An Interpreted Language with Debugging Interface for a Micro Controller

Noriaki Mitsunaga
Dept. of Technology Education,
Osaka Kyoiku University
4-698-1 Asahigaoka, Kashiwara, Osaka, Japan
Email: mitunaga@cc.osaka-kyoiku.ac.jp

***Abstract*—We have developed an interpreted language for a beginner to use understanding programming and electronic circuits and making his/her creation. Its interpreter runs on Arduino's micro controller and its grammar resembles to Arduino language (C/C++ programming language). It also has built in debugging interface to visualize pin values and variables. In this paper, we report the implementation of the language and a debugging tool. The language and the tool could be used in education course for micro controller programming to help students.**

I. Introduction

Micro controllers (micro-computers) are becoming more powerful and cheaper than ever and now used in variety of product areas. On top of that, it is said that the number of designers and pupils who use micros is increasing in recent years. They use micros for prototyping and/or learning computers having fun.

There are many programing environments that are popular among them for simplicity of use such as BASIC Stamp [1] and other BASIC compilers, Arduino [2], and mbed [3]. The basis of simplicity is that they make possible to use micros' internal and external circuits, such as digital inputs/outputs, analog inputs, PWM outputs, a character LCD, and so on, with just a few sentences for typical use cases. Also many small circuit modules, which can be easily used with such programming environment, are commercially available.

With those environments and modules, you can use micros just understanding the basic nature of a programing language and the circuit to use. To understand the nature, writing a program, wiring a circuit and verifying the understanding by watching the behavior will be one of the best ways. However, what happens when a beginner finds his/her work is not working as expected, for example it keeps silence? An expert might use a debugger software to investigate in problems in the program, a circuit tester or an oscilloscope to check the circuit. Those tools visualize internals of the program status and electronic circuits and help debugging. However, a beginner tends to avoid using them even if those tools are at hand since he/she needs to learn how to use the tools.

Then, it will be good help for a beginner if there is a programming environment that has facility of a debugger, a circuit tester, and an oscilloscope which can be used without further training. Meanwhile, it is said that an interpreted language is easier to learn than a compiled language since a programmer can interactively check his/her language understanding by the reflective use of the evaluator.

In this paper, we report our interpreted language with debugging interface for a micro controller. It is compatible with Arduino language in a way. The debugging interface enables visualizing variables and pins' status of the micro. The interpreter is implemented on Arduino Uno and the visualization tool (terminal software) runs on a Windows PC.

II. Interpreted Language with Debugging Interface

A. Development environments for micro controllers

Table IV shows comparison of development environments for micro controllers which are regarded as suitable for beginners. They are seven BASIC interpreters/compilers, two visual programming languages, Spin language, and Arduino language which resembles to C/C++ programming language. BASIC is the majority in the table probably because BASIC is regarded as suitable for beginners for a long time. However, rapid popularization of Arduino suggests that there is no verification that grammar of BASIC is more suitable than other programming languages such as C/C++. Also the majority of the environments adopts compiler languages. This is to overcome the limitation of small memory size of MCUs and comply with the demand for execution speed.

The 12Blocks [6] and Scratch [12] has visualization of variables, I/Os, and currently executed statement. Three environments has visualization with simulation tool or debuggers. Other six environments do not have visualization of variables or I/Os or debugging tool then you need so called printf debugging, a circuit tester, or an oscilloscope. Then, most of environments do not have stress on visualization or debugging support.

PIC Pico Basic [4] and Arduino BASIC [5] implements interpreted languages. Then you can use the evaluator reflectively or directly without writing a program. Although they do not have IDE or visualization tools, you can stop the running program and use print command to check variables.

Scratch is a visual programming language targeted for children in age from 8 to 16. You can check a meaning of an icon by clicking and watching how it works reflectively. It also has visualization of currently running statement (icon) and variables. These kinds of visualization would be useful

for people for all ages. Meanwhile, a MCU board is used just for I/O and you always need a PC to run a Scratch program.

B. Design of interpreted language with debugging interface

It should be appreciated that interpreter on an MCU would have limitation due to small size of ROM and RAM sizes compared to PCs. Then it is desired that the user can easily migrate from an interpreted language to a compiled language to overcome the limitation. Both languages should have similar grammar and library APIs.

From the point of implementer's view, it is easier to implement such library APIs of the interpreted language if you can include libraries of the compiled language. It should be noted that some commercially available languages prohibit such uses.

Then we decided that the target MCU board is Arduino and the interpreted language should have grammars and APIs resembles to Arduino language. Arduino is popular for beginners and more than 200,000 Arduino are already shipped. Arduino's libraries are mostly offered with GPL or Lesser GPLs without limitations noted above. Then the interpreter is implemented on Arduino language.

The interpreted language is mostly compatible with Arduino language but has difference as following. It evaluates statements from the beginning of a program but does not support definition of new function since the size of the program will be small. You don't need to declare use of a variable but the name is limited to a single characters from a to z. All of the computing is done in 16 bits signed integer (i.e. short). Table V shows brief specifications of the language.

III. Interpreter with debugging interface

A. Language and interpreter on Arduino

The grammar of the interpreted language resembles to the C programming language and supports digital inputs/outputs, analog inputs, PWM outputs, R/C servo outputs, and tone outputs as Arduino language does. We implemented the language interpreter on an Arduino Uno board [1]. Arduino Uno has an ATmega328P which is an Atmel's AVR architecture micro controller with 32kbytes ROM, 2kbytes RAM, and 1kbytes EEPROM and runs at 16MHz. The size of the interpreter is about 20kbytes and 600bytes are available for user program. The user program is stored in the ASCII text format. The execution speed of the user program is about 1/1500 of the speed of a compiled binary written in C (Arduino) language that runs on the same micro controller. This is due to the time spent for interpreting the user program.

The interpreter has a shell (user interface) via asynchronous serial port at 115.2kbps. Fig.1 shows the message sent from the interpreter at the startup of the micro. Table I shows the commands for the shell. When it prompts "OK", you can request another command or to evaluate an equation. Fig. 2 shows how it evaluates equations and how you write and run a program. The program blinks Arduino's on board LED (pin 13) at 1 [Hz]. The micro controller with the interpreter can run a program without a PC by saving the program into the EEPROM and enabling automatic execution at startup by "save" and "autorun" command.

[1]The interpreter and the terminal are available from http://n.mtng.org/ele/arduino/iarduino.html

TABLE I
Commands for the interpreter's shell

Command	Description
animate	run program with animation
autorun	run program on EEPROM at startup
debug	run program showing current statement
edit <#>	edit program line <#>
list	list program
noauto	do not run program on EEPROM at startup
prog	start inputting a program, type end to finish
run	run program
save	save program on EEPROM
step	run program step-by-step
<eq>	evaluate equation <eq>

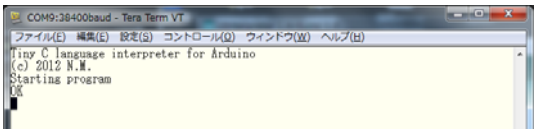

Fig. 1. The interpreter shows a message and a prompt at startup

B. Debugging interface

User shell and debugging interface shares a serial port. All debugging message start with 0x7f and discriminated from user's inputs or shell's outputs. Table II shows the messages. A message is composed of 0x7f, protocol number, and data. The length of a message varies.

C. Programing and visualization tool

We built a terminal software (Fig. 3) to ease programming and visualizing current status of running program. The values (26 variables, 3 digital ports (8 pins each at the maximum), and 6 analog inputs) are displayed and updated every 100[ms]. It is checked to work on a low-end Windows PC with Atom processor (1.6GHz × 1 core). The upper left window is

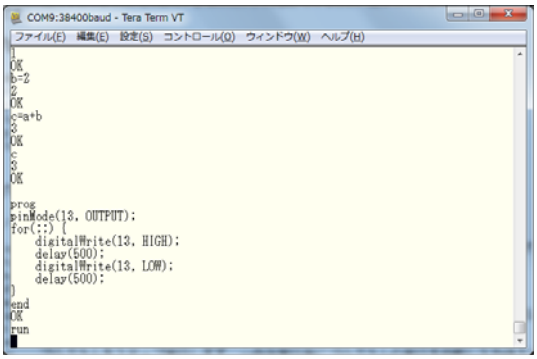

Fig. 2. You can communicate with the interpreter through a serial port

TABLE II
BRIEF DESCRIPTION OF THE DEBUGGING INTERFACE

Proto.#	Description of request/response
0x00	version number
0x01	values of variables
0x04	values of digital IN/OUT
0x0d	program list
0x0e	currently running range in prog.
0x0f	elapsed time from reset in ms
0x10	setup pin's direction
0x11	digital OUT
0x12	PWM OUT
0x13	R/C servo OUT
0x14	setup pin to use for R/C servo output

the main window that has the facility of serial terminal and displays values of variables and analog/digital I/Os. The upper right window shows changes of analog/digital I/Os in a graph. You can check a value by placing the mouse cursor on a curve. The left below is the editor window. It highlights a line that is currently executed on the micro controller. The right below is a window to control pins on Arduino board. You can select a function (digital input/output, analog input, PWM output or R/C servo output) from drop down menu for each pin and set the value by radio buttons or a slider. It also displays a statement corresponds to your operation.

D. Execution speed of interpreted language

We have measured execution speed of interpreted language with the program in Fig. 4. It outputs digital pulse on pin 2 of Arduino and the pulse width is measured by an oscilloscope. Table III shows how the pulse width changes according to the statement put at line 4 of the program. You can see that the execution speed of the interpreted language is about 1/1500 of the compiled language (Arduino-1.0 i.e. GNU C/C++ compiler). You also notice that the pulse width of division is the same as addition since the interpreter mostly uses the time of MCU for interpreting a statement. There is tendency that delay function waits longer than specified with the interpreted language. Meanwhile, the overhead of use of visualization tool is about 5[ms] at most.

The execution speed of interpreted language is not enough to implement software PWM for modulating light, controlling a motor's speed, or scanning matrix LEDs. However, you can use hardware PWMs with the interpreted language. Then, the interpreted language would be useful for small controlling applications whose control frequency is under 20 or 10[Hz], which is enough for handling some human operations and change of ambient lights and so on.

E. Migration to compiled language

You can use debugging interface of the interpreter with Arduino IDE. This will ease migration from interpreted language to Arduino language. Fig. 5 shows an example program. Lines 1 to 4, 6, and 10 are added to the BareMinimum example of Arduino language. By adding those lines, you can monitor and change values of I/O pins on the fly with the

```
1   volatile int a = 2;
2   for (;;) {
3   digitalWrite(2, HIGH);
4   // put a statement here
5   digitalWrite(2, LOW);
6   }
```

Fig. 4. Program used to measure execution speeds. The digitalWrite function outputs H or L according to the second argument on the pin indicated by the first argument. The interpreted language does not need volatile int.

TABLE III
PULSE WIDTHS MEASURED WITH THE PROGRAM IN FIG. 4. BOTH H AND L PERIODS ARE MEASURED.

	Arduino-1.0	interpreted language	with visualization
none	H:3.9us L:3.9us	H:6.0ms L:6.1ms	H:6.0ms∼8.6ms L:6.1ms∼9.2ms
$a = a + 1$	H:4.6us L:3.9us	H:6.1ms H:6.1ms	H:6.1ms∼9.0ms H:6.1ms∼9.2ms
$a = a * 3$	H:4.8us L:3.9us	H:6.1ms L:6.1ms	H:6.1ms∼9.0ms L:6.1ms∼9.2ms
$a = 10/a$	H:44us L:3.9us	H:6.1ms L:6.1ms	H:6.6ms∼9.0ms L:6.1ms∼9.2ms
delay(1)*1	H:1.0ms L:3.9us	H:9.3ms L:6.1ms	H:9.5ms∼13ms L:6.1ms∼9.2ms
delay(10)	- -	H:18ms L:6.1ms	H:18ms∼23ms L:6.1ms∼9.2ms
delay(100)	- -	H:11 × 10ms L:6.1ms	H:11 × 10ms*2 L:6.1ms∼9.2ms

*1 unit of delay is in milliseconds
*2 varies from 108ms to 111ms

terminal program (Fig. 3) or a monitor program (Fig. 6). As for serial interface, the Serial.print functions can be used as usual and debug.available and debug.read can be used instead of Serial.available and Serial.read functions. The delay function is replaced with a function which checks serial interface periodically so that you can use it without care.

```
1 #include <EEPROM.h>
2 #include <Servo.h>
3 #include <iArduino.h>

4 iArduinoHandleProtocol debug;

5 void setup() {
6   debug.begin();
7  // put your setup code here
8}

9 void loop() {
10  debug.check();
11  // put your main code here
12}
```

Fig. 5. An example program to use debugging interface of the interpreted language.

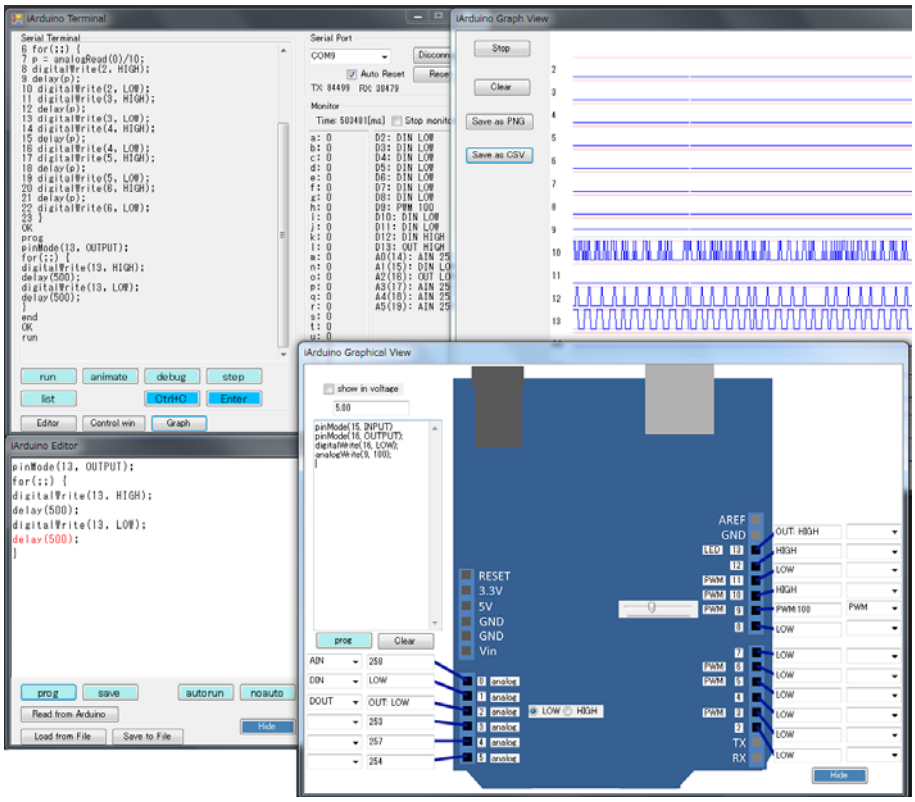

Fig. 3. This is the software to help writing and debugging program for the interpreted language. The upper left includes the serial terminal and displays values of variables and I/Os. The upper right shows changes of analog/digital I/Os in a graph. The left below is the editor window. It highlights a line that is currently executed on the micro. The right below is a window to control pins on Arduino board. You can control pins while the program is running.

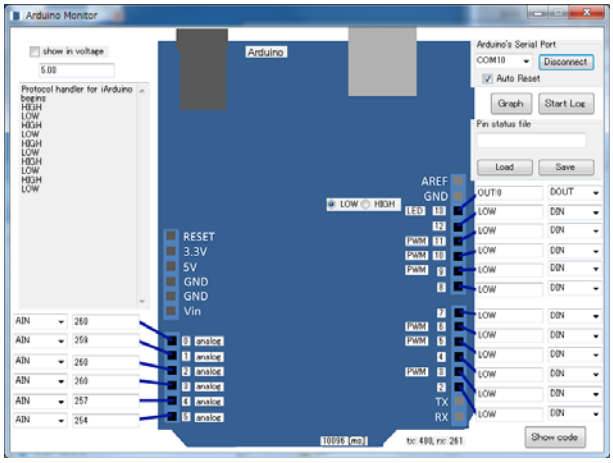

Fig. 6. A monitor program for Arduino to monitor and modify pin values on the fly. Left box in the window is a serial terminal. It can draw graph and log values too. It communicates with Arduino by debugging protocol of the interpreted language.

IV. Conclusion

We have introduced our interpreted language with debugging interface for a micro controller and the terminal software to use with. The interpreter for the language runs on Arduino and the terminal software enables to see internals of the program and pins' status. We hope these tools would be useful for both beginners and experts.

References

[1] Parallax Inc.:BASIC Stamp Windows Editor, (http://www.parallax.com/tabid/295/Default.aspx)

[2] Arduino project: Arduino, (http://arduino.cc/)

[3] mbed (http://mbed.org/)

[4] picobe:PIC Pico Basic, (http://www14.ocn.ne.jp/~picobe/)

[5] Mike Field:Arduino BASIC, (http://ec2-122-248-210-243.ap-southeast-1.compute.amazonaws.com/mediawiki/index.php/Arduino_Basic)

[6] HannoWare:12Blocks, (http://12blocks.com/)

[7] Parallax Inc.:Propeller/Spin Tool Software, (http://www.parallax.com/tabid/407/Default.aspx)

[8] Revolution Education Ltd.:PICAXE Programming Editor, (http://www.picaxe.com/)

[9] MCS Electronics:BACOM-AVR, (http://www.mcselec.com/index.php?option=com_content&task=view&id=14&Itemid=41)

[10] mikroElektronika:mikroBASIC Pro for PIC, (http://www.mikroe.com/eng/categories/view/98/mikrobasic/)

[11] MicroEngineering Labs.:PICBASIC PRO, (http://melabs.com/)

[12] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond:The Scratch Programming Language and Environment, in the *ACM Transaction on Computer Education*, vol.10, no.4, pp.16:1-16:15, 2010.

[13] The Playful Invention Company: PicoBoard, (http://www.picocricket.com/picoboard.html)

TABLE IV
COMPARISON OF DEVELOPMENT ENVIRONMENTS FOR MICRO CONTROLLERS

name (version) (remarks)	hardware	language	language processing	reflective use of evaluator	visualization of I/O, vars, statements	fee for development environment
PIC Pico Basic (20111218) [4]	Microchip PIC24FJ64GA002	BASIC	interpreter (on MCU)	YES	NO (print debug only)	free
Arduino BASIC (ver. 0.03) [5]	Arduino	BASIC	interpreter (on MCU)	YES	NO (print debug only)	free
12Blocks (2.0.2) [6]	Parallax Propeller, PICAXE and so on	visual plogramming languae	compiler	NO	YES(table and graph) (none for PICAXE)	$49∼
BASIC Stamp Windows Editor (v2.5.2) [1]	Parallax BASIC Stamp module	BASIC	compiler (intermediate language)	NO	NO	free
Propeller/Spin Tool Software (v1.3) [7]	Parallax Propeller	Spin language	compiler	NO	NO	free
Arduino IDE(Arduino-1.0) [2]	Arduino	Arduino language (C/C++ based)	compiler	NO	NO	free
PICAXE Programming Editor (ver. 5.5.0) [8]	PICAXE (Microchip PIC MCU with dedicated firmware)	BASIC	compiler (intermediate language)	NO	NO	free
BACOM-AVR (2.0.5.0) [2]	Atmel AVR MCU	BASIC	compiler	NO	on simulator	$0∼
mikroBASIC Pro for PIC (5.40) [10]	Microchip PIC MCU	BASIC	compiler	NO	with debugger	$0∼
PICBASIC PRO (3.0.5) [11]	Microchip PIC MCU	BASIC	compiler	NO	with debugger	$49.95∼
Scratch (1.4) [12] (always needs a PC)	PicoBoard [13] (compatible boards available)	visual programming language	interpreter (on PC)	YES	YES(table)	free

TABLE V
IARDUINO LANGUAGE SUPPORTS FOLLOWING VARIABLES, CONTROL STATEMENTS, CONSTANTS, OPERATORS, LITERALS, AND FUNCTIONS

name of variables	a-z (16 bits int aka. short)
control statements	if, else, for, while, break, continue
constants	LOW, HIGH, INPUT, OUTPUT, true, false
operators	+, -, *, /, %, &, |, ˆ, &&, ||, <, <=, >, >=, ==, ! =, >>, <<
literals	decimal, hexadecimal, binary digits (16 bits int only)
functions	abs, analogRead, analogWrite, delay, digitalRead, digitalWrite, max, millis, min, noTone, rand, pinMode, servo?.attach*1, servo?.write, tone, print

*1 put a number between 0 and 11 into '?'s of servo?.attach and servo?.write