



Application Note

*A Digital Shift Indicator for
Motorsports Competition*

AN005901-Z8X0400



This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters

910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

© 2000 by ZiLOG, Inc. All rights reserved. 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concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded.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. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.



Table of Contents

General Overview	1
Discussion	1
Functional Description	1
Automotive Ignition Systems	4
Firmware	10
Summary	14
Technical Support	15
Assembling the Application Code	15
Application Source Code	15
Test Procedure	34
Equipment Used	34
General Test Setup	35
Test Procedure	35
References	37
Appendix	38

Acknowledgements

Project Lead Engineer

John D. Conder

Application and Support Engineer

John D. Conder

System and Code Development

John D. Conder



A Digital Shift Indicator for Motorsports Competition

General Overview

A shift indicator is an essential part of competitive motorsports. The indicator allows the driver to concentrate on navigating the course without having to check the tachometer for transmission shift points. The indicator is essentially a light placed in the driver's normal field of vision. When the engine reaches a predetermined number of revolutions per minute (rpm) the light comes on, informing the driver that it is time to shift to the next higher gear.

Traditional shift indicator designs are primarily analog frequency to voltage converter circuits. This traditional design converts the current pulses from the engine's ignition system into a DC voltage that is compared to a reference voltage, which represents the shift point. The major drawback of this type of design is that a compromise must be made between the amount of ripple in the DC output at low rpm and the circuit's response time. In addition, because it depends on passive components, this circuit is relatively inaccurate and requires provisions for tuning.

This Application Note presents a simple digital solution using the Z8PE003 microcontroller, which overcomes the drawbacks of the analog design. Also, the digital approach allows features to be added at no cost, which, if implemented in analog circuitry, would require many additional components.

Discussion

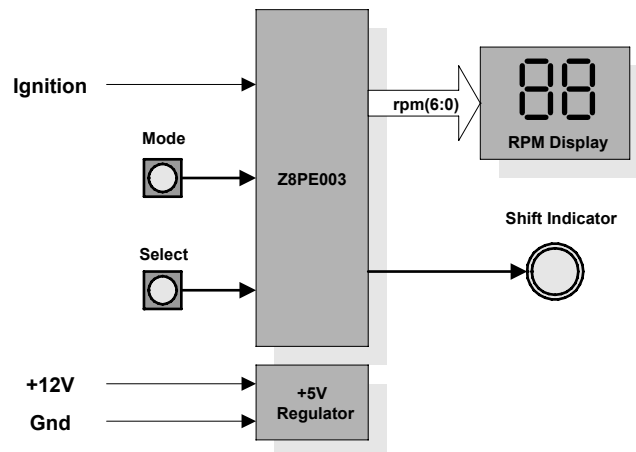
Functional Description

To reduce complexity, the shift indicator application presented here is designed to reflect a typical automotive application. Therefore it is limited to conventional ignition systems (12V, single spark, single coil) of 4-, 6-, or 8- cylinder engines that are 4-stroke (2 revolutions per power cycle). These engines typically operate from a few hundred rpm at idle to about 8000 rpm at maximum speed. These engines' power curves peak near the upper end of the rpm extremes. Shifting gears is usually desirable near this power peak. Based on these general characteristics, this application is designed around the following operational parameters:

- Operating range: 500 to 9000 rpm
- Shift indication range: 1000 to 7900 rpm
- Shift point resolution: 100 rpm

To facilitate programming the shift point, the application provides a 2-digit numeric display and two push-buttons (Mode and Select) as shown in the functional block diagram in Figure 1. Upon power-up, the display and buttons are used to set the engine configuration and the shift rpm. After these values are entered, the display defaults to presenting the current engine speed in units of 100 rpm. If the engine speed falls outside the operational limits (500 to 9000), RPM Display is set to “- -” and the Shift Indicator is turned off.

Figure 1. Functional Block Diagram



Reconfigure the shift rpm at any time by pressing the Mode button to initiate the configuration sequence. This sequence is as follows:

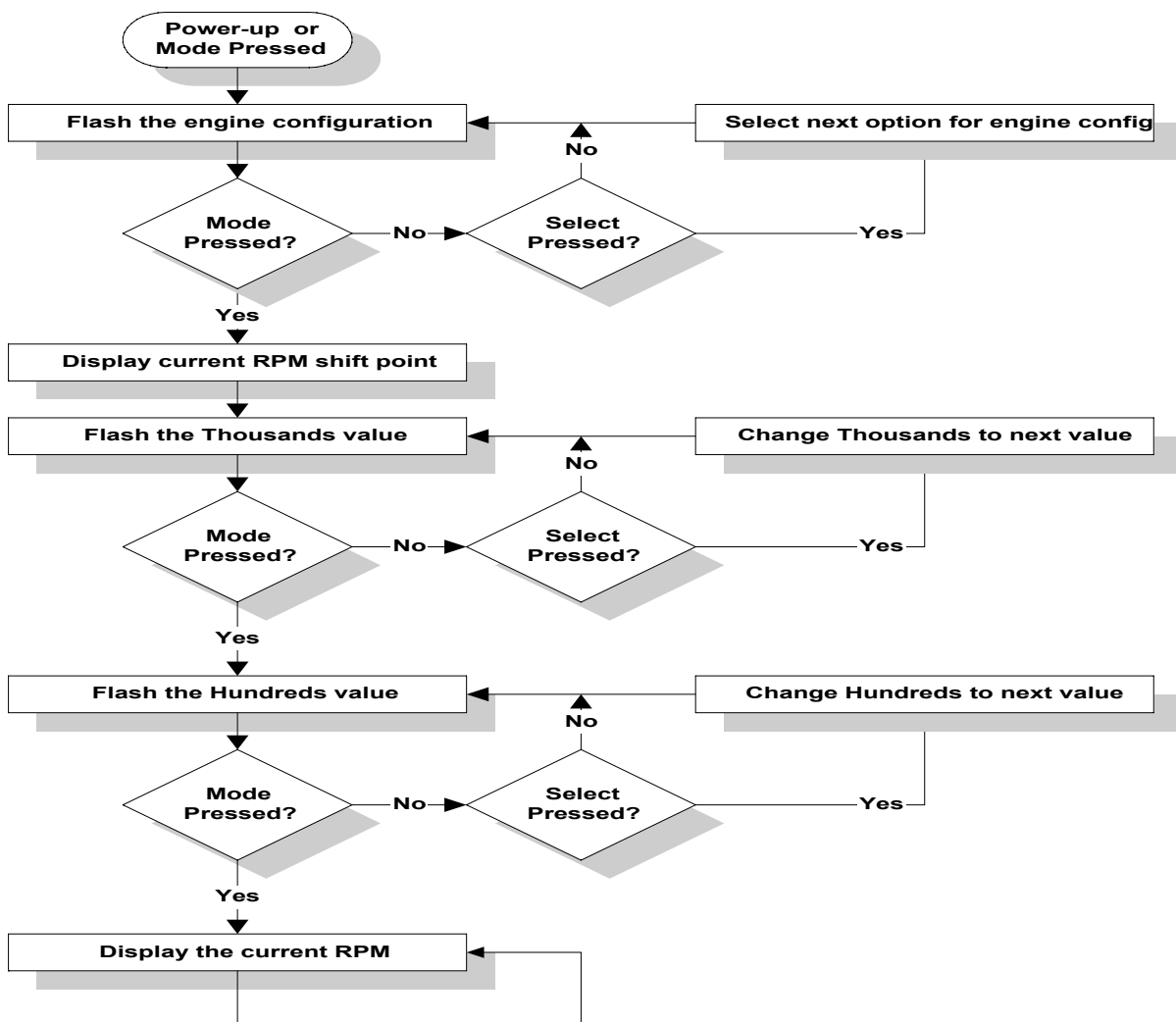
1. Specify a 4-, 6-, or 8- cylinder engine.
2. Specify the thousands digit for the required shift rpm.
3. Specify the hundreds digit for the required shift rpm.

During Step 1, the current engine configuration is indicated by displaying *E* and flashing 4, 6, or 8. Use the Select button to move among the 3 options. When the required configuration displays, press the Mode button to advance the sequence to Step 2. During Steps 2 and 3, the current shift rpm, in units of 100 rpm, is displayed. For Step 2, the thousands digit flashes to signify that it is being configured. Pressing the Select button advances the thousands digit from 1 through 7.



Pressing the Mode button saves the thousands value and advances the sequence to Step 3. For Step 3, the hundreds digit flashes. Pressing the Select button now advances the hundreds digit from 0 through 9. Pressing the Mode button now saves the hundreds value and exits the configuration sequence. The flowchart in Figure 2 shows the configuration sequence. Note that the default configuration on power-up is for an 8- cylinder engine and a shift point of 6000 rpm.

Figure 2. Configuration Sequence Flowchart



The shift indicator used in this application is a large (10mm), ultra-bright LED. The LED was chosen over a conventional 12V lamp because it uses less power, requires no reflector, is more reliable, and is much easier to mount to a printed cir-

cuit board. If preferred, a lamp could be substituted with minor changes to the drive circuitry.

The Shift Indicator is forced off during configuration and while the rpm is outside the operational limits. During normal operation within the limits, the indicator is turned off until the rpm is equal to or exceeds the shift point. If the shift point is exceeded by 500 rpm or more, the Shift Indicator flashes rapidly.

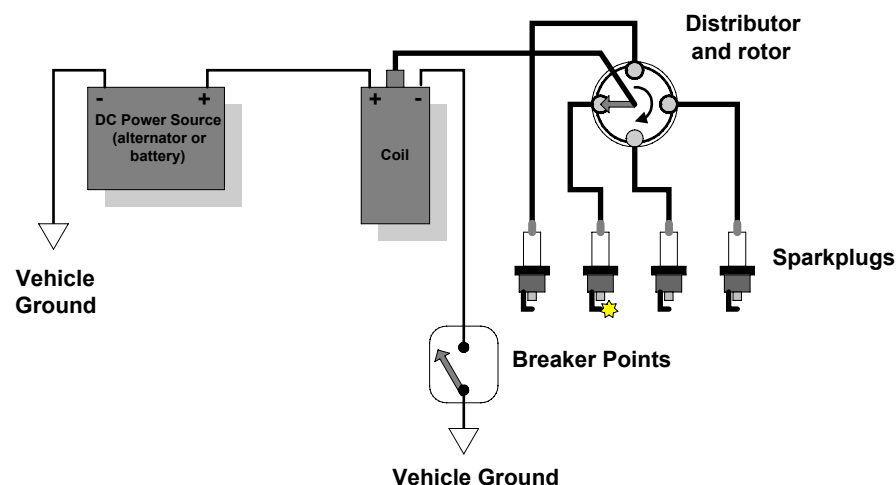
Automotive Ignition Systems

The following brief overview of automotive ignition system operation leads to a better understanding of this application's functions. Figure 3 provides a simplified block diagram of a conventional system for a 4-cylinder engine. The engine consists of the following major components:

- A DC power source
- An energy storage device/voltage converter (coil)
- Ignition breaker points
- High-voltage distributor and rotor
- Spark plugs

A 12V battery supplies the DC power until the engine is running. With the engine running, an alternator (AC generator with rectified and regulated DC output) recharges the battery and supplies vehicle power. The ignition system converts the DC power into a high-voltage spark, which ignites the air/fuel mixture in each of the engine's cylinders at the appropriate time.

Figure 3. Simplified Automotive Ignition Circuit





The ignition system essentially consists of two circuits, a low-voltage circuit and a high-voltage circuit. The coil stores energy while the breaker points are closed and current is flowing in the low-voltage circuit. When the breaker points open, current flow in this circuit is interrupted, and the voltage of the coil primary winding rises rapidly. The coil has a secondary winding that is part of the high-voltage circuit. The ratio of the coil windings is approximately 1:400. As a result, the voltage in the high-voltage circuit rises to a peak of about 40KV, which is sufficient to jump the gap at the spark plug and ignite the air/fuel mixture in the engine cylinder. The spark continues until the energy stored in the coil is dissipated. The position of the rotor within the distributor controls the timing of this spark. The distributor and rotor effectively act as high voltage switches for the spark plugs.

In the past two decades, variations of the basic ignition system were created to improve performance and reliability. Most notably, the mechanical breaker points were replaced by an electronic switch, which is now referred to as electronic ignition. In some instances, multiple coils are used to eliminate the distributor and rotor. Discussing these systems is beyond the scope of this document. Fortunately, however, all these systems have a common characteristic. For all systems, the voltage at the negative terminal of the coil can be monitored to determine the engine RPM. When the breaker points are closed, the voltage is near 0. When the points are open, voltage rises to approximately 12-15V, which is the ignition system DC voltage. The frequency of this waveform is a direct indication of the engine speed.

Processing Analysis

For single-coil systems, the relationship between ignition frequency and rpm is as follows:

$$\text{rpm} = [60 \times (\text{ignition frequency}) \times 2] \div (\#\text{engine cylinders})$$

Where:

- 60 is the number of seconds in a minute
- 2 is the number of engine revolutions required to complete an ignition cycle for a 4-stroke engine

For a microcontroller-based application, it is easier to measure the period of the ignition waveform rather than the frequency. Therefore the equation can now be simplified and rewritten as follows:

$$\text{rpm} = 120 \div (\text{ignition period} \times \#\text{engine cylinders})$$

Because the microcontroller is measuring the ignition period in microseconds and it is simpler not to work with fractions of a second, the equation can be further rewritten as follows:



$$\text{rpm} = 120,000,000 \div [(\text{ignition period in } \mu\text{sec.}) \times (\text{\#engine cylinders})]$$

This application supports 4-, 6-, and 8-cylinder engines and displays the rpm in units of hundred rpm, allowing the equation to be reduced to its final form:

$$\text{Hundred rpm} = \text{Constant} \div \text{ignition period in } \mu\text{sec.}$$

Where:

- Constant = 300,000 for 4-cylinder engines
- Constant = 200,000 for 6-cylinder engines
- Constant = 150,000 for 8-cylinder engines

The number of bytes required to represent the ignition period is a direct function of the low rpm operational limit. The accuracy of the high rpm measurement is a direct function of the timer's minimum resolution. Increasing the resolution to obtain greater accuracy causes the number of ignition period bytes to increase, adding complexity to the rpm calculation. The optimal solution is to set the oscillator frequency to the lowest value that assures accuracy but does not force the data words to become unnecessarily large. This value is derived from analysis of the operational parameters. For this application, the shift point upper limit is 7900 rpm and the resolution is 100 rpm. Therefore, the worst case is distinguishing between 7800 rpm and 7900 rpm for the 8-cylinder engine. The ignition period for this scenario is 1898 microseconds at 7900 rpm and 1923 microseconds at 7800 rpm, resulting in a difference of only 25 microseconds. By selecting an 8 MHz oscillator frequency, the timers have a resolution of 1 microsecond and the instructions all execute in 1.25 microseconds. Put into these terms, the worse case scenario represents 25 timer counts or approximately 20 instructions. By designing the firmware so that the ignition interrupt is generally unobstructed by other processing, this analysis demonstrates that an 8 MHz clock rate provides sufficient accuracy for distinguishing the upper rpm values. With the accuracy validated, the next concern is the number of bytes required to represent the ignition period at the shift point lower limit. The 4-cylinder engine represents the worst case scenario for an ignition period with a value of 60,000 microseconds at 500 rpm. This value is represented in 16 bits, or 2 bytes. However, the worst-case rpm calculation constant is 300,000, requiring 19 bits, or 3 bytes. Therefore, the rpm calculation, requires an unsigned 24-bit by 16-bit division routine yielding an 8-bit quotient as the hundred rpm value. A division of this format is not too complex to program. Therefore, the 8 MHz oscillator frequency is a good choice for this application.



System Design

The Appendix provides a schematic for this application. As shown in the schematic, the microcontroller (U1) used in the application is the Z8PE003, which is from ZiLOG's Z8Plus™ microcontroller family. The Z8PE003 is an 8-bit controller with 1K bytes of One-Time Programmable (OTP) program memory, 64 bytes of RAM, two 16-bit counter-timers, 14 general purpose I/O pins, and an analog comparator. The I/O pins are configured as a 6-bit port, Port B, and an 8-bit port Port A. All of the I/O pins can be configured individually as inputs or outputs.

The microcontroller RAM is configured as 4 register banks, Bank0 through Bank3, with each bank containing 16 registers, R0 through R15. The registers of Bank0 are dedicated for processing variables and state registers as shown in Table 1. Bank1 is used to hold the most recent eight ignition time values so that a running average can be calculated. The hundred rpm calculation uses this average value. Bank2 is allocated as a general-purpose area for intermediate calculations. Bank3 is reserved for the firmware stack.

Table 1. ZA8PE003 RAM Map

Address	Name	Description
00h	AppState	Application State register—contains system operational flags/status
01h	ShiftRpm	Shift Point RPM—decimal representation of shift point in hundred rpm
02h	EngineCode	Engine Code—number of engine cylinders (4, 6, or 8)
03h	IgnUpdPntr	Ignition Update Pointer—location of oldest ignition time in workspace
04h	IgnTimeMS	Ignition Time Most Significant byte—upper byte of latest ignition time
05h	IgnTimeLS	Ignition Time Least Significant byte—lower byte of latest ignition time
06h	TimeOut	Value of RTC most significant byte corresponding to ignition time-out
07h		Unused
08h	IgnAvgMS	Ignition Average Most Significant byte—upper byte of average ignition time
09h	IgnAvgLS	Ignition Average Least Significant byte—lower byte of average ignition time
0Ah	AvgRPM	Average RPM—results of Hundred RPM calculation
0Bh	KRpmDsply	Thousand RPM Display—LED segment pattern for latest Thousands digit
0Ch	HRpmDsply	Hundred RPM Display—LED segment pattern for latest Hundreds digit



Table 1. ZA8PE003 RAM Map (Continued)

Address	Name	Description
0Dh	DsplyState	Display State register—contains display flashing and shift indicator status
0Eh	RTCMSbyte	Real Time Clock Most Significant byte—upper byte of 24-bit word
0Fh	RTCMDbyte	Real Time Clock Middle byte—middle byte of 24-bit word
10h–1Fh	Bank1	Ignition Time Averaging workspace (8 16-bit values, msbyte @ even address)
20h–2Fh	Bank2	General Purpose workspace
30h–3Fh	Bank3	Firmware Stack

To provide an accurate time measurement, a crystal oscillator is used for the processor clock. Oscillator pins XTAL1 and XTAL2 are connected to the crystal (Y1) and to the 22pF stabilization capacitors, C1 and C2. Although the processor is capable of operating at 10 MHz, 8 MHz is selected as the optimum value. By utilizing an 8-MHz crystal, the microcontroller’s timers operate with a resolution of 1 microsecond, because the timer clocks are fixed at 1/8 of the oscillator frequency by the internal clock logic.

The microcontroller’s timers are constructed as a fixed 16-bit timer, T23, and two 8-bit timers, T0 and T1, which can be operated as a single 16-bit timer or as individual 8-bit timers. Because the ignition period requires up to 16 bits, timer T23 is selected for use as the ignition period timer. The timer is started on the rising edge of an ignition pulse and stopped on the next rising edge. Operated in this manner, the timer value represents the ignition period in microseconds. If the timer overflows, the overflow is an indication that the engine speed is below the operational limits of the unit. When that occurs, the period measurement is aborted and a new measurement is attempted on the next rising edge. For the display, it is desirable to multiplex the rpm digits at approximately 60 Hz and flash the digits at approximately 2 Hz. The multiplex step requires a 24-bit real-time clock (RTC). The RTC is implemented by using T0 as the least significant byte and using 2 register values as the upper bytes. T0 is configured to run continuously. Each time T0 overflows, the resulting interrupt triggers an update of the more significant bytes. The firmware determines the display setting based upon the logic level of the RTC bit that toggles nearest the required rate. Timer T1 is unused.

Port A is configured as all outputs for driving the displays. Bits 0 through 6 (PB0-PB6) drive the 7 segments of the LED display (U2) while bit 7 (PB7) drives the shift indicator LED (D1). The 7-segment display is a 2-digit common anode device with multiplexed segment pins. A separate common anode pin is provided for each digit. The digits are displayed by applying 5V at the common anode pin of the intended digit and driving the segment cathodes with a pattern that activates



the required segments. Resistors R1 through R7 are included to limit the current in each segment driver pin to approximately 10mA. Transistors Q1 and Q2 switch 5V to the common anode pins. The transistors are necessary because the current approaches 70mA when all of the segments are active, which is beyond the drive capability of the microcontroller output. Port B pins PB0 and PB1 are configured as outputs to drive Q1 and Q2. The shift indicator LED is directly driven by PB7 with R8 in line to limit the current to approximately 10mA. Note that the display intensity is a direct function of the current used to drive the LEDs. For this Application Note, directly driving the LEDs via the processor is acceptable to demonstrate use of the processor. However, for actual vehicle installation, it is more desirable to add a higher-current LED driver to increase display visibility.

The four remaining Port B pins are used as inputs. Pins PB2 and PB5 are used for the Mode and Select buttons, while pins PB3 and PB4 are used for the ignition system interface. Because the buttons are normally open, the Mode and Select inputs are held at logic level 1 by pull-up resistors R9 and R12. When the buttons are depressed, they short the microcontroller input pins to ground, creating a logic level 0 input. Note that pin PB5 also has a diode (D2) connecting it to V_{CC} . This diode provides over-voltage protection for the pin because it does not have an internal protection diode. Pin PB2 is used for the MODE function because it can be configured to provide an interrupt request to the processor. It is via this request that normal processing is suspended and the processor enters CONFIGURATION mode. In CONFIGURATION mode, pin PB5 (select) is polled to check for required changes in configuration. Pins PB3 and PB4 are configured for ANALOG mode. In ANALOG mode, PB4 is used as the positive input and PB3 is used as the negative input to a comparator. The negative terminal of the ignition system coil is connected to PB4 via a voltage divider formed by R10 and R11. The voltage divider is necessary because the ignition system voltages exceed the limits of the microcontroller inputs. PB4 is connected to the center point of the divider.

Another voltage divider is formed by resistors R14 and R15. PB3 is connected to the center point of this divider to create a reference voltage at the comparator's negative input. When the voltage at PB4 exceeds the PB3 reference voltage, the comparator output switches from logic level 0 to logic level 1. The comparator output transitions generate interrupts to the processor, which are used to determine the period of the ignition waveform. For this application, the reference voltage is set to approximately 1.6V. Because the purpose is to provide compensation for the voltage at the negative terminal of the coil when the breaker points are closed, the reference voltage must be at least 0.5V greater than the voltage at the negative terminal. The setting of 1.6V should be sufficient for most ignition systems.

A 5V linear voltage regulator (U3) supplies steady 5V power to the application even though the vehicle voltage may range from 12V to 15V. The input side of the regulator is stabilized by a 10- μ F capacitor (C3). The 5V output is stabilized by a 0.1 μ F capacitor (C4). Although nothing is shown in the schematic, the addition of a transient suppression device and reverse voltage protection to the voltage regu-



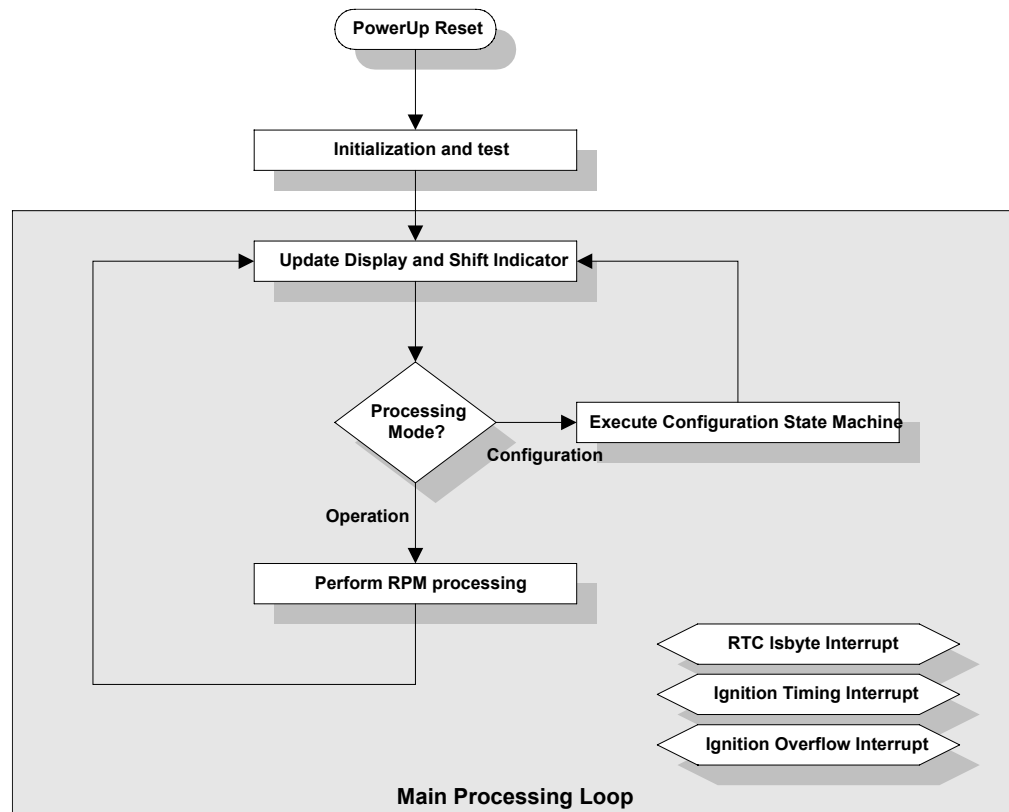
lator input is highly recommended if the application is to be installed in a vehicle. This recommendation is because the automotive electrical system is a very noisy environment. Brief high-voltage transients are possible from the various motors and solenoids connected to the system. Also, the voltage regulator can briefly go over-voltage while attempting to deal with drastic changes in load current.

Firmware

Most of the functions in this application reside in the Z8's firmware. This design feature allows enhanced functionality, such as real-time display of the rpm, to be included for very little additional cost compared to a similar analog solution. Figure 4 contains a high-level flowchart for the firmware. After power-up and initialization, the firmware enters a main processing loop that consists of either configuration or operational processing. This loop is executed continuously and is only suspended momentarily during operational processing to service either the Real Time Clock (RTC) interrupt, ignition timing interrupt, or the ignition overflow interrupt. The following paragraphs contain a detailed overview of the firmware functions.

The initialization and test segment of the code is responsible for initializing the microcontroller's I/O ports, configuration registers, and peripherals, and for executing a simple LED test. The first instruction executed after power-up disables the watch-dog timer (WDT). The processor requires that WDT configuration be performed on the first instruction. For this application, the WDT is disabled because it is not used. The remaining configuration registers are initialized next. After initializing the configuration registers, RAM is cleared and the application registers are initialized. The initialization configures the system so that it immediately enters the CONFIGURATION PROCESSING mode before performing any rpm processing. Configuration processing assures that the system is properly configured for the vehicle before rpm information is displayed. The LED test is the final function of the initialization and test code. The test turns on all segments for each display digit and the shift indicator for approximately 1/2 second, allowing the operator to verify that the display and indicator are functioning properly. On completion of the LED test, the interrupts are cleared, the RTC is started, and the main processing loop is entered.

Figure 4. High Level Firmware Flowchart



The main processing loop consists of four major code segments:

- Display update
- Processing mode decision
- Configuration state update
- RPM processing

Display update is performed on every loop cycle. However, the remainder of the processing is either configuration update or rpm processing. The processing decision segment determines the processing to be performed.

The display update code segment manages the multiplexed data of the rpm display, flashes the digits as required, turns the shift indicator on or off, and flashes the shift indicator. Table 2 defines the bits of the application state register. The status of the display state register, detailed in Table 3, and the value of the RTC control the display. The display state register contains an enable bit for flashing each



digit of the rpm display (bits 3 and 4) as well as the shift indicator (bit 5). It also contains the state bit (bit 7) that specifies whether or not the shift indicator is active. The *flash bits* are set by the configuration update or rpm processing code based upon the current situation. Bit 5 of the middle RTC byte (address 0Fh) determines which digit is enabled. This bit toggles approximately every 8.2msec, corresponding to a multiplex rate of about 61 Hz. The thousands digit for the rpm is enabled when the bit is 0 and the hundreds digit is enabled when it is 1. Bit 2 of the most significant RTC byte (address 0Eh) determines if a flashing digit is on or off. This bit toggles approximately every 262 msec, corresponding to a flash rate of about 2 Hz. If bit 2 is 0 and the corresponding flash bit is enabled, the digit is blanked rather than displayed. Bit 7 of the middle RTC byte determines if the shift indicator is on or off when it is enabled to flash. This bit toggles approximately every 32.8 msec, corresponding to a flash rate of about 15 Hz.

Table 2. Application State Register Bit Definitions

Bit	Name	Description
0	IgnActBit	When 1, indicates an ignition timing cycle is in progress
1	IgnUpdate	When 1, indicates a new ignition time is available for processing
2	ECCfgBit	When 1, indicates engine code is being configured
3	KSCfgBit	When 1, indicates thousands rpm shift point is being configured
4	HSCfgBit	When 1, indicates hundreds rpm shift point is being configured
5		Unused
6	RPMOut	When 1, indicates operational rpm limits have been exceeded
7	IgnDead	When 1, indicates ignition pulses are not being detected

Table 3. Display State Register Bit Definitions

Bit	Name	Description
0		Unused
1		Unused
2		Unused
3	BlinkDig1	When 1, indicates that the left digit of the RPM Display is flashing
4	BlinkDig2	When 1, indicates that the right digit of the RPM Display is flashing
5	BlinkShft	When 1, indicates that the shift indicator is flashing
6		Unused
7	ShftLEDBit	When 1, indicates that the shift indicator is turned on

After updating the display and shift indicator, the firmware determines what type of processing to perform. This decision is based on the setting of the three configu-



ration state bits (bits 2, 3, and 4) in the application state register (detailed in Table 2). These bits correspond to the three stages of the configuration process. Bit 2 corresponds to the engine code configuration state. Bits 3 and 4 correspond to setting the shift point rpm. Bit 3 denotes the thousand rpm configuration state and bit 4 denotes the hundred rpm configuration state. If none of the bits are set, operational processing is performed.

The three configuration processing states effectively form a state machine that configures the unit for operational processing. When the state machine is entered, the states are stepped through in sequence. All operational processing is suspended until the state machine is finished. In general, the state machine operates based on the user pressing either the Mode button or the Select button. Pressing the Select button rotates through the setting options for the given state, while pressing the Mode button causes a jump to the next state. In the first state, the engine configuration code is established. The engine code options are 4, 6, and 8, corresponding to the number of engine cylinders. In this state, the display is configured as *Ex*, where *x* is the engine code, which is flashing. In the second state, the thousand rpm digit is specified for the shift rpm. The values for this state are 1 through 7 corresponding to 1000 through 7000 rpm. While in this state, the shift rpm is displayed as *KH*, where *K* is the thousand rpm value, which is flashing, and *H* is the hundred rpm value. In the third and final state, the hundred rpm digit is specified for the shift rpm. The values in this state are 0 through 9, corresponding to 0 through 900 rpm. While in this state, the display is the same as for state 2, except the hundred rpm digit is flashing.

Operational processing consists of waiting for an updated ignition period, updating the average period with the new value, and calculating the hundred rpm value from the updated average. No significant processing is performed until a new timing value is detected. The reception of ignition pulses drives the process of determining a new timing value.

Detection of a rising ignition pulse generates an IRQ4 interrupt. This interrupt is always unmasked while the system is operational. The firmware immediately vectors to the interrupt handler, which determines if this interrupt is the start of a new timing cycle or the end of the current one. This determination is made by checking the status of IgnActBit (bit 0) in the Application State register. This bit is set on the start of a cycle and cleared on completion. If a start situation is determined, the T23 16-bit counter is enabled and IgnActBit is set. The interrupt handler returns to the original routine. If an end of cycle is determined, the T23 timer is immediately stopped. The time difference is calculated from the counter value and is stored in the ignition time averaging workspace by overwriting the oldest value. The fact that a new timing value is available is denoted in the Application State register by setting IgnUpdate (bit 1). This bit is reset to denote the end of the timing cycle and RPMOut (bit 6) and IgnDead (bit 7) are cleared because the cycle was successfully completed. Finally, the T23 timer is reloaded and the Timeout register



(address 06h) is initialized to the value of the RTC most significant byte. The handler then returns to the original routine.

When the operational code segment detects from IgnUpdate (bit 1) that a new timing value is available, it begins the rpm calculation. The first step of the calculation updates the average ignition time. An average time is used to help compensate for minor errors in measuring the ignition time. The average is calculated by adding the most recent eight timing values and dividing the sum by 8. The Hundred RPM value is determined from the 16-bit average ignition time and a 24-bit constant. A unique constant value exists for each of the supported engine configurations. The proper constant is selected for the configuration and a 24-bit by 16-bit unsigned division is performed, which results in an 8-bit quotient. The result is compared to the operational limits to determine if the result is out of range. If the result is outside the operational limits, RPMOut (bit 6) is set. Otherwise, the result is converted to a decimal value for display. A table is consulted to convert each digit of the value into its respective 7-segment pattern. The value is also compared to the shift point. If the value is greater than the shift point, ShftLEDBit (bit 7) of the Display State register is set. Otherwise, ShftLEDBit (bit 7) is cleared. If the value is greater than the shift point by 500 rpm or more, BlinkShft (bit 5) is set. Otherwise, the bit is cleared. Finally, IgnUpdate (bit 1) is cleared, allowing another timing cycle to begin, and the routine returns to the beginning of the loop.

The operational processing segment must also deal with error conditions, such as rpm values that are outside the operational limits and engines that stop running. When an ignition period is obtained, but the value is outside the operational range, RPMOut (bit 6) is set in the Application State register. In very low rpm situations, the 16-bit timer may overflow, resulting in the generation of an interrupt (IRQ5). The interrupt handler stops the T23 timer, reloads it for the next cycle, clears IgnActBit (bit 0) to denote that the cycle was aborted, and sets RPMOut (bit 6). If the ignition stops, IgnDead (bit 7) is set in the Application State register. This situation is detected by comparing the most significant byte of the RTC to the contents of the TimeOut register. The TimeOut register is loaded with the current value of the RTC most significant byte each time an ignition period is successfully measured. If the RTC most significant byte is ever equal to the TimeOut value, then there have been no ignition pulses for approximately 16 seconds and IgnDead (bit 7) is set. If either RPMOut (bit 6) or IgnDead (bit 7) is set, the display is forced to the “- -” value and the shift indicator is turned off.

Summary

This application effectively demonstrates the replacement of analog circuitry with a digital microcontroller for increased accuracy and ease of use. It also demonstrates that when a microcontroller solution is chosen, additional features and functions are implemented easily with little or no additional components.



Technical Support

Assembling the Application Code

Any Z8 assembler can be used to assemble the application code, but use of ZiLOG Developer Studio (ZDS) is recommended. This integrated suite of software tools allows for program file handling, editing, real-time emulation and debugging when used with the appropriate emulator. Future versions of the ZDS incorporate a C-Compiler, simulator and trace buffer. See ZiLOG's web page at www.zilog.com for news and free downloads of the ZDS.

Place the `.ASM` file and `.INC` files in their own subdirectory. Invoke ZDS and select a new project from the file menu. Under Target Selection, select Family. Under Master Select, select Z8Plus. Under Project Target, select Z8PE003. Select the appropriate emulator type to be used. Browse to fill in the project name by clicking on the "..." key. Select the subdirectory containing the `.ASM` and `.INC` files, name the project, (the extension is added), click Save, and the first ZDS screen reappears with the project name, path and file extension filled in. If everything seems acceptable, click OK.

Click on the Project tab and select Add to Project, then select Files. Double click on the `shift.s` file. This file and the `.INC` file are displayed in the project window. Next, click on the Build tab and select Build. The Output window displays the assembly results. The standard assembler and linker settings produce listing and hex files, along with the ZDS files, in the same subdirectory. Save the project and its files by clicking on the File tab and selecting these options. The ZDS Project File is included and when the ZDS is installed, skip the above steps for program assembly.

Programming a One-Time Programmable (OTP) is accomplished by selecting the OTP option with the hex code installed. Never install the OTP until access to it is required, either for blank checking, verification or programming. Insert a blank Z8PE003 into the OTP socket and click on the program OTP selection. Differences exist between earlier GUIs and the ZDS, so read and understand the operation of the software in use. Pad unused memory locations with `FFh` before programming. If padding unused memory locations is not consistently approached, differences in the checksum can occur.

Application Source Code

This application utilizes the following source files:

- `shift.s`—the main code
- `RegDef.inc`—constant definitions and register assignments



Instead of displaying each file separately, the files are listed exactly in the order and location they are <.included> in the main source, shift.s. This listing order is similar to the output listing file ShiftLight.lst.

```
*****
*      Module Name:      Digital Shift Indicator
*      Copyright:       ZiLOG Inc.
*      Date:            10/18/99
*      Created by:      John D. Conder
*      Modified by:
*
*      Description:     This module contains the code for using the
*                      Z8PE003 microcontroller to create a digital shift
*                      indicator for motorsports competition.
*
*                      As designed, the shift indicator is applicable to
*                      conventional ignition systems of 4 cycle engines.
*                      It is also programmable, allowing the user to
*                      select between 4, 6, and 8 cylinder engine
*                      configurations and trigger points from 1000
*                      to 7900 RPM in increments of 100 RPM.
*
*
*****
*//////////////////////////////////////*
*//////////////////////////////////////*
*//////////////////////////////////////*
*      Include section
*//////////////////////////////////////*
*
;      Include the register and constant definitions
include      "RegDef.inc"

;=====
;=      TITLE:          RegDef.inc
;=
;=      DATE:           October 14 1999
;=
;=      PURPOSE:       Register and constant definitions for the=
;=                      Shift Indicator app note
;=
;=
;=      FILE TYPE:     .included header file
;=
;=
;=      ASSEMBLER:     ZiLOG ZDS/ZMASM
;=
;=      PROGRAMMER:    John Conder
;=
;=====

;      BIT DEFINITIONS

;      Port A
```



```

;
; pins      6 7 8 9 10 11 12 13  Function          Polarity      I/O
; bits [7 6 5 4 3 2 1 0]
;          | | | | | | | |
;          | | | | | | | | Display SegA      lo-true        0
;          | | | | | | | | Display SegB      lo-true        0
;          | | | | | | | | Display SegC      lo-true        0
;          | | | | | | | | Display SegD      lo-true        0
;          | | | | | | | | Display SegE      lo-true        0
;          | | | | | | | | Display SegF      lo-true        0
;          | | | | | | | | Display SegG      lo-true        0
;          | | | | | | | | Shift LED         lo-true        0
;
PTADIR      .equ  0D2h          ; PortA Direction register

PA_SFR      .equ  00000000b    ; PTASFR(0D3h)- PortA Special Fctn
;          ; 1=Open-Drain, 0=Push-Pull
PA_Dir      .equ  11111111b    ; PTADIR(0D2h)- PortA Direction
;          ; 1=Output, 0=Input
PA_Init     .equ  11111111b    ; PTAOUT(0D1h)- Initial Value
Display     .equ  01111111b    ; LED Display segment bits
ShiftLED    .equ  10000000b    ; LED Shift indicator bit

;
; Port B
;
; pins      na na 5 4 3 2 1 18  Function          Polarity      I/O
; bits [7 6 5 4 3 2 1 0]
;          | | | | | | | |
;          | | | | | | | | Display Dig1      lo-true        0
;          | | | | | | | | Display Dig2      lo-true        0
;          | | | | | | | | Mode Switch      lo-true        I
;          | | | | | | | | Ign Reference      analog        I
;          | | | | | | | | Ignition          analog        I
;          | | | | | | | | Select Switch     lo-true        I
;
PTBDIR      .equ  0D6h          ; PortA Direction register

PB_SFR      .equ  00011100b    ; PTBSFR(0D7h)- PortB Special Fctn
;          xxxxxxx0          ; 1= Enable PB0 as SMR input
;          xxxxxx0x          ; 1= Enable PB1 as T0 output
;          xxxxx1xx          ; 1= Enable PB2 as IRQ3 input
;          xxxxlxxx          ; 1= Analog PB3/4, 0 = Digital
;          xxxlxxxx          ; 1= PB4 Interrupts enabled
;          000xxxxx          ; Reserved - Must be 0

PB_Dir      .equ  00000011b    ; PTBDIR(0D6h)- PortB Direction
;          ; 1=Output, 0=Input
PB_Init     .equ  00000011b    ; PTBOUT(0D5h)- Initial Value
Digit1      .equ  00000001b    ; LED Display Digit 1 bit
Digit2      .equ  00000010b    ; LED Display Digit 2 bit
Digits      .equ  00000011b    ; LED Display digits
M_Switch    .equ  00000100b    ; Mode Switch input bit
Ignition    .equ  00010000b    ; Ignition input bit
S_Switch    .equ  00100000b    ; Select Switch input bit

;
; Timers
; T0 is used as the lsbyte of a 24-bit RTC for display muxing.

```



```

; T2 and T3 are used as a 16-bit ignition time counter (res = 1us).
; WatchDog Timer is unused.

TMR_Init          .equ  00000000b  ; TCTLLO(0C0h)- Timer Control Reg
;                xxxxx000          ; T01 Configuration
;                ;                ; T0  T1  T01
;
;                ; 001=enab      disab
;                ; 010=disab     enab
;                ; 011=enab     enab
;                ; 100=          enab*
;                ; 101=enab*    disab
;                ; 110=disab     enab*
;                ; 111=enab*    enab*
;                ; Note: * = auto reload active
;                xxx00xxxx        ; Reserved - Must be 0
;                xx0xxxxx         ; 1=T23 enabled with auto reload
;                00xxxxxx         ; Reserved - Must be 0
RTC_Start         .equ  00000101b  ; Start Real Time Counter (T0)
Ign_Start        .equ  00100000b  ; Start Ignition Time Counter (T23)

; T0 (Real Time Counter)
;
T0_Reload        .equ  0FFh        ; T0AR(0C8h)- T0 Reload Value
T0_Value         .equ  0FFh        ; T0VAL(0CCh)- T0 Count Value

; T23 (Real Time Counter)
;
T2AR             .equ  0C8h        ; Timer2 auto reload register
T3AR             .equ  0C9h        ; Timer3 auto reload register

T2_Reload        .equ  0FFh        ; T2AR(0C8h)- T2 Reload Value
T3_Reload        .equ  0FFh        ; T3AR(0C9h)- T3 Reload Value
T2_Value         .equ  0FFh        ; T2VAL(0CAh)- T2 Count Value
T3_Value         .equ  0FFh        ; T3VAL(0CBh)- T3 Count Value

; Watch Dog Timer
;
WDT_Init         .equ  00001000b  ; TCTLHI(0C1h)- WDT Configuration
;                xxxxx000          ; Reserved - Must be 0
;                xxxxlxxx         ; 1=Stop Mode disabled, 0=enabled
;                x000xxxx         ; WDT Timeout Value
;                ;                ; 000= disabled
;                ; 001=          65,536 TpC
;                ; 010=         131,072 TpC
;                ; 011=         262,144 TpC
;                ; 100=         524,288 TpC
;                ; 101=        1,048,576 TpC
;                ; 110=        2,097,152 TpC
;                ; 111=        8,388,608 TpC
;                0xxxxxxx         ; 1=WDT enabled in Halt, 0=disabled

; Interrupt definitions
;
IMR_Init         .equ  00000001b  ; IMASK(0FBh)- Interrupt Mask
;                xx000000          ; 0=IRQ5-IRQ0 disabled
;                x0xxxxxx         ; Reserved - Must be 0

```



```

;          0xxxxxxx          ; 0=Global Interrupt disable
IMR_IgnEnab      .equ  00110000b  ; Enable IRQ4(ignition) & IRQ5(time-
out)
IMR_IgnOvr      .equ  00100000b  ; Enable IRQ5(T2) for ignition timeout

IRQ_Init        .equ  00000000b  ; IREQ(0FAh) - Interrupt Request
;          xx000000          ; 0=Clear request bits 5-0
;          00xxxxxx          ; Reserved - Must be 0
IRQ_RTCBit      .equ  00000001b  ; T0 (RTC) interrupt bit
IRQ_ModeBit     .equ  00001000b  ; PB2 (Mode) interrupt bit
IRQ_IgnBit      .equ  00010000b  ; PB4 (Ignition) interrupt bit
IRQ_OvrFlo      .equ  00100000b  ; T2 (Overflow) interrupt bit

```

; Application Definitions

```

;
RegBot          .equ  000h        ; Bottom register
RegTop          .equ  03Fh        ; Top register
StackTop        .equ  RegTop+1    ; Top of Stack

AppRP           .equ  000h        ; Application Register Pointer
AvgRP           .equ  010h        ; Averaging Space Reg Pointer
GenRP           .equ  020h        ; General Purpose Reg Pointer
DsplyBlank      .equ  11111111b   ; Display blank value

```

; Application State Register Bits

```

;
AppInit         .equ  00000100b   ; AppState initialization
IgnActBit       .equ  00000001b   ; Ignition Timing Active State Bit
IgnUpdate       .equ  00000010b   ; Ignition Update Available State Bit
ECCfgBit        .equ  00000100b   ; Engine Code Configuration State
KSCfgBit        .equ  00001000b   ; Thousand Rpm Shift Config State
HSCfgBit        .equ  00010000b   ; Hundred Rpm Shift Config State
RPMOut          .equ  01000000b   ; RPM out of range state bit
IgnDead         .equ  10000000b   ; Ignition dead state bit

ConfigBits      .equ  00011100b   ; Configuration State bits

```

; Display State Register Bits

```

;
DsplyInit       .equ  00010000b   ; DsplyState initialization
BlinkDig1       .equ  00001000b   ; Digit 1 blink enable bit
BlinkDig2       .equ  00010000b   ; Digit 2 blink enable bit
BlinkShft       .equ  00100000b   ; Shift LED blink enable bit
ShftLEDBit     .equ  10000000b   ; Shift LED state bit

BlinkBits       .equ  00111000b   ; Blink Enable bits

```

```

*****
*      Global variables
*****
      DEFINE      REGDATA, SPACE=RFILE

;      RAM MAP
      SEGMENT     REGDATA

;
;      Register Bank0 - AppRP (000h-00Fh)

```



```

AppState          DS      1          ; R0 - Application State Register
; bits [7 6 5 4 3 2 1 0]
;          | | | | | | | |
;          | | | | | | | | Ign Timing Active (1=active, 0=inactive)
;          | | | | | | | | Ign Update Available (1=available)
;          | | | | | | | | Engine Code Configuration State bit
;          | | | | | | | | K RPM Shift Configuration State bit
;          | | | | | | | | H RPM Shift Configuration State bit
;          | | | | | | | | unused
;          | | | | | | | | RPM out of range (1=out of range)
;          | | | | | | | | Ignition Dead (1=dead, 0=active)
;
ShiftRpm          DS      1          ; R1 - Shift LED activation RPM
; bits [7 6 5 4 3 2 1 0]
;          | | | | | | | |
;          | | | | | | | | Hundred RPM value (0-9dec)
;          | | | | | | | | Thousand RPM value (1-7dec)
;
EngineCode        DS      1          ; R2 - Number of engine cylinders
IgnUpdPntr        DS      1          ; R3 - Ignition time workspace pointer
IgnTimeMS         DS      1          ; R4 - Ign Time Min Value ms byte
IgnTimeLS         DS      1          ; R5 - Ign Time Min Value ls byte
TimeOut           DS      1          ; R6 - Ignition timeout value
unused            DS      1          ; R7 - unused
IgnAvgMS          DS      1          ; R8 - Ign Count Average (usec)ms byte
IgnAvgLS          DS      1          ; R9 - Ign Count Average (usec)ls byte
AvgRPM            DS      1          ; R10 - Average RPM value (100 RPM)
KRpmDsply         DS      1          ; R11 - Thousand RPM display value
HRpmDsply         DS      1          ; R12 - Hundred RPM display value
DsplyState        DS      1          ; R13 - LED Display state register

; bits [7 6 5 4 3 2 1 0]
;          | | | | | | | |
;          | | | | | | | | unused
;          | | | | | | | | unused
;          | | | | | | | | unused
;          | | | | | | | | Blink digit 1 (1=blink,0=continuous)
;          | | | | | | | | Blink digit 2 (1=blink,0=continuous)
;          | | | | | | | | Blink shift (1=blink,0=continuous)
;          | | | | | | | | unused
;          | | | | | | | | Shift LED state (0=active)
;
RTCMSbyte         DS      1          ; R14 - MSbyte of the 24bit RTC
RTCMDbyte         DS      1          ; R15 - Midbyte of the 24bit RTC
;
; Register Bank1 - AvgRP (010h-01Fh)
;          DS      16          ; R0-R15 - Ignition time averaging space
;
; Register Bank2 - GenRP (020h-02Fh)
;          DS      16          ; Gen purpose: R0-R7 App,R8-R15 Interrupts
;
; END RAM MAP

```

```

*****
* Global function declarations
*****

```



```

; none

*****
*      Interrupt Vectors
*****

        SEGMENT code

;      vector  reset = Init
vector  irq0  = RTClSB0      ; IRQ0 T0  timeout
vector  irq1  = PB4f        ; IRQ1 PB4 falling edge
vector  irq2  = TMR1        ; IRQ2 T1  timeout
vector  irq3  = ModeButn    ; IRQ3 PB2 falling edge
vector  irq4  = IgnPulse    ; IRQ4 PB4 rising edge
vector  irq5  = Overflow    ; IRQ5 T2  timeout

        DS      012h        ; Locate code @ 0020h
                                ; The offset of 012h is added to
                                ; the address resulting from the
                                ; last vector assignment (00Eh)

Init:
; Initialize Ports and Configuration Registers
        ld      TCTLHI,#WDT_Init    ; Disable Watch Dog Timer
        di                                ; Disable Interrupts
        ld      PTASFR,#PA_SFR      ; Init PortA Special Fctn register
        ld      PTADIR,#PA_Dir      ; Init PortA Direction register
        ld      PTAOUT,#PA_Init     ; Init PortA Output register
        ld      PTBSFR,#PB_SFR      ; Init PortB Special Fctn register
        ld      PTBDIR,#PB_Dir      ; Init PortB Direction register
        ld      PTBOUT,#PB_Init     ; Init PortB Output register
        ld      TCTLLO,#TMR_Init    ; Init Timer control register
        ld      T0VAL,#T0_Value     ; Init T0 value register
        ld      T2VAL,#T2_Value     ; Init T2 value register
        ld      T3VAL,#T3_Value     ; Init T3 value register
        ld      T0ARLO,#T0_Reload   ; Init T0 reload register
        ld      T2AR,#T2_Reload     ; Init T2 reload register
        ld      T3AR,#T3_Reload     ; Init T3 reload register
        ld      IMASK,#IMR_Init     ; Init Interrupt mask
        ld      IREQ,#IRQ_Init      ; Init Interrupt request register
        srp     #AppRP              ; Set reg pointer to General Bank
        ld      SPL,#StackTop       ; Init Stack Pointer

; Init RAM Register Bank for debug clarity
        ld      RegTop,#RegTop-1    ; Init Pointer to Top of Ram
ClrRam:
        clr     @RegTop              ; Clear register
        dec     RegTop              ; Move the pointer down
        jr     nz,ClrRam            ; If not @ reg 0, jump to loop
        clr     RegTop              ; else, clear top

; Init Application Registers
        ld      R0,#AppInit         ; Initialize Application state reg
        ld      R13,#DsplyInit      ; Initialize Display state reg
        ld      R14,#0FFh           ; Initialize the RTC MSbyte
        ld      R15,#0FFh           ; Initialize the RTC Midbyte
        ld      R6,#0FFh            ; Initialize Ignition timeout
        ld      R2,#008h            ; *** Default - Set for 8 cylinder engine
        ld      R1,#060h            ; *** Default - Set shift rpm to 6000

```



```

        ld    R3,#AvgRP           ; Init Ignition Average Workspace Pointer
        ld    R11,#00000110b     ; Init digit 1 with "E"
        ld    R12,#00000000b     ; Init digit 2 with "8"

; Execute LED test
        call  LEDTest           ; Perform LED test

; Prepare for main processing loop
        clr   IREQ              ; Clear all pending interrupts
        ld    TCTLLO,#RTC_Start  ; Start real time clock (T0)
        ei                               ; Enable interrupts

*****
*           MainLoop
*
*   This primary section of the program is essentially a
*   loop that cycles continuously to maintain the LED display and
*   check for user initiated changes in the configuration. Note that
*   on Power-up, the loop is forced directly into configuration prior
*   to performing any RPM processing.
*
*****

MainLoop:
; Maintain the display.
;
;   T0, RTCMDbyte, and RTCMSbyte together form a 24-bit real time
;   clock with a resolution of lusec. This clock is used to maintain
;   the various timing functions of the display. These timing
;   fuctions are:
;   - multiplex digit1 and digit2 at approximately 61Hz
;   - blink any enabled digit or the shift LED
;
; Update the digital display
        or    PTAOUT,#07Fh       ; Clear all display bits
        or    PTBOUT,#003h       ;
        tm    R15,#020h          ; Determine which digit to enable
        jr    z,EnabledDig1     ; If RTC bit 13 = 0, enable digit1
                                   ; else, enable digit2
        and   PTAOUT,HRpmDsply   ; Update display with digit2 value
        tm    R13,#BlinkDig2    ; Determine if digit2 is blinking
        jr    z,Digit2On       ; If zero, not blinking - jump
        tm    R14,#004h         ; else, determine if on or off
        jr    z,Digit2On       ; If zero, turn on digit2
        or    PTBOUT,#Digit2    ; else, turn off digit2
        jr    UpdateShift      ;

Digit2On:
        and   PTBOUT,#~Digit2   ; Turn digit2 on
        jr    UpdateShift      ;

EnabledDig1:
        and   PTAOUT,KRpmDsply  ; Update display with digit1 value
        tm    R13,#BlinkDig1    ; Determine if digit1 is blinking
        jr    z,Digit1On       ; If zero, not blinking - jump
        tm    R14,#004h         ; else, determine if on or off
        jr    z,Digit1On       ; If zero, turn on digit1
        or    PTBOUT,#Digit1    ; else, turn off digit1
        jr    UpdateShift      ;

Digit1On:

```



```

        and    PTBOUT,#~Digit1        ; Turn digit1 on

UpdateShift:
; Update the Shift indicator LED
    tm    R13,#ShftLEDBit            ; Determine if shift is on/off
    jr    z,ShiftLEDOff              ; If zero, shift is inactive
    tm    R13,#BlinkShft            ; else, determine if blinking
    jr    z,ShiftLEDOn              ; If zero, not blinking - turn on
    tm    R15,#080h                  ; else, determine if on/off
    jr    z,ShiftLEDOn              ; If zero, turn on shift LED
ShiftLEDOff:
    or    PTAOUT,#ShiftLED          ; Turn off shift indicator LED
    jr    ProcState                  ;
ShiftLEDOn:
    and    PTAOUT,#~ShiftLED        ; Turn on shift indicator LED

ProcState:
; Determine processing state
; The system will either be in an operational state or one of 3
; configuration states, based upon the setting of the state bits in
; the AppState register.  If any of the state bits are set, the
; system is in a configuration state and all rpm processing is
; disabled.  If no bits are set, the unit is operational and rpm
; processing is performed based upon the system configuration.
; Pressing the Mode button causes a transition to the configuration
; state from the operational state.
;
    tcm    AppState,#KSCfgBit        ; Check for K RPM Shift config
    jr    z,KSCfgState              ; Jump if in that state
;
    tcm    AppState,#HSCfgBit        ; Check for H RPM Shift config
    jp    z,HSCfgState              ; Jump if in that state
;
    tcm    AppState,#ECCfgBit        ; Check for Engine Code Config
    jr    z,ECCfgState              ; Jump if in that state
; otherwise, check mode button
    tm    IREQ,#IRQ_ModeBit          ; Test for Config Mode request
    jp    z,OpState                 ; Jump to Op state if no request
; otherwise, enter config mode
;
; Engine Code Configuration State
;
ECCfgEntry:
    and    IMASK,#~IMR_IgnEnab      ; Disable Ignition interrupts
    and    AppState,#~ConfigBits    ; Clear state bits
    and    AppState,#~AvgValid      ; Clear rpm average valid bit
    or     AppState,#ECCfgBit        ; Place unit in Engine Config mode
    and    DsplyState,#~BlinkBits    ; Clear the blink enable bits
    and    DsplyState,#~ShftLEDBit  ; Clear the shift indicator bit
    or     DsplyState,#BlinkDig2    ; Blink the #cylinders
    call   ModePress                 ; Process the mode button press
    call   ECDisplay                 ; Display the current Engine Code
    jp    MainLoop                  ; Return to start of main loop

ECCfgState
    tcm    IREQ,#IRQ_ModeBit        ; Test for Mode button pressed
    jr    z,KSCfgEntry              ; If true, jump to K Rpm Config
    tcm    PTBIN,#S_Switch          ; else, test Select button

```



```

        jp      z,MainLoop          ; If not pressed, return to loop
        ; else, modify the selection
    call    SelPress              ; Process the select button press
    add    EngineCode,#002h      ; Increase the #engine cylinders
    cp     EngineCode,#008h      ; Check for max value
    jr     ule,ECValueOK         ; Jump around reset if value is OK
    ld     EngineCode,#004h      ; Reset #engine cylinders to 4
ECValueOK:
    call   ECDisplay             ; Display the current Engine Code
    jp     MainLoop
;
; Thousand Rpm Shift Configuration State
;
KSCfgEntry
    and    AppState,#~ConfigBits ; Clear state bits
    or     AppState,#KSCfgBit    ; Place in K Rpm Config mode
    and    DsplyState,#~BlinkBits ; Clear the blink enable bits
    or     DsplyState,#BlinkDig1 ; Blink the K Rpm Digit
    call   ModePress             ; Process the mode button press
    call   RpmDsply
    jp     MainLoop              ; Return to start of main loop

KSCfgState:
    tcm    IREQ,#IRQ_ModeBit     ; Test for Mode button pressed
    jr     z,HSCfgEntry          ; If true, jump to H Rpm Config
    tcm    PTBIN,#S_Switch       ; else, test Select button
    jp     z,MainLoop            ; If not pressed, return to loop
    ; else, modify the selection
    call   SelPress              ; Process the select button press
    add    ShiftRpm,#010h        ; Increase the K Rpm value
    cp     ShiftRpm,#079h        ; Check for max value
    jr     ule,KRpmValueOK       ; Jump around reset if value is OK
    and    ShiftRpm,#00Fh        ; Reset K Rpm to 1
    or     ShiftRpm,#010h        ;
KRpmValueOK:
    call   RpmDsply              ; Display the current Shift Rpm
    jp     MainLoop
;
; Hundred Rpm Shift Configuration State
;
HSCfgEntry:
    and    AppState,#~ConfigBits ; Clear state bits
    or     AppState,#HSCfgBit    ; Place in H Rpm Config mode
    and    DsplyState,#~BlinkBits ; Clear the blink enable bits
    or     DsplyState,#BlinkDig2 ; Blink the H Rpm Digit
    call   ModePress             ; Process the mode button press
    jp     MainLoop              ; Return to start of main loop

HSCfgState:
    tcm    IREQ,#IRQ_ModeBit     ; Test for Mode button pressed
    jr     z,ConfigModeExit      ; If true, jump to exit Config
    tcm    PTBIN,#S_Switch       ; else, test Select button
    jp     z,MainLoop            ; If not pressed, return to loop
    ; else, modify the selection
    call   SelPress              ; Process the select button press
    add    ShiftRpm,#001h        ; Increase the H Rpm value
    da     ShiftRpm              ; Correct for BCD representation
    tm     ShiftRpm,#00Fh        ; Check for overflow into K value

```



```

        jr    nz,HRpmValueOK          ; If 1's in low nibble - ok, jump
        sub   ShiftRpm,#010h         ; else, correct overflow
HRpmValueOK:
        call  RpmDsply               ; Display the current Shift Rpm
        jp    MainLoop
;
; Configuration Mode Exit
;
ConfigModeExit:
        call  ModePress              ; Process the mode button press
        clr   AppState               ; Reinitialize app state reg
        clr   DsplyState             ; Reinitialize display state reg
        ld    KRpmDsply,#0FFh        ; Blank the display
        ld    HRpmDsply,#0FFh        ;
        ld    TimeOut,RTCMSbyte      ; Create timeout value
        sub   TimeOut,#080h          ;
        or    IMASK,#IMR_IgnEnab     ; Re-enable Ignition interrupts
        and   IREQ,#001h             ; Clear pending interrupts
        jp    MainLoop               ; Return to the processing loop

OpState:
; Operational State
; Update the ignition timing average data with the latest time if
; available. The data is updated via a circular pointer that
; rotates through the 8 16-bit Registers allocated for storage.
; Via this method, each value remains in the calculation for 8
; cycles.
;
        tcm   AppState,#RPMOut       ; Check for ign timing overflow
        jp    z,RpmInvalid            ; Jump on overflow
        tcm   AppState,#IgnUpdate     ; else,check for updated time
        jr    z,ProcUpdate            ; Jump if update available
        tcm   AppState,#IgnDead       ; else,check for dead ignition
        jp    z,RpmInvalid            ; Jump if ignition is dead
        cp    R6,R14                  ; else, check for timeout
        jp    ne,MainLoop              ; Jump if no timeout
        or    AppState,#IgnDead       ; else, ignition is dead
        jp    RpmInvalid              ;

ProcUpdate:
        push  RP                      ; Save the register pointer
        srp   #GenRP                  ; Point to general workspace
        ld    R0,#008h                ; Initialize loop counter
        clr   R1                      ; Clear summation ms byte
        clr   R2                      ; middle byte
        clr   R3                      ; least significant byte

UpdAvgSum:
        inc   IgnUpdPntr              ; Bump pointer to ls byte
        add   R3,@IgnUpdPntr          ; Add ignition time ls byte
        dec   IgnUpdPntr              ; Move pointer to ms byte
        adc   R2,@IgnUpdPntr          ; Add ignition time ms byte
        adc   R1,#000h                ; Add carry to summation ms byte
        add   IgnUpdPntr,#002h        ; Move pointer to next pair
        cp    IgnUpdPntr,#AvgRP+16   ; Compare pointer to top of space
        jr    ult,PointerOK           ; If < top of workspace, OK
        ld    IgnUpdPntr,#AvgRP      ; else, force to bottom

PointerOK:
        djnz  R0,UpdAvgSum            ; Dec loop count, jump if not 0
;

```



```

; Divide by 8 to complete the average calculation
ld R0,#003h ; Initialize division loop counter
UpdAvgDivLp:
rcf ; Clear carry for ms byte shift
rrc R1 ; Shift ms byte
rrc R2 ; Shift middle byte
rrc R3 ; Shift ls byte
djnz R0,UpdAvgDivLp ; Jump if not at end of loop
; otherwise division complete
ld IgnAvgLS,R3 ; Store Average ls byte
ld IgnAvgMS,R2 ; Store Average ms byte

; Calculate Average RPM
; RPM=(Conversion Constant/Average Ignition Time)
;
RPMAvgCalc:
; Prepare for division subroutine
ld R10,#>RpmCnvCnst ; Create Conversion Constant pn
ld R11,#<RpmCnvCnst ;
ld R12,EngineCode ; Create table offset for engine
sub R12,#004h ; Adjust for "0"
rl R12 ; Mpy by 2 (each constant 4 bytes)
add R11,R12 ; Add offset to pointer address
adc R10,#000h ;
ldc R4,@RR10 ; Obtain ms byte of constant
incw RR10 ; Move the pointer
ldc R5,@RR10 ; Obtain mid byte of constant
incw RR10 ; Move the pointer
ldc R6,@RR10 ; Obtain ls byte of constant

call Div24x16 ; RPM=(constant/average ign time)
; R6 contains quotient

; Verify RPM range prior to saving for display
cp R6,#005h ; Check for RPM under 500
jr ult,RpmOvrUnder ; Jump to invalid display if below 500
cp R6,#05Ah ; Check for RPM over 9000
jr ugt,RpmOvrUnder ; Jump to invalid display if over 9000
; otherwise, check for shift limit
call Hex2Dec ; Obtain decimal RPM value for compare
ld AvgRPM,R6 ; Store decimal RPM
cp R6,ShiftRpm ; Compare RPM to Shift limit value
jr ult,NoShiftLED ; Jump to disable LED if < limit
; otherwise, activate shift LED
or DsplyState,#ShftLEDBit ; Activate shift state bit
ld R0,ShiftRpm ; Create shift+500rpm
add R0,#005h ;
cp R6,R0 ; Compare RPM to shift+500
jr ult,NoShftBlink ; If < shift+500, jump around
or DsplyState,#BlinkShft ; else, activate shift blinking
jr DisplayGen ;

NoShiftLED:
and DsplyState,#~ShftLEDBit ; Deactivate shift state bit
NoShftBlink:
and DsplyState,#~BlinkShft ; Deactivate blinking flag

; Create display values
DisplayGen:
ld R1,R6 ; Prepare for conversion

```



```

        call   BCDto7Seg          ; Convert H RPM to display pattern
        ld    HRpmDsply,R1       ; Store the pattern
        ld    R1,AvgRPM         ; Restore RPM
        swap  R1                 ; Prepare for conversion
        call  BCDto7Seg          ; Convert K RPM to display pattern
        ld    KRpmDsply,R1      ; Store the pattern
        jr    ProcLpEnd         ; Finish loop processing
RpmInvalid:
        push  RP                 ;
RpmOvrUnder:
        ld    KRpmDsply,#0BFh   ; Sore pattern for "--"
        ld    HRpmDsply,#0BFh   ;
        and   DsplyState,#~ShftLEDBit ; Deactivate shift state bit
ProcLpEnd:
        and   AppState,#~IgnUpdate ; Clear update available flag
        pop   RP                 ; Restore register pointer
        jp    MainLoop          ; Return to beginning of loop
    
```

```

*****
*   Function Name:      LEDTest
*
*   Returns:           Nothing
*   Entry values:      None
*   Description:       This routine activates all LEDs momentarily to
*                       verify they are functioning.
*   Notes:             This routine leaves the RPM display with "E8"
*                       being displayed.
*****
    
```

```

LEDTest:
        push  RP                 ; Save register pointer
        srp   #GenRP            ; Point to general space
        ld    PTAOUT,#080h     ; Turn on all Display segments
        ld    PTBOUT,#002h     ; Enable Digit1
        ld    R1,#003h         ; Setup delay of half a second
        call  WaitNmsec        ; Wait
        ld    PTBOUT,#001h     ; Enable Digit2
        ld    R1,#003h         ; Setup delay of half a second
        call  WaitNmsec        ; Wait
        ld    PTBOUT,#003h     ; Turn off the Display
        ld    PTAOUT,#07Fh     ; Turn on Shift Indicator
        ld    R1,#003h         ; Setup delay of half a second
        call  WaitNmsec        ; Wait
        ld    PTAOUT,#0FFh     ; Turn off indicator & segments
        ld    KRpmDsply,#086h  ; Load K digit with "E"
        ld    HRpmDsply,#080h  ; Load H digit with "8"
        pop   RP                 ; replace original reg pointer
        ret
    
```

```

*****
*   Function Name:      WaitNmsec
*
*   Returns:           Nothing
*   Entry values:      RP=GenRP, R1=N value
*   Description:       This routine creates a N*0.16sec delay
*   Notes:             Loop = N*((2*1.25us*65536)+ 2*1.25us)
*****
    
```



```

*                               assuming OSC = 8MHz (1.25us instruction time)
*
*****
WaitNmsec:
    clr    R2                    ; Init Midbyte
    clr    R3                    ; Init LSbyte
WNmsecLoop:
    decw   RR2                  ; Decrement lower 16 bits
    jr     nz,WNmsecLoop        ; Loop until = 0
    dec    R1                    ; Decrement upper 8 bits
    jr     nz,WNmsecLoop        ; Loop until = 0
    ret

*****
* Function Name:                ModePress
*
* Returns:                     Nothing
* Entry values:                None
* Description:                 This routine waits for the mode button to resettle
*                               back to "1" after being pressed.
* Notes:                      The display is blanked while the button is pressed
*
*****
ModePress:
    push   RP                    ; Save register pointer
    srp    #GenRP                ; Move to general space
    or     PTAOUT,#ShiftLED      ; Blank the shift indicator
    or     PTBOUT,#Digits        ; Blank the display digits
Wait4ModeRel:
    and    IREQ,#~IRQ_ModeBit    ; Clear the mode button bit
    ld     R1,#001h              ; Setup for delay of 0.16msec
    call   WaitNmsec             ; Wait
    tm     PTBIN,#M_Switch        ; Mode button = 1?
    jr     z,Wait4ModeRel        ; If not, jump back to wait
    tm     IREQ,#IRQ_ModeBit     ; else, look for bouncing
    jr     nz,Wait4ModeRel       ; If bounce detected, wait
    pop    RP                    ; else, restore reg pointer
    ret

*****
* Function Name:                SelPress
*
* Returns:                     Nothing
* Entry values:                None
* Description:                 This routine waits for select button to resettle
*                               back to "1" after being pressed.
* Notes:                      The display is blanked while the button is pressed
*
*****
SelPress:
    push   RP                    ; Save register pointer
    srp    #GenRP                ; Move to general space
    or     PTAOUT,#ShiftLED      ; Blank the shift indicator
    or     PTBOUT,#Digits        ; Blank the display digits
Wait4SelRel:

```



dR1,#001h; Setup for delay of 0.16msec

```

call WaitNmsec           ; Wait
tm  PTBIN,#S_Switch     ; Select button = 1?
jr  z,Wait4SelRel       ; If not, jump back to wait
ld  R1,#001h            ; else, wait 0.16msec & exit
call WaitNmsec           ; Wait
pop  RP                  ; Restore reg pointer
ret

```

```

*****
*  Function Name:      ECDisplay
*
*  Returns:           Nothing
*  Entry values:      None
*  Description:       This routine configures the display for the engine
*                    configuration code.
*  Notes:             "E" is displayed on digit1, the number of engine
*                    cylinders is blinked on digit2.
*

```

```

*****
ECDisplay:
  push  RP              ; Save the register pointer
  srp   #GenRP          ; Move to general space
  ld    KRpmDsply,#10000110b ; Init digit 1 with "E"
  ld    R1,EngineCode   ; Obtain the engine code
  call  BCDto7Seg       ; Convert to display pattern
  ld    HRpmDsply,R1    ; Overlay the display pattern
  pop   RP              ; restore the register pointer
  ret

```

```

*****
*  Function Name:      RpmDsply
*
*  Returns:           Nothing
*  Entry values:      None
*  Description:       This routine configures the display for the shift
*                    RPM value.
*  Notes:
*

```

```

*****
RpmDsply:
  push  RP              ; Save the register pointer
  srp   #GenRP          ; Move to general space
  ld    R1,ShiftRpm     ; Obtain Shift Rpm value
  call  BCDto7Seg       ; Convert to display pattern
  ld    HRpmDsply,R1    ; Store H RPM pattern
  ld    R1,ShiftRpm     ; Reload Shift Rpm value
  swap  R1              ; Place K RPM in low nibble
  call  BCDto7Seg       ; Convert to display pattern
  ld    KRpmDsply,R1    ; Store K RPM pattern
  pop   RP              ; restore the register pointer
  ret

```

```

*****

```



```
* Function Name:      BCDto7Seg
*
* Returns:           7segment display pattern for BCD value.
* Entry values:     Lower nibble R1 loaded with BCD RPM value.
* Description:      This routine converts BCD to 7segment LED pattern.
* Notes:            R0 is modified.
*
```

```
BCDto7Seg:
    and    R1,#00Fh          ; Mask off upper nibble
    ld     R0,#>DigPatterns  ; Obtain the pattern table address
    add    R1,#<DigPatterns  ; Add address to BCD value
    adc    R0,#000h         ; Add the overflow
    ldc    R1,@RR0          ; Obtain the pattern
    ret
```

```
* Function Name:      Div24x16
*
* Returns:           8-bit quotient R6, 16-bit remainder R4-R5
*                   carry flag set on overflow. R0 is also modified
*                   by the routine.
* Entry values:     24-bit dividend R4-R6, 16-bit divisor R2-R3
*                   The registers are assumed to be preloaded.
* Description:      This routine performs the division of a 24-bit
*                   unsigned value by a 16-bit unsigned value,
*                   resulting in an 8-bit unsigned quotient. The
*                   algorithm repetitively shifts the dividend left.
*                   If the high order bit shifted out is a one or if
*                   the resulting high-order dividend word is greater
*                   than or equal to the divisor, the divisor is
*                   subtracted from the high byte of the dividend. As
*                   the low-order bits of the dividend are vacated by
*                   the shift left, the resulting partial-quotient
*                   bits are rotated in.
* Notes:
*
```

```
; 24-bit Divide Routine definitions
```

```
;
Count          .equ  R0
Divsr_msw      .equ  R2
Divsr_lsw      .equ  R3
Divnd_msw      .equ  R4
Divnd_mid      .equ  R5
Divnd_lsw      .equ  R6
```

```
Div24x16:
    ld     Count,#008h      ; Initialize loop counter
;
; *** Check if result will fit in 8bits
;
    cp     Divsr_msw,Divnd_msw ; Compare msb words
    jr     ugt,D24_Loop       ; If divsr > then OK
    jr     ult,D24_Ovrflo     ; If divsr < then overflow
    cp     Divsr_lsw,Divnd_mid ; If equal, compare lsb words
    jr     ugt,D24_Loop       ; If divsr > then OK
```



```

D24_Ovrflo:
    scf                ; Overflow - set carry to 1
    ret
;
; *** Result will fit, perform division algorithm
;
D24_Loop:
    rlc    Divnd_lsw    ; Dividend/2
    rlc    Divnd_mid    ;
    rlc    Divnd_msw    ;
    jr     c,D24_Subtr
    cp    Divsr_msw,Divnd_msw    ; Compare msb words
    jr     ugt,D24_Next    ;
    jr     ult,D24_Subtr    ; Divisor < Dividend
    cp    Divsr_lsw,Divnd_mid    ; Compare lsb words
    jr     ugt,D24_Next    ;

D24_Subtr:
    sub    Divnd_mid,Divsr_lsw    ; Subtract lsb words
    sbc    Divnd_msw,Divsr_msw    ; Subtract msb words

    scf                ; To be shifted into result

D24_Next:
    djnz   Count,D24_Loop    ; Update count
    ; Done!
    rlc    Divnd_lsw    ;
    ret

```

```

*****
*   Function Name:      Hex2Dec
*
*   Returns:           BCD value in R6
*   Entry values:      RP=GenRP, R6=hex value to be converted
*   Description:       This routine converts a single byte hex value
*                       into its BCD equivalent.
*   Notes:             Assumes the value is not greater than 99dec
*                       Contents of R1,R2,R3 destroyed
*****

```

```

Hex2Dec:
;
    ld     R1,#007h    ; Initilize loop counter
    ld     R2,#001h    ; Initilize incremental value
    clr    R3          ; Clear the result register
    rcf    R3          ; Clear the carry flag

ConvertLoop:
    rrc    R6          ; Obtain next bit
    jr     nc,NoAdd    ; Jump if no add is necessary
    ; otherwise, add to decimal
    add    R3,R2      ; Add the decimal increment
    da     R3          ; Decimal adjust

NoAdd:
    add    R2,R2      ; Double the incremental value
    da     R2          ; Create decimal equivalent
    djnz   R1,ConvertLoop    ; Continue to loop for 7 bits

    ld     R6,R3      ; Return decimal output to R6
    ret

```



```

*****
*   IRQ0 (T0 overflow) Interrupt Service
*
*   This routine detects an overflow of the T0 8-bit counter.  This
*   counter is the least significant byte of the 24-bit real time
*   counter used for display timing.
*
*   Procedure: T0 reaching count of 0 causes IRQ0 service.
*   The 2 upper bytes of the real time clock are decremented.
*
*****
RTClsb0:
    push    RP                ; Save register pointer
    srp     #AppRP            ; Move to Application space
    decw   RR14              ; Decrement upper RTC bytes
    pop     RP                ; Restore original reg pointer
    iret

*****
*   IRQ4 (PB4 rising edge) Interrupt Service
*
*   This routine performs the ignition timing function.  A 16-bit
*   value representing the number of microseconds between edges of the
*   ignition pulse is determined.
*
*   Procedure: Rising ignition pulse edge causes IRQ4 service.
*   If a timing cycle is in progress, it is terminated and the elapsed
*   time is recorded.  If a timing cycle is not in progress, one is
*   initiated.  Note that the IRQ5 interrupt is enabled during the
*   timing cycle to detect timer rollover resulting from an absence of
*   ignition pulses or RPMs far below the design spec.
*
*****
IgnPulse:
    tm     AppState,#IgnActBit    ; Check for active timing cycle
    jr     z,CycleStart          ; If not active, jump to start one

; End of ignition timing cycle detected - calculate the elapsed time
    and    TCTLLO,#~Ign_Start    ; Stop the Ignition timer
    tm     AppState,#IgnUpdate   ; Check completion of last update
    jr     nz,CycleEndExit      ; Jump to abort the timing update
    push   RP                    ; Save register pointer
    srp    #GenRP                ; Move to general workspace
    ld     R14,#T3_Value         ; Load Timer initial value
    ld     R15,#T2_Value         ;
    sub    R15,T2VAL             ; Calculate time difference
    sbc    R14,T3VAL            ;
    ld     @IgnUpdPntr,R14       ; Store the ignition time
    inc    IgnUpdPntr           ; Move pointer to 1s byte
    ld     @IgnUpdPntr,R15       ;
    inc    IgnUpdPntr           ; Move pointer to next pair
    cp     IgnUpdPntr,#AvgRP+16 ; Check for pointer overflow
    jp     ult,IgnPntrOK        ; Jump if OK
    ld     IgnUpdPntr,#AvgRP     ; else, reset the pointer
IgnPntrOK:

```



```

        or    AppState,#IgnUpdate      ; Activate time available flag
        pop   RP                       ; Restore original reg pointer
CycleEndExit:
        and   AppState,#~IgnActBit    ; Clear the timing active bit
        and   AppState,#~RPMOut      ; Clear the ignition overflow bit
        ld    T2VAL,#T2_Value         ; Prepare T23 for next cycle
        ld    T3VAL,#T3_Value         ;
        and   AppState,#~IgnDead     ; Clear the dead ignition flag
        ld    TimeOut,RTCMSbyte       ; Create the timeout value
        sub   TimeOut,#080h           ;
        ired

CycleStart:
; Start of ignition timing cycle detected - start ignition timing
        or    TCTLLO,#Ign_Start       ; Start the Ignition timer
        or    AppState,#IgnActBit    ; Set ignition timing active flag
        ired

*****
*      IRQ5 (T2 overflow) Interrupt Service
*
*      This routine detects an overflow of the 16-bit ignition time count
*      which represents an RPM that is below the operational lower limit.
*
*      Procedure: T23 reaching count of 0 causes IRQ5 service.
*      The current timing cycle is aborted and the error condition is
*      denoted by activating the RPMOut flag in the AppState register.
*****
Overflow:
        and   TCTLLO,#~Ign_Start      ; Stop the Ignition timer
        and   IREQ,#~IRQ_OvrFlo      ; Clear the interrupt bit
        and   AppState,#~IgnActBit    ; Clear the timing active bit
        or    AppState,#RPMOut       ; Set the ignition overflow bit
        ld    T2VAL,#T2_Value         ; Prepare T23 for next cycle
        ld    T3VAL,#T3_Value         ;
        ired

*****
*      Unused Interrupt Service
*****

; Empty IRQ's defined earlier so that the processor will have a 16 bit
; address in memory to jump to and return from in the case of a stray
; or glitch interrupt.

PB4f:
TMR1:
ModeButn:
        ired                           ; Dummy handler for unused interrupts

*****
*      System Constants
*****
;

```



```
RpmCnvCnst:
; Conversion Constant=
; [(1000000*60/#ignition fires per revolution)]/Scale Factor
; Scale Factor=100 since we're displaying RPM/100: ie X.X Krpm
; for 4 cylinder: [(1000000*60/2)/100]= 300000 (0493E0h)
; for 6 cylinder: [(1000000*60/3)/100]= 200000 (030D40h)
; for 8 cylinder: [(1000000*60/4)/100]= 150000 (0249F0h)
;
DB 004h ; 4 cylinder conversion MS byte
DB 093h ; 4 cylinder conversion MD byte
DB 0E0h ; 4 cylinder conversion LS byte
DB 000h ; dummy
DB 003h ; 6 cylinder conversion MS byte
DB 00Dh ; 6 cylinder conversion MD byte
DB 040h ; 6 cylinder conversion LS byte
DB 000h ; dummy
DB 002h ; 8 cylinder conversion MS byte
DB 049h ; 8 cylinder conversion MD byte
DB 0F0h ; 8 cylinder conversion LS byte
DigPatterns:
DB 11000000b ; "0"
DB 11111001b ; "1"
DB 10100100b ; "2"
DB 10110000b ; "3"
DB 10011001b ; "4"
DB 10010010b ; "5"
DB 10000010b ; "6"
DB 11111000b ; "7"
DB 10000000b ; "8"
DB 10010000b ; "9"
*****
; End of main program.
end
```

Test Procedure

Equipment Used

Testing the application requires the following items:

- Target application board built according to the schematic in the Appendix
- 12V, 1A bench supply (for application power)
- Windows 95/98/NT-based PC with ZDS 2.11 or higher installed
- Z8ICE001ZEM (Z8Plus Emulator)
- 8V @ 0.8 A power supply (for emulator power)
- Signal generator
- Oscilloscope



Note that the signal generator must be capable of producing a 12V rectangular waveform with a duty cycle of approximately 30%.

General Test Setup

Exercise the application by either burning an OTP (stand-alone) or running the application from the emulator.

To execute the application from the emulator, follow the instructions for assembling the application code as detailed in the previous section. If the emulator runs the application, make certain that the JP1 jumper is disconnected before connecting the target cable to the application.

Caution: Failure to disconnect the JP1 jumper may damage the emulator.

Configure the remaining emulator jumpers as required based on the instructions in the emulator user's manual and then connect the emulator target cable to the application.

With the signal generator disconnected from the application, turn it on and configure it for the following output:

- Rectangular waveform, 20% to 50% duty cycle
- 12V peak-to-peak amplitude with 0V DC offset
- Frequency 100 Hz

Verify these settings using the oscilloscope. With the power supply turned off, connect the +12V and GND signals of the application accordingly. Turn on the +12V power supply and verify $5V \pm 0.5V$ at pin 14 of U1 using the oscilloscope. If the voltage is outside this range, check the power supply settings and connections as well as the application circuit to correct the problem. Leave the signal generator disconnected. If using the emulator, start the emulation software and run the application code.

Test Procedure

After applying power to the unit and starting operation, notice the LED test being executed. In this test, each digit of the display is turned on for about $\frac{1}{2}$ second and then the shift indicator is turned on for about $\frac{1}{2}$ second. If the shift indicator or any of the display segments are not illuminated, the unit is defective.

After the LED test, the unit enters CONFIGURATION mode. CONFIGURATION mode is indicated by *E8* displayed with the 8 flashing. This flashing indicates default configuration on power-up, which represents the 8-cylinder engine option. Press the Select button one time to change the display to *E4* for the 4-cylinder option. Press Select again to change the display to *E6* for the 6-cylinder option.



Another press displays the original *E8* configuration. Press the Mode button one time while *E8* is displayed to select SHIFT RPM.

Now, the unit is displaying 60, with the 6 flashing. This flashing indicates the shift rpm default value of 6000 rpm. The 6 is flashing to indicate that the thousands digit is being configured. Press the Select button to rotate the thousands digit through the values from 1 to 7. Press the Select button until 50 is displayed and then press the Mode button one time. The thousands digit is configured to 5 and the hundreds digit is flashing indicating it is ready to be configured. Press the Select button now to rotate the hundreds digit through the values from 0 to 9. Press the Select button until 55 is displayed and then press the Mode button one time. The configuration process is complete and the unit begins operation using the new configuration (8-cylinder engine, 5500-rpm shift point). The display is blanked until either an rpm calculation is completed or the ignition is declared dead.

Because the signal generator is not connected, after approximately 16 seconds of blank display, the value “- -” is displayed indicating that no ignition pulses were received. Connect the signal generator’s ground, or common output, to the application’s GND signal and the signal generator’s waveform output to the application’s IGNITION signal. The display indicates the current rpm based on the signal generator frequency. Use the oscilloscope to measure the waveform at pin 4 of U1. The waveform’s frequency should be 100Hz with an amplitude of approximately 4V, and if not, adjust the waveform accordingly. The result is 15 being displayed, indicating 1500 rpm as the current engine speed. Changing the frequency of the waveform results in a corresponding change in rpm. The relationship for this configuration is:

$$\text{RPM} = 15 \times \text{signal generator frequency in Hz}$$

(for example, 100Hz = 1500 rpm)

Slowly increase the signal generator’s frequency and notice the corresponding change in rpm being displayed. Increase the frequency just enough to cause the shift indicator to remain illuminated. Measure the frequency of this waveform using the oscilloscope. The waveform is equal to or slightly greater than 366.7Hz. This value corresponds to the 5500 rpm shift point configured previously. Continue increasing the signal generator’s frequency until the shift indicator begins to flash constantly. Measure the waveform’s frequency using the oscilloscope. The waveform must be equal to or slightly greater than 400.0Hz. This value corresponds to 500 rpm over the shift point, which is the point at which the indicator is flashed as a warning to the driver. Continue to increase the frequency slowly until “- -” displays constantly. Use the oscilloscope to measure the frequency. The waveform must be equal to or slightly greater than 600.0Hz. This frequency corresponds to the unit’s upper operational limit of 9000 rpm. Decreasing the frequency from this point allows the unit to resume operation. Decrease the frequency until “- -” is constantly displayed. Use the oscilloscope to



measure the frequency. The waveform must be equal to or slightly less than 33.3 Hz. This waveform corresponds to the unit's lower operational limit of 500 rpm. Slightly increasing the frequency allows the unit to resume operation.

Reset the signal generator to a frequency of 100Hz and press the Mode button. Pressing the Mode button causes the unit to re-enter CONFIGURATION mode. CONFIGURATION mode is confirmed by a display of *E8* with the *8* flashing. Press the Select button to change the configuration to *E4* and press the Mode button three times to begin operation. The unit is now configured to operate with a 4-cylinder engine and a 5500 rpm shift point.

The unit should be displaying *30*, indicating a current engine speed of 3000 rpm. The relationship for this configuration is:

$$\text{RPM} = 30 \times \text{signal generator frequency in Hz}$$

(for example, 100Hz = 3000 rpm)

Slowly increase the signal generator's frequency and notice the corresponding change in rpm being displayed. Increase the frequency just enough to cause the shift indicator to remain illuminated. Measure the frequency of this waveform using the oscilloscope. The waveform should be equal to or slightly greater than 183.3Hz. This frequency corresponds to the 5500 rpm shift point configured previously. Continue increasing the generator's frequency until the shift indicator flashes constantly. Again, use the oscilloscope to measure the waveform's frequency. The waveform must be equal to or slightly greater than 200.0Hz. This frequency corresponds to 500 rpm over the shift point.

Reset the signal generator to 100Hz and press the Mode button to cause the unit to reenter CONFIGURATION mode. *E4* now displays a flashing *4*, which is the current engine code. Press the Select button to change the value to *E6*, then press the Mode button three times to begin operation. The unit is now configured to operate with a 6-cylinder engine and a 5500-rpm shift point.

The unit displays *20*, indicating a current engine speed of 2000 rpm. The relationship for this configuration is:

$$\text{RPM} = 20 \times \text{signal generator frequency in Hz}$$

(for example, 100Hz = 2000 rpm)

As with the other configurations, change the frequency settings and verify the unit's operation. If required, reenter CONFIGURATION mode to change the shift point to other values for verification.

References

1. *Z8 Microcontroller User's Manual*, UM95Z80010, ZiLOG, Inc., 1995.
2. *Z8Plus User's Manual*, UM001000-Z8X0199, ZiLOG, Inc., 1999.

3. *Z8PE003 Preliminary Product Specification*, DS007500-Z8X0399, ZiLOG, Inc., 1999.
4. *ZiLOG Macro Cross Assembler User's Manual*, UM003601, ZiLOG, Inc., 1997.

Appendix

Figure 5. Digital Shift Indicator Schematic

