Interfacing The Am9511 Arithmetic Processing Unit

Marvin L. De Jong
Department of Mathematics-Physics
The School of the Ozarks
Pt. Lookout, MO 65726

Introduction

If you are interested in a hardware solution to the problem of addition, subtraction, multiplication, division, and functions such as sine, cosine, tangent, square root, exponential, logarithm and their inverse functions, then the Am9511 integrated circuit will be of interest to you. The Am9511 Arithmetic Processing Unit is a product of Advanced Micro Devices Inc., 901 Thompson Place, Sunnyvale, CA 94086. It performs signed multiplication, addition, subtraction and division with either 16-bit integers or 32-bit integers, in twos complement form. It also does these operations and evaluates a variety of functions (mentioned above) in a 32-bit floating point form. In the floating point form, the mantissa of the number is represented by 24 bits (equivalent to approximately seven significant decimal digits). The exponent is represented by six bits and a sign bit, giving a range of numbers that can be represented from roughly 10^{-19} to 10^{+19}. The one bit not accounted for so far is the sign of the mantissa. Thus, the Am9511 should satisfy most of the calculating needs of microcomputer users. It is important to point out that the Am9511 is a *binary* device as opposed to a *BCD* device. If you intend to use it like a calculator, then appropriate BCD-to-binary and binary-to-BCD routines will be needed to input and output numbers.

Timing of the various control pins on the Am9511 is one of the most important considerations in constructing an interface between it at the microprocessor. The timing requirements seem to be more relaxed in the most recent specification sheets, but my original specifications were quite complex. Perhaps it would be easy to interface the Am9511 somewhere in the address space, using address lines and control lines to operate it. However, given the complexities of the original timing diagrams, we used an interface adapter (the 6522, although any of the other popular interface adapters such as the 6530 can also be used with our programs). One port is used for data transfers, while several pins of the other port on the interface adapter is used to control the Am9511. These techniques produce an extremely simple interface at the expense of some overhead in software.

Before proceeding to the details of the circuit and the driver programs it should be pointed out that if you are interested in building and using this or some other circuit that uses the Am9511, you will want to get complete specification sheets, a publication called "Algorithm Details for the Am9511 Arithmetic Processing Unit," and a card-type Am9511 reference card. All three of these publications are available from Advanced Micro Devices. The Am9511 itself costs about $200, a number which may cause you to turn to the next article. A few mail order houses such as Advanced Computer Products are beginning to list the chip in their advertisements. Be sure to request all the literature mentioned above because you will need it to know how to use the chip. Space does not permit us to write a complete description of all the features of the chip.

The Am9511 Interface Circuit

The interface circuit is given in Figure 1. It is very simple because the complexity is absorbed in the software that must accompany this circuit. As noted, any 6502 system such as the SUPERKIM, KIM-1, AIM 65, etc., may be used, and any two-port interface adapter can be used. Be sure to include the 0.01 microfarad bypass capacitors, keep the leads between the Am9511 and the microcomputer short, and tie the unused control inputs ($\overline{\text{EACK}}$ and $\overline{\text{SVACK}}$) to logic one as shown in Figure 1. I will not reveal how many hours of grief the failure to follow these standard procedures cost me. Keep it simple, neat, and don't try any shortcuts. Also follow the usual procedures in handling integrated circuits that are susceptible to damage by static discharge. This is not your typical El Cheapo IC: $200 makes it irreplaceable. Avoid any Benjamin Franklin type experiments.

The Driver Subroutines

Listing 1 gives five subroutines that work with the interface circuit in Figure 1 to operate the Am9511. The subroutines are:

1. **RESET** - A subroutine that is used to reset the Am9511 either after power is applied or to clear the Am9511 to a known condition. This subroutine must be called after power-up and before using the Am9511.
2. **WRITE** - This subroutine transfers a byte of data in the accumulator of the 6502 to the stack of the Am9511.
3. **COMMAND** - A subroutine that transfers an eight-bit command word from the accumulator

www.commodore.ca

of the 6502 to the command register of the Am9511.

4. **READ** - Subroutine READ takes one byte of data (part of the answer) from the stack of the Am9511 and returns it to the X - register in the 6502.

5. **STATUS** - This subroutine reads the status register of the Am9511 and transfers its contents to the X - register in the 6502.

The comments in the various subroutines should be studied in connection with the Am9511 specification sheets to understand the functions of the various instructions. We only note here that each of the access subroutines, WRITE, COMMAND, READ, and STATUS, wait for the Am9511 to signal that an operation is complete when its $\overline{\text{PAUSE}}$ pin returns to logic one.

We will describe a few operations with the Am9511 to illustrate how the subroutines work. Refer to the literature mentioned previously for more details on the stack operation. The Am9511 stack may be regarded either as an eight-level, 16-bit wide stack, or as a four-level, 32-bit wide stack. Writing once to the Am9511 places an 8-bit word on the stack. However, since all of the "words" operated on by the Am9511 are either 16 bits or 32 bits wide, you must write at least 16 data bits (two bytes) to fill a 16-bit stack location. You must write four bytes to fill a 32-bit stack location. The *last* level filled (either 16 bits or 32 bits wide) is called TOS (acronym for top of stack). The level filled *previously* is referred to as NOS (next on stack).

An example will clarify the operation of the stack. Suppose we wish to add two 16-bit integers (they must be in twos complement form). Using the WRITE subroutine, we write the least-significant byte of one of the numbers to the Am9511 stack. Call this byte B1. Next we write B2, the most-significant byte of the same integer, to the Am9511. This puts a 16-bit integer onto TOS, the top level of the stack. The other addend, call it A1 and A2 for the least-significant and most-significant bytes respectively, is placed on the TOS by calling subroutine WRITE two more times. Now number B (B1 and B2) is in NOS and A (A1 and A2) is in TOS. The command code for a 16-bit addition, $6C, is now placed in the 6502 accumulator and subroutine COMMAND is called. The Am9511 adds TOS to NOS and puts the result into TOS. The result R, consisting of the most-significant byte R1 and the least-significant byte R2 of the 16-bit answer, is obtained by calling subroutine READ. The first call of READ retrieves the most-significant byte R2, and the second call of READ retrieves the least-significant byte of the result R. The status register can be read to see if the addition produced a carry or an overflow.

Subtraction follows exactly the same pattern. The minuend M is loaded on the stack, followed by the subtrahend S to obtain the difference D where D = M - S. After M and S are loaded on the stack, the subtraction command ($2D for a 32-bit word) will result in the difference D in TOS. Calling subroutine READ (twice for a 16-bit integer, four times for a 32-bit integer) gives the answer in the order from most-significant byte to least-significant byte. In division, the dividend is loaded on the stack followed by the divisor, and the quotient is read after the operation is completed. Some of you will recognize that the Am9511 uses RPN.

A program to illustrate these 16-bit operations is given in Listing 2. Suppose we wish to subtract $32FC from $FF5B. We would load $5B into location $0004, $FF into location $0003, $FC into location $0002, and $32 would be loaded into location $0001. The 16-bit subtraction command for the Am9511, $6D, would be loaded into location $0000. The program in Listing 2 will call the appropriate subroutines and place the answer in locations $00FF (most-significant byte) and $00FE (least-significant byte). This program can be used to test many of the operations of the Am9511, including sine, cosine, etc., by loading a 32-bit number (fixed or floating-point representation) on the stack, and then placing a command on the stack. It is a nice simple test program, but remember that many of the Am9511 functions require that the argument is in floating point form, so to find the square root of four requires that you convert four to a floating-point number. The Am9511 will do this if you either cannot or will not.

A word about execution time may be useful at this point. Instructions take from 16 clock cycles for a 16-bit integer addition to several thousand clock cycles for functions like sine, cosine, etc. We operated our Am9511 at 1MHz, but it can be operated at 2MHz and other versions go as high as 4MHz. Clearly the subroutines in Listing 1 require a significant amount of overhead for the simple integer operations, but become insignificant in terms of time overhead when the complex functions are called. Perhaps some reader will design an interface where instructions like STA DATA, STA COMMAND, LDA DATA, and LDA STATUS can be used instead of the subroutines. The difficulty is in working out the necessary timing requirements for the READ and WRITE operations of the 6502. The Am9511 timing seems to be more closely related to 8080A systems than either 6502 systems or 6800 systems.

Our final illustrative program is one that was designed to generate a sine table consisting of one cycle of a sine wave residing in one page of memory. The amplitude of the sine wave is $7F00, in other words, we found $7F00*Sin[Y*(Pi/128] where Y is a number that varied from $00 to $FF (0 to 255). This result was converted to a 16-bit fixed point format, and the most-significant byte was stored in a table in page $0E, while the least-significant byte was stored in a table in page $0F. Note that the result will be in twos complement form, so at location $0E80 in the

www.commodore.ca

table when we are exactly half-way through the sine wave, you will find $00, but at location $0E81 you will find the first negative value of the sine wave and it is $FC, the one in the most-significant bit of the 16-bit result indicating a minus number.

What do you do with a sine wave table? You could read it out to a D/A converter at various rates and play a tune, or you could add a series of sine waves to make a more complex sound. My purpose was to test the AM9511 and in the future I will use the sine wave table as part of a fast-Fourier transform program (I hope). Instead of synthesizing music I would really like to synthesize $20 bills. Let me know if you succeed.

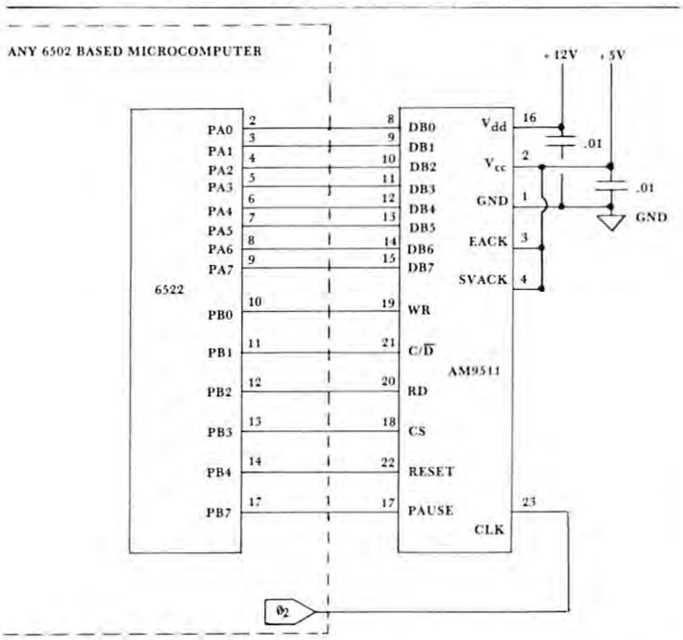

Figure 1.
Interfacing the AM9511 Arithmetic Processing Unit to a 6522 VIA Chip. Other interface adapters that may be used include the 6520, the 6530 and the 6532. No special handshaking pins are used.

Listing 1 Subroutines to drive the AM9511

0300 A9 1F	**RESET**	**LDA $1F**	Make PB0 - PB4
0302 8D 02 A0		**STA PBDD**	output pins to control the AM9511.
0305 A9 0F		**LDA $0F**	RESET pin to
0307 8D 00 A0		**STA PBD**	logic zero.
030A A9 1F		**LDA $1F**	Hold RESET high
030C 8D 00 A0		**STA PBD**	for at least five
030F EA		**NOP**	clock cycles.
0310 EA		**NOP**	
0311 A9 0F		**LDA $0F**	Bring RESET pin
0313 8D 00 A0		**STA PBD**	to logic zero to run the AM9511.
0316 60		**RTS**	Return to the calling program.
	**************		
0320 8D 01 A0	**WRITE**	**STA PAD**	A contains the
0323 A9 04		**LDA $04**	byte to be written
0325 8D 00 A0		**STA PBD**	to the AM9511 (A = accumulator) CS low, C/D low, WR low.
0328 AD 00 A0	**WAIT**	**LDA PBD**	Read PBD to see if PAUSE pin is at
032B 10 FB		**BPL WAIT**	logic zero (no data transfer allowed).
032D A9 FF		**LDA $FF**	If PAUSE is high,
032F 8D 03 A0		**STA PADD**	make PAD an output port to transfer data to the AM9511.
0332 EE 00 A0		**INC PBD**	Bring WR high to complete data transfer.
0335 A9 0F		**LDA $0F**	Next bring CS, C/D high.
0337 8D 00 A0		**STA PBD**	
033A A9 00		**LDA $00**	Now make Port A
033C 8D 03 A0		**STA PADD**	(PAD) an input port again.
033F 60		**RTS**	Return to the calling program.
	**************		
0340 8D 01 A0	**COMMAND**	**STA PAD**	A contains the command for the AM9511.
0343 A9 06		**LDA $06**	CS low, C/D
0345 8D 00 A0		**STA PBD**	high, WR low.
0348 AD 00 A0	**LOAF**	**LDA PBD**	Is PAUSE low?
034B 10 FB		**BPL LOAF**	Yes, then wait until it goes high.
034D A9 FF		**LDA $FF**	Make Port A an
034F 8D 03 A0		**STA PADD**	output port.
0352 EE 00 A0		**INC PBD**	Bring WR high.
0355 A9 0F		**LDA $0F**	Bring other control pins high.
0357 8D 00 A0		**STA PBD**	
035A A9 00		**LDA $00**	Return Port A to
035C 8D 03 A0		**STA PADD**	input status.
035F 60		**RTS**	
0360 A9 01	**READ**	**LDA $01**	CS low, C/D low,
0362 8D 00 A0		**STA PBD**	RD low.
0365 AD 00 A0	**LOITER**	**LDA PBD**	Read PBD to see if PAUSE is low.
0368 10 FB		**BPL LOITER**	If it is, then wait
036A AE 01 A0		**LDX PAD**	until it goes high. Am9511 output to X register.
036D A9 0F		**LDA $0F**	Bring control pins
036F 8D 00 A0		**STA PBD**	high.
0372 60		**RTS**	Return to calling program with output in X.
	**************		
0380 A9 03	**STATUS**	**LDA $03**	CS low, C/D
0382 8D 00 A0		**STA PBD**	high, RD low.
0385 AD 00 A0	**DELAY**	**LDA PBD**	Is PAUSE low?
0388 10 FB		**BPL DELAY**	Yes, then wait until it goes high.
038A AE 01 A0		**LDX PAD**	Read status register of AM9511
038D A9 0F		**LDA $0F**	and keep it in the X register.
038F 8D 00 A0		**STA PBD**	Bring control pins high.
0392 60		**RTS**	Status is in X upon return.

www.commodore.ca

Listing 2 Program that loads four bytes (32 bits) and a command into the Am9511

```
0400 20 00 03  START   JSR RESET      Reset the AM9511
                                      to start using it.
0403 A2 03             LDX #03        Initialize X to
                                      count four bytes.
0405 B5 01     LOOP    LDA DATA,X     Get byte from the
                                      data table.
0407 20 20 03          JSR WRITE      Write the byte in-
                                      to
                                      the Am9511.
040A CA                DEX            Decrement byte
                                      counter.
040B 10 F8             BPL LOOP       Loop until four
                                      bytes are written.
040D A5 00             LDA CMND       Get command
                                      byte from location
                                      $0000.
040F 20 40 03          JSR            Write command
                       COMMAND        to the AM9511.
0412 20 60 03          JSR READ       Get MSB of 16-
                                      bit answer.
0415 86 FF             STX MSB        Put most-signifi-
                                      cant byte here.
0417 20 60 03          JSR READ       Get LSB of 16-
                                      bit answer.
041A 86 FE             STX LSB        Put least-signifi-
                                      cant byte in
                                      $00FE.
041C 00                BRK            End sample pro-
                                      gram here.
```

Listing 3. Sine table generator.

```
0500 20 00 03  SINE    JSR RESET      Reset the
                                      Am9511.
0503 A9 1A             LDA $1A        Push Pi
0505 20 40 03          JSR            (3.14159...) on
                       COMMAND        TOS by writing
                                      $1A to
                                      Am9511.
0508 A9 80             LDA $80        Load 128 =
050A 20 20 03          JSR WRITE      $0080 on TOS,
050D A9 00             LDA $00        Pi is pushed
050F 20 20 03          JSR WRITE      down to NOS.
0512 A9 1D             LDA $1D        Convert 128 =
0514 20 40 03          JSR            $0080 from
                       COMMAND        fixed point to
                                      to floating
                                      point form.
0517 A9 13             LDA $13        Divide NOS by
0519 20 40 03          JSR            TOS (Pi/128),
                       COMMAND        result onto
                                      TOS.
051C A0 00             LDY $00        Y serves as
                                      counter for 256
                                      points.
051E A9 37     REPEAT  LDA $37        Duplicate NOS
0520 20 40 03          JSR            with TOS.
                       COMMAND        Pi/128 is now
                                      in TOS and
                                      NOS.
0523 98                TYA            Duplicate Y in
                                      accumulator.
0524 20 20 03          JSR WRITE      Push down
                                      TOS.
0527 A9 00             LDA $00        stack, Y into
0529 20 20 03          JSR WRITE      TOS.
052C A9 1D             LDA $1D        Change Y into
052E 20 40 03          JSR            floating point
                       COMMAND        form.
0531 A9 12             LDA $12        Multiply to get
0533 20 40 03          JSR            Y*(Pi/128).
                       COMMAND        Result to NOS.
                                      Pop stack up.
0536 A9 02             LDA $02        Take SIN[Y*
0538 20 40 03          JSR            (Pi/128)], result
                       COMMAND        to TOS.
053B A9 00             LDA $00        Push $7F00 on
053D 20 20 03          JSR WRITE      stack.
0540 A9 7F             LDA $7F
0542 20 20 03          JSR WRITE
0545 A9 1D             LDA $1D        Convert $7F00
0547 20 40 03          JSR             = 32512 to
                       COMMAND        floating point
                                      form.
054A A9 12             LDA $12        Find 32512*
054C 20 40 03          JSR            SIN[Y*(Pi/
                       COMMAND        128)], result to
                                      NOS, pop
                                      stack up.
054F A9 1F             LDA $1F        Convert that
0551 20 40 03          JSR            number to
                       COMMAND        fixed point
                                      format.
0554 20 60 03          JSR READ       Get MSB of
0557 8A                TXA            16-bit result in
                                      X register.
0558 99 00 0E          STA MSB,Y      Store it in a
                                      table in page
                                      $0E.
055B 20 60 03          JSR READ       Get LSB of 16-
055E 8A                TXA            bit result.
055F 99 00 0F          STA LSB,Y      Store it in a
                                      table in page
                                      $0F.
0562 C8                INY            Increment Y
                                      counter.
0563 D0 B9             BNE REPEAT     Repeat until
                                      table is filled.
0565 00                BRK            Break to the
                                      monitor.
```

©

www.commodore.ca