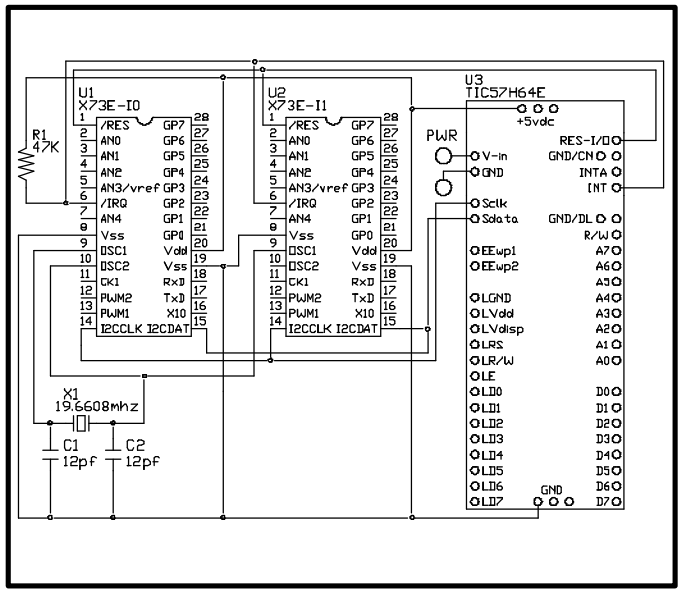
TICkit X-Tender

1.1 What is an X-tender, and what can I use it for?

The TICkit X-tender is a peripheral IC that connects to any I2C buss, typically a TICkit micro-controller. This single 28 pin IC sub-processor provides additional resources for the controller that it is connected to. 128 bytes of static RAM, 8 general purpose input output lines, a real-time-clock/calander, 5 A/D channels, a 16bit timer, 2 Capture-Compare-PWM modules, a SIN lookup table, an ATN lookup table, A buffered unipolar stepper motor driver, 4 8bit 100hz PWM generators for use as pulse generators or elementary D/A channels, buffered rs232 module, and a CSMA/CD with computed backoff network module are contained in an X-tender IC.

The TICkit controls the X-tender by using the I2C_WRITE and I2C_READ operations that are built into TICkit Interpreters (version 3.0 and higher). These commands issue 2 or 3 byte messages on the I2C clock and data lines. The same lines that are used by the program storage EEprom. Concequently, none of the TICkit I/O lines are lost to communicate with up to 8 X-tender devices.

1.2 Hardware Interconnection



Connecting to an X-tender could not be simpler. Just hook pin number 14 of the X-tender to pin number 10 of the TICkit IC and hook pin number 15 of the X-tender to pin number 9 of the TICkit IC. (The TICkit Assembly Board has a two pin socket labeled I2C which is connected to these two pins. I2C data is the lower pin and I2C clock is the upper pin of this socket pair) Apply power to the X-tender and the two devices are electrically connected. Most circuits will also connect the reset of the TICkit to the reset of the Xtender devices. (note: there has been some suggestion that the PIC16C73 fails to reset properly when using its internal voltage detection and Power-On reset circuit). The diagram illustrates the connection of two Xtender devices and a TICkit. Longer Connection schemes should terminate the I2C wires at each physical end with resistors to +5vdc. The TICkit assembly board can serve as one physical end since it already terminates the I2C buss with two 22K resistors.

The cable characteristics as well as the termination resistance on the I2C buss is very important. The buss runs at 400kbps and is very sensitive to the rising and falling edge shapes of the signals. If many devices are to be placed on the buss, or if these devices are to be very far apart, high quality twisted pair or shielded wire should be used. Termination resistance may also need to be determined experimentally. Experience suggests that 22K ohm resistance on both physical ends of the buss works the best, but this value is influenced by cabling and line loading. The 4mhz TICkit device is considerably less sensitive to buss constraints since its throughput is less than 100kbps.

1.3 Software Interconnection

8 or more X-tender devices can be connected to a TICkit in parallel. The TICkit is able to control each device independently by using the addressing characteristics of the I2C buss. Each X-tender is ordered with a unique I2C 3 bit address. This address is contained in the part number of the specific X-tender you are connecting. Obviously, each X-tender on an I2C bus must be unique to prevent messages from getting confused. Each of the 16bit commands consists of an address, a read/write flag, and an 8bit command. On the basis of this command, the X-tender will either listen for an 8bit data byte or it will prepare to send an 8bit data byte at the next inquire command.

1.4 X-tender commands

The command table sumarizes the commands that control an X-tender peripheral IC. To form the command address word, the address contained in the part number must be multiplied by 0x200, added to hexidecimal 0x8000 then added to the command value shown in the table. For example, If the part number of an Xtender IC is X73C-I1, the final suffex gives the I2C address as one. To write an ASCII letter 'A' to the RS232 output buffer, the following FBASIC line could be used:

```
i2c_write( 0x8294w, 'A' )
```

Reading from the RS232 buffer is also easy. In this case assume the part number for the IC is X73C-I7. To read a value from the buffer the line could be:

```
=( temp_variable, i2c_read( 0x8e94w ))
```

When an I2C read or write function is unable to communicate with an Xtender, the address argument will be cleared. The following example tests to ensure an Xtender IC acknowleged the command:

```
=( address_com, 0x8e94w )
i2c_write( address_com, 'A' )
IF <=( address_com, 0x00ff )
    handle_error()
ENDIF
```

1.5 Stepper Driver Module

The stepper driver is designed to control unipolar stepper motors. Four of the general purpose I/O pins (GP4 thru GP7) are configured as coil control outputs when the stepper module is enabled. GP2 and GP3 are sampled as the limit inputs if the "go until limit" mode of the stepper module is utilized. (The user must configure GP2 or GP3 as inputs when limits are used.) The stepper module is double buffered so it holds two stepper movement commands, the current command and the next command. Each command consists of an 8 bit period per step, and a signed 16 bit count of steps to execute. When the currenct count reaches zero, the next step command is shifted to the current registers, an interrupt is generated, and the step command is executed. The period register is in 1/6400 sec units. The stepper module has four modes of operation. The first is to simply step until the current count reaches zero. The other three modes will step until the current count is zero, or until one, the other, or both of the GP2 and GP3 limit inputs go to a low level. This is used to implement limits and initial index searches. Once a limit or terminal count has been reached the next step command will be loaded and executed in unconditional step mode. A sample stepper circuit using the Xtender is contained at the end of this data sheet.

1.6 Serial Communications Module

The Xtender implements an intellegent buffered serial port. This port can be configured in one of four modes of operation. Three of these modes implement different buffering strategies for asynchonous RS232 type communications. The fourth mode implements a shared wire network packet protocol. This protocol sends 8 byte packets to another Xtender device with a specified node address. The protocol uses Carrier Sense with Multiple Access combined with Collision Detection and psuedo random backoff to achieve a master-less network. This means that any node which wishes to communicate with another simply loads the destination node address, loads the 8 byte message into the transmit buffer and waits for a response from the Xtender indicating a successful or failed transmission. The Xtender automatically retries 32 times. Likewise, the Xtender takes care of receiving messages just as easily, by interrupting the controller when a message has been received. The Xtender will automatically inform other sending nodes if it is busy, and automatically records the source of the current packet received.

Buffered asynchronous serial simply divides 16 bytes of buffer space among the transmit and receive tasks. Mode 1 buffers the received data 18 deep and buffers the transmitted data 2 deep. Mode 2 buffers the transmitted data 18 deep and buffers the received data 2 deep. Mode 3 buffers the received data 10 deep and buffers the transmitted data 10 deep. The Xtender interrupts the controller whenever received data is present and whenever there is space available in the transmit buffer.

1.7 General Purpose PWM module

Four of the general purpose I/O pins (GP0 thru GP3) can be configured in the Xtender to be 8 bit 100 hz PWM outputs. This is different from the 10bit dedicated PWM module mentioned below. These outputs are controlled internally by the Xtender software to be exactly 100 hz signals with a duty cycle between 0 and 255/256. Because the rate of these signals in known (100 hz), these signals can also be used to control RC type servos as a pulse train. The on time of each signal is equal to its value multiplied by 1/25600 second. To achieve a duty cycle of 256/256 the pwm control must be turned off for that output and the pin set to a continuous high level. PWM on GP2 or GP3 will interfere with limit sensing if the stepper motor module is used in limit sensing mode.

These four PWM outputs can also be used with a capacitor to ground to acheive a rudimentary 8 bit D/A capability. Assuming the load of on the capacitor is minimum (impedance greater than 10k ohms), the output voltage swing will range linearly between 0 volt and 1275/256 (4.98) volts. Settling time is between 1/100 and 1/10 second.

1.8 A/D Module

The 5/4 channel A/D capability of the Xtender IC is native to the PIC16C73. Simply enter the A/D mode in the control register and read the result of the conversion. The conversion takes place at the time of the read. The A/D is fast enough that the I2C buss is simply halted for a fraction of a I2C byte time to get the conversion result. One of the 5 A/D inputs can be configured as a voltage reference instead of as an input channel. A reading of 255 will be equal to an input of the voltage reference. If the reference is selected as internal, a reading of 255 corresponds to Vdd of the Xtender IC.

1.9 16 bit Event Counter or Oscillator based timer

The 16bit Counter/Timer module is also native to the PIC16C73. This 16bit counter can count rising or falling edge events on pin 11 of the Xtender, or it can internally count Oscillator time cycles. One count equals 1/4915200 second. A prescaller can also be assigned to divide events or time cycles before the counter to acheive a greater count range.

Two additional pins can be configured to use the 16bit counter/timer for input capture, or for compare output. These input modules, called the CCP modules can store the count at the time of a rising or falling edge, or can output a high signal when the 16bit count exceedes a pre-set comparison value. This can be useful for generating square waves or for timing pulse events.

1.10 CCP modules for 10bit dedicated PWM

The two pins mentioned in the previous section can also be configured in the CCP modules to perform a 10 bit PWM function. The PWM function uses an internal counter to determine the period and duty cycle of the PWM signal. The period is determined by loading an 8bit value into the register designated timer 2 and the duty cycle is determined by setting the CCP register appropriately. See the PIC16C73 documentation or VersaTech BBS for more details on dedicated PWM and CCP functions

1.11 The Real Time 32 bit seconds counter

The Xtender is programmed to count the number of seconds that have elapsed since either the power was applied to the device, or since the 32 bit seconds counting register was last set. There is also a 1/100 second counter accessable to the controller. The 32 bit seconds counter is buffered, so reading the LSB of the count will capture the entire count. Conversely, when writing a count to the counter, the counter will only be updated after the MSB is written to the Xtender.

1.12 1/100, 1/10, 1, and 10 second time base

The Xtender can generate an interrupt every 1/100, 1/10, 1, or 10 seconds when told to do so. Simply setting the corresponding bit will enable the time base. When the time next time interval is reached, the cooresponding interrupt flag bit will be set. If the interrupt enable for this flag is set, the interrupt output pin (pin 6) will be pulled low. The Controller must reset the interrupt flag bit immediately to ensure the next interval will be sensed. The Controller may either respond to interrupts, or simply poll the interrupt flag register.

1.13 Trigonometric Tables

Two trigonometric lookup tables are implemented on the Xtender. A SIN lookup table for the region from 0 degrees to 89.65 degrees is provided with an output range between 0 and 256. An ATN lookup table for the ratio region of 255/256 to 0. These tables obviously require some cooersion by the controller for the specific application.

The SIN table uses an 8 bit angle value scaled between 0 degrees as 0 and 89.65 degrees as 255. The controller will need to adjust the output for the specific quadrant the SIN or COS is using on the basis of the angle involved. Furthermore, the output of the table for angles close to 90 degrees will be zero, The controller will need to understand that an output of zero in these regions is to be interpretted as 256. Most real world applications will already have quadrant and full scale logic requirements, so these tables should provide a good working basis for most trigonometic uses.

The ATN table uses an 8bit ratio value scaled between 0 for a 0 degree vale and 255/256 which cooresponds to an angular output of very near 45 degrees. The controller will need to consider the sign and magnitude of the ratio and adjust the table output for the proper octant. If a ratio is greater than 1, (256 table input), then the recipricol should be entered into the table and a value of 90 degrees less the table result will be the quadrant one angle. This ATN scheme avoids the undefined values associated with 90 degrees or the large ratios for tangents of angles between 45 degrees and 90 degrees.

1.14 Static Random Access Memory

128 bytes of SRAM are available for general volitale storage purposes. FBASIC's SEQUENCE and RECORD directives can be used to allocate and access Xtender SRAM in an organized fashion.

1.15 General Purpose Input and Output

The eight general purpose pins can be used by the stepper module, the general purpose pwm module, or be used simply as steady state outputs or unlatched inputs. The output is double buffered which assures that pins can be switched between inputs and outputs without adversely effecting other output pin levels. The outputs can be set absolutly, or two special commands can be used to set the specified pins high/low while leaving the other pins unchanged. Likewise the direction of the general purpose pins can be set to input/output for specified pins while leaving the unspecified pins' directions unchanged.

1.16 Xtender Implementation Limits

The current version of the Xtender has not been completely tested with regard to the CSMA/CD network protocol. Register assignments for the network can be assumed to be accurate in later releases.

1.17 Coding Example using a Header file

Although the I2C extender is controlled using 16bit numeric commands, it is best not to use these values in your program directly. This makes the program harder to write and to read. To simplify the control of the Xtender, a file called "xtn73e.lib" is contained on all revision 5 and later release disks that defines textual names for all of the commands.

The program to the right is an example of how to use this header file and the symbolic constants it contains to make a readable program. You will also notice the use of the '|' compiler operator which sums the adjacent constants while the program compiles. This is used extensively with Xtender commands to form a full 16bit command from the xtender device address and specific commands.

```
; program to demonstrate how xtender RAM can be used with the SEQUENCE
; directive to create a convenient pool of static symbolic storage

DEF tic57_e LIB fbasic.lib
LIB xtn73e.lib

SEQUENCE 0 xtn_dev0     ; address of the sequence beginning for XTN
device 0
SEQUENCE 0 byte var1    ; variable one
SEQUENCE 0 word var_arr[ 25 ]   ; word array of 25

FUNC none main
    LOCAL byte trash
BEGIN
    =( trash, i2c_read( xtn_dev0 | xtn_reset ))
    ; next line is used to assign a value to a RAM location in the
Xtender
    i2c_write( var1, 139b )
    con_out( i2c_read( var1 ))
    =( trash, 25b )
    REP
        --( trash )
        i2c_write( var_arr[ to_word( trash )], +( trash, 100b ))
    UNTIL ==( trash, 0b )

    =( trash, 0b )
    REP
        con_out( i2c_read( var_arr[ to_word( trash )] ))
        ++( trash )
```

Xtender command summary. (consult the xtn73_e.lib file for symbolic names)

R/W	Command	Description of Command
1	xxxx xxxx	Read data from previous command Execute folloing a repeate start bit.
x	0aaa aaaa	Read/Write RAM at address
x	1000 0000	Read/Write Port Pins Writing the Port Pin buffer directly may interfere with GP PWM or Stepper functions. Port Set high/lowfunctions are prefered.
x	1000 0001	Read/Write Port Direction Register
0	1000 0010	Write A/D control register byte format = 00cc c1rp ccc = channel number r = 1 selects RA3 as voltage reference p = 1 powers the A/D resistive ladders
1	1000 0010	Read A/D conversion result
x	1000 0011	Read/ Write Timer 1 control register byte format = 00pp 00ce pp = prescale division (1,2,4,8) c = 1 selects external clock, 0 is osc/4 e = 1 enables timer 1 counting
x	1000 0100	Read/Write Timer 2 control register btye format = 0sss sepp ssss = postscaler division value e = 1 enables timer 2 counting pp = prescaler division (1, 4, 16, 16)
x	1000 0101	Read/Write Timer 2 count register
x	1000 0110	Read/Write Timer 2 period register
x	1000 0111	Read/Write Timer 1 count (low byte) This command captures the count when reading for high byte to ensure validity. This command halts counting when writing until the high byte is written.
x	1000 1000	Read/Write Timer 1 count (high byte) Writes to this register restart counter 1.
x	1000 1001	Read/Write Capture-Compare Control 1 byte format = 00rr mmmm mmmm = mode as follows 0000 = CCP 1 off (no function) 0100 = Capture on every falling edge 0101 = Capture on every rising edge 0110 = Capture on 4th rising edge 0111 = Capture on 16th rising edge 1000 = Compare, output 1 on match 1001 = Compare, output 0 on match 1010 = Compare, interrupt on match 1011 = Compare, clear tmr1 on match 11xx = Pulse Width Modulation rr = 10 bit resolution low order bits keep as 00 for 8bit resolution
x	1000 1010	Read/Write Capture-Compare Control 2 byte format = 00rr mmmm mmmm = mode as follows 0000 = CCP 2 off (no function) 0100 = Capture on every falling edge 0101 = Capture on every rising edge 0110 = Capture on 4th rising edge 0111 = Capture on 16th rising edge 1000 = Compare, output 1 on match 1001 = Compare, output 0 on match 1010 = Compare, interrupt on match 1011 = Compare, Start A/D conv. and clear tmr1 11xx = Pulse Width Modulation rr = 10 bit resolution low order bits keep as 00 for 8 bit resolution.
x	1000 1011	Read/Write Capt-Comp Reg 1 low byte
x	1000 1100	Read/Write Capt-Comp Reg 1 high byte
x	1000 1101	Read/Write Capt-Comp Reg 2 low byte
x	1000 1110	Read/Write Capt-Comp Reg 2 high byte
x	1000 1111	Read/Write RTC 1/100 second tic count Reads from this register capture entire RTC count .

R/W	Command	Description of Command
x	1001 0000	Read/Write RTC seconds, byte 0 (LSB)
x	1001 0001	Read/Write RTC seconds, byte 1
x	1001 0010	Read/Write RTC seconds, byte 2
x	1001 0011	Read/Write RTC seconds, byte 3 (MSB) Writes to this register load the entire RTC count.
0	1001 0100	Write RS232 transmit buffer.
1	1001 0100	Read RS232 receive buffer.
x	1001 0101	Read/Write Interrupt Enable Register 1 byte format as follows: bit 0 = 1/100 second interrupt bit 1 = 1/10 second interrupt bit 2 = 1 second interrupt bit 3 = 10 second interrupt bit 4 = RS232 received character bit 5 = RS232 xmit buffer empty bit 6 = RS232 net message received bit 7 = RS232 net message xmitted
x	1001 0110	Read/Write Interrupt Enable Register 2 bit 0 = Compare interrupt for CCP1 bit 1 = Compare interrupt for CCP2 bit 2 = Current Stepper Sequence Finished bit 3 = Timer 1 overflow. bit 4 = RS232 net error (xmit retrys exceeded)
0	1001 0111	Acknowlege Interrupt 1 and clear Clear by setting the bit that cooresponds to the Enable and Flag bit. RS232 interrupts can be cleared only by reading or writing the buffer.
1	1001 0111	Read Interrupt Flag register 1 Flags int conditions. Same format as enable 1
0	1001 1000	Acknowlege Interrupt 2 and clear Clear by setting the cooresponding bit.
1	1001 1000	Read Interrupt Flag register 2 Flags int. conditions. Same format as enable 2
x	1001 1001	Read/Write RS232 transmit status byte format = mpeo cccc cccc = buffer count m = xmit buffer mode p = 1 packet being sent/data being sent e = xmit buffer is empty o = overflow (used internally)
x	1001 1010	Read/Write RS232 receive status byte format = mpfo cccc cccc = buffer count m = receive buffer mode p = packet received f = framing error has occured o = overrun error has occured
0	1001 1011	Set General Purpose pin output low Only pins with bits set high will be cleared.
1	1001 1011	Read Port output buffer (not pin levels).
0	1001 1100	Set General Purpose pin output high Only pins with bits set low will be set.
1	1001 1100	Read Port output buffer (not pin levels).
0	1001 1101	Set General Purpose pin to output
1	1001 1101	Read IC Revision Number.
0	1001 1110	Set General Purpose pin to input
1	1001 1110	Read IC Revision Number then Reset IC.
x	1001 1111	Read/Write G. P. PWM and stepper motor control. Configure Four general purpose pins to output pulses with 256 levels of duty cycle for PWM. Frequency is 100 cycles per second. Configure higher four general purpose I/O pins for unipolar stepper motor phases. byte format = hlgo pppp pppp = PWM pin control for GP 0,1,2,3 o = 1 turns step signals on for GP 4,5,6,7 g = 1 execute step command l = 1 executes steps until GP4 is low h = 1 executes steps until GP5 is low

R/W	Command	Description of Command
x	1010 0000	Read/Write General Purpose pin PWM 0 Pin is high between 0 and 255 intervals. one interval = 1/25600 second.
x	1010 0001	Read/Write General Purpose pin PWM 1
x	1010 0010	Read/Write General Purpose pin PWM 2
x	1010 0011	Read/Write General Purpose pin PWM 3
x	1010 0100	Read/Write Current Stepper Count (low)
x	1010 0101	Read/Write Current Stepper Count (high)
x	1010 0110	Read/Write Current Stepper Period
x	1010 0111	Read/Write Next Stepper Count (low)
x	1010 1000	Read/Write Next Stepper Count (high)
x	1010 1001	Read/Write Next Stepper Period
0	1010 1010	Write RS232 Baud Rate Divisor / 64
1	1010 1010	Read Baud Rate Divisor (div 64 or 16)
0	1010 1011	Read/Write RS232 Baud Rate Divisor / 16
1	1010 1011	Read Baud Rate Divisor (0=64, 255=16)
x	1010 1100	Read/Write SIN angle (0-255 = 0-89.95 degrees)
x	1010 1101	Read/Write ATN ratio (0-255 = 0 to .9995)
x	1010 1110	Read/Write Destination Address for Net Packet Write to this register only when in Net Mode. Writing to this regiser resets the xmit packet pointer to the first byte of packet. Reading this register initiates packet transmission.
x	1010 1111	Read/Write Source Address of Net Packet Write to this register only when in Net Mode. Writing to this register resets the receive packet pointer to the first byte of packet. Writing this register enables additional packet reception.

R/W	Command	Description of Command
x	1011 0000	Read/Write Network Node Address This is the Identification for this device on a Net. Values of 1 to 127 are suggested for upward compatibility with variable length packets.
x	1011 0001	Read/Write Network Status Register. Writing to this register may produce unpredictable results. Byte Format = ixds a00w i = Ignore activity in this packet x = Transmitting packet d = Data/ Header info s = Source/Dest or Checksum/Data a = Acknowlege byte w = waiting for reception of data
x	1011 0010	Not Implemented
x	1011 0011	Not Implemented
x	1011 0100	Not Implemented
x	1011 0101	Not Implemented
x	1011 0110	Not Implemented
x	1011 0111	Not Implemented
x	1011 1000	Not Implemented
x	1011 1001	Not Implemented
x	1011 1010	Not Implemented
x	1011 1011	Not Implemented
x	1011 1100	Not Implemented
x	1011 1101	Not Implemented
x	1011 1110	Not Implemented
x	1011 1111	Not Implemented

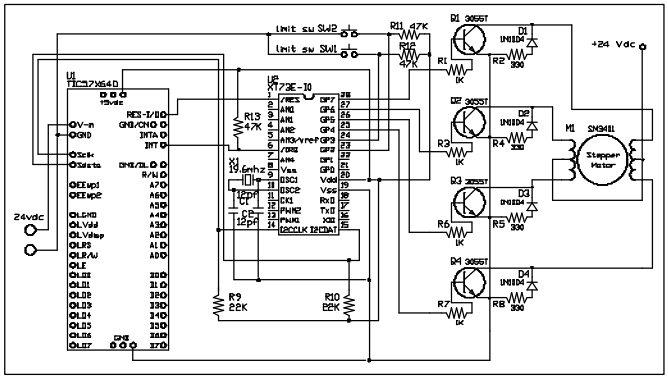
```
; sample program to demonstrate control of a
; stepper motor with an I2C xtender IC.

DEF tic57_e
LIB fbasic.lib

LIB xtn73e.lib    ; include xtender defines
LIB constrin.lib

GLOBAL long step_com
ALIAS byte step_comhi step_com 1
ALIAS byte step_comlo step_com 0

FUNC none main
BEGIN
  delay( 10 )
  con_string( "Reseting Xtender: revision " )
  con_out( i2c_read( xtn_dev0 | xtn_reset ))
  con_string( "\r\l" )
  ; initializing stepper motor includes
  ; programming sequences and stepping
  ; backwards until a limit switch closes or
  ; until max steps is reached
  ; normally a micro-switch or optical
  ; transistor would shunt GP 2 to ground
  ; when the motor moves a mechanism to a
  ; reference position
  con_string( "Initializing Stepper Motor\r\l" )
  i2c_write( xtn_dev0 | xtn_gp_cont, xtn_stepen )
  ; enable stepper outputs
  delay( 500 )
  i2c_write( xtn_dev0 | xtn_step_curlow, 0b )
  ; setup current
  i2c_write( xtn_dev0 | xtn_step_curhigh, 15b )
  ; for 15*256 steps
  i2c_write( xtn_dev0 | xtn_step_curper, 16b )
  ; 16/6400 sec per step
  i2c_write( xtn_dev0 | xtn_step_nextlow, 0b )
  ; setup next sequence as 0
  i2c_write( xtn_dev0 | xtn_step_nexthigh, 0b )
  ; steps, halts motor for
  i2c_write( xtn_dev0 | xtn_step_nextper, 0b )
  ; next command
```




```
  WHILE and( 0y11100000b,~
  ~i2c_read( xtn_dev0 | xtn_gp_cont ))
    delay( 10 )      ; TICkit could be doing other
    ; useful stuff here
  LOOP

  REP
    con_string( "Enter step command~
    ~ (-32000 to 32000): " )
    =( step_com, con_in_long( 0xfffffw ))
    con_string( " Executing\r\l" )
    i2c_write( xtn_dev0 | xtn_step_curlow,~
    ~ step_comlo )    ; setup current
    i2c_write( xtn_dev0 | xtn_step_curhigh,~
    ~ step_comhi )    ; for 15*256 steps
    i2c_write( xtn_dev0 | xtn_step_curper, 16b )
    ; 16/6400 sec per step
    i2c_write( xtn_dev0 | xtn_gp_cont,~
    ~ xtn_stepen | xtn_stepgo )
    WHILE and( 0y11100000b,~
    ~ i2c_read( xtn_dev0 | xtn_gp_cont ))
      delay( 10 )
      ; TICkit could be doing other
      ; useful stuff here
    LOOP
  UNTIL ==( step_com, 0b )

  REP
```