AT91 Assembler Code Startup Sequence for C Code Applications Software Based on the AT91SAM7S64 Evaluation Board

AT91
ARM® Thumb®
Microcontrollers

Application
Note

1. Introduction

For reasons of modularity and portability most application code for the AT91SAM7S ARM®-based microcontrollers is written in C. However, the startup sequence required to initialize the ARM Processor Mode and certain key peripherals is heavily dependent on the register architecture and memory mapping processor. For this reason the C-startup sequence is written in assembler.

This Application Note describes an example of the AT91SAM7S C-startup sequence. It is based on the C-startup sequence for the AT91SAM7S64 Evaluation Board working with the IAR 4.11A Development Tools. Further examples of C-startup sequences are available in the AT91 software package.

The C-startup sequence is activated on power-up and after a reset.

6131A–ATARM–04-Mar-05

2. C-Startup Sequence

Following reset, the processor starts to fetch instructions from 0x0, therefore there must be some executable code accessible from that address. In an AT91SAM7S embedded system, this requires embedded Flash to be present, at least initially, at address 0x0.

The simplest layout is accomplished by locating the application in embedded Flash at address 0x0 in the memory map. The application can then branch to the real entry point when it executes its first instruction at the reset vector at address 0x0.

All applications written for AT91SAM7S ARM-based systems are embedded applications that are contained in embedded Flash and execute on reset.

There are a number of factors that must be considered when writing embedded operating systems, or embedded applications that execute from reset without an operating system, including:

- Reset entry point in embedded Flash.
- Initializing the execution environment, such as exception vectors, stacks, I/Os.
- Initializing the application.
- For example, copying initialization values for initialized variables from Flash to RAM and resetting all other variables to zero.
- Linking an embedded executable image to place code and data in specific locations in RAM memory.

For an embedded application without an operating system, the code in Flash must provide a way for the application to initialize itself and start executing. No automatic initialization takes place on reset, therefore the application entry point must perform some initialization before it can call any C code.

The initialization code, located at address zero after reset, must:

- Mark the default entry point for the initialization code.
- Set up exception vectors.
- Initialize the memory system.
- Initialize any critical I/O devices.
- Initialize the stack pointer registers.
- Initialize any Register required by the interrupt system.
- Enable interrupts (if handled by the initialization code).
- Change processor mode if necessary.
- Change processor state if necessary.

After the environment has been initialized, the sequence continues with the application initialization and should enter the C code.

The C-startup file is the first file executed at power on and performs initialization of the microcontroller from the reset vector up to the calling of the application’s main routine.

The main program should be a closed loop and should not return. The ARM core begins executing instructions from address 0x0 at reset. For an AT91SAM7S embedded system this means in embedded Flash at address 0x0 when the system is reset.

3. C - Startup Example

A generic start-up file is included within this Application Note and others are available in the AT91 software package. The example described is based on the AT91SAM7S64 Evaluation Board, C-startup sequence working with IAR V4.11A Development Tool and debugging in embedded Flash Memory or RAM Memory. This file must be modified in order to fit the needs of the user application.

The AT91SAM7S64 Evaluation Board is described in the AT91 software package inside the "compil" subdirectory. Each of these subdirectories contains the following files:

- The board.h file, defines the components of the board in C.
- One Cstartup.s79 file, defines standard boot for the board according to the software development tools used.
- One Cstartup_xxx.c file, defines standard low level initialization for the board according to the software development tools used.

The AT91 software package provides C-Startup files that explain how to boot an AT91SAM7S device and how to branch to the main C function. The C-Startup file takes into account the specific features of the device, the board specific characteristics and the debug level required.

Note: The software example is delivered "As Is" without warranty or condition of any kind, either express, implied or statutory. This includes without limitation any warranty or condition with respect to merchantability or fitness for any particular purpose, or against the infringements of intellectual property rights of others.

3.1 Area Definition

In an ARM assembly language source file, the start of the module is marked by the PROGRAM directive, this directive sets the module for the linker. Following this module the second directive defines the segment named RSEG and the specific segment ICODE that defines C-startup and exception code area.

At the reset, the ARM core sets ARM Mode and fetches a 32-bit ARM Instruction. The CODE32 directive sets the subsequent instructions to be interpreted as 32-bit ARM Instructions.

An embedded image is placed in embedded Flash at 0x0 by the assembler ORG directive.

```
;------------------------------------------------------------------
    PROGRAM ?RESET
    RSEG   ICODE:CODE:ROOT(2)
    CODE32   ; Always ARM mode after reset
    org   0
reset:
```

3.2 Setup Exception Vectors

Exception Vectors are setup sequentially through the address space with branches to nearby labels or branches and links to subroutines. During the normal flow of execution through a program, the program counter increases enable the processor to handle events generated by internal or external sources. Processor exceptions occur when the normal flow of execution is diverted. Examples of such events are:

- Externally generated interrupts.
- An attempt by the processor to execute an Undefined Instruction.

The previous processor status Is preserved in SPSR. The Link register (R14) and the stack register (R13) are also preserved by the hard-coded sequence. When handling such exceptions, so that execution of the program that was running when the exception occurred can resume when the appropriate exception routine has completed, the initialization code must set up the required exception vectors (see Table 3-1).

The Flash is located at address 0x0 and the vectors consist of a sequence of hard-coded instructions to branch to the handler for each exception. These vectors are mapped at address 0x0....

Table 3-1. Exception Vectors

Exception	Description
Reset	Occurs when the processor reset pin is asserted. This exception is only expected to occur for signalling power-up, or for resetting. A soft reset can be done by branching to the reset vector.
Undefined Instruction	Occurs if neither the processor, or any attached coprocessor, recognizes the currently executing instruction.
Software Interrupt (SWI)	This is a user-defined synchronous interrupt instruction. It allows a program running in User Mode, for example, to request privileged operations that run in Supervisor Mode, such as an RTOS function.
Prefetch Abort	Occurs when the processor attempts to execute an instruction that has prefetched from an illegal address.
Data Abort	Occurs when a data transfer instruction attempts to load or store data at an illegal address.
IRQ	Occurs when the processor external interrupt request pin is asserted (LOW) and the I bit in the CPSR is clear.
FIQ	Occurs when the processor external fast interrupt request pin is asserted (LOW) and the F bit in the CPSR is clear or when an internal interrupt is redirected by Fast Forcing.

Processor exception handling is controlled by a vector table. The vector table is a reserved area of 32 bytes, usually at the bottom of the memory map. It has one word of space allocated to each exception type, and one word that is currently reserved. Because there is not enough space to contain the full code for a handler except for the FIQ interrupt, the vector entry for each exception type contains a branch instruction or load pc instruction to continue execution with the appropriate handler.The FIQ exception handler can be written directly at the exception vector.

```
;--------------------
;- Exception vectors
;--------------------
       B     InitReset ; 0x00 Reset handler
undefvec:
       B     ndefvec ; 0x04 Undefined Instruction
swivec:
       B     swivec ; 0x08 Software Interrupt
pabtvec:
       B     pabtvec ; 0x0C Prefetch Abort
dabtvec:
       B     dabtvec ; 0x10 Data Abort
rsvdvec:
       B     rsvdvec ; 0x14 reserved
irqvec:
       B     I RQ_Handler_Entry   ; 0x18 IRQ
fiqvec:                       ; FIQ Handling
```

Table 3-2. Exception Vector Mapping

Mapping	Exception Vectors
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt (SWI)
0x0000 000C	Prefetch Abort
0x0000 0010	Data Abort
0x0000 0014	Reserved
0x0000 0018	IRQ
0x0000 001C	FIQ

3.3 Reset Handler

From here, the code is executed from address 0.

```
;----------------
;- Reset Handler
;----------------
InitReset:
```

3.4 Low Level Initialization

After reset, the PLL, Embedded Flash Controller and Watchdog are not configured and some peripherals that must be initialized before enabling interrupts should be considered as critical. If these peripherals are not initialized at this point, they might cause spurious interrupts when interrupts are enabled.

The AT91F_LowLevelInit function is defined in the C file from the AT91 software packages associated to the evaluation board.

To call this function before C initialization, the assembly C-startup sets the C stack at the end of RAM address. This function can be a write in Thumb® instruction or ARM instruction and called by the BX interworking ARM instruction.

```
;-----------------------
;- Low level init
;-----------------------
;- minimum C initialization
;- call  AT91F_LowLevelInit(void)
       ldr     r13,=__iramend; temporary stack in internal RAM
;--Call Low level init function in ABSOLUTE through the Interworking
       ldr     r0,=AT91F_LowLevelInit
       mov     lr, pc
       bx      r0
```

3.5 AT91F_LowLevelInit Function

This function performs very low level hardware initialization. The function initializes itself.

- Flash Wait state and time setting depend on the PLL setting and the external oscillator.

At reset, the AT91SAM7S microcontroller starts with Flash default value at slow clock.

- Disable the Watchdog.

At reset, the AT91SAM7S microcontroller has enabled the Watchdog.

- Set the PLL.

At reset, the AT91SAM7S microcontroller starts with the internal slow clock RC oscillator to minimize the power required to start up the system and the main oscillator is disabled. The PLL can be started by setting the configuration to run with the PLL to speed up the startup sequence.

- AIC vector initialization

After reset, the Advanced Interrupt Controller (AIC) is not configured. The AT91F_LowLevelInit function initializes the AIC by setting up the default interrupt vectors. The default Interrupt handler functions are defined in the C-startup file. These functions can be re-write.

The following function is used for AT91SAM7S64 evaluation board initialization. The specific directive “@ ICODE” links the object code in the C-startup segment area.

```
void AT91F_LowLevelInit(void) @ "ICODE"
{
 int            i;
 AT91PS_PMC     pPMC = AT91C_BASE_PMC;
//* Set Embedded Flash Controller
  AT91C_BASE_MC->MC_FMR= ((AT91C_MC_FMCN)&(50 <<16)) | AT91C_MC_FWS_1FWS;
//* Watchdog Disable
  AT91C_BASE_WDTC->WDTC_WDMR= AT91C_SYSC_WDDIS;
//* Set MCK
// 1 Enabling the Main Oscillator:
  pPMC->PMC_MOR= ((AT91C_CKGR_OSCOUNT & (0x06<<8) | AT91C_CKGR_MOSCEN));
// Startup time
  while(!(pPMC->PMC_SR & AT91C_PMC_MOSCS));
//* Set PLL
  pPMC->PMC_PLLR= ((AT91C_CKGR_DIV & 0x05) |
                         (AT91C_CKGR_PLLCOUNT & (16<<8)) |
                         (AT91C_CKGR_MUL & (25<<16)));
//* Startup time
  while(!(pPMC->PMC_SR & AT91C_PMC_LOCK));
//* select the PLL clock divided by 2
  pPMC->PMC_MCKR= AT91C_PMC_CSS_PLL_CLK | AT91C_PMC_PRES_CLK_2;
//* Set default interrupts handler vectors
  AT91C_BASE_AIC->AIC_SVR[0]= (int) AT91F_Default_FIQ_handler;
  for (i=1;i < 31; i++)
  {
    AT91C_BASE_AIC->AIC_SVR[i]= (int) AT91F_Default_IRQ_handler;
  }
  AT91C_BASE_AIC->AIC_SPU= (int) AT91F_Spurious_handler;
}
```

3.6 Initialize ARM Mode Registers

Interrupt, and supervisor stacks are located at the top of RAM memory. Generally, abort-status, undefined instruction and user stacks are not used in a simple embedded system.

The C-startup code initializes the stack pointer registers. Depending on the interrupts and exceptions desired, some or all of the following stack pointers may require initialization:

- Supervisor stack must always be initialized.
- IRQ stack must be initialized if IRQ interrupts are used. It must be initialized before interrupts are enabled.
- FIQ Stack is not used by the standard AT91SAM7S FIQ Handler and does not need initialization. Only the FIQ register needs initialization.
- Abort-status stacks must be initialized if data and prefetch abort are handled.
- Undefined Instruction stack must be initialized if undefined instructions are handled.

Assuming that the IRQ handler is used, the interrupt stack requires 2 words x 8 priority level x 4 bytes when using the vectoring. The Interrupt Stack must be adjusted depending on the interrupt handlers. Other stacks are not defined to gain memory size.

```
;--------------------
;- Setup each mode
;--------------------
        RSEG   INTRAMEND_REMAP
#define  __iramend SFB(INTRAMEND_REMAP)
        ldr  r0, =__iramend
;- Set up Fast Interrupt Mode and set FIQ Mode Stack
        msr   CPSR_c, #ARM_MODE_FIQ | I_BIT | F_BIT
;- Init the FIQ register
        ldr  r8, =AT91C_BASE_AIC
;- Set up Interrupt Mode and set IRQ Mode Stack
        msr    CPSR_c, #ARM_MODE_IRQ | I_BIT | F_BIT
        mov    r13, r0                      ; Init stack IRQ
        sub    r0, r0, #IRQ_STACK_SIZE
```

3.7 Change Processor Mode and Enable Interrupts

The initialization code can now enable interrupts if necessary, by clearing the interrupt disable bits in the CPSR. This is the earliest point that it is safe to enable interrupts. At this stage the processor is still in Supervisor Mode.

```
;---------------------------------------------------------------------------
-
;- Setup Application Operating Mode Enable the interrupts and set Stack
;---------------------------------------------------------------------------
-
        msr   CPSR_c, #ARM_MODE_SVC
        mov   r13, r0
```

3.8 Initialize Software Variable and Branch to Main Function

The next task is to initialize the data memory by entering a loop that writes zeroes into allocations used for data storage and code. This may seem superfluous, but there are two reasons for this:

1. In C language, any non-initialized variable is supposed to contain zero as an initial value.
2. This makes the program behavior reproducible, even if not all variables are initialized explicitly.
   – The table of initial values for the initialized variable (in the C language sense) is copied to the location in RAM where the variables are positioned.
   – The linker puts the initial values in the same order as the variables in RAM, thus a mere block copy is sufficient for this initialization.
   – The linker puts the RAM code source segment in the Embedded Flash area.

The initial values for any initialized variables and RAM code must be copied from Flash to RAM. All other variables must be initialized to zero. The "__Segment_init" function copy the corresponding embedded Flash code in the RAM area and initialize the data segments.

The C initialization is processed by an IAR setup function __segment _init, this function is included in the C-IAR library and can be used in Thumb or ARM instruction sets.

```
;-----------------------------------------------------
;- Branch on C initialization code (with interworking)
;-----------------------------------------------------
   EXTERN__segment_init
; Initialize segments.
; __segment_init is assumed to use
        ldr  r0,=__segment_init
        mov  lr, pc
        bx   r0
```

When the compiler compiles a function called main(), it generates a PUBLIC reference to the symbol main. The function main() should be a closed loop and should not return.

```
;-----------------------------------------------------
;- Branch on C code Main function (with interworking)
;-----------------------------------------------------
      EXTERN main
      ldr   r0,=main
      bx    r0
```

4. Revision History

Doc. Rev	Date	Comments	Change Request Ref.
6131A	04-Mar-05	First issue - Qualified	

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature