

Development Kit For the PIC[®] MCU

Exercise Book

Robotics

March 2010



Custom Computer Services, Inc.
Brookfield, Wisconsin, USA
262-522-6500

Copyright © 2010 Custom Computer Services, Inc.

All rights reserved worldwide. No part of this work may be reproduced or copied in any form by any means—electronic, graphic or mechanical, including photocopying, recording, taping or information retrieval systems—without written permission.

PIC[®] and PICmicro[®] are registered trademarks of Microchip Technology Inc. in the USA and in other countries.



Custom Computer Services, Inc. proudly supports the Microchip brand with highly optimized C compilers and embedded software development tools.

1

UNPACKING AND INSTALLATION

Inventory

- Use of this kit requires a PC with Windows 95, 98, ME, NT, 2000 or XP. The PC must have a spare 9-pin serial port, a USB port, a CD-ROM drive and 75MB of disk space.
- The diagram on the following page shows each component in the Robot Kit. Ensure every item is present.

Software Setup

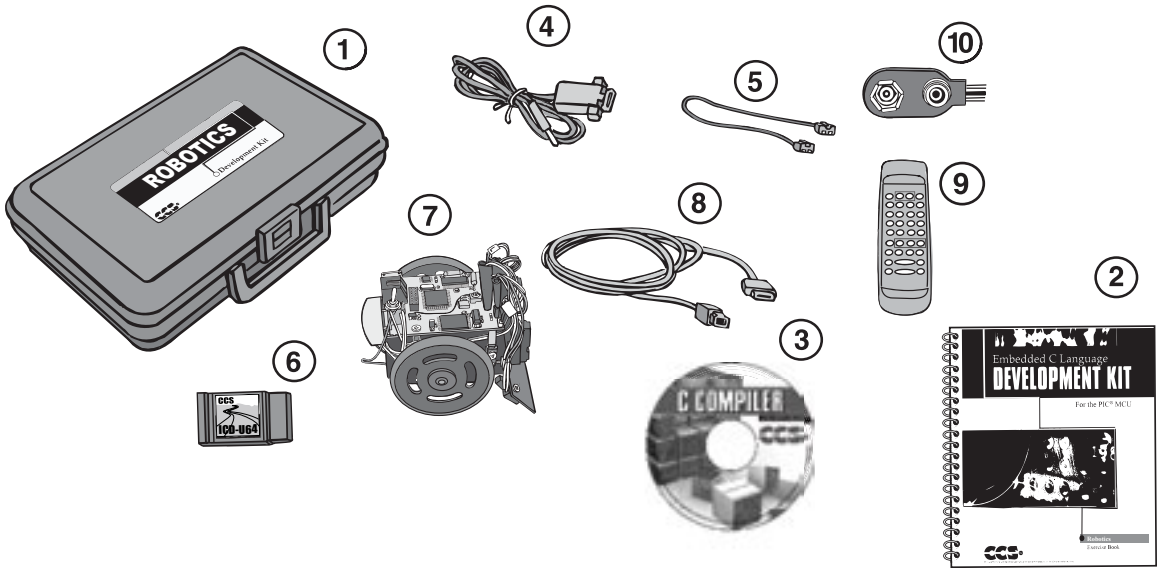
- Insert the CD into the CD-ROM drive and wait for the install program to start. If the computer is not set up to auto-run CDs, click on the **START** button and select **RUN**. Enter **D:\SETUP1.EXE** where D: is the drive letter for your CD-ROM drive.
- Click on **Install** and use the default settings for all subsequent prompts. Click **NEXT**, **OK**, **CONTINUE**, etc. as required.
- Double click the compiler icon on the desktop. In the PCW IDE, click **Help > About** and verify an IDE and PCM version number is shown; this ensures the software was correctly installed. Exit the IDE.

Hardware Setup

- Connect the PC to the ICD(6) using the USB cable.⁽¹⁾ Connect the prototyping board (10) to the ICD using the modular cable. The first time the ICD-U is connected to the PC Windows would detect new hardware. Install the USB driver from the CD or website using the new hardware wizard. The driver needs to be installed properly before the device can be used.
- The LED should be red⁽²⁾ on the ICD-U to indicate the unit is connected properly.
- Run the Programmer Control Software by clicking on the CCSLOAD icon on the desktop. Use CCSLOAD Help File for assistance.
- The software will auto-detect the programmer and target board and the LED should be illuminated green. If any errors are detected, go to Diagnostic tab. If all tests pass, the hardware is installed properly.
- Disconnect the hardware until the exercise in chapter four is reached. Always flip the power switch to the off position (away from the speaker) before connecting and disconnecting hardware.

⁽¹⁾ ICS-S40 can also be used in place of ICD-U. Connect it to an available serial port on the PC using the 9 pin serial cable. There is no driver required for S40.

⁽²⁾ ICD-U40 units will be dimly illuminated green and may blink while connecting.



- ① Carrying case
- ② Exercise Booklet
- ③ CD-ROM of Compiler Software (optional)
- ④ Serial PC to controller board Cable (RS-232)
- ⑤ Modular Cable (ICD to controller board)
- ⑥ ICD unit allows programming and debugging PIC® MCU parts from a PC
- ⑦ Robot Assembly Kit (see Chapter 2 for detailed parts list) with Robot Controller
- ⑧ Serial (or USB) PC to ICD Cable
- ⑨ Infrared Remote Control
- ⑩ Battery Snap

2

ROBOT ASSEMBLY

Parts list

- ☐ Ensure each of the following items is present.

*Note: See Appendix A for extra Robot Assembly Instructions

Quantity	Part Description
8	#4-40 .375" Phillips pan head sheet metal screw (pointed tip)
4	#6-32 .500" Phillips pan head machine screw (flat tip)
5	#4-40 .375" Phillips pan head machine screw (flat tip)
5	#4-40 Hex nut
4	#6-32 Hex nut
2	#4 Internal tooth lock washer
4	#4 Nylon washer
4	1.375" Standoff
1	Chassis body
1	Chassis scoop
1	9V Battery snap
1	4 AA Battery holder
2	2" Velcro hook strip
2	2" Velcro loop strip
2	Servo
2	Wheel
2	Rubber bands
3	Line sensor (QRB1134)
2	Proximity sensor (Sharp GP2D12)
2	Proximity sensor cable
1	Controller board
2	Extra Servo hardware

Step 1: Attaching the Line Sensors

Parts used

Quantity	Part Description
3	#4-40 .375" Phillips pan head machine screw (flat tip)
3	#4-40 Hex nut
3	Line sensor (QRB1134)
1	Chassis scoop

Assembly

Place a line sensor on the broad surface of the scoop with the flange pointing downward. Point the sensor away from the flange and attach it with a #4-40 .375" flat tip screw and #4-40 hex nut. The nut should be placed on the sensor side of the scoop. See **Figure 1**.

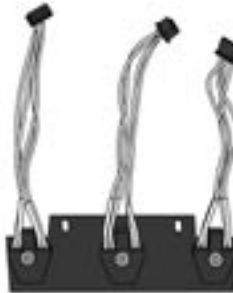


Figure 1

Step 2: Attaching the Proximity Sensors and Scoop

Parts used

Quantity	Part Description
2	#4-40 .375" Phillips pan head machine screw (flat tip)
2	#4-40 Hex nut
2	#4 Internal tooth lock washer
2	Proximity sensor (Sharp GP2D12)
1	Chassis body

2

ROBOT ASSEMBLY (CONT.)

Assembly

Route the wires for the center line sensor through the notch in the chassis body. Position the scoop against the small flange on the chassis body and orient so the broad portion points opposite of the flanges on the chassis body. Place a proximity sensor on the front side of the scoop with the white receptacle toward the center. Thread a #4-40 .375" flat tip screw through the proximity sensor, chassis scoop and chassis body. Fasten using a #4 internal tooth lock washer and #4-40 hex nut. Repeat for the second proximity sensor. See **Figure 2**.

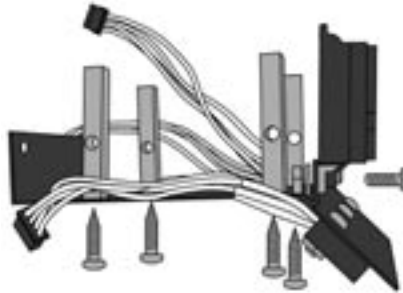


Figure 2

Step 3: Attaching the Standoffs

Parts used

Quantity	Part Description
2	#4-40 .375" Phillips pan head sheet metal screw (pointed tip)
2	1.375" Standoff

Assembly

Place the chassis body so the two flanges are pointing upwards. Examine one of the standoffs. Notice the hole through the side is closer to one end. This end will be attached to the chassis body. Orient the hole so it is *not* facing the flanges on the chassis body. Use four #4-40 .375" pointed tip sheet metal screws to attach the standoffs to the chassis body. See **Figure 3**.

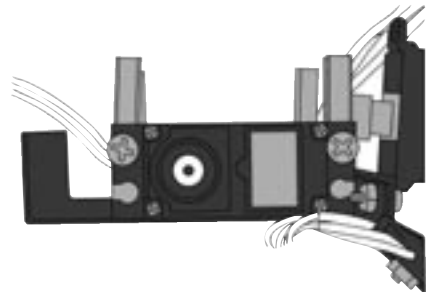


Figure 3

Step 4: Modifying the Servos

Servos are constructed to have a limited range of movement because they are designed to be used for ailerons, rudders, or steering. They must be modified for continuous rotation before attaching them to the robot.

First, remove and set aside the black screw and white wheel. Next, remove the four screws holding the case together. Carefully open the case by pressing on the drive gear and pulling up on the case. Take out the white potentiometer drive plate (see **Figure 4**) located on the underside of the drive gear (with bearing on top).

The plate and wheel will not be used in these exercise procedures; however, they may be useful in other applications. Cut the tab from the top of the drive gear until it is flush with the surface. One easy way to do this is to use a side cutter, as shown in **Figure 5**.



Figure 4

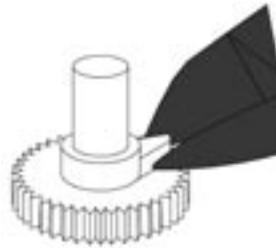


Figure 5

Step 5: Attaching the Servos

Parts used

Quantity	Part Description
4	#6-32 .500" Phillips pan head machine screw (flat tip)
4	#6-32 Hex nut
2	Servo

Assembly

Position the servos between the standoffs with the drive gear closer to the larger flange on the chassis body. Use the #6-32 .500" flat tip screws and #6-32 hex nuts to secure the servos to the standoffs. Point the screws toward the center of the robot. Do not pinch the wires for the center line sensor between the servos.

2

ROBOT ASSEMBLY (CONT.)

Step 6: Attaching the Batteries

Parts used

Quantity	Part Description
1	9V Battery snap
1	4 AA Battery holder
1	9V Battery (not included)
4	AA Battery (not included)
2	2" Velcro hook strip
2	2" Velcro loop strip

Assembly

Apply the two Velcro loop strips to the bottom of the chassis and the two Velcro hook strips to the 4 AA battery holder. Insert four AA batteries into the battery holder and attach it to the chassis body.

Attach the 9V battery snap to the 9V battery. Place the battery between the large flange on the chassis body and the rear standoffs.

CCS recommends using Nickel-Metal-Hydride, or NiMH, rechargeable batteries.

