AN219

Using Microcontrollers in Digital Signal Processing Applications

1. Introduction

Digital signal processing algorithms are powerful tools that provide algorithmic solutions to common problems. For example, digital filters provide several benefits over their analog counterparts. These algorithms are traditionally implemented using dedicated digital signal processing (DSP) chips, FPGAs, or RISC processors. While these solutions are very efficient at their purpose, they only perform one function in the system and can be both expensive and large. This application note discusses an alternative solution using a Silicon Labs microcontroller to implement DSP algorithms in less space and still have plenty of CPU bandwidth available for other tasks.

This application note discusses the implementation of three DSP solutions on the C8051F12x and C8051F36x family of microcontrollers:

- FIR filters
- Goertzel Algorithm used for DTMF decoding
- FFT algorithm

For each of these topics, we introduce the algorithm, discuss the implementation of these algorithms on the DSP-enabled MCUs using the multiply and accumulate (MAC) engine, and provide a list of the CPU bandwidth and memory usage.

1.1. Key Points

The 100 peak MIPS CPU, 2-cycle 16x16 MAC engine and on-chip ADC and DAC make the C8051F12x and C8051F36x well suited to DSP applications. Using these resources on a C8051F36x microcontroller, a 5x5 mm 8-bit MCU can process data in real-time for FIR filters and Goertzel Algorithms for DTMF decoding and implement a full FFT.

2. Digital FIR Filters

Filters have many applications, including narrowing the input waveform to a band of interest and notching out undesired noise. Digital filters have some benefits over their analog counterparts. For example, they are extremely reconfigurable since they only rely on digital numbers, which are easily changeable, to determine the filter behavior. The response of analog filters is determined by external components, which must be replaced if the filter's behavior is to be altered. Additionally, digital filters typically require fewer external components, which reduces manufacturing cost and improves reliability. External components, such as resistors and capacitors, can also be sensitive to temperature change and aging effects, which can alter the filter's behavior if the environment changes. Since a digital filter is algorithmic, its behavior is not affected when the environment changes.

There are two types of digital filters: infinite impulse response (IIR) and finite impulse response (FIR). IIR filters have a non-zero response over time to an input impulse. The output of an IIR filter relies on both previous inputs and the previous outputs. FIR filters settle to zero over time. The output of an FIR filter relies on previous inputs only and does not rely on previous outputs.

2.1. Digital Filter Algorithms

The digital filter equations are based on the following basic transfer function shown in the z domain:

$$Y(z) = H(z)X(z),$$

where *Y(z)* is the filter output, *X(z)* is the filter input, and *H(z)* is the transfer function of the filter.

H(z) can be expanded as follows:

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + \ldots + a(n+1)z^{-n}}X(z)$$

where *a* and *b* are sets of coefficients and *z* is a delay element.

Copyright © 2008 by Silicon Laboratories

2.1.1. IIR Filter Algorithm

The IIR topology extends directly from this equation by moving the denominator of the expanded *H(z)* to the left side of the equation:

$$(a(1)+a(2)z^{-1}+\ldots+a(n+1)z^{-n})Y(z) = (b(1)+b(2)z^{-1}+\ldots+b(n+1)z^{-n})X(z)$$

In the time domain, this equation appears as follows:

$$a(1)y(k) + a(2)y(k-1)+\ldots+a(n+1)y(k-n) = b(1)x(k) + b(2)x(k-1)+\ldots+b(n+1)x(k-n)$$

where *y(k)* represents the current filter output, *x(k)* represents the current input, *y(k-1)* represents the previous output, *x(k-1)* represents the previous input, and so on. If this equation is solved for *y(k)*:

$$y(k) = \frac{b(1)x(k) + b(2)x(k-1) + \ldots + b(n+1)x(k-n) - a(2)y(k-1) - \ldots - a(n+1)y(k-n)}{a(1)}$$

This equation shows that the IIR filter is a feedback system, which generates the current output based on the current and previous inputs as well as the previous outputs. The IIR structure has unique advantages and drawbacks. The main advantage of the IIR structure is that it provides a frequency response comparable to an FIR filter of a higher order. This results in fewer calculations necessary to implement the filter. IIR filters can suffer from instability because they rely on feedback. As a result, they are more difficult to design and special care must be taken to prevent an unstable system. IIR filters may also have a non-linear phase response, which can make them inappropriate for some applications where linear phase is necessary. Finally, because they rely on past outputs, they tend to be more sensitive to quantization noise, making them difficult to implement with 16-bit fixed point hardware. Generally, 32-bit hardware is necessary for an IIR filter implementation.

2.1.2. FIR Filter Algorithm

In contrast, an FIR filter has no feedback. The filter transfer function can be derived in the same way as before. However, there is only one *a* coefficient and it is equal to one (a(1) = 1). When solving the equation for *y(k)*:

$$y(k) = \frac{b(1)x(k) + b(2)x(k-1) + \ldots + b(n+1)x(k-n)}{a(1)}$$

For the FIR algorithm, the current output is generated based only on the current and previous inputs. In effect, an FIR is a weighted sum operation.

FIR filters have several advantages and drawbacks. One of the main advantages is that FIR filters are inherently stable. This characteristic makes designing FIR filters easier than designing IIR filters. In addition, FIR filters can provide linear phase response which may be important for some applications. Another important advantage of FIR filters is that they are more resistant to quantization noise in their coefficients. As a result, they can be readily implemented using 16-bit fixed point hardware such as the Multiply and Accumulate module on the C8051F12x and C8051F36x. The main drawback of FIR filters is that they require significantly more mathematical operations to achieve a response similar to an IIR filter. Because of the ease of design and their compatibility with fixed-point microcontrollers, the FIR filter will be the focus of the implementation discussion for the rest of this application note.

Replacing a(1)=1 and *C* for the *b* constants, the equation for the FIR filter is as follows:

$$y(n) = C_0x(n) + C_1x(n-1) + C_2x(n-2) + C_3x(n-3) + \ldots,$$

where *y(n)* is the most recent filter output and *x(n)* is the most recent filter input. The filter does rely on previous inputs, as shown by the *x(n-1)*, *x(n-2)*, etc. terms. The C_x constants determine the filter response and can be derived using many different algorithms, each yielding different characteristics.

This algorithm works as follows:

The first input, *x(1)* is multiplied by C_0. The output *y(1)* is as follows:

$$y(1) = C_0x(1)$$

The *x(1)* input is then saved for the next pass through the FIR algorithm.

The second input, *x(2)* is multiplied by C_0 and the previous input *x(1)* is multiplied by C_1. The output *y(2)* is as follows:

$$y(2) = C_0x(2) + C_1x(1)$$

The *x(1)* and *x(2)* inputs are saved for the next input *x(3)*, and so on.

The order of an FIR filter is equal to one less than the number of constants and is an indication of the degree of complexity and the number of input samples that need to be stored. The higher the order, the better the characteristics of the filter (sharper curve and flatter response in the non-attenuation region).

2.2. FIR Algorithm Implementation on the C8051F12x and C8051F36x

The C8051F12x and C8051F36x MAC engine is uniquely suited to implement FIR algorithms. Each pass through the filter requires multiplies and accumulates, which the MAC engine was designed to implement quickly and efficiently. Coupled with the 100 MIPS 8051 processor, the ‘F12x and ‘F36x are able to calculate the FIR filter algorithm in real time while still leaving ample CPU resources available for other tasks.

2.2.1. Implementation Optimizations

In the FIR algorithm, the previous inputs to the filter are used in each output calculation. Instead of shuffling these data points through an array to place the newest input in the same place (address 0, for instance), the FIR algorithm can use a circular buffer structure to handle the flow of input samples. The circular buffer uses an array and saved indexes to overwrite the oldest sample with the newest sample and to process the inputs in their proper order. This structure provides a way to correctly match input samples with their corresponding filter coefficients without excessive data movement overhead.

FIR Filters have an interesting coefficient mirroring property that allows for significant optimization of the filter algorithm. After generating the coefficients for a particular filter, the coefficients will always be mirrored around the center coefficient. For an example, in an n order filter, the first coefficient C0 is equal to the last coefficient Cn, the coefficient C1 is equal to the coefficient Cn-1, etc. A benefit of this property is that half of the instructions used to load the coefficients into the MAC can be avoided. Instead, each coefficient can be loaded into the MAC and followed sequentially by the two samples that will be multiplied with it. This reduces the data movement operations to the MAC by approximately 25% and offers substantially better filter performance.

Furthermore, some filters have a second property where every other coefficient has a value of zero that allows for even more optimization. This occurs in Half-Band filters which have a frequency response that is symmetric about 1/2 of the Nyquist rate (1/4th of the Sampling Rate). These multiplications with the zero value coefficients do not need to be performed, as the result will just be zero and the accumulated output will not change. Removing these unnecessary multiplications from the filter loop has a substantial impact on execution time.

2.2.2. FIR Filter Example

An application uses an FIR filter to perform a task. For example, a voice application may use a low-pass FIR filter to attenuate frequencies above 4 kHz. To demonstrate the FIR algorithm on the DSP-enabled MCUs, the *FIR_Demo.c* programs measure the frequency response of the filter between 50 Hz and 5 kHz. The programs record the input RMS value and the output RMS value of the filter at the current frequency and print the frequency, input RMS value, and output RMS value to the UART. They then increase the generated frequency and begin again. The programs utilize the IDAC to generate the frequency sweep and use an ADC sampling frequency of 10 kHz. The RMS value for the input and output are calculated and used in the output power calculation.

This application note FIR example code takes advantage of the circular buffer and mirroring optimizations, since these are properties of all FIR filters. The Half-Band property is only applicable to some FIR designs, so this optimization is not included. Figure 1 illustrates the FIR firmware procedure.

