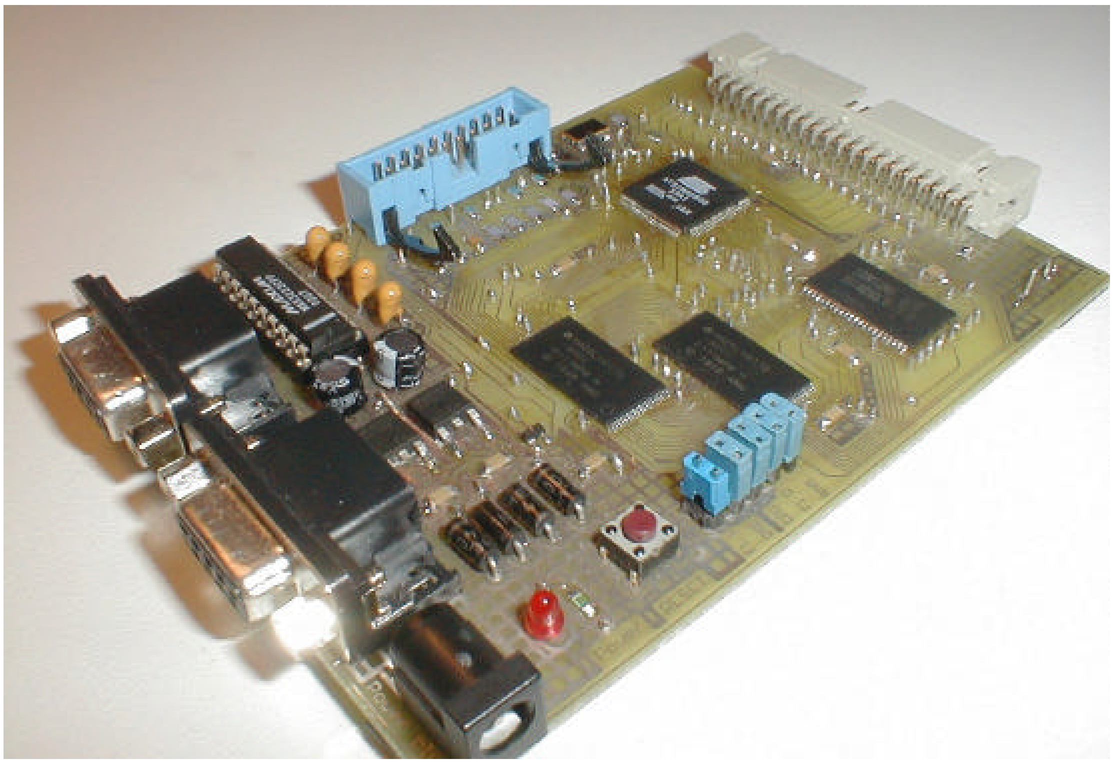
Test of Onboard Computer For DTUSat



Ørsted • DTU

Technical University of Denmark

Malte Breiting, c973568
Jonas Sølvhøj, c973442

Table of Contents

Introduction

This report covers the work done by the Onboard Computer group during the last 3 weeks of January 2002. The main objective of this work was to examine how the semiconductors selected for the Onboard Computer react when they are irradiated.
This report will describe in greater detail what happened to the parts, and the software used during tests

The work done during the three weeks

PCBs for the tests had to be produced. Some small errors of our current schematic were eliminated and a new PCB was routed by hand in order to make it smaller than our first test board. We then produced the PCB's and did the soldering of all components (and vias) with exception of the processor and the flash. The routing was quite time consuming (about a day), and equal amount of time was used during the manufacturing and soldering of the boards.
Two days was used to rewrite some old test software, as well as to write some new.
Most part of the time was used to solder the irradiated processors and the flashs to the boards – and to remove them again. Actually the soldering didn't take much time, but the troubleshooting (e.g. finding bad soldering, short circuits) did. We had to be very careful in this process, at we did not know if the reason the processor didn't work was bad soldering – or malfunction as a function of radiation. Most time of the three weeks were used in this step.

In addition to this we also did the following:

- *Interface meeting*
  We had a meeting with the rest of the hardware groups of the satellite project with the purpose of specifying the electrical interfaces between all hardware. This report will not cover this, as it has already been documented, and is available at the DTUsat web site in the System Engineering group. The preparation of the meeting and the meeting itself took a day.
- *Preliminary Design Review*
  We met with all groups of the project in order to find out if the current design were able to cover all aspects of the satellite we are building, if all subsystems can be built together to form one functional satellite, as well as to discover if critical parts of the design were missing. The time used for preparation and the meeting was about 1½day.

Radiation in Space

To ensure that the computer will work in space it is important to test the components durability when they are exposed to long-term radiation. This is an important ting to know, as the radiation in space may shorten the lifetime of the semiconductors or make them malfunction. In order to examine what happens with the semiconductors, a number of both the processors (AT91M40800) and the flashs (AM29LV017D) were sent to Risø, where they were irradiated with doses of 1, 2, 5, and 10 kRad. Back on our desk they were then soldered onto a PCB, and tests concerning functionality (e.g.

if we were able to write to, and read from the flash), and measurements on power consumption were carried out.[1] Besides the need not to exceed the limits of the power budget it is also important to know the power dissipation in order to set a suitable threshold for the latch-up protection (a latch-up is detected when the component draws a relatively large current compared to normal operation).

The mProcessor

Each of the processors that had been exposed to radiation was mounted on one of our 3 PCB's after which the functionality of the processors was verified. Then power dissipation was measured. The measurements were done in various situations to test the power consumption of all the built-in processors functions.

Test Software

We needed two types of software for the tests: One for testing functionality, and one for the current measurements.

Functionality

The table below sums up, what the different programs did and for which functionality they were used for to test. The source code for the programs is listed in the appendix.

Program	*Action*	*Shows whether…*
blink internal	Runs in internal RAM Writes different values to the I/O port of the processor	the processor is able to run programs from internal memory and whether the I/O-ports works
blink external	Runs in external RAM Writes different values to the I/O port of the processor	the processor is able to run a program from external RAM, ie. read from and write to external RAM
ram-check	Runs in internal RAM Erases the external RAM and tries to write the values 0xAA, 0x55, and 0xFF to all addresses of the external RAM	the entire external RAM is accessible, i.e. the address- and data-busses is working
watchdog	Sets up the watchdog-timer, and waits longer and longer time before it resets the watchdog	the watchdog timer is working and whether the processor is able to boot from external flash
usart	Sets up the usart and waits until a character is received. Then returns the character+1	the usart is working

In addition a program named "nop" was used for the current measurements. The program sets up all I/O ports as high outputs, and then goes into an infinite loop.

[1] The topic is described further in our paper "Onboard Computer For Pico Satellite" located at http://www.dtusat.dtu.dk/files/download/244/pmp.pdf

Using our JTAG interface, we were able to disable different features of the processor. In the measurements below, the program “nop” was executed with different features disabled.

The Results

All the processors passed the functionality tests. The results of the power consumption tests are shown in the table below:

Board	**Yellow**	**Red**	**Green**	**Red**	**Green**	**Red**	**Green**
Radiation (kRad)	0	0	1	2	2	5	10
Comment			Fast radiated	Slow radiated	Fast radiated	Fast radiated	Fast radiated
Currents in mA							
blink - internal	23,8	24,5	26,3	26,5	32,9	32,5	34,6
blink - eksternal	18,8	18,9	20,8	17,6	17,9	19,3	20,2
Infinit internal loop							
"- All	20,8	19,2	19,7	22,3	31,5	30,4	29,1
"- PIO	16,7	15,5	15,3	18,6	28,3	20,1	25,5
"- TC0	16,7	15,5	15,2	18,5	27,9	20,6	25,6
"- TC1	16,7	15,5	15,2	18,5	28,0	20,6	25,5
"- TC2	16,8	15,5	15,2	18,5	28,2	20,6	25,5
"- USART0	17,8	16,4	16,1	19,4	29,3	20,5	26,4
"- USART1	17,9	16,5	16,2	19,5	29,8	20,6	26,4
"- None	16,5	15,1	14,7	18	28,5	19	25

