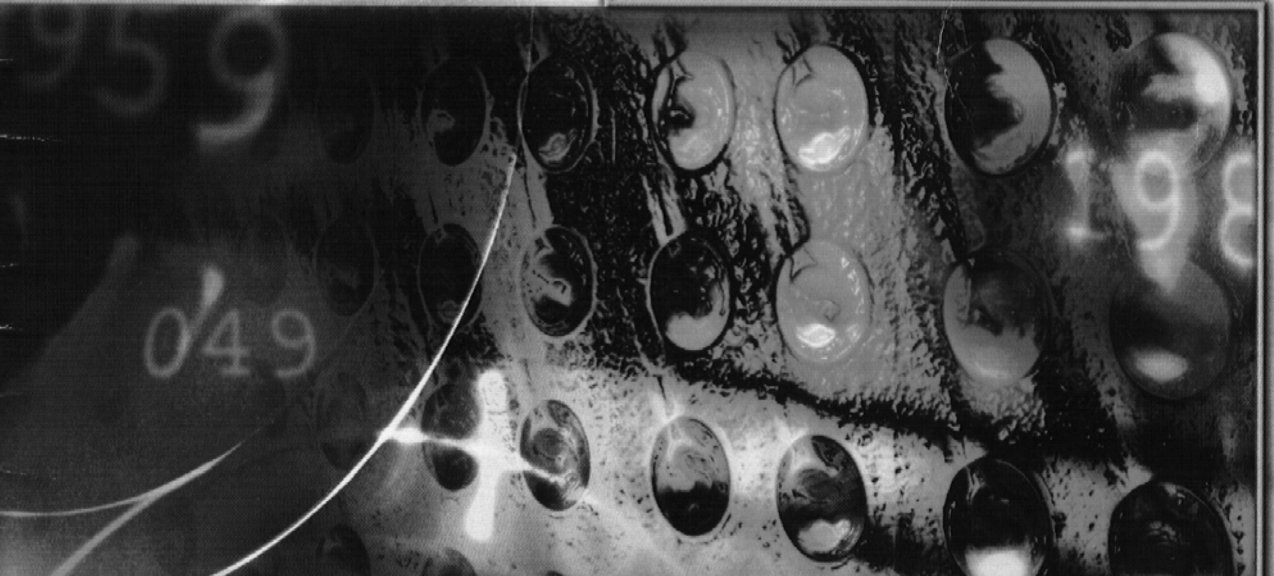
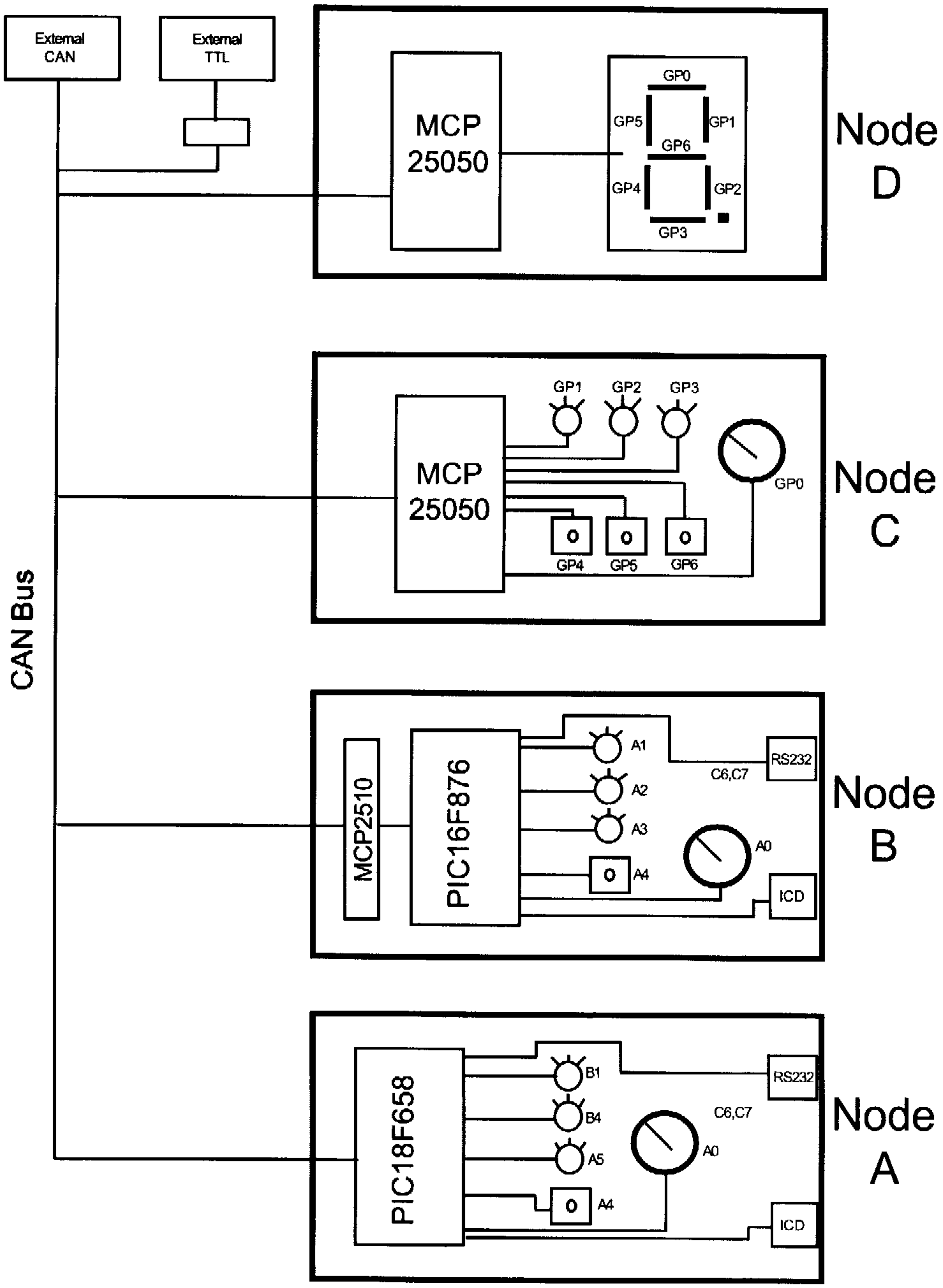
Embedded C Language DEVELOPMENT KIT

For the PICmicro® MCU



CAN Bus
Exercise Book



External CAN
External TTL
CAN Bus
MCP 25050
GP0
GP1
GP2
GP3
GP4
GP5
GP6
Node D
MCP 25050
GP1
GP2
GP3
GP0
GP4
GP5
GP6
Node C
MCP2510
PIC16F876
A1
A2
A3
A4
A0
RS232
C6,C7
ICD
Node B
PIC18F658
B1
B4
A5
A4
A0
RS232
C6,C7
ICD
Node A

Embedded C Language Development Kit For the PICmicro® MCU

Exercise Book

CAN Bus

Brookfield, Wisconsin USA
262-797-0455

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 CD-Rom drive and a 5 meg of disk space.
- ☐ The diagram on the following page shows each component in this kit. Make sure all items are present.

