SMX® STMicro STM32746G-DISC Board Support Notes
IAR

September 29, 2017
XiaoYi Chen

Summary

The IAR STM32746G-DISCOVERY board is built around a 32-bit STMicro STM32F746NG Cortex-M7 microcontroller. Cortex-M7 is an implementation of the ARM-M architecture, which is significantly different from the traditional ARM design (used by ARM7, 9, etc). **Please read the ARM-M section of the SMX Target Guide** for information about this architecture.

The BSP code consists of BSP\ARM\bspm.c and some other files in the directories BSP\ARM\STM32 and BSP\ARM\STM32\STM32F7xx. In BSPs for other processors, we keep bsp.c in the processor or board directory, but since the features it handles are all standard in the ARM-M core, we can share one file, bspm.c, for all ARM-M processors, even from different vendors. Sharing the file saves duplicating the code many times. The files in the STMicro subdirectory are from STMicro's firmware library. Only some of the files are used; add others to the App project, as needed. All changes we made are tagged "MDI:".

Notes:

1. There is a separate prefix file (.h) in the CFG directory for each board. For IAR, the one to use is selected in **iararm.h**.

   Testing was done on this board:

   - STM32746G-DISC. CPU STM32F746, NGH6U, 7BA3D VQ Z, PHL 7B 512

2. All project files for this processor add the compiler switch **--no_unaligned_access** (see Extra Options tab in compiler settings) to Debug and Release build targets, to avoid processor faults on access to unaligned data in SDRAM. It tells the compiler not to generate unaligned data. The problem is caused because STMicro located SDRAM in the memory range for Device Memory, which requires 4-byte alignment. This switch has to be used in **all** project files for it to be effective. It is used for boards without SDRAM too, in case the project is cloned for another board that does have it.

Jumper and Switch Settings:

- JP1 set to 5V Ext and provide external 5V 1A power through JP2. Internal power sources don't provide enough current to drive USBH; it causes transmit errors in the USBH controller.

IRQ Map

IRQ	Description	Interrupt Priority Level	Notes
Int 15	SysTick	0x20 or 0x00	smx tick; all timing based on this
59	SDIO DMA2 Stream3	0x40	smxFS
61	Ethernet	0x40	smxNS
77	USB OTG HS	0x80	smxUSBD/smxUSBH

Notes:

1. SysTick has no IRQ number because it precedes the range of IRQs. IRQ0 is Int 16. SysTick does not appear in irq_table[] and is hooked and configured specially, not using the usual BSP functions.
2. The STM32 processors support only 4 bits of priority levels (starting at the MSB), so there are only 16 priority levels: 0x00, 0x10, 0x20, … 0xF0. Lower number is higher priority. Interrupts with the same priority level are prioritized by their IRQ number. The lower numbered IRQ has higher priority.
3. ISRs that use smx_ISR_ENTER/smx_ISR_EXIT should not be set to a lower value (higher priority) than SB_ARMM_BASEPRI_VALUE (XBASE\barmm.h) because such level(s) are non-maskable and such ISRs must be maskable. These lower level(s) are reserved for special ISRs by using BASEPRI rather than PRIMASK to disable interrupts. SB_ARMM_DISABLE_WITH_BASEPRI in XBASE\barmm.h selects which method is used. See section Architectural Notes/ ISR Priority Level in the SMX Target Guide.
4. Interrupt priorities can be changed as desired. However, the priority of SysTick generally should be highest maskable level since all smx timing depends on it.

Memory Map (Debug, Release, and ROM targets — Internal Flash, Internal SRAM)

Base Address	Size	Memory	Sections	Notes
0x00000000	1MB	Internal Flash	.intvec, .text, .rodata	Alias of 0x00200000 or 0x08000000
0x00200000	1MB	Internal Flash (ITCM)	.intvec, .text, .rodata	Code and Read Only data. Faster bus.
0x08000000	1MB	Internal Flash (AXIM)	.intvec, .text, .rodata	Code and Read Only data. Slower bus.
0x20000000	64KB	Internal DTCM SRAM	SDAR, CSTACK, intvec	EVT, SDAR and System Stack, .intvec (EVT) copied here by startup code. See note about DTCM in Memory Configuration section.
0x20010000	256KB	Internal SRAM	.noinit, .bss, .data, ADAR, EMAC_BUFFER	Data and ADAR. smxNS frame buffers.
0x60000000	16MB	External NOR Flash		Quad SPI Flash Memory
0xC0000000	32MB	External SDRAM	LCD Buffer	LCD Buffer

