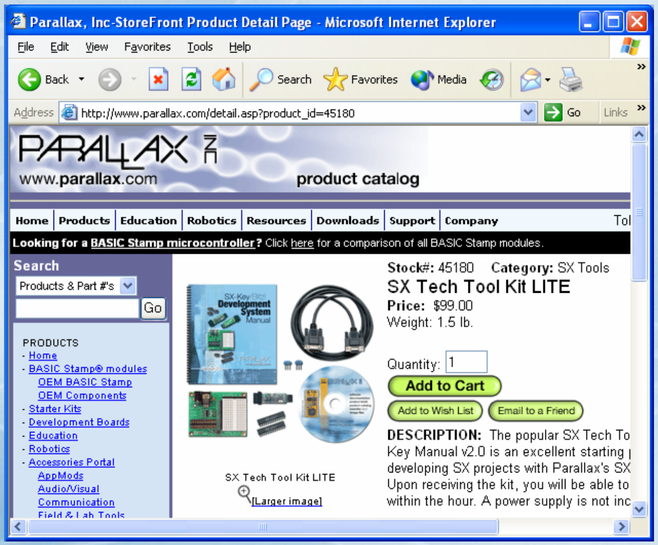
SX-Tech Toolkit and SX/B


Parallax, Inc-StoreFront Product Detail Page - Microsoft Internet Explorer

Address http://www.parallax.com/detail.asp?product_id=45180

www.parallax.com product catalog

Home | Products | Education | Robotics | Resources | Downloads | Support | Company

Looking for a BASIC Stamp microcontroller? Click here for a comparison of all BASIC Stamp modules.

Search

PRODUCTS
- Home
- BASIC Stamp® modules
  - OEM BASIC Stamp
  - OEM Components
- Starter Kits
- Development Boards
- Education
- Robotics
- Accessories Portal
  - AppMods
  - Audio/Visual
  - Communication
  - Field & Lab Tools

Stock#: 45180 **Category:** SX Tools

SX Tech Tool Kit LITE

Price: $99.00

Weight: 1.5 lb.

Quantity: 1

Add to Cart

Add to Wish List | Email to a Friend

DESCRIPTION: The popular SX Tech To
Key Manual v2.0 is an excellent starting
developing SX projects with Parallax's SX
Upon receiving the kit, you will be able to
within the hour. A power supply is not inc

SX Tech Tool Kit LITE

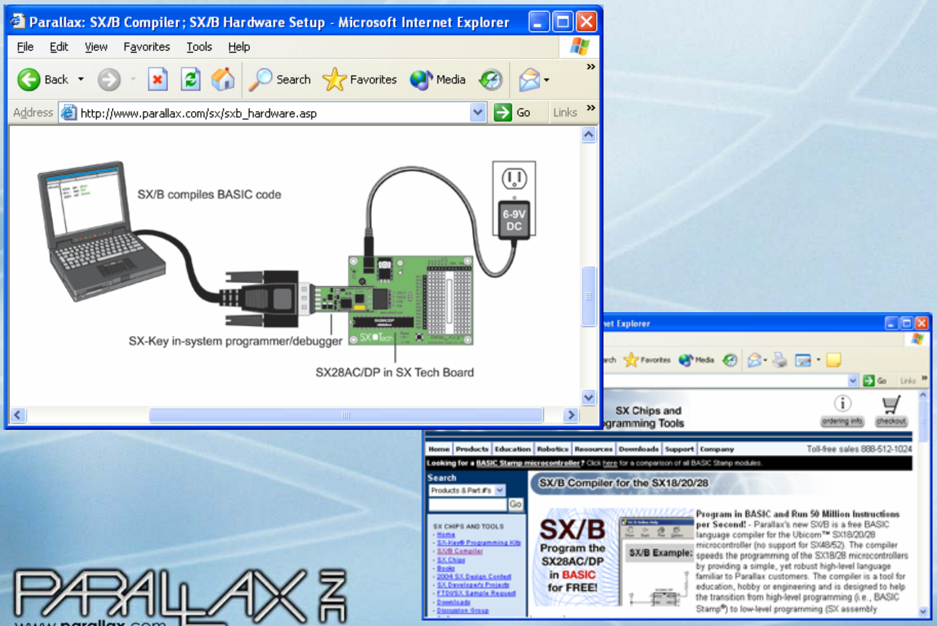
[Larger image]

www.parallax.com

Point out that we are experimenting with the SX Tech Toolkit, which has the SX-Key programming and debugging tool, the SX Tech carrier board with socket and breadboard, two 28 pin versions of the SX chip, a 4 MHz oscillator, a 50 MHz oscillator, and a serial cable for programming. (Dr. Matthews, there's a USB to serial converter included in your package in case your PCs are USB only.)

Explain that this kit's 28 pin SX microcontroller has 20 I/O pins, and that there's also a 48 pin version with 36 I/O pins (and some extra RAM and Flash program memory too). Emphasize that you can do really involved projects with the SX28 and that many students with projects will never encounter limitations that would make them consider the SX48. Additionally, SX28 chips can easily be networked for larger projects as well as group projects where individual students are making modules. While SX48 chips are only available in SMT packages, there is also an SX48 Proto Board (Stock#: 45300) if the project really does requires that much memory and/or I/O pins.

SX-Tech Toolkit and SX/B


Parallax: SX/B Compiler; SX/B Hardware Setup - Microsoft Internet Explorer
http://www.parallax.com/sx/sxb_hardware.asp
SX/B compiles BASIC code
6-9V DC
SX-Key in-system programmer/debugger
SX28AC/DP in SX Tech Board
SX Chips and Programming Tools
Home | Products | Education | Robotics | Resources | Downloads | Support | Company
Toll-free sales 888-512-1024
SX/B Compiler for the SX18/20/28
SX/B Program the SX28AC/DP in BASIC for FREE!
SX/B Example:

Looking for a BASIC Stamp microcontroller? Click here for a comparison of all BASIC Stamp modules.

Program in BASIC and Run 50 Million Instructions per Second! - Parallax's new SX/B is a free BASIC language compiler for the Ubicom™ SX18/20/28 microcontroller (no support for SX48/52). The compiler speeds the programming of the SX18/28 microcontrollers by providing a simple, yet robust high-level language familiar to Parallax customers. The compiler is a tool for education, hobby or engineering and is designed to help the transition from high-level programming (i.e., BASIC Stamp®) to low-level programming (SX assembly

www.parallax.com

Explain the power supply and programming/debugging connection between board and PC.

The software for programming and debugging is called the SX Key IDE (integrated development environment).

This IDE supports two languages: SX/B (BASIC) and SX assembly. Programs can also be a mixture of the two languages.

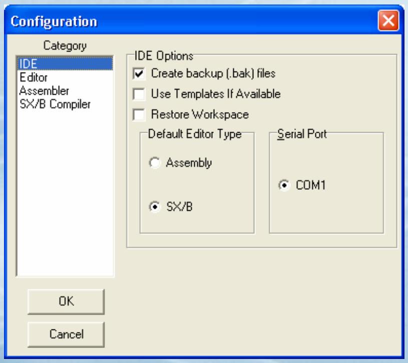
Download the SX Key IDE from http://www.parallax.com/sx/sxb.asp.

Download the SX Key/Blitz Development System Manual from the same page (http://www.parallax.com/sx/sxb.asp.)

Connect the SX Tech board (with SX chip) as shown in the slide.

Run the software.

