Freescale Semiconductor
Application Note

Document Number: AN4389
Rev. 0, 11/2011

PXS30 Self Test Control Unit (STCU) Reset Configuration Data in Shadow Flash

by: David Connelly
Microcontroller Solutions Group

1 Introduction

PXS30 device has a block of logic called the Self Test Control Unit (STCU) which handles built in self test (BIST) at reset. BIST can be enabled or disabled depending on certain data values in the shadow flash configuration fields, which are sampled at reset to determine the self test startup sequence. This Application note explains the data values and locations for the STCU reset configuration and how to modify them to enable or disable the self-test at startup.

2 Reset Options and Default Device Configuration

The default configuration for PXS30 bypasses the internal self test at reset. On other devices such as PXS20, have the shadow flash configuration data fields programmed with values that enable self test by default. For safety critical applications, enabling self test at reset is a requirement to meet certain safety levels. Any faults

Contents

detected are routed to a central Fault Collection and Control Unit (FCCU) for error handling. If the self test startup is bypassed, it can reduce power up current and shorten total reset time. The configuration chosen should be based on the targeted application.

To change the default configuration, the shadow block has to be erased and re-programmed with new data. The flash erase and programming sequences have to be followed to accomplish this, which are documented in the PXS30 Reference Manual available at www.freescale.com. Some examples on how to erase and program are given in Appendix A.

Prior to modifying the shadow flash, special care has to be taken to preserve the censorship and security pass code, and the user option bits (Table 2 and Table 3) which also exist in the shadow flash. In addition there are reset words that must be restored after the shadow flash is erased. If the erase or program operation fails and the censorship or security pass code are not correctly programmed then the device may become locked and unusable. Basically, the shadow block should not be erased without understanding all of the critical information that has to be re-programmed before a reset is applied. More information on the reset vector and flash boot code is available in the PXS30 Reference Manual.

To summarize recommended steps to update STCU fields in shadow flash:

1. Read entire shadow flash space and modify STCU fields accordingly (see Section Appendix A, “Erase and Program Shadow Flash Sample Code for detail).
2. Erase shadow flash
3. Program new STCU Data and reset of shadow flash configuration fields
4. Verify newly programmed data
5. Apply Power On Reset

NOTE

There are two shadow blocks on PXS30, the second of which does not contain any vital boot, STCU, or censorship information. The second shadow block can be used to debug the erase and program steps. This way if a mistake occurs, the device will not be locked. Once the program is debugged on the secondary shadow block, then the base address of the erase and program operations can be updated to the main shadow flash to update the STCU fields.

Valid STCU reset data can be seen in Table 1, 2, and 3 below. The addresses shown are located in shadow flash block for STCU bypassed, STCU enabled, and STCU enabled-stay-in-reset.

Table 1. Shadow Data for Self Test Startup Bypassed

STCU BYPASS – Shadow Block Base Address 0x0020_0000				
Offset VALID SHADOW SCAN DATA	**0x10** 0x05AA55AF	**0x14** 0x00000000	**0x18** FFFFFFFF	**0x1C** 0xFFFFFFFF
STCU BYPASS	**0x20** 0x00000100	**0x24** 0x0008000C	**0x28** 0xFFFFFFFF	**0x2C** 0xFFFFFFFF
STOP WORD	**0x30** 0x00000000	**0x34** 0x00000001	**0x38** 0xFFFFFFFF	**0x3C** 0xFFFFFFFF

Note: Data in location 0x40-0xA0 is 0xFFFFFFFF

Table 2. Shadow Data for Self Test Startup Enabled, Critical Fault Mapping

STCU ENABLE- Shadow Block Base Address 0x0020_0000				
Offset VALID SHADOW SCAN DATA	**0x10** 0x05AA55AF	**0x14** 0x00000000	**0x18** FFFFFFFF	**0x1C** 0xFFFFFFFF

Note: Data in location 0x20-0xA0 is 0xFFFFFFFF

Note: STCU register configuration comes from Test Flash exclusively

Table 3. Shadow Data for Self Test Startup Enabled, non-Critical Fault Mapping

STCU ENABLE - Non-Critical Fault Mapping - Base Address 0x0020_0000				
Offset VALID SHADOW SCAN DATA	**0x10** 0x05AA55AF	**0x14** 0x00000000	**0x18** FFFFFFFF	**0x1C** 0xFFFFFFFF
MBIST0 selected	**0x20** 0x10000000	**0x24** 0x0008000C	**0x28** 0xFFFFFFFF	**0x2C** 0xFFFFFFFF
Expected CRC	**0x30** 0x3371802D	**0x34** 0x00080014	**0x38** 0xFFFFFFFF	**0x3C** 0xFFFFFFFF
LBIST SIR Key	**0x40** 0xFE35AB20	**0x44** 0x00080038	**0x48** 0xFFFFFFFF	**0x4C** 0xFFFFFFFF
LBIST Critical FM - set to Non-Critical	**0x50** 0x00000000	**0x54** 0x00080030	**0x58** 0xFFFFFFFF	**0x5C** 0xFFFFFFFF
MBIST Key	**0x60** 0x751AC490	**0x64** 0x00080060	**0x68** 0xFFFFFFFF	**0x6C** 0xFFFFFFFF
MBIST Critical FM - set to Non-Critical	**0x70** 0x00000000	**0x74** 0x00080050	**0x78** 0xFFFFFFFF	**0x7C** 0xFFFFFFFF
MBIST Critical FM - set to Non-Critical	**0x80** 0x00000000	**0x84** 0x00080054	**0x88** 0xFFFFFFFF	**0x8C** 0xFFFFFFFF
STOP WORD	**0x90** 0xFFFFFFFF	**0x94** 0xFFFFFFFF	**0x98** 0xFFFFFFFF	**0x9C** 0xFFFFFFFF

Note: STCU register configuration loaded from Shadow Flash

Table 4. Shadow Data for Self Test Startup Enabled, non-Critical Fault Mapping (Device Stay in Reset upon fault)

STCU ENABLE - Stay In Reset - Shadow Block Base Address 0x0020_0000				
Offset VALID SHADOW SCAN DATA	**0x10** 0x05AA55AF	**0x14** 0x00000000	**0x18** FFFFFFFF	**0x1C** 0xFFFFFFFF
MBIST0 selected	**0x20** 0x10000000	**0x24** 0x0008000C	**0x28** 0xFFFFFFFF	**0x2C** 0xFFFFFFFF
Expected CRC	**0x30** 0xE2384E33	**0x34** 0x00080014	**0x38** 0xFFFFFFFF	**0x3C** 0xFFFFFFFF
LBIST SIR Key	**0x40** 0xFE35AB20	**0x44** 0x00080038	**0x48** 0xFFFFFFFF	**0x4C** 0xFFFFFFFF
LBIST Critical FM - set to Non-Critical	**0x50** 0x00000007	**0x54** 0x00080034	**0x58** 0xFFFFFFFF	**0x5C** 0xFFFFFFFF
MBIST Key	**0x60** 0x751AC490	**0x64** 0x00080060	**0x68** 0xFFFFFFFF	**0x6C** 0xFFFFFFFF
MBIST Critical FM - set to Non-Critical	**0x70** 0xFFFFFFFF	**0x74** 0x00080058	**0x78** 0xFFFFFFFF	**0x7C** 0xFFFFFFFF
MBIST Critical FM - set to Non-Critical	**0x80** 0x03FFFFFF	**0x84** 0x0008005C	**0x88** 0xFFFFFFFF	**0x8C** 0xFFFFFFFF
STOP WORD	**0x90** 0x00000000	**0x94** 0x00000001	**0x98** 0xFFFFFFFF	**0x9C** 0xFFFFFFFF

Censorship and Serial Password exist in the shadow flash and also have to be preserved. The locations are shown below.

Table 5. Censorship & Serial Passcode – Shadow block Base Address 0x0020_0000

Offset Censorship	**0x3DD0** 0xFFFFFFFF	**0x3DD4** 0xFFFFFFFF	**0x3DD8** 0xFEEDFACE	**0x3DDC** 0xCAFEBEEF
Serial Password	**0x3DE0** 0x55AA55AA	**0x3DE4** 0x55AA55AA	**0x3DE8** 0xFFFFFFFF	**0x3DEC** 0xFFFFFFFF

There is a 32-bit User Option word located at 0x3E18 offset in the shadow flash, and given below are a few examples for different configurations. For further detail, please refer to Section 49.3.1.8 User Option Status (SSCM_UOPS) Register in the PXS30 Reference Manual available at www.freescale.com.

User Option Word in Shadow Flash – Examples

Table 6. Reset Configuration Word – DPM & Watchdog Enabled – Shadow block Base Address 0x0020_0000

Offset User Option	**0x3E10** 0xFFFFFFFF	**0x3E14** 0xFFFFFFFF	**0x3E18** **0xFFBFFFFF**	**0x3E1C** 0xFFFFFFFF

Table 7. DPM and Watchdog Disabled

Offset User Option	**0x3E10** 0xFFFFFFFF	**0x3E14** 0xFFFFFFFF	**0x3E18** **0x7FBFFFFF**	**0x3E1C** 0xFFFFFFFF

Table 8. Reset Configuration Word – LSM & Watchdog Enabled – Shadow block Base Address 0x0020_0000

Offset User Option	**0x3E10** 0xFFFFFFFF	**0x3E14** 0xFFFFFFFF	**0x3E18** **0xFFFFFFFF**	**0x3E1C** 0xFFFFFFFF

Table 9. Reset Configuration Word – LSM & Watchdog Disabled – Shadow block Base Address 0x0020_0000

Offset User Option	**0x3E10** 0xFFFFFFFF	**0x3E14** 0xFFFFFFFF	**0x3E18** **0x7FFFFFFF**	**0x3E1C** 0xFFFFFFFF

NOTE

Changing from DPM mode, which is the default configuration, to LSM mode will change the overall system memory map.

3 Revision history

Table 10. Changes made April 2012[1]

Section	Description
Front page	Add SafeAssure branding.
Back page	Apply new back page format.

1 No substantive changes were made to the content of this document; therefore the revision number was not incremented.

Appendix A Erase and Program Shadow Flash Sample Code

Example 1 – Self Test Startup Enabled

```
/**** global ****/
uint32_t  shadow_data[BLK_16KB / WORD_SIZE_IN_BYTE];

/*********************************************************************/
/* Fnct: SetSTCUEnable_Shadow                                        */
/* Desc: This Function will program STCU bypass data to shadow block */
/* Input:  none                                                      */
/* Return: PASS or FAIL                                              */
/*********************************************************************/
uint32_t SetSTCUEnable_Shadow()
{
        uint32_t addr, i, status;

        /*********************************/
        /* Get All Data From Shadow block  */
        /*********************************/
        for (i = 0, addr = SHADOW_BASE; addr < (SHADOW_BASE + SHADOW_SIZE); addr+=4, i++)
        {
                shadow_data[i] = ReadAddrLong(addr);
        }

        /*****************************************/
```

```
        /* Change stcu data read from shadow block*/
        /* Configure for stcu bypass        */
        /* Re-program on the next step      */
        /*******************************************/
        i = (STCU_OFFSET / 4); // stcu_data[0] location - index into array of 32-bit values

        #if 1
                /* Cut 2 Enable BIST, critical fault mapping */
                shadow_data[i] = 0x05AA55AF;//SHADOW_BASE + 0x10
                shadow_data[i+1] = 0x00000000;
                for (i = (i + 2); i < 45; i++)
                {
                        shadow_data[i] = 0xFFFFFFFF;
                }
                /* Reset the device, program MMU, and Verify STCU register contents: */
                /*
                        RUN = 0
                        CFG.BYP = 0
                        ERR = 0x0000_0000
                        LBS = 0x0000_0007
                        LBE = 0x0000_0007
                        MBSL = 0xFFFF_FFFF
                        MSBH = 0x03FF_FFFF
                        MBEL = 0xFFFF_FFFF
                        MBEH = 0x03FF_FFFF
                        CRCR, CRCE = 0xB3C5BF31  */

        #else
                /* Cut 2 Enable, Non-critical fault mapping (optional) */
                shadow_data[i] = 0x05AA55AF;//Valid Shadow Scan data - SHADOW_BASE + 0x10
                shadow_data[i+1] = 0x00000000;
                shadow_data[i+2] = 0xFFFFFFFF;
                shadow_data[i+3] = 0xFFFFFFFF;
                shadow_data[i+4] = 0x10000000;//MBIST0 Selected - SHADOW_BASE + 0x20
                shadow_data[i+5] = 0x0008000C;
                shadow_data[i+6] = 0xFFFFFFFF;
                shadow_data[i+7] = 0xFFFFFFFF;
                shadow_data[i+8] = 0x3371802D;//Expected CRC - SHADOW_BASE + 0x30 (updated
for cut2)
                shadow_data[i+9] = 0x00080014;
                shadow_data[i+10] = 0xFFFFFFFF;
                shadow_data[i+11] = 0xFFFFFFFF;
                shadow_data[i+12] = 0xFE35AB20;//LBIST Key - SHADOW_BASE + 0x40
                shadow_data[i+13] = 0x00080038;
                shadow_data[i+14] = 0xFFFFFFFF;
                shadow_data[i+15] = 0xFFFFFFFF;
                shadow_data[i+16] = 0x00000000; //LBIST Critical FM Register, set to
non-critical fault - SHADOW_BASE + 0x50
                shadow_data[i+17] = 0x00080030;
                shadow_data[i+18] = 0xFFFFFFFF;
                shadow_data[i+19] = 0xFFFFFFFF;

                shadow_data[i+20] = 0x751AC490;//MBIST Key - SHADOW_BASE + 0x60
                shadow_data[i+21] = 0x00080060;
                shadow_data[i+22] = 0xFFFFFFFF;
                shadow_data[i+23] = 0xFFFFFFFF;
```

```
                shadow_data[i+24] = 0x00000000; //MBIST Critical FM Low register, set to
non-critical fault - SHADOW_BASE + 0x70
                shadow_data[i+25] = 0x00080050;
                shadow_data[i+26] = 0xFFFFFFFF;
                shadow_data[i+27] = 0xFFFFFFFF;
                shadow_data[i+28] = 0x00000000;//MBIST Critical FM High register, set to
non-critical fault - SHADOW_BASE + 0x80
                shadow_data[i+29] = 0x00080054;
                shadow_data[i+30] = 0xFFFFFFFF;
                shadow_data[i+31] = 0xFFFFFFFF;
                shadow_data[i+32] = 0xFFFFFFFF;//STOP Word (all blank) - SHADOW_BASE + 0x90
                shadow_data[i+33] = 0xFFFFFFFF;
                shadow_data[i+34] = 0xFFFFFFFF;
                shadow_data[i+35] = 0xFFFFFFFF;
                for (i = (i + 36); i < 45; i++)
                {
                        shadow_data[i] = 0xFFFFFFFF;
                }
                /* Reset the device, program MMU, and Verify STCU register contents */
                /*
                RUN = 0
                CFG.BYP = 0
                ERR = 0x0000_0000
                LBS = 0x0000_0007
                LBE = 0x0000_0007
                MBSL = 0xFFFF_FFFF
                MSBH = 0x03FF_FFFF
                MBEL = 0xFFFF_FFFF
                MBEH = 0x03FF_FFFF
                CRCR, CRCE = 0x337180D2
                */


                #if 0
                        /* Cut 2 Enable STCU Stay in Reset (optional) */
                        shadow_data[i] = 0x05AA55AF;//Valid Shadow Scan data - SHADOW_BASE +
0x10
                        shadow_data[i+1] = 0x00000000;
                        shadow_data[i+2] = 0xFFFFFFFF;
                        shadow_data[i+3] = 0xFFFFFFFF;
                        shadow_data[i+4] = 0x10000000;//MBIST0 Selected - SHADOW_BASE + 0x20
                        shadow_data[i+5] = 0x0008000C;
                        shadow_data[i+6] = 0xFFFFFFFF;
                        shadow_data[i+7] = 0xFFFFFFFF;
                        shadow_data[i+8] = 0xE2384E33;//Expected CRC - SHADOW_BASE + 0x30
                        shadow_data[i+9] = 0x00080014;
                        shadow_data[i+10] = 0xFFFFFFFF;
                        shadow_data[i+11] = 0xFFFFFFFF;
                        shadow_data[i+12] = 0xFE35AB20;//LBIST Stay In Reset Key -
SHADOW_BASE + 0x40
                        shadow_data[i+13] = 0x00080038;
                        shadow_data[i+14] = 0xFFFFFFFF;
                        shadow_data[i+15] = 0xFFFFFFFF;
                        shadow_data[i+16] = 0x00000007; //LBIST Stay in Reset - SHADOW_BASE
+ 0x50
                        shadow_data[i+17] = 0x00080034;
                        shadow_data[i+18] = 0xFFFFFFFF;
```

```
                        shadow_data[i+19] = 0xFFFFFFFF;

                        shadow_data[i+20] = 0x751AC490;//MBIST Stay In Reset Key -
SHADOW_BASE + 0x60
                        shadow_data[i+21] = 0x00080060;
                        shadow_data[i+22] = 0xFFFFFFFF;
                        shadow_data[i+23] = 0xFFFFFFFF;
                        shadow_data[i+24] = 0xFFFFFFFF; //MBIST Stay in Reset - SHADOW_BASE
+ 0x70
                        shadow_data[i+25] = 0x00080058;
                        shadow_data[i+26] = 0xFFFFFFFF;
                        shadow_data[i+27] = 0xFFFFFFFF;
                        shadow_data[i+28] = 0xFFFFFFFF;//Stop Word - SHADOW_BASE + 0x80
                        shadow_data[i+29] = 0xFFFFFFFF;
                        shadow_data[i+30] = 0xFFFFFFFF;
                        shadow_data[i+31] = 0xFFFFFFFF;
                        for (i = (i + 32); i < 45; i++)
                        {
                                shadow_data[i] = 0xFFFFFFFF;
                        }
                        /* Reset the device, program MMU, and Verify STCU register contents:
*/
                        /*
                                RUN = 0
                                CFG.BYP = 0
                                ERR = 0x0000_0000
                                LBS = 0x0000_0007
                                LBE = 0x0000_0007
                                MBSL = 0xFFFF_FFFF
                                MSBH = 0x03FF_FFFF
                                MBEL = 0xFFFF_FFFF
                                MBEH = 0x03FF_FFFF
                                CRCR, CRCE = 0xE2384E33 */

                #endif //#if 0 - stay in reset

        #endif

        /* Unlock Flash and enable for erase/pgm */
        status = pFlashInit(&ssdConfig);
        FlashSetup(ptrFlash);

        /* DO NOT STOP DEBUGGER DURING ERASE OR PROGRAM OR THE DEVICE MAY BECOME LOCKED !!! */
        if (status == PASS)
        {
                /* Erase and Re-program new data */
                status = EraseShadow(SHADOW_BASE);
                status += ProgramShadow(SHADOW_BASE, shadow_data);
        }

        return(status);
}
```

Example 2– Self Test Startup Bypassed

```
/**** global ****/
uint32_t  shadow_data[BLK_16KB / WORD_SIZE_IN_BYTE];
```

```
/*********************************************************************/
/* Fnct: SetSTCUBypass_Shadow                                        */
/* Desc: This Function will program STCU bypass data to shadow block */
/* Input:  none                                                      */
/* Return: PASS or FAIL                                              */
/*********************************************************************/
uint32_t SetSTCUBypass_Shadow()
{
        uint32_t addr, i, status;

        /***********************************/
        /* Get All Data From Shadow block  */
        /***********************************/
        for (i = 0, addr = SHADOW_BASE; addr < (SHADOW_BASE + SHADOW_SIZE); addr+=4, i++)
        {
                shadow_data[i] = ReadAddrLong(addr);
        }

        /*****************************************/
        /* Change stcu data read from shadow block*/
        /* Configure for stcu bypass        */
        /* Re-program on the next step      */
        /*****************************************/
        i = (STCU_OFFSET / 4); // stcu_data[0] location - index into array of 32-bit values
        #if 1
                /* Cut 2 Bypass - latest version */
                shadow_data[i] = 0x05AA55AF;//SHADOW_BASE + 0x10
                shadow_data[i+1] = 0x00000000;
                shadow_data[i+2] = 0xFFFFFFFF;
                shadow_data[i+3] = 0xFFFFFFFF;
                shadow_data[i+4] = 0x00000100;//SHADOW_BASE + 0x20
                shadow_data[i+5] = 0x0008000C;
                shadow_data[i+6] = 0xFFFFFFFF;
                shadow_data[i+7] = 0xFFFFFFFF;
                for (i = (i + 8); i < 45; i++)//STOP Word - blank data
                {
                        shadow_data[i] = 0xFFFFFFFF;
                }
        #else
                /* Cut 1 Bypass - old silicon */
                shadow_data[i] = 0x05AA55AF;//SHADOW_BASE + 0x10
                shadow_data[i+1] = 0x00000000;
                shadow_data[i+2] = 0xFFFFFFFF;
                shadow_data[i+3] = 0xFFFFFFFF;
                shadow_data[i+4] = 0x00000001;//SHADOW_BASE + 0x20
                shadow_data[i+5] = 0x00080000;
                shadow_data[i+6] = 0xFFFFFFFF;
                shadow_data[i+7] = 0xFFFFFFFF;
                shadow_data[i+8] = 0x00000100;//SHADOW_BASE + 0x30
                shadow_data[i+9] = 0x0008000C;
                shadow_data[i+10] = 0xFFFFFFFF;
                shadow_data[i+11] = 0xFFFFFFFF;
                shadow_data[i+12] = 0x00000000;//SHADOW_BASE + 0x40
                shadow_data[i+13] = 0x00000001;
                for (i = (i + 14); i < 45; i++)
                {
                        shadow_data[i] = 0xFFFFFFFF;
```

```
                }
        #endif

        /* Unlock Flash and enable for erase/pgm */
        status = pFlashInit(&ssdConfig);
        FlashSetup(ptrFlash);

        /* DO NOT STOP DEBUGGER DURING ERASE OR PROGRAM OR THE DEVICE MAY BECOME LOCKED !!! */
        if (status == PASS)
        {
                /* Erase and Re-program new data */
                status = EraseShadow(SHADOW_BASE);
                status += ProgramShadow(SHADOW_BASE, shadow_data);
        }

        return(status);
}
/*************************************************************************/
//      Fnct:    FlashSetup                                              //
//      Desc:    This function will write pass code to each of the block protect //
//                       registers to enable block select registers.     //
//                                                                       //
//      Parms:   ptrFlash  - pointer to source control regsiters to be unlocked//
//                                                                       //
//      Return:  None                                                    //
/*************************************************************************/
void FlashSetup(psFLASH *ptrFlash)
{
        uint8_t module;

        /* Code Flash */
        for(module = 0; module < NUM_MODULES; module++)
        {
                ptrFlash[module]->LML.R = FLASH_LMLR_PASSWORD;// unlock LML Register
                ptrFlash[module]->LML.R = 0x80100000;   // Set LME = 1, SLOCK = 1
                ptrFlash[module]->HBL.R = FLASH_HLR_PASSWORD;// unlock HBL Register
                ptrFlash[module]->HBL.R = 0x80000000;   // Set HBE = 1
                ptrFlash[module]->SLL.R = FLASH_SLL_PASSWORD;// unlock SLL Register
                ptrFlash[module]->SLL.R = 0x80100000;   // Set SLE = 1, SSLOCK = 1

                ptrFlash[module]->MCR.R  = MCR_EER | MCR_RWE | MCR_EDC;// Clear EER, RWE and
SBC, if any.
                ptrFlash[module]->UT0.R = FLASH_UT_PASSWORD;// Enable user-test mode
                ptrFlash[module]->UT0.R |= UT0_SBCE;// Enable Single Bit Correction Reporting
                ptrFlash[module]->UT0.R &= ~(UT0_UTE);// Disable UTE
        }

        /* Data Flash */
        ptrFlash_D->LML.R = FLASH_LMLR_PASSWORD;// unlock LML Register
        ptrFlash_D->LML.R = 0x80100000;   // Set LME = 1, TSLK = 1, unlock Low/Mid Blocks
        ptrFlash_D->SLL.R = FLASH_SLL_PASSWORD;// unlock SLL Register
        ptrFlash_D->SLL.R = 0x80100000;   // Set SLE = 1, SSLOCK = 1
        ptrFlash_D->MCR.R  = MCR_EER | MCR_RWE | MCR_EDC;// Clear EER, RWE and SBC, if any.

}
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4389
Rev. 0
11/2011