Explanation of the table rows:

- Board – There are 3 different test PCBs
- Radiation – The amount of radiation that the component has been exposed to
- Comment – Comments about the radiation given to the component
- “blink - internal” – A measurement of the current when running the program “blink” in the internal RAM
- “blink - external” – A measurement of the current when running the program “blink” in the external RAM
- “infinite internal loop” – A number of measurement of the current when running a simple program that performs a loop operation in the internal RAM and with only some specific clocks of the processors special functions’ clock enabled
    - All – The infinite internal loop with all the special functions’ clocks enabled
    - PIO – The infinite internal loop with only the PIO clock enabled
    - TC0 – The infinite internal loop with only the TC0 clock enabled
    - TC1 – The infinite internal loop with only the TC1 clock enabled
    - TC2 – The infinite internal loop with only the TC2 clock enabled
    - USART0 – The infinite internal loop with only the USART0 clock enabled
    - USART1 – The infinite internal loop with only the PIO clock enabled
    - None - The infinite internal loop with none of the special functions’ clocks enabled

The measurements were performed with a standard multimeter and the precision is therefore limited. However the precision is good enough to determine the latch-up current threshold.
The results in this test vary much and they are difficult to elaborate much from. The only sure thing to note is that the radiation defiantly has a negative effect on the power consumption. There are several possible reasons that can have caused these variances in the measurements.

- Bad soldering
- Too much heat when soldering can damage the component
- The test was done on 3 different PCBs

As all the processors worked after the radiation, and the current consumption of the one exposed to 10 kRad did not even double, we believe that the processor will be able to operate in space with success.

The Flash Memory

Similar to the CPU it is essential to know how the flash component will react to long-term radiation exposure. The flash components were radiated with 1, 2 and 5 kRad and one was left irradiated. The measurements on the flash were done in three scenarios “read”, ”write” and “standby” mode.

Test Software

To test functionality of the flash, a program that first erases the entire flash (i.e. setting all bits to 1), and then writes zeroes to all addresses was created. The program returns an error by blinking with the LEDs on the test board, if it is not able to write zeros to all addresses.

To do tests concerning current, a program that continuously reads from the flash was created. Also a program that continuously writes to the flash was created. Finally the program “nop” also used when testing the processor was used to measure the current, when the flash was in standby mode.

The source code of the programs is listed in the appendix.

The Results

During the functionality test no errors were found.

The test results of the current measurements are shown in the table below.

Program\Radiation	**0 kRad**	**1 kRad**	**2 krad**	**5 kRad**
Read	3.1 mA	3.1 mA	3.1 mA	3.1 mA
Write	7.3 mA	7.7 mA	7.7 mA	7.6 mA
Standby	0 mA	0 mA	0 mA	0 mA

The test was conducted by measuring the current with a multimeter. In this test the multimeter was not exact enough to show any indication of exposition to the radiation. This result leads us to conclude that if the computer on DTUSat only

receives up to 5 kRad of radiation then the flash will continue to work and it will not increase its power consumption. Due to lack of components it was not possible to test more flash components, though a 10kRad test would have been nice. However it is not likely that the satellite will be irradiated with more than 5 kRad during the first year in space, which is the lifetime we expect the satellite to have.
This test does not show if bit flips occur while the component is exposed to radiation. This will be tested at Rigshospitalet at a later point.

Conclusion

The result of the measurements does not give any reason to be concerned to whether the components can survive the long-term radiation in space, since 10kRad is a much larger amount of radiation than we expect to encounter. The measurements also make it possible to determine the threshold for latch-up detection for the flight design.
However the tests do not show how the components will react to high-energy particles, which can cause latch-up. The latch-up test will be made at a later point in collaboration with the other hardware groups.
The tests we have conducted have resulted in the making of 3 good test boards that are well suited for software development.