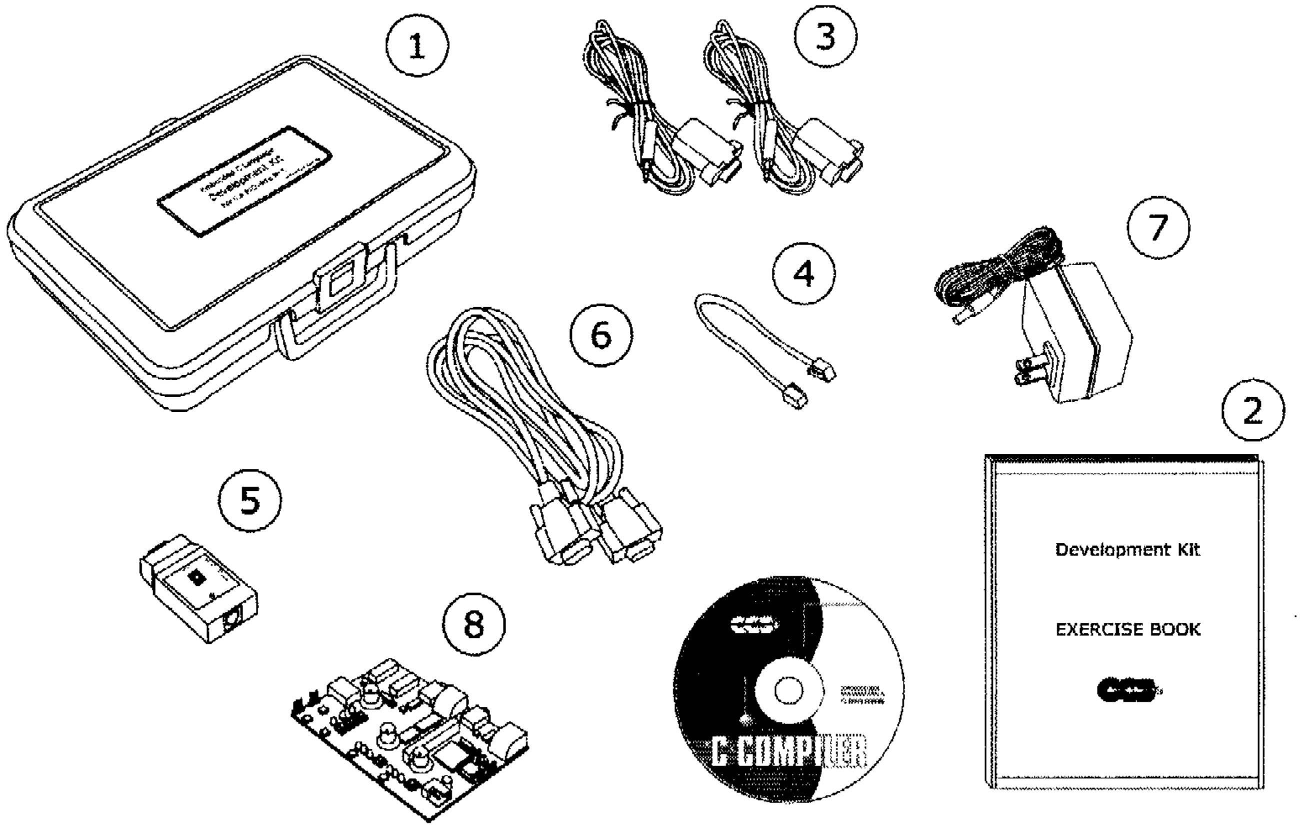
Software

- ☐ Insert the CD into the computer and wait for the install program to start. If your computer is not set up to auto-run CDs, then select **START>RUN** and enter **D:\SETUP1.EXE** where D: is the drive letter for your CD drive.
- ☐ Click on **Install** and use the default settings for all subsequent prompts. Click NEXT, OK, CONTINUE...as required.
- ☐ Identify a directory to be used for the programs in this booklet. The install program will have created an empty directory **c:\program files\picc\projects** that may be used for this purpose. However, the example files can be found on the compiler CD-rom, located in the directory D:/CCS/CAN Exercise Book/, where D:/ is the location of your CD-rom.
- ☐ Select the compiler icon on the desktop. In the PCW IDE click **Help>About** and verify a version number. This is shown for the IDE and/or PCM. This ensures the software is installed properly. Exit the software.

Hardware

- ☐ Connect the PC to the ICD (5) using the 9 pin cable (6) for ICD-S or the USB[1] for ICD-U and connect prototype board (8) to the ICD using the modular cable (4). Plug in the AC adaptor (7) and plug it into the prototype board (8). See the diagram in the section 4 for help.
- ☐ The LED on the ICD should be on.
- ☐ Run the program at **Start>Programs>PICC> ICD**. If the program reports a communication error, select a different COM port until you find the port connected to the ICD-S.
- ☐ Select **Check COMM**, then **Test ICD**, then **Test Target**. If all tests pass, then the hardware is installed properly.
- ☐ Disconnect the hardware until you are ready for Exercise 3. Always disconnect the power to the prototype board before connection/disconnecting the ICD or changing the jumper wires to the prototype board.

[1]The differences between ICD-S and ICD-U are as follows: ICD-S uses RS-232 to connect to a PC, while the ICD-U uses USB. An USB driver must also be installed before an ICD-U can be used. Install this driver after connecting the ICD-U to the PC and prototype board as described above. Windows will display a New Hardware Found notification. Insert the disk with the drivers and follow the wizard to complete the installation. RS-232 communication between the PC and programs on the PIC functions identically with both ICDs.



① Carrying case.
② This exercise booklet.
③ Two PC cables
④ Modular cable for ICD
⑤ ICD
⑥ ICD PC cable (or USB cable)
⑦ AC Adapter
⑧ CAN Bus prototyping board

2 Using the Integrated Development Environment (IDE)

Editor

- ❑ Open the PCW IDE. If any files are open, click **File>Close All**
- ❑ Click File>Open. Select the file: **c:\program files\picc\examples\ex_stwt.c**
- ❑ Scroll down to the bottom of this file. Notice the editor shows 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 along with setting bookmarks and various C specific functions.
- ❑ Review the editor option settings by clicking on **Options/Editor Properties**. The IDE allows selection of the tab size, editor colors, fonts and many more. The **Options/Customize** allows to select icons that should also appear on the toolbar.

Compiler

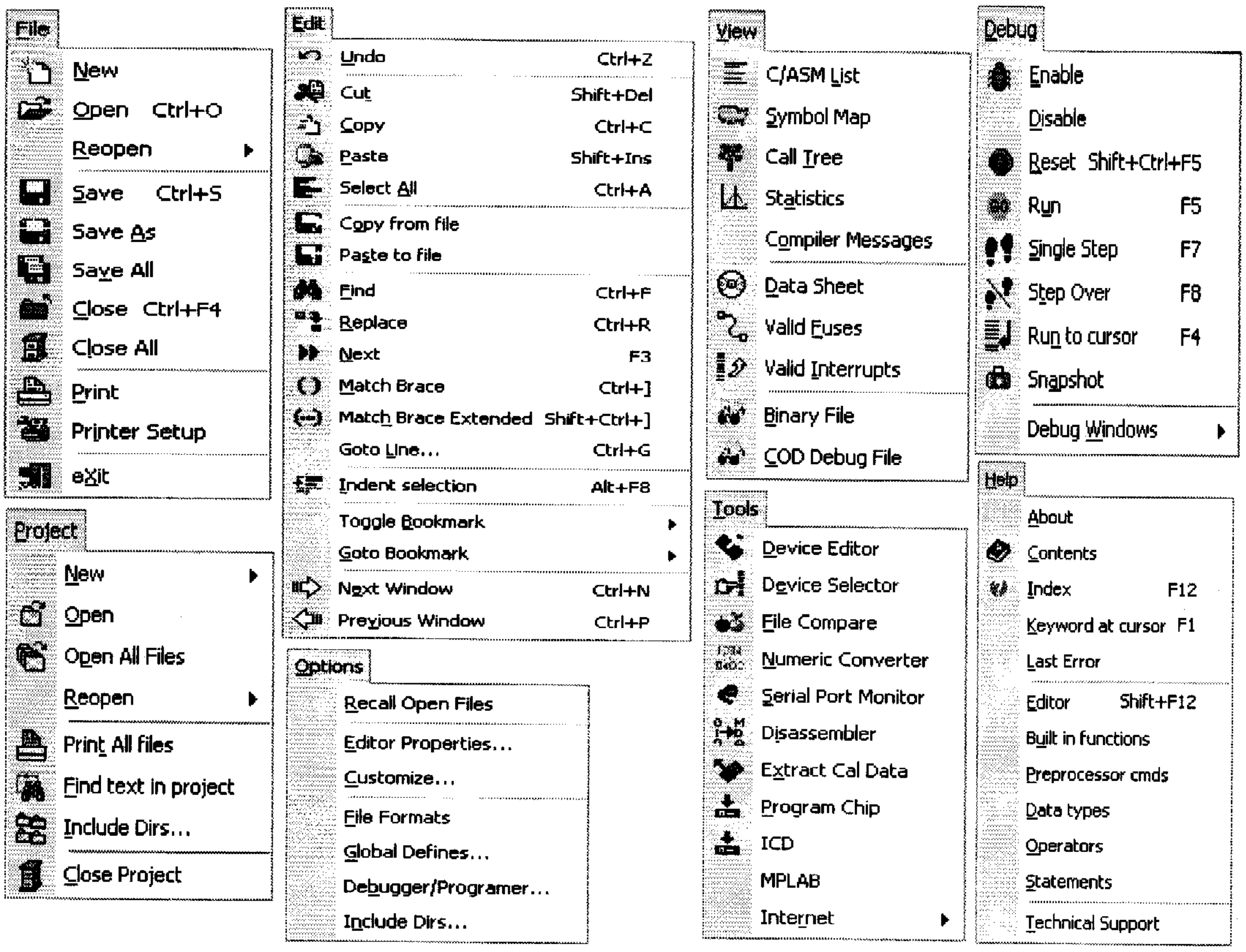
- ❑ Use the white box on the toolbar to select the compiler. CCS offers different compilers for each family of Microchip parts. Some of exercises in this booklet are for the PIC16F876A chip, an 14-bit opcode part. Make sure **14 bit** is selected in the white box. Other programs are for the PIC18F458 part, a 16-bit Op Code. Select PIC18 for that part.
- ❑ The main program compiled is always shown in the lower right corner of the IDE. If this is not the file you want to compile, then click on the tab of the file you want to compile. Right click into editor and select **Make file project**.
- ❑ Click **Options>Include Dirs...** 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 to point to the device .h files and the device drivers.
- ❑ Normally the file formats need not be changed and global defines are not used in these exercises. To review these setting, click **Options>File Formats** and **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 **View>Symbol Map**. This file shows how the RAM in the micro-controller 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 **View>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 location. Switch to the Symbol Map to find the memory location where int_count is located.
- ☐ Click **View>Data Sheet**, then View. This brings up the Microchip data sheet for the microprocessor being used in the current project.



3 CAN Bus Prototyping Board Overview

❑ The CCS CAN Bus prototyping board has a CAN bus with four nodes on the same board. A block diagram is within the front cover of this booklet. The four independent nodes are as follows:

NODE A - PIC18F458

❑ This node features a Microchip PIC18F458 chip. This chip has a built-in CAN bus controller. There is also an I/O block that provides access to spare I/O pins on the PIC. The pinout is as follows:

+5	B6	B4	B2	B0	D6	D4	D2	D0	C4	C2	C0	A4	A2	E2	E0	G
+5	B7	B5	B3	B1	D7	D5	D3	D1	C5	C3	C1	A5	A3	A1	E1	G

❑ The following I/O features are also a part of NODE A:

- ♦ Three LEDs (Red, Yellow, Green)
  - LED is lit by outputting a LOW to the I/O pin.
- ♦ One push-button
  - I/O pin reads LOW when the button is pressed.
- ♦ Pot to provide an analog voltage source
  - 0 Volts full counterclockwise, 5 Volts full clockwise
- ♦ RS-232 port

NODE B - PIC16F876

❑ This node features a Microchip PIC16F876 chip. This chip does NOT have a built-in CAN bus controller. Instead, an external MCP2510 CAN bus controller is used. This scheme could be used with any PIC microcontroller.

❑ The following I/O features are also a part of NODE B:

- ♦ Three LEDs (Red, Yellow, Green)
  - LED is lit by outputting a LOW to the I/O pin.
- ♦ One push-button
  - I/O pin reads LOW when the button is pressed.
- ♦ Pot to provide an analog voltage source
  - 0 Volts full counterclockwise, 5 Volts full clockwise
- ♦ RS-232 port

❑ Programs may be downloaded and optionally debugged using the ICD connector.

NODE C - “Dumb” I/O Unit

- ❑ This node features a Microchip MCP25050 chip. This chip is pre-programmed with address information and provides CAN bus access to the 8 I/O pins. The following items are connected to the I/O pins:
  - ♦ Three LEDs (Red, Yellow, Green)
    - LED is lit by outputting a LOW to the I/O pin.
  - ♦ Three push-buttons
    - I/O pin reads LOW when the button is pressed.
  - ♦ Pot to provide an analog voltage source
    - 0 Volts full counterclockwise, 5 Volts full clockwise
  - ♦ RS-232 port
- ❑ This chip may be programmed by removing it from the socket and using the Pro Mate II from Microchip to load in the address information.

NODE D - “Dumb” 7 Segment LED

- ❑ This node features a Microchip MCP25050 chip. This chip is pre-programmed with address information and provides CAN bus access to the 8 I/O pins. The I/O pins are connected to a 7 Segment LED. This allows a number to be displayed via the CAN bus. A LOW on the I/O pin lights the segment. For example, outputting a 0xC0 in the GP port will light a 0. A 0xF9 will light a 1.
- ❑ This chip may be programmed by removing it from the socket and using the Pro Mate II from Microchip to load in the address information.

Notes:

- Both Node C & D use a Microchip MCP25050 CAN Bus chip.
- This chip is a complete CAN Bus Node that allows eight general input or output pins, up to four A/D converter inputs and two PWM outputs
- This chip can be configured by programming an internal EEPROM with the addresses and modes of operation.
- The chip can also be programmed over the CAN Bus.

4 Compiling and Running a Program

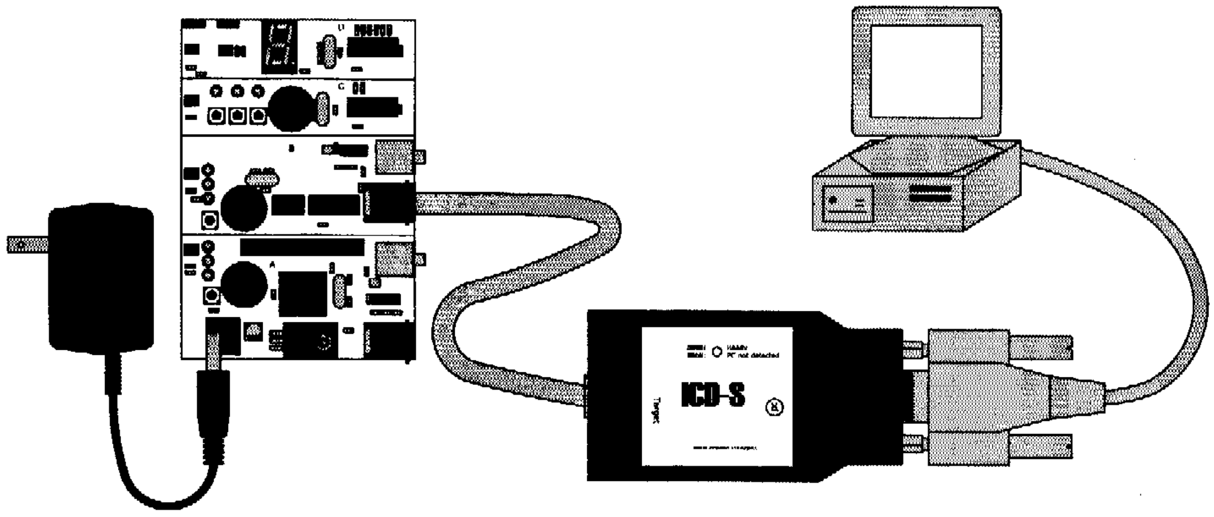
- ❑ Open the PCW IDE. If any files are open, click **File>Close All**
- ❑ Click **File>New** and enter the file name **EX3.C**
- ❑ Type in the following program and **Compile**

```
#include <18f458.h>
#device ICD=TRUE
#fuses  HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)

#define GREEN_LED PIN_A5

main () {
     while(TRUE) {
          output_low(GREEN_LED);
          delay_ms(1000);
          output_high(GREEN_LED);
          delay_ms(1000);
     }
}
```

Notes:

- The first four lines of this program define the basic hardware environment. The chip being used is the PIC18F458, running at 20MHz with the ICD debugger.
- The #define is used to enhance readability by referring to GREEN_LED in the program instead of PIN_A5.
- The "while (TRUE)" is a simple way to create a loop that never stops.
- Note that the "output_low" turns the LED on because the other end of the LED is +5V. This is done because the chip can tolerate more current when a pin is low than when it is high.
- The "delay_ms(1000)" is a one second delay (1000 milliseconds).

❑ Connect the ICD to



❑ Click **Debugger>Enable** and wait for the program to load.

❑ Click the green go icon: GO

❑ Expect the debugger window status block to turn yellow, indicating the program is running.

The green LED on the protoboard should be flashing. One second on and one second off.

❑ The program can be stopped by clicking on the stop icon: S

5 CAN Bus Overview

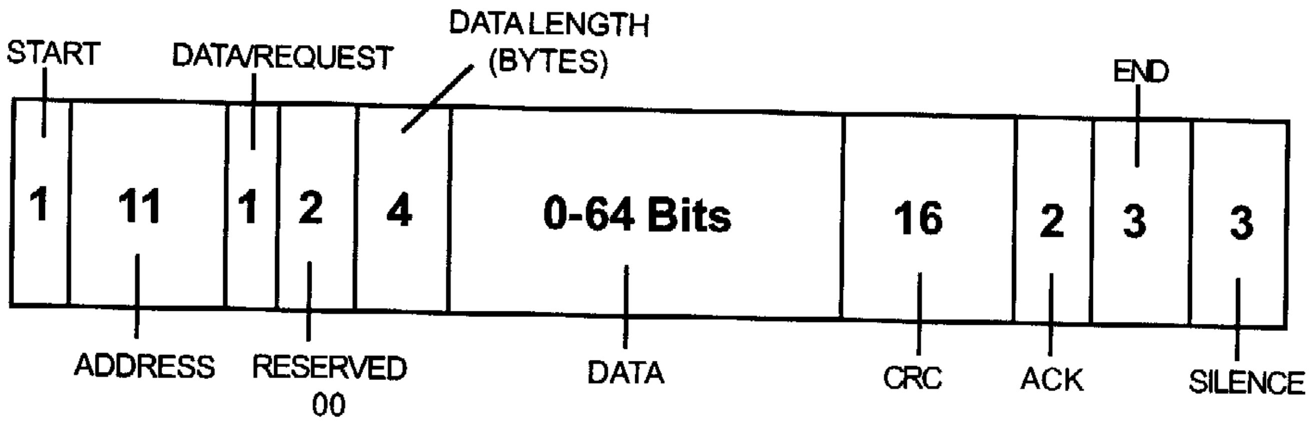
- Nodes on the CAN bus transmit data frames. Each frame has a identifier number identifying the frame content. The frame is available to all other nodes. Nodes that have an interest in the frame (based on the identifier) use the data. Other nodes simply ignore the data.

  SIMPLE 4 NODE EXAMPLE:

  - Node A: Speed detector
    Every 100 ms sends a frame with identifier 1 and data indicating the vehicle speed.
  - Node B: Speedometer display
    Looks only for identifier 1 data on the bus. Takes the data and displays it on a digital display.
  - Node C: Cruise control panel
    Pressing the SET button sends an identifier 2 frame. Pressing the OFF button sends an identifier 3 frame. Neither frame has data.
  - Node D: Cruise control module
    Module turns on with an identifier 2 frame and off with an identifier 3 frame. The module uses data from identifier 1 frames to adjust the vehicle speed.

- Notice this is not a command/response type of protocol. Nodes that have something to say will say it. Nodes that need to know something will wait for what they need. A higher level protocol can be implemented to provide more control. Notice Node C actually can control how Node D works. If a node needs a certain type of data, it can post a request on the bus for a frame with a particular identifier. The node responsible for that identifier will respond. A system design should assign a given identifier (or set of identifiers) to only one node.

BASIC FRAME FORMAT:



- See chapter 6 for the Extended Format.

GENERAL RULES:

- All nodes on the bus verify the frame and if any node, detects an error that node asserts a NACK. When any node asserts a NACK for a frame all nodes must ignore the frame even if the node did not find an error in the frame. The sender re-transmits NACKed frames.
- A node that NACKs a lot of messages or has a lot of messages NACKed is put on probation (Error Passive state). In this state, the nodes activity is restricted. If the problem persists, the node must stop all bus transmission and ignore all incoming packets. This rule is self-enforced by each node keeping local statistics.
- A node does not start transmitting unless the bus is quiet for three bit times. If two nodes start a frame at the same time, one node will bow out while the identifier is being transmitted. The node to drop out will be the one that first tries to send a one-bit, when the other send a 0. The 0 is dominant and the sender of the one will realize there is a collision. This means lower numbered identifiers have a higher priority.
- The CAN bus permits an alternate format message with a 29 bit identifier. All the examples we use will be with an 29 bit identifier. Frames with 11 and 29 bit identifiers can co-exist on the same bus.

PHYSICAL:

- There is no universal standard for the physical CAN bus. It requires an open drain type of bus. It could be a single wire, fiber optic or two wire differential bus. The latter is the physical bus used on the CCS CAN bus prototyping board. The Philips PCA82C251 chips are used to interface the bus to the TTL controllers. This complies with ISO standard 11898 for use in Automotive and Industrial applications
- The bit rate can be as fast as one million bits per second.
- The start of frame bit is used by the receiver to determine the exact bit time.
- Whenever a transmitter on the bus sends five identical bits, it will send an extra bit with the reverse polarity. This is referred to as a stuffed bit. The receiver will ignore the stuffed bits. If a receiver detects six or more bits that are the same, then it is considered an automatic error.

6 Simple PIC18 Transmitter

❑ Enter the following program:

```
#include <18F458.h>
#fuses HS,NOPROTECT,NOLVP,NOWDT
#use delay(clock=20000000)
#include <can-18xxx8.c>

#define WRITE_REGISTER_D_ID  0x400

void write_7_segment(int value) {
   const int lcd_seg[10]={0x40,0x79,0x24,0x30,0x19,
                          0x12,0x02,0x78,0x00,0x10};
   int buffer[3];

   buffer[0]=0x1E;      //addr of gplat
   buffer[1]=0x7F;      //mask
   buffer[2]=lcd_seg[value];
   can_putd(WRITE_REGISTER_D_ID, buffer, 3, 1, TRUE, FALSE);
}

void main() {
   int i=0;

   can_init();
   can_putd(0x100,0,0,1,TRUE,FALSE);  //send an on-bus message
                                      //to wake up mcp250x0's
   delay_ms(1000);                    //wait for node c to power-up

   while(TRUE) {
      write_7_segment(i);
      delay_ms(1000);
      if(++i==10)
         i=0;
   }
}
```

❑ Compile the program and load into Node A and run the program as was done in Chapter 4.

❑ This program should display 0-9 on the 7 segment LED.

Notes:

- The include file “can-18xxx8.c” has the functions required to interface to the PIC18 CAN bus controller.
- The call to can_init() starts the interface.
- This program is designed to send data to Node D. The identifier for Node D is programmed as 0x400. The MCP25050 device accepts a three byte command...
- The can_putd functions have the following parameters:

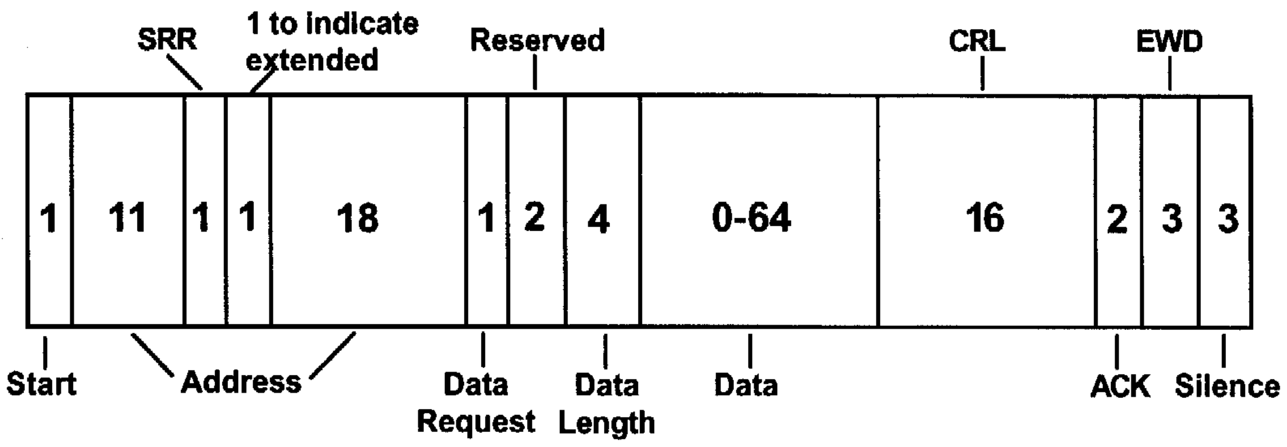
  Identifier
  Data pointer
  Number of data bytes
  Priority (0-3) determines the order messages are sent
  Flag to indicate 29 bit identifier
  Flag to indicate if this is a data frame
  (FALSE) or request for frame (TRUE)

  This call query up a frame for transmission on the bus.
- The MCP250xx units require one error free message after power-up to switch to normal state. The first CAN_putd, to 0x100, sends an empty message which takes the MCP250xx from power-up to normal.

Before Moving On:

- For future examples, copy the lines in this example before “void main() {“ into an include file named CCSCANA.C. In the future, examples will add to this file to build a library of functions specific to the CCS CAN prototype board.

Extended Format (29 Bit ID)



7 Using the MCP250xx for Output

- ❑ The MCP250xx parts used on Nodes C and D allow for discrete input, output and analog input. These parts have internal registers that set the device ID, the directions of the pins, valueso f the outputs, scheduling information for outgoing frames and more. These registers are initialized by programming the part on a Microchip Pro Mate II. The registers can also be raed and modified at run time.
- ❑ The MCP250xx part for Node D has been programmed with a base ID of 0x400. The low three bits of the ID specify a function. for example, 0x400 is a write register command, 0x404 is a write configuration comand. Table 4-2 in the data sheet explains the identifier usage.
- ❑ The write register command has three bytes of data namely, a register, mask and value. The value is written to the specified register changing only the bits specified in the mask. For example, in the previous program, a frame was sent with ID 0x400 and data 0x1E, 0x7F, 0x40. Ox1E is the output latch for the GP pins. 0x7F caused GP7 to be unchanged (connected to decimal point). The value 0x40 puts a low on pins GP0 to GP5 and a high on GP6. Note that the registers listed in the data sheet table 3-1 use addresses for the internal EEPROM. The RAM addresses are 0x1C higher.

