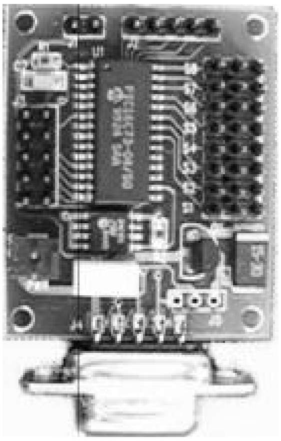
SV203B/C

Servo Motor Controller Board



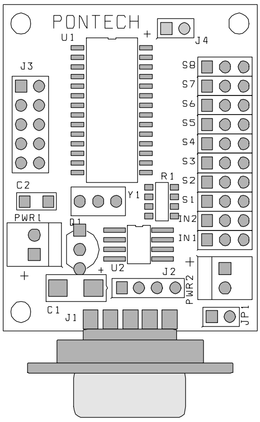
Supplement

Rev. 1.10

PONTECH (714) 985-9286 Fax (714)985-9288
2700 E. Imperial Hwy., Suite - Brea, CA 92821

SV203B/C - Table Of Contents

SV203B/C - Servo Motor Controller Board



This is the supplemental manual for the SV203B/C servo motor controller. The SV203B/C is functionally equivalent to the SV203 with the added feature of being able to run a stand alone BASIC program stored on board the 8k EEPROM (Electronically Erasable Programmable Read Only Memory). Programs of approximately 2000 lines can be written and downloaded to the SV203B/C from an IBM compatible PC. The SV203C has the added ability of receiving infrared signals from a Sony TV remote or equivalent, which can be interpreted by the stand alone BASIC program, sent out a serial port, or both. In addition the SV203C can also transmit Sony TV remote codes. This can be used to control another SV203C or a Sony TV.

SV203B/C - Feature List

SV203B/C

- Controls 1 to 8 servos per board, 8-bit precision.
- Servo port can be reconfigured for digital output to drive on/off devices. Source/Sink 25mA per pin.
- Interface to PC through RS232 port, 2400-19.2k bps (9600bps default).
- User definable board ID number, allowing multiple boards to share the same serial line.
- 5-Channel, 8-bit A/D input port for reading 0 - 5 Volts (control servo positions via Joystick/pot).
- An SPI port for shifting serial data in/out.
- 8k EEPROM for storing standalone routines or special movements (version B & C).
- IR-Receiver feature (version C).
- IR-Transmitter feature (version C).
- Servo Connectors: 3 pin sip, Futaba J-type connectors.
- Power supply: 4.8V to 6.0V.

SV203B/C - Programming Process

In order for the controller board to run stand alone, sequences of commands must be saved into its EEPROM. These commands are in the form of a BASIC program that is edited on a PC. Once a program is written it must be compiled and downloaded with the BASIC compiler (SVBAS.EXE) which outputs a hex object file that can then be downloaded to the SV203B/C.
If there are no errors in the source code, the SVBAS program will automatically download the object code to the board.

If the serial port number is not specified, it will assume the default setting, which is COM1. To select other COM ports see Compiler Options – p.29, or type:

```
        C:> SVBAS <filename> /Px
```
where x is the port number

The program can then be run by typing:
```
        C:> SVBAS /R or
        C:> SVBAS /RUN
```

If you want the program to start running as soon as the board receives power then the Auto-start flag in the EEPROM must be set less than 255, or type the following command:

```
        C:> SVBAS /AUTO
```

Once the on board BASIC program is running, it may be interrupted by typing:

```
        C:> SVBAS /S  or
        C:> SVBAS /STOP
```

SV203B/C - Programming Process

Here are the steps you might take to program the SV203B/C board.

1) First, use a text editor to create the source code. The file should be saved with extension .SV

```
      C:> EDIT TEST.SV
```

the contents of TEST.SV may look something like this

```
      Dim I as Byte       ‘Define one Byte
      Main:               ‘Label
        For I = 1 to 5    ‘Loop for 5 Times
          Servo1 = 10     ‘Move to position 10
          Delay 1000      ‘Delay for 1 sec.
          Servo1 = 100    ‘Move to position 100
          Delay 1000      ‘Delay for 1 sec.
        Next              ‘Loop
      End                 ‘Stop Running
```

2) Connect the serial port of the host PC to the board and supply power to the board. We will use COM2 for the rest of the examples.

3) To compile and download the source to COM2, type:

```
      C:> SVBAS Test /P2
```

If there is an error, go back to step 1 and fix the error. If there are no errors, the compiler will begin to download the program to the SV203B/C board.

To run the program on the SV203B/C connected to COM2, type:

```
      C:> SVBAS /R /P2
```

A servo connected to S1 connector on the SV203B/C will move to position 10, delay for one second, then move to position 100 and delay for another second. The cycle will repeat for a total of five times and stop at position 100 about 10 seconds later.

The SVBAS compiler uses a syntax loosely based on the QBASIC language by Microsoft ®. Because of the memory capacity of the EEPROM and limited power of the processor, not all the commands are implemented.

