Application Note

Z8 Encore! XP® F1680 Microstepping Controller

AN027201-0408

Z8 Encore! XP® Flash Microcontrollers

Abstract

Zilog's Z8 Encore! XP® F1680 Series incorporates a best-in-class feature set that is optimized for *stepper motor microstepping* controls. The features of Z8 Encore! XP F1680 include:

- A fast 11 MHz internal oscillator
- Two analog comparators
- 10-bit Analog-to-Digital Converter (ADC)
- Multichannel PWM timer
- Three general-purpose timers

This Application Note describes how to drive a unipolar stepper motor using Z8 Encore! XP F1680 onboard analog comparators for one-shot feedback current limiting, and the multichannel timer as a microstepper sine/cosine current generator.

Note: *The source code file associated with this application note, AN0272-SC01 is available for download on* www.zilog.com.

Developing the Application with Z8 Encore! XP F1680

Theory of Operation

Microstepping, or sine/cosine microstepping, is a stepper motor drive technique in which the current in the motor windings is controlled to approximate a sinusoidal waveform. Microstepping produces a much smoother rotation than full step drive, provides greater resolution and freedom from resonance problems, as there are more steps per revolution.

In conventional full step drive, an equal amount of current is applied to each of the motor's stator coils. The magnetic rotor aligns itself in the coil's magnetic field. With each motor step current is reversed in one of the coils and the rotor realigns to the new magnetic field. This moves the rotor one motor step or 90 degrees.

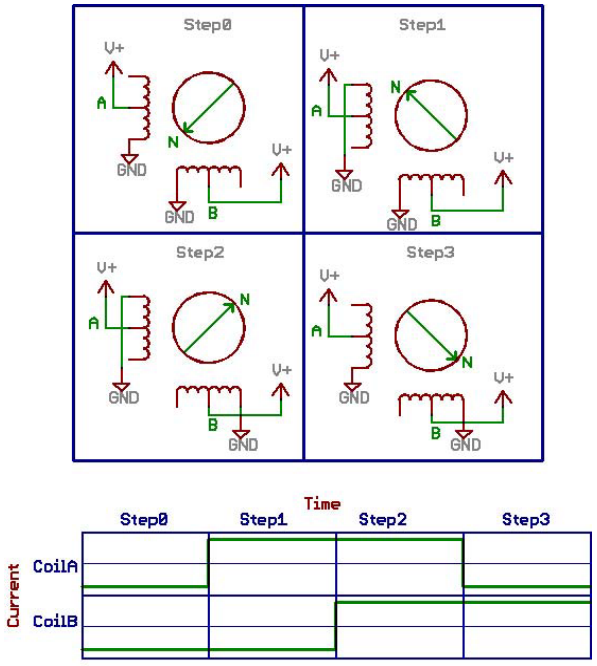

Figure 1. Microstepping

In microstepping, varying amounts of current is applied to the motor's coils so that the magnetic field smoothly transitions from one polarity to the next. Each full step is now divided into several microsteps of varying current, which produces a larger number of magnetic fields that the rotor can align with. The result is smoother motor rotation, quieter operation, and greater motor resolution.

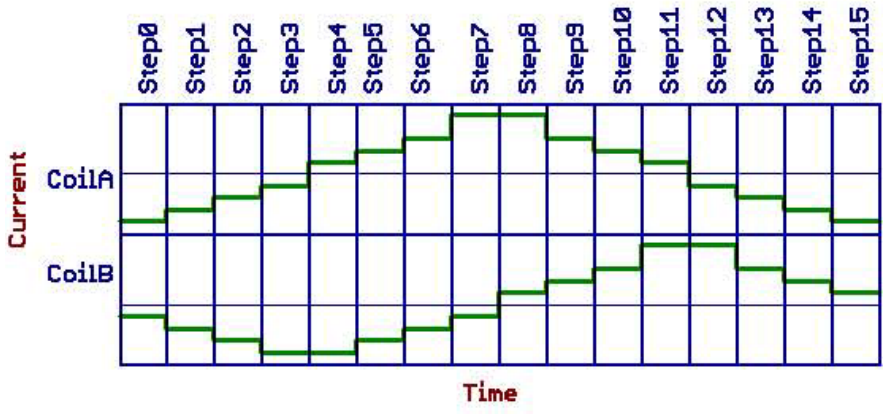

Figure 2. Microsteps of Varying Current

Discussion

Hardware Architecture

The motor requires two current generators, one for each coil. For demonstration purposes the application includes the following:

- A potentiometer for adjusting motor speed
- A switch to turn the motor ON and OFF
- A switch to reverse the direction
- A switch to advance the motor one step

User Interface

Switches S1, S2, and S3 are pulled down when pressed and are pulled up by the F1680's internal pull-ups. Speed potentiometer R16 is read by the F1680's internal Successive Approximation Register (SAR) ADC, using V_{DD} as a reference.

Current Generator

The motor's two current generators use several features of Z8 Encore! XP F1680. Figure 3 displays one of the two current generators.

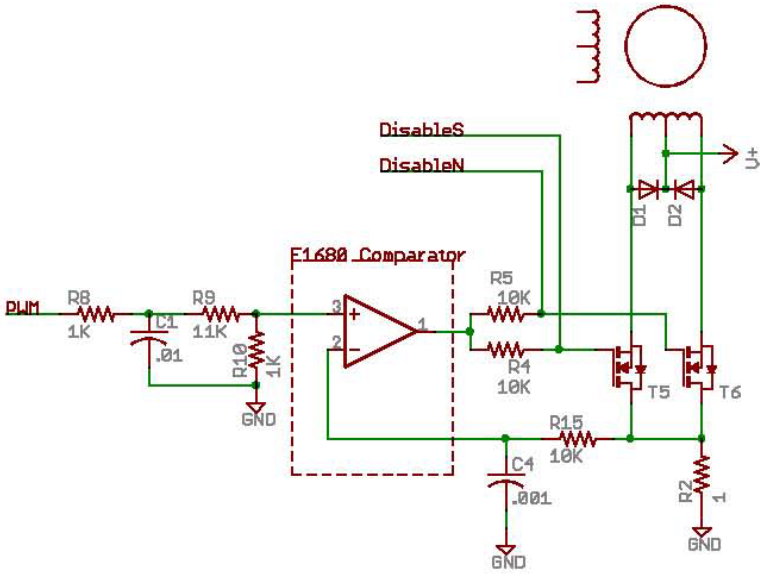

Figure 3. Current Generator

Z8 Encore! XP® F1680 provides a PWM signal, which is averaged by R8 and C1. The duty cycle of PWM represents the desired current to be produced by the generator. R9 and R10 attenuate the averaged PWM signal so that the PWM signal is scaled properly for the non-inverting input of the F1680's comparator. R2 is a current sense resistor for the motor's winding while R15 and C4 provide a small amount of delay for the comparator. When T5 or T6 is turned ON, current starts building in the motor winding and the voltage drop across R2 also builds till F1680's comparator trips at the desired motor current. In some applications an additional diode may be required to protect the body diode of each of the field effect transistors (FETs) from excessive dissipation.

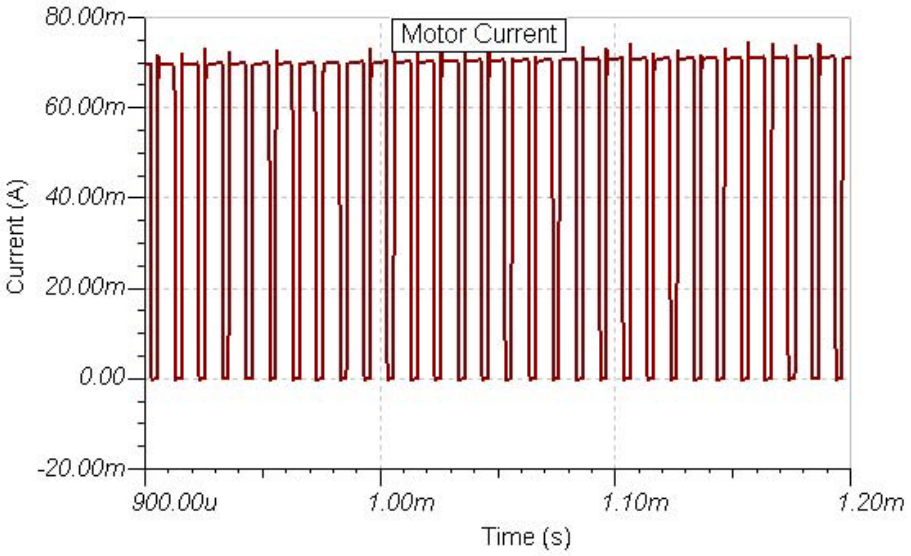

Figure 4. Simulation

The **DisableS** and **DisableN** signals determine the coil's polarity. These I/O pins are configured as open-drain. Thus when the port pin is High, appropriate transistor is turned ON with the PWM signal from the comparator whereas, when the port pin is pulled Low, it turns OFF the coil by pulling the transistor's gate Low.

Hardware Details

Multi-Channel Timer

Z8 Encore! XP F1680's Multi-Channel Timer provides two PWMs required for the current generator's reference. The Multi-Channel Timer has a 16-bit up/down counter with a prescaler and four independent capture/compare channels that can be used for ONE-SHOT, CONTINUOUS, PWM, or CAPTURE modes. In PWM Output operation, the channel generates a PWM output signal on the channel output pin (TOutA, B, C, or D). The channel output toggles whenever the timer count matches the channel compare value in the match registers (MCTCHyH and MCTCHyL) and when the count terminates.

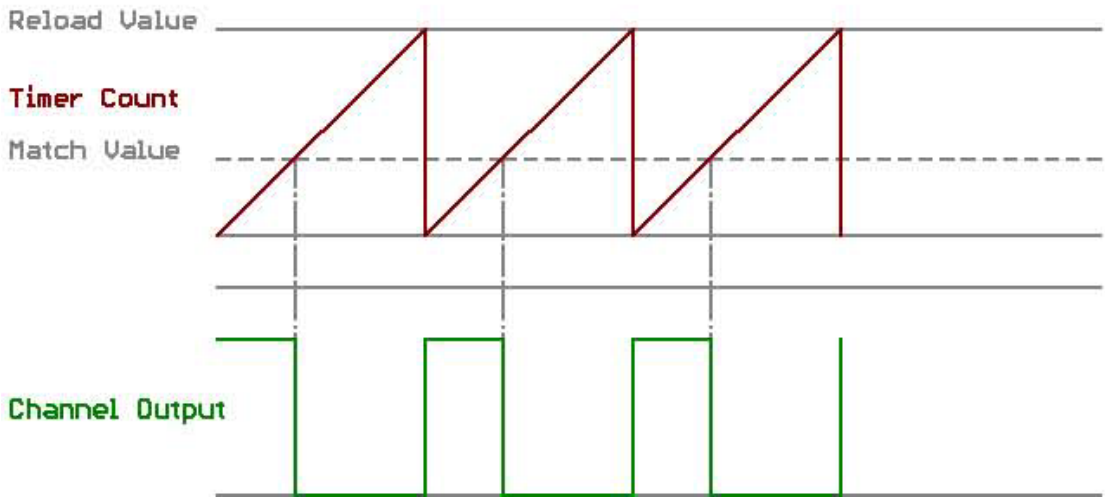

Figure 5. Multi-Channel Timer

The MCTCHyH and MCTCHyL match registers are buffered and are written to at any time without corrupting the PWM. Any changes to the registers are delayed until the next timer end count.

Comparator

Z8 Encore! XP F1680 features two onboard comparators with programmable internal references. In this application the comparators are used as freestanding comparators that free the design from an additional external hardware.

Open-Drain Outputs

An additional feature of the Z8 Encore! Series microcontrollers is the ability for the ports to be configured as open-drain. This means that the output port pin will not source any current when it is High but will sink current when it is Low. The ability to use port pins as open-drain helps to solve awkward design problems that would otherwise require additional external hardware.

Software Implementation

IdleTimers()

`IdleTimers()` function configures the following three timers that are required to run the microstepper application:

Timer0—Timer0 runs in the CONTINUOUS mode and is used for Main loop timing with a frequency of 60 Hz.

Timer1—Timer1 executes in the CONTINUOUS mode and its frequency is varied to use it for the motor's speed control. This timer triggers the interrupt service routine (ISR) that changes current applied to the motor’s windings.

Multichannel Timer—The multichannel timer is used to generate the two PWM signals used for the motor's current regulators.

IdlePorts()

`IdlePorts()` function configures the ports’ direction, pull-ups, and any alternate functions that the ports require.

RefreshPorts()

Similar to `IdlePorts()`, `RefreshPorts()` function also configures the ports' direction, pull-ups and any alternate functions. In noisy environments port directions can be corrupted, therefore it is preferred to refresh the port direction register regularly.

MotorISR()

The actual motor driving is performed within a single interrupt function, which services the motor's current generators, and coil enables. `MotorISR()` is the Timer1 ISR and is serviced when Timer1 expires, or when requested manually by pressing the Step Key.

`MotorISR()` function first reloads Timer1's reload register with the current speed. Since this interrupt is serviced when Timer1 expires, Timer1 is close to zero and is safely reloaded with a new value. If the Timer is reloaded with a value lower than its current value, then the timer continues to count up until it overflowed past `0xFFFF` and then back to the value loaded in the reload register. This overflow would momentarily stall the motor so that the timer must be loaded with its new value at the start of the ISR.

Next, the function uses the variable `motorFwd` to determine the direction in which the motor is turned, that is, forward or backward. The forward direction increments the pointers used for driving the motor steps and a reverse direction decrements the pointers. There are three pointers, two for the current generators and one for the coil enables. `CoilPtr` points to the ROM table Coils[] which defines the coil enables for each full step. `CoilPtr` is changed whenever the polarity of a coil changes. PhaseAPtr and PhaseBPtr point at ROM tables PhaseA[] and PhaseB[] that hold timer values corresponding to each microstep. The PhaseA[] and PhaseB[] table values are selected to produce a sine and cosine current function for the motor's coils and are loaded into the Multi-Channel Timer Match registers to generate a PWM signal that is averaged and used as the comparator's reference. Including interrupt latency, the entire ISR requires minimum 23.5 µs to execute, allowing for fine microsteps and a high Pulse Per Minute speed.

Main Loop

The **Main** loop cycles at 60 Hz and performs all the tasks required for the application. First, all the port modes are refreshed to insure that they are always in the correct direction, pull-ups are enabled, alternate functions are selected, open-drain pins are correctly configured and interrupts are enabled. Next, the functions for reading the keyboard and pot are called and then the User Interface function is called. The Main loop then waits for the 60 Hz timer to expire before repeating.

DebounceKeyboard()

`DebounceKeyboard()` function is called once during every Main loop to read and debounce the keyboard. This function is highly flexible and can be used in a wide variety of applications. The function first calls `ReadKeyboard()` function to return the status of any key that is pressed. Before considering any keypress valid, the keyboard must be enabled by debouncing the condition of all open keys. After the keyboard is enabled the same key must be closed for a number of Main loop cycles before it is considered a valid keypress. Once the keypress is considered valid the keyboard is disabled to prevent the system from seeing repeat keypresses. The keyboard is not enabled again until all keys have been debounced in the open condition again. Certain keys are allowed to repeat if they are held closed and in this application the Step key repeats if it is held down. The speed of the repeat, or slewing, has two rates: when the key is first closed it repeats at a slow rate for a number of cycles and then speeds up and repeats at a much higher rate.

UserInterface()

`UserInterface()` function is called once per Main loop. This function acts upon the keys that are pressed and determines if the motor should turn ON, turn OFF, change speed, change direction or step.

ReadKeyboard()

`ReadKeyboard()` function is called once per Main loop by `DebounceKeyboard()` function. This function performs the keyboard read at the I/O level.

ReadPot()

`ReadPot()` function is called once per Main loop by the `DebounceKeyboard()` function. This function uses the ADC to read the position of the speed potentiometer.

Wait()

`Wait()` function polls the Timer0 interrupt Flag and pauses execution until the timer trips the Flag. The interrupt Flag is reset again before exiting.

Testing

A circuit board is built as per the schematic in the reference section, and a Nippon PF35T-48L4 motor is used as the load.

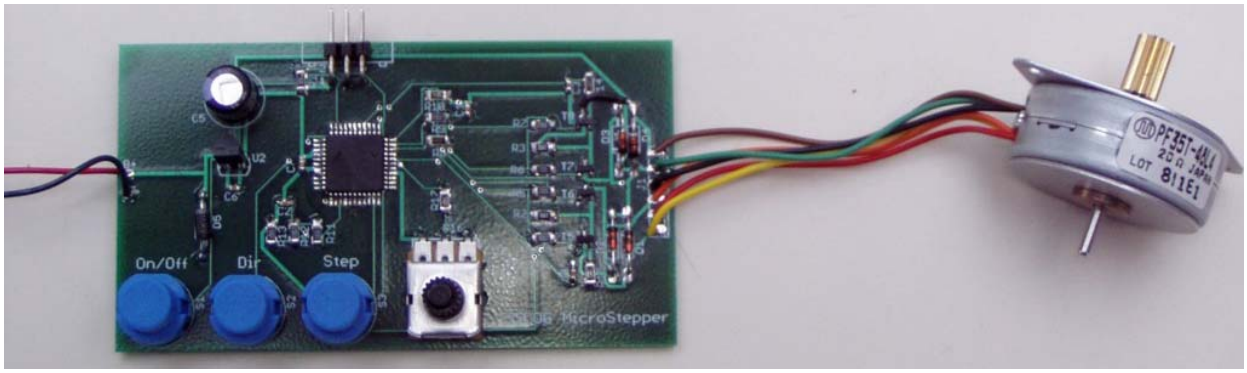

Figure 6. Printed Circuit Board Connected to Motor

The motor is tested from a minimum speed of 19 pulses per second to 190 pulses per second in both directions, full speed and single stepped. The coil current is measured for proper operation by reading the voltage drop across R2 and R3 with an oscilloscope.

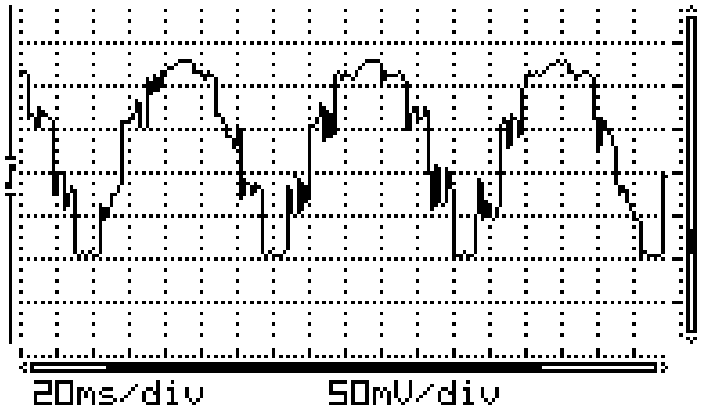

Figure 7. Coil Current

The execution time of the Main loop is measured at 47 µs and the execution time of the ISR is measured at 21 µs. The maximum software microstep speed in this application is calculated using the following equation:

1sec-(47 µs*60)/21 µs= 47485 microsteps/second

Modifications

Different Motors

Different motor may require different drive voltage and current. Changing the value of the sense resistors, R2 and R3, changes the motor current. Larger motors may also require changing T5-T8 or adding diodes to protect the body diodes of these transistors.

Additional Microsteps

Additional microsteps can be added by changing the definition `MicroSteps` in `StepperIO.h` file and by adding additional entries to the tables PhaseA[] and PhaseB[].

Compensated Sine/cosine Profiles

As written, the software generates ideal sine/cosine current profiles for the motor. However, an ideal current profile does not guarantee the best step accuracy. Because of varying air gap area, air gap distance, and magnetic hysteresis, the flux vector direction and magnitude deviate from the ideal sine/cosine behavior in the motor. The accuracy can be improved for a particular motor by altering the entries in tables PhaseA[] and PhaseB[] and tailoring the sine/cosine profile for the motor.

Summary

Z8 Encore! XP® F1680 Series has an ideal set of peripherals for low-cost microstepping motor control. The onboard comparators can be configured for one-shot feedback current limiting, while the multichannel timer provides a PWM reference for a sine/cosine current generator. The compact solution requires only 12 I/O pins, and a single interrupt for all microstepping functions.

References

The documents associated with Z8 Encore! XP® F1680 available at www.zilog.com are provided below:

- Z8 Encore! XP® F1680 Series Product Specification (PS0250)
- eZ8 CPU User Manual (UM0128)
- ZDS II—Z8 Encore!® User Manual (UM0130)
- Control of Stepper Motors (http://www.cs.uiowa.edu/~jones/step/)

Appendix A—Schematics

Appendix A displays the schematic for Z8 Encore! XP® F1680 Microstepping Controller (see Figure 8).

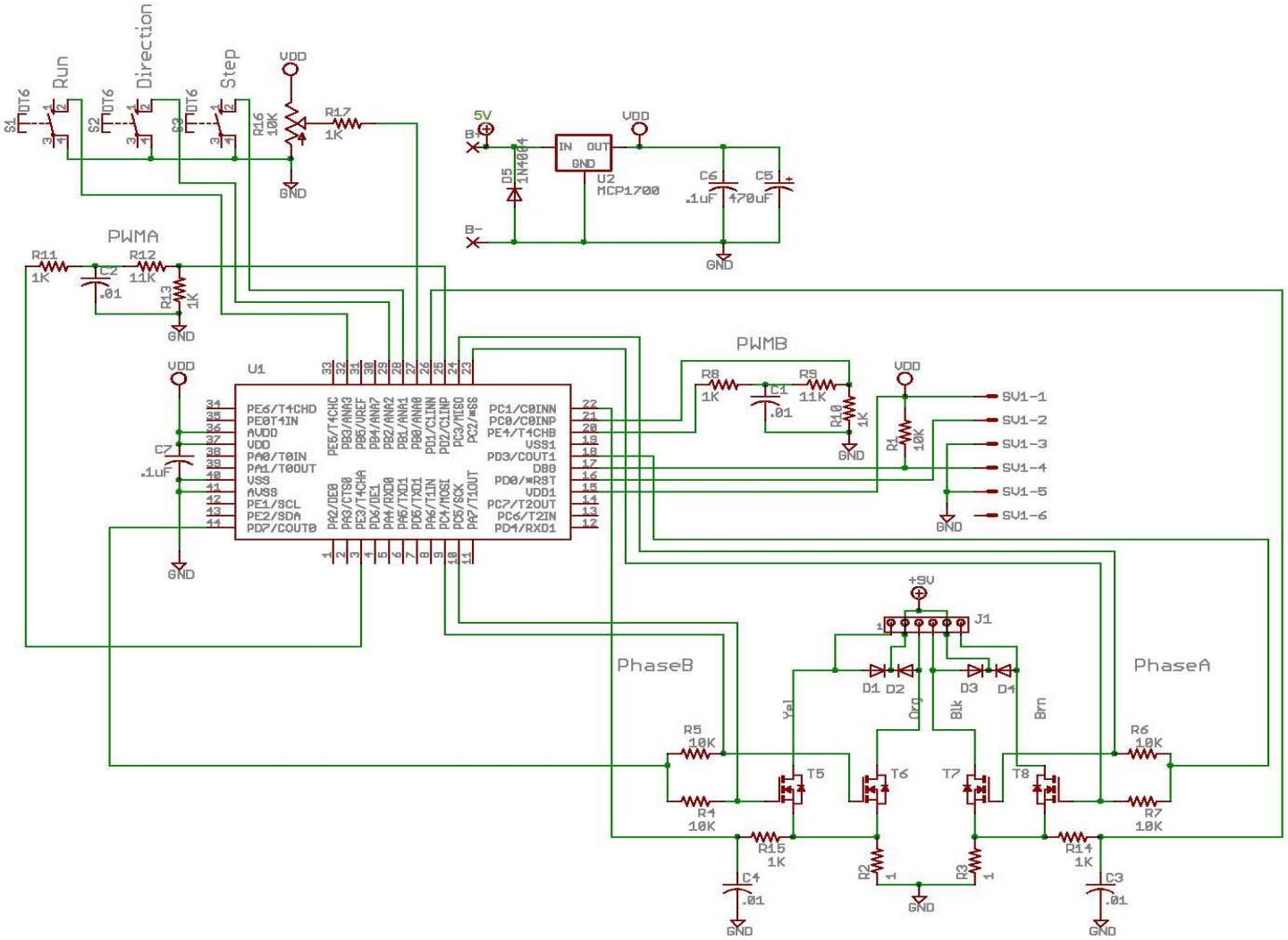

Figure 8. Schematic for Z8 Encore! XP® F1680 Microstepping Controller

Appendix B—Flowcharts

This Appendix displays the flowcharts of the API functions (Figure 9 through Figure 15).

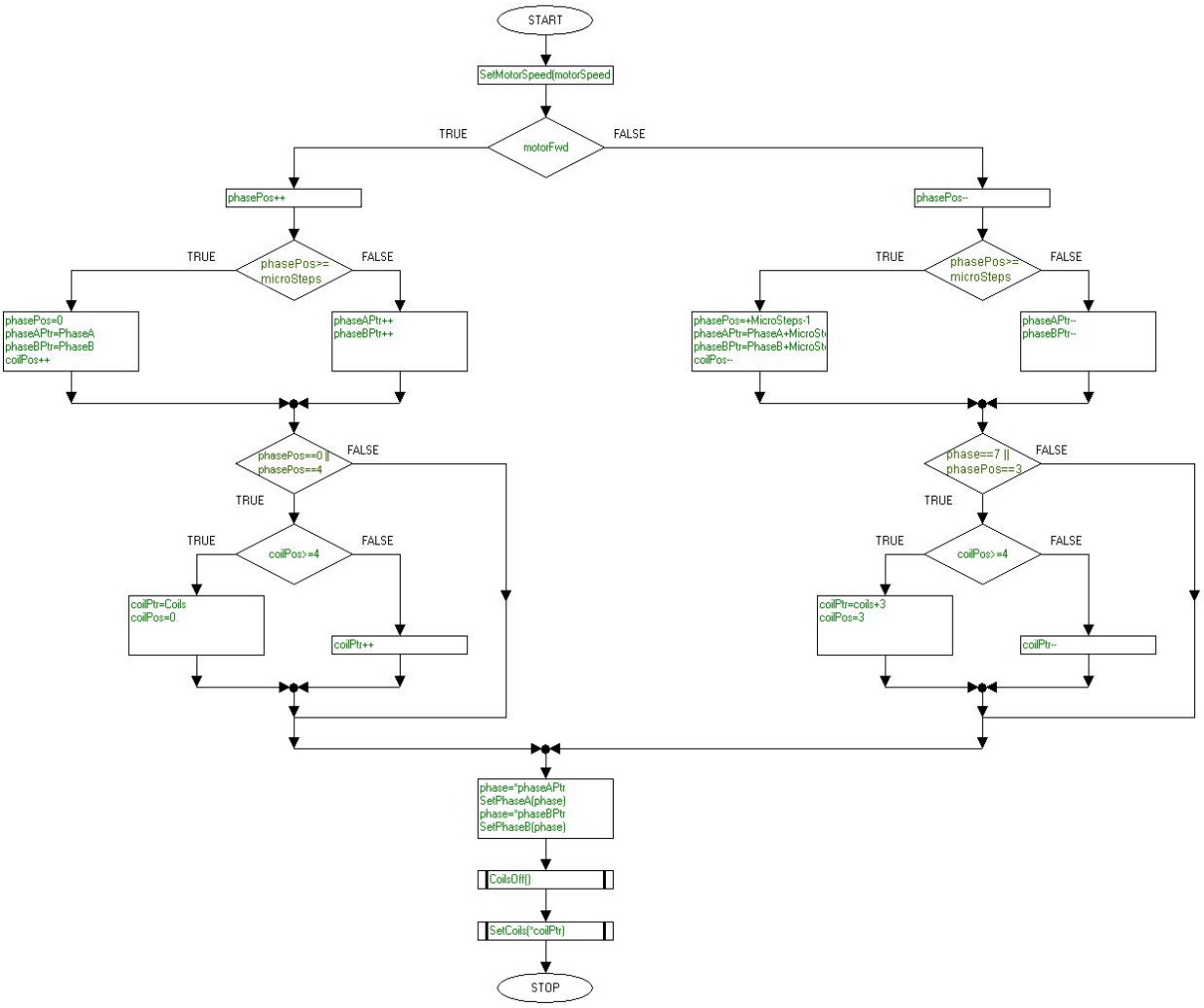

Figure 9. `MotorISR()` Function

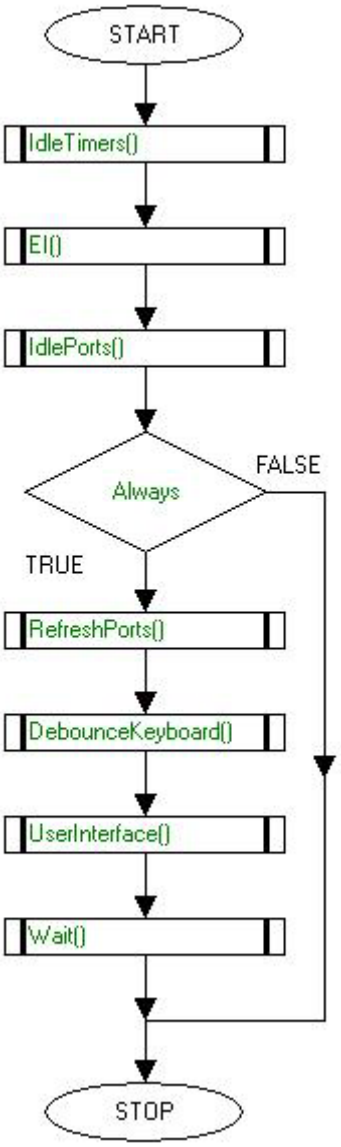

Figure 10. `Main()` Function

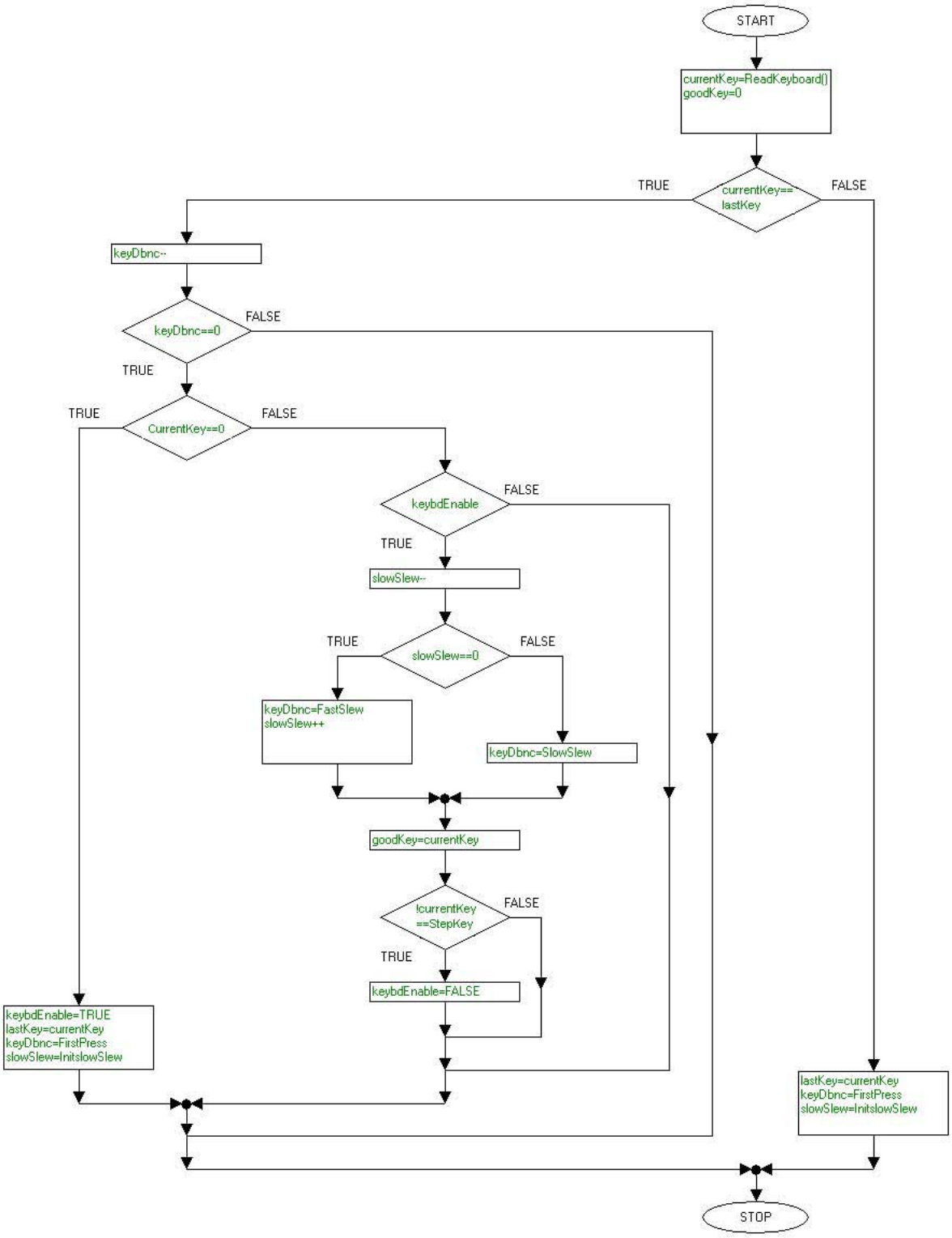

Figure 11. `DebounceKeyboard()` Function

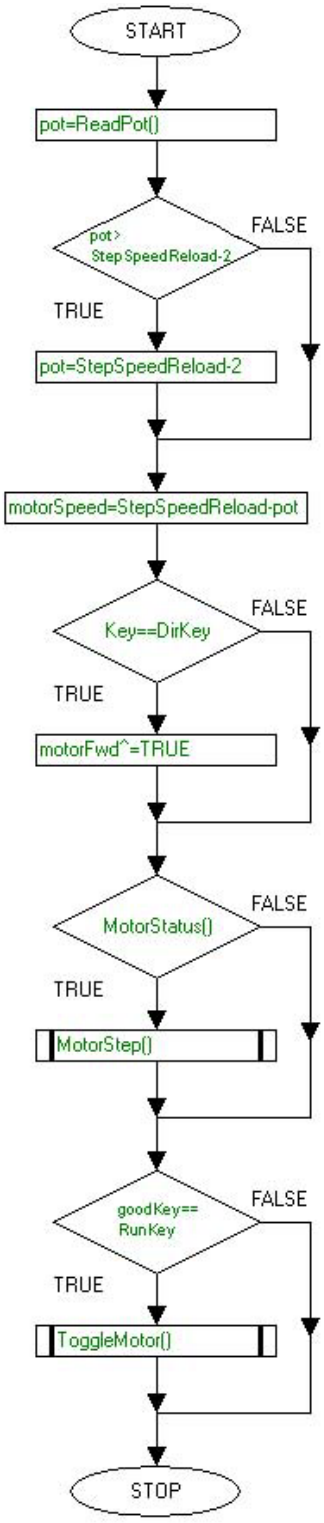

Figure 12. `UserInterface()` Function

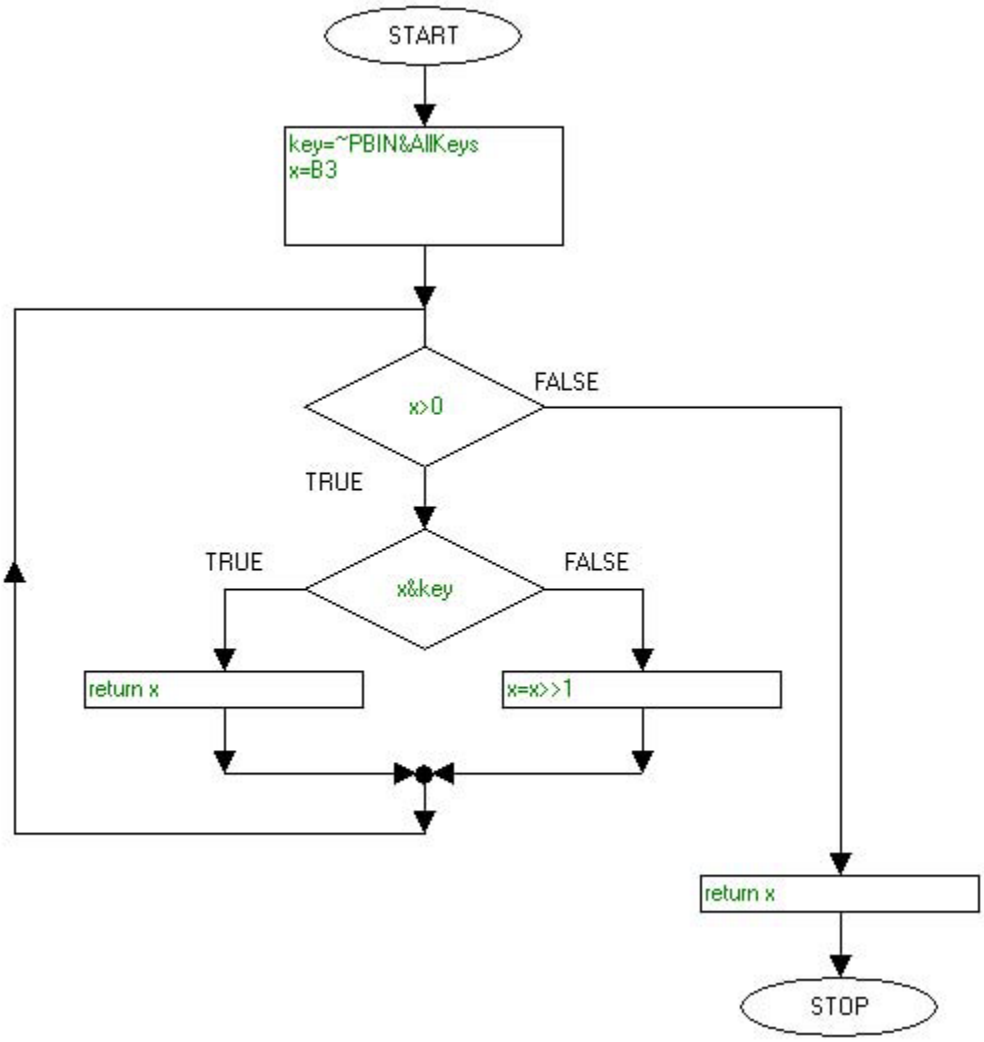

Figure 13. `ReadKeyboard()` Function

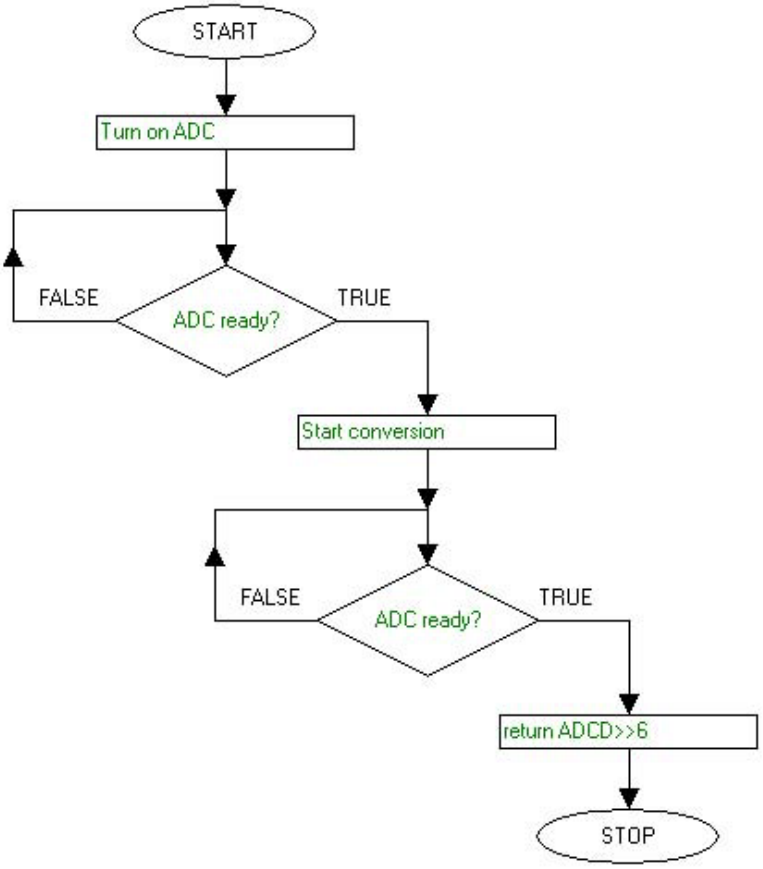

Figure 14. `ReadPot()` Function

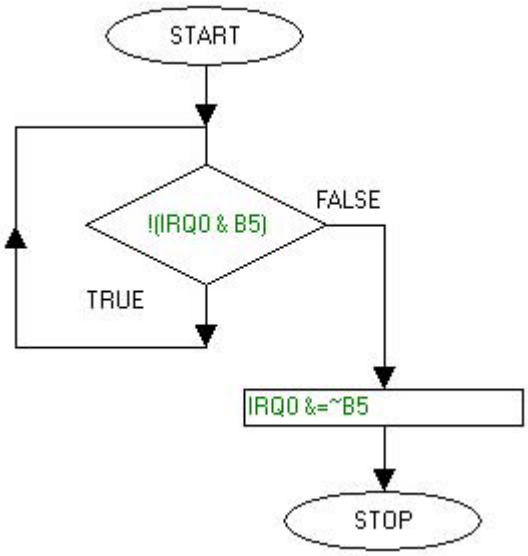

Figure 15. `Wait()` Function

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.