SXSim – A Simulator for the SX Family of Microcontrollers for Microsoft Windows

Alpha 3 Release

Copyright ©, 2004
by Guenther Daubach
Im Eulenflug 25
D-51399 Burscheid, Germany
Tel./Fax: +49 2174 – 785 931

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 software. Please note that SXSim may not work correctly with LST files that were created by other versions, or types of SX assemblers because SXSim makes certain assumptions on the structure of the LST files, unique to the SX-Key SASM.

SXSim is – by no means – yet finished, or fully tested. It is in Alpha state, 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 only support the “small” SX devices, i.e. SX 18/20/28, but not the SX 48/52 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”. But there are always some matters that could be improved:

- 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 immediately.

- Due to the internal structure of the SX, you can only define one breakpoint at a time with SX-Key and any other SX development system. SXSim allows you to define as many breakpoints as you like, and it automatically saves the breakpoint positions, 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 that possible.

- Due to the internal structure of the SX, the “breakpointed” instruction is always executed before the program stops. This means that breakpoints on JMP or CALL instructions actually halt the program execution on the first instruction of the JMP target, or the first instruction of the 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, or the elapsed execution time at the specified clock frequency. It also analyzes the clock cycles of an ISR code.

- SXSim performs checks for 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 deep. It also displays a "Stack gauge", which gives you an idea of how deep you have nested subroutine calls in a program.

- SXSim comes with a "Walk" mode, i.e. a "slow motion" mode where you may select the motion speed. You can also select if active breakpoints shall stop the "Walk" mode, or not.

- SXSim also comes with an I/O Panel that allows you to connect eight "Virtual LEDs" to any output pin of the simulated SX, and eight "Virtual Pushbuttons/Switches" that you may connect to the input pins of the simulated SX. This is a very rudimentary version of an I/O Panel, and it will be definitely enhanced in future versions of SXSim.

  The I/O Panel also has a special "signal" that is "connected" to the simulated SX RTCC input.

