SmartFusion cSoC: DC Fan Control Using PWM

Table of Contents

Introduction

The SmartFusion® customizable system-on-chip (cSoC) device contains a hard embedded microcontroller subsystem (MSS), programmable analog circuitry, and FPGA fabric consisting of logic tiles, static random access memory (SRAM), and phase-locked loops (PLLs). The MSS consists of a 100 MHz ARM® Cortex™-M3 processor, advanced high-performance bus (AHB) matrix, system registers, Ethernet MAC, peripheral DMA (PDMA), real-time counter (RTC), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), fabric interface controller (FIC), the Philips Inter-Integrated Circuit (I2C), serial peripheral interface (SPI), external memory controller (EMC), embedded flashROM (eFROM), Watchdog Timer, 10/100 Ethernet Controller, GPIO block, in-application programming (IAP) and system registers. The programmable analog block contains the analog computing engine (ACE) and the analog front-end (AFE), consisting of ADCs, DACs, active bipolar prescalers (ABPS), comparators, current monitors, and temperature monitor circuitry.

This application note describes how to control 2-, 3-, and 4-wire fans on the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board. Using this design example you can control and monitor up to 4 fans. It also supports low frequency and high frequency fan drive signal.

A basic understanding of the SmartFusion design flow is assumed. Refer to the *Using UART with SmartFusion along with Libero SoC v10.0 User's Guide* to understand the SmartFusion design flow.

Overview of Fans

In recent years there has been a growing interest in integrated circuits for controlling the speed of cooling fans used in various electronic equipments.

Day by day the size of electronic devices, especially consumer electronic devices, is decreasing significantly, but the amount of heat generated by the chip due to the complexity of applications running on it is increasing. There are two ways to remove the generated heat. One method is using passive components such as heat sinks and heat pipes and the second is using active components such as cooling fans. In many consumer and other equipments it has been observed that passive components are inefficient due to size and cost. The effective way of removing heat is by using cooling fans. Cooling fans generate air flow around the heat generating sources.

Adding cooling fans to the system will increase power consumption. This is an important factor when the system is running on battery. The cooling fans are noisy as they are mechanical components.

The above problems can be effectively managed by controlling the speed of the fan. Running the fan at lower speed consumes less power, increases the battery life, reduces the noise emitted by the fan, and increases the life span of the fan.

Methods of Speed Control

1. ON/OFF control: The simplest method is ON/OFF control. The fan can be switched ON using a drive signal when the temperature of the particular component exceeds the threshold or it can be switched OFF when the temperature is under the threshold level. Even though the method is very simple, the disadvantage is that the moment it is switched ON, the fan starts running at full speed and thus consumes a lot of power, additionally creating noise.

2. Linear control: In this method the DC voltage applied to the fan is controlled though a voltage regulator. To run the fan at a lower speed the voltage is decreased, and to run at a higher speed the voltage is increased. The advantage of this method is that voltage will always be applied to the fan. Thus, at any given time the tachometer (tacho) signal can be present and measured. This is not possible in the pulse width modulator (PWM) control technique. The disadvantages of this method are that using a voltage regulator will increase the cost and space of the board, and a dedicated controller is required to control the voltage regulator.

3. PWM control: The most widely used method for controlling the fan speed is PWM control. In this method a PWM drive signal is applied to a field-effect transistor (FET), which is connected to the high or low side of the fan. The fan will be on/off at a particular frequency and the speed of the fan is controlled by the duty cycle of the PWM signal. In this method, the voltage applied to the fan is always either full or zero. The biggest advantages of this method are simplicity of design, less external circuitry, and less cost. The disadvantage of this method is that the tacho signal will be chopped by the PWM drive signal as the voltage is not always applied. In fact, the tacho signal will follow the PWM signal. To get around this problem, you should adopt the "pulse stretching" technique to get the tacho information.

Figure 1 shows the tacho signal chopped/altered from the actual tacho signal due to the PWM drive signal.
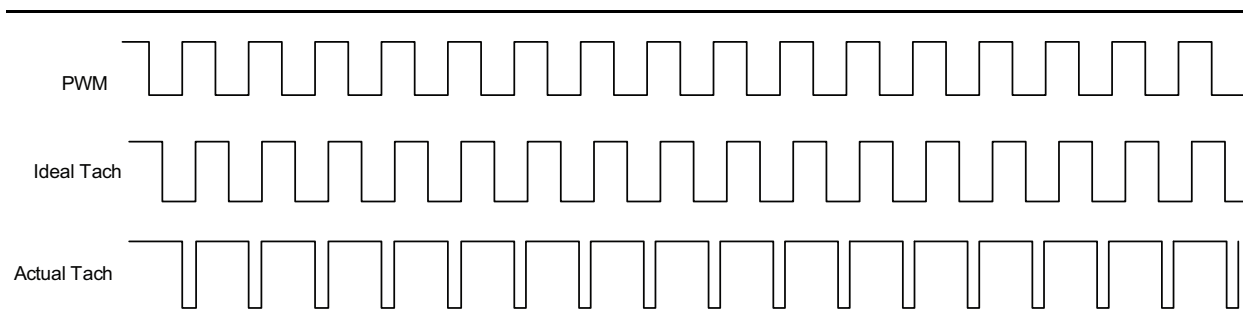
Figure 1 • **Tachometer Pulses**

Figure 2 shows the reconstruction of tacho signal using the pulse stretching mechanism from the actual tacho signal.
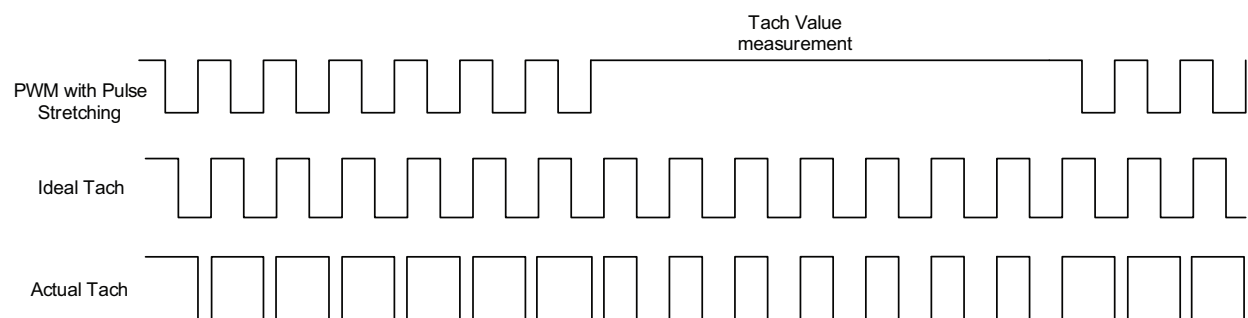
*Figure 2* • **Reconstruction of Tachometer Pulses**

Types of DC Fans

There are three main types of DC fans:

1. 2-wire fan
2. 3-wire fan
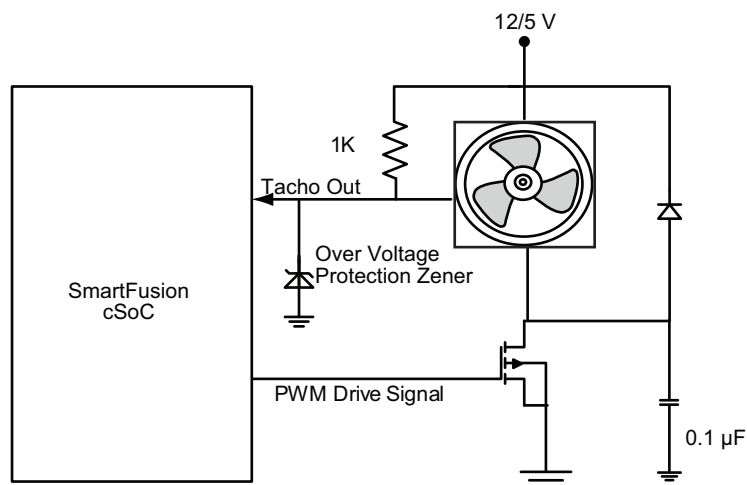3. 4-wire fan

2-Wire Fan

The 2-wire fan has two pins: power, and ground. This fan can be controlled either by varying the DC voltage or by using the low frequency PWM drive signal. The 2-wire fan does not have a tacho signal, so there is no information available to measure the fan speed. This kind of control is called "open loop." Figure 3 shows an example of possible 2-wire fan circuitry with SmartFusion cSoCs.
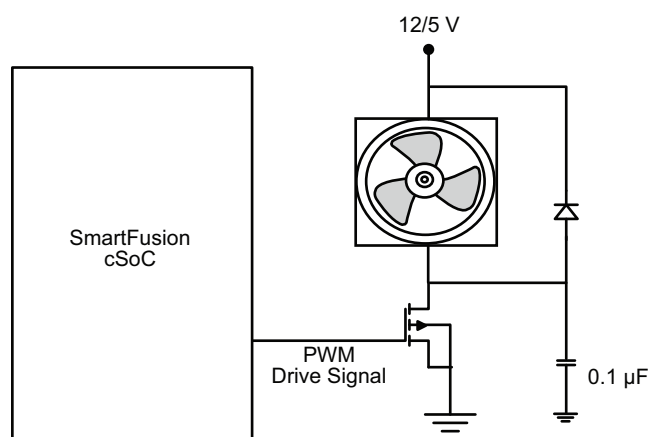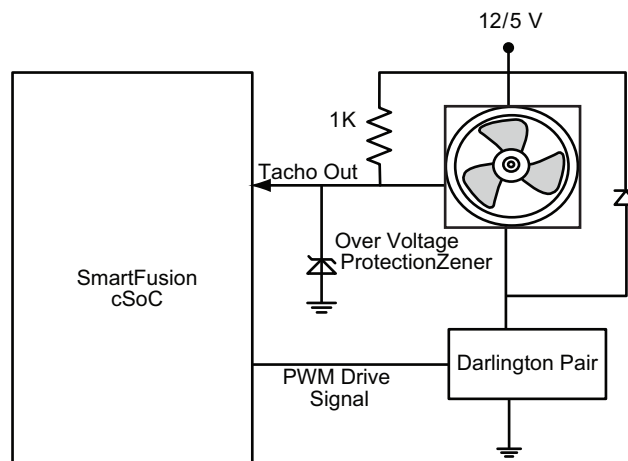

Figure 3 • **2-Wire Fan Speed Control Circuit Using SmartFusion cSoC**

3-Wire Fan

The 3-wire fan has three pins: power, ground, and tachometer. This fan can be controlled either by varying the DC voltage or by using the low frequency PWM drive signal. The 3-wire fan has a tacho signal, which indicates the speed of the fan. Based on the tachometer readings, you can adjust the speed of the fan, which is called "closed loop" control. In some scenarios the 3-wire fan can also be controlled in an open loop. Figure 4 and Figure 5 show examples of possible 3-wire fan circuitry with SmartFusion cSoCs.

Figure 4 • **3-Wire Fan Speed Control Circuit Using SmartFusion cSoC and FET**
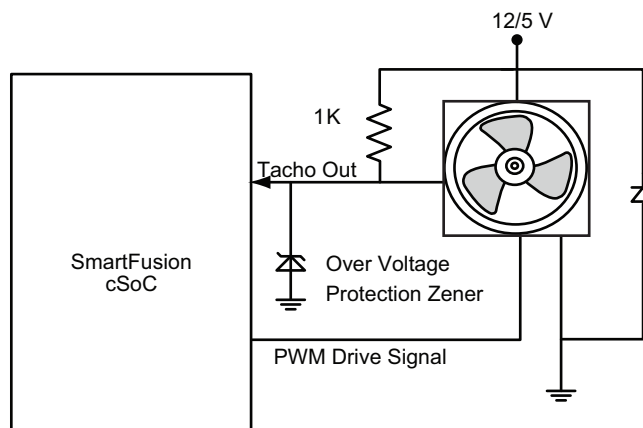

Figure 5 • **3-Wire Fan Speed Control Circuit Using SmartFusion cSoC and Darlington Pair**

4-Wire Fan

The 4-wire fan has four pins: power, ground, tachometer, and PWM input. PWM input is used to control the fan speed. In a 4-wire fan, instead of power on/off to the entire fan, power is provided only to the coils, thus making tacho information available at any point of time.

In this type of fan, the pulse stretching technique is not required. To reduce the commutation noise, the PWM frequency can be increased to more than 20 KHz. With this, the noise made by the coils while commuting will be out of audible range.

Figure 6 shows an example of possible 4-wire fan circuitry with SmartFusion cSoCs.

Figure 6 • **4-Wire Fan Speed Control Circuit Using SmartFusion cSoC**

Design Overview

SmartFusion cSoCs are the only devices that can integrate FPGA fabric, 32-bit ARM Cortex-M3 processor, programmable analog, and rich interfaces. These unique features make the SmartFusion cSoC an ideal choice for DC fan speed monitoring and control applications.

This application note is intended to highlight the unique features of SmartFusion cSoCs in demonstrating how to control different types of DC fans. Using this application note you can control 2-, 3-, and 4-wire fans. This application note provides detailed information on how to control the different DC fans and how to customize the design for specific requirements.

Features

1. Controls 2-/3-/4-wire fans
2. Supports up to 8-pole fan
3. Controls and monitors up to 4 fans
4. Configurable PWM frequency to support High/Low frequency drive signal
5. Configurable minimum PWM duty cycle to support different fans
6. Open loop/Closed loop support
7. Full ON/OFF support
8. GUI to configure the fan (through UART)

This design is implemented using the SmartFusion MSS and FPGA fabric. The main modules in this design are given below:

- Configuration and speed controller
- PWM controller
- Tachometer
- Digital filter

Configuration and Speed Controller

This module is implemented in the software, which will run on the SmartFusion Cortex-M3 processor. The configuration and speed controller does two basic tasks: getting the configuration data from the GUI application and setting the appropriate registers, and controlling the speed of the fan.

The speed controller configures the PWM, which is implemented in the FPGA fabric, and gets the tachometer measurements from the tachometer measurement unit, which is also implemented in the FPGA fabric. Based on the configuration it automatically controls the fan speed. Supported features are given below:

- **Number of poles**: This specifies the number of tacho pulses per revolution. The current implementation supports up to 8 poles.
- **PWM frequency**: This specifies the frequency of the fan drive signal; this can be configured to different frequencies to accommodate the different types of fans, based on the external circuitry.
- **Maximum RPM**: This specifies the maximum RPM of the fan; this is used in the calculation of PWM duty cycle for the specified RPM in the open loop configuration.
- **Minimum Duty cycle**: This specifies the minimum PWM duty cycle required to turn on the fan. This varies from fan to fan because the fan's high-/low-side is connected to the FET/BJT drive circuit.
- **2-, 3-, 4-wire support**: Based on the selection, the speed controller manages the operation and control of a particular fan. For a 2-wire fan the controller calculates and sets the PWM duty cycle based on the maximum RPM and minimum PWM duty cycle, assuming the speed of the fan is related linearly to the PWM duty cycle.

  For a 3-wire fan, the controller does the pulse stretching, reads the tachometer value, recalculates and then sets the PWM duty cycle; this process continues forever in the closed loop. In the open loop, the controller calculates and sets the PWM duty cycle based on the maximum RPM and minimum PWM duty cycle, assuming that the speed of the fan is related linearly to the PWM duty cycle.

  For a 4-wire fan, the controller reads the tachometer value, recalculates, and sets the PWM frequency. This process continues forever in the closed loop. In the loop, the controller calculates and sets the PWM duty cycle based on the maximum RPM and minimum PWM duty cycle, assuming the speed of the fan is linear to the PWM duty cycle.

- Control methods are the following:
  - **Open loop**: In this method, the PWM duty cycle for the desired RPM is calculated based on the maximum RPM and minimum PWM duty cycle, assuming that the speed of the fan is related linearly to the PWM duty cycle, and that there will be no correction to the PWM duty cycle based on the tacho measurement feedback.
  - **Closed loop**: In this method, the PWM duty cycle for the desired RPM is calculated based on the feedback from the tacho measurement. This is applicable only for 3-wire and 4-wire fans.
  - **Full ON**: This turns the fan on at full RPM.
  - **Full OFF**: This switches the fan off.

Customizations

Unlike the fan controller ASICs, which are dedicated for defined functionality, SmartFusion cSoCs have a rich set of peripherals with programmable analog and programmable FPGA fabric. This unique feature enables you to implement the additional functionalities required for a specific application.

- Fan speed controller based on temperature using the analog block
- Configuration and control through SPI/I^{2}C/Ethernet or any custom protocol using FPGA fabric
- Ease in adding the custom fault indications
- Ease in increasing the number of monitoring channels at software level
- Ease in increasing the number of control channels using FPGA fabric
- Data logging of fan to determine the system's functionality with respect to temperature and power
- Calibration of fan to determine the fan health

Applications

- Networking
- Desktop and notebook PCs
- Projectors
- Telecommunications
- Embedded systems

Design Description

This design can be used to drive various types of fans. Using the unique features of the SmartFusion cSoC, the design is partitioned into software and hardware, where the dedicated functions such as PWM tacho measurement are implemented in hardware and the configuration and controller functionalities are implemented in software. Figure 7 illustrates the architecture of the fan controller.
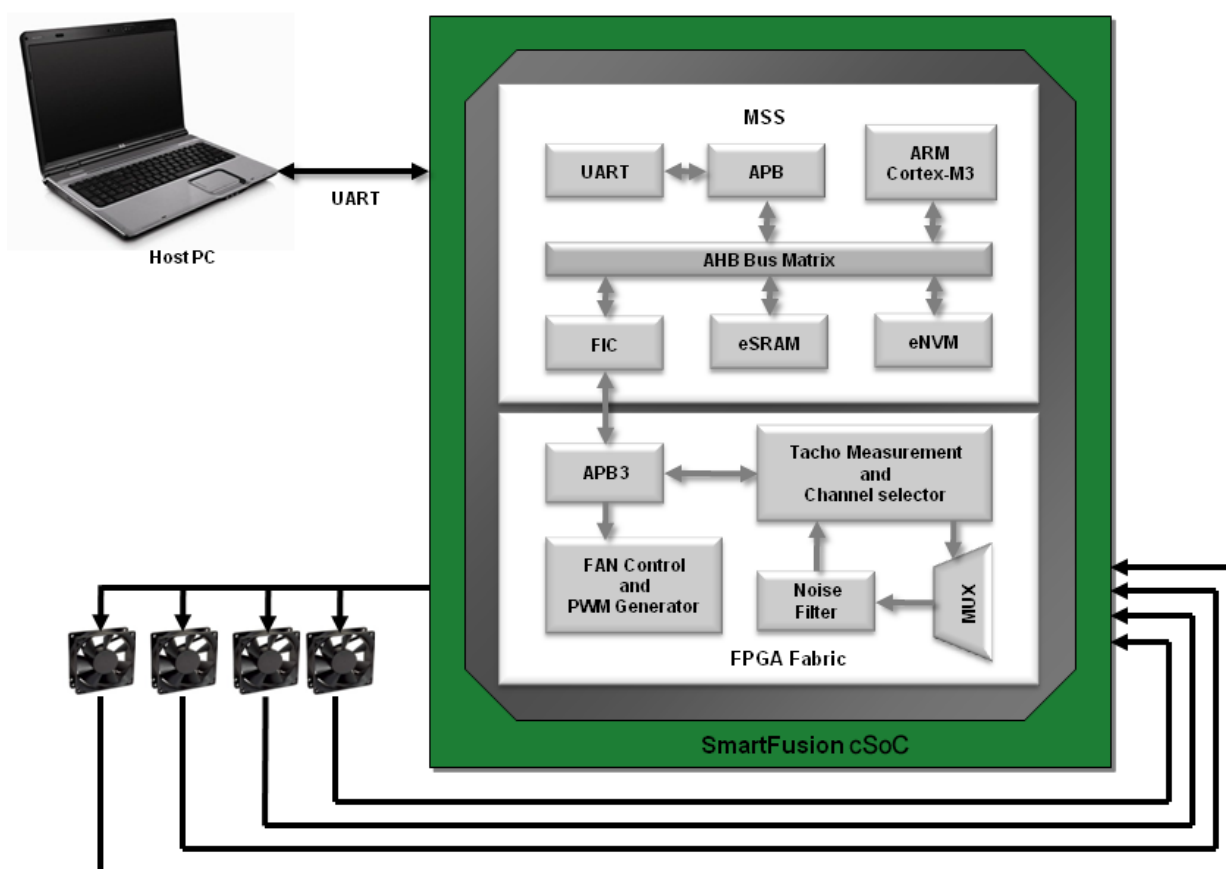
Figure 7 • **Architecture of Fan Controller**

This design is mainly divided into two parts: hardware and software implementation.

Hardware Implementation

The hardware implementation consists of a PWM generator, noise filter, tacho measurement, channel selector, and control logic. These blocks are connected to the MSS through the APB interface. The APB slave interface is implemented in the FPGA fabric and directly connected to the MSS FIC interface. For more details on how to connect FPGA logic MSS, refer to the *Connecting User Logic to the SmartFusion Microcontroller Subsystem* application note.

The tacho signals from the fans are connected to the 4:1 multiplexer and the select lines to the multiplexer are driven from tacho measurement and channel selector. Then the ARM Cortex-M3 processor configures the channel selector to select one of the tacho input channels. In case of the 3-wire fan, the tacho measurement block does the pulse stretching of the PWM.

The output of the multiplexer is fed into the noise filter, which removes the glitches around the rising and falling edge of the tacho signal, where the noise filter acts as digital Schmitt trigger.

The filtered tacho signal is fed to the tacho measurement block. The tacho measurement block counts the number of system clocks between the minimum of 2 rising edges and a maximum of 15 rising edges, which in turn can be translated to the time period of the tacho signal at software level. The configuration of the number of rising edges to determine the period of the tacho helps in reducing the noise.

After counting the system clocks between the configured rising edges, the tacho measurement block gives an interrupt to the Cortex-M3 processor, which in turn determines the current speed of the fan and estimates the new PWM duty cycle to tune to the desired RPM. To support fault generation, you can implement fault generation if the tacho measurement block does not give an interrupt to the Cortex-M3 processor for a specified time.

The PWM generator generates the PWM signal based on the PWM frequency and duty cycle configuration from the ARM Cortex-M3 processor. It supports pulse stretching of PWM, based on the PWM pulse stretch enable signal from the tacho measurement block.

Figure 8 illustrates the Fan Control Hardware Design in SmartDesign. The Fan Control hardware block consists of mss_fan_control block and pwm_n_tach block, which has PWM, tacho measurement, and digital filter sub-blocks.
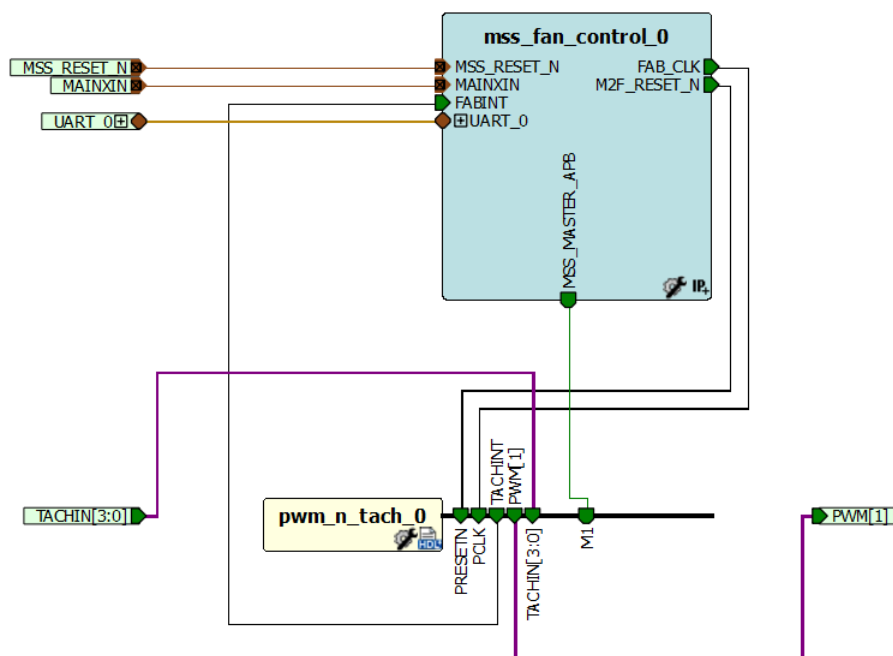

Figure 8 • **Fan Controller in SmartDesign**

Figure 9 illustrates the rising edges of the raw tacho output coming from the fan (wave 2, green color), and filtered tacho signal (wave 1, yellow color) in the FPGA fabric. The raw tacho signal has very poor rise time and it has ripples during the transition from Low to High, causing improper detection of the tacho signal in the FPGA fabric. That is why a digital filter is used. The filtered tacho signal has very good rise time and is ripple free.

Figure 9 • **Rising Edge of Raw Tacho and Filtered Tacho**

Figure 10 illustrates the falling edges of the raw tacho output coming from the fan (wave 2, green color), and filtered tacho signal (wave 1, yellow color) in the FPGA fabric.

Figure 10 • **Falling Edge of Raw Tacho and Filtered Tacho**

Figure 11 illustrates the raw tacho output coming from the fan (wave 2, green color), and filtered tacho signal (wave 1, yellow color) in the FPGA fabric.
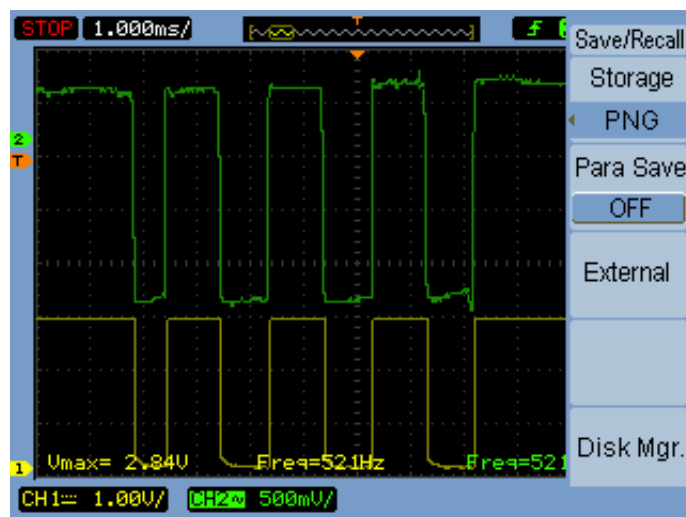

***Figure 11 •* Raw Tacho and Filtered Tacho 1**

Software Implementation

The software implementation contains the configuration and control block. The configuration block communicates with the graphical user interface (GUI) application running on the host PC and gets the configuration data.

Based on the configuration, the controller configures FPGA blocks with the PWM frequency, duty cycle, channel select, and fan type to support pulse stretching and enable or disable.

In case of 2-wire fan, the controller calculates the PWM duty cycle based on the desired speed and minimum PWM duty cycle assuming that the fan speed and PWM duty cycle are linear. You can calculate or implement the PWM duty cycle calculation based on the desired speed, and minimum PWM duty cycle, assuming that the fan speed and PWM duty cycle are exponential or any desired proportion.

In the case of 3- or 4-wire fans, the software receives an interrupt from the tacho measurement block after the measurement is enabled. The software reads the tacho values and translates them into time period and calculates the current speed. Based on the current and desired speed, the software adjusts the PWM duty cycle.

Following are the APIs used in the design:

1. uint8_t fan_menu_n_action(uint8_t menu_print): This function reads the input from the GUI through UART and configures the fan for ON/OFF, closed loop, open loop and updates the corresponding configurations.

2. void init_fan_controller(void): This function initializes the PWM and timer 1, and enables the fabric IRQ.

3. void Fabric_IRQHandler(void): This function reads the tacho measurement values and indicates that which was read from the FPGA tacho measurement block has been successful. It reconfigures timer 1 when it expires and enables the tacho measurement block in the FPGA fabric to get the latest value, so that it tunes the PWM duty cycle.

4. void PWM_init_fan(): This function performs the initialization of the PWM with frequency, duty cycle, and other configurations.

5. void PWM_enable_fan(): This function enables the generation of the PWM signal to the fans.

6. void PWM_disable_fan(): This function disables the generation of the PWM signal to the fans.

7. void fan_closed_loop_control(void): This function translates the tacho values which are read in the Fabric_IRQHandler(void) function to the RPM, compares them against the desired RPM, and adjusts the PWM duty cycle.

8. void fan_open_loop_control(void): This function adjusts the PWM duty cycle based on the desired RPM and minimum duty cycle, assuming that the fan's RPM and PWM duty cycle are linearly related.

9. void Timer1_IRQHandler(void): This function enables the tacho measurement block in the FPGA fabric to get the current value. For a 3-wire fan it enables the pulse stretching.

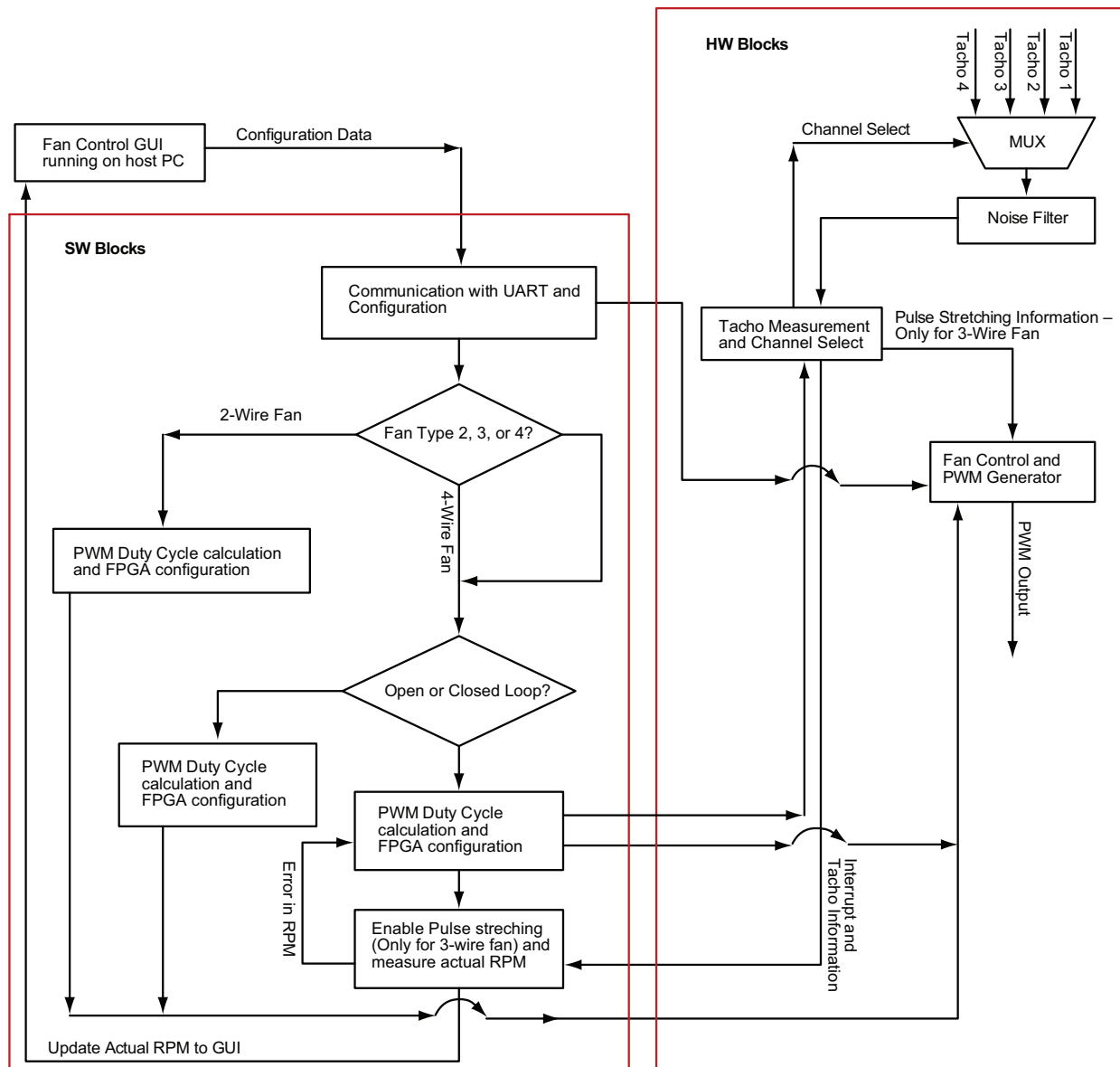Figure 12 illustrates the hardware and software design flow chart.

Figure 12 • Hardware and Software Design Flow Chart

Libero® System-on-Chip (SoC) projects are provided in the design files attached with this design example (refer to "Appendix A – Design Files and Programming Files" section on page 13).

Board Specific Settings

The design example provided with this application note has been tested on the SmartFusion Development Kit Board and the SmartFusion Evaluation Kit Board using external DC fan driver circuit. You need to develop a DC fan driver circuit as shown in Figure 3, Figure 4, Figure 5, or Figure 6 using external components to test this design example. The signals that need to be interfaced with external DC fan driver circuit are mapped to the 5x1 Header (J22) on the SmartFusion Evaluation Kit Board and to Direct C header (J13) on the SmartFusion Development Kit Board.

Table 1 gives the pin mapping between external DC fan driver circuit and the 5x1 Header (J22) on the SmartFusion Evaluation Kit Board.

Table 1 • **Connection Details of External DC Fan Driver Circuit to the SmartFusion Evaluation Kit Board**

S.No	Signal Name	Direction	J22 Header Pin Number
1	PWM	Output	1
2	Tacho 1	Input	2
3	Tacho 2	Input	3
4	Tacho 3	Input	4
5	Tacho 4	Input	5

Table 2 gives the pin mapping between external DC fan driver circuit and Direct C header (J13) on SmartFusion Development Kit Board.

Table 2 • **Connection Details of External DC Fan Driver Circuit to the SmartFusion Development Kit Board**

S.No	Signal Name	Direction	J13 Header Pin Number
1	PWM	Output	1
2	Tacho 1	Input	2
3	Tacho 2	Input	3
4	Tacho 3	Input	4
5	Tacho 4	Input	5

How to Run the Demo

1. Download and install the Microsoft.NET Framework 4 from the Microsoft website.
2. Design fan controller circuit as shown in Figure 3, Figure 4, Figure 5, or Figure 6.
3. Connect the fan controller circuit board to the SmartFusion Evaluation Kit Board or the SmartFusion Development Kit Board, as mentioned in Table 1 or Table 2.
4. Program the SmartFusion Evaluation Kit Board or the SmartFusion Development Kit Board with the generated or provided STAPL file (refer to "Appendix A – Design Files and Programming Files" section on page 13) using FlashPro and then power cycle the board.
5. Invoke the SoftConsole IDE, from Libero SoC project (refer to "Appendix A – Design Files and Programming Files" section on page 13), and launch the debugger.
6. Launch the GUI (DC Fan Control.exe). Set parameters as required. You can leave the default parameters as it is and if you are running the design with one of the above mentioned circuits. Figure 13 shows the DC Fan Control GUI with default settings.
7. Click **Configure**.
8. Click **Run**.

Figure 13 • **DC Fan Control GUI**

Release Mode

The release mode programming file (STAPL) is also provided. Refer to the Readme.txt file included in the programming zip file for more information.

Refer to the *Building Executable Image in Release Mode and Loading into eNVM tutorial* for more information on building an application in release mode.

Conclusion

This application note describes the capability of SmartFusion cSoCs to control different types of DC fans (2-, 3-, and 4-wire DC fans). The FPGA fabric is used to implement tacho measurement and PWM generation. This effectively demonstrates the use of SmartFusion cSoC for software and hardware partitioning to improve overall system performance.

Appendix A – Design Files and Programming Files

You can download the design files from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC377_DF.

The design zip file consists of Libero SoC projects and programming file (*.stp) for A2F500 and A2F200. Refer to the Readme.txt file included in the design file for directory structure and description.

You can download the programming files (*.stp) in release mode from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC377_PF.

The programming zip file consists of STAPL programming file (*.stp) for A2F500-DEV-KIT and A2F-EVAL-KIT, and a Readme.txt file.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 1 (February 2012)	The Figure 8 was updated (SAR 35849).	8
	The "How to Run the Demo" section was modified (SAR 35849).	12
	The "Release Mode" section was added (SAR 35849).	13
	The "Appendix A – Design Files and Programming Files" section was modified (SAR 35849).	13

Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*

51900245-1/02.12