AVR335: Digital Sound Recorder with AVR® and DataFlash®

Features

- **Digital Voice Recorder**
- **8-bit Sound Recording**
- **8 kHz Sampling Rate**
- **Sound Frequency up to 4000 Hz**
- **Maximum Recording Time 4 1/4 Minutes**
- **Very Small Board Size**
- **Only 550 Bytes of Code**

Introduction

This application note describes how to record, store and play back sound using any AVR microcontroller with A/D converter, the AT45DB161B DataFlash memory and a few extra components.

This application note shows in detail the usage of the A/D Converter for sound recording, the Serial Peripheral Interface – SPI – for accessing the external DataFlash memory and the Pulse Width Modulation – PWM – for playback. Typical applications that would require one or more of these blocks are temperature loggers, telephone answering machines, or digital voice recorders.

The AT45DB161B DataFlash is a 2.7 volt only, Serial-interface Flash memory. Its 16M-bit of memory are organized as 4096 pages of 528 bytes each. In addition to its main memory, the DataFlash contains two SRAM data buffers of 528 bytes each. The buffers allow a virtually continuous data stream to be written to the DataFlash.

The AT45DB161B uses an SPI serial interface to sequentially access its data. This interface facilitates hardware layout, increases system reliability, minimizes switching noise, and reduces package size and active pin count. Typical applications are image storage, data storage and digital voice storage. The DataFlash operates at SPI clock frequencies up to 20 MHz with a typical active read current consumption of 4 mA. It operates from a single voltage power supply (from 2.7V to 3.6V) for both the write and read operations.

Its serial interface is compatible to the Serial Peripheral Interface – SPI – Modes 0 and 3, thus it can easily be interfaced to the AVR microcontroller.

In this application note the AVR AT90S8535 is used to take analog samples from a microphone and convert them to digital values. Its built-in SPI controls data transfers to and from the DataFlash. The PWM feature of the AVR is used for playback. The code size is very small (< 550 bytes), the application will therefore also fit into smaller AVR devices.

8-bit AVR® RISC Microcontroller

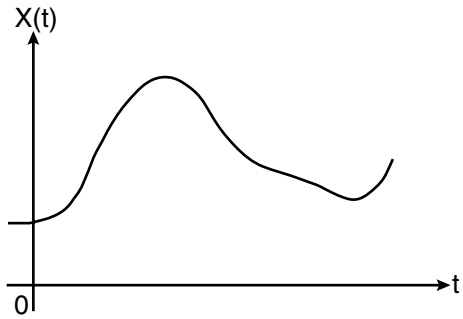
Application Note

Rev. 1456B–AVR–03/02

Theory of Operation

Before the analog speech signal can be stored in the DataFlash it has to be converted into a digital signal. This is done in multiple steps.

Figure 1. The Example Analog Signal



First, the analog signal (Figure 1) is converted into a time discrete signal by taking periodic samples (Figure 2). 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 signal frequency. Otherwise the periodic continuation of the signal in the frequency domain would result in spectral overlap, called “aliasing”. Such an aliased signal can not be uniquely recovered from its samples.

A speech signal contains its major information below 3000 Hz. 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 3000 Hz the sampling frequency must be 6000 Hz. 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 A/D-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 (Figure 3). The number of digital values is called “resolution” and is always limited, for example to 256 values for an 8-bit digital signal or 10 values in this example. Therefore quantization of analog signals always results in a loss of information. This “quantization error” is inversly proportional to the resolution of the digital signal. It is also inversly proportional to the signal’s “dynamic range”, the range between minimum and maximum values (3 to 8 in this example). The A/D converter of the AT90S8535 microcontroller can be adjusted to the dynamic range of the signal by setting AGND and AREF to the minimum and maximum signal values.

On the other hand, the microphone amplifier can be adjusted to cover the ADC’s dynamic range as presented later.

Both methods reduce the quantization error. In addition, the latter method also increases the signal-to-noise ratio – SNR – and should therefore be preferred.

Figure 4 shows the digital values that represent the analog signal. These are the values that are read as ADC conversion results.

In this application, the signal has a minimum and a maximum value which are never exceeded. The parts of the signal below the minimum and above the maximum value do not contain any information. They can be removed in order to save memory.

This is done by downshifting the whole signal and discarding the part above the “max” value (Figure 5).

Figure 2. The Time Discrete Signal

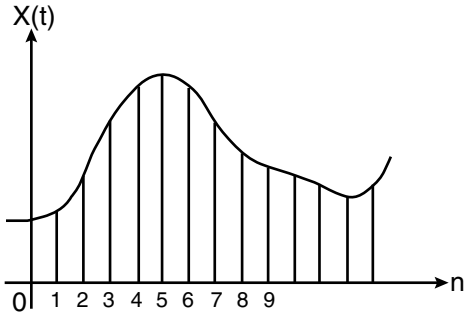


Figure 3. The Quantized Signal

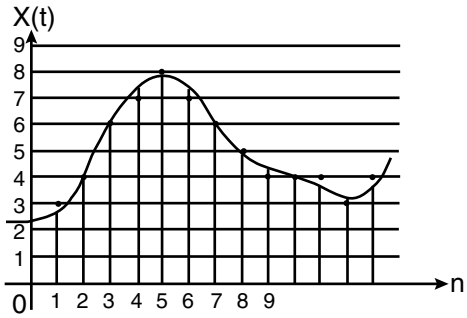


Figure 4. The Digital Signal

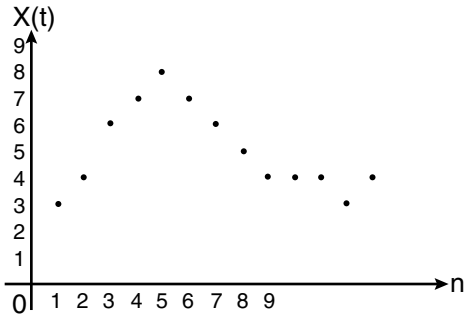
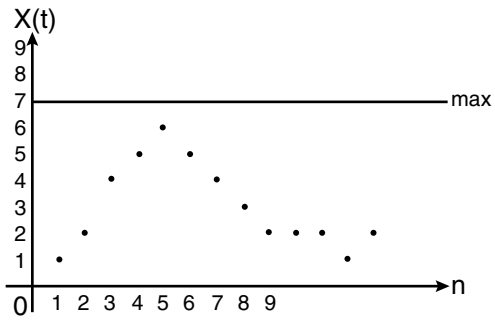


Figure 5. The Bit-reduced Digital Signal



In this application the resulting signal has 8 bits. This signal can now be stored in the DataFlash.

The DataFlash does not require a separate erase cycle prior to programming. When using the "Buffer to Main Memory Page Program with Built-In Erase" or "Main Memory Page Program Through Buffer" commands, the DataFlash will automatically erase the specified page within the memory array before programming the actual data. If systems require faster programming throughputs (greater than 200K bps), then areas of the main memory array can be pre-erased to reduce overall programming times. An optional "Page Erase" command is provided to erase a single page of memory while the optional "Block Erase" command allows eight pages of memory to be erased at one time. When pre-erasing portions of the main memory array, the "Buffer to Main Memory Page Program without Built-In Erase" command should be utilized to achieve faster programming times.

The first method is the most code efficient, as no extra erase cycles have to be implemented. However this application note uses the block erase to illustrate how large portions of the memory can be pre-erased if so desired. Erasing the entire memory can take up to a few seconds.

After the memory has been erased, data can be recorded until all pages are filled up.

For writing to the DataFlash, Buffer 1 is used. When this buffer is filled up (with 528 samples) the buffer is written to the main memory while the 529th conversion is done. Data is recorded until the “Record” button is released or the memory is full. If the entire memory is filled up, no new data can be stored before the DataFlash is erased. If the memory is only partly filled and the “Record” button is pressed a second time, the new data is appended directly to the existing data.

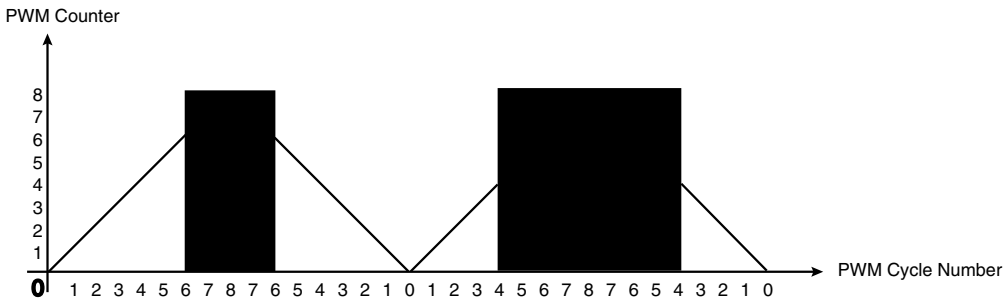
Playback of sound is always started at the beginning of the DataFlash. It stops when either all recorded data is played back or when the “Playback” button is released.

The DataFlash allows reading back data either directly from a main memory page or by copying a page to one of the two buffers and reading from the buffer. The direct access method is not suitable for this application as two addresses, one for page and one for byte position, and a long initialization sequence have to be transferred to the DataFlash for each single byte. This takes much longer than one PWM cycle, which is 510 clock cycles for an 8-bit PWM signal.

Therefore, one memory page is copied to one of the two buffers. While data is read from this buffer the next memory page is copied into the other buffer. When all data has been read from the first buffer reading continues on the other buffer, while the first one is reloaded.

Reading data from the DataFlash buffer is synchronized to the PWM frequency.

Figure 6. Two Example PWM Cycles



The digital value is played back by using pulse width modulation (PWM). In Figure 6, samples 2 and 3 of the example signal are shown. One cycle of the PWM signal consists of a counter counting up to the maximum value that can be represented by the given resolution (8 in this example), and counting down to zero again. The output is switched on when the PWM counter matches the value of the digital signal value and is switched off when it falls below this value again. Therefore the dark area represents the power of the signal at that sample. Figure 7 shows the PWM output signal for the example signal.

The PWM frequency has to be at least twice the signal frequency. A PWM frequency at least four times higher is recommended, depending on the output filter.

This can be achieved either by reducing signal frequency, increasing system clock frequency or reducing signal resolution.

Figure 7. The Filtered PWM Output Signal

In this application the cut-off frequency of the output filter is set to 4000 Hz, which is roughly one quarter of the PWM frequency (15,686 Hz).

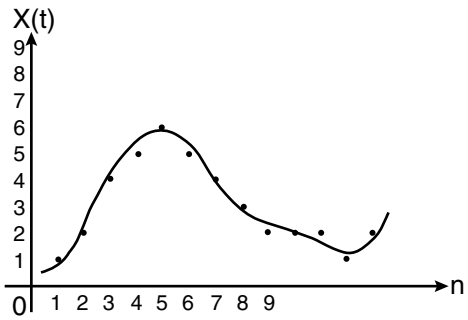
The system clock speed and the PWM resolution determines the PWM frequency.

With an 8 MHz system clock, the frequency for a 10-bit PWM is 3922 Hz (8 MHz / $2 \bullet 2^{10}$ = 3922 Hz), 7843 Hz for 9-bit resolution, and 15,686 Hz for 8-bit resolution.

Only the last value is high enough to serve as carrier frequency for the 4000 Hz signal. Therefore, the original 10-bit digital sample is converted to 8 bits.

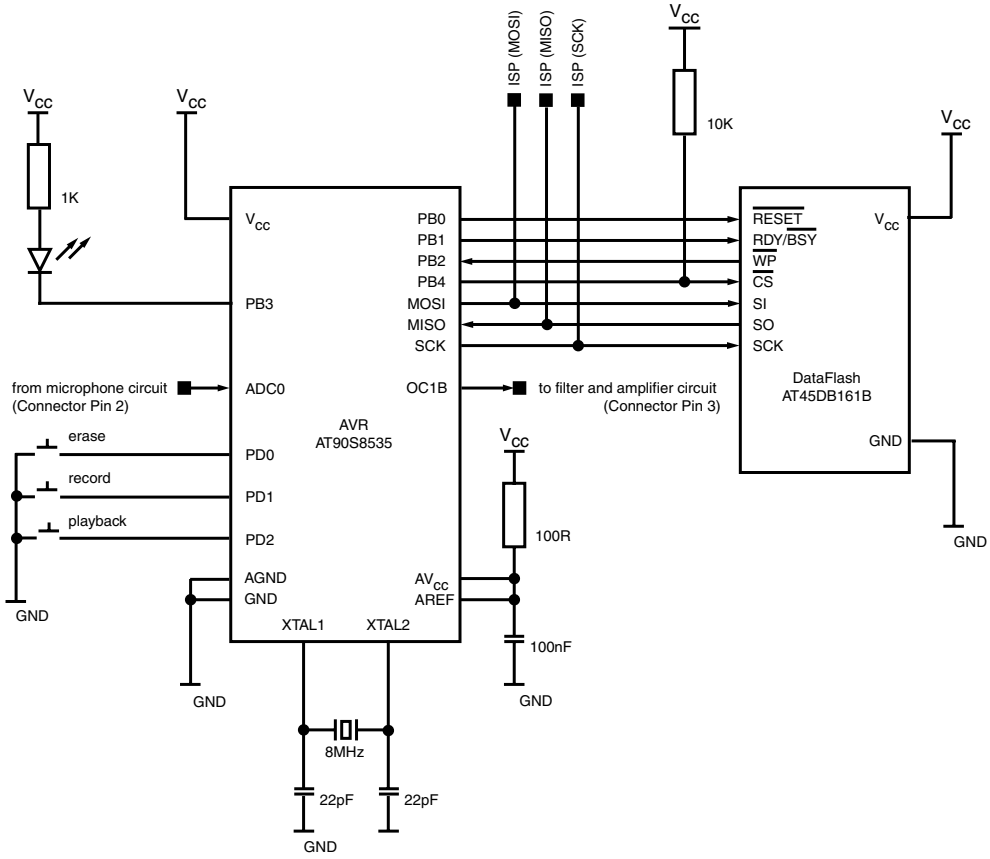
The output filter smoothens the output signal and removes the high-frequency PWM carrier signal. The resulting output signal for the example signal now looks somehow like the drawing in Figure 8. Except for the quantization errors (which are very large in this example as only 8 digital values are used) and a missing amplification, the signal looks almost like the analog input signal (Figure 1).

Figure 8. The PWM Output Signal



Microcontroller and Memory Circuit

Figure 9. Microcontroller and Memory Circuit Diagram



The user can control the sound system with three pushbuttons, called “Erase”, “Record” and “Playback”. If the pushbuttons are not pressed, the internal pull-up resistors provide V_{CC} at PD0 - PD2. Pushing a button pulls the input line to GND.

As feedback for the user, an LED indicates the status of the system.

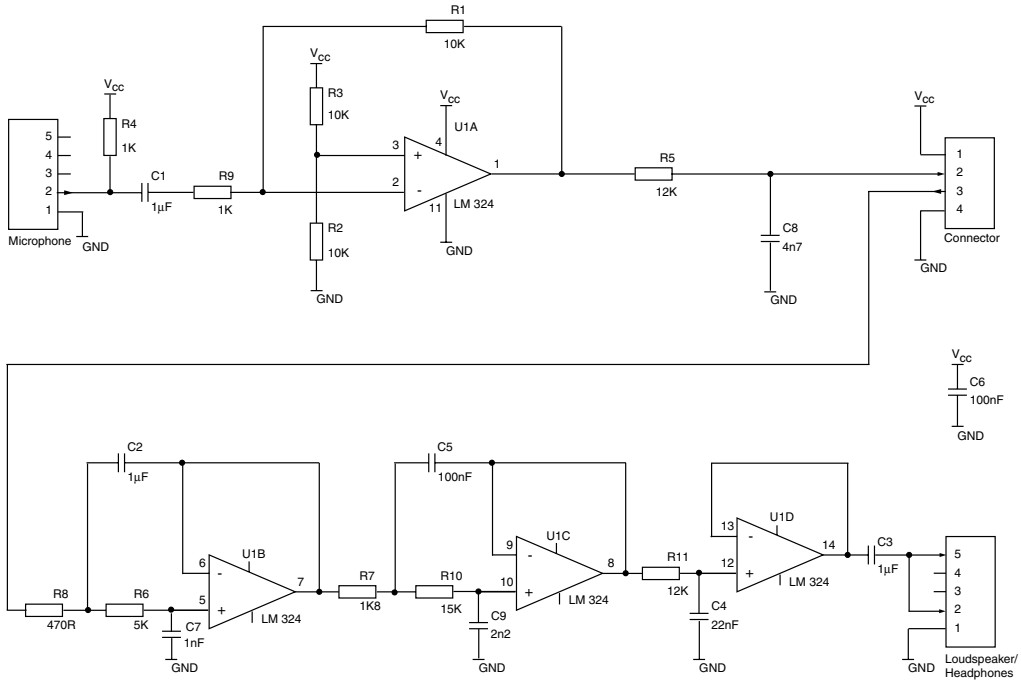
The DataFlash is directly connected to the AVR microcontroller using the SPI bus. In case the ISP feature is used to reprogram the AVR, the pull-up resistor on the Chip Select line ($\overline{CS}$) prevents the DataFlash from going active. If the ISP feature is not used, this resistor can be omitted.

The analog voltage, AVCC, is connected to V_{CC} by an RC low-pass filter. The reference voltage is set to AVCC.

The oscillator crystal with two 22 pF decoupling capacitors generates the system clock.

Microphone and Speaker Circuit

Figure 10. Microphone and Speaker Circuit Diagram



The microphone amplifier is a simple inverting amplifier. The gain is set with R1 and R9 (gain = R1 / R9). R4 is used to power the microphone and C1 blocks any DC component to the amplifier. R2 and R3 set the offset. R5 and C8 form a simple first order low-pass filter. In addition R5 protects the amplifier from any damage if the output is short-circuited.

The speaker circuit consists of a 5th-order, low-pass Chebychev filter and a unary-gain amplifier.

The filter is made up by two stagger-tuned, 2nd-order Chebychev filters (R6, R7, R8, C2, C7 and R7, R10, R11, C9, C5) and a passive 1st-order filter (R11, C4). The cut-off frequencies of these three filters are slightly shifted against each other (“staggered”) to limit passband ripple of the whole filter circuit. The overall cut-off frequency is set to 4000 Hz, which is roughly one-quarter of the PWM frequency (15,686 Hz).

The unary-gain amplifier prevents the circuit from getting feedback from the output.

C3 blocks any DC component to the speaker.

Implementation

Setup

When the program is started the ports have to be set up. This is done in the “setup” subroutine.

The SPI protocol defines one device as a master and the other devices connected to this master as slaves. In this application the AVR microcontroller functions as a master and the DataFlash as a slave. As the AT90S8535 is the only master in this application the SS pin can be used as an I/O pin.

The SPI of the AT90S8535 is defined as an alternative function of Port B (PB5 to PB7). In this application the control signals for the DataFlash are also set up on Port B (PB0 to PB2 and PB4). The free pin (PB3) is used to control the status LED. For master setup,

the signals Serial Clock (SCK), Master Out/Slave In (MOSI), Chip Select ($\overline{CS}$), Write Protect ($\overline{WP}$) and Reset ($\overline{RST}$) are outputs, whereas Master In/Slave Out (MISO) and Ready/Busy (RDY/$\overline{BSY}$) are inputs. With PB3 for the LED also defined as an output the Data Direction Register for Port B is set up as 0xBD.

Then the PortB is set to a defined status with all outputs high and internal pull-up resistors on the inputs.

The A/D converter of the AT90S8535 is connected to PortA. Therefore PortA is defined as a high-impedance input.

PortD serves as an input for the pushbuttons and as an output for the PWM signal. Here the PWM function of Timer1 on the output pin PD4 is used.

In the end, interrupts are enabled. In this application two interrupts (“ADC” and “Timer1 Overflow”) are used, which are enabled and disabled directly in the subroutine when they are required.

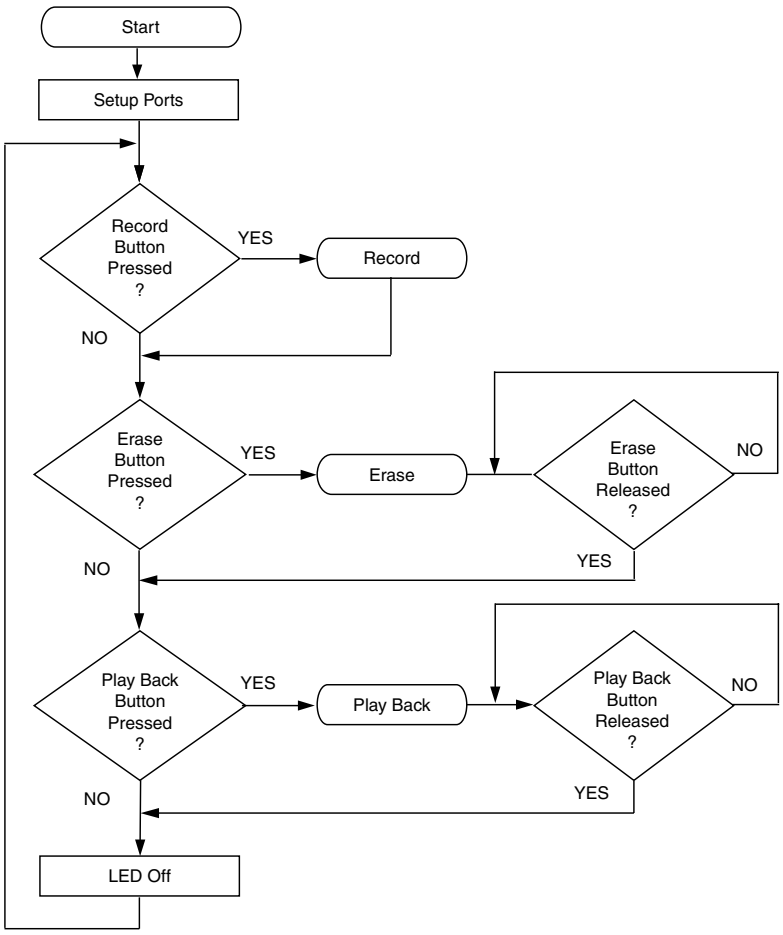
The Main Loop

In the main loop, the three pushbuttons are scanned. If one of them is pressed, the LED is turned on to show that the system is busy and the corresponding subroutine is called.

An extra loop is performed, until the button is released, as a software debounce for the “Erase” and “Playback” functions.

During the main loop, the LED is turned off to indicate that the system is running idle.

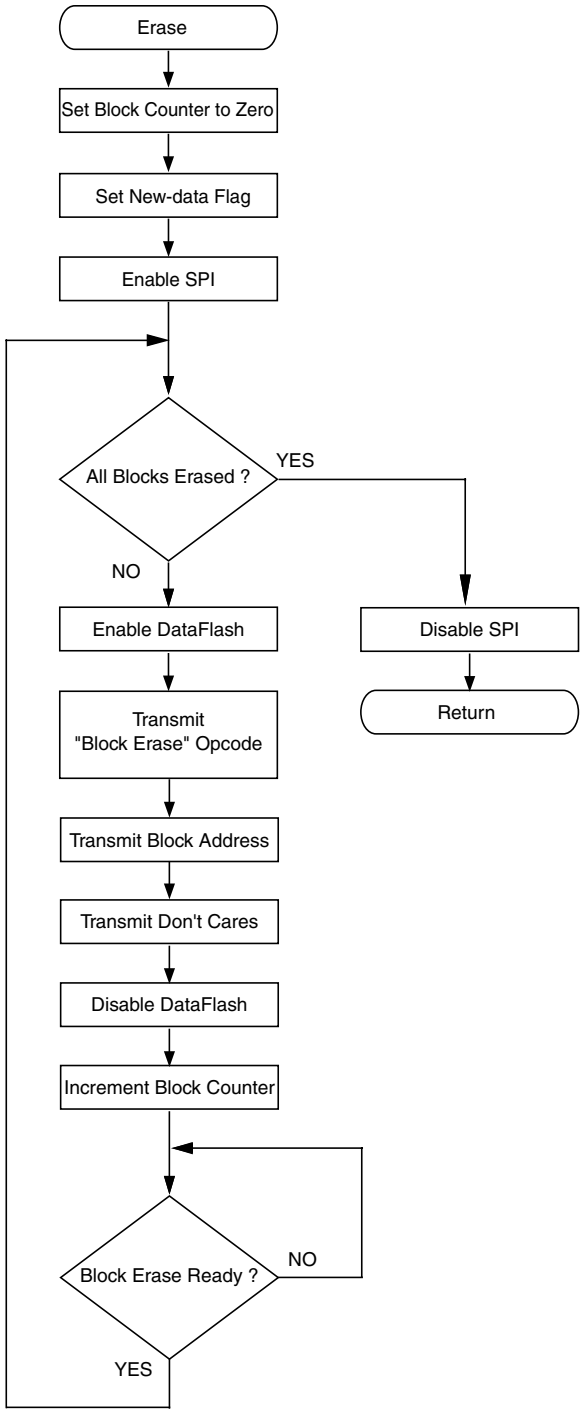
Figure 11. The Main Loop



Erase

The DataFlash can be optionally pre-erased.

Figure 12. Erase



When the “Erase” subroutine is called, a flag is set which indicates that in the next recording cycle the new data can be stored at the beginning of the DataFlash.

The SPI has to be set up for accessing the DataFlash. No interrupts are used here. The data order for the DataFlash is MSB first and the AT90S8535 is the master.

The DataFlash accepts either the SCK signal being low when $\overline{CS}$ toggles from high to low (SPI Mode 0) or the SCK signal being high when $\overline{CS}$ toggles from high to low (SPI Mode 3) with a positive clock phase. In this application the SPI is set up in Mode 3. In order to get the fastest data transfer possible, the lowest clock division is chosen, running the SPI bus at 2 MHz if an oscillator crystal of 8 MHz is used.

To perform a block erase, the $\overline{CS}$ line is driven low and the opcode 0x50 is loaded into the DataFlash followed by two reserved bits (zeros), the 9-bit block address, and 13 don't care bits. This sequence is transferred to the slave bytewise. After each byte, the SPI Status Register – SPSR – is checked until the SPI Interrupt Flag indicates that the serial transfer is complete. After the whole sequence is written, erasing of the block is started when the $\overline{CS}$ line is driven high. The Ready/Busy pin is driven low by the DataFlash until the block is erased. Then the next block will be erased in the same way as the current. This takes place until all 512 blocks are erased. An erased location reads 0xFF.

Record

The record subroutine consists of the setup of the A/D converter and an empty loop which is performed as long as the "Record" button is pressed. The ADC0 pin is used in this application which requires the ADC Multiplexer Select Register (ADMUX) being set to zero. In the ADC Control and Status Register (ADCSR) the ADC is enabled with a clock division factor of 32, set to single conversion mode, interrupts enabled, and the interrupt flag is cleared. The A/D conversion is also immediately started. The first conversion takes longer than the following conversions (832 oscillator cycles instead of 448). After this time, the ADC interrupt occurs indicating that the conversion is finished and the result can be read out of the ADC Data Register.

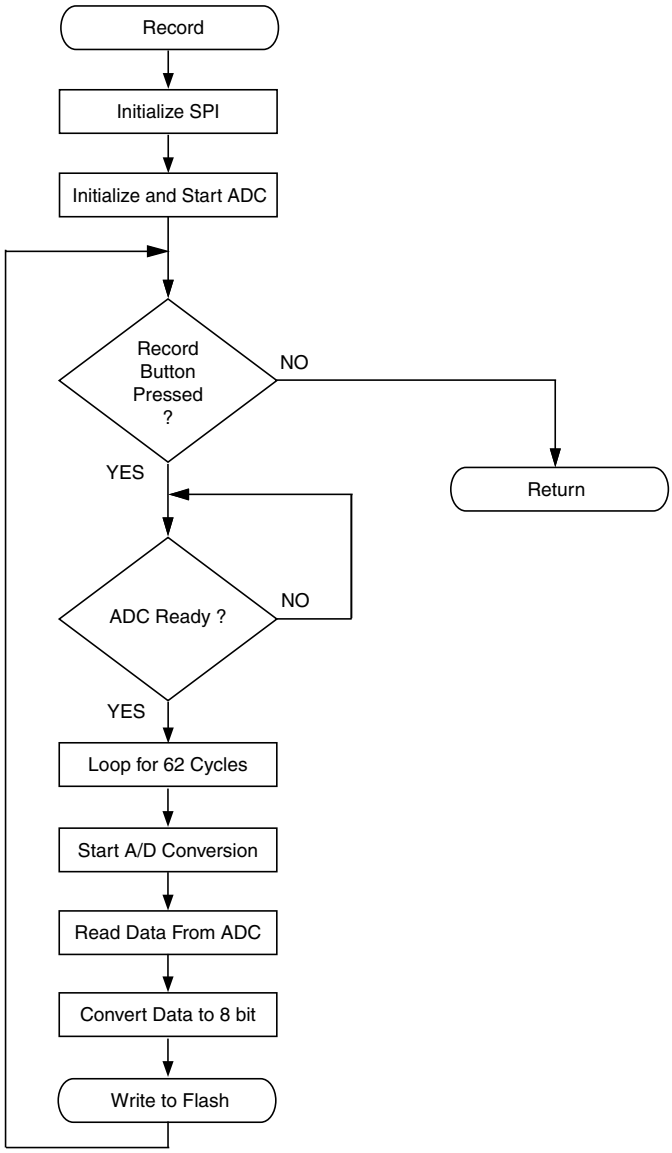
The analog signal from the microphone circuit is sampled at 15,686 Hz. This is the same frequency as the output (PWM) frequency.

To achieve a sampling frequency of 15,686 Hz, a sample has to be taken every 510 cycles (15,686 Hz x 510 = 8 MHz). To get one A/D conversion result, each 510 clock cycles the ADC is run in single conversion mode with an ADC clock division by 32. A single conversion takes 14 ADC cycles. Therefore a conversion will be ready after 14 x 32 = 448 cycles.

When a conversion is finished an interrupt occurs. The interrupt routine then performs a loop to fill in the missing 510 - 448 = 62 cycles, before a new A/D conversion is started.

