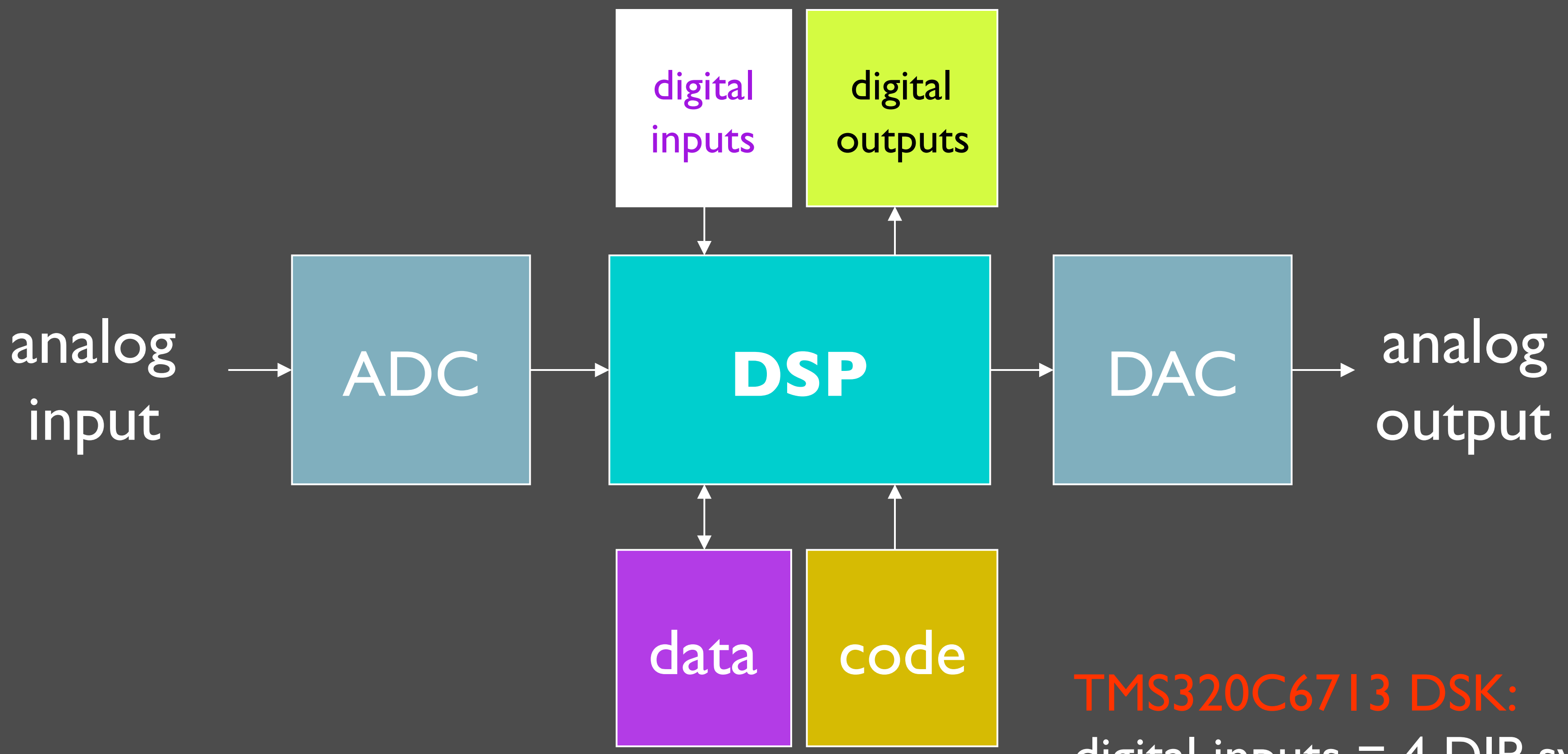
D. Richard Brown III
Associate Professor
Worcester Polytechnic Institute
Electrical and Computer Engineering Department
drb@ece.wpi.edu

28-October-2010

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 c6713dsk.hlp (on course website).

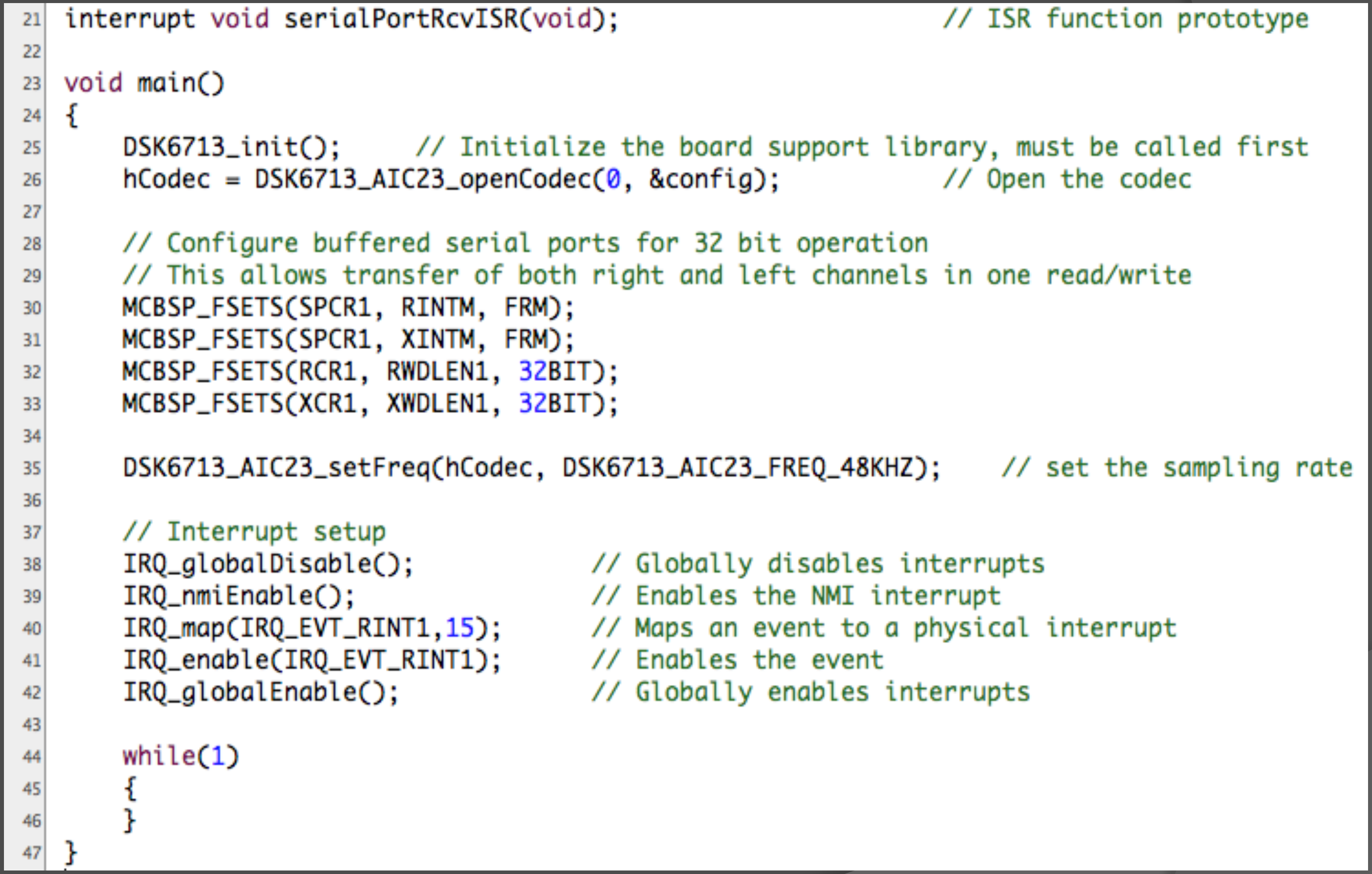
AIC23 Codec

- 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)

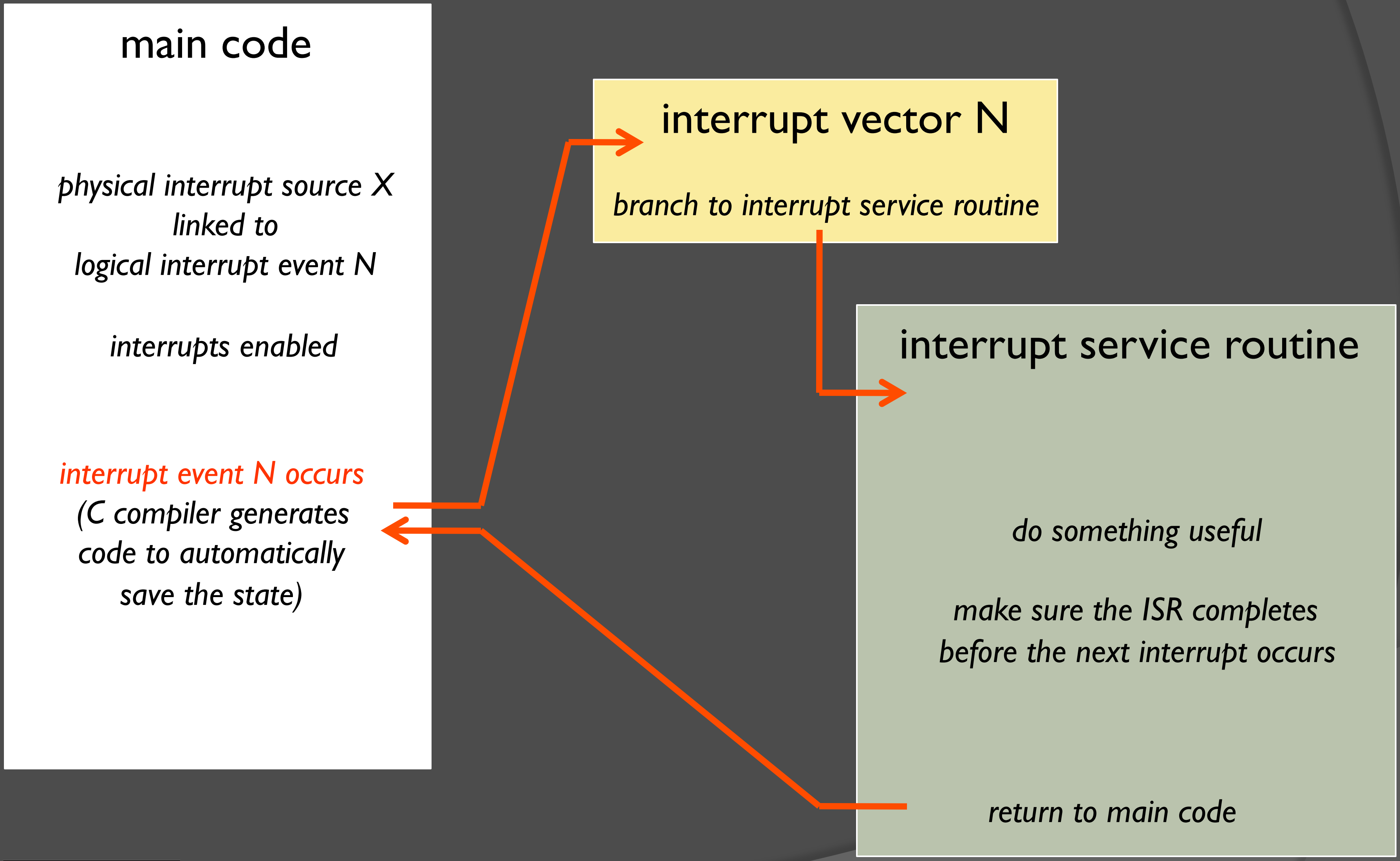


```
interrupt void serialPortRcvISR(void);                      // ISR function prototype

void main()
{
    DSK6713_init();     // Initialize the board support library, must be called first
    hCodec = DSK6713_AIC23_openCodec(0, &config);           // Open the codec

    // Configure buffered serial ports for 32 bit operation
    // This allows transfer of both right and left channels in one read/write
    MCBSP_FSETS(SPCR1, RINTM, FRM);
    MCBSP_FSETS(SPCR1, XINTM, FRM);
    MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
    MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);

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

    // Interrupt setup
    IRQ_globalDisable();            // Globally disables interrupts
    IRQ_nmiEnable();                // Enables the NMI interrupt
    IRQ_map(IRQ_EVT_RINT1,15);      // Maps an event to a physical interrupt
    IRQ_enable(IRQ_EVT_RINT1);      // Enables the event
    IRQ_globalEnable();             // Globally enables interrupts

    while(1)
    {
    }
}
```

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



main code

physical interrupt source X
linked to
logical interrupt event N

interrupts enabled

interrupt event N occurs
(C compiler generates code to automatically save the state)

interrupt vector N

branch to interrupt service routine

interrupt service routine

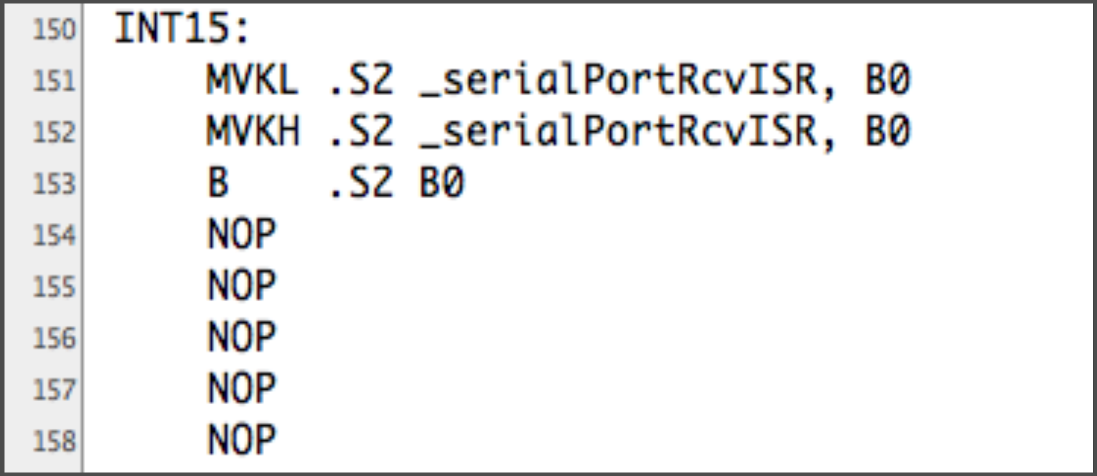
do something useful

make sure the ISR completes before the next interrupt occurs

return to main code

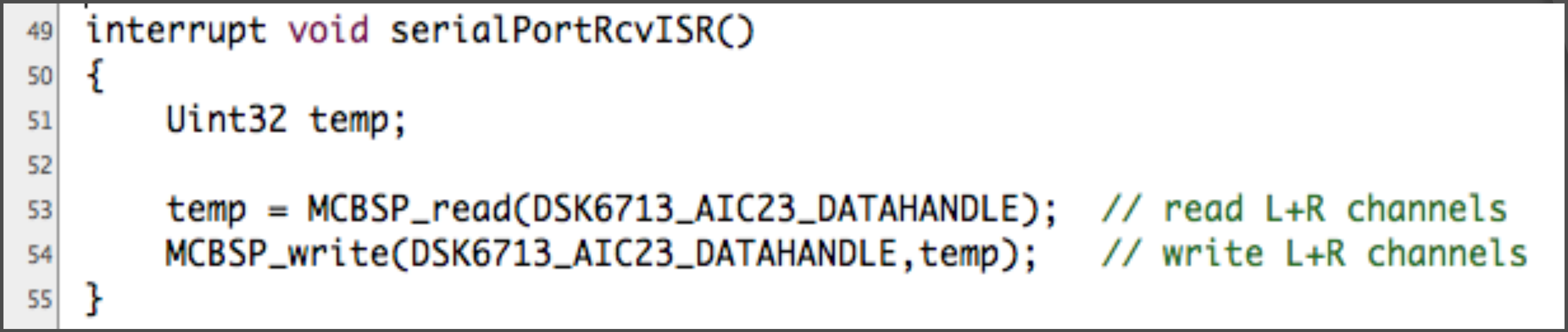
Interrupt Vector

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



```
INT15:
    MVKL .S2 _serialPortRcvISR, B0
    MVKH .S2 _serialPortRcvISR, B0
    B    .S2 B0
    NOP
    NOP
    NOP
    NOP
    NOP
```

A Simple Interrupt Service Routine



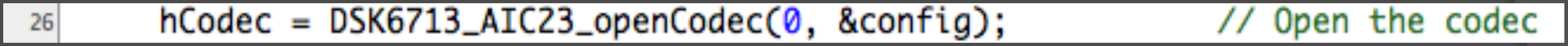
```
interrupt void serialPortRcvISR()
{
    Uint32 temp;

    temp = MCBSP_read(DSK6713_AIC23_DATAHANDLE);  // read L+R channels
    MCBSP_write(DSK6713_AIC23_DATAHANDLE,temp);   // write L+R channels
}
```

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

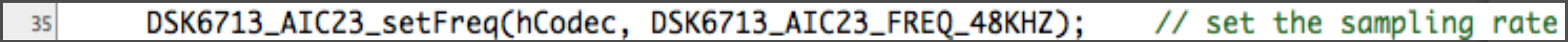
Here we open the codec with the default configuration:



```
    hCodec = DSK6713_AIC23_openCodec(0, &config);          // Open the codec
```

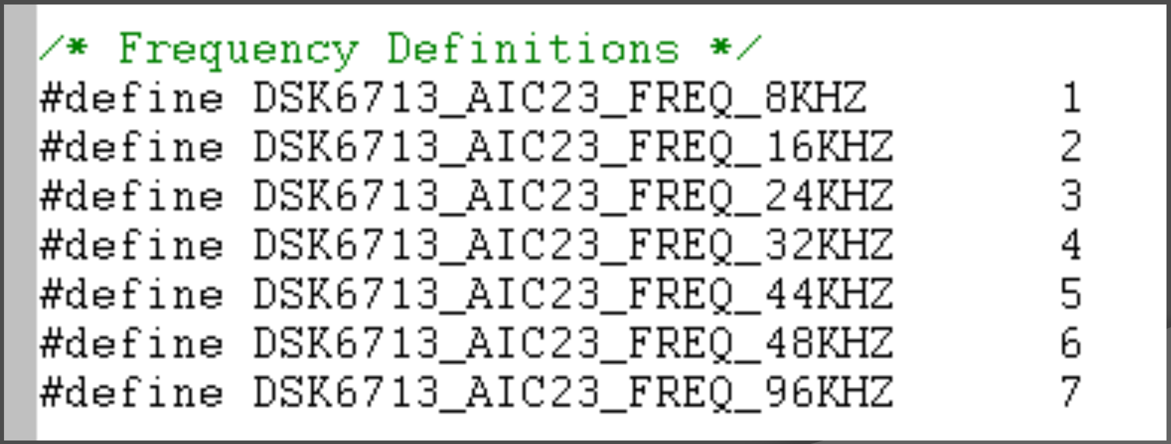
The structure "config" is declared in dsk6713_aic23.h

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



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

Frequency definitions are in dsk6713_aic.h



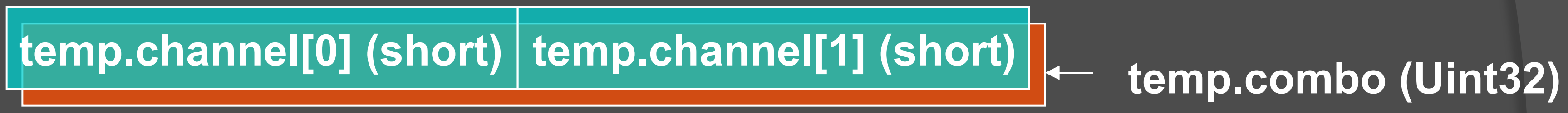
```
/* Frequency Definitions */
#define DSK6713_AIC23_FREQ_8KHZ        1
#define DSK6713_AIC23_FREQ_16KHZ       2
#define DSK6713_AIC23_FREQ_24KHZ       3
#define DSK6713_AIC23_FREQ_32KHZ       4
#define DSK6713_AIC23_FREQ_44KHZ       5
#define DSK6713_AIC23_FREQ_48KHZ       6
#define DSK6713_AIC23_FREQ_96KHZ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

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];
```

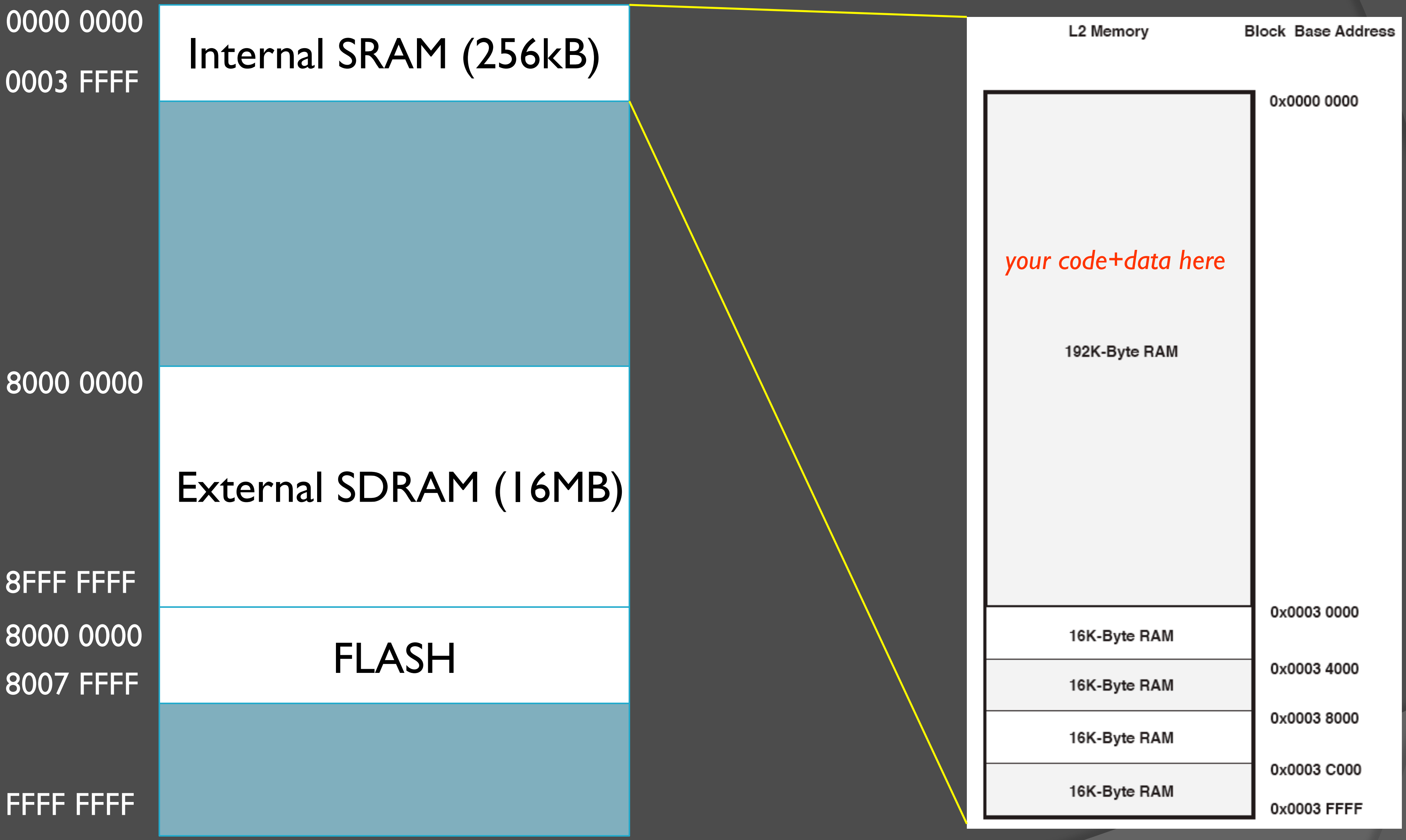
Final Remarks on DSP/Codec Interface

- In 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.
- 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
- 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



0000 0000

0003 FFFF

Internal SRAM (256kB)

8000 0000

8FFF FFFF

External SDRAM (16MB)

8000 0000

8007 FFFF

FLASH

FFFF FFFF

L2 Memory	Block Base Address
	0x0000 0000
your code+data here 192K-Byte RAM	
	0x0003 0000
16K-Byte RAM	
	0x0003 4000
16K-Byte RAM	
	0x0003 8000
16K-Byte RAM	
	0x0003 C000
16K-Byte RAM	
	0x0003 FFFF

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

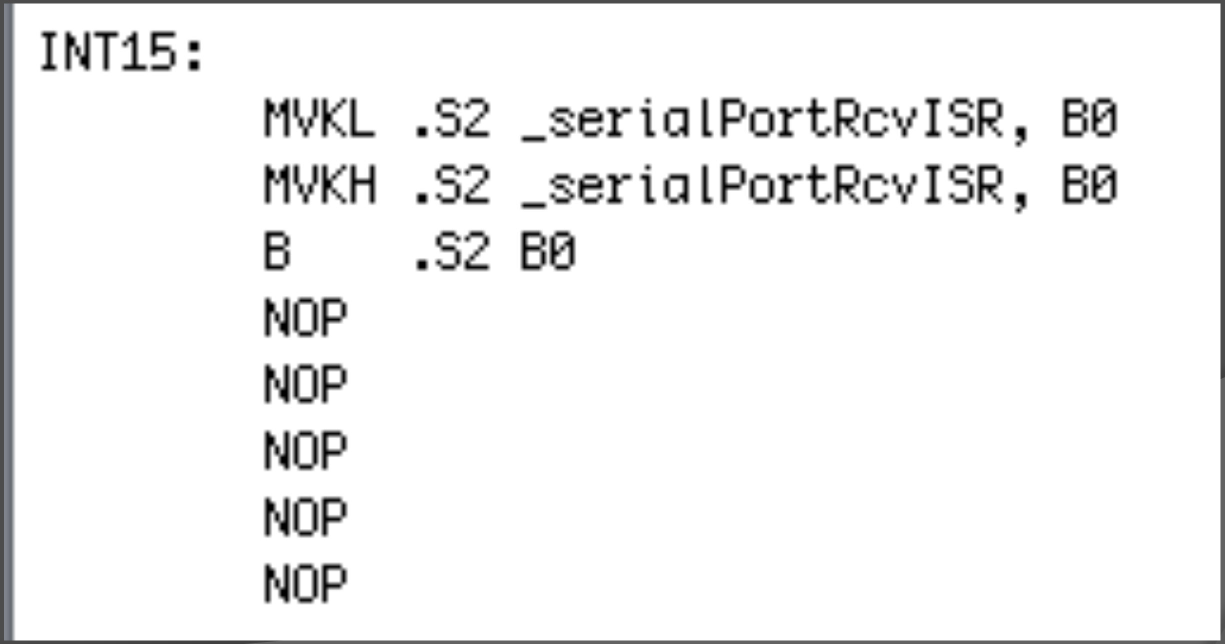
Addresses 00000000-0002FFFF correspond to the lowest 192kB of internal memory (SRAM) and are labeled "IRAM".

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:



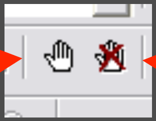
```
INT15:
        MVKL .S2 _serialPortRcvISR, B0
        MVKH .S2 _serialPortRcvISR, B0
        B    .S2 B0
        NOP
        NOP
        NOP
        NOP
        NOP
```

Debugging and Other Useful Features of the CCS IDE

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

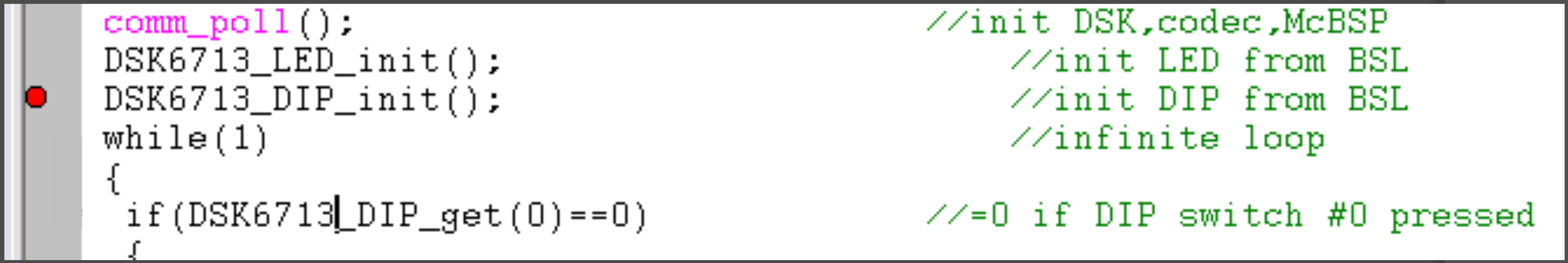
Breakpoints

toggle
break point



clear all
break points

break point



```
comm_poll();                                 //init DSK,codec,McBSP
DSK6713_LED_init();                             //init LED from BSL
DSK6713_DIP_init();                             //init DIP from BSL
while(1)                                        //infinite loop
{
 if(DSK6713_DIP_get(0)==0)                   //=0 if DIP switch #0 pressed
 {
```

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

Breakpoints

source step into

source step over

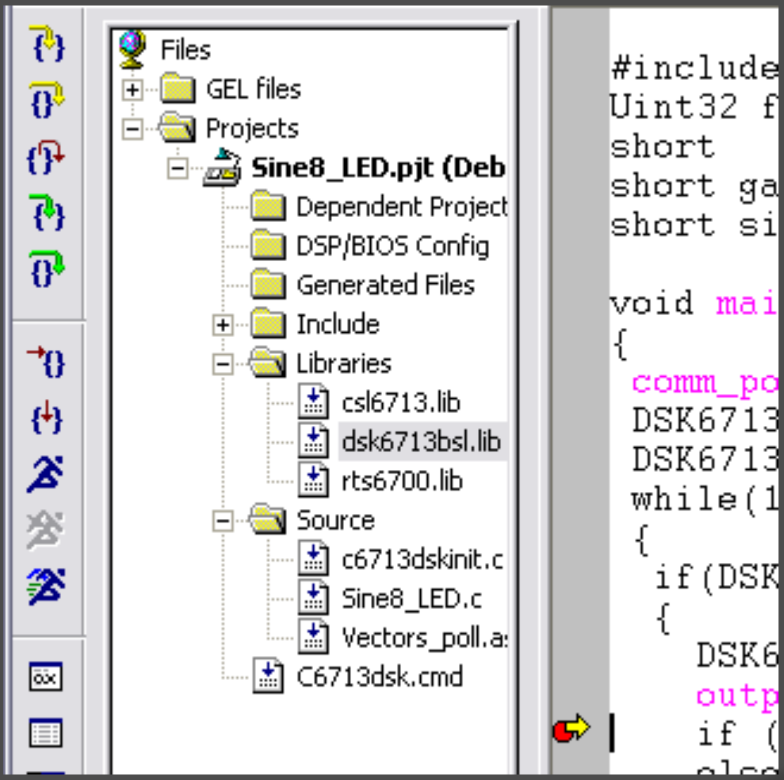
step out

ASM step into

ASM step over

run to cursor

set progam counter to cursor

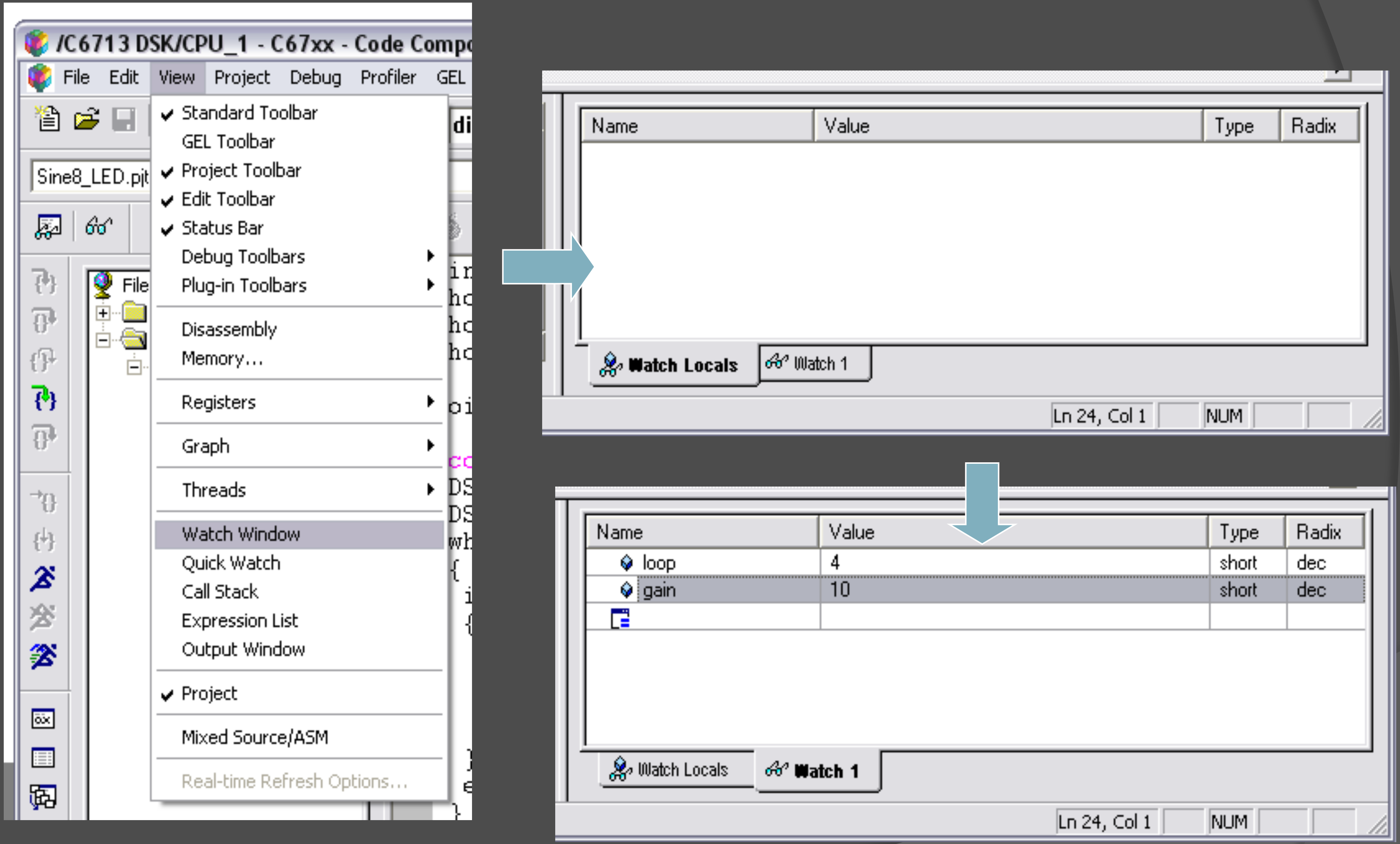


- Files
  - GEL files
  - Projects
    - Sine8_LED.pjt (Deb
      - Dependent Project
      - DSP/BIOS Config
      - Generated Files
      - Include
      - Libraries
        - csl6713.lib
        - dsk6713bsl.lib
        - rts6700.lib
      - Source
        - c6713dskinit.c
        - Sine8_LED.c
        - Vectors_poll.a:
      - C6713dsk.cmd

```
#include
Uint32 f
short
short ga
short si

void mai
{
 comm_po
 DSK6713
 DSK6713
 while(1
 {
  if(DSK
  {
   DSK6
   outp
   if (
```

"Run to Cursor" is a handy shortcut instead of setting a breakpoint

Watch Variables

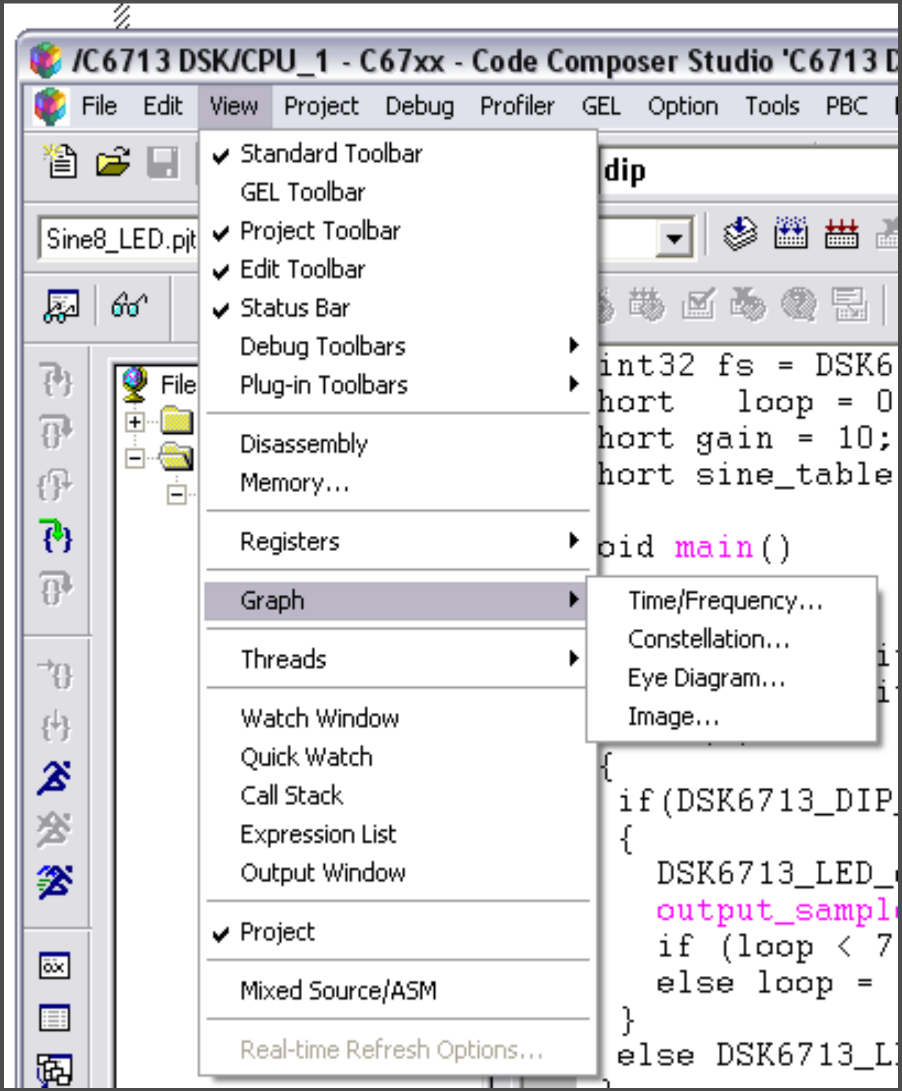


Name	Value	Type	Radix
loop	4	short	dec
gain	10	short	dec

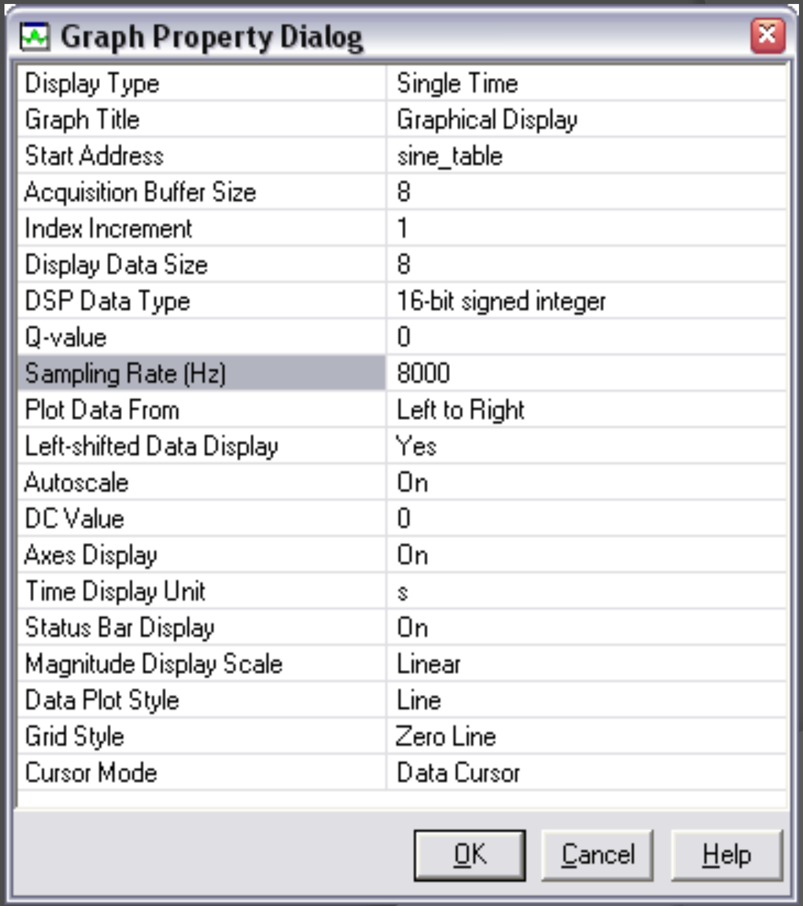
Watch Variables

- 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

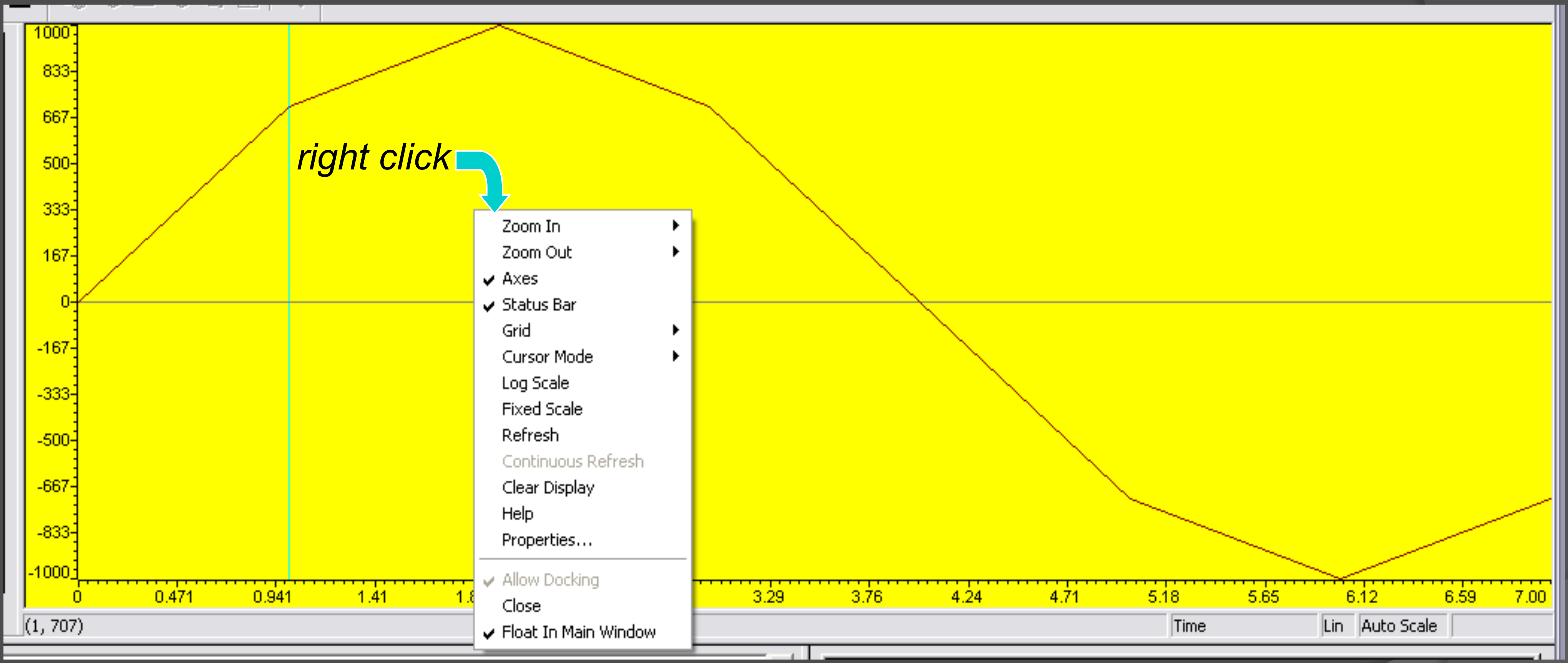


```
int32 fs = DSK6
hort   loop = 0
hort gain = 10;
hort sine_table

oid main()
                    i
                    i
{
 if(DSK6713_DIP
 {
   DSK6713_LED_
   output_sampl
   if (loop < 7
   else loop =
 }
 else DSK6713_L
```



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	s
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Line
Grid Style	Zero Line
Cursor Mode	Data Cursor

Graph Windows: Plotting Arrays of Data

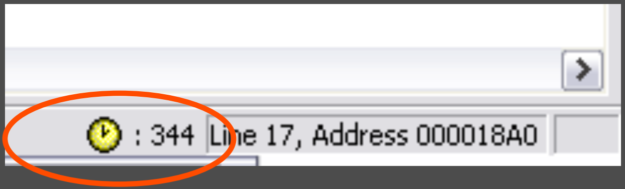


Profiling Your Code and Making it More Efficient

- 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. 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
11. Clock will show raw number of execution cycles between breakpoints.



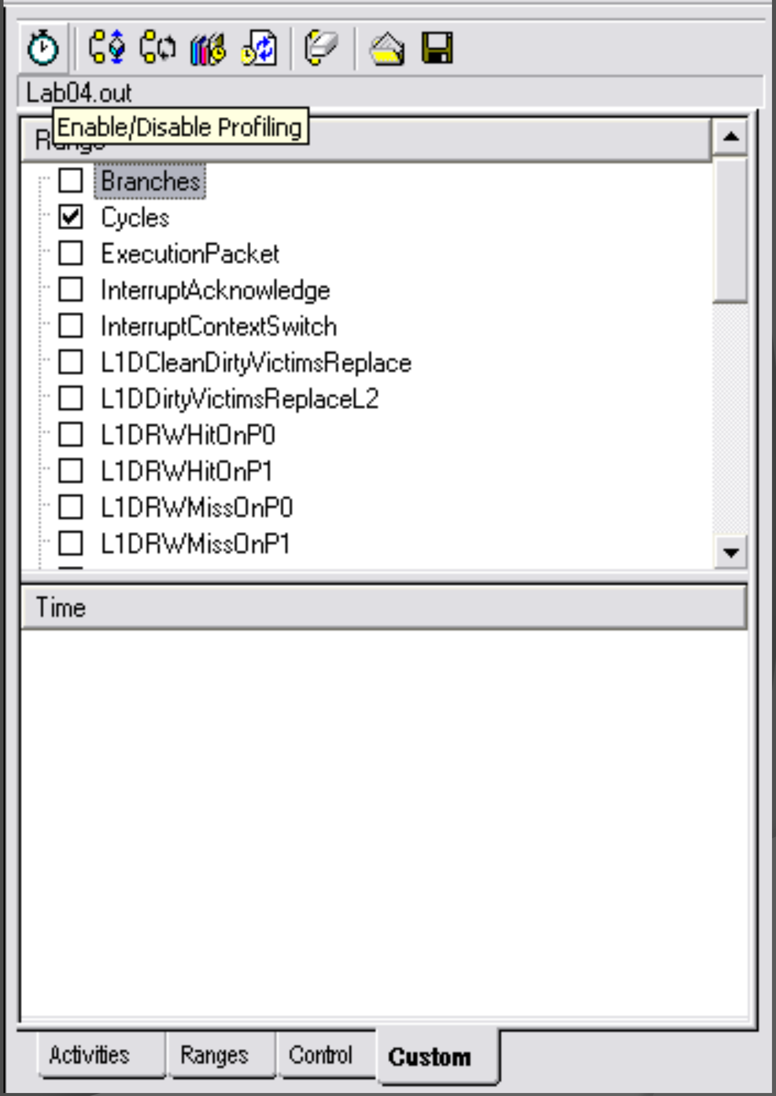
: 344 Line 17, Address 000018A0

Tip: You can save your breakpoints, graphs, and watch windows with File -> Workspace -> Save Workspace As

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

Repeat steps 1-4 previous method.

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



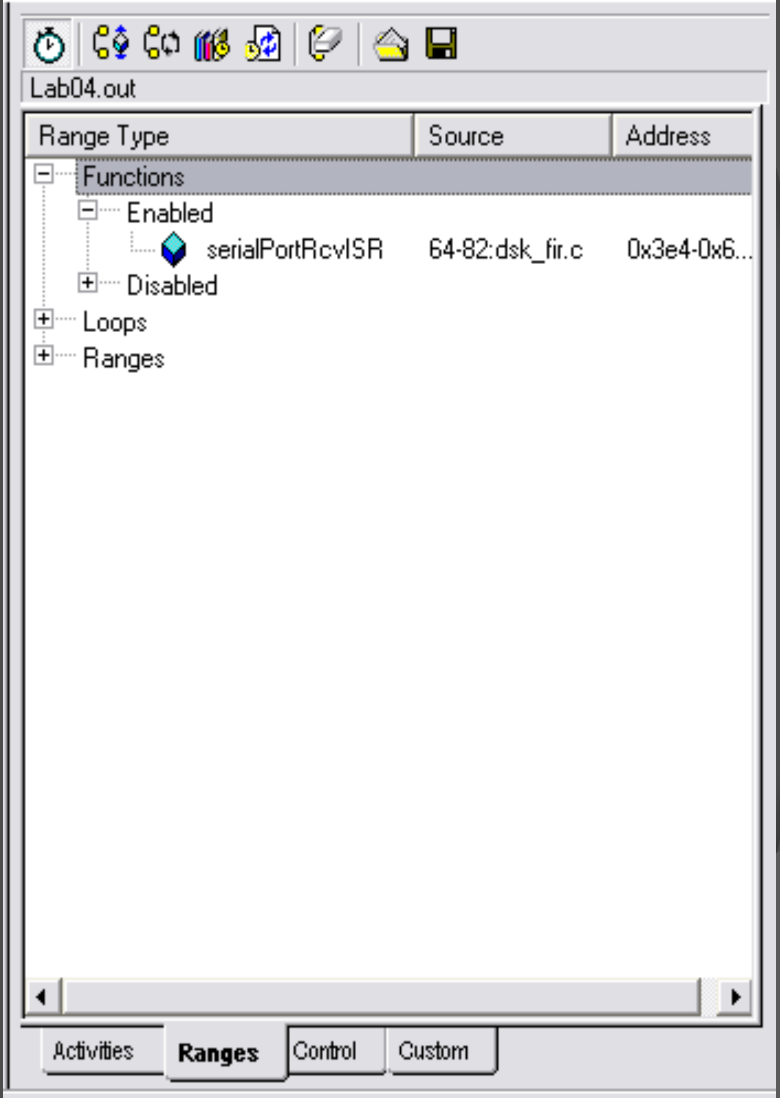
Lab04.out

Enable/Disable Profiling

- □ Branches
- ☑ Cycles
- □ ExecutionPacket
- □ InterruptAcknowledge
- □ InterruptContextSwitch
- □ L1DCleanDirtyVictimsReplace
- □ L1DDirtyVictimsReplaceL2
- □ L1DRWHitOnP0
- □ L1DRWHitOnP1
- □ L1DRWMissOnP0
- □ L1DRWMissOnP1

