Signal Processing

Multi-Channel Buffer Serial Port (McBSP)

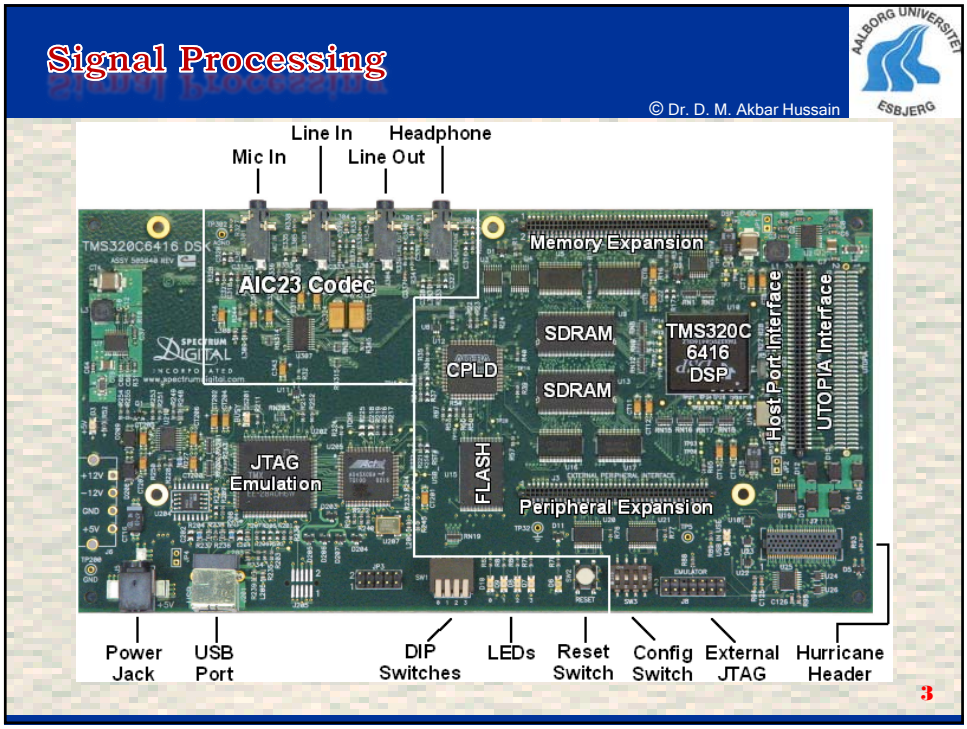
1

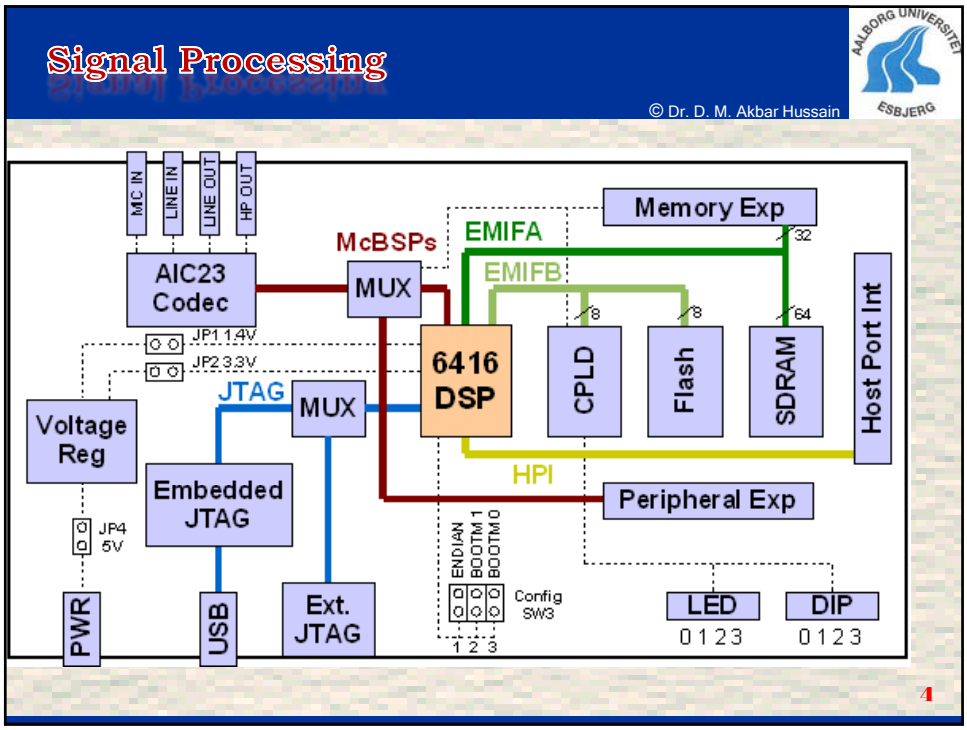
Signal Processing

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

2

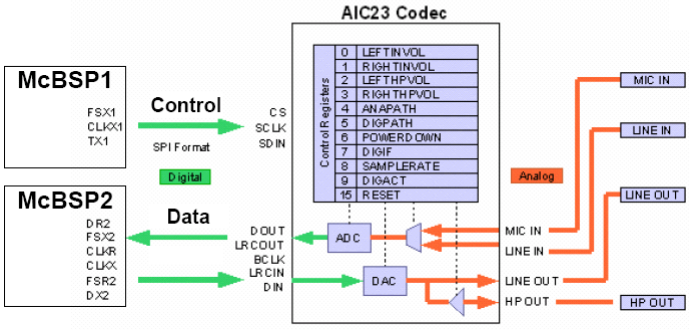

Signal Processing
© Dr. D. M. Akbar Hussain
Line In
Mic In
Headphone
Line Out
TMS320C6416 DSK
AIC23 Codec
Memory Expansion
SDRAM
SDRAM
CPLD
TMS320C 6416 DSP
Host Port Interface
UTOPIA Interface
FLASH
JTAG Emulation
Peripheral Expansion
Power Jack
USB Port
DIP Switches
LEDs
Reset Switch
Config Switch
External JTAG
Hurricane Header
3


Signal Processing
© Dr. D. M. Akbar Hussain
MIC IN
LINE IN
LINE OUT
HP OUT
AIC23 Codec
McBSPs
MUX
EMIFA
EMIFB
Memory Exp
32
8
8
64
JP1 1.4V
JP2 3.3V
6416 DSP
CPLD
Flash
SDRAM
Host Port Int
JTAG
MUX
Voltage Reg
Embedded JTAG
HPI
Peripheral Exp
JP4 5V
ENDIAN
BOOTM 1
BOOTM 0
Config SW3
1 2 3
LED
0 1 2 3
DIP
0 1 2 3
PWR
USB
Ext. JTAG
4

Signal Processing

© Dr. D. M. Akbar Hussain

C6416 DSK: McBSP ↔ Codec Interface

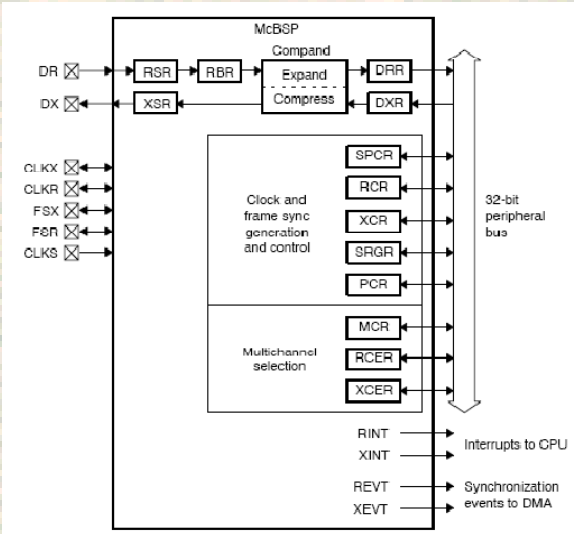


- McBSP1 connected to program AIC23's control registers
- McBSP2 is used to transfer data to A/D and D/A converters
- Programmable frequency: 8K, 16K, 24K, 32K, 44.1K, 48K, 96K
- 24-bit converter, Digital transfer widths: 16-bits, 20-bits, 24-bits, 32-bits

5

Signal Processing

© Dr. D. M. Akbar Hussain



6

Signal Processing

Receive	RSR	Receive Shift Reg
	RBR	Receive Buffer Reg
	DRR	Data Receive Reg
Transmit	XSR	Transmit Shift Reg
	DXR	Data Transmit Reg
Control	SPCR	Serial Port Control Reg
	RCR	Receive Control Reg
	XCR	Transmit Control Reg
	SRGR	Sample Rate Generator
	PCR	Pin Control Reg

7

Signal Processing

Determining 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.

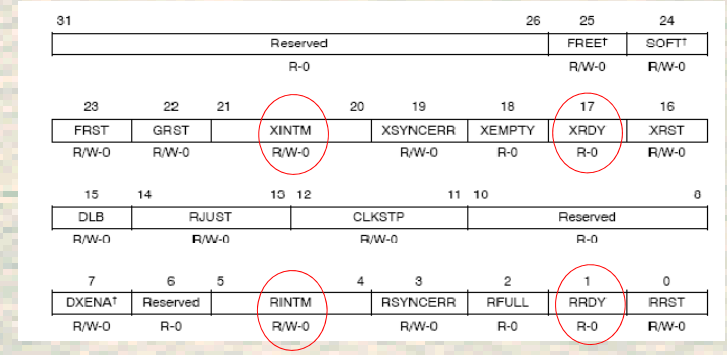
8

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



Signal Processing

SPCR

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

11

Signal Processing

SPCR

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

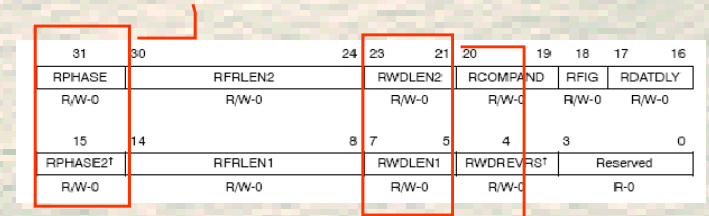
Once that data has been read by either the CPU or the DMA/EDMA controller, RRDY is cleared to 0.

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

12

RCR

Single phase or double phase (0, 1)



31	30 24	23 21	20	19 18	17 16
RPHASE	RFRLEN2	RWDLEN2	RCOMPAND	RFIG	RDATDLY
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14 8	7 5	4	3 0
RPHASE2†	RFRLEN1	RWDLEN1	RWDREVRS†	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	R-0

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

RCR

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.

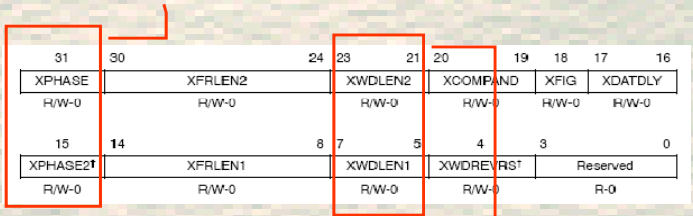
RCR

(R/X)WDLEN1/2	Element Length (Bits)
000	8
001	12
010	16
011	20
100	24
101	32
110	Reserved
111	Reserved

15

XCR

Single phase or double phase (0, 1)



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

16

Signal Processing

SRGR SAMPLE RATE GENERATOR REGISTER

31	30	29	28	27 ... 16
GSYNC	CLKSP	CLKSM	FSGM	FPER
R/W-0	R/W-0	R/W-1	R/W-0	R/W-0

15 ... 8	7 ... 0
FWID	CLKGDV
R/W-0	R/W-1

CLKGDV: SRG clock divider value.

FPER: Frame period value plus 1 specifies when the next frame sync signal becomes ready.

FSGM: Sample rate generator transmit frame synchronization bit.

CLKSM: Internal (CPU) or External (Pin) clock to derive.

17

Signal Processing

PCR PIN CONTROL REGISTER

31 ... 16
Reserved
R-0

15 ... 14	13	12	11	10	9	8
Reserved	XIOEN	RIOEN	FSXM	FSRM	CLKXM	CLKRM
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
Reserved†	CLKSSTAT	DXSTAT	DRSTAT	FSXP	FSRP	CLKXP	CLKRP
R-0	R-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0

CLKRM: Receiver clock mode bit (Input or Output).

CLKXM: Transmit clock mode bit (Input or Output).

FSRM: Receive frame synchronization mode bit (Internal 'by sample rate generator' or External).

FSXM: Transmit frame synchronization mode bit (Internal or External).

18

Volatile Variable

Signal Processing
© Dr. D. M. Akbar Hussain

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/BIOS GUI configuration tool.**

Writing directly to the serial port registers:

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

```
#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:

```
MCBSP_Handle hMcbsp;
```

(2) Open the serial port:

```
hMcbsp = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
```

Signal Processing

AALBORG UNIVERSITET ESBJERG

CSL programming procedure continues:

(3) Configure the serial port:

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

(4) Close the Serial Port after use:

```
MCBSP_close(hMcbsp);
```

25

Signal Processing

AALBORG UNIVERSITET ESBJERG

DSP/BIOS GUI Interface:

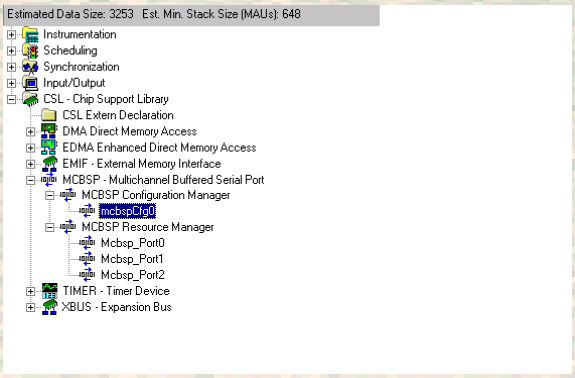
- With this method the configuration structure is created graphically and the setup code is generated automatically.

26

Signal Processing

Procedure:

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

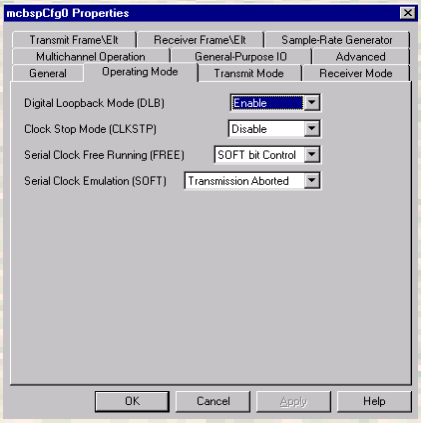


27

Signal Processing

Procedure:

Right click and select "Properties", see the figure below, and then select "Advanced" and fill all parameters as shown below:

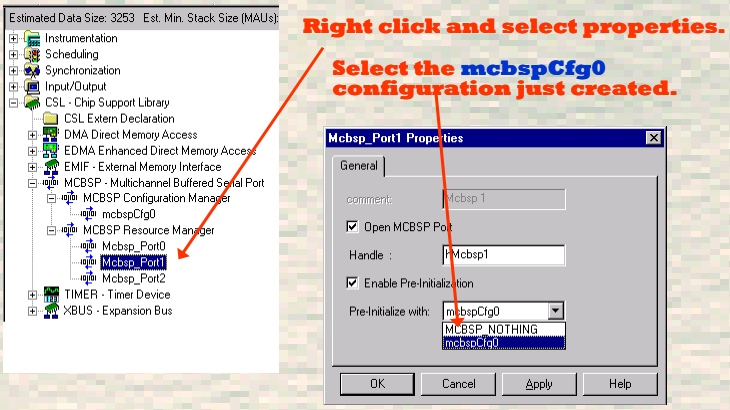


28

Signal Processing

Procedure:

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



29

Signal Processing

Procedure:

A file is then generated that contains the configuration code. The file generated for this example is shown on the next slide.

30

Signal Processing

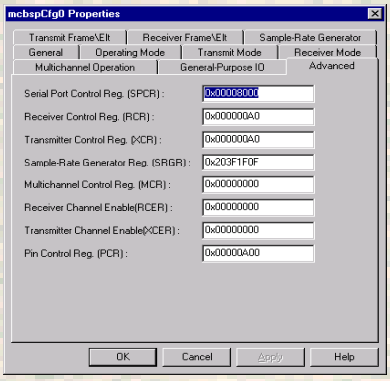

```
/*   Do *not* directly modify this file.  It was    */
/*   generated by the Configuration Tool; any  */
/*   changes risk being overwritten.               */

/* INPUT mcbsp1.cdb */

/*  Include Header File  */
#include "mcbsp1cfg.h"

/*  Config Structures */
MCBSP_Config mcbspCfg0 = {
    0x00008000,        /*  Serial Port Control Reg. (SPCR)   */
    0x000000A0,        /*  Receiver Control Reg. (RCR)   */
    0x000000A0,        /*  Transmitter Control Reg. (XCR)   */
    0x203F1F0F,        /*  Sample-Rate Generator Reg. (SRGR)   */
    0x00000000,        /*  Multichannel Control Reg. (MCR)   */
    0x00000000,        /*  Receiver Channel Enable(RCER)   */
    0x00000000,        /*  Transmitter Channel Enable(XCER)   */
    0x00000A00         /*  Pin Control Reg. (PCR)   */
};

/*  Handles  */
MCBSP_Handle hMcbsp1;

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

31

Signal Processing

Notice:

Values in the code generated are the same as the values inserted using the GUI interface.




```
/*   Do *not* directly modify this file.  It was    */
/*  Config Structures */
MCBSP_Config mcbspCfg0 = {
    0x00008000,        /*  Serial Port Control Reg. (SPCR)   */
    0x000000A0,        /*  Receiver Control Reg. (RCR)   */
    0x000000A0,        /*  Transmitter Control Reg. (XCR)   */
    0x203F1F0F,        /*  Sample-Rate Generator Reg. (SRGR)   */
    0x00000000,        /*  Multichannel Control Reg. (MCR)   */
    0x00000000,        /*  Receiver Channel Enable(RCER)   */
    0x00000000,        /*  Transmitter Channel Enable(XCER)   */
    0x00000A00         /*  Pin Control Reg. (PCR)   */
};
```

32

Signal Processing


```
MCBSP_Config mcbspCfg1 = {
    0x00000000,         /*  Serial Port Control Reg. (SPCR)    */
    0x00000000,         /*  Receiver Control Reg. (RCR)    */
    0x00010080,         /*  Transmitter Control Reg. (XCR)    */
    0x30180002,         /*  Sample-Rate Generator Reg. (SRGR)    */
    0x00000000,         /*  Multichannel Control Reg. (MCR)    */
    0x00000000,         /*  Receiver Channel Enable(RCER)    */
    0x00000000,         /*  Transmitter Channel Enable(XCER)    */
    0x00000A00          /*  Pin Control Reg. (PCR)    */
};
```

33

Signal Processing


```
#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;
}
```

34

Signal Processing

AALBORG UNIVERSITET ESBJERG

```
/* Writing to McBSP */
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;
}
/* Reading from McBSP */
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_DRR;
          return input_reg;
}
main()
{
.................
}
```

For m Own Reference: MCBSP_DRR_DR_symval

35