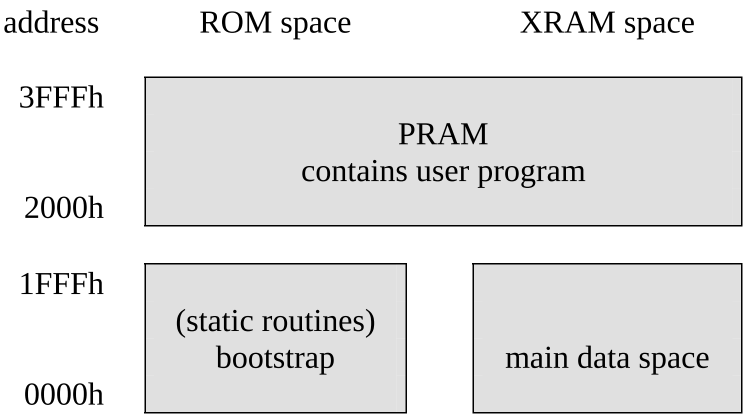
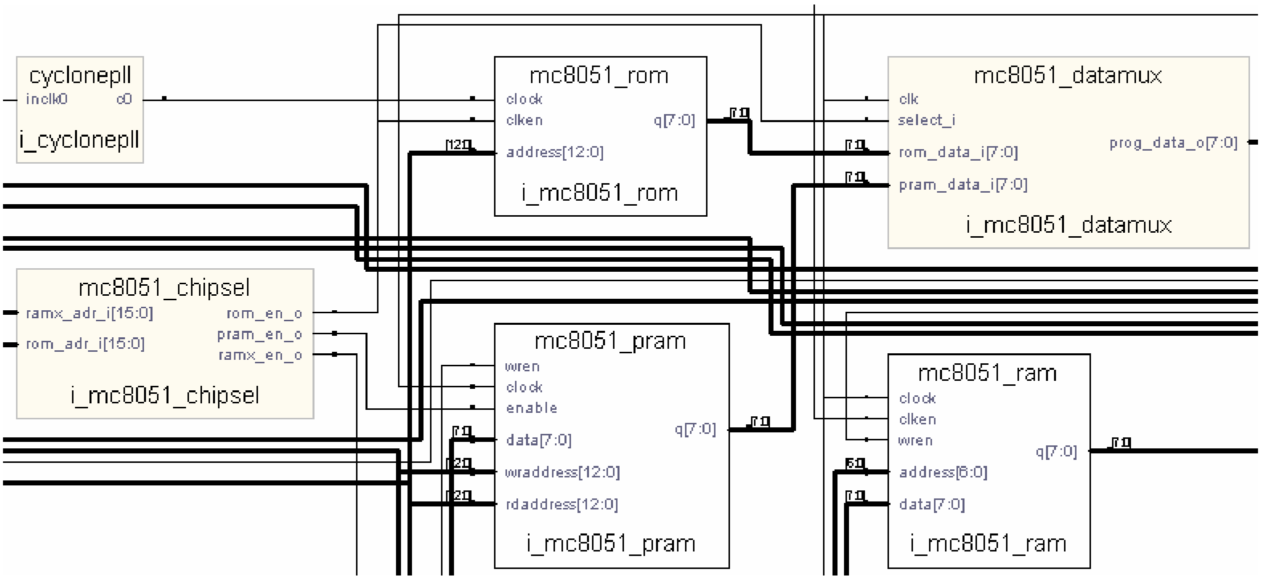
General Description

The following program allows the MC8051 microcontroller to load most of its code into a part of the external data memory (XRAM) over a serial link after power up. This program can be then executed out of the program memory (PRAM) for normal operation.
Any static low level routines that are unlikely to change over time can be fixed into the permanent program memory (ROM) along with the bootstrap loader which is used to load the main routine which calls the static parts of the program into the PRAM.
In the following, the memory map of the reference design is listed.



<u>Architectural Changes</u>

To apply the bootstrap to the MC8051, two more entities have to be added to the top level design. Since there is the need of writing to and reading from the PRAM, it is located at both memory spaces. Due to this fact, a dual ported RAM is used for the PRAM entity. As there are also memory blocks that are dedicated to only one of the memory spaces (ROM or XRAM), a chip selection has to be applied. For the ROM section it is also necessary to select the source, from which the code should be taken (ROM or PRAM). This is done by a simple data multiplexer.



User Program

For a correct operation of the user program in conjunction with the bootstrap, the origins for the jumps to the interrupt service routines and the main startup routine should be recalculated. For the reference design, the base address of the PRAM has to be added to all addresses mentioned in the user program. Note that Timer 1 is still running as baudrate timer at the startup of the user program. The following section shows the beginning of an user program.

```
ORG 02000h
LJMP Main

ORG 02003h      ; jumps to the ISRs
LJMP Ext0

...

ORG 02023h
LJMP Ser0

ORG 02050h      ; user program startup
Main: CLR TR1   ; T1 is still running
...
```


Download Protocol

The bootstrap program allows downloading a hexadecimal Intel Hex file over an asynchronous serial link to the PRAM of the MC8051. In the reference design, the serial port is configured for the 8–N–1 format at 4800 baud. No hardware handshaking between the MC8051 and the host is implemented.
The bootstrap loader is configured to remap all interrupt vectors to the downloaded program.

- When the bootstrap program starts up, it sends a prompt character '=' up the serial link to the host.

- The host may then send the hexadecimal program file down the serial link.

- At any time, the host may send an escape character to abort and restart the download process from scratch, beginning from the prompt. This procedure may be used to restart if a download error occurs.

- At the end of a Hex file download, a colon prompt ':' is returned. If an error occurred, a flag value will also be returned. The flag is a bit map of possible exceptions and represents more than one problem.

  01h non hexadecimal characters found embedded in a data line
  02h bad record type found
  04h incorrect line checksum found
  08h no data found
  10h incremented address overflowed
  20h data write did not verify correctly

- If an error occurs, the bootstrap program will refuse to execute the downloaded program. The download may be retried by first sending an escape character. Until the escape is received, the bootstrap program will refuse to accept any data and will echo a question mark '?' for any character sent.

- After a valid file was downloaded, the bootstrap program will send a message containing the file checksum. This is the arithmetic sum of all of the data bytes embedded in the Hex file lines truncated to 16 bits. This checksum appears in parentheses: '(abcd)'. The execution of the program may then be started by telling the bootstrap program the correct starting address. The format for this is to send a slash '/' followed by the address in ASCII hexadecimal, followed by a carriage return. For the reference design, '/2000<CR>' should be sent over the serial link.

- If the address is accepted, a sign ("@") is returned before executing the jump to the downloaded file.

