Chapter 8

Basava Technology – a software and hardware standard for a Processor Enhanced Memory Module
8.1 Introduction

In this chapter, we discuss a novel approach called Basava Technology for realizing speech recognition systems. It addresses the challenges in the area of real-time processing with minimal resources and efficient system design.

Basava Technology is an open industry standard for adding one or more DSPs (processors) in standardized modules. It covers hardware, software and system issues related to adding parallel processing in the memory space of a main processing unit (MPU). It became an Electronic Industry Alliance of Japan (EIAJ) in 1997 and a Joint Electron Device Engineering Council (JEDEC) standard in 1998. EIAJ consists of major electronic suppliers in Japan and Korea with representative companies such as Mitsubishi, NEC, Toshiba, Samsung, etc. JEDEC is a worldwide body consisting of companies from across the world with representative companies such as IBM, Compaq, etc. Both EIAJ and JEDEC accepted and standardized the electrical interface, referred to as Processor Enhanced Memory Module (PEMM). IEEE has authorized a Study Group for standardizing the software interface.

8.2 Problems solved by this standardized approach

Inserting a processor in the main memory space of a host processor without impacting its performance poses several challenges. With memory speeds typically at 100MHz and increasing, host processor and board designers have to optimize the memory bus and pay special attention to bus capacitance, data line length and so on. Inserting a processor in this critical path such that it is transparent to the host performance needs to ensure that these considerations are not violated. In this chapter, we discuss approaches that ensure that these considerations are not compromised.
In addition, how do you detect the presence of the processor(s) in the memory space? How do you identify the physical addresses at which the processor or processors are located? How can you guarantee that the host processor’s operating system (OS) does not page out the physical memory locations assigned to the processor? How do you control the processor i.e. how do you download data and program to the processor? How do you signal when a task is completed especially when an interrupt signal is not available on the memory bus? Finally, what is the benefit of this approach? What applications are speeded up and by how much?

8.3 Motivation

There are several different reasons for proposing this at standard. These are first enumerated and explained in detail further.

Mismatch between processor and memory speeds.

Inadequate Bandwidth. Multimedia applications are limited by computational speed, memory size and bandwidth.

Hardware upgrade. There is no simple or standardized method of scaling processing power.

Many applications in signal processing, speech, and image processing are memory intensive, well structured, and amenable to partitioning. Small operations or routines are frequently and iteratively applied to large sets of data. Euclidean distance calculation is one such example.

Processor technology has focused on improving raw processing speed. As an example, the mega hertz (MHz) of the latest member of Texas Instruments Inc. processor is about 600 MHz compared to 5MHz of the first generation processor. As long as the computations are on-chip, these devices provide adequate throughput. However, several applications in speech, signal, and image processing are memory intensive and the gain in raw processing speed is lost when the processor has to fetch and process data from slower off-chip memories.
Memory technology, on the other hand, has focused on increasing the density of storage bits. As an example, consider the number of memory devices required to store a 480x240 sixteen-bit image. With increasing device densities, the number of chips required to store the image reduces, a desired effect. However, as shown in preceding table, the maximum bandwidth suffers. Dense memories defeat multiple bus architectures.

<table>
<thead>
<tr>
<th>Density</th>
<th>64K</th>
<th>256K</th>
<th>1M</th>
<th>4M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory technology</td>
<td>8K x 8</td>
<td>32K x 8</td>
<td>128K x 8</td>
<td>512K x 8</td>
</tr>
<tr>
<td>Number of memories to store 480 x 240</td>
<td>29</td>
<td>8</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>Maximum bandwidth 8-bit data bus/IC 20-ns access time (Mbytes/s)</td>
<td>1450</td>
<td>400</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Time to read image (ms)</td>
<td>0.16</td>
<td>0.58</td>
<td>2.3</td>
<td>4.6</td>
</tr>
</tbody>
</table>

The combined effect of increasing processing speed of processors and increasing density of memory devices is further aggravating the CPU-to-memory bandwidth, a paramount issue in computer system design. Figure 8-1 shows the processor and memory technology trends over the last several years.
8.4 Basava Technology – What is it?

