Teuthida Technologies Home

All about embedded systems engineering.

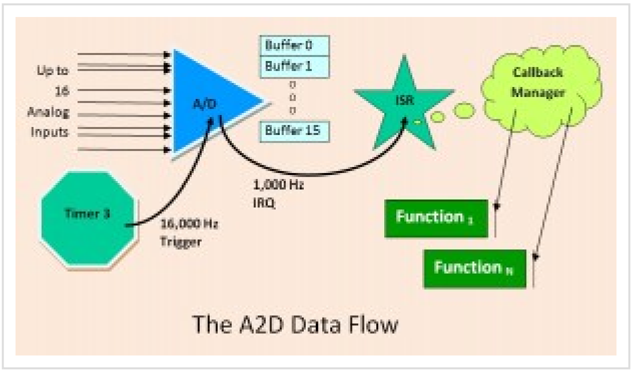
Modules: The A2D Manager

Posted on **September 1, 2012**

The examination of modular programming so far has included a look at module coupling, module event propagation, and the CallBack Manager mechanism. This posting delves into the operation of an actual data source module, the A2D Manager. First a little background:

In this space, a number of articles have dealt with getting the most from an embedded, resistive touch screen. In all of the articles that have examined the touch screen, the topic that has been omitted until now is “how are the touch screen voltages actually measured?” The answer lies with the PIC-32s capable Analog to Digital Converter (A2D or A/D Converter). A glance at the A2D reference manual gives some idea of the power of this peripheral. A common complication is the fact that the A2D is often required to operate multiple analog devices. In this case those devices are “unspecified” analog sensors.

A typical solution to the requirement to share a device is to use a mutual exclusion (or MUTEX) semaphore to enforce serial reuse of the resource. The only problem with this approach is that it has poor timing consistency. The rate of analog measurements would be subject to timing variation due the order of task execution. Instead the approach used is to place control of the A2D in the hands of an A2D manager. The A2D Manager handles the reading of analog data and serves up this data to client tasks. The goal of the A2D manager was to read all the analog data possible at a constant, predictable rate, make that data available to tasks that need it, and to use as few resources as possible.



The diagram to the left summarizes the operation of the A2D Manager. After initialization in the emInitA2DManager function, the A2D manager does not use a task. It does use a programmable timer and an interrupt service routine (ISR). All 16 analog input channels are scanned once per millisecond, under the control of the timer. At the end of the millisecond, the ISR is triggered by the A2D and copies the data from the A2D to a buffer with the following definition:

```
typedef struct
{
    uint16_t seq;       // A counter. 1 count per cycle.
    uint16_t analog;    // 0 == analog pin, 1 == digital pin.
    uint16_t data[16]; // A2D data.
} emA2DRESULTS;
```

To share data with client tasks, the ISR then calls a callback list to inform interested parties that data is available. The callback list itself is one of those rare instances of global data that is required. This is declared as:

```
extern emPCALLBACKLIST emA2DIsrCallBackList;
```

The "Isr" in the name reminds coders of client code that this callback occurs in the context of an interrupt service routine, so brevity is a requirement. The A2D Manager initializes the list with:

```
emA2DIsrCallBackList =
  emCreateCallBackList(A2D_CALLBACK_LIST_ENTRIES);
```

Callback lists are managed by the Callback Manager; previously discussed. An example of a client adding a callback to the list is:

```
emAddCallBack(emA2DIsrCallBackList, &touchCB, emFront);
```

In which the function touchCB is added to the emA2DIsrCallBackList in the first free slot. When the A2D Manager is ready to signal an event, the code for this is simply:

```
// Execute the A2D callback list.
emExecuteCallBackPV(emA2DIsrCallBackList,
                    &xHigherPriorityTaskWoken);
```

In which the first parameter is the callback list and the second is a pointer to a flag used by the FreeRTOS to indicate that a higher priority task is now ready to run and that a task switch is needed when the interrupt service routine concludes.

It goes without saying that the A2D manager is a compromise. It does however meet its goals of providing a constant 1 K Hz data rate; It does read all 16 possible channels; and it does not tie up a task or use a lot of resources. What resources are used then?

1. Timer 3 is used to generate Start Conversion pulses at a rate of 16,000 Hz. This consumes no CPU bandwidth.
2. The A2D module is used to read all 16 possible channels and buffer them. This consumes no CPU bandwidth.
3. An interrupt is triggered every 16th sample, or 1,000 Hz. At this point, the A2D buffers are full of data that are copied to a buffer.
4. A Callback list is used to update clients of the A2D manager.
5. 36 bytes of RAM for the shared data structure and the the callback list uses an additional 4+8*A2D_CALLBACK_LIST_ENTRIES bytes.

In review, the Callback Manager is a good mechanism for supporting the case where one data/event source needs to reach multiple data/event sinks. In the examination of the Message Queue or Message Pump the case of multiple data/event sources and a single data/event sink will be discussed.

As always, your comments and thoughts are encouraged and welcomed.

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Embedded Development** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=1019]** .

1 THOUGHT ON “MODULES: THE A2D MANAGER”

Arduino lover
on **September 13, 2012 at 10:37 am** said:

If you like everything related to micro-controllers visit my blog:

All about Embedded Systems

Comments are closed.