Application Note

AN 114

Interfacing the X9408/X9418 XDCP to 8051 Microcontrollers

by Applications Staff,

This application note describes the routines for the control of an X9408 or X9418 digitally controllable potentiometer. The X9408/X9418 devices have a variety of different instructions that provide flexibility to the designer. Additionally, the nonvolatile nature of the device allows for stored wiper positions that can be retrieved after power cycles.

The following code implements all of the available X9408/X9418 instructions using a standard bi-directional bus protocol. Although the subroutines occupy about 300 bytes of program memory, designers who won't need to implement all of the instructions can shorten the code by removing any unnecessary routines. However, this will necessitate the reassembly of the code.

For those instructions which program the nonvolatile data registers (XFR_WCR, GXFR_WCR, & WRITE_DR), acknowledge polling has been implemented to determine an early completion of the internal write cycle. Although this is automatically handled by the routines, a word or two regarding the procedure should be informative. After issuing a start condition, the master sends a slave address and receives an acknowledge. It then issues an instruction byte to the X9408/X9418 and again receives an acknowledge. If necessary, it now transmits the data byte and receives a final acknowledge. The master must then initiate a stop condition which will cause the X9408/X9418 to begin an internal write cycle. The X9408/X9418 pins go to high impedance until this internal cycle is complete. The
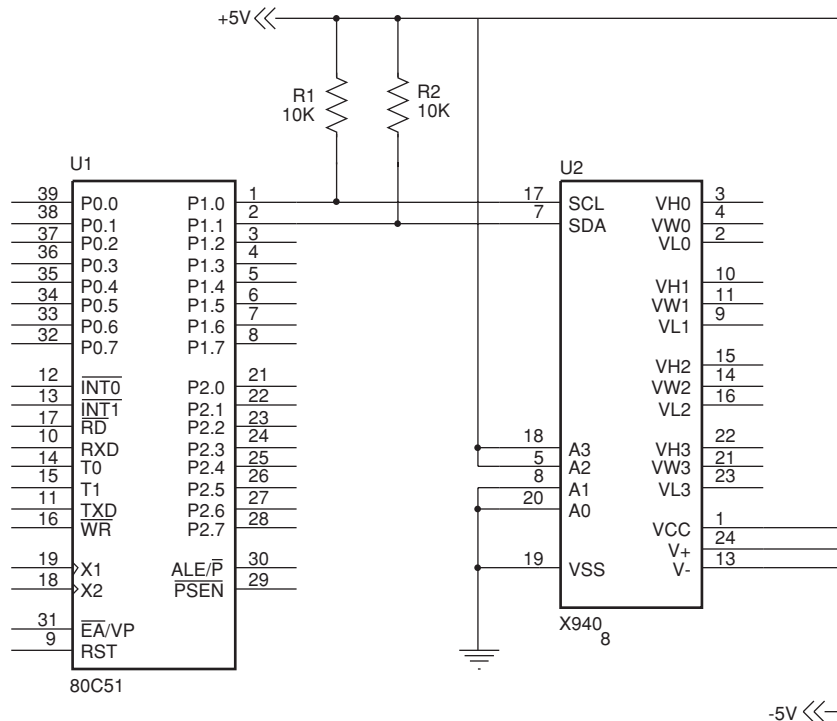

Figure 1. Connecting the X9408 to an 80C51 microcontroller

master can now begin acknowledge polling by successively sending start conditions followed by "dummy" instructions. When the X9408/X9418 finally answers with an acknowledge, the internal write cycle has been completed. The master must then initiate a stop condition. After the next start condition, the X9408/X9418 is ready to receive further instructions.

In the code listing, an assumption was made that the code executes upon a reset of the microcontroller. That is, the code is loaded into low memory, however this can be changed with an ORG assembler directive. Simple MAIN program routines are included in the code listing. These can be modified for different device addresses, different registers and different DCPs within the device.

In this listing, the commands cause an X9408/X9418 (at A3A2A1A0 = 1100 to be accessed.) The listing also includes some instructions that are specific to the Cygnal 80C51 processor. These should be examined and modified, as needed, for the specific 80C51 in the system. The commands issued in the "Main" section of the code are simple assignment and call sequences.

In Figure 1, a representative hardware connection between the X9408 and an 8051 family microcontroller is shown. The pull-up resistors on the SDA and SCL lines are determined by the total capacitance of all of the devices connected to the bus, which is about 18pF.

80C51 MICROCONTROLLER ROUTINES FOR MANIPULATING AN X9408

```
;------------------------------------------------------------------------------
;
; 80C51 MICROCONTROLLER ROUTINES FOR MANIPULATING AN X9408
;                       QUAD EEPOT
;
;                   (C) XICOR INC. 2002
;                                           CEM
;
;       FILE NAME  :  X9408_8051.TXT
;       TARGET MCU:   Cygnal C8051F000
;       DESCRIPTION:
;
; This code provides basic 80C51 code for commmunicating with and
; controlling the X9408 quad digital potentiometer. In this listing
; is code that implements all of the available X9408 instructions.
; The X9408 communicates via a 2-wire bus that is similar, but a little
; different from the I2C bus. This code is very generic and can be
; simplified and shortened by removing any unnecessary routines.
;
; For those instructions which program the nonvolatile data registers
; (XFR_WCR, GFXR_WCR, and WRITE_DR) this program provides acknowledge
; polling to determine early completion of the internal write cycle.
; Although this is handled automatically by the routines, some background
; might be helpful.
;
; After issuing a start condition, the master sends a slave address
; and receives and acknowledge (ACK).  The master then sends an instruction
; byte to the X9408 and again receives an ACK. If necessary, the master sends
; a data byte and receives a final ACK. The master then initiates a stop
; condition to signal the X9408 to begin an internal nonvolatile write
; cycle. When the write cycle begins, the I/O pins go to a high impedance state
; and remain in this state until the nonvolatile write is complete.
;
; Immediately following the stop condition, the master can begin acknowledge
; polling by successively sending start conditions, followed by "dummy"
; instructions. When the X9408 finally answers with an acknowledge, the
; internal write cycle is completed. The master then issues a stop
; condition. After the next start condition, the X9408 is ready to receive
; further instructions.
;
```

```
; This code give the flexibility to communicate with up to 16 different X9408
; devices on the same bus. It does this by using a register, named "ADDR_BYTE".
; This register is loaded with the specific slave address and address of the
; desired X9408 device. The register can be saved if there is only one X9408
; on the bus, by making ADDR_BYTE a constant.
;
; An 80C51 register is used to identify the particular X9408 register or DCP, or both,
; are used for a particular operation. There are various constants available for
; easy selection of the WCR and DR combination. The contents of the register
; is appended to the specific instruction in the "instr_gen' routine.
;
; A register is used as a counter for keeping track of the number of bits sent
; in each byte.

; A register is used for the increment/decrement instruction to specify up or
; down movement of the wiper. For each command, the master loads the "PULSES"
; register with a direction bit and 6 bits of count. If the MSB is a 1
; the wiper increments the specified number of tap positions. If the MSB
; is a 0 the wiper decrements the specified number of tap positions.
;
; A register is used to hold the specific command being executed. This allows
; the instruction to be built up and sent to the X9408.
;
; In the MAIN section are sample main code segments showing how to use the
; various subroutines.
;
; This code was tested on a Cygnal 80C51 microcontroller, using the Cygnal
; tools. The specific routines required to set up the Cygnal processor
; are identified and are probably not needed for other standard 8051 devices.
; Since each 8051 may have specific requirements that are not handled in this
; code, the programmer is advised to check the setup needs of the specific
; 80C51 derivation that is being used.
;
;------------------------------------------------------------------------------
; I/O Definition
;------------------------------------------------------------------------------

SCL            bit    p1.0   ; 80C51 pin used AS SCL
SDA            bit    p1.1   ; 80C51 pin used AS SDA

;------------------------------------------------------------------------------
; Register Definition
;------------------------------------------------------------------------------

$include (c8051f000.inc); Include regsiter definition file (Cygnal).

TEMP           equ    r1     ; Scratch register
COUNT          equ    r2     ; Loop counting register
PULSES         equ    r3     ; Bits -> DIR 0 ##### (#=pulses = 0 to 64)
COMMAND        equ    r4     ; Instruction (I.E. 0,4,8,12,16,...)
ID             equ    r5     ; Bits -> 0 0 0 0 R1 R0 P1 P0
ADDR_BYTE      equ    r6     ; Bits -> 0 1 0 1 A3 A2 A1 A0
DATA_BYTE      equ    r7     ; Bits -> CM DW D5 D4 D3 D2 D1 D0

;------------------------------------------------------------------------------
; Constant Definition
;------------------------------------------------------------------------------

SLAVE_ADR0     equ    050h
SLAVE_ADR1     equ    051h
SLAVE_ADR2     equ    052h
SLAVE_ADR3     equ    053h
SLAVE_ADR4     equ    054h
SLAVE_ADR5     equ    055h
SLAVE_ADR6     equ    056h
```

```
SLAVE_ADR7      equ     057h
SLAVE_ADR8      equ     058h
SLAVE_ADR9      equ     059h
SLAVE_ADR10     equ     05Ah
SLAVE_ADR11     equ     05Bh
SLAVE_ADR12     equ     05Ch
SLAVE_ADR13     equ     05Dh
SLAVE_ADR14     equ     05Eh
SLAVE_ADR15     equ     05Fh
;
WCR_0           equ     00h
WCR_1           equ     01h
WCR_2           equ     02h
WCR_3           equ     03h
;
DR_0            equ     00h
DR_1            equ     04h
DR_2            equ     08h
DR_3            equ     0Ch
;
DCP0_R0         equ     00h
DCP0_R1         equ     04h
DCP0_R2         equ     08h
DCP0_R3         equ     0Ch
;
DCP1_R0         equ     01h
DCP1_R1         equ     05h
DCP1_R2         equ     09h
DCP1_R3         equ     0Dh
;
DCP2_R0         equ     02h
DCP2_R1         equ     06h
DCP2_R2         equ     0Ah
DCP2_R3         equ     0Eh
;
DCP3_R0         equ     03h
DCP3_R1         equ     07h
DCP3_R2         equ     0Bh
DCP3_R3         equ     0Fh
;

READWCR         equ     0
WRITEWCR        equ     4
READDR          equ     8
WRITEDR         equ     12
XFRDR           equ     16
XFRWCR          equ     20
GXFRDR          equ     24
GXFRWCR         equ     28
INCDECWIPER     equ     32


;-----------------------------------------------------------------------------
; INTERNAL RAM
;-----------------------------------------------------------------------------

STACK_TOP       equ     060H    ; Stack top


;-----------------------------------------------------------------------------
; RESET and INTERRUPT VECTORS
;-----------------------------------------------------------------------------

        cseg AT 0
        ljmp main                       ; Locate a jump to the start of code at
```

```
;------------------------------------------------------------------------
; CODE SEGMENT
;------------------------------------------------------------------------

Code_Seg  segment  CODE

        rseg    Code_Seg        Switch to this code segment.
        using                   Specify register bank for the following
                                program code.

;------------------------------------------------------------------------
;
; NAME: execute
; FUNCTION: Determines which X9408 instruction is issued,
;           then executes
; INPUTS: COMMAND
; OUTPUTS: none
; CALLS: read_wcr, read_dr, write_wcr, write_dr, xfr_dr,
;           xfr_wcr, gxfr_dr, gxfr_wcr, inc_wiper
; AFFECTED: DPTR, A
;
;------------------------------------------------------------------------

execute:
        mov     dptr,#first  ; Get Base Address
        mov     a,COMMAND    ; Jump Offset
        jmp     @a+dptr      ; Jump to instruction handler

first:
        call    read_wcr     ; COMMAND #0
        ret
        call    write_wcr    ; COMMAND #4
        ret
        call    read_dr      ; COMMAND #8
        ret
        call    write_dr     ; COMMAND #12
        ret
        call    xfr_dr       ; COMMAND #16
        ret
        call    xfr_wcr      ; COMMAND #20
        ret
        call    gxfr_dr      ; COMMAND #24
        ret
        call    gxfr_wcr     ; COMMAND #28
        ret
        call    inc_wiper    ; COMMAND #32
        ret

;------------------------------------------------------------------------
;
; The following routines handle each X9408 instruction.
; These are called by the "execute" routine.
;
; read_wcrReads a WCR and returns its value in DATA_BYTE
; write_wcrWrites the value in DATA_BYTE to a WCR
; read_drReads a Data Register and returns its value in DATA_BYTE
; write_drWrites the value in DATA_BYTE to a data register
; xfr_drTransfers the value in a data register to its WCR
; xfr_wcrTransfers the value in a WCR to one of its data registers
; gxfr_drGlobal transfer of data registers to WCRs
; gxfr_wcrGlobal transfer of WCRs to Data Registers
; inc_wiperSingle Step Increment/Decrement of wiper position for WCR
;
```

```
; FUNCTION: Appends bits R1, R0, P1, P0 to the appropriate
;       Instruction code and passes the instruction byte to the
;       Instruction Generator.
; INPUTS: ID
; OUTPUTS: NONE
; CALLS: instr_gen
; AFFECTED: ID,A,DPTR
;
;------------------------------------------------------------------------
read_wcr:
        mov     a,ID            ; Get bits x x P1 P0
        orl     a,#090h         ; Append to read WCR instruction code
        mov     ID,a            ; Save the result
        mov     dptr,#case1     ; Jump to the base addr for this instruciton
        call    instr_gen
        ret

write_wcr:
        mov     a,ID            ; Get bits x x P1 P0
        orl     a,#0A0h      ; Append to Write WCR instruction code
        mov     ID,a            ; Save the result
        mov     dptr,#case2     ; Jump to the base addr for this instruction
        call    instr_gen
        ret

read_dr:
        mov     a,ID            ; Get bits R1 R0 P1 P0
        orl     a,#0B0h         ; Append to Read DR instruction code
        mov     ID,a            ; Save the result
        mov     dptr,#case1     ; Jump to the base addr for this instruction
        call    instr_gen
        ret

write_dr:
        mov     a,ID            ; Get bits R1 R0 P1 P0
        orl     a,#0C0h         ; Append to Write DR instruction code
        mov     ID, a           ; Save the result
        mov     dptr,#case3     ; Jump to the base addr for this instruction
        call    instr_gen
        ret

xfr_dr:
        mov     a,ID            ; Get bits R1 R0 P1 P0
        orl     a,#0D0h         ; Append to the XFR DR instruction code
        mov     ID, a           ; Save the result
        mov     dptr,#case4     ; Jump to the addr for this instruction
        call    instr_gen
        ret

xfr_wcr:
        mov     a,ID         ; Get bits R1 R0 P1 P0
        orl     a,#0E0h      ; Append to the XFR WCR instruction code
        mov     ID, a        ; Save the result
        mov     dptr,#case5 ; Jump to the addr for this instruction
        call    instr_gen
        ret

gxfr_dr:
        mov     a,ID            ; Get bits R1 R0 x x
        orl     a,#010h         ; Append to the GXFR DR instruction code
        mov     ID, a           ; Save the result
        mov     dptr,#case4     ; Jump to the addr for this instruction
        call    instr_gen
        ret
```

```
gxfr_wcr:
        mov    a,ID         ; Get bits R1 R0 x x
        orl    a,#080h      ; Append to the GXFR WCR instruction code
        mov    ID, a        ; Save the result
        mov    dptr,#case5  ; Jump to the addr for this instruction
        call   instr_gen
        ret

inc_wiper:
        mov    a,ID         ; Get bits x x P1 P0
        orl    a,#020h      ; Append to the Incr Wiper instruction code
        mov    ID,a         ; Save the result
        mov    dptr,#case6  ; Jump to the addr for this instruction
        call   instr_gen
        ret

;------------------------------------------------------------------------
;
; NAME: instr_gen (Instruction generator)
; FUNCTION: Issues appropriate I2C protocol for each X9408 instruction
; INPUTS: ADDR_BYTE, ID, PULSES, DPTR, DATA_BYTE
; OUTPUTS: DATA_BYTE
; CALLS: start_cond, stop_cond, send_byte, send_bit, get_byte, polling
; AFFECTED: DATA_BYTE, A, COUNT
;
;------------------------------------------------------------------------

instr_gen:
        call   start_cond   ; Issue an I2C start condition
        mov    a,ADDR_BYTE   ; Send X9408 slave/address byte
        call   send_byte
        jc     stop_gen     ; if NACK, end...
        mov    a,ID         ; Send X9408 instruction byte
        call   send_byte
        jc     stop_gen     ; if NACK, end...
        clr    a            ; Reset offset before jump
        jmp    @a +dptr     ; Jump to various instruction cases

case6:
        mov    a,PULSES      ; A <- Bits DIR X D5 D4 D3 D2 D1 D0
        anl    a,#00111111b ; A <- Bits 0 0 D5 D4 D3 D2 D1 D0
        mov    COUNT, a     ; Save as the number of pulses
        mov    a,PULSES
        anl    a,#10000000b ; A <- Bits DIR 0 0 0 0 0 0 0

wiper_lp:
        call   send_bit      ; Send the bit (a single pulse)
        djnz   COUNT,wiper_lp ; Continue until all pulses are sent

case4:
        jmp    stop_gen      ; If program gets here, then it is done

case2:
        mov    a,DATA_BYTE  ; Send X9408 data byte
        call   send_byte
        jmp    stop_gen

case1:
        call   get_byte      ; Receive X9408 Data Byte
        jmp    stop_gen

case3:
        mov    a,DATA_BYTE  ; Send X9408 Data Byte
        call   send_byte
```

```
        call    stop_cond       ; Issue a stop condition
        call    polling         ; Begin Acknowledge Polling
        jmp     stop_gen

case5:
        call    stop_cond       ; Issue a stop condition
        call    polling         ; Begin Acknowledge Polling

stop_gen:
        call    stop_cond       ; I2C Transmission Over!
        ret

;-----------------------------------------------------------------------------
;
; NAME: send_byte
; FUNCTION: Sends 8 bits (from MSB to LSB) to SDA and reads 1 bit from SDA
; INPUTS: A
; OUTPUTS: NONE
; CALLS: send_bit, get_bit
; AFFECTED: COUNT, TEMP, A
;
;-----------------------------------------------------------------------------

send_byte:
        mov     COUNT,#8        ; Set loop for 8 repetitions
        mov     TEMP,a          ; store as shifted byte (no shift)

bit_loop:
        mov     a,TEMP          ; Retrieve last saved shifted byte
        anl     a,#10000000b    ; Mask for MSB (Most Significant Bit)
        call    send_bit        ; Place this bit on SDA

next_bit:
        mov     a,TEMP          ; Retrieve last saved shifted byte
        rl      a               ; Rotate all bits 1 position left
        mov     TEMP,a          ; Store this updated shifted byte
        djnz    COUNT,bit_loop
        setb    SDA             ; let SDA go high after 8th bit
        call    clock           ; When all 8 bits done, read SDA line
                                ; (ACKnowledge pulse)
        ret

;-----------------------------------------------------------------------------
;
; NAME: send_bit
; FUNCTION: Places a bit on SDA and initiates a clock pulse on SCL
; INPUTS: A
; OUTPUTS: NONE
; CALLS: clock
; AFFECTED: SDA
;
;-----------------------------------------------------------------------------

send_bit:
        clr     SDA             ; Pull SDA Low
        jz      sent_zero       ; Should SDA really be LOW?
        setb    SDA             ; If Not, pull SDA HIGH

sent_zero:
        call    clock           ; Initiate a clock pulse
        ret

;-----------------------------------------------------------------------------
;
```

```
; NAME: clock
; FUNCTION: Issues a LOW-HIGH-LOW clock pulse of sufficient duration
;       & reads SDA during the high phase, just in case its needed
; INPUTS: NONE
; OUTPUTS: C
; CALLS: NONE
; AFFECTED: SCL, C
;
;-------------------------------------------------------------------------------
clock:
        nop                     ; Let SDA Set-up
        setb    SCL             ; Pull SCL HIGH and hold
        nop
        nop
        nop
        mov     c,SDA           ; Move SDA bit into carry flag
        clr     SCL             ; Pull SCL LOW
        ret

;-------------------------------------------------------------------------------
;
; NAME: get_byte
; FUNCTION: Receives 8 bits from SDA (MSB to LSB) and sends 1 bit to SDA
; INPUTS: NONE
; OUTPUTS: DATA_BYTE
; CALLS: clock, send_bit
; AFFECTED: COUNT, SDA, A, DATA_BYTE
;
;-------------------------------------------------------------------------------

get_byte:
        setb    SDA             ; Receiver shouldn't drive SDA low
        mov     COUNT,#8        ; Set Loop count to 8 repetitions

get_loop:
        call    clock           ; Clock in the current bit
        rlc     a               ; Reconstruct byte using left shifts
        djnz    COUNT,get_loop
        mov     DATA_BYTE,a     ; Store retrieved Byte for user
        clr     a               ; A <- LOW (Sending a 0)
        call    send_bit        ; Send an acknowledge
        ret

;-------------------------------------------------------------------------------
;
; NAME: start_cond (Start Condition)
; FUNCTION: Issues an I2C bus start condition
; INPUTS: NONE
; OUTPUTS: NONE
; CALLS: NONE
; AFFECTED: SDA, SCL
;
;-------------------------------------------------------------------------------

start_cond:
        setb    SDA             ; Pull SDA HIGH and allow set-up
        setb    SCL             ; Pull SCL HIGH and hold
        nop
        nop
        nop
        nop
        clr     SDA             ;Pull SDA LOW (SCL=HIGH) and hold
        nop
        nop
        nop
        nop
```

```
        clr     SCL             ;Complete clock pulse
        ret


;------------------------------------------------------------------------
;
; NAME: stop_cond (Stop condition)
; FUNCTION: Issues an I2C bus stop condition
; INPUTS: NONE
; OUTPUTS: NONE
; CALLS: NONE
; AFFECTED: SDA, SCL
;
;------------------------------------------------------------------------

stop_cond:
        clr     SDA             ; Pull SDA LOW and hold
        setb    SCL             ; Pull SCL HIGH and hold
        nop
        nop
        nop
        nop
        setb    SDA             ; Pull SDA HIGH (SCL=HIGH)
        ret


;------------------------------------------------------------------------
;
; NAME: ack_send (Send Acknowledge)
; FUNCTION: Sends an acknowledge bit to complete SDA line data reads
; INPUTS: NONE
; OUTPUTS: NONE
; CALLS: send_bit
; AFFECTED: A
;
;------------------------------------------------------------------------

ack_send:
        clr     a               ; A <- LOW (Sending a 0)
        call    SEND_BIT        ; Send the bit!
        ret


;------------------------------------------------------------------------
;
; NAME: polling (Acknowledge polling for XFR_WCR, WRITE_DR, GXFR_WCR)
; FUNCTION: Sends dummy commands to X9408 during an internal write cycle
;       so that the end of the cycle is marked by an acknowledge
; INPUTS: ADDR_BYTE
; OUTPUTS: NONE
; CALLS: start_cond, send_byte
; AFFECTED: C
;
;------------------------------------------------------------------------

polling:
        call    START_COND      ; Re-establish I2C protocol
        mov     a,ADDR_BYTE     ; Attempt to send a dummy command

again:
        call    SEND_BYTE
        jc      POLLING         ; If C=1, then there was no ACK

        ret
```

```
;--------------------------------------------------------------------------
;
; PUT MAIN PROGRAM HERE...
;
; Below are sample main programs calling the various command routines
;
;--------------------------------------------------------------------------

main:

        mov     SP, #STACK_TOP; Initialize stack pointer


;--------------------------------------------------------------------------
;
; The following section is required for the Cygnal processor. This could
; change for different versions of the 80C51.
;
; Disable the WDT. (IRQs not enabled at this point.)
; If interrupts were enabled, they would need to be explicitly disabled
; so that the 2nd move to WDTCN occurs no more than four clock
; cycles after the first move to WDTCN.

        clr     EA              ; Disable interupts

        mov     WDTCN, #0DEh; Cygnal processor specific
        mov     WDTCN, #0ADh; Cygnal processor specific

; Enable the Port I/O Crossbar

        mov     XBR2, #40h     ; Cygnal processor specific (enable weak pull ups)

        mov     PRT1CF, #00h ; Cygnal processor specific
                                ; Set no ports as push-pull (this processor
                                ; operates from 3.3V, but the X9408 operates from
                                ; 5V, so the 8051 outputs must be pulled up to 5V
                                ; with external resistors.)

;--------------------------------------------------------------------------
;
; The following are sample code segments for use in the main program...
; The potentiometer was A0-A3 pins were set to address 0Ch.
;
;--------------------------------------------------------------------------

write_2_wcr:
        mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
        mov     ID, #WCR_2     ; Specify WCR for DCP#2
        mov     COMMAND, #WRITEWCR; Write to WCR
        mov     DATA_BYTE, #43; Set wiper position to tap 43
        call    execute

read_from_wcr:
        mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
        mov     ID, #WCR_2     ; Specify WCR for DCP#2
        mov     COMMAND, #READWCR; Read WCR
        call    execute        ; WCR value is in DATA_BYTE

write_2_dr:
        mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
        mov     ID, #DCP2_R1; Specify DR#1 for DCP#2
        mov     COMMAND, #WRITEDR; Write to DR
        mov     DATA_BYTE, #21; Set data value to 21
        call    execute
```

```
read_from_dr:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #DCP2_R1; Specify DR#1 for DCP#2
      mov     COMMAND, #READDR; Read DR
      call    execute          ; DR value is in DATA_BYTE

mov_dr_2_wcr:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #DCP2_R1; Specify DR#1 to WCR of DCP#2
      mov     COMMAND, #XFRDR; Transfer DR to WCR
      call    execute

mov_wcr_2_dr:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #DCP2_R1; Specify WCR to DR#1 of DCP#2
      mov     COMMAND, #XFRWCR; Transfer WCRto DR
      call    execute

global_dr_2_wcr:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #DR_1      ; Specify DR#1 to WCR
      mov     COMMAND, #GXFRDR; Transfer DR to WCR
      call    execute

global_wcr_2_dr:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #DR_1      ; Specify WCR to DR#1 of DCP#2
      mov     COMMAND, #GXFRWCR; Transfer WCRto DR
      call    execute

decr_wiper:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #WCR_2     ; Select DCP#2
      mov     PULSES, #0Fh; Decrement DCP#2 for 16 pulses
      mov     COMMAND, #INCDECWIPER; INC wiper
      call    execute

incr_wiper:
      mov     ADDR_BYTE, #SLAVE_ADR12; Load Slave address byte
      mov     ID, #WCR_2     ; Select DCP#2
      mov     PULSES, #8Fh; Increment DCP#2 for 16 pulses
      mov     COMMAND, #INCDECWIPER; DEC wiper
      call    execute

      END
```