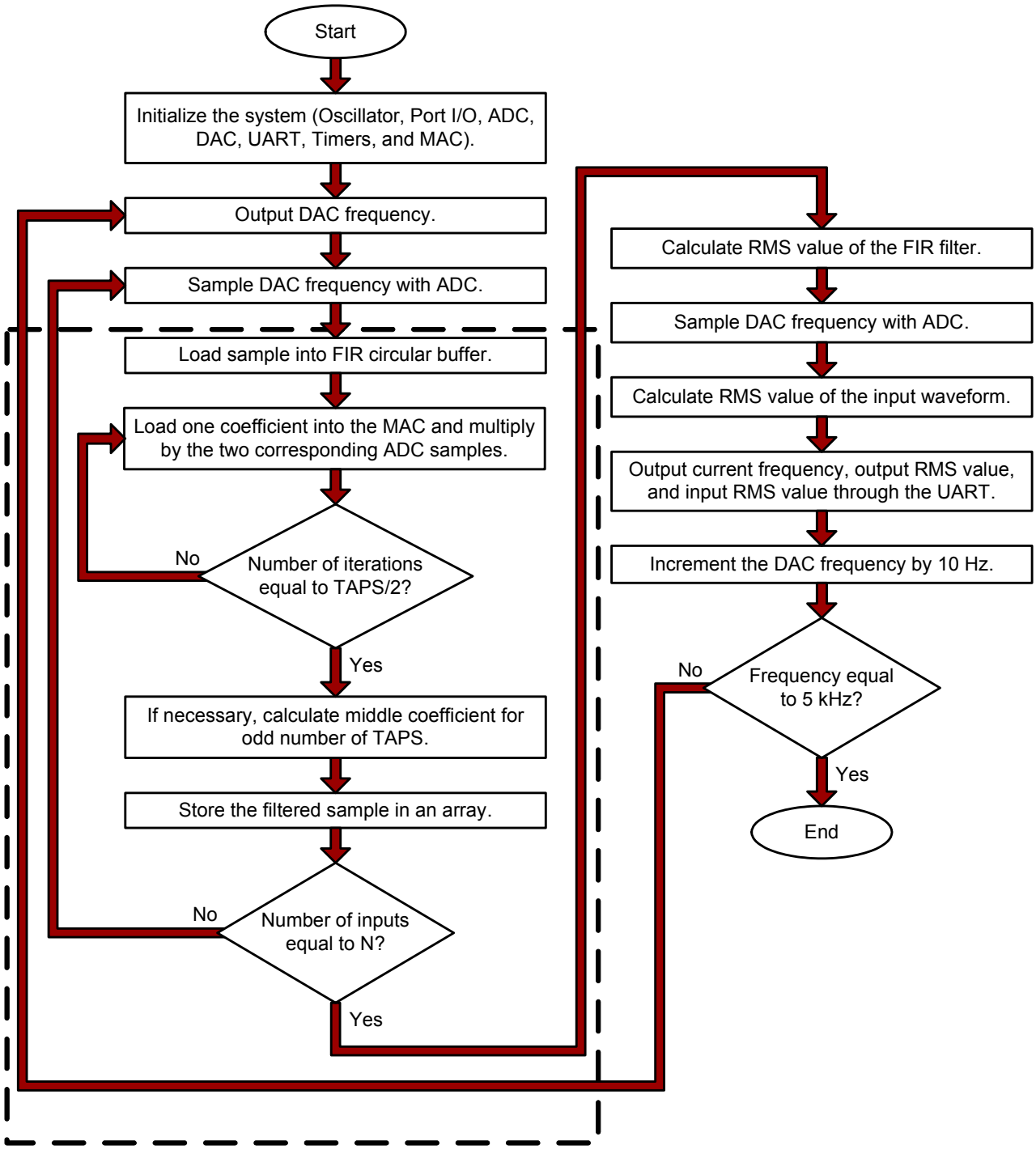


Figure 1. FIR Filter Firmware Flow Diagram

Figures 2 through 5 illustrate the frequency responses of several different filters designed using FDATool (MATLAB) and implemented on the C8051F12x and C8051F36x family of devices. In all cases, the filter response output from the microcontroller matches the filter response designed in FDATool.

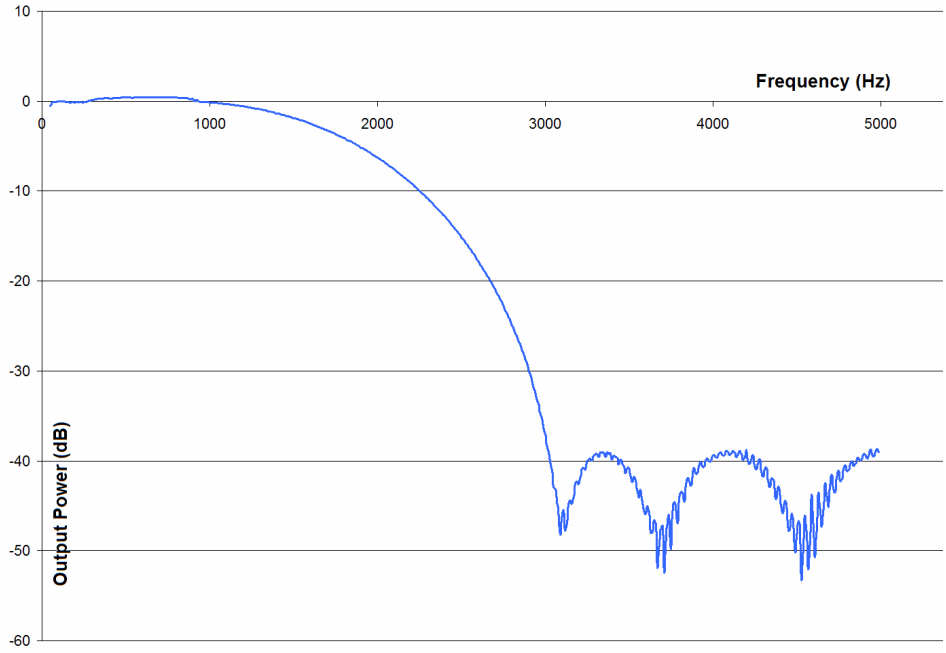


Figure 2. 10th Order Low-Pass Filter

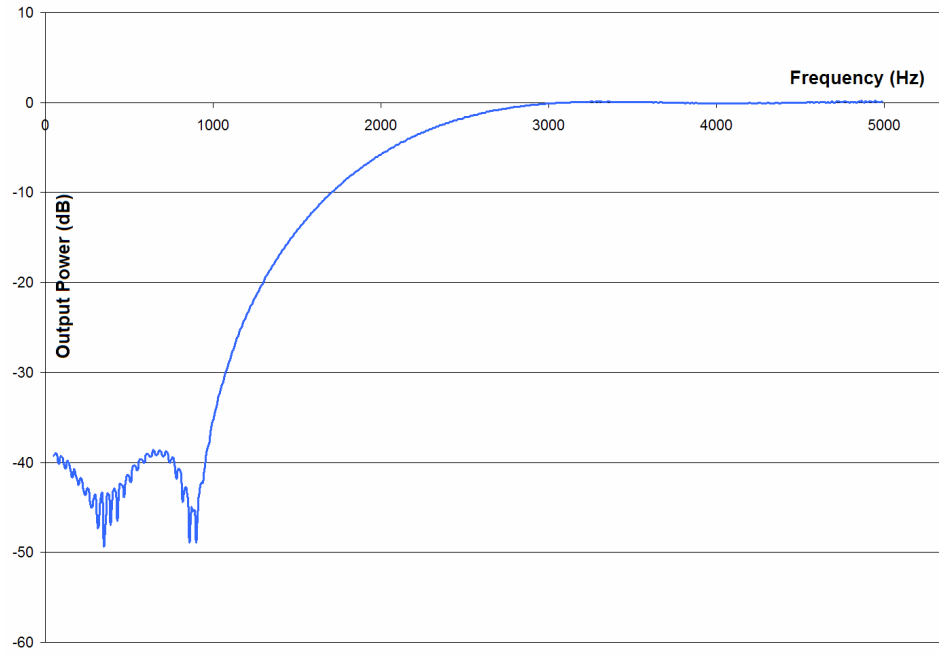


Figure 3. 10th Order High-Pass Filter

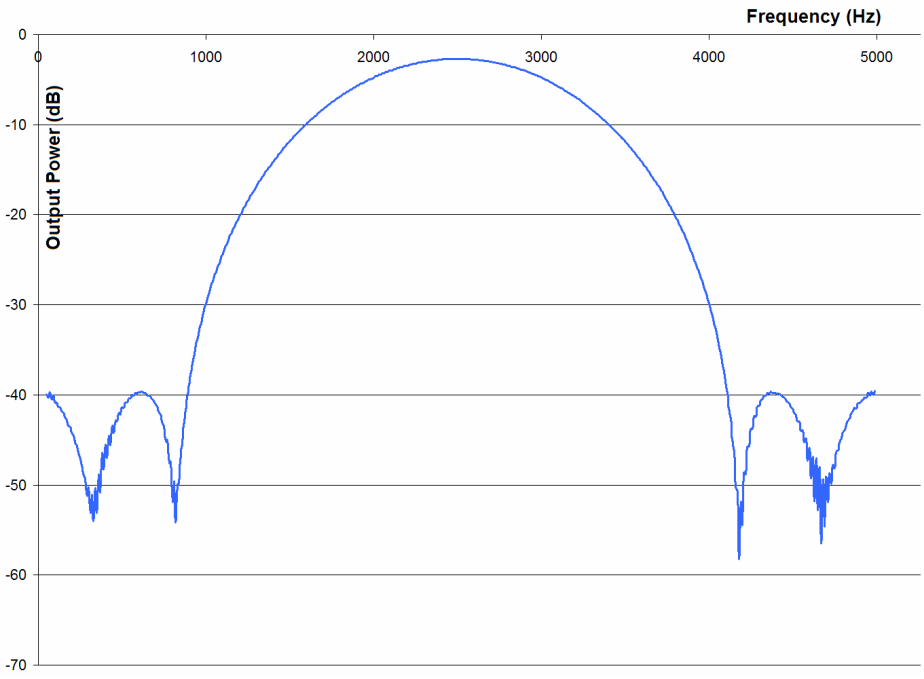


Figure 4. 10th Order Band-Pass Filter

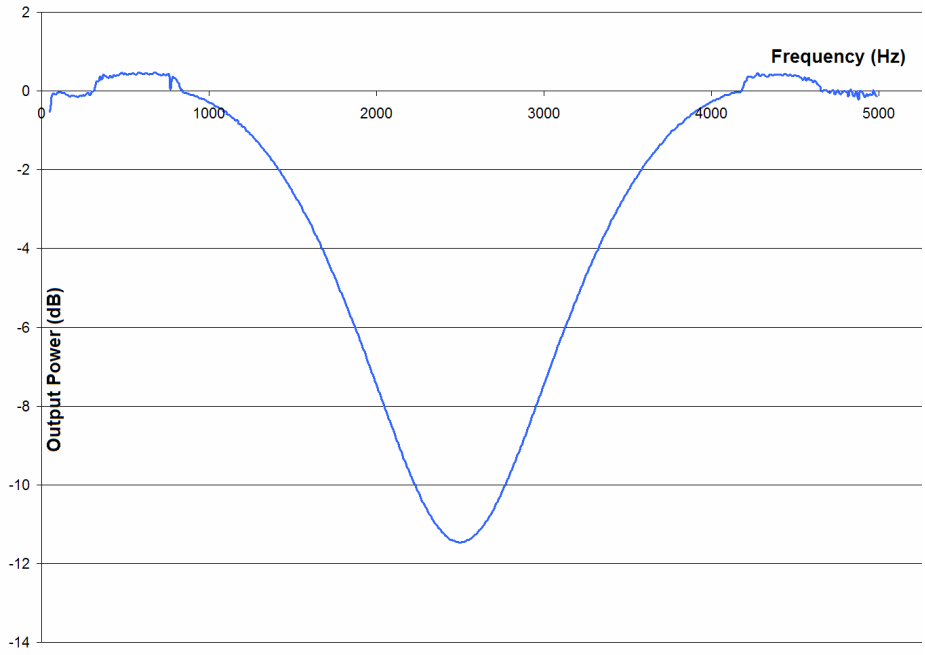


Figure 5. 10th Order Band-Stop Filter

2.3. Running the FIR Demo

Compiling the FIR Demo example code will require either the full version of the Keil compiler (because of the code size and the sqrt() function) or SDCC. This firmware is located in the associated application note software package available on the Silicon Labs MCU Applications webpage.

For the hardware setup specific to the C8051F120 Target Board or C8051F360 ToolStick daughter card, refer to "5. Hardware Setup" on page 22.

To recompile the program, open the Silicon Laboratories IDE and add the appropriate *FIR_Demo.c* file to the project and build. Under the Project→Tool Chain Integration menu, select the compiler and executable paths. The example code is intended for either the C8051F120 Target Board or the C8051F360 ToolStick daughter card, though the programs can be modified for alternate platforms. Build the project, connect to the target device, and download the code.

Connect to the C8051F120 using the RS-232 connector on the Target Board and a terminal program (like HyperTerminal). Connect to the 'F360 ToolStick daughter card board using the ToolStick Terminal program. Save the output to a file and use the *FIR_graph.xls* Excel spreadsheet to graph the filter response (instructions can be found in the spreadsheet).

2.3.1. Performance

Using three low-pass filters of three different orders, the performance of the FIR filter was measured in system clock cycles and the CPU bandwidth used (with the ADC sampling rate of 10 kHz).

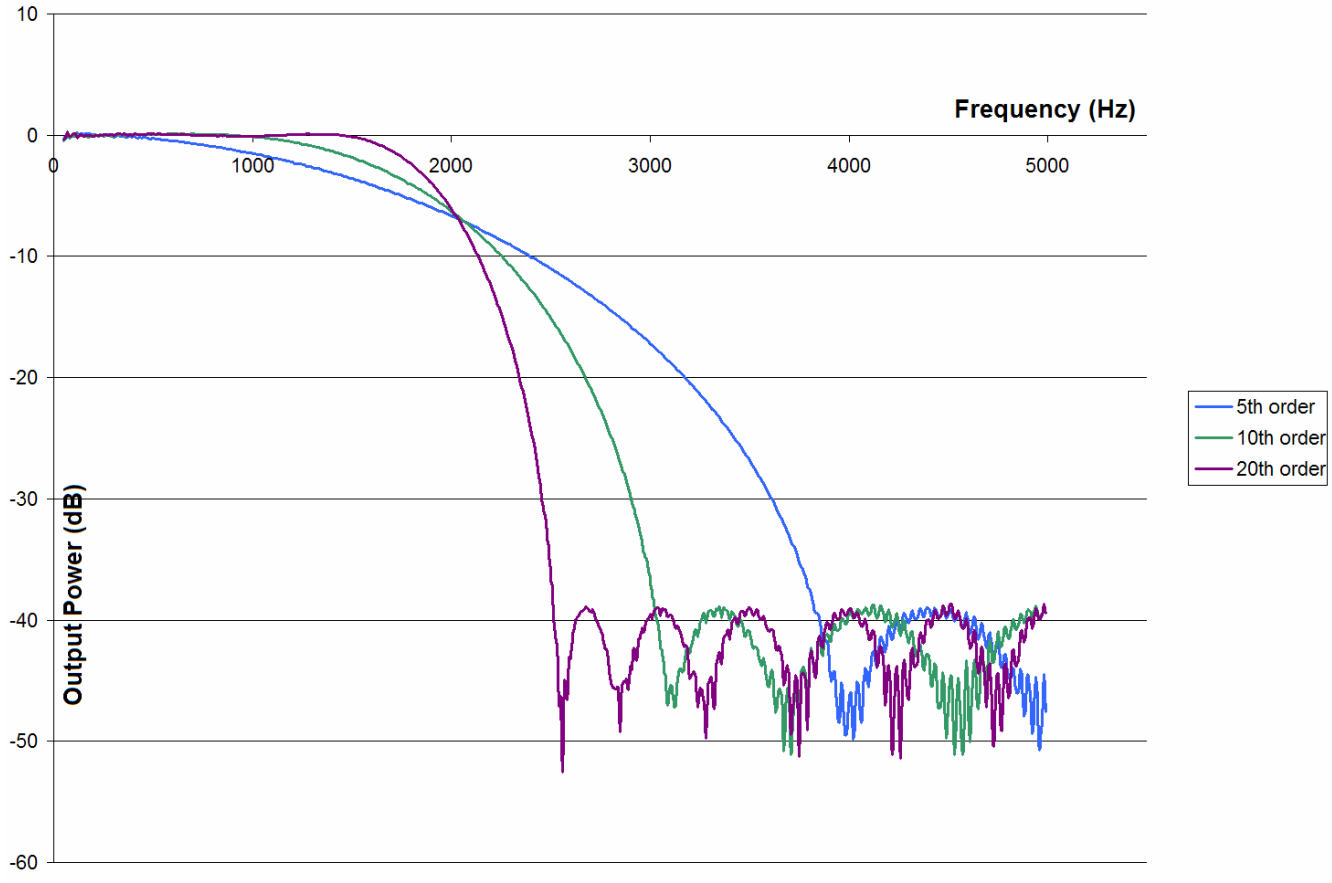


Figure 6. 5th, 10th, and 20th Order Low-Pass Filter Comparison

The performance of the C8051F12x and C8051F36x families of microcontrollers are as follows with the three different filter orders:

Code Segment	25 MIPS MAC implementation			98 MIPS MAC implementation		
	Clock cycles	us	MCU bandwidth	Clock cycles	us	MCU bandwidth
5th order FIR filter	289	11.6	11.6%	289	2.9	2.9%
10th order FIR filter	513	20.5	20.5%	513	5.2	5.2%
20th order FIR filter	913	36.5	36.5%	913	9.3	9.3%

Even with a 20th order filter, the filter only takes 9.3% of the microcontroller bandwidth, allowing the MCU to complete many other tasks or, if no other tasks are required, sleep and conserve power when not in use.

3. Goertzel Algorithm

Many embedded systems are interested in a single or small set of frequencies in an input waveform. The Goertzel Algorithm is a useful tool when these frequencies of interest are known.

The Goertzel Algorithm is a specialized algorithm intended to detect the presence of a single frequency. It is implemented in the form of a two-pole IIR filter, though the derivation comes from a single-bin Discrete Fourier Transform output*.

***Note:** Lyons, Richard. *Understanding Digital Signal Processing.* Second Edition. 2004.

The Goertzel equations are as follows:

$$Q_0 = (coef_k \times Q_1[n]) - Q_2[n] + x[n],$$
$$Q_1 = Q_0[n - 1],$$
$$Q_2 = Q_1[n - 1],$$

where $x[n]$ is the current input, Q_0 is the latest output, Q_1 is the output from the previous iteration, and Q_2 is the output from two iterations ago. The coefficient $coef_k$ is dependant upon certain system parameters like the target frequency and the total number of inputs N. The power of the input waveform at a particular frequency is as follows:

$$Power = magnitude^2 = Q_1 2[\mathrm{N}] + Q_2 2[N] - (coef_k \times Q_1[N] \times Q_2[N])$$

Because of the Discrete Fourier Transform influence in the Goertzel equations, the algorithm does not have a valid output until n, the current input number, is equal to N, which is the total number of inputs used by the algorithm. This means that the output of the filter is not valid until it has processed N input samples.

3.1. Goertzel Algorithm for DTMF Applications

Dual tone multi-frequency (DTMF) uses four frequencies to represent four rows and four frequencies to represent four columns. The grid created by overlaying the row frequencies and column frequencies is the touch-tone telephone keypad. Each button on the keypad is represented by a waveform that is the combination of the row frequency and column frequency. The tones chosen for the row tones and column tones are specifically non-multiples of each other so one tone is not easily mistaken for another.

The tones for DTMF are contained within the range of normal speech. To prevent false positives of DTMF tones during a conversation, the second harmonics of the tones are used. If the input waveform contains a stronger than expected second harmonic of a DTMF frequency, it is most likely that the input waveform is speech instead of a DTMF tone.

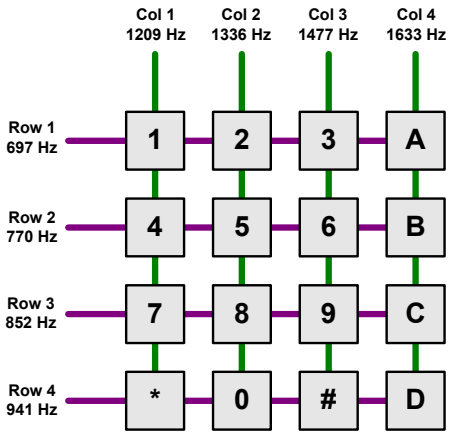


Figure 7. DTMF Keypad

DTMF tone generation is an easy problem that can be solved by stepping through constant SINE tables and adding the tones together. For example, the “5” tone is the combination of Row 2 tone of 770 Hz and the Column 2 tone of 1336 Hz, as shown in Figure 8.

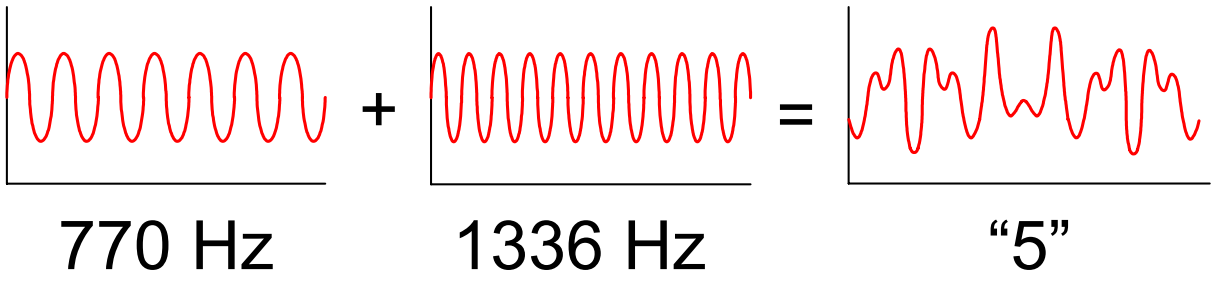


Figure 8. Figure 2.2. DTMF Tone Generation from a Row Tone and Column Tone

DTMF detection, however, requires the system to search for the presence of one row tone and one column tone and to differentiate speech from a pure DTMF tone. Since the Goertzel Algorithm is relatively quick to calculate and doesn’t require storage for past inputs that no longer actively participate in the calculation, it is a viable solution to implementing DTMF decoding.

The Goertzel equations for DTMF applications then become the following:

$$Q_0 = (coef_k \times Q_1[n]) - Q_2[n] + x[n],$$
$$Q_1 = Q_0[n - 1],$$
$$Q_2 = Q_1[n - 1],$$

$x[n]$ = ADC Sample

$k = 0.5 \times ((N \times \text{DTMF\_Target\_Frequency})/\text{Sampling\_Rate})$

N = number of samples per sample set

$coef_k = 2\cos((2 \times k) / N)$

3.2. Goertzel Algorithm for DTMF Implementation on the C8051F12x and C8051F36x

The MAC engine and 100 MIPS core CPU speed enable the C801F12x and C8051F36x to implement the Goertzel Algorithm for DTMF tone detection easily and quickly. The equations for the Goertzel Algorithm are a series of multiplies and additions suited to the MAC.

3.2.1. Implementation Optimizations

Using the Goertzel Algorithm for DTMF requires 16 filters: 8 for the base DTMF frequencies and 8 for the DTMF frequency second harmonics. However, these two sets of filters do not need to be calculated concurrently. If they are separated into two groups, the DTMF tone can be detected sooner and the second harmonic can be checked after the initial tone detection. Additionally, the memory requirements are greatly lessened by separating the two filters, as the storage can be reused between the two sets.

3.2.2. Goertzel DTMF Example

The *DTMF_Demo.c* programs generate the DTMF tones using the on-chip DAC and a constant SINE table. The example code detects the tones using the Goertzel Algorithm in the ADC ISR (Interrupt Service Routine). The DAC updates the output waveforms at 100 kHz and the ADC samples the input at 8 kHz. The program displays a keypad using the UART and requests a tone to generate. When a tone is requested, that tone is generated on the DAC for a set amount of time, and the ADC samples the waveform and determines if a DTMF tone is present. If a tone is detected, an indicator is printed to the UART. In the application example code, the 8 base frequencies and 8 second harmonics are separated into two sets of filters to optimize memory usage. Figure 9 illustrates the Goertzel DTMF firmware procedure.

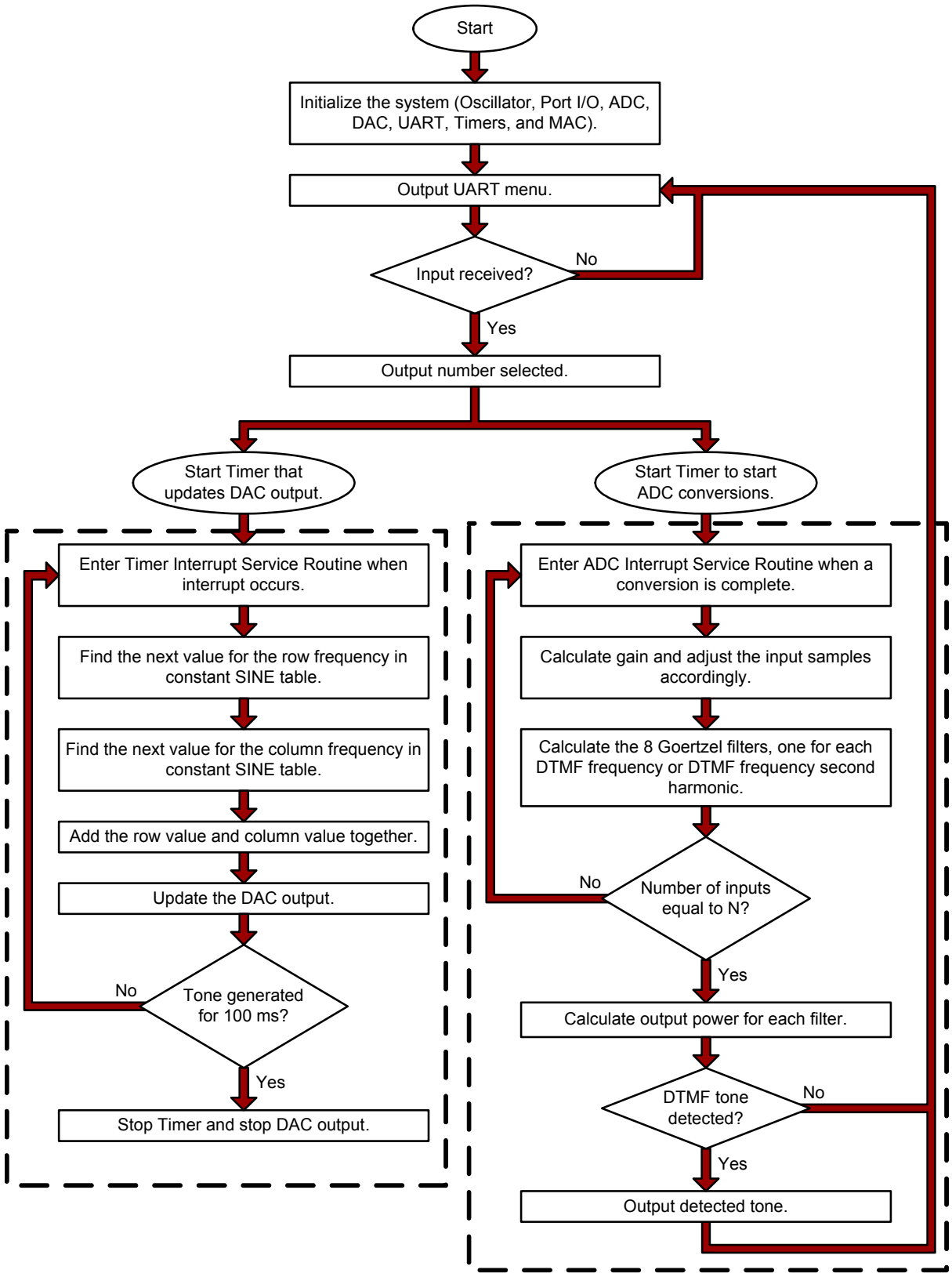


Figure 9. Goertzel DTMF Firmware Flow Diagram

Because the example code generates the DTMF tone and detects it on the same device, there is a synchronization between the systems that would not normally exist in an application. Separate generation and detection code is also provided with this application note for systems where the two actions occur asynchronously on separate platforms.

3.3. Running the Goertzel DTMF Demo

The Goertzel DTMF Demo requires either the full version of the Keil compiler (because the code size is larger than 4 kB) or SDCC. This firmware is located in the associated application note software package available on the Silicon Labs Applications webpage.

For the hardware setup specific to the C8051F120 Target Board or C8051F360 ToolStick daughter card, refer to "5. Hardware Setup" on page 22.

To recompile the program, open the Silicon Laboratories IDE and add the *DTMF_Demo.c* file to the project and build. Under the Project→Tool Chain Integration menu, select the appropriate compiler and executable paths. The project is intended for either the C8051F120 Target Board or the C8051F360 ToolStick daughter card, though it can be modified for alternate platforms. Build the project, connect to the target, and download the code.

Connect to the C8051F120 using the RS-232 connector on the Target Board and a terminal program (like HyperTerminal). Connect to the ‘F360 ToolStick daughter card board using the ToolStick Terminal program. Navigate the UART menu to generate DTMF tones. The UART will output when a tone has been generated and if a tone was detected before reprinting the menu.

3.4. Performance

The MAC engine and 100 MIPS core CPU speed allow the C8051F12x and C8051F36x family of microcontrollers to greatly outperform a 25 MIPS CPU running the same algorithm. This allows the ‘F12x and ‘F36x to calculate the algorithms for eight filters concurrently and in real-time as the inputs are gathered. Furthermore, the ‘F12x and ‘F36x can calculate the power for each frequency in real-time.

Code Segment	25 MIPS non-MAC implementation		98 MIPS non-MAC implementation		98 MIPS MAC implementation	
	Clock cycles	us	Clock cycles	us	Clock cycles	us
8 Goertzel filters in ADC ISR	2095	83.8	2095	21.4	1018	10.4
Power calculations	13,113	524.5	13,113	133.8	1743	17.8
Total time for 200 input samples	432,000	17,285	432,000	4409	205,000	2095

With the normal implementation running at 25 MIPS, the power calculations must occur in the background in between the calculations for each ADC input, and because so much of each sampling period is taken by the calculations themselves, the power calculation results may be rather delayed. However, the ‘F12x and ‘F36x implementation using the MAC and 100 MIPS CPU bandwidth are more than eight times faster and the filters and power calculations can be completed entirely in the ADC ISR (Interrupt Service Routine) before the next ADC sample. Because of this, the power calculations and Goertzel filters don’t take nearly as much time away from other tasks. In a 25 MIPS device, the DTMF application will take most of the bandwidth of the device, but the DSP-enabled microcontrollers can easily complete other tasks.

Furthermore, the normal implementation takes more code space (5355 bytes of code versus 5057 bytes of code) and more RAM (165 bytes versus 126 bytes) to implement than the MAC version of the algorithm.

4. Fast Fourier Transform

The Fourier Transform takes a continuous time-domain signal as its input and calculates the frequency content of the signal. In real systems with ADC inputs, however, the time-domain signal is discrete and not continuous, so the Discrete Fourier Transform (DFT) must be used. The Fast Fourier Transform (FFT) generates the same output as the DFT but much more efficiently.

The FFT takes the input data array and breaks it down into halves recursively until the data is in pairs. Then, the FFT calculates the 2-point FFT for the data and uses the outputs to calculate the 4-point FFT. The outputs of the 4-point FFT are then used to calculate the 8-point FFT, and so forth, until the N-point FFT is complete.

The DFT requires N^2 complex calculations to generate the output, where N is the number of points in the DFT. The FFT, however, only requires $N/2 \times \log_2 N$ complex calculations. As the number of input points to the FFT (N) increases, the FFT efficiency is vastly superior compared to the DFT.

Table 1. DFT and FFT Complex Calculations for Varying N

N (number of input samples)	8	256	1024	8192
DFT (complex calculations)	64	65536	1,048,576	67,108,864
FFT (complex calculations)	12	1024	5120	53,248

The FFT allows for frequency analysis in a system and is a staple of any DSP catalog. Where the FFT is traditionally implemented on DSPs, DSP-enabled MCUs have FFT capability in an embedded system with the flexibility of a general-purpose programmable microcontroller.

4.1. FFT Algorithm

The FFT works by taking an N-point input data array and dividing it into halves recursively until the 2-point data pairs are left. These 2-point pairs are then combined to create the 4-point results, and the 4-point pairs are combined to create the 8-point results, and so forth. As a result, N must be a power of 2 (2, 4, 8, 16, 32, 64, etc.).

The 2-point combination or stage is the basic building block of the FFT. This algorithm is repeated for each proceeding stage. The 2-point “butterfly” is calculated as show in figure.

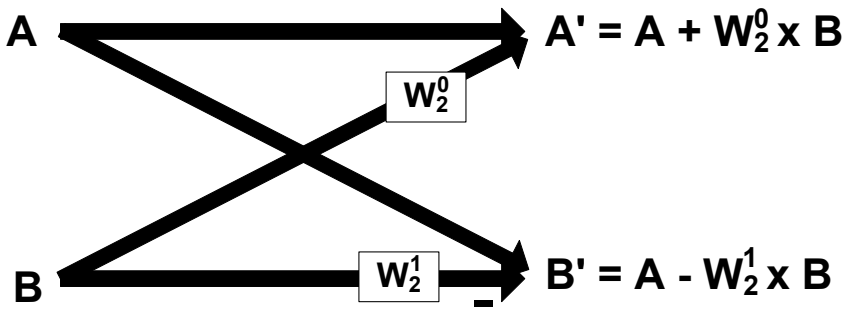


Figure 10. 2-Point FFT Butterfly Structure

The *W* factor shown in the diagram is the “twiddle.” The twiddle is a sine/cosine factor calculated based on the number of points in the current stage. The equation for the twiddle is as follows:

$$W_N^m = e^{-j2n\pi m/N} = \cos(2\pi m/N) - \sin(2\pi m/N)$$

where N is 2 for a 2-point stage and m is 0 to N-1 (so 0 and 1 in this case). Note that *A* and *B* are both complex numbers, so they both contain real and imaginary components.

Two 2-point butterflies are combined to create the 4-point FFT. The *A* and *B* outputs of the first 2-point butterfly become A_1 and A_2, and the *A* and *B* outputs of the second 2-point butterfly become B_1 and B_2. The 4-point FFT is then as shown in Figure 11.

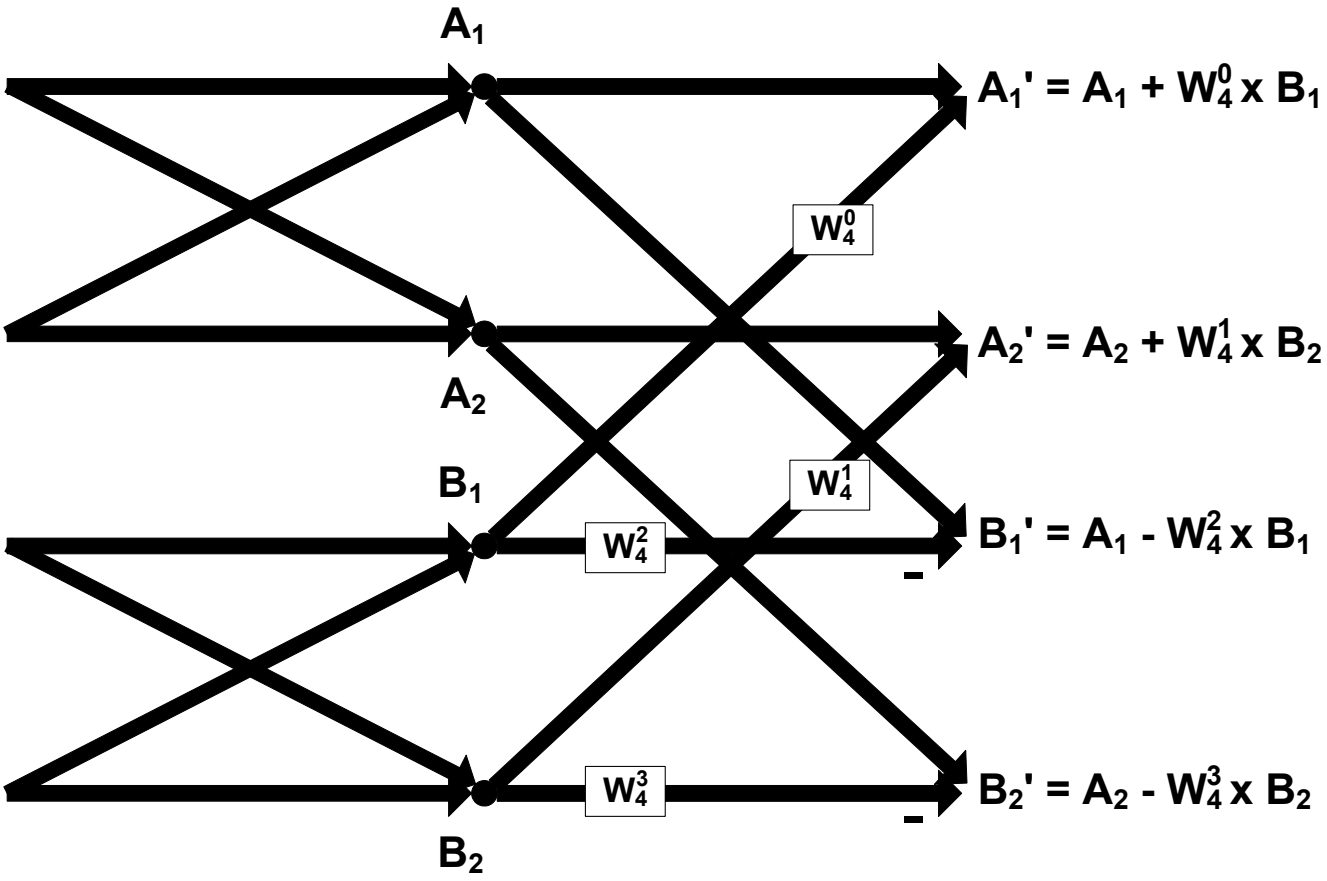


Figure 11. 4-Point FFT Structure

In the case of the 4-point FFT, the *W* factors have m from 0 to 3 and N is 4.

Similarly, the 8-point FFT is a combination of two 4-point FFTs:

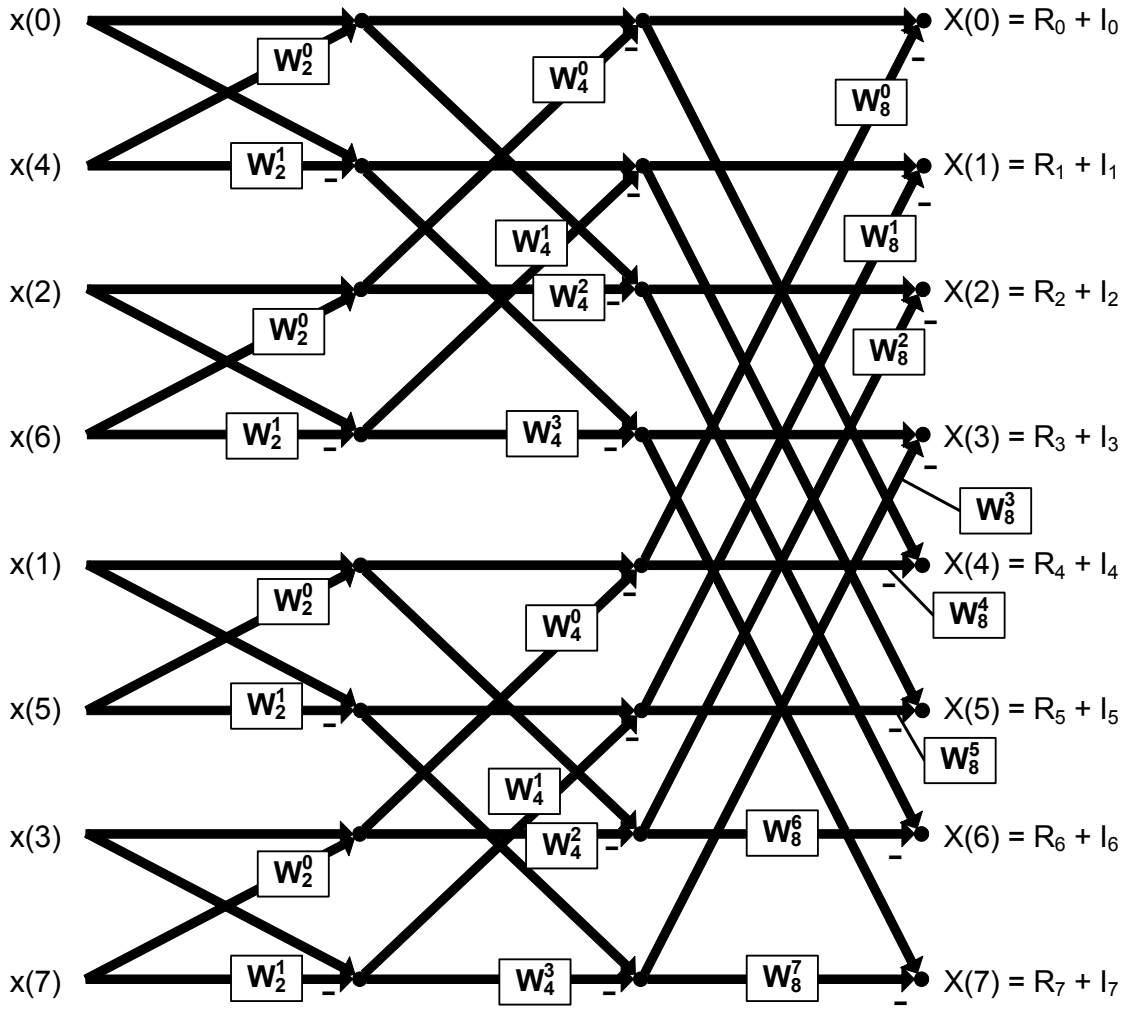
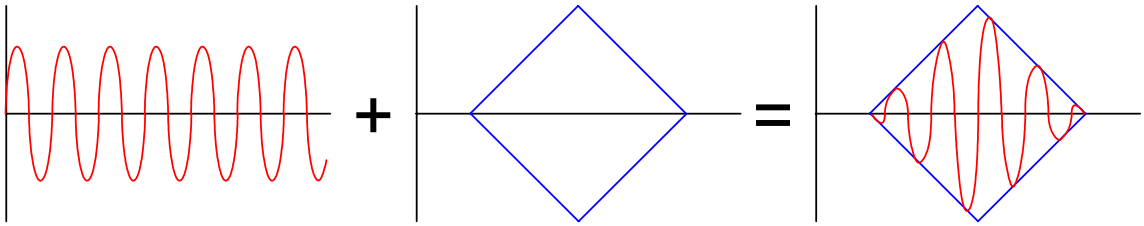


Figure 12. 8-point FFT Structure

For each stage of the FFT, the number of complex data points remains the same (32 data points in the 2-point stage, 32 data points in the 4-point stage), so the FFT is a computationally intense algorithm, especially as N becomes large. Furthermore, any errors in the early stages will compound in the later stages, so the more accurate the calculations, the better the FFT algorithm implementation.

4.1.1. Windowing

If the sampling frequency is not a perfect multiple of the input waveform, the input data set will have a discontinuity between the first data point and the last data point. This discontinuity can cause false energy in the FFT. To remove this, a Window is used that conforms the input waveform to a particular shape. This Window alters the amplitude of the waveform, but does not change the frequency components.



Triangle Window

Figure 13. Windowing the FFT Data to Make Endpoints Continuous

While the Window helps with false energy, one side-effect is the energy tends to spread more between bins. The width of the main lobe and the amplitude of the side lobes differs between different Window functions*. Several examples of Windows are Hamming, Hanning, Blackman, and Triangle.

***Note:** Lyons, Richard. *Understanding Digital Signal Processing.* Second Edition. 2004.

4.1.2. Bit Reversal

The input data in the form of an array is not accessed linearly in the FFT algorithm. The first two values combined in the 2-point butterfly for a 16-point FFT are the data at addresses 0 and 8 (dividing the data in half, 0 and 8 are the first data in each half). The next two addresses that are combined are 4 with 12, then 2 with 10 and 6 with 14.

With a small FFT, calculating the new indexes each pass through is trivial. However, the more complex the FFT, the more time it takes to calculate the indexes. These index calculations can be bypassed if the input array is reordered in a “bit-reversed” fashion. For example, see Figure 14 which shows a 16-point FFT with 16 input data points.

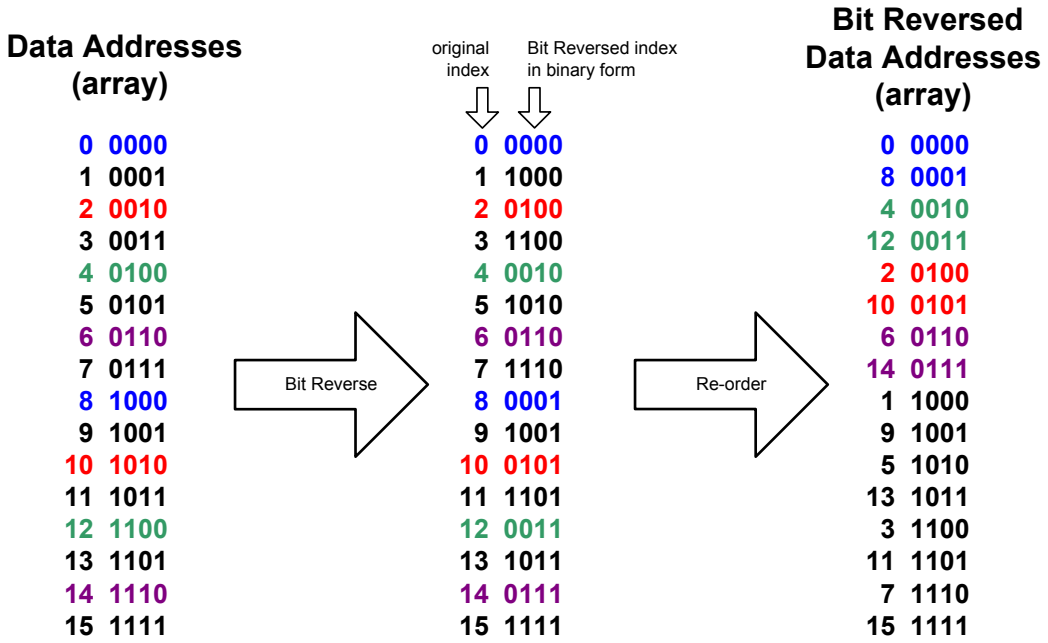


Figure 14. Data Address Bit Reversal

The data pairs that combine in the first several 2-point FFTs are colored in blue (0 and 8), green (4 and 12), red (2 and 10), and purple (6 and 14). The decimal value of the addresses is shown along with the binary form to make clear the original value of the address after the bit reversal. In the bit reversal, bit 3 is swapped with bit 0 and bit 2 is swapped with bit 1, so that an address of ‘1’, or 0001, is translated to an address of ‘8’, or 1000.

Indexing is now reduced to simple linear progression through the array. Notice that all the even addresses appear at the beginning of the array and all the odd addresses appear at the end.

4.1.3. Interpreting the FFT Output

The FFT output is a series of “bins” that represent the amount of energy in a frequency band. Each bin represents a cycles-per-interval value. For example, with the following 64-point FFT output:

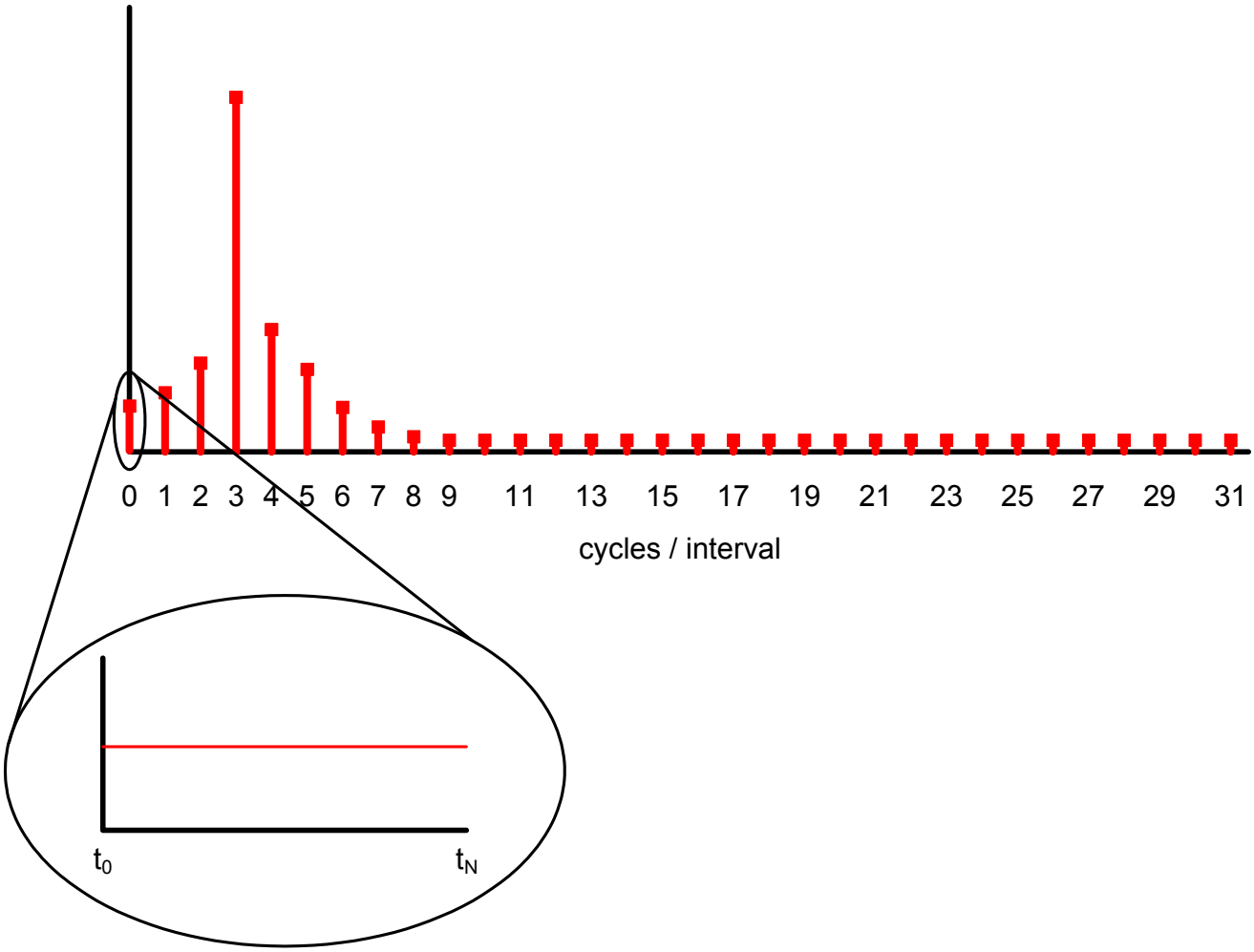


Figure 15. Bin 0 of an FFT output

The 0 bin represents 0 cycles/interval or a dc value. Similarly, the 1 bin represents 1 cycle/interval.

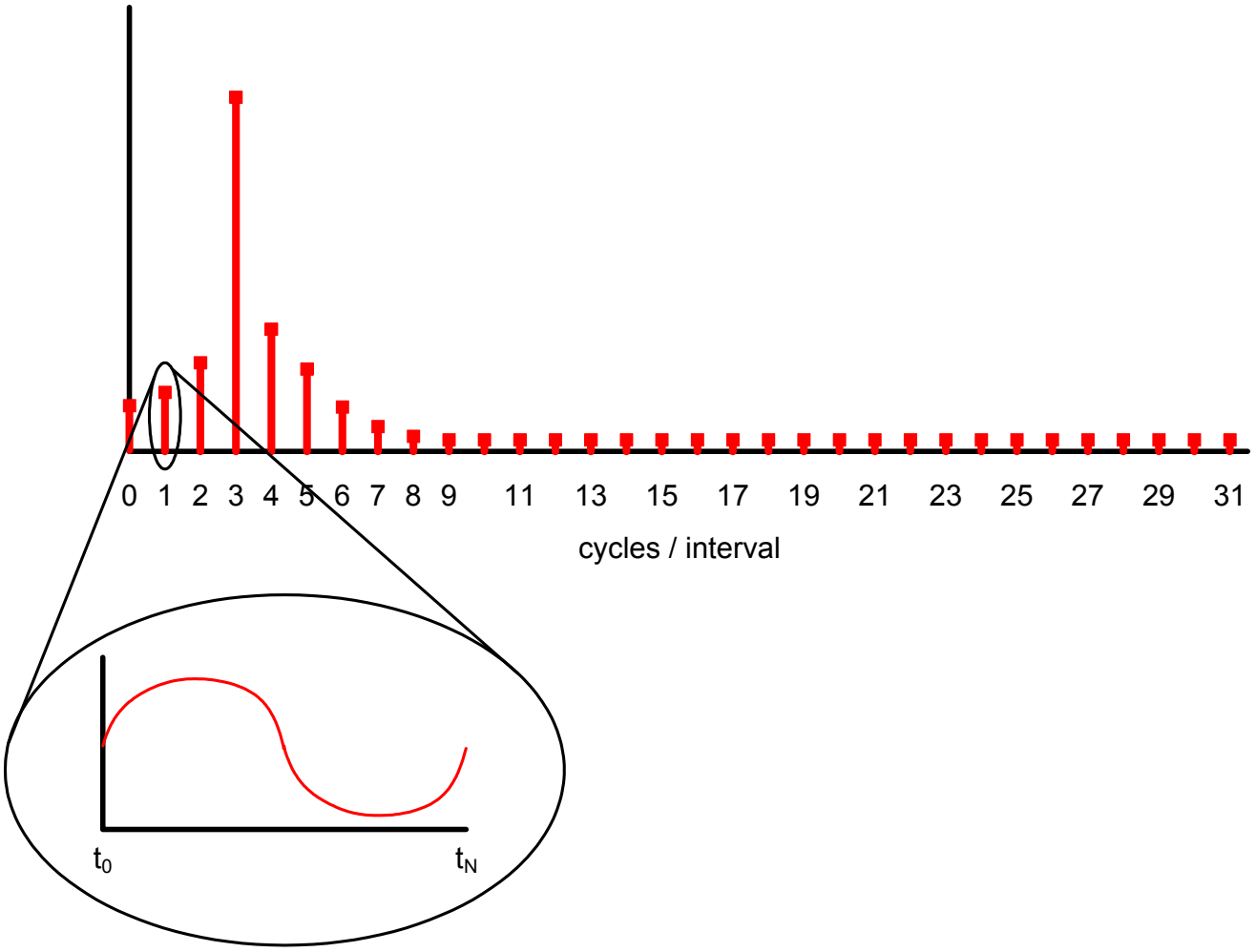


Figure 16. Bin 1 of an FFT output

The interval is the total time represented by the samples, or N x $t_{sampling}$, where $t_{sampling}$ is the sampling period. Each bin, then, represents a frequency that's a fraction of the sampling frequency. The bins can be converted to frequency by the following equation:

$$f_{bin} = (bin/N) \times f_{sampling}$$

4.2. FFT Algorithm Implementation on the C8051F12x and C8051F36x

The C8051F12x and C8051F36x, with their MAC engine and 100 MIPS peak CPU, can process the Windowing and FFT calculations much more quickly than many 8051 platforms. The Windowing routines involve multiplying the input data points by a set of constants, which can be made faster using the MAC. The FFT algorithm is a set of additions and multiplications that is also suited to the MAC.

4.2.1. Implementation Optimizations

In an unoptimized FFT, the twiddle would be calculated for every complex calculation. However, there are many cases in the FFT where the sine or cosine functions are 0, 1, or –1. In these instances, a large speed savings is reached by optimizing the equations beforehand to remove the twiddle calculation and the zero terms. For example, the sine terms are always zero in a 2-point FFT and can be removed from the 2-point calculations.

4.2.2. FFT Example

The example code generates a waveform using the IDAC and samples 256 data points of that waveform using the ADC at 10 kHz for a 256-point FFT. After the inputs are sampled, the data is run through the Windowing routine using the Blackman Window, the Bit Reversal routine, and the FFT algorithm. The final outputs (Real and Imaginary) are displayed using the UART along with the first half of the bin numbers (up until 1/2 $f_{sampling}$). Figure 17 displays the FFT firmware procedure.

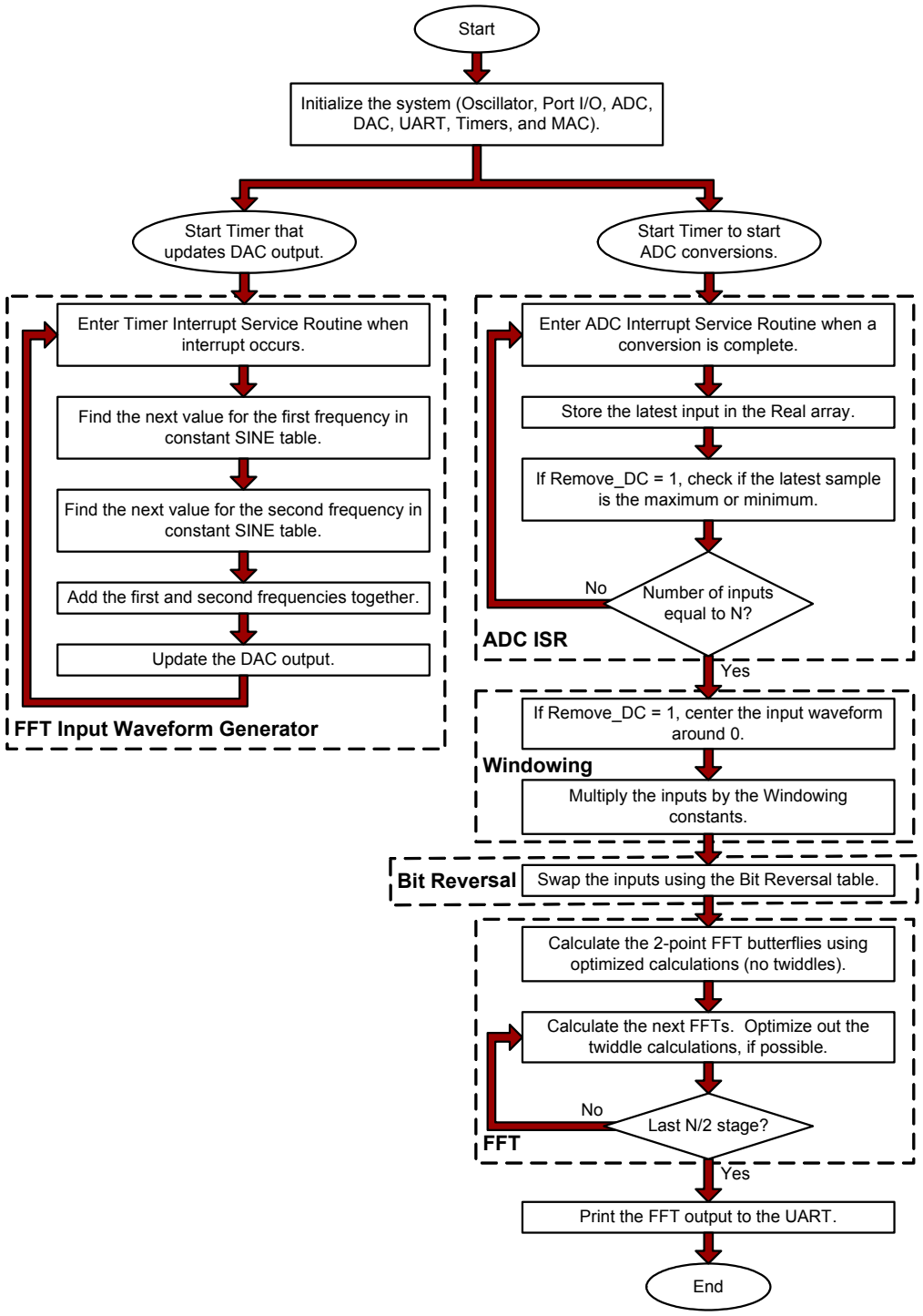


Figure 17. FFT Firmware Flow Diagram

The sampled waveform in *FFT_Demo.c* is a combination of 770 Hz and 1336 Hz. The graphed output from Excel is as shown in Figure 18.

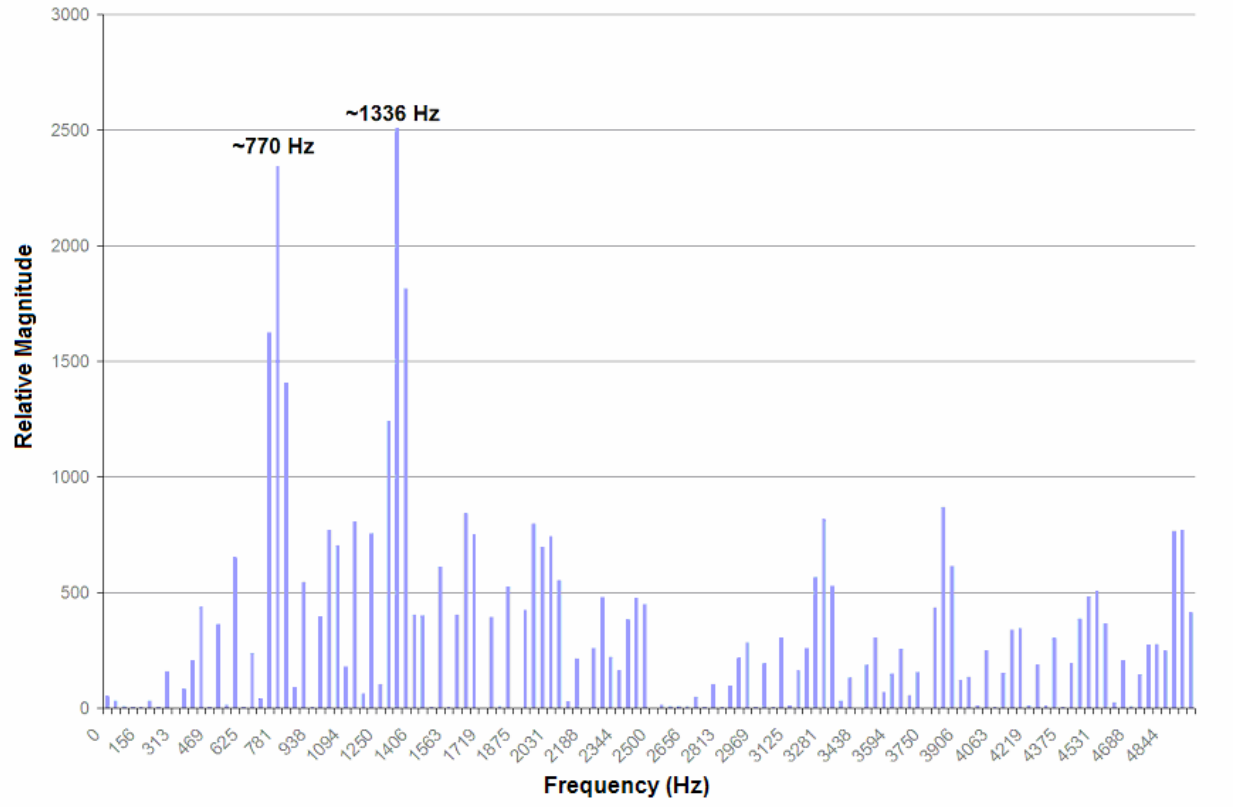


Figure 18. FFT output from 256-point FFT with 770 Hz and 1336 Hz Input

The FFT output indicates that 770 Hz and 1336 Hz tones were properly detected in the input waveform.

4.3. Running the FFT Demo

The FFT Demo requires either the full version of the Keil compiler (because the code size is larger than 4 kB) or SDCC. This firmware is located in the associated application note software package available on the Silicon Labs Applications webpage.

For the hardware setup specific to the C8051F120 Target Board or C8051F360 ToolStick daughter card, refer to "5. Hardware Setup" on page 22.

To recompile the program, open the Silicon Laboratories IDE and add the *FFT_Demo.c* file to the project and build. Under the Project→Tool Chain Integration menu, select the appropriate compiler and executable paths. The project is intended for either the C8051F120 Target Board or the C8051F360 ToolStick daughter card, though it can be modified for other platforms. Build the project, connect to the target, and download the code.

Connect to the C8051F120 using the RS-232 connector on the Target Board and a terminal program (like HyperTerminal). Connect to the ‘F360 ToolStick daughter card board using the ToolStick Terminal program. Save the output to a file and use the *FFT_graph.xls* Excel spreadsheet to graph the frequency content of the input waveform (instructions can be found in the spreadsheet).

4.4. Performance

On the C8051F120 and C8051F360, the MAC implementation of the FFT is almost 8 times faster than the non-MAC implementation at 25 MHz and almost 2 times faster than the non-MAC implementation at 98 MHz.

Code Segment	25 MIPS non-MAC implementation		98 MIPS non-MAC implementation		98 MIPS MAC implementation	
	Clock cycles	ms	Clock cycles	ms	Clock cycles	ms
Windowing	107,916	4.3	107,916	1.1	75,583	0.8
Bit Reversal	29,800	1.2	29,800	0.3	29,800	0.3
FFT	1,467,071	58.7	1,467,071	15.0	819,009	8.3
Total	1,604,842	64.2	1,604,842	16.4	924,433	9.4

In addition, the non-MAC implementation requires 5259 bytes of code space and 91 bytes of RAM. The MAC implementation only requires 4616 bytes of code space and 67 bytes of RAM. Not only is the MAC implementation quite a bit faster, but it also requires fewer resources.

5. Hardware Setup

The hardware setup descriptions apply to all of the different examples provided in this application note.

5.1. C8051F120 Target Board Instructions

Connect the DAC0 and AIN0.0 pins together using the J11 shorting block (DAC0 is pin 3 and AIN0.0 is pin 7 on J11). Additionally, verify the 4.7 µF C19 and 0.1 µF C22 VREF capacitors are populated and J22 connects the output on the VREF pin to the VREF0 and VREFD input pins.

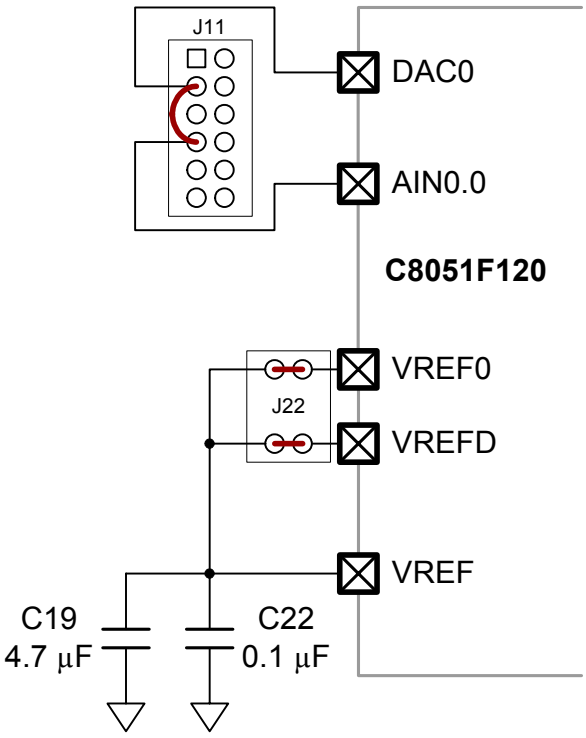


Figure 19. C8051F120 Target Board Hardware Configuration

5.2. C8051F360 ToolStick Daughter Card Instructions

Connect the P0.1 and P1.1 pins together using the testpoints on the board. Verify R7 (1 kΩ resistor), C5 (0.1 µF capacitor), and C1 (10 µF capacitor) are populated.

Important Note: When using the C8051F36x ToolStick for the FIR filter example, make sure the C5 0.1 µF capacitor on the IDAC pin (P0.1) is replaced with a 100 pF capacitor so the IDAC sine wave is not attenuated.

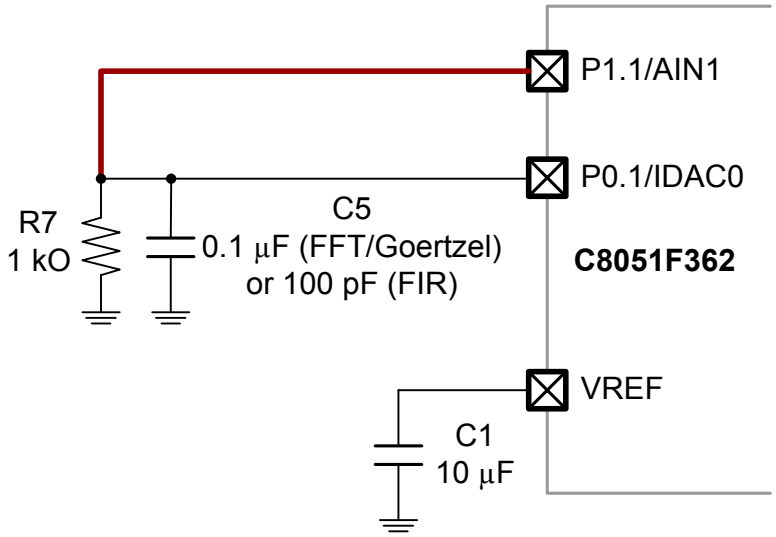


Figure 20. C8051F360 ToolStick Daughter Card Hardware Configuration

DOCUMENT CHANGE LIST

Revision 0.1 to 0.2

- Corrected table units in Section "2.3.1. Performance" on page 7 and Section "3.4. Performance" on page 12.

NOTES:

CONTACT INFORMATION

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Email: MCUinfo@silabs.com
Internet: www.silabs.com

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.

Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders.