Interfacing IPSiLog serial SRAM to a R8C/13 Controller using GPIO-Pins

OVERVIEW

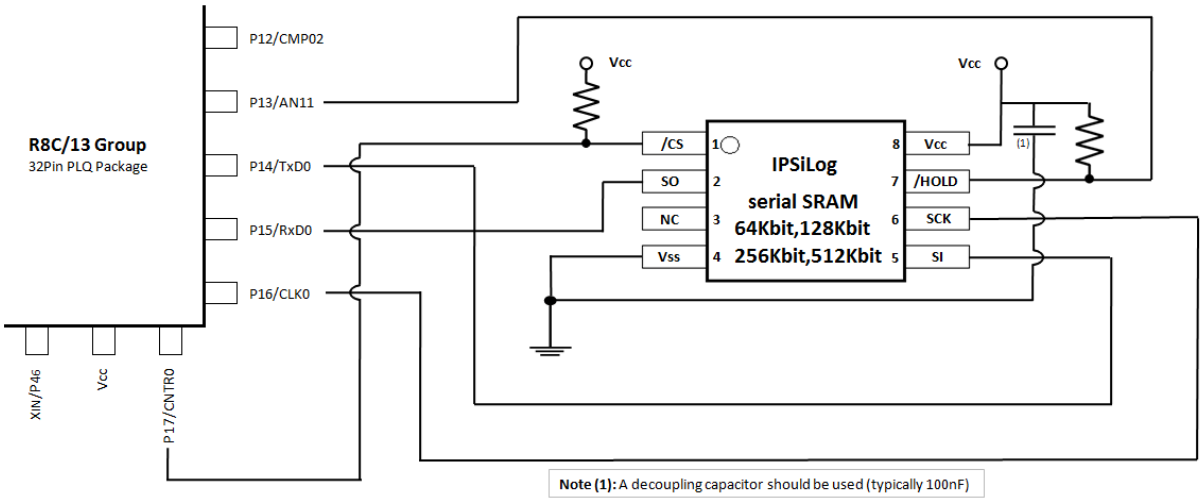
The serial SRAM family of IPSiLog products offers the customer serial high speed, low power, standalone SRAM memory at writing speeds up to 20MHz at various densities starting at 64Kbit. They are an alternative choice to the traditional parallel architecture that saves both board area and also I/O count on the MCU. They are the ideal solution for data stream applications.
These devices employ an industry standard 4-wire synchronous SPI interface that can easily be connected to a microcontroller with SPI-Interface or using GPIO pins. A clock SCK is used to latch data into and out of the device, and a chip select pin is used to enable/disable the device.

The IPSiLog serial SRAM is supporting two SPI-Modes: SPI-Mode 0 (CPOL:0, CPHA:0) and SPI-Mode 3 (CPOL:1, CPHA:1). In both Modes data are clocked into the device on the rising SCK edge and clocked out on the falling edge of SCK. The difference between Modes 0 and 3 is simply whether SCK starts low or high when /CS is asserted low.

A dedicated SPI port on a microcontroller makes the choice an easy one in those cases. However, many microcontrollers do not have a dedicated SPI port, so the use of bit-banging provides a means to use general purpose I/O pins. This method involves software that controls the I/O port.

This application note demonstrates, how to connect the IPSiLog serial SRAM to a R8C/13 Controller from RENESAS. The application was created and tested by the RENESAS HEW and E8 Debugger, using "Renesas M16C Standard Toolchain 5.43.00".

Hardware connection



Application software

The software demonstrates:

1. How to read/write to Status Register (BYTE-Mode)
2. How to read the Memory-Size Register (BYTE-Mode)
3. How to read/write single Memory-Address in BYTE-Mode
4. How to read/write the Memory in PAGE-Mode
5. How to read/write the Memory in PSEQ-Mode
6. How to read/write the Memory in VRTM-Mode
7. How to "split" the Memory in two functional blocks using VRTM-Mode

The fundamental function for "bit-banging" the IO-Pins is:

```
write_cmd(unsigned char outgoing_byte);
```

With this function all other functions to access the IPSiLog serial SRAM are built.

