this text.

The area to the right of the main window contains the display array of the SX memory banks above bank 0, similar to the SX-Key IDE.

According to the SX-Key IDE, the titles of the memory bank columns are “1x”, “3x”, etc. If you prefer titles like “0x”, “1x”, etc. for “straight” addresses, left-click on any of these titles in order to toggle the display mode. The current setting will be saved, and restored on a later session.

After you have loaded a list file, part of the file contents are displayed in the lower window area, the so-called “list file area”.

Manually Changing Register Contents

As in SX-Key, you may manually change register contents. Left-click on the hexadecimal representation of a register to activate the input mode. You will notice that the background color of this area will change to blue.

Type in the new value, and hit Enter to accept the new value, or Esc to leave without changing the current value. You may also hit any of the arrow keys to accept the new value you have entered. The up and down arrow keys move the blue edit marker to the previous or next register address, and the left and right arrow keys move the blue edit marker to the previous or next bank of registers. This feature is new to SXSim Version 2.8, and it is handy when you need to change subsequent register contents.

Alternatively, when a register also has a binary representation, you may left-click on any bit to toggle its state.

Setting Breakpoints/Watchpoints

When you simulate an SX program at maximum speed with SXSim using the “Run” mode, it often is helpful to stop program execution at a certain instruction in order to inspect the contents of registers at that line, or to continue program execution in single steps in order to analyze the further program flow from that point on. Therefore, SXSim allows you to set breakpoints on any line in the list file that contains executable code.

Sometimes, it makes more sense, not to halt program execution on a certain instruction, but to update the display of register contents instead.

Due to internal limitations of a “real” SX, the SX-Key IDE only allows you to specify one breakpoint at a time, and it does not allow to specify watchpoints at all. On the other hand, SXSim allows you to specify as many breakpoints/watchpoints as you like.

The SXSim list file area looks similar to SX-Key’s list window with the exception that each line has a small check-box to the left. Left-click on any check-box of a line containing executable code in order to set a breakpoint/watchpoint there (when you click a check-box in a line that does not contain executable code, nothing will happen at all). Check-boxes of lines that are “breakpointed/watchpointed” have a small check marker. In order to remove a breakpoint/watchpoint, simply click on a checked box again.

When the list file contains a BREAK directive, SXSim automatically sets a breakpoint/watchpoint at the next executable line following the BREAK.

Usually, when SXSim executes a program in “Run” mode, program execution will stop on each “breakpointed” line. As an option, marked lines can also act as “watchpoints” (see the description of the Commands window for more details). With this option selected, program execution does not stop on marked lines but the status display is refreshed instead.

Important Note: Other than the SX-Key IDE, SXSim will halt program execution on a “breakpointed” line *before* the instruction in this line is executed where the SX-Key IDE (due to the internal SX structure) halts program execution only *after* the instruction has been executed.

“One-Time” Breakpoints – “Run to Right-Click” Mode

This handy feature is new to Version 2.08.06 of SXSim. In order to use it, make sure that the “Run” radio button above the “FindPC” button is selected (the operation of “Jump to Right-Click” will be explained in the next section).

Sometimes, you might want to execute an SX program to a certain code line, from there to another line, etc. at highest speed. You could set a breakpoint on the first target line, click “Run” to simulate the code to this breakpoint, remove the breakpoint, set another one to the next line of interest, etc. This is a bit cumbersome, and when the program “hits” another breakpoint which is eventually set, it will halt there, so you need to click “Run” again until the target line is finally reached.

Right-clicking on any line in the List window that contains an executable instruction sets a temporary breakpoint on this line, enters “Run” mode, and the program is simulated at full speed until this temporary breakpoint is reached, while all other breakpoints are ignored in this mode.

“Jump to Right-Click” Mode

When the “Jump to Right-Click” radio button (above the “Find Label” button) is selected, right-clicking on a line in the List window that contains an executable instruction does not set a temporary breakpoint, and let SXSim simulate the program at full speed up to the marked line. Instead, the program counter (PC) is simply set to the address found in that program line. When you do a “Step”, “Walk”, or “Run”, program simulation will continue with the instruction in the line, you have right-clicked on. This is similar to double-clicking on a list line in the SX-Key debugger.

Please note that this mode can cause some trouble: As the contents of the program counter (PC) is modified, all instructions between the one previously addressed by the PC, and the newly addressed instruction will be skipped. So, no register contents (including the flags in

the status register, and the RTCC) will be updated, as it would be the case when the skipped instructions were executed, as in the “Run to Right-Click” mode.

You should keep this in mind when using the feature. As SXSim does not “know” how many instruction cycles the skipped instructions would take, it can no longer correctly update the “Cycles”, and “Time” fields. Therefore, they are reset to zero as soon as you right-click on an instruction line in this mode.

Other “strange” situations may occur when you use this feature. For example, when you right-click an instruction line in a subroutine, and continue the simulation from there, you will get a “RET without prior CALL” message as soon as SXSim executes the RET instruction because the associated CALL instruction was skipped. A similar situation will occur when you right-click on an instruction inside the ISR code when the RETI or RETIW instruction is reached.

Automated Saving of SXSim Sessions

Whenever you load another list file, or quit SXSim, the active breakpoints, the position and status of the windows and various other information of the current session will be saved to a file with the list file’s name, and an “.SIM” extension in the folder from where the list file was loaded. When you re-load this list file later, all breakpoints, the window positions and other settings will be restored again from the associated SIM file.

The Navigation Buttons

Above the list file section, there are some buttons that are unique to SXSim. These buttons can be used to quickly navigate through the list file, and to perform other helpful tasks.

CLR BPs

Click this button to clear all breakpoints that have been set.

Prev BP

Click this button to position the current list cursor on the previous line that contains a breakpoint (if any).

Next BP

Click this button to position the current list cursor on the next line that contains a breakpoint (if any).

Find Main

Positions the current list cursor to the line that contains the main program entry point, i.e. the address that is specified with the RESET directive in the source code. Please note that SXSim gathers this information from the RESET directive, i.e. it is not necessary to define a specific label, like “Main”, or “ResetEntry” for this address, but there must be a RESET directive in the list file, or an error message will be displayed when the list file is loaded.

Find ISR

Positions the current list cursor to address 0 which is the entry point of the SX interrupt service routine (if one has been defined).

Find PC

Positions the current list cursor to the location in program memory, currently addressed by the PC register.

Find Label

Click this button to open a dialog box where you can enter the name of a symbolic label. You may enter names of global or local labels (with a leading colon). When the specified label exists, the current list cursor will be positioned on the line containing that label. This function is not case-sensitive, i.e. it does not matter if you enter label names in upper-, lower-, or mixed case.

