TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

Harnessing the Power of C++ in an Embedded DSP/BIOS™ System

A Case Study

delResearch

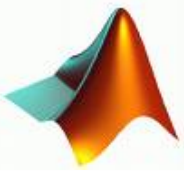
Jim DellaMorte

http://www.delresearch.com

Jim.dellamorte@delresearch.com

delResearch

- Specializing in DSP software solutions
  - Algorithm design
  - Real-time optimized implementation
    - RAM, ROM, MIPS, Power
  - System level DSP software
- Innovation, on time & on budget
  - Medical, consumer, industrial
  - Speech, data, measurement




Minds in Motion

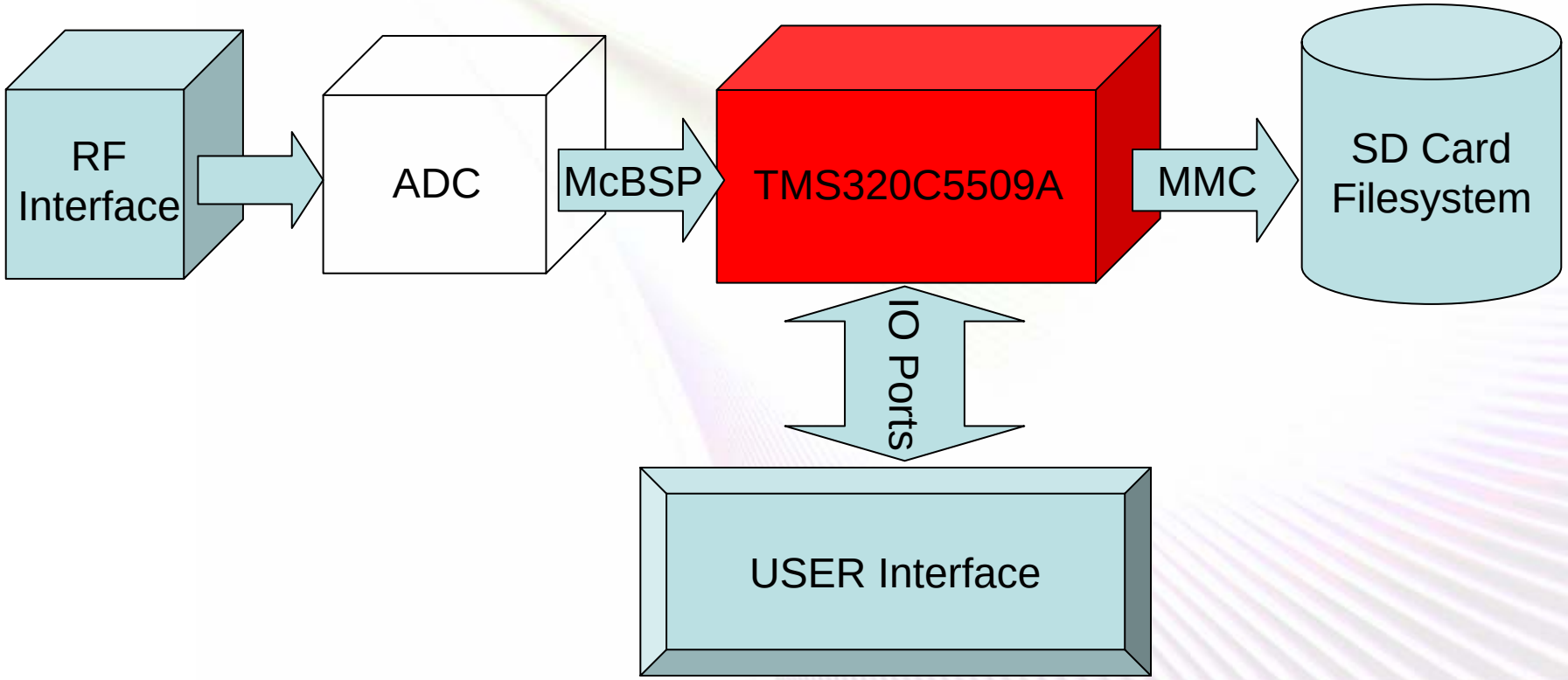
Design Requirements

- C5509A Based Design
- Packet Modem for BPSK Radio Data
  - 1.4 Mhz sampling rate
  - 3-5 mS packets every 10 or so Ms
- Battery Powered
- BIOS System
  - Store demodulated data to flash sard

