AVR222: 8-Point Moving Average Filter

Features

- **31-Word Subroutine Filters Data Arrays up to 256 Bytes**
- **Runable Demo Program**

Introduction

The moving average filter is a simple Low Pass FIR (Finite Impulse Response) filter commonly used for smoothening an array of sampled data. This application implements an 8-point filter to simplify the average calculation. The application note gives an excellent demonstration of how the powerful addressing modes in the AVR architecture can be utilized.

Theory

The moving average filter can be imagined as a window of a certain size (in this case 8) moving along the array, one element at a time. The middle element of the window (in this case element #4) is replaced with the average of all elements in the window. See Figure 1: The 8-Point Averaging Window. It is however important to remember the value of new elements and not make the replacement until the window has passed. This must be done since all averages shall be based on the original data in the array.

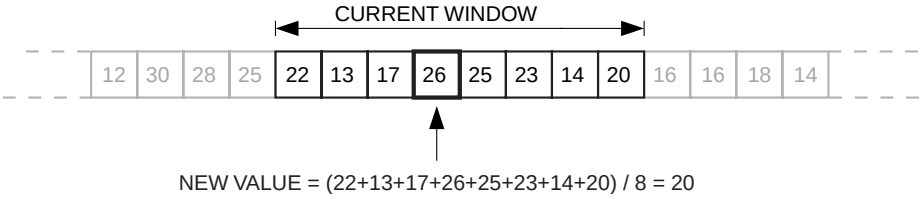

Figure 1. The 8-Point Averaging Window

When the ends of the array is filtered and parts of the window is outside the array, the averaging must be done on less elements than when the entire window is inside the array. This implementation leaves the ends of the array unfiltered to save code. For an 8-point filter, this means that when n elements are filtered, elements 1, 2, 3, and n-3, n-2, n-1, n remain unchanged when filtering is complete. For many applications, this is no problem.

Implementation

The application defines an 8-byte ring buffer (R0-R7) which always holds the data in the current averaging window. The filter routine calculates the sum of the window and computes the average, which is stored back in the array. The AVR's three pointers are assigned the following functions:

- Z points to the array element to be replaced
- Y points inside the ring buffer when the sum of the buffer contents is calculated in a program loop.
- X is the ring pointer which holds the position of new values to the buffer.

8-Bit AVR Microcontroller with Downloadable Flash

Application Note

0940A-A–8/97

Usage

To filter an array in SRAM, use the following procedure:

1. Load ZH with the high address of the first element in the array
2. Load ZL with the low address of the first element in the array.
3. Load the register variable “t_size” with the number of elements in the table.
4. Call “mav8”.

Algorithm Description

The following procedure describes how the sorter is implemented on the AVR:

Initialization

1. Clear the X and Y pointers (point to R0).

Fill Ring Buffer Initially:

2. Get the SRAM contents at Z and increment Z.
3. Store in register at Y and increment Y.
4. If Y not 8, goto Step 2.

Find Average

5. Clear the 16-bit register variable “AH:AL” (Average Value)
6. Clear YL (point to R0).
7. Get the register contents at Y.
8. Add to “AH:AL”.
9. If Y not 8, goto Step 8.
10. Divide “AH:AL” by 8

Write Back Average and Get Next Value to Buffer

11. Get SRAM contents at Z+5 (Next value to buffer)
12. Store to register at X and increment X.
13. Clear the highest 5 bits of XL to make it point to the start of the buffer if the end is passed.
14. Store AL at Z and increment Z.
15. Decrement “t_size”
16. If “t_size” is not zero (end of array is reached) goto Step 5.

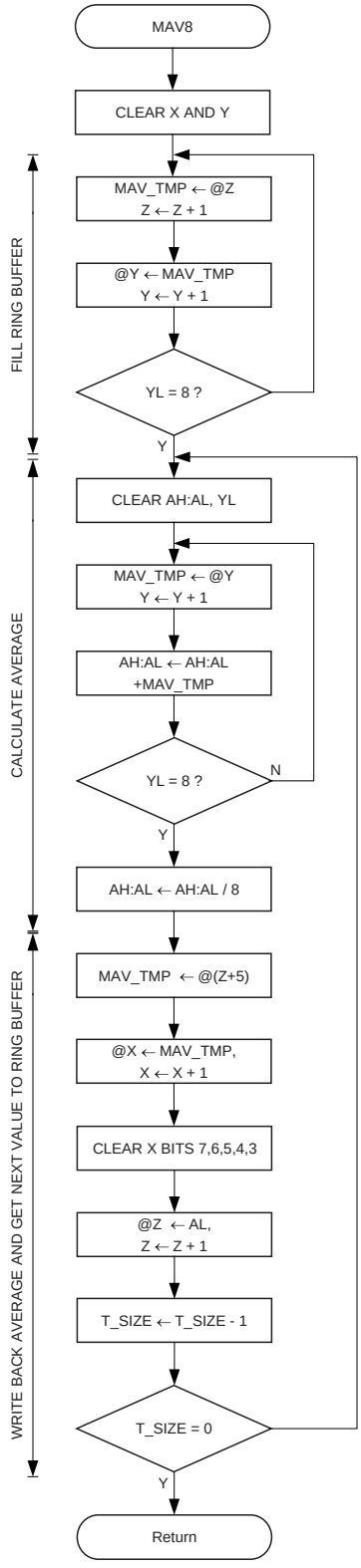

Figure 2. “mav8” Flow Chart

Performance

Table 1. "mav8" Register Usage

Register	Input	Internal	Output
R0-R7		Ring buffer	
R8		"mav_tmp" - temporary storage	
R9		"AL" - average low byte	
R10		"AH" - average high byte	
R16	"t_size" -number of elements	"t_size" - loop counter	
R26		XL	
R27		XH	
R28		YL	
R29		YH	
R30	Z - address of first element	ZL	
R31	Z - address of first element	ZH	

Table 2. "mav8" Performance Figures

Parameter	Value
Code Size (Words)	30 + return
Execution Time (Cycles)	59 + 75 x (SIZE - 7) + return
Register Usage	• Low registers :11 • High registers :1 • Pointers :X, Y, Z
Interrupts Usage	None
Peripherals Usage	None

Note: SIZE = Number of bytes to filter

Test/Example Program

"avr222.asm" contains a test program which copies 60 bytes of random data from the program memory to SRAM and calls "mav8" to filter the data. The test program is well suited for running under the AVR Studio.