When you activate the Find Label function again, previously searched labels are stored internally, i.e. you may select any of them from the drop-down list.

Find Text

This function is similar to “Find Label”, but it allows you to search for any text in the list file (except in comments which are ignored by default). Again, the search is not case-sensitive, and previous search patterns are internally stored so that you can select them again from the drop-down list. Please note that this function always starts searching the text pattern from the current list cursor position towards the end of the list file, and then continues the search from the first line in the list file again.

Press F3 to continue the search for the most recently entered search pattern without the need to click the “Find Text” button once again.

By default, the “Find Text” function does not search text in comments. Mark the “Search comments” checkbox in the Find Text dialog to have SXSim search through all text, including comments.

The Commands Window

This window contains the items that are used to control SXSim. Some of them are “well-known” to SX-Key users, and some are unique to SXSim. You may drag this window to any suitable position of the desktop. It’s position will be saved in the SIM file associated to the current list file, i.e. when you re-open a list file, the window will be automatically placed at its former location. The Commands window stays always on top of other windows, so that it can’t be hidden when you maximize another window, like SXSim’s main window.

The Status Display

At the top of the Commands window, the current status of SXSim is shown:

- Idle – Waiting for a command, no simulation running.
- walking – The program is simulated in Walk mode.
- running – The program is simulated at maximum speed in Run mode.

Step

Click this button to execute the instruction that is currently highlighted by the list cursor in the list file section of the SXSim Main window. Any changes to the SX registers will be displayed, and recently changed registers are highlighted with a red background.

Bck

While single-stepping, stepping-over, walking, or running a program, it may happen that you go “too fast”, e. g. you click the Step button once more before noticing some significant register has changed, etc, or you click the Stop button too late while in the Walk or Run modes.

SXSim keeps track of the last 1000 instructions executed, and allows you to “roll them back”, i.e. to “undo” them. Click the “Bck” button to revert to the previous step(s).

Note that this button is inactive when no instructions have been recorded so far, or when all recorded instructions have been “un-done”.

No matter, in what mode the instructions are executed (in Step, Step Over, Walk, or Run Mode), they will all be recorded in the undo buffer. As this is a circular buffer, recording does not stop when the buffer is full. Instead, the oldest recordings will be lost.

The vertical bar at the left of the Main Window acts as a “gauge” indicating the “fill level” of this undo buffer.

The Undo feature can also become handy when SXSim reports an error or a warning. In such cases, “stepping back” might help you finding out the reason for that specific error or warning.

Step Over

This button acts similar to the Step button with the exception that subroutines will be “stepped over”. When the next instruction to be executed is a CALL, and you click “Step Over”, program execution runs at full speed until a matching RET, RETP, or RETW instruction is executed. When you have stepped “into” a subroutine, using the Step button, and click Step Over “inside” the subroutine, the remaining commands, including the final RET, RETP, or RETW instructions will be executed at full speed.

Step Over is a bit “tricky” with nested subroutines. SXSim keeps track of the nesting depth of subroutines. Whenever you click Step Over on a CALL instruction, or inside a subroutine, instructions are executed at full speed until a return instruction ends up at the same nesting level when Step Over was clicked.

Here is an example:

```
Sub1
        nop                     ;1
        call    Sub2            ;2
        nop                     ;3
        ret                     ;4

Sub2
        nop                     ;5
        nop                     ;6
        ret                     ;7

Main
        nop                     ;8
        call Sub1               ;9
        clr w                   ;10
        jmp Main                ;11
```

When you click Skip Over in line 9, Sub1, and the nested calls to Sub2 and Sub3 will be executed at full speed, and the execution finally halts at line 10. When you single-step into Sub1 to the nop in line 1, and click Skip Over there, the remaining instructions in Sub1 and the nested calls to Sub2 and Sub3 are executed at full speed, and the execution again halts at line 10.

When you single-step into Sub2 to the nop in line 5, and click Step Over there, the two nop’s and the ret will be executed at full speed, and execution halts on the nop in line 3 in Sub1.

Walk

Click this button to repeatedly execute instructions in “slow motion”, beginning with the currently highlighted instruction in the list file section. As in the “Step” mode, any changes of SX registers will be displayed, and recently changed registers are highlighted with a red background.

Click the “Stop” button to cancel the “Walk” mode. Clicking the “Reset” button instead, will also cancel the “Walk” mode, but it additionally performs a simulated SX reset.

The “Walk” speed can be adjusted by the controls that are described next.

-

Left-click this button to decrease the delay between execution steps while the “Walk” mode is active. (The current delay value in ms will be displayed in the field between the “-“ and “+” buttons. Left-click on that field to restore the default value of 100 ms).

The “Walk” speed is saved in the SIM file, and will be restored when you re-load the list file later.

+

Left-click this button to increase the delay between execution steps while the “Walk” mode is active. (The current delay value in ms will be displayed in the field left of this button – left-click into that field to restore the default value of 100 ms).

The “Walk” speed is saved in the SIM file, and will be restored when you re-load the list file later.

Brk. Walk

When this box is checked, the Walk mode will terminate on any active breakpoints.

The status of this checkbox is saved in the SIM file, and will be restored when you re-load the list file later.

Run

This starts SXSim’s run mode, which is a “hyper Walk mode”, i.e. the instructions will be executed as fast as possible without refreshing the register display after each instruction but only once, every second.

Click the “Stop” button to cancel the “Run” mode. Clicking the “Reset” button instead, will also cancel the “Run” mode, but in addition, it performs a simulated SX reset.

Watchpt.

Usually (when this box is un-checked), a breakpoint set on an instruction line will halt program execution in “Run” mode, allowing you to continue further program execution either in single steps, in “Walk” mode, or in “Run” mode again.

With the “Watchpt.” box checked, the “Run” mode operates differently:

- The register display will no longer be refreshed every second (click the “Poll” button instead at any time to force a refresh if necessary).
- Program execution is not halted on lines marked with a breakpoint. Instead, the register display will be refreshed when such lines are executed. This means, that marked lines are now used to trigger watchpoints, i.e. to automatically do a “Poll” instead of breakpoints.

This feature may be handy, when you want to know the contents of a variable after a certain instruction has been executed. As with breakpoints, you also my set more than one watchpoint at any time.

Poll

While the “Run” mode is active, the Main Window’s display will automatically be refreshed every second. Click the “Poll” button at any time to refresh the display in between. When watchpoints are active, the Main Window’s display will no longer be refreshed every second but only when an instruction with a watchpoint is executed. In order to refresh the display in between, you my click the “Poll” button at any time.

Stop

Click this button to terminate the “Walk” and “Run” modes at any time.

Reset

Click this button to reset the simulated SX. This will also terminate active “Walk” and “Run” modes. After a reset, various SX registers will be initialized to their defaults, where the file registers will remain unchanged, similar to a “real” SX when the /MCLR pin is pulled low, and released to high again.

Reset also clears the undo buffer.

PWR

This button performs the same actions as the Reset button. In addition, the file registers will be initialized with random values, similar to a “real” SX when it is powered up.

PWR also clears the undo buffer.

I/O Panel

Click this button to display the I/O Panel window. We will address the I/O panel in more detail in another chapter, below.

Load File

Click this button to open and load a list file after launching SXSim, or when you want to load another list file while SXSim is already active. Whenever you open another list file, or when you terminate SXSim, various settings of SXSim (like the active breakpoints) are saved in the directory where the list file exists, using the list file name together with the “.SIM” extension.

When you load a new list file, SXSim automatically checks if a matching “.SIM” file exists, and restores the most recent SXSim settings in that case. SXSim also keeps track if the list file has been changed since the last version of the “.SIM” file has been saved. This may be the case when you have re-assembled the related SX program in the meantime. As you may have added, removed, or re-located instructions, the breakpoints that were eventually saved at the end of the last SXSim session might no longer be correct. Therefore, when breakpoints were defined, SXSim displays a dialog box in this case, allowing you to either restore the breakpoints, or to ignore them.

While SXSim is running, it periodically checks if the currently loaded list file has been modified by another application in the meantime (e.g. by SX-Key). If this is the case, a dialog will be shown, giving you the choice to update the list file, or to go ahead with the currently loaded version.

Load Last

SXSim saves the name of the most recently opened list file in the Windows registry. To reload it, click the “Load Last” button. This is handy when you work on one project for a while. So there is no need to select this file from the Open File dialog each time you want to simulate the project again. Leave the mouse cursor for a while over this button. The tooltip text will then show you the file that will be loaded when you click the button.

Quit

The function of this button is obvious – it terminates SXSim after saving the current simulation settings, as described before. Instead of clicking the “Quit” button, you may also click the close-box at the top-left of the SXSim main window.

The Run ISR Checkbox

When single-stepping a program with an ISR controlled by RTCC roll-overs, it is often annoying to also have to single-step through the ISR instructions, or to run the ISR instructions with a breakpoint set at the RETI, or RETIW instruction to continue single-stepping from there. When the “Run ISR” checkbox is activated, ISR code will be executed at full speed automatically when you “single-step”, or “walk” a program.

This is a small demonstration program to show you how this works:

```
DEVICE  SX28, TURBO, OSCHS2, OPTIONX
IRC_CAL IRC_FAST
FREQ    50_000_000
RESET   Main

        ORG 0
ISR
        inc     8
        inc     8
        call    FromISR
        mov     w, #-50
        retiw

FromISR
        dec     8
        ret

Main
        clr     8
        clr     9
        mov     w, #%10001000
        mov     !option, w
Loop
        inc 9
        jmp     Loop
end
```

Make sure that the “Run ISR” box is un-checked, and then single-step through this program, and note how the ISR is entered when the RTCC overflows. After you have stepped through the ISR code (including the subroutine called from there), check the “Run ISR” box and continue single-stepping while watching the contents of the RTCC. When it overflows $FF, you will notice that stepping does not enter the ISR code but continues executing the main program instructions. Nevertheless, the ISR code has been executed. You can prove this by checking the contents of register $08 (it was incremented), and by the total cycles executed (they were increased by the number of cycles required for the ISR code).

The Check RETI Checkbox

By default, when SXSim is about to execute RETI, or RETIW instructions, it checks if an interrupt is currently active, and reports an error when this is not the case, as it is most likely that program execution will continue in “nowhere land” in this case.

The now disclosed “secret instructions”, and the shadow stacks make it possible to use the RETI and RETIW instructions for other purposes than returning from an interrupt. When you simulate programs making use of such “features”, uncheck this box.

The Check RTCC Checkbox

For precisely timed applications using ISRs triggered by RTCC overflows, it is important that an RTCC overflow never occurs while the ISR code is still being executed because in this case, the next RTCC overflow interrupt would be lost.

By default, SXSim issues a warning message when RTCC overflows while the ISR code is still being executed. SXSim will also issue a warning message when the final RETIW instruction in an ISR causes an RTCC overflow, i.e. when the current contents of RTCC plus the contents of the W register yield in a value above 255.

When timing is not a major concern, you may turn off these warnings by un-checking this box.

The Clock Section

This section contains some information about the SX clock. When SXSim finds a FREQ directive in the list file, the clock frequency specified together with this directive is displayed in the “Clock/MHz” field, else the text “unknown” is shown.

The “Cycles” field below the “Clock/MHz” field displays the total number of clock cycles that have elapsed for program execution so far. The next field, below displays the execution time that has elapsed so far. The contents of these fields will be cleared when you click the “Reset” button, or load another list file. You may also clear them in between by clicking the “CLR” button to the right of the Time/µs field.

When the “Auto” checkbox to the right of the “CLR” button is marked, the two field are automatically cleared when you click the “Run” button in the Commands window. This is handy when you want to know the number of cycles required to execute a program between two breakpoints, or between running from a breakpoint until the program is stopped at the same breakpoint the next time.

The “ISR Cycles” field displays useful information about the ISR timing. By default, the maximum number of clock cycles having been executed in the ISR so far is shown. Click the button currently named “Max” to display the minimum number of clock cycles the ISR has executed so far – the button’s caption will change to “Min” then. When you click this button again, it’s caption changes to “Avg”, and the “ISR Cycles” field displays the average number of clock cycles the ISR has executed so far. This value is calculated by dividing the total number of ISR execution cycles by the number of ISR calls. It should give a good estimate how many instruction cycles “remain” for the main program in average. Click the button again to cycle back to the “Max” display mode.

Click the “CLR” button to the right of this field to clear the displayed value.

Note that active “Hyper Counters”, described later, don’t allow for further timing measurements. In this case, these fields will display “???” when a “Hyper Counter” has been changed its contents for the first time.

The Stack Section

The eight fields under the “Stack” caption are used to display the current contents of the call stack. When a CALL instruction is executed, you will notice that the return address is displayed in the field at the bottom. In case the called subroutine calls another one, you will notice that the already stored return address is advanced on position up, and that now the second return address is displayed in the bottom field. The eight-level stack of the SX allows for a nesting depth of eight. In a “real” SX, the first return address gets lost when this value is exceeded. SXSim keeps track of the nesting level, and reports an error in case the nesting level is exceeded.