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

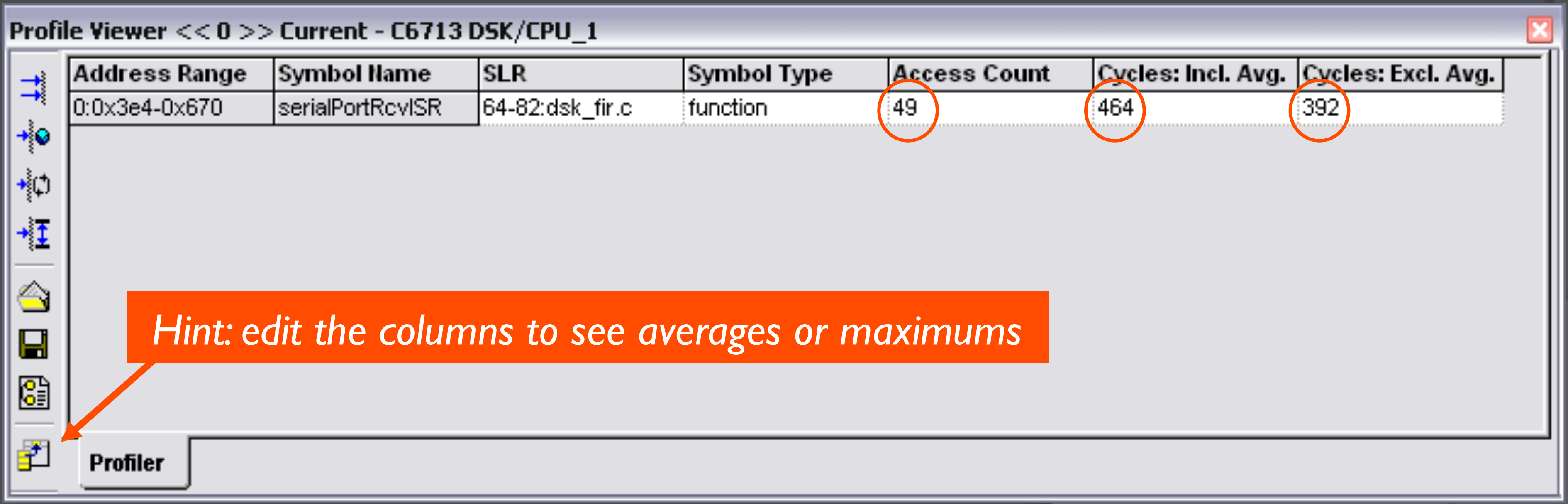
10. Select Ranges tab
11. Highlight code you want to profile and drag into ranges window (hint: you can drag whole functions into this window)
12. Repeat for other ranges if desired



Range Type	Source	Address
⊟ Functions		
⊟ Enabled		
serialPortRcvISR	64-82:dsk_fir.c	0x3e4-0x6...
⊞ Disabled		
⊞ Loops		
⊞ Ranges		

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



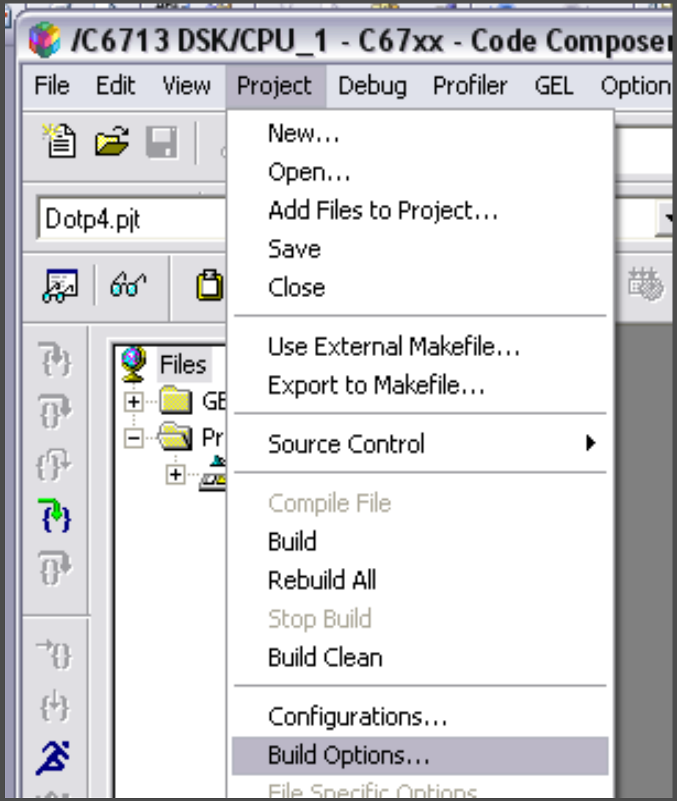
Address Range	Symbol Name	SLR	Symbol Type	Access Count	Cycles: Incl. Avg.	Cycles: Excl. Avg.
0:0x3e4-0x670	serialPortRcvISR	64-82:dsk_fir.c	function	49	464	392

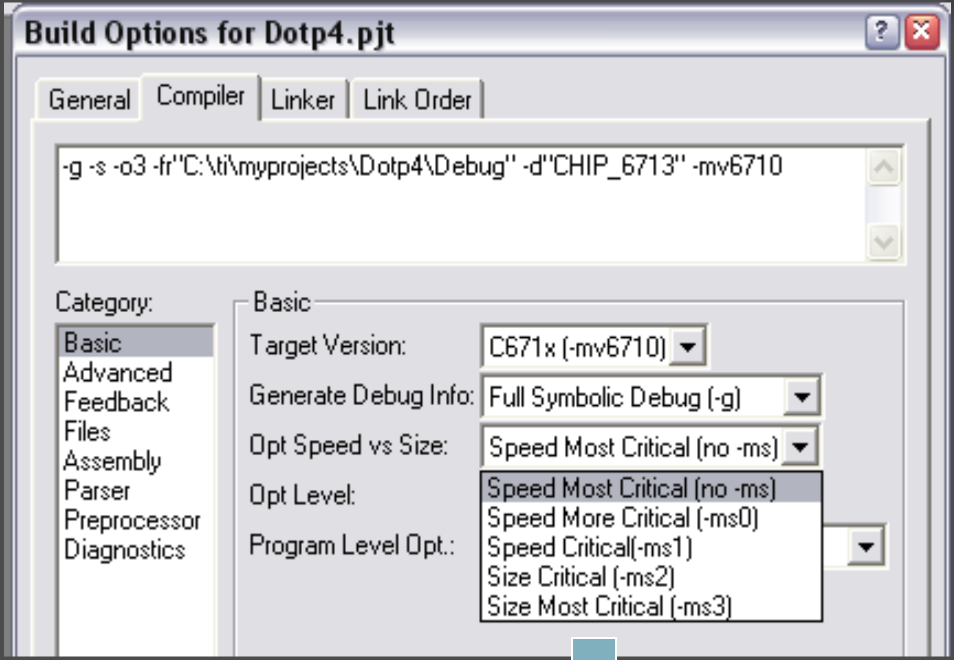
Hint: edit the columns to see averages or maximums

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





```
-g -s -o3 -fr"C:\ti\myprojects\Dotp4\Debug" -d"CHIP_6713" -mv6710
```