For more information on battery types, please visit this website:

<http://www.junun.org/MarkIII/Manual/Appendix.jsp>

Step 7: Attaching the Controller Board

Parts used

Quantity	Part Description
4	#4-40 .375" Phillips pan head sheet metal screw (pointed tip)
4	#4 Nylon washer
1	Controller board
2	Proximity sensor cable

Assembly

Set the controller board on top of the standoffs with the speaker facing the rear of the robot. Thread four #4-40 .375" pointed tip screws through the #4 nylon washers then fasten the controller board to the standoffs. Do not over tighten.

Refer to the diagram on the inside front cover to aid in the following cable connections: Plug the left, right, and center line sensors into their respective four pin receptacles. Plug the left and right proximity sensors into their corresponding three pin receptacles near the center of the controller board.

Route the Servo wires under the controller board and out the front of the robot. Plug the connector into the 3-pin header located on the same side as the Servo. The orange wire should be toward the speaker side of the robot.

Flip the switch away from the speaker into the OFF position. The four AA battery pack is attached to the 4-pin header near the speaker. Plug the wires into the two posts closest to the speaker with the brown ground wire on the middle post. Plug the 9V battery snap onto the other two posts with the brown ground wire on the middle post.

USING THE INTEGRATED DEVELOPMENT ENVIRONMENT

Editor

- Open the PCW IDE. If any files are open, click **File>Close All**.
- Click **File>Open>Source File**. Select the file: **c:\Program Files\PICC\Examples\Ex_stwt.c**.
- Scroll down to the bottom of this file. Notice the editor displays comments, preprocessor directives and C keywords in different colors.
- Move the cursor over the **Set_timer0** and click. Press the F1 key. Notice a help file description for **set_timer0** appears. The cursor may be placed on any keyword or built-in function and F1 will find help for the item.
- Review the editor special functions by clicking on **Edit**. The IDE allows various standard cut, paste and copy functions.
- Review the editor's option settings by clicking on **Options/Editor Properties**. The IDE allows selection of the tab size, editor colors, font and many more. Click on **Options/Customize** to select which icons appear on the toolbars.

Compiler

- Use the drop-down box under **Compile** to select the compiler. CCS offers different compilers for each family of Microchip parts. All the exercise in this booklet are for the PIC16F877A chip, a 14-bit opcode part. Make sure **PCM 14 bit** is selected in the drop-down box under the **Compiler** tab.
- The current file is always shown in the bottom of the IDE. If the desired file to compile is not shown, click on the tab of that particular file. Right click on the editor and select **Make file project**.
- Click **Options>Project Options>IncludeFiles** and review the list of directories the compiler uses to search for included files. The install program should have put two directories in this list: **devices** and **drivers**.
- Normally, the file formats need not be changed and global defines are not used in these exercises. To review these settings, click **Options>Project Options>Output Files** and **Options>Project Options>Global Defines**.
- Click **Compile>Compile** or the compile icon to compile. Notice the compilation box shows the files created and the amount of ROM and RAM used by this program. Press any key to remove the compilation box.

Viewer

- ❑ Click **Compile>Symbol Map**. This file shows how the RAM in the microcontroller is used. Identifiers that start with @ are compiler generated variables. Notice some locations are used by more than one item. This is because those variables are not active at the same time.
- ❑ Click **Copile>C/ASM list**. This file shows the original C code and the assembly code generated for the C. Scroll down to the line:

```
int_count=INTS_PER_SECOND;
```

- ❑ Notice there are two assembly instructions generated. The first loads 4C into the W register. INTS_PER_SECOND is #defined in the file to 76. 4C hex is 76 decimal. The second instruction moves W into memory. Switch to the Symbol Map to find the memory location is where int_count is located.
- ❑ Click **View>Data Sheet**, then View. This brings up the Microchip data sheet for the microcontroller being used in the current project.

Click here for the file menu. Files and Projects are created, opened, or closed using this menu.

Compiles current selected unit, does NOT link/build into a HEX file.

Compiles all units that have changed since last build, links/builds into a HEX file.

Compiles all units regardless if they have changed since last build, links/builds into a HEX file.

Compile ribbon.

Place cursor over each icon and press F1 for help.

Click the help icon for the help menu. The technical support wizard and download manager are accessed using this menu.

Quick view of supported devices.

Place cursor here for slide out boxes. All of the current project's source and output files can be seen here.

The screenshot shows the MPLAB IDE interface. The top menu bar includes File, Edit, Search, Options, Compile, View, Tools, Debug, Document, and User Tools. The toolbar contains icons for Compile, Build, Build All, Clean, LookUpPart, Program Chip, Debug, C/ASM List, Symbol Map, Call Tree, and Help. The main window displays the C source code for 'ex_extseg.c' with assembly instructions generated for the line 'int_count=INTS_PER_SECOND;'. The left sidebar shows the Project Explorer with folders for Files, Sources, and Output. The right sidebar shows the Help menu with options like Contents, Index, Error at cursor, Debugger Help, Editor, Data types, Operators, Statements, Emprocessor cmds, Built in functions, Technical Support, Check for software updates, Internet, and About.

4

PROGRAMMING

- Open the PCW IDE. If there are any open files, click **File > Close All**.
- Click **File > New > Source File** and enter the filename `ex4.c`.
- Enter the following source code then compile.

```
#include <16F877A.h>
#fuses HS,NOLVP,NOWDT,NOPROTECT,NOBROWNOUT,PUT
#use delay(clock=10000000)

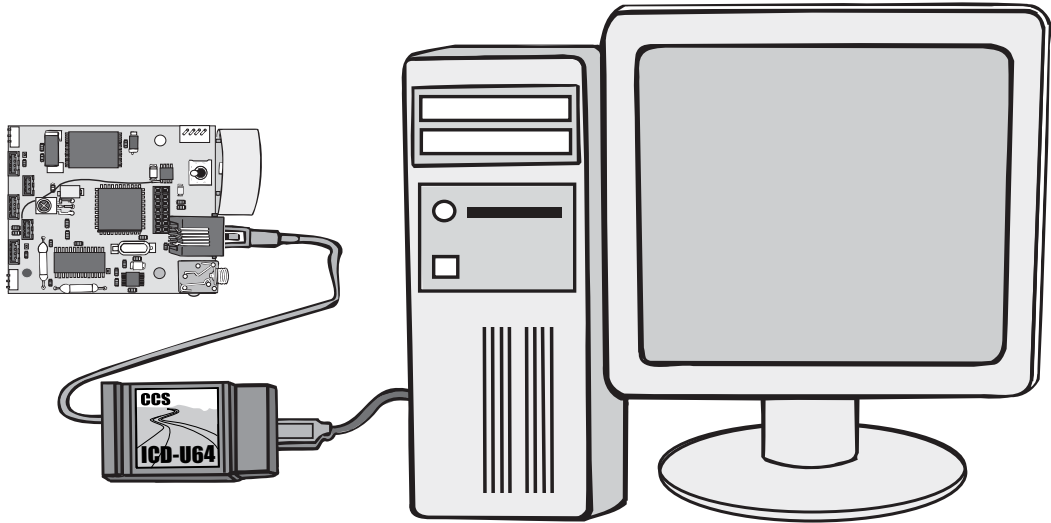
#include <wts701.c>

void main() {
    char text[] = "Hello world";
    tts_init();

    while(TRUE) {
        tts_sendText(text);
        delay_ms(5000);
    }
}
```

NOTES

- The first three lines of the source code define the hardware environment. The microcontroller being used is the PIC16F877A running at 10MHz. Click on **View > Valid Fuses** to read about the different fuse settings. Fuses control the microcontroller's configuration word.
- The line `#include <wts701.c>` includes the drivers for using the text to speech converter. This chip is described in detail in Chapter 11.
- The statement `while(TRUE)` is a simple way to create a loop that never stops. Infinite loops are very common in main for embedded systems.
- The statement `delay_ms(5000);` is a five second delay (5000 ms).



- Connect the ICD to the Prototyping board using the modular cable, and connect the ICD to the PC. Power up the Prototyping board.
- Click on **Tools > ICD** to download the program to the controller board. Once completed, the robot should say “Hello World” every five seconds. Power down the robot.
- Highlight the first three lines of source code then click **Edit > Paste to file**. Name the file robot.h. This header file is used in the remaining example programs.

5

DEBUGGING











- ❑ Open robot.h and insert `#device ICD=TRUE` on the line after `#include <16F877A.h>` to compile in debug mode.
- ❑ Create a file called ex5.c, type in the following source code. Right click in the editor and select **Make file project**, then compile.






```
#include "robot.h"

int8 sum(int8 a, int8 b) {
    return a+b;
}

void main() {
    int8 x = 2, y = 3;

    while(TRUE) {
        x = sum(x, y);
    }
}
```

- ❑ Start the debugger by clicking **Debug > Enable Debugger**. After the program is loaded onto the controller board, click the step over  icon until the yellow arrow passes `x = sum(x, y)`. Each click causes a line of code to be executed. The line with the arrow is next to the executed but has not done so. Clicking the step over  icon on `x = sum(x, y)` caused the entire function to be executed in one click. The debugger stepped over the function.
- ❑ Now click the single step  icon twice. The arrow should point at `return a+b`. The single step  icon caused the debugger to step into the function. Press single step  icon a few more times to return to main.
- ❑ Click the Watches tab, then the plus  icon to add a watch. Enter `x` or choose `x` from the list of variables and click Add Watch. The current value of `x` is shown. Continue to press the step over  and single step  icons to watch the value of `x` change. Notice how the value of `x` is not displayed when inside the `sum()` function because it is not available in the source code at this time.
- ❑ Click the  icon to allow the program to run normally. Click the stop  icon to halt execution. The debugger arrow will point to where the program was halted.

- ❑ In the editor, click on return a+b to move the cursor to that line. Click the Breaks tab and click the plus  icon to set a breakpoint. The program will be halted every time this line of code is reached. Click the  icon. The debugger will stop at the breakpoint. Practice setting breakpoints at different locations to learn how they work.
- ❑ Click **Compile > C/ASM List**. Find the line with the debugger arrow. Notice one assembly instruction was already executed and the arrow has passed the breakpoint. This is a side effect of the debugger. Sometimes breakpoints slip by one ASM instruction.
- ❑ Click the step over  and single step  icons a few times. Notice that the debugger is stepping through one assembly instruction per click instead of one entire C line.
- ❑ Change return a+b to return a-b and recompile. Step over the call to sum and examine the value of x. The int data type by default is not signed, so x cannot be the expected -1. The modular arithmetic works like a car odometer in reverse, only in binary. For example, 00000001 minus 1 is 00000000; subtract another 1 to get 11111111, or decimal 255.
- ❑ Press the reset icon and step up to `x = sum(x, y)`. Click the Eval tab. This pane allows a one-time expression evaluation. Type in `x+y` and click Eval for the debugger to calculate the result. The complete expression may also be put in the watches pane as well. Now enter `y=1` and click Eval. This expression will change the value of y if the “Keep side effects” checkbox is checked. Check this box and click Eval again. Click the Watches tab then step over the call to sum to verify the value of x was calculated with new y value.
- ❑ Set a break point at `x = sum(x, y)` then click the Break Log tab. Check the Log checkbox, make sure break 1 is selected, and enter x in the edit box. Press the  icon. Each time the breakpoint is reached, the debugger will retrieve the value of x, add it to the log, and continue execution.
- ❑ Remove `#device ICD=TRUE` from robot.h before continuing onto the remaining exercises.

6

BASIC MOVEMENT

- ❑ The robot is equipped with two Servos that individually drive the left and right wheels. This allows for very tight turns and a variety of speeds. The Servos are controlled by sending a high pulse every 20ms. The length of the pulse tells the Servo how fast it should move in a particular direction. Pulses range from 0.9ms to 2.1ms with 1.5ms as center. The Servos are calibrated with the center as stop. They can be spun in either direction by adjusting the pulse width to either side of center. The Servos must be calibrated before they are used.
- ❑ Create `calibrate_servos.c` and enter in the following source code:

```
#include "robot.h"

#define CALIBRATE_SERVOS
#include <servos.c>

void main() {
    init_servos();
    for(;;)      // Loop forever
}
```

- ❑ Compile the program and download it to the controller board by clicking **Tools > ICD**. Once the program is running, the Servos should start to spin. Using your fingers, turn the potentiometer until the motors stop spinning. Reassemble the Servos.
- ❑ Attach the wheels by aligning them with the teeth on the drive gear and pressing them on. Use the small black screws from the Servo parts to secure the wheels to the Servos. The robot is now able to move under its own power.
- ❑ Create a new file called `ex6.c` and type in the following source code to control the robot's movement:


```

#include "robot.h"
#include <servos.c>

void main() {
    init_servos();           // Initialize servo control
    delay_ms(3000);         // Wait 3 seconds before moving

    for(;;) {
        set_servo(LEFT, FORWARD, 4); // Go forward at maximum speed
        set_servo(RIGHT, FORWARD, 4);
        delay_ms(3000);

        set_servo(LEFT, BACKWARD, 4); // Spin in a tight circle
        delay_ms(2000);

        set_servo(LEFT, FORWARD, 0); // Stop for 1 second
        set_servo(RIGHT, FORWARD, 0);
        delay_ms(1000);

        set_servo(LEFT, BACKWARD, 1); // Go backward at a slow speed
        set_servo(RIGHT, BACKWARD, 1);
        delay_ms(5000);

        set_servo(LEFT, FORWARD, 1); // Turn in a large circle
        set_servo(RIGHT, FORWARD, 3);
        delay_ms(4000);

    }
}

```

- Compile the program then download to the controller board by clicking **Tools > ICD**. Once the program is finished downloading, turn off the robot. Unplug the ICD cable from the controller board and set the robot on the floor in an open area. Turn the robot back on and watch it move.
- Servos.c contains the functions needed to setup and control the Servos. Right click on servos.c and click Open "servos.c". Scroll down to init_servos(). The init_servos() function performs the necessary setup. It starts by configuring how timer one increments then sets CCP1 and CCP2 to trigger an interrupt when their respective registers match timer one. The left and right adjustments are set to zero to halt the Servos; interrupts are then enabled.
- The next function, set_servo(), is used to control the Servo speeds. It selects an adjustment from the servo_speeds look up table. The interrupt service routines use this adjustment as an offset from the center pulse width to spin a Servo in either direction.
- Experiment controlling the robot's movement by creating a new series of movement instructions in **EX6.C**.

- ❑ RS-232 is a popular communication protocol used on most PCs and many embedded systems. Two signal wires transmit and receive data while a third wire is used for ground. The PIC16F877A has built in hardware to buffer serial data when using pin C6 for transmitting and C7 for receiving. The compiler is able to use any pins, but will take advantage of the built in hardware when using C6 and C7.
- ❑ Add the following line of code at the end of robot.h to enable RS-232 communication:
`#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)`
- ❑ The `#use rs232` directive enables the following functions:
 - `void putc (char c);`
 - `void printf (char* s);`
 - `char getc();`
 - `int1 kbit();`
- ❑ Use `putc()` and `getc()` to send and receive a single character over the RS-232 connection, respectively. The use of `printf()` calls `putc()` multiple times to send multiple characters. Use `kbit()` to test if a character is sitting in the receive buffer. Read more about these functions and their derivatives in the help or reference manual included with the compiler.
- ❑ Create a new source file called `ex7.c` and type in the following source code:

```
#include "robot.h"
#include <servos.c>

void main() {
    init_servos(); // Initialize servo control

    printf("\n\r(F)orward, (B)ackward, (L)eft, (R)ight, (S)top\n\r");

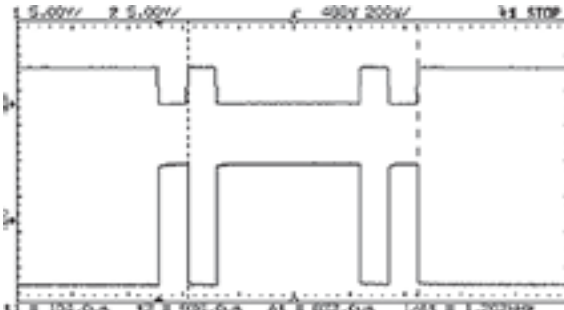
    for(;;) {
        switch(toupper(getc())) {
            case 'F': // Move the robot forward
                set_servo(LEFT, FORWARD, 4);
                set_servo(RIGHT, FORWARD, 4);
                break;
            case 'B': // Move the robot backward
                set_servo(LEFT, BACKWARD, 4);
                set_servo(RIGHT, BACKWARD, 4);
                break;
            case 'L': // Turn the robot left
                set_servo(LEFT, BACKWARD, 4);
                set_servo(RIGHT, FORWARD, 4);
                break;
        }
    }
}

(continued...)
```

```

(continued...)
    case 'R':                                // Turn the robot right
        set_servo(LEFT, FORWARD, 4);
        set_servo(RIGHT, BACKWARD, 4);
        break;
    case 'S':                                // Stop
        set_servo(LEFT, FORWARD, 0);
        set_servo(RIGHT, FORWARD, 0);
        break;
}
}
}

```

- ❑ Compile the program and load it to the controller board. Disconnect the robot from the ICD. Connect the robot to the PC with the 9-pin serial cable. Click **Tools > Serial Port Monitor** within the PCW IDE. Configure the COMM port by clicking **Configuration > Set port options**. Set the baud rate to 9600, parity to none, data bits to 8, stop bits to 1, and flow control to none. Make sure the correct COMM port is selected. Set the robot in an open area on the floor. The keyboard and serial cable now act as a tethered remote control. Practice controlling the robot by pressing keys that correspond to a direction.
- ❑ RS-232 sends a series of bits at the hardware level. The baud= option specifies how many bits are sent per second. The bit stream as specified above is a start bit (always 0), 8 data bits (lsb first) and a stop bit (always 1). The data line then remains at the logic 1 level. The number of bits may be changed with a bits= option. A 0 is represented as a positive voltage (+3V to +12V) and a 1 is represented as a negative voltage (-3V to -12V). Since the PIC16F877A outputs only 0V and 5V, a level converter is required to interface to standard RS-232 devices such as a PC. A popular converter chip is the MAX232. See the schematic in the back cover for details. The diagram above shows a single character 'A' (01000001) sent at 9600 baud. The top waveform is from the PIC16F877A, the bottom waveform is from the MAX232, and the 8 data bits are between the dotted lines. Each bit is 104us.
 
- ❑ See input.c in the Drivers directory and ex_float.c, ex_exsio.c and ex_sisr.c in the examples directory for further information on using RS-232.

- ❑ Create file called ex8.c and type in the following source code:

```
#include "robot.h"
#include <servos.c>
#include <stdlib.h>
#include <input.c>

struct {
    struct servo {
        int8 speed, dir;
    } l, r;
    int8 time;
} cfgs[10];

void main() {
    init_servos();

    for(;;) {
        int8 i, j, time;

        for(i=0;i<10; ++i)
        {
            do {
                printf("\n\n\rLeft speed: ");
                cfgs[i].l.speed = get_int();
            } while (cfgs[i].l.speed > 4);

            do {
                printf("\n\n\rLeft directions (F, B): ");
                cfgs[i].l.dir=getc();
            } while((cfgs[i].l.dir != 'F') && (cfgs[i].l.dir != 'B'));

            if(cfgs[i].l.dir == 'F')
                cfgs[i].l.dir = 0;

            do {
                printf("\n\n\rRight speed: ");
                cfgs[i].r.speed = get_int();
            } while (cfgs[i].r.speed > 4);

            do {
                printf("\n\n\rRight direction (F, B): ");
                cfgs[i].r.dir = getc();
            } while((cfgs[i].r.dir != 'F') && (cfgs[i].r.dir != 'B'));
```

(continued...)

(...continued)

```
if(cfgs[i].r.dir == 'F' )
    cfgs[i].r.dir = 0;

    printf("\n\rTime in s: ");
    time = get_int();
}

for(i=0; i<10; ++i)
{
    set_servo(LEFT, cfgs[i].l.dir, cfgs[i].l.speed);
    set_servo(RIGHT, cfgs[i].r.dir, cfgs[i].r.speed);

    for(j=0; j < time; ++j) {
        delay_ms(1000);
    }
}
stop_servos();
}
```

- Compile the program and download it to the controller board.
- The program stores 10 different Servo configurations. Each configuration controls the speed, direction and duration of a turn. Once all 10 settings are completed, they are played back.
- This example makes use of structures (structs). A struct is a way to group a common data set. For instance, the struct Servo contains all the information needed to control a Servo. Data inside a struct is allocated contiguously in RAM.

9

PROXIMITY SENSING

- ❑ An important attribute of a moving robot is being able to detect if an obstacle is in its path. Avoiding objects can prevent damage to the robot or the obstruction. For example, delivering robots in hospitals use similar sensors to avoid bumping into people. The sensors included in the kit are able to detect objects from 5cm to 28cm away. They contain an infrared (IR) LED and an IR collector. The LED is repeatedly pulsed to emit a signal. The collector circuit detects the IR signal when reflected off of objects. The received signal quality is translated into an analog voltage. Using the analog to digital converter (ADC), the robot can detect an obstacle and measure its distance away.
- ❑ Create ex9.c and type in the following source code:

```
#include "robot.h"
#include <GP2D12.c>
#include <servos.c>

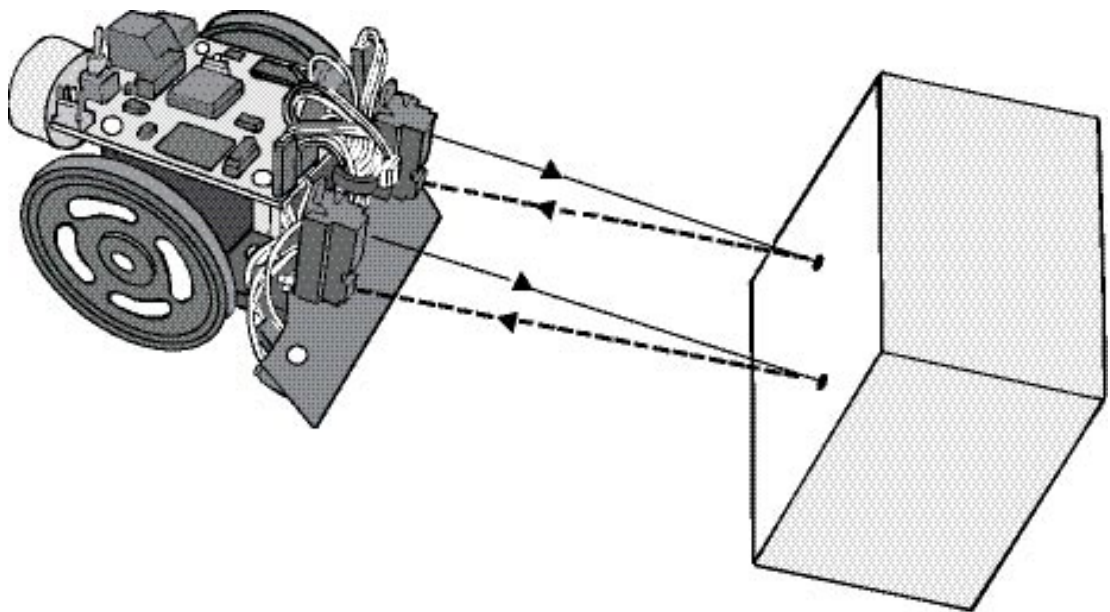
void main() {
    init_servos();
    init_objectSensors();

    for(;;) {
        int8 leftSensor, rightSensor;

        read_ObjectSensors(leftSensor, rightSensor);

        if(leftSensor > 100 || rightSensor > 100) {
            set_servo(LEFT, FORWARD, 0);
            set_servo(RIGHT, FORWARD, 0);
        } else if(leftSensor < 80 && rightSensor < 80) {
            set_servo(LEFT, FORWARD, 4);
            set_servo(RIGHT, FORWARD, 4);
        }
    }
}
```

- ❑ Compile the program and download it to the controller board. Set the robot on the floor and point it toward a nearby obstacle. When the robot becomes close enough, it will sense the increased voltage from the sensors and stop. The sensors function best when used with white reflective objects rather than dark non-reflective objects.
- ❑ In the source code, the Servo speeds are not altered in the region between 80 and 100. This creates a buffer zone in front of the robot to prevent the motors from twitching as the ADC reading fluctuates above and below 100.
- ❑ Right click on GP2D12.c and select Open. Scroll down to `init_objectSensors()`. This function configures the ADC to use the internal clock and sets which analog capable pins are analog. The ADC on the PIC16F877A is capable of both 8 and 10 bit conversions. The ADC resolution can be configured by inserting `#device ADC=8` or `ADC=10` on its own line after `#include <16F877A.h>` in `robot.h`.



10

LINE FOLLOWING

- ❑ The line sensors on the bottom front of the robot can be used for a variety of purposes. They can be used to follow lines, detect boundaries, or even read a change in reflection to navigate a robot soccer field. These sensors are similar to the proximity sensors. Each one contains an infrared LED and a phototransistor, but no oscillating circuit. The phototransistor reacts to IR emissions from the LED only when a reflective object is placed very close to the sensor. It reacts by adjusting the current flow through the sensor. Since the microcontroller cannot measure current, the output voltage is connected to ground through a resistor; this creates a collector amplifier. The voltage level is measured by the analog to digital converter (ADC).
- ❑ Create a file called `ex10.c` and type in the following source code:

```
#include "robot.h"
#include <line_tracker.c>
#include <servos.c>

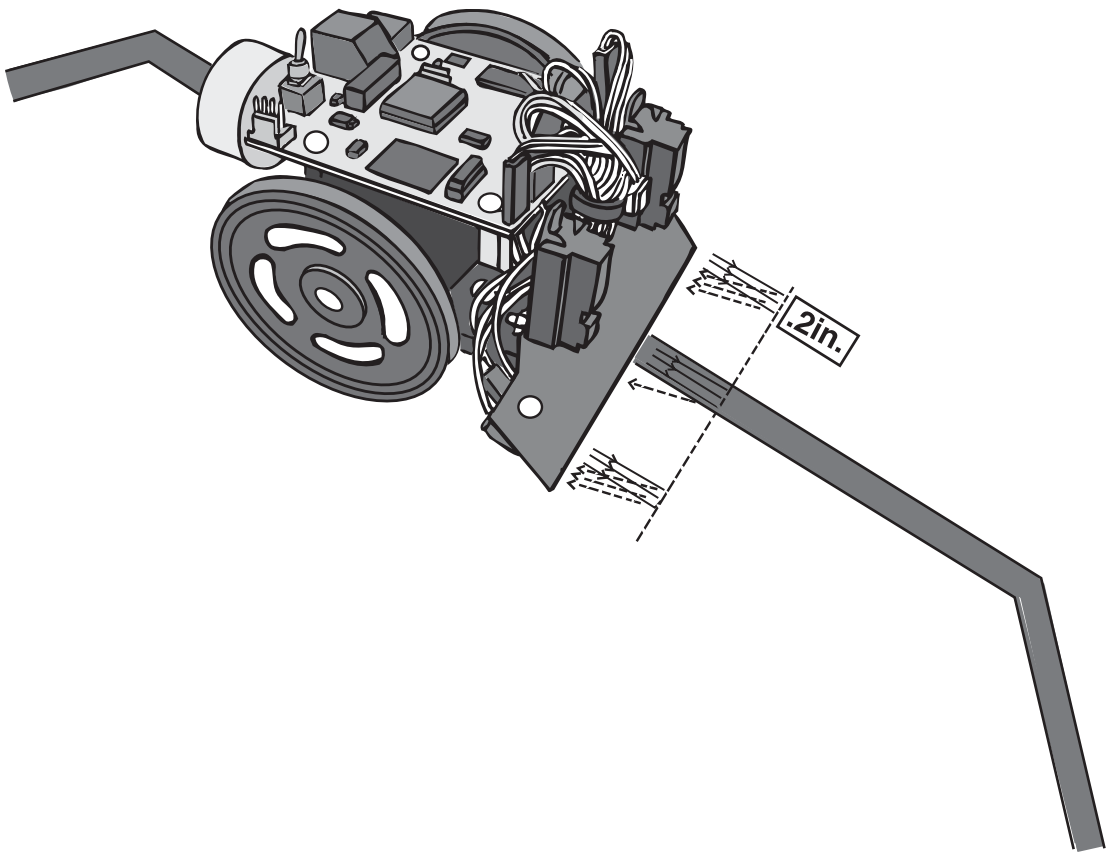
void main() {
    init_servos();
    init_lineTracker();

    for(;;) {
        int l, r;

        // Check if sensors are over black or white
        l = lt_check(LT_LEFT);
        r = lt_check(LT_RIGHT);


        // Check if it should turn right, left, or go straight
        if(l && !r) {
            set_servo(LEFT, FORWARD, 0);
            set_servo(RIGHT, FORWARD, 3);
        } else if(r && !l) {
            set_servo(RIGHT, FORWARD, 0);
            set_servo(LEFT, FORWARD, 3);
        } else {
            set_servo(RIGHT, FORWARD, 3);
            set_servo(LEFT, FORWARD, 3);
        }
    }
}
```


- ❑ Compile the program and download it to the controller board. Create a circular course for the robot to follow by making a black line on a light colored surface. A great way to make the line is to use black electrical tape. After creating the course, set the robot over the line and turn it on. As it encounters a corner, it will stop one motor to turn and follow the line. This is a very simple example that does not use the center sensor; therefore, the robot may have troubles on tight corners.
- ❑ As an exercise, create a new program that uses the center sensor to make smarter turns. For example, turn left quickly when the center and left sensor are both over the line and slowly when only the left sensor is over the line.



- RS-232 printf statements can be a good tool to help debug a program. It does, however, require an extra hardware setup to use. If the ICD is being used as a debug tool, the compiler can direct `putc()` and `getc()` through the debugger interface to the debugger screen. Add the following line to `ex7.c`:

```
#use rs232 (DEBUGGER)
```

- Compile and load the program into the Prototyping board.
- Click GO, then click the **Monitor** tab.
- A prompt should appear. Enter some data to confirm that the program is working.
- Stop and reset the program.
- Click the debugger **Break Log** tab, check the LOG box, set the breakpoint as 1 and expression as `L`. Result is the value of the number being converted.
- Click GO, then click.
- The **Log** tab and notice that each time the breakpoint was hit the value of the `L` variable was logged. In this case the breakpoint did not cause a full stop of the program, it just logged the value of the requested expression and kept on going.
- Stop the program.
- Uncheck the LOG box under the log tab.
- Enter watches for `L` and `r`.
- Click GO.
- When the break is reached click on the snapshot icon: 
- Check **Time** and **Watches**, uncheck everything else.
- If a printer is connected to the PC select **Printer**, otherwise select **Unique file**.
- Click on the **Now** button.
- Notice the requested data (time and watches) are either printed or written to a file as requested.
- Click on the snapshot icon again and this time select **Append to file**, put in a filename of `EX11.TXT` and check **After each single step**.
- Check **Last C line executed** in addition to the **Time** and **Watch** selected already and close the snapshot window.
- Reset and then Step Over for a while.
- Use **File>Open>Any File** to find the file `EX11.TXT` (by default in the Debugger Profiles directory) after setting the file type to all files.
- Notice the log of what happened with each step over command.
- Uncheck the **After each single step** in the snapshot window.

- When the break is reached click on the **Peripherals** tab and select Timer 0.
- Shown will be the registers associated with timer 0. Although this program does not use timer 0 the timer is always running so there is a value in the **TMR0** register. Write this value down.
- Clear the breakpoints and set a new breakpoint.
- Click GO.
- Check the **TMR0** register again. If the new value is higher than the previous value then subtract the previous value from the current value. Otherwise, add 256 to the current value and then subtract the previous value (because the timer flipped over).
- The number we now have is the number of clock ticks it took to execute the switch and addition. A clock tick by default is 0.2ms. Multiply your number of ticks by 0.2 to find the time in ms. Note that the timers (and all peripherals) are frozen as soon as the program stops running.

FURTHER STUDY

- A The debugger **Eval** tab can be used to evaluate a C expression. This includes assignments. Set a break before the switch statement and use the Eval window to change the operator being used. For example, type a + but change it to a - before the switch.*
- B Set a break on the switch statement and when reached, change to the C/ASM view and single step through the switch statement. Look up the instructions executed in the PIC18F4520 data sheet to see how the switch statement is implemented. This implementation is dependent on the case items being close to each other. Change * to ~ and then see how the implementation changes.*

- ❑ One way for a robot to interact with the people around it is to emit sounds ranging from simple beeps to understandable speech. The robot is equipped with a text to speech converter and a speaker.
- ❑ Create a new file called `ex11.c` and type in the following code:

```
#include "robot.h"
#include <wts701.c>
#include <stdlib.h>
#include <input.c>

void main() {
    char key = 0;
    char text[50];
    text[0] = '\\0';

    tts_init(); // Initialize the text to speech chip

    for(;;) {
        switch(key) {
            case 'E':
                printf("Text: ");
                get_string(text, sizeof(text)); // Enter text to speak
            case 'R':
                tts_sendText(text); // Repeat the text
                break;
            case 'P':
                printf("Set pitch (0-6): ");
                tts_setPitch(get_int()); // Set the pitch
                break;
            case 'S':
                printf("Set speed (0-4): ");
                tts_setSpeed(get_int()); // Set the speed
                break;
        }
    }
}
```

(continued...)

(continued...)

```
        case 'V':
            printf("Set volume (0-7): ");
            tts_setVolume(get_int());           // Set the volume
            break;
        default:
            printf("(E)nter text, (R)peat, (P)itch, (S)peed,
                (V)olume");
            break;
    }
    printf("\n\nr");
    key = toupper(getc());
}
}
```

- Connect the RS-232 port to the PC and open the serial port monitor. Compile the program then load it into the controller board. Press 'E' and enter some text. The text is sent to the text to speech converter using an SPI protocol.
- The Winbond text-to-speech converter offers an extensive set of commands. Speech playback can be paused, resumed or stopped and the volume, pitch, or speed can be altered to change the sound.
- Text is converted in a few different ways. Most words are spoken exactly as they are entered, but the chip also responds to abbreviations. For example, "Mar" is pronounced as "March" and "MN" as "Minnesota." Numbers like "23" are spoken as "twenty-three", but larger numbers like "94087" are spoken as "nine four zero eight seven." There is a limited amount of space for users to program their own abbreviations. Once an abbreviation has been written, it cannot be erased without reprogramming the entire chip. However, they can be deactivated. Read through the WTS701 user's manual at <http://www.winbond-usa.com/> for an extensive list of abbreviations and in text control commands.
- Open the driver, wts701.c, and review the functions described in the comments at the top of the file. Experiment with some of the functions to change the playback and speech sound.

- ❑ The robot is also equipped with an electronic compass. Similar to the line tracking sensors, the compass offers a way to guide the moving robot. However, the compass avoids the limitations of the line by allowing the robot to move freely and still know where it is heading.
- ❑ The electronic compass on the robot is composed of an integrated circuit (IC), two SEN-L inductors and two resistors. The IC measures the amount of time it takes to send a series of pulses through an inductor/resistor circuit. The signal is generated in both directions to achieve a zero centered magnetic field measurement. Once calibrated, taking the arctangent of this measurement on two different axes creates a compass.
- ❑ Calibration is a critical step before using the compass. Local magnetic interference, such as the speaker magnet or moving Servos, renders the compass useless without calibration. The compass is calibrated by taking a reading for north and a reading for south, then averaging the component vectors. Another step to fine-tune the calibration is to enter the number of degrees of declination for your area. This information can be found at <http://www.ngdc.noaa.gov/seg/geomag/jsp/Declination.jsp> Declination is the number of degrees between magnetic north and true north.
- ❑ Create ex12.c and type in the following source code:

```
#include "robot.h"
#include <compass.c>

void main() {
    output_high(PIN_C0); // Disable the select line for the text to
                        // speech chip to prevent interference
    compass_init();
    compass_calibrate();

    for(;;) {
        printf("Angle: %lu\n\r", compass_getAngle());
        delay_ms(500);
    }
}
```

- ❑ Connect the ICD and RS-232 cable to the robot and run the serial port monitor. Compile the program and download it to the controller board. Follow the instructions to calibrate the compass. If you do not wish to use degrees of declination in your calibration, simply enter 0 when prompted. For the best results, perform the calibration on a flat surface with the robot a few feet away from the computer monitor. The monitor emits a changing magnetic field, which will distort the calibration values. After calibrating, the current heading is displayed in degrees; North is 0, East is 270, South is 180, and West is 90 degrees.
- ❑ Open `compass.c` and scroll down to end of `compass_calibrate()`. This function makes use of the internal EEPROM on the PIC16F877A. EEPROM is memory that retains its information when disconnected from power. However, data can be written a limited number of times. The EEPROM on the PIC16F877A allows roughly 1 million writes per address. Once the calibration values are calculated, they are stored in EEPROM. The function `compass_init()` reads these values upon startup, so they only have to be calculated once.
- ❑ Comment out the `compass_calibrate()` and recompile. Open `ICD.exe` located in the installation directory. Click on **Advanced....** Under Erase Modes, select **Erase when needed**. Press OK and exit the ICD software. Now when a program is downloaded to the controller board, only a memory area that needs to be programmed will be erased. For this example, the flash area will be erased and the data EEPROM will remain intact. This is how the calibration values are saved when downloading new firmware. Download the new program by clicking **Tools > ICD**. It should behave exactly the same as before.

14

IR PROTOCOL

- ❑ Included in the robot kit is a TV/VCR remote control. When a button is pressed, it emits a signal via the IR LED. Almost every remote manufacturer defines a different communications protocol for their devices to prevent interference with another device. For example, a television should not turn off each time a DVD player is turned on with a remote..
- ❑ The IR remote protocol uses a carrier frequency of approximately 36kHz to emit a Manchester encoded signal. 0s and 1s cannot be sent simply by assigning LED on to 1 and LED off to 0. False triggers from ambient light would cause unwanted commands to be received. Instead, the LED is pulsed at the carrier frequency for a certain length of time. The IR receiver contains a band pass filter designed to only accept a signal at this frequency and the receiving system measures the time. To switch between sending high and low, the carrier frequency signal is modulated.
- ❑ Manchester encoding works by starting with a known bit value, 0 or 1. The value is repeated by sending a square wave at a certain frequency. To change the value of the bit, the amount of low or high time is extended by half a period to change the phase of the wave. The square wave is repeated again until a change in the bit's value is needed.
- ❑ **Figure 6** shows the carrier frequency from the remote control on top and the demodulated signal sent to the microcontroller from the receiver on the bottom. When the signal is present, as shown on the left side of the graph, the output to the microcontroller is low. The output is high when the signal is completely modulated, as shown on the right side. Notice the delay of roughly 200us between the start of a bit period and the output change from the sensor. This is another protection to avoid sending false information by ensuring that a signal is actually present before making a change.

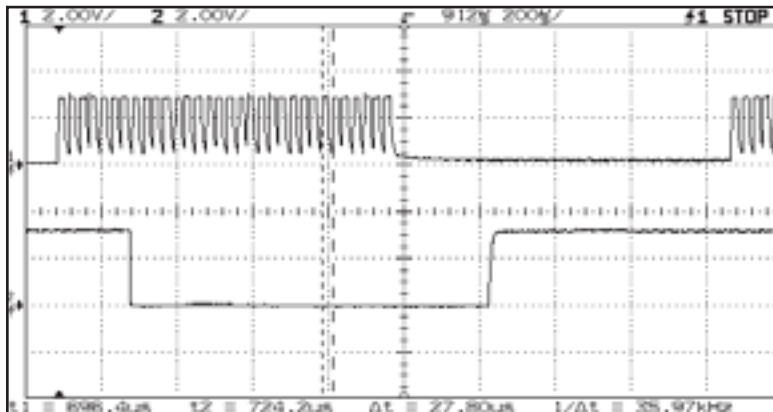


Figure 6

- Figure 7** shows the entire Manchester encoded power command sent to the VCR. Follow the wave on the bottom to learn how the signal is decoded. The first low and high pulses are of equal length meaning they represent one bit. As part of the protocol, this first bit has the value of one. The next long low pulse means the value of the bit has changed to a zero. The next two high and low pulses are short meaning the value of the bit has remained the same. Thus far, 1000 has been decoded in binary. The next four pulses are all long, representing 1010. Follow the remaining pulses to see that 1000101001100 was received.

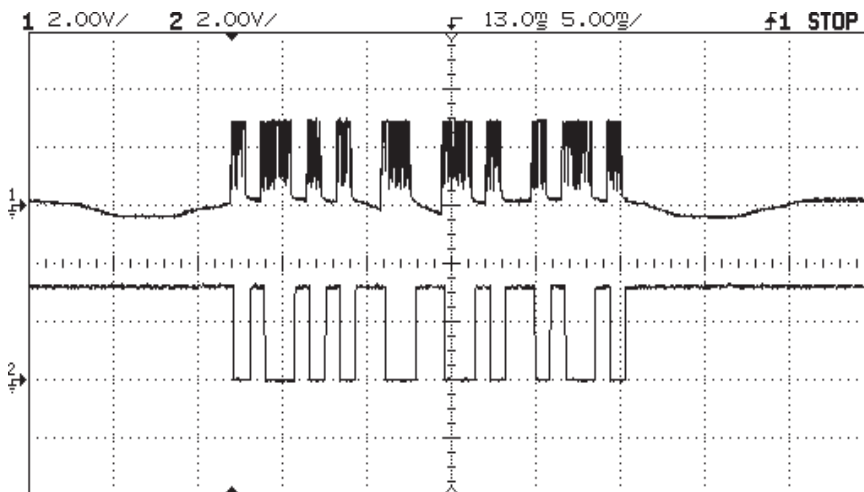


Figure 7

- The transmission time for one command is slightly longer than 23ms and commands are repeatedly sent every 114ms. By the time a button is pressed and released, the same command is sent multiple times. This makes a remote easier to use if the first command was received incorrectly because it does not force the user to repress the button. However, it could cause the same command to be received multiple times. The IR protocol for the remote included in the kit tries to prevent the problem by alternating the second bit between 1 and 0. Thus, the other form of the power command is 1100101001100. The receiving system can monitor this bit and only respond to new commands, like changing the power state, or ignore the bit and respond multiple times, such as turning the volume up.

- ❑ Now that the principle behind IR communication has been explained, it will be used to untether the robot from the computer while still allowing external control.
- ❑ Create a file called `ex14.c` and type in the following source code:

```
#include "robot.h"
#include <N9085UD.c>
#include <servos.c>

void main() {
    int8 button;

    init_servos();
    init_IRremote();

    for(;;) {
        readIRremote(button, TRUE);           // Read a button press

        if(button < VCR_FIVE) {              // Set the backward speed
            set_servo(LEFT, BACKWARD, button); // to 0, 1, 2, 3, or 4
            set_servo(RIGHT, BACKWARD, button);
        }

        switch(button) {
            case VCR_STOP:
                set_servo(LEFT, FORWARD, 0);   // Press stop to stop
                set_servo(RIGHT, FORWARD, 0);
                break;
            case VCR_FASTFWD:
                set_servo(LEFT, FORWARD, 3);   // Press FF to turn right
                set_servo(RIGHT, BACKWARD, 0);
                break;
            case VCR_REWIND:
                set_servo(LEFT, BACKWARD, 0);  // Press RW to turn left
                set_servo(RIGHT, FORWARD, 3);
                break;
            case VCR_PLAY:
                set_servo(LEFT, FORWARD, 4);   // Press play to go forward
                set_servo(RIGHT, FORWARD, 4);
                break;
        }
    }
}
```

- ❑ Compile the program and download it to the controller board. Insert batteries into the remote and press the VCR button to put the remote in VCR mode. Press the different buttons on the remote to control the direction the robot is moving.
- ❑ The drivers are composed of two primary parts. The first part searches for the long high pulse between signals. It then captures the signal by measuring the length of the high and low pulses and checks the pulses for quality. Sometimes interference and ambient light cause the IR receiver to send false data to the microcontroller in the form of signal spikes. The second part takes the received Manchester encoded signal and decodes it, returning an 8-bit button code and the value of the alternating bit to detect repeated commanded.
- ❑ Right click on N9085UD.c and select Open. Scroll to the error code, VCR buttons and TV buttons section. This area defines the values returned by readIRremote(). Use the defines as in the example to add further functionality to the software.

NOTES

- Both the remote and Servo drivers use timer one. The remote drivers use it to measure the pulse widths of the incoming signal while the Servo drivers use CCP1 and CCP2 to compare the current value of timer one to send pulses. The timer one setup is calculated during compile time by examining the environment and the number of timer one divisions. The setup must remain constant for the drivers to function correctly. Edit the constant after `TIMER_1_DIV` if the number of timer one divisions needs to be changed.

16

ADVANCED IR CONTROL

- ❑ A remote control can be used to control movement, speak pre-defined phrases, start and stop processes, sleep and more.
- ❑ Create a file called ex15.c and type in the following source code:

```
#include "robot.h"
#include <N9085UD.c>
#include <servos.c>
#include <wts701.c>
#include <GP2D12.c>

int1 collisionDetect() {
    int8 leftSensor, rightSensor;

    read_ObjectSensors(leftSensor, rightSensor);
    if(leftSensor > 100 || rightSensor > 100) {return 1;}
    else if(leftSensor < 80 && rightSensor < 80) {return 0;}
}

void main() {
    int1 buttonRepeat = 0, runCollisionDetect = TRUE;
    int8 button;
    char phrase1[] = "Hello, my name is pic-robot";
    char phrase2[] = "Danger Phil Robinson, danger!";

    tts_init();
    init_servos();
    init_IRremote();
    init_objectSensors();

    for(;;) {
        if(runCollisionDetect && collisionDetect()) {
            stop_servos();
        }
    }
}

(continued...)
```

(continued...)

```
    if(readIRremote(button, FALSE) != buttonRepeat) {
        switch(button) {
            case VCR_ONE:      tts_sendText(phrase1);      break;
            case VCR_TWO:      tts_sendText(phrase2);      break;
            case VCR_PLAY:     set_servo(LEFT, FORWARD, 1);
                               set_servo(RIGHT, FORWARD, 1); break;
            case VCR_REWIND:   set_servo(LEFT, BACKWARD, 1);
                               set_servo(RIGHT, FORWARD, 3); break;
            case VCR_FASTFWD:  set_servo(LEFT, FORWARD, 3);
                               set_servo(RIGHT, BACKWARD, 1); break;
            case VCR_STOP:     stop_servos();              break;
            case VCR_POWER:    ++runCollisionDetect;       break;
        }
        if(button != VCR_TV_ERR)
            ++buttonRepeat;
    }
}
```

- ❑ Compile the program and download to the controller board. Set the robot in an open area on the floor. Press the different buttons to make the robot speak and move around.
- ❑ The part of the IR protocol for detecting repeated buttons is used in this example. It was not necessary in ex14.c because it does not matter if the robot is told to head in the same direction multiple times. In ex15.c, text is being sent to the text to speech chip and collision detection is being turned on and off with one button. Without the repeat button check, collision detection would be enabled and disabled many times with one press of the power button.

REAL TIME OPERATING SYSTEM (RTOS) BASICS

- ❑ The purpose of an operating system is to schedule different tasks competing for the same resources. Preemptive scheduling is a common method. Each process gets its own process block, which contains a stack and copies of the various registers, such as the program counter. When the operating system preemptively stops a process, the process is not yet finished. Its state must be saved in its process block. The method of saving one process block and restoring another is called a context switch. Each process is repeatedly run for a very short time, so it seems like they are all running at once. A preemptive scheduling system is not well suited for the PIC[®] MCU because of the way the stack is implemented. The CCS C compiler uses a different method similar to batch processing, where each task is allowed to run until completion.
- ❑ The built-in operating system is comprised of three basic parts: major cycles, minor cycles and task execution rates. One complete run of all active tasks is one major cycle. The execution rates are used to calculate the amount of time in a minor cycle. Each task's rate must be a multiple of the minor cycle time. To achieve exact rates, the operating system wastes time not used by the tasks. This is done by counting the number of minor cycles until another task is run. Tasks can be stopped, started and even ended early to give up the rest of their time to the next task. If requested, the operating system will keep timing statistics. It will store the shortest and longest runtime of a task as well as the total time it has been running. Timing information is useful when configuring the different tasks. Time information ensures each task is given enough time to execute and that all resources are available.
- ❑ It is often necessary for one task to transmit some information to another task. The built-in operating system offers an easy way to handle these messages. Each task is given a message buffer to store incoming messages. Next, it then polls for received messages and reads them if present. Each message is one byte.
- ❑ Sometimes two or more tasks need to use the same memory region or port. An integer called a semaphore makes this possible. When a task needs to access a restricted region, it waits for the semaphore to be greater than zero. Once true, the semaphore is decremented, preventing other tasks from accessing the restricted area. The task then enters what is called a critical section. This block of code must be executed quickly. After the critical section, the task signals the restricted area and the area is opened by incrementing the semaphore. Any clean-up is then done in the remainder section.

- ❑ Create a file called ex16.c and type in the following source code:

```
#include "robot.h"
#use rtos(timer=0, minor_cycle=1ms)

#task(rate=20ms, queue=8)
void task1()
{
    if(rtos_msg_poll() > 0)
    {
        putc(rtos_msg_read());
    }
}

#task(rate=20ms)
void task2()
{
    if(kbhit())
    {
        rtos_msg_send(task1, getc());
    }
}

void main()
{
    rtos_run();
}
```

- ❑ Compile the program and download to the controller board. Connect the RS-232 cable and press buttons on the keyboard. Task one is set to run every 20ms. Task 1 checks if there is a character in the RS-232 buffer, and sends it in a message to task two. Task 2 checks its message buffer for data every 20ms. If there is data, Task 2 sends it over RS-232 to the computer. These two tasks together create a simple character echoing system.

- ❑ Using tasks to sample the different sensors on the robot makes it easy to handle the variety of inputs. Since the operating system has the capability to enable and disable tasks, it is easy to control which sensors are active.
- ❑ Create a file called `ex17.c` and type in the following source code:

```
#include "robot.h"
#include <N9085UD.c>
#include <servos.c>
#include <GP2D12.c>
#include <wts701.c>
#use rtos(timer=0)

#task(rate=20ms, queue=5)           //Task 2
void task_servoControl()
{
    if(rtos_msg_poll() > 0)
    {
        switch(rtos_msg_read())
        {
            case 0x10:    stop_servos();
                        break;
            case 0x11:    set_servo(RIGHT, FORWARD, 3);
                        set_servo(LEFT, FORWARD, 0);
break;
            case 0x12:    set_servo(RIGHT, FORWARD, 0);
                        set_servo(LEFT, FORWARD, 3);
break;
            case 0x13:    set_servo(RIGHT, FORWARD, 3);
                        set_servo(LEFT, FORWARD, 3);
break;
        }
    }
}

(continued...)
```



```

(continued...)

#task(rate=10ms)    //Task 1
void task_collision()
{
    if(read_leftObjectSensor() > 100
        || read_rightObjectSensor() > 100)
    {
        rtos_msg_send(task_servoControl, 0x10);
    }
}

#task(rate=200ms)    //Task 3
void task_servo_IR()
{
    static int8 repeat = 0;
    int8 button;

    if(readIRremote(button, FALSE) != repeat)
    {
        switch(button)
        {
            case VCR_PLAY:          rtos_msg_send(task_servoControl,
0x13); break;
            case VCR_REWIND: rtos_msg_send(task_servoControl, 0x12);
break;
            case VCR_FASTFWD:      rtos_msg_send(task_servoControl,
0x11); break;
            case VCR_STOP:         rtos_msg_send(task_servoControl,
0x10); break;
        }

        if(button != VCR_TV_ERR)
        {
            ++repeat;
        }
    }
}

#task(rate=100ms)    //Task 4
void task_IR()
{
    static int8 pStates = 0x07;
    int8 button;
    char msg[23];
(continued...)

```

```
(continued...)

if(readIRremote(button, FALSE))
{
    if(button < 4)
    {
        if(bit_test(pStates, button))
        {
            bit_clear(pStates, button);

            switch(button)
            {
                case 1:    rtos_disable(task_servoControl);
                           sprintf(msg, "Servo control disabled");
                           break;

                case 2:    rtos_disable(task_collision);
                           sprintf(msg, "Collision disabled");
                           break;

                case 3:    rtos_disable(task_servo_IR);
                           sprintf(msg, "Remote Disabled");
                           break;

            }
        }
        else
        {
            bit_set(pStates, button);

            switch(button)
            {
                case 1:    rtos_enable(task_servoControl);
                           sprintf(msg, "Servo control enabled");
                           break;

                case 2:    rtos_enable(task_collision);
                           sprintf(msg, "Collision enabled");
                           break;

                case 3:    rtos_enable(task_servo_IR);
                           sprintf(msg, "Remote Enabled");
                           break;

            }
        }
        tts_sendtext(msg);
        tts_waitConversion();
    }
}

(continued...)
```

(continued...)

```
void main()
{
    init_servos();
    init_IRremote();
    init_objectSensors();
    tts_init();
    rtos_run();
}
```

- Download the program to the controller board. Test the program by pressing different buttons on the remote. Drive the robot toward an obstacle. Once it stops, press the Play button to go forward again. Notice how long it takes the robot to stop. Change the rate of the collision detection task so it is not executed as often and repeat. The robot should drive forward longer before stopping again.
- Each task performs a different function. The first task (task_servoControl) will check if a movement command is in the movement queue. If so, it changes the Servo motion to match the command. The second task (task_collision) uses the proximity sensors to sense any objects in the robot's path. The third task (task_servo_IR) accepts commands from the remote that change the Servo motion and stores the command in the movement queue. The fourth task (task_IR) enables and disables the other three tasks based on commands from the remote.
- The first three tasks can be disabled and enabled by pressing the corresponding numbered button on the remote. The behavior of the robot will change depending on which tasks are enabled at the time. The following chart will outline the expected behavior of the robot.

Task Name			Robot Behaviors
servoControl	Collision	Servo_IR	
			Continues last movement indefinitely.
		<input checked="" type="checkbox"/>	Stores movement commands.
	<input checked="" type="checkbox"/>		Avoids collisions.
<input checked="" type="checkbox"/>			Changes movement.
	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Stores movement commands and avoids collisions.
<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Stores movement commands and changes movement.
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		Changes movement and avoids collisions.
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Performs all tasks (default)

- Create a file called ex18.c and type in the following source code:

```
#include "robot.h"
#include <N9085UD.c>
#include <servos.c>
#include <wts701.c>
#include <GP2D12.c>
#include <compass.c>
#include <line_tracker.c>

int1 collisionDetect()
{
    int8 leftSensor, rightSensor;

    read_ObjectSensors(leftSensor, rightSensor);
    if(leftSensor > 100 || rightSensor > 100)
        return 1;
    else if(leftSensor < 80 && rightSensor < 80)
        return 0;
}

void main()
{
    int1 l, r, runCollisionDetect = TRUE, buttonRepeat = 0;
    int8 button;
    char s[14];

    tts_init();
    init_servos();
    compass_init();
    init_IRremote();
    init_lineTracker();
    init_objectSensors();

    for(;;)
    {
        if(runCollisionDetect && collisionDetect())
        {
            stop_servos();
        }
        else
        {
            l = lt_check(LT_LEFT);
            r = lt_check(LT_RIGHT);
        }
    }
}
(continued...)
```

```

(continued...)
    if(l && !r)
    {
        set_servo(LEFT, FORWARD, 0);
        set_servo(RIGHT, FORWARD, 3);
    }
    else if(r && !l)
    {
        set_servo(RIGHT, FORWARD, 0);
        set_servo(LEFT, FORWARD, 3);
    }
    else
    {
        set_servo(RIGHT, FORWARD, 3);
        set_servo(LEFT, FORWARD, 3);
    }
    if(readIRremote(button, FALSE) != buttonRepeat)
    {
        switch(button)
        {
            case VCR_POWER: ++runCollisionDetect;
break;

            case VCR_EJECT:
                stop_servos();
                delay_ms(20);
                sprintf(s, "%lu degrees\0", compass_getAngle());
                tts_sendText(s);
                break;
        }
        if(button != VCR_TV_ERR)
            ++buttonRepeat;
    }
}
}
}

```

- ❑ Compile the program and download it to the controller board. The program makes use of all parts on the robot, except for RS-232. The line tracking sensors and Servos are used to follow a line. The proximity sensors detect objects in the robot's path. Finally, a talking compass gathers the current robot orientation and sends it to the text-to-speech chip.

Appendix A

CCS recommends using Nickel-Metal-Hydrate, or NiMH, rechargeable batteries.

For more information on battery types, please visit this website:

<http://www.junun.org/MarkIII/Manual/Appendix.jsp>

For more detailed instructions on assembling the robot, check out the manufacturer's website at: <http://junun.org>

Advanced Project Ideas

The example programs throughout this exercise book are intended to introduce the reader to C programming on the Microchip PIC[®] and to controlling the robot. The sensor readings are used in basic ways just to show how they operate. This section of the book contains suggestions for using the included sensors and suggestions for other sensors that can be added to the expansion port.

Chapter 9 used the proximity sensors as one logical unit. It treated the left and right sensors equally and halted the robot if anything was detected. A smarter design would examine the value returned by each sensor independently and make a judgment for whether to turn left or right to avoid the object in advance. The distance from an object could also be compared to the speed of the servos to control how fast to turn.

The robot comes with everything needed to enter a robot sumo competition. The line sensors detect the edge of the ring, the proximity sensors find other robots, and the wheels have rubber grips for better traction; even the text to speech chip can be used for a victory slogan. See the Robot Resources section for robot sumo guides and rules.

Chapter 12 discussed the use of the electronic compass, but did not use it for navigation. Create a program that uses the compass to travel in a certain direction. If the robot encounters an obstacle, it should attempt to go around and return to its original path. Note that the Servos create magnetic interference, so the robot should be halted when taking a compass reading.

A simple way to navigate a maze is to make a left turn whenever possible. Create a more sophisticated method by remembering wrong turns to solve it quicker. Store the solution to the maze so it can be repeated or reversed.

Create a firefighting robot by inserting a photoresistor and thermoresistor into the expansion ports on the controller board. Use them to find sources of light and heat without getting too close. The photoresistor could also be used to make the robot follow a flashlight.

Combine the firefighting robot with the maze navigation to find a fire, extinguish the fire, and return the robot back to where it started.

Use the IR remote to select a series of movements. Save the movements and how long the robot drove in each direction. After 10 direction changes or the press of a special key, replay the series of movements.

Create a soccer-playing robot. Have it find a ball and push it into a net without stepping out of bounds. Many robot soccer fields are created with varying shades of color allowing the robot to use the analog voltage readings from the line sensors to know exactly where it is.

Robot Resources

Visit some of the following websites for more information on robotics, robot parts, robot competitions and design ideas.

Internet Address	Features
http://www.robotroom.com	An informative guide to designing and building robots and an illustrated guide to the robot sumo competition.
http://www.robotlympics.net	Information about a weekend of robot competitions. Rules for most events are available.
http://www.robotics.com	Information about robot clubs, parts and accessories. Check out their extensive list of robots for ideas.
http://www.robotbooks.com	A great site to find literature for a variety of topics such as robot sports, electronics, mechanics, and minds.
http://www.parallax.com	Visit the robot pages for information and to purchase more robot components, accessories, and kits.

On The Web

Comprehensive list of PIC [®] MCU Development tools and information	www.mcuspace.com
Comprehensive list of PICmicro [®] Development tools and information	www.pic-c.com/links
Microchip Home Page	www.microchip.com
CCS Compiler/Tools Home Page	www.ccsinfo.com
CCS Compiler/Tools Software Update Page	www.ccsinfo.com click: Support → Downloads
C Compiler User Message Exchange	www.ccsinfo.com/forum
Device Datasheets List	www.ccsinfo.com click: Support → Device Datasheets
C Compiler Technical Support	support@ccsinfo.com

Appendix B

RS-232

- RS-232 is a popular communication protocol used on most PCs and many embedded systems. Two signal wires transmit and receive data and a third wire is for ground. The PIC16F877A has built in hardware to buffer serial data when using pin C6 for transmitting and C7 for receiving. The compiler can use any pins, but will take advantage of the built in hardware when using C6 and C7.
- Add the following line of code to enable RS-232 communication:
`#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)`
- The `#use rs232` directive enables the following functions. Read more about them in the help or reference manual included with the compiler.
- Create a new source file called `ex_rs232.c` and type in the following code:

```
#include <16F877A.h>
#fuses HS,NOLVP,NOWDT,NOPROTECT,NOBROWNOUT
#use delay(clock=2000000)
#use rs232( baud=9600, xmit=PIN_C6, rcv=PIN_C7)

void main() {
    printf("Press keys and they will be echoed back\n\r");
    for(;;) {
        putc(getc());
    }
}
```
- Compile the program and load it to the protoboard.
- Connect the protoboard to the PC with the serial to 3.5mm stereo plug cable.
- Click **Tools > Serial Port Monitor** within the PCW IDE.
- Configure the COMM port if necessary by clicking **Configuration > Set port options**.
- Power the protoboard and press keys send characters.
- See `input.c` in the Drivers directory and `ex_float.c`, `ex_exsio.c` and `ex_sis.c` in the Examples directory for further information.

CCS Programmer Control Software

The CCSLOAD software will work for all the CCS device programmers and replaces the older ICD.EXE and MACHX.EXE software. The CCSLOAD software is stand-alone and does not require any other software on the PC. CCSLOAD supports ICD-Sxx, ICD-Uxx, Mach X, Load-n-Go, and PRIME8.

Powerful Command Line Options in Windows and Linux

- Specify operational settings at the execution level
- Set-up software to perform, tasks like save, set target Vdd
- Preset with operational or control settings for user

Easy to use Production Interface

- Simply point, click and program
- Additions to HEX file organization include associating comments or a graphic image to a file to better ensure proper file selection for programming
- Hands-Free mode auto programs each time a new target is connected to the programmer
- PC audio cues indicate success and fail

Extensive Diagnostics

- Each target pin connection can be individually tested
- Programming and debugging is tested with known good programs
- Various PC driver tests to identify specific driver installation problems

Enhanced Security Options

- Erase chips that failed programming
- Verify protected code cannot be read after programming
- File wide CRC checking

Automatic Serial Numbering Options

- Program memory or Data EEPROM
- Incremented, from a file list or by user prompt
- Binary, ASCII string or UNICODE string

CCS IDE owners can use the CCSLOAD program with:

- MPLAB®ICD 2/ICD 3
- MPLAB®REAL ICE™
- All CCS programmers and debuggers

How to Get Started:

Step 1: *Connect Programmer to PC and target board. Software will auto-detect the programmer and device.*

Step 2: *Select Hex File for target board.*

Step 3: *Select Test Target. Status bar will show current progress of the operation.*

Step 4: *Click "Write to Chip" to program the device.*

Use the Diagnostics tab for troubleshooting or the ccsload.chm help file for additional assistance.

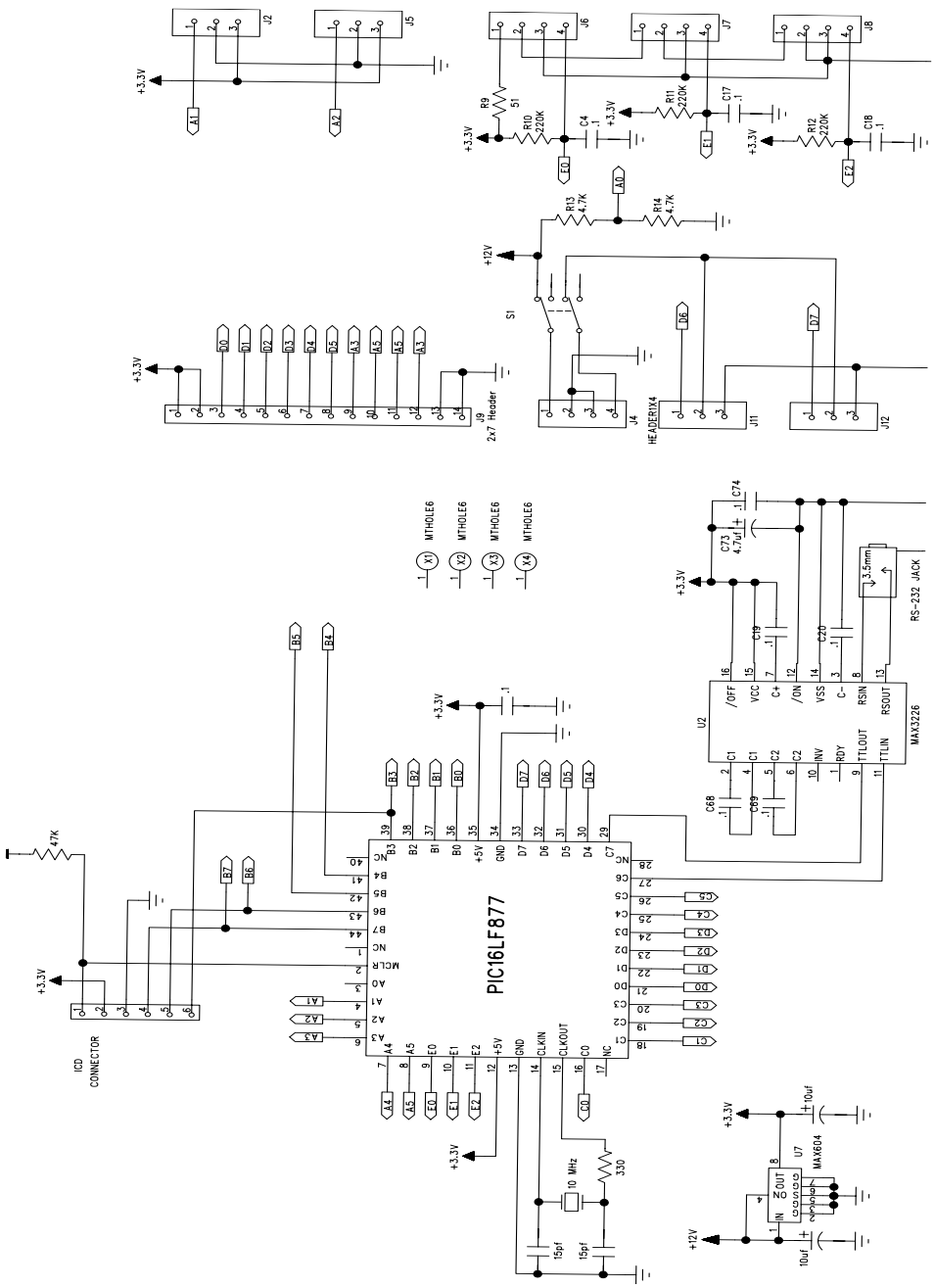
SERVO SPECIFICATIONS At 4.8V

ITEM	SIZE mm	WEIGHT g/oz	SPEED sec/60	TORQUE kg-cm/oz-in
S136G-MG	45.0X22.7X24.5	34/1.20	0.50	7.40/103
S136L-MG	45.0X22.7X24.5	34/1.20	0.33	8.00/111
S9102	39.5X20.0X35.8	42/1.48	0.18	2.50/35
S9102F	39.5X20.0X35.8	32/1.13	0.09	2.50/35
S03N	39.5X20.0X39.6	41/1.44	0.23	2.40/47
S03N F	39.5X20.0X39.6	41/1.44	0.18	2.80/39
S03N XF	39.5X20.0X39.6	41/1.44	0.15	2.20/31
S03T	39.5X20.0X39.6	46/1.62	0.33	7.20/100
S03T F	39.5X20.0X39.6	46/1.62	0.27	5.80/81
S03T XF	39.5X20.0X39.6	46/1.62	0.21	5.00/69
S05	40.6X20.0X38.0	45/1.58	0.23	3.40/47
S05 2BBMG	40.6X20.0X38.0	64/2.26	0.23	3.40/47
S06	40.6X20.0X42.8	48/1.69	0.33	7.20/100
S06 2BBMG	40.6X20.0X42.8	73/2.58	0.33	7.40/103
S07	40.6X20.0X38.0	45/1.58	0.18	2.80/39
S07 2BBMG	40.6X20.0X38.0	64/2.26	0.18	2.80/39
S08	40.6X20.0X42.8	48/1.69	0.27	5.80/81
S08 2BBMG	40.6X20.0X42.8	73/2.58	0.27	6.00/83
S09	40.6X20.0X38.0	45/1.58	0.15	2.20/31
S09 2BBMG	40.6X20.0X38.0	64/2.26	0.15	2.50/35
S10	40.6X20.0X42.8	48/1.69	0.21	5.00/69
S10 2BBMG	40.6X20.0X42.8	73/2.58	0.21	5.60/78

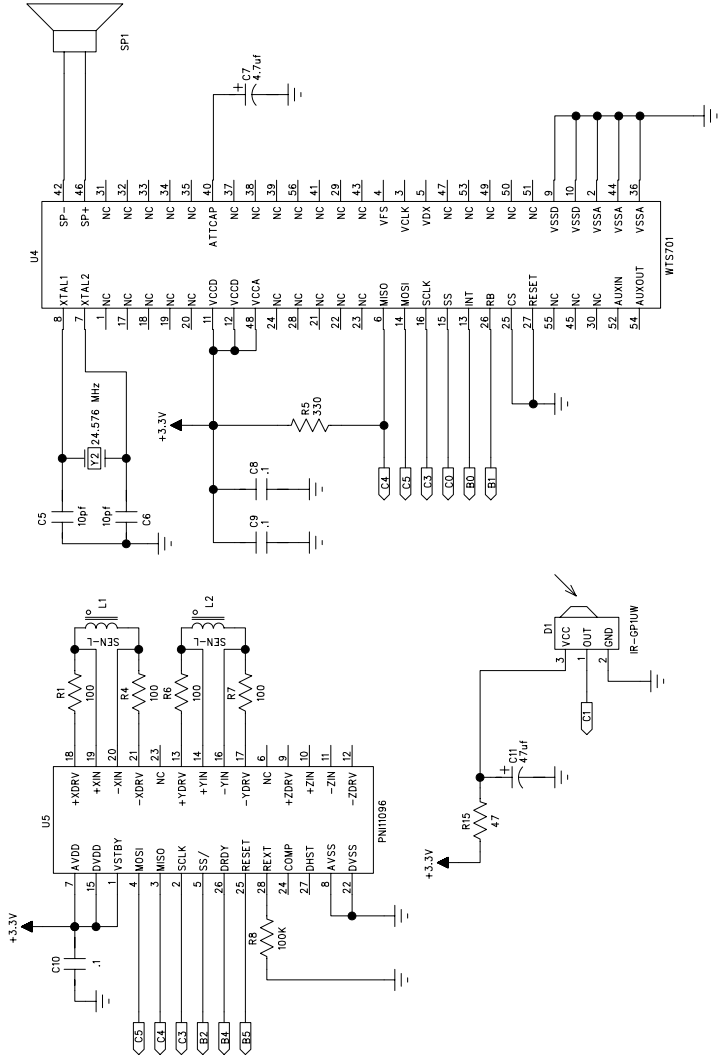
- GWS servos are designed compatible with RC systems manufactured by Futaba, JR, Hi-tec, Airtronics(Sanwa), Multiplex and any systems using 1500 μ S neutral.
- When installing the servo with the provided hardware, insert the screws through grommets (and eyelets) and tighten the screws until the grommets are slightly crushed for best shock absorption. Do not over-tighten the screws.
- New GWS servos are warranted against defects in material and workmanship. This warranty does not apply to any products which have been improperly installed, handled, abused, damaged in crash, nor been repaired or altered by unauthorized agencies. If any service is required, consult with the model shop where you originally purchased the products.
- Specifications are subject to change without notice.

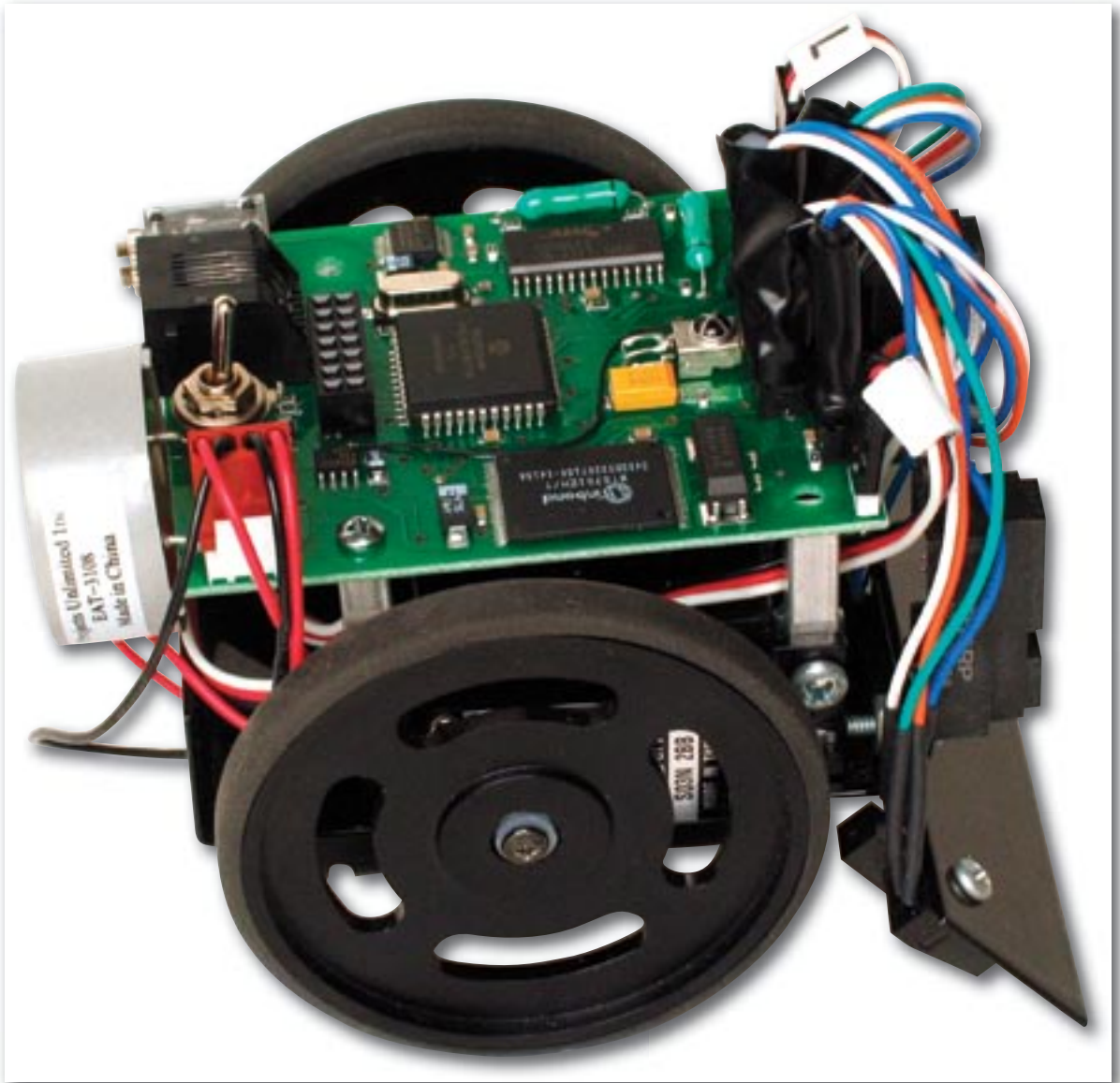
<http://www.gws.com.tw>
E-mail: export@gws.com.tw

MADE IN TAIWAN
2002/08



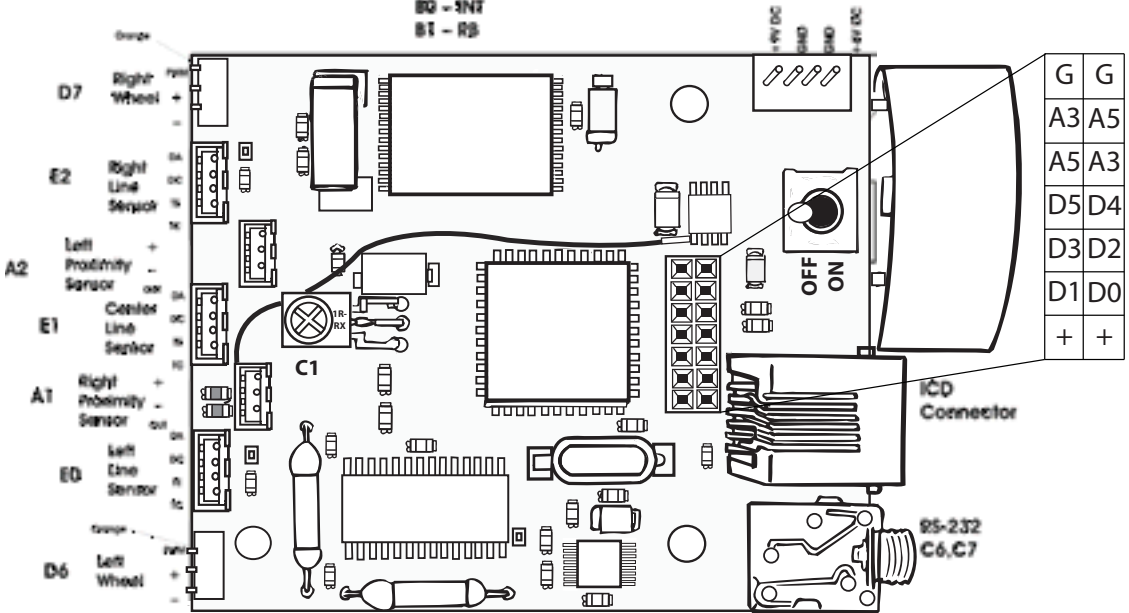
- 1 (X) M/HOLE6
- 2 (X) M/HOLE6
- 3 (X) M/HOLE6
- 4 (X) M/HOLE6





Text to Speech

- C4 - D0
- C5 - D1
- C3 - CLK
- C0 - SS
- B0 - INT
- B1 - RB



G	G
A3	A5
A5	A3
D5	D4
D3	D2
D1	D0
+	+

Compass

- C5 - OI
- C4 - DO
- C3 - CLK
- B2 - SS
- B4 - ROY
- B5 - Reset