MCF51JM Overview

- 32-bit ColdFire (V1) MCU w/ USB
  - 2.7~5.5V, up to 50 MHz
- descendent of the 68000 (68k) family, now a NXP product
  - same register set
- based on high-performance RISC CPU
  - 32-bit data bus, PC, registers, ALU!
  - 24-bit address bus
  - 2-stage pipelines for each instructions and operands
  - 2 operational modes: user, supervisor
- extensive library of on-board peripheral modules
- multiple operational modes
- single-wire background debug capability (BDM)
2: The MCF51JM Microcontroller

MCF51JM Modules

- Memory: Flash (128KB), RAM (16KB)
- ACMP: analog comparator
- ADC: 12-bit analog-to-digital (12 channels)
- BDM: background debug support, single-wire
- CAN: controller area network
- CMT: carrier modulator timer
- COP: computer operating properly
- IIC: inter-integrated circuit serial bus
- KBI: keyboard interrupt (8 inputs)
- LVD: low voltage detector
- MCG: multipurpose clock generator
- GPIO: I/O ports (51 GP + 6 Rapid GP pins)
- RTC: real-time counter
- SCI, SPI: serial interfaces (2 SCI, 2 SPI)
- TPM: timer/pulse-width modulator (6+2 channels)
- USBOTG: host/device support (dual-role)
2: The MCF51JM Microcontroller

**MCF51JM Packages**

- available in QFP, LQFP packages
  - 44, 64, 80 pins
- Firebird32 uses 64-pin version on a 40-pin DIP module

**Memory Map**

- MCU is *von Neumann*
  - RAM, ROM, IO Registers exist in single map
  - 24-bit address bus -> 16 MB address space
  - 128K FLASH: 0x00_0000..0x01_FFFF
    - includes vectors
  - 16K RAM: 0x80_0000..0x80_3FFF
  - IO Registers: 0xFF_8000..0xFF_FFFF
    - see Reference Manual
2: The MCF51JM Microcontroller

ColdFire CPU

- performs all computation / instruction execution

(CFPRM ColdFire Programmer’s Ref. Manual)

Programing Model – User Mode

- 16 general-purpose 32-bit registers (8 Data + 8 Address)
- 32-bit program counter, PC (top 8 bits forced to zero)
- 8-bit condition code register, CCR
- A7 doubles as SP
- this is User mode, Supervisor mode has additional registers...
2: The MCF51JM Microcontroller

Programming Model - Supervisor

- adds additional CPU registers for "privileged" operations

- MCF51JM includes:
  - 16-bit status register, SR (CCR is lower byte)
  - supervisor SP, OTHER_A7
  - VBR sets base address of vector table (defaults to 0)

Status Register (SR) Details

- system byte only available in supervisor mode
- T: 1=trace enable
- S: 0=user mode, 1=supervisor mode
- M: 1=master state, 0=interrupt state
- I: sets interrupt mask level 0..7
  - 0 = all interrupts enabled
  - 1 = all interrupts disabled (except IRQ pin)
Instruction Format

- ‘word’ = 16 bits (32 bits = ‘longword’)
- first word is instruction “op word”
  - specifies operation, instruction length, EA mode
- additional words specify operands

ColdFire Addressing Modes

<table>
<thead>
<tr>
<th>Addr. Mode</th>
<th>Generation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Operand given</td>
<td>Operand is byte, word, or longword after opword (use ‘#’ in assembly)</td>
</tr>
<tr>
<td>Absolute Short</td>
<td>EA given</td>
<td>16-bit operand EA follows instruction opword</td>
</tr>
<tr>
<td>Absolute Long</td>
<td>EA given</td>
<td>32-bit operand EA follows instruction opword</td>
</tr>
<tr>
<td>Data Register Direct</td>
<td>EA=Dn</td>
<td>Operand is in a Data register</td>
</tr>
<tr>
<td>Address Register Direct</td>
<td>EA=An</td>
<td>Operand is in an Address register</td>
</tr>
<tr>
<td>Address Register Indirect</td>
<td>EA=(An)</td>
<td>Address register contains EA of operand</td>
</tr>
<tr>
<td>Address Register Indirect with Postincrement</td>
<td>EA=(An); An += Size</td>
<td>Address register contains EA of operand &amp; gets incremented after use</td>
</tr>
<tr>
<td>Address Register Indirect with Predecrement</td>
<td>An -= Size; EA=(An)</td>
<td>Address register is first decremented then contains EA of operand</td>
</tr>
<tr>
<td>Address Register Indirect with Displacement</td>
<td>EA = (An)+d16</td>
<td>Operand address is sum of address register plus 16-bit signed displacement</td>
</tr>
<tr>
<td>Program Counter Indirect with Displacement</td>
<td>EA = (PC)+d16</td>
<td>Operand address is sum PC plus 16-bit signed displacement</td>
</tr>
</tbody>
</table>
**ColdFire Instruction Set Summary**

- organized by type of operation
  - data movement
  - program control
  - integer arithmetic
  - floating-point arithmetic (when FPU available)
  - logical operations
  - shift operations
  - bit manipulation
  - system control
  - cache maintenance

---

**Parallel I/O Ports**

- provides interface to general purpose I/O pins

(MCF51JM ColdFire Ref. Manual, ch. 9)
**2: The MCF51JM Microcontroller**

### I/O Ports
- up to 70 i/o pins on up to 9 ports
  - named A..I
- each 8-bit port has i/o register and DDR
  - also have pull-up, slew rate, drive strength, and interrupt control registers
- naming convention: PTxD, PTxDD
  - ex: PTAD, PTEDD
  - ref: table 4-2 in ref. manual
- also: individual bit access
  - ex: PTBD_PTBD2 = 1;

### Real-Time Counter (RTC) Module
- provides hardware time counting functions with optional interrupt

(MCF51JM ColdFire Ref. Manual, ch. 17)
RTC Components

- strictly internal, no external pins
- three input clock sources, software selectable
  - 1 kHz internal low-power oscillator (LPO)
  - 32 kHz internal clock (IRCLK)
  - external clock (ERCLK) – from MCG module
- software-programmable prescaler
- 8-bit up counter with 8-bit modulo match comparator
- software controlled interrupt on modulo match

RTC Block Diagram

shaded boxes represent RTC registers
RTC Register Summary

- only three 8-bit registers:
  - RTCSC = RTC Status and Control register
  - RTCCNT = 8-bit RTC Counter register
  - RTCMOD = 8-bit Modulo register
- all registers default to 0

<table>
<thead>
<tr>
<th>Name</th>
<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>RTCSC</td>
<td>R</td>
<td>W</td>
<td>RTIF</td>
<td>RTCLKS</td>
<td>RTIE</td>
<td>RTOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTCCNT</td>
<td>R</td>
<td>W</td>
<td>RTCONT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTCMOD</td>
<td>R</td>
<td>W</td>
<td>RTCMOD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RTC Status / Control Register

- RTIF (b7): Real-Time Interrupt Flag
  - sets when counter reaches modulo register
  - clear by writing a 1
- RTCLKS (b6-5): Real-Time Clock Source Select
  - chooses RTC clock input source
  - 00=LPO, 01=ERCLK, 1x=IRCLK
- RTIE (b4): Real-Time Interrupt Enable
  - enables RTC interrupts to CPU (when 1)
- RTCPS (b3-0): Real-Time Clock Prescaler Select
  - chooses binary- or decimal-based divide-by values
  - see table in ref. manual; note: 0000=Off
**RTC Usage Example**

- **msdelay():** a precise ms delay function

```c
// msdelay(): delay given number of milliseconds using Real-Time Counter module
void msdelay(int n)
{
    while (n-- > 0) {
        byte ctr = RTCCNT;  // take copy of current RTC Counter register
        while (RTCCNT == ctr) {}  // wait till it changes
    }
}
```

// include in initialization code:

```c
RTCSC = 0x08;  // enable RTC, select 1ms period from 1kHz internal clock
```

- requires RTCMOD > 0
- this version does not use RTIF leaving the RTC modulo feature fully available

---

**INTerrupt Controller (INTC) Module**

- prioritizes all system *exceptions* and performs all *vector* handling

Exception Handling

- **Exception**: an unscheduled (but sometimes planned) event that causes the CPU to depart (i.e. abort) from the normal fetch-decode-execute cycle
  - may or may not be *fault* related
- **Exception processing**:
  1. copy the Status Register (SR) then set SR[S] to switch to *supervisor mode*
  2. determine the appropriate exception *vector number* based on source/cause of exception
  3. save current context (PC, SR) in an 8-byte *exception frame* on the *system stack* (A7’)
  4. fetch to PC address of *exception handler* from *vector table*, resume normal instruction processing
  5. exception handler must end with RTE instruction, after which the interrupted instruction is restarted

Vector Table

- **ColdFire vector table**
  - up to 256 vectors, 4-bytes each
  - each vector contains address of respective exception handler
  - first 64 for CPU, 192 for other uses
    - i.e. peripheral/software/etc.
  - *Vector Base Register* (VBR) points to begin of table
    - this is a supervisor mode register
  - by default, table begins at location 0
    - uses 1st 1 KiB of memory map
- **MCF51JM exceptions**:
  - defines 64 for CPU + 39 for peripheral IRQs (103 total)
## 2: The MCF51JM Microcontroller

### ColdFire Exception Vectors (1/2)

<table>
<thead>
<tr>
<th>Vector Number</th>
<th>Vector Offset</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>Initial stack pointer</td>
</tr>
<tr>
<td>1</td>
<td>004</td>
<td>Initial program counter</td>
</tr>
<tr>
<td>2</td>
<td>008</td>
<td>Access error</td>
</tr>
<tr>
<td>3</td>
<td>00C</td>
<td>Address error</td>
</tr>
<tr>
<td>4</td>
<td>010</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>5</td>
<td>014</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>6-7</td>
<td>018-01C</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>020</td>
<td>Privilege violation</td>
</tr>
<tr>
<td>9</td>
<td>024</td>
<td>Trace</td>
</tr>
<tr>
<td>10</td>
<td>028</td>
<td>Unimplemented line-A opcode</td>
</tr>
<tr>
<td>11</td>
<td>02C</td>
<td>Unimplemented line-F opcode</td>
</tr>
<tr>
<td>12</td>
<td>030</td>
<td>Non-PC breakpoint debug interrupt</td>
</tr>
<tr>
<td>13</td>
<td>034</td>
<td>PC breakpoint debug interrupt</td>
</tr>
<tr>
<td>14</td>
<td>038</td>
<td>Format error</td>
</tr>
<tr>
<td>15</td>
<td>03C</td>
<td>Uninitialized interrupt</td>
</tr>
</tbody>
</table>

### ColdFire Exception Vectors (2/2)

<table>
<thead>
<tr>
<th>Vector Number</th>
<th>Vector Offset</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-23</td>
<td>040-05C</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>060</td>
<td>Spurious interrupt</td>
</tr>
<tr>
<td>25-31</td>
<td>064-07C</td>
<td>Level 1-7 autovectored interrupts</td>
</tr>
<tr>
<td>32-47</td>
<td>080-0BC</td>
<td>Trap #0-15 instructions</td>
</tr>
<tr>
<td>48</td>
<td>0C0</td>
<td>Floating-point branch on unordered condition</td>
</tr>
<tr>
<td>49</td>
<td>0C4</td>
<td>Floating-point inexact result</td>
</tr>
<tr>
<td>50</td>
<td>0C8</td>
<td>Floating-point divide-by-zero</td>
</tr>
<tr>
<td>51</td>
<td>0CC</td>
<td>Floating-point underflow</td>
</tr>
<tr>
<td>52</td>
<td>0D0</td>
<td>Floating-point operand error</td>
</tr>
<tr>
<td>53</td>
<td>0D4</td>
<td>Floating-point overflow</td>
</tr>
<tr>
<td>54</td>
<td>0D8</td>
<td>Floating-point input not-a-number (NAN)</td>
</tr>
<tr>
<td>55</td>
<td>0DC</td>
<td>Floating-point input denormalized number</td>
</tr>
<tr>
<td>56-60</td>
<td>0E0-0F0</td>
<td>Reserved</td>
</tr>
<tr>
<td>61</td>
<td>0F4</td>
<td>Unsupported instruction</td>
</tr>
<tr>
<td>62-63</td>
<td>0F8-0FC</td>
<td>Reserved</td>
</tr>
<tr>
<td>64-255</td>
<td>100-3FC</td>
<td>User-defined interrupts (I/O peripherals)</td>
</tr>
</tbody>
</table>
## 2: The MCF51JM Microcontroller

### MCF51JM Exceptions (1/2)

<table>
<thead>
<tr>
<th>Vector Number</th>
<th>Vector Offset</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>100</td>
<td>IRQ (pin)</td>
</tr>
<tr>
<td>65</td>
<td>104</td>
<td>Low Voltage Detect</td>
</tr>
<tr>
<td>66</td>
<td>108</td>
<td>Loss of Lock</td>
</tr>
<tr>
<td>67</td>
<td>110</td>
<td>SPI1</td>
</tr>
<tr>
<td>68</td>
<td>114</td>
<td>SPI2</td>
</tr>
<tr>
<td>69</td>
<td></td>
<td>USB_Status</td>
</tr>
<tr>
<td>70</td>
<td>118</td>
<td>-</td>
</tr>
<tr>
<td>71</td>
<td>120</td>
<td>TPM1 Channel 0</td>
</tr>
<tr>
<td>72</td>
<td>124</td>
<td>TPM1 Channel 1</td>
</tr>
<tr>
<td>73</td>
<td>128</td>
<td>TPM1 Channel 2</td>
</tr>
<tr>
<td>74</td>
<td>130</td>
<td>TPM1 Channel 3</td>
</tr>
<tr>
<td>75</td>
<td>134</td>
<td>TPM1 Channel 4</td>
</tr>
<tr>
<td>76</td>
<td>138</td>
<td>TPM1 Channel 5</td>
</tr>
<tr>
<td>77</td>
<td>140</td>
<td>TPM1 Overflow</td>
</tr>
<tr>
<td>78</td>
<td>144</td>
<td>SCI1 Error</td>
</tr>
<tr>
<td>79</td>
<td>148</td>
<td>SCI1 Receive</td>
</tr>
<tr>
<td>80</td>
<td>14C</td>
<td>SCI1 Transmit</td>
</tr>
<tr>
<td>81</td>
<td>150</td>
<td>SCI2 Error</td>
</tr>
<tr>
<td>82</td>
<td>154</td>
<td>SCI2 Receive</td>
</tr>
<tr>
<td>83</td>
<td>158</td>
<td>SCI2 Transmit</td>
</tr>
<tr>
<td>84</td>
<td>15C</td>
<td>KBI Interrupt</td>
</tr>
<tr>
<td>85</td>
<td>160</td>
<td>ADC Conversion</td>
</tr>
<tr>
<td>86</td>
<td>164</td>
<td>ACMP</td>
</tr>
<tr>
<td>87</td>
<td>168</td>
<td>ICC1</td>
</tr>
<tr>
<td>88</td>
<td>16C</td>
<td>RTC</td>
</tr>
<tr>
<td>89</td>
<td>170</td>
<td>ICC2</td>
</tr>
<tr>
<td>90</td>
<td>174</td>
<td>CM2</td>
</tr>
<tr>
<td>91</td>
<td>178</td>
<td>CAN Wakeup</td>
</tr>
<tr>
<td>92</td>
<td>17C</td>
<td>CAN Error</td>
</tr>
<tr>
<td>93</td>
<td>180</td>
<td>CAN Receive</td>
</tr>
<tr>
<td>94</td>
<td>184</td>
<td>CAN Transmit</td>
</tr>
<tr>
<td>95</td>
<td>188</td>
<td>RNGA Error</td>
</tr>
<tr>
<td>96</td>
<td>1A0-1BC</td>
<td>Force_lvl (i7..1)</td>
</tr>
</tbody>
</table>

### MCF51JM Exceptions (2/2)
2: The MCF51JM Microcontroller

**Interrupt Dispositions**

- some interrupts are *non-maskable* but most are *maskable*
  - programmer can enable / disable at will
- CPU CCR contains a 3-bit *interrupt priority* field for controlling maskable interrupts
  - 0..7, any level *below* current setting is disabled
  - resets to 7
- CodeWarrior’s `hidef.h` file contains relevant macros
  - `EnableInterrupts`: sets level to 0
  - `DisableInterrupts`: sets level to 7
- `derivative.h` defines all vector numbers
  - ex: `#define VectorNumber_Vrtc 91U`

---

**Spoiler alert! (how to program an ISR)**

- ex: an ISR to handle RTC interrupts
  - must also enable interrupts in the RTCSC register, i.e. RTIE must be set!

```c
// rtc_isr(): process interrupts from RTC module
interrupt VectorNumber_Vrtc void rtc_isr(void)
{
  RTCSC_RTIF = 1;   // acknowledge & reset RTIF flag
  // process RTC event – timekeeping, update LEDs, etc.
}
```

- ISRs can neither accept or return arguments
- "interrupt" causes ';' to be an RTE instead of RTS
  - and respective interrupt vector to be setup
2: The MCF51JM Microcontroller

**Timer/PWM (TPM) Module**

- a highly flexible peripheral module used to perform timing-related tasks in hardware

(MCF51JM ColdFire Ref. Manual, ch. 22)

**TPM Features**

- two TPMs
  - TPM1 has 6 channels, TPM2 has 2
- optionally uses PortE,F bits for external I/O
- input capture
  - measure characteristics of input pulses/signals
- output compare
  - generation of programmer-defined signals
  - pulse/periodic, frequency, duty cycle
- powerful interrupt capabilities
TPM Components

- 16-bit binary up counter driven by BUSCLK thru with programmable prescaler
  - 8 choices: ÷1 .. ÷128

- up to 8 channels, ea. 16-bits, programmable for input capture (IC) or output compare (OC) operation

- PWM generation capability

- set of control/status registers
2: The MCF51JM Microcontroller

TPM Register Summary

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Notes</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMxSC</td>
<td></td>
<td>TPM Status and Control Register</td>
<td>primary status / control functions: clock source select, prescale factor select, timer overflow interrupt enable</td>
</tr>
<tr>
<td>TPMxCNT</td>
<td>1</td>
<td>TPM Counter Register</td>
<td>16-bit binary up counter</td>
</tr>
<tr>
<td>TPMxMOD</td>
<td>1</td>
<td>TPM Counter Modulo Register</td>
<td>sets modulo value for CNT counter register</td>
</tr>
<tr>
<td>TPMxCnSC</td>
<td>2</td>
<td>TPM Channel n Status / Control Register</td>
<td>per channel status/control functions: mode select, edge/level select, channel interrupt enable</td>
</tr>
<tr>
<td>TPMxCnV</td>
<td>1,2</td>
<td>TPM Channel Value Register</td>
<td>CNT value at Input Capture or Output Compare event</td>
</tr>
</tbody>
</table>

Notes:
1) are 16-bit registers but also support H/L-byte access
2) repeat for as many channels as available

TPM Interrupts

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Local Enable</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOF</td>
<td>TOIE (TPMxSC)</td>
<td>Timer overflow</td>
<td>Timer Overflow interrupt</td>
</tr>
<tr>
<td>CHnF</td>
<td>CHnIE (TPMxCnSC)</td>
<td>Channel event</td>
<td>Input capture or output compare event occurred on channel n</td>
</tr>
</tbody>
</table>

Notes:
- each channel has its own interrupt vector
2: The MCF51JM Microcontroller

ADC Module

- a peripheral module providing 28 channels of 8-, 10- or 12-bit A/D conversion

(MCF51JM ColdFire Ref. Manual, ch. 21)

ADC Features

- 28 input channels (12 externally available)
  - 8-, 10- or 12-bit resolution
  - right justified, unsigned result
  - selectable ADC clock
  - conversion time under 2 us possible!
  - per command or continuous conversion modes
  - internal temperature sensor

- one interrupt source
  - conversion complete
ADC Module Components

- conversion clock selection & prescaler
- 32 inputs via analog multiplexer
- successive approx. register (SAR)
- compare function
- interrupt logic

ADC Block Diagram
## ADC Register Map

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Register Name</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>ADCSC1</td>
<td>Status and Control Register 1 selects channel, conversion mode, interrupt enable; provides conversion complete status (COCO)</td>
</tr>
<tr>
<td>0x0001</td>
<td>ADCSC2</td>
<td>Status and Control Register 2 sets conversion trigger and compare features</td>
</tr>
<tr>
<td>0x0002</td>
<td>ADCH</td>
<td>Data Result High Register top 2 (10-bit) or 4 (12-bit) bits of ADC result</td>
</tr>
<tr>
<td>0x0003</td>
<td>ADCL</td>
<td>Data Result Low Register bottom 8 bits of ADC result</td>
</tr>
<tr>
<td>0x0004</td>
<td>ADCCVH</td>
<td>Compare Value High Register high byte of compare value (when enabled)</td>
</tr>
<tr>
<td>0x0005</td>
<td>ADCCVL</td>
<td>Compare Value Low Register low byte of compare value (when enabled)</td>
</tr>
<tr>
<td>0x0006</td>
<td>ADCCFG</td>
<td>Configuration Register selects ADC clocking, sample time, # bits (8,10,12)</td>
</tr>
<tr>
<td>0x0007</td>
<td>APCTL1</td>
<td>Pin Control 1 Register disables pin I/O control, channels 0-7</td>
</tr>
<tr>
<td>0x0008</td>
<td>APCTL2</td>
<td>Pin Control 1 Register disables pin I/O control, channels 8-15</td>
</tr>
</tbody>
</table>

Above addresses are offsets from ADC base address (0xFFFF8010).

## ADC Channel Assignments (MCF51)

<table>
<thead>
<tr>
<th>ADCH*</th>
<th>Channel</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>AD0</td>
<td>PTB0/MISO2/ADP0</td>
</tr>
<tr>
<td>00001</td>
<td>AD1</td>
<td>PTB1/MOSI2/ADP1</td>
</tr>
<tr>
<td>00010</td>
<td>AD2</td>
<td>PTB2/PSCK2/ADP2</td>
</tr>
<tr>
<td>00011</td>
<td>AD3</td>
<td>PTB3/SS2/ADP3</td>
</tr>
<tr>
<td>00100</td>
<td>AD4</td>
<td>PTB4/KB1P4/ADP4</td>
</tr>
<tr>
<td>00101</td>
<td>AD5</td>
<td>PTB5/KB1P5/ADP5</td>
</tr>
<tr>
<td>00110</td>
<td>AD6</td>
<td>PTB6/ADP6</td>
</tr>
<tr>
<td>00111</td>
<td>AD7</td>
<td>PTB7/ADP7</td>
</tr>
<tr>
<td>01000</td>
<td>AD8</td>
<td>PTD0/ADP8/ACMP+</td>
</tr>
<tr>
<td>01001</td>
<td>AD9</td>
<td>PTD1/ADP9/ACMP-</td>
</tr>
<tr>
<td>01010</td>
<td>AD10</td>
<td>PTD3/KB3P3/ADP10</td>
</tr>
<tr>
<td>01011</td>
<td>AD11</td>
<td>PTD4/ADP11</td>
</tr>
<tr>
<td>01100</td>
<td>AD12</td>
<td>VREFL</td>
</tr>
<tr>
<td>01101</td>
<td>AD13</td>
<td>VREFL</td>
</tr>
<tr>
<td>01110</td>
<td>AD14</td>
<td>VREFL</td>
</tr>
<tr>
<td>01111</td>
<td>AD15</td>
<td>VREFL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADCH*</th>
<th>Channel</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>AD16</td>
<td>VREFL</td>
</tr>
<tr>
<td>10001</td>
<td>AD17</td>
<td>VREFL</td>
</tr>
<tr>
<td>10010</td>
<td>AD18</td>
<td>VREFL</td>
</tr>
<tr>
<td>10011</td>
<td>AD19</td>
<td>VREFL</td>
</tr>
<tr>
<td>10100</td>
<td>AD20</td>
<td>VREFL</td>
</tr>
<tr>
<td>10101</td>
<td>AD21</td>
<td>VREFL</td>
</tr>
<tr>
<td>10110</td>
<td>AD22</td>
<td>Reserved</td>
</tr>
<tr>
<td>10111</td>
<td>AD23</td>
<td>Reserved</td>
</tr>
<tr>
<td>11000</td>
<td>AD24</td>
<td>Reserved</td>
</tr>
<tr>
<td>11001</td>
<td>AD25</td>
<td>Reserved</td>
</tr>
<tr>
<td>11010</td>
<td>AD26</td>
<td>Temperature Sensor</td>
</tr>
<tr>
<td>11011</td>
<td>AD27</td>
<td>Interamp bandgap</td>
</tr>
<tr>
<td>11100</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>11101</td>
<td>VREFL</td>
<td>VDD</td>
</tr>
<tr>
<td>11110</td>
<td>VREFL</td>
<td>VSS</td>
</tr>
<tr>
<td>11111</td>
<td>ADC off</td>
<td>None</td>
</tr>
</tbody>
</table>

*bottom 5 bits written to the ADCSC1 register
2: The MCF51JM Microcontroller

ADC Application (no compare)

- initialization or per conversion:
  - select ADC clock source & divide-by, # bits for conversion (**ADCCFG**)
  - set pin control register bit(s) to switch between port i/o or ADC function
    - **ADPCx**: 0=port i/o, 1=A/D function

- per conversion:
  - select channel to convert (**ADCSC1**)
  - wait for conversion complete (**COCO**)
  - read data from result registers (**ADCRx**)

---

ADC Example

```c
#include "mcf51jm128.h"

// ADC initialization:
ADCCFG = ??? // select Busclk/4
+ ???; // 10-bit mode
ADPC1_ADPC4 = ?; // select A/D for channel 4 pin

// perform conversion
ADCSC1 = ?; // start conversion on PTB4
while (!ADCSC1_COCO()); // wait for conversion complete*
int adcresult = ADCR; // grab 2-byte result
```

*can also use “WAITFOR” macro here! i.e.:

```
WAITFOR(ADCSC1_COCO);
```
2: The MCF51JM Microcontroller

**SCI Module**

- implements full duplex, asynchronous serial communication at programmed baud rates with framing

  (MCF51JM ColdFire Ref. Manual, ch. 12)

**SCI Features**

- configurable baud rate
  - supports standard asynchronous rates
- 8- or 9-bit data format
  - for multi-node communications
- receiver error detection
  - parity, noise, framing errors
- 10 interrupt events
  - although shared through 3 common vectors
- uses pins on Ports C,E
  - most MCF members have multiple SCIs!
- implements a total of 7 r/w registers
2: The MCF51JM Microcontroller

SCI Components

- 13-bit baud rate divider
  - determines data rate for both Tx & Rx
- transmitter w/ 11-bit PISO shift register
  - serializes output data to TxD with framing
  - parity generator
  - 2 interrupt events shared on SCI Transmit vector
- receiver w/ 11-bit SIPO shift register
  - deserializes input data from RxD
  - 4 rx interrupt events on SCI Receive vector
  - 4 error events on SCI Error vector

SCI Transmitter Block Diagram
2: The MCF51JM Microcontroller

SCI Receiver Block Diagram

SCI Register Map

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SCIxBDH</td>
<td>SCI Baud rate register High</td>
</tr>
<tr>
<td>1</td>
<td>SCIxBDL</td>
<td>SCI Baud rate register Low</td>
</tr>
<tr>
<td>2</td>
<td>SCIxC1</td>
<td>SCI Control Register 1</td>
</tr>
<tr>
<td>3</td>
<td>SCIxC2</td>
<td>SCI Control Register 2</td>
</tr>
<tr>
<td>4</td>
<td>SCIxS1</td>
<td>SCI Status Register 1</td>
</tr>
<tr>
<td>5</td>
<td>SCIxS2</td>
<td>SCI Status Register 2</td>
</tr>
<tr>
<td>6</td>
<td>SCIxC3</td>
<td>SCI Control Register 3</td>
</tr>
<tr>
<td>7</td>
<td>SCIxD</td>
<td>SCI Data Register (Tx/Rx)</td>
</tr>
</tbody>
</table>

Above addresses are offsets from SCI base address.

- SCI1 base address = 0xFFFF8038
- SCI2 base address = 0xFFFF8040
Baud Rate Setting

- baud rate is determined from BUSCLK and 13 SBR bits divider in Baud rate register:
  \[ \text{SCI Baud} = \frac{\text{BUSCLK}}{16 \cdot \text{SBR}[12:0]} \]

- example SBR values:

<table>
<thead>
<tr>
<th>Baud</th>
<th>BUSCLK = 12MHz</th>
<th>% Error</th>
<th>BUSCLK = 16MHz</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2400</td>
<td>313</td>
<td>-0.16</td>
<td>417</td>
<td>-0.08</td>
</tr>
<tr>
<td>4800</td>
<td>156</td>
<td>0.16</td>
<td>208</td>
<td>0.16</td>
</tr>
<tr>
<td>9600</td>
<td>78</td>
<td>0.16</td>
<td>104</td>
<td>0.16</td>
</tr>
<tr>
<td>19200</td>
<td>39</td>
<td>0.16</td>
<td>52</td>
<td>0.16</td>
</tr>
<tr>
<td>38400</td>
<td>20</td>
<td>-2.34</td>
<td>26</td>
<td>0.16</td>
</tr>
</tbody>
</table>

NRZ Serial Data Format

- “Non Return to Zero”

- format determined by SCIxC1 bits:

<table>
<thead>
<tr>
<th>LOOPS</th>
<th>SCISW</th>
<th>RSRC</th>
<th>M</th>
<th>Wake</th>
<th>ILT</th>
<th>PE</th>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- M: mode (0=“1/8/1”, 1=“1/9/1”)
- PE: parity enable (0=off, 1=on)
- PT: parity type (0=even, 1=odd)
2: The MCF51JM Microcontroller

SCI Interrupts – 1/3

- SCI can generate interrupts
  - for any of 4 different events as enabled in SCIxC2 (top 4 “IE” bits):
    - Transmit, Transmit Complete share SCI Transmit vector
    - Receive, Idle share SCI Receive vector
  - also: SCI transmitter & receiver must be turned on!
    - by setting TE & RE (Enable) bits

SCI Interrupts – 2/3

- error interrupts
  - for any of 4 different error events as enabled in SCIxC3 (bottom 4 “IE” bits):
    - overrun, noise error, framing error, parity error
    - shared on SCI Error vector
SCI Interrupts – 3/3

- ISR must check SCIxS1 to determine cause of interrupt and react appropriately (only for multiple enabled events)
  - top 4 bits...
  
<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>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PF</td>
</tr>
</tbody>
</table>

- bottom 4 bits reflect error status of previously received character

SCI Operation – 1/2

- initialization
  - write baud registers set baud rate
    - SCIxBDH first, then SCIxBDL
    - can also just write 16 bit word to SCIxBD
  - write SCIxC1 to set data format
  - write SCIxC2 to turn on SCI & enable Tx/Rx interrupts as appropriate
  - optional: write SCIxC3 to turn on error interrupts if needed
  - also make sure SCI vectors are setup if using SCI interrupts!
SCI Code: Initialization

- activate SCI for 9600 baud (at 16 MHz bus)
  - (and no interrupts)

```c
// SCI initialize routine
void SCI1_init()
{
  SCI1BD = ???;   // set baud rate
  SCI1C1 = ???;   // no special settings needed
  SCI1C2 = ???;   // enable Tx & Rx
}
```

SCI Operation – 2/2

- transmission
  - wait for transmitter empty
    - TDRE=1 of SCIxS1
  - write new character to SCIxD

- reception
  - wait for character to be received
    - RDRF=1 in SCIxS1
  - read received char from SCIxD

Note: this approach will “block” the CPU!
- what if blocking is not desired?
SCI Code: Single Byte I/O

```c
void SCI1_putc(char c) // SCI output routine
{
    if (c == '\n')
        SCI1_putchar('\r'); // auto CR on LF
    while (!(SCI1_TDRE)) {} // wait for SCI Tx ready
    SCI1D = c; // send the char
}

char SCI1_getc() // SCI input routine
{
    char stat, chr;
    while (!(SCI1RDRF)) {} // wait for received char
    stat = SCI1S1 & 0xf; // check for OR,NF,FE,PE
    chr = SCI1D; // read received SCI char
    if (stat != 0) chr = '?'; // error check
    return chr; // return received char
}
```

SCI Code: String Output

```c
void SCI1_puts(char *s) // SCI string output routine
{
    while (*s) // while *s not terminating null,
        SCI1_putchar(*s++); // output 1 char & advance pointer
}
```