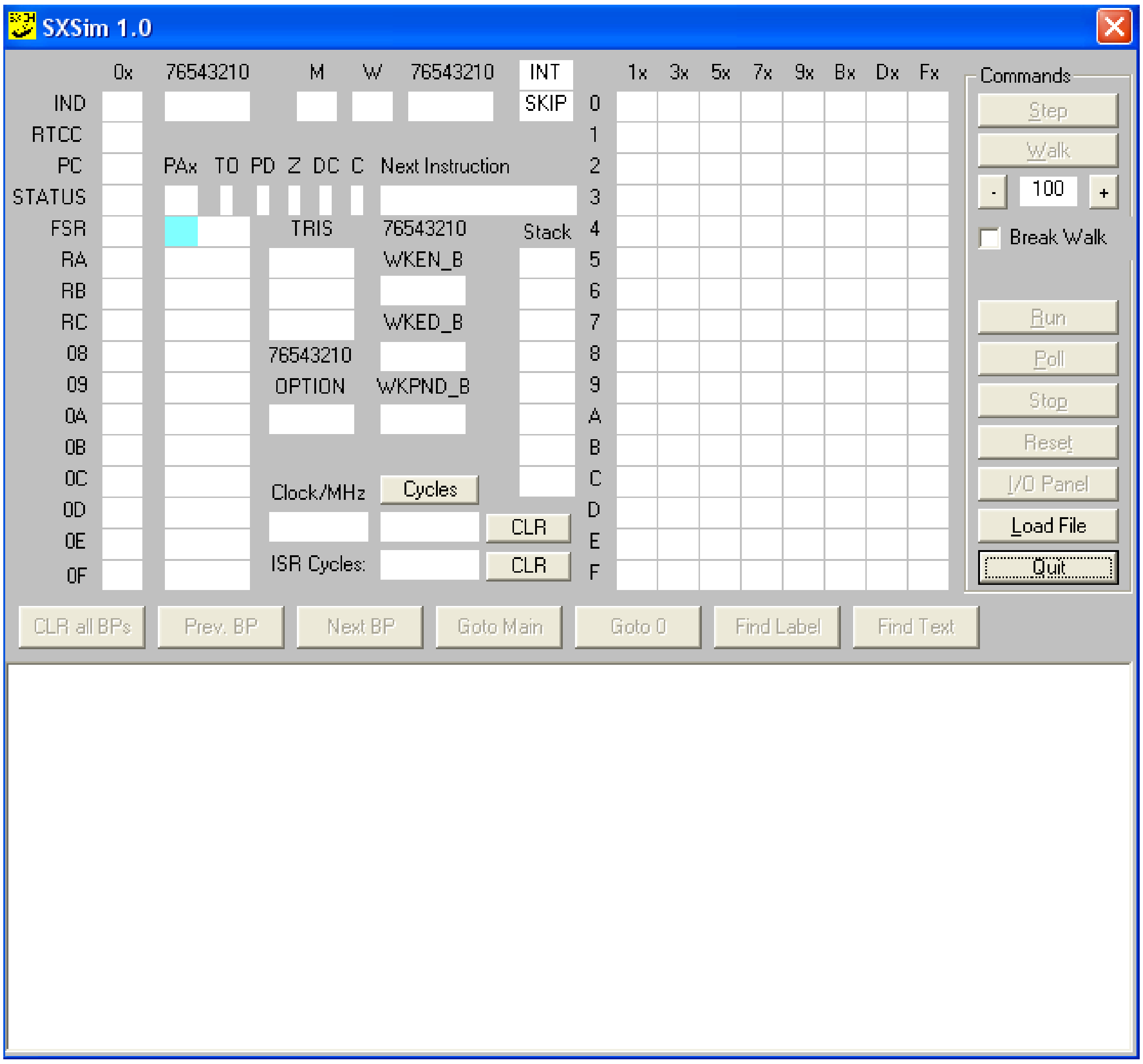
As you can see, SXSim comes with some features that are not available with "real-live" debuggers. Nevertheless, SXSim's target is not to replace such "real-live" debuggers, like the SX-Key – it has been designed as a utility for those users who like to "test before buy" a "real" SX development system,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

The SXSim User Interface

The user interface of SXSim has been designed very similar to the popular SX-Key IDE, Version 2.x, distributed by Parallax Inc.

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

Here is how the SXSim user-interface looks like when you launch SXSim:



