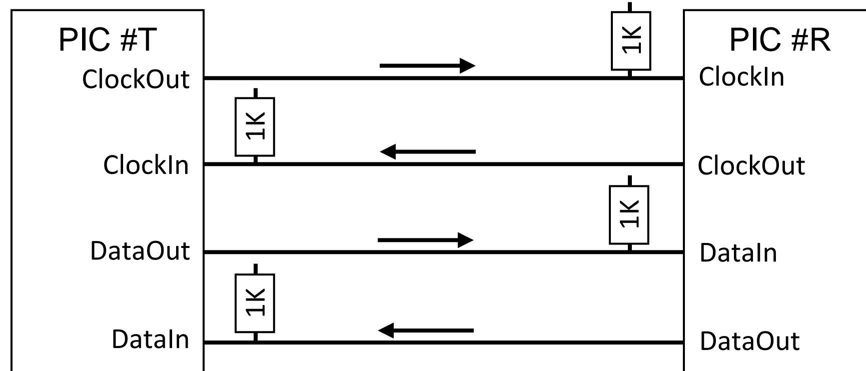
PIC-to-PIC communication test for my post-War Lionel control system

Communication between two PICs uses four dedicated wires. The communication is completely symmetrical; neither PIC is master or slave. If the physical distance between the PICs is short, they can be connected to each other without line drivers, as shown here. I have used arbitrary names PIC #T and PIC #R for these two PICS.



Each PIC has a ClockOut line which it keeps permanently configured for output. Each PIC's ClockOut line is the other PIC's ClockIn line, which are kept permanently configured for input. To guard against floating inputs, the ClockIn lines are tied high at their receiving ends by 1K resistors. Similarly, there are two data lines. Each PIC keeps its DataOut and DataIn lines permanently configured for output and input, respectively, with 1K pull-up resistors on the input ends.

Data is transmitted in packets containing 24 bits. Although the PICs store each packet as three separate bytes, the 24 bits in each packet are sent in one continuous stream.

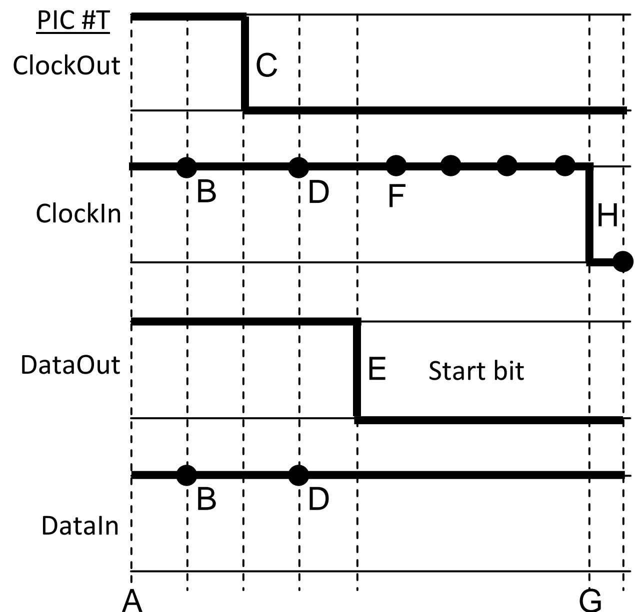
In general, PIC #T and PIC #R will be doing different jobs, so their main programs will be different. However, for communication purposes, the subroutines which transmit a packet and receive a packet are identical in the two PICs. In fact, the very same program is burned into the two chips for the test described below.

Either PIC can initiate a transmission by pulling its ClockOut line low. The main programs in both PICs poll their ClockIn lines continuously to detect the start of a transmission by the other PIC. Interrupts are not used to detect the start of a transmission.

Start of a communication

The following timing diagram shows the four communication lines at the start of a communication. All four lines are high. For the purposes of the following description, I will assume that it is PIC #T that wants to send a packet and that the other PIC (PIC #R) is the one that has to receive it. ("T" is for transmit and "R" is for receive.)

The labels along the left axix are the names of the four lines as stated from the point-of-view of transmitting PIC #T. Receiving PIC #R will use the opposite names to refer to these same four lines.



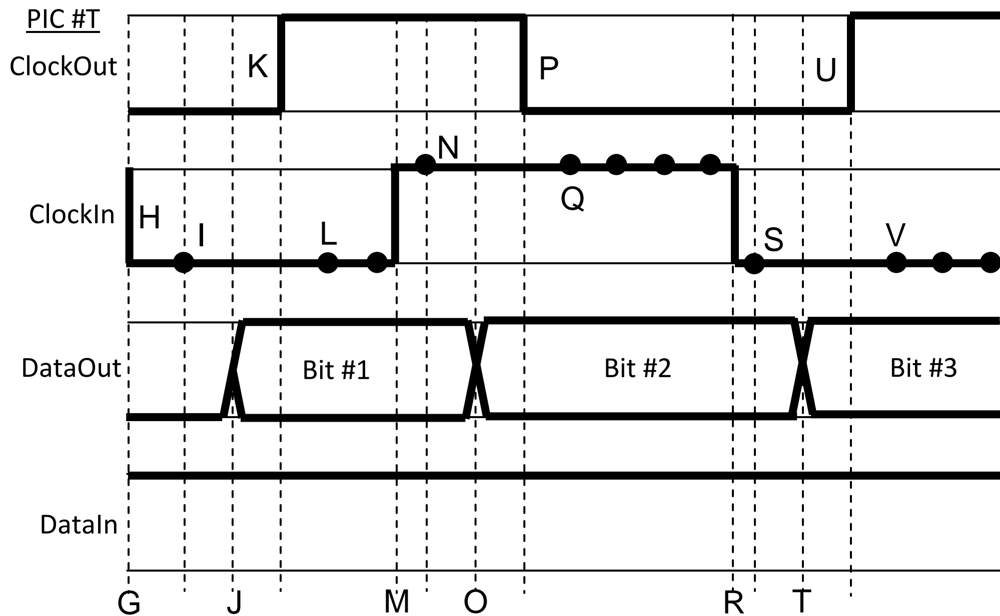
At time A, transmitting PIC #T decides to send the packet. PIC #T must first verify that its ClockIn and DataIn lines are high (event B). If either of these lines is low, then PIC #T aborts the transmission. If both lines are high, then PIC #T pulls its ClockOut line low (high-to-low transition C).

Then, PIC #T checks its ClockIn and DataIn lines again (event D), to confirm that both lines are still high. It is not impossible that both PICs might decide to send packets at exactly the same instant and that both would pull their ClockOut lines low at exactly the same time. It is not obvious how the two PICs could sort out who should transmit first. Therefore, the transmission subroutine is coded so that failure of test D is treated as a failure of the entire transmission. If that happened, PIC #T would immediately reset its ClockOut line high, and pass control back to the calling routine which ordered the transmission in the first place. (Note that both PICs are transmitting PICs if this conflict occurs.) Depending on the exact details of the timing, one or the other or even both PICs may pass control back to their calling routines. Since the calling routines in the two PICs will be doing different things, it is unlikely that a conflict will occur a second time once the calling routines get around to re-transmitting the failed packets.

If test D succeeds, then PIC #T pulls its DataOut line low to begin the Start bit (transition E) and begins to poll its ClockIn line (event F, repeated). At some time (time G, say), receiving PIC #R will have detected the start of the transmission and will pull its output clock line low (high-to-low transition H). PIC #T will detect this transition at the next poll of its ClockIn line (event I).

Transmission of the first two data bits

After PIC #R acknowledges the Start bit by pulling its output clock line low (transition H), the two PICs are in agreement about who is transmitting and who is receiving. The next timing diagram shows the four communication lines as the first two data bits are transmitted.



As quickly as it can after detecting ClockIn low (event I), PIC #T sets the first data bit (the LSB of the packet) onto its DataOut line (time J). It then sets its ClockOut line high (low-to-high transition K) and begins to poll its ClockIn line (event L, repeated).

Receiving PIC #R will be waiting for transition K, upon receipt of which it will read its DataIn line. Once it has read the data, PIC #R will set its ClockOut line high (time M).

Eventually, PIC #T's event L polling will bear fruit, and it will detect low-to-high transition M (event N). As quickly as it can after detecting ClockIn high, PIC #T sets the second data bit onto its DataOut line (time O). It immediately pulls its ClockOut line low (high-to-low transition P) and begins to poll its ClockIn line (event Q, repeated).

Receiving PIC #R will be waiting for high-to-low transition P, upon receipt of which it will read its DataIn line. Once it has read this second data bit, it will pull its ClockOut line low (time R).

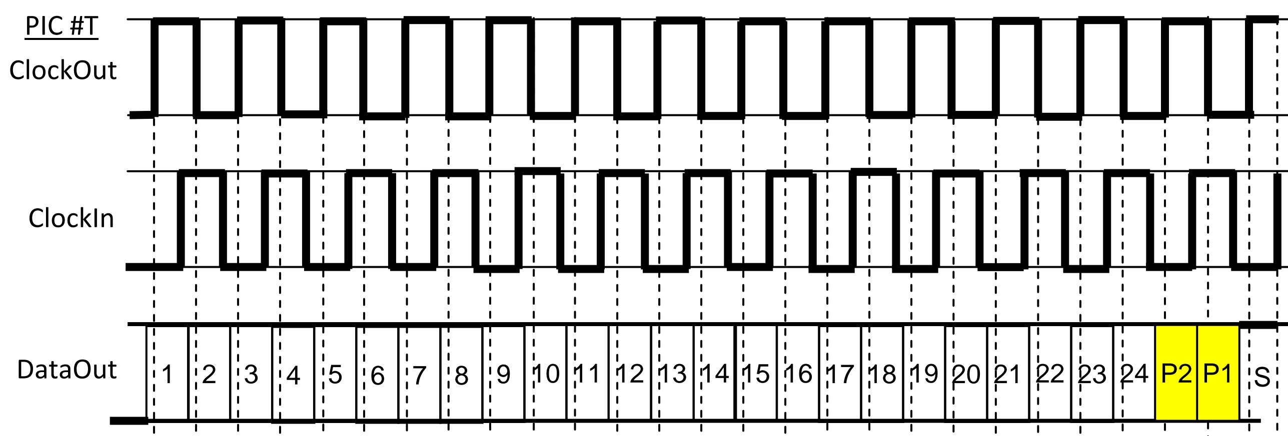
Eventually, PIC T's event Q polling will bear fruit, and it will detect high-to-low transition R (event S). As quickly as it can after detecting ClockIn low, PIC #T sets the third data bit onto its DataOut line (time T). It immediately sets the ClockOut line high (low-to-high transition U) and begins to poll its ClockIn line (event V, repeated). And so on for the remaining 21 data bits in the packet.

The lengths of these pulses is not fixed. Pulse length is determined by how quickly each PIC can carry out the necessary bit checks, reads or writes, and then toggle the state of its ClockOut line.

An interesting feature of this protocol is that the polarity of the clock pulses is reversed from each bit to the next. After setting the data for bits 1, 3, 5 and the following odd numbered bits, PIC #T sets its ClockOut line high. But after setting the data for bits 2, 4, 6 and the following even numbered bits, PIC #T pulls its ClockOut line low. Similarly, after PIC #R reads a data bit, it marks completion by toggling the state of its ClockOut line, either high-to-low or low-to-high.

Transmission of two parity bits and a stop bit

Two parity bits are transmitted immediately after the 24 data bits. The two parity bits are highlighted in yellow, and labeled P2 and P1, in the following timing diagram. The two parity bits use exactly the same clock handshake sequence as the 24 data bits. I have not shown PIC #T's DataIn line in this timing diagram -- as above, it remains high throughout this period.



A single Stop bit (S) is sent after the two parity bits. The Stop bit must be high, as indicated by the heavy line on the DataOut stream after the parity bits. It, too, continues the clock handshake sequence from before.

Parity is even, and the two parity bits are the two low-order bits in the cumulative count of high data bits in a packet. For example, if a 24-bit packet contains 13 high bits, then the parity count (presumably stored in an 8-bit register) after sending the 24 bits will be d'13', or b'00001101'. The two low-order parity bits are P1 = b'1' and P2 = b'0'. Note that P2 is transmitted before P1. (P2 is transmitted before P1 merely because it is slightly more convenient for the receiving subroutine to receive the two bits in this order.)

The timing diagram above ends at the instant when the receiving PIC #R has read the Stop bit and raised its output clock line (PIC #T's ClockIn line) high. Because an odd number of bits has been sent so far (27) the transmitting PIC's output clock line (PIC

#T's ClockOut line) will be at a high level. Since the Stop bit is still on the data line, PIC #T's DataOut line will also be at a high level.