Notes:

1. This uses the linker command file **stm32746gdis_irom_iram.icf** (on the Config tab of Linker settings) and no debug macro file (on the Setup tab of the Debugger settings). The checkbox **Use flash loader** must be checked (on the Download tab of the Debugger settings). The **Reset** type must be set to **Core** for J-Link (on the Setup tab of the J-Link/J-Trace settings) or **Normal** for ST-Link. Otherwise, execution will hang or fault in the startup code before main().

2. **SB_DEBUG_IN_ROM must be 1** in startup.c for **Debug** and **Release** targets. It should be automatically set correctly.
3. Use the IAR Flash Loader to store the image file into flash. Please see section "Using IAR Flash Loader to Flash the Application" in this document.
4. This memory map comes from Table 5. Memory mapping vs. Boot mode/physical remap in STM32F7x6 Advanced ARM-based 32-bit MCUs Reference Manual.
5. External SDRAM is initialized in SDRAM_Init() function. The driver code is supported by STMicroelectronics.

Memory Map (Debug and Release targets — External SDRAM) *** NOT SUPPORTED YET ***

Base Address	Size	Memory	Sections	Notes
0x00200000	1MB	Internal Flash (ITCM)		Unused. Faster bus.
0x08000000	1MB	Internal Flash (AXIM)		Unused. Slower bus.
0x10000000	64KB	Internal CCM SRAM	SDAR, CSTACK	SDAR and System Stack. See note about CCM in Memory Configuration section.
0x20000000	256KB	Internal SRAM	.intvec, EMAC_BUFFER	.intvec (EVT) copied here by startup code. smxNS frame buffers.
0x60000000	16MB	External NOR Flash		Quad SPI Flash Memory
0xC0000000	32MB	External SDRAM	.intvec, .text, .rodata, .noinit, .bss, .data, ADAR, LCD Buffer	Code, Data, ADAR and LCD Buffer

Notes:
1. This uses the linker command file **stm32746gdis_xram.icf** (on the Config tab of Linker settings) and the debug macro file **stm32746gdis_xram.mac** (on the Setup tab of the Debugger settings). The checkbox **Use flash loader** must be unchecked (on the Download tab of the Debugger settings). The **Reset** type must be set to **Core** (on the Setup tab of the J-Link/J-Trace settings). Otherwise, the SDRAM is reset after downloading finishes.
2. **SB_DEBUG_IN_ROM must be 0** in startup.c. It should be automatically set correctly.
3. Internal Flash ITCM and AXIM are the same physical memory but accessed via ITCM or AXIM interface, according to location addresses in .icf file.
4. This memory map comes from Table 5. Memory mapping vs. Boot mode/physical remap in STM32F7x6 Advanced ARM-based 32-bit MCUs Reference Manual.
5. The clock and External SDRAM are initialized in the .mac file.
6. MPU must be enabled for SDRAM; otherwise it crashes after downloading. Also, it is set bufferable, cacheable, and shareable.

Memory Configuration

The memory maps are shown above. These are achieved with the linker command files (.icf) in APP\IAR.AM\STM32.

RAM usage is affected by settings in APP\acfg.h. If running from internal memories only, we define SB_MIN_RAM in bsp.h to reduce many settings. You may be able to reduce it some more. In particular, NUM_STACKS has a big effect on RAM usage. As shipped, it is typically set to allow a few extra stacks beyond what the Protosystem/demos need, so you can add tasks. Use smxAware's stack display (text not

GAT) to see how many of them are actually being used after free-running the application briefly. You may want to tune this down, especially if you choose to allocate bound stacks instead (stack_size parameter of smx_TaskCreate() != 0).

SDAR is put in DTCM SRAM (tightly coupled memory) for performance. This SRAM is not on the bus; it is connected only to the CPU, so there are limits on what can be put into it. In particular, the EVT and peripheral data buffers cannot be put into it. SDAR and ADAR can be located where desired. See dynamically allocated regions in the Memory chapter of the smx User's Guide.

Peripherals Available on the STM32746G-DISC Board

1. SysTick – Used to generate the smx system tick. See bspm.c in BSP\ARM.
2. USART #1 – Initialized to 115.2Kbps / no parity / 8 bits / 1stop bit. Used as the SMX information port. This includes Protosystem/demo output messages and SMX errors. This is via mini ST-Link USB port CN14.
3. LED row – 1 LED, LD1 located between reset button and JP1. Supported by our led.c. See LED_task or LED_LSR in app.c.
4. LCD Display – Supported by our lcd.c to write text strings, which is just a simple mapping onto a few functions in STMicro LCD driver for this board. Demonstrated by lcddemo.c. Set SB_DISP_DEMO to 0 in bsp.h to disable the demo.
5. Ethernet – Supported by smxNS. Driver is XNS\drvsrc\arm\stm\stm32eth.c.
6. USB Device – Supported by smxUSBD. Driver is XUSBD\DCD\udsyndwc.*.
7. USB Host – Supported by smxUSBH, Driver is XUSBH\HCD\usyndwc.*.
8. USB OTG – TBD
9. MicroSD card interface – Supported by smxFS. Driver is XFS\fdmsd*.*.
10. Accelerometer, ADC, audio, buttons, camera, CAN, CRYP, DAC, DCMI, I2C, I2S, joystick, potentiometer, power management, RNG, RTC, SDIO, Smart Card, SPI, timers, watchdog are not supported.

Clock (PLL) Configuration

The clock speed is controlled by the crystal and PLL configuration, which is initialized by sb_ClocksInit() in bspm.c, using functions in the STMicro library.

Crystal / CLKIN	SYSCLK (CPU)	HCLK (AHB)	PCLK1 (APB1)	PCLK2 (APB2)	PLL48 CLK	PLLI2S
25.000 MHz	200 MHz	200 MHz	50 MHz	100 MHz	50 MHz	N/A

When running USB full speed (FS or HS controller) set SB_USB_FS to 1 in bsp.h to get:

Crystal / CLKIN	SYSCLK (CPU)	HCLK (AHB)	PCLK1 (APB1)	PCLK2 (APB2)	PLL48 CLK	PLLI2S
25.000 MHz	192 MHz	192 MHz	48 MHz	96 MHz	48 MHz	N/A MHz

The second case is needed so PLL48 is 48 MHz, which is required when running the USB FS or HS controller at full speed.

Timer Configuration

SysTick is used as the smx system tick, upon which all smx timing is based. The other timers are available for your use. Note that SysTick is part of the interrupt controller (NVIC), which is part of the ARM-M core, so it is documented in the section for the processor core.

USART Configuration

USART #1 is used for console i/o, in polled mode. Messages written by the Protosystem and demos using sb_ConWriteString(), etc are displayed to an ANSI terminal connected to port “USART1” (CN8 – the DB9 connector closest to the LCD) on the board using a null modem cable. Initialization is done by sb_PeripheralsInit() in bspm.c. It is set to 115200-N-8-1.

Ethernet Configuration (smxNS)

The STM32 Ethernet controller uses DMA to transfer directly between the controller and the smxNS frame buffers. The DMA is coordinated using transmit and receive buffer descriptors. The controller is programmed to communicate with the PHY using RMII. The Ethernet controller initialization function assigns a number of alternate function signals to the Ethernet controller, so applications that bring in other subsystems should review this initialization code.

SDIO Configuration (smxFS SD/SDHC)

Polling mode and DMA mode are supported. The connector on the board is MicroSD so we could not test an MMC card.

USB Device Configuration (smxUSBD)

There are two USB controllers in this processor, one for full speed and one for high speed. It is the Synopsys IP. On the board, USB_FS is connected to the FS controller, and USB_HS is connected to the HS controller via the external SMSC3300 ULPI transceiver. Unfortunately, USB_FS shares pins with the UART, so it is necessary to disable SMX console I/O to use it. We recommend using USB_HS unless you are using both smxUSBD and smxUSBH, in which case one must use USB_HS and the other USB_FS. Our device driver doesn’t support running both controllers in USB device mode simultaneously, but this can be supported in the future.

To use USB_HS, set SUD_STM32F2XX_USE_HS to 1 in udport.h.

To use USB_FS, set SUD_STM32F2XX_USE_HS to 0, and set SB_USB_FS to 1 in bsp.h and change the PLL_N setting accordingly in the debug .mac file (search for SB_USB_FS comment) so the PLL48 clock is correct. See section Clock (PLL) Configuration above. Also disable SMX console I/O by disabling SB_CON_IN and SB_CON_OUT to 0 in bcfg.h.

USB Host Configuration (smxUSBH)

The same discussion applies as in the USB Device section above, but it is SU_STM32F2XX_USE_HS in uport.h. Our host driver now supports running both controllers in USB host mode simultaneously (but it may not have been tested yet on this target). For this case, set SU_SYNOPSYS to 2 in ucfg.h and

SU_STM32F2XX_USE_HS to 1 in uport.c. As mentioned for smxUSBD, you must set SB_USB_FS to 1 in bsp.h and change the PLL_N setting accordingly in the debug .mac file (search for SB_USB_FS comment). You won't get terminal output because the FS port shares pins with the UART.

USB OTG Configuration (smxUSBO)

TBD

Debugging with IAR C-SPY

It is necessary to change the project settings to specify and configure it for the debug connection you are using. This is documented in the IAR EWARM User's Guide in Part 6: C-SPY Hardware Debugger Systems/ Hardware Specific Debugging. Also, see the documents about this linked by the IAR release notes (readme.htm). In the table of documents at the bottom of the release notes there are links to IAR C-SPY driver notes files.

If necessary, a .mac file is provided in the project directory that the debugger runs to configure the board before downloading code. The project file is already set to point to this file.

Note: Breakpoints are limited to 6 when running in Flash, which is much better than traditional ARMs which support only 2. Still, this can make debugging more difficult than in RAM. The debugger will refuse to set more breakpoints, and a run to cursor might actually use the mode that traps after each assembly instruction, taking a long time to reach the destination.

Debugging with ST-Link

ST-Link is the debug connection built into the board via the mini USB jack CN14. Select ST-Link in the project debug settings. See the previous section Debugging with IAR C-SPY.

Boot and Startup Code

The same startup code is linked to the Protosystem for all build targets, so we don't rely on any boot code in flash to bring up the board. Summary of startup:

__iar_program_start -> __cmain -> __low_level_init -> main() -> smx_Go() -> smx_SchedRunTasks() -> ainit() -> tasks

__iar_program_start is in the IAR run time library, DLIB, in cstartup_M.c. It calls __cmain in cmain.s, which calls __low_level_init, which is our startup code in BSP\ARM\STM32\STM32F7xx\startup.c, and then it calls other init routines before calling main(). main() is in APP\main.c.

At startup, the main stack is used. smx_Go() changes to the dual-stack model so that tasks use the process stack pointer (PSP), and ISRs, LSRs, and the scheduler use the main stack pointer (MSP). There is a brief time during startup before the first task runs that both pointers are writing to the same area of memory, corrupting it, but that is ok because it is never used.

Using IAR Flash Loader to Flash the Application

EWARM has a built-in flash loader for STM32 processors. It programs the internal flash on the chip. The ROM build target in the project file is already set up properly. See the EWARM User's Guide, Part 6: C-SPY Hardware Debugger Systems/ Hardware-Specific Debugging/ Using the Flash Loaders.

To download the app into flash, just press the Debug button as when debugging in RAM. The IDE automatically downloads the flash loader to RAM and starts it running to download and program the binary file into flash. The debugger comes up on the first instruction. You can run it in the debugger or you can power off the board, disconnect the JTAG unit, and power the board on. You can see it is running by the LED(s) blinking and terminal output.

Further Reading

Refer to the release notes in the DOC directory and the ARM-M section of the SMX Target Guide.