EE 3232 DIGITAL SYSTEMS III
6.4 ANALOG INTERFACING

March 2000 CPD

OBJECTIVE

To investigate the operation and interface requirements of an 8-channel analog output system that includes a 12-bit digital to analog converter, or DAC, and an 8-channel sample-hold. To investigate the operation and interface requirements of a 16-channel analog input system that includes a 12-bit analog to digital converter, or ADC, and two 8-channel analog multiplexers. Coding, justification, handshaking, and analog offset errors will also be investigated. Software will be developed for reading and writing the analog IO subsystems.

REFERENCES

1. "Microprocessors and Interfacing", 2nd Edition, R.V. Hall, pp 290 - 317.
2. "The 80C186XL/80C188XL Microprocessor Users Manual", available in the lab, also at http://developer.intel.com/design/intarch/manuals/
3. Section 8.3, "DAC667 DAC Data Sheets", also at http://www.burr-brown.com/Products/DataSheets/DAC667.html
4. Section 8.4, "ADS7806 ADC Data Sheets", also at http://www.burr-brown.com/Products/DataSheets/ADS7806.html
5. Section 8.5, "The SMP08 Sample Hold Device", also at http://www.analog.com/
6. Section 8.6, "The MAX358 Analog Multiplexer", also at http://www.maxim-ic.com/AnalogSM.htm
7. Section 7.7 - 7.8 SBC188 Schematics Sheets 7/11 and 8/11.
8. "The SBC188 Users Manual", available in the lab.
9. "Paradigm Debug/RT - 186 Users Guide", available in the lab.
10. "Paradigm Locate Reference Manual", available in the lab.

EQUIPMENT

1. SBC188 board.
2. Host PC.
3. IO interface board.
4. Two 16-pin ribbon cables.
6. Oscilloscope.
7. 4½ digit DVM.
8. Variable Power Supply (or fixed power supply with a potentiometer)
9. Signal Generator.

BACKGROUND: The Analog Output System

The SBC188 contains 8-channels of analog output implemented using a single 12-bit DAC, the DAC667, and an 8-channel sample-hold device, the SMP08. A simplified block diagram of the analog output system appears in Figure 6.4.1.

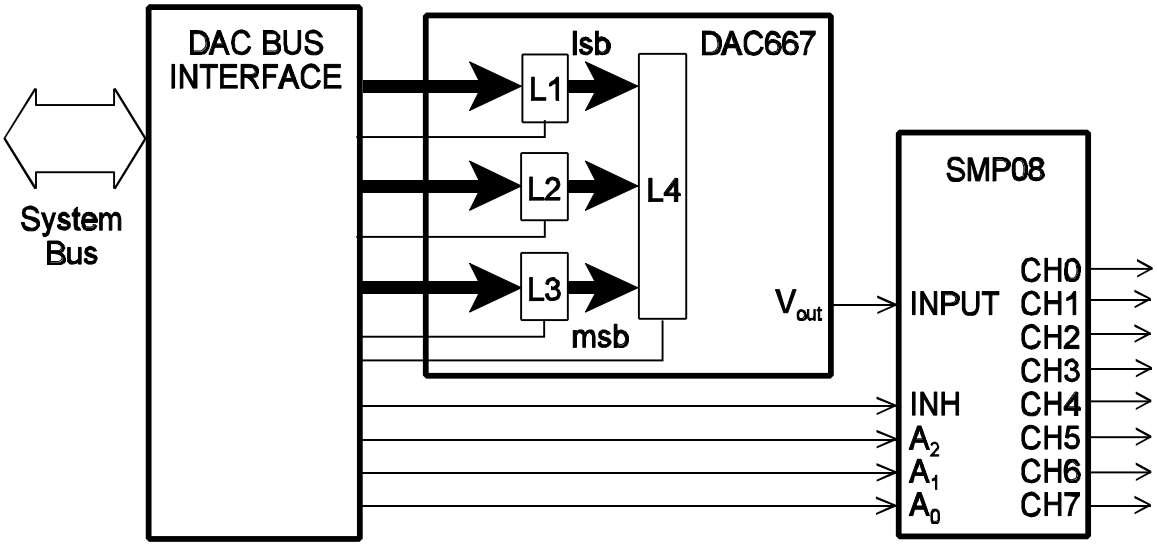


Figure 6.4.1. The Multi-Channel Analog Output System

Also consult the schematic, Sheet 7/11, in Section 7.7 and identify the following.

- The DAC667 digital to analog converter.
- The SMP08 sample hold device.
- The parallel output port, CH_SEL, implemented using a 74HCT574.
- The analog output channel connector, J701.

The DAC667 is a 12-bit DAC configured on the SBC188 as a 5V full scale, straight binary DAC.

