RoweBots
Research Inc.

White Paper – 3 August 2009

Ultra Tiny Linux meets

DSP Pipeline Challenge

Author: Kim Rowe,
Founder, RoweBots Research Inc.

DSP Applications on DSCs and MCUs

Building DSP applications on digital signal controllers (DSCs) and microcontrollers (MCUs) using the de facto applications programming interfaces (APIs) for our industry – embedded Linux and POSIX is something new. However, to maximize profits and minimize risks, this is the approach that is required. This article illustrates how to implement this approach.

Overall Software Best Practices

The overall approach of selecting an ultra tiny embedded Linux and POSIX compatible operating system to be the core software infrastructure component is to maximize profits and minimize both time and risk. This is optimization occurs because a standardized platform based on POSIX and Linux provides knowledge and experience for:

- fostering team communications,
- reusing tens of thousands of applications,
- purchasing reusable components,
- tapping into open source,
- ease staffing problems,
- getting access to knowledgeable outside consultants,
- and to get access to books and training.

This is the best choice today to reduce the overall risk and time to market of DSC and MCU based systems. This approach also fully supports component based models and lean product development.

Overall Software Best Practices

continued

When using embedded Linux and POSIX APIs, what is the best approach to build DSP systems which need significant processing done in software? The generally accepted way to do this is to use a DSP pipeline which eliminates all unnecessary buffer copying when implemented correctly.

DSP Pipeline Best Practices

The core approach to building a DSP pipeline in software rests on the following approach.

- Build a zero copy DSP pipeline.
- Allow threads to implement elements of the pipeline as components.
- Use message queues as inputs and outputs and pass only pointers to buffers.
- Use fixed size buffer managers which can be used by any thread to speed processing.
- Implement standard algorithms which receive an initialization message, get blocks of data from the input queue, process in blocks and post the results to the output queue, managing buffers as required
- Allow all threads to run at a priority which works for the system, typically with the I/O related threads as the highest priority.

From this, it is easy to see that the core building blocks for optimal standards based DSP component implementation are:

- threads
- initialization messages
- queues for input messages
- queues for output messages
- fixed size buffer management

Figure 1 shows a standard component with its inputs, outputs and buffer management.

The details of the POSIX and Linux compatible features required to provide optimal system level implementation are as follows. First, message queues with a queue of pointers to buffers is ideal, then ownership of the pointer to the buffer gives ownership of the buffer and it can be easily passed if required.

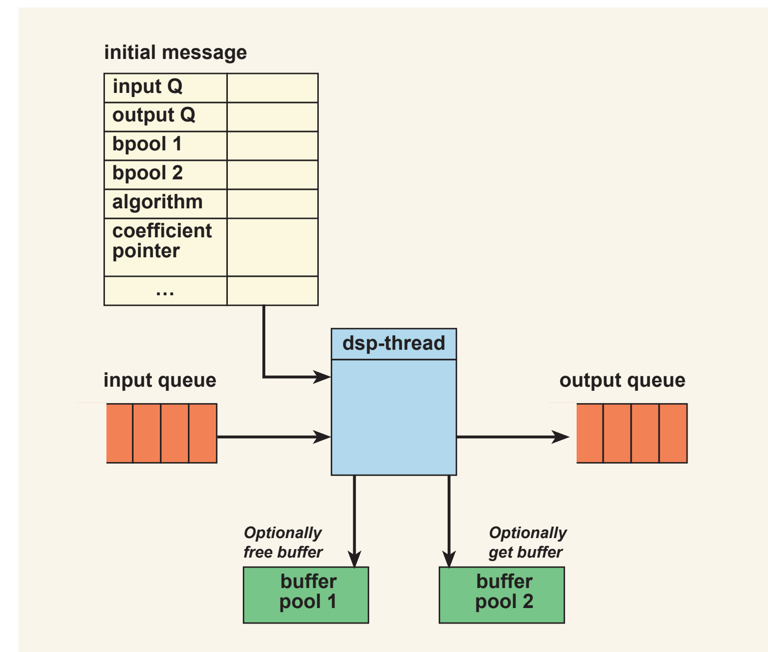

Figure 1: The simple structure of a DSP pipeline component

DSP Pipeline Best Practices

continued

Second, each thread should have an initial message which configures the processing. For example, the initial message will specify the input and output queues, signal processing parameters like coefficients, algorithm details or options, and necessary buffer pools.

Third, fixed size buffer managers with getting a buffer and freeing a buffer callable from interrupt service routines (ISRs) are key. This is required to allow the components to manage buffers without overhead. Availability to manage buffers in ISRs is required to eliminate any buffer copying.[1]

Priority based thread scheduling is used to make sure the critical components in the pipeline are processed in the correct order to maximize throughput. Often buffer shortages occur and system results should not be negatively affected by temporary data surges.

Message queue post from ISR and message queue try_receive from ISR are also required to ensure that the ISR can get a buffer, fill it and then pass the data on for processing without copying. Similarly an ISR can get an output request from the queue without blocking, process it and then return the empty buffer to the buffer manager or to another queue.

1.Some vendors implement a proprietary object or structure to communicate from the ISR to the thread. Invariably this results in an extra buffer copy when used.

Standardized DSP Component Psuedo Code

The pseudo code for the implementation of a standardized component is shown below. Note that this same structure applies to a broad set of DSP algorithms from FIR and IIR filters and FFTs to more exotic processing like zoom FFTs and feature extraction.

```
THREAD DSP_thread (void *mptr)
{
setup algorithm and start buffers using initial message
loop forever
  {   get data from input queue
      process data to output buffer
      send to output queue
      setup buffers for next iteration using initial message buffer info
  }
free all buffers
}
pseudo code for data input ISR and data output ISR
DSPnano_ISR  input_data( void *arg)
{
process arg if required
get buffer if required
process data into buffer
send to output queue
}
DSPnano_ISR  output_data( void *arg)
{
process arg if required
process data from buffer to output
send to output queue
return buffer
```


Rowebots Research Inc. © 2009,

DSP Pipeline Summary

continued

All block oriented signal processing algorithms can be implemented using this type of pipeline and all calls with the exception of the fixed size buffer management calls are fully POSIX compatible. Fixed size buffer management can be implemented using malloc and free to provide initial pools or used directly for buffer management for 100% compatible options.

All components are reusable using this approach – you can build up your library and use it for all projects. All parameters are passed in via an initial message, and reuse should only require knowledge of how to setup this message. All ISRs handle buffers directly so buffers are never copied unnecessarily. Overall the DSP pipeline offers an optimal solution to this software DSP challenge.

Practical DSP Pipeline Implementations

Unison (32 bit) and DSPnano (8/16 bit) from RoweBots offer both optimization for DSP and 100% POSIX and Linux compatibility. They run on a variety of DSCs and MCUs and have 20 DSP related features to make building pipelines fast and easy. The architecture is focused on using reusable components to maximize reuse and minimize development.

Out of the box, both DSPnano and Unison support all the features required to build a DSP pipeline quickly and easily.

With full support to eliminate buffer copying even at I/O boundaries, hardware acceleration can be added using external devices and data can be input back into the remainder of the pipe to complete processing very efficiently. Examples of this type of acceleration include dedicated ASICs and FPGAs. In some cases, the pipeline might even be in the FPGA's processor with hardware acceleration done on external components.

Other DSC and MCU implementations are not open and generally based on proprietary technologies focused on customer lock in. Choosing a standards based approach allows you the maximum in application portability and flexibility.

Summary

RoweBots two ultra tiny embedded Linux implementations, DSPnano (8/16 bit) and Unison (32 bit) are ideally suited to building reusable software components for DSP applications. Standardized components can be easily ported across a variety of MCUs including PIC24, dsPIC30, dsPIC33, PIC32, R8C, M16C, R32C, SH-2, SH-2A, H8S, H8SX, and ARM Cortex Mx. Porting to other Linux and POSIX based operating systems is fast and easy, and the key reason that DSPnano and Unison support 100% POSIX and Linux compatibility.

Rowebots Research Inc. © 2009,

White Paper – 3 August 2009

Ultra Tiny Linux meets

DSP Pipeline Challenge

Author: Kim Rowe,
Founder, RoweBots Research Inc.

Contact Information:

Kim Rowe, Founder

sales@rowebots.com

+1 519 208 0189

+1 519 498 6917

RoweBots
Research Inc.

Rowebots Research Inc. © 2009,