Digital Signal Processors: fundamentals & system design

Lecture 2

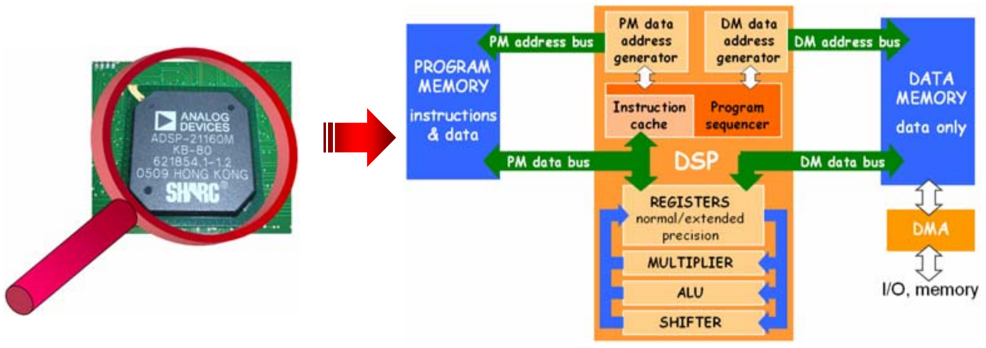
Maria Elena Angoletta
CERN

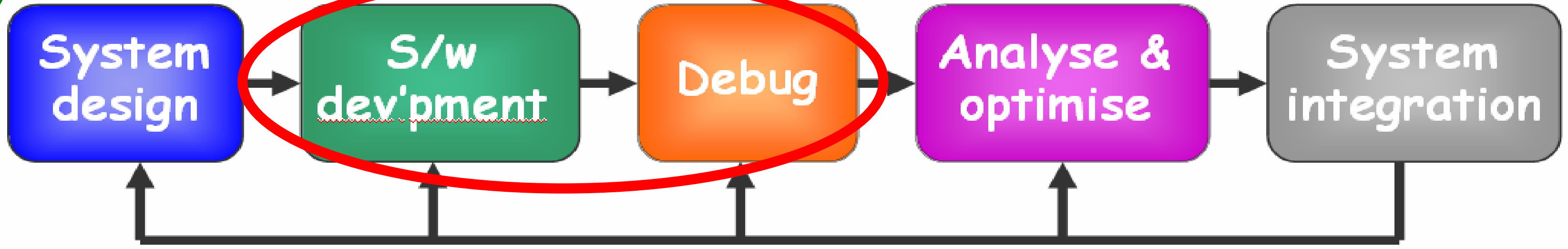
Topical CAS/Digital Signal Processing
Sigtuna, June 1-9, 2007

Lectures plan

Lecture 1

introduction, evolution, DSP core + peripherals

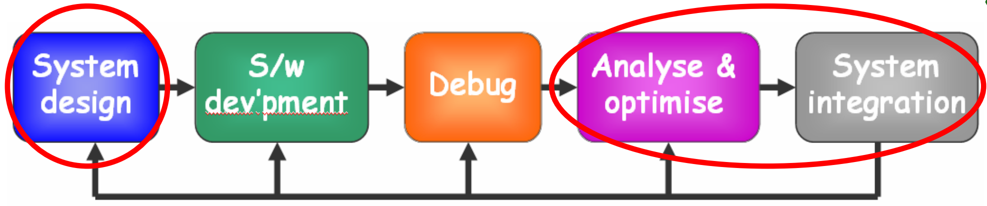




Lecture 2 DSP peripherals (cont'd), s/w dev'pment & debug.

Lecture 3

System optimisation, design & integration.



Lecture 2 - outline

Chapter 4 DSP peripherals (cont'd)

Chapter 5 RT design flow: introduction

Chapter 6 RT design flow: s/w development

Chapter 7 RT design flow: debugging

Chapter 4 topics

DSP peripherals

4.1 Introduction
4.2 Interconnect & I/O
4.3 Services
4.4 C6713 example

Yesterday

4.5 Memory interfacing
4.6 Data converter interfacing
4.7 DSP booting
Summary

Now

4.5 Memory interfacing

- **H/w interface often available in TI & ADI DSPs.**

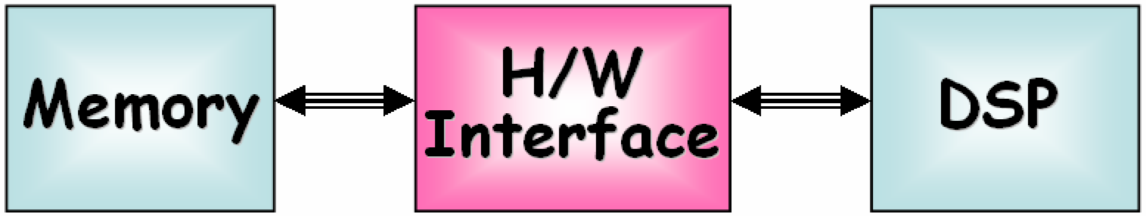
  Ex: TI External Memory InterFace (EMIF).

  - Glueless interface to SRAM, EPROM, Flash, SBSRAM, SDRAM.

    C6713: 32-bit EMIF, 512 MByte addressable ext. memory space.

- **No dedicated h/w interface → external h/w (ex: FPGA).**
  - Synchronous or asynchronous interface (DSP-driven).
  - Address decoding.
  - Careful with priority & interleaved memory access (data integrity).

Ex: ADI SHARC.

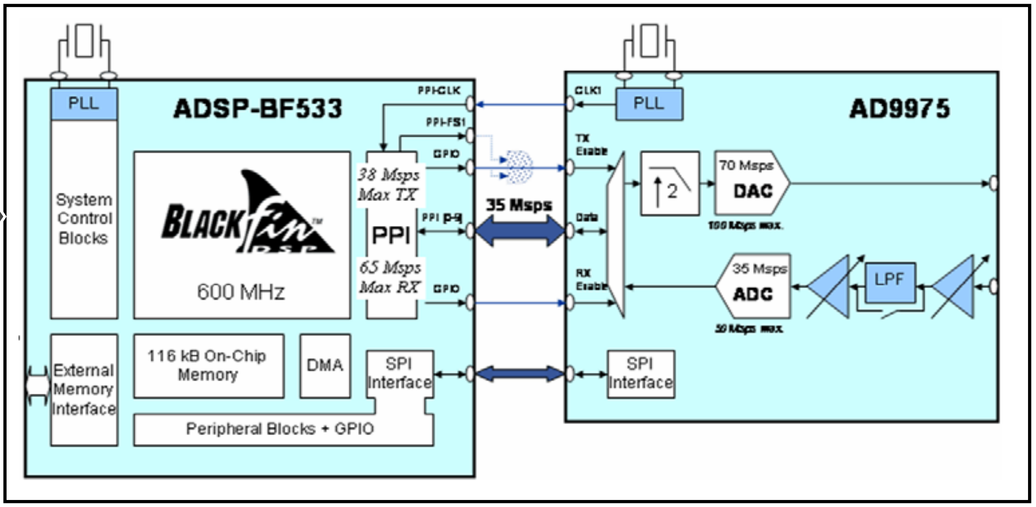


Generic DSP-external memory interfacing scheme.

4.6 Data converters interfacing

- TI: Serial interfaces McBSP, McASP +DMA.
  - Also EMIF in asynchronous mode + DMA.
- ADI: Parallel Peripheral Interface (PPI) on Blackfin .
  - Bidirectional data flow + Serial Port Interface (SPI) to init/configure converter.

ADSP-BF533 Blackfin to AD9975 mixed-signal modem front-end interface.

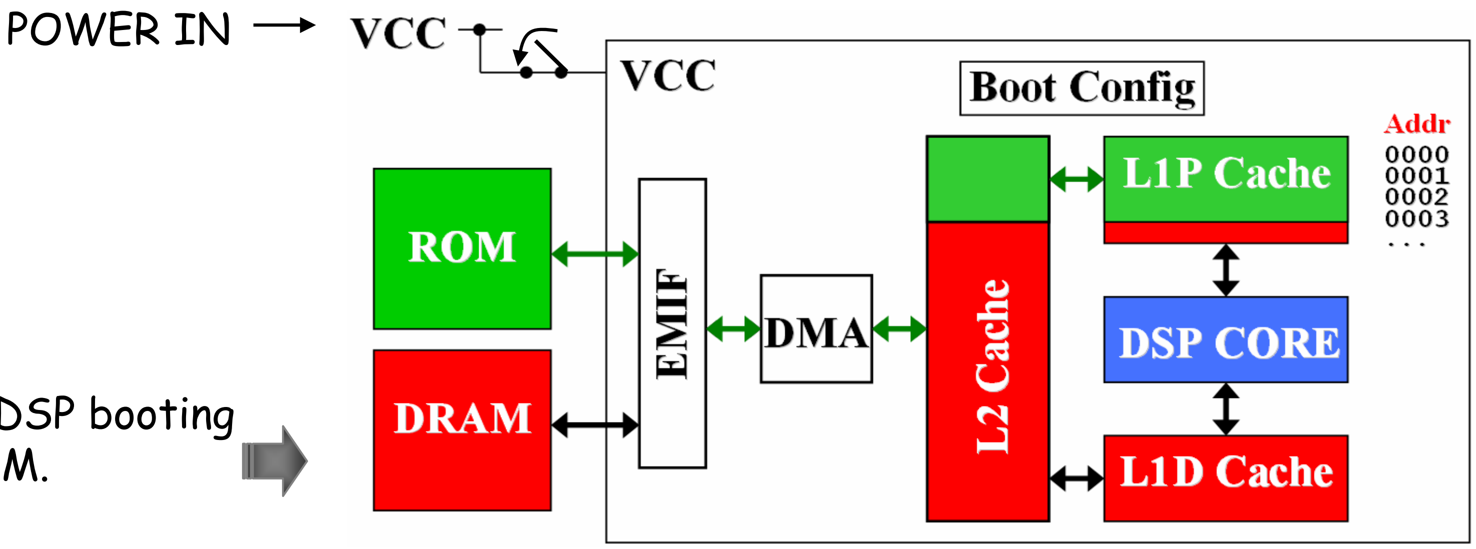


- General solution: FPGA to rebuffer/pre-process (ex.: filtering) data.
- Mixed-signal DSPs: on-board ADC/DAC.

  EX: ADSP-2199x family (8 channels, 14 bit, 20 MSPS ADC).

4.7 DSP booting

- **Debugging**: executable files uploaded to DSP via JTAG.
- **Exploitation**: DSP boots without JTAG.
- Booting mode defined by DSP input pins.
- Methods:
  - **No-boot**: DSP fetches instructions directly from EPROM/FLASH.
  - **ROM boot**: DSP reads formatted boot stream from ROM.
  - **Host boot**: DSP stalled until host configures memory.

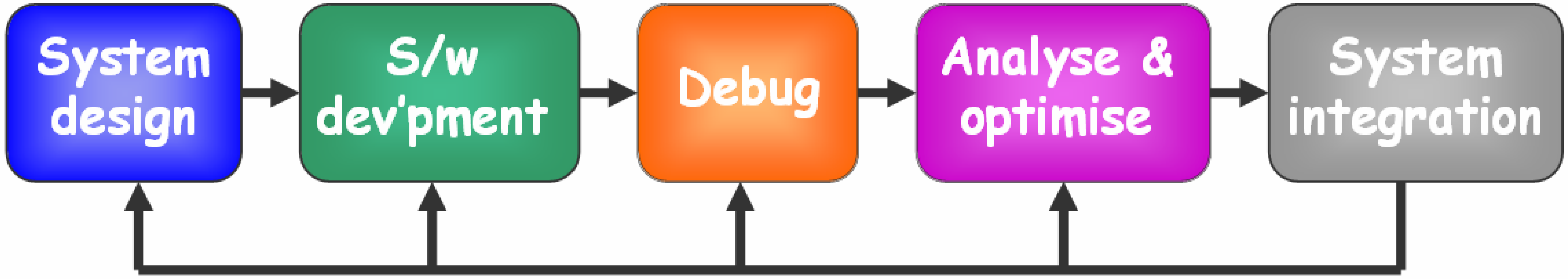


TI 'C6x DSP booting from ROM.

Chapter 4 summary

- Peripherals: wide range & important parameters for DSP choice.
- Interconnect & data I/O: serial + parallel interfaces.
- Services: PLL, timers, JTAG, power management…
- Memory interfaces
  - Dedicated: ex. TI EMIF
  - FPGA: DSP-driven synchronous/asynchronous
- Data converters interfaces
  - Serial or parallel
- JTAG
  - Load code / debug
  - For exploitation DSP boots from memory.

5. RT design flow: introduction



Defines
- architecture
- interfaces
- data flow
- control

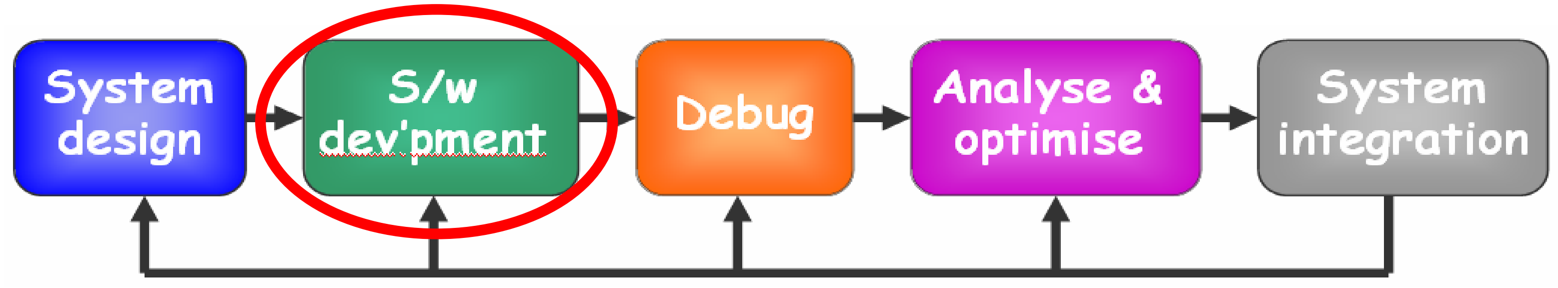
Debugs
- Simulation
- Emulation

Integrates
- within controls infrastructure

Develops
- s/w project(s)
- code
- config. file

Analyses/optimises
- Evaluate performance
- Optimise selected parts

RT design flow: s/w development



6.1 DSP programming - intro.

6.2 Development setup + environment.

6.3 Languages: assembly, C, C++, graphical.

6.4 RTOS.

6.5 Code building process.

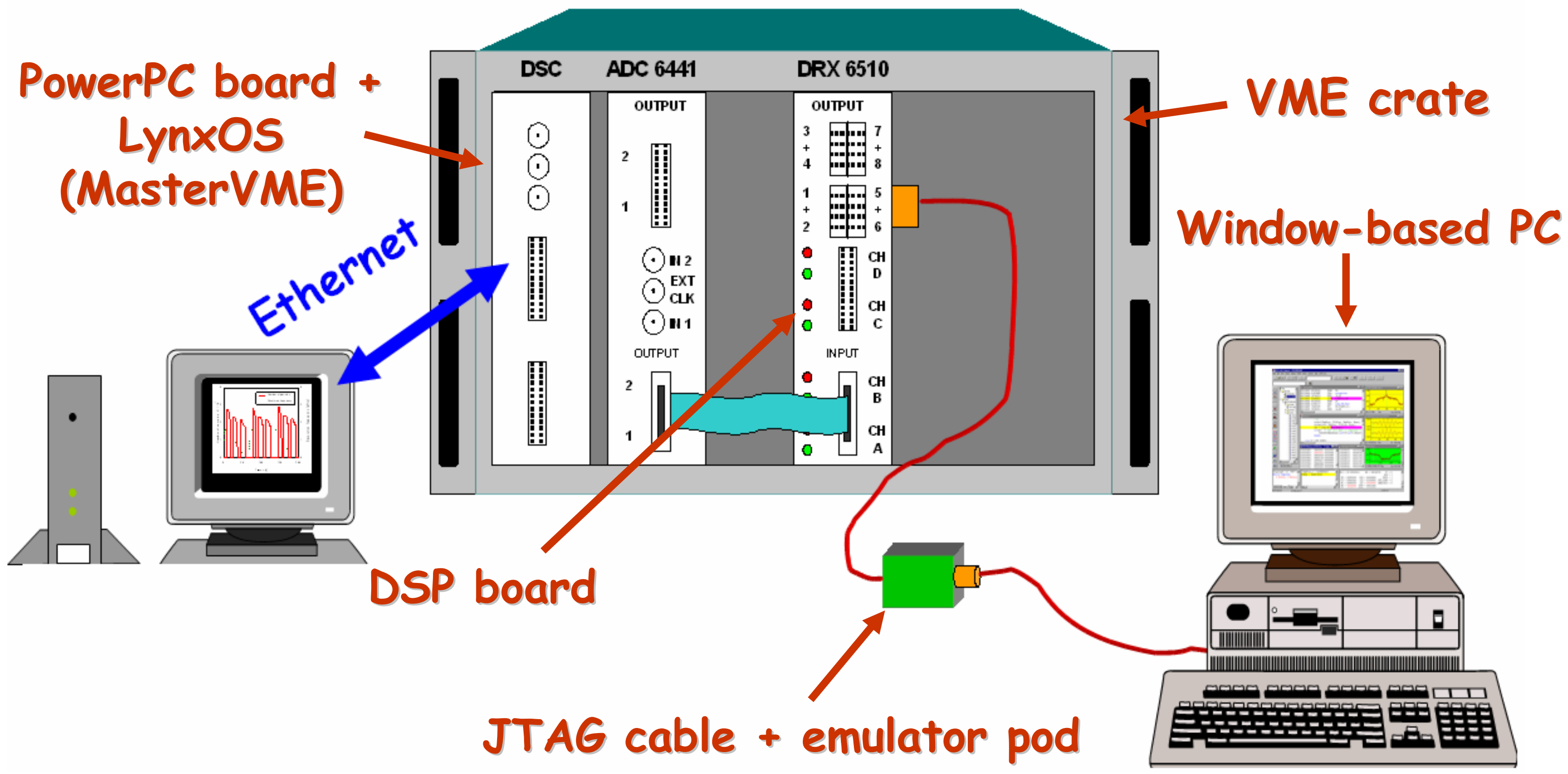
Summary

6.1 DSP programming - intro

- DSPs: programmed by software.
- Languages:
  - Assembly
  - high-level languages (ANSI C, C extensions/dialects, C++ …)
- High-level software tools (ex. MATLAB, National Instruments …) to automatically generate files. → Rapid prototyping!
- Cross-compilation: code developed & compiled on different machine (PC, SUN…) then uploaded to DSP & executed.
- Code building tools from DSP manufacturers.
- Trend: more complex, powerful & user-friendly development tools.

6.2 Development: setup

System use from Control Room DSP code development/debugging



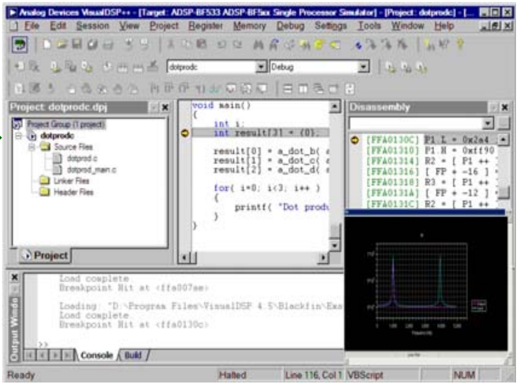
Code development setup. Example: AD beam intensity measurement (TI 'C40 DSP), CERN '98.

6.2 Development: environment

- Integrated Development Environment (IDE): editor, debugger, project manager, profiler.
- Developed & sold (~ 4000 USD) by DSP manufacturer.
- Licenses: mostly per project (not floating).

ADI

- **VisualDSP++** (PC/Windows)
- Two releases: for 16-bit & 32-bit DSPs.
- Licensed, per-family basis.
- Floating licenses available.
- Fully functional, 90-days free evaluation.


Analog Devices VisualDSP++ - [Target: ADSP-BF533 ADSP-BF5xx Single Processor Simulator] - [Project: dotprodc] - [
File Edit Session View Project Register Memory Debug Settings Tools Window Help
dotprodc
Debug
Project: dotprodc.dpj
Project Group (1 project)
dotprodc
Source Files
dotprod.c
dotprod_main.c
Linker Files
Header Files
Project
Disassembly
Output Window
Console
Build
Ready
Halted
Line 116, Col 1
VBScript
NUM

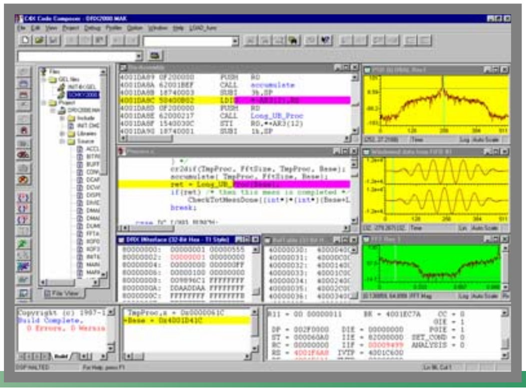
```
void main()
{
    int i;
    int result[3] = {0};

    result[0] = a_dot_b( 
    result[1] = a_dot_c( 
    result[2] = a_dot_d( 

    for( i=0; i<3; i++ )
    {
        printf( "Dot produ
    }
}
```

```
[FFA0130C] P1.L = 0x2a4
[FFA01310] P1.H = 0xff90
[FFA01314] R2 = [ P1 ++
[FFA01316] [ FP + -16 ]
[FFA01318] R3 = [ P1 ++
[FFA0131A] [ FP + -12 ]
[FFA0131C] R2 = [ P1 ++
```

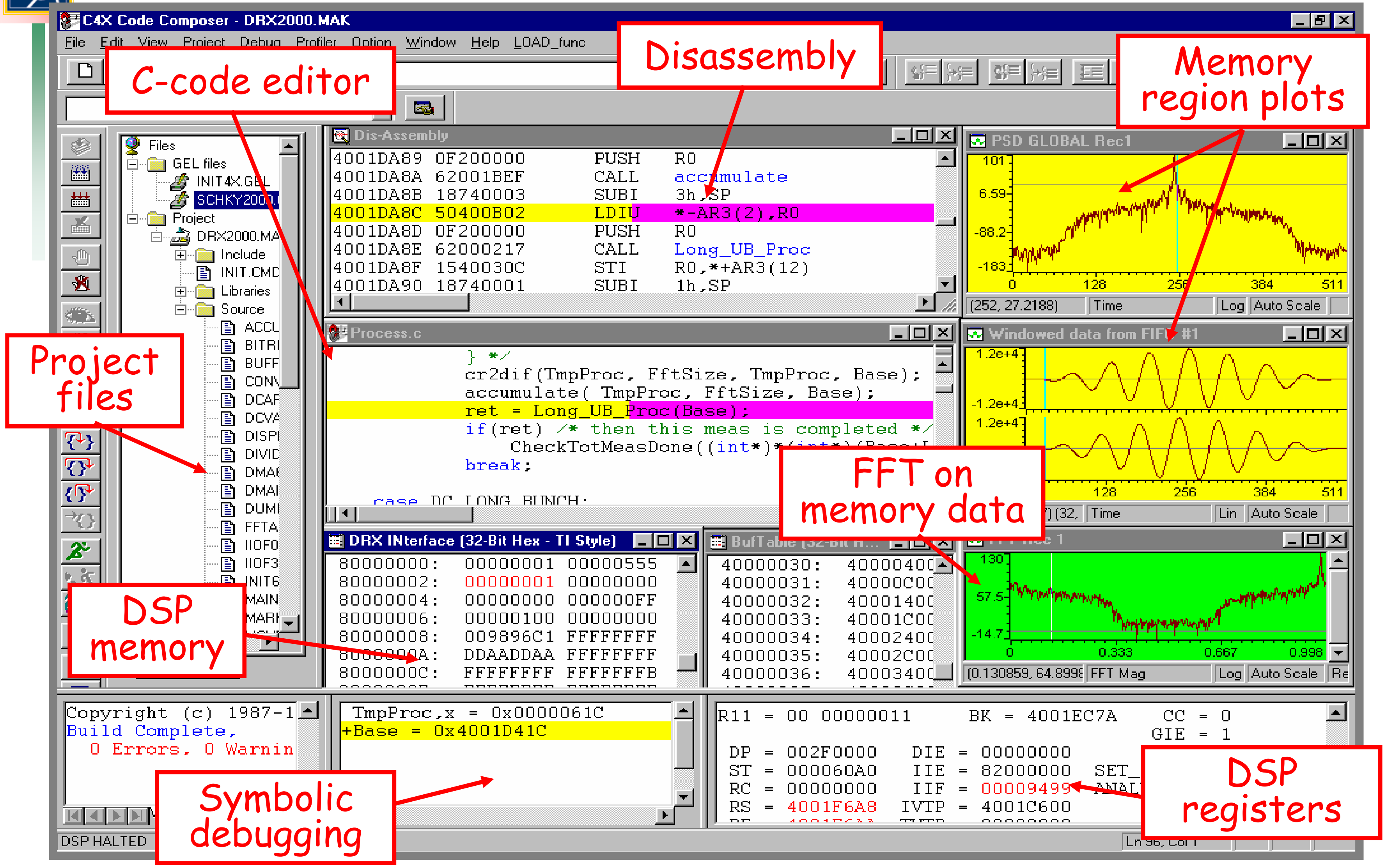
```
Load complete
Breakpoint Hit at <ffa007ae>

Loading: "D:\Program Files\VisualDSP 4.5\Blackfin\Exa
Load complete
Breakpoint Hit at <ffa0130c>
>>
```

TI

- **Code Composer Studio** (mostly PC/Windows).
- Different version each family.
- No floating licenses.
- Fully functional, 90-days free evaluation.



6.2 Development: Code Composer Studio



Dis-Assembly
```
4001DA89  0F200000     PUSH   R0
4001DA8A  62001BEF     CALL   accumulate
4001DA8B  18740003     SUBI   3h,SP
4001DA8C  50400B02     LDIU   *-AR3(2),R0
4001DA8D  0F200000     PUSH   R0
4001DA8E  62000217     CALL   Long_UB_Proc
4001DA8F  1540030C     STI    R0,*+AR3(12)
4001DA90  18740001     SUBI   1h,SP
```

Process.c
```
          } */
          cr2dif(TmpProc, FftSize, TmpProc, Base);
          accumulate( TmpProc, FftSize, Base);
          ret = Long_UB_Proc(Base);
          if(ret) /* then this meas is completed */
              CheckTotMeasDone((int*)*(int*)(Base+I
          break;

     case DC_LONG_BUNCH:
```

DRX INterface [32-Bit Hex - TI Style]
```
80000000:  00000001  00000555
80000002:  00000001  00000000
80000004:  00000000  000000FF
80000006:  00000100  00000000
80000008:  009896C1  FFFFFFFF
8000000A:  DDAADDAA  FFFFFFFF
8000000C:  FFFFFFFF  FFFFFFFB
```

BufTable [32-Bit H...
```
40000030:  4000040C
40000031:  40000C0C
40000032:  4000140C
40000033:  40001C0C
40000034:  4000240C
40000035:  40002C0C
40000036:  4000340C
```

```
[252, 27.2188]
[0.130859, 64.8998
```

```
Copyright (c) 1987-1
Build Complete,
  0 Errors, 0 Warnin
```

```
TmpProc,x = 0x0000061C
+Base = 0x4001D41C
```

```
R11 = 00 00000011     BK = 4001EC7A     CC = 0
                                        GIE = 1
 DP = 002F0000    DIE = 00000000
 ST = 000060A0    IIE = 82000000    SET_
 RC = 00000000    IIF = 00009499    ANAL
 RS = 4001F6A8   IVTP = 4001C600
```

Code Composer for TI 'C40 DSPs – screen dump taken in 1999.

6.3 Programming languages

- Choice of programming language: depends on processor
  - supported languages
  - workload → optimisation level.
- Now many choices:
  - compilers generate efficient code
  - hand-optimising difficult: h/w complexity!
- Main choices:

a) Assembly

b) High-Level Languages (HLL): ANSI/ISO C, C extensions, C++

c) Graphical languages

6.3a) Assembly

- Code next to the machine: works with registers.
- Needed: DSP architecture detailed knowledge.
- Takes longer to develop/to understand other people's code.
- Grammar/style depends on manufacturer / DSP family.

→ Limited portability / reusability.

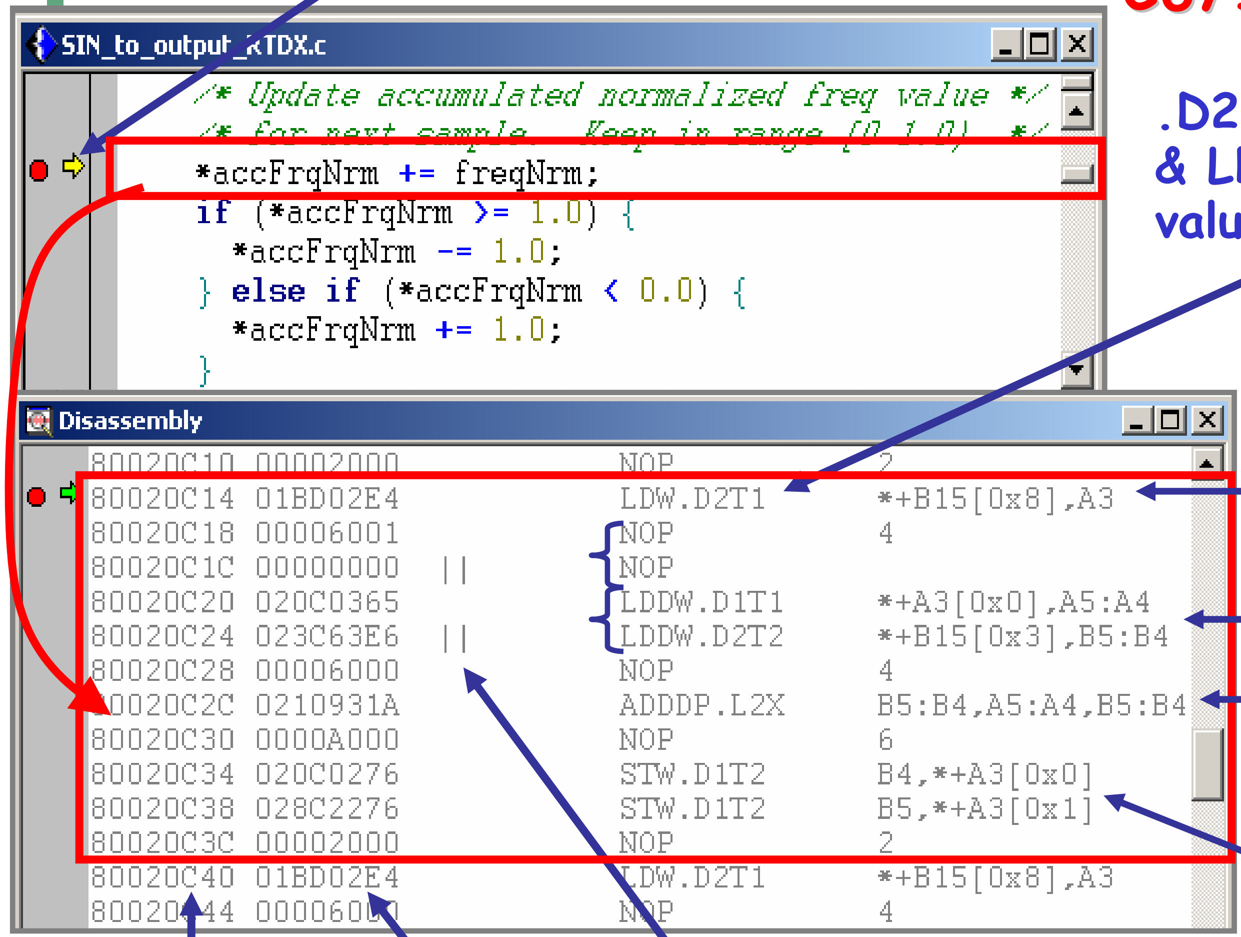
Assembly stiles comparison.

Operation	Traditional assembly	Algebraic assembly
Move registers contents	mov R7, R0	R7 = R0
Addition	add R0, R1, R2	R0 = R1 + R2

Conditional jump	beq R1, R2, _loc	comp (R1,R2);
		if eq jump _loc;

6.3a) Assembly [2]

C6713 assembly example



```
/* Update accumulated normalized freq value */
/* for next sample.  Keep in range [0 1.0)  */
    *accFrqNrm += freqNrm;
    if (*accFrqNrm >= 1.0) {
      *accFrqNrm -= 1.0;
    } else if (*accFrqNrm < 0.0) {
      *accFrqNrm += 1.0;
    }
```

```
80020C10  00002000          NOP          2
80020C14  01BD02E4          LDW.D2T1     *+B15[0x8],A3
80020C18  00006001          NOP          4
80020C1C  00000000     ||   NOP
80020C20  020C0365          LDDW.D1T1    *+A3[0x0],A5:A4
80020C24  023C63E6     ||   LDDW.D2T2    *+B15[0x3],B5:B4
80020C28  00006000          NOP          4
80020C2C  0210931A          ADDDP.L2X    B5:B4,A5:A4,B5:B4
80020C30  0000A000          NOP          6
80020C34  020C0276          STW.D1T2     B4,*+A3[0x0]
80020C38  028C2276          STW.D1T2     B5,*+A3[0x1]
80020C3C  00002000          NOP          2
80020C40  01BD02E4          LDW.D2T1     *+B15[0x8],A3
80020C44  00006000          NOP          4
```

.D2 unit generates address & LD1 data path places value →A register file

Load 32-bit →A3

Load 2x32 bit →{A5,A4} & 2x32 bit →{B5,B4}

Add 2x32 bit →(B5,B4}

Instruction address

Machine code

Parallel instructions

Store → memory

6.3b) ANSI/ISO C language

- ☺ Popular/known → easier (faster) than assembly to develop.
- ☺ Supports control structures & low-level bit manipulation.
- ☺ Understandable & ~ portable (*but* limitations!).

- ☹ Typically slower & larger code size
- ☹ No support for DSP h/w features (ex: circular buffers, non-flat memory space) & fixed point fractional variables.
- ☹ C compiler data alignment may be incompatible with DSP → bus errors
- ☹ C compiler data-type sizes not standardized: may not fit DSP native data sizes!
  → s/w emulation (***slow***) replaces h/w implementation (***fast***).
  Ex: ADI TigerSHARC 64-bit double operations.

6.3b) ANSI/ISO C language [2]

"portable C" is machine-dependent (if you want *efficient* code)!

Data-type sizes on different DSPs.

'C6713 DSP		
Data type	**# bits**	**Representation**
char	8	ASCII
short	16	2's compl. / binary
int	32	2's compl. / binary
long	40	2's compl. / binary
float	32	IEEE 32-bit
double	64	IEEE 64-bit

C6713: h/w support for single & double precision float operations!

int size	Processor
16	ADI '21xx, TI 'C54, C55
24	Freescale 56x
32	ADI Blackfin, TI 'C6x
32	ADI SHARC, TigerSHARC

char size	Processor
8	ADI Blackfin
16	ADI '21xx, TI 'C54, 'C55
24	Freescale 56x
32	ADI Blackfin, TI 'C6x
32	ADI SHARC, TigerSHARC

6.3b) ANSI/ISO C language [3]

- "Embedded" C widely used on DSPs
  - **Intrinsics**: operators converted to efficient assembly code.

