

## A Serial Interface for the Schacht Comby

In order to maintain the usability of the Comby for the weaving community I have designed a retrofit Serial Interface. The interface installs inside the Comby control box by simply plugging it in. See the pictures on the next page.

The interface uses the Macomber loom protocol. To use it you must have software which has the Macomber loom driver. There are several. The interface has been tested using Fiberworks PCW but any software with Macomber loom driver would be expected to perform the same. Mr Rick Hart of Macomber Looms has kindly given his permission to use the Macomber Loom protocol.

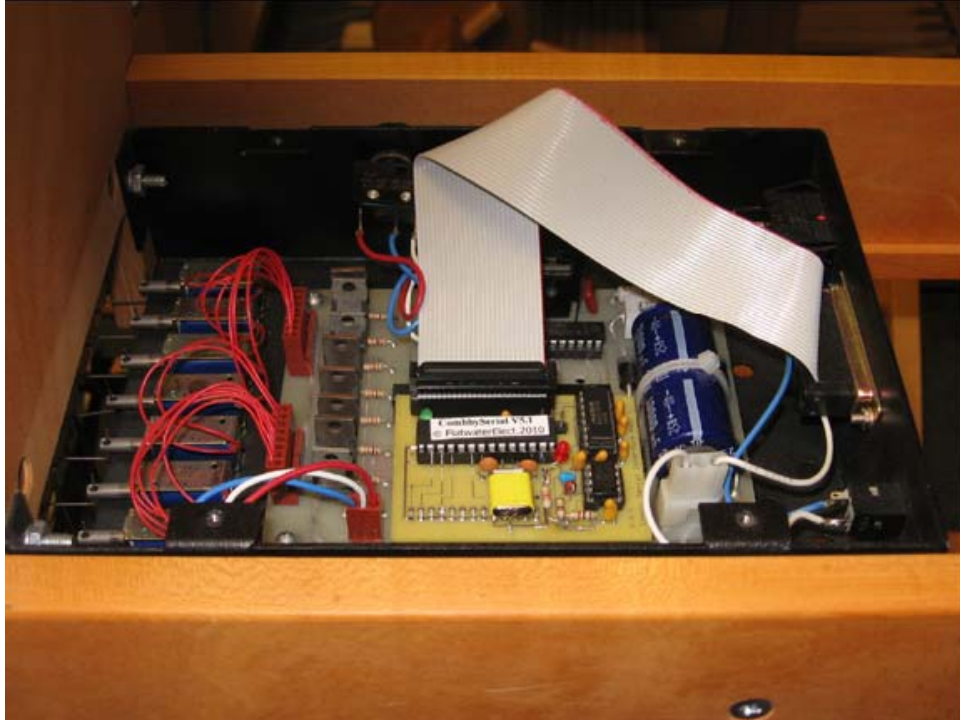
The interface is available from me, or it is possible to build one of your own using the schematics and software listing provided at the end of this document. There is also a version that uses the SLIPS protocol should your software support that instead of Macomber.

### Open Source Information

The software is published under the GPL Open Source License for all to use. Likewise, the hardware description is published under the TAPR Open Source Hardware license: <http://www.tapr.org/ohl.html>

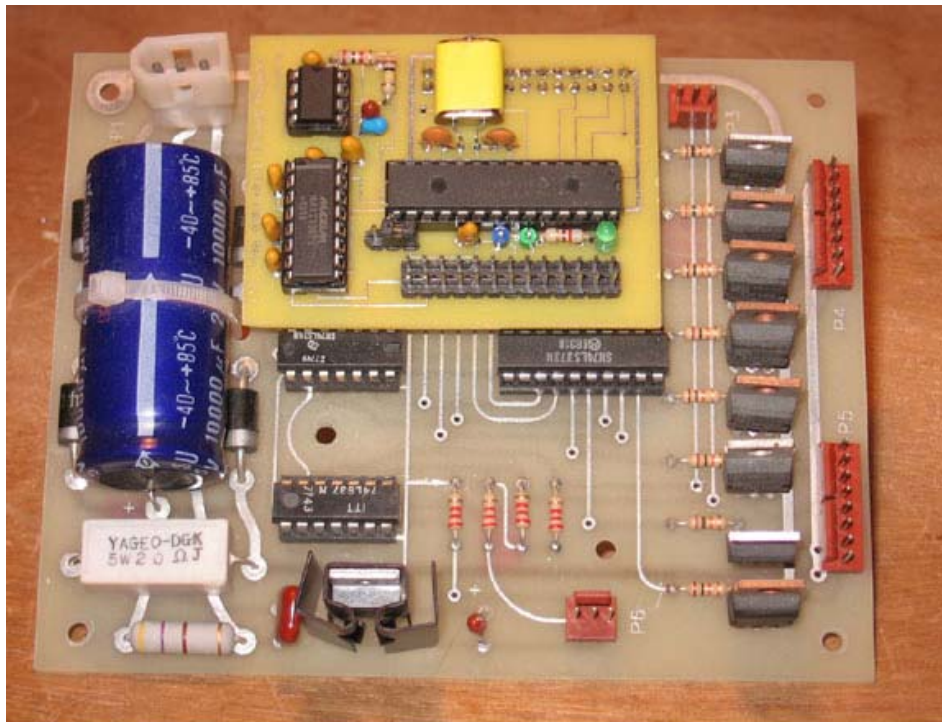
For further info:  
[www.flatwaterfarm.com](http://www.flatwaterfarm.com)  
[jcacord@gmail.com](mailto:jcacord@gmail.com)

John Acord  
Whidbey Island, WA  
12/12/2010



Interface Installed in Comby Control Box (above)

Interface Plugged Onto Comby Control Board (below)



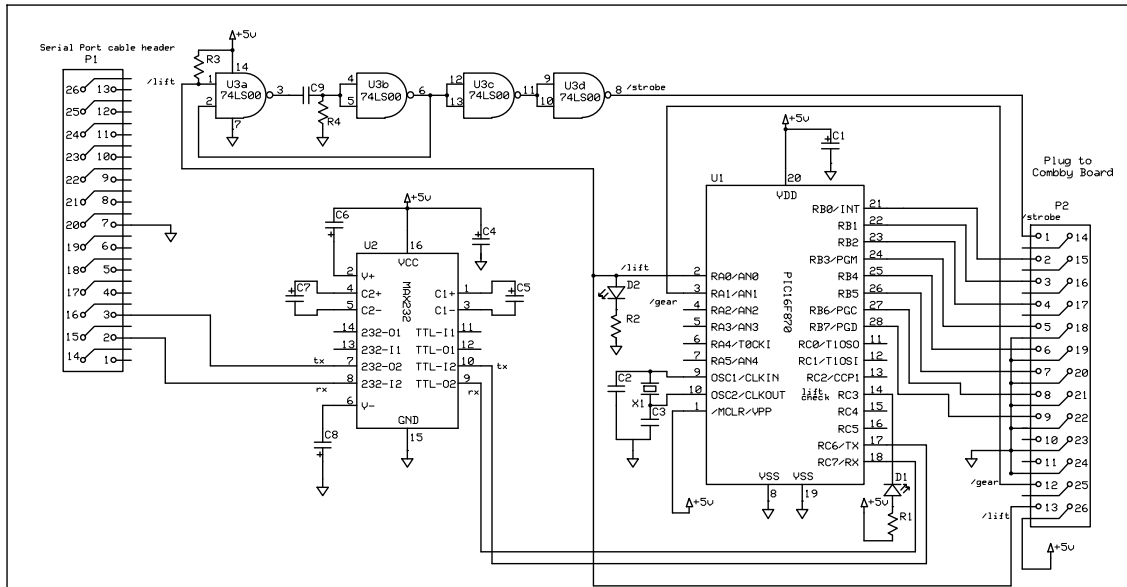
## *Technical Documentation Section*

In order to install the interface requires only a screwdriver and careful finger work. At the top of the loom is a box for the shaft control and connection to the Computer. After removing the four screws that hold the cover onto the box you will see the circuit board which controls the solenoids. There is a ribbon cable which is the connection to the Computer. Carefully, using a little end to end rocking motion on the connector, remove the ribbon cable from the Comby board. Next, while holding the Serial Interface Circuit board in one hand plug the ribbon cable into the connector on the Serial Interface board. Be careful to align the ribbon cable connector so that it is centered on the connector and makes contact with all the pins. (If not centered it won't hurt anything but you will not communicate with the computer!)

Next you will plug the interface onto the Comby circuit board where the ribbon cable was removed. See the picture on the next page. Again, center the connector so it is centered and engages all the pins. There should be an even gap between the interface board edge and the two connectors along the edge of the Comby board. Leave the cover off until you get the loom running so you can observe the green and red led's (indicator lamps). Refer to the "Diagnostics" section above for how the led's work.

You will need a "serial printer" cable between the Comby and the computer. This is a 25-pin to 9-pin cable, straight through (not a modem cable). Alternately you can use a 9-pin cable and a 9-pin to 25-pin adapter. The 25-pin end of the cable will be a 'male' connector and the 9-pin end will be a 'female' end.

# Interface Schematic Diagram



**Notes:**

- This is a plug in Serial Port to Comby Loom communications interface.
- 5V. Supply is taken from the Comby board.
- Loom control is retained on Comby board.
- Serial Port connection re-uses the Comby parallel port cable; requires a 9-pin to 25-pin serial port cable external to the enclosure.
- Manufactured under the TAPR Open Source Licence <http://www.tapr.org/OHL>

<b>Flatwater Electronics</b>		
<b>Comby Serial Interface</b>		
John Acord	Rev 4.0 12/12/2010	Page 1 of 1

## Parts List

- |             |                                      |
|-------------|--------------------------------------|
| U1          | PIC16F870                            |
| U2          | MAX232                               |
| U3          | 74LS00                               |
| P1          | 26 PIN DUAL ROW 0.1" PLUG (M)        |
| P2          | 26 PIN DUAL ROW 0.1" SOCKET (F)      |
| X1          | 3.68 HZ CRYSTAL (see pic data sheet) |
| D1, D2      | LED , 10ma NOM.                      |
| R1,R2       | 820 Ω 1/8 W                          |
| R3          | 100 KΩ 1/8W                          |
| R4          | 56 K Ω 1/8 W                         |
| C1,C4       | 1.0 µf TANTALUM                      |
| C2,C3       | 18 pf CERAMIC                        |
| C5,C6,C7,C8 | 0.1 µf TANTALUM                      |
| C9          | 0.1 µf CERAMIC                       |

## Interface Software

The interface is controlled by a program in a Microchip PIC16F870 Microcontroller. Following is the assembly language listing.

```
;Macom5.asm      12/12/2010
;Copyright 2010 John C. Acord
; This program is free software: you can redistribute it and/or modify
;it under the terms of the GNU General Public License as published by
;the Free Software Foundation, either version 3 of the License, or
;(at your option) any later version.
;
;This program is distributed in the hope that it will be useful,
;but WITHOUT ANY WARRANTY; without even the implied warranty of
;MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see
<http://www.gnu.org/licenses/>.
;jcacord@gmail.com

;Macomber protocol for Combby

    #include P16f870.inc

    __CONFIG _XT_OSC & _WDT_OFF & _CP_OFF & _LVP_OFF & _BODEN_OFF

    errorlevel -302          ;suppress "not in bank 0" message

;general storage locations

insv equ 0x20      ;temp storage for input
tip  equ 0x21      ;treadling in progres
operf equ 0x22     ;status bits for flags
insv1 equ 0x23     ;temp storage input lo
insv2 equ 0x24     ;temp storage input hi
reply equ 0x25     ;reply request bits
psav equ 0x26     ;buffer pointer save

;buffer storage area

rbuf equ 0x60      ;input buffer for shaft data (lifo)

reset org 0x00
      goto init
```

```

prog  org    0x10

init  clrf   operf
      clrf   porta
      clrf   portb
      bsf    status,rp0 ;go to bank 1
      movlw 0x06
      movwf adcon1      ;set port a as digital inputs
      movlw 0xcf
      movwf trisa       ;set port a bits are inputs
      movlw 0x00
      movwf trisb       ;set all port b bits as outputs
      movlw 0xf7
      movwf trisc       ;set portc.3 as output(led)
      movlw 0x2f
                        ;oscillator dependent value, 3.68Mhz xtal
      movwf spbrg       ;1200 baud
      bsf    txsta,txen ;enable serial port transmit, asyc mode
      bcf    status,rp0 ;back to bank 0
      bsf    rcsta,spen ;now enable the serial port(config port c bits)
      bsf    rcsta,cren ;and finally enable reception
      movlw rbuf        ;initial buffer pointer
      movwf fsr         ;set inderect addressing pointer
      bsf    portc,3    ;initally turn led off

; <esc> starts fill buffer sequence
; <cr> ends fill buffer sequence and
; sends buffer contents to PC

; in the event that the loom gets behind
; and cannot keep up with received date
; a receive buffer overflow will be set
; if ovf then reset any command in process
; and start over

; if a command sequence is started and does not end
; in a <cr> data will accumulate in the buffer
; resulting eventually in a buffer overflow
; when buffer overflow is detected the command
; sequence is reset and the buffer pointer reset
; aborting any command in progress

; begin by echoing startup message to see if PC will see us

      call  cmsg

main  btfss rcsta,oerr ;receive error?
      goto main1      ;no, continue with input check
      call  rcer       ;error processing & reset function

;no receive error, check for incoming
main1 btfsc pirl,rcif ;anything coming from pc?
      goto main2      ;yes, go read it

```

```

;nothing incoming, check for command seq
;possible to get "stuck" here so
;still need to impliment
;timeout if cmd./cr
        btfsc operf,0 ;are we in a command sequence(command
completed)?
        goto main1    ;stay around and finish before treadingling
        goto rort     ;not in comand seq, go check treadingling

main2 movf  rcreg,w    ;got something coming in, see what it is
      movwf insv      ;get the received byte
      btfsc operf,0   ;save a copy
      goto main3      ;command in progress?
      movf  insv,w    ;yes, process command string
      xorlw 0x1b      ;no, check if received byte is start command
      btfss status,z ;is <esc>?
      goto rort       ;no command yet, go check response or treadingling
      bsf   operf,0   ;set command in progress
      goto main       ;start command sequence loop
main3 call rec        ;write to buffer & check for eol
      goto main       ;stay in loop until command done

rort  btfss reply,0   ;is response scheduled?
      goto rort2      ;go check for comcheck
      btfss txsta,txif ;test for xmit shift reg empty
      goto istr       ;try again next pass
      call send       ;send the "Information Report" to PC
      goto istr

rort2 btfsc reply,3   ;is comcheck scheduled?
      call cmsg       ;send communications check
      bcf   reply,3   ;clear flag

      ;this is the check for/handling of treadingling sequence

istr  btfsc tip,0     ;was the treadle down? (tip=1 if was down)
      goto tred       ;yes, go check if back up
      btfss porta,0   ;is the treadle going down? (porta.0=0 if down)
      goto tr1        ;yes, set flag and turn on led
      goto main       ;exit
tred  btfss porta,0   ;is treadle back up? (porta.0=1 if up)
      goto main       ;no, treadle still down
      goto tr2        ;yes, go clear flag & turn off led
tr1   bsf   tip,0     ;set treadle down flag
      bcf   portc,3   ;turn on led
      goto main       ;done
tr2   clr   tip       ;treadle up, clear treadle in progress flag
      bsf   portc,3   ;treadle up, turn off led
      bsf   reply,0   ;
      bsf   reply,1   ;
      goto main       ;all done, go around again

```

```
;*****subroutines*****
```

```
;receive buffer routine. fill buffer until <cr>
```

```
;then echo received string back to PC
```

```
rec  movf  insv,w      ;get a copy of input
      movwf indf      ;write to buffer (indirect addressing)
      incf  fsr,f      ;advance pointer

                          ;check for buffer overflow
      movf  fsr,w      ;get a copy of pointer
      xorlw 0x6f       ;15-bytes max, else res pointer & try again
      btfsc status,z   ;buffer overflow?
      goto  bexit      ;yes, buffer overflow, clear & reset

                          ;ok. check for eol
      movf  insv,w      ;get input again for test
      xorlw 0x0d       ;<cr>
      btfss status,z   ;is it <cr>?
      return           ;no, go get more until <cr>

                          ;eol, so now process the buffer
      movf  fsr,w      ;save current buffer pointer
      movwf psav
      movlw rbuf
      movwf fsr        ;back to start of buffer
      movf  indf,w      ;get first byte in
      xorlw 0x23       ;<#>
      btfss status,z   ;shaft data message?
      goto  rec4        ;no, check for other commands
      movf  psav,w      ;get current buffer ptr
      movwf fsr        ;fsr -> next buffer location to read
      decf  fsr,f       ;fsr -> <cr> entry in buffer
      decf  fsr,f       ;now fsr -> shaft data

                          ;now ready to get the shaft data
                          ;from the buffer and
                          ;send it to the loom

      movf  indf,w      ;read last byte in, low byte shaft data
      movwf insv1       ;save low byte
      btfss insv1,6     ;shaft data > 9?
      goto  rec2
      movlw 0x07
      subwf insv1,f     ;convert ascii
rec2  decf  fsr,f       ;set for hi byte shaft data
      movf  indf,w      ;get hi byte
      movwf insv2       ;save hi byte
      btfss insv2,6     ;shaft data > 9?
      goto  rec3
      movlw 0x07
      subwf insv2,f     ;convert ascii

                          ;now unpack and concatenate
```

```

rec3  movlw 0x0f          ;two nibbles of shaft data
      andwf insv1,f      ;mask hi nibble
      movlw 0x0f          ;clear hi nibble & save back to insv1
      andwf insv2,f      ;mask hi nibble
      andwf insv2,f      ;clear lo nibble & save back to insv2
      swapf insv2,f      ;prepare to concatenate nibbles
      movf insv2,w       ;get it for concatenate
      iorwf insv1,w      ;nibbles => byte, result in w
      movwf portb       ;write to loom port
      goto bexit        ;done

rec4  movlw rbuf         ;back to start of buffer
      movwf fsr          ;read comand input
      movf indf,w        ;<nak>
      xorlw 0x15         ;is comand com test?
      btfss status,z     ;schedule communications check
      goto rec5          ;done
      bsf reply,3
      goto bexit

rec5  movlw rbuf         ;back to start of buffer
      movwf fsr          ;read comand input
      xorlw 0x3f         ;is it Responce command?
      btfss status,z     ;no, ignore and start over
      goto bexit
      bsf reply,0
      bsf reply,1       ;set flags for message

      ;exit routine
bexit movlw rbuf         ;initial buffer pointer
      movwf fsr          ;set inderect addressing pointer
      clrf operf         ;reset message flag
      return

      ;send message to PCW that the loom is ready for next pick

send  btfss reply,1     ;first pass?
      goto s1           ;no, second pass
      movlw 0x1b         ;<esc>
      movwf txreg        ;send to PC
      movlw 0x06         ;<ack>
      movwf txreg        ;two bytes back to back
      bcf reply,1       ;clear first flag
      return            ;done this pass

      ;second pass

      ;wait for two bytes sent
      ;then send next two

s1    bsf status,rp0    ;go to bank 1
s2    btfss txsta,trmt  ;done with two bytes?
      goto s2           ;wait for transmitter ready
      bcf status,rp0    ;back to bank 0
      movlw 0x06 0x35 ;cycle complete (correction for compatibility with Macomber specification)
      movwf txreg        ;send to PC

```

```

        movlw 0x0d          ;<cr>
        movwf txreg        ;two bytes back to back
        bcf  reply,0       ;clear event
        return

cmsg  movlw 0x1b          ;<esc>
      movwf txreg        ;send to PC
      movlw 0x15         ;<nak>
      movwf txreg        ;two bytes back to back
                          ;wait for two bytes sent
                          ;then send next
      bsf  status,rp0    ;go to bank 1
cmsg1 btfss txsta,trmt   ;done with two bytes?
      goto cmsg1         ;wait for transmitter ready
      bcf  status,rp0    ;back to bank 0
      movlw 0x0d         ;<cr>
      movwf txreg        ;finish the reply
      return

;receiver overflow handling

rcer  bcf  rcsta,cren    ;clear the error
      movf rcreg,w       ;fetch the received byte twice
      movf rcreg,w       ;to make sure all is clear
      bsf  rcsta,cren    ;and re-enable the receiver
      clrf operf        ;clear any ops and try again
      return

END

```