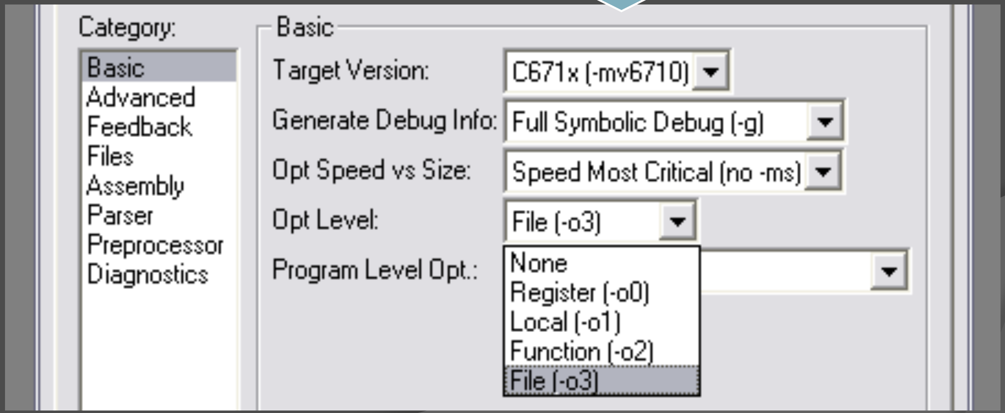
Target Version: C671x (-mv6710)

Generate Debug Info: Full Symbolic Debug (-g)

Opt Speed vs Size: Speed Most Critical (no -ms)

Opt Level:

Program Level Opt.:



Target Version: C671x (-mv6710)

Generate Debug Info: Full Symbolic Debug (-g)

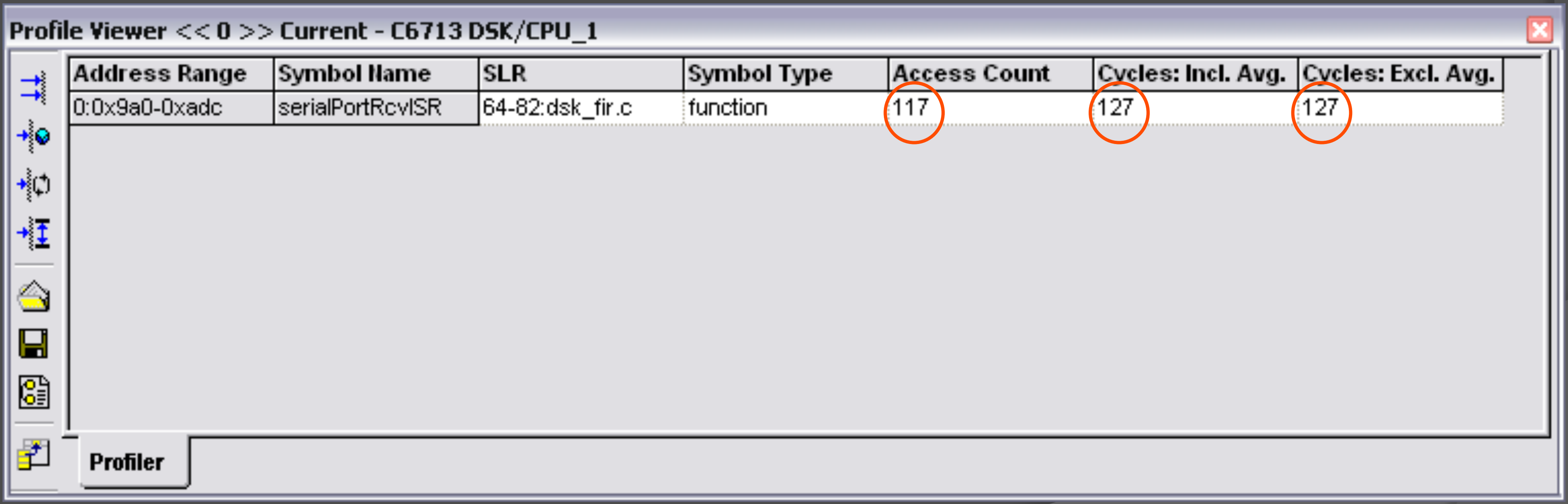
Opt Speed vs Size: Speed Most Critical (no -ms)

Opt Level: File (-o3)

Program Level Opt.:

Profiling results after compiler optimization

- In this example, we get a 3x-4x improvement with "Speed Most Critical" and "File (-o3)" optimization
- Optimization gains can be much larger, e.g. 20x



Profile Viewer << 0 >> Current - C6713 DSK/CPU_1

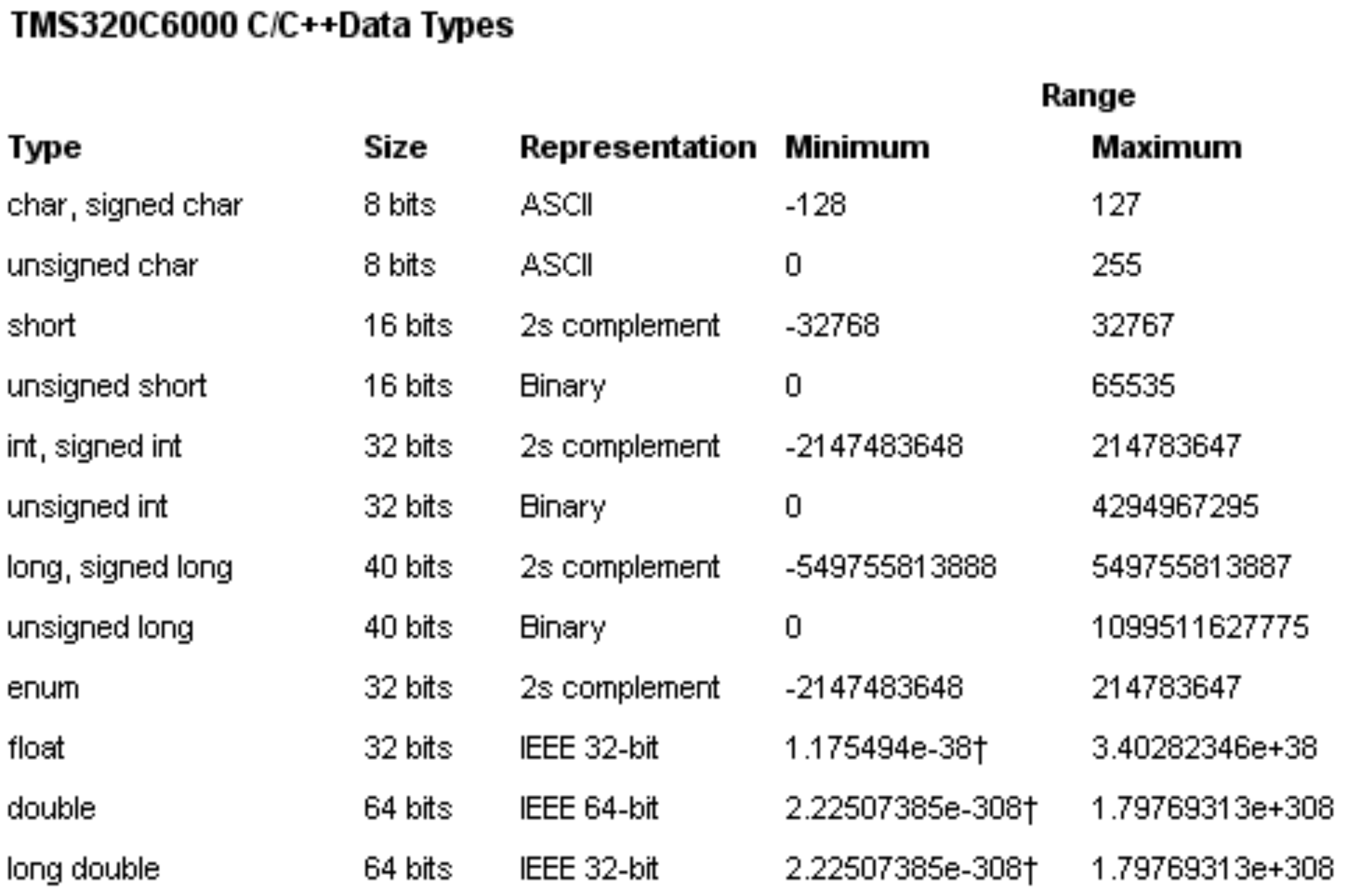
Address Range	Symbol Name	SLR	Symbol Type	Access Count	Cycles: Incl. Avg.	Cycles: Excl. Avg.
0:0x9a0-0xadc	serialPortRcvISR	64-82:dsk_fir.c	function	117	127	127

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 "cycle accurate 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	
Type	**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 1
  - 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. Please ask questions if you are unsure.
- Lab 1 Part 3: Signal Squaring
  - Simple example of 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