The following functions were created to access the serial SRAM:

```
void register_write(unsigned char address, unsigned char data);
unsigned char register_read(unsigned char register_name);
void byte_write(unsigned int address, unsigned char data);
unsigned char byte_read(unsigned int address);
void get_memsize(void);
void stream_read(unsigned int address, unsigned int length);
void stream_write_fix(unsigned int address, unsigned int length,int value);
```

The source code of main program:

```
/*****************************************************************************/
/*                                                                           */
/*  FILE        :IPSiLog_SRAM_R8C_GPIO.c                                     */
/*  DATE        :Aug. 26, 2010                                               */
/*  DESCRIPTION :main program file.                                          */
/*  CPU GROUP   :R8C/13                                                      */
/*                                                                           */
/*  Heinz Gruber / IPSiLog Semiconductor GmbH                                */
/*                                                                           */
/*  This application example demonstrates the fundamental function           */
/*  of IPSiLog serial SRAM devices of IP12xxxx family connected to           */
/*  a Renesas R8C/13 microcontroller using GPIO-Pins and bit banging.        */
/*  It can be easily converted to other Controller of R8C/M16C family.       */
/*                                                                           */
/*  This application example is for demonstration purpose only.              */
/*  Copyright 2010 IPSiLog Semiconductor GmbH                               */
/*****************************************************************************/

//****************************************************************************
// User Includes & defines
//****************************************************************************
#include "sfr_r813.h"                   // a structure to access all of the device registers
#define enable_interrupt asm("FSET I");                   // all Interrupts ON
#define disable_interrupt asm("FCLR I");                  // all Interrupts OFF
#define HOLD 3      // P1_3 HOLD
#define SDI  4      // P1_4 SDI
#define SDO  5      // P1_5 SDO
#define SCK  6      // P1_6 SPI
#define CS   7  // P1_7 SPI
#define HI(x) p1 |=(1<<(x))
#define LO(x) p1 &=~(1<<(x))
#define read_SR register_read(SRAM_RDSR)
#define write_SR(x) register_write(SRAM_WRSR,x)
#define read_RDMI register_read(SRAM_RDMI)

//****************************************************************************
// Commands that can be sent to the SRAM
//****************************************************************************
#define SRAM_RD   0x03       // Read data memory
#define SRAM_WR   0x02       // Write data memory
#define SRAM_RDSR 0x05                  // Read status register
#define SRAM_WRSR 0x01                  // Write status register
#define SRAM_RDMI 0x0E                  // Read MemorySize & ID Code

//****************************************************************************
// Configuration options that can be written to the status register
//****************************************************************************
#define MODE_BYTE    0x00    // BYTE mode (default)
#define MODE_PAGE    0x80    // Page mode
#define MODE_VRTM    0x40    // Virtual Chip Mode
#define MODE_PSEQ    0xC0    // Aligned Sequential Mode
#define HOLD_enable  0x00    // Hold Function enable
#define HOLD_disable 0x01    // Hold Function disable

//****************************************************************************
// Other SRAM defines
//****************************************************************************
//#define SPI_MODE 0                    // IPSiLog SRAM can handle MODE 0 (CPOL:0, CPHA:0)
#define SPI_MODE 3                      // and MODE 3 (CPOL:1, CPHA:1)

//****************************************************************************
// Declaration of function prototypes
//****************************************************************************
void main_clock_high_speed(void);
void port_init(void);
void init_all(void);
unsigned char write_cmd(unsigned char outgoing_byte);
void register_write(unsigned char address, unsigned char data);
unsigned char register_read(unsigned char register_name);
void byte_write(unsigned int address, unsigned char data);
unsigned char byte_read(unsigned int address);
void get_memsize(void);
void clear_output_array(unsigned int size);
void stream_read(unsigned int address, unsigned int length);
void stream_write_fix(unsigned int address, unsigned int length,int value);

//****************************************************************************
// Definition of variables
//****************************************************************************
unsigned char result, output_data[256];
unsigned long address,start_address,my_address,max_address,data,i,length;
unsigned int data2,value,size;

void main(void)
{
        init_all();

//-------------- example for BYTE operations on registers ----------------------------

        write_SR(MODE_BYTE + HOLD_disable);             //BYTE-Mode + HOLD off
        result=read_SR;                                         // Read Status register
        get_memsize();              // Read Memsize register to variable max_address

//-----------------------------------------------------------------------------------

//-------------- example for BYTE operations on SRAM Memory -------------------------
```

```
          write_SR(MODE_BYTE + HOLD_disable);    // BYTE-Mode + HOLD off
          address=0x00ff;                                // Define the address
          value=0x11;                                    // Define the value
          byte_write(address,value);             // Write value to address
          result=byte_read(address);             // Read back value

//-------------------------------------------------------------------------------

//-------------- example for PAGE operations in SRAM Memory --------------------------
//-------------- you can select any start address/page -------------------------------
//-------------- in PAGE-Mode the device loops within the PAGE-border-----------------

          write_SR(MODE_PAGE + HOLD_disable);    // PAGE-Mode + HOLD off
          start_address=0x0000;                          // Start e.g. 1st page, address "0"
          length=32;                                     // 32byte = 1page
          value = 0x11;                                  // example: "00010001"
          stream_write_fix(start_address,length,value);    // fill page with value
          clear_output_array(256);               // clear the output_data[256] array
          stream_read(start_address, length);    // read page-content into output_data[]

//-------------------------------------------------------------------------------
//-------------- example for PSEQ operations in SRAM Memory -------------------------
//-------------- PSEQ-Mode is always automatically aligned to the low page-border -----
//-------------- The PSEQ-Mode loops over the hole Memory-Array ----------------------

          write_SR(MODE_PSEQ + HOLD_disable);    // PSEQ-Mode + HOLD off
          start_address=0x0000;                          // Start e.g. address "0"
          length=max_address;                    // full memory-size
          value = 0xAA;                                          // e.g. Checker bord "10101010
          stream_write_fix(start_address,length,value);    // fill total memory value
          clear_output_array(256);                       // clear the output_data[256] array
          stream_read(start_address, 256);               // read 256byte into output_data[]

//-------------------------------------------------------------------------------
//-------------- example for VRTM operations in SRAM Memory -------------------------
//-------------- you can select any start address/page -------------------------------
//- The VRTM-Mode the device loops between start address and upper Memory-Border ------
//- Data in Memory lower than the start address are not overwritten in VRTM-Mode ------

          write_SR(MODE_VRTM + HOLD_disable);            // VRTM-Mode + HOLD off
          start_address=0x0000;                                  // Start e.g. address "0"
          length=max_address;                            // full memory-size
          value = 0xAA;                                    // Checker bord
          stream_write_fix(0x0fff,max_address,value);      // fill memory staring 0x0fff with checker bord
          clear_output_array(256);                       // clear the output_data[256] array
          stream_read(start_address, 256);               // read 256byte into output_data[]

//-------------------------------------------------------------------------------
//- How to split the SRAM into normal SRAM-Array and Ring-Buffer array by VRTM-Mode ---
//-------------- you can select any start address/page -------------------------------
//- The VRTM-Mode the device loops between start address and upper Memory-Border ------
//- Data in Memory lower than the start address are not overwritten in VRTM-Mode ------

          write_SR(MODE_PSEQ + HOLD_disable);        // PSEQ-Mode + HOLD off
          my_address=max_address/2;                  // e.g. half of Memory-size
          value = 0x00;                                    // example: "0"
          stream_write_fix(0,my_address,value);      // Fill lower memory to my_address with value
          write_SR(MODE_VRTM + HOLD_disable);        // VRTM-Mode + HOLD off
          value = 0xAA;                                    // example: Checker bord
          stream_write_fix(my_address+1,max_address,value); // fill memory staring my_address+1 with value