Appendix

blink

```
int main() {
     int i, dir = -1,lys;
     int last;

     *(unsigned int*)(0xFFFF0000) = 0xff;
     *(unsigned int*)(0xFFFF0010) = 0xff;
     *(unsigned int*)(0xFFFF0030) = 0xff;
     last = 1;
     lys = 2;


     while (1) {
          *(unsigned int*)(0xFFFF0034) = lys;

          for(i=0; i< 10000; i++) {
               if (i%10==0) *(unsigned int*)(0xFFFF0034) = last;
               if ((i+9)%10==0) *(unsigned int*)(0xFFFF0030) = last;
          }

          *(unsigned int*)(0xFFFF0030) = lys;
          last = lys;

          if (dir==-1) {
               lys = lys << 1;
          } else {
               lys = lys >> 1;
          }

          if (lys == 256 || lys==1) dir =- dir;
     }
}
```

ram-check

```
#define PIO_BASE        0xFFFF0000
#define PIO_PER         PIO_BASE + 0x00
#define PIO_OER         PIO_BASE + 0x10
#define PIO_SODR        PIO_BASE + 0x30
#define PIO_CODR        PIO_BASE + 0x34

#define LED0            0x01
#define LED1            0x02
#define LED2            0x04
#define LED3            0x08
#define LED4            0x10
#define LED5            0x20
#define LED6            0x40
#define LED7            0x80
#define LED_ALL         0xFF

void led_on(int what) {
     *(volatile int*)(PIO_CODR) = what;
}
```

```
void led_off(int what) {
     *(volatile int*)(PIO_SODR) = what;
}

void led_init() {
     *(volatile int*)(PIO_PER) = LED_ALL;
     *(volatile int*)(PIO_OER) = LED_ALL;
}

void sleep(int j) {
     int i;
     for (i=0; i<j*500000; i++);
}

void error(int wrk) {
     int i;
     int work = wrk;

     for (i=0; i<4; i++) {
          led_off(LED_ALL);
          led_on(work);
          sleep(20);
          work = work >> 8;
     }

     led_off(LED_ALL);
     exit();
}

int main() {
     int i;
     char *ram_start = (char*)0x02000000;
     char *ram_end   = (char*)0x02080000;
     char *work = ram_start;

     led_init();

     led_on(LED_ALL);
     sleep(10);
     led_off(LED_ALL);


     while(work != ram_end ) {
          *(work) = 0x00;
          work++;
     }

     work = ram_start;

     while(work != ram_end ) {
          if ( *(work) != 0x00) error((int)work);

          *(work) = 0x55;
       if ( *(work) != 0x55) error((int)work);

          *(work) = 0xAA;
       if ( *(work) != 0xAA) error((int)work);

          *(work) = 0xFF;
       if ( *(work) != 0xFF) error((int)work);
          work++;
     }

     led_on(LED_ALL);

     return 0;
}
```

watchdog

```
#define WD_OMR     *(volatile int*)(0xFFFF8000)
#define WD_CMR     *(volatile int*)(0xFFFF8004)
#define WD_CR      *(volatile int*)(0xFFFF8008)
#define WD_SR      *(volatile int*)(0xFFFF800C)

#define PIO_PER    *(volatile int*)(0xFFFF0000)
#define PIO_OER    *(volatile int*)(0xFFFF0010)
#define PIO_SODR   *(volatile int*)(0xFFFF0030)
#define PIO_CODR   *(volatile int*)(0xFFFF0034)

int main() {
     int i, j;

     PIO_PER = 0xFF;
     PIO_OER = 0xFF;
     PIO_SODR = 0xFF;

     PIO_CODR = 0xAA;
     for(j=0; j<50000; j++);
     PIO_SODR = 0xFF;

     PIO_CODR = 0x55;
     for(j=0; j<50000; j++);
     PIO_SODR = 0xFF;

     PIO_CODR = 0xAA;
     for(j=0; j<50000; j++);
     PIO_SODR = 0xFF;

     PIO_CODR = 0x55;
     for(j=0; j<50000; j++);
     PIO_SODR = 0xFF;


     WD_OMR = 0x2340; /* Clear WDEN */
     WD_CMR = 0x373E; /* HPCV=15, WDCLKS=MCK/128 */
     WD_CR  = 0xC071; /* Restart timer */
     WD_OMR = 0x2343; /* Set WDEN, RSTEN */

     while(1) {
          for(i=1; i<256; i++) {
               PIO_SODR=0xFF;
               PIO_CODR = i;

               for(j=0; j<1000*i; j++);
               WD_CR  = 0xC071; /* Restart timer */
          }
     }

}
```

