Tiny Logo for the S2

Michael Daumling

1 Introduction

The Parallax S2 robot is a rolling box of robotics features. Apart from fine-tuned movements, it comes with two ultrasound obstacle sensors, three light sensors, and two sensors for line tracking. Four programmable LEDs, a sound generator, and a microphone are welcome add-ons. A serial cable connects it to the PC for programming purposes.

This Tiny Logo implementation lets users program the S2 in the Logo language. For such a small device, Tiny Logo is surprisingly complete. You can define variables and procedures; even limited string handling is possible. It lacks lists and list handling, however, apart from run lists that are parts of a program. All you need to program the S2 is a serial terminal program.

2 Starting up

To program the S2 robot, a Windows PC is needed. The S2 comes with an optional serial-to-USB cable. When inserting the cable into the PC, Windows installs a driver that creates a serial port which the Parallax software can use to talk to the robot.

To program the S2 robot with the Tiny Logo EEPROM, go to the Parallax web site and download the Propellent tool from http://www.parallax.com/Portals/0/Downloads/sw/propeller/Propellent-v1.3-(R2).zip. The file is a packed archive that Windows can unpack into a directory of your choice.

After unpacking the archive, open a DOS command window and navigate to the directory that contains the Propellent.exe file. Connect the S2 to the PC with the Parallax serial-to-USB cable, turn the S2 on, and run the following command:

```
propellent /eeprom {directory where the EEPROM image is}\tinylogo.eeprom
```

This command stores the Tiny Logo image into the S2's permanent memory. If all goes well, the S2 responds with a short chime, and Tiny Logo is ready to go.

To talk to Tiny Logo, you will need a serial terminal program. There are countless versions available on the Internet. Parallax offers a free download, the Parallax Serial Terminal, available at http://www.parallax.com/Portals/0/Downloads/sw/propeller/Parallax-Serial-Terminal.exe.

To find out which port the robot is connected to, use the following command:

```
propellent /id
```

The program displays the name of the serial port, e.g. "COM3". The port settings are 115.200 baud, 8 data bits, 1 stop bit, and no parity.

3 First Steps

After turning on the S2, the robot waits for the user to press the ENTER key, and then displays its prompt:

```
Tiny Logo V1.0
>
```

The robot is now ready to run Logo commands. Try to enter

```
LIGHTS 9 9 9
```

The command causes the three status LEDs to blink green.

Tiny Logo is a limited version of Logo with the following limitations:

- There are no values TRUE or FALSE. Every non-zero value is considered to be TRUE.
- The numeric range is from -83.886 to +83.886, with two decimal places.
- A name can either be a procedure or a value, not both; property lists are unsupported.
- Strings are supported, but apart from setting, reading, comparing, and printing strings, Tiny Logo does not offer any string manipulation commands. There is no automatic conversion between strings and numbers. Thus, the value “123 is different form the number 123. String constants are limited to 255 characters.
- The MAKE command only accepts a quoted name as its first input.
  The TO command is supported, but formal inputs are limited to 15, and there are no optional or rest inputs.
- There is rudimentary support for lists. A command like S.LIGHT can return a list of up to three element, each elements having a value between 0 and 254 (255 is used to indicate a missing element). Actually, the list is stored internally as a 32 bit value. The ITEM command returns the first, second, or third element of a list. Runlists as part of a procedure are supported; see the IF or WHAILE commands for an example. Parentheses are supported to feed more or less than the default number of inputs to a procedure or command.

Tiny Logo recognizes the following special characters:

Ctrl-C: Interrupts a running program and stops execution.

Ctrl-D: Resets Logo and enters command mode.

Ctrl-E: Disconnects Logo.

Ctrl-H or Delete: Deletes the last character.

Enter: executes the line.

4 List of commands

4.1 Math Commands

LSH :value :bits	Converts both inputs to integers, and shifts the first input left by the number of bits that the second input says. If the second input is negative, shifts right instead.
AND :a :b	Converts both inputs to integers, and outputs a bitwise AND on both inputs.
OR :a :b	Converts both inputs to integers, and outputs a bitwise OR on both inputs.

XOR :a :b	Converts both inputs to integers, and outputs a bitwise XOR on both inputs.
NOT :value	Converts its input to integer, and output the inverse of its bits.
LOGAND :a :b	Outputs -1 if both inputs are non-zero.
LOGOR :a :b	Outputs -1 if any input is non-zero.
LOGNOT :value	Outputs -1 if the input is zero, or 0 if it is non-zero.
SIN :degrees	Outputs the sine of its input in degrees.
COS :degrees	Outputs the cosine of its input in degrees.
SIN :degrees	Outputs the sine of its input in degrees.
SQRT :value	Outputs the square root of its input. A negative value is a runtime error.

4.2 Math Operators

Math operators take two numbers. The result must not exceed the range of -83.886 to +83.886, or Tiny Logo reports a runtime error. The multiply, divide, and remainder operators take precedence over all other operators. Addition and subtraction take precedence of comparison operators. The operators can either be used as infix or as prefix operators. Thus, + 2 3 is the same as 2 + 3.

Comparison operators can also compare two strings.

*	Multiplies two numbers; a divisor of zero is a runtime error.
/	Divides two numbers; a divisor of zero is a runtime error.
%	Calculates the remainder of two numbers; a divisor of zero is a runtime error.
+	Adds two numbers.
-	Subtracts two numbers.
=	Compares two inputs; outputs -1 if both inputs have the same type and are equal, 0 otherwise.
<>	Compares two inputs; outputs -1 if both inputs have the same type and are not equal, 0 otherwise.
<	Compares two inputs; outputs -1 if both inputs have the same type, and the first input is less than the second input, 0 otherwise.
<=	Compares two inputs; outputs -1 if both inputs have the same type, and the first input is less than or equal to the second input, 0 otherwise.
>	Compares two inputs; outputs -1 if both inputs have the same type, and the first input is greater than the second input, 0 otherwise.
>=	Compares two inputs; outputs -1 if both inputs have the same type, and the first input is greater than or equal to the second input, 0 otherwise.

4.3 Workspace Commands

MAKE "name value	Stores the second input in the name given as the first input. MAKE only accepts a quoted name as its first input.
THING value	Outputs the value of the name given as its input.
PO "name	Prints a name; this is either the value that is stored, or the compiled representation of a procedure.
POPRS	Prints all built-in primitives.
POPS	Prints all procedures.
PONS	Prints all names that contain values.
ER name	Erases a procedure or a name.
ERALL	Erases everything.

4.4 Procedure Commands

TO name inputs	Defines a procedure. There can be a maximum of 15 formal inputs, and inputs must be names with a colon; lists (optional or rest inputs) are unsupported. The prompt changes to the name of the procedure. A single END ends the procedure definition.
END	Ends a procedure definition.
STOP	Ends the execution of a procedure and returns control to the calling procedure, or to toplevel.
OP value	Ends the execution of a procedure and returns control to the calling procedure, or to toplevel. The input to OP becomes the output of the procedure.
TOPLEVEL	Ends the execution of a procedure and returns control to toplevel.

IF condition [then]
IF cond [then][else]
	IF checks its first input. If the input is non-zero, IF runs the run-list given as the second input. If the input is zero, IF does nothing unless a second list is supplied as input; in that case, IF runs the second run-list.

REPEAT times [list]	Runs the second input, which must be a run-list for the number of times given as its first input.
WHILE [cond][run-list]	While runs its first input, which must be a run-list. If the result of running the list is non-zero, WHILE runs the second input, which also must be a run-list. WHILE repeat this process until the execution of the first input returns a value of zero, or no value at all.

4.5 List commands

ITEM n list	Report the nth item of a list. The list cannot be a literal list; it must be a list returned by a command such as S.LIGHTS. n is either 1, 2, or 3.

4.6 Debugging

PR value (PR value value …)	Prints the given values.
TRACE value	Turns on tracing. The input to TRACE is a value from 0 to 7, and can be one or more of the following: 1 – Tiny Logo prints a message when it enters or exits a procedure. 2 – Tiny Logo prints each line number as it is executed. 3- Tiny Logo prints each opcode as it is executed.
BT	Prints a list of active procedures.
MEM	Tiny Logo prints three values: the number of bytes left for procedure storage, for data like strings, and the number of free names left that you can define.

4.7 S2 Commands

These commands provide access to the features of the S2 robot.

LT degrees	Turns the S1 left by the given number of degrees.
RT degrees	Turns the S1 right by the given number of degrees.
FD distance	Moves the S2 forward by the given number of millimeters.
BK distance	Moves the S2 back by the given number of millimeters.
ARC degrees distance	Move the S2 by the given distance in millimeters, but make it move in an arc. The S2 will stop after the distance and point to the new direction, which is the second input in degrees. If the degrees are negative, it turns to the left; if positive, it turns to the right.
SETSPEED speed	Sets the speed of the S2's movements. The input is a value between 0.1 and 1, where 1 is full speed. Initially, the S2 moves at half speed (0.5).
MOTORS left right (MOROTS lt rt dur)	Turns on the motors of the S2 so it moves forward. The command returns immediately, and the procedure can check the S2's sensors while the S2 is moving. MOTORS has two default inputs, which are the values for the left and right motor. These values may vary between 255 (full power back) and 255 (full power forward). If enclosed in parentheses, a third input is the duration in milliseconds (max 10000). To turn the motors off, use MOTORS 0 0.
S.LIGHT	Outputs the value of the S2's light sensors. The output is a list of three values value between 0 and 254; the more light the sensor receives, the higher the value is.
S.OBSTACLE	Checks the ultrasonic sensors of the S2, and returns a list of two elements. The first element is the left sensor value, and the second element is the right sensor value. The values are 0 or 1.
S.LINE	Checks the line tracing sensors of the S2, and returns a list of two elements. The first element is the left sensor value, and the second element is the right sensor value. The values are 0 or 1.

MOVING?	Outputs a non-zero value if the motors are on, and the S2 is moving.
STALLED?	Outputs a non-zero value if the motors are on, but the S2 is not moving.
TIMER (TIMER value)	Control a timer. The command outputs the current value of a timer in milliseconds. When the timer reaches 83.886, it stops counting. If enclosed in parentheses, a starting value may be added to make the timer start from thet value. (TIMER 0), for example, resets the timer to zero.
BEEP	Issue a beep.
NOTES duration f1 f2	Plays up to two tones. The duration is given in milliseconds. The other two inputs are the frequency of the note, which is a value between 0 and 10,000 Hertz. The value 0 means not to play a note. The command NOTE 500 440 880, for example, plays two A's, which are one octave apart, for the duration of 500 milliseconds.
LIGHTS left mid right	Controls the three status LEDs. Logo does not control the blue power LED. The three inputs control the values of the left, the middle, and the right LED. The following values are possible: 0 – off 1 – red 2 – orange 3 – yellow 4 – chartreuse 5 – green 6 – dim red 7 – dim green 8 – blinking red 9 – blinking green 10 – blinking alternate red and green
COUNTDOWN secs	Issues a countdown for the given number of seconds. The S2 turns on the three status LEDs to red, and beeps every second. The closer the countdown nears the end, the higher the beep will be, and finally, the three LEDS are turned off one after one. When the countdown is over, the S2 chimes. This command is very handy if the S2 is programmed, and is about to be put on the floor or on a table; it provides enough time to put the S2 into whatever starting position before the program executes.
BATT	Outputs the battery level in volts.

4.8 Additional commands

These commands are intended for internal debugging, or to download precompiled programs to the Tiny Logo interpreter.

.PROMPT on	Selects verbose mode. If the input value is 1, the usual verbose mode applies, where the S2 prompts for user input, and displays user-readable messages. If the input is 0, the S2 does not display any prompt, and messages and results are printed in a condensed mode that is better suitable for parsing by a computer.
.DOWNLD bytes	Download a precompiled program into the S2. The input is the number of

	bytes to download. The S2 responds with a special message, requesting for a chunk of bytes. See below for a detailed description.
.MAP	Prints the internal representation of the global map for debugging purposes.
.HEAP which	Dumps the contents of a heap to the console. If the input is 0, Tiny Logo dumps the data heap; if nonzero, Tiny Logo dumps the procedure heap.
(.DUMP) .DUMP "P .DUMP "G .DUMP "D .DUMP address (.DUMP address len) (.DUMP addr len fmt)	Dumps memory. With no inputs, .DUMP continues the dump at the last address where it left off. The first input is either the address, or one of "D (data heap), "P (program heap), or "G (global heap). The optional second input is the number of bytes to dump, and the optional third input is one of the values 1, 2, or 4, for byte, word, or double word format.

5 Internal documentation

This chapter deals with the internals of the Tiny Logo interpreter. It describes the format of the opcodes, and the download process.

5.1 Format of values

All values are 32-bit entities. The top eight bits are flags, while the lower 24 bits are the value. The top bits are as follows:

0x80000000	The value is a number between -83,886.07 and +83,886.07.
0x40000000	The value is a string; the lower 16 bits contain the address or the handle of the Pascal-style string.
0x20000000	The value is a list of up to three elements. Every element has a value between 0 and 254, where the first element is in bits 17 to 23, the 2nd element is in bits 8 to 15, and the third value is in bits 0 to 7. The value 255 is reserved as "no value", so list with less than three elements can exist. List values are only return values from primitive calls.
0x10000000	The value is a procedure. The lower 16 bits contain the handle of the procedure.
0x08000000	The value is a primitive. The lower 16 bits contain the address of the subroutine that implements the primitive.
0x04000000	This bit is used together with the bit 0x40000000 (the string bit). If set, the lower 16 bits do not contain the address, but the heap handle of the string.
0x02000000	Do not list this procedure in the POPRS command.

5.2 Heap handles

The procedure and data heaps implement a simple heap with pointer-to-pointer handles. A heap handle is the address of a word that contains the actual physical address. This makes it possible to compact the heap when a heap element is deleted. Because RAM is very limited, a heap never contains any gaps; Tiny Logo compacts the heap every time an element is deleted.

5.3 Maps

Tiny Logo uses two maps. The biggest one is the global data map that contains all primitives, procedures, and values. Tiny Logo creates a temporary second map during the execution of the TO command that contains the names of the inputs to a procedure. A map is an unbalanced binary tree.

5.4 Program execution

Once the user has entered a line, Tiny Logo compiles the line into opcodes and executes these opcodes by calling the routine eval(). It that routine returns a value, Tiny Logo displays that value.

If Tiny Logo finds a procedure name or a primitive opcode, it opens a procedure frame. This frame contains the following data:

WORD	pointer to last frame
WORD	Pointer to last user-defined procedure frame
WORD	Address or procedure name (0 for primitives)
WORD	Handle of the heap element containing the procedure (0 for primitives)
BYTE	opcode of a primitive (0 for procedures)
BYTE	stack level
BYTE	number of default inputs
BYTE	a flag, set to nonzero to stop a procedure
DWORD …	the inputs the procedure follow; the frame grows as inputs are added

Tiny Logo the determines the number of times to call eval() and to add the resulting value as input to the frame. If the opcode is the first opcode inside a list in parentheses, it, continues to add inputs until the closing-parenthesis opcode (0x87) is found. If the opcode is not encloses in a parenthesized list, Tiny Logo uses the default number of inputs that follow the opcode to call eval() the respective number of times. After Logo has pushed all inputs, it activates the frame and executes the procedure or primitive. If the execution of the frame returns a value, it either stores the value as input (if there is a frame to store inputs into), or it displays the value on the console.

5.5 Opcodes

There are three groups of opcodes:

- Integers: The integer values 0 to 127 are directly encoded as a single byte.
- Special opcodes: These opcodes occupy the range from 0x80 to 0x8F.
- Primitives: The remaining values from 0x90 to 0xFF are primitives.

5.5.1 Special opcodes

0x80 value	Values from 0.01 to 2.55 are encoded in a single byte following the opcode 0x80. Thus, the opcode 0x80 0xC0 would represent the number 1.92.
0x81 lo-byte hi-byte	Values from 2.56 to 655.35 are encoded in two bytes byte following the opcode 0x81 in big-endian format.
0x82 lo mid hi	Values from 655.36 to 83,886.07 are encoded in three bytes byte following the opcode 0x82 in big-endian format.
0x83 len data…	0x83 prefixes a string of 0 to 255 characters in size. The string length follows, followed by the string bytes.

0x84 len data…	User-defined procedure names are just like strings; the opcode 0x84 is followed by the size and the string bytes. This opcode opens a procedure frame.
0x85 len data…	Global names are just like strings; the opcode 0x85 is followed by the size and the string bytes. The opcode pushes the value of the variable into the currently open frame.
0x86 lo-byte hi-byte	The opcode 0x86 signals the beginning of a list in parentheses. The size of all opcodes within that list follows as a two-byte integer in big endian format. As an example, the command (PR 4 5) is encoded as 0x86 0x05 0x00 0x96 0x01 0x04 0x05 0x87.
0x87	End of a parenthesized list.
0x88 lo-byte hi-byte	The opcode 0x86 signals the beginning of a runlist in square brackets. The size of all opcodes within that list follows as a two-byte integer in big endian format. As an example, the command [PR 4 5] is encoded as 0x88 0x05 0x00 0x96 0x01 0x04 0x05 0x88.
0x89	End of a runlist in square brackets.
0x8A line	The opcode 0x8a signals the beginning of a procedure line. The line number (1-255) follows as a single byte. If the procedure is larger than 255 lines, the line number stays at 255.
0x8B	Recursive call. This opcode prepares the current frame for a recursive call.
0x8C index	The opcode 0x8A fetches the value of the currently executing frame. The index is the nth value of the current frame.
0x8D index	The opcode 0x8B evaluates the next opcode(s) and stores the result into the nth value of the current frame.
0x8E	Define a procedure in download mode. The first byte following the opcode is the number of defined arguments. The second byte is the length of the procedure name, followed by the procedure name. After that, a two-byte value (in big endian format) follows, which is the number of compiled bytes that follow to form the procedure.

5.5.2 Primitives

A primitive is an opcode, followed by the number of default arguments. Each primitive opcode opens a call frame. If the primitive is not at the beginning of a list in parentheses, the opcode evaluates the following opcodes for the given number of times, and stores the results as inputs for the given procedure. As soon as the number of default arguments has been reached, the frame is executed.

0x90	TO
0x91	END
0x92	STOP
0x93	OP
0x94	MEM

0x95	.DOWNLD
0x96	PR
0x97	TRACE
0x98	BT
0x99	.PROMPT
0x9A	*
0x9B	/
0x9C	%
0x9D	+
0x9E	-
0x9F	=
0xA0	<>
0xA1	<
0xA2	>
0xA3	<=
0xA4	>=
0xA5	LSH
0xA6	LOGAND
0xA7	LOGOR
0xA8	LOGXOR
0xA9	LOGNOT
0xAA	AND
0xAB	OR
0xAC	NOT
0xAD	SIN
0xAE	COS
0xAF	SQRT
0xB0	MAKE
0xB1	THING
0xB2	PO
0xB3	POALL
0xB4	PONS