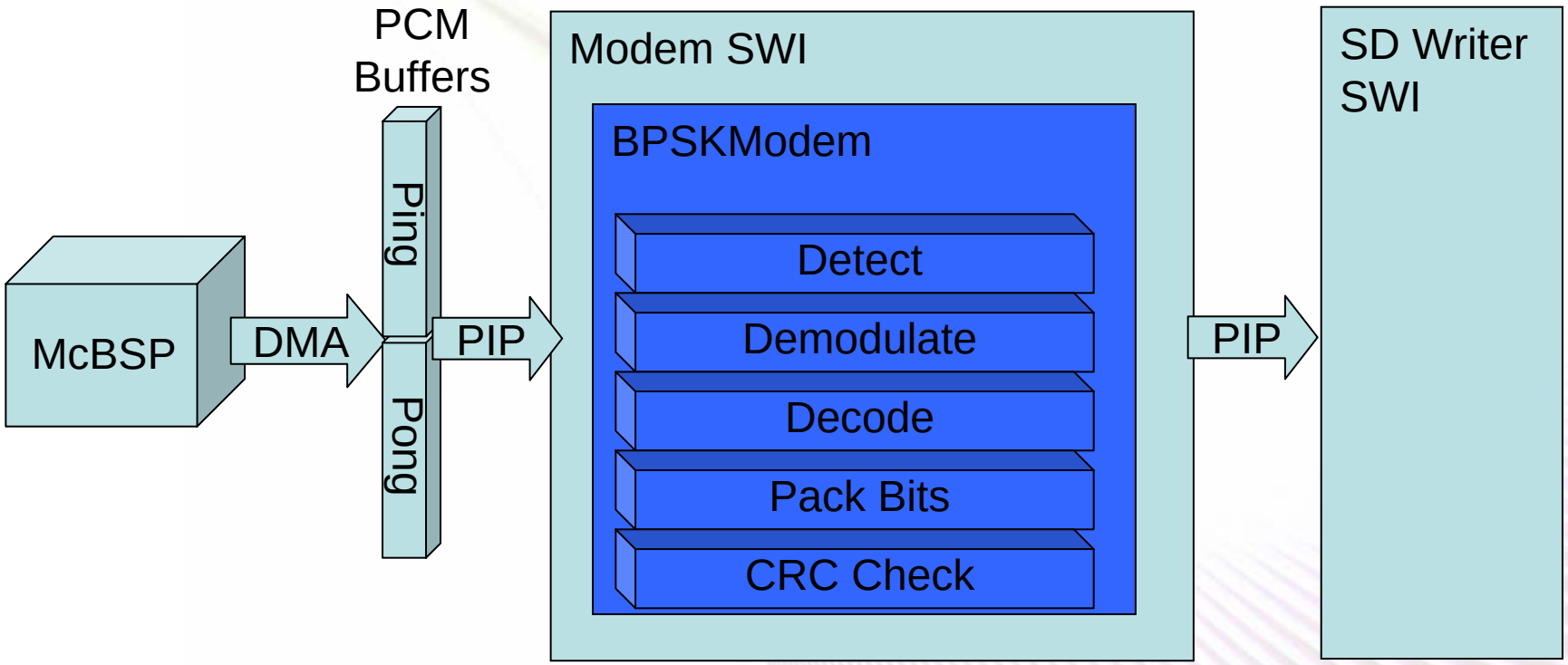
Project Requirements

- Real Project for Paying Customer
  - Deadlines
  - Power budgets
  - Experimentation limited
    - Implement what we knew would work
- Speedy Development
- Easy Maintenance

Block Diagram



Data Flow


PCM
Buffers
Modem SWI
BPSKModem
SD Writer
SWI
McBSP
DMA
Ping
Pong
PIP
Detect
Demodulate
Decode
Pack Bits
CRC Check
PIP

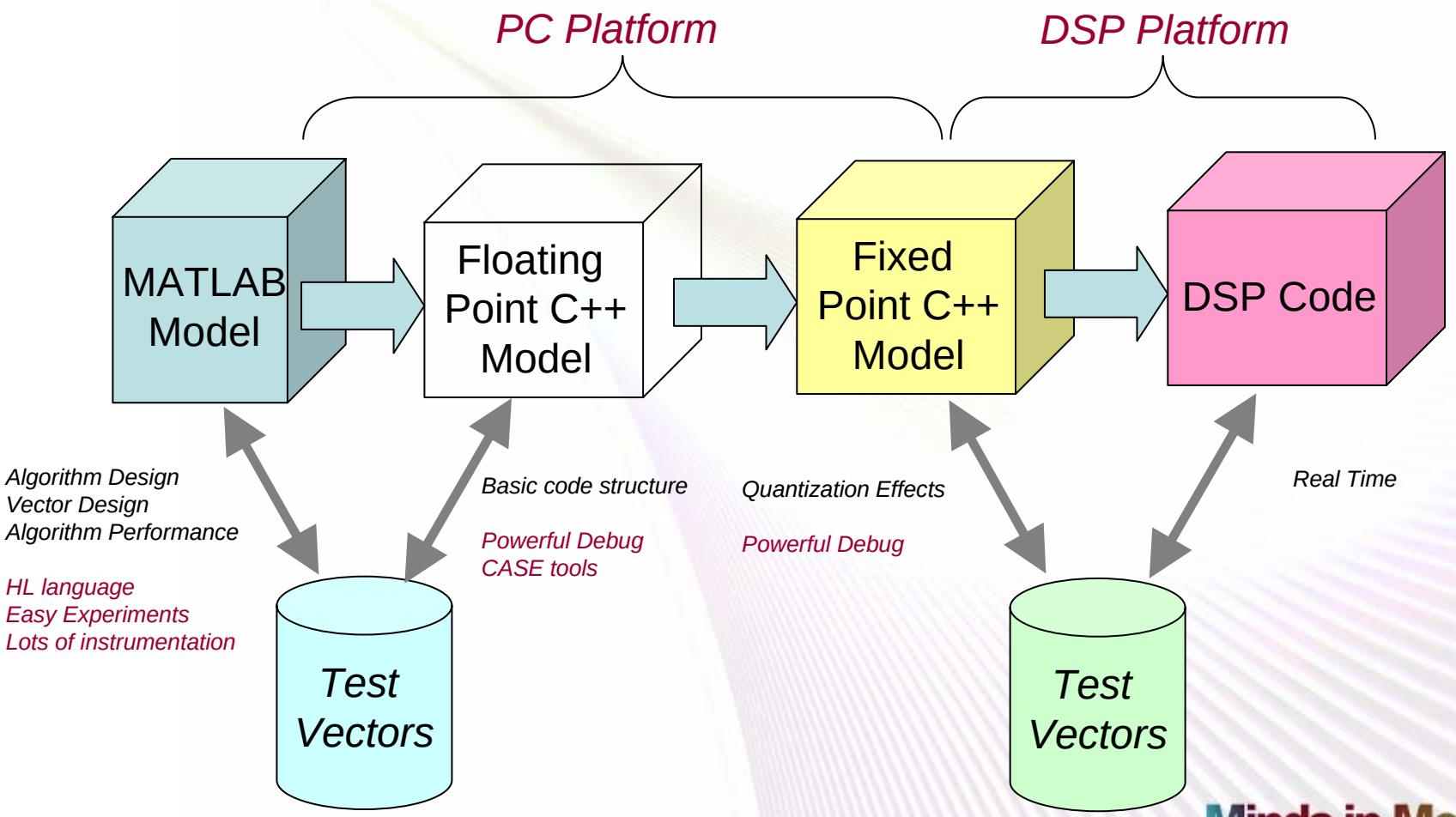
Goals in Development process

- Efficient Code
  - MIPS, power, memory, part count
- Classes for Future Development
  - Object oriented geared for reuse
- Testable
  - Bit match criteria minimizes ambiguity
- Ease of Debugging
- Ease of Long Term Support, Post Fielding
- *On schedule, On Budget*

Why C++

- Object Oriented
  - Inheritance
  - Contained data
  - Fosters code reuse
- Easier To Read Code
- Efficient
  - Development is quick
  - Runtime (when properly done)

Model Based Approach



Minds in Motion

Model Based Benefits

1. Collect anomalous cases from the field
2. Quickly localize the problem in minutes
   - Substitute vectors in DSP code
     - *Tests if it is an implementation or system problem*
   - Substitute vectors in Fixed Model
     - *Tests if it a quantization problem*
   - Substitute vectors in Float Model
     - *Tests if it is a C++ coding problem*
   - Substitute vectors in MATLAB™ Model
     - *Tests if it is an algorithmic problem*
3. Fix the issue in the offending model(s)

Development Flow

- All models incorporate a file to file mode
- Same structure for all models
  - Frame size
  - Data format
  - Control
- DSP code functions the same on Simulator or Platform
  - DSP/BIOS facilitates
  - More debug capability on simulator

Smart, Efficient C++ Practices Applications to DSP

- Model Generation
  - Float/Fixed/Real-time
- Memory Management
- Optimizing
- Interfacing with BIOS

C++ Model

- *Requirements*
  - *Develop algorithm in floating point*
  - *Allow substitution of fixed modules in float code/vice versa*
  - *No overhead for real-time*
- *Approach*
  - *Floating point class*
  - *Fixed point class*
  - *Casting class*

Floating Point Class

- *Class Module*
- *Implements algorithm module with native float data types*
- *Easy to modify*
- *Can run on workstation or native float parts*
- *Insensitive to dynamic range issues*
- *Can contain casted fixed point modules*
  - *Incremental 'Integerization'*

Floating Point Class Example

```
class DiffDemod
{
public:
   DiffDemod(Int32 Fs = DEFAULTFS, Int32 Fc= DEFAULTFC){
        Init(Fs, Fc);
   }
   void Init(Int32 Fs = DEFAULTFS, Int32 Fc= DEFAULTFC){
        CorLag = Fs/Fc;
        for(int i = 0; i < sizeof(CorBuf)/sizeof(Float);
                                        CorBuf[i++] = 0);
   };
   void Process(Float *PcmIn, Float *BaudSignal, Uint16\
                      Framelen);
private:
   Float      CorBuf[MAX_FS_PER_FB];
   Int32      CorLag;
};
```

Fixed Point Class

- class Module_i
- For each float class same functionality implemented in fixed point
- Objective criteria for fixed point performance relative to float
- Fixed point class can stand alone
  - Can be run on fixed point parts without float math libs

Fixed Point Class Example

```
class DiffDemod_i
{
   public:
        DiffDemod_i(Int32 Fs = DEFAULTFS, Int32 Fc= DEFAULTFC){
                Init(Fs, Fc);
        }
        void Init(Int32 Fs = DEFAULTFS, Int32 Fc= DEFAULTFC){
                CorLag = Fs/Fc;
                for(int i = 0; i < sizeof(CorBuf)/sizeof(I16);
                        CorBuf[i++] = 0);
        };
        void Process(I16 *PcmIn, I16 *BaudSignal, Uint16 Framelen);
        void Process(I16 *PcmIn, I32 *BaudSignal, Uint16 Framelen);
        void Process(I16 *PcmIn, I32 *BaudSignal, Uint16 Framelen,\
                        Integrator_i &SignalLevel);
   private:
        I16     CorBuf[MAX_FS_PER_FB];
        Int16   CorLag;         /**< Number of samples per frame */
};
```

Casting Classes

- Module_c
- Casting class allows fixed code to be substituted into float model
- Inherits float, includes fixed
  - Fixed code runs at same time as float
  - Compare results
- Can be substituted by simply changing data type

Casting Class Example

```
class DiffDemod_c : public DiffDemod
{
public:
        DiffDemod_c(Int32 Fs = DEFAULTFS, Int32 Fc= DEFAULTFC)
                                :DiffDemod(Fs, Fc)

        {
                Init(Fs, Fc);
        }
        void Init(Int32 Fs = DEFAULTFS, Int32 Fc= DEFAULTFC)
                                                {
                FixPt.Init( Fs , Fc);
        };
        void Process(Float *PcmIn, Float *BaudSignal, Uint16
                Framelen)
private:
        DiffDemod_i FixPt;
};
```

Memory Management

- *All DSP memories are not created equal*
  - *Dual access Single access*
  - *Wait-states*
- *Faster memory is expensive and always limited*
- *How memory is used determines MIPS and Power*
- *Goals for memory management*
  - *Maximize use of faster memory by overlaying*
  - *Make it simple to move blocks around and try scenarios*
  - *Allocation itself must be quick*

Memory Management (cont.)

- Use static allocation in multiple sections
  - Quick & simple
  - Allows for reuse of temporary buffers
  - Can easily assign to various sections
- TMS320 C++ has hooks to facilitate memory allocation
  - Override `new` statement cleanly
  - Override `delete` statement cleanly

Static Allocation

- Memory Management Method employed in this project
  - Great for single channel fixed function
  - No Memory Manager Overhead
- Class declared on BSS
  - Requires default create parameters
  - Split up Constructor and Init function
    - Separate memory allocation from initialization

`new` and `delete` overriding

- Many possibilities
  - Data type driven `new/delete`
    - Aligned memory
    - Temp memory
    - Simple fixed malloc()
  - Instrumentation included
  - Ease of BIOS integration with `memalloc()`

Optimizing

- Process Overview
- Function Overriding
- Automatic Inlining
- Interfacing Outside of C++
  - Interfacing with C Code
  - Interfacing with Assembly Language
  - Interfacing with DSP/BIOS

Quantization Process

- Begin with the Float C++ model
- Start at innermost float function
  - Create a casting function
  - Test the quantization impact alone
  - Write the equivalent fixed point subroutine utilizing prototype matching
  - Verify performance with newly incorporated fixed point subroutine
- Continue until the entire algorithm consists of casted fixed point subroutines
- Write the main loop entirely calling fixed point routines without casting functions

Function Overriding

- C++ Function calls are data type driven
  - Allows programmer to choose precision needed without cluttering code
  - Spend cycles where needed, keep code clean
- Demod Process Call
  - Called Before AGC in Detector (32 bit result)
  - Called After AGC during demodulation (16 Bit Result)

void Process(I16 *PcmIn, I16 *BaudSignal, Uint16 Framelen);
void Process(I16 *PcmIn, I32 *BaudSignal, Uint16 Framelen);

Automatic Inlining

- C++ has nice method of Inlining code
  - Write procedures with Class definition in header file
  - Easy to maintain code
- Eliminate function call overhead
- Used for heavily used small code blocks
  - IIR Averaging filters
  - Demodulation blocks

Combining Inlining /Overriding

- Sometimes optimization requires mixing 2 functions together
  - Example Filter, and calculate energy
    - Save stores if energy calc done on the fly
  - Usually creates ugly hard to maintain code
- Create new process Call
  - void Process(I16 *PcmIn, I32 *BaudSignal, Uint16 Framelen, Integrator_i &SignalLevel)
  - Passed Class for SignalLevel becomes inlined into process call
    - 2 Classes remain distinct

Interfacing Outside of C++

- Often need to work outside of C++
  - Mixed environments dictated by
    - BIOS
    - Need for hand coded assembly
    - Requirement to interface with other C libraries
- C Calling C++ (BIOS for example)
  - Declare extern “c” function within C++ code
  - Entry point to C++ Code
- C++ Calling C or Asm
  - Extern “C” reference to external functions

Results

- *Little to no overhead*
  - *Simplified Optimization process*
  - *Could argue < MIPS than straight C*
- *Self documenting*
- *Forces good practices*
- *Reusable*

TI Developer Conference
March 7-9, 2007 • Dallas, TX

Harnessing the Power of C++ in an Embedded DSP/BIOS System

A Case Study

Jim DellaMorte
http://www.delresearch.com
Jim.dellamorte@delresearch.com

Minds in Motion

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265