Using Secure RAM Access

Introduction

Operation conditions can be a challenge for data transfer. At worst case data change their value while being transferred. This can affect address information as well as data to be transferred. The resulting risks would be:
1. write/read corrupt data to a correct address
2. write/read correct data to/from a wrong address
In both cases the initiator of the data transfer would not be informed that the data transfer has been corrupt.
Writing to a memory with corrupt data will result in writing correct data to the wrong address or in writing wrong data to a correct address. This is not only painful because data are corrupt or lost, since the user will not even get a notice of this corrupt memory-access, application may crash or on increased risk for tempering

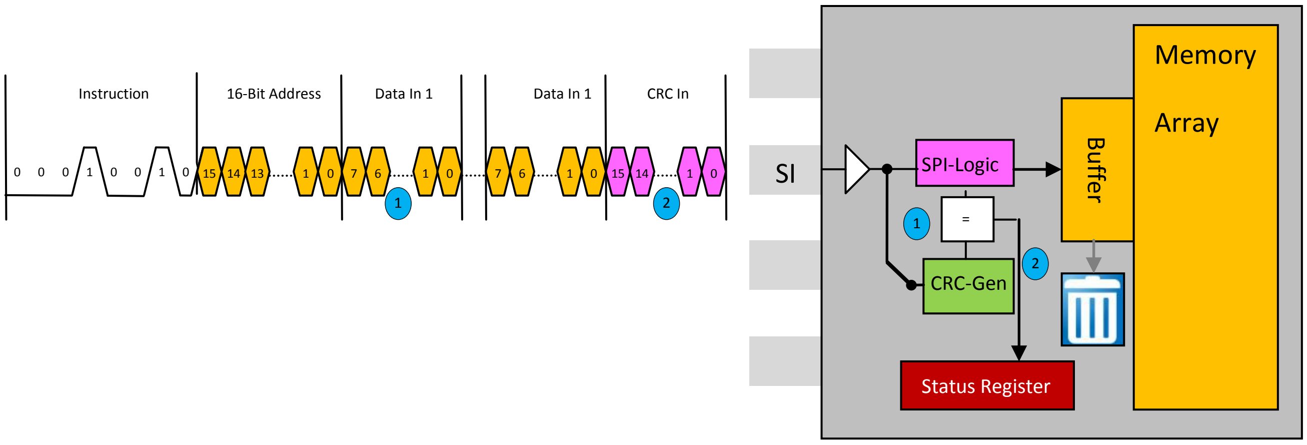
In order to grant data integrity even in harsh environments Anvo-Systems Dresden introduced the option for secure RAM access. The idea is to calculate a checksum from all transferred address and data bits, append this checksum to the transferred data and compare the transferred checksum with the locally calculated checksum. If the comparison fails the transferred data will be refused.

Transferred Data			Check CRC	Resulting Action	
Address	Data	CRC		Secure READ	Secure WRITE
o.k.	o.k.	o.k.	o.k.	accept data	write data to array
fail	don't care	don't care	fail	discard data, read again	discard data, set error flag
don't care	fail	don't care	fail	discard data, read again	discard data, set error flag
don't care	don't care	fail	fail	discard data, read again	discard data, set error flag

SECURE WRITE

Secure RAM access (read as well as write) is always executed as burst of 2 byte address followed by 64 or 32 byte data and 2 byte checksum. Page roll over is defined for Secure Read and Secure Write.

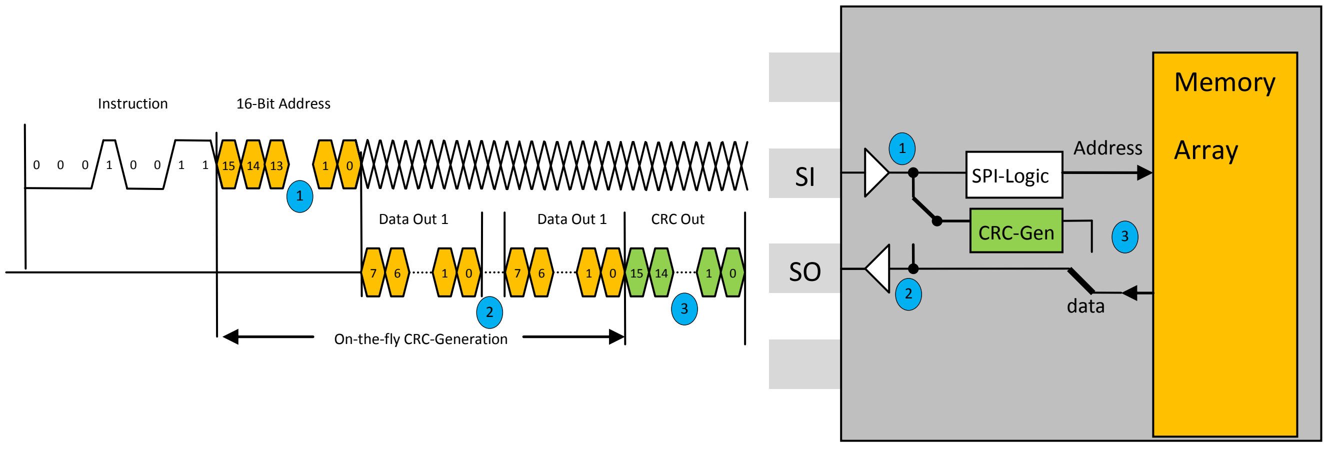
Executing a SECURE WRITE operation, the incoming address and data are written to a temporary buffer.
Since the nvSRAM calculates the checksum from this data in parallel, no time is lost and as soon as the pre-calculated checksum is received, both checksum can be compared. Only if checksum comparison is pass, data are written to the array. If the comparison fails, no data are written to the array and the Secure WRITE Monitoring flag in the status register is set.



In Order to get informed about the success of a secure WRITE, the transmitting side need to read the status register and check the /SWM flag.

SECURE READ

Executing a SECURE READ the nvSRAM will calculate the checksum from the address received (1) and the data sent (2). The checksum will be sent immediately after sending the last data.
The receiving unit need to buffer the data, calculate the checksum and compare the own checksum with the checksum received.



Page roll over is defined for Secure Read.

The Checksum Algorithm

For checksum calculation the CRC16-CCITT polynomial x16+x12+x5+1 is used. The initial value is 0xFFFF.

Note: For CRC-calculation only valid address bits are used (13bit for 8kx8, 15bit for 32kx8)

Example Strings (Starting Address = blue, crc = red)

8kx8 nvSRAM: (page buffer 32 byte)
0100202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f**0c0d**

32kx8 nvSRAM:
5500BBDD02030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B
40002E2F30313233343536EE38393A3B3C3D3E3F**927A**

64kx8 nvSRAM:
5500BBDD02030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B
40002E2F30313233343536EE38393A3B3C3D3E3F**2D17**

In order to finish any secure memory access operation, the on-the-fly calculated checksum will be compared with the pre-calculated checksum.

It is possible to apply secure read to data written with standard write as well as with secure write.

Checksum Calculation Code Example 2(8Kx8-nvSRAM)

```
// checksum calculation for 13bit address and 32 byte data
// buffer[0] = address_h
// buffer[1] = address_l
// buffer[2:65] = data

int i = 0;
int j = 0;
unsigned int temp = 0;
unsigned int crc = 0xffff;         // need to be initiated to 0xffff!

// *** address_H has 5 significant address bits only. Calculate separate! *****

crc = crc ^ (write_to_ram[0] << 11);   // shift address_h to msb of crc
for (j=0; j<5; j++)  // 8 bits/byte
{
  if ((crc & 0x8000) == 0x8000 ) // if x^16 = 1
    { crc = (crc << 1) ^ 0x1021;} // x^12+x^5+1
  else
    {crc = (crc << 1);}
  crc = crc & 0xffff; // trim shouldn't be needed
}
// *** now calculate crc for the remaining bytes *****
for (i=1; i<34; i++) // Second address byte and all data
{
  crc = crc ^ (write_to_ram[i] << 8);
  for (j=0; j<8; j++)  // 8 bits/byte
    {
    if ((crc & 0x8000) == 0x8000 ) // if x^16 = 1
      { crc = (crc << 1) ^ 0x1021;} // x^12+x^5+1
    else
      {crc = (crc << 1);}
    crc = crc & 0xffff; // trim shouldn't be needed
                    }
// done!
```

Checksum Calculation Code Example 1(32Kx8-nvSRAM)

```
// checksum calculation for 15bit address and 64 byte data
// buffer[1:0] = address
// buffer[2:65] = data

int i = 0;
int j = 0;
unsigned int temp = 0;
unsigned int crc = 0xffff;        // need to be initiated to 0xffff!

// *** address_H has 7 address bits only. Calculate separate! *****
crc = crc ^ (buffer[0] << 9);
for (j=0; j<7; j++) // 7 bits only
{
  if ((crc & 0x8000) == 0x8000 ) // if x^16 = 1
  {
    crc = (crc << 1) ^ 0x1021; // x^12+x^5+1
  }
  else
  {
    crc = (crc << 1);
  }
}
// calculate remaining bytes
for (i=1; i<66; i++) // starting address and all data bytes
{
  crc = crc ^ (buffer[i] << 8);
  for (j=0; j<8; j++) // 8 bits/byte
  {
    if ((crc & 0x8000) == 0x8000 ) // if x^16 = 1
    {
      crc = (crc << 1) ^ 0x1021; // x^12+x^5+1
    }
    else
    {
      crc = (crc << 1);}
      crc = crc & 0xffff; // trim shouldn't be needed
    }
  }
// done!
```

References:

http://www.lammertbies.nl/comm/info/crc-calculation.html

http://en.wikipedia.org/wiki/Cyclic_redundancy_check

http://en.wikipedia.org/wiki/Computation_of_CRC

Version History

Date	Version	Changes
4-Jan 2012	1.0	Initial version
11-Jun-12	1.1	Added comments to source code checksum calculation
05-Feb-13	1.2	Added checksum calculation for 64Kb-devices

Author: Jan Kabus

Email: applications@anvo-systems-dresden.com
Web: www.anvo-systems-dresden.com