HOW TO USE: External Analog to Digital Converter

Files: adc_external.c/h
Variable Prefix: adc_external_
Macro Prefix: ADCX_

Overview:
The external analog to digital converter module was developed and tested on the LPC2194/01 microprocessor by Nicolas Champagne-Williamson on Ranger's motor board in April and May 2009.

Features:
- Supports 8 single channels at 13 bits of resolution or 4 differential channels or 14 bits of resolution of conversion
- Exponential averaging filter
- Schedule and external interrupt driven
- Uses SPI over SSP

Setup:
- Within the adc_external.h file...
    + copy the 'Hardware Setup' section into the 'setup_hardware()' function of the 'hardware_setup.c' file.
        - This will setup the SPI over SSP, the pin selection and directions, and will send setup data over the spi to the ADS7871 registers and empties the Tx and Rx FIFO buffers
    + copy the interrupt setup section into the interrupt section of the hardware_setup.c 'setup_hardware()' function
        - The default is for the External Interrupt to use IRQ slot 0. Change the slot according to priority on your specific board
    + copy the 'Software Setup' portion into the 'software_setup.h' file.
        - Set these values according to your code
        - *_GAIN: the gain for the output of each channel's filter; float value
        - *_OFFSET: the offset for the output of each channel's filter; float value
        - *_FILTER: the filter coefficient for the exponential averaging filter (see: **Filter**)
            + Should be a value between 0 (no filter) and 14, inclusive. The filter is only employed on data once a non-zero value has been added. This limits the effect of beginning zeros on later data, especially with larger coefficients.
        - **struct Filter{...}:** you only need one of these defined in the software_setup.h file, so check to make sure it is not doubly defined if some other module also uses

this struct.

Conversions:
- To convert a channel, call the function **adc_external_convert(short* schedule)**
- The function accepts an array of channels and gains to be converted, NULL terminated. The order in the array is CH, GAIN, CH, GAIN, CH, GAIN, NULL.
    + for convenience, channel and gain macros have been provided and are located in 'adc_external.h'
- Conversions will occur asynchronously from the main code scheduler until all conversions have completed. Starting another conversion schedule before the last has completed is an error and should be avoided before runtime.
- The results of conversions can be accessed using 'adc_external_get_int(short channel)' (for the raw filter int) or 'adc_external_get_float(short channel)' (for the filtered value * gain + offset)

Filter:
- The exponential averaging filter uses the **filter struct** to store the values of each channel. This module uses four filters - one for each channel. The struct stores the current value of the filter, the data count, and the coefficient. The filter works with values bit-shifted left by 14 for maximum averaging resolution. If the data fluctuates between 0 and 1, we want the average to be 0.5. When a new value is added to the filter, it is first shifted left 14, and 1/2^coefficient of this new value is added to (1 - 1/2^coefficient) of the old data. The coefficient has to be between 0 and 14, inclusive. A coefficient of 0 means no filtering; the filter simply stores the latest value.

Examples: (NOTE: this will shortly be changed)
Method 1: Using a scheduler to access results at a later time...

```
short schedule[] = {    ADCX_CH1,          ADCX_GAIN1,
                        ADCX_CH2,          ADCX_GAIN1,
                        ADCX_CH5,          ADCX_GAIN4,
                        ADCX_DIFF_P0N1,    ADCX_GAIN1,
                        NULL};
adc_external_convert_all(schedule);
//at a later time you can access the results...
int result = adc_external_get_int(ADCX_CH1);
```

Method 2: Waiting until conversion have completed...

```
short schedule[] = {    ADCX_CH1,          ADCX_GAIN1,
                        ADCX_CH2,          ADCX_GAIN1,
                        ADCX_CH5,          ADCX_GAIN4,
                        ADCX_DIFF_P0N1,    ADCX_GAIN1,
                        NULL};
adc_external_convert_all(schedule);
//Wait for all of the conversions to finish...
while (!adc_external_done_converting()){}
int result = adc_external_get_int(ADCX_CH1);
```

For Development

Global Variables:

struct Filter **adcx_filters[16]:** This variable will hold the filter structs for each of the 8 single channels and the 4 differential channels (*2 because each differential channel can have its polarity switched).

float **adcx_gains[16]:** This array will hold the gains for each of the 16 channels.

float **adcx_offsets[16]:** This array will hold the offsets for each of the 16 channels.

short **adcx_index:** This index variable will know where in the conversion schedule we are. The index will always be 2*conversion#, because for each conversion there is also a gain associated with it, so the index needs to be increased by 2 to access the next conversion.

short* **adcx_convert_schedule:** This null terminated array will contain the conversions to be completed when a user calls adc_external_convert_all(schedule). There are macros in the header file for the channel and gain values.

int **adcx_status:** This variable will be set to any one of the flags described in the header file: ADCX_DONE, ADCX_NOT_DONE, or ADCX_FINISHING. The status will be ADCX_DONE when no conversion is occurring. The status will be ADCX_NOT_DONE when the module is in the middle of the conversion, and it will be ADCX_FINISHING when the module has reached the last conversion and needs to send the last dummy conversion to read in the final conversion result.

Public Functions:

void **adc_external_write**(unsigned char **data**): puts data into the transmit buffer to be sent to the external ADC. If the transmit buffer is full, sends an error.

void **adc_eternal_register_write**(unsigned char **reg**, unsigned char **w_data**): writes w_data to the register. On the external ADC chip we used, the TI-ADS7871, to start a register write, we first send the register address (0's mean register mode, writing, 8 bit data) by adding it to the transmit buffer, and then we send the data by adding it too to the register buffer, waiting in between if the buffer is full. This function will only be called at hardware setup.

unsigned char **adc_external_register_read**(int **reg**): reads in the value of the given register on the external ADC chip. It first clears the receive buffer, then sends the read register command and a dummy byte. Because the SPI talking to the external ADC is duplex, the command byte will read in junk, and the sent dummy byte will read in the value of the requested register. We return this second value.

unsigned char **adc_external_read_buffer(**void): This function checks if the receive buffer is not empty, returning the value in the buffer if it is (not empty), and sending an error and returning 0 if it is empty.

void **adc_external_convert_all**(short* schedule): Converts all the channels specified in schedule. Begins by checking if the previous conversion finished; if it didn't send an error. Set the status of the module to ADCX_NOT_DONE, and begin the current batch of conversions. It starts by sending the first convert command. An external interrupt from the ADC will tell it when the conversion is complete and will call the AtoD() Interrupt Service Routine (ISR).

void **AtoD(**void) __irq: This ISR will be called every time a conversion finished and triggers and external interrupt. It will continue the conversions until we have completed all of them. We clear out the buffers of the junk data that was read in during the command writing (duplex), then write the command to begin the next conversion. This continues until the last conversion. When the interrupt signals that this final conversion is complete, we send a dummy conversion so that we know we've given it enough time to read back the final results into our receive buffer. We filter all read in files according to filter values specified in the software setup. At a later time, most likely specified by the scheduler, these filtered values can be read in and used by other programs and modules. For more understanding, look at the timing diagram below:

int **adc_external_get_int**(short **channel):** Returns the int value that is stored within that channel's filter. Within the filter, this value is stored in the 'average' field. Note that because the filter shifts values left by 14, this value will be much much larger than the raw data.

struct Filter* **adc_external_get_filter**(short **channel**): Returns the filter associated with the given channel. This will allow a user to update filter values and coefficients, and they can access all fields, such as average or count, directly.

float **adc_external_get_float**(short **channel**): This returns the scaled value from the filter for the given channel. The way this is calculated is *average* * *gain* + *offset*. All of these values have been set in the software setup file.