User Operation

SD Card Preparation

The digital recorder only supports the FAT16 filesystem. The SD card must be completely defragmented or freshly formatted. To format the card with the FAT16 filesystem under Windows:

- Right-click on the SD card drive.
- Select "Format"
- Under "File system," select "FAT" or "FAT16."

Configuration and Scheduling

A configuration file placed on the SD card and named "CONFIG.TXT" is used to configure the sampling rate, recording interval, and sleep interval. Options can be in any order, one per line. The configuration file ends with the string "end." A sample configuration file is listed below:

```
interval 1800
duration 30
rate 44100
end
```

This configuration file instructs the digital recorder to record 30 seconds of data with 1800 seconds (30 minutes) between cycles. Note that the 30 seconds is included in the total interval, so a complete cycle occurs in 1830 seconds. The sampling rate is set to 44100 kHz. Standard windows sampling rates (e.g., 22050 and 44100) are fully supported. Other rates may result in buffer overruns (sampling rate too high) or invalid WAV files (nonstandard sampling rates).

After the configuration file is placed on the freshly-formatted SD card, place the SD card in the recorder. Press the "Configure" button to instruct the recorder to read the configuration file and start a recording cycle. The green LED will flash 5 times rapidly to indicate the configuration file has been read successfully. A solid red LED indicates a card error. If ejecting and reseating the card yields the same results, the card may be unsupported or damaged.

Recorded data will be stored on the card with sequential filenames starting with "DAT00001.WAV," incremented for each new recording.

One-Shot Recording

Press the "One Shot" button to immediately record a 10-second sequence of data. The data will be stored on the card with the next available sequential filename. This feature can be used to perform quick checks of the volume and microphone.

Collecting the Data

The "Eject" button **must** be pressed prior to removing the SD card. This allows the digital recorder to finish any pending card operations before ejecting the card. Failure to press the eject button may result in damage to the digital recorder or the SD card.

After pressing eject, the green LED will glow continuously when it is safe to remove the card. On card removal, the green LED will turn off.

Software

Functional Overview

The digital recorder's firmware is compiled with the IAR workbench, but with the appropriate MSP430-compatible compiler and the necessary header files, any compiler and JTAG programmer is able to update the device's firmware. All firmware is written in c. Source files are divided into major functions:

- adc_fcns.c - Implements the ADC sampling and interrupt routine
- clock.c - Wrappers for software-based clock control
- config.c - User configuration and configuration file handling
- fat.c - Implements a minimal, fast FAT16 interface
- main.c - Implements the main software control loop
- power.c - Implements various power control functions
- pushbuttons.c - Handles external interrupts triggered by the pushbuttons
- sd.c - Implements the low-level SD Card interface
- sleep.c - Functions related to low-power mode and wakeup
- spi.c - SPI bus helper functions for the SD Card interface
- wav.c - Constructs a valid WAV header

State Machine

The main control loop consists of a software-based finite state machine in the file `main.c`. The following states are defined:

- S_IDLE - Idle, waiting for card insertion and configuration
- S_INIT - SD Card initialization (entered on user request)
- S_READ_CONFIG - Reading and parsing the configuration file
- S_ERROR - An error was encountered during initialization
- S_ACQUIRE - Acquiring new data
- S_SLEEP - Sleeping between acquisitions
- S_POSTSLEEP - Cleanup and wakeup following a sleep
- S_PERM_ERROR - An error was encountered during acquisition

Refer to the source code for more detailed descriptions of the tasks performed in each state.

Wakeup and Interrupts

The state machine may be interrupted at any time by an external interrupt (i.e., a user presses a button) or by an internal interrupt (timer or ADC acquisition). Interrupts are flagged via global variables, and then acknowledged and at predefined points in the state machine when it is safe to perform the requested action.

Since the processor can only respond to interrupts when in its sleep state (i.e., normal program flow is halted), a race condition may be triggered if the processor is in its normal operating mode and the following sequence occurs:

1. Test for a pending pushbutton interrupt.
2. Pushbutton interrupt triggered. Condition flagged.
3. Sleep state entered.

At this point, the sleep state can no longer be interrupted by the pushbutton interrupt and the button will not be acknowledged until the scheduled wakeup time.

To eliminate this condition, the global interrupt flag must be disabled before testing the condition of the pushbutton interrupt, if that test will result in the processor entering sleep mode. The processor's sleep mode will re-enable global interrupts in one atomic operation.

Normal program flow is to resume execution after the sleep interval has elapsed.

Clock Management

The MSP430 uses a multiple clock architecture. Different clock signals may source different functional units. The source is software-selectable. The digital recorder uses two main clocks, sourced from an external 8MHz crystal and an external 32.768 kHz crystal. The 8MHz clock is used for normal program execution and sample interval timing during wakeup/acquisition modes. During sleep modes, the 8MHz clock is completely disabled. The 32.768 kHz clock is used to source a real-time clock counter which allows accurate timing during sleep intervals. The MSP430 architecture allows interrupts to be serviced even in sleep modes.

The onboard DCO (digitally controlled oscillator) is used during power-up initialization and as an intermediate "safe" clock during various operating modes and when switching between clocks.

SD Card Interface

The SD Card interface is implemented as low-level initialization, block reading, and block writing functions. SD block transfers are performed using the MSP430 DMA controller. The primary user functions are:

- sd_card_present() - Checks if a card is physically inserted in the slot.
- sd_read_block() - Reads a single 512-byte block from the card.
- sd_write_block() - Writes a single 512-byte block to the card.
- sd_initialize() - Initializes the SD card.

Other low-level functions for direct communication to the card exist. For more detail, refer to the source code.

FAT16 Interface

The FAT16 interface is implemented as a collection of user functions. The FAT16 is optimized for fast writes and will not handle fragmented filesystems. The overhead involved in following cluster chains makes this a necessary optimization in order to record real-time sound at high sampling rates.

- init_fat() - Initializes the FAT and builds all needed structures.
- open_file() - Opens a single file.
- fast_write_file() - Writes to a file. Assumes no fragmentation.
- close_file() - Closes a file that has been written to.
- read_file() - Reads a block from a file.

ADC

The file "adc.c" implements audio sampling. ADC sampling is triggered off TIMERB, which is sourced off the main 8MHz clock and divided to get as close to the user-requested sampling rate as possible. The typical error in the sampling rate is less than 1% at this clock frequency.

The ADC source is oversampled at 4x the requested sampling rate, averaged, shifted in sign, and normalized to 16 bits. 4 samples are obtained in a single burst every sampling period. This oversampling slightly reduces the quantization error, reducing the noise floor of the ADC12.

Samples are obtained by interrupt and stored in a user buffer. The user must supply two 512-byte preallocated buffers to the ADC sampling routine. The ADC will fill each buffer, swap buffers, and flag the variable "adcSem". The user is responsible for detecting changes to adcSem, reading the data buffer, and resetting adcSem. Buffer overruns are not detected.

User functions are described below:

- init_ADC() - Initialize the ADC and set the sampling rate

- start_sampling() - Start the sampler
- stop_sampling() - Stop the sampler