When a RET, RETP, or RETW instruction is executed, you will notice how the lowest address is “popped” off the stack, and that higher return addresses (if any) drop down one position.

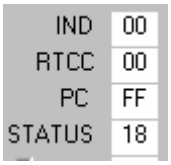
Executing a RET, RETP, or RETW instruction without a prior CALL on a “real” SX leads to unpredictable results leading the program flow into “Nirvana”. SXSim checks for such situations, and reports a Stack Underflow error when this option is selected.

As return addresses are pushed on the stack, you will notice that all fields that once held a return address will be highlighted. When the addresses are “popped” off the stack again, these highlights will remain active, acting as a “gauge” indicating the maximum “stack fill level”. This is handy to check the maximum subroutine nesting in a program. The “gauge” is cleared when you click the “Reset” button, or load another list file.

Note: When the list file does not contain a STACKX, or OPTIONX directive, the stack is limited to two locations only, and only the lower two fields of the stack will be visible, and stack nesting depth is limited to two.

The RTCC display

Usually, the special registers section in the main window looks like this:

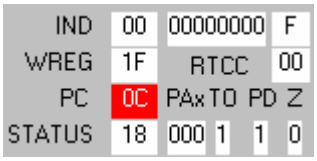


The current contents of the RTCC register is displayed in hex notation in the second field from top. The SX "maps" the RTCC register into RAM address 01 when bit 7 in the OPTION register is set.

When this bit is cleared, the W register (or WREG) is mapped into this address instead. Both, the SX-Key debugger, and SXSim display the contents of the W register in the second field in this case, and the field's caption reads "W" instead of "RTCC" in this case.

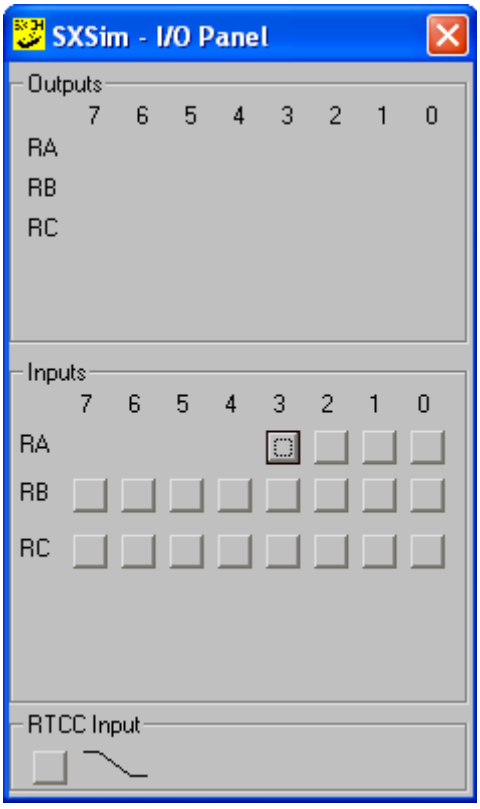
Thus, when the W register is mapped into address 01, you can no longer watch the contents of RTCC.

Therefore, SXSim, versions 2.08.06 and higher display an additional "RTCC" field to the right of the WREG field in order to show the contents of RTCC even though it is no longer mapped into address 01, i.e. a mov $01, $ff instruction will change the contents of W instead of RTCC. Nevertheless, even when W is mapped into address 01, RTCC will count up (when enabled), and trigger an interrupt on roll-ower, and the retiw instruction works as expected.



The “Smart” I/O Panel

Click the “I/O Panel” button to display the I/O Panel window:



The I/O panel contains three major sections, named “Outputs”, “Inputs”, and “RTCC Input”.

The “Outputs” section contains the “virtual LEDs” that are connected to the SX port pins configured as outputs. At start-up, there will be no LEDs active because by default, the SX sets all pins to inputs.

Later, when you simulate a program, “LEDs” will become visible for all port pins as they become configured as outputs, and the “LEDs” will be “turned on” when the output pin is set to high level by the simulated program later.

Like in SX-Key, you may click on a bit in the binary representation of the port register in order to toggle the bit. As an alternative, in SXSim, you may also click on an “LED” in the I/O Panel in order to toggle the associated port output bit.

The “Inputs” section contains pushbuttons/switches that are “connected” to the input port pins of the simulated SX. At start-up, or reset, there are pushbuttons available for all port pins because the SX sets all pins as inputs then. When any of the port pins is set to an output by the simulated program, the associated pushbutton will become invisible, and an “LED” will be shown in the “Outputs” section instead.

When an input button is visible, you may either left-click it in order to “push” the button (the assigned port pin will assume high level in this case as long as you keep the button “pushed”), or right-click the button to toggle its state. Whenever a button is “pushed”, i.e. left-clicked, or “toggled on”, it will be displayed in green.

The “RTCC Input” section contains one “virtual pushbutton” which represents the RTCC input. Note that this pushbutton can only be “pushed”, i.e. clicked with the left mouse button – it can’t be locked like the other input buttons. This button is visible as long as bit 5 in the

OPTION register is set, i.e. as long as the external RTCC clock is enabled (the default at start-up and reset). We will discuss this section a bit later.

Click the “Close” button at the top left to hide the I/O panel. In order to display the I/O panel again, click the “I/O Panel” button in the “Commands” window again.

When terminating SXSim, or when you select another LST file by clicking the “Load File” button, the current I/O Panel status (visibility, and position) is saved in the “.SIM” file for later restore

A Sample SXSim Session Featuring the I/O Panel

Enter the program, below, using the SX-Key IDE, and assemble it using SASM (not the “old” SX-Key Assembler). Next, launch SXSim, and click the “Load File” button.

In the file select dialog that opens next, navigate to the TestPortInt.lst file that was generated by SASM, and open it.

When the SX-Key IDE has a launch button for SXSim, simply click this button after entering the source code. The IDE will automatically assemble it, and then invoke SXSim (provided that there are no errors in the code).

Here is the source code that is used to generate TestPortInt.lst:

```
LIST          Q = 37
DEVICE        SX28L, TURBO, STACKX, OSCHS2
IRC_CAL       IRC_FAST
FREQ          50_000_000
RESET         Main

        org     $000
ISR
        mode    $09                     ; Select WKPND_B
        clr     w
        mov     !rb, w                  ; Exchange WKPND_B and w
        test    w                       ; Any WKPND_B bits set?
snz
          jmp   :RTCCInt                ; No, must be an RTCC interrupt
        xor     ra, w                   ; Toggle ra.1 or ra.0 depending on
                                        ; which WKPND_B it (1 or 0) is set
        reti
:RTCCInt
        xor     ra, #%00000100          ; Toggle RA.2
        mov     RTCC, #250
        reti
Main
        mode    $0b                     ; Select WKEN_B
        mov     !rb, #%11111100         ; Enable interrupts on RB.1 and RB.0
        mode    $0a                     ; Select WKED_B
        mov     !rb, #%11111101         ; Positive edge on RB.1, negative on
; RB.0
        mode    $0f                     ; Select TRIS
        mov     !ra, #%11111000         ; RA.2 ... RA.0 are outputs
        mov     RTCC, #250
        mov     !option, #%10101000     ; Enable RTCC interrupts, external
                                        ; RTCC clock, positive edges, no
                                        ; prescaler.
        jmp     $
```

The Main code enables interrupts/wakeups on positive edges at port B.1 and on negative edges at port B.0, and configures port A bits 2 through 0 as outputs.

After initializing the RTCC to 250, it enables RTCC interrupts, and sets the RTCC to be incremented by positive signal edges applied to the external RTCC input with no prescaler.

Finally, an endless loop is entered.

In the Interrupt Service Routine (ISR), the contents of the WKPND_B register and w are exchanged. As w is cleared before, WKPND_B is cleared afterwards (which is important to avoid more interrupts before another bit in WKPND_B is set again).

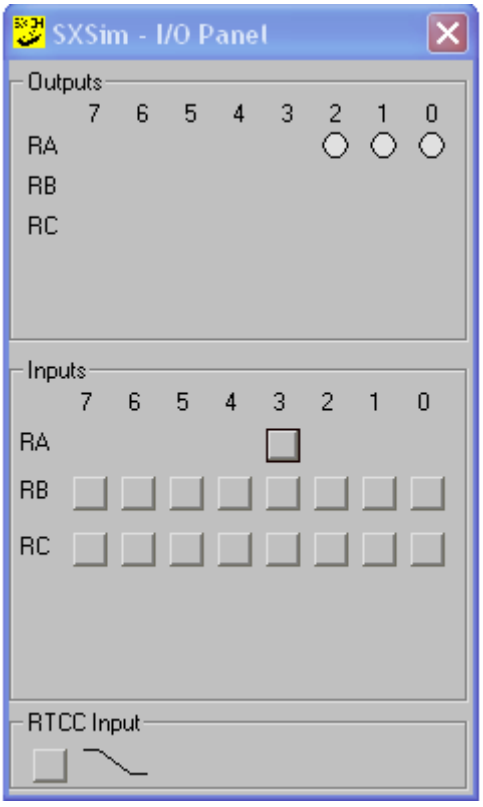
When no bits in WKPND_B are set, execution is continued at :RTCCInt, else the contents of WKPND_B is used to toggle the output pins RA1 and RA0. We will discuss the code following :RTCCInt later).

After TestPortInt.LST has been loaded, click the “I/O Panel” button to display the I/O Panel. It will display the default I/O port configuration, i.e. input buttons for all port pins and no output “LEDs”.

If possible, position the SXSim main window and the I/O panel on the Windows desktop so that both windows don’t overlap each other.

First, single-step through the instructions of the Main program section, and note how the three “LEDs” assigned to RA2, RA1, and RA0 become visible, and how the input buttons are hidden after the instructions for MOV !RA, #%11111000 have been executed.

Pins RA2...0 have been configured as outputs now, so you may no longer force these pins to high or low level with pushbuttons “connected” to them. The I/O panel should now look like this:



Other than real LEDs, the LEDs shown in the I/O panel are “clickable”. A left-click on any of the LEDs toggles the state of the associated output. Note how the port state displayed in the main window changes as you click an LED. This is similar to clicking a single bit in the binary representation of the ports in the main window.

As you step over the instructions to set the OPTION register, you will notice that the symbol right of the RTCC input button has changed from  to  . This is because bit 4 in the OPTION register is cleared now, which selects positive (rising) edges to trigger the RTCC, where the default is negative (falling) edges, i.e. OPTION.4 = 1.

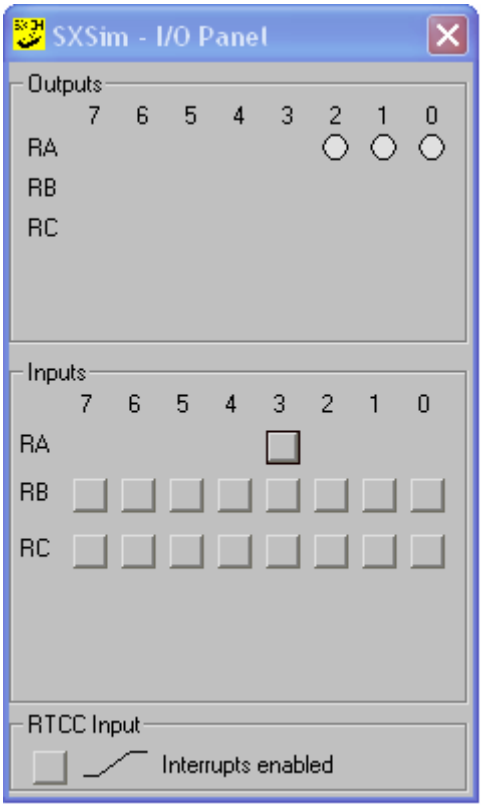
Next, “Walk” the program, and left-click and hold the RB.1 input button. In the SXSim main window, you will see how the program flow enters the ISR, and how the “LED” assigned to RA.1 is turned on. Note that the ISR code is entered as soon as you “push” the button because RB.1 is configured to trigger interrupts on positive edges. Release and push the button again for several times, and note how the RA.1 “LED” is toggled.

Next, click and hold the RB.0 input button for a while. You will notice that the ISR code will not be executed before you finally release this button again because RB.0 was configured to trigger interrupts on negative edges. Click and release this button a couple of times, and watch how the RA.0 “LED” is toggled.

The RTCC Input section

As bit 5 in the OPTION register is set, i.e. external RTCC clock signals are enabled. Therefore, the button in the RTCC Input section is visible (which is the default at start-up, or after a reset). As mentioned before, the “edge symbol” indicates which signal edge triggers the RTCC.

When bit 6 in the OPTION register is cleared, i.e. RTCC interrupts are enabled, you will notice that the text “Interrupts enabled” is displayed in the RTCC Input section.



After you have tested the port B interrupts as described above, continue “Walking” the program.