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

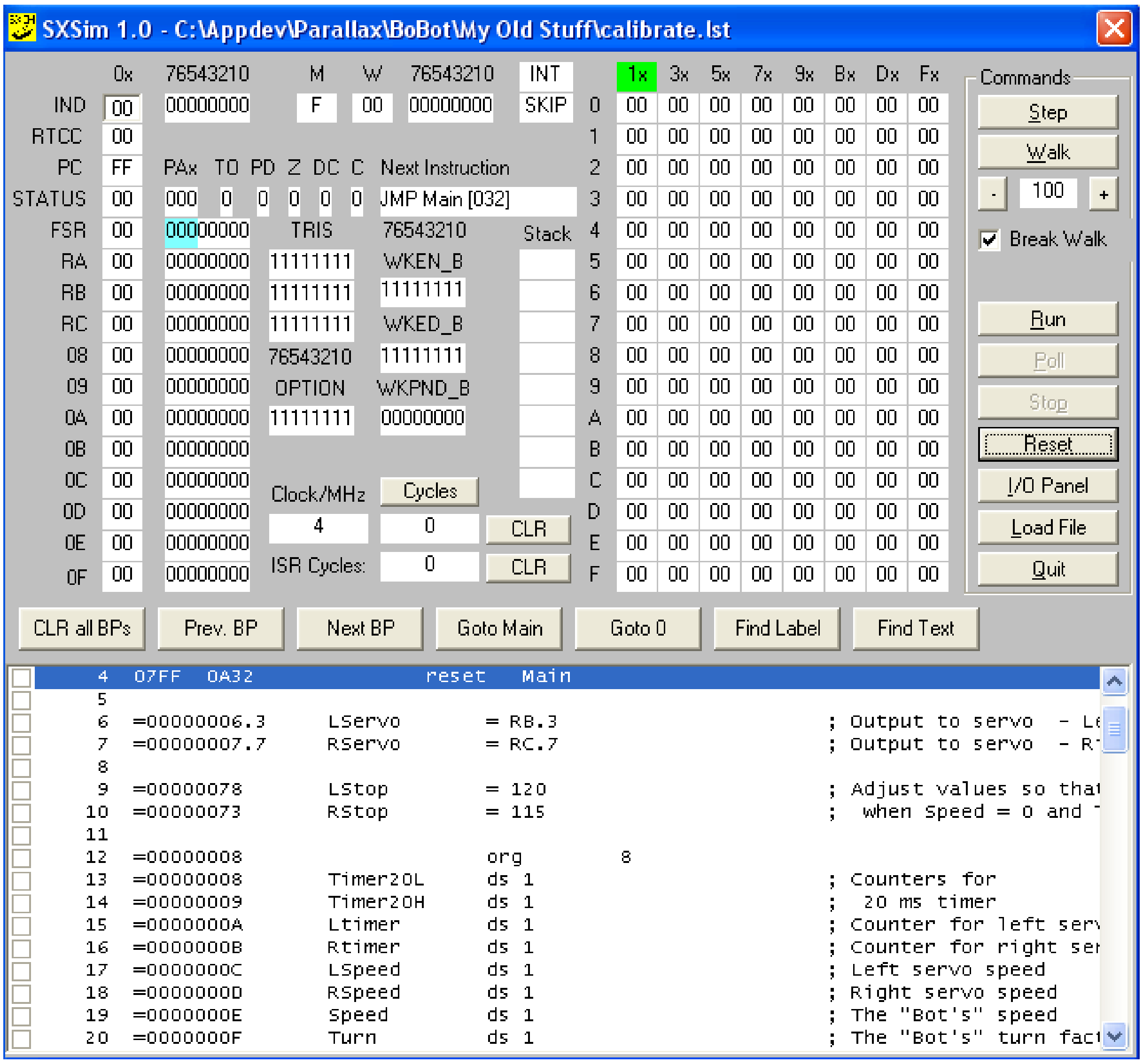
First, you may notice that SXSim does not have a menu bar, like SX-Key. Instead, all SXSim operations can be executed by clicking the buttons, or other items that are shown in the main window.

The “Commands” section to the right acts as SXSim’s “Control Center”, similar to the one in SX-Key. For now, there are only two buttons active in this area: “Load File”, and “Quit”.

Before performing any meaningful operation with SXSim, it is necessary that you load the list file that SASM has created when you have assembled the program, you want to simulate. In other words, any program that you want to simulate 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 you want to simulate.

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


SXSim 1.0 - C:\Appdev\Parallax\BoBot\My Old Stuff\calibrate.lst

```
         0x   76543210       M    W   76543210   INT     1x 3x 5x 7x 9x Bx Dx Fx
IND      00   00000000       F    00  00000000   SKIP  0 00 00 00 00 00 00 00 00
RTCC     00                                            1 00 00 00 00 00 00 00 00
PC       FF   PAx TO PD Z DC C  Next Instruction      2 00 00 00 00 00 00 00 00
STATUS   00   000  0  0 0  0 0  JMP Main [032]         3 00 00 00 00 00 00 00 00
FSR      00   00000000  TRIS     76543210     Stack    4 00 00 00 00 00 00 00 00
RA       00   00000000  11111111 WKEN_B                5 00 00 00 00 00 00 00 00
RB       00   00000000  11111111 11111111              6 00 00 00 00 00 00 00 00
RC       00   00000000  11111111 WKED_B                7 00 00 00 00 00 00 00 00
08       00   00000000  76543210 11111111              8 00 00 00 00 00 00 00 00
09       00   00000000  OPTION   WKPND_B               9 00 00 00 00 00 00 00 00
0A       00   00000000  11111111 00000000              A 00 00 00 00 00 00 00 00
0B       00   00000000                                 B 00 00 00 00 00 00 00 00
0C       00   00000000  Clock/MHz  Cycles              C 00 00 00 00 00 00 00 00
0D       00   00000000      4        0      CLR        D 00 00 00 00 00 00 00 00
0E       00   00000000                                 E 00 00 00 00 00 00 00 00
0F       00   00000000  ISR Cycles:  0      CLR        F 00 00 00 00 00 00 00 00
```

Commands: Step, Walk, - 100 +, ☑ Break Walk, Run, Poll, Stop, Reset, I/O Panel, Load File, Quit

CLR all BPs | Prev. BP | Next BP | Goto Main | Goto 0 | Find Label | Find Text

```
       07FF   0A32             reset    Main

       =00000006.3     LServo       = RB.3                 ; Output to servo  - Le
       =00000007.7     RServo       = RC.7                 ; Output to servo  - R

       =00000078       LStop        = 120                  ; Adjust values so that
       =00000073       RStop        = 115                  ;  when Speed = 0 and

       =00000008                    org         8
       =00000008       Timer20L     ds  1                  ; Counters  for
       =00000009       Timer20H     ds  1                  ;  20 ms timer
       =0000000A       Ltimer       ds  1                  ; Counter for left serv
       =0000000B       Rtimer       ds  1                  ; Counter for right ser
       =0000000C       LSpeed       ds  1                  ; Left servo speed
       =0000000D       RSpeed       ds  1                  ; Right servo speed
       =0000000E       Speed        ds  1                  ; The "Bot's" speed
       =0000000F       Turn         ds  1                  ; The "Bot's" turn fact
```

As in SX-Key, the upper half of the window shows the “SX internals”, where the lower half displays a view into the application’s list file that is currently open. In SXSim, this file is not shown in a separate window that may be moved, re-sized, or closed, but in the lower fixed area of the main window.

In the upper half of the window display, the register contents of Bank 0 in hexadecimal, and binary representation are displayed, 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.

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

The center area that is used in SX-Key to display the current instruction code has been removed in SXSim. Instead, the “Next Instruction” field has been introduced which shows 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 of 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” display, and the CALL-Stack display.

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 SK-Key IDE.

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

Again, this area is similar to SX-Key’s list window with the exception that here, each line has a small check-box to the left. Click on any check-box of a line containing executable code in order to set a breakpoint 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” have a small check marker. In order to remove a breakpoint, simply click on a checked box again to remove the breakpoint.

Whenever you load another list file, or quit SXSim, the active breakpoints, and some other information will be saved to a file with the list file’s name, and the “.SIM” extension. When you re-load the list file later, all breakpoints, and other settings will be restored again.

Above the list file section, there are some buttons that are unique to SXSim. These buttons are used to quickly navigate through the list file.

CLR all BPs

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

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).

Goto Main

Positions the current list cursor to the line that contains the label “Main”. This label is commonly used to mark the program’s main entry point, i.e. the RESET directive would read “RESET Main”. Although you are free to assign any other name to the main entry point, it makes sense to use “Main” in order to use that SXSim feature (the SX-Key IDE has a similar function – which is another good reason to use that name).

Goto 0
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 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 entered 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). 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 of the list file.

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

The “Commands” Section

This section contains the controls that are used to control SXSim. Some of them are “well-known” to users of SX-Key, but some are unique to SXSim.

Step

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

Walk

Click this button to repeatedly execute instructions starting at the instruction that is currently highlighted by the list cursor in the list file section of the SXSim window in “slow motion”. As in the “Step” mode, any changes of SX registers will be displayed, and recently changed registers are highlighted with a read background.

Click the “Stop” button to cancel the “Walk” mode. Clicking the “Reset” button instead, also cancels the “Walk” mode, but also 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 right of this button – left-click into that field to restore the default value of 100 ms).

+

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).

Break Walk

When this box is checked, the Walk mode will be terminated on code lines that are marked with a breakpoint.

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 in intervals of one second only.

Click the "Stop" button to cancel the "Run" mode. Clicking the "Reset" button instead, also cancels the "Run" mode, but also performs a simulated SX reset.

Poll

While the "Run" mode is active, the register display will only be updated every second. Click the "Poll" button at any time to refresh the register display in between.

Stop

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

Reset

Click this button to perform a reset of the simulated SX. This will also terminate active "Walk" and "Run" modes. After a reset, various SX register will be initialized to their defaults.

Please note that a reset in SXSim also clears registers that are not automatically cleared in a "real" SX, like the file registers in the various banks.

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 active. Whenever you open another list file, or when you terminate SXSim, some settings of SXSim (like breakpoints, I/O panel configuration, etc.) 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). When this is the case, a dialog will be shown, and you have the choice to update the list file, or to go ahead with the currently loaded version.

Quit

The function of this button is obvious – it terminates SXSim, but saves the currents 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 window.

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 field to the right of the “Clock/MHz” field either displays the total number of clock cycles that has elapsed for program execution so far. When SXSim “knows” the clock frequency, you may toggle the field display between the total number of elapsed clock cycles, or the total elapsed time in µs by clicking the Button above the field, initially named “Cycles”. The contents of this field will be cleared when you click the “Reset” button, or load another list file. You may also clear it in between by clicking the “CLR” button to the right of this field.

The ISR Cycles field shows the maximum number of clock cycles required to execute the ISR code, provided that there is ISR code at all, and that this code was executed at least once. For ISRs with variable execution times, the field always displays the greatest number of clock cycles so far. This information is quite helpful to check if the ISR code is too large to be executed within one interrupt period when you are using RTCC timed interrupts.

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

The Stack Section

The eight field under the “Stack” title 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 subroutine, you will see how the already stored return address is advanced on position up, and how now the second return address is displayed in the bottom field. The eight-level stack of the SX allows 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, or RETW instruction is executed, you will notice how the lowest address is “popped” from the stack, and how higher return addresses (if any) drop down one position.

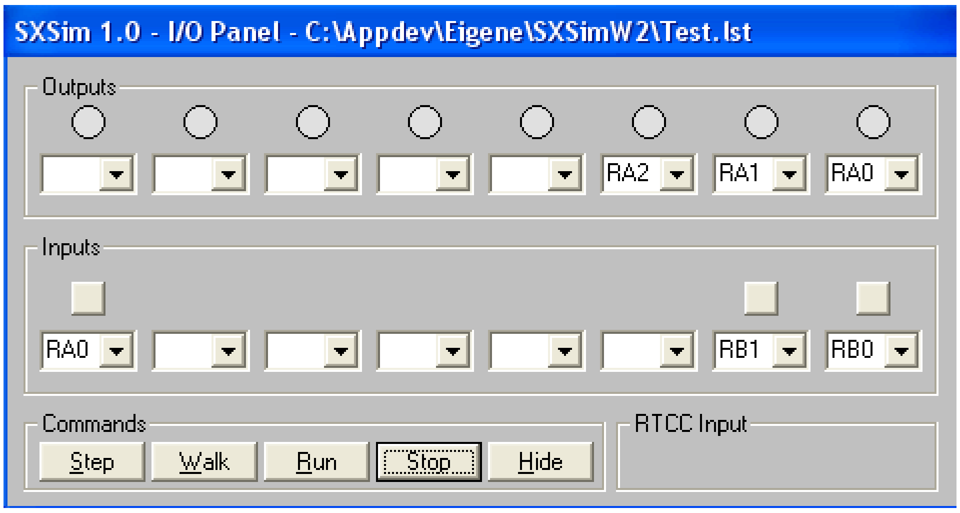
Executing a RET, or RETW instruction without a prior CALL on a “real” SX leads to unpredictable results with the program running into “Nirvana”. SXSim checks for such situations, and reports a Stack Underflow error.

As return addresses are pushed on the stack, you will notice that all fields that had contained a return address will be highlighted. When the addresses are “popped” from 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 I/O Panel

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



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

The “Outputs” section contains eight “virtual LEDs” that may be connected to any of the port pins of the simulated SX.

Either click into the text area of the combo boxes below the “LEDs”, and enter a legal port bit information, e.g. “RB3”, or select the port bit from the drop-down list. An empty field means that the LED is “not connected”.

When an LED is “connected” to one of the SX I/O pins, it’s color will change to red as soon as the assigned output has high level.

The “Inputs” section contains eight pushbuttons/switches that may be connected to any of the port pins of the simulated SX.

Similar to the “Outputs” section, you may select an SX I/O pin for each button from the associated combo boxes. Please note that an input button will only become visible when the assigned SX port pin has been defined as an input. By default, all SX I/O pins are inputs after reset, but this may be changed later by MOV !Rx, w instructions later.

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), or right-click the button to toggle its state. Whenever a button is “pressed”, i.e. left-clicked, or “toggled on”, it will be displayed in green.

The “Commands” section of the I/O Panel contains most of the buttons that are available in the “Commands” section of the main display. The can be alternatively used to single-step, walk, run, or stop a simulation.

Click the “Hide” button to hide the I/O panel. In order to display the I/O panel again, click the “I/O Panel” button in the main display again.

When terminating SXSim, or when you select another LST file by clicking the "Load File" button, the current I/O Panel port bit assignments, together with some other project-specific settings, like breakpoints are saved in a file with the LST file name and a ".SIM" extension. When you re-load the same LST the next time, breakpoints, and port-assignments for the I/O Panel will be restored automatically,

A Sample SXSim Session Featuring the I/O Panel

Copy the files TEST.LST, and TEST.SIM that came with SXSim, launch SXSim, and click the "Load File" button.

In the file select dialog that opens next, navigate to the TEST.LST file, and open it.

Here is the source code that was used to generate TEST.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
      mov   ra, w               ; Output the WKPND_B bits to port A
      reti
:RTCCInt
      xor   ra, #%00000100      ; Toggle RA.2
      mov   RTCC, #250
      reti

      org   $100
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 on port B.1 and negative edges on port B.0, and configures port A bits 2 through 0 as outputs.

Then, the RTCC register is initialized to 250, and RTCC interrupts on positive edges of the RTCC input are enabled with no prescaler.

Finally, an endless loop is entered.

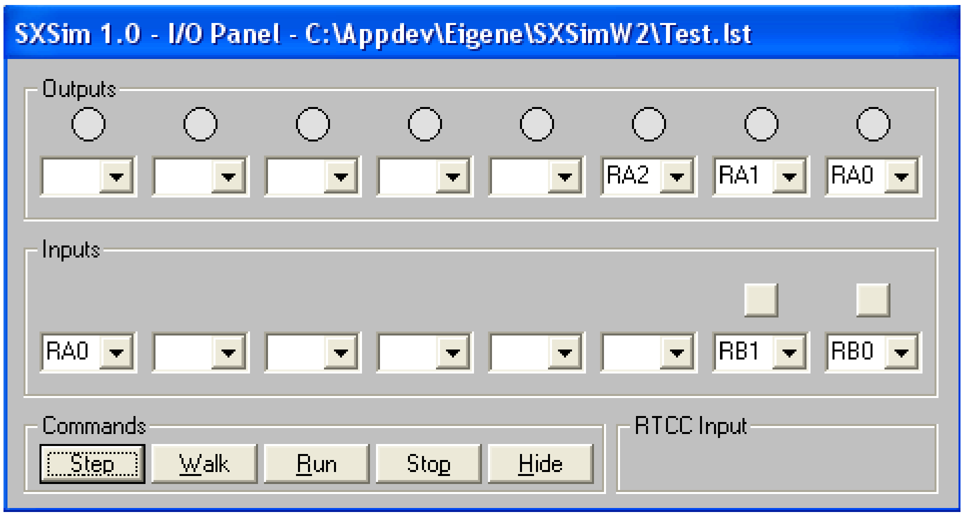
In the Interrupt Service Routine (ISR), the contents of the WKPND_B register and w are exchanged. As w was cleared before, WKPND_B is cleared afterwards (which is important to avoid more interrupts before another bit in WKPND_B is set again).

When no bits in WKPND_B were set, execution is continued at :RTCCInt, else the former contents of WKPND_B is copied to the port A register, and the ISR is terminated with the RETW instruction (we will discuss the code following :RTCCInt later).

After TEST.LST has been loaded, click the "I/O Panel" button to show the I/O Panel. It should have the input and output configuration, as shown above.

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 button assigned to RA0 becomes invisible after the instructions for MOV !RA, #%11111000 have been executed. RA0 has been configured as an output now, so you may no longer force this pin to high or low level with a pushbutton "connected" to it. The I/O panel should now look like this:



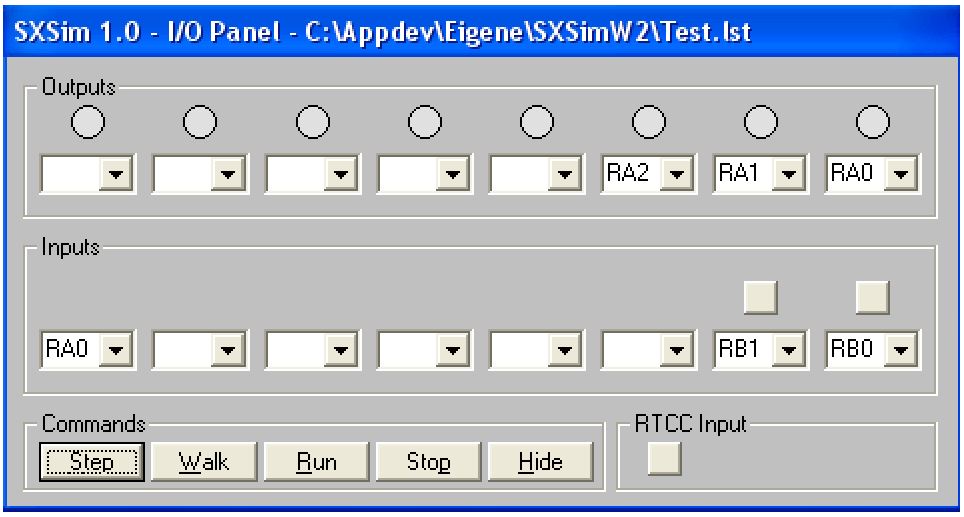
As you step over the instructions to set the OPTION register, a button will become visible in the RTCC Input section (we will discuss this in more detail later).

Next, "Walk" the program, and click and hold the button "connected" to the RB.1 input. In the SXSim main window, you will see how the program flow enters the ISR, and how the "LED" assigned to RA1 goes on. Note that the ISR code is entered as soon as you have started "holding down" the button because RB.1 is configured to trigger interrupts on positive edges. Release the button again.

Next, click and hold the button "connected" to the RB.0 input for a while. You will notice that the ISR code is not entered before you release this button again, because RB.0 was configured to trigger interrupts on negative edges.

The RTCC Input section

When bit 6 in the OPTION register is cleared, i.e. RTCC interrupts are enabled, and bit 5 in the OPTION register is set, i.e. when external RTCC clock signals are enabled, the button in the RTCC Input section becomes visible:



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 is incremented 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 that this is the case.

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 always displays the maximum number of ISR cycles required for an ISR execution so far.

The END

This ends the SXSim documentation for now. Please stay tuned, more features will be coming soon…

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

Guenther Daubach
g.daubach@mda-burscheid.de