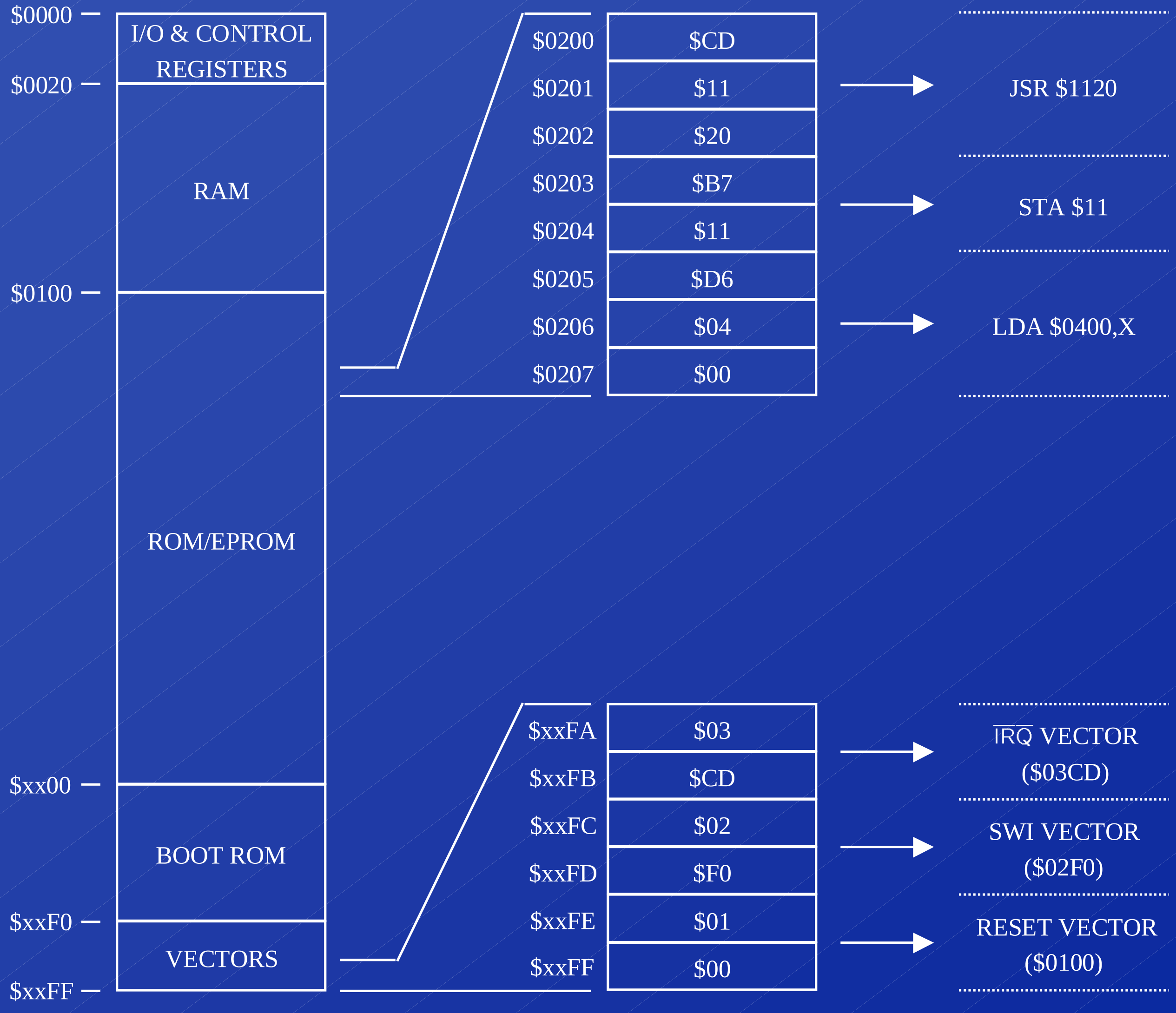
An Introduction to Motorola's 68HC05 Family of 8-Bit Microcontrollers

Table of Contents

- CPU Overview
- Instruction Set
- Addressing Modes
- Sample HC05 Code Example
- Smart Light Dimmer Application Example
- Bicycling Computer Application Example
- Other 68HC05 Family Peripherals

68HC05 Memory Organization

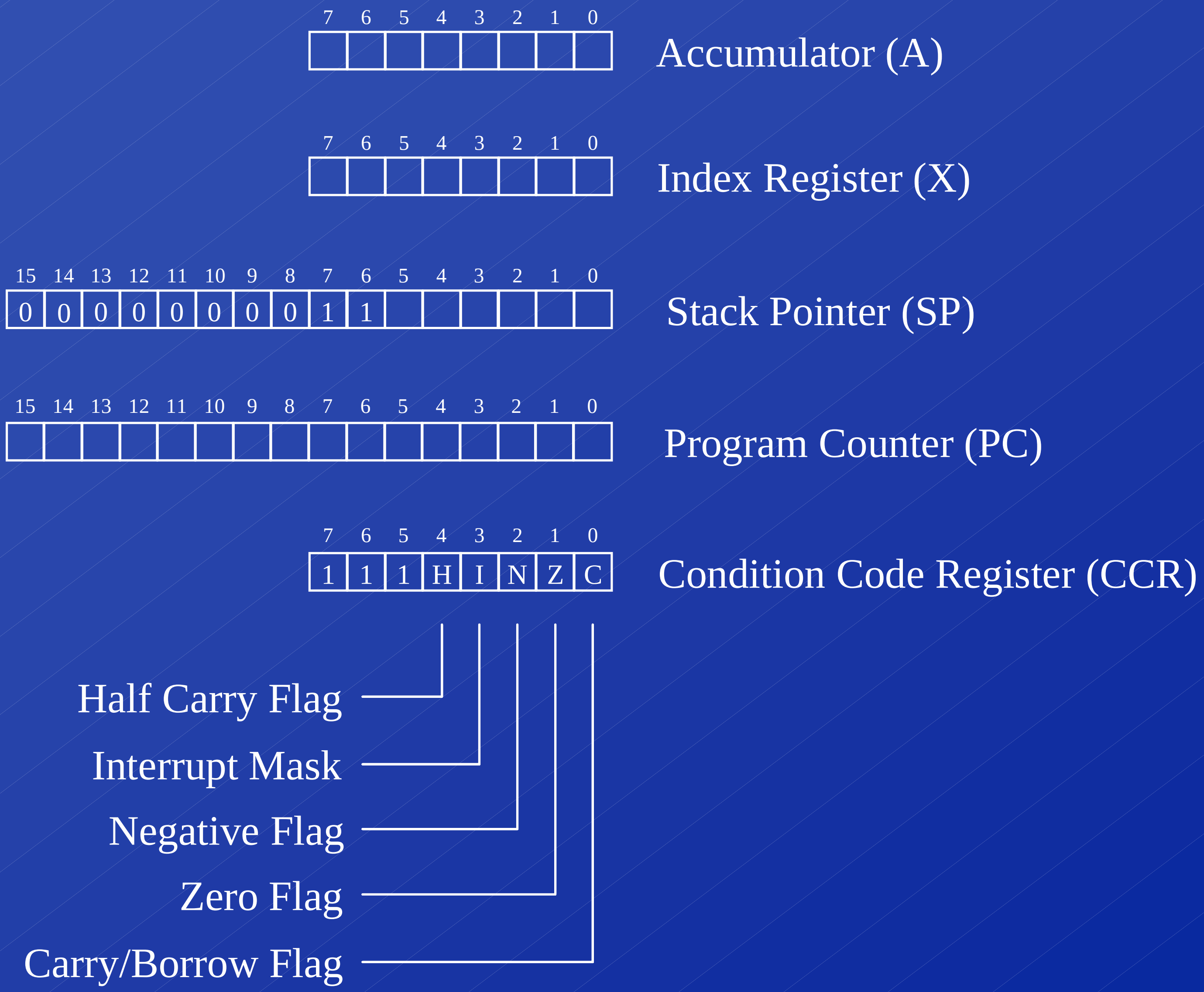


$0000	I/O & CONTROL REGISTERS
$0020	RAM
$0100	ROM/EPROM
$xx00	BOOT ROM
$xxF0	VECTORS
$xxFF	

$0200	$CD	JSR $1120
$0201	$11	
$0202	$20	
$0203	$B7	STA $11
$0204	$11	
$0205	$D6	LDA $0400,X
$0206	$04	
$0207	$00	

$xxFA	$03	$\overline{\text{IRQ}}$ VECTOR ($03CD)
$xxFB	$CD	
$xxFC	$02	SWI VECTOR ($02F0)
$xxFD	$F0	
$xxFE	$01	RESET VECTOR ($0100)
$xxFF	$00	

68HC05 Programmer’s Model



Accumulator (A)

7	6	5	4	3	2	1	0

Index Register (X)

7	6	5	4	3	2	1	0

Stack Pointer (SP)

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

Program Counter (PC)

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

Condition Code Register (CCR)

7	6	5	4	3	2	1	0
1	1	1	H	I	N	Z	C

Half Carry Flag

Interrupt Mask

Negative Flag

Zero Flag

Carry/Borrow Flag

Program Counter Operation

The program counter (PC) increments by one after each byte of an instruction or operand is read. Jumps, branches, returns, and interrupts load the PC with a new value.

Program Counter	Opcode/Operand Read	Instruction
$1000	$B6	LDA $80
$1001	$80	
$1002	$47	ASRA
$1003	$47	ASRA
$1004	$4C	I NCA
$1005	$B7	STA $80
$1006	$80	
$1007	$CD	J SR $13FE
$1008	$13	
$1009	$FE	
$13FE	$4F	CLRA

MOTOROLA

Stack Pointer Operation

before
JSR SUBROUTINE

$00FF	xx	← SP
$00FE	xx	
$00FD	xx	
$00FC	xx	

after
JSR SUBROUTINE

$00FF	Low Byte of Return Address	
$00FE	High Byte of Return Address	
$00FD	xx	← SP
$00FC	xx	

after
RTS

$00FF	Low Byte of Return Address	← SP
$00FE	High Byte of Return Address	
$00FD	xx	
$00FC	xx	

before
INTERRUPT

$00FF	xx	← SP
$00FE	xx	
$00FD	xx	
$00FC	xx	
$00FB	xx	
$00FA	xx	

after
INTERRUPT

$00FF	Low Byte of Return Address	
$00FE	High Byte of Return Address	
$00FD	Index Register	
$00FC	Accumulator	
$00FB	Condition Code Register	
$00FA	xx	← SP

after
RTI

$00FF	Low Byte of Return Address	← SP
$00FE	High Byte of Return Address	
$00FD	Index Register	
$00FC	Accumulator	
$00FB	Condition Code Register	
$00FA	xx	

‘xx’ indicates that contents of memory location are not known

68HC05 Instruction Set — Part 1

Memory Reads & Writes

LDA	load the accumulator
LDX	load the index register
STA	store the accumulator
STX	store the index register

Register Transfers

TAX	transfer the accumulator to the index register
TXA	transfer the index register to the accumulator

Clear Memory & Registers

CLR	clear a memory location
CLRA	clear the accumulator
CLRX	clear the index register

Arithmetic

ADD	add to the accumulator
ADC	add to the accumulator with carry
SUB	subtract from the accumulator
SBC	subtract from the accumulator with borrow
MUL	multiply the accumulator by the index register
NEG	negate (take the 2’s complement of) a memory location
NEGA	negate (take the 2’s complement of) the accumulator
NEGX	negate (take the 2’s complement of) the index register

Decrement & Increment Memory & Registers

`INC`	increment a memory location by one
`INCA`	increment the accumulator by one
`INCX`	increment the index register by one
`DEC`	decrement a memory location by one
`DECA`	decrement the accumulator by one
`DECX`	decrement the index register by one

Boolean Logic

`AND`	logical AND of the accumulator and an operand
`ORA`	inclusive OR of the accumulator and an operand
`EOR`	exclusive OR of the accumulator and an operand
`COM`	take the one’s complement of (invert) a memory location
`COMA`	take the one’s complement of (invert) the accumulator
`COMX`	take the one’s complement of (invert) the index register

Shift Memory & Registers

ASL	arithmetically shift a memory location left by one bit
ASLA	arithmetically shift the accumulator left by one bit
ASLX	arithmetically shift the index register left by one bit
ASR	arithmetically shift a memory location right by one bit
ASRA	arithmetically shift the accumulator right by one bit
ASRX	arithmetically shift the index register right by one bit
LSL	logically shift a memory location left by one bit
LSLA	logically shift the accumulator left by one bit
LSLX	logically shift the index register left by one bit
LSR	logically shift a memory location right by one bit
LSRA	logically shift the accumulator right by one bit
LSRX	logically shift the index register right by one bit

68HC05 Instruction Set — Part 5

Rotate Memory & Registers

ROL	rotate a memory location left by one bit
ROLA	rotate the accumulator left by one bit
ROLX	rotate the index register left by one bit
ROR	rotate a memory location right by one bit
RORA	rotate the accumulator right by one bit
RORX	rotate the index register right by one bit

Test Registers & Memory

BIT	bit test the accumulator and set the N or Z flags
CMP	compare an operand to the accumulator
CPX	compare an operand to the index register
TST	test a memory location and set the N or Z flags
TSTA	test the accumulator and set the N or Z flags
TSTX	test the index register and set the N or Z flags

Branches on Condition Code Register Bits

BCC	branch if carry clear (C = 0)
BCS	branch if carry set (C = 1)
BEQ	branch if equal (Z = 0)
BNE	branch if not equal (Z = 1)
BHCC	branch if half carry clear (H = 0)
BHCS	branch if half carry set (H = 1)
BHI	branch if higher (C or Z = 0)
BHS	branch if higher or same (C = 0)
BLS	branch if lower or same (C or Z = 1)
BLO	branch if lower (C = 1)
BMI	branch if minus (N = 1)
BPL	branch if plus (N = 0)
BMC	branch if interrupts are not masked (I = 0)
BMS	branch if interrupts are masked (I = 1)

MOTOROLA

68HC05 Instruction Set — Part 7

Other Branches

BIH	branch if $\overline{\text{IRQ}}$ pin is high
BIL	branch if $\overline{\text{IRQ}}$ pin is low
BRA	branch always
BRN	branch never
BSR	branch to subroutine and save return address on stack

Single Bit Operations

BCLR	clear the designated memory bit
BSET	set the designated memory bit
BRCLR	branch if the designated memory bit is clear
BRSET	branch if the designated memory bit is set

MOTOROLA

68HC05 Instruction Set — Part 8

Jumps & Returns

JMP	jump to specified address
JSR	jump to subroutine and save return address on stack
RTS	pull address from stack and return from subroutine
RTI	pull registers from stack and return from interrupt

Miscellaneous Control

CLC	clear the condition code register carry bit
SEC	set the condition code register carry bit
CLI	clear the condition code register interrupt mask bit
SEI	set the condition code register interrupt mask bit
SWI	software initiated interrupt
RSP	reset the stack pointer to $00FF
NOP	no operation
WAIT	enable interrupts and halt the CPU
STOP	enable interrupts and stop the oscillator

Addressing Modes — Summary

Several different addressing modes are available to support the data requirements of different 68HC05 instructions.

Inherent	(INH)
Immediate	(IMM)
Extended	(EXT)
Direct	(DIR)
Indexed, 16-Bit Offset	(IX2)
Indexed, 8-Bit Offset	(IX1)
Indexed, No Offset	(IX)
Relative	(REL)
Bit Set and Clear	(BSC)
Bit Test and Branch	(BTB)

Addressing Modes — INH

The operand of an instruction that uses inherent addressing is implied by or *inherent* in the instruction’s opcode.

Some instructions explicitly name registers…

```
ASLA, CLRX, DECA, INCX, ROLA, RORX, RSP, TAX, TXA
```

Others explicitly name condition code register bits…

```
CLC, CLI, SEC, SEI
```

Still others affect one or more unnamed registers…

```
MUL, RTI, RTS, STOP, SWI, WAIT
```

And some have no operands whatsoever…

```
NOP
```

Addressing Modes — IMM

The operand of an instruction that uses immediate addressing *immediately* follows the instruction's opcode in memory.

Immediate addressing is often using with `LDA` and `LDX`…

```
LDA     #$40
LDX     #$80
```

As well as with `ADC`, `ADD`, `SBC`, and `SUB` for arithmetic operations…

```
ADC     #$01
SUB     #$02
```

…`CMP`, `CPX`, and `BIT` for register comparison and testing…

```
BIT     #$C4
CPX     #$FF
```

And with `AND`, `EOR`, and `ORA` for combinatorial logic…

```
AND     #$03
ORA     #$FC
```

Addressing Modes — EXT

Instructions that use extended addressing can read from or write to any location in the 68HC05 memory map.

Extended addressing is often used with LDA, LDX, STA, and STX…

```
LDA     $4000
STX     $0130
```

As well as with ADC, ADD, SBC, and SUB for arithmetic operations…

```
SBC     $01F1
```

…CMP, CPX, and BIT for register comparison and memory testing…

```
CMP     $08C3
```

…with AND, EOR, and ORA for combinatorial logic…

```
EOR     $0325
```

And with JMP and JSR for program flow changes…

```
JMP     $1200
JSR     $3040
```

MOTOROLA

Addressing Modes — DIR

Instructions that use direct addressing can only read from or write to memory locations $00 to $FF.

All read-modify-write instructions support direct addressing…

```
ASL    $00
ASR    $FF
CLR    $02
COM    $FD
DEC    $04
INC    $FB
LSL    $06
LSR    $F9
NEG    $08
ROL    $F7
ROR    $0A
TST    $F5
```

All instructions that support extended addressing also support direct addressing.

Addressing Modes — IX2

When indexed addressing with 16-bit offsets is used, target addresses are calculated by taking the unsigned sum of the contents of the index register and the 16-bit offset.

Example instructions include loads and stores…

```
    LDA    $4000,X
    STX    $03F8,X
```

…arithmetic and combinatorial logic operations…

```
    SBC    $01F1,X
    EOR    $18FF,X
```

…`CMP`, `CPX`, and `BIT` for register comparison and memory testing…

```
    CMP    $08C3,X
```

And `JMP` and `JSR` for program flow changes…

```
    JSR    $0F4C,X
```

The same group of instructions that can use extended addressing is also the only group of instructions that can use indexed addressing with 16-bit offsets.

MOTOROLA

Addressing Modes — IX1

Instructions that use indexed addressing with 8-bit offsets can read from or write to any memory location between $0000 and $01FE inclusive.

All read-modify-write instructions support this addressing mode…

```
ASL    $00,X
ASR    $FF,X
CLR    $02,X
COM    $FD,X
DEC    $04,X
INC    $FB,X
LSL    $06,X
LSR    $F9,X
NEG    $08,X
ROL    $F7,X
ROR    $0A,X
TST    $F5,X
```

Likewise, all instructions that can use direct addressing can also use indexed addressing with 8-bit offsets.

Addressing Modes — IX

The target address for an instruction that uses indexed addressing without an offset is simply the contents of the index register zero extended to 16 bits.

All read-modify-write instructions support this addressing mode…

```
ASL     ,X
ASR     ,X
CLR     ,X
COM     ,X
DEC     ,X
INC     ,X
LSL     ,X
LSR     ,X
NEG     ,X
ROL     ,X
ROR     ,X
TST     ,X
```

All instructions that can use direct addressing and indexed addressing with 8-bit offsets can also use indexed addressing without offsets.

Addressing Modes — REL

Relative addressing is used only by branch instructions to calculate the target address of a change in program flow *relative* to the value of the program counter (PC).

Each branch instruction requires two bytes of storage — one for the branch opcode and one for the signed two’s complement 8-bit relative offset.

This offset is relative to the address of the next instruction, which is the address of the branch instruction plus two.

Consider the following line of code…

```
HERE    BEQ     THERE
```

If the label HERE equates to address $1000 and this is a **FORWARD** branch, the target address can be between $1002 (offset of $00) and $1081 (offset of $7F).

Similarly, if the label HERE equates to address $1000 and this is a **REVERSE** branch, the target address can be between $0F82 (offset of $80) and $1000 (offset of $FE).

Bit Set and Clear — BSC

The bit set and clear (BSC) addressing mode is used only by the BSET and BCLR instructions. Like other read-modify-write instructions, BSET and BCLR take a direct address. There are eight BSET and BCLR opcodes, one for each bit in a byte.

Consider the following line of code…

```
BSET   n, $00
```

In this example, $0 \le n \le 7$ and denotes one of the eight bits in a byte. This assembles to one of the BSET opcodes (calculated at $10 + 2n) and the direct address $00.

BCLR instructions are formed the same way…

```
BCLR   n, $00
```

As above, $0 \le n \le 7$ and denotes one of the eight bits in a byte. This assembles to one of the BCLR opcodes (calculated at $11 + 2n) and the direct address $00.

Bit Test and Branch — BTB

The bit test and branch (BTB) addressing mode is used only by the BRSET and BRCLR instructions. BRSET and BRCLR take a direct address and have eight opcodes to denote each bit in a byte, just like BSET and BCLR.

Consider the following line of code…

```
BRSET  n, $00, TARGET
```

In this example, $0 \leq n \leq 7$ and denotes one of the eight bits in a byte. This assembles to one of the BRSET opcodes (calculated at $00 + 2n), the direct address $00, and an offset to TARGET relative to the address of the instruction that follows BRSET.

BRCLR instructions are formed the same way…

```
BRCLR  n, $00, TARGET
```

As above, $0 \leq n \leq 7$ and denotes one of the eight bits in a byte. This assembles to one of the BRCLR opcodes (calculated at $01 + 2n) , the direct address $00, and an offset to TARGET relative to the address of the instruction that follows BRCLR.

MOTOROLA

A Sample 68HC05 Program

The sample function that follows finds the cosine of an angle between 0 and 180 degrees inclusive by interpolating the result from a look up table.

The table consists of 46 elements representing the cosine of every fourth degree, again, from 0 to 180 degrees inclusive, scaled by 127.

A simple linear interpolation is performed using these standardized equations:

$$\text{Cosine of Given } \theta = \text{Cosine of Known } \theta \text{ - DELTA}$$

$$\text{DELTA} =$$

$$\frac{\text{Cosine of Known Lower } \theta \text{ - Cosine of Known Upper } \theta}{\text{Known Upper } \theta \text{ - Known Lower } \theta} \times (\text{Given } \theta \text{ - Known Lower } \theta)$$

Sample Program Listing — Part 1

```
* The function begins by reading the given angle, THETA, from
* on-chip RAM (using direct mode addressing) and dividing it by
* four.  This is used as an offset into the look up table.

FIND_COSINE    ldx   THETA
               lsrx
               lsrx

* Using indexed addressing with a 16-bit offset, the cosine of
* the known lower angle is loaded into the accumulator, and the
* cosine of the known upper angle is subtracted from it.  This
* difference is then divided by four, which is the difference
* between the known upper angle and the known lower angle.  Save
* this result in the index register to take the delta product.

               lda   COSINE_TABLE,X
               sub   COSINE_TABLE + 1,X
               lsra
               lsra
               tax
```

Sample Program Listing — Part 2

```
* Take the difference between the given angle and the known lower
* angle by logically ANDing the given angle with three.  Now take
* the product of the two DELTA terms.  MUL stores its product MSB
* in the index register and LSB in the accumulator.

               lda   THETA
               and   #$03
               mul

* Because this product is always a small number, it will reside
* only in the accumulator; the index register will be zero.  Once
* again, use the given angle to look up the cosine of the known
* lower angle.  Negating the accumulator and adding the cosine of
* the known lower angle returns the cosine of the given angle.

               ldx   THETA
               lsrx
               lsrx
               nega
               add   COSINE_TABLE ,X
               sta   THETA_COSINE
```

MOTOROLA

Sample Program Listing — Part 3

```
* This is the look up table used for the cosine interpolation
* function.

*                      0,   4,   8,  12,  16,  20,  24,  28,  32
COSINE_TABLE   fcb   $7F, $7E, $7D, $7C, $7A, $77, $74, $70, $6B

*                     36,  40,  44,  48,  52,  56,  60,  64,  68
               fcb   $66, $61, $5B, $54, $4E, $47, $3F, $37, $2F

*                     72,  76,  80,  84,  88,  92,  96, 100, 104
               fcb   $27, $1E, $16, $0D, $04, $FC, $F3, $EA, $E2

*                    108, 112, 116, 120, 124, 128, 132, 136, 140
               fcb   $D9, $D1, $C9, $C1, $B9, $B2, $AC, $A5, $9F

*                    144, 148, 152, 156, 160, 164, 168, 172, 176
               fcb   $9A, $95, $90, $8C, $89, $86, $84, $83, $82

*                    180
               fcb   $81
```

MOTOROLA

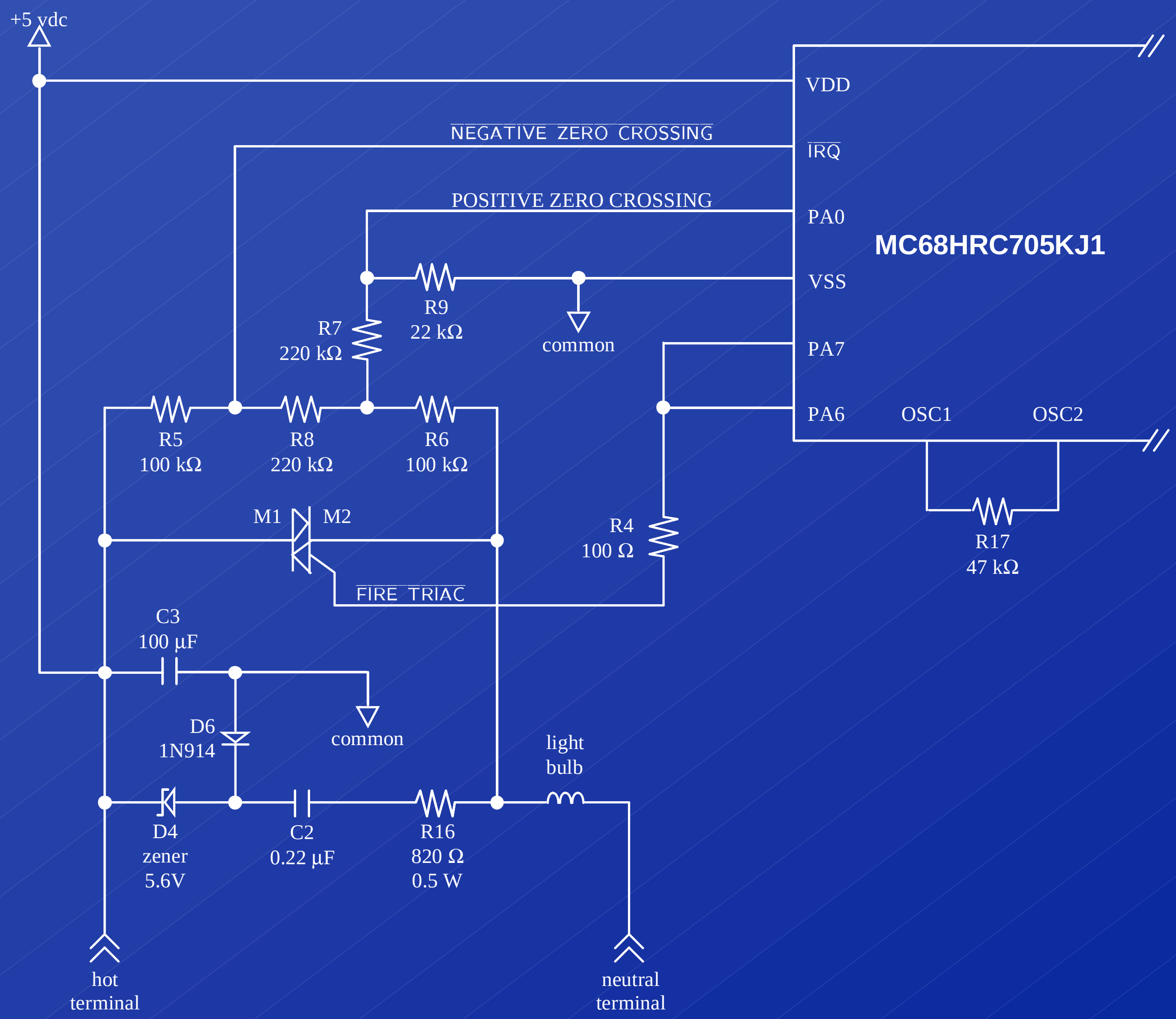
Smart Light Dimmer Application

- Smart Light Dimmer
  - MC68HC705KJ1 Overview
  - Schematics
  - Input & Output Ports
  - Multifunction Timer

MC68HC705KJ1 Features

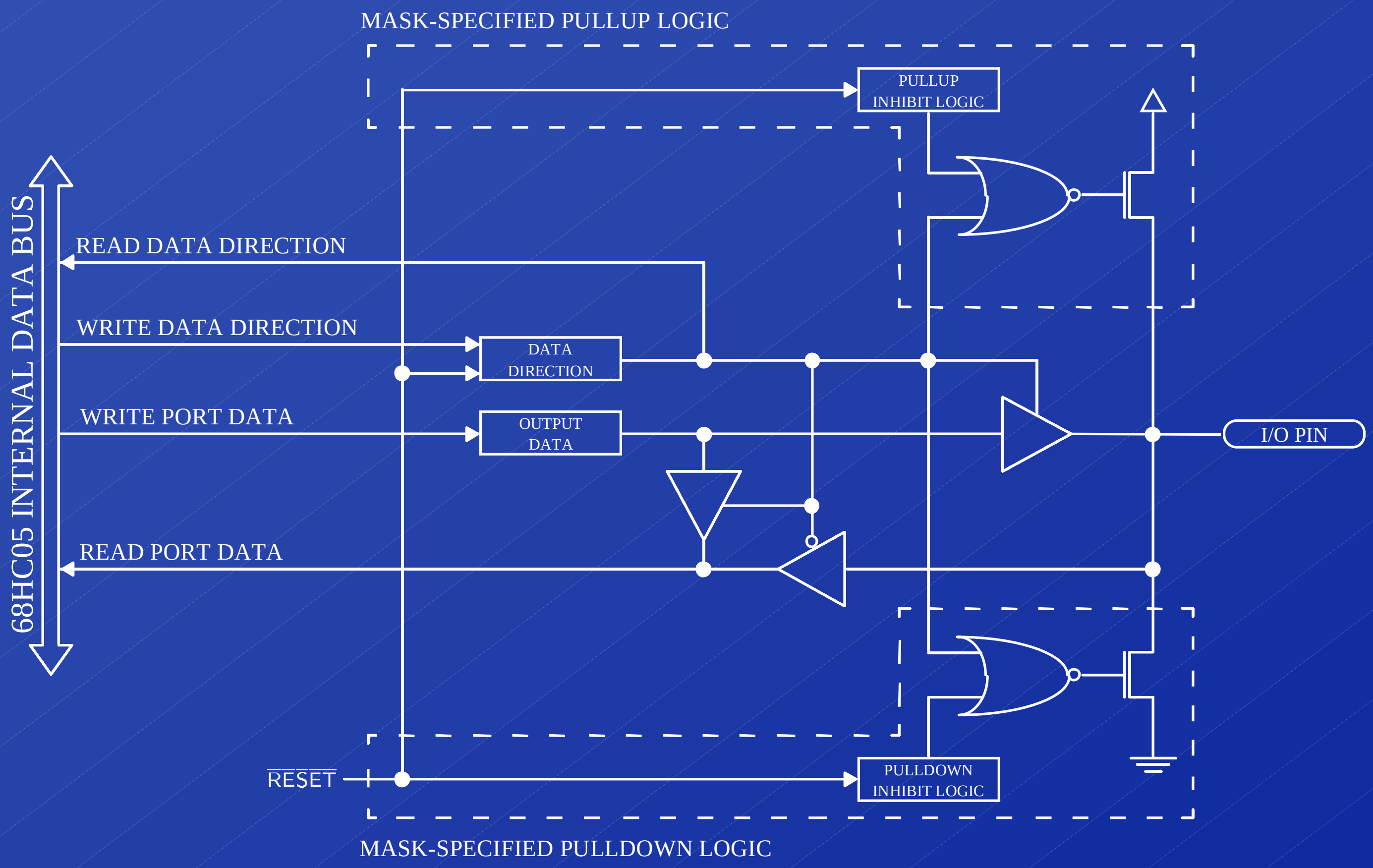
- 16-Pin Plastic DIP, Ceramic DIP, and SOIC Packages
- 4 MHz Maximum Operating Frequency at 5 Volts
- 1240 Bytes of EPROM
- 64 Bytes of RAM
- Multifunction Timer with 15-Stage Ripple Counter
- Computer Operating Properly (COP) Watchdog Timer
- 10 Bidirectional I/O Pins
  - Software Programmable Pulldown Devices on All I/O Pins
  - 10 mA Current Sink Capability on All I/O Pins
  - Optional Active High Interrupt Capability on 4 I/O Pins
- Selectable Sensitivity on External Interrupt Request Line
- On-Chip Oscillator for Crystal, Ceramic Resonator, or Resistor-Capacitor Network
- Internal Steering Diode and Pullup Device from $\overline{\text{RESET}}$ Pin to VDD

Smart Light Dimmer Schematic

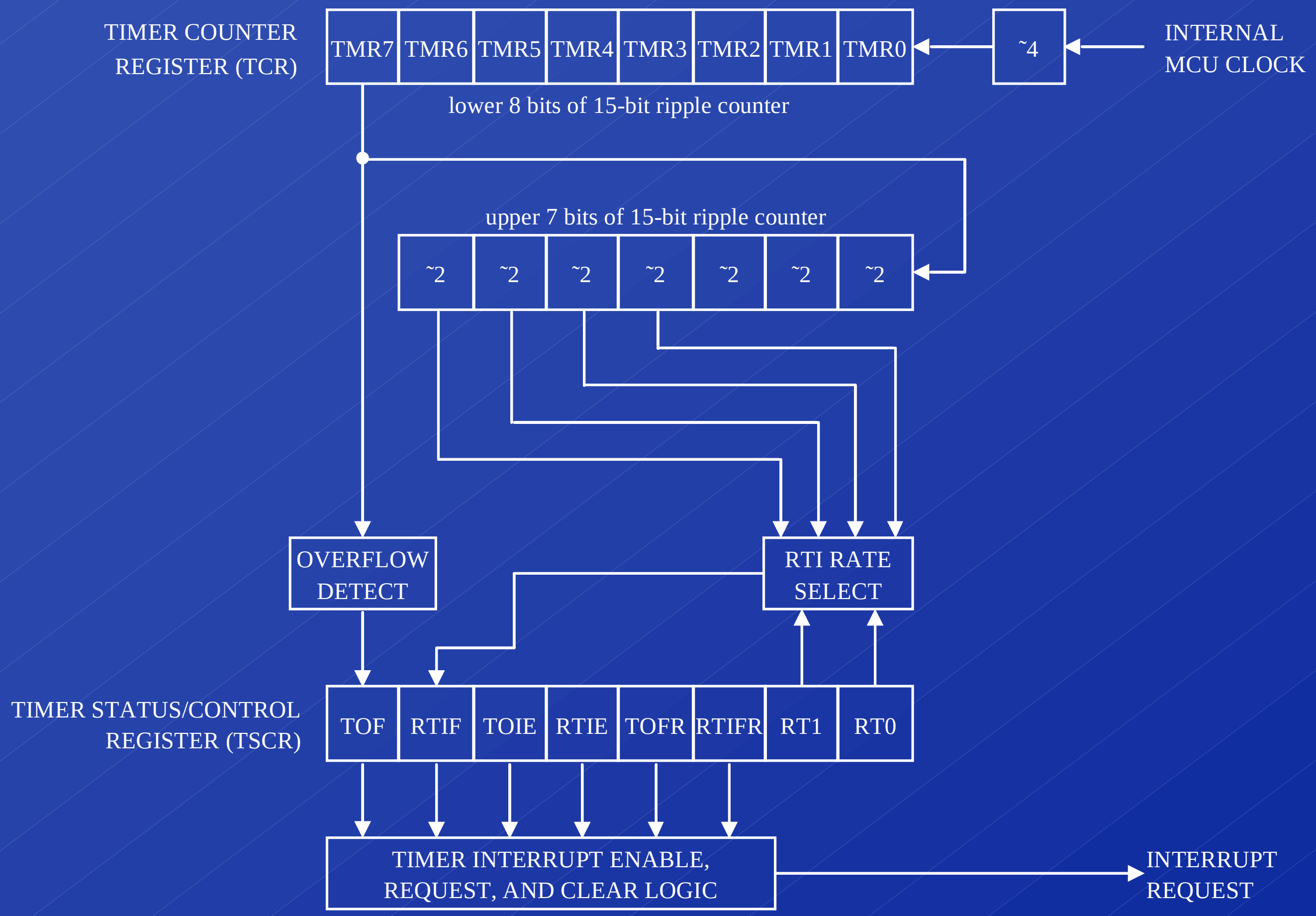


MOTOROLA

Bi-directional Port Pin Logic



Multifunction Timer (MFT)



Bicycling Computer Application

➠Bicycling Computer

- ➪ MC68HC705P6A Overview
- ➪ Block Diagram
- ➪ Analog-to-Digital Converter
- ➪ 16-bit Capture/Compare Timer
- ➪ Serial Input/Output Port

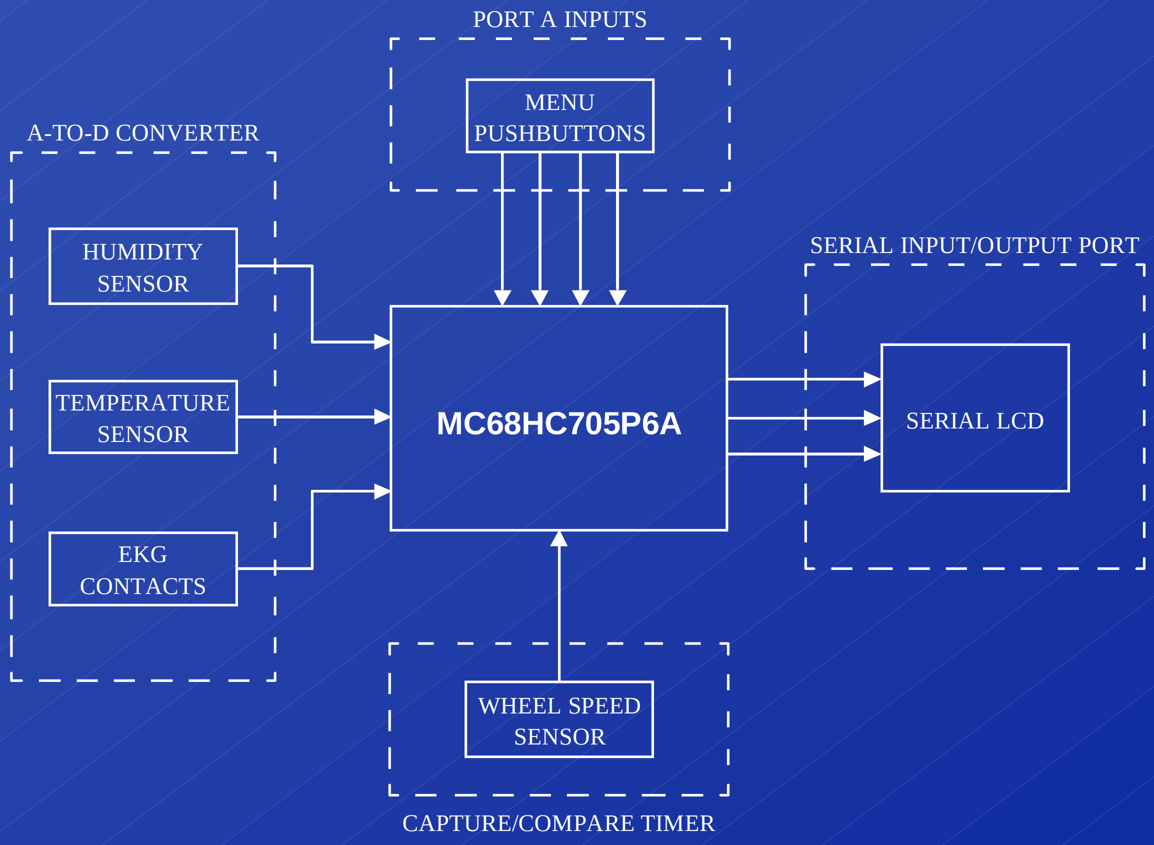
MOTOROLA

MC68HC705P6A Features

- 28-Pin Plastic DIP, Ceramic DIP, and SOIC Packages
- 2.1 MHz Maximum Operating Frequency at 5 Volts
- 4672 Bytes of EPROM
- 176 Bytes of RAM
- 16-Bit Timer with Input Capture, Output Compare, and Counter Overflow
- Computer Operating Properly (COP) Watchdog Timer
- Full Duplex, Bidirectional Serial Input/Output Port (SIOP) with 4 Baud Rates
- 4-Channel, 8-Bit Analog-to-Digital Converter
- 21 Discrete Input/Output Pins
  - 20 Bidirectional Pins (Port A[7:0], Port C[7:0], Port D5)
  - 1 Input Only Pin (Port D7)
  - Software Programmable Pullup Devices on Port A[7:0]
  - Optional Active High Interrupt Capability on Port A[7:0]
  - 10 mA Current Sink Capability on Port C[1:0]

MOTOROLA

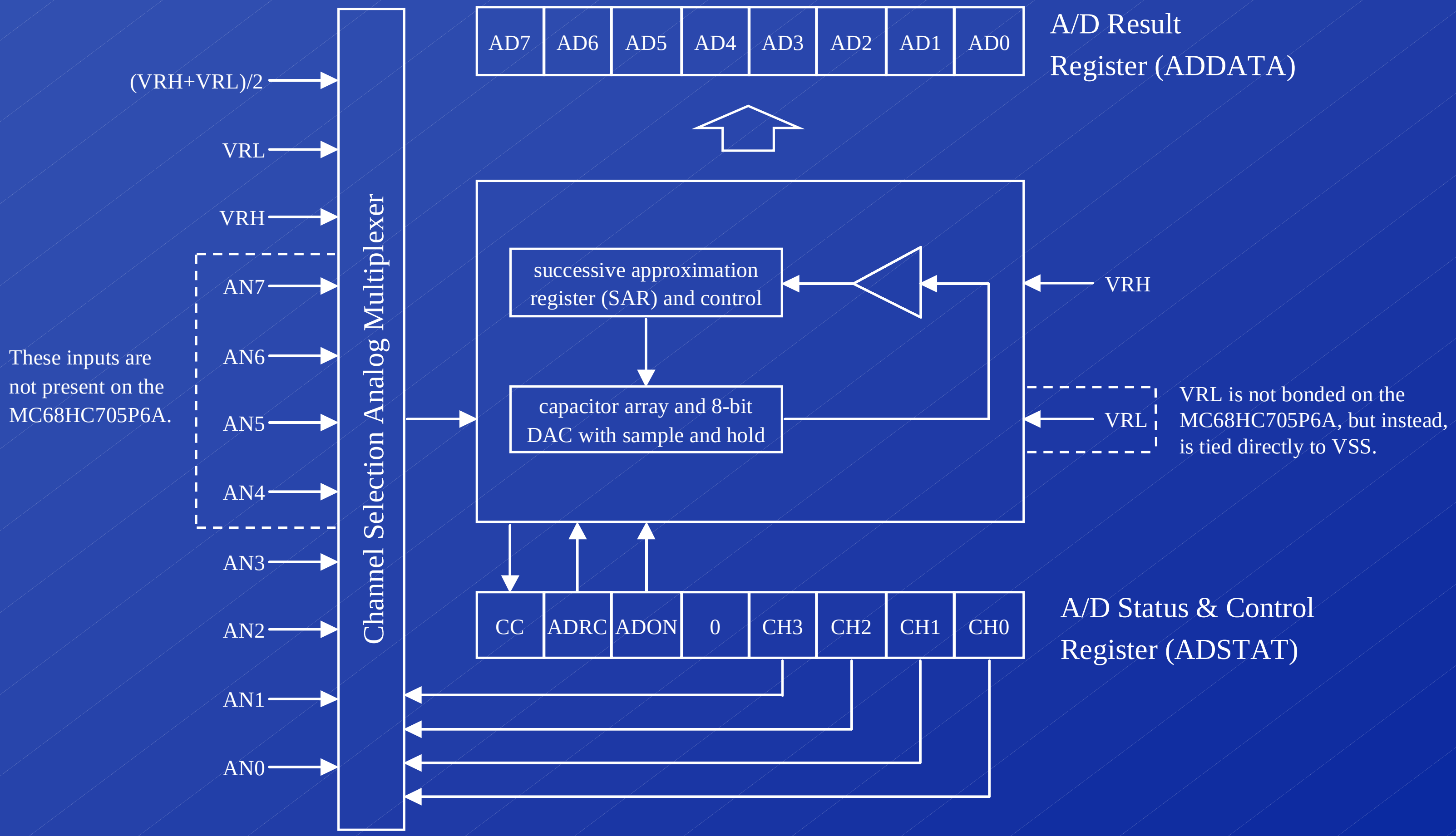
Cycling Computer Block Diagram



98/07/02

MOTOROLA

Analog-to-Digital Converter (ADC)


AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
(VRH+VRL)/2
VRL
VRH
AN7
AN6
AN5
AN4
AN3
AN2
AN1
AN0
Channel Selection Analog Multiplexer
successive approximation register (SAR) and control
capacitor array and 8-bit DAC with sample and hold
VRH
VRL
CC
ADRC
ADON
0
CH3
CH2
CH1
CH0

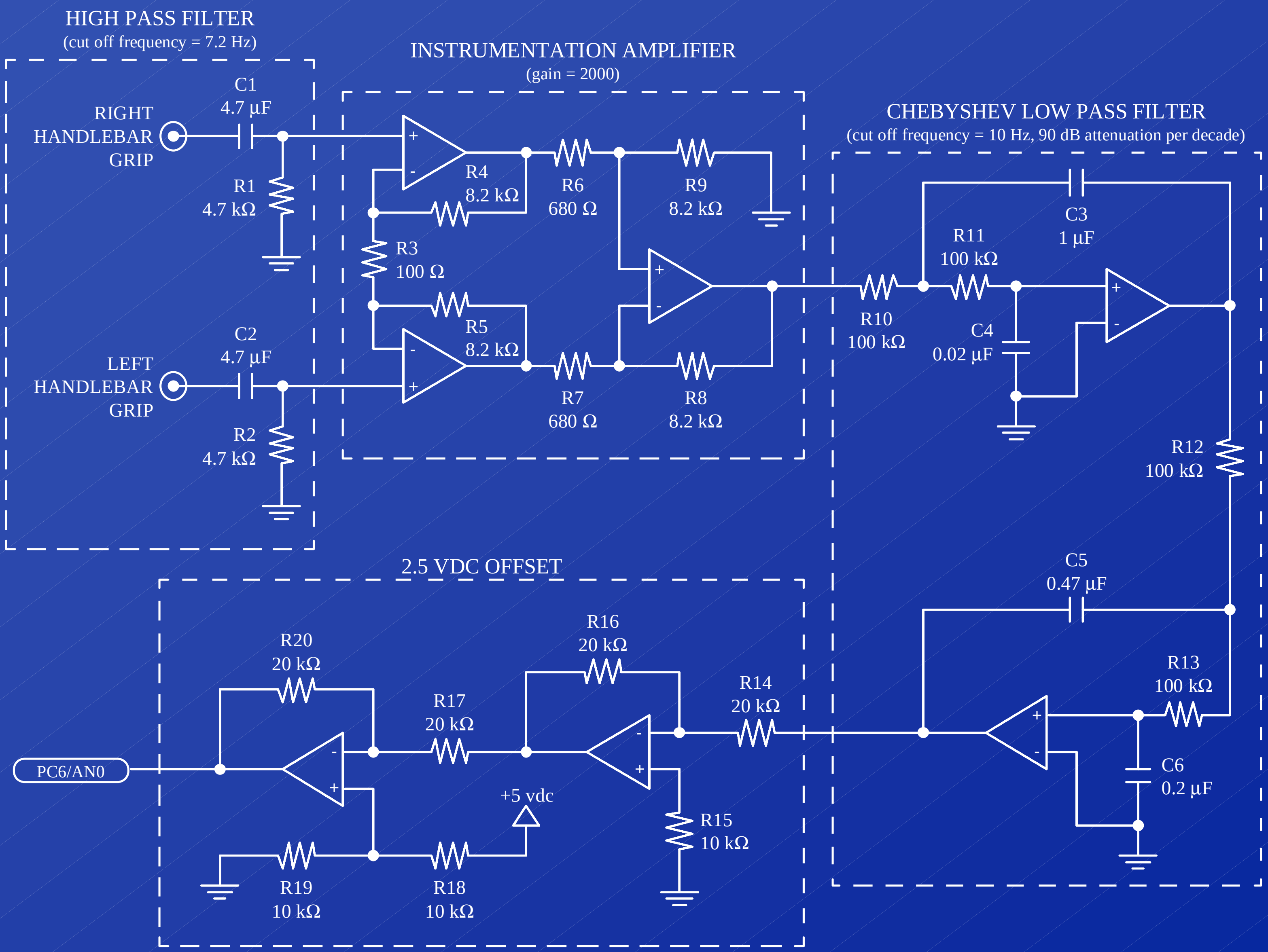
A/D Result Register (ADDATA)

These inputs are not present on the MC68HC705P6A.

VRL is not bonded on the MC68HC705P6A, but instead, is tied directly to VSS.

A/D Status & Control Register (ADSTAT)

EKG Signal Conditioning for the ADC



HIGH PASS FILTER
(cut off frequency = 7.2 Hz)

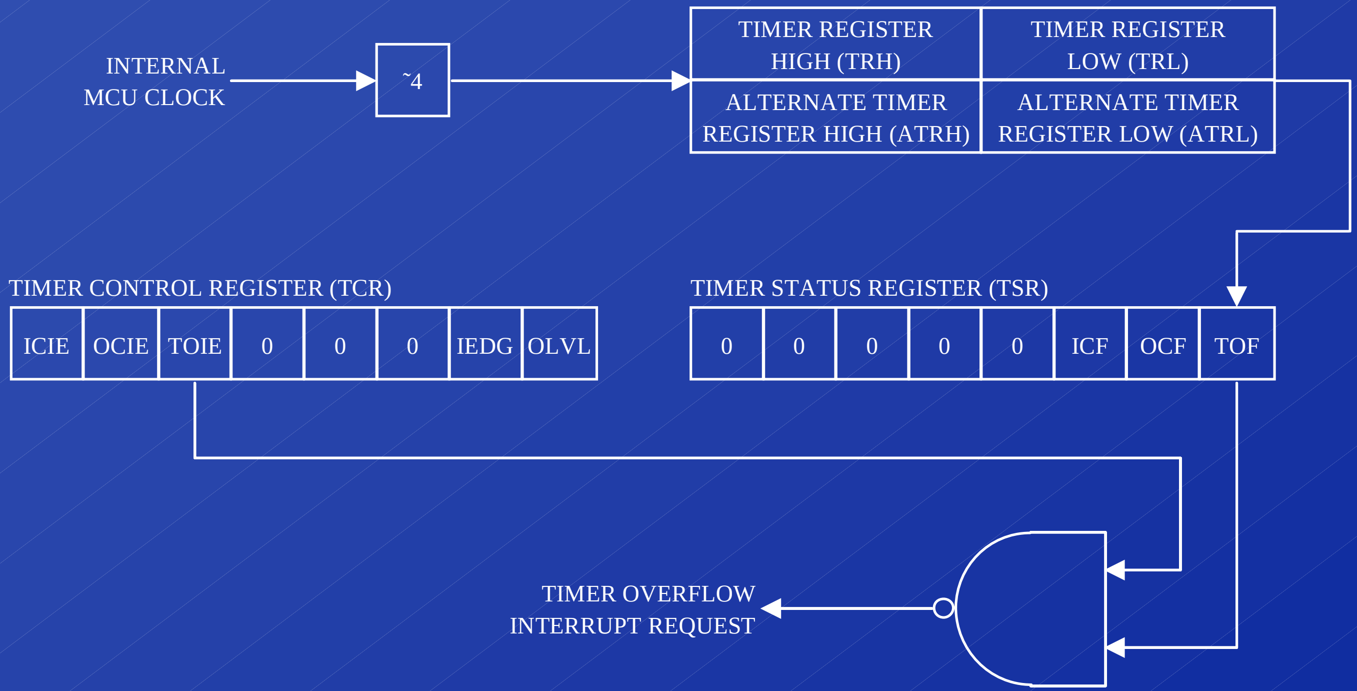
INSTRUMENTATION AMPLIFIER
(gain = 2000)

CHEBYSHEV LOW PASS FILTER
(cut off frequency = 10 Hz, 90 dB attenuation per decade)

2.5 VDC OFFSET

MOTOROLA

16-Bit Timer Overflow



INTERNAL
MCU CLOCK

˜4

TIMER REGISTER HIGH (TRH)	TIMER REGISTER LOW (TRL)
ALTERNATE TIMER REGISTER HIGH (ATRH)	ALTERNATE TIMER REGISTER LOW (ATRL)

TIMER CONTROL REGISTER (TCR)

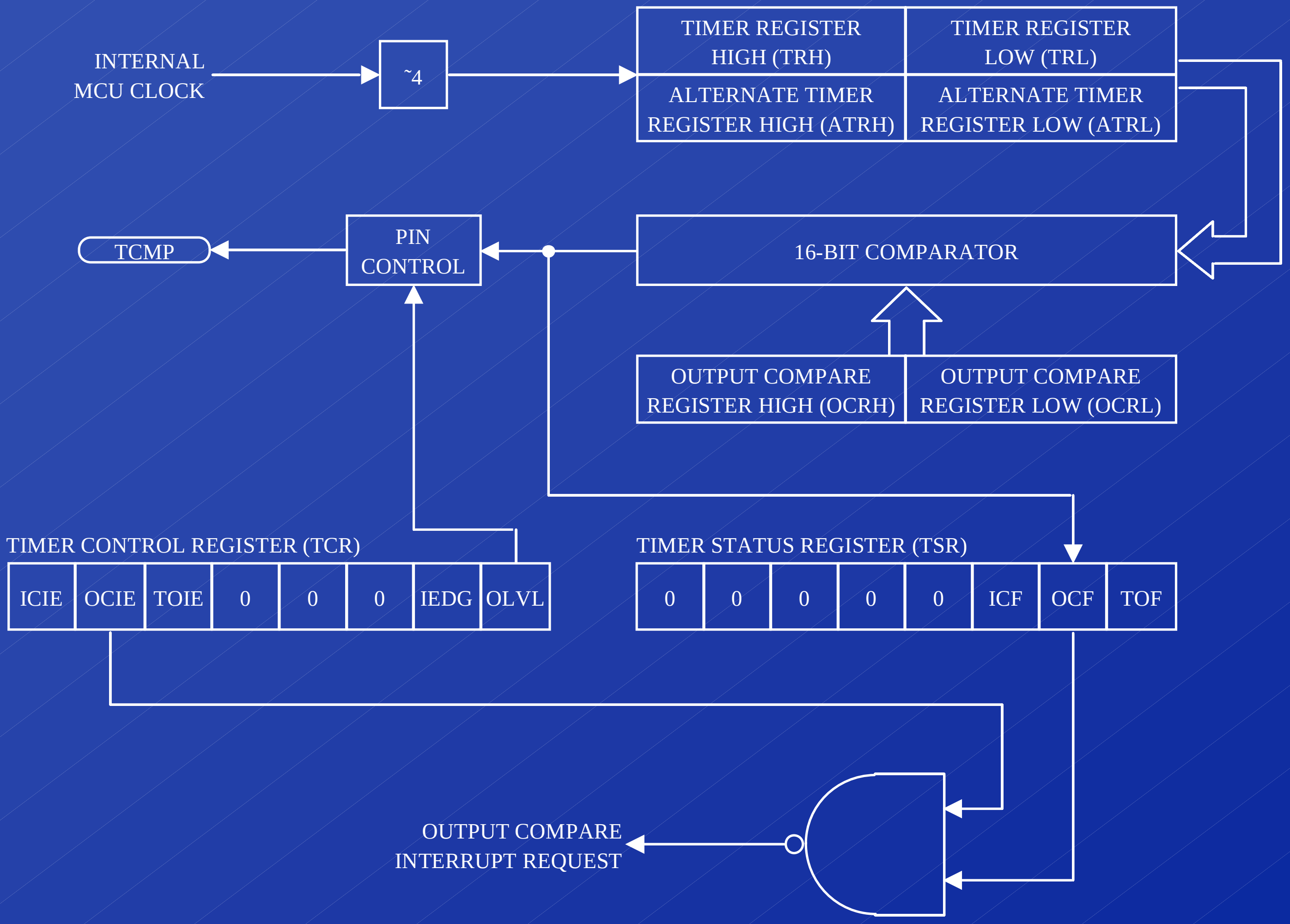
ICIE	OCIE	TOIE	0	0	0	IEDG	OLVL

TIMER STATUS REGISTER (TSR)

0	0	0	0	0	ICF	OCF	TOF

TIMER OVERFLOW
INTERRUPT REQUEST

16-Bit Timer Output Compare


INTERNAL
MCU CLOCK
˜4
TIMER REGISTER HIGH (TRH)
TIMER REGISTER LOW (TRL)
ALTERNATE TIMER REGISTER HIGH (ATRH)
ALTERNATE TIMER REGISTER LOW (ATRL)
TCMP
PIN CONTROL
16-BIT COMPARATOR
OUTPUT COMPARE REGISTER HIGH (OCRH)
OUTPUT COMPARE REGISTER LOW (OCRL)
OUTPUT COMPARE
INTERRUPT REQUEST

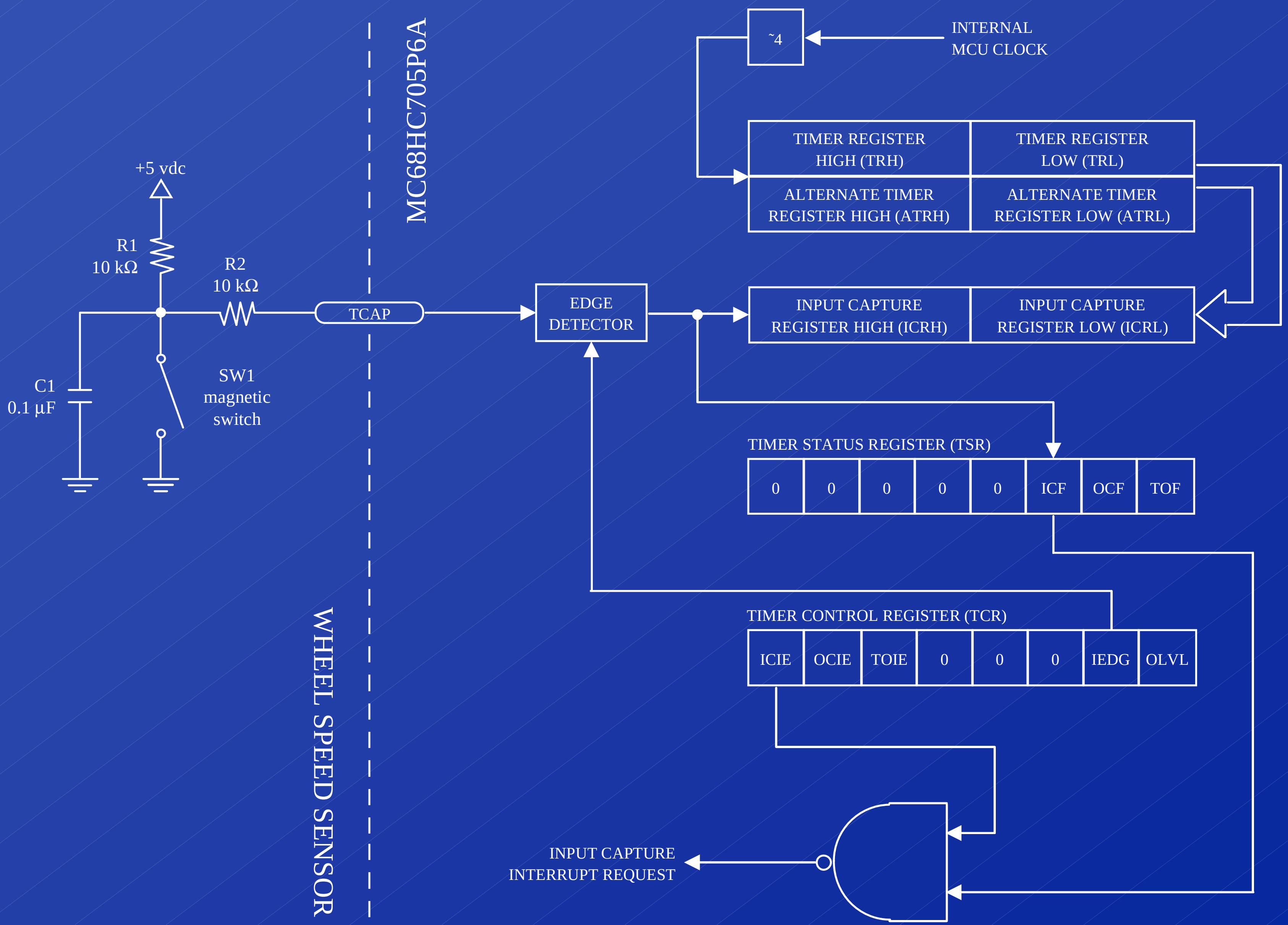
TIMER CONTROL REGISTER (TCR)

