Section 7

Display/Terminal Management Processor (TMP)

Section 7 Contents

National Semiconductor

TMP™

Terminal Management Processor

The TMP (NS405 series) is a single-chip CRT terminal display controller. The TMP is supported by the MOLE™ development system and replaces all the following LSI circuits commonly found in a terminal:

- Microprocessor
- Program ROM
- 64 x 8 RAM
- CRT controller
- DMA controller
- Character generator
- UART
- BAUD rate generator
- Parallel I/O controller
- Timer

The TMP offers complete CRT control over a wide scope of high-density circuit applications including phones, keyboard integration assignments, logic analyzers and more.

The NS455 Terminal Management Processor (TMP) demo board is available for design support.

Highly compact, the TMP board reduces previously necessary board space dramatically while providing 100% emulation of a classic low-end terminal. The board can also be used for TMP evaluation or as a vehicle for designing-in the NS405 device.

The board which is controlled by a preprogrammed NS455, needs only a video monitor, ASCII encoded keyboard, and power supply to provide your complete terminal. Should you wish to write your own program, no problem.

The cross-assembler software provides the capability. The board will execute custom programs through up to 8k of off-chip memory.

The TMP demo board comes complete with operating manual, program source listing, board schematic, board layout, and all necessary connectors.

When you're ready to design your own TMP system, turn to National's MOLE development system. By using this system-comprised of brain board, personality board and software—you bring dedicated development support to the TMP chip, making design-in extremely fast and simple.

National Semiconductor

PRELIMINARY

NS405-Series Display/Terminal Management Processor (TMP)

General Description

The NS405 is a CRT terminal controller on a chip. It is a microcomputer system which replaces the following LSI circuits commonly found in a CRT data terminal:

- Microcomputer
- CRT Controller
- DMA Controller
- Character Generator
- UART
- Baud Rate Generator
- Interrupt Controller
- Parallel I/O Controller
- Timer

In addition the NS405 includes powerful attribute logic, two graphics display modes, and fast video output circuits.

The NS405 is primarily intended for use in low-cost terminals, but contains many features which make it a superior building block for "smart" terminals and word processing systems.

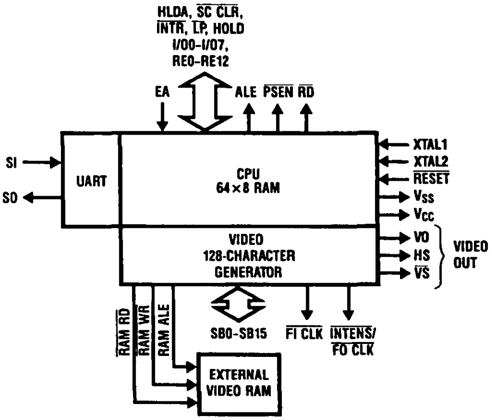
The NS405 interfaces easily to the display monitor, keyboard, display memory, and I/O ports. The architecture and instruction set are derived from the 8048-series microcontrollers. The instruction set has been enhanced and the architecture tailored to allow the NS405 CPU to efficiently manage a large display memory and an extensive interrupt environment.

The TMP can be used to easily and inexpensively add a display to many systems where it was previously impractical, it is not limited to terminal applications.

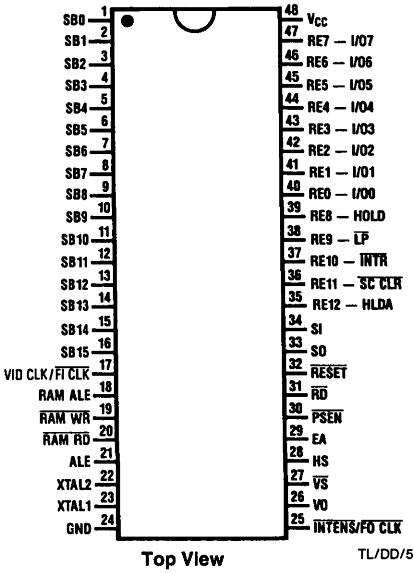
Features

- Enhanced 8048 instruction set and architecture
- Up to 8k x 8 ROM external with ROM expand bus
- On-board RAM 64 x 8
- Programmable display format
- On-board video memory management unit
- 16-bit bidirectional display memory bus (direct video and attribute RAM interface)
- Built-in timer
- Real-time clock (may be programmed for 1 Hz)
- Video control signals
- Eight independent attributes
- Pixel and block graphics display modes
- Programmable cursor characteristics
- Programmable CRT refresh rate
- Light pen feature
- UART, programmable baud rate up to 19.2k baud
- Character generator (128 characters 7 x 11 max)
- Single 5-volt supply @ 110 mA (typ)
- Up to 18 MHz video dot rate (12 MHz CPU clock)
- 48-pin package
- 8-bit parallel I/O port (multiplexed with external ROM)
- Extensive I/O expansion capabilities
- Up to 64k by 8 or 16 video RAM

Block and Connection Diagrams



TL/DD/5526-1



Top View

TL/DD/5526-2

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to V_{SS}*	−0.5V to +7.0V
Power Dissipation	1.5W
ESD	2000V

*EA, SI and VSYNC may be subjected to V_{SS} + 15V.

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operations should be limited to those conditions specified under DC Electrical Characteristics.*

DC Electrical Characteristics

T_A = 0°C to +70°C, V_{CC} = +5V ±10%, V_{SS} = 0V, unless otherwise specified

Symbol	Parameter	Test Conditions	Min	Max	Units
V_{IL1}	Input Low Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)		−0.5	0.8	V
V_{IH1}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)		2.0	V_{CC}	V
V_{IL2}	Input Low Voltage (XTAL1, XTAL2, $\overline{RESET}$)		−0.5	0.6	V
V_{IH2}	Input High Voltage (XTAL1, XTAL2, $\overline{RESET}$)		3.8	V_{CC}	V
V_{OL}	Output Low Voltage (All Except $\overline{INTENS}$, VO)	I_{OL} = 2.0 mA		0.4	V
V_{OH}	Output High Voltage (All Except $\overline{INTENS}$, VO)	I_{OH} = −125 μA	2.4	V_{CC}	V
V_{OL}	Output Low Voltage ($\overline{INTENS}$, VO)	I_{OL} = 5.0 mA		0.4	V
V_{OH}	Output High Voltage ($\overline{INTENS}$, VO)	I_{OH} = −500 μA	2.4		V
I_{IL}	Input Leakage Current (EA, $\overline{INT}$, SI)	$V_{SS} \leq V_{IN} \leq V_{CC}$		±10	μA
I_{OL}	Output Leakage Current (ROM Expand Bus, High Impedance State)	$V_{CC} \geq V_{IN} \geq V_{SS} + 0.45$		±10	μA
I_{OL}	Output Leakage Current (System Bus, High Impedance State)	$V_{CC} \geq V_{IN} \geq V_{SS} + 0.45$		±100	μA
I_{CC}	Total Supply Current	T_A = 25°C		150	mA

AC Electrical Characteristics

T_A = 0°C to +70°C, V_{CC} = +5V ±10%, V_{SS} = 0V, unless otherwise specified

Symbol	Parameter	Min	Max	Units
CPU AND ROM EXPAND BUS TIMING				
F_{XTAL}	Crystal Frequency	3	18	MHz
F_{CPU}	CPU Frequency	3	12	MHz
t_{CY}	CPU Cycle Time	1.25	7.5	μs
t_{DF}	Video Dot Time	55.5	333.3	ns
t_{LL}	ALE Pulse Width (Note 1)	125		ns
t_{AL}	Address Setup to ALE (Note 1)	55		ns
t_{LA}	Address Hold from ALE (Note 1)	40		ns
t_{CC}	Control Pulse Width $\overline{PSEN}$, $\overline{RD}$ (Note 1)	250		ns
t_{DR}	Data Hold (Notes 1, 4)	0	100	ns
t_{RD}	$\overline{PSEN}$, $\overline{RD}$ to Data In (Note 1)		220	ns
t_{AD}	Address Setup to Data In (Note 1)		360	ns
t_{AFC}	Address Float to $\overline{RD}$, $\overline{PSEN}$ (Notes 1, 5)	0		ns
t_{CAF}	$\overline{PSEN}$ to Address Float (Notes 1, 5)	−10	+10	ns
t_{DAL}	Data Setup to ALE (RE0–7, 11, 12) (Note 1)	55		ns
t_{ALD}	Data Hold from ALE (RE0–7, 11, 12) (Note 1)	40		ns
t_{CIS}	Control Input Setup to ALE (RE8, 9, 10) (Note 1)	240		ns
t_{CIH}	Control Input Hold from ALE (RE8, 9, 10) (Notes 1, 4)	75	125	ns

AC Electrical Characteristics

T_A = 0°C to +70°C, V_{CC} = +5V ±10%, V_{SS} = 0V, unless otherwise specified (Continued)

Symbol	Parameter	Min	Max	Units
SYSTEM BUS TIMING				
t_{EL}	RAM ALE Low Time (Note 1)	250		ns
t_{EH}	RAM ALE High Time (Note 1)	100		ns
t_{AS}	Address Setup to RAM ALE (Note 1)	20		ns
t_{AH}	Address Hold from RAM ALE (Note 1)	10		ns
t_{RR}	$\overline{\text{RAM RD}}$ Width (Note 1)	210		ns
t_{AR}	Address Setup to $\overline{\text{RAM RD}}$ (Note 1)	80		ns
t_{RRD}	Data Access from $\overline{\text{RAM RD}}$ (Note 1)		140	ns
t_{RDR}	Data Hold from $\overline{\text{RAM RD}}$ (Notes 1, 4)	0	60	ns
t_{WFI}	$\overline{\text{FIFO In Clock}}$ Width (Note 1)	210		ns
t_{WW}	$\overline{\text{RAM WR}}$ Strobe Width (Note 1)	130		ns
t_{AW}	Address Setup to $\overline{\text{RAM WR}}$ (Note 1)	120		ns
t_{DW}	Data Setup to $\overline{\text{RAM WR}}$ (Note 1)	10		ns
t_{WD}	Data Hold from $\overline{\text{RAM WR}}$ (Note 1)	20		ns
VIDEO TIMING				
t_{DF}	Dot Period = $\frac{1}{f_c}$ (Note 1)	55		ns
t_{VID}	Video Blank Time (Note 1)	5	15	ns
t_{VI}	Skew, $\overline{\text{Intensity}}$ to Dot 0 (Note 1)	−15	15	ns
t_{FOV}	$\overline{\text{FIFO Out Clock}}$ to Dot 0 (Note 1)		15	ns
t_{WFOH}	$\overline{\text{FIFO Out Clock}}$ Width High (Note 1, Note 2)	55	165	ns

*⅓ CPU cycle.

**1 Dot time is 55 ns.

Note 1: Control outputs C_L = 80 pF; ROM Expand Bus outputs C_L = 150 pF; System Bus outputs C_L = 100 pF; V_{OUT} & $\overline{\text{INTENS}}$ outputs C_L = 50 pF; F_{XTAL} = 18 MHz; F_{CPU} = 12 MHz. XTAL1 & XTAL2 driven externally per *Figure 12b* with 50% duty cycle.

Note 2: $\overline{\text{FO CLK}}$ duty cycle is shown above.

Note 3: Hold request is latched. It is honored at the start of the next vertical retrace.

Note 4: Max spec. listed for user information only, to prevent bus contention. Maximum value not tested.

Note 5: Not tested.

Character Cell Width	$\overline{\text{FIFO Out}}$ HIGH	$\overline{\text{FIFO Out}}$ LOW
6	1 dot	5 dots
7	2 dots	5 dots
8	2 dots	6 dots
9	3 dots	6 dots
10	3 dots	7 dots

Input Hold Times

T_A = 25°C, V_{CC} = +5V ±10%, V_{SS} = 0V

Input	Min Active Time
Reset	50 ms (power up) 5 CPU Cycles (after power up)
$\overline{\text{External Interrupt}}$	2 CPU Cycle
$\overline{\text{Light Pen}}$	1 CPU Cycle
I/O Input	1 CPU Cycle
Hold Request	1 CPU Cycle (Note 3)

FIFO

Fall through should not be greater than 4 character times (character time = $1/f_{XTAL}$ × #dots/cell).

Throughput rate must be at least the character rate (character rate = 1/character time).

Capacitance $T_A = 25°C, V_{CC} = V_{SS} = 0V$

Symbol	Parameter	Test Conditions	Min	Max	Units
C_{IN}	Input Capacitance	$F_C = 1$ MHz (Note 5)		10	pF
C_{OUT}	Output and Reset	Unmeasured Pins Returned to V_{SS} (Note 5)		20	pF

AC Electrical Characteristics in CPU Cycle Time

CPU AND ROM EXPAND BUS TIMING (FOR REFERENCE ONLY)

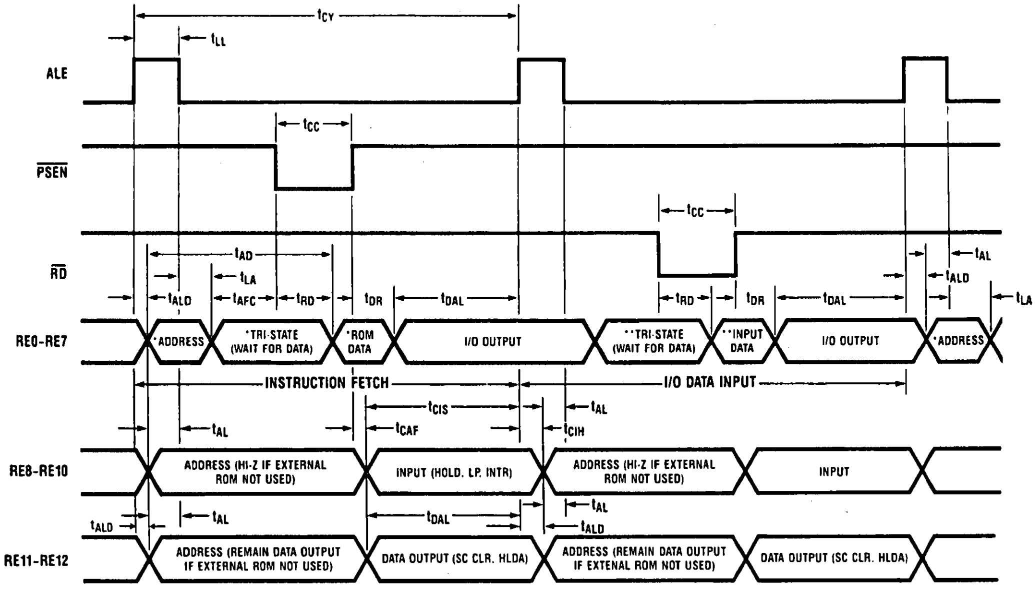
Symbol	Parameter		Typ
t_{LL}	ALE Pulse Width		$14\ t_{CY/60}$
t_{AL}	Address Setup to ALE		$8\ t_{CY/60}$
t_{LA}	Address Hold from ALE		$6\ t_{CY/60}$
t_{CC}	Control Pulse Width	$\overline{\text{PSEN}}$ $\overline{\text{RD}}$	$24\ t_{CY/60}$ $36\ t_{CY/60}$
t_{CY}	CPU Cycle Time		$60\ t_{CY/60} = 15/f_{CPU} = \dfrac{15}{f_{XTAL} \div 1 \text{ or } \div 1.5}$
t_{DR}	Data Hold		$-2\ t_{CY/60}$
t_{RD}	Control Pulse to Data In	$\overline{\text{PSEN}}$ $\overline{\text{RD}}$	$18\ t_{CY/60}$ $30\ t_{CY/60}$
t_{AD}	Address Setup to Data In		$32\ t_{CY/60}$
t_{AFC}	Address Float to	$\overline{\text{PSEN}}$ $\overline{\text{RD}}$	$2\ t_{CY/60}$ $2\ t_{CY/60}$
t_{CAF}	PSEN to Address Float		$0\ t_{CY/60}$
t_{DAL}	Data Setup to ALE	RE0–7 RE8–10 RE11–12	$6\ t_{CY/60}$ $-2\ t_{CY/60}$ $16\ t_{CY/60}$
t_{ALD}	Data Hold from ALE	RE0–7 RE8–12	$2\ t_{CY/60}$ $6\ t_{CY/60}$

SYSTEM BUS TIMING (FOR REFERENCE ONLY)

Symbol	Parameter	Ticks	
		Min	Max
t_{EL}	RAM ALE Low Time	$14\ t_{CY/60} - 42$ ns	
t_{EH}	RAM ALE High Time	$6\ t_{CY/60} - 25$ ns	
t_{AS}	Address Setup to RAM ALE	$4\ t_{CY/60} - 60$ ns	
t_{AH}	Address Hold from RAM ALE	$2\ t_{CY/60} - 40$ ns	
t_{RCY}	Read or Write Cycle Time		
t_{RR}	$\overline{\text{RAM RD}}$ Width	$12\ t_{CY/60} - 40$ ns	
t_{AR}	Address Setup to $\overline{\text{RAM RD}}$	$6\ t_{CY/60} - 45$ ns	
t_{RRD}	Data Access from $\overline{\text{RAM RD}}$		$10\ t_{CY/60} - 70$ ns
t_{RDR}	Data Hold from $\overline{\text{RAM RD}}$		
t_{WFI}	$\overline{\text{FIFO In Clock}}$ Width	$12\ t_{CY/60} - 40$ ns	
t_{WW}	$\overline{\text{RAM WR}}$ Strobe Width	$8\ t_{CY/60} - 27$ ns	
t_{AW}	Address Setup to $\overline{\text{RAM WR}}$	$10\ t_{CY/60} - 90$ ns	
t_{DW}	Data Setup to $\overline{\text{RAM WR}}$	$2\ t_{CY/60} - 30$ ns	
t_{WD}	Data Hold from $\overline{\text{RAM WR}}$	$2\ t_{CY/60} - 20$ ns	

Timing Waveforms

ROM Expand Bus Timing (In Port Instruction Is Shown)



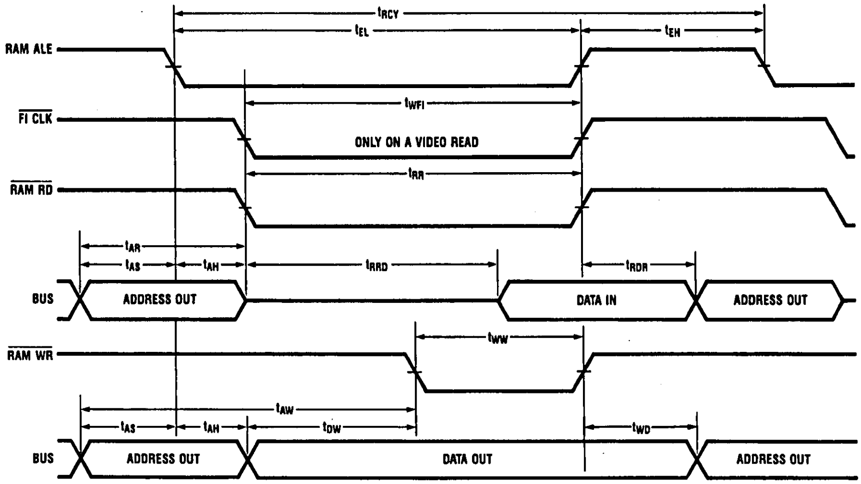
tCY
tLL
ALE
tCC
PSEN
tCC
RD
tAD
tLA
tALD
tAFC
tRD
tDR
tDAL
tRD
tDR
tDAL
tAL
tALD
tLA
RE0-RE7
*ADDRESS
*TRI-STATE (WAIT FOR DATA)
*ROM DATA
I/O OUTPUT
**TRI-STATE (WAIT FOR DATA)
**INPUT DATA
I/O OUTPUT
*ADDRESS
INSTRUCTION FETCH
I/O DATA INPUT
tCIS
tAL
tAL
tCAF
tCIH
RE8-RE10
ADDRESS (HI-Z IF EXTERNAL ROM NOT USED)
INPUT (HOLD, LP, INTR)
ADDRESS (HI-Z IF EXTERNAL ROM NOT USED)
INPUT
tALD
tAL
tDAL
tAL
tALD
RE11-RE12
ADDRESS (REMAIN DATA OUTPUT IF EXTERNAL ROM NOT USED)
DATA OUTPUT (SC CLR, HLDA)
ADDRESS (REMAIN DATA OUTPUT IF EXTENAL ROM NOT USED)
DATA OUTPUT (SC CLR, HLDA)

TL/DD/5526-3

*Remain I/O OUTPUT if External ROM not used.

**I/O Data input or 2nd ROM byte of 2 byte instruction. Otherwise remain I/O OUTPUT.

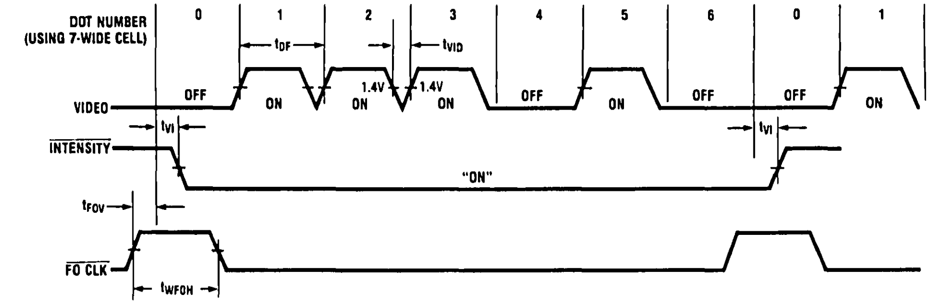
System Bus Timing



TL/DD/5526-4

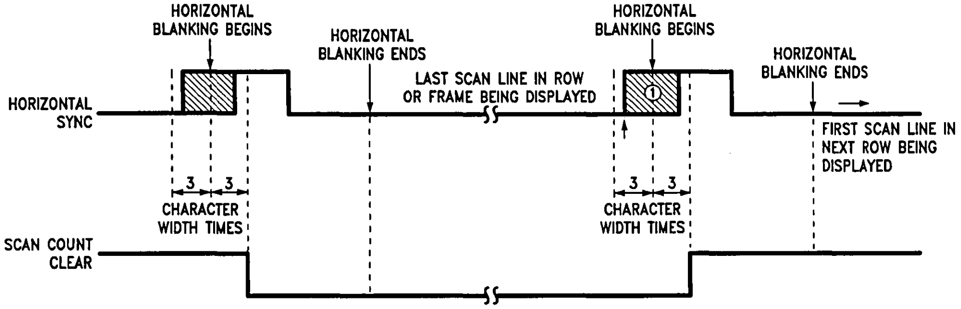
Timing Waveforms (Continued)

Video Timing



TL/DD/5526-5

Scan Count Clear Timing

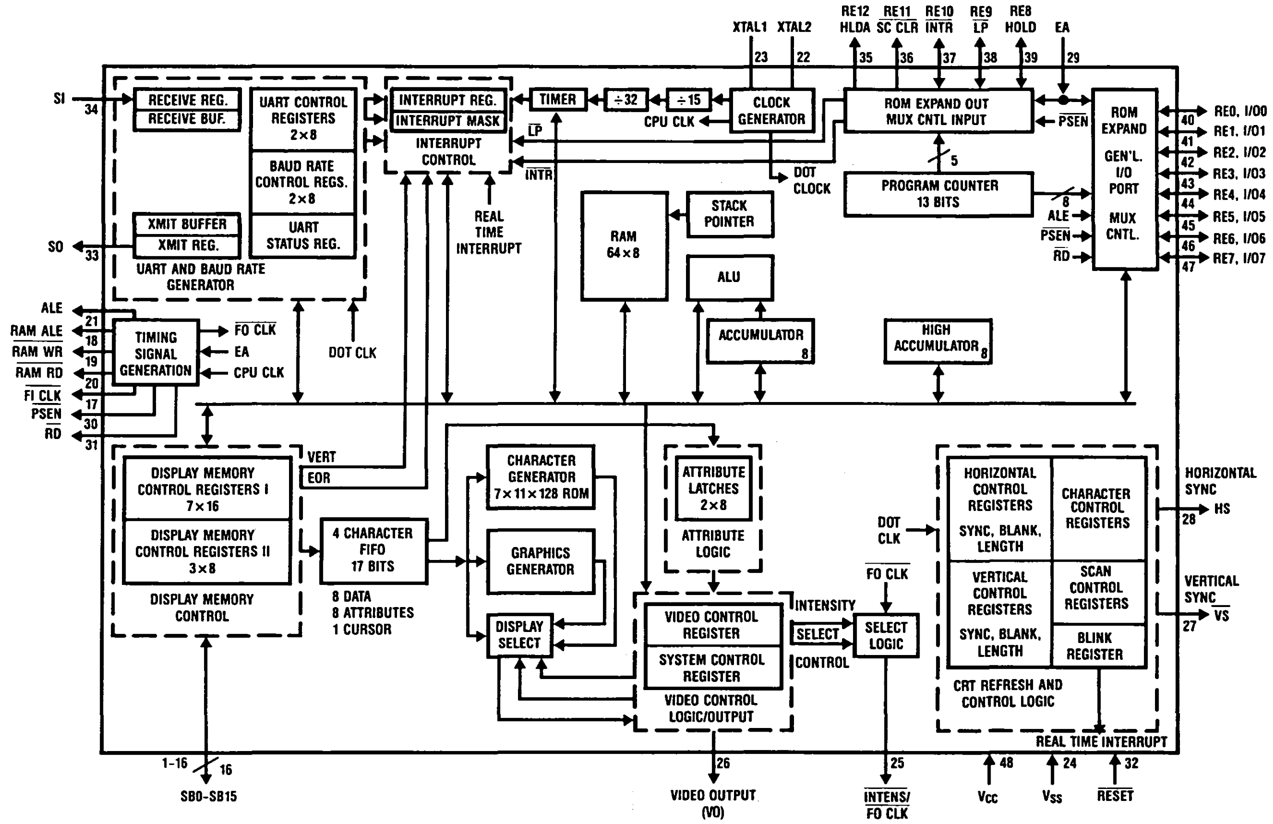


TL/DD/5526-6

For external character generation this edge is used to clock CLEAR into scan line counter. The edge must come before Scan Count Clear goes away, but not before the video controller has brought in all necessary display information for the last scan line.

Block Diagram

NS405-Series Detailed



TL/DD/5526-7

1.0 Functional Pin Descriptions

1.1 SUPPLIES

Pin	Name	Function
48	V_{CC} — Power	5V ±10%
24	V_{SS} — Ground Reference	
1.2 INPUT SIGNALS		
23, 22	XTAL1, XTAL2 — Crystal 1, 2:	Crystal connections for clock oscillator (3–18 MHz).
29	EA — External Access:	Pull HIGH (V_{IH_2})
32	$\overline{\text{RESET}}$	An active low input that initializes the processor. The $\overline{\text{RESET}}$ input is also used for internal ROM verification.
34	SI — Serial Input:	Drives receiver section of UART (true data).
1.3 OUTPUT SIGNALS		
33	SO — Serial Output:	Driven by transmitter section of UART (true data).
21	ALE — Address Latch Enable:	ROM address is available on the ROM Expand Bus and may be latched on the falling edge of ALE. Port output data may be latched on the rising edge of ALE. ALE pulses are always present, even if EA is tied low.
30	$\overline{\text{PSEN}}$ — Program Store Enable:	Enable external ROM output drivers when low. $\overline{\text{PSEN}}$ is idle (high) when the CPU fetches from internal ROM.
31	$\overline{\text{RD}}$ — Read Port Data:	Accept Port input data on ROM Expand Bus RE0–RE7 while low. ROM Expand Bus is in high impedance state while $\overline{\text{RD}}$ is low.
28	HS — Horizontal Sync	The rising edge of HS is controlled by the Horizontal Sync Begin Register and the falling edge is controlled by the Horizontal Sync End Register. HS is disabled (low) if bit 5 of the Video Control Register = 0.
27	$\overline{\text{VS}}$ — Vertical Sync Output:	The falling edge of $\overline{\text{VS}}$ is controlled by the Vertical Sync Begin Register and the rising edge is controlled by the Vertical Sync End Register. $\overline{\text{VS}}$ is at TRI-STATE if bit 5 of the Video Control Register = 0.
26	VO — Video Output:	High = beam on, low = beam off. VO is disabled (low) if bit 5 of the Video Control Register = 0.
25	$\overline{\text{INTENS}}/\overline{\text{FO CLK}}$	(Shared pin) $\overline{\text{INTENS}}$ Signal under attribute control may be used to switch the bistable brightness of display characters. $\overline{\text{FIFO Out Clock}}$ may be used to clock data from an external FIFO in synchronism with data from the internal FIFO. Both CANNOT be used simultaneously.
17	VID CLK/$\overline{\text{FI CLK}}$ — Video Dot Clock Out/ FIFO IN CLOCK	(Shared pin) The rising edge of the Video Dot Clock may be used to clock the data out of the video output pin. FIFO In Clock may be used to clock data from an extended attribute RAM into an external FIFO in synchronism with the data loaded into the internal FIFO. Both CANNOT be used simultaneously.
18	RAM ALE — RAM Address Latch Enable:	RAM address is available on the System Bus and may be latched on the falling edge of RAM ALE. Only operational when Display RAM accesses being performed. Otherwise high.
20	$\overline{\text{RAM RD}}$ — RAM Read:	Enable display RAM data onto the System Bus when $\overline{\text{RAM RD}}$ is low.
19	$\overline{\text{RAM WR}}$ — RAM Write:	Data to RAM is available on the System Bus and may be written at the rising edge of $\overline{\text{RAM WR}}$.
1.4 BUS — I/O		
1–8	SB0–SB7 — System Bus 0–7:	Display RAM address is output while RAM ALE is high and may be latched on the falling edge of RAM ALE. System Bus accepts data input while $\overline{\text{RAM RD}}$ is low and outputs data while $\overline{\text{RAM WR}}$ is low.
9–16	SB8–SB15 — System Bus 8–15:	Normally, Display RAM address is output and held on these pins for the full read or write cycle. However, if bit 4 of the System Control Register is set, these pins function bidirectionally like SB0–SB7 to allow 16-bit data words for attribute operation.
35–47	RE0–12 — ROM Expand Bus 0–12:	Used for program ROM expansion as described below. Time multiplexed with I/O port and system control signals. I/O port and system control signals only if no external ROM used.
40–47	RE0–RE7	Low order ROM address is output and may be latched on the falling edge of ALE. Enable ROM data to this Bus when $\overline{\text{PSEN}}$ is low. Enable I/O port input data to the Bus when $\overline{\text{RD}}$ is low. Use the rising edge of ALE to latch port output data.

1.0 Functional Pin Description (Continued)

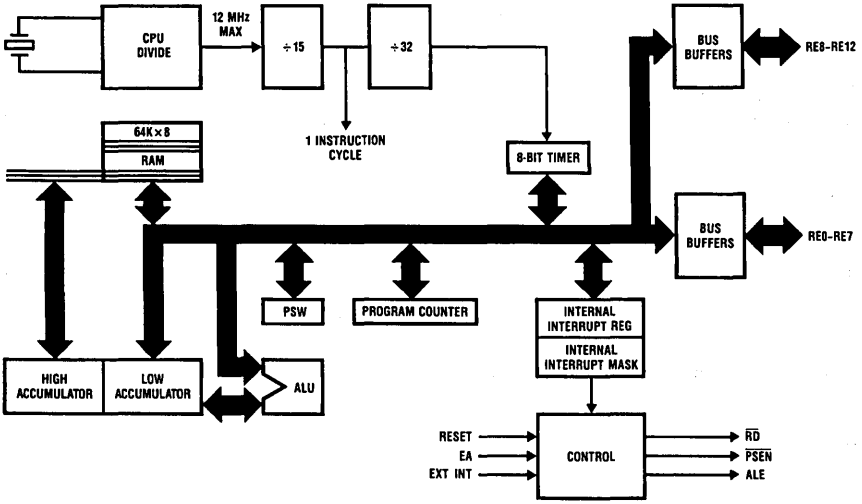
Pin	Name	Function
39–35	RE8–RE12	Five most significant bits of the ROM address are output during ALE and remain stable until data is read in during $\overline{\text{PSEN}}$. These pins are multiplexed with the HLDA, $\overline{\text{INTR}}$, $\overline{\text{LP}}$, $\overline{\text{SC CLR}}$, and HOLD signals.
37	$\overline{\text{INTR}}$ — Interrupt: RE10	An active low input that interrupts the processor if the external interrupt is enabled. Because it shares a pin with RE10, $\overline{\text{INTR}}$ may be driven directly only if no external ROM is used (EA is low). Otherwise must be driven through a 3.9k resistor.*
38	$\overline{\text{LP}}$ — Light Pen Interrupt: RE9	An active low input that interrupts the processor if internal interrupts are enabled and bit 5 in the Interrupt Mask Register is set. Because it shares a pin with RE9, $\overline{\text{LP}}$ may be driven directly only if EA is low. Otherwise, must be driven through a 3.9k resistor.*
39	HOLD — HOLD request: RE8	When high, requests that the NS405 enter the Hold mode. When in the Hold mode the System Bus will be in a high impedance state. The Hold mode is granted at the beginning of the next vertical retrace. Because it shares a pin with RE8, HOLD may be driven directly only if EA is low. Otherwise, must be driven through a 3.9k resistor.*
35	HLDA — Hold Acknowledge: RE12	This output is asserted in response to Hold and provides handshake capability with another processor (active high). For more detailed information see Section 3.0 Slave Processing. Because HLDA shares a pin with RE12, the HLDA state is preset only during the interval preceding the rising edge of ALE. However, if no external ROM is used, HLDA is a steady state output and need not be latched externally.
36	$\overline{\text{SC CLR}}$ — Scan Count Clear: RE11	This output clears an external scan counter when used with an external character generator. It is a low going pulse which occurs during the horizontal retrace preceding the first scan line of each character row. Because $\overline{\text{SC CLR}}$ shares a pin with the RE11, the correct $\overline{\text{SC CLR}}$ state is present only during the interval preceding the rising edge of ALE. However, if no external ROM is used, $\overline{\text{SC CLR}}$ is a steady state output and need not be latched externally.