Intrinsic	Description
double _rsqrdp(double src);	Returns approximate 64-bit double square root reciprocal
double _fabs(double src);	Returns absolute value of src
unit _enable_interrupts(void);	Returns previous interrupt state & enables interrupts

Ex: some C6713 intrinsics

  - **C-language extensions**: specialised data type/constructs added.

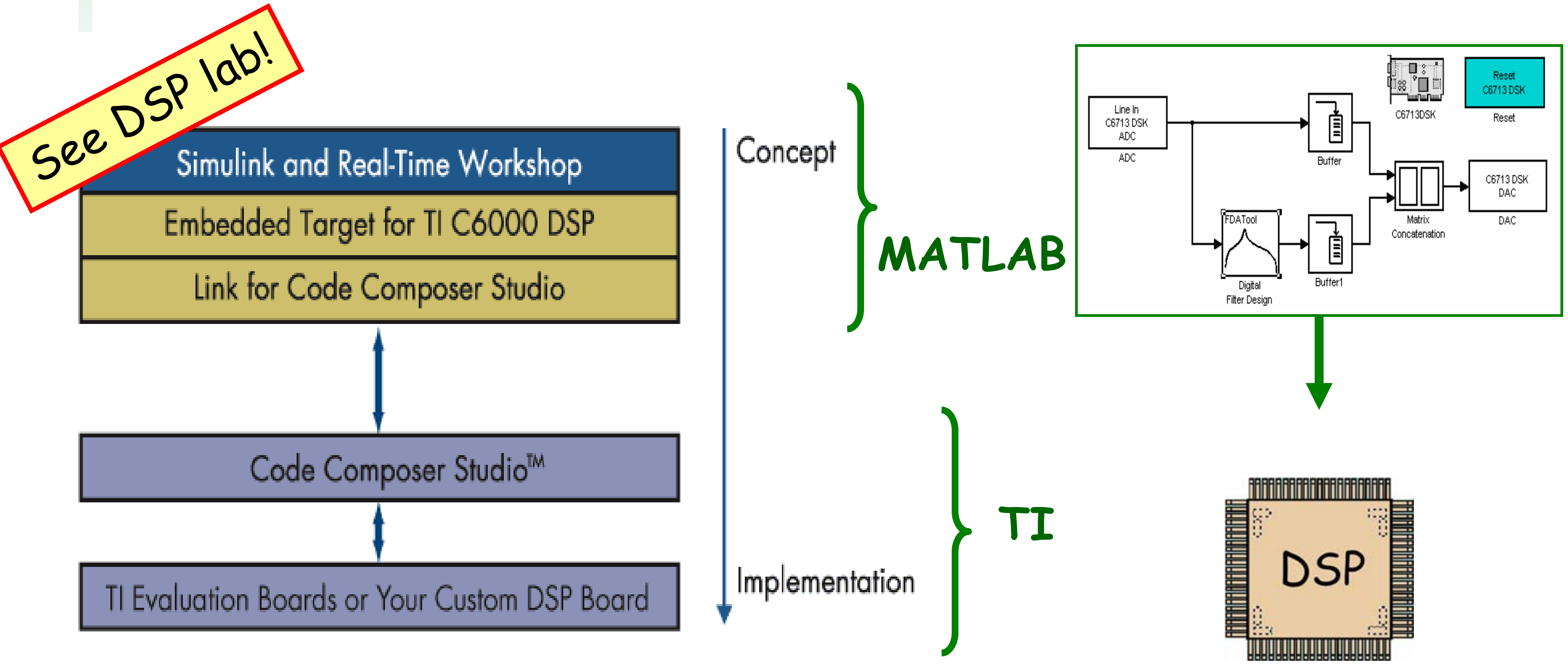
NB: Project "build" options often allows forcing ANSI C compatibility.

- "Embedded" C++ used on DSPs, too.
  - Trimmed version: no multiple inheritance, exception handling → more efficient code & smaller executables.

6.3c) Graphical DSP programming

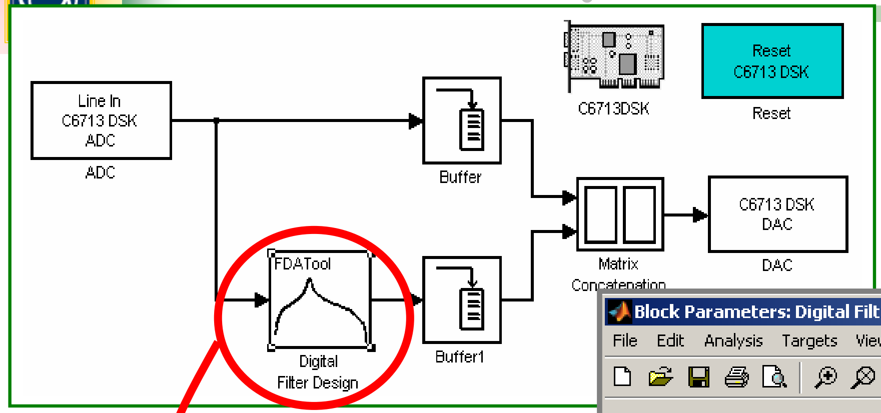
- Graphical programming can also generate DSP code.
- Ex: Matlab, Hypersignal RIDE (now NI), LabVIEW DSP Module.

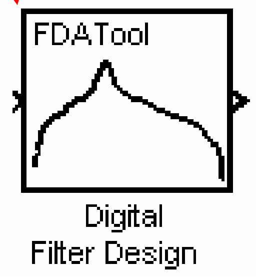
Matlab: generates source files from model, compiles & upload to DSPs.



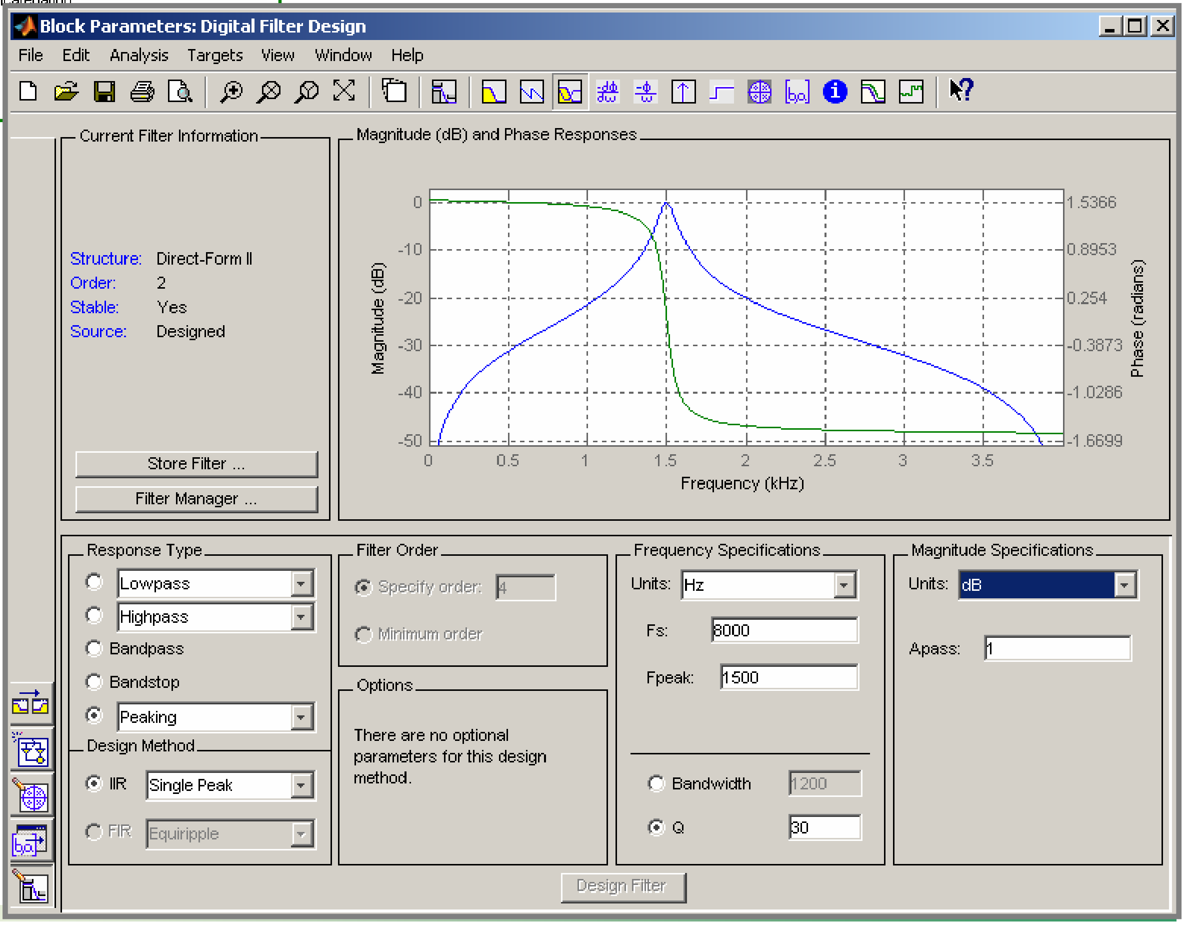
6.3c) Graphical DSP programming [2]

Example: MATLAB





Digital filter block



Current Filter Information

Structure:	Direct-Form II
Order:	2
Stable:	Yes
Source:	Designed

Response Type: Lowpass, Highpass, Bandpass, Bandstop, [✓ Peaking]

Design Method: [✓ IIR] Single Peak; FIR Equiripple

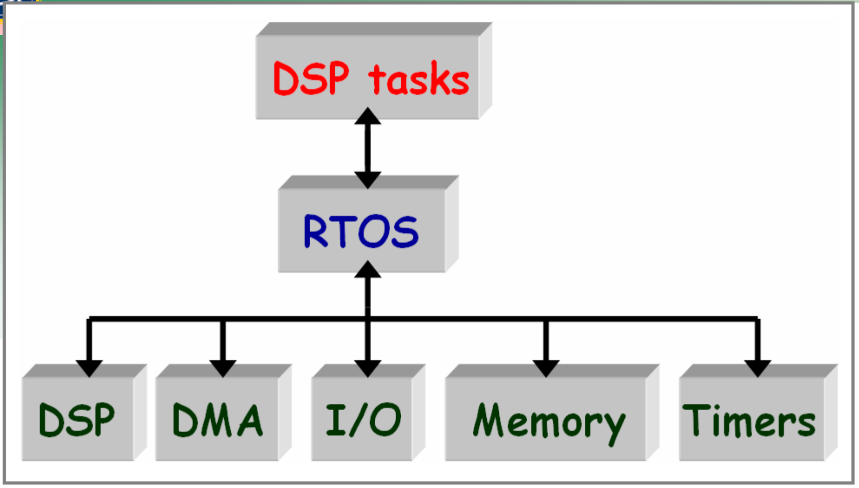
Filter Order: [✓ Specify order:] 4; Minimum order

Options: There are no optional parameters for this design method.

Frequency Specifications: Units: Hz; Fs: 8000; Fpeak: 1500; Bandwidth 1200; [✓ Q] 30

Magnitude Specifications: Units: dB; Apass: 1

6.4 RTOS



Embedded DSP software component.

RTOS

- loaded to DSP @boot time.
- manages DSP programs (*tasks*).
- uses DSP resources (ex: timers).
- *API for tasks-peripherals interfacing.*

Typical features

- Task-based + priorities (***scheduler***).
- Multi-tasking: time-sharing, often preemptive (*NOT* cooperative).
- Small memory footprint.