ICIE	OCIE	TOIE	0	0	0	IEDG	OLVL

TIMER STATUS REGISTER (TSR)

0	0	0	0	0	ICF	OCF	TOF

16-Bit Timer Input Capture



TIMER STATUS REGISTER (TSR)

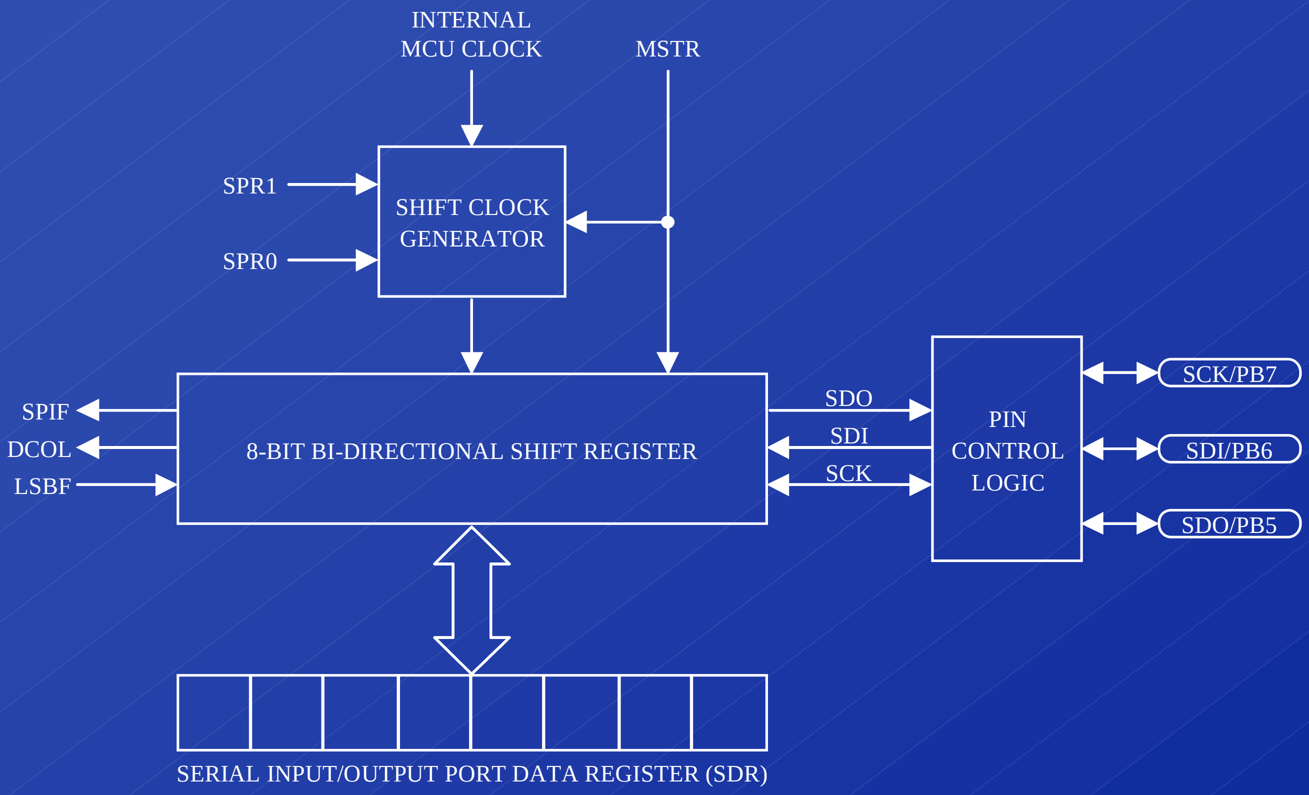
0	0	0	0	0	ICF	OCF	TOF

TIMER CONTROL REGISTER (TCR)

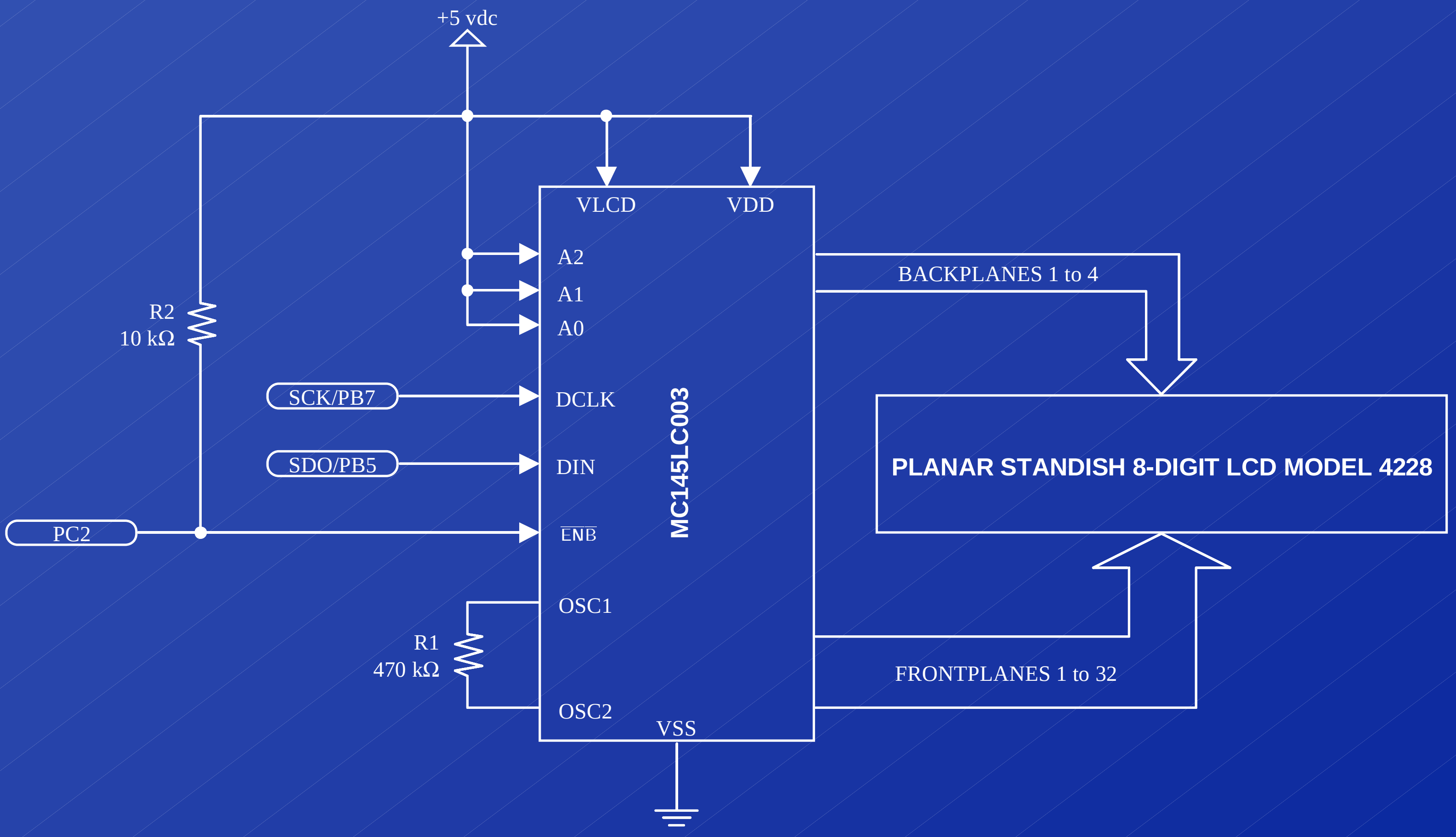
ICIE	OCIE	TOIE	0	0	0	IEDG	OLVL

MOTOROLA

Serial Input/Output Port (SIOP)



SIOP to LCD Interface

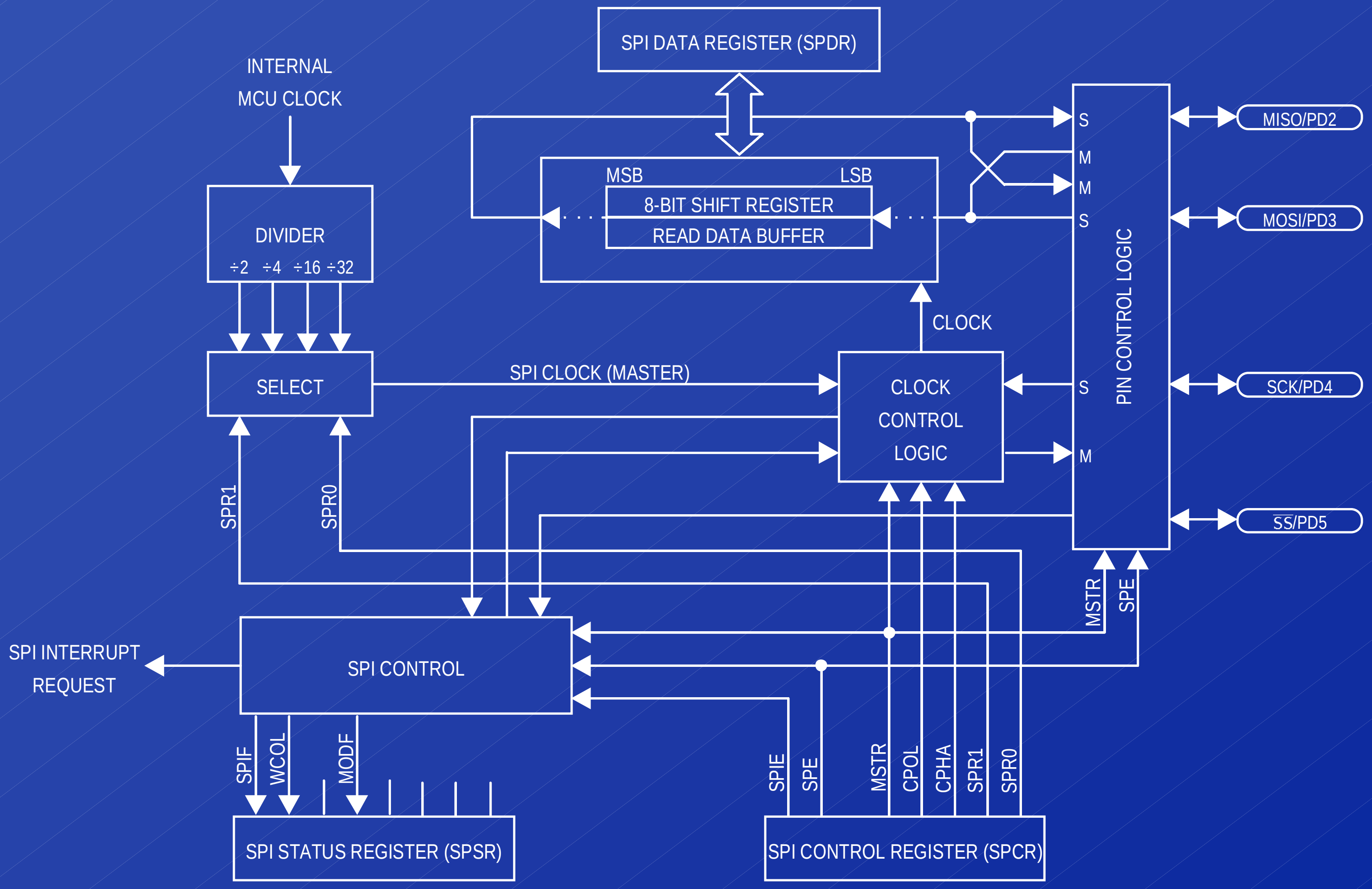


Other 68HC05 Family Peripherals

- Serial Peripheral Interface
- Serial Communications Interface
- Enhanced Serial Communications Interface
- Pulse Length Modulation Timer
- Liquid Crystal Display Driver

MOTOROLA

Serial Peripheral Interface (SPI)



Serial Communications Interface (SCI)

The serial communications interface (SCI) is the universal asynchronous receiver/transmitter (UART) on 68HC05 devices. It has the following features:

- Full duplex operation
- 32 baud rate selections
- 8- or 9-bit character lengths
- Separately enabled receiver and transmitter
- Wake up on idle line or address mark
- Optional interrupt generation upon transmit data register empty, transmission complete, receive data register full, receiver over-run, and idle line conditions
- Detection of receiver framing, noise, and over-run errors

An Enhanced SCI — The SCI+

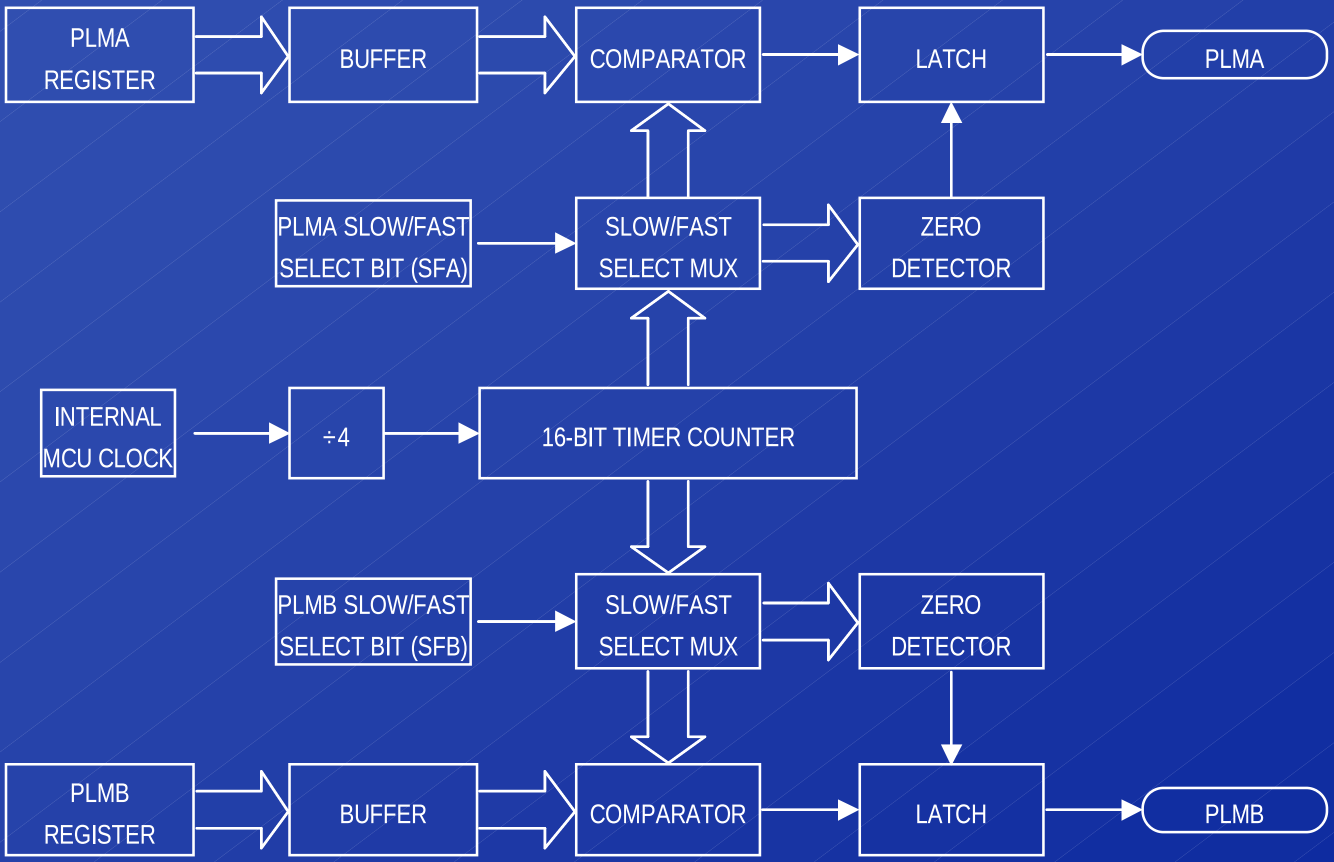
In addition to the capabilities of the standard SCI, the enhanced serial communications interface (SCI+) supports…

- Separate transmitter and receiver baud rates
- Output of the transmitter clock on the dedicated SCLK pin
- SCLK phase and polarity control
- Output-only, least significant bit first, synchronous transfers

The SCI+ essentially adds a simple, master mode, SPI-like, synchronous transfer capability to the standard SCI's UART features.

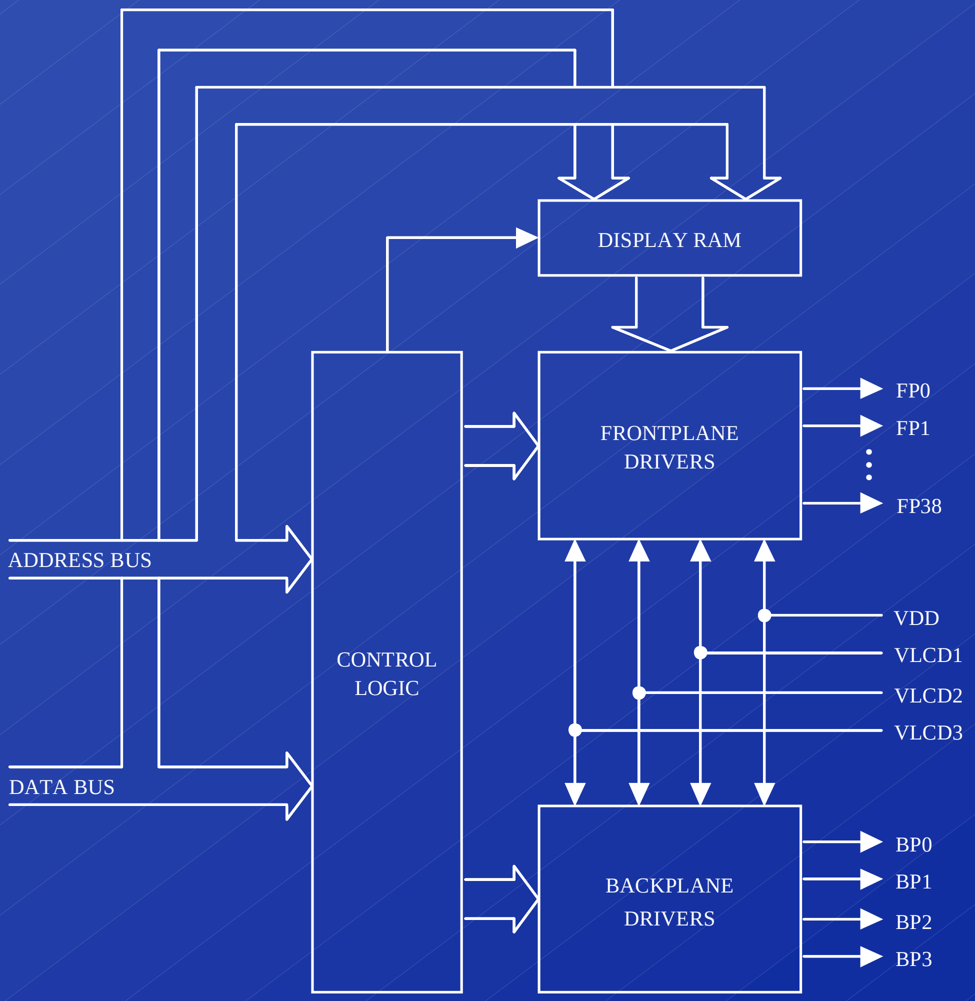
MOTOROLA

Pulse Length Modulation Timer



MOTOROLA

Liquid Crystal Display (LCD) Driver


DISPLAY RAM
FRONTPLANE DRIVERS
FP0
FP1
FP38
ADDRESS BUS
CONTROL LOGIC
VDD
VLCD1
VLCD2
VLCD3
DATA BUS
BACKPLANE DRIVERS
BP0
BP1
BP2
BP3