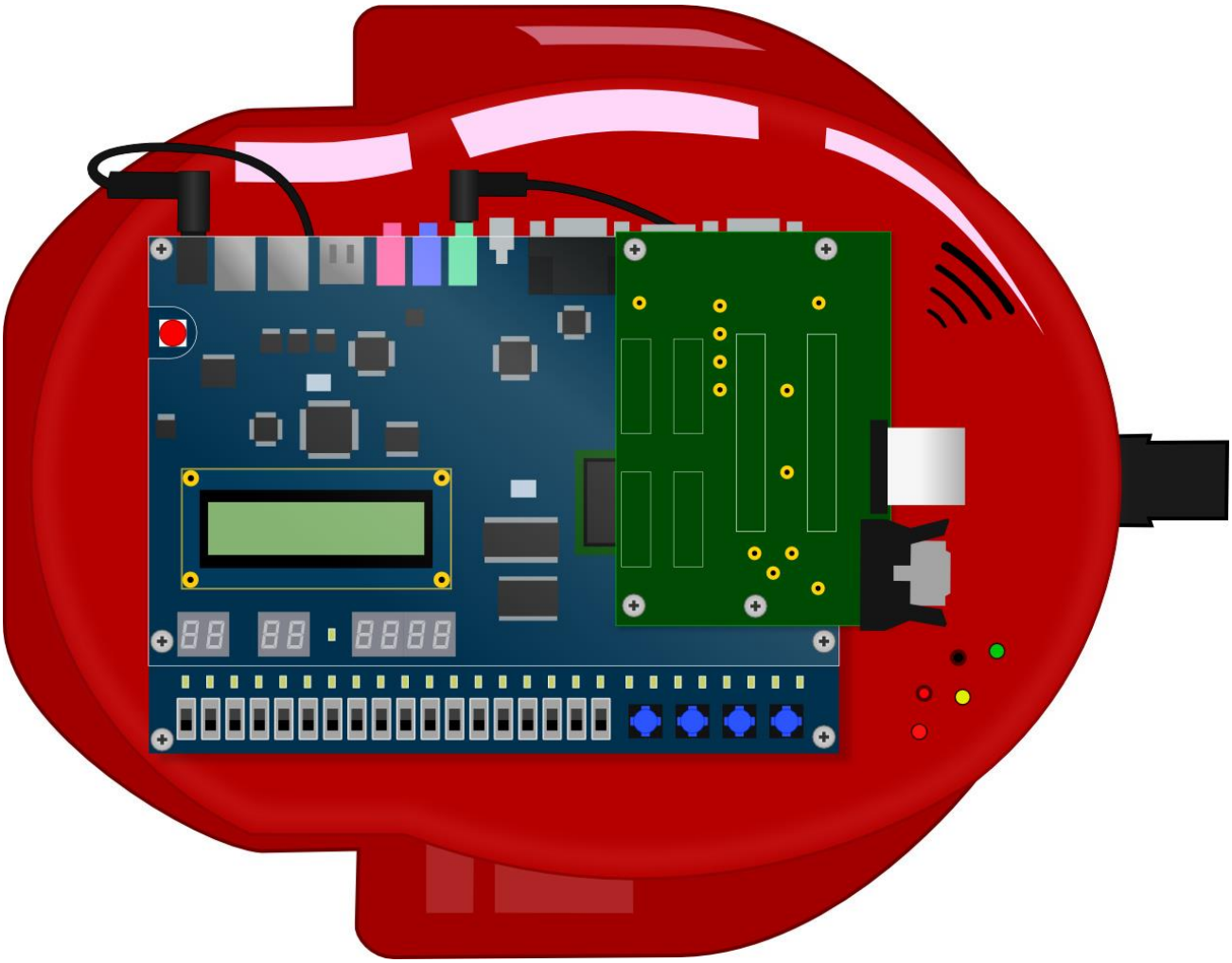
DE2Bot User's Manual
Georgia Institute of Technology ECE2031



Kevin Johnson

Updated 10 March 2016

Introduction

This document is intended for the end-user of the DE2Bot: students in ECE2031. It provides an overview of the hardware, a walkthrough of the built-in self-test program, and a programming guide for use with the version of SCOMP provided during the final design project.

Table of Contents

1–DE2Bot Hardware Overview

The DE2Bot is comprised of the commercially-available AmigoBot with its electronics removed and replaced with custom hardware and an Altera DE2 FPGA development board. This configuration allows complete control of the robot hardware using custom digital circuits created within the FPGA.

Locations of important features are shown below in Figure 1.

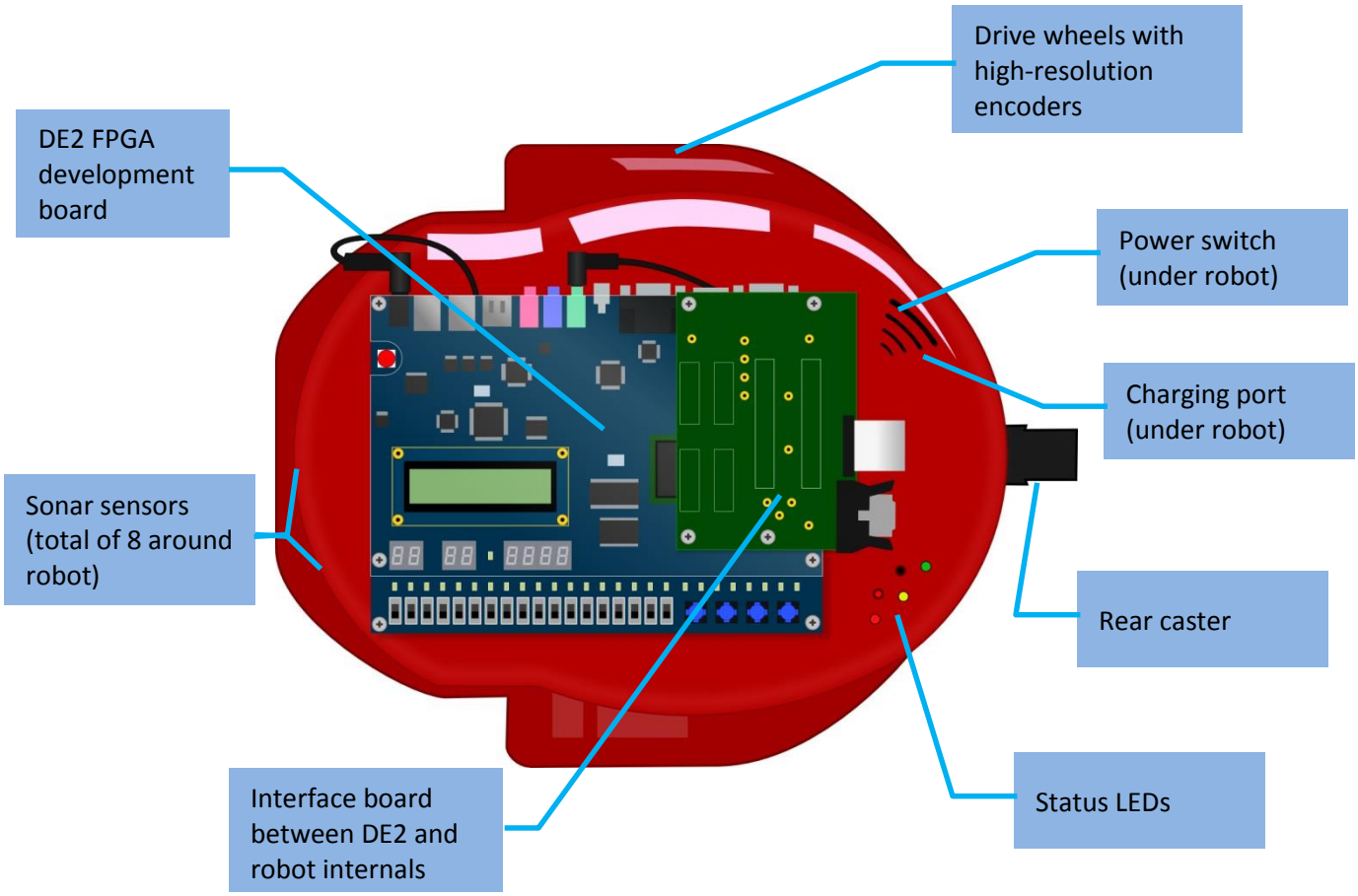


Figure 1. Locations of important DE2Bot features.