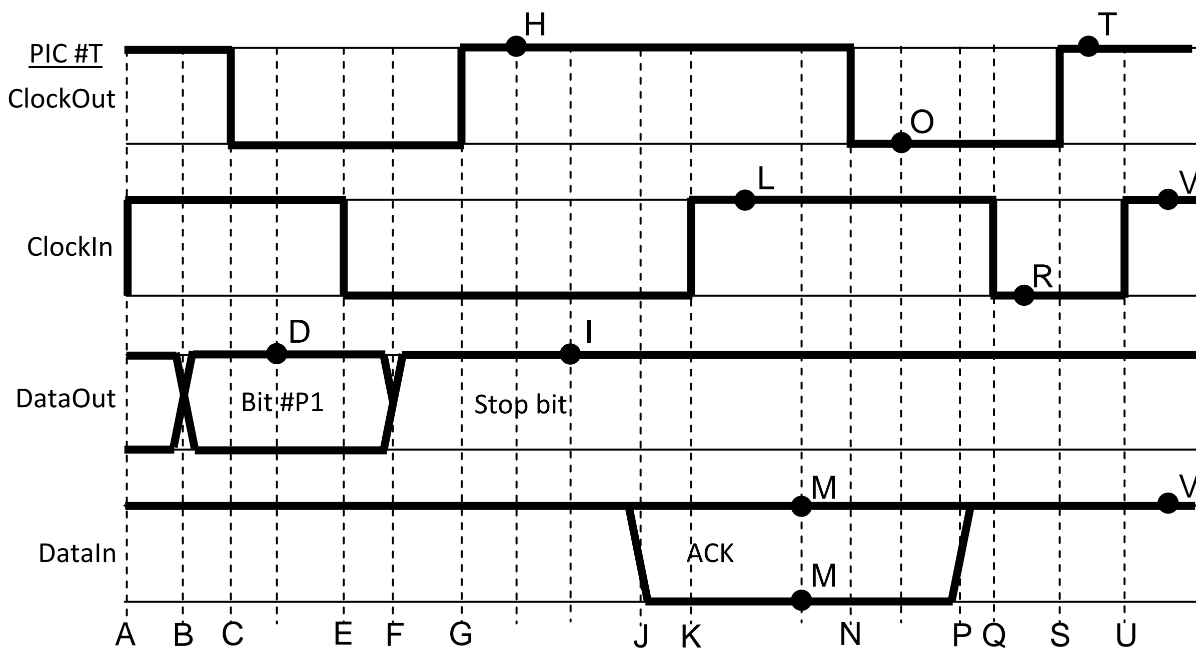
The Acknowledgement sequence

The Acknowledgement sequence at the end of a packet is intended to do more than simply allow receiving PIC #R to tell transmitting PIC #T that it received the packet. The sequence also allows the receiving PIC to tell the transmitting PIC whether or not the packet was received correctly.

The following timing diagram shows in detail what happens, beginning with the end of parity bit P2.

At time A (I will start a new alphabet of numbering), receiving PIC #R has finished reading parity bit P2 and has raised its output clock line (PIC #T's ClockIn line) high. As soon as it detects that transition, PIC #T sets its DataOut line to the value of parity bit P1 (time B). It then pulls its ClockOut line low (time C).

As soon as receiving PIC #R detects high-to-low clock transition C, it will read parity bit P1 (event D), after which it will pull the ClockIn line low (time E). When PIC #T detects ClockIn low, it will set the Stop bit onto the DataOut line (time F) and then raise its ClockOut line high (time G). The Stop bit is always high.



As soon as receiving PIC #R detects ClockOut high (event H), it will read the Stop bit (event I) and confirm that it is high. By comparing the two parity bits with its own internal count of the parity, receiving PIC #R will then determine whether or not the packet it has received is valid. It will set its DataOut line (PIC #T's DataIn line) to the

appropriate value (time J): low if the packet is valid and high if it is not. After having set its DataOut line, PIC #R will set its ClockOut line high (time K).

As soon as it detects ClockIn high (event L), PIC #T will read the Acknowledgement bit (event M) and then pull its ClockOut line low (time N).

When PIC #R detects this high-to-low clock transition (event O), it will react by setting its DataOut line high (time P), thus ending the Acknowledgement bit. PIC #R will then pull its output clock line low (time Q).

When transmitting PIC #T detects PIC #R's ClockOut transition (event R), it will raise its ClockOut line from low to high (time S). This ends the communication from the point-of-view of transmitting PIC #T.

When receiving PIC #R detects PIC #T's ClockOut transition (event T), it will raise its ClockOut line from low to high (time U). This ends the communication from the point-of-view of receiving PIC #R.

PIC #T will not be allowed to start a new transmission until it verifies that both of its ClockIn and DataIn lines are high (event V).

The Acknowledgement bit is set low for a successful communication and high for a failure. If things go wrong during reception, receiving PIC #R can simply stop processing and leave its DataOut line high, which transmitting PIC #T will eventually detect and interpret as a failed communication.

Failures arise for four reasons:

1. Something happens which causes one or the other PIC to miss an input clock transition. This cannot be detected directly, but will lead to one or more of the following errors.
2. Parity error
3. Stop bit is not high
4. Time-outs

The test program listed in Appendix "C" below detects 24 different errors. The error code gives a clue about how much progress was made sending or receiving with the packet before failure occurred.

```
; Error codes while receiving a packet:-
;   Code 0x01: 1.0ms TO while waiting for an odd Data bit to become ready
;   Code 0x02: 1.0ms TO while waiting for an even Data bit to become ready
;   Code 0x03: 1.0ms TO while waiting for Parity bit #2 to become ready
;   Code 0x04: 1.0ms TO while waiting for Parity bit #1 to become ready
;   Code 0x05: 1.0ms TO while waiting for the Stop bit to become ready
;   Code 0x06: The Stop bit is low
;   Code 0x07: Parity error
```

```
;   Code 0x08: 1.0ms TO while waiting for Acknowledge bit to be read
;   Code 0x09: 1.0ms TO while waiting for final Clock transition
;   Code 0x0A: Reserved for possible future use
;
; Error codes while sending a packet:-
;   Code 0x0B: Receiver is not ready - ClockIn line is low on first test
;   Code 0x0C: Receiver is not ready - Data line is low on first test
;   Code 0x0D: Receiver is not ready - ClockIn line is low on second test
;   Code 0x0E: Receiver is not ready - Data line is low on second test
;   Code 0x0F: 20ms TO while waiting for receiver to set Clock low
;   Code 0x10: 1.0ms TO while waiting for an odd Data bit to be read
;   Code 0x11: 1.0ms TO while waiting for an even Data bit to be read
;   Code 0x12: 1.0ms TO while waiting for Parity bit #2 to be read
;   Code 0x13: 1.0ms TO while waiting for Parity bit #1 to be read
;   Code 0x14: 1.0ms TO while waiting for Acknowledgement bit to become ready
;   Code 0x15: 1.0ms TO while waiting for penultimate ClockIn transition
;   Code 0x16: Acknowledgement bit is high
;   Code 0x17: Reserved for possible future use
;
; Error codes during PICA processing:-
;   Code 0x18: Round-trip error in low-order packet byte; XOR is not all ones
;   Code 0x19: Round-trip error in middle packet byte; XOR is not all ones
;   Code 0x1A: Round-trip error in high-order packet byte; XOR is not all ones
```

Time-out detection

The duration of individual bits is not monitored. Instead, the time elapsed since the start of the communication is monitored. If, for any reason, the communication is not completed within the prescribed interval, the communication is deemed to have failed.

The length of the prescribed interval depends on the PIC's clock speed and, perhaps more importantly, on the length and quality of the cable which connects them. The test program uses one millisecond as the prescribed interval. Time-outs are detected using Timer0 interrupts, and the Timer0 module is configured to generate an interrupt one millisecond after the Timer0 count register is reset. (Note that the Interrupt Service Routine has only one very small job -- it sets one bit in a User-defined register. The Interrupt Service Routine does not play any active role in the communication.)

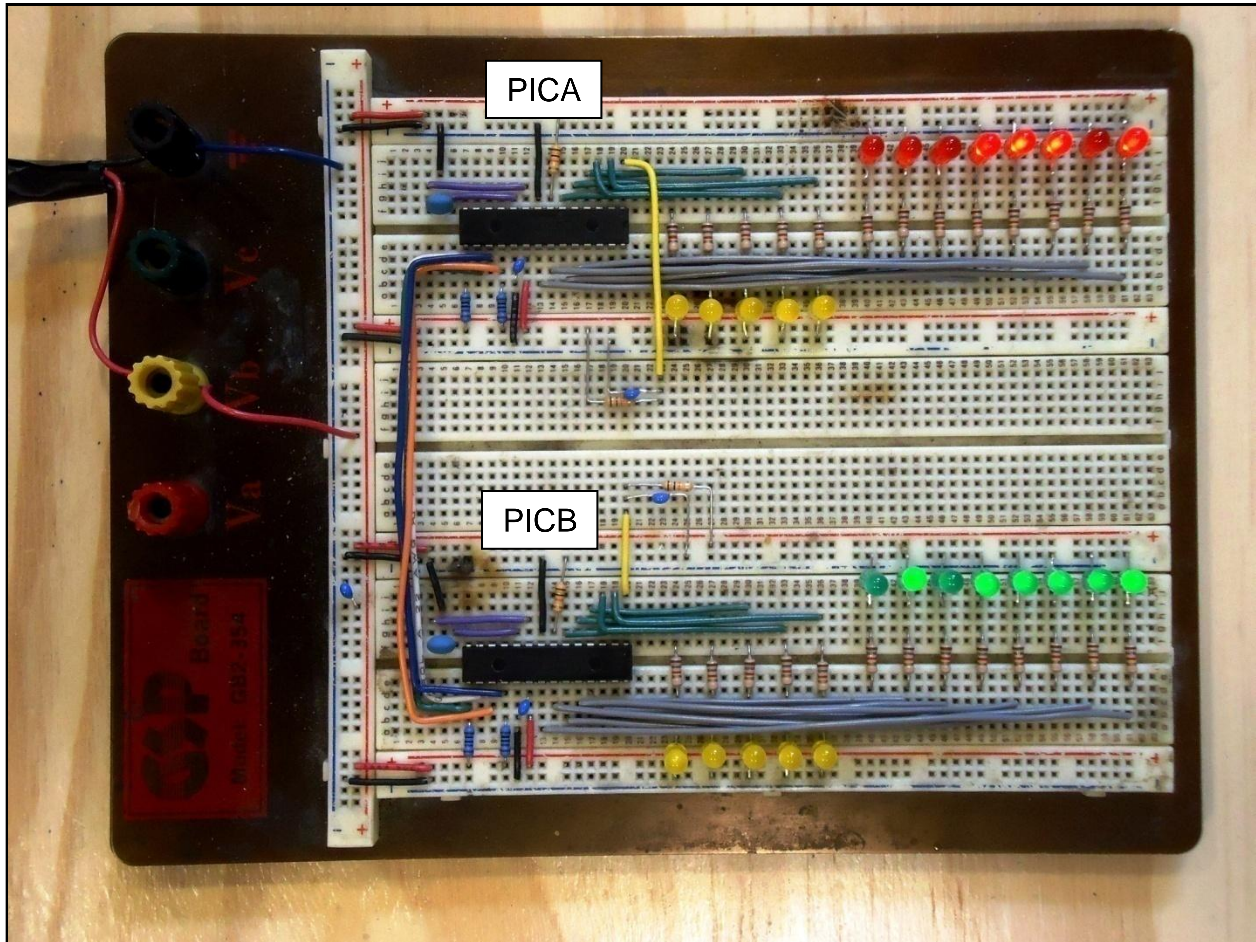
In fact, there are two different types of time-out. The one I have just applies to the period of time after receiving PIC #R recognizes that a communication has started and pulls its ClockOut line low. Once the receiving PIC responds, the stream of bits is sent quite quickly.

The other time-out applies to the period of time between transmitting PIC #T pulling its ClockOut line low to signal its intention to send a packet and receiving PIC #R responding by pulling its Clockout line low. This interval is application specific -- it depends on how busy the receiving PIC's main program is. The test program uses 20 milliseconds as the prescribed interval for this type of time-out. (Note that one could use the ClockIn line to trigger an interrupt, perhaps by tying it to the RB0 pin. Then, the reception process could be carried out within the Interrupt Service Routine itself.

However, there are other dangers and drawbacks to using the ISR to handle communications directly.)

Breadboard for the test program

Testing is needed to ensure that the timing is not "too fast" for the connecting cable. My preliminary test used the following breadboard, whose schematic is given in Appendix "A" below.



The test rig has two PIC16F882s, labeled PICA and PICB in the photograph. In the test, the two PICs send packets back and forth. The algorithm to create and verify packet contents is the following.

PICA begins the process. It uses three 8-bit User-registers, named MasterCount0, MasterCount1 and MasterCount2, to hold a 24-bit integer which is the packet to be transmitted. MasterCount is initially set to zero, which is to say, b'0000 0000 0000 0000 0000 0000' and this packet is transmitted to PICB.

PICB is programmed to receive a 24-bit packet, complement it and transmit it back to PICA. When PICB receives the first packet, it is twos-complemented into b'1111 1111 1111 1111 1111 1111' and transmitted back to PICA.

