AVR 8-bit Microcontrollers

AVR42777: Digital Sound Recorder using DAC with ATtiny817

APPLICATION NOTE

Features

- Digital sound recorder using ATtiny817
- 8-bit recording
- Configurable sampling rate for high quality sound recording or longer record time
- Very small code size
- Optional data storage options, both using SPI:
  - Raw data on an SD card using multiple sector write
  - Use a serial DataFlash
- Only small changes necessary for use on other AVR devices
- Uses ADC, DAC, SPI, timer, and event system peripherals
- Exemplifies changing a register with CCP and using interrupts

Introduction

This application note describes how to record, store, and play back sound using an Atmel® AVR® microcontroller with ADC and DAC peripherals, using either an SD card or a serial flash to store raw data. It details the usage of the ADC for sound recording, the Serial Peripheral Interface – SPI – for data storage device interfacing and DAC for playback, with the use of a timer to define sampling frequency and event system for inter-peripheral signalling. Typical applications that would require one or more of these blocks are temperature loggers, telephone answering machines, or digital voice recorders. The products used in this application note include the ATtiny817 Xplained Pro board, both the I/O1 and OLED1 Xplained Pro Extension Kits, and a few extra components for the microphone and speaker circuit. Alternatively, the ATtiny817 Parrot Field Engagement Board can be used.

The ATtiny817 is used to take analog samples from a microphone and convert them to digital values with its ADC. Its built-in SPI interface controls data transfers to and from the SD card (or DataFlash). The DAC is used for playback. The sampling and playback frequency is defined using a timer, with inter-peripheral signalling via the event system and interrupts. The code size is fairly small (around 2kB including a data storage interface driver) meaning the application is suitable for smaller AVR devices.

Two accompanying projects are available for this application note in Atmel START. The implementation of sound data sampling and playback is identical for both; the only difference is the mode of data storage. The first stores raw data on an SD card, and is designed to be used with an ATtiny817 Xplained Pro and external components. The second project is for the ATtiny817 Parrot Field Engagement Board, and uses a serial DataFlash for data storage.

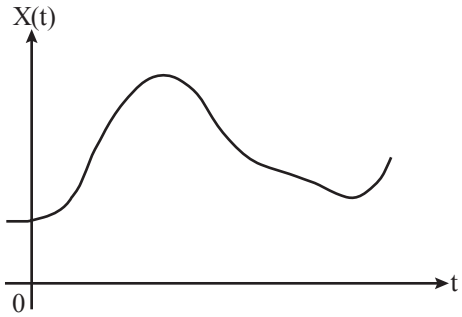
Table of Contents

1. Theory of Operation

Analog to Digital Conversion

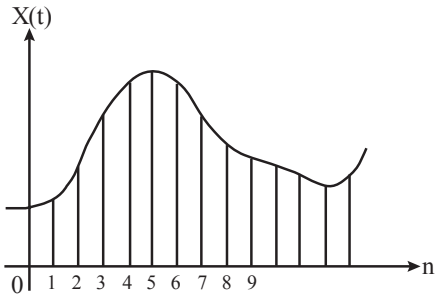
Before an analog voice signal can be stored digitally, it must be converted to a digital signal. This is done in multiple steps.

Figure 1-1. Example Analog Signal



First, the analog signal, depicted in the figure above, is converted to a time discrete signal by taking periodic samples, as shown in the figure below. The time interval between two samples is called the “sampling period” and its reciprocal the “sampling frequency”. According to the sampling theorem, the sampling frequency has to be at least double the maximum frequency component present in the sampled signal. Otherwise the periodic continuation of the signal in the frequency domain would result in spectral overlap, called “aliasing”. An aliased signal can not be uniquely reconstructed from its samples.

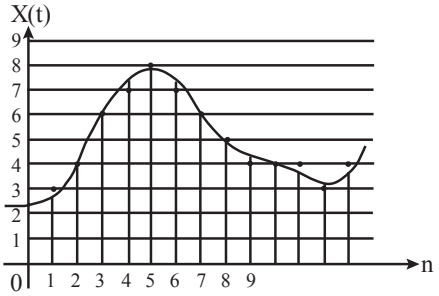
Figure 1-2. Time Discrete Signal



A speech signal contains its major information below 3kHz, therefore a low-pass filter can be used to band-limit the signal. For an ideal low-pass filter with a cut-off frequency of 3000Hz the sampling frequency must be 6000 Hz or more. Depending on the filter, the filter slope is more or less steep. Especially for a first order filter like the RC-filter used in this application it is necessary to choose a much higher sampling frequency. The upper limit is set by the features of the analog-to-digital converter.

Determining the digital values that represent the analog samples taken at this sampling frequency is called quantization. The analog signal is quantized by assigning an analog value to the nearest allowed digital value, as depicted in the figure below. The number of available digital values is called resolution and is always limited, for example to 256 values for an 8-bit digital sample. Therefore quantization of analog signals always results in a loss of information. This quantization error is inversely proportional to the resolution of the digital signal. It is also inversely proportional to the signal’s dynamic range, the range between minimum and maximum values. The conversion range of the AVR ADC can be adjusted to the dynamic range of the signal by setting the voltage reference to a maximum value suitable for the application.

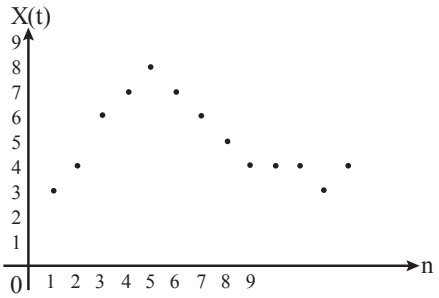
Figure 1-3. Quantized Signal



Alternately, the microphone amplifier can be designed to cover the ADC's dynamic range. Both methods reduce the quantization error.

The figure below shows the digital values that represent the analog signal. These are the values that are read as ADC conversion results, and can be stored in memory.

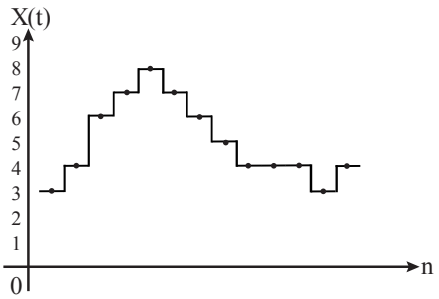
Figure 1-4. Digital Signal



Digital to Analog Conversion

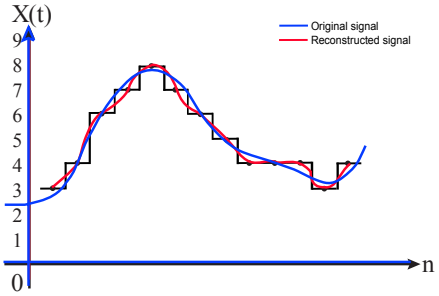
The AVR Digital-to-Analog Converter peripheral can be used to convert the digitally stored values to an analog output. By setting the voltage reference correctly to define maximum output value and using a timer to replicate sampling frequency as output frequency, the original signal is reconstructed. Each sample is held on the output for one output period (corresponding to the time between when samples were taken). For the example signal, this would look similar to the figure below.

Figure 1-5. Reconstructed Signal



The output filter smoothens the signal from the DAC. For the example, the output signal would end up similar to the figure below. The signal is very similar to the original analog input signal, except for error from quantization, which is large in the depicted example as the 3-bit samples only have eight possible values.

Figure 1-6. Output Signal After Filtering

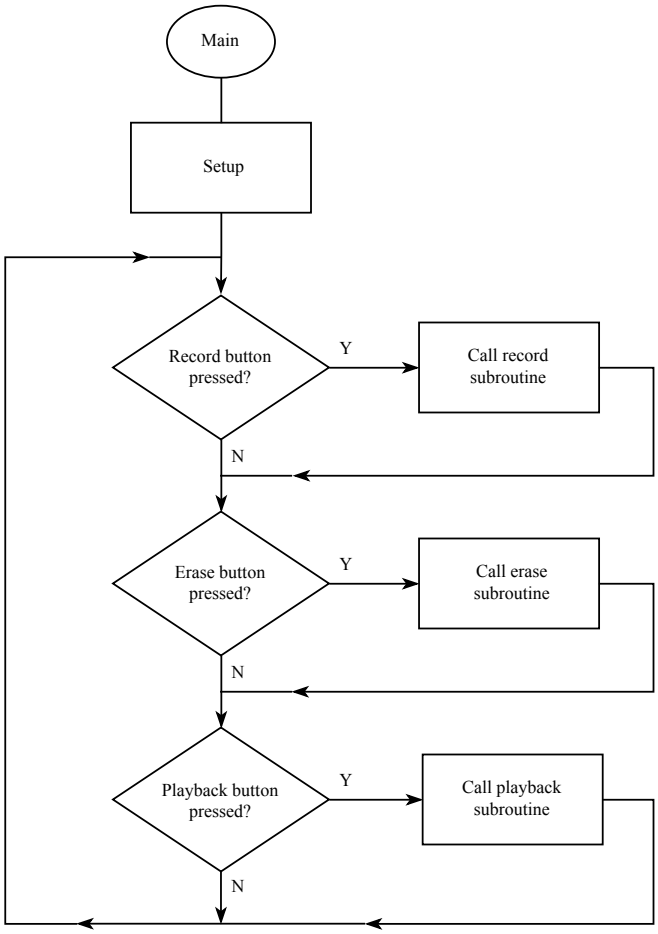


2. Implementation

Setup and Main Loop

When the program is started, setup of the external memory and internal peripherals is initiated. This involves setup of the SPI for data storage device interfacing, ADC for sampling of the microphone, DAC for output to the speaker, PORT settings for buttons, a timer for correct sampling and playback frequency, and event system for signalling the ADC. Separate buttons are set up to trigger the record, playback, and erase subroutines. This functionality is depicted in the figure below.

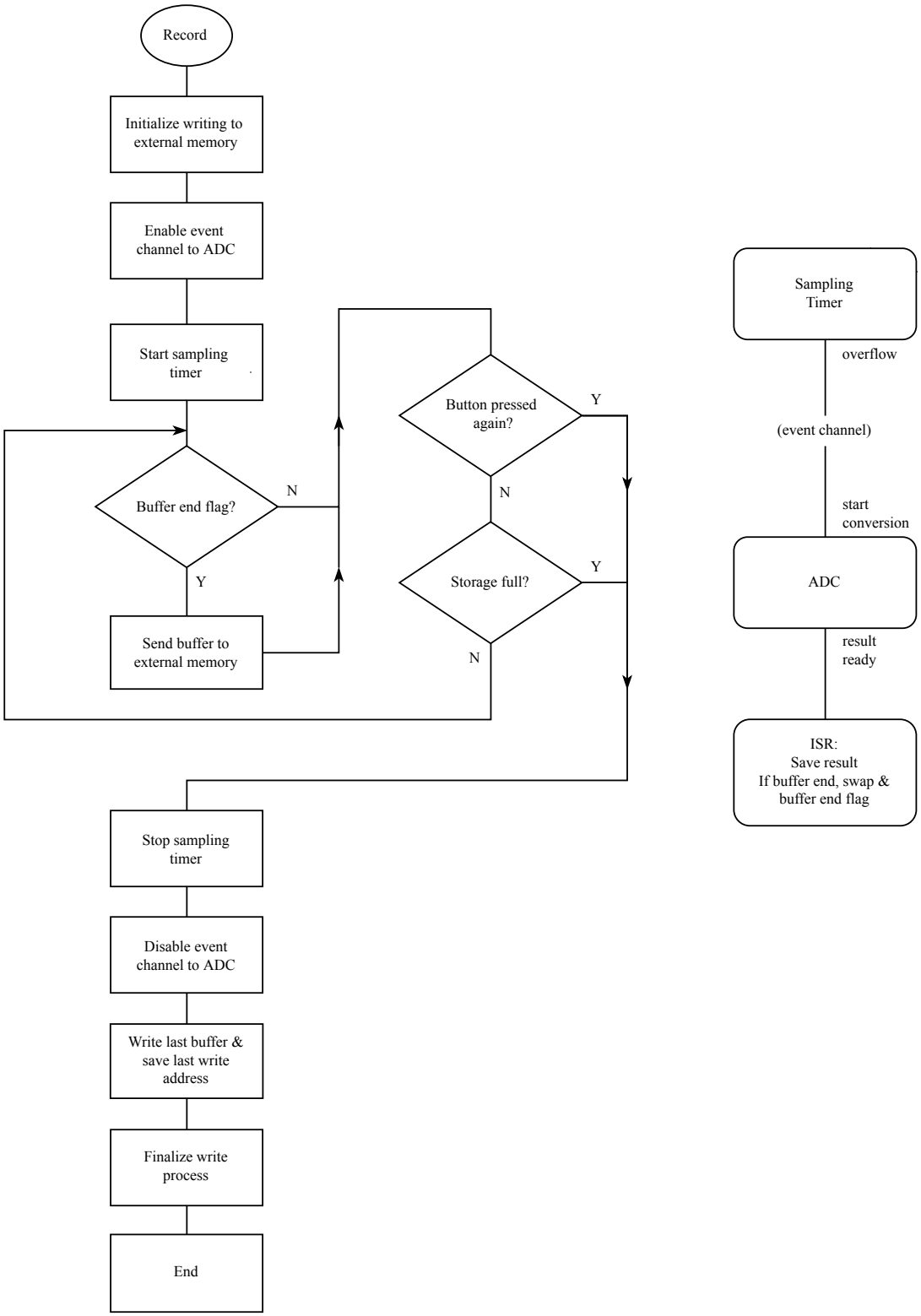
Figure 2-1. Main Loop



Record

Upon the record button being pressed, the routine depicted in the figure below is initiated. The flowchart on the left indicates functionality in software, and the right indicates hardware. The ADC is triggered to start a conversion when the sampling timer overflows. An ISR is triggered when the conversion result is ready, which stores the result in a buffer. When the buffer is full it is sent to the external memory by the software routine (signalling from the ISR to the software routine is via a globally defined flag). Two alternating buffers are used to avoid loss of data.

Figure 2-2. Record Subroutine

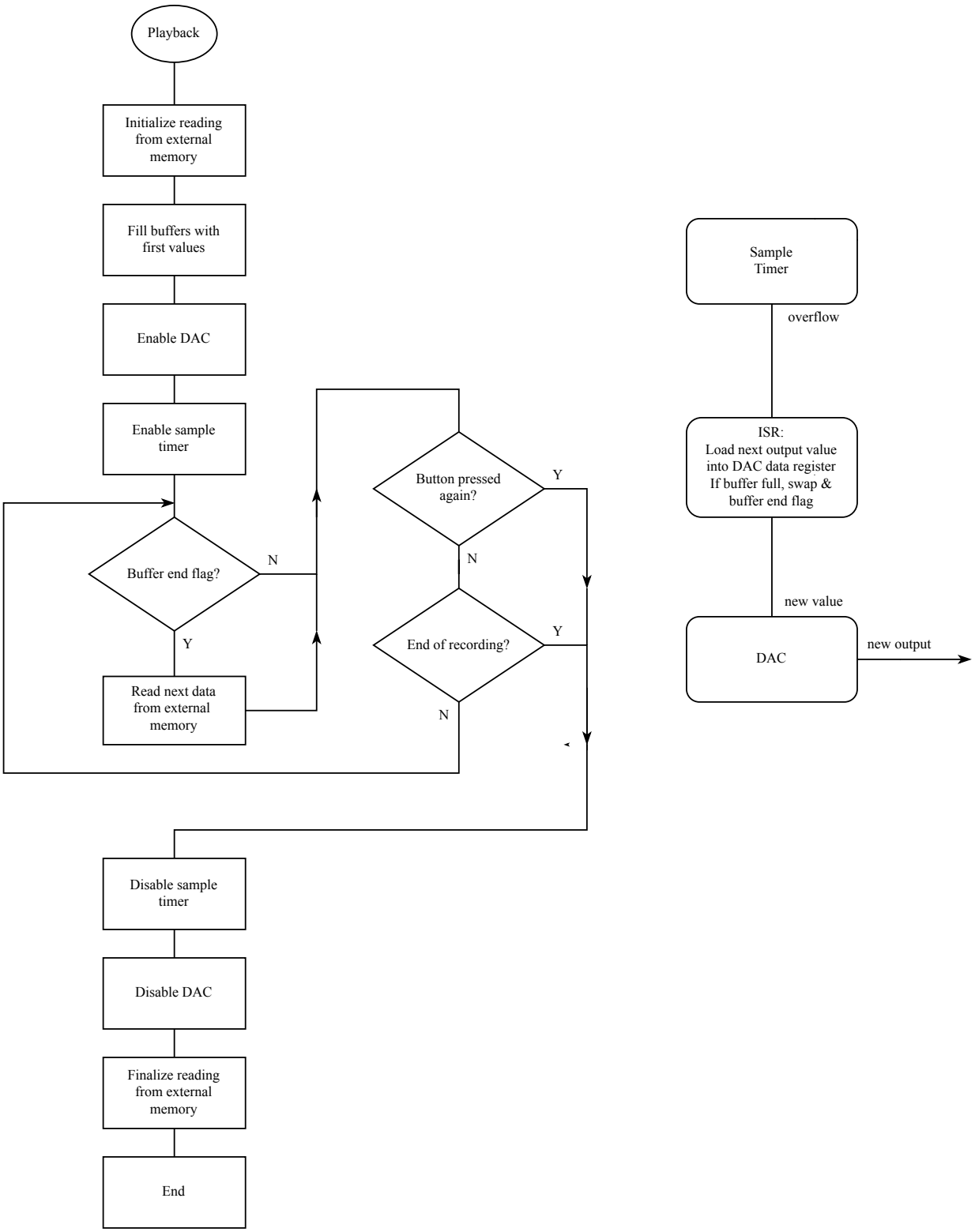


Playback

When the playback button is pressed, the routine depicted in the figure below is initiated. Again, the flowcharts on the left and right reflect functionality in software and hardware respectively. The sample

timer overflow ISR updates the DAC data register in order to change the output at the same rate as the samples were taken. This results in a reconstructed signal, which is as accurate as possible.