1.1 Main Feature Descriptions

DE2 Development Board

The Altera DE2 board provides access to a Cyclone II FPGA as well as various I/O, such as:

- 18 slide switches
- four push buttons
- 27 LEDs
- a 16x2-character LCD
- eight 7-segment displays
- audio in and out with ADC/DAC
- VGA video output
- an RS-232 serial port
- SD card slot.

The DE2 on the DE2Bot connects to the robot's internal circuitry through its GPIO ports, allowing direct digital control of all robot functions.

Note that the DE2's power button (red button at top left of board) should not be used. Leave the DE2 ON and use the robot's main power switch to turn the DE2Bot on and off.

Sonar Distance Sensors

The DE2Bot is equipped with eight sonar transducers that can be used to measure distances to objects. The sensors are arranged around the robot as shown in Figure 2 and numbered clockwise starting with Sonar 0, which is facing left (from the robots forward orientation; downwards in the figure).

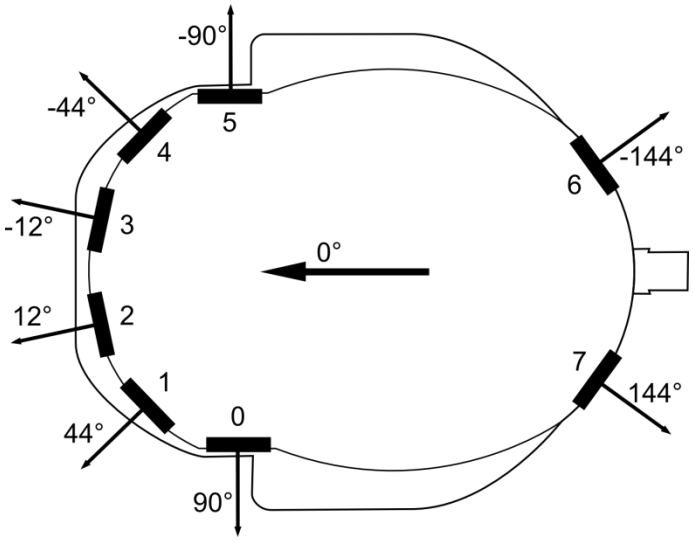


Figure 2. Sonar sensor numbering, positions, and directions.

The sonar sensors can measure distances from 15cm up to 5m or more depending on the properties of the object. The default resolution of the sensors is 1mm, though accuracy may vary ±1cm.

Each sonar sensor can be enabled independently. The sonar firing rate is 25Hz divided between all enabled sonars: if only one sonar is enabled, it is refreshed at 25Hz; if all eight sonars are enabled, the refresh rate of any particular sensor is just over 3Hz (25Hz/8).

Wheels and Encoders

The DE2Bot has two drive wheels, one on each side, allowing it to use differential steering to move around smooth or dense-carpeted surfaces. A rear caster wheel helps to support the robot without interfering with movement. The drive wheels are powered by DC motors through a reduction gearbox.

Each motor is equipped with a quadrature encoder which can be used to keep track of relative angular wheel position and thereby calculate velocity and acceleration. The addition of specialized hardware within the FPGA enables dead reckoning estimation of robot position.

- The control circuitry for the motors includes a watchdog timer that disables the motors if no 'alive' signal is received for approximately one second. In the default Quartus project for ECE2031, an additional safety mechanism blocks the 'alive' signal until SW17 has been toggled (both up and down) after power-up or reset. This ensures that the robot does not move immediately after being programmed, and guarantees that the robot will stop when PB0 is pressed.

Battery and Charge Port

The DE2Bot contains a 5.5Ah rechargeable LiPo battery, enabling several hours of 'idle' use or approximately an hour of continuous running between charges. A charge port underneath the robot provides easy attachment of an external charger.

- Note that when the robot's main power switch is ON, the charging port is disconnected from the battery. **In order to charge, the power switch must be in the OFF position.**

Care should be taken to never discharge the battery below 14V. Doing so will reduce the life of the battery and may cause permanent damage.

Wireless Serial Connection

An internally-mounted XBee wireless communication module enables remote communication with a central PC through the DE2's RS-232 port. By default, it transparently emulates a direct serial connection.

For the serial communication to be useful, some "back-end" software must be running on the host PC, which will vary from semester to semester. Check with lab administrators for current functionality.

- The UART is internally rate-limited to ensure consistent results when many robots are communicating at the same time. See the UART details in section 3.4 for more information.

2–Self-test Operation

On power-up, a self-test program is automatically loaded from non-volatile memory. This program enables the user to quickly test for proper operation of the DE2Bot hardware.

Note: the self-test program uses PB0 as RESET. Press PB0 to restart the program at any time.

2.1 Power-up Tests

As soon as the DE2Bot is turned on (or when the self-test is restarted with PB0):

- The DE2Bot beeps briefly
- An approximate battery percentage remaining is displayed (in decimal) on the HEX5 and HEX4 seven-segment displays
- The battery percentage is displayed as a bar graph on red LEDs 0-14
    - A fully-charged battery will light all LEDs 0-14. A dead battery will light only LED 0.
- Green LEDs 0, 1, and 2 light, mirroring the inactive state of pushbuttons 1, 2, and 3 respectively
    - Pressing a PB will turn off the respective green LED.
- The LCD displays a menu prompting the user to choose “Self Test” or “Troubleshoot”

Power-up Errors

If the battery is too low to safely operate the DE2Bot, the user is warned with beeps, flashes, and a written warning on the LCD. In this case, turn off the DE2Bot immediately and plug it in to a charger.

If nothing happens when the DE2Bot is turned on, there is likely a problem with the battery or power circuitry. Turn the DE2Bot switch to the OFF position and notify an administrator.

2.2 Automated Self-test

At the LCD prompt after power-up, pressing PB1 will begin a mostly-automated self-test routine.

The LCD will provide prompts that allow the user to execute the automated self-test without this document, but detailed information is provided here for first-time users or in the case of errors.

1) Battery Check

The current battery voltage is displayed in decimal on the LCD screen. Battery voltage should be 14-17V for proper robot operation.

2) Sonar Test

Each of the eight sonar sensors is tested, starting with Sonar0 (left-facing sonar) and proceeding clockwise. Each sonar is polled until either a valid reading is obtained, or 5 seconds elapse.

If the test pauses on a particular sonar, move an object (such as your hand) in front of that sonar so that a reading can be obtained. The current sonar is indicated on the red LEDs, or you can listen for the characteristic ‘clicking’ sound.

Once all sonar sensors are tested and working, the message “All sonars are working” is displayed on the LCD, and the program automatically proceeds to the next test.

Sonar Errors

If a sonar does not return a valid reading within 5 seconds, it is assumed to be defective. At the end of the sonar test, the green LEDs display which sonar(s) are not working. Note the number(s), and return the DE2Bot to an administrator.

3) Encoder Test

Warning: The test immediately following this test may cause the robot to move under its own power. Ensure that the DE2Bot is either on the floor in a clear area, or its wheels are raised off of the supporting surface. Continuing this test with the robot on a table can cause it to fall when the following test begins.

Once the sonar test is complete, the LCD will display "Rotate left wheel 30+ degrees". At this prompt, manually rotate the left wheel in either direction until the LCD changes to "Rotate right wheel 30+ degrees", then repeat the rotation with the right wheel.

During this test, the current encoder position value is displayed on HEX3-0.

Encoder Errors

If no wheel motion is detected within 10 seconds, the test fails and an error is displayed on the LCD. Inform an administrator.

4) Motor Test

Immediately after the encoder test completes, the motor test begins. If the safety switch (SW17) has not been toggled since reset, the LCD will prompt "Toggle SW17", at which point you should raise and lower SW17. Once the safety is disabled, the left wheel will begin turning forwards and the LCD will display "Left wheel turning? 2-N/1-Y". If the wheel is turning, press PB1. If not, press PB2. The test will then repeat with the right wheel.

Motor Errors

If either wheel does not turn when expected:

If previous tests indicated that the battery was below 30%, the battery might be too low to operate the motors. Turn the DE2Bot off and plug it in to a charger.

If the battery is well charged, there is likely a problem with the motors or supporting electronics. Notify an administrator.