Please notice that the field “ISR Cycles” should display 13 after you have clicked one of the two buttons assigned to RB1 and RB0 at least once, indicating that the ISR execution has taken 13 clock cycles (this includes the cycles required to enter the ISR, and to execute the final RETI instruction.

Now click the button in the RTCC Input section several times. Note how the RTCC register increments each time you click that button. When RTCC overflows from FF to 00, the ISR

code will be executed, but this time, it branches to :RTCCInt, where it toggles bit RA.2, and the “LED” “connected” to that output confirms this.

Now, check the contents of the “ISR Cycles” field again. It should display 18, i.e. the code to handle the RTCC interrupt took longer than the code to handle the port B interrupts. The “ISR Cycles” field, by default, displays the maximum number of ISR cycles required for an ISR execution so far. Click the “Max” button once to display the minimum ISR cycles (the button’s title has changed to “Min”). Click the button once more to display the average number of ISR cycles required so far.

The “SX Secrets”

The designers of the SX silicon have implemented some instructions and internal storage dedicated to support “on-chip debugging” that were not published so far. In the meantime, Ubicom has allowed that this information may be disclosed. Therefore, SXSim now supports most of these “secret instructions”.

Please note that using any of these instructions may cause that debugging such a program on a “real” SX chip may be no longer possible, or may cause erroneous results. On the other hand, SXSim can easily simulate these features.

The FIFO instruction (Op-Code $047)

The SX has an internal 8-level FIFO (first-in-first-out) buffer, normally used to store information while a debug session is performed but this buffer can also be handy as temporary storage without using RAM locations for this purpose.

The FIFO is organized as an 8 byte circular buffer with the W register as the data “source” and “sink”.

Whenever a FIFO instruction is executed, the contents of W are pushed on that stack, and the contents of the eighth level are rolled into W.

Simulate the following program to see how it works:

```
LIST Q = 37
DEVICE  SX28L, TURBO, STACKX, OSCHS2
IRC_CAL IRC_FAST
FREQ    50_000_000
RESET   Main

FIFO macro
dw    $047
endm
org   $000
Main
mov   w, #1
Loop
fifo
jmp   Loop
```

While single-stepping through the code, note how the contents of W (01) is transferred into the lowest level of the FIFO buffer after the first execution of the FIFO macro. After this, W holds 00 (the value, rolled from the topmost (8^{th}) buffer level into W).

Continue single-stepping through the code, and note how the 01 “advances up” in the FIFO buffer until it “arrives” in W again after the 9^{th} execution of the FIFO macro.

The PUSH and POP Instructions

When the SX enters into the interrupt service routine, the contents of the registers W, STATUS, FSR and PC are automatically saved in so-called “shadow” registers, and when a

RETI or RETIW instruction is executed these registers are automatically restored from these "shadow" registers. This great feature makes it unnecessary to explicitly save and restore any of these register contents within the interrupt service routine.

But this is only the "half of the truth". Actually, the "shadow registers" for each of the W, STATUS, FSR, and PC registers are "mini stacks" with two levels. When an interrupt occurs, the contents of the W, STATUS, FSR, and PC registers are pushed into the first level of these "mini stacks", and they are popped-off again when a RETI, or RETIW instruction is executed.

Again, the 2nd level of the "mini stacks" was implemented by the SX designers for on-chip debugging purposes.

When debugging is not required, one level of the "mini stacks" may be used to store temporary values outside of the "regular" SX file registers. In applications without interrupts, even both levels may be used.

SXSim versions 1.50 and higher display the current contents of the "mini stacks" in the "Shadow" column. Note that the PC-stack is 12 bits wide, where the other stacks are 8 bits wide.

SXSim supports the following "secret instructions" that deal with the "mini stacks":

PUSH W ($048)	- pushes the contents of W on the W stack
PUSH STATUS ($049)	- pushes the contents of W on the STATUS stack
PUSH FSR ($04a)	- pushes the contents of W on the FSR stack
PUSH PC ($04b)	- pushes the contents of the lower 4 bits of M, and W on the PC stack
POP W ($04c)	- pops off the recently pushed value from the W stack into W
POP STATUS ($04d)	- pops off the recently pushed value from the STATUS stack into W
POP FSR ($04e)	- pops off the recently pushed value from the FSR stack into W
POP PC ($04f)	- pops off the recently pushed value from the PC stack into M (4 bits) and W

Using these PUSH and POP instructions, applications might push certain values on the "mini stacks", and later use a just one RETI instruction to pop off values into the W, STATUS, FSR, and PC registers.

In this case, RETI is not used to return from an interrupt service routine. By default, SXSim keeps track of active interrupts, and reports an error when a RETI, or RETIW instruction is going to be executed when no interrupt is active. To overrun this test, un-check the "Check RETI" box below the Stack section in the SXSim window.

The MOV W, !OPTION Instruction ($001)

This "secret instruction" is also supported by SXSim, Version 1.50 and higher.

Other Unsupported "Secret Instructions"

There are some more "secret instructions", like activating the debug mode ($044), shift the OSC pin in/out W ($045), and setting the breakpoint register ($046) that are not supported by SXSim.

SXSim “Hyper Counters”

Even on a fast Windows PC, simulated SX programs run a lot slower, compared to the execution times on a “real SX silicon”. For example, when your application contains timer counters that are decremented in nested loops within the ISR to achieve a delay of – say – one second, with SXSim, it may take hours to run this kind of code until the counters all read zero, even in “Run” mode.

On the other hand, once, you have checked that such counters decrement as expected, for example, it is not necessary to confirm that the next hundreds of decrements are also done correctly. Therefore, you may declare any of the global registers at address $08…$0F, and any of the registers in the register banks as “Hyper Counters”.

Right-click on the hex display section of any of these registers to toggle between the “Hyper Counter” and “regular” modes. When the “Hyper Counter” property is selected for a register, its background color is displayed in green.

The following instructions are handled differently when the fr-part of the instruction refers to a register with active “Hyper Counter” property:

dec fr

The contents of fr will immediately assume zero, and the Z flag is set.

decsz fr

The contents of fr will immediately assume zero, and the Z flag is set, i.e. the skip will be executed.

inc fr

The contents of fr will immediately assume zero, and the Z flag is set.

incsz fr

The contents of fr will immediately assume zero, and the Z flag is set, i.e. the skip will be executed.

movsz w, ++fr

The contents of w is set to 0, fr is set to 255, and the Z flag is set, i.e. the skip will be executed.

movsz w, --fr

The contents of w is set to 0, fr is set to 1, and the Z flag is set, i.e. the skip will be executed.