Advantages

- H/w abstraction
- Task management
- System debug & optimisation
- Memory protection …

6.4 RTOS [2]

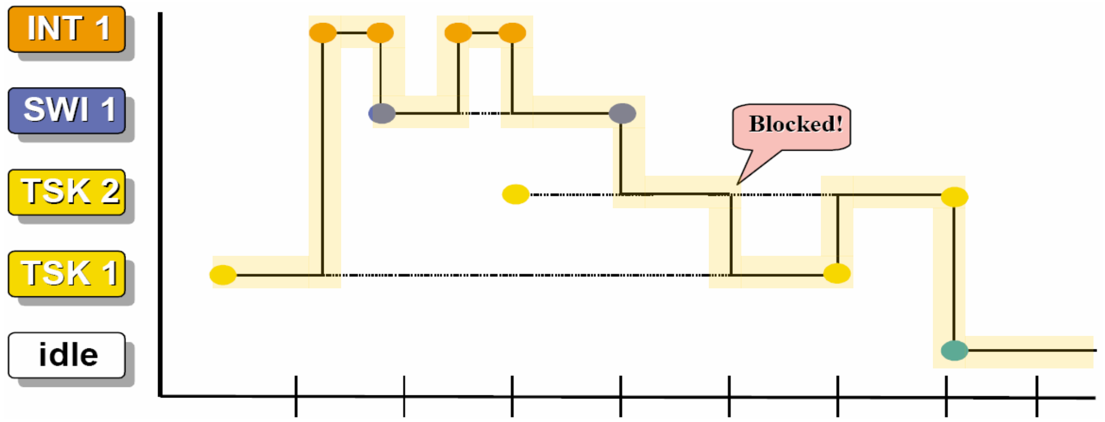
- High RTOS turnover + royalties often required.
- Embedded Linux: uClinux (soft-RT), RT-Linux, RTAI.
- ADI & TI: scalable RTOS to *optionally* include in DSP code.

ADI: VisualDSP++ Kernel (VDK).

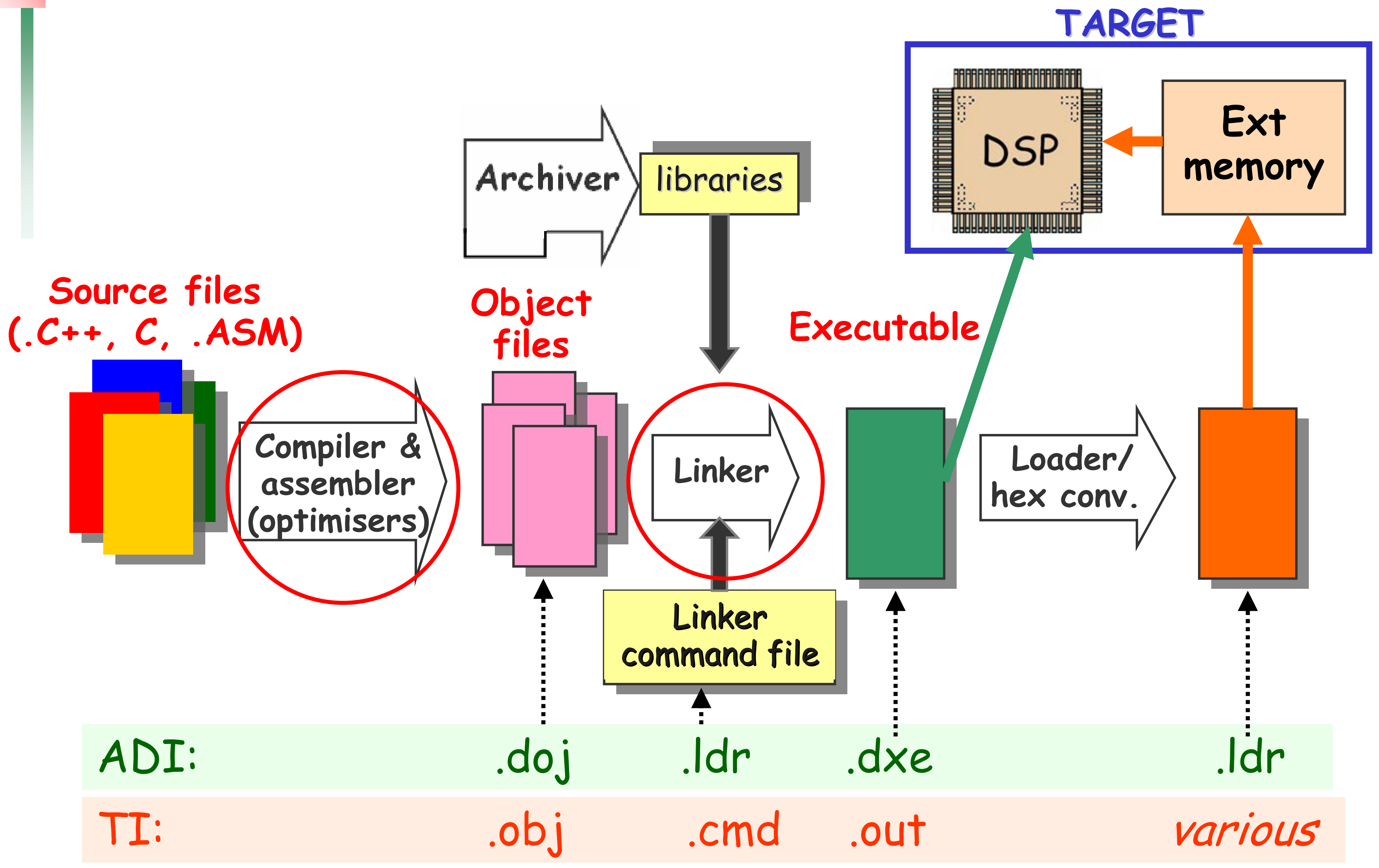
TI: DSP BIOS Kernel

- Preemptive scheduler + multitasking support.
- Chip Support Library (**CSL**) to control on-chip peripherals.
- Real Time Data eXchange (**RTDX**) support [→ chapter 7]

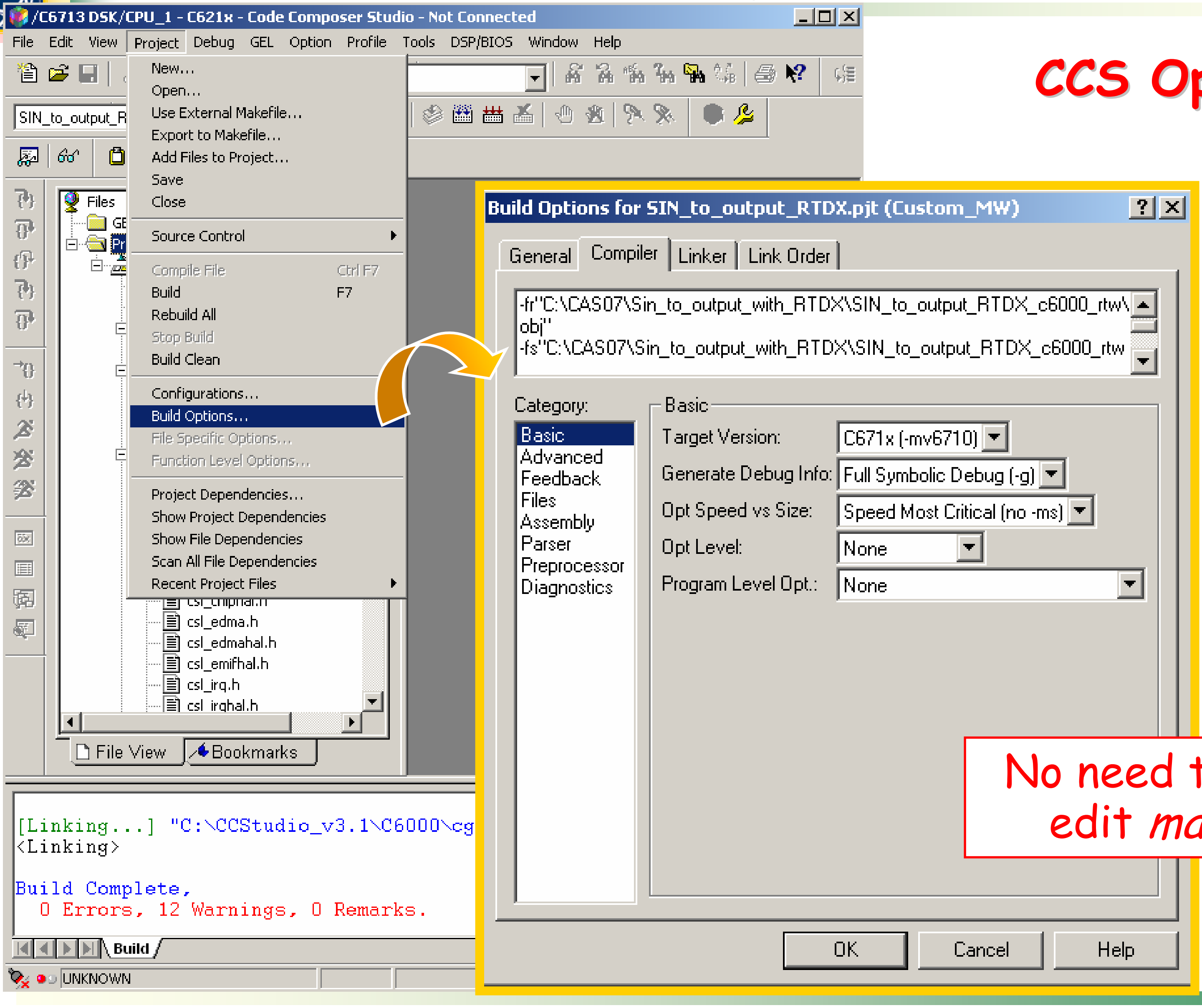
DSP BIOS tasks ⇨



6.5 Code building process



6.5 Code building process [2]



```
[Linking...] "C:\CCStudio_v3.1\C6000\cg
<Linking>

Build Complete,
  0 Errors, 12 Warnings, 0 Remarks.
```

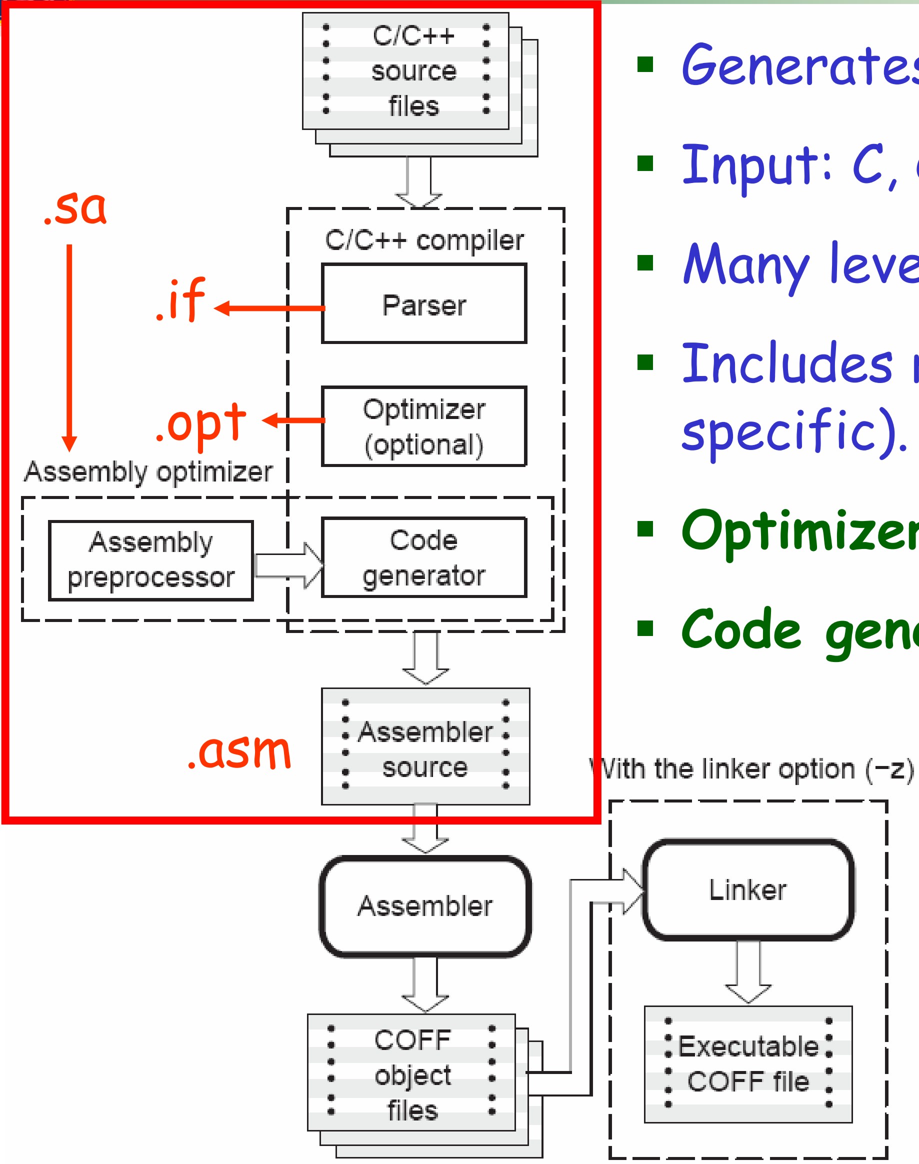
```
-fr"C:\CAS07\Sin_to_output_with_RTDX\SIN_to_output_RTDX_c6000_rtw\
obj"
-fs"C:\CAS07\Sin_to_output_with_RTDX\SIN_to_output_RTDX_c6000_rtw
```

Category: Basic, Advanced, Feedback, Files, Assembly, Parser, Preprocessor, Diagnostics

Basic
- Target Version: C671x (-mv6710)
- Generate Debug Info: Full Symbolic Debug (-g)
- Opt Speed vs Size: Speed Most Critical (no -ms)
- Opt Level: None
- Program Level Opt.: None

CCS Options

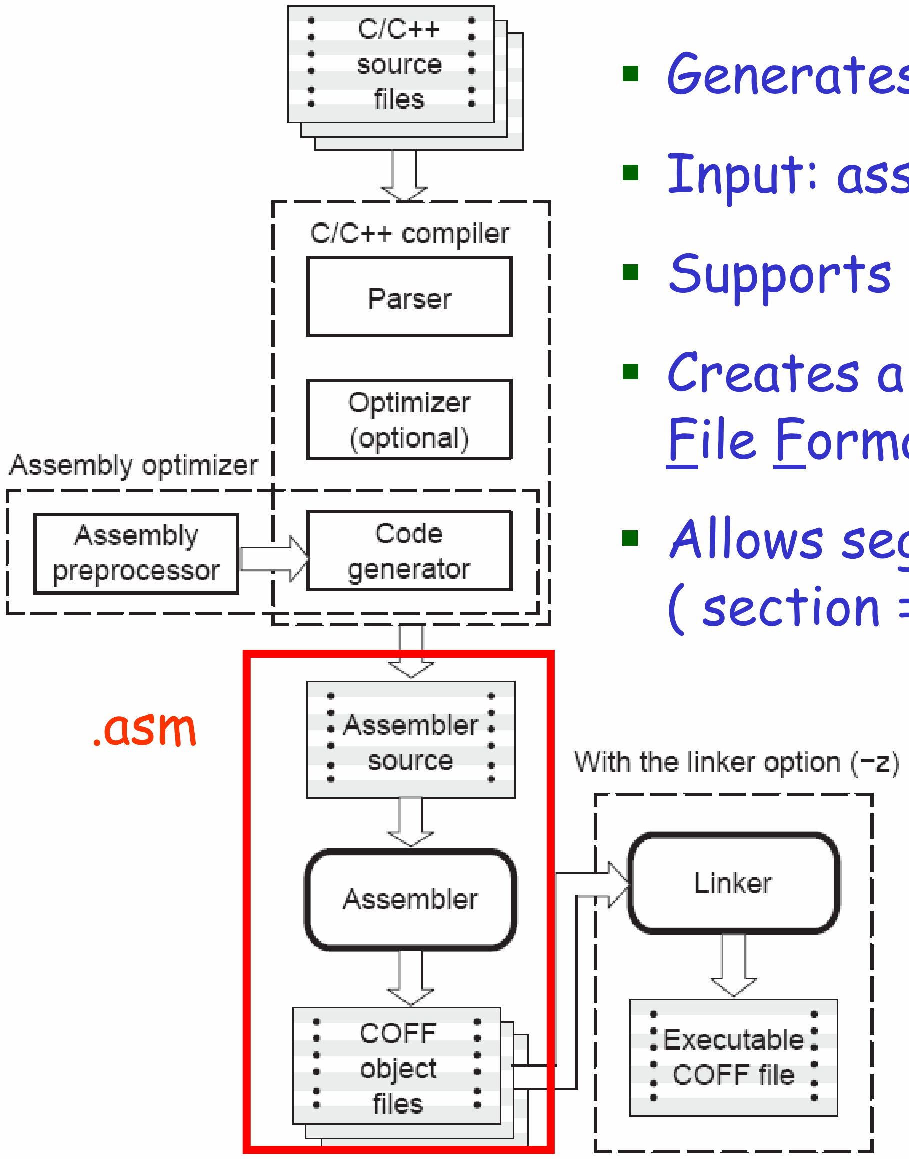
No need to manually edit *makefiles* !



- Generates 'C6x assembly code.
- Input: C, C++ & linear assembly files.
- Many levels of optimisation* (selectable).
- Includes real-time library (non target-specific).
- **Optimizer**: high-level optimisation.
- **Code generator**: target-specific optimisation.
- **Assembly optimiser**: hand-written linear assembly (extension .sa) optimisation.

* *Optimisation: may modify code behaviour!* [→ chapter 8]

6.5b) Assembler: TI 'C6x

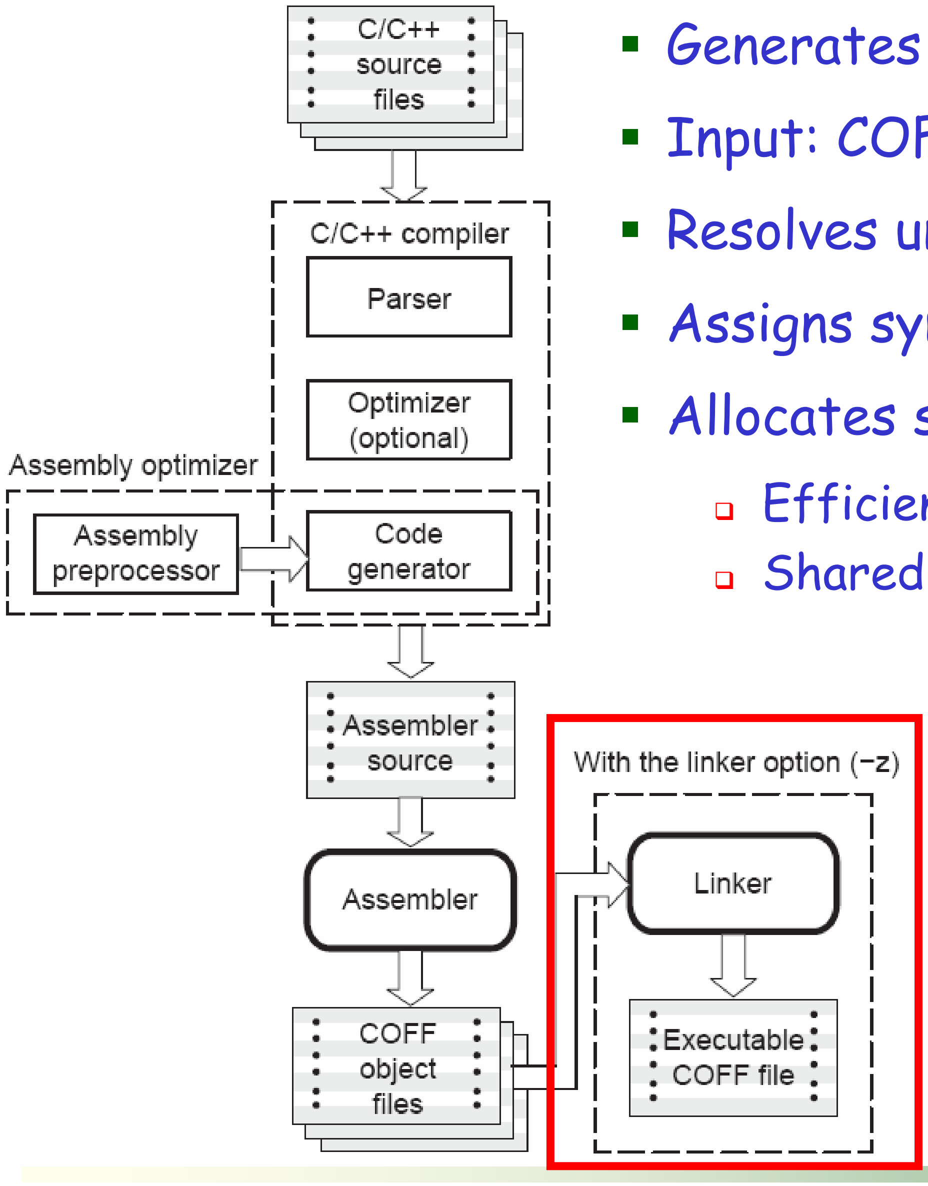


- Generates machine language object files
- Input: assembly files.
- Supports macros (inline/library).
- Creates a object file: Common Object File Format (**COFF**).
- Allows segmenting code into *sections* (section = smaller unit of object file).

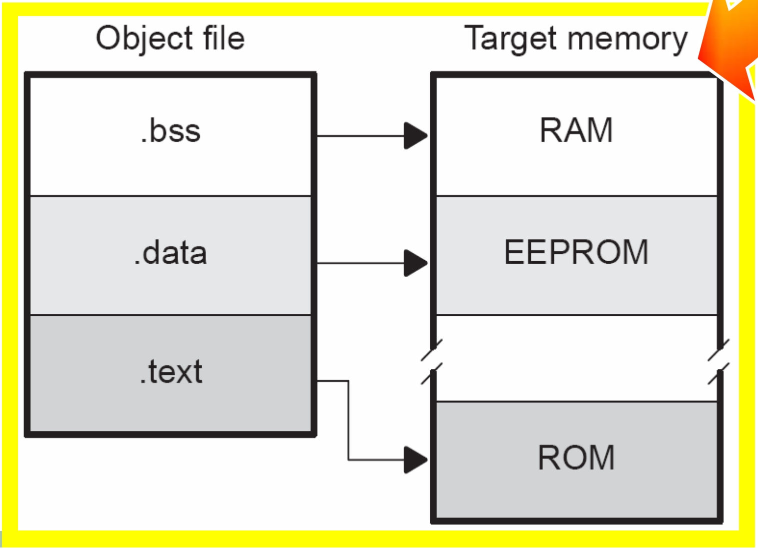
COFF basic sections

- **.text**: executable code
- **.data**: initialised data
- **.bss**: space for un-initialised variables.

