An Embedded Forth Environment for a Programmable Bus Interface

Michael Pandolfo
Stratus Computer, Inc.
55 Fairbanks Blvd.
Marlborough, MA 01752

Abstract

The Programmable StrataBus Interface (PSI) is a board that mediates high speed data transfer between a customer-supplied peripheral and the duplexed, fault-tolerant bus of a Stratus computer module. Forth-83 was implemented for the PSI in order to replace the existing resident debugger with a more flexible facility. It resides in a 64K portion of memory and runs as a separate task.

It should come as no surprise that the interactive, extensible nature of Forth provides the PSI programmer with more power than the precompiled debugger. Additionally, Forth has proven to be a natural fit for the I/O architecture of the board. With the Forth subsystem, the ease in which new functions can be defined has given the PSI an ad hoc control capability; programmers can now direct hardware functions as easily as they have controlled software in the past.

Rational

Simply stated, the raison d'etre of the PSI is found within its name: the StrataBus. In addition to a patented transfer protocol, what makes the StrataBus unique is its method of ensuring data integrity. The bus is duplexed, and each side is checked for validity at the clock rate of eight megahertz. Upon detection of an error, the faulting bus of the pair is immediately shut down, and the partner bus is used until the system determines that the cause of the fault no longer exists. Likewise, each hardware component in a Stratus module is actually a pair of identical boards running in lockstep. Only if both boards simultaneously fail will the component be unavailable. A board not designed precisely to these data integrity requirements can cause complete system failure. The PSI is provided by Stratus to reduce this design requirement on third parties and to eliminate the need to design for the timing of the StrataBus.

Description

The PSI has two separate interfaces: one to the module's StrataBus and one that addresses up to four peripheral devices. Each interface supports programmed I/O and DMA. Sixteen kilobytes of high-speed RAM serves as a transfer buffer between the two DMA channels. The StrataBus DMA interface can transfer up to 2 megabytes of data per second. The PSI bus DMA allows up to 2 million transfers per second. Programmed I/O to VOS (the Stratus Virtual Operating System) memory is performed through a memory window under program control. Programmed I/O to a peripheral is performed using one of 128 PSI addressable locations.

All board functions are mediated by an on-board 68000 processor. It is possible, indeed desirable, to have two DMAs and one program I/O running at the same time. In its simplest and most efficient use, the processor initiates DMAs at the request of VOS or the peripheral, sets flags and threads buffers in VOS memory, and coordinates the arbitration protocol for the PSI bus.

The rest of the PSI hardware complement consists of 256K program memory, a timer circuit, bus drivers/receivers, and comparators. The latter implement board level error detection: circuitry is duplicated, and comparators constantly monitor the signals from both sides of the board. If any signal fails to match, the board immediately shuts itself down and is "red-lighted," preventing any data from leaving the failed board. Fault tolerance is provided by a second PSI board that was configured and instructed to execute in lockstep with the first: this second board continues the processing without taking any special action.

Firmware for the PSI is downline-loaded from VOS. The firmware is written using a standard VOS language translator (e.g., PL/I or Pascal), and is supported by a subroutine library which provides various functions such as task management, timer services, and DMA initiation. The firmware program may consist of from one to twenty interruptable, prioritized tasks. The PSI is also supplied with a debugging task that can be run without modification.

Role of Forth as a Debugger

The initial reason for implementing Forth was that the supplied PSI debugger is relatively primitive. It has requests for reading and writing memory and registers, for tracing the stacks of configured tasks, and for managing breakpoints. In addition, the debugger has minimal a symbolic reference capability. In the course of programming the PSI for timing studies, the author found that he was frequently adding new requests. A completely new firmware

image was required to implement even the most minor change in the behavior of the debugger. With the addition of Forth as a background task, new functions are now added nearly as easily as they can be conceived, and certainly as quickly. Debugging the board is made easier because newly found bugs can be analyzed with equally newly created functions.

Description of Forth Implementation

Forth-83 was implemented for the PSI utilizing 64K of program memory. It conforms to the language described by the FORTH Standards Team, dated August, 1983. In addition to the Double Number Extension Word Set, it contains a PSI Extension Word Set, described in Table 1. The device layer uses the services of VOS to complete both keyboard and disk I/O processing. Disk I/O is performed by requesting a read or write of a VOS file record. Terminal output is implemented by writing a string into the VOS-resident PSI mailbox, which is received by a VOS process. Terminal input is similar: another location in the PSI mailbox is filled by a VOS process and read in its entirety by a PSI task, which feeds the individual characters to Forth during KEY invocation.

!ABS	16b 32b --	Store 16b at absolute address
@ABS	32b -- 16b	Fetch 16b from absolute address
C!ABS	16b 32b --	Store byte at absolute address
C@ABS	32b -- 16b	Fetch byte from absolute address
D!ABS	32b 32b --	Store 32b at absolute address
D@ABS	32b -- 32b	Fetch 32b from absolute address
DICTIONARY-ORIGIN	-- 32b	Leave absolute address

Table 1.
PSI Extension Word Set

After producing a working version of the system, its size was reduced from 64K to 32K, so that it would require roughly as much space as the original debugger. The Forth environment runs as a separate task in the PSI, with a skeleton of the original debugger serving as the event handler as well as the KEY interface.

Transcending the Debugger

After using Forth as a debugger for a while it became apparent that Forth could provide a means to manipulate the I/O circuits on the PSI as easily as the utility subroutines did. The main characteristic of the PSI that aids in this is that all on board devices, registers, and control bits are memory mapped into the 68000's address space. Several

screens of words provide nearly all the functional capability found in the conventional subroutine library.

A clear example of the advantage Forth provides can be found in Figures 1 and 2. Figure 1 compares the steps necessary to read VOS location 0026610A, which is the start of the PSI control block. Whereas the debugger requires the programmer to juggle registers and addresses, DUMP-VOS needs simply a VOS address and a byte count. Figure 2 displays the Forth code used for the example. Especially important is the ability to debug and change (Forth) code completely interactively: in this way, the board's DMA, PIO, and VOS window circuitry are immediately available for manipulation, just as the switches on an older style processor were. When implementing a new peripheral for the PSI, the interface to the new device can be dynamically configured and tested. PSI Forth has been changed from a software debugging tool to a rapid prototyping aid.

```
set.w FFFF8006 266
dump 1C010A 20
1C010A  00010304 01000400 00008000 00000000
1C011A  0021E41A 0021E000 00030000 00008000

0026 610A 20 DUMP-VOS
001C010A  0001 0304 0100 0400 0000 8000 0000 0000
001C011A  0021 E41A 0021 E000 0003 0000 0000 8000
```

Figure 1.
Reading VOS Memory

```
scr#   15
  0  (  WORDS TO READ VOS MEMORY 86-03-24 MAP )
  1
  2  HEX
  3  0 S>D 2CONSTANT RAM          ( RAM START: 00000000 )
  4  0 1C 2CONSTANT WINDOW        ( ..INTO VOS PAGE: 001C0000 )
  5  8006 S>D 2CONSTANT WBR       ( WINDOW BASE REG: FFFF8006 )
  6
  7  : SET-WBR                    ( HIGH LOW -- )
  8    C 0 DO 2/ LOOP F AND
  9    SWAP 10 * + WBR !ABS ;
 10
 11  : DUMP-VOS                   ( HIGH LOW COUNT -- )
 12    >R                         ( SAVE COUNT )
 13    DUP FFF AND >R             ( SAVE OFFSET )
 14    SET-WBR WINDOW R> S>D D+ R> DUMP ;
 15
```

Figure 2.
Screen for Reading VOS Memory