AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR >

7. BUTTERFLY SPI & AT45 DATAFLASH

A MORON'S GUIDE TO SPI & THE AT45 DATAFLASH v1.2

by RetroDan@GMail.com

CONTENTS:

THE SERIAL PERIPHERAL INTERFACE (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial standard by Motorola. Its popularity has grown and is now available on many chips including the AVRs.

SPI operates in full duplex (two-way) mode with one "Master" chip controlling “Slaves.” It exchanges data eight bits at-a-time with speeds from 1Mhz and up.

SPI uses two shift registers to form an inter-chip circular buffer. As

serial data is shifted from the Master to the Slave, data is being shifted from the Slave back to the Master. The clock signal sent by the Master (SCK) controls the speed and the shifting:

```
  MASTER [0]-----(MOSI)---->[0] SLAVE
   SHIFT [1]  (MASTER-OUT   [1] SHIFT
REGISTER [2]    /SLAVE-IN)  [2] REGISTER
      ^ [3]                [3] |
      | [4]                [4] |
      | [5]  (MASTER-IN    [5] |
      | [6]   /SLAVE-OUT)  [6] V
        [7]<----(MISO)-----[7]
```

When the Master sends a bit to the Slave, a bit from the Slave moves back to the Master at the same time, in a circular buffer across both chips like a large sixteen bit ring. After eight shifts the two bytes are exchanged, whatever was in the Master ends up in the Slave, and the data in the Slave is moved to the Master. So an SPI read and write are the same command.

```
 MASTER MOSI [   ]------------->[  ] SLAVE
 MASTER MISO [   ]<-------------[  ]
SERIAL CLOCK [SCK]--->--->-->-->[SCK]
```

SPI & THE BUTTERFLY

The Atmega169 contained on the Butterfly Demo Board contains an SPI subsystem, and to send a byte to the AT45 we load it into a data register and the system takes care of the clocking and shifting as bytes are exchanged. This can be used to either read or write from the AT45 since both operation are the same command.

To set-up for an SPI write we configure the Data Direction Register (DDR_SPI) such that the MOSI and SCK lines are used for output. Then we activate the SPI module by setting the SPE bit and the Master Mode MSTR bit in the SPI Configuration Register (SPCR). At the same time we set the pre-scaler/divider to divide-by-sixteen by setting the (SPR0) bit:

```
LDI A,(1<<DD_MOSI)|(1<<DD_SCK)
OUT DDRB,A        ;SET MOSI & SCK PINS FOR OUTP
```

```
      LDI A,0b0101_0001 ;(SPE,MSTR,SPR0)
      OUT SPCR,A        ;ENABLE SPI AS MASTER...
                        ;AND SET PRESCALER TO /16
```

Once the SPI system is active, we put the data into the SPI Data Register (SPDR) and it is automatically transmitted. We wait for the system to complete the transfer by waiting for the SPI Flag (SPIF) to be set.

```
      OUT SPDR,A       ;LOAD BYTE TO TRANSMIT
WAIT: SBIS SPSR,SPIF   ;WATCH SPI FLAG
      RJMP WAIT        ;WAIT FOR COMPLETION
```

THE AT45 DATAFLASH

The AT45DB041B DataFlash is a popular memory chip that is included on the AVR Butterfly board and some other Atmel products. It contains four megabits of storage (512K Bytes) and uses SPI for communications. It is hard wired as an SPI Slave to the main processor.

The DataFlash contains 2,048 pages of 264 bytes each for a total of 540,672 bytes and two internal RAM buffers of 264 bytes each.

```
         .---> 264 BYTE BUFFER1 ----->  FLASH MEMORY
SPI ---->|                              2048 PAGES
         `---> 254 BYTE BUFFFER2 ---->  264 BYTES/PAG
```

Most commands to the AT45 take the this form: an OPCODE byte followed by three or more bytes. The exception is the status command which is only one byte long.

```
        [COMMAND-OPCODE][BYTE][BYTE][BYTE]
```

When used for addressing the bytes are defined as follows:

```
  [rrrr_pppp] [pppp_pppb] [bbbb_bbbb]
Where: r – reserved
       p - page number
       b – byte address within page
```

Note that both the page address and byte-address span over two bytes. If you are not going to use the extra 8 bytes per page, you can

address the first 256 bytes of each page by just using the lower byte.

A TUNE PLAYER PROGRAM

To test our read & write of the AT45 we first construct a small program that plays a tune on the Butterfly's speaker. Later we develop routines to write the tune data to the DataFlash and finally we will read-in the data from the DataFlash and play it on the speaker.

First we tell the assembler to include the definitions for the ATmega169 processor of the Butterfly from the file M169DEF.INC, we define two registers that we will be using “A” & “B” and we set a constant SPEED that controls the tone and speed of the tune.

```
.INCLUDE "M169DEF.INC"       ;AVR ATMEGA169 (BUTTERFL
.DEF A       = R16           ;GENERAL PURPOSE ACCUMUL
.DEF B       = R18
.SET SPEED   = 16
```

Since we are not using any interrupts we can start our program in the bottom of RAM at location $0000. First we set-up a stack in the top of RAM:

```
.ORG $0000
RESET:
        LDI  A,LOW(RAMEND)   ;SET-UP A RETURN STACK..
        OUT SPL,A            ;AT THE TOP OF MEMORY
        LDI A,HIGH(RAMEND)
        OUT SPH,A
```

The speaker on the Butterfly is connected to Port B, Pin 5 so we set the Data Direction Register for Port B (DDRB) with a one in the 5th bit.

```
        SBI   DDRB,5         ;SET PORTB5 FOR OUTPUT T
```

We are using the Z pointer register to access the tune data so we set it to point at the first note:

```
        RCALL SET_POINTER    ;SET Z-POINTER TO SONG D
```

The main loop of the program fetches a note from the data area through the Z pointer and holds the tone for a short duration, then loops back to do it again:

```
MLUPE:  RCALL FETCH_NOTE     ;GRAB A NOTE
        RCALL HOLD_TONE      ;PLAY IT
         RJMP MLUPE          ;LOOP-BACK DO IT AGAIN
```

This routine fetches a note from our tune data and increments the Z-pointer. It looks at the note and if it is zero, we have hit the end of the tune, so it resets the pointer and re-fetches the first note:

```
FETCH_NOTE:
         LPM A,Z+            ;FETCH A NOTE & INCREMEN
         CPI A,0             ;IS THE NOTE ZERO?
          BRNE NOT_ZERO      ;IF NOT ZERO THEN RETURN
         RCALL SET_POINTER   ;IF ZERO/END RESET POINT
          RJMP FETCH_NOTE    ;GO BACK AND READ FIRST
NOT_ZERO:  RET
```

This routine sets the Z-pointer to the first note of our tune:

```
SET_POINTER:
         LDI  ZL,LOW(SONG_DATA*2)  ;SET THE Z-POINTER
         LDI  ZH,HIGH(SONG_DATA*2)
          RET
```

The HOLD_TONE routine calls the routine that toggles the speaker 255 times:

```
HOLD_TONE:                    ;SPEAKER TOGGLE CALLED
HT_LUPE: RCALL FREQ           ;CLICK THE SPEAKER
           DEC R10
            BRNE HT_LUPE
             RET                    ;RETURN
```

This is the heart of the sound producing part of the program. It toggles the speaker with the SBI command, then it executes a small pause controlled by the SPEED constant in the “B” register. Then it executes another pause routine controlled by the contents of the “A” register. The result if called repeatedly is a tone on the speaker with a frequency that is globally controlled by the SPEED constant and individual notes controlled by the data in the “A” register:

```
FREQ:   PUSH  A               ;SAVE TONE/FREQUENCY
        SBI   PINB,5          ;TOGGLE SPEAKER
FLUPE:  LDI   B,SPEED         ;WAIT BETWEEN CLICKS..
```

```
FLUP2:  DEC   B                ;DETERMINS FREQUENCY..
         BRNE FLUP2            ;BASED ON VALUE OF "A"
        DEC   A                ;SUBTRACT ONE FROM A
         BRNE FLUPE            ;WAIT UNTIL IT REACHES
        POP   A                ;RESTORE TONE
             RET
```

If we put all these pieces of code together we have the following program:

```
;---------------------------------------;
; SONG PROGRAM TO TEST AT45 READ/WRITES ;
;---------------------------------------;

.INCLUDE "M169DEF.INC"       ;AVR ATMEGA169 (BUTTERFL
.DEF A       = R16           ;GENERAL PURPOSE ACCUMUL
.DEF B       = R18
.SET SPEED   = 16

.ORG $0000
RESET:
        LDI  A,LOW(RAMEND)   ;SET-UP A RETURN STACK..
        OUT SPL,A            ;AT THE TOP OF MEMORY
        LDI A,HIGH(RAMEND)
        OUT SPH,A
        SBI   DDRB,5         ;SET PORTB5 FOR OUTPUT T
        RCALL SET_POINTER    ;SET Z-POINTER TO SONG D

MLUPE:  RCALL FETCH_NOTE     ;GRAB A NOTE
        RCALL HOLD_TONE      ;PLAY IT
         RJMP MLUPE          ;LOOP-BACK DO IT AGAIN

FETCH_NOTE:
         LPM A,Z+            ;FETCH A NOTE & INCREMEN
         CPI A,0             ;IS THE NOTE ZERO?
          BRNE NOT_ZERO      ;IF NOT ZERO THEN RETURN
         RCALL SET_POINTER   ;IF ZERO/END RESET POINT
          RJMP FETCH_NOTE    ;GO BACK AND READ FIRST
NOT_ZERO:  RET

SET_POINTER:
```

```
         LDI  ZL,LOW(SONG_DATA*2)  ;SET THE Z-POINTER
         LDI  ZH,HIGH(SONG_DATA*2)
          RET

HOLD_TONE:                      ;SPEAKER TOGGLE CALLED
HT_LUPE: RCALL FREQ             ;CLICK THE SPEAKER
             DEC R10
          BRNE HT_LUPE
           RET                  ;RETURN

FREQ:   PUSH  A                 ;SAVE TONE/FREQUENCY
        SBI   PINB,5            ;TOGGLE SPEAKER
FLUPE:  LDI   B,SPEED           ;WAIT BETWEEN CLICKS..
FLUP2:  DEC   B                 ;DETERMINS FREQUENCY..
         BRNE FLUP2             ;BASED ON VALUE OF "A"
        DEC   A                 ;SUBTRACT ONE FROM A
         BRNE FLUPE             ;WAIT UNTIL IT REACHES
        POP   A                 ;RESTORE TONE
             RET

;-------------[ D A T A   A R E A ]-------------

SONG_DATA:
.DB  106,106,106,106,106,106
.DB  127,127,127,127,127,127
.DB  159,159,159,159,159,159
.DB  213,213,213,213,213,213
.DB  169,169,169,169,159,159
.DB  190,190,190,190,159,159
.DB  213,213,213,213,213,213
.DB  213,213,213,213,213,213
.DB  0,0
```

Assuming we entered the program correctly, the speaker should play a small tune.

WRITE/READ BLOCK FROM AT45 DATAFLASH

Previously we played a small tune stored in program memory. We want to write this tune data to the first page of the AT45 DataFlash,

then read it back in from the DataFlash and play it on the speaker. If the tune played is the same we know we were successful in writing and reading back our data.

First we initialize the AT45, then we put it into sequential-write mode and copy our tune data to the DataFlash. We wait 20ms for the write to complete, then we put the AT45 into sequential-read mode and go into an infinite loop that reads-back the data and plays it on the speaker. If we can hear our tune, we knows it was written and read-back:

```
        RCALL AT45_INIT         ;INITIALIZE THE AT45
        RCALL AT45_SEND_COMMAND ;PUT AT45 IN WRITE MO
        RCALL AT45_WRITE_256    ;WRITE TUNE DATA
MAIN:   RCALL AT45_READ_COMMAND ;PUT 1T45 IN READ MOD
MLUPE:  RCALL FETCH_NOTE        ;FETCH NOTE FROM AT45
        RCALL HOLD_TONE         ;PLAY IT
         RJMP MLUPE             ;LOOP-BACK DO IT AGAI
```

INITIALIZING THE SPI

To set-up the SPI system we first make sure that Port B pins 0-2 are configured for outputs, then we set the SPI Enable Bit (SPE), the Master-Mode Bit (MSTR) and the Phase and Polarity Bits (CPHA) & (CPOL) are all set to one in the SPI Control Register (SPCR). This configured the SPI as the Master and SPI Mode Three:

```
AT45_INIT:
          SBI  DDRB,PORTB0        ;CONNECTED TO AT45
          SBI  DDRB,PORTB1        ;CONNECTED TO CLOCK
          SBI  DDRB,PORTB2        ;MASTER-OUT, SLAVE
          LDI   A,0b0101_1100     ;SPE,MSTR,CPHA,CPOL
          OUT   SPCR,A            ;ENABLE SPI AS MAST
             RET
```

PREPARING THE AT45 FOR SEQUENTIAL WRITE

To prepare the AT45 for a command op-code we first toggle the chip select line (/CS) connected to Port B, Pin 0 to get its attention, then we

send the command for sequential write ($82) followed by a three-byte address of where to start. Since we are starting at byte-zero of page-zero, we send all zeros:

```
AT45_SEND_COMMAND:
       SBI   PORTB,PORTB0     ;BRING /CS HIGH...
       CBI   PORTB,PORTB0     ;THEN DROP IT (SIGNALS
       LDI   A,0x82           ;SEND SEQUENTIAL WRITE
       RCALL AT45_SEND
       CLR   A                ;SEND 1ST ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A                ;SEND 2ND ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A                ;SEND 3RD ADDRESS BYTE
       RCALL AT45_SEND
        RET
```

SEQUENTIAL BLOCK WRITE TO AT45

To write our block of data to the DataFlash we first point the Z-Register to the start of our tune data, then we use the LPM command to read our data into the “A” register and increment the Z-Pointer. When we write to the AT45 we also receive a byte at the same time, we don't use this data this time but if we are going to check for zero to mark the end of our block of code, we have to save the byte in “A” on the stack. Then we send our byte to the DataFlash, then we restore “A” and see if what we sent was a zero. If not we go back and write more bytes:

```
AT45_WRITE_256:
        RCALL SET_POINTER         ;POINTS THE Z-REGIS
W256LP: LPM   A,Z+                ;FETCH BYTE FROM TU
        PUSH  A                   ;SAVE BYTE
        RCALL AT45_SEND           ;SEND IT TO AT45
        POP   A                   ;RESTORE BYTE
        CPI   A,0                 ;CHECK IF AT END
         BRNE W256LP              ;IF NOT AT END, GO
```

When we hit zero, the end of our block of data, we signal we are done by taking the chip select line (/CS) high and wait for the AT45 to finish.

```
        SBI   PORTB,PORTB0        ;BRING /CS HIGH = E
```

```
        RCALL WAIT_20MS           ;WAIT 20MS FOR WRIT
         RET
```

PREPARING THE AT45 FOR SEQUENTIAL READ

Preparing the AT45 for a sequential read is very similar to setting it up for the sequential write. We toggle the chip select line (/CS) to prepare the DataFlash for a command. We send the op-code for sequential write ($52) then we send three address byte, since we are starting at byte zero or page zero we send all zeros. However, the sequential read command is eight bytes long so we send an additional four bytes of zero:

```
AT45_READ_COMMAND:
       SBI   PORTB,PORTB0   ;BRING /CS HIGH
       CBI   PORTB,PORTB0   ;NOW DROP IT, SIGNALS STA
       LDI   A,0x52         ;READ COMMAND OPCODE
       RCALL AT45_SEND
       CLR   A              ;SEND 1ST ADDRESS BYTE rr
       RCALL AT45_SEND
       CLR   A              ;SEND 2ND ADDRESS BYTE pp
       RCALL AT45_SEND
       CLR   A              ;SEND 3RD ADDRESS BYTE bb
       RCALL AT45_SEND
       CLR   A
       RCALL AT45_SEND      ;SEND DUMMY
       CLR   A
       RCALL AT45_SEND      ;SEND DUMMY
       CLR   A
       RCALL AT45_SEND      ;SEND DUMMY
       CLR   A
       RCALL AT45_SEND      ;SEND DUMMY
        RET
```

THE READ/WRITE ROUTINE

Since data is exchanged when we talk to the AT45, the read and write routine is the same code. We put our data byte into the SPI Data Register (SPDR) which triggers the SPI system in the ATmega169 to

automatically exchange a byte with the DataFlash. We poll the SPI Flag (SPIF) to wait for the shifting to complete. Then we read in the returned byte from the SPDR Register:

```
AT45_READ:
AT45_SEND:
      OUT   SPDR,A           ;SEND OUT A BYTE TO AT45
ALUPE:
      IN   A,SPSR            ;WATCH THE SPIF FLAG
      SBRS  A,SPIF           ;0=BUSY
       RJMP ALUPE
      IN    A,SPDR           ;GRAB INCOMING BYTE
       RET
```

COMPLETE READ/WRITE BLOCK PROGRAM

```
;---------------------------------------;
; SONG PROGRAM TO TEST AT45 READ/WRITES ;
;---------------------------------------;

.INCLUDE "M169DEF.INC"         ;AVR ATMEGA169 (BUTTER
.DEF A       = R16             ;GENERAL PURPOSE ACCUM
.DEF B       = R18
.DEF N       = R20
.SET SPEED   = 16

.ORG $0000
RESET:
        LDI   A,LOW(RAMEND)     ;SET-UP A RETURN STAC
        OUT   SPL,A             ;AT THE TOP OF MEMORY
        LDI   A,HIGH(RAMEND)
        OUT   SPH,A
        SBI   DDRB,5            ;SET PORTB5 FOR OUTPU
        RCALL AT45_INIT         ;INITIALIZE THE AT45
        RCALL AT45_SEND_COMMAND ;PUT AT45 IN WRITE MO
        RCALL AT45_WRITE_256    ;WRITE TUNE DATA

;--------------;
; MAIN PROGRAM ;
;--------------;
```

```
MAIN:   RCALL AT45_READ_COMMAND ;PUT AT45 IN READ MOD
MLUPE:  RCALL FETCH_NOTE        ;FETCH NOTE FROM AT45
        RCALL HOLD_TONE         ;PLAY IT
         RJMP MLUPE             ;LOOP-BACK DO IT AGAI


;----------------------;
; S U B R O U T I N E S ;
;----------------------;
;--------------------------------------;
; FETCH A NOTE AND PLAY IT FOR 255 LOOPS ;
; AT END IT RESETS POINTER AND CONTINUES ;
;--------------------------------------;
FETCH_NOTE:
         RCALL AT45_READ
         CPI   A,0              ;IS THE NOTE ZERO?
          BRNE NOT_ZERO         ;IF NOT ZERO THEN RE
         RCALL SET_POINTER      ;IF ZERO/END RESET P
         RCALL AT45_READ_COMMAND
          RJMP FETCH_NOTE       ;GO BACK AND READ FI
NOT_ZERO:  RET

;------------------------;
; RESET THE DATA POINTER ;
;------------------------;
SET_POINTER:
         LDI  ZL,LOW(SONG_DATA*2)  ;SET THE Z-POINTER
         LDI  ZH,HIGH(SONG_DATA*2)
          RET
;----------------------------------------;
; TOGGLES SPEAKER 255 TIMES, NOTE IS IN "A" ;
;----------------------------------------;
HOLD_TONE:                      ;SPEAKER TOGGLE CALLE
HT_LUPE: RCALL FREQ             ;CLICK THE SPEAKER
             DEC   R10
          BRNE HT_LUPE
           RET                  ;RETURN

;----------------------------;
; MAKES ONE TOGGLE OF SPEAKER ;
```

```
;-----------------------------;
FREQ:   PUSH  A                 ;SAVE TONE/FREQUENCY
        SBI   PINB,5            ;TOGGLE SPEAKER
FLUPE:  LDI   B,SPEED           ;WAIT BETWEEN CLICKS.
FLUP2:  DEC   B                 ;DETERMINS FREQUENCY.
         BRNE FLUP2             ;BASED ON VALUE OF "A
        DEC   A                 ;SUBTRACT ONE FROM A
         BRNE FLUPE             ;WAIT UNTIL IT REACHE
        POP   A                 ;RESTORE TONE
             RET

;---------------------------------;
; A T 4 5    S U B R O U T I N E S ;
;---------------------------------;

;-----------------------------------------------;
; INITIALIZE SPI FOR AT45 THANKS TO CHUCK BAIRD ;
;-----------------------------------------------;
AT45_INIT:
           SBI  DDRB,PORTB0         ;CONNECTED TO AT4
           SBI  DDRB,PORTB1         ;CONNECTED TO CLO
           SBI  DDRB,PORTB2         ;MASTER-OUT, SLAV
           LDI   A,0b0101_1100      ;(SPE,MSTR,CPHA,C
           OUT   SPCR,A             ;ENABLE SPI AS MA
            RET

;---------------------------------------;
; SEND A FULL 4-BYTE WRITE COMMAND TO AT45 ;
;---------------------------------------;
AT45_SEND_COMMAND:
       SBI   PORTB,PORTB0      ;BRING /CS HIGH...
       CBI   PORTB,PORTB0      ;THEN DROP IT (SIGNALS
       LDI   A,0x82            ;SEND SEQUENTIAL WRITE
       RCALL AT45_SEND
       CLR   A                 ;SEND 1ST ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A                 ;SEND 2ND ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A                 ;SEND 3RD ADDRESS BYTE
       RCALL AT45_SEND
```

```
        RET

;-----------------------------------------------;
; SEND A FULL 7-BYTE SERIAL READ COMMAND TO AT45 ;
; PHONEM MUST CONTAIN PAGE NUMBER               ;
;-----------------------------------------------;
AT45_READ_COMMAND:
       SBI   PORTB,PORTB0     ;BRING /CS HIGH
       CBI   PORTB,PORTB0     ;NOW DROP IT, SIGNALS S
       LDI   A,0x52           ;READ COMMAND OPCODE
       RCALL AT45_SEND
       CLR   A                ;SEND 1ST ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A                ;SEND 2ND ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A                ;SEND 3RD ADDRESS BYTE
       RCALL AT45_SEND
       CLR   A
       RCALL AT45_SEND        ;SEND DUMMY
       CLR   A
       RCALL AT45_SEND        ;SEND DUMMY
       CLR   A
       RCALL AT45_SEND        ;SEND DUMMY
       CLR   A
       RCALL AT45_SEND        ;SEND DUMMY
        RET

;-------------------------;
; WRITE 256 BYTES TO AT45 ;
;-------------------------;
AT45_WRITE_256:
       RCALL SET_POINTER         ;POINTS THE Z-REGIS
W256LP: LPM   A,Z+               ;FETCH BYTE FROM TU
       PUSH  A                   ;SAVE BYTE
       RCALL AT45_SEND           ;SEND IT TO AT45
       POP   A                   ;RESTORE BYTE
       CPI   A,0                 ;CHECK IF AT END
        BRNE W256LP              ;IF NOT AT END, GO
       SBI   PORTB,PORTB0        ;BRING /CS HIGH = E
       RCALL WAIT_20MS           ;WAIT 20MS FOR WRIT
```

```
        RET

;-----------------------------------;
; SEND & RECIEVE ONE BYTE FROM AT45 ;
; THANKS TO CHUCK BAIRD             ;
;-----------------------------------;
AT45_READ:
AT45_SEND:
       OUT   SPDR,A           ;SEND OUT A BYTE TO AT4
ALUPE: IN    A,SPSR           ;WATCH THE SPIF FLAG
       SBRS  A,SPIF           ;0=BUSY
        RJMP ALUPE
       IN    A,SPDR           ;GRAB INCOMING BYTE
        RET

;---------------;
; WAIT ROUTINES ;
;---------------;
WAIT_10US:
           LDI   A,30              ;WAIT AT LEAST 10
WT10L: DEC   A
            BRNE WT10L

WAIT_20MS:
        LDI   XH,0xB3         ;WAIT 20ms (45,824 LOOP
WT20L:  SBIW  XH:XL,1
         BRNE WT20L
          RET

;-------------[ D A T A   A R E A ]-------------

SONG_DATA:
.DB  106,106,106,106,106,106
.DB  127,127,127,127,127,127
.DB  159,159,159,159,159,159
.DB  213,213,213,213,213,213
.DB  169,169,169,169,159,159
.DB  190,190,190,190,159,159
.DB  213,213,213,213,213,213
.DB  213,213,213,213,213,213
```

```
.DB  0,0
```

Comments

You do not have permission to add comments.

Sign in | Recent Site Activity | Report Abuse | Print Page | Powered By Google Sites