

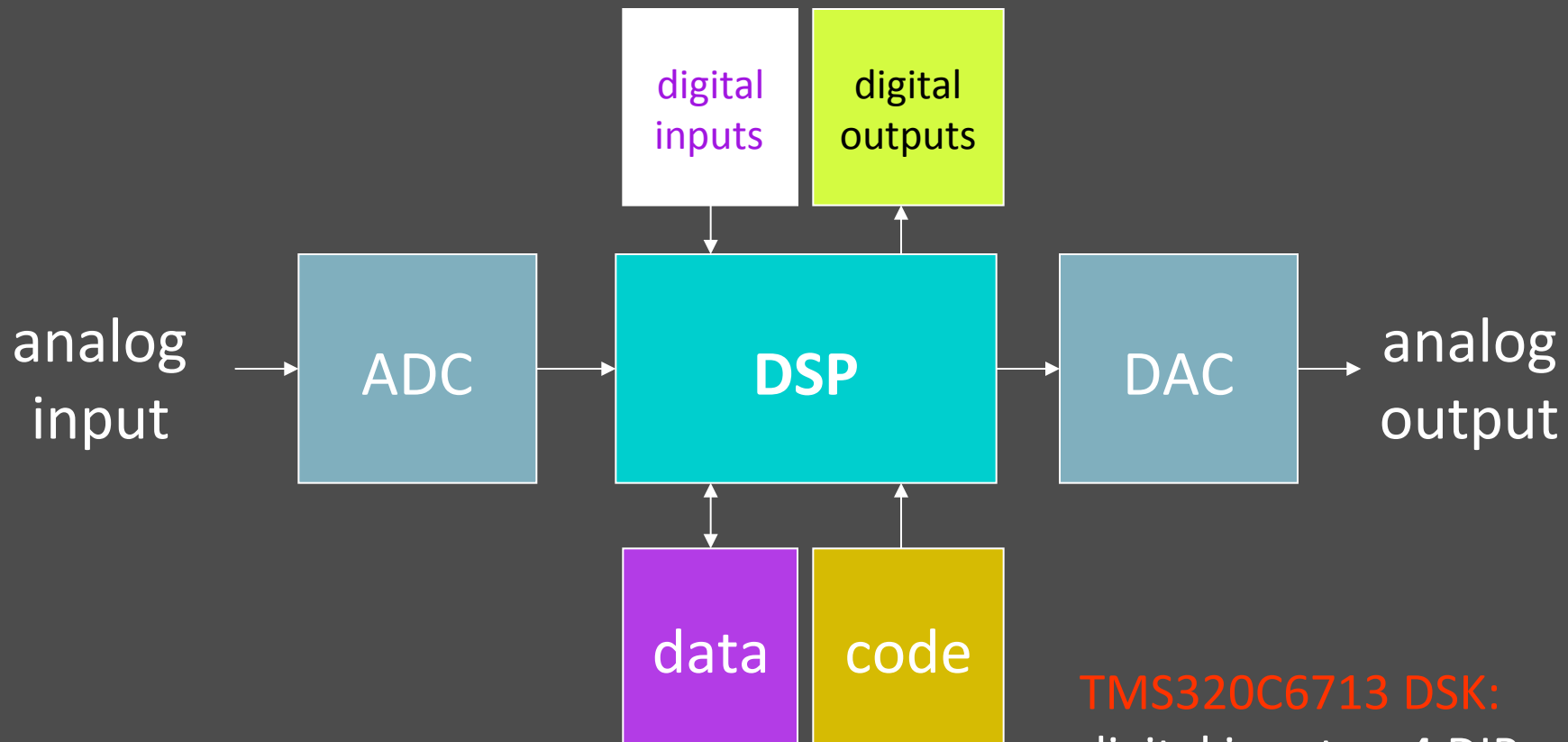
D. Richard Brown III
Associate Professor
Worcester Polytechnic Institute
Electrical and Computer Engineering Department
drb@ece.wpi.edu

27-October-2011

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



Interfacing a DSP With the Real World



TMS320C6713 DSK:

digital inputs = 4 DIP switches

digital outputs = 4 LEDs

ADC and DAC = AIC23 codec

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 **Board Support Library API** (on course website).



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:

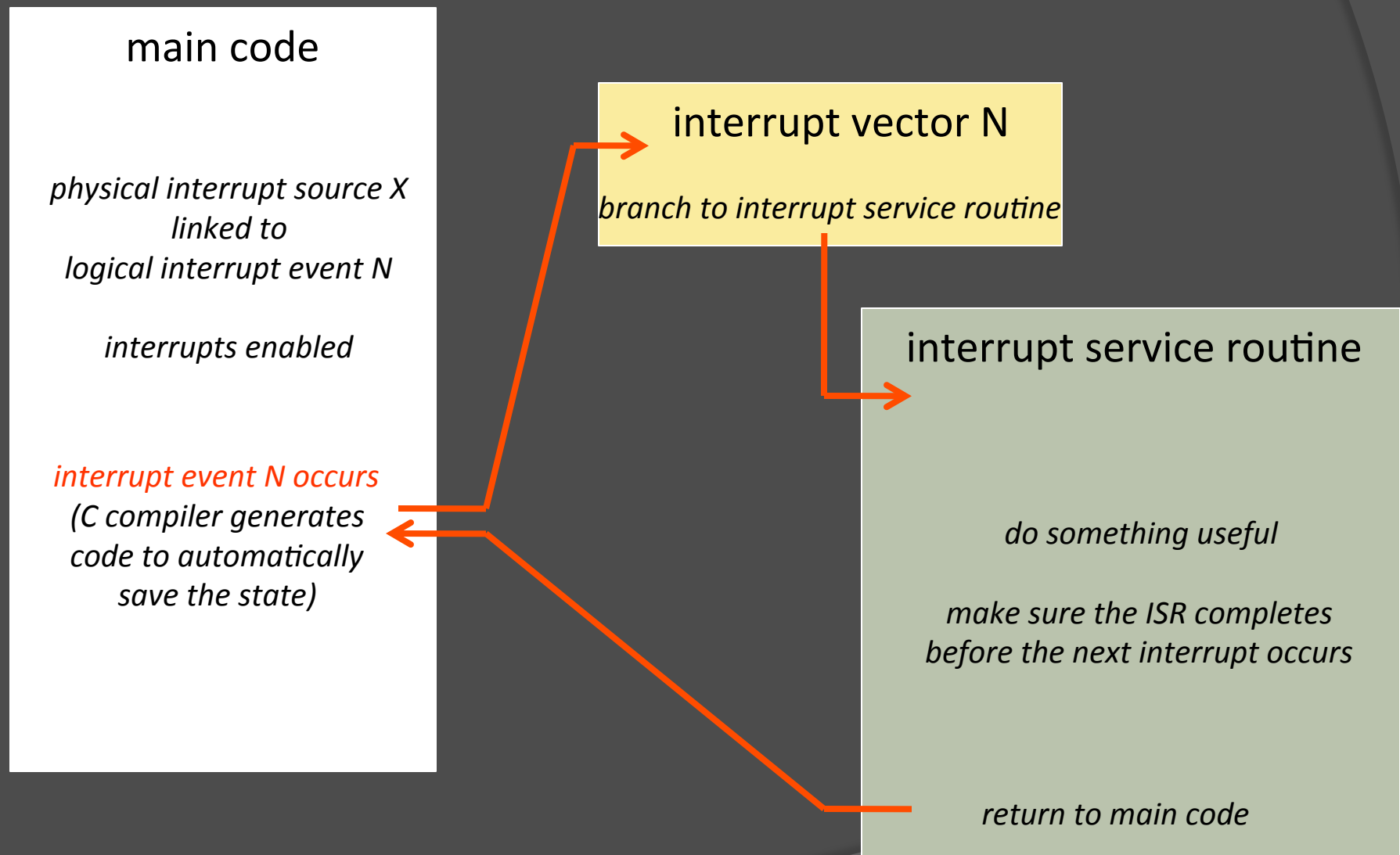
1. Initialize the DSK
2. Open the codec with the default configuration.
3. Configure multi-channel buffered serial port (McBSP)
 - SPCR = serial port control register
 - RCR = receive control register
 - XCR = transmit control register
 - See [SPRU508e.pdf](#)
4. Set the sampling rate
5. Configure and enable interrupts
6. Do normal processing (we just enter a loop here)

```
21 interrupt void serialPortRcvISR(void);           // ISR function prototype
22
23 void main()
24 {
25     DSK6713_init();    // Initialize the board support library, must be called first
26     hCodec = DSK6713_AIC23_openCodec(0, &config);    // Open the codec
27
28     // Configure buffered serial ports for 32 bit operation
29     // This allows transfer of both right and left channels in one read/write
30     McBSP_FSETS(SPCR1, RINTM, FRM);
31     McBSP_FSETS(SPCR1, XINTM, FRM);
32     McBSP_FSETS(RCR1, RWDLEN1, 32BIT);
33     McBSP_FSETS(XCR1, XWDLEN1, 32BIT);
34
35     DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ);    // set the sampling rate
36
37     // Interrupt setup
38     IRQ_globalDisable();    // Globally disables interrupts
39     IRQ_nmiEnable();    // Enables the NMI interrupt
40     IRQ_map(IRQ_EVT_RINT1, 15);    // Maps an event to a physical interrupt
41     IRQ_enable(IRQ_EVT_RINT1);    // Enables the event
42     IRQ_globalEnable();    // Globally enables interrupts
43
44     while(1)
45     {
46     }
47 }
```

AIC23 Codec: Interrupts

- ⦿ We will use an **interrupt interface** between the DSP and the codec.
- ⦿ DSP 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 physical “interrupt sources” (timers, serial ports, codec, ...)
 - 12 logical “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



Interrupt Vector

- We usually link the physical **codec interrupt** to **INT15**.
- The ISR in this example is called “**serialPortRcvISR**” (you can rename it if you like).
- C function “x” is called “_x” in ASM files.
- The interrupt vector is usually in the **vectors.asm** file:
- Each interrupt vector must be exactly 8 ASM instructions

```
150 INT15:
151     MVKL .S2 _serialPortRcvISR, B0
152     MVKH .S2 _serialPortRcvISR, B0
153     B     .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     temp = MCBSP_read(DSK6713_AIC23_DATAHANDLE); // read L+R channels  
54     MCBSP_write(DSK6713_AIC23_DATAHANDLE,temp); // 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
```

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:

```
35 DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_48KHZ); // set the sampling rate
```

Frequency definitions are in `dsk6713_aic.h`

```
/* Frequency Definitions */
#define DSK6713_AIC23_FREQ_8KHZ      1
#define DSK6713_AIC23_FREQ_16KHZ    2
#define DSK6713_AIC23_FREQ_24KHZ    3
#define DSK6713_AIC23_FREQ_32KHZ    4
#define DSK6713_AIC23_FREQ_44KHZ    5
#define DSK6713_AIC23_FREQ_48KHZ    6
#define DSK6713_AIC23_FREQ_96KHZ    7
```

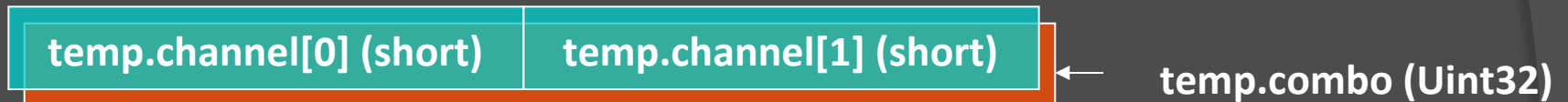
*This is actually
44.1kHz*

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)

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;
```



```
// 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];
```

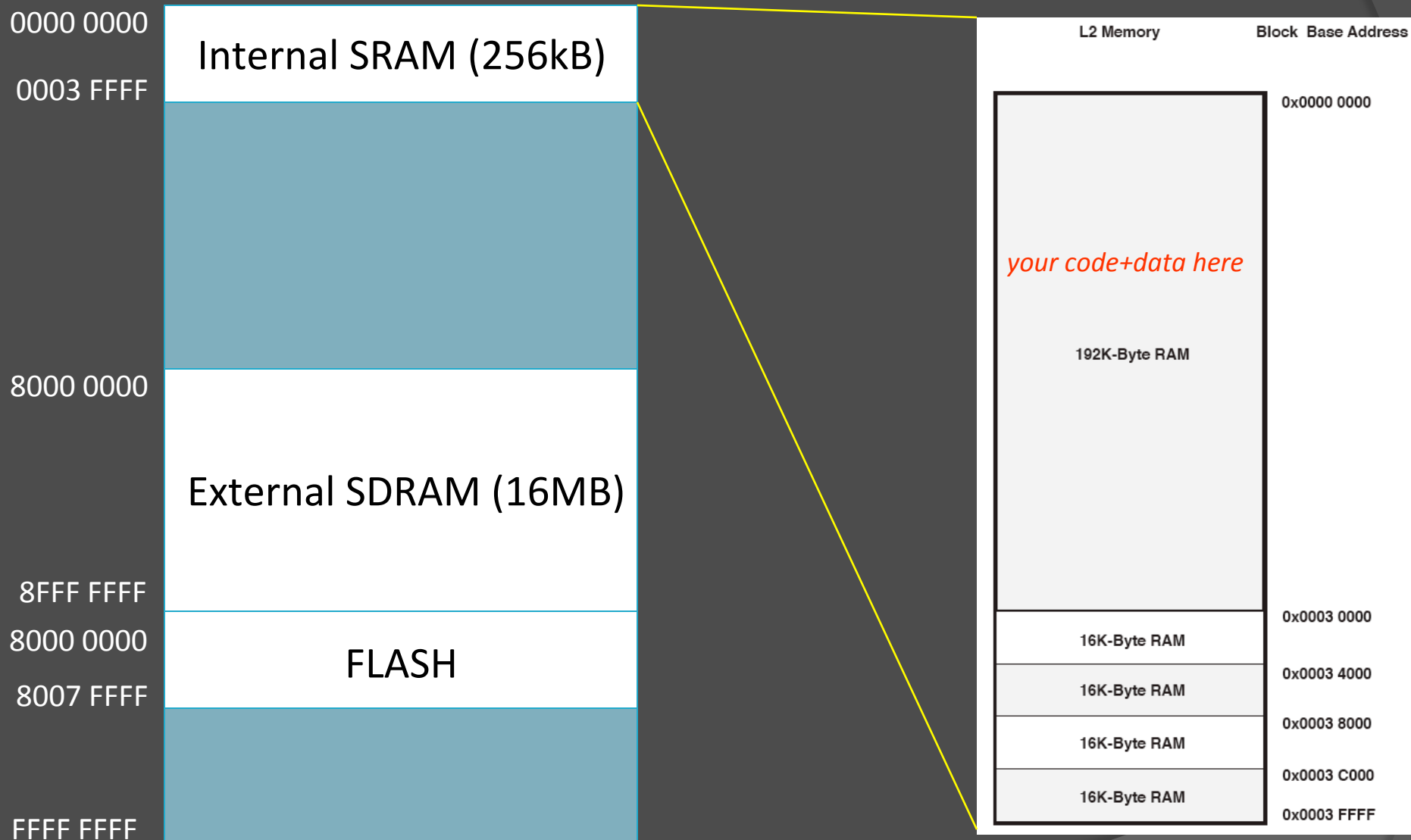
Final Remarks on DSP/Codec Interface

- In most real-time DSP applications, you process samples as they become available from the codec's ADC (sample-by-sample operation).
- This means that **all processing will be done in the ISR.**
 - **MCBSP_read()**
 - **--- processing here ---**
 - **MCBSP_write()**
- 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).

C6713 DSK Memory Architecture

- ◎ TSM320C6713 DSP chip has 256kB internal SRAM
 - Up to 64kB of this SRAM can be configured as shared L2 cache
- ◎ DSK provides additional 16MB external RAM (SDRAM)
- ◎ DSK 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 ECE4703 should fit in the C6713 internal SRAM

TMS320C6713 DSK Memory Map



Linker Command File Example

MEMORY

```
{  
    vecs:      o = 00000000h    l = 00000200h  
    IRAM:      o = 00000200h    l = 0002FE00h  
    CE0:       o = 80000000h    l = 01000000h  
}
```

SECTIONS

```
{  
    .vectors    >    vecs  
    .cinit      >    IRAM  
    .text       >    IRAM  
    .stack      >    IRAM  
    .bss        >    IRAM  
    .const      >    IRAM  
    .data       >    IRAM  
    .far        >    IRAM  
    .switch     >    IRAM  
    .sysmem     >    IRAM  
    .tables     >    IRAM  
    .cio        >    IRAM  
}
```

Code goes here

Data goes here

Addresses 00000000-0002FFFF correspond to the lowest 192kB of internal memory (SRAM) and are labeled "IRAM".

External memory is mapped to address range 80000000 – 80FFFFFF. This is 16MB and is labeled "CE0".

Both code and data are placed in the C6713 internal SRAM in this example. Interrupt vectors are also in SRAM.

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
```

Debugging and Other Useful Features of the CCS IDE

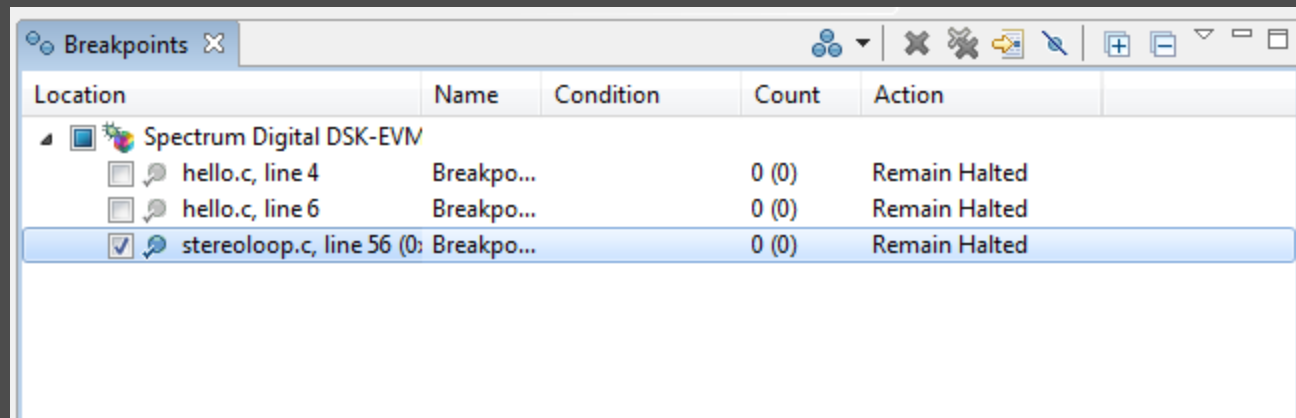
- ⦿ Breakpoints and stepping through your code
- ⦿ Watch variables
- ⦿ Registers
- ⦿ Plotting arrays of data

Breakpoints: Just Double-Click

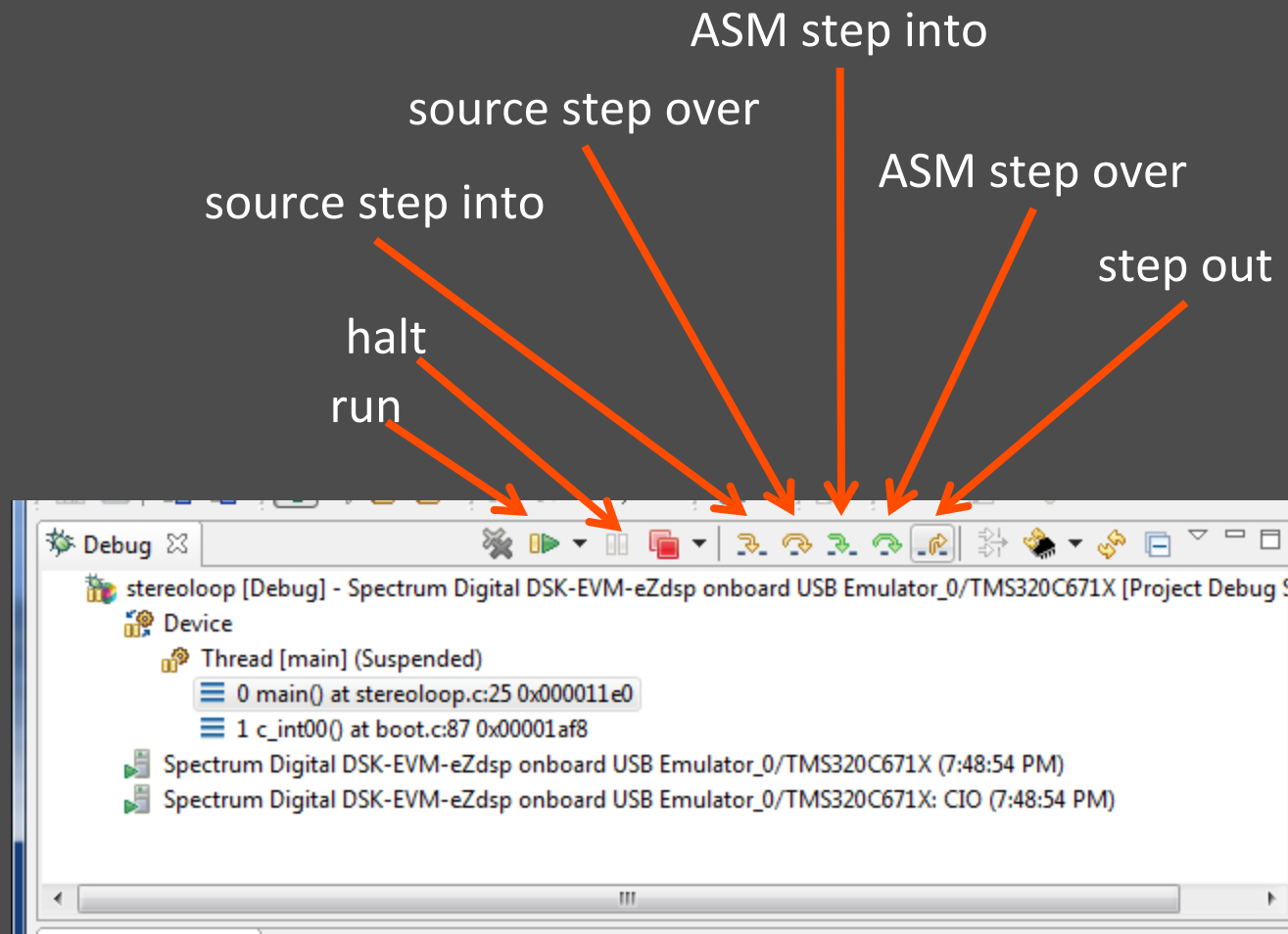
break point

```
51
52 interrupt void serialPortRcvISR()
53 {
54     union {Uint32 combo; short channel[2];} temp;
55
56     temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
57     // Note that right channel is in temp.channel[0]
58     // Note that left channel is in temp.channel[1]
59
60     MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);
61 }
```

- **Breakpoints:** stop code execution at this point to allow state examination and step-by-step execution.
- Also try View->Breakpoints

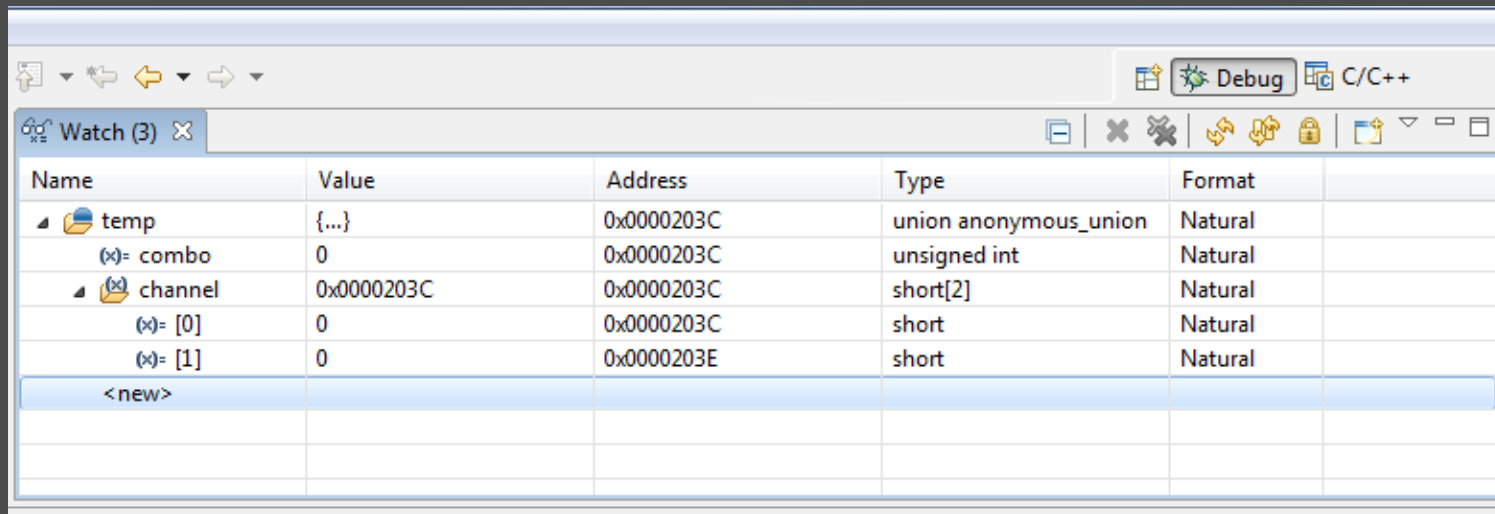


Using Breakpoints



Watch Variables

- View->Watch

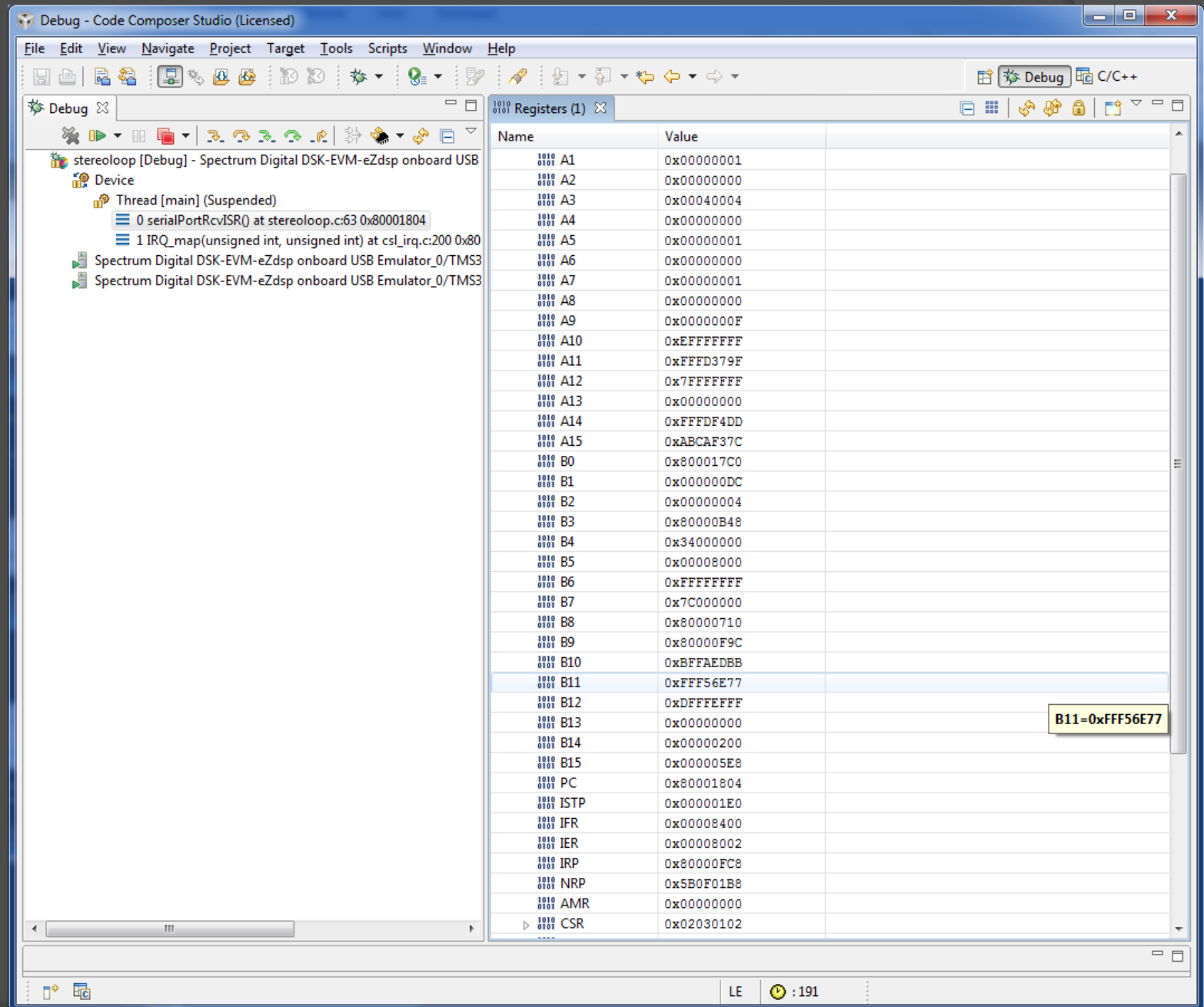


The screenshot shows the Visual Studio Watch window with the following data:

Name	Value	Address	Type	Format
temp	{...}	0x0000203C	union anonymous_union	Natural
(x)= combo	0	0x0000203C	unsigned int	Natural
(x) channel	0x0000203C	0x0000203C	short[2]	Natural
(x)= [0]	0	0x0000203C	short	Natural
(x)= [1]	0	0x0000203E	short	Natural
<new>				

- Type in any variable name currently in scope.
- Tip: Right click on <new> to easily add global variables.

Registers: View->Registers



Debug - Code Composer Studio (Licensed)

File Edit View Navigate Project Target Tools Scripts Window Help

Debug [StereoLoop] - Spectrum Digital DSK-EVM-eZdsp onboard USB

Device

Thread [main] (Suspended)

0 serialPortRcvISR() at stereoLoop.c:63 0x80001804

1 IRQ_map(unsigned int, unsigned int) at csL_irq.c:200 0x80

Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator_0/TMS3

Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator_0/TMS3

Registers (1)

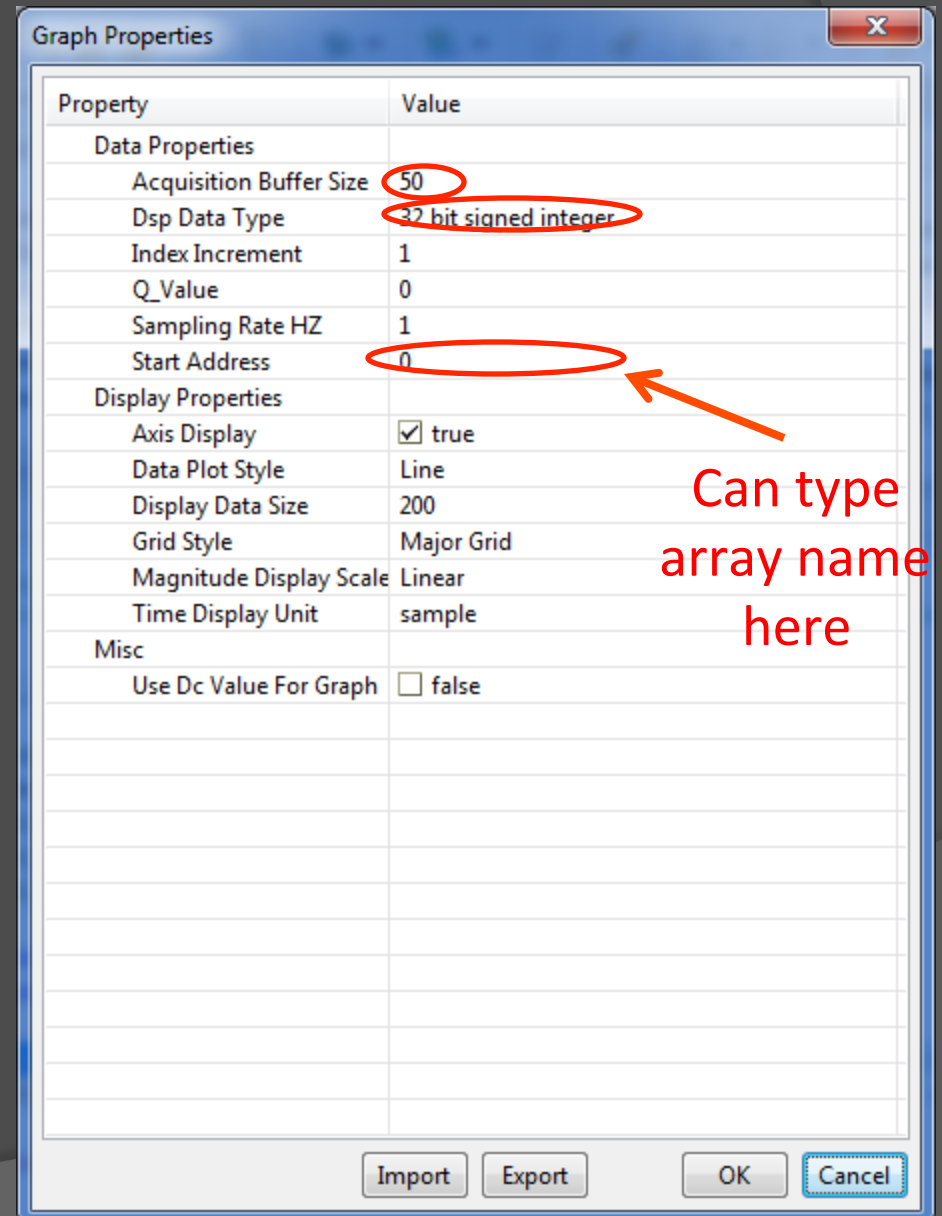
Name	Value
A1	0x00000001
A2	0x00000000
A3	0x00040004
A4	0x00000000
A5	0x00000001
A6	0x00000000
A7	0x00000001
A8	0x00000000
A9	0x0000000F
A10	0xEFFFFFFF
A11	0xFFFFD379F
A12	0x7FFFFFFF
A13	0x00000000
A14	0xFFFFDF4DD
A15	0xABCAF37C
B0	0x800017C0
B1	0x000000DC
B2	0x00000004
B3	0x80000B48
B4	0x34000000
B5	0x00008000
B6	0xFFFFFFFF
B7	0x7C000000
B8	0x80000710
B9	0x80000F9C
B10	0xBFFFAEDBB
B11	0xFFFF56E77
B12	0xDFFFFFFF
B13	0x00000000
B14	0x00000200
B15	0x000005E8
PC	0x80001804
ISTP	0x000001E0
IFR	0x00008400
IER	0x00008002
IRP	0x80000FC8
NRP	0x5B0F01B8
AMR	0x00000000
CSR	0x02030102

B11=0xFFFF56E77

LE : 191

Plotting Arrays of Data

- Tools -> Graph ->
(Typically Single Time)



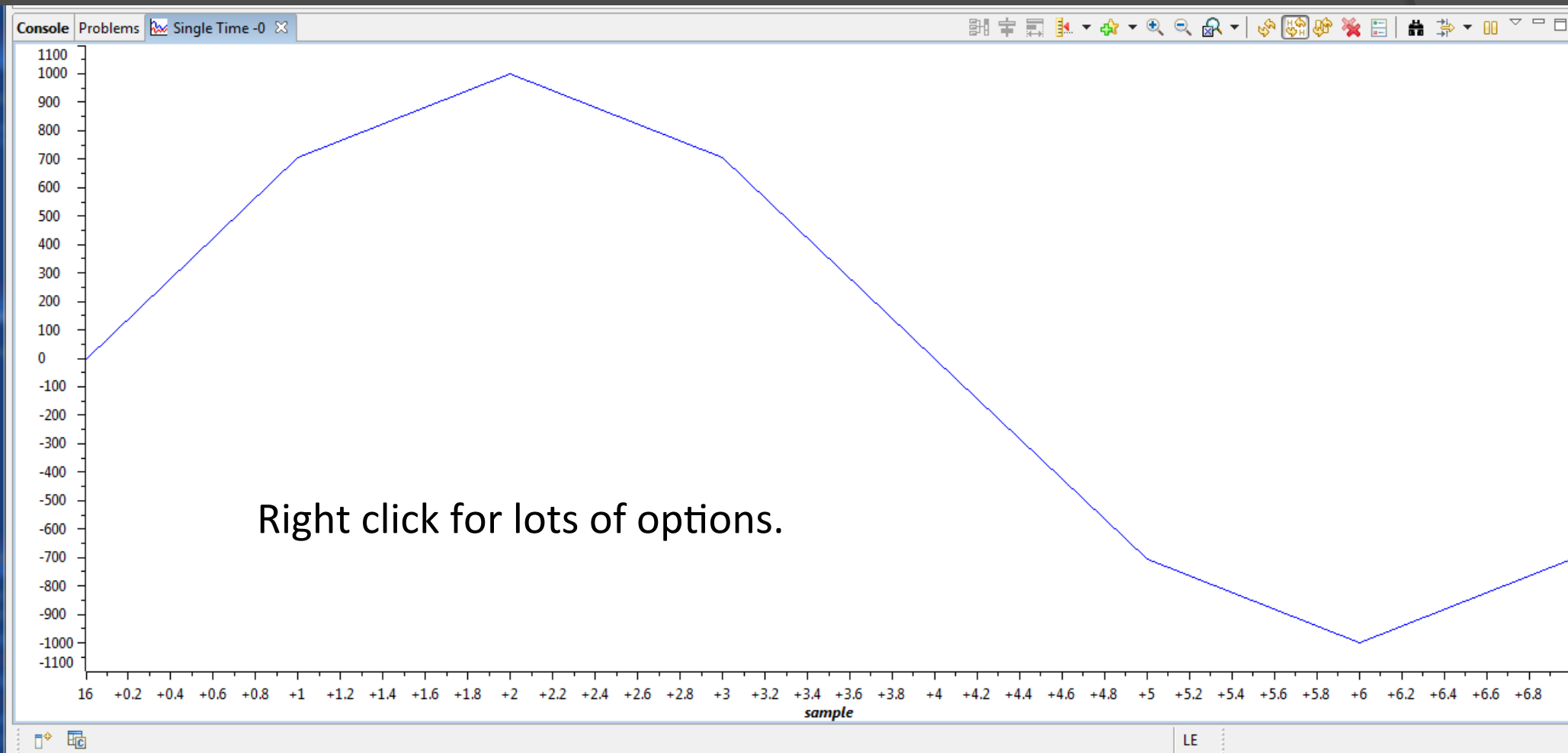
The image shows a 'Graph Properties' dialog box with a table of properties and values. Red circles highlight the values '50', '32 bit signed integer', and '0'. A red arrow points from the text 'Can type array name here' to the '0' in the 'Start Address' row.

Property	Value
Data Properties	
Acquisition Buffer Size	50
Dsp Data Type	32 bit signed integer
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	0
Display Properties	
Axis Display	☒ true
Data Plot Style	Line
Display Data Size	200
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	☐ false

Can type array name here

Buttons: Import, Export, OK, Cancel

Graph Windows: Plotting Arrays of Data



Profiling Your Code and Making it More Efficient

- How to estimate the **execution time** of your code.
- How to use the **optimizing compiler** to produce more efficient code.
- Other factors affecting the efficiency of your code.

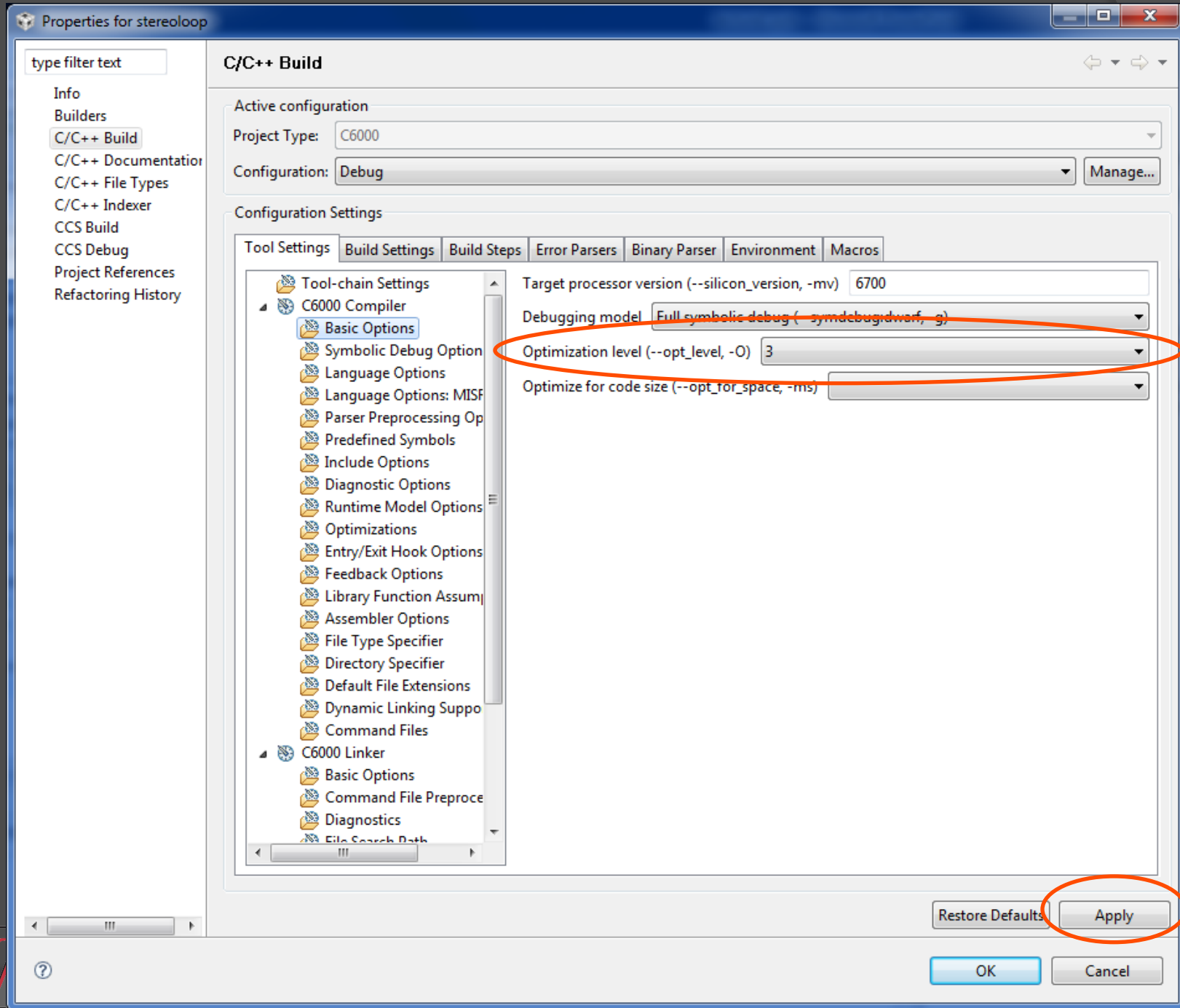
How to estimate code execution time when connected to the DSK

1. Open the source file you wish to profile
2. Set two breakpoints for the start/end of the code range you wish to profile

```
53  
54 interrupt void serialPortRcvISR()  
55 {  
56     union {Uint32 combo; short channel[2];} temp;  
57  
58     temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);  
59     // Note that right channel is in temp.channel[0]  
60     // Note that left channel is in temp.channel[1]  
61  
62     MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);  
63 }  
64  
65
```

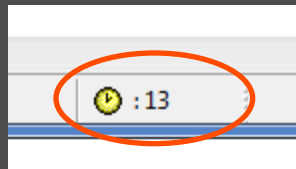
3. Build it and load .out file to the DSK
4. Target -> Clock -> Enable
5. Target -> Clock -> View
6. Run to the first breakpoint
7. Target -> Clock -> Reset (or double click the clock to reset the clock to zero)
8. Run to the second breakpoint
9. Clock will show raw number of execution cycles between breakpoints.

Optimizing Compiler



Profiling results after compiler optimization

- Rebuild and reload the program to the DSK
- Use your breakpoint/clock method to profile the execution time
- In this example, we get a 5x-6x improvement with Level-3 Optimization
- Optimization gains can be much larger, e.g. 20x



Limitations of hardware profiling

- Variability of results
- Profiling is known to be somewhat inaccurate when connected to real hardware
- Breakpoint/clock profiling method may not always work with compiler-optimized code
- For the best results, TI recommends profiling your code in a **cycle accurate simulator**:
 - New target configuration:
 - Connection = Texas Instruments Simulator
 - Device = C6713 Device Cycle Accurate Simulator, Little Endian
 - Need to create a new project for the simulator and copy your functions/code for profiling to this project without calls to board-specific functions
 - Tools -> Profile -> Setup and then Tools-> Profile -> View

Debug - simulortester.c - Code Composer Studio (Licensed)

File Edit View Navigate Project Target Tools Scripts Window Help

Debug Profile Setup

Simulator [Debug] - C6713 Device Cycle Accurate Simulator, Little Endian_0/TMS320C6713 [Project Debug Session]

Device

- Thread [main] (Suspended)
 - 0 myfunc() at simulortester.c:53 0x00000a24
 - 1 main() at simulortester.c:38 0x0000096c
 - 2 C:\int000 at boot.c:87 0x00001238
- C6713 Device Cycle Accurate Simulator, Little Endian_0/TMS320C6713 (8:46:06 AM)
- C6713 Device Cycle Accurate Simulator, Little Endian_0/TMS320C6713: CIO (8:46:06 AM)

Not running on DSK

simulortester.c

```
26 void myfunc(void);
27
28 void main()
29 {
30
31     // Configure buffered serial ports for 32 bit operation
32     // This allows transfer of both right and left channels in one read/write
33     MCBSP_FSETS(SPCR1, RINTM, FRM);
34     MCBSP_FSETS(SPCR1, XINTM, FRM);
35     MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
36     MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
37
38     while(1)                // main loop - do nothing but wait for interrupts
39     {
40         myfunc();
41     }
42 }
43
44 void myfunc()
45 {
46     union {Uint32 combo; short channel[2];} temp;
47     short i;
48
49     temp.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
50     // Note that right channel is in temp.channel[0]
51     // Note that left channel is in temp.channel[1]
52
53     for (i=0;i<100;i++)
54         z += x[i]*y[i];
55
56     MCBSP_write(DSK6713_AIC23_DATAHANDLE, temp.combo);
57 }
58
59
```

23 float x[100] = {0};
24 float y[100] = {0};
25 float z = 0;

All calls to BSL functions removed

Code from ISR placed in a regular function called from main()

LE Writable Smart Insert 53:1

Tools -> Profile -> Setup, then Tools-> Profile -> View

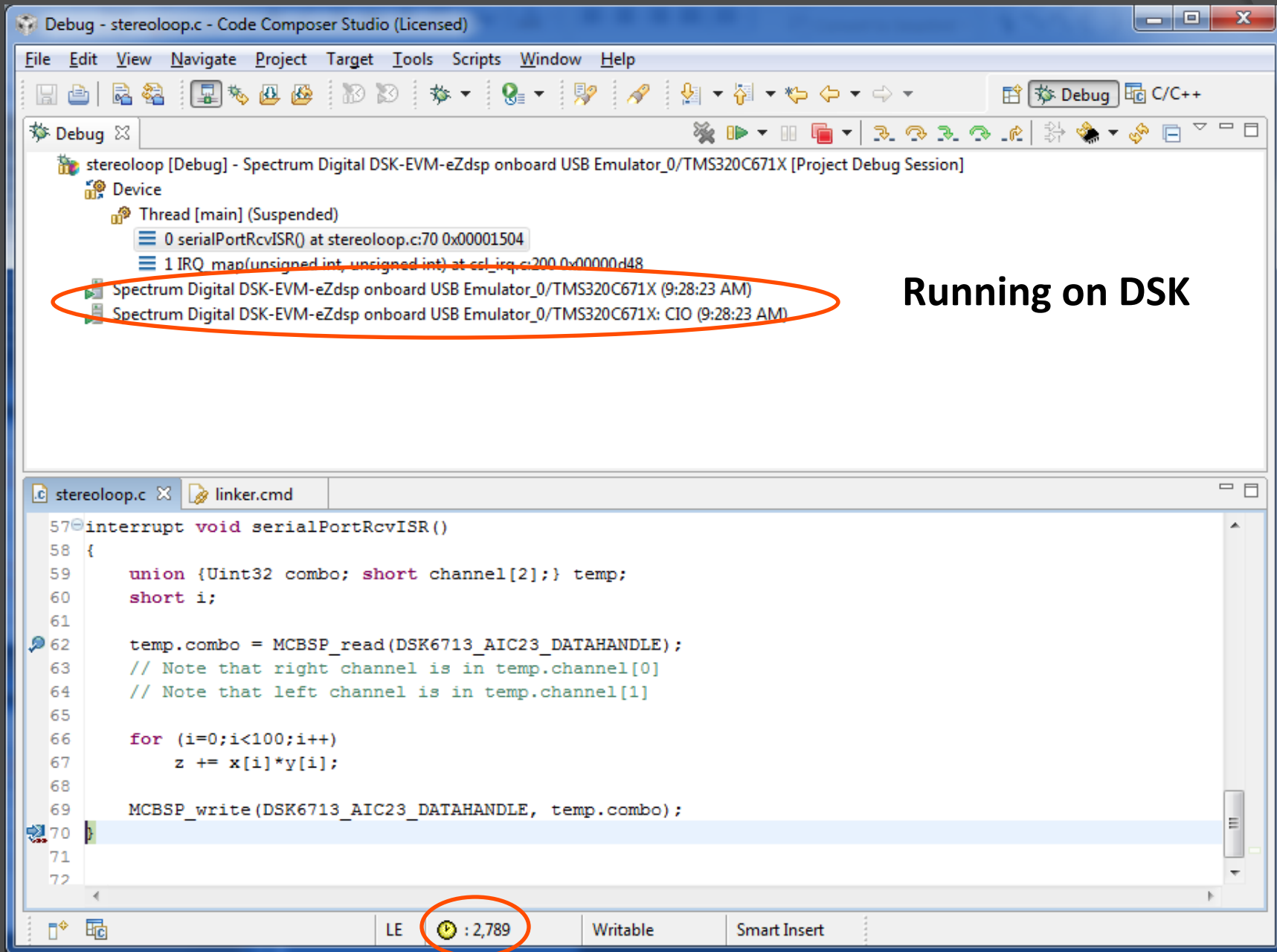
The screenshot shows the Code Composer Studio (CCS) interface. The top menu bar includes File, Edit, View, Navigate, Project, Target, Tools, Scripts, Window, and Help. The 'Tools' menu is open, showing 'Profile' and 'Setup'. The 'Profile Setup' dialog is displayed, showing the configuration for 'Configuration 1'. The 'Active On' field is set to 'TMS320C6713'. Under the 'Activities' section, the checkbox for 'Profile all Functions for Total Cycles' is checked and circled in red. Below this, there are buttons for 'Properties...' and 'Collect Code Coverage and Exclusive Profile Data'. At the bottom of the dialog are 'Activate', 'Save', and 'Revert' buttons.

The main editor window shows the source code for 'simulatortester.c'. The code includes a function 'myfunc()' which contains a union, a short variable 'i', and a loop. The line 'for (i=0;i<100;i++)' is highlighted in green.

The bottom panel shows the 'Profile' view, which displays a table of profiling data. The table has columns for Name, Calls, Excl Count Min, Excl Count Max, Excl Count Average, Excl Count Total, Incl Count Min, Incl Count Max, Incl Count Average, Incl Count Total, Filename, Line Number, and Start Address. The 'myfunc()' row is highlighted, and the 'Incl Count Max' value of 4097 is circled in red.

Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filename	Line Number	Start Address
main()	1	-	-	34680.00	34680	-	-	11476328.00	11476328	C:\User...	38	2272
myfunc()	2891	3943	4064	3941.70	11395454	3959	4097	3957.70	11441721	C:\User...	45	2420

Compare to Breakpoint/Clock Method



Simulator includes overhead of function call/return and allocating/deallocating local variables.

Other factors affecting code efficiency

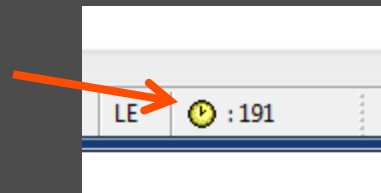
Memory

- 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

Example: Stereoloop project,
changing .text and .data
to external SDRAM:



About 2.5x slower
than SRAM (can be worse)

TMS320C6000 C/C++ Data Types

Type	Size	Representation	Range	
			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 1
 - Lab/lecture slides
 - Reference material noted in slides
 - Textbooks listed in syllabus
 - **Please make sure you understand what you are doing.** Please ask questions if you are unsure.
- ◎ Lab 1 Part 3: Signal Squaring
 - Simple example of (memoryless) non-linear signal processing
 - Sometimes used in synchronization algorithms
 - You want the analog input signal to use the full range of the ADC but avoid clipping (clipping = very bad nonlinear distortion)
 - You also want to avoid clipping in the output
 - Careful analysis of the output will reveal certain “features” of the AIC 23