DSP-Based Handprinted Character Recognition

Application Report

Alan Josephson
Information Technology Group

SPRA143
October 1994

Introduction

The market for pen-based computers is growing. Pen-based computers include notebook-sized tablets, pocket organizers, and handheld computers (HHC). Most pen-based computers offer handprinted character recognition (HCR), and some are beginning to offer cursive handwriting recognition. Most implementations of pen-based computers with HCR suffer from slow response times and inaccurate recognition. The HCR algorithm is typically implemented on the CPU of the pen-based computer.

This report describes how to implement HCR for real-time applications. Such applications are found in a variety of industries, including financial trading, healthcare, and transportation.

Users of handheld computers require fast response and high accuracy recognition rates. To meet these requirements, the execution of HCR tasks in a handheld computer is shared among a pen-input processor, a TMS320C5x digital signal processor (DSP) residing on a Type II PCMCIA-compatible card, and the main CPU. The input processor digitizes and filters handprinted input written on a resistive pad. The DSP manages pen-stroke and character libraries. The DSP performs character-based matching by using these libraries and digitized strokes provided in real-time by the input processor. The DSP provides the character string with the best match to the input data, along with a set of possible alternatives, to the main CPU. In addition, the DSP can handle high-level verification of character recognition, such as constraining the matching to a dictionary of valid character string inputs. The main CPU handles inking of data to the LCD, establishing a recognition context, and communicating pen-stroke data to the DSP.

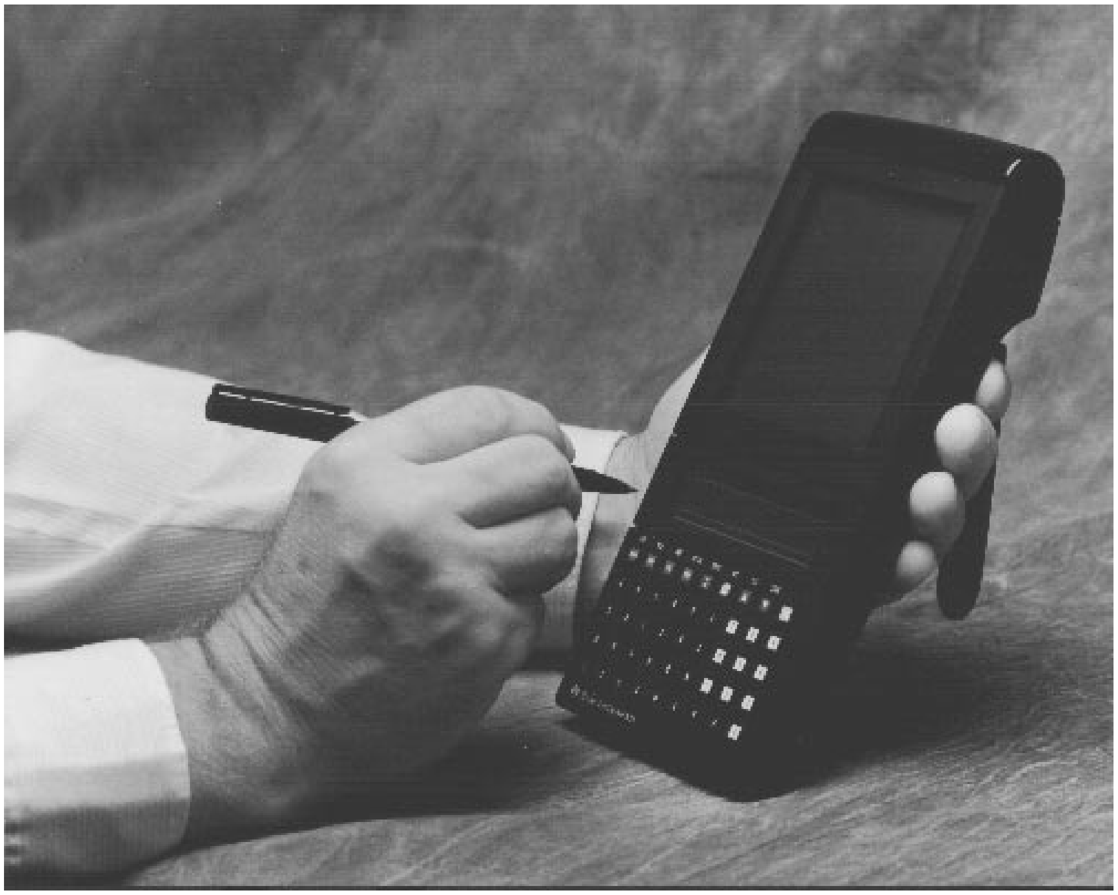
The prototype HHC platform described in this paper was developed in conjunction with Commodity Exchange, Inc. of New York (COMEX) as a means for traders and brokers to input trades to the exchange and electronically receive price and trade order information. The HHC platform system can also be used in other industries that require communicating with pen-input computers and wireless LANs.