6.5c) Linker: TI 'C6x

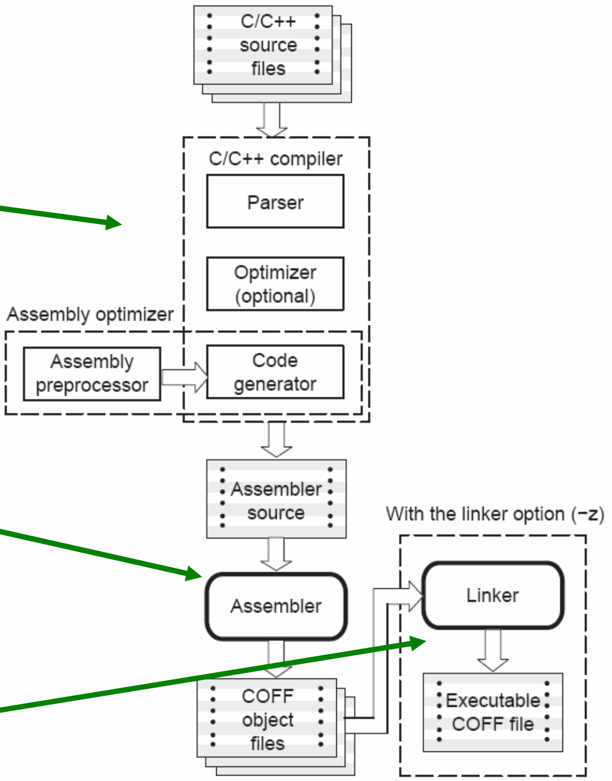


- Generates executable modules.
- Input: COFF object files.
- Resolves undefined external references.
- Assigns symbols/section final addresses.
- Allocates sections:
  - Efficient memory access speed.
  - Shared memory map implementation.



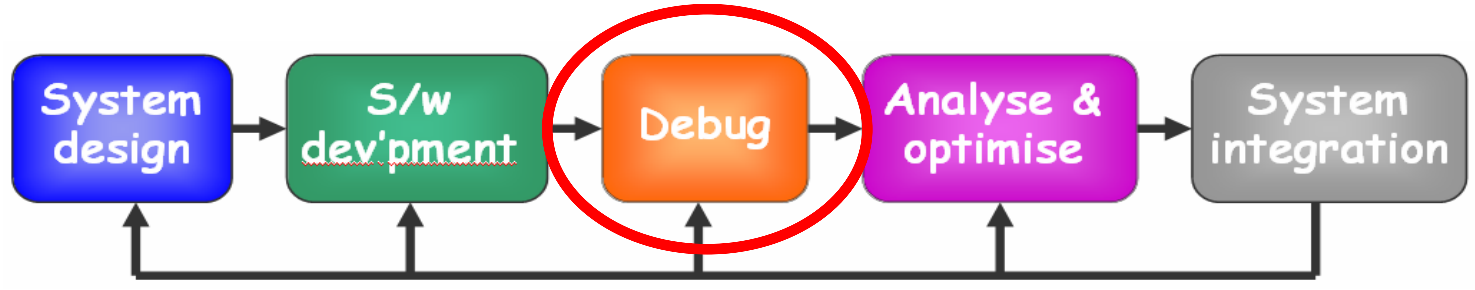
Chapter 6 summary

- DSPs: programmed by s/w via manufacturer-provided development environment.
- Languages: assembly, C, C++, graphical…
- RTOS available for task/resource management.
- Code building process:
  - **Compiler**:
    - generates assembly code.
    - provides code optimisation
  - **Assembler**:
    - Generates machine code
  - **Linker**:
    - generates executable modules
    - allocates sections to memory.



Chapter 7 topics

RT design flow: debugging



7.1 Bugs & debugging

7.2 Simulation

7.3 Emulation

7.4 Traditional emulation techniques

7.5 Real-time debugging

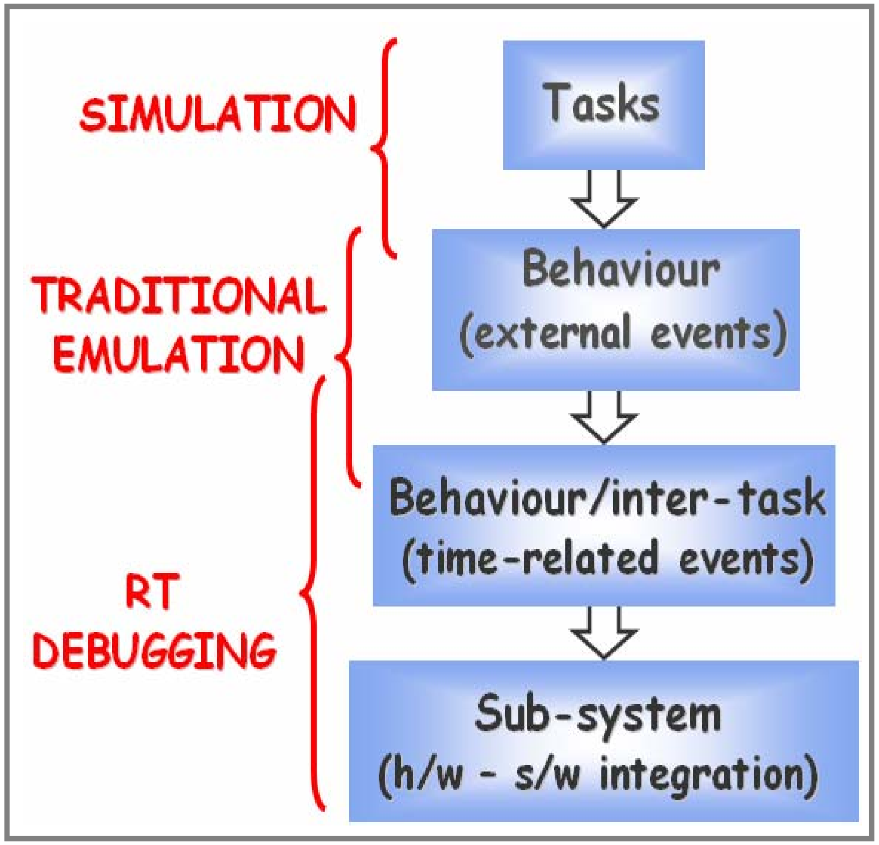
Summary

7.1 Bugs & debugging

Executable code: no compilation/linker errors but … **does it do what it should?**

- **Bugs**:
  - Repeatable
  - Intermittent (*tough!*)
  - Due to implementation : src code
  - Not due to implementation : h/w…
- **Approaches**:
  - Simulation
  - Traditional emulation
  - Real-time debugging
- **First debug, then switch optimisation ON [→ chapter 8]**

Debugging phase: most critical & least predictable !


SIMULATION
TRADITIONAL EMULATION
RT DEBUGGING
Tasks
Behaviour (external events)
Behaviour/inter-task (time-related events)
Sub-system (h/w - s/w integration)

Test & debugging steps

7.2 Simulation

- S/w DSP simulator: included with development environment.
- Simulated: $\left\{ \begin{array}{l} \text{CPU instruction set} \\ \text{Peripherals (ex: EDMA, caches…)} \\ \text{External interrupts} \end{array} \right.$

☺ Highly repeatable! Ex: external events difficult to *exactly* repeat in h/w.

☺ Task testing. Ex: algorithms, logical errors …

☺ Measurement of execution duration (*CAVEAT*: limitations!).

☺ Testing possible before h/w available.

☺ TI CCS: ***rewind*** feature with 'C5x/'C6x simulators.

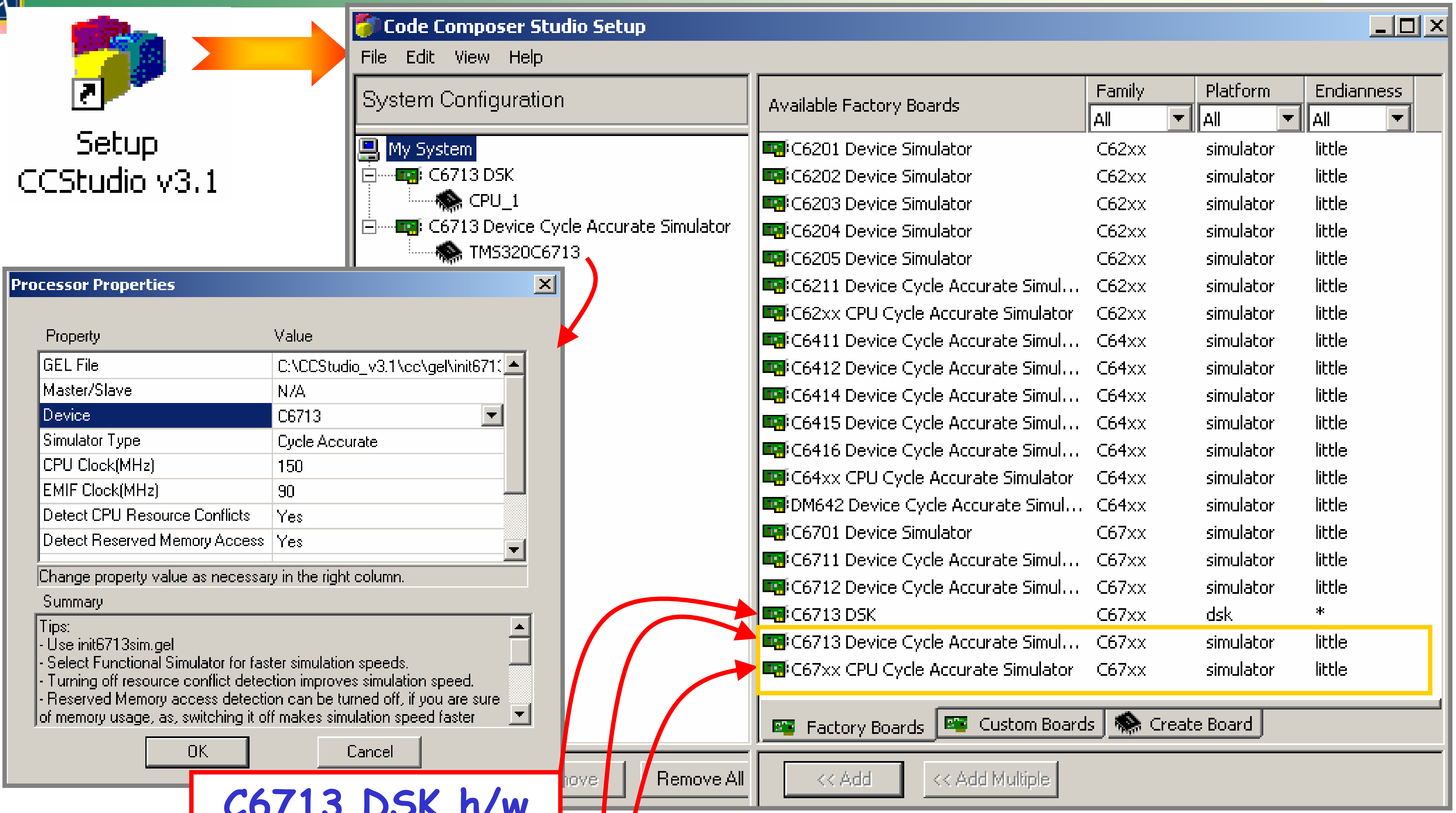
☹ S/w simulation: slower than real h/w.

$\longrightarrow$ Often different simulators for same target.

- Traditional + real-time debug techniques available with simulation.

7.2 Simulation: TI C6x Simulators

Setup CCStudio v3.1



System Configuration

- My System
  - C6713 DSK
    - CPU_1
  - C6713 Device Cycle Accurate Simulator
    - TMS320C6713

Available Factory Boards	Family	Platform	Endianness
	All	All	All
C6201 Device Simulator	C62xx	simulator	little
C6202 Device Simulator	C62xx	simulator	little
C6203 Device Simulator	C62xx	simulator	little
C6204 Device Simulator	C62xx	simulator	little
C6205 Device Simulator	C62xx	simulator	little
C6211 Device Cycle Accurate Simul...	C62xx	simulator	little
C62xx CPU Cycle Accurate Simulator	C62xx	simulator	little
C6411 Device Cycle Accurate Simul...	C64xx	simulator	little
C6412 Device Cycle Accurate Simul...	C64xx	simulator	little
C6414 Device Cycle Accurate Simul...	C64xx	simulator	little
C6415 Device Cycle Accurate Simul...	C64xx	simulator	little
C6416 Device Cycle Accurate Simul...	C64xx	simulator	little
C64xx CPU Cycle Accurate Simulator	C64xx	simulator	little
DM642 Device Cycle Accurate Simul...	C64xx	simulator	little
C6701 Device Simulator	C67xx	simulator	little
C6711 Device Cycle Accurate Simul...	C67xx	simulator	little
C6712 Device Cycle Accurate Simul...	C67xx	simulator	little
C6713 DSK	C67xx	dsk	*
C6713 Device Cycle Accurate Simul...	C67xx	simulator	little
C67xx CPU Cycle Accurate Simulator	C67xx	simulator	little

Property	Value
GEL File	C:\CCStudio_v3.1\cc\gel\init671:
Master/Slave	N/A
Device	C6713
Simulator Type	Cycle Accurate
CPU Clock(MHz)	150
EMIF Clock(MHz)	90
Detect CPU Resource Conflicts	Yes
Detect Reserved Memory Access	Yes

Change property value as necessary in the right column.