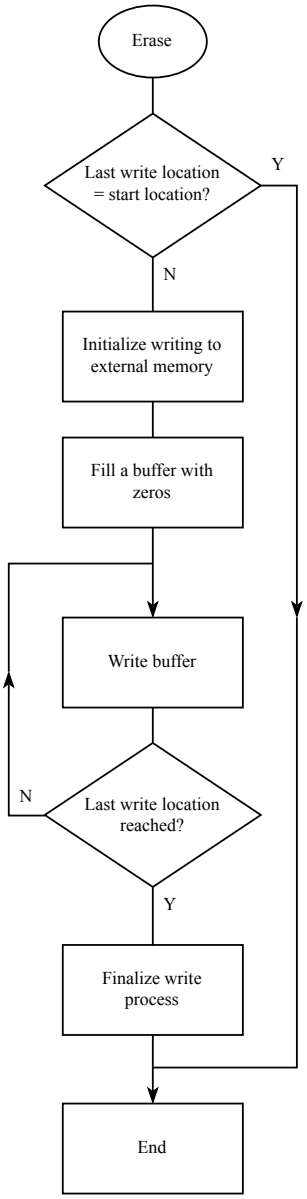
Figure 2-3. Playback Subroutine



Erase

When the erase button is pressed, the routine depicted in the figure below is initiated. This routine erases the external memory between the start of memory and the last write location.

Figure 2-4. Erase Subroutine



3. Required Hardware

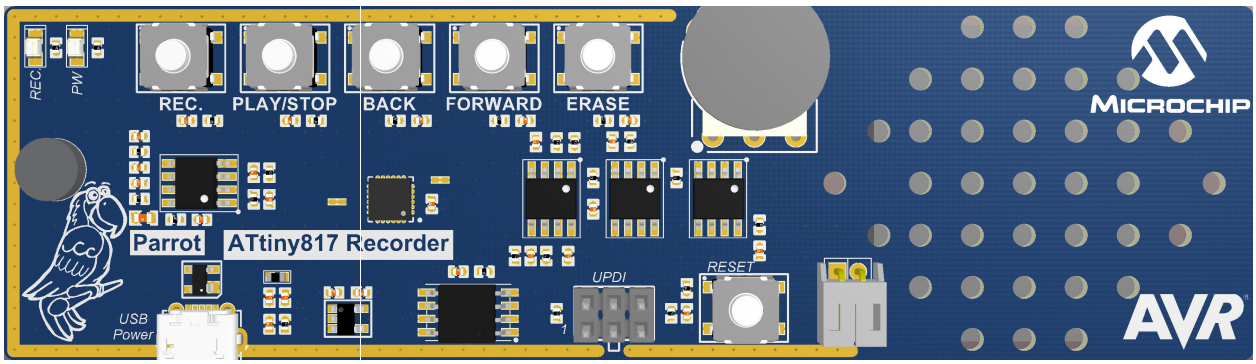
For this application note, there are two hardware setup options:

- Parrot Field Engagement Board
- Evaluation Kit Setup with ATtiny817 Xplained Pro

3.1. Parrot Field Engagement Board

The ATtiny817 Parrot Field Engagement Board employs the operational implementation outlined in this application note. It uses an 8Mb SPI Serial Flash for data storage. With the default sampling frequency it is possible to record for just over one minute using the provided firmware. Operation involves five buttons: record, play/stop, forward, back, and erase. The board is depicted in the figure below. For more information, see the Parrot Field Engagement Board Hardware User Guide.

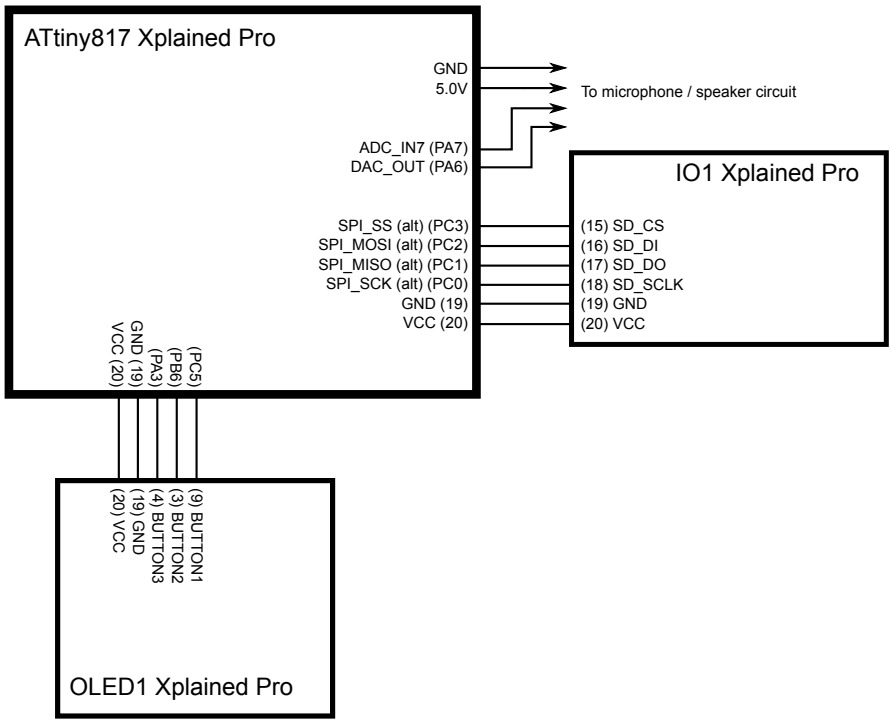
Figure 3-1. ATtiny817 Parrot Field Engagement Board



3.2. Hardware Setup Using an Evaluation Kit

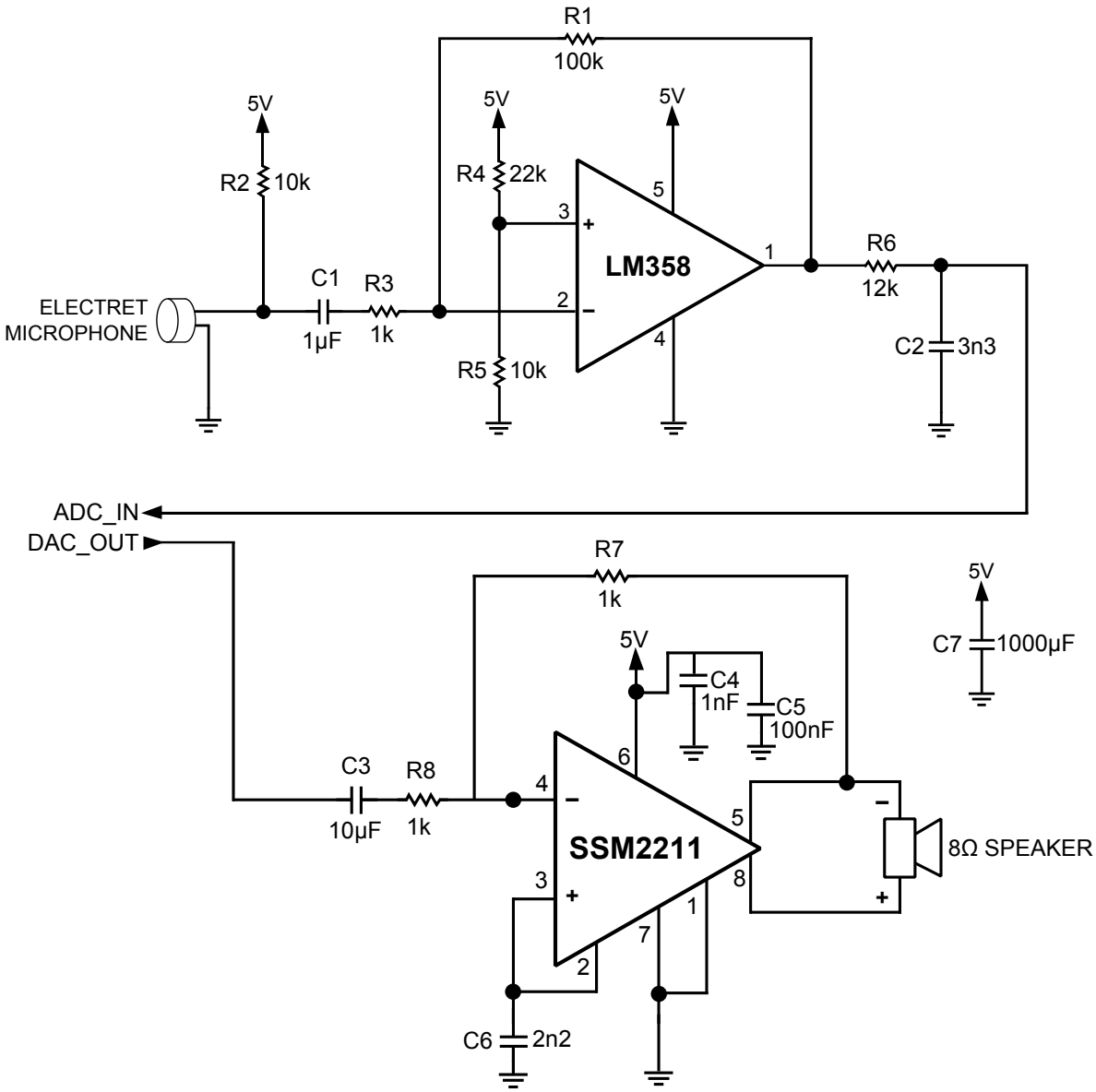
The figure below shows the necessary connections between the ATtiny817 Xplained Pro board and the required extension kits. The EXT1 header connects with the SD card via the SPI peripheral alternate pin locations, as well as the microphone and speaker circuit. The EXT3 header connects with the buttons used to trigger record, erase, and playback. It should be noted that the extension kits cannot be plugged in directly to the ATtiny817 Xplained Pro board in this case, but should be connected accordingly using jumper cables; the reasons for this include access to other signals (in the case of EXT1) and pins associated with required functionality not being connected (in the case of EXT3).

Figure 3-2. Xplained Pro Board Connections



The microphone and speaker circuit are easily made with a few extra components. This circuit should be modified depending on the microphone, speaker, and op amps available. An example is shown in the figure below.

Figure 3-3. Example Microphone and Speaker Circuit



The microphone amplifier is a simple inverting amplifier. The gain is set with R1 and R3 (gain = R1/R3). R2 is used to set the appropriate bias voltage for the microphone and C1 blocks any DC component from reaching the amplifier. R4 and R5 define the offset. R6 and C2 form a simple first order low-pass filter. In addition R6 protects the amplifier from any damage if the output is short-circuited.

The speaker circuit uses an amplifier especially designed for audio. The gain from the audio input to the speaker is set by R7 and R8 (gain = 2*R7/R8). Power supply filtering is via C4, C5, and C7. C6 provides a low impedance AC path to ground to enhance power supply noise rejection. C3 is an input coupling capacitor, which creates a high pass filter. More information can be found in the SSM2211 Datasheet*.
Note: * http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2211.pdf.

3.2.1. Writing Raw Data to an SD Card via SPI

One of the example projects accompanying this application note utilizes an SD card for data storage without a file system. This means data is written to and read from the SD card in raw format. The interface used is SPI, and the process to do this is detailed here* by ELM-Chan. The driver files included with this application enable multiple sector read and write, allowing raw data to be stored efficiently. This

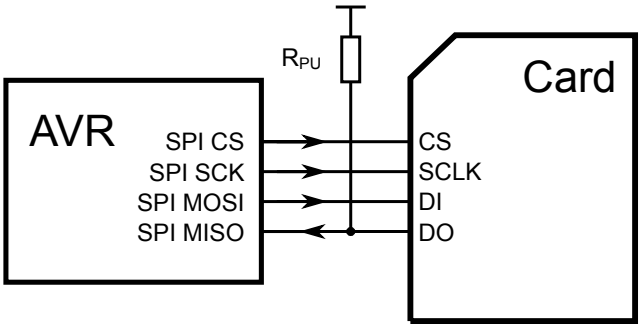
process minimizes code size and card busy time, but does not allow portability as there is no file system used; a PC can therefore not easily be used to read written raw data.

Note: * http://elm-chan.org/docs/mmc/mmc_e.html (images in this chapter are modified versions of images sourced from here).

SD Card Interfacing with SPI

Control of multimedia and SD cards without a native host interface is possible by using the card's SPI mode. An AVR SPI peripheral can be used for this with ease. The communication protocol is relatively simple, using SPI mode 0. The pin setup for the SD card can be seen in the figure below. The MISO signal should be pulled high with a pull-up resistor.

Figure 3-4. Pin Connections Between AVR and SD Card



SPI Command and Response

Any operation begins with a command sequence, as depicted in the figure below. A command frame is sent to the SD card and it replies with a response indicating the current status within command response time (N_{CR}), which is 0 to 8 bytes for SD cards. The flags contained within the response byte can be seen in Figure 3-6, and additionally to this an R3 or R7 response is defined as an R1 response with trailing 32-bit data. For most commands, a response of 0 is considered successful. A table of the commands used for storing raw data on an SD card can be seen in Table 3-1. More information on commands can be found in the specification sheets from MMCA and SDCA.

Figure 3-5. Command Sequence

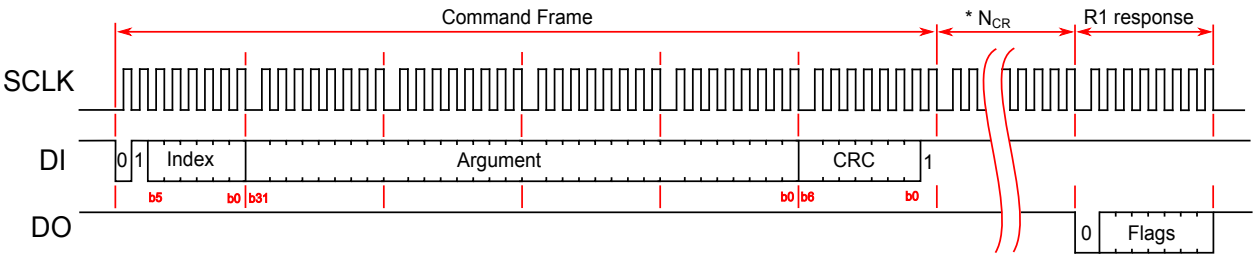
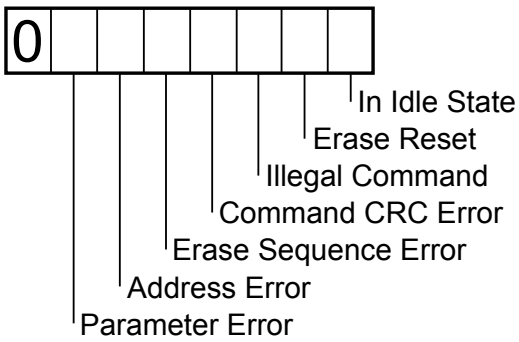


Figure 3-6. SPI Response

Command Response

R1 Response



R3 Response

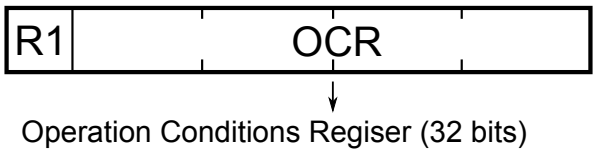


Table 3-1. SPI Commands Used

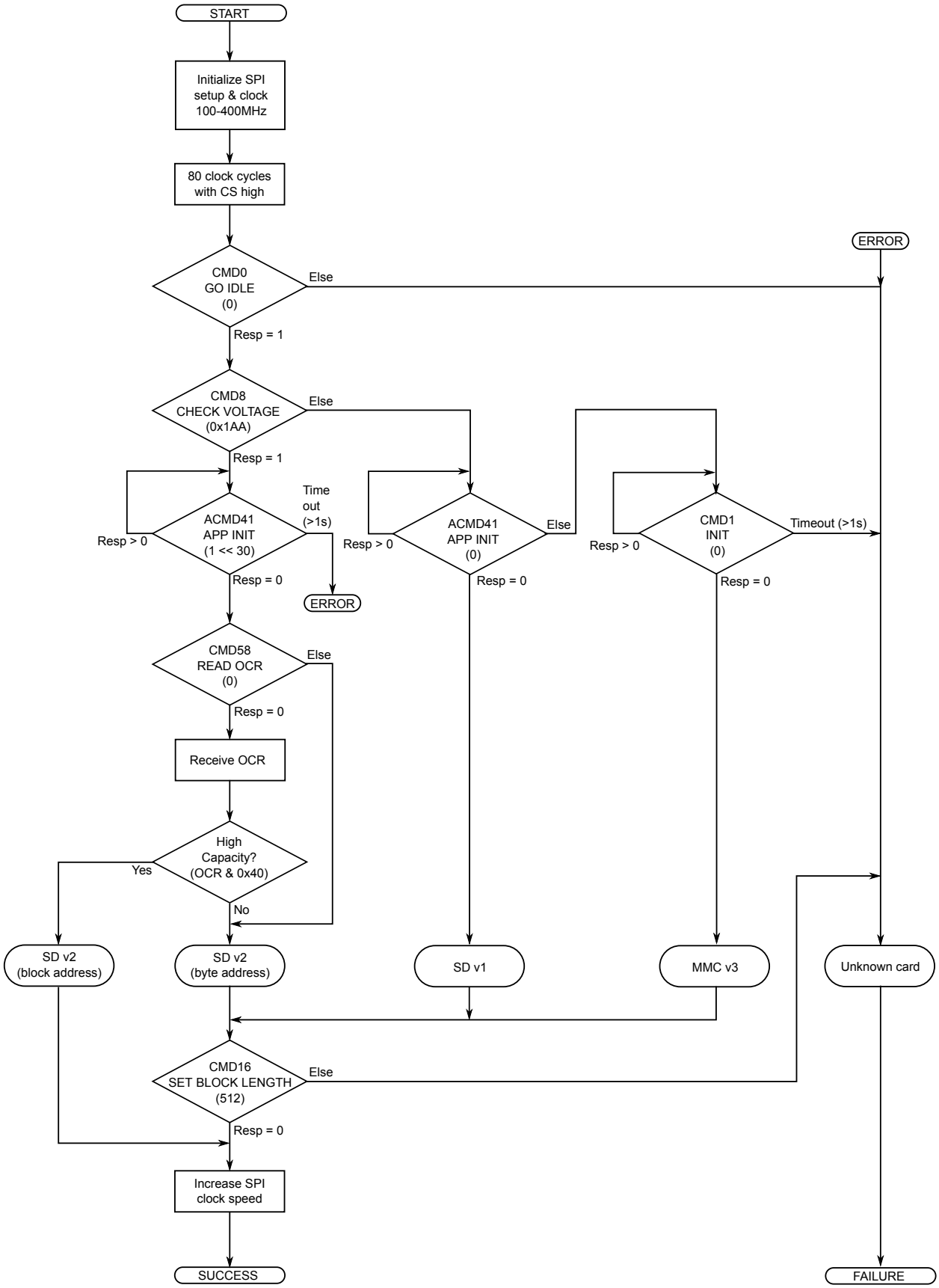
CMD index	Abbreviation	Description
CMD0	GO_IDLE	Software reset
CMD1	INIT	Initiate initialization process
ACMD41(*)	APP_INIT	For SDC only. Initiate initialization process.
CMD8	CHECK_V	For SDC v2 only. Check voltage range.
CMD12	STOP_READ	Stop reading data
CMD16	SET_BLOCKLEN	Change R/W block size
CMD18	READ_MULTI_BLOCK	Read multiple blocks
CMD25	WRITE_MULTI_BLOCK	Write multiple blocks
CMD55	ACMD_LEADING	Leading command of ACMD<n> command
CMD58	READ_OCR	Read Operation Conditions Register

Note: * ACMD<n> means a command sequence of CMD55-CMD<n>.

Initialization

The initialization process for operating an SD card in SPI mode is depicted in the figure below. This function should be the first to be called when using the driver files accompanying this application note. When this process fails continually, the SD card may need to be re-inserted from the card slot on the extension kit.

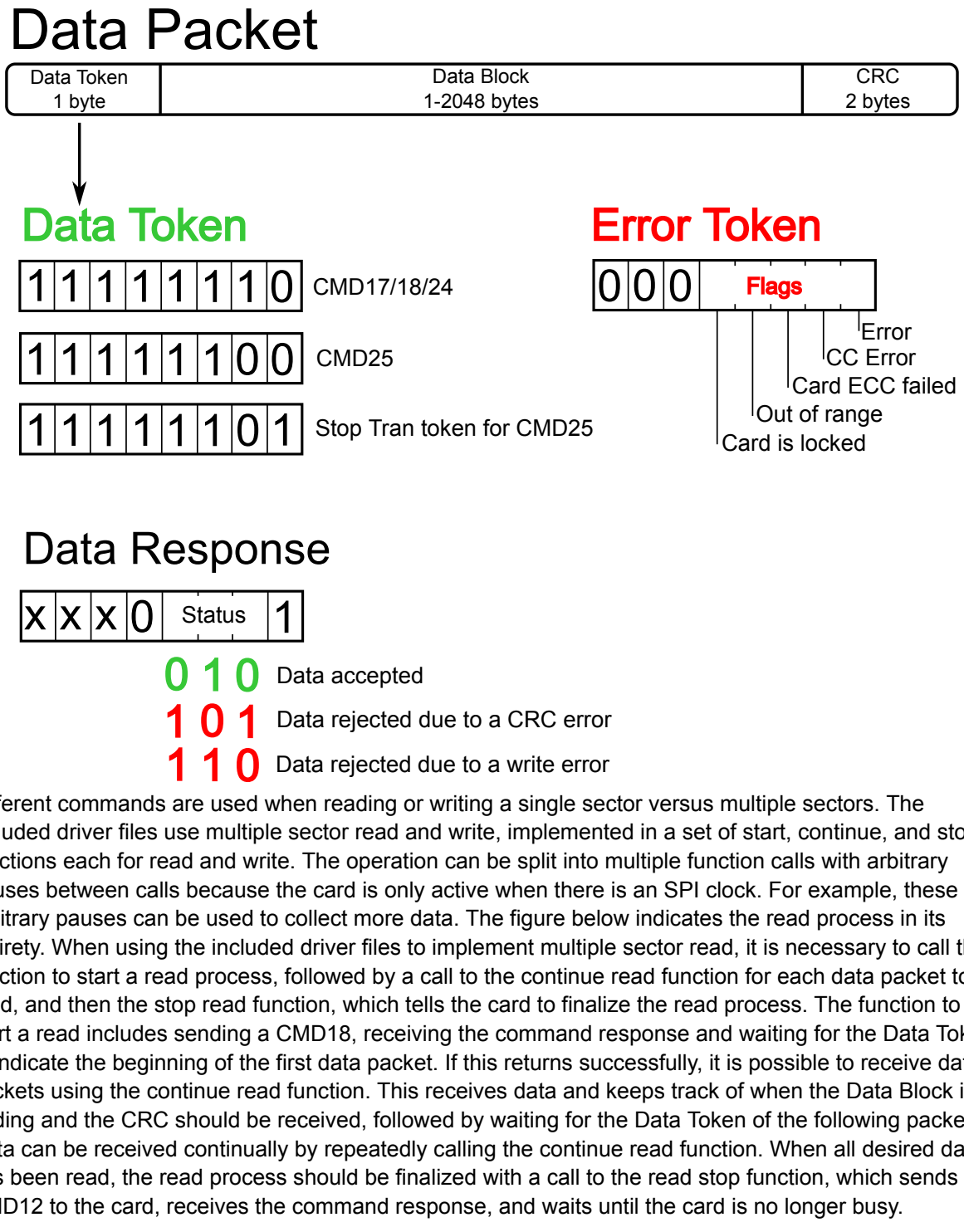
Figure 3-7. SDC/MMC Initialization in SPI Mode



Data Access

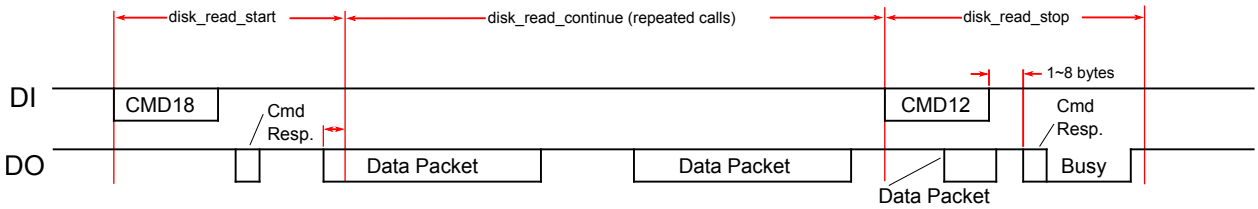
Once the SD card is correctly initialized, data transactions are possible. The format of data packets and relevant indicator bytes can be seen in the figure below. A data packet consists of a start token (Data Token), the data itself (Data Block), and a two-byte CRC value. This packet structure applies when both reading and writing, however the Data Token will vary depending on the operation in progress. The Error Token will replace the Data Token during a read in the event of an error. The Data Response byte contains status during a write.

Figure 3-8. Data Packet and Data Response



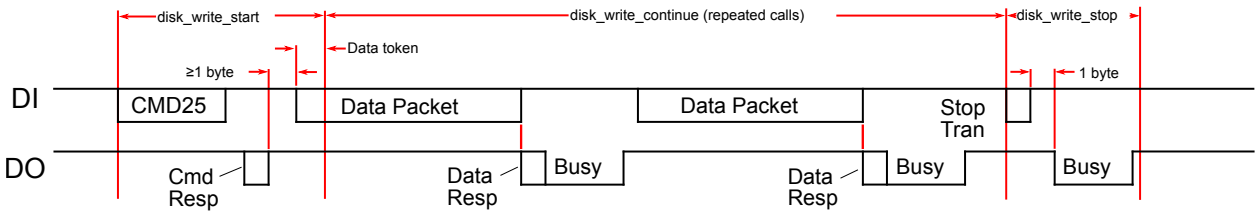
Different commands are used when reading or writing a single sector versus multiple sectors. The included driver files use multiple sector read and write, implemented in a set of start, continue, and stop functions each for read and write. The operation can be split into multiple function calls with arbitrary pauses between calls because the card is only active when there is an SPI clock. For example, these arbitrary pauses can be used to collect more data. The figure below indicates the read process in its entirety. When using the included driver files to implement multiple sector read, it is necessary to call the function to start a read process, followed by a call to the continue read function for each data packet to be read, and then the stop read function, which tells the card to finalize the read process. The function to start a read includes sending a CMD18, receiving the command response and waiting for the Data Token to indicate the beginning of the first data packet. If this returns successfully, it is possible to receive data packets using the continue read function. This receives data and keeps track of when the Data Block is ending and the CRC should be received, followed by waiting for the Data Token of the following packet. Data can be received continually by repeatedly calling the continue read function. When all desired data has been read, the read process should be finalized with a call to the read stop function, which sends a CMD12 to the card, receives the command response, and waits until the card is no longer busy.

Figure 3-9. Multiple Sector Read Process



The multiple sector write process is depicted in the figure below. When using the driver files, the start write function should be called first to begin a write process. This sends a CMD25 to the card, receives the command response, and sends one dummy byte, which is required before sending the first Data Packet. The write continue function can then be used to send data. This function also keeps track of when the Data Block should be followed by two CRC bytes, and then verifies the Data Response and waits for the end of the card busy time. Data can be continually sent by repeatedly calling the write continue function. When all necessary data has been written to the card, the write process should be finalized using the stop write function. This sends a Stop Token (Data Token for CMD25) to indicate to the card that all the data has been sent. The function will return after the final card busy time has ended.

Figure 3-10. Multiple Sector Write Process



3.2.1.1. Storing Digital Sound as Raw Data on an SD Card

The process defined above is used with one of the accompanying example projects provided to store a digitally recorded signal externally to the AVR device. This enables a device with limited memory such as the ATtiny817 to perform data collection on a scale not possible with its internal memory. The described start, continue, and stop functions for both read and write are used in the record and playback functions. Using the SD card in SPI mode enables the data to be read and written at a rate defined by the SPI clock and at intervals defined by how fast the end of the current buffer is reached.

3.2.1.2. Limitations

This application note demonstrates that a complex task can be performed using a device with limited memory, such as the ATtiny817. However, this implementation does have some limitations.

Data Portability

The data representing the recorded digital voice signal cannot be easily read by a PC. Even though it is stored on an SD card, there is no file system used, so the data is stored in a raw format without a structure readable by typical PC software.

Power Stability

The SD card used to store data may enter an undefined state if its power source goes below a certain level (also relevant when power toggles). This will result in the SD card responding correctly during initialization and when writing data, however it may cause an error when trying to perform a read operation. This requires the card to be removed from the slot and re-inserted, triggering a reinitialization. Read can then be performed correctly. This state is indicated by a record operation performing correctly, followed by a playback with no sound and a periodic flash of the LED. After the SD card has been removed and reinserted, the playback button should be pressed again, at which time the recorded data can be played back successfully.

4. Get Source Code from Atmel START

The example code is available through Atmel START, which is a web-based tool that enables configuration of application code through a graphical user interface. The code can be downloaded for both Atmel Studio 7.0 and IAR™ IDE via the **Examples**-link below, or the **BROWSE EXAMPLES** button on the Atmel START front page.

Web page: http://start.atmel.com/

Documentation: http://start.atmel.com/static/help/index.html

Examples: http://start.atmel.com/#examples

In the Examples-browser, search for: AVR42777 (press **User Guide** in Atmel START for detailed requirements for the example project).

Double-click the downloaded .atzip file and the project will be imported to Atmel Studio 7.0.

For information on how to import the project in IAR, press the **Documentation**-link above, select ‘Atmel Start Output in External Tools' and 'IAR Embedded Workbench®'.

4.1. Code Configuration

Two projects are available to accompany this application note:

- AVR42777 Digital Sound Recorder: uses raw data on an SD card for storing the recorded data.
- AVR42777 Parrot: uses a serial DataFlash for data storage.

Both use the same peripheral setup to implement recording and playback.

Optionally, the sampling frequency can be changed in order to either increase the sound quality (sampling frequency increased) or increase the amount of possible recording time (sampling frequency decreased). To do this, simply change the hash define "SAMPLE_FREQ" at the top of the main file (this will be either voice_recorder.c or parrot.c depending on which project has been chosen).

5. Revision History

Doc. Rev.	Date	Comments
42777A	10/2016	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**