Self-test Finish

Once the motor test is complete, the LCD will display "Self Test Finish PB1 – Main Menu". If any errors occurred during the self-test, a red LED will be lit as follows:

- LED0-7 indicate sonar 0-7 errors
- LED8 and LED9 indicate left and right encoder errors
- LED9 and LED10 indicate left and right motor errors

Press PB1 to return to the main menu.

2.3 Manual Tests

From the main menu, pressing PB2 will enter manual-test (troubleshooting) mode, where specific hardware can be tested more thoroughly.

Entering Tests

Once in the troubleshooting main menu, raising a switch and pressing PB1 will enter the corresponding test - see Table 1 below. If multiple switches are up, the lowest-indexed test is selected. While in a test, pressing PB2 and PB3 together will return to the troubleshooting test selection mode. Use PB0 to return to the main menu.

TABLE 1
MANUAL TEST SELECTION

Switch Raised	Hardware Tested
SW0	Battery
SW1	Speaker
SW2	Switches and Pushbuttons
SW3	Sonars
SW4	LEDs, 7-segment displays, LCD
SW5	Wheel encoders
SW6	Motors
SW7	UART

Battery Test (SW0)

The battery voltage is continuously read and displayed on the LCD (in decimal) and 7-segment display (in hex).

Speaker (SW1)

The robot emits a stream of beeps with 0.15s on and 0.5s off. The LEDs light when the beep should be on.

Switches and Pushbuttons (SW2)

Switches 0-16 are reflected on red LEDs 0-16. The pushbuttons are reflected on the green LEDs 0-2.

Sonars (SW3)

Switches 0-7 will individually enable sonars 0-7. The value returned by the sonar is displayed on the 7-segment display in hexadecimal. If more than one sonar is enabled, only the lowest-indexed one's value is displayed.

LEDs, 7-segment displays, LCD (SW4)

All LEDs flash at 1Hz. The 7-segment displays alternate between 0x1111 and 0xEEEE (exercising all segments). The LCD alternates between blank and black.

Wheel Encoders (SW5)

SW0 up/down selects between the left and right wheels. The selected wheel's current position value is displayed on HEX3-0 and the immediate velocity on HEX7-4.

Motors (SW6)

Hold PB1 to power the right motor and PB2 to power the left motor. Raise SW0 to reverse the right motor, and SW1 to reverse the left motor.

UART (SW7)

The UART begins sending one byte every second, starting at 0x01 and incrementing each time. The transmitted value is displayed on SSEG1. If a byte is received through the UART, it is displayed on SSEG2 and the robot beeps.

3–Programming Guide

At the beginning of the final project in ECE2031, students are provided with a Quartus project containing the SCOMP processor and many IO devices which interface with the DE2 and DE2Bot hardware. Each of these devices is assigned an IO address in the SCOMP system as detailed in the Quick Reference section below.

3.1 Changes to SCOMP

SCOMP and its Quartus project have been modified from lab 8 in the following ways:

- Central IO_DECODER device replaces AND/NAND decoders for each peripheral.
- IO_CYCLE and IO_WRITE operation slightly modified to improve stability.
- SCOMP program memory doubled to 2048 words.
    - A side effect of this is that the operand is now 11 bits, so immediate instructions (like ADDI) can now use 11-bit numbers (±1023).
- SCOMP subroutine stack depth increased to 10.
- SHIFT instruction changed from "logical" to "arithmetic" to better support mathematical operations.
- All SCOMP instructions in Table 7.1 of the lab manual implemented.
- Additional SCOMP instructions added:
    - LOADI - load immediate (immediate value is limited to 11-bit 2's complement)
    - SEI, CLI, and RETI - used for new interrupt system, as described below

3.2 SCOMP Peripheral Interrupt System

Interrupts are disabled by default, and will not interfere with normal operation if left unused. This section is safe to ignore if you do not plan on using interrupts. This section assumes some prior knowledge of interrupts.

Four external pins, PCINT[3..0], have been added to SCOMP, each of which can cause an interrupt, and each of which has a dedicated interrupt vector. Table 2 shows the default source for each of the interrupt pins. For more information about each source, refer to that device's description in section 3.4.

TABLE 2

INTERRUPT DEVICES, PINS, AND VECTORS

PCINT Pin	Connected Device	Description	Interrupt Vector
PCINT[0]	Configurable Timer	Periodic interrupts at intervals of 10ms-5min	0x001
PCINT[1]	Sonar Alarms	Sonar detected an object closer than alarm distance	0x002
PCINT[2]	UART Data Ready	Data has been received from the UART	0x003
PCINT[3]	Stall Warning	Indicates that one or both motors have been stalled.	0x004

Enabling and Disabling Interrupts

At reset, all interrupt pins are disabled and cannot cause interrupts. Each pin can be individually enabled using the SEI (SEt Interrupts) instruction, which takes a 4-bit mask specifying which interrupts to enable: a '1' in the mask will enable that interrupt, while a '0' will have no effect (even if that interrupt is already enabled). For example, SEI &B1111 will enable all interrupts. SEI &B1001 will enable interrupts 0 and 3, while leaving 1 and 2 unmodified. SEI &B0000 has no effect. CLI (CLear Interrupts) is similarly used to disable interrupts: CLI &B1111 will disable all interrupts, while CLI &B0000 will have no effect.

Interrupt Service Routines

When an interrupt event (a rising edge on an interrupt pin) occurs, SCOMP execution is redirected to the memory locations described in Table 2. These locations must contain either a RETI instruction (if that interrupt is not being used) or a JUMP instruction to the appropriate interrupt service routine (ISR). A template ASM file with examples can be found in Appendix B.

3.3 IO Device Quick Reference

TABLE 3
SCOMP QUARTUS PROJECT I/O DEVICE DESCRIPTIONS

Name	IO Address	IN/OUT	Description
SWITCHES	0x00	IN	Read DE2 switches SW15-S0.
LEDS	0x01	OUT	Write to DE2 LEDs LEDR15-LEDR0.
TIMER	0x02	IN/OUT	Read 10Hz timer count. Write anything to reset to 0.
XIO*	0x03	IN	Read PB3-PB1, SW16, SAFETY signal, and some GPIO.
SSEG1	0x04	OUT	Write to left 4-digit seven-segment display.
SSEG2	0x05	OUT	Write to right 4-digit seven-segment display.
LCD	0x06	OUT	Write to LCD (16-bit hexadecimal).
XLEDS	0x07	OUT	Write to DE2 LEDs LEDG7-LEDG0 and LEDR17/LEDR16
BEEP	0x0A	OUT	Write 1-7 for beep frequency (250Hz*N). Write 0 to turn off beep.
CTIMER	0x0C	OUT	Configurable timer for periodic interrupts (see interrupt section)
LPOS*	0x80	IN	Read the current position of the left wheel encoder; 1.05mm/tick.
LVEL*	0x82	IN	Read the current velocity of the left wheel; 1.05mm/s.
LVELCMD*	0x83	OUT	Write the desired velocity of the left wheel; 1.05mm/s.
RPOS*	0x88	IN	Read the current position of the right wheel encoder; 1.05mm/tick.
RVEL*	0x8A	IN	Read the current velocity of the right wheel; 1.05mm/s.
RVELCMD*	0x8B	OUT	Write the desired velocity of the right wheel; 1.05mm/s.
I2C_CMD*	0x90	OUT	Write configuration information to the I2C controller.
I2C_DATA*	0x91	IN/OUT	Read or write data from/to the I2C controller.
I2C_RDY*	0x92	IN/OUT	Begin I2C transaction or check transaction status.
UART_DAT	0x98	IN/OUT	Read or write one byte (low-byte of AC) from/to the UART.
UART_RDY	0x99	IN	Read the status of the UART controller.
DIST0 - DIST7*	0xA8 - 0xAF	IN	Read the measured distance from Sonar0 - Sonar7.
SONALARM*	0xB0	IN/OUT	Write the alarm distance; read the alarm register.
SONARINT*	0xB1	OUT	Write bits 0-7 to enable interrupts from sonar0 - sonar7.
SONAREN*	0xB2	OUT	Write bits 0-7 to enable sonar0 - sonar7.
XPOS*	0xC0	IN	Read dead-reckoning X position estimation; 1.05mm/count.
YPOS*	0xC1	IN	Read dead-reckoning Y position estimation; 1.05mm/count.
THETA*	0xC2	IN	Read dead-reckoning angle estimation in degrees; always [0,359].
RESETPOS	0xC3	OUT	Reset dead-reckoning odometer: X,Y,θ = 0,0,0.

