Texas Instruments ExpressDSP Algorithm Standard

Prof. Brian L. Evans

Dept. of Electrical and Computer Engineering

The University of Texas at Austin

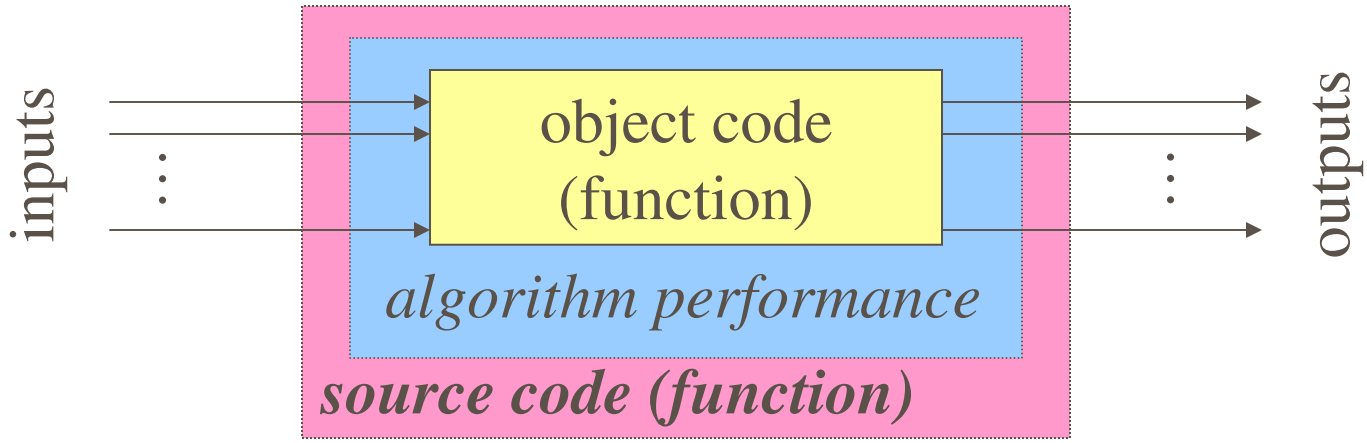
August 7, 2001

Outline

- Introduction
- ExpressDSP requirements and rules
- Source code
- Algorithm performance characterization
- C6000-specific guidelines
- Direct memory access (DMA) resource rules
- Conclusion

Reference: http://www-s.ti.com/sc/techlit/spru352

Introduction: Algorithm Model



- Transformation of inputs into outputs
- Interoperate (play well) with other algorithms
- Decoupled from scheduling and resource allocation (e.g. dynamic memory allocation)

Introduction: Abstraction

- Maurice Wilkes: “There is no problem in computer programming that cannot be solved by an added level of indirection.”
  - Achieve device independence by decoupling algorithms and physical peripherals
  - Achieve algorithm interchangeability by using function pointers

Maurice Wilkes is a hardware designer and the father of microcoding. He is from England and is 88 years old. He is one of the few foreign members of the US National Academy of Engineering.

Introduction: Abstraction

- Jim Gray: “There is no performance problem that cannot be solved by eliminating a level of indirection.”

Jim Gray is software designer who is an expert in transaction processing and databases. He is in his early 50s and works at Microsoft Research in the San Francisco Bay Area. Jim is a member of the US National Academy of Engineering.

ExpressDSP Requirements

- Algorithms from multiple vendors can be integrated into a single system
- Algorithms are framework agnostic
- Algorithms can be deployed in statically scheduled and dynamically scheduled run-time environments
- Algorithms can be distributed in binary form (e.g. Hyperception's Hypersignal approach)
- Algorithm integration does not require compilation but may require reconfiguration and relinking

ExpressDSP Rules

- Source code (*levels 1 and 2*)
  - Programming guidelines: C callable, reentrant, etc.
  - Component model: modules, naming conventions, etc.
- Algorithm performance characterization (*level 2*)
  - Data and program memory
  - Interrupt latency (delay upon invocation)
  - Execution time (throughput)
- DMA resource rules (*level 2*)
- DSP-specific (*level 3*)

Programming Guidelines (*level 1*)

R1 Algorithms must follow run-time conventions imposed by TI's implementation of the C language

- Only externally visible functions must be C callable
- All functions must be careful in use of the stack

R2 Algorithms must be reentrant

- Only modify data on stack or in instance "object"
- Treat global and static variables as read-only data
- Do not employ self-modifying code
- Disable interrupts (disabling all thread scheduling) around sections of code that violate these rules

Reentrant Example

```
/* previous input values */
int iz0 = 0, iz1 = 0;
void PRE_filter(int piInput[],
      int iLength) {
  int i;
  for (i = 0; i < iLength; i++) {
    int tmp = piInput[i] – iz0 +
        (13*iz1 + 16) / 32;
    iz1 = iz0;
    iz0 = piInput[i];
    piInput[i] = tmp;
  }
}
```

Not reentrant

state

```
void PRE_filter(int piInput[],
      int iLength, int piPrevInput[]) {
  int i;
  for (i = 0; i < iLength; i++) {
    int iTmp = piInput[i] –
       piPrevInput[0] +
       (13*piPrevInput[1] + 16) / 32;
    piPrevInput[1] = piPrevInput[0];
    piPrevInput[0] = piInput[i];
    piInput[i] = iTmp;
  }
}
```

Reentrant

Component Model (*level 2*)

R7 All header files must support multiple inclusions within a single source file

R8 All external definitions must be API identifiers or API and vendor prefixed (e.g. H263_UT)

R11 All modules must supply an initialization (e.g. FIR_init) and finalization (e.g. FIR_exit) method

- Init function: initialize global data
- Exit function: run-time debugging assertions for sanity checks (usually empty in production code)

Component Model (*level 2*)

Convention	*Description*	*Example*
Variables and functions	Begin with lower case (after prefix)	LOG_printf()
Constants	All uppercase	G729_FRAMELEN
Data types	Title case after prefix	LOG_Obj or int
Structure fields	Lowercase	buffer
Macros	Same as constants or functions as appropriate	LOG_getbuf()

R10 Naming Conventions

FIR Module

```
typedef struct FIR_Params {
  int iFrameLen;
  int *piCoeff;
} FIR_Params;
const FIR_Params FIR_PARAMS
      = { 64, NULL };
typedef struct FIR_Obj {
  int iHist[16];
  int iFrameLen;
  int *piCoeff;
} FIR_Obj;
void FIR_init(void) { }
void FIR_exit(void) { }
```

```
typedef struct FIR_Obj* FIR_Handle;
FIR_Handle FIR_create(
       FIR_Obj *pFir,
       const FIR_Params *pParams) {
  if (pFir) {
    if (pParams == NULL) {
      pParams = &FIR_PARAMS;
    }
    pFir->iFrameLen =
      pParams->iFrameLen;
    pFir->piCoeff = pParams->piCoeff;
    memset(pFir->iHist, 0,
       sizeof(pFir->iHist));
  }
  return(pFir);
}
void FIR_delete(FIR_Handle fir) { }
```

Algorithm Performance (*level 2*)

- Algorithms declare memory usage in bytes
  - R19 Worst-case heap data memory (inc. alignment)
  - R20 Worst-case stack space data memory
  - R21 Worst-case static data memory
  - R22 Program memory
- Algorithms declare
  - R23 Worst-case interrupt latency
  - R24 Typical period and worst case execution time
- Include this information in the comment header

Example: Worst-case Heap Usage

	DARAM		SARAM		External	
	Size	Align	Size	Align	Size	Align
Scratch	0	0	1920	0	0	0
Persistent	0	0	0	0	1440	0

Example requires 960 16-bit words of single-access on-chip memory, 720 16-bit words of external persistent memory.

Entries in table may be functions of algorithm parameters.

C6000-specific Rules

R25 All C6000 algorithms must be supplied in little endian format.

- Guideline 13 All C6000 algorithms should be supplied in both little and big endian formats

R26 All C6000 algorithms must access all static and global data as far data.

R27 C6000 algorithms must never assume placement in on-chip memory; they must properly operate with program memory operated in cache mode.

C6000 Register Usage

Register	Usage	Type
A0-A9	General purpose	Scratch
A10-A14	General purpose	Preserve
A15	Frame pointer	Preserve
A16-A31 (C64x)	General purpose	Scratch

Preserve: function must restore its value on exit

Register	Usage	Type
B0-B9	General purpose	Scratch
B10-B13	General purpose	Preserve
B14	Data page pointer	Preserve
B15	Stack pointer	Preserve
B16-B31 (C64x)	General purpose	Scratch

DMA Resource Rules

- DMA is used to move large blocks of memory on and off chip
- Algorithms cannot access DMA registers directly
- Algorithms cannot assume a particular physical DMA channel
- Algorithms must access the DMA resources through a handle using the specific asynchronous copy (ACPY) APIs.
- IDMA R1 All data transfer must be completed before return to caller

DMA Resource Rules

IDMA R2 All algorithms using the DMA resource must implement the IDMA interface

IDMA R3 Each of the IDMA methods implemented must be independently relocateable.

IDMA R4 All algorithms must state the maximum number of concurrent DMA transfers for each logical channel

IDMA R5 All algorithms must characterize the average and maximum size of the data transfers per logical channel for each operation.

Top 10 XDAIS Rules

1. All global and static variables must be constants
2. Must use C calling conventions
3. All include (.h) files must have an #ifdef structure to prevent it from being evaluated twice
4. All visible (external) variables, functions, and constants must be prefixed by PACKAGE_ORG, e.g. H263_UT.
5. Never use absolute (hardcoded) addresses (allows data to be relocated and properly aligned)

Top 10 XDAIS Rules

6. Must use IDMA instead of regular DMA (i.e. DAT_copy)
7. Never allocate or deallocate dynamic memory (no use of malloc, calloc, or free)
8. Disable interrupts (disables all thread scheduling) around sections of code that are not reentrant
9. Always check to see if a DSP/BIOS, chipset library, or I/O function is allowed (most are not)
10. Compile in **far** mode, e.g. in Code Composer

Issues for Future Standards

- Version control
- Licensing, encryption, and IP protection
- Installation and verification (digital signatures)
- Documentation and online help