THE SBC GAZETTE

An Efficient A/D Interface

Richard Olivo

We use AIM-65's for the acquisition and storage of transient analog data. Once digitized and stored, the data are played back to oscilloscopes or to chart recorders for permanent copies. The interface requirements are fairly demanding in terms of speed (a sample is taken every 100 microseconds) and sensitivity (the input signal is a bipolar voltage of about 50 millivolts). I have designed an efficient, general purpose analog/digital interface which connects to the user 6522 versatile interface adapter on the AIM, and could be used without modification with other 6522-based parallel ports.

It is designed around two relatively inexpensive, high-performance integrated circuits from Analog Devices, Inc. (Box 280, Norwood, MA 02062). I chose these devices for their ease of use (they require no external parts), their speed, and their moderate cost. Analog-to-digital conversion is performed by the AD570 ($29), an 8-bit device with an input range of +5 to -5 volts and a 25-microsecond conversion time. Digital-to-analog output is provided by the 8-bit AD558 ($11), which has an output range of 0 to 10 volts and which requires about 2 microseconds to produce a full-scale change in the output voltage. The data lines of these two devices are connected to the 6522's ports A and B, with control lines CA1 and CA2 also used. Control lines CB1 and CB2 are free; we use them to receive trigger pulses (CB1) and to output sync pulses (CB2). I added circuitry to protect CB1 and CB2 against inadvertent application of non-TTL voltages, but this protection could be omitted. The interface also provides preamplification of the input with variable gain from 1 to 1000 (in 1,2,5 steps), a zero-offset adjustment, and a choice between direct- or capacitor-coupling for the input. The output connector is switchable between the preamplifier's output and the output of the digital-to-analog converter. The full interface circuitry is shown in Figure 1.

Although programs for data collection can become lengthy, the actual routines for analog input and output are brief. Output through the digital-to-analog converter is extremely simple, since the device is connected in its transparent (unlatched) mode. In this mode, a byte written to Port B appears immediately as a steady analog voltage at the converter. The AD570 input converter, on the other hand, requires control lines to start conversion and to detect that data are ready. In my circuit, control line CA2 initiates conversion by supplying a TTL pulse, which must last at least two microseconds. Conversion begins on the falling phase of the pulse, which also blanks the AD570's data lines. After 25 microseconds, the end of conversion and the appearance of new data are signalled by the $\overline{DR}$ line (pin 17 of the AD570, connected to control line CA1); this line goes low and remains low until the next conversion is initiated. Thus, a routine to acquire a byte of analog data must accomplish four tasks: it must initiate conversion by making CA2 go high and then low, it must update a storage vector to be used to place the new data in memory, it must examine CA1 to check if the new data are ready, and finally it must read Port A and store the data in memory. If the storage buffer is more than one page of memory (as in my system), the minimum practical sampling interval approaches 100 microseconds, even though the converter itself requires only 25 microseconds. The remainder of the time is taken up by the software.

In my program, data sampling is interrupt driven. Timer T1 on the AIM's 6522 versatile interface adapter is set to generate interrupts at 100-microsecond intervals and, at each interrupt, an interrupt service routine is executed to acquire one byte of data. Prior to enabling interrupts, however, a series of initialization steps must be carried out. The data-direction registers are set to make Port A an input and Port B an output; the auxiliary control register is set to place timer T1 in free-running mode with the output to PB7 disabled; the peripheral control register makes CA1 interrupt on a negative transition (used to indicate that data are ready); T1 is loaded with the two-byte value of the sampling interval, expressed in microseconds; the address of the interrupt service routine is placed in the AIM's IRQV2 (or whichever address the 6502 jumps through when an interrupt occurs); and the interrupt enable register is set up to allow timer T1 to interrupt. Details of how to use interrupts may be found in the AIM User's Guide, the 6500 Programming Manual, or in Leventhal's *6502 Assembly Language Programming* (Osborne/McGraw-Hill). In addition, if data are to be saved in memory, a storage index and a storage vector on page zero must be given their initial values prior to the first interrupt.

Once initialization is finished and the interrupt has been enabled, the microprocessor will jump to the interrupt service routine every time T1 times out. The background task while interrupts are occurring is to scan the keyboard for single-character commands that disable the interrupts and cause jumps to playback or other data-collection modes. The interrupt service routine, shown in Program 1, first saves registers A and Y on the stack, and then executes the four input tasks that were described earlier. It assumes that T1 is the only possible source of interrupts, since it does not check the interrupt flag register for T1's flag, as would be done if there were other sources of interrupts. The program stores data in a multi-page ring buffer accessed through indirect indexed addressing. The storage vector is updated between the start of conversion and first checking for data ready, since this is time that would otherwise be wasted while waiting for conversion to finish. Part of updating the vector involves checking to see if the end of the buffer has been reached; if it has, the vector is reset to point to the beginning of the buffer. When the data are ready, the program reads Port A, stores the data in memory, and echoes the data to the output converter through Port B.

Source listing of the interrupt service routine. The routine assumes that timer T1 is the only source of interrupts.

```
INTSV1 PHA              ; PROTECT REGISTERS A & Y.
       TYA
       PHA
       LDA PCR          ; PULSE CA2 TO START A/D:
       ORA #%00001110
       STA PCR          ; FORCE CA2 HIGH
       AND #%11111101
       STA PCR          ; FORCE CA2 LOW
       LDY YVALST       ; GET STORAGE INDEX
       INY              ; UPDATE IT, &
       STY YVALST       ; SAVE INDEX VALUE
       BNE INTSV2       ; BRANCH IF Y NOT 0
       INC STVEC+1      ; ELSE UPDATE STVEC PAGE.
       LDA STVEC+1      ; CHECK NEW PAGE:
       CMP #MAXPAG      ; IS IT END OF BUFFER?
       BNE INTSV2       ; IF NOT END, CHECK A/D
       LDA #MINPAG      ; ELSE WRAP TO START
       STA STVEC+1      ; OF BUFFER.
INTSV2 LDA IFR          ; IS A/D DATA READY?
       AND #%00000010   ; ISOLATE CA1 FLAG
       BEQ INTSV2       ; LOOP IF CA1 STILL CLEAR,
       LDA PORTA        ; ELSE GET A/D DATA,
       STA (STVEC),Y    ; PUT IT IN MEMORY,
       STA PORTB        ; & ECHO IT TO D/A.
       LDA T1L          ; CLEAR T1 FLAG IN IFR
       PLA              ; RESTORE REGISTERS Y & A,
       TAY
       PLA
       RTI              ; AND RETURN...
```

Program 1.

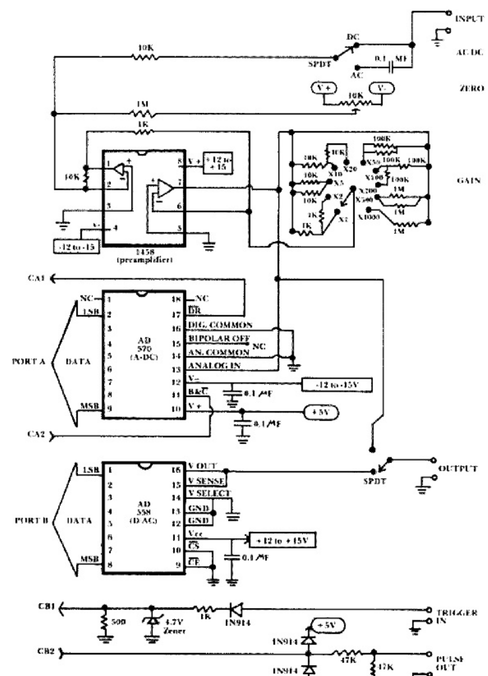

Figure 1.