*Unused control inputs must be terminated

2.0 Functional Description

2.1 CPU

The CPU of the NS405 is patterned after the 8048 single chip microcomputer (see *Figure 1).*



TL/DD/5526-8

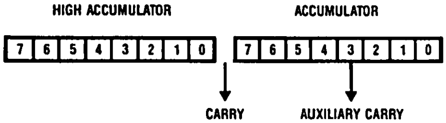
FIGURE 1. NS405 Series CPU Block Diagram

2.0 Functional Description (Continued)

2.1.1 Accumulator — High Accumulator

In addition to the regular 8-bit Accumulator, there is an 8-bit High Accumulator extension to facilitate the 16-bit operations required for display memory management. The HACC/ACC pair is usually used in conjunction with the 16-bit RAM pointer registers (RA, R0 and RB, R1, CURSOR, HOME, BEGD and ENDD) to effect video data transfers. In addition, external attribute memory is loaded in a 16-bit transfer operation. Any instruction which causes a carry or borrow out of the low accumulator will affect the high accumulator (see *Figure 2*).

Auxiliary carry is used only when converting the accumulator contents from binary to BCD (binary coded decimal) using the DA A instruction. The auxiliary carry flag can be cleared by moving a zero into bit 6 of the program status word.

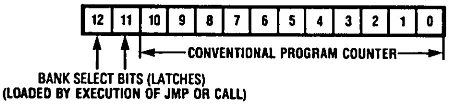


TL/DD/5526-9

FIGURE 2. CPU Accumulator

2.1.2 Program Counter (PC)

The Program Counter is a 13-bit wide register which provides program addressing for the CPU. The lower 11 bits operate like a conventional program counter while the upper 2 bits are actually latches. These 2 latches are automatically loaded from the bank select flip-flops (PSW bits 3, 4) whenever a JMP or CALL instruction is executed. The bank select flip-flops in turn are only modified upon the execution of a Select Memory Bank Instruction or modification of the PSW (see *Figure 3*).


12 11 10 9 8 7 6 5 4 3 2 1 0
CONVENTIONAL PROGRAM COUNTER
BANK SELECT BITS (LATCHES)
(LOADED BY EXECUTION OF JMP OR CALL)

TL/DD/5526-10

FIGURE 3. TMP Program Counter

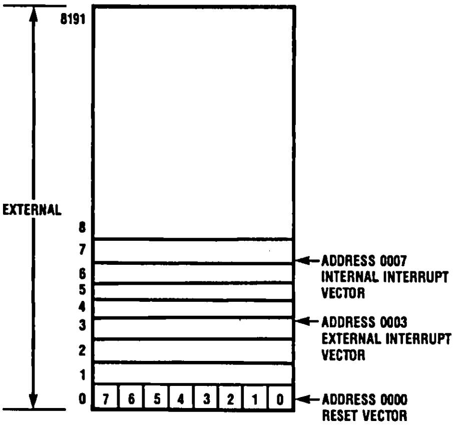
2.1.3 Program Memory

Memory is subdivided into 2k banks with accesses limited to the currently selected bank unless a Bank Change sequence has been executed. Upon reaching the end of a memory bank, the program counter will wrap around and point to the beginning of the current bank.

Each bank is further subdivided into pages of 256 bytes each, with 8 pages in every bank. The conditional JUMP instructions are restricted to operate within the memory page that they reside in.

Because of the sequence which the CALL instruction executes when pushing and loading the PC, it is possible to easily call and return from subroutines located in different memory banks (see *Figure 4*).

Upon executing an RET or RETR instruction for a call from one memory bank into another, a SEL MBx instruction should be excuted to restore the memory bank select flip-flops to their original bank. However, no SEL MBx is needed after an interrupt since the flip-flops were never modified.



TL/DD/5526-11

FIGURE 4. Program Memory Map

2.1.4 Program Status Word Bit Assignments

Bit Position	Contents
0	Stack Pointer Bit, S0
1	Stack Pointer Bit, S1
2	Stack Pointer Bit, S2
3*	Memory Bank Select Bit 0
4*	Memory Bank Select Bit 1
5*	Register Bank Select Bit (0 = Bank 0, 1 = Bank 1)
6*	Auxiliary Carry. A carry from Bit 3 to Bit 4 generated by an add operation. Used only by the decimal adjust (DA A) instruction.
7*	Carry. A bit indicating the preceding operation resulted in an overflow or an underflow from the 8-bit accumulator.

***Note 1:** Bits 3 through 7 are saved on the stack by subroutine calls or interrupts. Bits 3 and 4 are restored upon execution of an RET instruction, whereas all 5 bits are restored by RETR.

Note 2: F0 is not saved on the stack (as in an 8048).

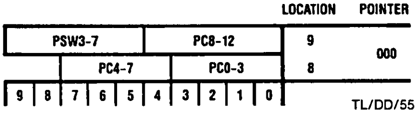
Note 3: Bits 0–5 cleared on a RESET.

2.1.5 Stack Pointer (SP)

The stack pointer is an independent 3-bit counter which points to designated locations in the internal RAM that holds subroutine return parameters. The stack itself is located in RAM locations 8–23 (see *Figure 5*).

Each entry in the stack takes up two bytes and contains both the PC and status bits. When reset to zero, the stack pointer actually points to locations 8 and 9 in RAM. Since the stack pointer is a simple up/down counter, an overflow will cause the deepest stack entry to be lost (the counter overflows from 111 to 000 and underflows from 000 to 111).

Note: If the level of subroutine nesting is less than eight (8), the unneeded stack locations may be used as RAM.



TL/DD/5526-12

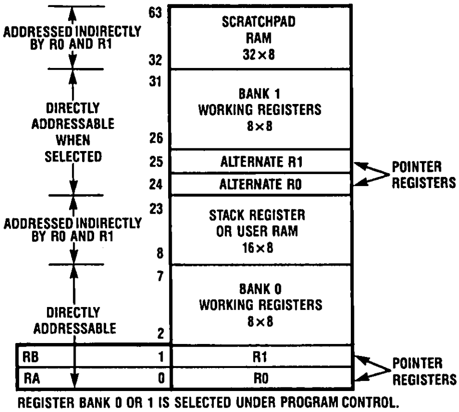
Note: The odd numbered RAM bytes in the stack area have two (2) extra bits to allow for storage of the bank select switch bits. This feature allows interrupt routines and subroutines to be located outside the current 2k program memory bank.

FIGURE 5. Typical Stack Composition

2.0 Functional Description (Continued)

2.1.6 Data Memory (On-Chip RAM)

The data memory nominally consists of 64 8-bit locations and is utilized for working registers, the subroutine stack, pointer registers and scratch pad. There are two sets of working/pointer registers (R0–R7) which are selected by the Select RAM Bank instruction. The stack area is located in locations 8–23. Locations 32–63 contain the scratch pad memory. To facilitate 16-bit Video Memory Management there are two 8-bit extension registers (RA and RB) which are associated with the R0 and R1 registers respectively of whichever RAM bank is currently selected (see *Figure 6*). i.e., There is only one RA register and only one RB register.

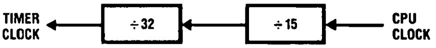

ADDRESSED INDIRECTLY BY R0 AND R1
63
32
SCRATCHPAD RAM 32×8
31
DIRECTLY ADDRESSABLE WHEN SELECTED
26
BANK 1 WORKING REGISTERS 8×8
25
ALTERNATE R1
24
ALTERNATE R0
POINTER REGISTERS
23
ADDRESSED INDIRECTLY BY R0 AND R1
STACK REGISTER OR USER RAM 16×8
8
7
DIRECTLY ADDRESSABLE
BANK 0 WORKING REGISTERS 8×8
2
RB
1
R1
RA
0
R0
POINTER REGISTERS
REGISTER BANK 0 OR 1 IS SELECTED UNDER PROGRAM CONTROL.

TL/DD/5526-13

FIGURE 6. RAM Memory Map

2.1.7 Timer

The On-Board Timer is an 8-bit up counter which sets the Timer Overflow Flag and generates an internal interrupt (if enabled) whenever it overflows from FF to zero. The Timer may be stopped, started, loaded and read from by the CPU. The Timer clock is derived from the CPU clock as shown in *Figure 7*. Whenever a Start Timer instruction is executed the ÷32 is initialized to its zero state to insure a full count measurement. After overflow the timer keeps counting until the next FF to zero overflow at which time the overflow flag will be set and another interrupt generated. The overflow flag can only be reset through the JTF and JNTF instructions.



TL/DD/5526-14

FIGURE 7. Timer Clock Generation

2.1.8 Interrupts

The interrupt circuitry handles two generic classes of interrupt conditions called Internal and External. Either class has its own master control which can be activated through software enable and disable instructions. On an interrupt service the currently executing instruction is completed, then two CPU cycles are used as the program counter and bits 3–7 of the PSW are pushed onto the stack and stack pointer is incremented. Then the interrupt vector address (3 or 7) is loaded into the PC and service started. Whenever an interrupt condition is being serviced all other interrupts of either class are locked out until a RETR instruction is executed to conclude interrupt service. If both an external and internal interrupt arrive at the same time, the external interrupt is recognized first.

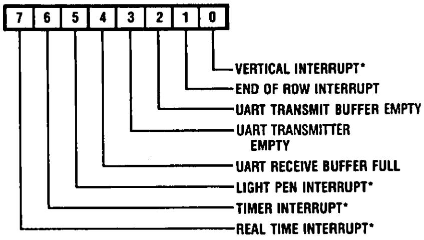
2.1.8.1 External Interrupt

The External Interrupt consists solely of the shared $\overline{\text{INTR}}$/RE10 pin. External interrupts on this pin will be detected if the setup and hold times as shown in the timing diagrams are met. This pin is a level sampled interrupt which means that as long as the pin is low during the sampling window an interrupt will be generated. In addition, the $\overline{\text{INTR}}$ pin is the only external pin whose logic state can be directly tested through software.

2.1.8.2 Internal Interrupts

The Internal Interrupts consist of seven internal operational conditions plus the light pen arranged in an 8-bit wide register as shown in *Figure 8*. Activation of an internal interrupt condition causes a corresponding register bit to be set, *Figure 9*. Each internal interrupt may be individually masked out through the Interrupt Mask register which has the same bit assignments as the Interrupt register and can be loaded from the accumulator. A zero in the Interrupt Mask register inhibits the interrupt and a one enables it. Further interrupt processing is as shown. To determine which of the eight internal conditions caused the interrupt the CPU must read the Interrupt register into the accumulator. To acknowledge receipt of the interrupt certain bits are automatically cleared on a read while others are reset upon service of the particular interrupt.

The conditions under which each of the interrupts are generated and cleared are as follows:



TL/DD/5526-16

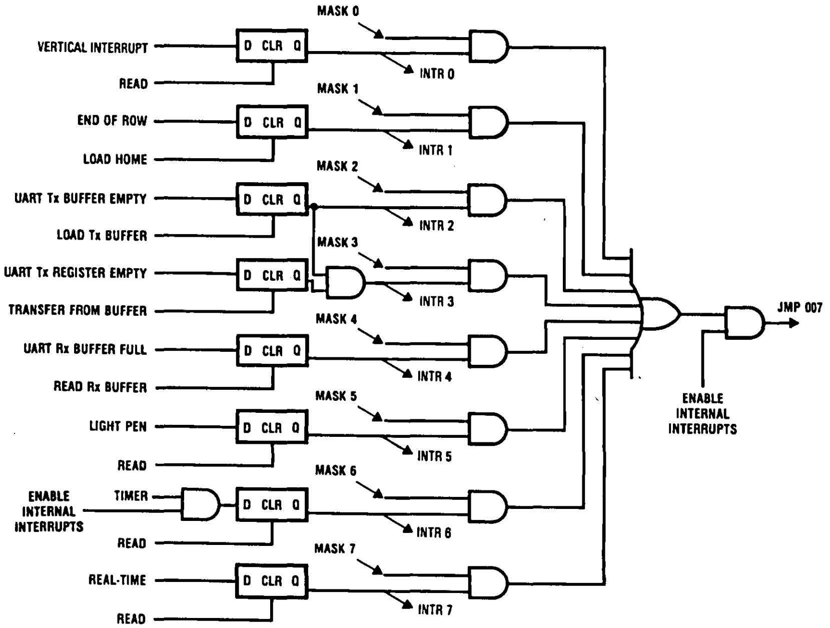
Note: The interrupt flags indicated by an asterisk (*) are cleared when the Interrupt Register is read.

FIGURE 8. Internal Interrupt Register

Bit

0 Vertical Interrupt—Generates an interrupt at the end of the display row designated by the Vertical Interrupt Register. Interrupt bit cleared on a CPU read of the interrupt register. If VIR > Vertical Length Register no interrupt will be generated.

2.0 Functional Description (Continued)



TL/DD/5526-15

FIGURE 9. Internal Interrupt Processing

Bit

1 End of Row Interrupt—Generates an interrupt at the end of each display row when the Current Row Start Register is updated for the next row. Used in conjunction with the Row Sequencing Control Bit (5) in the System Control Register to implement Row Pointer Look-Up Tables and Horizontally Split Screens. Interrupt bit cleared on a CPU write to the Home Register. Does not generate interrupts for those rows blanked during vertical blanking.

2 UART Transmit Buffer Empty—Generates an interrupt when the Transmit Buffer empties out after dumping a character into the Transmit Shift Register. Interrupt bit cleared on a CPU write to the Transmit Buffer.

3 Transmitter Empty—Generates an interrupt when BOTH the Transmit Buffer and Transmit Shift Register are empty. The interrupt bit is cleared when the CPU loads the transmit buffer.

4 UART Receiver Buffer Full—Generates an interrupt when the Receiver Buffer fills up with a character from the Receive Shift Register. Interrupt bit cleared on a CPU read of the Receiver Buffer.

5 Light Pen Interrupt—Generates an interrupt on each falling edge detected on the shared $\overline{LP}$/RE9 pin. Since only falling edges generate interrupts and the input is sampled each CPU Cycle, a high level must be sampled between falling edges in order to be considered a new interrupt. This interrupt is used to latch the light pen position registers. For further information see Light Pen Description. Interrupt bit cleared on a CPU read of the interrupt register.

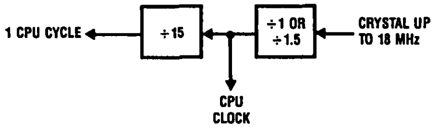
Bit

6 Timer Interrupt—Generates an interrupt when the internal 8-bit Timer overflows from FF to 00. Interrupt bit cleared on a CPU read of the interrupt register.

7 Real-Time Interrupt—Generates interrupts at a software programmable frequency that is generally in the Hertz range. (See CPU Clock Generation.) Thus permitting the implementation of a real-time clock or timer. Interrupt bit cleared on a CPU read of the interrupt register.

2.1.9 Clock Generation

All chip clocks are derived from the one external crystal connected between pins 22 and 23. This master clock also doubles as the video dot clock. The crystal frequency is constrained to lie within the range of 3 to 18 MHz. The CPU clock is derived from the crystal clock by either using it directly or by dividing down by a factor of 1.5 *(Figure 10).*



TL/DD/5526-17

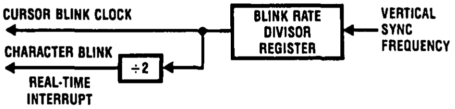
FIGURE 10. CPU Clock Generation

The choice is software programmable through bit 0 in the System Control Register. The exact selection is made in consideration of the fact that the CPU clock must lie within the range of 3 to 12 MHz. In addition, the choice of divide by modes will also impact the display character cell width due to the nature of the video controller. Specifically with ÷1.5

2.0 Functional Description (Continued)

the cell width must be ≥ 8 dots wide whereas with ÷1 the cell width must be ≥ 6 dots wide.

The low clock rates necessary to implement Cursor Blinking, Character Blinking and the Real-Time Interrupt are derived by passing the vertical sync frequency through a 5-bit Blink Rate Divisor Register, *(Figure 11)*. The resultant frequency is used as the Cursor Blink Clock. This clock is then further divided by 2 to yield the Character Blink and Real-Time Interrupt Clocks. For example, to get a 1 Hz real time interrupt, with a 60 Hz system, set the 5 bit Divisor Register to 30 in order to yield a 2 Hz signal which is then divided by 2.

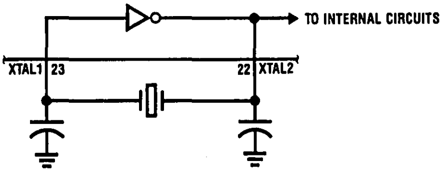


TL/DD/5526-18

FIGURE 11. Blink Clock Generation

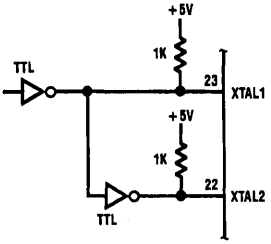
2.1.10 Oscillator Operation

The on-board oscillator circuit consists of a phase inverter which, when used with an external parallel resonant tank, *(Figure 12a)*, will yield the required oscillator clock. Crystals should be specified for AT cut and parallel resonant operation with the desired load capacitance (typically 20 pF). If one desires to externally generate the clock and input it to the chip, he may do so by driving XTAL1 (pin 23) and XTAL2 (pin 22) as shown in *Figure 12b*.



TL/DD/5526-19

FIGURE 12a. TMP Oscillator



TL/DD/5526-20

Note: Use AS TTL devices if faster than 12 MHz.

FIGURE 12b. External Oscillator Mode

2.2 DISPLAY MEMORY CONTROLLER

The video display data resides in the external Video Memory which is managed by the Display Memory Controller (DMC) through the System Bus. Either the CPU or the Video Controller may access the display memory by presenting its requests to the DMC. A maximum of three Video Memory accesses (Reads or Writes) can be performed by the DMC during each CPU instruction execution cycle. Because the CPU can access the Video Memory, one may expand CPU I/O or data memory by memory mapping into the Video Memory space. Up to 64k locations may be addressed over the 16-bit System Bus. Data word widths may be 8 or 16 bits depending upon whether external character attribute selection is used. The actual bus multiplexing mode is controlled by bit 4 in the System Control register. The Video Controller has the highest priority in obtaining Video Memory accesses with the CPU getting in on a space available basis. If all memory accesses are being taken by the Video Controller (rarely), the CPU is put into a wait state should it try to access video memory. To ease accessing requirements and boost throughput the Video Controller utilizes a 4-level data FIFO which is normally kept full of display data.

2.2.1 Display Memory Control Registers

In order to facilitate the management of video data for such features as a Screen scroll, memory paging and row lookup the DMC utilizes a number of registers which address the video RAM space. Each of these pointers is 16 bits wide and writable or readable from the 16-bit HACC/ACC pair as the case may be. There are 2 video data accessing modes as determined by bit 5 in the SCR, Sequential and Table Lookup. The functions of the pointer registers vary depending upon the accessing mode selected. Their designators are:

HOME = Home address register. Read and write.
BEGD = Beginning of diplay RAM. Write only.
ENDD = End of display RAM. Write only.
CURS = Cursor address register. Read, Write, Increment, Decrement.
SROW = Status section register. Write only.
CRSR = Current row start register. Not directly accessed.

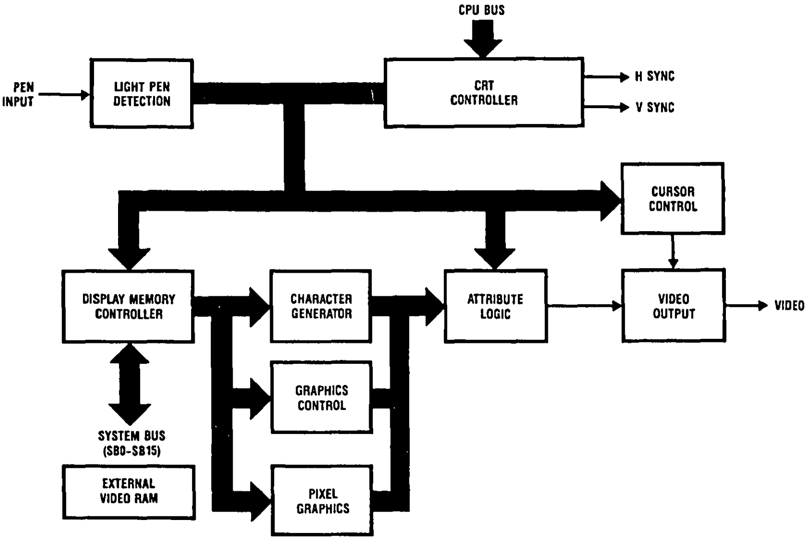
2.2.2 Sequential Access Mode

In this mode display data is accessed from sequential address locations in the video memory until the data requirements for the current screen field are fulfilled. The location from which the first display character is taken is the one pointed to by the HOME register. By modifying the contents of HOME one may implement a row scroll or paging operation. The BEGD and ENDD are used to control the wraparound condition when HOME gets near the end of available display RAM as determined by ENDD. In this instance, when sequential accessing brings us to the end of memory as pointed to by ENDD, the controller wraps around by jumping back to the beginning of display memory as pointed to by BEGD. The value in ENDD should be the last location in display memory + 1. Also the size of the display memory between BEGD and ENDD (ENDD − BEGD) must be an integral number of display rows. The CURS in both accessing modes merely identifies the current cursor position in display memory so that the cursor characteristics can be inserted into the video at the appropriate character position.

In addition to the display of normal video data one may elect to have a special status section displayed using data from a separate section of video memory. The status section would consist of an integral number of display rows on the bottom of the screen. This feature operates by reloading the video RAM pointer with the contents of SROW when the desired row position at which to start the status section comes up. The particular row at which the status display starts is defined in the Timing Chain. Once the video RAM pointer is jumped to SROW, data accessing again proceeds sequentially from there until the data requirements for the current field are satisfied.

2.0 Functional Description (Continued)

TMP Video Section



TL/DD/5526-21

Whether a status section is used or not, upon accessing all of the data necessary to display a field, the video RAM pointer is reset to HOME in preparation for the display of a new field.

2.2.3 Table Lookup Mode

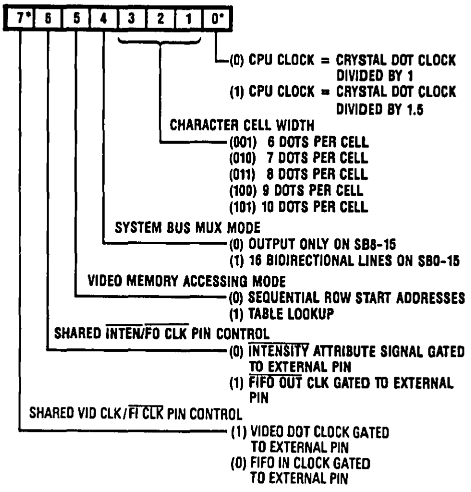
The CRSR (transparent to the user) is a pointer to the address of the first character in a display row. It is required because each time a scan line is displayed, all display characters in the row must be accessed anew. Since a row is made up of a number of scan lines, we must recover the address of the first character in the row for each scan in the row. After a row is done, the CRSR is normally advanced to point to the first character in the next row.

In table look-up mode the starting memory location of the next row is loaded into the CRSR from the HOME register at the end of each row. The HOME register was presumably updated by the CPU since the last end of row.

A CRSR load also generates the internal End of Row interrupt which the CPU will use as a signal to reload HOME. Finally, reloading HOME will clear out the End of Row interrupt. If the status section feature is used, upon reaching the begin status row location the CRSR will be loaded with SROW instead of HOME for that row. After which CRSR will revert back to load from HOME for the remaining rows on the screen.

2.3 SYSTEM CONTROL REGISTER

Through the System Control Register (SCR) the user specifies several important chip operational conditions. It is an 8-bit write only register which is loaded from the CPU accumulator.


7* 6 5 4 3 2 1 0*
(0) CPU CLOCK = CRYSTAL DOT CLOCK DIVIDED BY 1
(1) CPU CLOCK = CRYSTAL DOT CLOCK DIVIDED BY 1.5
CHARACTER CELL WIDTH
(001) 6 DOTS PER CELL
(010) 7 DOTS PER CELL
(011) 8 DOTS PER CELL
(100) 9 DOTS PER CELL
(101) 10 DOTS PER CELL
SYSTEM BUS MUX MODE
(0) OUTPUT ONLY ON SB8-15
(1) 16 BIDIRECTIONAL LINES ON SB0-15
VIDEO MEMORY ACCESSING MODE
(0) SEQUENTIAL ROW START ADDRESSES
(1) TABLE LOOKUP
SHARED INTEN/FO CLK PIN CONTROL
(0) INTENSITY ATTRIBUTE SIGNAL GATED TO EXTERNAL PIN
(1) FIFO OUT CLK GATED TO EXTERNAL PIN
SHARED VID CLK/FI CLK PIN CONTROL
(1) VIDEO DOT CLOCK GATED TO EXTERNAL PIN
(0) FIFO IN CLOCK GATED TO EXTERNAL PIN

TL/DD/5526-22

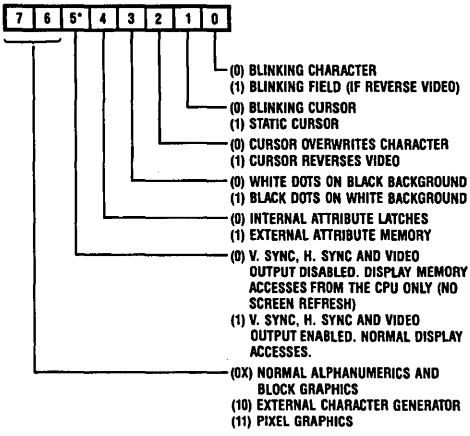
*Bit 0 is set to 1 by RESET and bit 7 is set to 0 by RESET.

FIGURE 13. System Control Register

2.4 VIDEO CONTROL REGISTER

Through the Video Control Register (VCR) the user specifies several video display features to the chip. It is an 8-bit write only register which is loaded from the CPU accumulator.

2.0 Functional Description (Continued)


7
6
5*
4
3
2
1
0
(0) BLINKING CHARACTER
(1) BLINKING FIELD (IF REVERSE VIDEO)
(0) BLINKING CURSOR
(1) STATIC CURSOR
(0) CURSOR OVERWRITES CHARACTER
(1) CURSOR REVERSES VIDEO
(0) WHITE DOTS ON BLACK BACKGROUND
(1) BLACK DOTS ON WHITE BACKGROUND
(0) INTERNAL ATTRIBUTE LATCHES
(1) EXTERNAL ATTRIBUTE MEMORY
(0) V. SYNC, H. SYNC AND VIDEO OUTPUT DISABLED. DISPLAY MEMORY ACCESSES FROM THE CPU ONLY (NO SCREEN REFRESH)
(1) V. SYNC, H. SYNC AND VIDEO OUTPUT ENABLED. NORMAL DISPLAY ACCESSES.
(0X) NORMAL ALPHANUMERICS AND BLOCK GRAPHICS
(10) EXTERNAL CHARACTER GENERATOR
(11) PIXEL GRAPHICS

TL/DD/5526-23

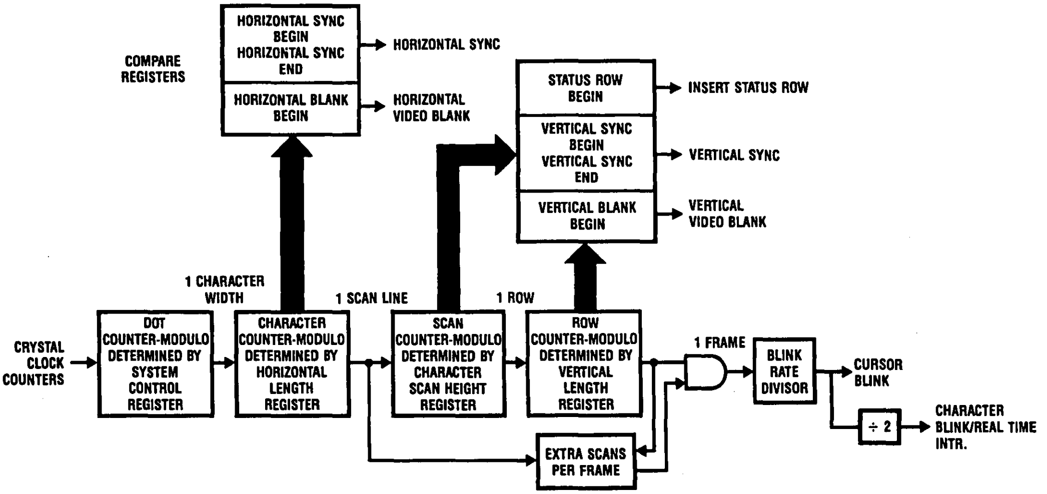
*Bit 5 is set to 0 by RESET.

FIGURE 14. Video Control Register

2.5 CRT REFRESH LOGIC

All video timing and clocking signals are derived from a series of counters and comparators called the Video Timing Chain. The chain is driven by the dot/crystal clock and ultimately divides down to the very slow blink clock, *(Figure 15)*. By having the program initialize the registers in the chain a user may specify all aspects of video generation.

The chain also controls the size and placement of the cursor and underline attribute within a character cell as well as the cell partitioning for block graphics display. All totaled, the chain consists of 14 wire only registers. They are loaded indirectly by using the Timing Chain Pointer (TCP), a 4-bit pointer to registers in the chain, and the MOV @TCP, A instruction.


COMPARE REGISTERS
HORIZONTAL SYNC BEGIN HORIZONTAL SYNC END
HORIZONTAL SYNC
HORIZONTAL BLANK BEGIN
HORIZONTAL VIDEO BLANK
STATUS ROW BEGIN
INSERT STATUS ROW
VERTICAL SYNC BEGIN VERTICAL SYNC END
VERTICAL SYNC
VERTICAL BLANK BEGIN
VERTICAL VIDEO BLANK
1 CHARACTER WIDTH
1 SCAN LINE
1 ROW
CRYSTAL CLOCK COUNTERS
DOT COUNTER-MODULO DETERMINED BY SYSTEM CONTROL REGISTER
CHARACTER COUNTER-MODULO DETERMINED BY HORIZONTAL LENGTH REGISTER
SCAN COUNTER-MODULO DETERMINED BY CHARACTER SCAN HEIGHT REGISTER
ROW COUNTER-MODULO DETERMINED BY VERTICAL LENGTH REGISTER
1 FRAME
BLINK RATE DIVISOR
CURSOR BLINK
÷ 2
CHARACTER BLINK/REAL TIME INTR.
EXTRA SCANS PER FRAME

TL/DD/5526-24

FIGURE 15. TMP Video Timing Chain

2.5.1 TMP Timing Chain Registers

TCP **Horizontal Timing**

0 Horizontal Length Register — HLR 7 bits

— Total number of character cells in a horizontal scan and retrace.

— Enter desired count − 1

1 Horizontal Blank Begin Register — HBR 7 bits (Characters/Row)

— Character position in horizontal scan after which horizontal blanking begins.

— Enter desired number of displayed characters/row − 1.

2 Horizontal Sync Begin Register — HSBR 7 bits

— Character position in horizontal scan after which horizontal sync begins (rising edge), HSBR ≤ HLR.

— Enter desired count + 2.

2.0 Functional Description (Continued)

2.5.1 TMP Timing Chain Registers (Continued)

TCP **Horizontal Timing**

3 Horizontal Sync End Register — HSER 7 bits

— Character position in horizontal scan after which horizontal sync ends (falling edge), HSER ≤ HLR.

— Enter desired count + 2.

Note: The polarity of the horizontal sync signal can be inverted by switching the values in the two horizontal sync registers.

TCP **Character Height Definition**

4 High Nibble Character Scan Height Register — CSHR 4 bits (see *Figure 16a*)

— Scan line height of a character cell.

— Enter desired number of scan lines − 1.

4 Low Nibble Extra Scans/Frame — ES/F 4 bits

— Number of extra scans to be added to a frame if desired.

— Enter desired number of extra scans −1.

— To get no extra scans make ES/F = CSHR. ES/F must be ≤ CSHR.

TCP **Vertical Timing**

5 Vertical Length Register — VLR 5 bits

— Total number of display and retrace rows in a frame.

— Enter desired number of rows − 1.

6 Vertical Blank Register — VBR 5 bits (Rows/Screen)

— Row position in vertical scan after which vertical blanking begins, VBR < VLR.

— Enter desired number of displayed rows − 1.

7 High Nibble Vertical Sync Begin Register — VSBR 4 bits

— Scan line position in first blank row at which vertical sync begins (falling edge). Sync starts 1 char time after blanking for that line starts (except when VSBR = CSHR sync will start 1 char time after blanking of the last displayed scan line).

— Enter desired scan line position − 1.

7 Low Nibble Vertical Sync End Register — VSER 4 bits

— Scan line position after start of vertical sync at which vertical sync ends (rising edge). Sync ends 1 char time after horizontal blanking for that scan line start.

— Enter desired scan line position − 1.

Note: If VSER = VSBR there will be no vertical sync signal.

8 Status Row Begin Register — SRBR 5 bits

— Row count after which the status row is inserted.

— Enter desired row position − 1.

TCP **Cursor and Graphics Control**

9 Upper 5 Bits Blink Rate 5 bits

— Divider driven by the vertical sync frequency to yield the slow cursor, character and real-time blink rates.

— Enter desired divisor − 1.

9 Lower 3 Bits Blink Duty Cycle 3 bits

— Approximate ON time of blink signal.

— 000 = shortest, 111 = longest (100 = 50% duty cycle).

10 Graphics Column Register — GCR 8 bits

— Assign dot positions to left, middle and right character cell columns for block graphics operation.

11 Graphics Row Register — GRR 8 bits

— Defines scan count at which middle row for block graphics characters begins (upper nibble) and at which bottom row begins (lower nibble). The middle row (upper nibble) must be ≥ 1.

— Enter desired scan count − 1.

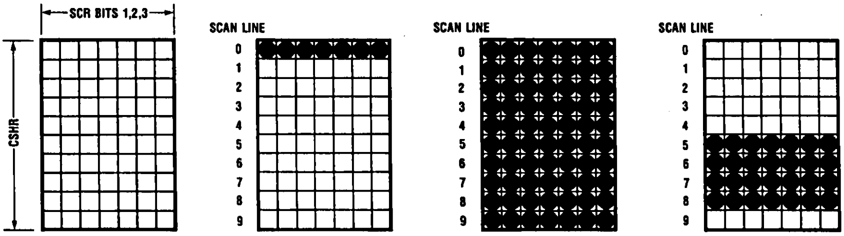
12 Underline Size Register — USR 8 bits (see *Figures 16a, b, c*)

— Defines the beginning (upper nibble) and ending (lower nibble) scan lines for the underline attribute. Values must be ≤ CSHR.

13 Cursor Size Register — CSR 8 bits (see *Figures 16a, b, c*)

— Defines the beginning (upper nibble) and ending (lower nibble) scan lines for the cursor. Values must be ≤ CSHR.

2.0 Functional Description (Continued)



a. Character Cell Format Specification

b. Underline Size Register = 90

c. Cursor Size Register = 9A (0B, 0C, 0D, 0E, 0F) may also be used

d. Cursor Size Register = 48

TL/DD/5526-25

FIGURE 16. Underline and Cursor Register Operation

Note: The internal cursor flip-flop gets set to ON whenever a scan line corresponding to the begin cursor nibble is reached, and gets set to cursor OFF whenever a scan line corresponding to the end cursor nibble is reached. The cursor attributes are inserted whenever the character position being displayed corresponds to the one pointed to by the cursor address register. A similar situation applies for characters with the underline attribute selected. Therefore, care should be taken when setting the ES/F register and setting the cursor and underline sizes. In particular the ES/F value should not be between the upper nibble and lower nibble values of the underline size register or between the upper nibble and lower nibble values of the cursor size register. To use the cursor as a pointer without displaying it, set the lower nibble of the cursor size register to a value less than CSHR and the upper nibble to a value greater than CSHR.

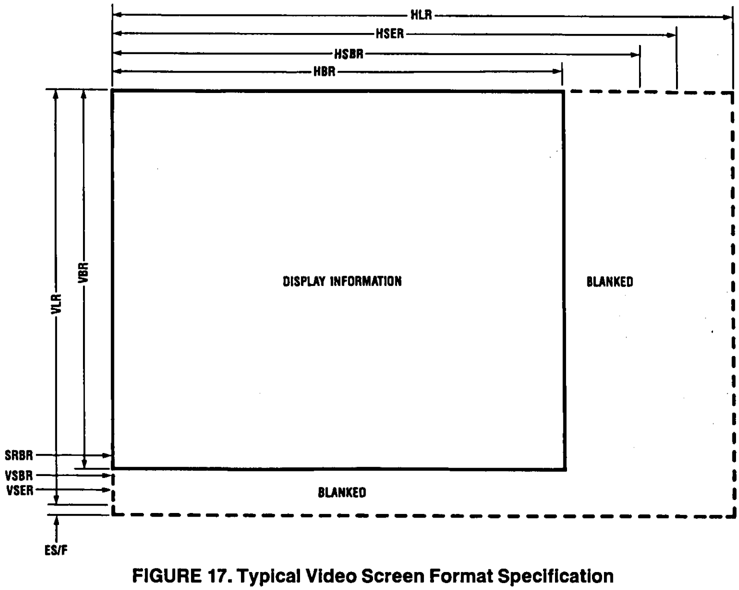
2.5.2 TIMING CHAIN LOAD VALUE EXAMPLE

It is desired to have a display field of 80 columns by 25 rows with the last screen row being a status row. It has been determined that 25 character width times will be necessary to complete horizontal retrace and that Horizontal sync should be positioned to start a full seven character times after blanking and end twenty characters after blanking to give us a total sync width of 13 character times. (See *Figure 17* for example.)

Additionally, vertical retrace will take 23 scan line times to complete with vertical sync starting three scan line times after vertical blanking begins and occupying a total period of 11 scan lines.

It is desired to make the character cells 12 scan lines tall. The cursor will be a block shape and occupy the bottom 11 scan lines in a cell. The underline attribute will actually be a strike through dash occupying the 4th scan line from the top in a cell.

Our line width is 80 displayed characters plus 25 for retrace making HLR = 80 + 25 − 1 = 104. Blanking will start after the 80th character so HBR = 80 − 1 = 79. To achieve seven character times after horizontal blanking, HSBR = 87 + 2 = 89. To achieve twenty character times after blanking HSER = 100 + 2 = 102 (note 102 − 89 = 13 total). Cell height is 12 lines so CSHR = 12 − 1 = 11. Since there are 12 scan lines per cell or row, vertical retrace will require 23/12 = 1 row and 11 scan lines. This makes our total row count VLR = 25 + 1 − 1 = 25 and ES/F = 11 − 1 = 10. Thus, timing chain location 4 would be coded: 1011 1010. We will display 25 rows so VBR = 25 − 1 = 24. Vertical sync will start at the beginning of the fourth scan



TL/DD/5526-26

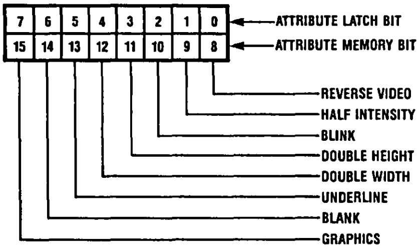
FIGURE 17. Typical Video Screen Format Specification

2.0 Functional Description (Continued)

line of the row after blanking begins so VSBR = 4 − 1 = 3. It will run for 11 scan lines or specifically the 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2 ending at the beginning of the 3rd so VSER = 3 − 1 = 2. The status row will be after the 24th so SRBR = 24 − 1 = 23. To specify the underline and cursor sizes one must remember that the first scan line is numbered 0. To get our 11 line block cursor we begin after the 0 line and end at the end of the 11 line making CSR = 0000 1011. The underline dash will be USR = 0011 0100. Note that the CSHR determines the scan counter modulo and if a scan compare register value (ES/F, VSBR, VSER, USR, CSR) is never reached, the signal end or begin will never be initiated.

2.6 ATTRIBUTES

Eight independent attributes may be inserted itno the video dot stream to affect display characters on either an individual or global basis. The eight attributes along with their control word bit assignments are detailed in *Figure 18.* The scope with which a particular set of attributes affects the display depends upon whether attribute control is internal or external as determined by bit 4 in the VCR.



TL/DD/5526-27

FIGURE 18. Attribute Bit Assignments

Attributes are present if the corresponding bit is a ZERO (low).

2.6.1 Internal Attribute Selection

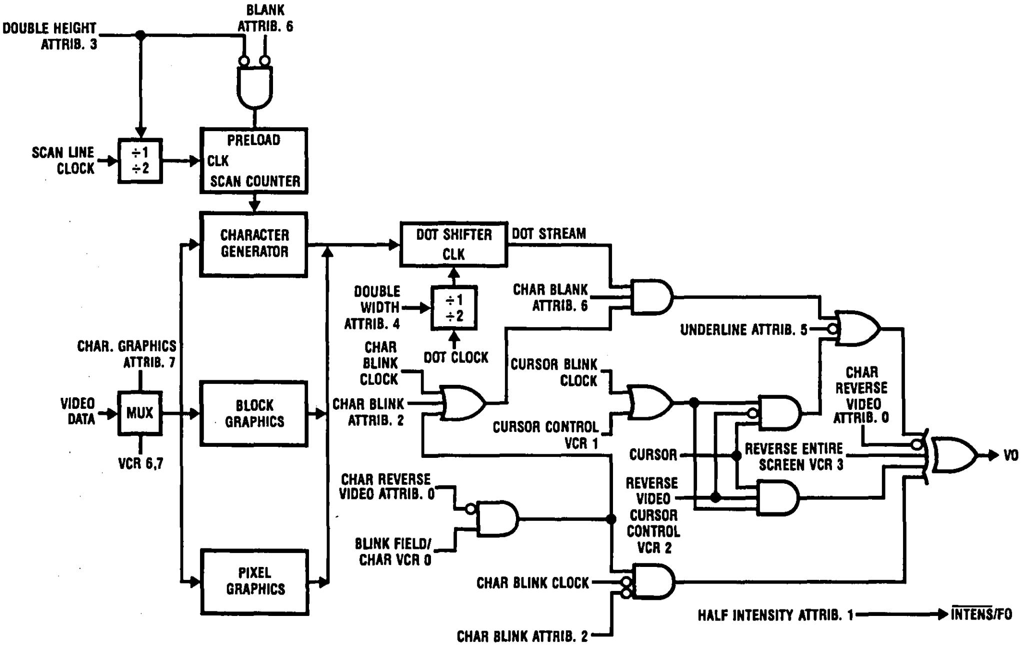
In internal mode attribute control comes from one of two internal attribute latches designated AL0 and AL1, either of which is directly loadable from the CPU accumulator. The choice of which of the two is used for a particular display character is determined by bit 7 (MSB) in the display memory data byte with 0 = AL0 and 1 = AL1. (Characters are represented in display memory as ASCII values occupying the low 7 bits of each 8-bit byte thus leaving bit 7 free for attribute control.)

2.6.2 External Attribute Selection

In external mode each display character has associated with it, a dedicated attribute field in the form of a high 8-bit extension to the regular display memory character byte. To use this mode the system bus msut be configured for 16-bit bidirectional operation (SCR bit 4 = 1) and external attributes must be selected (VCR bit 4 = 1).

2.6.3 Attribute Processing

Each of the eight attributes may be independently enabled thus yielding a number of possible combinations. The exact processing involved is shown in *Figure 19.* Note that attributes are always present. Whether any of them are active depends upon the particular control bit being enabled in the latch or memory.



TL/DD/5526-28

FIGURE 19. TMP Attribute Processing

2.0 Functional Description (Continued)

2.6.4 Attribute Operation

Reverse Video: A character and its surrounding cell are reversed in video from what was selected for the rest of the screen.

Half Intensity: To use the half intensity function the shared $\overline{\text{INTENSITY/FO CLK}}$ pin (25) must be selected for INTENSITY operation by setting SCR bit 6 low. In operation the half intensity pin will be low whenever a character for which the attribute is active is being displayed. To perform the actual attenuation function external circuitry must be connected between the $\overline{\text{INTEN}}$ and Video Output pins. In fact the signal may be used for another purpose such as switching between two colors.

Blink: A character or the field around it blinks as selected by VCR bit 0.

Double Height: A designated character is stretched out so that it will occupy a 2-row tall space. This attribute is implemented by slowing down by half the scan line stepping to the internal character generator. To use this attribute the desired double high character must be placed into the two display memory locations corresponding to the top and bottom row positions. For both locations the double high attribute is set. In addition the Blank attribute for the bottom character is also set to tell the controller it is the bottom half of a double high character. The double high attribute has no effect on element graphics or on pixel graphics displays. If an external character generator is used special circuitry must be employed to implement double high characters.

Double Width: A designated character is stretched out so that it will occupy a 2-character cell wide space. This attribute is implemented by slowing down by half the clock to the video dot shifter. To use this attribute the desired double wide character must be placed in the left character position and the double wide attribute bit set. The following character position (right) can have any character as it will be ignored.

Underline: If set this attribute causes the underline figure to be added to the video dot stream. Since the underline, like the cursor, can be specified as to position and size in the character cell, the underline can be an overline, block, strike through or any one of a number of effects. The underline overwrites any dot where it overlaps the character.

Blank/Double High Bottom: A character is inhibited from being displayed while still allowing it to be stored in the display memory. If this attribute and the double height attribute are set for the same character, the normal blank function is disabled for that character position and the character is displayed as the bottom half of a double height character.

Graphics: This attribute determines whether the video memory data byte as accessed by the display memory controller is routed through the character generator or block graphics control logic. If routed through the block graphics logic (attribute active) the effect on the video display will be as described in the Block Graphics section. Note that because Block Graphics mode is selected as an attribute it may be mixed in with normal alphanumerics characters. Also all other attributes with the exception of double height operate on the block graphics characters.

2.7 CHARACTER GENERATOR

The internal character generator holds 128 characters in a 7 x 11 matrix. The standard character sets are addressed using 7-bit ASCII codes stored in the display memory. When operating with fonts smaller than the maximum of 7 x 11, zeroes are encoded into the unused bits. When putting out a character the video controller always starts character generation on the second scan line of a row, leaving the first scan line blank. Similarly, the first (left) column in a character cell is blanked with character generation starting on the second column. Therefore, the specified cell size must be one greater in height and width than the display characters (including descenders) otherwise they will be chopped off. If the character cells are larger than the internal 7 x 11 matrix, blank dots will be put out after exhausting the internal generator (See *Figure 20* for example.)

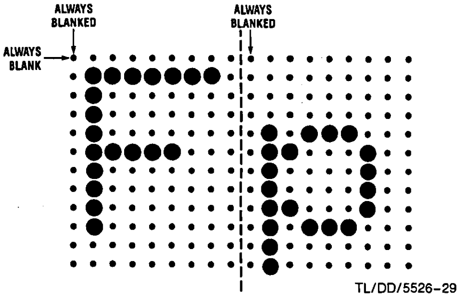


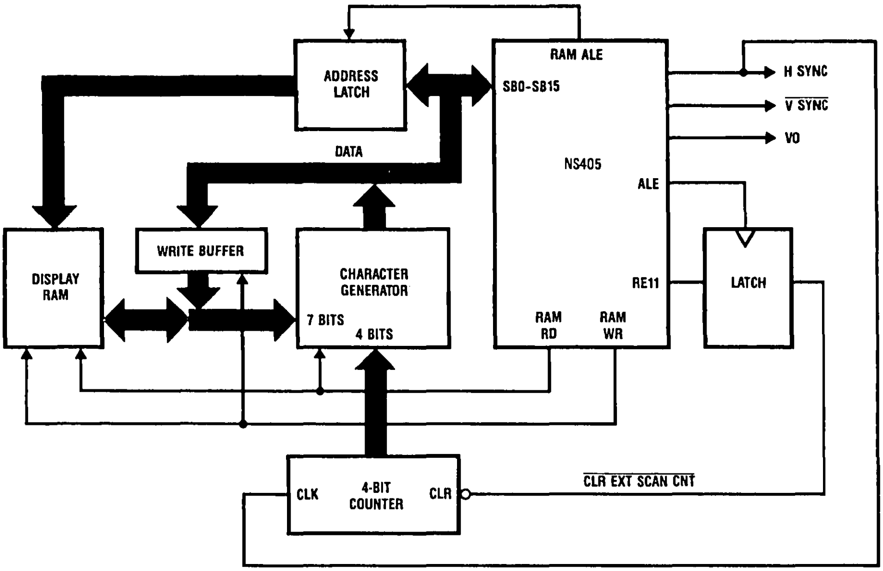
FIGURE 20. Character Cell Format

2.7.1 External Character Generation

The chip may be used with an external character generator by switching over to a pixel graphic display mode with modified address stepping as controlled by VCR bits 6, 7. In this mode an external character generator supplies pixel data to the chip as depicted in *Figure 21*. Character addressing comes from the display memory and scan line stepping from a 4-bit counter clocked by the Horizontal Sync. Scan line synchronization is achieved by using the Scan Count Clear signal coming out on RE11, pin 36. After the display of a row it pulses low to initialize the scan line counter for the start of a new row. In pixel mode both the character and any spacing between characters must be encoded into the external character generator. In addition, the chip will access and use at most 8 bits of pixel data for each character cell. However, if the cell width is specified to be 9 or 10, the ninth and tenth dots will repeat what was coded into the first. Therefore, assuming at least one dot spacing between characters, external fonts can at most be seven dots wide.

No limitations apply to the height of a character as long as the external generator can supply all of the scan lines as specified by the CSHR. As in regular pixel mode the LSB brought in is the first dot put out.

Since the eighth data bit is used for character generation it cannot effectively be used for internal attribute latch selection although one of the latches will be selected every data byte. Therefore, both internal attribute latches must be loaded with the same values. If external attribute operation is specified the full 8-bit high order attribute field is available for usage.

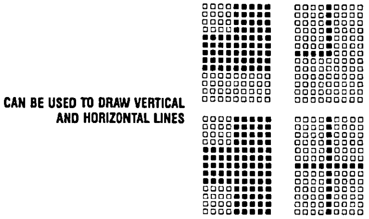


TL/DD/5526-30

FIGURE 21. External Character Set Implementation

2.8 BLOCK GRAPHICS

Block graphics is an alternative display mode to normal alphanumerics which is selected through attribute bit 7. Example *(Figure 22)*. It can operate on a character cell by character cell basis (see Attributes) and words by rerouting display memory bytes through the Block graphics logic instead of the internal character generator.



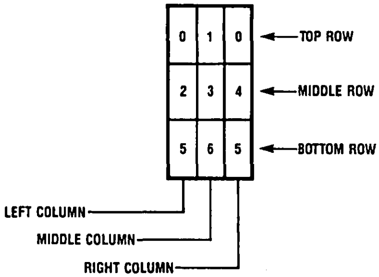
TL/DD/5526-31

FIGURE 22. Example Block Graphics Display Patterns

The Graphics Logic operates by partitioning the character cell space into nine possible areas as shown in *Figure 23* and then using the seven lower bits in the display data byte to turn these areas on or off. In this way one can draw contiguous lines or simple geometric figures while at the same time displaying alphanumeric characters in other cells.

The partitioning of the cell is controlled by two timing chain registers which specify two Horizontal and two Vertical cut off points to the graphics logic. Through these two registers one can make the sections as large or as small as desired, even eliminating sections entirely. Note that data bits 0 and 5 each control two sections as depicted in *Figure 23*.

2.8.1 Graphics Partitioning



TL/DD/5526-32

FIGURE 23. Block Graphics Cell Partitioning

The registers defining the graphics areas function as follows:

The Graphics Row Register — 8 bits (GRR) is divided into the following two (2) registers:

- Graphics Middle Row, (GMR):
  Defines the scan count at which the middle row begins (4 most significant bits of GRR).
- Graphics Bottom Row, (GBR):
  Defines the scan count at which the bottom row begins (4 least significant bits of GRR).

See *Figure 24.1a* for row example.

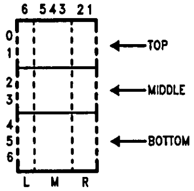
2.0 Functional Description (Continued)

The Graphics Column Register — 8 bits (GCR) controls vertical partitioning through bit patterns as follows: (See *Figure 24.*)

7	6	5	4	3	2	1	0	←GCR
L	L	L	R	R	X	X	X	←6 DOT FIELD WIDTH
L	L	L	R	R	R	X	X	←7 DOT FIELD WIDTH
L	L	L	R	R	R	R	X	←8 DOT FIELD WIDTH
L	L	L	L	R	R	R	R	←9 DOT FIELD WIDTH

TL/DD/5526-33

FIGURE 24. Block Graphics Column Partitioning



TL/DD/5526-44

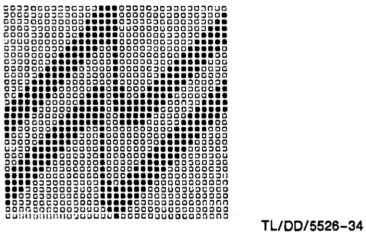
GRR = 24
GCR = 60 (0110 0XXX)
cell size = 6 x 7

FIGURE 24.1a Block Graphics Example

For all bits in the Graphics Column Register, a one assigns that bit position to the middle column. A zero in an L bit position assigns that bit position to the left column. A zero in an R bit position assigns that bit position to the right column. There is always at least one middle dot although the left and right sections may be eliminated entirely. For 10 dot wide cells the 10th bit will repeat the 9th bit. An easy way to determine the column partitioning is to fill the GCR with all ones, thereby making it one large middle section. Then, starting from the outermost L and R bit positions, put zeros in until the left and right sections are the sizes needed.

2.9 PIXEL GRAPHICS

When bits 6 and 7 of the Video Control Register are both set to 1, the character generator and block graphics circuits are disabled. Video output directly reflects the contents of the display memory byte on a pixel (dot) per bit basis with data output LSB first. Example *(Figure 25).*



TL/DD/5526-34

FIGURE 25. Example Pixel Graphics

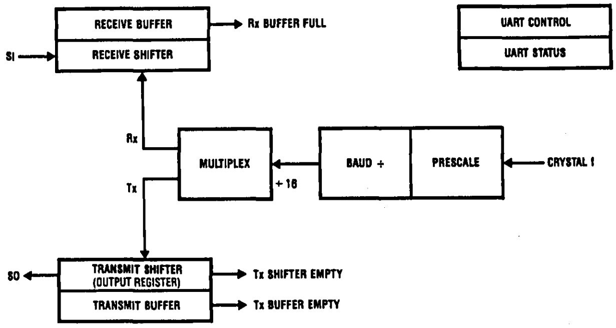
Nine bits at a time are accessed from each video memory location with as many bits being used as defined in the character cell width specification. If a cell width of 10 is specified the 10th bit will merely repeat the 9th bit. Attributes are still operable in pixel mode, on a data byte basis, with internal and external operation possible. With internal attribute latch operation the same values must be loaded into both latches since the usual latch select bit is now being used for pixel control. Unless, however, only a 7 dot wide cell is used leaving the 8th bit free. With external attribute operation we are now limited to a 7-bit attribute field since pixel data can now occupy 9 of the 16 bus bits. Because of this the LSB attribute, Reverse Video is totally disabled from operation in Pixel Graphic mode. This also applies to internal attribute latch operation. Note, however, that reverse entire screen video is still operable. Address sequencing through the video memory is sequential with as many data bytes being read in as is necessary to satisfy the pixel requirements of the screen.

2.10 LIGHT PEN

Activation of the light pen interrupt causes the horizontal and vertical screen position of the currently displayed character to be latched into the Horizontal Light Pen Register HPEN (7 bits) and Vertical Light Pen Register VPEN (5 bits) respectively. Both HPEN and VPEN may be read into the CPU accumulator. The values latched remain in VPEN and HPEN until another light pen interrupt latches new values.

2.11 UART

The UART features full duplex operation with double buffered Receive and Transmit sections. Baud rate generation is fully programmable through a 2-stage divider chain. CPU control of the UART is extensive with polled or interrupt driven operation possible.



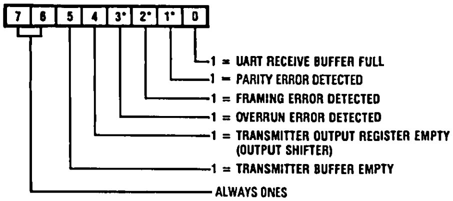
TL/DD/5526-35

FIGURE 26. TMP UART Block Diagram

2.0 Functional Description (Continued)

2.11.1 UART Control

UART Status Register (STAT): Contains error and status bits which reflect the internal state of the UART. Read into CPU accumulator. Bits 0, 5 are the same as those found in the internal interrupt register.


7
6
5
4
3*
2*
1*
0
1 = UART RECEIVE BUFFER FULL
1 = PARITY ERROR DETECTED
1 = FRAMING ERROR DETECTED
1 = OVERRUN ERROR DETECTED
1 = TRANSMITTER OUTPUT REGISTER EMPTY (OUTPUT SHIFTER)
1 = TRANSMITTER BUFFER EMPTY
ALWAYS ONES

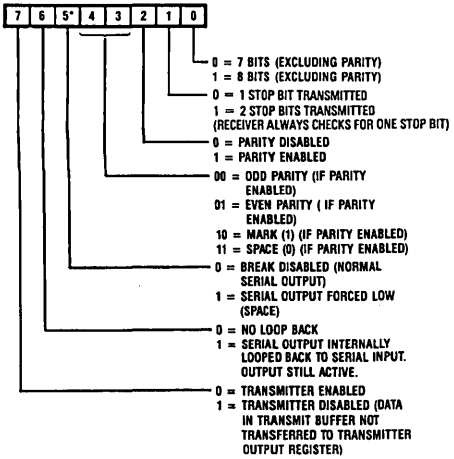
TL/DD/5526-36

UART Status Register bits 1, 2, 3 are only cleared on a chip reset or a read of the UART Receive Buffer. If another word were to come in before the Receive Buffer could be read the errors associated with the new word would add to those already present. The receipt of a new word can cause the three bits to go from a 0 to a 1, but not from a 1 to a 0.

FIGURE 27. UART Status Register

Note: The Transmit Output Register Empty flag is set to one whenever the transmitter is idle. The flag is reset to zero when a data character is transferred from the Transmit Buffer to the Output Register. This transfer does not occur until the next rising edge of the internal UART Transmit Clock. The Transmitter Output Register Empty flag occurs at the beginning of the last stop bit.

UART Control Register (UCR): Contains control bits which configure the format of transmitted data and tests made upon received data. Written to from CPU accumulator.


7
6
5*
4
3
2
1
0
0 = 7 BITS (EXCLUDING PARITY)
1 = 8 BITS (EXCLUDING PARITY)
0 = 1 STOP BIT TRANSMITTED
1 = 2 STOP BITS TRANSMITTED
(RECEIVER ALWAYS CHECKS FOR ONE STOP BIT)
0 = PARITY DISABLED
1 = PARITY ENABLED
00 = ODD PARITY (IF PARITY ENABLED)
01 = EVEN PARITY (IF PARITY ENABLED)
10 = MARK (1) (IF PARITY ENABLED)
11 = SPACE (0) (IF PARITY ENABLED)
0 = BREAK DISABLED (NORMAL SERIAL OUTPUT)
1 = SERIAL OUTPUT FORCED LOW (SPACE)
0 = NO LOOP BACK
1 = SERIAL OUTPUT INTERNALLY LOOPED BACK TO SERIAL INPUT. OUTPUT STILL ACTIVE.
0 = TRANSMITTER ENABLED
1 = TRANSMITTER DISABLED (DATA IN TRANSMIT BUFFER NOT TRANSFERRED TO TRANSMITTER OUTPUT REGISTER)

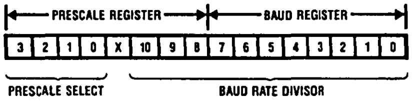
TL/DD/5526-37

*Bit 5 set to 0 by RESET.

FIGURE 28. UART Control Register

2.11.2 Baud Clock Generation

The basic BAUD clock is derived from the crystal frequency through a two-stage divider chain consisting of a 3.5–11 prescale and an 11-bit binary counter. *(Figure 29).* The divide factors are specified through 2 write only registers shown in *Figure 30.* Note that the 11-bit Baud Rate Divisior spills over into the Prescale Select Register. The correspondences between the 4-bit Prescale Select and Prescale factors is shown in Table I. There are many ways to calculate the two divisor factors but one particularly effective method would be to try to achieve a 1.8432 MHz frequency coming out of the first stage then use the BAUD Rate Divisor factors shown in Table II.



TL/DD/5526-39

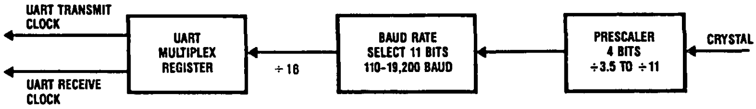
FIGURE 30. UART BAUD Clock Divider Registers

TABLE I. Prescale Factors

Prescale Select	Prescale Factor
0000	3.5
0001	4
0010	4.5
0011	5
0100	5.5
0101	6
0110	6.5
0111	7
1000	7.5
1001	8
1010	8.5
1011	9
1100	9.5
1101	10
1110	10.5
1111	11

TABLE II. Baud Rate Divisors (1.8432 MHz Input)

Baud Rate	Baud Rate Divisor (N − 1)
110 (110.03)	1046
134.5 (134.58)	855
150	767
300	383
600	191
1200	95
1800	63
2400	47
3600	31
4800	23
7200	15
9600	11
19200	5



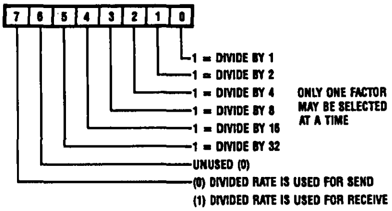
TL/DD/5526-38

FIGURE 29. UART BAUD Clock Generation

2.0 Functional Description (Continued)

The frequency coming out of the BAUD Rate Divisor is then passed through the UART Multiplex Register. Through the UART Multiplex Register one can specify that the Transmitter or Receiver clock be the same or a power of two multiple of the other.

UART Multiplex Register (UMX): Contains the bits which determine the divisor which is used to count down from the primary baud rate when different rates are used for send and receive (eight bits).


7
6
5
4
3
2
1
0
1 = DIVIDE BY 1
1 = DIVIDE BY 2
1 = DIVIDE BY 4
1 = DIVIDE BY 8
1 = DIVIDE BY 16
1 = DIVIDE BY 32
ONLY ONE FACTOR MAY BE SELECTED AT A TIME
UNUSED (0)
(0) DIVIDED RATE IS USED FOR SEND
(1) DIVIDED RATE IS USED FOR RECEIVE

TL/DD/5526-40

FIGURE 31. UART Multiplex Register

The actual baud rate may be found from:

BR = Fc/(16*N*P*D)

Where:

BR is the Baud Rate

Fc is the external crystal frequency

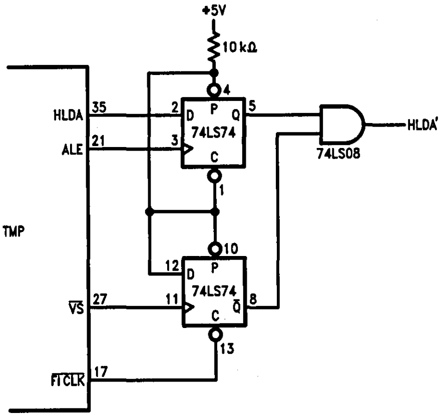
N is one plus the value of the Baud Rate Divisor contained in the Baud Rate Select Register and the Prescale Select Register.

P is the Prescale Divide Factor Selected by the value in the Prescale Select Register.

D is the Multiplex Register Divide Factor

3.0 Slave Processing

The TMP may be used as a slave video controller by having a host system perform Direct Memory Accesses into the display RAM. To assist in implementing such a system the chip features two DMA control pins—HOLD (Hold Request) and HLDA (Hold Acknowledge). These two signals come out on shared ROM Expand Bus pins RE8 and RE12. To request a DMA access a host would activate HOLD (active high and await the acknowledging HLDA from the TMP before proceeding with the DMA. The TMP only allows DMA operations during the vertical blanking period and will activate HLDA in response to a HOLD shortly after vertical blanking starts. In DMA mode all 16 TMP System Bus drivers are tri-stated while the bus control signals RAM ALE, RAM RD, RAM WR go to their inactive (high) states. A HOLD request must arrive two CPU cycles before vertical blanking starts; otherwise it will miss that retrace cycle and will have to wait until the next one, one frame later. Once DMA mode is entered, it is maintained for the duration of vertical blanking regardless of the state of HOLD. Near the end of vertical blanking the DMA mode will terminate in preparation for the display of the next frame, but the HLDA will NOT turn off. Specifically, this will occur one scan time before the end of vertical blanking. It is up to the designer to be sure that the host is off the BUS before this happens or suffer bus contention with the video controller. He can do this by either predetermining the length of time the host has to remain on the bus, or by using the end of vertical sync (as shown in *Figure 32*) to signal the end of a safe DMA period. If during DMA the CPU attempts to do a display memory access it would be put into a wait state until DMA is concluded and normal memory accessing is resumed.



TL/DD/5526-45

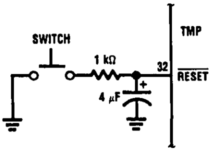
Vertical sync should be programmed to end as late as possible, but must end at least one scan time before the end of vertical blanking.

FIGURE 32

4.0 Reset

The TMP will reset if the $\overline{RESET}$ (32) pin is held at a logic low (< 0.8V) for at least five CPU cycle times. This pre-supposes that the V_{CC} is up, stable and within operational limits (+5V ±10%) and that the oscillator is running. For a power on reset, time must be allowed for the power supplies to stabilize (typically 50 ms) and the oscillator to start up. If power supply noise or ripple causes V_{CC} to exceed the +5V ±10% limits neither reset nor operation is guaranteed.

Internally, the $\overline{RESET}$ pin has a depletion load pullup that typically acts as a 30 µA current source from V_{CC} in the voltage range of interest. A typical reset circuit with a 0.5 second reset pulse is shown in *Figure 33*.



TL/DD/5526-41

FIGURE 33. Typical Reset Circuit

4.0 Reset (Continued)

During RESET a number of internal registers are initialized as follows:

4.1 CPU

CPU Clock divide = 1.5 (SCR bit 0 = 1)
Shared VIDCLK/$\overline{\text{FI CLK}}$ = 0 (SCR bit 7 = 0, $\overline{\text{FI CLK}}$ gated to external pin)
Program Counter = 0
Stack Pointer = 0
Program Memory Bank = 0
RAM Register Bank = 0
Timer Stopped
Instruction Register cleared
F0 and F1 cleared

4.2 INTERRUPTS

Internal and External Interrupts disabled
Internal Interrupt Register set to 000011X0

4.3 UART

Receiver initialized to look for start bit
Status Register set to 11110000
Transmitter initialized to wait for OUT XMTR instruction
Control Register bit 5 =0 (No BREAK)

4.4 VIDEO

Video generation shutdown (VCR bit 5 = 0)
FIFO Cleared Out

Timing Chain Character Counter	= 0	IN TEST MODE ONLY
Timing Chain Scan Counter	= 0	
Timing Chain Row Counter	= 0	
Timing Chain Blink Counter	= 0	

4.5 PIN STATES AT RESET

Pins 1–8 (SB0–7)	In TRI-STATE during reset and until either the CPU executes a MOVX instruction or bit 5 of the VCR is set.
Pins 9–16 (SB8–15)	If bit 4 of the SCR is set, SB8–15 will behave like SB0–7. If bit 4 of the SCR is cleared, SB8–15 will act as outputs (any of which may be either high or low). Note that bit 4 of the SCR may be one or zero at power-up.
Pin 17 (VID CLK/$\overline{\text{FI CLK}}$)	High during reset and until bit 5 of the VCR is set.
Pin 18 (RAM ALE)	High during reset and until the CPU executes a MOVX instruction or bit 5 of the VCR is set.
Pin 19 ($\overline{\text{RAM WR}}$)	High during reset and until the CPU executes a MOVX (of the output to display RAM variety) instruction.
Pin 20 ($\overline{\text{RAM RD}}$)	High during reset and until either the CPU executes a MOVX instruction or bit 5 of the VCR is set.
Pin 21 (ALE)	Pulses continuously.
Pin 22 (XTAL 2)	Crystal input or master clock input.
Pin 23 (XTAL 1)	Crystal input.
Pin 24 (Gnd.)	
Pin 25 ($\overline{\text{INTENS/FO CLK}}$)	May be either high or low during reset.
Pin 26 (VO)	Low (because of asserted blanking signals) from reset until bit 5 of the VCR is set.
Pin 27 ($\overline{\text{VS}}$)	In TRI-STATE mode upon RESET, enabled when bit 5 of the VCR is set.
Pin 28 (HS)	Low from reset until bit 5 of the VCR is set.
Pin 29 (EA)	Input only. (must be tied HIGH (V_{IH_2}))

4.0 Reset (Continued)

Pin 30 ($\overline{\text{PSEN}}$) Active during reset.

Pin 31 ($\overline{\text{RD}}$) High during reset and until an IN PORT instruction is executed.

Pin 32 ($\overline{\text{RESET}}$) Input only.

Pin 33 (SO) High during reset and until an OUT XMTR instruction is executed.

Pin 34 (SI) Input only.

Pin 35 (RE12/HLDA) If HOLD is low: low during reset. If HOLD is high: low at falling edge of ALE and during $\overline{\text{PSEN}}$, may be low or high at rising edge of ALE.

Pin 36 (RE11/$\overline{\text{SC CLR}}$) If reset asserted: low at falling edge of ALE and during $\overline{\text{PSEN}}$, sampled value of internal Scan Count Clear signal is output at rising edge of ALE.

Pin 37 (RE10/$\overline{\text{INTR}}$)
Pin 38 (RE9/$\overline{\text{LPEN}}$)
Pin 39 (RE8/HLDR)
If reset asserted: low at falling edge of ALE and during $\overline{\text{PSEN}}$. Always in TRI-STATE at rising edge of ALE.

Pins 40–47 (RE0–7; I/O0–7) If reset asserted: low at falling edge of ALE, in TRI-STATE during $\overline{\text{PSEN}}$, and may be either high or low at the rising edge of ALE.

Pin 48 (V_{CC})

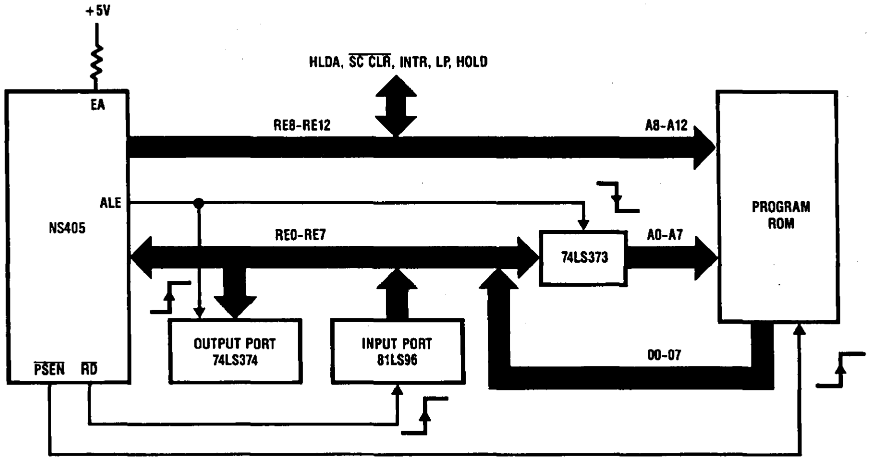
5.0 Extra Attributes

One may want to expand the external attribute field by adding more bits so that functions such as color (Red—Green—Blue drive) or grey scale may be implemented. Like the eight attributes which the chip handles internally these extra attributes would operate on a character cell basis. To add attribute bits one would have to duplicate the internal 4 level character/attribute FIFO externally using fast MSI chips. To assist in handling the external FIFO circuitry the TMP features two FIFO clocking signals on pins 17 and 25. The FIFO IN Clock ($\overline{\text{FI CLK}}$) is used to strobe attribute data into the external FIFO circuits in synchronism with the internal TMP FIFO. Its timing is identical to $\overline{\text{RAM RD}}$ but is only active when the video does a display RAM read to load its FIFO. The FIFO OUT Clock ($\overline{\text{FO CLK}}$) pulses for 1-3 bit times each time the video starts the display of a new character cell. The external FIFO would use the rising edge of this signal to clock out or latch the attribute output.

In order for the TMP CPU to access the additional attribute bits special bus gating arrangements would have to be worked out on the System Bus (Video Data Bus is at most 16 bits wide). Unless one were to run with internal attributes or only use a few of the external attributes in which case the unused bits could be used with the external FIFO. Whenever using the $\overline{\text{FO CLK}}$ the Intensity attribute is disabled since they both share the same pin.

6.0 TMP BUS Interfacing

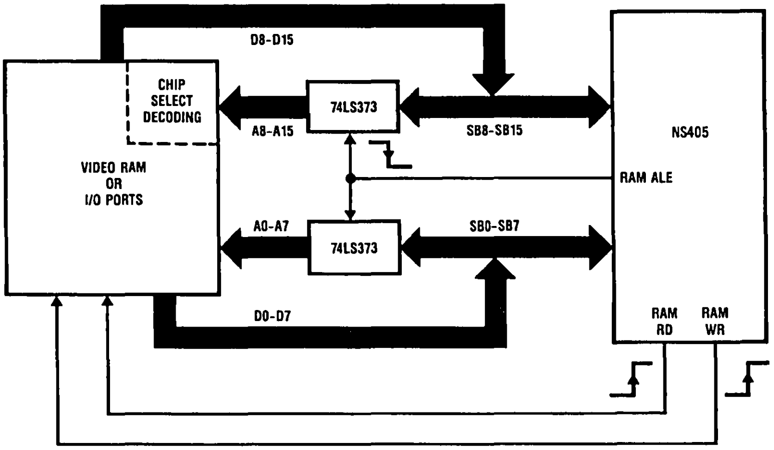
The two external buses on the TMP, ROM Expand and System are easily interfaced to as shown in *Figures 34* and *35*. Important bus information output from the chip is latched using the rising or falling edges of the various control signals. I/O port information is read in through a TRI-STATE® buffer chip such as an 81LS96.



TL/DD/5526-42

FIGURE 34. TMP ROM Expand BUS

6.0 TMP BUS Interfacing (Continued)



TL/DD/5526-43

FIGURE 35. TMP System Bus

TMP Registers (Excluding Timing Chain Registers)

TMP Registers		Associated Intructions
		CPU SECTION
A	= Accumulator — 8 bits	ADD A,Rr ADD A,#data ADD A,@Rr ADDC A, Rr ADDC A,#data ADDC A,@Rr ANL A,Rr ANL A,#data ANL A,@Rr CLR A CPL A DAA DEC A DEC Rr INC A INC Rr INC @Rr MOV A,Rr MOV A,@Rr MOV A,#data MOV Rr,A MOV Rr,#data MOV @Rr,A MOV @Rr,#data MOVP A,@A MOVP3 A,@A RL A RLC A RR A RRC A ORL A,Rr ORL A,@Rr ORL A,#data SWAP A XCH A,Rr XCH A,@Rr XCHD A,@Rr XRL A,Rr XRL A,@Rr XRL A,#data JBn addr JNZ addr JZ addr DJNZ Rr,addr
#data	= data immediate	
Rr	= Register	
@Rr	= Register pointed to by R0 or R1	
*HACC	= High Accumulator — 8 bits	*MOV A,HACC *MOV HACC,A
C	= Carry Bit	CLR C CPL C JNC addr JC addr
*LONG R0	= Register Pair, R0, RA	*DECL R0 *INCL R0 *MOVL A,R0 *MOVL R0,A *MOVX A,@R0 *MOVX @R0,A
*LONG R1	= Register Pair R1, RB	*DECL R1 *INCL R1 *MOVL A,R1 *MOVL R1,A *MOVX A,@R1 *MOVX @R1,A
T	= Timer — 8 bits	MOV A,T MOV T,A STOP T STRT T *JNTF addr JTF addr
F0	= Flag 0	CLR F0 CPL F0 JF0 addr *JNF0 addr
F1	= Flag 1	CLR F1 CPL F1 JF1 addr *JNF1 addr
INTR	= Interrupt Register — 8 bits	MOV A,INTR JNXI addr JXI addr *DIS II DIS XI *EN II EN XI

TMP Registers (Excluding Timing Chain Registers) (Continued)

TMP Registers		Associated Instructions		
		CPU SECTION (Continued)		
MASK	= Internal Interrupt MasK — 8 bits	*MOV MASK,A		
PSW	= Program Status Word — 8 bits	MOV A,PSW	MOV PSW,A	
PORT	= 8 bit I/O Port	ANL PORT,#data	IN PORT	
		ORL PORT,#data	OUT PORT	
Miscellaneous Instructions		CALL addr	JMP addr	JMPP @A
		NOP	RET	RETR
		SEL MB0	SEL MB1	*SEL MB2
		*SEL MB3	SEL RB0	SEL RB1
		VIDEO MANAGEMENT		
			Associated Instructions	
SCR	= System Control Register — 8 bits		*MOV SCR,A	
VCR	= Video Control Register — 8 bits		*MOV VCR,A	
HOME	= Home Address Register — 16 bits		*MOV A,HOME	*MOV HOME,A
CURS	= Cursor Address Register — 16 bits	*DEC CURS	*INC CURS	*MOVX A,@CURS
		*MOV CURS,A	*MOV A,CURS	*MOVX @CURS,A
BEGD	= Beginning of Display RAM Register — 16 bits		*MOV BEGD,A	
ENDD	= End of Display RAM Register — 16 bits		*MOV ENDD,A	
SROW	= Status Row Register — 16 bits		*MOV SROW,A	
AL0	= Attribute Latch 0 — 8 bits		*MOV AL0,A	
AL1	= Attribute Latch 1 — 8 bits		*MOV AL1,A	
HPEN	= Horizontal Light Pen Register — 7 bits		*MOV A,HPEN	
VPEN	= Vertical Light Pen Register — 5 bits		*MOV A,VPEN	
VINT	= Vertical Interrupt Register — 5 bits		*MOV VINT,A	
		UART CONTROL		
PSR	= Prescale Register (UART) — 8 bits		*MOV PSR,A	
BAUD	= Baud Rate Select Register — 8 bits		*MOV BAUD,A	
UCR	= UART Control Register — 8 bits		*MOV UCR,A	
UMX	= UART Multiplex Register — 8 bits		*MOV UMX,A	
STAT	= Status Latch (UART) — 6 bits		*MOV A,STAT	
RCVR	= UART Receive Buffer — 8 bits		*IN RCVR	
XMTR	= UART Transmit Buffer — 8 bits		*OUT XMTR	
TCP	= Timing Chain Pointer		*MOV TCP,A	
@TCP	= Register Pointed to by TCP		*MOV @TCP,A	

*New instruction added to 8048 subset.

Symbol Definitions

Symbol	Definition
AC	Auxiliary Carry Flag
addr	Program Memory Address
b	Bit Designator (b = 0 − 7)
BS	RAM Bank Switch
data	Number or Expression (8 bits)
DBF	Program Memory Bank Select Bits (2)
EXI	External Interrupt Pin
F0, F1	Internal Flags
P	I/O Port (8 bits)

Symbol	Definition
PC	Program Counter
SP	Stack Pointer
TF	Timer Flag
#	Prefix for Immediate Data
@	Prefix for Indirect Address
( )	Contents of Register
(( ))	Contents of Memory Location pointed to by designated register
←	Replaced by

Instruction Set

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
ADD A, Rr	0 1 1 0 1 r r r	(A) ← (A) + (Rr) for r= 0 − 7	Add contents of designated register to the Accumulator (8-bit operation)	1	1	*	*	*		
ADD A, #data	0 0 0 0 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(A) ← (A) + data	Add immediate the specified data to the Accumulator (8-bit operation)	2	2	*	*	*		
ADD A, @ Rr	0 1 1 0 0 0 0 r	(A) ← (A) + ((Rr)) for r = 0 − 1	Add indirect the contents of data memory pointed to by Rr to the Accumulator (8-bit operation)	1	1	*	*	*		
ADDC A, Rr	0 1 1 1 1 r r r	(A) ← (A) + (C) + (Rr) for r = 0 − 7	Add with carry the contents of the designated register to the Accumulator (8-bit operation)	1	1	*	*	*		
ADDC A, # data	0 0 0 1 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(A) ← (A) + (C) + data	Add immediate with carry the specified data to the Accumulator (8-bit operation)	2	2	*	*	*		
ADDC A, @ Rr	0 1 1 1 0 0 0 r	(A) ← (A) + (C) + ((Rr)) for r = 0 − 1	Add indirect with carry the contents of data memory pointed to by Rr to the Accumulator (8-bit operation)	1	1	*	*	*		
ANL A, Rr	0 1 0 1 1 r r r	(A) ← (A) AND (Rr) for r = 0 − 7	Logical AND contents of designated register with Accumulator (8-bit operation)	1	1					
ANL A, # data	0 1 0 1 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(A) ← (A) AND data	Logical AND specified Immediate Data with Accumulator (8-bit operation)	2	2					
ANL A, @ Rr	0 1 0 1 0 0 0 r	(A) ← (A) AND ((Rr)) for r = 0 − 1	Logical AND indirect the contents of data memory pointed to by Rr with Accumulator (8-bit operation)	1	1					
ANL PORT, # data	0 1 1 1 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(P) ← (P) AND data	Logical AND immediate specified data with output port (8-bit operation)	2	2					
CALL addr	a10 a9 a8 1 0 1 0 0 a7 a6 a5 a4 a3 a2 a1 a0	((SP)) ← (PC0-12) ((SP)) ← (PSW3-7) (SP) ← (SP) + 1 (PC8-10) ← addr 8-10 (PC0-7) ← addr 0-7 (PC11-12 ← DBF 0, 1	Call designated subroutine	2	2					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
CLR A	0 0 1 0 0 1 1 1	(A) ← 0	Clear the Accumulator	1	1					
CLR C	1 0 0 1 0 1 1 1	(C) ← 0	Clear carry bit	1	1	*				
CLR F0	1 0 0 0 0 1 0 1	(FO) ← 0	Clear Flag 0	1	1				*	
CLR F1	1 0 1 0 0 1 0 1	(F1) ← 0	Clear Flag 1	1	1					*
CPL A	0 0 1 1 0 1 1 1	(A) ← NOT (A)	Complement the contents of the Accumulator (8-bit operation)	1	1					
CPL C	1 0 1 0 0 1 1 1	(C) ← NOT (C)	Complement carry bit	1	1	*				
CPL F0	1 0 0 1 0 1 0 1	(F0) ← NOT (F0)	Complement Flag 0	1	1				*	
CPL F1	1 0 1 1 0 1 0 1	(F1) ← NOT (F1)	Complement Flag 1	1	1					*
DA A	0 1 0 1 0 1 1 1		Decimal Adjust the contents of the Accumulator (8-bit operation)	1	1	*	*			
DEC A	0 0 0 0 0 1 1 1	(HACC, A) ← (HACC, A) − 1	Decrement by 1 the contents of HACC/ACC	1	1	*		*		
DEC CURS	0 0 0 0 1 0 1 0	(CURS) ← (CURS) − 1	Decrement by 1 the contents of the Cursor Address Register	1	1					
DEC Rr	1 1 0 0 1 r r r	(Rr) ← (Rr) − 1	Decrement by 1 the contents of the designated register (8-bit operation)	1	1	*				
DECL Rr	0 0 0 0 1 0 0 r	(Rr) ← (Rr) − 1 for r = 0 − 1	Decrement by 1 the contents of the designated 16-bit register pair	1	1					
DIS II	0 0 1 1 0 1 0 1		Disable internal interrupts	1	1					
DIS XI	0 0 0 1 0 1 0 1		Disable external interrupts	1	1					
DJNZ Rr, addr	1 1 1 0 1 r r r a7 a6 a5 a4 a3 a2 a1 a0	(Rr) ← (Rr) − 1 for r = 0 − 7 If (Rr) ≠ 0 do (PC0−7) ← addr If (Rr) = 0 do (PC) ← PC + 2	Decrement the specified register and Jump if not zero to designated address within page (8-bit decrement)	2	2					
EN II	0 0 1 0 0 1 0 1		Enable internal interrupts.	1	1					
EN XI	0 0 0 0 0 1 0 1		Enable external interrupt.	1	1					
INC A	0 0 0 1 0 1 1 1	(HACC, A) ← (HACC, A) + 1	Increment by 1 the contents of HACC/A.	1	1	*		*		
INC CURS	0 0 1 1 1 0 1 0	(CURS) ← (CURS) + 1	Increment by 1 the contents of the Cursor Address Register.	1	1					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
INC Rr	0 0 0 1 1 r r r	(Rr) ← (Rr) + 1 for r = 0 − 7	Increment by 1 the contents of the designated register (8-bit increment)	1	1	*				
INC @ Rr	0 0 0 1 0 0 0 r	((Rr)) ← ((Rr)) + 1 for r = 0 − 1	Increment in direct the contents of data memory pointed to by Rr (8-bit increment)	1	1	*				
INCL Rr	0 0 1 1 1 0 0 r	(Rr) ← (Rr) + 1 for r = 0 − 1	Increment by 1 the contents of the designated 16-bit register pair	1	1					
IN PORT	1 1 1 0 0 0 0 1	(A) ← (P)	Input data from port into Accumulator (8-bit transfer)	2	1					
IN RCVR	1 1 1 0 0 0 0 0	(A) ← (RCVR)	Input contents of UART Receive buffer into Accumulator (8-bit transfer). Also, clears Receive Buffer Full interrupt.	1	1					
JBb addr	b2 b1 b0 1 0 0 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if (b) = 1 (PC) ← (PC) + 2 if (b) = 0 for b = 0 − 7	Jump to specified address within page if Accumulator bit is set	2	2					
JC addr	1 1 1 1 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if C = 1 (PC) ← (PC) + 2 if C = 0	Jump to specified address within page if Carry flag is set	2	2					
JF0 addr	1 0 0 1 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if F0 = 1 (PC) ← (PC) + 2 if F0 = 0	Jump to specified address within page if Flag F0 is set	2	2					
JF1 addr	0 1 1 1 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if F1 = 1 (PC) ← (PC) + 2 if F1 = 0	Jump to specified address within page if Flag F1 is set	2	2					
JMP addr	a10 a9 a8 0 0 1 0 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC8–10) ← addr 8–10 (PC0–7) ← addr 0–7 (PC11–12) ← DBF 0, 1	Direct Jump to specified address within 2k Bank	2	2					
JMPP @ A	1 0 1 0 0 0 1 1	(PC0–7) ← ((A))	Jump indirect within page to the address specified in the memory location pointed to by the Accumulator	2	1					
JNC addr	1 1 1 0 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if C = 0 (PC) ← (PC) + 2 if C = 1	Jump within page to specified address if Carry flag is 0	2	2					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
JNF0 addr	1 0 0 0 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if F0 = 0 (PC) ← (PC) + 2 if F0 = 1	Jump within page to specified address if F0 is 0	2	2					
JNF1 addr	0 1 1 0 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if F1 = 0 (PC) ← (PC) + 2 if F1 = 1	Jump within page to specified address if F1 is 0	2	2					
JNTF addr	0 0 0 0 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if TF = 0 (PC) ← (PC) + 2 if TF = 1, (TF) ← 0	Jump within page to specified address if Timer flag is reset. If not, continue and reset TF	2	2					
JNXI addr	1 0 1 0 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if EXI = LOW (PC) ← (PC) + 2 if EXI = HIGH	Jump within page to specified address if External Interrupt pin is LOW	2	2					
JNZ addr	1 1 0 1 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if A ≠ 0 (PC) ← (PC) + 2 if A = 0	Jump within page to specified address if Accumulator is not 0	2	2					
JTF addr	0 0 0 1 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if TF = 1, (TF) ← 0 (PC) ← (PC) + 2 if TF = 0	Jump within page to specified address if Timer flag is set. If jump taken Timer flag reset	2	2					
JXI addr	1 0 1 1 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if EXI = HIGH (PC) ← (PC) + 2 if EXI = LOW	Jump within page to specified address if External Interrupt pin is HIGH	2	2					
JZ addr	1 1 0 0 0 1 1 0 a7 a6 a5 a4 a3 a2 a1 a0	(PC0–7) ← addr if A = 0 (PC) ← (PC) + 2 if A ≠ 0	Jump within page to specified address if Accumulator is 0	2	2					
MOV A, CURS	1 0 0 1 1 0 1 1	(HACC/A) ← (CURS)	Copy the contents of the Cursor Address Register into the HACC/A (16-bit transfer)	1	1			*		
MOV A, HACC	1 1 1 0 0 0 1 0	(A) ← (HACC)	Copy contents of the High Accumulator into the Low Accumulator (8-bit transfer)	1	1					
MOV A, HOME	1 0 0 1 1 0 1 0	(HACC/A) ← (HOME)	Copy the contents of the Home Address register into the HACC/A (16-bit transfer)	1	1			*		
MOV A, HPEN	0 0 1 1 1 1 1 1	(A0–6) ← (HPEN) (A7) ← O	Copy the contents of the Horizontal Light Pen Register into the Accumulator (7-bit transfer, A7 cleared)	1	1					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
MOV A, INTR	1 0 0 0 1 1 0 0	(A) ← (INTR)	Copy the contents of the Interrupt Register into the Accumulator (8-bit transfer)	1	1					
MOV A, PSW	1 1 0 0 0 1 1 1	(A) ← (PSW)	Copy contents of the Program Status word into the Accumulator (8-bit transfer)	1	1					
MOV A, Rr	1 1 1 1 1 r r r	(A) ← (Rr) for r = 0 − 7	Copy the contents of the designated Register into the Accumulator (8-bit transfer)							
MOV A, STAT	1 0 0 1 1 1 0 0	(A0–5) ← (STAT) (A6–7) ← 11	Copy the contents of the UART Status Latch into the Accumulator (6-bit transfer, A6 and A7 set)	1	1					
MOV A, T	0 1 0 0 0 0 1 0	(A) ← (T)	Copy the contents of the Timer into the Accumulator (8-bit transfer)	1	1					
MOV A, VPEN	0 0 1 1 1 1 1 0	(A0–4) ← (VPEN) (A5–7) ← O	Copy contents of the Vertical Light Pen Register into the Accumulator (5-bit transfer, A5–A7 cleared)	1	1					
MOV A, @ Rr	1 1 1 1 0 0 0 r	(A) ← ((Rr)) for r = 0 − 1	Copy indirect the contents of data memory pointed to by Rr into the Accumulator (8-bit transfer)	1	1					
MOV A, # data	0 0 1 0 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(A) ← data	Load immediate the specified data into the Accumulator (8-bit load)	2	2					
MOV AL0, A	0 0 1 1 1 1 0 0	(AL0) ← (A)	Copy the contents of the Accumulator into Attribute Latch 0 (8-bit transfer)	1	1					
MOV AL1, A	0 0 1 1 1 1 0 1	(AL1) ← (A)	Copy the contents of the Accumulator into Attribute Latch 1 (8-bit transfer)	1	1					
MOV BAUD, A	0 0 0 0 0 0 1 0	(BAUD) ← (A)	Copy the contents of the Accumulator into the UART Baud Rate Select Register (8-bit transfer)	1	1					
MOV BEGD, A	0 0 0 0 1 1 0 1	(BEGD) ← (HACC/A)	Copy the contents of HACC/A into the Beginning of Display RAM Register (16-bit transfer)	1	1					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
MOV CURS, A	1 0 0 0 1 0 1 1	(CURS) ← (HACC/A)	Copy the contents of HACC/A into the Cursor Address Register (16-bit transfer)	1	1					
MOV ENDD, A	0 0 0 0 1 1 0 0	(ENDD) ← (HACC/A)	Copy the contents of HACC/A into the End of Display RAM Register (16-bit transfer)	1	1					
MOV HACC, A	1 1 0 0 0 0 1 0	(HACC) ← (A)	Copy the contents of the Low Accumulator into the High Accumulator (8-bit transfer)	1	1			*		
MOV HOME, A	1 0 0 0 1 0 1 0	(HOME) ← (HACC/A)	Copy the contents of HACC/A into the Home Address Register (16-bit transfer)	1	1					
MOV MASK, A	1 0 0 0 0 0 1 0	(MASK) ← (A)	Copy the contents of the Accumulator into the Interrupt Mask Register (8-bit transfer)	1	1					
MOV PSR, A	0 0 1 0 0 0 1 0	(PSR) ← (A)	Copy the contents of the Accumulator into the UART Prescale Register (8-bit transfer)	1	1					
MOV PSW, A	1 1 0 1 0 1 1 1	(PSW) ← (A)	Copy contents of the Accumulator into the Program Status Word (8-bit transfer)	1	1	*	*			
MOV Rr, A	1 0 1 0 1 r r r	(Rr) ← (A) for r = 0 − 7	Copy contents of the Accumulator into the designated register (8-bit transfer)	1	1					
MOV SCR, A	0 1 0 1 0 1 0 1	(SCR) ← (A)	Copy contents of the Accumulator into the System Control Register (8-bit transfer)	1	1					
MOV SROW, A	0 0 0 0 1 1 1 0	(SROW) ← (HACC/A)	Copy the contents of HACC/A into the Status Row Register (16-bit transfer)	1	1					
MOV T, A	0 1 1 0 0 0 1 0	(T) ← (A)	Copy the contents of the Accumulator into the Timer (8-bit transfer)	1	1					
MOV TCP, A	1 0 0 0 0 1 1 1	(TCP) ← (A)	Copy the contents of the Accumulator into the Timing Chain Pointer	1	1					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
MOV UCR, A	0 0 0 0 0 0 0 1	(UCR) ← (A)	Copy the contents of the Accumulator into the UART Control Register (8-bit transfer)	1	1					
MOV VCR, A	0 1 0 0 0 1 0 1	(VCR) ← (A)	Copy the contents of the Accumulator into the Video Control Register (8-bit transfer)	1	1					
MOV VINT, A	1 0 1 0 0 0 1 0	(VINT) ← (A)	Copy the contents of the Accumulator into the Vertical Interrupt Register	1	1					
MOV Rr, # data	1 0 1 1 1 r r r d7 d6 d5 d4 d3 d2 d1 d0	(Rr) ← data for r = 0 − 7	Load immediate the specified data into the designated register (8-bit load)	2	2					
MOV @ Rr, A	1 0 1 0 0 0 0 r	((Rr)) ← (A) for r = 0 − 1	Copy indirect the contents of the Accumulator into the data memory location pointed to by Rr (8-bit transfer)	1	1					
MOV @ Rr, # data	1 0 1 1 0 0 0 r d7 d6 d5 d4 d3 d2 d1 d0	((Rr)) ← data for r = 0 − 1	Load indirect the specified immediate data into the data memory location pointed to by Rr (8-bit load)	2	2					
MOV @ TCP, A	1 0 1 1 0 1 1 1	((TCP)) ← (A) (TCP) ← (TCP) + 1	Copy indirect the contents of the Accumulator into the Timing Chain Register pointed to by TCP. Contents of TCP incremented by 1	1	1					
MOV UMX, A	0 0 1 1 0 0 1 1	(UMX) ← (A)	Copy the contents of the Accumulator into the UART Multiplex Register (8-bit transfer)	1	1					
MOVL A, R0	1 0 0 1 1 0 0 0	(HACC/A) ← (RA, R0)	Copy the contents of RA, R0 into HACC/A (16-bit transfer)	1	1			*		
MOVL A, R1	1 0 0 1 1 0 0 1	(HACC/A) ← (RB, R1)	Copy the contents of RB, R1 into HACC/A (16-bit transfer)	1	1			*		
MOVL R0, A	1 0 0 0 1 0 0 0	(RA, R0) ← (HACC/A)	Copy the contents of HACC/A into RA, R0 (16-bit transfer)	1	1					
MOVL R1, A	1 0 0 0 1 0 0 1	(RB, R1) ← (HACC/A)	Copy the contents of HACC/A into RB, R1 (16-bit transfer)	1	1					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
MOVP A, @ A	1 0 1 1 0 0 1 1	(PC0–7) ← (A) (A) ← ((PC)) (PC0–7) ← (old PC0–7) + 1	Replace low 8 bits of PC with A. Load indirect within page the contents of the memory location pointed to by new PC into Accumulator. Restore PC with old value plus 1. Operates in all memory banks.	2	1					
MOVP3 A, @ A	1 1 1 1 0 0 1 1	(PC0–7) ← (A) (PC8–10) ← 011 (A) ← ((PC)) (PC) ← (old PC) + 1	Replace low 8 bits of PC with A. Next 3 bits replaced with 011. Load indirect within page 3 the contents of the memory location pointed to by new PC into the Accumulator. Restore PC with old value plus 1. Operates in all memory banks.	2	1					
MOVX A, @ CURS	1 0 0 1 1 1 0 1	(HACC/A) ← ((CURS))	Copy indirect the contents of display memory as pointed to by CURS into HACC/A (16-bit transfer)	Min. 2	1			*		
MOVX A, @ R0	1 0 0 1 0 0 0 0	(HACC/A) ← ((RA, R0))	Copy indirect the contents of display memory as pointed to by RA, R0 into HACC/A (16-bit transfer)	Min. 2	1			*		
MOVX A, @ R1	1 0 0 1 0 0 0 1	(HACC/A) ← ((RB, R1))	Copy indirect the contents of display memory as pointed to by RB, R1 into HACC/A (16-bit transfer)	Min. 2	1			*		
MOVX @ CURS, A	1 0 0 0 1 1 0 1	((CURS)) ← (HACC/A)	Copy indirect the contents of HACC/A into the display memory location as pointed to by CURS (16-bit transfer)	Min. 2	1					
MOVX @ R0, A	1 0 0 0 0 0 0 0	((RA, R0)) ← (HACC/A)	Copy indirect the contents of HACC/A into the display memory location as pointed to by RA, R0 (16-bit transfer)	Min. 2	1					

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
MOVX @ R1, A	1 0 0 0 0 0 0 1	((RB, R1)) ← (HACC/A)	Copy indirect the contents of HACC/A into the display memory location pointed to by RB, R1 (16-bit transfer)	Min. 2	1					
NOP	0 0 0 0 0 0 0 0		No Operation	1	1					
ORL A, Rr	0 1 0 0 1 r r r	(A) ← (A) OR (Rr) for r = 0 − 7	Logical OR contents of designated register with Accumulator (8-bit transfer)	1	1					
ORL A, @ Rr	0 1 0 0 0 0 0 r	(A) ← (A) OR ((Rr)) for r = 0 − 1	Logical OR indirect the contents of the data memory location pointed to by Rr with Accumulator (8-bit operation)	1	1					
ORL A, # data	0 1 0 0 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(A) ← (A) OR data	Logical OR the specified immediate data with the Accumulator (8-bit operation)	2	2					
ORL PORT, # data	0 1 1 0 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(P) ← (P) OR data	Logical OR immediate specified data with output port	2	2					
OUT PORT	1 1 0 0 0 0 0 1	(P) ← (A)	Output the contents of the Accumulator to the I/O Port (8-bit transfer)	2	1					
OUT XMTR	1 1 0 0 0 0 0 0	(XMTR) ← (A)	Copy the contents of the Accumulator into the UART Transmit Buffer (8-bit transfer). Also clears Transmit Buffer empty interrupt	1	1					
RET	1 0 0 0 0 0 1 1	(SP) ← (SP) − 1 (PC0−12) ← ((SP))	Return from subroutine without restoring Program Status Word bits 5−7	2	1					
RETR	1 0 0 1 0 0 1 1	(SP) ← (SP) − 1 (PC0−12) ← ((SP)) (PSW 3−7) ← ((SP))	Return from Subroutine restoring Program Status Word (use for all returns from interrupts)	2	1	*	*			
RL A	1 1 1 0 0 1 1 1	(A_{n+1}) ← (An) for n = 0 − 6 (A0) ← (A7)	Rotate Accumulator left by 1 bit without carry	1	1					
RLC A	1 1 1 1 0 1 1 1	(A_{n+1}) ← (An) for n = 0 − 6 (A0) ← (C) (C) ← (A7)	Rotate Accumulator left by 1 bit through carry	1	1	*				

Instruction Set (Continued)

Mnemonic	Machine Code	Function	Description	Cycles	Bytes	Flags				
						C	AC	HACC	F0	F1
RR A	0 1 1 1 0 1 1 1	(An) ← A_{n+1} for n = 0 – 6	Rotate Accumulator right by 1 bit without carry	1	1					
RRC A	0 1 1 0 0 1 1 1	(An) ← A_{n+1} for n = 0 –6 (A7) ← (C) (C) ← (A0)	Rotate Accumulator right by 1 bit through carry	1	1	*				
SEL MB0	1 1 0 0 0 1 0 1	(DBF) ← 00	Select Bank 0 (0–2047) of Program Memory	1	1					
SEL MB1	1 1 0 1 0 1 0 1	(DBF) ← 01	Select Bank 1 (2048–4095) of Program Memory	1	1					
SEL MB2	1 1 1 0 0 1 0 1	(DBF) ← 10	Select Bank 2 (4096–6143) of Program Memory	1	1					
SEL MB3	1 1 1 1 0 1 0 1	(DBF) ← 11	Select Bank 3 (6144–8191) of Program Memory	1	1					
SEL RBn	1 1 n 0 0 0 1 1	(BS) ← n for n = 0 – 1	Select Data RAM Bank (0–7) or 1 (24–31)	1	1					
STOP T	0 1 1 0 0 1 0 1		Stop Timer	1	1					
STRT T	0 1 1 1 0 1 0 1		Start Timer	1	1					
SWAP A	0 1 0 0 0 1 1 1	(A4–A7) ⟷ (A0–A3)	SWAP 4 bit nibbles in Accumulator	1	1					
XCH A, Rr	0 0 1 0 1 r r r	(A) ⟷ (Rr) for r = 0 – 7	Exchange the Accumulator and contents of designated register (8-bit transfer)	1	1					
XCH A, @ Rr	0 0 1 0 0 0 0 r	(A) ⟷ ((Rr)) for r = 0 – 1	Exchange indirect the contents of the Accumulator and the data memory location pointed to by Rr (8-bit transfer)	1	1					
XCHD A, @ Rr	0 0 1 1 0 0 0 r	(A0–3) ⟷ ((Rr)) 0–3 for r = 0 – 1	Exchange indirect the low 4 bits of the Accumulator and the data memory location pointed to by Rr (4-bit transfer)	1	1					
XRL A, Rr	1 1 0 1 1 r r r	(A) ← (A) XOR (Rr) for r = 0 – 7	Logical XOR contents of designated register with Accumulator (8-bit transfer)	1	1					
XRL A, @ Rr	1 1 0 1 0 0 0 r	(A) ← (A) XOR ((Rr)) for r = 0 – 1	Logical XOR indirect the contents of the data memory location pointed to by Rr with the Accumulator	1	1					
XRL A, # data	1 1 0 1 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0	(A) ← (A) XOR data	Logical XOR the immediate specified data with the Accumulator	2	2					

TMP Opcode Chart

LSN

MSN	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	MOV UCR, A	MOV BAUD, A	ADD A, #data	JMP (page 0)	EN XI	JNTF	DEC A	DECL R0	DECL R1	DEC CURS		MOV ENDD, A	MOV BECD, A	MOV SROW, A	
1	INC @R0	INC @R1	JB0	ADDC A, #data	CALL (page 0)	DIS XI	JTF	INC A	INC R0	INC R1	INC R2	INC R3	INC R4	INC R5	INC R6	INC R7
2	XCH A, @R0	XCH A, @R1	MOV PSR, A	MOV A, #data	JMP (page 1)	EN II		CLR A	XCH A, R0	XCH A, R1	XCH A, R2	XCH A, R3	XCH A, R4	XCH A, R5	XCH A, R6	XCH A, R7
3	XCHD A, @R0	XCHD A, @R1	JB1	MOV UMX, A	CALL (page 1)	DIS II		CPL A	INCL R0	INCL R1	INC CURS		MOV AL0, A	MOV AL1, A	MOV A, VPEN	MOV A, HFEN
4	ORL A, @R0	ORL A, @R1	MOV A,T	ORL A, #data	JMP (page 2)	MOV VCR, A		SWAP A	ORL A, R0	ORL A, R1	ORL A, R2	ORL A, R3	ORL A, R4	ORL A, R5	ORL A, R6	ORL A, R7
5	ANL A, @R0	ANL A, @R1	JB2	ANL A, #data	CALL (page 2)	MOV SCR, A		DA A	ANL A, R0	ANL A, R1	ANL A, R2	ANL A, R3	ANL A, R4	ANL A, R5	ANL A, R6	ANL A, R7
6	ADD A, @R0	ADD A, @R1	MOV T,A	ORL PORT, #data	JMP (page 3)	STOP T	JNF1	RRC A	ADD A, R0	ADD A, R1	ADD A, R2	ADD A, R3	ADD A, R4	ADD A, R5	ADD A, R6	ADD A, R7
7	ADDC A, @R0	ADDC A, @R1	JB3	ANL PORT, #data	CALL (page 3)	STRT T	JF1	RR A	ADDC A, R0	ADDC A, R1	ADDC A, R2	ADDC A, R3	ADDC A, R4	ADDC A, R5	ADDC A, R6	ADDC A, R7
8	MOVX @R0, A	MOVX @R1, A	MOV MASK, A	RET	JMP (page 4)	CLR F0	JNF0	MOV TCP, A	MOVL R0, A	MOVL R1, A	MOV HOME, A	MOV CURS, A	MOV A, INTR	MOVX @CURS, A		
9	MOVX A, @R0	MOVX A, @R1	JB4	RETR	CALL (page 4)	CPL F0	JF0	CLR C	MOVL A, R2	MOVL A, R1	MOV A, HOME	MOV A, CURS	MOV A, STAT	MOVX A, @ CURS		
A	MOV @R0, A	MOV @R1, A	MOV VINT, A	JMPP @A	JMP (page 5)	CLR F1	JNXI	CPL C	MOV R0, A	MOV R1, A	MOV R2, A	MOV R3, A	MOV R4, A	MOV R5, A	MOV R6, A	MOV R7, A
B	MOV @R0, #data	MOV @R1, #data	JB5	MOVP A, @A	CALL (page 5)	CPL F1	JXI	MOV @TCP, A	MOV R0, #data	MOV R1, #data	MOV R2, #data	MOV R3, #data	MOV R4, #data	MOV R5, #data	MOV R6, #data	MOV R7, #data
C	OUT XMTR	OUT PORT	MOV HACC, A	SEL RB0	JMP (page 6)	SEL MB0	JZ	MOV A, PSW	DEC R0	DEC R1	DEC R2	DEC R3	DEC R4	DEC R5	DEC R6	DEC R7
D	XRL A, @R0	XRL A, @R1	JB6	XRL A, #data	CALL (page 6)	SEL MB1	JNZ	MOV PSW, A	XRL A, R0	XRL A, R1	XRL A, R2	XRL A, R3	XRL A, R4	XRL A, R5	XRL A, R6	XRL A, R7
E	IN RCVR	IN PORT	MOV A, HACC	SEL RB1	JMP (page 7)	SEL MB2	JNC	RL A	DJNZ R0	DJNZ R1	DJNZ R2	DJNZ R3	DJNZ R4	DJNZ R5	DJNZ R6	DJNZ R7
F	MOV A, @R0	MOV A, @R1	JB7	MOVP3 A, @A	CALL (page 7)	SEL MB3	JC	RLC A	MOV A, R0	MOV A, R1	MOV A, R2	MOV A, R3	MOV A, R4	MOV A, R5	MOV A, R6	MOV A, R7

Ordering Information

ORDER PART NUMBERS

ROMless	NS405-A12N NS405-B12N NS405-C12N	NS405-B18N

Throughput Considerations In NS405 System Planning

National Semiconductor
Application Brief 14
James Murashige

The intricate timing relationships inherent in video generation require that a designer have a firm grasp of the fundamentals of NS405 operation in order to achieve his design objectives. Towards this end the key facets of NS405 operation will be examined and examples given.

The NS405 is a complete video controller that reads in video data, processes it and outputs it to a CRT. Given this, one may derive all essential operating parameters from the following two statements:

1. You must be able to read in video data faster than you output it.
2. Video data accesses are based on the CPU cycle which in turn is based on the crystal or dot clock.

Application of these two statements immediately leads to a limitation on the character cell width as follows:

if f = crystal frequency or dot clock

then $(f \div 1) \div 15$ or $(f \div 1.5) \div 15$ = CPU Instruction Execution Clock Frequency

Since there are three video data accesses each CPU Instruction Execution cycle, there are $3 * (f \div 1) \div 15$ or $3 * (f \div 1.5) \div 15$ video data accesses per second.

if w = dot width of character cell then $f \div w$ = number of character cells being displayed per second.

Statement 1 says that video data accesses/sec ≥ display characters/sec

for CPU Clock ÷ 1	for CPU Clock ÷ 1.5
$3 * (f \div 1) \div 15 \geq f \div w$	$3 * (f \div 1.5) \div 15 \geq f \div w$
$f \div 5 \geq f \div w$	$(3 * f) \div 22.5 \geq f \div w$
$1/5 \geq 1/w$	$3/22.5 \geq 1/w$
$w \geq 5$	$w \geq 7.5$

So depending on the CPU clock divide factor (÷1 or ÷1.5) the character cell width must be a minimum as shown.

Cell width also impacts CPU throughput since both the CPU and Video controller vie for video memory access through the DMA controller. The rules of access are simple and straightforward. The Video Controller gets as many of the accesses as it needs with the CPU getting any left over. The maximum access rate as already shown is $f \div 5$ or $f \div 7.5$ depending on the CPU clock divide. If the CPU attempts a video memory access when things are very busy it will be put into a wait state and remain frozen until things clear up. Of course, no display characters are necessary when the display is blanked, so during the horizontal and vertical retrace periods the CPU has unlimited access to video memory.

Normally, the CPU doesn't have to wait until horizontal retrace to get into video memory, but exactly how often it can get in during a display line requires analysis of the worst case video requirements.

Since the results can vary dramatically depending on the parameters chosen, two typical cases will be presented.

I. With a dot clock of 18 MHz the display line consists of 80 character cells, 9 dots across. Since the CPU clock divide must be 1.5 the video memory access rate is 18 MHz ÷ 7.5 = 2.4 MHz.

To display one line requires (9×80)/18 MHz = 40 us.

In one line time there are 2.4 MHz×40 us = 96 video memory accesses. Of the 96, 80 are required for the characters displayed in the line leaving 16 available for the CPU. This is an average of one every six video memory accesses or once every two CPU instruction cycles. This would be fine since all CPU video memory instructions require two instruction cycles to execute anyway. However, in addition to the DMA controller the video circuits also employ a four level FIFO to insure a smooth data flow. The FIFO is normally kept full at four in which case it stops accessing video data and allows the CPU to have all the accesses. However, the FIFO can drop down quite far before starting to fill up again by taking all of the video memory accesses. The net effect is that instead of being evenly distributed, the accesses available to the CPU are clumped together with long gaps between clumps. Taking the worst case condition of the FIFO being completely empty and having to fill to four by taking the accesses which the CPU could have gotten, the longest gap is (4×6) + 5 = 29 accesses ≈ 10 CPU instruction cycles. Generally speaking this tends to happen towards the middle of a line since the FIFO is filled prior to the start of a line and tries to end a line empty. In fact, accesses for video are performed up to the second to the last display character. The FIFO prefetch for the next line is performed shortly after horizontal blanking starts.

II. If the dot clock is now 12 MHz with a display line of 80 character cells 7 dots across the CPU clock divide can be 1.

The video memory access rate is 12 MHz ÷ 5 = 2.4 MHz.

To do one line requires (7×80)/12 MHz = 46.7 us. In one line time there are 2.4 MHz×46.7 us = 112 video memory accesses. Of the 112, 32 are now available to the CPU. This averages out to one every 3.5. Figuring the FIFO in, the worst case wait for the CPU becomes (4×3.5) + 2.5 = 16.5 accesses ≈ 6 CPU instruction cycles. A significant improvement over the first example.

In general, to maximize CPU access to video memory one must maximize the average number of "free" accesses during the display time. The number of free accesses as a fraction of the total number available is:

$(w - 5d)/w$ Where w = character cell dot width
d = CPU divide factor of 1 or 1.5

As can be seen, throughput performance depends entirely on the cell width and CPU clock divide. To maximize performance one would try to choose a large w and a d of 1.

Applying the delay imposed by the four level FIFO, the maximum CPU delay in accessing video memory becomes =

$(4w + 5d)/(w - 5d)$ Memory cycles

AB-16

NS405-Series TMP External Interrupt Processing

National Semiconductor
Application Brief 16
James Murashige

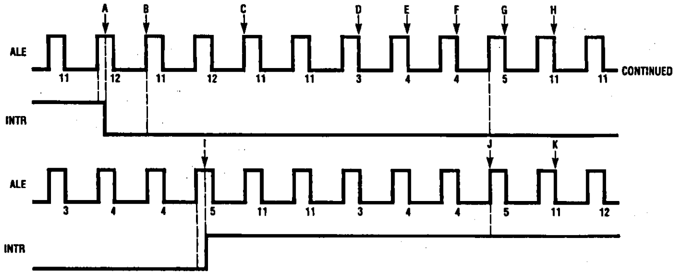
The TMP External Interrupt (INTR) is a level sampled interrupt input. Specifically this means that the input is sampled once each CPU cycle with interrupts being generated as long as the sampled input is a logic low. INTR shares pin 37 with RE10 and is sampled on each ALE rising edge as shown in the data sheet. If a logic low level is detected, interrupt service will commence if interrupts had been previously enabled with an EN XI instruction. Service consists of finishing up the currently executing instruction, pushing the PC and other pertinent information onto the stack, disabling all interrupts while in service and finally performing a JUMP to location 003. Upon completion of service a RETR would be executed to pop the stack and return to where we left off in the main program.

The exact timing involved may be observed through the example program of *Figure 1* and its instruction execution sequence in *Figure 2.* In *Figure 2* the numbers shown on the falling ALE edges are the program addresses put out by the TMP. As written the program will loop endlessly unless diverted by an external interrupt such as point A in *Figure 2.* Since it just missed the previous rising ALE edge it will not be until point B that the logic low INTR is read in. However, by then the CPU will have started execution of the first byte of the JMP 11 instruction. Since instructions are always finished once started, it will not be until point C that we begin interrupt service. At this point the next address would have been back at 11 but we now want to service the interrupt and push the stack. Stack pushing or popping takes 2 CPU cycles so the two address 11's shown following point C are dummies. Finally, we start interrupt service at point D by outputting address 003 and reading in the IN PORT instruction. Since the IN PORT instruction is only 1 byte long but takes 2 CPU cycles to execute, the address "4" at point E is a dummy and isn't really needed until point F when we read in the RETR instruction. Like IN PORT, RETR is a 1 byte instruction that takes 2 CPU cycles to execute. Therefore, the address "5" at point G is redundant. Upon returning from subroutine we immediately push the stack again (point H) since the interrupt is still there. Note that we immediately push the stack and do not execute the JMP at 11. Once more we go through the interrupt service routine but this time the interrupt ends at point I. Since it missed the preceding rising ALE edge where it was still seen as a logic low, we will immediately execute another interrupt service routine as shown. Finally, at point J as we prepare to return from service, INTR will be seen as a logic high and from point K onward execution will proceed normally.

When enabling and disabling interrupts, the rules for when you will and will not service them are predicated on the latest sampled interrupt level and last instruction executed. This is illustrated by the example program of *Figure 3* and instruction execution sequences of *Figure 4.* As shown in *Figure 4a,* the interrupt goes low at point A and will be sampled at the rising ALE of point B. However, since the current executing instruction (DIS XI at location 13) must be completed before starting interrupt service, the interrupt will be

```
ADDRESS          OPCODE          MNEMONIC
  000              04            JMP 010          ;RESET VECTOR
  001              10
  002
  003              E1            IN PORT          ;EXTERNAL INTERRUPT VECTOR
  004              93            RETR
  005
  006
  007
  008
  009
  00A
  00B
  00C
  00D
  00E
  00F
  010              05            EN XI            ;MAIN PROGRAM
  011              04            JMP 001
  012              11
  013
  014
  015
```

FIGURE 1. INTR Service Timing Example Program



TL/DD/6972–1

FIGURE 2. INTR Service Timing

locked out. Execution continues unperturbed until the interrupt is re-enabled with an EN XI from location 11, point F. Although the interrupt went logic high at point E it was still sampled as a logic low at point D.

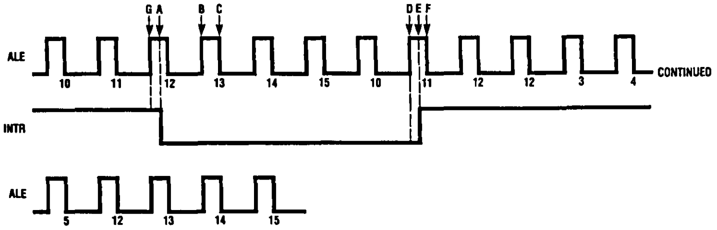
Therefore, after executing the EN XI at location 11, interrupt service will commence as shown. If the interrupt had gone logic high before point D it would have been sampled high and no interrupt service would have been performed.

Returning to the missed interrupt at point A, if the interrupt low had come in time to be sampled at point G, the instruction at 12 would have been the last one executed before interrupt service started as demonstrated in *Figure 4b.*

Although describing the external interrupt, all of the service sequences presented may be directly applied to TMP internal interrupts.

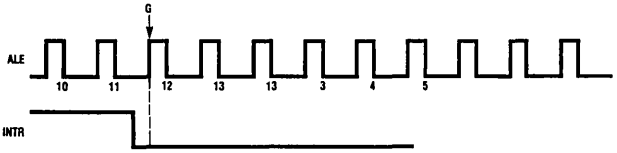
```
ADDRESS        OPCODE         MNEMONIC
000            04             JMP 010        ;RESET VECTOR
001            10
002
003            00             NOP            ;EXTERNAL INTERRUPT VECTOR
004            93             RETR
005
006
007
008
009
00A
00B
00C
00D
00E
00F
010            00             NOP            ;MAIN PROGRAM
011            05             EN XI
012            00             NOP
013            15             DIS XI
014            04             JMP 010
015            10
016
```

FIGURE 3. INTR Enable/Disable Timing Example Program



TL/DD/6972-2

FIGURE 4a. INTR Enable/Disable Timing



TL/DD/6972-3

FIGURE 4b

AN-354

TMP Row and Attribute Table Lookup Operation

National Semiconductor
Application Note 354
James Murashige

This note describes in detail the operation of the TMP Attribute Demo Program - TAD. Although a short program, it nicely demonstrates row table lookup operation in the TMP while at the same time putting out a visual display of the various video attributes available in the chip. While this display management approach is much more involved than normal sequential lookup mode, it is necessary when attemping to do fast screen updates or line editing with the TMP.

The hardware environment for which the program was written is the TMP Demo board. Appropriate references to and descriptions of the hardware will be made as necessary. For those who have not seen it, the net function of the program is to put up and manage a single frame of video data. In the top half of the display the same message is repeated 5 times but each time with a different set of attributes. In the lower half of the display are 4 rows representing the 128 possible block graphics patterns. All of the attribute effects displayed are achieved by updating the internal AL0 attribute latch at the end of each display row. At the same time a message table lookup is performed in order to obtain the appropriate character string that will work with the new attribute set selected.

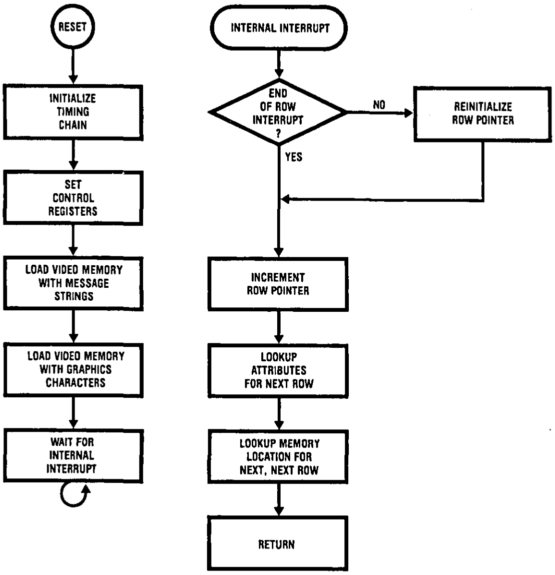
The flowchart for the program is shown in *Figure 1.* As you can see, the program essentially consists of initialization and waiting for and servicing video interrupts to manage the screen display. Initialization starts at BEGIN with the Vertical Interrupt Register and Timing Chain being loaded first. The Vertical interrupt is used for end of frame synchronization and is set to activate after the 27th row. The Timing Chain is loaded as follows:

TCP 0	Horizontal Length	=	104
1	Characters/Row	=	80
2	Horizontal Sync Begin	=	84
3	Horizontal Sync End	=	100
4	Character Height	=	10
	Extra Scans/Frame	=	2
5	Vertical Length	=	27
6	Vertical Blank	=	25
7	Vertical Sync Begin/End	=	7,3
8	Status Row Begin	=	31
9	Blink Rate/D.C.	=	F4H
10	Graphics Column Register	=	30H
11	Graphics Row Register	=	36H
12	Underline Size Register	=	89H
13	Cursor Size Register	=	09H

Given these values, one can ascertain that the display is 80 columns across and 25 rows tall. The character cell height is 10 scan lines and no status line will be displayed. The character underline is the bottom most scan line in a cell and the cursor occupies an entire cell. The partitioning of the block graphics cells is as follows:

```
0011100
0011100
0011100
2233344
2233344
5566655
5566655
5566655
5566655
```

TAD Flowchart


RESET
INITIALIZE TIMING CHAIN
SET CONTROL REGISTERS
LOAD VIDEO MEMORY WITH MESSAGE STRINGS
LOAD VIDEO MEMORY WITH GRAPHICS CHARACTERS
WAIT FOR INTERNAL INTERRUPT
INTERNAL INTERRUPT
END OF ROW INTERRUPT ?
NO
REINITIALIZE ROW POINTER
YES
INCREMENT ROW POINTER
LOOKUP ATTRIBUTES FOR NEXT ROW
LOOKUP MEMORY LOCATION FOR NEXT, NEXT ROW
RETURN

TL/C/5729-1

FIGURE 1

Following timing chain initialization various system registers are set to configure the chip to operate in its hardware environment. The video memory is a 2kX8 NMC2116 located between addresses 000-7FF. The crystal dot clock is 12 Mhz allowing us to use divide by 1 to generate the CPU clock. Accordingly the SCR is set to 24H (SB8-15 address output only, cell width = 7, divide by 1 for CPU clock, row table lookup operation). RAM Bank 0 is selected and HOME, BEGD, RA/RO are cleared. ENDD and CURS are set to 7FFFH and AL1 is set to FFH (no attributes selected). Video display memory (80×25 char) is then cleared out by storing spaces at all of the memory locations. Along with the spaces, attribute latch 1 is specified to be used. Video is then turned on by setting the VCR to 21H (normal alphanumeric display, internal attribute latch operation, normal video).

Next, the message tables are built up in the video memory. By updating the attribute latch AL0 each row, the entire screen display can be constructed from the 7 message rows stored in memory. Each of the message rows consist of 80 consecutive characters and are called up for display by loading the HOME register with the address of the first character in the row. The background characters in each of the rows are the spaces previously stored. Each of the display characters stored use attribute latch AL0 which is updated each row. The first row (0-79) consists entirely of spaces to provide us with a blank display row. The second row (80-159) has the message "tmp does it BETTER!" for normal and double high display. The third row (160-239) contains "ttmmpp ddooeess iitt BBEETTTTEERR!!" for double wide and double size display. Rows 4-7 contain 32 block graphics characters per row for a total of 128 patterns. The 128 characters stored are merely all binary combinations of the low 7 data bits in ascending order. The 32 characters in each row are stored in every other memory location to achieve a blank space between characters. For all of the message rows, data is positioned to give a centered display on the screen.

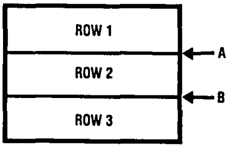
With initialization accomplished, we set the interrupt mask, re-enable interrupts and wait for a video interrupt.

Video display management is performed by the internal interrupt service routine located at 007 and consists of updating the HOME register and AL0 at the end of each display row. To accomplish this, a row counter (R3) is used as a pointer into the data lookup tables which follow the interrupt service routine. The R3 row counter is incremented on each End of Row interrupt or preset and incremented on a re-synching Vertical Interrupt.

Because the next row pointers are pipelined in the video memory controller, an understanding of End of Row and Vertical Interrupt operation is necessary in order to correctly set up the interrupt service routine and lookup tables. In table lookup mode, the Current Row Start Register (CRSR), which is a pointer to the first character address in a row, is automatically reloaded from the HOME register after the display of the last scan line in a row, a few characters into horizontal blanking. The timing of the CRSR reload when operating in sequential lookup mode is the same but in this case the pointer is advanced by the character width of the display row. It is the reloading of CRSR either in sequential or table lookup modes that generates the End of Row interrupt. The duration of the signal is ⅓ CPU cycle making it a one time event each row. The End of Row interrupt register bit is cleared when a reload of HOME, i.e., MOV HOME, A is executed. A simple example will illustrate the pipelining involved. In *Figure 2*, at the end of Row 1 (Point A) an EOR interrupt is generated. In preparation for this event HOME should have been loaded with the starting address of ROW 2 since the interrupt is generated when CRSR reloads from HOME. In service of the EOR, the program would load HOME with the starting address of ROW 3 in preparation for the EOR interrupt at Point B. However, notice that we have an entire row time from A to B to do the HOME reload. Finally note that EOR's are generated at the end of all rows except those blanked during vertical blanking. Vertical Interrupt operates with the same timing as End of Row except that it is specified to occur at the end of a particular row designated by the Vertical Interrupt Register. The row that it is specified to occur on must be <= Vertical Length Register (timing chain rows are counted starting from 0). Otherwise, it will never occur since the row counter will never count up that far. Usually Vertical Interrupt is specified to occur on a row blanked during vertical blanking so that it may be used as a frame sync signal.

Returning to TAD, *Figure 3* shows the interrupt positioning for all of the rows on the screen including the blanked ones. There are 25 displayed rows and 2 blanked ones in a frame for a total of 27. In addition, there are 2 extra scan lines which may be ignored as far as interrupt operation is concerned. Vertical Interrupt is set to occur at the end of the last row in the frame as shown. Row pointer operation for rows 2 to 24 is pipelined as described in *Figure 2*. At the end of ROW 24 (point E) the CRSR will be loading the pointer to ROW 25 and the interrupt service will load HOME with the pointer to ROW 1. At the end of ROW 25 (point F) the CRSR will load the pointer to ROW 1 and save it for the next frame. Since no EOR's are generated during vertical blanking, CRSR will remain static until ROW 1. At this point, it doesn't matter what the interrupt service loads into HOME and AL0 since the Vertical Interrupt at ROW 27 will reset the row counter and perform a new lookup for HOME and AL0. A Vertical Interrupt will not do a CRSR load, thus the pointer to ROW 1 will be preserved. At Vertical Interrupt, the row counter will be reset to 0 and we will want to do a pointer lookup for ROW 2 in preparation for the CRSR load at the end of ROW 1 (point A). Correspondingly, the row pointer lookup tables are organized 2 to 25, 1. Since the attribute latches aren't pipelined, the AL0 lookup table is arranged 1 to 25 since the new attribute set will be needed immediately for the display of the next row.

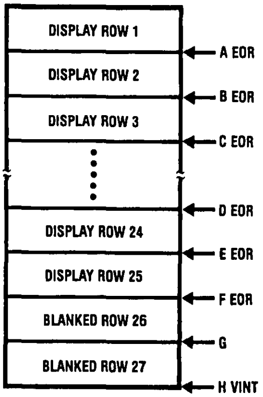
Row Table Lookup Pipelining



TL/C/5729-2

FIGURE 2

TAD Interrupt Positioning



TL/C/5729-3

FIGURE 3

TMP Attribute Demo Program

```


               .TITLE MAIN, "TMP ATTRIBUTE DEMO - TAD'

               ;                         James Murashige 10/05/83
               ;This program displays the various character attributes available with
               ;the TMP by dynamically updating the attribute latch each display row.
               ;In addition it uses End of Row and Vertical interrupts to perform row
               ;table lookup screen refreshing.

          0000 LINE 1  =    0     ;LINE 1 START, ALL BLANKS
          0050 LINE 2  =   80     ;LINE 2 START, NORMAL MESSAGE
          00A0 LINE 3  =  160     ;LINE 3 START, DOUBLE WIDE MESSAGE
          00F0 LINE 4  =  240     ;LINE 4 START, FIRST GRAPHICS LINE
          0140 LINE 5  =  320     ;LINE 5 START, SECOND GRAPHICS LINE
          0190 LINE 6  =  400     ;LINE 6 START, THIRD GRAPHICS LINE
          01E0 LINE 7  =  480     ;LINE 7 START, FOURTH GRAPHICS LINE



          0000 . = 00                        ;START AT PROGRAM LOCATION 0

     0000 0468 RESET;  JMP BEGIN             ;VECTOR TO RESET CODE

          0003 . = 03

               EXI:                          ;VECTOR TO EXTERNAL INTERRUPT PROCESSING

          0007 . = 07

               INI:                          ;VECTOR TO INTERNAL INTERRUPT PROCESSING

     0007 8C           MOV A, INTR         ;READ INTERRPUT REGISTER
     0008 320C         JBI EOR             ;HAVE AN EOR INTERRPUT
     000A BBFF         MOV R3, #OFF        ;VINT INTERRUPT
     000C 1B   EOR:    INC R3              ;INCREMENT TO DO NEXT ROW
     000D 234F         MOV A, #ATTO        ;GET ATTRIBUTE LATCH 0
     000F 6B           ADD A, R3
     0010 B3           MOVP A, @A
     0011 3C           MOV ALO, A          ;LOAD ATTRIBTE LATCH 0

     0012 231D         MOV A, #HOMHIG      ;GET HOME HIGH ORDER BYTE
     0014 6B           ADD A, R3
```

TMP Attribute Demo Program (Continued)

```
0015 B3              MOVP A, @A
0016 C2              MOV HACC, A
0017 2336            MOV A, #HOMLOW  ;GET HOME LOW ORDER BYTE
0019 6B              ADD A, R3
001A B3              MOVP A, @A
001B 8A              MOV HOME, A     ;LOAD HOME
001C 93              RETR
             .FORM

             ;HOME HIGH ORDER BYTE LOOKUP TABLE

001D 00      HOMHIG: .BYTE 0            ;ROW 2
001E 00              .BYTE 0            ;ROW 3
001F 00              .BYTE 0            ;ROW 4
0020 00              .BYTE 0            ;ROW 5
0021 00              .BYTE 0            ;ROW 6
0022 00              .BYTE 0            ;ROW 7
0023 00              .BYTE 0            ;ROW 8
0024 00              .BYTE 0            ;ROW 9
0025 00              .BYTE 0            ;ROW 10
0026 00              .BYTE 0            ;ROW 11
0027 00              .BYTE 0            ;ROW 12
0028 00              .BYTE 0            ;ROW 13
0029 00              .BYTE 0            ;ROW 14
002A 00              .BYTE 0            ;ROW 15
002B 00              .BYTE 0            ;ROW 16
002C 00              .BYTE 0            ;ROW 17
002D 00              .BYTE 0            ;ROW 18
002E 01              .BYTE H(LINE5)     ;ROW 19
002F 00              .BYTE 0            ;ROW 20
0030 01              .BYTE H(LINE6)     ;ROW 21
0031 00              .BYTE 0            ;ROW 22
0032 01              .BYTE H(LINE7)     ;ROW 23
0033 00              .BYTE 0            ;ROW 24
0034 00              .BYTE 0            ;ROW 25
0035 00              .BYTE 0            ;ROW 1
0036 00              .FORM

             ;HOME LOW ORDER BYTE LOOKUP TABLE

0036 00      HOMLOW: .BYTE 0            ;ROW 2 BLANK
0037 50              .BYTE L(LINE2)     ;ROW 3 NORMAL
0038 00              .BYTE 0            ;ROW 4 BLANK
0039 A0              .BYTE L(LINE3)     ;ROW 5 DOUBLE WIDE
003A 00              .BYTE 0            ;ROW 6 BLANK
003B 50              .BYTE L(LINE2)     ;ROW 7 DOUBLE HIGH
003C 50              .BYTE L(LINE2)     ;ROW 8 DOUBLE HIGH
003D 00              .BYTE 0            ;ROW 9 BLANK
003E A0              .BYTE L(LINE3)     ;ROW 10 DOUBLE SIZE
003F A0              .BYTE L(LINE3)     ;ROW 11 DOUBLE SIZE
0040 00              .BYTE 0            ;ROW 12 BLANK
0041 A0              .BYTE L(LINE3)     ;ROW 13 DOUBLE SIZE
0042 A0              .BYTE L(LINE3)     ;ROW 14 DOUBLE SIZE
0043 00              .BYTE L(LINE1)     ;ROW 15 BLANK
0044 00              .BYTE L(LINE1)     ;ROW 16 BLANK
0045 F0              .BYTE L(LINE4)     ;ROW 17 GRAPHICS
0046 00              .BYTE L(LINE1)     ;ROW 18 BLANK
0047 40              .BYTE L(LINE5)     ;ROW 19 GRAPHICS
0048 00              .BYTE L(LINE1)     ;ROW 20 BLANK
0049 90              .BYTE L(LINE6)     ;ROW 21 GRAPHICS
004A 00              .BYTE L(LINE1)     ;ROW 22 BLANK
004B E0              .BYTE L(LINE7)     ;ROW 23 GRAPHICS
004C 00              .BYTE L(LINE1)     ;ROW 24 BLANK
004D 00              .BYTE L(LINE1)     ;ROW 25 BLANK
004E 00              .BYTE L(LINE1)     ;ROW 1 BLANK
```

TMP Attribute Demo Program (Continued)

```
                   .FORM

                   ;ATTRIBUTE LATCH 0 LOOKUP TABLE

004F FF            ATT0:   .BYTE 0FF          ;ROW 1
0050 FF                    .BYTE 0FF          :ROW 2
0051 FF                    .BYTE 0FF          :ROW 3
0052 FF                    .BYTE 0FF          :ROW 4
0053 EF                    .BYTE 0EF          :ROW 5
0054 FF                    .BYTE 0FF          :ROW 6
0055 F7                    .BYTE 0F7          :ROW 7
0056 B7                    .BYTE 0B7          :ROW 8
0057 FF                    .BYTE 0FF          :ROW 9
0058 E7                    .BYTE 0E7          :ROW 10
0059 A7                    .BYTE 0A7          :ROW 11
005A FF                    .BYTE 0FF          :ROW 12
005B E2                    .BYTE 0E2          :ROW 13
005C 82                    .BYTE 082          :ROW 14
005D FF                    .BYTE 0FF          :ROW 15
005E FF                    .BYTE 0FF          :ROW 16
005F 7F                    .BYTE 07F          :ROW 17
0060 FF                    .BYTE 0FF          :ROW 18
0061 7F                    .BYTE 07F          :ROW 19
0062 FF                    .BYTE 0FF          :ROW 20
0063 7F                    .BYTE 07F          :ROW 21
0064 FF                    .BYTE 0FF          :ROW 22
0065 7F                    .BYTE 07F          :ROW 23
0066 FF                    .BYTE 0FF          :ROW 24
0067 FF                    .BYTE 0FF          :ROW 25
                   .FORM

                   ;START OF INITIALIZING CODE

0068 15            BEGIN;  DIS XI          ;INTERRUPTS OFF FOR NOW
0069 35                    DIS II
006A 65                    STOP T
006B 231A                  MOV A, #26
006D A2                    MOV VINT, A
006E 27                    CLR A           ;SET UP TIMING CHAIN FOR DEMO BOARD
006F 87                    MOV TCP, A
0070 2367                  MOV A, #103     ;HORIZONTAL LENGTH
0072 B7                    MOV @TCP, A
0073 234F                  MOV A, #79      ;CHARACTERS/ROW
0075 B7                    MOV @TCP, A
0076 2353                  MOV A, #83      ;HORIZONTAL SYNC BEGIN
0078 B7                    MOV @TCP, A
0079 2363                  MOV A, #99      ;HORIZONTAL SYNC END
007B B7                    MOV @TCP, A
007C 2391                  MOV A, #091     ;CHARACTER HEIGHT/EXTRA SCANS
007E B7                    MOV @TCP, A
007F 231A                  MOV A, #26      ;VERTICAL LENGTH
0081 B7                    MOV @TCP, A
0082 2318                  MOV A, #24      ;VERTICAL BLANK
0084 B7                    MOV @TCP, A
0085 2362                  MOV A, #062     ;VERTICAL SYNC BEGIN/END
0087 B7                    MOV @TCP, A
0088 231E                  MOV A, #30      ;STATUS ROW BEGIN
008A B7                    MOV @TCP, A
008B 23F4                  MOV A, #0F4     ;BLINK RATE
008D B7                    MOV @TCP, A
008E 2330                  MOV A, #030     ;GRAPHICS COLUMN REGISTER
0090 B7                    MOV @TCP, A
0091 2336                  MOV A, #036     ;GRAPHICS ROW REGISTER
0093 B7                    MOV @TCP, A
0094 2389                  MOV A, #089     ;UNDERLINE SIZE REGISTER
```

TMP Attribute Demo Program (Continued)

```
0096 B7              MOV @TCP, A
0097 2309            MOV A, #009     ;CURSOR SIZE REGISTER
0099 B7              MOV @TCP, A


            .FORM
009A 2324            MOV A, #024     ;SET SYSTEM CONTROL REGISTER
009C 55              MOV SCR, A      ;8 BI,7 DOTS, DIVIDE 1, TABLE LOOKUP

009D C3              SEL RB0         ;SELECT RAM BANK 0
009E 27              CLR A           ;SET RAM POINTERS
009F C2              MOV HACC, A
00A0 8A              MOV HOME, A
00A1 0D              MOV BEGD, A
00A2 88              MOVL R0, A      ;CLEAR MEMORY POINTER

00A3 237F            MOV A, #07F
00A5 C2              MOV HACC, A
00A6 23FF            MOV A, #0FF
00A8 0C              MOV ENDD, A
00A9 8B              MOV CURS, A
00AA 3D              MOV AL1, A      ;NO ATTRIBUTES FOR LATCH 1

            ;CLEAR OUT MEMORY

00AB BD19            MOV R5, #25     ;DO 25 ROWS
00AD BA50            MOV R2, #80     ;DO 80 CHARACTERS PER ROW
00AF 23A0            MOV A, #0A0     ;INITIALIZE FOR A SPACE, ATTRIBUTE LATCH 1

00B1 80     LOOP:    MOVX @R0, A     ;STORE A CHARACTER
00B2 38              INCL R0         ;INCREMENT POINTER
00B3 EAB1            DJNZ R2, LOOP   ;TEST IF ROW DONE
00B5 BA50            MOV R2, #80
00B7 EDB1            DJNZ R5, LOOP   ;TEST IF SCREEN DONE

00B9 2321            MOV A, #021     ;SET VCR FOR INTERNAL ATTRIBUTES
00BB 45              MOV VCR, A      :INTERNAL CHARACTER GENERATOR


            ;FIRST LINE ARE ALL BLANKS, SECOND LINE HAS SINGLE SPACING MESSAGE
00BC 2300            MOV A, #H(LINE2+30)  ;SET R0 POINTER TO FIRST LINE
00BE C2              MOV HACC, A
00BF 236E            MOV A, #L(LINE2+30)
00C1 88              MOVL R0, A
00C2 BAE0            MOV R2, #L(MSG1)  ;SET R2 TO MESSAGE #1
00C4 BB13            MOV R3, #19     ;SET R3 TO MESSAGE LENGTH
00C6 FA     DISP1:   MOV A, R2
00C7 B3              MOV A, @A       ;DISPLAY NORMAL MESSAGE
00C8 80              MOVX @R0,A
00C9 38              INCL R0
00CA 1A              INC R2
00CB EBC6            DJNZ R3, DISP1
            .FORM
            ;THIRD LINE HAS DOUBLE WIDE MESSAGE
00CD 98              MOVL A, R0                ;SET R0 POINTER
00CE 0334            ADD A, #(31 + 21)        ;LINES3 + 21
00D0 88              MOVL R0, A
00D1 BAE0            MOV R2, #L(MSG1)
00D3 BB13            MOV R3, #19
00D5 FA     DISP2:   MOV A, R2
00D6 B3              MOVP A, @A      ;DISPLAY DOUBLE WIDE
00D7 80              MOVX @R0, A
00D8 38              INCL R0
00D9 80              MOVX @R0, A
00DA 38              INCL R0
```

TMP Attribute Demo Program (Continued)

```
00DB 1A                INC R2
00DC EBD5              DJNZ R3, DISP2
00DE 04F3              JMP FOURTH

00E0 74       MSQ1:    .BYTE 'tmp does it BETTER!'

              ;FOURTH LINE STARTS GRAPHICS CHARACTERS DISPLAY
00F3 98       FOURTH:  MOVL A, R0
00F4 031D              ADD A, #(21 + 8)     ;LINE4 + 8
00F6 88                MOVL R0, A
00F7 BB04              MOV R3, #4       ;DO 4 LINES
00F9 BA20              MOV R2, #32      ;DO 32 GRAPHICS CHARACTERS PER LINE
00FB 2300              MOV A, #000       ;ATTRIBUTE LATCH 0 SELECTED
00FD 2400              JMP BLOOP

     0100       . = 0100

0100 80       BLOOP:   MOVX @R0, A      ;STORE CHARACTER
0101 38                INCL R0
0102 38                INCL R0
0103 17                INC A
0104 EA00              DJNZ R2, BLOOP

0106 BA20              MOV R2, #32     ;INITIALIZE FOR NEW ROW
0108 AC                MOV R4, A       ;TEMPORARY SAVE A
0109 98                MOVL A, R0
010A 0310              ADD A, #(8+8)   ;POINT TO NEXT LINE
010C 88                MOVL R0, A
010D FC                MOV A, R4       ;RESTORE A
010E EB00              DJNZ R3, BLOOP  ;CONTINUE IF NOT THROUGH

              ;REENABLE INTERNAL INTERRUPTS AND MASK OFF UNUSED ONES
0110 2303              MOV A, #03
0112 82                MOV MASK, A
0113 25                EN II           ;REENABLE INTERNALS
0114 2414     PAU:     JMP PAU         ;WAIT FOR A VIDEO INTERRUPT
              END
ATTO    004F    BEGIN   0068     BLOOP   0100     DISP1   00C6
DISP2   00D5    EOR     000C     EXI     0003 *   FOURTH  00F3
HOMHIG  001D    HOMLOW  0036     INI     0007 *   LINE1   0000
LINE2   0050    LINE3   00A0     LINE4   00F0     LINE5   0140
LINE6   0190    LINE7   01E0     LOOP    00B1     MSG1    00E0
PAU     0114    RESET   0000 *

NO ERROR LINES
  272 ROM BYTES USED

SOURCE CHECKSUM=CF60
OBJECT CHECKSUM=0576

INPUT FILE A: TAD. MAC
LISTING FILE A: TAD. PRN
OBJECT FILE A: TAD. LM
```

TMP - Dynamic RAM Interfacing

National Semiconductor
Application Note 355
James Murashige

TMPs Interface easily and directly to dynamic RAMs as illustrated in the basic TMP system schematic of *Figure 1*. In addition to providing the necessary Read/Write cycle control, the TMP will also automatically refresh the memories through the video controller, further easing interface requirements.

The circuitry to the right of the TMP provides program memory interfacing and I/O support while to the left lie the dynamic video RAM circuits. The memory width shown here is 8 bits although 16 bits can easily be accommodated. Using the 64K × 1 dynamic RAMs shown the entire video memory space is filled with RAM. However, by using a slightly modified addressing configuration smaller memroy chips could be substituted.

The requisite dynamic RAM control signals $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ are generated directly from the system bus control signals RAM ALE, $\overline{RAM\ RD}$ and $\overline{RAM\ WR}$. RAM ALE is used directly as $\overline{RAS}$ while $\overline{RAM\ WR}$ serves as $\overline{WE}$. $\overline{CAS}$ is the logical AND of $\overline{RAM\ RD}$ and $\overline{RAM\ WR}$. The 16 system bus bits are multiplexed down to the 8 bit RAM address vector by the two 74LS157's under the control of the RAM ALE. As configured, the row and column addresses strobed in are SBO-7 and SB8-15 respectively.

With the configuration shown, the pertinent TMP Read and Write cycle timing parameters for *Figure 2* are listed in Table 1. Going through the table one sees that the TMP easily interfaces to 150 ns access RAMs and will routinely work with 200 ns RAMs. The four parameters which may be a tight squeeze for 200 ns RAMs are:

1. t_{RAC}— Access Time from $\overline{RAS}$ is max 150 ns, typ 220 ns. This is a basic access time requirement which necessitates fast parts.
2. t_{RAH}— Row Address Hold Time is min 10 ns, typ 15 ns. This parameter is entirely dependent on the switching speed of the 74LS157.
3. t_{RCD}— $\overline{RAS}$ to $\overline{CAS}$ Delay Time is min 10 ns, typ 50 ns. This parameter isn't too critical since most dynamic RAMs internally gate the $\overline{CAS}$ signal should it come along too early.
4. t_{RP}— $\overline{RAS}$ Precharge Time is min 100 ns, typ 135 ns. Since $\overline{RAS}$ is actually the RAM ALE signal t_{RP} is the high time of RAM ALE.

However, rather than getting faster RAMs one could also meet spec by running the TMP CPU slower, thereby stretching out the allowable access time.

Since the TMP video controller will regularly and automatically access video memory in order to obtain characters for display, one may have dynamic RAM refreshing performed automatically by making sure that the required number of consecutive address locations (ROW Addresses) are accessed in the alloted time. Typically this is 128 ROW addresses in 2 ms.

For example, in a typical system we may have an 80 column by 25 row display with each row consisting of 10 scan lines. Each scan line has a period of 60.67 us. The vertical blank period consists of 25 scan lines for a total duration of 1.52 ms. Assuming sequential rather than table lookup operation, 80 consecutive character addresses are accessed each scan line and a 160 consecutive character addresses are accessed every 2 rows; more than enough to refresh all of the $\overline{RAS}$ rows. Of course one must be sure that the memory addresses of any two consecutive rows encompass all 128 possible $\overline{RAS}$ addresses. In the middle of the screen the worst case refresh period is 11 scan lines (.667 ms), since to do 160 consecutive addresses requires one complete row plus the first scan line of the next row. At the bottom of the screen the refresh period must also include the vertical blank time since no video characters are accessed then. In this case refresh stretches out to a worst case 2.184 ms.

Although in this example we exceeded the 2 ms refresh period, there are a number of things that we could do to get things back into spec. For example, we could cut down on vertical blank time, use memory chips with longer refresh periods, or have the CPU refresh video memory during vertical retrace. Taking the case of using different memory chips, another popular refreshing arrangement is 256 row addresses in 4 ms. In the middle of the screen this gives us a worst case period of 10 + 10 + 10 + 1 scan lines or 31 × 60.67 us = 1.88 ms. Adding in the vertical blanking period the absolute worst case refresh delay is 1.88 + 1.52 = 3.4 ms. Of course in this arrangement, making sure that any four consecutive rows encompass all 256 RAS addresses is much more difficult.

When operating in pixel mode meeting refresh requirements isn't as difficult since each scan line will access a different set of consecutive RAM addresses.

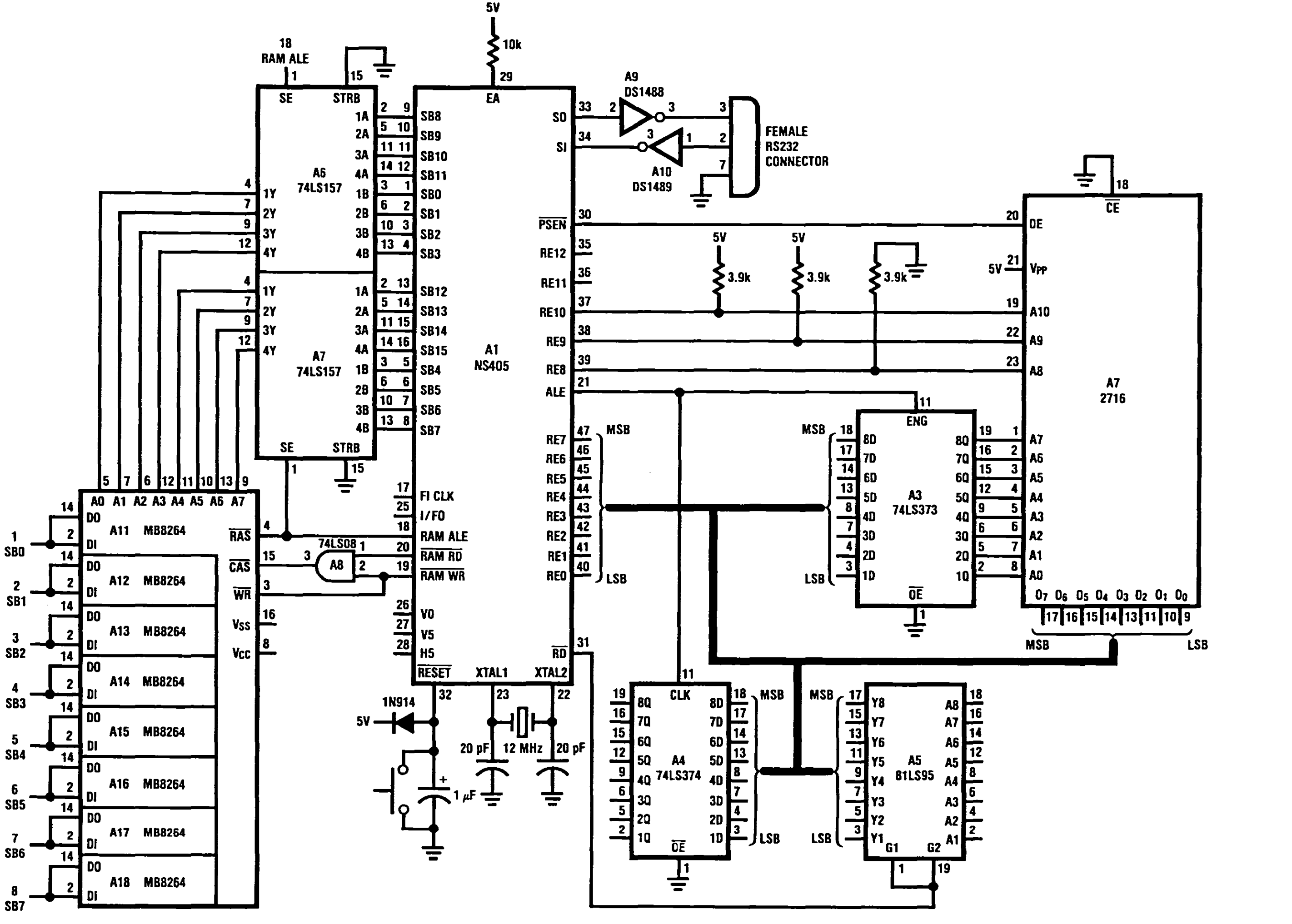
Returning to the circuit of *Figure 1*, we have assumed that SB0–7 are multiplexed address/data while SB8-15 output addresses only. Since the RAM addresses are latched in 8 bits at a time there is no need for a separate latch for SB0–7 since all 8 bits are clocked in on the falling ALE edge. However, when operating with smaller 8K or 16K RAMs where only 7 bits are clocked in at a time, latching arrangements for SB7 must be made. An example of this is shown in *Figure 3* where bits SB0–7 are all latched by the 74LS373.

Normally I/O registers, as well as other memory banks, will also be memory mapped into the 64K video RAM space. In order to do this some sort of chip enabling scheme must be worked out since the dynamic RAMs have no direct enable control. One possibility is shown in *Figure 4* where the RAM bank $\overline{CAS}$ and $\overline{WE}$ are disabled unless selected by the 74LS138 decoder. In this way the RAM output drivers will remain TRI-STATE® and no data will be written unless the bank is selected. However, memory refreshing as controlled by $\overline{RAS}$ will still be performed on each RAM bank.

By expanding on these basic examples a memory configuration for the TMP utilizing dynamic memories may be quickly and easily worked out.

TABLE 1. TMP Dynamic RAM Interface Timing 12 MHz CPU

Symbol	Parameter	Min	Typ	Max	Units
t_{AR}	Column Address Hold Time Referenced to $\overline{RAS}$	250	280		ns
t_{ASC}	Column Address Set Up Time–Dependent on Switching of 74LS157	25	35		ns
t_{ASR}	Row Address Set Up Time	20	90		ns
t_{CAC}	Access Time from $\overline{CAS}$		180	140	ns
t_{CAH}	Column Address Hold Time	140	250		ns
t_{CAS}	$\overline{CAS}$ Pulse Width	140	160		ns
t_{CP}	$\overline{CAS}$ Precharge Time	140	166		ns
t_{CRP}	$\overline{CAS}$ to $\overline{RAS}$ Precharge Time	100	136		ns
t_{CSH}	$\overline{CAS}$ Hold Time	250	280		ns
t_{CWL}	Write Command to $\overline{CAS}$ Lead Time	140	160		ns
t_{DH}	Data In Hold Time	160	175		ns
t_{DHR}	Data In Hold Time Referenced to $\overline{RAS}$	180	310		ns
t_{DS}	Data In Set Up Time	10	50		ns
t_{OFF}	Output Buffer Turn Off Delay	0		60	ns
t_{RAC}	Access Time from $\overline{RAS}$		220	150	ns
t_{RAH}	Row Address Hold Time–Dependent on Switching of 74LS157	10	15		ns
t_{RAS}	$\overline{RAS}$ Pulse Width	250	280		ns
t_{RC}	Random Read/Write Cycle Time	416			ns
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	10	50		ns
t_{RCH}	Read Command Hold Time	100	175		ns
t_{RCS}	Read Command Set Up Time	100	175		ns
t_{RP}	$\overline{RAS}$ Precharge Time	100	135		ns
t_{RRH}	Read Command Hold Time Referenced to $\overline{RAS}$	100	175		ns
t_{RSH}	$\overline{RAS}$ Hold Time	140	160		ns
t_{RWL}	Write Command to $\overline{RAS}$ Lead Time	140	150		ns
t_{WCH}	Write Command Hold Time	140	150		ns
t_{WCR}	Write Command Hold Time Referenced to $\overline{RAS}$	160	275		ns
t_{WCS}	Write Command Set Up Time–Dependent on Delay of 74LS08	5	11		ns
t_{WP}	Write Command Pulse Width	140	150		ns

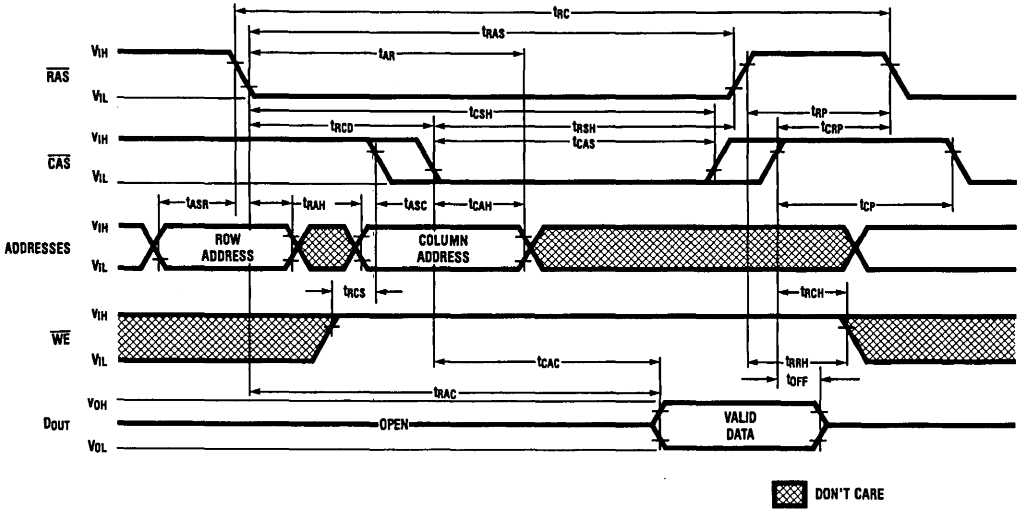


TL/C/5732-1

FIGURE 1. TMP with 64K Dynamic Memory

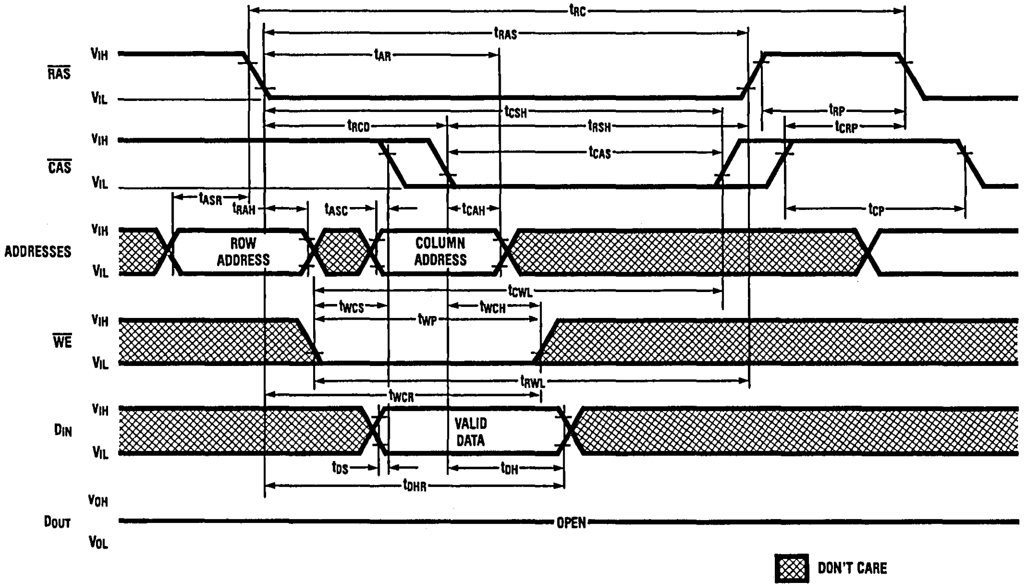
Timing Diagrams

Read Cycle Timing Diagram



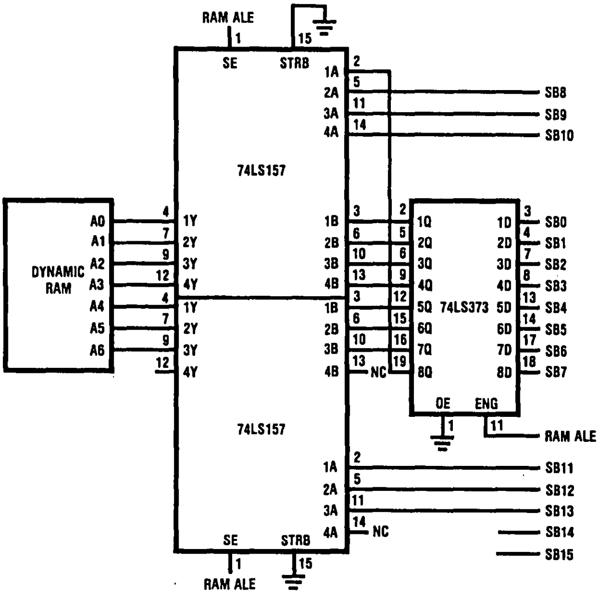
TL/C/5732-2

Write Cycle (Early Write)



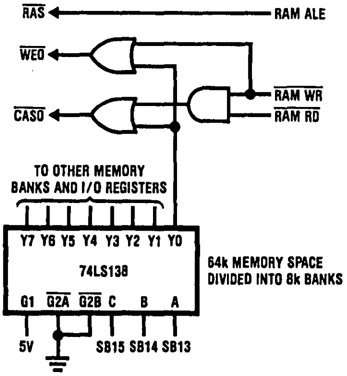
TL/C/5732-3

FIGURE 2.



TL/C/5732-4

FIGURE 3. TMP Address Multiplexing for 16K Dynamic RAMs



TL/C/5732-5

FIGURE 4. Chip Enabling Dynamic RAMs

TMP External Character Generation

National Semiconductor
Application Note 367
James Murashige

Built into the TMP video circuitry is the ability to access an external character generator to display custom FONT sets. In addition to the flexibility afforded by user selectable FONTs, by going "external" the number of different character patterns directly addressable is virtually limitless. On the other hand the disadvantages of going external are the additional hardware necessary to control data routing and the general need to use faster memories.

Figure 1 shows a minimum configuration with which to do external character generation. In the TMP, external character generation is selected through Video Control Register bits 6, 7 and is a cross between normal alphanumeric and pixel graphics display modes. Like normal alphanumeric mode the TMP sequences through the video memory address space based upon the screen format specification. But instead of routing the data through the internal character generator, it is treated as pixel data and directly inserted into the video dot stream. In effect what we are doing externally is duplicating the internal character generator ROM. In external mode video attributes are fully operational except for double height and block graphics.

Operation of the circuit shown is straight forward and follows a pipe-line approach. On a video data read the display memory address is output onto the system bus with the 8 low order bits being latched by the 74LS373. On the RAM RD signal the 2116 display RAM ouputs a data character onto the pipeline bus which is used to address the MM52116 character generator which in turn deposits the required pixel data onto the system bus so that it may be read in. The 2116 determines which character is to be looked up in the 52116 while the 74LS163 tells the character generator which row in the character we wish to look at. The 74LS163 is a counter which is appropriately clocked by the horizontal sync pulse so that we will advance each scan line to point to the next row in the character FONT. At the end of each screen row the counter must be cleared in preparation for the display of a new row. This is the function of the Scan Count Clear signal which is available as a multiplexed output on the RE11 pin. It is a low going signal which pulses for 1 scan line time during the last scan line in a screen ROW. Its timing is shown in Figure 2. Note that since the 74LS163 is a fully synchronous counter the clear input will not be accepted until the very last H-Sync clock pulse in the screen row. Because of the necessity to not clear the counter before all pixel data is brought in, nor to delay clocking lest the Scan Count Clear pulse be missed, the starting H-Sync clock edge must be postioned close to the start of horizontal blanking.

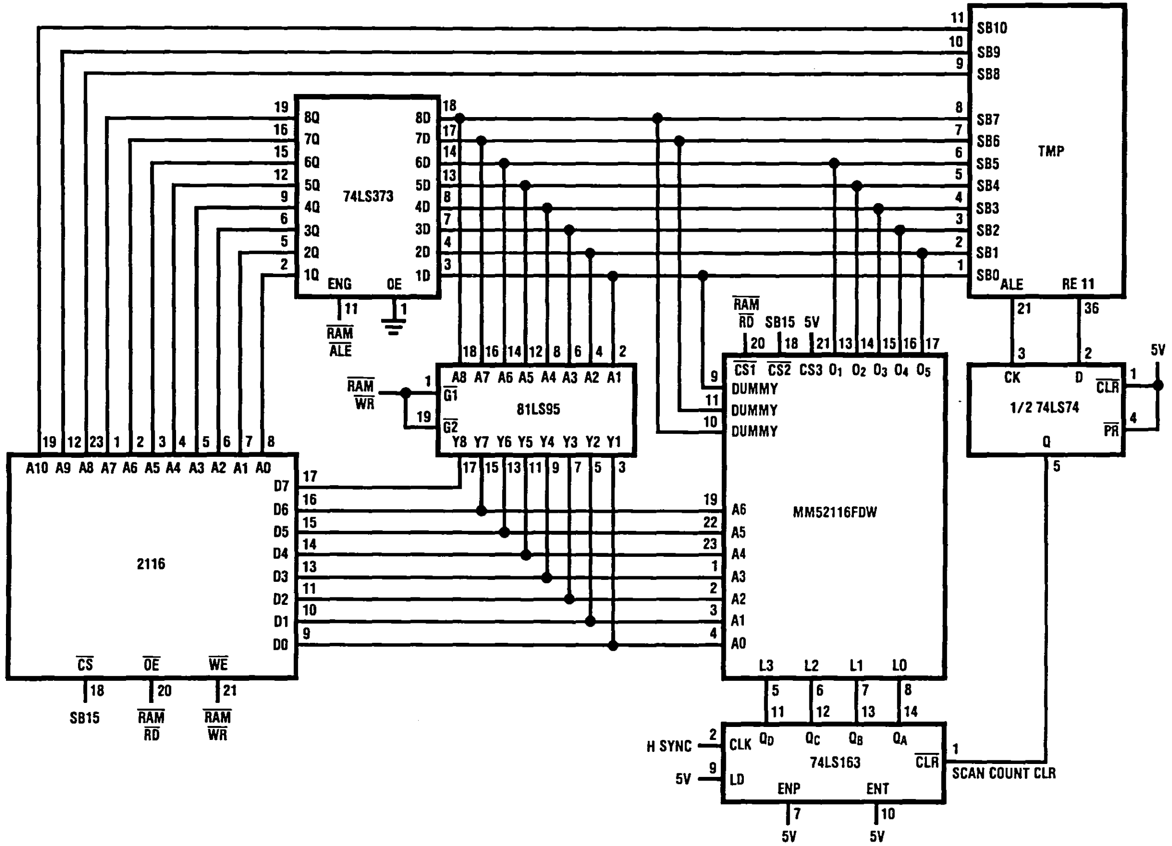
Continuing with the read operation, we see that video RAM is only accessed if SB15 is low, i.e., the lower 32K. Note that the 52116 used here contains 128 characters in a 5 × 7 FONT. Consequently, it has 5 data output lines connected to the system bus. The other three "dummy" lines shown connected are actually output bits which are always 0 by default, thus giving us blank spaces. There are two reasons why the character bits start on SB1. The first is that since everything brought in is considered pixel data, spaces between characters must be externally inserted. The second is that the video controller always brings in 8 bits even though the cell width can be defined to be 9 or 10. In these instances the 9th and 10th bits repeat what was encoded into the SB0 bit. As a result external characters can practically be at most 7 dots wide although the cells can be up to 8, 9, or 10 dots wide. Cell and/or character heights can be up to 16 lines tall as specified by the Character Scan Height Register.

On a video memory write, data is routed through the 81LS95, onto the pipe-line bus and into the 2116. Writing into the 2116 is controlled by RAM WR as shown. Ordinarily the MSB data bit is used for internal attribute latch selection and could be directly connected to the SB7 line if character cells were specified to at most be 7 dots wide. Otherwise SB7 will be needed for pixel generation as shown in Figure 1, thereby rendering internal attribute latch selection useless. In this case both internal attribute latches would have to be loaded with the same values. As shown here, 7 video RAM data bits are used to address the 128 possible characters in the 52116. If a larger character generator were available, additional data bits could be used to select from a larger character set. Since the TMP features a 16-bit multiplexed address/data bus, by using all 16 available data bits we could address 65,536 different character patterns

With the video data pipe-lined as shown, very fast memory circuits are required for external character generation. With a 12 MHz CPU clock, character pixel data must be available within a max of 220ns (typ. 300ns) after an address goes out. To accomplish this the character generator will typically have to be bipolar and the video RAM fast MOS. However, if faster memories are a problem, access times may be stretched out by slowing down the CPU clock since video RAM cycllng is based on the CPU clock. For instance with a CPU clock of 8 MHz, access time stretches out to 385 ns max, 500 ns typ. If using the divide by 1.5 factor on the crystal to obtain the slower CPU clock, remember that due to system constraints the character cell MUST BE AT LEAST 8 DOTS WIDE. In Figure 1 the 2116 output enable is shown being driven by RAM RD. Although this may seem redundant and will slow things down (why not just leave the output enabled?) it is necessary in order to avoid bus conflict when doing a memory write operation.

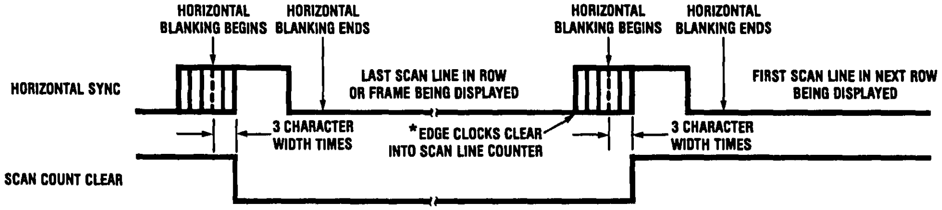
By expanding on this basic circuit, numerous options such as external attributes, expanded character sets and dynamic RAM may be added to achieve the desired end system.

TMP EXTERNAL CHARACTER GENERATOR



TL/C/5731-1

SCAN COUNT CLEAR TIMING


HORIZONTAL BLANKING BEGINS
HORIZONTAL BLANKING ENDS
HORIZONTAL BLANKING BEGINS
HORIZONTAL BLANKING ENDS
HORIZONTAL SYNC
LAST SCAN LINE IN ROW OR FRAME BEING DISPLAYED
FIRST SCAN LINE IN NEXT ROW BEING DISPLAYED
3 CHARACTER WIDTH TIMES
*EDGE CLOCKS CLEAR INTO SCAN LINE COUNTER
3 CHARACTER WIDTH TIMES
SCAN COUNT CLEAR

TL/C/5731-2

* Edge must come before Scan Count Clear goes away but not before the video controller has brought in all necessary display information for the last scan line. Edge should not be more than 3 character widths from the beginning of blanking.

NS405 TMP Logic Analyzer

National Semiconductor
Application Note 369
James Murashige

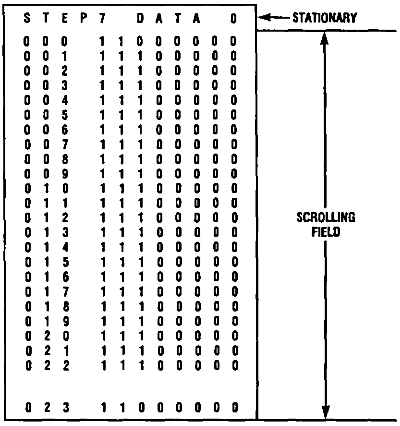
INTRODUCTION

The NS405 TMP is ideally suited for use in Test and Instrumentation equipment as the system or display controller. To demonstrate this, the following note describes how to turn the NS405 Demo Board into a simple 8 bit Logic State Analyzer. Featured in this system is a data capacity of 156 eight bit words, 21 μs data acquisition time, keyboard command entry, UP/DOWN rolling scroll and 24 line data display.

SYSTEM ARCHITECTURE

All of the necessary resources to build our system are available in a TMP Demo Board system when normally set up as a data terminal. Commands are entered through the attached ASCII encoded keyboard with data being strobed on the external interrupt. Data words are input through the switch configuration register SW2 by strobing the Light Pen interrupt. Video is output to the attached display monitor. The only real difference between our Logic Analyzer and the Data Terminal is the ROM software in U9 running the TMP. An overview of the system is shown in *Figure 1*.

In order to maximize the available 2k of video RAM, a display line length of 13 was chosen. This yields 157 lines of display information (157 × 13 = 2041), one of which is used to display title information. Thus our display data field consists of 156 lines of information, any 24 of which may be displayed at any given time. On each line is displayed the STEP number, followed by 2 spaces and 8 bits of 1 or 0 information. A typical display pattern is illustrated in *Figure 2*. By manipulating the pointer registers in the TMP DMA controller, the Title line is made to be stationary while the rest of the screen scrolls. This is accomplished by reversing the roles of the HOME register and Status Section SROW pointer. Specifically HOME points to the last row in memory which holds the title information while the status section is set to start after the display of the first row. Scrolling is accomplished by bumping the SROW pointer up or down 1 line width and checking for end of memory conditions.



```
S T E P 7    D A T A    0
0 0 0    1 1 0 0 0 0 0 0
0 0 1    1 1 1 0 0 0 0 0
0 0 2    1 1 1 0 0 0 0 0
0 0 3    1 1 1 0 0 0 0 0
0 0 4    1 1 1 0 0 0 0 0
0 0 5    1 1 1 0 0 0 0 0
0 0 6    1 1 1 0 0 0 0 0
0 0 7    1 1 1 0 0 0 0 0
0 0 8    1 1 1 0 0 0 0 0
0 0 9    1 1 1 0 0 0 0 0
0 1 0    1 1 1 0 0 0 0 0
0 1 1    1 1 1 0 0 0 0 0
0 1 2    1 1 1 0 0 0 0 0
0 1 3    1 1 1 0 0 0 0 0
0 1 4    1 1 1 0 0 0 0 0
0 1 5    1 1 1 0 0 0 0 0
0 1 6    1 1 1 0 0 0 0 0
0 1 7    1 1 1 0 0 0 0 0
0 1 8    1 1 1 0 0 0 0 0
0 1 9    1 1 1 0 0 0 0 0
0 2 0    1 1 1 0 0 0 0 0
0 2 1    1 1 1 0 0 0 0 0
0 2 2    1 1 1 0 0 0 0 0

0 2 3    1 1 0 0 0 0 0 0
```

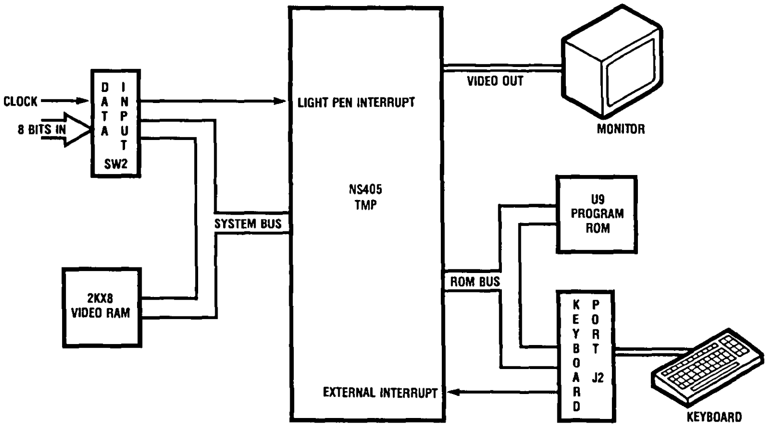
TL/DD/6970-2

FIGURE 2. TMP Logic Analyzer Screen Format

SYSTEM SOFTWARE

Since the system must rely on external events at several points before proceeding with processing, an interrupt driven approach was taken in structuring the software. A flowchart for the main program is shown in *Figure 3*. After system initialization there are 2 levels of processing associated with our logic analyzer operation. The first is a wait for an external interrupt signifying a new keyboard command. Referring to the keyboard service routine in *Figure 4*, the key is first read in and decoded as to function. In our simple system there are only 3 commands:

S or s = Start data acquisition
U or u = Scroll display up
I or i = Scroll display down



TL/DD/6970-1

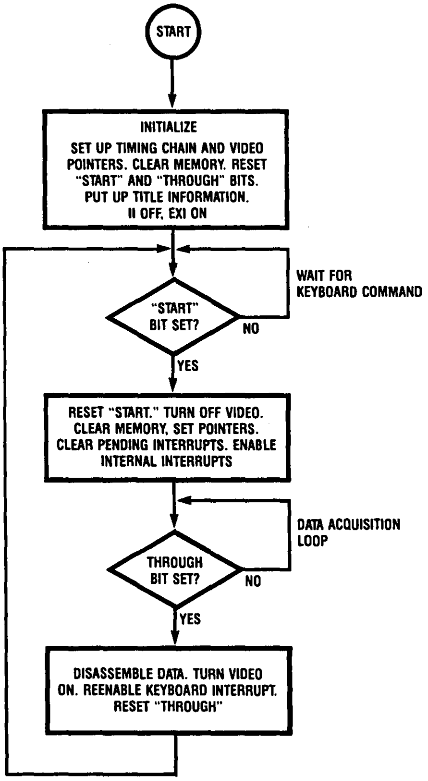
FIGURE 1. TMP Logic Analyzer (System Overview)

The scrolling functions are easily handled in the service routine by bumping the memory pointers and checking for an end of memory condition. A command to start data acquisition moves us to our second level of processing—the actual acquisition and display of data.

In both the keyboard and data acquisition interrupt service routines, flags F0 and F1 are used to pass system status back and forth from the main program. In this way the main program holds at major points while the service routines accomplish their functions. The data acquisition routine does nothing more than read data in from the SW2 port, store it in video memory and check a loop counter to see whether we have read in enough data. Since the Light Pen interrupt is being used, only high to low transitions will initiate an SW2 read. While very little is being done in data acquisition, it is time consuming because it's done in software. A count of instructions yields a worst case processing time of 21 μs between data strobes. In addition, since the data isn't latched it must remain stable until the actual read occurs. Following data acquisition, the stored data words are disassembled into their ASCII "1's" and "0's" patterns and the data entires numbered. With data acquistion completed, the program returns to await another keyboard command.

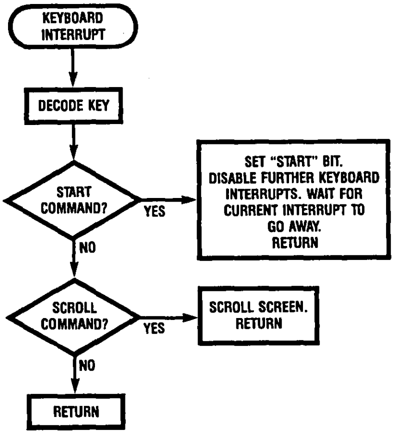
SUMMARY

As demonstrated, the NS405 is very effective as a display controller in a video instrumentation system. Certain functions, however, such as data acquisition are better left to dedicated hardware controllers. Nevertheless, the system as presented is still a very useful diagnostic tool. Through small enhancements to the hardware and software, features such as word recognition, number base conversion, wider data words and loop delay may readily be added.


START
INITIALIZE
SET UP TIMING CHAIN AND VIDEO POINTERS. CLEAR MEMORY. RESET "START" AND "THROUGH" BITS. PUT UP TITLE INFORMATION. II OFF. EXI ON
WAIT FOR KEYBOARD COMMAND
"START" BIT SET?
NO
YES
RESET "START." TURN OFF VIDEO. CLEAR MEMORY, SET POINTERS. CLEAR PENDING INTERRUPTS. ENABLE INTERNAL INTERRUPTS
DATA ACQUISITION LOOP
THROUGH BIT SET?
NO
YES
DISASSEMBLE DATA. TURN VIDEO ON. REENABLE KEYBOARD INTERRUPT. RESET "THROUGH"

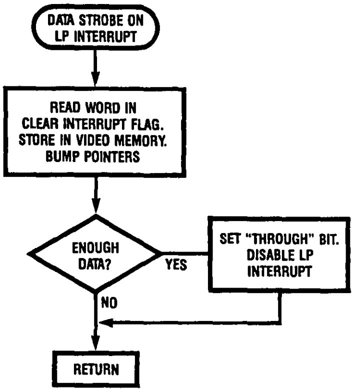
TL/DD/6970-3

FIGURE 3. TMP Demoboard Logic Analyzer Main Program


KEYBOARD INTERRUPT
DECODE KEY
START COMMAND?
YES
SET "START" BIT. DISABLE FURTHER KEYBOARD INTERRUPTS. WAIT FOR CURRENT INTERRUPT TO GO AWAY. RETURN
NO
SCROLL COMMAND?
YES
SCROLL SCREEN. RETURN
NO
RETURN

TL/DD/6970-4

FIGURE 4. TMP Demoboard Logic Analyzer Command Input Routine


DATA STROBE ON LP INTERRUPT
READ WORD IN CLEAR INTERRUPT FLAG. STORE IN VIDEO MEMORY. BUMP POINTERS
ENOUGH DATA?
YES
SET "THROUGH" BIT. DISABLE LP INTERRUPT
NO
RETURN

TL/DD/6970-5

FIGURE 5. TMP Demoboard Logic Analyzer Data Acquisition Routine

```
               .TITLE  MAIN,'TMP LOGIC ANALYZER DEMO'

               ;                       James Murashige 2/09/84
               ;This program turns the TMP Demo board into a simple 8 bit logic analyzer.
               ;Command inputs are entered from the attached ASCII keyboard while data
               ;acquisition takes place through the switch configuration socket, SW2.
               ;The DIP switch may have to be unsoldered from the board.  Data is strobed
               ;in with an external clock applied to Light Pen Interrupt on W8A.  Each time
               ;data acqusition is started 156 words of 8 bits each are acquired and displayed.
               ;Display is in the form of STEP location and the associated 8 bit 1's and 0's
               ;pattern.
               ;Commands are  S = Start data acqusition
               ;              U = Scroll display up
               ;              I = Scroll display down



     07DF      LSTLIN = 07DF       ;START OF LAST LINE
     07EB      MEMEND = 07EB       ;END OF MEMORY
     07EC      STLIN  = 07EC       ;START OF TITLE LINE
     0020      VON    = 020         ;VIDEO ON
     0000      VOFF   = 000         ;VIDEO OFF



     0000      . = 00             ;START AT PROGRAM LOCATION 0

0000 0452      RESET:  JMP BEGIN      ;VECTOR TO RESET CODE

     0003      . = 03

0003 0412      EXI:    JMP KEY          ;VECTOR TO KEYBOARD COMMAND DECODE

     0007      . = 07

               INI:                  ;DATA STROBE INTERRUPT SERVICE
0007 8C               MOV A,INTR      ;CLEAR OUT INTERRUPT
0008 91               MOVX A,@R1      ;GET DATA CHARACTER
0009 80               MOVX @RO,A      ;STORE CHARACTER AWAY
000A 98               MOVL A,RO       ;BUMP RO POINTER
000B 6B               ADD A,R3
000C 88               MOVL RO,A
000D EA11             DJNZ R2,NOTRU   ;CHECK IF THROUGH

000F B5               CPL F1    ;YES THROUGH, SET INDICATOR BIT
0010 35               DIS II    ;DISABLE LP INTERRUPT

0011 93        NOTRU:  RETR        ;RETURN
               .FORM
               ;KEYBOARD COMMAND DECODE
0012 E1        KEY:    IN PORT         ;KEYBOARD DATA READ
0013 53DF             ANL A,#ODF      ;CONVERT LOWER TO UPPER CASE
0015 AA               MOV R2,A        ;SAVE COPY IN R2
0016 D353             XRL A,#'S'
0018 C627             JZ START        ;GOTO START
001A FA               MOV A,R2
001B D355             XRL A,#'U'
001D C62B             JZ UP     ;GOTO SCROLL UP
001F FA               MOV A,R2
0020 D349             XRL A,#'I'
0022 C63C             JZ DOWN  ;GOTO SCROLL DOWN
0024 A624      CKOFF:  JNXI CKOFF       ;NOT A VALID KEY & WAIT FOR EXI TO GO AWAY
0026 93               RETR         ;RETURN

0027 95        START:  CPL FO   ;START BIT SET
0028 15               DIS XI   ;DISABLE FURTHER KEYBOARD INTERRUPTS
0029 0424             JMP CKOFF

002B 99        UP:     MOVL A,R1       ;SCROLL UP
```

TL/DD/6970-6

```
002C 030D           ADD A,#13       ;ADVANCE TO NEXT ROW
002E 89             MOVL R1,A       ;SAVE NEW VALUE
002F 0314           ADD A,#L(-L(STLIN))     ;CHECK FOR END OF DISPLAY
0031 E2             MOV A,HACC      ;SUBTRACT STLIN FROM A
0032 03F8           ADD A,#L(-H(STLIN) - 1) ;CARRY WILL BE SET IF A WAS > OR =
0034 E639           JNC UPTRU               ;NEW VALUE OK, LOAD SROW AND RETURN
0036 27             CLR A       ;RESET SROW TO BEGINNING
0037 C2             MOV HACC,A
0038 89             MOVL R1,A
0039 99      UPTRU:  MOVL A,R1       ;LOAD R1 INTO SROW
003A 0E             MOV SROW,A
003B 93             RETR

003C 99      DOWN:   MOVL A,R1       ;SCROLL DOWN
003D 03F3           ADD A,#-13      ;SUBTRACT TO NEXT ROW
003F AA             MOV R2,A        ;TEMP SAVE OF LOW ORDER
0040 E2             MOV A,HACC      ;NOW DO UPPER HALF
0041 03FF           ADD A,#OFF      ;CARRY WILL BE SET IF A WAS 12 OR MORE
0043 F64D           JC DNTRU        ;NEW VALUE OK, LOAD VALUE INTO SROW
0045 2307           MOV A,#H(LSTLIN)        ;RESET SROW TO LAST ROW
0047 C2             MOV HACC,A
0048 23DF           MOV A, #L(LSTLIN)
004A 89             MOVL R1,A
004B 0E             MOV SROW,A
004C 93             RETR
004D C2      DNTRU:  MOV HACC,A
004E FA             MOV A,R2
004F 89             MOVL R1,A
0050 0E             MOV SROW,A
0051 93             RETR
             .FORM

             ;START OF INITIALIZING CODE

0052 C5      BEGIN:  SEL MBO
0053 C3             SEL RBO
0054 15             DIS XI    ;INTERRUPTS OFF FOR NOW
0055 35             DIS II
0056 65             STOP T    ;TIMER OFF
0057 27             CLR A      ;SET UP TIMING CHAIN FOR DEMO BOARD
0058 87             MOV TCP, A
0059 2367           MOV A, #103     ;HORIZONTAL LENGTH
005B B7             MOV @TCP, A
005C 230C           MOV A, #12      ;CHARACTERS/ROW
005E B7             MOV @TCP, A
005F 2353           MOV A, #83      ;HORIZONTAL SYNC BEGIN
0061 B7             MOV @TCP, A
0062 2363           MOV A, #99      ;HORIZONTAL SYNC END
0064 B7             MOV @TCP, A
0065 2391           MOV A, #091     ;CHARACTER HEIGHT/ EXTRA SCANS
0067 B7             MOV @TCP, A
0068 231A           MOV A, #26      ;VERTICAL LENGTH
006A B7             MOV @TCP, A
006B 2318           MOV A, #24      ;VERTICAL BLANK
006D B7             MOV @TCP, A
006E 2362           MOV A, #062     ;VERTICAL SYNC BEGIN/END
0070 B7             MOV @TCP, A
0071 2300           MOV A, #00      ;STATUS ROW BEGIN
0073 B7             MOV @TCP, A
0074 23F4           MOV A, #0F4     ;BLINK RATE
0076 B7             MOV @TCP, A
0077 2330           MOV A, #030     ;GRAPHICS COLUMN REGISTER
0079 B7             MOV @TCP, A
007A 2336           MOV A, #036     ;GRAPHICS ROW REGISTER
007C B7             MOV @TCP, A
007D 2389           MOV A, #089     ;UNDERLINE SIZE REGISTER
007F B7             MOV @TCP, A
0080 2309           MOV A, #009     ;CURSOR SIZE REGISTER
0082 B7             MOV @TCP,A
```

TL/DD/6970-7

```
                .FORM
0083 2304               MOV A, #004     ;SET SYSTEM CONTROL REGISTER
0085 55                 MOV SCR, A      ;8 BI, 7 DOTS, DIVIDE 1, SEQUENTIAL LOOKUP

0086 27                 CLR A      ;SET VIDEO RAM POINTERS
0087 C2                 MOV HACC, A     ;ACCUMULATOR CLEARED
0088 0D                 MOV BEGD, A
0089 0E                 MOV SROW, A     ;SROW WILL BE OUT HOME
008A 88                 MOVL RO, A      ;CLEAR MEMORY POINTER
008B 89                 MOVL R1, A      ;1 IS SROW IMAGE
008C 2307               MOV A, #H(MEMEND +1)
008E C2                 MOV HACC, A
008F 23EC               MOV A,#L(MEMEND +1)
0091 0C                 MOV ENDD,A      ;SET END OF MEMORY POINTER
0092 8A                 MOV HOME, A     ;SET POINTER TO TITLE ROW
0093 23FF               MOV A, #OFF
0095 C2                 MOV HACC,A
0096 8B                 MOV CURS, A     ;NO CURSOR
0097 3C                 MOV ALO, A      ;NO ATTRIBUTES FOR LATCH 0
0098 3D                 MOV AL1, A      ;NO ATTRIBUTES FOR LATCH 1
0099 85                 CLR FO     ;FO IS "START" BIT
009A A5                 CLR F1     ;F1 IS "THROUGH" BIT
009B 2320               MOV A,#020
009D 82                 MOV MASK,A      ;SET INTERRUPT MASK

009E 3456               CALL MEMCLR     ;CLEAR VIDEO MEMORY

00A0 05                 EN XI      ;REENABLE EXTERNAL INTERRUPTS
00A1 2400               JMP LINNUM      ;DISPLAY TITLE INFORMATION

00A3 86A3       KEYIN:  JNFO KEYIN      ;WAIT FOR KEYBOARD INPUT

                .FORM

                ;DATA ACQUISITION ROUTINES

00A5 85                 CLR FO     ;CLEAR START BIT
00A6 2300               MOV A,#VOFF     ;VIDEO OFF
00A8 45                 MOV VCR,A
00A9 3456               CALL MEMCLR     ;CLEAR VIDEO MEMORY
00AB 27                 CLR A
00AC C2                 MOV HACC,A
00AD 0E                 MOV SROW,A      ;RESET SROW TO BEGINNING
00AE BA9C               MOV R2, #156    ;SET LOOP COUNTER FOR # WORDS TO READ
00B0 2305               MOV A,#5
00B2 88                 MOVL RO,A       ;SET MEMORY POINTER TO FIRST DATA DEPOSIT
00B3 23C0               MOV A,#OCO
00B5 C2                 MOV HACC,A
00B6 89                 MOVL R1,A       ;LOAD R1 WITH ADDRESS OF SWITCH REGISTER
00B7 BB0D               MOV R3,#13      ;LOAD R3 WITH POINTER BUMP CONSTANT
00B9 8C                 MOV A,INTR      ;CLEAR OUT ANY PENDING INTERRUPTS
00BA 25                 EN II      ;REENABLE LP INTERRUPT

00BB 66BB       DATAIN: JNF1 DATAIN     ;WAIT FOR "THROUGH" BIT TO SET

                ;DISASSEMBLE DATA INTO DISPLAY FORMAT
00BD A5                 CLR F1     ;RESET "THROUGH"
00BE BA9C               MOV R2,#156     ;DISASSEMBLE DATA, LOAD WORD COUNTER
00C0 27                 CLR A
00C1 C2                 MOV HACC,A
00C2 2305               MOV A,#5
00C4 88                 MOVL RO,A       ;SET MEMORY POINTER TO FIRST DATA DEPOSIT
00C5 BB08               MOV R3,#8       ;DO 8 BITS
00C7 90         UNASS:  MOVX A,@RO       ;LOAD IN DATA BYTE
00C8 04CB               JMP DEPST
00CA FC         RETRIV: MOV A,R4        ;RETRIEVE CHARACTER
00CB F7         DEPST:  RLC A      ;ROTATE BIT INTO CARRY
00CC AC                 MOV R4,A        ;TEMPORARY SAVE
00CD F6D3               JC ONE     ;STORE A "1"
00CF 2330               MOV A,#'O'      ;STORE A "O"
00D1 04D5               JMP CONT
```

TL/DD/6970-8

```
OOD3 2331    ONE:    MOV A,#'1'       ;STORE A "1"
OOD5 80      CONT:   MOVX @RO,A       ;STORE CHARACTER
OOD6 38              INCL RO  ;INCREMENT POINTER
OOD7 EBCA            DJNZ R3,RETRIV  ;CONTINUE IF WORD NOT DONE

OOD9 98              MOVL A,RO        ;BUMP MEMORY POINTER TO NEXT WORD
OODA 0305            ADD A,#5
OODC 88              MOVL RO,A
OODD BBO8            MOV R3,#8        ;RESET BIT COUNTER
OODF EAC7            DJNZ R2,UNASS    ;CONTINUE IF ALL LOCATIONS NOT DONE
OOE1 2400            JMP LINNUM       ;JUMP TO NEXT PAGE
             .FORM
             ;LINE NUMBERING ROUTINES
     0100    . = 0100

0100 27      LINNUM: CLR A
0101 C2              MOV HACC,A
0102 88              MOVL RO,A        ;CLEAR MEMORY POINTER
0103 BA9C            MOV R2,#156      ;DO 156 LINES
0105 BBOA            MOV R3,#10       ;SET HUNDREDS POINTER
0107 BCOA            MOV R4,#10       ;SET TENS POINTER
0109 BDOA            MOV R5,#10       ;SET ONES POINTER

010B FB      NUMLP:  MOV A,R3         ;LOOK UP HUNDREDS ASCII CODE
010C 343B            CALL LKUP
010E 80              MOVX @RO,A       ;STORE HUNDREDS ASCII CODE
010F 38              INCL RO
0110 FC              MOV A,R4         ;LOOK UP TENS ASCII CODE
0111 343B            CALL LKUP
0113 80              MOVX @RO,A       ;STORE TENS ASCII CODE
0114 38              INCL RO
0115 FD              MOV A,R5         ;LOOK UP ONES ASCII CODE
0116 343B            CALL LKUP
0118 80              MOVX @RO,A       ;STORE ONES ASCII CODE

0119 98              MOVL A,RO        ;BUMP RO TO NEXT LINE
011A 030B            ADD A,#11
011C 88              MOVL RO,A

011D ED26            DJNZ R5,CONNUM  ;INCREMENT ONES POINTER
011F BDOA            MOV R5,#10       ;MUST NOW INCREMENT TENS
0121 EC26            DJNZ R4,CONNUM  ;INCREMENT TENS POINTER
0123 BCOA            MOV R4,#10       ;MUST NOW INCREMENT HUNDREDS
0125 CB              DEC R3

0126 EAOB    CONNUM: DJNZ R2,NUMLP    ;DO ANOTHER ROW

0128 9A              MOV A,HOME       ;PUT UP TITLE LINE
0129 88              MOVL RO,A
012A BAOD            MOV R2,#13       ;LOOKUP 13 CHARACTERS
012C BB49            MOV R3,#L(TITLE)         ;LOAD POINTER TO ASCII TITLE STRING

012E FB      TITLP:  MOV A,R3
012F B3              MOVP A,@A
0130 80              MOVX @RO,A
0131 38              INCL RO
0132 1B              INC R3
0133 EA2E            DJNZ R2,TITLP

             .FORM

0135 2320            MOV A,#VON       ;TURN VIDEO BACK ON
0137 45              MOV VCR,A
0138 05              EN XI      ;REENABLE KEYBOARD INTERRUPTS
0139 04A3            JMP KEYIN        ;RETURN AND WAIT FOR NEXT START

             ;SUBROUTINES

013B 033E    LKUP:   ADD A,#L(CHAR)-1;
013D B3              MOVP A,@A        ;LOOK UP ASCII NUMBER
013E 93              RETR       ;RETURN
```

TL/DD/6970-9

```

    013F 39       CHAR:   .BYTE '9876543210'

    0149 53       TITLE:  .BYTE 'STEP 7 DATA 0'

    0156 27       MEMCLR: CLR A   ;VIDEO MEMORY CLEAR LOOP
    0157 C2              MOV HACC,A      ;ACCUMULATOR CLEAR
    0158 88              MOVL R0,A       ;R0 CLEAR
    0159 BA00            MOV R2,#0       ;INNER LOOP COUNTER SET
    015B BB08            MOV R3, #8      ;OUTER LOOP COUNTER SET
    015D 2320            MOV A,#020      ;SPACE CHARACTER
    015F 80       MCLRLP: MOVX @R0,A      ;STORE A CHARACTER
    0160 38              INCL R0  ;INCREMENT POINTER
    0161 EA5F            DJNZ R2,MCLRLP  ;TEST INNER LOOP
    0163 BA00            MOV R2,#0       ;RELOAD INNER LOOP COUNTER
    0165 EB5F            DJNZ R3, MCLRLP ;TEST OUTER LOOP
    0167 93              RETR        ;THROUGH, RETURN


                  .END
BEGIN    0052     CHAR     013F     CKOFF    0024     CONNUM   0126
CONT     00D5     DATAIN   00BB     DEPST    00CB     DNTRU    004D
DOWN     003C     EXI      0003 *   INI      0007 *   KEY      0012
KEYIN    00A3     LINNUM   0100     LKUP     013B     LSTLIN   07DF
MCLRLP   015F     MEMCLR   0156     MEMEND   07EB     NOTRU    0011
NUMLP    010B     ONE      00D3     RESET    0000 *   RETRIV   00CA
START    0027     STLIN    07EC     TITLE    0149     TITLP    012E
UNASS    00C7     UP       002B     UPTRU    0039     VOFF     0000
VON      0020

NO ERROR LINES

  328  ROM BYTES USED

SOURCE CHECKSUM = 40D8
OBJECT CHECKSUM = 0649

INPUT   FILE A:LOGIC.MAC
LISTING FILE A:LOGIC.PRN
OBJECT  FILE A:LOGIC.LM
```

TL/DD/6970–10

Building an Inexpensive but Powerful Color Terminal

National Semiconductor
Application Note 374
Leigh Cropper

Historically, the design of a color CRT terminal has involved a significant upgrade of the circuit for a monochrome terminal. The result was a stiff increase in price for the electronics as well as for the monitor when going from monochrome to color. As a result, most companies built monochrome terminals and a few built color terminals only.

On the personal computer front where separate monitors are common, manufacturers have started to offer video cards which will support either a monochrome or a color monitor. More recently, color terminals have begun to appear which are extensions of monochrome terminal families. They require a board full of I.C.s for even the most space efficient designs. But now, using the TMP, you can do the same job with just one VLSI chip and a half dozen 7400 family TTL chips.

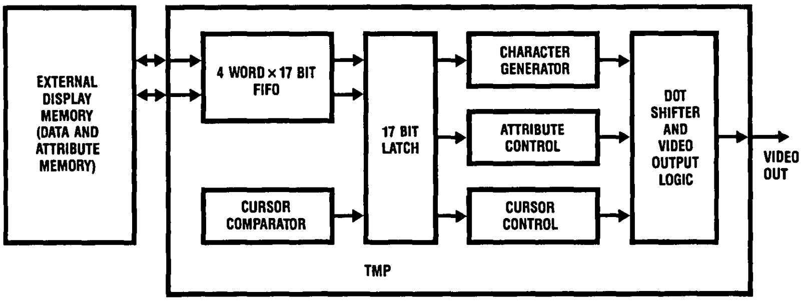
The National Semiconductor NS405 Series Terminal Management Processor (TMP) was originally conceived as a monochrome "terminal on a chip". However, the design team took special pains to build in "hooks" to allow users to augment the basic features of the TMP. In particular the TMP supports almost unlimited attribute expansion, and therein lies the key to adding color to a TMP-based terminal. Even nicer, the addition of color attributes does not sacrifice any of the other powerful features provided by the TMP.

Here, we will delve into a little of the mechanics of TMP attribute handling. The diagram in *Figure 1* shows the path of the attribute bits (normally 8) from the display memory into the TMP, through the FIFO, the attribute control logic, and finally to the video output section where the attributes are combined with the serialized video output.

Because the display memory space may be large (up to 64k x 16), it is easy to store many more attribute bits by adding display memory chips. A 2k x 8 RAM will hold 8 attribute bits for every location on an 80 row by 25 line display. However, in order to implement color attributes, three problems must be examined: (1) how to let both the CPU and the display controller address the extra attribute memory in a practical manner; (2) how to imitate the behavior of the internal FIFO and maintain proper synchronization; and (3) how to combine the color attributes and the video output signal.

Before addressing the three problems in detail, a discussion of the number and type of color attributes is in order. The simplest type of color display would require only 3 bits (red, green, and blue). That allows a character to be displayed in any of 7 colors over a black background or, when reverse video is asserted, the character is black on a colored background. For independent control of both the foreground and background colors, 6 bits are required. To get more shades of color, add more bits.

A practical approach employs a 2k x 8 RAM for the color attribute memory. Three of the bits control the foreground color, three control the background color and the remaining two may be used to adjust intensity (1 for foreground and 1 for background).

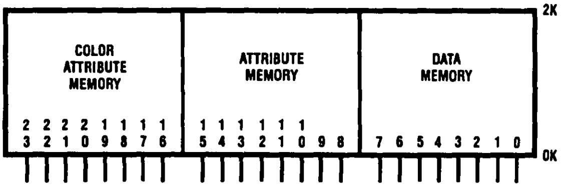


TL/DD/7923-1

FIGURE 1. TMP Attribute Processing

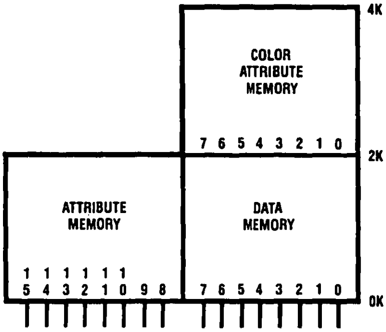
Now let's tackle the problems one by one.

1. COLOR ATTRIBUTE MEMORY ADDRESSING. When fetching data for the display, we need to get 24 bits in parallel (8 data, 8 attribute and 8 color attribute). But when the CPU accesses memory, it can handle only 16 bits at a time, so the CPU must be able to read and write color attributes in a different bank of memory from that where the data and ordinary attributes are stored. For an 80 character by 25 row display the memory could be mapped as shown in *Figures 2* and *3:*



TL/DD/7923-2

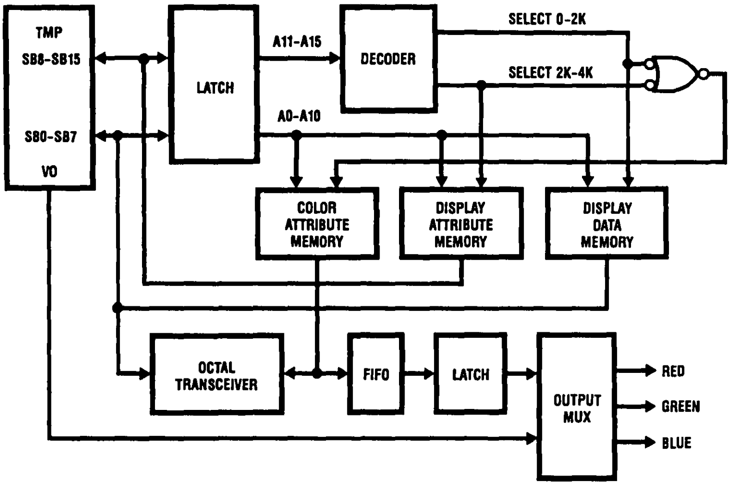
FIGURE 2. Memory Map as Selected for Screen Refresh



TL/DD/7923-3

FIGURE 3. Memory Map as Selected for CPU Access

The mapping is implemented by the following circuit:



TL/DD/7923-4

FIGURE 4. Color Attribute Memory Mapping Circuit

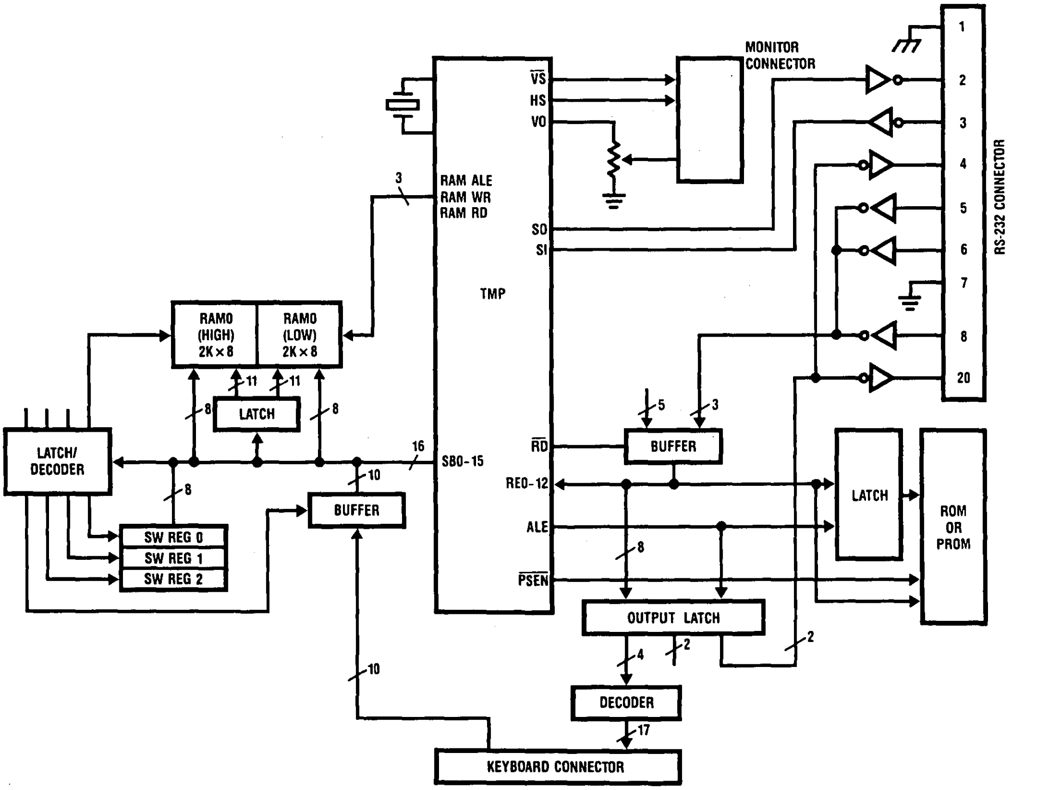
During a display refresh cycle the color attribute memory is selected by the low bank select (the same select signal that enables the data and attribute memories). However, the color attribute bits drive the external FIFO's, whereas the output from the other two memories is routed through the TMP. The data path from the color attribute memory to the TMP is buffered by an octal transceiver which is disabled when the low bank is selected. During a CPU access to color attribute memory, the high bank select enables the color attribute memory and the octal transceiver. The direction control of the transceiver is controlled by the RAM RD signal from the TMP.

2. EXTERNAL FIFO SYNCHRONIZATION. The TMP provides FI CLK (FIFO Input Clock) and FO CLK (FIFO Output clock) signals which may be used to clock an external FIFO. The FI CLK signal is identical in timing and duty cycle to the RAM RD signal except that FI CLK is disabled (stays high) when the CPU accesses display memory. When the 74LS224 is used as an external FIFO, FI CLK must be inverted. The rising edge of FO CLK occurs when output of the internal FIFO is loaded into the internal dot shifter. The FO CLK is used to empty a word from the external FIFO and clock it into an octal latch.
3. COMBINING COLOR ATTRIBUTES WITH VIDEO. When using foreground and background color attributes, a 74LS157 multiplexer works nicely to switch between the two. The eight color attribute bits from the latch are separated into groups of four. The video output signal is used to switch the multiplexer. When the video output is high, foreground attributes are selected: and when the video output is low, background attributes are selected. The outputs of the multiplexer (red, green, blue and intensity) directly drive the color monitor inputs. A minor problem arises because the video output from the TMP already includes the blanking signal. That makes it impossible to differentiate between a series of spaces in the middle of the screen and the horizontal blanking interval. In either case, the video output is low. The easiest solution is handled in software. Let's assume that we want an 80 column display and are using three 2K x 8 memory chips for the data, attribute and color memories. We set up the TMP for 81 columns and then configure the program so that the 81st column always contains a space code with all attribute bits off (including color). That way the background color will always be black during both horizontal and vertical retrace. The cost is 25 locations in each of the memories, but we can afford that many because an 80 x 25 display requires 2000 locations, leaving 48 free.

A Practical Example

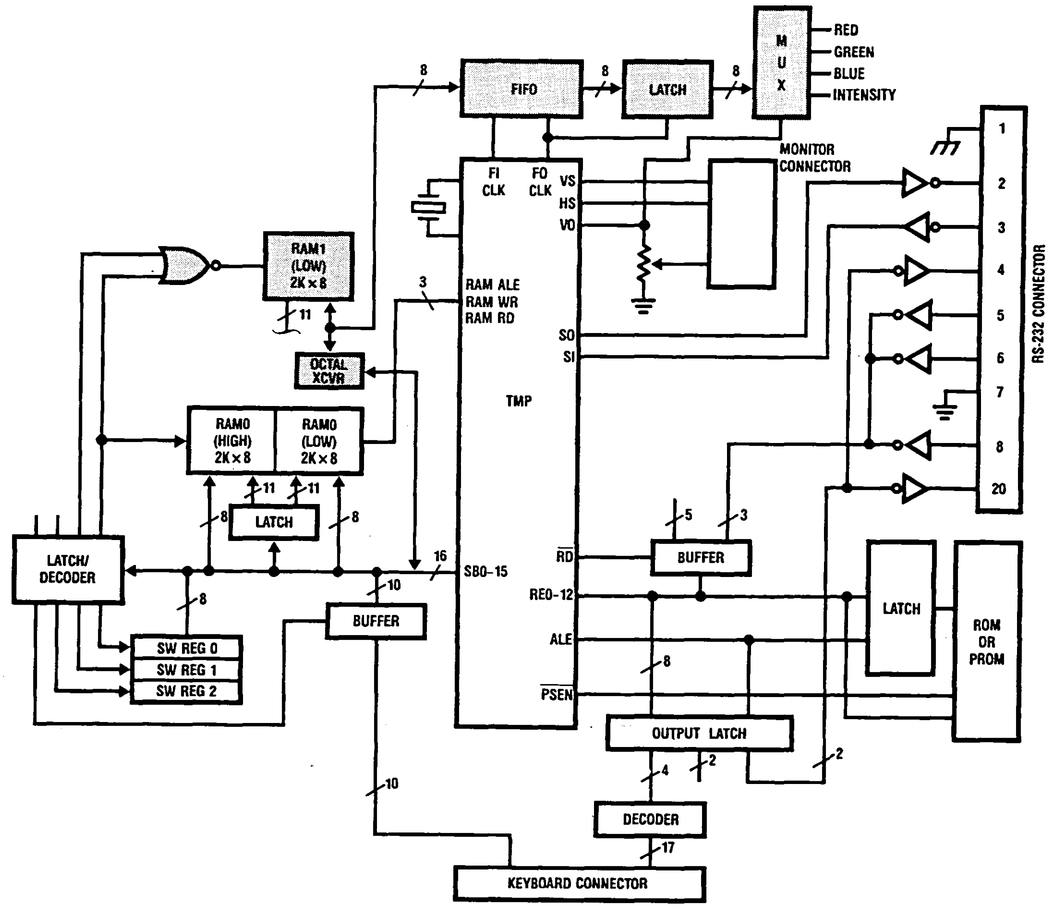
Here we will present a color terminal circuit with the associated program as an example of what you may want to do. We started with the terminal design of the TMP development board. See the block diagram in *Figure 5*.

The block diagram of the color terminal (with the old portions of the original monochrome terminal unshaded and the new color circuits shaded) appears in *Figure 6*. The new circuitry was added in the prototyping area of the development board.



TL/DD/7923-5

FIGURE 5. TMP Development Board Block Diagram



TL/DD/7923–6

FIGURE 6. TMP Development Board Color Circuitry Block Diagram

COLOR ATTRIBUTE BIT ASSIGNMENTS

The bit assignments are:

Bit 0	—	Blue foreground
Bit 1	—	Green foreground
Bit 2	—	Red foreground
Bit 3	—	Blue background
Bit 4	—	Green background
Bit 5	—	Red background
Bit 6	—	Foreground intensity
Bit 7	—	Background intensity

Without using the intensity control bits you get 8 foreground colors: red, green, blue, magenta, cyan, yellow, white, and black (beam off). The same 8 colors may be independently selected for the background. There are several RGB monitors available in the moderate price range with sufficient bandwidth to work with a 12 MHz TMP. Some of them include a separate intensity (or luminence) input. Others include internal decoding circuitry which provides the ability to handle 4 bits of color input and provide as many as 16 different colors.

The demonstration program which runs on the development board allows limited color support. The Escape, V sequence from the keyboard or the receiver prompts the program to treat the next character received as an eight bit color attribute byte with the bit assignments as listed above. That byte is written to the color attribute memory as each succeeding character is received, until another escape, V sequence is encountered. The table which follows includes the foreground and background color combinations for characters which can be entered from the keyboard, but it ignores the effect of the 2 high-order bits (foreground and background intensity).

COLOR COMBINATIONS FOR RGB MONITORS

Table I gives the Foreground/Background color combinations that occur when using the '<ESC> Vv' Escape sequence. To set the current color attribute, all that you need to do is select the color combination from the Table below, and send it to the NS405 as part of the <ESC> Vx sequence. For example, '<ESC> V"' causes the Foreground color to be green and the Background color to be red . . . not all that pleasing, to my tastes, but choose what you will.

TABLE I
Foreground/Background Color Combinations

Char	Fore/Back	Char	Fore/Back	Char	Fore/Back
sp	Black/Red	6	Yellow/Yellow	K	Cyan/Blue
!	Blue/Red	7	White/Yellow	L	Red/Blue
"	Green/Red	8	Black/White	M	Magenta/Blue
#	Cyan/Red	9	Blue/White	N	Yellow/Blue
$	Red/Red	:	Green/White	O	White/Blue
%	Magenta/Red	;	Cyan/White	P	Black/Green
&	Yellow/Red	<	Red/White	Q	Blue/Green
'	White/Red	=	Magenta/White	R	Green/Green
(	Black/Magenta	>	Yellow/White	S	Cyan/Green
)	Blue/Magenta	?	White/White	T	Red/Green
*	Green/Magenta	@	Black/Black	U	Magenta/Green
+	Cyan/Magenta	A	Blue/Black	V	Yellow/Green
,	Red/Magenta	B	Green/Black	W	White/Green
-	Magenta/Magenta	C	Cyan/Black	X	Black/Cyan
.	Yellow/Magenta	D	Red/Black	Y	Blue/Cyan
/	White/Magenta	E	Magenta/Black	Z	Green/Cyan
0	Black/Yellow	F	Yellow/Black	[	Cyan/Cyan
1	Blue/Yellow	G	White/Blue	\	Red/Cyan
2	Green/Yellow	H	Black/Blue	]	Magenta/Cyan
3	Cyan/Yellow	I	Blue/Blue	^	Yellow/Cyan
4	Red/Yellow	J	Green/Blue	-	White/Cyan
5	Magenta/Yellow				

TMP Extended Program Memory Application Note

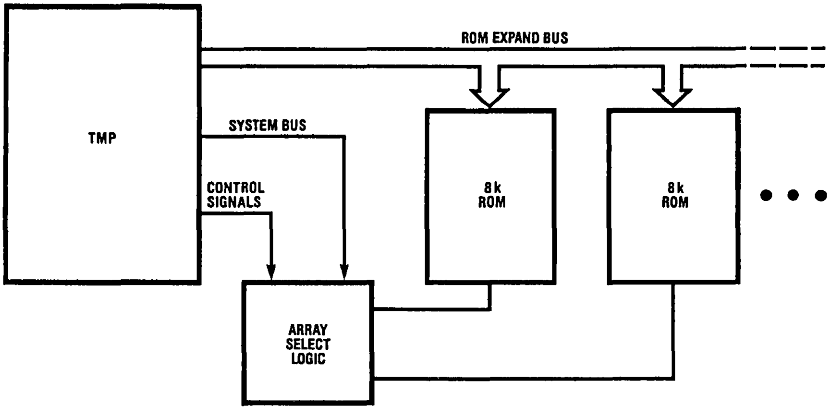
National Semiconductor
Application Note 399
Richard Lazovick

OVERVIEW/INTRODUCTION

The purpose of this application note is to describe methods for expanding the program memory of the NS405 series TERMINAL MANAGEMENT PROCESSOR (TMP) and to provide direction in software techniques for utilizing the expanded memory efficiently. The chip has a built-in capability of addressing up to 8k of external program memory (ROM), via the ROM Expand Bus, and 64k of video display memory (RAM), via the System Bus. Although 8k of program memory is sufficient for most applications there are many applications, such as emulating multiple terminals or using many look-up tables, that require still more memory. However, it is very rare that the entire 64k of video RAM is used since that is more than enough memory to store two screens of data in the pixel mode or thirty-two screens of data in the alphanumeric/block graphics mode. Therefore it is practical to use a video memory address to switch between two or more 8k memory arrays.

The idea behind using a bank select switch to change from one memory array to another is not new, nor is it difficult, and when implemented properly it can be a very useful tool. The TMP has all the necessary control signals to make both the software and hardware straight-forward.

Block Diagram



TL/DD/8430–1

SOFTWARE

For purposes of demonstration it will be easier to look at the software aspects of using an array select switch first, then designing the hardware to implement it.

The easiest case occurs if we use less than 16k of display memory. Then we have two system bus address lines available to select either of our two arrays. To switch arrays all we have to do is read from (or write to) an address that uses the address line you wish to toggle. It is safer to read from the address since we do not want to change data in memory at the location addressed by the lower order address lines.

Suppose we choose SB14 to select the low order array and SB15 to select the high order array. The program steps we would go through to switch from the low array to the high array could be:

```
MOV   A,#080        ;Load HACC w/ 80H to set SB15 HI
MOV   HACC,A        ;and set SB14 LO. We do not care
MOVL  RO,A          ;about the other address bits.
MOVX  A,@RO         ;SB15 goes HI.
```

In general we will want to switch arrays several times, and we will want to be able to conveniently control the destination address in the new array.

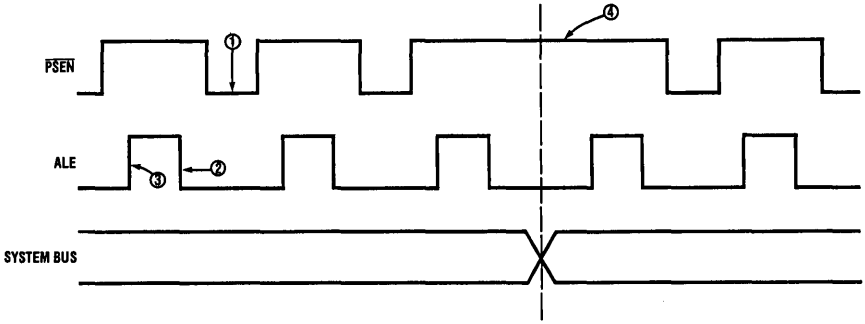
Since it is very cumbersome to rewrite the whole sequence everytime, let's mimic the internal select memory bank command (SEL MBx) by using a subroutine and a CALL followed by a JMP to conveniently control our array switching.

```
        CALL  SELHA         ;Select HI order array.
        JMP   HERE          ;Jump to HERE in new array.

SELHA:  MOV   A,#080        ;Load HACC w/ 80H to set SB15 HI
        MOV   HACC,A        ;and set SB14 LO. We do not care
        MOVL  RO,A          ;about the other address bits.
        MOVX  A,@RO         ;SB15 goes HI.
        RET                 ;Return to execute the jump.
```

Now each time we switch to the high order array all we have to do is execute a CALL and a JMP.

System Signals Timing Diagram



TL/DD/8430-2

Note 1: Enable ROM output drivers.

Note 2: ROM address available.

Note 3: RE bus addresses changes during rising edge of ALE and are stable by falling edge.

Note 4: No $\overline{\text{PSEN}}$ signal present during last cycle of MOVX instruction, however $\overline{\text{PSEN}}$ is active during both RET cycles.

HARDWARE

Now that we have made the software simple and straightforward we have to look at what hardware is necessary to implement it.

We want to: 1) create two mutually exclusive enable signals—one for each array,

2) be able to easily use and latch the address line signals, and

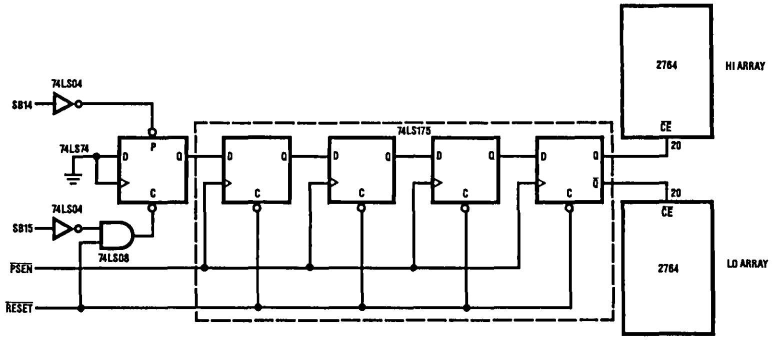
3) delay the actual switching of arrays until after the jump instruction, with the new address, is read into the TMP from the old array.

Looking at the program we see that the system bus changes after the MOVX instruction with the RET and JMP instructions still to be read in from memory before we actually want to switch arrays. Each instruction takes two cycles, therefore we want to delay our array switching signal by four cycles. Looking at all the output signals on the TMP there are two possible signals to use as a clock to delay the array switching signal. These signals are $\overline{\text{PSEN}}$ and ALE (see System Signals Timing Diagram).

The main disadvantage of using ALE is that whereas we want a rising edge to clock the flip-flops used for the delay, the ROM addresses are not stable until the falling edge of ALE. Therefore, we save one inverter by using $\overline{\text{PSEN}}$.

One possible circuit implementation is shown below:

Circuit Diagram



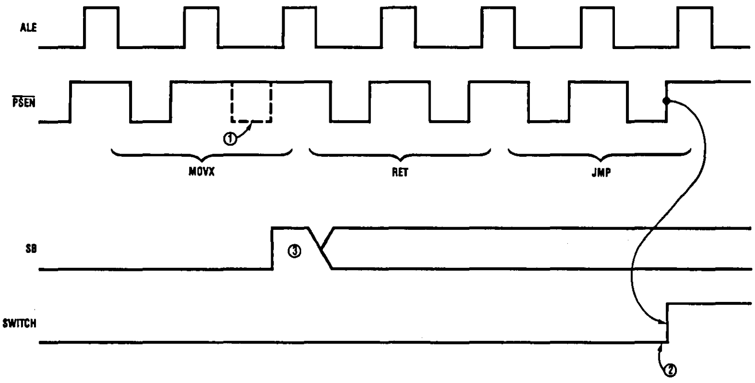
TL/DD/8430-3

The first flip-flop latches one of the two system bus signals and the next four delay the array switching signal by four $\overline{\text{PSEN}}$ cycles. The two inverters are there so that we trigger off a ONE on the address. If the system bus was configured as a 16 bit address/data bus (bit 4 of SCR set) then the latched address lines would have to be used. Since it is always desirable to have the flip-flops in a known state at power up, some sort of reset circuitry should be used (e.g., by tying the power up reset circuit to the clear inputs on the flip-flops), or both arrays should have identical reset sequences that include setting the flip-flops to a known state.

LOOKING IN DEPTH

Now that we have the basic software format we are going to use and the hardware to implement it, let's look at what is actually happening (see Array Switching Timing Diagram 1). The system bus line switches after the first cycle of the MOVX instruction, but there is no $\overline{\text{PSEN}}$ during the second cycle. Then there are two $\overline{\text{PSEN}}$ signals during the RET instruction and two $\overline{\text{PSEN}}$ signals during the JMP instruction. Just after the second byte of the JMP is read into the TMP, the arrays are switched, the PC gets loaded with the new address, and the program continues execution as normal in the new array from the address indicated in the JMP instruction. Be sure you understand the Array Switching Timing Diagram 1 before continuing.

Array Switching Timing Diagram 1



TL/DD/8430-4

Note 1: No $\overline{\text{PSEN}}$ signal.

Note 2: Arrays switch here.

Note 3: Valid approximately 360 ns.

ADDENDUMS

Although it can be a problem when trying to execute a call across array boundaries, the problem can be easily overcome as can the confusion that arises when many array switchings occur. All that one needs to do is to organize the program memory efficiently. One such scheme would be to set aside a block of memory in each bank, such as the last page to use for memory mapping. For example if we wanted to jump from location HOME in the LO array to location HERE in the HI array and then back to HOME we could map our memory as shown below:

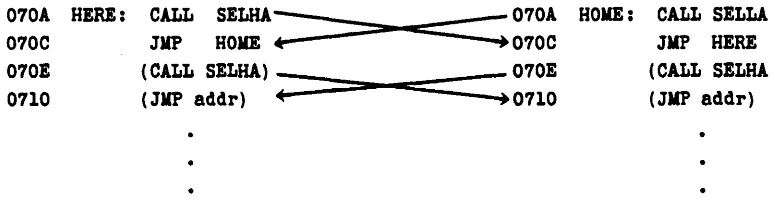
```
                  Lo Array                                  Hi Array

MAIN PROGRAM:
                     •
                     •
                     •                                         •
      0500  HOME:  JMP   HERE                                  •
                     •                                         •
                     •                        0680  HERE:  NOP
                     •                        0681         JMP  HOME
                                                               •
                                                               •
                                                               •

ARRAY SWITCHING SUBROUTINE:

      0700  SELHA: MOV   A,#080               0700  SELLA: MOV  A,#040
      0702         MOV   HACC,A               0702         MOV  HACC,A
      0703         MOVL, R0,A                 0703         MOVL R0,A
      0704         MOVX  A,@R0                0704         MOVX A,R0
      0705         NOP                        0705         NOP
      0706         NOP                        0706         NOP
      0707         RET                        0707         RET
```

MEMORY MAP:



```
      070A  HERE:  CALL  SELHA                070A  HOME:  CALL SELLA
      070C         JMP   HOME                 070C         JMP  HERE
      070E        (CALL SELHA)                070E        (CALL SELHA)
      0710        (JMP addr)                  0710        (JMP addr)
                     •                                         •
                     •                                         •
                     •                                         •
```

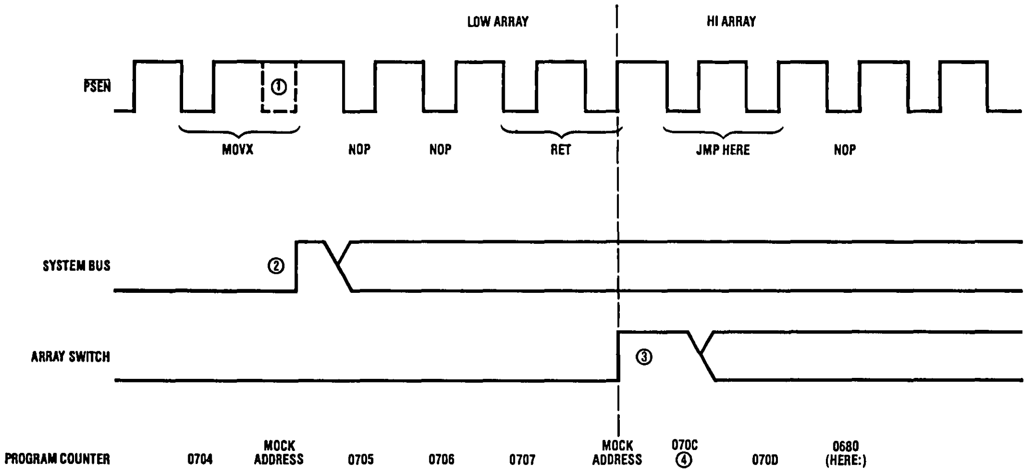
TL/DD/8430-6

Note: The arrows show which JMP corresponds to which subroutine call. For example the JMP HOME at location 070C in the LO ARRAY corresponds to the CALL at location 070A in the HI ARRAY. The ( )'s show how the CALL's and JMP's can be strung together in a neat pattern.

Notice that there are two NOP's in the subroutine. Since the JMP after the CALL was moved to the new array the two NOP's were added to the subroutine so that the actual array switching occurs just after the completion of the RET instruction. The PC is then loaded with the new jump value, loaded in from the new array, and we continue execution as expected. Be sure to understand the timing before going any further (see Array Switching Timing Diagram 2). The way the memory map is set up it is easy to organize and keep track of jumps using the pattern indicated in the example. This method also eliminates any problems with the assembler searching for undefined labels.

ADDENDUMS (Continued)

Array Switching Timing Diagram 2



TL/DD/8430-5

Note 1: Missing $\overline{\text{PSEN}}$ signal.

Note 2: System bus changes.

Note 3: Arrays switch.

Note 4: Now in new array.

Since there are two extra NOP's in the switching array subroutine the hardware can now be simplified and the system speed increased by removing two of the flip-flops from the chain. Through the use of the SEL MBx commands the memory map can be located in any page of any memory bank. For example if we wanted to jump to location HERE in memory bank 2 of the HI array from memory bank 1 of the LO array (after having removed two flip-flops) we could map our memory as shown below:

Lo Array   Hi Array

ARRAY SWITCHING SUBROUTINE:

```
0700  SELHA:  MOV   A,#080            0700  SELLA:  MOV   A,#040
0702          MOV   HACC,A            0702          MOV   HACC,A
0703          MOVL  R0,A              0703          MOVL  R0,A
0704          MOVX  A,@R0             0704          MOVX  A,@R0
0705          RET                     0705          RET
```

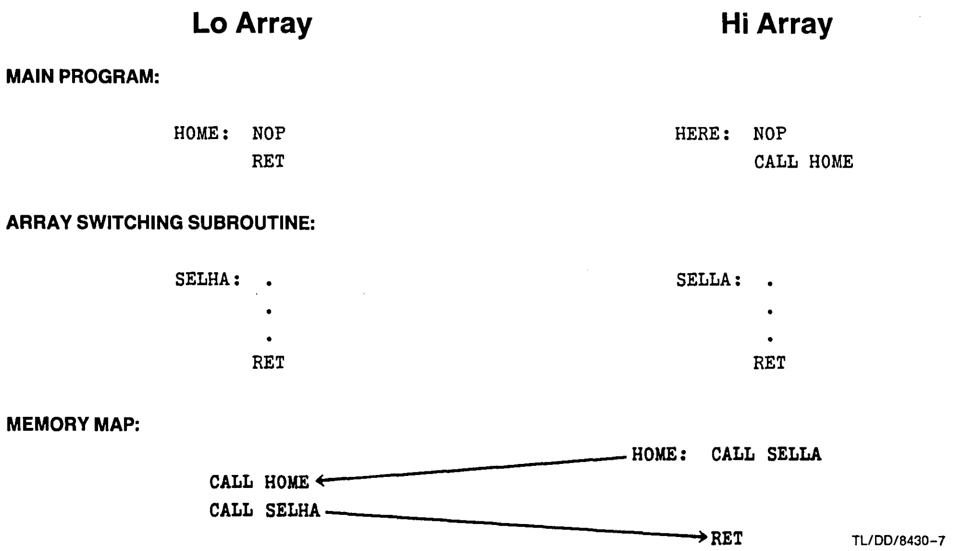
MEMORY MAP:

```
070A  HERE:   CALL  SELHA             070A
070C                                  070C          SEL   MB2
070E                                  070E          JMP   HERE
```

MAIN PROGRAM:

```
0800  HOME:   SEL   MB0
0801          JMP   HERE
                                      1000  HERE:   NOP
```

If a call into the other array is necessary a similar pattern to that above could be used. Start by replacing the JMP's with CALL's to the desired subroutine and appropriately placing returns. For example (here it comes) if we wanted to CALL HOME from HERE we could memory map as shown below.



```
            Lo Array                                    Hi Array
MAIN PROGRAM:

              HOME:  NOP                          HERE:  NOP
                     RET                                 CALL HOME

ARRAY SWITCHING SUBROUTINE:

              SELHA:  .                           SELLA:  .
                      .                                   .
                      .                                   .
                     RET                                 RET

MEMORY MAP:
                                          HOME:   CALL SELLA
                 CALL HOME
                 CALL SELHA
                                                  RET
```

TL/DD/8430-7

Since calling between different memory banks is not straight forward it is advisable to be very careful when doing it, or to limit calls between arrays only to those that reside in the same memory bank.

HELPFUL HINTS

These schemes can all be modified to multiple arrays and easier or fancier mappings, however there are a few things to keep in mind.

1) If using a system bus address line to toggle the array, don't use that line as part of an actual display memory address.
2) The MOVX instruction can require more than two cycles depending on system bus contention, however we are only concerned with the last two cycles and the $\overline{\text{PSEN}}$ signals that occur after the system bus line changes.
3) If using interrupts—disable them while switching arrays and keep all time critical routines in the same array.
4) A demux or decoder can be used to select memory arrays or decode address lines when more than two 8k arrays are implemented or more than 16k of video RAM is being used.
5) If extra memory is needed, but a good deal of the program memory is data storage, the data could be stored in the video memory space instead of implementing a new array.
6) If the TMP is going to be used in a noisy environment or the system bus is configured as a 16 bit bidirectional bus a synchronous latch should be used to assure stable levels on SB14 and SB15.
7) The given array switching circuit can be implemented with the demo board by wiring it into an extension board that can be plugged into the prom socket U9. Wire the two new proms in parallel with each other and with a cable that can plug directly into the prom socket. However, instead of using pin 20 from the demo board, use the two array enable lines as the chip enables for the 2764 proms.

   Also use SB12 and SB13 instead of SB14 and SB15.