PICA can easily verify the round trip by exclusive-ORing the packet it transmitted with the packet it received back. The XOR value for every round trip will be 0xFFFFFF.

Then PIC increments MasterCount, whose new value will be 0x000001. PICB will complement it and transmit back 0xFFFFFE. The third pair will be 0x000002 and 0xFFFFFD, and so on. In every case, the result of PICA's XOR operation on the packet it sent and the packet it received back should be 0xFFFFFF.

Both PICs have the same setup for displays. Eight LEDs are wired to portB. Every time either PIC sends a packet, it displays eight bits of the packet on portB. PICA's display LEDs are red; PICB's are green.

Each PIC also has five yellow LEDs to display errors, wired to pins RC4-RC0. In the event of an error, the PIC displays a non-zero error code and then enters an infinite do-nothing loop with that error code displayed. Obviously, if one PIC detects an error and stops processing, the other PIC will encounter an error very quickly afterwards. (This would not be suitable in a production system, of course, when recovery from errors is paramount.)

Both PICs are programmed with the same Microchip Assembly code, which is listed in Appendix "C" below. The subroutines which send and receive packets are identical, but the main programs are different. Note in the photograph above that PICA's pin RB0 is tied high through a 10K resistor but PICB's pin RB0 is tied low through a 10K resistor. When the PICs are powered up, the initialization sequence reads the RB0 pin to determine which version of the main program this PIC should run.

In the photograph above, the portB LEDs of PICA (red) are showing b'00011101' and the portB LEDs of PICB (green) are showing b'01011111'. This means that the packet which has just been sent from PICA to PIC B is b'00011101xxxxxxxx01011111'. PICA displays the high-order byte of the packet, PICB displays the low-order byte. The middle byte is not displayed by either PIC. The program is about one-eighth of the way through. The test ends after packet 0xFFFFFF has made the round trip, after which PICA flashes 0x55 on its red portB LEDs.

Realized communication speed

It took 2 hours, 22 minutes and 7 seconds for the program to run to completion. That is a total of 8,527 seconds.

During this time, PICA sent 2^{24} = 16,777,216 packets to PICB. PICB sent one packet back for each packet received, so 33,554,432 packets went through the connecting

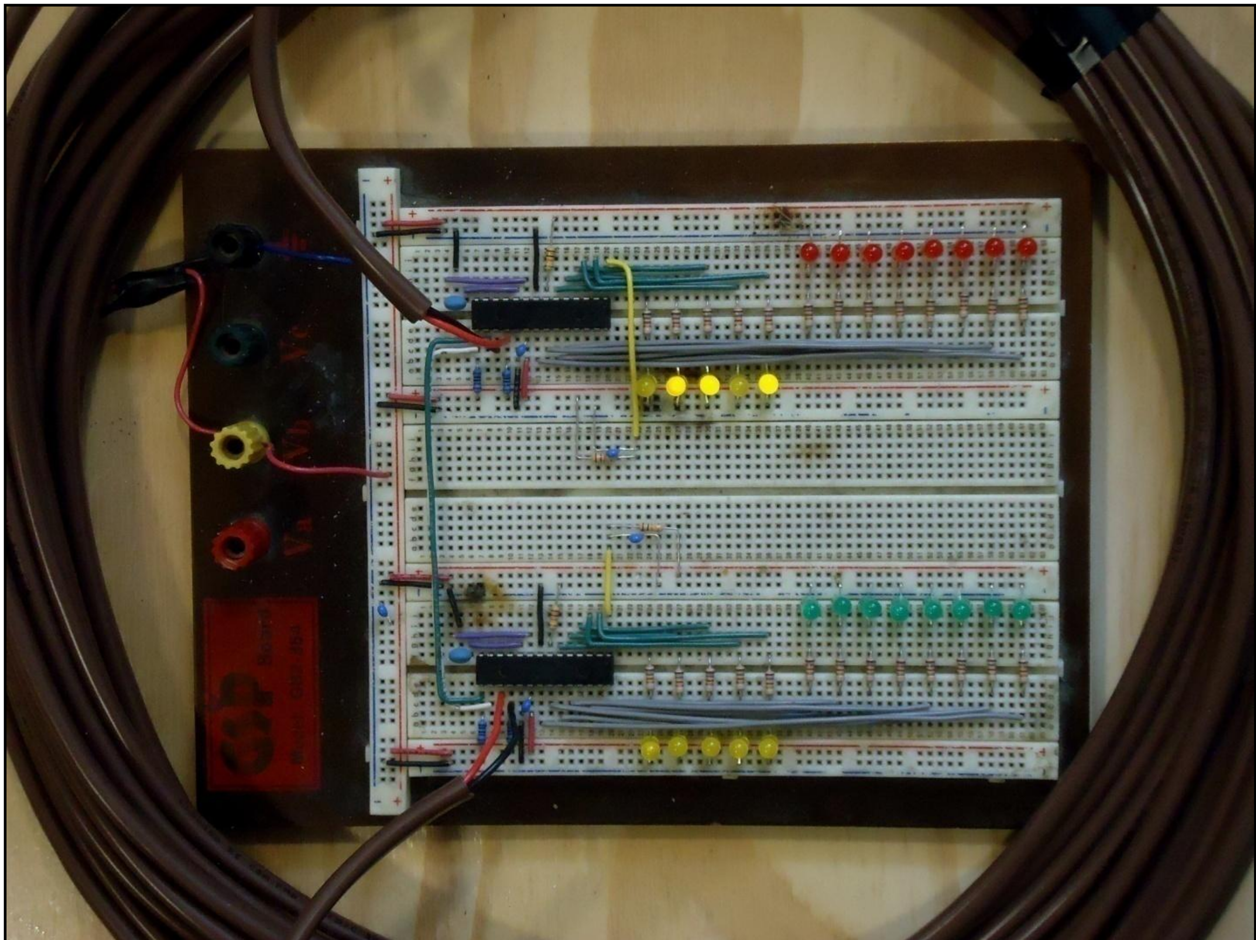
wires. Each packet contains 24 information bits, so a total of 24 * 33,554,432 = 805,603,368 information bits were transmitted.

The effective speed is calculated as 805,603,368 / 8,527 = 94,442 information bits per second. Stated in the reciprocal, this is 10.59 microseconds per information bit.

Note that this is the "realized" speed. It does not take into account the Start and Stop bits, the two parity bits and the Acknowledgement bit, all of which accompany each packet. This speed also includes the time taken by the main programs to process the packets received and prepare to send out reply packets.

Trying a longer, heavier cable

The following photograph shows what I tried next. I replaced two of the short hookup wires -- the ClockOut and DataOut lines of PICA -- with a 50-foot length of AWG18-gauge thermostat wire.



The communication failed. The LEDs of PICA show that the transmission of the very first packet (0x000000) failed with error code 0x0D (on the yellow LEDs). This error arises when the transmitting PIC (PICA in this instance) is trying to start a transmission.

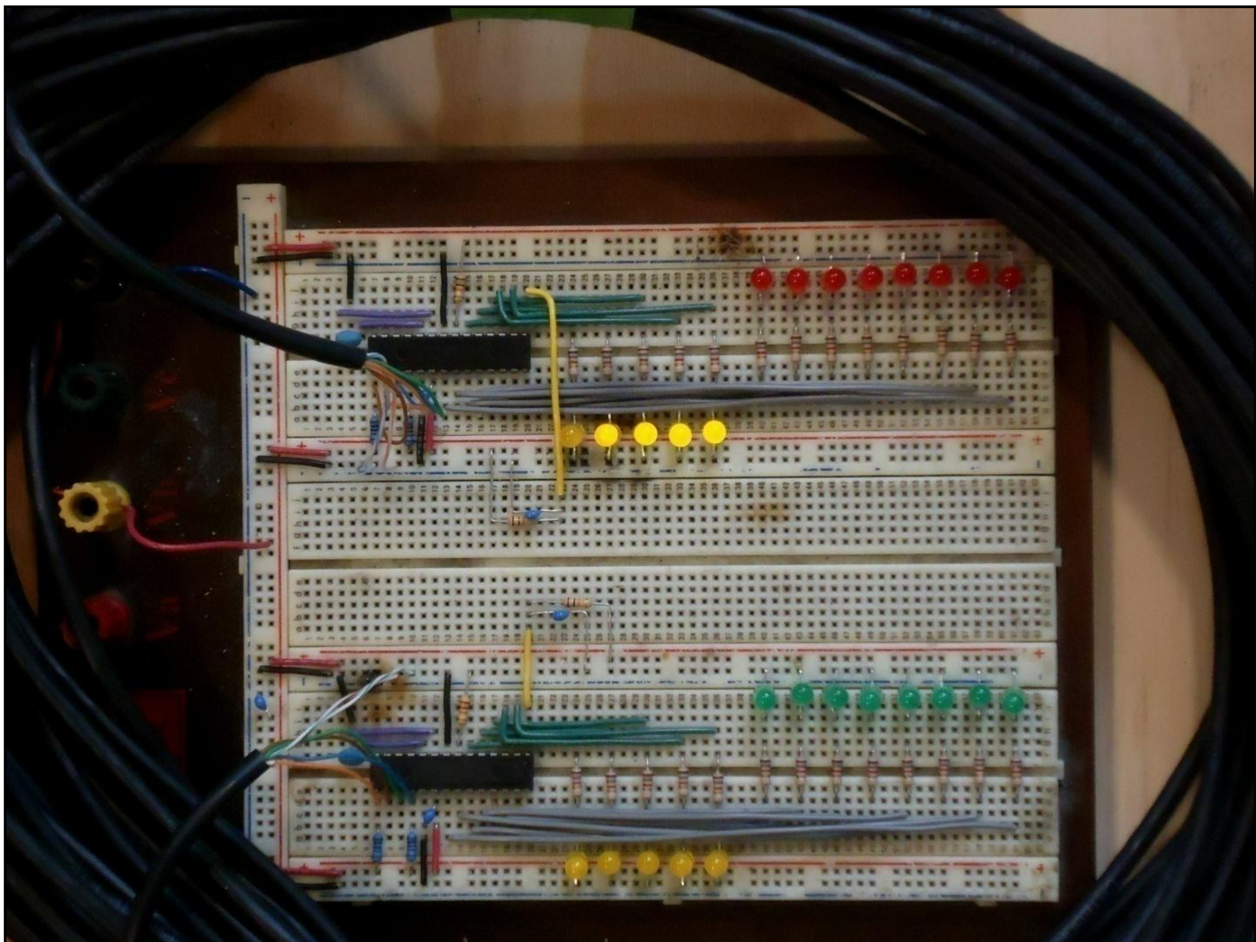
As described above, before doing anything, the transmitting PIC first checks to ensure that the ClockIn and DataIn lines from the other PIC are high. Only then does the transmitting PIC pulls its ClockOut line low. It immediately re-checks the ClockIn and DataIn lines to ensure that they are still high, and that the other PIC had not started its own transmission at exactly the same time.

And here is where things went wrong. When PICA read its ClockIn line after pulling its ClockOut line low, it found that the ClockIn line was now low. This is a spurious result.

What has happened is that the wires in the thermostat cable are just too big and too long for the PIC's output transistors to handle. The excessive load on the ClockOut pin has adversely affected the voltage on the neighbouring ClockIn pin.

Trying a 100-foot ethernet cable

The following photograph shows what I tried after that. I connected the two PICs with a 100-foot length of Cat5E cable. This was an unshielded, four-pair cable with AWG24-gauge solid copper conductors.



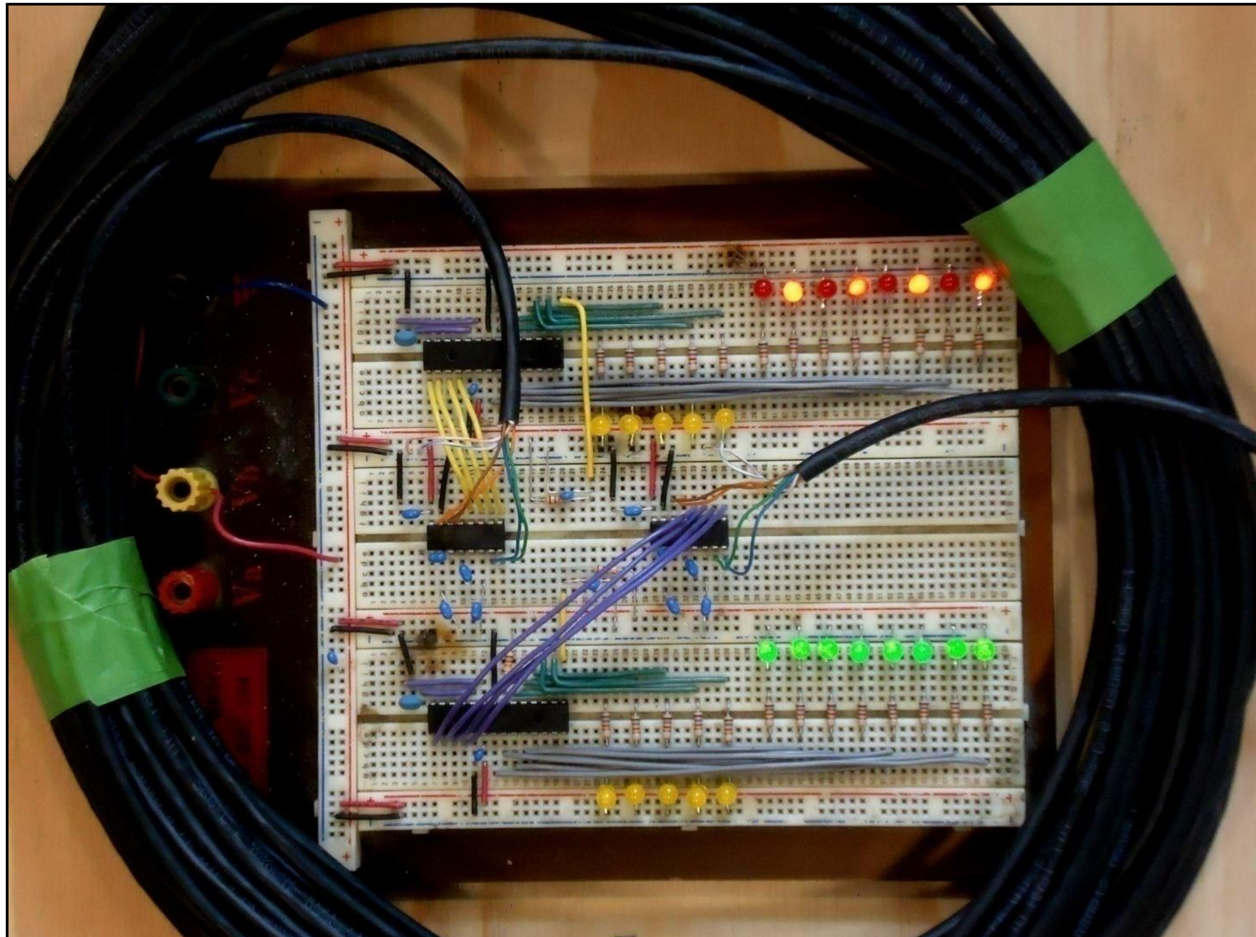
Once again, transmission of the very first packet failed. But error code 0x0F shows that the process did get a little bit further than before. This error code is a 20ms time-out after the transmitting PIC has pulled its ClockOut line low. In this case, PICA was able to verify that ClockIn and DataIn remained high after it pulled its ClockOut line low. But the receiving PIC did not reply -- it never pulled its ClockOut line low in response.

One must conclude that the cable was too long and too heavy for PICA's ClockOut high-to-low transition to propagate all the way to PICB. We know from the photograph that PICB did not receive anything -- it is not showing any error code, so it is still waiting for a packet to begin.

Using MAX232E line drivers

Failures with longer cables is not unexpected. The output circuits of the PIC's I/O pins are not intended to drive capacitive loads. For anything other than short pieces of hookup wire, line drivers must be used.

The MAX232E has been a workhorse of line drivers ever since line drivers were invented. The standard chip has two output channels and two input channels -- perfect for my 4-wire system. The MAX232E can be used with a standard 5V power supply. It uses external capacitors to generate the +8.5V and -8.5V voltages required by the RS232 protocol. The schematic with MAX232E drivers wired in is set out in Appendix "B" below. The following photograph shows the breadboard.



It is clear from the photograph that the test has run successfully to completion. PICA's red display LEDs are flashing completion code 0x55, PICB's green display LEDs are showing the high byte of the final packet it received (0xFFFFFF), and there are no yellow error lights.

Realized communication speed with a 100-foot ethernet cable

This time, it took 3 hours, 52 minutes and 45 seconds for the program to run to completion. That is a total of 13,965 seconds.

During this time, a total of 24 * 33,554,432 = 805,603,368 information bits were transmitted (the same as before).

The realized speed is calculated as 805,603,368 / 13,965 = 57,666 information bits per second. Stated in the reciprocal, this is 17.34 microseconds per information bit. By way of comparison, the realized speed when short pieces of hookup wire were used was 10.59 microseconds per bit. The time needed for the test is increased by 13,965 - 8,527= 5,438 seconds, or 64%, when 100 feet of ethernet cable is used. Why?

There are two sources of delay: propagation delays and quantization in the detection loop in the receiving PIC.

Propagation delays

Light travels in a vacuum at 300,000,000 meters/second. 100 feet of vacuum is 30.48 meters long, and it takes light 30.48 / 300,000,000 = 102 nanoseconds to travel that far. Electricity travels through copper cable at about 80% of light's speed in a vacuum. Therefore, it takes about 102 / 0.8 = 127 nanoseconds for a voltage transition in one of the Clock lines to travel from one end to the other through the ethernet cable.

There are 24 information bits per packet, but six other bits are sent as well: a start and stop bit, two parity bits, the Acknowledgement bit and a post-Acknowledgement bit. At the start of each of these 30 bits, one PIC sends an edge-transition down the wire. The PIC at the other end must receive this edge, and respond with an edge-transition of its own, before the first PIC can move on to the next bit. As a result, 60 edge-transitions must travel through the cable for each packet. Since 33,554,432 packets are involved, there are a total of 60 * 33,554,432 = 2,013,265,920 propagations through the cable. 127 nanoseconds per propagation adds a total of 255.69 seconds to the test time.

The MAX232 chips have their own propagation delay. The datasheet quotes the receiver-side propagation delay as 500 nanoseconds. Based on the quoted slew rate of 3 V/us, the propagation delay on the driver-side is approximately the same. Since there is a MAX232E chip at each end of the cable, the chips add a one microsecond delay to each bit sent through. That adds a total extra delay of 1μs * 2,013,265,920 = 2,013.27 seconds to the test time.

These two propagation delays account for 2,268.96 seconds, or 41.7%, of the extra 5,438 seconds needed when the ethernet cable is used. The rest of the delay -- about 3,169 seconds -- arises from the following factor.

Quantization in the detection loop in the receiving PIC

The following snippet of Microchip Assembly code is a typical loop used to detect a high-to-low or low-to-high transition of the ClockIn line.

```
SOP9
        ; Wait until ClockIn goes high.  It will go high after the other PIC
        ; has read Data bits #1, #3 ... #23.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    SOP10               ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP9                ; No time-out, so keep waiting
        movlw   0x10                ; Error code #16
        goto    Error_Flash         ; Time-out error
SOP10
```

The ClockIn line is one of the pins of portC, which is read by the `movf` instruction. The entire contents of the port are saved in register `portCmirror`. (Note: although this save is not essential for this loop, it is essential to have the correct values available further down in the code when this PIC sets or resets its own ClockOut line.)

If the desired ClockIn transition has occurred, then the `btfsc` instruction passes control to the next step in the procedure, which starts at label `SOP10`.

If the desired ClockIn transition has not yet occurred, one cannot immediately read portC again. One must first test to see if a time-out has occurred. If a time-out has occurred, the ISR will have set the `TimerZero` bit in User-register `IntFlags`, and the `btfss` instruction will pass control to the error routine, along with the appropriate error code. Only if the time-out test succeeds will control loop back to label `SOP9` and another test of portC.

Suppose the propagation delay causes this loop to miss the edge transition that occurred when short pieces of hookup wire were used, and requires the loop to run through one more iteration. The question is: how long does one iteration of this loop take? The following schedule sets out the number of instruction cycles needed.

```
        movf    portC,w             ; 1 cycle
        movwf   portCmirror         ; 1 cycle
        btfsc   portCmirror,ClockIn ; 2 cycles
        goto    SOP10               ; not executed if missed
        btfss   IntFlags,TimerZero  ; 1 cycle
        goto    SOP9                ; 2 cycles
        movlw   0x10                ; not executed if missed
```

```
goto    Error_Flash        ; not executed if missed
```

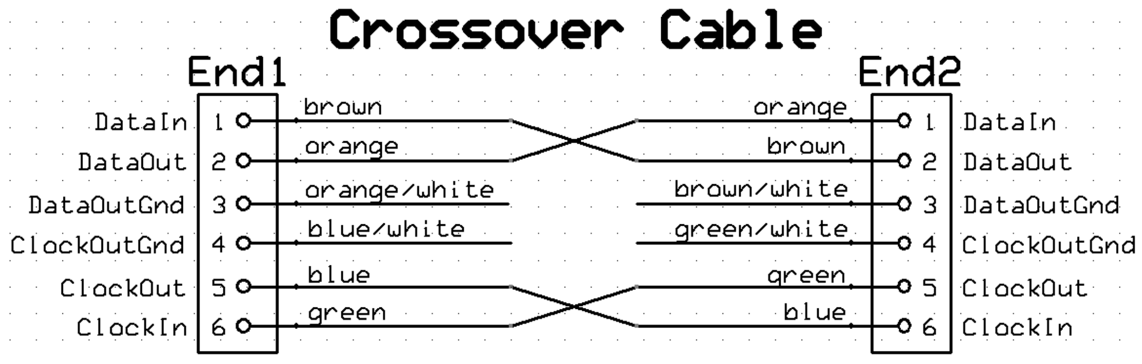
One extra iteration takes seven instruction cycles. Since the crystal frequency is 20MHz, each instruction cycle takes 200 nanoseconds, and these extra seven instructions take 1.4 microseconds.

We can calculate how many extra iterations were required during the test program's complete run. 3,169 seconds of delay divided by 1.4 microseconds per iteration is about 2,264,000,000 iterations of the loop. In other words, 2,263,600,000 detection loops were delayed by one iteration during the course of the test program. The entire test involved a total of 2,013,265,920 edge transitions. It looks like each high-to-low or low-to-high edge transition of a Clock line caused the program to run through one extra iteration of the detection loop.

Crossover cables

Each PIC's ClockOut and DataOut lines are the other PIC's ClockIn and DataIn lines. This is an asymmetry. There are two philosophies when it comes to handling this kind of asymmetry. In one philosophy, the two PICs are wired up exactly the same way (so, for example, the ClockOut line is pin RC6 on both) and the asymmetry is dealt with in the connecting cable (so, for example, the ClockOut wire at one end is treated as the ClockIn wire at the other end). In the other philosophy, the two ends of the cable are made identical, and the two PICs have different connections to their communication terminals.

I prefer the former approach. As the schematic diagram shows, the wiring for each PIC and each MAX232E chip can then be made exactly the same. This makes preparing schematic diagrams and laying out PCB boards a lot easier. But, it means that care must be taken when connecting the cables to the board-edge terminal strips. The schematic diagram shows the cable details like this:



The Cat5E cable has four twisted-wire pairs. The signal conductors have solid coloured insulation. The conductors with white insulation are tied to ground, but at the active end only. All eight conductors in the cable are used, but there are only six galvanic connections at each end.

In my application (control of post-War Lionel trains), the wires are attached to screw-terminal-blocks. I found that using RJ45 plugs and sockets was not robust enough for my purpose. I certainly do not want to be clambering around underneath the layout pulling on cables trying to find a loose plug.

If you intend to use RJ45 plugs, be advised that the colour scheme shown in my diagram of the crossover cable is not any one of the ethernet standards.

Concluding remarks

The MAX232E line drivers are single-sided. Each chip has two transmission channels and two reception channels, and only one wire is required for each. It is used assuming that all four wires have a common ground.

To get higher speeds, newer and more advanced line drivers would have to be used. They can drive the same ethernet cable and its four twisted-wire pairs. But they drive each twisted wire-pair like a miniature balanced RF transmission line. The drivers use external resistors to feed each wire-pair at its characteristic impedance, and the receivers use external resistors to match the characteristic impedance of the wire-pair. Impedance-matching ensures that as much electrical power as possible is transferred from the transmitting device to the receiving device.

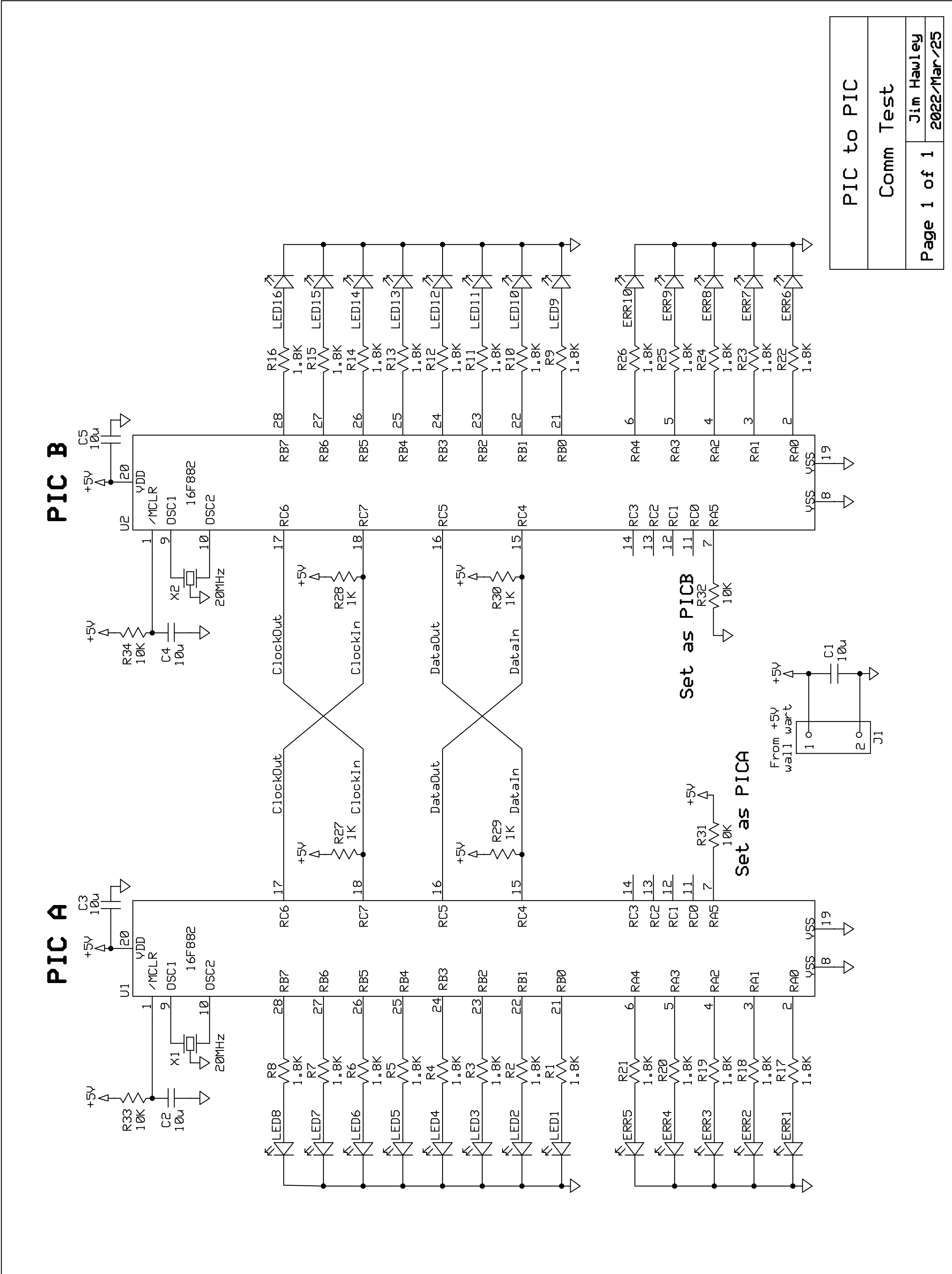
Jim Hawley

May 2022

(As always, an email describing errors or omissions would be appreciated.)

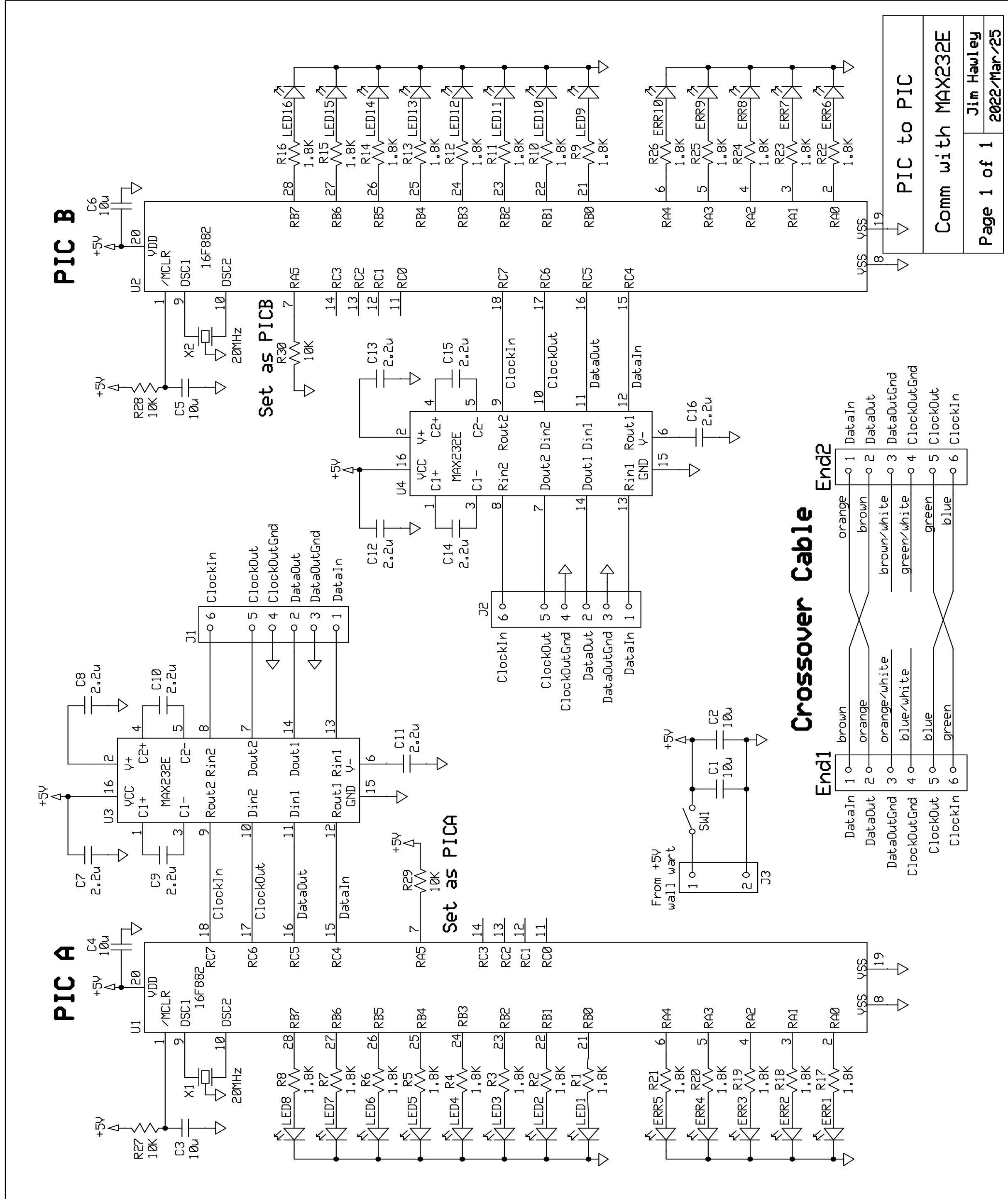
Appendix "A"

Basic 4-wire PIC-to-PIC communication



Appendix "B"

PIC-to-PIC communication using MAX232E line drivers



Appendix "C"

Microchip Assembly code for the 16F882

```
; Program for PIC-to-PIC communication test using 16F882 microprocessor
;
; ******************************************************************************
; Notes
; ******************************************************************************
;
; 1. Timer0 is the only source of interrupts.  The Timer0 module is always
;    enabled, the Timer0 count register is always being incremented and Timer0
;    interrupts occur continually.
;
; 2. When a Timer0 interrupt occurs, the only thing that the Interrupt Service
;    Routine does is to set a flag bit, IntFlags<TimerZero>.  This flag is
;    irrelevant and is ignored unless the routine being executed is looking for
;    time-outs.
;
; 3. The Timer0 module is configured to generate an interrupt approximately one
;    millisecond after being initiated.  The Timer0 configuration is set as
;    follows:-
;    (i)   The Timer0 pre-scalar is set to 32:1, so each increment in the
;          Timer0 count register takes 32 instruction cycles.
;    (ii)  With a 20MHz crystal, each instruction cycle takes 0.2us, so each
;          increment in the Timer0 count register takes 32 * 0.2us = 6.4us.
;    (iii) At the start of any procedure for which time-outs are to be detected,
;          the Timer0 count register is initialized to d'99' = 0x63.  It will
;          therefore take d'157' increments of the Timer0 count register before
;          it rolls over from d'255' to zero and triggers an interrupt.
;    (iv)  6.4us * d'157' = 1,004.8us, or 1.0048ms.  This is close enough to the
;          one millisecond interval desired.
;
; ******************************************************************************
; The routines in this program are grouped into the following blocks:
;     A. Definition of system registers
;     B. Definition of user registers
;     C. Interrupt Service Routine
;     D. Initialization of system registers
;     E. Initialization of user registers
;     F. Main programs
;     G. Hardware routines to communicate with the "other" PIC
;     H. Subroutine Error_Flash
;     I. Miscellaneous and timing subroutines
;
; Configuration Words for 16F882
;               b<13>=1     Disable in-circuit debugger
;               b<12>=0     Disable Low-Voltage Programming
;               b<11>=0     Disable fail-safe clock monitor
;               b<10>=0     Disable internal/external switchover
;               b<9-8>=00   Disable brown-out reset
;               b<7>=1      Turn OFF EEPROM memory protection
;               b<6>=1      Turn OFF program memory protection
;               b<5>=1      Set standard /MCLR operation
;               b<4>=1      Disable power-up timer
;               b<3>=0      Disable watch-dog timer
```

```
;               b<2-0>=010  Set HS oscillator gain
  #include      "p16F882.inc"
  processor     16F882
  __CONFIG      _CONFIG1,0x20F2  ; b'xx10 0000 1111 0010'
  __CONFIG      _CONFIG2,0x3FFF
;
; Crystal frequency is 20MHz, so the instruction cycle time is 200ns.
;
; ******************************************************************************
; Block A - Definition of PIC 16F882 system registers
; ******************************************************************************
;
; Registers in bank 0
TMR0            equ     0x01            ; Timer0 count register
STATUS          equ     0x03            ; Status register
carry           equ             0x00    ; carry from MSB occurred
zero            equ             0x02    ; result of operation is zero
page0           equ             0x05    ; register bank selector low bit
page1           equ             0x06    ; register bank selector high bit
portA           equ     0x05
portB           equ     0x06
portC           equ     0x07
INTCON          equ     0x0B            ; Interrupt control register
gie             equ             0x07    ; global interrupt enable
tmr0ie          equ             0x05    ; Timer0 interrupt enable
tmr0if          equ             0x02    ; Timer0 interrupt flag
T1CON           equ     0x10            ; Timer1 control register
SSPCON          equ     0x14            ; Synch serial port control reg 1
CCP1CON         equ     0x17            ; Capture/Compare/PWM control reg 1
RCSTA           equ     0x18            ; Receive status and control register
CCP2CON         equ     0x1D            ; Capture/Compare/PWM control reg 2
ADCON0          equ     0x1F            ; Analogue-to-digital control reg 0
;
; Registers in bank 1
OPTION_REG      equ     0x81            ; Option register
TRISA           equ     0x85            ; portA pin I/O direction
TRISB           equ     0x86            ; portB pin I/O direction
TRISC           equ     0x87            ; portC pin I/O direction
PCON            equ     0x8E            ; Power control register
WPUB            equ     0x95            ; portB weak pull-up resistors
IOCB            equ     0x96            ; portB interrupt-on-change
PSTRCON         equ     0x9D            ; pulse steering control register
;
; Registers in Bank2
CM1CON0         equ     0x107           ; Comparator C1 control register 0
CM2CON0         equ     0x108           ; Comparator C2 control register 0
CM2CON1         equ     0x109           ; Comparator C2 control register 1
;
; Registers in bank 3
ANSEL           equ     0x188           ; Analogue select channels 0-7
ANSELH          equ     0x189           ; Analogue select channels 8-13
;
f               equ             0x01    ; f and w identify destination register
w               equ             0x00
;
; ******************************************************************************
; Block B - Definition of user registers - Accessible only in bank 0
```

```
; ******************************************************************************
;
; I/O ports
portAmirror     equ     0x20
ERR0            equ             0x00    ; Output - Error display LED (LSB)
ERR1            equ             0x01    ; Output - Error display LED
ERR2            equ             0x02    ; Output - Error display LED
ERR3            equ             0x03    ; Output - Error display LED
ERR4            equ             0x04    ; Output - Error display LED (MSB)
PICSelect       equ             0x05    ; Input - PICA or PICB select
;
portBmirror     equ     0x21
LED0            equ             0x00    ; Output - Display LED (LSB)
LED1            equ             0x01    ; Output - Display LED
LED2            equ             0x02    ; Output - Display LED
LED3            equ             0x03    ; Output - Display LED
LED4            equ             0x04    ; Output - Display LED
LED5            equ             0x05    ; Output - Display LED
LED6            equ             0x06    ; Output - Display LED
LED7            equ             0x07    ; Output - Display LED (MSB)
;
portCmirror     equ     0x22
ncRC0           equ             0x00    ; Output - not connected
ncRC1           equ             0x01    ; Output - not connected
ncRC2           equ             0x02    ; Output - not connected
ncRC3           equ             0x03    ; Output - not connected
DataIn          equ             0x04    ; Input - DataIn
DataOut         equ             0x05    ; Output - DataOut
ClockOut        equ             0x06    ; Input - ClockOut
ClockIn         equ             0x07    ; Output - ClockIn
;
; Registers containing the integer to be transmitted
MasterCount0    equ     0x23            ; First Byte (LSB)
MasterCount1    equ     0x24            ; Second byte
MasterCount2    equ     0x25            ; Third byte (MSB)
;
; Variables used for transmitting a packet
OutPacket0      equ     0x26            ; First byte in packet (LSB)
OutPacket1      equ     0x27            ; Second byte in packet
OutPacket2      equ     0x28            ; Third byte in packet (MSB)
OutBitCount     equ     0x29            ; Counter of bits in a byte
OutParity       equ     0x2A            ; Cumulative parity
OutAckRecd      equ     0x2B            ; Acknowledgement bit received
;
; Variables used for receiving a packet
InPacket0       equ     0x2C            ; First byte in packet (LSB)
InPacket1       equ     0x2D            ; Second byte in packet
InPacket2       equ     0x2E            ; Third byte in packet (MSB)
InBitCount      equ     0x2F            ; Counter of bits in the packet
InParityCalc    equ     0x30            ; Cumulative parity (as calculated)
InParityRecd    equ     0x31            ; Parity bits #1 and 2 (as received)
;
; Flag bits to identify interrupts
IntFlags        equ     0x32            ; Flags for interrupt identification
TimerZero       equ             0x00
;
; Temporary registers used in the subroutines indicated
```

```
tempDus         equ     0x33            ; microsecond-delay subroutines
tempDms         equ     0x34            ; millisecond-delay subroutines
TOCount         equ     0x35            ; Counter for 20ms time-out detection
;
; Registers used for saving w and STATUS registers before executing ISR
w_temp          equ     0x70
status_temp     equ     0x71
;
; *******************************************************************************
; Hard start
; *******************************************************************************
;
        org     0x0000
HardStart
        bcf     INTCON,gie
        goto    InitializeSystemRegisters
;
; *******************************************************************************
; Block C - Interrupt Service Routine
; *******************************************************************************
;
        org     0x0004
ISR
        ; Disable global interrupts
        bcf     INTCON,gie
        ; Save current STATUS and w-reg.  Swaps do not affect status bits.
        movwf   w_temp
        swapf   STATUS,w
        movwf   status_temp
        ; Branch based on the Timer0 interrupt flag
        btfss   INTCON,tmr0if
        goto    ISR_Finish          ; Goto since not a Timer0 interrupt
        ; Tell the program that a Timer0 interrupt has occurred
        bsf     IntFlags,TimerZero
        bcf     INTCON,tmr0if       ; Clear the Timer0 interrupt flag
ISR_Finish
        ; End-of-interrupt
        swapf   status_temp,w       ; Retrieve the original STATUS and w-reg
        movwf   STATUS
        swapf   w_temp,f
        swapf   w_temp,w
        retfie                      ; Re-enable global interrupts and return
;
; *******************************************************************************
; Block D - Initialization of system registers
; *******************************************************************************
;
InitializeSystemRegisters
        ; Select register bank 0
        bcf     STATUS,page0
        bcf     STATUS,page1
        ; INTCON=0 disables all interrupt activity (affects portB)
        clrf    INTCON
        ; T1CON=0 disables Timer1 (affects portC)
        clrf    T1CON
        ; SSPCON<5>=0 disables synchronous serial port (affects portA and portC)
        clrf    SSPCON
```

```
; CCP1CON=0 disables Enhanced C/C/P module (affects portB and portC)
clrf    CCP1CON
; RCSTA=0 disables the serial port (affects portC)
clrf    RCSTA
; CCP2CON=0 disables C/C/P module (affects portC)
clrf    CCP2CON
; ADCON0=0 disables the A/D module (affects portA)
clrf    ADCON0
;
; Select register bank 1
bsf     STATUS,page0
bcf     STATUS,page1
; Configure OPTION_REG
; <7>=1 disable PortB pull-up resistors
; <6>=0 RB0 interrupt on falling edge
; <5>=0 internal clock (Fosc/4) drives Timer0
; <4>=0 increment Timer0 on low-to-high
; <3>=0 assign prescalar to Timer0
; <2-0>=100 set Timer0 prescalar 32:1
movlw   0x84
movwf   OPTION_REG
; Configure RA5 for input; all other pins of portA for output
movlw   0x20
movwf   TRISA
; Configure all pins of portB for output
movlw   0x00
movwf   TRISB
; Configure RC4 and RC7 for input; all other pins of portC for output
movlw   0x90
movwf   TRISC
; PCON<4-5>=0 disables wake-up and brown-out resets
bcf     PCON,5
bcf     PCON,4
; WPUB=0 disables weak pull-up resistors (affects portB)
clrf    WPUB
; IOCB=0 disables Interrupt-on-change  (affects portB)
clrf    IOCB
; PSTRCON=0 zeroes the steering pin assignments (affects portC)
clrf    PSTRCON
;
; Select register bank 2
bcf     STATUS,page0
bsf     STATUS,page1
; CM1CON0=0 disables Comparator 1 module (affects portA)
clrf    CM1CON0
; CM2CON0=0 disables Comparator 2 module (affects portA)
clrf    CM2CON0
; CM2CON1=0 disables Comparator 2 module (affects portA and portB)
clrf    CM2CON1
;
; Select register bank 3
bsf     STATUS,page0
bsf     STATUS,page1
; Ensure that all pins are digital I/O, not analogue
clrf    ANSEL                ; Set portA pins as digital I/O
clrf    ANSELH               ; Set portB pins as digital I/O
;
```

```
        ; Reselect register bank 0 before continuing
        bcf     STATUS,page0
        bcf     STATUS,page1
;
; ******************************************************************************
; Block E - Initialization of user registers
; ******************************************************************************
;
InitializeUserRegisters
        ; Initialize portA (turn off the error LEDs)
        clrf    portAmirror
        movf    portAmirror,w
        movwf   portA
        ; Initialize portB (turn off the display LEDs)
        clrf    portBmirror
        movf    portBmirror,w
        movwf   portB
        ; Initialize portC (set ClockOut and DataOut latches high)
        clrf    portCmirror
        bsf     portCmirror,ClockOut
        bsf     portCmirror,DataOut
        movf    portCmirror,w
        movwf   portC
        ; Wait 100ms for everything to stabilize
        call    del100ms
        ; Enable Timer0 interrupts
        bsf     INTCON,tmr0ie
        bsf     INTCON,gie
        ; Read RA5 to determine whether this is PICA or PICB
        movf    portA,w
        movwf   portAmirror
        btfss   portAmirror,PICSelect
        goto    MainProgram_PICB    ; RA5 low means this is PICB
        goto    MainProgram_PICA    ; RA5 high means this is PICA
;
; ******************************************************************************
; Block F - Main programs
; ******************************************************************************
;
MainProgram_PICA
        ; Initialize the value of the master count
        movlw   0x00
        movwf   MasterCount0
        movwf   MasterCount1
        movwf   MasterCount2
        ; Wait 500ms before sending the first packet, to ensure PICB is ready
        call    del500ms
MP_PICA_Loop    ; <-- This is the start of the main loop for PICA
        ; Load the master count into the output registers
        movf    MasterCount0,w
        movwf   OutPacket0
        movf    MasterCount1,w
        movwf   OutPacket1
        movf    MasterCount2,w
        movwf   OutPacket2
        ; Before sending, display the high-order byte in the packet on portB
        movf    MasterCount2,w
```

```
        movwf   portB
        ; Send the packet to PICB
        call    SendOnePacket
        ; Wait for the reply
        call    WaitForAPacketToStart
        call    ReceiveOnePacket
        ; XOR the master count (sent) with the packet received - Low-order byte
        movf    MasterCount0,w
        xorwf   InPacket0,w         ; w <-- MasterCount0 XOR InPacket0
        xorlw   0xFF                ; 2nd XOR result should be all zeroes
        btfsc   STATUS,zero         ; Z=1 if the 2nd XOR is all zeroes
        goto    MP_PICA_1           ; Goto since 2nd XOR is all zeroes
        movlw   0x18                ; Error code #24
        goto    Error_Flash         ; Round-trip error in Packet byte #0
MP_PICA_1
        ; XOR the master count (sent) with the packet received - Middle byte
        movf    MasterCount1,w
        xorwf   InPacket1,w         ; w <-- MasterCount1 XOR InPacket1
        xorlw   0xFF                ; 2nd XOR result should be all zeroes
        btfsc   STATUS,zero         ; Z=1 if the 2nd XOR is all zeroes
        goto    MP_PICA_2           ; Goto since 2nd XOR is all zeroes
        movlw   0x19                ; Error code #25
        goto    Error_Flash         ; Round-trip error in Packet byte #1
MP_PICA_2
        ; XOR the master count (sent) with the packet received - High-order byte
        movf    MasterCount2,w
        xorwf   InPacket2,w         ; w <-- MasterCount2 XOR InPacket2
        xorlw   0xFF                ; 2nd XOR result should be all zeroes
        btfsc   STATUS,zero         ; Z=1 if the 2nd XOR is all zeroes
        goto    MP_PICA_3           ; Goto since 2nd XOR is all zeroes
        movlw   0x1A                ; Error code #26
        goto    Error_Flash         ; Round-trip error in Packet byte #2
MP_PICA_3
        ; Increment the value of the master count
        ; Remember that the incf instruction does not set the Carry flag, so
        ; explicit addition must be used.
        movf    MasterCount0,w
        addlw   0x01
        movwf   MasterCount0
        ; Check for carry from low-order byte
        btfss   STATUS,carry
        goto    MP_PICA_Loop        ; No carry, so send the packet
        ; There was a carry, so increment the middle byte
        movf    MasterCount1,w
        addlw   0x01
        movwf   MasterCount1
        ; Check for carry from middle byte
        btfss   STATUS,carry
        goto    MP_PICA_Loop        ; No carry, so send the packet
        ; There was a carry, so increment the high-order byte
        movf    MasterCount2,w
        addlw   0x01
        movwf   MasterCount2
        ; Program is finished if there is a carry from the high-order byte
        btfss   STATUS,carry
        goto    MP_PICA_Loop        ; No carry, so send the packet
        ; When finished, display a flashing 0x55 on portB
```

```
MP_PICA_4
        movlw   0x55
        movwf   portB
        call    del500ms
        movlw   0x00
        movwf   portB
        call    del500ms
        goto    MP_PICA_4
;
MainProgram_PICB ; <-- This is the start of the main loop for PICB
        ; Wait for a packet to be received
        call    WaitForAPacketToStart
        call    ReceiveOnePacket
        ; Before complementing, display the low-order byte received on portB
        movf    InPacket0,w
        movwf   portB
        ; Complement the packet received; store the result in output registers
        comf    InPacket0,w
        movwf   OutPacket0
        comf    InPacket1,w
        movwf   OutPacket1
        comf    InPacket2,w
        movwf   OutPacket2
        ; Send the complemented packet to PICA
        call    SendOnePacket
        ; Start waiting for the next packet
        goto    MainProgram_PICB
;
; ******************************************************************************
; Block G - Hardware routines to communicate with the "other" PIC
; Subroutines:-
;   WaitForAPacketToStart() - Waits for a new packet to start
;   ReadOnePacket() - Receives one packet from the other PIC
;   SendOnePacket() - Sends one packet to the other PIC
; ******************************************************************************
;
WaitForAPacketToStart
; This subroutine is called when the PIC has nothing to send, but is waiting for
; the other PIC to start sending the next packet.  This subroutine does not
; detect time-outs.  If the other PIC never starts to send a packet, this
; subroutine will loop indefinitely.  A packet is ready to start when both the
; ClockIn and DataIn lines are read low.
        ; Read portC and save in register portCmirror
        movf    portC,w
        movwf   portCmirror
        ; Test if the ClockIn line is low
        btfsc   portCmirror,ClockIn
        goto    WaitForAPacketToStart   ; ClockIn is still high; keep waiting
        ; The ClockIn line is low; test if the DataIn line is also low
        btfsc   portCmirror,DataIn
        goto    WaitForAPacketToStart   ; Data line is still high; keep waiting
        return                          ; Both lines are low, so return
;
ReceiveOnePacket
; This subroutine is called when the ClockIn and DataIn lines are read low,
; which is the signal that the other PIC is ready to send a packet.  This
; subroutine contains the complete procedure to receive the packet, including
```

```
; sending the appropriate Acknowledgement bit.  The three bytes received are
; saved in user registers InPacket0, InPacket1 and InPacket2.  This subroutine
; returns only if the reception succeeded.  If reception fails, it will enter an
; infinite error loop.
        ; Clear the cumulative parity byte
        clrf    InParityCalc
        ; Initialize the bit counter to count 24 bits
        movlw   0x0C                ; 0x0C = d'12'
        movwf   InBitCount
        ; Initialize the Timer0 count register for 1.000ms time-outs
        movlw   0x63                ; 0x63 = d'99
        movwf   TMR0
        ; Clear the Timer0 interrupt flag in user register IntFlags
        bcf     IntFlags,TimerZero
        ; The Timer0 interrupt flag in system register INTCON does not need to
        ; be cleared since: (i) it is cleared at the end of each ISR cycle and
        ; (ii) register TMR0 was just reset and insufficient time has elapsed
        ; during the last two instructions for another Timer0 interrupt to
        ; occur.
        ; Pull ClockOut low to mark the end of the Start bit
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
ROB1    ; <-- This is the start of the 24-bit loop
        ; **********************************************************************
        ; This is the start of Data bits #1, #3 ... #23
        ; **********************************************************************
        ; Wait until ClockIn goes high.  It will go high when the other PIC has
        ; placed Data bits #1, #3 ... #23 on the DataIn line.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    ROB2                ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB1                ; No time-out, so keep waiting
        movlw   0x01                ; Error code #1
        goto    Error_Flash         ; Time-out error
ROB2
        ; Read Data bits #1, #3 ... #23.
        movf    portC,w
        movwf   portCmirror
        ; Set the Carry flag to the value of Data bits #1, #3 ... #23
        bsf     STATUS,carry        ; Assume the Carry flag should be high
        btfss   portCmirror,DataIn
        bcf     STATUS,carry        ; No, the Carry flag should be low
        ; Right-shift the Carry flag into InPacket2<7>
        rrf     InPacket2,f
        ; Right-shift into InPacket1
        rrf     InPacket1,f
        ; Right-shift into InPacket0
        rrf     InPacket0,f
        ; If Data bits #1, #3 ... #23 are high, increment the cumulative parity
        btfsc   InPacket2,7
        incf    InParityCalc,f
        ; Set ClockOut high to mark the end of the (odd) Data bit
        bsf     portCmirror,ClockOut
```

```
        movf    portCmirror,w
        movwf   portC
ROB3
        ; *********************************************************************
        ; This is the start of Data bits #2, #4 ... #24
        ; *********************************************************************
        ; Wait until ClockIn goes low.  It will go low when the other PIC has
        ; placed Data bits #2, #4 ... #24 on the DataIn line.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    ROB4                ; Goto since ClockIn is now low
        ; ClockIn is still high; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB3                ; No time-out, so keep waiting
        movlw   0x02                ; Error code #2
        goto    Error_Flash         ; Time-out error
ROB4
        ; Read Data bit #2, #4 ... #24.
        movf    portC,w
        movwf   portCmirror
        ; Set the Carry flag to the value of Data bits #2, #4 ... #24
        bsf     STATUS,carry        ; Assume the Carry flag should be high
        btfss   portCmirror,DataIn
        bcf     STATUS,carry        ; No, the Carry flag should be low
        ; Right-shift the Carry flag into InPacket2<7>
        rrf     InPacket2,f
        ; Right-shift into InPacket1
        rrf     InPacket1,f
        ; Right-shift into InPacket0
        rrf     InPacket0,f
        ; If Data bits #2, #4 ... #24 are high, increment the cumulative parity
        btfsc   InPacket2,7
        incf    InParityCalc,f
        ; Pull ClockOut low to mark the end of the (even) Data bit
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
        ; Decrement the bit counter, and quit after 24 bits
        decfsz  InBitCount,f
        goto    ROB1                ; Goto the next (odd) bit
ROB5
        ; *********************************************************************
        ; This is the start of Partity bit #2
        ; *********************************************************************
        ; Wait until ClockIn goes high.  It will go high when the other PIC has
        ; placed Parity bit #2 on the DataIn line.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    ROB6                ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB5                ; No time-out, so keep waiting
        movlw   0x03                ; Error code #3
        goto    Error_Flash         ; Time-out error
ROB6
```

```
        ; Read Parity bit #2.
        movf    portC,w
        movwf   portCmirror
        ; Set the Carry flag to the value of Parity bit #2
        bsf     STATUS,carry        ; Assume the Carry flag should be high
        btfss   portCmirror,DataIn
        bcf     STATUS,carry
        ; Left-shift the Carry flag into InParityRecd<0>
        rlf     InParityRecd,f
        ; Set ClockOut high to mark the end of Parity bit #2
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
ROB7
        ; ***********************************************************************
        ; This is the start of Parity bit #1
        ; ***********************************************************************
        ; Wait until ClockIn goes low.  It will go low when the other PIC has
        ; placed parity bit #1 on the DataIn line.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    ROB8                ; Goto since ClockIn is now low
        ; ClockIn is still high; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB7                ; No time-out, so keep waiting
        movlw   0x04                ; Error code #4
        goto    Error_Flash         ; Time-out error
ROB8
        ; Read Parity bit #1.
        movf    portC,w
        movwf   portCmirror
        ; Set the Carry flag to the value of Parity bit #1
        bsf     STATUS,carry        ; Assume the Carry flag should be high
        btfss   portCmirror,DataIn
        bcf     STATUS,carry        ; No, the Carry flag should be low
        ; Left-shift the Carry flag into InParityRecd<0>
        rlf     InParityRecd,f
        ; Pull ClockOut low to mark the end of Parity bit #1
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
ROB9
        ; ***********************************************************************
        ; This is the start of the Stop bit
        ; ***********************************************************************
        ; Wait until ClockIn goes high.  It will go high when the other PIC has
        ; placed the Stop bit on the DataIn line.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    ROB10               ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB9                ; No time-out, so keep waiting
        movlw   0x05                ; Error code #5
        goto    Error_Flash         ; Time-out error
```

```
ROB10
        ; Read the Stop bit.
        movf    portC,w
        movwf   portCmirror
        ; ***********************************************************************
        ; This is the end of the Stop bit
        ; ***********************************************************************
        ; Test the Stop bit and branch accordingly
        btfsc   portCmirror,DataIn
        goto    ROB11                   ; Goto since Stop bit is high
        movlw   0x06                    ; Error code #6
        goto    Error_Flash             ; Stop bit error
ROB11
        ; Test the parity and branch accordingly
        movf    InParityCalc,w
        xorwf   InParityRecd,w          ; w <-- InParityCalc XOR InParityRecd
        andlw   0x03                    ; Keep only the two low-order bits
        btfsc   STATUS,zero             ; Z=1 if the two parities are the same
        goto    ROB_AckOK               ; Goto since the parity is OK
        movlw   0x07                    ; Error code #7
        goto    Error_Flash             ; Parity error
ROB_AckOK
        ; Both parity and Stop bit are OK, so Acknowledge success
        ; Pull the DataOut line low
        bcf     portCmirror,DataOut
        movf    portCmirror,w
        movwf   portC
        ; Set ClockOut high to report that the Acknowledgement bit is ready
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
ROB12
        ; Wait until ClockIn goes low.  It will go low when the other PIC has
        ; read the Acknowledgement bit.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    ROB13                   ; Goto since ClockIn is now low
        ; ClockIn is still high; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB12                   ; No time-out, so keep waiting
        movlw   0x08                    ; Error code #8
        goto    Error_Flash             ; Time-out error
ROB13
        ; ***********************************************************************
        ; This is the end of the Acknowledgement bit
        ; ***********************************************************************
        ; Set the DataOut line high
        bsf     portCmirror,DataOut
        movf    portCmirror,w
        movwf   portC
        ; Pull ClockOut low - This is our penultimate clock transition
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
ROB14
        ; Wait until ClockIn goes high.  It will go high when the other PIC has
```

```
        ; read our penultimate clock transition.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    ROB15                   ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    ROB14                   ; No time-out, so keep waiting
        movlw   0x09                    ; Error code #9
        goto    Error_Flash             ; Time-out error
ROB15
        ; Set ClockOut high
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
        ; Return success
        retlw   0x00
;
SendOnePacket
; This subroutine is called when this PIC wants to send a packet to the other
; PIC.  This subroutine contains the complete procedure to send a packet,
; including: (i) an initial verification that the other PIC is ready to receive,
; and (ii) the initial wait for the other PIC to respond to the Ready-to-send
; signal.  The three bytes to be sent are stored in user registers OutPacket0,
; OutPacket1 and OutPacket2.  This subroutine returns only if the transmission
; succeeds.  If transmission fails, it will enter an infinite error loop.
; Caution:-
; 1. OutPacket0, OutPacket1 and OutPacket2 are changed during execution.
        ;
        ; **********************************************************************
        ; Step #1: Verify that a transmission can be started
        ; **********************************************************************
        ; Verify that the ClockIn line is high.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    SOP1                    ; Goto since ClockIn line is high
        movlw   0x0B                    ; Error code #11
        goto    Error_Flash             ; Receiver not ready on first test
SOP1
        ; Verify that the DataIn line is high
        btfsc   portCmirror,DataIn
        goto    SOP2                    ; Goto since Data line is high
        movlw   0x0C                    ; Error code #12
        goto    Error_Flash             ; Receiver not ready on first test
SOP2
        ; Pull the ClockOut line low
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
        ; Verify that the ClockIn line is still high.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    SOP3                    ; Goto since ClockIn line is still high
        movlw   0x0D                    ; Error code #13
        goto    Error_Flash             ; Receiver not ready on second test
```

```
SOP3
        ; Verify that the DataIn line is still high
        btfsc   portCmirror,DataIn
        goto    SOP4                    ; Goto since Data line is still high
        movlw   0x0E                    ; Error code #14
        goto    Error_Flash             ; Receiver not ready on second test
SOP4
        ; Pull the DataOut line low to begin the Start bit
        bcf     portCmirror,DataOut
        movf    portCmirror,w
        movwf   portC
        ;
        ; ***********************************************************************
        ; Step #2: Wait up to 20ms for the other PIC to respond
        ; ***********************************************************************
        ; This wait is implemented by allowing the Timer0 module to cycle
        ; through 20 interrupts, each being one millisecond long.
        ; Initialize user register TOCounter to count up to 20 interrupts
        movlw   0x14                    ; 0x14 = d'20'
        movwf   TOCount
        ; Clear the Timer0 interrupt flag in user register IntFlags
        bcf     IntFlags,TimerZero
SOP5
        ; Initialize the Timer0 count register for each of the 20 cycles
        movlw   0x63                    ; 0x63 = d'99'
        movwf   TMR0
        ; The Timer0 interrupt flag in system register INTCON does not need to
        ; be cleared since: (i) it is cleared at the end of each ISR cycle and
        ; (ii) register TMR0 was just reset and insufficient time has elapsed
        ; during the last two instructions for another Timer0 interrupt to
        ; occur.
SOP6
        ; Check to see if the other PIC has pulled the ClockIn line low.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    SOP7                    ; Goto since the ClockIn line is now low
        ; The ClockIn line is still high; check for 1ms time-out
        btfss   IntFlags,TimerZero
        goto    SOP6                    ; No 1ms time-out, so keep waiting
        ; 1ms time-out while waiting for the other PIC to respond
        bcf     IntFlags,TimerZero
        decfsz  TOCount,f               ; Have we gone through 20 roll-overs?
        goto    SOP5                    ; Keep waiting; start another Timer0 cycle
        ; 20ms time-out while waiting for the receiver PIC to respond
        movlw   0x0F                    ; Error code #15
        goto    Error_Flash             ; Receiver did not respond within 20ms
        ;
        ; ***********************************************************************
        ; Step #3: The other PIC is now ready to receive
        ; ***********************************************************************
SOP7
        ; Clear the cumulative parity byte
        clrf    OutParity
        ; Initialize the bit counter to count 24 bits
        movlw   0x0C                    ; 0x0C = d'12'
        movwf   OutBitCount
```

```
        ; Re-initialize the Timer0 count register (for one 1ms time-out)
        movlw   0x63                    ; 0x63 = d'99'
        movwf   TMR0
        ; Clear the Timer0 interrupt flag in user register IntFlags
        bcf     IntFlags,TimerZero
        ; The Timer0 interrupt flag in system register INTCON does not need to
        ; be cleared since: (i) it is cleared at the end of each ISR cycle and
        ; (ii) register TMR0 was just reset and insufficient time has elapsed
        ; during the last two instructions for another Timer0 interrupt to
        ; occur.
SOP8    ; <-- This is the start of the 24-bit loop
        ; **********************************************************************
        ; This is the start of Data bits #1, #3 ... #23
        ; **********************************************************************
        ; Right-shift the next Data bit (the LSB) into the Carry flag
        rrf     OutPacket2,f
        rrf     OutPacket1,f
        rrf     OutPacket0,f
        ; Set the DataOut line to match the Carry flag
        bsf     portCmirror,DataOut     ; Assume the Data bit should be high
        btfss   STATUS,carry            ; C=1 means the Data bit is high
        bcf     portCmirror,DataOut     ; Since C=0, set the Data bit low
        movf    portCmirror,w
        movwf   portC
        ; If necessary, increment the cumulative parity in register OutParity
        btfsc   STATUS,carry        ; The Carry flag still holds the Data bit
        incf    OutParity,f         ; C=1 means increment parity
        ; Set ClockOut high to signal that the (odd) Data bit is ready
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
SOP9
        ; Wait until ClockIn goes high.  It will go high after the other PIC has
        ; read Data bits #1, #3 ... #23.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    SOP10               ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP9                ; No time-out, so keep waiting
        movlw   0x10                ; Error code #16
        goto    Error_Flash         ; Time-out error
SOP10
        ; **********************************************************************
        ; This is the start of Data bits #2, #4 ... #24
        ; **********************************************************************
        ; Right-shift the next Data bit (the new LSB) into the Carry flag
        rrf     OutPacket2,f
        rrf     OutPacket1,f
        rrf     OutPacket0,f
        ; Set the DataOut line to match the Carry flag
        bsf     portCmirror,DataOut     ; Assume the Data bit should be high
        btfss   STATUS,carry            ; C=1 means the Data bit is high
        bcf     portCmirror,DataOut     ; Since C=0, set the Data bit low
        movf    portCmirror,w
        movwf   portC
```

```
        ; If necessary, increment the cumulative parity in register OutParity
        btfsc   STATUS,carry          ; The Carry flag still holds the Data bit
        incf    OutParity,f           ; C=1 means increment parity
        ; Set ClockOut low to signal that the (even) Data bit is ready
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
SOP11
        ; Wait until ClockIn goes low.  It will go low after the other PIC has
        ; read Data bits #2, #4 ... #24.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    SOP12                 ; Goto since ClockIn is now low
        ; ClockIn is still high; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP11                 ; No time-out, so keep waiting
        movlw   0x11                  ; Error code #17
        goto    Error_Flash           ; Time-out error
SOP12
        ; Decrement the bit counter, and quit after 24 bits
        decfsz  OutBitCount,f
        goto    SOP8                  ; Goto the next (even) bit
        ; **********************************************************************
        ; This is the start of Parity bit #2
        ; **********************************************************************
        ; Set the DataOut line to match Parity bit #2
        bsf     portCmirror,DataOut       ; Assume the Data bit should be high
        btfss   OutParity,1               ; OutParity<1> is Parity bit #2
        bcf     portCmirror,DataOut       ; Bad assumption; set the Data bit low
        movf    portCmirror,w
        movwf   portC
        ; Set ClockOut high to signal that Parity bit #2 is ready
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
SOP13
        ; Wait until ClockIn goes high.  It will go high when the other PIC has
        ; read Parity bit #2.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    SOP14                 ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP13                 ; No time-out, so keep waiting
        movlw   0x12                  ; Error code #18
        goto    Error_Flash           ; Time-out error
SOP14
        ; **********************************************************************
        ; This is the start of Parity bit #1
        ; **********************************************************************
        ; Set the DataOut line to match Parity bit #1
        bsf     portCmirror,DataOut       ; Assume the Data bit should be high
        btfss   OutParity,0               ; OutParity<0> is Parity bit #1
        bcf     portCmirror,DataOut       ; Bad assumption; set the Data bit low
        movf    portCmirror,w
```

```
        movwf   portC
        ; Set ClockOut low to signal that Parity bit #1 is ready
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
SOP15
        ; Wait until ClockIn goes low.  It will go low when the other PIC has
        ; read Parity bit #1.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    SOP16                   ; Goto since ClockIn is now low
        ; ClockIn is still high; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP15                   ; No time-out, so keep waiting
        movlw   0x13                    ; Error code #19
        goto    Error_Flash             ; Time-out error
SOP16
        ; ***********************************************************************
        ; This is the start of the Stop bit
        ; ***********************************************************************
        ; Set the DataOut line high
        bsf     portCmirror,DataOut     ; Assume the Data bit should be high
        movf    portCmirror,w
        movwf   portC
        ; Set ClockOut high to signal that the Stop bit is ready
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
SOP17
        ; ***********************************************************************
        ; This is the start of the Acknowledgement bit
        ; ***********************************************************************
        ; Wait until ClockIn goes high.  It will go high when the other PIC has
        ; placed the Acknowledgement bit on the Data line.
        movf    portC,w
        movwf   portCmirror
        btfsc   portCmirror,ClockIn
        goto    SOP18                   ; Goto since ClockIn is now high
        ; ClockIn is still low; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP17                   ; No time-out, so keep waiting
        movlw   0x14                    ; Error code #20
        goto    Error_Flash             ; Time-out error
SOP18
        ; Read the Acknowledgement bit.
        movf    portC,w
        movwf   portCmirror
        ; Set the Carry flag to the value of the Acknowledgement bit
        bsf     STATUS,carry
        btfss   portCmirror,DataIn
        bcf     STATUS,carry
        ; Left-shift the Carry flag into OutAckRecd<0>
        rlf     OutAckRecd,f
        ; Pull ClockOut low to signal that the Acknowledgement has been read
        bcf     portCmirror,ClockOut
        movf    portCmirror,w
```

```
        movwf   portC
SOP19
        ; Wait until ClockIn goes low.  It will go low when the other PIC has
        ; read our penultimate clock transition.
        movf    portC,w
        movwf   portCmirror
        btfss   portCmirror,ClockIn
        goto    SOP20               ; Goto since ClockIn is now low
        ; ClockIn is still high; check for time-out
        btfss   IntFlags,TimerZero
        goto    SOP19               ; No time-out, so keep waiting
        movlw   0x15                ; Error code #21
        goto    Error_Flash         ; Time-out error
SOP20
        ; Set ClockOut high
        bsf     portCmirror,ClockOut
        movf    portCmirror,w
        movwf   portC
        ; **********************************************************************
        ; Check the Acknowledgement bit
        ; **********************************************************************
        ; Test the Acknowledgement bit and branch accordingly
        btfss   OutAckRecd,0
        retlw   0x00                ; Acknowledgement is low, so return success
        movlw   0x16                ; Error code #22
        goto    Error_Flash         ; Acknowledgement bit is high
;
; ******************************************************************************
; Block H - Subroutine Error_Flash flashes a unique non-zero error code on
;           the five low-order bits of portA.  The error code is lit up for
;           one-half second, alternating with half-second blanks.  The error
;           code is passed in to this subroutine in the w-register.
;
; Error codes while receiving a packet:-
;   Code 0x01: 1.0ms TO while waiting for an odd Data bit to become ready
;   Code 0x02: 1.0ms TO while waiting for an even Data bit to become ready
;   Code 0x03: 1.0ms TO while waiting for Parity bit #2 to become ready
;   Code 0x04: 1.0ms TO while waiting for Parity bit #1 to become ready
;   Code 0x05: 1.0ms TO while waiting for the Stop bit to become ready
;   Code 0x06: The Stop bit is low
;   Code 0x07: Parity error
;   Code 0x08: 1.0ms TO while waiting for Acknowledge bit to be read
;   Code 0x09: 1.0ms TO while waiting for final Clock transition
;   Code 0x0A: Reserved for possible future use
;
; Error codes while sending a packet:-
;   Code 0x0B: Receiver is not ready - ClockIn line is low on first test
;   Code 0x0C: Receiver is not ready - Data line is low on first test
;   Code 0x0D: Receiver is not ready - ClockIn line is low on second test
;   Code 0x0E: Receiver is not ready - Data line is low on second test
;   Code 0x0F: 20ms TO while waiting for receiver to set Clock low
;   Code 0x10: 1.0ms TO while waiting for an odd Data bit to be read
;   Code 0x11: 1.0ms TO while waiting for an even Data bit to be read
;   Code 0x12: 1.0ms TO while waiting for Parity bit #2 to be read
;   Code 0x13: 1.0ms TO while waiting for Parity bit #1 to be read
;   Code 0x14: 1.0ms TO while waiting for Acknowledgement bit to become ready
;   Code 0x15: 1.0ms TO while waiting for penultimate ClockIn transition
```

```
;   Code 0x16: Acknowledgement bit is high
;   Code 0x17: Reserved for possible future use
;
; Error codes during PICA processing:-
;   Code 0x18: Round-trip error in low-order packet byte; XOR is not all ones
;   Code 0x19: Round-trip error in middle packet byte; XOR is not all ones
;   Code 0x1A: Round-trip error in high-order packet byte; XOR is not all ones
; ******************************************************************************
;
Error_Flash
        movwf   portAmirror
EF1
        movf    portAmirror,w
        movwf   portA
        call    del500ms
        movlw   0x00
        movwf   portA
        call    del500ms
        goto    EF1
;
; ******************************************************************************
; Block I - Miscellaneous and timing subroutines
; Subroutines:-
;   del5us - timed delay of exactly 5us
;   del10us - timed delay of exactly 10us
;   del15us - timed delay of exactly 15us
;   del50us - timed delay of exactly 50us
;   del100us - timed delay of exactly 100us
;   del1ms - timed delay of approximately 1ms
;   del10ms - timed delay of approximately 10ms
;   del100ms - timed delay of approximately 100ms
;   del500ms - timed delay of approximately 500ms
; ******************************************************************************
;
del5us
; This subroutine is a timed delay of exactly 5 microseconds, including the
; invoking "call".  This is equal to 25 instruction cycles at 20MHz.
        movlw   0x05                      ; 1 cycle; 0x05 = d'5'
        movwf   tempDus                   ; 1 cycle
D5us
        nop                               ; 5 cycles
        decfsz  tempDus,f                 ; 4 interim tests + 2 final = 6 cycles
        goto    D5us                      ; 4 x 2 cycles = 8 cycles
        return                            ; 2 cycles
;
del10us
; This subroutine is a timed delay of exactly 10 microseconds, including the
; invoking "call".  This is equal to 50 instruction cycles at 20MHz.
        movlw   0x0B                      ; 1 cycle; 0x0B = d'11'
        movwf   tempDus                   ; 1 cycle
D10us
        nop                               ; 11 cycles
        decfsz  tempDus,f                 ; 10 interim tests + 2 final = 12 cycles
        goto    D10us                     ; 10 x 2 cycles = 20 cycles
        nop                               ; nop adds one cycle
        return                            ; 2 cycles
;
```

```
del15us
; This subroutine is a timed delay of exactly 15 microseconds, including the
; invoking "call".  This is equal to 75 instruction cycles at 20MHz.
        movlw   0x11                    ; 1 cycle; 0x11 = d'17'
        movwf   tempDus                 ; 1 cycle
D15us
        nop                             ; 17 cycles
        decfsz  tempDus,f               ; 16 interim tests + 2 final = 18 cycles
        goto    D15us                   ; 16 x 2 cycles = 32 cycles
        nop                             ; nops add 2 cycles
        nop
        return                          ; 2 cycles
;
del50us
; This subroutine is a timed delay of exactly 50 microseconds, including the
; invoking "call".  This is equal to 250 instruction cycles at 20MHz.
        nop                             ; 1 cycle
        movlw   0x3D                    ; 1 cycle; 0x3D = d'61'
        movwf   tempDus                 ; 1 cycle
D50us
        nop                             ; 61 cycles
        decfsz  tempDus,f               ; 60 interim tests + 2 final = 62 cycles
        goto    D50us                   ; 60 x 2 cycles = 120 cycles
        return                          ; 2 cycles
;
del100us
; This subroutine is a timed delay of exactly 100 microseconds, including
; the invoking "call".  This is equal to 500 instruction cycles are 20MHz.
        nop                             ; 5 cycles for the nop's
        nop
        nop
        nop
        nop
        movlw   0x62                    ; 1 cycle; 0x62 = d'98'
        movwf   tempDus                 ; 1 cycle
D100us
        nop                             ; 98 cycles
        nop                             ; 98 cycles
        decfsz  tempDus,f               ; 97 interim tests + 2 final = 99 cycles
        goto    D100us                  ; 97 x 2 cycles = 194 cycles
        return                          ; 2 cycles
;
del1ms
; This subroutine is a timed delay of about one millisecond.  It calls
; subroutine del100us() ten times.  User register tempDms is used to count ten
; calls to subroutine del100us.
        movlw   0x0A                    ; 0x0A = d'10'
        movwf   tempDms
D1ms
        call    del100us
        decfsz  tempDms,f
        goto    D1ms
        return
;
del10ms
; This subroutine is a timed delay of about ten milliseconds.  It calls
; subroutine del100us() one hundred times.  User register tempDms is used to
```

```
; count 100 calls to subroutine del00us.
        movlw   0x64                    ; 0x64 = d'100'
        movwf   tempDms
D10ms
        call    del100us
        decfsz  tempDms,f
        goto    D10ms
        return
;
del100ms
; This subroutine is a timed delay of about 100 milliseconds.  It calls
; subroutine del10ms() ten times.
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        call    del10ms
        return
;
del500ms
; This subroutine is a timed delay of about 500 milliseconds.  It calls
; subroutine del100ms() five times.
        call    del100ms
        call    del100ms
        call    del100ms
        call    del100ms
        call    del100ms
        return
;
        END                             ; end assembly
```