Summary

There are times you may want to control the QED Board's digital-to-analog converter (DAC) from assembly language because you may need to precisely control the timing of DAC outputs or run the DAC near full speed. For example, you may want to feed an audio waveform to the DAC and control the rate at which samples are output, while guaranteeing that samples are put out with very uniform timing. While the QED kernel's words **>DAC** and **(>DAC)** are very good for putting out single samples you may want to better control the timing of the output samples than you can using those. This application note shows you how you can achieve that.

Description

This application note provides the following two blocks of code that you can modify for your own purposes:

1. The QED Forth Kernel's source code for the word **(>DAC)**; and,

2. An example which outputs samples to the DAC from within an interrupt service routine.

Each is described in detail.

1. The Source Code for the routine (>DAC)

The first block of code is the QED Forth Kernel's source code for the word **(>DAC)** It shows how to write to the DAC through the SPI (Serial Peripheral Interface) using assembly code. For a basic understanding of the operation of the routine please read the glossary entries for the words **>DAC** and **(>DAC)**.

The DAC communicates with the 68HC11 processor on the serial peripheral interface (SPI), a high speed serial bus. Details of the DAC's hardware requirements, including the data sheet for the chip (an Analog Devices DAC-8841) are provided in the QED Hardware Manual. Writing a byte to a processor register signals the processor to send the byte out serially. As the byte is shifted out to the DAC another byte is simultaneously shifted from the DAC into the processor. In the following code the routine **ACCB<->SPI** transfers the byte by writing it to the **SPDR** (serial peripheral data register) register, and loops on a status register (**SPSR**) until the byte transfer is completed. The status register must be read before the next byte transfer can take place. **(>DAC)** calls the word **ACCB<->SPI** to first shift a channel number to the DAC, then again to shift in the data byte. Finally, the DAC is chip selected; that is, the DAC's load pin is strobed by writing to the DAC's address.

The following code is heavily commented and should be self explanatory:

```
\ ************************************************************************
\ ****                                                                ****
\ ****                    The source code for (>DAC)                  ****
\ ****                                                                ****
\ **** This code is provided only to customers of Mosaic Industries' QED ****
\ ****   Board for use in assisting them in coding their applications.   ****
\ ****           Copyright 1997 by Mosaic Industries, Inc.            ****
\ ****                                                                ****
\ ************************************************************************

HEX

80C0 CONSTANT  DAC.ADDR      \ thru 80CF;a byte written to any of these
                             \ addresses causes the onboard PAL to assert
                             \ the DAC.CS (its load pin) high

8028 CONSTANT  SPCR          \ SPI (serial peripheral interface) control register
                             \ This register must be written to by the words
                             \ INIT.SPI or INIT.A/D12&DAC before calling (>DAC)
                             \ (>DAC) and >DAC do not use it directly.

8029 CONSTANT  SPSR          \ SPI status register. This register is read to
                             \ determine when the SPI transfer is complete.

802A CONSTANT  SPDR          \ SPI data register. A write into this register
                             \ initiates bidirectional data transfer

  80 CONSTANT  SPI.TRANSFER.COMPLETE \ Poll this flag in SPSR to detect
                             \ completion of data transfer.


CODE ACCB<->SPI ( -- | needs byte to be transferred in in ACCB )
    \ This routine shifts a byte in accumulator B out the SPI and replaces
    \ accumulator B with the byte received.  ACCB is sent via the SPI to
    \ whatever slave is active.  The slave's response is returned in ACCB.
    \ Only accumulator B is affected by this word.
    SPDR EXT STAB                  \ Send out ACCB via the SPI.
                                   \ A simple store initiates the SPI transfer.
    SPI.TRANSFER.COMPLETE IMM LDAB \ Load ACCB with the transfer complete mask.
    BEGIN,                         \ Loop on the status register and wait until
        SPSR EXT BITB              \ the transfer complete bit is set.  Transfer
    NE UNTIL,                      \ occurs in 4 usec on a QED-3 Board.
                                   \ Even in applications in which we don't need
                                   \ to loop on the status bit we still must
                                   \ read SPSR before sending out another byte.
    SPDR EXT LDAB                  \ Load ACCB with the incoming SPI byte.
    RTS
END.CODE
```

```
CODE (>DAC) ( u\n -- )
    \ This routine sends a byte to a particular DAC channel.  Note that it
    \ assumes that the SPI is available for use by the DAC.  Use the word >DAC
    \ to resolve contention when multiple slaves could be using the SPI.  The
    \ DAC is written to by first shifting in the channel# via the SPI, then
    \ shifting in the data, then strobing the load pin on the chip.
    \ In the stack picture, u is the data byte to be sent to the DAC and n is
    \ the channel number where: 0 <= u (data) <= 255 ; 1 <= n (channel) <= 8
    01 IND,Y LDAB          \ Load ACCB with the channel# from the stack
                           \ channel# is the lsbyte of the top stack item and
                           \ register Y holds the parameter stack pointer.
    ACCB<->SPI EXT JSR     \ Send the channel# to the DAC.
    03 IND,Y LDAB          \ Load ACCB with the data byte from the stack.
                           \ The data is the lsbyte of the buried stack item.
    ACCB<->SPI EXT JSR     \ Send the data byte to the DAC.
    DAC.ADDR EXT STAB      \ Write to the DAC address to strobe the data from the
                           \ DAC's internal shift register into its internal data
                           \ register.  The load, or chip select, signal is
                           \ generated by a PAL on the QED Board.
    4 IMM LDAB             \ Increment the stack pointer to drop two stack
    ABY                    \ cells.
    RTS
```

2. Outputting to the DAC from within an Interrupt Service Routine

The second block of code shows how you might sequentially write 8-bit samples from a buffer into the DAC from within an interrupt service routine. To understand its operation first read the description and code for **(>DAC)** above. This code is very similar.

In this example it is assumed that every time the interrupt service routine is called another byte of data from a queue is output to the DAC. For the greatest speed possible the queue is kept in the QED Board's common memory or on the same page as the code that calls it so that no page change is needed to address it. A 16-bit address keeps track of the current position in the queue and is incremented each time the interrupt service routine is called. In order for the queue to wrap around, every time the address is incremented it is compared to the end address of the buffer (queue) and reset to the starting address if needed.

For the greatest speed the address computations and test are done while the SPI is outputting the channel number to the DAC, and because this takes a little more than the 4 usec needed for the serial transfer we don't need to loop on the SPI's status register. Instead, a single dummy read of the status register prepares the SPI for the next transfer. Subsequently, the data is sent to the DAC by the SPI. While the data byte is being transferred you can either loop on the status register as is done in **(>DAC)** above, or, to save time, you can execute any other code that needs to be done in the interrupt service routine. After sufficient time has elapsed, a single dummy read of the status register prepares the SPI for the next transfer, and writing to the DAC's address strobes its load pin.

The interrupt service routine should be called by an interrupt tied to one of the 68HC11's timers so that you can easily control its timing.

```
\ ************************************************************************
\ ****                                                                ****
\ ****                  An example of using the DAC                   ****
\ ****             from inside an interrupt service routine:          ****
\ ****                                                                ****
\ ****                                                                ****
\ **** This code is provided only to customers of Mosaic Industries' QED ****
\ ****   Board for use in assisting them in coding their applications.  ****
\ ****           Copyright 1997 by Mosaic Industries, Inc.            ****
\ ****                                                                ****
\ ************************************************************************

01 CONSTANT  DAC.CHANNEL  \ This can be changed to any of the DAC channels 1-8.
VARIABLE Base.Addr        \ The 16-bit base address of the sample stream
                          \ to be sent out.  It is assumed here that the
                          \ samples are contained in a queue extending from
                          \ the Base.Addr to the End.Addr. The queue can be
                          \ any length as long as it fits on a single page.
                          \ The samples can be on the common page or on the same
                          \ page as the interrupt service routine.  Base.Addr
                          \ and End.Addr must be initialized before the first
                          \ time the interrupt is serviced.

VARIABLE End.Addr         \ The 16-bit final address of the sample stream. It
                          \ should actually point to the address just beyond
                          \ the last sample.

VARIABLE Addr.Pointer     \ Before the first invocation of the interrupt
                          \ service routine the Addr.Pointer should be
                          \ initialized to the Base.Addr.  The Addr.Pointer
                          \ steps through the sample queue wrapping back to
                          \ the beginning when it gets to the end.
```

```
\ Before the Interrupt.Service is called the first time an initialization
\ routine should call INIT.A/D12&DAC and also initialize Base.Addr, End.Addr,
\ and Addr.Pointer.

CODE Interrupt.Service
    \ ...
    \ ...
    \ ...
    \ ...
    \ other code within the interrupt service routine
    \ ...
    \ ...
    \ ...
    DAC.CHANNEL IMM LDAB          \ Load ACCB with the DAC channel.
    SPDR EXT STAB                 \ Send out ACCB via the SPI.
                                  \ A simple store initiates the SPI transfer.
    Addr.Pointer DROP EXT LDX     \ While we're waiting 4 usec for the SPI
                                  \ we have time to load the sample and do the
                                  \ address computations.
    00 IND,X LDAB                 \ Fetch the next sample from memory into ACCB
    INX                           \ Increment the address to point to the next
                                  \ sample.
    End.Addr DROP EXT CPX         \ Compare the address to the end address.
    EQ IF,                        \ Are we pointing to the buffer's end?
        Base.Addr DROP EXT LDX    \ If so, reload reg X with the buffer's start
    ENDIF,                        \ address.
                                  \ Since beginning of SPI tansfer 5.25 to 6.5
                                  \ usec have elapsed.
    SPSR EXT LDAA                 \ A dummmy read of the SPI's status register
                                  \ prepares it for the next transfer.
    SPDR EXT STAB                 \ Send out data in ACCB via the SPI.
                                  \ A simple store initiates the SPI transfer.
    Addr.Pointer DROP EXT STX     \ Store the new address.


\  Here, you can either loop on the status flag like this....
    SPI.TRANSFER.COMPLETE IMM LDAB \ Load ACCB with the transfer complete mask.
    BEGIN,                        \ Loop on the status register and wait until
        SPSR EXT BITB             \ the transfer complete bit is set.  Transfer
    NE UNTIL,                     \ occurs in 4 usec on a QED-3 Board.

\ or do other things that take at least 2.75 usec (1.25 usec was already taken
\ by the STX above) followed by a dummy read of the status flag as,
    \ ...
    \ ...
    \ other code that must be in the interrupt service routine anyway
    \ ...
    \ ...
    SPSR EXT LDAA                 \ A dummmy read of the SPI's status register
                                  \ prepares it for the next transfer.

\ Finally, strobe the DAC so that the data transferred to it appears as its
\ analog output.
    DAC.ADDR EXT STAB             \ Write to the DAC address to strobe the data
                                  \ from the DAC's internal shift register into
                                  \ its internal data register.  The load, or
                                  \ chip select, signal is generated by a PAL
```

```
\ on the QED Board.

\ Then continue with your interrupt service routine.
\ ...
\ ...
\ ...
\ ...

    RTS                             \ Return from the interrupt service.
END-CODE                            \ Note that an RTS is used instead of an RTI.
```


Mosaic Industries

5437 Central Ave Suite 1, Newark, CA 94560 Telephone: (510) 790-8222 Fax: (510) 790-0925