INTERRUPTS & POLLING

I. Real-time data input/output handling

A. Interrupts - An interrupt is an event that stops the current process in the CPU so that the CPU can attend to the task needing completion because of the event.

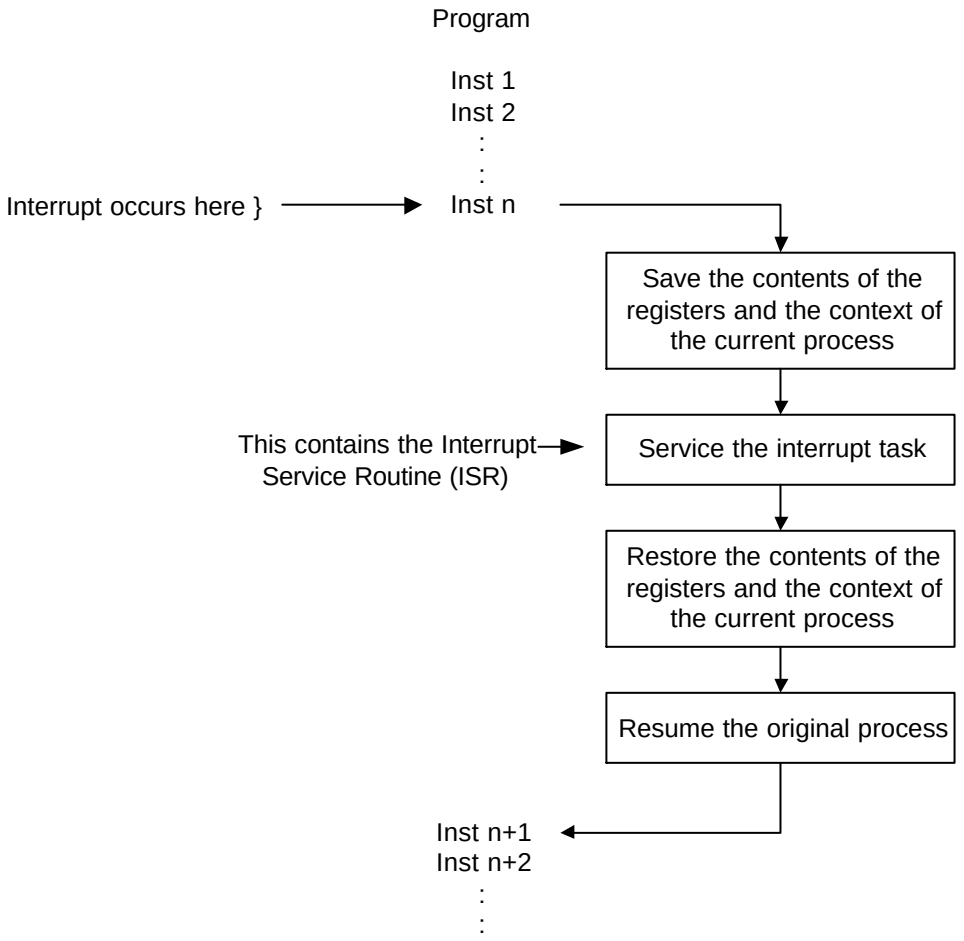
In data handling, an interrupt indicates data can be read or written to a device.

B. Polling - A polling-based program (non-interrupt driven) continuously polls or tests whether or not data are ready to be received or transmitted. This scheme is less efficient than the interrupt scheme.

II. Interrupts

A. Interrupts and the '6711

1. The general process:
Servicing an interrupt involves saving the context of the current process, completing the interrupt task (interrupt service routine), restoring the registers and the process context, and resuming the original process.



2. Types of interrupts on the ‘C6000 CPUs

RESET	Highest priority
NMI	
INT4	
INT5	
INT6	
INT7	
INT8	
INT9	
INT10	
INT11	
INT12	
INT13	
INT14	
INT15	Lowest priority

The RESET signal resets the CPU and has the highest priority.

NMI, the nonmaskable interrupt, is the second–highest priority interrupt and is generally used to alert the CPU of a serious hardware problem such as an imminent power or memory failure. [A nonmaskable interrupt means that the interrupt cannot be ignored or disabled by the system.]

INT4-INT15 are maskable interrupts which means that the processor can mask or temporarily ignore the interrupt if it needs to so it can finish something else that it is doing. These interrupts can be associated with external devices, on–chip peripherals (timers), software control, or not be available. Have to look at the user manual for the particular CPU.

3. There are eight registers in the ‘C6711 that control servicing interrupts.

Interrupt Control Registers

a. CSR (control status register): contains the global interrupt enable (GIE) bit and other control/status bits
b. IER (interrupt enable register): enables/disables individual interrupts
c. IFR (interrupt flag register): displays status of interrupts – if interrupt is pending
d. ISR (interrupt set register): sets pending interrupts
e. ICR (interrupt clear register): clears pending interrupts
f. ISTP (interrupt service table pointer): locates an ISR
g. IRP (interrupt return pointer)
h. NRP (nonmaskable interrupt return pointer)

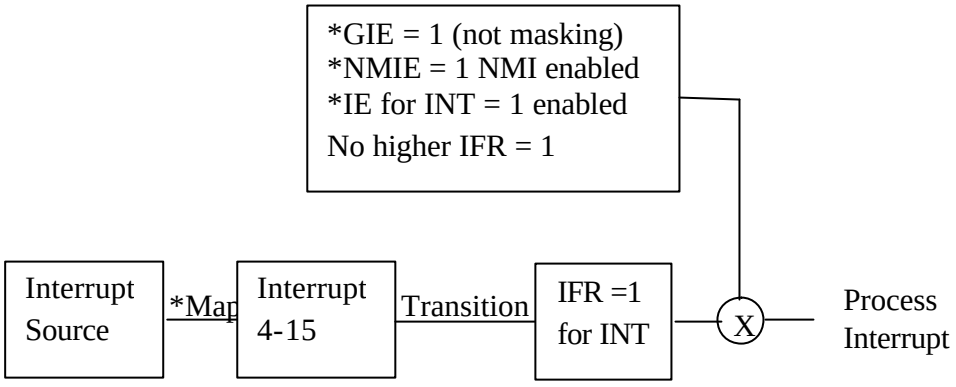
4. How a maskable interrupt is generated

An appropriate transition on an interrupt pin sets the pending status of the interrupt within the interrupt flag register (IFR). If the interrupt is properly enabled, the CPU begins processing the interrupt and redirecting program flow to the interrupt service routine.

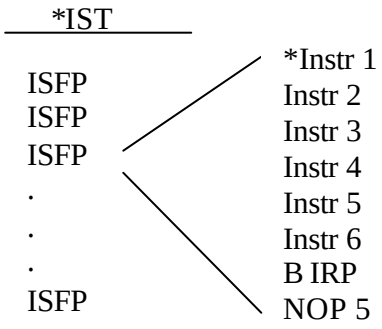
What does properly enabled mean?

Assuming that a maskable interrupt does not occur during the delay slots of a branch (this includes conditional branches that do not complete execution due to a false condition), the following conditions must be met to process a maskable interrupt:

- The global interrupt enable bit (GIE) bit in the control status register (CSR) is set to 1.
- The NMIE bit in the interrupt enable register (IER) is set to1.
- The corresponding interrupt enable (IE) bit in the IER is set to1.
- The corresponding interrupt occurs, which sets the corresponding bit in the IFR to 1 and there are no higher priority interrupt flag (IF) bits set in the IFR.



5. How an interrupt is processed



When the CPU begins processing an interrupt, the interrupt service table (IST) is used. The IST is a table of fetch packets that contain code for servicing the interrupts. The IST consists of 16 consecutive fetch packets. Each interrupt service fetch packet (ISFP) contains eight instructions used to service the interrupt. One of the instructions is a branch to the interrupt return pointer instruction (B IRP).

If the interrupt service routine for an interrupt is too large to fit in a single FP, a branch to the location of additional interrupt service routine code is required.

The IST is located between 0h and 200h, = 16 fetch packets in the table, each fetch packet = 8 instructions * 32 bits/instruction or = 32 bytes. 16*32=512=200h

The reset FP must be at address 0. However, the FPs associated with the other interrupts can be relocated. The relocatable address can be specified by using the interrupt service table pointer (ISTP) register.

* indicates the things we need to do as shown in the rest of this section

6. Mapping interrupt sources to CPU interrupts

There are 16 interrupt sources, each with a selection number. The CPU, however, has 12 interrupts available for use. So an interrupt source must be mapped to a CPU interrupt. This is done by setting appropriate bits of the two memory mapped Interrupt Multiplex Registers.

TMS320C621x/C671x Available Interrupts

Interrupt Selection Number	Interrupt Acronym	Interrupt Description
00000b	DSPINT	Host port host to DSP interrupt
00001b	TINT0	Timer 0 interrupt
00010b	TINT1	Timer 1 interrupt
00011b	SD_INT	EMIF SDRAM timer interrupt
00100b	EXT_INT4	External interrupt 4
00101b	EXT_INT5	External interrupt 5
00110b	EXT_INT6	External interrupt 6
00111b	EXT_INT7	External interrupt 7
01000b	EDMA_INT	EDMA channel (0 through 15) interrupt
01001b	Reserved	Not used
01010b	Reserved	Not used
01011b	Reserved	Not used
01100b	XINT0	McBSP 0 transmit interrupt
01101b	RINT0	McBSP 0 receive interrupt
01110b	XINT1	McBSP 1 transmit interrupt
01111b	RINT1	McBSP 1 receive interrupt
other		Reserved

Table 14–5. Interrupt Selector Registers

Byte Address	Abbreviation	Name	Description	Section
019C0000h	MUXH	Interrupt multiplexer high	Selects which interrupts drive CPU interrupts 10–15 (INT10–15)	14.4.2
019C0004h	MUXL	Interrupt multiplexer low	Selects which interrupts drive CPU interrupts 4–9 (INT4–INT9)	14.4.2
019C0008h	EXTPOL	External interrupt polarity	Sets the polarity of the external interrupts (EXT_INT4–EXT_INT7)	14.4.1

Interrupt Multiplexer High (INT10 - INT15) (address 0x19c0000)

	INTSEL15		INTSEL14		INTSEL13		INTSEL12		INTSEL11		INTSEL10

Interrupt Multiplexer Low (INT4 - INT9) (address 0x19c0004)

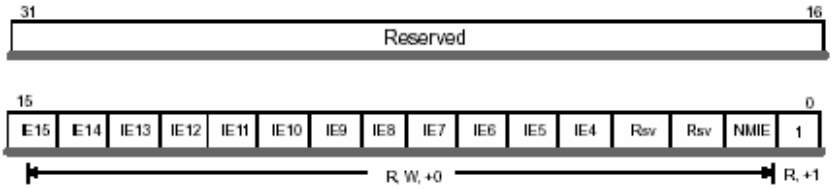
Example of code to initialize INT12 with 0101b:

```
MVKL   0x19c0000, A1
MVKH   0x19c0000, A1
LDW    *A1, A0
CLR    A0, 10, 13, A0
SET    A0, 10, 10, A0
SET    A0, 12, 12, A0
STW    A0, *A1
```

7. Set registers for GIE, NMIE, IE

Example to enable Individual Interrupts

Interrupt Enable Register (IER)



Legend: R = Readable by the MVC instruction
W = Writeable by the MVC instruction

Enabling Interrupt 7

In C:

```
#include <c6x.h> /* IER defined here */
void enable_INT7 (void)
{
    IER = IER | 0x080;
}
```