The current implementation of HCR is integral to the HHC and runs on an MC68000 processor, along with other system and application software. This paper shows how a PCMCIA card containing static memory and a general-purpose DSP can be used to implement HCR in a multiprocessor setting. A Type II PCMCIA-compatible card containing a TMS320C5x DSP and 256K bytes of SRAM (referred to as a *DSP/memory card* or *DSP card*) is under development by the Texas Instruments Semiconductor Division [1, 2, 3]. The architectural and functional aspects of this card that are relevant to the implementation of HCR are discussed.

Architecture

The prototype HHC is shown in Figure 1. The processors and I/O devices for implementing HCR are described in the following paragraphs.

Figure 1. Prototype HHC Platform With Pen Input



Host Processor

The MC68302 is the main processor used in the HHC. The 68302 is an integrated multiprotocol processor consisting of an MC68000 core, a system integration block (SIB), and a communications processor (CP). The CP is connected to the core through the SIB, not the data bus, and can operate independently of the core. This feature allows multiple tasks to be implemented in hardware, providing increased system speed and better power management. The 68302 has the ability to put the core and the CP to sleep independently, allowing large power savings. The return to a normal operating state is very quick and undetectable by the user. The MC68000 core is referred to as the *host processor.*

Input Processor

Any standard ball-point pen, pencil, or stylus can be used to enter handprinted input and signatures onto a resistive, opaque X-Y digitizing pad, located between the LCD and the elastomeric keyboard. An Intel 8051-like Signetics S87C552 microprocessor performs input preprocessing and provides low power modes of operation used in power management. This processor is awakened through hardware whenever a key is pressed, the discrete matrix touchscreen is touched, the digitizing pad is touched, or data is received over its serial line from the host processor. The firmware drives the A/D circuitry, which biases the digitizing pad and gets X-Y and touch-detect readings. Higher level software averages the X-Y points and reports the filtered *strokes* (a pen down, followed by a stream of points, followed by a pen up) over the serial line to the host processor for recognition or for signature compression. As information is sent to the main processor for recognition and/or storage, it is presented, or *inked*, on the LCD to provide feedback to the user. Key presses and touchscreen touches are reported similarly.

DSP/Memory Card

The DSP/memory card can be used either as standard memory or as a multifunction peripheral device. The HCR (and other) DSP algorithms can be loaded into the card by a host processor in the same way it writes to any PCMCIA memory. Once the program is loaded, the host can command the DSP to execute the algorithm as a CP. Among the key features of the DSP/memory card used in this implementation of HCR are on-board logic to arbitrate the memory bus between the DSP and the host, direct interrupt control and handshake between the host and DSP, and host control of DSP operating speeds for power management.

System-Level Software

A real-time operating system (RTOS) with facilities for multitasking and interprocess communications runs on the host. The application program interface (API) implements the application programmers' view of the operating system. Included in this interface are functions to accept input from the keyboard, the touch screen, the digitizing pad, or the communications system. Also included are functions to output data to the liquid crystal display (LCD), to handle communications, to access RAM, and to access the PCMCIA-compatible card.

The input subsystem routines allow application programmers to manage the input queue that records user inputs from the touch screen, keyboard, and HCR subsystem. The modularity of the HCR subsystem makes porting it to the DSP/memory card straightforward. Initialization routines for the HCR subsystem are available from within applications. Communication of recognition parameters between applications and the HCR subsystem occurs through APIs that manipulate the recognition context, which is composed of inking parameters, active model databases, active gesture sets, and active constraint dictionaries. The HCR subsystem is activated asynchronously when serial data from the digitization subsystem is received. After processing the digitizer data, the HCR subsystem sends a notification to the application (similar to those sent for keyboard and touch screen events) that the recognition results are awaiting processing. The application is then responsible for invoking final translation and constraint of the recognition result through function calls to the HCR subsystem running on the DSP/memory card.

To conserve power when the HHC is not in use, it can be placed, under software control, into one of two sleep modes: shallow sleep or deep sleep. In the shallow sleep mode, the processor is active, but some of the nonessential services have been turned off. The application is unaware of the shallow sleep mode that is managed by the HHC system software. In the deep sleep mode, almost all services are turned off, the internal status of the processor is saved, and the HHC uses the minimum power required to wake up automatically when an interrupt occurs from the keyboard, touch screen, digitizing pad, communications system, or DSP.

HCR Subsystem Description

The HCR algorithm embodies an operator-trainable stroke-based approach. The operator can enter models for individual characters (alphabetic and numeric) during interactive training sessions. These models make up *model databases* that are employed by the recognition software as a basis for translating the operator's handwritten input from within applications. As the operator writes on the digitizing pad, strokes are digitized into a set of discrete points that are used as input to the recognition subsystem. Strokes are thinned so that not all points are retained; strokes are normalized with respect to a common scaling factor. After normalization, the stroke is compared to all strokes in the currently active stroke database, and a *degree of match* or *penalty* is determined for each. Once this process has been repeated for all strokes in the current input, the sequence of strokes is *parsed* into a set of potential symbol matches, utilizing the models in the currently active model database as references. As possible recognition results are computed, they are stored — along with their associated penalties — to be reported to the application. The HCR approach does not require the operator to write within a specified grid. Spatial separation between characters is not essential for recognition, and the user may overlap and overwrite characters.

Application-generated contexts allow the recognition software to disambiguate between otherwise indistinguishable models from different databases (for example, numeric 0, 1, and 5 from alphabetic O, I, and S). Using context-sensitive dictionaries to constrain fields *before* the recognized result is reported to the user causes the perceived recognition accuracy to be higher than it would be if only character-based recognition were being used, and it causes it to be *much* higher when alphabetic and numeric contexts are available. Additionally, context provides a means for increasing recognition speed, because the database of models to be searched is smaller.

HCR Subsystem Implementation

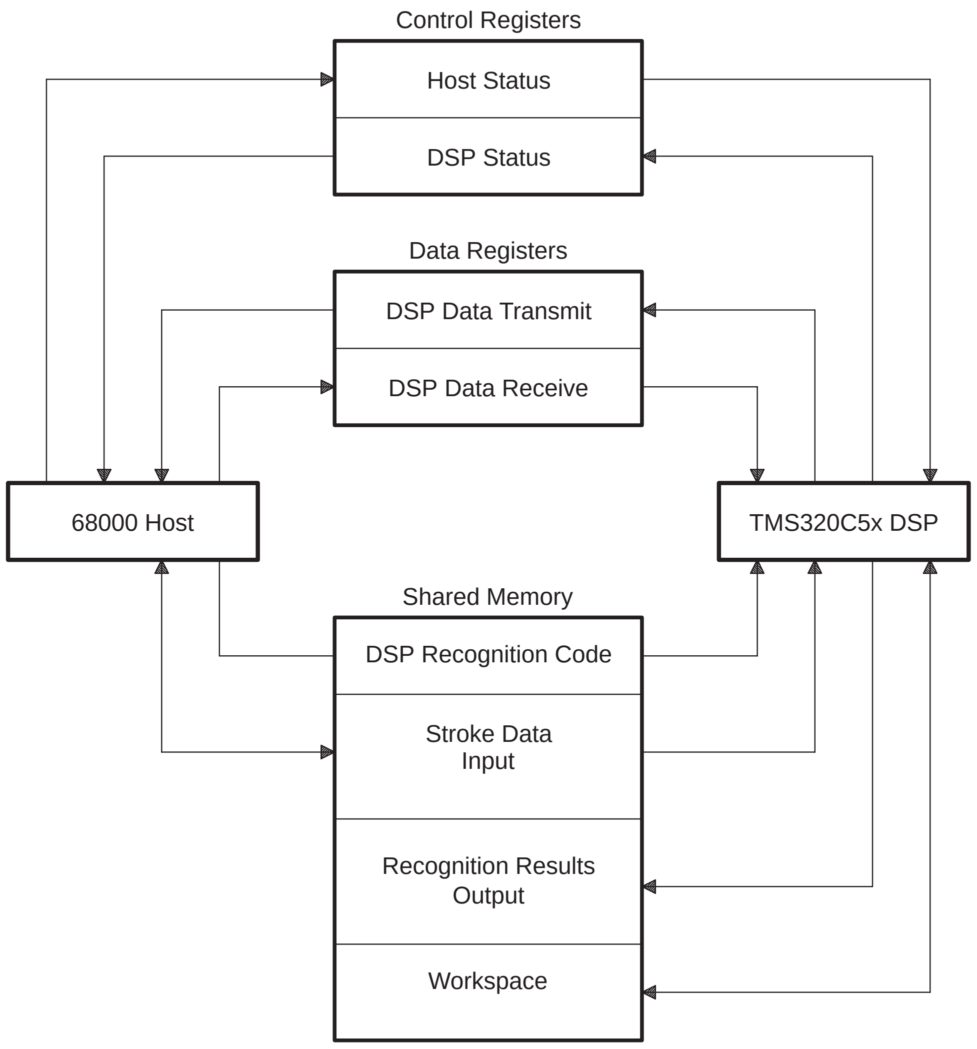
The incremental nature of the HCR algorithm makes it a natural candidate for exploiting the parallelism offered by the DSP CP. The host processor is responsible for receiving strokes from the input processor, optionally inking the digitized points to the LCD, performing high-level filtering of the digitized strokes, and communicating the strokes to the DSP for processing. Additionally, application-level software running on the host processor communicates contextual recognition parameters and requests for recognition results to the DSP. The DSP incrementally processes the strokes as they arrive from the host by forming partial recognition results. Also, the DSP — in response to the host's requests — sets recognition parameters and generates final translations of pen input on the basis of the recognition parameters.

Memory Organization of the DSP Card

Figure 2 shows the DSP card's memory map as used by the HCR subsystem. The shared memory is partitioned into code memory, stroke data (input), recognition results (output), and a workspace for the HCR subsystem running on the DSP. The arrows in the diagram indicate read/write privileges for the host processor and DSP. This partition of the shared memory is a design choice, based on the read/write privileges. If some segment of the card memory requires write privileges for both the host processor and DSP, card logic in an onboard FPGA/ASIC handles this memory contention by giving precedence to the host.

The DSP code is loaded into the shared card memory by the host during initialization of the HHC. The host then switches the DSP card from *standard mode* (in which the DSP is inactive) to *smart mode*, at which time the DSP begins execution from the code segment. The DSP initializes variables in its workspace and enters a processing loop, awaiting commands from the host processor.

Figure 2. DSP Card Memory Organization for HCR



Interprocessor Communication

The host processor and DSP communicate through dedicated 16-bit data, status, and control registers in the FPGA/ASIC on the DSP/memory card (see Figure 2). Read and write access to the DSP data transmit and DSP data receive registers is enforced by the onboard logic.

