PicOS

High rate ADC sampler

Version 2.01
February 2007

Preamble

This note describes the praxis interface to the ADC sampler, whose purpose is to provide a means for convenient acquisition of analog samples at regular intervals. This interface conflicts with the slow (polled) interface to ADC pins described in the *PinOps* document. This means that when the ADC sampler is configured into the system and running, the ADC converter is not available for any other purpose. However, the ADC sampler can be stopped and then restarted. While the sampler is stopped, the ADC converted can be reset and assigned to other functions. When the sampler is resumed, it will reclaim the ADC converter.

Board setup

The exact parameters of the ADC sampler are defined on a per-board basis. To enable the sampler, set **`ADC_SAMPLER`** to 1 in *board_options.sys*. Then the system will expect a file named *board_adc_sampler.h* in the board directory. That file will describe the configuration of ADC inputs to the sampler as well as the sampling rate. For illustration, look into *PicOS/MSP430/BOARDS/SFU_HEART*.

Up to 8 analog channels can be sampled simultaneously (or rather in a Round-Robin fashion). Let this number be *N*. One full sample consists of *N* values collected from the respective channels in one cycle. The individual per-channel samples will be called *subsamples* – to avoid confusion.

For fine tuning of the sampling rate (and generally for high sampling rates), a high-speed (second) crystal is needed (e.g., 1-6MHz). The setup involves Timer A strobing the subsamples. When the last subsample in a sample is extracted, the sample (consisting of *N* subsamples) is stored in a circular buffer for retrieval by the praxis.

Consider the example in *SFU_HEART*. Suppose that the sample size *N* is 8, and the sampling rate is 500 samples per second. This means that Timer A will have to go off at the rate of 8 x 500 = 4000 times per second to strobe the conversions of individual subsamples. Thus, the setting of the respective TA register is *clock rate* / 4000, which must be an integer number. Note that the maximum clock rate is directly determined by the crystal rate.

Note that whatever is to perceived as the actual sample by the praxis can also be described in *board_adc_sampler.h*. The file defines a macro, **`adcs_sample(b)`**, which assumes that **`b`** is an array of **`word`**-sized items and is supposed to load the array with the sample. The constant **`ADCS_SAMPLE_LENGTH`** determines the size of one full sample in **`word`**s.

Praxis-level operations

Once the ADC sampler has been configured into the system, the praxis has access to the following functions:

```
word adcs_start (word bufsize)
```

The operation starts the sampler (grabbing the ADC converter for its exclusive use). The argument specifies the size of the circular buffer (in samples) that will be malloc'ed and used by the sampler while it is active. The role of the buffer is to compensate for the jitter of sample extraction by the praxis. Following **`adcs_start`**, the buffer gets filled with samples at the prescribed sampling rate.

Note that the actual size of the memory area for the circular buffer is equal to **ADCS_SAMPLE_SIZE * (bufsize + 1) word**s, i.e., twice that many bytes. The function returns **ERROR** (nonzero) if the buffer could not be allocated (because of memory shortage), and zero otherwise. If the sampler is already started, the function triggers a system error.

```
void adcs_stop ()
```

The function stops the sampler and deallocates the circular buffer. The ADC converter can be used for other purposes. If the sampler is not active when the function is called, it has no effect.

```
Boolean adcs_get_sample (word state, word *sbuf)
```

The function attempts to retrieve a sample from the circular buffer and store it into **sbuf**. If a sample is available, the function returns **YES**. Otherwise, its behavior depends on **state**. If **state** is **WNONE**, the function immediately returns with **NO**. Otherwise, the current process is blocked until at least one sample gets deposited into the buffer, in which case it will be resumed in the indicated **state**.

```
word adcs_overflow ()
```

The function returns the number of overflow conditions on the circular buffer since its last call. An overflow condition occurs whenever the sampler cannot deposit a new sample because the buffer is full, i.e., the praxis cannot catch up with the sampler. Thus, this is simply the number of lost samples. After each call to **adcs_overflow**, the overflow counter is reset to zero.