In ASM:

```
_asm_set_INT7
    MVC .S2 IER, B0
    SET .L2 B0,7, 7, B0
    MVC .S2 B0, IER
```

8. Interrupt service table – this is the vectors file

Vectors_11.asm Vector file for interrupt-driven program using INT11

```
        .ref    _c_int11        ;ISR used in C program
```

```
            .ref    _c_int00        ;entry address for a C program
            .sect   "vectors"       ;section for vectors
RESET_RST:  mvkl    .S2     _c_int00,B0     ;lower 16 bits --> B0
            mvkh  .S2   _c_int00,B0 ;upper 16 bits --> B0
            B       .S2     B0              ;branch to entry address
            NOP                     ;NOPs for remainder of FP
            NOP                     ;to fill 0x20 Bytes
            NOP
            NOP
            NOP
NMI_RST:    .loop 8
            NOP                     ;fill with 8 NOPs
            .endloop
RESV1:      .loop 8
            NOP
            .endloop
RESV2:      .loop 8
            NOP
            .endloop
INT4:       .loop 8
            NOP
            .endloop
INT5:       .loop 8
            NOP
            .endloop
INT6:       .loop 8
            NOP
            .endloop
INT7:       .loop 8
            NOP
            .endloop
INT8:       .loop 8
            NOP
            .endloop
INT9:       .loop 8
            NOP
            .endloop
INT10:      .loop 8
            NOP
            .endloop

INT11:      b       _c_int11                ;branch to ISR
            .loop 7
            NOP
            .endloop

INT12:      .loop 8
            NOP
            .endloop
INT13:      .loop 8
            NOP
            .endloop
INT14:      .loop 8
            NOP
            .endloop
INT15:      .loop 8
```

```
NOP
.endloop
```

9. C Program with the ISR

```
interrupt void c_int11()  //interrupt service routine
```

B. Example of Interrupts and the AD535 Codec

1. Files for use with the codec:

C6xdsk.cmd – sets up the memory map
C6x.h – defines some of the registers like IER, CSR in C:\ti\c6000\cgtools\include
C6x11dsk.h in C:\ti\c6000\dsk6x11\include – header file that defines addresses of external memory interface, the serial ports, etc IML, IMH registers defined here
C6xinterrupts.h – contains init functions for interrupt. Sets up registers.
C6xdskinit.h – header file with the function prototypes
C6xdskinit.c – functions to initialize the DSK, the codec, serial ports, and for input/output.

2. C6xinterrupts.h - excerpts

The interrupt selectors are defined

```
#define XINT0     0xC         /* 01100b  XINT0 McBSP 0 transmit interrupt */
#define RINT0     0xD         /* 01101b  RINT0 McBSP 0 receive interrupt */

*  Interrupt Initialization Functions

/* Enable Interrupts Globally (set GIE bit in CSR = 1)   */
void enableGlobalINT(void)
{
  CSR |= 0x1;    /* bitwise OR */
}

/* Enable NMI (non-maskable interrupt); must be enabled
 * or no other interrupts can be recognized by 'C6000 CPU */
void enableNMI(void)
{
  IER = _set(IER, 1, 1);       /* use an assembly intrinsinc */
}

/* Enable a specific interrupt;
 *  (INTnumber = {4,5,6,...,15})                    */
void enableSpecificINT(int INTnumber)
{
  IER = _set(IER, INTnumber, INTnumber);
}
```

The following does the mapping of interrupt source to CPU interrupt:

```
void config_Interrupt_Selector(int INTnumber, int INTsource)
{
   /* INTnumber = {4,5,6,...,15}
      INTsource = see #define list above
  */
```

3. C6xdskinit.h – header file with the function prototypes

```
//C6xdskinit.h Function prototypes for routines in c6xdskinit.c

void mcbsp0_init();   /* C callable library functions */
void mcbsp0_write(int);
int mcbsp0_read();
void TLC320AD535_Init();
void c6x_dsk_init();
void comm_poll();
void comm_intr();
int input_sample();
void output_sample(int);
```

4. C6xdskinit.c – functions to initialize the DSK, the codec, serial ports, and for input/output.

```
void comm_intr()                        //for communication/init using interrupt
{
  polling = 0;                                  //if interrupt-driven
  c6x_dsk_init();                       //call init DSK function
  config_Interrupt_Selector(11, XINT0); //using transmit interrupt INT11 when data in
buffer to be transmitted
  enableSpecificINT(11);              //for specific interrupt
  enableNMI();                          //enable NMI
  enableGlobalINT();                            //enable GIE for global interrupt
  mcbsp0_write(0);                              //write to SP0 – this will generate an
interrupt
}
```

5. Other programs

Vectors program with a branch for interrupt 11

C program with the ISR

```
//sine8_intr.c Sine generation using 8 points, f=Fs/(# of points)
//Comm routines and support files included in C6xdskinit.c

short loop = 0;
short sin_table[8] = {0,707,1000,707,0,-707,-1000,-707}; //sine values
short amplitude = 10;     //gain factor

interrupt void c_int11()  //interrupt service routine
{
 output_sample(sin_table[loop]*amplitude); //output each sine value – continues
to generate interrupt
 if (loop < 7) ++loop;    //increment index loop
 else loop = 0;           //reinit index @ end of buffer
 return;                  //return from interrupt
}

void main()
{
```

```
  comm_intr();          //init DSK, codec, McBSP  set-up for interrupt based
communications
  while(1);             //infinite loop
}
```

AD535 codec samples/writes at 8kHz. An interrupt occurs every sample period T=0.125ms. Within one period of the sine wave, 8 data values (0.125ms apart) are output to generate a sinusoidal signal. The period of the output signal is T=8*(0.125ms) = 1ms, frequency = 1kHz.

III. Polling

A. Register used for polling is the serial port control register (SPCR)

1. For reading, test bit 1 (second LSB) of the register to see if the port is ready to be read. This is done by doing an AND of the SPCR with 0x2.
2. For writing, AND SPCR with 0x20000 to test bit 17, the transmit ready bit.

B. Example of Polling and the Codec

1. c6xdskinit.c program

```
void comm_poll()                    //for communication/init using polling
{
   polling = 1;                     //if polling
   c6x_dsk_init();                  //call init DSK function
}


int input_sample()                  //added for input
{
   return mcbsp0_read();            //read from McBSP0
}


int mcbsp0_read()                   //function for reading
{
 int temp;

 if (polling)
 {
  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;  //test to see if receive ready
register is enabled
  while ( temp == 0)
    temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
 }
 temp = *(unsigned volatile int *)McBSP0_DRR;  //DRR = data read register
 return temp;
}
```

2. C program

```
//loop_poll.c Loop program using polling, output=input
//Comm routines and support files included in C6xdskinit.c

void main()
```

```
{
  int sample_data;

  comm_poll();                    //init DSK, codec, McBSP
  while(1)                                //infinite loop
  {
    sample_data = input_sample(); //input sample
    output_sample(sample_data);   //output sample
  }
}
```

IV. Code initialization

A. C programs

Programs start by going through a reset initialization code. This is in order to start at a defined initial location. Upon power-up, the system goes to the reset location in memory, which usually includes a branch to the beginning of code to be executed. This is accomplished through the use of an assembly language program.

Before you can run a C/C++ program, you must create the C/C++ run-time environment. The C/C++ boot routine performs this task using a function called c_int00. The run-time-support source library, rts.src, contains the source to this routine.

To begin running the system, the c_int00 function can be branched to or called, but it is usually vectored to by reset hardware. This is done in the vectors program which essentially upon an 'interrupt' of reset branches to c_int00.

The c_int00 function performs the following tasks to initialize the environment:

1) It defines a section called .stack for the system stack and sets up the initial stack pointers.
2) It initializes global variables.
3) It calls the function main to run the C/C++ program.

B. ASM programs

To initialize an assembly language program, you need the vectors program to branch to the name of the routine of the main program upon reset. For example,

Vectorsa.asm

```
;vectors_dotp4a.asm Vector file for dotp4a project

        .ref    init     ;starting addr in init file
        .sect       "vectors" ;in section vectors
rst:    mvkl .s2    init,b0   ;init addr 16 LSB -->B0
        mvkh .s2    init,b0   ;init addr 16 MSB -->B0
        b           b0        ;branch to addr init
        nop
        nop
        nop
        nop
        nop
```