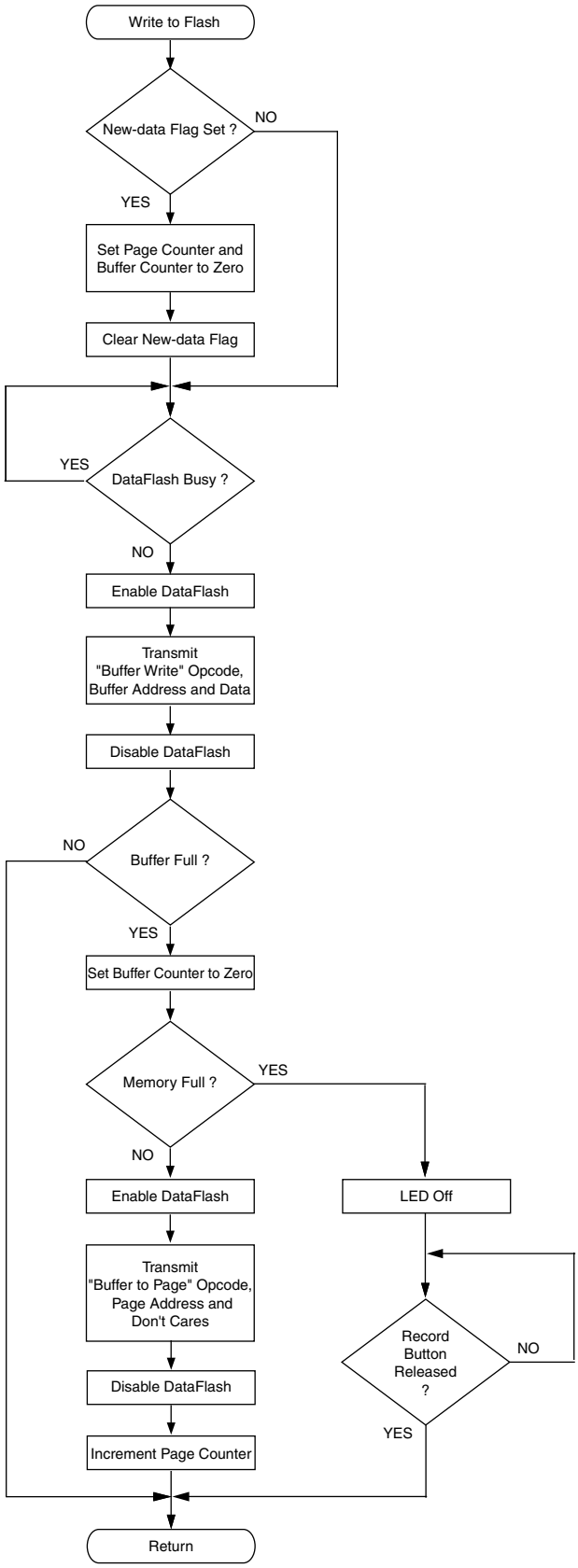
The 10-bit conversion result represents the value at the A/D converter input pin 2 cycles after the conversion has started. These 10 bits cover the range from AGND to AREF, which is 0 to 5V in this application. The microphone circuit output signal, however, is limited to the range of 2.3V to 3.5V. Therefore the 10-bit conversion result is subtracted by a value representing the minimum input voltage. This is 0x1D5 for 2.3V. The part of the data representing signal values above 3.5V is removed by cutting off the two MSBs. This is done automatically when the conversion result is handed over to the "write to flash" subroutine, as its variable "flash_data" is defined as type "char" (8-bit). The final 8-bit data has then to be written to the DataFlash before the next A/D conversion interrupt occurs.

Figure 13. Record



Write to DataFlash

Figure 14. Write to DataFlash



Writing data to the DataFlash is done by writing first to a buffer and when this buffer is full writing it's contents to one page of the main memory.

In the subroutine "write_to_flash" the variable "j" represents the byte number in the buffer and the variable "k" the page number the buffer will be written to. If the new-data flag indicates that the DataFlash is empty, both counters are set to zero.

If the memory already contains some data, the variables indicate the next free location in memory, which ensures that new data is directly appended to the memory contents.

In order to preserve the contents of these variables across two function calls, they are declared as static variables.

To write data to the buffer, the $\overline{CS}$ line is driven low and the opcode 0x84 is loaded into the DataFlash. This is followed by 14 don't care bits and the 10-bit address for the position within the buffer. Then the 8-bit data is entered.

This sequence is transferred to the slave bytewise. After each byte the SPI Status Register – SPSR – is checked until the SPI Interrupt Flag indicates that the serial transfer is complete. After the whole sequence is written the $\overline{CS}$ line is driven high.

If the buffer is full and there are empty pages left, the buffer is copied to the next page of the DataFlash. As the memory has been erased earlier, data can be written without additional erasing.

If the memory is filled, a loop is executed until the "Record" button is released. Any data recorded while the memory is full will be lost.

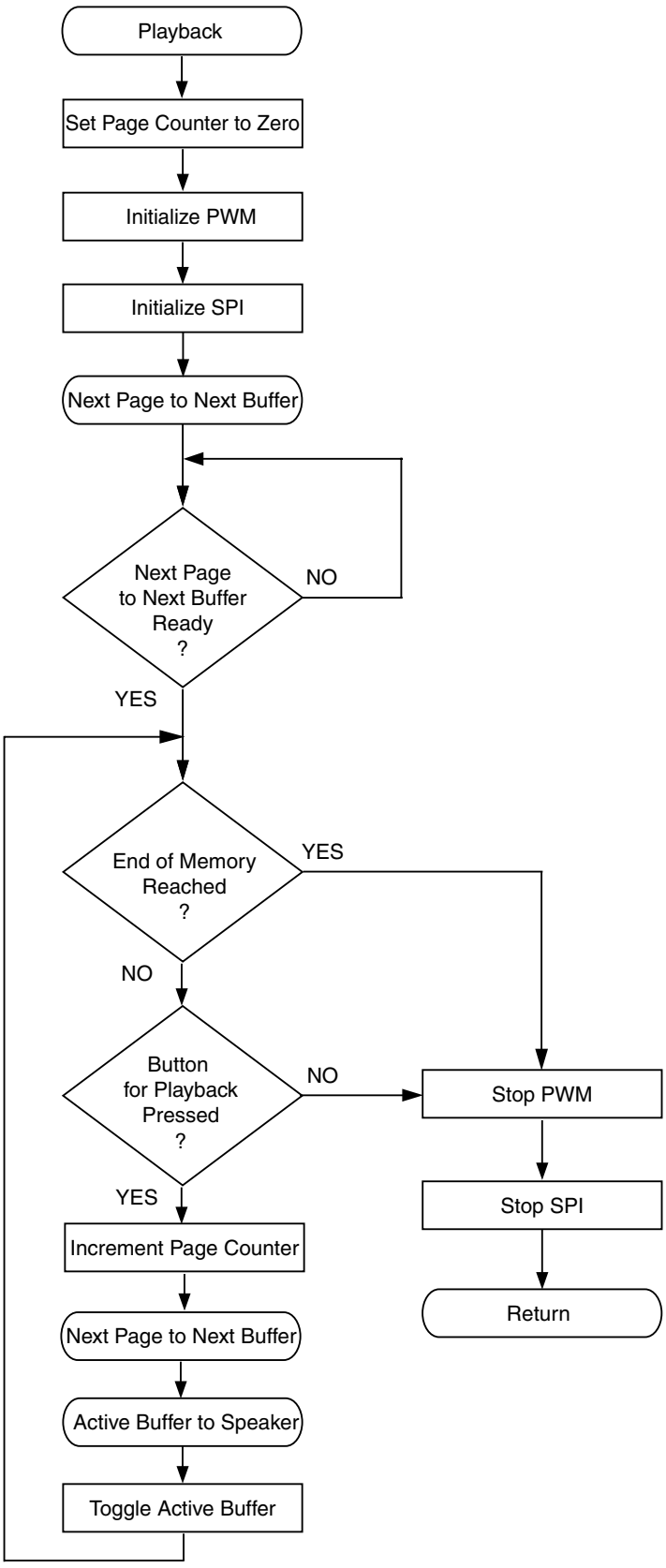
Playback

In the "Playback" subroutine, the contents of the DataFlash are read out and modulated as an 8-bit PWM running at 15,686 Hz. To achieve higher speed, data is not read out directly from the main memory but alternately transferred to one of the two buffers and then read from the buffer. In the meantime the next memory page is copied into the other buffer. For the PWM, the 16-bit Timer/Counter1 is used with the PWM output on OC1B. This is defined in the Timer/Counter Control Registers A and B (TCCRA/TCCRB). For running the PWM at the highest possible frequency, the PWM clock divider is set to 1.

When the set-up is done, the first page is copied into Buffer 1 by driving the $\overline{CS}$ line low and transferring the appropriate commands to the DataFlash. The page-to-buffer transfer is started when the $\overline{CS}$ line is driven high again. When the Ready/Busy pin is driven high by the DataFlash, Buffer 1 contains valid data. Then the next page transfer to Buffer 2 is started. As both buffers are independent from each other, data can already be read from Buffer 1 while the DataFlash is still busy copying data from the second page to Buffer 2.

For reading a byte from a buffer, a dummy value has to be written to the DataFlash. A write action of the master to an SPI slave causes their SPI Data Register – SPDR – to be interchanged. After writing a dummy byte to the DataFlash, the SPDR of the AVR microcontroller contains the output data from the DataFlash.

Figure 15. Playback



When the PWM counter contains the value “0”, a Timer1 overflow interrupt occurs. This interrupt is used to synchronize data output from the DataFlash to the PWM frequency. When a value from the buffer has been shifted to the AVR microcontroller, a loop is performed until the Timer1 overflow interrupt occurs. Then the data is written to the Timer/Counter1 Output Compare Register B (OCR1B), being automatically latched to the PWM output when the PWM counter contains its maximum value (255 for 8-bit PWM).

After the last value of the buffer is read, the active buffer is toggled.

If the entire memory has been played back, all interrupts are disabled and the Timer/Counter1 is stopped.

Figure 16. Next Page to Next Buffer

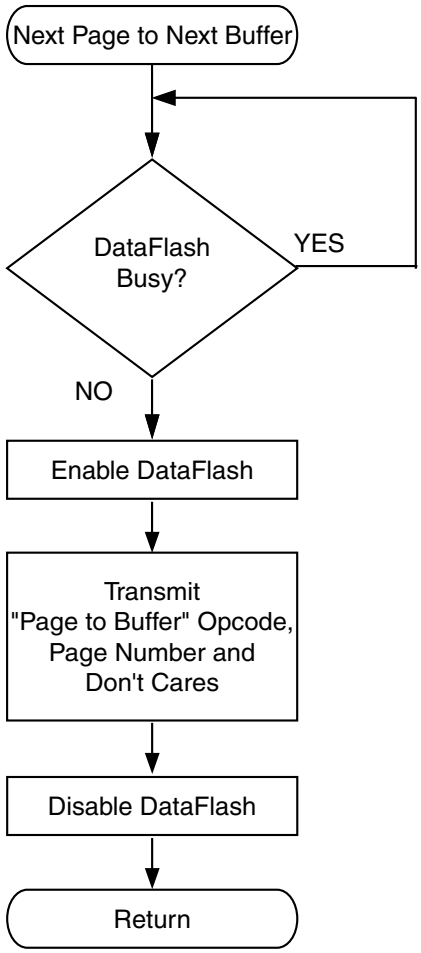
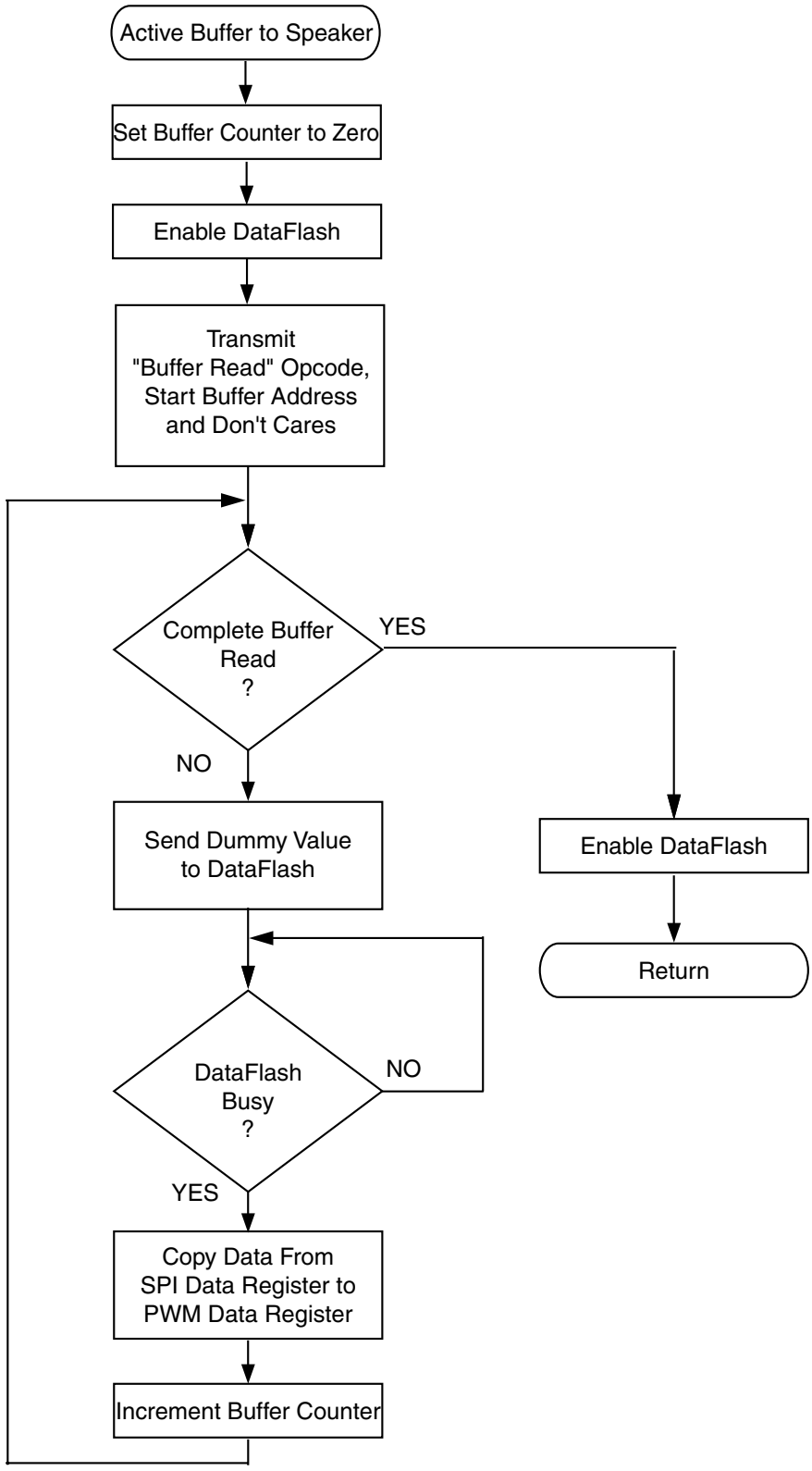


Figure 17. Active Buffer to Speaker



Using the STK200 Development Board

The application described in this note can be tested and modified using the STK200 Development Board. In this case some points have to be noticed.

Chip Socket

This application uses the A/D converter. Therefore the microcontroller has to be placed in the socket labelled “A/D parts” and the microphone amplifier output connected to the header connectors labelled “Analog”.

Jumpers

According to the set-up in the “setup_all” subroutine all jumpers on pins used for other purposes than pushbuttons or the LED have to be removed. For the described application these are on Port B the jumpers 0 to 2 and 4 to 7 and on Port D jumper 4.

SPI Resistors

In order to avoid interference between the on-board SPI and devices connected to the pin headers labelled “Port B”, 10 kΩ resistors are inserted between the chip socket and the Port B headers PB 5 to PB 7. If the DataFlash is going to be connected to these pin headers, the resistors have to be bypassed by soldering a bridge across their connectors on the reverse side of the STK200.

Using the On-board SPI

Short circuiting the resistors between the chip socket and the Port B header connectors may cause some problems if using the on-board SPI for program download and verification, when a device is connected to the Port B header connectors. If problems occur it will help either to disconnect the device during program download and verification, or to solder a 10 kΩ resistor between PB4 and V_{CC} according to Figure 9.

Modification and Optimization

The microphone output signal may vary depending on the type of microphone used. To achieve best results it is important to choose the microphone amplifier gain that delivers a maximum output signal closest to AREF.

Data is written into the DataFlash almost as it is read from the A/D converter. Compressing this data might be possible and useful if a longer recording time or a stereo signal is required.

In this application two ways of implementing a status flag are shown.

One way is to use a global variable (i.e. the “wait” variable used in the “playback” subroutine). The other way is to use an unused bit in a register. In the “erase” subroutine, the ACIS1 bit of the Analog Comparator Control And Status Register (ACSR) is used to indicate that new data has to be stored next. As long as the analog comparator is not used this does not have any negative effects on the program performance, but frees one register from a blocking global variable.

The sampling frequency of 15,686 Hz (respectively 510 clock cycles) is generated by an ADC interrupt and a delay loop. This can be replaced by an independent timer (Timer/Counter0 or Timer/Counter2), if they are not used on other purposes.

References

1. Proakis, J.G. and Manolakis, D.G. (1992)
   Digital Signal Processing: Principles, Algorithms, and Applications
   Second Edition
2. Datasheets:
   Atmel AVR AT90S8535
   Atmel AT45DB161B DataFlash

Resources

Table 1. Peripheral Usage

Peripheral	Description	Interrupts
Timer 1	8-bit PWM	Timer 1 Overflow (PWM Counter at Zero)
3 I/O pins PORT B	SPI to Access DataFlash	
4 I/O pins PORT B	DataFlash Control Lines	
1 I/O pin PORT B	Status LED	
1 I/O pin PORT A	ADC Input	A/D conversion ready
3 I/O pins PORT D	Pushbuttons	
1 I/O pin PORT D	PWM Output	

Bill of Materials

Table 2. Microcontroller and Memory Circuit

Component	Value	Description
R1	10 kΩ	Pull-up Resistor for DataFlash “Chip Select” Line
R2	1 kΩ	LED Resistor
R3	100Ω	Analog Voltage Filter Resistor
LED		Status Indicator
C1, C2	22 pF	Clock Signal Circuit Capacitors
C3	100 nF	Analog Voltage Filter Capacitor
Oscillator Crystal	8 MHz	Clock Signal Generation
DataFlash	AT45DB161B	16M-bit Serial Interface Flash Memory
AVR	AT90S8535	Enhanced RISC Flash Microcontroller

Table 3. Microphone and Speaker Circuit

Component	Value	Description
R1	10 kΩ	Feedback Resistor for Microphone Amplifier
R2	10 kΩ	Offset for Microphone Amplifier
R3	10 kΩ	Offset for Microphone Amplifier
R4	1 kΩ	Microphone Power Resistor
R5	12 kΩ	Microphone RC Filter Resistor
R6	5 kΩ	Chebychev Filter Resistor
R7	1 kΩ	Chebychev Filter Resistor
R8	470Ω	Chebychev Filter Resistor
R9	1 kΩ	Input Resistor for Microphone Amplifier
R10	15 kΩ	Chebychev Filter Resistor

Table 3. Microphone and Speaker Circuit (Continued)

Component	Value	Description
R11	12 kΩ	Earphones RC Filter Resistor
C1	1 µF	AC Coupling for Microphone
C2	1 µF	Chebychev Filter Capacitor
C3	1 µF	AC Coupling for Earphones
C4	22 nF	Earphones RC Filter Capacitor
C5	100 nF	Chebychev Filter Capacitor
C6	100 nF	De-coupling Capacitor
C7	1 nF	Chebychev Filter Capacitor
C8	4.7 nF	Earphones RC Filter Capacitor
C9	2.2 nF	Chebychev Filter Capacitor
U1	LM324	Quad Op-amp
2 standard jack sockets	3.5 mm	
Microphone		Standard PC Microphone with 3.5 mm Connector
Earphones		Standard with 3.5 mm Connector

Sample C Code

```
/* Check the Atmel Web Site for latest version of the code.
/*
Erase all pages if desired.
Write data to buffer 1. If buffer is full, write buffer to page.
Read DataFlash alternating through buffer1 and buffer2 into the data
register.
*/

#include "io8535.h"
#include <ina90.h>
#include "stdlib.h"
#include "dataflash.h"

// prototypes
void setup (void);
void erasing (void);
void recording (void);
void interrupt[ADC_vect] sample_ready (void);
void write_to_flash (unsigned char ad_data);
void playback (void);
void next_page_to_next_buffer (unsigned char active_buffer, unsigned int
page_counter);
void interrupt[TIMER1_OVF1_vect] out_now(void);
void active_buffer_to_speaker (unsigned char active_buffer);

// global variables
volatile unsigned char wait = 0;


void setup(void)
{
    DDRB = 0xBD;         // SPI Port initialisation
                         // SCK, MISO, MOSI,  CS, LED,  WP , RDYBSY, RST
                         // PB7, PB6,  PB5,  PB4, PB3, PB2 ,  PB1,   PB0
                         //  O    I     O     O    O    O      I      O
                         //  1    0     1     1    1    1      0      1

    PORTB = 0xFF;        // all outputs high, inputs have pullups (LED is off)
    DDRA = 0x00;         // define port A as an input
    PORTA = 0x00;
    DDRD = 0x10;         // define port D as an input (D4: output)

    _SEI();              // enable interrupts
}


void erasing(void)
{
    unsigned int block_counter = 0;
    unsigned char temp = 0x80;
```

```
    ACSR |= 0x02;    // set signal flag that new data has to be recorded next

    // interrupt disabled, SPI port enabled, master mode, MSB first,
    // SPI mode 3, Fcl/4
    SPCR = 0x5C;

    while (block_counter < 512)
    {
        PORTB &= ~DF_CHIP_SELECT; // enable DataFlash

        SPDR = BLOCK_ERASE;
        while (!(SPSR & temp));   // wait for data transfer to be completed
        SPDR = (char)(block_counter>>3);
        while (!(SPSR & temp));   // wait for data transfer to be completed
        SPDR = (char)(block_counter<<5);
        while (!(SPSR & temp));   // wait for data transfer to be completed
        SPDR = 0x00;              // don't cares
        while (!(SPSR & temp));   // wait for data transfer to be completed

        PORTB |= DF_CHIP_SELECT;  // disable DataFlash

        block_counter++;
        while(!(PINB & 0x02));    // wait until block is erased
    }
    SPCR = 0x00;                  //disable SPI
}

void recording(void)
{
    // interrupt disabled, SPI port enabled, master mode, MSB first,
    // SPI mode 3, Fcl/4
    SPCR = 0x5C;
    ADMUX = 0x00;                 // A/D converter input pin number = 0
    ADCSR = 0xDD;                 // single A/D conversion, fCK/32,
                                  // conversion now started
    while (!(PIND & 8));          // loop while button for recording (button
                                  // 3) is pressed

    ADCSR = 0x00;                 // disable AD converter
    SPCR = 0x00;                  // disable SPI
}


void interrupt[ADC_vect] sample_ready(void)
{
    unsigned char count = 0;

    while (count < 6) count++;    // wait some cycles
    ADCSR |= 0x40;                // start new A/D conversion
```

```
    write_to_flash(ADC-0x1D5);    // read data, convert to 8 bit
                                  // and store in flash
}

void write_to_flash(unsigned char flash_data)
{
    static unsigned int buffer_counter;
    static unsigned int page_counter;
    unsigned char temp = 0x80;

    if((ACSR & 0x02))             // if flag is set that new data has
                                  // to be written
    {
        buffer_counter = 0;
        page_counter = 0;         // reset the counter if new data
                                  // has to be written
        ACSR &= 0xFD;             // clear the signal flag
    }

    while(!(PINB & 0x02));        // check if flash is busy

    PORTB &= ~DF_CHIP_SELECT;     // enable DataFlash

    SPDR = BUFFER_1_WRITE;
    while (!(SPSR & temp));       // wait for data transfer to be completed
    SPDR = 0x00;                  // don't cares
    while (!(SPSR & temp));       // wait for data transfer to be completed
    SPDR = (char)(buffer_counter>>8);// don't cares plus first two bits
                                     // of buffer address
    while (!(SPSR & temp));       // wait for data transfer to be completed
    SPDR = (char)buffer_counter;  // buffer address (max. 2^8 = 256 pages)
    while (!(SPSR & temp));       // wait for data transfer to be completed
    SPDR = flash_data;            // write data into SPI Data Register
    while (!(SPSR & temp));       // wait for data transfer to be completed

    PORTB |= DF_CHIP_SELECT;      // disable DataFlash

    buffer_counter++;

    if (buffer_counter > 528)     // if buffer full write buffer into
                                  // memory page
    {
        buffer_counter = 0;
        if (page_counter < 4096)  // if memory is not full
        {
            PORTB &= ~DF_CHIP_SELECT;// enable DataFlash

            SPDR = B1_TO_MM_PAGE_PROG_WITHOUT_ERASE;// write data from
                                                    //buffer1 to page
            while (!(SPSR & temp)); // wait for data transfer to be completed
            SPDR = (char)(page_counter>>6);
```

```
            while (!(SPSR & temp));// wait for data transfer to be completed
            SPDR = (char)(page_counter<<2);
            while (!(SPSR & temp)); // wait for data transfer to be completed
            SPDR = 0x00;            // don't cares
            while (!(SPSR & temp)); // wait for data transfer to be completed

            PORTB |= DF_CHIP_SELECT;// disable DataFlash

            page_counter++;
        }
        else
        {
            PORTB |= 0x08;          // turn LED off
            while (!(PIND & 8));    // wait until button for recording
                                    // (button 3) is released
        }
    }
}


void playback(void)
{
    unsigned int page_counter = 0;
    unsigned int buffer_counter = 0;
    unsigned char active_buffer = 1;// active buffer = buffer1
    unsigned char temp = 0x80;

    TCCR1A = 0x21;                  // 8 bit PWM, using COM1B
    TCNT1 = 0x00;                   // set counter1 to zero
    TIFR = 0x04;                    // clear counter1 overflow flag
    TIMSK = 0x04;                   // enable counter1 overflow interrupt
    TCCR1B = 0x01;                  // counter1 clock prescale = 1
    OCR1B = 0x00;                   // set output compare register B to zero

    // interrupt disabled, SPI port enabled, master mode, MSB first,
    // SPI mode 3, Fcl/4
    SPCR = 0x5C;

    // read page0 to buffer1
    next_page_to_next_buffer (active_buffer, page_counter);

    while (!(PINB & 0x02));         // wait until page0 to buffer1
                                    // transaction is finished

    while ((page_counter < 4095)&(!(PIND & 2)))// while button for playback
                                               // (button 1) is pressed
    {
        page_counter++;             // now take next page

        next_page_to_next_buffer (active_buffer, page_counter);
        active_buffer_to_speaker (active_buffer);
```

```
        if (active_buffer == 1) // if buffer1 is the active buffer
        {
        active_buffer++;        // set buffer2 as active buffer
        }
        else                    // else
        {
        active_buffer--;        // set buffer1 as active buffer
        }
    }
    TIMSK = 0x00;               // disable all interrupts
    TCCR1B = 0x00;              // stop counter1
    SPCR = 0x00;                // disable SPI
}


void next_page_to_next_buffer (unsigned char active_buffer, unsigned int
page_counter)
{
    unsigned char temp = 0x80;

    while(!(PINB & 0x02));      // wait until flash is not busy

    PORTB &= ~DF_CHIP_SELECT;   // enable DataFlash

    if (active_buffer == 1)     // if buffer1 is the active buffer
    {
        SPDR = MM_PAGE_TO_B2_XFER; // transfer next page to buffer2
    }
    else                        // else
    {
        SPDR = MM_PAGE_TO_B1_XFER; // transfer next page to buffer1
    }
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = (char)(page_counter >> 6);
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = (char)(page_counter << 2);
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = 0x00;                // write don't care byte
    while (!(SPSR & temp));     // wait for data transfer to be completed
    PORTB |= DF_CHIP_SELECT;    // disable DataFlash and start transaction
}


void interrupt[TIMER1_OVF1_vect] out_now(void)
{
   wait = 0;                    // an interrupt has occured
}
```

```
void active_buffer_to_speaker (unsigned char active_buffer)
{
    // until active buffer not empty read active buffer to speaker

    unsigned int buffer_counter = 0;
    unsigned char temp = 0x80;

    PORTB &= ~DF_CHIP_SELECT;   // enable DataFlash

    if (active_buffer == 1)     // if buffer1 is the active buffer
    {
        SPDR = BUFFER_1_READ;   // read from buffer1
    }
    else                        // else
    {
        SPDR = BUFFER_2_READ;   // read from buffer2
    }
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = 0x00;                // write don't care byte
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = 0x00;                // write don't care byte
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = 0x00;                // start at buffer address 0
    while (!(SPSR & temp));     // wait for data transfer to be completed
    SPDR = 0x00;                // write don't care byte
    while (!(SPSR & temp));     // wait for data transfer to be completed

    while (buffer_counter < 528)
    {
        SPDR = 0xFF;            // write dummy value to start register shift
        while (!(SPSR & temp)); // wait for data transfer to be completed
        while(wait);            // wait for timer1 overflow interrupt
        OCR1B = SPDR;           // play data from shift register
        wait = 1;               // clear the signal flag
        buffer_counter++;
     }
     PORTB |= DF_CHIP_SELECT;   // disable DataFlash
}


void main(void)
{
    setup();

    for(;;)
    {
        if (!(PIND & 8))        // if button for recording (button 3)
                                // is pressed
        {
            PORTB &= 0xF7;      // turn LED on
```

```
            recording();
        }
        if (!(PIND & 4))        // if button for erasing (button 2)
                                // is pressed
        {
            PORTB &= 0xF7;      // turn LED on
            erasing();
            while (!(PIND & 4));// wait until button for erasing (button 2)
                                // is released
        }
        if (!(PIND & 2))        // if button for playback (button 1)
                                // is pressed
        {
            PORTB &= 0xF7;      // turn LED on
            playback();
            while (!(PIND & 2));// wait until button for playback (button 1)
                                // is released
        }
    PORTB |= 0x08;              // turn LED off while running idle
    }
}
```

DataFlash.h

```
// changed on 19.04.1999
// for use with the 8535

#include "ina90.h"
#pragma language=extended

// DataFlash reset port pin (PB 0)
#define DF_RESET 0x01

// DataFlash ready/busy status port pin (PB 1)
#define DF_RDY_BUSY 0x02

// DataFlash boot sector write protection (PB 2)
#define DF_WRITE_PROTECT 0x04

// DataFlash chip select port pin (PB 4)
#define DF_CHIP_SELECT 0x10

// buffer 1
#define BUFFER_1 0x00

// buffer 2
#define BUFFER_2 0x01

// defines for all opcodes

// buffer 1 write
```

```
#define BUFFER_1_WRITE 0x84

// buffer 2 write
#define BUFFER_2_WRITE 0x87

// buffer 1 read (change to 0xD4 for SPI mode 0,3)
#define BUFFER_1_READ 0x54

// buffer 2 read (change to 0xD6 for SPI mode 0,3)
#define BUFFER_2_READ 0x56

// buffer 1 to main memory page program with built-in erase
#define B1_TO_MM_PAGE_PROG_WITH_ERASE 0x83

// buffer 2 to main memory page program with built-in erase
#define B2_TO_MM_PAGE_PROG_WITH_ERASE 0x86

// buffer 1 to main memory page program without built-in erase
#define B1_TO_MM_PAGE_PROG_WITHOUT_ERASE 0x88

// buffer 2 to main memory page program without built-in erase
#define B2_TO_MM_PAGE_PROG_WITHOUT_ERASE 0x89

// main memory page program through buffer 1
#define MM_PAGE_PROG_THROUGH_B1 0x82

// main memory page program through buffer 2
#define MM_PAGE_PROG_THROUGH_B2 0x85

// auto page rewrite through buffer 1
#define AUTO_PAGE_REWRITE_THROUGH_B1 0x58

// auto page rewrite through buffer 2
#define AUTO_PAGE_REWRITE_THROUGH_B2 0x59

// main memory page compare to buffer 1
#define MM_PAGE_TO_B1_COMP 0x60

// main memory page compare to buffer 2
#define MM_PAGE_TO_B2_COMP 0x61

// main memory page to buffer 1 transfer
#define MM_PAGE_TO_B1_XFER 0x53

// main memory page to buffer 2 transfer
#define MM_PAGE_TO_B2_XFER 0x55

// DataFlash status register for reading density, compare status,
// and ready/busy status (change to 0xD7 for SPI mode 0,3)
#define STATUS_REGISTER 0x57
```

```
// main memory page read (change to 0xD2 for SPI mode 0,3)
#define MAIN_MEMORY_PAGE_READ 0x52

// erase a 528 byte page
#define PAGE_ERASE 0x81

// erase 512 pages
#define BLOCK_ERASE 0x50

#define TRUE                0xff
#define FALSE               0x00
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory
Atmel Corporate
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 436-4270
FAX 1(408) 436-4314

Microcontrollers
Atmel Corporate
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 436-4270
FAX 1(408) 436-4314

Atmel Nantes
La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

Atmel Colorado Springs
1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Atmel Smart Card ICs
Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Atmel Heilbronn
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

Atmel Colorado Springs
1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

1456B–AVR–03/02 0M