Design Document: Variable Speed Control Fan with Integrated Circuit

Table of Contents

[Design Task Description 1](#_Toc88826282)

[Success Criteria 2](#_Toc88826283)

[User Interaction 2](#_Toc88826284)

[Components available for user interaction 2](#_Toc88826285)

[7 Segment display 2](#_Toc88826286)

[LEDs 2](#_Toc88826287)

[Keys 2](#_Toc88826288)

[Switches 2](#_Toc88826289)

[Design/Flowchart 2](#_Toc88826290)

[Functions/Blocks 3](#_Toc88826291)

[Character2HexConverter 3](#_Toc88826292)

[Digit2HexConverter (Get\_Hex) 4](#_Toc88826293)

[NumberLengthFinder 4](#_Toc88826294)

[Display 4](#_Toc88826295)

[Delay 4](#_Toc88826296)

[LED\_State 5](#_Toc88826297)

[EncoderReader 5](#_Toc88826298)

[ModeSelector 5](#_Toc88826299)

[SetPin 6](#_Toc88826300)

[PinGet 6](#_Toc88826301)

[Main 6](#_Toc88826302)

[Initialisation 7](#_Toc88826303)

[Recursive Loop 7](#_Toc88826304)

[Termination 8](#_Toc88826305)

[Fallback Position 8](#_Toc88826306)

# Design Task Description

The object of this design project is to write a single program to be uploaded onto the DE1-SOC Altera Board connect with a peripheral extension board to allow for variable speed control of a fans speed according to user input. The system is assumed to be used in a standalone product, so I/O via the PC is discouraged.

## Success Criteria

* User input
  + Feedback from system from user input
* Control of fan speed

# User Interaction

## Components available for user interaction

* 7 segment display- will need pointers for the two sections of the display (see Lab activity description for specific pointers)
* LEDs- will need pointer for LED address (see Lab activity description for specific pointers)
* Keys- will need pointers for the keys/pressbutton (see Lab activity description for specific pointers)
* Switches- will need pointers for the switch address (see Lab activity description for specific pointers)

## 7 Segment display

To be used as visual feedback:

* Can show current speed
  + Can show speed with a decimal point or varying degree of significant figures upon user input
* Can show the speed the user has set (Closed loop control only)
* Can display words
  + Open/closed for when setting control type
  + On/Off for start and stop commands

Fallback position: Will solely show speed with no decimal point

## LEDs

These will be used to set a moving bar that displays the duty cycle for the system

## Keys

Used to select modes of what to display

## Switches

Switch 9 is used for turning on and off the fan, switch 8 is used to select open or closed control, and the other 0-7 are used to set the desired speed in binary.

# Design/Flowchart

The Program will work with an initialisation of require variables and pointer and then a recursive while loop. Switch 9 will turn the fan on and off while also starting the control as well. The user has two methods of control dictated by the state of Switch 8 with high being closed loop control and low being open loop. Open loop control takes the encoder and has a value between 0 and 100 that you can increase by spinning clockwise and decrease by spinning counterclockwise this will change the duty cycle displayed on the LEDs and change the PWM control to the Fan. Closed loop control takes the value of switches 0-7 and converts the values into a single 8-digit binary number to set as the desired speed. Then, PID control is used to adjust the duty of the PWM to get the fans actual speed to equal the desired speed. What information gets displayed is dictated by Keys 0-2 with the options to display actual speed of the fan, desired speed of the fan, or absolute speed error.

Diagram

Description automatically generated

# Functions/Blocks

## Character2HexConverter

This function converts C to hex to be used by the display to show when in closed loop control mode

Input Parameter:

* String (str)- string for conversion

Return:

* Char2Hex (int)- hex for that string

Other Variables:

* N/A

## Digit2HexConverter (Get\_Hex)

This function defines and sets digits to their corresponding hexadecimal code for the seven-segment display. A suggested method is to use cases to assign hex code to each number. Fall back method is using a mathematical function.

Input Parameter:

* DecimalNum (int)- number for conversion

Return:

* Dec2HexVal (int)- hex for that integer

Other Variables:

* N/A

## NumberLengthFinder

This function finds the length of the number that will outputted on the seven-segment display in the display function later. A suggested method is taking the number and using a division operation using 10, as the integer would be in base ten, then set the new number to the original variable and do this until the number is not greater than 0. With the count being the size of the decimal number. Fallback position is if statements for all the possible lengths (much less efficient method).

Input Parameter:

* DecimalNum (int)

Return:

* Count (int)

Other Variables:

* N/A

## Display

This function is for taking the number that should be displayed and assigning it to the correct section of the display. A suggested method is

Input Parameters:

* NumberToDisplay (int)

Return:

* N/A

Other Variables:

### Delay

This function is a set format for implementing delays within the main loop and or other functions. A suggested method is time by number of clock cycles that are required to elapse to get the specified delay. This is done by converting time required for the delay into a number of clock cycles and looping until the number of clock cycles that pass is equal to the converted number of cycles for the required time.

Input Parameters: DelayTime (int)

Return:

* N/A

Other Variables:

* SystemCounter (int)- the counter found on the hardware
* Previous\_time (int)- time when starting the delay

### LED\_State

This function would be used to change the state of each LED

A suggested method is to use the LEDs to display the duty cycle of the PWM signal by taking the value of the duty cycle (0-100) and update the LEDs from the left to right using a /10 operation. This results in an increment of 10% per LED.

Input Parameters: DutyCycle (int)

Return:

* N/A

Other Variables:

* LED\_Position (int)
* LEDs (volatile int): a pointer variable for the LEDs

### EncoderReader

This function will create a count for the encoder so when it is rotated to clockwise the counter increases and decrease when rotated counterclockwise. A suggested method for this is comparing the A and B signals from the encoder to determine direction and then increase/decrease the counter that can be used to set the duty cycle later.

Input Parameters:

* EnA(int)- bit containing state of signal A from encoder
* EnB(int)- bit containing state of signal A from encoder
* PrevEn(int)- previous encoder count value bounded between 0-100 to help set PWM duty cycle later

Return:

* PrevEn(int)

Other variables:

* IsRotating (int)- if equal to 1 then the encoder is rotating. If equal to 0 then encoder is stationary.

## ModeSelector

This function is used to allow the keys to be used to by the user to select the information they want displayed from, Actual Speed in RPM, Desired Speed in RPM, and absolute Speed error. A suggested method is to use cases to set the press of a button to the array and set that to the variable “KeyState”. It also takes the Switches states to find the desired speed inputted by the user in binary.

Input Parameters:

* KeyStates (int)- variable with all the states of the keys to know if they have been pressed or not
* SwitchStates (int)- variable with all the states of the Switches to know the desired RPM inputted by the user
* ActualSpeed (int)- speed of the fan
* DesiredSpeed (int)- target speed for the fan set by user
* Error (int)- the absolute RPM error = abs(ActualSpeed - DesiredSpeed)
* Duty (int)- duty cyle variable

Return:

* KeyStates (int)

Other Variables:

* N/A

## SetPin

This function is used to set a bit of GPIO to either high or low. A Suggested method to use two if statements one for a high case and another for a low case.

Input Parameter:

* BitLocation (int)- for the GPIO port. It is the desired pin number that needs be set
* IsHigh (int)- if one the bit is set to high if 0 it is set low

Return:

* N/A

Other Variables:

* N/A

## PinGet

This function reads a specific pin on the GPIO port and returns its value. A suggested method for this is just a single return block

Input Parameter:

* PinLocation (int)- which pin number is desired

Return:

* (\*GPIOPort & (1 << PinLocation)) >> PinLocation;
  + this should return the value of the bit at the desired location.

Other Variables:

* \*GPIOPort- this is the pointer for the GPIO port

## Main

Within this function the functions above are used within. This function will have a while (1){} that will act as a recursive loop. This function contains script for:

* Initialisation/Setup
* Encoder
* PWM
* Tachometer

In this recursive loop Switch 9 will be set as the on/off switch for the fan using an if statement controlled by the switch as an initialise program which sets the counters for the tachometer, PWM, and the “global” Counter (on board).

### Initialisation

This process would start with initialising the FPGA using EE30186\_start(). Then, set the pointers to the specific pins for the outputs of the components that are not a part of the board and set the variables.

* GPIO
  + Bit 1 is the speed indicating pin which will be used as a tachometer
  + Bit 3 is the fan PWM driver
  + Bit 17 is the A signal for the encoder
  + Bit 19 is the B signal for the encoder
* Encoder
  + Pin location for signal A is 17 (int)
  + Pin location for signal B is 19 (int)
* Fan
  + PWM pin for fan is 3 (int)
* Tachometer
  + Pin location for fan tachometer signal is 1
  + TachometerCounter[] (double): this variable is an array to hold the clock counts
  + TachometerClock (double): this is the clock counter
  + TachometerState (int): holds the state of the tachometer
  + TachometerFrequency[] (int): holds the calculated frequency values
  + AverageFrequency (int): variable for calculating an average frequency from the array of values TachometerFrequency[]
  + ActualSpeed (int) calculated from the average frequency and the time elapsed
* PWM
  + Counter\_Freq (int)- this is a variable used for the frequency of the FPGA which is 50000000 (50MHz)
  + Fsw (int)- this is the fsw of the PWM
  + ActiveDuty (float)- bounded between 0 and 100
  + DutyTime (int)- this is a variable to keep the duty time
  + PWM\_Prev\_time (int)
* PID
  + Init\_pid (int): variable for when closed loop is selected to start the PID block. 1 for closed loop control and 0 for open loop control
  + PrevCount (int): a counter for the PID process
* Counter
  + Will need a pointer for 32bit counter operating at a 50MHz frequency

### Recursive Loop

Next there will be a recursive using while (1){} with an if loop so that when SW9 is turned on so is the system. This will need a variable “initialised” (int) to keep track of this with 1 being on and 0 being off. The working processes of the system when on will go within the if statement for the on condition

#### Tachometer Reader

This will use the PinGet Function to read the value of the tachometer pin when the state is read high the function will start the counter to find the time between positive edges. It will use the array TachometerCounters and the variable TachometerClock to find the frequency.

#### Encoder reader

When trying to reading the encoder, the PrevEn will be subtracted from the counter to set a number of loop cycles so as not to constantly be taking readings. A suggested number of loop cycles would be 4000 but should be refined with testing.

#### Mode selector

This section would just call the ModeSelector function and run the function.

#### PID

This process uses ModeSelector to take the defined parameters and:

* set the DesiredSpeed by adding a minimum speed and adjusted PrevEn value. The adjustment factor would be decided in testing (recommend start point of multiply by 10).
* Create a variable DT(int), found by subtracting PrevCount from Counter
* Defining K values as 3 variables
  + Kp, Ki, Kd (all float) these are constants found through testing
* During the closed mode from ModeSelector, Duty +=Kp\*error +Ki\*error\*DT+Kd\*error/DT;
* During open mode Duty=PrevEn and will display Duty on the display

#### PWM

This process will set and use the PWM signal.

* DutyOn (int)- = (Duty/100)\*fsw
* LED state will be set with Duty to show the duty cycle in the LEDs constantly

## Termination

This will be done by using EE30186\_end() and returning 0 to shut down the system exit the main loop

# Fallback Position

A fallback position for this project is to go to its simplest form by having solely open loop control which involves the user turning the encoder to change the duty counter and thus change the fans speed then use display to only show the actual speed. The system will start the fan on power to the board. This ultimately accomplishes the two success criteria of having user input and controlling the speed of the fan.