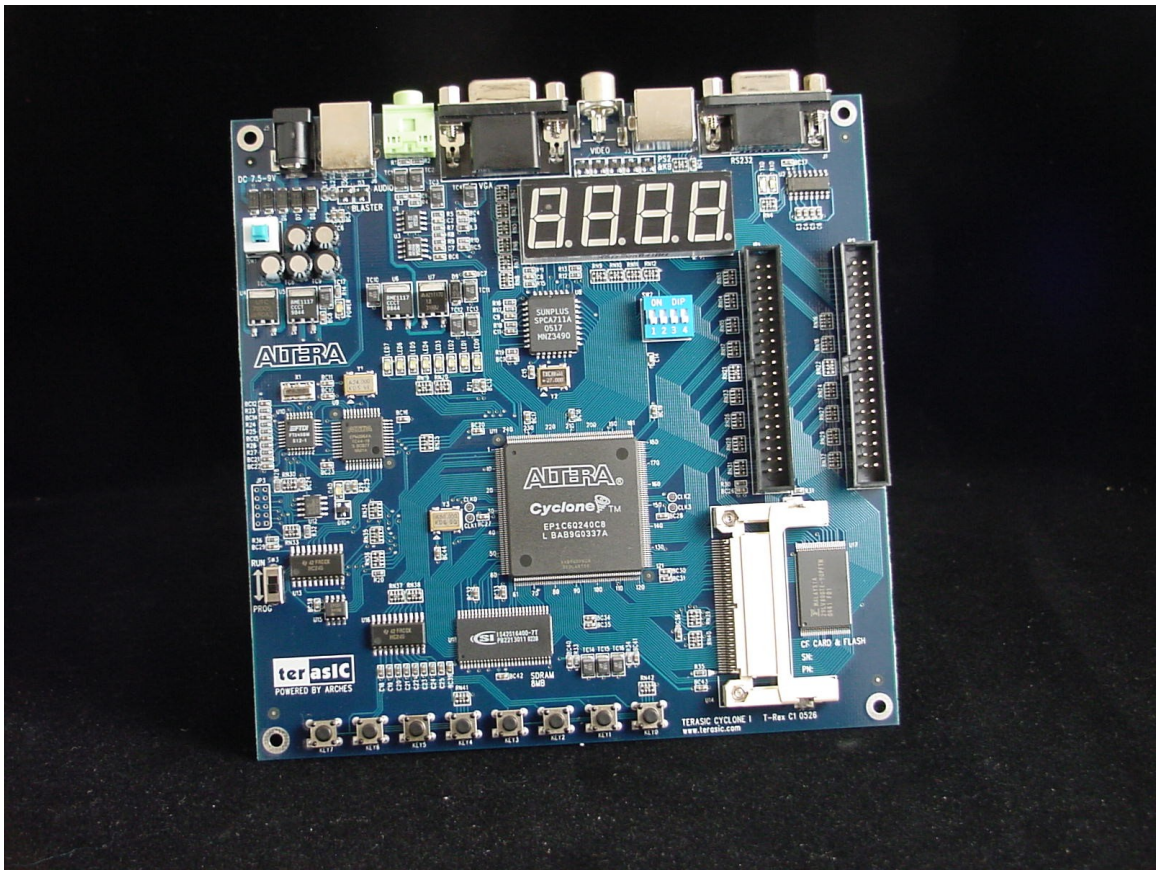
MultimediO User's Guide

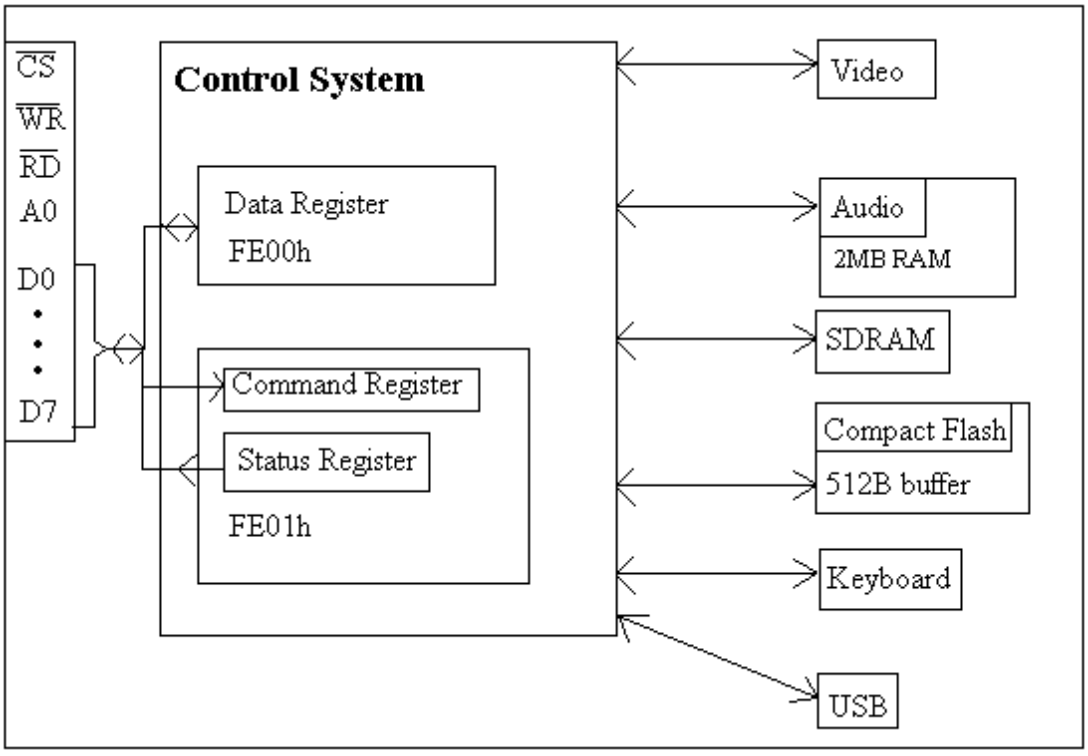


This guide will instruct you in the use of MultimediO, your gateway to all things peripheral. The FPGA boards provided to you come pre-burned with the appropriate programming file, and are ready for immediate use. If you would like to learn more about programming the FPGA, please consult the programming guide, located on the 6.115 website.

Device Summary

MultimediO is an FPGA based device designed to interface with the 8051 family of microcontrollers and provide access to an assortment of peripherals, including a video system capable of resolutions up to 800x600, and a 16-bit CD quality audio system. Figure 1, shown below, is a functional block diagram of the device. Table 1, also located below, provides a description of all pins of the device.

Figure 1. MultimediO Block Diagram.



Pin Description

Symbol	Type	Name/Description
D0-D7	I/O	Bi-directional, tri-state data bus lines. Used for all communication between the microcontroller and MultimediO.
A0	I	Address line. Used to select between the data register and command register.
/CS	I	Chip Select, active low. The device will only respond to read and write signals when chip select is asserted, otherwise those signals will be ignored.
/WR	I	Write, active low. This signal is asserted by the microcontroller to write data.
/RD	I	Read, active low. This signal is asserted by the microcontroller to read data.

Table 1. Pin Description.

Functional Description

MultimediO consists of a collection of independent subsystems, all of which can be operated simultaneously. Table 2, shown below, provides a summary of the functionality of these subsystems. State information for each subsystem can be determined by reading from the status register, see interface description for details.

Subsystem	Description	Relevant Commands
Video	MultimediO features a sprite based, 8-bit RGB, multiple resolution video system. In order to display graphics, you must create a sprite, which is a persistent, 2-dimensional image. Sprites can either be created by directly providing pixel data, or through the use of pixel data stored on a Compact Flash card. MultimediO uses an 8-bit true-color scheme, whereby each byte of pixel data directly represents the intensities of the red, green, and blue components of a pixel (as opposed to a color-palette based system, in which a byte of pixel data specifies an index in a color table). The system can operate in resolutions of 640x480 and 800x600.	Make Sprite, Move Sprite, Edit Sprite, Make Sprite From CF, Set VGA Parameters
Audio	MultimediO possesses a 16-bit, 44.1 Khz stereo audio system. It is designed to play audio files in the .WAV format, or raw audio data. Audio Data is stored in a 2MB RAM. This RAM is organized as 2^{20} 16 bit words.	Load Audio Data, Load Audio Data from CF, Play Audio
Keyboard	This subsystem facilitates the use of a standard PS/2 keyboard. Each key press is recorded, and converted to the standard ASCII representation for the corresponding character. These ASCII values are then stored in a buffer which can be accessed using the relevant command.	Read Keyboard
Compact Flash	MultimediO can be used to read data from a Compact Flash card. This data can either be loaded to a buffer, from which individual bytes can be read, or can be directly sent to the video or audio subsystems. IMPORTANT: After issuing ANY command which uses the Compact Flash card, the status byte should be read until the Compact Flash Busy Bit is 0. No other commands should be issued to the device when this bit is 1.	Load Compact Flash Buffer, Read Compact Flash Buffer, Make Sprite from CF, Load Audio Data from CF
USB	The USB subsystem is compliant to version 2.0 of the USB spec and is capable of interfacing with low speed, full speed, and high speed devices. Bus powered devices cannot be use.	USB Data Transfer

Table 2. Functional Description.

Interface Description

Communication between the microcontroller and MultimediO is accomplished through the use of two registers, a bidirectional data register and a multiplexed command/status register, located in MultimediO. In this document, it is assumed that the data register is located at FE00h and the command/status register is located at FE01h in the XIO select space. If you have chosen to locate MultimediO at a different offset, adjust addresses accordingly. A write to FE01h accesses the command register, which is used to issue commands to device; a read from FE01h accesses the status register, which is used to determine the current status of MultimediO.

In order to issue a command to the device, such as Make Sprite or Read Keyboard, the microcontroller must first write the corresponding command byte to the command register. A full list of commands, and their respective command bytes, is found below in the section titled Instruction Format. Next, the microcontroller should write each byte of data required by the instruction, such as the pixel information used by the Make Sprite command, to the data register. Then, if the instruction causes MultimediO to produce one or more byte of data for the microcontroller, such as the Read Keyboard instruction, the microcontroller should read the data register to retrieve these bytes. Lastly, the microcontroller should write a NOP command to the command register. The device is then ready to receive its next command from the microcontroller.

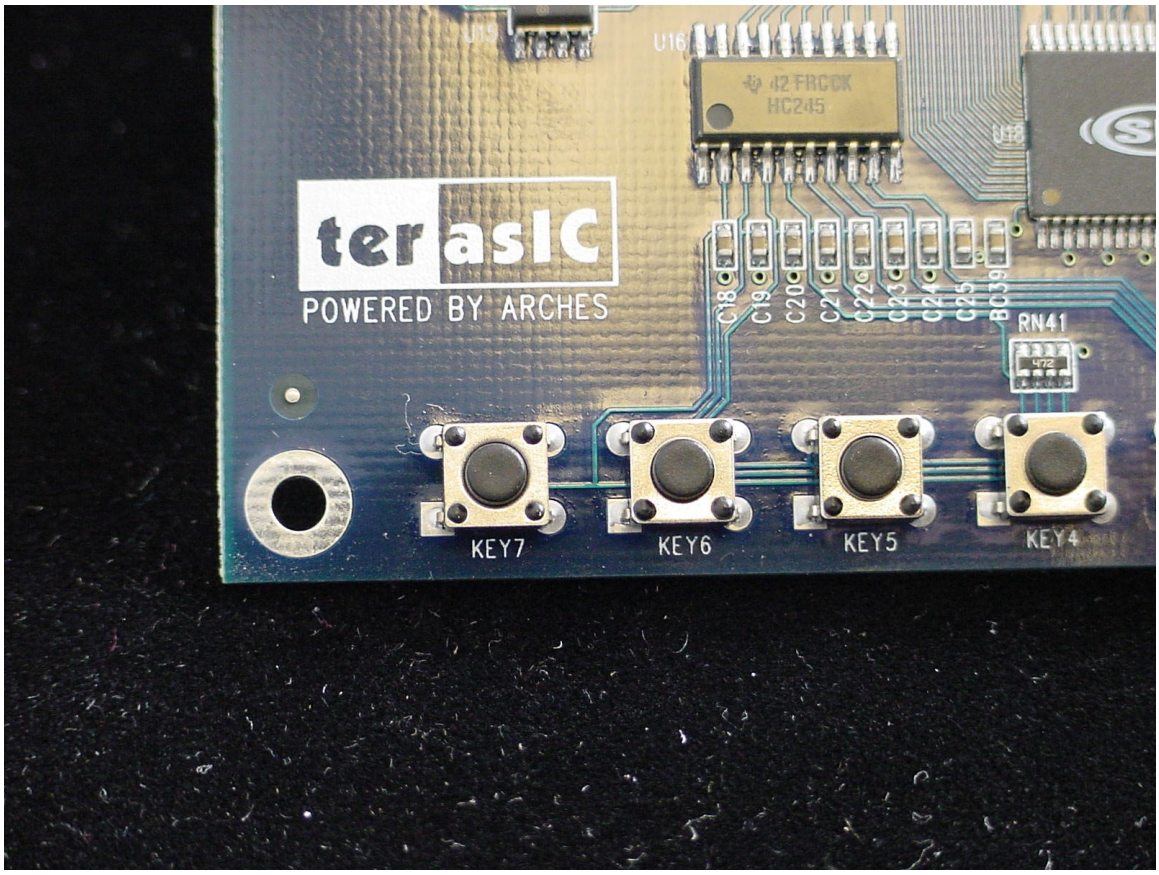
The status of MultimediO can be determined by reading a status byte, which is done by reading from the command/status register. Each bit of the status byte is a flag that represents certain state information. The bits of the status byte are defined as:

Bit	7	6	5	4	3	2	1	0
Function	RFU	RFU	USB Data Pending	USB Buffer Full	USB Busy	Keyboard Data Available	CF Busy	Ready

Bit 0: Ready- At startup, this bit is 0. This bit is set to 1 when the device has finished its power up sequence. Once set, it is only cleared if the device is reset, at which point the device will again enter its power up sequence, and again set the bit when the sequence finishes.
Bit 1: Compact Flash Busy- This bit is 1 when the Compact Flash card is busy, 0 otherwise. **IMPORTANT**: NO COMMAND, INCLUDING NOP, SHOULD BE ISSUED TO THE DEVICE WHEN THIS BIT IS 1.
Bit 2: Keyboard Data Available- This bit is 1 when there is keyboard data in the buffer waiting to be read, and 0 if the buffer is empty
Bit 3: USB Busy- This bit is 1 when the USB subsystem is busy, 0 otherwise. No USB commands should be issued when this bit is 1.
Bit 4: USB Buffer Full- This bit is 1 when the USB write buffer is full, 0 otherwise. No data should be written to the buffer when this bit is 1.
Bit 5: USB Data Pending- This bit is 1 when there is USB data in the write buffer that has yet to be sent to the device.

All other bits are unspecified/reserved for future use (RFU).

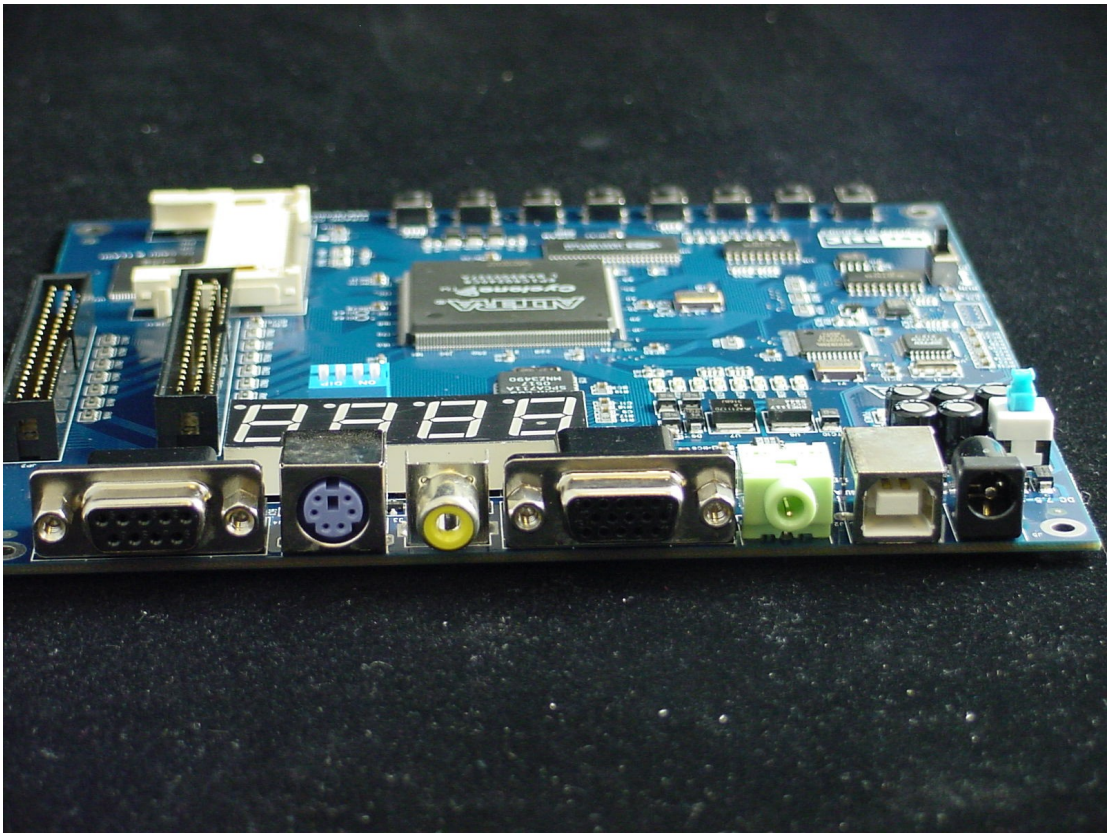
MultimediO's reset button is located in the lower left corner of the board, near the TerasIC logo; it is labeled Key7. This section of the board is pictured below. Pressing this button for 1 second resets the device. IMPORTANT: Do NOT reset the device while switch 1 on the blue and white bank of dip switches is in the on (up) position. Move this switch to the off (down) position before resetting the device.



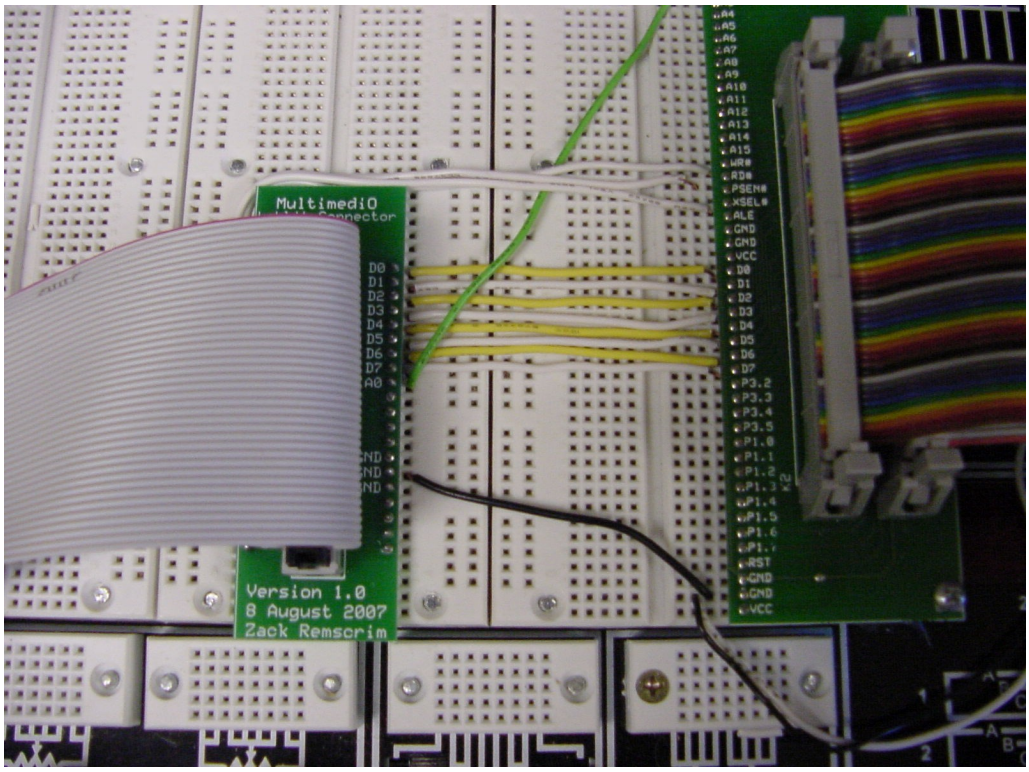
Device Connection and Powerup Sequence

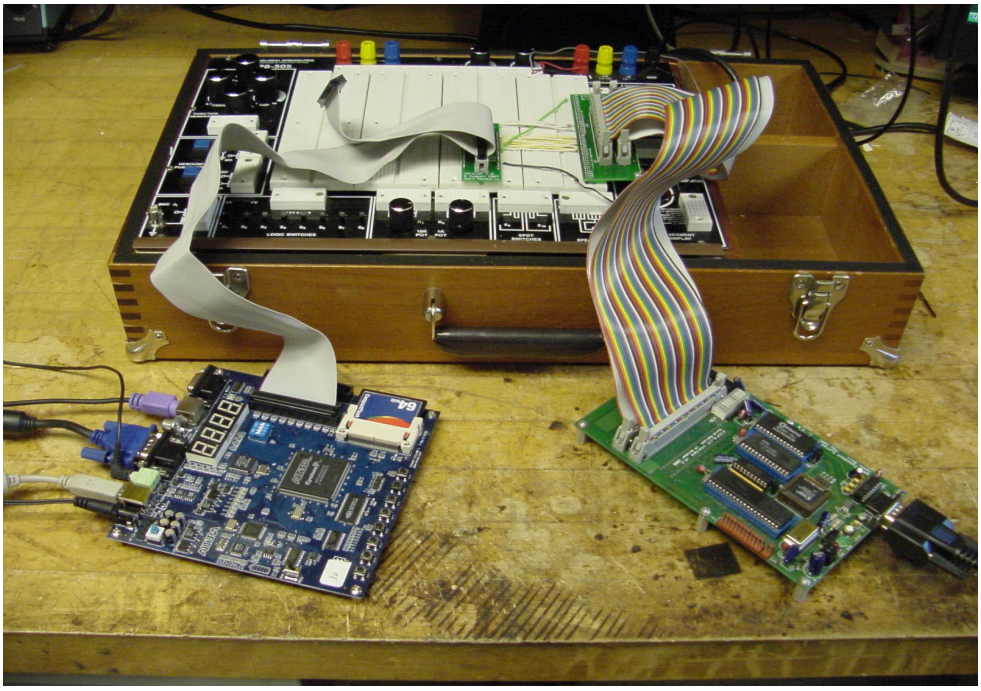
In order to use MultimediO, the following sequence should be followed. Do not turn on the device until instructed to do so.

- Connect all external devices, such as a monitor or keyboard, to MultimediO. Also, connect the AC power adapter.

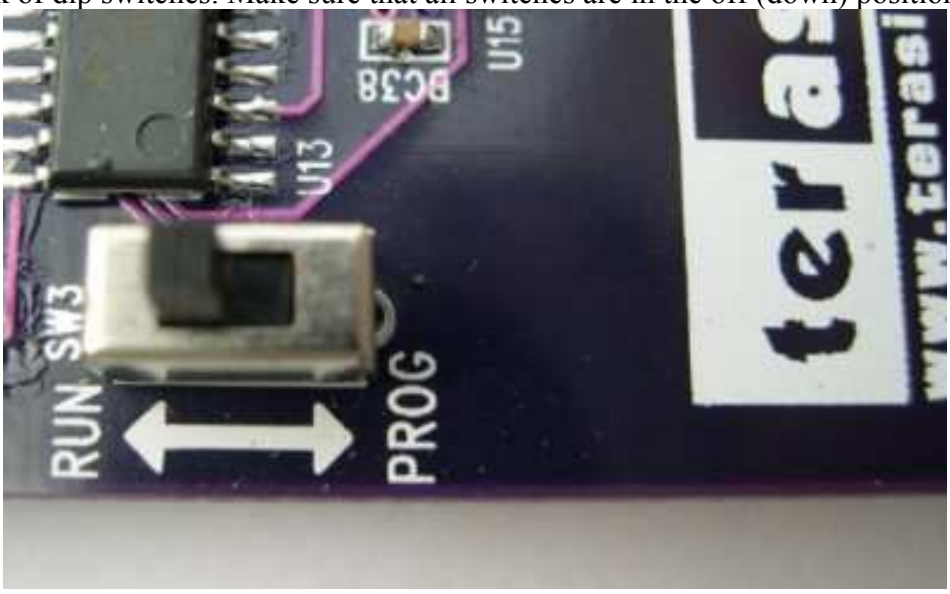


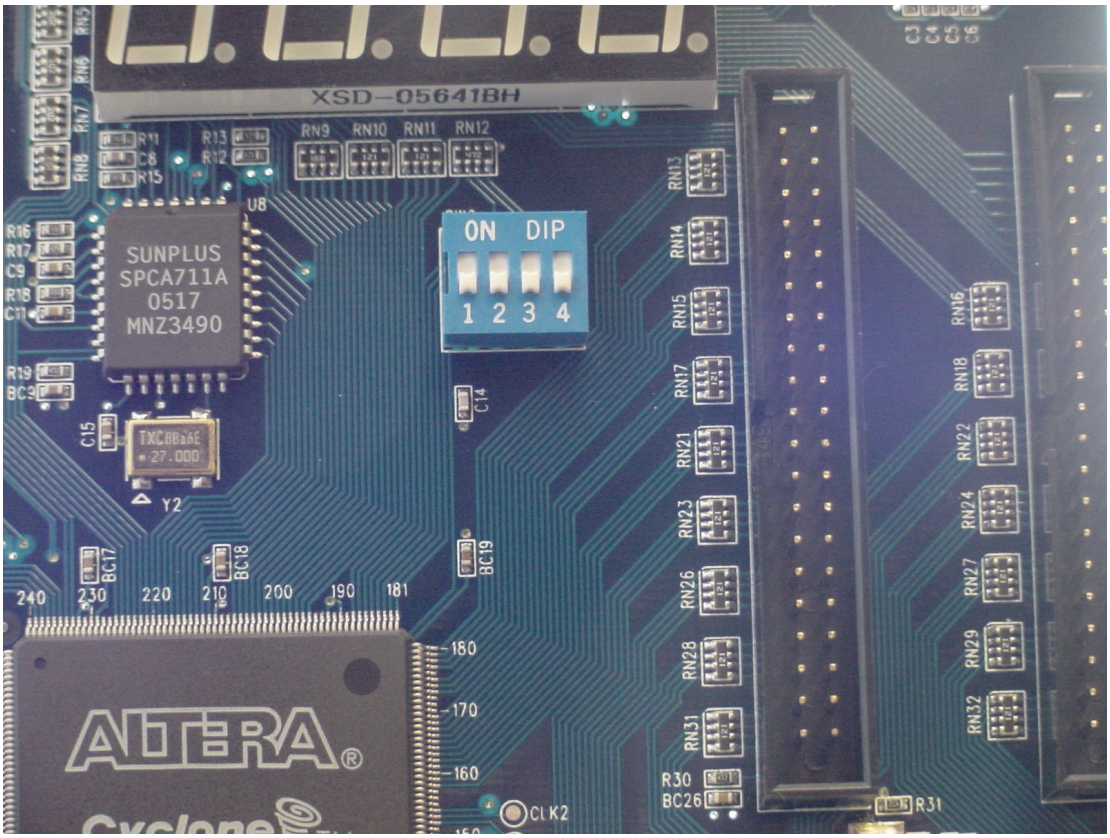
•Connect MultimediO to the labkit using the provided “labkit connector” board and an IDE cable. Make sure that the cable is connected to JP1, not JP2, on the FPGA board.



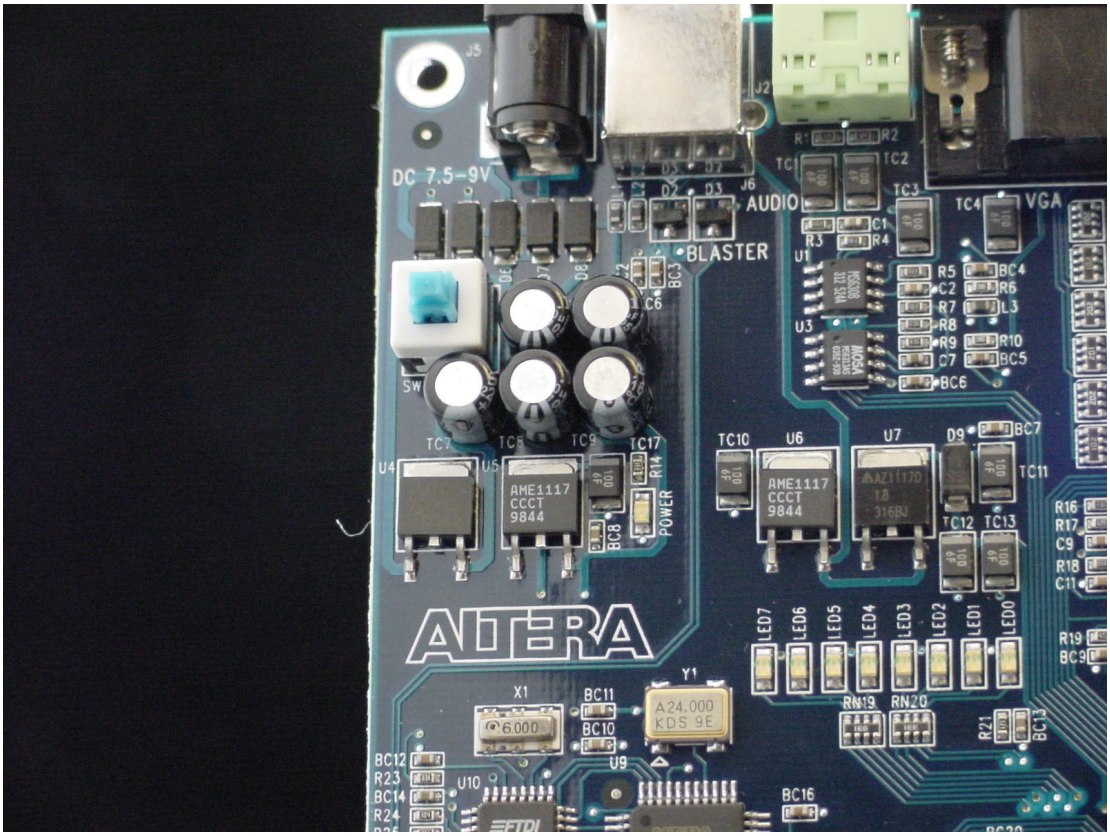


•Make sure the Run/Prog switch is in the “Run” position. Locate the blue and white bank of dip switches. Make sure that all switches are in the off (down) position.

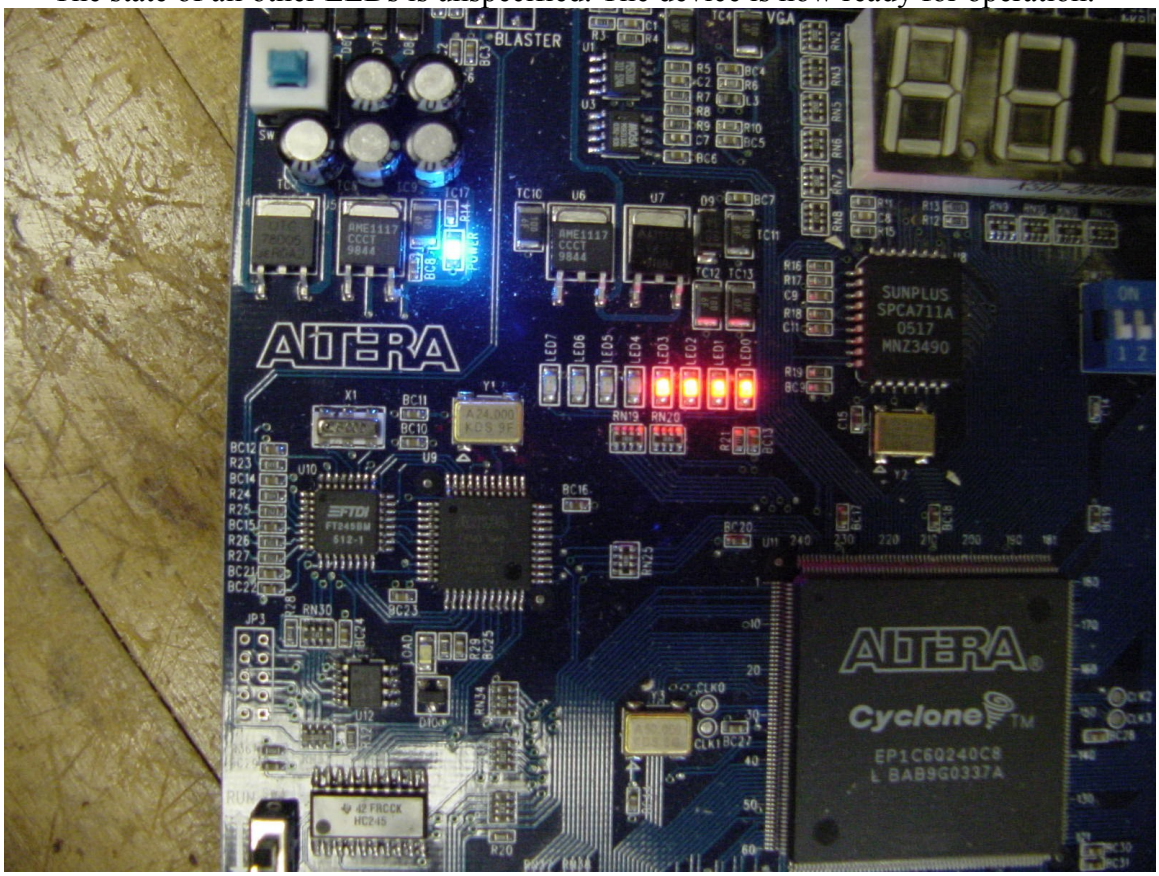




•Turn on power to the labkit, then turn on MultimediO by pressing the blue power button.



•Move switch 1 on the bank of dip switches to the on (up) position. Leave all other dip switches in the off position.
•If everything has been done properly, LED0, LED1, LED2, and LED3 will all be lit. The state of all other LEDs is unspecified. The device is now ready for operation.



Instruction Format

The following table contains a list of all instructions, with corresponding 8 bit command bytes (expressed in hex) and a full explanation of their use. To initiate any of these instructions, simply write the appropriate command byte to the command register (located at fe01h), then write all data needed by the instruction to the data register (located at fe00h) one byte at a time, and finally send a NOP command to the command register. It is important to always send at least one NOP command to the device at the end of an instruction. All opcodes not specified are invalid/reserved for future use.

Name	Command Byte	Description
NOP	00h	Upon receiving this command, the device will ignore all writes to its data register, and will not drive the 8-bit databus. When executing this instruction, the device will continue to drive the display, but will otherwise be completely inactive. If a blank screen is desired, simply move all sprites off screen using the Move Sprite

Name	Command Byte	Description
		command.
Make Sprite This command is used to create a new sprite.	01h	The data written to the data register should have the following form: byte 0: low 8 bits of x coordinate of top left corner of sprite byte 1: 6 zeros, high 2 bits of x coordinate of top left corner of sprite byte 2: low 8 bits of y coordinate of top left corner of sprite byte 3: 6 zeros, high 2 bits of y coordinate of top left corner of sprite byte 4: x length (8 bits) byte 5: y length (8 bits) byte 6: low 8 bits of area byte 7: high 8 bits of area byte 8-n: color information of each pixel, left to right, top to bottom Color information is specified using an 8-bit truecolor RGB scheme in which the first 3 bits correspond to red, the next 3 bits correspond to green, and the final 2 bits correspond to blue. The newly created sprite will have an ID number given by the number of sprite created before it. Thus, the first sprite will be 0, the next sprite will be 1, etc. This ID number is used by the Move Sprite command.
Move Sprite This command is used to move an existing sprite to a new location.	02h	The data written to the data register should be of the form: byte 0: high 2 bits of new x, high 2 bits of new y, sprite number(4 bit) byte 1: low 8 bits of new x byte 2: low 8 bits of new y Where the “sprite number” is the ID number assigned to a given sprite by the Make Sprite command, and “new x” and “new y” refer to the x and y coordinates, respectively, of the top left corner of the sprite. It should be noted that it is perfectly valid to move a sprite off screen.
Read Keyboard	03h	No data should be written to the data register when using this command. After issuing this command, the ascii

Name	Command Byte	Description
This command is used to read the ascii value of a key pressed on the keyboard.		value can be retrieved by reading from the data register. If no key has been pressed, this value will be 00h. Otherwise, it will be the standard ascii value of the first character in the keyboard buffer. Characters are added to the buffer in the order in which they are entered from the keyboard. Each use of the Read Keyboard command removes exactly one character from the buffer, if there are any characters in the buffer. The buffer can store a maximum of 128 ascii characters. While the buffer is full, any key presses will be ignored.
Load Compact Flash Buffer This command is used to load a sector of the Compact Flash card to the buffer.	04h	The data written to the data register should be of the form: byte0: low 8 bits of sector address byte1: next higher 8 bits of sector address byte2: next higher 8 bits of sector address byte3: 4 zeros, high 4 bits of sector address byte4: The literal 01h This command loads the specified 512 byte sector of the compact flash card into a buffer. This buffer can then be accessed using the Read Compact Flash Buffer Command. Note that due to the relatively low speed of Compact Flash, this command may take several machine cycles to execute. In order to determine if the command has finished, read the status byte (by reading the command register). Until the Compact Flash busy bit has cleared, no command should be issued, including the nop needed at the end of all commands.
Read Compact Flash Buffer This command is used to read data from the Compact Flash buffer.	05h	The data written to the data register should be of the form: byte 0: word number This command accesses the specified 16 bit word in the Compact Flash buffer. The Load Compact Flash Buffer command is used to populate this buffer. After a sector is loaded into the buffer, this command can be used as many times as is desired to access individual words in the buffer. After issuing the command, the desired data can be retrieved by reading the data register twice. The first byte read will be the high order byte of the word, the second byte read will be the low order byte of the word.
Make Sprite	06h	The data written to the data register should have the

Name	Command Byte	Description
From CF This command is used to create a new sprite from data stored on the Compact Flash card.		following form: byte 0: low 8 bits of x coordinate of top left corner of pixel byte 1: 6 zeros, high 2 bits of x coordinate of top left corner of pixel byte 2: low 8 bits of y coordinate of top left corner of pixel byte 3: 6 zeros, high 2 bits of y coordinate of top left corner of pixel byte 4: x length (8 bits) byte 5: y length (8 bits) byte 6: low 8 bits of area byte 7: high 8 bits of area byte 8: low 8 bits of sector address byte 9: next higher 8 bits of sector address byte 10: next higher 8 bits of sector address byte 11: 4 zeros, high 4 bits of sector address byte 12: number of adjacent sectors containing data This command essentially combines the Load Compact Flash Buffer, Read Compact Flash Buffer, and Make Sprite commands. The same functionality can be achieved by using those commands individually; however, this command exists for convenience and speed. The sector address referenced above is the first sector that contains pixel information. The pixel information stored in the Compact Flash card must be aligned to the start of the sector. The file stored on the Compact Flash Card should contain raw pixel information only (similar to 24-bit .bmp files, which consist of a short header followed by pixel data). As is the case for the Load Compact Flash Buffer Command, this command may take several cycles to complete. Until the Compact Flash busy bit has cleared, no command should be issued, including the nop needed at the end of all commands.
Edit Sprite This command is used to edit an existing sprite.	07h	The data written to the data register should be of the form: byte0: sprite number byte1: low 8 bits of x coordinate byte2: low 8 bits of y coordinate byte3: 8 zeros byte4: width of rectangle being edited

Name	Command Byte	Description
		byte5: height of rectangle being edited byte6-n: New pixel data. This command is used to edit the pixel data of an existing sprite. It does not change the size of the sprite or move the sprite. The x and y coordinates referenced above refer to the top left corner of the rectangle being edited. These coordinates are in the frame of the sprite, not the absolute reference frame used to position sprites. The width and height parameters specify the size of the rectangle being edited. The remaining bytes are the new RGB values of the pixels, in the same order as the Make Sprite command. NOTE: The width and height parameters must both be even numbers.
Load Audio Data This command is used to transfer audio data into MultimediO's RAM	08h	Byte 0:low 8 bits of start address Byte 1: 4 zeros, high 4 bits of start address Byte 2-n: Audio Data This command is used to load audio data into Multimedio's Audio RAM. The start address specifies the high 12 bits of the 20 bit address at which audio data will start being stored. The low 8 bits are always 00h. If the amount of audio data supplied is sufficiently long that the new entry enters a portion of RAM occupied by another audio file, that other file will be overwritten. Note that the section of RAM being accessed is reserved for Audio only. This command must be used before any attempt is made to play audio. Audio data should be in the .wav file format.
Load Audio Data From CF This command is used to transfer audio data from the Compact Flash card to RAM	09h	Byte 0:Low 8 bits of start address Byte 1: 4 zeros, high 4 bits of start address Byte 2: low 8 bits of sector address Byte 3: next higher 8 bits of sector address Byte 4: next higher 8 bits of sector address Byte 5: 4 zeros, high 4 bits of sector address Byte 6: number of adjacent sectors containing data The term “start address” refers to the high 12 bits of the 20 bit location in the Audio RAM at which the audio data will be stored. The term “sector address” refers to the

Name	Command Byte	Description
		address of the first sector on the Compact Flash Card that contains the desired audio information. This command combines the functionality of the Load Compact Flash Buffer, Read Compact Flash Buffer, and Load Audio Data commands. As is the case for the Load Compact Flash Buffer Command, this command may take several cycles to complete. The status byte should be read until the Compact Flash card is no longer busy before issuing any other command, including the nop command needed at the end of any command.
Play Audio This command is used to play an audio clip	0Ah	The data written to the data register should be of the form: Byte 0:Low 8 bits of start address Byte 1:next 8 bits of start address Byte 2:low 4 bits of end address, high 4 bits of start address Byte 3:next 8 bits of end address Byte 4:high 8 bits of end address This command causes MultimediO to play the audio file located between the start address and the end address in the Audio RAM. The audio file is loaded through the Load Audio Data or the Load Audio Data From CF command.
Set VGA Parameters This command is used to set resolution and background color.	0Bh	The data written to the data register should be of the form: Byte 0:Resolution code Byte 1:RGB color value of background. Allowable values for the resolution code are: 00h: 800x600 01h: 640x480 All other values are not allowed. 800x600 is the default value that the device is set to on power up.
USB Data Transfer This command is used to transfer data to or from the USB port.	0Ch	The data written to the data register should be of the form: Byte 0:Command code Byte 1-n: data bytes (for load write buffer only) The command code specifies which type of USB operation is desired. The valid codes are:

Name	Command Byte	Description
		01h: Load Write Buffer 02h: Send Write Buffer 03h: Read Byte Additional data bytes should only be sent when using the load write buffer operation, which loads a group of bytes to a write buffer, located in MultimediO. This buffer stores data that will eventually be written to the USB port, using the Send Write Buffer command. No more than 64 bytes can be loaded into the buffer at any one time. The status of the buffer can be determined by reading the status byte. The send write buffer operation sends the entire contents of the write buffer to the USB port. Each byte of data is sent serially, most significant bit first. After using this operation, the write buffer will be empty. The read byte command reads a single byte from the read buffer. No additional data should be written to the data register. After writing the command code, the data byte can be retrieved by reading from the data register. This is the only USB operation in which data should be read from the data register.

Appendix I- Sample Code

```
;;This program demonstrates how to create and display a simple sprite on a VGA
;;monitor using MultimediO. The sprite is a red square, located at x=32 pixels, y=128
;;pixels on the screen, with a side length of 32 pixels. The screen is driven at the default
;;resolution of 800x600.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 8000h
mov dptr, #0fe01h ;command register
mov A, #01h; make sprite command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #20h ;x=32, low bits
movx @dptr, A
mov A, #00h ;x=32, high bits
movx @dptr, A
mov A, #80h ;y=128, low bits
movx @dptr, A
mov A, #00h ;y=128, high bits
movx @dptr, A
mov A, #20h ;width=32
movx @dptr, A
mov A, #20h ;height=32
movx @dptr, A
mov A, #00h ;area=1024, low bits
movx @dptr, A
mov A, #04h ;area=1024, high bits
movx @dptr, A
mov A, #0E0h ;red
mov R3, #04h
pixelLoop:
        lcall out256
        djnz R3, pixelLoop
mov dptr, #0fe01h ;command register
mov A, #00h ;nop command
movx @dptr, A
hang:
        NOP
        sjmp hang
;subroutine out256
;outputs the contents of A 256 times to address specified by dptr
out256:
        mov R2, #00h
        out256Loop:
                movx @dptr, A
                djnz R2, out256Loop
ret
```

```
;;This program reads pixel data from a compact flash card  and uses that data to create a
;;sprite by first transferring the data to local ram (the RAM located in the R31JP), then
;;using the Make Sprite command
org 8000h
;load sector into the buffer
mov dptr, #0fe01h ;command register
mov A, #04h; load compact flash buffer command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #05h; low byte of sector address
movx @dptr, A
mov A, #08h; next higher byte of sector address
movx @dptr, A
mov A, #00h; next higher byte of sector address
movx @dptr, A
mov A, #00h; 4 zeros and high nibble of sector address
movx @dptr, A
mov dptr, #0fe01h ;command register
mov A, #00h; nop
movx @dptr, A
mov R1, #02h
mov R0, #00h
pauseLoop0: ;pause a bit to allow compact flash to finish reading
        pauseLoop1:
                nop
        djnz R0,pauseLoop1
djnz R1, pauseLoop0
;read the buffer, transfer to local ram
mov R0, #00h; current word
mov R1, #80h; counter
mov R3, #0e0h; high byte of temp storage
mov R4, #00h; low byte of temp storage
readLoop:
        mov dptr, #0fe01h ;command register
        mov A, #05h; read compact flash buffer command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, R0
        movx @dptr, A
        inc R0
        movx A, @dptr ;read high byte
        mov R2, A
        movx A, @dptr ;read low byte
        mov P1,A
        mov dpl, R4
        mov dph, R3
```

```
        movx @dptr, A
        inc dptr
        mov A, R2
        movx @dptr, A
        inc R4
        inc R4
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop
        movx @dptr, A
djnz R1, readLoop
mov dptr, #0fe01h ;command register
mov A, #01h; make sprite command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #20h ;x=32, low bits
movx @dptr, A
mov A, #00h ;x=32, high bits
movx @dptr, A
mov A, #80h ;y=128, low bits
movx @dptr, A
mov A, #00h ;y=128, high bits
movx @dptr, A
mov A, #10h ;width=16
movx @dptr, A
mov A, #10h ;height=16
movx @dptr, A
mov A, #00h ;area=256, low bits
movx @dptr, A
mov A, #01h ;area=256, high bits
movx @dptr, A
mov R4, #00h; low byte of temp storage
mov R1, #00h; counter
pixelLoop:
        mov dph, R3
        mov dpl, R4
        movx A,@dptr
        mov dptr, #0fe00h ;data register
        movx @dptr, A
        inc R4
djnz R1, pixelLoop
mov dptr, #0fe01h ;command register
mov A, #00h; nop
movx @dptr, A
hang:
        nop
sjmp hang
```

```
;;This program creates a sprite from data stored on the compact flash card. It creates the
;;same sprite as the above program, but does so using the Make Sprite from CF
;;command, which transfers data directly from the CF card to the module responsible for
;;creating sprites.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 8000h
mov dptr, #0fe01h ;command register
mov A, #06h; make sprite from cf command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #28h ;x=40, low bits
movx @dptr, A
mov A, #00h ;x=40, high bits
movx @dptr, A
mov A, #80h ;y=128, low bits
movx @dptr, A
mov A, #00h ;y=128, high bits
movx @dptr, A
mov A, #20h ;width=32
movx @dptr, A
mov A, #20h ;height=32
movx @dptr, A
mov A, #00h ;area=1024, low bits
movx @dptr, A
mov A, #04h ;area=1024, high bits
movx @dptr, A
mov A, #05h; low byte of sector address
movx @dptr, A
mov A, #08h; next higher byte of sector address
movx @dptr, A
mov A, #00h; next higher byte of sector address
movx @dptr, A
mov A, #00h; 4 zeros and high nibble of sector address
movx @dptr, A
mov A, #02h; number of sectors
movx @dptr, A
mov dptr, #0fe01h ;command register
pauseLoop:
        movx A,@dptr ;read status byte
jb acc.1,pauseLoop ;loop until cf card isn't busy
nop ;pause one extra cycle
mov A, #00h; nop
movx @dptr, A
hang:
        NOP
        sjmp hang
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;This program creates a sprite in the same manner as the first demo program, then edits
;;the sprite. Specifically, it changes a small square in the center of the sprite from red to
;;green
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 8000h
mov dptr, #0fe01h ;command register
mov A, #01h; make sprite command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #20h ;x=32, low bits
movx @dptr, A
mov A, #00h ;x=32, high bits
movx @dptr, A
mov A, #80h ;y=128, low bits
movx @dptr, A
mov A, #00h ;y=128, high bits
movx @dptr, A
mov A, #20h ;width=32
movx @dptr, A
mov A, #20h ;height=32
movx @dptr, A
mov A, #00h ;area=1024, low bits
movx @dptr, A
mov A, #04h ;area=1024, high bits
movx @dptr, A
mov A, #0E0h ;red
mov R3, #04h
pixelLoop:
        lcall out256
        djnz R3, pixelLoop
mov dptr, #0fe01h ;command register
mov A, #00h ;nop command
movx @dptr, A
mov A, #07h ;edit sprite command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #00h ;sprite 0
movx @dptr, A
mov A, #08h ;x=8, low bits
movx @dptr, A
mov A, #08h ;y=8, low bits
movx @dptr, A
mov A, #00h ;4 zeros, high bits of x and y
movx @dptr, A
mov A, #10h ;width
```

```
movx @dptr, A
mov A, #10h ;height
movx @dptr, A
mov A, #1ch ;green
lcall out256
mov dptr, #0fe01h ;command register
mov A, #00h ;nop command
movx @dptr, A
hang:
        NOP
        sjmp hang

;subroutine out256
;outputs the contents of A 256 times
;to address specified by dptr
out256:
        mov R2, #00h
        out256Loop:
                movx @dptr, A
                djnz R2, out256Loop
ret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;This program reads pixel data from a compact flash card  and uses that data to create a
;;simple animation of a flying duck.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 8000h
mov R0,#03h
makeLoop: ;creates sprites
        mov dptr, #0fe01h ;command register
        mov A, #06h; make sprite from cf command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #20h ;x=32, low bits
        movx @dptr, A
        mov A, #00h ;x=32, high bits
        movx @dptr, A
        mov A, #80h ;y=128, low bits
        movx @dptr, A
        mov A, #00h ;y=128, high bits
        movx @dptr, A
        mov A, #20h ;width=32
        movx @dptr, A
        mov A, #20h ;height=32
        movx @dptr, A
        mov A, #00h ;area=1024, low bits
        movx @dptr, A
        mov A, #04h ;area=1024, high bits
        movx @dptr, A
        mov A, #03h; low byte of sector address (base)
        add A, R0; low byte of sector address (half offset)
        add A, R0; low byte of sector address (half offset)
        movx @dptr, A
        mov A, #08h; next higher byte of sector address
        movx @dptr, A
        mov A, #00h; next higher byte of sector address
        movx @dptr, A
        mov A, #00h; 4 zeros and high nibble of sector address
        movx @dptr, A
        mov A, #02h; number of sectors
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        pauseLoop:
                movx A,@dptr ;read status byte
        jb acc.1,pauseLoop ;loop until cf card isn't busy
        nop ;pause one extra cycle
        mov A, #00h; nop
        movx @dptr, A
```

```
        mov R1, #20h
djnz R0,makeLoop

mov R1, #20h ;sprite x
mov R2, #80h; sprite y
mov R3, #00h; frame of animation, 0 indexed, frame 1 is same as frame 3, only low 2 bits
matter
animLoop:
        mov R4,#1ah ;counter
        upRight:
                mov R5, #04h; counter
                upRInner:
                        inc R1
                        dec R2
                        lcall moveAll
                        lcall pause20
                djnz R5, upRInner
                mov A,R3
                inc A;update frame
                anl A, #03h ;mask high 6 bits
                mov R3,A
        djnz R4, upRight
        mov R4,#1ah ;counter
        downRight:
                mov R5, #04h; counter
                downRInner:
                        inc R1
                        inc R2
                        lcall moveAll
                        lcall pause20
                djnz R5, downRInner
                mov A,R3
                inc A;update frame
                anl A, #03h ;mask high 6 bits
                mov R3,A
        djnz R4, downRight
        mov R4,#1ah ;counter
        downLeft:
                mov R5, #04h; counter
                downLInner:
                        dec R1
                        inc R2
                        lcall moveAll
                        lcall pause20
                djnz R5, downLInner
                mov A,R3
```

```
                inc A;update frame
                anl A, #03h ;mask high 6 bits
                mov R3,A
        djnz R4, downLeft
        mov R4,#1ah ;counter
        upLeft:
                mov R5, #04h; counter
                upLInner:
                        dec R1
                        dec R2
                        lcall moveAll
                        lcall pause20
                djnz R5, upLInner
                mov A,R3
                inc A;update frame
                anl A, #03h ;mask high 6 bits
                mov R3,A
        djnz R4, upLeft
ljmp animLoop

;subroutine moveAll
;moves active sprite to R1,R2 and innactive sprite offscreen
moveAll:
        mov A, R3
        jz frame0
        dec A
        jz frame1
        dec A
        jz frame2
        frame1:
                mov R0, #01h
                lcall moveSpr
                mov R0, #00h
                lcall moveOff
                mov R0, #02h
                lcall moveoff
        ljmp doneMove
        frame0:
                mov R0, #00h
                lcall moveSpr
                mov R0, #01h
                lcall moveOff
                mov R0, #02h
                lcall moveoff
        ljmp doneMove
        frame2:
```

```
                mov R0, #02h
                lcall moveSpr
                mov R0, #00h
                lcall moveOff
                mov R0, #01h
                lcall moveoff

        doneMove:
ret


;subroutine moveSpr
;moves sprite R0 to R1,R2
moveSpr:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, R0
        movx @dptr, A
        mov A, R1
        movx @dptr, A
        mov A, R2
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop
        movx @dptr, A
ret

;subroutine moveOff
;moves sprite with sprite number R0 offscreen
moveOff:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, R0
        orl A, #30h ;need high bits of y to be 3h
        movx @dptr, A
        mov A, #00h
        movx @dptr, A
        mov A, #80h
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop
        movx @dptr, A
```

```
ret

;subroutine pause20
;pauses for 20 miliseconds(approx)
pause20:
        mov R6, #23h
        pauseLoop0:
                lcall pause256
                djnz R6, pauseLoop0
ret


;subroutine pause256
;pauses for 256 cycles
pause256:
        mov R7, #0ffh
        pauseLoop1:
                nop
                djnz R7,pauseLoop1
ret


test:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #00h
        movx @dptr, A
        mov A, #20h
        movx @dptr, A
        mov A, #20h
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop
        movx @dptr, A

        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #01h
        movx @dptr, A
        mov A, #20h
        movx @dptr, A
        mov A, #80h
```

```
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop
        movx @dptr, A

        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #02h
        movx @dptr, A
        mov A, #80h
        movx @dptr, A
        mov A, #80h
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop
        movx @dptr, A
ret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;This program demonstrates the audio system. It creates an audio file that consists of a
;;simple triangle wave and transfers it to MultimediO. It then issues the
;;command to play that file repeatedly
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 8000h
mov dptr, #0fe01h ;command register
mov A, #08h; load audio data command
movx @dptr, A
mov dptr, #0fe00h ;data register
mov A, #00h; low byte of address
movx @dptr, A
mov A, #00h; 4 zeroes, high 4 bits of address
movx @dptr, A
lcall triaOut
mov dptr, #0fe01h ;command register
mov A, #00h; nop
movx @dptr, A

mov R4, #00h ;counter
mov R3, #04h ;counter
playLpO:
	playLpI:
		mov A, #0ah; play audio
		movx @dptr, A
		mov dptr, #0fe00h ;data register
		mov A, #00h; low byte of start address
		movx @dptr, A
		mov A, #00h; next byte of start address
		movx @dptr, A
		mov A, #00h; low 4 bits of end address, high 4 bits of start address
		movx @dptr, A
		mov A, #10h; next byte of end address
		movx @dptr, A
		mov A, #00h; high byte of end address
		movx @dptr, A
		mov dptr, #0fe01h ;command register
		mov A, #00h; nop
		movx @dptr, A
		lcall pause6
	djnz R4, playLpI
	mov P1, R3
djnz R3, playLpO

hang:
	nop
```

```
sjmp hang
;subroutine triaOut
;outputs one period of triangle wave (128 bytes)
triaOut:
        mov R1, #00h;low byte of audio word
        mov R2, #00h;high byte of audio word
        mov R6, #04h; counter
        outLp:
                mov R5, #20h ;counter
                upLoop:
                        mov A, R1
                        clr C
                        add A, #55h
                        mov R1, A
                        movx @dptr, A
                        mov A, R2
                        addc A, #05h
                        mov R2, A
                        movx @dptr, A
                        mov P1, A
                djnz R5, upLoop
                mov R5, #20h ;counter
                downLoop:
                        mov A, R1
                                clr C
                        subb A, #55h
                        mov R1, A
                                movx @dptr, A
                        mov A, R2
                        subb A, #05h
                        mov R2, A
                        movx @dptr, A
                djnz R5, downLoop
        djnz R6, outLp
ret
;subroutine pause6
;pauses for about 6 ms
pause6:
        mov R0, #00h
        mov R1, #0ah
        pLoop:
                pauseLp:
                        nop
                djnz R0, pauseLp
        djnz R1, pLoop
ret
```

```
;;Lecture Demo
;;The purpose of this program is to demonstrate a variety of MultimediO's functions.
;;Specifically, this program shows how to create, move, and edit sprites, as well as how to
;;load and play sound files and how to get input from the keyboard.
org 8000h
lcall init
mainLp:
        lcall procCmd
        lcall mainPause
sjmp mainLp


;subroutine init
;initializes all
init:
        mov dptr, #0fe01h ;command register
        mov A, #06h; make sprite from cf command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #28h ;x=40, low bits
        movx @dptr, A
        mov A, #00h ;x=40, high bits
        movx @dptr, A
        mov A, #80h ;y=128, low bits
        movx @dptr, A
        mov A, #00h ;y=128, high bits
        movx @dptr, A
        mov A, #0f8h ;width=248
        movx @dptr, A
        mov A, #0feh ;height=254
        movx @dptr, A
        mov A, #10h ;area=62992, low bits
        movx @dptr, A
        mov A, #0f6h ;area=62992, high bits
        movx @dptr, A
        mov A, #93h; low byte of sector address
        movx @dptr, A
        mov A, #06h; next higher byte of sector address
        movx @dptr, A
        mov A, #00h; next higher byte of sector address
        movx @dptr, A
        mov A, #00h; 4 zeros and high nibble of sector address
        movx @dptr, A
        mov A, #7ch; number of sectors
        movx @dptr, A
        mov dptr, #0fe01h ;command register
```

```
        pauseLoop:
                movx A,@dptr ;read status byte
        jb acc.1,pauseLoop ;loop until cf card isn't busy
        nop ;pause one extra cycle
        mov A, #00h; nop
        movx @dptr, A
        mov R7, #00h; stores last created sprite
        mov R6, #00h; indicates that no audio data has been loaded yet
ret

;subroutine procCmd
;processes commands
procCmd:
        lcall readKbrd
        jnz doCmd
        ret
        doCmd:
                cjne A, #31h, notMake;checks for 1(make sprite)
                lcall menuOff
                lcall makeSpri
                lcall menuOn
                ret
        notMake:
                cjne A, #32h, notMove;checks for 2(move sprite)
                lcall menuOff
                lcall moveSpri
                lcall menuOn
                ret
        notMove:
                cjne A, #33h, notEdit;checks for 3(edit sprite)
                lcall menuOff
                lcall editSpri
                lcall menuOn
                ret
        notEdit:
                cjne A, #34h, none;checks for 4(play sound)
                lcall menuOff
                lcall playSnd
                lcall menuOn
                ret
        none:
                ret
ret

;subroutine menuOff
;moves menu offscreen
```

```
menuOff:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #0f0h ;high bits of new x and y, sprite number
        movx @dptr, A
        mov A, #0ffh; low bits of new x
        movx @dptr, A
        mov A, #0ffh; low bits of new y
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop command
        movx @dptr, A
ret

;subroutine menuOn
;moves menu onscreen
menuOn:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #00h ;high bits of new x and y, sprite number
        movx @dptr, A
        mov A, #28h ;x=40, low bits
        movx @dptr, A
        mov A, #80h ;y=128, low bits
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop command
        movx @dptr, A
ret

;subroutine mainPause
;pauses during main loop
mainPause:
        mov R0, #00h
        mov R5, #10h
        mainPz0:
                mainPz1:
                        nop
                djnz R0, mainPz1
        djnz R5, mainPz0
ret
```

```
;subroutine readKbrd
;sends command to read keyboard, transfers data to A
readKbrd:
        mov A, #03h; read keyboard command
        mov dptr, #0fe01h ;command reg
        movx @dptr, A
        mov dptr, #0fe00h; data reg
        movx A, @dptr
        mov R0, A; save value
        mov A, #00h; nop command
        mov dptr, #0fe01h ;command reg
        movx @dptr, A
        mov A, R0; recall value
ret

;subroutine makeSpri
;makes a sprite, specifically a red square with top left
;corner at x=32 pixels,y=128 pixels on and side length 32 pixels
;this sprite will be displayed on the vga monitor
makeSpri:
        mov dptr, #0fe01h ;command register
        mov A, #01h; make sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #20h ;x=32, low bits
        movx @dptr, A
        mov A, #00h ;x=32, high bits
        movx @dptr, A
        mov A, #80h ;y=128, low bits
        movx @dptr, A
        mov A, #00h ;y=128, high bits
        movx @dptr, A
        mov A, #20h ;width=32
        movx @dptr, A
        mov A, #20h ;height=32
        movx @dptr, A
        mov A, #00h ;area=1024, low bits
        movx @dptr, A
        mov A, #04h ;area=1024, high bits
        movx @dptr, A
        mov A, #0E0h ;red
        mov R3, #04h
        pixelLoop:
                lcall out256
                djnz R3, pixelLoop
        mov dptr, #0fe01h ;command register
```

```
        mov A, #00h ;nop command
        movx @dptr, A
        inc R7; increments current sprite number
        makeMenu:
                lcall mainPause2
                lcall readKbrd
        cjne A,#6dh,makeMenu ;checks for M
        lcall colGarb
ret

;subroutine moveSpri
;creates a sprite and moves it around the screen
moveSpri:
        mov dptr, #0fe01h ;command register
        mov A, #01h; make sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #20h ;x=32, low bits
        movx @dptr, A
        mov A, #00h ;x=32, high bits
        movx @dptr, A
        mov A, #80h ;y=128, low bits
        movx @dptr, A
        mov A, #00h ;y=128, high bits
        movx @dptr, A
        mov A, #20h ;width=32
        movx @dptr, A
        mov A, #20h ;height=32
        movx @dptr, A
        mov A, #00h ;area=1024, low bits
        movx @dptr, A
        mov A, #04h ;area=1024, high bits
        movx @dptr, A
        mov A, #0e0h ;red
        mov R3, #04h
        pixelLoopc:
                lcall out256
                djnz R3, pixelLoopc
        mov dptr, #0fe01h ;command register
        mov A, #00h ;nop command
        movx @dptr, A
        inc R7; increments current sprite number
        movLp:
                mov R1, #20h ;x
                mov R2, #80h ;y
                mov R3, #40h ;distance to move
```

```
                upR:
                        inc R1
                        dec R2
                        lcall moveAct
                        jz moveDone
                djnz R3, upR
                mov R3, #40h ;distance to move
                dnR:
                        inc R1
                        inc R2
                        lcall moveAct
                        jz moveDone
                djnz R3, dnR
                mov R3, #40h ;distance to move
                dnL:
                        dec R1
                        inc R2
                        lcall moveAct
                        jz moveDone
                djnz R3, dnL
                mov R3, #40h ;distance to move
                upL:
                        dec R1
                        dec R2
                        lcall moveAct
                        jz moveDone
                djnz R3, upL
        sjmp movLp
        moveDone:
                lcall colGarb
ret

;subroutine moveAct
;moves active sprite to R1, R2
moveAct:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, R7; sprite number
        movx @dptr, A
        mov A, R1; low bits of new x
        movx @dptr, A
        mov A, R2; low bits of new y
        movx @dptr, A
        mov dptr, #0fe01h ;command register
```

```
        mov A, #00h; nop command
        movx @dptr, A
        moveMenu:
                lcall mainPause
                lcall readKbrd
                cjne A,#6dh,notMoveM ;checks for M
                mov A, #00h
                ret
                notMoveM:
                        mov A, #01h
ret

;subroutine playSnd
;plays an audio clip from the Compact Flash card
playSnd:
        mov A, R6
        jnz starPlay
        lcall loadSnd
        starPlay:
                mov dptr, #0fe01h ;command register
                mov A, #0Ah; play audio command
                movx @dptr, A
                mov dptr, #0fe00h ;data register
                mov A, #00h; low byte of start address
                movx @dptr, A
                mov A, #00h; next byte of start address
                movx @dptr, A
                mov A, #00h; low 4 bits of end address, high 4 bits of start address
                movx @dptr, A
                mov A, #0c0h; next byte of end address
                movx @dptr, A
                mov A, #30h; high byte of end address
                movx @dptr, A
                mov dptr, #0fe01h ;command register
                mov A, #00h; nop
                movx @dptr, A

                playMenu:
                        lcall mainPause
                        lcall readKbrd
                cjne A,#6dh,playMenu ;checks for M
ret

;subroutine loadSnd
;loads audio clip
loadSnd:
```

```
        mov R0, #04h ;load count
        mov R1, #3bh; low byte of sector address
        mov R2, #03h; high byte of sector address
        loadLp:
                mov dptr, #0fe01h ;command register
                mov A, #09h; load audio data from cf command
                movx @dptr, A
                mov dptr, #0fe00h ;data register
                mov A, R1 ;low byte of start address
                clr C
                subb A, #3bh
                movx @dptr, A
                mov A, R2 ;high byte of start address
                subb A, #03h
                movx @dptr, A
                mov A, R1; low byte of sector address
                movx @dptr, A
                mov A, R2; next higher byte of sector address
                movx @dptr, A
                mov A, #00h; next higher byte of sector address
                movx @dptr, A
                mov A, #00h; 4 zeros and high nibble of sector address
                movx @dptr, A
                mov A, #0ffh; number of sectors
                movx @dptr, A
                mov dptr, #0fe01h ;command register
                pauseLoopc:
                        movx A,@dptr ;read status byte
                jb acc.1,pauseLoopc ;loop until cf card isn't busy
                nop ;pause one extra cycle
                mov A, #00h; nop
                movx @dptr, A
                mov A, R1
                clr C
                add A, #0ffh
                mov R1, A
                mov A, R2
                addc A, #00h
                mov R2, A
        djnz R0, loadLp
        mov R6, #01h
ret

;subroutine editSpri
;creates a sprite (same as makeSpri)
;then edits that sprite
```

```
editSpri:
        mov dptr, #0fe01h ;command register
        mov A, #01h; make sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #20h ;x=32, low bits
        movx @dptr, A
        mov A, #00h ;x=32, high bits
        movx @dptr, A
        mov A, #80h ;y=128, low bits
        movx @dptr, A
        mov A, #00h ;y=128, high bits
        movx @dptr, A
        mov A, #20h ;width=32
        movx @dptr, A
        mov A, #20h ;height=32
        movx @dptr, A
        mov A, #00h ;area=1024, low bits
        movx @dptr, A
        mov A, #04h ;area=1024, high bits
        movx @dptr, A
        mov A, #0E0h ;red
        mov R3, #04h
        pixelLoopb:
                lcall out256
                djnz R3, pixelLoopb
        mov dptr, #0fe01h ;command register
        mov A, #00h ;nop command
        movx @dptr, A
        inc R7; increments current sprite number
        mov A, #07h ;edit sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, R7 ;sprite number
        movx @dptr, A
        mov A, #08h ;x=8, low bits
        movx @dptr, A
        mov A, #08h ;y=8, low bits
        movx @dptr, A
        mov A, #00h ;8 zeros
        movx @dptr, A
        mov A, #10h ;width
        movx @dptr, A
        mov A, #10h ;height
        movx @dptr, A
        mov A, #1ch ;green
```

```
        mov R0, #00h
        editLp:
                movx @dptr, A
        djnz R0, editLp
        mov dptr, #0fe01h ;command register
        mov A, #00h ;nop command
        movx @dptr, A
        editMenu:
                lcall readKbrd
                cjne A,#6dh,editMenu ;checks for M
        lcall colGarb
ret

;subroutine colGarb
;performs garbage collection, moves last active sprite offscreen
colGarb:
        mov dptr, #0fe01h ;command register
        mov A, #02h; move sprite command
        movx @dptr, A
        mov dptr, #0fe00h ;data register
        mov A, #0f0h ;high bits of new x and y
        add A, R7;sprite number
        movx @dptr, A
        mov A, #0ffh; low bits of new x
        movx @dptr, A
        mov A, #0ffh; low bits of new y
        movx @dptr, A
        mov dptr, #0fe01h ;command register
        mov A, #00h; nop command
        movx @dptr, A
ret

;subroutine out256
;outputs the contents of A 256 times
;to address specified by dptr
out256:
        mov R2, #00h
        out256Loop:
                movx @dptr, A
                djnz R2, out256Loop
ret
```