Naming Conventions
Functions and variables must be unique and begin with an Alpha character (a-z, A-Z), and may be followed by alpha, numeric or underscore characters. Other characters may work but are not supported.
The following names are legal:

```
okay1
okay_1
```

The following name is illegal:

```
1toMany   ‘Starts with a non-alpha char
```

Case Insensitive
The SVBAS compiler is not case sensitive for variables, constants, labels, function names or reserved words. A literal between quotes is, however, case sensitive. The following two code fragments do the same thing:

```
PRINT “Hello World” ‘Fragment one
print “Hello World” ‘Fragment two
```

Comments
Comments can be put on any line. Once a ‘ character is encountered, the compiler treats the rest of the line as a comment.

Note: A literal is any data that is hard-coded to the program.
i.e.
```
A = 5            ‘5 is the literal
PRINT “Hello”  ‘”Hello” is the literal
```

Colon Delimiter (Multiple commands on one line)
More than one command can exist on a line. The maximum line length the SVBAS compiler can handle is 256 characters. Commands are separated with the “:” (colon) character.

```
; : PRINT “World”
```

Supported Operators

=	assignment
+	addition
-	subtraction

Bit-wise (used during an assignment)
AND, OR, XOR, NOT

i.e. `A = B AND 00001111b`

*Note: Numeric literal representation can be decimal or binary

```
i.e. 5                ‘decimal
     00000101b        ‘binary
```

Logical (used in an if statement expression)
=, <, <=, >, >=, <>, AND, OR, NOT

i.e. `If A >= 5 AND B = 3 THEN ‘Code fragment`

Predefined variables
There are several predefined variables in the SVBAS.INC file. These variables are to make programming the built-in functions and ports simpler. A complete listing of SVBAS.INC is in Appendix C.

Labels and Flow Control
Labels must start at the first column of a line and must end with a colon. There can not be any space between the label name and the colon. When jumping to a label with a GOTO or other flow control statement, the colon must not follow the label name.

Compiler Limitations

The compiler only allows 8-bit or 1-bit variables. Constants can be 16 bits but the only command that can use 16-bit constants is DELAY.

The SVBAS compiler does not support local variables in subroutines. All variables are global to all functions.

Subroutines do not support argument passing, use global variables instead.

RAM space on the SV203B/C is limited to 60 bytes and is shared with the CALL and expression evaluation stack.

RAM space on the SV203B/C also takes over most of the RS232 serial buffer used in SV203 non-standalone board. This means the buffer is only 20 characters long and not 80 characters long as in the SV203.

Multiplication and division operators are not implemented.

*Note: Parameters in [] are optional
Parameters in <> are required
| means "or" (one of the possible choices)

ASC

Return a byte value for an ASCII character. This function is useful for transmitting an IR value.

Syntax:
ASC <character>
character is an ASCII character encapsulated by double quotes

Example:

```
DIM ASCII AS BYTE
ASCII = ASC “A”
```

CALL

A statement that temporarily transfers control to a BASIC subroutine. When the subroutine ends with an END SUB or EXIT SUB, program flow will continue at the next line after the CALL statement.

Syntax:
CALL <subName>
subName is the name of the SUB

Example:

```
CALL SlowMove
```

CONST

A non-executable statement that declares a symbolic constant to use in place of numeric values. Symbols declared with a CONST are evaluated at compile time and do not require any RAM.

Syntax:
CONST <constName> = <value>
constName is a name that follows SVBAS naming rules, value is a constant number.

Example:

```
CONST a = 5
```

DIM
A declaration statement that names a variable and allocates storage space for it.

Syntax1:
DIM <variable> [AS BYTE]
variable is a BASIC variable name
all variables are shared for all procedures

Syntax2:
DIM <bitVariable> AS BIT <bitNumber> of <variable>
bitVariable is a BASIC variable name
all bit variables are shared for all procedures
bitNumber is a number between 0 and 7
variable is a declared variable
note: a variable declared as a bit can only be assigned to 0 or 1, False or True

Example:
```
DIM A AS BYTE               ‘Declare one byte
DIM bA2 AS BYTE 2 OF A      ‘Declare one bit
```

DELAY
Built in function that delays n milliseconds.

Syntax:
DELAY <variable | constant | literal>
variable is a BASIC byte variable name
constant and literal are two byte unsigned values (0 - 65535)

Example:
```
DIM A AS BYTE          ‘Declare one byte

A = 100
DELAY A ‘pause for .1 second or

FOR A = 1 to 10 ‘pause for 5 minutes (10 x 30sec)
  DELAY 30000 ‘pause for 30 seconds
NEXT
```

DO...LOOP

A flow control statement that repeats a block of statements while a condition is true or until a condition becomes true.

Syntax 1:

```
DO [{WHILE | UNTIL} expression]
   [statement block]
LOOP
```

Syntax 2:

```
DO
   [statement block]
LOOP [{WHILE | UNTIL} expression]
```

expression is a Boolean expression that will return non-zero (true), or zero (false)
statement block is any number of statements on one or more lines which are to be executed as long as expression is true. WHILE continues to loop while expression is true. UNTIL continues to loop until expression is true.

Example:

```
DO                          ‘Loop forever
   PRINT “ABC”
LOOP                        ‘Goto DO
```

END

A BASIC declaration that ends a program’s execution.

Syntax:

```
 END
```

EPEEK

Read an SV203B/C EEPROM address.

Syntax:
EPEEK <address>
address is the EEPROM address in question, see Appendix B – Memory Maps for more information. (value from 0 to 8191)

Example:

```
PRINT EPEEK(1) ‘Send the power-up position of servo 1 out the serial
               ‘port
```

EPOKE

Modify an EEPROM value.

WARNING: Because of the limited ERASE/WRITE cycles of the EEPROM, excessive writes to the EEPROM can permanently damage the device.

Syntax:
EPOKE <address>, <value>
address is the EEPROM location to be modified (value from 0 to 8191)
value is the value to be stored at address.

Example:

```
EPOKE 1, 200 ‘Set the power position of servo 1 to 200
```

EXIT

A flow control statement that exits a, DO...LOOP, FOR...NEXT loop, or SUB.

Syntax:

```
EXIT {DO | FOR | SUB}
```

Example:

```
DIM a AS byte
FOR a = 1 to 20
  IF a = 10 THEN EXIT FOR
  PRINT a
NEXT
```

FOR...NEXT

A flow control statement that repeats a block of statements a specified number of times.

Syntax:

```
FOR counter = start TO end [STEP increment]
  [statements]
NEXT
```

counter is a numeric variable used as the loop counter
start is the initial value of the counter
end is the final value of the counter
increment is the amount the counter is incremented each time through the loop

Example:

```
DIM a AS byte
FOR a = 50 to 1 STEP -2  'Count down from 50 to 1 in steps on 2
  PRINT a
NEXT
```

SV203B/C - GOTO, IF...ELSE...THEN

GOTO

A flow control statement that branches unconditionally to the specified line.

Syntax:

```
GOTO <label>
```

label is the label of the line to execute next. This line must be in the same procedure or subroutine as the GOTO statement

Example:

```
start:  PRINT “Hello World”
GOTO start
```

IF ...THEN ...ELSE

A flow control statement that allows conditional execution or branching, based on the evaluation of an expression that must be either true or false.

Syntax:

```
  IF expression THEN
     [statement block]
  [ELSEIF expression THEN]
     [statement block]
  . . .
  [ELSE]
     [statement block]
  END IF

  IF expression THEN part
```

expression is a Boolean expression that must return non-zero (true) or zero (false)

statement block consists of any number of statements on one or more lines.

Example:

```
IF a >= 10 THEN
  c = 20
ELSEIF b < 5 THEN
  c = 1
END IF
```

IRSEND

Built in function that generates the Sony Remote 40kHz AM signal based on the value stored in IRreg, to be transmitted via an IR LED. Value can be 0 - 255

Example:

```
IRreg = ASC("F") 'Value for the POWER button
CALL Irsend
```

LET

Assigns an expression to a variable.

Syntax:

[LET] <variable> = <expression | constant | literal>
Note: LET is optional and not required.

Example:

```
LET Servo1 = 128        'Move Servo 1 to position 128
Servo1 = 128            'Comparable to above
```

PEEK

Read a value stored in RAM.

Syntax:

PEEK <address>
address is the RAM address in question, See memory map appendix for more information.

Example:

```
PRINT EPEEK(51) 'Send the current position of servo1 out the serial port
```

POKE

Modify a RAM value.

Syntax:

POKE <address>, <value>
address is the RAM location to be modified
value is the value to be stored at address

Example:

```
POKE 1, 200 'Set the current position of servo 1 to 200
```

PRINT

A device output statement that outputs data to the RS232 serial port.

Syntax:
PRINT [expression-list][{,|;}]
If all arguments are omitted, a blank line is printed.
If expression-list is included, the values of the expressions are printed on the screen.
The expressions in the list may be numeric or string expressions. (String literal must be enclosed in quotation marks.)
The optional characters "," and ";" can be use to send a tab or no return respectively at the end of the string.

Example:
```
PRINT “Hello World!”
PRINT a
```

SPIIN

Built in function that shifts data from the SPI port into SPIreg.

Example:
```
CALL SPIin
PRINT SPIreg
```

SPIOUT

Built in function that shifts data from the SPIreg out the SPI port.

Example:
```
SPIreg = 55
CALL SPIout
```

SUB...END SUB

A procedure statement that marks the beginning and end of a subprogram.

Syntax:

```
 SUB name
    [statements]
    [EXIT SUB]
    [statements]
 END SUB
```

Example:

```
SUB PrintHello
    Print “Hello”
END SUB
```

SV203B/C - Example Programs

One of the most productive ways to learn about a programming language is to look at examples written by others, so here they are…

Moving Servos

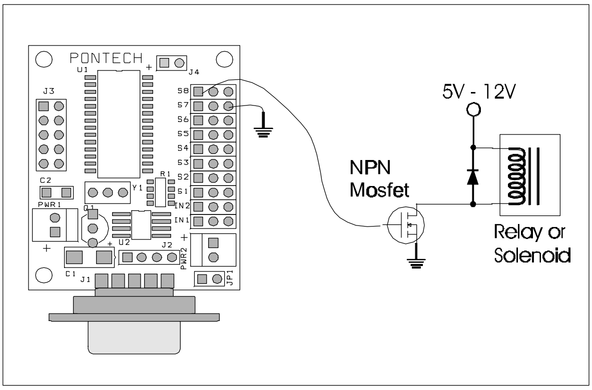
This program will move servo 1 to servo 8 through positions 15 to 215 in steps of 50 with one-second delay between moves. The variables Servo1 through Servo8 are predefined in the file SVBAS.INC. By assigning a value from 0 to 255 to ServoX, you are affecting the PWM for that servo channel, 0 = No PWM, 1 = 0.6ms pulse width, and 255 = 2.4ms pulse width.

```
DIM position AS byte
start:
FOR position = 15 TO 215 STEP 50
   Servo1 = position ‘Servo1 - Servo8 are
   Servo2 = position ‘predefined in the
   Servo3 = position ‘SVBAS.INC file
   Servo4 = position
   Servo5 = position
   Servo6 = position
   Servo7 = position
   Servo8 = position
   DELAY 1000 ‘Delay for 1 second
NEXT
GOTO start
```

Using a Servo pin to control a relay

This program reconfigures Servo8 output to switch a relay on and off every second.

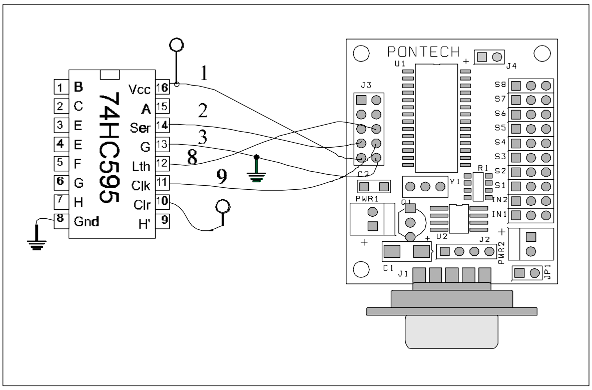
```
‘Port B is the SV203 servo PWM port,
‘RB is defined in SVBAS.INC
DIM relay8 AS bit 7 of RB

Servo8 = 0 ‘Turn off Servo8 PWM
start:
   relay8 = 0  ‘Turn off relay
   DELAY 1000  ‘Delay for 1 second
   relay8 = 1  ‘Turn on relay
   DELAY 1000  ‘Delay for 1 second
GOTO start
```



SPI Output

This program requires a 74HC595 (serial to parallel shift register with output latch). The 74HC595 serial in and serial clock are connected to the SPI port and the latch clock is connected to RA4 of the SV203B/C. The program shifts out an alternating bit patteren and latches the value to the output of the 74HC595.

```
DIM bTRISA5 as Bit 5 of TRISA
DIM bSPIlatch as BIT 5 of RA

BTRISA5 = 0    ‘Set RA.5 as output
bSPIlatch = 1  ‘Set latch clock high
Main:
  SPIreg = 01010101b  ‘Set value to be shifted
  call SPIout         ‘Shift out the data
  bSPIlatch = 0       ‘pulse latch clock
  bSPIlatch = 1
  delay 1000          ‘Pause one second

  SPIreg = 10101010b  ‘Alternate the bit pattern
  call SPIout         ‘Shift out the data
  bSPIlatch = 0       ‘pulse latch clock
  bSPIlatch = 1
  delay 1000          ‘Pause one second
Goto Main
```



Slow Moves

With the use of FOR loops and short delays, a slow move command can be implemented.

```
DIM I as byte : DIM Pos as byte     ‘Pos is the new
Const slowDelay = 5                 ‘position

Sub Slow1
  IF Servo1 > Pos THEN
    FOR I = Servo1 TO Pos           ‘Slowly increase
      Servo1 = I : Delay slowDelay  ‘to Pos
    Next
  ELSE
    FOR I = Servo1 TO Pos Step -1   ‘Slowly decrease
      Servo1 = I : Delay slowDelay  ‘to Pos
    Next
  END IF
END SUB
```

Modifying SV203B/C EEPROM Value

This simple program modifies EEPROM address 1 of the SV203B/C. This address determines the power-up position of servo 1.

```
‘Check if value is already set to avoid excessive
 writes the EEPROM
IF EPEEK(1) <> 230 THEN EPOKE 1, 230
```

Control a PIC port / Modify PIC address

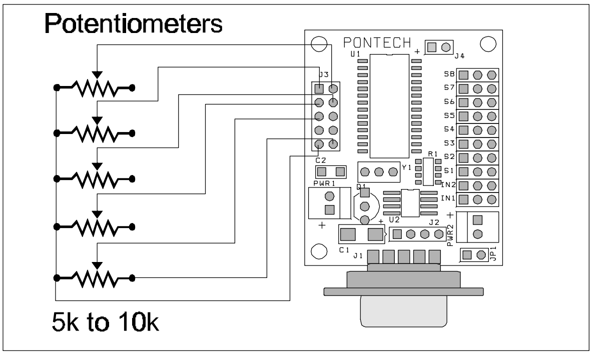
This simple program modifies the contents of internal RAM and demonstrates the equivalence of setting the position of Servo 1 using either RAM address 51 of the SV203B/C or the Servo1 variable defined in the SVBAS.INC file.

```
POKE 51,25 ‘Set Servo1 to position 25
PRINT PEEK(51) ‘Send out current servo position

Servo1 = 25 ‘Identical to above PEEK, POKE example
PRINT Servo1
```