Example:

- Send a frame with ID 0x400 and data 0x1E, 0x80, 0x00 to turn on the DP

0x400	– Node D
0x1E	– Output Latch register
0x80	– Only change Bit 7
0x00	– All zeros (only Bit 7 relevant) A 0 or low lights the segment

❑ Node C has three LEDs: Red (GP1), Yellow (GP2) and Green (GP3). Add the following function to ccscana.c:

```
#define WRITE_REGISTER_C_ID  0x300
enum colors {RED=0,YELLOW=1,GREEN=2};

void write_c_led(colors led, short on) {
    int buffer[3];

    buffer[0]=0x1E;
    buffer[1]=0x02<<led;
    if(on)
        buffer[2]=0;
    else
        buffer[2]=0xff;
    can_putd(WRITE_REGISTER_C_ID, buffer, 3, 1, TRUE,
FALSE);       }
```

❑ Then add the following logic to the main program loop in Ex6.c. See Ex7.c in example programs folder from C Compiler CD-rom:

```
write_c_led(GREEN, i>1);
write_c_led(YELLOW, i>4);
write_c_led(RED, i>7);
```

❑ The program should display 0-9 on the LED and light the green, yellow and red LEDs on Node C, if, according to the value, is >1, >4, >7 respectively.

8 Using the MCP250xx for Input

- ❑ The MCP250xx part used on Node C has been programmed to send a frame whenever one of the push-buttons change value (GP4-GP6).
- ❑ The following program will read CAN bus messages looking for that specific ID. It will then light a LED depending on the button pressed.
- ❑ Add this line to ccsana.h:

```
#define NODE_C_PUSHBUTTON_ID  0x303
```

- ❑ Then enter, compile and load the following into Node A:

```
#include <ccscana.c>

void main() {
   int buffer[8],rx_len,rxstat;
   int32 rx_id;

      can_init();

      can_putd(0x100,0,0,1,TRUE,FALSE);  //send an on-bus message
                                          //to wake up mcp250x0's
      delay_ms(1000);                    //wait for node c to power-up

   while(TRUE) {
      if ( can_kbhit() ) {
         if(can_getd(rx_id, &buffer[0], rx_len, rxstat))
        if (rx_id == NODE_C_PUSHBUTTON_ID) {
            write_c_led(YELLOW, !bit_test(buffer[1],4));
            write_c_led(GREEN, !bit_test(buffer[1],5));
            write_c_led(RED, !bit_test(buffer[1],6));
         }
      }
   }
}
```

❑ The write_c_led function calls send a frame to Node C to light a LED. We will now add a program to Node B to look for this same data and perform the same action at Node B.

```
#include <16F876A.H>
#fuses HS,NOPROTECT,NOLVP,NOWDT
#use delay(clock=2500000)

#include <can-mcp2510.c>

#define RED_LED  PIN_A1
#define YELLOW_LED  PIN_A2
#define GREEN_LED  PIN_A3
#define WRITE_REGISTER_C_ID  0x300

main() {
   int32 rx_id;
   int rx_len, rxstat, buffer[8];

   can_init();

   while(TRUE) {
      if ( can_kbhit() ) {
         if(can_getd(rx_id, &buffer[0], rx_len, rxstat))
            if ((rx_id == WRITE_REGISTER_C_ID)&&
                 (buffer[0] == 0x1e)) {
               if(buffer[1]&2)
                 output_bit(GREEN_LED, !bit_test(buffer[2],1));
               if(buffer[1]&4)
                 output_bit(YELLOW_LED, !bit_test(buffer[2],2));
               if(buffer[1]&8)
                 output_bit(RED_LED, !bit_test(buffer[2],4));
            }
      }
   }
}
```

9 Using the MCP250xx for Analog Input and Scheduling Data

- The MCP25050 can be configured for up to four analog inputs. The A/D converter is 10 bits (0-1023). The following program makes a request for the ID with A/D results 10 times per second, then waits for the frame to be sent with that ID. This is a clear example to show these features. However, it is not a good scheme for a real application. This program will hang if the MCP25050 does not answer.
- Enter this program, compile and load into Node A. Test the program by turning the Node C pot. Node A should use the A/D reading to display a number 0-9 on the Node D LED.

```
#include <ccscana.c>

void main() {
     int1 waiting;
     int buffer[8],rx_len,rxstat;
     int32 rx_id;
     int16 ad_val;

     can_init();

     can_putd(0x100,0,0,1,TRUE,FALSE);  //send an on-bus message
                                        //to wake up mcp250x0's
     delay_ms(1000);                    //wait for node c to power-up

      while(TRUE) {
         delay_ms(100);
         can_putd(WRITE_REGISTER_C_ID, 0, 8, 1, TRUE, TRUE);
         waiting=TRUE;
         while(waiting) {
              if ( can_kbhit() )
                if(can_getd(rx_id, &buffer[0], rx_len, rxstat))
                   if (rx_id == WRITE_REGISTER_C_ID) {
                 write_7_segment(buffer[2]/26);
                 waiting=false;
              }
         }
     }
}
```

- ❑ The rate the data is updated to the display is determined by the delay_ms line. Try a delay_ms(1000) to get a feel for how that lag works. Then try a delay_ms(1).
- ❑ Up to this point, we have relied on the MCP250xx settings to pre-programmed into the EEPROM. In this next program, the pre-programmed settings will be changed. This chip has the capability to send certain messages when certain events happen, or on some regular basis. We will program the chip to send out the analog frame roughly 10 times per second.
- ❑ Add #define NODE_C_SCHEDULED 0x301 to ccscana.c.

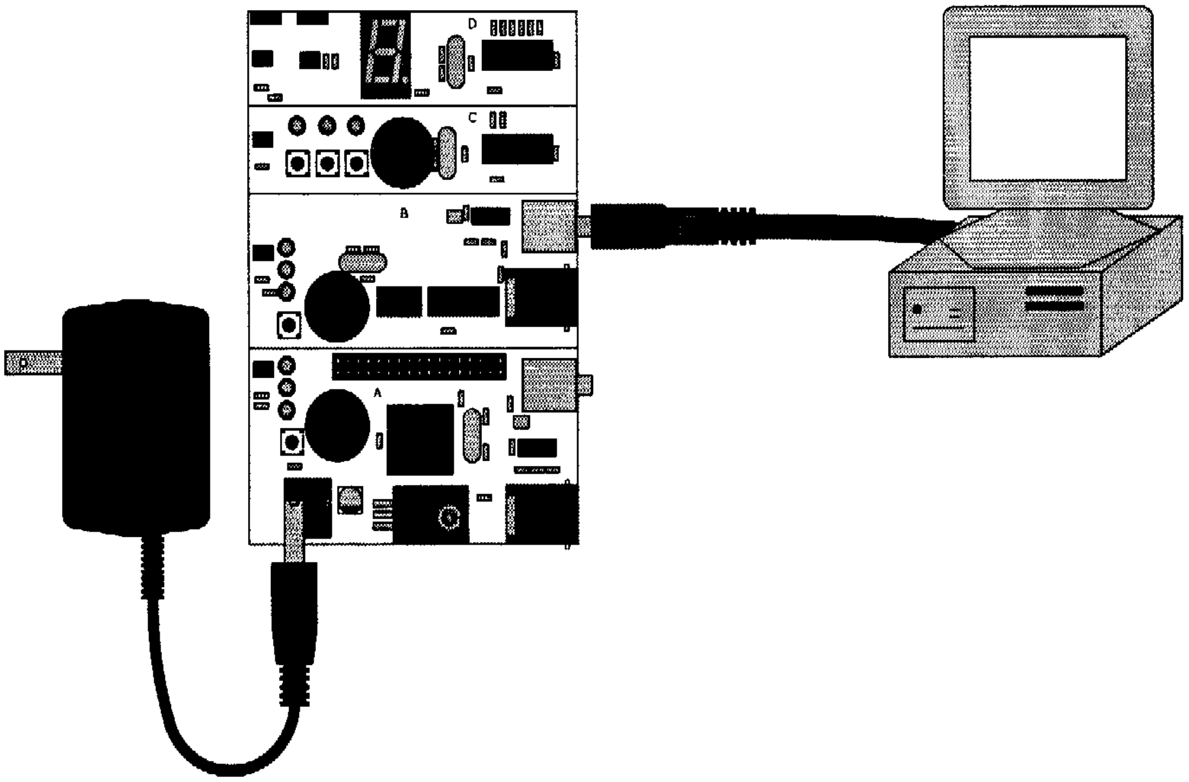
```
#include <ccscana.c>

void main() {
   int buffer[8],rx_len,rxstat;
   int32 rx_id;

   can_init();

   can_putd(0x100,0,0,1,TRUE,FALSE);  //send an on-bus message
                                      //to wake up mcp250x0's
   delay_ms(1000);                    //wait for node c to power-up

   buffer[0]=0x2C;
   buffer[1]=0xFF;
   buffer[2]=0xD7; // Sched ON, For READ ADC, clock *4096 *16 * 7
   can_putd(WRITE_REGISTER_C_ID, buffer, 3, 1, TRUE, FALSE);

   while(TRUE) {
      if ( can_kbhit() ) {
         if(can_getd(rx_id, &buffer[0], rx_len, rxstat)) {
            if (rx_id == NODE_C_SCHEDULED) {
               write_7_segment(buffer[2]/26);
            }
         }
      }
   }
}
```

10 A CAN Bus Monitor

- ❑ The following program is intended for Node B. It will take all frames from the CAN bus and send them over RS-232 link. A PC must be connected to the RS-232 port to view the data. Use the SIOW program to view the data on the RS-232 port.

```
#include <16F876A.H>
#fuses HS,NOPROTECT,NOLVP,NOWDT
#use delay(clock=2500000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

#include <can-mcp2510.c>

void main() {
   int32 rx_id;
   int i, rx_len, buffer[8];
   struct rx_stat rxstat;

   can_init();

   while(TRUE) {
      if ( can_kbhit() ) {
         if(can_getd(rx_id, &buffer[0], rx_len, rxstat)) {
            printf("%LX:  (%U) ",rx_id,rx_len);
            if (!rxstat.rtr) {
                  for(i=0;i<rx_len;i++)
                     printf("%X ",buffer[i]);
            }
            if (rxstat.rtr) {printf(" R ");}
            if (rxstat.err_ovfl) {printf(" O ");}
            if (rxstat.inv) {printf(" I ");}
            printf("\r\n");
         }
      }
   }
}
```

- ❑ Enter, compile and load this program into Node B. Load the EX8A.C program into Node A.
- ❑ Notice the CAN bus activity between Nodes A and C are mentioned and reported over the RS-232 port.



Sample Output:

```
00000300:  (8)  R
00000401:  (0)
00000301:  (0)
00000303:  (2) 40 3E
00000300:  (3) 1E 04 FF
00000300:  (3) 1E 08 FF
00000303:  (2) 40 3C
00000300:  (3) 1E 04 FF
00000300:  (3) 1E 08 FF
```

11 Data Filtering

- Looking at the previous program it is clear that the processor must spend time reading every frame on the CAN bus. This processing time is spent even though that node only has interest in one message type. With a large number of nodes on the CAN bus, this can cause considerable wasted processing time. The solution is to get the CAN bus controller hardware to filter the data and only bother the microcontroller with data that is of interest. The following are several popular methods for filtering.

  - **BCAN – Basic CAN**

    The system is designed such that various bits in ID are used to group common frames together. A mask and reference ID are programmed into the CAN bus controller. If (FRAME_ID & MASK) == REF_ID then the frame is saved for the microcontroller, otherwise it is discarded. It is common in a BCAN controller to assign a priority to outgoing frames. This way as the controller waits for bus time messages can be sorted.

    Advanced variations of BCAN can allow multiple masks and reference IDs to be specified.

    BCAN is the scheme used on the Microchip CAN controllers. Microchip has two buffers. One allows a mask and two reference IDs. The other allows a mask and four reference IDs.

  - **FCAN – Full CAN**

    A list of all possible IDs of interest to the microcontroller is programmed into the CAN controller. A buffer is allocated in the controller for each ID. The microcontroller can then poll for data by checking buffers of interest or program certain ID's to generate an interrupt. The same buffer scheme is used for outgoing frames. The FCAN controller can handle requests for a particular ID without microcontroller intervention.

    Consider the previous program. If we had a FCAN controller then instead of waiting for a message and then acting on it the software could just request the last frame for a given ID and use the data. The same data might be used over and over until it is replaced.

    Advanced variations of FCAN allow BCAN like masks to be applied to buffers.

• **DCAN – Direct CAN**

This is a hybrid approch where you get BCAN like masks and reference Ids, FCAN like indivisual receive buffers, and a BCAN like transmit buffer.

• **TTCAN – Time Triggered CAN**

The bus bandwidth is split into time slots. Specific frame ID's are assigned to certain timeslots. This limits the frequency for the data and helps nodes to know when to be looking for data.

❑ The following program will set up filtering on the Node B data monitoring program. We will set themask and referene ID to only monitor data to Node D. Load EX9.C into Node A and EX10 (with the following additions) into Node B at the start of main():

```
 can_set_mode(CAN_OP_CONFIG);    //must be in config mode
                                //before params can be set
can_set_id(RX0MASK,0xFF00,TRUE);
can_set_id(RX0FILTER0,0x400,TRUE);
can_set_id(RX0FILTER1,0x400,TRUE);

can_set_id(RX1MASK,0xFF00,TRUE);
can_set_id(RX1FILTER2,0x400,TRUE);
can_set_id(RX1FILTER3,0x400,TRUE);
can_set_id(RX1FILTER4,0x400,TRUE);
can_set_id(RX1FILTER5,0x400,TRUE);
can_set_mode(CAN_OP_NORMAL);
```

12 Getting Off the Prototyping Board

❑ PHYSICAL

As previously noted, there is no standard physical interface. The PCA82C251 chips used on the prototype board use a popular two- wire CAN bus. Connections can be made directly from the prototyping board to an external CAN bus via the three pin connector at the top of the board (CANL, CANH and Ground). When using this connection over some distance, a 120 ohm resistor should be put on both ends of the bus. This driver chip can handle up to 110 nodes and a total bus length of 100 feet. The bus can be much longer if a slow bit time is used.

An extra driver chip has been installed on the prototype board. This allows for an easy connection to an external CAN controller that has TTL output. The three pin connection has Transmit, Receive and Ground connections to the spare PCA82C251 chip.

Some CAN Transceivers

		Nodes	**Speed**	**Fault Tolerant**	
Philips	PCA82C251	110	1 meg	NO	
	PCA82C252	15	125k	YES	
	TJA1054	32	125k	YES	Low EMC
Maxim	MAX3058	32	1 meg	NO	
	MAX3050	32	2 meg	NO	
	MAX3054	32	250k	YES	
TI	SN65LBC031		500k	NO	
	SN65HVD251	120	1 meg	NO	
	SN65HVD232	120	1 meg	NO	3.3V

❑ TIMING

All nodes on the bus must have the same target bit time. The fastest time allowed by the PCA82C251 is 1 million bits per second.

A single bit time is divided into four segments:

Sync period
Propagation period (allow for delays between nodes)
Phase 1 period
Phase 2 period

The data is sampled for the bit between phase 1 and phase 2.

Each of the four segment times may be programmed in terms of a base time (Time Quanta or Tq).

The baud rate settings are made in the .h files (like can-18xxx8.h). We have made the following settings:

Sync period = 1 Tq
Propagation period = 3 Tq
Phase 1 period = 6 Tq
Phase 2 period = 6 Tq

The total bit time is therfor 16 Tq.

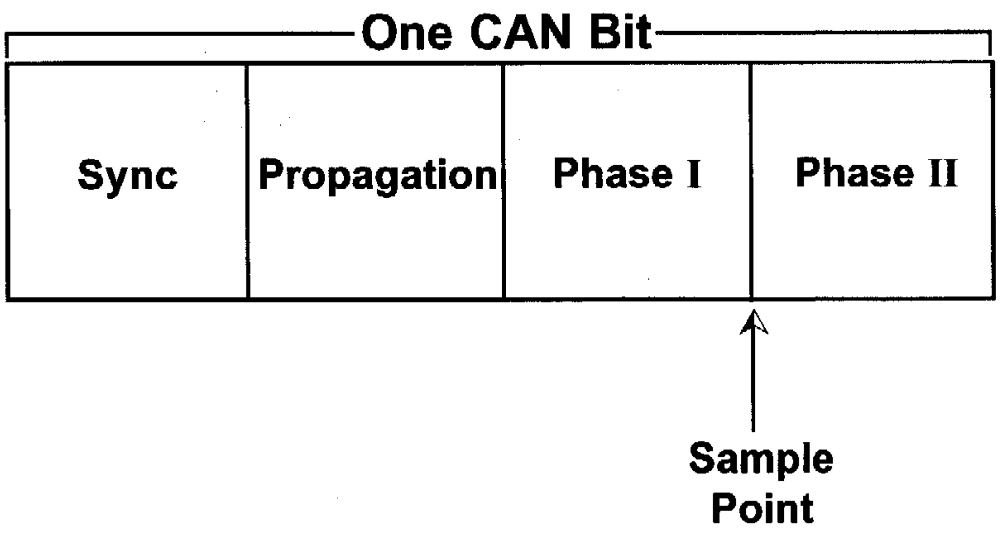
Tq is set via the prescaller. The formula is:

Tq = (2 x (prescaller+1))/clock

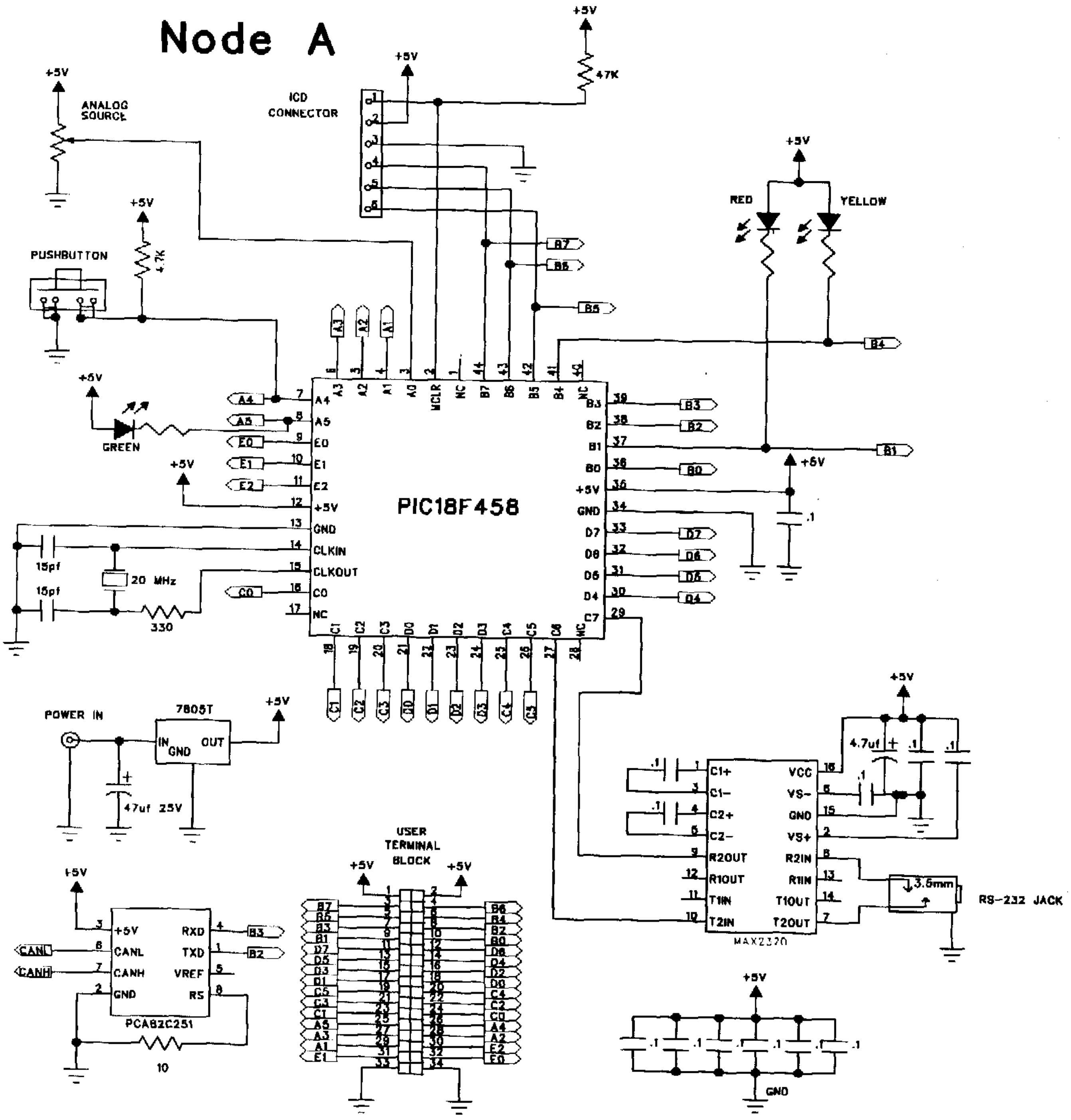
We use a clock of 20 mhz and have the prescaller set to 4. Therefor:

Tq = (2 x (4+1))/20000000 = 0.1 us

And the bit time is 1.6 us or 125K.

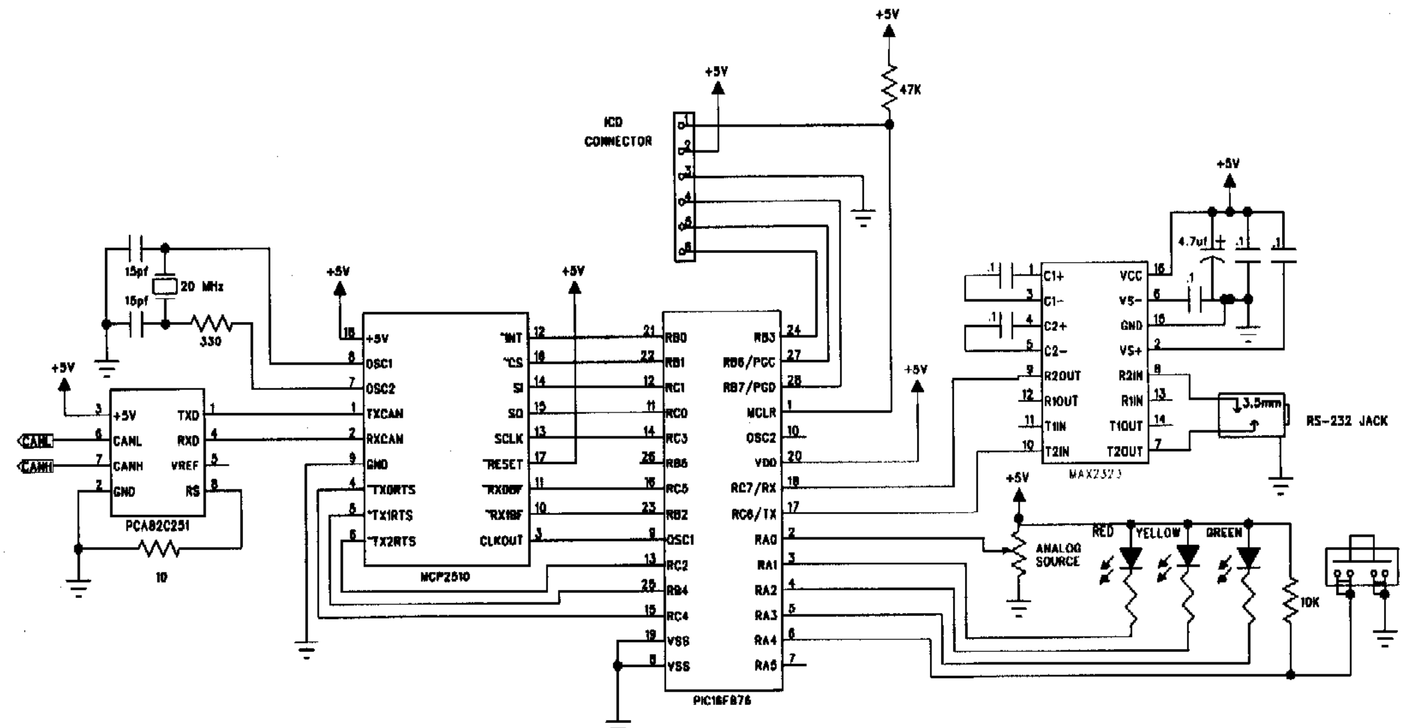


Node A



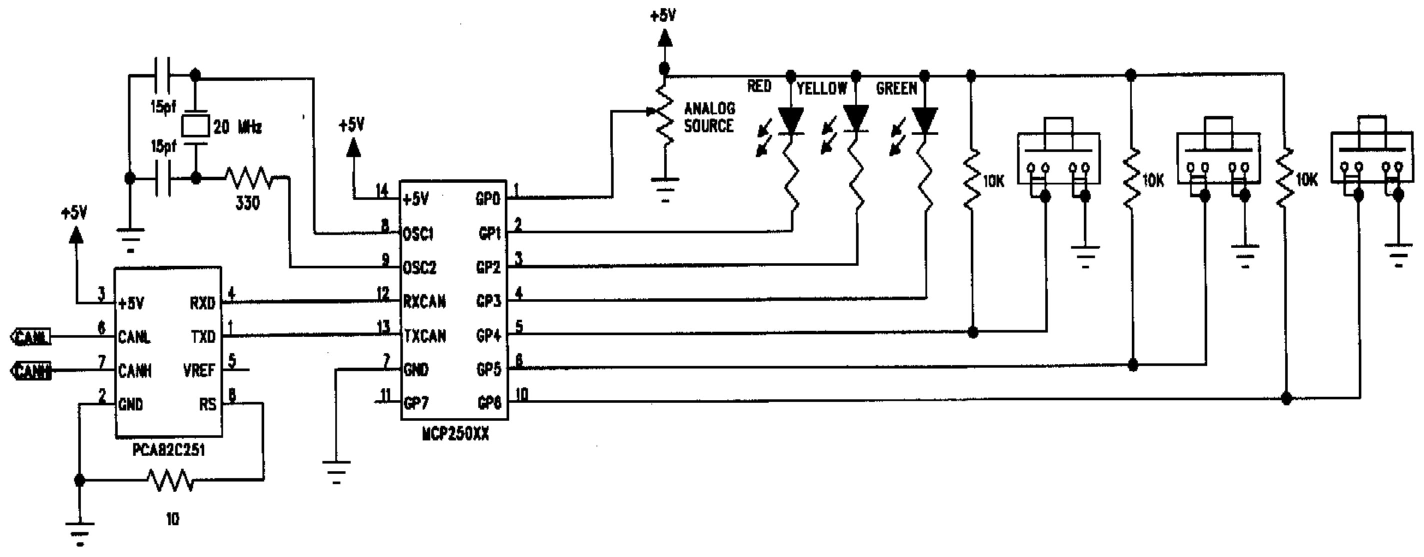
ANALOG SOURCE
ICD CONNECTOR
PUSHBUTTON
+5V
47K
4.7K
RED
YELLOW
GREEN
PIC18F458
15pf
20 MHz
330
POWER IN
7805T
IN
GND
OUT
47uf 25V
4.7uf
USER TERMINAL BLOCK
PCA82C251
+5V
CANL
CANH
GND
RXD
TXD
VREF
RS
10
MAX232O
C1+
C1-
C2+
C2-
R2OUT
R1OUT
T1IN
T2IN
VCC
VS-
GND
VS+
R2IN
R1IN
T1OUT
T2OUT
3.5mm
RS-232 JACK
GND
MCLR
NC
CLKIN
CLKOUT

Node B



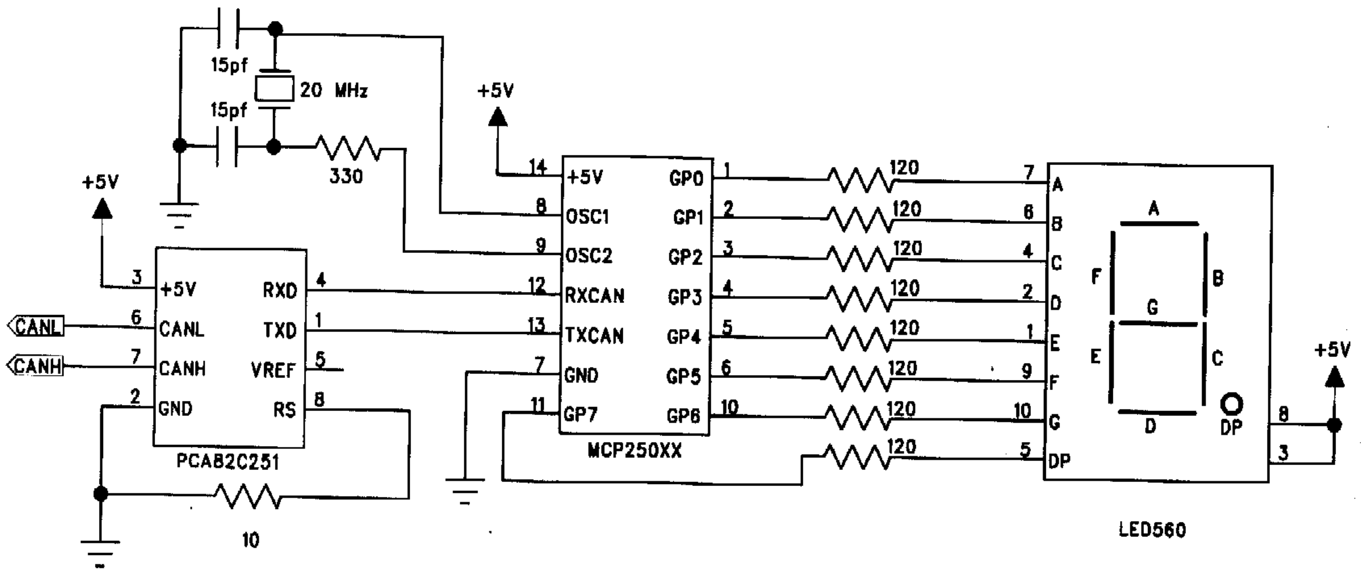
ICD CONNECTOR
+5V
47K
15pf
20 MHz
330
PCA82C251
10
MCP2510
PIC16F876
MAX232
4.7uf
.1
3.5mm
RS-232 JACK
ANALOG SOURCE
RED
YELLOW
GREEN
10K
CANL
CANH

Node C

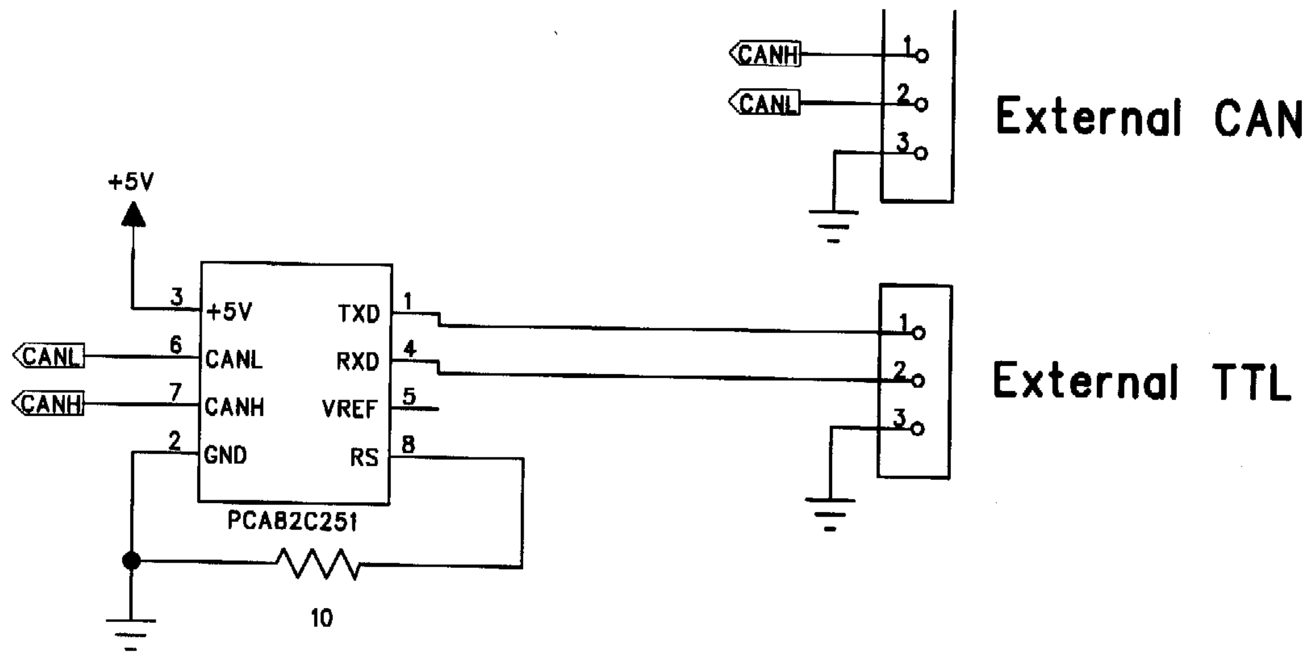


+5V
15pf
20 MHz
330
PCA82C251
10
MCP250XX
ANALOG SOURCE
RED
YELLOW
GREEN
10K
CANL
CANH

Node D



External



RTHER REFERENCES:

e official CAN website:	http://www.can.bosch.com/
nline CAN Tutorials:	http://www.kvaser.com/can/edu/
st of CAN resources:	http://www.can-cia.de
N Forum:	http://groups.yahoo.com/group/CANbus/messages
nufacturers of CAN Bus nitors and performance tools:	http://www.intrepidcs.com/mcp2510/PIC18F452