Basava Technology is an open industry standard for adding one or more processors in standardized modules. Basava concept is a new computer architecture and system concept that enables an user to easily enhance the system performance and add new functions by just plugging in a memory module with one or more processors embedded in the module while supporting downloadable, multiple functions software.

This concept is shown in Figure 8-2. The host processor addresses a large memory space that is usually divided amongst several memory banks or modules. These modules are standard memory modules that are standardized by worldwide bodies such as JEDEC. In one of these modules, a processor is embedded and such a module is referred to as a Processor-Enhanced-Memory-Module (PEMM). This is shown in Figure 8-3.

![Figure 8-2 Basava Concept](image)

**Legends:**
- MPU: Main Processing Unit
- DSP: Digital Signal Processor

Figure 8-2 Basava Concept
8.5 Benefits of Basava Technology

There are several benefits to having a memory-mapped processor embedded in a standardized module. These benefits are listed below and explained later in this section.

Simultaneous, High Bandwidth Memory access by main processor and embedded processor without contention.

Processor independent memory module

Easily scalable – just add more modules

Coprocessor in main memory has highest bandwidth with main processor.

Local processing in module further increases total system bandwidth

Bank switching for streaming Audio/Video decode

8.5.1 Simultaneous, High Bandwidth Memory access by main processor and embedded processor(s) without contention

As shown in Figure 8-4 the processor in the memory module has access to a portion of the physical memory available. A separate, decoupled memory bus allows on module processes to be invisible externally. While the embedded processor is working on one bank of the memory module, the host processor can work on the other memory bank in parallel without contention. However, host processor
accesses to the same memory bank require bus switching as shown in Figure 8-4.

![Figure 8-4 Architecture Overview](image)

### 8.5.2 Processor independent memory module

Depending on the type of application, one can populate the module with the appropriate processor. The design and operation of the module from the host processor is independent of the type of processor populated. For example, one can utilize a RISC processor or a Digital Signal Processor (DSP) on a control processor. The operation and architecture of the communication does not change. This benefit is shown in Figure 8-5.
8.5.3 Easily scalable – just add more modules

It is very easy to scale the processing power of the system depending on application need. As shown in Figure 8-6, just add another module if more processing power is needed. This assumes that the host processor has access to a large memory address space, which is supported by multiple modules.
8.5.4 Coprocessor in main memory has highest bandwidth with main processor

In most computer systems, the memory bus is separated from the peripheral bus that has additional signals for control and other signaling schemes. These peripheral buses have been standardized either by some companies or by an industry consortium. For example, the ISA bus, EISA bus and PCI bus, and now USB have been driven by Intel. The EISA bus supports 32bit transfers at a rate of 8MHz; the PCI can support 32bits at 33MHz. As shown in Figure 8-7, a processor in the memory space will have access to the highest bandwidth possible; for example, 64bit transfers at rates in excess of a 100MHz.
8.5.5 Local processing in module further increases total system bandwidth

Consider the multiplication of a matrix by a vector, which is typical in many signal-processing algorithms. Let B be a 100x100 matrix and A be a 1x100 vector. When this algorithm is implemented on a typical host processor with access to a standard memory module, as shown in Figure 8-8, the bus traffic is 10,000 and from memory module to the processor. However, in the case of a module with a processor, the direction of traffic is from the host processor to the memory module, and the bus traffic is 100, only a small fraction of the previous bus traffic. In addition, the host processor is now freed to do additional tasks, while the module processor is processing the vector. In addition to reduced traffic on the bus, freed cycles on the host processor, the overall power consumed by the system is also reduced since the bus traffic is reduced. This is because data movement on the bus leads to increased switching of the
bus, which has its own capacitance. However, when the data movement is localized to the module and removed from the main bus, switching is reduced and thereby the power as well.