usart

```
#define US0_CR      *(volatile int*)0xfffd0000
#define US0_MR      *(volatile int*)0xfffd0004
#define US0_IDR     *(volatile int*)0xfffd000C
#define US0_CSR     *(volatile int*)0xfffd0014
#define US0_RHR     *(volatile int*)0xfffd0018
#define US0_THR     *(volatile int*)0xfffd001c
#define US0_BRGR    *(volatile int*)0xfffd0020
#define US0_RTOR    *(volatile int*)0xfffd0024
#define US0_TTGR    *(volatile int*)0xfffd0028
#define US0_RCR     *(volatile int*)0xfffd0034
#define US0_TCR     *(volatile int*)0xfffd003C

#define PIO_PDR     *(volatile int*)0xffff0004
#define PS_PCER     *(volatile int*)0xffff4004

#define PIO_BASE    0xFFFF0000
#define PIO_PER     PIO_BASE + 0x00
#define PIO_OER     PIO_BASE + 0x10
#define PIO_SODR    PIO_BASE + 0x30
#define PIO_CODR    PIO_BASE + 0x34

#define LED0        0x01
#define LED1        0x02
#define LED2        0x04
#define LED3        0x08
#define LED4        0x10
#define LED5        0x20
#define LED6        0x40
#define LED7        0x80
#define LED_ALL     0xFF

void led_on(int what) {
     *(volatile int*)(PIO_CODR) = what;
}

void led_off(int what) {
     *(volatile int*)(PIO_SODR) = what;
}
void led_init() {
     *(volatile int*)(PIO_PER) = LED_ALL;
     *(volatile int*)(PIO_OER) = LED_ALL;
}
void sleep(int j) {
     int i;
     for (i=0; i<j*500000; i++);
}

int main (void) {
    int value=0;
    US0_CR = 0x000001ac;
    US0_RTOR = 0;
    US0_TTGR = 0;
    US0_RCR = 0;
    US0_TCR = 0;
    US0_MR = 0x000008c0;
    US0_IDR = 0xffffffff;
    US0_BRGR = 0x00000050;
    PS_PCER = 0xffffffff;
    PIO_PDR = 0x0070e000;
```

```
     US0_CR = 0x00000050;
     led_init();

     led_on(LED_ALL);
     sleep(5);
     led_off(LED_ALL);

    while(1) {
        value = US0_RHR;
        led_on(value);
        led_off(value);

        US0_THR = value+1;
        for ( ; (US0_CSR & (1 << 1)) == 0 ; );

    }

    return(0);
}
```

flash-read

```
int main() {
     int i;


     *(unsigned int*)(0xFFFF0000) = 0xff;
     *(unsigned int*)(0xFFFF0010) = 0xff;
     *(unsigned int*)(0xFFFF0034) = 0xff;

    for (i=0; i<5000000; i++);

     *(unsigned int*)(0xFFFF0000) = 0xff;
     *(unsigned int*)(0xFFFF0010) = 0xff;
     *(unsigned int*)(0xFFFF0030) = 0xff;


     while (1) {
         i = *(volatile int *)(0x03000000);
     }

}
```

flash-write

```
#define FLASH_Unlock_first       0xAA
#define FLASH_Unlock_second      0x55
#define FLASH_Unlock_sector_erase0x80
#define FLASH_Program            0xA0
#define FLASH_Sector_erase       0x30

#define FLASH_Sector_size         0x10000

#define BIT_3_MASK               0x08

#define FLASH_ERR_OK             0x00
#define FLASH_ERR_TIMEOUT        0x01
#define FLASH_ERR_VERIFY         0x02

#define TIMEOUT                   50000
```

```
#define PIO_BASE        0xFFFF0000
#define PIO_PER          PIO_BASE + 0x00
#define PIO_OER         PIO_BASE + 0x10
#define PIO_SODR         PIO_BASE + 0x30
#define PIO_CODR         PIO_BASE + 0x34

#define LED0            0x01
#define LED1            0x02
#define LED2            0x04
#define LED3            0x08
#define LED4            0x10
#define LED5            0x20
#define LED6            0x40
#define LED7            0x80
#define LED_ALL         0xFF

extern unsigned char prog_array[];
extern long prog_array_size();

void led_on(int what) {
     *(volatile int*)(PIO_CODR) = what;
}

void led_off(int what) {
     *(volatile int*)(PIO_SODR) = what;
}

void led_init() {
     *(volatile int*)(PIO_PER) = LED_ALL;
     *(volatile int*)(PIO_OER) = LED_ALL;
}

int program_byte(void *adr, unsigned char byte) {
   volatile unsigned char *flash=(volatile unsigned char*)FLASH_BASE;
     volatile unsigned char *prg_adr = (volatile unsigned char *)adr;
     int timeout = TIMEOUT;
     int res = FLASH_ERR_OK;

     /* Write Program Command Sequence */
     *flash = FLASH_Unlock_first;
     *flash = FLASH_Unlock_second;
     *flash = FLASH_Program;
     *prg_adr = byte;

     /* Data Poll and Verify */
     while (--timeout > 0) {
        if (*prg_adr == byte) break;
     }

     if (timeout < 0) res = FLASH_ERR_TIMEOUT;

     return res;
}


int erase_sector(void *sect) {
   volatile unsigned char *flash=(volatile unsigned char*)FLASH_BASE;
     volatile unsigned char *sect_adr=(volatile unsigned char *)sect;
     volatile unsigned char *adr;
     int timeout = TIMEOUT;
```

```
	int res = FLASH_ERR_OK;

	/* Write Erase Command Sequence */
	*flash = FLASH_Unlock_first;
	*flash = FLASH_Unlock_second;
	*flash = FLASH_Unlock_sector_erase;
	*flash = FLASH_Unlock_first;
	*flash = FLASH_Unlock_second;
	*sect_adr = FLASH_Sector_erase;

	/* Wait for erase timer to timeout */
	while ( (*sect_adr & BIT_3_MASK) == 1 );

	/* Data Poll from system */
	while (--timeout > 0) {
		if (*sect_adr == 0xFF ) break;
	}

	if (timeout < 0) res = FLASH_ERR_TIMEOUT;

	/* Verify data */
	for (adr = sect_adr; adr < sect_adr + FLASH_Sector_size; adr++){
		if (*adr != 0xFF && res == FLASH_ERR_OK) {
			res = FLASH_ERR_VERIFY;
			break;
		}
	}

	return res;
}

void sleep(int j) {
	int i;
	for (i=0; i<j*500000; i++);
}

int main() {
	int i=0;
	char *flash_start = (char*)FLASH_BASE;
	char *flash_end   = (char*)(FLASH_BASE + 0x200000);
	int  flash_block_size = 0x10000;
	char *work = flash_start;

	led_init();

	led_on(LED_ALL);
	sleep(10);
	led_off(LED_ALL);
		while (work != flash_end) {
		led_off(LED_ALL);
		led_on(i++);
		erase_sector(work);
		sleep(10);
		work += flash_block_size;
	}

	led_on(LED_ALL);
	sleep(10);
	led_off(LED_ALL);

	work = flash_start;
```

```
     sleep(10);

     while(work != flash_end) {
       program_byte(work, ((int)work) % 0xff);
       work++;
     }

     led_on(LED_ALL);

     return 0;
}
```

flash-check

```
#define FLASH_Unlock_first       0xAA
#define FLASH_Unlock_second      0x55
#define FLASH_Unlock_sector_erase0x80
#define FLASH_Program            0xA0
#define FLASH_Sector_erase       0x30

#define FLASH_Sector_size         0x10000

#define BIT_3_MASK               0x08

#define FLASH_ERR_OK             0x00
#define FLASH_ERR_TIMEOUT        0x01
#define FLASH_ERR_VERIFY         0x02

#define TIMEOUT                   50000

#define PIO_BASE        0xFFFF0000
#define PIO_PER          PIO_BASE + 0x00
#define PIO_OER         PIO_BASE + 0x10
#define PIO_SODR         PIO_BASE + 0x30
#define PIO_CODR         PIO_BASE + 0x34

#define LED0            0x01
#define LED1            0x02
#define LED2            0x04
#define LED3            0x08
#define LED4            0x10
#define LED5            0x20
#define LED6            0x40
#define LED7            0x80
#define LED_ALL         0xFF

void led_on(int what) {
     *(volatile int*)(PIO_CODR) = what;
}

void led_off(int what) {
     *(volatile int*)(PIO_SODR) = what;
}

void led_init() {
     *(volatile int*)(PIO_PER) = LED_ALL;
     *(volatile int*)(PIO_OER) = LED_ALL;
}
```

```
int program_byte(void *adr, unsigned char byte) {
   volatile unsigned char *flash=(volatile unsigned char*)FLASH_BASE;
     volatile unsigned char *prg_adr = (volatile unsigned char *)adr;
     int timeout = TIMEOUT;
     int res = FLASH_ERR_OK;

     /* Write Program Command Sequence */
     *flash = FLASH_Unlock_first;
     *flash = FLASH_Unlock_second;
     *flash = FLASH_Program;
     *prg_adr = byte;

     /* Data Poll and Verify */
     while (--timeout > 0) {
          if (*prg_adr == byte) break;
     }

     if (timeout < 0) res = FLASH_ERR_TIMEOUT;

     return res;
}


int erase_sector(void *sect) {
   volatile unsigned char *flash=(volatile unsigned char*)FLASH_BASE;
     volatile unsigned char *sect_adr=(volatile unsigned char *)sect;
     volatile unsigned char *adr;
     int timeout = TIMEOUT;
     int res = FLASH_ERR_OK;

     /* Write Erase Command Sequence */
     *flash = FLASH_Unlock_first;
     *flash = FLASH_Unlock_second;
     *flash = FLASH_Unlock_sector_erase;
     *flash = FLASH_Unlock_first;
     *flash = FLASH_Unlock_second;
     *sect_adr = FLASH_Sector_erase;

     /* Wait for erase timer to timeout */
     while ( (*sect_adr & BIT_3_MASK) == 1 );

     /* Data Poll from system */
     while (--timeout > 0) {
          if (*sect_adr == 0xFF ) break;
     }

     if (timeout < 0) res = FLASH_ERR_TIMEOUT;

     /* Verify data */
     for (adr = sect_adr; adr < sect_adr + FLASH_Sector_size; adr++){
          if (*adr != 0xFF && res == FLASH_ERR_OK) {
               res = FLASH_ERR_VERIFY;
               break;
          }
     }

     return res;
}

void sleep(int j) {
     int i;
     for (i=0; i<j*500000; i++);
}

void error(int wrk) {
     int i;
     int work = wrk;
     led_off(LED_ALL);
     led_on(0xAA);
     sleep(2);
     led_off(LED_ALL);
     led_on(0x55);
     sleep(2);
     led_off(LED_ALL);
     led_on(0xAA);
     sleep(2);
     led_off(LED_ALL);
     led_on(0x55);
     sleep(2);

     for (i=0; i<4; i++) {
         led_on(LED_ALL);
         sleep(2);
          led_off(LED_ALL);
          led_on(work);
          sleep(20);
          work = work >> 8;
     }

     led_off(LED_ALL);
     exit();
}


int main() {

     int i=0;
     char *flash_start = (char*)FLASH_BASE;
     char *flash_end   = (char*)(FLASH_BASE + 0x200000);
     int  flash_block_size = 0x10000;
     char *work = flash_start;

     led_init();

     led_on(LED_ALL);
     sleep(10);
     led_off(LED_ALL);

     while (work != flash_end) {
          led_off(LED_ALL);
          led_on(i++);
          erase_sector(work);
          sleep(10);
          work += flash_block_size;
     }

     led_on(LED_ALL);
     sleep(10);
     led_off(LED_ALL);

     work = flash_start;
     sleep(10);

     while(work != flash_end) {
          if ( (*(work) & 0xFF) != 0xFF) {
                error((int)work);
          }

          program_byte(work, 0x00);
       if ( *(work) != 0x00) error((int)work);

       work++;
     }

     led_on(LED_ALL);

     return 0;
}
```

ram-check

```
#define PIO_BASE        0xFFFF0000
#define PIO_PER         PIO_BASE + 0x00
#define PIO_OER         PIO_BASE + 0x10
#define PIO_SODR        PIO_BASE + 0x30
#define PIO_CODR        PIO_BASE + 0x34

#define LED0                 0x01
#define LED1                 0x02
#define LED2                 0x04
#define LED3                 0x08
#define LED4                 0x10
#define LED5                 0x20
#define LED6                 0x40
#define LED7                 0x80
#define LED_ALL              0xFF

void led_on(int what) {
     *(volatile int*)(PIO_CODR) = what;
}

void led_off(int what) {
     *(volatile int*)(PIO_SODR) = what;
}

void led_init() {
     *(volatile int*)(PIO_PER) = LED_ALL;
     *(volatile int*)(PIO_OER) = LED_ALL;
}

void sleep(int j) {
     int i;
     for (i=0; i<j*500000; i++);
}

void error(int wrk, int err) {
     int i;
     int work = wrk;
     led_off(LED_ALL);
     led_on(0xAA);
     sleep(2);
     led_off(LED_ALL);
     led_on(0x55);
     sleep(2);
     led_off(LED_ALL);
     led_on(0xAA);
     sleep(2);
     led_off(LED_ALL);
     led_on(0x55);
     sleep(2);

     for (i=0; i<4; i++) {
         led_on(LED_ALL);
         sleep(2);
          led_off(LED_ALL);
          led_on(work);
          sleep(20);
          work = work >> 8;
     }

     led_on(LED_ALL);
     sleep(2);
     led_off(LED_ALL);
     led_on(err);
     sleep(20);

     led_off(LED_ALL);
     exit();
}

int main() {
     int i;
     char *ram_start = (char*)0x02000000;
     char *ram_end   = (char*)0x02080000;
     char *work = ram_start;

     led_init();

     led_on(LED_ALL);
     sleep(10);
     led_off(LED_ALL);

     while(work != ram_end ) {
          *(work) = 0x00;
          work++;
     }

     work = ram_start;

     while(work != ram_end ) {
          if ( *(work)&0xff != 0x00)
              error((int)work,(int)(*(work)&0xff) );

          *(work) = 0x55;
          if ( *(work) != 0x55) error((int)work,(int)(*(work)&0xff));

          *(work) = 0xAA;
          if ( *(work) != 0xAA) error((int)work,(int)(*(work)&0xff));

          *(work) = 0xFF;
          if ( *(work) != 0xFF) error((int)work,(int)(*(work)&0xff));
```

```
        work++;
    }

    led_on(LED_ALL);

    return 0;
}
```