ISD - the ideal complement to XTC

A P Haley, H P Oakford, C L Stephens

Computer Solutions Ltd
1 Goomore Lane, Chertsey, Surrey KT16 9AP, England

The availability of a high level programming language such as polyFORTH on a wide range of microprocessors coupled with the low cost of the current generation of PCs suggests that significant savings can be made if Cross Target Compilers (XTC's) can be made available on the PC. However, the use of a Cross Target Compiler immediately loses the interactive development of hardware oriented parts of an application that is so characteristic of FORTH and from which so many of its gains derive.

This paper will describe a package called In Situ Development (ISD) which aims to provide an easily implemented technique to allow developers of standalone applications hardware and software to take advantage of Cross Target Compilers without losing direct contact with their hardware.

Cross Target Compilers (XTC)

The FORTH, Inc dialect of the FORTH language polyFORTH is available in compatible 16 bit configurations for a large number of processors, specifically :-

8086, 8085, 6809, 1802, PDP-11, 6502

and with minor modifications to suit individual machines (eg High/Low byte swapping) all the Target Compilers for these machines are the same. Hence, it is relatively simple to use a development system for micro A to produce code that will run on micro B. With the wide availability of low cost Personal Computers there is a large financial incentive to use these machines as development systems even if their CPU (8086) is not going to be used in the application.

However, when one is familiar with using FORTH to interact with the hardware during development, Cross Target Compiling a complex real time application into a dedicated board can mean slow turnround and isolation from the hardware.

It is worth at this point identifying what capabilities inherent in FORTH are lost if Cross Compilation is used :-

a) It is not possible to test the machine code parts of any application without transferring to the target board.

b) It is not possible to check the speed of critical words without transferring to the target board.

c) It is not possible to check interrupt handling.

d) Hence, it is not possible to fully test Multi-Tasking areas.

Perhaps most importantly

e) It is not possible to access the I/O bus of the application.

f) Hence, it is not possible to test the performance of interfaces, activators, transducers etc.

It should be stressed that these are not faults in FORTH, no other Cross Target Compiler system provides them. They are in fact advantages of FORTH that are being lost because of the remoteness of the Target environment from the development environment.

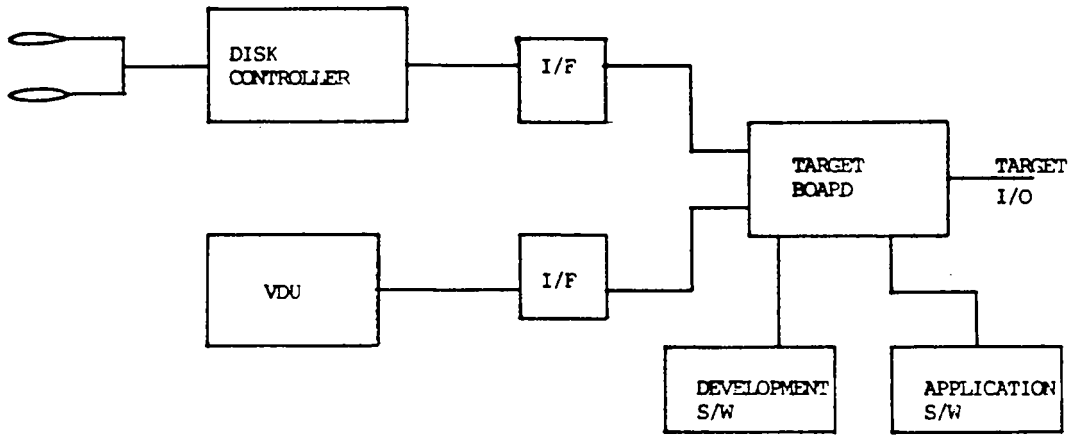
The first and most straightforward way to improve the development process is to use a ROM emulator that can be loaded from the development system, which cuts out the ROM burning step as well as the fiddly mechanical removal and replacement of chips. While this improves the turnround time, it still means that for a complex real time task with significant I/O and multi-tasking the chances of a program change solving the problem under consideration are only about 1 in 3 and many iterations are needed.

In Situ Development (ISD)

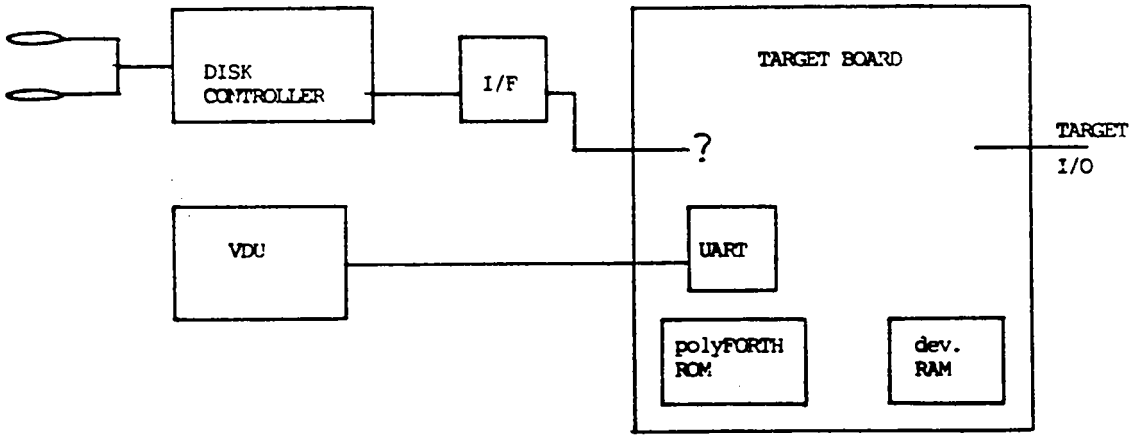
ISD is a technique designed to overcome the problems of developing code for single board computers (SBC's) which cannot be made part of a conventional development system. Clearly, if an SBC to be used in an application as a member of a popular Bus system such as MULTIBUS, then the most advantageous development system for that board is to add disk drivers and any other cards required so that the development is done on that card.

However, if the card is bought in for a specific application then the costs of the disk interface and the chassis may represent a high overhead for a single job. If the board is to be manufactured specially for a product, the board layout would be made more complex and extra components required if it is to conform to a standard.

The idea of ISD is to provide a development system that will run on the target hardware with the minimum of hardware construction required and the maximum ability to re-utilize equipment at the end of the project. The items required are as follows :-



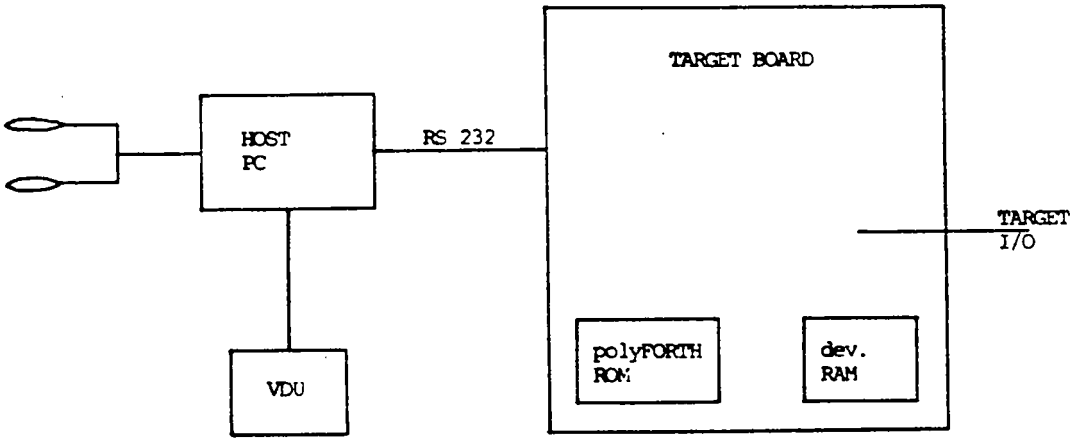
The majority of intermediate sized projects that we have encountered with high I/O content and multi-tasking have not had a disk but have had 8K ROM, at least 4K RAM and a serial I/O device. This configuration is enough for polyFORTH which gets the normal Editor, Assembler, Compiler, Interpreter, Nucleus words and I/O drivers into 7K and requires 2K of RAM for disk buffers. In these cases the diagram becomes :-



An analysis of the requirements of the disk controller and interface show that it must be disk compatible with other media likely to be in use and that its transfer rate, while the higher the better, does not need to be very fast as during development it is typically transferring only one or two 1K blocks for each step of program development (human response time/boredom tolerance - 4 seconds). This means that with suitable software, the disk controller and VDU could share the serial interface and leads inevitably to the use of the low cost PC as a display/floppy combination communicating with the Target via a

serial link.

A suitable program (written in FORTH of course) residing on the Host, can now take input from the Host's keyboard and pass it to the Target. In the Target, normal FORTH keyboard interpretation takes place and terminal output goes back up the serial port to the Host where it is routed to the screen. Should a command such as **9 LOAD** be entered into the Host, then it is transmitted to the Target as a character string, interpreted and the disk driver in the Target is called. This disk driver calls the serial driver to pass a control character requesting a numbered disk block from the Host. The Host program does not display this control character but instead transmits the required block to the Target. The Target system then becomes a full development system capable of providing the programmer with rapid interactive access to FORTH words and hence to timings and I/O control.



In this configuration, there is no reason why the Host and Target should be the same CPU. If the development RAM can be large (eg 32K) then once all the application was tested on the target board, the Target FORTH would be quite capable of target compiling a ROM based system. However, with the RS232 link, this would be slow, and how would the original Target polyFORTH ROM be generated initially? If a Cross Target Compiler on the Host is used then it can both generate the Target FORTH system and later can allow compilation of tested code to replace the polyFORTH ROM with the application ROM.

If the Target system does not have the resources to allow even the basic 8K system then all is not lost, as for development purposes, it is usually relatively straightforward to build a board that piggy-backs onto the Target system via the CPU socket to provide any necessary extra development facilities such as UART or more ROM and RAM.

ISD The Protocal

The link protocal between the Host and Target is very simple, the Target is always the master and controls the link operation. The only handshake used is at the end of a block transfer when the Host responds with ACK if no disk failure has occurred or NAK if a disk fault was present.

Only 4 control codes are emitted by the Target :-

Code 0 - no action - used to allow null characters

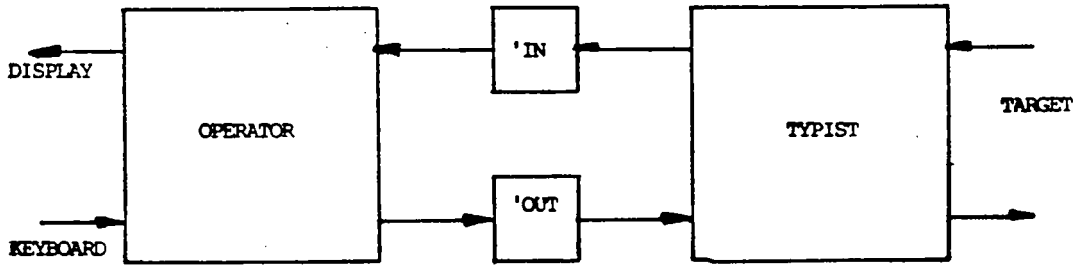
Code 1 - **BLOCK** - Target read followed by a 5 ASCII character block number, 1K data and an ACK/NAK are transmitted by the Host.

Code 2 - **BUFFER** - Target write followed by a 5 ASCII character block number and 1K of data. An ACK/NAK is transmitted by the Host.

Code 3 - **FLUSH** - Ensures that the Host also does a **FLUSH** followed by an ACK/NAK from the Host.

ISD The Host End

Two tasks exist in the Host, one to service the display and keyboard - this is the normal **OPERATOR** task. The second is the normal serial I/O device task **TYPIST**. Communications are via two byte variables **'IN** and **'OUT**.



TYPIST runs a program COMMS which cycles testing two things.

1. If a character has been received by the UART (?KEY is non zero) then either a disk Read/Write is performed or it is sent to 'IN as soon as that variable is empty - this test ensures that the Target does not overrun the speed at which a potentially slow operating system can output to the display.

2. If the variable 'OUT holds a character, it is removed and output to the Target without any tests ensuring that the system cannot lock up even if the Target is in an infinite loop.

```
 0     ( In-Situ Development COMMS task)
 1 : ?BLK ( n)   PAD 7 BLANK  PAD 1+ 5 STRAIGHT  PAD NUMBER ;
 2
 3 : READ   ?BLK  BLOCK  1024 )TYPE ;
 4
 5 : WRITE   ?BLK  PAD 1024 STRAIGHT  BUFFER PAD SWAP 1024 MOVE
 6    UPDATE ;
 7
 8 CREATE 'ACTS   ' EXIT ,  ' READ ,  ' WRITE ,  ' FLUSH ,
 9
10 : .ACK   DISK 2+ @  IF  15 ( NAK)  ELSE  06 ( ACK)  THEN EMIT ;
11
12 : ACT ( n)   2* 'ACTS +  @EXECUTE  .ACK ;
13
14
15
```

```
 0     ( In-Situ Development COMMS task)
 1 CVARIABLE 'IN   CVARIABLE 'OUT
 2
 3 : )OPERATOR ( n)   BEGIN  PAUSE  'IN C@ 0= UNTIL  'IN C! ;
 4
 5 : COMMS   TYPIST ACTIVATE
 6    BEGIN  PAUSE  ?KEY ?DUP
 7       IF  DUP 4 ( IF  ACT  ELSE  )OPERATOR  THEN  THEN
 8       'OUT C@  ?DUP IF  0 'OUT C!  EMIT  THEN
 9    AGAIN ;
10
11 : SERVICE   BEGIN  PAUSE
12       BEGIN  'IN C@  ?DUP WHILE  0 'IN C!  EMIT  REPEAT
13       ?KEY ?DUP  IF  DUP 'OUT C!  27 =   ELSE  0   THEN
14   UNTIL ;
15
```

OPERATOR runs SERVICE which outputs the contents of 'IN to the screen and inputs from the keyboard, if there is a character there, to 'OUT unless it is an ESC in which case control is returned to the Hosts keyboard interpreter.

One additional use of this part of the package is that it allows the Host or PC to be used as a terminal which has numerous applications in other areas.

ISD The Target End

In the Target hardware, it is necessary to produce alternative versions of (BLOCK) and (BUFFER), the words that control input and output of blocks. These words are all written in high level and so are common to all Target CPUs.

```
0      ( Target ISD disc access)
1 : .BLK ( n)   BASE @ )R  DECIMAL  DUP (# # # # # # #) TYPE
2    R) BASE ! ;
3
4 : !ACK   KEY  06 = ( ACK)  0= DISK 2+ +! ;
5
6 : IN-BUFF ( n a)  01 EMIT  SWAP .BLK  1024 STRAIGHT  !ACK ;
7
8 : OUT-BUFF ( n a)  02 EMIT  SWAP .BLK  1024 TYPE  !ACK ;
9
10 : (BLOCK) ( n - a)  ?ABSENT  'BUFFER @EXECUTE
11    2DUP  IN-BUFF  ESTABLISH ;
12
13 : (BUFFER) ( n - a)   ?UPDATED  OVER OUT-BUFF ;
14
15 : FLUSH   FLUSH  03 EMIT  !ACK ;
```

One other change that is made is that the keyboard prompt is changed to Ok so that the user can differentiate between Host and Target system communications.

The production of a Target polyFORTH ROM in the first place is perhaps the most complex part of getting an ISD system working. The standard polyFORTH source provides initialisation code and drivers for the "normal" UART for the particular CPU. If that is the chip set to be used, it is simply a case of setting up ROM, RAM and I/O addresses and replacing the disk drivers with the versions described above. If a "foreign" or new UART or Interrupt structure is to be used, then the first step is to Cross Target Compile on the Host a Target ROM that will communicate normally with a VDU. Once this is running, the serial link drivers may be installed and tested.

The Speed Consequences

Four possible points determine the speed limitations of the system :-

1. Output of the characters to the screen by the **OPERATOR** task this is typically a limitation with MSDOS systems such as

the IBM-PC which is slow.

2. It is assumed that keyboard speed will limit the rate at which values will be input to the Host.

3. Input speed for disk blocks to the Target may be limited by slow CPU's if polled I/O is used (eg 1802).

4. Input to the Host from the Target may be limited by OS problems.

The use of hardware or XON/XOFF character handshakes would have significantly limited the systems that could be linked or complicated writing their drivers and so was rejected.

In the worst case, (MSDOS Host - 1802 Target) all communications could take place without faults at 2400 Baud. This speed is workable but the programmer tends to use the Host for editing and adopts a loading strategy that minimises the number of full system loads adopted.

The experienced FORTH programmer is encouraged to modify the code to best suit any particular hardware. The most useful enhancement to the system is interrupt driven queued input from the Target to COMMS. This queue needs to be 2K bytes deep and then the system can easily handle 9600 Baud links to the Target. At this speed, interaction with the system is not perceptably different to a normal development system and editing takes place on the Target without switching backwards and forwards to the Host.

Conculsions

As major European suppliers of FORTH systems for use in industrial control environments we set out as a requirement of this package that it should both generalize and simplify the procedure of linking a Target SBC with a PC to provide a robust development system. The package described has achieved that objective for the novice FORTH programmer without jeopardizing the expert's ability to optimise the configuration where that is possible. This technique then enables a wide range of applications, that would not otherwise do so, to benefit from FORTHs many advantages.