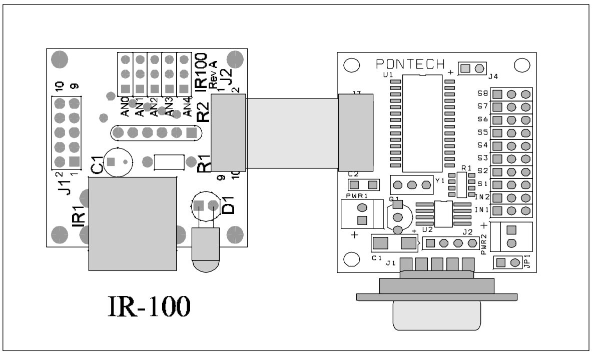
Reading A/D

This program reads A/D (Analog to Digital) channel 1 through channel 5 and then moves Servo1 through Servo5 respectively, to the position of the A/D value. Note that channels 1 to 5 are AD0 to AD4.

```
‘ADRES, ASCON0 and ADCON1 are defined in SVBAS.INC

  ADCON1 = 0 ‘Set all A/D ports to Analog operation
start:
  ADCON0 = 11000001b ‘Select AD0
  bADSTART = 1       ‘Start Conversion
  Servo1 = ADRES     ‘Assign result to Servo1
  ADCON0 = 11001001b ‘Select AD1
  bADSTART = 1       ‘Start Conversion
  Servo2 = ADRES     ‘Assign result to Servo2
  ADCON0 = 11010001b ‘Select AD2
  bADSTART = 1       ‘Start Conversion
  Servo3 = ADRES     ‘Assign result to Servo3
  ADCON0 = 11011001b ‘Select AD3
  bADSTART = 1       ‘Start Conversion
  Servo4 = ADRES     ‘Assign result to Servo4
  ADCON0 = 11101001b ‘Select AD4
  bADSTART = 1       ‘Start Conversion
  Servo5 = ADRES     ‘Assign result to Servo5
GOTO start
END
```



To use the infrared (IR) function, the IR-100 unit must be connected to port C or connector J3 of the SV203C board.



Any IR remote controller made for a Sony TV or any universal remote set to emulate a Sony TV remote will work with the SV203C. The SV203C will pick up most of the buttons that are on the remote. This includes the number keys 0 to 9, Enter, Power, Channel Up, Channel Down, Volume Up, Volume Down, TV/VCR and Muting Buttons.

The board must be programmed to detect the IR signal.

```
bIRenable = 1         ‘Enable IR receiver

DO                    ‘Loop forever
  IF bIRreceive Then  ‘If IR received in Buffer
    PRINT IRreg       ‘Send IR value out serial port
    bIRreceive = 0    ‘Clear flag for next receive
  END IF              ‘End of IF statement
LOOP                  ‘Loop back to DO
```

This program will wait in an idle loop until an IR signal is received. The value received will be sent out the serial port and wait for another IR signal.

SV203B/C - SV203C IR Receiver Functions

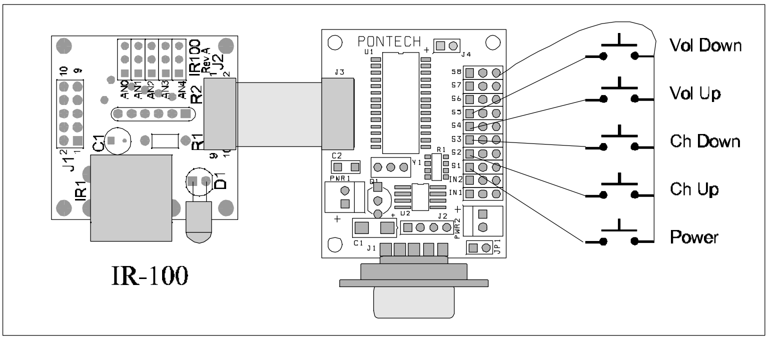
This program detects a number from 1 to 8 from an IR remote to select a servo. Once a number is pressed the channel up/down is used to move the servo in increments of three.

```
DIM SelectServo AS byte
bIRenable = 1          ‘Enable IR receiver

DO                     ‘Loop forever
  IF bIRreceive Then  ‘If IR received in Buffer
    IF IRreg >= ASC(“1”) and IRreg <= ASC(“9”) then
      servoSelect = IRreg     ‘Save Selected Servo
    ELSE
      IF IRreg = ASC(“A”) then Call ChannelUP
      IF IRreg = ASC(“B”) then Call ChannelDOWN
    END IF
    bIRreceive = 0     ‘Clear flag for next IRreceive
  END IF
LOOP                   ‘Loop back to DO

SUB ChannelUP
  IF SelectServo = ASC(“1”) then Servo1 = Servo1 + 3
  IF SelectServo = ASC(“2”) then Servo2 = Servo2 + 3
  IF SelectServo = ASC(“3”) then Servo3 = Servo3 + 3
  IF SelectServo = ASC(“4”) then Servo4 = Servo4 + 3
  IF SelectServo = ASC(“5”) then Servo5 = Servo5 + 3
  IF SelectServo = ASC(“6”) then Servo6 = Servo6 + 3
  IF SelectServo = ASC(“7”) then Servo7 = Servo7 + 3
  IF SelectServo = ASC(“8”) then Servo8 = Servo8 + 3
END SUB

SUB ChannelDOWN
  IF SelectServo = ASC(“1”) then Servo1 = Servo1 - 3
  IF SelectServo = ASC(“2”) then Servo2 = Servo2 - 3
  IF SelectServo = ASC(“3”) then Servo3 = Servo3 - 3
  IF SelectServo = ASC(“4”) then Servo4 = Servo4 - 3
  IF SelectServo = ASC(“5”) then Servo5 = Servo5 - 3
  IF SelectServo = ASC(“6”) then Servo6 = Servo6 - 3
  IF SelectServo = ASC(“7”) then Servo7 = Servo7 - 3
  IF SelectServo = ASC(“8”) then Servo8 = Servo8 - 3
END SUB
```

The SV203C has an IR transmit feature that allows the board to send Sony style signals over an infrared emitting diode connected to pins 5 and 6 of port C.

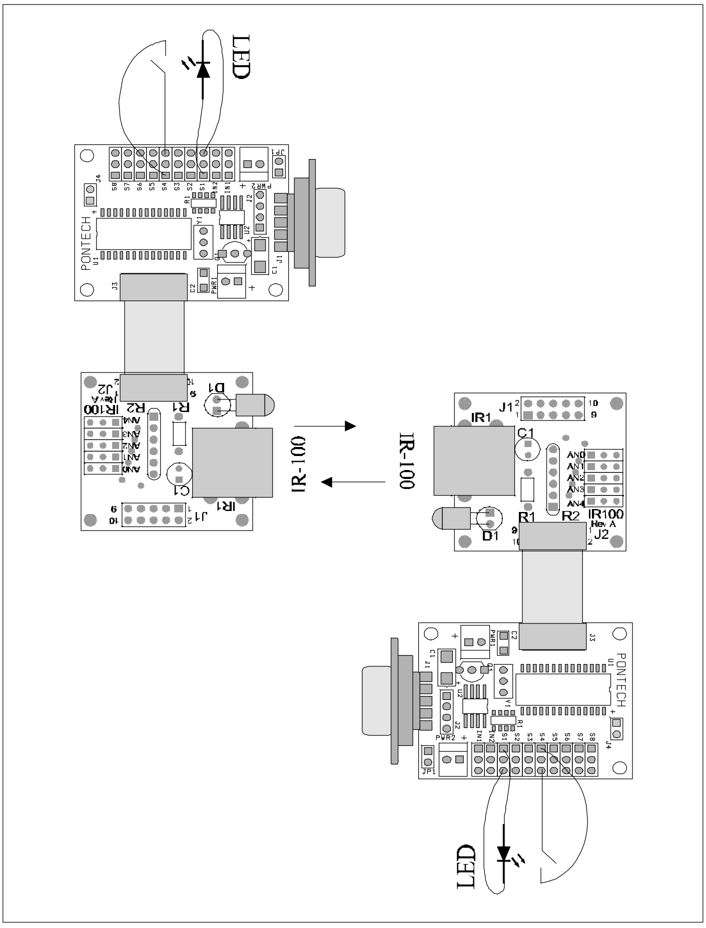
This program reconfigures port B as a digital input. When a button is pressed, an IR signal corresponding to each button will be sent out the IR LED to control a SONY TV or another SV203C board remotely.

```
DIM bPWR   as BIT 0 of RB
DIM bChUp  as BIT 1 of RB
DIM bChDn  as BIT 2 of RB
DIM bVolUp as BIT 3 of RB
DIM bVolDn as BIT 4 of RB
DIM OPTION at 129 ‘Register for port B pullup
DIM bRBPU  as BIT 7 of OPTION

Servo1 = 0 : Servo2 = 0 : Servo3 = 0 : Servo4 = 0
Servo5 = 0 : Servo6 = 0 : Servo7 = 0 : Servo8 = 0
bRBPU = 0              ‘Enable weak Port B pull-up
TRISB = 11111111b      ‘Configure Port B as all inputs
DO
  if not bPWR   then IRreg = ASC(“F”) : Call IRsend
  if not bChUp  then IRreg = ASC(“A”) : Call IRsend
  if not bChDn  then IRreg = ASC(“B”) : Call IRsend
  if not bVolUp then IRreg = ASC(“C”) : Call IRsend
  if not bVolDn then IRreg = ASC(“D”) : Call IRsend
  Delay 5  ‘Small delay for debounce
LOOP                   ‘Loop back to DO
```



SV203B/C – IR Bi-directional Functions

The following program allows for bi-directional infrared communications between two SV203C boards. The LED (red light) stays on and the infrared signals are transmitted continuously as long as the button is pressed. When it is not pressed, the program is listening for an infrared signal from another SV203C. When the button of the second SV203C running this program is pressed, the red light of the first board will continuously flash.



```
DIM bRED_LED AS BIT 0 of RB
DIM bBUTTON1 AS BIT 3 of RB
DIM OPTION at 129
DIM bRBPU as BIT 7 of OPTION

'Added so the IR rx can be reset
'DIM CCP2CON at 29
'DIM bRC1 as BIT 1 of TRISC

DIM i AS byte

' Turn off servo PWM
Servo1 = 0 : Servo2 = 0 : Servo3 = 0 : Servo4 = 0
Servo5 = 0 : Servo6 = 0 : Servo7 = 0 : Servo8 = 0

'Enable weak Port B pull-up
bRBPU = 0
'RB data direction
TRISB = 00011000b

'Enable IR receiver
bIRenable = 1

start:
   'flashes LED when it is first turned on
   call flash
   call flash

test:
    'if button 1 is pushed
    if not bBUTTON1 then 'if bBUTTON1 = 0
        'Transmit IR start signal
        IRreg = ASC("1")
        call IRsend
        'set the board back to receive mode
        call IRStopSend
        'turn on the red light
        bRED_LED = 1
    else
        'if button1 is not pushed then keep light off
        bRED_LED = 0;
    end if
    'if receive from remote controller
```

```
 if bIRreceive then
        ' do somethin with IFreg
        'As long as "1" on controller is pushed,
        'keep flashing
        if IRreg = ASC(“1”) then
           call flash
        end if
        bIRreceive = 0;
    end if

goto test

SUB flash
     bRED_LED = 1;
     delay 100
     bRED_LED = 0;
     delay 100
end SUB

'Once a value is transmitted, the board will
'stay in transmit-only state
'this subroutine will return board to its
'receive mode
SUB IRstopSend
    'Change from PWM mode to Capture mode
    CCP2CON = 4
    'Set the IR RX pin of port C to input
    bTRISC1 = 1
end SUB
```

SV203B/C - SVBAS Compiler Options

The BASIC compiler converts QBASIC like code into a PONTECH assembly. If there are no errors, it will assemble the code and download it to the board connected to the serial port.

```
SVBAS <filename> <options>
```

Command Line Options

```
/R or /RUN
   Start program currently in EEPROM running.

/S or /STOP
   Stop running program.

/P1  /P2  /P3  /P4      (default /P1 → COM1)
   Select COM port SV203B/C is connected to

/AUTO
   Turn on Auto-Run (Start running from power-up)

/NOAUTO
   Turn off Auto-Run (Don’t run from power-up)

/C
   Compile only (Generates assembly code)

/A
   Assemble only (Generates hex object code)

/D
   Download only (Downloads hex object code)

/L
   Generate listing file of Assembly and Hex object.
```

RAM Memory Map:

Address (m)	Description	Name
5	Port A	RA
6	Port B	RB
7	Port C	RC
30	A/D Result	ADRES
31	A/D Configuration 0	ADCON0
51	Servo 1 Position	SERVO1
52	Servo 2 Position	SERVO2
53	Servo 3 Position	SERVO3
54	Servo 4 Position	SERVO4
51	Servo 5 Position	SERVO5
52	Servo 6 Position	SERVO6
53	Servo 7 Position	SERVO7
54	Servo 8 Position	SERVO8
60	SPI Configuration	SPICON
61	SPI Register	SPIREG
62	IR Configuration	IRCON
63	IR Register	IRREG
133	Port A Direction	TRISA
134	Port C Direction	TRISB
135	Port C Direction	TRISC
159	A/D Configuration 1	ADCON1

Note: All other RAM locations not listed are used by the system and should not be used.

EEPROM Memory Map:

Address (m)	Usage	Factory default	Note
0	Board ID #	1	
1	Initial Servo #1 Value	128	0 = off or Digital
2	Initial Servo #2 Value	128	0 = off or Digital
3	Initial Servo #3 Value	128	0 = off or Digital
4	Initial Servo #4 Value	128	0 = off or Digital
5	Initial Servo #5 Value	128	0 = off or Digital
6	Initial Servo #6 Value	128	0 = off or Digital
7	Initial Servo #7 Value	128	0 = off or Digital
8	Initial Servo #8 Value	128	0 = off or Digital

9	Baud Rate	50 (9600 baud)	25 (19200 baud)
			100(4800 baud)
			200(2400 baud)

10	Pre Enable Flag	1	1=Yes 0=No
11	Shift Configuration Register	0	MSB, valid on Rising, 8-bit
12-14	Reserved by PONTECH	Unknown	Unused
15	Auto-run at Power-Up	Unchanged	0-254 = auto-run 255 = no auto-run
16-63	Reserved by PONTECH	Unknown	Unused
64-8191	Program Space	Unknown	User Program

IR Button Table:

Button	ASC()	Decimal
1 - 9	“1” - “9”	49 - 57
0	“:”	58
Enter	“<”	60
Channel Up	“A”	65
Channel Down	“B”	66
Volume Up	“C”	67
Volume Down	“D”	68
Muting	“E”	69
Power	“F”	70
TV/VCR	“V”	86

ADCON0

R/W	R/W	R/W	R/W	R/W	R/W	U	R/W
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	–	ADON
bit7							bit0

Register:	ADCON0	W:	Writable bit
Address:	1Fh	R:	Readable bit
POR value:	00h	U:	Unimplemented, read as '0'

ADON: A/D on bit
1 = A/D converter module is operating.
0 = A/D converter module is shut off and consumes no operating current.

Reserved.

GO/$\overline{\text{DONE}}$: A/D conversion status bit.
If ADON = 1
1 = A/D conversion is progress. Setting this bit starts an A/D conversion.
0 = A/D conversion not in progress / completed. This bit is automatically cleared by hardware when the A/D conversion is completed.
If ADON = 0
This bit is forced to zero.

CHS <2:0>: Analog channel select.
000 = channel 0 (RA0/AN0)
001 = channel 1 (RA1/AN1)
010 = channel 2 (RA2/AN2)
011 = channel 3 (RA3/AN3/VREF)
100 = channel 4 (RA5/AN4)
101 = channel 5 (RE0/AN5)†
110 = channel 6 (RE1/AN6 †
111 = channel 7 (RE2/AN7) †

ADCS<1:0>: A/D conversion clock select.
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock is derived from an RC oscillator)

† Not available in PIC16C73

ADCON1

U	U	U	U	U	R/W	R/W	R/W
—	—	—	—	—	PCFG2	PCFG1	PCFG0
bit7							bit0

Register:	ADCON1	W:	Writable
Address:	9Fh	R:	Readable
POR value:	00h	U:	Unimplemented, read as '0'

PCFG<2:0>: A/D port configuration bits.
These bits configure the analog port pins to the various modes of operation.

