

# Design of Embedded Systems Using WHYP

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

## Abstract

Forth has proved to be an ideal language for the design of embedded systems. WHYP (pronounced *whip*) is a version of Forth developed at Oakland University and used in a senior/graduate course in the design of embedded systems. A 16-bit version of WHYP has been written for the Motorola 68HC11 and a 32-bit version has been written for the Motorola 68332 microcontroller. In each case 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 small kernel that resides in the target system. WHYP is a subroutine-threaded Forth in which a large number of WHYP words are resident in the target system in the form of executable subroutines. These words are executed by sending the address of the subroutine to the target system over a serial line. All header and dictionary information is maintained in the C++ program in the PC. In addition, all compiler-type Forth words are maintained in a separate immediate-word dictionary in the PC and are executed as C++ functions on the PC, inasmuch as they are never needed in the final stand-alone target system. This paper will describe the experience in using WHYP in the classroom with three different 68HC11 parts. The latest version of WHYP includes many built-in words to ease the design of embedded systems including words for designing fuzzy logic controllers and implementing multitasking.

## Introduction

The last twenty years have seen a dramatic change in the technology associated with microprocessors and microcomputer interfacing. A definite trend in microcomputer interfacing and in digital design in general is a shift from hardware design to software design. Microcomputer interfacing has always involved both hardware and software considerations. However, the increasingly large-scale integration of the hardware together with sophisticated software tools for designing hardware means that even traditional hardware design is becoming more and more a software activity.

I have taught microprocessor courses at Oakland University for the past twenty years. These have included courses that taught assembly language for several different microprocessors including the 6800, 6502, 6809, 68000, and 8086. For the past fifteen years I have taught an embedded systems course that used Forth to design embedded systems using microcontrollers such as the Motorola 68HC11 and 68332. I have used numerous versions of Forth in these classes including MaxForth, eForth, ouForth, and most recently WHYP.

WHYP stands for *Words to Help You Program* and is a subroutine threaded Forth designed specifically for embedded systems [1-3]. Versions of WHYP have been written for both the Motorola 68HC11 and 68332 families of microcontrollers. The 68HC11 version has been used with the A8, E9, E20, D3, and D0 parts. A simple 68HC11 kernel is used in the target system to communicate with the host PC as shown in Figure 1. The executable part of WHYP representing over 300 WHYP words is stored in under 6 Kbytes of memory in the target system (e.g. the 68HC711E9 EPROM) as 68HC11 subroutines. The WHYP words in the target system communicate with a host C++ program that is running on a PC through an asynchronous serial line. To execute a particular WHYP word the address of the subroutine is sent from the PC to the target system as shown in Figure 1. All headers are maintained in a dictionary in the C++ program. When WHYP is run on the PC it looks and feels like Forth. Only those Forth words which are usefully executed in embedded systems actually exist in 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.

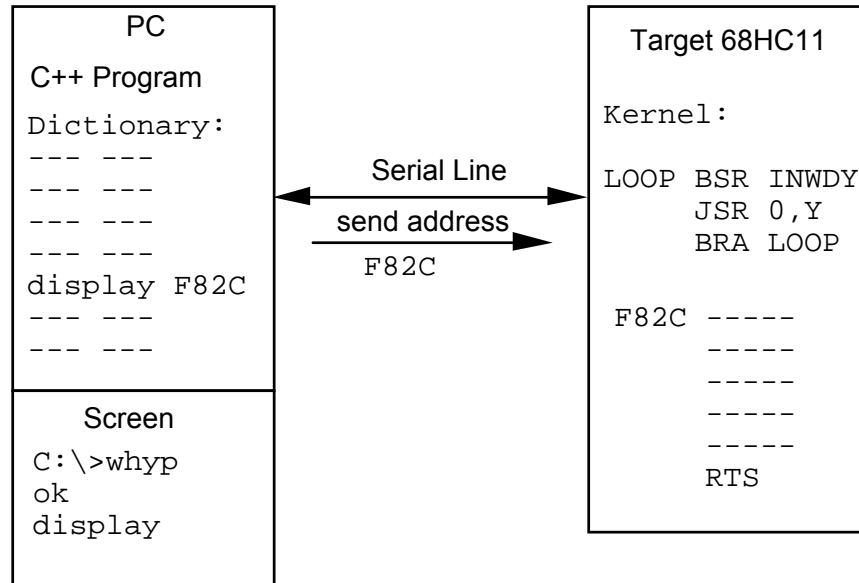


Figure 1 The structure of WHYP

## Design of Embedded Systems Course

The version of WHYP used in the course is designed to be embedded in a 68HC711E9 microcontroller operating in the single-chip mode on a Motorola EVBU board. Inasmuch as students can buy this board for under \$68.00 this represents a very cost-effective and powerful development tool for producing single-chip applications. 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.

The WHYP code is stored in the EPROM of the 68HC711E9 starting at address \$E000. This same code can be stored in a 2764 EPROM and replace the Buffalo monitor at addresses \$E000-\$FFFF on a Motorola EVB evaluation board. This board has 16 Kbytes of RAM including 4 Kbytes from \$D000-\$DFFF. The 4 Kbytes of the EPROM from \$D000-\$DFFF on the 68HC711E9 on the EVBU board can be used by the user to store an application program. An easy way to generate this code is to run the program on an EVB board where the \$D000-\$DFFF address range is in RAM, upload the code to an s-record file using the WHYP word *S.FILE*, and then program the 68HC711E9 using the s-record file.

It is also possible to load a user program from a disk file into a segment of the PC memory instead of the target system at offset addresses that map to EPROM addresses in the target system. This code can then be uploaded to an s-record file that can be used to program the EPROM in the target system.

The topics covered in the embedded systems course are shown in Figure 2. In Part I the various hardware features of the 68HC11 can be explored in an interactive way using WHYP. Many built-in WHYP words make this easy. For example, the statement *3 ADCONV* will read channel 3 of the 8-channel A/D converter four times and leave the average of these four readings on the stack. Similarly, the statement *5 SEND.SPI* will send the 8-bit value 5 out the synchronous serial (SPI) line (MOSI pin). The SS pin on the 68HC11 (PD5) can be set low or high using the statement *SS.LO* and *SS.HI* respectively.

<p>Part I: Exploring the 68HC11 Using WHYP</p> <ol style="list-style-type: none"> <li>1. The 68HC11 Family of Microcontrollers</li> <li>2. Programming in WHYP</li> <li>3. Parallel Interfacing</li> <li>4. The Serial Peripheral Interface (SPI)</li> <li>5. Analog-to-Digital Converters</li> <li>6. Timers</li> <li>7. The Serial Communication Interface (SCI)</li> </ol> <p>Part II: Software Development Using WHYP</p> <ol style="list-style-type: none"> <li>8. Designing With Interrupts</li> <li>9. WHYP Arithmetic</li> <li>10. Strings and Number Conversions</li> <li>11. WHYP Defining Words</li> <li>12. Data Structures</li> <li>13. Fuzzy Control</li> <li>14. Multitasking</li> </ol> <p>Part III: Inside WHYP</p> <ol style="list-style-type: none"> <li>15. The 68HC11 WHYP Kernel</li> <li>16. 68HC11 Primitive WHYP Words</li> <li>17. High-level Built-in WHYP Words</li> <li>18. WHYP C++ Classes</li> <li>19. The C++ WHYP Host</li> </ol> <p>Part IV: 32-bit Microcontrollers</p> <ol style="list-style-type: none"> <li>20. A 32-bit WHYP for the Motorola 68020</li> <li>21. The M68332 Microcontroller</li> <li>22. The 68332 Timer Processor Unit (TPU)</li> </ol>
--

Figure 2 Topics covered in embedded systems course

In Part II of the course WHYP is used to develop sophisticated software systems on the 68HC11. The words *INT:* and *RTI:* are used to write interrupt service routines in WHYP [1]. A complete set of 16-bit and 32-bit words for doing signed arithmetic are built into WHYP. This includes a table of sine values (0 – 90\_) and the arcsine word *ASIN.* WHYP contains the usual collection of Forth words for string and number conversions including the word *(.)* which converts a single number to a counted ASCII string. WHYP contains the defining words *CREATE...DOES>* as well as a built-in queue data structure. There are also built-in WHYP words to make it easy to implement a fuzzy controller [4] and a standard Forth multitasker.

Part III of the course looks at how WHYP works. The WHYP kernel and primitive words are written in 68HC11 assembly language. The 68HC11 index register *X* is used as the data stack pointer and the system stack is used as the return stack. The high-level WHYP words get compiled to 68HC11 subroutines that are also stored in the target system. The complete C++ program that is run on the PC is discussed including C++ classes for a linklist, uart, queue, dictionary, and s-record.

In Part IV of the course a 32-bit version of WHYP is described that can be downloaded into a Motorola 68332 microcontroller target system. In this case all stack values are 32-bits and all of the WHYP words are written in 68020 assembly language. This version of WHYP makes it easy to set up and control interactively the 68332 Timer Processor Unit (TPU) which is a separate co-processor that can be programmed to produce a variety of timer functions on 16 different input/output channels.

As an example of the kinds of experiments done in this class we will consider the design of a digital compass using WHYP.

## Design of a Digital Compass

A small analog hall-effect transducer (No. 1525) is produced by Dinsmore Instrument Company (1814 Remell St., Flint, MI 48503) that can be used to sense direction of the horizontal component of the earth's magnetic field. The sensor has six leads: four for power and ground and two output channels, *A* and *B*. The two outputs, *A* and *B*, are quadrature encoded sinusoidal waves whose relative phase depends on the orientation of the sensor with respect to the earth's magnetic field as shown in Figure 3. The output voltages vary from a minimum of about 2.1 volts (*MinAD*) to about 2.9 volts (*MaxAD*). These two signals are fed to channels 2 and 3 of the 68HC11's A/D converter.

The WHYP code for the digital compass is shown in Listing 1. The *A* and *B* voltages are read using the word *read.A/D* (*-- A B*). The values *A* and *B* are signed values relative to the zero, or *dc*, level of the sine waves. This *dc* value is continually measured by keeping track of the maximum and minimum voltages read in the two variables *MaxAD* and *MinAD*. The variable *Amp* contains the amplitude of the sine waves, *MaxAD* - *dc*.

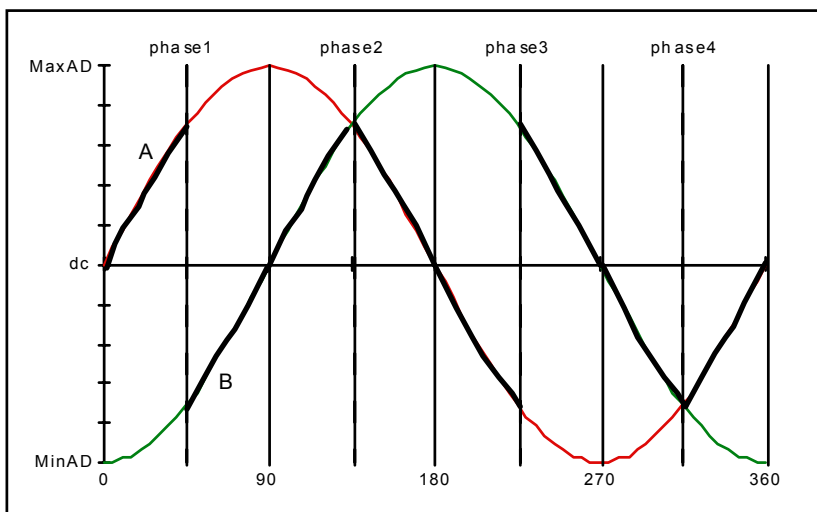


Figure 3 Quadrature encoded output of Dinsmore Hall-effect compass transducer

For maximum accuracy it is best to use the nearly linear regions of the curves shown as heavy lines in Figure 3. These are the regions in which small degree changes produce the largest voltage change. The word *read.compass* (*-- deg*) in Listing 1 determines which of the four phases in Figure 3 the reading is in by comparing the signs of *A* and *B*. Once the phase is determined the appropriate curve to use (corresponding to the heavy lines in Figure 3) is determined in the words *phasex* ( $x = 1-4$ ) by comparing the absolute values of *A* and *B*. The built-in arcsine WHYP word *ASIN* is used to compute the degree.

The degree value (0 - 359) is displayed on three common-cathode 7-segment displays using the Motorola MC14499 Decoder/Driver with Serial Interface chip which can drive up to four 7-segment displays from a synchronous serial line. The 68HC11 SPI port is used to drive these displays using the word *.4leds* (*n --*) shown in Listing 1. This word takes the degree value, *n*, on the stack and first converts it to an ASCII string using (*U.*) and then packs two *BCD* digits per byte (with leading blanks) using the word *pack2* (*addr -- c*) in Listing 1. Two of these packed bytes are then sent to the MC14499 using the WHYP word *SEND.SPI*. The SS line of the 68HC11 is connected to the enable pin of the MC14499 which must be low to shift the data in and then goes high to latch the data into the chip. This chip has a built-in oscillator that scans the 7-segment data to each of the three 7-segment displays continuously.

## Listing 1

```
\      Digital compass      File: COMPASS.4TH

VARIABLE Amp      \ max amplitude of sin wave
VARIABLE MaxAD    \ max A/D reading
VARIABLE MinAD    \ min A/D reading
DECIMAL
\ 4 LEDs Using the MC14499 Decoder/Driver with Serial Interface

: pack2      ( addr -- c )
             DUP C@
             4 LSHIFT      \ addr c1
             SWAP 1+ C@    \ c1 c2
             15 AND OR ;

: .4leds     ( n -- )
             SS.LO 10 BASE !
             (U.) 4 SWAP - \ addr #blanks
             FOR          \ addr
               1- 15 OVER C! \ store F for blank
             NEXT
             DUP pack2 SEND.SPI      \ 1st digit
             2+ pack2 SEND.SPI      \ 2nd and 3rd digit
             SS.HI ;

\ Compass Using Dinsmore Analog Hall-Effect Sensor

: check.maxmin ( n -- )      \ update MaxAD & MinAD
              DUP MaxAD @ MAX \ n mx
              MaxAD !        \ n
              MinAD @ MIN    \ mn
              MinAD ! ;

: get.dc      ( -- dc )      \ get zero reading of sine
waves
              MaxAD @ MinAD @ \ mx mn
              OVER + 2/      \ mx dc
              TUCK - Amp ! ; \ Amp = MaxAD - dc

: read.A/D    ( -- A B )
              3 ADCONV      \ b
              DUP check.maxmin \ b
              2 ADCONV      \ b a
              DUP check.maxmin \ b a
              get.dc        \ b a dc
              TUCK -        \ b dc A
              -ROT - ;      \ A B

: ASIN.scale  ( n -- deg )   \ scaled arc sine
              10000 Amp @ DUP 0= \ make sure Amp <> 0
              IF                \ if Amp = 0
                DROP OVER      \ take arcsine of 1
              THEN
              */ ASIN ;

: ACOS.scale  ( n -- deg )   \ scaled arc cosine
              ASIN.scale 90 SWAP - ;

: phasel     ( A B -- deg )
              ABS 2DUP <
              IF
                DROP ASIN.scale
              ELSE
                NIP ACOS.scale
              THEN ;
```

### Listing 1 (cont.)

```
: phase2      ( A B -- deg )
              2DUP >
              IF
                NIP ASIN.scale 90 +
              ELSE
                DROP ACOS.scale 90 +
              THEN ;

: phase3      ( A B -- deg )
              SWAP ABS 2DUP >
              IF
                NIP ASIN.scale 180 +
              ELSE
                DROP ACOS.scale 180 +
              THEN ;

: phase4      ( A B -- deg )
              2DUP <
              IF
                NIP ABS ASIN.scale 270 +
              ELSE
                DROP ABS ACOS.scale 270 +
              THEN ;

: read.compass ( -- deg )
              read.A/D                \ A B
              2DUP XOR 0<             \ phase 1 or 3
              IF
                2DUP >
                IF
                  phase1
                ELSE
                  phase3
                THEN
              ELSE                       \ phase 2 or 4
                2DUP + 0>
                IF
                  phase2
                ELSE
                  phase4
                THEN
              THEN ;

: compass     ( -- )
              SPI.INIT
              0 MaxAD !
              255 MinAD !
              BEGIN
                read.compass
                359 SWAP - .41eds
                5000 FOR NEXT
              AGAIN ;
```

### References

1. R. E. Haskell, "WHYP – A C++ Based Version of ouForth for the Motorola 68HC11," Proc. 1995 Rochester Forth Conference, Rochester, NY, pp. 46-49, June 21-24, 1995.
2. R. E. Haskell, "ouForth – a Subroutine Threaded Forth for Embedded Systems," Proc. 1993 Rochester Forth Conference, Rochester, NY, pp. 62-66, June 23-26, 1993.
3. R. E. Haskell, "Design of a Subroutine Threaded Forth for Embedded Systems," Proc. 1992 FORML Conference, Pacific Grove, CA, pp. 58-62, November 27-29, 1992.
4. R. E. Haskell, "Fuzzy Control in Forth," Proc. 1994 Rochester Forth Conference, Rochester, NY, pp. 28-31, June 22-25, 1994.