MC8051 Bootstrap Program

The following assembler code should be executed at the startup of the MC8051 to store a program, which is received over the serial interface, into the PRAM.

```
LF        EQU 0Ah     ; line feed char
CR        EQU 0DH     ; carriage return char
ESC       EQU 1Bh     ; escape char
StartChar EQU ':'     ; line start char, HEX file
Slash     EQU '/'     ; load startup address char
Skip      EQU 13      ; skip state value

Ch        DATA 0Fh    ; last char received
State     DATA 10h    ; state in process
DataByte  DATA 11h    ; last data byte received
ByteCount DATA 12h    ; data byte count, HEX line
HighAddr  DATA 13h    ; address of data byte read
LowAddr   DATA 14h
RecType   DATA 15h    ; HEX line record type
ChkSum    DATA 16h    ; calculated checksum
HASave    DATA 17h    ; address from last data line
LASave    DATA 18h
FilChkHi  DATA 19h    ; file checksum
FilChkLo  DATA 1Ah

Flags     DATA 20h    ; state condition flags
HexFlag   BIT Flags.0 ; hex char found
EndFlag   BIT Flags.1 ; end record found
DoneFlag  BIT Flags.2 ; process complete

EFlags      DATA 21h      ; exception flags
ErrFlag1    BIT EFlags.0  ; non-hex char found
ErrFlag2    BIT EFlags.1  ; invalid record type
ErrFlag3    BIT EFlags.2  ; wrong line checksum
ErrFlag4    BIT EFlags.3  ; no data received
ErrFlag5    BIT EFlags.4  ; address overflow
ErrFlag6    BIT EFlags.5  ; data memory verify error
DatSkipFlag BIT Flags.3   ; ignore data

                        ; remap interrupt vectors
ExInt0 EQU 02003h       ; X0
T0Int  EQU 0200Bh       ; T0
ExInt1 EQU 02013h       ; X1
T1Int  EQU 0201Bh       ; T1
SerInt EQU 02023h       ; S0

ORG 0000h
LJMP Start    ; start bootstrap

ORG 0003h
 LJMP ExInt0  ; call ISR of X0
RETI

ORG 000Bh
 LJMP T0Int   ; call ISR of T0
```



```
RETI

ORG 0013h
 LJMP ExInt1  ; call ISR of X1
RETI

ORG 001Bh
 LJMP T1Int   ; call ISR of T1
RETI

ORG 0023h
 LJMP SerInt  ; call ISR of S0
RETI

ORG 00050h
Start: MOV IE,#0   ; set up all regs
 MOV SP,#5Fh
 ACALL SerStart    ; setup serial port
 ACALL CRLF        ; send <CRLF>
 MOV A,#'='
 ACALL PutChar     ; send prompt
 ACALL HexIn       ; read HEX file

 ACALL ErrPrt      ; send error flags

 MOV A,EFlags
 JZ LongOK         ; execute prog if no errors
 LJMP ErrLoop
 LongOK: LJMP HexOK


ErrLoop: MOV A,#'?'   ; tell if errors
 ACALL PutChar
 ACALL GetChar        ; wait for escape
 SJMP ErrLoop

HexOK: MOV EFlags,#0  ; clear flags for retry
 ACALL GetChar        ; look for startup char
 CJNE A,#Slash,HexOK

 ACALL GetByte        ; get startup high address
 JB ErrFlag1,HexOK
 MOV HighAddr,DataByte

 ACALL GetByte        ; get startup low address
 JB ErrFlag1,HexOK
 MOV LowAddr,DataByte

 ACALL GetChar        ; look for <CR>
 CJNE A,#CR,HexOK

 MOV A,#'@'           ; send confirmation
 ACALL PutChar
 HexTI: JNB TI,HexTI  ; complete transmission
 PUSH LowAddr
 PUSH HighAddr
```



```
  RET                    ; execute downloaded prog


 HexIn: CLR A       ; HEX file input routine
  MOV State,A
  MOV Flags,A
  MOV HighAddr,A
  MOV LowAddr,A
  MOV HASave,A
  MOV LASave,A
  MOV ChkSum,A
  MOV FilChkHi,A
  MOV FilChkLo,A
  MOV Eflags,A
  SETB ErrFlag4     ; set 'no data' flag

 StateLoop: ACALL GetChar  ; get char

  ACALL AscHex      ; convert ASCII to hex
  MOV Ch,A
  MOV P1,Ch         ; display hex char
  ACALL GoState     ; find next state

  JNB DoneFlag,StateLoop   ; loop until finished

  ACALL PutChar     ; send checksum
  MOV A,#'('
  ACALL PutChar
  MOV A,FilChkHi
  ACALL PrByte
  MOV A,FilChkLo
  ACALL PrByte
  MOV A,#')'
  ACALL PutChar
  ACALL CRLF
  RET

 GoState: MOV A,State  ; execute state routine
  ANL A,#0Fh           ; within table range
  RL  A                ; adjust offset for jump
  MOV DPTR,#StateTable
  JMP @A+DPTR          ; go to current state

 ; HEX line format:
 ;     ':'  byte_count  AH  AL  record_type  data  checksum

 StateTable: AJMP StWait  ; 0 - wait for start
  AJMP StLeft    ; 1 - 1st nibble of count
  AJMP StGetCnt  ; 2 - get count
  AJMP StLeft    ; 3 - 1st nibble of address byte 1
  AJMP StGetAd1  ; 4 - get address byte 1
  AJMP StLeft    ; 5 - 1st nibble of address byte 2
  AJMP StGetAd2  ; 6 - get address byte 2
  AJMP StLeft    ; 7 - 1st nibble of record type
  AJMP StGetRec  ; 8 - get record type
  AJMP StLeft    ; 9 - 1st nibble of data byte
```



```
 AJMP StGetDat  ; 10 - get data byte
 AJMP StLeft    ; 11 - 1st nibble of checksum
 AJMP StGetChk  ; 12 - get checksum
 AJMP StSkip    ; 13 - skip data after error condition
 AJMP BadState  ; 14 - invalid state
 AJMP BadState  ; 15 - invalid state

StWait: MOV A,Ch          ; wait for HEX line start
 CJNE A,#StartChar,SWEX
 INC State
SWEX: RET

StLeft: MOV A,Ch          ; process 1st nibble of any byte
 JNB HexFlag,SLERR
 ANL A,#0Fh
 SWAP A
 MOV DataByte,A
 INC State
 RET

SLERR: SETB ErrFlag1      ; non-hex char found
 SETB DoneFlag
 RET

StRight: MOV A,Ch         ; process 2nd nibble of any byte
 JNB HexFlag,SRERR
 ANL A,#0Fh
 ORL A,DataByte
 MOV DataByte,A
 ADD A,ChkSum
 MOV ChkSum,A
 RET

SRERR: SETB ErrFlag1      ; non-hex char found
 SETB DoneFlag
 RET

StGetCnt: ACALL StRight  ; get data byte count for HEX line
 MOV A,DataByte
 MOV ByteCount,A
 INC State
 RET

StGetAd1: ACALL StRight  ; get upper address byte for HEX line
 MOV A,DataByte
 MOV HighAddr,A
 INC State
 RET

StGetAd2: ACALL STRight  ; get lower address byte for HEX line
 MOV A,DataByte
 MOV LowAddr,A
 INC State
 RET

StGetRec: ACALL StRight  ; get record type for HEX line
```

```
 MOV A,DataByte
 MOV RecType,A
 JZ SGRDat               ; jump if data record
 CJNE A,#1,SGRErr        ; check for end record
 SETB EndFlag
 SETB DatSkipFlag        ; ignore data in end record
 MOV State,#11
 SJMP SGREX
SGRDat: INC State
SGREX: RET

SGRErr: SETB ErrFlag2    ; invalid record type
 SETB DoneFlag
 RET

StGetDat: ACALL StRight  ; get data byte
 JB DatSkipFlag,SGD1     ; if no data skip flag

 ACALL Store             ; store byte in memory

 MOV A,DataByte          ; update file checksum
 ADD A,FilChkLo
 MOV FilChkLo,A
 CLR A
 ADDC A,FilChkHi
 MOV FilChkHi,A
 MOV A,DataByte
SGD1: DJNZ ByteCount,SGDEX   ; proof if last data byte
 INC State
 SJMP SGDEX2

SGDEX: DEC State         ; setup state for next data byte
SGDEX2: RET

StGetChk: ACALL StRight  ; get checksum
 JNB EndFlag,SGC1        ; check for end record
 SETB DoneFlag
 SJMP SGCEX

SGC1: MOV A,ChkSum       ; getc calculated checksum
 JNZ SGCErr
 MOV ChkSum,#0
 MOV State,#0            ; HEX line done
 MOV LASave,LowAddr      ; save address for later check
 MOV HASave,HighAddr
SGCEX: RET

SGCErr: SETB ErrFlag3    ; line checksum error
 SETB DoneFlag
 RET

StSkip: RET     ; skip any additional data sent in HEX line

BadState: MOV State,#Skip   ; invalid state, should never happen
 RET
```

```
Store: MOV DPH,HighAddr  ; save data byte to prog- dsRAM
 MOV DPL,LowAddr
 MOV A,DataByte
 MOVX @DPTR,A            ; store data byte

 CLR ErrFlag4            ; data found in HEX file
 INC DPTR
 MOV HighAddr, DPH       ; save next address
 MOV LowAddr, DPL
 CLR A
 CJNE A,HighAddr,StoreEx ; check if address overflow
 CJNE A,LowAddr,StoreEx  ; where both bytes are 0
 SETB ErrFlag5           ; set address overflow flag
StoreEx: RET

StoreErr: SETB ErrFlag6  ; data storage verify error
 SETB DoneFlag
 RET


SerStart: MOV A,PCON     ; set up serial port to 4k8 baud
 SETB ACC.7
 MOV PCON,A
 MOV TH1,#0EFh
 MOV TL1,#0EFh
 MOV TMOD,#20h
 MOV TCON,#40h
 MOV SCON,#52h
 RET

GetByte: ACALL GetChar   ; get a hex byte from serial port
 ACALL AscHex
 MOV Ch,A
 ACALL StLeft            ; 1st nibble
 ACALL GetChar
 ACALL AscHex
 MOV Ch,A
 ACALL StRight           ; 2nd nibble
 RET

GetChar: JNB RI,GetChar  ; get a char from the serial port
 CLR RI
 MOV A,SBUF
 CJNE A,#ESC,GCEX
 LJMP Start
GCEX: RET

PutChar: JNB TI,PutChar  ; output a char to serial port
 CLR TI
 MOV SBUF,A
 RET
                         ; convert char from ASCII to hex
AscHex: CJNE A,#'0',AH1  ; 1st check for ASCII numbers
AH1: JC AHBad            ; char less than '0'
 CJNE A,#'9'+1,AH2
AH2: JC AHVal09          ; char between '0' and '9'
```

```
 CJNE A,#'A',AH3         ; 2nd check for ASCII upper case letters
AH3: JC AHBad            ; char less than 'A'
 CJNE A,#'F'+1,AH4
AH4: JC AHValAF          ; char between 'A' and 'F'

 CJNE A,#'a',AH5         ; 3rd check for ASCII lower case letters
AH5: JC AHBad            ; char less than 'a'
 CJNE A,#'f'+1,AH6
AH6: JNC AHBad           ; char between 'a' and 'f'
 CLR C
 SUBB A,#27h             ; pre-adjust char to get a value, ASCII letter
 SJMP AHVal09

AHBad: CLR HexFlag       ; char is non-hex, set error flag
 SJMP AHEX
AHValAF: CLR C
 SUBB A,#7               ; pre-adjust char to get a value, ASCII number
AHVal09: CLR C
 SUBB A,#'0'             ; adjust char to get a hex value
 SETB HexFlag            ; flag char as valid
AHEX: RET


HexAsc: ANL A,#0Fh       ; convert hex nibble to ASCII char

 CJNE A,#0Ah,HA1         ; check value range
HA1: JC HAVal09          ; value is 0 to 9
 ADD A,#7                ; value is A to F, pre-adjust char
HAVal09: ADD A,#'0'      ; adjust value to ASCII char
 RET

ErrPrt: MOV A,#':'       ; tell error flags to host
 CALL PutChar            ; 1st send prompt
 MOV A,Eflags
 JZ ErrPrtEx             ; send error flags if an error occured
 CALL PrByte
ErrPrtEx: RET

CRLF: MOV A,#CR          ; output a <CRLF> to serial port
 CALL PutChar
 MOV A,#LF
 CALL PutChar
 RET

PrByte: PUSH ACC         ; output a byte to serial port
 SWAP A
 CALL HexAsc             ; get upper nibble
 CALL PutChar
 POP ACC
 CALL HexAsc             ; get lower nibble
 CALL PutChar
 RET

END
```

Host Bootstrap Program

The following C code is used to download an Intel Hex file over the serial port to the MC8051.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <dos.h>

#define PORT 0      // port which should be used
#define SPACE 200   // space in ms to wait between chars

void init();                      // search for ports
void initBaud(int COM, unsigned char b);  // init serial port
int readReady(int COM);           // proof if port is ready for read
int writeReady(int COM);          // proof if port is ready for write
unsigned char serWrite(unsigned char b,int COM); // send byte
unsigned char serRead(int COM); // receive byte

int PortAnz=0;                    // number of ports detected
unsigned int Ports[4];            // addresses of detected ports

int main()
{
  unsigned char read=0xFF;        // return value after read
  unsigned char write=0xFF;       // return value after write
  char hexfile[30]="d:\\file.hex";  // source Intel Hex file
  int prompt=0;                   // prompt detected flag
  int c;                          // char read from source
  unsigned char d;                // char downloaded via serial port
  FILE *hex;

  init();    // search for all ports

  initBaud(PORT, 211);         // init port: AL=11010011b
                               //            4800 Baud, 8-N-1
  printf("\n");

  while (!kbhit() && (prompt == 0))   // wait for prompt or
  {                                   // abort by user
     while(!readReady(PORT));
     read=serRead(PORT);
     if (read != 0xFF)
     {
       printf("%c", read);     // display all chars get from uC
       if (read == 0x3D)
      prompt = 1;              // prompt '=' detected
     }
  }

  printf("\n");
```

```
hex = fopen(hexfile, "r");      // open source file
if ((hex)  && (prompt == 1))    // begin download
{
  do                        // loop until end of file
  {
   delay(SPACE);            // apply space between chars

   c = fgetc(hex);          // get next char from source
   if (c != EOF)
   {
    if (c == 10)            // display line by line
    printf("\n");
    else
    putch(c);

    d = (unsigned char) c;        // convert for download

    while (!writeReady(PORT));  // wait until port is ready

    write=serWrite(d, PORT);    // write char to port
    if (write == 0xFF)
    {
     fprintf(stderr, "error: cannot write to SER%d.\n", PORT);
     break;
    }
   }
  } while (c != EOF);
}
else
 fprintf(stderr, "error: cannot open HexFile, no prompt.\n");

fclose(hex);       // close source file
delay(SPACE);

while (!kbhit())              // file downloaded -> set startup
{
  while(!readReady(PORT));   // get status flag from uC
  read=serRead(PORT);
  if (read != 0xFF)
  {
    int i;       // loop counter

    printf("%c", read);
    if (read == ':')
    { printf("\nsend startup address (2000h)\n");

      for(i=0; i < 6; i+=1)         // send '/2000<CR>'
      {
        while (!writeReady(PORT));
        switch (i)
        {
          case 0: { write=serWrite('/', PORT); break; }
          case 1: { write=serWrite('2', PORT); break; }
          case 2: { write=serWrite('0', PORT); break; }
          case 3: { write=serWrite('0', PORT); break; }
```

```
          case 4: { write=serWrite('0', PORT); break; }
          case 5: { write=serWrite( 13, PORT); break; }
           }
        delay(SPACE);
      }
    }
  }
 }
 return 0;
}


void init()                      // search for ports
{
  unsigned int far* ptr;         // points to BIOS address
  int i;                         // port offset

  ptr=(unsigned int far *)MK_FP(0x0040,0);  // address 40h, offset 0

  for (i=0; i<=4; i++)           // proof up to four ports
  {
    printf("\nport #%i: ", i);   // display port number
    printf("%X", *(ptr+i));      // and address
    Ports[i]=*(ptr+i);           // store port address
    if(Ports[i]==0)
      break;                     // no more ports
    else
      PortAnz++;
  }
  printf("\nnumber of ports: %i\n", PortAnz);

  return;
}

void initBaud(int COM, unsigned char b)    // init serial port
{
  asm{
    mov AH,0x0                   // AH = 0 -> data ready
    mov DX,COM                   // DX -> number of port
    mov AL,b                     // AL -> port configuration
    int 0x14                     // init port
  }
  return;
}

int readReady(int COM)  // proof if port is ready for read
{                       // -> look at status: port address + 5, LSB
  unsigned char b;
  b=inportb(Ports[COM]+5);   // get status byte
  return (b&1);              // return status bit
}

int writeReady(int COM) // proof if port is ready for write
{                       // -> look at status: port address + 5, bit 6
  unsigned char b;
  b=inportb(Ports[COM]+5);   // get status byte
```

```
  return (b&64);              // return masked status bit
}

unsigned char serRead(int COM)  // receive byte
{
  unsigned char c;

  if(readReady(COM))          // proof if ready for read
  {
   c=inportb(Ports[COM]);     // get byte from port
   return c;
  }
  return 0xff;
}

unsigned char serWrite(unsigned char b,int COM)  // send byte
{
  if(writeReady(COM))         // proof if ready for write
  {
   outportb(Ports[COM],b);    // write to port
   return 0;
  }
  return 0xff;
}
```