Multi-Channel Buffer Serial Port (McBSP)

Hardware & Software (TMS320C6416)

- 600 MHz ‘C6416 DSP
- AIC23 Stereo Codec
- External Memory
  - 16M Bytes SDRAM
  - 512K Bytes Flash ROM
- 4 user accessible LED’s and DIP Switches
- Daughter card expansion
- Software Board Configuration through registers implemented in CPLD
- JTAG Emulation through on-board JTAG emulator with USB host interface or external emulator
- Power Supply & Parallel Port Cable
C6416 DSK: McBSP ↔ Codec Interface

- McBSP1 connected to program AIC23's control registers
- McBSP2 is used to transfer data to A/D and D/A converters
- Programmable frequency: 8K, 16K, 24K, 32K, 44.1K, 48K, 96K
- 24-bit converter, Digital transfer widths: 16-bits, 20-bits, 24-bits, 32-bits
Determine Ready Status

- The **RRDY** and **XRDY** bits in **SPCR** indicate the ready state of the McBSP receiver and transmitter, respectively.

- Writes and Reads from the serial port can be synchronized by well known existing methods.
Determining Ready Status

1. **Polling RRDY and XRDY bits.**

2. **Using the events sent to the DMA or EDMA controller (REVT & XEVT).**

3. **Using the interrupts to the CPU (RINT and XINT) that the events generate.**

   Importantly, Reading DRR and writing to DXR affects RRDY and XRDY bits.

---

**SPCR**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>RRDY</td>
</tr>
<tr>
<td>27</td>
<td>RINT</td>
</tr>
<tr>
<td>26</td>
<td>XRDY</td>
</tr>
<tr>
<td>25</td>
<td>XINT</td>
</tr>
<tr>
<td>24</td>
<td>OUT</td>
</tr>
<tr>
<td>19</td>
<td>XEMPTY</td>
</tr>
<tr>
<td>18</td>
<td>XSYNCERR</td>
</tr>
<tr>
<td>17</td>
<td>XSYNC</td>
</tr>
<tr>
<td>16</td>
<td>XCEST</td>
</tr>
<tr>
<td>15</td>
<td>CLKSTP</td>
</tr>
<tr>
<td>14</td>
<td>CLKST</td>
</tr>
<tr>
<td>13</td>
<td>DLYST</td>
</tr>
<tr>
<td>12</td>
<td>DLY</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>EKST</td>
</tr>
<tr>
<td>6</td>
<td>EKST</td>
</tr>
<tr>
<td>5</td>
<td>EKST</td>
</tr>
<tr>
<td>4</td>
<td>EKST</td>
</tr>
<tr>
<td>3</td>
<td>EKST</td>
</tr>
<tr>
<td>2</td>
<td>EKST</td>
</tr>
<tr>
<td>1</td>
<td>EKST</td>
</tr>
<tr>
<td>0</td>
<td>EKST</td>
</tr>
</tbody>
</table>
The receive interrupt (RINT) and transmit interrupt (XINT) signals inform the CPU of changes to the serial port status. Four options exist for configuring these interrupts.

These options are set by the receive/transmit interrupt mode bits (RINTM and XINTM) in SPCR.

The possible values of the mode, and the configurations they represent are:

1. \((R/X)\text{INTM} = 00b\) Interrupt on every serial element by tracking the \((R/X)\text{RDY}\) bits in SPCR.
2. \((R/X)\text{INTM} = 01b\) Interrupt at the end of a sub-frame (16 elements or less) within a frame.
3. \((R/X)\text{INTM} = 10b\) Interrupt on detection of frame synchronization pulses.
4. \((R/X)\text{INTM} = 11b\) Interrupt on frame synchronization error.

\(\text{RRDY} = 1\) indicates that the \(\text{RBR}\) contents have been copied to \(\text{DRR}\) and that the data can now be read by either the CPU or the DMA/EDMA controller.

Once that data has been read by either the CPU or the DMA/EDMA controller, \(\text{RRDY}\) is cleared to 0.

Also, at device reset or serial port receiver reset \((\text{RRST} = 0)\), the \(\text{RRDY}\) bit is cleared to 0 to indicate that no data has been received and loaded into \(\text{DRR}\).
**Signal Processing**

**RCR**

*Single phase or double phase (0, 1)*

Determines the word length of the element: 8, 12, 16, 20, 24, 32

The (R/X)WDLEN1/2 fields in the receive/transmit control register (RCR and XCR) determine the element length in bits per element for the receiver and the transmitter for each phase of the frame.

If (R/X)PHASE = 0, indicating a single-phase frame, (R/X)WDLEN2 is not used by the McBSP and its value does not matter.
### RCR

**(R/X)WDLN1:2**

<table>
<thead>
<tr>
<th>Element Length (Dits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
</tr>
<tr>
<td>001</td>
</tr>
<tr>
<td>010</td>
</tr>
<tr>
<td>011</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>111</td>
</tr>
</tbody>
</table>

- **Reserved**

- **Reserved**

### XCR

**Single phase or double phase (0, 1)**

- **Determines the word length of the element:**
  - 8, 12, 16, 20, 24, 32
**SRGR SAMPLE RATE GENERATOR REGISTER**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSYNC</td>
<td>SRG clock divider value.</td>
</tr>
<tr>
<td>CLKSP</td>
<td>Frame period value plus 1 specifies when the next frame sync signal becomes ready.</td>
</tr>
<tr>
<td>CLKSM</td>
<td>Sample rate generator transmit frame synchronization bit.</td>
</tr>
<tr>
<td>FSQM</td>
<td>Internal (CPU) or External (Pin) clock to derive.</td>
</tr>
<tr>
<td>FPER</td>
<td>Frame period value plus 1 specifies when the next frame sync signal becomes ready.</td>
</tr>
<tr>
<td>FSGM</td>
<td>Sample rate generator transmit frame synchronization bit.</td>
</tr>
<tr>
<td>CLKSM</td>
<td>Internal (CPU) or External (Pin) clock to derive.</td>
</tr>
<tr>
<td>RWD</td>
<td>Internal (CPU) or External (Pin) clock to derive.</td>
</tr>
<tr>
<td>CLKGDV</td>
<td>Internal (CPU) or External (Pin) clock to derive.</td>
</tr>
</tbody>
</table>

**PCR PIN CONTROL REGISTER**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>XIOEN</td>
<td>Receiver clock mode bit (Input or Output).</td>
</tr>
<tr>
<td>RCIOEN</td>
<td>Receiver clock mode bit (Input or Output).</td>
</tr>
<tr>
<td>FSXM</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
<tr>
<td>FSRM</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
<tr>
<td>CLKSM</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
<tr>
<td>CLKRM</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
<tr>
<td>CLKSTAT</td>
<td>Receiver clock mode bit (Input or Output).</td>
</tr>
<tr>
<td>DXSTAT</td>
<td>Receiver clock mode bit (Input or Output).</td>
</tr>
<tr>
<td>DRSTAT</td>
<td>Receiver clock mode bit (Input or Output).</td>
</tr>
<tr>
<td>FSRP</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
<tr>
<td>CLKP</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
<tr>
<td>CLKRP</td>
<td>Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).</td>
</tr>
</tbody>
</table>
Volatile Variable

In computer programming, a variable or object declared with the volatile keyword may be modified externally from the declaring object.

For example, a variable that might be concurrently modified by multiple threads (without locks or a similar form of mutual exclusion) should be declared volatile.

Machines having memory mapped input/output, pointer to a device may be declared volatile which will not be optimized by the compiler because their value can change at any time.
Programming the Serial Port:

- Three methods available for programming the serial port:
  1. Writing directly to the serial port registers.
  2. Using the Chip Support Library (CSL).
  3. Graphically using the DSP/BIO5 GUI configuration tool.

Writing directly to the serial port registers:

- This method is straightforward however, it relies on a good understanding of the serial port functionality.
- This method can be tedious and is prone to errors.

```c
#include <c6416dsk.h>

void mcbsp0_init()
{
    *(unsigned volatile int *)McBSP0_SPCR = 0;
    *(unsigned volatile int *)McBSP0_PCR = 0;
    *(unsigned volatile int *)McBSP0_RCR = 0x10040;
    *(unsigned volatile int *)McBSP0_XCR = 0x10040;
    *(unsigned volatile int *)McBSP0_DXR = 0;
    *(unsigned volatile int *)McBSP0_SPCR = 0x12001;
}
```
Using the Chip Support Library:

- The CSL provides a C language interface for configuring and controlling the on-chip peripherals, in this case the Serial Ports.
- The library is modular with each module corresponding to a specific peripheral. This has the advantage of reducing the code size.
- Some modules rely on other modules also being included, for example the IRQ module is required when using the EDMA module.

CSL programming procedure:

(1) Create handles for the serial ports:

```c
MCBSP_Handle hMcbsp;
```

(2) Open the serial port:

```c
hMcbsp = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
```
CSL programming procedure continues:

(3) Configure the serial port:

```c
MCBSP_config(hMcbsp, &MyConfiguration);
```

(4) Close the Serial Port after use:

```c
MCBSP_close(hMcbsp);
```

DSP/BIOS GUI Interface:

- With this method the configuration structure is created graphically and the setup code is generated automatically.
Lecture 4: (DE6 2011)

Procedure:

1. **Create a configuration using the McBSP Configuration manager (eg. mcbspCfg0).**

   - **Procedure:**
   - Right click and select “Properties”, see the figure below, and then select “Advanced” and fill all parameters as shown below:
Procedure:

Select the serial port you would like to use from the MCBSP Resource manager (e.g., Mcbsp_Port1).

Right click and select properties.

Select the mcbspCfg0 configuration just created.

A file is then generated that contains the configuration code. The file generated for this example is shown on the next slide.
# Lecture 4: (DE6 2011) 16

```c
/* Do *not* directly modify this file. It was */
/* generated by the Configuration Tool: any */
/* changes risk being overwritten. */

/* INPUT mcbsp1.cdb */
/* Include Header File */
#include "mcbsp1cfg.h"

/* Config Structures */
MCBSP_Config mcbspCfg0 = {
  0x00008000, /* Serial Port Control Reg. (SPCR) */
  0x000000A0, /* Receiver Control Reg. (RCCR) */
  0x000000A0, /* Transmitter Control Reg. (TCCR) */
  0x203F1F0F, /* Sample-Rate Generator Reg. (SRGR) */
  0x00000000, /* Multichannel Control Reg. (MCCR) */
  0x00000000, /* Receiver Channel Enable (RCER) */
  0x00000000, /* Transmitter Channel Enable (TCER) */
  0x00000A00  /* Pin Control Reg. (PCCR) */
};

/* Handles */
MCBSP_Handle hMcbsp1;

/* === CSL_cfgInit() ===*/
void CSL_cfgInit()
{
  hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
  MCBSP_config(hMcbsp1, &mcbspCfg0);
}
```

---

**Notice:**

Values in the code generated are the same as the values inserted using the GUI interface.
```c
#include <c6416dsk.h>

void mcbsp_init()
{
    /* Reset the McBSP */
    *(unsigned volatile int *)McBSP_SPCR = 0;

    /* Setting Pin Control Register; Default */
    *(unsigned volatile int *)McBSP_PCR = 0;

    /* Setting RCR, 16 bit receive, No Companding, 1 bit delay */
    *(unsigned volatile int *)McBSP_RCR = 0x10040;

    /* Setting TXR, 16 bit transmit, No Companding, 1 bit delay */
    *(unsigned volatile int *)McBSP_XCR = 0x10040;

    /* Clear Data Transmission Register */
    *(unsigned volatile int *)McBSP_DXR = 0;

    /* Now Enabling the port operation through SPCR */
    *(unsigned volatile int *)McBSP_SPCR = 0x12001;
}
```
void mcbsp_write (int out_data)
{
    int output_reg;
    output_reg = *(unsigned volatile int *) McBSP_SPCR & 0x20000;
    while (output_reg == 0)
    {
        output_reg = *(unsigned volatile int *) McBSP_SPCR & 0x20000;
        *(unsigned volatile int *) McBSP_DXR = out_data;
    }
}

int mcbsp_read()
{
    int input_reg;
    input_reg = *(unsigned volatile int *) McBSP_SPCR & 0x2;
    while (input_reg == 0)
    {
        input_reg = *(unsigned volatile int *) McBSP_SPCR & 0x2;
        input_reg = *(unsigned volatile int *) McBSP_DKR;
        return input_reg;
    }
}

main()
{
    ....................
}

For own Reference: McBSP_DRR_DR_symval