

# Prevedba C programa in funkcij v ARM zbirnik

## Primer projekta: C\_PIO\_LED\_OnOff\_Delay\_Demo

Optimizacija :

- Nastavitve optimizacije spreminjate v Project->Settings->Compiler->Options
- O2 nastavev je bila uporabljena za optimizacijo. Ta nastavev zelo vpliva na to, kakšen je preveden program v zbirnem jeziku. Optimizacija O3 z vidika razumevanja in preglednosti programa naredi preveč (loop unrolling, ...).

Glavni program: main.c

```
#include "Main.h"
#include "AT91SAM9260.h"

/* Zaradi napake v prevajalniku mora biti definirana ta spremenljivka */
int dummy = 1;

#define PORT  AT91C_BASE_PIOC          // PIN PORTC1
#define PIN   1                       // PIN PORTC1

void initPIO_Port(AT91S_PIO *port) {
    port->PIO_PER=1<<PIN;
    port->PIO_OER=1<<PIN;
} // initPIO_Port

void LED_On(AT91PS_PIO port) {
    port->PIO_CODR=1<<PIN;
}

void LED_Off(AT91PS_PIO port) {
    port->PIO_SODR=1<<PIN;
}

void Delay(unsigned int Counter) {
    volatile unsigned int i,j,tmp;

    for (i=0;i<Counter;i++) {
        for (j=0;j<48000;j++) {
            tmp=j ;
        };
    };
}

int main(void) {
    initPIO_Port(PORT);

    /* Loop for ever */
    while (1) {
        Delay(30);
        LED_On(PORT);
        Delay(30);
        LED_Off(PORT);
    };
}
```

Podatkovna struktura za dostop do registrov za PIO vrata A,B,C:

AT91SAM9260.h (izvleček):

```
typedef volatile unsigned int AT91_REG;// Hardware register definition

#define AT91C_BASE_PIOA ((AT91PS_PIO) 0xFFFFF400) // (PIOA) Base Address
#define AT91C_BASE_PIOB ((AT91PS_PIO) 0xFFFFF600) // (PIOB) Base Address
#define AT91C_BASE_PIOC ((AT91PS_PIO) 0xFFFFF800) // (PIOC) Base Address

// *****
//          SOFTWARE API DEFINITION FOR Parallel Input Output Controller
// *****
typedef struct _AT91S_PIO {
    AT91_REG PIO_PER;    // PIO Enable Register
    AT91_REG PIO_PDR;    // PIO Disable Register
    AT91_REG PIO_PSR;    // PIO Status Register
    AT91_REG Reserved0[1]; //
    AT91_REG PIO_OER;    // Output Enable Register
    AT91_REG PIO_ODR;    // Output Disable Register
    AT91_REG PIO_OSR;    // Output Status Register
    AT91_REG Reserved1[1]; //
    AT91_REG PIO_IFER;   // Input Filter Enable Register
    AT91_REG PIO_IFDR;   // Input Filter Disable Register
    AT91_REG PIO_IFSR;   // Input Filter Status Register
    AT91_REG Reserved2[1]; //
    AT91_REG PIO_SODR;   // Set Output Data Register
    AT91_REG PIO_CODR;   // Clear Output Data Register
    AT91_REG PIO_ODSR;   // Output Data Status Register
    AT91_REG PIO_PDSR;   // Pin Data Status Register
    AT91_REG PIO_IER;    // Interrupt Enable Register
    AT91_REG PIO_IDR;    // Interrupt Disable Register
    AT91_REG PIO_IMR;    // Interrupt Mask Register
    AT91_REG PIO_ISR;    // Interrupt Status Register
    AT91_REG PIO_MDER;   // Multi-driver Enable Register
    AT91_REG PIO_MDDR;   // Multi-driver Disable Register
    AT91_REG PIO_MDSR;   // Multi-driver Status Register
    AT91_REG Reserved3[1]; //
    AT91_REG PIO_PPUDR;  // Pull-up Disable Register
    AT91_REG PIO_PPUER;  // Pull-up Enable Register
    AT91_REG PIO_PPUSR;  // Pull-up Status Register
    AT91_REG Reserved4[1]; //
    AT91_REG PIO_ASR;    // Select A Register
    AT91_REG PIO_BSR;    // Select B Register
    AT91_REG PIO_ABSR;   // AB Select Status Register
    AT91_REG Reserved5[9]; //
    AT91_REG PIO_OWER;   // Output Write Enable Register
    AT91_REG PIO_OWDR;   // Output Write Disable Register
    AT91_REG PIO_OWSR;   // Output Write Status Register
} AT91S_PIO, *AT91PS_PIO;
```

Primer prevajanja programa main.c v zbirnik (main.s) z nastavitvijo optimizacije O2.

Zbirni jezik	C programski jezik
<pre> mov    r3, #2 str    r3, [r0, #0] str    r3, [r0, #16] bx     lr </pre>	<pre> void <b>initPIO_Port</b> (AT91S_PIO *port) {     port-&gt;PIO_PER=1&lt;&lt;PIN;     port-&gt;PIO_OER=1&lt;&lt;PIN; } // initPIO_Port </pre>
<pre> mov    r3, #2 str    r3, [r0, #52] bx     lr </pre>	<pre> void <b>LED_On</b> (AT91PS_PIO port) {     port-&gt;PIO_CODR=1&lt;&lt;PIN; } </pre>
<pre> mov    r3, #2 str    r3, [r0, #48] bx     lr </pre>	<pre> void <b>LED_Off</b> (AT91PS_PIO port) {     port-&gt;PIO_SODR=1&lt;&lt;PIN; } </pre>
<p>Delay:</p> <p>@ Function supports interworking.          @ args = 0, pretend = 0, frame = 16          @ frame_needed = 0,          uses_anonymous_args = 0          @ link register save eliminated.</p> <pre> sub    sp, sp, #16 mov    r1, #0      @ unsigned int i str    r1, [sp, #4] @ var i ldr    r3, [sp, #4] cmp    r0, r3 bls    .L4 ldr    r2, .L14 @ 47999  .L9: str    r1, [sp, #8] @ var j ldr    r3, [sp, #8] cmp    r3, r2 bhi    .L8  .L10: ldr    r3, [sp, #8] @ var j str    r3, [sp, #12] @ tmp = j ldr    r3, [sp, #8] @ var j add    r3, r3, #1 str    r3, [sp, #8] @ var j ldr    r3, [sp, #8] @ var j cmp    r3, r2 bls    .L10  .L8: ldr    r3, [sp, #4] @ var i add    r3, r3, #1 str    r3, [sp, #4] @ var i ldr    r3, [sp, #4] @ var i cmp    r3, r0 bcc    .L9  .L4: add    sp, sp, #16 bx     lr .align 2  .L14: .word 47999 </pre>	<pre> void <b>Delay</b>(unsigned int Counter) {      volatile unsigned int i,j,tmp;      // Prostor za lokalne spremenljivke     for (i=0;i&lt;Counter;i++) {          // Counter(r0) &lt;= i(r3) ?          for (j=0;j&lt;48000;j++) {              // j &gt; 47999 ?              tmp=j ;              // j++              // j &lt;= 47999 ?          };          // i++          // i &lt; Counter ?      };  } </pre>

Spr.	Nasl.
i	sp+4
j	sp+8
tmp	sp+12

<pre> main:     @ Function supports interworking.     @ Volatile: function does not return.     @ args = 0, pretend = 0, frame = 0     @ frame_needed = 0, uses_anonymous_args = 0     mvn    r5, #0     mov    r4, #2     .     stmfd  sp!, {r3, lr}     str    r4, [r5, #-2047] @ F800     str    r4, [r5, #-2031] @ F810  .L17:     mov    r0, #30     bl    Delay      mov    r0, #30     str    r4, [r5, #-1995] @ F834     bl    Delay     str    r4, [r5, #-1999] @ F838     b     .L17 </pre>	<pre> int main(void) {      initPIO_Port(PORT);      /* Loop for ever */     while (1) {          Delay(30);          LED_On(PORT);         Delay(30);         LED_Off(PORT);     }; } </pre>
---	---

## Opisi optimizacijskih nastavitev :

Vir: <https://gcc.gnu.org/onlinedocs/gcc-4.1.0/gcc/Optimize-Options.html>

### 8.3.1.3 Optimization Levels

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the subprogram and get exactly the results you would expect from the source code.

Turning on optimization makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

If you use multiple `-O` options, with or without level numbers, the last such option is the one that is effective.

The default is optimization off. This results in the fastest compile times, but GNAT makes absolutely no attempt to optimize, and the generated programs are considerably larger and slower than when optimization is enabled. You can use the ``-O'` switch (the permitted forms are ``-O0'`, ``-O1'`, ``-O2'`, ``-O3'`, and ``-Os'`) to ``gcc'` to control the optimization level:

- ``-O0'`  
No optimization (the default); generates unoptimized code but has the fastest compilation time.

Note that many other compilers do fairly extensive optimization even if 'no optimization' is specified. With gcc, it is very unusual to use `-O0` for production if execution time is of any concern, since `-O0` really does mean no optimization at all. This difference between gcc and other compilers should be kept in mind when doing performance comparisons.

- ``-O1'`  
Moderate optimization; optimizes reasonably well but does not degrade compilation time significantly.
- ``-O2'`  
Full optimization; generates highly optimized code and has the slowest compilation time.
- ``-O3'`  
Full optimization as in ``-O2'`; also uses more aggressive automatic inlining of subprograms within a unit ([Inlining of Subprograms](#)) and attempts to vectorize loops.
- ``-Os'`  
Optimize space usage (code and data) of resulting program.

Higher optimization levels perform more global transformations on the program and apply more expensive analysis algorithms in order to generate faster and more compact code. The price in compilation time, and the resulting improvement in execution time, both depend on the particular application and the hardware environment. You should experiment to find the best level for your application.

Since the precise set of optimizations done at each level will vary from release to release (and sometime from target to target), it is best to think of the optimization settings in general terms. See the 'Options That Control Optimization' section in *Using the GNU Compiler Collection (GCC)* for details about the '-O' settings and a number of '-f' options that individually enable or disable specific optimizations.

Unlike some other compilation systems, 'gcc' has been tested extensively at all optimization levels. There are some bugs which appear only with optimization turned on, but there have also been bugs which show up only in 'unoptimized' code. Selecting a lower level of optimization does not improve the reliability of the code generator, which in practice is highly reliable at all optimization levels.

Note regarding the use of '-O3': The use of this optimization level is generally discouraged with GNAT, since it often results in larger executables which may run more slowly. See further discussion of this point in [Inlining of Subprograms](#).