Summary

Tips:
- Use init6713sim.gel
- Select Functional Simulator for faster simulation speeds.
- Turning off resource conflict detection improves simulation speed.
- Reserved Memory access detection can be turned off, if you are sure of memory usage, as, switching it off makes simulation speed faster

C6713 DSK h/w

CPU Cycle Accurate Simulator: models instruction set.

Device Cycle Accurate Simulator: models instruction set + device peripherals.

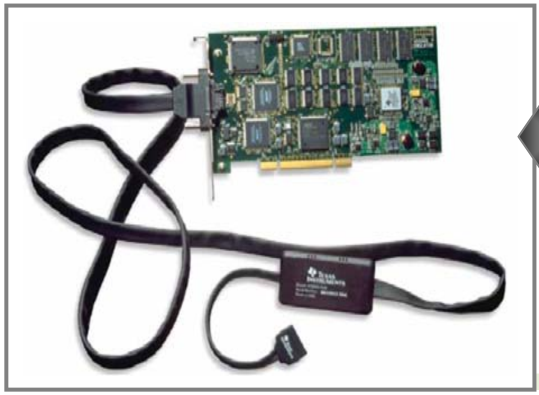
7.3 Emulation

- **System-on-a-Chip (SOC):** system functionality *[processor, memory, logic elements, peripherals…]* on single silicon chip..
  - ☺ Small-size devices: faster, cheaper, reliable, low power …
  - ☹ **Vanishing visibility**: a) impossible to probe pins (BGA packages); b) many signals not available @pins anyhow.

- **Emulation**: debug components embedded to restore visibility.
  - **Monitor-based emulation**: supervisor program (*monitor*) runs on DSP.
  - **Pod-based In Circuit Emulation (ICE)**: DSP replaced by special version with additional h/w (*emulator pod*).
  - **Scan-based emulation (JTAG)**: debugging logic + dedicated interface added to DSP.

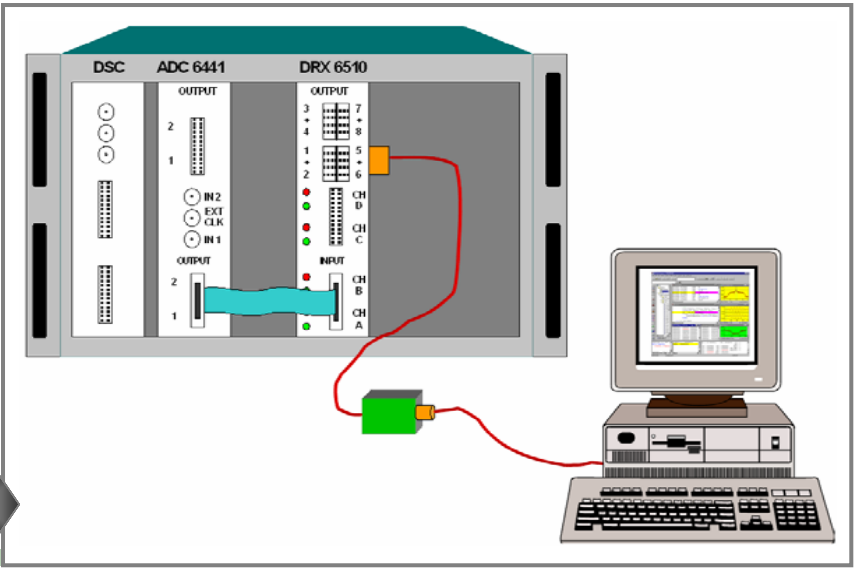
7.3 Emulation: scan-based [2]

- **Components**:
  - On-chip debug facilities
  - Emulation controller: controls info flow to/from target.
    - *Functions*: run control + capture/record DSP activity.
    - *Location*: external pod or on DSP board.
  - Debugger program on host: visualization & user interface
- **Capabilities:** visibility into DSP processor, registers, memory.
- **Interfaces**:
  - DSP board: 14-pin header, USB
  - Host: Parallel/PCI/USB…



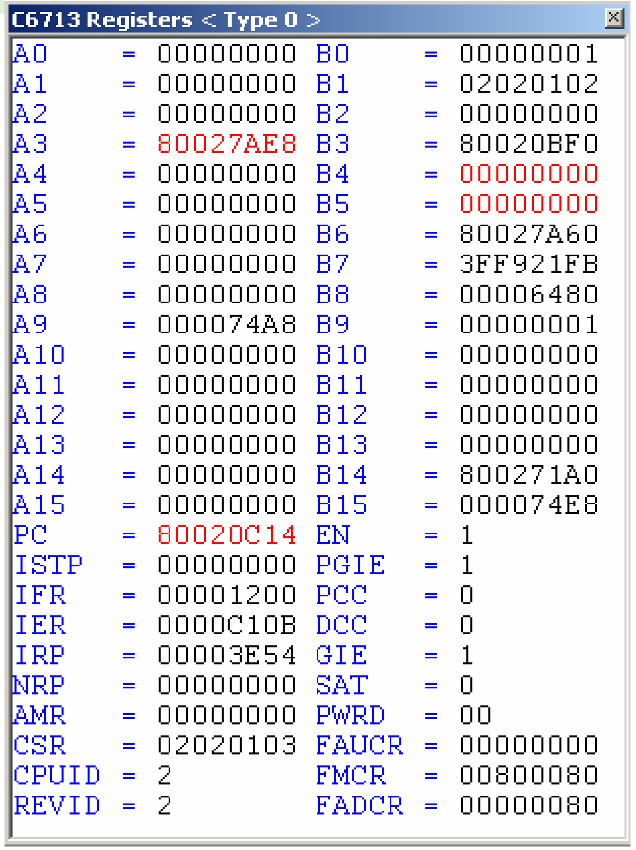
TI XDS560 emulator

Debug setup example



7.4 Traditional emulation techniques

- **Source-level debugging**
  - See assembly-executed instruction
  - Variables/memory accessed via name/address.
- **Breakpoints**
  - Freeze entire DSP → examine registers, plot memory range, dump data to file…
  - Software: replace instruction with one creating exception.
  - Hardware: address monitoring stops execution for specified code fetch.
  - Triggerable by event detectors.



C6713 Registers < Type 0 >

```
A0     = 00000000 B0     = 00000001
A1     = 00000000 B1     = 02020102
A2     = 00000000 B2     = 00000000
A3     = 80027AE8 B3     = 80020BF0
A4     = 00000000 B4     = 00000000
A5     = 00000000 B5     = 00000000
A6     = 00000000 B6     = 80027A60
A7     = 00000000 B7     = 3FF921FB
A8     = 00000000 B8     = 00006480
A9     = 000074A8 B9     = 00000001
A10    = 00000000 B10    = 00000000
A11    = 00000000 B11    = 00000000
A12    = 00000000 B12    = 00000000
A13    = 00000000 B13    = 00000000
A14    = 00000000 B14    = 800271A0
A15    = 00000000 B15    = 000074E8
PC     = 80020C14 EN     = 1
ISTP   = 00000000 PGIE   = 1
IFR    = 00001200 PCC    = 0
IER    = 0000C10B DCC    = 0
IRP    = 00003E54 GIE    = 1
NRP    = 00000000 SAT    = 0
AMR    = 00000000 PWRD   = 00
CSR    = 02020103 FAUCR  = 00000000
CPUID  = 2        FMCR   = 00800080
REVID  = 2        FADCR  = 00000080
```

C6713 registers: debugger view.

- **Others**
  - printf(), LOG_printf…

Stop-mode debugging: intrusive & possibly misleading!

7.5 Real-time techniques

New technology for real-time data exchange host ⇄ target without stopping the DSP.

ADI: Background Telemetry Channel (BTC).

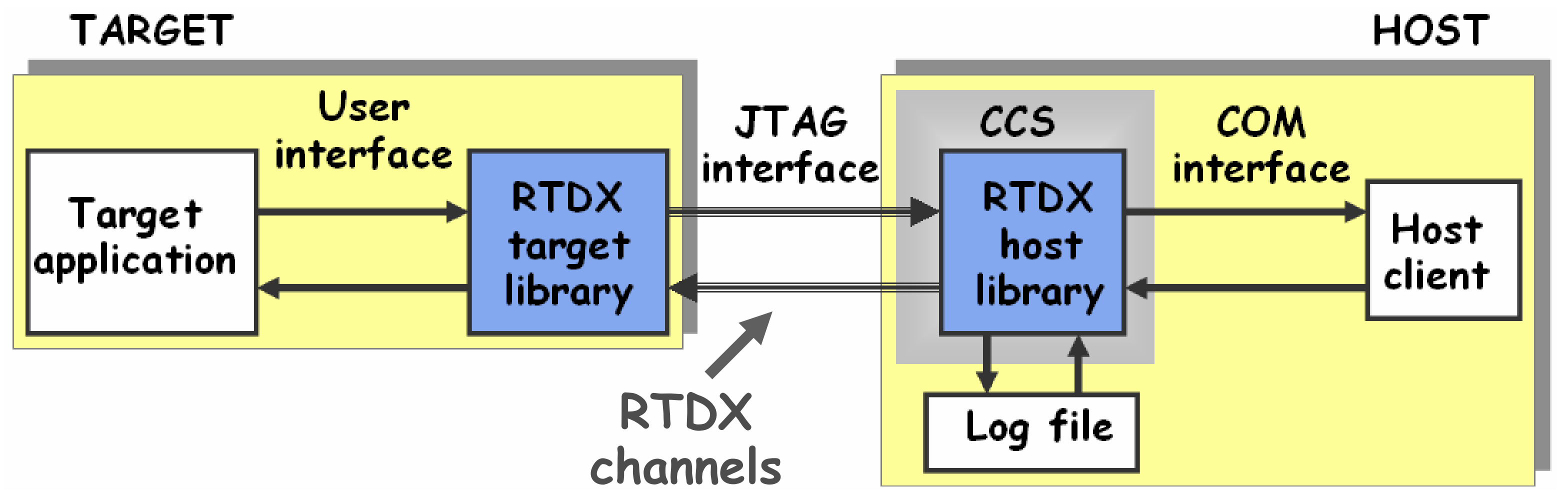
- Shared group of registers accessible (read/write) by DSP & host.
- Supported on Blackfin + ADSP-219x.

TI: Real-Time Data eXchange (RTDX)

- See next slide

- Data retrieved in real time with minimal impact to DSP run.
- Data can be transferred to/from DSP.

7.5 Real-time techniques: TI RTDX



COM intf. clients: VisualBasic, VisualC++, Excel, LabView, Matlab...

Emulation type	Speed
RTDX + XDS510	10-20 kBytes/s
RTDX + USB (ex: DSK board)	10-20 kBytes/s
RTDX + XDS560	≤ 130 kBytes/s
High-speed RTDX + XDS560	> 2 Mbytes/s

TI & RTDX: data transfer speed as function of the emulator.

NB: RTDX can also be simulated

Chapter 7 summary

- Debug phase: most critical & least predictable
- Debug first, switch optimisation ON after!
- Debug steps:
  - **Simulator**
    - No h/w needed
    - Different simulator types available
  - **Emulator**
    - Works with h/w
    - Traditional techniques: stop-mode
    - Real-time techniques: host-DSP data exchange when DSP runs.