Software Settings

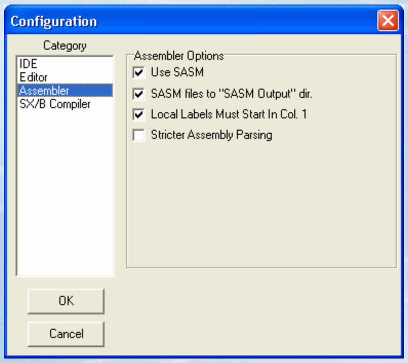

Configuration

Category: IDE [✓ IDE] Editor, Assembler, SX/B Compiler

IDE Options
- [✓] Create backup (.bak) files
- [ ] Use Templates If Available
- [ ] Restore Workspace

Default Editor Type: Assembly [✓ SX/B]

Serial Port: [✓ COM1]


Configuration

Category: IDE, Editor, [✓ Assembler] SX/B Compiler

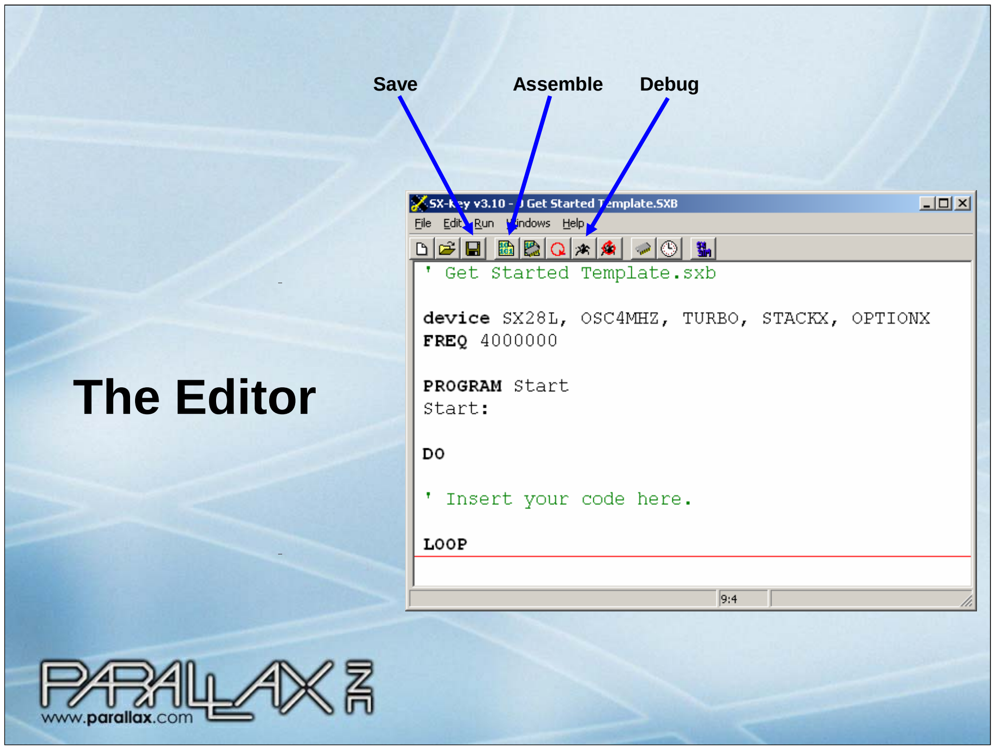
Assembler Options
- [✓] Use SASM
- [✓] SASM files to "SASM Output" dir.
- [✓] Local Labels Must Start In Col. 1
- [ ] Stricter Assembly Parsing

www.parallax.com

3

If the software doesn’t automatically make you look at these screens, click Run and Select Configure. Click the IDE category, and make sure SX/B is selected.

Click the Assembler category, and make sure the boxes shown on the right are checked/not checked as shown.


Save
Assemble
Debug
SX-Key v3.10 - 0 Get Started Template.SXB
File Edit Run Windows Help
9:4
www.parallax.com

The Editor

```
' Get Started Template.sxb

device SX28L, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ 4000000

PROGRAM Start
Start:

DO

' Insert your code here.

LOOP
```

The example programs are in a zip file that accompanies this slide show. This program is an empty template that does nothing (over and over again) since there are no commands between the DO and LOOP keywords. The device directives tell the IDE that you are writing programs for the 28 pin SX chip (SX28L), that the chip should use its internal 4 MHz oscillator (OSC4MHZ). There are lots of different oscillator settings for either internal or external oscillators. From the Device Settings table in the *SX-Key/Blitz Development System Manual*, you can find the settings for the SX Tech Toolkit's external 4 MHz and 50 MHz oscillators (OSCXT2 and SXCHS3). There are three more settings that are beyond the scope of this introduction, but they are necessary for the SX/B compiler, and there's almost never a reason to change them or leave them out.

NOTE: You can use the *SX-Key/Blitz Development System Manual* to find out about device directives. You can also click Help -> SX/B Help to find out about most of the other keywords in the program (DO, LOOP, PROGRAM, FREQ, etc.).

If you end up using the debugging tools (next slide) FREQ 4000000 tells the SX-key to clock the SX chip at 4 MHz while it's in debugging mode. If it's running on its own, it will instead use the device…OSC4MHZ directive.

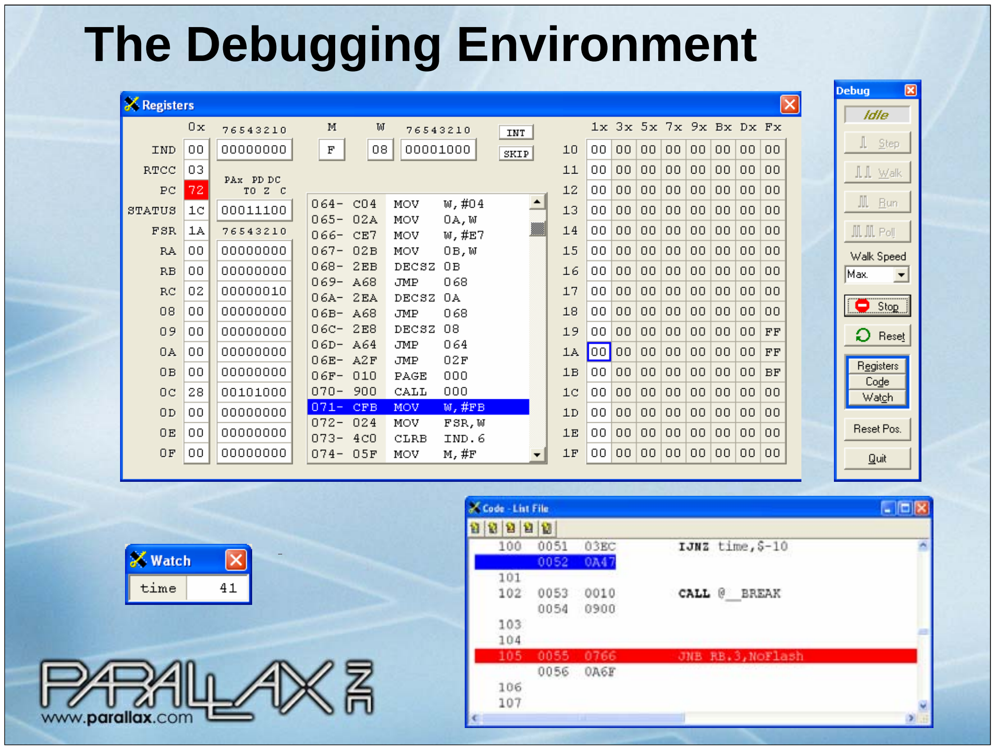
PROGRAM Start tells the compiler to start at the Start: label. Sometimes, there is an interrupt service routine between the PROGRAM directive and the label that signifies the beginning of the program. This will be explained in the sixth and last demonstration program that accompanies this slide show.

To debug a program, click the Save, Assemble and Debug buttons (in that order). There are also buttons for Run and Program that you would use to program the chip to run its program on its own, without being connected to the software and SX-Key. There are also more options you can see by clicking the IDE's Run menu.

Open "0 Get Started Template.SXB" (File -> Open).

Then, click the Save, Assemble and Debug buttons.

The Debugging Environment



Registers

	0x	76543210
IND	00	00000000
RTCC	03	
PC	72	
STATUS	1C	00011100
FSR	1A	
RA	00	00000000
RB	00	00000000
RC	02	00000010
08	00	00000000
09	00	00000000
0A	00	00000000
0B	00	00000000
0C	28	00101000
0D	00	00000000
0E	00	00000000
0F	00	00000000

PAx PD DC
TO Z C

M	W	76543210
F	08	00001000

INT
SKIP

```
064-  C04   MOV    W,#04
065-  02A   MOV    0A,W
066-  CE7   MOV    W,#E7
067-  02B   MOV    0B,W
068-  2EB   DECSZ  0B
069-  A68   JMP    068
06A-  2EA   DECSZ  0A
06B-  A68   JMP    068
06C-  2E8   DECSZ  08
06D-  A64   JMP    064
06E-  A2F   JMP    02F
06F-  010   PAGE   000
070-  900   CALL   000
071-  CFB   MOV    W,#FB
072-  024   MOV    FSR,W
073-  4C0   CLRB   IND.6
074-  05F   MOV    M,#F
```

	1x	3x	5x	7x	9x	Bx	Dx	Fx
10	00	00	00	00	00	00	00	00
11	00	00	00	00	00	00	00	00
12	00	00	00	00	00	00	00	00
13	00	00	00	00	00	00	00	00
14	00	00	00	00	00	00	00	00
15	00	00	00	00	00	00	00	00
16	00	00	00	00	00	00	00	00
17	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00
19	00	00	00	00	00	00	00	FF
1A	00	00	00	00	00	00	00	FF
1B	00	00	00	00	00	00	00	BF
1C	00	00	00	00	00	00	00	00
1D	00	00	00	00	00	00	00	00
1E	00	00	00	00	00	00	00	00
1F	00	00	00	00	00	00	00	00

Watch

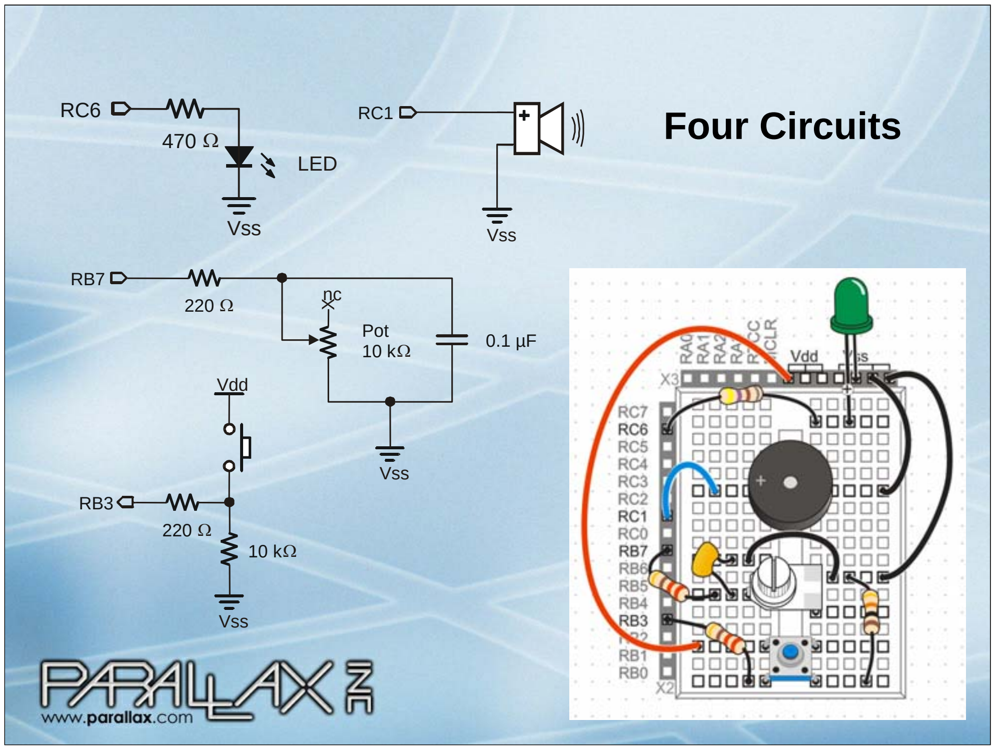
time	41

Code - List File

```
100   0051   03EC          IJNZ time,$-10
      0052   0A47
101
102   0053   0010          CALL @__BREAK
      0054   0900
103
104
105   0055   0766          JNB RB.3,NoFlash
      0056   0A6F
106
107
```

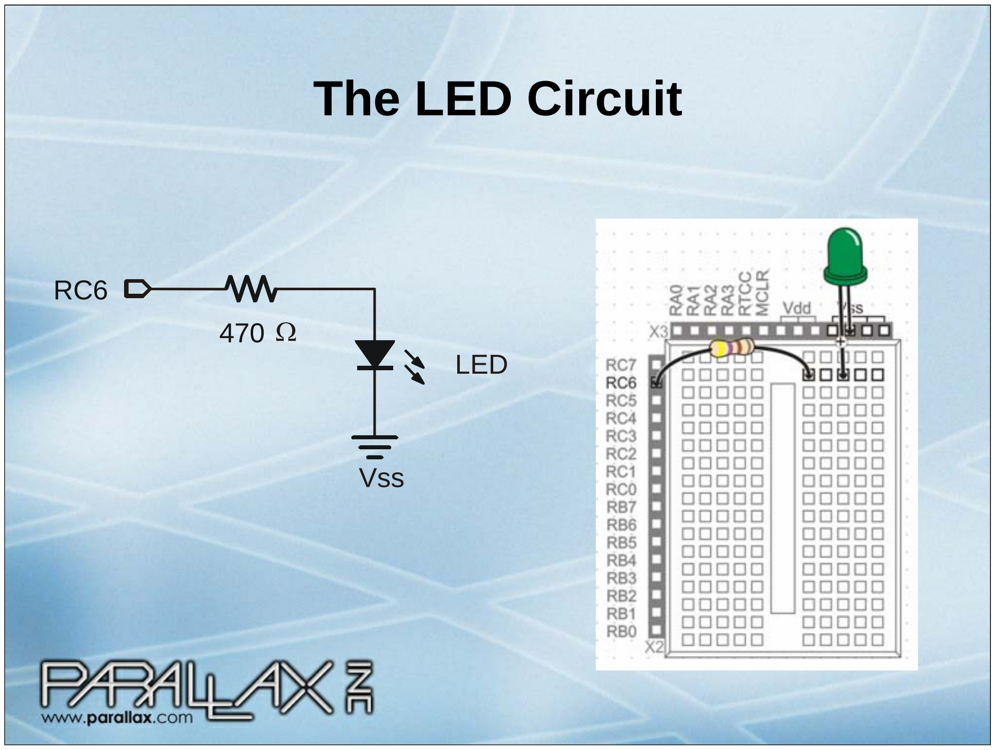
The debugging environment will look similar to this. The registers window shows the state of the SX chip's RAM. The Code – List file shows the assembly version of the SX/B code, along with commented SX/B code commented to the right of the corresponding assembly code. The Code – List File window also shows the line number and machine codes. The blue bar in the Code – List File window shows the current command getting executed, and you can click a line in the window to make the red line appear. That's a breakpoint, and by clicking it again, you can toggle the breakpoint off. The Debug window has buttons for stop, Walk, Run, and Poll. Poll is ideal for when you have a WATCH directive, which will cause the Watch window to show the value of one or more variables. The Poll mode updates all the registers (and the Watch window) every it passes the break point. The break point will cause other debug modes such as Run, and Walk to halt when they get there. Run is for full speed execution, and Stop is to manually halt full speed execution.

Click Quit to exit the Debugging environment.

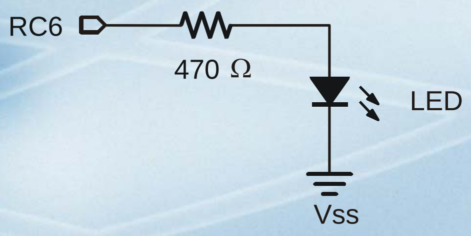


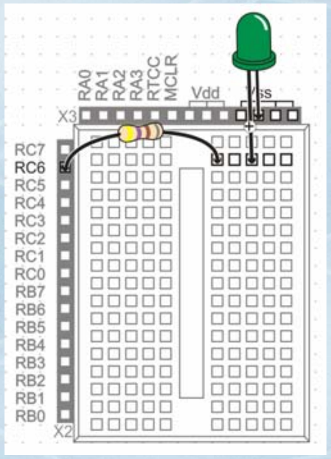
Four Circuits

Here are the four circuits the example programs will work with along with a wiring diagram. For more information about these circuits, consult What's a Microcontroller by Andy Lindsay (Chapters 2, 3, 5, and 8). It's available for PDF download (no charge) from www.parallax.com.



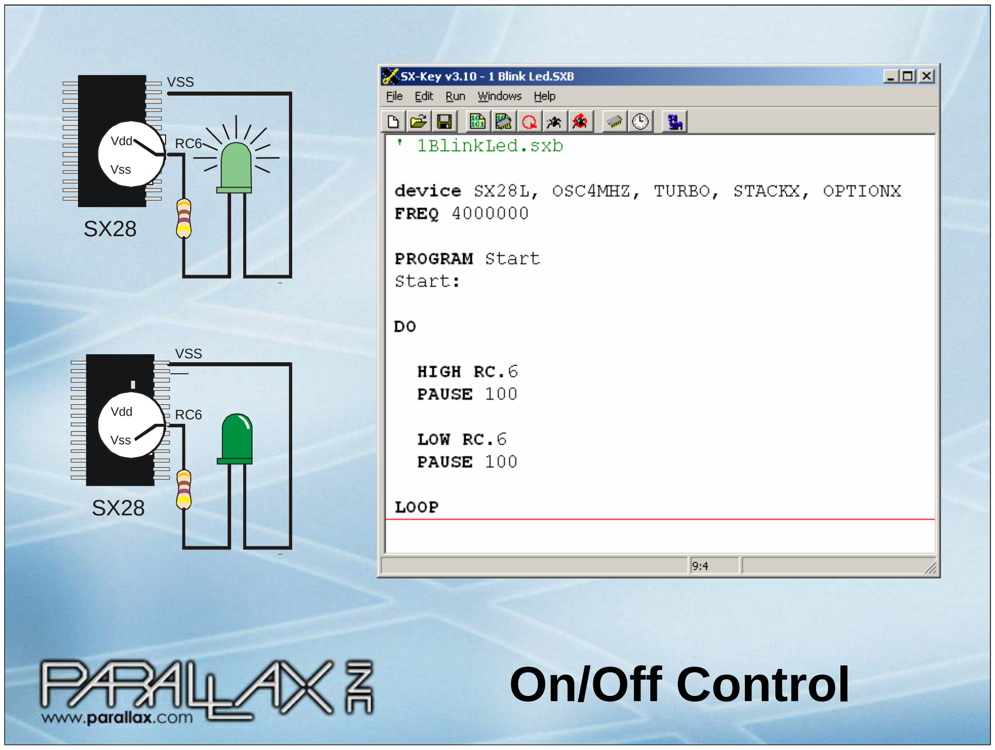
The LED Circuit





www.parallax.com

We’ll start with the LED circuit. Explain LED circuit theory of operation.
Note that the LED is connected to RC6!!!

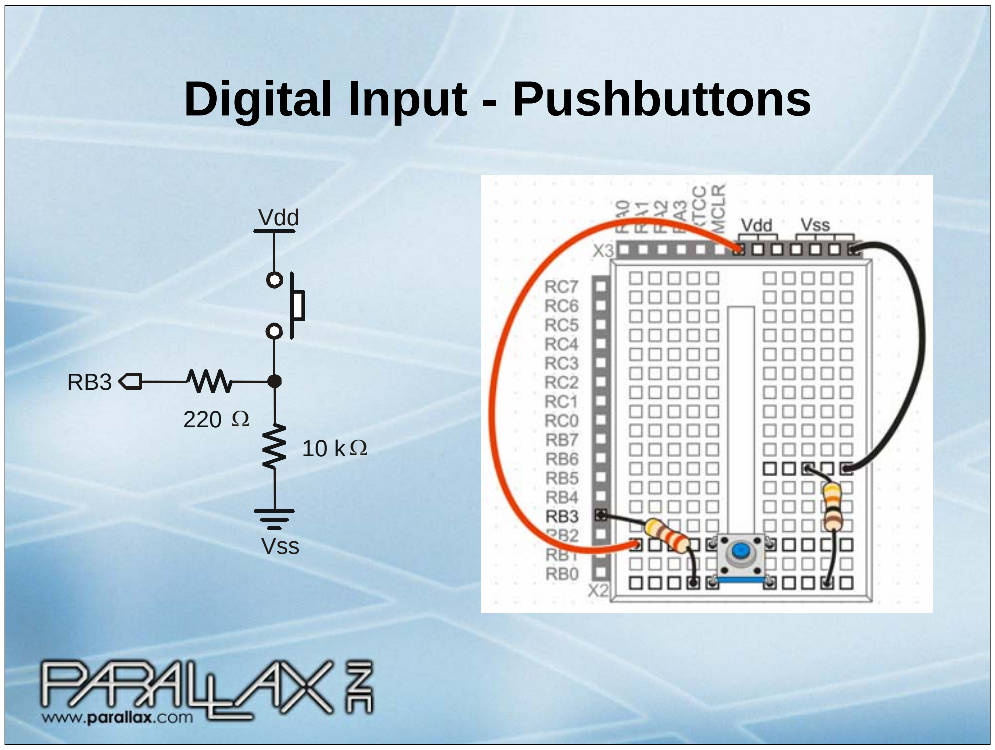


```
' 1BlinkLed.sxb

device SX28L, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ 4000000

PROGRAM Start
Start:

DO

  HIGH RC.6
  PAUSE 100

  LOW RC.6
  PAUSE 100

LOOP
```

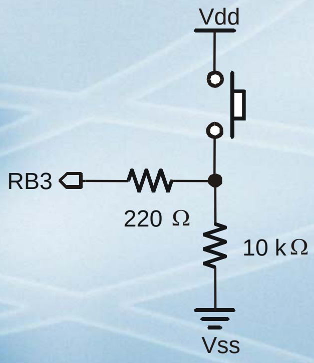
On/Off Control

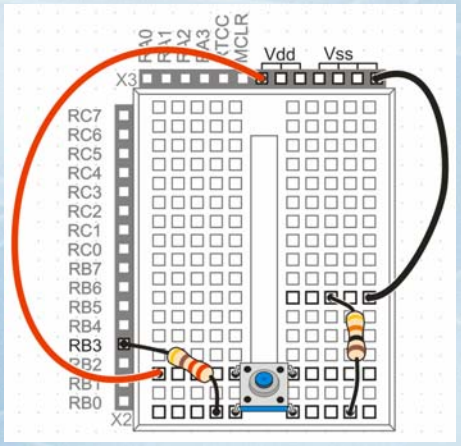
Instruct group to open “1 Blink Led.SXB”.

Explain how the chip is programmed to short RC6 alternately to Vdd and Vss using the HIGH and LOW commands. The PAUSE commands make the LED blink on/off at human speed instead of electronic speed. Because the HIGH, LOW, and PAUSE commands are between DO and LOOP, they are repeated indefinitely.



Digital Input - Pushbuttons

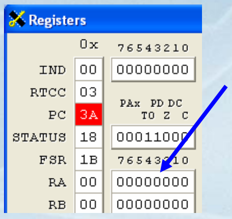




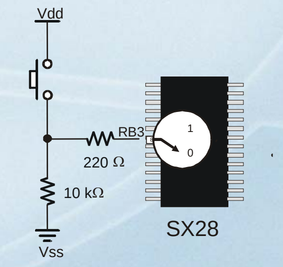
www.parallax.com

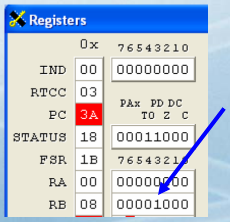
Commands like HIGH, LOW, and TOGGLE can make the pin an output. However it will default to a high impedance input of none of those commands are used. As an input, the I/O pin will sense 0 V (Vss) if the button is not pressed, or 5 V if the pushbutton is pressed. The 220 ohm resistor is there to protect the I/O pin in case a LOW command is used on RB3 by accident. Without the 220 resistor, this command would cause the chip to try to short RB3 to ground, and it would battle with the Vdd connection until its I/O drivers fry.

Sensing a Pushbutton

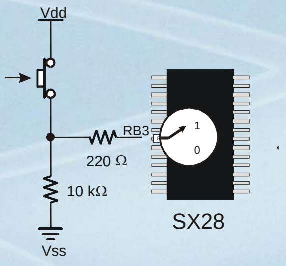


```
Registers
          0x   76543210
   IND    00   00000000
  RTCC    03
                PAx PD DC
    PC    3A      TO  Z  C
STATUS    18   00011000
   FSR    1B   76543210
    RA    00   00000000
    RB    00   00000000
```





```
Registers
          0x   76543210
   IND    00   00000000
  RTCC    03
                PAx PD DC
    PC    3A      TO  Z  C
STATUS    18   00011000
   FSR    1B   76543210
    RA    00   00000000
    RB    08   00001000
```



www.parallax.com

10

The RB register stores the states of all the RB bank of I/O pins. When you press and release the pushbutton, other RB pins that are floating may change state, but RB.3 will change with the pushbutton. To observe this:

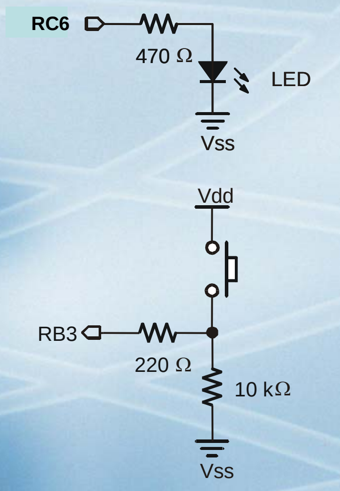
Save, Run, Debug

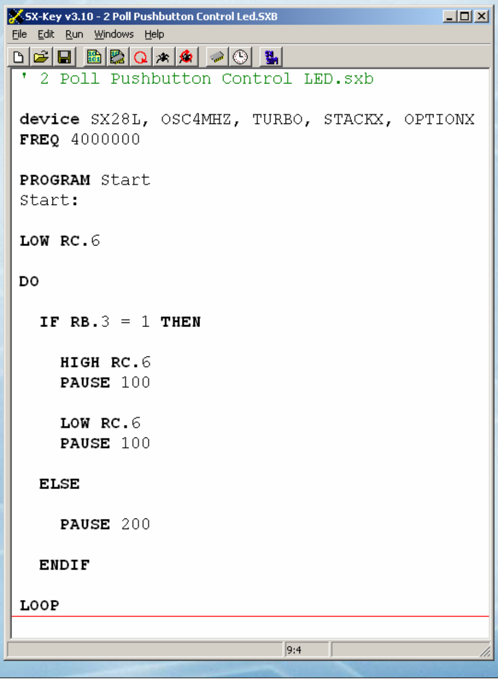
Set a breakpoint at the DO keyword in the Code – File List window.