Now examine sheet 7/11 of the SBC188 schematic more closely and identify the interface of the following DAC667 pins with the SBC188 system bus: DB0-DB11, A0-A4 and /CS. The pins A0-A4 are used to select one of a number of possible internal double buffering configurations built into the DAC667.

You should also identify the interface of the SMP08 pins, IN, A, B, C and INH. The pins INH, A, B and C are connected to bits 4-7 of output port 0x0158. When INH = '0', A, B and C are used to select the analog output channel, Chi, that ***tracks or samples*** the DAC output V_{out}. When INH = '1' the analog channels, CH0-CH7, all ***store or hold*** the previous analog values.

By consulting the IO Port Map section in your Lab Manual, you should verify that the DAC667 and SMP08 are IO-mapped in the SBC188 system as follows.

Port 0x0162: A write to port 0x0162 latches (into an internal buffer) the 8-most significant bits of the 12-bits to be sent to the DAC.

Port 0x0164: A write to port 0x0164 latches the 4-least significant bits, left justified and simultaneously loads the 8-most significant bits into the DAC. Read page 1 and the Section on Interface Logic on pages 4-5 of the **DAC667 Data Sheets** in Section 8. Ensure that you understand the selected double buffering configuration as it

relates to the operation of ports 0x162 and 0x164!

Port 0x0158: A write to port 0x0158 selects one of the analog output channels to be updated (***and*** one of the analog input channels to be converted). Bits 7-5, are used to select which analog output of the SMP08 is to be put into tracking (or sampling) mode. Bit-4 is the sample hold inhibit input, INH. Bit-4 must = '0' to force the selected analog output channel to track the DAC667 analog output. To store (or hold) the analog output, bit-4 must then be brought to logic '1'. Bits 3-0 are used to select the **analog input channel** that is applied to the R2IN pin of the ADS7806 analog to digital converter.

BACKGROUND: The Analog Input System

The SBC188 contains 16-channels of analog input implemented using a single ADC, the ADS7806, and two 8-channel analog multiplexers, the MAX358's. A simplified block diagram of the analog input system appears in Figure 6.4.2.

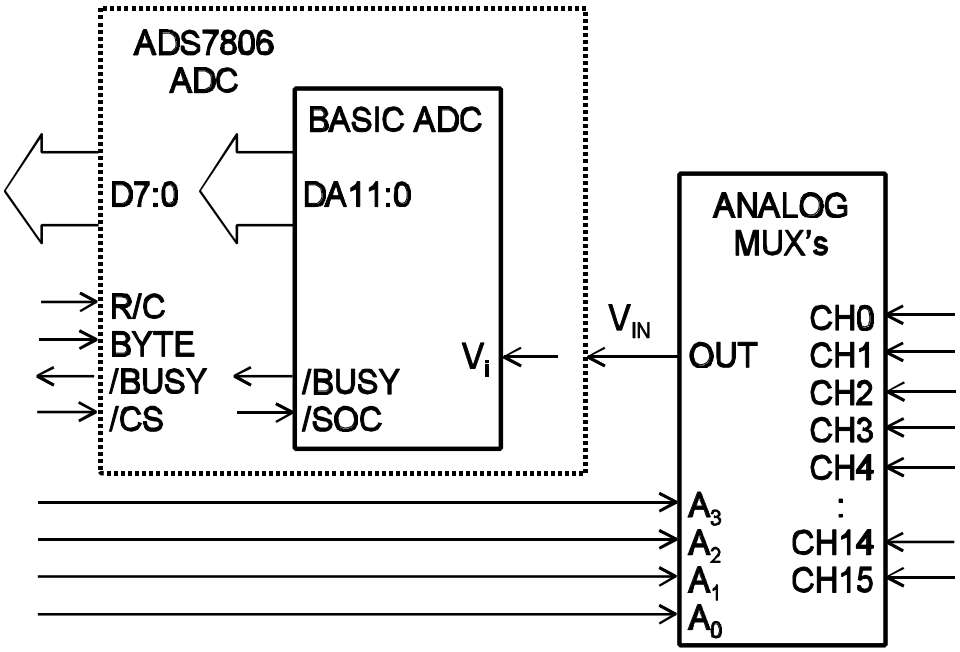


Figure 6.4.2. The Multi-Channel Analog Input System.

Also consult the schematic, Sheet 8/11, of the SBC188 schematics in Section 7.8 and identify the following:

- The ADS7806 analog to digital converter.
- The MAX538 analog multiplexers.
- The parallel output port, CH_SEL, implemented using a 74HCT574.
- The analog channel input connectors, J801 and J802.

The ADS7806 is a 12-bit unipolar, 5V full scale, straight binary analog to digital converter.

By examining the schematic of the ADS7806 more closely, identify the interface of the following MAX358 pins with the SBC188 system bus: A0-A2 and ENA. When ENA = '1' the inputs A0-A2 select the analog input, IN0-IN7, at connector J801-2 that is 'switched' to the analog output pin, OUT. This analog signal passes through several op-amps and is subsequently applied to the ADC input R2IN. Also examine the interface of the ADS7806 pins: DB0-DB7, /CS, BYTE, R/C and BUSY. What is the purpose of the pots connected to the pins REG and R1IN? What is the purpose of the LT1079 op-amps?

Read page 4 of the ADS7806 Data Sheets for a pin out and description of the ADC pins. A more detailed description of the ADC operation modes used in this experiment can also be found on pages 7-16 in the sections entitled: Basic Operation - Parallel Output, Starting a Conversion, Reading Data - Parallel Output, Input Ranges, Hardware Calibration and Reference.

By consulting the IO Port Map section in your Lab Manual, you should verify that the analog input system is IO-mapped in the SBC188 system as follows.

Port 0x0168: A read of port 0x0168 starts a 12-bit analog to digital conversion.

Port 0x0150: The /BUSY status of the ADC is read as bit-3 of port 0x150. The /BUSY status bit is low during conversion.

Port 0x016A: A read from port 0x016A will retrieve the most significant 8-bits from the ADC.

Port 0x016E: A read of port 0x016E will retrieve the least significant 4-bits (of a 12-bit conversion) left justified.

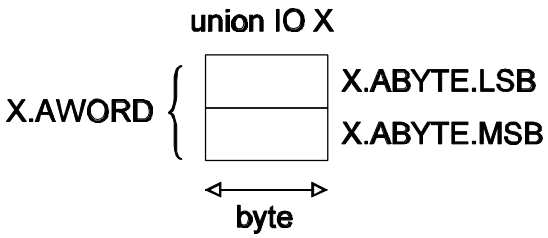
Port 0x016C: A read of port 0x016C will retrieve the least significant 4-bits (left justified) and simultaneously start a new conversion while reading the low byte.

Port 0x0158: A write to port 0x0158 selects one of the analog input channels (and one of the analog output channels). Bits 3-0 are used to select the analog input channel that is applied to the ADS7806. Bit-4 is normally set = '1' to inhibit changes at the analog output channels while ADC conversions are performed. **NOTE**: A delay of at least 4 microseconds must exist between the selection of an analog INPUT channel and the start of an analog conversion to ensure the analog multiplexer output has settled.

PREPARATION: Software Development

1. Write a properly documented subroutine, **void DAC_PUT_WORD(unsigned DATA, unsigned char CHANNEL_NO)**, that writes the unsigned word passed as DATA to the DAC and selects analog output channel, CHANNEL_NO. DATA may need to be copied into a special data structure so that the hi-order and lo-order bytes that comprise DATA may be addressed separately. One way to do this in C is to define the union IO structure as follows,

```
union IO {
        struct {
                char LSB;
                char MSB;
        } ABYTE;
        Unsigned AWORD;
};
```



An intermediate variable of type IO may be declared as, union IO X. One may assign, X.AWORD = DATA. The entire word or component bytes of X may now be accessed as, X.AWORD, X.ABYTE.LSB or X.ABYTE.MSB!

To prevent analog glitching at the output of the sample-hold device, the SMP08 should be inhibited (by setting bit-4 of port CH_SEL, 0x0158 = '1') before the data is written to the DAC. After the two bytes of data are written to the DAC the appropriate channel number is selected with bit-4 of port CH_SEL, 0x0158 = '0'. Assume the 12-bit data is stored left justified in the 16-bit variable, DATA.

2. Using DAC_PUT_WORD(), write a properly documented program **SQUARE_WAVE** that causes the output of DAC Channel 0 to switch endlessly between its maximum and minimum voltage levels.

3. Using DAC_PUT_WORD(), write a properly documented program **SAWTOOTH_WAVE** that causes the DAC Channel 0 to generate a negative slope sawtooth waveform as follows,

```
x = 0;
while (TRUE) {
        DAC_PUT_WORD(x, 0);
        x = x - 256;
        }
```

4. * Write a program, **WAVE_FORM**, by modifying the program in Part 5 to generate an arbitrary periodic waveform at the DAC output by continuously indexing through the elements of a predefined array.

```
while (TRUE) {
        i = 0;
        While (i <= M ) {
                DAC_PUT_WORD(x[i], 0);
                i = i + 1;
        }
}
```

5. Write a properly documented procedure **unsigned ADC_GET_WORD(unsigned char CHANNEL_NO)** that performs an analog to digital conversion and returns the converted data as follows.

```
Select analog input channel;
Delay for at least 4 microseconds;
Start conversion;
Wait for conversion to finish;
Read ADC data;
Return ADC data;
```

6. Write a properly documented program **ANALOG_IO** that continuously performs analog to digital conversions on channel 0 and writes the result to the DAC analog output channel 7.

```
While (TRUE) {
        X = ADC_GET_WORD(0);
        Y = X;
        DAC_PUT_WORD(Y, 7);
        }
```

7. Modify the program ANALOG_IO (call the new program **FILTER_IO**), such that the program implements a first oder digital filter, i.e., the relation X ➜ Y is given by, Y = 0.75*Y + 0.25*X;

8. Modify the program, ANALOG_IO, (call the new program, **PEAK_IO**), such that the program implements a peak detector, i.e., the output Y represents the largest (or most positive) value that X has achieved up until this point in time.

9. * Write a program, **ACQUIRE**, that performs data acquisition from a selected input channel by filling an array with converted data. The program performs periodic analog conversions and writes the converted data into the next element of the array until the array is filled.

EXPERIMENT

Part I. DIGITAL to ANALOG CONVERTERS

1. Connect a 4½ digit DVM to the analog output of the DAC and complete the following table. Use the debug monitor I/O command to output data to the DAC.

DAC Data	Ideal DAC Output (V)	Measured DAC Output(V)
0000 0000 0000		
0111 1111 1111		
1111 1111 1111		

Comment on the DAC offset and gain errors. Identify the components on sheet 7/11 of the SBC188 schematic that may be used to null the offset and gain errors.

2. Load, execute and debug the program, SQUARE_WAVE. Use an oscilloscope to observe the settling time and compare with the DAC667 specification by consulting page 2 of the DAC667 Data Sheets.

3. Load, execute and debug the program, SAWTOOTH_WAVE. What determines the period of the waveform? Observe the output on a oscilloscope. Is the size of each voltage step as expected? How would you alter the program so as to generate a sawtooth with 'opposite' slope.

4. * Load, execute and debug the program, **WAVE_FORM**.

Part II. ANALOG TO DIGITAL CONVERTERS

5. You may use the debug monitor I/O command to initiate 12-bit analog to digital conversions, and to read the converted data. Using a DVM and a variable DC voltage complete the following table.

Analog Input	ADC Output (hex)
4.9990 3.750V 2.500V 1.250V 0.000V	

Comment on the ADC offset and gain errors.

6. Load, execute and debug the program, ANALOG_IO. Observe the DAC output on an oscilloscope when the input is a sinusoid. It is especially interesting to observe the (aliasing) effects as the frequency is increased! What is the maximum number of conversions per second that your program can realize?

7. Load, execute and debug the programs, FILTER_IO and PEAK_IO.

8. * Load, execute and debug the program, ACQUIRE.

* **ADVANCED SECTIONS**

SUMMARY

1. Why are double buffers required with the 12-bit DAC?

2. Give an equation that relates input and output of a straight binary DAC. Repeat for a straight binary ADC.

3. Why handshaking is necessary with the ADC?

4. How can you null gain and offset errors for the DAC and ADC?

EXTENSIONS

Rather than performing analog conversions continuously it may be desirable in an application to synchronize analog conversions to accurately and uniformly spaced instants in time. One way of achieving this is through the use of periodic interrupts generated by an internal 80C188XL timer. The following describes how the 80C188XL timer and interrupt controller may be programmed for periodic interrupt generation!

First the interrupt vector table entry for an internal 80C188XL timer must be initialized. If we select Timer 1 as the interrupt source then the associated interrupt type number is 18 (decimal). Refer to page 8-9 in the 80C188XL Microprocessor Users Manual.

To allow a wide range of possible programmed frequencies, the clock input of Timer 1 may be pre-scaled by Timer 2. In this case the software must perform the following.

- Clear the 16-bit timer count registers T1CNT (port address 0xFF58) and T2CNT (port address 0xFF60).
- Write N to the 16-bit compare register for Timer 2, i.e., T2CMPA (port address 0xFF62). This sets the input clock frequency of Timer 1 = 4 MHz ÷ N.
- Write M to the 16-bit compare register for Timer 1, i.e., T1CMPA (port address 0xFF5A). This programs a periodic interrupt of frequency = (4 MHz ÷ N) ÷ M.
- Write 0xC001 to T2CON (port address 0xFF66) to program continuous counting.
- Write 0xE009 to T1CON (port address 0xFF5E) to enable interrupt generation and pre-scaling by Timer 2.

Finally the 80C188XL interrupt controller needs to be initialized to unmask timer interrupts by writing 0x0000 to the 16-bit port TCUCON (address 0xFF32).

You should be able to explain how to modify the programs, ANALOG_IO, FILTER_IO and PEAK_IO, such that the IO operations are synchronized to a timer interrupt. What are the advantages and disadvantages of the interrupt driven interface?