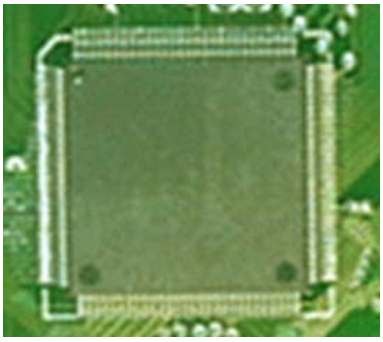
Embedded Microcontroller Units

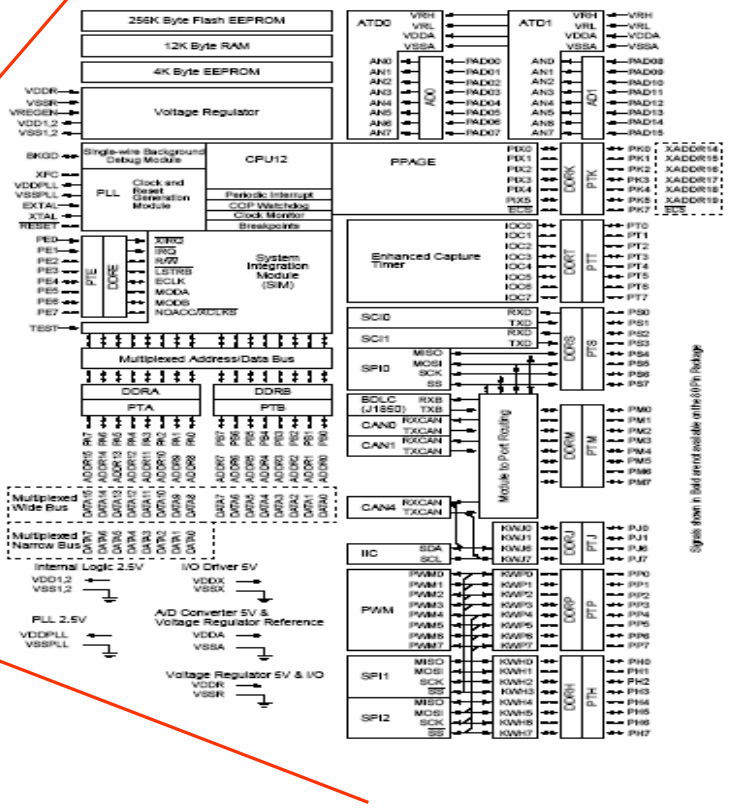
- Example: Freescale microcontroller's HCS12
- Lab systems
- A Real time clock for HCS12
- Primary memory disposition
- Serial communication with HCS12
- AD conversion with HCS12
- PWM generation with HCS12

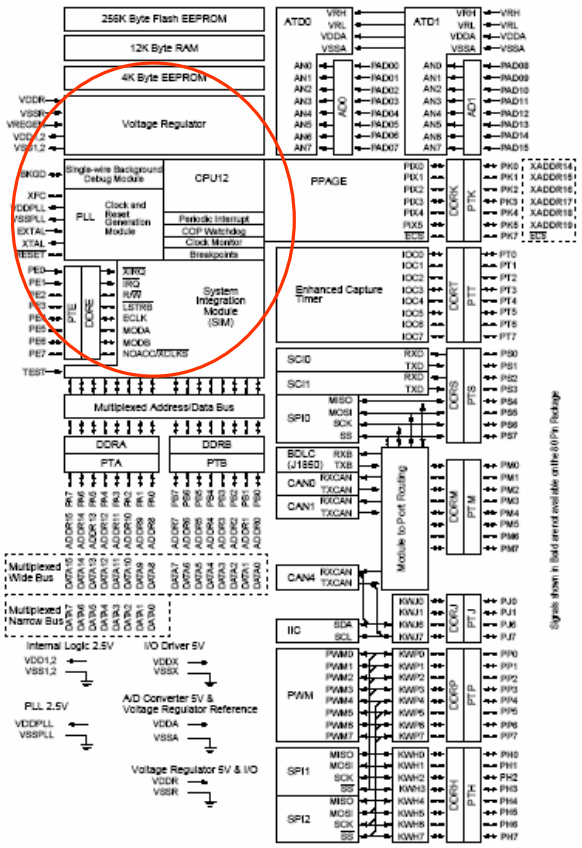
Freescale HCS12

- HCS12 memory map and utilization
- CPU control (core), clocks and timers
- Random Access Memory
  - RWM, FLASH, EEPROM
- Peripherals
  - Parallel Input/Output:
  - serial
  - AD
  - PWM

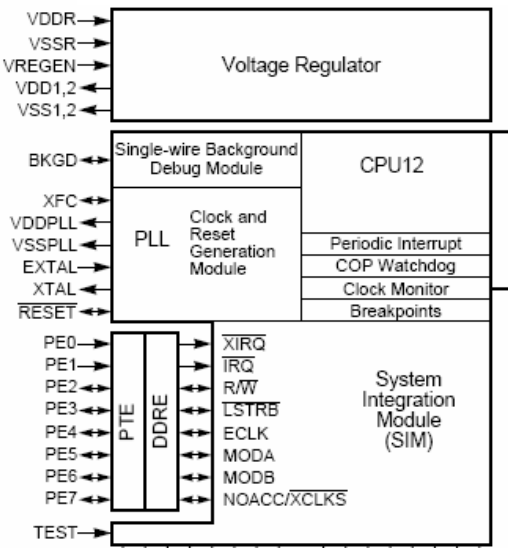


HCS12DG256, block diagram





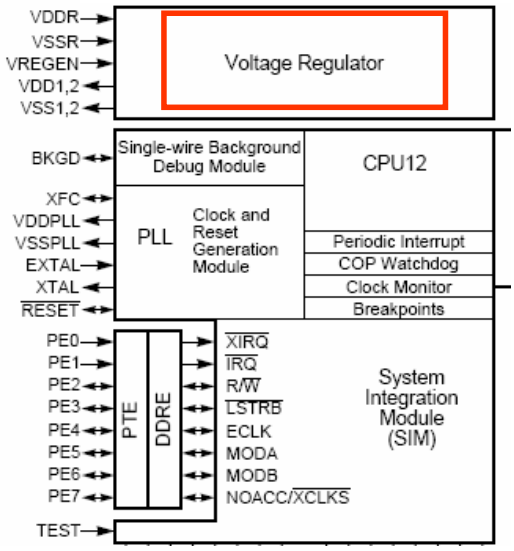
HCS12DG256, "core"



HCS12DG256, ”core”

Voltage Regulator

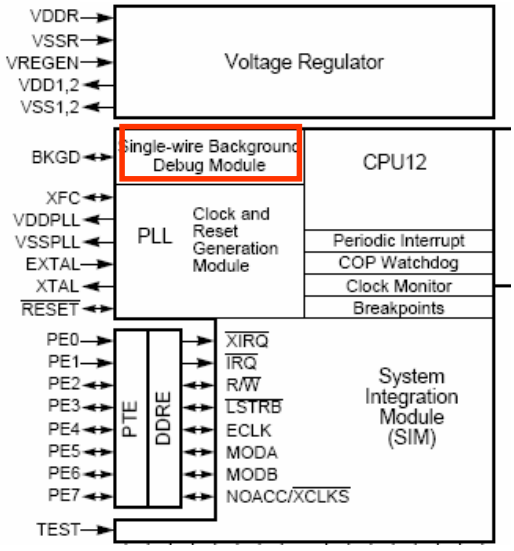
- The chip is normally supplied with a single power supply.
- The need for different supplies is accomodated by on-chip circuits.
- In particular, supply for the built in AD converter can be supplied.



HCS12DG256, ”core”

Debug facilities

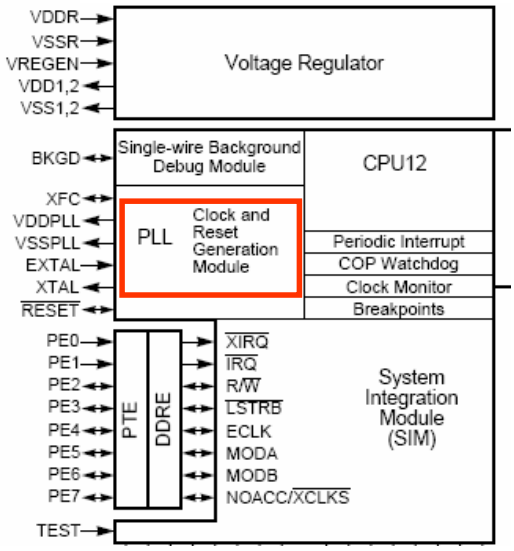
- A single pin can be connected to an external device and facilitate debugging support, such as:
  - read/write from/to memory
  - single step the program
  - run the program
  - and so on...



HCS12DG256, ”core”

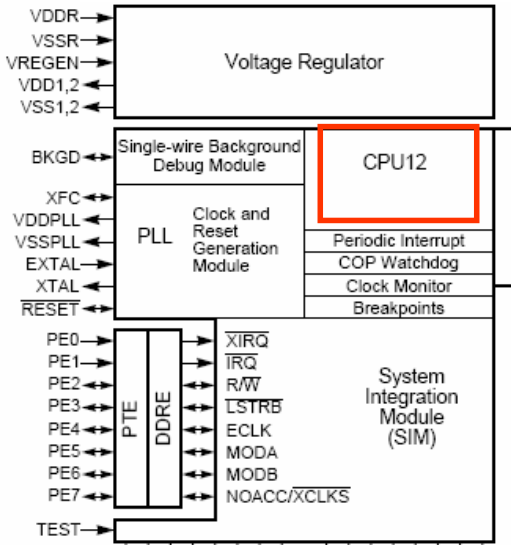
Clock frequencies

- A single external crystal provides clock oscillator input to the chip
- Higher frequencies are generated on chip



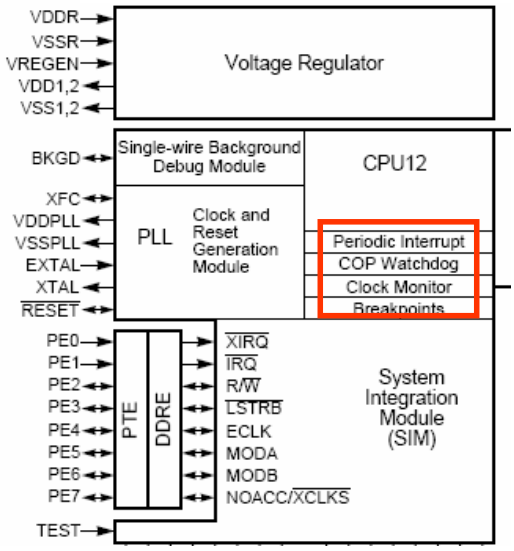
HCS12DG256, ”core”

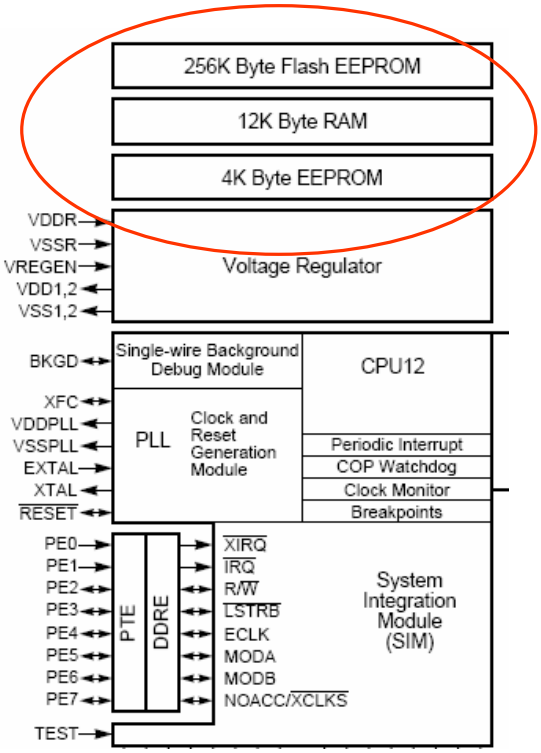
Central Processing unit CPU12



HCS12DG256, ”core”

Real time support





Primary Memory

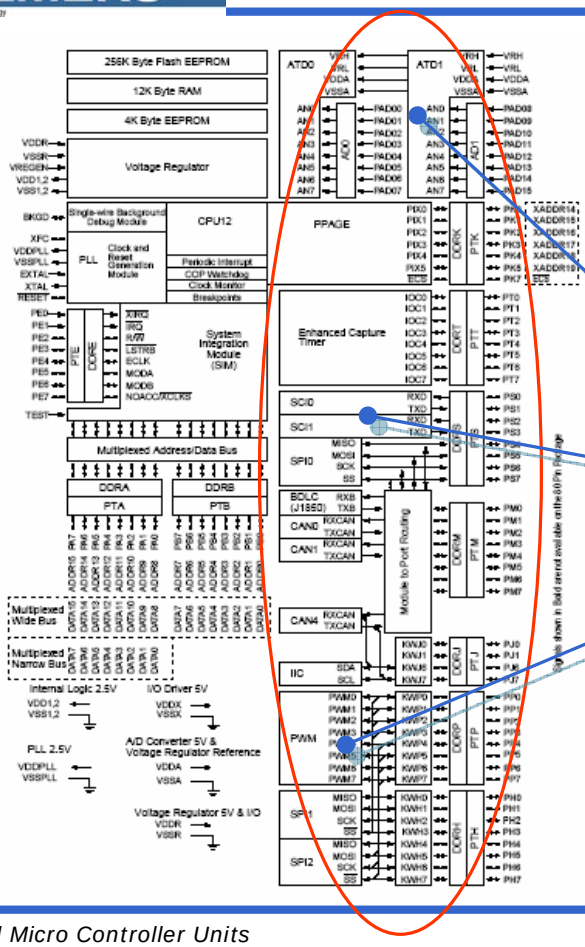
Non Volatile memory

- Banked FLASH memory (16 k Pages) 48 kB in memory map.
- Max 4 kB Electric Erasable PROM

Volatile memory

- 12 kB RAM

RAM and EEPROM are relocatable



Peripherals in HCS12DG256

AD – Analog to Digital conversion

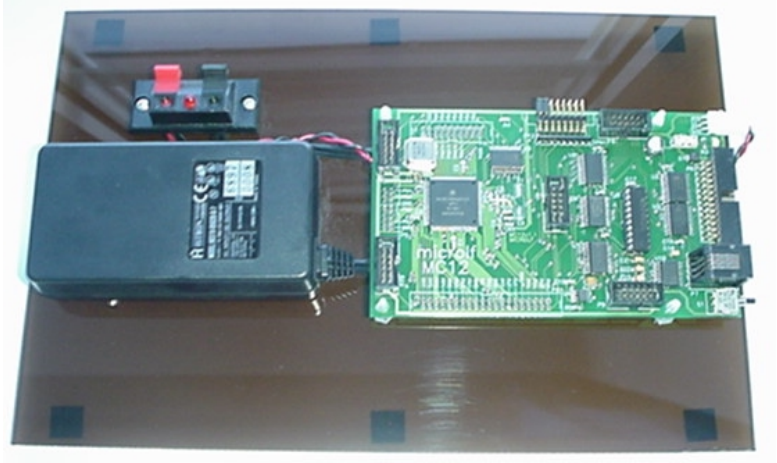
SCI – Serial Asynchronous communication

PWM – Pulse Width Modulation

Etc…

Lab system: MC12

- HCS12 microcontroller
- Debug with built in software debugger
- 8 MHz crystal
- Add on cards with different types of peripherals

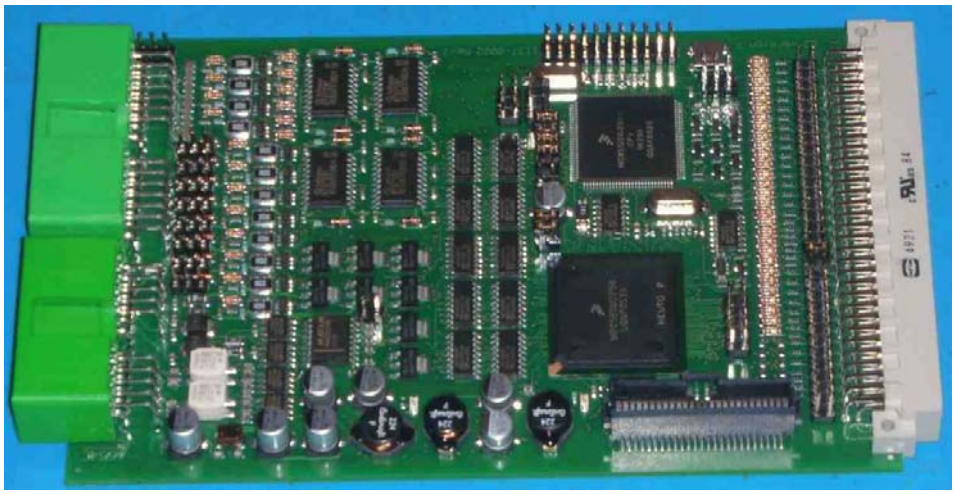


Lab system: GAST G1

- HCS12 microcontroller
- Debug with built in software debugger
- 8 MHz crystal
- Back plane expansion

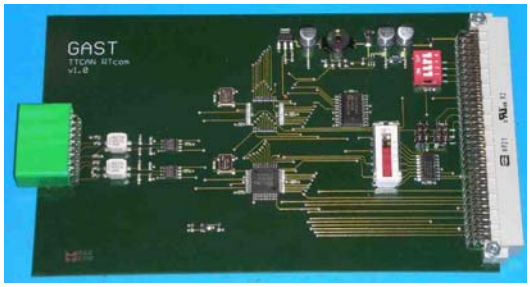
Lab system: GAST G2

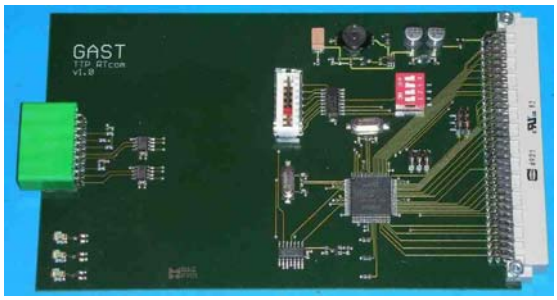
- Power PC 565 microcontroller
- HCS12 microcontroller
- Back plane expansion

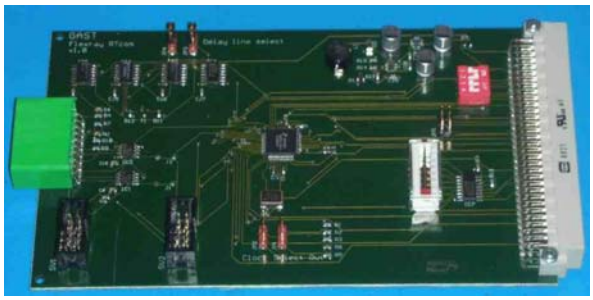


Lab system: GAST Real Time COMMunication boards

- Time Triggered CAN (TTCAN)
- Time Triggered Protocol (TTP/C)
- Flexray

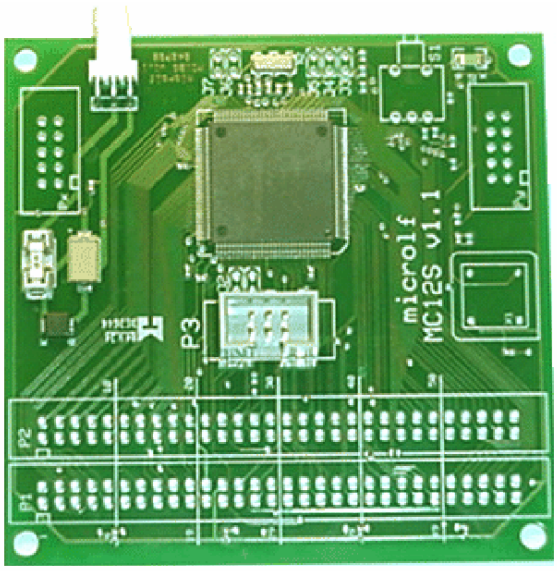




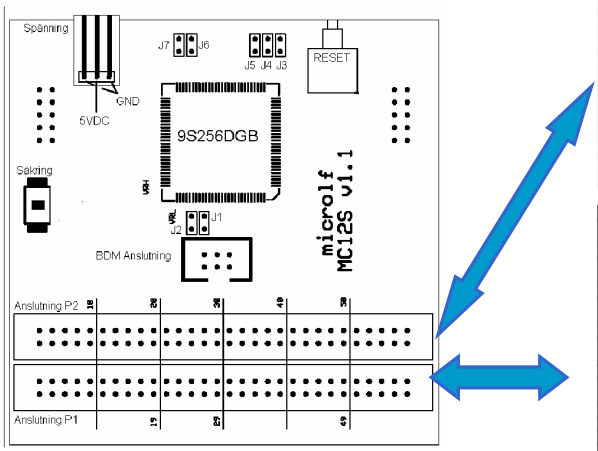


Lab system: MC12S

- HCS12 microcontroller
- Debug through BDM
- 8 MHz crystal
- Input/Output pins in two connectors



MC12S, connections



1	GND	PH7	2
3	PH6	PH5	4
5	PH4	PH3	6
7	PH2	PH1	8
9	PH0	VCC	**10**
11	GND	NC	12
13	VRL	PAD0	14
15	PAD1	PAD2	16
17	PAD3	PAD4	18
19	PAD5	PAD6	**20**
21	PAD7	VRH	22
23	NC	VCC	24
25	GND	PJ7	26
27	PJ6	PJ1	28
29	PJ0	VCC	**30**
31	GND	PM7	32
33	PM6	PM5	34
35	PM4	PM3	36
37	PM2	PM1	38
39	PM0	VCC	**40**
41	GND	PK7	42
43	NC	PK5	44
45	PK4	PK3	46
47	PK2	PK1	48
49	PK0	VCC	**50**
51	GND	NOACC/PE7	52
53	MODB/PE6	MODA/PE5	54
55	ECLK/PE4	LSTRB/PE3	56
57	R/W/PE2	IRQ/PE1	58
59	XIRQ/PE0	VCC	60

1	GND	PA0	2
3	PA1	PA2	4
5	PA3	PA4	6
7	PA5	PA6	8
9	PA7	VCC	**10**
11	VRL	PAD8	12
13	PAD9	PAD10	14
15	PAD11	PAD12	16
17	PAD13	PAD14	18
19	PAD15	VRH	**20**
21	GND	PS0	22
23	PS1	PS2	24
25	PS3	PS4	26
27	PS5	PS6	28
29	PS7	VCC	**30**
31	GND	PP7	32
33	PP6	PP5	34
35	PP4	PP3	36
37	PP2	PP1	38
39	PP0	VCC	**40**
41	GND	PT0	42
43	PT1	PT2	44
45	PT3	PT4	46
47	PT5	PT6	48
49	PT7	VCC	**50**
51	GND	PB0	52
53	PB1	PB2	54
55	PB3	PB4	56
57	PB5	PB6	58
59	PB7	VCC	60

Clock generation in HCS12

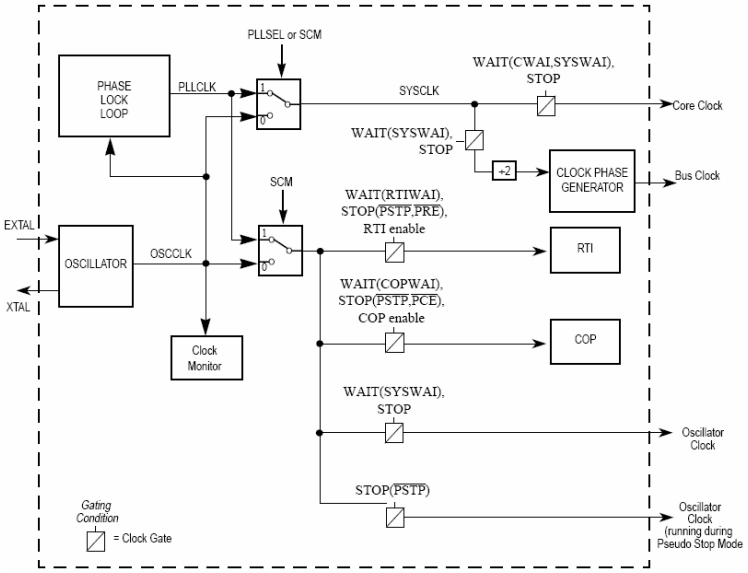
HCS12 has programmable bus speed (max 25 MHz). Controlled by CRG (Clock Reset Generator) modul.

Address Offset	Use	Access
$_00	CRG Synthesizer Register (SYNR)	R/W
$_01	CRG Reference Divider Register (REFDV)	R/W
$_02	CRG Test Flags Register (CTFLG)[1]	R/W
$_03	CRG Flags Register (CRGFLG)	R/W
$_04	CRG Interrupt Enable Register (CRGINT)	R/W
$_05	CRG Clock Select Register (CLKSEL)	R/W
$_06	CRG PLL Control Register (PLLCTL)	R/W
$_07	CRG RTI Control Register (RTICTL)	R/W
$_08	CRG COP Control Register (COPCTL)	R/W
$_09	CRG Force and Bypass Test Register (FORBYP)[2]	R/W
$_0A	CRG Test Control Register (CTCTL)[3]	R/W
$_0B	CRG COP Arm/Timer Reset (ARMCOP)	R/W

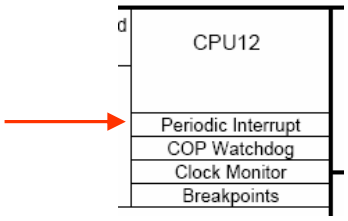
NOTES:
1. CTFLG is intended for factory test purposes only.
2. FORBYP is intended for factory test purposes only.
3. CTCTL is intended for factory test purposes only.

$$PLLCLK = 2 \times OSCCLK \times \frac{(SYNR+1)}{(REFDV+1)}$$

BusClock (E) = PLLCLK/2



Real time clock with HCS12



”Address Offset” = $34 in DG256

Address Offset	Use	Access
$_00	CRG Synthesizer Register (SYNR)	R/W
$_01	CRG Reference Divider Register (REFDV)	R/W
$_02	CRG Test Flags Register (CTFLG)[1]	R/W
$_03	CRG Flags Register (CRGFLG)	R/W
$_04	CRG Interrupt Enable Register (CRGINT)	R/W
$_05	CRG Clock Select Register (CLKSEL)	R/W
$_06	CRG PLL Control Register (PLLCTL)	R/W
$_07	CRG RTI Control Register (RTICTL)	R/W
$_08	CRG COP Control Register (COPCTL)	R/W
$_09	CRG Force and Bypass Test Register (FORBYP)[2]	R/W
$_0A	CRG Test Control Register (CTCTL)[3]	R/W
$_0B	CRG COP Arm/Timer Reset (ARMCOP)	R/W

NOTES:
1. CTFLG is intended for factory test purposes only.
2. FORBYP is intended for factory test purposes only.
3. CTCTL is intended for factory test purposes only.

Three registers are used to control the clock.

Real time clock with HCS12

CRGINT (adress $38)
Used to enable interrupts

7	6	5	4	3	2	1	0
RTIE	0	0	LOCKIE	0	0	SCMIE	0

•RTIE: Aktivera avbrott från RTI-funktionen. Denna bit måste sättas till 1 för att avbrott ska genereras.
•LOCKIE,SCMIE, rarely used. Should be 0.

RTICTL (adress $3B)
Used to specify a time base for the clock.

7	6	5	4	3	2	1	0
0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0

System timebase, e.g. 125 ns (8 MHz) is multiplied by a number specified by these bits (see below).

Real time clock with HCS12, approx: 1 ms interval

RTR [3:0]	RTR[6:4]							
	000 (OFF)	001	010	011	100	101	110	111
0000	OFF	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
0001	OFF	$2x2^{10}$	$2x2^{11}$	$2x2^{12}$	$2x2^{13}$	$2x2^{14}$	$2x2^{15}$	$2x2^{16}$
0010	OFF	$3x2^{10}$	$3x2^{11}$	$3x2^{12}$	$3x2^{13}$	$3x2^{14}$	$3x2^{15}$	$3x2^{16}$
0011	OFF	$4x2^{10}$	$4x2^{11}$	$4x2^{12}$	$4x2^{13}$	$4x2^{14}$	$4x2^{15}$	$4x2^{16}$
0100	OFF	$5x2^{10}$	$5x2^{11}$	$5x2^{12}$	$5x2^{13}$	$5x2^{14}$	$5x2^{15}$	$5x2^{16}$
0101	OFF	$6x2^{10}$	$6x2^{11}$	$6x2^{12}$	$6x2^{13}$	$6x2^{14}$	$6x2^{15}$	$6x2^{16}$
0110	OFF	$7x2^{10}$	$7x2^{11}$	$7x2^{12}$	$7x2^{13}$	$7x2^{14}$	$7x2^{15}$	$7x2^{16}$
0111	OFF	**$8x2^{10}$**	$8x2^{11}$	$8x2^{12}$	$8x2^{13}$	$8x2^{14}$	$8x2^{15}$	$8x2^{16}$
1000	OFF	$9x2^{10}$	$9x2^{11}$	$9x2^{12}$	$9x2^{13}$	$9x2^{14}$	$9x2^{15}$	$9x2^{16}$
1001	OFF	$10x2^{10}$	$10x2^{11}$	$10x2^{12}$	$10x2^{13}$	$10x2^{14}$	$10x2^{15}$	$10x2^{16}$
1010	OFF	$11x2^{10}$	$11x2^{11}$	$11x2^{12}$	$11x2^{13}$	$11x2^{14}$	$11x2^{15}$	$11x2^{16}$
1011	OFF	$12x2^{10}$	$12x2^{11}$	$12x2^{12}$	$12x2^{13}$	$12x2^{14}$	$12x2^{15}$	$12x2^{16}$
1100	OFF	$13x2^{10}$	$13x2^{11}$	$13x2^{12}$	$13x2^{13}$	$13x2^{14}$	$13x2^{15}$	$13x2^{16}$
1101	OFF	$14x2^{10}$	$14x2^{11}$	$14x2^{12}$	$14x2^{13}$	$14x2^{14}$	$14x2^{15}$	$14x2^{16}$
1110	OFF	$15x2^{10}$	$15x2^{11}$	$15x2^{12}$	$15x2^{13}$	$15x2^{14}$	$15x2^{15}$	$15x2^{16}$
1111	OFF	$16x2^{10}$	$16x2^{11}$	$16x2^{12}$	$16x2^{13}$	$16x2^{14}$	$16x2^{15}$	$16x2^{16}$

Real time clock with HCS12, interrupt handling

CRGFLG (adress $37)
Status bits, represents timer status.

7	6	5	4	3	2	1	0
RTIF	PORF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM

- RTIF: Set to 1 by timer when a time interval has elapsed. Interrupt request is cleared by software writing 1 to this bit. (!)
- Other bits, not used here, should be cleared.

Real time clock with HCS12, sample program

```
__interrupt void timer_irq( void )
{
        /* At interrupt,  clear interrupt request */
         *(unsigned char *) 0x0037 |= 0x80;
}
```

```
void    timer_init( void )
{
        /* Setup IRQ vector (address to handler) */
        *(unsigned short *) 0x3FF0 = (unsigned short) timer_irq;
        /* Enable RTI interrupts */
        *(unsigned char *) 0x0038 = 0x80;
        /* Time base for interrupts */
        *(unsigned char *) 0x003B = 0x17;
        /* Clear CPU I-flag, i.e. enable interrups */
        __asm(" cli");
}
```


”Watchdog” with HCS12

Address Offset	Use	Access
$_00	CRG Synthesizer Register (SYNR)	R/W
$_01	CRG Reference Divider Register (REFDV)	R/W
$_02	CRG Test Flags Register (CTFLG)[1]	R/W
$_03	CRG Flags Register (CRGFLG)	R/W
$_04	CRG Interrupt Enable Register (CRGINT)	R/W
$_05	CRG Clock Select Register (CLKSEL)	R/W
$_06	CRG PLL Control Register (PLLCTL)	R/W
$_07	CRG RTI Control Register (RTICTL)	R/W
$_08	CRG COP Control Register (COPCTL)	R/W
$_09	CRG Force and Bypass Test Register (FORBYP)[2]	R/W
$_0A	CRG Test Control Register (CTCTL)[3]	R/W
$_0B	CRG COP Arm/Timer Reset (ARMCOP)	R/W

NOTES:
1. CTFLG is intended for factory test purposes only.
2. FORBYP is intended for factory test purposes only.
3. CTCTL is intended for factory test purposes only.

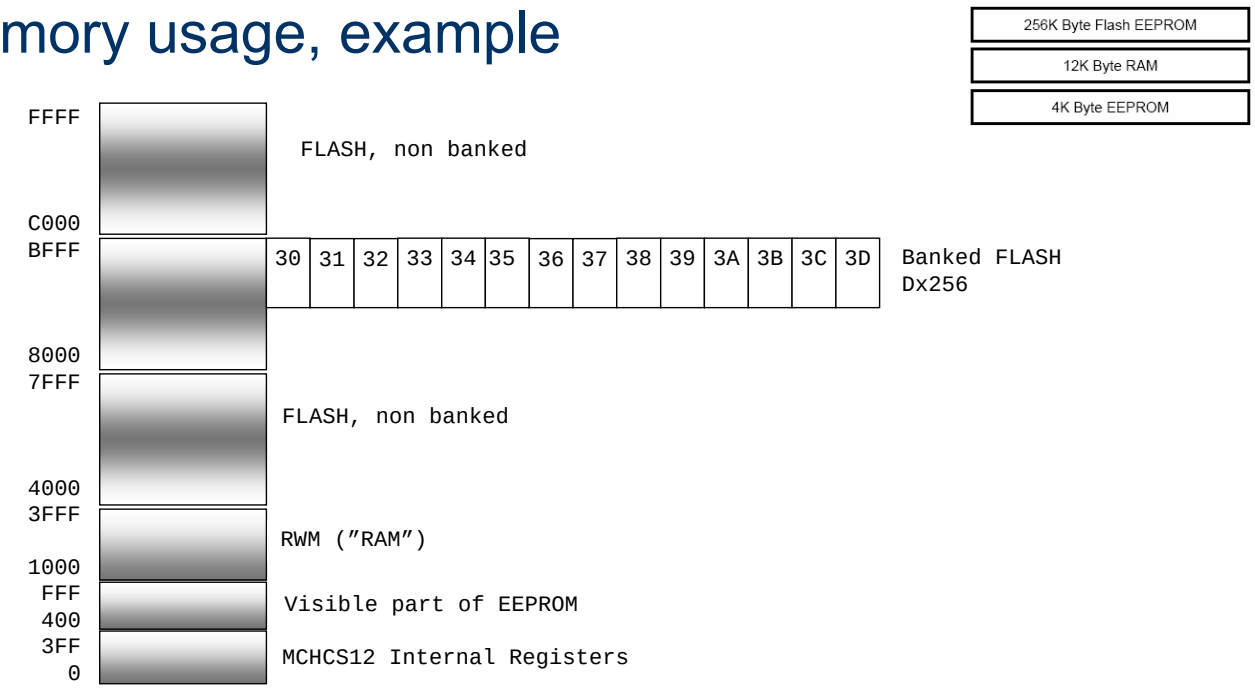
COP – Computer Operating Properly

Separate timer.

Used to alarm on severe application program errors.

Application should regulary write to the ARMCOP register,otherwise ”COP Fail” is generated.

Memory usage, example



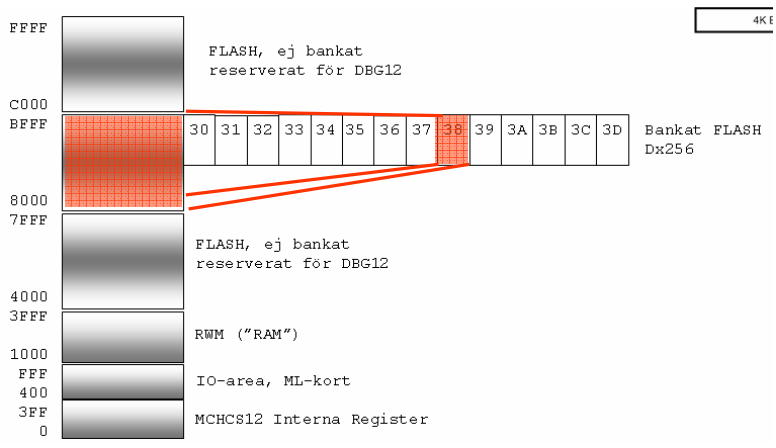
Banked memory

FFFF C000
BFFF **8000**
7FFF 4000
3FFF 0000

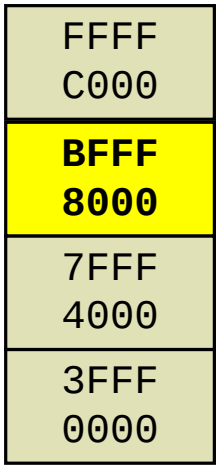
A bank is selected through a register ”PAGE” (adress $30):

EXAMPLE:

```
movb #0x38,0x30
```



Banked memory



Special instructions:

```
call  sub:
      push page
      push address
      jmp sub

rtc   return from far call:
      pop address
      pop  page
      jmp  PC
```


Banked memory, EXAMPLE assembly language

Note that address is divided in two parts for the”CALL”-instruction

```
*
*         FARCALL.S12
*
*         Example: using ”paged” memory

    ORG   $1000
    NOP

    CALL  (farbank&0xFFFF),((farbank>>16)&0xFF)  -- 308000
    NOP
*   ---------------------------

    ORG   $308000
farbank:
    LDAB  #$30
    RTC
```

RWM

FLASH

Banked memory, EXAMPLE 'C'- language

```
__farseg far_segment void main2( void )
{
    /* appliction */
}

void init( void )
{
    /* initialisation, interrupt
handling etc */
}

void main( void )
{

    /* inits in "near segment code" */
    init();
    /* application in "far memory" */
    main2();
}
```

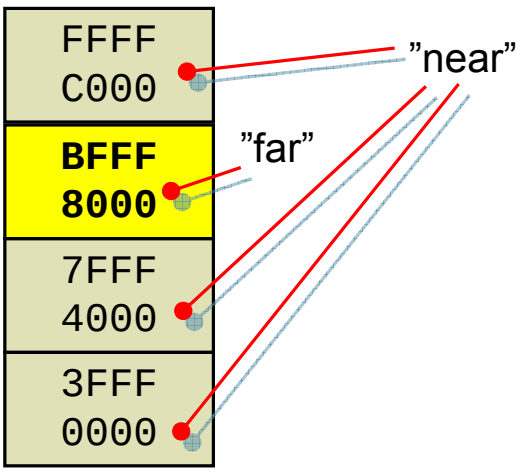
FLASH

RWM

```
...
group( rx , far_group )
{
    far_segment
}
...

layout
{
    0x1000,0x3C80 <= test_group,
    0x3F80,0x3FFF <= interrupt_vectors,
    0x308000,0x30BFFF <= far_group
}
```

Script for linkage…

Banked memory, rules of thumb…



- "startup"-has to be placed in "near" segment.
- Interrupt handlers has to be placed in "near" segment.
- Functions must have appropriate prototype declarations.
- "far" segment calls has increased overhead. Frequently used functions should therefore be placed in "near" segment.

SCI – Serial Communication Interface

Two identical devices:

SCI 0 = Offset 0xC8

SCI1 = Offset 0xD0

Offset	Use	Access
$_0	SCI Baud Rate Register High (SCIBDH)	Read/Write
$_1	SCI Baud Rate Register Low (SCIBDL)	Read/Write
$_2	SCI Control Register1 (SCICR1)	Read/Write
$_3	SCI Control Register 2 (SCICR2)	Read/Write
$_4	SCI Status Register 1 (SCISR1)}	Read
$_5	SCI Status Register 2(SCISR2)	Read/Write
$_6	SCI Data Register High (SCIDRH)	Read/Write
$_7	SCI Data Register Low (SCIDRL)	Read/Write

BaudRate = BusClock/(16 x BR)

BusClock = 8×10^6 , $1 \leq BR \leq 8192$

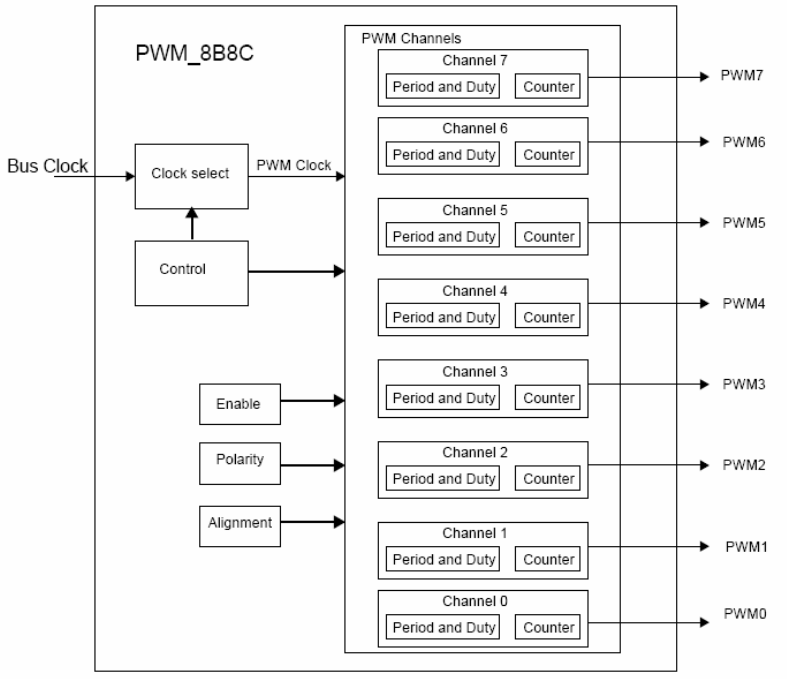
SCI – Initialization example, 9600 Baud

```
typedef struct sSCI{
        volatile unsigned short scibd;
        volatile unsigned char scicr1;
        volatile unsigned char scicr2;
        volatile unsigned char scisr1;
        volatile unsigned char scisr2;
        volatile unsigned char scidrh;
        volatile unsigned char scidrl;
}SCI, PSCI*;

#define SCI0_BASE       0x00C8
#define BAUD            8000000/(16*9600)
...
/* init SCI */
PSCI    sci = (PSCI) SCI0_BASE;
sci->scibd  = BAUD;
sci->scicr1 = 0;
sci->scicr2 = 0xC;
...
```

SCI – Input and output

```
/* send a character through SCI0 */
void    _outchar( int c )
{
        /* wait for TDRE==1 */
        while( ( sci->scisr1  & 0x80 )== 0) {};
        /* send the character */
        sci->scidrl  = (unsigned char) c;
}

/* receive a character through SCI0 */
 int    _inchar( void )
{
        /* wait for character, RDRF==1  */
        while( ! ( sci->scisr1  & 0x20 ) ) {};

        /* return the character  */
        return (int) ( sci->scidrl );
}
```

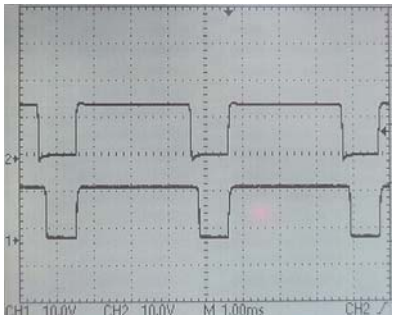

PWM – Pulse Width Modulation



8 * 8 bits

or

4 * 16 bits counters

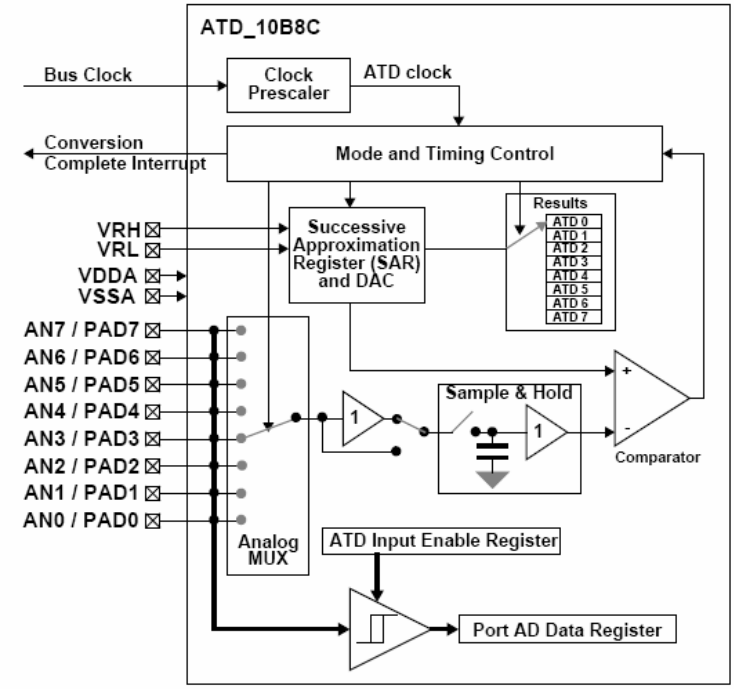


PWM – Pulse Width Modulation

Address	Use	Access
$_00	PWM Enable Register (PWME)	R/W
$_01	PWM Polarity Register (PWMPOL)	R/W
$_02	PWM Clock Select Register (PWMCLK)	R/W
$_03	PWM Prescale Clock Select Register (PWMPRCLK)	R/W
$_04	PWM Center Align Enable Register (PWMCAE)	R/W
$_05	PWM Control Register (PWMCTL)	R/W
$_06	PWM Test Register (PWMTST)[1]	R/W
$_07	PWM Prescale Counter Register (PWMPRSC)[2]	R/W
$_08	PWM Scale A Register (PWMSCLA)	R/W
$_09	PWM Scale B Register (PWMSCLB)	R/W
$_0A	PWM Scale A Counter Register (PWMSCNTA)[3]	R/W
$_0B	PWM Scale B Counter Register (PWMSCNTB)[4]	R/W
$_0C	PWM Channel 0 Counter Register (PWMCNT0)	R/W
$_0D	PWM Channel 1 Counter Register (PWMCNT1)	R/W
$_0E	PWM Channel 2 Counter Register (PWMCNT2)	R/W
$_0F	PWM Channel 3 Counter Register (PWMCNT3)	R/W
$_10	PWM Channel 4 Counter Register (PWMCNT4)	R/W
$_11	PWM Channel 5 Counter Register (PWMCNT5)	R/W
$_12	PWM Channel 6 Counter Register (PWMCNT6)	R/W

```
/* PWM init */
PPWM    pwm = (PPWM) PWM_BASE;

/* low level starts period */
pwm->pwmpol = 0;

/* approx 4 ms period */
pwm->pwmprclk = 0x77;

/* we use PWM 6 */
pwm->pwmper6 = 0xFF;

/* start with 50% duty cycle.. */
pwm->pwmdty6 = 0x80;

/* Activate PWM channel 6 */
pwm->pwme = 0x40;
```


AD – Analog to Digital conversion



Multiplexed.

8 channels.

AD – Analog to Digital conversion

Address Offset	Use	Access
$_00	ATD Control Register 0 (ATDCTL0)[1]	R
$_01	ATD Control Register 1 (ATDCTL1)[2]	R
$_02	ATD Control Register 2 (ATDCTL2)	R/W
$_03	ATD Control Register 3 (ATDCTL3)	R/W
$_04	ATD Control Register 4 (ATDCTL4)	R/W
$_05	ATD Control Register 5 (ATDCTL5)	R/W
$_06	ATD Status Register 0 (ATDSTAT0)	R/W
$_07	Unimplemented	
$_08	ATD Test Register 0 (ATDTEST0)[3]	R
$_09	ATD Test Register 1 (ATDTEST1)	R/W
$_0A	Unimplemented	
$_0B	ATD Status Register 1 (ATDSTAT1)	R
$_0C	Unimplemented	
$_0D	ATD Input Enable Register (ATDDIEN)	R/W
$_0E	Unimplemented	
$_0F	Port Data Register (PORTAD)	R
$_10, $_11	ATD Result Register 0 (ATDDR0H, ATDDR0L)	R/W
$_12, $_13	ATD Result Register 1 (ATDDR1H, ATDDR1L)	R/W
$_14, $_15	ATD Result Register 2 (ATDDR2H, ATDDR2L)	R/W
$_16, $_17	ATD Result Register 3 (ATDDR3H, ATDDR3L)	R/W
$_18, $_19	ATD Result Register 4 (ATDDR4H, ATDDR4L)	R/W
$_1A, $_1B	ATD Result Register 5 (ATDDR5H, ATDDR5L)	R/W
$_1C, $_1D	ATD Result Register 6 (ATDDR6H, ATDDR6L)	R/W
$_1E, $_1F	ATD Result Register 7 (ATDDR7H, ATDDR7L)	R/W

```
/* AD init */
PAD ad = (PAD) AD_BASE;

/* Right justify result, Unsigned result
  Scan (continous mode ), AD channel 6 */
 ad->atd0ctl5 = 0xA6;

/* 8 bit resolution,  16 A/D conversion
   clock cycles, prescaler: divide by 12 */
 ad->atd0ctl4 = 0xE5;

/* A single conversion/sequence */
 ad->atd0ctl3 = 0x40;

/* Normal mode, fast flag clear */
 ad->atd0ctl2 = 0xC0;

/* Read result ...*/
if ( ad->atd0stat0 & 0x80 )
{       /* AD is ready... */
        bits = ad->atd0dr0l;
}
...
```


Summary

we have got a brief introduction to

- The Freescale microcontroller HCS12
- Different Lab systems
- The use of peripherals

which finishes today's lecture ...