* additional details in following sections

3.4 Detailed Description of Select Devices

XIO

The value read from XIO contains the following signals:

- XIO[15..5] : GPIO pins on the DE2 header
- XIO[4] : SAFETY signal, which indicates whether or not SW17 has been toggled
- XIO[3] : SW16
- XIO[2..0] : Pushbuttons PB3 - PB1 (PB0 is global reset and cannot be read)

Note that the pushbuttons are active-low: a pressed pushbutton will appear as a '0' in XIO.

Sonar Sensors

Each sonar sensor can be independently enabled through the SONAREN register. Bits 0-7 of this register correspond to sonars 0-7; e.g. writing 0b11000000 will enable only Sonar6 and Sonar7, and writing 0b11111111 will enable all sonars.

Each enabled sonar provides its measurement at the corresponding register DIST0-DIST7. This value is in mm and has a resolution of 1mm, but its accuracy (both linearity and offset) is undefined and typically varies ±1cm. The minimum typical readable distance is approximately 15cm and the maximum is 6m, though distant objects may not return sufficient ping be detected depending on their shape, size, and material.

If no ping is returned (usually because either nothing is in front of the sonar, or the object is angled such that the ping bounces in another direction), the measurement value is set to the maximum positive value of 0x7FFF.

Sonars update in a round-robin fashion at 25Hz, skipping any that are not enabled. If all sonars are enabled, each particular measurement will update at 3.125Hz (25Hz/8). If only one sonar is enabled, it will update at the full 25Hz. Thus, it is beneficial to only enable the sonars that you plan to read values from.

Sonar Interrupts

If not using interrupts, this information is not needed.

The value written to IO location SONALARM is used as an alarm distance by the sonar module: if any enabled sonar measures an object closer than that distance, a bit is set in an alarm register, which can then be read from the same SONALARM IO address. A '1' in any bit 0-7 of the returned value indicates that the corresponding sonar (0-7) detected something too close during its previous measurement.

This system can also cause an interrupt in SCOMP on the relevant pin. By default, any enabled sonar that measures a close object will cause an interrupt; however, a mask can be written to IO location SONARINT: any '0' bit in that mask will disable interrupts from that sonar. For example, writing &B00100001 will only allow the two side sonars (0 and 5) to cause interrupts.

Note that a sonar must still be enabled through the SONAREN register in order to cause interrupts. Also, the SONARINT mask does *not* affect the alarm register read from SONALARM; any enabled sonar can always cause bits to be set in SONALARM.

The sonar module will cause an interrupt with each new measurement that fits the alarm condition, so be aware that it might cause interrupts at up to 25Hz (the read rate of the sonars). Interrupts can be disabled at any time by configuring either the sonar module (through SONARINT) or SCOMP (using CLI).

Wheel Position, Velocity, and Velocity Commands

The values read from LPOS and RPOS provide the wheel encoder counts since reset. The encoders provide 304 ticks/revolution, which corresponds to linear movement of approximately 1.05mm/count. LVEL and RVEL

provide approximations of wheel velocity by sampling the position every 0.1s and providing the difference multiplied by 10; the units are thus approximately 1.05mm/s.

LVELCMD and RVELCMD accept values in the same units as LVEL and RVEL, and attempt to control the wheel velocities to match that value. Be aware that very low speeds (usually <50mm/s) may have difficulty overcoming the static friction of the motors, gearboxes, axles, and wheels, and so may result in no movement or spurious movement. However, once moving, the lower bits of LVELCMD and RVELCMD do provide additional resolution to the speed.

The values sent to LVELCMD and RVELCMD should not exceed ±511. If a value outside that range is provided, the motor controller will interpret it as 0 (stopped). Sending any value to RESETPOS will reset the positions to 0, as well as stop the robot

The acceleration (including deceleration) of each wheel is controlled to 512units/s. An important side effect of this is that overshoot can be approximated and accounted for. In general, the distance required to change velocity is $\frac{{v_1}^2 - {v_2}^2}{2a}$, so, simplified and applied to this robot, the distance required to stop can be estimated by $VEL^2/1024$ (where both VEL and the resulting distance are in robot units). Extending this to rotations, assuming in-place rotations with equal but opposite wheel velocities VEL_{turn} , overshoot (in degrees) can be estimated as ${VEL_{turn}}^2/2030$.

Odometry

The Quartus DE2Bot project contains a device that performs dead-reckoning odometry: continuously integrating the movement of the wheels to maintain an estimate of the robot's position and heading. This estimation can be read from IO registers XPOS, YPOS, and THETA. The units for the XPOS and YPOS are 1.05mm/count – the same as the resolution of LPOS and RPOS. THETA is in degrees.

At power-up or reset, the position of the robot defaults to X,Y,θ=0,0,0. The coordinate system is shown in Figure 3: the reset orientation is defined as facing the positive X direction, with positive Y to the left, and theta following the normal right-handed convention (with Z upwards).

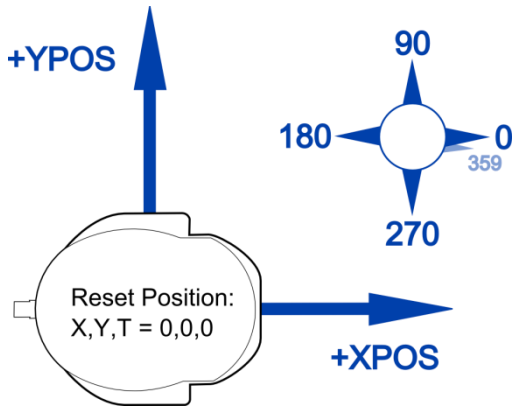


Figure 3. Coordinate system used for DE2Bot odometry.

The value of THETA will always be [0,359]; rotating counterclockwise past 359 will roll over to 0, and rotating clockwise past 0 will roll over to 359.

Writing any value to the RESETPOS register will reset all position registers – XPOS, YPOS, THETA, LPOS, and RPOS – to 0, as well as stop the robot (set LVELCMD and RVELCMD to 0).

Dead-reckoning is highly susceptible to accumulated error from wheel slippage, wheel-size and wheel-base errors, mathematical rounding, and other sources. The odometry values will likely contain significant error after as little as a few meters of travel or one rotation of the robot, and much of the error will *not* be systematic (i.e.

there is no way to correct for it). Frequent turning generally exacerbates this problem, as turns cause the most wheel slippage, and angular errors result in increasing linear errors.

I²C Controller and Battery Voltage

The DE2Bot contains an I²C bus, which is currently used to communicate with the A/D converter that measures the battery voltage. SCOMP interfaces with the I²C bus through a controller with three I/O registers:

- `I2C_CMD`: write-only; contains configuration information for the controller.
  - bits 15-12: number of bytes to write (0, 1, or 2)
  - bits 11-9: number of bytes to read (0, 1, or 2)
  - bits 8-1: 7-bit I²C address of device to communicate with; excludes RnW bit
  - bit 0: ignored; the RnW bit is set on the fly according to the current operation
- `I2C_DATA`: read/write; data to send, and data received.
  - If transmitting or receiving one byte, bits 7-0 are used
  - If transmitting or receiving two bytes, bits 15-9 are the first byte, then bits 7-0
- `I2C_RDY`: read/write; status indicator
  - Writing to I2C_RDY begins an I2C transaction; set up I2C_CMD and I2C_DATA first.
  - Reading I2C_RDY will return zero if the controller is idle, or non-zero if a communication is in progress. Do not modify I2C_CMD or I2C_DATA while I2C_RDY reads as non-zero.

Wireless Serial Communication

Writing to `UART_DAT` will transmit a single byte. Because SCOMP's IO system is 16-bit, only the low byte is sent; the high byte is discarded.

Important: the UART is internally rate-limited to ensure that the available bandwidth is shared among all active robots. You must adhere to the following rules while transmitting serial data:

1. At most 10 bytes may be sent every 200ms (five times per second).
2. All bytes within a transmission period must be sent together (within 1ms of the preceding byte) so that they can be grouped in to a single wireless packet.

The UART peripheral contains a FIFO transmit buffer, so SCOMP may write bytes without considering the actual transmission time as long as the above rules are followed. However, any bytes that break the above rules are discarded.

Received data is stored in a 64-level FIFO receive buffer and is accessed by reading from `UART_DAT`, which automatically removes the byte from the buffer. If the buffer was non-empty (i.e. the byte is valid), bit 15 will be set.

Reading the `UART_RDY` register provides the following status information:

- Bits 6-0 : number of bytes currently in the receive buffer (maximum 63)
- Bit 7 : data available; '1' if receive buffer is not empty

Configurable Timer Interrupt Source

If not using interrupts, this information is not needed.

Writing to the `CTIMER` IO location sets the period of the timer interrupts in increments of 10ms. For example, sending "1" will cause an interrupt every 10ms; writing "30,000" will cause an interrupt after 5 minutes. The value is interpreted as unsigned, with a maximum value of 0xFFFF (slightly over 10 minutes).

Writing to the device automatically resets the internal counter, but the first increment of the counter could occur at any time within the following 10ms; thus, the first interrupt will actually occur between N and N-1 intervals (of 10ms). Thereafter, interrupts will occur at the correct intervals.

3.5 Good Practices for Robot Programming

This section details some recommended practices for safe and effective use of the DE2Bot.

At Program Start

As soon as the program starts or is reset, the following should be done in order:

1. Immediately stop the robot by writing 0 to LVELCMD and RVELCMD
2. Check the battery voltage, and prevent execution if it is below 13V
3. Wait for the safety switch (SW17) to be toggled
4. Wait for some form of user input (e.g. pressing a PB)

An example of this initialization procedure can be found in the code in Appendix A.

Testing Values

Two points must be kept in mind when making decisions based on values obtained from LPOS/RPOS, odometry, sonars, or any other real-world measurement:

- There is no guarantee that a particular value will occur, so never test for an exact value. Instead, always test for a range.
    - Example1: polling LPOS while the robot is moving might return 0xFE at one sample and 0x100 at the next sample, so testing for 0xFF will never pass.
        - Testing for ≥0xFF would correctly trigger even if 0xFF itself never occurs.
    - Example2: many values are impossible to obtain from peripherals because of limited range or resolution. For example, LVEL and RVEL only have a resolution of 10 units, so testing for exactly 15 will never pass.
- Be aware of edge conditions, which can erroneously cause tests to pass or fail.
    - Example 1: Ideally THETA will be 0 after reset, but any small turn clockwise will change it to 359, in which case a test for theta>X (intended to check if the robot has turned a certain amount counterclockwise) will immediately pass.

Modularity

Code is much easier to debug, maintain, and extend if it is designed "bottom-to-top"; i.e. basic functionality written first, which is used to perform larger but still basic tasks, which are then used to complete the overall function of the code. It is easier to find errors in a small function tested independently than in a monolithic does-all piece of software.

Appendix A:

Example starting point for ASM code.

```
; SimpleRobotProgram.asm
; Created by Kevin Johnson
; (no copyright applied; edit freely, no attribution necessary)
; This program does basic initialization of the DE2Bot
; and provides an example of some peripherals.

; Section labels are for clarity only.


ORG        &H000       ;Begin program at x000
;***************************************************************
;* Initialization
;***************************************************************
Init:
    ; Always a good idea to make sure the robot
    ; stops in the event of a reset.
    LOAD   Zero
    OUT    LVELCMD     ; Stop motors
    OUT    RVELCMD
    OUT    SONAREN     ; Disable sonar (optional)

    CALL   SetupI2C    ; Configure the I2C to read the battery voltage
    CALL   BattCheck   ; Get battery voltage (and end if too low).
    OUT    LCD         ; Display batt voltage on LCD

WaitForSafety:
    ; Wait for safety switch to be toggled
    IN     XIO         ; XIO contains SAFETY signal
    AND    Mask4       ; SAFETY signal is bit 4
    JPOS   WaitForUser ; If ready, jump to wait for PB3
    IN     TIMER       ; We'll use the timer value to
    AND    Mask1       ;  blink LED17 as a reminder to toggle SW17
    SHIFT  8           ; Shift over to LED17
    OUT    XLEDS       ; LED17 blinks at 2.5Hz (10Hz/4)
    JUMP   WaitForSafety

WaitForUser:
    ; Wait for user to press PB3
    IN     TIMER       ; We'll blink the LEDs above PB3
    AND    Mask1
    SHIFT  5           ; Both LEDG6 and LEDG7
    STORE  Temp        ; (overkill, but looks nice)
    SHIFT  1
    OR     Temp
    OUT    XLEDS
    IN     XIO         ; XIO contains KEYs
    AND    Mask2       ; KEY3 mask (KEY0 is reset and can't be read)
    JPOS   WaitForUser ; not ready (KEYs are active-low, hence JPOS)
    LOAD   Zero
    OUT    XLEDS       ; clear LEDs once ready to continue

;***************************************************************
;* Main code
;***************************************************************
Main: ; "Real" program starts here.
    OUT    RESETPOS    ; reset odometry in case wheels moved after programming

; The following code ("Center" through "DeadZone") is purely for example.
; It attempts to gently keep the robot facing 0 degrees, showing how the
```

```
; odometer and motor controllers work.
Center:
    ; The 0/359 jump in THETA can be difficult to deal with.
    ; This code shows one way to handle it: by moving the
    ; discontinuity away from the current heading.
    IN     THETA       ; get the current angular position
    ADDI   -180        ; test whether facing 0-179 or 180-359
    JPOS   NegAngle    ; robot facing 180-360; handle that separately
PosAngle:
    ADDI   180         ; undo previous subtraction
    JUMP   CheckAngle  ; THETA positive, so carry on
NegAngle:
    ADDI   -180        ; finish conversion to negative angle:
                       ;  angles 180 to 359 become -180 to -1

CheckAngle:
    ; AC now contains the +/- angular error from 0, meaning that
    ;  the discontinuity is at 179/-180 instead of 0/359
    OUT    LCD         ; Good data to display for debugging
    ; As an example of math, multiply the error by 5 :
    ; (AC + AC<<2) = AC*5
    STORE  Temp
    SHIFT  2          ; divide by two
    ADD    Temp        ; add original value

    ; Cap velcmd at +/-100 (a slow speed)
    JPOS   CapPos      ; handle +/- separately
CapNeg:
    ADD    DeadZone    ; if close to 0, don't do anything
    JPOS   NoTurn      ; (don't do anything)
    SUB    DeadZone    ; restore original value
    ADDI   100         ; check for <-100
    JPOS   NegOK       ; it was not <-100, so carry on
    LOAD   Zero        ; it was <-100, so clear excess
NegOK:
    ADDI   -100        ; undo the previous addition
    JUMP   SendToMotors
CapPos:
    SUB    DeadZone    ; if close to 0, don't do anything
    JNEG   NoTurn
    ADD    DeadZone    ; restore original value
    ADDI   -100
    JNEG   PosOK       ; it was not >100, so carry on
    LOAD   Zero        ; it was >100, so clear excess
PosOK:
    ADDI   100         ; undo the previous subtraction
    JUMP   SendToMotors
NoTurn:
    LOAD   Zero
    JUMP   SendToMotors

    ; The desired velocity (angular error * 1.5, capped at
    ;  +/-100, and with a 2-degree dead zone) is now in AC
SendToMotors:
    ; Since we want to spin in place, we need to send inverted
    ;  velocities to the wheels.
    STORE  Temp        ; store calculated desired velocity
    ; send the direct value to the left wheel
;   ADD    FMid        ; Could add an offset vel here to move forward
```

```
    OUT    LVELCMD
    OUT    SSEG1       ; for debugging purposes
    ; send the negated number to the right wheel
    LOAD   Zero
    SUB    Temp        ; AC = 0 - AC
;   ADD    Fmid        ; Could add an offset vel here to move forward
    OUT    RVELCMD
    OUT    SSEG2       ; debugging

    JUMP   Center      ; repeat forever

DeadZone:  DW 10       ; Actual deadzone will be /5 due to scaling above.
                       ; Note that you can place data anywhere.
                       ; Just be careful that it doesn't get executed.

Die:
; Sometimes it's useful to permanently stop execution.
; This will also catch the execution if it accidentally
; falls through from above.
    LOAD   Zero         ; Stop everything.
    OUT    LVELCMD
    OUT    RVELCMD
    OUT    SONAREN
    LOAD   DEAD         ; An indication that we are dead
    OUT    SSEG2
Forever:
    JUMP   Forever      ; Do this forever.
DEAD: DW &HDEAD


;***************************************************************
;* Subroutines
;***************************************************************

; Subroutine to wait (block) for 1 second
Wait1:
    OUT    TIMER
Wloop:
    IN     TIMER
    OUT    XLEDS       ; User-feedback that a pause is occurring.
    ADDI   -10         ; 1 second in 10Hz.
    JNEG   Wloop
    RETURN

; Subroutine to wait the number of counts currently in AC
WaitAC:
    STORE  WaitTime
    OUT    Timer
WACLoop:
    IN     Timer
    OUT    XLEDS       ; User-feedback that a pause is occurring.
    SUB    WaitTime
    JNEG   WACLoop
    RETURN
    WaitTime: DW 0     ; "local" variable.

; This subroutine will get the battery voltage,
; and stop program execution if it is too low.
; SetupI2C must be executed prior to this.
```

```
BattCheck:
    CALL   GetBattLvl
    JZERO  BattCheck   ; A/D hasn't had time to initialize
    SUB    MinBatt
    JNEG   DeadBatt
    ADD    MinBatt     ; get original value back
    RETURN
; If the battery is too low, we want to make
; sure that the user realizes it...
DeadBatt:
    LOAD   Four
    OUT    BEEP        ; start beep sound
    CALL   GetBattLvl  ; get the battery level
    OUT    SSEG1       ; display it everywhere
    OUT    SSEG2
    OUT    LCD
    LOAD   Zero
    ADDI   -1          ; 0xFFFF
    OUT    LEDS        ; all LEDs on
    OUT    XLEDS
    CALL   Wait1       ; 1 second
    Load   Zero
    OUT    BEEP        ; stop beeping
    LOAD   Zero
    OUT    LEDS        ; LEDs off
    OUT    XLEDS
    CALL   Wait1       ; 1 second
    JUMP   DeadBatt    ; repeat forever

; Subroutine to read the A/D (battery voltage)
; Assumes that SetupI2C has been run
GetBattLvl:
    LOAD   I2CRCmd     ; 0x0190 (write 0B, read 1B, addr 0x90)
    OUT    I2C_CMD     ; to I2C_CMD
    OUT    I2C_RDY     ; start the communication
    CALL   BlockI2C    ; wait for it to finish
    IN     I2C_DATA    ; get the returned data
    RETURN

; Subroutine to configure the I2C for reading batt voltage
; Only needs to be done once after each reset.
SetupI2C:
    CALL   BlockI2C    ; wait for idle
    LOAD   I2CWCmd     ; 0x1190 (write 1B, read 1B, addr 0x90)
    OUT    I2C_CMD     ; to I2C_CMD register
    LOAD   Zero        ; 0x0000 (A/D port 0, no increment)
    OUT    I2C_DATA    ; to I2C_DATA register
    OUT    I2C_RDY     ; start the communication
    CALL   BlockI2C    ; wait for it to finish
    RETURN

; Subroutine to block until I2C device is idle
BlockI2C:
    LOAD   Zero
    STORE  Temp        ; Used to check for timeout
BI2CL:
    LOAD   Temp
    ADDI   1           ; this will result in ~0.1s timeout
    STORE  Temp
```

```
    JZERO  I2CError    ; Timeout occurred; error
    IN     I2C_RDY     ; Read busy signal
    JPOS   BI2CL       ; If not 0, try again
    RETURN             ; Else return
I2CError:
    LOAD   Zero
    ADDI   &H12C       ; "I2C"
    OUT    SSEG1
    OUT    SSEG2       ; display error message
    JUMP   I2CError

; Subroutine to send AC value through the UART,
; formatted for default base station code:
; [ AC(15..8) | AC(7..0) | \lf ]
; Note that special characters such as \lf are
; escaped with the value 0x1B, thus the literal
; value 0x1B must be sent as 0x1B1B, should it occur.
UARTSend:
    STORE  UARTTemp
    SHIFT  -8
    ADDI   -27   ; escape character
    JZERO  UEsc1
    ADDI   27
    OUT    UART_DAT
    JUMP   USend2
UEsc1:
    ADDI   27
    OUT    UART_DAT
    OUT    UART_DAT
USend2:
    LOAD   UARTTemp
    AND    LowByte
    ADDI   -27   ; escape character
    JZERO  UEsc2
    ADDI   27
    OUT    UART_DAT
    RETURN
UEsc2:
    ADDI   27
    OUT    UART_DAT
    OUT    UART_DAT
    RETURN
    UARTTemp: DW 0

UARTNL:
    LOAD   NL
    OUT    UART_DAT
    SHIFT  -8
    OUT    UART_DAT
    RETURN
    NL: DW &H0A1B

;***************************************************************
;* Variables
;***************************************************************
Temp:     DW 0 ; "Temp" is not a great name, but can be useful

;***************************************************************
;* Constants
```

```
;* (though there is nothing stopping you from writing to these)
;*****************************************************************
NegOne:   DW -1
Zero:     DW 0
One:      DW 1
Two:      DW 2
Three:    DW 3
Four:     DW 4
Five:     DW 5
Six:      DW 6
Seven:    DW 7
Eight:    DW 8
Nine:     DW 9
Ten:      DW 10

; Some bit masks.
; Masks of multiple bits can be constructed by ORing these
; 1-bit masks together.
Mask0:    DW &B00000001
Mask1:    DW &B00000010
Mask2:    DW &B00000100
Mask3:    DW &B00001000
Mask4:    DW &B00010000
Mask5:    DW &B00100000
Mask6:    DW &B01000000
Mask7:    DW &B10000000
LowByte:  DW &HFF      ; binary 00000000 1111111
LowNibl:  DW &HF       ; 0000 0000 0000 1111

; some useful movement values
OneMeter: DW 961       ; ~1m in 1.05mm units
HalfMeter: DW 481      ; ~0.5m in 1.05mm units
TwoFeet:  DW 586       ; ~2ft in 1.05mm units
Deg90:    DW 90        ; 90 degrees in odometry units
Deg180:   DW 180       ; 180
Deg270:   DW 270       ; 270
Deg360:   DW 360       ; can never actually happen; for math only
FSlow:    DW 100       ; 100 is about the lowest velocity value that will move
RSlow:    DW -100
FMid:     DW 350       ; 350 is a medium speed
RMid:     DW -350
FFast:    DW 500       ; 500 is almost max speed (511 is max)
RFast:    DW -500

MinBatt:  DW 140       ; 14.0V - minimum safe battery voltage
I2CWCmd:  DW &H1190    ; write one i2c byte, read one byte, addr 0x90
I2CRCmd:  DW &H0190    ; write nothing, read one byte, addr 0x90

;*****************************************************************
;* IO address space map
;*****************************************************************
SWITCHES: EQU &H00  ; slide switches
LEDS:     EQU &H01  ; red LEDs
TIMER:    EQU &H02  ; timer, usually running at 10 Hz
XIO:      EQU &H03  ; pushbuttons and some misc. inputs
SSEG1:    EQU &H04  ; seven-segment display (4-digits only)
SSEG2:    EQU &H05  ; seven-segment display (4-digits only)
LCD:      EQU &H06  ; primitive 4-digit LCD display
XLEDS:    EQU &H07  ; Green LEDs (and Red LED16+17)
```

```
BEEP:     EQU &H0A  ; Control the beep
CTIMER:   EQU &H0C  ; Configurable timer for interrupts
LPOS:     EQU &H80  ; left wheel encoder position (read only)
LVEL:     EQU &H82  ; current left wheel velocity (read only)
LVELCMD:  EQU &H83  ; left wheel velocity command (write only)
RPOS:     EQU &H88  ; same values for right wheel...
RVEL:     EQU &H8A  ; ...
RVELCMD:  EQU &H8B  ; ...
I2C_CMD:  EQU &H90  ; I2C module's CMD register,
I2C_DATA: EQU &H91  ; ... DATA register,
I2C_RDY:  EQU &H92  ; ... and BUSY register
UART_DAT: EQU &H98  ; UART data
UART_RDY: EQU &H98  ; UART status
SONAR:    EQU &HA0  ; base address for more than 16 registers....
DIST0:    EQU &HA8  ; the eight sonar distance readings
DIST1:    EQU &HA9  ; ...
DIST2:    EQU &HAA  ; ...
DIST3:    EQU &HAB  ; ...
DIST4:    EQU &HAC  ; ...
DIST5:    EQU &HAD  ; ...
DIST6:    EQU &HAE  ; ...
DIST7:    EQU &HAF  ; ...
SONALARM: EQU &HB0  ; Write alarm distance; read alarm register
SONARINT: EQU &HB1  ; Write mask for sonar interrupts
SONAREN:  EQU &HB2  ; register to control which sonars are enabled
XPOS:     EQU &HC0  ; Current X-position (read only)
YPOS:     EQU &HC1  ; Y-position
THETA:    EQU &HC2  ; Current rotational position of robot (0-359)
RESETPOS: EQU &HC3  ; write anything here to reset odometry to 0
```

Appendix B:

Example ASM with Interrupt Handling

```
; InterruptsExample.asm
; Created by Kevin Johnson
; (no copyright applied; edit freely, no attribution necessary)
; This program does basic initialization of the DE2Bot
; and provides an example of SCOMP's interrupt system.

; Section labels are for clarity only.


;***************************************************************
;* Jump Table
;***************************************************************
; When an interrupt occurs, execution is redirected to one of
; these addresses (depending on the interrupt source), which
; need to either contain RETI (return from interrupt) if not
; used, or a JUMP instruction to the desired interrupt service
; routine (ISR).  The first location is the reset vector, and
; should be a JUMP instruction to the beginning of your normal
; code.
ORG        &H000       ; Jump table is located in mem 0-4
    JUMP   Init        ; Reset vector
    JUMP   Sonar_ISR   ; Sonar interrupt
    JUMP   CTimer_ISR  ; Timer interrupt
    RETI               ; UART interrupt (unused here)
    JUMP   Stall_ISR   ; Motor stall interrupt
;***************************************************************
;* Initialization
;***************************************************************
Init:
    ; Always a good idea to make sure the robot
    ; stops in the event of a reset.
    LOAD   Zero
    OUT    LVELCMD     ; Stop motors
    OUT    RVELCMD
    OUT    SONAREN     ; Disable sonar (optional)

    CALL   SetupI2C    ; Configure the I2C to read the battery voltage
    CALL   BattCheck   ; Get battery voltage (and end if too low).
    OUT    LCD         ; Display batt voltage on LCD

WaitForSafety:         ; Wait for safety switch to be toggled
    IN     XIO         ; XIO contains SAFETY signal
    AND    Mask4       ; SAFETY signal is bit 4
    JPOS   WaitForUser ; If ready, jump to wait for PB3
    IN     TIMER       ; We'll use the timer value to
    AND    Mask1       ;  blink LED17 as a reminder to toggle SW17
    SHIFT  8           ; Shift over to LED17
    OUT    XLEDS       ; LED17 blinks at 2.5Hz (10Hz/4)
    JUMP   WaitForSafety

WaitForUser:           ; Wait for user to press PB3
    IN     TIMER       ; We'll blink the LEDs above PB3
    AND    Mask1
    SHIFT  5           ; Both LEDG6 and LEDG7
    STORE  Temp        ; (overkill, but looks nice)
    SHIFT  1
    OR     Temp
    OUT    XLEDS
    IN     XIO         ; XIO contains KEYs
```

```
    AND    Mask2       ; KEY3 mask (KEY0 is reset and can't be read)
    JPOS   WaitForUser ; not ready (KEYs are active-low, hence JPOS)
    LOAD   Zero
    OUT    XLEDS       ; clear LEDs once ready to continue

;***************************************************************
;* Main code
;***************************************************************
Main: ; "Real" program starts here.
    OUT    RESETPOS    ; reset odometry in case wheels moved after programming

; This code (as a whole) will use Sonar0 and Sonar5 to turn
; the robot away from whichever sonar detects an object within 1ft.
; As long as a sonar remains triggered, the robot will keep turning
; away from it, effectively making it end up facing away from the
; last thing it 'saw'.
Config:
    LOAD   Zero
    STORE  DesTheta    ; reset the variables
    STORE  TCount
    STORE  SonCount

    LOADI  &B100001
    OUT    SONAREN     ; turn on sonars 0 and 5
    LOADI  254
    OUT    SONALARM    ; set alarm distance to 254mm

    LOADI  5
    OUT    CTIMER      ; configure timer for 0.01*5=0.05s interrupts

    SEI    &B1011      ; enable the desired interrupts

InfLoop:
    ; everything will be handled by interrupts, so the main
    ; code just displays some values that are updated by
    ; the ISRs.
    LOAD   TCount      ; The timer ISR increments this at 20Hz
    OUT    SSEG1
    LOAD   SonCount    ; The sonar ISR increments this each warning
    OUT    SSEG2
    JUMP   InfLoop


;***************************************************************
;* Interrupt Service Routines
;***************************************************************
CTimer_ISR:  ; Timer interrupt
; The timer interrupt will be used to turn the robot,
; correcting for turn rate, etc., at 20Hz.
    LOAD   TCount
    ADDI   1
    STORE  TCount

    IN     THETA       ; get current angle
    STORE  NowTheta    ; save for later use
    SUB    DesTheta    ; subtract desired angle
    CALL   Mod360      ; remove negative numbers
    ADDI   -180        ; test which semicircle error is in
    JPOS   NeedLeft    ; >180 means need left turn
```

```
    JUMP   NeedRight   ; otherwise, need right turn
NeedLeft:
    LOAD   DesTheta
    SUB    NowTheta    ; get the turn error
    CALL   Mod360      ; fix errors around 0
    SUB    DeadZone
    JNEG   NoTurn      ; stop moving if close
    ADD    DeadZone
    ADDI   -100        ; check if >100
    JNEG   TurnLeft
    LOAD   Zero        ; remove excess
TurnLeft:
    ADDI   100         ; replace the 100 from before
    SHIFT  2           ; multiply by 4
    OUT    RVELCMD     ; set right wheel forward
    XOR    NegOne
    ADDI   1           ; negate number
    OUT    LVELCMD     ; set left wheel backwards
    RETI               ; exit ISR
NeedRight:
    LOAD   NowTheta
    SUB    DesTheta    ; get the turn error
    CALL   Mod360      ; fix errors around 0
    SUB    DeadZone
    JNEG   NoTurn      ; stop moving if close
    ADD    DeadZone
    ADDI   -100        ; check if >100
    JNEG   TurnRight
    LOAD   Zero        ; remove excess
TurnRight:
    ADDI   100         ; replace the 100 from before
    SHIFT  2           ; multiply by 4
    OUT    LVELCMD     ; set left wheel forward
    XOR    NegOne
    ADDI   1           ; negate number
    OUT    RVELCMD     ; set left wheel backwards
    RETI               ; exit ISR
NoTurn:
    LOAD   Zero
    OUT    LVELCMD
    OUT    RVELCMD
    RETI

    NowTheta: DW 0
    DeadZone: DW 3
    TCount: DW 0

Sonar_ISR:   ; Sonar interrupt
; The sonar interrupt will update the "desired heading"
; variable according to which sonar caused the interrupt.
    LOAD   SonCount
    ADDI   1
    STORE  SonCount
    IN     SONALARM    ; get the alarm register
    AND    Mask05      ; keep only bits 0 and 5
    STORE  Temp
    JZERO  NoChange    ; nothing set, no turn
    XOR    Mask05      ; check for BOTH bits set
    JZERO  NoChange    ;  in which case, don't turn
```

```
    LOAD   Temp
    AND    Mask0       ; check for left sonar
    JPOS   SetLeft
    JUMP   SetRight    ; only remaining possibility is right
    Mask05: DW &B100001
NoChange:
    RETI               ; don't need to turn; just exit ISR
SetLeft:
    IN     THETA       ; get current angle
    SUB    TDist       ; set turn distance
    CALL   Mod360      ; calculate mod 360
    STORE  DesTheta    ; set the desired angle
    RETI               ; exit ISR
SetRight:
    IN     THETA       ; get current angle
    ADD    TDist       ; set turn distance
    CALL   Mod360      ; calculate mod 360
    STORE  DesTheta    ; set the desired angle
    RETI               ; exit ISR
    SonCount: DW 0
Stall_ISR:   ; Motor stall interrupt
    RETI
    TDist: DW 100

DesTheta: DW 0

;***************************************************************
;* Subroutines
;***************************************************************

; Subroutine to wait (block) for 1 second
Wait1:
    OUT    TIMER
Wloop:
    IN     TIMER
    OUT    XLEDS       ; User-feedback that a pause is occurring.
    ADDI   -10         ; 1 second in 10Hz.
    JNEG   Wloop
    RETURN

; Subroutine to wait the number of counts currently in AC
WaitAC:
    STORE  WaitTime
    OUT    Timer
WACLoop:
    IN     Timer
    OUT    XLEDS       ; User-feedback that a pause is occurring.
    SUB    WaitTime
    JNEG   WACLoop
    RETURN
    WaitTime: DW 0     ; "local" variable.

; Converts an angle to [0,359]
Mod360:
    JNEG   M360N       ; loop exit condition
    ADDI   -360        ; start removing 360 at a time
    JUMP   Mod360      ; keep going until negative
M360N:
    ADDI   360         ; get back to positive
```

```
    JNEG   M360N       ; (keep adding 360 until non-negative)
    RETURN

; This subroutine will get the battery voltage,
; and stop program execution if it is too low.
; SetupI2C must be executed prior to this.
BattCheck:
    CALL   GetBattLvl
    JZERO  BattCheck   ; A/D hasn't had time to initialize
    SUB    MinBatt
    JNEG   DeadBatt
    ADD    MinBatt     ; get original value back
    RETURN
; If the battery is too low, we want to make
; sure that the user realizes it...
DeadBatt:
    LOAD   Four
    OUT    BEEP        ; start beep sound
    CALL   GetBattLvl  ; get the battery level
    OUT    SSEG1       ; display it everywhere
    OUT    SSEG2
    OUT    LCD
    LOAD   Zero
    ADDI   -1          ; 0xFFFF
    OUT    LEDS        ; all LEDs on
    OUT    XLEDS
    CALL   Wait1       ; 1 second
    Load   Zero
    OUT    BEEP        ; stop beeping
    LOAD   Zero
    OUT    LEDS        ; LEDs off
    OUT    XLEDS
    CALL   Wait1       ; 1 second
    JUMP   DeadBatt    ; repeat forever

; Subroutine to read the A/D (battery voltage)
; Assumes that SetupI2C has been run
GetBattLvl:
    LOAD   I2CRCmd     ; 0x0190 (write 0B, read 1B, addr 0x90)
    OUT    I2C_CMD     ; to I2C_CMD
    OUT    I2C_RDY     ; start the communication
    CALL   BlockI2C    ; wait for it to finish
    IN     I2C_DATA    ; get the returned data
    RETURN

; Subroutine to configure the I2C for reading batt voltage
; Only needs to be done once after each reset.
SetupI2C:
    CALL   BlockI2C    ; wait for idle
    LOAD   I2CWCmd     ; 0x1190 (write 1B, read 1B, addr 0x90)
    OUT    I2C_CMD     ; to I2C_CMD register
    LOAD   Zero        ; 0x0000 (A/D port 0, no increment)
    OUT    I2C_DATA    ; to I2C_DATA register
    OUT    I2C_RDY     ; start the communication
    CALL   BlockI2C    ; wait for it to finish
    RETURN

; Subroutine to block until I2C device is idle
BlockI2C:
```

```
    LOAD   Zero
    STORE  Temp        ; Used to check for timeout
BI2CL:
    LOAD   Temp
    ADDI   1           ; this will result in ~0.1s timeout
    STORE  Temp
    JZERO  I2CError    ; Timeout occurred; error
    IN     I2C_RDY     ; Read busy signal
    JPOS   BI2CL       ; If not 0, try again
    RETURN             ; Else return
I2CError:
    LOAD   Zero
    ADDI   &H12C       ; "I2C"
    OUT    SSEG1
    OUT    SSEG2       ; display error message
    JUMP   I2CError


;***************************************************************
;* Variables
;***************************************************************
Temp:     DW 0 ; "Temp" is not a great name, but can be useful

;***************************************************************
;* Constants
;* (though there is nothing stopping you from writing to these)
;***************************************************************
NegOne:   DW -1
Zero:     DW 0
One:      DW 1
Two:      DW 2
Three:    DW 3
Four:     DW 4
Five:     DW 5
Six:      DW 6
Seven:    DW 7
Eight:    DW 8
Nine:     DW 9
Ten:      DW 10

; Some bit masks.
; Masks of multiple bits can be constructed by ORing these
; 1-bit masks together.
Mask0:    DW &B00000001
Mask1:    DW &B00000010
Mask2:    DW &B00000100
Mask3:    DW &B00001000
Mask4:    DW &B00010000
Mask5:    DW &B00100000
Mask6:    DW &B01000000
Mask7:    DW &B10000000
LowByte:  DW &HFF      ; binary 00000000 1111111
LowNibl:  DW &HF       ; 0000 0000 0000 1111

; some useful movement values
OneMeter: DW 952       ; ~1m in 1.05mm units
HalfMeter: DW 476      ; ~0.5m in 1.05mm units
TwoFeet:  DW 581       ; ~2ft in 1.05mm units
Deg90:    DW 90        ; 90 degrees in odometry units
```

```
Deg180:   DW 180       ; 180
Deg270:   DW 270       ; 270
Deg360:   DW 360       ; can never actually happen; for math only
FSlow:    DW 100       ; 100 is about the lowest velocity value that will move
RSlow:    DW -100
FMid:     DW 350       ; 350 is a medium speed
RMid:     DW -350
FFast:    DW 500       ; 500 is almost max speed (511 is max)
RFast:    DW -500

MinBatt:  DW 140       ; 14.0V - minimum safe battery voltage
I2CWCmd:  DW &H1190    ; write one i2c byte, read one byte, addr 0x90
I2CRCmd:  DW &H0190    ; write nothing, read one byte, addr 0x90

;***************************************************************
;* IO address space map
;***************************************************************
SWITCHES: EQU &H00  ; slide switches
LEDS:     EQU &H01  ; red LEDs
TIMER:    EQU &H02  ; timer, usually running at 10 Hz
XIO:      EQU &H03  ; pushbuttons and some misc. inputs
SSEG1:    EQU &H04  ; seven-segment display (4-digits only)
SSEG2:    EQU &H05  ; seven-segment display (4-digits only)
LCD:      EQU &H06  ; primitive 4-digit LCD display
XLEDS:    EQU &H07  ; Green LEDs (and Red LED16+17)
BEEP:     EQU &H0A  ; Control the beep
CTIMER:   EQU &H0C  ; Configurable timer for interrupts
LPOS:     EQU &H80  ; left wheel encoder position (read only)
LVEL:     EQU &H82  ; current left wheel velocity (read only)
LVELCMD:  EQU &H83  ; left wheel velocity command (write only)
RPOS:     EQU &H88  ; same values for right wheel...
RVEL:     EQU &H8A  ; ...
RVELCMD:  EQU &H8B  ; ...
I2C_CMD:  EQU &H90  ; I2C module's CMD register,
I2C_DATA: EQU &H91  ; ... DATA register,
I2C_RDY:  EQU &H92  ; ... and BUSY register
UART_DAT: EQU &H98  ; UART data
UART_RDY: EQU &H98  ; UART status
SONAR:    EQU &HA0  ; base address for more than 16 registers....
DIST0:    EQU &HA8  ; the eight sonar distance readings
DIST1:    EQU &HA9  ; ...
DIST2:    EQU &HAA  ; ...
DIST3:    EQU &HAB  ; ...
DIST4:    EQU &HAC  ; ...
DIST5:    EQU &HAD  ; ...
DIST6:    EQU &HAE  ; ...
DIST7:    EQU &HAF  ; ...
SONALARM: EQU &HB0  ; Write alarm distance; read alarm register
SONARINT: EQU &HB1  ; Write mask for sonar interrupts
SONAREN:  EQU &HB2  ; register to control which sonars are enabled
XPOS:     EQU &HC0  ; Current X-position (read only)
YPOS:     EQU &HC1  ; Y-position
THETA:    EQU &HC2  ; Current rotational position of robot (0-359)
RESETPOS: EQU &HC3  ; write anything here to reset odometry to 0
```