![Diagram](image)

Figure 8-8 Local Processing Further Increases Total System Bandwidth

**8.5.6 Bank switching for streaming Audio/Video decode**

Streaming audio and video decoding applications are increasingly become popular with the proliferation of the Internet. These applications require large bandwidth, memory and processing power, which are inherently available in the memory module with one or more embedded processors. With bank switching feature supported as shown in Figure 8-9, the embedded processor can be decoding the data in one bank while the host processor is streaming data or frames into the other bank.
8.6 Architecture of PEMM Processor

![Diagram of PEMM Processor Architecture]

Figure 8-9 Bank switching for streaming Audio/Video decode

![Diagram of DSP-Enhanced Memory Module Block Diagram]

Figure 8-10 DSP-Enhanced Memory Module Block Diagram (1 Bank Shown)
The block diagram for the DSP Enhanced Memory Module, also referred to later as a Processor Enhanced Memory Module (PEMM), is shown in Figure 8-10. Much like a standard module, the PEMM also contains standard DRAMs (EDO or SDRAM) which are connected to the PC’s memory bus via the DIMM card edge. In addition to these memories, this module also contains crossbars, a Bus Controller IC (ASIC), SRAM for the DSP, and the DSP itself.

The PEMM may be configured for registered or non-registered access, depending upon the system application.

A serial EEPROM is also included on the PEMM. This EEPROM, used today for Serial Presence Detect (SPD), contains data describing the DRAM memory type and timing which can be read by the host system at power-up. On the PEMM, this EEPROM will also be used to detect the presence of the DSP, its local memory size, speed, and other pertinent information.

Figure 8-10 also shows the structure of the data busses on the module. The memory bus, located at the card edge is connected to the Shared Bus via crossbars which are used to isolate the PC’s memory bus from this bank of the module when the DSP is accessing the shared DRAM. These crossbars may be located internal or external to the Bus Controller IC.

A secondary bus, the DSP Local Bus, is also present on the module. This bus is completely decoupled from the CPU bus, allowing the DSP to access the local SRAM at any time without being in contention with the CPU for shared memory. The local SRAM is optional, in that it is included for optimal performance only, and can be removed for issues of cost.

The Bus Controller has the task of controlling all traffic on the module. Specifically, it has the following duties:

Provide isolation (or control of isolation) from the PC’s memory bus to the shared DRAM.

Monitor PC bus activity at all times, and generate the MWAIT signal when appropriate.
Provide address and data mapping from the shared DRAM to the DSP.
Contain the host-to-DSP interface, as well as all status registers.
Generate the MIRQ interrupt signal to interrupt the host at the DSP's request.

8.7 Theory of Operation

The PEMM operates in three modes.

1. Standard Mode
2. Configuration mode
3. Smart mode

8.7.1 Standard Mode

In the standard mode the DSP is reset and the DSP is completely off the bus (i.e. -HOLDA signal is asserted low) giving the PC low power optimized access to the memories. This is the default mode that occurs when the card is plugged in. In this mode, the card appears to the PC and is used by the PC as a standard memory card only. The on-board DSP does not become active until specifically done so by the host PC. The size of the memory available on the card is specified by the Card Information Structure (CIS) in the attribute memory. (Note that the Card Configuration Registers in Attribute Memory are available in this Mode.)

8.7.2 Configuration mode

The Configuration Mode is entered when the module is accessed at a pre-defined location (SPD defined) with a specific data signature. After this signature has been successfully received, DSP setup registers are mapped into the module's memory space, and the host may configure the DSP (and SRAM) with program and data information.

8.7.3 Smart Mode

In the smart mode, the DSP may be activated and share memory with the PC. In this mode, smart mode registers
become active and available to PC and DSP. The first 16 bytes of common memory are remapped into physical locations in the ASIC for the PC (PC Smart Mode Registers or PCSMMR’s). PCSMMR’s are DSP control and status registers that become available to the PC. Meanwhile DSP Smart Mode Memory Registers which are control and status registers to the DSP become available to the DSP as 16 words in DSP I/O space. Communication between DSP and PC may be done via a set of other SMMR’s called Host Communication Registers (DSPTXD, DSPRXD) as either interrupt or polling driven. When PC accesses PCSMMR’s, the DSP operation is not halted, as no bus arbitration is needed during these accesses. There are also other bus arbitration optimizing features that may be used depending on the application.

8.7.4 Switching from Standard Mode to Configuration and Smart Modes

It is very important to ensure that the DSP on the memory module is not inadvertently turned on. This is ensured by requiring that the host processor write a predetermined signature pattern, a specified number of times, successively to the same specific physical address. When the PC writes a DSP signature pattern, 0xA320 (Attention 320), four times consecutively to the SIGR, the DSP is activated and card is switched from standard mode to the smart mode. Note that A320 is a valid X86 instruction and may occur as part of the normal PC operation. For the signature to be valid, PC must write the same pattern to the same location in succession without any other reads or writes to any other location in the PC card. Also note that address 400h functions as normal common memory location, though monitored. Once the first signature code is detected, any of the following actions by the PC (before the signature stream is completed), disqualifies the stream from being valid.

1. Any read from any valid address on the card
2. Any write to any address other than the Signature address) on the card
3. A write to Signature address with another pattern.

Once a valid signature is detected, the MODCTL bit is set in the DSP Control Register. The card is switched from standard mode to smart mode and the first 16 bytes of PC's common memory is remapped into the ASIC. Any subsequent PC access to this block of common memory is redirected into the ASIC. To get out of Smart Mode a 0 should be written to the MODCTL bit. An alternate method of switching in and out of Smart Mode is to use the MODE bit in the CCSR located in Attribute Memory. Writing a 1 will cause the card to switch into Smart Mode while a 0 will switch back to Standard. Note that these two methods are OR'ed together, so either one active in Smart Mode will supersede the other in Standard Mode. Note that a good method of testing whether the card is in Smart mode is reading a PCSMMR and checking for default values.

8.7.5 Bus Arbitration

While in the Smart Mode, the Bus Controller has the full duty of controlling all host and DSP traffic on the module, as well as “snooping” the memory bus for CPU access during the time the DSP has control over the shared memory. The following procedure describes the Smart Mode Shared Memory Bus Arbitration protocol:

1. If the DSP wishes to access the shared memory, it may only do so when the memory is eligible for DSP access. Eligibility is determined by a number of factors including idle time, precharge time, internal bank status (SDRAM only), and previous CPU command history (SDRAM only).

2. Once the DSP gains access to memory, the crossbar switches disconnect the external memory bus from the Shared Memory Bus of the specific module bank. The CPU retains full connection and undisturbed access to the other bank on the module as well as to all other modules. (Note, the DSP may only access one bank on the module at a time, and may not access banks on other modules.)
3. If the CPU requests access of the shared bank by issuing a RAS cycle (DRAM) or an Active, Read, or Write command (SDRAM), while the DSP is also accessing the shared bank, the Bus Controller will give priority to the CPU by immediately interrupting the DSP access. The Bus Controller will also immediately assert the MWAIT signal, stalling the requested CPU memory access until the Shared Bus is relinquished back to the CPU. In the SDRAM case, the Bus Controller will also be responsible for re-opening any memory banks closed by DSP access.

4. After the Shared Bus has been switched back to the CPU, the MWAIT signal will be cleared and the CPU may continue to access the shared bank without interruption. The DSP may not access the shared memory again until the next point of eligibility (back to step 1).

In the Smart Mode, the Bus Controller shall also drive the MIRQ signal to interrupt the host at the DSP's request. Such an interrupt will allow the host CPU to execute other tasks while not having to poll the DSP for status updates.

8.7.6 Host Communication

In addition to the host being able to communicate by writing directly to the DSP's local memory as well as shared memory, an interrupt based communication scheme is also provided. DSP communicates to the host PC via its dedicated communication registers DSPTXD and DSPRXD. These registers reside in the I/O space of the DSP. For the PC they are located in Common Memory, but also dual-mapped into PC I/O space. The I/O location is selectable in the COR register. Both PC and DSP may use h/w interrupts or s/w polling for communicating to the other device. Figure 8-11 shows a block diagram of the structure of this communication.
PC must enable the communication interrupts by setting RXINTEN or TXINTEN to 1 to be interrupted. A 1 in either TXFULL or RXEMPTY (in DSPSR) will generate an interrupt to the PC. Note that the PC Card Standard defines 1 interrupt line so after an interrupt is detected, DSPSR must be read to determine the source. The interrupt is cleared by PC reading DSPTXD or writing to DSPRXD respectively. Note that writing 0 to the interrupt enables will disable, but not clear the interrupt. A 1 in either RXFULL or TXEMPTY bits (PCSR) will generate an interrupt signal to the DSP (-INT3). This interrupt may be masked off in the DSP algorithm if not used. The interrupt signals stay valid until the DSP reads from the DSPRXD or writes to the DSPTXD. Reading or writing to the data register clears the appropriate interrupts to the DSP. Once again writing 0 to the interrupt enables (C54x IMR register) will only disable the interrupt, not clear it. Also note the clearing/priming protocol required to enable the next -IREQ. This is described in detail under Bit 12 and later in this section. Communication between DSP and PC may also be done via s/w polling as shown in Figure 8-11 and discussed.

8.8 DSP-Memory Module Device Driver

Device drivers are the pieces of system software working between applications and hardware. The main role is to
encapsulate and shield the hardware detail from applications. Drivers export a set of APIs (Application Programming Interfaces) that applications can call to utilize the hardware devices. Drivers also interact with the operating system to enable the hardware, request and manage system resources (e.g. system memory and IRQ), handle system messages, and communicate with other device drivers.

![Figure 8-12 PEMM Device Driver Configuration](image)

The PEMM device driver works in the Microsoft Windows'95 environment, functioning between the PEMM application and the PEMM hardware. The driver performs two main tasks. Firstly, it manages the physical memory on the PEMM hardware. Secondly it interacts with the hardware logic (Bus Controller) on the PEMM to manage the DSP program execution. The driver is organized into 4 modules, namely Memvxd.vxd and Memdll.dll to manage memory, and Dspvxd.vxd and Dspdll.dll to manage DSP program execution (see Figure 8-12: PEMM Device Driver Configuration). The driver is described in the following paragraphs.

### 8.8.1 Memory Manager

The Memory Manager manages the memory on the PEMM, implements request/release algorithms, exports APIs that applications can call to use the PEMM memory, implements
memory de-fragmentation algorithms, and handles the memory usage book-keeping.

The Memory Manager commits and takes full control of the module's memory when Windows '95 first starts. By seeking the full control of the PEMM memory, we can freely move and swap out application memory blocks to create space when we start a DSP task. The memory manager exports APIs in both ring 0 (system software level) and ring 3 (application level) such that both system software (VxD, virtual device driver) and applications can use the PEMM memory. The PEMM memory works like ordinary system memory from the view of applications and system software.

Accessing DSP-MM memory is exactly the same as ordinary system memory without any overhead!!

Overhead is incurred only in the memory request and release phases, and all the memory accesses are exactly the same as ordinary system memory accesses. Moreover our driver supports additional advantages. For example, applications can request to physically-lock system memory if they cannot tolerate the time-consuming hard disk swapping in ordinary memory requests.

The Memory Manager keeps track of memory requests and memory usage of the PEMM memory. Also it keeps track of the processes which have requested memory, since some
processes may terminate without releasing the memory. We use the best-fit allocation algorithm to assign memory to requests. The manager implements a relocation algorithm to de-fragment memory to improve memory efficiency. Moreover, to create space for DSP program execution, the manager implements a swapping algorithm to move memory blocks outside the PEMM to other parts of the system memory. All these operations work seamlessly without interfering with the applications. The memory manager also disables the CPU caching and flushes the internal cache when necessary so that the DSP can see the correct data.

As mentioned the manager is implemented as two modules, the ring 0 (maximum privilege level) Memvxd.vxd and ring 3 (application level privilege) Memdll.dll DLL (dynamic link library). Most of the functions are implemented in Memvxd.vxd. Memdll.dll implements the interface for Memvxd.vxd to ring 3 applications. We also provide another application, Meminfo.exe, to display memory usage on the PEMM.

8.8.2 DSP Execution Manager

The DSP Execution Manager manages DSP execution on the PEMM. It implements the algorithms and APIs that applications can use to download and start DSP applications. It also takes care of the synchronization issues. The DSP Execution Manager implements different protocols to communicate with the PEMM hardware. They include PEMM hardware discovery, initialization, PEMM hardware mode switching, DSP program download and executing, and synchronization. The basic core operations are memory-mapped I/O to access hardware registers and interrupt hooking to receive requests from hardware. The DSP Execution Manager interacts with the Windows'95 Configuration Manager to request the memory-mapped I/O window and IRQ, complying with the Windows PnP specification.

The DSP Execution Manager implements the COFF file loader to parse and read DSP programs. When an
application calls the driver’s program loading API, the execution manager requests from Memory Manager space to load the DSP program and data. If there is not enough room on the PEMM, the Memory Manager will perform swapping and relocation to create space. After the DSP program and data are loaded into the PEMM memory, applications can call another driver API to start DSP program execution. The driver will create a new thread, namely worker thread, to handle DSP program execution. The worker thread initializes and starts the DSP task, then suspends itself. Later when the DSP finishes program execution, the hardware will generate an interrupt to the CPU. This interrupt will be trapped by our driver interrupt handler, and the handler will in-turn resume the worker thread, which performs the necessary clean-up and synchronizes with the main thread (CPU side task).

All the thread details will be encapsulated from the application developer. From the application’s point of view, it makes one API call to start DSP program execution, proceeds with its own task, and makes another API call to synchronize with the DSP task and gets the result. The multithreading and synchronization models impose minimum overhead in CPU cycles and ease the application development.

The DSP Execution Manager is implemented as two modules, a ring 0 Dspvxd.vxd and ring 3 Dspdll.dll DLL.

8.9 Applications

Potential applications are numerous. Just about any application that can benefit from additional and/or faster processing power can benefit from a PEMM. Currently the focus is on applications where the PEMM can have the biggest impact or be an enabling technology. These include real time 3-D graphics, video processing, compression/decompression, network and internet applications, image processing and high quality audio applications. To achieve a broad application base, the work of several groups both within and outside TI are being leveraged in addition to new
application development. In addition to PEMM aware applications (i.e. applications which can use the PEMM module directly through the driver), the PEMM can also support existing applications through hardware acceleration of audio and graphics under Microsoft's OS. This kind of acceleration requires a driver which announces the PEMM's capabilities to the OS, and the OS will use the PEMM just as it would a graphics or audio card for hardware acceleration. Finally, with the help of Microsoft (and other OS makers), the operating systems themselves can become PEMM aware applications and take advantage of its features (when present) for many routines. This benefits not only the operating systems but also many PEMM unaware applications which call routines provided by the OS.

![Figure 8-13 PEMM Applications](image)

### 8.10 Performance Benchmarking

Several experiments were done to estimate the speed of the C6x vs. a 200MHz Pentium at common multimedia computations (DCT/ IDCT and vector sums). These tests indicate that many multimedia tasks can see a 5-6 fold speed boost just by being executed on the C6x rather than on the host.
Dividing the task between both processors can further increase performance.

![Bar Chart](image)

Figure 8-14 Raw Processing Power

In addition to providing additional raw processing power, the PEMM also provides a way around a different and ever increasing bottleneck. Currently most DSP cards reside on the PCI bus, which is not only four times as slow (i.e. 1/4 the bandwidth) of the memory bus, but also conflicts directly with the memory bus in that the two cannot be used simultaneously. As processing speeds increase the demand for data bandwidth to and from the processors also increases. However, using both the PCI bus and the memory quickly leads to bus contention even when accesses are made to conflict as little as possible (i.e. the best case scenario). In Figure 8-15 or example an ideal situation would allow five times as much bandwidth to reach the C6x on the PCI bus. However as bandwidth needs increase, contention with the memory bus reduces the five to one ratio until no processing can be done on the C6x.

![Graph](image)
Figure 8-15 Bus congestion for C6x bus with Pentium 200 as host, 5:1 on PCI processing assumed (% of bus capacity vs % of max. b.w.)

With the PEMM module a new bus is introduced (the DSP Local Bus) which not only provides bandwidth equal to the memory bus, but also does not conflict with memory bus access to other memory modules by the host. In effect, the total bandwidth for data access to/from main memory is doubled. In addition, if more than one PEMM is used, the processing power and memory bandwidth again increases and no bus contention exists as long as the host is working with other memory modules. If another processing unit is added to the PCI bus, even in the best case with coordinated bus sharing, the amount of bandwidth per processor is reduced.