STIRLING MOS
SINGLE CHIP SPECTRUM PROCESSOR

Systems Design & Integration Department
Measurement Systems Center
HP Laboratories

INTRODUCTION

The 1LP1 CMOS CPU is a single chip central processor unit in Hewlett Packard’s Spectrum processor family. The processor, with an integral translation lookaside buffer and instruction cache, has been designed in HP’s CMOS40 process to provide a low cost, high performance processor engine for various products such as personal computers, single board microcomputers, instrumentation controllers, and workstations.

The development of this chip has been a cooperative effort involving HPL, as the designing entity, the Northwest Integrated Circuits Division (NID) as the integrated circuit process and package developer, and the Entry Systems Operation of the Information Technology Group as the customer.

OBJECTIVES

* A Complete Implementation of the Level 1 Spectrum Architecture.
* Greater than 1.5 Spectrum MIPS.
* Low Cost
* Surface Mount Packaging

FEATURES

* 8 MHZ Peak Instruction Rate.
* 48 Bit Virtual Address Range
* Multiplexed Address & Data Bus (27b Address, 32b Data).
* 64 MByte Maximum Physical Memory.
* 64 MByte I/O Space.
* On-Chip Translation Look-aside Buffer, 2-way Set-Associative by 32 Entries.
* On-chip Instruction Cache, 256 Bytes.
* 5-Stage Pipeline.
* Special Function Unit & Coprocessor Interface.
* Extensive Interrupt System.
* Boot-up Execution from an 8 bit Wide ROM.
* Diagnostic Interface Port for Test & Debug
* 5 Volt Power Supply.
* 1 Watt Dissipation.
* TTL Compatible Inputs & Outputs.
* 120pF Address/Data Drive Compatibility.
* 84 Pin Surface Mount Package.

GENERAL DESCRIPTION

A block diagram of the Stirling Processor chip, partitioned into ten functional units connected by major interunit buses is shown in Figure 1. The physical floorplan of the chip is shown in Figure 2.

The processor architecture is general register based, with thirty-two 32-bit general registers implemented in the R_unit. This general register file is capable of performing two reads and two writes in one cycle, with up to four different register addresses. The three register Spectrum ALU instructions are reflected in a three bus organization for the main execution datapath. The B B and B X are the two operand buses from the general register file (R_unit), supplying operands to the execution unit (E unit), and other units. The B RES is the result bus, carrying the result to be written back to the general register file. This internal
bus organization, with full 32-bit internal datapaths, allows the basic single cycle execution of all arithmetic, logical, extract, deposit and some system control instructions. In general, any instruction whose execution phase involves only the E unit, the Space Register (SR) unit and the Control Register (CR) unit has an effective execution time of one cycle.

The other major interunit buses are related to memory accessing functions which involve the Memory Management (MM) Unit, the Instructional Fetch (IF) unit and the Bus Interface (BI) unit. The B_VA (Virtual Address) bus carries either the Instruction Address calculated in the IA unit, or the Data Address calculated in the E unit, to the MM unit to read the 64-entry on-chip Translation Lookaside Buffer (TLB), for the virtual to physical address translation and protection checking. For a load or store instruction, the resultant physical address is sent via the B_PA (Physical Address) Bus to the BI unit, where an external bus transaction will be initiated for a memory read or write operation.

For an instruction fetch, the B_VA bus carries the instruction address generated in the Instruction Address (IA) unit to the IF unit, where the 256-byte on-chip Instruction Cache (I-Cache) is read simultaneously with the TLB. The I-Cache tags (C_TAG) are sent from the IF unit to the MM unit which checks for TLB hit and I-Cache hit in parallel. If both the TLB and I-Cache hit, then the desired instruction is sent via the NEWIR lines from the IF unit to the C unit, for instruction decoding and overall control signals generation.

If there is an I-Cache miss, the physical address on the B_PA bus is used by the BI unit to initiate a quad-word line fetch from the main memory for the missing I-Cache line. The new I-Cache tag is written from the Physical Address Latch (PARL) during the cachefill operation.

**PIPELINE OPERATION**

The pipeline may be described as a five-stage synchronous pipe:

```
| IA | IF | DR | E | W |
```

where

- **IA** = Instruction Address calculation stage
- **IF** = Instruction Fetch-stage
- **DR** = Decode and Read GR-stage
- **E** = Execute-stage
- **W** = Write-stage

55-5-3
During the IA-stage, the next sequential address or the branch target address is calculated in the IA_Unit. At the end of the IA-stage, the Program Counter (PC) is updated. (In some pipeline nomenclature, the IA-stage is not explicitly singled out as a pipe stage, and this pipeline would be described as a four-stage pipe.)

At the beginning of the IF-stage, the correct instruction address is sent via the B_VA bus to both the on-chip Instruction Cache (I-Cache) in the IF_unit or the on-chip Translation Lookaside Buffer (TLB) in the MM unit to begin a simultaneous check for TLB hit, protection validity and I-Cache hit for the instruction being fetched. If all three conditions are met, then the instruction read from the I-Cache is loaded into the Instruction Register (IR) at the end of one cycle. If a TLB miss or protection validity check fails, then an I-Fetch interrupt is saved by the Interrupt Controller. This generates an Interrupt Acknowledge stage in place of this instruction’s W-stage. If an I-Cache miss occurs, then a cachefill sequence is initiated with a quad-word read from main memory. If both an I-Fetch interrupt and an I-Cache miss occur, the I-Fetch interrupt takes precedence. In all three cases, the instruction requested is loaded into the IR at the end of the IF-stage.

During the DR stage, the instruction in the IR is decoded in the C_unit. The decoded signals are latched into a pipeline register at the end of the DR-stage. Simultaneously, the two general registers, specified by fixed fields in the instruction, are also read out from the General Register (GR) file in the R_unit and latched into the pipeline registers, XE and BE, at the end of the DR-stage.

At the beginning of the E-stage, the XE and BE registers, or the appropriate GR-bypasses are dumped onto the internal B_X and B_E busses. The ALU or Shifter/Mask-Merger in the E_unit is activated during this stage for arithmetic, logical and extract/deposit instructions. Reading of the Space Register file in the SR_unit, the Control Register file in the CR_unit or the Processor Status Word (PSW) in the C_unit is also done during this stage. In the absence of pending interruptions, interlocks or nullifications, the writing of the Space Register file, the Control Register file or the PSW occurs at the end of the E-stage. Any data or results to be written into the General Register file are latched at the end of the E-stage into pipeline registers.

At the beginning of the W-stage, again assuming no interruptions, interlocks or nullifications, the data to be written into the GR file is transferred to the "WA" write port of the GR file via the B_RES bus. It is during the W-stage that the contents of the B_RES bus are actually written into the selected general register.
Because of this delayed write, a GR bypass feature is implemented whereby the result (of write GRi) available at the end of one instruction's E-stage may be used as an operand (of read GRi) for the next instruction's E-stage. This is equivalent to tri-stating the B.RES bus directly onto either the B.X or B.B bus at the beginning of each E-stage. A similar GR bypass has also been implemented for the "WL" write port of the GR file. This write port is used mainly for data to be loaded into the GR file from the external memory.

The pipeline is designed so that all pipeline variations occur in the W-stage. The W-stage can vary in length from one to many cycles. Figure 3 gives some examples of variations in the W-stage for different instructions. The decision as to which W-stage to go into is made during an instruction's E-stage. For multicycle W-stages, the pipe controller freezes processing except for those units executing the multicycle W-stage. At some setup time before the end of the last cycle of the W-stage, the pipe controller unfreezes the rest of the pipe.

When the pipe is full, or in steady state, there are five different instructions being executed with one in each stage of the pipe. For sequential instructions, while instruction n is in its W(n)-stage, instruction n+1 is being executed in its E(n+1)-stage and deciding which W(n+1) stage to go into next. Instruction n+2 is in its DR(n+2)-stage. Instruction n+3 is in its IF(n+3)-stage and instruction n+4 is in its IA(n+4) stage. Any IF (n+3) cachemiss would cause further suspension of the pipe, elongating the W(n)-stage. The situation for branch instructions is illustrated in figure 3 (d).

For multicycle instructions, or instructions with multicycle W-stages, the synchronization of the pipe stages is still maintained. That is, when an instruction has a multicycle stage, all the other instructions in the pipe are suspended in their current pipe stage until the multicycle stage is done. Hence, every instruction enters and leaves each pipe stage in the same order that it first entered the pipe. This design philosophy simplifies the pipeline control.

Another underlying pipeline design philosophy is to employ prevention rather than cure. That is, the Control State Machine and the pipeline controller do not issue the start of potentially destructive operations that may require later abortion and the saving of states for back-ups or restorations. By allocating all potentially destructive operations of an instruction in its W-stage, preventing the W-stage execution is equivalent to never having the instruction executed.

Other than the steady-state, full pipe operation described above, the pipeline controller also performs functions such as emptying
the pipe for interrupt acknowledge stages, and filling the pipe for bootup, reset and interrupt handler fetching.

A certain amount of overlap between internal pipeline execution and external bus transactions is also designed into the system to improve performance. The pipeline remains frozen for bus transactions only until it has determined that the rest of the transaction will proceed without further interruptions.
STIRLING BLOCK DIAGRAM

Note: All data paths are 32 bits wide, unless explicitly labelled otherwise.

Figure 1
Figure 3: Examples of Stirling Pipeline Stages

(a) Normal 1-cycle W-stage (e.g. ALU class):

<table>
<thead>
<tr>
<th>IA_n</th>
<th>IF_n</th>
<th>DR_n</th>
<th>E_n</th>
<th>W_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA_n+1</td>
<td>IF_n+1</td>
<td>DR_n+1</td>
<td>E_n+1</td>
<td>W_n+1</td>
</tr>
<tr>
<td>IA_n+2</td>
<td>IF_n+2</td>
<td>DR_n+2</td>
<td>E_n+2</td>
<td>W_n+2</td>
</tr>
<tr>
<td>IA_n+3</td>
<td>IF_n+3</td>
<td>DR_n+3</td>
<td>E_n+3</td>
<td>W_n+3</td>
</tr>
<tr>
<td>IA_n+4</td>
<td>IF_n+4</td>
<td>DR_n+4</td>
<td>E_n+4</td>
<td>W_n+4</td>
</tr>
</tbody>
</table>

(b) Multicycle W-stage (e.g. Store):

<table>
<thead>
<tr>
<th>IA_n</th>
<th>IF_n</th>
<th>DR_n</th>
<th>E_n</th>
<th>T_n</th>
<th>ADR_n</th>
<th>W_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA_n+1</td>
<td>IF_n+1</td>
<td>DR_n+1</td>
<td>E_n+1</td>
<td>T_n+1</td>
<td>ADR_n+1</td>
<td>W_n+1</td>
</tr>
<tr>
<td>IA_n+2</td>
<td>IF_n+2</td>
<td>DR_n+2</td>
<td>E_n+2</td>
<td>T_n+2</td>
<td>ADR_n+2</td>
<td>W_n+2</td>
</tr>
<tr>
<td>IA_n+3</td>
<td>IF_n+3</td>
<td>DR_n+3</td>
<td>E_n+3</td>
<td>T_n+3</td>
<td>ADR_n+3</td>
<td>W_n+3</td>
</tr>
<tr>
<td>IA_n+4</td>
<td>IF_n+4</td>
<td>DR_n+4</td>
<td>E_n+4</td>
<td>T_n+4</td>
<td>ADR_n+4</td>
<td>W_n+4</td>
</tr>
</tbody>
</table>

(c) Multicycle W-stage with WL-substage (e.g. Load):

<table>
<thead>
<tr>
<th>IA_n</th>
<th>IF_n</th>
<th>DR_n</th>
<th>E_n</th>
<th>T_n</th>
<th>ADR_n</th>
<th>D_n</th>
<th>WL_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA_n+1</td>
<td>IF_n+1</td>
<td>DR_n+1</td>
<td>E_n+1</td>
<td>T_n+1</td>
<td>ADR_n+1</td>
<td>D_n+1</td>
<td>WL_n+1</td>
</tr>
<tr>
<td>IA_n+2</td>
<td>IF_n+2</td>
<td>DR_n+2</td>
<td>E_n+2</td>
<td>T_n+2</td>
<td>ADR_n+2</td>
<td>D_n+2</td>
<td>WL_n+2</td>
</tr>
<tr>
<td>IA_n+3</td>
<td>IF_n+3</td>
<td>DR_n+3</td>
<td>E_n+3</td>
<td>T_n+3</td>
<td>ADR_n+3</td>
<td>D_n+3</td>
<td>WL_n+3</td>
</tr>
<tr>
<td>IA_n+4</td>
<td>IF_n+4</td>
<td>DR_n+4</td>
<td>E_n+4</td>
<td>T_n+4</td>
<td>ADR_n+4</td>
<td>D_n+4</td>
<td>WL_n+4</td>
</tr>
</tbody>
</table>

(d) Multicycle W-stage for Branch instructions:

<table>
<thead>
<tr>
<th>IA_n</th>
<th>IF_n</th>
<th>DR_n</th>
<th>E_n</th>
<th>W_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA_n+1</td>
<td>IF_n+1</td>
<td>DR_n+1</td>
<td>E_n+1</td>
<td>W_n+1</td>
</tr>
<tr>
<td>IA_n+2</td>
<td>IF_n+2</td>
<td>DR_n+2</td>
<td>E_n+2</td>
<td>W_n+2</td>
</tr>
<tr>
<td>IA_t</td>
<td>IF*</td>
<td>DR*</td>
<td>IA_*+1</td>
<td>IF*+1</td>
</tr>
<tr>
<td>IA_*+2</td>
<td>IF*+2</td>
<td>W_n+3</td>
<td>W_n+4</td>
<td></td>
</tr>
</tbody>
</table>

where *=t or n+2, depending on whether the branch was taken or not.