AVR32708: AVR32 UC3A and UC3B Flash JTAG Programming Algorithms

Features

- **Low level JTAG programming algorithms for UC3A and UC3B devices internal FLASH**

1. Introduction

The aim of this application note is to provide 3rd party programmer vendors, the JTAG programming specification for AT32UC3x products.

The following algorithms assume that programmer has knowledge of the IEEE®1149.1 JTAG norm as well as the information in UC3 product datasheet concerning JTAG and Service Access Bus sections. See section 5. Related Documents.

2. Product signature

Each UC3 product is identified by a unique JTAG signature made a 32-bit value. This signature is mentioned for each product in the product datasheet.

The following public (standard-defined) JTAG instruction IDCODE is used to read the JTAG signature.

The IDCODE is the default instruction loaded into the JTAG on power-up or after a JTAG reset.

This selects the 32-bit JTAG IDCODE register.

If the flash controller is in reset the ID code will be undefined.

Note that if the flash controller is in reset, the security bit in the device is set and JTAG instructions using the Service Access Bus will be blocked.

32-bit AVR® Microcontrollers

Application Note

32070A–AVR32–11/07

3. Programming via JTAG interface

3.1 Constants & addresses:

3.1.1 Flash controller addresses

The following addresses are used by the Flash controller. All offsets are relative to the base address allocated to this Flash controller.

For AT32UC3A & AT32UC3B devices, Flash controller mapping has changed with latest revision of silicon:

For AT32UC3 devices, the JTAG ID code is the following 0x<revision><part number>03F.

On AT32UC3A device for revision less than 7 (Revision H) and on AT32UC3B device for revision less than 3, the flash controller mapping is the following:

Register definition	Name	Address
Flash Control Register	FCR	0x00
Flash Command Register	FCMD	0x04
Flash Status Register	FSR	0x08
Flash General Purpose Fuses Register	FGPFR	0x0C

On AT32UC3A devices with revision equal or above 7 (Revision H) and on AT32UC3B devices with revision equal or above 3 (Revision C), the flash controller mapping is the following:

Register definition	Name	Address
Flash Control Register	FCR	0x00
Flash Command Register	FCMD	0x04
Flash Status Register	FSR	0x08
Flash General Purpose Fuses Register High	FGPFRHI	0x0C
Flash General Purpose Fuses Register Low	FGPFRLO	0x10

3.1.2 Flash controller constants:

Constant definition	Name	Value
Flash Status Ready Mask address	FSR_FRDY_MASK	0x00000001
Flash Status Ready Offset	FSR_FRDY_OFFSET	0
Flash Status Programming Enable Mask Address	FSR_PROGE_MASK	0x00000008
Flash Status Programming Enable Offset	FSR_PROGE_OFFSET	3
Flash Status Lock Enable Mask Address	FSR_LOCKE_MASK	0x00000004
Flash Status Lock Enable Offset	FSR_LOCKE_OFFSET	2
Flash Status Size Mask Address	FSR_FSZ_MASK	0x0000E000
Flash Status Size Offset	FSR_FSZ_OFFSET	13
Flash Command Mask Address	FCMD_FCMD_MASK	0x0000001F
Flash Command Offset	FCMD_FCMD_OFFSET	0
Flash Command Page Enable Mask Address	FCMD_PAGEN_MASK	0x00FFFF00
Flash Command Page Enable Offset	FCMD_PAGEN_OFFSET	8
Flash Command Key Mask Address	FCMD_KEY_MASK	0xFF000000
Flash Command Key Offset	FCMD_KEY_OFFSET	24
Flash GP Fuse Lock Mask Address	FGPFR_LOCK_MASK	0x0000FFFF
Flash GP Fuse Lock Offset	FGPFR_LOCK_OFFSET	0
Number of words in a FLASH page	WORDS_PER_PAGE	128
Number of bytes in a FLASH page	BYTES_PER_PAGE	WORDS_PER_PAGE * 4
User Page Offset	USER_PAGE_OFFSET	0x00800000
Write Protect command	WRITE_PROTECT_KEY	0xA5000000
Write Page command	CMD_WRITE_PAGE	1
Erase Page command	CMD_ERASE_PAGE	2
Clear Page Buffer command	CMD_CLEAR_PAGE_BUFFER	3
Lock Region command	CMD_LOCK_REGION	4
Unlock Region command	CMD_UNLOCK_REGION	5
Erase All command	CMD_ERASE_ALL	6
Write General Purpose Fuse Bit command	CMD_WRITE_GP_FUSE_BIT	7
Erase General Purpose Fuse Bit command	CMD_ERASE_GP_FUSE_BIT	8
Set Security Bit command	CMD_SET_SECURITY_BIT	9
Program General Purpose Fuse Byte command	CMD_PROGRAM_GP_FUSE_BYTE	10
Write User Page command	CMD_WRITE_USER_PAGE	13
Erase User Page command	CMD_ERASE_USER_PAGE	14

3.1.3 Low level Flash Memory functions:

The read & write memory access are made thru the JTAG interface: **ReadMemoryWord** & **WriteMemoryWord**.

getRegister(address) is a ReadMemoryWord access to the "address" location.

setRegister(address, value) is a WriteMemoryWord access to the "address" location.

3.1.3.1 WriteCommand(Command):

WriteCommand(Command):

```
{
    WriteMemoryWord(FCMD, Command);
}
```

3.1.3.2 WaitFlashReady:

WaitFlashReady:

```
{
    Timeout=1 Second
    while (Not Timeout)
    {
        uint32_t fsrReg = getRegister(FSR);
        // If LOCKE bit in FSR set
        if((fsrReg & FSR_LOCKE_MASK) >> FSR_LOCKE_OFFSET )
            Error: Programming of at least one locked lock region has happend since
                   the last read of FSR.
        // If PROGE bit in FSR set
        if((fsrReg & FSR_PROGE_MASK) >> FSR_PROGE_OFFSET )
            Error: An invalid command and/or bad keywords were written in the Flash
                   Command Register.

        // Read FRDY bit in FSR
        if ((fsrReg & FSR_FRDY_MASK) >> FSR_FRDY_OFFSET)
            return;    // FLASH ready for next operation
    }
}
```

3.1.3.3 ClearPageBuffer

```
ClearPageBuffer:
{
    uint32_t command = WRITE_PROTECT_KEY | CMD_CLEAR_PAGE_BUFFER;
    WaitFlashReady
    WriteCommand(Command);
    WaitFlashReady
}
```

3.1.3.4 GetInternalFlashSize

```
Unsigned int GetInternalFlashSize:
{
    uint32_t fsrReg = getRegister(FSR);
    unsigned int fsz = (fsrReg & FSR_FSZ_MASK) >> FSR_FSZ_OFFSET;
    unsigned int size = 0;
    switch (fsz)
    {
        case 0:    size = 32 * 1024;
                   break;
        case 1:    size = 64 * 1024;
                   break;
        case 2:    size = 128 * 1024;
                   break;
        case 3:    size = 256 * 1024;
                   break;
        case 4:    size = 384 * 1024;
                   break;
        case 5:    size = 512 * 1024;
                   break;
        case 6:    size = 768 * 1024;
                   break;
        case 7:    size = 1024 * 1024;
                   break;
        default:
                   Error: Unknown flash size
    }
    return size;
}
```

3.1.4 Unlock Region

```
UnlockRegion(unsigned int Offset, unsigned int Size)
{
    if (Offset >= USER_PAGE_OFFSET
      && Offset < USER_PAGE_OFFSET + BYTES_PER_PAGE)
        return; // the user page doesn't need unlocking
    if (Offset >= mDeviceSize || Offset+size > mDeviceSize)
        Error: Region to be unlock lies outside flash address space.
    int lastpagetounlock = ((Offset+Size) / BYTES_PER_PAGE)
    // compute start offset of page to write to
    uint32_t page = Offset & ~(BYTES_PER_PAGE - 1);
    int pagenr = ((Offset) / BYTES_PER_PAGE)
    while (pagenr <= lastpagetounlock)
    {
        uint32_t command = WRITE_PROTECT_KEY | CMD_UNLOCK_REGION;
        // include the correct page number in the command
        command |= ((pagenr << FCMD_PAGEN_OFFSET) & FCMD_PAGEN_MASK);
        // Unlocking page: pagenr
        WaitFlashReady
        WriteCommand(command);  // execute unlock page command
        WaitFlashReady
        page += BYTES_PER_PAGE;
        Offset = page;
        pagenr =  ((Offset) / BYTES_PER_PAGE)
    }
}
```

3.1.5 Unlock entire FLASH

Call the UnlockRegion function with Offset: 0 and total size of the device

```
UnlockEntireFlash:
{
    DeviceSize=GetInternalFlashSize;
    UnlockRegion(0, DeviceSize);
}
```

3.1.6 Erase sequence:

```
EraseSequence:
{
    WaitForFlashReady
    Command=WRITE_PROTECT_KEY | CMD_ERASE_ALL
    WriteCommand(Command):
    WaitForFlashReady
}
```

3.1.7 Erase User Page sequence:

```
EraseUserPage:
{
    uint32_t command = WRITE_PROTECT_KEY | CMD_ERASE_USER_PAGE;
    WaitFlashReady
    WriteCommand(command);  // execute user page erase command
    WaitFlashReady
}
```

3.1.8 Erase region sequence:

```
EraseRegionSequence(Offset, Size):
{
    if (Offset >= USER_PAGE_OFFSET
       && Offset < USER_PAGE_OFFSET + BYTES_PER_PAGE)
            EraseUserPage

    lastpagetoerase =(Offset+Size) / BYTES_PER_PAGE
    page = Offset & ~(BYTES_PER_PAGE - 1) // compute start offset of page to write to
    pagenr = (Offset) / BYTES_PER_PAGE
    while (pagenr <= lastpagetoerase)
    {
            Command = WRITE_PROTECT_KEY | CMD_ERASE_PAGE
            Command |= ((pagenr << FCMD_PAGEN_OFFSET) & FCMD_PAGEN_MASK)
            // include the correct page number in the command
            WaitFlashReady
            WriteCommand(Command)  // execute page erase command
            WaitFlashReady

            page += BYTES_PER_PAGE;
            Offset = page;
            pagenr = (Offset) / BYTES_PER_PAGE
    }
}
```

3.1.9 Programming sequences:

3.1.9.1 Program User Page sequence:

```
ProgramUserPage(Offset - USER_PAGE_OFFSET, DataBuffer):
{
    if (Offset >= BYTES_PER_PAGE || Offset + Length(DataBuffer) > BYTES_PER_PAGE)
        Error: Tried to program past user page boundary

    // Packet bufferPacket(BYTES_PER_PAGE)  define a buffer packet
    // to manipulate the data

    // If the packet to be written is smaller than the user page we fill the
    // remaining space with existing data

    if (Offset > 0 || Length(DataBuffer) < BYTES_PER_PAGE)
        ReadMemory(mBaseAddress + USER_PAGE_OFFSET, bufferPacket, 0);
    // Must clear the page buffer before writing to it.
    ClearPageBuffer();

    int bytesLeftInPacket = Length(DataBuffer);
    int i = 0;  // data packet index
    // Fill buffer packet
    while (bytesLeftInPacket > 0)
    {
        bufferPacket.writeSingleByte(Offset++, ReadSingleByte(DataBuffer)
        i++;
        bytesLeftInPacket--;
    }
    // Write page buffer
    WriteMemory(mBaseAddress + USER_PAGE_OFFSET, bufferPacket);
    uint32_t command = WRITE_PROTECT_KEY | CMD_WRITE_USER_PAGE;
    WaitFlashReady
    WriteCommand(command);  // execute user page write command
    WaitFlashReady
}
```

3.1.9.2 Program sequence:

```
ProgramSequence(Offset, DataBuffer):
{
    if (Offset >= USER_PAGE_OFFSET
      && Offset < USER_PAGE_OFFSET + BYTES_PER_PAGE)
        ProgramUserPage(Offset - USER_PAGE_OFFSET, DataBuffer)

    if (Offset >= mDeviceSize || Offset+Length(DataBuffer) > mDeviceSize)
        Error: Region to be programmed lies outside flash address space.

    // compute start offset of page to write to
    uint32_t page = Offset & ~(BYTES_PER_PAGE - 1);
    unsigned int bytesLeft = (Length(DataBuffer));
    int dataOffset = 0; // current offset in the data packet
    Packet bufferPacket(BYTES_PER_PAGE);    // we write one page at a time

    // Loop until all bytes in data has been written
    while (bytesLeft > 0)
    {
        bufferPacket.clear(0xff);

        // Must clear the page buffer before writing to it.
        ClearPageBuffer();

        /* Keeps track of how many bytes to write to the bufferPacket.
        * If the start offset is not aligned on a page boundary, we will not fill
        * the bufferPacket completely. This is also the case when the number of
        * bytes left to write is less than the size of a page. If the bufferPacket
        * is not filled completely we first read the current flash content into
        * the packet. This way we will always preserve existing flash data
        * adjacent to the new data we wish to write.
        */
        int bytesLeftInPacket = min((page+BYTES_PER_PAGE-Offset), bytesLeft);
        int bufferOffset = Offset % BYTES_PER_PAGE;
        if (bufferOffset != 0 || bytesLeftInPacket != (BYTES_PER_PAGE))
        {
            ReadMemory(mBaseAddress + page, bufferPacket, 0);
        }
        for (int i = 0; i < bytesLeftInPacket; ++i)
        {
            bufferPacket.writeSingleByte(bufferOffset++,
                                         ReadSingleByte(DataBuffer));

            Offset++;
```

```
                }
                WriteMemory(mBaseAddress + page, bufferPacket);
                int pagenr =  ((Offset) / BYTES_PER_PAGE)
                uint32_t command = WRITE_PROTECT_KEY | CMD_WRITE_PAGE;
                // include the correct page number in the command
                command |= pagenr << FCMD_PAGEN_OFFSET;
                WaitFlashReady
                WriteCommand(command);  // execute page write command
                WaitFlashReady

                page += BYTES_PER_PAGE;
                Offset = page;
                bytesLeft -= bytesLeftInPacket;
            }
        }
```

3.1.10 Fuses management sequences:

3.1.10.1 Get General Purpose Fuse bits

```
bool GetGeneralPurposeFuseBit(int Index)
{
        if (Index > 31 || Index < 0)
                Error: Index out of range

        uint32_t fgpfrReg = getRegister(FGPFR);
        return fgpfrReg & (1 << Index);
}
```

Note: FGPFR register becomes FGPFRHI or FGPFRLO for new silicon revisions, see 3.1.1 Flash controller addresses chapter.

3.1.10.2 Set General Purpose Fuse bits

```
SetGeneralPurposeFuseBit(int Index, bool Value):
{
        if (Index > 31 || Index < 0)
                Error:  Index out of range.

        uint32_t command = WRITE_PROTECT_KEY | (index << FCMD_PAGEN_OFFSET);
        if (Value)  // erase bit
                command |= CMD_ERASE_GP_FUSE_BIT;
        else       // program bit
                command |= CMD_WRITE_GP_FUSE_BIT;
        WaitFlashReady
        WriteCommand(command);
        WaitFlashReady
}
```

3.1.10.3 *Set General Purpose Fuse Byte*

```
SetGeneralPurposeFuseByte(int Index, unsigned char Value):
{
    if (Index > 3 || Index < 0)
        Error: Index out of range

    uint32_t command = WRITE_PROTECT_KEY | CMD_PROGRAM_GP_FUSE_BYTE
                     | Index << FCMD_PAGEN_OFFSET
                     | Value << (FCMD_PAGEN_OFFSET + 2);

    WaitFlashReady
    WriteCommand(command);
    WaitFlashReady
}
```

3.1.11 Security management sequences:

```
SetSecurityBit()
{
    uint32_t command = WRITE_PROTECT_KEY | CMD_SET_SECURITY_BIT;
    WaitFlashReady
    WriteCommand(command)
    // Wait at least 2 seconds
    Wait(2s)
}
```

4. Limitations

None

5. Related Documents

- IEEE1149.1 JTAG Norm
- AT32UC3A and AT32UC3B product datasheets: JTAG & Service Access Bus chapter

Headquarters

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

International

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

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel 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

Product Contact

Web Site
www.atmel.com

Technical Support
Enter Product Line E-mail

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature