Designing an application oriented terminal

by J. P. KOHLI
NCR Corporation
Dayton, Ohio

INTRODUCTION

Application oriented Terminals have been around for quite some time and traditionally they have been hard wired. The approach, however, has been changing recently due to the availability of low cost RAM memory and microprocessors. The main reason for the trend is that although the hard wired Terminals cost less, they do not provide or have the flexibility a customer is provided with a programmable Terminal. However, the cost of programming becomes an important consideration and will vary depending upon what data is available with the Terminal to aid the programmer. In designing the Banking Terminal, Honeywell used a combination of the traditional hard wired implementation approach and the complete programmability approach. Honeywell also provided for a COBOL type “FITAL” (Financial Terminal Application Language) user level language for the customer, to aid programming the transaction sequences which allowed the customer to reduce programming cost. The limited programmability approach reduces the amount of programming, while it does not sacrifice flexibility available to the customer to tailor the Terminal transactions to suit his requirements.

This paper presents the Honeywell design approach by describing:

1. Bank Teller Terminal 7340 and its features.
2. Operation of the Terminal.
3. Firmware/RAM Architecture of the Terminal.
4. Real Time Processing Considerations in the Firmware Design.
5. Customer Programming in RAM.
6. Programming Aid to the Customer (FITAL Language).
7. Conclusions—Advantages and disadvantages of using the Honeywell approach versus hard wired or completely programmable terminal design approach.

TERMINAL DESCRIPTION

Bank Teller Terminal (BTT 7340) is a data entry device designed to be located in a teller window to record and control transactions in a banking environment. The Terminal logs transactions and prints on a passbook and a journal through real time communications to a computer, thus aiding the teller in accepting and processing transactions. The transaction sequence is customer programmable and is loaded in the Terminal memory through the communication line via the host processing facility. The Terminal has on-line/off-line diagnostic modes. It is a self-contained unit and does not require a separate controller. Some of the salient features are described below and shown in Figure 1.

(a) **Keyboard**—Used to enter transactions from the Terminal. Includes alpha, numeric, and programmable functional key capabilities.
(b) **Printer**—160 column printer is used for printing the Journal (audit trail of transactions), passbook, and customer receipt.
(c) **Status Lights**—Ten indicator lights that indicate the status of the Terminal and the communication line at any given time.
(d) **Mode Switch**—Used to control the Terminal mode (On-line, Off-line, or Diagnostic mode).
(e) **Tutorial Lights**—Up to 28 programmable indicator lights that can be interlaced with the transaction sequence providing tutorial lead-through to the teller and/or providing a pictorial history of keyboard action while entering a transaction.
(f) **Dynamic Data Display**—A 32 character alpha/numeric display allows display of keyboard entry and editing and provides up to eight 32 character lines for forms fill-out or Inquiry display via communication messages.
(g) **Booking Keys**—Provide for teller identification when booking the Terminal.

TERMINAL OPERATION

During power turn on a sequence of hardware reliability checks are performed automatically. If the checks pass, the Terminal is ready to accept the user program tables via communications with the central processor or a backup Load Cassette Unit. Once the tables parameters are loaded, the table processor in the Terminal ROM memory processes the tables. Each table contains information for illuminating tutorial lights, performing specific terminal functions and address of the next table to be processed.
There are twenty-four different types of tables available to the user. To aid the user, a COBOL type language "FITAL" is available for programming the various terminal functions.

TERMINAL Firmware/RAM ARCHITECTURE

The heart of the Terminal is the Basic Logic Unit (BLU), a Honeywell developed microprocessor which has been used in a number of other Honeywell Terminals. Figure 2 shows the overall architecture of the Terminal. The hard wired programming in ROM (firmware), the customer programming of loadable list processing type tables in RAM, the standard interrupt and Input/Output inter-

1. Communications control procedure for interface with the Host Processor.
2. Loader and processor for the RAM tables.
3. Various Terminal Input/Output functions: i.e., open passbook door, print passbook, buffering keyboard inputs, interface Data Display, etc.
4. Reliability checks and test diagnostics for the Terminal.
5. Add routine to perform various adding, subtracting functions.
6. Reset capability for the Terminal.
7. Auto insertion of decimal point and right justified printing of amount fields.

face of the microprocessor, through the Device Oriented Electronics (DOE), help drive the Terminal. The logic to process the customer tables and specific processes unique to the Terminal are in the ROM; the customer programmable parameter tables are loaded in RAM. In addition to the tables, subroutines may be written in BLU machine language to achieve functions beyond those provided by the tables. For the microprocessor, ROM, and RAM are interchangeable and, therefore, the question becomes one of which functions of the Terminal should be in ROM and which functions should be stored in RAM. Obviously programming the Terminal functions in ROM is cheaper but not flexible; whereas providing them through user parameter tables is more expensive but allows the user control of the transaction sequences per his needs. Therefore, the major consideration in deciding between ROM and RAM was that the customer should be able to program the Terminal per his requirements with a minimum of programming. Subsequently, all the Ter-

1. Transaction Sequences and Responses
2. Functioning of Tutorial Lights during Transactions
3. Formats for Journal, Passbook, Receipt and Validation Printing
4. Ability to Override Terminal Messages from the CPU
5. Ability to use Mode Switch, Function and Booking Keys in a desired way
6. Ability to use Customized Account Number Verifying Routine (check digit)
7. Ability to Load and Call Special Routines
8. Interpretation of Function Keys
9. Specifying Fields as Alpha/Numeric or Numeric Only
10. Checking of Acceptable Field Length

Figure 1—Bank teller terminal—7340

Figure 2—Firmware/RAM architecture of the terminal—BTT 7340

Figure 3—Functions programmed in ROM

Figure 4—Functions available in RAM through programmable tables
minal features that are fixed and unique to the Terminal (For Example: Opening the passbook door, moving the carriage, causing the print head to operate, communicating with the processor, test and diagnostics for Terminal, etc.) are programmed in ROM. However, controlling the sequence of steps within a transaction, formats of printing on passbook and Journal, selected illumination of tutorial lamps and interpretation of function key codes, etc., are considered programmable and are stored in RAM by the user program. These programmable functions are provided to the customer by list processing type program tables. There is a table processor in ROM that processes these RAM tables after the customer has loaded them into the Terminal's memory. The customer programs these tables per his banking environment using "FITAL" language and associated compiler and then loads them into the Terminal's memory through the use of a loader (also programmed in ROM). Figure 3 shows the various Terminal functions programmed in ROM. Figure 4 lists the Terminal functions available to the customer through the use of the program tables.

REAL TIME PROCESSING CONSIDERATIONS IN THE Firmware DESIGN

The Terminal is designed to communicate with the host processor during transaction processing in real time. In addition, while communicating with the host processor, keyboard data may be entered and the printer may be activated. Since these functions involve simultaneity of operations for the Terminal, a mini-operating system in firmware was designed to handle these independent parallel Terminal operations. The design includes a central executive which processes five subexecutives in partitional segments as shown in Figure 5. The choice of five subexecutives and the amount of partitional segments to be executed each time, was dictated by parallelism in operations and the speed of operations; e.g., communication line can communicate up to 4800 baud synchronous, printer can print characters at the rate of 20 characters/second, the keyboard data can be entered at 10 characters/second.

Functions that different subexecutives perform are as follows:

1. Communication Executive
   Performs the loader function for loading customer programmed tables in RAM and provides for communication to and from the host processor.
2. Keyboard Executive
   Performs keyboard functions of receiving and storing keyboard data to be processed by the application executive.
3. Time Out Executive
   This executive continuously checks for the expiration of various time outs that may be set by printer or application executives. It signals the completion of the time out to the other executives.
4. Printer Executive
   Performs all the printer functions of printing on the Journal and the passbook.
5. Application Executive
   Performs the rest of the execution for the Terminal
which includes:

(a) Processing the customer program tables in RAM.
(b) Performing various terminal activities such as passbook control functions, processing data received from communication and keyboard executives, providing data to the printer executive, displaying terminal status, etc.
(c) Test and diagnostics for the Terminal.
(d) Power on initialization and reliability checks on the Terminal.

The central executive keeps the Terminal performing a number of activities in parallel by executing each of the above described programs in partitional segments.

CUSTOMER PROGRAMMING IN RAM

The customer programs the transaction sequences, tutorial lights, Journal and passbook printing formats, etc., through the use of program tables. In addition, a number of loadable BLU executive routines provided by Honeywell can also be loaded and called by the tables. These routines can be written for the user thus providing special function capability; e.g., a check digit routine on account number field may use the user’s algorithm to validate the account number in a transaction. Since the functioning of the Terminal components has been programmed in firmware, the user is left with the programming that determines sequence and formats of transactions per his requirements. This helps reduce customer programming effort considerably, while not sacrificing flexibility.

PROGRAMMING AID TO THE CUSTOMER (FITAL LANGUAGE)

In order to make user’s programming and debugging effort still easier, Honeywell provides him with a COBOL based higher level application oriented language called “FITAL” and an associated “FITAL” compiler. “FITAL” compiler is written in COBOL and runs on Honeywell 2000 and 600 Series. The language is COBOL “like” such as the

```
IDENTIFICATION DIVISION.
PROGRAM WRITTEN BY GEO. KLINE
DATE: JULY 23, 1973
PROGRAM-ID: A.

ENVIRONMENT DIVISION.

SEQUENCE NUMBER

SOURCE LISTING---------IDENTIFICATION DIVISION BEGINNING ADDRESS BIT CODING

IDENTIFICATION DIVISION.
PROGRAM WRITTEN BY GEO. KLINE
DATE: JULY 23, 1973
PROGRAM-ID: A.

ENVIRONMENT DIVISION.

SEQUENCE NUMBER

SOURCE LISTING---------ENVIRONMENT DIVISION BEGINNING ADDRESS BIT CODING

TERMINAL SECTION.
TYPE IS 1.
HEIGHT-SIZE IS 2.
KEYBOARD-TYPE IS 3,

TUTORIAL SECTION.

DATA DIVISION.

SEQUENCE NUMBER

SOURCE LISTING---------DATA DIVISION BEGINNING ADDRESS BIT CODING

CONSTANT SECTION.
ASSIGNS TO THREE AN-ASCII-VALUE-OF-HEX 33.
ASSIGNS TO PLUSKEY AN-ASCII-VALUE-OF-HEX 2B.
ASSIGNS TO ENEROY AN-ASCII-VALUE-OF-HEX 3D.

DESCRIPTION SECTION.
LBB, 'PRINT-DISPLAY',

PROCEDURE DIVISION.
```

Figure 7—Sample program listing for FITAL
<table>
<thead>
<tr>
<th>Sequence Number</th>
<th>Source Listing</th>
<th>Procedure Division</th>
<th>Beginning Address</th>
<th>Bit Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>DECLARES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>TELLER=2 GO TO BOOK1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000020</td>
<td>TELLER=3 GO TO IDLE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000030</td>
<td>TELLER=4 GO TO IDLE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000040</td>
<td>TELLER=5 GO TO IDLE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000050</td>
<td>TELLER=6 GO TO IDLE.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000060</td>
<td>TELLER=7 GO TO BOOK1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000070</td>
<td>TELLER=8 GO TO BOOK1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000080</td>
<td>TELLER=9 GO TO BOOK1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000090</td>
<td>END-DECLARATIVES.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7—Sample program listing for FITAL (Continued)
Figure 7—Sample program listing for FITAL (Continued)
Figure 7—Sample program listing for FITAL (Continued)
FUNCTION | HARD WIRED APPROACH | COMPLETELY PROGRAMMABLE TERMINAL APPROACH | BANK TELLER TERMINAL 7340 APPROACH
--- | --- | --- | ---
1. Flexibility in Customer Programming | Not Available | Available | Limited availability
2. Customer Programming Effort | None | Whole Terminal is to be programmed. | Programming kept to a minimum.
3. Systems Effort in Programming | None | Required in most cases. | None with the availability of FITAL.
5. Loading the Terminal at Power On Time | Not Required | Complete program has to be loaded in the memory. | Due to the firmware, the amount of program to be loaded is much shorter than that for the programmable Terminal.
6. Speed of Execution | Fast - No overhead of programming exists | Relatively slow as all the functions have to be programmed in software. | Not as fast as hard wired, but fast enough to maintain Terminal-Computer interface and Operator-Terminal interface without error or delay.

Figure 8—Advantages and disadvantages of Honeywell approach versus hard wired/completely programmable approach

**FITAL program example**

Consider a simple program to accept a transaction from Teller A only, print A to identify teller, insert an ASCII 3 into message to Central Processor (CP), accept an alpha/numeric field of data from the operation, wait for END OF TRANSACTION key from the operator, and then if:

- Off-Line—Print the field, then set Terminal idle
- On-Line—Send message to CP, receive and process response from CP, then set Terminal idle.

Operator transaction key stroke entry

A (Alpha/Numeric Field) + END

Journal Printing Layout:

(Printed at entry time from operator input)

A (Alpha/Numeric Field)

Position 1  Position 15

(Alpha/Numeric Field)

Linefeed  Position 10

A transaction flow chart for the example above is shown in Figure 6. The program listing appears in Figure 7.

**CONCLUSIONS**

The Honeywell approach has provided a good compromise between a hard wired Terminal and a completely programmable Terminal. This has helped Honeywell provide a cost competitive Terminal with sufficient flexibility for the customer to tailor the Terminal to his needs. The rationale used in the design has been that the customer programming requirements should be kept to a minimum, restricted only to job oriented functions, while not sacrificing the flexibility to the customer. This was accomplished by fixing all real time and mechanism control functions in ROM while allowing user program control of all transaction related functions.

Figure 8 summarizes some of the advantages and disadvantages of the Honeywell approach in comparison with the hard wired or the completely programmable terminal approach. Honeywell feels that the savings in programming cost to the user and savings in hardware memory cost are sufficient to justify the minimal limitations in customer programmability.

**ACKNOWLEDGMENT**

The author wishes to acknowledge H. K. Barger, R. L. Bibles, M. Gilbert, G. H. Kline, A. F. Maggio, and G. B. Voelker for their valuable suggestions on the paper and Honeywell Information Systems for granting permission for presenting the paper.