Programming Sequences for the QF1Da512

1) Introduction

The QF1Da512 SavFIRe is an extremely powerful, flexible, and inexpensive FIR engine. It has a straightforward SPI interface that allows it to be easily integrated into most systems. To help facilitate this integration, this document describes a typical programming sequence used to program the chip. It also provides C-code examples that can be used as a basis for programming by a user.

This application note is intended to be used in conjunction with the QF1Da512 SavFIRe Datasheet. This Datasheet contains information indirectly referenced in this document. Data such as RSTn timing parameters, logic levels, and SPI timing waveform diagrams are located in the Datasheet. The latest QF1Da512 SavFIRe Datasheet can be downloaded from the main Quickfilter Website.

2) Programming Sequence

Only a few simple steps are required to program the QF1Da512 and enable it to begin processing data. Figure 1 below shows such a sequence. Please refer to Figure 1 as needed during the following explanations.

It is important that the QF1Da512 be in a known state before attempting access to any of its internal registers. Step 1 must ensure the chip has adequate voltage on its power pins and that its input logic levels are within proper ranges. If RSTn is held low throughout Step 1, then in Step 2 RSTn may simply be deasserted. If RSTn follows VCC high during the application of power, it is necessary to pulse RSTn low for a short amount of time to force all of the QF1Da512's internal state logic to known values.

Step 3 places the QF1Da512 into Configuration Mode and disables filtering. The QF1Da512 powers up in this mode by default, with its dSDO pin tristated, however it is recommended to actively place it into Configuration Mode.

Steps 4 and 5 may be swapped, if desired. The coefficients and configuration registers may be programmed in any order. This is true as long as the last register written is the CONFIG register in which the FILT_EN bit is not set until all such programming is complete.

Once the FILT_EN bit is set in Step 6, the QF1Da512 will be processing data.

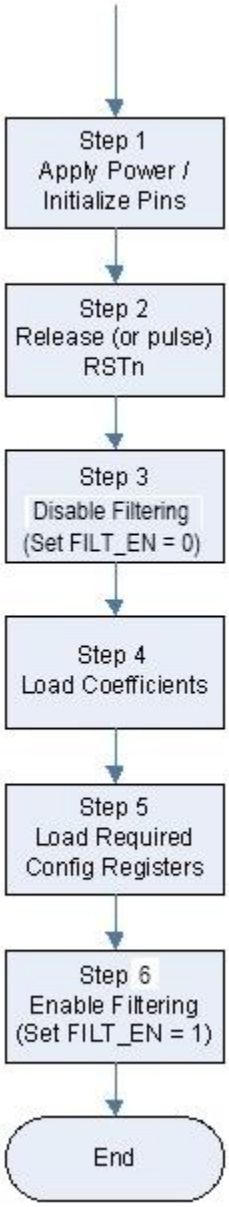

Figure 1 - Programming Sequence for the QF1Da512

3) SPI Timing

As stated in the QF1Da512 SavFIRe Datasheet, reading and writing the configuration register or coefficient values requires an SPI sequence consisting of 4 or more bytes (32-bits) of information. Figure 2 below shows the detailed SPI timing diagram. The information is delivered in this order:

8-bit Op Code (indicating a Configuration Read, Configuration Write, Coefficient Read, or Coefficient Write)
6-bit Don't Care Field
8-bit Address to indicate the config register address or coefficient number
2-bit Don't Care Field (used to allow time for the QF1Da512 to respond to read Op-codes
8-bit / 32-bit data fields (depending on if config or coefficient values are being accessed

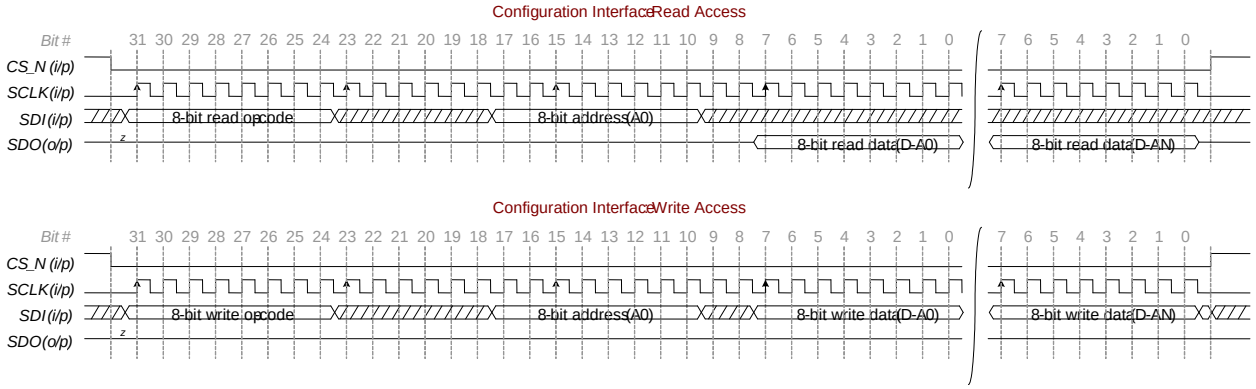

Figure 2 - Configuration Data Timing

4) Code Examples

The following is an example of C-code that will perform the writing of config and coefficient data to the chip. It is derived from actual working C-code. It is assumed the reader is familiar with C-language constructs. This code is for example only and comes as-is. It will need to be adapted to the user's specific processor and system in order for it to be able to function in the user's environment. The code is written in a generic style and doesn't rely on built-in microprocessor hardware (such as an integrated UART of some kind) in order to function.

C-code summary of functions and variables:

```
_QF_file0_SavFIReConfigRegisters        Constant 8-bit array that holds the config register values
_QF_file0_SavFIReCoefficients           Constant 32-bit array that holds the coefficient values
QF1D512_WriteConfigByte()               Function to write a byte to the config space.  Address and Data are passed as arguments
QF1D512_WriteCoefficients()             Function to write all coefficients.  Total # of coefficients is passed on the command line.
                                        All other information is passed as global variables.
send_eight()                            Function to write 8-bits of information.  Data is stored in the outputBUFFER_8bit global variable.
send_sixteen()                          Function to write 16-bits of information.  Data is stored in the outputBUFFER_16bit global variable
main()                                  This is the entrance point of the code.
```

C-code declaration of Configuration Registers and Coefficient Values:

```
/*    Begin Initial declarations                    */
/*    Number of elements in SavFIReConfigRegisters. */
#define  _QF_file0_NUM_SAVFIRE_REGISTERS 34

/*    Number of coefficients in _QF_file0_SavFIReCoefficients */
#define  _QF_file0_NUM_SAVFIRE_COEFFICIENTS 4

/*   Array of SavFIRe configration values */
const unsigned char  _QF_file0_SavFIReConfigRegisters[ _QF_file0_NUM_SAVFIRE_REGISTERS ] =
{
             /* Dec Addr        Name   Description */
             /* --- ----        ----   ----------- */
    0x00,    /* 000  00      TEST_RW    */
    0xC0,    /* 192  01      CHIP_ID    */
    0x01,    /* 001  02      VERSION    */
    0x00,    /* 000  03      CONTROL    */
    0x63,    /* 099  04      DCONFIG    */
    0x02,    /* 002  05      FCONFIG    */
    0x06,    /* 006  06   NUM_TAPS_0    */
    0x00,    /* 000  07   NUM_TAPS_1    */
    0x00,    /* 000  08     DECIMATE    */
    0x00,    /* 000  09    HD_OFFSET    */
    0x00,    /* 000  0A      HD_SIZE    */
    0x00,    /* 000  0B      HD_MASK    */
    0x00,    /* 000  0C     HD_VALUE    */
    0x01,    /* 001  0D  DATA_OFFSET    */
    0x18,    /* 024  0E    DATA_SIZE    */
    0x00,    /* 000  0F       GAIN_0    */
    0x10,    /* 016  10       GAIN_1    */
    0xFF,    /* 255  11     THRESH_0    */
    0xFF,    /* 255  12     THRESH_1    */
    0x00,    /* 000  13      MULTI_0    */
    0x10,    /* 016  14      MULTI_1    */
    0x00,    /* 000  15      ADDOR_0    */
    0x00,    /* 000  16      ADDOR_1    */
    0x03,    /* 003  17       IO_TST    */
    0x05,    /* 005  18     IO_RST_N    */
    0x01,    /* 001  19      IO_DCLK    */
    0x01,    /* 001  1A      IO_DSEL    */
    0x01,    /* 001  1B       IO_DIN    */
    0x05,    /* 005  1C      IO_CS_N    */
    0x01,    /* 001  1D      IO_SCLK    */
    0x01,    /* 001  1E       IO_SDI    */
    0x04,    /* 004  1F       IO_SDO    */
    0x04,    /* 004  20      IO_DOUT    */
    0x40     /* 064  21         TEST    */
};


/*    Array of SavFIRe coefficient values */
const unsigned long  _QF_file0_SavFIReCoefficients[ _QF_file0_NUM_SAVFIRE_COEFFICIENTS ] =
{
                  /*   Num     Fractional */
                  /*   ---     ---------- */
    0x00763393,   /*   000    0.003607222345 */
    0x06C3BB81,   /*   001    0.052848279942 */
    0x1F40FC9B,   /*   002    0.244170737918 */
    0x330A28A2    /*   003    0.398747519590 */
};
```

C-code Main Function:

```
//------- Begin Main C Function ------------------------------------------------


//------- Variables -----------------------------------------------------------

uint16_t Number_of_Coefficients;
uint8_t  Config_Reg_Temp;                       // Temporarily holds the config ref info prior to writing to the chip.
uint16_t  cfg_address;

const unsigned char * Config_Reg_array_pointer;   // This will point at the configuration array.
const unsigned long * Coeff_array_pointer;        // This will point at the coefficient array.
const unsigned char * Config_Reg_array_base;      // This holds the base address for a config array.


//----- Begin main() Code ----------------------------------------------------------

main()
          {
          cCSn = 1;

          RSTn = 0;                                                                 // Reset the chip
          RSTn = 1;

          cCSn = 0;                                                                 // Take the chip select low

          Config_Reg_array_pointer = &_QF_file0_SavFIReConfigRegisters[4];      // Point at the configuration register data.
                                                                                    // Don't need to write the first 3 values at all.
                                                                                    // The CONTROL register will be written later.
                                                                                    // In this implementation the config registers are
                                                                                    // written first and then the coefficient values.

          Number_of_Coefficients   = _QF_file0_NUM_SAVFIRE_COEFFICIENTS;        // Determine how many coefficients we have.

          QF1D512_WriteConfigByte( 0x03, 0x00 );                                    // Clear FILT_EN (will be off if chip was just powered up);

          cfg_address = 4;

          // write the Configuration registers.
          for ( i = 4; i < 33; i++)
                    {                                                               // There are 30 registers used.
                    Config_Reg_Temp = pgm_read_byte(Config_Reg_array_pointer);
                    QF1D512_WriteConfigByte( cfg_address, Config_Reg_Temp );
                    Config_Reg_array_pointer = Config_Reg_array_pointer + 1;    // Point at the next table entry.
                    cfg_address = cfg_address + 1;
                    }

          // Then write all of the Coefficients.
          QF1D512_WriteCoefficients(Number_of_Coefficients );

          QF1D512_WriteConfigByte( 0x03, 0x01 );                                    // Set FILT_EN

          //Take the chip select high.
          cCSn = 1;
          }
```

C-code Subroutines:

```
//---- Begin Subroutine Code ------------



//------- Variables ------------------------------------------------------------
unsigned char outputBUFFER_8bit;
unsigned char outputBIT_8bit;
unsigned int outputBUFFER_16bit;
unsigned int outputBIT_16bit;
unsigned long outputBUFFER_32bit;
unsigned long outputBIT_32bit;

const unsigned int * Coeff_base_addr;   // This will point at the start of each coefficient array
const unsigned int * Coeff_array_addr;  // This will point at the configuration arrays.

extern unsigned int active_config;               //Indicates which QF1Da512 is being addressed.


//----- Writes a single byte of data to the config space as the passed in Address.
void QF1D512_WriteConfigByte( unsigned int Address, unsigned char Value )
          {

          // Put the Write-Configuration-Byte opcode into the 8-bit output buffer.
          outputBUFFER_8bit = 0x82;

          // Put Configuration Register address into output buffer.
          outputBUFFER_16bit = Address;

          // Left shift the address 2 bits, since only 14 are needed.
          outputBUFFER_16bit = outputBUFFER_16bit << 2;

          // Take the chip select low.
          cCSn = 0;

          // First shift out the Write-opcode bits on the appropriate pin with SCLK toggling.
          send_eight();

          // Next shift out the Configuration Byte Address bits on the appropriate pin with SCLK toggling.
          send_sixteen();

          // Lastly, shift out the Configuration Byte Data on the appropriate pin with SCLK toggling.
          outputBUFFER_8bit = Value;    //Load the data into the buffer.

          send_eight();

          // Take SCLK low
          cSCK = 0;

          // Take cCSn high.
          cCSn = 1;

          }
```

```
//--------- Writes all coefficients to the chip --------
void QF1D512_WriteCoefficients( unsigned int Length )
          {
          unsigned int i;
          unsigned int j;

          // Put the WRITE Coefficient Byte opcode into the 8-bit output buffer.
          outputBUFFER_8bit = 0x86;

          // Put the first Coefficient Register address into output buffer.
          outputBUFFER_16bit = 0x0000;

          // Left shift the address 2 bits, since only 14 are needed.
          outputBUFFER_16bit = outputBUFFER_16bit << 2;

          // Take the chip select low.
          cCSn = 0;

          // First shift out the WRITE opcode bits on the appropriate pin with SCLK toggling.
          send_eight();

          // Next shift out the Coefficient Byte Address bits on the appropriate pin with SCLK toggling.
          send_sixteen();

          Coeff_base_addr = &_QF_file0_SavFIReCoefficients[0];        //Point at Coeff 0 space.

          Coeff_array_addr = Coeff_base_addr;

          for (j = 0; j < Length; j++)
                    {
                    outputBUFFER_32bit = _QF_file0_SavFIReCoefficients[j];      // Load the coefficient into a temporary variable.
                    Coeff_array_addr = Coeff_array_addr + 0x0001;               // A pointer will increment to the next element

                    // Send the 32-bit coefficient word (expanded from 16 bits).
                    for (i = 0; i < 32; i++)
                              {
                              outputBIT_32bit = (outputBUFFER_32bit & 0x80000000);        //Take the next bit to be transmitted.
                              outputBIT_32bit = outputBIT_32bit >> 31;                    // JS - don't need this line of code

                              // Send off the bit.
                              cSCK = 0;                                                   //Take SCLK low.

                              if    (outputBIT_32bit == 0)  { cSDI = 0); }                //Take MOSI low.
                              else                          { cSDI = 1;  }                //Take MOSI high.

                              cSCK = 1;                                                   //Take SCLK high.

                              outputBUFFER_32bit = outputBUFFER_32bit << 1;     //Move the next bit into the MSb
                              }
                    }

          //Take SCLK low
          cSCK = 0;

          //Take cCSn high.
          cCSn = 1;
}
```

```
//-------- Writes 8 bits of data (stored in the outputBUFFER_8bit global variable) to the chip
void send_eight()
          {
          uint8_t m;

          // First shift out the WRITE opcode bits on the appropriate pin with SCLK toggling.
          for (m = 0; m < 8; m++)
                    {
                    outputBIT_8bit = (outputBUFFER_8bit & 0x80);      //Take the next bit to be transmitted.
                    outputBIT_8bit = outputBIT_8bit >> 7;

                    // Send off the bit.
                    cSCK = 0;                                         //Take SCLK low.

                    if (outputBIT_8bit == 0)      { cSDI = 0; }       //Take MOSI low.
                    else                          { cSDI = 1; }       //Take MOSI high.

                    cSCK = 1;                                         //Take SCLK high.

                    outputBUFFER_8bit = outputBUFFER_8bit << 1;       //Move the next bit into the MSb
                    }
          }
```

```
//-------- Writes 16 bits of data (stored in the outputBUFFER_16bit global variable) to the chip
void send_sixteen()
          {
          uint8_t k;

          for (k = 0; k < 16; k++)
                    {
                    outputBIT_16bit = (outputBUFFER_16bit & 0x8000);  //Take the next bit to be transmitted.
                    outputBIT_16bit = outputBIT_16bit >> 15;

                    // Send off the bit.
                    cSCK = 0;                                         //Take SCLK low.

                    if (outputBIT_16bit == 0)     { cSDI = 0; }       //Take MOSI low.
                    else                          { cSDI = 1; }       //Take MOSI high.

                    cSCK = 1;                                         //Take SCLK high.

                    outputBUFFER_16bit = outputBUFFER_16bit << 1;     //Move the next bit into the MSb
                    }
          }
```

Contact Information:

Quickfilter Technologies, Inc.
1024 S. Greenville Avenue, Suite 100
Allen, TX 75002-3324

General:info@quickfilter.net
Applications:apps@quickfilter.net
Sales:sales@quickfilter.net
Phone:214-547-0460
Fax:214-547-0481