

WHYP – A C++ Based Version of *ouForth* for the Motorola 68HC11

Richard E. Haskell
Computer Science and Engineering Department
Oakland University
Rochester, Michigan 48309

Abstract

WHYP (pronounced whip) is a dialect of *ouForth* that is designed to reside in embedded systems. The user interacts with WHYP by running a C++ program on a PC that is connected to the target system through a serial port. The C++ program communicates with a 192-byte kernel that resides in the target system. WHYP contains four kinds of Forth words: primitive Forth words written in the native assembly language of the target; high-level, common Forth words written as colon definitions; high-level Forth words that are specific to the operation of the target microcontroller; and compiler-type Forth words that are written as C++ functions in the PC program. All header and dictionary information is maintained in the C++ program in the PC and only executable, subroutine-threaded code is resident in the target system. An example of WHYP that contains 107 primitive and over 170 high-level Forth words stored in less than half of the 12-Kbyte EPROM of a 68HC711E9 running in the single-chip mode on Motorola's EVBU board is presented. As the user adds additional words they are stored in the 512-byte EEPROM of the 68HC711E9 and can then be added to the microcontroller's EPROM by the user.

Introduction

WHYP (Words to Help You Program) was developed to ease the process of learning a new microcontroller and to provide a low-cost development environment for programming single-chip microcontrollers. It is used in a graduate course on embedded systems design taught at Oakland University where students can buy for under \$68.00 a Motorola EVBU board containing a 68HC711E9 operating in the single-chip mode. The executable part of WHYP is stored in less than half of the 12-Kbyte EPROM of the 68HC711E9 and represents over 250 Forth words. WHYP is based on *ouForth* [1,2] which is a subroutine threaded Forth in which all headers are maintained in a PC connected to the target system through a serial port.

In addition to the E9 part of the 68HC11, versions of WHYP have been implemented for the D3 part on the M68HC711D3EVB board and for the A8 part on the M68HC11EVB board. A 32-bit version of WHYP is currently under development for use on 68000-based embedded systems.

When WHYP is run on the PC it looks and feels like Forth but no attempt is made to be ANSI compatible. Only those Forth words which are usefully executed in embedded systems are actually downloaded to the target system. Other compiler-type Forth words such as IF...THEN, BEGIN...UNTIL, and CREATE...DOES> are implemented as C++ functions on the PC. All of this is transparent to the user, who thinks he or she is programming in Forth directly on the target system.

Over half of the WHYP words stored in the target system are not standard Forth words but are unique to the operation of the particular microcontroller. After discussing the structure of WHYP in the next section, the following section will discuss particular WHYP words that have been implemented for the Motorola 68HC711E9.

Structure of WHYP

In the earlier version of *ouForth* the host program running on the PC was written in 8086 assembly language [2]. In the current version of WHYP this host program is written in C++. This program contains classes for a UART, dictionary, linked list, queue, and s-record. All header information is maintained in the PC in two separate dictionaries. The first dictionary is defined as an object of the C++ class *dict*. In addition to storing header information, this class maintains an image of actual 68HC11 code in the PC segment *tseg*. The class *dict* contains member functions to perform such operations as adding a word to the dictionary, building the dictionary, loading 68HC11 code from an s-record file into *tseg*, searching for a word in the dictionary, compiling colon definitions in the dictionary, saving *tseg* as an s-record file, and saving the headers in a header file. When WHYP is run it loads in a header file and corresponding 68HC11 code from an s-record file. These files could have been generated by WHYP as the result of adding user words to the dictionary. This first dictionary contains primitive Forth words that are implemented in 68HC11 assembly language and high-level Forth words defined as colon definitions.

The second dictionary is used to store all immediate words. These words are only needed at compile time and therefore all the code associated with these words can be maintained in the PC. This immediate dictionary is implemented in

C++ as an array of character strings. This array of strings is searched for a particular parsed word and the word is executed using a switch statement.

WHYP on a Motorola 68HC711E9

The Motorola 68HC711E9 contains a 12-Kbyte EPROM that can be programmed by the user. It also contains 512 bytes of RAM, 512 bytes of EEPROM, an 8-channel A/D converter, a flexible timer that can handle input captures and output compares, an 8-bit input port, an 8-bit input/output port, and both asynchronous and synchronous serial ports. The WHYP kernel takes up less than 200 bytes in the EPROM. This kernel is similar to the ouForth kernel described in [1]. In WHYP the index register *X* is used for the Forth data stack pointer and the HC11 system stack pointer is used for the Forth return stack pointer. The kernel waits for an address to be sent from the PC to the target system through the serial port and then executes the subroutine at that address. The kernel contains three basic subroutines. *TPUSH* pushes the next word from the PC onto the data stack. *TPOP* pops the top of the data stack and sends that word to the PC. *TBLKST* stores *n* bytes from the PC starting at a specified address. If the address is within the range of the EEPROM then *TBLKST* stores the bytes there by programming the EEPROM.

In addition to the kernel the subroutines corresponding to over 250 Forth words are also stored in less than 6 Kbytes of the 12-Kbyte EPROM on the 68HC711E9. These contain all of the Forth words you would expect for stack manipulation, arithmetic, memory access, logical operations, and I/O to the PC. Words such as *U.* and *.S* are implemented by sending an operation code value to the PC followed by the appropriate data. The C++ program running on the PC is then responsible for actually displaying the characters on the screen.

Over half of these built-in WHYP words are unique to the operation of the 68HC11. A sample of some of these words is given in Table 1. Note that they allow the user to immediately access the various features of the 68HC11 without having to program the HC11 registers directly. Of course the user can access these registers and WHYP has made this easy by defining all the Motorola register names as CONSTANTS which return the address of the register. For example, using the word HI in Table 1, a user could set bit 3 of Port B high by typing

```
PORTB 3 HI
```

Interrupts are often awkward to implement in Forth. WHYP attempts to simplify the use of interrupts by providing a number of built-in interrupt-related words. The HC11 has 20 different sources of interrupts. The interrupt vectors for these 20 interrupts are stored at addresses \$FFD6 – \$FFFF which is at the top of the EPROM in the 68HC711E9. WHYP stores RAM addresses from a 3-byte jump table in these locations. This jump table is stored between addresses \$00C4 and \$00FF in the HC11 RAM. WHYP defines a CONSTANT name for each of these interrupt vector jump table locations. For example, the statement

```
00DC    CONSTANT    TOC2.IVEC    \ Timer output compare 2
```

defines *TOC2.IVEC* to be the address of a jump instruction that is to be executed when a timer output compare interrupt occurs. When setting up an interrupt the user must store a *7E* (the opcode of *JMP*) at *TOC2.IVEC* followed by the address of the interrupt service routine. This process is aided by use of the WHYP word *SET.INTVEC* which is defined as

```
      ( set interrupt vector in EVBU jump table )
: SET.INTVEC  ( intser.addr jmp.tbl.addr -- )
              7E OVER C!          \ JMP opcode
              1+ ! ;
```

If the user's interrupt service routine is called *OC2.INTSER* then to set the interrupt vector, the user would define the following word.

```
: SET.OC2.INTVEC  ( -- )
                  [ ' OC2.INTSER ] LITERAL
                  TOC2.IVEC SET.INTVEC ;
```

This word would then be included in the initialization part of the program.

WHYP makes it easy to write interrupt service routines in high-level Forth by providing the words *INT:* and *RTI:* to use in place of the normal colon and semicolon. As an example, Listing 1 shows the code for some built-in WHYP words that can be used to produce a real-time interrupt every 32.77 msec. Note that the interrupt service routine, *RTI.INTSER*, simply increments the variable *TICKS*. The word *TICK.DELAY* can then be called to delay any number of ticks. In addition to creating a header for the defined interrupt service routine, the word *INT:* also moves the data stack pointer (*X*) to a new

location so that high-level Forth words can be used to program the interrupt service routine. The word *RTI*; compiles the return from interrupt opcode (\$3B).

Table 1 – 68HC11 WHYP Words

EEPROM		
ERASE.BULK	(--)	\ Erase all 512 bytes of EEPROM
ERASE.BYTE	(addr --)	\ Erase EEPROM byte at <i>addr</i>
PROG.EEByte	(c addr --)	\ Program byte <i>c</i> at EEPROM <i>addr</i>
EEC!	(c addr --)	\ Store <i>c</i> at EEPROM <i>addr</i> . Erase byte if necessary
EE!	(n addr --)	\ Store <i>n</i> at EEPROM <i>addr</i> and <i>addr</i> +1.
TURNKEY	(cfa --)	\ Execute word with <i>cfa</i> at boot (' JUNK TURNKEY)
Ports		
HI	(addr b# --)	\ set bit <i>b#</i> hi in byte at <i>addr</i>
LO	(addr b# --)	\ clear bit <i>b#</i> in byte at <i>addr</i>
?HI	(addr b# -- f)	\ Is bit <i>b#</i> of <i>addr</i> hi?
Timer		
@TCOUNT	(-- n)	\ Read counter
OC2F.CLR	(--)	\ Clear OC2F - bit 6 - in TFLG1
25.MSEC	(cnt -- cnt+50000)	\ wait 25 msec.
S.DELAY	(n --)	\ delay <i>n</i> seconds
A/D Converter		
ADCONV.ON	(--)	\ Turn A/D converter on
ADCONV.OFF	(--)	\ Turn A/D converter off
?ADCONV.DONE	(-- f)	\ Is conversion done?
ADCONV	(ch# -- val)	\ Avg 4 readings from channel <i>ch#</i>
ADCONV03	(-- v3 v2 v1 v0)	\ Convert channels 0 - 3
ADCONV47	(-- v4 v5 v6 v7)	\ Convert channels 4 - 7
Serial Communication Interface - SCI		
SCINIT	(--)	\ Initialize SCI port
TX!	(c --)	\ Transmit character <i>c</i>
?RX	(-- c tf ff)	\ Receive character
BAUD	(n --)	\ Set baud rate
Serial Peripheral Interface - SPI		
INITSPI.M	(--)	\ Initialize SPI as Master)
INITSPI.S	(--)	\ Initialize SPI as Slave)
?SPI.DONE	(-- f)	\ Is SPIF flag set?
SPIF.CLR	(--)	\ Clear SPIF flag
SS.HI	(--)	\ Set SS pin high
SS.LO	(--)	\ Set SS pin low
SPI!	(c --)	\ Send <i>c</i> out SPI data register
SPI@	(-- c)	\ Read SPI data register

Other useful WHYP words that are built into the 68HC711E9 EPROM include a sine lookup table that can be used to find sine and arc sine values. A character queue data structure is implemented with words *CHECKQ* (-- *c tf | ff*) to check and get a byte from the queue and *QSTORE* (*c* --) to store a byte in the queue. The word *DUMP* (*addr cnt* --) will produce a hex dump of *cnt* bytes of memory starting at address *addr*.

The Motorola EVBU board includes a MC68HC68T1 peripheral device which contains a real-time clock/calendar, a 32 x 8 static RAM, and a synchronous, serial, three-wire interface for communication with the HC11 via the SPI port. WHYP supports this device with the following words:

T1.INIT	(--)	\ initialize T1
SET.TIME	(hr min sec --)	\ set time
.TIME	(--)	\ display time
@T1	(addr -- byte)	\ fetch <i>byte</i> at <i>addr</i>
!T1	(byte addr --)	\ store <i>byte</i> at <i>addr</i>

Listing 1 – Real-time Interrupt -- Used for delay

```
VARIABLE  TICKS
HEX

: RTIF.CLR      ( -- )          \ clear RTI flag
                40 TFLG2 C! ;

: RTI.SET32     ( -- )          \ set RTI rate to 32.77 msec
                PACTL 0 HI
                PACTL 1 HI ;

INT: RTI.INTSER ( -- )          \ increment TICKS
                1 TICKS +!
                RTIF.CLR

RTI;

: SET.RTI.INTVEC ( -- )
                [ ' RTI.INTSER ] LITERAL
                RTI.IVEC SET.INTVEC ;

: RTI.INT.ENABLE ( -- )
                TMSK2 6 HI ;

: RTI.INT.DISABLE ( -- )
                TMSK2 6 LO ;

: TICK.DELAY   ( n -- )          \ delay n ticks
                >R TICKS @
                \ ticks0
                BEGIN
                TICKS @ OVER -    \ ticks0 elapsed
                R@ U>=
                UNTIL
                R> 2DROP ;
```

Summary

WHYP is a Forth system that turns the 68HC711E9 microcontroller on Motorola's EVBU board into a powerful embedded system development tool for producing single-chip applications. Inasmuch as students can buy this board for under \$68.00 this represents a very cost-effective development system. Students simply buy their own boards and develop their projects on their own PCs. In addition to standard Forth words, WHYP contains a large number of built-in words that allows a user easy access to the special features of the microcontroller. WHYP also contains built-in words for implementing a fuzzy controller. In this case the user can download the membership functions and fuzzy rules into the HC11 EEPROM and use most of the fuzzy control words already in the EPROM to implement the fuzzy control algorithm. This system has been used successfully in several fuzzy control applications.

References

1. R. E. Haskell, "Design of a Subroutine Threaded Forth for Embedded Systems," Proc. 1992 FORML Conference, pp. 58-62, Pacific Grove, CA, November 27-29, 1992.
2. R. E. Haskell, "ouForth – a Subroutine Threaded Forth for Embedded Systems," Proc. 1993 Rochester Forth Conference, pp. 62-66, Rochester, NY, June 23-26, 1993.