PCFG<2:0>	RA0	RA1	RA2	RA5	RA3	RE0	RE1	RE2	Ref
000	A	A	A	A	A	A	A	A	VDD
001	A	A	A	A	VREF	A	A	A	RA3
010	A	A	A	A	A	D	D	D	VDD
011	A	A	A	A	VREF	D	D	D	RA3
100	A	A	D	D	A	D	D	D	VDD
101	A	A	D	D	VREF	D	D	D	RA3
11X	D	D	D	D	D	D	D	D	–

A = Analog input

D = Digital input/output depending on corresponding TRIS bit

SV203B/C - Appendix C SVBAS.INC

```
'***** Include file for SV203B/C *****
DIM     RA      AT      05h     'Port A
DIM     RB      AT      06h     'Port B
DIM     RC      AT      07h     'Port C

DIM     TRISA   AT      85h     'Port A Direction
DIM     TRISB   AT      86h     'Port B Direction
DIM     TRISC   AT      87h     'Port C Direction

DIM     ADRES   AT      1Eh     'A/D register
DIM     ADCON0  AT      1Fh     'A/D configuration 0
DIM     ADCON1  AT      9Fh     'A/D configuration 1
DIM     bADSTART as BIT 2 of ADCON0 'A/D start flag

'Interrupt Enable for IR feature
DIM     PIE2    AT      8Dh

DIM     STAT    AT      32h     'Status
DIM     bCARRY  as BIT 0 of STAT
DIM     bZERO   as BIT 2 of STAT

DIM     STACK   AT      27h     'Stack Pointer

DIM     SERVO1  AT      51      'Servo 1 position
DIM     SERVO2  AT      52      'Servo 2 position
DIM     SERVO3  AT      53      'Servo 3 position
DIM     SERVO4  AT      54      'Servo 4 position
DIM     SERVO5  AT      55      'Servo 5 position
DIM     SERVO6  AT      56      'Servo 6 position
DIM     SERVO7  AT      57      'Servo 7 position
DIM     SERVO8  AT      58      'Servo 8 position

DIM     SPICON  AT      60      'SPI configuration
DIM     SPIREG  AT      61      'SPI register

DIM     IRCON   AT      62      'IR configuration
DIM     IRREG   AT      63      'IR register

'Enable IR Flag
DIM     bIRenable  as  BIT 0 of PIE2
'IR valid and received Flag
DIM     bIRreceive as  BIT 5 of IRCON
```

SV203B/C - Appendix D ASCII Table

000 (nul)	016 (dle)	032 (sp)	048 **0**	064 **@**	080 **P**	096 `	112 **p**
001 (soh)	017 (dc1)	033 **!**	049 **1**	065 **A**	081 **Q**	097 **a**	113 **q**
002 (stx)	018 (dc2)	034 **"**	050 **2**	066 **B**	082 **R**	098 **b**	114 **r**
003 (etx)	019 (dc3)	035 **#**	051 **3**	067 **C**	083 **S**	099 **c**	115 **s**
004 (eot)	020 (dc4)	036 **$**	052 **4**	068 **D**	084 **T**	100 **d**	116 **t**
005 (enq)	021 (nak)	037 **%**	053 **5**	069 **E**	085 **U**	101 **e**	117 **u**
006 (ack)	022 (syn)	038 **&**	054 **6**	070 **F**	086 **V**	102 **f**	118 **v**
007 (bel)	023 (etb)	039 **'**	055 **7**	071 **G**	087 **W**	103 **g**	119 **w**
008 (bs)	024 (can)	040 **(**	056 **8**	072 **H**	088 **X**	104 **h**	120 **x**
009 (tab)	025 (em)	041 **)**	057 **9**	073 **I**	089 **Y**	105 **I**	121 **y**
010 (lf)	026 (eof)	042 *****	058 **:**	074 **J**	090 **Z**	106 **j**	122 **z**
011 (vt)	027 (esc)	043 +	059 **;**	075 **K**	091 **[**	107 **k**	123 **{**
012 (np)	028 (fs)	044 **,**	060 <	076 **L**	092 \	108 **l**	124 |
013 (cr)	029 (gs)	045 **-**	061 **=**	077 **M**	093 **]**	109 **m**	125 **}**
014 (so)	030 (rs)	046 **.**	062 >	078 **N**	094 ^	110 **n**	126 ~
015 (si)	031 (us)	047 /	063 **?**	079 **O**	095 **_**	111 **o**	127 (del)

Warranty

Pontech warrants its products against defects in materials and workmanship for a period of 90 days.

If you discover a defect, Pontech will, at its option, repair, replace, or refund the purchase price. Simply return the product with a description of the problem and a copy of your invoice (if you do not have your invoice, please include your name and telephone number).

The warranty does no apply if product has been damaged by accident, abuse, or misuse.

Copyright and Trademarks

Disclaimer of Liability

Pontech is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of property, and any costs or recovering, reprogramming, or reproducing and data stored in or used with Pontech products.

PONTECH

2700 E. Imperial Hwy., Suite N – Brea, CA 92821
Phone: (714) 985-9286 Fax: (714) 985-9288