Click Poll in the Debug window.

Press and release the pushbutton while watching the RB3 bit carefully. It will be 1 if the I/O pin senses above 2.5 V or 0 if it's below 2.5 V.

Pushbutton Control of LED Circuit




SX-Key v3.10 - 2 Poll Pushbutton Control Led.SXB
File Edit Run Windows Help
9:4

```
' 2 Poll Pushbutton Control LED.sxb

device SX28L, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ 4000000

PROGRAM Start
Start:

LOW RC.6

DO

  IF RB.3 = 1 THEN

    HIGH RC.6
    PAUSE 100

    LOW RC.6
    PAUSE 100

  ELSE

    PAUSE 200

  ENDIF

LOOP
```

www.parallax.com

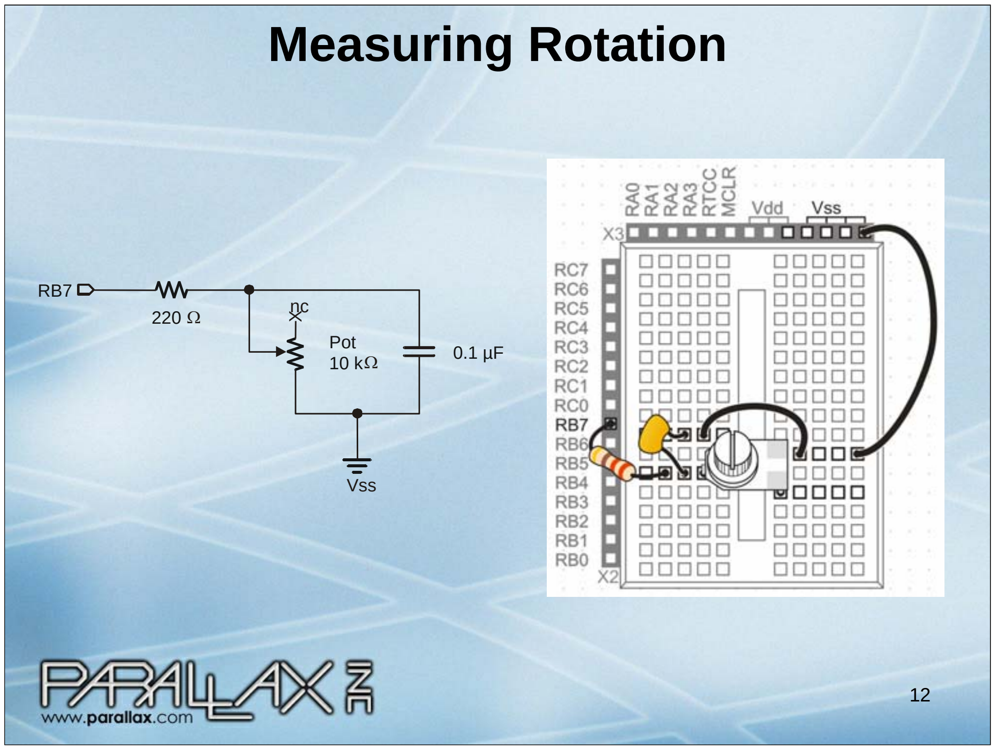
11

This program uses IF…THEN statements to make the LED blink when the pushbutton is pressed (RB.3 = 1). Otherwise (ELSE), it just pauses for 2/10 of a second and checks again.

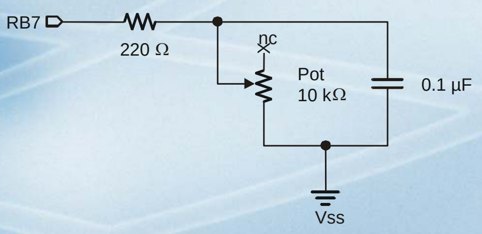
Open/Save/Compile/Run “2 Poll Pushbutton Control LED.sxb”

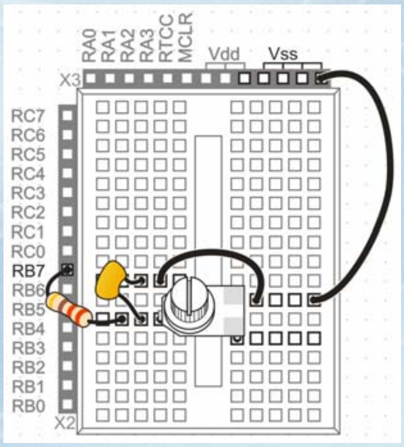
Note that nothing is happening.

Press and hold the pushbutton and verify that it causes the light to blink.



Measuring Rotation

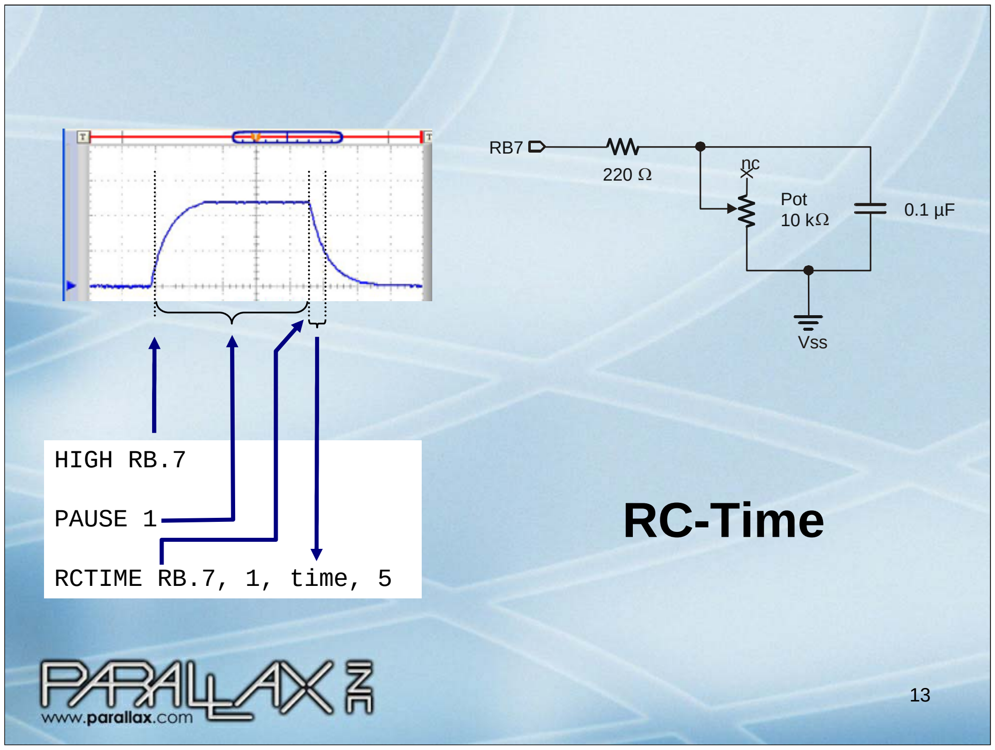




www.parallax.com

12

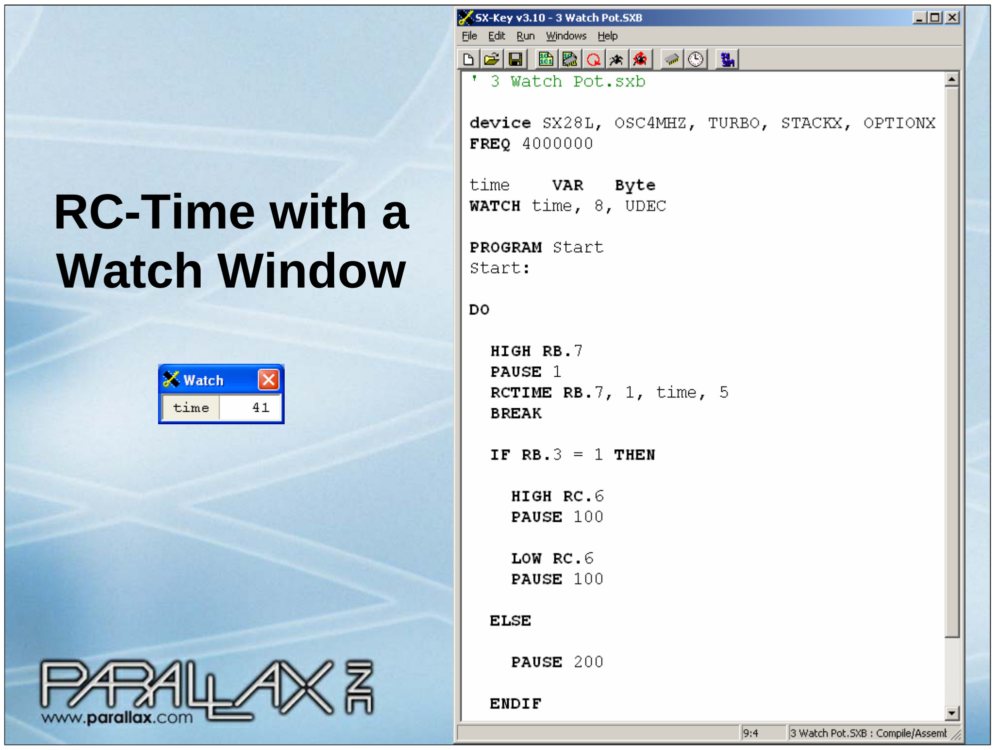
I/O pins can also be bidirectional. To measure the resistance of this potentiometer (pot), the I/O pin can be set to output high to charge the capacitor. Then, it can be changed to input, and the SX chip can measure the amount of time it takes the capacitor to discharge to the 2.5 V input logic threshold voltage through the POT. This decay time would be directly proportional to the pot's resistance if it weren't for the voltage divider caused by the 220 resistor.



RC-Time

```
HIGH RB.7

PAUSE 1

RCTIME RB.7, 1, time, 5
```

Although you could write an assembly routine to do this accurately, or a SX/B routine to do it fairly well, there’s already an SX/B command called RCTIME that results in assembly code that measures the RC decay time for you. The SX/B command is called RCTIME. Try the example program shown in the next slide, and then click Help -> SX/B help to find out more about the RCTIME command.



RC-Time with a Watch Window

Watch
time 41
SX-Key v3.10 - 3 Watch Pot.SXB
File Edit Run Windows Help
9:4
3 Watch Pot.SXB : Compile/Assemt
www.parallax.com

```
' 3 Watch Pot.sxb

device SX28L, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ 4000000

time     VAR   Byte
WATCH time, 8, UDEC

PROGRAM Start
Start:

DO

  HIGH RB.7
  PAUSE 1
  RCTIME RB.7, 1, time, 5
  BREAK

  IF RB.3 = 1 THEN

    HIGH RC.6
    PAUSE 100

    LOW RC.6
    PAUSE 100

  ELSE

    PAUSE 200

  ENDIF
```

A variable declaration at the beginning of the program named a byte of the SX chip's RAM time. A WATCH directive was added to display the value the time variable stores at the breakpoint. A BREAK directive was added to cause the Debugging environment to refresh its display (including the watch window) after it takes each RCTIME measurement.

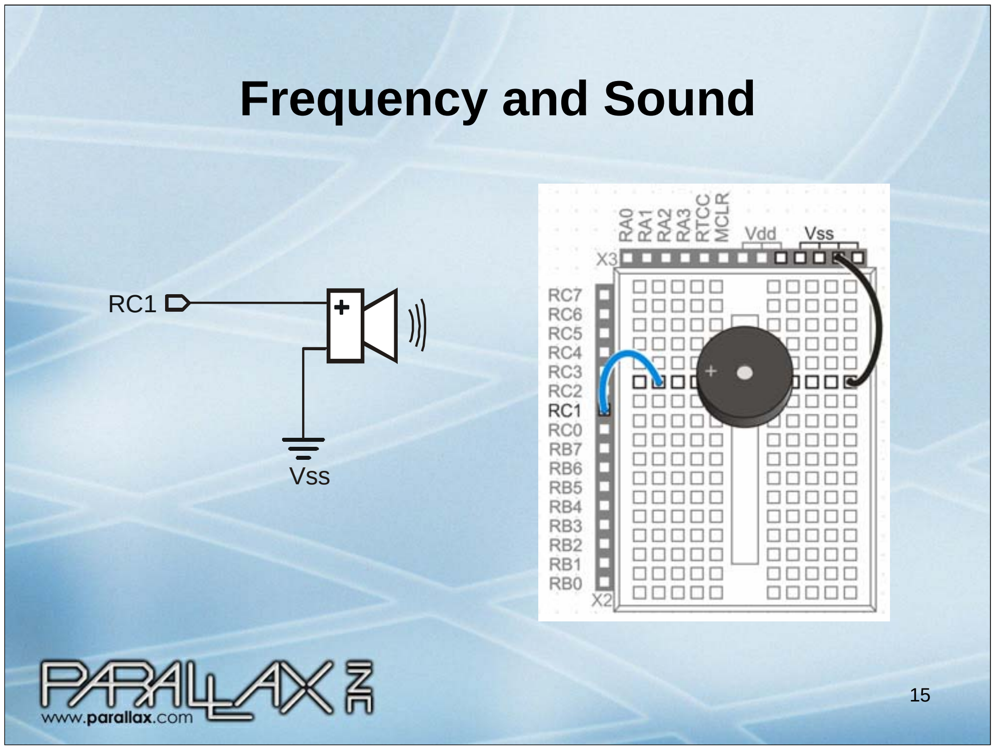
Open/Save/Compile/Debug "3 Watch Pot.sxb"

Click Poll in the Debug Window

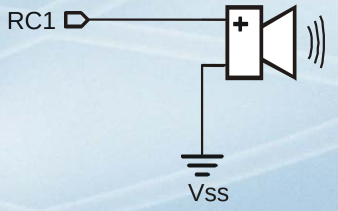
Turn the pot and watch the Watch window's value update with a value that corresponds to decay time, which in turn corresponds to resistance measured.

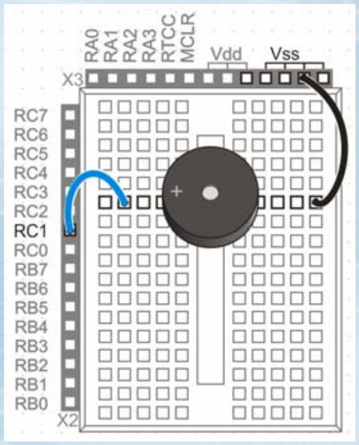
Note that the POT code has just been added to the LED code, so if you press and hold the pushbutton, the LED will still blink.

Rotate your pot so that the time is below 50 before moving on to the next example.



Frequency and Sound

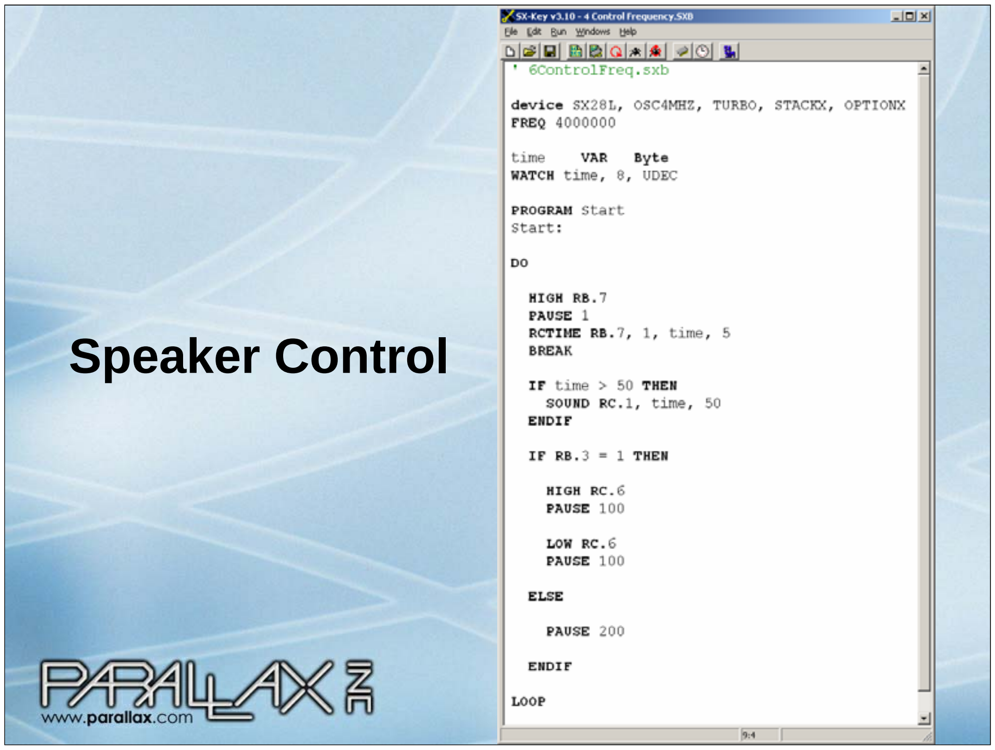




www.parallax.com

15

This piezospeaker works pretty well between about 1.5 and 3.6 kHz. It's resonant frequency is 4.5 kHz, at which point it's no longer musical, but it's perfect for smoke detector alarms.



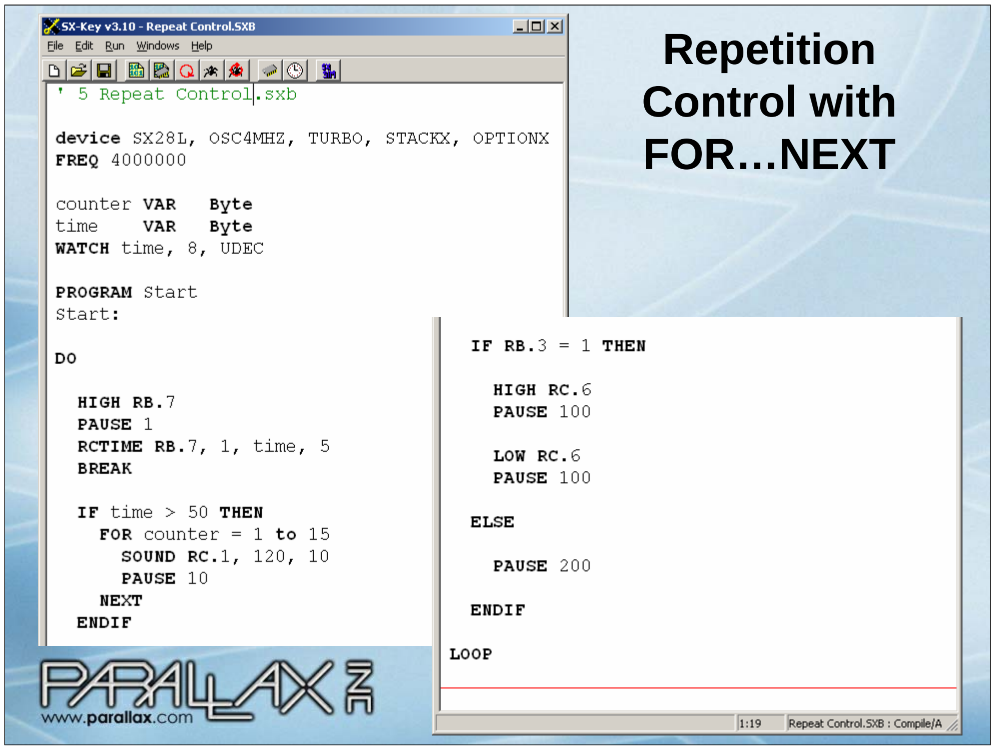
Speaker Control

```
' 6ControlFreq.sxb

device SX28L, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ 4000000

time     VAR   Byte
WATCH time, 8, UDEC

PROGRAM Start
Start:

DO

  HIGH RB.7
  PAUSE 1
  RCTIME RB.7, 1, time, 5
  BREAK

  IF time > 50 THEN
    SOUND RC.1, time, 50
  ENDIF

  IF RB.3 = 1 THEN

    HIGH RC.6
    PAUSE 100

    LOW RC.6
    PAUSE 100

  ELSE

    PAUSE 200

  ENDIF

LOOP
```

This program triggers a sound alarm when the pot from the previous example is twisted beyond a certain threshold (time > 50). Note that the time variable is also used by the time command. So, as you twist the pot higher above 50, the pitch will increase too.

NOTE: This example was written before SX/B was updated to support Word variables. With the advent of SX/B word variables, there’s also a new command called FREQOUT, which allows you to specify a frequency in Hz.

Also, note that the pushbutton LED control is still in effect, though it behaves somewhat differently when the program takes extra time to beep.



Repetition Control with FOR…NEXT

SX-Key v3.10 - Repeat Control.SXB

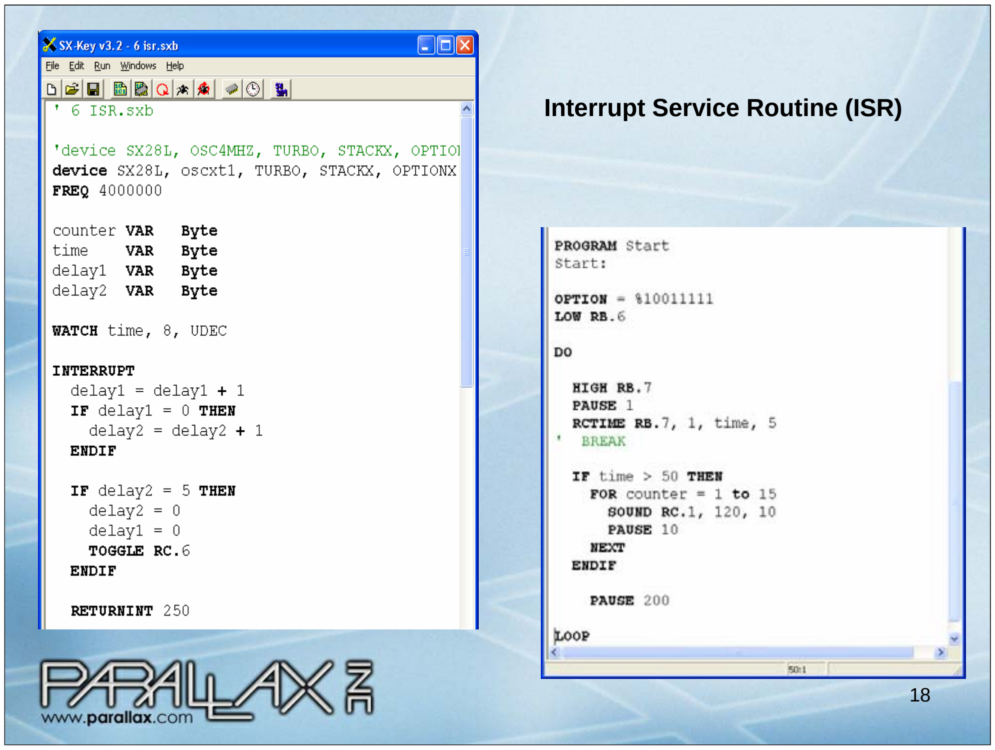
```
' 5 Repeat Control.sxb

device SX28L, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ 4000000

counter VAR   Byte
time    VAR   Byte
WATCH time, 8, UDEC

PROGRAM Start
Start:

DO

  HIGH RB.7
  PAUSE 1
  RCTIME RB.7, 1, time, 5
  BREAK

  IF time > 50 THEN
    FOR counter = 1 to 15
      SOUND RC.1, 120, 10
      PAUSE 10
    NEXT
  ENDIF
```

```
  IF RB.3 = 1 THEN

    HIGH RC.6
    PAUSE 100

    LOW RC.6
    PAUSE 100

  ELSE

    PAUSE 200

  ENDIF

LOOP
```

1:19   Repeat Control.SXB : Compile/A

www.parallax.com

This extension of the previous example program plays a multi-beep alarm instead of tracking the pot by frequency. This demonstrates the FOR…NEXT loop. The only part it doesn't show is the fact that the value of counter increments each pass through the FOR…NEXT loop. Try adding a watch directive and moving the BREAK to observe this counting action (I haven't tested this yet).



Interrupt Service Routine (ISR)

```
' 6 ISR.sxb

'device SX28L, OSC4MHZ, TURBO, STACKX, OPTIO
device SX28L, oscxt1, TURBO, STACKX, OPTIONX
FREQ 4000000

counter VAR   Byte
time    VAR   Byte
delay1  VAR   Byte
delay2  VAR   Byte

WATCH time, 8, UDEC

INTERRUPT
  delay1 = delay1 + 1
  IF delay1 = 0 THEN
    delay2 = delay2 + 1
  ENDIF

  IF delay2 = 5 THEN
    delay2 = 0
    delay1 = 0
    TOGGLE RC.6
  ENDIF

  RETURNINT 250
```

```
PROGRAM Start
Start:

OPTION = %10011111
LOW RB.6

DO

  HIGH RB.7
  PAUSE 1
  RCTIME RB.7, 1, time, 5
'  BREAK

  IF time > 50 THEN
    FOR counter = 1 to 15
      SOUND RC.1, 120, 10
      PAUSE 10
    NEXT
  ENDIF

    PAUSE 200

LOOP
```

An interrupt service routine is a segment of code that runs if an interrupt occurs. Interrupts can be one-shot, from a pin state change or periodic, based on a number of clock ticks. This example in the slide uses clock ticks. RETURNINT 250 sets a register called RTCC (real time clock counter) to 255. The SX chip's system automatically subtracts 1 from this register at a rate determined by the lowest four bits of the OPTION register (which is set to %10011111 for this example). When the RTCC gets to 0, the interrupt service routine gets serviced again, and the RTCC is set to 250 again. In this example, OPTION = %10011111 causes RTCC to be decremented every 256 clock pulses. If the lowest four bits of option was 1110, it would be once every 128 clock ticks. 1101 would be once ever 64 clock ticks, and so on down to 1000, which is once every 2 clock ticks. For once every clock tick, use %10010000 (all four low bits are zero).

Since the ISR gets serviced every 250 RTCC decrements regardless of whether the speaker is playing a note, or the PAUSE command is in progress or an RCTIME measurement is happening. Every 250 clock ticks, the delay1 variable in the ISR code is incremented by 1, and every time delay1 rolls over from 255 to 0, delay2 gets bumped up by 1. When delay2 gets up to 5, the LED state is toggled (high to low or low to high). The net result isn't all that exciting, all that happens is the LED flashes on/off regardless of what else is happening in the program. However, the applications are exciting. For example, you can use it to write serial communication code, take periodic A/D measurements, deliver servo pulses periodically, and much more.

Open/Save/Compile/Run (IMPORTANT: Don't use Debug, use Run).

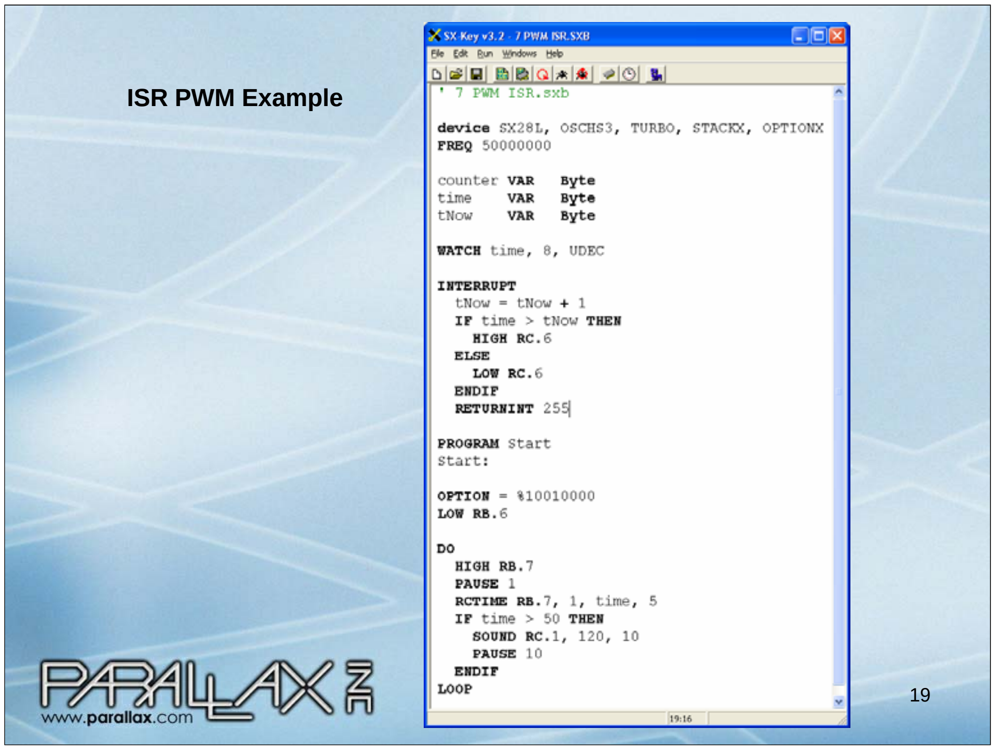
Note that the LED flashes on/off at a constant rate no matter what.

If you want to unplug the cable, and run this independent of the Debugging environment, click Run and select Program.

Take a look at the device directive, it's the one for a 4 MHz resonator, so insert the 4 MHz resonator into the OSC1/OSC2 socket.

Then, you can unplug the SX-Key, and the application will still run because it's depending on its own clock, not the SX-Key's.

IMPORTANT: Always unplug the resonator before using the Debugging environment.



ISR PWM Example

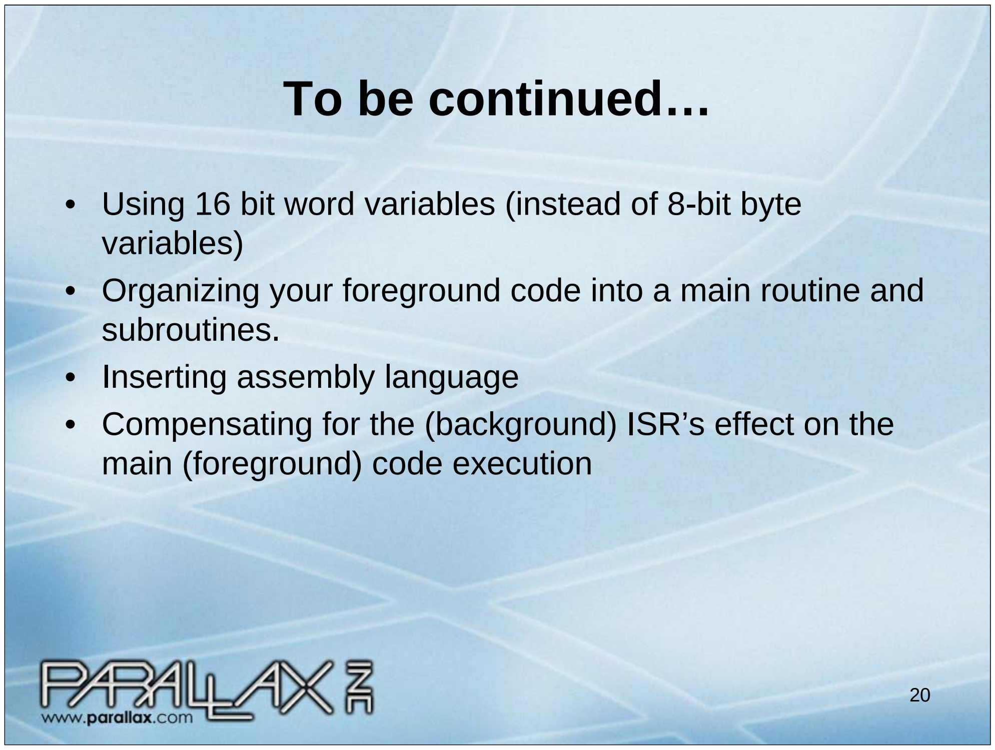
```
' 7 PWM ISR.sxb

device SX28L, OSCHS3, TURBO, STACKX, OPTIONX
FREQ 50000000

counter VAR   Byte
time    VAR   Byte
tNow    VAR   Byte

WATCH time, 8, UDEC

INTERRUPT
  tNow = tNow + 1
  IF time > tNow THEN
    HIGH RC.6
  ELSE
    LOW RC.6
  ENDIF
  RETURNINT 255

PROGRAM Start
Start:

OPTION = %10010000
LOW RB.6

DO
  HIGH RB.7
  PAUSE 1
  RCTIME RB.7, 1, time, 5
  IF time > 50 THEN
    SOUND RC.1, 120, 10
    PAUSE 10
  ENDIF
LOOP
```

19

Here is another example ISR example. This one runs at 50 MHz instead of 4 MHz. If you want to run it independently (disconnected from programming cable), you'll need to use the 50 MHz resonator included with the kit.

This program allows you to adjust the duty cycle to the LED with the potentiometer. To verify this program, simply adjust the potentiometer, and the LED's brightness will adjust accordingly because the RCTIME command's time variable also controls the duty cycle in the INTERRUPT (interrupt service routine) section.

To be continued…



To be continued…

- Using 16 bit word variables (instead of 8-bit byte variables)
- Organizing your foreground code into a main routine and subroutines.
- Inserting assembly language
- Compensating for the (background) ISR’s effect on the main (foreground) code execution

www.parallax.com

20