2006-08-16
Recursion Co., Ltd.
Osamu Tamura

ForCy

The Embedded Language Processor

1. Introduction

In the development of embedded systems, there is a need to understand multiple CPU architectures and to be able to use dedicated development environments. This situation gives rise to concerns that long term maintenance, including maintenance of the development environment itself, could become very difficult as the environment evolves over time. In fact, such maintenance problems have already been observed. For example, finding compatible CPU's for ten year old FA systems is very difficult. Reconstruction of the environment to generate the same object code has become almost impossible. In cross development, it is foreseen that reconstruction of older development environments will become increasingly more difficult because manufacturers continually update integrated environments and operating systems as improvements are made in them. In addressing these concerns, I have developed a technique to embed compilers into systems equipped with microprocessors. This will provide for minimum maintenance in the long term when effecting repairs or functional enhancements.

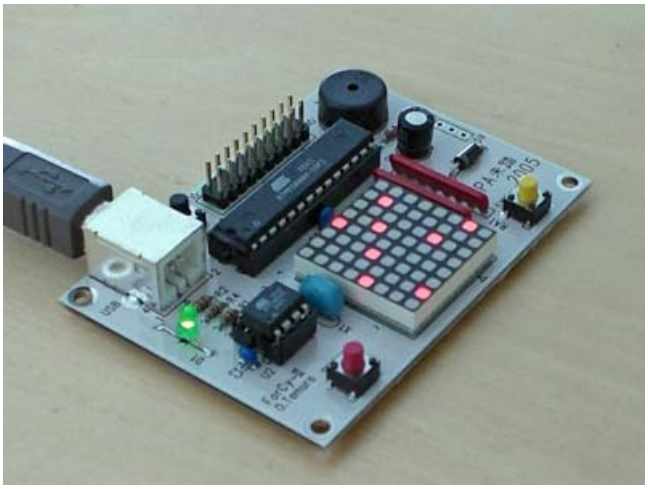

Photo 1: Example Implementation of the ForCy

2. Virtualization of CPU Architectures

CPU's implemented with intermediate code interpreters can be operated as virtual stack machines. One CPU can easily be replaced by another CPU equipped with the same interpreter. With the increased processing power of recent low-end microcontrollers, some decrease in processing efficiency may be acceptable for many applications.

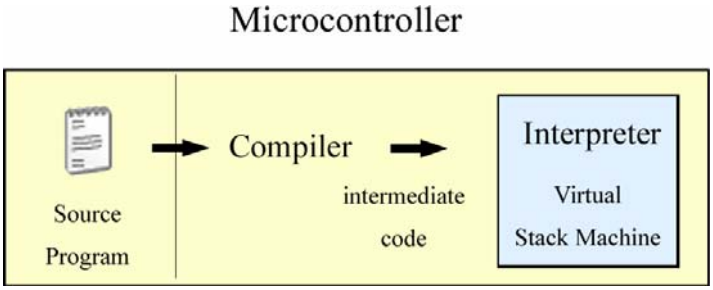

Figure 1: Built-in Compiler and Virtual Stack Machine

The architecture of virtual stack machine is much simpler than JAVA VSM. Like FORTH, it has two stacks. User program can not access the return stack directly.

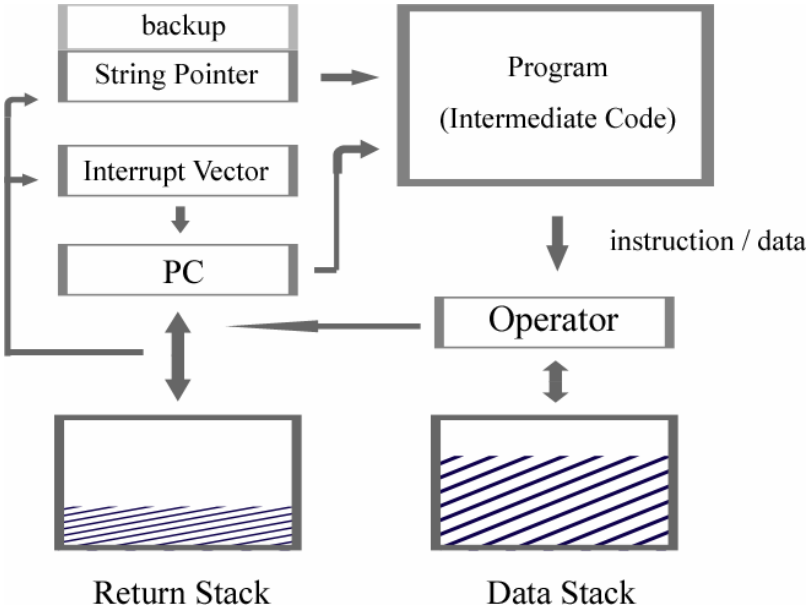

Figure 2: ForCy Stack Machine Architecture

3. Embedded Language Processors

Microcomputers embedded with source programs and language processors require only text editors and a user interface in order to repair or replace code. Source code is internally encoded into intermediate code during the transfer from the host PC. Alternately, the source code can be embedded into the compiling device before compilation. The intermediate code is stored in programmable flash memory or EEPROM's.

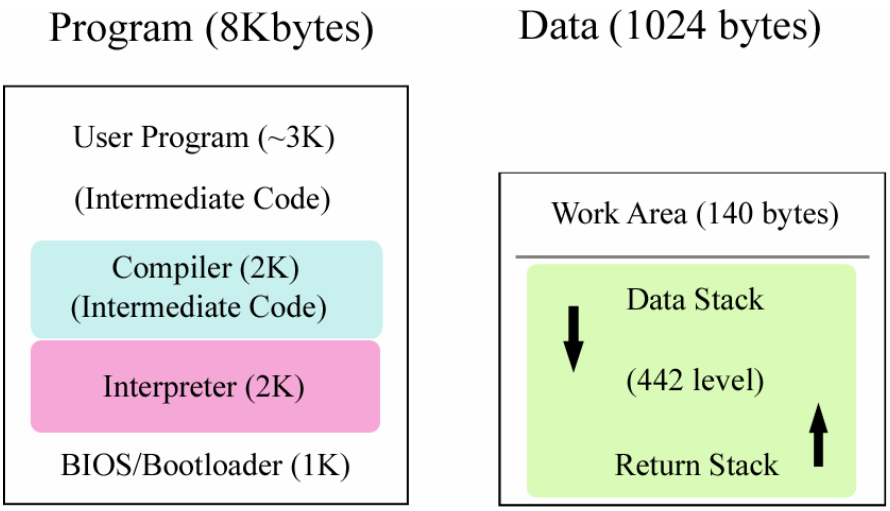

Figure 4: Memory usage of ATmega88

4. Reducing the Overall Bulk of Compilers

The language specification has been written to reduce the code bulk of the compiler. It is reduced to be compatible with the limited amount of memory (in the range of several kilobytes) available to low-end microcomputers. In addition, the language offers low implementation cost and high scalability through modifications and simplification allowing improved program flow. ForCy is a programming language in a postfix notation similar to the C language with PostScript syntax. It generates intermediate code for virtual stack machines. This compiler can handle three data types, such as 16-bit unsigned integers, arrays and strings. Pointer operations are prohibited, and tail callings are optimized to reduce stack consumption. The main process of code generation is identification of words and assignment of byte codes in virtually one-to-one correspondence. A parser counts the length of { } blocks and generates code to be disregarded. Due to the small RAM capacity, the one-pass processing is designed to compile into self-programmable flash memory while receiving source text through a serial communications interface.

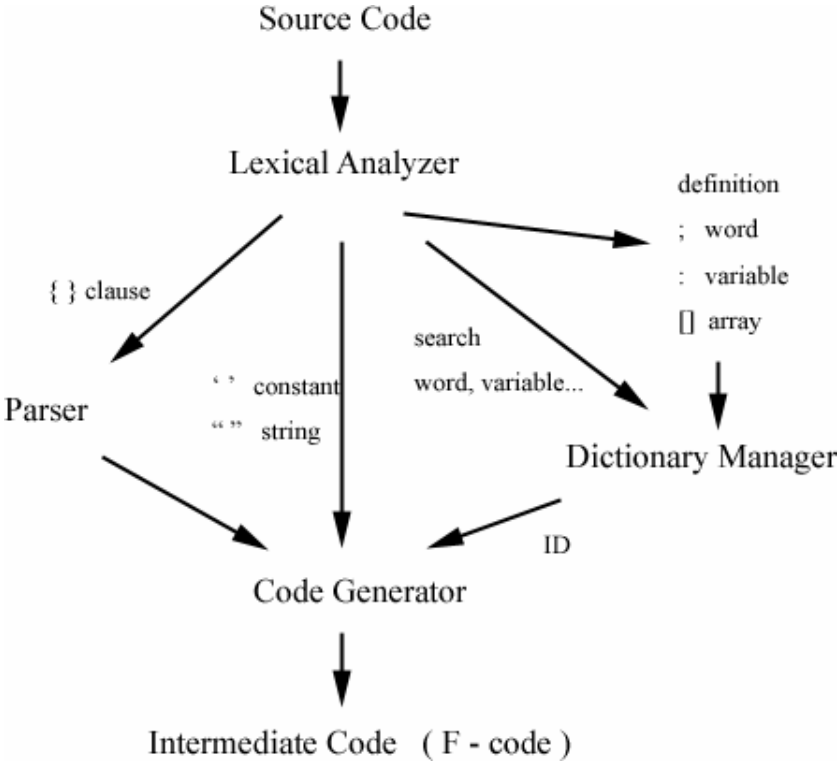

Figure 3: Internal Structure of the ForCy Compiler

The interpreter sequentially calls processes depending on intermediate codes to operate the data stack and the return stack. The compiler itself is also encoded into intermediate codes and operates on the interpreter. The entire language processors including compilers and interpreters can be implemented in 4 to 6 kilobytes of code.
Although ForCy began from FORTH, it has the control syntax easy to get used to C programmer.

```
@s { } do for break while continue if ifelse case switch of self
. .. nip swap dup over @ =
== != < > <= >= ++ -- + - * / % div << >> & | ^ ~ && || ! min max rand
sfr =sfr interrupt clk sec
@c? @c %c %s %d %x
```

List 1: ForCy System Defined Words

```
        "¥n¥tHello, World.¥n" %s                    //  show string
:hello

        10 0                                        //  repeat 10 times
        {
                hello
                ++
        } for ..
:hello_10

        ;i [10]a [10]b [10]c                        //  array

        10 0
        {
                dup =i
                i a   i b   *   i =c                //  c[i] = a[i] * b[i];
                ++
        } for ..
:c=ab

        100 < {
                10 < {
                        ' ' %c
                } if
                ' ' %c
        } if
        %d
:%3d                                                //  show 3 digits

        ;x ;y                                       //  multiply table

        1 =y
        {
                y 9 <= while                        //  repeat from 1 to 9
                1 =x
                {
                        x 9 <= while                //  repeat from 1 to 9
                        x y * %3d                   //  show x * y
                        ++ =x
                } do .
                '¥r' %c '¥n' %c                     //  CR-LF
                ++ =y
        } do .
:multiply

        1 > { dup -- self * } if
:factorial
```

List 2: Example of ForCy Program Code

5. Implementation

This simplified language was implemented using an Atmel ATmega88 (8K words ROM and 1024 bytes RAM). It sequentially compiles program text transmitted through a RS-232C interface. It then writes the results into a flash memory area and interprets it. It can calculate the first 100 digits of the value of PI in 3 seconds using an internal 8 MHz clock. This language is also implemented with a PIC12F683 (interpreter only), PIC16F88, R8C/Tiny and M32R. An interrupt mechanism is available in ATmega88, R8C/Tiny and M32R. Thus it can easily activate tasks at regular intervals and process multiple tasks in quasi-parallel fashion.

USB interface, matrix LED, buttons and piezo-electric buzzer are added on board to evaluate this embedded language processor.

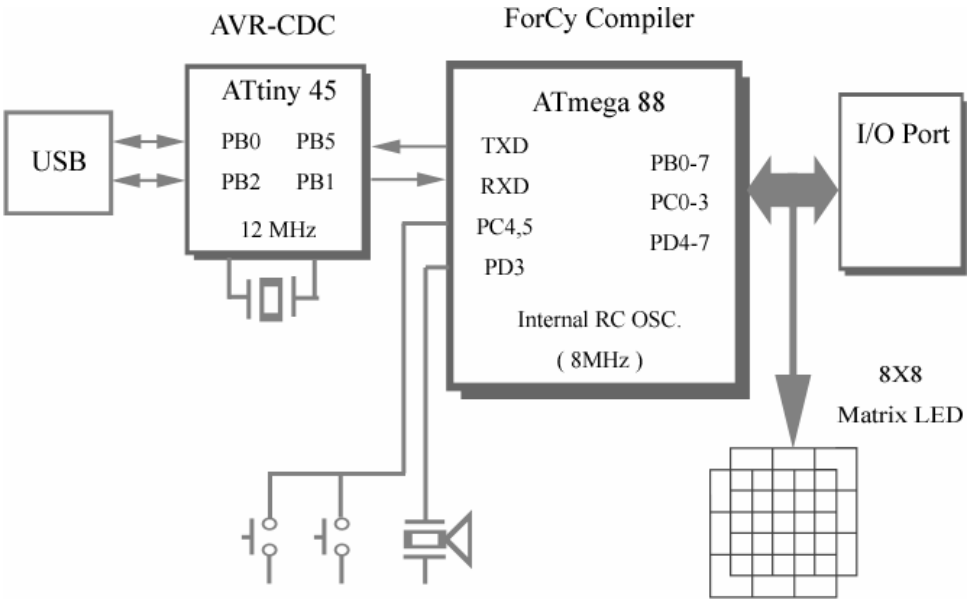

Figure 5: ForCy Evaluation System Hardware Block Diagram

Building ForCy executable on MCU is a little complicated. To reduce the total code size and to enhance the portability, compiler is programmed by ForCy itself.

step 1. Make compiler and interpreter in C.
step 2. Generate ForCy executable on PC using some compiler (e.g. Visual C++).
step 3. Make compiler described by ForCy.
step 4. Generate ForCy compiler intermediate code.
step 5. Make interpreter by the assembler depending on MCU.
step 6. Assemble and Link with ForCy intermediate code.

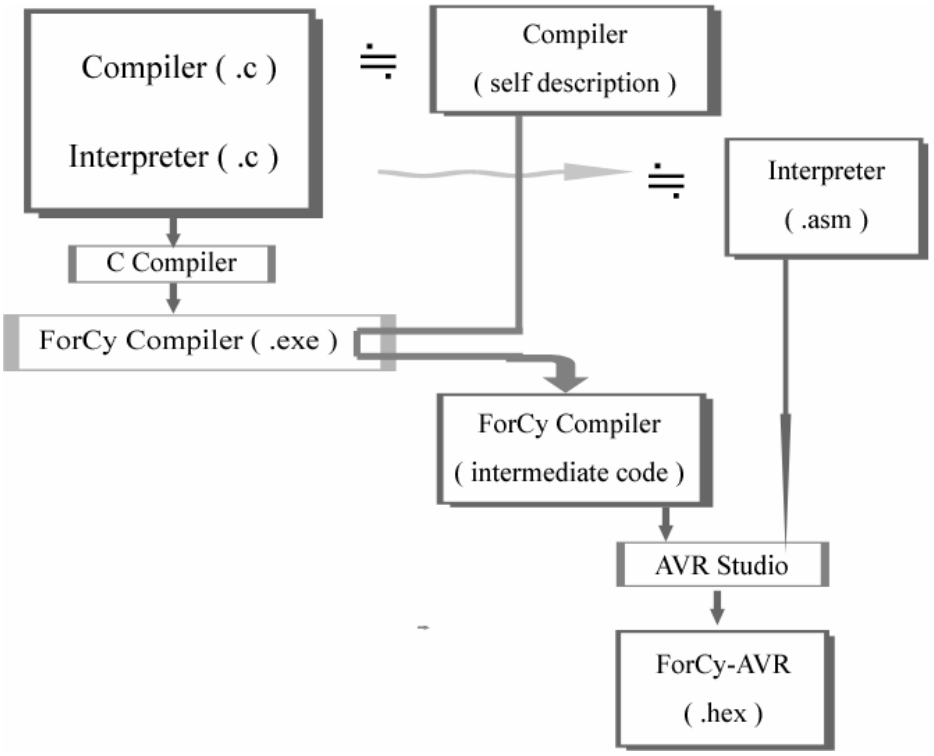

Figure 6: Build Process of the ForCy Executable

```
//      compile
0xff =err
0 =iptr
{
        lex =len          //      lexical analyzer
        len while

        //      parse
        0 token
        {
                ':' {             //      register word
                        ' ' %c puts

                        0 code_out                // iRET
                        cstart 0 1 dic_add =err   // iWRD
                        cptr ++ =cstart           //      keep next entry
                } case

                '¥'' {            //      constant
                        1 token number_out
                } case

                ';' {   //      register variable
                        vptr 4 1 dic_add =err     // iVAR
                        vptr ++ =vptr
                } case

                '[' {             //      register array
                        vptr 6                    // iARY
                        1
                        {
                                dup token ']' == nip break
                                ++
                        } do
                        ++
                        dic_add =err
                        1 atoi vptr + =vptr

                } case

                //      default
                0
                0 token '=' == nip {
                        ++
                } if =eq

                0 dic_find
                0xff ==
                0 token isdigit nip && {
                        .. 0 atoi 2
                } if

                0xff != {
                        1 >> {
                                word_out
                                number_out
                                variable_out
                                array_out
                        } of
                }
                {
                        .. 0xfe =err      // err: undefined word
                } ifelse

        } switch .

        err 0xff != nip break
} do
```

List 3: Self-Programmed ForCy Compiler (portion)

```
SysLP:
                movw    zL, cpL
                clr     tH
                lpm     tL, z+          ; get {} block length
                st      -y, zH          ; push {} entry to RS
                st      -y, zL
                add     zL, tL          ; add length to PC
                adc     zH, tH
                movw    cpL, zL         ; skip {} block
                rjmp    CodeNext

SysFor:
                ld      tH, -x
                ld      tL, -x
                adiw    xL, 2
                cp      tL, dtL
                cpc     tH, dtH         ; index==limit?
                brne    SysDo
                adiw    yL, 2           ; exit
                rjmp    CodeNext
SysDo:
                cpi     cpL, 0
                brne    do_nc
                dec     cpH
do_nc:
                dec     cpL             ; PC--
                ld      tL, y
                ldd     tH, y+1         ; copy {} entry
                st      -y, cpH         ; push PC to RS
                st      -y, cpL
                movw    cpL, tL         ; set {} entry to PC
                rjmp    CodeNext
SysDo_:
                ldd     cpL, y+2        ; copy {} entry to PC
                ldd     cpH, y+3
                rjmp    CodeNext

SysBreak:
                movw    tL, dtL
                ld      dtH, -x
                ld      dtL, -x
                or      tL, tH
                breq    brk_end         ; TOS != 0?
                ld      cpL, y+
                ld      cpH, y+         ; pop PC from RS
                adiw    yL, 2
                inc     cpL             ; PC++
                brne    brk_end
                inc     cpH
brk_end:
                rjmp    CodeNext
```

List 4: ForCy Interpreter for ATmega88 (portion)

6. Communication

To add USB interface to the ForCy evaluation system, CDC protocol is implemented into ATtiny45. AVR-CDC is a USB-RS232C interface using CDC (Communication Device Class) protocol on USB 1.1. Although CDC is a part of USB 2.0 standard, it works on low speed USB. ATtiny45 can handle the 4800bps 8N1 well enough.

1. No dedicated driver necessary. CDC loads Windows built-in usbser.sys.
2. Very low cost. With ATtiny45, this is the cheapest solution for the USB-RS232C interface.

An information file (avrcdc.inf) is needed when connecting it to Windows PC first. The Virtual COM Port appears after the connection established. No procedure is necessary for the CDC connection on Macintosh OS X.
The AVR-CDC portion of a circuit is simple. Vcc should be less than 3.6V to suit USB specification. Quartz crystal is recommended instead of ceramic resonator to avoid SYNC errors. Program size is 2.8KB. Since ATtiny45 has no USART, RS-232C timing (4800bps) is made by 8bit timers and the data is transmitted by USI.

The AVR-CDC is based on Object Development's AVR-USB. I released the added portion as free software (GPL2).

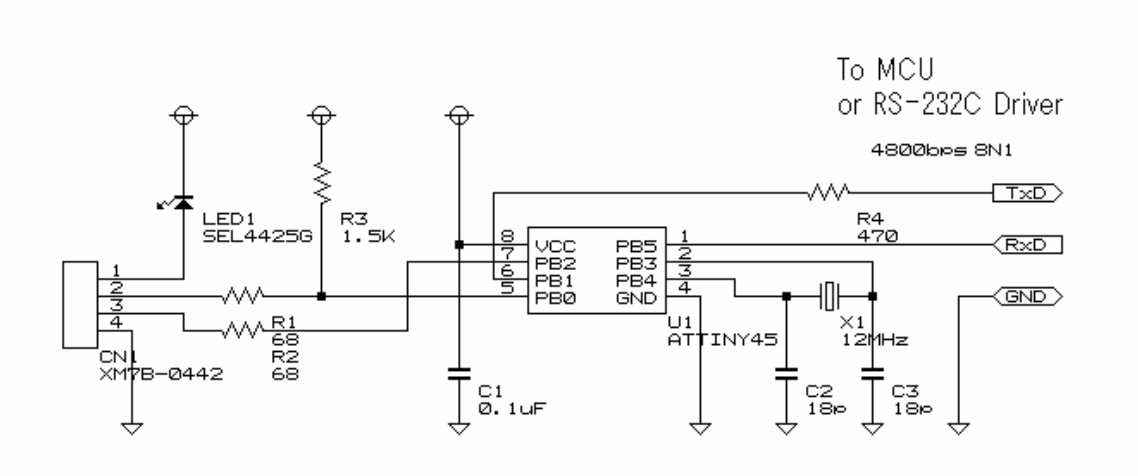

Figure 7: USB-RS232C Interface on Low Speed CDC

7. Usage

Build the circuit and write firmware (forcyavr.hex and cdctiny.hex) into ATmega88 and ATtiny45. High Voltage Serial Programming is necessary for ATtiny45.

When connecting with a USB port of Windows XP/2000 first, a Driver Setup Dialog appears. Specify the folder in which "avrcdc.inf" exists, without searching automatically. Although it is warned that the driver is not certified, confirm it. It only loads Windows' built-in usbser.sys. Then, the Virtual COM port appears.

Set up 'Hyperterminal' with the virtual COM port at 4800bps, 8 data bits, Non-parity, 1 stop bit, No flow control. Choose control keys act as 'Windows keys' if you prefer Ctrl-V to paste.
Establish the connection after attaching the system to a USB port. Close the connection before detaching it from a USB port.

ForCy monitor commands

#	enter compile mode
Esc or ¥	return to command mode
g	execute
d	dump intermediate code
$	set auto start

Hit '#' key and '>' prompt appears. Transfer/paste ForCy program text. While transferring and compiling the program, registered words are displayed. After the last word was registered, hit Esc key. Then hit 'g' to execute. These keys can be included to source code for quick operation.
'Auto Start' automatically start program after reset. Resetting with pressing SW2 button cancels it.
Boot loader is available for updating firmware. After resetting with pressing both SW1 and SW2, transfer HEX file to rewrite the top 7KB of the flash memory. With the boot loader, you can enjoy C/Assembler programming after getting tired of ForCy.

8. Summary

This method was invented and is being developed mainly to reduce development loads of embedded systems and to ensure ease of long-term system maintenance. The following applications are expected to be available:

(1) Control of microcomputers in FA systems for a long-term maintenance.
(2) Customization of behavior for experimental equipment and sequencers.
(3) Integrated management of multiple processors in standalone systems (e.g. robots).
(4) Educational material for microcomputer applications (programmable with only text editors and user interface software).

The ForCy evaluation system has been introduced as an educational material at a vocational school.