Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Not For New Designs
Core Processor	CPU16
Core Size	16-Bit
Speed	16MHz
Connectivity	EBI/EMI, SCI, SPI
Peripherals	POR, PWM, WDT
Number of I/O	16
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	132-BQFP Bumpered
Supplier Device Package	132-PQFP (24.13x24.13)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68hc16z1ceh16

LIST OF TABLES
(Continued)

Table 4-2 Instruction Set Summary (Continued)

Mnemonic	Operation	Description	Address	Instruction			Condition Codes							
			Mode	Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
LDAB	Load B	(M) ⇒ B	IND8, X	C5	ff	6	—	—	—	—	Δ	Δ	0	Δ
			IND8, Y	D5	ff	6								
			IND8, Z	E5	ff	6								
			IMM8	F5	ii	2								
			IND16, X	17C5	gggg	6								
			IND16, Y	17D5	gggg	6								
			IND16, Z	17E5	gggg	6								
			EXT	17F5	hh ll	6								
			E, X	27C5	—	6								
			E, Y	27D5	—	6								
			E, Z	27E5	—	6								
LDD	Load D	(M : M + 1) ⇒ D	IND8, X	85	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	95	ff	6								
			IND8, Z	A5	ff	6								
			IMM16	37B5	jj kk	4								
			IND16, X	37C5	gggg	6								
			IND16, Y	37D5	gggg	6								
			IND16, Z	37E5	gggg	6								
			EXT	37F5	hh ll	6								
			E, X	2785	—	6								
			E, Y	2795	—	6								
			E, Z	27A5	—	6								
LDE	Load E	(M : M + 1) ⇒ E	IMM16	3735	jj kk	4	—	—	—	—	Δ	Δ	0	—
			IND16, X	3745	gggg	6								
			IND16, Y	3755	gggg	6								
			IND16, Z	3765	gggg	6								
			EXT	3775	hh ll	6								
LDED	Load Concatenated E and D	(M : M + 1) ⇒ E (M + 2 : M + 3) ⇒ D	EXT	2771	hh ll	8	—	—	—	—	—	—	—	—
LDHI	Initialize H and I	$(M : M + 1)_X$ ⇒ H R $(M : M + 1)_Y$ ⇒ I R	INH	27B0	—	8	—	—	—	—	—	—	—	—
LDS	Load SP	(M : M + 1) ⇒ SP	IND8, X	CF	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DF	ff	6								
			IND8, Z	EF	ff	6								
			IND16, X	17CF	gggg	6								
			IND16, Y	17DF	gggg	6								
			IND16, Z	17EF	gggg	6								
			EXT	17FF	hh ll	6								
			IMM16	37BF	jj kk	4								
LDX	Load IX	(M : M + 1) ⇒ IX	IND8, X	CC	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DC	ff	6								
			IND8, Z	EC	ff	6								
			IMM16	37BC	jj kk	4								
			IND16, X	17CC	gggg	6								
			IND16, Y	17DC	gggg	6								
			IND16, Z	17EC	gggg	6								
			EXT	17FC	hh ll	6								
LDY	Load IY	(M : M + 1) ⇒ IY	IND8, X	CD	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DD	ff	6								
			IND8, Z	ED	ff	6								
			IMM16	37BD	jj kk	4								
			IND16, X	17CD	gggg	6								
			IND16, Y	17DD	gggg	6								
			IND16, Z	17ED	gggg	6								
			EXT	17FD	hh ll	6								
LDZ	Load IZ	(M : M + 1) ⇒ IZ	IND8, X	CE	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	DE	ff	6								
			IND8, Z	EE	ff	6								
			IMM16	37BE	jj kk	4								
			IND16, X	17CE	gggg	6								
			IND16, Y	17DE	gggg	6								
			IND16, Z	17EE	gggg	6								
			EXT	17FE	hh ll	6								

NOTE

The external bus interface does not latch data when an external bus cycle is terminated by a bus error. When this occurs during an instruction prefetch, the IMB precharge state (bus pulled high, or $FF) is latched into the CPU16 instruction register, with indeterminate results.

5.6.5.2 Double Bus Faults

Exception processing for bus error exceptions follows the standard exception processing sequence. Refer to **4.13 Exceptions** for more information. However, two special cases of bus error, called double bus faults, can abort exception processing.

$\overline{\text{BERR}}$ assertion is not detected until an instruction is complete. The $\overline{\text{BERR}}$ latch is cleared by the first instruction of the $\overline{\text{BERR}}$ exception handler. Double bus fault occurs in two ways:

1. When bus error exception processing begins, and a second $\overline{\text{BERR}}$ is detected before the first instruction of the exception handler is executed.
2. When one or more bus errors occur before the first instruction after a $\overline{\text{RESET}}$ exception is executed.

Multiple bus errors within a single instruction that can generate multiple bus cycles cause a single bus error exception after the instruction has been executed.

Immediately after assertion of a second $\overline{\text{BERR}}$, the MCU halts and drives the $\overline{\text{HALT}}$ line low. Only a reset can restart a halted MCU. However, bus arbitration can still occur. Refer to **5.6.6 External Bus Arbitration** for more information. A bus error or address error that occurs after exception processing has been completed (during the execution of the exception handler routine, or later) does not cause a double bus fault. The MCU continues to retry the same bus cycle as long as the external hardware requests it.

5.6.5.3 Halt Operation

When $\overline{\text{HALT}}$ is asserted while $\overline{\text{BERR}}$ is not asserted, the MCU halts external bus activity after negation of $\overline{\text{DSACK}}$. The MCU may complete the current word transfer in progress. For a long-word to byte transfer, this could be after S2 or S4. For a word to byte transfer, activity ceases after S2.

Negating and reasserting $\overline{\text{HALT}}$ according to timing requirements provides single-step (bus cycle to bus cycle) operation. The $\overline{\text{HALT}}$ signal affects external bus cycles only, so that a program that does not use external bus can continue executing. During dynamically-sized 8-bit transfers, external bus activity may not stop at the next cycle boundary. Occurrence of a bus error while $\overline{\text{HALT}}$ is asserted causes the CPU16 to process a bus error exception.

When the MCU completes a bus cycle while the $\overline{\text{HALT}}$ signal is asserted, the data bus goes into a high-impedance state and the $\overline{\text{AS}}$ and $\overline{\text{DS}}$ signals are driven to their inactive states. Address, function code, size, and read/write signals remain in the same state.

The SIM clock synthesizer provides clock signals to the other MCU modules. After the clock is running and MSTRST is asserted for at least four clock cycles, these modules reset. V_{DD} ramp time and VCO frequency ramp time determine how long the four cycles take. Worst case is approximately 15 milliseconds. During this period, module port pins may be in an indeterminate state. While input-only pins can be put in a known state by external pull-up resistors, external logic on input/output or output-only pins during this time must condition the lines. Active drivers require high-impedance buffers or isolation resistors to prevent conflict.

Figure 5-20 is a timing diagram for power-on reset. It shows the relationships between $\overline{\text{RESET}}$, V_{DD}, and bus signals.

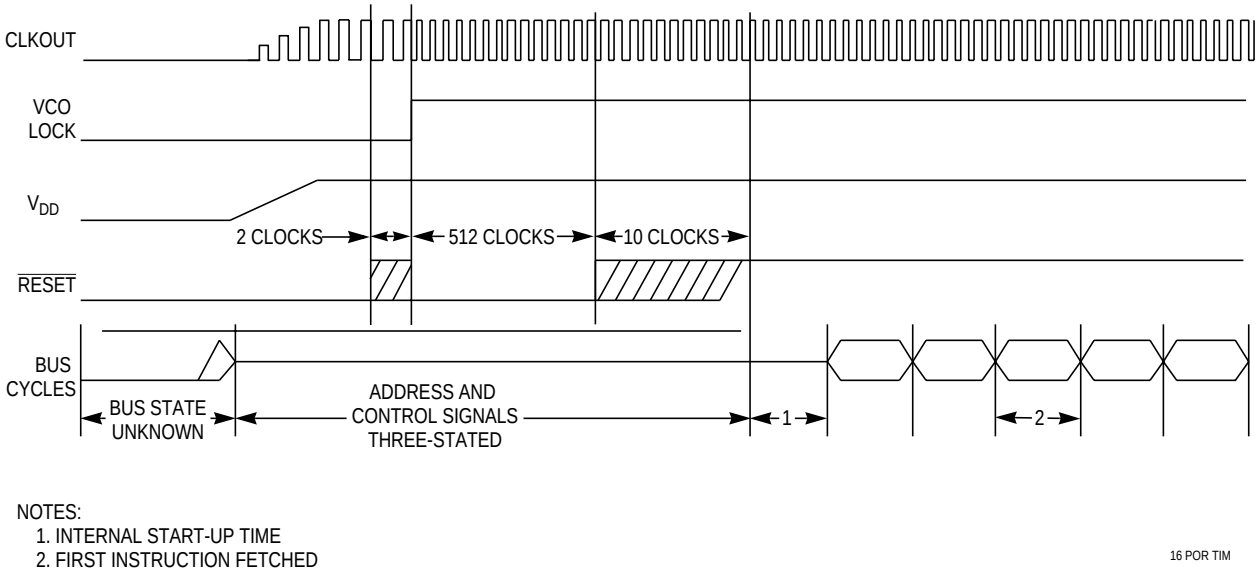


Figure 5-20 Power-On Reset

5.7.8 Use of the Three-State Control Pin

Asserting the three-state control (TSC) input causes the MCU to put all output drivers in a disabled, high-impedance state. The signal must remain asserted for approximately ten clock cycles in order for drivers to change state.

When the internal clock synthesizer is used (MODCLK held high during reset), synthesizer ramp-up time affects how long the ten cycles take. Worst case is approximately 20 milliseconds from TSC assertion.

When an external clock signal is applied (MODCLK held low during reset), pins go to high-impedance state as soon after TSC assertion as approximately ten clock pulses have been applied to the EXTAL pin.

To ensure that the MCCI stops in a known state, assert the STOP bit before executing the CPU LPSTOP instruction. Before asserting the STOP bit, disable the SPI (clear the SPE bit) and disable the SCI receivers and transmitters (clear the RE and TE bits). Complete transfers in progress before disabling the SPI and SCI interfaces.

Once the STOP bit is asserted, it can be cleared by system software or by reset.

10.2.1.2 Privilege Levels

The supervisor bit (SUPV) in the MMCR has no effect since the CPU16 operates only in the supervisor mode.

10.2.1.3 MCCI Interrupts

The interrupt request level of each of the three MCCI interfaces can be programmed to a value of zero (interrupts disabled) through seven (highest priority). These levels are selected by the ILSCIA and ILSCIB fields in the SCI interrupt level register (ILSCI) and the ILSPI field in the SPI interrupt level register (ILSPI). In case two or more MCCI submodules request an interrupt simultaneously and are assigned the same interrupt request level, the SPI submodule is given the highest priority and SCIB is given the lowest.

When an interrupt is requested which is at a higher level than the interrupt mask in the CPU status register, the CPU initiates an interrupt acknowledge cycle. During this cycle, the MCCI compares its interrupt request level to the level recognized by the CPU. If a match occurs, arbitration with other modules begins.

Interrupting modules present their arbitration number on the IMB, and the module with the highest number wins. The arbitration number for the MCCI is programmed into the interrupt arbitration (IARB) field of the MMCR. Each module should be assigned a unique arbitration number. The reset value of the IARB field is $0, which prevents the MCCI from arbitrating during an interrupt acknowledge cycle. The IARB field should be initialized by system software to a value from $F (highest priority) through $1 (lowest priority). Otherwise, the CPU identifies any interrupts generated as spurious and takes a spurious-interrupt exception.

If the MCCI wins the arbitration, it generates an interrupt vector that uniquely identifies the interrupting serial interface. The six MSBs are read from the interrupt vector (INTV) field in the MCCI interrupt vector register (MIVR). The two LSBs are assigned by the MCCI according to the interrupting serial interface, as indicated in **Table 10-1**.

Table 10-1 MCCI Interrupt Vectors

Interface	INTV[1:0]
SCIA	00
SCIB	01
SPI	10

10.5 MCCI Initialization

After reset, the MCCI remains in an idle state. Several registers must be initialized before serial operations begin. A general sequence guide for initialization follows.

A. Global
   1. Configure MMCR
      a. Write an interrupt arbitration number greater than zero into the IARB field.
      b. Clear the STOP bit if it is not already cleared.
   2. Interrupt vector and interrupt level registers (MIVR, ILSPI, and ILSCI)
      a. Write the SPI/SCI interrupt vector into MIVR.
      b. Write the SPI interrupt request level into the ILSPI and the interrupt request levels for the two SCI interfaces into the ILSCI.
   3. Port data register
      a. Write a data word to PORTMC.
      b. Read a port pin state from PORTMCP.
   4. Pin control registers
      a. Establish the direction of MCCI pins by writing to the MDDR.
      b. Assign pin functions by writing to the MPAR.

B. Serial Peripheral Interface
   1. Configure SPCR
      a. Write a transfer rate value into the BAUD field.
      b. Determine clock phase (CPHA) and clock polarity (CPOL).
      c. Specify an 8- or 16-bit transfer (SIZE) and MSB- or LSB-first transfer mode (LSBF).
      d. Select master or slave operating mode (MSTR).
      e. Enable or disable wired-OR operation (WOMP).
      f. Enable or disable SPI interrupts (SPIE).
      g. Enable the SPI by setting the SPE bit.

C. Serial Communication Interface (SCIA/SCIB)
   1. To transmit, read the SCSR, and then write transmit data to the SCDR. This clears the TDRE and TC indicators in the SCSR.
      a. SCI control register 0 (SCCR0)
      b. Write a baud rate value into the BR field.
   2. Configure SCCR1
      a. Select 8- or 9-bit frame format (M).
      b. Determine use (PE) and type (PT) of parity generation or detection.
      c. To receive, set the RE and RIE bits in SCCR1. Select use (RWU) and type (WAKE) of receiver wakeup. Select idle-line detection type (ILT) and enable or disable idle-line interrupt (ILIE).
      d. To transmit, set TE and TIE bits in SCCR1, and enable or disable WOMC and TCIE bits. Disable break transmission (SBK) for normal operation.

Table A-7 Low Voltage Clock Control Timing

(V_{DD} and V_{DDSYN} = 2.7 to 3.6 Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H)

Num	Characteristic	Symbol	Min	Max	Unit
1	PLL Reference Frequency Range[1] MC68CM16Z1	f_{ref}	 3.2	 4.2	 MHz
2	PLL Reference Frequency Range[1] MC68CK16Z1 MC68CK16Z4	f_{ref}	 20 20	 50 50	 kHz kHz
3	System Frequency[2] On-Chip PLL System Frequency Slow On-Chip PLL System Frequency Fast On-Chip PLL System Frequency External Clock Operation	f_{sys}	dc 4 (f_{ref}) 4 (f_{ref}) /128 dc	16.78 16.78 16.78 16.78	MHz
4	PLL Lock Time[1, 7, 8, 9] Changing W or Y in SYNCR or exiting from LPSTOP[3] Warm Start-Up[4] Cold Start-Up (fast reference option only)[5]	t_{lpll}	—	 20 50 75	ms
5	VCO Frequency[6]	f_{VCO}	—	2 (f_{sys} max)	ms
6	Limp Mode Clock Frequency SYNCR X bit = 0 SYNCR X bit = 1	f_{limp}	 — —	 f_{sys} max /2 f_{sys} max	 MHz
7	CLKOUT Jitter[1, 7, 8, 9, 10] Short term (5 µs interval) Long term (500 µs interval)	J_{clk}	 –0.5 –0.05	 0.5 0.05	 %

NOTES:
1. Refer to notes in **Table A-10**.

Table A-11 Low Voltage 16.78-MHz DC Characteristics

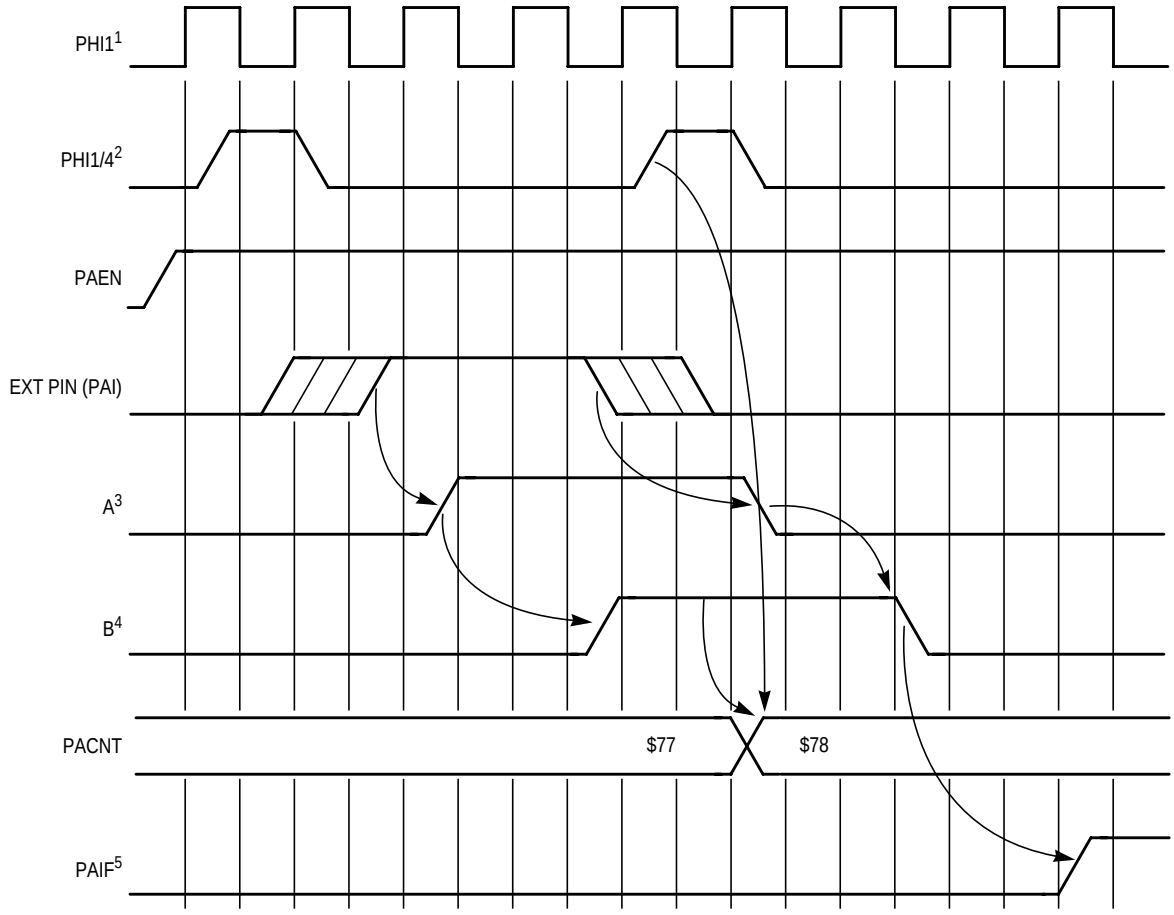
(V_{DD} and V_{DDSYN} = 2.7 to 3.6Vdc, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H)

Num	Characteristic	Symbol	Min	Max	Unit
1	Input High Voltage	V_{IH}	0.7 (V_{DD})	V_{DD} + 0.3	V
2	Input Low Voltage	V_{IL}	V_{SS} – 0.3	0.2 (V_{DD})	V
3	Input Hysteresis[1]	V_{HYS}	0.5	—	V
4	Input Leakage Current [2] V_{in} = V_{DD} or V_{SS} Input-only pins	I_{in}	–2.5	2.5	µA
5	High Impedance (Off-State) Leakage Current[2] V_{in} = V_{DD} or V_{SS} All input/output and output pins	I_{OZ}	–2.5	2.5	µA
6	CMOS Output High Voltage[2, 3] I_{OH} = –10.0 µA Group 1, 2, 4 input/output and output pins	V_{OH}	V_{DD} –0.2	—	V
7	CMOS Output Low Voltage[2] I_{OL} = 10.0 µA Group 1, 2, 4 input/output and output pins	V_{OL}	—	0.2	V
8	Output High Voltage[2, 3] I_{OH} = –0.4 mA Group 1, 2, 4 input/output and output pins	V_{OH}	V_{DD} –0.5	—	V
9	Output Low Voltage[2] I_{OL} = 0.8 mA Group 1 I/O pins, CLKOUT, FREEZE/QUOT, IPIPE0 I_{OL} = 2.6 mA Group 2 and group 4 I/O pins, $\overline{CSBOOT}$, $\overline{BG}/\overline{CS}$ I_{OL} = 6 mA Group 3	V_{OL}	— — —	0.4 0.4 0.4	V
10	Three State Control Input High Voltage	V_{IHTSC}	7.2	9.1	V
11	Data Bus Mode Select Pull-up Current [4] V_{in} = V_{IL} V_{in} = V_{IH}	I_{MSP}	— –8	–95 —	µA
12	V_{DD} Supply Current[5] Run[6] LPSTOP, 4.194 MHz crystal, VCO Off (STSIM = 0)[7] LPSTOP, 32.768 kHz crystal, VCO Off (STSIM = 0)[7] LPSTOP, external clock input frequency = max f_{sys} WAIT[8]	I_{DD} S_{IDD} S_{IDD} S_{IDD} W_{IDD}	 — — — —	50 2 260 3.0 23	mA mA µA mA mA
13	Clock Synthesizer Operating Voltage	V_{DDSYN}	2.7	3.6	V
14	MC68CM16Z1V_{DDSYN} Supply Current[4] VCO on, crystal reference, maximum f_{sys}[7] External clock, maximum f_{sys} LPSTOP, 4.194 MHz crystal reference, VCO off (STSIM = 0)[7] V_{DD} powered down	I_{DDSYN}	— — — —	2 2.5 2 2	mA mA mA mA
14A	MC68CK16Z1/Z4 V_{DDSYN} Supply Current[4] VCO on, crystal reference, maximum f_{sys}[7] External clock, maximum f_{sys} LPSTOP, 32.768 kHz crystal reference, VCO off (STSIM = 0)[7] 32.768 kHz, V_{DD} powered down	I_{DDSYN}	— — — —	655 2.5 150 70	µA mA µA µA
15	RAM Standby Voltage[9] Specified V_{DD} applied V_{DD} = V_{SS}	V_{SB}	0.0 2.7	V_{DD} 3.6	V
16	MC68CK16Z1/Z4 RAM Standby Current[4, 9, 10] Normal RAM operation $V_{DD} > V_{SB} – 0.5$ V Transient condition $V_{SB} – 0.5$ V ≥ V_{DD} ≥ V_{SS} + 0.5 V Standby operation $V_{DD} < V_{SS}$ + 0.5 V	I_{SB}	— — —	10 3 50	µA mA µA

Table A-16 16.78-MHz AC Timing

(V_{DD} and V_{DDSYN} = 5.0 Vdc ± 10 %, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H)[1]

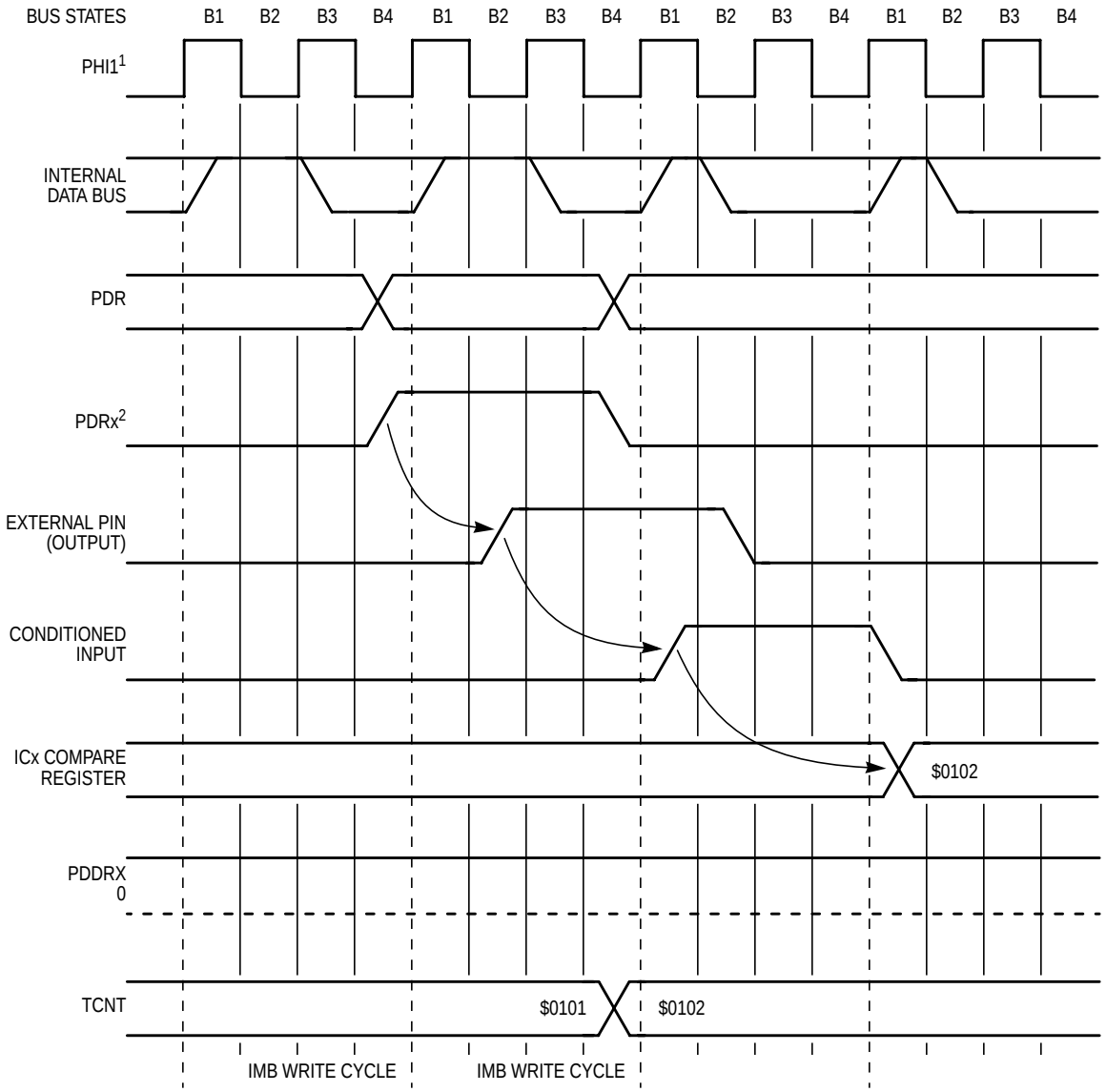
Num	Characteristic	Symbol	Min	Max	Unit
F1	Frequency of Operation	f	—	16.78	MHz
1	Clock Period	t_{cyc}	59.6	—	ns
1A	ECLK Period	t_{Ecyc}	476	—	ns
1B	External Clock Input Period[2]	t_{Xcyc}	59.6	—	ns
2, 3	Clock Pulse Width[3]	t_{CW}	24	—	ns
2A, 3A	ECLK Pulse Width	t_{ECW}	236	—	ns
2B, 3B	External Clock Input High/Low Time[2]	t_{XCHL}	29.8	—	ns
4, 5	CLKOUT Rise and Fall Time	t_{Crf}	—	5	ns
4A, 5A	Rise and Fall Time (All Outputs except CLKOUT)	t_{rf}	—	8	ns
4B, 5B	External Clock Input Rise and Fall Time[3]	t_{XCrf}	—	5	ns
6	Clock High to ADDR, FC, SIZE Valid[4]	t_{CHAV}	0	29	ns
7	Clock High to ADDR, Data, FC, SIZE, High Impedance	t_{CHAZx}	0	59	ns
8	Clock High to ADDR, FC, SIZE, Invalid	t_{CHAZn}	0	—	ns
9	Clock Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Asserted[4]	t_{CLSA}	2	24	ns
9A	$\overline{AS}$ to $\overline{DS}$ or $\overline{CS}$ Asserted (Read)[5]	t_{STSA}	-15	15	ns
11	ADDR, FC, SIZE Valid to $\overline{AS}$, $\overline{CS}$, (and $\overline{DS}$ Read) Asserted	t_{AVSA}	15	—	ns
12	Clock Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated	t_{CLSN}	2	29	ns
13	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to ADDR, FC SIZE Invalid (Address Hold)	t_{SNAI}	15	—	ns
14	$\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Width Asserted	t_{SWA}	100	—	ns
14A	$\overline{DS}$, $\overline{CS}$ Width Asserted (Write)	t_{SWAW}	45	—	ns
14B	$\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Width Asserted (Fast Cycle)	t_{SWDW}	40	—	ns
15	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Width Negated[6]	t_{SN}	40	—	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ High Impedance	t_{CHSZ}	—	59	ns
17	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to R/$\overline{W}$ High	t_{SNRN}	15	—	ns
18	Clock High to R/$\overline{W}$ High	t_{CHRH}	0	29	ns
20	Clock High to R/$\overline{W}$ Low	t_{CHRL}	0	29	ns
21	R/$\overline{W}$ High to $\overline{AS}$, $\overline{CS}$ Asserted	t_{RAAA}	15	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$, $\overline{CS}$ Asserted (Write)	t_{RASA}	70	—	ns
23	Clock High to Data Out Valid	t_{CHDO}	—	29	ns
24	Data Out Valid to Negating Edge of $\overline{AS}$, $\overline{CS}$ (Fast Write Cycle)	t_{DVASN}	15	—	ns
25	$\overline{DS}$, $\overline{CS}$ Negated to Data Out Invalid (Data Out Hold)	t_{SNDOI}	15	—	ns
26	Data Out Valid to $\overline{DS}$, $\overline{CS}$ Asserted (Write)	t_{DVSA}	15	—	ns
27	Data In Valid to Clock Low (Data Setup)[4]	t_{DICL}	5	—	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$ Asserted to Clock Low (Setup Time)	t_{BELCL}	20	—	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACK}$[1:0], $\overline{BERR}$, $\overline{HALT}$, $\overline{AVEC}$ Negated	t_{SNDN}	0	80	ns
29	$\overline{DS}$, $\overline{CS}$ Negated to Data In Invalid (Data In Hold)[7]	t_{SNDI}	0	—	ns
29A	$\overline{DS}$, $\overline{CS}$ Negated to Data In High Impedance[7, 8]	t_{SHDI}	—	55	ns



NOTES:
1. PHI1 HAS THE SAME FREQUENCY AS THE SYSTEM CLOCK; HOWEVER, IT DOES NOT HAVE THE SAME TIMING.
2. PHI1/4 CLOCKS PACNT WHEN GT-PAIF IS ASSERTED.
3. A = PAI SIGNAL AFTER THE SYNCHRONIZER.
4. B = "A" AFTER THE DIGITAL FILTER.
5. PAIF IS ASSERTED WHEN PAI IS NEGATED.

PULSE ACCUM GATED MODE

Figure A-26 Pulse Accumulator — Gated Mode (Count While Pin High)



NOTES:
1. PHI1 IS THE SAME FREQUENCY AS THE SYSTEM CLOCK; HOWEVER, IT DOES NOT HAVE THE SAME TIMING.
2. WHEN THE BIT VALUE IS DRIVEN ON THE PIN, THE INPUT CIRCUIT SEES THE SIGNAL. AFTER IT IS CONDITIONED, IT CAUSES THE CONTENTS OF THE TCNT TO BE LATCHED INTO THE ICx COMPARE REGISTER.

GENERAL PURPOSE OUTPUT

Figure A-32 General-Purpose Output (Causes Input Capture)

Table B-1 M68HC16 Z-Series Ordering Information (Continued)

(Shaded cells indicate preliminary part numbers)

Device	Crystal Input	Operating Voltage	Package Type	Temperature	Frequency (MHz)	Package Order Quantity	Order Number
MC68HC16Z4	32 kHz	5 V	144-Pin TQFP	–40 to +85°C	16 MHz	2	SPMCK16Z4CPV16
						60	MCK68HC16Z4CPV16
						300	MCK16Z4CPV16B1
					20 MHz	2	SPMCK16Z4CPV20
						60	MCK68HC16Z4CPV20
						300	MCK16Z4CPV20B1
					25 MHz	2	SPMCK16Z4CPV25
						60	MCK68HC16Z4CPV25
						300	MCK16Z4CPV25B1
				–40 to +105°C	16 MHz	2	SPMCK16Z4VPV16
						60	MCK68HC16Z4VPV16
						300	MCK16Z4VPV16B1
					20 MHz	2	SPMCK16Z4VPV20
						60	MCK68HC16Z4VPV20
						300	MCK16Z4VPV20B1
					25 MHz	2	SPMCK16Z4VPV25
						60	MCK68HC16Z4VPV25
						300	MCK16Z4VPV25B1
				–40 to +125°C	16 MHz	2	SPMCK16Z4MPV16
						60	MCK68HC16Z4MPV16
						300	MCK16Z4MPV16B1
					20 MHz	2	SPMCK16Z4MPV20
						60	MCK68HC16Z4MPV20
						300	MCK16Z4MPV20B1
		2.7 V	132-Pin PQFP	–40 to +85°C	16 MHz	2	SPMCCK16Z4CFC16
						36	MC68CK16Z4CFC16
						180	MCCK16Z4CFC16B1
			144-Pin TQFP	–40 to +85°C	16 MHz	2	SPMCCK16Z4CPV16
						60	MC68CK16Z4CPV16
						300	MCCK16Z4CPV16B1

D.4.4 ROM Bootstrap Words

ROMBS0 — ROM Bootstrap Word 0 **$YFF830**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT USED				ZK[3:0]				SK[3:0]				PK[3:0]			

ROMBS1 — ROM Bootstrap Word 1 **$YFF832**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC[15:0]															

ROMBS2 — ROM Bootstrap Word 2 **$YFF834**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SP[15:0]															

ROMBS3 — ROM Bootstrap Word 3 **$YFF836**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IZ[15:0]															

Typically, CPU16 reset vectors reside in non-volatile memory and are fetched when the CPU16 comes out of reset. These four words can be used as reset vectors with the contents specified at mask time. The content of these words cannot be changed. On generic (blank ROM) MC68HC16Z2 and MC68HC16Z3 devices, ROMBS[0:3] are masked to $0000. When the ROM on the MC68HC16Z2 and MC68HC16Z3 is masked with customer specific code, ROMBS[0:3] respond to system addresses $00000 to $00006 during the reset vector fetch if $\overline{\text{BOOT}}$ = 0.

```
        JSR     SEND_CH         ;go send out the byte
        AIX     #$01            ;increment IX to point to the next byte
        BRA     SEND_STRING     ;loop back and do next byte in string
STRING_DONE:
        JSR     DELAY           ;wait for a moment
        RTS                     ;go back to whence we came

SEND_CH:                        ;subroutine to send out one byte to SCI
        LDAA    SCSR            ;read SCI status reg to check/clear TDRE bit
        ANDA    #$01            ;check only the TDRE flag bit
        BEQ     SEND_CH         ;if TDR is not empty, go back to check it
                                ;again
        LDAA    #$00            ;clear A to send a full word to SCDR ($FFC0E)
        STD     SCDR            ;transmit one ASCII character to the screen
LOOP5:  LDAB    SCSR+1
        ANDB    #$80            ;test the TC bit (transfer complete)
        BEQ     LOOP5           ;continue to wait until TC is set
        RTS                     ;return to send_string subroutine

*****  The STRINGs  *****

STRING  DC      'The System Clock is now running at 16.777 MHz...',0a,0d,00
SEC_STR DC      'check this out!',0a,0d,00,00
STRING2 DC      'The System Clock is now running at 4.194 MHz...',0a,0d,00


*****  Interrupts/Exceptions  *****

BDM:    BGND                    ;exception vectors point here
                                ;and put the user into background debug mode
```

E.2.1.4 Example 4 - Software Watchdog, Periodic Interrupt, and Autovector Demo

```
*       Description : This program demonstrates the software watchdog,
*                     the periodic interrupt, and an autovector.
*                     The periodic interrupt runs a clock which is updated
*                     on the dummy terminal.  Every eight seconds the COP
*                     will force a reset unless IRQ6 is grounded.  When IRQ6
*                     is pulled low, an autovectored interrupt routine will
*                     “feed” the watchdog with $55 and $AA, and the
*                     clock will run without being reset on the dummy terminal.
*
*************************************************************************

        INCLUDE         'EQUATES.ASM'   ;table of EQUates for common registers
        INCLUDE         'ORG00000.ASM'  ;initialize reset vectors
        INCLUDE         'ORG00008.ASM'  ;initialize interrupt vectors


        ORG     $002C           ;put address of autovector routine
        DC.W      AUTOV         ;at the level 6 vector (IRQ6)

        ORG     $0070           ;put address of periodic interrupt routine
        DC.W      VECRT         ;at 1st user defined interrupt vector
```

E.2.3 QSM/SCI Programming Example

The following programming example involves using a port of the serial communication interface (SCI), one of the serial interfaces of the queued serial module (QSM), to display a message on a dummy terminal.

Refer to **SECTION 9 QUEUED SERIAL MODULE** for more information on the QSM or the SCI.

E.2.3.1 Example 6 - Using an SCI Port

```
*    Description : This program uses the SCI port to display
*                  a shameless message on a dummy terminal.  It includes
*                  a subroutine to print a single character to the SCI
*                  and a subroutine that uses the single character
*                  subroutine to print an entire string.
*
*************************************************************************

        INCLUDE  'EQUATES.ASM'     ;table of EQUates for common register
addresses
        INCLUDE  'ORG00000.ASM'    ;initialize reset vector
        INCLUDE  'ORG00008.ASM'    ;initialize interrupt vectors

         ORG  $0200                ;start program after exception vector table

*****  Initialize  *****

INIT:

        INCLUDE  'INITSYS.ASM'     ;initially set EK=F, XK=0, YK=0, ZK=0
                                   ;set sys clock at 16.78 MHz, disable COP
        INCLUDE  'INITRAM.ASM'     ;turn on internal SRAM at $10000
                                   ;set stack (SK=1, SP=03FE)
        INCLUDE  'INITSCI.ASM'     ;set the SCI baud rate to 9600 baud
                                   ;enable the SCI receiver and transmitter

        LDAB #$00
        TBXK                       ;set XK to bank 0 for STRING access
        LDAB #$01
        TBZK                       ;set ZK to bank 1 for delay counter access
        LDZ  #$0000                ;clear IZ for later use with delay counter

*****  Main Program  *****

MAIN    LDX  #STRING               ;point to the beginning of ASCII string
        JSR  SEND_STRING           ;go output the ASCII string
        BRA  MAIN                  ;branch back to main

*****  Subroutines  *****

SEND_STRING:               ;subroutine to send out the entire ASCII string
        LDAB 0,X           ;get next byte in string as pointed to by IX
        BEQ  STRING_DONE   ;if B=00, then goto delay between messages
        JSR  SEND_CH       ;go send out the byte
```

–N–

–O–

–P–

–Q–