0xB5	POPS
0xB6	POPRS
0xB7	ER
0xB8	ERALL
0xB9	IF
0xBA	REPEAT
0xBB	WHILE
0xBC	TOPLEVEL
0xBD	ITEM
0xBE…0xCF	Free to use
0xD0	S.LIGHT
0xD1	S.LINE
0xD2	S.OBSTACLE
0xD3	SETSPEED
0xD4	FD
0xD5	BK
0xD6	LT
0xD7	RT
0xD8	ARC
0xD9	TIMER
0xDA	STALLED?
0xDB	MOVING?
0xDC	BEEP
0xDD	NOTES
0xDE	LIGHTS
0xDF	COUNTDN
0xE0	MOTORS
0xE1	BATT
0xE2…0xEF	Free to use
0xF0	.DUMP
0xF1	.MAP
0xF2	.HEAP

0xF3…0xFF	Free to use

5.6 Downloading and condensed mode

Tiny Logo accepts precompiled Logo programs for direct execution. The .DOWNLD opcode takes as input the number of bytes to store. Tiny Logo enters a loop, where is asks for up to 16 bytes to store with the string Ctrl-C plus "RDY nn" plus CR, where nn is a value between 1 and 16. Over the received data, it creates a checksum byte by shifting the received byte left by 1 bit and XORing that value into the CRC byte (it starts with a value of 0).

After receiving the number of bytes that the .DOWNLD command indicated, Tiny Logo outputs a Ctrl-C plus "RDY 1" plus CR, and expects the sender to send its checksum byte. If both bytes match, Tiny Logo sends Ctrl-C plus "RDY 0" plus a CR to indicate the end of the download, and starts execution of the downloaded code.

It is a good idea to use the command .PROMPT 0 to make Tiny Logo send all errors and messages in condensed format. Tiny Logo sends all data with a starting Ctrl-C character to distinguish the data from any data that the PR command might generate, and terminates all data with a CR.

Results are sent as "OK " plus the value. If the execution of a command did not output anything, Tiny Logo sends "OK nothing".

Errors are sent as "ERR nn", where nn is an error code. If the error contains additional data, like the name of a missing value or procedure, that data follows. For example, the error "NN is not a name" is sent as "ERR 16 NN". Error messages can have up to two additional values.

The string "STOP" is sent whenever Logo is interrupted with a Ctrl-C character. Logo then sends a backtrace of the current execution stack. Each line starts with the text "BT ", followed by the trace line.

5.7 Error messages

The part of the message in italics are parts that Tiny Logo fills in, or supplies in condensed mode. As an example, in condensed mode, the error message "LIGHTS needs a number as its 1. input" is transmitted as "ERR 25 LIGHTS 1" in condensed mode.

1	Division by 0
2	Out of memory
3	Too many nested procedure calls
4	Word is > 255 characters
5	Text too large
6	Bad number base
7	Bad digit
8	Overflow
9	Checksum error
10	Square root of a negative number
11	Missing closing … *) or]*

12	Too many closing … *) or]*
13	Bad bytecode *value*
14	*XXXX* needs more inputs
15	You don't say what to do with *XXXXX*
16	*XXXXX* is not a name
17	*XXXXX* is not a procedure
18	*XXXXX* is a name
19	*XXXXX* is a primitive
20	*XXXXX* is a procedure
21	*XXXXX* has more than 15 inputs
22	*XXXXX* cannot be part of a procedure
23	*XXXXX* must be part of a procedure
24	*XXXXX* needs a word as its *n*. input
25	*XXXXX* needs a number as its *n*. input
26	*XXXXX* needs a list as its *n*. input
27	*XXXXX* 's input is out of range
28	*XXXXX* got nothing as its *n*. input