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, ”core”

Voltage Regulator

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

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

Central Processing unit CPU12
HCS12DG256, "core"

Embedded Micro Controller Units

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
Clock generation in HCS12

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

\[
\text{PLLCLK} = 2 \times \text{OSCCLK} \times \left( \frac{\text{SYNR} - 1}{\text{REFDV} + 1} \right)
\]

\[
\text{BusClock (E)} = \frac{\text{PLLCLK}}{2}
\]
Real time clock with HCS12

"Address Offset" = $34 in DG256

Three registers are used to control the clock.

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Use</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00</td>
<td>CRG Synthesizer Register (SYNR)</td>
<td>R/W</td>
</tr>
<tr>
<td>$01</td>
<td>CRG Reference Divider Register (REFDV)</td>
<td>R/W</td>
</tr>
<tr>
<td>$02</td>
<td>CRG Test Flags Register (CTFLG)</td>
<td>R/W</td>
</tr>
<tr>
<td>$03</td>
<td>CRG Flags Register (CRGFLG)</td>
<td>R/W</td>
</tr>
<tr>
<td>$04</td>
<td>CRG Interrupt Enable Register (CRGINT)</td>
<td>R/W</td>
</tr>
<tr>
<td>$05</td>
<td>CRG Clock Select Register (CLKSEL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$06</td>
<td>CRG PLL Control Register (PLLCCTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$07</td>
<td>CRG RTI Control Register (RTICTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$08</td>
<td>CRG COP Control Register (COPCTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$09</td>
<td>CRG Force and Bypass Test Register (FORBYP)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0A</td>
<td>CRG Test Control Register (CTCTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$0B</td>
<td>CRG COP AnmTimer Reset (ARMCT)</td>
<td>R/W</td>
</tr>
</tbody>
</table>

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.

CRGINT (adress $38)
Used to enable interrupts

- 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.

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

<table>
<thead>
<tr>
<th>RTR (3:0)</th>
<th>RTR[6:4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 (OFF)</td>
<td>001 010 011 100 101 110 111</td>
</tr>
<tr>
<td>0001</td>
<td>2 10 2 11 2 12 2 13 2 14 2 15 2 16</td>
</tr>
<tr>
<td>0010</td>
<td>3 2 10 3 2 11 3 2 12 3 2 13 3 2 14 3 2 15 3 2 16</td>
</tr>
<tr>
<td>0011</td>
<td>4 2 10 4 2 11 4 2 12 4 2 13 4 2 14 4 2 15 4 2 16</td>
</tr>
<tr>
<td>0100</td>
<td>5 2 10 5 2 11 5 2 12 5 2 13 5 2 14 5 2 15 5 2 16</td>
</tr>
<tr>
<td>0101</td>
<td>6 2 10 6 2 11 6 2 12 6 2 13 6 2 14 6 2 15 6 2 16</td>
</tr>
<tr>
<td>0110</td>
<td>7 2 10 7 2 11 7 2 12 7 2 13 7 2 14 7 2 15 7 2 16</td>
</tr>
<tr>
<td>0111</td>
<td>8 2 10 8 2 11 8 2 12 8 2 13 8 2 14 8 2 15 8 2 16</td>
</tr>
<tr>
<td>1000</td>
<td>9 2 10 9 2 11 9 2 12 9 2 13 9 2 14 9 2 15 9 2 16</td>
</tr>
<tr>
<td>1001</td>
<td>10 2 10 10 2 11 10 2 12 10 2 13 10 2 14 10 2 15 10 2 16</td>
</tr>
<tr>
<td>1010</td>
<td>11 2 10 11 2 11 11 2 12 11 2 13 11 2 14 11 2 15 11 2 16</td>
</tr>
<tr>
<td>1011</td>
<td>12 2 10 12 2 11 12 2 12 12 2 13 12 2 14 12 2 15 12 2 16</td>
</tr>
<tr>
<td>1100</td>
<td>13 2 10 13 2 11 13 2 12 13 2 13 13 2 14 13 2 15 13 2 16</td>
</tr>
<tr>
<td>1101</td>
<td>14 2 10 14 2 11 14 2 12 14 2 13 14 2 14 14 2 15 14 2 16</td>
</tr>
<tr>
<td>1110</td>
<td>15 2 10 15 2 11 15 2 12 15 2 13 15 2 14 15 2 15 15 2 16</td>
</tr>
<tr>
<td>1111</td>
<td>16 2 10 16 2 11 16 2 12 16 2 13 16 2 14 16 2 15 16 2 16</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTIF</td>
<td>PORF</td>
<td>0</td>
<td>LOCKIF</td>
<td>LOCK</td>
<td>TRACK</td>
<td>SCMIF</td>
<td>SCM</td>
</tr>
</tbody>
</table>

- 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

```c
__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 interrupts */
    __asm(" cli");
}
```

"Watchdog" with HCS12

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Use</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>$.00</td>
<td>CRG Synthesizer Register (SYNR)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.01</td>
<td>CRG Reference Divider Register (REFDV)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.02</td>
<td>CRG Test Flags Register (CTFLG)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.03</td>
<td>CRG Flags Register (CRGFLG)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.04</td>
<td>CRG Interrupt Enable Register (CRGIN)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.05</td>
<td>CRG Clock Select Register (CLKSEL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.06</td>
<td>CRG PLL Control Register (PLLCTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.07</td>
<td>CRG RTI Control Register (RTICTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.08</td>
<td>CRG COP Control Register (COPCTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.09</td>
<td>CRG Force and Bypass Test Register (FORBYP)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.0A</td>
<td>CRG Test Control Register (CTCTL)</td>
<td>R/W</td>
</tr>
<tr>
<td>$.0B</td>
<td>CRG COP Arm/Timer Reset (ARMCOP)</td>
<td>R/W</td>
</tr>
</tbody>
</table>

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.

Application should regular write to the ARMCOP register, otherwise "COP Fail" is generated.
Memory usage, example

Banked memory

A bank is selected through a register

"PAGE" (address $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

```assembly
*  FARCALL.S12
*  Example: using “paged” memory
ORG  $1000
NOP
CALL  (farbank&0xFFFF),((farbank>>16)&0xFF)  -- 308000
NOP
*  ---------------------------
ORG  $308000
farbank:
LDAB #30
RTC

```
Banked memory, EXAMPLE 'C'- language

```c
__farseg far_segment void main2( void ) {
    /* application */
}

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

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

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

<table>
<thead>
<tr>
<th>Offset</th>
<th>Use</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>$_0$</td>
<td>SCI Baud Rate Register High (SCIBDH)</td>
<td>Read/Write</td>
</tr>
<tr>
<td>$_1$</td>
<td>SCI Baud Rate Register Low (SCIBDL)</td>
<td>Read/Write</td>
</tr>
<tr>
<td>$_2$</td>
<td>SCI Control Register1 (SCICR1)</td>
<td>Read/Write</td>
</tr>
<tr>
<td>$_3$</td>
<td>SCI Control Register2 (SCICR2)</td>
<td>Read/Write</td>
</tr>
<tr>
<td>$_4$</td>
<td>SCI Status Register 1 (SCISR1)</td>
<td>Read</td>
</tr>
<tr>
<td>$_5$</td>
<td>SCI Status Register 2(SCISR2)</td>
<td>Read/Write</td>
</tr>
<tr>
<td>$_6$</td>
<td>SCI Data Register High (SCIDRH)</td>
<td>Read/Write</td>
</tr>
<tr>
<td>$_7$</td>
<td>SCI Data Register Low (SCIDRL)</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

BaudRate = BusClock/(16 x BR)

BusClock = 8 x 10^6, 1 ≤ BR ≤ 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

```c
/* 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
- 4 * 16 bits counters
PWM – Pulse Width Modulation

`/* 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

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

/* Right justify result, Unsigned result
Scan (continuous mode ), AD channel 6 */
ad->atd0ctl15 = 0xA6;

/* 8 bit resolution, 16 A/D conversion
 clock cycles, prescaler: divide by 12 */
ad->atd0ctl14 = 0xE5;

/* A single conversion/sequence */
ad->atd0ctl13 = 0x40;

/* Normal mode, fast flag clear */
ad->atd0ctl12 = 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 ...