MICROPROCESSOR IMPLEMENTATION OF THE HP-IL PROTOCOL

GARY MUHONEN

MOUNTAIN COMPUTER, INC.

January 9, 1983

Abstract

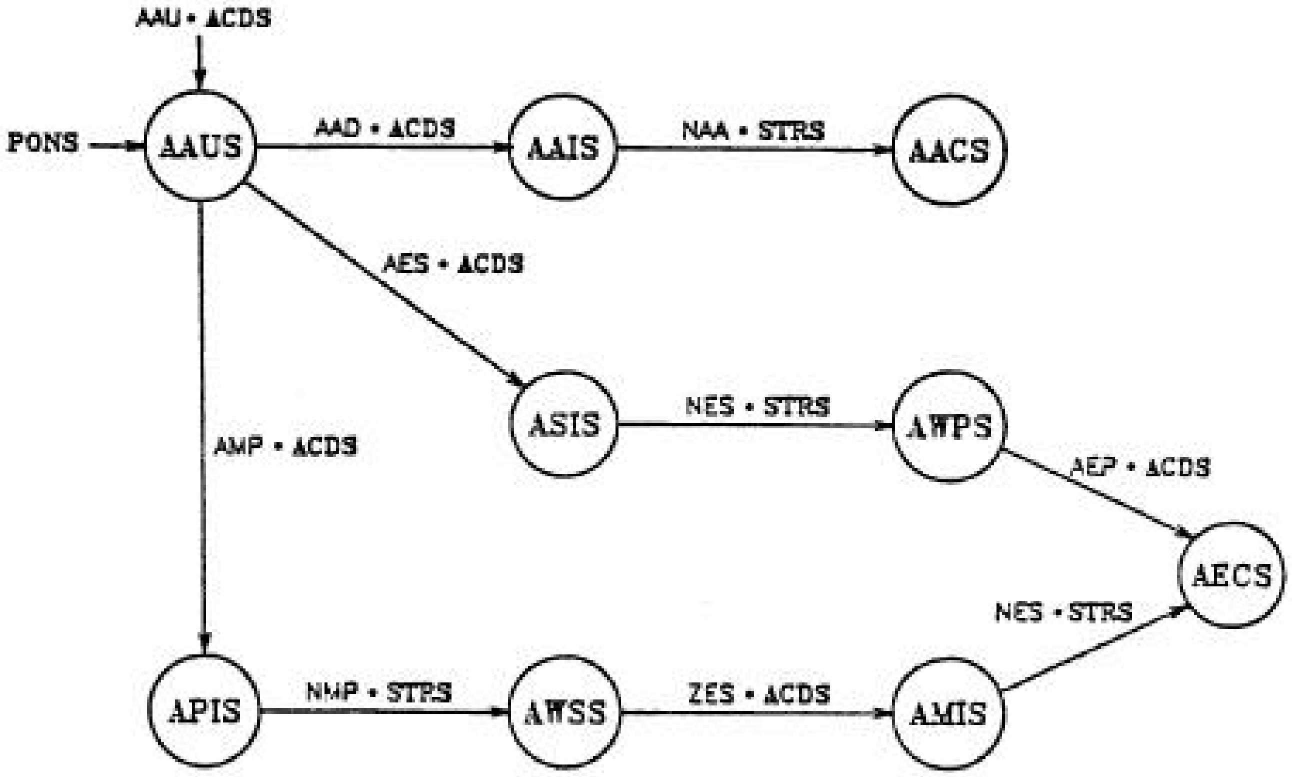
This paper describes a design approach to implementing the HP-IL protocol in a microprocessor. The software makes use of state machines to follow the HP-IL state diagrams provided by Hewlett-Packard. This approach provides many advantages over conventional straight line programming.

Hewlett Packard recently announced a new interface system called HP-IL (Hewlett Packard Interface Loop). It is a powerful two wire network that can connect up to 961 peripherals to a host computer via a daisy chain interconnection scheme. HP-IL is well suited for existing personal computers, and the new wave of portable computers. HP currently has HP-IL available for the Series 80 computers, HP75C, and the HP41C.

To make a peripheral work on the HP-IL, the designer has several options. HP provides a parallel converter (HP82166A) that has the HP-IL protocol implemented in it with a parallel port as the interface to the designer's product. Using the converter provides an easy solution to the interface design, but has several limitations. The cost of this unit may be excessive for some applications, and the complete protocol is not implemented in the converter. Additionally, the parallel port may not be what the designer needs.

Another approach to HP-IL implementation is to design the HP-IL protocol and desired peripheral together. This can be done by using a microprocessor and its associated hardware to accomplish the desired task. HP has provided several items to make the design task easier. HP provides a chip that contains a major portion of the low end HP-IL protocol. This chip will interface directly with the designer's microprocessor. Additionally HP sells the documentation set that completely describes the HP-IL protocol.

The HP-IL protocol is defined as a series of state diagrams in the HP-IL reference specification. Figure 1 shows one of the diagrams. A state diagram is similar to a flow chart, and each describes part of the entire HP-IL system. This state diagram describes the auto addressing method for HP-IL. The system is always in one and only one of these states. On another diagram of the HP-IL protocol, one and only one state will be active in that state diagram. The system at any given point in time can be totally described by listing which state each of the state diagrams is in. The arrows show the direction and what needs to happen for the state to change from one state to the next. In this way the state diagrams are connected. For example, a state on one diagram may change if the state of another diagram changes. This approach to describing a system is very elegant as it can describe a very complex system in terms of many individual components.



Messages

AAD	auto address	NAA	next auto address
AAU	auto address unconfigure	NES	next extended secondary
AEP	auto extended primary	NMP	next multiple primary
AES	auto extended secondary	ZES	zero extended secondary
AMP	auto multiple primary		

Interface States

AACS	auto address configured state	ACDS	acceptor data state (from AH)
AAIS	auto address increment state (links to SH)	PONS	power on state (from PD)
AAUS	auto address unconfigured state (links to R)	STRS	source transfer state (from SH)
AECS	auto extended configured state (links to L,T)		
AMIS	auto multiple increment state (links to SH)		
APIS	auto primary increment state (links to SH)		
ASIS	auto secondary increment state (links to SH)		
AWPS	auto wait for primary state (links to R)		
AWSS	auto wait for secondary state (links to R)		

Figure 1

Writing the code for a system described by state diagrams can be difficult and tedious, because the diagrams are interrelated. Conventional straight line coding will work, but there are several pitfalls. First the state diagram must be converted to flow charts, and since the diagrams are intertwined there is a high risk of missing some details. Another approach is to write the software using a state machine approach, which is written directly from the state diagram. Here, the software is written in pieces, one for each state diagram. The actual device that is to be made into a HP-IL peripheral would be designed in terms of a state diagram and it would interact with the other state diagrams to pass data and control information.

In the actual microprocessor implementation, a state table can be implemented. Each byte of the table can represent the state of one state machine. As a machine changes state it would update the byte in the state table to its current state. If one state machine needed to check the state of another state machine, it would look in the table to check its state. In this way, each machine is independent, but can check the conditions of its environment.

The software's main core could consist of a series of subroutine calls to the various state machines, and they would each follow the instructions of the state diagram. If a state change or action was called for, it would do it, and return to the main core. The last subroutine might be a device dependent routine that handles the specifics of what this particular device is to perform.

This implementation has several unique advantages over conventional coding. The designer can look at the state table and easily see what the entire state of the system is. This greatly accellerates debugging. Also, since the system is modular, and broken into small segments, changes to one section of code are not likely to adversely effect others.

This approach was recently used in designing the Mountain Computer HP-IL Eprom Programmer. Debugging was straight-forward and simplified due to this approach. With future HP-IL products, eighty percent of the code will remain the same, and only the device dependent sections will change.

APPENDIX

MICROPROCESSOR IMPLEMENTATION OF THE HP-IL PROTOCOL

GARY MUHONEN
MOUNTAIN COMPUTER, INC.

January 9, 1983

The following contains actual source code listings for part of the HP-IL protocol implemented for the 8039 microprocessor. Page A2 shows the main core of the program which starts with the initialization, followed by calls to each of the state machines in the system. The rest of the program are the state machine subroutines and the device handling state machine.

Page A3 starts the listing for the auto address state machine (See Figure 1 for state machine diagram). The diagram shows nine states and the top six were implemented in this listing (Auto Multiple addressing was not required for our application). The subroutine starts by looking at the value AA which indicates what state this state machine is in. Then we jump to the proper state processing area.

Each state is coded into two sections, 0 and 1. For example the AAUS state has label AAUSØ and AAUS1 in it. Normally the program goes directly to the suffix label to check if conditions are such that a change in state is required. If a change in state is required, then the jump is made to the Ø suffix label. In this area any initialization is done, and the state value (AA in this case) is changed to reflect the new state.

Within each state, the arrows shown in the state diagram are checked to see if a change in state is required. For example, if we are in the AAUS state we check to see if a AAD command has been received and that the acceptor handshake machine is in the ACDS state. If this is true the Auto Address Machine moves to the AAIS state. Following this procedure the entire system is put together.

```
AVOCET SYSTEMS 8048 CROSS-ASSEMBLER -  VERSION 1.3

SOURCE FILE NAME: EPROM19.ASM


                ;
0200                    ORG     0200H
                ;
                ;
                ;
                ;AA - AUTO ADDRESS STATE MACHINE
                ;6 STATES AAUS, AAIS, AACS, ASIS, AWPS, ASCS
                ;
                ;JMP TO PROPER STATE
                ;
0200 B925       AASM:   MOV     R1,#AA
0202 F1                 MOV     A,@R1
0203 121C               JB0     AAUS1
0205 323C               JB1     AAIS1
0207 5258               JB2     AACS1
0209 7266               JB3     ASIS1
020B 927F               JB4     AWPS1
020D B298               JB5     AECS1
                ;
                ;AAUS STATE
                ;
                ;SET DEFAULT ADDRESSES
                ;
020F C5         AAUS0:  SEL     RB0                     ;SET SECONDARY ADDR TO DEFAULT
0210 BE1F               MOV     R6,#LSADDEF
0212 D5                 SEL     RB1
0213 231F               MOV     A,#LADRDEF              ;SET PRIMARY ADDRESS TO DEFAULT IN PIL
0215 B804               MOV     R0,#REG4
0217 D42E               CALL    WRPIL
0219 B101               MOV     @R1,#AAUS               ;SET AA=AAUS
021B 83                 RET                             ;RETURN TO PREVENT AAU.ACDS CHECK AGAIN TO INFINITE LOOP HERE
                ;
                ;IF AAU.ACDS THEN AAUS
                ;IF AAD.ACDS THEN AAIS
                ;IF AES.ACDS THEN ASIS
                ;
021C D483       AAUS1:  CALL    AAUCK                   ;CHECK FOR ACDS.CMD.AAU
021E 960F               JNZ     AAUS0                   ;JMP IF AAU
0220 D4AA       AAUS1A: CALL    RADCK                   ;CHECK FOR ACDS.RDY.VALID ADDR
0222 C632               JZ      AAUS1R                  ;RETURN IF NOT
0224 FA                 MOV     A,R2                    ;CHECK FOR AAD
0225 53E0               ANL     A,#AADMSK
0227 D380               XRL     A,#AAD
0229 C633               JZ      AAIS0                   ;JMP IF AAD
022B FA                 MOV     A,R2                    ;CHECK FOR AES
022C 53E0               ANL     A,#AESMSK
022E D3C0               XRL     A,#AES
0230 C65D               JZ      ASIS0                   ;JMP IF AES
0232 83         AAUS1R: RET
                ;
                ;AAIS STATE
                ;
                ;SET TMPADR (R7) = RMESS.ADR (STRIP NON ADDRESS BITS)
                ;INCR RMESS FOR NAA GENERATION
                ;
0233 FA         AAIS0:  MOV     A,R2                    ;SAVE ADDR IN R7
0234 531F               ANL     A,#ADRMSK
0236 C5                 SEL     RB0
0237 AF                 MOV     R7,A
0238 D5                 SEL     RB1
0239 1A                 INC     R2                      ;INCREMENT RMESS FOR NAA GENERATION
023A B102               MOV     @R1,#AAIS               ;SET AA=AAIS
                ;
                ;IF AAU.ACDS THEN AAUS
                ;IF NAA.DACS THEN AACS
                ;
023C D483       AAIS1:  CALL    AAUCK                   ;CHECK FOR AAU.ACDS
023E 960F               JNZ     AAUS0                   ;JMP IF TRUE
0240 FA                 MOV     A,R2                    ;CHECK FOR NAA, NOT CHECKING FOR RDY
0241 53E0               ANL     A,#AADMSK
0243 D3B0               XRL     A,#AAD
0245 964E               JNZ     AAIS1R                  ;RETURN IF -NAA
0247 B822               MOV     R0,#D                   ;CHECK FOR DACS
0249 F0                 MOV     A,@R0
024A 5302               ANL     A,#DACS
024C 964F               JNZ     AACS0                   ;JMP IF DACS
 24E 83         AAIS1R: RET
                ;
                ;AACS STATE
                ;
                ;SET REGISTER 4 OF PIL CHIP TO TMPADR
                ;
024F C5         AACS0:  SEL     RB0                     ;WRITE TMPADR TO PIL REG4
0250 FF                 MOV     A,R7
0251 D5                 SEL     RB1
0252 B804               MOV     R0,#REG4
0254 D42E               CALL    WRPIL
0256 B104               MOV     @R1,#AACS               ;SET AA=AACS
                ;
                ;IF AAU.ACDS THEN AAUS0
                ;
0258 D483       AACS1:  CALL    AAUCK                   ;CHECK FOR AAU.ACDS
025A 960F               JNZ     AAUS0                   ;JMP IF TRUE
025C 83                 RET
                ;
                ;ASIS STATE
                ;
                ;SET TMPADR=RMESS.ADRMSK (STRIP NON ADDRESS BITS)
                ;INCR RMESS FOR NES GENERATION
                ;
025D FA         ASIS0:  MOV     A,R2                    ;STORE RMESS.ADRMSK IN TMPADR
 25E 531F               ANL     A,#ADRMSK
 0260 C5                SEL     RB0
 0261 AF                MOV     R7,A
 0262 D5                SEL     RB1
 0263 1A                INC     R2                      ;INC RMESS FOR NES GENERATION
 0264 B108              MOV     @R1,#ASIS               ;SET AA=ASIS
                ;
                ;IF AAU.ACDS THEN AAUS0
                ;IF NES.DACS THEN AWPS0
                ;
 0266 D483      ASIS1:  CALL    AAUCK                   ;CHECK FOR AAU.ACDS
 0268 960F              JNZ     AAUS0
 026A FA                MOV     A,R2                    ;CHECK FOR NES, NOT CHECKING FOR RDY
 026B 53E0              ANL     A,#AESMSK
 026D D3C0              XRL     A,#AES
 026F 9678              JNZ     ASIS1R                  ;RETURN IF -NEW
 0271 B822              MOV     R0,#D                   ;CHECK FOR DACS
 0273 F0                MOV     A,@R0
 0274 5302              ANL     A,#DACS
 0276 9679              JNZ     AWPS0                   ;JMP IF TRUE
 0278 83        ASIS1R: RET
                ;
                ;AWPS STATE
                ;
                ;SET SECADR = TMPADR
                ;
 0279 C5        AWPS0:  SEL     RB0                     ;SET R6=R7
 027A FF                MOV     A,R7
 027B AE                MOV     R6,A
 027C D5                SEL     RB1
 027D B110              MOV     @R1,#AWPS               ;SET AA=AWPS
                ;
                ;IF AAU.ACDS THEN AAUS0
                ;IF AEP.ACDS THE AECS0
                ;
 027F D483      AWPS1:  CALL    AAUCK                   ;CHECK FOR AAU.ACDS
 0281 960F              JNZ     AAUS0                   ;JMP IF TRUE
 0283 D44A              CALL    RADCK                   ;CHECK FOR ACDS.RDY.VALID ADDR
 0285 C68E              JZ      AWPS1R                  ;RETURN IF FALSE
 0287 FA                MOV     A,R2                    ;CHECK FOR AEP
 0288 53E0              ANL     A,#AEPMSK
 028A D3A0              XRL     A,#AEP
 028C C68F              JZ      AECS0                   ;JMP IF TRUE
 028E 83        AWPS1R: RET
                ;
                ;AECS STATE
                ;
                ;SET REG4= RMESS.ADRMSK
                ;
 028F FA        AECS0:  MOV     A,R2                    ;SAVE PRIMARY ADDR TO PIL REG4
 0290 531F              ANL     A,#ADRMSK
 0292 B804              MOV     R0,#REG4
```

SOURCE FILE NAME: EPROM19.ASM

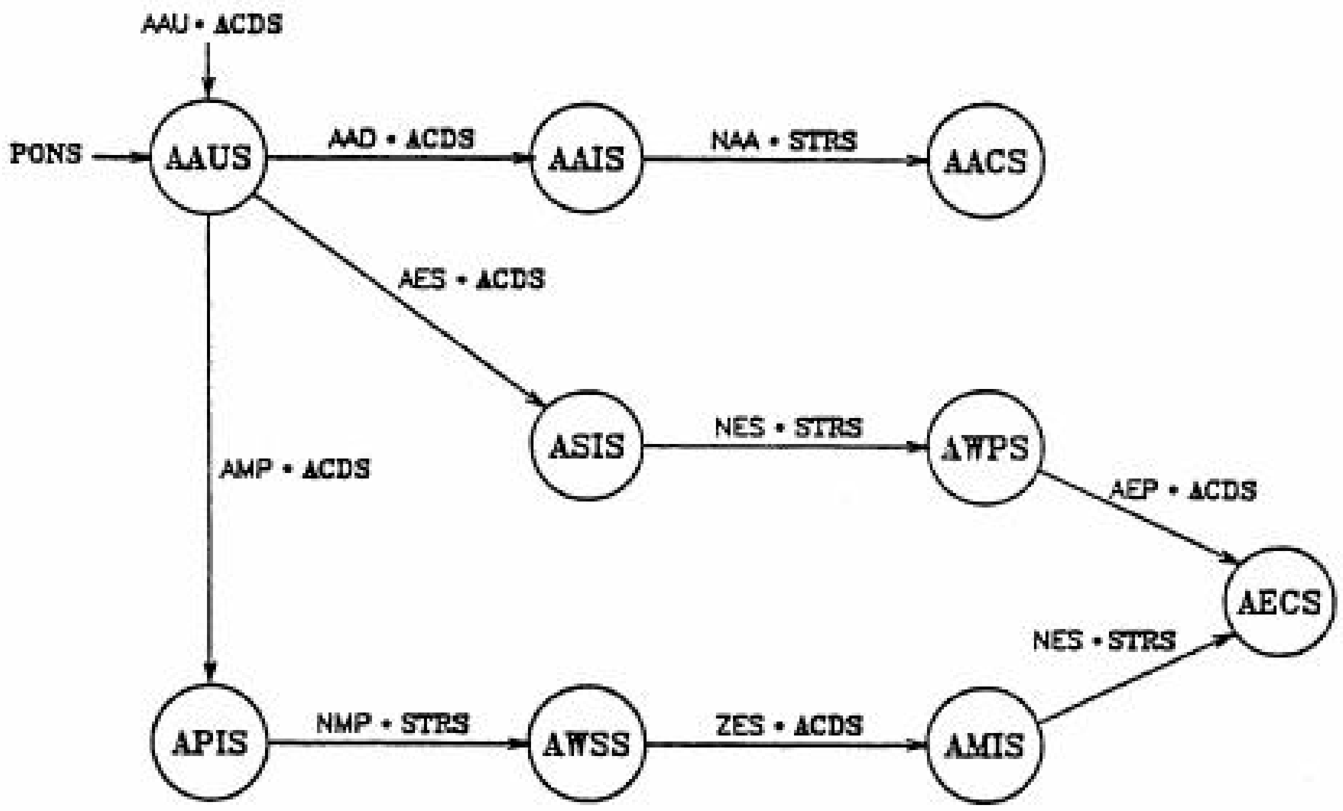
```
0294 342E             CALL    WRPIL
 296 B120             MOV     @R1,#AECS              ;SET AA=AECS
             ;
             ;IF AAU.ACDS THEN AAUS0
             ;
0298 3483    AECS1:   CALL    AAUCK                  ;CHECK FOR AAU.ACDS
029A 960F             JNZ     AAUS0                  ;JMP IF TRUE
029C 83               RET
```

AVOCET SYSTEMS 8048 CROSS-ASSEMBLER - VERSION 1.3

SOURCE FILE NAME: EPROM19.ASM

```
                ;
                ;
                ;
                ;***** PROGRAM STARTS HERE *****
                ;
                ;
                ;
                ;RESTART LOCATIONS
                ;
 0000                   ORG     0H
                ;POWER ON AND RESET START HERE
                ;
                ;INITIALIZE THE CHIP FOR POWER ON
                ;
 0000 D5        INIT:   SEL     RB1             ;NORMAL STATE FOR REGISTER BANK
 0001 E5                SEL     MB0             ;INITIAL BANK SELECT
 0002 142E              CALL    INITPC          ;INITIALIZE THE PIL CHIP
 0004 144B              CALL    INITSM          ;INITIALIZE THE STATE ACHINES
 0006 145B              CALL    INITAD          ;INITIALIZE LOOP ADDR FOR POWER UP
 0008 F5                SEL     MB1
 0009 1400              CALL    INITD           ;INITIALIZE THE DEVICE
 000B E5                SEL     MB0
                ;
                ;ALL STATE MACHINES ARE SERVICED AND THEN THE DEVICE IS SERVICED.
                ;THIS IS THE CORE TO THE SYSTEM, WITH THE REST BEING SUBROUTINES.
                ;
 000C D416      MAIN:   CALL    CHPSTS          ;GET STATUS OF PIL CHIP
 000E 1463              CALL    AHSM            ;CALL ACCEPTOR HANDSHAKE MACHINE
 0010 3400              CALL    SHSM            ;CALL SOURCE HANDSHAKE MACHINE
 0012 345B              CALL    DSM             ;CALL DRIVER MACHINE
 0014 B425              CALL    DCSM            ;CALL DEVICE CLEAR MACHINE
 0016 7467              CALL    DTSM            ;CALL DEVICE TRIGGER MACHINE
 0018 5400              CALL    AASM            ;CALL AUTO ADDRESS MACHINE
 001A 34A3              CALL    LESM            ;CALL LISTEN EXTENDED MACHINE
 001C 7408              CALL    LSM             ;CALL LISTEN MACHINE
 001E 549D              CALL    TESM            ;CALL TALKER EXTENDED MACHINE
 0020 9400              CALL    TSM             ;CALL TALKER MACHINE
 0022 748C              CALL    RLSSM           ;CALL REMOTE LOCAL SECONDARY MACHINE
 0024 74AB              CALL    RLSM            ;CALL REMOTE LOCAT MACHINE
 0026 F5                SEL     MB1
 0027 1417              CALL    DEVICE          ;CALL DEVICE SERVICE ROUTINE
 0029 E5                SEL     MB0
 002A D41C              CALL    CPHND           ;CALL THE PIL CHIP HANDSHAKE CHECKER
 002C 040C              JMP     MAIN            ;GO TO START OF MAIN CORE
```

Figure 1



Messages

AAD	auto address	NAA	next auto address
AAU	auto address unconfigure	NES	next extended secondary
AEP	auto extended primary	NMP	next multiple primary
AES	auto extended secondary	ZES	zero extended secondary
AMP	auto multiple primary		

Interface States

AACS	auto address configured state	ACDS	acceptor data state (from AH)
AAIS	auto address increment state (links to SH)	PONS	power on state (from PD)
AAUS	auto address unconfigured state (links to R)	STRS	source transfer state (from SH)
AECS	auto extended configured state. (links to L,T)		
AMIS	auto multiple increment state (links to SH)		
APIS	auto primary increment state (links to SH)		
ASIS	auto secondary increment state (links to SH)		
AWPS	auto wait for primary state (links to R)		
AWSS	auto wait for secondary state (links to R)		