D. Richard Brown III Associate Professor Worcester Polytechnic Institute Electrical and Computer Engineering Department drb@ece.wpi.edu

29-October-2009

ECE4703 REAL-TIME DSP INTERFACING WITH I/O, DEBUGGING, AND PROFILING



Interfacing a DSP With the Real World





DIP Switches and LEDs

LED and DIP switch interface functions are provided in dsk6713bsl.lib.

Initialize the DSK with the BSL function DSK6713_init();
Initialize DIP/LEDs with DSK6713_DIP_init() and/or DSK6713_LED_init()
Read state of DIP switches with DSK6713_DIP_get(n)
Change state of LEDs with DSK6713_LED_on(n) or DSK6713_LED_off(n) or DSK6713_LED_toggle(n)
where n=0, 1, 2, or 3.

Documentation is available in C:\CCStudio_v3.1\docs\hlp\c6713dsk.hlp



AIC23 Codec

- AIC23 codec performs both ADC and DAC functions
- Stereo input and output (left+right channels)
- Initialization steps:
 - Initialize the DSK with the BSL function DSK6713_init();
 - Open the codec with the BSL function hCodec = DSK6713_AIC23_openCodec(0,&config);
 - "hCodec" is the codec "handle". You can think of this as a unique address of the codec on the McBSP bus.
 - "config" is the default configuration of the codec. See the header file dsk6713_aic23.h and the AIC23 codec datasheet (link on the course web page) for details.
 - Optional: Set the codec sampling frequency.
 - Configure the McBSP to transmit/receive 32 bits (two 16 bit samples) with the CSL function McBSP_FSETS()
 - Set up and enable interrupts



Codec Initialization Example (from Kehtarnavaz)

Initialization steps:	21 interrupt void serialPortRcvISR(void);	<pre>// ISR function prototype</pre>
	22	
I. Initialize the DSK	23 void main()	
2 Open the codec with	24	
the default	<pre>25 DSK6713_init(); // Initialize the board su</pre>	ipport library, must be called first
	<pre>26 hCodec = DSK6713_AIC23_openCodec(0, &config);</pre>	// Open the codec
configuration.	27	
3. Configure multi-	28 // Configure buffered serial ports for 32 bit ((This allows transfer of both right and laft)	operation
channel buffered serial	29 // INIS ALLOWS TRANSFER OF DOTH FIGHT ANA LEFT MCRED ESETS (SPCP1 PINTH EPN);	channels in one read/write
channel Duneled Serial	MCBSP_FSEIS(SPCR1, KINIM, FKM);	
port (MCBSP)	MCDSP_FSEIS(SPCKI, AINIM, FKM);	
 SPCR = serial port 	MCBSP_FSETS(KCKI, KMDLENI, SZBIT),	
control register	33 MCD3F_F3EF3(ACRI, ANDELNI, 32DIT);	
 RCR = receive control 	35 DSK6713 ATC23 setErea(bCodec DSK6713 ATC23 FR	FO 48KH7): // set the sampling rate
register	36	teq_former, // see the sumpting face
 XCR = transmit control 	37 // Interrupt setup	
register	38 IRO_alobalDisable(): // Globally di	sables interrupts
See SPRU508e.pdf	39 IRO_nmiEnable(); // Enables the	NMI interrupt
4. Set the sampling rate	40 IRQ_map(IRQ_EVT_RINT1,15); // Maps an eve	ent to a physical interrupt
	41 IRQ_enable(IRQ_EVT_RINT1); // Enables the	e event
5. Configure and enable	42 IRQ_globalEnable(); // Globally en	nables interrupts
interrupts	43	
6 Do normal processing	44 while(1)	
(us just anten a l	45 {	
(we just enter a loop	46 }	
here)	47 }	



AIC23 Codec: Interrupts

- We will use an interrupt interface between the DSP and the codec.
- OSP can do useful things while waiting for samples to arrive from codec, e.g. check DIP switches
- C6x interrupt basics:
 - Interrupt sources must be mapped to interrupt events
 - 16 "interrupt sources" (timers, serial ports, codec, ...)
 - 12 "interrupt events" (INT4 to INT15)
 - Interrupt events have associated "interrupt vectors". An "interrupt vector" is a special pointer to the start of the "interrupt service routine" (ISR).
 - Interrupt vectors must be set up in your code (usually in the file "vectors.asm").
 - You are also responsible for writing the ISR.



Interrupts

main code

interrupt source linked to interrupt event

interrupts enabled

interrupt event N occurs

(C compiler generates code to automatically save the state)

 \leftarrow

sinterrupt vector N

branch to interrupt service routine

interrupt service routine

do something useful

make sure the ISR completes before the next interrupt occurs

return to main code



Interrupt Vector

- We have linked the codec interrupt event to INTI5.
- The ISR in this example is called "serialPortRcvISR".
- The interrupt vector is usually in the vectors.asm file:

150	INT15:		
151	MVKL	.S2 _serialPortRcvISR, B	0
152	MVKH	.S2 _serialPortRcvISR, B	0
153	В	.S2 B0	
154	NOP		
155	NOP		
156	NOP		
157	NOP		
158	NOP		



A Simple Interrupt Service Routine

49	interrupt void serialPortRcvISR()	
50	{	
51	Uint32 temp;	
52		
53	<pre>temp = MCBSP_read(DSK6713_AIC23_DATAHANDLE);</pre>	// read L+R channels
54	<pre>MCBSP_write(DSK6713_AIC23_DATAHANDLE,temp);</pre>	// write L+R channels
55	}	

Remarks:

- MCBSP_read() requests L+R samples from the codec's ADC
- MCBSP_write() sends L+R samples to the codec's DAC
- This ISR simply reads in samples and then sends them back out.



Setting the Codec Sampling Frequency

Here we open the codec with the default configuration:

26 hCodec = DSK6713_AIC23_openCodec(0, &config);

// Open the codec

7

The structure "config" is declared in dsk6713_aic23.h

Rather than editing the default configuration in the header file, we can change the sampling frequency after the initial configuration:

DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ); // set the sampling rate Frequency definitions are in dsk6713_aic.h

∕* Frequ	ency Definitions */
#define	DSK6713_AIC23_FREQ_8KHZ
#define	DSK6713_AIC23_FREQ_16KHZ
#define	DSK6713_AIC23_FREQ_24KHZ
#define	DSK6713_AIC23_FREQ_32KHZ
#define	DSK6713_AIC23_FREQ_44KHZ
#define	DSK6713_AIC23_FREQ_48KHZ
#define	DSK6713_AIC23_FREQ_96KHZ



Other Codec Configuration

- Line input volume level (individually controllable for left and right channels)
- Headphone output volume level (individually controllable for left and right channels)
- Digital word size (16, 20, 24, or 32 bit)
- Other settings, e.g. byte order, etc. For more details, see:
 - dsk6713_aic23.h
 - AIC23 codec datasheet (link on course web page)
 - C:\CCStudio_v3.1\docs\hlp\c6713dsk.hlp



Codec Data Format and How To Separate the Left/Right Channels

// we can use the union construct in C to have
// the same memory referenced by two different variables
union {Uint32 combo; short channel[2];} temp;

temp.channel[0] (short) temp.channel[1] (short)

temp.combo (Uint32)

// the McBSP functions require that we
// read/write data to/from the Uint32 variable
temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);

// but if we want to access the left/right channels individually
// we can do this through the short variables
Leftchannel = temp.channel[1];
Rightchannel = temp.channel[0];



Final Remarks on DSP/Codec Interface

- In real-time DSP applications, we will process samples as they become available from the codec's ADC.
- This means that all processing will be done in the ISR.
- The ISR must run in real-time, i.e. the total execution time must be less than one sampling period.
- You can do DIP/LED processing outside of the ISR (in your main code).
- Look at Kehtarnavaz Lab 2 for examples.



C6713 DSK Memory Architecture

- TSM320C6713 DSP chip has 256kB internal SRAM
 - Up to 64kB of this SRAM can be configured as shared L2 cache
- OSK provides additional I6MB external RAM (SDRAM)
- OSK also provides 512kB external FLASH memory
- Code location (.text in linker command file)
 - internal SRAM memory (fast)
 - external SDRAM memory (typically 2-4x slower, depends on cache configuration)
- Data location (.data in linker command file)
 - internal SRAM memory (fast)
 - external SDRAM memory (slower, depends on datatypes and cache configuration)
- Code+data for all projects assigned in this class should fit in the C6713 internal SRAM



TMS320C6713 DSK Memory Map

0000 0000 0003 FFFF	Internal SRAM (256kB)	L2 Memory E	Block Base Address
		your code+data here	0x0000 0000
8000 0000			
	External SDRAM (16MB)		
8FFF FFFF			
8000 0000	ΓΙ Δ S Η	16K-Byte RAM	0x0003 0000
8007 FFFF		16K-Byte RAM	0x0003 4000
		16K-Byte RAM	0x0003 8000
FFFF FFFF		16K-Byte RAM	0x0003 C000 0x0003 FFFF



Linker Command File Example

MORY			
vecs:	o =	00000000h	1 = 00000200h
IRAM:	0 =	00000200h	1 = 0002FE00h
CE0:	O =	80000000h	1 = 0100000h
CTIONS			
.vectors	>	vecs	Çode goes here
.cinit	>	IRAM	
.text	>	IRAM	
.stack	>	IRAM	
.bss	>	IRAM /	
.const	>	IRAM 🖌	Addresses 00000000-0002EEEE correspond to the lowest
.data	>	IRAM	192kB of internal memory (SRAM) and are labeled "IRAM".
.far	>	IRAM	
.switch	>	IRAM	External memory is mapped to address range 8000000 –
.sysmem	>	IRAM	80FFFFFF. This is 16MB and is labeled "CEO".
.tables	>	IRAM	
.cio	>	IRAM	Both code and data are placed in the $C6713$ internal SRAM in
			this example. Interrupt vectors are also in SRAM.
	MORY vecs: IRAM: CE0: CTIONS .vectors .cinit .text .stack .bss .const .data .far .switch .sysmem .tables .cio	MORY VeCS: 0 = IRAM: 0 = CEO: 0 = CEO: 0 = CTIONS .vectors > .cinit > .text > .stack > .stack > .stack > .const > .const > .data > .far > .switch > .sysmem > .tables > .cio >	MORY vecs: o = 0000000h IRAM: o = 00000200h CE0: o = 8000000h CE0: vectors > vecs .cinit > IRAM .text > IRAM .text > IRAM .stack > IRAM .bss > IRAM .const > IRAM .data > IRAM .far > IRAM .far > IRAM .switch > IRAM .sysmem > IRAM .tables > IRAM .cio > IRAM





vectors.asm

- This file contains your interrupt vectors
- ".sect" directive at top of file tells linker where (in memory) to put the code
- Each interrupt vector is composed of exactly 8 assembly language instructions

```
• Example:
```

```
INT15:

MVKL .S2 _serialPortRcvISR, B0

MVKH .S2 _serialPortRcvISR, B0

B .S2 B0

NOP

NOP

NOP

NOP

NOP

NOP
```



Debugging and Other Useful Features of the CCS IDE

- Breakpoints
- Probe points
- Watch variables
- Plotting arrays of data
- Animation
- General Extension Language (GEL)



Breakpoints and Probe Points



- Breakpoints: stop code execution at this point to allow state examination and step-by-step execution.
- Probe points: force window updates and/or read/write samples from/to a file at a specific point in your code.



Breakpoints





Watch Variables



	Name	Value		Туре	Radix
	1				
.					
1	🔏 Watch Locals 🚳 🕅	atch 1			
			Ln 24, Col 1		
-					
	Name	Value		Туре	Radix
	🛛 🖗 loop	4		short	dec
	💊 gain	10		short	dec
	<u> </u>				
	🎉 Watch Locals 🛛 💞 🗰	atch 1			
			Ln 24, Col 1	NUM	

Watch Variables

- In the Watch Locals tab, the debugger automatically displays the Name, Value, and Type of the variables that are *local* to the currently executing function.
- In the Watch tab, the debugger displays the Name, Value, and Type of the local and global variables and expressions that you specify.
- Can add/delete tabs.



Plotting Arrays of Data

😻 /C6713 DSK/CPU_1 - C67xx - Code Composer Studio 'C6713 D									
🌾 File Edit	View Project	Debug	Profiler	GEL	Option	Tools	PBC I		
12 🖻 🔛	✓ Standard To GEL Toolbar	olbar		di	Р				
Sine8_LED.pit	 Project Toolb Edit Toolbar 	Dar			•	ڬ 😫	₩ ₹		
疑 667	✓ Status Bar			6	₩ ₫	b Q			
🕑 👰 File	Plug-in Toolb	ars ars		ir he	it32 f	`s = loon	DSK6 = O		
₽ ₽	Disassembly			hc	rt ga	in =	10;		
{}•	Memory				rt sı	ne_t	apie		
(1)	Registers			▶ pi	d mai	n ()			
-{}*	Graph			•	Time/Fred Constellat	juency	· 11		
->0	Threads			►	Eye Diagr	am	1: i		
{ \ }	Watch Windo	W			Image				
ž	Call Stack			ł	f(DSK	6713	_DIP		
X	Expression L	ist		{	. ` 	710]		
2		IOW		-	outp	0713_ ut_s	ampl		
ōx	✓ Project	1		-	if (loop	< 7 n =		
	Mixed Source	e/ASM		-)	- 	. 100	r -		
<u>弱</u>	Real-time Re	fresh Op	tions	L	else D	SK67	13_L:		

💀 Graph Property Dialog	×
Display Type	Single Time
Graph Title	Graphical Display
Start Address	sine_table
Acquisition Buffer Size	8
Index Increment	1
Display Data Size	8
DSP Data Type	16-bit signed integer
Q-value	0
Sampling Rate (Hz)	8000
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	8
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Line
Grid Style	Zero Line
Cursor Mode	Data Cursor
P	<u>O</u> K <u>C</u> ancel <u>H</u> elp



Graph Windows: Plotting Arrays of Data





Profiling Your Code and Making it More Efficient

It is the stimate the execution time of your code.

Output: A set of the optimizing compiler to produce more efficient code.

 Other factors affecting the efficiency of your code.



How to estimate code execution time when connected to the DSK

- I. Start CCS with the C6713 DSK connected
- 2. Debug -> Connect (or alt+C)
- 3. Open project, build it, and load .out file to the DSK
- 4. Open the source file you wish to profile
- 5. Set two breakpoints for the start/end of the code range you wish to profile
- 6. Profile -> Clock -> Enable
- 7. Profile -> Clock -> View
- 8. Run to the first breakpoint
- 9. Reset the clock
- 10. Run to the second breakpoint
- II. Clock will show raw number of execution cycles between breakpoints.



Tip: You can save your breakpoints, probe points, graphs, and watch windows with

File -> Workspace -> Save Workspace As



Another method for estimating code execution time (part 1 of 3)

Repeat steps I-4 previous method.

- Clear any breakpoints in your code
- 6. Profile -> Setup
- 7. Click on Custom tab
- 8. Select "Cycles"
- 9. Click on clock (enable profiling)

🍈 🗘 🕼 😼 🕼 🗠 🖬 Lab04.out	
Renable/Disable Profiling	
r 🗌 Branches	
Cycles	
ExecutionPacket	
🗌 🗆 InterruptAcknowledge	
InterruptContextSwitch	
L1DCleanDirtyVictimsReplace	
L1DDirtyVictimsReplaceL2	
	•
Time	
Activities Ranges Control Custom	



Another method for estimating code execution time (part 2 of 3)

10. Select Ranges tab

- II. Highlight code you want to profile and drag into ranges window (hint: you can drag whole functions into this window)
- 12. Repeat for other ranges if desired

💽 🛟 🗘 📢 😼 🔛 🗠 Lab04.out		
Range Type	Source	Address
Functions		
Enabled	64-82:dsk_fir.c	0x3e4-0x6
I⊞ ····· Ranges		
Activities Ranges Control C	ustom	



Another method for estimating code execution time (part 3 of 3)

13. Profile -> Viewer

- 14. Run (let it run for a minute or more)
- 15. Halt
- 16. Observe profiling results in Profile Viewer window

Profi	Profile Viewer << 0 >> Current - C6713 D5K/CPU_1							
	Address Range	Symbol Name	SLR	Symbol Type	Access Count	Cycles: Incl. Avg.	Cycles: Excl. Avg.	
→ ≹ 3	0:0x3e4-0x670	serialPortRcvISR	64-82:dsk_fir.c	function	49	464	392	
* ♀					\smile	\smile	\smile	
→ ¢								
+1								
	Llinter	dit the colum						
	Hint: ed	ait the colun	ins to see av	rerages or m	naximums			
8								
2	Profiler							
NYTECHNIC								

What does it mean?

- Access count is the number of times that CCS profiled the function
 - Note that the function was probably called more than 49 times. CCS only timed it 49 times.
- Inclusive average is the average number of cycles needed to run the function including any calls to subroutines
- Exclusive average is the average number of cycles needed to run the function excluding any calls to subroutines



Optimizing Compiler



Build Options f	or Dotp4.pjt	2 🛛							
General Compiler Linker Link Order									
-g -s -o3 -fr''C:\ti\	vmyprojects\Dotp4\Deb	ug'' -d''CHIP_6713'' -mv6710 🗾							
Category: Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Basic Target Version: Generate Debug Info: Opt Speed vs Size: Opt Level: Program Level Opt.:	C671x (-mv6710) Full Symbolic Debug (-g) Speed Most Critical (no -ms) Speed Most Critical (no -ms) Speed More Critical (-ms0) Speed Critical(-ms1) Size Critical (-ms2) Size Most Critical (-ms3)							
Category: Basic Advanced Feedback Files Assembly Parser Preprocessor Diagnostics	Basic Target Version: Generate Debug Info: Opt Speed vs Size: Opt Level: Program Level Opt.:	C671x (-mv6710) Full Symbolic Debug (-g) Speed Most Critical (no -ms) File (-o3) None Register (-o0) Local (-o1) Function (-o2) File (-o3)							



Profiling results after compiler optimization

 In this example, we get a 3x-4x improvement with "Speed Most Critical" and "File (-03)" optimization

• Optimization gains can be much larger, e.g. 20x

Profile Viewer << 0 >> Current - C6713 DSK/CPU_1								
-	Address Range	Symbol Name	SLR	Symbol Type	Access Count	Cycles: Incl. Avg.	Cycles: Excl. Avg.	
→	0:0x9a0-0xadc	serialPortRcvISR	64-82:dsk_fir.c	function	117	127	127	
* ♥					\smile	\smile	\smile	
→ [C)								
→II								
8								
7	Profiler							
POLYTECK								

Limitations of hardware profiling

- Breakpoint/clock profiling method may not work with compiler-optimized code
- Profile -> View method is known to be somewhat inaccurate when connected to real hardware (see "profiling limitations" in CCS help)
 - Accuracy is better when only one or two ranges are profiled
 - Best accuracy is achieved by running a simulator



Other factors affecting code efficiency

- Memory
 - Code location (.text in linker command file)
 - internal SRAM memory (fast)
 - external SDRAM memory (typically 2-4x slower, depends on cache configuration)
 - Data location (.data in linker command file)
 - internal SRAM memory (fast)
 - external SDRAM memory (slower, depends on datatypes and cache configuration)
- Data types
 - Slowest execution is double-precision floating point
 - Fastest execution is fixed point, e.g. short



TMS320C6000 C/C++Data Types

			Range		
Туре	Size	Representation	Minimum	Maximum	
char, signed char	8 bits	ASCII	-128	127	
unsigned char	8 bits	ASCII	0	255	
short	16 bits	2s complement	-32768	32767	
unsigned short	16 bits	Binary	0	65535	
int, signed int	32 bits	2s complement	-2147483648	214783647	
unsigned int	32 bits	Binary	0	4294967295	
long, signed long	40 bits	2s complement	-549755813888	549755813887	
unsigned long	40 bits	Binary	0	1099511627775	
enum	32 bits	2s complement	-2147483648	214783647	
float	32 bits	IEEE 32-bit	1.175494e-38†	3.40282346e+38	
double	64 bits	IEEE 64-bit	2.22507385e-308†	1.79769313e+308	
long double	64 bits	IEEE 32-bit	2.22507385e-308†	1.79769313e+308	



Final Remarks

You should have enough information to complete Lab I

- Refer to Lab 2 example code and discussions in Kehtarnavaz
- Lecture notes
- Reference material noted in lecture notes
- Please make sure you understand what you are doing. Don't just copy and paste from Kehtarnavaz.
- Lab I Part 3: Level Meter
 - Important practical consideration in real DSP systems
 - The goal is to use the full range of the ADC and DAC but avoid clipping (clipping = very bad nonlinear distortion)
 - Your level meter code may be useful for later projects