          clear_output_array(256);                       // clear the output_data[256] array
          stream_read(my_address-128, 256);              // read 256byte into output_data[]

//-------------------------------------------------------------------------------

}         // main loop end

/*+++++++++++++++++++++++++++++++++ Functions +++++++++++++++++++++++++++++++++++++++/
/*************************************************************************************
Name: init_all
Parameters: None
Returns: None
Description: Booting up
*************************************************************************************/
void init_all(void){
          disable_interrupt;                     // Interrupts off
          main_clock_high_speed();               // Main clock on
          enable_interrupt;                      // Interrupts on
          port_init();                                   // Init the ports
}

/*************************************************************************************
Name: main_clock_high_speed
Parameters: None
Returns: None
Description: Change for on chip oscillator to main clock, no division
*************************************************************************************/
void main_clock_high_speed(void)
{
          prc0=1;                                // Protect off
          cm13=1;                                // Xin Xout pin using
          cm15=1;                                // XCIN-XCOUT drive capacity select bit : HIGH
          cm05=0;                                // Xin on (main clock stop = off)
          cm16=0;                                // Main clock = No devision mode
          cm17 = 0;                    // Main clock = No devision mode
          cm06 = 0;                    // CM16 and CM17 enable
          asm("nop");                            // Wait for stable oscillation
          asm("nop");
          asm("nop");
          asm("nop");
```

```
          ocd2 = 0;                    // Select main clock
          prc0 = 0;                    // Protect on
}

/***********************************************************************************
Name: port_init
Parameters: None
Returns: None
Description: Initial setting of Port
***********************************************************************************/
void port_init(void){
          pd1 = (1<<CS)|(1<<SDI)|(1<<SCK)|(1<<HOLD);       // Port1 Bits output
          HI(HOLD);
          HI(CS);
          HI(SDI);
          if (SPI_MODE==0) LO(SCK);                        // SPI-Mode: 0
          else HI(SCK);                                              // SPI-Mode: 3
}

/***********************************************************************************
Name: register_write
Parameters: register_name, data
Returns: None
Description: Write a 8bit value to a register
***********************************************************************************/
void register_write(unsigned char address, unsigned char data)
{
   unsigned char in_byte;
    LO(CS);                                                 //Select SRAM
    in_byte = write_cmd(address);               //SPI write instruction to register
    in_byte = write_cmd(data);                              //SPI write 8-bit data to register
    HI(CS);                                                 //Deselect SRAM
}

/***********************************************************************************
Name: register_read
Parameters: register_name
Returns: Register value 8bit
Description: Read a 8bit value from sram register
***********************************************************************************/
unsigned char register_read(unsigned char register_name)
{
    char in_byte;
    LO(CS);                                                 //Select SRAM
          in_byte = write_cmd(register_name);    //SPI read 8-bit register from SRAM
          in_byte = write_cmd(0x00);             //Send nothing, get back the register value
    HI(CS);                                                 //Deselect SRAM
    return(in_byte);                                        //Return value
}

/***********************************************************************************
Name: byte_write
Parameters: address(16bit), data(8bit)
Returns: None
Description: Write a 8bit value to a memory address
***********************************************************************************/
void byte_write(unsigned int address, unsigned char data)
{
   unsigned char in_byte;
    LO(CS);                                                 //Select SRAM
    in_byte = write_cmd(SRAM_WR);               //SPI write instruction to memory
    in_byte = write_cmd((address >> 8) & 0xff); //SPI send upper 8bit of address
    in_byte = write_cmd(address & 0xff );       //SPI send lower 8bit of address
    in_byte = write_cmd(data);                              //SPI write 8-bit data to memory at address
    HI(CS);                                                 //Deselect SRAM