Please note that SXSim can no longer keep track of the number of cycles executed, or the elapsed execution time when “Hyper Counters” are active, and any of the above listed instructions are applied to a “Hyper Counter”, i.e. in this case, the displayed information about clock cycles or total execution time is no longer accurate. The same is true, when any of the instructions listed above is applied to a “Hyper Counter” register from within the ISR – in this case, the “ISR cycles” field is no longer accurate. Therefore, as soon as a “Hyper Counter” has been changed, the fields in the Main Window display “???” instead of clock cycles, or elapsed time values.

SXSim Error/Warning Messages

As SXSim “runs” a simulated SX controller, it can perform several plausibility checks while code is executed which a debugger running on “real SX silicon” can’t do.

Here is a list of the messages:

Errors:

When SXSim detects an error, it stops program simulation when the Walk or Run modes are active, and displays one of the following messages:

Bad call address, or wrong page!

A CALL instruction refers to an area in program memory where no code exists, possibly because of a missing PAGE instruction.

Bad jump address, or wrong page!

A JMP instruction refers to an area in program memory where no code exists, possibly because of a missing PAGE instruction.

Bad jump or return address!

A JMP or RET instruction points to an area of the program memory where no code exists. Check if a PAGE instruction exists before the JMP, or use RETP instead of RET.

CALL nesting too deep!

The maximum subroutine call nesting has been exceeded (8 in default, or 2 in “Compatibility” mode), i.e. the subroutine return stack has overflowed.

Could not open the LST file passed via the command line!

This error is reported when SXSim cannot find the list file in the directory passed together with the file name in the command line.

RESET directive not found!

SXSim searches the list file for a RESET directive in order to find out where program execution shall begin. If this directive is missing, SXSim can’t simulate the program.

RET without prior CALL!

A RET instruction outside of a subroutine was found.

RETI without active interrupt!

An RETI instruction outside of the interrupt service routine was found.

You may disable this check by un-marking the “Check RETI” checkbox in the SXSim main window when using the “secret SX instructions”.

RETIW without active interrupt!

An RETIW instruction outside of the interrupt service routine was found.

You may disable this check by un-marking the “Check RETI” checkbox in the SXSim main window when using the “secret SX instructions”.

RETP without prior CALL!

An RETP instruction outside of a subroutine was found.

RETW without prior CALL!

An RETW instruction outside of a subroutine was found.

RETIW has caused an RTCC overflow

The RETIW instruction adds the contents of W to the RTCC register. This warning is issued when the result of this addition is greater than 255. In applications that rely on precisely timed interrupts, such situations should be avoided. RTCC checking can be de-activated by un-checking the “Check RTCC” box in SXSim’s main window.

RTCC overflow within the ISR

While the last interrupt is still being serviced by the ISR, another RTCC overflow has occurred. This is the case when the total number of instruction cycles in the ISR is greater than the interrupt period defined with the last RETIW instruction. RTCC checking can be de-activated by un-checking the “Check RTCC” box in SXSim’s main window.

The Program has no entry point

SXSim requires that the entry point of the main program is specified in the list file, i.e. that the source code file contains a RESET directive in order to begin the simulation at one well-defined address.

You may only open LST files with SXSim

This error is reported when you try to drag and drop a file icon that is not assigned to a list file on the SXSim icon, or pass a file name with an extension other than “LST” on the command line.

Warnings:

Address 0 not found.

The “Find ISR” function did not find valid code at address 0 (the ISR entry-point).

Label not found.

The “Find Label” function could not find the requested label.

List file has been modified - restore Breakpoints?

This message shows up when you re-load a list file that has been modified by the SxKey IDE since the last SXSim session. When you have modified the original SRC file by adding or removing instructions, the breakpoints saved in the associated SIM file and the new resulting list file may no longer match.

Click “No” to not restore the breakpoints. When you click “Yes”, you should check the locations of the defined breakpoints using the “Prev. BP” and “Next BP” functions.

List file has been modified by another application - reload it?

SXSim periodically checks if the list file stored on the hard disk differs from the one currently loaded. When a difference has been detected, this message appears, and you have the choice to re-load the modified file, or to continue the simulation with the currently loaded version.

No chip type specified – assuming SX18. Continue simulation?

SXSim requires an information about the chip type to simulate in order to select the right memory sizes, I/O ports, etc. Similar to SASM, SXSim assumes an SX18 when the list file does not contain a DEVICE SX?? direction. Click “Yes” to continue. In case, you want to simulate a different chip type, click “No”, add the desired DEVICE specification to the source code, assemble it again, and then start the simuilation.

Text not found.

The “Find Text” function could not find any matching text in the currently loaded list file. When searching for text in comments, make sure that the “Search comments” box is checked in the “Find Text” dialog box.

The SIM Files

Whenever you open a list file with SXSim, and close it later, SXSim saves a file with the name of the list file, but an ".SIM" extension in the folder where the list file is located. This file is used to save information about the current SXSim session. When you open the same list file later, the most important settings of the previous session will be restored from the SIM file:

- Recent date/time stamp of the LST file in order to detect if the LST file has been modified between two SXSim sessions.
- The breakpoints.
- The status of the Brk.Walk checkbox.
- The position of the Commands window.
- The visible status of the I/O panel window.
- The position of the I/O panel window.
- The mode of the register column headers.
- The position and size of the main debug window.
- The status of the Check RETI and Check RTCC checkboxes.
- The selected Walk speed.
- The "Hyper Counter" properties of the file registers.

This makes it easy to continue a simulation session with a specific list file after having a break.

SX48/52 Support

New since version 2.01 of SXSim is the support of the "large" SX48/52 devices. SXSim parses the DEVICE setting when loading a list file, and automatically adjusts itself to the SX type specified. This means that the size of the simulated program memory, the size of the available RAM (File Registers) and the addressing modes for file registers is adapted to the "silicon under simulation".

Please note that this version does not support the two multi-purpose timers/counters that are available on the SX 48/52 chips. I think I should create a new window in a future version of SXSim to display the states of these timers/counters.

The End

This ends the SXSim documentation for now. Please stay tuned, more features will be coming soon (or whenever I have the time)…

If you have any comments, suggestions, wishes, or found a bug, please feel free to contact me at any time.

Günther Daubach

g.daubach@mda-burscheid.de

Version History

October 27, 2004 (Version 1.42 - RTCC bug fixed)

November 14, 2004 (Version 1.43 - RL/RR bug fixed)

December 07, 2004 (Version 1.44 - IREAD bug fixed)

December 08, 2004 (Version 1.45 - JMP PC+W bug fixed)

December 11, 2004 (Version 1.50 - supports “secret” SX instructions)

December 13, 2004 (Version 1.51 - allows to turn off RETI/RETIW checking)

December 21, 2004 (Version 1.52 - handling of the “shadow stacks” fixed)

December 22, 2004 (Version 1.53 - new single-step undo feature)

January 05, 2005 (Version 1.54 - several bug-fixes and new features)

January 12, 2005 (Version 1.55 - bug-fixes and new “Hyper Counter” feature - see the new doc file)

January 15, 2005 (Version 1.56 - bug-fixes, enhanced “Hyper Counters”, RTCC checks, enhanced undo, random register values on reset, error messages when SX 48/52 files are opened - see the new doc file)

January 19, 2005 (Version 1.57 – bug in undo function fixed in order to correctly restore the M register, and the highlight in the list window).

January 30, 2005 (Version 2.01 – several bug-fixes and enhancements, like an option to pass the list file name via the command line, and enhanced ISR cycle statistics – plus last, not least, SX48/52 support).

January 31, 2005 (Version 2.02) – a bug fixed in the SX48/52 simulation.

February 02, 2005 (Version 2.03) – several bugs in the SX58/52 simulation fixed. SX18 assumed when no device is specified in the list file. LST file icons may now be dragged and dropped on the SXSim icon in order to launch SXSim with that file.

February 05, 2005 (Version 2.03)

Bugs fixed:

- When a value was moved into a port register, the states of the input lines were overwritten.
- After reloading a list file after it was modified by another application the window positions and sizes were restored to their defaults.
- On systems with two video monitors, an overflow run-time error could occur when the Commands window, or the I/O panel was moved to an extremely right screen position.
- When the main window was minimized, a run-time error “A form can’t be moved or resized while it is minimized” occurred before.
- Clicking the “Find Main” button caused an error message “Bad jump or return address”.
- When modifying registers in the highest register bank of a simulated SX48/52, an overflow run-time error could occur.
- OPTION bit 6 was ignored, i.e. whenever there was a RTCC roll-over, an interrupt was executed.

New features:

- A new option “Run ISR” allows to execute ISR code at full speed while “single-stepping”, or “walking” a program.
- The “LEDs” in the I/O panel can now be clicked in order to toggle the state of the associated outputs.
- When a port bit state is changed by clicking on the bit in the main window, the “LED” in the I/O panel changes its state accordingly.
- When there is no STACKX, or OPTIONX device directive in the list file, OPTION bits 7 and 6 are now treated as read only.
- When the main window is widened, the Commands window is “pushed” to the right in case the main window would overlap it.
- OPTION bit 7 is respected now. This means when this bit is set, the RTCC is mapped into address $01, when the bit is clear, the W register is mapped into this location.

March 05, 2005 (Version 2.05)

- Fixed the add pc+w (jmp px+w) command which could result in bad jump addresses.
- Bad handling of "hyper counters" in upper memory banks fixed.
- Last device category ("small" or "large") is now saved in the SIM file. When the category gets changed by an external application since the last session, a message box signals that the session settings can't be restored.
- Separate display fields for “Cycles” and “Time/µs” instead of one field with a toggle button.
- New “Auto” checkbox. When checked, the cycle counter is automatically cleared when the Run button is clicked. This is handy to measure the cycles starting from a breakpoint until this breakpoint is reached again.
- Mode register will be initialized to 0x1F on reset for SX 48/52 devices now.
- Mode register is saved/restored on interrupts for SX48/52 devices now.
- Added an API call to keep the Commands window always on top.

June 10, 2005 (Version 2.06)

- Elapsed time field and title is now adjusted to µs, ms, or s.
- Added display for which port configuration register is currently addressed by the M register
- On reset, RTCC is initialized to 0 now, and FSR is initialized to a random value.
- Changed parsing of the cross reference section in a list file in order to handle long symbol names
- Changed parsing of lines containing device settings to only read settings from lines with a leading DEVICE directive
- Fixed wrong handling of the MOV PC, W instruction
- Adapted highlighting of the FSR bits affected by a BANK instruction depending on the current device type
- The current list file name/path is no longer displayed in the main window’s caption as long names were truncated. Instead, the name is now displayed in a box above the list display.
- Changed parsing of DEVICE SX?? directives to correctly determining the device type when the list file contains conditional assembly.

- The results of MOV Rx, w instructions are now internally buffered. This means that port output bits are now correctly set when a later MOV !Rx, w instruction follows that turns inputs into outputs.

Version 2.07

- Various bugs fixed, and several suggestions from users implemented.

Version 2.08.03

- New “Watchpoint” feature introduced. As an option, program execution halts on marked lines as before (breakpoints), or continues running the program, updating the variable display instead. When the Watchpoint mode is active, the display will no longer be automatically updated every second while the Run mode is active.
- Editing of register contents enhanced. Using any of the arrow keys confirms the change made to the current register, and marks the previous/next register, or bank for editing.
- New “PWR” button added. Clicking PWR resets SXSim, like “Reset” does. Similar to a power-on with a “real” SX, the contents of the file registers will contain random values. The “Reset” button now simulates the behavior of a “real” SX with a low-high transition on the /MCLR pin, i.e. a reset is performed but the register contents remain unchanged.

Version 2.08.04

- Bad handling of RTCC prescaler fixed.
- Wrong cycle count of skip instructions fixed.

Version 2.08.05

- Added display for internal program counter
- Allow for “odd” clock frequencies
- Handling of SKIP instruction fixed, i.e. both instruction codes $0602, and $0702 are handled as SKIP now.

Version 2.08.06

- Added RTCC display that comes up when WRED is mapped into address $01 to allow for watching the RTCC.
- In RUN mode, the list display no longer jumps back to the first line.
- When SXSim is terminated with the main window minimized, it will be opened with its default size when the associated LST file is opened again later.
- When TRIS bits are changed, the I/O Panel is updated correctly now in RUN mode.
- Breakpoints now work on targets of JMP w instructions.
- Fixed wrong clock cycle count of skip instructions, clocks are incremented by 2 in case of a skip, instead of 3.
- Fixed display of SKIP status. Former versions always highlighted the SKIP indicator, no matter if a skip would follow, or not.
- Added a new “Run-To” and “Jump-To” mode, i.e. right-clicking on a valid instruction line in the list window when the “Run-To” mode is selected, sets a temporary breakpoint on that line, and activates the “Run” mode. The program is executed until this temporary breakpoint has been reached – all other breakpoints are ignored in this mode. When the “Jump-To” mode is selected instead, the program counter (PC) will be set to the address found in the instruction line, and all instructions between the former PC, and the new location will be skipped.

- Added a status display in the Commands window, showing the current status: “Idle”, “running”, or “walking”.