The MC68HC11 Microcontoller
Lecture Overheads

Sokol Salius
Chalmers Lindholmen

August 26, 2004

What would we learn about ?

- I. INTRODUCING MICROCONTROLLER TECHNOLOGY
    1. Microcontroller Concepts.
- II. SOFTWARE
    2. Programming.
    3. The Stack, Subroutines, Interrupts, and Resets.
    4. Cross Assembly and Program Development.
- III. HARDWARE
    5. Bus Concepts and Modes of Operation.
    6. Microcontroller Hardware.
    7. Clocked Operation.

What would we learn about ? (cont)

- IV. INTERFACING
    8. Interfacing Concepts.
    9. Parallel Input/Output.
    10. Serial Subsystems.
    11. Programmable Timer Operations.
    12. The Analog Converter Subsystem.
    13. Applications Control Software.

Applications of Microcontrollers (M16C)

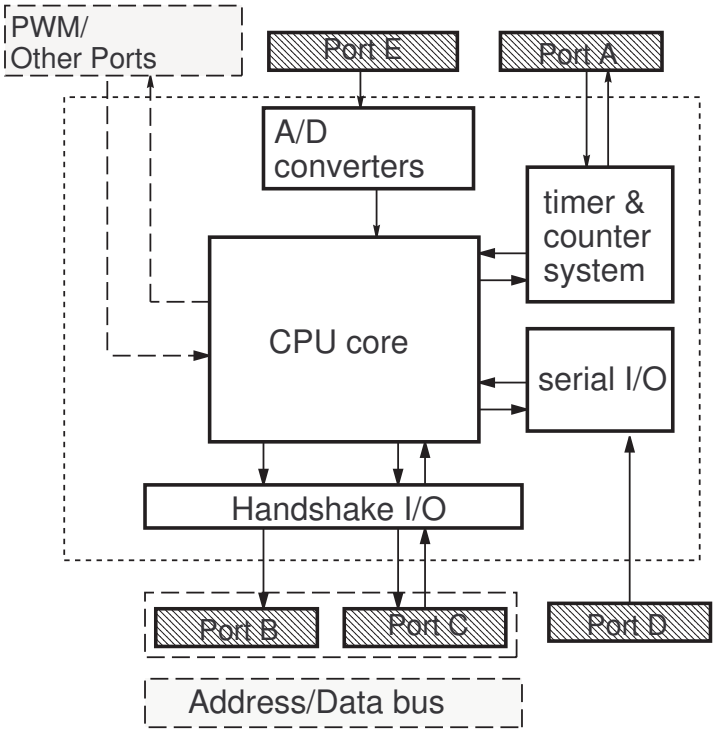
- Automotive: engine management, body comfort, security and safety, car information systems, telematics
- HVAC: heating, ventilation, air conditioning equipment, boiler control systems
- Utility metering: usage meters for electricity, gas,water,heat and power, automated meter reading systems
- White goods: Washers, dryers, dishwashers, ovens, refrigerators, freezers
- Security: fire and intrusion detection systems, sensors, CCTV systems
- Small appliances: weight scales, shavers, vacuum cleaners, sewing machines, Expresso machines
- EPOS : card readers, cash registers, Bar code readers, money handling equipment, vending machines
- Digital audio/video: DVD equipment, CD/RW products, TVs, VCRs, Set-top-boxes, portable audio devices, stereo sets, remote controls

- Industrial automation: industrial drives and pumps, robotics, door openers
- Home networking: XDSL/ISDN terminals, adapters, ISDN telephones
- Health care: fitness/glucose measurement equipment, pain releif/muscle stimulation products
- PC/Workstations: power management function, keyboard controllers

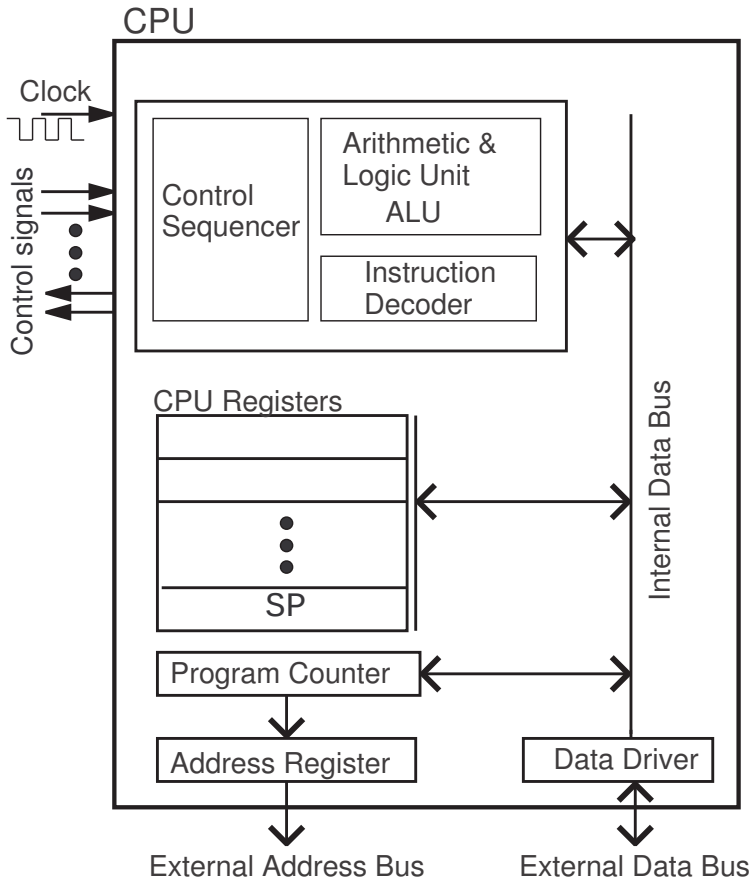
Applications of MC68HC11

- Transmission and engine control modules (Chrysler)
- Digital instruments clusters (Ford)
- Drive and emission control (Jeep Cherokee)
- Multikeyset office phone system (Motorola)
- Automatic cameras (Canon EOS model)
- Hard disk controller (Conners, Inc.)
- Scanner for reading product codes (Laser Wand)
- Cellular telephones (AT&T)
- Ski binding to reduce knee injuries (Fiock), etc.
- 'Rug Warrier' experimental home robot

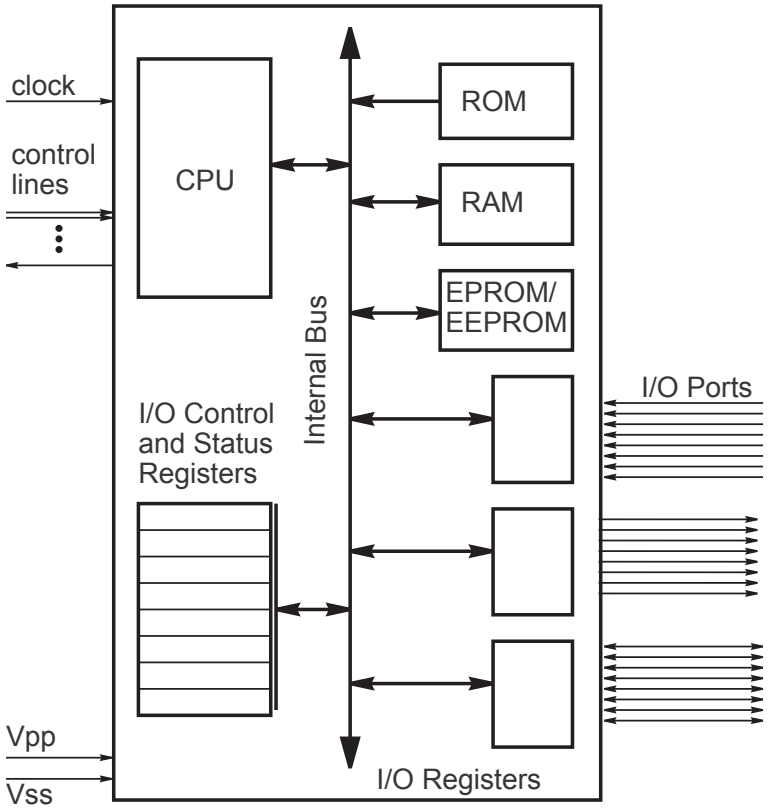
Simplified Block Diagram of MC68HC11E



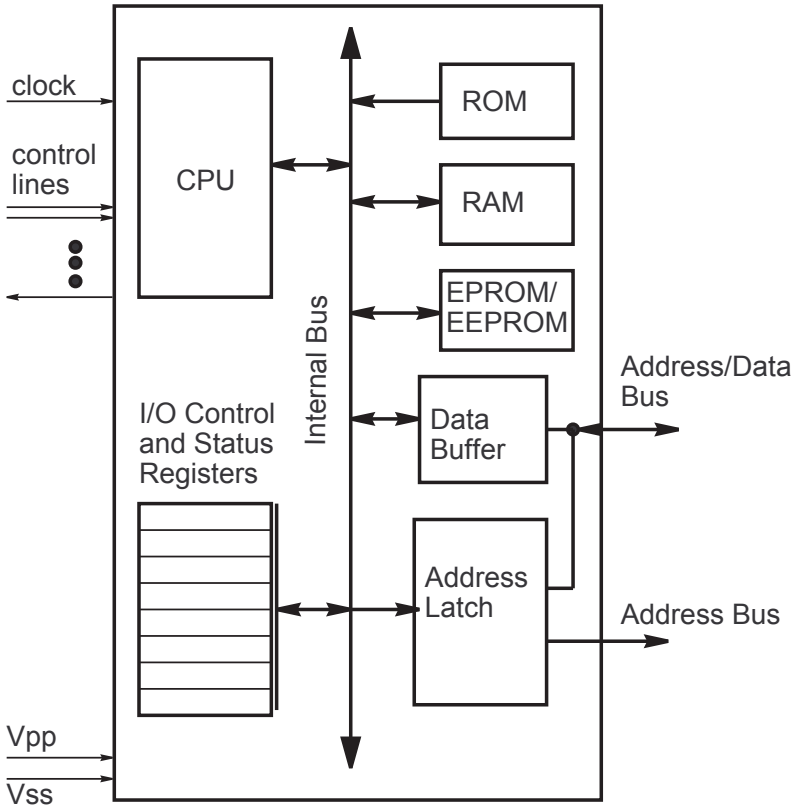
CPU Block Diagram



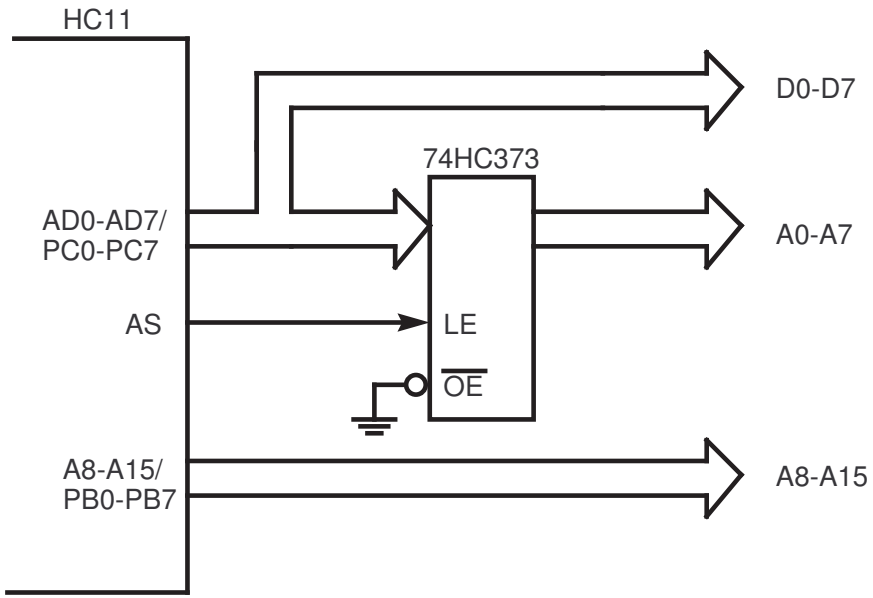
MC68HC11 in single chip mode



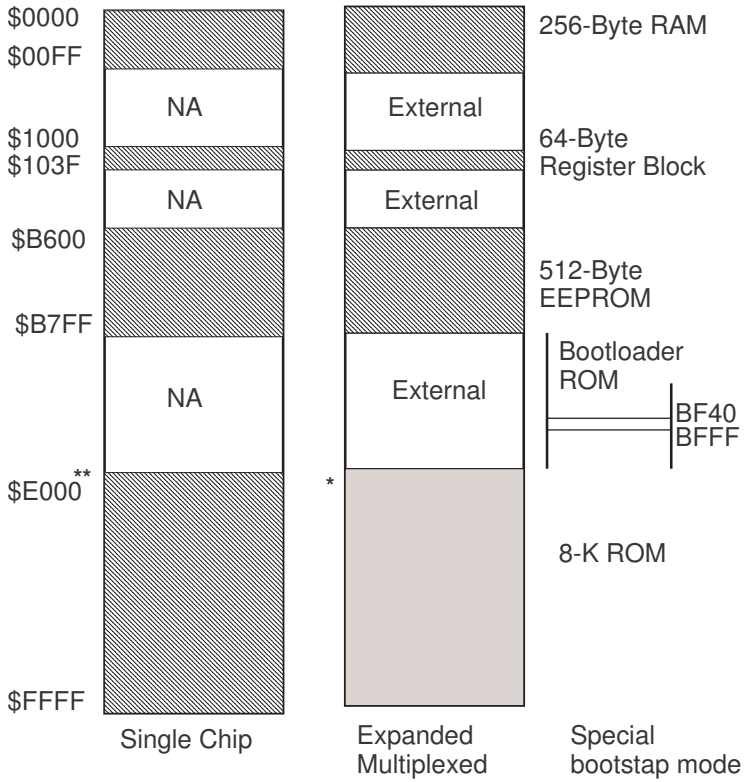
MC68HC11 in Expanded Multiplexed Mode



Demultiplexing of Address/Data Bus



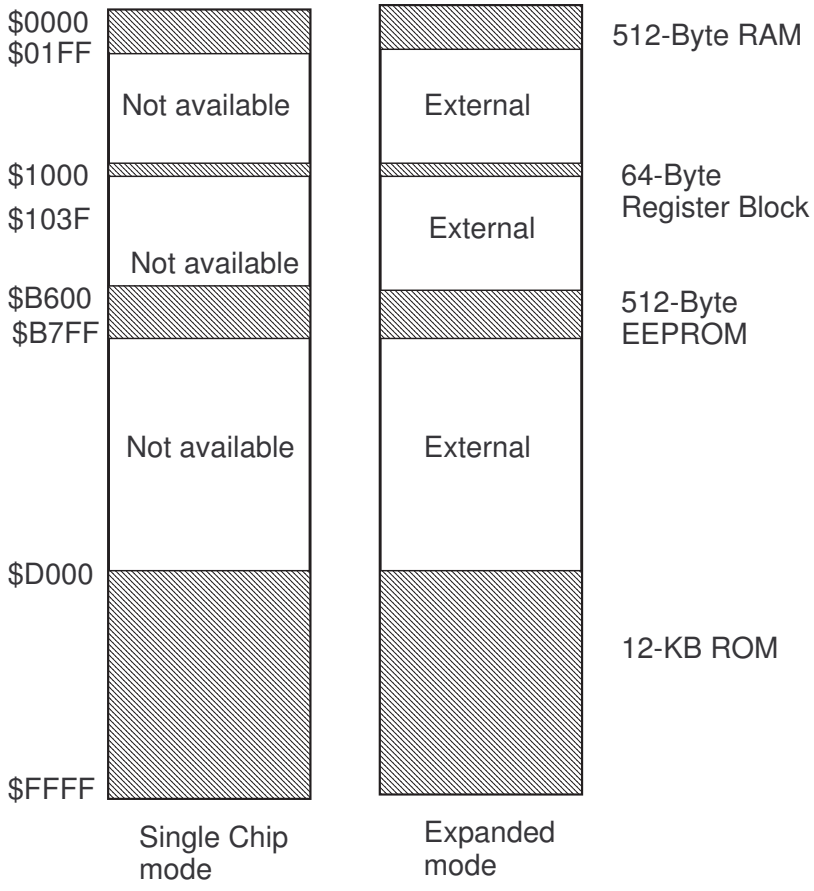
Memory map of the 68HC11A1/A8



***ROM can be disabled by ROMON bit in CONFIG**

**MC68HC11A8 delivered with ROM disabled

Memory map of the 68HC711E9



Device	RAM	ROM	EPROM	EEPROM	COMMENTS
MC68HC11A8	256	8K	0	512	16-bit Timer, 8-bit A/D, SCI, SPI
MC68HC11A1	256	0	0	512	
MC68HC11D3	192	0	4K	0	16-bit timer, SCI, SPI
MC68HC11D0	192	0			
MC68HC11E9	512	12K	0	512	16-bit Timer, 8-Ch A/D, SCI, SPI
MC68HC711E9	512	0	12K	512	
MC68HC11F1	1024	0	0	512	Nonmultiplexed bus, 8 channel 8-bit A/D, 4 chip selects, SCI, SPI
MC68HCG7	512	24K	0	0	Nonmultiplexed bus, 8 channel 10-bit A/D, 4 channel PWM, SCI, SPI, 66 I/O pins
MC68HCM2	1280	0K	32K	640	Nonmultiplexed bus, 8 channel 8-bit A/D, 4 channel PWM, DMA, On-chip math coprocessor, SCI, 2 SPI
MC68HC11N4	768	24K	0	640	Nonmultiplexed bus, 12-channel 8-bit A/D, 2 channel 8-bit D/A, 6 channel PWM, On-chip math coprocessor, SCI, SPI

μController versus μProcessor (in implementing a system)

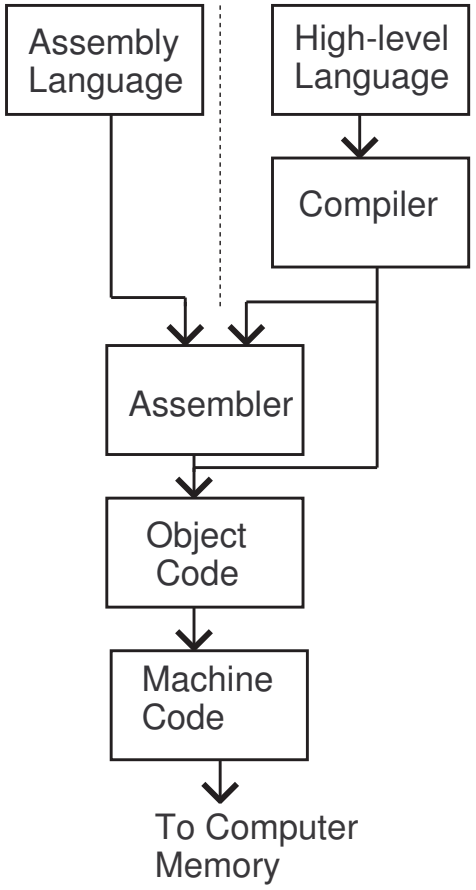
PROS

+ Fewer chips
+ Lower cost,smaller
+ Lower power
+ More user I/O pins
+ Simpler design
+ Fewer connections
+Higher reliability

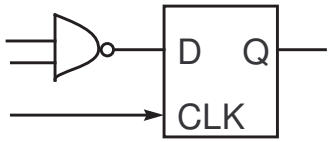
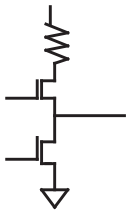
CONS

− Reduced flexibility
− Limited expansion
− Limited performance
− Limited I/O
− Tradeoff to fit all in a chip

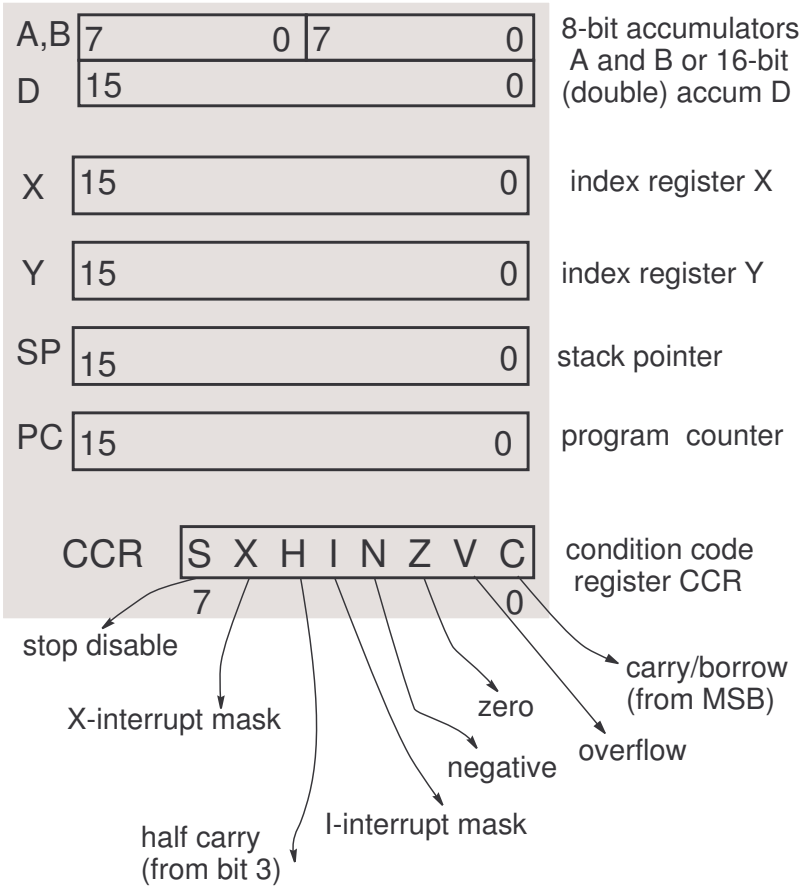
From Source to machine code



Machine Levels

High Level	++sum;
Assembly	LDAA sum INCA STAA sum
Machine	10000110 10100000 0100 1100 1001 0111 10100000
Register Transfer	Fetch Intruction Increment PC Load ALU with SUM ...
Gate	
Circuit	

Programmer's Model of MC68HC11



Instruction Set

1. **Accumulator and Memory Instructions**

- loads, stores and transfers:
  `CLR, CLRB, LDAA, PSHA, PULB, STAB, STD, TAP, TBA, XGDY, etc.`
- arithmetic operations:
  `ABX, ADCB, CMPB, DECA, NEGB, INCB, SUBD, TSTA`, etc.
- multiply and divide:
  `MUL, FDIV, IDIV`
- logical operations:
  `ANDA, ANDB, COM, COMA, EORA, ORAB`, etc.
- data testing and bit manipulations:
  `BITA, BITB, BCLR, BSET, BRCLR, BRSET`
- shifts and rotates:
  `ASLD, ASRA, LSLB,LSRA, ROLA, RORB`, etc.

2. **Stack and index registers instructions**

   - ABX,CPY, DES, DEY, INX, LDX, PULY, PHSX, TSX, TYS, XGDX, etc.

3. **Condition code register instruction**

   - CLS,CLI,CLV,SEC,SEI,SEV,TAP,TPA

4. **Program Control Instructions**

   - branches:
     BCC, BCS, BGE, BHI, BLO, BLS, BMI, BNE, BRCLR, BRSET, BVC, etc.
   - jumps:
     JMP
   - subroutine calls and returns:
     BSR, JSR, RTS
   - interrupt handling:
     RTI, SWI, WAI
   - miscellaneous:
     NOP, STOP, TEST

5. **Data (Bits) Testing:**

   - BITA M, BITB M (ACCx $\cdot$ M)

6. **Bits manipulation**

   - BSET M mask (M $\leftarrow$ M $+$ mask)
   - BCLR M mask (M $\leftarrow$ M $\cdot \overline{\text{mask}}$)

   Examples:
   - BSET $0,X $07 (xxxxx111 $\leftarrow$ xxxxxxxx+00000111)
   - BCLR $0X ~$07 (xxxxx000 $\leftarrow$ xxxxxxxx $\cdot$ 11111000)

7. **Branch instructions based on bit values**:

- BRSET M mask rel $(\overline{\mathrm{M}} \cdot \mathrm{mask} == 0)$
- BRCLR M mask rel $(\mathrm{M} \cdot \mathrm{mask} == 0)$

Examples:

- BRSET 0,X # $07 LABEL1
  (if $\overline{\mathrm{xxxxxxxx}} \cdot 00000111 = 00000\overline{\mathrm{xxx}} == 00000000$)
- BRCLR 0,X #$07 LABEL2
  (if $\mathrm{xxxxxxxx} \cdot 00000111 = 00000\mathrm{xxx} == 00000000$)

Addressing modes

Addressing modes define how the microcontroller calculates an address

inherent	CLRA		data is inherent to CPU—no external address is needed
immediate	LDX	#$1000	data is located 'immediately' in the instruction
extended	STAA	$1000	the 16-bit address of the a memory byte is specified in the instruction
direct	STAB	>$20	the 8-bit address ($00–$FF) of chip's RAM is specified in the instruction
indexed	LDAB	$05,X	the*(effective) address* is obtained from register X content plus the offset
relative	BNE	$0C	*location* is specified by an offset from current instruction

List file (addressing modes)

1000		ORG	$1000	
1000 8E 00 FF		LDS	#$FF	*Immediate*
1003 C6 80		LDAB	#$80	
1005 CE C0 00		LDX	#$C000	
1008 B7 00 C0		STAA	$C0	*Extended*
100B 97 C0		STAA	>$C0	*Direct*
100D E3 00		ADDD	0,X	*Indexed*
100F 18 A7 03		STAA	3,Y	
1012 1B		ABA		*Inherent*
1013 08		INX		
1014 20 00	PREV	BRA	NEXT	*Relative*
1016 22 FC	NEXT	BHI	PREV	
1018 24 02		BCC	OK	
101A 20 FE	LOOP	BRA	LOOP	
101C 7E D0 00	OK	JMP	$D000	*Absolute*

‘HC11’s missing conditional jump’

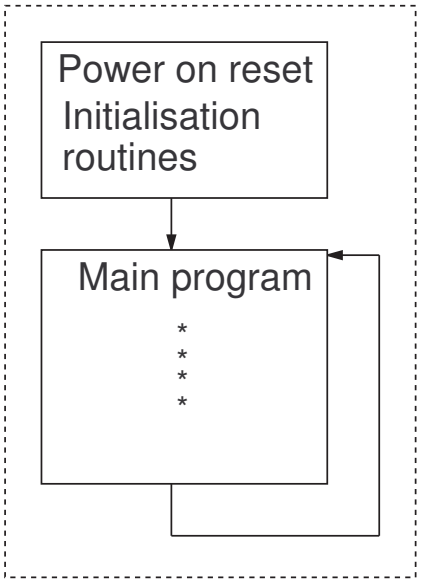
```
*--------------------------------
*   Example: "JNE"
*--------------------------------
   ...
   DECA
*--------------------
   BEQ    CONT
   JMP    LABEL
CONT
*--------------------
     ...
LABEL
     ...
```

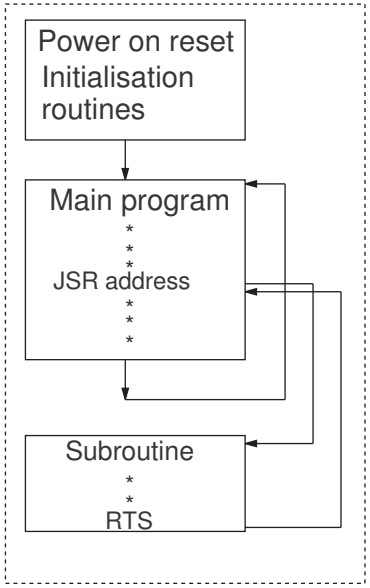
Programming constructs

```
* --------------------
*   if ... else
*   returns 'F' or 'T' in  B
* --------------------
*if
    TSTA
    BEQ AdT    if (condition TRUE)
*else
     LDAB #'F'
     BRA  AdF
  AdT  LDAB #'T'  set TRUE flag
  AdF
```

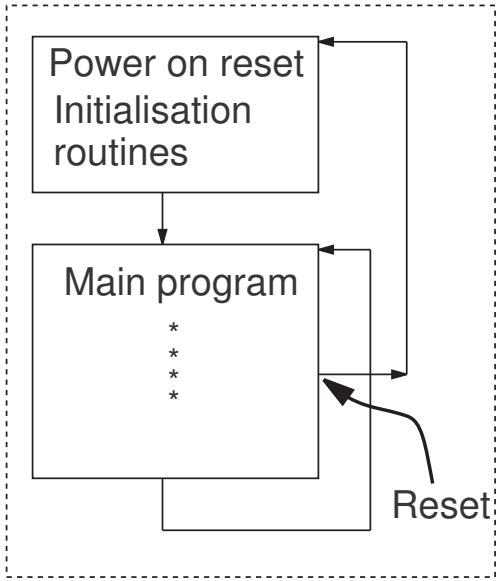
```
*----------------------
*   do ... while ...
*
*-----------------------
*do
       LDAB      #10
   DOW LDAA    $0,X   read from
       STAA     $0,Y   write to
       INX
       INY
 *while (B) < 10
        BNE     DOW
```

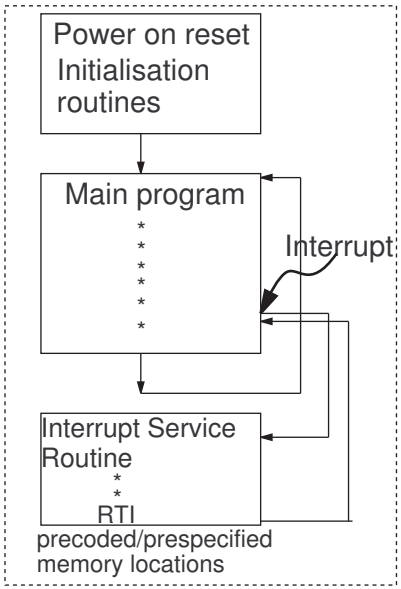
Normal Execution





Execution in presence of Resets and Interrupts





Resets and Interrupts

- Resets
  - Power on reset(POR)
  - $\overline{\mathrm{RESET}}$
  - Clock monitor fail
  - COP fail
- Non-maskable Interrupts:
  - $\overline{\mathrm{XIRQ}}$
  - Illegal opcode (ILLOP)
  - Software interrupts(SWI)

- Maskable interrupts:
  - Timer overflow
  - PAC overflow, PAC input edge
  - SPI transfer complete
  - SCI Serial system
  - $\overline{\mathrm{IRQ}}$
  - RTII
  - TIC1–3
  - TOC1–5

- Features
  - Generated by on-chip peripherals
  - Have fixed interrupt priorities
  - Any interrupt can be elevated to the highest priority level–by modifying HPRIO
  - *TIC1–3, PAC IC, STRA/AS can be configured as external interrupt sources(in single-chip mode)*

Address vector	Interrupt source	CCR mask	Localmask	Pseudovector
FFD6	SCI	I bit	TIE/TCIE/RIE/ILIE	3FD6
FFD8	SPI	I bit	SPIE	3FD8
FFDA	PAIE	I bit	PAII	3FDA
FFDC	PAO	I bit	PAOVI	3FDC
FFDE	TOF	I bit	TOI	3FDE
FFE0	TOC5	I bit	OC5I	3FE0
FFE2	TOC4	I bit	OC4I	3FE2
FFE4	TOC3	I bit	OC3I	3FE4
FFE6	TOC2	I bit	OC2I	3FE6
FFE8	TOC1	I bit	OC1I	3FE8
FFEA	TIC3	I bit	IC3I	3FEA
FFEC	TIC2	I bit	IC2I	3FEC
FFEE	TIC1	I bit	IC1I	3FEE
FFF0	RTI	I bit	RTI	3FF0
FFF2	IRQ	I bit	None/STAI	3FF2
FFF4	XIRQ	X bit	None	3FF4
FFF6	SWI	None	None	3FF6
FFF8	ILLOP	None	None	3FF8
FFFA	COP	None	NOCOP	3FFA
FFFC	CLM	None	CME	3FFC
FFFE	RESET	None	No	3FFE

Table 1: Interrupt Vector Assignments

Monitor Mappings of Interrupts

```
*    BUFFALO Listing
*    mapping of vectors in RAM ($3FF6--$3FFE)
       ...

EE3B  JTOF   LDX  $3FDE      *RAM vector address  ($3FDE)
      JMP    0,X             *Pseudo-vector in RAM of vector JTOC
EE40  JTOC5  LDX  $3FE0      *RAM vector address ($3FE0)
      JMP    0,X             *Pseudo-vector in RAM of vector TOC5
       ...
       ...
FFDE  VTOF   FDB  JTOF        *TOF     interrupt vector FFDE
FFE0  VTOC5  FDB  JTOC5       *TOC5    interrupt vector  FFE0
      ...
FFFE  VRST   FDB BUFFALO      *Reset vector
```

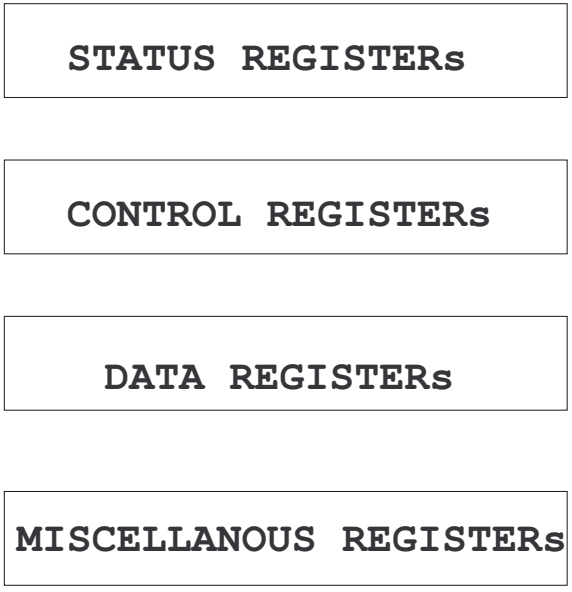
Initialisation of interrupt vectors

```
LDX  #TOF_ISR
STX  #3FDE
LDX  #TOC5_ISR
STX  #3FF0
  ...

*----------
TOF_ISR           *TOF User interrupt service routines
  ...
  RTI
*----------
TOC5_ISR           *TOC5 User interrupt service routines
  ...
  RTI
```

PART III
I/O INTERFACES/SUBSYSTEMS

- Programmable Timers
  - Output compare
  - Input capture
  - Pulse accumulator
  - Real-time interrupt
- The Serial Subsystems
  - Serial Peripheral Interface
  - Serial Communication Interface
- Parallel Input/Output
  - Simple parallel I/O
  - Handshake parallel I/O
- A/D converter
- Other subsystems

- Generic registers

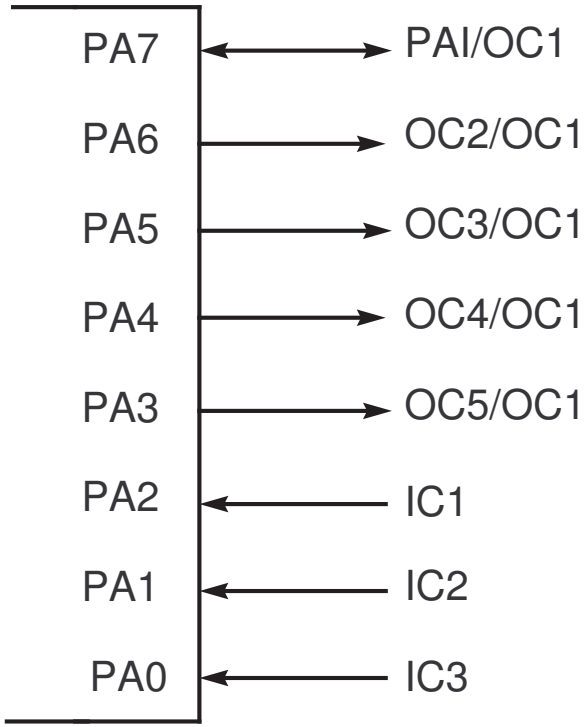


Main Timer & and Real Time Interrupt

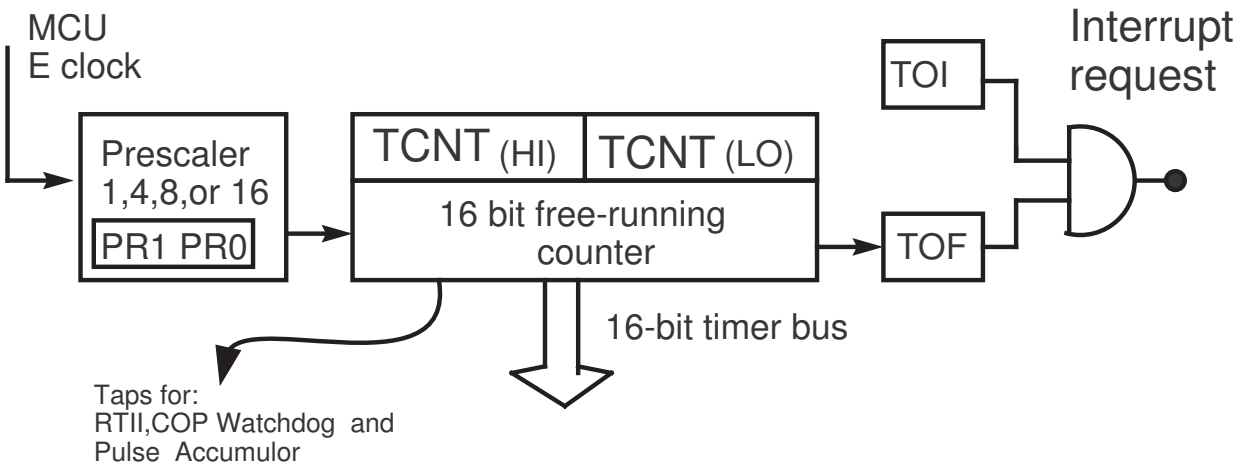
- Timer subsystem functions/modules
  - Output compare
  - Input capture
  - Pulse accumulator,also
    RTII (real time interrupt)
    COP (computer operating properly)

- Applications
  - Decisions in *Real time*
  - Precise timing of events , e.g.
    (i) time measurement of input events,
    (ii) timing of output events
  - Counting of events
  - Periodic real time interrupts
  - Watchdog

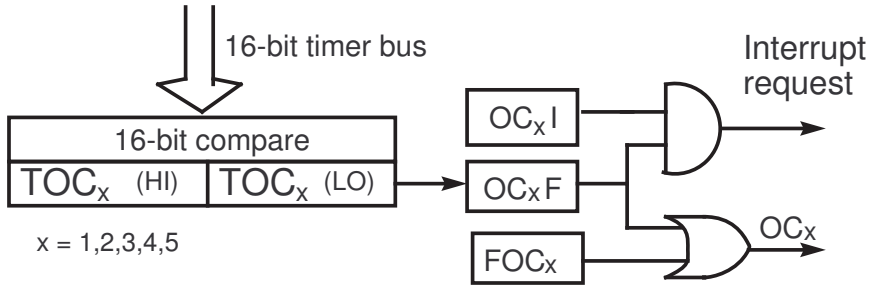
Port A pins

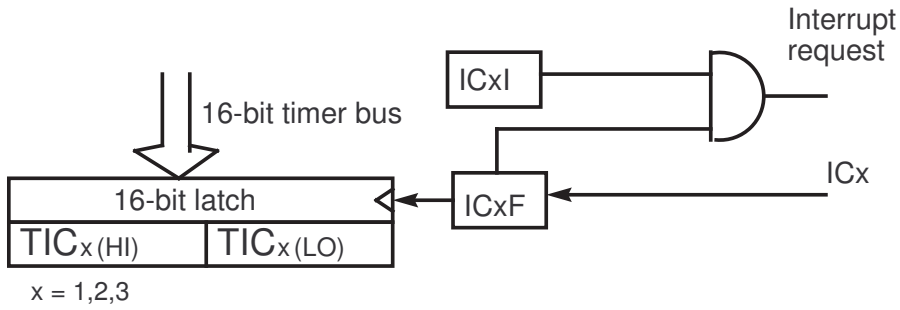

PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PAI/OC1
OC2/OC1
OC3/OC1
OC4/OC1
OC5/OC1
IC1
IC2
IC3

Free-running counter and prescaler

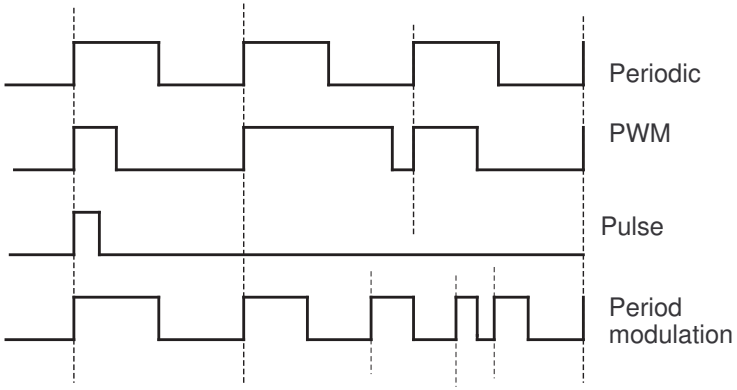


Timer Output compare & Input capture

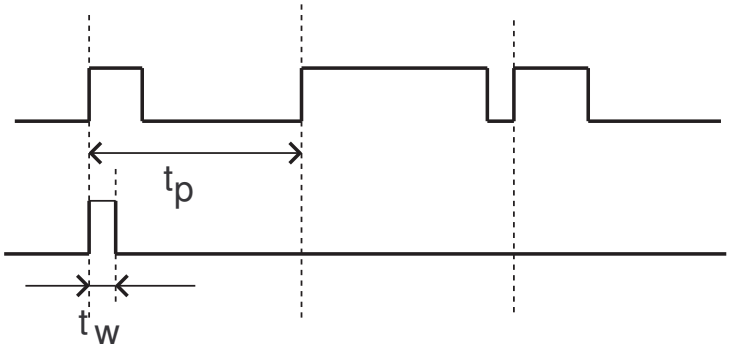




OC,IC Applications



Periodic pulse/waveform generation



Period/pulse-width measurement

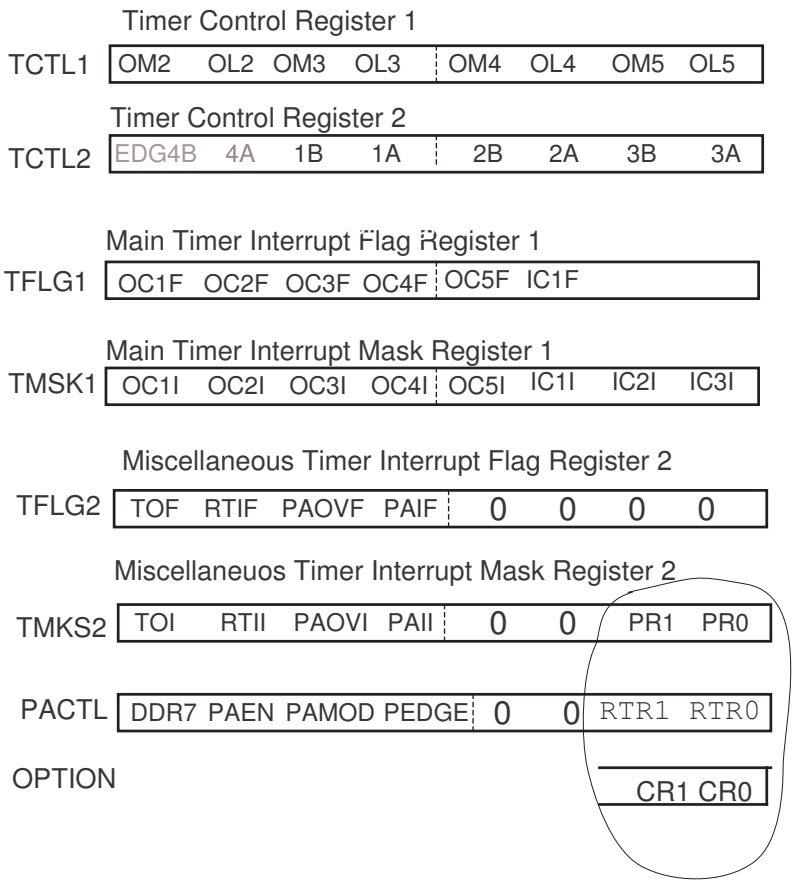
Typical (timer) software sequence

- Configure the control registers
- Write to the data register (if required)
- Wait for a flag to be set
- Clear the flag
- Read or write data as required

- Configure the control registers & interrupt vector(s)
- Write to the data register (if required)
- WAI (wait for interrupt)

- Clear the flag
- Read or write data as required
- RTI

Timer subsystem's control and status registers

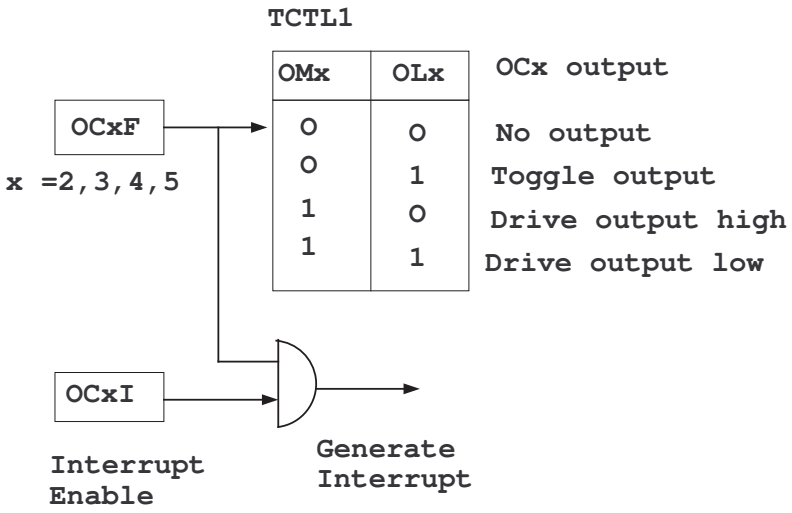


Rates Selection

	TMSK2 PR1 PR0	PACTL RTR1 RTR0	OPTION CR1 CR0
	Timer prescaler resolution/overflow	*RTI rate select*	*COP time out*
00	500 ns/ 32,77ms	4.10 ms	*16.384 ms*
01	2 us / 131.1 ms	8.19 ms	*65.536 ms*
10	4us / 262.1 ms	16.38 ms	*262.14 ms*
00	8us / 524.3 ms	32.77 ms	*1.049 s*

E-clock 2 MHz

Output Compare



Additional OC1 registers

OC1M

OC1D

On succesful compare on TOC1 the data in OC1D are output to selected (OC1M) lines (PA3--PA7)

One shot pulse

```
*10 ms one-shot pulse on OC2/PA6; E = 2MHz, pre-scaling 1
 PWIDTH  EQU  $2000  *E =2MHz
 BASEREG EQU  $1000
     ORG     $2000
     LDX     #BASEREG
     LDD     TCNT,X  Prevent
     STD     TOC2,X  premature OC
     BSET    PORTA,X,$40 PA6/OC2 high
     LDAA    #$80
     STAA    TCTL1,X drive output low
     LDAA    #$40
     STAA    TFLG1,X clear OC2F is set
     LDD     TCNT,X
     ADDD    #PWIDTH- 17
     STD     TOC2,X
```

```
*Wait for trigger by polling for OC2F high
 PULSE1
 BRCLR   TFLG1,X,$40,PULSE now output low
 BCLR    PORTA,X,$40 clear latch for PA6
 LDAA    #$40
 STAA    TFLG1,X     clear OC2F
 BCLR    TCTL1,X $80 disconnect OC2
 OK      BRA OK
```

OC1 example !

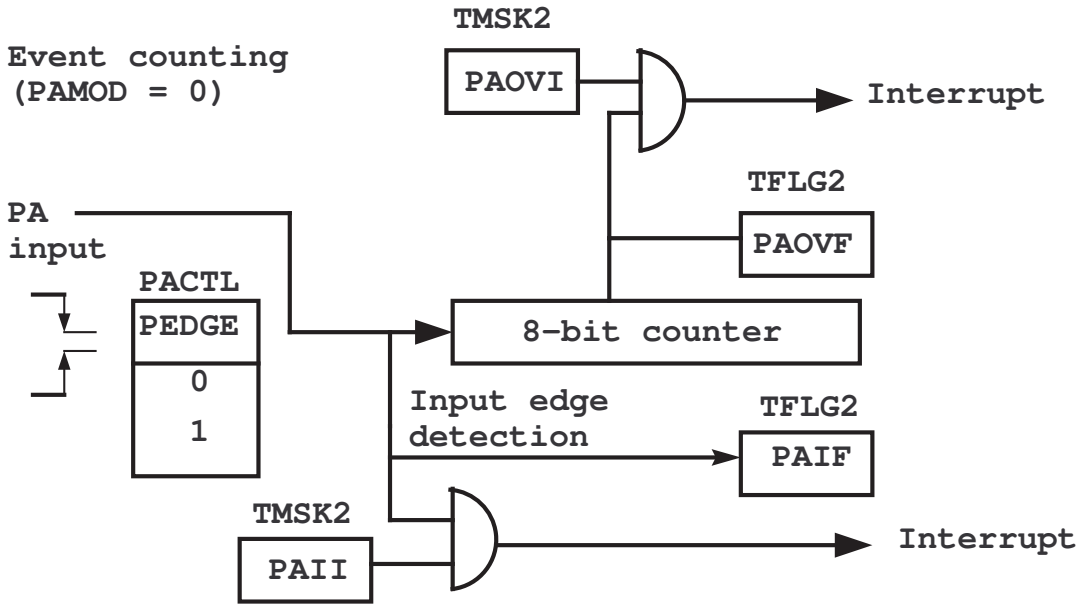
```
*The following code would generates the pattern 0xx11(out of pins PA7,PA4,PA3)
*on a successful TOC1 compare. Uses OC1M and OC1D registers
    LDX     #BASEREG
    BSET    PACTL,X,$80  *Makes PA7 output

    LDAA    #$98
    STAA    OC1M,X       sets OC1M bits 7,4,3 (clears the others)
*                         and drives PA 4,3 high, PA7 low
    LDAA    #$18         sets OC1D4,3, clears OC1D7
    STAA    $OC1D X

    BSET    TMSLK1,X \$80  *Enables OC1I interrupt
    CLI
    ...
```

IC Capture

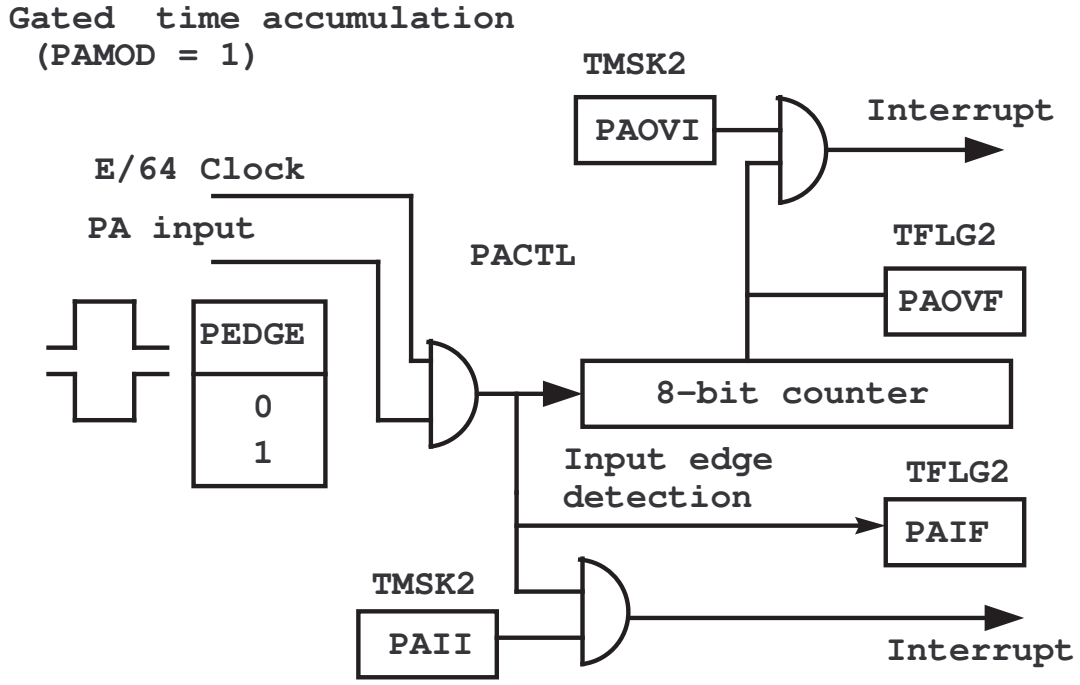
Measurement of pulse width

```
*polling  rising and falling edges of IC1
    ORG     $2000                                  P_FALL
    LDX     #$1000   Register base                     BRCLR   TFLG1,X $04 P_FALL
    LDAA    #$10                                       LDD     TIC1,X
    STAA    TCTL2,X  to captures rising edge           SUBD    RISETIME
    LDAA    #$04                                       STD     PULSEWIDTH
    STAA    TFLG1,X  clear IC1F flag if set        AGN BRA     AGN
P_RISE                                             RISETIME     FDB $0000
    BRCLR   TFLG1,X $04 P_RISE                     PULSEWIDTH   FDB $0000
    LDD     TIC1,X
    STD     RISETIME
    LDAA    #$20
    STAA    TCTL2,X  to capture falling edge
    LDAA    #$04
    STAA    TFLG1,X  clear flag IC1F if set
```

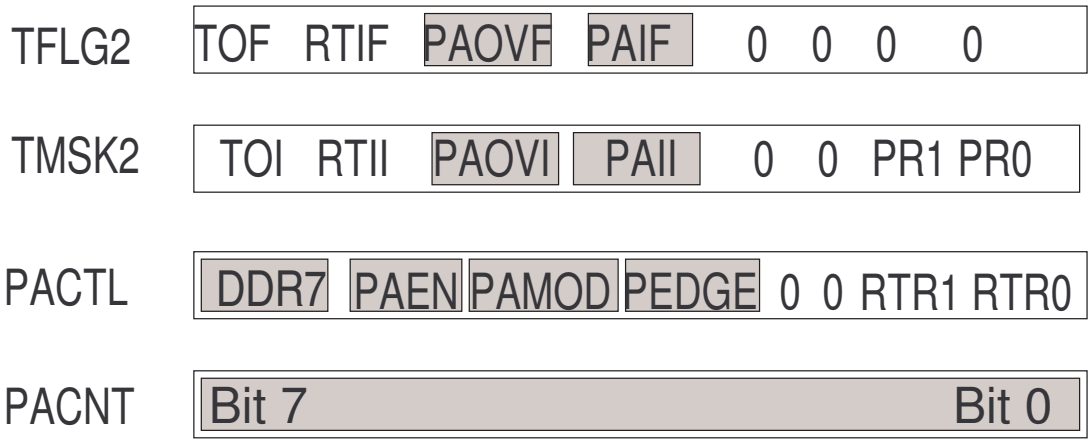
PA Event Counting mode



PA Time Accumulation mode



PA-related registers and bits



PA in counting mode

```
*counts the number of cars painted
       ORG $4000
INIT  BSET PACTL,X $40                          PAOVF_ISR
 *PAEN 1 PEDGE 0 PAMODE 0                             BCLR   TFLG2,X~$20
       LDD  #PAOVF_ISR                                BSET   TMSK2,X $20
       STD  $3FDC           *PA0 pseudovector         JSR    STOP_PAINT
       BCLR  TFLG2,X ~\$20   *clear PAOVF
       BSET TMSK2,X  \$20    *enable PAOVI             RTS
       LDAA     COUNT       *number of cars           *--------------
       NEGA                 *2's complement           COUNT  RMB    1
       STAA   PACNT,X
       CLI
       WAI                  *wait for interrupt
```

Real Time Interrupts

```
*a task is run (periodically) each second
T_COUNT   EQU  244    *(244 ticks =1s)
INITRTI
    LDX  #RTI_ISR
    STX  $3FF0       *RTI pseudo vector
    CLR  TICKS
    BCLR FLAG2 ~$40  *clear RTIF
    BSET TMKSK1 $40  *enable RTI
    CLI              *enable interrupts
    WAI
    BRA .
```

```
*interrupt sevice routine
 RTI_ISR  BCLR    FLAG2 ~$40
          LDAA     TICKS
          INCA
          STAA     TICKS
          CMPA     #T_COUNT
          BEQ      SEC
          RTI

  SEC     JSR    TASK
          RTI

  TICKS   FCB   $00
```

COP timer & clock monitor reset

- Computer operating properly (COP) timer

```
AGAIN       LDAA    #$55
            STAA    COPRST  arms the COP  clearing mechanism
*                      time critical code
*                      could be entered here between
            LDAA    #$AA
            STAA    COPRST  clears the COP timer (prior to timing out)
            BRA     AGAIN
```

- Clock monitor reset

  Reset the system if no clock is detected in a preset (RC) time

Serial Communication Interface

- Asynchronous communication with remote devices
- Interfacing standard RS232 (or EIA 232A)
  The standard defines signal levels, connectors, and pin assignment
- One transmitter many receivers
- The basic unit of information is the *character* or *data frame*
- *Baud rate* defines the number of bits (including start,stop and parity bits) per second

SCI(cont.)

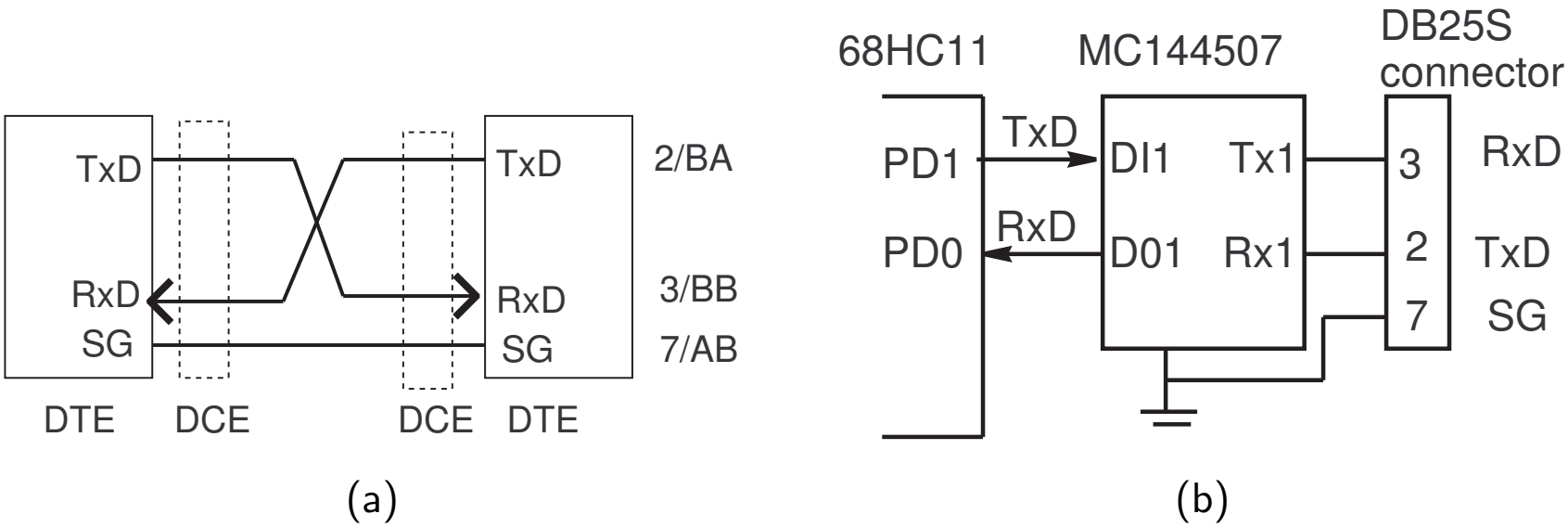
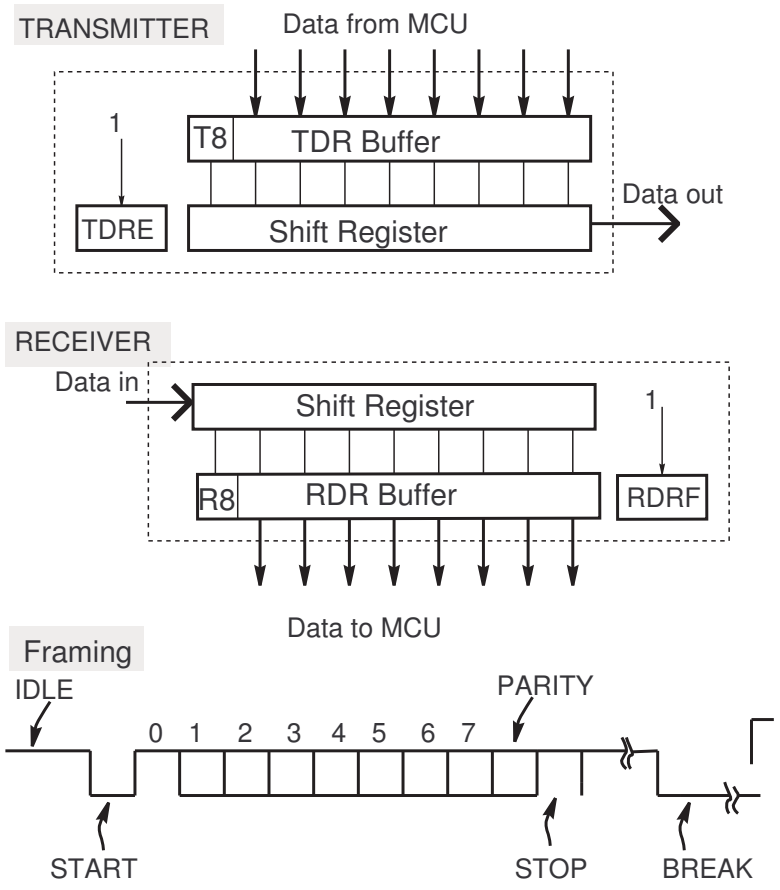
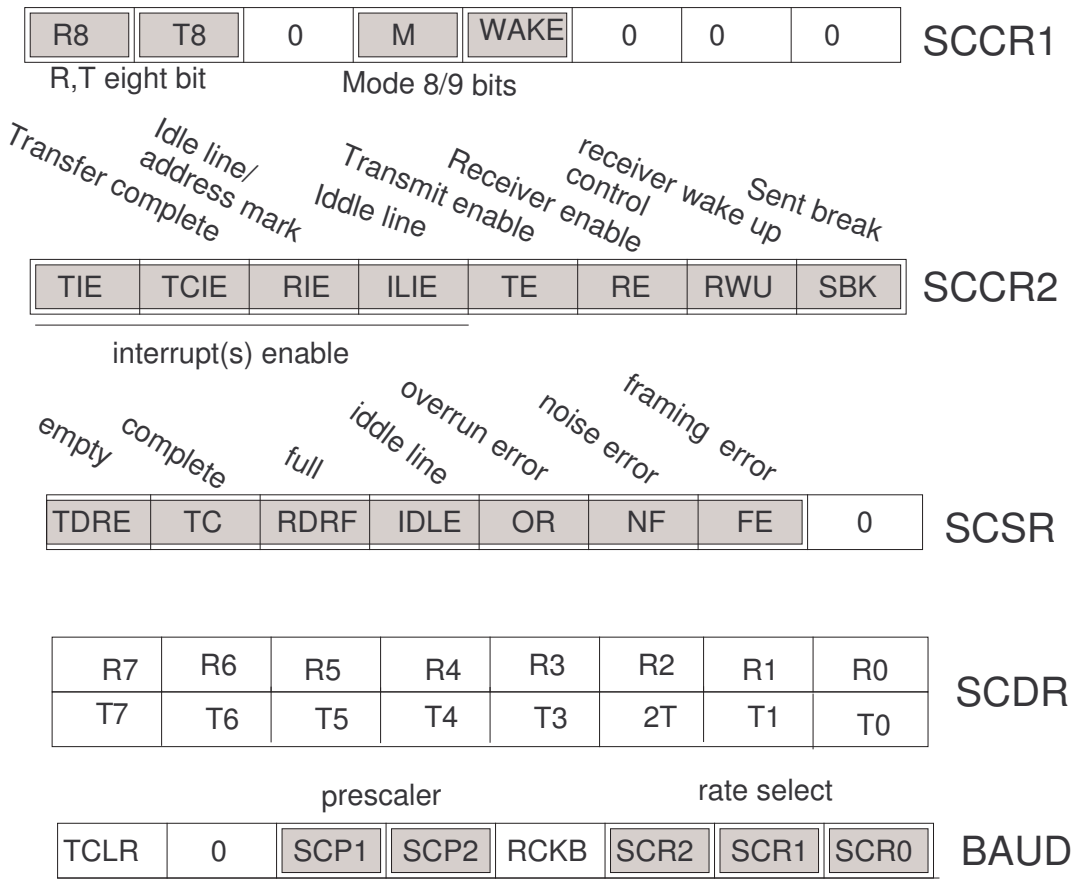


Figure 1: (a)Null MODEM and (b)RS232 DCE (MC interface)

Transmission in SCI system



SCI registers and control bits

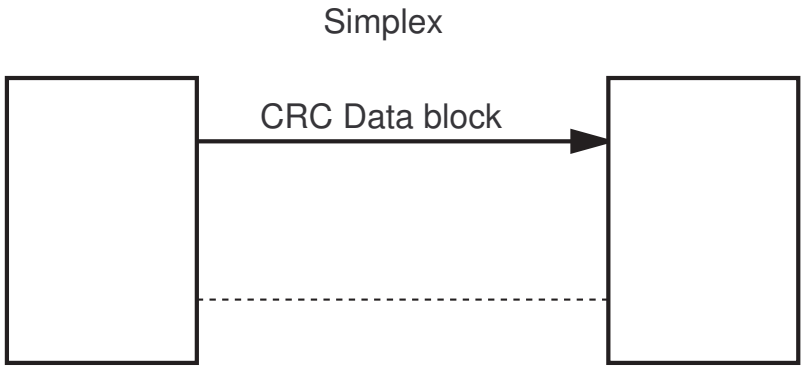


Using SCI

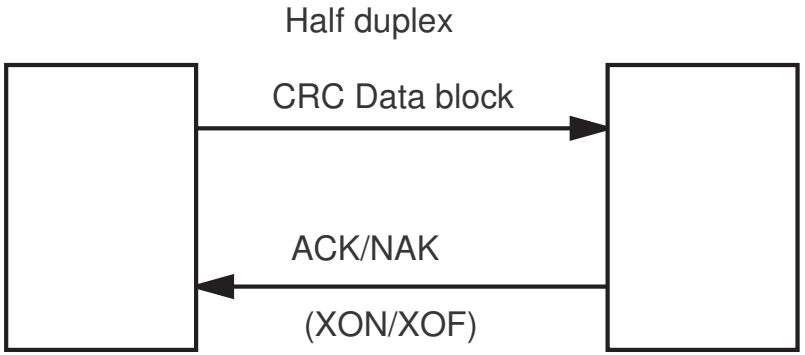
```
INITSCI LDAA  #$30    9600 baud
        STAA  BAUD    baud register
        LDAA  #$00
        STAA  SCCR1   1 start, 8 bits, 1 stop
        LDAA  #$0C
        STAA  SCCR2   transmitter receiver enable
        RTS
```

```
 OUTSCI                                  |   INSCI
    LDAB SCCR     Read status reg.       |      LDAA SCCR    Read status reg.
    BITB #$80     Check TDRE             |      ANDA  #$20   Check RDRF
    BEQ  OUTSCI   Loop until TDRE= 1     |      BEQ  INSCI   Wait for data
    ANDA #$7F     Mask parity            |      LDAA SCDAT   Read data
    STAA SCDAT    Send character         |      ANDA  #$7F   Mask parity
    RTS                                  |      RTS
```

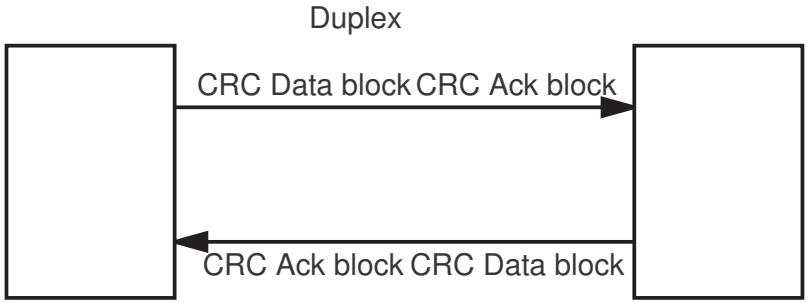
Protocols



(a)

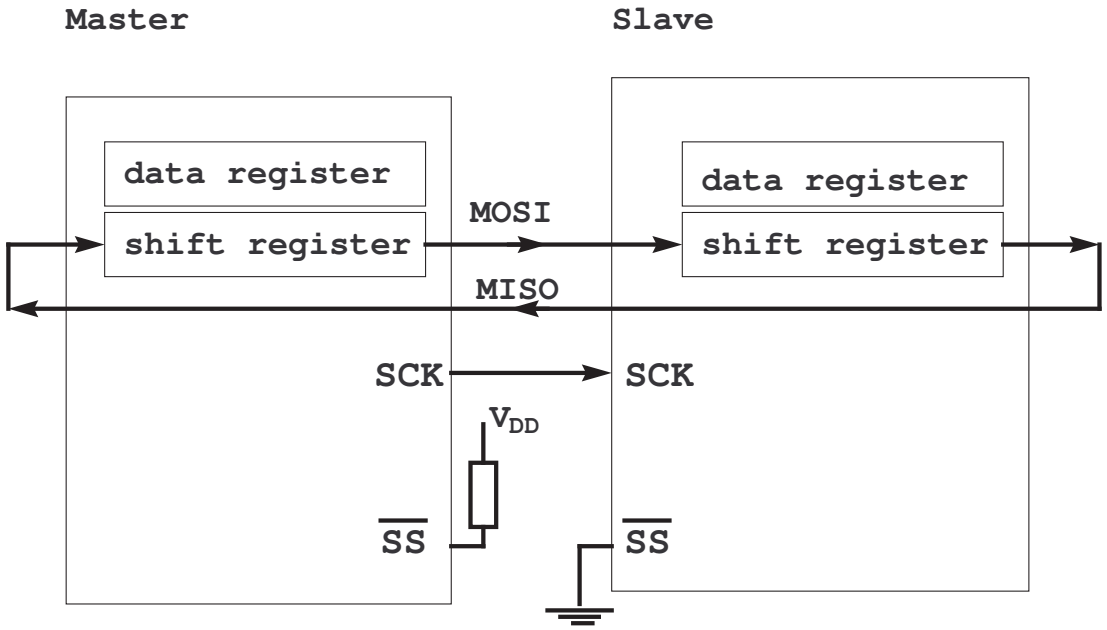


(b)



(c)

SPI basics


Master
Slave
data register
shift register
data register
shift register
MOSI
MISO
SCK
SCK
V_{DD}
$\overline{SS}$
$\overline{SS}$

SPI pins description

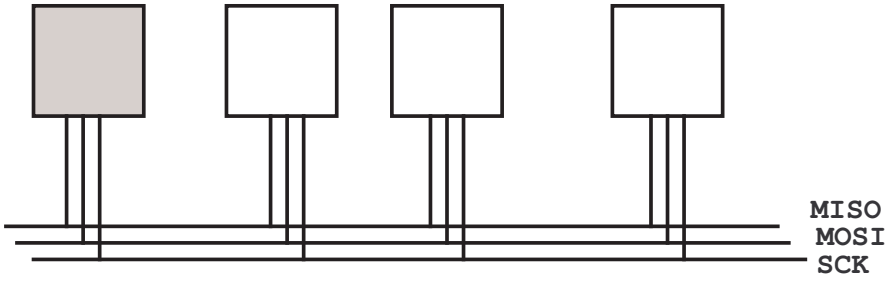
PortD /SPI signal	Master mode	Slave mode
PD2 /MISO	Input	Output
PD3 / MOSI	Output	Input
PD4 / SCK	Output	Input
PD5 / $\overline{\mathrm{SS}}$	Programmable	Input

IC that use synchronous serial interface

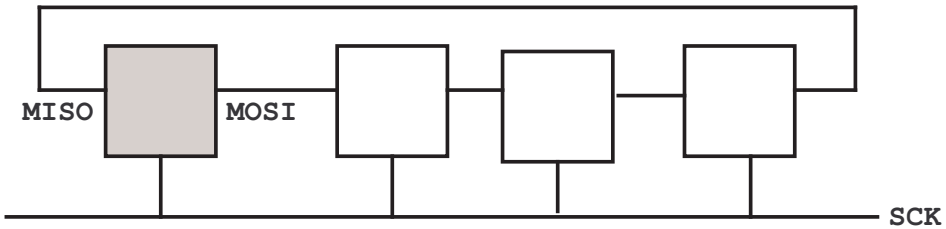
- PLL frequency synthesizers (MC145156)
- Seven-segments display decoders/drivers (MC14499)
- LCD display decoders/drivers (MC145453)
- ADC (MC145041)
- DAC (MC144110, DAC-8840)
- Shift registers (74HC589,74HC595)
- Real time clocks (MC68HC68T1)
- ISDN transceivers
- Serial RAM chip (DS1200), etc.

SPI Topologies

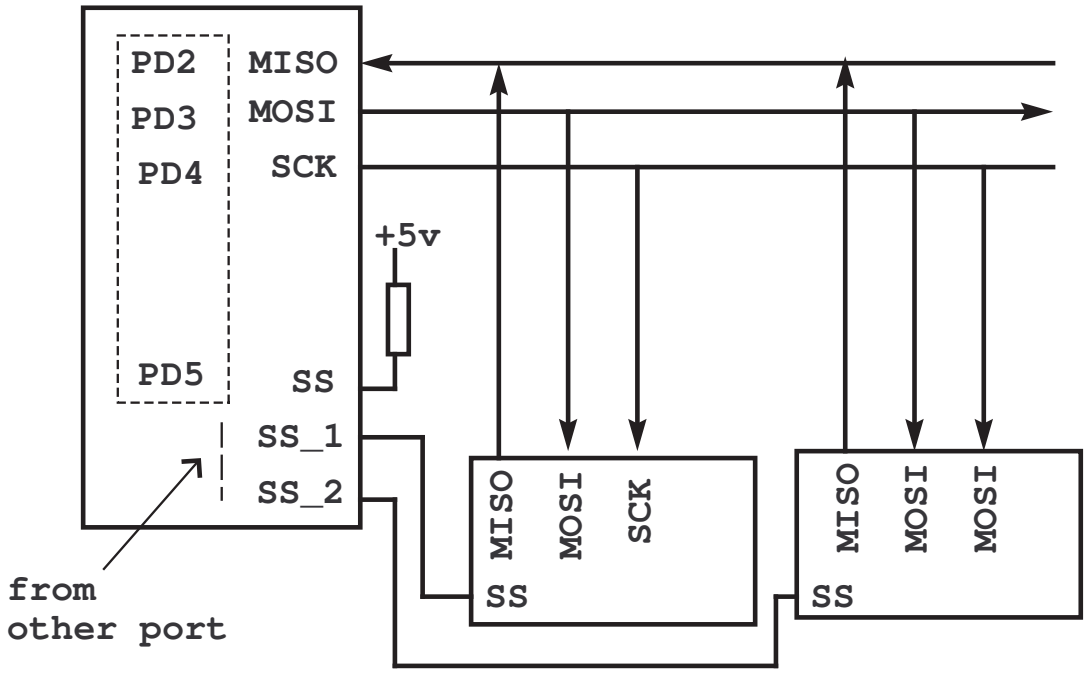
Bus topology



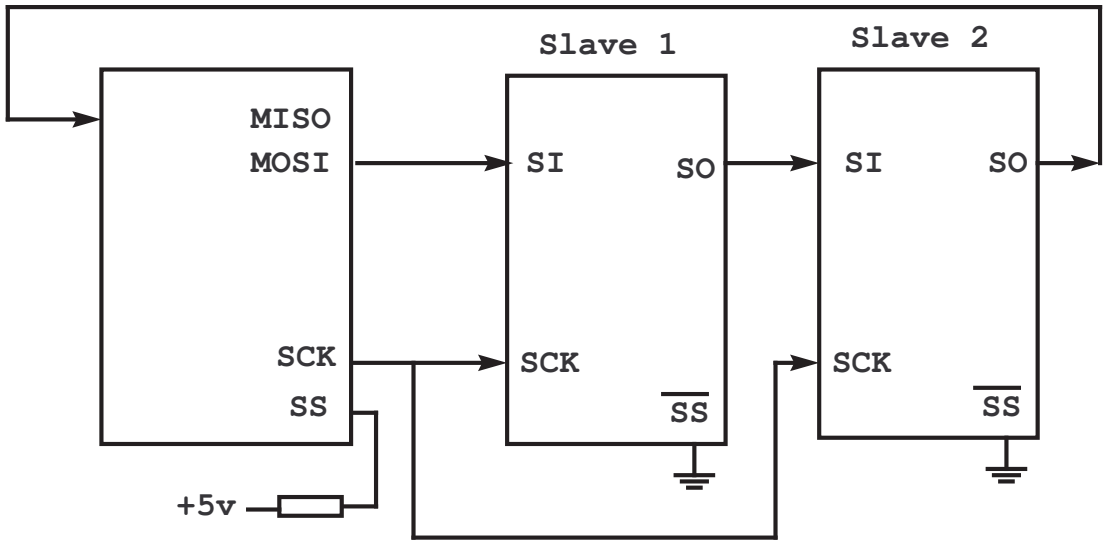
Cascade topology



SPI Bus topology



SPI Cascade topology



SPI Operations

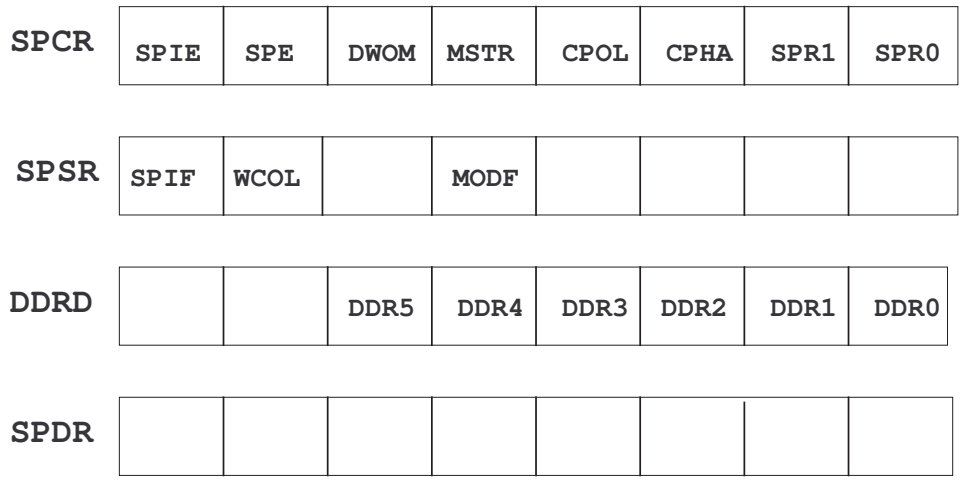
Master

1. Write a byte to data register SPDR ($\overline{\mathrm{SS}}$ previously asserted low)
2. Wait for SPIF to be set (deassert slave $\overline{\mathrm{SS}}$ high)
3. read a byte from SPDR

Slave

1. write a byte to SPDR
2. Wait for SPIF to be set
3. Reads a byte from data register SPDR

SPI Registers



SPI control and status registers

SPI Control Register

SPIE	SPI Interrupt enable (1/0)
SPE	SPI system enable (1/0)
DWOM	Port D wire-or mode (0/1) push-pull/open drain
MSTR	master/slave select (1/0)
CPOL	clock polarity active (0/1) active high/low
CPHA	CPHA equal zero/one format (0/1)
SPR1 SPR0	(E/2 E/4 /E16 E/32)

SPI Status Register

SPIF	SPI transfer complete flag; sets at one at the end of an SPI transfer [a]
WCOL	write collision flag; sets if SPDR is written while a transfer is in progress [b]
MODF	Mode fault error flag; sets if $\overline{\mathrm{SS}}$ goes active low while SPI is configured as a master[c]

[a] cleared by reading SPSR with SPIF set followed by an access of the SPDR

[b] cleared by reading SPSR with WCOL set followed by an access of SPDR

[c] cleared by reading SPSR with MODF set followed by a write to *SPCR*

Master SPI Operations: Example

Demonstrates SPI byte output and input (slave always enabled)

```
DATA EQU    0
     ORG     $4000
     LDX     #$1000
     LDAA    #$38      enable SPI outputs
     STAA    DDRD,X    data direction registers  bits 5-0
     LDAA    #$57
     STAA    SPCR,X    SPI master
*                      CPHA=1,CPOL=0,clock rate 1/32
     LDAA    DATA      get some data
    STAA     SPDR,X    and transmit it
POLL
     TST     SPSR,X    wait for transfer complete
     PBL     POLL      branch if plus <0
     LDAA    SPDR,X    gets data from the slave
                       (and clears SPIF)
DONE  BRA     DONE
```

Slave SPI Operations: Example

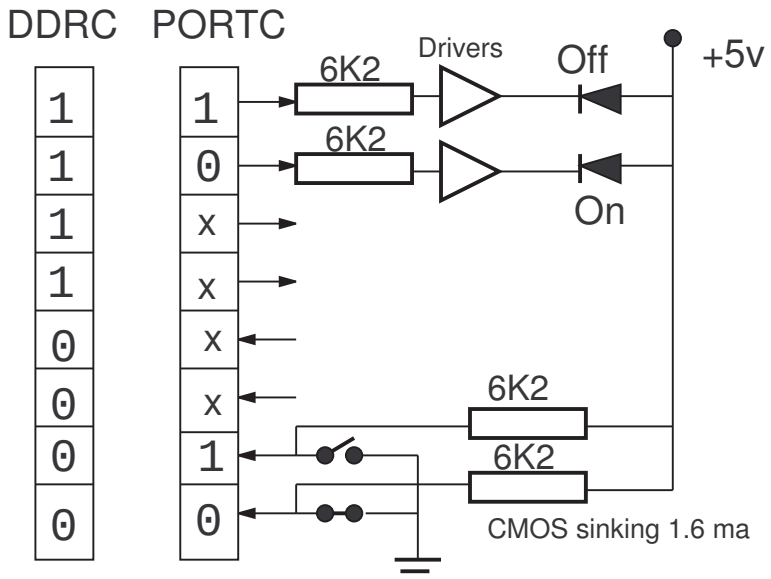
Demonstrates SPI byte output and input (slave always enabled)

```
DATA EQU    0
     ORG     $4000
     LDX     #$1000
     LDAA    #$04      Enable MISO output
     STAA    DDRD,X    others forced as inputs
     LDAA    #$47
     STAA    SPCR,X    SPI slave,CPHA=1,CPOL=0,
                       Clock rate don't care
     LDAA    DATA      Get some data
    STAA     SPDR,X    and send to data register
 POLL                  Wait for master clock to shift
                       it out and master's date in
     TST     SPSR,X    wait for transfer complete
     PBL     POLL      until master transfer complete
     LDAA    SPDR,X    get data from the slave
                       (also clears SPIF)
DONE  BRA     DONE
```

Parallel Input/Output

- General Purpose I/O
  - I/O is done by *simply* reading or writing to I/O ports
  - All five HC11 ports (A, B, C, D and E) can be used for general (simple) I/O
  - Each port has a corresponding *data register*, PORTA, PORTB,PORTC, PORTD and PORTE
  - Bidirectional ports D and C have corresponding *data direction registers* DDRC and DDRD
- Strobed and Full-Handshake I/O modes
  - Port C alternate latched register (PORTCL)
  - Parallel I/O control register PIOC
  - STRA input pin (related to PORTCL)
  - STRB output pin (related to PORTB)

General Purpose I/O



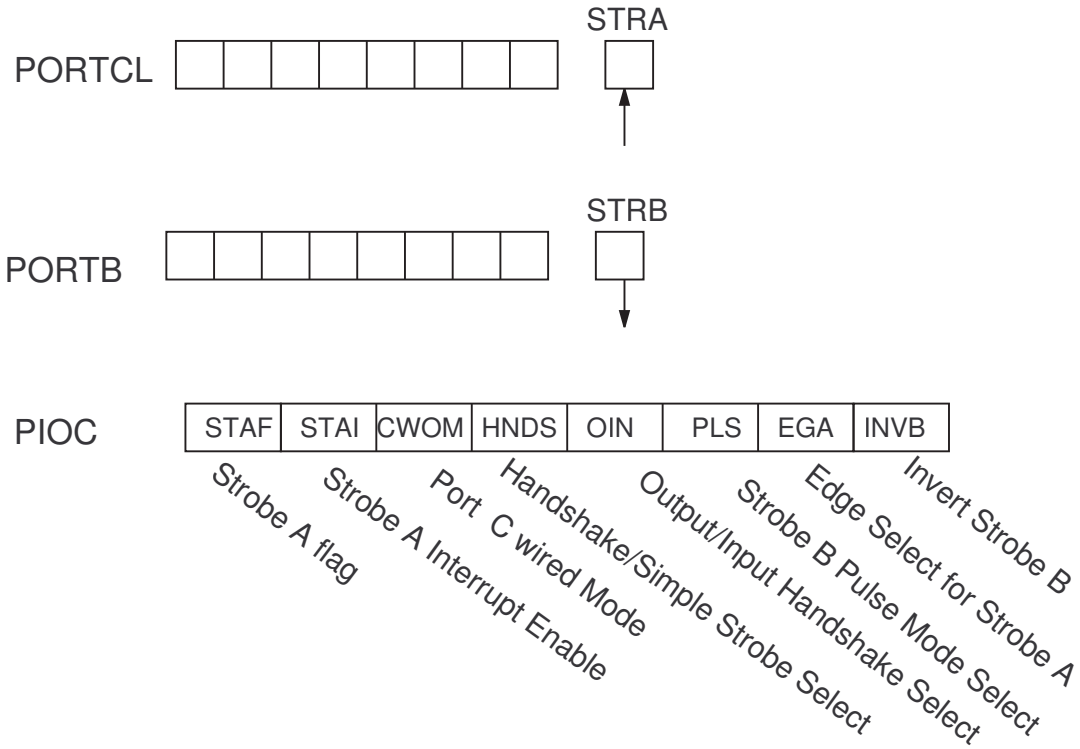
*Relevant control and data registers DDRC and PORTC

Simple (general purpose) I/O
Relevant registers

```
*Example
    LDAA    #$F0
    STAA    DDRC,X    %1111000 PC7-4 ouput, PC3-0 input
    LDAA    #$80
    STAA    PORTC,X   %1000 xxxx PC7 high, PC6 low
    LDAA    PORTC,X   value red %xxxx xx10
```

*where X points to the base of register block

Strobed I/O
Relevant registers



Strobed I/O - A sample program

*The MC receives data from peripheral(on STRA edge)
and echoes them back (STRB low for two clock cycles)[1].

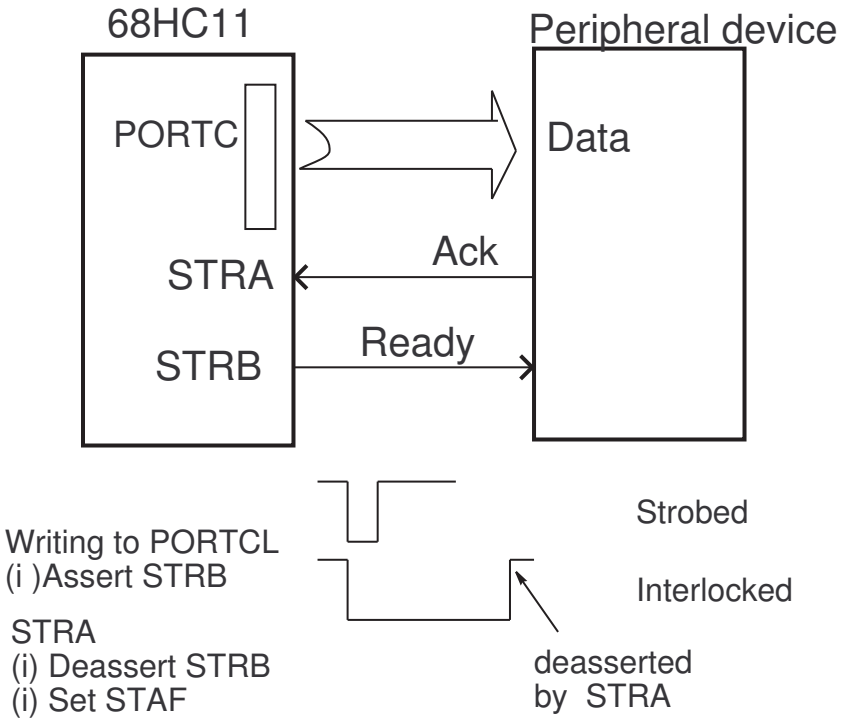
```
    ORG     $C000       start address
    JSR     SET_IRQ_VECTOR
    LDY     #DPTR       Y points to data to be received
    BSET    PIOC,X $40  enable STRA interrupt
    CLI                 clear global interrupt mask(bit I in CCR )
RPT WAI                 wait for interrupt from falling edge STRA input
    STAA    0,Y         save received data
    STAA    PORTB,X     echo back
    INY
     ...
    BRA RPT
```

[1] In non-handshake mode STRA STRA pin serves as edge-detection interrupt source

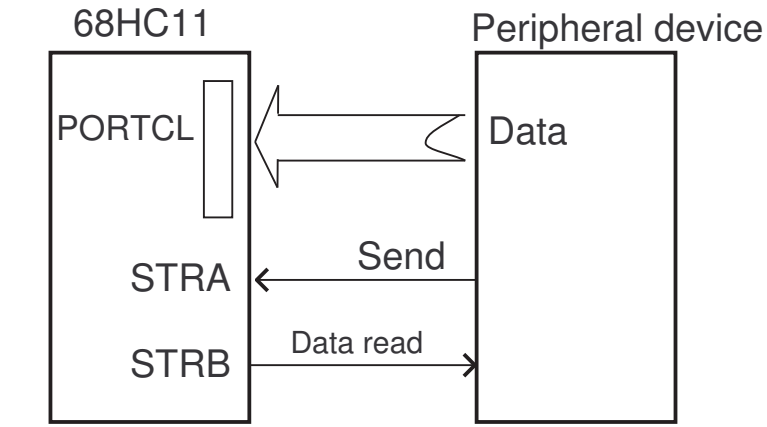
Strobed I/O - A sample program (cont.)

```
SET_IRQ_VECTOR
    LDX #SIO_ISR
    STX  $3FF2
    RTS

%Interrupt service routine
SIO_ISR LDAA    PIOC,X      the instructions pair
        LDAA    PORTCL,X    clear STAF
        RTI
```

Full Output Handshake

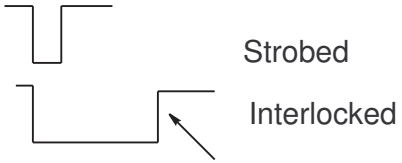


Full Input Handshake



STRA
(i)Data latched
(ii)Set STAF
(iii)Deassert STRB

Reading Port C
(i) Assert STRB



An Input Handshake Program

```
INIT_HNDS
    PSHA
*strobe mode first
*clear STAF if set
    LDAA PIOC,X
    LDAA PORTCL,X
*Port C Input
    LDAA    \#\$00
    STAA    DDRC,X
*Input Handshake Mode
        LDAA    #$10
        STAA    PIOC,X %HNDS 1
        PULA
        RTS
```

```
IN_HNDS
*Uses input handshake to read port C;
*polls for STRA transitions
  *Inputs strobed data and clear STAF
       LDAA PORTCL,X
       RTS
 *---------------------------------
 RDDATA
 *Read data from peripheral
       JSR INIT_HNDS
       JSR IN_HNDS
          ...
        BRA RPT
            RTS
```

Summary of Features

- Central Processing Unit (**CPU**)
  - Up to 5/3 Mhz (5/3V)
  - Two 8-bit and one 16 bit accumulator
  - Two 16-bit index registers
  - 16- bit stack pointer
  - (Powerful) bit-manipulation instructions
  - Six (powerful)addressing modes
  - Memory-mapped I/O and special functions
  - 16x16 integer and fractional divide
  - 8x8 multiply
  - Power-saving stop and wait modes

- EEPROM
  - On-chip byte-erasable **EEPROM**
  - No separate voltage required

- Expanded Bus memory interface
  - 64 KB addressing space
  - (Some 68HC11s) Up to 1 MB addressing space
  - Either multiplexed or non-multiplexed interfaces

- Serial Communication Interface (SCI)[2]
  - Standard mark/space non-return to zero format
  - Full duplex operation
  - Double buffering of both receiver and transmitter
  - Programmable 8-bit or 9-bit character length
  - Error detection—at 1/126 of a bit time
  - Baud rate generator with programmable baud rate
  - Idle line and address mark wakeup methods
  - Receiver framing error detection
  - Break send capability
  - Optional parity checking and generation
  - Separate transmitter, receiver and error interrupt vectors

[2] used for asynchronous communication with a terminal, computer or microcontroller network over long distance (**RS-232**)

- Serial Peripheral Interface (**SPI**[3])
  - Full duplex, three wire synchronous transfers
  - Master or slave operation
  - Bit frequency is E/2 (Master) E/1 (Slave)
  - Four programmable master bit-rates
  - Programmable clock polarity and phase
  - End of transition interrupt flag

- Lower-Power Operation
  - Low power for voltages (3–5V)
  - Phase-locked loop(**PLL**) clock synthesizer circuit[4]

- High Performance Timer
  - Free running 16 bit counter
  - Programmable prescaler
  - Overflow interrupt
  - Separate function interrupts

[3] used for synchronous communication with peripheral devices over short distances (usually on a single PCB) such as shift register, serial EEPROM, LCD or ADC subsystems

[4] PLL reduces the clock speed

- Additional Features
  - Multiple *input capture* and *output compare* functions
  - Real-time interrupts
  - Computer Operating Properly Watchdog (**COP**)
  - Pulse accumulator for external event counting or gated accumulations
  - Optional **PWM**[5]
  - Optional event counter system for advance timing operation

- Analog to digital Converter (**ADC**)
  - Linear successive approximation
  - 8-bit or 10-bit resolution
  - Single or continuous conversion modes
  - Multiple result registers
  - Selectable ADC clock
  - Analog mutiplexer[6]

[5] offering up to six channels and up to 16-bit PWM

[6] allows variable number of channels with a single ADC

- Pulse-width modulation
  - Up to six PWM channels [7]
  - Software selectable duty cycles (from 0–100%)

[7] which can create continuous waveforms with programmable rates