}

/***********************************************************************************
Name: byte_read
Parameters: address
Returns: Memory value 8bit
Description: Read a 8bit value from sram memory
***********************************************************************************/
unsigned char byte_read(unsigned int address)
{
    char in_byte;
    LO(CS);                                                 //Select SRAM
    in_byte = write_cmd(SRAM_RD);               //SPI read instruction to memory
    in_byte = write_cmd((address >> 8) & 0xff); //SPI send upper 8bit of address
    in_byte = write_cmd(address & 0xff );       //SPI send lower 8bit of address
    in_byte = write_cmd(0x00);                              //Send nothing, get back the memory value
    HI(CS);                                                 //Deselect SRAM
    return(in_byte);                                        //Return value
}

/***********************************************************************************
Name: get_memsize
Parameters: None
Returns: None
Description: Get Memory size from sizeID register and set max_address
***********************************************************************************/
void get_memsize(void){
                    data2=read_RDMI;            // read out sizeID register
                    data2= data2 & 0x0f;                    // only lower 4 bits
                    switch(data2) {
                              case 0:   max_address=0x1fff; // 64kbit
                              break;
                              case 1:   max_address=0x3fff; // 128kbit
                              break;
                              case 2:   max_address=0x7fff; // 256kbit
                              break;
```

```
                            case 3:   max_address=0xffff; // 512kbit
                            break;
                    }
}

/*****************************************************************************
Name: write_cmd
Parameters: cmd
Returns: recv
Description: Write a 8bitcommand to SPI latched into at rising edge
*****************************************************************************/
unsigned char write_cmd(unsigned char cmd){
          register unsigned char i;
          register unsigned char recv;
          recv = 0;
          LO(CS);
          for(i=0;i<8;i++) {
                    LO(SCK);
                    if(cmd&0x80) HI(SDI); else LO(SDI);
                    HI(SCK);
                    recv<<=1;
                    if(p1&(1<<SDO)) {
                              recv|=0x01;
                    }
                    if (SPI_MODE==0) LO(SCK);
                    cmd<<=1;
          }
          return recv;
}

/*****************************************************************************
Name: stream_write_fix
Parameters: address, lenght, value
Returns: nothing
Description: Write memory in PSEQ-Mode with fixed value
*****************************************************************************/
void stream_write_fix(unsigned int address, unsigned int length,int value)
{
    char in_byte;
    LO(CS);                                             //Select SRAM
    in_byte = write_cmd(SRAM_WR);                 //SPI write instruction to memory
    in_byte = write_cmd((address >> 8) & 0xff); //SPI send upper 8bit of address
          in_byte = write_cmd(address & 0xff );  //SPI send lower 8bit of address
                    for (i=0;i<=length;i++){
          in_byte = write_cmd(value);
          }
    HI(CS);                                             //Deselect SRAM
}

/*****************************************************************************
Name: clear_output_array
Parameters: None
Returns: None
Description: Fills an output_array with sequential data e.g. "0"
*****************************************************************************/
void clear_output_array(unsigned int size){
          for(i=0;i<size;i++) output_data[i]=0;
}

/*****************************************************************************
Name: stream_read
Parameters: address
Returns: Read values in Array output_data
Description: Read length-byte value from sram
*****************************************************************************/
void stream_read(unsigned int address, unsigned int length)
{
    char in_byte;
    LO(CS);                                             //Select SRAM
    in_byte = write_cmd(SRAM_RD);                 //SPI read instruction to memory
    in_byte = write_cmd((address >> 8) & 0xff); //SPI send upper 8bit of address
          in_byte = write_cmd(address & 0xff );  //SPI send lower 8bit of address
          for (i=0;i<length;i++){
          output_data[i] = write_cmd(0x00);      //Send nothing, get back the memory value
          }
    HI(CS);                                             //Deselect SRAM
}
```

The source-program and related header-files can be downloaded from IPSiLog Homepage.
www.ipsilog.com
The zip-File name is IPSiLog_SRAM_R8C_GPIO.zip