Host-to-DSP Communication

The host communicates to the DSP by writing commands to the DSP data receive register. The host has only write access to this register; any host read from this register causes invalid data to be read. The DSP has only read access to the DSP data receive register; any DSP write to this register is ignored. A host write to this register generates an interrupt to the DSP. Similarly, a DSP read from this register generates an interrupt to the host. The host status register is accessed by the DSP to determine the status of host communication registers.

The DSP recognition code is a stand-alone application that does not run under an operating system. The top-level DSP recognition code consists of a command processing loop that interprets commands written to the DSP data receive register. Thus, when a stroke becomes available to the host from the input processor, it is communicated to the DSP by first copying it to the card's shared memory. The host then issues a command to the DSP to process the stroke. Commands to initialize and reset HCR parameters are communicated similarly. Whenever the DSP has no pending commands from the host, it enters its lowest power mode [4]. Writes to the DSP data receive register awaken the DSP for further processing.

DSP-to-Host Communication

The DSP communicates to the host by writing commands to the host data receive register. The DSP has only write access to this register; any DSP read from this register causes invalid data to be read. The host has only read access to the host data receive register; any host write to this register is ignored. A DSP write to this register generates an interrupt to the host. Similarly, a host read from this register generates an interrupt to the DSP. The DSP status register is accessed by the host to determine the status of DSP communication registers.

The RTOS running on the host uses interprocess communication primitives to provide race-free synchronization and communication mechanisms. The RTOS supports message queues that are not bound to any task. Tasks may send messages to a queue, and several tasks may request messages from the RTOS. When the DSP writes to the host data receive register, an interrupt service routine on the host places the DSP message on the input subsystem queue. This implementation provides a seamless interface between APIs running on the host processor and the HCR subsystem running on the DSP.

Application Command Protocols

The application command table (ACT) for the HCR subsystem is shown in Table 1. The INITIALIZE, RESET_HCR, and BUILD_RESULT commands have no parameters and are sent directly to the DSP data receive register by the host's APIs. The SET_PARMS and PROCESS_STROKE commands are sent after their parameters have been written to the input section of the shared memory. The DSP sends a RESULT_READY after it has generated the recognition results and written them to the output section of the shared memory.

Table 1. Application Command Table for HCR Subsystem

Command Name	Host to DSP?	Parameters	Command ID	Function
INITIALIZE	True	None	00	Initialize variables on DSP for HCR
RESET_HCR	True	None	01	Reset HCR context for new entry
SET_PARMS	True	Database, dictionary, # of return strings	02	Set HCR parameters
PROCESS_STROKE	True	Stroke data	03	Perform incremental recognition on next stroke
BUILD_RESULT	True	None	04	Generate recognition result(s) based on parameters
RESULT_READY	False	Recognition results	10	Signal that the HCR results are ready

Results

As of this writing, the implementation of HCR using the DSP/memory card is not yet complete. Porting of the HCR recognition software to a PC-resident EVM board containing a TMS320C5x DSP and sufficient memory to emulate the DSP/memory card is in progress. Initial results indicate that the overhead in transferring data between the two processors is minimal and that a high degree of parallelism is possible. The final porting of the code depends on availability of the DSP/memory card.

References

1. Pawate, Basavaraj, Frantz, G.A., and Chirayil, Raj, “System Design Using Memory With a Processor Having Communication With Host Processor”, Texas Instruments (patent pending), 1993.

2. Pawate, Basavaraj, Frantz, G.A., and Chirayil, Raj, “System Design Using Memory With a Host Processor for Activating a CP”, Texas Instruments (patent pending), 1993.

3. Chirayil, Raj, and Pawate, Basavaraj, “PCMCIA DSP/Memory Card Specification”, Texas Instruments, November, 1992.

4. *TMS320C5x User’s Guide*, Texas Instruments, 1993.

5. Pawate, Raj, “BASAVA Concept”, Texas Instruments, November, 1991.