Connection of a Forth target with a Forth host

Willi Stricker

Springe, Germany,

September, 12, 2012

Basic properties of host target communucation

As most modern micro processors/controllers the STRIP Forth processor [1] is equipped with a special boot program for programming and debugging. It is started by activating a special boot pin while restarting the processor (activating the reset pin).

This program is a very short but very versatile one, it can be used to connect common Forth targets to any (Forth) host in a very universal way. A Computer, for example a PC, controls a target system, for example an evaluation board.

In general the connection needs two programs, one inside the host and one inside the target (the latter is the boot program). Both of them are mostly identical but have some differences.

Demands for the cooperation:

The target should seem to be an integrated part of the host's own system.
The target contains program parts necessary for its own operation only.
All program parts that are used for configuration, management compilation programming etc. of the target instructions are inside the host.
The communication is done by calling target instructions (actual Forth words) by the host.

The complete operation is controlled by the host. The target is connected via an interface with the host (for example a serial interface like RS232 or USB, using the UART format and byte by byte transfer). The host is master, the target is slave. The host contains all auxiliary programs like compiler, interpreter, assembler, editor, disk system. It also contains the headers of the target's functions („name“ and „link“).

Execution of a target instruction by the host

1. The host sends the input parameters to the target,
2. the host sends the target instruction code to the target,
3. the target executes the instruction,
4. the target sends the manipulated output parameters to the host

Note for the parameters:
In a Forth System the parameters are on the parameter stack. The program has no information about the amount of parameters the instruction is needing for input and output. So the whole stack is sent to the target, it manipulates the parameters it needs, and sends the remaining stack back to the host. To minimize the data transfer the amount of parameters can be limited by the host (e.g. 10 or 16) to save time for the transfer.

Working course at execution of a target instruction by the host

host		**Target**
working		waiting for an instruction
sending parameters	=>	getting parameters
sending instruction	=>	getting instruction
waiting for response		executing instruction
receiving parameters	<=	sending parameters
continuing working		waiting for the next instruction

Course of the communucation

Byte order for the parameters: first low byte then high byte, If the count is zero, no parameters are transmitted.

1st byte = start byte, always zero
2nd byte = parameter count
3rd byte = 1st parameter, low byte
4th byte = 1st parameter, high byte
. .
. .
(2n+1)th byte = nth parameter, low byte
(2n+2)th byte = nth parameter, high byte

Transfer order:
The stack is always sent starting from down (bottom of stack BOS).
Conclusion: At sending the stack has to be rolled up from bottom. At receiving the parameters are simply pushed onto the stack.

Auxiliary Instructions and programs for the communication

Receiving and sending one byte from or to the interface hardware

These instructions are actually hardware dependant. It is just a possible example. It is assumed that there are two UART registers, a control register UACON and a data register UADATA. The control register has control bits for receiving selected by a receive mask RECMASK and for transmitting selected by a transmit mask TRAMASK.

Receive one byte:
```
: GETBYTE   ( -> byte )
    BEGIN
      UACON C@
      RECMASK AND
    UNTIL
    UADATA C@
;
```

Transmit one byte:
```
: PUTBYTE   ( byte -> )
    BEGIN
      UACON C@
      TRAMASK AND
    UNTIL
    UADATA C!
;
```

Handshake

A handshake is provided, it is mandatory because the time delay of the instruction execution of the target is unpredictible. Here it is done by software. a hardware handshake is basicly possible but not recommended because most interfaces don't provide a hardware handshake (e.g. evaluation kits).
The host (master) sends a byte and waites afterwords for another one of the target even if it expects one it has to send a dummy first. The target (slave) waites for a byte from the host and sends one afterwords even if it has to send one.

Host sends a byte with handshake:
```
: PUTCHAR ( char -> )
    PUTBYTE
    GETBYTE ( dummy = sent byte )
    DROP
;
```

Host receives a byte with handshake:
```
: GETCHAR ( -> char)
    0
    PUTBYTE ( dummy = 0 )
    GETBYTE
;
```

Target sends a byte with handshake:
```
: PUTCHAR ( char -> )
    GETBYTE ( dummy = 0 )
    DROP
    PUTBYTE
;
```

Target receives a byte with handshake:
```
: GETCHAR ( -> char )
    GETBYTE
    DUP
    PUTBYTE ( dummy = received byte )
;
```

Program for sending the parameters

```
: PUTPAR    ( n Parameter -> )
    0 PUTCHAR       send start byte
    SP@ DUP         parameter count
    PUTCHAR         send count
    BEGIN           loop for sending  the n Parameters
      DUP           count
```

```
    WHILE
      DUP
      PICK          pick next parameter
      DUP PUTCHAR   send low byte
      CSWAP         swap bytes
      PUTCHAR       send high byte
      1-            decrement count
    REPEAT
    DROP
    0 SP!            stack initialisation
;
```

Program for receiving the parameters

```
: GETPAR ( -> n Parameters)
    BEGIN
      GETCHAR 0=    receive start byte + test
    UNTIL
    GETCHAR         receive parameter count
    BEGIN           loop for receiving n parameters
      DUP           count
    WHILE
      GETCHAR       receive low byte
      GETCHAR       receive high byte
      CSWAP OR SWAP concatenate to one word
      1-            decrement count
    REPEAT
    DROP
;
```

The instructions GETPAR and PUTPAR are identical for host and target, only the handshake instructions GETCHAR and PUTCHAR are different.

Target communucation program

The target uses a minimum Forth kernel, that at least contains the instruction for the communucation.
The program is an indefinite loop, reacting at demand by the host only.
The target obviously needs an initialisation of the Interface hardware, that sets the mode of the interface „PORTINIT“. This program is completely hardware dependant.

```
: COMMUNIC ( -> )
    PORTINIT ( -> )  port initialisation
    BEGIN            indefinite loop
      GETPAR         receive the parameters + instruction (cfa)
      EXECUTE        execute the instruction
      PUTPAR         return the resulting parameters
    AGAIN            return always
;
```

Note:This is the boot program of the STRIP Forth processor [1].

Construction of a target instruction in the host program

The target instruction in the host is made of a standard header (name and link) followed by an instruction (cfa of a Forth word) whose name is DOTARGET and followed by the code address of the target instruction (inside the target). Before target instructions can be accessed by the host, the interface has to be initialized.

| Haeder (target instruction name, link) |
| DOTARGET |
| target instruction address |

The host instruction „DOTARGET“ has the following definition:

```
: DOTARGET
    R>      get memory address from return stack
    @       get cfa of target instruction
    PUTPAR  send parameters to the target – instruction execution
    GETPAR  receive parameters from the target – result
;
```

Reference:

[1] Willi Stricker:
„A Processor as Hardware Version of the Forth Virtual Machine“; EuroForth 2011proceedings.